SILICIEL : Contributions à l'architecture des cicuits intégrés at à la compilation du silicium

Jean-Pierre Schoellkopf

THESE

présentée à

l' Université Scientifique et Médicale de Grenoble
et à
l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR D'ETAT ES-SCIENCES
« Informatique »

par

Jean-Pierre SCHOELLKOPF

SILICIEL

CONTRIBUTIONS A L'ARCHITECTURE DES CIRCUITS INTEGRES ET A LA COMPILATION DU SILICIUM.

Thèse soutenue le 22 avril 1985 devant la commission d'examen

G. VEILLON	Président
F. ANCEAU	Rapporteur
J. BOREL	Examinateurs
J. VUILLEMIN	
R. GUEDJ	
E. HOERBST	
G. NOGUEZ	

UNIVERSITÉ
SCIENTIFIQUE ET MÉDICALE
DE GRENOBLE

Saint-Martin-d'Hères, le

Domaine Universitaire, Bâtiment administratif Sciences
B. P. n° 68
38402 SAINT-MARTIN-D'HERES CEDEX

Président de l'Université :

M. TANCHE

Année Universitaire 1984-1985

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES

Ne figurent pas dans la liste les professeurs de Médecine-Pharmacie

* **PROFESSEURS DE 1ère CLASSE**

ARNAUD Paul	Chimie Organique
ARVIEU Robert	Physique Nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique Approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean Claude	Physique Expérimentale C.N.R.S.
BARJON Robert	Physique Nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques-Maths Appliquées
BELORISKY Elie	Physique C.E.N.G. - D.R.F.
BENZAKEN Claude	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean-Paul	Mathématiques Pures
BILLET Jean	Géographie
BOEHLER Jean-Paul	Mécanique
BONNIER Jane Marie	Chimie Générale
BOUCHEZ Robert	Physique Nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie Organique
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean-Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures

Téléphone (76) 51.46.00
Adresse télégraphique : EXACSCIENCES 38 GRENOBLE

CYROT Michel	Physique du Solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude	M.I.A.G. Maths Appliquées
DEPORTES Charles	Chimie Minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des Plasmas
DOUCE Rolland	Physiologie végétale
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques Pures
GAGNAIRE Didier	Chimie Physique
GASTINEL Noël	Analyse Numérique Maths Appliquées
GERBER Robert	Mathématiques Pures
GERMAIN Jean Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jea-René	Mathématiques Pures
KAHANE André détaché	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques Pures
KRAKOWIAK Sacha	Mathématiques Appliquées
KUPKA Yvon	Mathématiques Pures
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences Nucléaires I.S.N.
MACHE Régis	Physiologie Végétale
MAYNARD Roger	Physique du Solide
MICHEL Robert	Minéralogie et Pétrographie (Géologie)
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie végétale)
PAYAN J. Jacques détaché	Mathématiques Pures
PEBAY PEYROULA J. Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique

PIERRARD Jean Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SAKAROVITCH Michel	Mathématiques appliquées
SENGEL Philippe	Biologie Animale
SERGERAERT Françis	Mathématiques Pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique Nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques Appliquées
VAUQUOIS Bernard	Mathématiques Appliquées
VIALLON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE

ADIBA Michel	Mathématiques Pures
ANTOINE Pierre	Géologie
ARMAND Gilbert	Géographie
AURIAULT Jean-Louis	Mécanique
BARET Paul	Chimie
BEGUIN Claude	Chimie Organique
BLANCHI J. Pierre	STAPS
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BRUANDET J. François	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
CHIARAMELLA Yves	Mathématiques Appliquées
COURT Jean	Chimie
DEMAILLY Jean Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DERRIEN Jacques	Physique
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie

GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaire
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean Paul	Biochimie
KERKOVE Claude	Géologie
LEBRETON Alain	Mathématiques Appliquées
LONGEQUEUE Nicole	Sciences Nucléaires I.S.N.
LUCAS Robert	Physique
LUNA Domingo	Mathématiques Pures
MANDARON Paul	Biologie
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique CNRS - CRTBT
OUDET Bruno détaché	Mathématiques Appliquées
PELMONT Jean	Biochimie
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Jean-Claude détaché	Physique du Solide
PIBOULE Michel	Géologie
PIERRE Jean Louis	Chimie Organique
PIERY Yvette	Biologie
RAYNAUD Hervé	Mathématiques Appliquées
RIEDJMANN Christine	Mathématiques Pures
ROBERT Gilles	Mathématiques Pures
ROBERT Jean Bernard	Chimie Physique
ROSSI André	Physiologie végétale
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie Animale
SERVE Denis	Chimie
SOUTIF Jeanne	Physique
SCHOOL Pierre Claude	Mathématiques Appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VIVIAN Robert	Géographie

****

UNIVERSITÉ
SCIENTIFIQUE ET MÉDICALE
DE GRENOBLE

Saint-Martin-d'Hères, le

Domaine Universitaire, Bâtiment administratif Sciences
B. P. n° 68
38402 SAINT-MARTIN-D'HERES CEDEX

Président de l'Université :

M. TANCHE

Année Universitaire 1984-1985

MEMBRES DU CORPS ENSEIGNANT DE L'IUT 1

* PROFESSEURS DE 1ère CLASSE

BUISSON Roger	Physique IUT 1
DODU Jacques	Mécanique Appliquée IUT 1
NEGRE Robert	Génie Civil IUT 1

* PROFESSEURS DE 2ème CLASSE

ARMAND Yves	Chimie IUT 1
BOUTHINON Michel	EEA. IUT 1
BRUGEL Lucien	Energétique IUT 1
CHECHIKIAN Alain	EEA IUT 1
CHENAVAS Jean	Physique IUT 1
CONTE René	Physique IUT 1
GOSSE Jean Pierre	EEA IUT 1
GROS Yves	Physique IUT 1
KUHN Gérard	Physique IUT 1
MARECHAL Jean	Mécanique IUT 1
MICHOULIER Jean	Physique IUT 1
MONLLOR Christian	EEA IUT 1
NOUGARET Marcel	automatique IUT1
PEFFEN René	Métallurgie IUT 1
PERARD Jacques	EEA IUT 1
PERRAUD Robert	Chimie IUT 1
TERRIEZ Jean Michel	Génie Mécanique IUT 1

****

Téléphone (76) 51.46.00
Adresse télégraphique : EXACSCIENCES 38 GRENOBLE

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

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

Professeurs de 1ère Catégorie

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

Professeurs de 2ème catégorie

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

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

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

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président: Daniel BLOCH

Vice-Présidents: René CARRE
Hervé CHERADAME
Jean-Pierre LONGEQUEUE

Année universitaire 1983-1984

Professeur des Universités

ANCEAU	François	E.N.S.I.M.A.G
BARIBAUD	Michel	E.N.S.E.R.G
BARRAUD	Alain	E.N.S.I.E.G
BAUDELET	Bernard	E.N.S.I.E.G
BESSON	Jean	E.N.S.E.E.G
BLIMAN	Samuel	E.N.S.E.R.G
BLOCH	Daniel	E.N.S.I.E.G
BOIS	Philippe	E.N.S.H.G
BONNETAIN	Lucien	E.N.S.E.E.G
BONNIER	Etienne	E.N.S.E.E.G
BOUVARD	Maurice	E.N.S.H.G
BRISSONNEAU	Pierre	E.N.S.I.E.G
BUYLE BODIN	Maurice	E.N.S.E.R.G
CAVAIGNAC	Jean-François	E.N.S.I.E.G
CHARTIER	Germain	E.N.S.I.E.G
CHENEVIER	Pierre	E.N.S.E.R.G
CHERADAME	Hervé	U.E.R.M.C.P.P
CHERUY	Arlette	E.N.S.I.E.G
CHIAVERINA	Jean	U.E.R.M.C.P.P
COHEN	Joseph	E.N.S.E.R.G
COUMES	André	E.N.S.E.R.G
DURAND	Francis	E.N.S.E.E.G
DURAND	Jean-louis	E.N.S.I.E.G
FELICI	Noël	E.N.S.I.E.G
FONLUPT	Jean	E.N.S.I.M.A.G
FOULARD	Claude	E.N.S.I.E.G
GANDINI	Alessandro	U.E.R.M.C.P.P
GAUBERT	Claude	E.N.S.I.E.G
GENTIL	Pierre	E.N.S.E.R.G
GUERIN	Bernard	E.N.S.E.R.G
GUYOT	Pierre	E.N.S.E.E.G
IVANES	Marcel	E.N.S.I.E.G
JALINIER	Jean-Michel	E.N.S.I.E.G
JAUSSAUD	Pierre	E.N.S.I.E.G
JOUBERT	Jean-Claude	E.N.S.I.E.G
JOURDAIN	Geneviève	E.N.S.I.E.G
LACOUME	Jean-Louis	E.N.S.I.E.G
LATOMBE	Jean-Claude	E.N.S.I.M.A.G
LESIEUR	Marcel	E.N.S.H.G
LESPINARD	Georges	E.N.S.H.G
LONGEQUEUE	Jean-Pierre	E.N.S.I.E.G
LOUCHET	François	E.N.S.E.E.G
MASSELOT	Christian	E.N.S.I.E.G
MAZARE	Guy	E.N.S.I.M.A.G
MOREAU	René	E.N.S.H.G
MORET	Roger	E.N.S.I.E.G
MOSSIERE	Jacques	E.N.S.I.M.A.G
PARIAUD	Jean-Charles	E.N.S.E.E.G
PAUTHENET	René	E.N.S.I.E.G
PERRET	René	E.N.S.I.E.G
PERRET	Robert	E.N.S.I.E.G
PIAU	Jean-Michel	E.N.S.H.G
POLOUJADOFF	Michel	E.N.S.I.E.G
POUPOT	Christian	E.N.S.E.R.G
RAMEAU	Jean-Jacques	E.N.S.E.E.G
RENAUD	Maurice	U.E.R.M.C.P.P
ROBERT	André	U.E.R.M.C.P.P
ROBERT	François	E.N.S.I.M.A.G
SABONNADIERE	Jean-Claude	E.N.S.I.E.G
SAUCIER	Gabrielle	E.N.S.I.M.A.G
SCHLENKER	Claire	E.N.S.I.E.G
SCHLENKER	Michel	E.N.S.I.E.G
SERMET	Pierre	E.N.S.E.R.G
SILVY	Jacques	U.E.R.M.C.P.P
SOHM	Jean-Claude	E.N.S.E.E.G
SOUQUET	Jean-Loùis	E.N.S.E.E.G
VEILLON	Gérard	E.N.S.I.M.A.G
ZADWORNY	François	E.N.S.E.R.G

Professeurs Associés

BLACKWELDER	Ronald	E.N.S.H.G
HAYASHI	Hirashi	E.N.S.I.E.G
PURDY	Gary	E.N.S.E.E.G

Professeurs Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis	
CHATELIN	Françoise	

Chercheurs du C.N.R.S

FRUCHART	Robert	Directeur de recherche
JORRAND	Philippe	Directeur de recherche
VACHAUD	Georges	Directeur de recherche
ALLIBERT	Michel	Maître de recherche
ANSARA	Ibrahim	Maître de recherche
ARMAND	Michel	Maître de recherche
BINDER	Gilbert	Maître de recherche
BORNARD	Guy	Maître de recherche
CARRE	René	Maître de recherche
DAVID	René	Maître de recherche
DEPORTES	Jacques	Maître de recherche
DRIOLE	Jean	Maître de recherche
GIGNOUX	Damien	Maître de recherche
GIVORD	Dominique	Maître de recherche
GUELIN	Pierre	Maître de recherche
HOPFINGER	Emil	Maître de recherche
JOUD	Jean-Charles	Maître de recherche
KAMARINOS	Georges	Maître de recherche
KLEITZ	Michel	Maître de recherche
LANDAU	Ioan-Dore	Maître de recherche
LASJAUNIAS	Jean-Claude	Maître de recherche
MERMET	Jean	Maître de recherche
MUNIER	Jacques	Maître de recherche
PIAU	Monique	Maître de recherche
PORTESEIL	Jean-Louis	Maître de recherche
THOLENCE	Jean-Louis	Maître de recherche
VERDILLON	André	Maître de recherche
SUERY	Michel	Maître de recherche

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

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

ALLIBERT	Colette	DIARD	Jean Paul	NGUYEN TRUONG	Bernadette
BERNARD	Claude	EUSTATHOPOULOS	Nicolas	RAVAINE	Denis
BONNET	Roland	FOSTER	Panayotis	SAINFORT	(CENG)
CAILLET	Marcel	GALERIE	Alain	SARRAZIN	Pierre
CHATILLON	Catherine	HAMMOU	Abdelkader	SIMON	Jean Paul
CHATILLON	Christian	MALMEJAC	Yves (CENG)	TOUZAIN	Philippe
COULON	Michel	MARTIN GARIN	Régina	URBAIN	Georges (Laboratoire des ultra-réfractaires ODEILLO).

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

BARIBAUD	Michel	CHEHIKIAN	Alain	HERAULT	Jeanny
BOREL	Joseph	DOLMAZON	Jean Marc	MONLLOR	Christian
CHOVET	Alain				

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

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

E.N.S.H.G.

ALEMANY	Antoine	MICHEL	Jean Marie	ROWE	Alain
BOIS	Daniel	OBLED	Charles	VAUCLIN	Michel
DARVE	Félix			WACK	Bernard

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

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

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

CHARUEL	Robert

C.E.N.G.

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

Laboratoires extérieurs :

C.N.E.T.

DEMOULIN	Eric	GERBER	Roland	MERCKEL	Gérard
DEVINE	R.A.B.			PAULEAU	Yves

I.N.S.A. Lyon

GAUBERT	C.

****

SILICIEL mon mari !

(variation sur une manchette de Libération de Mai 1984)

à Christine

à mes parents

à mes amis

- RESUME -

Cette thèse présente des contributions dans les domaines de l'architecture des ordinateurs réalisés sous la forme d'un Circuit Intégré. Un assembleur de silicium, appelé LUBRICK, permet de décrire, dans un langage de programmation, la constitution d'un assemblage hiérarchisé de cellules pour réaliser la description complète des masques d'un Circuit Intégré.

La compilation du silicium, discipline qui consiste à déduire les masques d'un circuit en partant d'une description fonctionnelle, est ici abordée sous un angle pratique, avec la présentation d'un compilateur prototype d'une forme de partie contrôle et des présentations de modèles topologiques de parties opératives et de parties contrôle qui servent de cible au compilateur.

- MOTS-CLES -

Architecture des Ordinateurs, Circuits Intégrés, VLSI, Technologie MOS, Chemin de données, Parties Opératives, Parties Contrôle, Séquenceur, Machine microprogrammée, Assembleur de Silicium, Compilation du Silicium, logique dynamique, Conception Assistée par Ordinateur (CAO).

Remerciements :

Gérard VEILLON, directeur de l'ENSIMAG, me fait le grand honneur de présider le jury; je l'en remercie.

François ANCEAU, ex-professeur à l'ENSIMAG/INPG, chef de projet à BULL-SYSTEMES, guide depuis 1972 mes activités "universitaires" et nous entreprenons actuellement ensemble un ambitieux projet "industriel". Je ne me sens pas pour autant dispensé de le remercier pour ses encouragements à terminer la rédaction de cette thèse, et pour les précieux conseils et les directives qu'il a pu me progiguer ou m'imposer, tout en me témoignant sa confiance et en m'offrant la liberté d'entreprendre nécessaire à mon épanouissement.

Jean VUILLEMIN, ingénieur de recherches à l'INRIA, a bien voulu juger le manuscript de cette thèse; je le remercie pour ses critiques constructives qui m'ont aidé à améliorer ma présentation pour en faciliter la lecture (bien qu'il soit toujours très dur de remettre une nouvelle fois l'ouvrage sur le métier).

Joseph BOREL, directeur du Département Technologie à THOMSOM-EFCIS, est ici remercié d'avoir consacré une part de son précieux temps pour lire mon manuscript, rédiger un rapport, et participer au jury, apportant ainsi une forme de caution à une recherche qui affirme des prétentions de "qualité industrielle". Je n'oublierai pas Jean-Pierre MOREAU, ingénieur à THOMSON-EFCIS, pour ses précieuses appréciations.

Richard GUEDJ, ex-ingénieur de recherche au LCR (THOMSON) et créateur de SIGRID, qui a suivi ce travail et m'a encouragé à approfondir mes réflexions.

Gérard NOGUEZ, professeur à l'INSTITUT de PROGRAMMATION (PARIS 6), me fait une grande joie de participer à mon jury: je dois avouer que le cours que j'ai subi en 1972-73, dispensé par lui-même, sur les fondements des structures matérielles, m'a fortement influencé dans le choix de mon domaine d'activité, en contribuant à développer en moi le "virus" de la conception des ordinateurs.

Finally Doctor Egon HOERBST, from SIEMENS, who is present in the jury in spite of the language problem: during several years, we had very interesting discussions in the field of VLSI design with him and people in his research team.

Cette thèse est le fruit d'un travail personnel de recherches et de rédaction: elle n'aurait cependant pas pu voir le jour sans le concours, sous diverses formes, de chacun des membres de l'Equipe de Recherches en Architecture d'Ordinateurs de l'IMAG/TIM3. La liste serait trop longue, il faudrait commencer par Robert FORTIER et Jean-Pierre MARTIN en 1972-73, citer Gérard BAILLE et Jacques LAURENT entre 1974 et 1979, ne pas oublier les anciens comme Alain GUYOT et Gilles BERGER-SABBATEL, ni les moins anciens comme Souheil MARINE, Ahmed JERRAYA, et tous les autres qui par leurs questions ou leurs suggestions ont contribué à l'avancement des travaux de recherche de cette Equipe et, indirectement, à la rédaction de cette thèse: qu'ils soient tous assurés de ma gratitude et de mon amitié.

Le travail le plus ingrat, celui de la "frappe", est généralement cité en avant-dernière position, avant le service de reprographie, et je ne suis pas assez insolent pour rompre avec la tradition, car Madame Hélène DIAZ, qui a effectué la "saisie" d'une grosse partie du texte sur microordinateur, a trop d'humilité pour accepter les remerciements qu'elle mérite: elle a su s'adapter à un système de traitement de textes qui, malgré la qualité des résultats fournis, est resté rudimentaire et difficile à conduire, de l'aveu de son auteur lui-même.

Enfin, le service de reprographie de l'IMAG, sans lequel cette thèse resterait secrète, est vivement remercié pour son travail efficace et sérieux. Que son chef, Daniel IGLESIAS, veuille bien m'excuser pour les nombreux rendez-vous manqués.

AVERTISSEMENTS ET PRESENTATION DU CONTEXTE

Cette thèse présente des contributions personnelles de l'auteur, sur quelques-uns des nombreux sujets appartenant au domaine de la conception des ordinateurs intégrés. Ce domaine, très vaste, n'est abordé ici que ponctuellement par quelques-uns de ces aspects, avec cependant deux thèmes centraux: l'approche topologique et la compilation.

Seule l'introduction tente de présenter le domaine dans son ensemble, en décrivant une méthodologie de conception à l'élaboration de laquelle l'auteur a contribué, au sein de l'Equipe de Recherches en Architecture d'Ordinateurs de l'IMAG. Cette méthodologie est le fruit de dix années d'expériences menées dans une équipe, chacun des membres de cette équipe ayant contribué à l'enrichissement de l'ensemble; cette méthodologie s'est affinée au cours des années et elle a du s'adapter à l'évolution de la technologie, mais elle a gardé une constance dans ses objectifs: permettre la conception d'ordinateurs et de circuits intégrés de qualité. Une conclusion importante peut être tirée de cette expérience: l'objectif de qualité ne peut pas être atteint par la seule approche théorique; il est impératif de faire appel à une expérience concrète, dans un domaine essentiellement technique, ce qui va à l'encontre d'une théorie unificatrice et requiert des expérimentations pratiques sur de nombreux exemples. On trouvera cependant dans cette thèse des essais de formalisation des problèmes par une approche théorique destinée à clarifier certains problèmes difficiles.

L'ouvrage est organisé en trois parties indépendantes, qui tentent chacune de couvrir un sujet, non pas d'une manière théorique et exhaustive, mais plutot en proposant une approche expérimentale personnelle concrétisée par la présentation d'une réalisation "prototype". On trouvera ainsi, après une présentation de chacun des problèmes abordés, une solution pratique qui permet sinon de les résoudre, du moins de les traiter et de progresser dans leur connaissance. On touche ici à des sujets pour lesquels il existe un grand nombre de solutions, car ce sont des sujets techniques, et il serait indécent d'affirmer que la solution proposée est autre chose qu'une des solutions possibles. On a cependant recherché une cohérence entre les solutions proposées pour résoudre des problèmes fortement liés par une constante: les contraintes

topologiques très fortes qui régissent l'architecture interne des ordinateurs intégrés.

Cette thèse touche principalement à deux disciplines: l'architecture des ordinateurs et les outils informatiques d'aide à la conception des circuits intégrés. La première discipline n'est abordée que par l'architecture INTERNE des circuits intégrés, la seconde n'étant sollicitée que pour résoudre les besoins de la première, dans le cadre d'une activité appelée "compilation du silicium".

La compilation du silicium est abordée ici d'une manière pratique, sous la forme d'un compilateur prototype qui, partant de la description de la fonction que doit réaliser un ordinateur, produit la description des masques d'un circuit intégré qui réalise cette fonction. La présentation de ce compilateur prototype tente de faire apparaitre les problèmes à résoudre, propose des solutions, l'objectif de l'auteur étant d'apporter une contribution pratique dans ce vaste domaine technique, avec l'objectif de qualité cité plus haut qui s'avère etre un handicap par rapport à d'autres chercheurs qui n'en ont pas fait leur objectif prioritaire (certains allant même jusqu'à affirmer que le dessin des masques d'un circuit est un travail sale, alors que les tâches nobles appartiennent à la recherche théorique).

Cet objectif de qualité est présent dans chacune des parties de cette thèse:

- l'assembleur de silicium LUBRICK a pour ambition de permettre la génération du dessin de masques de circuits les plus denses possible, sans erreur, et dans des délais courts;
- l'implantation des parties opératives selon une technique de tranches est ici présentée comme illustration de l'utilisation d'un assembleur de silicium dans le cas d'un modèle qui a fait preuve de sa qualité;
- l'approche topologique proposée pour les parties contrôle doit permettre de maitriser l'implantation de ces parties, de réduire leur surface, et tient compte de l'impératif besoin de confier leur dessin à un compilateur de silicium qui saura seul maitriser leur complexité et fournir une implantation dense et correcte.

AIDE AU LECTEUR

Les sujets abordés dans cette thèse sont variés, et la présentation est organisée en trois parties indépendantes:

- la première partie présente un Assembleur de Silicium,
- la seconde partie discute un modèle des Parties Opératives,
- la troisième partie propose un modèle des Parties Contrôle.

La lecture de la seconde partie sera facilitée si le lecteur s'est déjà familiarisé avec la première partie, car on y aborde des aspects topologiques de construction de certains blocs entrant dans la réalisation des Parties Opératives, les exemples étant bien sûr décrits en utilisant l'Assembleur de Silicium "LUBRICK". Cette deuxième partie est découpée en deux chapitres, le premier présentant des généralités et proposant un modèle général, le second étant consacré à la conception architecturale de la Partie Opérative d'un processeur particulier: le processeur P-CODE.

La troisième partie peut être abordée indépendamment des deux premières: elle traite divers aspects relatifs aux Parties Contrôle, et elle est organisée en cinq chapitres.

Le premier chapitre expose une approche TOPOLOGIQUE GLOBALE (au niveau du plan de masse d'un ordinateur intégré), tandis que le second chapitre propose des solutions pour compiler une description fonctionnelle et en déduire la réalisation intégrée (approche par la COMPILATION).

Les chapitres suivants présentent des applications et des exemples: une proposition de topologie pour les structures microprogrammées (chapitre 4), une organisation topologique pour la structure à générateur d'instants et son COMPILATEUR prototype "MACSIM" (chapitre 3); enfin le chapitre 5 aborde des problèmes de circuiterie dynamique multiphase, en proposant un modèle général, pour la technologie N-MOS seulement, le but de l'auteur étant de montrer qu'il faut non seulement trouver des modèles architecturaux, mais aussi des modèles électriques avant d'envisager la compilation du silicium, si l'on veut avoir quelques chances de produire un circuit de qualité par compilation.

AVANT-PROPOS

ET

INTRODUCTION

SOMMAIRE:

AVANT-PROPOS

L'ARchiTECture d'ordinateur est un mélange d'ART et de TEChniques, ce qui peut laisser planner un doute sur la valeur scientifique de cette discipline...

Ce travail présente des approches de solutions pour les tâches complexes de conception d'un ordinateur intégré, et suggère des techniques de réalisation adaptées aux technologies nouvelles: les technologies des Circuits Intégrés (ou CI), et plus spécialement pour les technologies MOS (Métal-Oxyde-Semiconducteur).

Les techniques proposées s'apparentent à une discipline nouvelle appelée "compilation du silicium", qui regroupe les activités multiples allant de la conception architecturale jusqu'au dessin des masques des Circuits Intégrés.

L'architecte d'ordinateur se trouve embarqué dans une aventure technologique dont personne ne peut encore prédire les limites: aujourd'hui (en 1984) 500.000 transistors sur une puce d'environ UN centimètre-carré, demain (en 1986) un Million de transistors ... ?

Les données du problème de conception changent avec l'avènement d'une nouvelle technologie, et le très haut degré d'intégration (VLSI = Very Large Scale Integration) qui est proposé par les technologues est en fait une révolution, aussi bien pour la manière de penser les problèmes que pour la mise en oeuvre des outils qui permettront de trouver des solutions:

- quelles architectures nouvelles peut-on intégrer ?
- comment gérer les millions d'informations qui servent à leur description?

L'architecture d'ordinateur est très dépendante du rapport qui

existe entre la complexité et le prix des composants disponibles. Historiquement, on définit des périodes (ères) marquées chacune par l'existence d'une famille technologique dont les caractéristiques ont permis de construire des familles d'ordinateurs. Les évolutions les plus sensibles sont dues à un progrès technologique significatif: passage du tube au transistor, puis de quelques transistors sur une "puce" à quelques centaines, et actuellement à quelques centaines de milliers...
Chaque révolution technologique donne naissance à une nouvelle classe d'ordinateurs: les mini-ordinateurs sont apparus avec le "catalogue" TTL, puis les microprocesseurs ont vu le jour, donnant le départ au formidable développement de la technologie (pour les technologies MOS tout spécialement) qui à son tour a donné naissance à la microinformatique puis à l'ordinateur familial ...

Les approches proposées dans cette thèse sont basées sur deux expériences:
- celle des industriels qui ont mis sur le marché des ordinateurs intégrés et ont ainsi "publié" l'état de leur art,
- et celle d'une équipe de recherches qui a conçu et réalisé dans les dix années passées de nombreuses architectures d'ordinateurs, avec des technologies diverses (circuits discrets, macrocomposants puis circuits intégrés).

