THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR INGENIEUR

par

Altamiro Amadeu SUZIM

ETUDE DES PARTIES OPERATIVES A ELEMENTS MODULAIRES POUR PROCESSEURS MONOLITHIQUES.

Thèse soutenue le 6 novembre 1981 devant la Commission d'Examen :

Monsieur	L. BOLLIET	: Président
Messieurs	F. ANCEAU C. TRULLEMANS J.-L. LARDY M. VERGNIAULT	Examinateurs

Microeletrônica - SBC/II

Projeto: Circuitos integrados

CNPq 3.04.03 00-6

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

Vice-Présidents : M. Georges LESPINARD
M. René PAUTHENET

PROFESSEURS DES UNIVERSITES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	BENOIT Jean	Radioélectricité
	BESSON Jean	Chimie Minérale
	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du Solide - Cristallographie
	BOIS Philippe	Mécanique
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUVARD Maurice	Génie Mécanique
	BRISSONNEAU Pierre	Physique des Matériaux
	BUYLE-BODIN Maurice	Electronique
	CHARTIER Germain	Electronique
	CHERADAME Hervé	Chimie Physique Macromoléculaires
Mme	CHERUY Arlette	Automatique
MM.	CHIAVERINA Jean	Biologie, Biochimie, Agronomie
	COHEN Joseph	Electronique
	COUMES André	Electronique
	DURAND Francis	Métallurgie
	DURAND Jean-Louis	Physique Nucléaire et Corpusculaire
	FELICI Noël	Electrotechnique
	FOULARD Claude	Automatique
	GUYOT Pierre	Métallurgie Physique
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	LACOUME Jean-Louis	Géographie - Traitement du Signal
	LANCIA Roland	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	LESPINARD Georges	Mécanique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	MOREAU René	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PARIAUD Jean-Charles	Chimie - Physique
	PAUTHENET René	Physique du Solide - Cristallographie
	PERRET René	Automatique

.../...

MM.	PERRET Robert	Electrotechnique
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	POUPOT Christian	Electronique - Automatique
	RAMEAU Jean-Jacques	Chimie
	ROBERT André	Chimie Appliquée et des matériaux
	ROBERT François	Analyse numérique
	SABONNADIERE Jean-Claude	Electrotechnique
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie - Physique
Mme	SCHLENKER Claire	Physique du Solide - Cristallographie
MM.	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahïm	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KAMARINOS Georges	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)

E.N.S.E.E.G.

MM.	ALLIBERT Michel	
	BERNARD Claude	
	CAILLET Marcel	
Mme	CHATILLON Catherine	
MM.	COULON Michel	
	HAMMOU Abdelkader	
	JOUD Jean-Charles	
	RAVAINE Denis	
	SAINFORT	C.E.N.G.

.../...

MM. SARRAZIN Pierre
SOUQUET Jean-Louis
TOUZAIN Philippe
URBAIN Georges — Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.E.E.

MM. BISCONDI Michel
BOOS Jean-Yves
GUILHOT Bernard
KOBILANSKI André
LALAUZE René
LANCELOT Françis
LE COZE Jean
LESBATS Pierre
SOUSTELLE Michel
THEVENOT François
THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

E.N.S.E.R.G.

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

E.N.S.I.E.G.

MM. BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
JAUSSAUD Pierre
Mme JOURDAIN Geneviève
MM. LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

M. DELHAYE Jean-Marc

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

MM. COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

A Neli, Daniela et Carolina

qui l'ont vécu avec moi.

Je tiens à remercier,

Monsieur Louis BOLLIET, professeur à l'Université de Grenoble, de m'avoir fait l'honneur de présider ce jury,

Monsieur François ANCEAU, professeur à l'INPG-ENSIMAG, qui m'a orienté tout au long de ce travail,

Monsieur Jean-Louis LARDY, Monsieur Charles TRULLEMANS et Monsieur Michel VERGNIAULT d'avoir accepté de faire partie du jury ainsi que pour les observations qu'ils m'ont faites,

Monsieur Jean-Pierre SCHOELLKOPF pour ses conseils lors de la révision du manuscrit et des articles publiés,

Madame Hélène DIAZ pour sa dactylographie intelligente,

Monsieur Christian BERNARD d'avoir joué l'expérience de l'utilisation de ce système,

Monsieur Jacques RAYMOND pour sa collaboration dans les opérations du système graphique,

Tous les membres de l'équipe qui m'ont fait part de leur amitié,

L'équipe de reprographie de l'IMAG où j'ai trouvé la sympathie et la compétence,

Les organisations suivantes, dont l'appui a rendu possible ce travail :

- *l'Universidade Federal do Rio Grande do Sul (UFRGS) Brésil, et en particulier le Département d'Ingéniérie Electrique,*
- *le Conselho Nacional de Desenvolvimiento Científico e Technológico, CNPQ, (Brésil),*
- *l'Institut National Polytechnique de Grenoble, INPG,*
- *le Centre Régional des Oeuvres universitaires, CROUS,*
- *le Curso de Pós-Graduação em Ciência da Computação, CPGCC-UFRGS,*

Mes parents et amis qui ont été présents malgré la distance et tous ceux qui par leur amitié et leur sympathie, ont rendu plus agréable et enrichissant notre séjour en France.

TABLE DES MATIERES

Introduction 1

1 - INTRODUCTION

Un circuit intégré monolithique est un assemblage de composants électroniques fabriqués et reliés sur une seule tranche de silicium.

Il existe plusieurs technologies qui permettent la réalisation de tels circuits. Elles ont en commun le fait que chaque étape du processus de fabrication est appliquée simultanément à tous les éléments de la même nature sans influencer ceux de nature différente. Cela implique que l'ensemble des opérations de fabrication requises est indépendant du nombre de composants réalisés.

Les circuits logiques complexes voient leur coût se réduire avec l'amélioration des techniques d'intégration. En fait, le coût de fabrication par porte logique est presque inversement proportionnel à la densité d'intégration. Cela suscite un effort très important de la part des constructeurs pour augmenter cette densité. On est passé, en une vingtaine d'années, de quelques portes à des milliers de portes par circuit. Il y a un formidable progrés technique.

Evidemment, cet ensemble de portes doit être organisé pour exécuter la fonction désirée: c'est ce qu'on appelle la conception du circuit. Le travail nécessaire à la conception d'un circuit n'est pas proportionnel au nombre de portes, mais à sa complexité logique (mesurée, par exemple, par le nombre d'états et la complexité du vocabulaire d'un automate équivalent).

La microélectronique est une activité multi-disciplinaire. Le schéma de la figure 1.1. [WEI 81] montre les principales étapes suivies par un produit dans une industrie.

Il n'y a rien d'extraordinaire dans cette séquence d'activités telle qu'elle est présentée dans le graphe. Elle s'applique à beaucoup d'autres prduits industriels. Ce qui est remarquable, par contre, c'est l'interaction accentuée

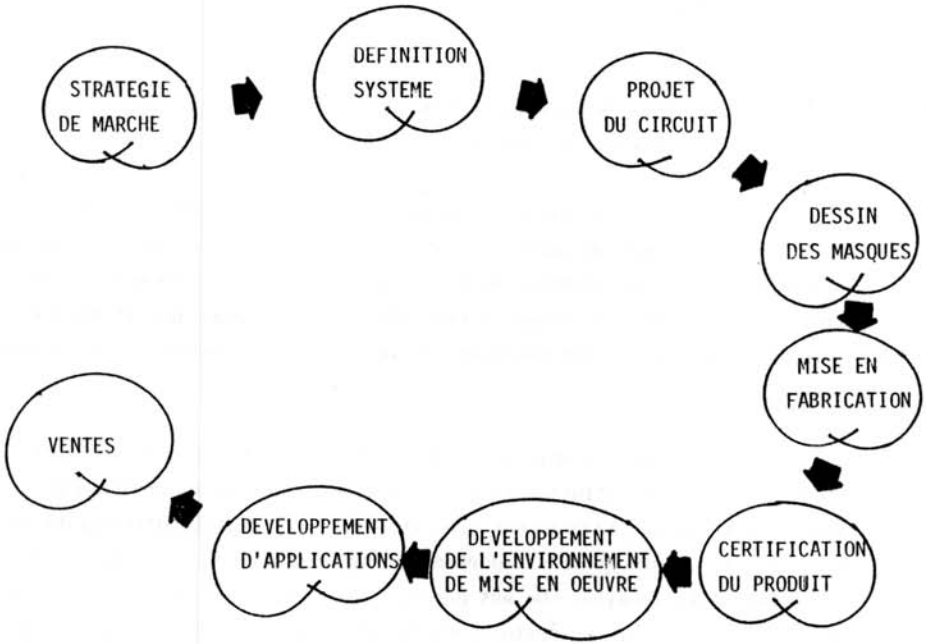


Figure .1. - Principales étapes de l'élaboration d'un circuit intégré.

entre les différentes étapes et le niveau d'automatisation qu'on peut y appliquer. L'interaction rend le graphe qu'on a présenté beaucoup plus complexe : presque toutes les activités sont reliées entre elles par des liens bidirectionnels.

L'automatisation s'applique à plusieurs étapes du processus d'élaboration d'un circuit intégré. Il y a par xemple :

- l'automatisation dans la phase de dessin des masques: on utilise des systèmes graphiques pour la description, l'assemblage, les corrections ;
- la fabrication automatique des masques à partir du dessin, par des machines pilotées par ordinateur ;
- le test automatique des circuits etc...

Ce travail se situe dans la phase "dessin des masques". Pour des circuits d'une densité importante (au-dessus 1000 portes logiques), des dizaines de milliers de motifs graphiques doivent être dessinés. Le dessin du circuit est une tâche qui exige beaucoup de travail (sauf pour des circuits très répétitifs comme les mémoires, grâce aux outils de système graphique).

Plusieurs techniques sont actuellement utilisées pour raccourcir l'effort de dessin. Ces techniques (comme le dessin symbolique, les bibliothèques de cellules, les réseaux prédiffusés, etc...) sont efficaces au niveau du temps de dessin, mais ont des domaines d'utilisation spécifiques à cause des contraintes imposées au concepteur.

Pour la VLSI, de nouvelles méthodes de conception de circuits sont nécessaires. Les circuits ne sont plus vus comme un ensemble de portes logiques mais plutôt comme une structuration de blocs fonctionnels.

L'approche choisie consiste à traiter, non un circuit particulier mais une classe de circuits. Cette approche est basée sur une décomposition fonctionnelle des machines en deux parties: partie opérative et partie contrôle [ANC 76]. La stratégie présentée a une influence sur d'autres phases de la conception des circuits, en particulier sur le projet du circuit. Un modèle de machine est offert au concepteur. Il doit orienter la conception de son circuit dans le sens de l'utilisation de blocs fonctionnels dont la structure est pré-définie. Des outils spécialisés qui permettent une réalisation automatique optimisée basée sur les caractéristiques particulières de chaque bloc sont développés.

La partie contrôle est réalisée essentiellement par des PLAs et des ROMs dont la régularité est exploitée par les outils de génération automatique. Ces outils acceptent des restrictions de forme dans le but de générer des blocs qui s'assemblent harmonieusement.

La partie opérative, but spécifique de ce travail, est construite à partir de cellules fonctionnelles. Les idées de base de l'outil de génération des parties opératives sont les suivantes :

- Tranche de bit (bit-slice)

L'idée de tranche de bit découle naturellement du concept de "mot" qui est un vecteur de bits, concept largement utilisé en informatique. Dans le domaine de la conception des circuits intégrés, cette idée est renforcée, en ajoutant à son sens fonctionnel un sens topologique : une partie opérative de n bits (qui traite des mots de n bits) sera construite par empilement de n tranches de un bit. Un examen de l'architecture interne de quelques microprocesseurs montre une évolution dans le sens de l'utilisation de cette technique qui se prête bien à l'automatisation.

- Assemblage de cellules fonctionnelles

Un ensemble de fonctions de base de la partie opérative est déduit à partir d'un modèle, développé dans le texte. Un ensemble de cellules est ensuite dessiné. Chaque cellule est conçue pour exécuter une fonction spécifique. La partie opérative est construite par assemblage sélectif de certaines cellules.

- Conception orientée vers un bus

Les cellules sont dessinées en fonction d'une structure pré-définie de bus, c'est-à-dire qu'une structure de bus est choisie avant de dessiner les cellules. Les positions des lignes d'alimentation sont fixées a priori et toutes les cellules doivent respecter les conventions de position des lignes

de bus et alimentation. Ainsi, au moment de l'assemblage, il n'y a pas de connexions à faire: elles sont traitées en même temps que le placement des cellules. Cette technique permet de travailler avec des cellules rigides qui ont été dessinées avec le souci d'optimiser l'utilisation de surface pour chaque cellule. Une préoccupation majeure est de ne pas dégrader la densité pendant l'assemblage : les solutions retenues seront présentées dans la thèse. En particulier, les cellules sont toutes de la même hauteur (ou d'une hauteur multiple de celle du module de base). Contrairement à la technique présentée dans [JOH 79], il n'est pas nécessaire de modifier la taille des cellules, ce qui dégrade manifestement la densité puisque des espaces vides sont introduits quand on étire les cellules. Notre but est d'offrir des moyens de conception automatique tout en gardant une densité proche de celle obtenue par le dessin manuel.

C'est toute une discipline de conception qui est en jeu. Ce travail en est l'un des modules. D'autres travaux de l'équipe de recherche en Architecture des Ordinateurs détaillent les aspects complémentaires: l'étude de la topologie globale du circuit [REI 81], la réalisation de la partie contrôle [PER 80], [OBR 81], test [COU 81] etc...

Une vue d'ensemble de la démarche présentée est donnée par le schéma de la figure 1.2 En partant du modèle de la partie opérative, l'ensemble des primitives qu'on se propose de réaliser est défini. A chaque primitive, on fait correspondre une cellule (la réalisation de la primitive) que l'on a dessinée et stockée dans une bibliothèque du système graphique. Une description de haut niveau est interprétée par une programme qui la transforme en une séquence d'appels de cellules. Un programme de niveau intermédiaire calcule ensuite la position de ces cellules et génère un texte dans le langage graphique.

Ce système est fait de telle façon qu'il puisse évoluer au fur et à mesure que des circuits sont réalisés. On part d'une structure de bus sous laquelle

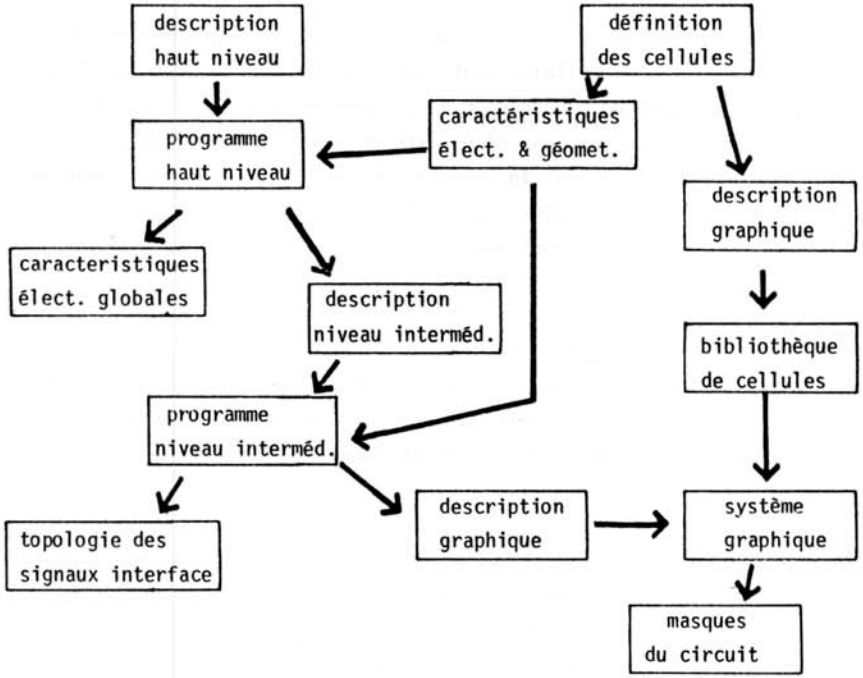


Fig 1.2 Vue du Système

sont dessinées les cellules de base et les cellules spéciales nécessitées par une certaine application. Pour chaque nouvelle application, il suffira de dessiner les cellules qui n'existent pas encore. A mesure que le système évolue, la probabilité de trouver des cellules présentes dans la bibliothèque augmente. Ceci amène le concepteur à chercher à résoudre son problème avec les primitives disponibles dans le système et à y insérer, s'il en a vraiment le besoin, une nouvelle primitive.

On présentera donc :

1 - La méthodologie utilisée, le modèle de machine et les primitives de base nécessaires pour construire les parties opératives.
La description d'un algorithme, moyennant l'utilisation d'un ensemble de primitives ad hoc, est le point de départ de la démarche. Compte tenu des restrictions matérielles, les primitives complexes ne seront pas réalisées directement au niveau matériel mais par un algorithme d'interprétation qui n'utilise que des primitives plus simples. Un modèle de machine, extrait d'observations de circuits existants, précise les primitives qu'on se propose de réaliser. Ce modèle s'appuie sur l'idée de tranches de bit.

2 - La réalisation des primitives consiste à définir un ensemble de cellules fonctionnelles. Chaque cellule réalise une fonction spécifique qui correspond à la réalisation d'une primitive. Les cellules sont conçues de telle façon qu'elles puissent travailler ensemble: leur assemblage constitue, en fait, l'activité de génération d'une partie opérative. Une étude du comportement électrique est effectuée pour certifier que leur fonctionnement est correct du point de vue électrique.

3 - Le dessin du jeu de masques destiné à la fabrication du circuit intégré est le but final du travail. Ce dessin est le résultat de l'assemblage de cellules prédessinées et stockées dans une bibliothèque du système informatique. Les problèmes de topologie et d'assemblage sont donc étudiés. Un outil est ensuite présenté, qui permet l'assemblage de ces cellules à partir d'une

description de haut niveau fournie par le concepteur. Dans le but d'isoler les aspects fonctionnels de l'aspect purement graphique, l'outil est décomposé en deux parties :

- un programme dit de haut niveau qui traduit un appel de fonction (primitive) en une séquence d'appels de briques (dessin des cellules) ;
- un programme dit de niveau intermédiaire qui calcule l'emplacement des briques et génère un texte dans le langage graphique.

Un ensemble de quelques cellules dessinées est présenté dans l'annexe, ainsi que leur assemblage en vue de réaliser la partie opérative d'un circuit contrôleur de communications, en cours d'étude en coopération avec la société EFCIS.

Méthodologie 2

2 - METHODOLOGIE

2.1. INTRODUCTION

Dans ce chapitre on étudie la méthodologie générale utilisée pour réaliser des circuits logiques décrits par leur comportement. On traite ici deux aspects :

- La méthodologie de base, où une machine séquentielle est assimilée à la réalisation d'un algorithme. A partir de la description de la machine par un algorithme, on descend jusqu'à sa réalisation matérielle en appliquant des opérations d'interprétation selon la méthode descendante [SCH77]. On décompose l'algorithme en deux parties: le séquencement et les actions. A ces deux parties, on fait correspondre deux parties de la machine: la partie contrôle qui réalise le séquencement et la partie opérative qui réalise les actions.

- La modélisation de la partie opérative où l'on présente un modèle pour en déduire un ensemble de primitives. La réalisation de ces primitives jusqu'au niveau des masques est le but de ce travail et on l'étudiera en détail. Les primitives correspondent à des ressources d'action mises à disposition du concepteur qui doit créer une version matérielle d'un algorithme. La nature de ces primitives est générale et elles sont donc utilisables dans tous les algorithmes. Toutefois, leur organisation varie selon les besoins du concepteur. La réutilisation de ces primitives dans sa forme finale (le dessin) est un atout important de cette méthode, vu l'effort considérable qu'il est nécessaire de dépenser pour les obtenir.

2.2. LES ALGORITHMES ET LA STRUCTURE INTERPRETATIVE

Le comportement d'une machine séquencielle peut être décrit sous la forme d'un algorithme A. L'expression de A dans un langage L, représentée A/L est un programme P du langage L.

Le même algorithme peut être réalisé sur du matériel. La réalisation de A sur un matériel M, représentée A//M est une machine M.

Un algorithme peut avoir différentes formes, aussi bien dans sa réalisation matérielle que dans son expression graphique (programmes dans différents langages, organigrammes, etc...). Ces différentes formes sont représentées dans la figure 2.1 où P_1, P_2, P_3 ... sont les différents programmes (ou expressions graphiques), M_1, M_2, M_3 ... sont les différentes machines. L_1, L_2... sont des langages et M_1, M_2, M_3 sont différentes organisations matérielles.

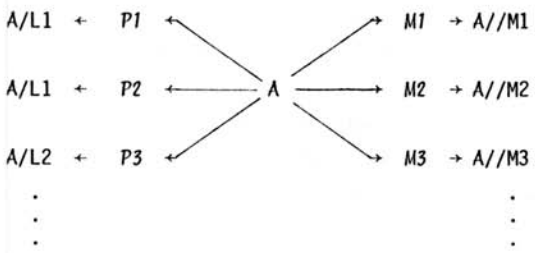


Figure 2.1 - Différentes réalisations de l'algorithme A .

Les langages dans lesquels est exprimé l'algorithme A constituent des ensembles possibles de ressources. Les ressources d'un langage sont dites des primitives. Les ressources matérielles sont appelées composants.

Notre but est la construction de machines informatiques, c'est-à-dire, la réalisation matérielle d'algorithmes. Dans le domaine des circuits intégrés, les machines sont réalisées par une séquence de traitement d'un matériau où sont définies les formes géométriques qui représentent les composantes à l'aide d'un jeu de masques. Les masques sont faits par des machines pilotées par des ordinateurs et il faut donc concevoir le circuit, dessiner ses masques et les rentrer dans l'ordinateur: c'est la conception du circuit.

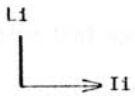
L'activité "circuits intégrés" se prête à l'automatisation à différents niveaux et dans plusieurs domaines (comme la simulation, la saisie des données des masques, le dessin, le test, ...). On s'intéresse ici en particulier à la

conception du circuit jusqu'au niveau des masques. L'approche choisie est de considérer les circuits, non pas chacun comme un cas à part, mais comme différentes réalisations d'une même structure de base ; un peu comme différents programmes écrits dans un même langage (tout langage n'est pas optimal pour tous les programmes comme cette structure de base n'est pas la meilleure pour tous les circuits). Les circuits visés sont les circuits qui réalisent des algorithmes complexes (évalué par le nombre d'états d'un automate de Moore équivalent, ayant un vocabulaire d'entrée non trivial).

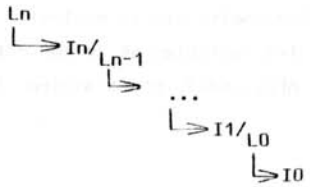
Un circuit dont l'algorithme est exprimé par un programme peut être réalisé directement à partir de ce programme. Si les primitives du langage utilisé sont réalisables sur du matériel, alors une transposition directe est possible.

Si les primitives du langage utilisé sont trop complexes et que leur réalisation directe sur du matériel n'est pas possible (ou n'est pas intéressante), on va procéder à des descriptions de plus en plus fines, en réduisant à chaque niveau la complexité des primitives utilisées. Le dernier niveau atteint ne doit donc avoir que des primitives réalisables sur du matériel.

Un interpréteur Ii du langage Li est un algorithme capable d'exécuter tout programme écrit dans ce langage. Le processus d'interprétation est noté :



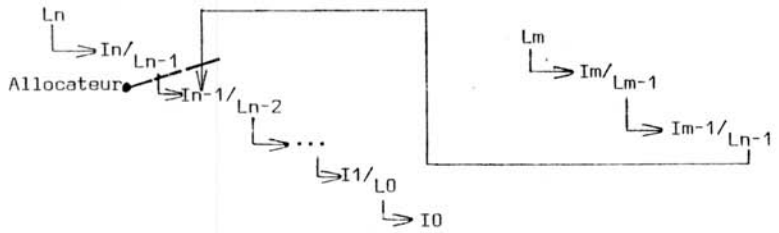
L'interpréteur lui-même, peut être programmé dans un langage Li-1. Le processus d'interprétation sera répété, s'il le faut, jusqu'à atteindre le niveau de langage désiré. Le résultat est la chaîne :



IO sera réalisé de manière matérielle. Sa réalisation est la machine M capable d'exécuter, indirectement, les instructions de Ln via les interprétations imbriquées.

La structure interprétative permet un isolement entre les différentes couches de langage. Il est évident que l'indépendance entre les couches est d'autant plus grande que ces couches sont plus éloignées. Il en découle une grande indépendance entre les caractéristiques externes (spécification au niveau supérieur) et la réalisation interne des machines, spécialement pour les plus complexes, c'est-à-dire, celles qui contiennent plusieurs niveaux d'interprétation.

On peut avoir plusieurs couches externes réalisées avec la même machine physique, au prix d'allocateur, comme on le voit dans le schéma suivant :



L'allocateur sert à gérer le partage In-1 entre les deux programmes qu'il a à interpréter.

Les primitives d'un langage sont constituées des ensembles d'opérateurs et de variables.

Exemple :

Soit le microprocesseur 6800. Prenons seulement l'algorithme d'interprétation des instructions (interruption, halt, etc... exclus). Les primitives dont dispose le programmeur sont les variables et le jeu d'instructions. Les variables sont A, B, X, PC, SP, CC et M[0...64K], c'est-à-dire, les registres et la mémoire.

Remarque : Dans le jeu d'instructions mentionné ci-dessus, sont comprises les instructions qui exécutent des actions sur les variables et celles qui contrôlent le flux de l'exécution.

Un algorithme transforme un ensemble de données (entrées) dans un autre ensemble de données (sorties). On peut représenter cette structure par le schéma de la figure 2.2.a, ou, plus synthétiquement, par celui de 2.2.b. Dans ce cas, les variables sont internes et représentent ce dont a besoin un algorithme pour réaliser ses opérations.

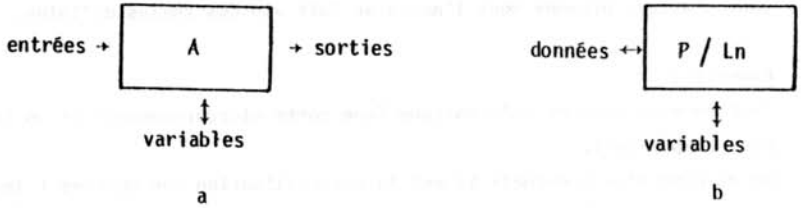


Figure 2.2-Un algorithme et son environnement

L'algorithme, exprimé dans un langage Ln est noté A/Ln. Un interpréteur de Ln doit accéder au programme écrit en Ln aussi bien qu'aux variables et données qu'il manipule. Le programme, les variables et les données sont donc des données pour l'interpréteur. L'interpréteur utilisera en plus, des variables de travail internes. Le phénomène se répète tout au long des couches d'interprétation, ce qu'on schématise dans la figure 2.3.

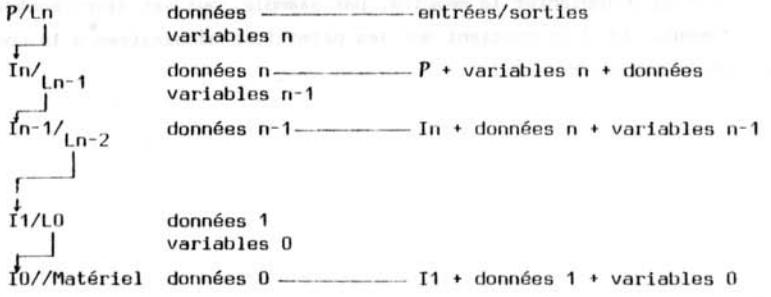


Figure 2.3. - Enchaînement des niveaux d'interprétation

L0 est un langage de description de matériel et I0 est un interpréteur physique réalisé sur du matériel. I0 est donc figé. I0 n'interprète pas tous les programmes écrits en L0 mais un seul.

Chaque niveau d'interprétation correspond à un langage dont les primitives sont plus élémentaires, qui permet donc une description plus fine des actions. En parallèle on considère une hiérarchie des données pour n'implanter sur le matériel de I0 qu'un petit sous-ensemble des données traitées. Les limitations du matériel sont connues du concepteur et il peut avoir une idée de ce qu'il est possible d'implanter. Les autres données sont implantées dans d'autres organes dont l'accès se fait via des règles définies.

