Étude des méthodologies de conception, outils de synthèse et de génération automatiques de parties contrôles de microprocesseurs

Régine Etienne

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
« Génie Informatique»

par

ETIENNE Régine

ETUDE DES METHODOLOGIES DE CONCEPTION, OUTILS DE SYNTHESE ET DE GENERATION AUTOMATIQUES DE PARTIES CONTROLES DE MICROPROCESSEURS.

Thèse soutenue le 29 juin 1983 devant la commission d'examen.

L. BOLLIET	Président
F. ANCEAU	
R. GERBER	Examinateurs
J.P. MOREAU	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

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

.../...

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

PROFESSEURS ASSOCIES

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

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

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

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

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

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

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

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

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

.../...

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

.../...

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

LABORATOIRES EXTERIEURS

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

Je tiens à remercier

- l'équipe d'Architecture des Ordinateurs de l'IMAG et surtout le Professeur F.Anceau pour m'avoir guidée dans le domaine de la Micro-Électronique et pour m'avoir encouragée à chercher aux Etats-Unis la source d'information contenue dans ce travail.

- l'équipe du Professeur C.Sequin à l'Université de Berkeley et celle de Mr R.Lorie au laboratoire de recherche d'IBM à San José pour m'avoir accueillie dans leur groupe et pour m'avoir permis de travailler sur divers aspects de la conception de circuits intégrés.

- Mr H.Reich, du laboratoire de recherche d'IBM San José, pour ses conseils et discussions ainsi que Mr P.Wilms, du laboratoire de recherche d'IBM San José également, pour son aide précieuse tout au long de la mise en forme de ce travail.

TABLE DES MATIERES

INTRODUCTION

INTRODUCTION

En l'espace de quelques années, les méthodes de conception et les procédés de fabrication de systèmes logiques ont évolué de manière considérable. La possibilité d'intégrer sur des surfaces de quelques millimètres carrés plusieurs milliers de transistors a stimulé de nombreuses équipes de recherche et de développement à mettre au point un environnement informatique supportant à la fois les phases de conception et de fabrication de ces systèmes.

Cet ensemble de logiciels fait appel à des disciplines informatiques très diverses telles que le développement de langages spécialisés, d'interpréteurs, de compilateurs, d'algorithmes spéciaux permettant de réaliser des simulations, vérifications ou optimisations. L'utilisation de matériel plus perfectionné, conjointement à ces outils logiciels permet d'augmenter la complexité des circuits réalisés tout en maintenant un coût de production minimal.

L'avènement des circuits intégrés n'est pas seulement le fruit des progrès réalisés dans le domaine de l'informatique. Leur existence est le résultat d'efforts conjoints autant en physique du solide que dans les procédés physico-chimiques de fabrication.

Ces trois domaines de recherche sont très dynamiques: les résultats obtenus depuis cinq ans ont permis de doubler tous les deux ans environ les performances et la complexité de tels circuits. Les systèmes informatiques d'aide à la conception changent tellement rapidement que tout essai de description de tels systèmes est vite périmé. Toutefois des méthodologies de conception s'affinent et restent valides sur des échelles de temps plus importantes.

Outre les systèmes logiques digitaux, il est possible d'intégrer sur silicium des systèmes analogiques permettant de mettre sur pied des applications telles que des systèmes de traitement d'images, de reconnaissance de la parole, etc..

Dans cet ouvrage, nous nous limiterons à l'étude structurelle et fonctionnelle de systèmes logiques digitaux appelés MICROPROCESSEURS. Cette étude est constituée de quatre parties distinctes:

1. la présentation d'une méthodologie générale pour la conception de circuits intégrés,

2. la présentation d'une étude réalisée à l'IMAG dans le groupe d'Architecture des Ordinateurs concernant le développement d'outils de simulation et de synthèse de parties contrôles de microprocesseurs,

3. la présentation d'outils logiciels et matériels types nécessaires à une conception assistée par ordinateur de tels circuits,

4. la présentation d'une étude réalisée dans le laboratoire de recherche d'IBM San José (Californie) concernant le développement d'un éditeur graphique pour la conception de circuits intégrés utilisant comme système de gestion de données, System R, le système de gestion de bases de données relationnelles développé, lui aussi, à IBM San José.

Au cours de ce travail, les termes 'microprocesseur' et 'circuit intégré' seront utilisés indifféremment.

1. APPROCHE METHODOLOGIQUE DE LA CONCEPTION DE MICROPROCESSEURS

1.1 DÉFINITIONS D'UN MICROPROCESSEUR

1.1.1 Définition sémantique

Le terme *MICROPROCESSEUR*, comme l'indique sa structure sémantique, est un processeur réalisé à une échelle microscopique.

En tant que *PROCESSEUR*, sa fonction principale est d'exécuter des opérations définies sous la forme d'instructions. La nature et la complexité de ces opérations ont beaucoup évolué durant ces cinq dernières années depuis de simples additions logiques jusqu'au calcul de fonctions trigonométriques, au traitement de textes, etc... Néanmoins, aussi diverses qu'elles puissent paraître , ces opérations se concrétisent toutes par un assemblage de portes logiques.

Le préfixe *MICRO* se réfère à la taille du circuit réalisant ces opérations. Physiquement, l'ensemble de ces portes logiques est intégré sur des plaquettes de silicium monocristallin selon une technologie particulière. Nous ne nous étendrons pas sur les différentes technologies utilisées pour la fabrication de ces circuits, mais il est important de noter que les caractéristiques de la technologie utilisée doivent être prises en compte lors de la conception de tels circuits. Nous reviendrons sur ce point au chapitre 3.

1.1.2 Définition algorithmique

Dans la mesure où le rôle d'un microprocesseur est l'exécution d'un certain processus, il peut être fonctionnellement assimilé à ce processus. Cela signifie que son comportement peut être décrit à l'aide d'un ensemble d'algorithmes coopérants.

La définition comportementale (ou algorithmique) du processeur comporte les deux aspects suivants:

1. définition du langage de commandes que doit interpréter le microprocesseur,

2. description des algorithmes internes permettant d'éxécuter ces commandes.

L'interprétation de ces deux descriptions débouche sur la définition structurelle de ce processeur, c'est-à-dire la description des éléments électroniques utilisés pour la réalisation physique du processeur.

1.1.3 Définition structurelle - Notion de partie opérative/partie contrôle

Un microprocesseur peut être défini d'une autre façon. Du point de vue utilisateur, il se présente comme une boîte noire qui exécute un certain nombre d'instructions. Cette boîte noire communique avec son environnement à travers un interface clairement défini: celui-ci interprète en entrée un jeu de commandes défini par le concepteur et fournit en sortie le résultat des opérations effectuées.

Cette boîte noire est constituée de deux entités fonctionnelles distinctes: une partie opérative et une partie contrôle. Cette décomposition d'un microprocesseur en Partie Opérative (PO) et en Partie Contrôle (PC) est classique. Elle peut être définie comme suit:

1. la Partie Opérative est chargée de manipuler et transformer les données reçues;

2. la Partie Contrôle a pour rôle d'animer la Partie Opérative. Elle lui envoie à chaque instant un certain nombre de commandes de manière à assurer l'exécution de l'instruction reçue. (Cette instruction est communiquée au processeur à travers son interface).

Ces deux parties coopèrent entre elles en échangeant un certain nombre de signaux, la partie opérative renseignant la partie contrôle de l'état dans lequel elle se trouve et la partie contrôle lui envoyant des commandes permettant d'exécuter des opérations élémentaires.

La Figure 1.1 montre les relations existant entre ces deux entités coopérantes.

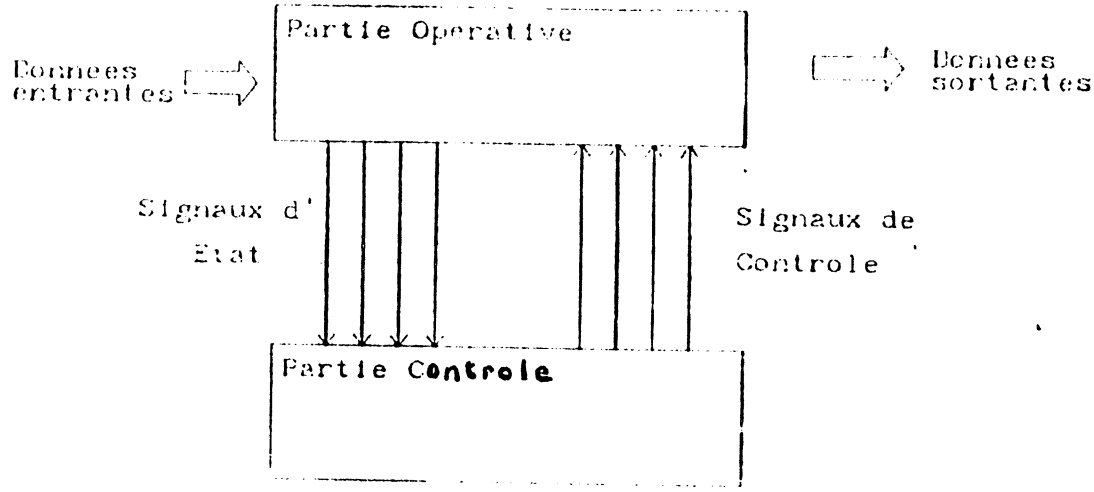


Figure 1.1: Partie Opérative/Partie Contrôle

La définition de ces deux entités résulte de la démarche suivante: la description de l'algorithme exécutant ce jeu de commandes est réalisée à l'aide d'un langage, choisi par le concepteur. De cette description deux types d'informations sont extraits, comme l'illustre la figure suivante:

1. à partir des éléments sémantiques du langage de description, un ensemble de structures de données. Ces structures caractérisent la partie opérative du circuit, la partie qui contient les données.

2. à partir de l'enchaînement des actions décrites par cet algorithme, la partie contrôle.

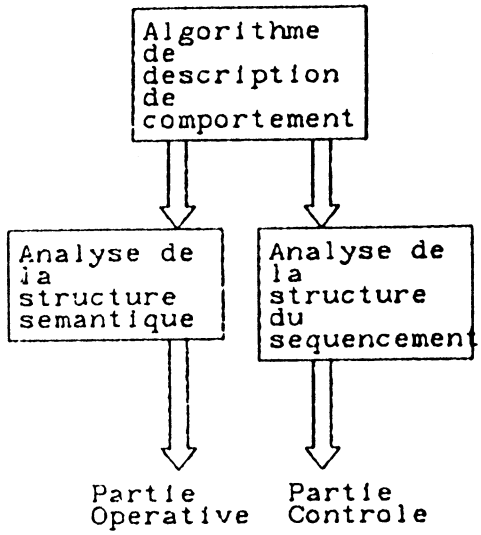


Figure 1.2: Extraction de la PO/PC à partir de l'algorithme décrivant le fonctionnement du microprocesseur

Ces deux définitions, fonctionnelle et structurelle, sont complémentaires, l'une décrivant le comportement du processus à réaliser, l'autre sa structure d'implantation.

1.2 MÉTHODOLOGIE DE CONCEPTION

L'accroissement de la complexité des algorithmes intégrés sur silicium a nécessité la mise au point d'une méthodologie de conception désormais classique. Elle se compose des étapes suivantes:

1. une démarche dite 'DESCENDANTE' permettant de décomposer une description complexe de comportement en sous-algorithmes décrivant des comportements plus simples;

2. une démarche dite 'ASCENDANTE' permettant, à partir d'éléments physiques prédéfinis, de réaliser des structures plus complexes.

De plus, trois différents types de description sont utilisés pour décrire un microprocesseur:

1. une étape de description comportementale durant laquelle le concep- teur décompose les descriptions fonctionnelles initiales en une série d'algorithmes de complexité décroissante;

2. une étape de description structurelle pendant laquelle le concepteur décrit sous forme modulaire les structures d'implantation des algorithmes utilisés;

3. une étape de description physique pendant laquelle le concepteur doit décrire son circuit en tenant compte d'une technologie donnée.

Dans la majorité des cas, les descriptions comportementales et structurelles sont réalisées conjointement en suivant une démarche descendante. La description physique, par contre, est effectuée indépendamment en suivant une approche ascendante.

Il est important de noter que dans tous les cas une approche hiérarchique est utilisée. Le partitionnement induit, physique, structurel ou algorithmique, permet de réduire localement les complexités et d'assurer ainsi une conception structurée.

1.3 DESCRIPTIONS COMPORTEMENTALES

Le terme description comportementale équivaut ici à description fonctionnelle, le comportement d'un circuit à l'ensemble des fonctions qu'il réalise.

Les descriptions comportementales d'un microprocesseur peuvent être décomposées hiérarchiquement en différents niveaux de descriptions de

plus en plus fins, de la description procédurale de haut niveau jusqu'au fonctionnement électrique du circuit.

La décomposition hiérarchique du comportement d'un microprocesseur commence par l'interprétation de l'ensemble des commandes à exécuter.

Ces dernières se présentent sous la forme d'instructions de format variable mais généralement composées des champs suivants:

1. un champ code opération spécifiant le type d'actions à générer,

2. un champ adresse spécifiant le type d'opérations à effectuer pour localiser les opérandes,

3. un champ opérande constitué d'un ou deux opérandes sur lesquels sont effectuées certaines opérations.

Chacun de ces champs est interprété de manière spécifique et des fonctions types peuvent être extraites. L'exemple suivant illustre la décomposition hiérarchique d'une instruction.

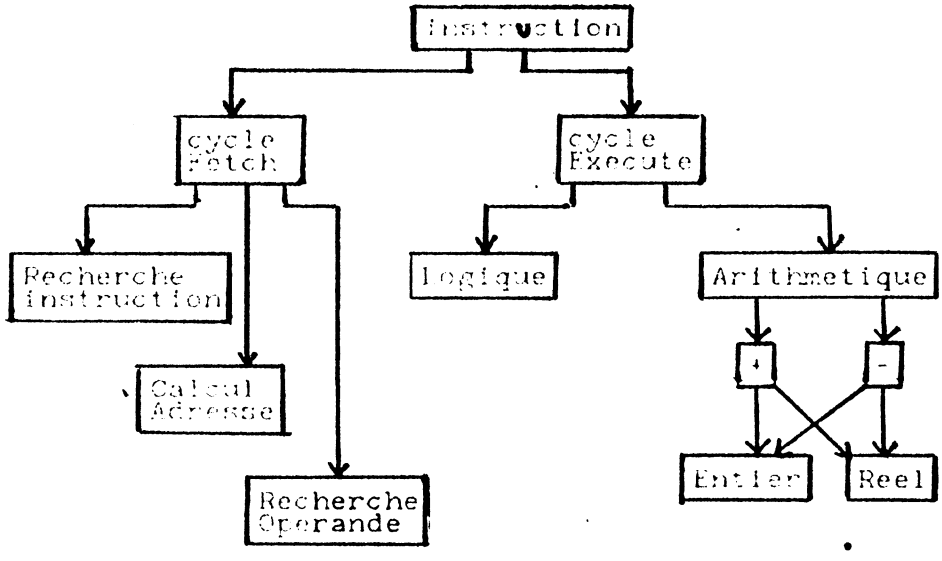


Figure 1.3: Décomposition d'une instruction

Il est à noter que cette décomposition fonctionnelle est indépendante de la composition physique du circuit. Elle peut donc être entreprise avant que toute contrainte d'ordre physique ou technologique ne soit prise en compte.

1.4 DESCRIPTIONS STRUCTURELLES

Le terme description structurelle signifie ici description de la structure d'un circuit en termes de modules, de blocs. Ces descriptions structurelles sont elles aussi décomposables hiérarchiquement, de la description de plan de masse (haut niveau) jusqu'à la description de modules élémentaires (transistor - inverseur).

La description structurelle d'un circuit est, de manière générale, réalisée en même temps que sa description comportementale. Ces deux descriptions sont étroitement liées car elles correspondent à deux aspects d'un même problème: comment décrire une fonction et comment décrire le support physique permettant de réaliser cette fonction.

La description structurelle d'un microprocesseur est, elle aussi, réalisée de manière hiérarchique et est, par nature, modulaire. En effet, dans la mesure où des segmentations fonctionnelles sont réalisées pour réduire la complexité des algorithmes à implémenter, leurs structures d'implantation sont, elles aussi, décomposées en structures plus simples.

Les principaux avantages d'une description structurelle hiérarchisée et modulaire sont de deux ordres.

1. Tout d'abord, le circuit à réaliser est décrit sous la forme de modules dont les interfaces sont clairement définies. La lisibilité des descriptions en est par conséquent accrue.

2. De plus, la mise en évidence de fonctions élémentaires (et donc de structures élémentaires d'implantation) favorise la standardisation de ces modules. Leur génération automatique est aujourd'hui pratique courante. Elle permet de réduire considérablement le

temps de conception en augmentant l'aspect répétitif des descriptions structurelles.

1.5 DESCRIPTIONS PHYSIQUES

Le terme description physique regroupe ici à la fois des descriptions géométriques de bas niveau (description des masques) et le comportement électrique du circuit au niveau le plus fin.

La description physique d'un circuit est de manière générale réalisée par assemblage progressif d'objects physiques communément nommés CELLULES. Elle est aussi de nature hiérarchique car le concepteur construit à partir de ces cellules élémentaires des structures physiques plus complexes jusqu' à générer la description physique totale du circuit.

L'élément de base à partir duquel le circuit est décrit est le transistor. Sa description physique varie selon la technologie utilisée pour réaliser le circuit.

A partir du transistor, des fonctions logiques élémentaires sont réalisables telles que l'inverseur, l'interrupteur, les portes NAND ou NOR, le point mémoire. Des registres, multiplexeurs, unités de calcul peuvent ensuite être décrits à partir de transistors, points mémoires et inverseurs.

La figure suivante illustre cette conception hiérarchique ascendante.

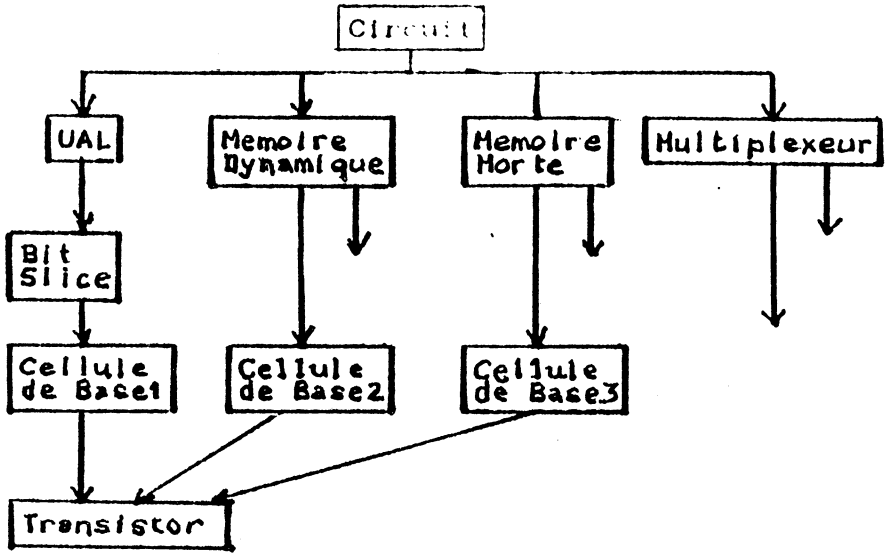


Figure 1.4: Description hiérarchique physique

Le lien entre description physique et structurelle du circuit est réalisé par la description de la structure physique permettant d'implanter ces blocs structurels.

1.6 INTERACTIONS ENTRE DESCRIPTIONS ET INFLUENCE SUR LES PERFORMANCES DU CIRCUIT

La distinction entre ces trois types de descriptions, fonctionnelle, structurelle et physique est importante. Elle donne au concepteur une démarche conceptuelle systématique pour décrire des circuits intégrés. Toutefois leur degré d'interaction est très étroit. En effet, la structuration fonctionnelle d'un circuit a une grande influence, autant sur ses caractéristiques topologiques (surfaces, interconnexions), que sur ses caractéristiques électriques (vitesse de propagation des signaux, etc...). De même, la structure physique d'un circuit conditionne très fortement son comportement électrique et donc fonctionnel. De nombreuses itérations sont donc nécessaires avant d'obtenir une description physique du circuit qui respecte l'ensemble des spécifications initiales du circuit. Celles-ci sont réalisées aujourd'hui de manière automatique par un ensemble de logiciels spécialisés. Nous reviendrons plus en détail sur cet aspect dans la troisième partie de cet ouvrage.

1.7 IMPORTANCE DU TEMPS DANS LA CONCEPTION DE CIRCUITS INTÉGRÉS

La prise en compte du temps durant les phases de description fonctionnelle et structurelle est importante car elle conditionne les performances du circuit que l'on souhaite réaliser.

La notion de TEMPS intervient de différentes façons selon le niveau hiérarchique auquel le concepteur se situe. Il est utilisé essentiellement:

1. comme trame de référence pour l'enchaînement des actions permettant l'exécution d'une instruction donnée;

2. comme élément de validation de commandes de structures physiques, telles que la lecture ou l'écriture dans les registres, mémoires.

1.7.1 Représentation du temps

Le facteur temps permet au concepteur:

1. de synchroniser entre elles les unités fonctionnelles constituant ce processeur pour assurer l'exécution correcte des tâches telles que le cycle de recherche d'une instruction ou bien le calcul de l'adresse d'un opérande;

2. de réguler et de séquencer le déroulement des actions élémentaires contrôlant le fonctionnement de ces unités.

De manière générale, les instructions exécutées par un microprocesseur sont décomposées en une séquence d'actions élémentaires qui utilisent le temps comme support de référence. Différentes représentations du temps sont utilisées à chacune de ces étapes. Elles sont toutes issues des deux phases fondamentales, communément nommées phi1 et phi2. Elles sont, de manière générale, obtenues par combinaisons linéaires de ces dernières. L'exemple suivant propose une décomposition de ces deux phases phi1 et phi2 en quatre autres périodes élémentaires nommées ici T1,..,T4. Elles sont obtenues par division de l'horloge de base.

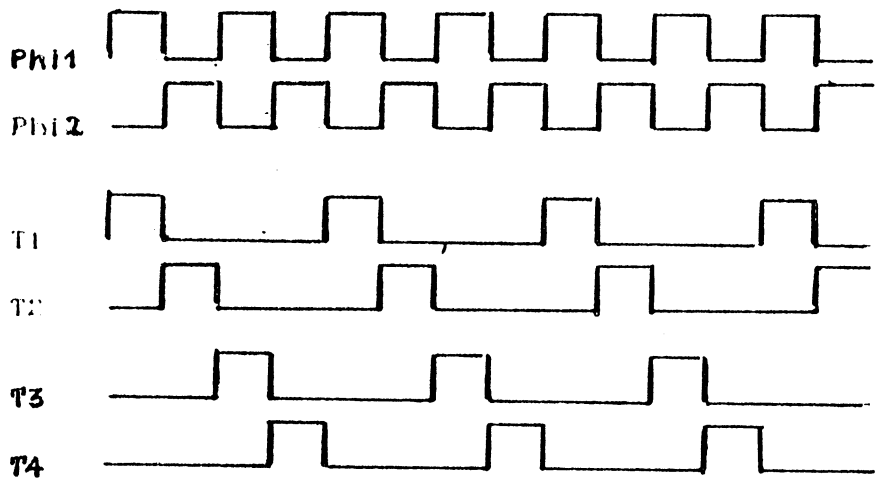


Figure 1.5: Décomposition du temps en périodes élémentaires.

Ces périodes élémentaires sont utilisées localement au sein de chaque bloc structurel et permettent de mesurer la vitesse de transfert des données à travers chacun d'eux. Chaque bloc fonctionnel possède un cycle de base qui lui est propre: le cycle d'une mémoire morte diffère de celui de l'unité arithmétique et logique par exemple, le cycle de base d'une instruction peut varier selon la complexité de cette dernière; l'addition du contenu de deux registres est plus rapide que le transfert du contenu d'un mot mémoire dans un autre. Des synchronisations doivent être réalisées par le concepteur de manière à organiser l'activité de ces blocs fonctionnels les uns par rapport aux autres.

1.7.2 Systèmes synchrones

Les microprocesseurs réalisés jusqu' à aujourd'hui sont caractérisés par le fait que leur fonctionnement est réglé sur une horloge de base, bi-phasée. Ce sont des systèmes *SYNCHRONES*.

Les systèmes synchrones présentent l'avantage d'être simples à générer. Ils permettent des simulations et vérifications temporelles du fonctionnement logique du circuit. Leur utilisation présente toutefois quelques limitations. En effet, la synchronisation de l'ensemble du circuit sur une horloge de fréquence donnée implique que toutes les unités fonctionnelles constituant ce circuit aient une fréquence de fonctionnement de base plus lente que celle de l'horloge. Ceci n'était guère génant il y a quelques années. Mais depuis que des circuits à très haute intégration sont réalisés, la rapidité interne que l'on peut maintenant atteindre soulève de nouveaux problèmes.

En effet, les dimensions des composants électroniques continuent à décroître: la largeur de la grille d'un transistor est maintenant de l'ordre du micron et demi (elle est passée de cinq puis à trois microns en quelques années). Les conséquences de cette diminution sont de deux ordres:

1. la vitesse de propagation des données à travers les structures d'implantation a beaucoup augmenté;

2. la vitesse de propagation de ces données à travers les bus (ou interconnexions) devient beaucoup trop lente. Ce déséquilibre s'explique par l'accroissement relatif de la longueur des interconnexions (et donc de leur résistance) par rapport à la taille des unités fonctionnelles.

De nouvelles directions sont actuellement explorées, en particulier, les systèmes asynchrones.

1.7.3 Systèmes asynchrones

Une stratégie différente consiste à concevoir un microprocesseur comme constitué d'unités fonctionnelles autonomes autant fonctionnellement que temporellement. Chacune d'entre elles fonctionne selon un rythme qui lui est propre (elles possèdent en particulier leur propre horloge de base). Les communications entre unités sont réalisées grâce à des protocoles d'échanges de données, activant certaines unités, mettant d'autres au repos.

L'utilisation de systèmes asynchrones dans la conception de VLSI commence a être l'objet de recherches approfondies [SEI79]. Ils représentent une alternative intéressante car ils permettent de tirer profit des accroissements de performances obtenus grâce à la diminution de la taille des composants et l'augmentation des densités sur silicium.

2. CONCEPTION ET RÉALISATION DE PARTIES CONTROLES DE MICROPROCESSEURS

2.1 PRÉSENTATION

Cette première partie présente la manière dont les méthodes de conception et de synthèse de parties contrôles de microprocesseurs ont été étudiées dans l'Equipe d'Architecture des Ordinateurs de l'IMAG (Institut de Mathématiques Appliquées de Grenoble). Ce travail a été entrepris dans le cadre du projet CAPRI (Conception Assistée de PRocesseurs Intégrés) dans le but de mettre au point un système intégré d'aide à la conception de microprocesseurs. Les principales orientations du projet se caractérisent par:

1. l'étude des méthodes de génération automatique de parties opératives de circuits intégrés [SUZ81],

2. l'étude des différentes méthodes de conception et de génération de parties contrôles de microprocesseurs [OBR82],

3. la génération automatique de PLAs (Programmable Logic Arrays) à partir d'équations logiques [PER82],

4. le développement d'outils d'évaluation topologique permettant de générer une topologie optimale de circuits intégrés en fonction de paramètres préalablement définis [REI81],

5. le développement d'un langage de description de matériel structurel et comportemental permettant à un ensemble d'outils logiciels de générer automatiquement les descriptions géométriques des masques en vue de la fabrication de circuits intégrés.

Le premier domaine que nous avons exploré concerne l'étude des différentes structures d'implantation de parties contrôles de microprocesseurs. Cette étude repose sur l'étude microphotographique de microprocesseurs commercialisés soit en France, soit aux USA. Ce premier chapitre est organisé comme suit:

1. présentation des méthodes classiques d'implantation de parties contrôles,

2. présentation des outils de description, simulation et synthèse développés à l'IMAG,

3. résultats et conclusions de cette étude.

2.2 METHODES CLASSIQUES D'IMPLANTATION DE PARTIE CONTROLE DE MICROPROCESSEURS

L'étude microphotographique des principaux microprocesseurs commercialisés en France [OBR82] nous a permis de déduire du schéma topologique de ces circuits un certain nombre d'informations concernant l'approche méthodologique utilisée, les outils d'aide à la conception disponibles ou encore l'état d'avancement de la technologie au moment de la conception du circuit. La diversité des schémas rencontrés est aussi une indication reflétant le fait qu'il n'existe pas qu'une seule méthode valide pour représenter matériellement des algorithmes de contrôle. Le schéma d'implantation choisi correspond en effet à un ensemble de conditions et de contraintes lié à l'environnement technologique et aux spécificités du cahier des charges.

Quatre méthodes caractéristiques d'implantation de parties contrôles sont présentées ici. Elles reflètent l'impact de l'évolution technologique et conceptuelle sur la structure physique de ces circuits depuis ces cinq dernières années.

2.2.1 Réalisation Câblée

La réalisation câblée de partie contrôle de microprocesseurs est une méthode aujourd'hui abandonnée mais qui était couramment utilisée au début de l'ère des microprocesseurs. Elle reflète le fait que l' algorithme de contrôle des circuits d'alors était directement interprété en termes de portes logiques NAND et NOR. Celles-ci étaient ensuite dessinées manuellement. Le schéma topologique issu de cette méthode de réalisation est de manière générale assez dense et très peu lisible. Il est aisément imaginable que peu de tests ont été réalisés sur de tels circuits et que des erreurs de logique pouvaient difficilement être corrigées. De plus, ce type de conception manuelle est très cher en temps de conception et en nombre d'erreurs humaines souvent difficiles à trouver.

L'exemple suivant représente la structure 'fonctionnelle' de la partie contrôle du Motorola M6800. Ce circuit, conçu en 1976 et composé d'environ 5.000 transistors, est un microprocesseur 8 bits exécutant un jeu d'instructions classiques [NEM81]. Les étapes de décomposition de l'algorithme de contrôle sont représentées comme suit: le code opération issu de la micro-instruction est décodé (étape 1); selon ce code, différents blocs fonctionnels peuvent être activés pour contrôler de manière adéquate l'ensemble de données parcourant la partie opérative.