Le fruit de ces expériences accumulées est une philosophie de conception basée sur:
- la construction ASCENDANTE d'outils et de méthodes d'implantation adaptés aux besions réels des concepteurs et aux contraintes draconiennes de minimisation de la surface globale d'un circuit,
- la définition progressive d'une méthode de conception DESCENDANTE qui ne sera complète que lorsque tous les outils seront opérationnels, et qui permettra alors un gain très important sur le temps et la sécurité de la conception, donc sur le coût d'intégration d'un nouveau circuit intégré.

INTRODUCTION

La "compilation du Silicium" est une discipline récente. Son but est la production, par des outils informatiques (programmes), d'une description des masques d'un Circuit Intégré (CI).

Cette définition simple ne la distingue pas de la Conception Assistée par Ordinateur (CAO), discipline plus générale qui l'englobe.
On établira une distinction entre:
- les outils informatiques qui servent à la compilation du Silicium,
- la méthodologie de définition et d'utilisation de ces outils, qui est basée sur une solide expérience de conception, et qui est sous-jacente dans cette présentation, sans toutefois en être le thème.

Les outils informatiques présentés ici pourraient être appelés de type "SILICIEL", parce que leur but est la production de circuits sur SILIcium à l'aide d'outils logiCIELs.

La méthodologie de définition de ces outils, dont l'ensemble constituera à terme un "compilateur de Silicium", repose sur l'expérience d'une équipe de recherches, dans différents domaines et avec différentes technologies:
- une expérience en "architecture d'ordinateur", ou comment organiser des composants pour bâtir un ordinateur qui satisfasse aux caractéristiques demandées;
- une expérience en "architecture interne des Circuits Intégrés", ou comment placer des éléments sur une pastille de Silicium en minimisant sa surface et le temps nécessaire à la réalisation du dessin de ces éléments.

"Compiler du Silicium" consiste à analyser une description

fonctionnelle abstraite d'un CI, pour en produire une description concrète: celle des masques qui serviront à sa fabrication.

- La description fonctionnelle d'un CI est un programme, écrit dans un langage de description fonctionnelle, qui décrit d'une manière non ambigue ce que doit faire le CI, ce qu'on appelle aussi son "comportement" dans un environnement donné (c'est donc un élément abstrait, de type logiciel);
- La description des masques d'un CI est un ensemble de rectangles, de couleurs différentes; elle peut être considérée comme un élément concret dans la mesure où elle représente le plan de fabrication du circuit intégré réel.

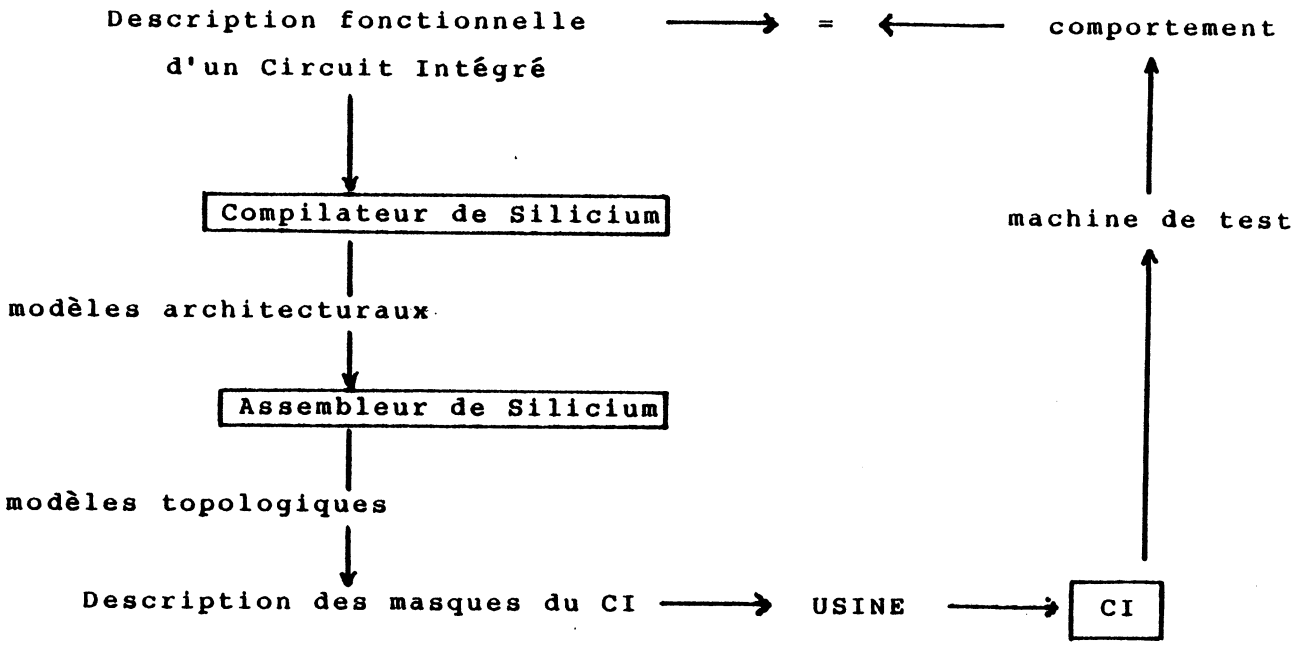
La méthodologie proposée, pqur la compilation du silicium, consiste à s'appuyer sur une étape intermédiaire, appelée "Assemblage de Silicium", pour laquelle nous avons défini un "Assembleur de Silicium" (voir chapitre LUBRICK), afin de distinguer:

- les problèmes de compilation, qui restent du domaine des techniques informatiques,
- les problèmes d'organisation fonctionnelle, qui sont du domaine de l'architecture d'ordinateur,
- les problèmes de topologie des Circuits Intégrés, qui se résument, en ce qui nous concerne, à l'obtention d'une surface minimale pour un circuit de performance maximale dans une technologie donnée.

Le lien entre ces niveaux est assuré par la définition préliminaire de modèles, ou structures d'accueil, qui peuvent être:

- visés comme cible par un compilateur,
- transformés en un dessin de CI par l'assembleur de Silicium.

Cette thèse présente ainsi: des structures d'accueil, un assembleur de Silicium qui fabrique des dessins, des approches de définition de modules d'un compilateur.


Description fonctionnelle
d'un Circuit Intégré
=
comportement
Compilateur de Silicium
machine de test
modèles architecturaux
Assembleur de Silicium
modèles topologiques
Description des masques du CI
USINE
CI

M E T H O D O L O G I E

A- GENERALITES

La conception d'un Circuit Intégré réalisant une fonction de type "unité centrale" d'ordinateur ou microprocesseur, recouvre les domaines de l'architecture fonctionnelle et de l'architecture matérielle.

A-1- L'architecture fonctionnelle réalise un lien entre l'utilisation informatique qui sera faite de l'ordinateur, et l'organisation des éléments fonctionnels qui constituent cet ordinateur.
La conception d'une telle architecture peut se situer à différents niveaux:

- le niveau de l'architecture d'un "système" construit autour de l'unité centrale: l'organisation de la mémoire primaire ou secondaire, l'existence de processeurs périphériques, les possibilités de parallélisee dans le traitement (multi-processeurs banalisés ou spécialisés), existence d'une mémoire virtuelle (paginée ou segmentée), etc...
- le niveau de l'architecture de l'unité centrale en tant que ressource informatique: intervient ici la nature et la qualité du jeu d'instructions, appelé aussi "langage-machine", caractérisé par sa richesse en registres, la puissance de ses opérations de traitement, ses possibilités d'adressage de la mémoire, etc...

La conception d'une architecture fonctionnelle est ainsi très liée au monde de l'informatique, et les performances informatiques de la machine en dépendront. On peut cependant préciser le rôle du langage de la machine, qui se situe à la frontière entre l'architecture fonctionnelle et l'architecture matérielle.

A-2- Le langage-machine d'un ordinateur établit le lien entre les utilisateurs informatiques et le concepteur de l'architecture

interne de la machine. Il est vu de l'extérieur par les programmeurs et par les compilateurs de langages de haut-niveau. Sa puissance informatique peut être mesurée, en terme d'encombrement statique des programmes, et les performances de l'architecture matérielle vont définir un temps d'exécution des instructions, donc des programmes.

A-3- Définition d'une architecture matérielle.

pour un langage-machine donné, on peut concevoir un grand nombre d'architectures matérielles différentes, ayant des performances et des coûts différents.

L'art de l'architecte consiste à concevoir une architecture matérielle qui satisfasse aux critères de performances imposés, avec un coût minimal de conception et de réalisation technologique. Dans le cadre des technologies des CI, la conception d'un ordinateur intégré représente un investissement d'autant plus important que l'architecture matérielle est complexe, investissement qui ne peut être envisagé que:

- si un marché suffisant existe,
- et si la conception du circuit a des chances d'aboutir (la mise au point d'un prototype d'un circuit de plusieurs centaines de milliers de transistors doit pouvoir être garantie avec un minimum de reprises).

La dernière contrainte est nouvelle, et draconienne par rapport aux technologies discrètes pour lesquelles les prototypes sont câblés, donc aisément modifiables.

La maîtrise de la complexité croissante passe ainsi par l'utilisation de méthodes et d'outils adaptés dont les buts sont:

- la réduction de la durée de la conception (de l'architecture au dessin des masques);
- la réduction des causes d'erreurs fonctionnelles, logiques, électriques ou topologiques;
- l'obtention d'un résultat de qualité, satisfaisant aux performances visées et tenant sur une surface de silicium minimale.

A-4- Le compilateur de Silicium:

Le temps de conception d'un circuit nouveau peut être réduit par l'utilisation d'outils informatiques appropriés, appelés "compilateurs de Silicium", qui doivent:
- produire des circuits corrects par construction,
- générer la description des masques de circuits de plus en plus complexes en un temps très réduit (10 millions de transistors en un mois, avant 1990, d'après [REF. WERNER]).

Wallich définit un "compilateur de Silicium" comme étant:

> "un PROGRAMME qui peut produire la description physique d'un CI (ses masques de fabrication) à partir de spécifications de haut-niveau" [REF. WALLICH].

Gross [REF. GROSS] fait observer à juste titre que cette définition est trop générale pour être précise, et que cette imprécision a donné naissance à deux écoles de pensée:
- la première école regroupe les gens qui croient que TOUS les problèmes de conception de circuits complexes (VLSI) pourront être résolus par l'emploi des techniques informatiques, éventuellement appuyées par des théories mathématiques ou des techniques nouvelles (intelligence artificielle);
- la seconde école est constituée par ceux qui restent sceptiques quant aux performances des compilateurs de silicium et continuent de prôner que les seules conceptions efficaces (en termes de surface) sont les conceptions manuelles.

La vérité doit se situer au milieu, car le domaine de l'architecture des Circuits Intégrés est très complexe, par son étendue: il recouvre toutes les disciplines se situant entre la technologie et l'informatique, en passant par les systèmes graphiques, les simulateurs électriques, logiques, fonctionnels, et tout le monde complexe de l'architecture d'ordinateur.

De nombreux efforts de développement sont en cours, sous forme de grands projets français ou européens (SYCOMORE, CVT, CERES, ESPRIT,...) dont le but est de bâtir des systèmes complets dans lesquels toutes les disciplines seront intégrées, sous la forme d'outils informatiques travaillant, à différents niveaux, sur une base de données commune. Cette approche représente un gros travail

de développement d'outils logiciels, et elle n'a de sens que si elle s'appuie sur une méthodologie de conception des CI pour laquelle les outils seront appropriés.

B- PRESENTATION DES METHODES DE CONCEPTION

La complexité de la conception d'un CI implique l'utilisation de nombreux niveaux de description, correspondant à différents niveaux d'abstraction. Cette décomposition se justifie par des besoins d'expression, des outils de modèlisation et de simulation plus ou moins proches de la réalité matérielle, et des besoins de simplification d'une complexité qu'on cherche à maîtriser.

B-1- LES NIVEAUX DE MODELISATION ET DE SIMULATION EXISTANTS

On distingue, classiquement, plusieurs niveaux qui sont:

B-1-a- Le niveau fonctionnel ou comportemental

Ce niveau permet la description d'un modèle fonctionnel d'un CI, intégré dans un système qui est lui aussi modèlisé à un haut niveau. Il existe des langages spécialisés, comme QNAP [Ref. QNAP] qui permettent de modèliser un système en termes de files d'attente et de "processeurs" qui réalisent un traitement non précisé, en un temps calculé en fonction du "service" demandé par le "client" servi. (On trouvera, par exemple, dans [Ref. ECL] une modèlisation en QNAP du système multi-processeur pipe-line qu'est PASC-HLL, décrit dans [Ref. SCH77], et les résultats de la simulation à ce niveau). D'autres langages spécialisés, comme SIMULA, permettent des descriptions plus précises du traitement réalisé par des modules fonctionnant en parallèle. Enfin, les langages algorithmiques de haut-niveau (PASCAL, FORTRAN, PL1, ...) sont souvent utilisés pour décrire un modèle logiciel de la fonctionnalité d'un circuit, afin d'en faire une simulation qui permet de valider des algorithmes généraux et d'obtenir des informations générales sur les performances du circuit modèlisé.

B-1-b- **Le niveau Transfert de Registres**

Il existe des langages de description spécialisés dans la modélisation d'un CI, de type unité centrale, par un ensemble de "registres" et d'opérateurs arithmétiques ou logiques. On décrit à ce niveau des transferts de données binaires entre des éléments de mémorisation et des opérateurs. Ce type de langage permet de décrire un modèle de CI, d'une manière assez proche d'une réalisation matérielle, sans toutefois détailler la réalisation précise des éléments de mémorisation ou de calcul: on décrit par exemple qu'un opérateur évalue l'addition de 2 nombres, sans préciser la manière dont cet opérateur est matériellement réalisé.

B-1-c- **Le niveau Logique**

Ce niveau peut être confondu, dans certains cas, avec le niveau précédent: il permet une description plus précise, en termes de portes logiques (ET, OU, NON-ET, NON-OU ...) de la réalisation des opérateurs, et il permet parfois l'introduction de retards, afin de modèliser les performances temporelles d'un réseau de portes logiques. Un tel niveau de modèlisation est suffisant pour les technologies discrètes (portes TTL ou cellules pré-caractérisées) pour lesquelles on connaît précisément la fonction et le retard de chaque cellule, et sachant que la réalisation complète du circuit n'est qu'un assemblage de cellules disjointes, de performances connues.

B-1-d- **Le niveau Interrupteur**

Les technologies MOS permettent d'utiliser les transistors comme des interrupteurs, et non plus seulement comme des composants internes d'une porte logique. Des montages complexes, à base d'interrupteurs, peuvent ainsi être conçus, qui réalisent des fonctions classiques ou nouvelles, sans faire apparaître de vraies portes logiques. Ce niveau, indispensable pour la modèlisation des circuits MOS, suppose l'introduction d'un "treillis" de valeurs, afin de modèliser le fonctionnement d'un réseau d'interrupteurs, qui se situe entre le niveau logique (0 et 1) et l'analogique (potentiels et courants).

B-1-e- Le niveau Electrique

Un réseau de transistors dimensionnés, de capacités et de résistances constitue le modèle électrique le plus proche de la réalité du CI; la description de ce réseau peut être "extraite" du dessin des masques du CI, par un programme appelé "extracteur de schéma électrique" [Ref. JERRAYA83, Ref. INRIA82, ...]. Ce programme calcule les dimensions des canaux des transistors, les valeurs des capacités des fils de connexion entre transistors, et fabrique une description du réseau électrique analysée par un simulateur électrique. Ce simulateur [Ref. MSINC, Ref. SPICE] calcule, en fonction des variations des entrées du réseau, les valeurs analogiques en tous les noeuds. L'utilisation de ce niveau de description et de simulation est limitée par le temps de calcul nécessaire pour l'extraction et par celui de la simulation: la simulation d'un réseau de plusieurs centaines de milliers de transistors dure une semaine sur un très gros ordinateur [Ref. HP9000], et on limite généralement la simulation électrique à quelques dizaines de transistors.

B-2- LES METHODOLOGIES DE CONCEPTION

La hiérarchie des niveaux de description présentée en I correspond à une hiérarchie de simulateurs utilisés au cours de la conception d'un CI. Plus on s'éloigne du niveau électrique, plus la complexité des simulateurs diminue, à cause de la réduction de la complexité (modèlisation), mais les temps de simulation ne décroissent pas dans la même proportion, car on simule beaucoup plus de cas: on peut même aller, au niveau fonctionnel, jusqu'à simuler l'exécution d'un programme ou même d'un système d'exploitation.
D'autre part, la distance entre le modèle simulé et le circuit réel ne fait qu'augmenter, et il est nécessaire d'assurer une cohérence entre les différents niveaux, sous peine de simuler une machine différente de la machine réelle.

B-2-a- **L'approche ascendante**

Partant du fait que seule la description des masques est un modèle exact du CI qui sera fabriqué, (ce qui suppose une bonne connaissance de la technologie de fabrication, et qui est généralement admis) on extrait de cette description volumineuse le schéma électrique complet du circuit: cette opération est coûteuse en temps de traitement si elle est faite sur le circuit complet; elle peut heureusement être faite par morceaux, profitant de la répétitivité de certains blocs pour n'en analyser qu'une partie. Idéalement, on pourrait alors injecter le schéma électrique extrait en entrée d'un simulateur électrique et faire exécuter une simulation complète du circuit; à l'heure actuelle, cette simulation n'est pas possible, parce que trop complexe, et parce qu'une séquence de simulation nécessite un nombre de pas d'autant plus grand que les phénomènes simulés sont proches de la réalité physique.

On cherche donc à simplifier la description fournie par l'extracteur, et on dispose alors de 2 solutions:

- la 1ère solution consiste à extraire de la description du schéma électrique une description plus macroscopique qui peut être soit du niveau "interrupteur", soit du niveau "logique". Cette nouvelle description est moins précise, mais elle est par contre plus légère à simuler par un simulateur du niveau correspondant, et les résultats de la simulation peuvent être comparés avec ceux qui sont donnés par la simulation du modèle qui a servi à la conception.
- la seconde solution consiste à extraire directement du dessin des masques une description de niveau "interrupteur" ou "logique", en partant du principe que la description électrique du circuit complet est inexploitable.

Cette approche ascendante peut être suivie pour la vérification d'une partie du circuit, (une ou plusieurs petites cellules, ou un bloc fonctionnel), dès que l'assemblage du dessin de cette partie est terminé: on remplace alors une description électrique fine, en bijection avec le dessin des masques, par une description équivalente plus grossière, mais on est certain que la modèlisation

n'a pas introduit d'erreurs. On peut ainsi remonter dans les niveaux de complexité, mais, s'il existe des outils informatiques pour remonter du masque au schéma électrique, puis au schéma du réseau d'interrupteurs, il n'en existe pas qui permettent de remonter automatiquement au niveau fonctionnel (certains sont en cours de développement [Ref. SYCOMORE]). La vérification de l'équivallence ne peut donc être faite que par comparaison des résultats donnés par des simulations, et le concepteur doit décider que la description extraite a un comportement correct.

B-2-b- **L'approche descendante**

Dans les niveaux supérieurs de complexité, c'est une approche descendante qui est suivie, pour se rapprocher, par étapes successives de raffinement, des niveaux inférieurs (la description du matériel); cette approche est celle des "compilateurs de silicium" idéaux qui, lorsqu'ils existeront, produiront la description des masques d'un ordinateur qui réalise à coup sûr la fonction demandée. Dans la pratique, une équipe de conception utilise successivement différents niveaux de langage (fonctionnel, transfert de registres, logique) au fur et à mesure que les choix architecturaux et structurels sont précisés. Il faut alors franchir le pas de l'intégration et dessiner les masques d'un circuit qui réalise la fonctionnalité décrite à un niveau logique, c'est-à-dire créer une structure matérielle qui permettra:

- de réaliser les opérations de transferts de données, de calculs et de stockage,
- de controler l'exécution des opérations précédentes afin de réaliser globalement les algorithmes décrits au niveau fonctionnel.

B-2-c- **L'approche mixte**

Dans la pratique, c'est toujours une approche mixte qui est suivie: approche descendante pour la fonctionnalité vue de l'extérieur, approche ascendante pour le dessin des masques. Le problème de cohérence se pose à la rencontre des deux approches: comment peut-on être certain que le circuit intégré décrit par ses masques réalise l'ensemble des fonctions décrites par une description fonctionnelle, sachant qu'il est impossible de faire une simulation

complète du circuit au niveau électrique?
On voit alors l'intérêt de l'extension proposée pour l'assembleur de silicium LUBRICK, qui consiste à déduire des opérations de construction du masque (opérations topologiques) les schémas topologiques, électriques, logiques et fonctionnels: on applique le principe que toute opération topologique (placements et connexions) induit la création de fonctions logiques.

Exemple: soit deux cellules topologiques A et B; lorsqu'on place B à droite de A, une connexion est fabriquée. Cette connexion physique supporte une connexion logique, par exemple entre la sortie d'une porte logique réalisée par la cellule A, et l'entrée d'une autre porte logique réalisée par la cellule B. A un autre niveau, lorsqu'on connecte deux grosses cellules, on peut considérer qu'on établit des liens fonctionnels et que le bloc résultant a une nouvelle fonctionnalité, et différents niveaux de description.

Cette approche de construction hiérarchique, guidée par la topologie, semble être prometteuse, et elle peut être mise en oeuvre par simple extension de la sémantique des opérateurs topologiques présentés dans le chapitre LUBRICK.

C- LA METHODOLOGIE PROPOSEE

Les grandes lignes de la méthodologie proposée, qui est sous-jacente dans les chapitres de cette thèse, ont été présentées dans [REF. ANC82] et [REF. ANC83]. Le projet de nom CAPRI (Conception Architecturale de PRocesseurs Intégrés) recouvre une grande partie des activités de recherche et de développement d'une équipe, en ce qui concerne la méthodologie et les outils.

On distingue, dans le méthodologie de CAPRI, trois axes principaux:

C-1- Le langage IRENE:

La définition d'un langage de haut-niveau, appelé IRENE, et présenté dans [REF. IRENE]. Ce langage permet de spécifier à la fois des descriptions fonctionnelles (de niveau classiquement appelé Transfert de Registres) et des descriptions de structures matérielles. Il a été conçu pour répondre à des besoins précis de spécification de la dualité suivante: on doit à la fois décrire une fonction à réaliser (description fonctionnelle ou comportementale) et une structure matérielle qui doit réaliser cette fonction (description structurelle). L'utilisation d'un langage unique assure ainsi une cohérence entre deux descriptions, l'une abstraite et l'autre concrète.

Le langage IRENE doit ainsi permettre de décrire, au départ de la conception, les spécifications fonctionnelles du CI à réaliser, et d'en simuler le comportement, à ce niveau, par l'utilisation d'un simulateur fonctionnel.
La description fonctionnelle ainsi validée est ensuite transformée, soit manuellement, soit automatiquement par un compilateur, en une description d'une structure qui a le même comportement lorsqu'on l'excite. Il faut pour cela définir une cible pour le compilateur: cette cible est une architecture matérielle prédéfinie.

C-2- La définition de structures d'accueil prédéfinies:

On entend par "structure d'accueil" un modèle architectural général

qui peut être particularisé pour une application donnée, tout comme une mémoire programmable électriquement peut accueillir une table de constantes.
La définition de tels modèles architecturaux est faite en partant d'organisations fonctionnelles connues quasi-indépendantes de la technologie: c'est le fruit d'une synthèse des connaissances des architectures de tous les ordinateurs existants. Il faut également tenir compte des contraintes propres à la technologie des CI, de telle sorte qu'on puisse associer à une organisation fonctionnelle connue une organisation matérielle performante: il faut que le modèle soit "intégrable", c'est-à-dire facile à dessiner, et implantable dans une surface minimale. On arrive ici au troisième axe de la méthodologie qui concerne l'approche topologique.

C-3- L'approche topologique:

L'importance des problèmes topologiques, trop souvent sous-estimés, ne fait que croître avec l'augmentation de la complexité des CI; bien que le taux d'intégration augmente d'une manière rapide, il n'est pas permis de gaspiller un millimètre-carré de silicium qui peut accueillir plusieurs milliers de transistors, si l'approche topologique est bonne. La proportion de la surface occupée par les transistors (surface active) sur la surface totale doit être maximisée, en minimisant la surface occupée seulement par des fils d'interconnexion. Une telle optimisation peut être faite au niveau de blocs fonctionnels, pour lesquels des solutions topologiques denses existent (voir le modèle pour les Parties Opératives), ainsi qu'au niveau global du plan de masse du circuit, si l'on est capable de maîtriser la forme des blocs fonctionnels et la position des fils qui les connectent aux autres blocs.
Ces contraintes ont conduit à:
- une recherche de solutions topologiques globales préalables à la définition de structures d'accueil;
- la définition d'un outil d'assemblage, l'assembleur de silicium LUBRICK, qui permet de programmer la construction de blocs denses, de formes variables, et de contenus variables remplis par un compilateur de silicium.

Le chapitre "conception d'un microprocesseur P-CODE" donnera un exemple de mise en oeuvre de la méthodologie, jusqu'au dessin des masques.

De nombreux développements sont en cours, pour la compilation du langage IRENE (projet européen CVT), pour la génération des parties contrôle et des parties opératives (projet national SYCOMORE et projet microprocesseur P-CODE), et dans le cadre de conception de processeurs intégrés spécialisés (un microordinateur C-MOS 8 bits, un microprocesseur 8 bits autotestable, un microprocesseur C-MOS 32 bits). Ces développements permettront, nous l'espérons, de prouver la validité de la méthodologie proposée.

PREMIERE PARTIE

P R E S E N T A T I O N

D U

S Y S T E M E

"L U B R I C K"

=

UN ASSEMBLEUR DE SILICIUM

Ce système a été présenté au congrès VLSI 83,
qui a eu lieu à Trondheim (Norvège) en Août 1983.

Le titre de la publication est:
"LUBRICK: a Silicon Assembler and its application
to data-path design for FISC"

SOMMAIRE:

I- I N T R O D U C T I O N

La construction du dessin d'un Circuit Intégré complexe (VLSI) composé de plusieurs centaines de milliers de transistors requiert:
- une bonne méthodologie de travail,
- une bonne approche topologique,
- et des outils informatiques appropriés,
afin d'obtenir un bon dessin dans des temps acceptables.

Disposant du système graphique LUCIE, conçu et développé dans l'équipe de recherches en Architecture d'ordinateurs de l'IMAG [REF. JERRAYA-GUYOT], qui constitue l'outil de dessin de bas niveau, et mettant à profit:
- une expérience du dessin de circuits réalisés "à la main",
- et les enseignements d'un outil prototype [REF. SUZIM],
nous avons cherché à définir un système simple et efficace, qui réponde aux besoins des concepteurs de circuits intégrés complexes, en partant de la constatation suivante:

> "un Circuit Intégré complexe est une hiérarchie de figures, dans laquelle les feuilles de l'arborescence sont de petites figures, appelées aussi cellules (ou Leaf-cells en anglais), qui peuvent généralement être dessinées à la main, et conçues pour s'assembler avec leurs voisines".

Partant de cette constatation, nous avons mis en oeuvre le principe suivant:

> "décrivons, sous la forme d'un programme écrit dans un langage de haut niveau à structure de bloc, la hiérarchie des figures dont l'assemblage donne le dessin du circuit."