Exemple :
Supposons un système informatique (une carte microprocesseur et un terminal, pour simplifier).
On dispose d'un terminal: il est la matérialisation des données (les entrées/sorties).
Le programme est écrit dans Ln. L'algorithme In qui interprète Ln accède aux données du programme via des règles d'accés. Il en résulte que Ln-1 dans lequel est écrit In ne contient pas l'ensemble des données mais seulement les primitives qui permettent de construire les opérations d'accés aux données (figure 2.4). On implémente de la même façon les boutons "marche/arrêt" et la lampe qui indique l'état "arrêté".
Si on descend d'un cran, on considère le microprocesseur lui-même. In-1 qui interprète Ln-1 est décrit dans le langage Ln-2. A ce niveau, on peut décider d'implanter la mémoire, par exemple, qui est alors accédée indirectement. Ln-2 ne contient que les primitives nécessaires à la construction du mécanisme d'accés.

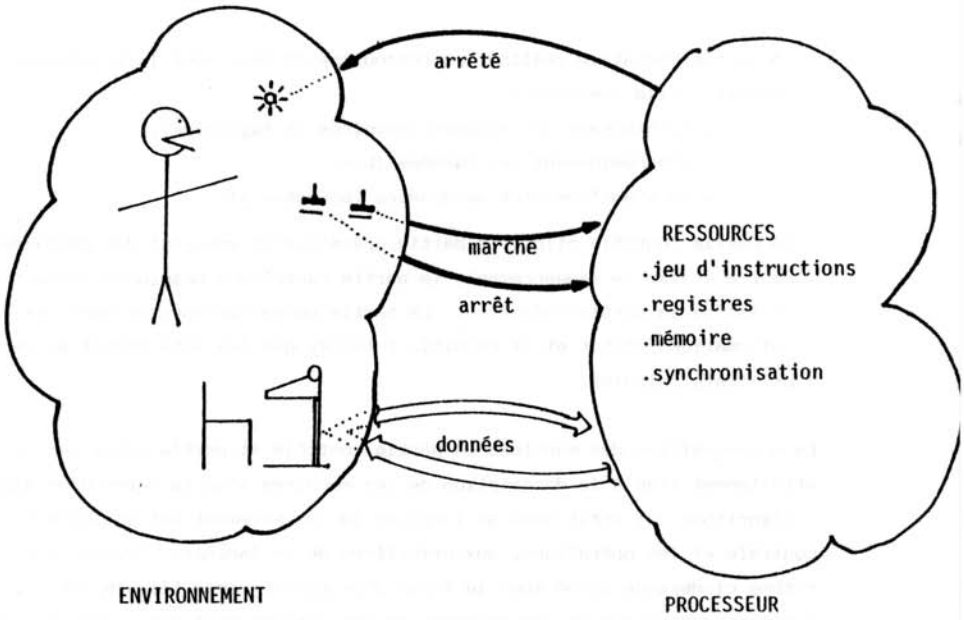


Figure 2.4 - Exemple d'implantation: les périphériques

Ce travail se restreint à la réalisation d'une partie des ressources de LO, en matériel: un sous-ensemble des données O et un sous-ensemble des actions. On étudiera par la suite une décomposition des circuits pour préciser les éléments qu'on se propose de réaliser.

2.3. NOTION DE PARTIE OPERATIVE ET PARTIE CONTROLE

Les machines séquentielles en général et les processeurs en particulier, peuvent être décomposés en deux parties: une partie contrôle et une partie opérative. Cette décomposition est faite, non seulement du point de vue

logique, mais aussi, souvent, du point de vue topologique.

- La partie opérative réalise les ressources de base pour le traitement des données. Elle comprend :
  - . le stockage des données (mémoires et registres),
  - . l'acheminement des données (bus),
  - . la transformation de données (opérateurs).
- La partie contrôle pilote la partie opérative en envoyant des commandes. Pour exécuter le séquencement, la partie contrôle a besoin de connaître l'état de la partie opérative. La partie opérative renvoie donc des informations d'état et de conditions telles que les mots d'état et des prédicats calculés.

La décomposition des machines en partie contrôle et partie opérative est étroitement liée à la description de ces machines sous la forme d'un algorithme. L'algorithme est écrit dans un langage: le séquencement est assimilé au contrôle et les opérations, aux primitives de ce langage. Supposons l'algorithme ci-dessous donné sous la forme d'un organigramme (figure 2.5.a). On présente séparément le séquencement et les actions pour dissocier les composants d'opérations de ceux de contrôle.

L'algorithme démarre avec le signal "début" en exécutant l'action 1.
Les actions 2, 3, 4, 1, 2, ... sont ensuite exécutées séquentiellement.
Pendant l'action 2, la partie contrôle analyse le prédicat "A = 0" fournie par la partie opérative. Si (A=0) est VRAI, elle s'arrête en envoyant le signal "fin", sinon, l'action 3 est activée.

La communication avec l'environnement se fait par les données et par les signaux "début" et "fin".

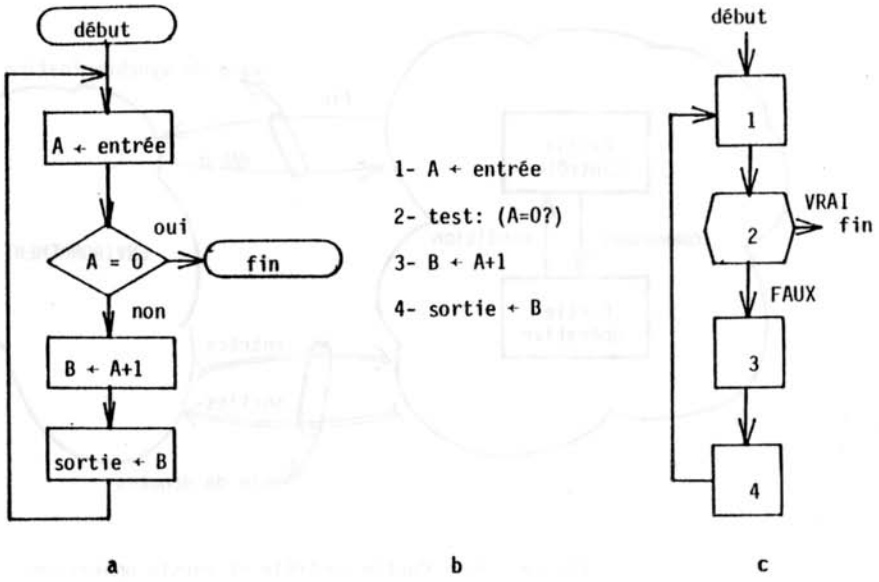


Figure 2.5. - Séquencement et actions dans un algorithme
a) Organigramme, b) Liste d'actions , c) Séquencement.

Dans la figure 2.6 on schématise ces différents éléments: la partie opérative, la partie contrôle et la communication avec l'environnement. La communication se fait par deux voies dites voie de données et voie de synchronisation. Les commandes sont : addition, affectation, etc... et la condition est le résultat du test (A=0).

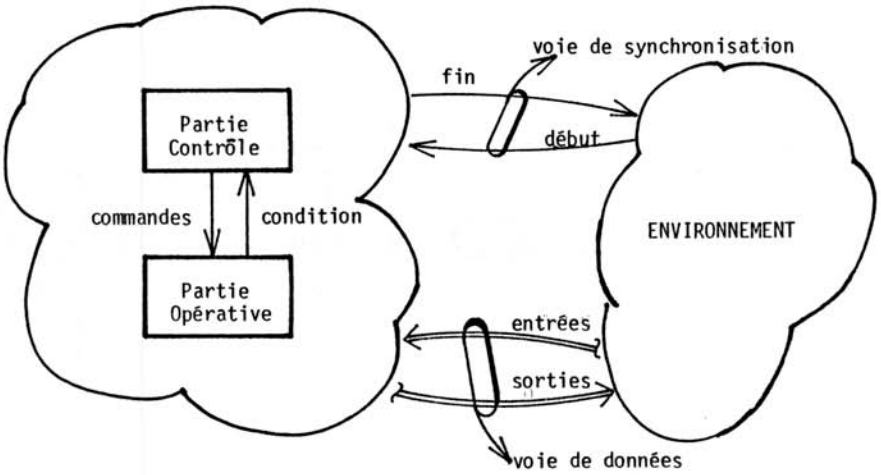


Figure 2.6. - Partie contrôle et partie opérative.

Si on applique cette décomposition à la structure interprétative présentée auparavant, on voit que les parties contrôle s'empilent, ce qui peut être représenté comme dans la figure 2.7. PC est une partie contrôle et PO est une partie opérative. Les indices sont les niveaux et les apostrophes distinguent les éléments d'un même niveau. Un double rectangle représente une partie opérative implantée.

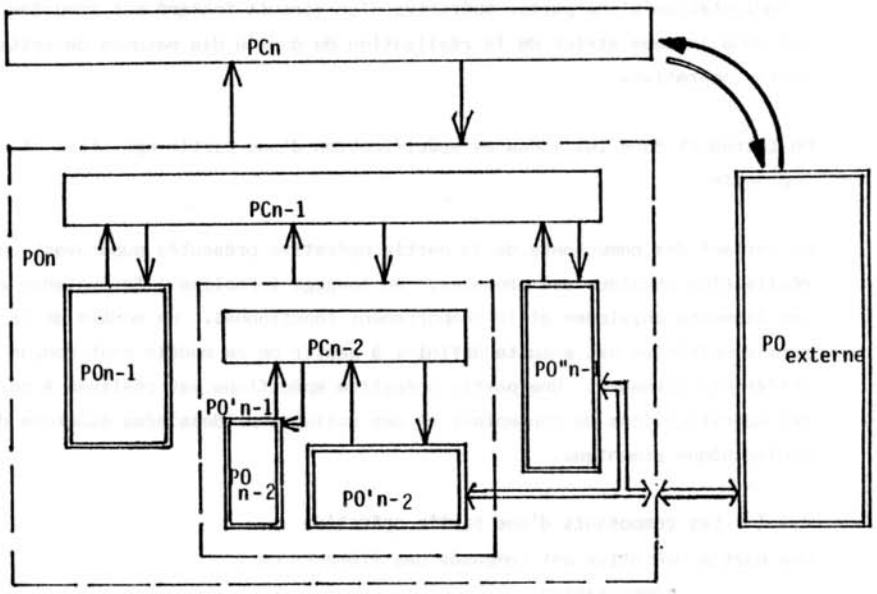


Figure 2.7. - Partie contrôle et partie opérative dans la structure interprétative.

Par PO externe on veut représenter un élément qui n'est pas synchrone avec PCn, comme un périphérique dans l'exemple de la figure 2.4 ; il est considéré comme une boîte noire de laquelle on ne connaît que les entrées/sorties (les voies de synchronisation et de données).
Dans cette même boîte noire la décomposition en niveaux d'interprétation peut se poursuivre.

2.4. IMPLANTATION DE LA PARTIE OPERATIVE

L'implantation d'une partie opérative d'un circuit intégré est comprise ici dans le sens strict de la réalisation du dessin des masques de cette partie opérative.

On introduit donc les idées de spécification d'une partie opérative et de topologie.

En partant des composants de la partie opérative présentés auparavant, une réalisation physique est proposée. Un langage formalise l'équivalence entre ces éléments physiques et le comportement fonctionnel. Un modèle de la partie opérative est ensuite défini ; à partir de ce modèle sont conçus les différents éléments. Une partie opérative spécifique est réalisée à partir des spécifications du concepteur et des cellules prédessinées stockées dans une bibliothèque graphique.

2.4.1. Les composants d'une partie opérative

Une partie opérative est composée des éléments de :

- mémorisation,
- communication,
- transformation de données.

2.4.1.1. La mémorisation est la réalisation d'un espace d'adressage. Comme on est au niveau de la machine physique, cet espace d'adressage représente l'implantation des données de LO. Cet espace d'adressage n'est pas homogène, c'est-à-dire qu'il est composé d'objets dont le comportement n'est pas identique. On les appelles tous mémoires parceque la fonction physique qu'ils remplissent est une mémorisation. Cette notion recouvre le stockage

- des constantes (lecture seule),
- des variables banalisées (lecture/écriture),
- des variables spécialisées (lecture/écriture),
- des mots de commandes et d'état (écriture seule).

a/ Les constantes sont rangées dans des mémoires à lecture seule. Ce sont des sources d'informations qui ont été enregistrées au moment de la conception du circuit.

b/ Les variables banalisées sont implantées comme des mémoires vives. Il est possible de lire et écrire ces variables et leur valeur initiale n'est pas définie.

c/ Les variables spécialisées sont semblables aux variables banalisées mais possèdent une propriété de plus: elles peuvent être chargées par la sortie d'une fonction.

Remarque : Dans la pratique, il arrive souvent que l'écriture dans les variables spécialisées soit restreinte au chargement d'une valeur calculée, c'est-à-dire à la sortie d'une fonction. C'est une simplification liée à la nature du problème traité puisqu'il s'avère que l'écriture à partir d'une autre source n'est pas utilisée. Du point de vue de la conception du circuit ceci représente une optimisation car il y a une commande de moins à générer.

2.4.1.2. Communication

Pour que les variables puissent communiquer entre elles il faut qu'il existe un chemin de données les reliant. En général, il n'est pas possible, ou il n'est pas économique, de relier chaque variable à toutes les autres. On construit donc des chemins qui sont partagés par les variables dont l'accés est contrôlé par des TMOS interrupteurs que l'on peut commander. Ces chemins sont appelés des bus. Ils représentent des variables instantanées de LO puisque leur valeur n'est valide qu'à la condition que la connexion soit active. Ils sont en fait des variables de IO (l'interpréteur du langage de phases) ; la phase est indivisible et la variable n'est donc utilisable qu'à ce niveau (une variable instantanée ne change pas pendant une phase).

2.4.1.3. Transformation de données

La transformation des données est une fonction combinatoire qui transforme une configuration de bits en entrée en une autre configuration en sortie.

Par exemple :

$y = f(x1, x2 \ldots xn)$.

Les machines qu'on se propose de réaliser auront seulement des fonctions d'une ou deux variables. Les fonctions plus complexes devront être réalisées par décomposition (interprétation). Toutes les fonctions d'une ou deux variables ne seront pas implantées, mais seulement les plus simples qui représentent des primitives suffisantes pour interpréter les autres. Dans l'état actuel de la technologie, par exemple, un multiplieur de 16 x 16 bits n'est implanté (câblé) que pour des processeurs spéciaux ; il est, par contre, réalisé par interprétation de manière beaucoup plus fréquente.

Remarque : Parmi les activités d'une partie opérative, il faut considérer les entrées/sorties. On va considérer les entrées/sorties comme faisant partie de l'espace d'adressage de LO. Au niveau de LO on ne les voit pas autrement puisque on n'accède qu'à des variables. La spécificité de ces éléments est liée à des circuits tels que des amplificateurs et sont traités dans la communication entre parties opératives. La synchronisation propre à ces échanges est gérée par un niveau supérieur.

2.4.2. Représentation symbolique de la partie opérative

On va définir des ensembles d'éléments, chaque ensemble comprenant des éléments d'une certaine catégorie. Les relations entre ces ensembles sont ensuite exprimées par un symbolisme pour arriver à une formalisation de la partie opérative.

2.4.2.1. Définition des éléments

L'espace d'adressage est constitué des éléments de mémorisation. Définissons alors :

C - ensemble des constantes

V - ensemble des variables banalisées

X - ensemble des variables spécialisées.

L'espace d'adressage est alors S = C U V U X. Toutes les variables de l'espace S n'ont pas les mêmes droits d'accés.

Définissons maintenant un ensemble D constitué des opérateurs dyadiques, c'est-à-dire des éléments de transformation de données à deux opérandes. Chaque opérateur doit accéder à deux variables en entrée et générer une variable en sortie.

On va appeler α et β deux variables instantanées, accessibles à tous les éléments de la partie opérative.

On a vu que les opérateurs réalisent une fonction et que les variables spécialisées sont chargées avec la valeur d'une fonction. On ne pense pas relier tous les opérateurs à toutes les variables spécialisées (sauf s'il n'y en a qu'un ou deux) sinon on retrouve le mêm problème que pour le chemin entre variables.

L'association opérateur-variable présente suffisamment d'intérêt pour être traitée à part. On définit alors le couple k=(d,x), dans lequel d est un opérateur et x la variable associée. Si les ensembles D et X ont la même coordinalité, on peut ordonner les éléments de chacun de ces ensembles et les arranger en couples ki=(di,xi). L'ensemble K sera donc défini par :

$$K = \bigcup_{1} ki \;/\; 1 \leq i \leq /D/.$$

Un opérateur d peut exécuter plusieurs fonctions. Physiquement, d n'a qu'une sortie. Les différentes fonctions seront distinguées par des paramètres envoyés par la partie contrôle. La fonction est distinguées par un indice qui l'identifie.

Exemple : Si d peut exécuter trois fonctions, on les appellera d^1, d^2 et d^3. Parfois l'indice sera remplacé par un symbole identifiant la fonction en vue d'une meilleure lisibilité : supposons que d^1 soit la fonction identité, on la noterait, dans ce cas, $d^{\equiv}$.

2.4.2.2. Symbolisme des relations entre variables

Pour exprimer le comportement du circuit, on va représenter les actions par des symboles définis ci-après. Tout le langage n'est pas complètement défini; c'est en fait une partie des primitives de LO. LO est un langage de transfert de registres dont on a beaucoup d'exemples dans la littérature. Le but est d'établir une correspondance bijective entre ces primitives et sa réalisation physique.

a/ Types des opérandes

Dans les expressions, on va utiliser les quatre types d'opérandes suivants :

- constante
- variable
- registre
- bus.

Nous appellerons variable, une variable banalisée et registre une variable spécialisée. Un bus est une variable instantanée. Les opérandes sont déclarés par un nom. Les constantes reçoivent leur valeur au moment de la déclaration.

Exemples: Si on a des déclarations comme :

- constante zéro=0, deux=2
- variable addr, base
- registre acu, tampon
- bus alfa, bêta

alors on peut faire référence à zéro, deux, addr... Un ensemble d'éléments du même type peut constituer une variable structurée.
Exemple: soit mem une variable structurée déclarée contenir cinq éléments. On fait référence aux trois premiers éléments par :

mem[1..3] ou mem[1,2,3].

Remarque : Pour faire référence aux bits d'un opérande on utilise la forme syntaxique suivante :

acu <7>,
addr <8..15>
mem [2] <1,7,15>.

Les bits sont numérotés de 0 à n, correspondant à l'ordre poids faibles vers poids forts.

Le nombre de bits des opérandes correspond à la taille de la partie opérative, qui est définie dans les déclarations.
Exemple : partie opérative nom type dyadique taille 8 bits.

b/ **Les actions**

On distingue les actions en :

- affectation
- transformation
- connexion.

. Une affectation est le chargement d'une variable mémorisée. On la représente par le symbole "←".

. Une transformation est l'effet d'une fonction sur une ou deux variables.
Une fonction d'une variable "a" s'écrit :

op a

et une fonction de deux variables "a" et "b" s'acrit :

a op b,

où op est l'opérateur. La fonction donne une valeur qui peut être attribuée à une variable.

. La connexion correspond au transfert d'une variable dans une variable instantanée, représenté par ":=",
Exemples d'actions :

- alfa := a
- b ← alfa op bêta

On n'étudie pas ici le séquencement dans le langage de phase L0. Il sera seulement précisé que le séquencement progresse pas à pas, et que, à chaque pas

un certain nombre d'actions peuvent être réalisées. Les actions qui se réalisent dans le même pas sont séparées par une virgule. Les pas sont séparés par des points-virgules.

Exemple d'un pas : soit les trois actions "alfa:=a", "bêta:=b"; "c ← alfa op bêta". Si ces trois actions constituent un pas, on les écrira :

```
      ====
      ====  ;
   →  alfa := a, bêta := b<0..7>,
      c ← alfa op bêta ;
   →  ====
      ====
      ====
```

Les flèches marquent les étapes dans le séquencement. Dans les langages de transfert de registres, les étapes sont identifiées par des étiquettes ou par une expression booléenne.

Les actions qui constituent une étape sont dites en parallélisme co-latéral. Cela signifie que l'on ne fait aucune hypothèse sur l'ordre dans lequel ces actions se réalisent ; dans le cas de deux affectations

a ← x , b ← y ;

on ne sait pas quelle est la première exécutée à cause de la dispersion des horloges.

Remarque : Les opérateurs sont des circuits combinatoires. Ils peuvent avoir deux types de variables en entrée : les variables mémorisées et les variables instantanées. Une variable instantanée est valide tant que la connexion est active, sinon elle est invalide (ou indéfinie). La sortie d'un opérateur monadique est du même type que son entrée. Pour un opérateur dynamique, l'état de la sortie est donné par le tableau ci-dessous :

entrée 1 / entrée 2	indéfinie	instantanée	mémorisée
indéfinie	indéfinie	indéfinie	indéfinie
instantanée	indéfinie	instantanée	instantanée
mémorisée	indéfinie	instantanée	mémorisée

2.5. MODELE GRAPHIQUE DE LA PARTIE OPERATIVE

Les éléments d'une partie opérative doivent travailler ensemble. Dans un circuit intégré, il faut la réaliser sur un plan (les circuits sont planaires). On présente donc un modèle qui tient compte de ces restrictions et duquel on déduit la structure topologique des divers éléments de la partie opérative.

2.5.1. Schéma général

Le schéma qu'on va présenter (figure 2.8) montre à la fois les éléments de la partie opérative et leur communication, y compris le sens de cette communication (unidirectionnelle ou bidirectionnelle). Il y a deux voies de communication α et β .

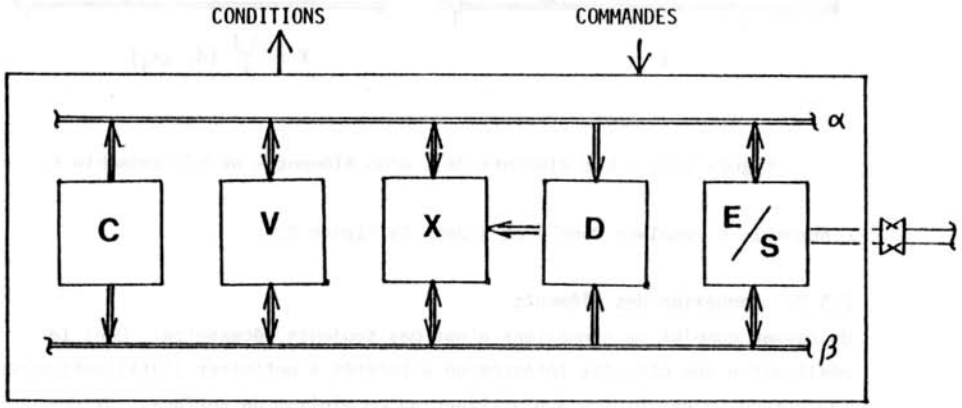


Figure 2.8 - Schéma graphique de la partie opérative.

Les blocs C, V, X et D sont les ensembles décrits auparavant, c'est-à-dire les constantes, les variables, les variables spécialisées et les opérateurs. Le bloc E/S sont les entrées/sorties.

Tous les opérateurs D ne sont pas liés à toutes les variables X. Plus exactement, si on reprend la définition de l'ensemble K, chaque élément de D est lié à une variable de X (variable associée). Un couple (d,x) donne un

élément k de K qui est représenté par la figure 2.9.a. L'ensemble K qu'on montre en 2.9.b a tous ses éléments connectés comme celui de 2.9.a.

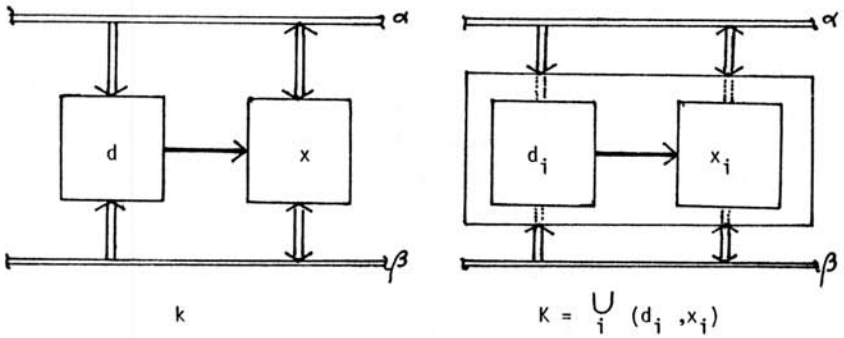


Figure 2.9. - Les éléments de K a/un élément k et b/l'ensemble K.

L'ensemble K remplace donc D et X dans la figure 2.8.

2.5.2. Connexion des éléments

Un réseau complet de connexions n'est pas toujours nécessaire. Dans la réalisation des circuits intégrés on a intérêt à optimiser l'utilisation de la surface, c'est-à-dire à n'utiliser qu'un minimum de surface. Si une commande n'est jamais utilisée, on va la supprimer car elle occupe de la surface, ainsi que les circuits qui la calculent.

Exemple : Supposons qu'une constante C1 apparaît toujours dans l'algorithme écrit en LO de la manière suivante :

```
alfa := C1,
```

où alfa est un bus (variable instantanée). On voit qu'il est inutile de réaliser la connexion bêta:=C1 car elle ne serait jamais utilisée.

Cette optimisation est repoussée dans la description de l'algorithme lui-même et est valable pour presque tous les éléments. Au point de vue de l'algorithme

elle consiste à trouver des cas comme celui donné dans l'exemple ci-dessous.
Supposons qu'on ait les deux étapes suivantes de l'algorithme :

1/ v1 ← alfa , alfa := c1
2/ v2 ← bêta , bêta := c1,
v3 ← alfa , alfa := x1 ;

On peut modifier l'étape 1 en changeant de bus utilisé (changer alfa par bêta) de manière à ce que c1 n'ait qu'une seule connexion (au bus bêta) au lieu de deux.

2.5.2.2. Les constantes (C) et les variables (V)

Les variables et les constantes présentées ont la même caractéristique du point de vue de la connectique. On étudie ici les variables, mais le raisonnement se transpose pour les constantes. L'ensemble des variables V dans une partie opérative à deux bus alfa et bêta , peut être décomposé en trois sous-ensembles:

V1 : variables connectées au bus alfa,
V2 : variables connectées au bus bêta,
V3 : variables connectées aux deux bus.

Graphiquement, on présente ces ensembles comme dans la figure 2.10.

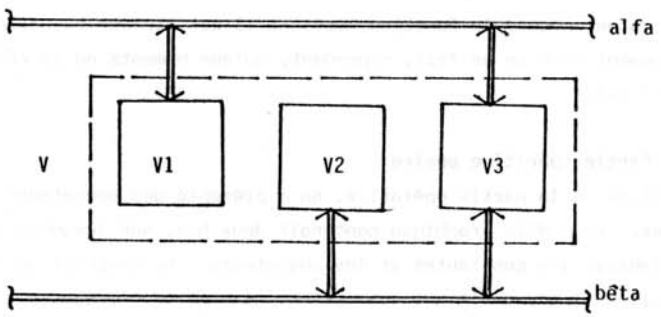


Figure 2.10 - Décomposition de V en trois sous-ensembles V1, V2, V3.

Remarque : Des sous-ensembles peuvent être vides, c'est-à-dire qu'il n'existe pas de variables d'une certaine classe. En particulier, si les trois sous-ensembles sont vides, cela signifie que V est vide et que l'on n'a aucune variable banalisée dans la partie opérative.

2.5.2.3. Les variables spécialisées

Les variables spécialisées ont trois chemins d'accés: un en écriture seule et deux chemins bidirectionnels. Le premier est toujours présent. Les deux autres sont variables quant à leur nombre et à leur nature :

a/ le nombre de chemins varie de la même manière que les variables banalisées ;

b/ la nature: on représente la connexion de la variable spécialisée avec le bus par un chemin bidirectionnel. Or, il s'avère que maintes applications ne l'utilisent que dans un sens. Cette caractéristique est remarquée dans l'algorithme par exemple, par l'absence d'instructions d'affectation dans cette variable à partir du bus.