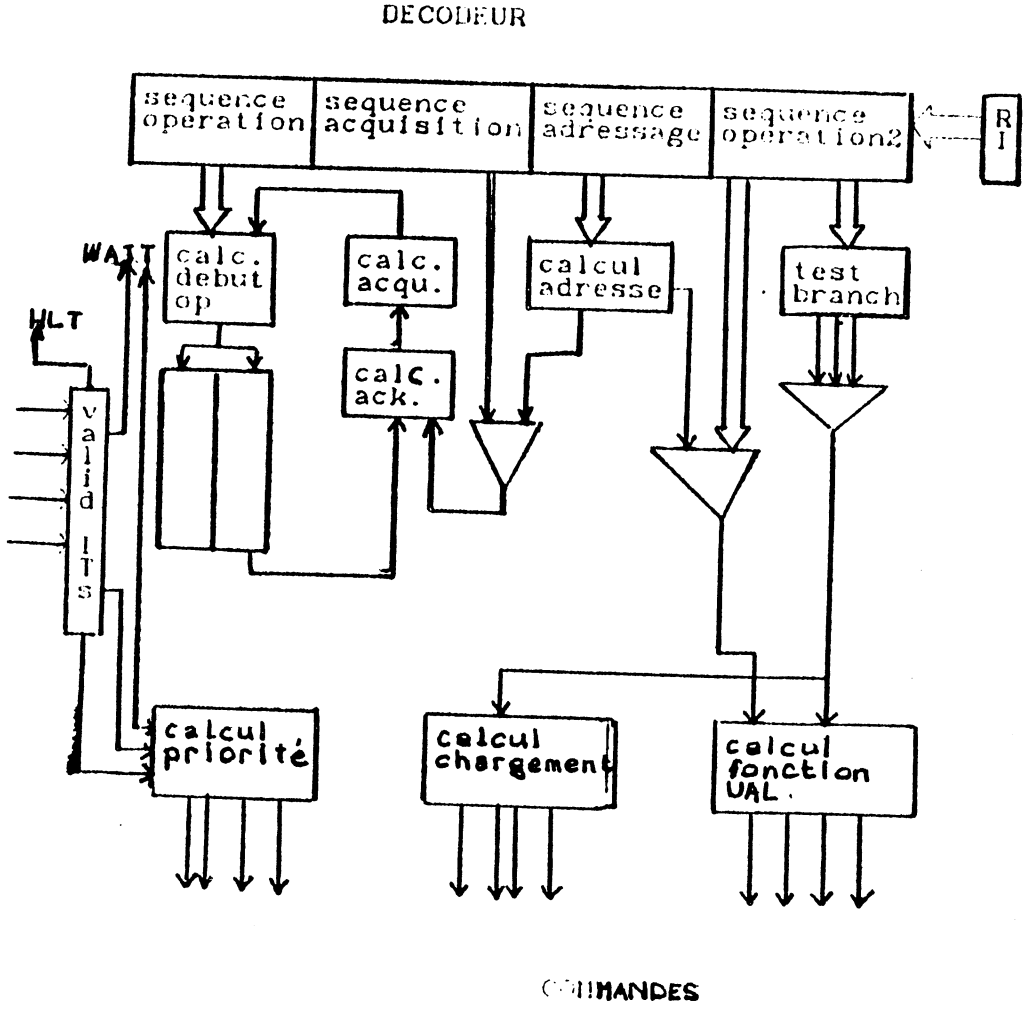


Figure 2.1: Présentation de la Partie Contrôle Cablée du M6800

2.2.2. Réalisation Microprogrammée

La microprogrammation est une méthode de réalisation qui a été très utilisée (et l'est encore) dans le cadre de la conception de mini et de micro ordinateurs. Pour la conception de circuits intégrés, elle représente une alternative intéressante par rapport à la réalisation câblée. En effet, l'algorithme, au lieu d'être interprété directement en termes de portes logiques, est représenté par une séquence de micro-instructions mémorisées sous la forme d'une matrice de transistors. Une micro-instruction est constituée de deux champs fonctionnellement distincts: un champ adresse indiquant l'adresse de la prochaine micro-instruction à exécuter et un champ commandes contenant la liste des commandes à envoyer à la partie opérative pour activer les organes désirés. Cette matrice de transistors est souvent très creuse car le champ commandes possède une entrée par commande pour chaque organe constituant la partie opérative; à chaque instant un nombre limité de ces commandes est activé. Quant au champ adresse, sa compacité dépend du type de codage utilisé.

Cette méthode de réalisation de parties contrôles est beaucoup plus simple que la précédente et surtout plus structurée, fait important lorsque la taille des algorithmes à représenter est très grande (comme cela est le cas pour réaliser des ordinateurs puissants). Toutefois, les contraintes dues à la limitation des surfaces utilisables sur silicium impose une optimisation de la surface occupée, optimisation mal respectée par ce type d'implantation. Une comparaison des taux d'occupation en surface entre une réalisation câblée et une réalisation microprogrammée d'un même circuit, le M6800, a été effectuée [OBR82]: la réalisation câblée occupe 57,5% de la surface totale du circuit contre 66% pour une réalisation microprogrammée du même algorithme de contrôle.

L'exemple suivant donne le schéma fonctionnel type d'une réalisation microprogrammée.

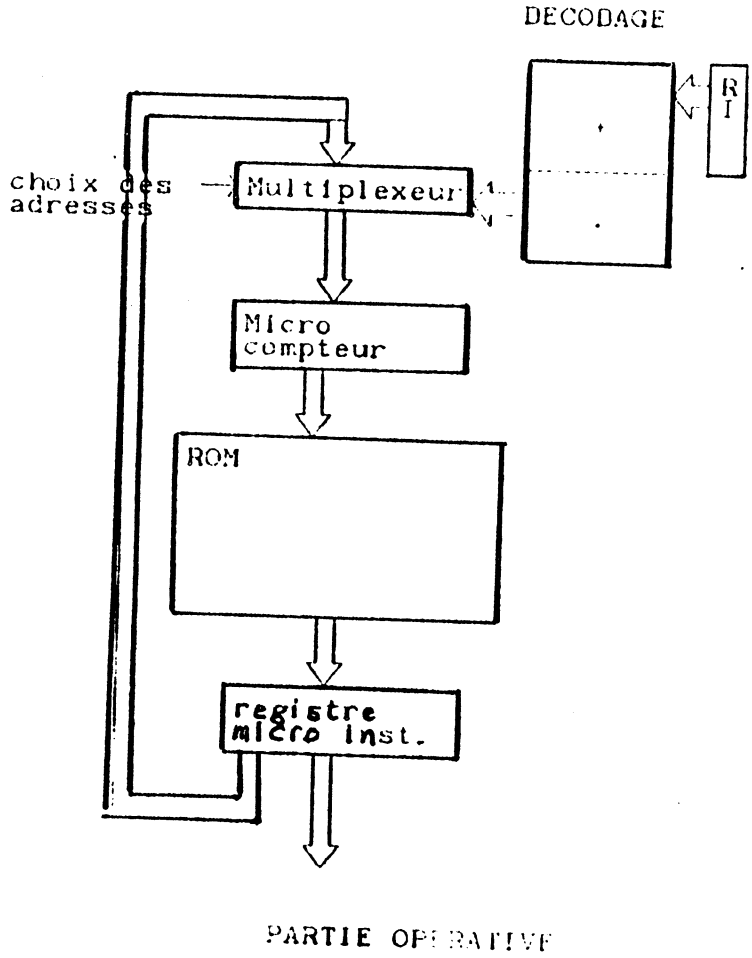


Figure 2.2: Représentation Fonctionnelle d'une Réalisation Microprogrammée

Quelques microprocesseurs ont eu leur partie contrôle réalisée ainsi: c'est notamment le cas du S2750. Sa structure interne est très déséquilibrée: 60% de sa surface est occupée par la ROM contenant le microprogramme. Il en résulte que son cycle de base est relativement lent (le temps de propagation des signaux à travers la ROM est proportionnel à sa taille).

2.2.3 Réalisation à l'aide de PLAs

A partir de représentations microprogrammées d'algorithmes de séquencement, de nombreux efforts ont été entrepris pour réaliser des optimisations en surface: entre 20 et 50% de diminution de surface a pu ainsi être obtenue [OBR82].

Le type de réalisation qui semble le plus adéquat pour la conception de circuits à très haute intégration consiste à décomposer fonctionnellement l'algorithme de contrôle en termes de sous-algorithmes coopérants et de les implanter physiquement séparément. . Chaque sous-algorithme est décrit comme un automate d'états finis par un ensemble d'équations logiques qui sont ensuite implantées sous forme de PLAs.

Un PLA (Programmable Logic Array, ou Table de Logique Programmable) est un ensemble de deux matrices ou 'plans'. Le premier plan est constitué d'un ensemble de portes logiques ET organisées sous forme structurée: il est assimilé à un circuit de décodage. Le second plan est un ensemble de portes OU qui a la même fonction qu'une mémoire morte (ROM).

La figure 2.4 propose une représentation d'un PLA ayant cinq entrées, A, B, C, E, VDD et trois sorties S1, S2, S3.

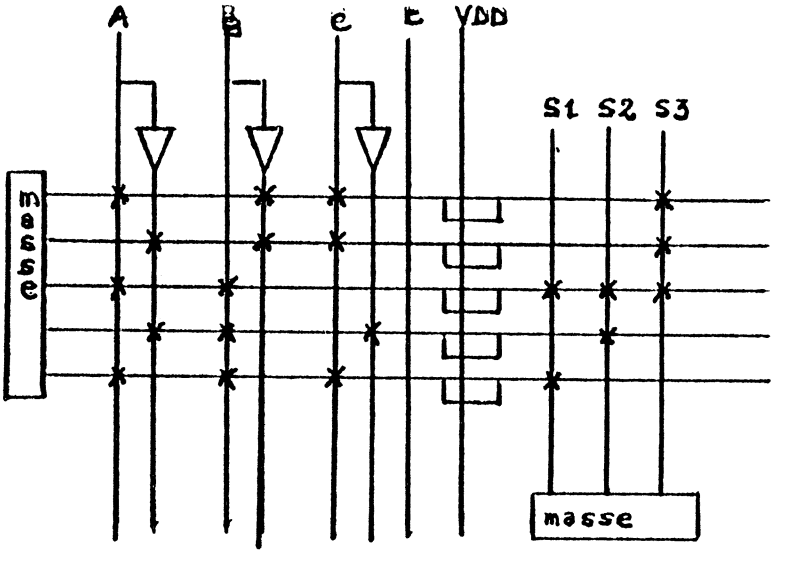


Figure 2.4: Représentation d'un PLA

Utilisant cette structure d'implémentation, la figure 2.5 propose un découpage fonctionnel du Motorola M6800 et sa représentation topologique.

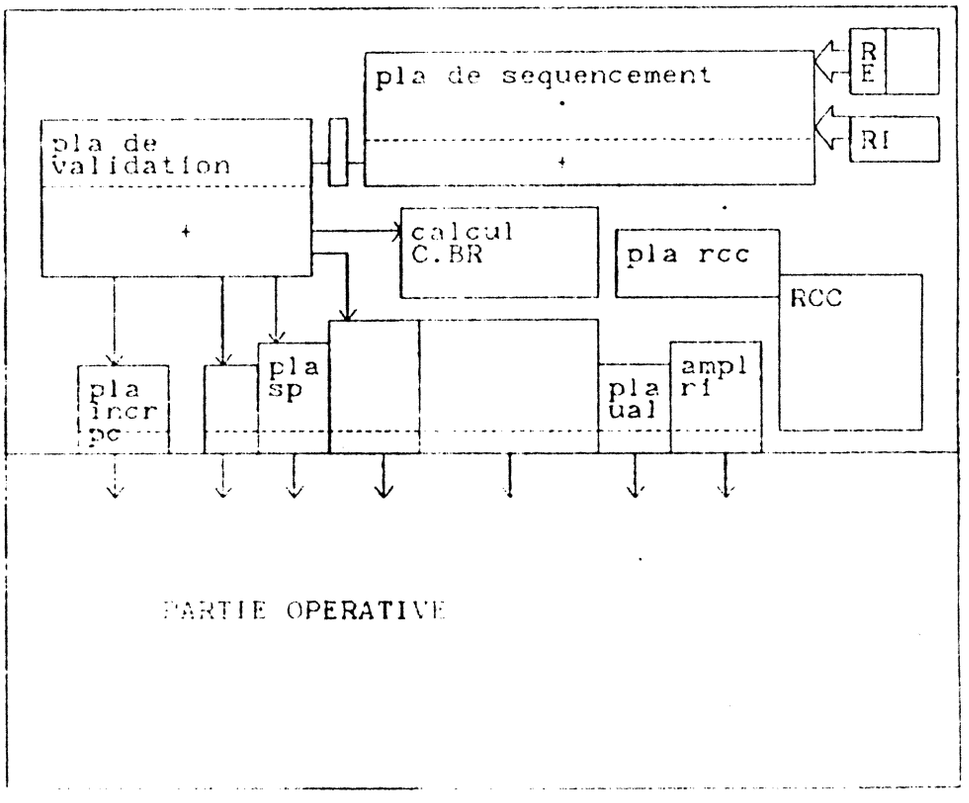


Figure 2.5: Représentation de la partie contrôle du M6800 à base de PLAs

Il est clair que le degré de liberté du concepteur quant au choix du découpage à réaliser est grand. Toutefois une trop fine segmentation fonctionnelle fait surgir des problèmes inexistants lors des réalisations mono-PLA ou mono-ROM. En effet, comment disposer ces blocs les uns par rapport aux autres de manière à minimiser leur distance par rapport aux organes de la partie opérative qu'ils contrôlent et à réduire le nombre de connexions? Les combinaisons possibles sont très nombreuses et avec l'augmentation de la complexité des algorithmes implantés aujourd'hui, des solutions ne peuvent être trouvées qu'avec l'aide d'outils informatiques. Un équilibre doit donc être trouvé entre une optimisation en surface et l'accroissement de la complexité des problèmes à résoudre lié à cet essai d'optimisation.

2.2.4 Réalisation basée sur une génération d'instants

La notion de temps est importante dans le contrôle du fonctionnement d'un microprocesseur. Ce temps est matérialisé par la présence d'une horloge de base bi-phasée (nous reviendrons plus en détail sur ces notions de synchronisation dans le chapitre suivant). Il est possible d'utiliser des représentations élémentaires du temps -uniquement les deux phases phi1 et phi2- comme c'est le cas pour le Motorola 6800. D'autres circuits, par contre, utilisent des combinaisons de ces phases de manière à s'abstraire du fonctionnement élémentaire du circuit. C'est le cas du circuit conçu par Intel, l'I 8085 où la notion d'instants a été introduite (un instant est une suite de phases élémentaires de longueur variable). Chaque instant permet de valider des propriétés générées par un décodeur d'instructions. Ces propriétés sont alors interprétées comme des commandes par les différents organes de la partie opérative. L'exemple suivant compare des circuits analogues: le ZILOG Z80 et l'INTEL I8085 qui interprétent un même jeu d'instructions et dont le séquencement interne de chacun est basé sur une génération d'instants.

Dans le cas du ZILOG, ces instants valident les propriétés issues d'un PLA grâce à un ensemble de portes logiques implantées de manière non structurée. Dans le cas de l'INTEL par contre, ces portes de validation sont organisées sous la forme d'un PLA . On remarque bien ici la différence de politique suivie par ces deux compagnies: ZILOG a créé un circuit compact (17 mm^2), rapide, avec un pourcentage important de logique aléatoire (cablée manuellement). INTEL, par contre, a créé un circuit plus large (25 mm^2), plus lent mais dont la conception a été plus automatisée: sa structure interne est plus régulière, plus lisible et plus simple à tester.

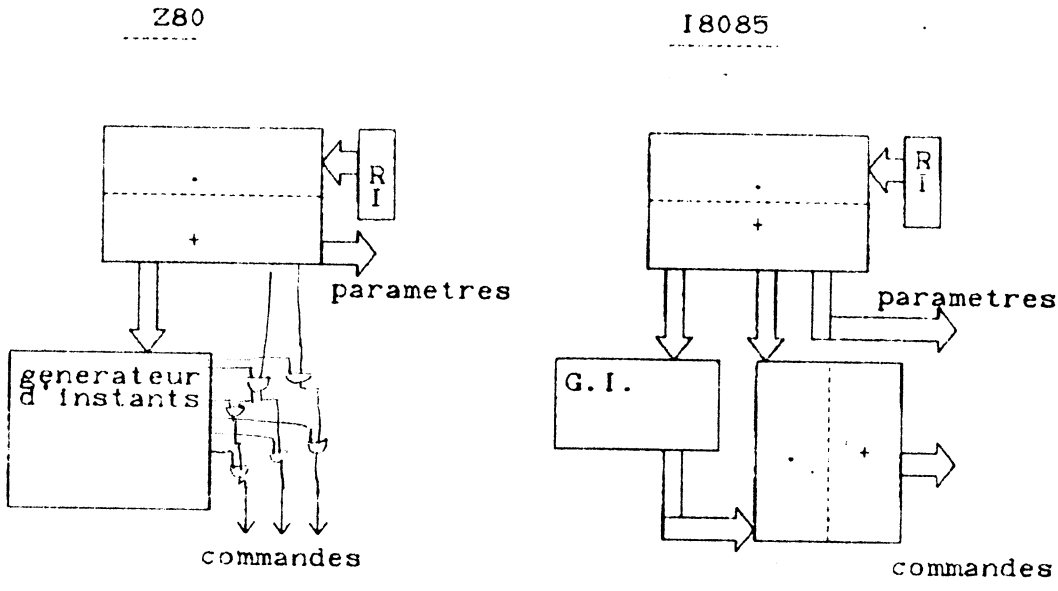


Figure 2.6: Parties contrôles des circuits Z80 et I8085

2.2.5 Interprétation Multi-Niveaux

La complexité croissante des algorithmes de contrôle a incité les concepteurs de circuits intégrés à étendre la notion d'instants et à créer des niveaux d'abstraction supplémentaires. Comme dans le cas précédent où des propriétés étaient validées par des combinaisons linéaires (nommées instants) de phases élémentaires phi1-phi2, des partionnements plus fins peuvent être réalisés (une sorte de factorisation successive) et de nouveaux groupes de propriétés peuvent être validés par des combinaisons linéaires de ces instants.

A chaque étape d'interprétation, de nouvelles unités de temps sont introduites qui valident ces propriétés en commandes d'organes de partie opérative. Des générateurs de temps (en général de simples registres à décalage) assurent leur production. De plus, des méchanismes de com-

mandes et de synchronisation entre ces différents générateurs de temps réalisent le lien entre ces étapes de décomposition.

Voici une représentation schématique d'une décomposition de séquencement sur deux niveaux d'interprétation. Les relations entre ces deux niveaux sont représentées par des signaux de synchronisation ou de commandes (relations Maître-Esclave entre eux).

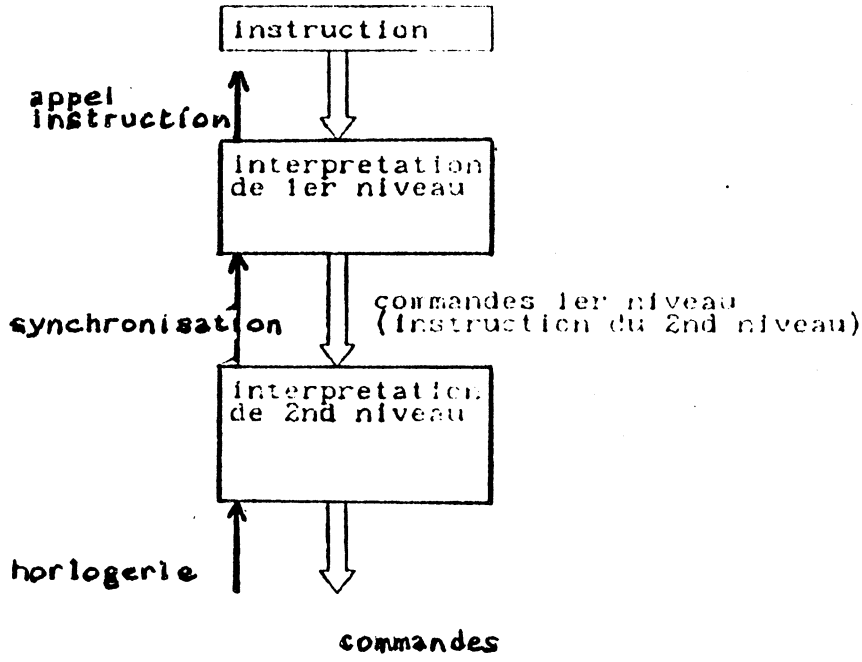


Figure 2.7: Décomposition de l'algorithme de séquencement en deux niveaux d'interprétation

2.2.6 Comparaison

L'étude de l'architecture interne des microprocesseurs conçus durant ces cinq dernières années permet de dégager une évolution très nette des techniques et méthodologies utilisées. La plus grande complexité des algorithmes à implémenter sur silicium s'est concrétisée par une plus grande segmentation et hiérarchisation des problèmes.

L'automatisation toujours plus importante de la phase de génération de structures physiques d'implantation impose aujourd'hui, pour des raisons de rentabilité, l'utilisation de PLAs comme support structurel de ces algorithmes de contrôle. Il n'en reste pas moins que les concepteurs restent libres de leur approche conceptuelle. L'utilisation de PLAs ne signifie pas que la microprogrammation n'existe plus. Seule sa représentation physique change car les contraintes physiques ne sont plus les mêmes (essentiellement des optimisations de surface pour augmenter la densité).

La génération d'instants est une alternative possible de la microprogrammation qui permet de hiérarchiser les générations de commandes. Cette méthode de conception va aussi dans le sens d'une plus grande densité des structures physiques générées transformant une génération mononiveau de commandes (ROM de microprogramme) en une génération multi-niveau (PLAs de validation, registres à décalage, etc..).

La méthode de réalisation câblée, bien que n'étant plus utilisée systématiquement, a encore son utilité. En effet, même aujourd'hui où les méthodes d'implantation sont beaucoup plus automatisées, il est souvent nécessaire d'avoir recours à cette méthode manuelle pour représenter des portions de circuits qui ne peuvent pas être générées automatiquement.

2.3 LES OUTILS D'AIDE A LA CONCEPTION ET A LA SYNTHESE DE PARTIES CONTROLES

2.3.1 Introduction

L'accroissement rapide de la complexité des circuits qu'il était possible d'intégrer a nécessité le développement d'une méthodologie de conception servant de référence de base pour le développement d'outils informatiques d'aide à la conception, à la production et au test de circuits intégrés. Nous en présentons ici les grandes lignes et nous aborderons ensuite le travail réalisé à Grenoble dans ce domaine dans le groupe d'Architecture des Ordinateurs en le situant par rapport aux recherches et résultats obtenus dans ce domaine en Europe et aux Etats Unis.

2.3.2 Description Hiérarchisée

S'ils étaient écrits en termes d'équations logiques, les algorithmes décrivant le fonctionnement interne d'un microprocesseur pourraient faire aujourd'hui l'objet d'un manuel de quelques centaines de pages et seraient très peu compréhensibles pour quiconque ne les aurait pas écrits.

Dès 1968, des outils de description fonctionnelle ont été développés. Ces outils sont en fait des langages de haut niveau, faciles à utiliser et permettant de décrire la logique des fonctionnements de manière lisible. Ces descriptions de 'haut niveau' donnent des renseignements sur le comportement global du circuit mais ne fournissent aucune indication sur la manière dont le circuit sera représenté physiquement.

En fait, la conception d'un circuit intégré, et en particulier la conception de sa partie contrôle, nécessite l'utilisation d'un ensemble de langages qui permettent de décrire ce circuit (ou cette partie contrôle) à différents niveaux de détail. Chacun de ces niveaux correspond à l'introduction d'informations supplémentaires nécessaires à l'implantation physique de ce circuit.

L'exemple suivant montre comment ces niveaux sont reliés entre eux par des interprétations successives de manière à fournir une description complète et détaillée du fonctionnement électrique et de la représentation géométrique (et disposition physique) du circuit.

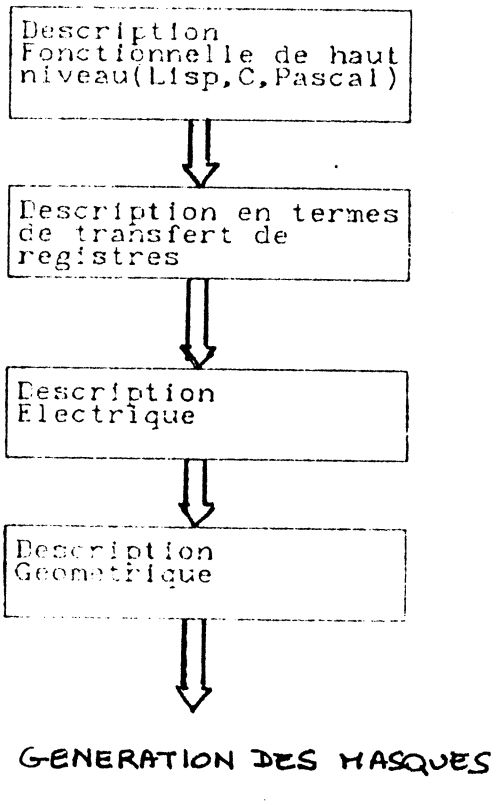


Figure 2.8: Interprétations successives de descriptions de circuits intégrés

A l'heure actuelle aux Etats Unis, des langages tels que LISP ou C ont inspiré le développememt de sous-langages spécifiques à la conception de circuits intégrés tels que DPL [BAT80] ou VIP (langage procédural de haut niveau développé à l'université de Stanford). Des interpréteurs de ces langages fournissent des descriptions intermédiaires qui servent de source pour des programmes de génération de structures, des programmes de test ou de compactage. (Nous reviendrons plus en détail sur ces outils dans le chapitre suivant).

En Europe aussi, des outils de description de comportement de systèmes logiques ont été développés tels que CAP ou GRAPHCET (inspiré des Réseaux de Pétri). Philips et Signetics ont utilisé CASSANDRE comme outil de description de comportement de systèmes logiques pour décrire les 2750 et 2752.

Notre but à Grenoble a été de mettre sur pied un système aussi complet que possible d'outils d'aide à la conception de circuits intégrés. Dans cette optique, nous nous sommes intéressés au développement d'un langage de description de matériel de type RTL (Langage de Transfert de Registres, appelé IRENE). Ce type de langage de description est plus fin que des descriptions comportementales de haut niveau. Il donne des indications sur les structures d'implantation choisies pour contenir les flots de données et de contrôle ainsi que sur le séquencement des activations de ces différentes structures. Les descriptions obtenues sont donc à la fois structurelles et comportementales.

En parallèle, nous nous sommes intéressés au développement d'un formalisme pour la description de circuits logiques. Son but est de fournir des descriptions détaillées sur les circuits combinatoires et logiques définis au niveau supérieur de Transfert de Registres. Ces descriptions sont ensuite utilisées par un ensemble de programmes pour générer des descriptions topologiques des masques (nécessaires pour la fabrication de ces circuits).

2.3.3 Introduction au langage RTL IRENE

Le langage de description de matériel qui est en cours d'implémentation sur Multics (MULTICS est le système d'exploitation implanté sur l'ordinateur HB68 de l'université de Grenoble), présente les caractéristiques suivantes:

1. Langage procédural de haut niveau (sur-ensemble de Pascal) dont l'utilisation est simple pour le concepteur. Il permet:

   a. de définir clairement les modules constituant le circuit intégré;

   b. d'appeler des procédures et des fonctions externes résidant en bibliothèque (exemple: la représentation structurelle des cellules de base constituant les mémoires ROMs ou RAMs, les UALs ...);

   c. de définir de manière structurelle les différentes fonctions combinatoires réalisées par le circuit (exemple: circuit de décalage, encodeur/décodeur, fonctions trigonométriques...);

   d. d'utiliser une syntaxe et une sémantique relativement simples permettant une correspondance bijective entre les différentes variables, opérateurs, fonctions utilisées et leur structure physique. Il permet par exemple de définir deux types de variables: (1) les variables mémorisées comme les Registres, les Latches, les RAMs; (2) des variables instantanées comme les Bus ou les Multiplexeurs. Il permet aussi de définir des structures telles que l'UAL, les circuits à décalage et de les utiliser en tant que variables;

   e. de décrire le comportement d'un circuit à l'aide d'un nombre réduit d'instructions conditionnelles, ou de branchement que l'on peut représenter physiquement de manière directe: la structure conditionnelle, par exemple, est représentée par un multiplexeur.

Des macros fonctions ainsi que des procédés de synchronisation ont été introduits de manière à modulariser et à réduire les descriptions.

La modularité et la sémantique du langage permettent:

1. une interprétation rapide, soit directement au niveau graphique pour décrire une structure d'implantation, soit au niveau logique pour décrire le fonctionnement d'un organe de contrôle par exemple;

2. le développement d'outils tels que des simulateurs, extracteurs de schémas logiques, de paramètres pour évaluer les performances du circuit (évaluation de la surface, du temps d'exécution des différentes instructions, etc..).

En particulier, dans la mesure où la partie contrôle se présente comme une entité constituée de blocs fonctionnels distincts, il est aisé de les concevoir séparément, de décrire leur comportement et d'en déduire leur représentation graphique.

2.3.4 Formalisme DELTA

2.3.4.1 Motivations

Le formalisme DELTA (Description d'Eléments Logiques pour la Traduction Automatique [ETI81], [NEM81] a été développé avec les objectifs suivants:

1. fournir une description d'éléments logiques rencontrés dans les circuits intégrés réalisés aujourd'hui. Les éléments sémantiques de base sont la 'copie symbolique' des composants électroniques classiques utilisés dans les microprocesseurs actuels;

2. fournir aux concepteurs de circuits logiques un outil de description de réseaux de portes logiques qui permettent d'en réaliser une génération automatique et structurée.

2.3.4.2 Bijection Formalisme DELTA - Représentation Matérielle

Une condition nécessaire liée à la génération automatique de portes logiques consiste à définir une correspondance bi-univoque entre les constituants du formalisme utilisé et les éléments physiques qu'il est supposé décrire.

Le formalisme DELTA en propose une, faisant intervenir des représentations symboliques d'agglomérats de portes élémentaires NAND, NOR et d'inverseurs. Aucune présupposition n'a été faite sur la manière dont ces éléments seront implantés sur silicium, ni sur le type de logique (dynamique, statique, inverseuse ou non), ni sur la technologie utilisée. Il nous est apparu nécessaire de nous dégager, à ce niveau de description, des contraintes électriques et technologiques.

On peut donc considérer le niveau de description DELTA comme le niveau pivot entre des descriptions fonctionnelles et des descriptions

électriques et géométriques, une sorte de niveau de logique formelle d'où une plus grande fonctionnalité a été mise en évidence.

2.3.4.3 Notion d'ETA

Tout processeur de logique peut être vu comme l'assemblage d'un certain nombre de circuits séquentiels (éléments de mémorisation, comme des bascules RS ou des registres) et de circuits combinatoires (exécutant des opérations logiques, comme des unités arithmétiques et logiques par exemple).

Si l'on ne s'intéresse qu'à leurs caractéristiques fonctionnelles, ces deux types de circuits logiques peuvent être considérés comme des 'boîtes noires' réalisant une certaine fonction. Les échanges d'information avec l'environnement extérieur sont réalisées à travers une interface clairement définie.

Nous avons concrétisé cette abstraction par la notion d'ETA. Un ETA est la représentation symbolique d'un triplet (E,T,A) où

E: représente l'ensemble des Entrées d'une boîte noire donnée,
T: la fonction de Transfert que réalise cette boîte noire,
A: la place (point de mémorisation physique ou symbolique: étiquette) caractérisant l'état d'Activité de cette boite. A cette place est aussi associée la sortie unique de cette boîte.

Nous avons dégagé deux types d'ETAs fonctionnellement distincts:

1. les ETA séquentiels auxquels sont associés des éléments de mémorisation physique qui peuvent être soit synchronisés sur une horloge de base, soit validés par certaines conditions;

2. les ETA combinatoires où la mémorisation est inexistante.

2.3.4.4 ETA séquentiel synchrone

Un ETA séquentiel synchrone est représenté graphiquement comme suit:

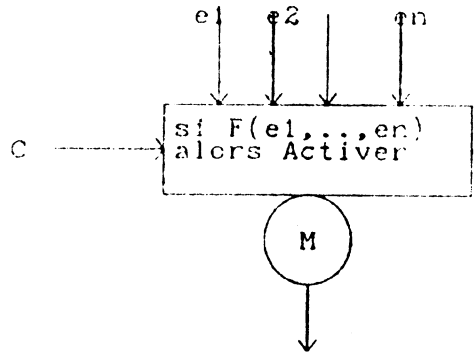


Figure 2.9: ETA séquentiel synchrone

où $(e_1,..,e_n)$ représente l'ensemble des entrées de l'ETA M, $F(e_1,..e_n)$ est la barrière d'activation de cet ETA et M la place associée au point de mémorisation.

L'instruction associée à cet ETA est une 'affectation' et est représentée par:

$$M := F(e_1,...,e_n)$$

Lorsque la commande C est activée, l'élément de mémorisation représenté par la place M est chargé de la valeur de la fonction booléenne $F(e_1,...,e_n)$.