Cette idée simple (de F.ANCEAU), qui s'apparente au principe du système LUCIFER de l'INRIA qui s'appuie sur un environnement LISP, a été approfondie et a donné naissance à LUBRICK, un système d'assemblage de BRIQUES, bâti au-dessus du système LUCIE, d'où son nom.

I-A- DEFINITIONS PRELIMINAIRES

I-A-1/ Choix de PASCAL:

Le langage algorithmique PASCAL à été retenu pour des raisons de disponibilité (sur microordinateurs et mégaordinateurs) et pour la bonne pratique que nous en avions; le langage LISP présente l'avantage de l'enrichissement dynamique de son environnement, et il serait effectivement mieux adapté à cette approche qui consiste à:

- décrire dans un langage de haut-niveau un programme de construction d'une figure,
- compiler ce programme,
- demander son exécution qui fait appel à des fonctions prédéfinies.

Le langage LISP ne fut pas choisi parce qu'il n'était pas "disponible" et donc mal connu; on peut cependant annoncer une implémentation de LUBRICK, ou de sa version améliorée et intégrée à un environnement plus complet, en Le_Lisp de l'INRIA [REF. CHAILLOUX] comme un objectif à court terme.

I-A-2/ Principe de correspondance:

Le premier principe de LUBRICK est la correspondance entre une figure et une fonction écrite en PASCAL (function);
l'exécution de cette fonction:

- retourne un "pointeur" sur une structure de donnée qui décrit l'aspect "sociologique" de la figure construite,
- et elle fabrique la description du dessin de cette figure, dans un fichier utilisable par le système graphique de dessin des masques associé, en générant des ordres de placement d'éléments graphiques (rectangles ou figures).

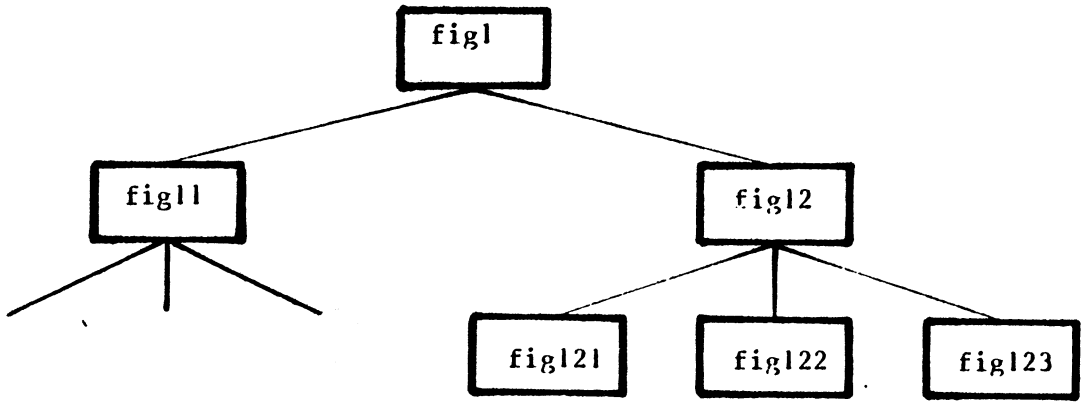
Prenons un exemple:

```
function ARO : pointeur;
  var i : pointeur;
  begin
          i := OPENFIG('ARO');   (* ouverture *)
             x
             x                   (* construction *)
             x
          ARO := CLOSEFIG(i)     (* fermeture *)
  end;
```

L'appel de la fonction ARO fabrique la description de la figure ARO pour le sytème graphique et retourne un pointeur sur sa description interne qui pourra être utilisée pour placer cette figure ARO dans une figure plus englobante.

La correspondance 1 pour 1 précédente peut être étendue à 2 arborescences:

- celle des figures d'un circuit complexe,
- et celle d'un programme PASCAL dans lequel l'imbrication des fonctions reflète la hiérarchie des figures, comme cela existe dans LUCIE et dans d'autres langages de description structurés.



Donnons un exemple de construction hiérarchisée:

LUCIE

```
fig fig1
 fig fig11
     x
     x construit fig11
     x
 ffig
 fig fig12
  fig fig121
       x
       x construit fig121
       x
  ffig
    x
    x construit fig12
    x
 ffig
 x
 x construit fig1
 x
ffig
```

PASCAL

```
function fig1 : pointeur;
     function fig11 : pointeur;
       begin
         x    construit fig11
       end;

     function fig12 : pointeur;
       function fig121 : pointeur;
         begin
           x   construit fig121
         end;

       begin
         x    construit fig12
       end;

begin
  x    construit fig1
end;
```

I-A-3/ TROIS NIVEAUX DE COMPLEXITE DANS LE DESSIN DES CI

L'expérience de construction d'un circuit complexe montre qu'il existe trois niveaux de complexité:

- le niveau des **"cellules"** elles-mêmes;

- le niveau de **"blocs fonctionnels"** pour lesquels des structures denses peuvent être trouvées, et qui peuvent être construits à

l'aide de simples opérateurs de juxtaposition et de répétition de "cellules" de base;

- le niveau d'un grand circuit pour lequel de gros blocs fonctionnels doivent être interconnectés par de nombreux fils, et sans qu'il existe une régularité triviale. Ce dernier niveau demande à la fois la mise en oeuvre de techniques de placement et routage, et une approche topologique globale qui doit pouvoir remettre en cause:
  i/ la forme des blocs,
  ii/ la position de leurs connecteurs,
  ce qui implique un outil puissant de gestion de plan de masse et un assembleur de blocs efficace pour essayer plusieurs formes à un coût raisonnable: les travaux relatifs à ce sujet sont exposés dans [REF. ANCEAU, REIS et REF. FLOPE].

I-A-4/ DEUX CLASSES DE FONCTIONS DANS LUBRICK

Les niveaux de complexité précédents font apparaître deux classes de fonctions à définir:

- une classe de fonctions de **placement** et de **tassement automatiques** de cellules les unes à côté des autres, (en respectant évidemment les règles de dessin) pour construire des blocs denses;

- une classe de fonctions de **connexion automatique** soit pour des cellules fortement liées à l'intérieur d'un même bloc, soit pour des cellules topologiquement indépendantes à relier par de nombreux fils.

I-A-5/ BESOINS ALGORITHMIQUES

Le concepteur a d'autre part besoin d'un langage algorithmique pour
- "calculer" des cellules,
- ou choisir parmi plusieurs formes possibles,

- ou choisir parmi plusieurs fonctions possibles,
- ou évaluer des paramètres,
- ou construire une cellule à base de rectangles ou de figures simples en suivant des algorithmes de construction ou des formules mathématiques.

Il est donc important de pouvoir disposer, dans un langage de programmation **unique**, à la fois:
- de fonctions d'assemblage d'un dessin,
- et de moyens de calculs.

I-B- **La solution retenue est:**

PASCAL TOUT ENTIER à la disposition du concepteur,

PLUS des fonctions prédéfinies appelables depuis n'importe quel programme d'assemblage de cellules.

Un tel principe peut évidemment être appliqué à tous les langages de programmation classiques (Fortran, PL1, langage C, ADA, LISP, ...).

Cette approche est liée à une méthodologie de conception des dessins des circuits intégrés qui préconise la conception programmée au détriment de la conception graphique, si ce n'est pour les petites cellules de base.

II- DEFINITIONS DE BASE

La mise en oeuvre du cahier des charges précédemment décrit nécessite d'abord un enrichissement de la notion de cellule (ou figure, ou BRIQUE), afin de faire réaliser par un programme écrit en PASCAL et faisant appel aux fonctions et procédures du système LUBRICK, le placement relatif des cellules les unes par rapport aux autres, et la génération des fils qui supportent les connexions

entre cellules.

Il faut pour cela décrire l'aspect SOCIOLOGIQUE d'une cellule, c'est-à-dire comment elle se place dans l'environnement des autres cellules qui vont constituer le circuit. Cet aspect SOCIOLOGIQUE est défini par:

II-A- **La forme précise d'une cellule**:

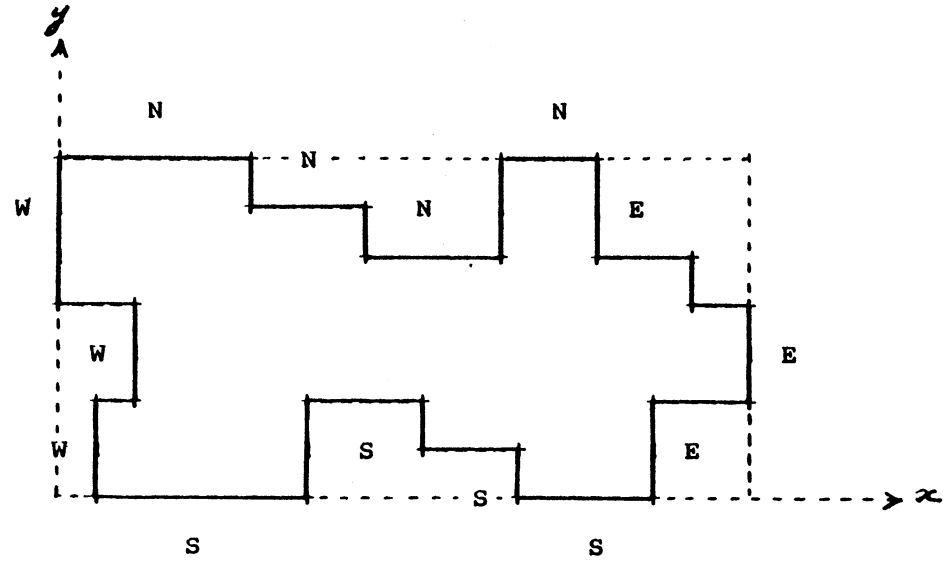
c'est toujours un polygone dont les côtés, appelés segments, sont colorés. Ce polygone définit les frontières extérieures de la cellule.

II-A-1/ Définition des segments:

On décompose la frontière polygonale de la cellule en 4 listes de segments, afin de faire apparaître séparément chacune des 4 frontières, correspondant aux 4 orientations NORD, EST, SUD et OUEST.

En effet, pour réaliser le placement relatif de 2 cellules, on n'a besoin que d'une seule frontière par cellule, et on compare 2 frontières opposées: les frontières NORD et SUD, ou les frontières EST et OUEST. Ce choix conduit à décrire uniquement des segments qui sont parallèles à leur axe de référence:

- des segments horizontaux pour les frontières NORD et SUD,
- des segments verticaux pour les frontières EST et OUEST.



II-A-2/ Définition des connecteurs d'une cellule:

Une cellule ne réalise pas forcément une fonction logique à elle-seule, mais elle possède toujours des points, ou plutôt des segments, qui sont destinés à réaliser la connexion avec des fils venant de l'extérieur, par juxtaposition de deux rectangles ayant un segment commun.

Un segment de connexion, ou CONNECTEUR, est toujours orienté dans l'une des 4 directions (NORD, EST, SUD ou OUEST), et nous avons pris l'hypothèse (restrictive) suivante:

> "il existe un chemin rectiligne, perpendiculaire à l'axe frontière, pour acheminer un fil de connexion depuis la frontière de la cellule jusqu'au segment de connexion";

nous renonçons en effet à chercher à calculer s'il existe un chemin non rectiligne entre un segment connecteur et l'extérieur de la cellule, car la contrainte imposée ne pénalise pas le concepteur des cellules, ni en termes de complexité, ni en termes de surface, et nous proposons des opérateurs d'assemblage qui conservent cette propriété.

On décrira ainsi 4 listes de connecteurs orientés, chaque connecteur étant décrit par le segment terminal qui connectera un fil venant de l'extérieur.

Par rapport à la définition topologique du segment, nous ajoutons à chaque connecteur une LISTE d'ATTRIBUTS, qui dans l'implémentation actuelle, ne contient qu'un seul ATTRIBUT appelé un TYPE: le concepteur dispose de la possibilité de TYPER les connecteurs, en étant libre de choisir le codage (par un nombre entier) et le nombre de types différents.
On peut ainsi facilement distinguer les connecteurs de type ENTREE, ou SORTIE, ou HORLOGE, ou ALIMENTATION, etc. La valeur du type sera conservée pendant l'assemblage, et on pourra soit vérifier la cohérence des types lors d'une connexion automatique, soit utiliser

les types d'une cellule résultat comme paramètres de certains opérateurs d'assemblage (voir CUP plus loin), pour des connexions complexes entre 2 blocs.

D'autres attributs peuvent être ajoutés:
par exemple:
- un NOM LOGIQUE, qui permettrait de construire la description logique de la figure construite, par application de règles d'assemblage qui dérivent la réalisation des connexions logiques de celle de connexions physiques (développement en cours...), et qui permettrait également d'établir une relation entre un fil LOGIQUE et une équipotentielle PHYSIQUE pour les besoins du test du circuit;
- des CARACTERISTIQUES ELECTRIQUES qui permettraient de cumuler des valeurs électriques (capacités ou résistances ou courants) au cours de l'assemblage d'un bloc, à partir des valeurs associées aux connecteurs, pour construire le schéma électrique résultant.

II-B- DESCRIPTION D'UNE CELLULE

La description d'une cellule se présente sous une **forme textuelle**, facile à modifier, facile à transporter, et qui est la seule forme lisible et imprimable évidemment.
Cette description peut être:
- entrée à la main "sous" un éditeur de texte,
- ou générée à partir de la digitalisation d'un dessin,
- ou enfin générée par un programme LUBRICK comme description de la cellule résultant de l'assemblage demandé.

La syntaxe de la forme textuelle est donnée en annexe: "manuel d'utilisation du système LUBRICK".

Cette description appelle quelques remarques:

- certaines frontières peuvent ne pas être décrites explicitement:

LUBRICK considère alors que ces frontières sont implicitement décrites par l'axe correspondant, avec un NIVEAU prédéfini, qui n'a pas d'existence physique, appelé le niveau CONTOUR: il n'y a pas de garde entre deux segments de niveau CONTOUR, ce qui permet de réaliser une juxtaposition de deux cellules ("abutment" en anglais).

- une ‹description› commence par un ‹mot-clé› suivi d'un ‹nombre›; ce ‹nombre› doit être égal au nombre de segments qui suivent. C'est une contrainte lorsqu'une description est faite manuellement, mais c'est une facilité pour observer un résultat: on sait immédiatement, par simple lecture, qu'on a fabriqué une cellule qui a 120 connecteurs NORD, 16 connecteurs OUEST, et 32 connecteurs EST, par exemple, sans avoir à les compter.

- les segments décrivant une frontière ou une liste de connecteurs peuvent être entrés dans un ordre quelconque; par contre, toute description résultant de l'exécution d'un programme LUBRICK présente les connecteurs dans un ordre croissant de leurs POSITIONs (de gauche à droite, ou de bas en haut), ce qui facilite les contrôles et les repérages.

- Cette description pourrait être mélangée avec celle de la figure elle-même, c'est-à-dire avec la forme LUCIE textuelle, mais cette dernière est inutile pour l'assemblage: elle ne sert que pour le dessin sur écran graphique ou sur papier. On aura donc toujours 2 fichiers pour décrire une figure de nom "TOTO":
  i/ le fichier de nom "lTOTO" contient la forme LUCIE textuelle ("fig TOTO ... ffig"),
  ii/ et le fichier de nom "xTOTO" contient la description textuelle des frontières et des connecteurs ("fig TOTO ... ffig").

Remarque: une amélioration souhaitable du système graphique LUCIE consisterait à superposer, sur le même écran, l'image des rectangles issus de la description LUCIE, et l'image des connecteurs issus de la description LUBRICK, afin de faciliter la

vérification visuelle des dessins.

II-C- GESTION ET STRUCTURATION DES INFORMATIONS

Nous avons décrit plus haut la représentation externe, dans des fichiers, des descriptifs de figures. Ces informations textuelles sont analysées et chargées dans la mémoire au cours de l'exécution.

Etant donné que l'on doit aussi bien manipuler des figures minuscules ayant un ou deux connecteurs que d'énormes figures pouvant avoir jusqu'à plusieurs centaines de connecteurs ou plus et de nombreuses figures internes, on doit gérer l'espace mémoire en PILE, en suivant la hiérarchie d'imbrication statique des fonctions pendant l'exécution: PASCAL gère automatiquement une pile pour les variables des procédures/fonctions, mais il est indispensable de disposer d'une autre zone gérée en Pile, par l'intermédiaire de pointeurs dont la durée de vie est liée à celle de la procédure dans laquelle ils sont déclarés.

II-C-1/ DEFINITION DES ETATS D'UNE FIGURE

Une figure peut être EXTERNE(x), OUVERTE(o) ou FERMEE(f).

- une figure est **EXTERNE** lorsque sa description se trouve dans la base de données (dans un fichier), c'est-à-dire à l'extérieur du programme LUBRICK. Sa description est obtenue par la fonction **GETFIG(NOM)**, qui retourne un pointeur sur la description lue dans le fichier "xNOM" et placée en mémoire.
  Tout placement de cette figure externe se traduira par la génération de l'instruction "figext NOM (x,y)" dans le fichier LUCIE.
  On peut appliquer n'importe quel opérateur géométrique sur une figure externe (répétition, rotation, symétrie).

- une figure est **OUVERTE** par la fonction **OPENFIG(NOM)**:
  une figure interne, de nom "NOM" est crée dans l'environnement courant; son accessibilité, ou portée, est limitée à sa famille dans l'arborescence PASCAL (sa mère, ses soeurs et ses descendants directes, les cousines étant exclues).
  La fonction OPENFIG retourne un pointeur sur une figure VIDE, seuls les opérateurs de construction qui lui seront appliqués pouvant la faire passer dans l'état NON-VIDE.
  Tant qu'une figure est ouverte, on ne peut ni lui appliquer un opérateur géométrique, ni la ranger dans la base de donnée.

- une figure devient FERMEE lorsqu'on lui applique la fonction **CLOSEFIG(pointeur)**.
  On ne peut pas fermer une figure vide; on ne peut en fait fermer qu'une figure OUVERTE et NON-VIDE.

La figure la plus englobante décrite par un programme LUBRICK devient automatiquement une figure EXTERNE lorsqu'on lui applique la fonction CLOSEFIG: sa description est transférée de la mémoire vers un fichier (procédure **PUTFIG(pointeur)** interne à LUBRICK), ce qui est la seule manière de conserver un résultat après la disparition du programme.

Cette propriété permet de "hiérarchiser" la conception, et de construire le circuit complet par sous-arbres, en conservant la description des sous-arbres dans la base de donnée: un sous-arbre peut ainsi être considéré comme une cellule de base ("leaf cell" ou feuille) par la cellule mère du sous-arbre qui l'accède par une fonction GETFIG.
Par contre, une figure interne devient FERMEE lorsqu'on lui applique CLOSEFIG, et elle peut alors être utilisée d'une manière interne dans le sous-arbre où elle se trouve.

Tout placement d'une figure INTERNE fermée produit la génération d'une instruction LUCIE "fig NOM (x,y)".

OPENFIG ⟶ OUVERTE VIDE

OUVERTE ⟶ Fonctions de Construction ⟶ OUVERTE NON-VIDE

Figure OUVERTE NON-VIDE ⟶ CLOSEFIG ⟶ FERMEE

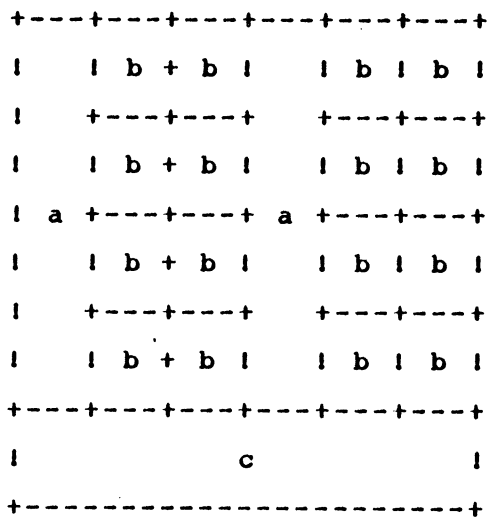
FERMEE ⟶ Fonctions Géométriques ⟶ FERMEE

Figure EXTERNE ⟶ GETFIG ⟶ FERMEE

Figure EXTERNE ⟵ PUTFIG ⟵ FERMEE

II-D- **EXEMPLE DE CONSTRUCTION**

Soit la figure 'fp', représentée par le dessin suivant, à construire à partir des 3 figures de base 'a', 'b' et 'c'.



1/ Exemple condensé de programmation en PASCAL-LUBRICK:

```
var i:pointeur;
begin
 i:=closefig(up(up(openfig('fp')
                   ,getfig('c'),1,0)
                ,repx(closefig(right(right(openfig('fi')
                                           ,getfig('a'),1,0)
                                     ,repy(getfig('b'),4),2,0
                                )
                      ,2)
                ,1,0)
              );
end;
```

2/ Autre forme équivalente (variables pointeurs):

```
var i,j,k,l:pointeur;
begin
  i:=openfig('fp');
     j:=openfig('fi');
        j:=right(j,getfig('a'),1,0);
           k:=repy(getfig('b'),4);
        j:=right(j,k,2,0);
     j:=closefig(j);
   l:=repx(j,2);
  i:=up(i,getfig('c');
  i:=up(i,l,1,0);
  i:=closefig(i);
end;
```

3/ Autre forme équivalente (fonctions imbriquées) :

```
var i:pointeur;
  function fi2:pointeur;
    function fi:pointeur;
      var vfi:pointeur;
         function repb:pointeur;
         begin
           repb := repy(getfig('b'),4)
         end;
       begin
         vfi:=openfig('fi');
         vfi:=right(right(vfi,getfig('a'),1,0),repb,2,0);
         fi :=closefig(vfi)
       end;
   begin
     fi2 := repx(fi,2)
   end;
begin
  i:=closefig(up(up(openfig('fp'),getfig('c'),1,0),fi2,1,0))
end;
```

III- LES OPERATEURS D'ASSEMBLAGE "EN-HAUT" ET "A-DROITE"

Les opérateurs EN-HAUT et A-DROITE servent à construire une figure par compaction de plusieurs autres figures, selon le principe suivant:

on enrichit la figure "p1" en court de construction d'une autre figure "p2" que l'on place:
- soit à sa droite (opérateur A-DROITE),
- soit au-dessus d'elle (opérateur EN-HAUT).

Les opérateurs EN-BAS et A-GAUCHE n'existent pas car ils impliqueraient la gestion de coordonnées négatives.

III-A- LA SYNTAXE PASCAL DES OPERATEURS EN-HAUT et A-DROITE

```
function EN-HAUT (*ou A-DROITE*)  ( p1 , p2 : pointeur;
                    nrépète , déplac : integer ) : pointeur;
```

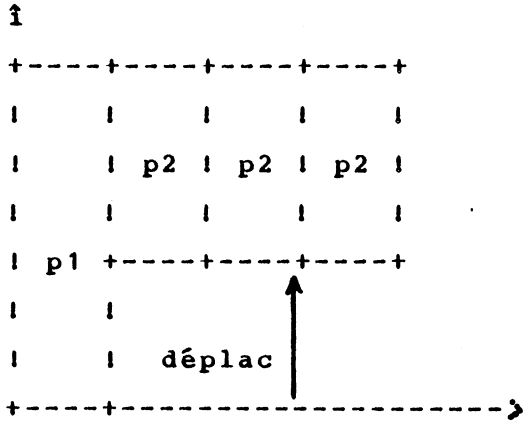
- le paramètre "p1" est le pointeur sur la figure en cours de construction. Cette figure, que nous appellerons "p1", doit être OUVERTE.

- le paramètre "p2" est un pointeur sur la figure à placer en haut (EN-HAUT) ou à droite (A-DROITE) de la figure "p1". Cette figure doit être FERMEE, elle peut être Interne au programme ou Externe.

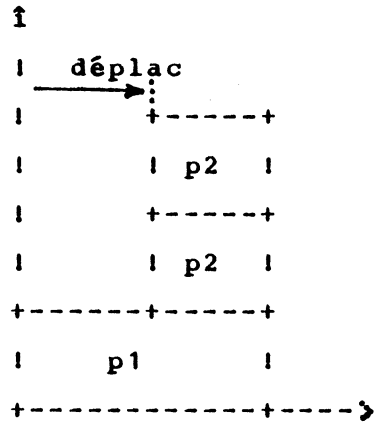
- le paramètre "nrépète" est un facteur de répétition de l'opération.

- le paramètre "déplac" est un déplacement VERTICAL pour A-DROITE, et un déplacement HORIZONTAL pour EN-HAUT.
  Ce déplacement est toujours un nombre >= 0, et il est relatif à l'un des axes de la figure courante "p1".

La figure suivante montre la combinaison d'une répétition et d'un déplacement ("nrépète">1 et "déplac">0).



A-DROITE(p1,p2,3,déplac)

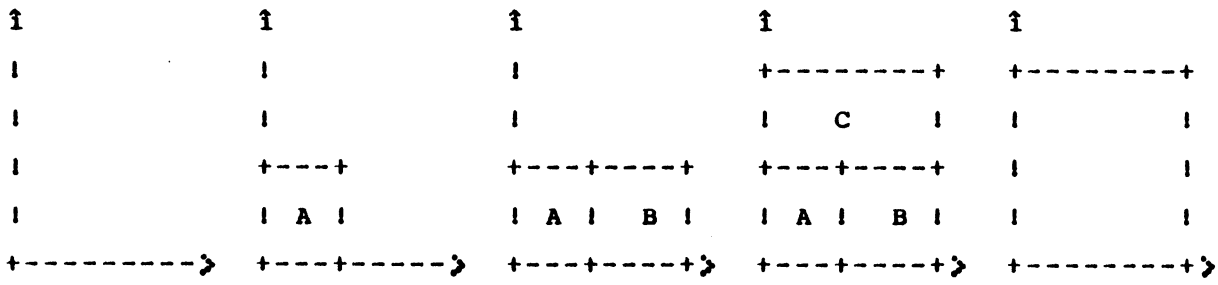


EN-HAUT(p1,p2,2,déplac)

III-B- LE PROCESSUS DE CONSTRUCTION

La définition de ces opérateurs d'assemblage correspond à un processus de construction rigide qui se déroule comme suit:

- 1/ On ouvre la figure courante (OPENFIG) (=> figure vide);
- 2/ On l'enrichit en lui plaçant des figures EN-HAUT ou A-DROITE;
- 3/ On la ferme (CLOSEFIG).



OPENFIG(x); A-DROITE(p,A); A-DROITE(p,B); EN-HAUT(p,C); CLOSEFIG(p);

L'assemblage précédent se décrit en Pascal-LUBRICK:

- d'une manière séquentielle:

```
p := OPENFIG  ('nom');
p := A-DROITE (p,GETFIG('A'),1,0);
p := A-DROITE (p,GETFIG('B'),1,0);
p := EN-HAUT  (p,GETFIG('C'),1,0);
p := CLOSEFIG (p);
```

- d'une manière condensée:

```
CLOSEFIG (EN-HAUT (A-DROITE (A-DROITE (OPENFIG('nom'),GETFIG('A'),1,0)
                    ,GETFIG('B'),1,0)
          ,GETFIG('C'),1,0)
         );
```

III-C- **SEMANTIQUE DE L'EXECUTION** de l'opérateur A-DROITE

On présente uniquement l'opérateur A-DROITE, sachant que l'opérateur EN-HAUT fonctionne d'une manière identique dans l'autre direction.