2.5.2.4. Les opérateurs

Les opérateurs (dyadiques) sont des fonctions de deux variables et génèrent une sortie: il y a donc trois connexions. On a utilisé des expressions du type

b ← alfa op bêta,

où alfa et bêta sont des bus et où b est une variable spécialisée. L'opérateur calcule en permanence la fonction "op", si il est lié aux bus alfa et bêta. Le chargement de b ne se fait, cependant, qu'aux moments où le résultat doit être mémorisé.

2.5.3. Partie opérative unaire

Dans l'étude de la partie opérative, on a présenté des opérateurs monadiques et dyadiques. Le modèle graphique contenait deux bus, sur lesquels se branchent les variables, les constantes et les opérateurs. On pourrait se demander ce qui changerait s'il n'y avait que des opérations unaires à exécuter. Tout d'abord ce cas existe et on le met bien en évidence, par exemple dans les microprocesseurs. Les raisons et les avantages de cette technique sont analysés brièvement.

Les microprocesseurs exécutent des opérations sur des données. Le programme (ensemble des opérations à exécuter) et une partie des données sont placés dans

dans une mémoire externe où il faut rechercher les primitives et les données (supposons que les entrées/sorties soient dans l'espace mémoire) et y remettre les résultats. Il y a là une opération d'adressage. Il faut remarquer encore qu'on fait souvent référence à des positions consécutives de mémoire.

Les microprocesseurs de la première génération (8080, 6800, Z80 ...) ont un espace d'adressage de la mémoire externe de 64 Koctets. Ce sont des processeurs qui traitent des données de 8 bits et qui adressent la mémoire en 16 bits. L'option 16 bits d'adresse - 8 bits de données est un compromis technique entre plusieurs facteurs mais qui, finalement, est apparu bien adapté aux applications de ces machines.

L'ensemble des opérations sur les données est assez riche ; elles sont du type registre-accumulateur et mémoire-accumulateur. La recherche de sophistication se place surtout dans l'enrichissement des modes d'adressage et de l'implantation de primitives de synchronisation.

Une structure interne en 8 bits pour les données et 16 bits pour les adresses pose des problèmes pour le traitement en 16 bits. Cela se remarque à la pauvreté des opérations sur les adresses. On se limite, en général, à des opérations du type :

- comparaison en 16 bits (ne donnant que les résultats "égal" ou "différent"),
- addition d'une valeur 8 bits sur les poids faibles d'un nombre en 16 bits,
- incrémentation/décrémentation en 16 bits,
- déplacement registre vers mémoire et vice-versa.

L'évolution de l'intégration permettant le placement d'un nombre de transistors supérieur de presqu'un ordre de grandeur à celui des microprocesseurs cités, a donné naissance à une nouvelle famille: les microprocesseurs 16 bits. Mais on est aussitôt passé à un espace d'adressage plus important, et le problème continue. Pour le 68000, par exemple, les adresses sont en 32 bits (actuellement, 24 bits seulement sortent du boîtier) ; il y a trois parties opératives de seize bits, une pour les données et deux, plus simples, pour les adresses, construites autour de deux bus segmentés. La partie adresse pourrait être sans doute optimisée (comme dans le Z8000, par exemple, où les mémoires accèdent, alternativement, à un bus ou à l'autre). Il semble néanmoins que l'intérêt de

la régularité et l'utilisation de cellules standard a plus d'importance que quelques millimètres carrés de silicium. On rappellera cet argument, d'ailleurs, pour encourager le concepteur à utiliser des cellules standardisées même s'il peut gagner un peu de surface en redessinant une cellule spécifique.

Finalement, il faut mentionner que les processeurs utilisent bien le fait d'avoir deux parties opératives indépendantes (qui peuvent communiquer entre elles) puisqu'ils doivent chercher les instructions dans des positions successives de la mémoire. La recherche de l'instruction suivante peut souvent être déclenchée en parallèle avec la dernière étape de l'instruction en cours, ce qui apporte une amélioration sensible aux performances de la machine.

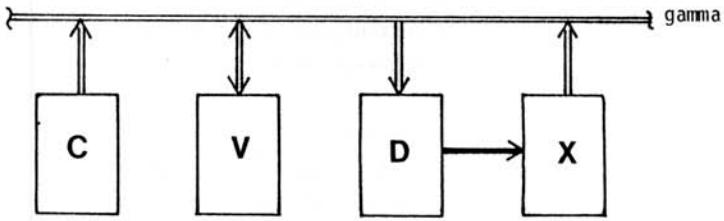


Figure 2.11 - Partie opérative monadique.

2.6. CONNEXION ENTRE PARTIES OPERATIVES

Deux parties opératives peuvent communiquer entre elles directement par leurs bus. Cela se fait par des connexions commandées qui relient les deux bus (figure 2.12). Ces connexions sont bidirectionnelles.

Figure 2.12 - Connexion entre deux bus

Il faut faire référence à la taille de la partie opérative. En effet, toutes les parties opératives n'ont pas la même taille (c'est-à-dire le même nombre de bits).

Jusqu'ici, on a supposé que les connexions ou affectations se faisaient sur tous les bits du bus (c'est-à-dire pour tous les bits de la partie opérative). Pour la connexion entre deux bus qui n'ont pas le même nombre de bits, il faut préciser les bits qui doivent communiquer. On représente la connexion entre deux bus par :

alfa :=: bêta,

où alfa et bêta sont deux bus. Si gamma est un bus de 16 bits et alfa est un bus de 8 bits, on pourrait écrire :

alfa :=: gamma <8..15>.

Remarque : Un bus est une variable instantanée et deux bus connectés sont aussi une variable intantanée. A un instant donné on ne peut avoir qu'une seule source d'information sur les deux bus. Si on a par exemple :

bêta := a,
alfa :=: bêta,
≡ ,

il est évident que pour que alfa soit valide, les deux connexions doivent être actives. Le symbole ":=:" n'établit pas une direction de transfert: il signifie qu'un bus suit l'autre. La connexion entre deux bus peut être maintenue sur plusieurs pas.

2.7. LES ENTREES/SORTIES

Les entrées/sorties sont traitées de la même façon que les variables ; en d'autres termes, elles sont ramenées dans l'espace d'adressage de l'algorithme. Les entrées/sorties assurent la communication entre une partie opérative et un bus externe. Du point de vue de la partie opérative, une entrée est une lecture d'un bus externe et une sortie est une écriture sur le bus externe. Une entrée/sortie permet les deux opérations: lecture et écriture (figure 2.13).

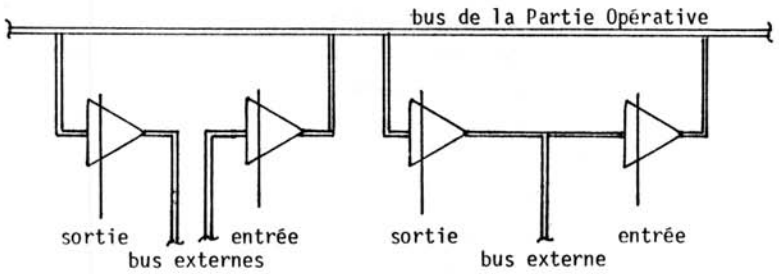


Figure 2.13 - Les entrées et les sorties

Des registres peuvent être placés entre le bus interne (bus de la partie opérative) et le bus externe si les deux bus ne sont pas synchrones. Dans la figure 2.14 on a une entrée/sortie avec deux registres, un pour l'entrée et l'autre pour la sortie.

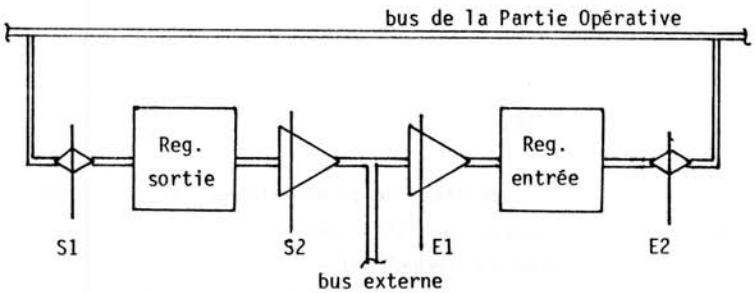


Figure 2.14 - Entrée/sortie avec registre en entrée et en sortie

Par hypothèse, les bus interne et externe ne peuvent pas être connectés directement, sinon l'opération serait tout simplement une connexion.

On va noter les entrées/sorties par le symbole "<=". Les exemples de la figure 2.13 seront écrits :

- pour une sortie : bus externe <= bus interne
- pour une entrée : bus interne <= bus externe.

L'opération sous-entend le traitement logique et électrique nécessaire pour réaliser la compatibilité entre les bus, s'il le faut.

L'exemple de la figure 2.14 comporte deux actions pour l'entrée et deux pour la sortie. Les actions S1 et E2 sont des transferts entre bus et registre et rentrent donc bien dans le cadre des opérations étudiées. S2 et E1 sont décrites :

- pour S2 : bus externe <= Rsortie,
- pour E1 : Rentrée <= bus externe.

Une entrée ou une sortie considèrent le bus externe comme un élément de communication unidirectionnel tandis que pour une entrée/sortie le bus externe est bidirectionnel.

2.8. ENSEMBLE DE CELLULES

Les éléments de la partie seront réalisés par des cellules fonctionnelles qui seront étudiées par la suite. Pour construire une partie opérative on assemble ces cellules qui sont donc faites pour travailler ensemble à tous points de vue: technologique, électrique, topologique, etc...

Cellules utilisées dans une partie opérative dyadique :

- Mémoire vive
  - . mémoire double accés
  - . mémoire à simple accés (sur un bus ou l'autre)
  - . mémoire associative
  - . mémoire avec remise à zéro asynchrone.

- Registre
  - . registre avec double accés (en plus de l'entrée de l'opérateur)
  - . registre à simple accés

- Constantes (les constantes seront en général en simple accés)

- Bus
  - . bus internes
  - . bus externes

- Opérateurs
  - . unité arithmétique et logique
  - . incrémenteur, décrémenteur, incrémenteur/décrémenteur
  - . décaleur (à gauche, à droite, bidirectionnel)
  - . multiplexeur
  - . amplificateur d'entrée, de sortie.

Remarque : Les opérateurs sont souvent liés à un circuit spécifique dans le sens que l'on a intérêt à avoir des opérateurs adaptés à l'algorithme, soit dans le sens de simplifier l'algorithme, soit de minimiser la surface utilisée. Parfois les deux avantages sont atteints simultanément. Si l'on n'utilise que des fonctions logiques très simples, il n'est pas utile d'implanter une unité arithmétique et logique. Si on fait systématiquement une opération de décalage, par exemple, on peut libérer le bus qui serait alors utilisé pour chaque opération de décalage en construisant un opérateur spécial. A ce point de vue, il est important de laisser au concepteur la possibilité de construire des opérateurs tout en utilisant des cellules du système.

Réalisation 3

3 - REALISATION

3.1. INTRODUCTION

Les systèmes digitaux binaires sont exprimés par des variables et des fonctions de commutation. Les variables sont associées à deux états bien définis. Ces deux états sont appelés VRAI et FAUX, OUVERT et FERME, 1(UN) et 0(ZERO), HAUT et BAS, etc... Les variables sont dites logiques ou booléennes.

Dans une réalisation physique, ces états sont associés à des grandeurs physiques comme le flux, la différence de potentiel, le courant, la vitesse, etc... Pour utiliser une grandeur physique représentative des variables logiques, il faut qu'on ait la possibilité technologique de construire des composants qui les manipulent comme les tubes à vide ou les transistors pour les grandeurs électriques. Il faut aussi qu'il soit possible de construire un ensemble génératif de fonctions booléennes (par exemple NOR) et que ces fonctions puissent être connectées. Soit la possibilité de :

- construire des dispositifs de commutation,
  réaliser un ensemble minimal de fonctions,
- connecter les fonctions.

3.2. LES ELEMENTS DE BASE

3.2.1. Les niveaux logiques

On commence par faire une application des variables logiques dans le domaine de la variable physique. Soient les valeurs logiques 0 et 1 et une grandeur physique X dans le domaine $(-\infty, +\infty)$. Une application est définie par la donnée d'une valeur de X appelée seuil et l'on établit que :

1 logique $\leftrightarrow$ X > seuil
0 logique $\leftrightarrow$ X $\leq$ seuil

Il est très difficile de réaliser des dispositifs qui distinguent exactement le seuil. Sans perte de généralité, le seuil est étendu d'une valeur à un intervalle. Prenons le seuil comme l'intervalle [S1,S2]. La nouvelle relation entre la variable logique et la variable X sera définie (figure 3.1) :

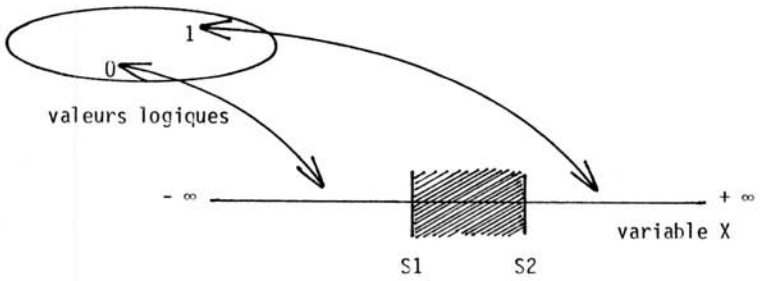


Figure 3.1. - Les valeurs logiques appliquées sur un domaine: les valeurs sont séparées par un intervalle.

Lorsque les variables logiques changent d'état, cela entraîne que les variables physiques transitent par la zone située entre les deux seuils, où elles n'ont pas de sens. Soient les deux variables a(continue) et b(bolléenne) dans la figure 3.2.

L'axe vertical donne l'évolution de a et l'axe horizontal celle de b ; quand a=a1, b change d'état.

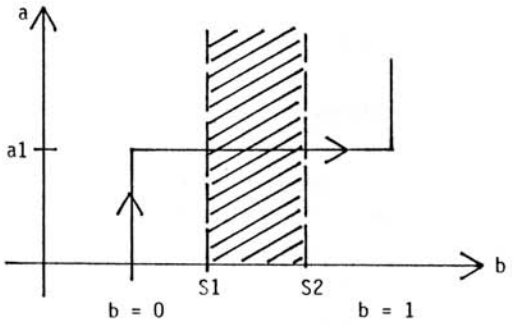


Figure 3.2. - Changement d'état d'une variable booléenne

Comme les variables, les fonctions changent l'état de leurs sorties en un temps fini. De plus, une fonction ne change pas nécessairement son état au même instant que ses variables d'entrée. Il en résulte qu'un système digital doit être étudié en rapport étroit avec son évolution temporelle car il y a des moments où quelques variables ou fonctions ont une valeur logique non définie.

3.2.2. Les dispositifs

Les dispositifs électriques, puis électroniques, prirent une position de premier rang dans le domaine des systèmes logiques. Les circuits électriques présentent en fait des caractéristiques qui les distinguent indiscutablement comme la vitesse, la facilité de connexion, la souplesse, les nombreux transducteurs et la miniaturisation. Sans étudier l'histoire des dispositifs électriques et électroniques, on peut mentionner les principaux pas suivis dans la découverte de ces dispositifs et des technologies qui ont permis leur utlisation généralisée: les relais, le tube à vide, la diode, le transistor à junction puis à effet de champ et enfin, les circuits intégrés.

Avec les circuits intégrés, plusieurs composants, ainsi que leurs connexions, sont fabriqués en même temps. On intègre donc des fonctions. L'évolution de l'intégration a permis la réalisation d'un nombre de plus en plus grand de fonctions dans des systèmes digitaux complexes. L'implication économique de tels "macro-composants" est évidente. D'un coté, leur fabrication en chaîne les rend peu coûteux et favorise donc la généralisation de leur utilisation, et par là, la mise en service de puissants moyens de recherche et de fabrication, poussés par un marché florissant. D'autre part, l'assemblage de ces macrocomposants a rendu possible la construction de gros systèmes tels que les ordinateurs.

Deux grandes familles technologiques sont utilisées pour la fabrication des circuits intégrés: la famille bipolaire (basée sur le transistor à jonctions) et la famille MOS (basée sur le transistor à effet de champ). Dans chaque

famille il existe plusieurs configurations possibles des éléments en vue de réaliser de la logique. On va examiner brièvement la configuration de base de chacune de ces configurations (appelées technologies) pour étudier ensuite la technologie retenue :

Les principaux éléments de la famille bipolaire sont : TTL, ECL et I²L.

- TTL : La configuration de base de cette technologie est montrée à la figure 3.3.a. A l'entrée, on a un transistor dit multi-émetteur (possédant plusieurs émetteurs. La sortie de ce circuit (s) est à zéro quand les trois entrées sont à un. La fonction exécutée est donc $\overline{s} = 11.12.13$ ou $s = \overline{11.12.13}$.

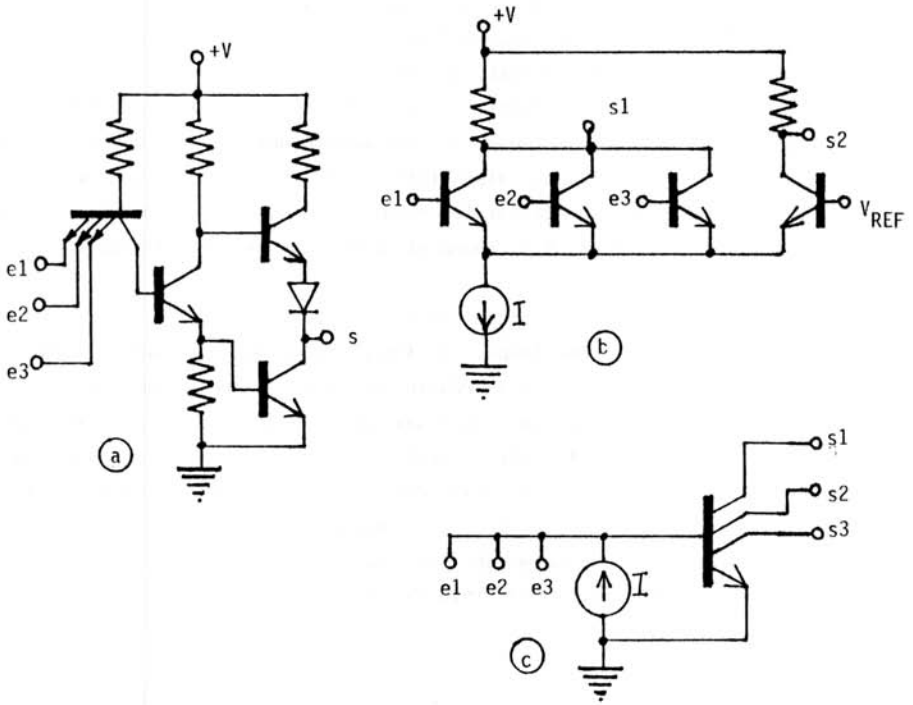


Figure 3.3. - Famille bipolaire.

- ECL : Le circuit de la figure 3.3.b est une configuration ECL simplifiée pour toirs entrées, e1, e2 et e3. Si une des entrées est à un niveau supérieur à VREF, le transistor auquel cette entrée est reliée, conduit et le transistor lié à VREF sera bloqué. On aura donc s1 au niveau bas et s2 au niveau haut. Si toutes les entrées sont à un niveau inférieur à VREF, le transistor lié à VREF conduit et tous les autres sont bloqués. s1 et s2 sont complémentaires. La fonction exécutée est :

$$s2 = I1 + I2 + I3$$

- I2L : La porte de base I2L est montrée dans la figure 3.3.c. On a un transistor multi-collecteur : si le transistor conduit, tous les collecteurs (les sorties s1, s2 et s3) sont au niveau bas. Si une entrée est au niveau bas, elle absorbe le courant I, sinon ce courant passe par le transistor qui conduit. Toutes les sorties ont la même valeur logique. La fonction logique exécutée est :

$$s1 = \overline{I1.I2.I3}$$

Famille MOS

La famille MOS est faite de différentes configurations des transistors à effet de champ du type N et du type P.

- NMOS: Les circuits construits avec des transistors de canal N s'appellent NMOS. Il y a deux configurations de base qui permettent de réaliser directement les deux fonctions NONET et NONOU (figures 3.5.a et 3.5.b). Les transistors conduisent quand le signal d'entrée est à un. Les fonctions sont donc :

$$s1 = \overline{I1.I2} \qquad s2 = \overline{I1 + I2}$$

La charge R est réalisée par un transistor déplété. Il est possible d'utiliser un transistor comme un interpréteur (3.4.c): I1 et I2 sont connectés quand le signal est à un.

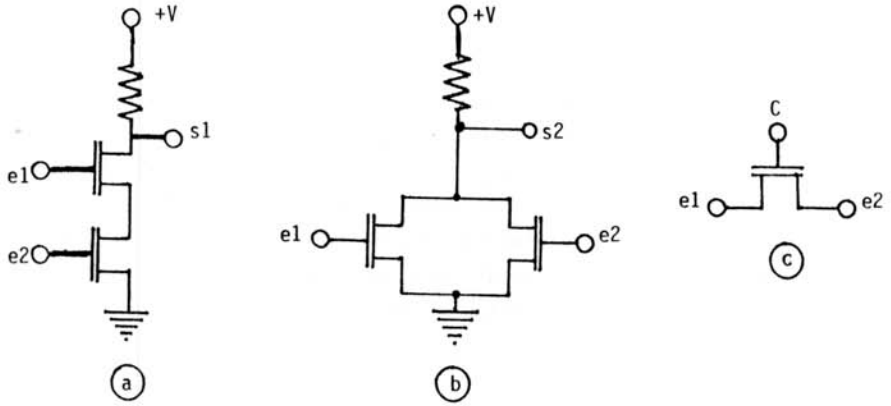


Figure 3.4. - Deux configurations de base NMOS (a et b) et le transistor de passage (c).

- CMOS : En CMOS, on dispose des deux types de transistors, les P et les N. Le signal d'entrée commande à la fois deux transistors en série: un des transistors conduit quand le signal d'entrée est à zéro tandis que l'autre conduit quand le signal d'entrée est à un. Dans la figure 3.5 on présente un inverseur, un NONET et un NONOU en CMOS.

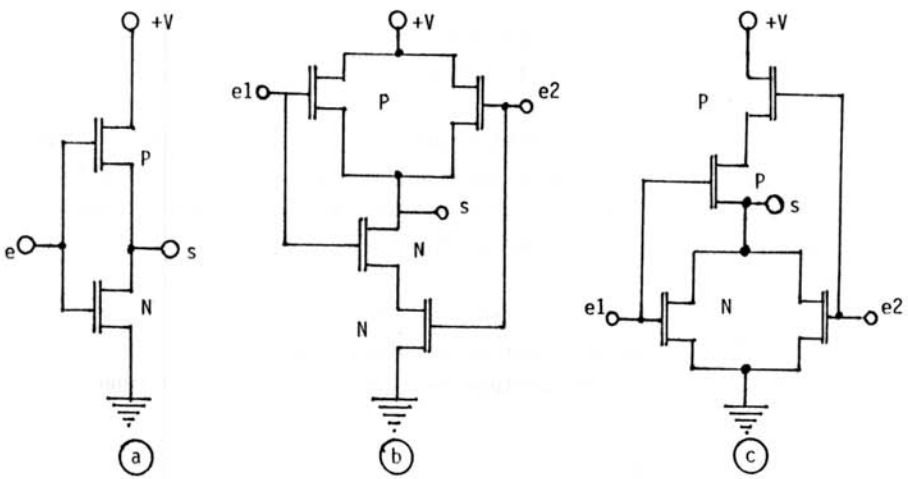


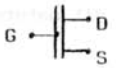
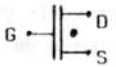
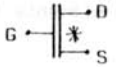
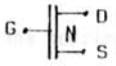
Figure 3.5. - Configuration CMOS, a/ inverseur, b/ NONET et c/ NONOU.

Remarque : Il y a deux techniques pour réaliser la grille des transistors MOS : grille en métal ou grille en silicium polycristallin. Dans chaque technologie, les circuits intégrés sont classés par la taille minimale de la grille des transistors. On parlera de technologie NMOS grille poly en 6 microns pour la technologie NMOS à grille silicium polycristallin dans laquelle la largeur minimale des grilles en poly est 6 microns.

On va travailler avec la technologie NMOS grille poly appelée HMOS.

3.2.2.1. Les transistors

En HMOS on a quatre composants de base et trois niveaux de conducteurs pour les interconnexions. Le circuit est alimenté en 5 volts. Les transistors, leur tension de seuil (Vt) typique ainsi que leur représentation symbolique dans la technologie utilisée, sont donnés dans le tableau ci-dessous :

type du transistor	Vt	symbole	
enrichi	0,6 V		G = grille
légèrement déplété	-2V		D = drain
fortement déplété	-4V		S = source
naturel	0V		

La caractéristique dynamique de transconductance d'un transistor est montrée dans la figure 3.6.

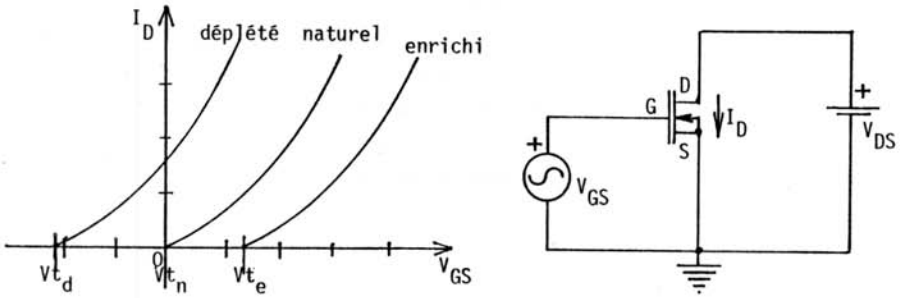


Figure 3.6. - Caractéristique du transistor MOS

Les différents transistors se distinguent par l'ordonnée du point Vt sur l'axe VGS. Si Vt est plus grand que zéro, le transistor est dit enrichi (ou à enrichissement), sinon il est dit déplété (ou à déplétion). Si Vt $\tilde{=}$ zéro, il est dit naturel.

La variable tension varie dans l'intervalle 0V-5V puisque le circuit est alimenté par une source de 5V. Les transistors déplétés ne sont donc pas utilisables comme éléments interrupteurs.

3.2.1.1. Le transistor enrichi

L'élément utilisé comme interrupteur est le transistor enrichi. Le comportement de ce composant est présenté dans la figure 3.7.

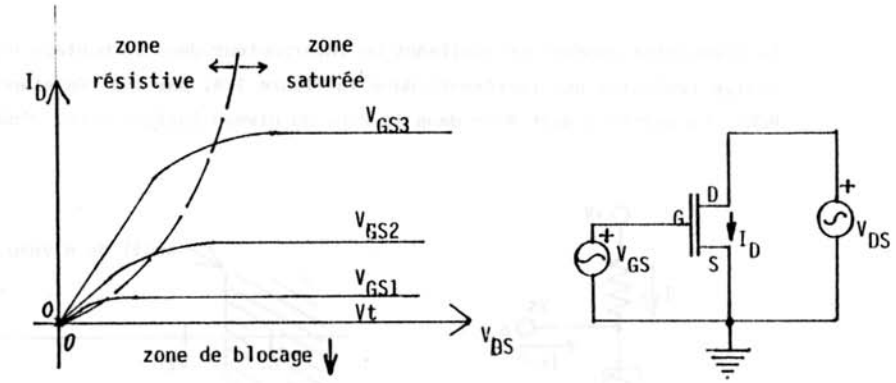


Figure 3.7. - Caractéristiques du transistor enrichi.

On y trouve trois zones :

- Zone résistive : (VGS - VDS) > Vt.

Dans cette zone, le transistor peut être assimilé à une résistance à l'origine. La valeur de la résistance dépend de VGS. Le courant est donné par l'équation :

$$ID = K\gamma \left((VGS - Vt) \times VDS - \frac{1}{2} VDS^2\right).$$

- Zone saturée : (VGS - VDS) < Vt.

Dans cette zone, le courant est pratiquement indépendant de VDS pour un VGS donné :

$$ID = K'\gamma \, (VGS - Vt)^2.$$

- Zone de blocage : VGS < Vt.

Dans cette zone, il n'y a pas de courant à travers le transistor.

Les transistors MOS utilisés ne sont pas des interrupteurs idéaux: quand ils sont conducteurs, ils ne sont pas un court-circuit pur et quand ils sont bloqués ils ne sont pas un circuit ouvert parfait. Néanmoins, le rapport très grand entre leurs résistances dans ces deux états leur permet d'être utilisés comme des interrupteurs, sous réserve de quelques restrictions.