2.3.4.5 ETA séquentiel asynchrone

Un ETA séquentiel asynchrone est représenté graphiquement comme suit:

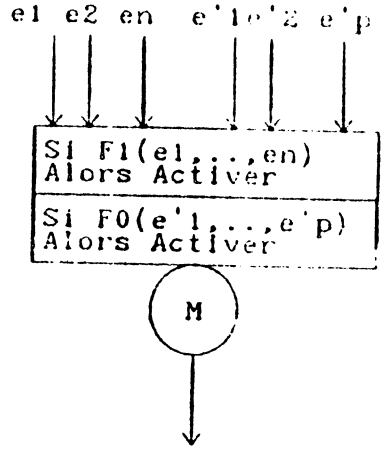


Figure 2.10: ETA séquentiel asynchrone

Dans ce cas, deux barrières d'activation sont associées à la place M. Celle-ci prend la valeur 1 si la fonction booléenne F1 vaut 1, 0 si F0 vaut 1, reste inchangée si F1 et F0 valent toutes deux zéro et est indéfinie si F1 et F0 valent toutes deux un.

Un ETA asynchrone sert à représenter des éléments de mémorisation tels que les bascules RS, par exemple. L'instruction qui lui est associée est représentée ainsi:

si F1=1 et F0=0 alors M := 1;
si F0=1 et F1=0 alors M :=0;
si F1=0 et F0=0 alors M inchangée;
si F1=1 et F0=1 alors INDETERMINATION;

2.3.4.6 ETA combinatoire

Un ETA combinatoire est représenté graphiquement comme suit:

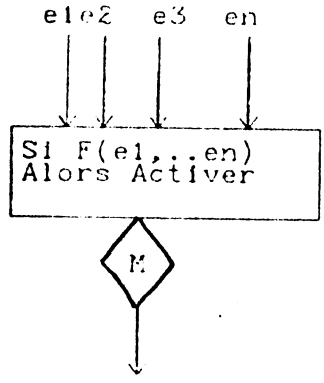


Figure 2.11: ETA combinatoire

Dans ce cas, la barrière d'activation n'en est plus exactement une puisque la fonction $F(e_1,..e_n)$ est évaluée dès que ses entrées sont activées. Le losange symbolise le point mémoire virtuel associé à la place M.

L'instruction associée à un ETA combinatoire est appelée 'instruction de connexion' et est représentée comme suit:

$$M = F(e_1,...,e_n)$$

La représentation sous la forme d'instruction conditionnelle de la fonction de transfert de cet ETA n'existe que pour maintenir l'homogénéité du formalisme. En réalité, la place M n'a ici aucun sens physique si ce n'est celui d'étiquette associée à la sortie du circuit combinatoire $F(e_1,...,e_n)$.

2.3.4.7 Eclatement et Codage des ETAs

De manière à diminuer la taille des descriptions que le concepteur est amené à manipuler, la notion d'ETA codé ou 'pré synchronisé' a été introduite. Cet ETA se représente graphiquement de la manière suivante:

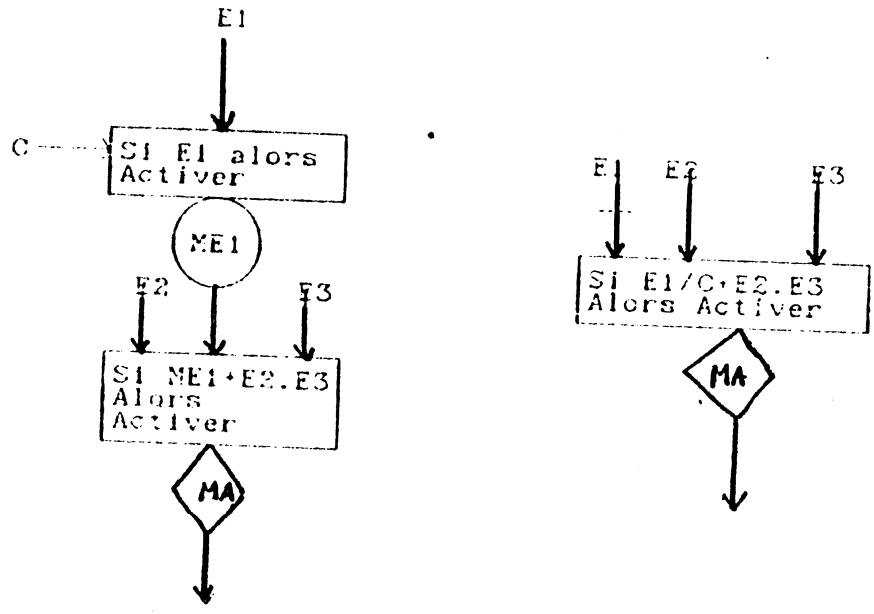


Figure 2.12: ETA présynchronisé

Ces deux représentations sont équivalentes fonctionnellement.

De même, de manière duale, il est possible de segmenter des ETAs de manière à pouvoir, par exemple, introduire des points de test. Le concepteur peut ainsi décomposer ses boîtes noires et les réduire en fonction de l'action qu'il veut entreprendre: simuler, tester ou générer le composant logique qu'il conçoit.

2.3.4.8 Comparaison avec les Réseaux de Pétri

La démarche ascendante qui consiste à utiliser des résultats obtenus pour modifier et améliorer les étapes de conception aboutissant à ces résultats est intéressante du point de vue du concepteur d'outils d'aide au développement de circuits. En effet, il est souvent difficile de juger a priori, de l'efficacité d'un outil s'il n'a pas été mis en utilisation et testé par des concepteurs.

Toutefois, l'état d'esprit dans lequel un concepteur de circuits intégrés aborde la conception est quelque peu différent. Il suit de manière générale une approche descendante. Décrire une partie contrôle consiste à décrire les tâches que doivent exécuter différents processus pour contrôler les organes de la partie opérative. Augmenter le parallélisme entre ces processus est l'une des préoccupations du concepteur.

Différents outils de modélisation existent déjà pour décrire des problèmes de synchronisation entre processus, de partage de ressources, etc. Les Réseaux de Pétri, Graphcet ont été développés à cette fin et l'on pourrait remettre en question le développement d'un nouveau formalisme de description de systèmes logiques, d'automates d'états finis alors qu'il en existe déjà de nombreux.

Il est vrai qu'il existe de nombreuses ressemblances entre les concepts utilisés dans les Réseaux de Pétri et ceux sous-jacents au formalisme DELTA: la notion de place, d'activation de barrières de transition, de jeton analogue à la propagation de signaux logiques. La figure suivante propose deux représentations d'un même automate d'états finis, l'une à l'aide du formalisme Delta, l'autre à l'aide des Réseaux de Pétri.

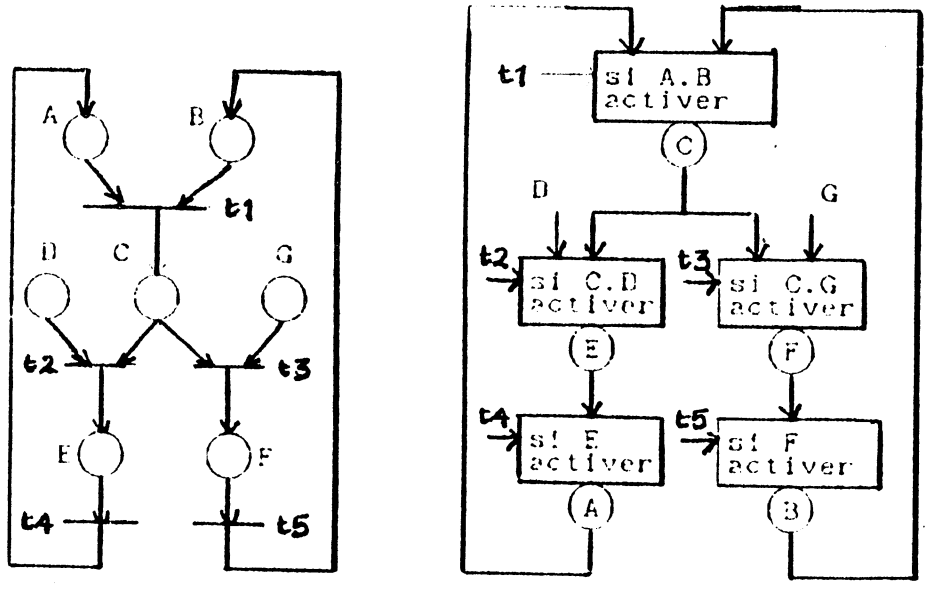


Figure 2.13: Comparaison DELTA-Réseaux de Pétri

Mais la principale différence entre DELTA et les Réseaux de Pétri se situe dans le niveau de représentation auquel l'on se situe.

Les Réseaux de Pétri permettent de représenter des problèmes d'ordre fonctionnel de parallélisme, de synchronisation, et font abstraction de la manière dont ces processus seront implantés plus tard physiquement.

Le formalisme DELTA, par contre, prend beaucoup plus en compte des considérations d'ordre structurel et son rôle principal n'est pas de formaliser des algorithmes de haut niveau mais plutôt d'être une étape intermédiaire entre une description fonctionnelle de haut niveau et son implantation physique. Il doit permettre au concepteur de représenter graphiquement des représentations logiques de systèmes d'où il peut en extraire rapidement le fonctionnement.

Une des conséquences de cette différence réside dans la représentation du séquencement. Dans les Réseaux de Pétri, la présence d'une horloge de base n'apparaît pas explicitement. Seuls des changements de valeurs des

transitions font évoluer le système. Le formalisme DELTA donne des renseignements plus précis sur la nature de ces transitions. Il indique par exemple qu'une commande est une combinaison linéaire de phases d'horloge. Des contraintes plus nombreuses doivent être prises en compte dans la mesure où le niveau de description est plus fin. La représentation du séquencement est alors explicitement introduite dans chaque constituant de base.

2.3.4.9 Stockage des algorithmes représentés dans le formalisme DELTA

Il est nécessaire de garder une bibliothèque de programmes concernant la représentation des algorithmes que l'on désire implanter sur silicium. Cela permet à différents programmes d'accéder aux informations dont ils ont besoin pour réaliser leur tâche (simulation, génération, compactage etc...).

On peut ainsi établir une bibliothèque d'outils d'aide à la conception de parties contrôles qui travaillent sur une sorte de banque de données centrale (pour l'instant réduite à un ensemble de fichiers dont l'accès n'est pas géré de manière spécifique, si ce n'est par Multics).

1. Représentation des ETAs

A chaque ETA est associée une entrée dans une table contenant l'ensemble de ses caractéristiques: son nom (nom associé au point de mémorisation physique ou virtuel), son type (séquentiel synchrone ou asynchrone, combinatoire), sa fonction d'activation, la commande ou phase élémentaire sur laquelle il est activable, et l'ensemble de ses successeurs.

Un analyseur sémantique vérifie le texte source et se charge de remplir cette table séquentiellement. Le champ **Type** indique si l'ETA est de type séquentiel ou combinatoire; **Lfctn** renseigne sur la longueur dans la table des fonctions d'activation de la fonction associée à l'ETA considéré; **Fctn** est l'adresse de début de cette fonction dans la table; **Adrmem** est l'adresse de début de la liste des successeurs de l'ETA courant dans la table représentant le graphe de séquencement. **Nbc** et **Nbs** sont respectivement

les nombres des ETAs successeurs et des commandes issues de l'ETA courant. Ptr est le lien chaînant l'ETA courant à l'ETA suivant ayant la même initiale que lui (utilisé par la fonction de hash-code).

La figure suivante indique la manière dont ces informations sont rangées en mémoire.

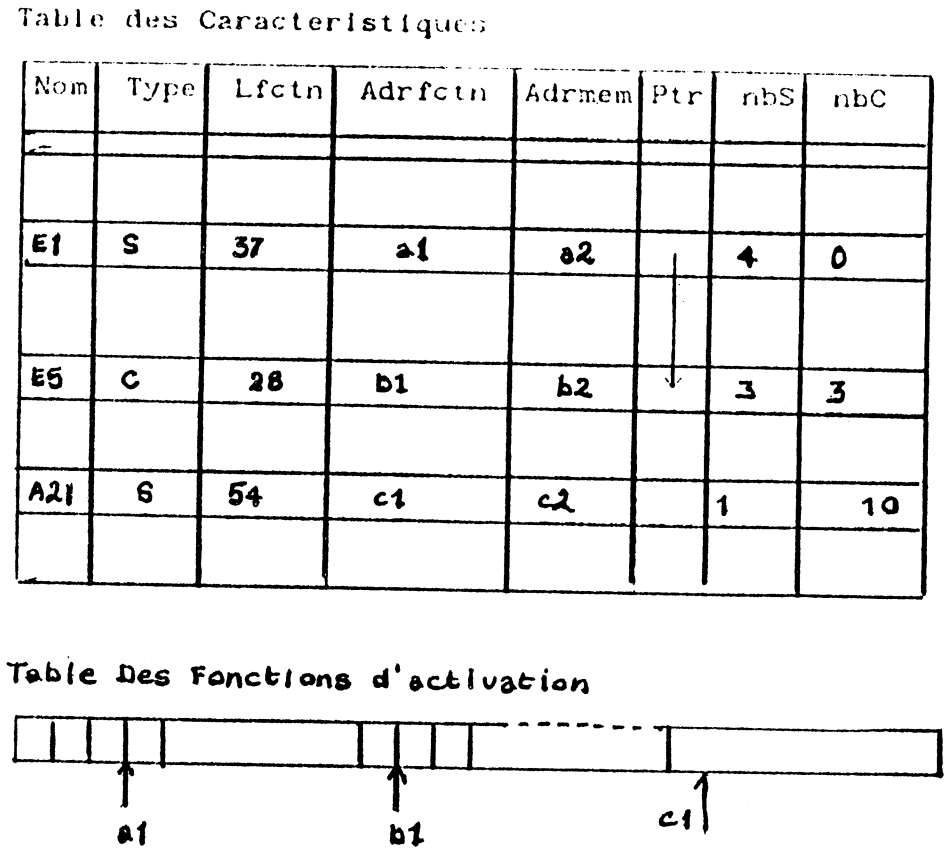


Figure 2.14: Représentation de la table des caractéristiques

2. Accès rapide par nom

L'implantation d'une fonction de hash-code garantit une accès rapide par nom des ETAs mémorisés. Une liste par initiale de nom d'ETA a été créée. Chaque entrée de la table des ETAs est liée à l'entrée suivante dont le nom commence par la même initiale. Une table de pointeurs de tête et de fin de liste pour chaque lettre de l'alphabet permet de compléter cette structure d'accès par nom.

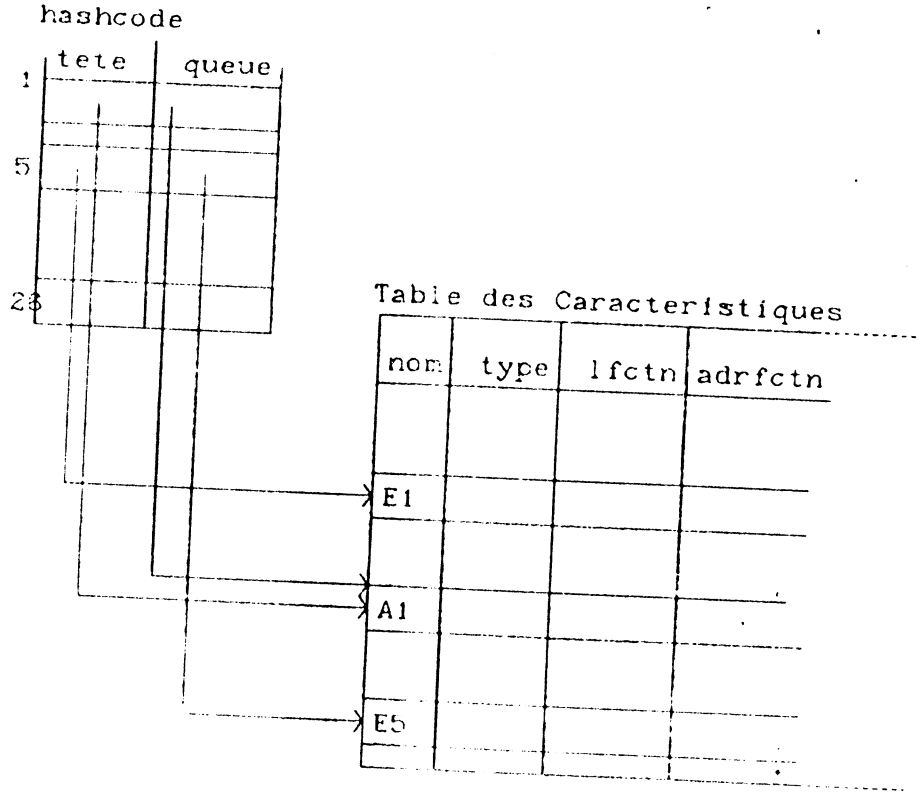


Figure 2.15: Structure d'accès par nom

3. Représentation de l'algorithme de séquencement

La structure du graphe de séquencement a été représentée par une table d'indices accédée comme suit: à chacun des ETAs possédant une entrée dans la table des caractéristiques est associé un pointeur vers la table du séquencement. Ce pointeur donne l'adresse à partir de laquelle est rangé, dans la table du séquencement, la séquence des indices, dans la table des caractéristiques, de tous ses ETAs successeurs. Le schéma suivant indique le lien entre ces deux tables

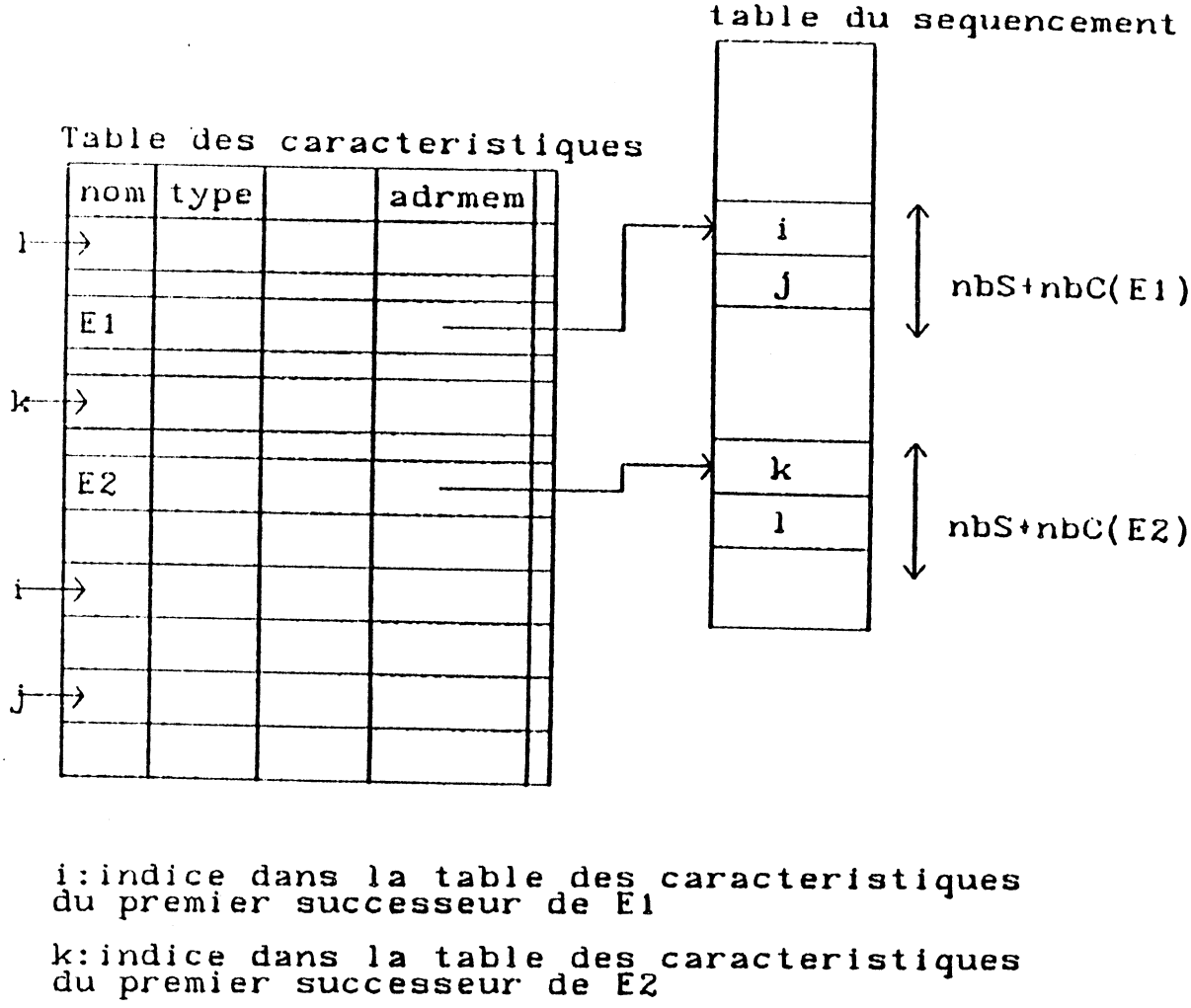


Figure 2.16: Table du graphe de séquencement

2.3.5 Simulateur DELTA

. Un simulateur a été développé dans le principal but de valider le formalisme DELTA et d'en estimer l'adéquation par rapport à la description des réseaux de portes logiques.

2.3.5.1 Principe de base

Ce simulateur est basé sur l'interprétation et l'exécution directe des instructions DELTA décrivant le fonctionnement des algorithmes à simuler.

Il suit pas à pas l'évolution du flot de contrôle à travers l'ensemble des ETAs qui définissent la structure d'implantation de l'algorithme et anticipe l'état dans lequel sera ce flot de contrôle sur la phase élémentaire suivante.

Sur chaque cycle élémentaire de l'horloge de base -biphasée- du système, l'évaluation de l'ensemble des ETAs prêts à être activés est réalisée ainsi que la détermination des ETAs qui seront activables sur le cycle suivant.

2.3.5.2 Fonctionnement

Un certain nombre d'actions sont exécutées de manière cyclique à chaque changement de couple de phases élémentaires: (phi1,phi2).

Le simulateur exécute les instructions décrivant chacun des ETAs de l'algorithme grâce à la séquence d'actions suivantes:

1. déterminer l'ensemble des commandes et des conditions d'activation valides sur la phase courante;

2. évaluer l'ensemble des ETAs séquentiels dont les prédécesseurs étaient actifs sur la phase précédente. Cette anticipation des futurs ETAs à activer permet de diminuer le nombre total d'évaluations de manière importante;

3. évaluer l'ensemble des ETAs combinatoires successeurs des ETAs séquentiels déterminés actifs en phase 2 de façon à connaître l'ensemble des ETAs séquentiels qui seront activables sur la phase suivante.

Un détail des algorithmes est présenté en Annexe 1.

2.3.5.3 Structure de données

Deux tables de rôle parfaitement symétrique sont définies. Elles sont alternativement utilisées en lecture et en écriture à chaque changement de phase. Si, sur la phase courante, la table T1 est utilisée en lecture, la liste des ETAs séquentiels qu'elle contient sera évaluée (ou plutôt, la fonction booléenne d'activation associée à cet ETA sera évaluée). La table T2, utilisée en écriture contiendra, après évaluation et mémorisation des ETAs combinatoires évalués, la liste des ETAs séquentiels qui devront être évalués sur la phase suivante.

2.3.5.4 Exemple

Voici un exemple illustrant le fonctionnement du simulateur sur une phase élémentaire phi-i ('+' symbolise le 'ou' logique et '.' le 'et' logique):

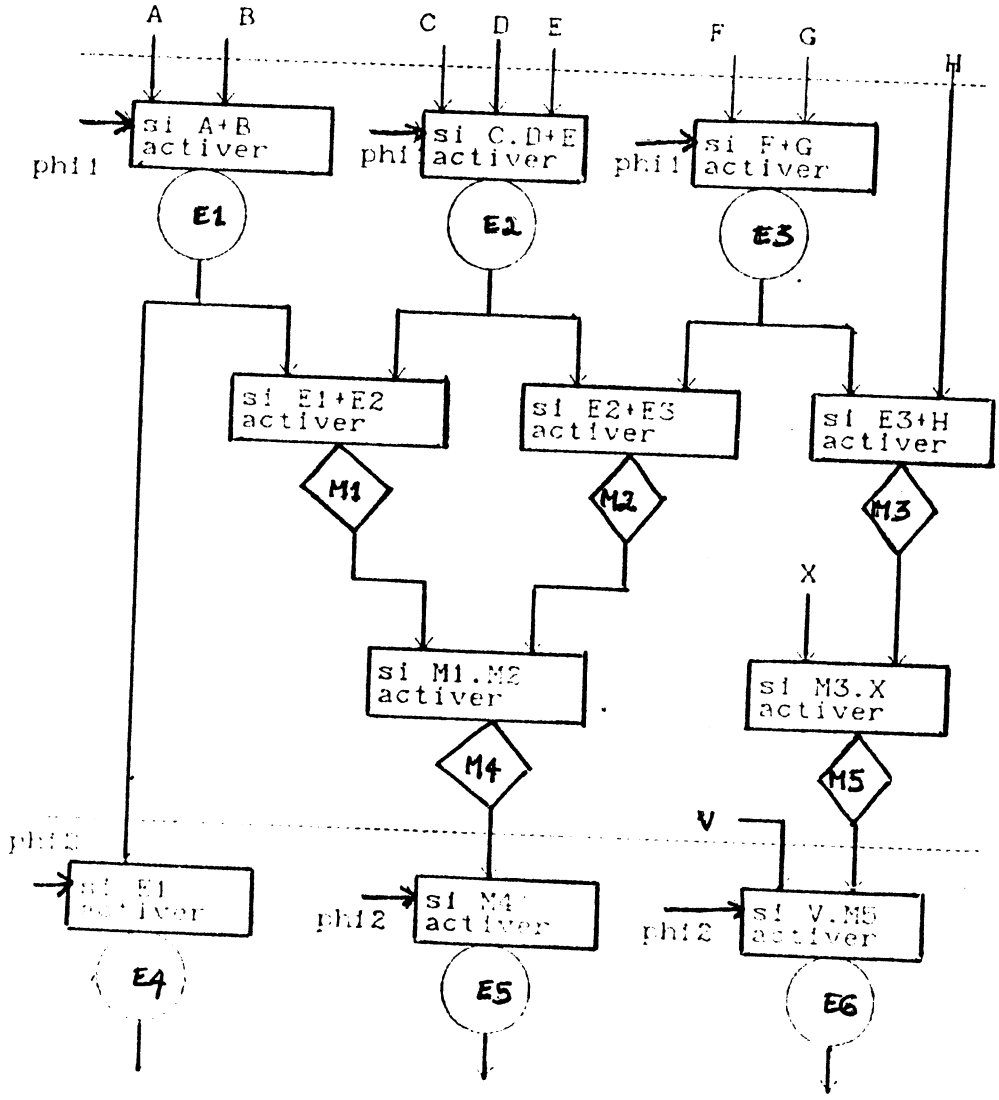


Figure 2.17: Exemple de description DELTA

Soient E1,E2,E3 les trois ETAs séquentiels activables en phase phi1, mémorisés dans la table T1. Supposons que l'évaluation de la fonction d'activation associée à ces ETAs ait déterminé E1 et E2 comme étant effectivement actifs.

L'ensemble des ETAs combinatoires issus de ces deux ETAs, E1 et E2, est évalué soit M1, puis M2,M4. L'évaluation des ETAs combinatoires est récursive et avance pas à pas en évaluant en chaîne tous les ETAs combinatoires successeurs d'ETAs combinatoires jusqu'à leur épuisement.

Dès qu'un ETA combinatoire possède comme successeur un ETA séquentiel, celui-ci est mémorisé dans la table utilisée en écriture. Dans notre exemple, E4 sera écrit dans T2; M1, M2 et M4 seront mémorisés dans des tables intermédiaires, puis évalués. Enfin, E5 sera rangé dans T2.

Puis, sur la phase phi2 suivante, la table T2 contenant E4 et E5 sera lue à son tour, et la table T1 sera utilisée en écriture. Les deux tables contiendront les informations suivantes pendant cette période élémentaire:

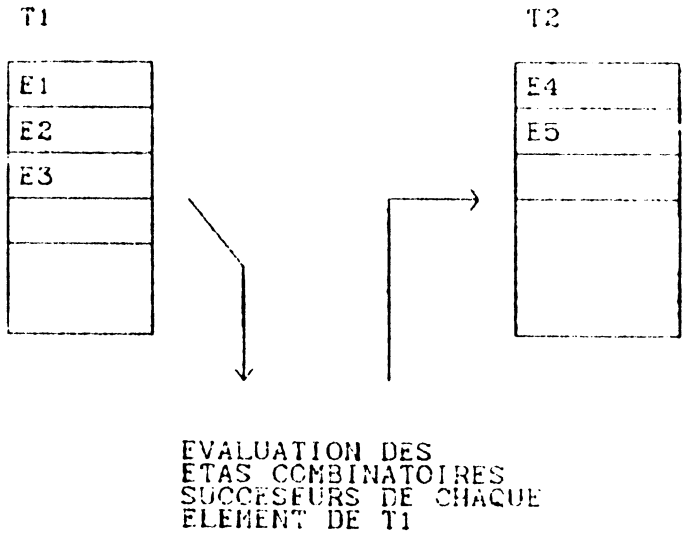


Figure 2.18: Illustration du contenu des deux tables T1,T2 après simulation du flot de contrôle sur une phase élémentaire

2.3.5.5 Conclusions

Ce simulateur a été testé sur quelques exemples dont la partie contrôle du circuit M6800. Il a permis de mettre en évidence un certain nombre d'erreurs de fonctionnement [NEM81].

Ses performances sont intéressantes car le temps moyen de simulation est d'environ 10 secondes pour une centaine d'ETAs ce qui représente à peu près un accroissement de 30% de la vitesse par rapport au simulateur de CASSANDRE par exemple.

Ce résultat s'explique par le fait que la quantité de données manipulées est minimale. Elle se limite en effet, sur chaque phase élémentaire, à l'évaluation des ETAs dont les prédécesseurs étaient actifs sur la phase antérieure.

2.3.6 Synthèse de PLAs à partir d'une description DELTA

Comme nous l'avons indiqué précédemment, une description d'algorithmes en formalisme DELTA permet la transition entre des descriptions de comportements et des descriptions de structures. Une telle description peut être générée soit automatiquement par des interpréteurs de langage RTL en formalisme DELTA soit 'manuellement' à partir des descriptions fonctionnelles de ces algorithmes.

La description d'algorithmes à ce niveau de conception reflète donc les découpages hiérarchiques et fonctionnels décidés lors des phases antérieures de la conception. Comment, par exemple, organiser géographiquement les barrières temporelles (registres, bascules RS) par rapport aux éléments de calcul combinatoire? Ces décisions influencent considérablement la composition structurelle de tous ces blocs fonctionnels. Nous supposerons donc que toutes ces décisions ont été prises et sont reflétées dans la structure même des descriptions DELTA fournies par le concepteur.

La description de chacun de ces blocs dans le formalisme DELTA est alors interprétée et générée automatiquement par un programme de synthèse de PLA que nous présentons ici. Nous avons choisi le PLA comme structure type d'implantation d'algorithmes car il semble être la structure la plus adaptée pour la génération automatique de circuits. Il fournit, une fois optimisé, des structures denses ce qui est important en LSI. En effet, la principale caractéristique des PLAs est la possibilité de représenter en parallèle des actions qui se déroulent en séquence, ce qui contribue à augmenter la densité des schémas d'implantation.

La figure suivante illustre cette tranformation de séquentialité en parallélisme grâce à l'utilisation de PLA.