L'exécution d'une opération de placement A-DROITE se décompose, dans le cas le plus simple, en 2 étapes:

- la 1-ère étape consiste à calculer le placement de la figure "p2" à droite de la figure "p1", en comparant les 2 frontières EST de "p1" et OUEST de "p2", et en tenant compte du paramètre "déplac";
- la 2-ème étape consiste à chercher si des connecteurs EST de "p1" se trouvent "en face de" connecteurs OUEST de "p2"; dans ce cas une connexion est établie, et elle est concrétisée par la génération d'un rectangle.

Une telle sémantique correspond à une technique d'implantation

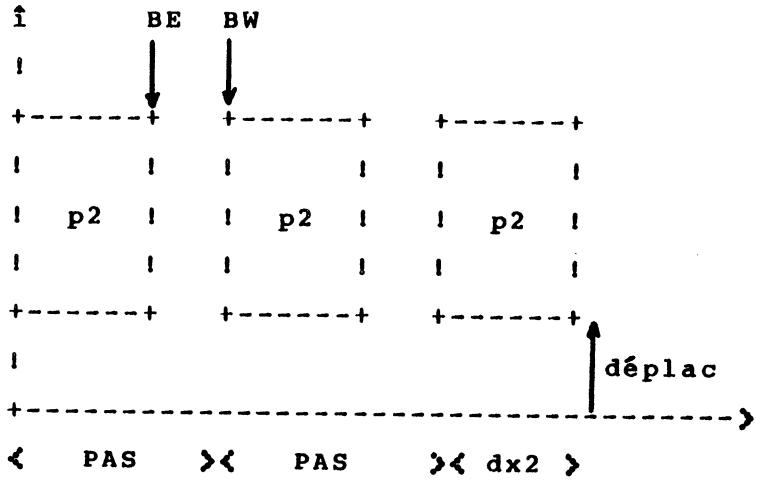
topologique:
elle ne peut être valablement mise en oeuvre que pour des cellules qui ont été, dès le départ, conçues pour se placer les unes à côté des autres, et pour s'interconnecter d'une manière systématique par des fils qui trouvent une place dans des "canaux" préexistants:
en d'autres termes, ce mécanisme de connexion automatique correspond à la définition préalable de "fils virtuels", placés à des positions fixes, les cellules ayant été conçues et dessinnées pour se placer "sous" ces fils.

Cette approche est très efficace pour les structures de type chemin de données:

- des fils horizontaux (par convention) portent des bus; ils sont situés à des ordonnées fixes pour chaque tranche de 1 bit; les connexions horizontales ne peuvent ainsi être supportées que par un **nombre fixe** de fils horizontaux, placés à des **ordonnées fixes**.

- dans le sens vertical, les cellules peuvent être répétées, et il suffit que leurs connecteurs orientés vers le NORD se trouvent aux mêmes abcisses que ceux qui sont orientés vers le SUD pour que les connexions automatiques puissent être réalisées.

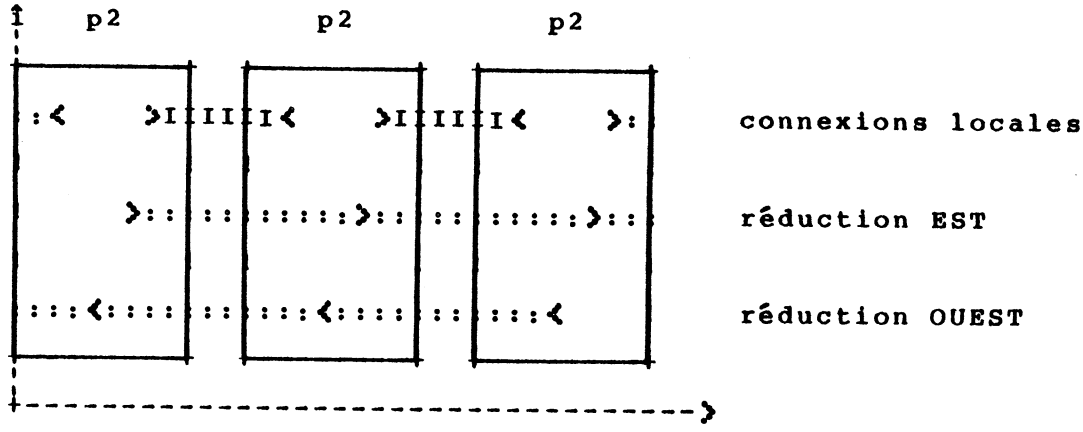
Des cas plus complexes peuvent se produire, à cause du facteur de répétition d'une part, et d'autre part lorsque 2 connecteurs orientés dans la même direction se trouvent en face: on applique alors une règle de REDUCTION, en ne conservant que le connecteur le plus éloigné de sa frontière, soit celui dont l'ECART avec la frontière est le plus grand.



Cas d'une répétition A-DROITE d'une figure vide

III-C-1/ Cas des connexions locales aux répétitions

Lorsque (nrépète > 1), il peut y avoir des connexions locales à établir entre les occurrences répétées de "p2", ou des réductions à effectuer, en appliquant la règle de l'ECART le plus grand.



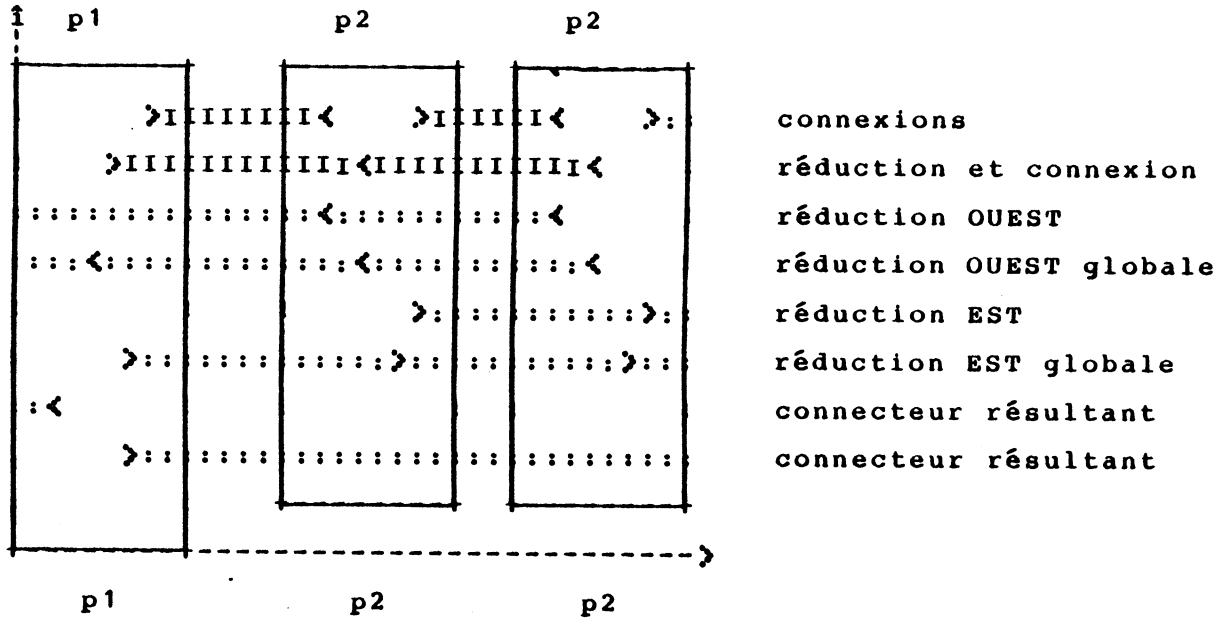
Différents cas de CONNEXIONS locales et REDUCTIONS

>IIII< = connexion réalisée

>::::: = connecteur résultant

III-C-2/ Cas général des connexions

Dans le cas le plus général, il peut y avoir des connexions locales aux répétitions, PLUS des connexions et des réductions entre la figure "p1" et les répétitions de "p2". Le traitement associé est relativement complexe. Le schéma suivant résume les différents cas traités.



Tous les cas de CONNEXIONS et de REDUCTIONS.

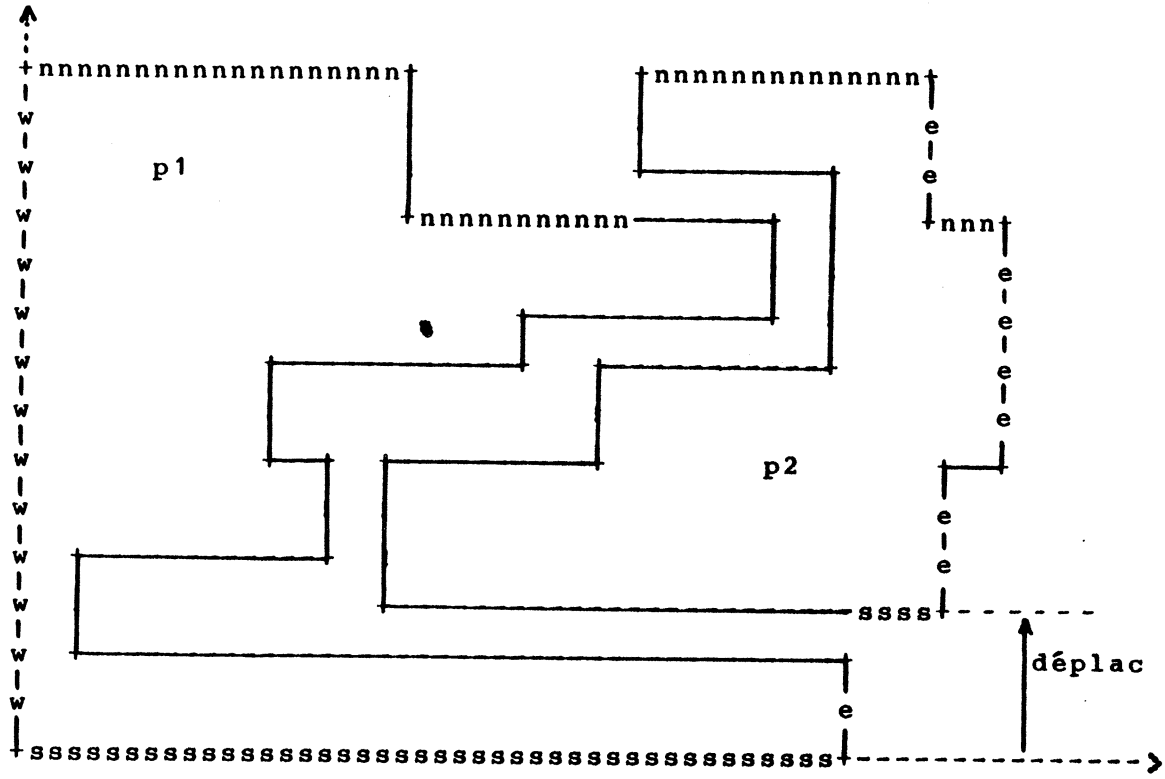
III-C-3/ Construction des frontières résultantes

Dans le cas d'une opération A-DROITE (p1,p2,..), la frontière OUEST résultante est celle de "p1", la frontière EST est celle de "p2", éventuellement décalée vers le haut de la valeur "déplac": ce décalage introduit un creux qui est répercuté dans la frontière

résultante.

Dans l'autre direction, il faut "concaténer" les deux frontières NORD et SUD, en répétant éventuellement celle de "p2".

Le problème est complexe par le fait que le tassement est fait en comparant deux frontières EST et OUEST (segments verticaux), et qu'on doit également comparer les segments horizontaux qui ferment les 2 polygones: ces derniers peuvent se cacher mutuellement. On effectue une projection sur un axe, et on ne retient que les segments qui sont à la distance la plus petite de cet axe.
Exemple:



Frontières résultantes (n=NORD, e=EST, s=SUD, w=WEST)

IV- ASSEMBLAGE DE "GROS" BLOCS FONCTIONNELS

On se place ici dans le cas de blocs (figures) qui ont été assemblés séparément, selon des techniques topologiques différentes, par exemple
- un chemin de donnée à partir de "tranches",
- et un PLA construit à partir de fils régulièrement espacés ou généré par un outil spécialisé de génération (par exemple [Ref. PAOLA]).

On connait la forme de ces blocs (leur encombrement dx et dy), ainsi que la nature et la localisation de leurs "pattes" d'entrée/sortie décrites comme des CONNECTEURS-LUBRICK.

RESTRICTION:
on se place uniquement dans la cas de DEUX figures, à connecter selon UNE seule direction, en comparant DEUX listes OPPOSEES de connecteurs.

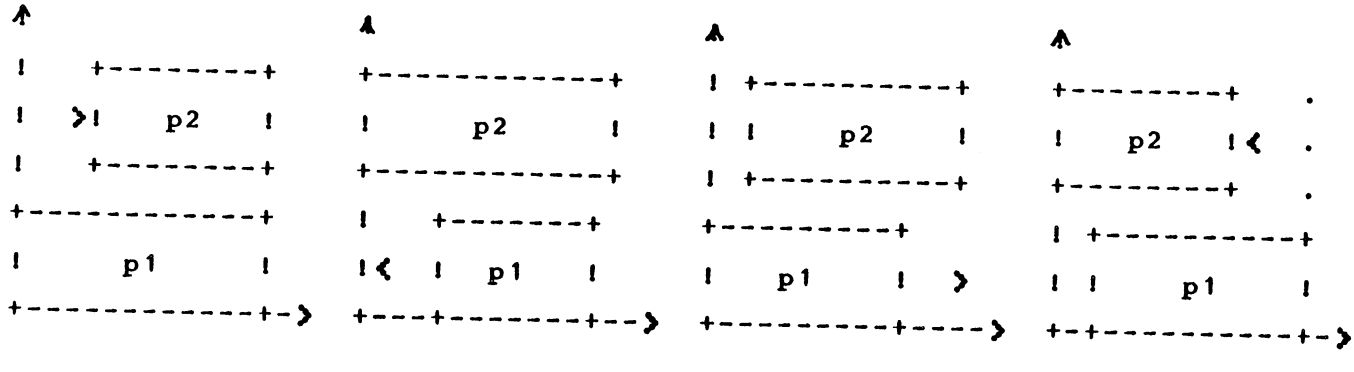
On ne peut pas, dans ce cas restrictif, résoudre TOUS les problèmes de placement et d'interconnexion de tous les blocs d'un circuit: l'Editeur de plan de masse est spécialisé dans ce travail. On se limite donc à REALISER un placement et des connexions entre DEUX figures, lorsque le concepteur du circuit le décide selon ses propres critères plus globaux que nous ne détaillerons pas ici.

Les opérations d'assemblage que nous présentons ici tentent de résoudre une partie des problèmes connus sous les noms (anglais) de RIVER ROUTING ou de CHANNEL ROUTING.

IV-A- DESCRIPTION DU PLACEMENT RELATIF DE DEUX BLOCS

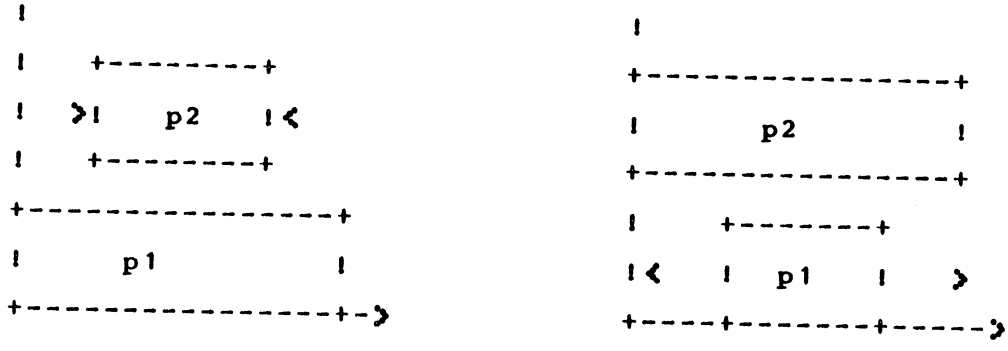
Examinons le cas du placement de la figure "p2" AU-DESSUS DE la figure "p1".

L'une des 2 figures, "p1", sert de **référence**: on spécifie le placement de "p2" par rapport à l'un des axes verticaux (EST ou OUEST) de "p1". Ce déplacement pouvant être négatif, il y a 4 combinaisons (ce qui montre d'ailleurs qu'on aurait aussi bien pu prendre "p2" comme référence).
Plutôt que d'indiquer le côté qui sert de référence par un paramètre, on utilise deux déplacements appelés LOFF (Left Offset) et ROFF (Right Offset):



(LOFF>0,ROFF=-1) (LOFF<0,ROFF=-1) (LOFF=-1,ROFF>0) (LOFF=-1,ROFF<0)

On spécifie un placement **Centré** ou réparti avec LOFF=0 et ROFF=0 pour les deux cas suivants:



(LOFF=0,ROFF=0) (LOFF=0,ROFF=0)

Dans tous les cas, la figure résultante englobe les deux figures; sa taille dépend du placement relatif, ainsi que de la zone qui sera nécessaire pour connecter les deux figures.

IV-B- LES OPERATEURS DE PLACEMENT ET CONNEXION DE 2 BLOCS

On distingue deux cas:

- le cas du **RIVER** ROUTING, pour lequel un seul niveau de fils est suffisant, car on sait par avance qu'il n'y a pas de croisement. On appellera ce type de connection **DIRECT**, et les deux fonctions LUBRICK **CUP** (CONNECT-UP) et **CRIGHT** (CONNECT-RIGHT) le réalisent.

- le cas du **CHANNEL** ROUTING, pour lequel deux niveaux de fil (POLY et ALU) sont nécessaires parce qu'il existe au moins un croisement.
  Ce cas est plus complexe, et les deux fonctions **CRUP** (Channel-Routing-UP) et **CRRIGHT** (Channel-Routing-RIGHT) le réalisent.

IV-B-1- LES OPERATEURS CUP et CRIGHT (Connexions DIRECTES)

Syntaxe PASCAL:

```
function CUP (* ou CRIGHT *)  ( p1,p2 : pointeur;
                                LOFF,ROFF,NIV : integer;
                                var TP : array[...] of pointeur )
                              : pointeur;
```

Le placement relatif des 2 figures pointées par "p1" et "p2" est donné par LOFF et ROFF, comme expliqué plus haut (en IV-A).

Le paramètre NIV indique le niveau de métalisation (Poly, ou Métal)
Le tableau TP passé en paramètre a pour indice un Numéro de TYPE de connecteur, et son utilisation dépend des différents cas de figures expliqués plus loin. Il sert à associer une information, de type pointeur, à chaque TYPE de connecteur.

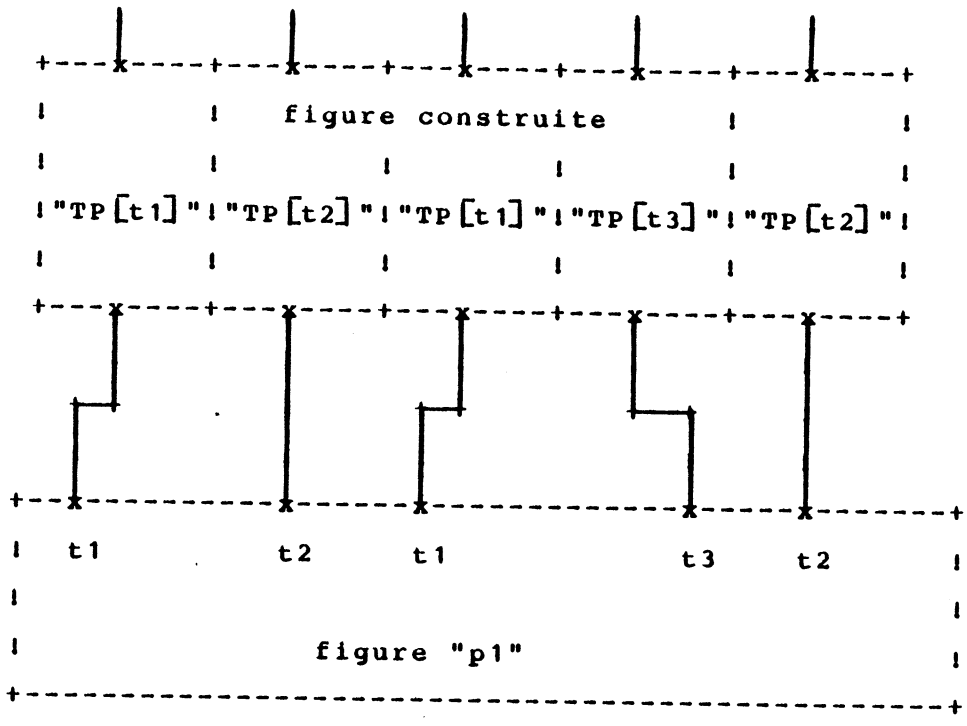
IV-B-1-a/ La connexion directe de 2 figures:
Lorsque 2 figures "p1" et "p2" sont passées en paramètre, c'est-à-dire lorsqu'aucun des 2 pointeurs ne vaut NIL, (p1<>NIL et p2<>NIL), la fonction CUP (ou CRIGHT) réalise une connexion

DIRECTE, après avoir calculé le placement spécifié par LOFF et ROFF.
La figure "p2" est placée au-dessus de la figure "p1" par l'opérateur CUP, et à droite de "p1" par l'opérateur CRIGHT.

On tient compte des valeurs trouvées dans le tableau TP:
- on ne connecte que les connecteurs de type "t" tels que (TP[t]<>NIL),
- et on ignore les autres, ce qui permet d'éliminer des connecteurs de certains TYPES qui n'ont pas à être connectés.

IV-B-1-b/ La connexion directe avec une figure "inconnue":
Lorsqu'on ne spécifie que l'une des 2 figures, c'est-à-dire que l'un des 2 pointeurs vaut NIL,
((p1<>NIL) et (p2=NIL)) ou ((p1=NIL) et (p2<>NIL))
on indique, au moyen du tableau TP comment construire la 2ème figure, en associant au type de connecteur "t" la figure pointée par TP[t].

Ainsi, si la figure "p1" a, dans l'ordre, des connecteurs NORD (pour CUP, et EST pour CRIGHT) de types (t1, t2, t1, t3, t2), la fonction construira, en utilisant l'opérateur RIGHT, une nouvelle figure constituée des figures "TP[t1]", "TP[t2]" et "TP[t3]", et réalisera ensuite la connexion de ces deux figures.
Si TP[t]=NIL, les connecteurs de type "t" sont ignorés.



Si ((p1=NIL) et (p2<>NIL)), ce sont les connecteurs SUD de "p2" (pour CUP, (ou les connecteurs OUEST pour CRIGHT), qui sont utilisés pour la construction de la nouvelle figure, et "p2" est ensuite placée au-dessus de cette nouvelle figure.

Un exemple qui montre la puissance de cette fonction est celui de l'assemblage d'une partie opérative: on obtient une figure "po" ayant plus d'une centaine de connecteurs NORD (140 pour POPY) décrivant les écarts, positions, largeurs, niveaux et TYPES des commandes et des indicateurs. Les commandes étant générées par des amplificateurs contrôlés par les signaux d'horloge (phases) ou par les horloges elles-mêmes, on a associé un NUMERO DE TYPE a chaque TYPE de commande, dans les cellules de base, et on retrouve ces numéros à la fin de l'assemblage.

Il suffit alors d'associer à chaque NUMERO de TYPE une figure, qui est l'amplificateur correspondant, en remplissant le tableau TP.

On écrit le programme Lubrick suivant:

```
begin
   (* remplit TP *)
      TP[0] := -1; (* on ignore le type 0 *)
      TP[1] := GETFIG('am1');
      TP[2] := GETFIG('am2');
            xxx
      TP[24]:= GETFIG('am24');

      (* placement centré *)
      p := CUP( GETFIG('po'),-1,0,0,poly,TP);
end;
```

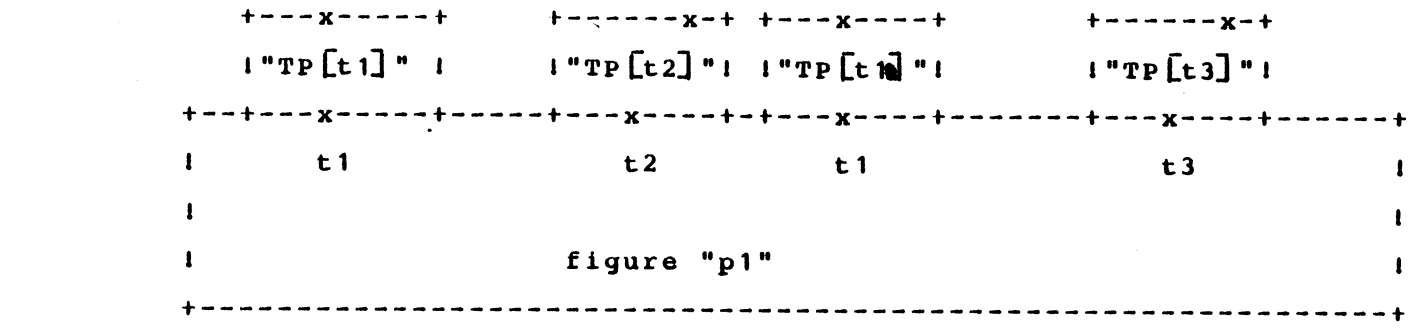
On récupère, par le pointeur "p", la figure résultante.

IV-B-1-c/ Aucun offset n'est défini:

Ce cas est une variante du cas précédent: on ne spécifie qu'une seule figure ("p1" ou "p2") et un tableau de pointeurs TP, mais on demande que chaque figure "TP[t]" soit placée sur le bord de la figure donnée.

Cette fonction est nécessaire pour ajouter de petites cellules en certains points d'une frontière, là où se trouvent certains connecteurs.
Cette variante est spécifiée en donnant (LOFF = ROFF = -1), c'est-à-dire aucune indication de placement relatif.