Le transistor conducteur réalisant un interrupteur dans le montage d'une charge résistive est représenté dans la figure 3.8. par une résistance appelée RON. La sortie s doit être dans la zone du niveau logique zéro, d'où :

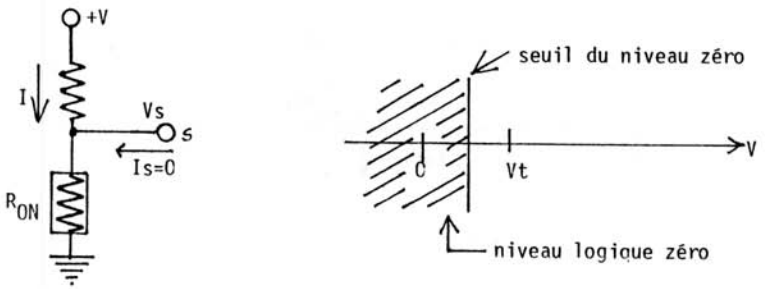


Figure 3.8. - Le transistor en conduction et le niveau zéro.

Pour que la porte suivante soit bloquée, Vs doit être inférieur à Vt.
Il n'y a pas de formule pour calculer la valeur du niveau 0. C'est un compromis entre plusieurs facteurs comme :

- variation de Vt et RON en fonction de la température de fonctionnement du circuit et des tolérances de fabrication ;
- immunité au bruit ;
- relation vitesse x consommation.

Sans discuter en détail ces points, on prend ici comme seuil du niveau zéro la valeur de 0,2 volts.

Remarque : La convention logique ci-dessus n'est utilisée qu'à l'intérieur du circuit intégré. La communication entre circuits peut être faite selon des conventions plus générales (niveaux TTL, par exemple).

La technologie fige les paramètres électriques des transistors. Le concepteur n'a que la liberté de définir la taille des transistors dans certaines limites.

Le transistor de la figure 3.9. est défini par L et W : L longueur et W largeur. On appelle aussi la surface W x L la zone active ou canal ; la forme du drain et de la source n'ont pas d'influence sur le comportement électrique du transistor lui-même. Une taille minimale est cependant spécifiée dans les règles de conception.

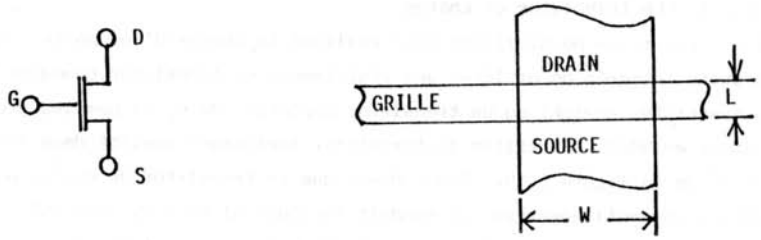


Figure 3.9. - Le transistor: zone active ou canal.

Une excursion de VG de 0 à 5V fait se déplacer le point s (figure 3.8) sur une droite de charge définie par R est les caractéristiques du transistor. Dans la pratique, le transistor se trouve en général dans la zone résistive pour le niveau zéro de sortie de la porte. La résistance (RON) est modulée par VG. Elle peut être donnée pour des valeurs discrètes de VG comme suit ([] signifie unité de surface) :

$VG = 3V \quad - RON_{[]} = 12\ K\Omega/[]$

$VG = LV \quad - RON_{[]} = 8\ K\Omega/[]$

$VG = 5V \quad - RON_{[]} = 6\ K\Omega/[]$

La résistance de l'interrupteur peut donc être choisie par le rapport des dimensions W et L (rapport de forme) :

$$RON = RON_{[]} \times \frac{L}{W}$$

On note :

$$Y = \frac{W}{L}$$

On a donc :

$$RON = \frac{1}{Y} . RON_{[]}$$

d'où :

$$RON = \frac{V_{SZ}}{I} = \frac{V_{SZ} \times R}{V - V_{SZ}}$$

Ces deux équations de RON permettent d'écrire :

$$\gamma \geq \frac{V - V_{SZ}}{V_{SZ} \times R \times RON_{\square}}$$

V_{SZ} est le niveau de tension choisi pour la valeur logique zéro.

3.2.2.1. Le transistor de charge

Il y a plusieurs possibilités pour réaliser la charge d'une porte. Dans les circuits intégrés on utilise: une résistance, un transistor commandé (CMOS), un transistor enrichi ou un transistor déplété. On ne va pas les étudier toutes: en HMOS on utilise le transistor légèrement déplété dans la configuration de la figure 3.10. Etant donné que le transistor de charge a une tension de seuil négative, il conduit toujours si Vs < V, pour VGS = 0. Il suffit donc de connaître le courant I pour VS égal au niveau zéro.

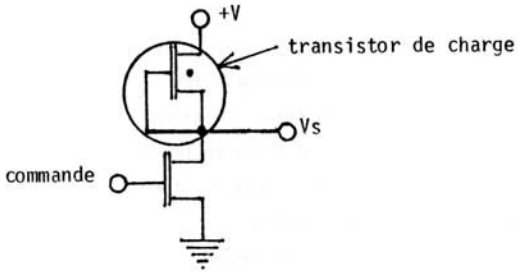


Figure 3.10 - L'inverseur avec un transistor déplété comme charge.

Le transistor de charge est saturé si VGS - Vt < VDS. Dans notre exemple (figure 3.10) et avec Vt = -2V et V=5V, on a :

Vs < 3V - le transistor de charge est saturé,

Vs < 3V - le transistor de charge est dans la zone résistive.

Au lieu d'une droite de charge, on a la caractéristique suivante pour l'inverseur :

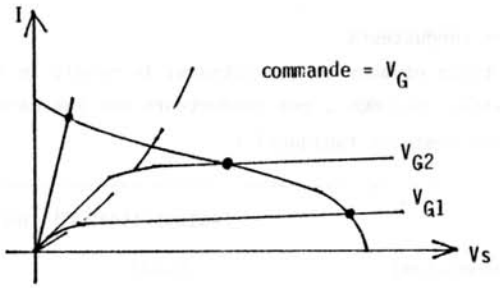


Figure 3.11 - Caractéristique de l'inverseur.

La fonction de transfert vs=f(vi) (la commande est appelée Vi) est montrée par la figure 3.12 pour un même transistor interrupteur et différents transistors de charge. Le transistor interrupteur sera dorénavant appelé transistor signal ; le mot interrupteur sera utilisé pour une configuration spécifique qu'on définira ultérieurement. Le rapport de forme du transistor

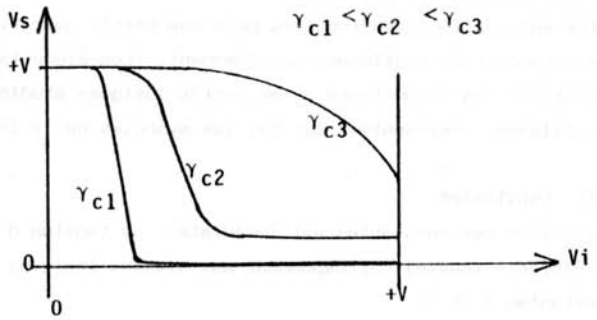


Figure 3.12 - Courbe de transfert Vi de l'inverseur.

signal sera noté γ_s et celle du transistor de charge γ_c. Pour le transistor de charge saturé, on peut donner le courant relativement au facteur de forme ($I_{\square}$ est le courant par unité de surface à Vs=0). Le courant Ic dans le transistor sera : $I = \gamma_c \times I_{\square}$

3.2.2.2. Les conducteurs

En MOS on a trois niveaux de conducteurs: le métal, le silicium polycristallin et la diffusion. En HMOS , ces conducteurs ont les caractéristiques électriques suivantes (valeurs typiques) :

	résistivité(Ω/□)	capacité/substrat(pFx10^{-4}/□)
métal (aluminium)	0,033	0,3
silicium	20 - 30	0,4 (5 pour le canal)
diffusion	22 - 34	1

Les connexions seront faites avec le conducteur de plus faible impédance (métal). Si cela n'est pas possible, on utilise du silicium polycristallin et finalement de la diffusion. Il est possible de connecter chaque conducteur aux deux autres par des contacts.

3.2.3. Réalisation des primitives

Les différentes primitives requises pour une partie opérative seront construites avec les dispositifs mentionnés précédemment. Les éléments de base sont les transistors et les conducteurs ; les portes logiques étudiées auparavant sont aussi utilisées, représentées ici par les symboles de la logique de commutation.

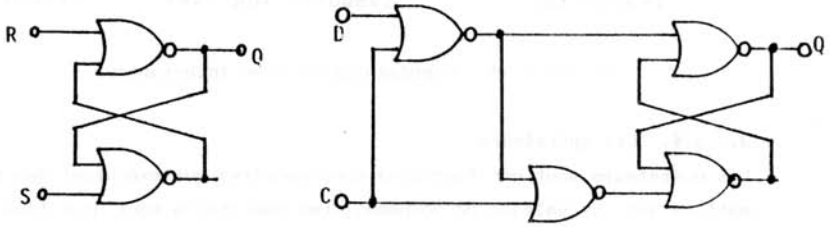
3.2.3.1. Constantes

La réalisation des constantes est immédiate. La tension d'alimentation (+V) et la référence (masse) correspondent aux niveaux logiques et fournissent les constantes 1 et 0.

3.2.3.2. Les variables mémorisées

Les variables mémorisées doivent conserver la valeur d'une variable intantanée, ou d'une autre variable mémorisée ou d'une fonction. Dans tous les cas, le stockage de l'information ne se fait pas continuement. Il y a un instant, dans la séquence des évènements, où le chargement doit se produire: c'est la réalisation physique de l'affectation. L'ordre envoyé à l'élément de mémorisation

pour exécuter le chargement sera appelé "commande de chargement" ou "commande d'écriture". Un circuit à réaction dont le gain de boucle est plus grand que 1, peut constituer un point mémoire bi-stable. C'est le cas du circuit de la figure 3.13.a, connu comme BASCULE RS. En 3.13-b le même circuit est précédé d'un étage pour commander son chargement.



S	R	Q_{p+1}
0	0	Q_p
0	1	0
1	0	1
1	1	*

C	D	Q_{p+1}
0	0	0
0	1	1
1	0	Q_p
1	1	Q_p

* état invalide
Q_p état précedent

Figure 3.13 - Bascule RS a/ circuit de base, b/ avec un circuit de commande en entrée.

On a défini la notion de variable instantanée basée sur l'idée de connexion: une variable instantanée est valable tant qu'elle est connectée. On sous-entend que la connexion peut être conditionnée par un interrupteur réalisé par un transistor. C'est dans ce sens précis que le mot interrupteur sera désormais utilisé. La figure 3.14 montre les différentes représentations utilisées. v2 est une variable instantanée. Il n'y a pas de restriction pour les variables v1 et C (mémorisée ou instantanée). L'activation de la commande correspond à la validation de la connexion.

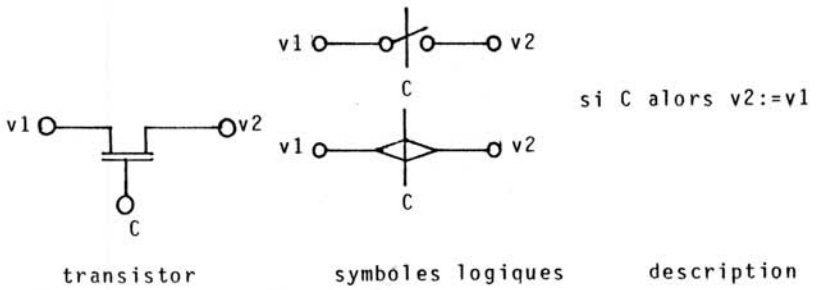


Figure 3.14 - Représentation d'un interrupteur

3.2.3.4. Les opérateurs

Les opérateurs sont des fonctions combinatoires qui exécutent des transformations sur les valeurs de données. Les opérateurs sont donc tous des fonctions logiques et on les réalise avec des portes logiques ou des montages électriques équivalents.

3.2.3.5. Les bus

Les bus sont des variables instantanées auxquelles peuvent être connectées directement (par des interrupteurs) ou indirectement (à travers des portes logiques) plusieurs variables mémorisées.

Exemple d'un bus :

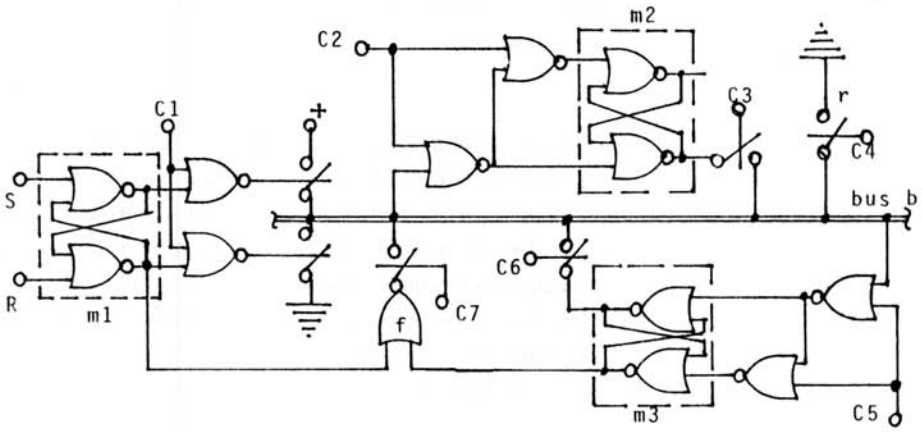


Figure 3.15 - Exemple d'un bus

La ligne b est un bus. m1, m2 et m3 sont des éléments de mémorisation (ram), r est une ROM et f est une fonction ($f = \neg(m1 \vee m3)$). Les éléments de mémorisation sont la réalisation des variables, la ROM est la réalisation d'une constante et la fonction f est la réalisation d'un opérateur. Un seul de ces éléments peut écrire sur le bus à un instant donné :

- m1 par la commande C1,
- m2 par la commande C3,
- m3 par la commande C6,
- r par la commande C4,
- ou f par la commande C7.

La valeur du bus peut être chargée en m2 par la commande C2, ou m3 par la commande C5. m2 et m3 peuvent être chargées simultanément. m1 est chargée à partir de l'extérieur par les lignes S et R.

Supposons qu'on veuille réaliser l'opration

$m3 = m3 \vee m1$,

avec les éléments de la figure 3.15. La séquence des opérations et les commandes activées sont présentées ci-dessous :

pas	actions	commandes	observations
1	b:=f, m2←b	C7, C2	$m2 = \neg(m3 \vee m1)$
2	b:=m2, m3←b	C3, C5	m3 = m2
3	b :=f, m1←0, m2←b	R, C7, C2	$m2 = \neg m3$
4	b:=m2, m3←b	C3, C5	m3 = m2

Remarque : La valeur de m1 a dû être détruite. On peut observer que le bus est utilisé à toutes les étapes. Le nombre de celles-ci serait réduit de moitié si on disposait directement de la fonction OU. On voit donc la nécessité de bien choisir les opérateurs et la connexion entre les éléments. Dans l'exemple précédent, on n'aurait pas eu besoin de détruire la valeur de m1 si l'oprateur avait en entrée le bus et m3, et m2 en sortie.

3.3. LES CHARGES EXTERNES DES PORTES

3.3.1. Modélisation des conducteurs

Les éléments d'un circuit sont distribués sur un plan. Les variables dont on vient de parler sont physiquement éloignées les unes des autres et pour les faire communiquer, il faut prévoir des connexions.

Exemple : l1, l2 ... l9 sont des liaisons.

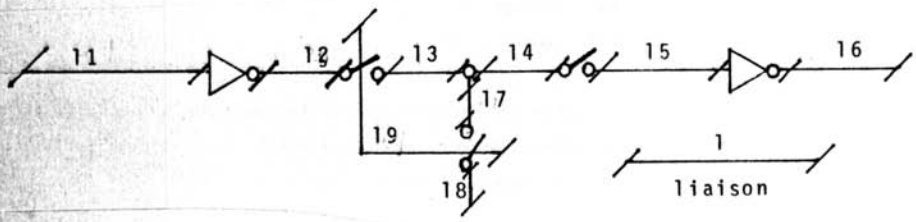


Figure 3.16 - Les liaisons

Les conducteurs utilisés ont une résistance et une capacité distribuées et on les représente comme dans la figure 3.17.a . Pour simplifier le calcul, les conducteurs sont modélisés par des éléments discrets (figure 3.17.b). Pour les circuits digitaux, trois cellules donnent un modèle assez satisfaisant et l'on utilise souvent le modèle à une cellule (non symétrique).

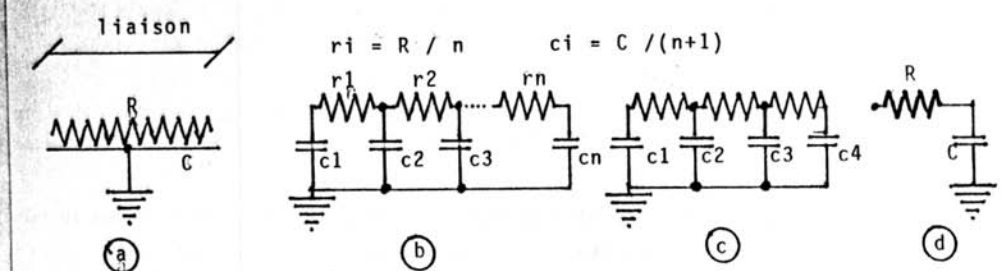


Figure 3.17 - Les liaisons, a/ représentations par des éléments distribués, b/ modèle en éléments discrets, c/ modèle utilisé: à trois et d/ à une cellule.

3.3.2. Une porte logique chargée

Une porte logique qui pilote une liaison doit donc fournir un courant quand sa sortie change d'état pour modifier la tension aux bornes de la capacité de cette liaison, en plus de sa propre capacité (on admet que les courants de fuite des capacités ne sont pas significatifs). On a fait une étude approximative de la forme des signaux pendant la commutation ; une forme plus exacte est obtenue par simulation électrique.

3.4. LA PARTIE OPERATIVE

3.4.1. Les bus

Les bus sont des moyens de communication entre des éléments d'un circuit. Ces éléments sont distribués sur une surface et pour les faire communiquer, un bus doit physiquement les atteindre. On a donc une liaison de largeur importante par rapport aux dimensions des éléments eux-mêmes, et, en conséquence, une charge associée relativement importante. Cette charge est en fait la caractéristique la plus importante des bus vis-à-vis du comportement électrique.

On dispose de trois conducteurs: métal, poly(silicium polycristallin) et diffusion. Il est naturel de choisir celui dont l'impédance est la plus faible pour des liaisons longues. C'est le cas des bus qu'on fera autant que possible en métal.

La résistivité du métal est de plusieurs ordres de grandeur inférieure à celle des autres conducteurs et des éléments actifs. Quand on utilise le métal comme liaison pour les données, on peut négliger sa résistance, en général (ce n'est pas le cas pour les alimentations). Pour l'étude du comportement du bus, on le modélise par une capacité qui représente l'addition de toutes les capacités qui sont liées au bus, plus sa propre capacité.

Les bus présentent, par définition, une charge importante. Ils peuvent atteindre quelques picofarads dans les parties opératives d'un microprocesseur.

Il n'est donc pas intéressant d'attaquer un bus par une porte simple. La solution d'introduire un amplificateur pour chaque élément qui attaque le bus est écartée a priori parce qu'elle est trop coûteuse en surface et, si l'amplificateur est à charge non commandée, elle est également trop coûteuse en puissance.

Le MOS signal de l'inverseur qu'on a étudié, et de toutes les portes logiques en général, est beaucoup moins résistif que le MOS de charge. En s'appuyant sur cette propriété, on peut envisager une autre solution: on précharge systématiquement le bus avant qu'une porte soit connectée. Si la porte est à zéro, elle décharge le bus, sinon la tension du bus n'est pas modifiée puisqu'elle est déjà à la bonne valeur. C'est une solution utilisée fréquemment et qu'on adoptera ici.

La technique de précharge présente évidemment le désavantage d'exiger une phase supplémentaire à chaque cycle, ce qui entraîne un temps mort pour le calcul. Les avantages sont importants (il y a d'ailleurs la possibilité d'utiliser ce temps en ajoutant d'autres éléments de mémorisation, mais on ne la traitera pas ici).

La précharge est faite par un transistor enrichi dans la configuration de la figure 3.18. P est la phase de précharge et C est la capacité équivalente du bus. Avec la précharge, la tension du bus est amenée à 3,8V avec P=5V (le transistor se bloque avec VGS = 1,2V à cause de l'effet du substrat).

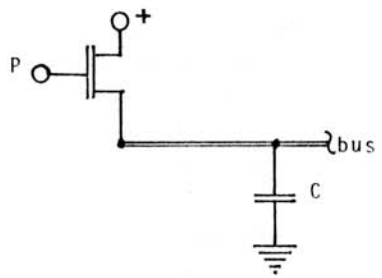


Figure 3.18 - Circuit de précharge.

La dimension du transistor de précharge dépend de la valeur de la capacité C du bus et du temps de précharge désiré. La valeur de 3,8V est atteinte asymptotiquement. Le niveau logique "un" doit donc être choisi un peu en dessous (3,5V par exemple) des 4V pris habituellement.

Si le bus n'est pas utilisé immédiatement après la précharge, des transistors de maintient très résistifs (qui fournissent juste un courant pour compenser les fuites) seront ajoutés.

Dans ces conditions, on va connecter au bus les divers éléments de la partie opérative.

3.4.2. Les constantes

Une constante "zéro" est réalisée par un transistor qui décharge le bus. Une constante un est implicite puisque le bus est déjà à "un" logique par la précharge. La taille du transistor qui réalise le zéro logique dépend de la charge du bus et de la durée de la phase de lecture.

3.4.3. Les portes

Une porte qui attaque le bus peut être similaire à un inverseur. Cette porte est connectée au bus par l'intermédiaire d'un interrupteur. Si elle à 1 logique (5V) aucune modification ne se produira puisque le bus a été préchargé. Si la porte est à zéro, elle déchargera le bus. Si nécessaire, il est possible d'augmenter la taille du MOS signal des portes qui attaquent le bus.

3.4.4. Les mémoires

La mémoire de la figure 3.13 utilisée dans un montage comme celui de la figure 3.15, est chargée avec la valeur du bus à travers une logique de contrôle : pour écrire dans la mémoire il suffit de s'assurer que les signaux sont logiquement corrects quant à leur durée (respectant le temps de basculement) et quant à leur relation avec les signaux de contrôle (la donnée doit rester stable un certain temps avant, et prendant, que le signal de contrôle est désactivé).

La lecture de la mémoire présente d'autres contraintes que le temps de décharge décharge du bus qu'on a étudié pour les portes. La bascule est un circuit à réaction et il n'est pas impossible que la mémoire bascule au lieu de décharger le bus. Il serait possible d'isoler le point mémoire par un inverseur (ou par un transistor à la masse comme dans les registres du INS 8070). Cependant, si l'on compare le nombre de transistors de ce type de mémoire (12 transistors) avec une cellule élémentaire de mémorisation constituée de deux inverseurs rebouclés (4 transistors) on voit que l'on est déjà loin d'un circuit minimal. Il faut encore ajouter à cela que les mémoires sont des cellules très utilisées et représentent une partie importante des parties opératives des circuits du type microprocesseur. On va donc essayer de trouver des circuits performants pour la cellule mémoire.

3.4.5. Mémoires à deux inverseurs

Le circuit de base d'une mémoire est constitué de deux inverseurs rebouclés comme dans la figure 3.19. Deux inverseurs minimaux ont été simulés dans le

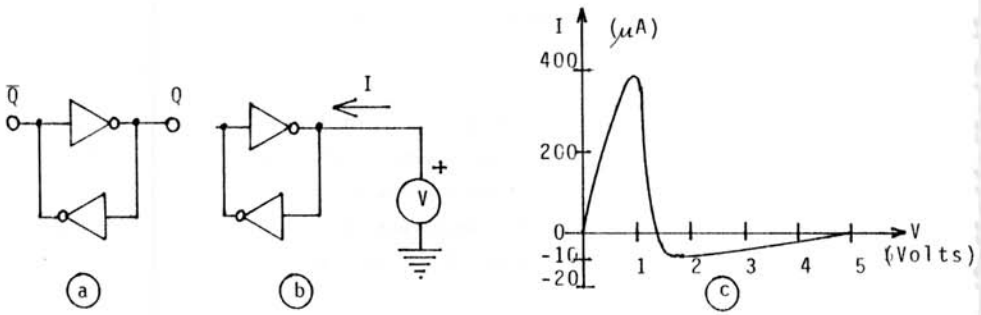


Figure 3.19 - Cellule mémoire, a/circuit de base, b/montage de test et c/caractéristique VxI (deux inverseurs minimaux)

montage 3.19.b et la caractéristique obtenue est montrée dans 3.19.c. Pour garantir que le point ne bascule pas, il suffit de s'assurer que le courant I est inférieur à 400 µA. Le moyen de le faire est de choisir un interrupteur dont la résistance limite le courant à la valeur mentionnée.

Une droite de charge qui passerait par les points (5V, 0µA) et le maximum de la courbe 3.19.c aurait une résistance de 10 KΩ, approximativement. Le résultat de la simulation (figure 3.20) montre que le point ne bascule pas pour V1 inférieur à 5V.

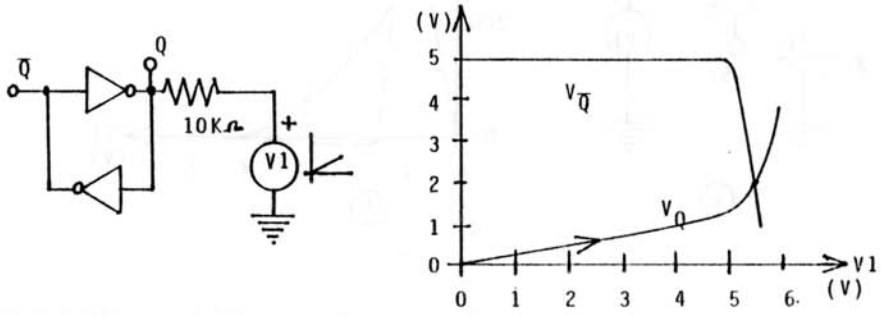


Figure 3.20 - Simulation du point mémoire
a/ circuit et b/ comportement de la bascule.

Les bus sont chargés jusqu'à 3,8V au maximum. On a donc simulé un interrupteur dans la configuration montrée en 3.21. Pour un transistor de L=4µ et W=8µ on a eu le résultat montré en trait continu dans la figure 3.21.b. Sur la même figure est portée en pointillé la caractéristique du point mémoire. Le point P est le point de fonctionnement au début de la lecture. Le point P a été choisi à peut près 20% en dessous du point de basculement. Cette marge doit être suffisante, compte tenu que les deux courbes se déplacent dans le même sens avec les variations de température.

On a donc obtenu une cellule qui ne bascule pas quand elle est lue sur un bus et cela indépendamment de la charge du bus. La conséquence est immédiate: il est impossible d'écrire la valeur logique "un" si la bascule est à "zéro". Par contre, si le bus est à zéro et la bascule à un, elle changera d'état au moment de la connexion au bus. Il faut donc travailler avec des bus préchargés.

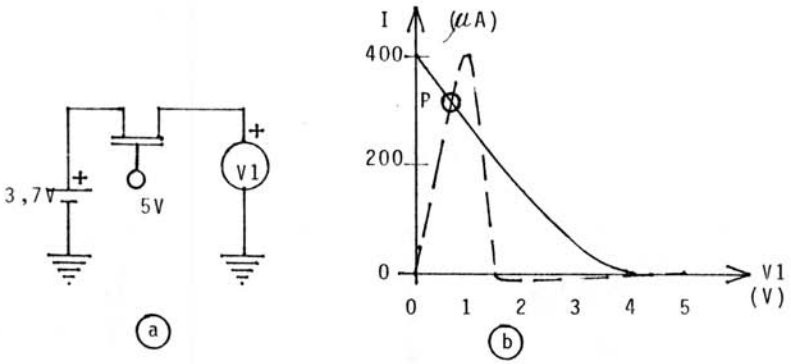


Figure 3.21 - L'interrupteur, a/ montage simulé, b/ caractéristique

3.4.6. Le registre

La solution pour l'écriture consiste à créer un autre chemin d'accés pour l'écriture : un interrupteur étudié pour faire basculer le point mémoire, c'est-à-dire un interrupteur moins résistif.