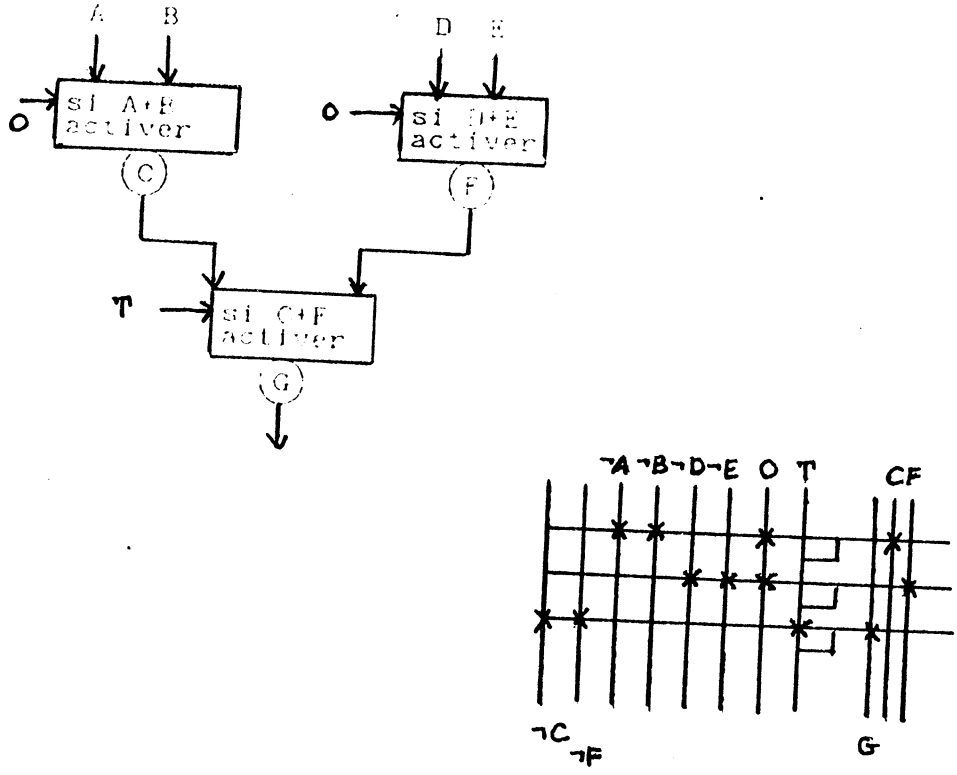


Figure 2.19: Passage d'une représentation séquentielle à une représentation parallèle

2.3.6.1 Etapes nécessaires à la génération de PLAs

Le passage d'une description séquentielle d'ETAs (ce que fournit une description DELTA) à une description parallèle est réalisée à l'aide d'un tri: La description DELTA est parcourue et reclassée de façon à regrouper les ETAs par phase d'activation commune. Les actions suivantes doivent être réalisées pour effectuer ce regroupement:

1. Eliminer l'ensemble des ETAs combinatoires en les intégrant aux ETAs séquentiels dont ils sont les prédécesseurs. Ceci est réalisé facilement dans la mesure où un ETA combinatoire peut grossièrement être assimilé à sa fonction de transfert. Un regroupement des fonctions est alors effectué. La figure suivante illustre ce regroupement. Il propose une forme condensée de l'exemple donné à la figure 2.17.

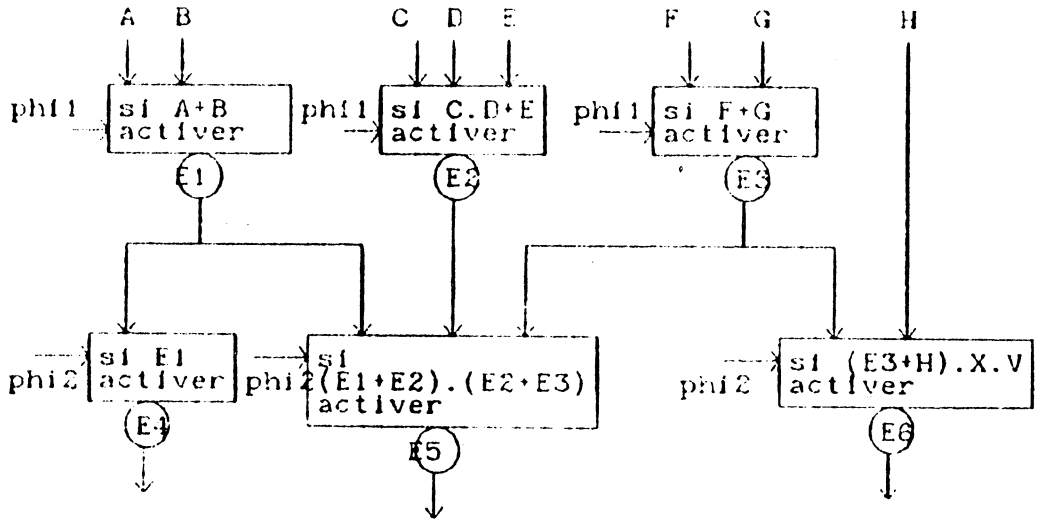


Figure 2.20: Représentation Condensée de l'Exemple de la Figure 2.17

Cette opération a pour but de n'obtenir que des ETAs séquentiels. Cela permet la transformation automatique des fonc-

tions d'activation de tous les ETAs présents en vue de leur génération sous forme de PLA.

2. Définir pour chaque ETA obtenu la phase élémentaire sur laquelle il est activable. Cela signifie en particulier, et si besoin est, de retrouver la phase sur laquelle la commande d'activation de cet ETA se déroule.

3. Réduire l'ensemble des successeurs activables à partir d'un ETA actif, de manière à produire une sortie unique. Cette condition d'unicité des successeurs actifs est nécessaire pour une génération de PLAs.

4. Transformer cette fonction d'activation de façon à ce qu'elle se présente sous la forme d'une somme de produits (cette représentation est nécessaire pour la génération des deux plans du PLA).

5. Trouver un codage optimal des ETAs pour chacun de ces groupes.

Lorsque toutes ces étapes ont été exécutées, la génération de deux matrices bits associées à ce groupe peut alors être effectuée. Ces deux matrices servent ensuite de source à un ensemble de programmes qui se chargent de les optimiser et de générer les descriptions des masques associés.

2.3.6.2 Elimination des ETAS combinatoires

Cette élimination est réalisée de la manière suivante: la table des caractéristiques associée au bloc fonctionnel à réaliser sous forme de PLA est balayée séquentiellement. Pour chaque ETA combinatoire rencontré, le remplacement de son étiquette par sa fonction d'activation dans la fonction d'activation de chacun de ses successeurs est effectué. La nouvelle description obtenue ne comporte plus que des ETAs séquentiels. Cette étape d'élimination des ETAs combinatoires est illustrée en Annexe 1.

2.3.6.3 Transformation des commandes d'activation

Il serait possible de resegmenter la description DELTA en cours de transformation en regroupant les ETAs possédant la même commande d'activation et de générer ainsi des PLAs pour chacun de ces sous groupes.

Cette nouvelle segmentation nous est apparue inutile et trop extrême. En effet la création de trop nombreuses entités générées de manière séparée accroît de manière considérable les problèmes d'interconnexions et de placement relatif de ces blocs. Nous avons en définitive opté pour une synthèse de PLA unique par couple de phases éléméntaires rencontré dans la description.

Lors du balayage de la table des caractéristiques, une table des commandes rencontrées est générée. Un second balayage permet de transformer les ETAs activables sur ces commandes: la partie commande de l'instruction est remplacée par la phase élémentaire sur laquelle cette commande est activée et la fonction d'activation de cette commande est intégrée dans la fonction d'activation de chacun des ETAs que cette commande contrôle.

2.3.6.4 Vérification de l'unicité des sorties du PLA

Avant d'entreprendre la phase de synthèse du PLA, le programme de génération doit s'assurer que les fonctions d'activation que l'on désire regrouper s'excluent mutuellement, c'est-à-dire qu'il n'est pas possible que deux ETAs distincts puissent être activés simultanément par l'évaluation d'une fonction donnée.

2.3.6.5 Génération de fonctions d'activation intégrables

Une fois que toutes ces transformations effectuées sur la description DELTA ont été rangées dans la table des caractéristiques, des modifications de forme de chaque fonction d'activation doivent être effectuées de façon à produire des fonctions se présentant comme une somme de produits (ou un produit de sommes).

Cette organisation permet de mettre en évidence les monômes du PLA (produits intermédiaires générés en sortie de la première matrice).

2.3.6.6 Choix d'un codage

La dernière étape avant la génération des matrices de bits consiste à coder les entrées du PLA (ou le nom des différents ETAs constituant cette description).

Le choix du codage est lié à la fonctionnalité que le concepteur veut implanter sous forme de PLA. S'il désire réaliser un PLA gérant le séquencement de son micro-programme, il se peut qu'il choisisse un système de pagination. Par contre, si son but est de produire un PLA de génération rapide des commandes, il peut choisir un codage minimisant le nombre de '1' utilisés dans ce codage. (Cette minimisation de bits positionnés à '1' minimise le nombre de transistors implantés dans le PLA et contribue à diminuer sa taille et donc à augmenter sa vitesse).

Différents algorithmes de codage des ETAs sont mis à la disposition du concepteur. Un paramètre permet de sélectionner le type de codage désiré lors de l'appel du programme de synthèse.

2.3.6.7 Prototype

Un prototype a été développé à l'IMAG. Il exécute les différentes étapes citées précédemment. Seul le codage par pagination a été implanté. Un détail des organigrammes utilisés est proposé en Annexe 1.

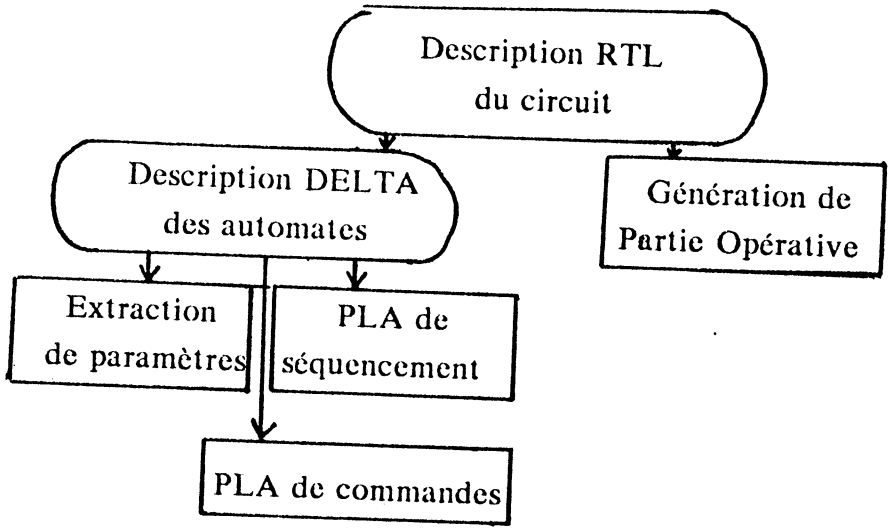
Une description Delta du microprocesseur M6800 de Motorola a été réalisée [NEM81]. Malheureusement, la génération automatique de sa partie contrôle n'a pas été entreprise. Plusieurs raisons à cela dont la redistribution des priorités des projets au sein de l'équipe du Professeur Anceau, mon départ en stage INRIA aux Etats-Unis dont le sujet était "L'étude des systèmes d'aide à la conception assistée (par ordinateur) de circuits intégrés".

Cette redistribution des priorités de projets a été entreprise de manière à aborder le problème de la génération automatique de parties contrôles à partir d'un niveau de description plus élevé.

En effet, il est peu intéressant de générer automatiquement des PLAs à partir de descriptions logiques élémentaires, trop longues à obtenir manuellement.

L'utilisation de langages de description de type RTL comme KARL ou IRENE (IMAG) comme langage source pour le système de génération automatique semble plus adéquat.

Le formalisme Delta trouve sa place dans ce système en tant que représentation intermédiaire générée directement (par programme) à partir d'une description RTL comme l'indique le schéma suivant.


Description RTL
du circuit
Description DELTA
des automates
Génération de
Partie Opérative
Extraction
de paramètres
PLA de
séquencement
PLA de commandes

3. SYSTEMES INTEGRES D'AIDE A LA CONCEPTION DE CIRCUITS A HAUTE INTEGRATION

3.1 INTRODUCTION

La densité des circuits intégrés digitaux double tous les deux ans environ. Elle se traduit physiquement par un accroissement du nombre de transistors intégrés sur silicium à surface d'implantation quasi-constante. La complexité des fonctions exécutées par ces circuits est, par là-même, fortement accrue. Ces résultats proviennent essentiellement:

1. des progrès dans les procédés de fabrication permettant, entre autres, une précision plus grande des techniques d'implantation, une diminution de la taille des transistors intégrés sur silicium, une augmentation des vitesses de transfert;

2. du développement d'un environnement logiciel pour l'automatisation de tâches autrefois manuelles (dessin, vérification).

L'environnement logiciel prend, à l'heure actuelle, une place de plus en plus importante pour qui veut concevoir et fabriquer des circuits intégrés. Il permet de diminuer à la fois le temps et le coût de conception, de manipuler de grandes quantités d'information et ainsi de contrôler la complexité des circuits conçus.

Le nombre de systèmes d'aide à la conception de circuits logiques a augmenté de manière considérable ces deux dernières années. De nombreux centres de recherche ou de développement de circuits intégrés ont mis au point leur propre environnement logiciel. Bien qu'ils soient développés indépendamment les uns des autres, ces systèmes suivent tous une méthodologie de conception semblable. Celle-ci peut être résumée en deux mots: RÉPÉTITION et HIÉRARCHIE.

Ces deux notions sont exploitées autant durant les phases de descriptions du circuit que dans les phases de vérification et de simulation des descriptions obtenues.

La notion de HIÉRARCHIE (ou STRUCTURE) a donné lieu à des représentations arborescentes de structures de données qui semblent être adéquates pour ce type de problèmes. Cette approche hiérarchique permet au concepteur d'aborder de manière progressive et modulaire la description de circuits logiques et d'appréhender des complexités de plus en plus importantes grâce à une segmentation fonctionnelle.

La notion de RÉPÉTITION a permis la génération de représentations géométriques d'éléments fonctionnels, structurées et régulières. Ces représentations sont de nature répétitive et sont générées par programmes. Ceci contribue à diminuer de manière considérable le temps de conception et les possibilités d'erreurs.

Répétition et Hiérarchie permettent aussi de diminuer le temps et le coût des phases de vérifications et simulations de descriptions: la structure répétitive des circuits permet d'effectuer ces vérifications sur le modèle de référence à partir duquel ces duplications sont réalisées; leur structure hiérarchique permet des vérifications et simulations aux différents niveaux fonctionnels et structurels dégagés. Ces étapes progressives de vérification donnent au concepteur le moyen de contrôler le fonctionnement du circuit qu'il conçoit avant de le fixer sur silicium.

Quel que soit l'environnement logiciel choisi -conception procédurale ou interactive, utilisation de systèmes centralisés ou stations autonomes de conception (réseaux de micro-ordinateurs travaillant sur la même banque de données centrale mais disposant de leurs propres logiciels d'aide à la conception)-, les outils d'aide au développement et à la génération utilisés reposent plus ou moins extensivement sur ces deux notions.

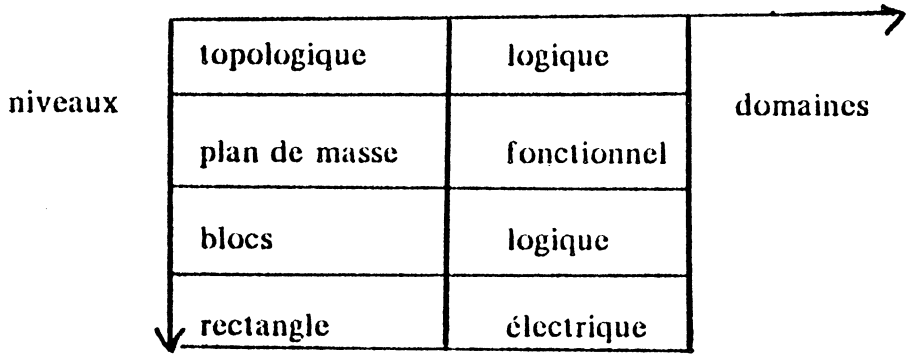
Notre but, dans ce chapitre, consiste à présenter les différents éléments, logiciels et matériels constituant un système intégré d'aide à la conception de circuits logiques à très haute intégration (VLSI). Nous détaillerons l'enchaînement logique entre les diverses étapes permettant la génération des masques du circuit (fichiers utilisés par les fabricants, qui décrivent la manière dont la fabrication doit être réalisée) ainsi que les orientations prises aujourd'hui en termes de support informatique et interface utilisateur.

3.2 MÉTHODOLOGIE DE CONCEPTION

La nécessité d'appréhender des problèmes de plus en plus complexes et de construire sur silicium des algorithmes permettant de les résoudre a suscité le développement de méthodologies de conception et d'un ensemble de logiciels concourant à la production des descriptions géométriques optimales de masques.

Comme nous l'avons déjà mentionné au cours du premier chapitre, trois différents domaines de représentations sont couramment utilisés lors de la conception de circuits intégrés, à savoir: fonctionnel, structurel et physique [CLE79].

Chacun de ces domaines peut être à son tour décomposé en différents niveaux organisés de manière hiérarchique comme l'illustre le schéma suivant:



Toutefois, il est souvent nécessaire de mener en parallèle des descriptions et évaluations relevant de domaines différents. De même qu'une description de plan de masse d'un circuit nécessite sa description au préalable en termes de transfert de registres, l'adéquation d'une configuration topologique donnée par un plan de masse est validée par un comportement logique et électrique acceptable des blocs fonctionnels décrits.

La démarche utilisée par le concepteur est rarement linéaire. De nombreuses interactions entre domaines et entre niveaux d'un même domaine sont nécessaires pour obtenir une description satisfaisante des masques du circuit.

Donc, à chacun de ces niveaux est associé un langage de description dont les composants sémantiques reflètent la spécificité de ce niveau. Le niveau de description graphique, par exemple, manipule des rectangles, des couleurs associées à chaque niveau de masque, des opérations telles que des rotations, symétries, translations, duplications, occurrences. Le niveau logique, par contre, manipule des entités fonctionnelles telles que le transistor, inverseur, point mémoire ainsi que des opérations appropriées: connexion, chargement de variables par exemple.

Le passage d'une représentation à la suivante (du niveau comportemental au niveau logique synchrone, ou bien du niveau logique au niveau graphique par exemple) est réalisé par interprétations successives des composants sémantiques et opérations associés à chacun de ces niveaux.

De plus, à chacun de ces niveaux, des vérifications et simulations des descriptions obtenues permettent des corrections et modifications avant qu'une nouvelle représentation, plus détaillée, ne soit entreprise.

Le schéma suivant reflète les relations entre ces niveaux ainsi que la manière dont ils sont constitués.

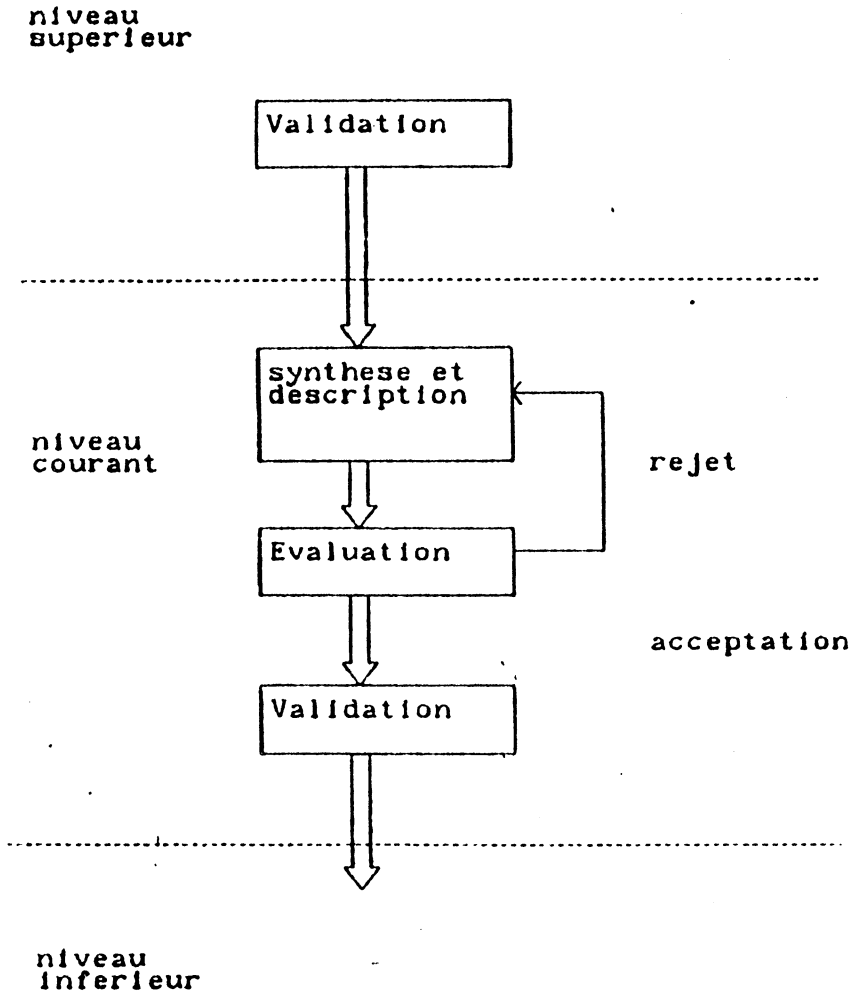


Figure 3.1: Représentation des outils de descriptions
mis en oeuvre à chaque niveau de conception

Chacun de ces niveaux est caractérisé par un ensemble de contraintes (taille des transistors, vitesse minimale à respecter, surface maximale disponible sur silicium) qui doivent être prises en compte successivement dans chacune des représentations de manière à produire, en dernier ressort, une représentation géométrique de l'algorithme à implanter.

Le fait de concevoir l'interprétation successive de ces descriptions comme la prise en compte de séries de contraintes caractéristiques à chacun des niveaux considérés est important. L'existence de ces descriptions permet le développement d'outils informatiques qui génèrent automatique-

ment de nouvelles descriptions. Elles peuvent être à leur tour vérifiées, simulées, optimisées par des programmes comprenant les structures sémantiques utilisées.

On peut dégager trois différents types de logiciels nécessaires pour assurer une conception correcte de circuits intégrés:

1. les outils de génération de descriptions, se composant de langages, interpréteurs, compilateurs;

2. les outils d'optimisation, de placement, d'interconnexion entre structures d'implantation (qui correspondent aux interfaces existant entre processus coopérants), de simulation, de vérification, etc..;

3. un système de gestion des données relatives au circuit. L'accès à ces dernières, leur modification, ajout ou suppression doivent être gérés et contrôlés de manière à assurer leur homogénéité et cohérence.

La figure suivante propose une structure globale type de systèmes d'aide à la conception. Les différents outils logiciels cités correspondent aux actions qui doivent être entreprises sur les diverses descriptions pour produire une description correcte des masques du circuit à concevoir.

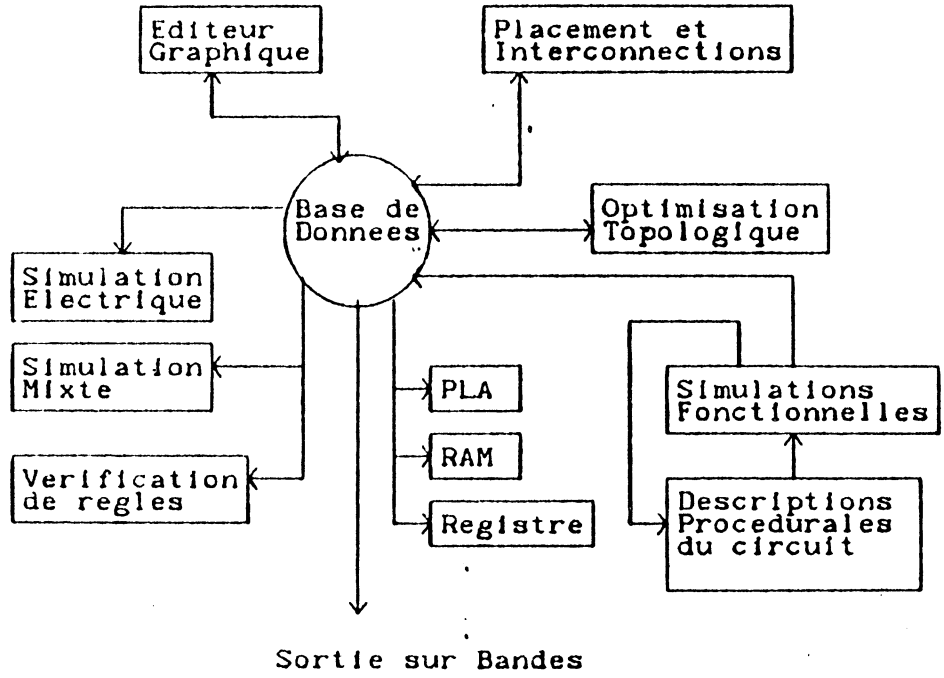


Figure 3.2: Structure Globale d'un Système Intégré pour la Conception de VLSI

3.3 LANGAGES DE DESCRIPTION

L'utilisation de langages pour décrire le comportement d'un système, ses relations avec son environnement, la manière dont il est réalisé est le support de base des différentes étapes de la conception de circuits intégrés.

L'importance de ces langages de description est liée au fait que, pour comprendre la structure d'un objet ou la fonction qu'il réalise, ce dernier doit être décrit selon un ensemble de règles déterministes. Le respect de ces règles -syntaxiques et sémantiques- est vérifié à chaque phase de description.

Tout système peut être représenté par la fonction qu'il réalise; les algorithmes décrivant son comportement sont interprétés successivement, d'une représentation comportementale de haut niveau jusqu'à une représentation physique (réalisée sur silicium).

Ces étapes de descriptions formelles sont aujourd'hui d'autant plus importantes que le processus d'automatisation de leur interprétation est d'utilisation courante.

Comme il a été indiqué dans le premier chapitre, trois types de descriptions sont nécessaires pour assurer une réalisation physique d'un algorithme: elles sont de nature comportementale, structurelle et physique.

Au cours de ce dernier lustre, un nombre considérable de langages permettant de telles descriptions ont été développés. Dans ce paragraphe, nous présentons les principaux langages utilisés aujourd'hui durant les différentes phases de la conception de circuits intégrés et décrivons la manière dont ils sont utilisés par le concepteur.

3.3.1 Langages de Description de Comportement

La description du comportement de l'algorithme à implanter est la première étape que le concepteur doit envisager.

L'utilisation de langages de programmation classiques tels Pascal ou PL/1 envisagée initialement pour la description du comportement de circuits intégrés s'est très rapidement avérée inadéquate. Ces langages sont par nature trop généraux pour les types de problèmes qui doivent être résolus ici et les étapes de compilation ou d'interprétation trop lentes.

Dans le cas présent, l'objectif à atteindre est l'implantation physique d'algorithmes respectant des contraintes temporelles (vitesse) et spatiales (faible surface, grande densité). Il a donc été important, pour des raisons de performance, de développer des langages - ou sous-langages de langages de programmation existants - prenant en compte ces contraintes, permettant ainsi de produire des descriptions claires et facilement simulables.

3.3.1.1 Descriptions procédurales

Des langages procéduraux de haut niveau tels que DPL (Design Procedural Language) développé au M.I.T., VIP développé à l'université de Stanford, ou HELL (INRIA) sont inspirés de LISP. LISP est un langage intéressant pour la description d'objets ou de fonctions et semble plus adapté pour ce type de problèmes que les langages de programmation classiques. Des descriptions modulaires d'entités fonctionnellement autonomes sont facilement réalisables laissant le concepteur libre de ses choix d'implantation. Elles sont, de plus, facilement interprétables.

L'approche utilisée consiste à décrire de manière globale le circuit à réaliser. Aucune contrainte structurelle n'est prise en compte. Néanmoins, une ébauche de structure est implicitement présente par l'existence des blocs procéduraux utilisés pour décrire ce circuit.

Cette approche comportementale, empreinte de considérations structurelles, a marqué les langages développés en Europe. Les Réseaux de Pétri, Graphcet ou Cap permettent de décrire des algorithmes et n'induisent aucune suggestion ou considération structurelles. Ils sont tous deux couramment utilisés mais leur support linguistique est trop faible pour permettre des interprétations jusqu'au niveau physique.

3.3.1.2 Paramétrisation

La possibilité de générer des descriptions paramétrées (similaires aux procédures avec paramètres des langages de programmation) est très intéressante car elle permet de mettre en relief des structures type traduisant une fonctionnalité donnée. Elle permet à la fois de modulariser et de réduire la taille des descriptions produites, souvent très importantes au niveau logique. De plus, le stockage de structures paramétrées, fonctions de production de structures physiques, permet de minimiser la quantité d'information mémorisée.

3.3.2 Langages de Description de Structures - Compilateurs de Silicium

La description des structures d'implantation des algorithmes réalisant les fonctions du circuit à intégrer peut être entreprise, elle aussi, à différents niveaux de hiérarchie. Les descriptions de matériel peuvent refléter la structure du circuit à un niveau architectural, à un niveau logique (structuré ou non).

Les descriptions architecturales de haut niveau sont très souvent implicites lorsque des langages tels que DPL sont utilisés. Au niveau plus fin de description de circuits logiques structurés, de nombreux langages ont été développés.

Ces langages, communément appelés Langages de Transfert de Registres, ont été développés dans le but d'associer à chaque type sémantique du langage une structure implantable physiquement qui ait une fonction bien définie. Par exemple, aux types REGISTRE, UAL, MULTIPLEXEUR ou PLA sont associées des fonctions logiques typiques. Cette correspondance permet, moyennant des choix souvent très restrictifs, une génération automatique de ces structures.

Ils possèdent de manière générale une syntaxe permettant de décrire des séquences de contrôle s'exécutant séquentiellement ou en parallèle. Celles-ci subissent ensuite une série de transformations avant d'être prêtes à être réalisées physiquement.

L'intérêt de ces langages réside dans le fait qu'ils permettent une description globale du circuit à un niveau de représentation plus élémentaire. Un circuit est décrit en termes de registres, mémoires, séquenceurs et la manière dont ces éléments sont activés dans le temps est précisée, pour assurer l'exécution correcte de l'algorithme que l'on désire implanter physiquement. La notion du temps est introduite de manière plus explicite grâce à la définition d'une horloge de base, de notions de synchronisation, etc..

3.3.2.1 Compilateurs de Silicium

Le passage d'une description structurelle d'un circuit en termes d'éléments de logique à sa représentation physique en vue de sa fabrication peut être entreprise de différentes manières et nécessiter des étapes supplémentaires de décomposition.

Dans le but de diminuer le temps nécessaire à la conception de tels circuits, des outils de génération automatique de descriptions physiques ont été développés. Ce passage automatique, d'une description logique en une représentation physique est, entre autres réalisée aujourd'hui par les 'Compilateurs de Silicium' [AYR79].

Ces compilateurs produisent des descriptions complètes de masques, tout comme les compilateurs de langages de programmation produisent des modules exécutables en mémoire centrale.

Comme cela a été expliqué clairement dans [MEA79], l'étape de compilation se compose des tâches suivantes:

1. extraire de la description globale du circuit (en général décrit dans un langage de Transfert de Registres) la partie opérative et la partie contrôle;

2. réaliser des actions appropriées pour chacune de ces deux descriptions: rechercher en bibliothèque la description standard des éléments servant à la construction de chacune des structures constituant la partie opérative, interpréter et regrouper les séquences d'actions contrôlant cette partie opérative de manière à les implanter automatiquement (sous forme de PLAs par exemple).

Différentes passes sont effectuées pour extraire toutes les informations nécessaires à cette génération. La description physique obtenue doit être ensuite optimisée, vérifiée.