Exemple:



Programmation en LUBRICK correspondante:

```
begin
        TP[t1]:=GETFIG('f1');
        TP[t2]:=GETFIG('f2');
        TP[t3]:=GETFIG('f3');
    p := CUP( p1,-1,-1,-1,0,TP);
end;
```

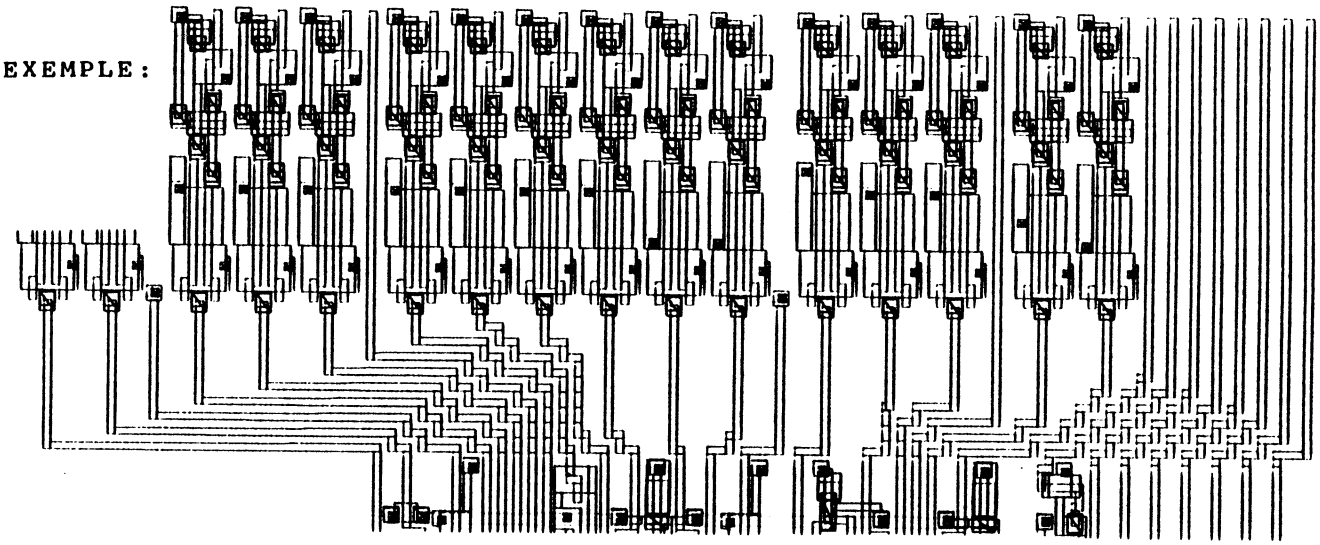
IV-B-1-d/ La réalisation des connexions:

On met en oeuvre un algorithme de génération de fils d'une seule couleur, (et sans raccords à 45 degrés dans la version courante). Cet algorithme produit des escaliers, en tenant compte des abcisses des points de départ et d'arrivée, et en évitant par des coudes les lignes qui sont utilisées par ses voisins.

Cet algorithme peut être paramétré par la couleur du fil: le POLY requiert un pas de 4 unités, le METAL un pas de 6 (règles normalisées). On peut facilement l'étendre à des fils de couleurs différentes mais restant parallèles.

Une amélioration souhaitable porterait sur la minimisation du nombre de rectangles générés.

EXEMPLE:



IV-B-2- **CONNEXIONS COMPLEXES (CHANNEL ROUTING)**

On se place toujours dans le cas restreint de deux figures "p1" et "p2".

- La function CRUP va connecter les connecteurs NORD de "p1" aux connecteurs SUD de "p2", et "p2" sera placée AU-DESSUS DE "p1", selon les valeurs de LOFF et ROFF (comme défini plus haut, en IV-A).

- La function CRRIGHT va faire de même, avec les connecteurs EST de "p1" et les connecteurs OUEST de "p2", plaçant "p2" A-DROITE DE "p1".

Définition syntaxique des fonctions:

```
function CRUP (* ou CRRIGHT *) ( p1,p2 : pointeur;
                                 LOFF,ROFF : integer;
                                 var TC : array[...] of integer;
                                 EQUIPOT : boolean )
                               : pointeur;
```

- Les paramètres "p1" et "p2", LOFF et ROFF, ont la même définition que pour les fonctions CUP et CRIGHT (définition donnée en IV-A).

- Le paramètre TC est un tableau d'entiers, dont le rôle varie selon les différents cas exposés plus loin.

- Le paramètre booléen EQUIPOT permet de distinguer entre une connexion par EQUIPOTENTIELLES et une connexion par POSITIONS.

IV-B-2-a/ Connexion par numéros d'ordre explicites:

Le tableau TC est utilisé par le concepteur pour définir les connexions à établir entre les 2 figures:

"TC[i1]=i2"

indique que le $i1^{ème}$ connecteur NORD de "p1" doit être connecté au $i2^{ème}$ connecteur SUD de "p2".

On ne peut donc spécifier qu'une bijection entre deux sous-ensembles de connecteurs.

IV-B-2-b/ connexions définies par les types:

Premier cas: (EQUIPOT = faux)

L'attribut de TYPE de chaque connecteur, et son rang dans la liste, définissent une bijection entre les deux listes par la relation suivante:

"le $i^{ème}$ connecteur NORD de "p1" de type t doit être connecté

au $i^{ème}$ connecteur SUD de "p2" de même type t".

De plus, on utilise le tableau TC pour indiquer par TC[k]=-1 qu'on doit ignorer les connecteurs de type k, et on détecte une erreur si la bijection ne peut pas être établie (on aurait pu ignorer les connecteurs qui n'ont pas de correspondant).

Cette définition réalise une connexion par NOMS, si on n'a pas 2 connecteurs NORD (ou SUD) qui ont le même attribut de type.
Elle permet également une définition mixte, avec des NOMS PATRONYMES qui désignent une famille de connecteurs de même type, le rang dans la liste des connecteurs distinguant les frères de la même famille.

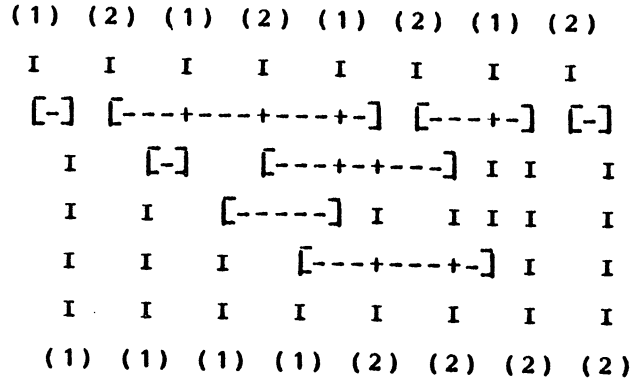
Deuxième cas : (EQUIPOT = vrai)

Chaque valeur d'un type de connecteur définit une EQUIPOTENTIELLE: tous les connecteurs NORD et SUD ayant le même attribut de type sont reliés entre eux par une ligne unique, et la relation est calculée.
Le tableau TC permet de spécifier si certaines équipotentielles doivent être ignorées (TC[k]=-1 indique qu'on ne doit pas relier les connecteurs de type k).

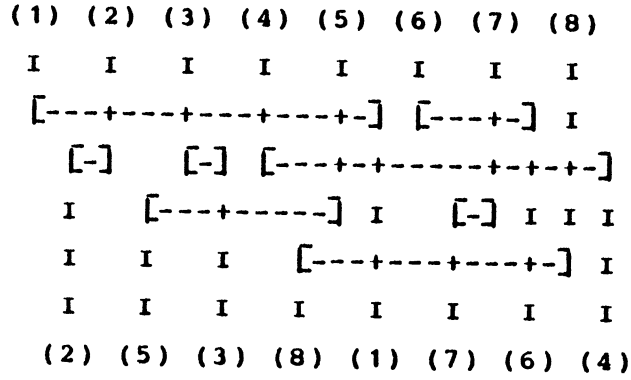
IV-B-3- EXEMPLES:

Dans les schémas qui suivent,
les "[" et "]" indiquent le début et la fin d'une équipotentielle, et dc
contact, les "x" indiquent un contact, et les "+" un croisement.

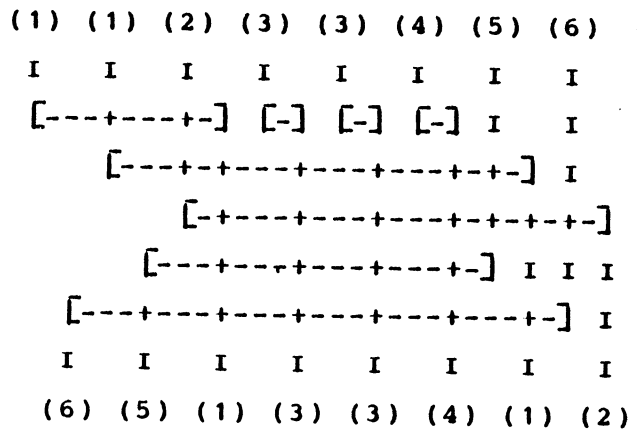
connexions par TYPES, par exemple (1)=entrée, (2)=sortie :



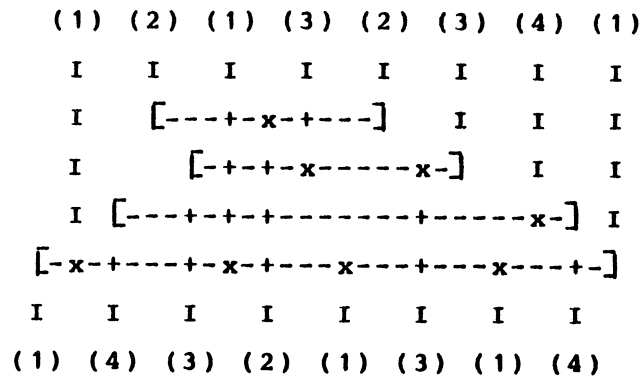
connexions par NOMS (tous les types sont différents) :



connexions par NOMS PATRONYMES (les familles (1) et (3)) :



connexions par équipotentielles ((1), (2), (3) et (4)):



Les dessins précédents sont fabriqués par un mini-programme d'évaluation qui, prenant deux figures LUBRICK, calcule le nombre de canaux nécessaires et fournit une représentation graphique symbolique sur un terminal alphanumérique.

IV-B-4- Réalisation par LUBRICK

La génération du dessin des masques réels est faite par LUBRICK, en appelant la function CRUP (ou CRRIGHT). Le cas des règles du CMP, avec des "bossages" de métal autour des contacts, implique un traitement non trivial pour compacter au maximum les lignes de métal et les contacts.

On génère d'abord la figure "channel" contenant les canaux de routage (en métal) et les contacts, puis on fait appel deux fois à la fonction CUP:

si on demande:

CRUP(p1,p2,...),

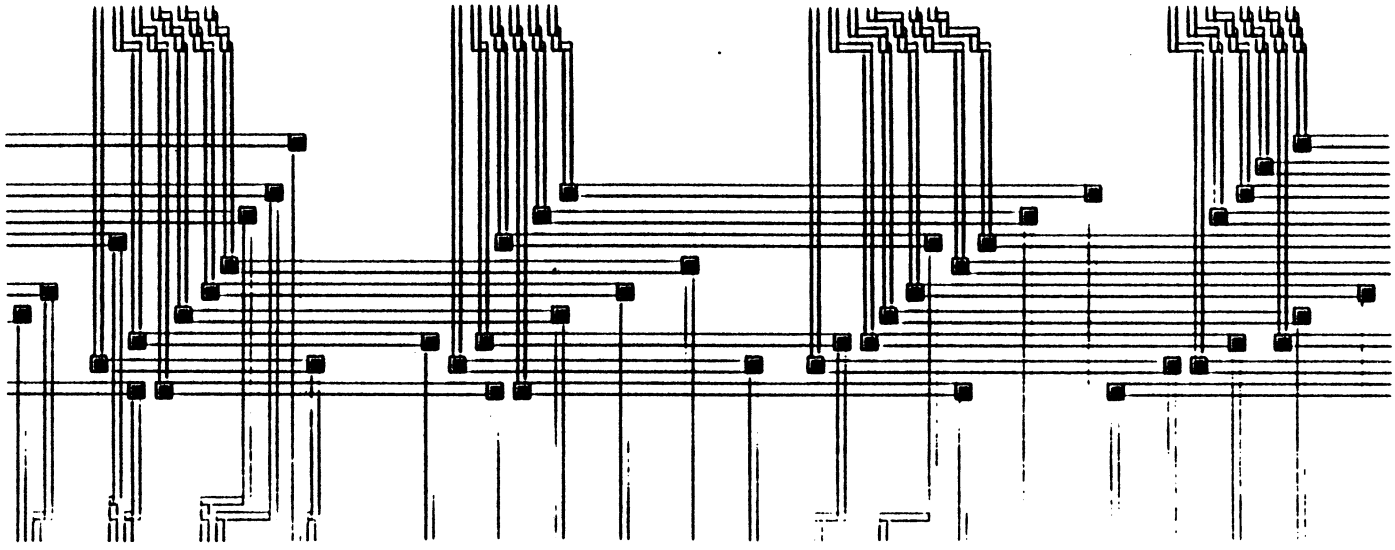
le programme LUBRICK:

- construit la figure interne "channel",

- puis exécute:

CUP(CUP(p1,channel,...), p2,...).

Exemple:



V- LES OPERATEURS GEOMETRIQUES

Les opérateurs géométriques s'appliquent uniquement sur des figures FERMEES, qui sont soit externes (accédées par GETFIG), soit internes après l'application de CLOSEFIG.

L'exécution d'un opérateur géométrique retourne un pointeur sur la figure résultante qui est FERMEE, et produit dans le fichier LUCIE les instructions de description d'une nouvelle figure interne.

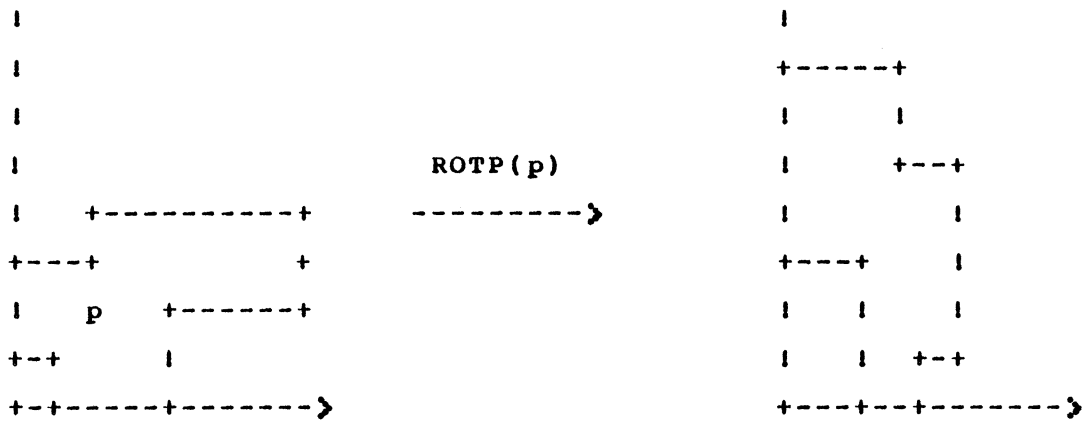
Syntaxe PASCAL des opérateurs:

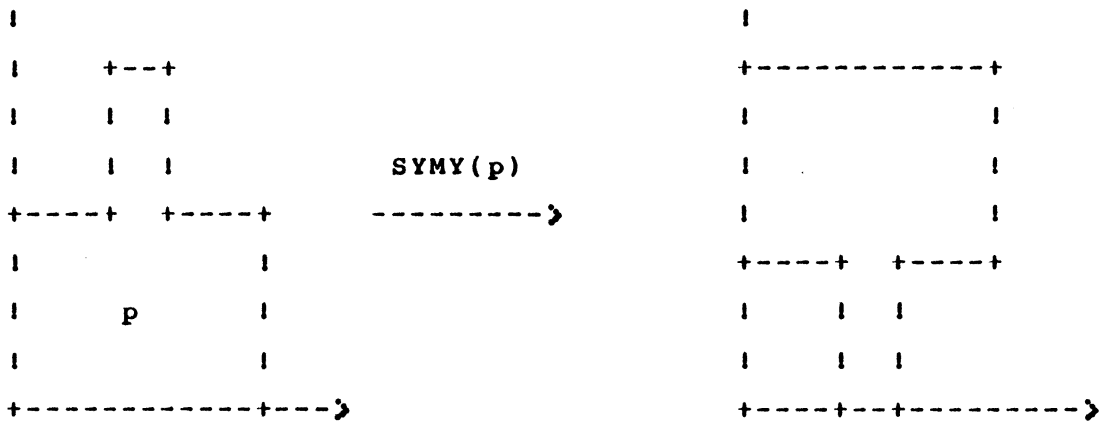
- rotation de +90° :
  function ROTP (p:pointeur) : pointeur;
- rotation de -90° :
  function ROTM (p:pointeur) : pointeur;
- symétrie selon l'axe vertical
  suivie d'une translation vers le bas:
  function SYMX (p:pointeur) : pointeur;
- symétrie selon l'axe horizontal
  suivie d'une translation vers la gauche:
  function SYMX (p:pointeur) ; pointeur;
- répétition dans le sens horizontal:
  function REPX (p:pointeur; n:integer) : pointeur;
- répétition dans le sens vertical:
  function REPY (p:pointeur; n:integer) : pointeur;

Les opérateurs précédents utilisent tous l'une des valeurs DX ou DY qui définit le cadre dans lequel s'inscrit la figure, cadre par rapport auquel sont définis les connecteurs et les frontières.
Il a déjà été mentionné que le point (0,0) de la figure LUBRICK doit être le même que le point (0,0) de la figure LUCIE;
il faut également noter que les opérateurs géométriques précédents ne fournissent un BON RESULTAT QUE SI les valeurs (DX,DY) définissent LE PLUS PETIT RECTANGLE ENGLOBANT de la figure LUCIE.

SUGGESTION: afin de ne pas avoir de surprise lorsqu'on fait tourner des figures par ROT ou SYM, on définira les valeurs DX et DY des cellules de base égales aux valeurs d'encombrement définies dans LUCIE, c'est-à-dire les valeurs XMAX et YMAX calculées par le traducteur LUCIE.

EXEMPLES:





La combinaison de RIGHT et de REPY, ou de UP et de REPX permet de plus de décrire facilement des assemblages matriciels, en utilisant l'une des 2 formes équivalentes:

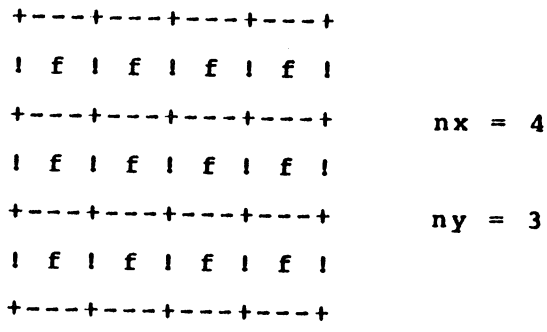
pour une matrice (nx * ny) d'une figure "f":

```
RIGHT(p,REPY(f,ny),nx,0)
```

ou

```
UP   (p,REPX(f,nx),ny,0)
```

qui produisent:



CARACTERISTIQUES DU SYSTEME "LUBRICK"

A/- LES VERSIONS OPERATIONNELLES:

Le système LUBRICK est un programme écrit en PASCAL, d'environ 2800 lignes. Il en existe deux versions:

- la 1-ère version est opérationnelle sous le système MULTICS. Elle met à profit l'existence d'une édition de lien dynamique sur ce système: c'est le programme LUBRICK lui-même qui "appelle" le programme écrit par l'utilisateur, après qu'il ait été compilé.

Structure d'un programme Lubrick-Multics:

```
program USER;
var .......          (* déclarations des variables *)
function up(...); EXTERNAL;
   x
   x                 (* déclarations des fonctions Lubrick *)
   x
begin
   i:=OPENFIG('essai');
        x
   i:=CLOSEFIG(i);
end.
```

Remarque: le début est le même pour tous les programmes, il est donc mis dans un fichier, et appelé par une directive "INCLUDE".

- la 2-ème version est opérationnelle sous le système PASCAL-UCSD, et fonctionne sur les microordinateurs Pascal MicroEngine ou Pascaline, nécissitant seulement une mémoire de 64 Koctets pour s'exécuter. Dans cette version, le programme a été découpé en "unités de compilation" (4) contenant comme "points d'entrée" les noms des fonctions LUBRICK accessibles à l'utilisateur. C'est,

pour cette version, le programme écrit par l'utilisateur qui est exécuté, l'édition de lien étant faite avant son lancement.

Structure d'un programme LUBRICK:

```
program essai;
uses lub1,lub2,lub3,lub4; (* unités Lubrick *)
var .....                 (* déclarations de variables *)
begin
  initlub;                (* initialisation *)
  i:=OPENFIG('essai');
    x
    x
  i:=CLOSEFIG(i);
end.
```

B/- ADAPTATION A DE NOUVELLES REGLES TECHNOLOGIQUES

Le système Lubrick n'est dépendant de la technologie que pour certaines constantes qui définissent:

- le calcul des distances ("gardes") entre certains niveaux de masque; la fonction interne à Lubrick FGARDE doit ainsi être reprogrammée, et les distances entre les niveaux sont définies dans un tableau à deux entrées sous la forme:

  GARDE[1,3] := 1; ....

- la correspondance entre les noms symboliques et les valeurs entières qui codent les niveaux de masque est définie dans un tableau sous la forme:

  COULEUR[1] := 'md'; ....

- la largeur minimum des lignes qui supportent les connecteurs est définie dans un tableau sous la forme:

  LARGEURMIN[1] := 2; ....

On peut ainsi particulariser le système très facilement pour l'adapter à une nouvelle technologie (ceci a été fait pour le N-MOS du CMP, un C-MOS symbolique, et le C-MOS du futur CMP-CMOS).

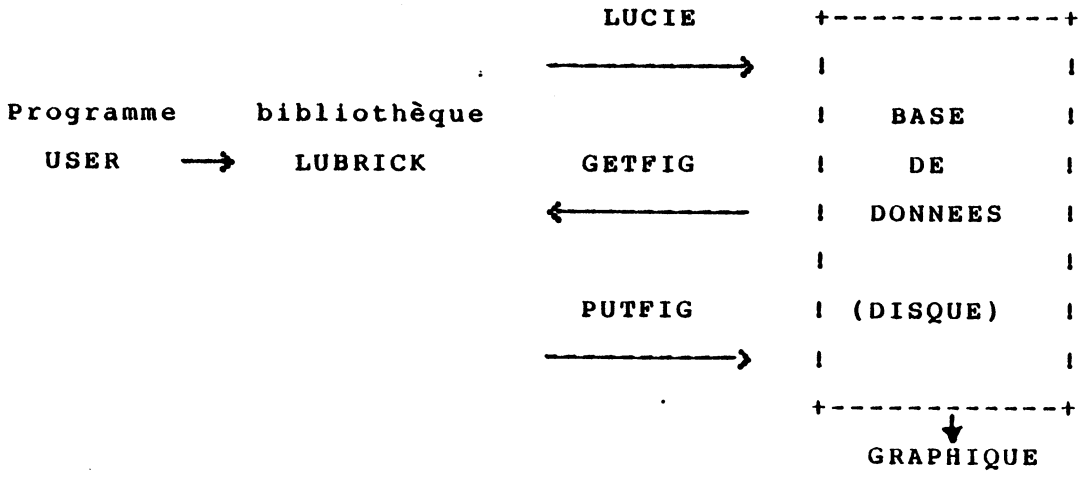
C/- LES ASPECTS "SYSTEME" ET "ENTREE/SORTIE".

Le programme Lubrick utilise 3 fichiers logiques, de type TEXT:

- le fichier écran (OUTPUT), pour afficher une trace de l'exécution et d'éventuels messages d'erreur;

- le fichier LUCIE, ouvert avec le nom de la figure principale construite par le programme (paramètre du 1-er appel de OPENFIG), dans lequel sont générées toutes les instructions Lucie de construction du dessin de cette figure.

- un fichier temporaire FICH qui est:
  ouvert en lecture à chaque appel de la fonction GETFIG, pour lire la description des frontières et des connecteurs d'une figure EXTERNE;
  ouvert en écriture à la fin du programme, pour réaliser par PUTFIG l'écriture sur disque de la description de la figure construite.



MANUEL D'UTILISATION DE LUBRICK VERSION 2

A- PRESENTATION DE L'ENVIRONNEMENT SYSTEME

Le système LUBRICK VERSION 2 est opérationnel sous UNIX 4.1; il est programmé en PASCAL (compilateur de la station METHEUS CV de Silicon Valley Software Inc.), un PASCAL très voisin du PASCAL-UCSD possédant les unités de compilation et les chaines de longueur variable (strings).

Cette version de LUBRICK est liée à la station de conception METHEUS CV, tant pour le langage de programmation utilisé que pour les formats des fichiers d'entrée et de sortie.

A-1- LES MODULES DU SYSTEME LUBRICK

Le système LUBRICK VERSION 2 comprend trois modules:

- un module exécutable, appelé PHL2LUB, qui analyse un fichier descriptif d'une cellule dessinée (".PHL" = PHysical Layout) pour en extraire les FRONTIERES et les CONNECTEURS, les premières servant aux calculs des placements relatifs, les seconds aux traitements des connexions physiques et logiques. Ce module génère un fichier de type ".LUB".

- un module pré-compilé, appelé LUBRICK.OBJ, qui contient toutes les fonctions dy système LUBRICK. Ce module est accédé à la fois pour la compilation et pour l'édition de lien des programmes PASCAL écrits par l'utilisateur, en mode "compilé".

- un module exécutable, appelé INTERLUB, qui est l'interpréteur des commandes du système LUBRICK, en mode "interprété".

Le système LUBRICK VERSION 2 utilise un fichier technologique, contenant un sous-ensemble des règles de dessin.

Ce fichier est utilisé à la fois par PHL2LUB pour la vérification de la largeur des canaux de routage, et par LUBRICK lui-meme pour les comparaisons de frontières.

Les fichiers descriptifs des cellules (.PHL et .LUB) contiennent le NOM de la technologie, ce qui permet de trouver automatiquement le fichier technologique.

A-2- LES PHASES D'EXECUTION ET LES FICHIERS :

Première phase = analyse d'une cellule:

fichier .PHL ---> PHL2LUB ---> fichier .LUB
---> fichier .PHL validé

Mode compilé :

fichier .PAS ---> Compilateur PASCAL ---> module compilé
---> édition de liens avec LUBRICK.OBJ ---> module exécutable
---> exécution ---> résultats

Mode interprété :

fichier .COM ---> INTERLUB ---> résultats

Résultats :

fichier .PHL = description hiérarchique du dessin (layout)

fichier .LUB = description des frontières et connecteurs résultants

fichier .IMS = description hiérarchique du réseau logique fabriqué.

B- PRESENTATION DES NOTIONS DE CELLULE

On distingue:

1- les cellules de base, qui sont dessinées "à la main" (à l'aide d'un éditeur graphique) et ne contiennent que des rectangles (ou trapèzes pour les segments à 45 degrés),

2- les cellules hiérarchisées qui contiennent:

- des "appels" ou "placements" de cellules de plus bas niveau,
- des rectangles qui matérialisent des connexions entre les cellules précédentes.

Ce sont ces dernières cellules (hiérarchisées) qui sont fabriquées par LUBRICK. La manière de les fabriquer est décrite:
- soit par un programme PASCAL (mode "compilé"),
- soit par un fichier de commande (mode "interprété").

Le choix entre les 2 modes (interprété et compilé) dépend de la complexité de la description: en effet le langage de commande du mode "interprété" ne permet pas de décrire des constructions complexes qui nécessitent l'utilisation d'un langage algorithmique. Par contre, en mode "compilé", le système LUBRICK est complètement immergé dans PASCAL, et toute la puissance de ce langage est ainsi disponible.