La résistance de l'interrupteur est fonction de sa tension de contrôle ; on peut donc jouer sur ce paramètre. Cette solution est donc dépendante de la forme du signal de contrôle et de la charge du bus pour un bon fonctionnement. Un circuit plus sûr est celui de la figure 3.22, où on introduit une résistance dans la boucle de réaction. Si R est suffisamment grand, la bascule peut même être attaquée par un inverseur minimal. Le circuit simulé (figure 3.22.b) contient trois inverseurs minimaux ; l'interrupteur n'est pas critique et la résistance est réalisée par un transistor long (> 100 KΩ). On voit, d'après la simulation, que la bascule passe de 0 à 1 quand le signal d'écriture atteint 3V et passe de 1 à 0 lorsque le même signal atteint 2,5V.

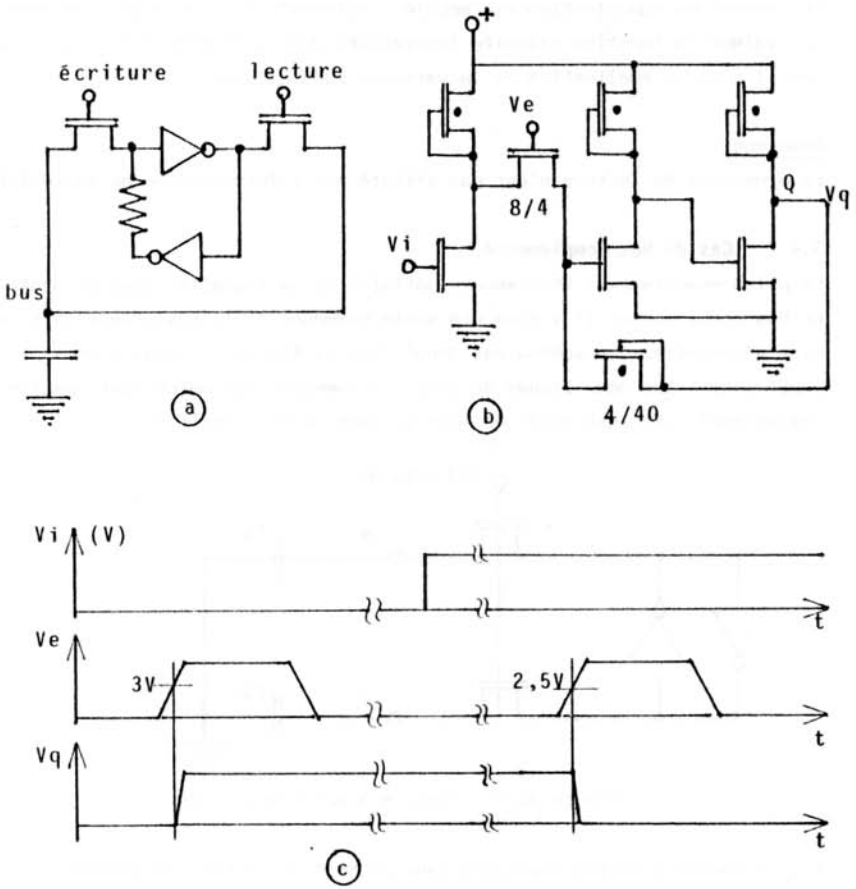


Figure 3.22 - Bascule avec résistance
a/ schéma, b/ circuit simulé et
c/ résultat de la simulation.

Ce circuit de mémorisation est appelé "registre" et il sera utilisé quand une valeur de fonction calculée (opérateur) doit être mémorisée. Dans ce cas il sera la réalisation de la variable spécialisée.

Remarque :
Le processus de lecture n'est pas affecté par l'introduction de la résistance.

3.4.7. Cas du bus complémenté

Le point mémoire peut être encore optimisé si on travaille avec un système de bus complémenté: il y aura une seule commande et la résistance ne sera plus nécessaire. Le schéma est donné dans la figure 3.23, où a et $\overline{a}$ représentent les deux lignes du bus. La commande est maintenant appelée "sélection": elle est utilisée pour la lecture et l'écriture.

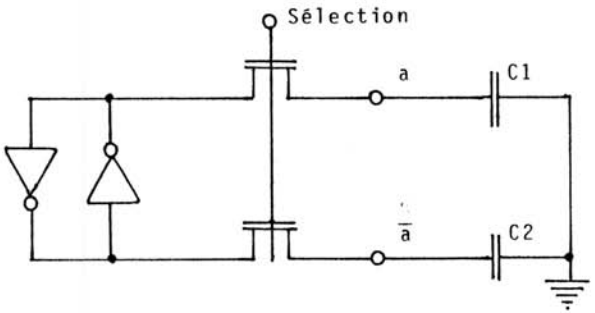


Figure 3.23 - Mémoire à bus complémenté.

Il y a encore d'autres avantages dans le système de bus complémenté :

- la consommation: la mémoire est attaquée symétriquement en écriture et son basculement est commandé par le bus. Des inverseurs de très faible consommation sont donc utilisés.
- si la charge du bus est trop importante, un amplificateur différentiel peut être raccordé au bus pour accélérer le processus de lecture.
- la valeur d'une variable et son complément seront disponibles sur toute la partie opérative.

Du point de vue électrique, le fonctionnement du bus bifilaire complémenté est le même que celui du bus unifilaire: précharge suivie d'une lecture qui décharge l'une ou l'autre des lignes du bus (figure 3.24).

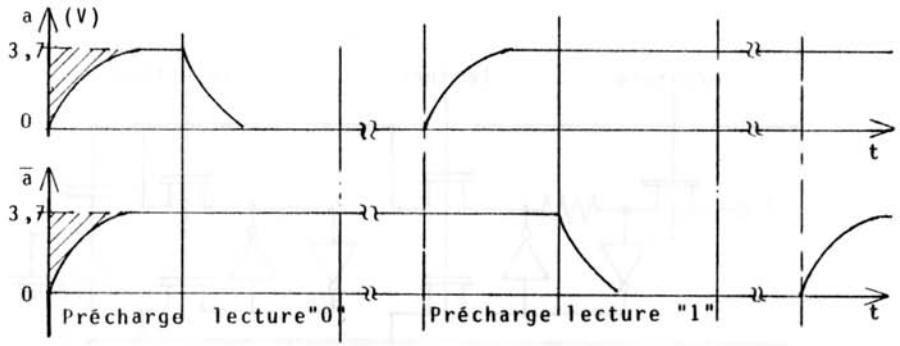


Figure 3.24 - Fonctionnement du bus complémenté.

3.4.8. Conséquences du système à bus complémenté

3.4.8.1. Les constantes

Avec le bus unifilaire, la constante zéro était un interrupteur à la masse et la constante "un" inexistante.

Sur le bus complémenté, il y aura toujours un interrupteur à la masse: on décharge la ligne "a" pour une constante zéro et on décharge la ligne "$\bar{a}$" pour une constante un (figure 3.25).

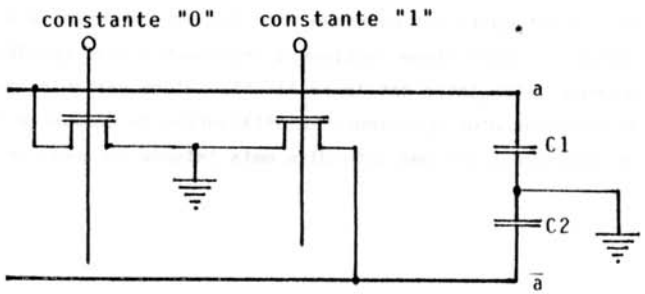


Figure 3.25 - Les constantes dans le système à bus complémenté

3.4.8.2. Les variables mémorisées

Les registres et les variables sont branchés selon le schéma de la figure 3.26. Les lignes de commandes du registre sont appelées lecture et écriture. La ligne de commande de la mémoire est appelée sélection et sert aussi bien pour la lecture que pour l'écriture.

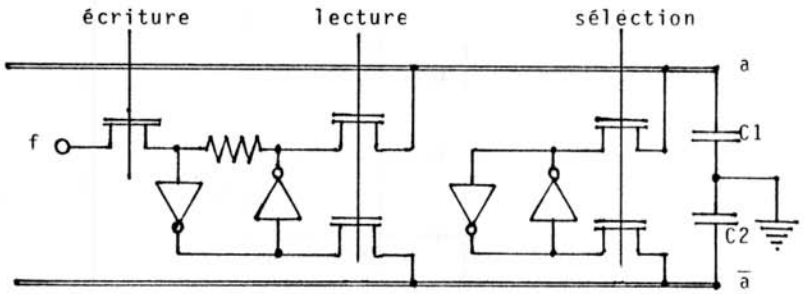


Figure 3.26 - Branchement du registre et de la mémoire.

- Le registre: l'écriture du registre se fait exactement comme dans le cas du bus unifilaire: la configuration du circuit pour l'écriture est la même. La lecture du registre sur le bus est semblable à la lecture du point mémoire.

Remarque :

La ligne de lecture du registre attaque un circuit identique à celui attaqué par la ligne sélection de la mémoire. On verra qu'il est possible de lire et d'écrire sur un point mémoire grâce à la seule ligne de sélection. On en conclut qu'il est possible d'écrire sur le registre à travers le chemin de lecture. Cette ligne continuera cependant à être appelée ligne de lecture puisque le registre est la réalisation d'une variable spécialisée qui reçoit le résultat d'un opérateur. L'utilisation du chemin de lecture pour écrire un registre n'est pas interdite mais laissée au choix du concepteur.

- **La mémoire:** l'écriture sur la mémoire se fait par l'activation du signal de sélection avec le bus stable contenant une valeur logique correcte, c'est-à-dire, une ligne à"1" et l'autre à "0". La lecture se fait en activant la ligne de sélection après que le bus ait été préchargé. On utilise le même circuit que dans le cas du bus unifilaire et on est donc sûr de ne pas faire basculer le point mémoire. La ligne du bus connectée à l'inverseur qui est à zéro est déchargée, l'état de l'autre ligne du bus n'est pas modifiée. La vitesse de variation de la tension du bus est inversement proportionnelle à la capacité de celui-ci. On voit le comportement du bus et du point mémoire pendant la lecture et l'écriture dans la simulation ci-dessous (figure 3.27).

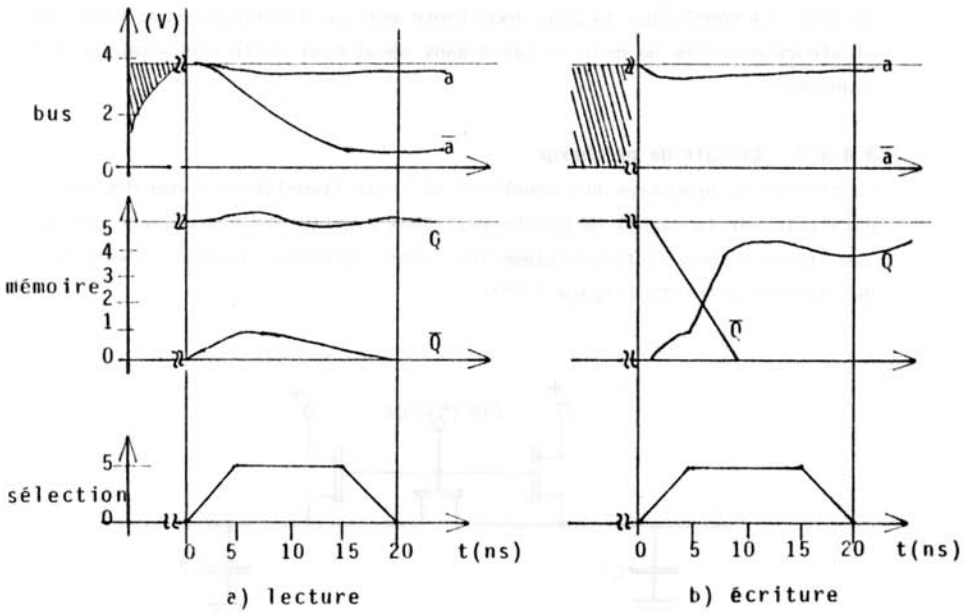


Figure 3.27 - Simulation du comportement du point mémoire
a/ lecture et b/ écriture.

Dans cette simulation, chaque ligne du bus est représentée par une capacité de 1 picofarad. Le point mémoire est constitué de deux inverseurs rebouclés, avec un MOS signal de 16/3,5 et un MOS charge 4/20. Les interrupteurs ont une géométrie 8/4.

En lecture, les lignes de bus sont écartées d'environ 2V pour une impulsion de sélection de 20ns (dont 5 ns pour la montée et autant pour la descente). Le point mémoire n'est pas sensiblement perturbé ; comme cela a été prévu dans l'étude de l'interrupteur, le côté du point mémoire qui est à "zéro" monte à presque 1 volt.

En écriture, le point mémoire bascule rapidement (pendant la montée du signal de sélection). Le bus perd environ 0,2 volts. L'écriture dégrade donc l'état du bus. La conclusion la plus importante est que l'écriture simultanée sur plusieurs mémoires ne peut se faire sans considérer cette accumulation d'influences.

3.4.8.3. Circuit de précharge

Le circuit de précharge est constitué de trois transistors commandés en parallèle par le signal de précharge. Deux transistors préchargent chacun les lignes du bus et le troisième les court-circuite, assurant l'égalité des niveaux atteints (figure 3.28).

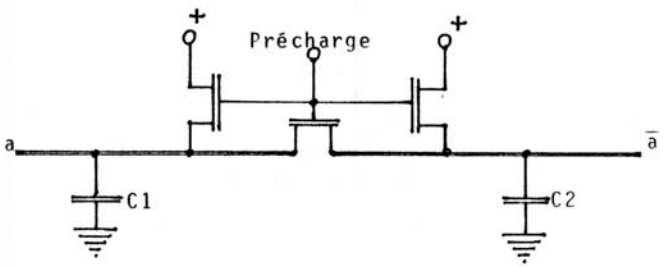


Figure 3.28 - Circuit de précharge pour bus bifilaire.

3.5. CONCLUSION

Dans ce chapître on a construit un ensemble de fonctions à partir des composants disponibles dans une technologie de circuits intégrés. Ces fonctions sont assemblées en vue de bâtir une partie opérative qui travaille sous la coordination d'une partie contrôle assurant la cohérence des opérations.

Les fonctions réalisées ont été déduites du modèle présenté dans le chapître précédent. Les opérateurs n'ont pas été traités en détail car ils sont trop dépendants de l'application. On a préféré laisser ce problème pour être étudié cas par cas: un opérateur "universel" est trop coûteux. La structure présentée est néamoins adaptée à l'insertion d'opérateurs particuliers comme on le verra dans l'annexe pour le circuit LISA.

Dans le prochain chapître, on développe la stratégie de dessin des fonctions étudiées ici ainsi que leur assemblage dans le but de construire des parties opératives.

Dessin 4

4 - DESSIN

4.1. INTRODUCTION

Le résultat de la conception d'un circuit intégré est le dessin d'un jeu de masques qui est utilisé dans le processus technologique en vue de produire le circuit désiré. La conception comprend aussi bien la définition du schéma logique que le dessin du jeu de masques correspondant.

Les circuits présentés dans le chapitre précédent seront donc dessinés. On étudie maintenant la technique mise en oeuvre pour passer d'une fonction au dessin des masques.

Le dessin des masques est fait pour une technologie précise. La technologie est définie par des "règles de dessin" qui seront présentées en annexe. Il existe des technologie qui sont assez proches les unes des autres en ce qui concerne ces règles, de telle façon que le dessin de la même fonction pour deux technologies différentes peut présenter des ressemblances accentuées : il y a des technologies qui préservent la disposition des éléments et d'autres qui préservent même la forme de la plupart des éléments (homologues). Les cellules sont dessinées dans une technologie mais beaucoup d'options prises resteraient valables pour d'autres technologies.

Dans le but d'un traitement automatique, on a essayé de formaliser les concepts traités. Le problème devient vite complexe ; on a traité les concepts de base qui sont utilisés dans la génération des parties opératives pour bien préciser leur signification. On définit ainsi la partie opérative et le langage d'assemblage.

4.2. INFLUENCE DES REGLES DE DESSIN

La première chose dont on a besoin pour dessiner les circuits est de connaître les règles de dessin. Ces règles définissent les contraintes qui doivent être satisfaites pour que le jeu de masques dessiné soit compatible avec les différentes phases du processus de fabrication.

Les règles de dessin donnent la dimension minimale des éléments et la distance minimale entre deux éléments différents. En NMOS, les éléments sont les conducteurs (polysilicium, diffusion et aluminium), les transistors et les constacts (diffusion-aluminium , diffusion-polysilicium et polysilicium-aluminium). Les technologies peuvent différer par le nombre d'éléments disponibles, la taille de ces éléments etc... Le dessin est donc fait dans une technologie donnée. Il est difficile de passer automatiquement un circuit d'une technologie à une autre technologie, même si elles sont assez semblables.

Dans la technologie utilisée, on dessine les masques suivants :

- métal,
- polysilicium,
- diffusion,
- contact (métal-diffusion et métal-polysilicium),
- pré-contact (diffusion-polysilicium),
- implantation.

Un transistor est défini par la superposition du polysilicium sur la diffusion.

4.3. STRUCTURE DU BUS

Le métal(aluminium)est de loin le moins résistif des conducteurs. Pour cette raison on a vu que les bus seraient réalisés si possible en métal. Les alimentations seront évidemment réalisées en métal.

Les commandes doivent atteindre les grilles des transistors commandés. On les

fera donc directement en polysilicium. Il est plus résistif et par conséquent une commande qui passe par plusieurs cellules sera retardée. Pour les parties opératives de 8 ou 16 bits, ce retard sera toutefois tolérable. On essaiera d'attaquer seulement des portes ou des transistors minimaux.

La topologie consistant à disposer l'aluminium dans un sens (pour les bus et les alimentations) et le polysilicium dans le sens perpendiculaire (pour les commandes) semble être une bonne stratégie: c'est celle qui est retenue.

Les lignes de métal sont disposées en bandes rectilignes. Sous une bande, on dessine un élément de la partie opérative. Ainsi l'accés aux bus et aux alimentations est facile. Pour une partie opérative dyadique, avec un système de bus complémenté, six lignes sont nécessaires par bit. Deux autres lignes sont ajoutées parce que le dessin de certaines cellules exige des ponts.

Les lignes d'aluminium sont disposées en des bandes rectilignes. Les règles technologiques définissent une largeur minimale des lignes d'aluminium (L) et un écartement minimal (E). La taille minimale d'un contact est donnée par un carré de CxC ; l'aluminium doit déborder de D la plage du contact (figure 4.1.a). Le pas d'aluminium est choisi :

$$PA = C + 2xD + E$$

car on ne veut pas avoir de restrictions pour la position des contacts (L est plus petit que C + 2D).

Les lignes d'alimentation sont élargies de Z pour les rendre moins résistives. Le pas de la partie opérative sera donc :

$$PP = 8 \times PA + 2xZ$$

La structure du bus est montrée dans la figure 4.1. Le pas de la partie opérative est défini par la structure du bus. Dans ce cas, il est égal à PP.

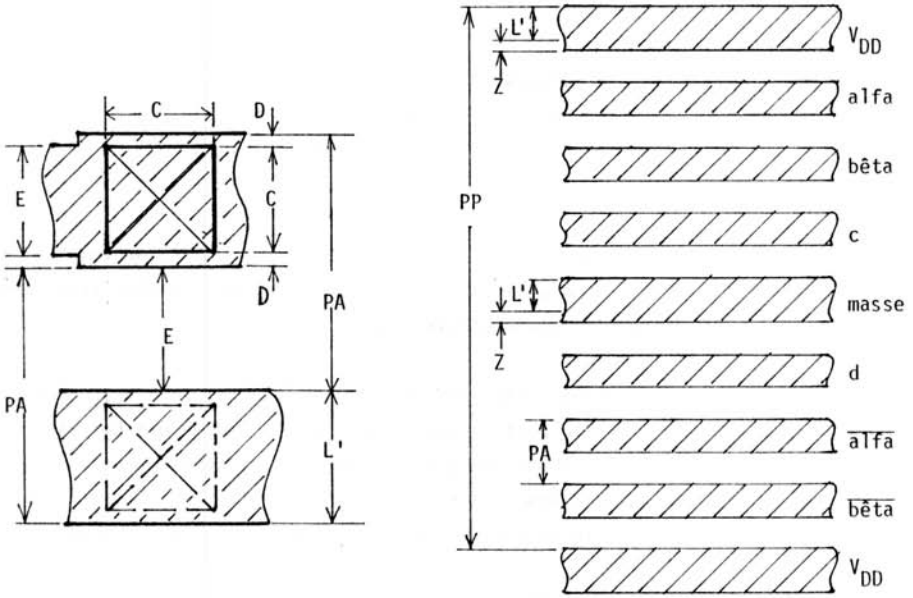


Figure 4.1. - Pas d'aluminium et structure de bus

Cette structure de bus est bien adaptée à l'implantation des fonctions simples et des points de RAM. Pour les fonctions plus complexes (au-delà de cinq portes), l'implantation sera plus délicate. Pour des éléments simples, comme une ROM, on n'a pas de contraintes. On remarque que dans les microprocesseurs, les fonctions complexes ne sont pas répétées tandis que la RAM, à elle seule, représente environ 30% de la surface de la partie opérative (et même davantage si l'on ne considère que les éléments actifs, c'est-à-dire si l'on exclut les lignes de connexion vers l'extérieur de la partie opérative) pour des microprocesseurs comme le 8085, Z80, Z800 et le 68000.

4.4. LES CELLULES

Une cellule est la réalisation d'une fonction. Les composants de la cellule sont dessinés sous la structure de bus qu'on vient de définir. Une cellule utilise une surface (un rectangle, pour le moment) de dimensions "l" et "h" (figure 4.2). Les lignes de commandes et le résultat du calcul des prédicats traversent la cellule dans le sens vertical tandis que les lignes de bus et d'alimentation traversent la cellule dans le sens horizontal. La hauteur des cellules est appelée le pas et correspond à la hauteur de la structure de bus. Un ensemble de cellules où toutes les cellules ont le même pas et utilisent les mêmes conventions pour les bus et les alimentations, constitue une famille. Les conventions concernent le comportement électrique et la disposition physique des lignes. La disposition physique est définie par les ordonnées et la largeur des différentes lignes: y1, y2 ... yn et Δy1, Δy2 ... Δyn dans la figure 4.2.

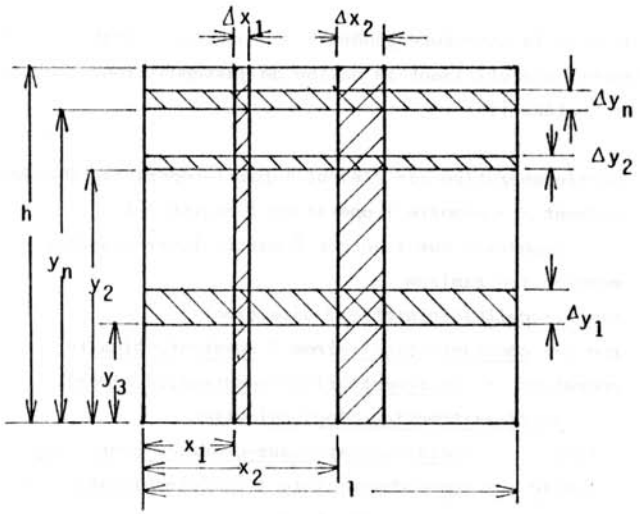


Figure 4.2. - La cellule

La largeur de la cellule "l" est libre et dépend de sa complexité entre autres. Les lignes de commeande et les prédicats sont différentes d'une cellule à l'autre en nombre, en position (x1, x2) et en dimension (Δx1, Δx2). Ces lignes sont choisies en accord avec la fonction exécutée. Pour la même cellule, la configuration des lignes doit être la même aux deux bords supérieur et inférieur.

Remarque :
Ces restrictions ne concernent que les bords des cellules. On est par contre relativement libre en ce qui concerne son intérieur.

4.5. LA PARTIE OPERATIVE

Une partie opérative est considérée, du point de vue graphique, d'un ensemble de cellules juxtaposées. On a vu que les cellules peuvent travailler ensemble pourvu que des signaux d'activation cohérents leur soient fournis. Pour construire une partie opérative, il ne nous reste donc qu'à assembler les cellules.

On peut décrire la structure générale des parties opératives à l'aide d'une forme grammaticale utilisant la notion de juxtapositions (notées||) et d'alternatives (notées |) :

partie-opérative ::= élément | partie-opérative || élément
élément ::= mémoire | opérateur || registre |
connexion-bus | entrée | sortie |entrée-sortie
mémoire ::= ram|rom
ram ::= cellule-bistable || sélection
rom ::= constante-binaire|rom || constante-binaire
opérateur ::= incrémenteur|décrémenteur|décaleur|
unité-arithmétique-logique|autres
autres ::= opérateurs-spéciaux-dessinés-pour-l'application
registre ::= commande-écriture || cellule-bistable || sélection
connexion-bus ::= interrupteur
entrée ::= ligne-externe || amplificateur || registre
ligne-externe ::= conducteur-dépassant-les-limites-de-la-partie-opérative

sortie ::= ram || amplificateur || ligne-externe
entrée-sortie ::= ram || amplificateur
ligne-externe || amplificateur || registre
amplificateur ::= amplificateur-à-deux-états|
amplificateur-à-trois-états

Remarque :
Les lignes externes sont dessinées jusqu'aux bords de la partie opérative, créant un chemin d'accés à l'intérieur. Le concepteur du circuit est reponsable de les prolonger jusqu'aux éléments extérieurs auxquels elles sont connectées.

Outil de génération

L'entrée du système de conception est une liste des éléments de la partie opérative à générer précédée d'une entête précisant le type (la famille) et le nombre de bits de la partie opérative. La sortie est un texte dans le langage graphique contenant les appels de figures enregistrées (les descriptions des cellules) avec l'emplacement absolu qu'elles occupent.

Le traitement est fait en deux étapes. Dans une première étape, un programme (dit de haut niveau) transforme la liste de fonctions fournies par l'utilisateur en une séquence de noms de figures indiquant la position relative qu'elles occupent, les unes par rapport aux autres. Cette séquence, dont la syntaxe est définie par la suite, est analysée par un autre programme qui calcule la position absolue des figures et génère le texte dans le langage graphique.

Deux raisons nous ont amené à prendre cette décision :

a/ décomposition des cellules : Si l'assemblage des cellules par juxtaposition est une opération facile à exécuter, elle entraîne, malheureusement, des pertes considérables de surface: les cellules doivent être dessinées de telle sorte que tous leurs composants soient suffisamment éloignés des bords pour que les règles de dessin soient respectées après l'assemblage. Il en résulte des

bandes vides entre les cellules. Pour cela, certaines cellules (pas toutes) ont été décomposées en des figures plus simples appelées briques. Les briques n'ont pas nécessairement un sens fonctionnel ; leur assemblage, cependant, en a un. On permet aux briques de se superposer, partiellement ou complètement. Toute figure élémentaire sera appelée brique, à partir de maintenant, sans se soucier si elle a, ou n'a pas, de sens au niveau fonctionnel.

b/ inclusion d'une fonction : Une fonction non connue par le programme de haut niveau peut être insérée dans la séquence générée à l'aide d'ure insertion dans le fichier de sortie. Il suffit d'ajouter à la bonne place et avec la syntaxe appropriée, le nom de la fonction à insérer: le programme de calcul de la position des briques les placera au bon endroit. Evidemment, le nom recherché doit exister dans le catalogue de briques car le programme y cherche la taille des briques.

4.5.1. Partie opérative de un bit

Les cellules peuvent être juxtaposées dans le sens horizontal. On s'assure, par des restrictions que les bus et les alimentations ne souffriront pas de coupure: on assemble donc les cellules sur un système de bus continu (rails).

Chaque cellule assure une fonction sur un seul bit: une constante, une variable mémorisée, etc... Leur assemblage en une bande constitue une partie opérative de un bit.