Dans de nombreuses applications, une telle compilation est une étape trop coûteuse en temps de calcul et trop peu interactive. Il est vrai que la liberté de décision et de choix de méthodes d'implantation dont disposait le concepteur lorsque les circuits étaient conçus manuellement est presque inexistante ici.

L'utilisation de 'Compilateurs de Silicium' a été très discutée au cours de ces dernières années. Le développement de systèmes plus interactifs d'aide à la conception, revalorisant la place du concepteur, a quelque peu ébranlé l'image de marque des compilateurs. Leur utilisation est toutefois peu répandue. Dans les organismes réalisant à la fois la conception et la fabrication de circuits, où le facteur rentabilité ne peut pas être négligé, ils regroupent en fait des outils très disparates et ne peuvent être considérés comme compilateurs au sens propre du terme. Par contre, pour la fabrication de circuits à la demande, des compilateurs de silicium inspirés de Mac Pitts [SIS82] semblent être prometteurs.

3.3.3 Langages de Description de Masques

3.3.3.1 CIF

Il existe aujourd'hui un langage standard de description de géométries pour la fabrication des masques de circuits intégrés appelé CIF (Caltech Intermediate Format). Conçu à Caltech (California Institute of Technology) en 1978, il a subi plusieurs transformations avant d'être utilisé comme format standard par les compagnies de fabrication de circuits.

Les constituants de base de ce langage sont des boîtes, rectangulaires ou polygonales, qui peuvent être organisées hiérarchiquement pour décrire des entités plus complexes. Seules sont manipulées des informations graphiques -dimensions et positionnement de ces boîtes- ainsi que le masque sur lequel chacune de ces boîtes doit être dessinée.

La macrogénération de source CIF à partir d'une description de plus haut niveau est l'étape finale du travail de conception. Elle est aujourd'hui

automatique. (Il est en effet difficilement concevable de produire des descriptions de quelques millions de rectangles manuellement).

3.3.3.2 Langages pour une Description Hiérarchique de Masques

Depuis CIF, de nombreux langages de description de géométries ont été développés. Ils s'en distinguent par le fait qu'ils utilisent des concepts linguistiques plus complexes [SAS81]. La notion de boîte utilisée par CIF est remplacée par des objets plus complexes tels que des transistors, inverseurs, points mémoires, etc...

L'apparition d'une hiérarchie au niveau de description physique des masques permet de diminuer par conséquent la taille des descriptions et de les rendre plus compréhensibles.

Ces langages de description de masques sont souvent des sous-ensembles de langages de programmation. STIF et SLL (développés à l'Université de Berkeley) [ELL81] sont issus du langage C. Une interprétation directe de ces descriptions de topologies en CIF (pour STIF et SLL) permet de générer automatiquement une description des masques. La description d'un inverseur en CIF et en STIF est proposée en Annexe 2.

A l'IMAG (Grenoble), un langage de description de masques pour l'enseignement, LUCIE, a été développé. Il est basé sur les mêmes concepts que CIF. Un éditeur graphique lui est associé pour lui permettre saisie et modifications interactives de descriptions.

Les étapes de simulation et vérification de la description topologique sont alors entreprises, non plus à partir de descriptions en CIF mais plutôt à partir de ces descriptions hiérarchiques sur base desquelles des extracteurs de schémas électriques, vérificateurs de règles [ARN82] peuvent être développés en utilisant ces notions de hiérarchies pour optimiser le temps nécessaire à ces vérifications.

3.3.4 Editeurs Graphiques

L'utilisation de systèmes graphiques interactifs s'est facilement étendue depuis l'apparition sur le marché des composants de mémoires à haute densité (de 64K et 128K octets) et de microprocesseurs tels que le Z8000 ou le M68000.

La diminution de leur coût a permis de réaliser directement en matériel des fonctions autrefois exécutées par programmes comme les paginations, le zoom, etc...

De plus, un transfert de logiciels s'est opéré au niveau des terminaux graphiques, transformant ces derniers en stations de travail quasi-autonomes.

Les nombreuses possibilités d'interaction offertes par ces systèmes a provoqué le développement de nombreux éditeurs graphiques, remplaçant l'utilisation classique des langages de description de géométries.

En fait, chaque instruction de ces langages a été transformée en une commande insérée au niveau de menus ou traduite directement sous forme de schéma.

3.3.4.1 Editeurs pour Descriptions Géométriques

Des éditeurs graphiques spécialisés ont été développés pour la conception de circuits intégrés. A l'université de Berkeley par exemple, deux éditeurs graphiques sont couramment utilisés par les étudiants: CAESAR, permettant la génération de géométries orthogonales [OUS82] et KIC utilisé pour la conception d'une variété plus grande de circuits électroniques [KEL82].

CAESAR se comporte comme un générateur de surfaces rectangulaires que le concepteur peut peindre, déplacer, dupliquer, couper, assembler en structures plus complexes et se limite à la génération de masques pour la technologie NMOS, la plus couramment utilisée aujourd'hui.

KIC, par contre, manipule des géométries de base plus complexes (polygones, cercles, frontières) et permet de produire des descriptions topologiques de masques pour diverses technologies: NMOS, CMOS, TTL, ICL, etc..

Il exite un très grand nombre d'éditeurs pour décrire des géométries; presque chaque centre concevant des circuits intégrés en a développé un: IGS à IBM (Interactive Graphics System), [CAR80], LUCIE à l'IMAG (Grenoble), CALMA, APPLICON (éditeurs commerciaux), etc.

3.3.4.2 Editeurs pour Descriptions Logiques

Une fois cette orientation prise, l'étape suivante consiste à concevoir des éditeurs graphiques pour la description de circuits logiques.

Ceux-ci, en fait, ont existé bien avant les éditeurs de descriptions de géométries comme le système BOLD chez IBM, développé il y a une dizaine d'années maintenant. Le problème actuel consiste à faire le lien entre les outils de descriptions logiques et topologiques pour les faire travailler de manière complémentaire et non pas indépendante. A cette fin des descriptions symboliques sont utilisées.

Une première approche consiste à développer des représentations graphiques simplifiées de certaines fonctions logiques couramment utilisées. Ces représentations réalisent une sorte de 'pont' entre des descriptions purement logiques (où aucune indication physique n'est fournie) et des descriptions topologiques (trop détaillées et donc source de nombreuses erreurs). Elles fournissent de plus des détails sur le comportement électrique des descriptions réalisées comme les formalismes MDMOS ou CRASH, tous deux développés au CNET à Grenoble.

Le premier formalisme développé dans ce sens a été réalisé à Caltech. Il permet de produire des diagrammes appelés diagrammes STICK, représentations graphiques simplifiées de structures d'implantation [PET82]. Les rectangles sont représentés par des lignes de différentes couleurs, des transistors par croisement de certaines lignes. A titre

d'exemple, la figure suivante propose une triple représentation d'un inverseur: logique, en sticks et étendue (à base de rectangles).

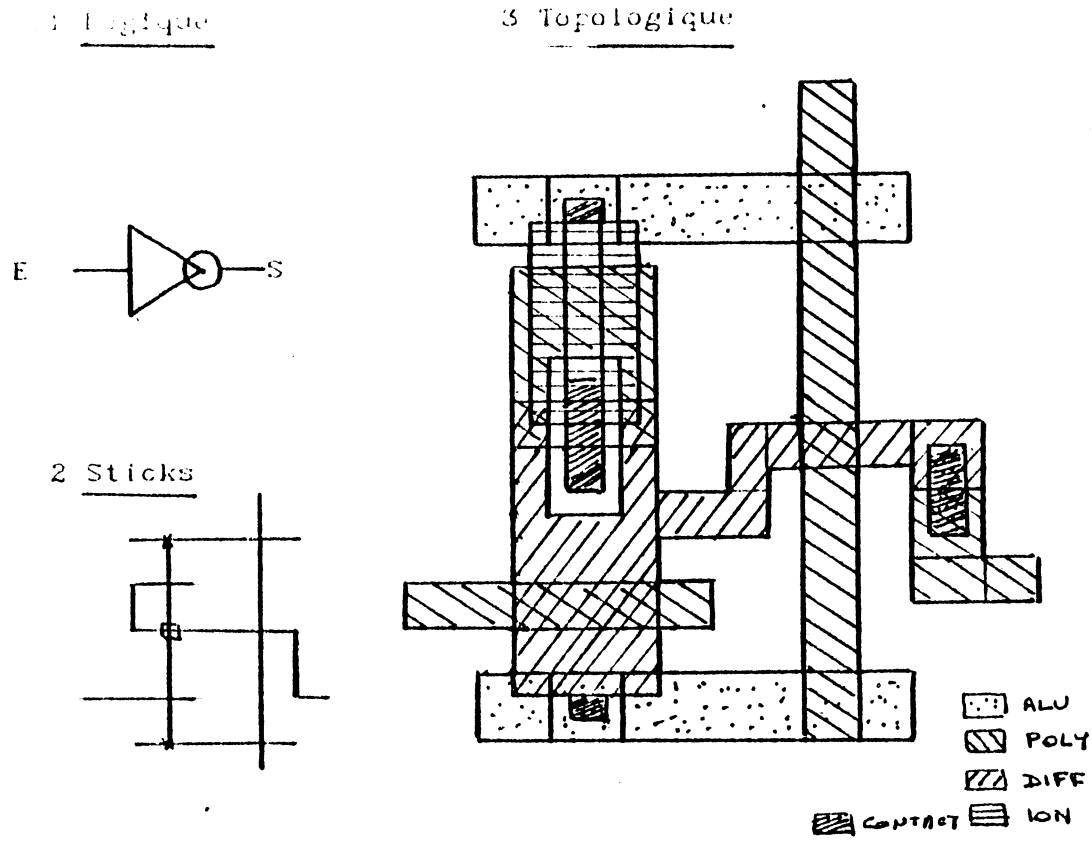


Figure 3.3: Représentations graphiques d'un inverseur

Ce formalisme permet au concepteur de ne pas se perdre dans les représentations détaillées géométriquement lorsqu'il est en phase d'assemblage et de se consacrer à leur enchaînement logique.

L'interprétation d'une description STICKS en une description étendue pour la production des masques est réalisée automatiquement par transformation des structures des données associées à ces descriptions. Celles-ci sont rangées soit en mémoire, soit sur fichiers, soit dans une base de données. Les renseignements électriques associés permettent à ce point d'effectuer des simulations électriques partielles.

De nombreux autres formalismes similaires existent aujourd'hui. Le laboratoire de recherche de Xerox Park, entre autres, a mis au point un système de conception graphique interprétant des schémas graphiques réalisés en utilisant le formalisme STICKS directement en CIF [PET82].

3.4 OUTILS ALGORITHMIQUES SPÉCIALISÉS

Outre les outils d'aide à la génération automatique de masques de circuits intégrés, une seconde classe d'outils a été développée en vue d'optimiser et de vérifier les descriptions obtenues après la phase de génération.

Les structures générées sont, par construction, répétitives (registres, mémoires dynamiques..) et occupent souvent une surface importante. A titre d'exemple, la partie opérative du RISC (Reduced Instruction Set Computer) unité de calcul 32 bits conçue à l'université de Berkeley, occupe 80% de la surface totale du circuit en structures régulières [JEN82]. Il en est de même pour la génération de parties contrôles, maintenant réalisées sous forme de PLAs [WES80].

3.4.1 Programmes d'optimisation topologiques

L'optimisation topologique est une étape nécessaire pour maintenir la surface du circuit dans des limites raisonnables. La plupart des programmes qui ont été développés dans ce sens compressent le circuit tout en maintenant les distances minimales imposées par les règles technologiques. Leur application sur des portions du circuit en cours de conception permet un assemblage de blocs optimisés. CABBAGE, développé à Berkeley opère dans ce sens [PED81], ainsi que TRICKY dans la chaîne DELPHINE à EFCIS.

3.4.2 Programmes de placement et d'interconnexion

Placement et interconnexions, bien qu'étroitement dépendant l'un de l'autre, sont des problèmes qui peuvent être abordés séparément.

En effet, le placement de blocs fonctionnels, ou "réalisation d'un plan de masse" peut être entrepris avant que le problème d'interconnexion ne soit examiné; il est de plus relativement indépendant de la technologie utilisée (seules les valeurs des pas de poly et pas de métal sont significatives pour l'évaluation des surfaces occupées par ces blocs fonctionnels [REI82]).

Les problèmes d'interconnexion, par contre, sont de nature plus complexe. Outre la minimisation du nombre et de la longueur des interconnexions entre blocs, ils dépendent plus directement de la technologie (mono ou multi couche de métal, positionnement des nappes de fils,...).

Bien que ces deux types de problèmes se résolvent par approximations successives en consommant énormément de temps calcul, les algorithmes d'interconnexion convergent en général beaucoup plus lentement que ceux de placement.

Des techniques spéciales sont utilisées pour accélérer le choix d'une solution. Toutefois les algorithmes développés sont assez largement dépendant de la technologie utilisée et peu transportables car ils dépendent fortement du système d'exploitation local.

3.4.3 Programmes de Vérification et de Simulation

Une fois obtenue une description physique globale du circuit, de nombreuses vérifications doivent être accomplies avant de confier la description obtenue aux fabricants. Il est important de vérifier:

1. que les règles de dessin, liées à la technologie utilisée, sont respectées. La phase de vérification de règles a jusqu'ici été entreprise sur des descriptions physiques globales; c'est-à-dire les vérificateurs de règles de dessin (DRC) interprètent des millions de données (qui correspondent aux positions relatives de tous les rectangles et cela sur tous les niveaux de masques). Les vérificateurs de règles conçus aujourd'hui tirent parti de la structure hiérarchique du circuit pour diminuer le temps de calcul nécessaire à cette étape. Une vérification progressive du respect des règles de dessin est entreprise à chaque génération d'une description géométrique d'un bloc fonctionnel; le temps nécessaire pour la vérification totale du circuit décroît alors de manière exponentielle. A l'Université de Berkeley, par exemple, un programme de vérification de règles topologiques pour la technologie NMOS a été développé: LYRA [ARN82]. Il teste localement les 'coins' des structures géométriques et progresse de manière 'excentrique'. Une nouvelle version de LYRA exploite la struture hiérarchique du circuit en cours de test.

2. que le circuit produit a un comportement électrique correct. Pour cela, une étape d'extraction de schémas logiques doit être réalisée à partir de la description géométrique. Ici encore, les mêmes problèmes de temps de calcul se posent et des techniques semblables sont utilisées pour rendre ces outils plus efficaces: partitionnement du circuit, exploitation de la hiérarchie [NEW82]. Cette phase d'extraction est très importante car elle permet ensuite de vérifier que chaque instruction que le circuit exécute respecte le timing défini à la phase de conception logique (vérification du timing au sein de chaque entité fonctionnelle, du retard lié à la propagation des signaux dans les bus entre blocs...).

3.5 UTILISATION DE BASES DE DONNÉES

Tout système informatique interprétant des grandes quantités d'information doit se munir d'un système de gestion de ces données de manière à assurer le contrôle de leur stockage et leur manipulation.

3.5.1 Banque de Données pour Systèmes Intégrés d'Aide à la Conception

Dans le domaine de la conception de circuits à haute intégration et jusqu'avant l'ère du VLSI, cette question était résolue par l'utilisation d'ensembles de fichiers gérés par le système d'exploitation local.

En effet, jusqu'à récemment, la principale préoccupation des informaticiens était la mise au point d'outils de synthèse et de génération. Ces outils interprétaient des ensembles de données spécifiques à un niveau de conception bien défini (algorithmique, logique, structurel).

A l'heure actuelle où la plupart des étapes de conception sont assistées par ordinateur, un souci d'intégrité et d'homogénéité des données entre ces différentes étapes apparaît. Ceci s'explique pour plusieurs raisons.

Tout d'abord, parce que les étapes de conception d'un circuit ne s'enchaînent pas de manière séquentielle: de nombreuses modifications de descriptions doivent être faites a posteriori. Cela signifie par exemple que la modification d'une description logique d'un algorithme doit se répercuter sur sa description comportementale. Les deux ensembles de données (ou plus, éventuellement) doivent être mis à jour automatiquement.

D'autre part, la généralisation de l'automatisation de la conception conjointement à l'accroissement des densités et complexités implantées sur silicium a contribué à augmenter de manière considérable les quantités d'information à manipuler. Bien que l'automatisation des générations de descriptions permette de diminuer grandement les risques d'erreurs, l'augmentation des complexités rend beaucoup plus difficile la localisation d'éventuelles erreurs. De nouveaux besoins surgissent, comme par exem-

ple, une plus grande grande sécurité des données ou encore la règlementation de leur accès. Ce dernier facteur est important dans la mesure où plusieurs concepteurs sont amenés à travailler en même temps sur différentes descriptions d'un même circuit.

3.5.2 Avantages et Inconvénients de l'Utilisation d'un Système de Gestion de Données

De nombreux systèmes de gestion de banques de données ont déjà été développés, pour des applications spécialisées: banques, chemins de fer, etc... L'utilisation de tels systèmes dans le cadre de la conception de circuits intégrés a été, et est toujours, très discutée.

Jusqu'à présent, les solutions trouvées consistaient à utiliser des systèmes de gestion de fichiers contenant les données relatives aux circuits.

Avec l'utilisation de LISP, une tendance aujourd'hui se dégage à stocker non plus les données relatives aux circuits mais plutôt les programmes permettant la génération de ces données.

C'est une alternative très séduisante aux systèmes de bases de données puisque les données sont accédées par interprétation de ces programmes LISP.

Des recherches dans cette voie sont effectuées en France à l'INRIA, ainsi que dans des universités américaines comme au MIT, par exemple.

Toutefois, dans le cadre du stage que j'ai effectué au laboratoire de recherche d'IBM à San Jose, Californie, nous nous sommes intéressés au problème de l'utilisation des systèmes de bases de données relationnelles pour la conception de circuits intégrés.

Les inconvénients que présentaient les premiers systèmes de gestion de banques de données étaient bien supérieurs aux avantages qu'ils proposaient: lenteur des accès, des modifications alors que le but fixé était de diminuer le temps et le coût de la conception. Aujourd'hui le problème se

pose un peu différemment. Le temps nécessaire pour concevoir un circuit ne dépend plus directement de la présence ou de l'absence de tel ou tel outil mais plutôt de sa souplesse d'utilisation. Plusieurs concepteurs doivent pouvoir travailler simultanément sur les mêmes descriptions. Bien évidemment l'introduction d'une interface supplémentaire entre le concepteur et les données sur lesquelles il travaille ralentit considérablement son travail, mais ce compromis semble être inévitable vu les possibilités d'intégration actuelles.

3.5.3 Choix d'un Système de Gestion de Données

La question suivante concerne le choix du type de système de gestion de données à utiliser. Bien sûr, une solution de facilité est de mettre sur pied un système de gestion de données supporté par le système d'exploitation sur lequel l'ensemble des outils d'aide à la conception est implanté. C'est en effet cette solution que l'on rencontre le plus couramment aujourd'hui dans les centres de recherche et de développement aux USA (à Stanford avec le système SCALD, à Berkeley: KIC2, CIF ainsi qu'à IBM), ou en France au CNET avec le système CASSIOPEE. Cette solution tend à multiplier le nombre de ces systèmes sans pour autant en assurer leur transportabilité.

De nouvelles recherches sont effectuées à l'heure actuelle, concernant l'adaptation et l'utilisation de systèmes de gestion de bases de données relationnelles, tels que System R, pour la conception de VLSI. L'applicabilité de ces systèmes a été étudiée dans de nombreuses universités [KAT82], [GUT82], [BEE82]. De nouvelles méthodes pour la représentation de données structurées et hiérarchisées sont développées dans ce sens. Nous reviendrons sur ce point dans le chapitre suivant.

En conclusion, on peut dire que l'utilisation de systèmes de gestion de banques de données pour la conception de circuits VLSI rencontre encore aujourd'hui de grandes résistances. Toutefois, dans la mesure où l'ensemble des données manipulées pendant la phase de conception est le noyau de tout système intégré d'aide à la conception, il est à prévoir d'ici à quelques années des modifications importantes dans les systèmes de gestion de

données de manière à garantir leur transportabilité et à accroître leur efficacité.

3.6 DES SYSTÈMES CENTRALISÉS AUX STATIONS AUTONOMES

L'évolution de la technologie au cours de ces dernières années a bouleversé l'environnement de travail des concepteurs de circuits logiques. L'apparition sur le marché des composants électroniques de microprosseurs de mémoires de grande capacité, à des prix réduits, a permis de décentraliser un nombre important de tâches vers les organes périphériques tels que des logiciels graphiques ou éditeurs de textes.

Dans le cadre de la conception de circuits VLSI, l'environnement logiciel et matériel en place a lui aussi évolué très rapidement vers une grande décentralisation des logiciels. A l'exception des universités qui conservent généralement un environnement logiciel relativement centralisé à cause de leur support financier limité, la plupart des centres de conception et de fabrication disposent, à l'heure actuelle, de stations de travail très performantes.

3.6.1 Organisation Matérielle Globale

Dans la mesure où un micro-ordinateur n'est pas beaucoup plus cher aujourd'hui qu'un simple terminal graphique d'il y a seulement trois ou quatre ans, la plupart des centres de recherche se sont équipés de réseaux de micro-ordinateurs. Ceux-ci servent de stations de travail à partir desquelles le concepteur peut guider les différentes étapes de la conception.

Une unité centrale de calcul est chargée du contrôle de ces stations, de l'exécution de tâches ne pouvant pas être entreprises localement: systèmes de gestion de la base de données, compilateurs des différents langages utilisés lors des étapes de descriptions comportementales et structurelles, extraction des schémas logiques à partir des descriptions géométriques des masques à des fins de vérification, etc..

L'organisation matérielle type qui semble être adoptée par de nombreux centres de conception ressemble au schéma suivant:

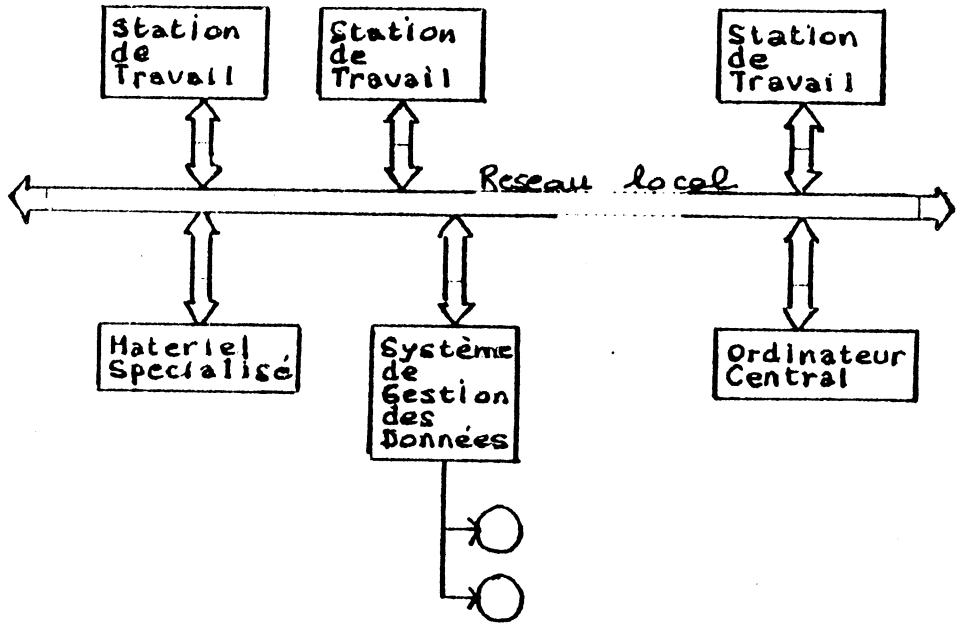


Figure 3.5: Organisation Matérielle d'un Système Intégré d'Aide à la Conception de Circuits Intégrés.

3.6.2 Interface Homme-Machine

L'accroissement de capacité de calcul au niveau de l'interface homme-machine a donné une nouvelle orientation aux logiciels en cours de développement. Une plus grande interaction est désormais possible entre le concepteur et l'ordinateur, et l'organisation temporelle des tâches en est modifiée.

Ce changement est particulièrement sensible dans les phases de conception graphique. En effet, l'utilisation d'éditeurs graphiques est surtout intéressante en tant qu'élément de navigation dans l'espace du circuit intégré à partir duquel de nombreuses opérations locales peuvent être entreprises: vérification de règles de dessin, optimisations topologiques

locales, conversations avec la banque de données pour modifications, extensions, etc...

La plupart des logiciels qui opéraient sur des descriptions globales et qui nécessitaient des temps de calcul très longs (vérification des règles topologiques entre autres) peuvent maintenant être exécutés de manière interactive sur des vues locales diminuant ainsi le temps pendant lequel le concepteur est inactif.

4. APPLICATION:

UN ÉDITEUR GRAPHIQUE POUR LA GÉNÉRATION DE MASQUES DE CIRCUITS INTÉGRÉS UTILISANT UN SYSTÈME DE GESTION DE BASES DE DONNÉES RELATIONNELLES

4.1 INTRODUCTION

L'utilisation de bases de données pour la conception de circuits intégrés est née de deux nécessités:

1. pouvoir réutiliser des structures d'implantation développées lors de la conception antérieure de circuits, afin de construire de nouveaux circuits à moindre coût et plus rapidement puisqu'un grand nombre de structures physiques sont déjà prêtes à être assemblées;

2. gérer de manière homogène l'ensemble des données relatives aux circuits conçus: cela signifie, établir des méthodes d'accès, des mécanismes de protection de données contre d'éventuelles erreurs de manipulation.

Dès 1977, des études concernant leur utilisation dans le cadre du LSI ont été entamées. Deux types de structures de données ont été retenues:

1. l'organisation des données sous forme d'arbres permettant de mettre en relief une certaine hiérarchie inhérente à la structure du circuit conçu;

2. les structures de données relationnelles, permettant un accès simple aux données.

Dans le premier cas, l'organisation hiérarchique des données est obtenue en utilisant des structures de listes pointées auxquelles l'utilisateur n'a pas accès directement.

D'autre part, le récent développement de systèmes de gestion de bases de données relationnelles propose un accès simple à des tables stockant ces données. Chaque type d'objet est représenté à l'aide d'une table constituée de champs ou domaines. Des opérations telles que l'union, l'intersection, la projection peuvent être effectuées sur ces tables, permettant d'avoir accès aux informations désirées.

L'exemple suivant illustre ces deux types d'organisation dans le cas du rangement d'informations générales concernant un circuit.

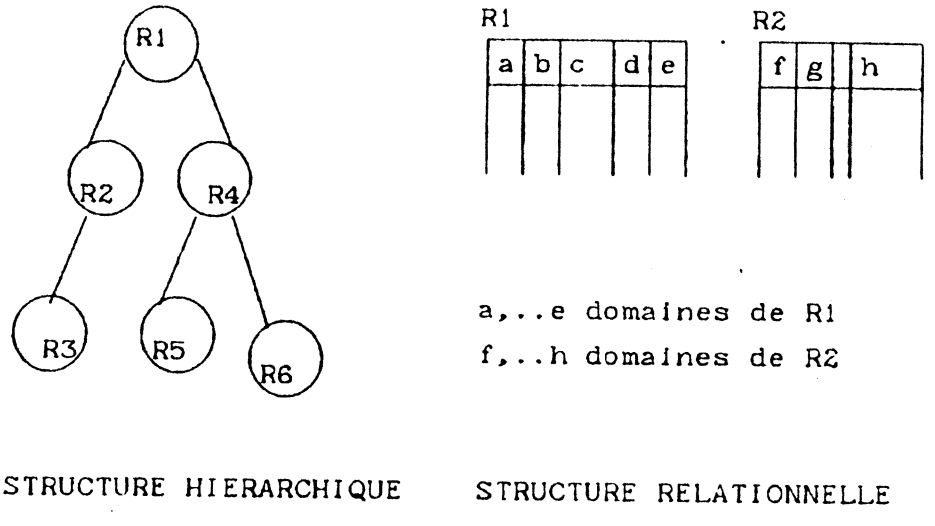


Figure 4.1: Illustration de structures hiérarchique et relationnelle de données rangées en bases de données

La majorité des bases de données développées ces dernières années pour la conception de circuits intégrés sont de nature hiérarchique. Leur fonctionnement reste totalement transparent à l'utilisateur. En effet, ce dernier travaille de manière générale sur une copie de travail contenant une vue particulière de cette base de données sans se préoccuper de la structure physique de stockage de ses données. Aucune interaction n'existe entre l'utilisateur et la base de données si ce n'est par l'intermédiaire de sa copie de travail.

Le récent développement des systèmes de gestion de bases de données relationnelles envisage l'interaction système-utilisateur de manière différente. La structure tabulaire de stockage de ces données est suffisamment simple et souple d'utilisation -et de gestion- pour permettre un accès direct aux données. Différents langages ont été développés dans ce sens tels que SQL, QBE, etc. Ces systèmes ont été largement utilisés pour des applications manipulant des structures de données relativement simples (applications bancaires, services de transport, etc..). La figure suivante illustre ces deux modes d'accès:

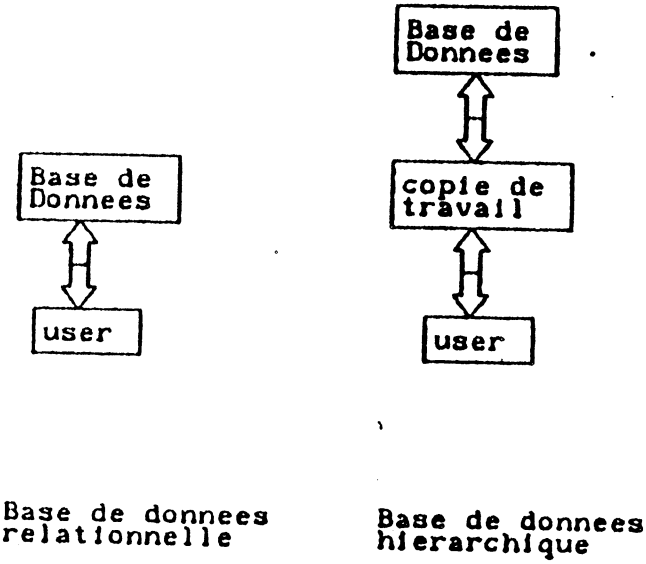


Figure 4.2: Interaction Utilisateur-Bases de Données
pour des Structures de Données Hiérarchique et Relationnelle

Dans le cadre de la conception de circuits intégrés, la complexité des données manipulées est telle que les structures relationnelles s'avèrent être bien moins efficaces que les structures hiérarchiques [GUT82], [BEE82]. En effet, avoir accès à l'ensemble des données caractéristiques d'une UAL (unité arithmétique et logique) nécessite la consultation non pas d'une seule mais d'un ensemble de tables contenant les informations que désire l'utilisateur. Le temps de lecture et d'écriture dans ces tables croît rapidement dans la mesure où un accès individuel doit être réalisé. Toutefois l'accès interactif et simple aux données que permettent ces systèmes est un facteur important qui a, jusqu'ici, permis leur croissance.

Des extensions de ces systèmes relationnels sont actuellement en cours de réalisation permettant d'intégrer la notion de hiérarchie au sein de ces structures relationnelles [LOR82], [KAT82].

System R, le système de gestion de bases de données relationnelles développé à IBM San José, est aujourd'hui modifié dans ce sens. Ce chapitre présente une application dont le but principal est la validation de ces modifications. La première partie est consacrée à la présentation de la notion d'Objet Complexe utilisé comme structure de données de base pour la description de circuits logiques; la seconde présente l'application en elle-même: le développement d'un éditeur graphique permettant la description géométrique de ces circuits.

4.2 NOTION D'OBJET COMPLEXE

La notion d'Objet Complexe a été développée au Laboratoire de Recherche d'IBM San José sur le système de gestion de bases de données relationnelles, System R [HAS81], [LOR81]. Cette extension de System R s'est avérée nécessaire car l'utilisation de simples relations comme support de représentation de circuits logiques n'était pas suffisante. En effet, un circuit intégré est caractérisé par des types de données tellement variés et des relations entre ces données tellement complexes, que l'utilisation de System R comme tel est peu adéquate. Il est important que ces différents types de données soient accessibles sous forme d'un tout car ils donnent des indications sur différentes représentations possibles d'un même objet.