B-1- LES CELLULES DE BASE

Les cellules de base contiennent des rectangles (et des trapèzes) qui représentent un certain montage électronique: l'ensemble de ces rectangles doit respecter des REGLES DE DESSIN, imposées par la technologie, et vérifiables par l'appel d'un programme de vérification (DRC = Design Rule Checker). Il représente d'autre part un certain montage électronique, constitué par un réseau de transistors, capacités et résistances, qui peut être extrait du dessin par l'appel d'un programme d'extraction (par exemple PHLEX = PHysical Layout EXtractor).

La validation d'une cellule de base est terminée lorsque:
1/- aucune règle de dessin n'est violée,
2/- les géométries des transistors et leurs interconnexions réalisent un montage satisfaisant tant sur le plan de la fonction logique réalisée que sur celui des performances, immunité aux bruits, etc. Ces dernières propriétés sont vérifiées par des simulations logiques et/ou électriques effectuées sur le réseau de transistors extrait du dessin (en utilisant PHLEX de Metheus ou PHL2SPI de Bull-Systèmes).

Une cellule ainsi validée peut alors être utilisée pour construire d'autres cellules plus complexes, par assemblage de plusieurs cellules validées.

B-2- LES CELLULES COMPLEXES

Une cellule complexe est hiérarchisée, en ce sens qu'elle fait appel à des cellules moins complexes: c'est un assemblage de cellules qui peuvent être soit des cellules de base (feuilles de la hiérarchie), soit des cellules elles-mêmes hiérarchisées.

La description du dessin d'une cellule complexe est hiérarchique; cette hiérachie peut être "mise à plat", pour arriver jusqu'au niveau des rectangles contenus dans les cellules de base.
De même, on peut "mettre à plat" un schéma d'interconnexion de cellules pour arriver jusqu'au niveau des transistors contenus dans les cellules de base.

B-3- L'ASSEMBLAGE DES CELLULES

Le système LUBRICK permet un assemblage hiérarchique de cellules: cet assemblage est décrit par un programme, dont l'exécution fabrique une cellule (c'est-à-dire des fichiers qui décrivent cette cellule).

L'assemblage de cellules comporte 2 aspects:
- le placement, qui doit respecter les règles de dessin;
- la réalisation des interconnexions.

Le système LUBRICK réalise automatiquement à la fois le placement spécifié par le concepteur (placement A DROITE DE, ou AU-DESSUS DE), et les interconnexions qui sont impliquées soit par le placement demandé, soit par une relation logique.

B-4- PREPARATION DES CELLULES DE BASE

B-4-a- LA NOTION DE CARCASSE:

Les cellules de base s'inscrivent dans une boite: c'est le plus petit rectangle qui englobe tous les rectangles internes, appelé aussi CARCASSE ou BBOX pour Bounding Box. Cette boite n'a pas d'existence matérielle, mais elle sert de référence et indique une limite pour le programme PHL2LUB qui vérifie la VALIDITE de la cellule de la manière suivante:

1/- la cellule doit avoir été controlée par le programme de vérification des règles de dessin (sa description l'indique explicitement) et elle ne contient

aucune indication d'erreur (rectangles de niveau ERREUR);

2/- pour chaque point de connexion de la cellule, il existe un CANAL DE ROUTAGE, matérialisé par un rectangle, entre le point de connexion et la CARCASSE, canal d'une largeur suffisante pour contenir un fil de connexion.

On vérifie ainsi, automatiquement, que la cellule ne contient aucune erreur interne de dessin, et que les connexions qui seront faites entre les cellules ne pourront pas introduire d'erreurs de dessin.

LA DEFINITION DES CONNECTEURS:

B-4-b- Un connecteur est défini TOPOLOGIQUEMENT par:

- un RECTANGLE,
- une ORIENTATION (NORD ou EST ou SUD ou OUEST).

Ces 2 informations définissent un segment de droite, orienté vers l'extérieur de la cellule (et un rectangle doit exister entre ce segment et la carcasse).

Les orientations sont codées par un nombre entier selon le tableau:

NORD = 1

OUEST = 4 EST = 2

SUD = 3

Cas particulier:
un connecteur d'orientation NULLE (=0) est ignoré; ceci permet de définir des noeuds internes à la cellule, pour nommer les équipotentielles internes qui sont ainsi repérables pour les simulations.

B-4-c- Un connecteur est défini LOGIQUEMENT par:

- un NOM LOGIQUE (chaine de caractères),
- un TYPE LOGIQUE, codé par un nombre entier positif ou nul.

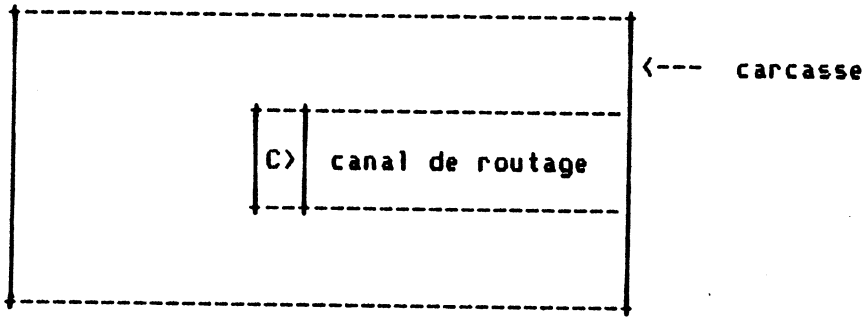
Cas particulier:

Le codage du type logique est libre, sauf la valeur ZERO qui est réservée pour définir des CANAUX DE TRANSPARENCE: un connecteur de type logique nul définit un canal de transparence horizontal (ou vertical), sur toute la largeur (ou hauteur) de la cellule, sans qu'il y ait une connexion à l'intérieur de la cellule.

On définit ainsi qu'un fil pourra passer par-dessus la cellule, sans s'y connecter, mais sans violer les règles de dessin en entrant en conflit avec des rectangles internes.

La déclaration de ces canaux de transparence est obligatoire, et leur existence est vérifiée à la fois par le controle local (DRC) et par LUBRICK au cours des assemblages.

Exemple de connecteur orienté vers l'EST:



Les connecteurs sont définis sous l'éditeur graphique PHLED en utilisant la notion de PORT GRAPHIQUE qui est décrit par:

- un rectangle,
- un niveau de masque,
- un nom logique (chaine de caractères),
- un premier attribut pour coder le type logique,
- un second attribut pour coder l'orientation.

C- DESCRIPTION D'UN ASSEMBLAGE DE CELLULES

C-1- LUBRICK INTERPRETE

Le programme INTERLUB analyse un fichier de commandes. Les commandes sont écrites dans un langage très simple dont voici la définition.

C-1-a- Les variables de l'interpréteur :

52 variables, dont le nom est constitué d'un DOLLAR ou d'un "&" suivi d'une LETTRE, sont utilisables. Elles peuvent contenir:
- soit une valeur entière,
- soit un pointeur sur une cellule.

Elles sont initialisées par la commande SET, sous l'une des 2 formes suivantes:

SET ⟨variable⟩ ⟨nombre-entier⟩ ---⟩ ⟨variable⟩ := ⟨nombre-entier⟩;

SET ⟨variable⟩ ⟨chaine⟩ ---⟩ ⟨variable⟩ := GETFIG(⟨chaine⟩);

La fonction LUBRICK GETFIG (⟨chaine⟩) analyse le fichier de nom "⟨chaine⟩.LUB" qui décrit les frontières et les connecteurs de la cellule de nom "⟨chaine⟩".

Ces variables peuvent ensuite être référencées comme opérandes des commandes de placement.

L'interpréteur vérifie, lorsqu'une variable est référencée:
- que son type (entier ou pointeur) est correct,
- que sa valeur a été initialisée.

La commande SET est également utilisable pour forcer les valeurs des variables de l'environnement d'exécution de LUBRICK qui sont:

- la variable PROMPT qui, lorsque sa valeur est TRUE, arrête l'exécution entre deux commandes (sa valeur par défaut est FALSE),
- la variable TRACECONN qui, lorsque sa valeur est TRUE, donne une TRACE sur l'écran de toutes les interconnexions réalisées (sa valeur par défaut est FALSE);

- les variables PATHLUB, PATHPHL et PATHIMS qui permettent d'indiquer un chemin d'accès, dans le système de fichiers UNIX, pour les fichiers de type .LUB, .PHL et .IMS (leur valeur par défaut est le directory courant).

C-1-b- Les commandes d'OUVERTURE et FERMETURE des cellules .

Les 2 commandes (() = optionnel) :

OPENFIG ⟨chaine⟩

CLOSEFIG (⟨variable⟩)

permettent de créer une hiérarchie de cellules, par imbrication de blocs.

Exemple:

```
OPENFIG A
  OPENFIG B
    OPENFIG C
    description de A:B:C
    CLOSEFIG &C
    OPENFIG D
    description de A:B:D
    CLOSEFIG &D
  description de A:B qui référence &C et &D
  CLOSEFIG &B
description de A qui référence &B
CLOSEFIG
```

La dernière commande CLOSEFIG ferme la cellule principale, et produit les fichiers .LUB, .PHL et .IMS qui la décrivent.

Les commandes CLOSEFIG intermédiaires fabriquent des cellules dites internes, dont le nom est construit automatiquement par LUBRICK (exemples: A:B, A:B:C, A:B:D).
Les cellules internes doivent obligatoirement être pointées par une variable de type pointeur, spécifiée en opérande de CLOSEFIG, pour pouvoir être appelées dans les commandes de construction de la cellule-mère.

C-1-c- Les commandes de placement .

Ces commandes permettent de placer une nouvelle cellule A DROITE DE (RIGHT), ou AU DESSUS DE (UP) la cellule courante (celle qui a été ouverte par la dernière commande OPENFIG).

La syntaxe de ces commandes est la suivante (() = optionnel) :

((répète)) (UP/RIGHT) ((transf)) (cellule)

ou

((répète)) (UPSLICE/RIGHTSLICE) ((transf)) (cellule) (pas)

(répète) ::= (nombre-entier) ou (variable-entière)

(transf) ::= (nombre-entier) ou (variable-entière)

(cellule) ::= (nom-de-cellule) ou (variable-pointeur)

(pas) ::= (nombre-entier) ou (variable-entière)

Les commandes UP et RIGHT correspondent à un placement AUTOMATIQUE, c'est-à-dire au placement minimal donné par la comparaison des frontières, en tenant compte des règles de dessin.

Le placement par la commande UP:
- l'abcisse du placement est définie par les connecteurs NORD et SUD les plus à gauche, s'ils existent, sinon par alignement des carcasses;
- l'ordonnée du placement est alors calculée en comparant les 2 frontières NORD et SUD.

Le placement par la commande RIGHT:
- l'ordonnée du placement est définie par les connecteurs EST et OUEST les plus bas, s'ils existent, sinon par alignement des carcasses;

Les commandes UPSLICE et RIGHTSLICE sont utilisées lorsque le PAS du placement

est connu, par exemple dans le cas d'une structure BIT-SLICE pour laquelle tous les bits sont implantés avec la même hauteur. Dans ce cas, LUBRICK vérifie que le placement demandé est possible.
Le pas est la distance entre le point de référence de la cellule courante et celui de la cellule placée. Après le placement, c'est ce dernier point qui devient point de référence de la cellule courante.
Le point de référence d'une cellule de base est défini comme un CONNECTEUR d'orientation NULLE ou comme une étiquette graphique (LABEL). Dans les 2 cas son nom doit être égal à "REF".

⟨répète⟩ est un facteur de répétition de la commande (sa valeur par défaut est UN).
Exemple: 16 RIGHT ARO indique qu'on place 16 fois la cellule de nom "ARO" à droite de la cellule courante.

⟨transform⟩ est le code de la transformation géométrique qu'il faut appliquer à la cellule avant de la placer. Les codes des transformations sont:

1	rotation de 90 degrés
2	" 180 "
3	" 270 "
4	symétrie autour de l'axe des Y
5	opération 4 suivie de opération 1
6	" 4 " " 2
7	" 4 " 3

Ces transformations sont toujours suivies d'une translation générée par LUBRICK.

ETUDE DES INTERCONNEXIONS

Après avoir calculé le placement relatif, LUBRICK étudie les connexions à réaliser en comparant les deux listes de connecteurs opposés (NORD et SUD, ou EST et OUEST); les connecteurs de ces 2 listes doivent obligatoirement se correspondre 2 à 2 selon les cas suivants:

- 2 connecteurs opposés ---⟩ génération d'un rectangle;
- 1 connecteur et un canal transparent)
  ou) ---⟩ pas de connexion
- 2 canaux transparents)

C-1-d- Les commandes de renommage des ports

Les connecteurs décrits dans les cellules de base définissent des PORTS LOGIQUES par leur NOM LOGIQUE.
Lorsqu'une cellule est placée dans la cellule courante, LUBRICK crée une INSTANCE de cette cellule, et lui donne automatiquement un NUMERO D'INSTANCIATION, ce qui permet de définir des noms uniques sous la forme:

<nom-de-cellule>=<numéro-d'instance>.<nom-de-port-logique>

Les NUMEROS D'INSTANCE sont incrémentés par LUBRICK, en partant de la valeur ZERO (la première cellule placée a pour numéro ZERO).

Les commandes de renommage sont:

1/ RENAMEPORT <chaine1> <chaine2>

le PORT de nom automatique <chaine1> prend le nom <chaine2>;

2/ MULTIRENAMEPORT <indice1> <indice2> <chaine1> <chaine2>

<chaine1> contient deux caractères "?" consécutifs, ce qui indique que tous les ports dont les noms sont obtenus en remplaçant "??" par les valeurs entières comprises entre <indice1> et <indice2> doivent être renommés.
Si <chaine2> ne contient pas la suite "??", tous les noms définis par <chaine1> et les indices sont remplacés par <chaine2>.

Exemple:

```
MULTIRENAMEPORT 0 3 ARO=??.BUS BUS(??)

        ARO=00.BUS  <---  BUS(00)
        ARO=01.BUS  <---  BUS(01)
        ARO=02.BUS  <---  BUS(02)
        ARO=03.BUS  <---  BUS(03)
```

Exemple avec <chaine2> constante:

```
MULTIRENAMEPORT 0 1 ARO=??.GND GND

        ARO=00.GND  <---  GND
        ARO=01.GND  <---  GND
```

Exemple avec un numéro d'instance fixe:

```
MULTIRENAMEPORT 15 16 ARO=31.C?? C(??)

        ARO=31.C15  <---  C(15)
        ARO=31.C16  <---  C(16)
```

C-1-e- Renommage des instances de cellules

Les commandes RENAMEINSTANCE et MULTIRENAMEINSTANCE ont la même syntaxe que les commandes de renommage des ports. Une restriction cependant: on ne peut pas donner le même nom à plusieurs instanciations de cellules différentes.
Cette commande permet de personnaliser certaines instanciations, par exemple en baptisant RI la cellule REGISTRE=12 pour faire le lien avec un schéma logique dans lequel ce même registre est appelé RI.

C-1-f- Renommage des noeuds internes

Les noeuds internes établissent les connexions entre les ports des cellules internes. Leur renommage est possible, par la commande:

```
RENAMENODE <chaine1> <chaine2> <chaine3>
```

dans laquelle <chaine1> et <chaine2> sont des noms de ports internes, et <chaine3> le nouveau nom du noeud.

Exemple:

```
RENAMEPORT ual=00.CIN  CIN
RENAMENODE ual=00.COUT ual=01.CIN COUT(00)
RENAMENODE ual=01.COUT ual=02.CIN COUT(01)
```

```
begin
      i:=OPENFIG('ARO');

          j1:=GETFIG('CELL1');
          j2:=GETFIG('CELL2');

          for k:=0 to 31
          do if (k mod 4)=0 then i:=UP(i,j1,1)
                            else i:=UP(i,j2,1);
      i:=CLOSEFIG(i);
 end.
```

C-2- LUBRICK "compilé"

Le système LUBRICK "compilé" est immergé dans le langage PASCAL en ce sens que le concepteur écrit un programme PASCAL dont l'exécution produit l'assemblage de cellules de base pour fabriquer une nouvelle cellule.

La directive "uses LUBRICK;" de PASCAL donne accès à toutes les fonctions et procédures du module LUBRICK.OBJ, qui est le résultat de la compilation de l'unité LUBRICK.PAS (3650 lignes de PASCAL).

Les procédures et fonctions accessibles sont, à une forme syntaxique près, les mêmes que celles qui sont accessibles en mode "interprété". Leur utilisation requiert cependant une bonne maitrise du langage PASCAL. On trouvera plus loin la définition de ces fonctions.

Exemple de LUBRICK

"compilé":	"interprété":
program ARO;	
uses LUBRICK;	
var i,j : integer;	
begin	
i:=OPENFIG('ARO');	OPENFIG ARO
j:=GETFIG('URE');	SET &j URE
i:=RIGHT(i,MIRRORY(j),1);	RIGHT 4 &j
i:=RIGHT(i,GETFIG('EX'),3);	3 RIGHT EX
i:=RIGHT(i,j,1);	RIGHT &j
i:=CLOSEFIG(i);	CLOSEFIG
end.	

L'exemple précédent montre une construction simple pour laquelle le mode interprété est suffisant; par contre, si la construction dépend de paramètres, ou si elle requiert de l'algorithmique, le mode compilé devra être utilisé.

Exemple:

```
program ARO;
uses LUBRICK;
var i,j1,j2,k : integer;
```

```
RENAMENODE ual=02.COUT ual=03.CIN COUT(02)
    ---
RENAMENODE ual=30.COUT ual=31.CIN COUT(30)
RENAMEPORT ual=31.COUT COUT(31)
```

On peut également utiliser la commande:

MULTIRENAMENODE ⟨indice1⟩ ⟨indice2⟩ ⟨chaine1⟩ ⟨chaine2⟩ ⟨chaine3⟩

Exemple:

```
MULTIRENAMENODE 0 31 ual=??.OUT latch=??.IN SUAL(??)
```

D E U X I E M E P A R T I E

U N M O D E L E

P O U R

L E S P A R T I E S O P E R A T I V E S

- G E N E R A T I O N
  P A R "L U B R I C K"

- A P P L I C A T I O N
  A L A C O N C E P T I O N
  D U P R O C E S S E U R "P - C O D E"

SOMMAIRE :

I N T R O D U C T I O N

Le modèle des Parties Opératives, présenté dans [REF. ANCEAU 82], est utilisé dans de nombreux circuits industriels de grande complexité, et il est reconnu comme étant LE BON MODELE, c'est-à-dire celui qui donne la meilleure densité d'intégration, tout en permettant l'implantation d'architectures diverses et performantes:

c'est le principe de la tranche de 1 bit (en anglais BIT-SLICE), répétée n fois pour obtenir une partie opérative travaillant sur des données de n bits.

Par pure convention, on dira qu'on superpose les bits dans le sens VERTICAL, en les répétant, et que des BUS HORIZONTAUX relient les blocs fonctionnels qui sont placés les uns A-DROITE des autres, tous étant de la même hauteur.

Ce principe permet un découpage fonctionnel en "sous-parties opératives", qui sont topologiquement placées les unes à côté des autres et qui peuvent fonctionner:

- soit en parallèlisme synchrone sur leurs bus privés,
- soit en synchronisme lorsqu'elles se transmettent des données, en établissant une communication entre 2 bus privés (par un interrupteur).

I- PRESENTATION DU MODELE

Le modèle topologique est simple, efficace en termes de surface et régularité, et un assembleur de Silicium comme LUBRICK est suffisant pour réaliser une génération automatique d'une telle structure:

- chaque bloc fonctionnel est construit en utilisant l'opérateur de placement EN-HAUT, ou l'opérateur de répétition;
- chaque sous-partie opérative est construite en plaçant les blocs

fonctionnels les uns A-DROITE des autres;

- lorsque toutes les sous-parties opératives ont été assemblées (chacune est contenue dans un rectangle), il suffit de les placer les unes A-DROITE des autres, les cellules contenant les interrupteurs de communication réalisant les connexions.

Remarque:
Dans le cas où les sous-parties opératives n'ont pas la même hauteur (parce qu'elles ne travaillent pas sur le même nombre de bits), elles peuvent être assemblées par l'opérateur "A-DROITE" de LUBRICK, avec un paramètre "déplac" égal à un multiple de la hauteur d'une tranche de 1 bit.

Exemple d'assemblage du microprocesseur PCODE:

La figure 'POAD' est la sous-partie opérative traitant les adresses sur 17 bits,
La figure 'PODA' est la sous-partie opérative traitant les données sur 16 bits,
La hauteur d'une tranche est égale à 54 unités.

Le programme LUBRICK d'assemblage de la partie opérative complète 'POTO' est:

```
var i:pointeur;
begin
 i:=CLOSEFIG(RIGHT(RIGHT(OPENFIG('POTO'),GETFIG('POAD'),1,0)
                  ,GETFIG('PODA'),1,54))
end.
```

I-A- LES ASPECTS TEMPORELS

Une partie opérative est constituée d'opérateurs, qui réalisent des opérations arithmétiques ou logiques, et de BUS qui servent à réaliser les transferts de données entre différents éléments de mémorisation, appelés aussi REGISTRES.

Le mécanisme de transfert associé aux bus est lui moins figé que la solution topologique généralement adoptée: on trouve le plus souvent des bus différentiels à précharge (par exemple dans le MC68000), avec un système de verrouillage rapide dès que la lecture d'un registre "source" est amorcée.
Un tel mécanisme, dépendant de propriétés analogiques critiques, ne peut pas être retenu dans tous les cas.

Le principe de la précharge des bus se justifie d'autant plus que les parties opératives deviennent plus complexes: en effet chaque fil de bus présente une capacité d'autant plus importante qu'il y a d'éléments dans la partie opérative; on définit alors d'un "temps mort" (une phase T1 par exemple), pour porter les 2 fils de chaque bus à un potentiel proche de la tension d'alimentation, en chargeant la capacité constituée par les fils et les drains/sources des transistors qui lui sont connectés.

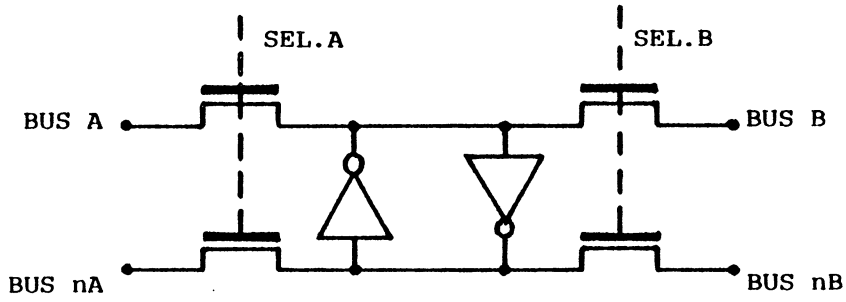
Une sélection d'un registre "source" ou "émetteur" réalise alors une décharge de l'un des fils du bus différentiel.

I-A-1- LE SYSTEME A 4 PHASES SANS VERROUILLAGE DES BUS

On décompose un cycle de transfert sur un bus en au moins 3 phases:
- la phase T1 est une précharge des bus,
- la lecture d'une source s'étend sur les phases T2 et T3,
- l'écriture dans une destination se produit en T3.

Cette solution suppose l'existence de bus différentiels, portés par 2 fils qui ont des valeurs logiques opposées, et des "points de

mémorisation" constitués de 2 inverseurs rebouclés, connectables par 2 interrupteurs sur les 2 fils du bus (voir FIGURE). Ces points de mémorisation sont auto-amplificateurs en ce sens qu'ils sont capables de décharger les bus auxquels ils sont connectés.



Dans ce cas, on définit l'"architecture temporelle" pour les opérateurs connectés sur les bus, comme suit:

- chaque opérateur doit posséder des tampons d'entrée, chargés au début de T3,
- et un tampon de sortie, chargé avec la valeur du résultat à la fin de T1 du cycle suivant;
  ce tampon de sortie peut être à nouveau sélectionné comme source d'un bus au début de la phase T2 du cycle suivant.

Ainsi, on constitue automatiquement un accumulateur avec les 2 tampons d'entrée et celui de sortie d'un opérateur.

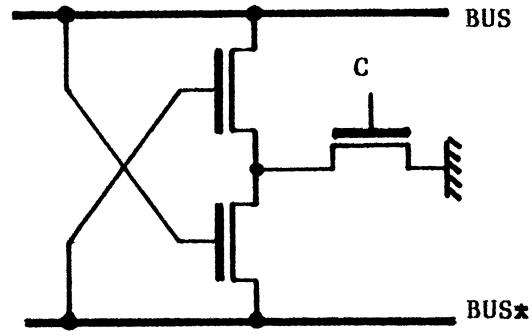
Si le temps de calcul dans l'opérateur le plus lent est supérieur à la durée des 2 phases T3 et T1, on ajoute une phase T4.

La phase T2 peut dans tous les cas être utilisée pour réaliser une précharge dans un mécanisme de retenue dynamique, puisque le résultat précédent a été verrouillé en T1, et que les nouveaux opérandes ne sont chargés qu'en T3.

C'est ce schéma temporel qui est implanté dans le microprocesseur PCODE.

I-A-2- LE SYSTEME A VERROUILLAGE DES BUS ET TAMPON EN SORTIE

La décharge des bus peut être longue si elle est faite par les points de mémorisation seuls, puisque la capacité du bus est importante; elle peut être accélérée par un amplificateur différentiel, réalisé selon le schéma suivant (Ref. photo du MC68000):



La commande C de la figure doit être activée après que l'un des 2 fils BUS ou BUS* ait commencé à descendre vers '0', et le fil qui est resté à '1' accélère la descente de l'autre fil.

Ce mécanisme implique un autre schéma temporel pour le fonctionnement des opérateurs;
on définit:
- une phase T1 de précharge des bus;
- une phase T2 de sélection d'une source sur les bus;
- une phase T3 d'amplification différentielle, à la fin de laquelle les bus sont "verrouillés".
- une phase T4 d'écriture dans un registre "destination";

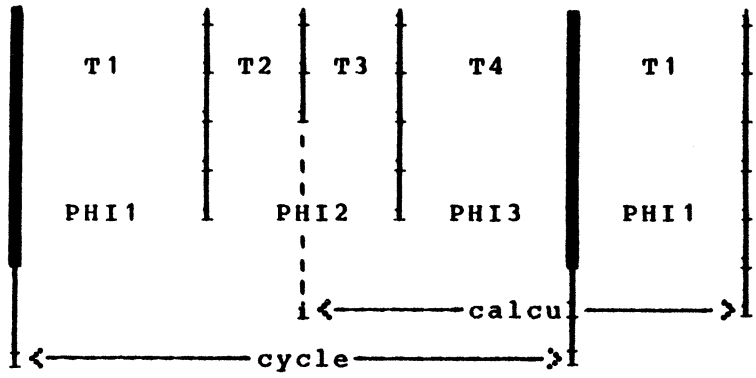
Les bus constituent en eux-mêmes, un élément de mémorisation temporaire, et ils sont considérés, dans le système du MC68000 par exemple, comme les tampons d'entrée des opérateurs. L'existence d'un amplificateur sur chaque bus permet de dessiner des points de mémorisation de taille minimale, dont le rôle se limite à créer une petite différence entre les 2 lignes du bus différentiel.