Cette partie opérative de un bit communique avec la partie contrôle par les lignes de commande et des prédicats qu'on appelle, improprement, signaux d'interface. La signification et la position de ces signaux sont dites topologie des signaux d'interface (figure 4.3). La partie opérative obtenue est un rectangle de dimensions h et lp, où h est le pas et lp est la somme des largeurs des cellules.

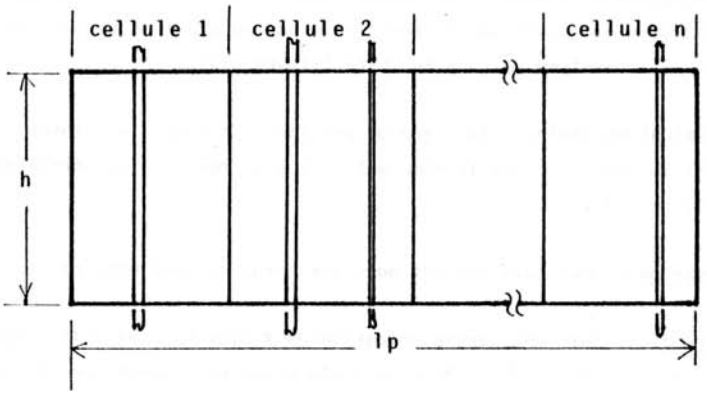


Figure 4.3. - Partie opérative de un bit.

4.5.2. Partie opérative de n bits

Plusieurs parties opératives de un bit, identiques, peuvent être accolées et commandées par la même partie contrôle, puisqu'elles ont la même topologie des signaux d'interface. On peut donc les empiler l'une sur l'autre, de telle sorte que ces signaux se raccordent. Si elles travaillent sur les mêmes données, elle exécuteront en parallèle les mêmes actions et il n'y aura pas d'incohérence entre les prédicats calculés.

On parle d'une partie opérative n bits dans le sens qu'elle traite des mots de n bits.

Dans une partie opérative qui travaille avec plusieurs bits, il est utile de générer des prédicats relatifs aux mots (certaines configurations de bits). Pour le calcul de prédicats sur les mots on utilisera deux stratégies: la fonction distribuée et el calcul en chaîne.

- **Fonction distribuée** : les règles pour la topologie des commandes et des prédicats se maintiennent, mais la valeur du prédicat ne peut être connue que par l'analyse de tous les bits (figure 4.4).

- **Calcul en chaîne** : la ligne du prédicat est coupée à l'intérieur de la cellule: elle n'a pas la même valeur à l'entrée et à la sortie de la cellule (figure 4.4).

Remarque : Les cellules qui sont aux bords peuvent être différentes.

Exemple : Soit une partie opérative de 4 bits (figure 4.4) composée de cellules A, B, ... E . A, C et E n'ont pas de rapport avec les bits voisins ; B réalise un "OU" distribué et D calcule la parité en chaîne.

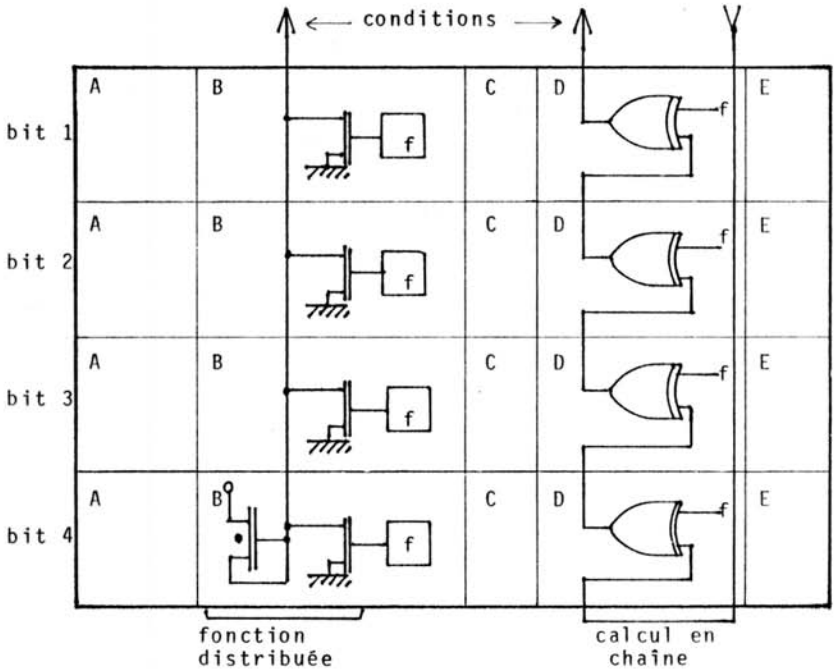


Fig 4.4 Partie Opérative 4 bits - Exemple de cellules

4.6. LE CATALOGUE DE CELLULES

Le catalogue de cellules est une bibliothèque ouverte dans le sens qu'elle croît à mesure que des cellules y sont ajoutées à partir des circuits en développement. Les règles à suivre sont :

- définition de la fonction et du circuit,
- dessin de la cellule,
- découpage en briques si nécessaire,
- description et stockage des briques dans le catalogue,
- indication au programme de haut niveau du nom et des règles d'assemblage correspondantes.

On a déjà une liste de cellules à réaliser, la technologie et la structure des bus sont définies, on passe donc au dessin d'une famille de cellules. Dans le but d'optimiser l'utilisation de la surface, les dessins sont faits de plusieurs manières différentes afin de choisir la meilleure. Pour comparer ces formes et pour ne pas retomber sur des solutions déjà abandonnées, le dessin sur papier est plus pratique. Les cellules sont en général assez petites. Elles comportent, en moyenne, une centaine de rectangles. La description des cellules ne prend pas plus de dix pour cent du temps de dessin pour les cellules que l'on a essayé de bien optimiser. La description des cellules a été faite dans le système graphique LUCIE [JER 80].

Les cellules sont connues par leur nom dans le système graphique: c'est le nom utilisé pour les assembler au moment de construire la partie opérative.

4.7. LE PROGRAMME DE HAUT NIVEAU

Le programme de haut niveau transforme les appels de fonction en une liste qui définit l'assemblage des briques. Le PHN (Programme de Haut Niveau) fait ainsi, implicitement, une traduction car le nom des fonctions ne correspond pas toujours au nom des briques qui les réalisent. Il est donc invisible pour

l'utilisateur que la fonction soit réalisée par une ou plusieurs briques.
Le PHN évolue en même temps que le catalogue de cellules: pour chaque fonction réalisée, un nouveau nom est ajouté à la liste des noms reconnus par le programme ; à ce nom est liée une procédure qui réalise l'algorithme d'assemblage.
Le programme travaille en interactif. L'utilisateur fournit au début le nom de la partie opérative et le nombre de bits (largeur du mot). Le nom de la partie opérative devient le nom du fichier de sortie. Les fonctions sont ensuite analysées une à une et traduites en des appels de briques concaténées au fichier de sortie.

Le corps du programme est simplement une boucle où le nom de fonction reçu en entrée est comparé à une constante alphanumérique (le nom connu) et qui génère les appeles de sous-programmes correspondant. Ajouter un nom, signifie donc l'insérer dans la suite de tests et ajouter une routine qui le traduit.

Un langage pour ce niveau n'a pas été défini. Il semble plus simple de considérer les informations supplémentaires dont on peut avoir besoin, comme des paramètres. C'est le rôle du sous-programme de faire l'analyse de ces paramètres. Le concepteur de la brique a donc toute liberté de définir la syntaxe et la signification de ces données. Cette solution a été satisfaisante pour notre cas (un seul concepteur et un nombre réduit de fonctions) mais pour un programme plus complexe, il serait intéressant de définir un langage.

Exemple :
On reprend l'exemple de la mémoire car il est intéressant sur l'aspect du PHN. Des fonctions non régulières, comme générer une constante ou une sortie, donnent des chaînes trop longues et d'autres qui ne se répètent pas donnent des expressions trop simples. Par exemple on appelle ici la fonction mémoire "ramd" pour "mémoire double accés". C'est une mémoire spécifique dont le circuit et le dessin sont bien précis (une autre mémoire double accés avec le même circuit mais qui aurait quelque modification dans le dessin serait appelée autrement). On veut construire des blocs mémoire avec quelques mots en simple accés sur un bus, d'autres en simple accés sur l'autre bus et d'autres en double accés.

On a défini que le nom de la fonction sera suivi par un entier n, suivi par n entiers i1, i2,... in :

ramd n (i1 i2 ... in).

ramd est le nom de la fonction.
n indique le nombre de mots du bloc mémoire. i1, i2,... in valent 1 ou 2 ou 3 et indiquent le mode de connexion de chaque cellule en partant de la gauche.

ik = 1 : mémoire simple accés sur le bus alfa,

ik = 2 : mémoire simple accés sur le bus bêta,

ik = 3 : mémoire double accés.

Pour optimiser l'utilisation de la surface, l'algorithme de traduction essaie de placer le plus possible les briques contact, de manière qu'elles puissent être partagées par deux cellules voisines. On teste les cellules pour voir si on peut les placer de telle façon que ce partage soit possible. Dans l'état actuel, cet algorithme ne regarde qu'une cellule en avant et ne se permet pas de changer leur ordre.

Le problème qui apparaît avec technique est celui des bords entre les fonctions. Il n'est pas commode de dessiner les briques en tenant compte de toutes les possibilités de voisinage avec toutes les autres briques. Le voisinage entre fonctions doit être traité par le programme principal et non pour chaque procédure. La solution adoptée (et qui n'a pas été programmée) est de créer des classes d'interface. Elles seront traitées plus loin dans ce chapître.

4.8. LANGAGE D'ASSEMBLAGE

Le Langage d'Assemblage de Briques (LAB) est un langage destiné à assembler des figures préalablement décrites et emmagasinées dans une bibliothèque.
Il est essentiel que le programme assembleur puisse accéder à la description des briques pour y récupérer leur encombrement sous la forme de deux paramètres: dimension "x" et dimension "y". Ainsi, la brique sera considérée comme un rectangle ayant l'encombrement "x" par "y".

Les briques sont dessinées dans le but de construire des parties opératives, elles aussi rectangulaires. Un langage qui ne travaille qu'avec des rectangles peut être défini par des règles très simples ; d'autant plus qu'on va restreindre également les résultats intermédiaires à des ectangles: deux rectangles donnent un rectangle quand ils sont combinés par un opérateur. Le langage établit un ordre de placement des briques d'où on peut déduire l'emplacement absolu, basé sur l'encombrement des briques.

La partie opérative croît dans un plan orienté de gauche à droite, et du bas vers le haut, par assemblage de briques ou figures. Les opérateurs du langage sont des opérateurs binaires qui indiquent si l'élément qui est ajouté doit être placé à droite, ou en haut de celui qui existe déjà. La juxtaposition exacte définie par les opérateurs peut être modifiée par des paramètres. Il y a aussi des paramètres pour changer l'orientation de la brique à placer (par défaut, elles sont placées comme décrites).

Définition du langage

La syntaxe du langage d'assemblage de briques est définie par les règles suivantes (les spécifications à l'intérieur des crochets sont optionelles) :

```
<fig> ::= <el>|<fig> <op> <el>
<el>  ::= <var> [!liste de paramètres!]
<var> ::= <brique>|{<fig>}|<rec>
<op>  ::= ↑|→| +
<liste de paramètres> ::= <paramètre>[<paramètre>]*
<paramètre> ::=  dx [=]<ae> | dy [=]<ae>
                 symx|symy|rot1|rot2|
                 ofx [=]<ae>|ofy [=]<ae>
                 abx [=]<ae>|aby [=]<ae>
<brique> ::= nom de brique
<ae> ::= expression arithmétique entière
<rec> ::= rectangle vide
```

Le langage reconnaît des structures parenthésées qui sont interprétées.
Exemple: Interprétation de l'expression {{a}→b↑c}→{{d}↑e}. Un nombre n appliqué à un arc signifie n applications de déduction.

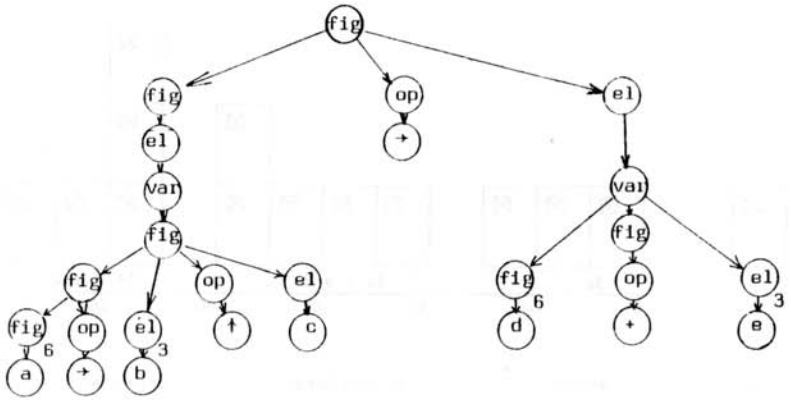


Figure 4.5. - Arbre d'interprétation

Sémantique des opérateurs

Les opérateurs définissent l'emplacement relatif de deux opérandes géométriques ; ils sont interprétés comme suit :

Opérateur "→"

Etant donnés deux opérandes, u de dimension dx_u et dy_u et v de dimension dx_v et dy_v , liés par l'opérateur "→", l'expression "u→v" signifie: placer v à droite de u. Le résultat est une figure de dimension

$$dx_f = dx_u + dx_v \quad \text{et} \quad dy_f = \max(dy_u, dy_v).$$

Opérateur "↑"

L'opérateur "↑" appliqué à deux opérandes comme définis auparavant est écrit "u↑v" et signifie: placer v au-dessus de u. Le résultat est une figure de dimension

$$dx_f = \max(dx_u, dx_v) \quad \text{et} \quad dy_f = dy_u + dy_v.$$

Opérateur "+"

L'opérateur "+" assemble deux briques par superposition. "u+v" signifie placer u sur v. Le résultat est une figure de dimension

$$dx_f = \max(dx_u, dx_v) \quad \text{et} \quad dy_f = \max(dy_u, dy_v).$$

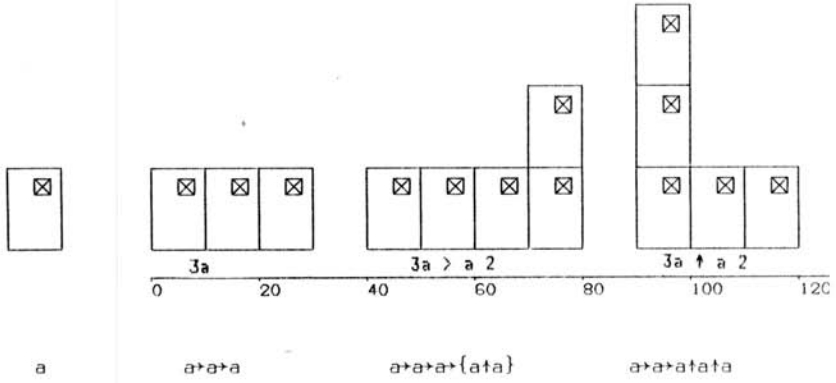


Figure 4.6. - Exemple d'application des opérateurs → et ↑

LES PARAMETRES

Un élément ("el" dans la définition du langage) peut avoir des paramètres. Ces paramètres modifient l'opération de base définie par l'opérateur graphique.

Sémantique des paramètres

- symx et symy :

Ces paramètres imposent une symétrie (par rapport à un axe x ou y). Il n'y a pas d'arguments dans les symétries. L'élément, figure ou brique, sera placé dans la même position qu'il occuperait si la symétrie n'avait pas été spécifiée. Les symétries ne changent pas les dimensions x et y de l'élément, elles ne changent pas non plus l'encombrement de la figure résultante.

- rot1 et rot2 :

Ces paramètres établissent une opération de rotation de $\pi/2$ et $-\pi 2$, respectivement. Les dimensions x et y de l'élément sont échangés. L'origine (les coordonnées du point plus à gauche et plus en bas) est maintenue.

- dx=ae et dy=ae :

Les dimensions de l'élément utilisées pour le calcul de l'encombrement de la figure résultante ne sont pas prises comme taille réelle de celui-ci mais remplacées par la valeur de l'expression.

- ofx=ae et ofy=ae :

Au moment de l'assemblage, les entités graphiques, considérées comme des rectangles, sont placées exactement côte à côte, c'est-à-dire qu'un côté du rectangle englobant une figure coïncidera avec un côté du rectangle englobant l'autre. Cette opération peut être modifiée par les paramètres d'écartement. Les figures seront écartées de "ae" unités dans le sens x (ou y) selon qu'on spécifie ofx (ou ofy). Si ae<0, une superposition aura lieu.

- abx=ae et aby=ae :

Ces paramètres imposent une coordonnée absolue à un élément (dans l'espace des coordonnées où se construit la figure). Tous les éléments qui viennent s'accoler par les opérateurs (→,↑,+) à un élément dont les coordonnées sont forcées à être absolues, deviennent, de fait, à coordonnées absolues. Les opérateurs n'ont pas d'effet sur ces éléments (ceux qui ont une coordonnée absolue spécifiée) mais la syntaxe du langage exige qu'un opérateur soit quand même spécifié. Si abx et ofx sont présents dans la même liste de paramètres, alors la position finale sera abx + ofx. Dans une liste, les paramètres sont pris en compte dans leur ordre d'écriture. Cet ordre n'est pas toujours indifférent.

DESCRIPTION DU PROGRAMME D'ASSEMBLAGE

L'outil d'assemblage est un programme écrit en PL1 (actuellement sur MULTICS) qui lit un fichier (segment) contenant une description LAB et qui génère un fichier de sortie en source LUCIE. A chaque nom de brique dans la description d'entrée, correspond une figure traduite (au sens de LUCIE).

Comme dans LUCIE il existe deux types de figures: les figures internes et les figures externes, les noms sont cherchés d'abord dans la liste de figures internes et ensuite, le cas échéant, dans l'environnement (directory) de l'utilisateur.

- Figure externe

Le nom de brique est utilisé pour accéder, dans l'espace de l'utilisateur, à un segment de même nom, contenant la description de la figure (géométrique) et, en particulier, la taille de cette figure. La taille (dimension x et dimension y) est utilisée pour le calcul des emplacements.
Dans la phase de génération du texte LUCIE, on produit la chaîne :

"figext <nom> (x,y)"

LUCIE accédera, lui aussi, à ce segment au moment de construire la figure finale.

- Figure interne

Il est convenu que l'utilisateur crée un segment "lfigin" contenant la description de la figure "figint". La traduction de "lfigin" par le traducteur LUCIE donne origine au fichier "figint" contenant la description de toutes les figures internes et leur encombrement. Ces figures constituent un premier ensemble de noms connus par l'assembleur.

Le but de ce traitement spécial est d'éviter la prolifération de petites figures dans l'espace disque de l'utilisateur. Ces petites figures sont en général des terminaisons ou des cellules de programmation qui ne sont décrites que par quelques rectangles.

Si une figure interne est utilisée, alors tout le segment "lfigin" est recopié en en-tête avant de commencer les appels de figure.

Les figures internes sont appelées par :

"fig nom (x,y)

On pourrait recopier sélectivement les figures utilisées (au lieu de toutes les figures) mais on ne gagne pas beaucoup puisque le segment doit préexister pour être traduit et qu'il ne peut contenir plus que 30 figures dans la version actuelle de LUCIE. Par contre, passer le nom du fichier en paramètre est une amélioration possible, simple à réaliser.

Exemple de structure du fichier "lfigin" :

```
fig figint
    fig a
    ...
    ffig
    fig b
    ...
    ffig
    ...
ffig
```

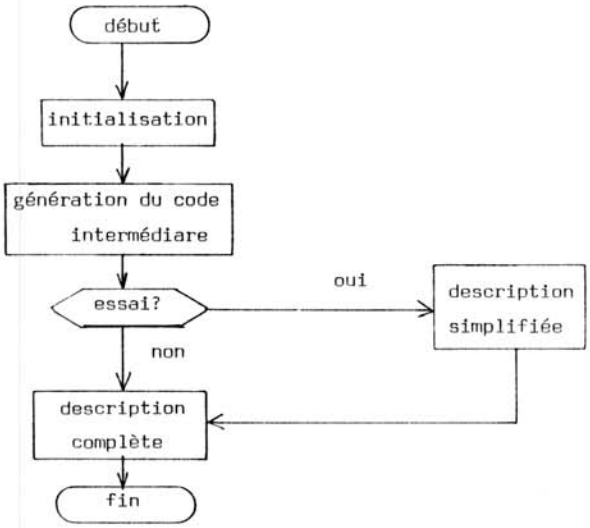
Les figures a et b sont des figures internes. On ne peut pas appeler des figures à un niveau d'imbrication supérieur à 2 . S'il existe deux figures de même nom, une interne et une autre externe, l'interne est utilisée.

L'appel à une figure qui n'existe pas est assemblé comme ayant un encombrement nul.

Les segments figures ne sont pas distincts des autres segments de l'utilisateur (programmes, texte, etc...). La vérification faite consiste à comparer le nom fourni avec le nom contenu dans le premier enregistrement pour les figures externes. Les figures internes sont prises dans un segment dont le nom peut être vérifié comme dans le cas des figures externes.

Structure du programme

Le programme contient quatre phases : initialisation, génération du code intermédiaire, génération (conditionnelle) de la description simplifiée et génération de la description complète.



- Initialisation

Le programme est appelé avec trois paramètres :

assemb p1 p2 p3

p1 est le fichier d'entrée,

p2 est le fichier de sortie,

p3 est, soit "essai" soit néant.

p2 est en fait le nom de la figure qui sera générée par la phase suivante, c'est-à-dire la traduction. Pour éviter le conflit de nom qui conduirait à la destruction du fichier source LUCIE, la convention suivante est établie: p2 est le nom de la figure finale et "1"|| p2 est le fichier qui contient la description (texte) LUCIE. En d'autres termes, on crée le fichier "1"|| p2 dont le premier enregistrement est

fig p2.

Après la traduction on aura alors la figure p2 dans le segment de nom p2.

Le paramètre p3 ou bien est égal à "essai" ou bien est invalide. Si p3 est égal à "essai", un fichier de nom essai est créé, qui contient la description de la figure p2 où les appels de figures sont remplacés par des rectangles de même taille que les figures correspondantes.

- Génération du code intermédiaire

La génération du code intermédiaire correspond à la construction d'une structure de données qui représente la figure décrite par le langage d'assemblage de briques. En particulier, chaque ouverture de figure par le caractère "{" cause l'empilement d'un nouveau contexte. Chaque ensemble de données représente une figure. Les éléments (au sens de LAB) sont placés relativement à la figure immédiatement englobante en considérant leur déplacement par rapport au dernier niveau d'imbrication franchi. Le code intermédiaire contient toutes les informations de la figure décrite: le nom des figures appelées, leur emplacement, les opérations géométriques... Il est alors possible de passer de ce code à un texte dans une forme acceptée par un langage graphique, pourvu que ce langage admette des appels de blocs externes.

- Algorithme d'interprétation de LAB

L'interprétation d'une description en LAB se fait à l'aide de trois procédures: figure, élément et variable ; elles sont schématiquement décrites ci-dessous. Dans ces procédures, d'autres procédures sont appelées :

. opérateur : est une fonction qui est vraie lorsque le caractère d'entrée est un opérateur : ↑ , → ou + .

. répétitionx et répétitiony : pour alléger la description des constructions du type a→a→a→...→a (n fois) peut s'écrire: n a. De la même façon a↑a↑a↑...↑a (n fois) peut être écrit: a n . Dans les deux cas, n est une expression arithmétique. Répétion évalue l'expression arithmétique ; si l'expression est absente, la valeur 1 est retournée.

. paramètres : analyse la liste de paramètres.

Corps de l'algorithme

```
procédure figure
    call élément
    tant que opérateur faire
        début
        call élément
        call opération
        fin
    fin figure
procédure élément
    répétitionx
    call variable
    call paramètres
    répétitiony
    fin élément
procédure variable
    si "{"
    alors faire
        début
        call figure
        sauter "}"
        fin
    sinon call brique
    fin variable
```

Structure de données

La structure de données contient un vecteur de données pour chaque composant traité. La structure, appelée "repères", regroupe les données suivantes pour chaque vecteur :

1 - repères

2 - ref : numéro du dernier vecteur utilisé si c'est une figure, sinon numéro de brique dans la liste interne des briques utilisées.

2 - offsetx : déplacement par rapport à la figure immédiatement englobante dans le sens x.
2 - offsety : idem dans le sens y.
2 - deltax : taille de la brique ou figure (sens x).
2 - deltay : taille dans le sens y,
2 - repx : nombre de fois que l'élément doit être répété dans le sens x,
2 - repy : idem dans le sens y,
2 - doffx : déplacement à ajouter au déplacement calculé par le programme (offsetx) dans le sens x,
2 - doffy : idemn dans le sens y,
2 - carct : caractéristique de déplacement,
3 - cnum : nombre d'opérations graphiques requises,
3 - géom (I) : opérations sollicitées :
géom = 1 symétrie en x
géom = 2 symétrie en y
géom = 3 rotation $\pi/2$
géom = 4 rotation de $-\pi/2$
3 - ctype : type de l'élément considéré :
ctype = 0 rectangle
ctype = 1 brique externe
ctype = 2 figure
ctype = 3 brique interne

Cette structure est traitée comme une liste imbriquée où l'on retrouve la forme d'arbre correspondant au langage d'assemblage d'origine. Les noeuds sont les figures et les feuilles, les briques et les rectangles.

Dès qu'une brique est utilisée, elle est enregistrée dans une liste (une sorte de liste de briques connues). Pour chaque nom de brique dans la chaîne d'entrée, la recherche débute toujours par cette liste. Elle a la forme suivante :

1 - brique
2 - nom (6 caractères): nom de brique

2 - btype := 1 brique externe
= 2 brique interne
2 - dimx : dimension en x de la brique
2 - dimy : dimension en y de la brique

La liste des briques est initialisée avec :
nom(1) = "rec"
nom(2) = "fim"
"fim" est une marque de fin de liste et chaque nouvelle brique utilisée est ajoutée à la liste avant l'étiquette "fim". Un nom de brique trouvé dans la chaîne d'entrée est comparé avec les noms existants jusqu'à cette étiquette. Si il n'y apparaît pas, la recherche se poursuit dans la liste de briques internes et ensuite dans le "directory" de l'utilisateur. Le numéro de brique (ref dans la structure du code intermédiare) identifie une brique par l'ordre qu'elle occupe dans cette liste.

Le nom "fim" ne peut pas être utilisé comme nom de brique.

Les briques internes sont gardées dans une liste qui contient :
1 - briqint
2 - nomint (6 caractères): nom de brique (pris dans la figure figint)
2 - dx : dimension en x,
2 - dy : dimension en y.

Génération du texte

Une routine spécifique transpose le code intermédiaire dans une description textuelle acceptable par un langage graphique. Le code intermédiaire contient une description complète de la figure et il est, a priori, possible de le transformer en tout langage qui traite les appels de blocs extérieurs et exécute les opérations primitives de placement, de rotation, de symétrie, de répétition. Actuellement, seul du langage LUCIE est généré.

La génération du texte LUCIE suit à peu près les étapes de la création du code intermédiaire. Dans LUCIE, les opérations de répétition, de symétrie et

de rotation valent une ouverture de bloc. Un noeud dans la structure du code inermédiaire correspond à un bloc : il sera alors enfermé dans une séquence rep...frep, sym...fsym, etc... s'il contient les opérations géométriques correspondantes.

Les opérations géométriques sont générées dans l'ordre suivant: répétition sur l'axe x, répétition sur l'axe y, autres opérations dans leur ordre d'écriture dans la liste d'entrée en langage LAB.

L'algorithme de génération du texte LUCIE a la forme suivante :

```
procédure fig (ofx,ofy,niv)
call opérations_géométriques
si figure
alors call fig (ofx+offsetx, ofy+offsety, niv+1)
sinon répéter pour tous les éléments de la figure
     début
          si nom="rec"
          alors écrire ("rec,...")
          sinon
            début
              call opérations_géométriques
              si briquinterne alors f="fig"
              si briquesterne alors f="figext"
              écrire (f,ofx+offsetx,ofy+offsety)
              call fin_opérations_géométriques
            fin
     fin
call fin_opérations_géométriques
fin fig.
```

4.9. EXEMPLE

4.9.1. Dessin de la mémoire

Nous allons maintenant montrer la génération des masques de la mémoire étudiée dans le chapitre précédent comme exemple pour présenter la démarche, depuis la définition de la fonction jusqu'à la génération des masques de la partie opérative (on ne génère qu'un morceau de la partie opérative: le bloc mémoire).