Une organisation hiérarchique possible de ces relations est illustrée dans l'exemple suivant. La racine de cet arbre contient les informations générales associées au circuit: le nom du créateur, la date de conception, etc... Chacune de ses relations-enfants contient des informations concernant les différentes macro-structures constituant ce circuit: l'UAL, les registres, mémoires, structures de contrôles utilisées ainsi que la manière dont elles communiquent entre elles. Aux niveaux les plus fins de description, un ensemble de règles, soit électriques soit topologiques doivent être présentes.

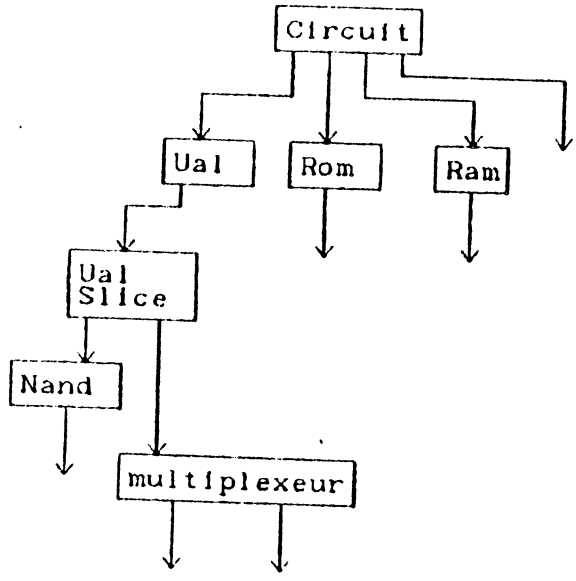


Figure 4.3: Organisation Hiérarchique des Données associées à un Circuit

Au niveau de gestion interne de la base de données, trois nouveaux types de données sont reconnus par l'analyseur sémantique:

1. le type IDENTIFIER, identificateur interne associé à chaque tuple créé dans l'une quelconque des tables,

2. le type COMPONENT-OF, identificateur 'pointant' vers le tuple de la relation parent associée,

3. le type REFERENCE, permettant des références à des tuples d'un même ou d'un autre objet complexe.

La structure hiérarchique associée à un circuit n'est pas hiérarchique au sens strict du terme. En effet, l'utilisation du type REFERENCE la transforme plutôt en réseau. L'existence de ce type permet de distinguer les données constituant un circuit, des données utilisées pour la conception qui n'entrent pas directement dans sa structure, comme la liste des règles de technologie par exemple. De plus, il permet d'éviter des duplications et donc de minimiser la quantité d'information rangée dans la base de données. Nous continuerons toutefois à désigner ce type d'organisation

'hiérarchique' car la notion de hiérarchie caractérise mieux l'organisation globale d'un circuit, l'existence de références n'ayant pour but qu'une clarification et une minimisation des données mémorisées.

En reprenant l'exemple précédent, chacune des tables utilisées peut être définie comme suit:

```
CREER TABLE CIRCUIT
    (       CID IDENTIFIER,
            CNAME CHAR(*),
            DATE CHAR(6),...)

CREER TABLE UAL
    (       UID IDENTIFIER,
            CID COMPONENT-OF(CIRCUIT),
            X INTEGER,
            Y INTEGER,...)

CREER TABLE SLICE
    (       SID IDENTIFIER,
            CID COMPONENT-OF(UAL),
            X INTEGER,
            Y INTEGER,...)

CREER TABLE NAND
    (       NID IDENTIFIER,
            SID COMPONENT-OF(SLICE),
            X INTEGER,
            Y INTEGER,
            RID REFERENCE(ELEC-RULE),..)
```

L'identificateur RID pointe sur un tuple particulier de la table ELEC-RULE contenant l'ensemble des règles électriques applicables. Cette table n'appartient pas à l'objet complexe CIRCUIT mais elle contient des informations dont a besoin le concepteur pour décrire son circuit.

Les extensions de System R permettent de maintenir l'intégrité de ces structures hiérarchiques à chaque opération réalisée sur la base de données. L'accès aux différentes tables constituant l'Objet Complexe est géré par le système dans le but d'effectuer rapidement des opérations telles que l'insertion, la modification ou la destruction de tuples.

4.3 UN ÉDITEUR GRAPHIQUE POUR LA DESCRIPTION GÉOMÉTRIQUE DE CIRCUITS INTÉGRÉS

4.3.1 Motivations

Cette application, réalisée dans le groupe de R. Lorie à IBM San José, a été entreprise essentiellement pour valider les extensions présentées dans le paragraphe précédent [ET182]. Toutefois elle trouve sa place dans un projet de plus grande envergure consistant à créer un système d'aide à la conception où chaque concepteur travaillerait sur une base de données locale à sa station de travail(micro-ordinateur). Les modifications sont effectuées sur une version particulière de la base de données, sous-ensemble de la base de données centrale. Ces modifications sont ensuite réintgrées en fin de transaction dans la base de données centrale. Un mécanisme de CHECK-IN, CHECK-OUT verrouille les données copiées en base de données locale [LOR82]. Différents types de verrouillage sont utilisés (lecture, écriture avec ou sans effacement,..) permettant à plusieurs concepteurs d'accéder à des informations communes en maintenant la cohérence des données.

4.3.2 Caractéristiques Générales

L'éditeur graphique que nous présentons permet de décrire interactivement la composition géométrique de circuits intégrés. Cet éditeur communique avec la base de données relationnelle locale contenant la description de cellules de base ou de structures plus complexes couramment utilisées (inverseur, point mémoire de RAM, de ROM, bit-slice d'UAL, etc...) soit

en lecture à des fins de vérifications, soit en lecture-écriture si des modifications doivent être effectuées sur les descriptions rangées dans la base de données. La description géométrique obtenue, une fois correcte, est traduite dans un format compris par le système de fabrication des masques local à IBM: GL/1 [LAM81].

Il est clair que des descriptions géométriques d'environ 30 mille transistors ne sont pas réalisées exclusivement à l'aide de tels éditeurs. Des programmes de génération automatique de PLAs, de décodeurs [PER82] permettent de limiter le rôle de ces éditeurs de géométries:

1. à la modification de structures prédéfinies pour les adapter aux caractéristiques du nouveau circuit;

2. au préassemblage de ces structures; à partir des positions initiales définies par le concepteur, des algorithmes de placement et d'interconnexions assurent une organisation topologique optimale (à l'étude à ce jour);

3. à la visualisation de certaines portions du circuit sur lesquelles des modifications et des vérifications restent à faire.

Les principales caractéristiques de cet éditeur, outre le fait qu'elles permettent au concepteur d'avoir accès à la base de données de manière interactive, ont été inspirées des points cités précédemment. Elles peuvent être résumées comme suit:

1. Cet éditeur est basé sur l'utilisation de deux écrans, (que l'on désignera par la suite par ÉcranC: Écran des commandes et ÉcranG: Écran Graphique), l'un étant l'écran associé au terminal utilisé, l'autre étant un terminal graphique supplémentaire. L'ensemble des messages échangés entre le concepteur et l'éditeur apparaît sur l'ÉcranC alors que la sélection des commandes graphiques, les changements d'états de l'éditeur sont réalisés sur l'ÉcranG via l'utilisation d'un curseur manuel ou d'un crayon lumineux. Cette dissociation nous paraît importante car elle

permet de ne pas surcharger l'écran graphique de messages envoyés par le système.

2. Il ne manipule que des géométries rectangulaires et orthogonales entre elles. Cette restriction peut paraître limitative car elle ne va pas dans le sens d'augmentation de la densité du circuit. Mais elle permet d'utiliser des outils logiciels de placement, d'interconnexions et de vérifications de règles topologiques plus simples et par là-même plus rapides.

3. Des étiquettes peuvent être créées sur n'importe quel niveau de masques, pour accroître la lisibilité des descriptions.

4. Les structures géométriques créées sont organisées de manière hiérarchique; les types 'copie simple' et 'copie multiple' (ou matrice) sont utilisés à ces fins. Des transformations graphiques peuvent être realisées sur ces copies: ce sont des combinaisons des symétries par rapport aux deux axes du système de référence et de rotations d'angles multiples de 90 degrés (au total huit orientations différentes de la copie de manière à respecter la structure orthogonale des descriptions). De plus, deux opérations spéciales ont été développées, permettant au concepteur (1) de modifier rapidement la structure hiérarchique des objets qu'il crée, (2) de créer interactivement des variants à un objet déjà existant. Ces deux opérations nous semblent importantes car elles permettent d'accroître le contrôle du concepteur sur l'objet qu'il crée.

5. A chaque objet sont associées deux représentations:

   a. une vue externe, composée d'un cadre (rectangle) et de connexions (surfaces rectangulaires indiquant les points de contact de cet objet avec son environnement),

   b. une vue interne donnant des indications sur sa composition structurelle.

En fonction du degré de détail désiré par le concepteur, la vue externe ou interne d'un objet sera représentée sur écran. L'exemple suivant montre trois représentations possibles d'un même objet selon le niveau hiérarchique choisi par le concepteur.

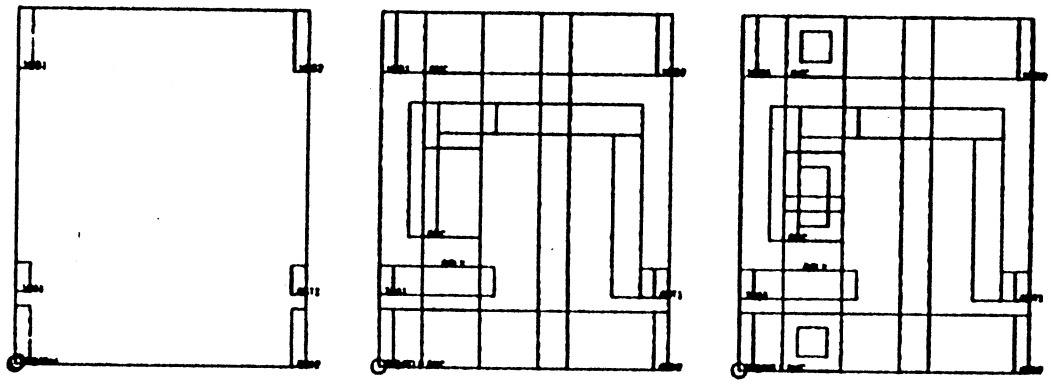


Figure 4.4: Trois Représentations Possibles
d'une Cellule de Base d'un Registre à décalage

Outre ces transformations (rotations, symétries) permises sur les copies, des opérations graphiques ont été définies, permettant d'effacer, de re-positionner certaines copies ou primitives et réaliser des modifications (compression ou extension) sur un ensemble d'objets. Cette dernière option est intéressante car elle permet de créer ou modifier, graphiquement et rapidement, des structures répétitives.

La liste ci-jointe ne représente pas l'énumération exhaustive des opérations que cet éditeur réalise. Elle essaie de mettre en relief certains points particuliers à cet éditeur. Une énumération détaillée des commandes qu'il exécute fait l'objet du paragraphe 4.3.4.

4.3.3 Structures de Données

Cette application est organisée autour de trois principales structures de données:

1. l'ensemble des données graphiques gérées par un Contrôleur Graphique PBS (Picture Building System) [PAL79];

2. l'ensemble des données rangées en base de données, modifiée à chaque opération graphique réalisée;

3. une copie en mémoire des objets en cours d'édition. Cet ensemble de données contient uniquement les informations dont a besoin le Contrôleur Graphique.

4.3.3.1 Structure de la base des données géométriques

La structure de données utilisée pour représenter les informations d'ordre géométrique associées aux éléments physiques référencés utilise la notion d'Objet Complexe précédemment définie. L'ensemble des données géométriques associé à un circuit est contenu dans les relations suivantes:

1. La table CELLULES contient les informations générales associées à ce circuit, le nom du créateur, la date de conception ou de modification, les commentaires renseignant sur la fonction du circuit par exemple. Cette table contient également les dimensions du circuit (vue externe).

2. La table RECTANGLES contient la liste des rectangles utilisés dans ce circuit. Chaque rectangle est caractérisé par le type de masque sur lequel il doit être dessiné, ainsi que par la position de deux de ses sommets, diagonalement opposés.

3. La table ETIQUETTES contient la liste des étiquettes utilisées. Chaque étiquette est caractérisée par un champ texte et une position dans l'espace cartésien associé au circuit.

4. La table INSTANCES contient la liste des 'cellules' utilisées comme copie simple dans le circuit considéré. Elle est composée des champs suivants: position de la copie, transformation

effectuée sur celle-ci (rotations, symétries: voir le paragraphe précédent) ainsi que l'identificateur interne de la cellule copiée.

5. La table MATRICES contient la liste des cellules utilisées comme copies multiples dans le circuit. Elle est composée des champs suivants: position de la copie, transformation effectuée sur celle-ci (rotations, symétries), le nombre de duplications verticales et horizontales, l'espacement entre chaque copie dans chaque direction ainsi que l'identificateur interne de la cellule dupliquée.

6. La table CONNEXIONS contient la liste des points (ou surfaces) de contact du circuit avec son environnement extérieur. Chaque connexion est caractérisée par un rectangle et une étiquette (utilisée pour des vérifications de connectivité).

L'organisation hiérarchique de ces tables est représentée par la figure suivante:

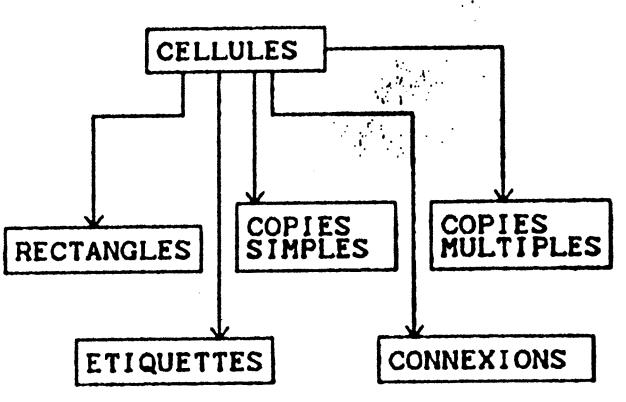


Figure 4.5: Représentation Hiérarchique de Données Géométriques en Base de Données

4.3.3.2 Représentation en mémoire des 'Objets Complexes'

La sélection d'une profondeur maximale de hiérarchie donne le choix au concepteur de visualiser une représentation particulière du circuit qu'il conçoit. La quantité d'information associée à cette représentation varie selon la profondeur choisie. Une copie de ces données (relatives à la vue partielle) est chargée en mémoire, accélérant ainsi le processus d'édition (par rapport au chargement global des données associées au circuit). Chaque cellule référencée durant une session de travail est donc chargée en mémoire de la manière suivante: une table centrale, analogue à la table CELLULES, contient le nom, les dimensions de la cellule ainsi que les pointeurs vers deux zones dans lesquelles sont rangées les informations associées à chaque vue. La liste des connexions est rangée dans l'espace associé à la vue externe, les quatre structures de listes doublement pointées contiennent les informations des tables RECTANGLES, INSTANCES, MATRICES et ETIQUETTES, dans l'espace associé à la vue interne. La figure suivante illustre l'organisation en mémoire de chaque Objet Complexe.

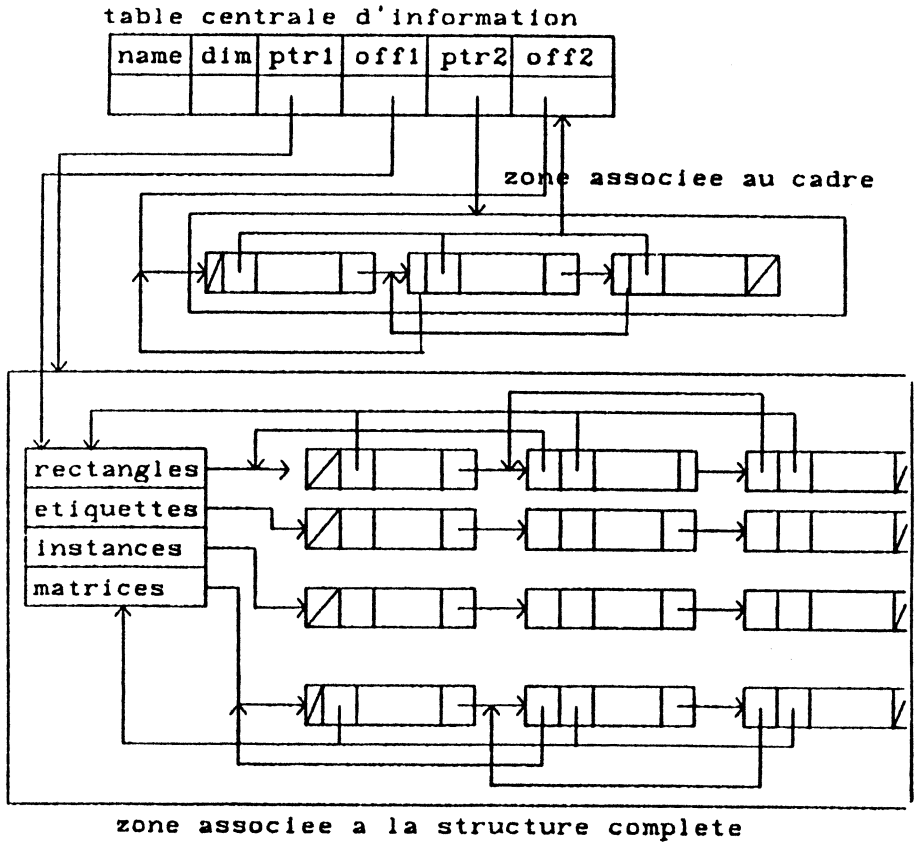


Figure 4.6: Représentation en Mémoire d'un Objet Complexe

La représentation complète des pointeurs est donnée pour les types connexion et matrice seulement, pour des raisons de clarté du schéma.

4.3.3.3 Structure de données graphiques

Dans la mesure où le Contrôleur Graphique travaille sur des structures de données qui lui sont propres, une interface a été réalisée, assurant la création, le chargement et la modification correctes de ces structures de données graphiques. Deux tables supplémentaires ont été créées en mémoire, l'une servant d'interface entre les données graphiques manipulées par le Contrôleur et les données rangées en mémoire, l'autre contenant

l'état interne de l'éditeur. Elles sont toutes deux modifiées à chaque opération entreprise par le concepteur.

4.3.4 Fonctions

Les différentes fonctions exécutées par cet éditeur peuvent être regroupées en quatre classes:

(1) les fonctions de définition,
(2) les opérations graphiques,
(3) les opérations effectuées sur la base de données,
(4) les modifications de l'état interne.

Sauf dans le cas des opérations sur la base de données, le séquencement des actions permettant l'exécution de ces fonctions est le suivant:

1. modifier l'état interne de l'éditeur -positionner les curseurs, afficher une échelle correcte, une grille de travail, le niveau de masque courant, etc..;

2. choisir la commande à exécuter, soit créer un élément de géométrie, soit entreprendre une modification graphique.

L'interprétation de ces fonctions est analogue à l'exécution de procédures paramétrées où les paramètres doivent être chargés avec des valeurs appropriées pour assurer le déroulement correct de la procédure.

4.3.4.1 Modifications de l'état interne

L'état interne de l'éditeur est modifiable à partir d'un menu composé des options suivantes:

1. deux curseurs, CR1 et CR2, deux points sélectionnés par le concepteur -grâce aux touches '1' et '2' du clavier- et dont les coordonnées sont mémorisées dans la table de l'état interne de l'éditeur. Ils sont utilisés ensemble ou individuellement pour

indiquer le positionnement de copies ou pour spécifier les dimensions des rectangles.

2. une grille, pouvant être sélectivement affichée et dont l'espacement est modifiable par le concepteur.

3. une échelle, indiquant le nombre d'unités (lambda) contenu sur sur l'écran et modifiable interactivement; le concepteur peut ainsi contrôler la quantité d'information affichée sur l'ÉcranG.

4. la sélection des niveaux de masques représentés graphiquement. Le concepteur peut choisir de ne travailler que sur le niveau métal par exemple et n'avoir que ce dernier représenté sur l'écran. Toutes les opérations entreprises par la suite n'affectent que les géométries du niveau représenté. Outre les sept niveaux classiques associés à la technologie NMOS, des niveaux symboliques ont été définis comme le niveau 'cadre' (ou frame) par exemple.

5. le niveau hiérarchique maximal que le concepteur veut voir représenté graphiquement.

4.3.4.2 Fonctions de définition

Un certain nombre de types de données peuvent être créés par l'utilisateur. La liste ci-jointe indique quelles sont les informations requises pour leur création. Une description plus structurée, sous forme de grammaire, est donnée en Annexe3.

1. Le Rectangle. Les deux curseurs fournissent la position de deux sommets diamétralement opposés. Le niveau de masques doit être choisi et mémorisé dans la table d'états internes.

2. L'Etiquette. Le curseur 1 fournit la position; le niveau de masque mémorisé indique le masque sur lequel cette étiquette sera dessinée. L'utilisateur doit ensuite entrer au clavier de son terminal le texte identifiant cette étiquette.

3. La Copie Simple. Outre la position donnée par la valeur du curseur 1, les informations complémentaires sont requises:

   a. le nom de l'objet à copier,
   b. la transformation (rotation ou/et symétrie) à effectuer sur cette copie,
   c. la position du curseur 1, celle-ci pouvant indiquer quatre positions différentes (les quatre sommets du rectangle associé à la vue externe de cet objet).

4. La Copie Multiple. Le concepteur doit fournir les mêmes informations que dans le cas de la copie simple. De plus, il doit spécifier:

   a. le nombre de duplications selon l'axe horizontal,
   b. le nombre de duplications selon l'axe vertical,
   c. l'espacement horizontal entre chaque copie,
   d. l'espacement vertical entre chaque copie,
   e. l'option suivante permet de créer des matrices dont la cellule de base n'est pas une copie simple mais une paire de copies symétriques par rapport à l'axe horizontal ou vertical. Elle permet le partage entre couples de cellules de base des bus externes comme les bus d'alimentation ou de mise à zéro. L'exemple suivant illustre cette propriété.
   Soit A l'objet représenté sur la Figure de gauche, et B l'objet représenté sur la Figure de droite. B, constitué de quatre copies de l'objet A, est construit en utilisant la commande 'Copie Multiple'.

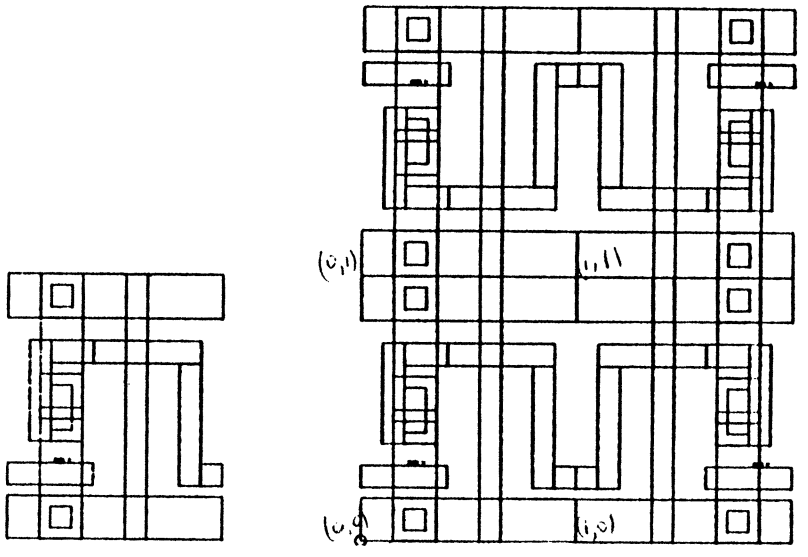


Figure 4.7: Option Miroir pour la création de matrices

La première copie de l'objet -positionnée en (0,0)- A n'a subi aucune transformation. Le nombre de duplications selon l'axe horizontal est deux, de même que selon l'axe vertical. L'espacement entre chaque copie est nul dans les deux cas. L'option miroir a été sélectionnée pour créer à la fois des paires horizontales et verticales. La deuxième copie de A, horizontalement, -positionnée en (1,0)- est la copie miroir de A par rapport à l'axe vertical. La troisième copie de A, verticalement, -positionnée en (0,1)- est la copie miroir de A par rapport à l'axe horizontal et la quatrième est la combinaison de ces deux options: cette dernière copie -positionnée en (1,1)- a été tournée de 180 degrés.

5. Le Cadre. Le concepteur doit définir la vue externe de l'objet qu'il crée s'il désire le référencer plus tard (en effet, le niveau hiérarchique choisi par défaut par le système étant 1, les vues externes des objets copiés seront chargées en mémoire; elles doivent donc exister au préalable). Les dimensions de cette vue externe peuvent être soit calculées par le système, soit choisies par le concepteur (elles sont alors données par la valeur des deux curseurs).

6. La Connexion permet de définir les surfaces rectangulaires à partir desquelles un circuit est en contact avec son environnement extérieur. Ce type de données n'a de sens que lorsque le concepteur travaille sur les vues externes d'objets. Le niveau topologique sur lequel ces connexions sont définies est fictif. Son équivalent logique est le point à partir duquel des bus peuvent être connectés. Outre le rectangle, une étiquette complète la définition d'une connexion. Cette donnée supplémentaire est utilisée ensuite pour des vérifications de connectivité lors de l'extraction de schémas logiques à partir de la description des masques.

Outre ces opérations de définition, deux commandes supplémentaires donnent au concepteur la possibilité de modifier la structure hiérarchique de circuits déjà créés ou d'en créer de nouveaux rapidement. La première peut être illustrée par l'exemple suivant: supposons que l'objet A soit composé de quatre références à l'objet B et de deux références à l'objet C et que B soit composé d'une référence à un objet D et d'une autre à l'objet E. Supposons encore que le concepteur désire modifier la structure géométrique d'une seule des références à B. Cette opération ne peut être permise que si la référence à B est remplacée par la copie complète de l'objet B. Cette opération réalisée par l'éditeur donne au concepteur une grande souplesse d'utilisation et de modifications de copies. Elle consiste ici, à:

a. détruire dans la structure interne de A une des références à B,
b. la remplacer par une copie complète de la structure interne de B.

La Figure 4.8 illustre cette transformation de la représentation hiérarchique de A.

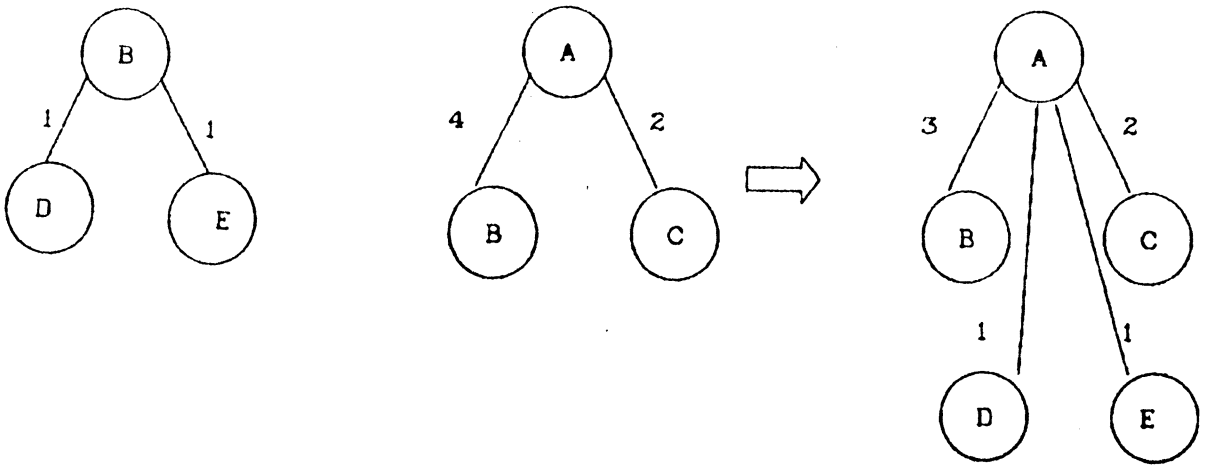


Figure 4.8: Passage de référence à copie.

La seconde commande permet de créer un nouvel objet A' à partir de l'objet A représenté sur l'ÉcranG. Cette opération est réalisée en choisissant i composants de A et en les regroupant pour constituer A'. Ces i composants sont soustraits de A pour former une cellule autonome. Sur l'ÉcranG n'apparaîtra que la nouvelle version de A. A' pourra être accédé dans une nouvelle session d'édition car il figure désormais en base de données. Dans l'exemple de la Figure 4.9, A' est construit en ôtant de A une référence à B et une autre à C.

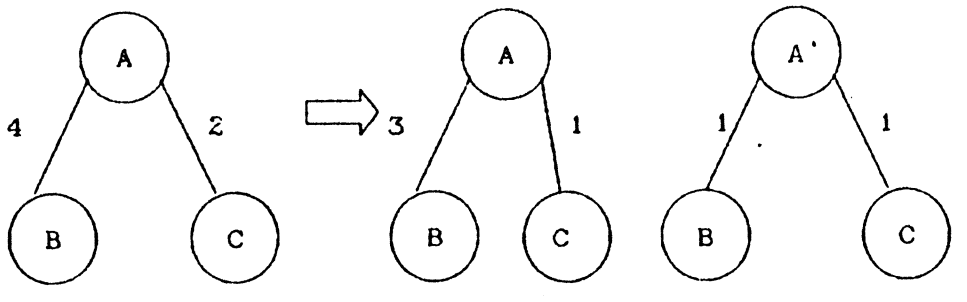


Figure 4.9: Réalisation de variants.

4.3.4.3 Opérations graphiques

Les opérations graphiques que le concepteur peut utiliser sont de deux ordres:

1. les opérations classiques telles que la translation (move), la modification (update) ou l'effacement (erase). Le nombre d'étapes requises pour les exécuter est minimal:

   a. choix de l'élément grâce au crayon lumineux,
   b. positionnement du curseur1 pour indiquer la nouvelle position ou le nouveau sommet du rectangle à modifier,
   c. choix de la commande (move, update ou erase);

2. un couple d'opérations 'spéciales' permettant de regrouper sous une seule action des modifications (identiques) d'un ensemble de structures. Elles nécessitent toutes deux l'utilisation d'une zone rectangulaire de travail appelée 'rectangle stretch' défini sur un niveau de masques virtuel nommé 'niveau stretch'.

   a. La première nécessite, outre ce rectangle stretch, un vecteur (défini par curseur1, curseur2) dont la direction et la longueur indiquent de combien les rectangles intersectant ce rectangle stretch seront étirés, (ou compressés si cette longueur est négative), et dans quelle direction. Si des copies intersectent ce rectangle stretch, elles seront déplacées d'une longueur et dans la direction indiquée par le vecteur d'étirement. L'exemple suivant montre le résultat d'une telle opération sur une cellule de base d'un registre à décalage.