Sachant que les bus seront de nouveau préchargés au début du cycle suivant, il est nécessaire de définir un tampon de sortie pour les opérateurs, tampon chargé soit à la fin de la phase T4 précédemment définie, soit à la fin d'une phase T5 si la durée totale du cycle est trop courte pour le calcul le plus long.

I-A-3- VERROUILLAGE DES BUS ET TAMPONS EN ENTREE DES OPERATEURS

Ce mécanisme diffère peu du précédent:
on remplace le tampon de sortie chargé en T4 ou T5, par des tampons d'entrée transparents pendant T3 et T4, ce qui permet de terminer un calcul pendant la phase de précharge T1 suivante.

Le découpage du cycle est alors:

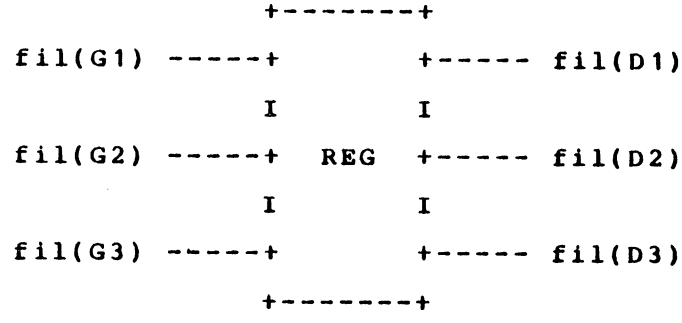


Un tel système, à trois temps principaux (PHI1, PHI2, PHI3), dans lequel le temps PHI2 est découpé en 2 demi-temps (T2 et T3), permet d'utiliser plus des 2 tiers du cycle (et même presque les 5 sixièmes) pour les calculs (T3+PHI3+PHI1), tout en ayant la possibilité de réaliser un système de retenue dynamique (préchargée en T2).

I-A-4- LECTURE SUR 1 FIL, ECRITURE DIFFERENTIELLE

Contrairement aux solutions précédentes pour lesquelles les 2 fils d'un bus servent aussi bien à la lecture qu'à l'écriture, cette autre solution consiste à faire la lecture sur un seul fil, et l'écriture à partir de 2 fils portés à des niveaux logiques opposés.

Un point de mémorisation, qui doit être auto-amplificateur, peut alors être lu sur plusieurs fils, et également écrit à partir de plusieurs couples de fils. Dans l'exemple suivant:



le registre REG peut être "lu" simultanément sur les 6 fils G1, G2, G3, D1, D2 et D3;
il peut être écrit à partir de l'un des 9 couples de fils (Gi, Dj).

Ceci permet la conception d'architectures complexes, dans lesquelles:

- les 6 fils précédents amènent 6 opérandes en entrée de 3 opérateurs, dans une première phase de lecture;
- puis les sorties des 3 opérateurs sont transférées, en différentiel, sur 3 couples de fils, soit par un mécanisme de précharge-décharge, soit par l'intermédiaire d'un amplificateur à sortie "3-états":
  il suffit alors de placer un tampon dynamique en entrée des opérateurs, et on réalise en un cycle complet 3 opérations en parallèle:

  Ri1 op1 Rj1 -> Rk1
  Ri2 op2 Rj2 -> Rk2
  Ri3 op3 Rj3 -> Rk3

Le cycle est constitué de 6 étapes:
1/- précharge des 6 fils,
2/- lecture des 6 opérandes sur les 6 fils,
3/- verrouillage des tampons d'entrée et début des calculs,
4/- précharge des 6 fils,
5/- lecture des 3 résultats de calcul sur les 3 couples de fils,
6/- écriture dans les 3 destinations.

Remarque: ce système impose le passage par un opérateur pour transférer le contenu d'un registre Ri dans un autre registre Rk.

I-A-5- **CONCLUSION**

Quel que soit le principe retenu, les mécanismes élémentaires de transfert d'information sur des bus doivent être choisis et étudiés au début de la phase de conception d'un processeur intégré:
ils définissent à la fois
- la puissance fonctionnelle d'une Partie Opérative,
- sa performance en terme de temps de cycle,
- et sa topologie en terme de hauteur;

En effet, le nombre de fils choisi définit la hauteur de chaque tranche de 1 bit, et le nombre de transferts qui pourront être exécuté en parallèle sur cette structure.

Un jeu de cellules de base peut alors être étudié, dessiné, et les mécanismes de base peuvent être simulés électriquement.

I-B- COMPILATION D'UNE DESCRIPTION DE HAUT-NIVEAU

Les structures de base présentées ci-dessus constituent des modèles généraux, qui n'ont d'intérêt que:
- si on sait les formaliser;
- si on sait les implanter.

On peut alors dire que le modèle d'une sous-partie opérative peut constituer une cible pour le compilateur d'un langage de haut-niveau utilisé pour décrire la fonctionnalité de la partie opérative à réaliser: ce compilateur doit affecter les variables rencontrées dans la description à des registres implantés dans une sous-partie opérative, et autour des opérateurs câblés qui réalisent les opérations décrites.

Cette compilation est en cours d'étude: elle doit permettre de faire apparaître, grâce au parallélisme des instructions présent dans le langage IRENE, le nombre de sous-parties opératives nécessaires, et leur contenu. Si toutes les fonctions rencontrées dans la description sont classiques, le compilateur pourra même générer le texte d'un programme LUBRICK qui, par l'appel de fonctions Lubrick et le placement des blocs fonctionnels les uns à côté des autres, produira le dessin des masques de la Partie Opérative complète.

Les paragraphes suivants présentent quelques fonctions LUBRICK qui, pour un jeu de cellules donné, construisent les blocs fonctionnels dont l'assemblage constitue une sous-partie opérative.
La liste des fonctions présentées n'est pas exhaustive, mais elle peut être enrichie, et le passage à une nouvelle technologie ne nécessite qu'un travail de dessin des cellules de base.

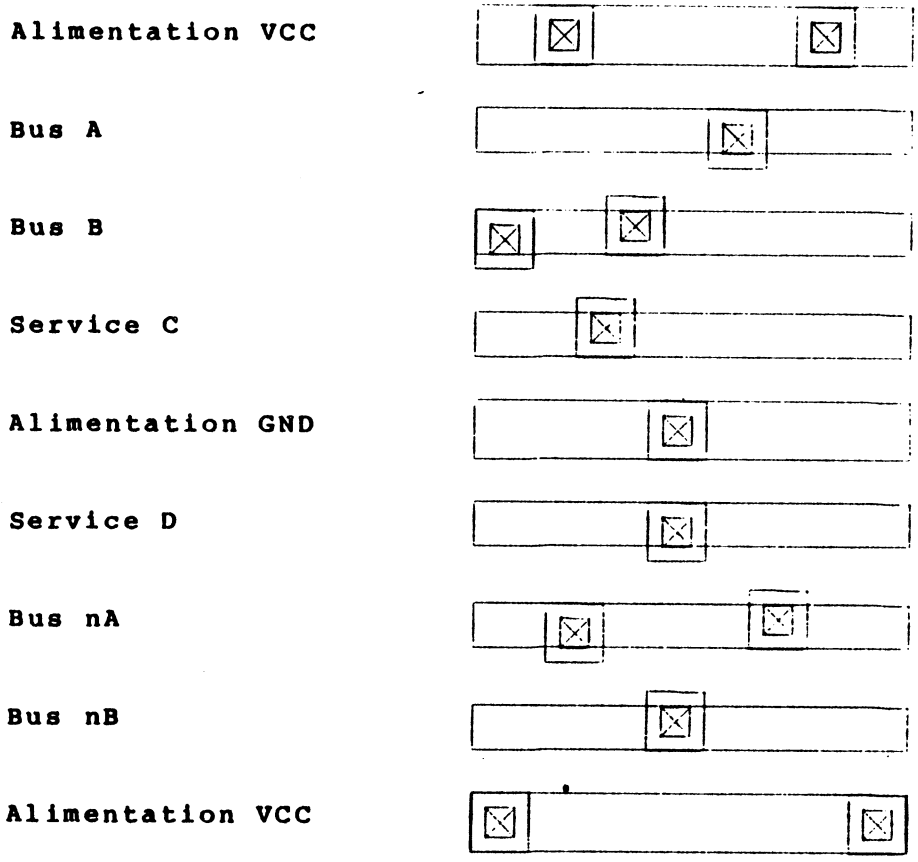
II- GENERATION DE SOUS-PARTIES OPERATIVES

On présente ici, à titre d'exemple, l'approche suivie pour la génération du dessin des opérateurs les plus classiques, en renvoyant à un autre chapitre (présentation du microprocesseur PCODE), et à un autre ouvrage [Ref. GALLAIS] pour les opérateurs spéciaux.

La génération présentée ici est basée sur:

- un principe architectural simple constitué par l'existence de 2 doubles bus différentiels servant à connecter les registres divers et les opérateurs;
- un principe topologique simple qui est celui des tranches de 1 bit de hauteur fixe définie au départ pour une technologie donnée, et pour un nombre de lignes fixe supportant les bus, les lignes de service et les alimentations.

Exemple de la tranche utilisée pour POPY, règles nMOS du CMP:



II-A- ETUDE D'UN OPERATEUR UNIVERSEL (UAL)

L'opérateur "universel" appelé aussi Unité Arithmétique et Logique (UAL) constitue l'organe de calcul d'une sous-partie opérative:

il effectue des calculs (arithmétiques ou logiques) entre 1 ou 2 opérandes qui proviennent de registres internes ou de registres d'Entrée/Sortie.

Appelons x et y les 2 opérandes, et [OPi] l'ensemble des opérations réalisables:
l'UAL calcule un ensemble de fonctions OP1(x,y), OP2(x,y), etc..., et elle reçoit physiquement 3 données:
- les 2 opérandes x et y,
- le code de l'opération OPi à effectuer.

Certains opérateurs, comme celui du B1700 [Ref. Burroughs], calculent en parallèle toutes les valeurs op1(x,y), op2(x,y), etc..., et le code OPi est utilisé uniquement pour sélectionner l'un des résultats: c'est en effet la solution la plus rapide, mais également la plus coûteuse en matériel.

On présentera plutôt une solution économique, qui peut d'ailleurs être d'une vitesse de calcul suffisante si une autre partie de la machine est plus lente (ce qui est le cas pour la partie contrôle, en général).

Cette solution économique consiste à utiliser les bits du code OPi comme commandes d'un circuit combinatoire **unique** capable de calculer **toutes** les fonctions possibles OP1(x,y), OP2(x,y), etc., mais une seule dans chaque cycle.

Il existe un grand nombre de réalisations possibles de cet opérateur UAL (il existe presqu'autant de solutions, ayant une grande similitude, qu'il y a de microprocesseurs sur le marché).

La réalisation présentée ici est issue d'un compromis entre diverses solutions connues: elle a le mérite de bien s'implanter et d'être rapide, avec un nombre de transistors faible, et une grande généralité, et elle peut être adaptée facilement pour le C-MOS.

II-A-1- L'OPERATEUR GENERAL

L'opérateur général est une extension d'un additionneur. Sa généralité est obtenue par 2 commandes C1 et C2:
- C1 invalide la fonction (x NON-OU y),
- C2 invalide la fonction (x ET y).

On peut ainsi calculer un OU EXCLUSIF :

x OUX y = (x ET ny)OU(nx ET y) = (x ET y)OU(nx NON-OU ny)

ainsi que les fonctions ET, NON-ET, OU, NON-OU et NOUX, si on dispose d'un moyen pour inverser la sortie.

Ce moyen va être fourni par un 2-ème étage, qui servira
- soit à combiner la sortie du 1-er étage avec une retenue entrante, pour faire une addition,
- soit avec un '0' ou un '1' pour réaliser ou non une inversion par une porte OU EXCLUSIF (OUX).

II-A-2- **LA RETENUE**

Le problème posé par la retenue est son temps de propagation d'une tranche de 1 bit vers la suivante: les retards s'accumulent dans le cas le plus défavorable, d'une manière quadratique, car c'est une propagation. D'excellents articles étudient ce problème et proposent de très bonnes solutions [REF. VUILLEMIN].

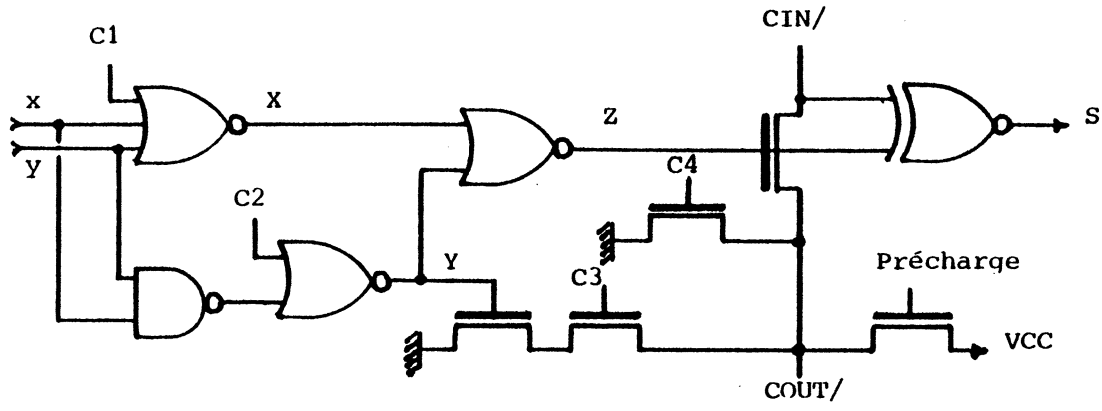
La solution présentée ici est voisine de celle qui est implantée dans le MC68000, avec un calcul anticipé par groupes de 4 bits:
- un groupe de 4 bits génère une retenue si et seulement si la somme de ses entrées vaut 15 et qu'il reçoit une retenue entrante;
- de même pour un groupe de 8 bits, si la somme de ses entrées vaut 255.

  Une simple porte NON-OU distribuée sur plusieurs bits (4, ou 8, ou plus) permet ainsi de prédire si un groupe va fournir une retenue pour le groupe suivant.

  A l'intérieur d'un groupe (de 4 bits par exemple), il faut évidemment propager la retenue bit après bit, et pour ce faire, une technique de précharge peut être employée: travaillant en logique négative, on peut précharger toutes les NON-retenues à '1', et ensuite soit les décharger localement (signal GENERER), soit propager la retenue entrante vers le bit suivant (signal PROPAGER).

II-A-3- REALISATION PROPOSEE

Toutes les fonctions précédentes sont réalisées par le montage suivant:



dans lequel: X= NON-OU(C1,x,y);

Y= NON-OU(C2,NON-ET(x,y));

Z= NON-OU(X,Y);

S= NOUX(Z,NON-retenue) = OUX(Z,retenue);

si C2=0, alors Y= ET(x,y) qui construit GENERER;
si C1=C2=0, alors Z= OUX(x,y) qui construit PROPAGER.

Les fonctions de base sont données par le tableau:

C1	C2	Z
0	0	OUX(x,y)
0	1	OU(x,y)
1	0	NON-ET(x,y)
1	1	'1'

ROLE DES COMMANDES C3 et C4:

1/- La commande C3=0 empêche la génération des retenues, donc

S= OUX(Z,'1') = nZ

2/- La commande C4=1 force la génération des retenues, donc

S= OUX(Z,'0') = Z

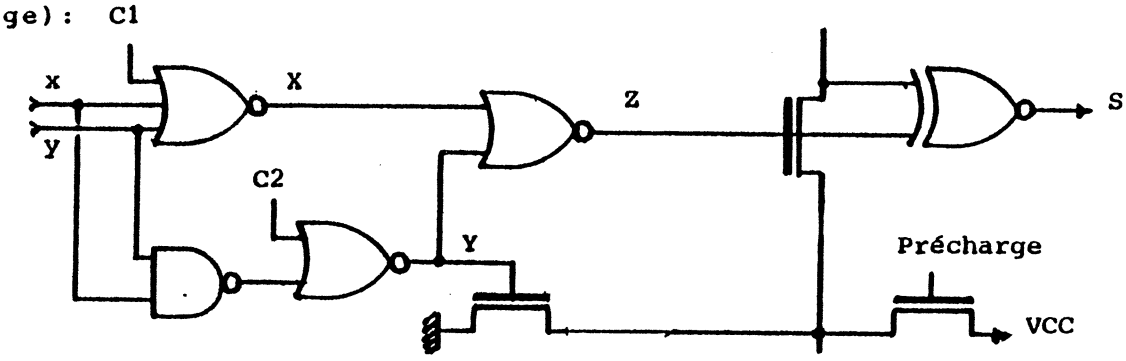
3/- Les commandes (C1=1) et (C2=1), combinées avec (C4=0), permettent l'addition d'une retenue à la valeur FFFF: selon la retenue entrante dans le bit No 0, on obtient soit 0000 soit FFFF (pour 16 bits).

On obtient le tableau complet:

C1	C2	C3	C4	Sortie S
0	0	1	0	ADD(x,y,retenue)
0	0	0	0	NOUX(x,y)
0	0	x	1	OUX(x,y)
0	1	0	0	NON-OU(x,y)
0	1	x	1	OU(x,y)
1	0	0	0	NON-ET(x,y)
1	0	x	1	ET(x,y)
1	1	1	0	ADD(-1,retenue)
1	1	0	0	0
1	1	x	1	-1

II-A-4- VERSIONS SIMPLIFIEES DE L'UAL

1/- Les commandes C3 et C4 ont pour seul rôle de permettre les calculs des fonctions indépendantes de la retenue (OUX(x,y)) ou les fonctions complémentaires. Si aucune de ces fonctions n'est utilisée, on simplifie, (en forçant (C2=1) pendant la phase de précharge): C1



2/- Si on ne fait que des additions, on peut se passer de C1 et de C2, mais il faut alors conserver C3 (en forçant (C3=0) pendant la phase de précharge), et on peut alors proposer un montage plus rapide pour cet additionneur:

Z= NON-OU(ET(x,y), ET(nx,ny));

G= NON-OU(nx,ny);

3/- un incrémenteur est un additionneur dont l'une des entrées est nulle: on simplifie et, si y=0, on obtient Z=x et G=0;
il ne reste donc que le 2-ème étage.

4/- si on ne fait pas d'addition, on simplifie le 2-ème étage: il n'y a plus de calcul de de retenue, et on peut simplement ajouter un calcul de nZ pour choisir entre Z et NZ, pour obtenir soit une fonction soit son complément.

II-A-5- GENERATION par LUBRICK

On définit une fonction PASCAL-LUBRICK avec un paramètre indiquant la famille des fonctions à réaliser; selon la valeur de ce paramètre, la fonction Lubrick choisit une cellule pour le 1-er

étage et un assemblage du 2-ème étage:

- le 1er étage peut être la cellule la plus complète ("uali"), ou un demi-additionneur ("addi"), ou un demi-incrémenteur quasi-vide ("inci"), ou la cellule logique ("logi").

- le 2ème étage peut être un assemblage de cellules de calcul de retenue anticipée, ou un simple multiplexeur.

La retenue anticipée est construite à l'aide de 6 cellules différentes:

- "cai00" pour la retenue entrante (bit de poids faible i=0);
- "cai0" pour les bits de poids 4i (i>0),
- "cai1" pour les bits de poids (4i+1) et (4i+2),
- "cai3" pour les bits de poids (4i+3),
- "canm1" pour l'avant-dernier bit,
- "can" pour le dernier bit (poids fort).

Les 2 dernières cellules ne sont nécessaires que si l'on désire un calcul des indicateurs (ou FLAGS) de débordement (OVERFLOW) et de retenue sortante (CARRY-OUT): un paramètre supplémentaire indique ce cas. La programmation de la fonction génératrice LUBRICK de nom "FUAL" est donnée en annexe.

B- ETUDE DES ENTREES DE L'OPERATEUR

L'opérateur réalise le calcul de op1(x,y), op2(x,y), etc...

Ses deux "opérandes" x et y, appelés entrées de l'opérateur, peuvent être sélectionnés parmi plusieurs "sources" qui peuvent être:

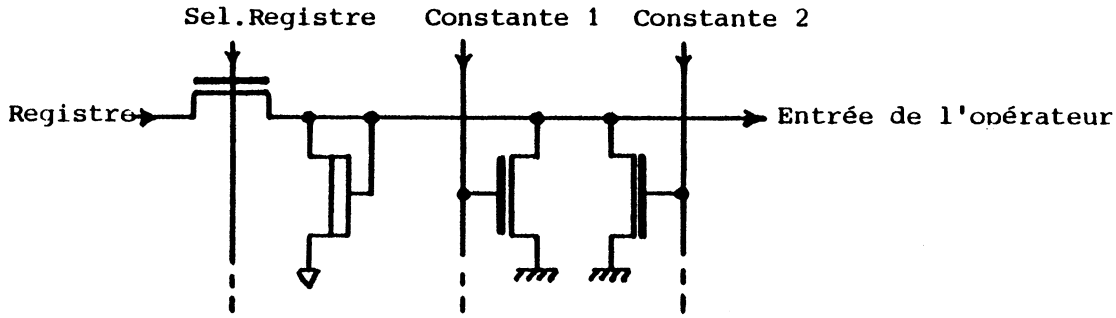
- des registres (tampon d'entrée) ou leur complément,
- des constantes,
- des bus ou leur complément, etc...

On étudie d'abord la génération de constantes.

II-B-1- GENERATION DE CONSTANTES

II-B-1-a- Génération directe:

Une constante est une valeur de n bits, si n est le nombre de bits de la PO. En n-MOS, on met à profit le fait que la valeur '0' est prioritaire sur la valeur '1', pour fabriquer plusieurs constantes sous la forme d'un NON-OU distribué, comme dans un PLA: un seul transistor de charge suffit pour toutes les constantes;
cette priorité peut également être utilisée pour réaliser un masquage de certains bits d'un registre contenant un opérande, selon le schéma suivant:



La génération d'un bloc de constantes par une fonction Lubrick nécessite les 5 cellules suivantes:

- "début": cellule contenant la sélection d'un registre opérande et un transistor de charge,
- "gau0" et "gau1" pour les valeurs '0' et '1' situées "à gauche",
- "dro0" et "dro1" pour les valeurs '0' et '1' situées "à droite".

On déclare un tableau TP de pointeurs à 2 dimensions:

```
var TP : array[0..1,0..1] of pointeur;
```

qu'on initialise par:

```
TP[0,0] := GETFIG('gau0'); TP[0,1] := GETFIG('gau1');
TP[1,0] := GETFIG('dro0'); TP[1,1] := GETFIG('dro1');
```

On programme l'assemblage d'une colonne par:

```
function COLONNE ( i , v : integer ) : pointeur;
  (* i=0 pour gauche, i=1 pour droite, v=valeur de la constante *)
var p:pointeur; j,x:integer; nom:alfa6;
begin
       fabnom('col',nom); (* fabrique un nom de cellule *)
       p := OPENFIG(nom);
       x:=v;    (* on décompose v en base 2 *)
       for j:=1 to nbits do
       begin
         EN-HAUT ( p , TP [i,x mod 2] , 1 , 0 );
         x := x div 2
       end;
       COLONNE := closefig(p)
end;
```

On peut alors compléter la fonction principale, qui reçoit en paramètre un tableau de valeurs TVAL:

```
function CONSTANTE ( nbit,nconst:integer;
        var TVAL : array[0..nmax] of integer ) : pointeur;
var p:pointeur; i:integer; nom:alfa6;
begin
  fabnom('cons',nom);
  p:=A-DROITE (OPENFIG(nom), REPY(GETFIG('debut'),nbit), 1, 0);
  for i:=1 to nbconst do
    A-DROITE ( p, COLONNE(i mod 2, TVAL[i]), 1, 0);
  CONSTANTE := CLOSEFIG(p)
end;
```

OPTIMISATION:

on optimise le nombre d'appels de la fonction EN-HAUT dans colonne en comptant le nombre de '0' ou de '1' consécutifs dans la valeur binaire de la constante:

```
(* j parcourt les poids des bits
   n compte le nombre de bits égaux consécutifs *)
j:=0;
while j<nbits do
begin  j:=j+1; n:=1;
       cas:= x mod 2; (* le 1-er d'une série *)
       x := x div 2;
       while (x mod 2=cas) and (j < nbits) do
       begin n:=n+1; j:=j+1; x:=x div 2 end;
       EN-HAUT ( p, TP [i,cas], n, 0)
end;
```

II-B-1-b- Génération optimisée:

La 1-ère optimisation triviale concerne la constante -1 (ou FFFF...) qui est automatiquement disponible, lorsqu'aucun opérande n'est sélectionné.

La seconde optimisation n'est pas toujours possible: elle s'appuie sur le principe que i commandes permettent de fabriquer 2^i constantes, parmi lesquelles se trouve la constante -1:
2 commandes a et b permettent de fabriquer les valeurs -1, x1, x2 et x3, x3 étant égal au ET bit-à-bit de x1 et x2.
On peut par exemple obtenir avec 2 bits de commande les valeurs (-1, +1, +2 et 0).

et avec 3 bits on peut fabriquer les valeurs:
(-1, -2, -3, -4, +3, +2, +1 et 0).

Le problème d'optimisation se pose à l'envers:
"étant données 2^n valeurs quelconques, est-il possible de les fabriquer à l'aide de seulement n fils de commande ?"

Un algorithme possible est le suivant:
on classe les valeurs à fabriquer par ordre décroissant du nombre de bits à '1' (dans leur représentation binaire), en prenant au

moins une valeur de chaque signe, jusqu'à obtenir n valeurs.

Proposition:
si une couverture des 2^n valeurs existe, elle est donnée par les n valeurs classées en tête, à cause de la priorité du '0'.

Si la couverture est incomplète, on cherche à ajouter une valeur qui n'est pas déjà couverte, afin d'obtenir des valeurs combinées supplémentaires, mais on n'est pas sûr d'obtenir la couverture complète.

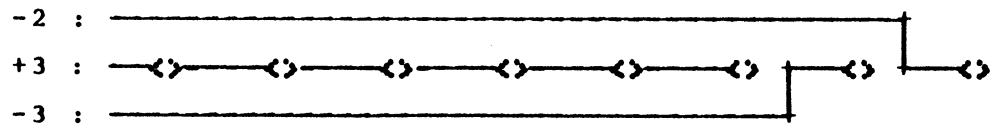
Exemples:

valeurs	couverture
(-3, -2, +3)	(-4, -3, -2, -1, 0, +1, +2, +3)
(-11,-10,+11)	(-12,-11,-10,-1, 0, +1, +2)

II-B-1-c/ **FONCTION DE GENERATION en LUBRICK:**

Lorsqu'on a trouvé les valeurs qui permettent la couverture, on génère des colonnes, avec une cellule de mise à '0' dans les positions correspondant aux bits valant '0': le dessin peut même dans certains cas être optimisé en surface en plaçant d'abord les constantes positives par ordre croissant, puis les constantes négatives par ordre croissant:

Exemple: soit les constantes (+3, -3, -2 et -1)



Dans cet exemple, la largeur utilisée est égale à celle d'une cellule de sélection plus deux lignes de commande placées de part et d'autre.