Remarque :
La technologie est déjà choisie et la structure de bus est celle en double bus complémenté.

- Définition de la fonction et du circuit

On veut réaliser la fonction mémoire avec un accés simple sur un bus ou sur l'autre, ou un accés double sur les deux bus. La figure 4.7 montre le schéma logique (alpha et bêta sont les deux bus).

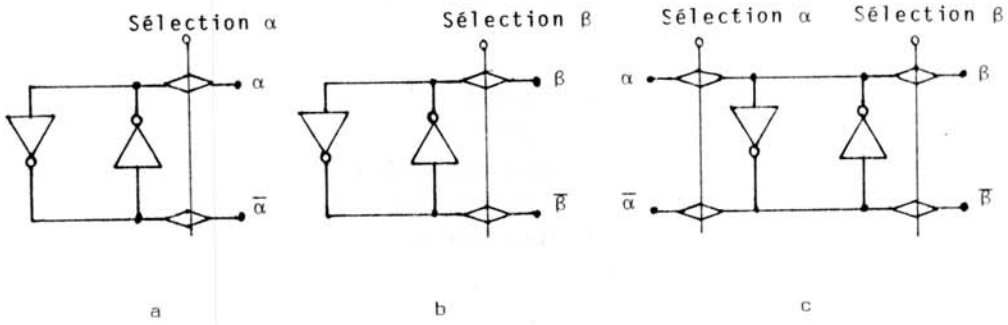


Figure 4.7 - Schéma logique de la mémoire sur une structure à double bus, a/simple accés alpha, b/simple accés bêta et c/double accés.

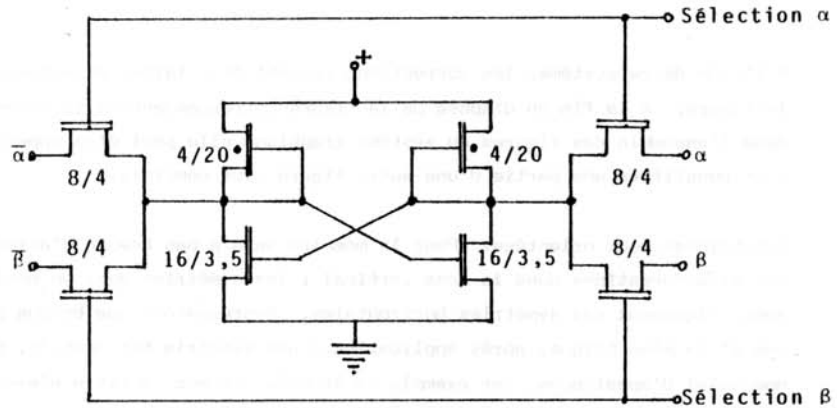


Figure 4.8. - Circuit de la mémoire(configuration double accés)

- Découpage en briques

Le concepteur a intérêt à placer les mots mémoires contigus. Les lignes de commande de la mémoire sont en général calculées par un décodeur. On est donc amené à la création de blocs mémoires dans les parties opératives. Pour avoir plus de souplesse dans la conception, la mémoire est faite à partir de cinq briques appelées ramcel, ca, cb, caa et cbb (figure 4.9) :

- ramcel est la brique de mémorisation (les deux inverseurs),
- ca et cb sont les sélections sur les bus alpha et bêta,
- caa et cbb, identiques à ca et cb, mais pour deux ramcel voisines,

Avec ces cinq briques, il est possible de construire toute configuration de mémoire (séquence de mots simple et double accés, dans n'importe quel ordre) tout en ayant une bonne optimisation de la surface.

Les briques sont décrites sous la forme d'un texte contenant tous les motifs dessinés et stockés dans un fichier. Ce texte est traduit par le système graphique qui génère une forme interne commune à tous ses programmes.

A l'aide de ce système, les corrections peuvent être faites directement sur la figure. A la fin on dispose de la figure correspondante à la brique, dans l'ensemble des figures du système graphique elle peut être appelée pour constituer une partie d'une autre figure plus complexe.

Les briques sont orientées. Pour la mémoire on n'a pas besoin d'effectuer des transformations dans le sens vertical ; les symétries dont on parle ici sont uniquement des symétries horizontales. Représentons une brique par son nom et la même brique, après application d'une symétrie horizontale, par le nom suivi d'apostrophe, par exemple ca et ca'. Citons, à titre d'exemple, quelques constructions possibles :

- ca' - ramcel - cb
- cb' - ramcel' - caa - ramcel
- ramcel - cbb - ramcel'

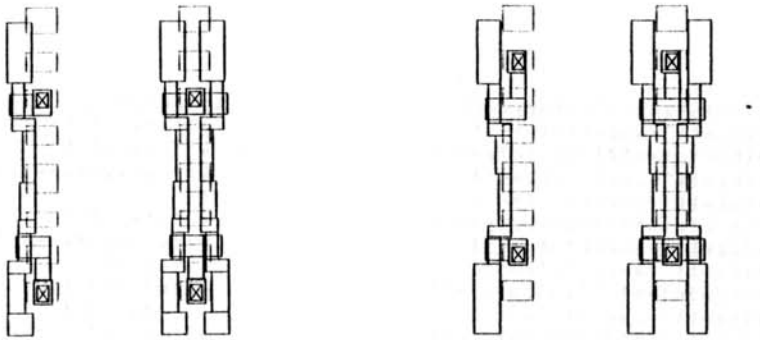
Remarque :
On ne mentionne pas, pour l'instant, de combien les briques doivent se superposer. Les constructions ramcel-ramcel ou ca-ramcel, par exemple, déterminent des briques indépendantes (elles ne se connectent pas).

Exemple :
Prenons, par exemple, un bloc mémoire pour une partie opérative de 4 bits, spécifié comme suit :

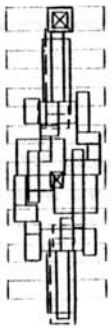
ramd 7 3 2 2 2 1 3 3

Cela signifie qu'on a besoin d'un bloc de 7 mots mémoire dont le premier est en double accés, les trois suivants sont en simple accés sur le bus bêta, le cinquième est en simple accés sur le bus alpha et les deux derniers en double accés.

A partir de ces données, le programme de haut niveau génère la séquence d'appels de briques dans le langage d'assemblage de briques, séquence montrée dans la figure 4.10.



cb cbb ca caa



ramcel

Figure 4.9 - Briques du point mémoire

```
(b20!ofx=1?!^ca!ofy=-?!                    4)!symx!
>(bcell^ramcel!ofy=-3!                      4)!ofx=-1?!
>(b20!ofx=12!^cbb!ofy=-?!                   4)!ofx=-12!
>(bcell^ramcel!ofy=-?!                      4)!symx,ofx=-12!
>(bcell^ramcel!ofy=-?!                      4)
>(b20!ofx=12!^cbb!ofy=-?!                   4)!ofx=-12!
>(bcell^ramcel!ofy=-?!                      4)!symx,ofx=-12!
>(bcell^ramcel!ofy=-?!                      4)!symx!
>(b20!ofx=1?!^caa!ofy=-?!                   4)!ofx=-1?!
>(bcell^ramcel!ofy=-?!                      4)!ofx=-1?!
>(b20!ofx=12!^cbb!ofy=-?!                   4)!ofx=-12!
>(bcell^ramcel!ofy=-?!                      4)!symx,ofx=-12!
>(b20!ofx=1?!^ca!ofy=-?!                    4)!ofx=-1?! .
```

Figure 4.10 - Séquence générée par le PHN.

Le programme d'assemblage de briques interprète la séquence de la figure 4.10 et génère du texte LUCIE, dont un extrait est montré dans la figure 4.11.

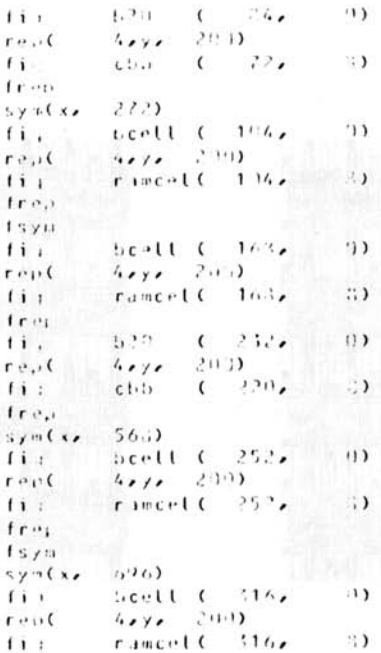


Figure 4.11 - Extrait du texte généré.

Le bloc mémoire généré apparaît dans la figure 4.12.a.

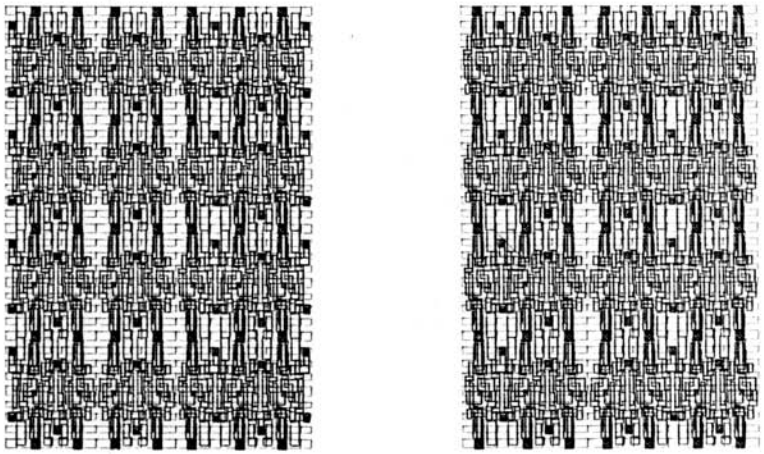
Remarque :

La distance entre deux cellules ramcel est à peu près constante. Dans cet exemple, le concepteur pourrait gagner de la surface (4% environ) en spécifiant autrement la mémoire, par exemple

ramd 7 1 3 2 2 3 3 2

Le bloc mémoire qui en résulte est montré dans la figure 4.12.b.

On peut aussi noter que les rangées de contacts qui bordent le bloc "a" de la figure 4.12.a. n'existent plus dans la figure 4.12.b. ; ils sont passés à l'intérieur.



a b

Figure 4.12. - Blocs mémoire générés
a/ pour la spécification 3 2 2 2 1 3 3
b/ pour la spécification 1 3 2 2 3 3 2

4.9.2. Génération d'une sortie

Pour montrer comment on génère des séquences du langage d'assemblage de briques, on a choisi le sous-programme qui construit des sorties perpendiculaires aux bus. C'est un sous-programme plus simple que celui qui interprète les appels de mémoire et il n'est pas trivial comme ceux qui génèrent une rangée de briques.

Les sorties sont des lignes en polysilicium connectées à la ligne de métal appelée "c" dans la structure de bus. Cette ligne fonctionne dans ce cas comme un pont métallique. On suppose qu'une rangée de cellules soit juxtaposée pour les piloter (voir entrées/sorties).

Les briques utilisées sont :

bs : base de la structure,
ssp : brique d'espacement,
sct : brique contact,
sli : brique ligne de polysilicium.

On demande une sortie par la phrase :

sorite n m1 m2 ...

où

n est le nombre de lignes de sortie,
m1, m2... sont les numéros des bits qu'on veut sortir.

Dans la figure 4.13, on a le listing du sous-programme.

Dans la figure 4.14 on a deux exemples de sorties pour une partie opérative de 4 bits, pour les deux demandes suivantes :

- sortie 3 1 3 4
- sortie 3 1 2 4.

Remarque :
Le premier caractère de la chaîne (">") a été enlevé et le point à la fin a été ajouté.

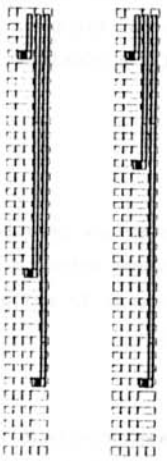
```
procedure sortie (xab :tabint ; ns : integer) ;

          xab : vecteur des sorties demandées

          ns  : numéro de sorties

   var a, x : integer ;
   begin
      writeln (fsortie, '>((', ns + 2, 'bs)') ;
      x:=ns;
      for a := nbit downto 1 do
         begin
            if x = 0
            then writeln (fsortie, '^(2ssp>', ns, 'sli)!ofy=-8!')
            else if xab [x] = a
               then begin
                     if x = ns
                     then writeln (fsortie, '^(', ns , 'ssp>sct)!ofy=-8!')
                     else writeln (fsortie, '^(', x , 'ssp>sct>', ns - x, 'sli)!ofy=-8!') ;
                     x := x - 1 ;
                  end
               else if x = ns
                  then writeln (fsortie, '^(', ns + 2, 'ssp)!ofy=-8!')
                  else writeln (fsortie, '^(', x + 2, 'ssp>', ns - x, 'sli)!ofy=-8!') ;
         end ;
      writeln (fsortie, ')') ;
   end ;
                                                    (*
```

Figure 4.13 - Listing du sous-programme générateur de sorties.



```
((          5bs)
"(          3ssp>sct)!ofy=-3!
"(          2ssp>sct>          1sli)!ofy=-3!
"(          3ssp>          2sli)!ofy=-3!
"(          1ssp>sct>          2sli)!ofy=-3!
) .
```

a.1

```
((          5bs)
"(          3ssp>sct)!ofy=-3!
"(          4ssp>          1sli)!ofy=-3!
"(          2ssp>sct>          1sli)!ofy=-3!
"(          1ssp>sct>          2sli)!ofy=-3!
) .
```

b.1

a.2 b.2

Figure 4.14 - Langage d'assemblage et dessin pour les sorties.
cas a/ sortie 3 1 3 4
b/ sortie 3 1 2 4

4.10. LES CLASSES D'INTERFACE

Il est très difficile de dessiner une famille de cellules de telle sorte que toute cellule s'encastre bien avec les cellules voisines pour toutes les possibilités d'assemblage. Dans le cas de la mémoire, par exemple, les briques sont optimisées pour bien s'assembler, mais il y a plusieurs configurations possibles pour le bord du bloc.

La stratégie adoptée est la suivante :

- Des classes d'interface sont définies. Les classes d'interface correspondent à une standardisation des bords verticaux des briques. Chaque brique a deux bords verticaux: droit et gauche. On dit que le bord gauche de la cellule "cel" appartient à la classe "x", par exemple, et on écrit :

$$g(cell) \in x.$$

- Une matrice carrée de n x n (où n est le nombre de classes) contient les conditions qui doivent être satisfaites quand deux bords sont mis en contact par assemblage.

La matrice pourrait être faite par toutes les cellules sans introduire les classes d'interface. Deux raisons nous ont amenés à introduire les classes d'interface :

a/ la table devient vite très grande et il faut l'actualiser pour chaque brique introduite ou modifiée ; il y a donc une source potentielle d'erreurs.

b/ la probabilité que deux briques s'assemblent de façon constructive est si faible qu'elle peut mettre en cause l'effort qu'on y apporte. Une façon constructive signifie ici que la surface occupée par deux briques assemblées est plus petite que la somme de leurs encombrements (mesurés par les rectangles englobants).

Encouragé par la facilité qu'apporte l'utilisation de classes déjà existantes (il n'a pas besoin d'écrire un algorithme d'assemblage), l'utilisateur s'efforcera de situer dans ces classes les bords de la brique qu'il dessine.

Evidemment le nombre de classes ne doit pas être trop petit sinon la contrainte devient trop restrictive.

Exemple de quelques classes :

- Classe 1 : aucune restriction. Les éléments de la brique sont dessinés librement dans le rectangle correspondant à la cellule ; ils peuvent même toucher les bords.
- Classe 2 : demi-garde. Tout élément est éloigné des bords d'une demi-garde la plus restrictive (sauf les bus, qui définissent alors l'encombrement).
- - Classe 3 : sortie en diffusion. C'est une classe 2 dans laquelle une sortie d'une porte est étendue jusqu'au bord (vertical). Elle est centrée sur la brique dans le sens de la hauteur, exactement en-dessous de la ligne de masse.
- Classe 4 : entrée en diffusion. Même chose que pour la classe 3, mais pour une entrée.

Remarque :

Une symétrie horizontale ne change pas la classe d'interface, sauf que la classe du bord droit devient celle du bord gauche et réciproquement. Une symétrie verticale change la classe d'interface sauf si elle est symétrique (les classes d'interface sont orientées).

On voit que deux bords de classe 1 doivent être séparées d'une distance égale à la plus grande garde des règles technologiques. Pour cela on ajoute une brique appelée brique d'espacement que ne contient que le métal correspondant à la structure de bus et on la place entre les deux cellules de classe 1.

Deux bords de la classe 2 peuvent être juxtaposés.

Un bord de la classe 3 est fait pour s'assembler avec un bord de la classe4. C'est de cette façon qu'on assemble un opérateur et un registre.

Les cellules ont été dessinées avant la définition des classes d'interface. L'assemblage traite ici chaque brique comme un cas particulier.

4.11. CONCLUSION

Ce chapitre a présenté la technique utilisée pour générer les parties opératives. A partir d'un ensemble de fonctions préalablement établi, un ensemble de briques a été dessiné. Leur assemblage est spécifié par un langage de haut niveau et le dessin de la partie opérative en découle automatiquement. Le passage est fait en plusieurs pas dans le but d'isoler les différentes activités. Les modules sont assez indépendants, ce qui évite les modifications importantes quand il y a des évolutions du système. L'introduction (ou la modification) d'une brique, par exemple, ne doit intervenir que sur le module de haut niveau, dans des procédures presque indépendantes qui traitent chaque fonction. Un changement du langage graphique ne doit intervenir que sur la phase finale du programme d'assemblage.

Dans l'ensemble des opérations du système, deux points n'ont pas été exposés car ils n'ont pas été complètement étudiés: les caractéristiques électriques globales de la partie opérative et la topologie des lignes d'interface.

- Caractéristiques électriques. Etant donné que le programme de haut niveau traite des différentes cellules, il est possible d'y insérer un algorithme qui calcule les caractéristiques globales de la partie opérative. Par exemple, si on connait la consommation individuelle de chaque cellule, on peut calculer la consommation totale de la partie opérative en additionnant les consommations de toutes les cellules utilisées. Il est possible de faire de même pour la charge du bus et la charge d'une ligne de commande ; cela permettrait de faire une estimation des performances de la partie opérative.

- Topologie des lignes d'interface. Les lignes d'interface sont les conducteurs qui touchent le bord de la partie opérative (dont une partie est utilisée pour communiquer avec la partie contrôle). Le programme d'assemblage calcule la position absolue des cellules et pourrait donc calculer la position de ces lignes.

Conclusion 5

5 - CONCLUSION

L'automatisation de la conception des circuits intégrés est un problème d'actualité. Tout le monde est d'accord sur son urgence. C'est la condition même du développement de la VLSI qui est en jeu. Faire des circuits de plus en plus complexes avec des moeyns rudimentaires n'est pas possible car le coût de conception devient insupportable.

Au début des circuits intégrés, l'effort était placé dans le domaine de la technologie. Avec l'intégration à grande échelle, il est apparu à côté de l'effort technologique, l'effort de conception qui exige de plus en plus d'énergie.

On peut distinguer deux classes dans les circuits de haute intégration :

- Les circuits dont le dessin se caractérise par une répétition massive de certains motifs (pas trop complexes), comme les mémoires. Pour ces circuits, l'effort est toujours mis dans le domaine de la technologie, y compris l'étude des composants, leur miniaturisation et leur réalisation industrielle.

- Les circuits dont l'algorithme exécuté est complexe, comme les microprocesseurs. Dans ce cas, le problème se pose surtout au niveau de la conception de la machine. Les derniers microprocesseurs mis sur le marché ont une complexité de l'ordre des unités centrales des miniordinateurs 16 bits des années 70. Si le travail exigé pour concevoir les microprocesseurs et les miniordinateurs est du même ordre de grandeur, leur prix de vente n'est pas comparable. L'équilibre peut se rétablir car le bas prix des microprocesseurs leur ouvre les portes d'une infinité de nouvelles applications. La quantité vendue associée à un coût de fabrication très faible par rapport aux unités centrales à composants discrets, permet de retrouver un équilibre.

De cette façon, si l'on pense seulement à une industrie qui mettrait sur le marché un nouveau microprocesseur tous les 4 ou 5 ans, on ne trouverait peut-être pas une justification pour tout l'effort mis en oeuvre actuellement pour automatiser la conception des circuits intégrés.

La course vers la complexité des microprocesseurs connaîtra peut-être d'autres revers: ces machines deviennent de plus en plus puissantes et de là apparaît une tendance à les utiliser surtout dans des systèmes plus proches de l'informatique par opposition aux systèmes plus rudimentaires où le microprocesseur constitue surtout le noyau d'une logique programmée (on n'utilisera pas un IAPX 432 pour contrôler une machine à laver).

Si le marché se restreint, il faut donc attendre plus longtemps pour que la quantité produite amortisse le coût de projet. La concurrence importante incite à lancer de nouveaux produits. La solution est donc de concevoir moins cher, c'est-à-dire d'automatiser la conception. Le microprocesseur n'est que la tête d'une famille de circuits ; autour de lui on trouve toute une gamme de produits: les circuits périphériques. Pour qu'un microprocesseur s'impose sur le marché, il faut qu'il soit entouré de périphériques dont la complexité, pour quelques-uns, n'est pas très éloignée de celle du processeur. Tout cela nécessite une équipe de conception performante, d'où le recours à l'automatisation de la conception, la CAO.

Il y a un autre domaine où l'automatisation de la conception est d'une importance capitale: ce sont les circuits à la demande. Ces circuits sont destinés à des applications spécifiques et la quantité produite ne justifierait peut-être pas leur fabrication si le temps de conception nécessaire est trop important. Certes, il peut exister des circuits dont la décision de fabrication n'est pas dépendante du coût de l'opération, mais ce sont des cas très particuliers. Pour les circuits à la demande, les moyens de CAO sont donc une nécessité.

La CAO comprend plusieurs activités dans la conception des circuits intégrés: simulation logique, simulation électrique, dessin des masques, passage automatique d'une description à une autre (de la description logique à la description des masques et vice-versa...) et même l'administration du projet et le test. On se restreint ici à la génération des masques, bien que la méthodologie de base ait des influences sur d'autres domaines. C'est l'aspect méthodologique qui distingue cette approche.

Il y a aujourd'hui plusieurs techniques disponibles industriellement :

- **Les réseaux prédiffusés**

Un réseau de centaines et même de milliers de portes, dont toutes les opérations de masquage, sauf une (la métallisation) sont déjà réalisées, est offert au concepteur qui "personnalise" son circuit en jouant sur les interconnexions. Des moyens importants de CAO sont aussi offerts par les constructeurs de ces réseaux, comme des bibliothèques de cellules, des outils de connexion automatique, de documentation automatique etc... Une fois prêt le masque de métallisation, le temps nécessaire pour réaliser le circuit est assez court, car il ne reste qu'une étape du processus technologique à réaliser (ce qui signifie aussi réduction du coût qui se fera sentir surtout sur les petites quantités). Cette technique n'est cependant pas intéressante pour les VLSI: la densité n'est pas élevée, à cause des interconnexions et de la sous-utilisation des portes.

- **Les bibliothèques de cellules**

La description au niveau des masques de fonctions logiques simples est stockée dans une bibliothèque informatique. La conception consiste à assembler sélectivement ces cellules et à les interconnecter. La densité s'améliore d'un peu car les cellules sont optimisées ; mais les canaux d'interconnexion sont toujours assez importants. Pour la fabrication, un jeu complet de masques est alors nécessaire.

- Le dessin symbolique,

Soit sous la forme de dessin sur grille (comme le DMOS), soit dans la forme de STICKS [MEA 80] (systèmes CABBAGE, TRICKY...), permet au concepteur de définir simplement chaque élément de son circuit. Le dessin des masques est ensuite fait automatiquement d'après la spécification symbolique. Le dessin du circuit n'est pas lié à des cellules prédéfinies et une utilisation plus optimisée de la surface est possible. Le prix payé est la nécessité de définir le circuit au niveau des transistors ou des portes, c'est-à-dire sans diminution entre le nombre d'élements du circuit et celui des éléments dessinés. Naturellement, il est possible de faire des cellules et on se trouvera dans le cas précédent.

- Les cellules fonctionnelles

Le Moyen d'obtenir de meilleurs résultats est de modifier la méthodologie de conception. Si de nouveaux outils sophistiqués sont requis pour les nouvelles générations de microprocesseurs et leurs familles [WEISS 81] il n'est pas moins vrai que des méthodologies appropriées sont nécessaires. Il serait intéressant de voir, à titre d'exemple, le résultat d'un outil qui générerait les masques d'un microprocesseur spécifié en termes d'équations logiques (sans les concepts de bus, registre, mémoire, régularité, etc...).

La méthodologie que l'on a étudiée dans ce travail est basée sur un modèle des machines informatiques, c'est-à-dire qu'on apporte une sémantique, dès le départ, aux éléments qui composent le modèle : on leur donne un sens et une structure prédéfinis et elles sont donc dessinées pour réaliser une fonction spécifique dans un contexte précis. Le dessin lui-même peut être fait à la main, sur papier, comme on l'a indiqué, ou par des moyens automatisés ; l'essentiel c'est que les cellules dessinées pour construire un bloc soient faites pour s'assembler.

Un point de ROM de microprogramme, un point de PLA ou une constante dans la partie opérative seront dessinés avec des contraintes différentes, même si leur réalisation électrique est équivalente. On peut dire la même chose pour une porte logique ou une cellule bistable.

La microélectronique recommence en quelque sorte l'histoire de l'industrie des ordinateurs, bien que dans un contexte différent. Situé à un carrefour du matériel et du logiciel, la microinformatique doit absorber l'expérience acquise avec le développement des systèmes digitaux et des systèmes logiciels. De la même façon qu'on passe d'un programme écrit dans un langage de haut niveau à un code machine, il semble raisonnable de penser qu'un chemin puisse exister entre une description de haut niveau et une machine sur silicium.

Des problèmes existent bien sûr, par exemple : quelle sera la surface de silicium utilisée pour réaliser une telle machine? ou, quelle forme aura-t-elle? ou, encore, sera-t-il possible de spécifier la performance envisagée? et, finalement, quels circuits seront faits avec cette technique?

Tous les circuits ne seront pas faits d'une façon entièrement automatique, comme on n'écrit pas tous les programmes en langage de haut niveau (quand on désire un programme très performant, on l'écrit à un niveau plus près du langage machine). Il y a d'autres dimensions à considérer dans un circuit que son fonctionnement: sa structure; sa forme, sa consommation électrique etc... Quand on parle de conception entièrement automatique, cela n'exclut pas une interaction possible avec le concepteur, comme dans le cas d'une conception assistée par le concepteur. En fait, au moment de décrire sa machine, le concepteur a déjà une idée, ne serait-ce que globale, de l'architecture et de la technique de réalisation de sa machine (fonction câblée ou PLA pour le contrôle du registre d'état, par exemple). La décomposition du travail de conception en étapes comme, par exemple, la mise en évidence de la topologie générale du circuit (plan de masse) suivie de l'étude de chaque bloc, permet au concepteur de suivre le déroulement du processus et de participer aux décisions prises en choisissant dans un ensemble d'options possibbles. Il serait important de pouvoir prédire le comportement temporel, soit par simulation au niveau du composant, soit au niveau des portes, en tenant compte du retard de propagation et de certains phénomènes parasites comme les charges importantes ou les temps de mémorisation des lignes après qu'elles soient déconnectées.

Sur le plan pratique, cependant, une évolution graduelle des outils de conception est préférable à une révolution des méthodes. C'est dans ce sens que ce travail constitue, à notre avis, une contribution.

La solution pour résoudre la complexité temporelle (c'est-à-dire la puissance de calcul nécessaire) émergera de l'intérieur même de ce processus par un phénomène de réalimentation dans le sens que les machines complexes et puissantes produites seront utilisées pour produire d'autres machines encore plus complexes et ainsi de suite.

En d'autres termes, la puissance de calcul pourrait devenir une ressource de plus en plus abondante. Il faut toutefois diriger cette force, c'est-à-dire programmer, ce qui demande de plus en plus de travail.