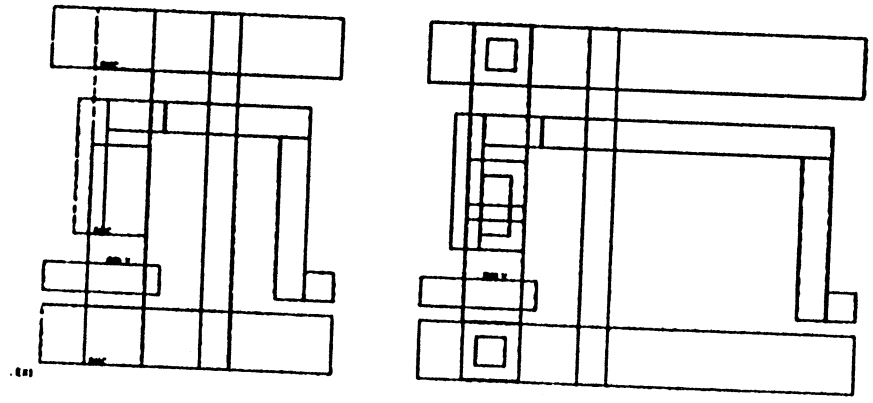


Figure 4.10: Etirement d'une cellule de base de registre à décalage

b. La seconde utilise aussi ce rectangle stretch, sans toutefois utiliser le vecteur d'étirement. Elle permet de réaliser des 'trous' dans l'objet représenté sur écran. Tous les rectangles (et copies, simples ou multiples), ayant une intersection non vide avec ce rectangle stretch seront effacés. L'exemple suivant illustre cette opération.

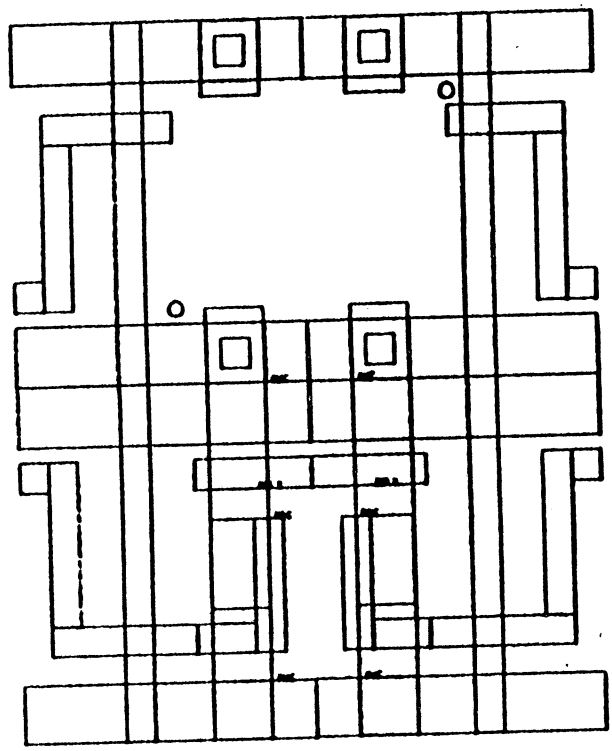


Figure 4.11: Réalisation de trous dans une matrice

4.3.4.4 Interactions avec la Base de Données

Le concepteur peut travailler sur la base de données de deux manières:

1. à partir du menu de commandes, il peut effacer de la base de données un objet (toutes les tables contenant des entrées relatives à cet objet sont mises à jour). Il peut aussi terminer une transaction de manière à fixer définitivement le travail qu'il vient d'effectuer ou bien décider d'effacer l'ensemble des opérations qu'il a effectuées depuis la transaction précédente;

2. à partir d'un mode spécial, il peut communiquer interactivement avec la base de données en utilisant le langage SQL pour effectuer des vérifications sur ses structures de données, des insertions ou effacements. Dans la mesure où chacune de ces sessions commence

et termine une transaction, la nouvelle version de la base de données est accessible par l'éditeur dès la transaction suivante.

4.3.5 Prototype

Le prototype réalisé au Laboratoire de Recherche d'IBM San José a été développé en PL/1 sous environnement CP/CMS. Il utilise System R comme système de gestion de base de données et PBS (Picture Building System) comme système de gestion des données graphiques. PBS permet à l'application de travailler sur un terminal graphique virtuel. Il se charge d'interpréter et de représenter graphiquement les données selon les conventions du terminal de travail.

Trois interfaces ont été réalisées, se chargeant de l'insertion ou des modifications des trois principales structures de données: graphiques, en mémoire et en base de données. Chacune des commandes accessibles par l'éditeur modifie ces trois structures de données et positionne les tables réalisant l'interface entre ces structures (table de l'état interne de l'éditeur ainsi que la table des clés permettant d'identifier en mémoire un élément choisi graphiquement).

4.3.6 Un Interpréteur de GL/1

Une interface permettant la création d'un fichier utilisable par le système de conception de circuits logiques IGS [CAR80] a été développée. Elle convertit les données relatives à un objet complexe rangé en mémoire dans le langage GL/1. GL/1 est un langage de description de primitives géométriques: rectangles, polygones, lignes, cercles organisés de manière hiérarchique sous forme de cellules plus complexes [LAM81]. Un objet géométrique décrit en GL/1 est organisé en une série d'instructions et contient un code opération suivi des données relatives à cette opération (définition, copie, copie multiple, etc..). Le début et la fin des descriptions sont indiqués par des codes opérations spéciaux. Son organisation est donnée en Annexe4.

Une des raisons d'être de cet interpréteur est la possibilité d'utiliser l'ensemble des outils de vérification disponibles sur EDS de manière à vérifier les circuits conçus par cet éditeur. En effet, l'état présent du prototype ne fournit au concepteur aucun outil de vérification ni d'interconnexion automatique de blocs, ni de simulation. Il est donc nécessaire de fournir une telle ouverture au concepteur.

4.4 CONCLUSION

Le travail présenté ici ne représente qu'une très faible part de ce que doit être un système efficace d'aide à la conception de circuits intégrés. Tout d'abord, il ne permet de créer que des géométries sans fournir de possibilités de vérifications de règles de dessin. Il ne permet pas non plus d'extraire des informations caractérisant son comportement électrique (résistivité, capacité des transistors qui le constituent, etc..).

D'autre part, aucune information d'ordre logique n'est intégrée actuellement dans la base de données. Il serait utile de définir, au niveau logique, un ensemble de types de données, un éditeur graphique les manipulant et surtout d'étendre la représentation en base de données des circuits décrits de manière à ce qu'elle regroupe à la fois des informations d'ordre logique et topologique sous le même Objet Complexe.

Enfin, il serait nécessaire, à partir de descriptions d'algorithmes de contrôle en langage RTL, de générer directement en base de données les descriptions géométriques des PLAs utilisés; un nouveau type de données graphiques (PLA) permettrait de visualiser leur représentation graphique en utilisant cet éditeur.

Muni de cet ensemble d'extensions, ce système graphique d'aide à la conception de circuits intégrés serait une concrétisation des différentes philosophies énoncées dans les chapitres précédents et permettrait une intégration aussi complète que possible de toutes les données relatives à un circuit, accessibles et modifiables interactivement par les concepteurs.

4.5 COMPARAISON AVEC LUCIE ET CALMA

Dans ce paragraphe, nous nous proposons d'effectuer une comparaison entre l'éditeur graphique présenté précédemment (que nous désignerons par la suite par ED) et deux autres éditeurs conçus pour la description de masques de circuits intégrés: LUCIE développé à l'IMAG (Grenoble) et CALMA, système commercialisé et couramment utilisé par les compagnies de fabrication et de conception de circuits.

Bien que ces éditeurs aient été développés à des fins très différentes (LUCIE a été conçu pour l'enseignement et ED est un projet de recherche), certains points de comparaison ont été retenus.

a. Interface utilisateur

Comme CALMA, ED propose un interface graphique basé sur l'utilisation de deux écrans, l'un servant à l'échange de messages entre le concepteur et le système, l'autre utilisé pour la représentation des schémas et des menus.

LUCIE, par contre, ne dispose que d'un terminal graphique; la liste des commandes exécutées et la représentation des dessins se partagent le même écran, réduisant ainsi la lisibilité.

L'ensemble des commandes d'ED, comme pour CALMA, est présenté sous forme de menus, sélectionnés directement sur écran, alors que l'envoi de commandes graphiques, en LUCIE, est plus complexe (utilisation conjointe du clavier et sélection de points sur écran). Le jeu de commandes d'ED se rapproche de celui de LUCIE par sa simplicité (50 commandes au lieu de 400 environ dans CALMA).

Au niveau des performances et opérations graphiques, ED se situe entre CALMA et LUCIE.

Bien que l'ensemble des opérations de définition et de création de figures d'ED soient similaires à celles de LUCIE, les opérations que propose ED sur les groupes de figures ou sur leurs structures internes, et le contrôle sélectif du détail des schémas le rendent plus souple d'utilisation et plus élaboré.

b. Accès aux données - Sécurité - Protection

Dans LUCIE et CALMA, l'aspect sécurité et protection des données lors des accès est laissé au soin du concepteur (réalisation de copies de travail) avec possibilité d'avoir accès à d'anciennes versions. Dans le cas d'ED, ces problèmes sont pris en charge et résolus par un système de gestion de bases de données qui permet la consultation de cette base de données soit par programme, soit interactivement par un langage de communication très simple.

La contrepartie de l'utilisation de ce logiciel supplémentaire est d'une part le ralentissement de l'accès aux données (très pénalisant dans ce domaine) et la nécessité d'une machine plus puissante capable de supporter de tels logiciels.

System R est un système trop général à mon avis pour être utilisé de manière compétitive pour la conception de circuits intégrés. Mais la mise au point de sous systèmes de gestion de bases de données relationnelles dédiées comme INGRES, par exemple, semble très prometteuse.

c. Extensibilité

Confronter les éditeurs précédents sur ce point est une tâche délicate car elle est étroitement liée aux motivations sous-jacentes à leur développement.

CALMA est un système commercial conçu exclusivement pour la saisie de descriptions de masques. Aucun outil de simulation électrique ou de vérification de règles de dessin n'y est, à ce jour, inclus et ne peut être incorporé.

LUCIE et ED sont des systèmes plus ouverts qui permettent chacun à leur manière le développement et l'ajout de logiciels supplémentaires.

LUCIE, grâce à la traduction de ses fichiers en un code intermédiaire permet le développement de logiciels d'aide au placement, à la vérification de schémas, ou à la génération de ROMs et PLAs.

De manière analogue, mais par accès à la base de données intégrant l'ensemble des informations relatives au circuit, des programmes de simulation, optimisation et placement doivent étendre ED à un système plus complet pour la génération de masques pour circuits intégrés.

CONCLUSION

CONCLUSION

Le travail présenté ici a été développé dans un double but:

1. présenter les méthodologies et un ensemble des outils informatiques mis en oeuvre pour mener à bien la conception de circuits intégrés;

2. présenter une contribution personnelle réalisée tout d'abord à Grenoble, puis au laboratoire de recherche d'IBM San José.

La méthodologie de conception présentée dans cet ouvrage s'insère dans les résultats de recherche sur le développement de circuits à haute intégration (LSI) effectués durant cette dernière décennie. Dans le but de produire des circuits exécutant des fonctions toujours plus complexes, elle propose une démarche hiérarchique de segmentation de problèmes ainsi que l'utilisation de plus en plus importante d'outils automatiques d'aide à la conception.

Dans la mesure où l'aide informatique intervient quasi systématiquement à chacune des étapes de la conception, il est difficile de dissocier la méthodologie utilisée des outils d'aide à la conception. Il me semble qu'une méthodologie de conception doit guider le développement d'outils d'aide à la conception; d'autre part, les résultats obtenus grâce à l'utilisation de tels outils permettent de réajuster ou même de modifier les méthodologies sous-jacentes. Une redéfinition constante des méthodes et des outils est nécessaire pour résoudre les problèmes de plus en plus complexes soulevés par l'évolution des techniques d'intégration et de production. Leur adéquation aux problèmes qu'ils doivent résoudre limite donc considérablement leur durée de vie.

Il est donc important:

1. de concevoir ces outils de manière modulaire de façon à limiter la portée des répercussions que certains changements dans un programme pourraient avoir sur le reste du système;

2. d'assurer une indépendance maximale entre les programmes développés et le matériel utilisé. Le renouvellement du matériel étant, à l'heure actuelle, beaucoup moins onéreux que le développement de nouveaux logiciels, il faut permettre l'utilisation de ces derniers sur plusieurs générations de machines (terminaux ou unités de calcul).

L'ensemble des outils d'aide à la génération de circuits se réfèrent toujours à un système de gestion de base de données (ou de programmes) regroupant l'ensemble des informations nécessaires.

Il est important d'assurer une stabilité maximale de ce noyau. Tout changement à son niveau aurait des répercussions très coûteuses sur l'ensemble des outils qui se réfèrent à lui.

Les outils d'aide au développement de circuits intégrés présentés servent à la fois comme support de développement et comme référence de base pour le développement d'outils encore plus adéquats pour les générations futures de circuits intégrés. Nous avons essayé de présenter ici l'état des recherches dans ce domaine de 1978 à 1982 ainsi que les orientations qui semblent se dessiner pour les années à venir. Seules des prévisions et anticipations à court terme peuvent être énoncées.

Les contributions présentées dans ce travail n'ont pas pu être testées à grande échelle faute de temps essentiellement, mais elles m'ont permis d'aborder le problème de la conception de circuits intégrés sous des angles différents et surtout d'avoir une connaissance plus réaliste des supports logiciels et matériels nécessaires pour concevoir de tels circuits.

BIBLIOGRAPHIE

CHAPITRE I

[ALL79] J.Allen,
'Requirements for a Research-Oriented Design System',
Caltech Conference on VLSI, January 1979.

[AYR79] R.Ayres,
'Silicon Compilation - A Hierarchical Use of PLAs',
Caltech Conference on VLSI, January 1979.

[CLE79] W.M.VanCleemput,
'Hierarchical Design for VLSI: Problems and Advantages',
Caltech Conference on VLSI, January 1979.

[CLE82] W.M.Van Cleemput,
'CAD Tools for Custom Integrated Circuit Design',
IEEE Compcon 82.

[MEA79] C.Mead, L.Conway,
'Introduction to VLSI Systems',
Addison-Wesley 1979.

[NEW82] M.Newell, D.Fitzpatrick,
'Exploiting Structure in Integrated Circuit Design Analysis',
1982 Conference on Advanced Research in VLSI, MIT.

[SEI79] C.Seitz,
'Self-Timed VLSI Systems',
Caltech Conference on VLSI, January 1979.

[SIS82] J.Sisking, J.Southard, K.Crouch,
'Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions',
1982 Conference on Advanced Research in VLSI, MIT.

[STE82] M.Stefik, D.Bobrow, A.Bell, H.Brown, L.Conway, C.Tong,
'The Partitioning of Concerns in Digital System Design',
1982 Conference on Advanced Research in VLSI, MIT.

CHAPITRE II

[BAT80] J.Batali, A.Hartheimer,
'The Design Procedure Language Manual',
Artificial Intelligence Laboratory, MIT,
Memo No.598, VLSI Memo 80-31 unpublished, Sept 1980.

[ETI81] R.Etienne,
'Formalisme DELTA',
Rapport de DEA, IMAG, Grenoble, Sept.81.

[NEM81] Nemour M.,
'Formalisme DELTA , un Outil de Description pour la Synthèse Automatique dans la Conception de Machines Séquentielles Synchrones',
Thèse de Docteur 3ème Cycle, IMAG, Grenoble, Dec.81.

[REI81] R.Reis,
'Topological Evaluator for VLSI circuits',
IMAG Research Report No.216, Grenoble, France, June 81.

[OBR82] M.Obrebska,
'Etude Comparative de Parties Contrôles de Microprocesseurs',
Thèse de Docteur-Ingénieur, IMAG, Juin 82.

[SUZ81] A.A Suzim,
'Etude des Parties Opératives à Eléments Modulaires pour Processeurs Monolithiques',
Thèse de Docteur-Ingénieur, IMAG, Nov.82.

CHAPITRE III

[ARN82] M.Arnold, J.Ousterhout,
'LYRA: A New Approach to Geometric Layout Rule Checking',
ACM/IEEE 19th Design Automation Conference, June 82.

[CAR80] P.Carmody,
'THE IGS System',
Proc.17, D.A.C 1980, p.430

[CLA80] D.Clary, R.Kirk, S.Sapiro,
'SIDS: A Symbolic Interactive Design System',
ACM/IEEE Design Automation Conference, 1980

[DON82] R.Donze, J.Sanders, M.Jenkins, G.Sporzynski,
'PHILO: A VLSI Design System',
ACM/IEEE 19th Design Automation Conference, June 82.

[ELL81] S.Ellis,
'SLL, a Symbolic Layout Language',
Master Thesis Report, EECS's Department, U.C Berkeley, Sept.81.

[JEN82] R.M.Jennings, T.H.Edmonson,
'A Low Cost Graphics System for VLSI',
Computer Graphics Magazine,Vol.14, No.4, July 82.

[KAH79] H.Kahn, A.Burston, D.Kinniment,
'ADL: An Hierarchical Logic Design Language',
Caltech Conference on VLSI, January 1979.

[KEL81] K.Keller,
'KIC, An Interactive Graphics Editor for Integrated Circuits',
Master Thesis Report, EECS Department, U.C Berkeley, Sept 81.

[LAN81] H.Landman,
'Automatic Layout of Optimized PLA Structures',
Master Thesis Report, EECS Department, U.C Berkeley, Nov 81.

[MAT82] R.Mathews, J.Newkirk, P.Eichenberger,
'A Target Language for Silicon Compilers',
IEEE Compcon 82.

[NEW81] R.Newton, D.Pederson, A.Vincetelli, C.Sequin,
'Design Aids for VLSI: The Berkeley Perpective',
IEEE Transactions on Circuits and Systems, No.7, July 81.

[OUS82] J.K.Ousterhout,
'Caesar: An Interactive Editor for VLSI Layout',
U.C Berkeley, IEEE Compcon 82.

[PAY82] M.Payne,
'An Integrated VLSI Design System',
VLSI Design Magazine, Jan 82.

[PER82] T.Perez, S.Chiquillanqui,
'PAOLA, A Tool for Topological Optimisations of Large PLAs',
Design Automation Conference 82, Las Vegas.

[PET82] P.Petit,
'Chipmonk: An Interactive VLSI Layout Tool',
Xerox PARC, IEEE Compcon 82.

[SAS82] S.Sastry, S.Klein,
'PLATES: a metric-free VLSI layout language',
1982 Conference on advanced research in VLSI, M.I.T.

[SMI80] V.Smith, R.J.Smith, D.A.Preston,
'COMET: A Fast Component Placer',
ACM/IEEE Design Automation Conference, 1980.

[TRI82] H.W.Trickey, J.D.Ullman,
'A Regular Expression Compiler',
IEEE Compcon 82.

[WES80] N.Weste, B.Acklend,
'An IC Design Station Needs a High Performance Color Graphics Display',
ACM/IEEE Design Automation Conference, 1980

CHAPITRE IV

[BEE82] A.Beetem, J.Milton, G.Wiederhold,
'Performance of Database Management Systems in VLSI Design',
IEEE bulletin on Database Engineering, Vol.5, No.2, June 1982.

[ETI83] R.Etienne, G.Hallmark,
'A VLSI Layout Editor Supported by a Relational Data Base System',
IBM Research Report, January 83.

[GUT82] A.Guttman, M.Stonebraker,
'Using a Relational Database Management System for Computer Aided Design Data',
IEEE bulletin on Database Engineering, Vol.5, No.2, June 1982.

[HAS81] R.Haskin, R.Lorie,
'On extending the functions of a Relational Database System',
IBM Research Report RJ3182, 11/11/81.

[KAT82] R.Katz,
'A Database Approach for Managing VLSI Design Data',
ACM/IEEE 19th Design Automation Conference, June 82.

[KOE82] W.Koenig, R.Lorie,
'Storage of VLSI Layout Designs in a Relationnal Database',
IBM Research Report RJ3548, 7/20/82.

[LAM81] D.Lambert,
'The GL/1 Language',
Proc.18 D.A.C. 1981, p.713

[LEY79] L.Leyking,
'Database Considerations for VLSI',
Caltech Conference on VLSI, January 1979.

[LOR81] R.Lorie,
'Issues in Database for Design Applications',
IBM Research Report RJ3176, 7/10/81.

[LOR81] R.Lorie,
'A project on Design Systems',
IEEE Bulletin on Database Engineering, Vol no.4, Sept. 81.

[LOR82] R.Lorie, W.Plouffe,
'Complex Objects and their use in Design Transactions',
IBM Research Report, RJ3706, December 1982.

[PAL79] F.Palermo, D.Weller,
'Picture Building System',
Proc. Compcon 1979.

ANNEXES

ANNEXE 1

Cette première annexe regroupe les organigrammes utilisés lors du développement à l'IMAG: (1) du simulateur DELTA, (2) du générateur de PLAs.

La figure A.11 illustre l'organisation modulaire du simulateur; les figures A.12 et A.13 détaillent les modules: Traiter ETAs séquentiels et Traiter ETAs Combinatoires.

La figure A.14 illustre l'organisation du Générateur de PLAs les figures A.15 et A.16 détaillent la structure des modules de génération de PLAs.

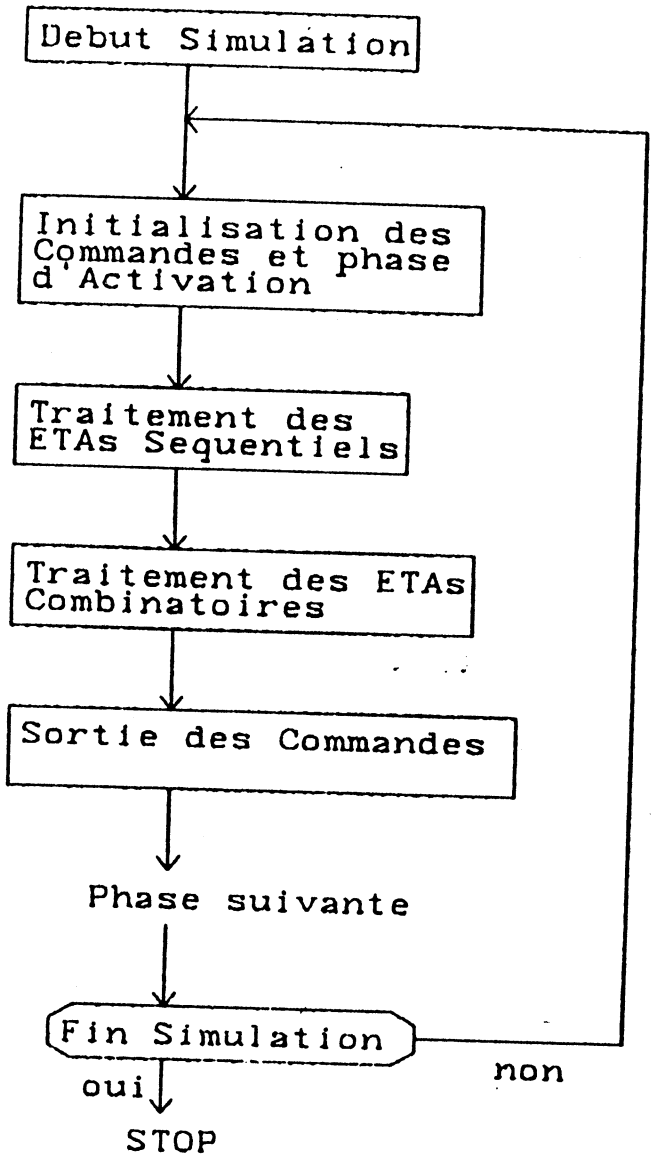


Figure A.11: Organisation Modulaire du Simulateur DELTA

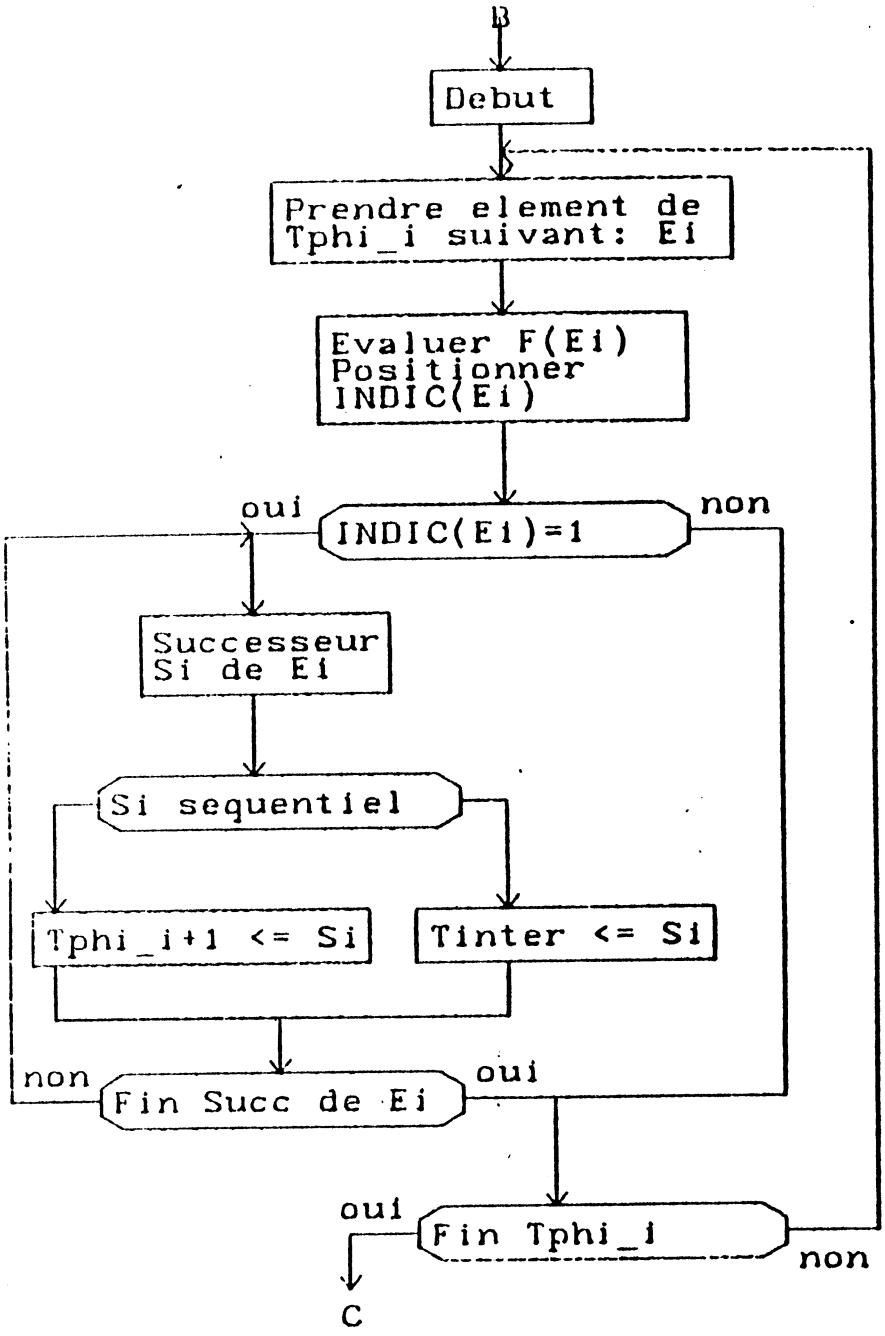


Figure A.12: Description du Module de Traitement des ETAs Séquentiels

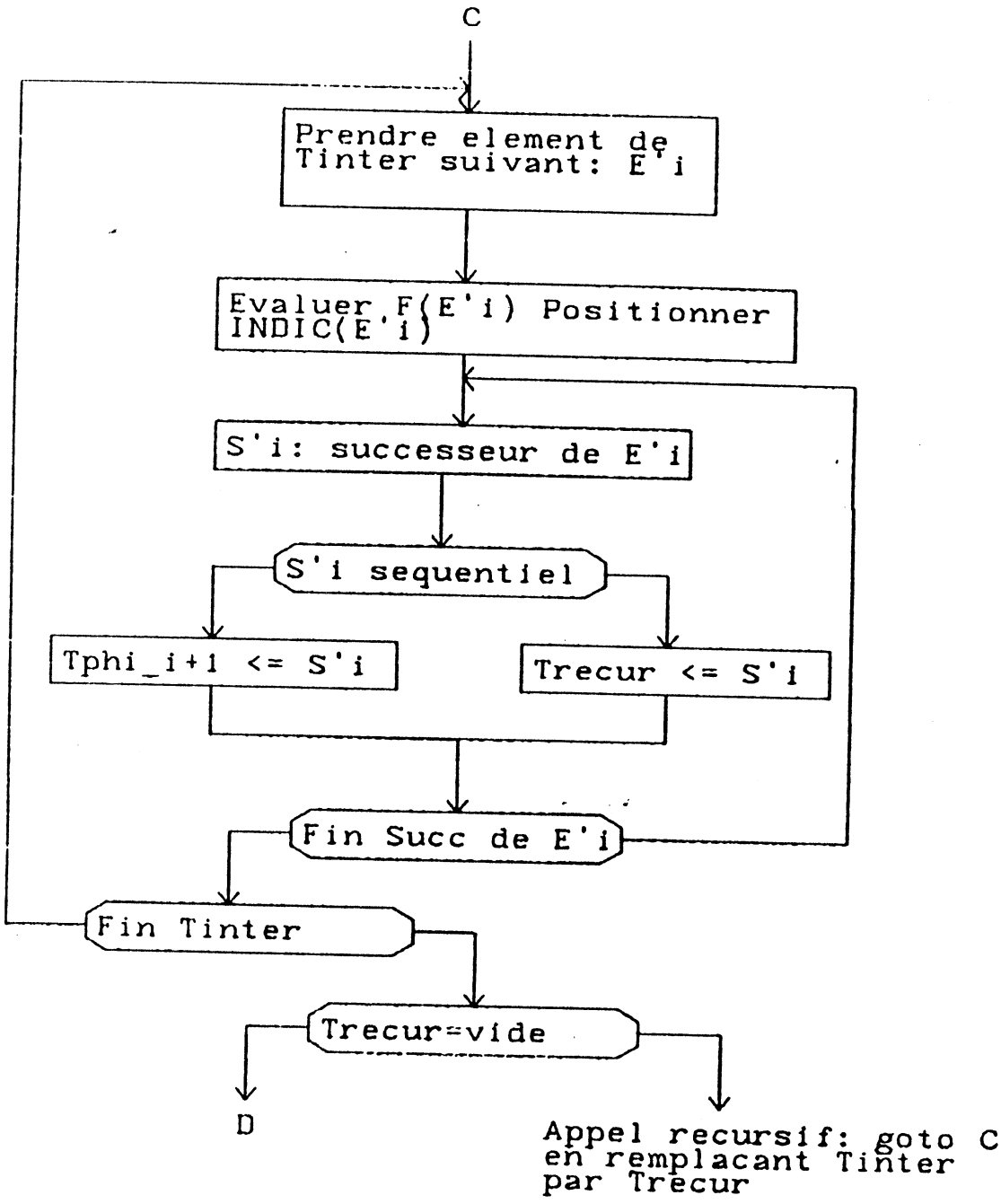


Figure A.13: Description du Module de Traitement des ETAs Combinatoires

Les tables Tinter et Trecur sont des tables intermédiaires utilisées pour le traitement des ETAs combinatoires.

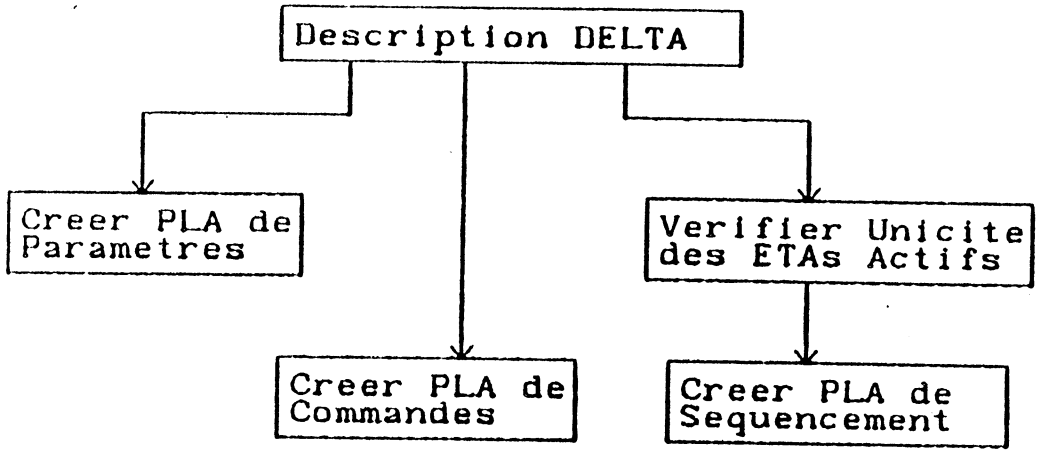


Figure A.14: Organisation Modulaire du Générateur d'une Partie Contrôle tri-PLA.

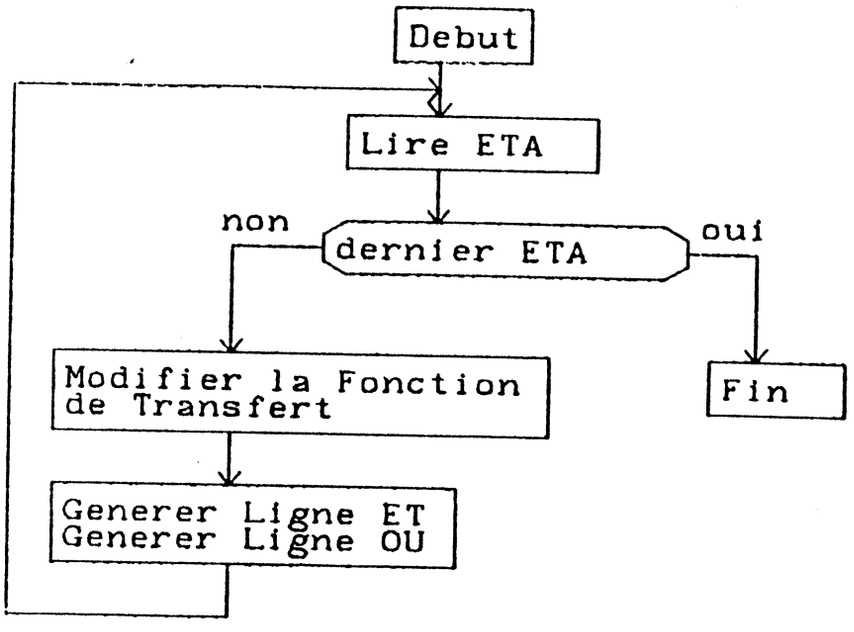


Figure A.15: Description du Module de Génération de PLA

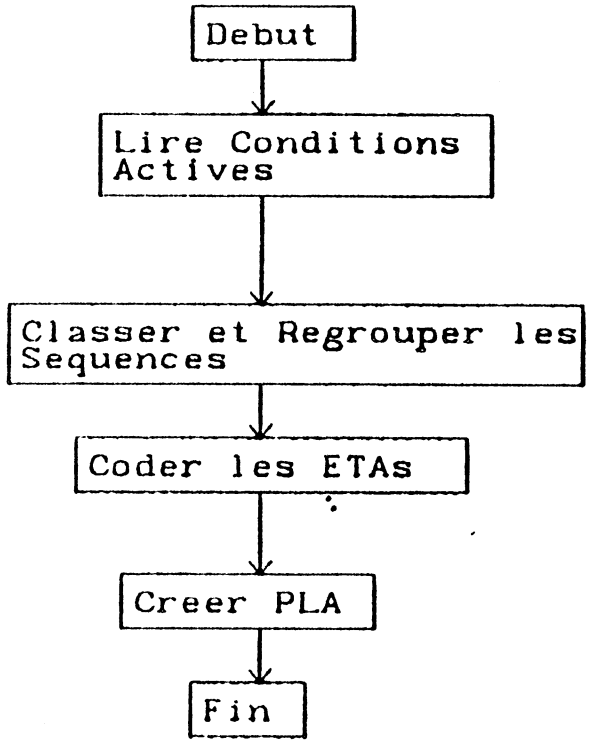


Figure A.16: Description du Module de Traitement du Séquencement

ANNEXE 2

Cette deuxième annexe propose deux exemples d'utilisation de langages de description de géométries, l'un utilisant STIF, développé à l'Université de Berkeley, l'autre CIF. La grammaire BNF associée à STIF est donnée ainsi que la description d'un inverseur paramétré. Le second exemple décrit deux cellules de base de noms n0 et n12 en CIF; les principaux mots-clés utilisés sont:

DC: Début Cellule;
EC: Fin Cellule;
B : Rectangle;
C : Copie;
L : Étiquette:
XX: Interconnexion;

Ils sont interprétés de la même manière que les mots-clés de STIF de même nom .

```
StifFile        = defCell {stifStmnt} endCell.
stifStmnt       = StifFile | DN_Stmnt | DV_Stmnt |
                  M_Stmnt | P_Stmnt | R_Stmnt | B_Stmnt | G_Stmnt | L_Stmnt |
                  FM_Stmnt | T_Stmnt | XX_Stmnt | C_Stmnt | AC_Stmnt |
                  K_Stmnt | SIM_Stmnt | ATT_Stmnt | userStmnt.
defCell         = DC name [parDecList] semi.
endCell         = EC [name] semi.
DN_Stmnt        = DN name eql pathName [parDefList] semi.
DV_Stmnt        = DV name eql value semi.
P_Stmnt         = P [ident] type path {colon path} semi.
M_Stmnt         = M [ident] type path {colon path} semi.
B_Stmnt         = B [ident] type value cma value cma value cma point semi.
R_Stmnt         = R [ident] type value cma point semi.
FM_Stmnt        = F M_Stmnt.
T_Stmnt         = T name type value path semi.
XX_Stmnt        = XX [ident] instName instName type value path semi.
C_Stmnt         = C [ident] pathName [parDefList] point semi.
AC_Stmnt        = AC [ident] name [versionSpec] [paramDefin]
                      arraySize cma point cma point semi.
K_Stmnt         = K relation semi.
SIM_Stmnt       = SIM type simParamDefin semi.
ATT_Stmnt       = ATT type attParamDefin semi.
comment         = "{" commentText "}".
commentText     = {commentChar} {comment} {commentChar}
commentChar     = any ASCII character except "{" or "}".
userStmnt       = digit userText semi.
userText        = {userChar} {comment} {userChar}.
userChar        = any ASCII character except ";" or "{" or "}".
parDecList      = "(" paramDeclar {cma paramDeclar} ")".
paramDeclar     = paramID [eql value] [ "[" value cma value "]" ].
parDefList      = "(" paramDefin {cma paramDefin} ")".
paramDefin      = paramID eql value | transform.
versionSpec     = "[" name {name} "]".
ident           = "<" name ">".
transform       = R eql value | MX | MY .
arraySize       = digit {digit} cma digit {digit}.
instName        = { name "." } name.
pathName        = { name "|" } name.
path            = point {cma point}.
point           = value cma value.
relation        = value rel value.

value           = "(" value ")" | wsp | number | name | expression | no content.
expression      = ~~Expressions in C-style using basic 4 operators.
number          = ~~Floating point numbers defined in the usual way.
```

Figure A.21: Grammaire Formelle Associée à STIF

```
  dc inva(x=8);
  fm np -11,-3,22,27+x;
  t grnleft nm 3,-11,1.5,-11,2.5;
  t grnright nm 3,11,1.5,11,2.5;
  t pwrleft nm 3,-11,18.5+x,-11,19.5+x;
  t pwrright nm 3,11,18.5+x,11,19.5+x;
  t in np 2,-11,7;
  t out nd 2,11,14;
  dv h=(21+x)/2;
  { Make an inverter symbol for the graphical representation }
  g a -6,4,7,h,7.5,h-0.5,8,h,7.5,h+0.5,7,h,-6,17+x,-6,4;
  g a -11,7,-6,h;
  g a 8,h,11,14;
  g a -11,2,11,2;
  g a -11,19+x,11,19+x;
  l "inverter" label 0,h;
  xx grnleft grnright nm 4,-10,2,10,2;
  xx pwrleft pwrright nm 4,-10,19+x,10,19+x;
  b nd 4,4,0,1,2;
  b nc 2,2,0,1,2;
  b nd 4,4,0,1,19+x;
  b nc 2,2,0,1,19+x;
  b nd 4,5,0,1,14.5;
  b nm 4,5,0,1,14.5;
  b nc 2,3,0,1,14.5;
  xx in inend np 2,(-10),7,3,7;
  xx outend out nd 2,1,14,10,14;
  b nd 2,19+x,0,1,(21+x)/2;
  b np 6,4,0,1,7;
  b np 6,x,0,1,15+x/2;
  ec inva;


dc invb;
  dv parm=16;
  { here we pass parameters  and forward reference inva}
  c inva(x=parm) 0,0;
  { here we set the local length scale factor to 100 }
  dn scale=100;
  ec invb;

  dc invc;
  {here we call down the heirarchy}
  c invb|inva 0,0;
  ec invc;
```

Figure A.22: Description d'un Inverseur Paramétré en STIP

```
379   DC n12;
380   (Symbol #110)
381   C n4 0.000000,49000.000000 ;
382   C n4(MY) 0.000000,-49000.000000 ;
383   C n9 -112000.000000,0.000000 ;
384   B NM 4000,32000,0.000000,0,-33000;
385   L "VDD" NM 0,-22500;
386   B NM 2000,12500,0.000000,0,23500;
387   L "GND" NM 0,23500;
388   B NM 2000,12500,0.000000,5250,23500;
389   B NM 2000,12500,0.000000,-5250,23500;
390   B NM 57500,2000,0.000000,-7000,23500;
391   L "VDD" NM 0,-37500;
392   L "GND" NM 0,37500;
393   L "INA0" NM -110000,-37500;
394   L "INB0" NM -110000,37500;
395   L "ACKA" NM -72500,-37500;
396   L "ACKB" NM -72500,37500;
397   L "INA1" NM -45000,-37500;
398   L "INB1" NM -45000,37500;
399   L "OUTA1" NM 35000,-37500;
400   L "OUTB1" NM 35000,37500;
401   L "REQA" NM 72500,-37500;
402   L "REQB" NM 72500,37500;
403   L "OUTA0" NM 110000,-37500;
404   L "OUTB0" NM 110000,37500;
405   L "checkgnd" NM 117500,0;
406   L "checknul" NM 119500,0;
407   L "chwckvdd" NM 121500,0;
408   XX none none NP 500, 112000,1500, 116000,1500, 116000,21500,
86250,21500, 86250,22500;
409   XX none none NP 500, 112000,14000, 113000,14000, 113000,18500,
58250,18500, 58250,22500;
410   XX none none NP 500, -112000,6500, -115500,6500, -115500,21000,
-95750,21000, -95750,22500;
411   XX none none NP 500, 112000,-500, 116000,-500, 116000,-21500,
86250,-21500, 86250,-22500;
412   XX none none NP 500, 112000,-13000, 113000,-13000, 113000,-18500,
58250,-18500, 58250,-22500;
413   XX none none NP 500, -112000,-8000, -115500,-8000, -115500,-21000,
-95750,-21000, -95750,-22500;
414   XX none none ND 500, 114000,8500, 114500,8500, 114500,20000,
83000,20000, 83000,22500;
415   XX none none ND 500, -113500,1500, -120500,1500, -120500,22500;
416   XX none none ND 500, -113500,14000, -113500,19000, -56500,19000,
-56500,22500;
417   XX none none ND 500, 114000,-8000, 114500,-8000, 114500,-20000,
83000,-20000, 83000,-22500;
418   XX none none ND 500, -113500,-500, -120500,-500, -120500,-22500;
419   XX none none ND 500, -113500,-13000, -113500,-19000, -56500,-19000,
-56500,-22500;
```

```
DC n1;
(Symbol #55)
C n0 0.000000,0.000000 ;
B ND 19250,4500,0.000000,12625,-23250;
B NC 1000,500,0.000000,3750,-25000;
B NP 500,1750,0.000000,5000,-23125;
B NP 18750,500,0.000000,14125,-22250;
B NP 18750,500,0.000000,14125,-24000;
B NC 1000,500,0.000000,6250,-25000;
B NM 1000,4250,0.000000,7000,-5125;
B ND 1000,2000,0.000000,7000,-3500;
B NC 500,1000,0.000000,7000,-3750;
B ND 16750,500,0.000000,15625,-2750;
B NC 1000,500,0.000000,8750,-25000;
B NC 1000,500,0.000000,11250,-25000;
B NC 1000,500,0.000000,13750,-25000;
B NC 1000,500,0.000000,16250,-25000;
B NC 1000,500,0.000000,18750,-25000;
B NC 1000,500,0.000000,21250,-25000;
B ND 2000,3250,0.000000,23000,-23125;
B NP 1000,1000,0.000000,23000,-25500;
B NC 500,500,0.000000,23000,-25500;
B NP 500,1250,0.000000,23250,-21625;
B NP 500,2000,0.000000,23250,-25000;
B ND 500,24000,0.000000,24000,-14500;
EC;
```

Figure A.24: Description de deux cellules n0 et n12 en CIF

ANNEXE 3

Cette troisième annexe donne la liste des commandes accessibles à partir de l'éditeur sous la forme d'une grammaire BNF. Les noms en lettres capitales représentent le éléments des menus que l'utilisateur sélectionne.

Les noms en capitales sont des éléments des différents menus affichés sur l'écran graphique. Les notations suivantes sont utilisées:

CR1 = CURSEUR1;
CR2 = CURSEUR2;
CS = COPIE-SIMPLE;
CM = COPIE-MULTIPLE;
STR-L = NIVEAU-STRETCH;
RECT = RECTANGLE;
cor(x) = 'choix de l'objet x sur l'écran';
ent(y) = 'entrer y sur le clavier à partir du clavier';
y = caractère / entier / point;
( )* = zéro ou plus d'occurrences de l'action inscrite entre parenthèse
( )+ = une ou plus d'occurrences de l'action inscrite entre parenthèse
+ = 'et';
/ = 'ou'.

1. COMMANDES DE DÉFINITION

RECTANGLE= cor(CR1) + cor(CR2) + cor(NIVEAU)
+ cor(RECTANGLE);
ÉTIQUETTE = cor(CR1) + cor(ÉTIQUETTE) + ent(texte);
FRONTIÈRE = ent('b') cor(CR1) + cor(CR2) + ent('c');
CONNEXION= cor(CR1) + cor(CR2) + cor(NIVEAU)
+ cor(CONNEXION) + ent(texte);
CS = cor(CR1) + cor(CS) + ent(nom-cellule)
+ ent(transform) + ent(rel-cur1,cadre);
CM = cor(CR1) + cor(CM) + ent(nom-cellule)
+ ent(transform) + ent(nbrepx)

```
+ ent(nbrepy) + ent(espx) + ent(espy)
+ ent(mirry) + ent(mirry);
```

2. COMMANDES D'OPÉRATIONS

```
MOVE        = cor(ob) + (cor(PICK-NEXT))* + cor(MOVE);
UPDATE      =  cor(ob) + (cor(PICK-NEXT))* + cor(UPDATE);
ERASE       =  cor(ob) + (cor(PICK-NEXT))* + cor(ERASE);
FLATTEN     = cor(ob) + (cor(PICK-NEXT))* + cor(FLATTEN);
HEIGHTEN    = cor(HEIGHTEN) + (cor(ob) + (cor(PICK-NEXT))*)+
              + cor(HEIGHTEN) + ent(cell name);
STRETCH     = (cor(CR1)+cor(CR2)+cor(STR-L)+cor(RECT))+
              + cor(CR1) + cor(CR2) + cor(STRETCH);
ETCH        = (cor(CR1)+cor(CR2)+cor(STR-L)+cor(RECT))+
              + cor(ETCH);
```

3. COMMANDES DE CHANGEMENT D'ÉTAT

```
NIVEAUX VISIBLES  = cor(NIVEAUX-VISIBLES) + (cor(lname))+;
NOM               = cor(NOM-CELLULE) + ent(texte);
ÉCHELLE           = cor(ECHELLE) + ent(number);
N.H.              = cor(NESTING-LEVELS) + ent(entier);
GRILLE            = cor(GRILLE);
ESPACE-GRILLE     = cor(ESPACE-GRILLE) + ent(xsp,ysp);
CURSEUR1          = cor(pt) + cor(CR1) / cor(pt) + ent('1')
                    / cor(CR1) + cor(CUR1) + ent(point);
CURSEUR2          = cor(pt) + cor(CR2) / cor(pt) + ent('1')
                    / cor(CR2) + cor(CUR2) + ent(point);
COIN__LL          = cor(pt) + cor(LL-C) / cor(pt) + ent('l')
                    / cor(LL__C) + cor(LL-C) + ent(point);

texte             = (caractère)+;
nom-cellule       = texte;
transform         = entier1;
rel-cur1,cadre    = entier2;
```

```
nbrepx          = entier;
nbrepy          = entier;
espx,xsp        = entier;
espy,ysp        = entier;
mirry           = '1' / '0';
mirrx           = '1' / '0';
point           = (entier,entier);'
entier          = 0/../9;
entier1         = 1/../8;
entier2         = 1/../4;
```

Les pages suivantes proposent quelques exemples d'édition de structures géométriques réalisées à l'aide de cet éditeur.

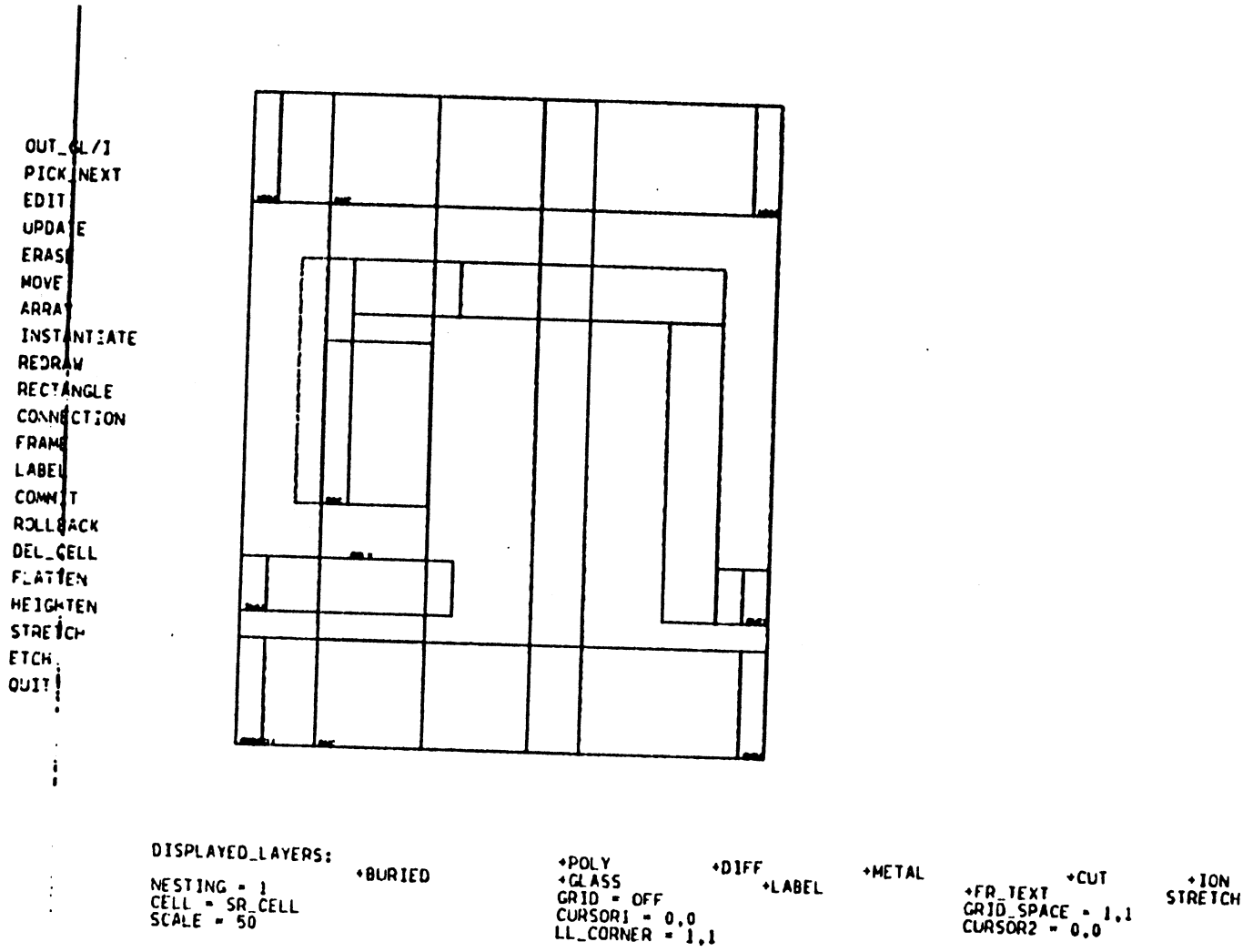


Figure A.31: Edition d'une cellule de base de registre à décalage: SR-CELL

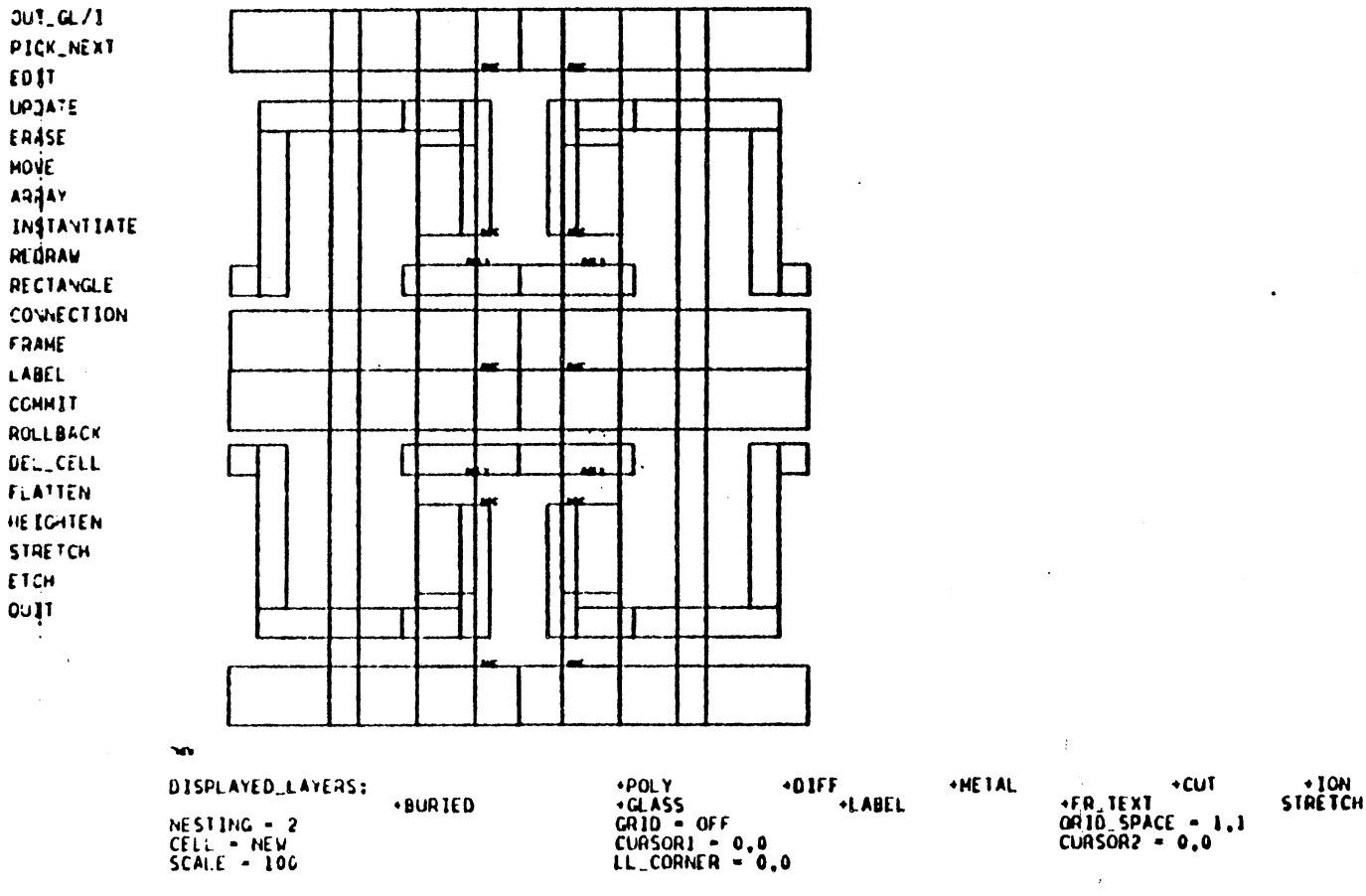


Figure A.32: Réalisation d'une matrice(2,2) M, utilisant comme cellule de base SR-CELL.

SR-CELL a été pivotée de 180 degrés et les options mirrx et mirry ont été positionnées à '1'.

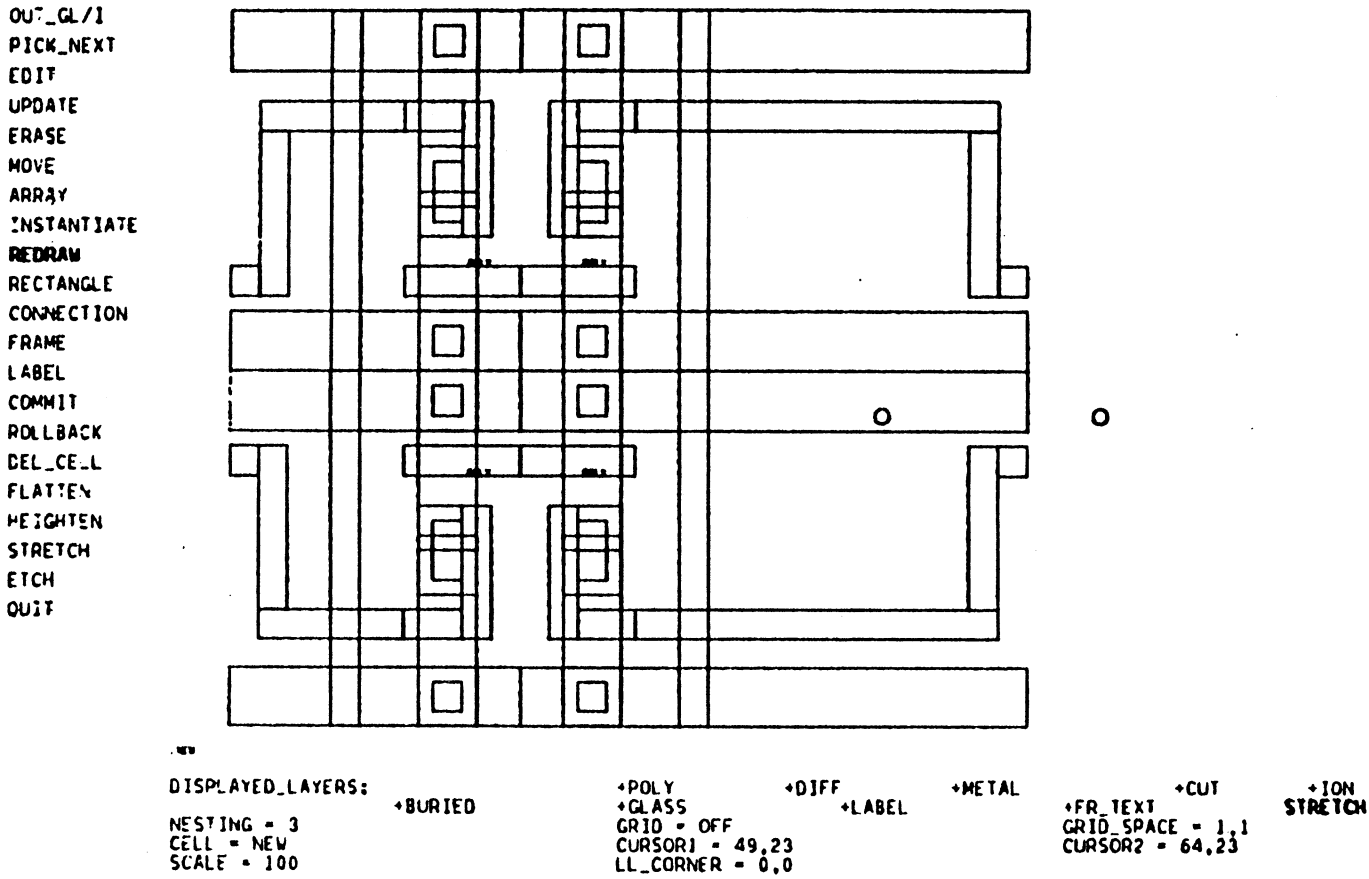


Figure A.33: Représentation de l'opération d'étirement sur cette matrice

De manière à pouvoir réaliser cette opération, la fonction 'Passage de Référence à Copie' exposée à la page 109 , a été effectuée (les références aux deux copies de SR-CELL modifiées ont été remplacées par deux copies complètes de SR-CELL dans la structure hiérarchique de M).

ANNEXE 4: INTERPRÉTEUR DE GL/1

Cette quatrième annexe donne quelques rènseignements supplémentaires sur la structure de cet interpréteur.

Il est organisé en trois modules fonctionnellement indépendants:

(1) un interpréteur des structures de données rangées en mémoire locale de l'éditeur,
(2) un formateur transformant chaque type de données en une instruction GL/1,
(3) un translateur transformant ces instructions en code binaire et les écrivant sur fichier.

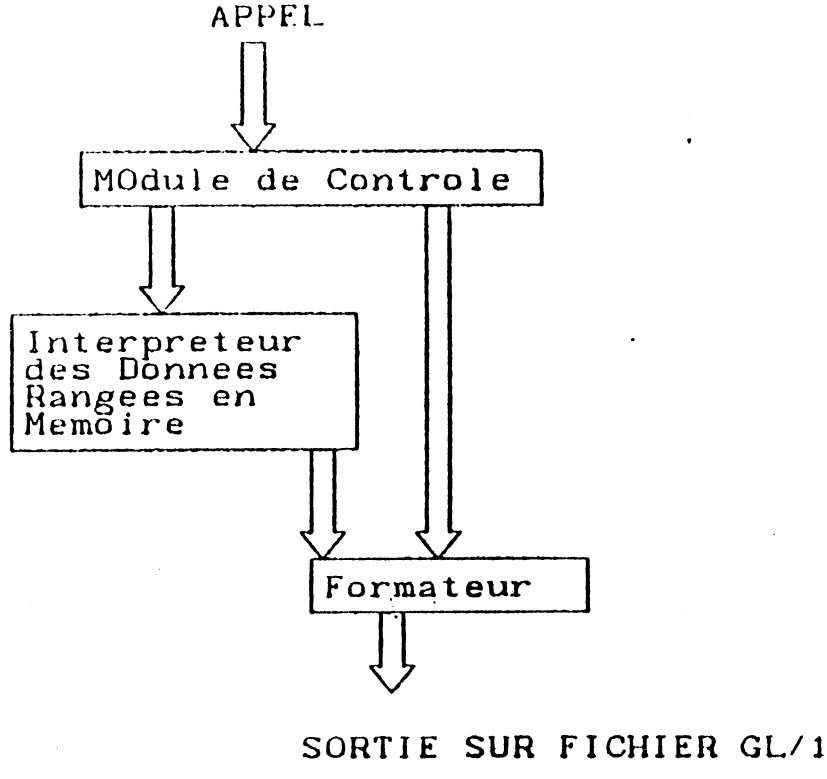


Figure A4: Représentation Modulaire du l'interpréteur de GL/1

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de Messieurs

. F. ANCEAU, Professeur

. R. GERBER, Responsable RCA au CNET Meylan

Mademoiselle ETIENNE Régine

est autorisée à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Informatique".

Fait à Grenoble, le 13 juin 1983

Le Président de l'I.N.P.-G.

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

P.O. le Vice-Président,