On ne peut pas achever cette dissertation sans évoquer des aspects humains associés au développement de la microélectronique, comme par exemple :

- Quelles seront les conséquences sociales d'une généralisation de l'informatique à tous les domaines de l'activité humaine?

Annexe

ANNEXE 1

Le circuit LISA

Le circuit LISA est un circuit de contrôle de communications de données pour ligne série. Il se présente comme un périphérique de microprocesseur avec lequel il communique par un bus de 8 bits.

Ce circuit [BER 80] est actuellement en phase de développement (EFCIS).

Les caractéristiques principales sont :
- allocation de la ligne en mode autoallouée (LISA) ou CSMA (DANUBE),
- vitesse maximale: 1 Mbits/s en mode autoalloué et 3 Mbits/s en mode CSMA,
- détection des conflits sur la ligne,
- calcul polymonial du code de détection d'erreurs,
- accusés de réception (simple et multiple),
- transfert de données avec l'UCP en mode programme ou par accés direct à la mémoire,
- possibilité de test.

De manière interne, ce circuit peut être vu comme un système à deux processeurs: un processeur d'émission et un processeur de réception. Chaque processeur comporte deux modules coopérants: un module 1 bit et un module 8 bits. La figure A.1 présente un schéma global du circuit.

Les fonctions principales de chaque bloc du circuit sont :

- Pour la réception :

Module 1 bit : séparation des données et de l'horloge (seulement pour le mode autoalloué), reconnaissance du message (détection des caractères-drapeaux) désérialisation et vérification du CRC, contrôle de la trame des accusés de réception et extraction des bits d'accusé de réception multiple.

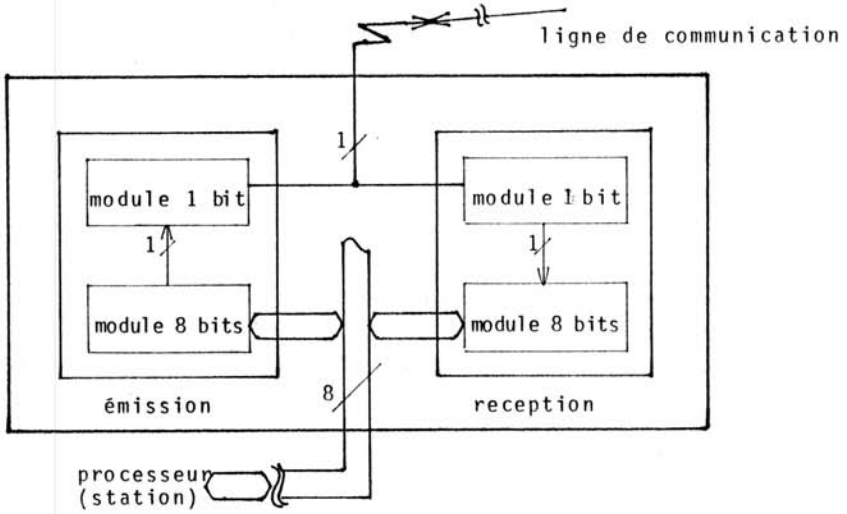


Figure A.1. - Structure du circuit LISA

Module 8 bits : reconnaissance des adresses destinataires, échange d'octets avec la station (CPU), comptage des octets, calcul de l'état final (avec déclanchement des interruptions).

- Pour l'émission :

Module 8 bits : échange d'octets avec la station, décomptage du nombre d'octets, construction du message et calcul de l'état final.

Module 1 bit : sérialisation des données, calcul du CRC, traitement des erreurs (conflit), envoi des accusés de réception.

A titre d'application de ce travail, nous avons réalisé la partie opérative commune des modules 8 bits des deux processeurs en y incluant les registres à décalage pour la sérialisation et désérialisation. La structure de cette partie opérative est montrée dans la figure A.2. Pour cette réalisation, on a utilisé des cellules de la famille double bus. Toutefois, on n'a utilisé que des opérateurs unaires. Les bus sont également utilisés d'une manière particulière :

- l'un des deux bus traverse toute la partie opérative et accède à la majeure partie des éléments de mémorisation: il est appelé bus externe et permet la communication avec l'extérieur.
- l'autre bus est segmenté en deux parties: l'une constitue le bus interne du module émission et l'autre partie, le bus interne de la partie réception.

Remarque :
Les noms des cellules sont ceux présentés dans la figure A.2.

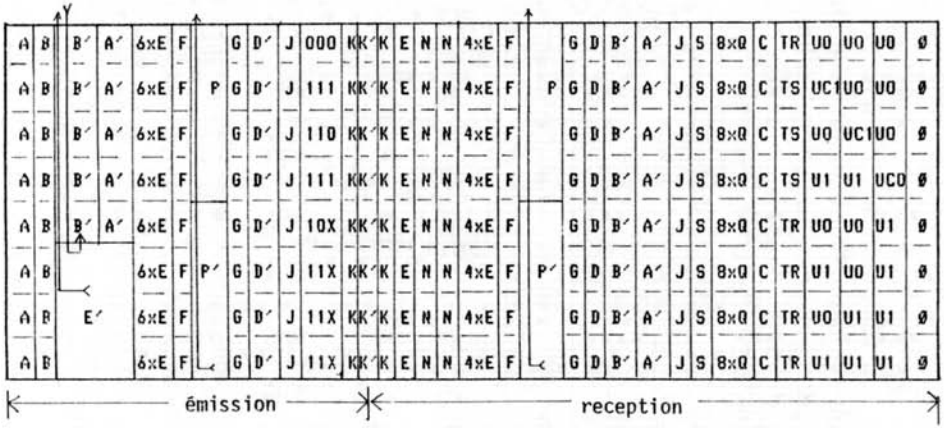


Figure A.2. - Cellules du circuit LISA

Une reproduction du dessin des masques est présentée dans la planche couleur ci-après.

Cellules A et B

Les cellules A et B fonctionnent comme un registre à décalage. Ce sont, en fait, deux cellules registre modifiées, entre lesquelles on a placé un opérateur de décalage. La figure A.3 donne le schéma logique, et la figure A.4 le dessin. (Les dessins sont présentés sans leur bord pour mettre en évidence l'imbrication verticale).

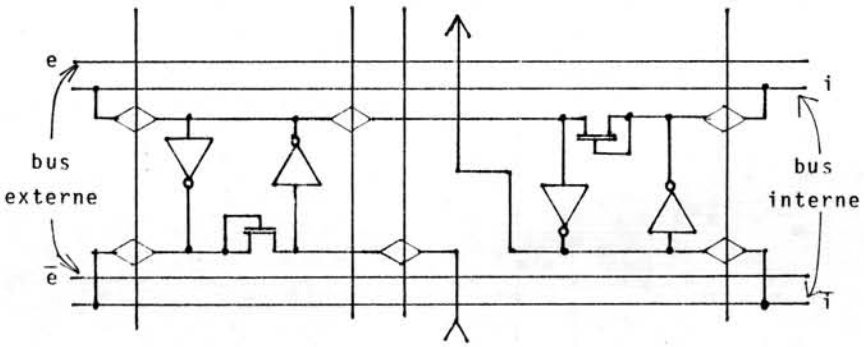


Figure A.3. - Schéma logique des cellules A et B.

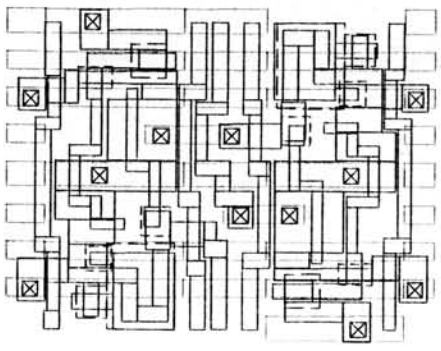


Figure A.4. - Dessin des cellules A et B

Les cellules E et F sont des cellules mémoires à double accés (E) et à simple accés sur le bus interne (F).

Les mêmes cellules ont été étudiées dans le chapître concernant le dessin [4.9] (6 X E signifie 6 cellules E).

La cellule N

La cellule N est une mémoire double accés, à laquelle on ajoute une remise à zéro asynchrone. La figure A.5 présente le schéma logique (a) et le dessin de cette cellule (b).

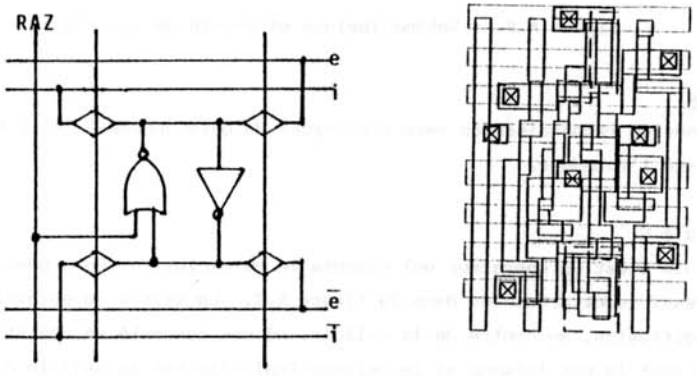


Figure A.5. - Schéma logique et dessin de la cellule N

La cellule Q

Q est une cellule de mémoire associative. Cet élément de mémorisation est chargé par le bus externe et le mot qui doit être reconnu est présenté sur le bus interne. La reconnaissance se fait sur toute la longueur du mot (il n'a pas été nécessaire d'inclure une fonction de masquage). Le schéma logique et le dessin de la cellule Q sont présentés dans la figure A.6.

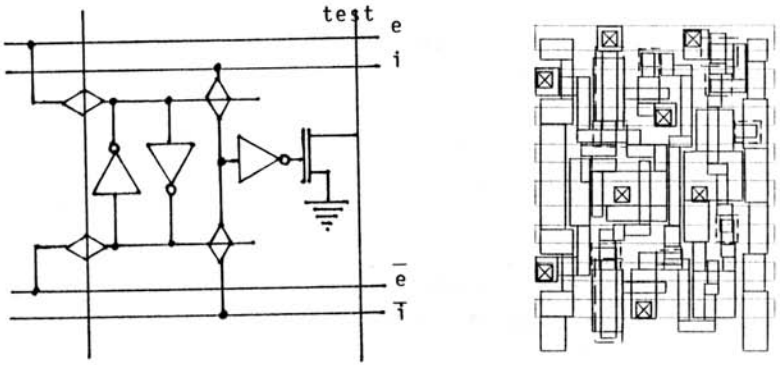


Figure A.6. - Schéma logique et dessin de la cellule Q

Remarque :
Les lignes de reconnaissance seront chargées du coté opposé à celui où sera placé l'encodeur.

La cellule G

La cellule G est un opérateur qui exécute le ou-exclusif. Le schéma logique et le dessin sont présentés dans la figure A.7. La sortie du ou-exclusif est en diffusion, au centre de la cellule, où est connecté un registre. Les entrées sont le bus interne et la retenue (calculée par la cellule P).

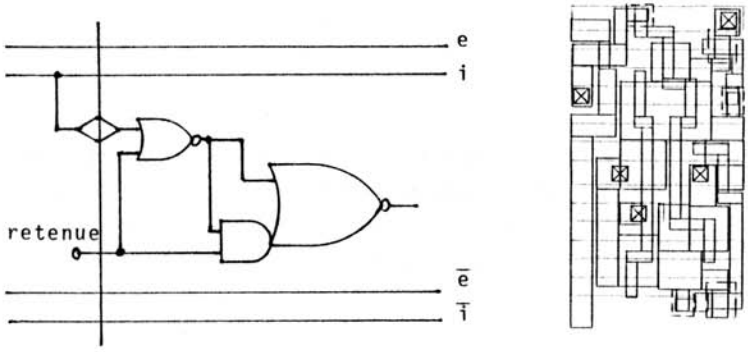


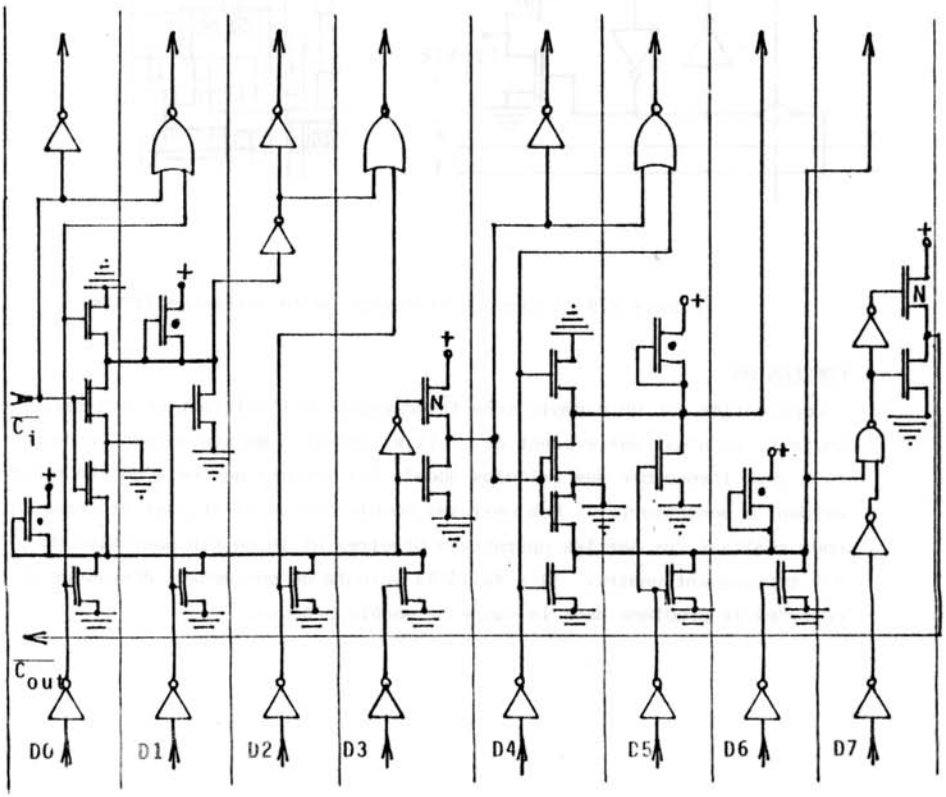
Figure A.7. - Cellule ou-exclusif (G).

Cellules P et P'

P et P' sont des cellules de calcul de la retenue par anticipation par blocs de quatre bits. P est la cellule de base qui peut être empilée pour faire, par exemple, un calcul de 8 ou 12 bits. Pour la dernière tranche de 4 bits, on a modifié la cellule de base pour deux raisons :

- permettre le retour de la retenue vers la partie contrôle,
- accélérer le calcul de la retenue des deux derniers bits car ils doivent traverser le ou-exclusif avant d'attaquer le registre (tampon).

Le résultat du câlcul de l'incrémentation (ou la décrémentation) est approximativement simultané avec la retenue sortante. Le chargement du tampon et de la bascule de retenue peuvent donc être faits par la même commande. La simulation de la propagation montre qu'une bascule connectée à la retenue sortante contient la bonne valeur 35 ns après que les données en entrée soient stabilisées (cas le plus défavorable).
La figure A.8 montre le schéma logique des cellules P et P'.



- Mémoire avec sortie amplifiée (cellule O)

Cette cellule se décompose en une cellule mémoire connectée au bus et un amplificateur constitué de deux transistors qui pilotent une ligne en sortie (figure A.9). Les deux inverseurs du point mémoire sont petits (faible sortance) tandis que les transistors de sortie sont assez grands. Les grilles de ces transistors sont activées directement par le bus au moment de l'écriture. Comme les bus ne montent qu'à 4 volts, l'excursion électrique de la ligne de sortie est aussi de 4 volts (avec un transistor naturel). Le point mémoire monte ensuite à 5 volts, entraînant la ligne de sortie.

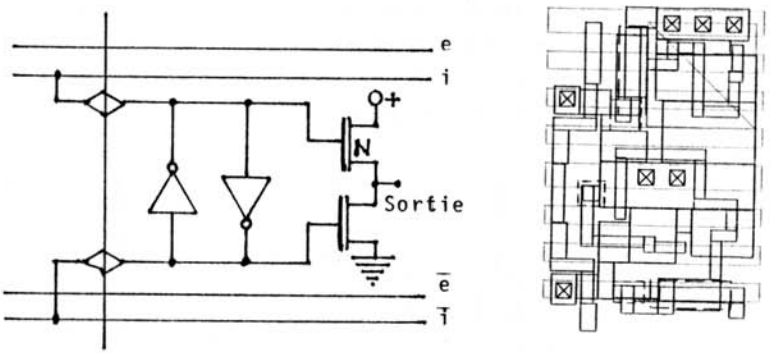


Figure A.9. - Circuit et dessin de la sortie amplifiée.

CONCLUSION

L'application est un exemple très intéressant de l'utilisation des cellules. En fait, il n'est pas évident de réaliser toute la partie opérative du circuit LISA avec l'ensemble des cellules double bus (encore que cela puisse paraître évident a posteriori!). Les cellules double bus en effet, ont été conçues pour réaliser des parties opératives binaires et la partie opérative de LISA est typiquement unaire. Il a fallu la volonté du concepteur d'essayer de résoudre le problème dans le cadre du modèle proposé.

On a donc été obligé de dessiner des cellules spécifiques à ce circuit. Des opérateurs spéciaux ont été étudiés et dessinés. Pour un autre circuit une grande partie du travail peut être récupérée, mais on aura besoin d'autres cellules qui devront être dessinées et ajoutées au système: c'est d'ailleurs ainsi que le système évolue.

ANNEXE 2

Le circuit LISA utilise une unité arithmétique et logique (UAL) incomplète. Le schéma d'une UAL générale a été étudié mais on n'en a pas dessiné les masques. Le schéma est présenté dans la figure A.10 et les fonctions exécutées sont données dans le tableau ci-dessous. On a 5 commandes C1, C2, C3, C4, C5 en entrée, et Ci est la retenue entrante.

C1	C4 ou Ci	C2	C3	C5=0	C5=1		
0	0	0	0	$a \oplus b$	a+b		
0	0	0	1	$a \vee b$	$a \vee b$		
0	0	1	0	$\overline{a \wedge b}$	ø	$\overline{a \wedge b}+1$	
0	0	1	1	1	1		
0	1	0	0	$a \equiv b$	a+b+1		
0	1	0	1	$\overline{a \vee b}$	*	$a \vee b+1$	
0	1	1	0	$a \wedge b$	ø		
0	1	1	1	0	0		
1	0	0	0	$a \equiv b$	**	a-b-1	$a+\overline{b}$
1	0	0	1	$a \vee \overline{b}$	ø	$a \vee \overline{b}$	
1	0	1	0	$\overline{a} \vee b$	ø		
1	0	1	1	1	1		
1	1	0	0	$a \oplus b$	$a+\overline{b}+1$	a-b	
1	1	0	1	$\overline{a} \wedge b$	***	$a \vee \overline{b}+1$	
1	1	1	0	$a \wedge \overline{b}$	ø		
1	1	1	1	0	0		

* : a+1 si b=0; b+1 si a=0

** : b-1 si $a=(FF...)_{16}$

***: $\overline{b-1}$ si a=0

ø : fonction invalide

Fonctions exécutées par l'UAL.

Cette cellule a une hauteur égale à deux pas de la structure de bus proposée et réalise les fonctions de l'UAL pour deux tranches de bit voisines. La recherche qui a été faite sur cette UAL s'est orientée dans deux directions:

- la régularité et la simplicité du schéma logique, tout en ayant un ensemble assez riche de fonctions,
- l'étude du calcul de la retenue afin de pouvoir répondre à différents impératifs sur le temps de propagation de cette retenue.

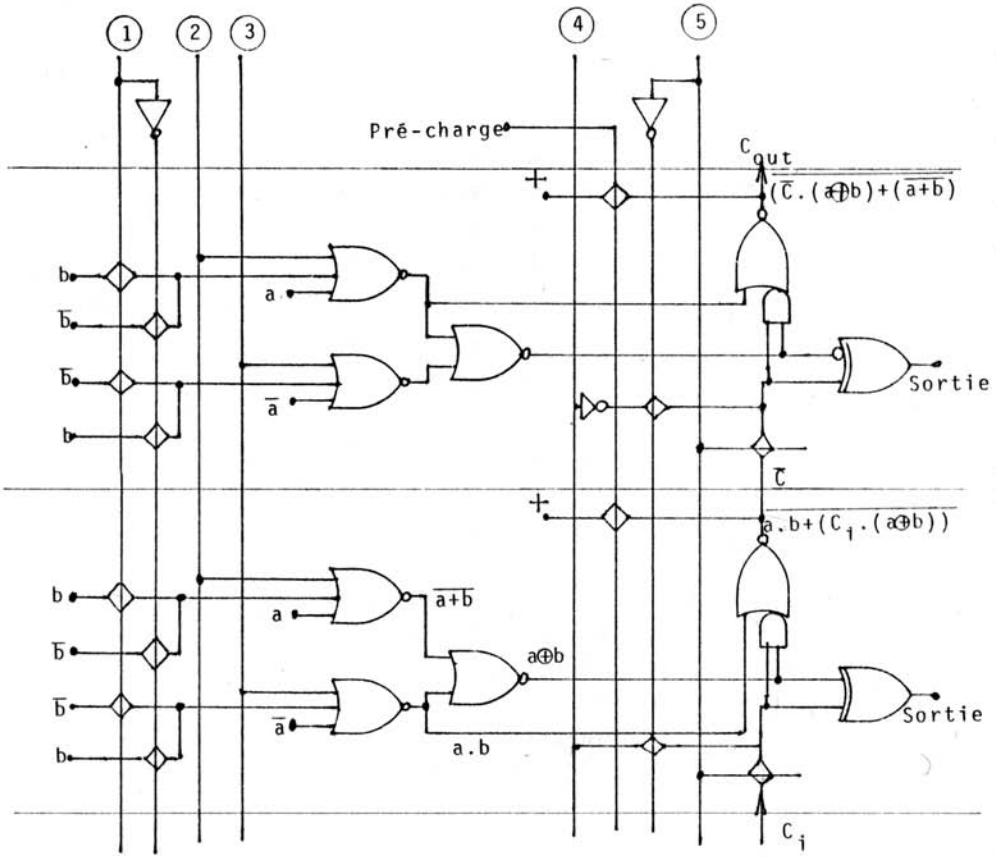


Figure A.10 - Schéma de l'unité arithmétique et logique.

Pour ce dernier point, un circuit de base est proposé, comportant une propagation série (une couche logique par bit). Cela implique une inversion de la retenue à chaque tranche. Le dessin doit être fait en trois briques: entrée, retenue et ou-exclusif, dans le but de pouvoir remplacer facilement cette chaîne de propagation par une autre (par exemple, avec propagation anticipée par blocs de 4 bits).

Remarque :
La commande 4 est utilisée dans le mode logique pour entrer une constante sur tous les bits au niveau du ou-exclusif. Dans le mode arithmétique, cette ligne peut être utilisée pour renvoyer à la partie contrôle la retenue sortante (ou la retenue entrante, selon le côté où se trouve la partie contrôle).

REFERENCES

...

[ANC76] F. Anceau "Architecture des Ordinateurs"
Edite par IRIA. Le Chesnay 1976.

[ANC81] F. Anceau "VLSI Processor Architecture"
ESSCIRC'81, pp 24-30. Freiburg Septembre 1981.

[BAR75] M. R. Barbacci "A Comparison of RTL for Describing Computers and Digital Systems"
IEEE- Trans. Computers vol C-24 n.2, pp 137-149 Fevrier 1975.

[BER80] C. Bernard "Conception Logique du Circuit LISA"
Rapport de DEA, Grenoble Septembre 1980.

[BIR80] G. J. A. Bird "Design of Continuous and Digital Electronics Systems"
McGraw-Hill Book Company Limited (UK) London 1980.

[BEY81] J. W. Beyers et allii "A 32 Bit VLSI CPU Chip"
IEEE Int. Solid State Circuits Conference
Vol XXIV, pp 104-105, Fevrier 1981

[CRA81] H. G. Cragon "Microprocessor Architecture which Reflects Software Requirements"
COMPCON81, pp 36-39, San Francisco. Fevrier 1981

[COU81] B. Courtois "Test et VLSI"
These Docteur d'Etat, USMG et INPG. Grenoble Juin 1981.

[DAR80] S. B. Baran "CAD for VLSI"
17th DAC, pg 642 Minneapolis Juin 1980.

[DUR79] A. Duret "Participation a la Conception et la Realisation en LSI, de la Partie Operative d'une Machine Integree"
These Docteur 3eme Cycle INPG Grenoble Decembre 1979.

[HOF81] B. Hofflinger "MOS Circuits"
Digital Technology Status and Trends pp 75-113
edit. Helmut Fainke, Oldenburg, Munchen 1981.

[HOL81] S. Hollock "Circuits Integres semi-Particulier - VLSI Rendue Facile"
Colloque International sur les Nouvelles Orientations des Circuits Integres, pp 108-111 Paris avril 1981.

[HOS80] R. Hoshikawa et allii "A 10000-Gate CMOS Lsi Processor"
ISSCC80, pp 106-107 San Francisco Fevrier 1980.

[JER80] A. A. Jerraya et M. Khessaira "LUCIE - Langage Universitaire de Conception de C. I. pour l'Enseignement"
Manuel d'Utilisation, ENSIMAG, Grenoble 1980.

[JOH79] D. Johannsen "Bristle Blocks: A Silicon Compiler"
16th DAC pp 310-312 San Diego, Juin 1979.

[MAR81] R. J. Markowitz "Software Impact on Microcomputer Architecture - A Case Study"
COMPCON81, pp 40-48 San Francisco Fevrier 1981.

[MER69] J. Mermet "Le Langage Cassandre"
Structure et Conception des Ordinateurs, pp
edit O.T.A.N. Dunod Paris 1971.

[MEA80] C. Mead et L. Conway "Introduction to VLSI Systems"
Addison-Wesley 1980.

[MIL79] J. Millman "Micro-Electronics Digital and Analog Circuits and Systems"
McGraw-Hill, New-York 1979.

[OBR81] M. Obrebska "Etude Comparative des Differentes Methodologies de Conception des Parties Controle des Microprocesseurs"
Colloque Int. sur les Nouvelles Orientations de C. I.
pp 185-191, Paris Avril 1981.

[PEA80] J. B. Peatmann "Digital Hardware Design"
McGraw-Hill, New-York 1980.

[PER80] T. Perez Segovia "Optimisation des PLAs"
Rapport de Recherche n. 216 ENSIMAG, Grenoble 1980.

[REI81] R. Reis "Evaluateur Topologique pour Circuits VLSI: Module d'Evaluation de ROM"
Rapport de Recherche n. 252, ENSIMAG, Grenoble Juin 1981.

[ROS80] L. M. Rosemberg "The Evolution of Design Automation to Meet the Challanges of VLSI"
17th DAC pp 3-11 Minneapolis Juin 1980.

[SCH77] J. P. Schoellkopf "Machine PASC-HLL: Definition d'une Architecture pipe-line pour une Unite Centrale Adaptee au Langage Pascal"
These Docteur 3eme Cycle INPG Grenoble Juin 1977.

[SUZ79] A. A. Suzim "Parties Operatives a Elements Modulaires"
Rapport de DEA. ENSIMAG. Grenoble 1979.

[SUZ80] A. A. Suzim "Modular Data Processing Units"
ESSCIRD'80 pp 287-289 Grenoble 1980.

[SUZ81] A. A. Suzim "Data Processing Section for Microprocessor-like Integrated Circuits"
IEEE Solid State Circuits Vol SC16 n.3 pp233-235, Juin 1981.

[SUZ81b] A. A. Suzim "Cell Assembling Language"
ESSCIRC'81, pp 87-89 Freiburg Septembre 1981.

[WEI81] D. G. Weiss "Computer Architecture in LSI - Lessons and Prospects
COMPCON81 pp 49-52 San Francisco Fevrier 1981.

"VLSI Design Tool Development at DEC"
LAMBDA Vol II n.1 pp 12-13, First Q. 1981

dernière page de la thèse

AUTORISATION DE SOUTENANCE

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

VU les rapports de présentation de

. Monsieur F. ANCEAU, Professeur
et de
. Monsieur J.L LARDY

Monsieur SUZIM Altamiro Amadeu

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR INGENIEUR, Spécialité "Génie Informatique".

Fait à Grenoble, le 23 octobre 1981

Le Président de l'I.N.P.-G.

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