II-B-2- SELECTION DES OPERANDES

Les opérandes d'un opérateur ne sont pas que des constantes, ils peuvent aussi être des variables, c'est-à-dire le contenu de tampons d'entrée de l'opérateur.
Les alternatives possibles sont limitées dans 2 directions:
- la fonctionnalité,
- la topologie.

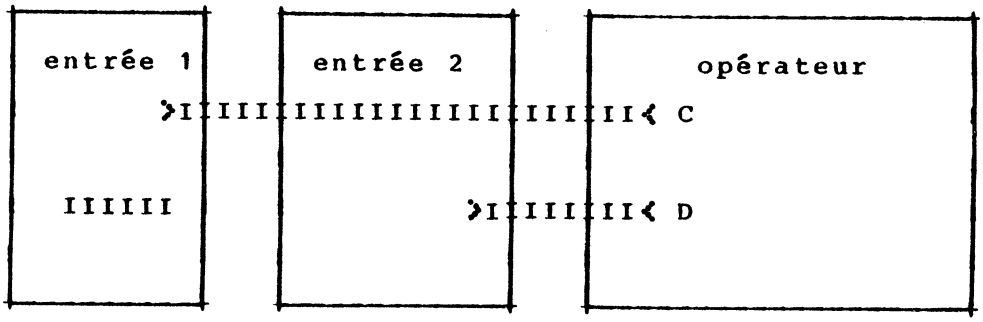
II-B-2-a- Alternatives fonctionnelles:

On peut définir un registre "tampon" d'entrée, qui peut être chargé à partir de plusieurs bus, en direct ou en complément; on a dans ce cas besoin d'un multiplexeur à l'entrée du registre tampon. L'autre alternative consiste à charger ce registre à partir d'une seule source (bus) et à placer un multiplexeur sur ses sorties directe ou complémentée.
Dans le dernier cas, on pourra, à plusieurs cycles d'intervalle, utiliser soit la valeur du registre, soit son complément, sans avoir à le modifier: on a donc un tampon d'entrée à caractère permanent, ce qui n'est pas le cas de la 1-ère alternative:
pour faire (R1+R2) au cycle i, puis (R3-R2) au cycle (i+1), on doit charger R2 puis non-R2 dans le même tampon d'entrée, au cours de 2 cycles consécutifs.

II-B-2-b- Limites topologiques:

Ces limites sont liées à l'allocation des lignes de métal qui sont des ressources critiques: le dessin d'un multiplexeur requiert l'utilisation de 2 lignes de métal (si l'on ne dispose pas de transistors fortement déplétés comme pour le MC68000, qui permettent des croisements POLY-DIFFUSION).
Dans le cas de la topologie présentée en introduction pour la tranche de 1 bit, 4 lignes de métal servent à véhiculer 2 bus différentiels, et il ne reste que 2 lignes de métal libres, qui sont appelées les 2 lignes "de service" C et D. Si l'on considère

les 2 entrées de l'opérateur, une seule de ces 2 lignes est libre pour l'entrée la plus proche de l'opérateur:



CONSEQUENCE: on ne peut réaliser un multiplexage que sur l'une des 2 entrées, que l'on placera la plus à gauche (dans le cas où la topologie de la tranche ne permet que 2 lignes de service).

RESTRICTION FONCTIONNELLE:
la soustraction (A-B) est réalisé par une addition (A + non-B +1), ce qui implique un multiplexage entre B et non-B; on ne pourra donc pas réaliser à la fois (A-B) et (B-A), parce qu'une seule des entrées peut être multiplexée, dans le cas des limites topologiques présentées.

CAS PARTICULIER: si l'un des bus ne sert qu'à acheminer une valeur vers un registre tampon, ses lignes peuvent devenir libres, et on peut les utiliser pour fabriquer 2 multiplexeurs, et même pour implanter l'opérateur et les cellules de retenue d'une manière plus dense. MAIS ce cas particulier impose une implantation de l'opérateur dans un coin de la partie opérative, ce qui ne satisfait pas à la généralité de l'approche proposée.

II-B-2-c- FONCTION LUBRICK

La fonction Lubrick implémentée permet de choisir la sélection des 2 entrées à partir des 2 bus:

- l'entrée acheminée vers l'opérateur par la ligne de service C est le 1er opérande, chargé à partir du bus A: elle peut être complémentée ou non;

- l'autre entrée, acheminée par la ligne de service D est le 2ème opérande, chargé à partir du bus B: elle peut être complémentée ou non;

Si les 2 entrées sont à complémenter, la fonction envoie un message d'erreur; sinon, elle place d'abord l'entrée à complémenter, puis l'autre, la connexion avec l'opérateur se faisant automatiquement par les lignes C et D (par des connecteurs LUBRICK).

La fonction LUBRICK utilise le registre standard "PMEM", chargé à gauche à partir de l'un des bus A ou B, et une cellule qui réalise un multiplexage avec une sortie soit sur C soit sur D:

Programmation de la fonction ENTREES:

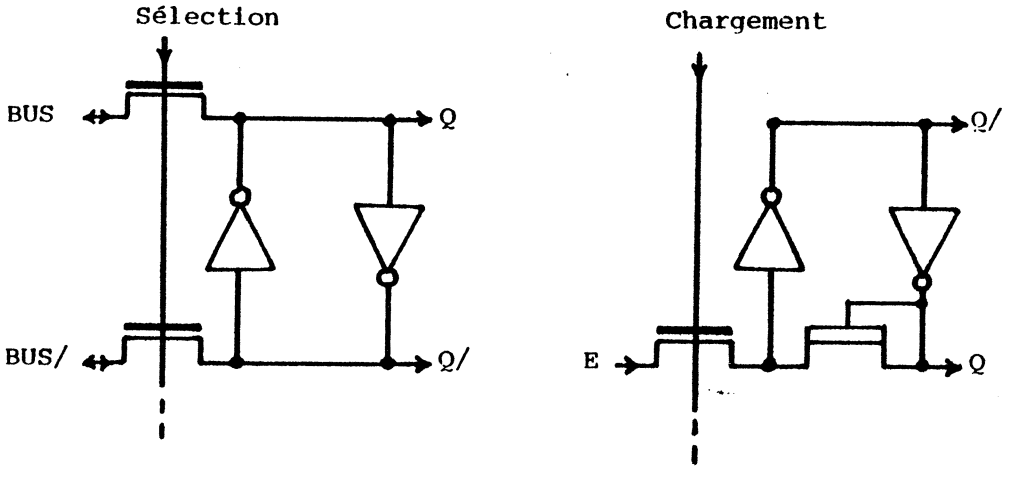
```
function ENTREES ( compl1,compl2: boolean ) : pointeur;
var p,preg,pe1,pe2:pointeur; nom:alfa6;

  function entree1 : pointeur;
  var p,pr:pointeur;
  begin fabnom('ent',nom);
    p := OPENFIG(nom);
     pr:=CLOSEFIG(A-DROITE(A-DROITE(OPENFIG('rega'),
                                    GETFIG('sag'),1,0),
                           preg,1,0));
     p:=A-DROITE(p,pr,1,0);
     (* ajout de connecteurs *)
     (* ajoutconn('e',off,pos,long,niv,type); *)
     (* teste si complément demandé *)
     if compl1 then p:=A-DROITE(p,GETFIG('muxc'),1,0);
    entree1:=CLOSEFIG(p)
  end;

  function entree2 : pointeur;
  (* idem entree1 en échangeant a et b, c et d *)
```

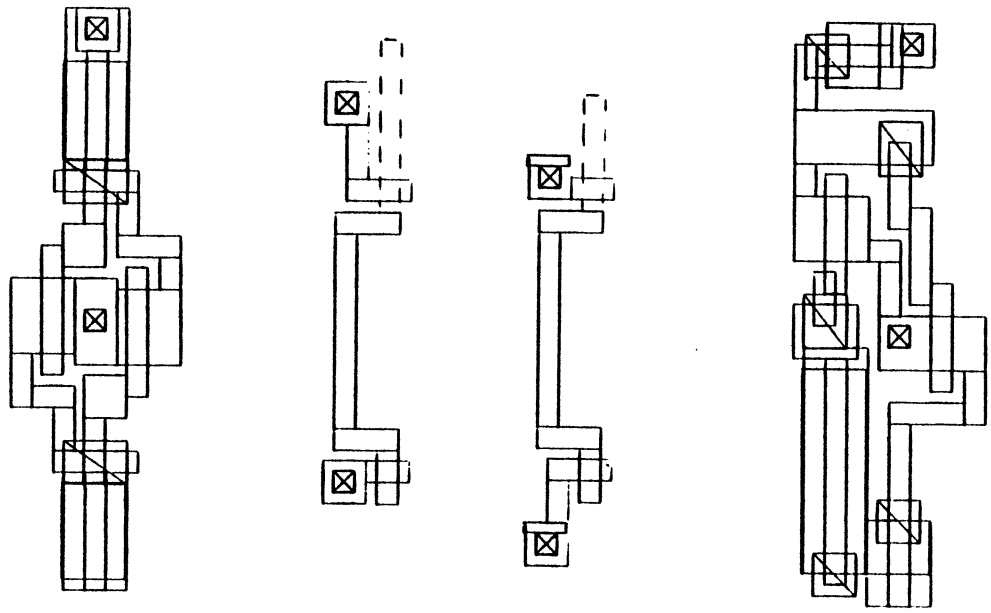
```
begin
  if (compl1 and compl2) then erreur('IMPOSSIBILITE TOPOLOGIQUE');
  fabnom('entr',nom);
  p:=OPENFIG(nom);
  preg := GETFIG('pmem'); (* point mémoire double entrée *)
  pe1 := repy(entree1,nbit);
  pe2 := repy(entree2,nbit);
  if compl1 then p:=A-DROITE(A-DROITE(p,pe1,1,0),pe2,1,0)
            else p:=A-DROITE(A-DROITE(p,pe2,1,0),pe1,1,0);
  entrees:=CLOSEFIG(p)
end;
```

II-C- **GENERATION DES REGISTRES**

Les registres apparaissent en plusieurs endroits d'une partie opérative. Ils peuvent être en lecture et/ou en écriture sur l'un des 2 bus, ou sur les 2, et ils peuvent également être chargés à partir des 2 lignes C et D qui peuvent dans certains cas constituer un bus différentiel.

On distingue:
- le registre standard "pmem" chargé en différentiel,
- le registre "regre" chargé à partir d'un seul fil.





CELLULES: pmem sag sbg regre

Pour chacun de ces registres, il est nécessaire de prévoir tous les cas de connexions, sous la forme de cellules qui se placeront soit à gauche soit à droite de la cellule-registre.
On a défini pour "pmem":

- "sag" = sélection sur le bus A,
- "sbg" = sélection sur le bus B,
- "scdg" = sélection sur les lignes C et D.

et on peut alors programmer une fonction LUBRICK de génération d'un registre, en définissant deux sélections possibles à droite et/ou à gauche selon le codage suivant:

0 = pas de sélection,
1 = sélection sur le bus A,
2 = sélection sur le bus B,
3 = sélection sur les lignes C et D.

```
function REGISTRE ( nbits, selg, seld : integer ) : pointeur;
var nom : alfa6;
     function UNBIT : pointeur;
     var i , j : pointeur;
     begin
         i:=OPENFIG('unbit');
           if (selg > 0) then
           begin
              case selg of
               1: j:=GETFIG('sag');
               2: j:=GETFIG('sbg');
               3: j:=GETFIG('scdg')
              end;
              i:= A-DROITE ( i, j, 1, 0);
           end;
         i:= A-DROITE ( i, GETFIG('pmem'), 1, 0);
           if (seld > 0) then
           begin
              case seld of
               1: j:=GETFIG('sag');
               2: j:=GETFIG('sbg');
               3: j:=GETFIG('scdg')
              end;
              i:= A-DROITE ( i, SYMX(j), 1, 0)
           end;
         i:=CLOSEFIG(i)
     end;

begin (* registre *)

  fabnom('reg',nom);

  REGISTRE := CLOSEFIG( EN-HAUT( OPENFIG(nom), UNBIT, nbits, 0));

end;
```

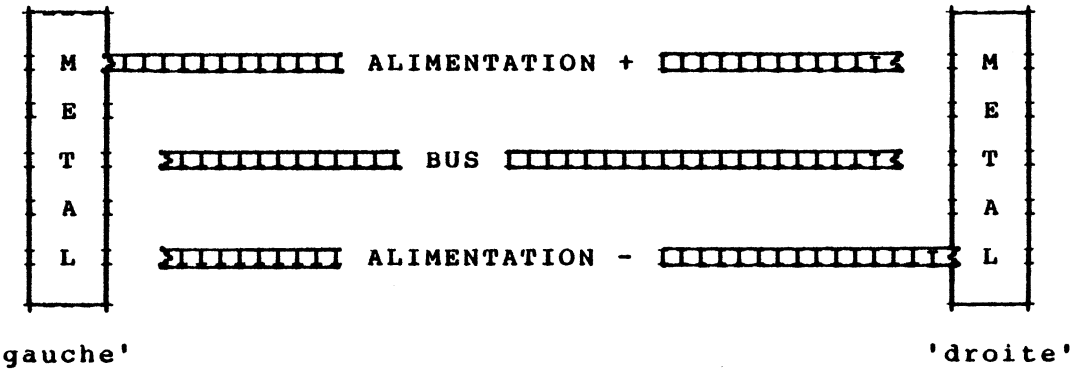
De la même manière, on programme:
- une fonction "registre chargé à partir d'un seul fil",
- et une fonction "bloc de registres" à double accès (voir la fonction FRAM en annexe).

II-D- GENERATION AUTOMATIQUE DES BUS ET ALIMENTATIONS

Toutes les cellules assemblées pour construire une partie opérative ont été conçues et dessinées en fonction de la définition de lignes de métal qui, passant au-dessus de chaque tranche de 1 bit, à des positions fixes, portent les bus et les alimentations.

L'utilisation de cette approche, et l'outil LUBRICK, simplifient la génération des rectangles qui matérialisent ces bus et ces alimentations: il suffit en effet de définir
- une cellule 'gauche' contenant des connecteurs orientés vers l'EST,
- une cellule 'droite' contenant des connecteurs orientés vers l'OUEST

et schématisées comme suit:



La génération automatique de tous les fils d'alimentation et de tous les fils des bus, et la réalisation du "PEIGNE" des alimentations sont faites par le programme LUBRICK suivant:

```
var i:pointeur;
begin
  i := OPENFIG('POALIM');
  i := A-DROITE( i, REPY( GETFIG('gauche'), nbits), 1, 0);
  i := A-DROITE( i, GETFIG('PO'), 1, 0);
  i := A-DROITE( i, REPY( GETFIG('droite'), nbits), 1, 0);
  i := CLOSEFIG(i);
end;
```

On trouvera en annexe des exemples de fonctions génératrices LUBRICK qui montrent la généralité de cette approche, et la puissance de l'outil, pourtant très simple, qu'est l'assembleur de silicium LUBRICK.

CONCEPTION D'UN PROCESSEUR P-CODE

INTRODUCTION

Ce chapitre présente un exemple de conception d'un processeur intégré: il s'agit d'un processeur, ou machine informatique, qui exécute un langage appelé le P-CODE, qui a été introduit en 1973 par l'équipe qui a défini le langage PASCAL (REF. N.WIRTH et K.JENSEN).

Ce langage sert de "cible", ou de "langage intermédiaire", pour les compilateurs du langage PASCAL; de part sa forme post-fixée, (ou notation Polonaise) il facilite l'écriture des fonctions de génération de code dans les compilateurs.

Il présente aussi l'avantage de pouvoir être:
- soit interprété par un programme écrit dans le langage-machine des microprocesseurs (APPLE II, Intel, Zilog...);
- soit interprété par un microprogramme, et il devient alors le langage-machine d'une machine P-CODE, ce qui est le cas du WD 9000 de Western Digital (Pascal MicroEngine et Pascaline);
- soit compilé à son tour, par un 2-ème compilateur, et c'est le cas des systèmes Motorola 68000, entre autres.

On citera au passage les études menées sur les langages intermédiaires adaptés à PASCAL, présentées dans [REF. FORTIER74, REF. SCH77, et REF. COREEN80], qui ont conduit à la définition du langage I-PASCAL exécuté par la machine PASC-HLL [REF. SCH77 et REF. BAILLE83].

Cette présentation ne discute pas la valeur du P-CODE, elle se contente de montrer la démarche de conception d'un processeur intégré; on dira cependant qu'il a été choisi de fabriquer un circuit compatible avec le WD 9000 afin de pouvoir le tester par substitution.

I- PRESENTATION DE LA MACHINE "VIRTUELLE" P-CODE

La machine P-CODE manipule des mots de 16 bits; elle peut en adresser jusqu'à 64K (mots) soit 128K Octets, avec une adresse de 16 bits.
La mémoire est gérée en PILE. Cette PILE contient aussi bien des valeurs de variables, des constantes, des adresses et les instructions du programme qui sont cadrées sur des Octets. La PILE, en son sommet, sert également au stockage des résultats intermédiaires au cours de l'évaluation des expressions, elle sert à passer les paramètres d'appel des procédures et à récupérer le résultat des fonctions.

Aucun registre n'est explicitement adressable par une instruction, bien qu'il en existe 5 qui sont implicitement lus ou modifiés par certaines instructions.

I-A- LES REGISTRES

Ce sont uniquement des registres d'adresse qui pointent sur la mémoire:
- SP = pointeur sur le sommet de la pile (Stack Pointer),
- BL = Base Locale pointant sur le descripteur de la procédure courante,
- BG = Base Globale pointant sur le descripteur de la procédure principale,
- BS = Base du Segment courant, i.e. celui qui contient la procédure courante,
- CO = Compteur Ordinal qui peut être relatif à BS ou non, et qui pointe sur l'octet contenant l'instruction en cours ou la suivante.

I-B- L'ADRESSAGE DES VARIABLES

- Les variables Locales sont adressées par rapport à BL,
- Les variables Globales par rapport à BG,

- Les variables Intermédiaires sont adressées après qu'on ait parcouru N fois le chaînage statique à partir du descripteur pointé par BL. Il est à noter que, pour une variable donnée, il faut parcourir plus ou moins de chaînages selon la profondeur de la procédure dans laquelle on la référence. Ainsi, le programme principal étant considéré comme une procédure-fille du "Système", les variables du "Système" sont référencées comme des variables intermédiaires, et leur accès est d'autant plus long qu'on se trouve à un niveau lexicographique élevé.

I-C- LE DESCRIPTEUR DE PROCEDURE

Ce descripteur est souvent appelé Marque de Pile (MSCW = Mark Stack Control Word en Anglais). Il se compose de 4 mots consécutifs contenant:

- le LIEN STATIQUE, qui permet de définir par des chaînages l'arbre statique du programme PASCAL;
- le LIEN DYNAMIQUE, qui mémorise l'historique dynamique des appels de procédures;
- l'adresse de RETOUR, relative à la base du Segment Appelant,
- le Numéro du Segment Appelant, s'il est différent du Segment courant, sinon Zéro.

Ce descripteur est fabriqué par les instructions d'appel de procédure;
le lien statique est lu pour accéder aux variables intermédiaires;
le lien dynamique et l'adresse de retour sont exploités par l'instruction de retour de procédure.

Le registre BG a un rôle particulier: c'est la base du programme principal de l'utilisateur qui est en fait la procédure locale lorsque le "Système" lui passe le contrôle: un programme principal commence donc toujours par une séquence d'instructions qui transfère la valeur de BL dans BG. C'est la séquence P-CODE WD9000:

"LDC(6); LSL(0); SPR"

du Pascal-Microengine, qui

1/- met la constante 6 sur la pile (c'est le numéro du registre

physique qui contient BG),

2/- puis le lien statique après avoir parcouru ZERO chaînage, c.a.d. la valeur de BL,

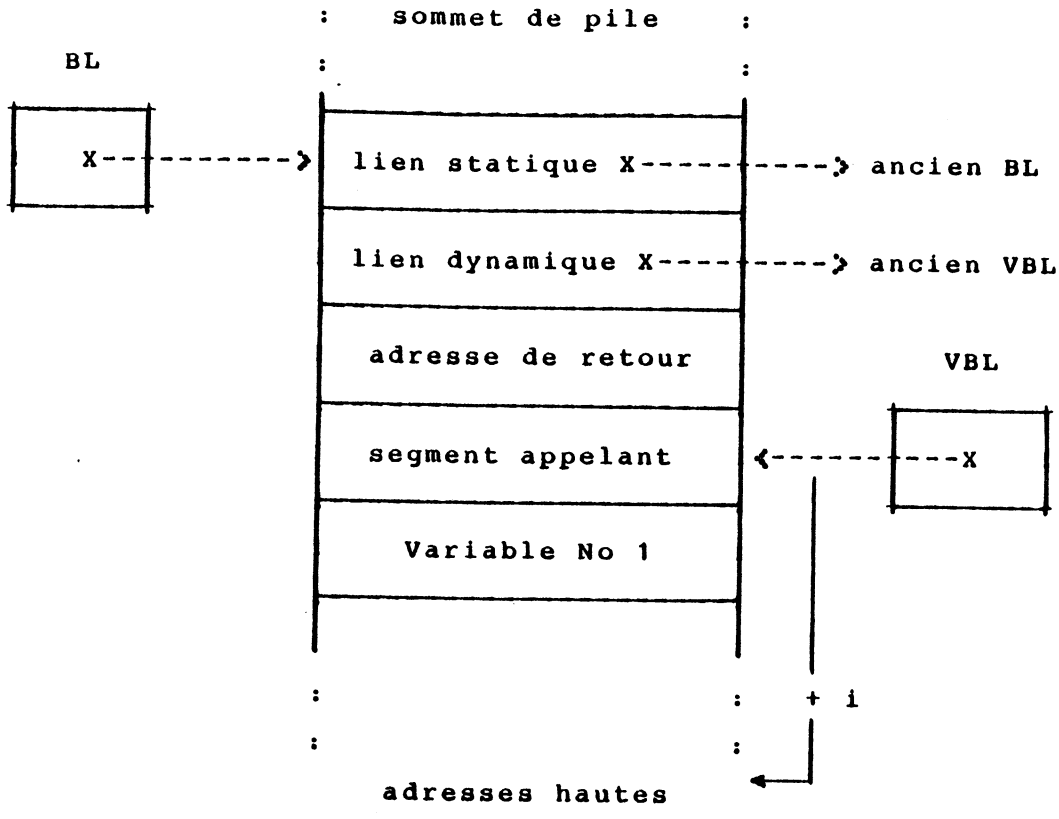
3/- puis charge le Sommet de pile (BL) dans le registre dont le numéro est touvé en sous-sommet (6), c.a.d. le registre BG . (ceci est une interprétation de l'auteur, faute de documentation).

II- LES CHOIX D'IMPLEMENTATION

II-A- LE CALCUL DE L'ADRESSE DES VARIABLES

Dans un état donné, il existe une procédure Globale, et une procédure Locale.
Les registres BL et BG pointent sur le 1-er mot de la marque de pile relative à l'une des 2 procedures (Locale ou Globale resp.). La variable numéro i d'une procédure se trouve à l'adresse (BASE+i+3). On peut donc:

- soit faire pointer BL et BG 3 mots plus loin, pour faire le calcul (BASE+i), mais on ralentit l'accès aux variables intermédiaires, qui est une opération qui peut être fréquente, surtout pour les Entrées/Sorties;
- soit créer deux registres internes appelés VBL et VBG qui contiennent les valeurs (BL+3) et (BG+3) respectivement; le registre VBL doit être mis à jour à chaque appel et à chaque retour de procédure, ce qui ne pose pas de problème car on dispose de la nouvelle valeur quand on construit la marque, et parce qu'on peut stocker l'ancienne valeur de VBL dans le LIEN DYNAMIQUE: telle est la solution retenue.



II-B- L'EVALUATION DES EXPRESSIONS

Le PCODE est un langage d'instructions de type post-fixé, appelé aussi "notation polonaise": l'exécution des instructions est réalisée sur une Pile (dernier entré- premier à sortir).
Plusieurs solutions ont été expérimentées pour accélérer l'accès aux opérandes situés au sommet de la pile:

- l'ordinateur HP3000 [REP. Hewlett Packard], utilise une mémoire de 4 mots, sur laquelle on déplace circulairement deux pointeurs, et qui constitue un "cache";

- le microprocesseur AMD9511 (Number Cruncher) [REF. Advanced Micro Device et REF. REGINE79], dispose d'une mémoire RAM interne, de 16 mots, et d'un triplet (POINTEUR- INCREMENTEUR-DECREMENTEUR) qui permet de disposer en permanence de 2 adresses (sur le sommet et le sous-sommet), et également de faire une mise à jour

automatique
* en cas de PUSH (POINTEUR := INCREMENTEUR),
* et en cas de POP (POINTEUR := DECREMENTEUR).
(Ce mécanisme n'a pas été publié, mais il a été découvert par l'auteur sur la photo du circuit intégré avec l'aide de E.PRESSON).

- l'ordinateur PASC-HLL, défini par l'auteur dans une précédente thèse [REF. SCH-77], remplace la pile par une file d'attente, ce qui permet une organisation pipe-line, avec un parallélisme entre le rangement de nouveaux opérandes au sommet de la pile (dans la file d'attente) et l'extraction des opérandes pour l'opération en cours.
  Deux des processeurs internes à l'ordinateur PASC-HLL (PINS et POP) possèdent cependant une pile interne, de 16 mots de 32 bits, qui constituent un "cache" sur deux piles situées en Mémoire Centrale: leur gestion est réalisée à l'aide de deux compteurs qui gèrent l'un l'adresse du sommet de pile interne, l'autre son remplissage: les débordements de ce dernier (passage de 15 à 0, ou de 0 à -1) indiquent si la pile interne est vide, ou si elle est pleine. On trouvera une description détaillée de cette implantation dans [REF. BAILLE-83].

Pour la présente intégration d'un processeur PCODE, aucun des mécanismes précédents n'est implanté; cependant, deux registres VSP et VSSP ont été créés, qui vont contenir au cours de l'évaluation des expressions, les valeurs des sommet et sous-sommet de pile: en effet, leur utilisation permet globalement un gain sur le nombre des accès à la Mémoire Centrale.

- un opérateur diadique effectue "VSP := VSSP op VSP;", puis range le résultat en mémoire (de telle sorte que la pile située en mémoire soit toujours à jour), et enfin recharge VSSP:
  on fait donc une écriture et une lecture seulement, au lieu de 2 lectures suivies d'une écriture;
- une instruction d'affectation fait seulement une écriture en

mémoire, à l'adresse contenue dans VSSP, du résultat contenu dans VSP;
- une instruction d'accès fait par contre un transfert de VSP dans VSSP, avant de lire une valeur qui est chargée dans VSP.

Ce mécanisme très simple, inventé par un Capitaine de l'armée française,
- ne coûte que 2 registres internes (VSP et VSSP);
- il ne nécessite aucune gestion particulière,
- il diminue le nombre des accès à la mémoire et augmente ainsi les performances
- (il pourrait être breveté comme l'a été le principe de PASC-HLL [REF. ANVAR-75]).

II-C- L'ACCES AUX INSTRUCTIONS ET AUX OPERANDES

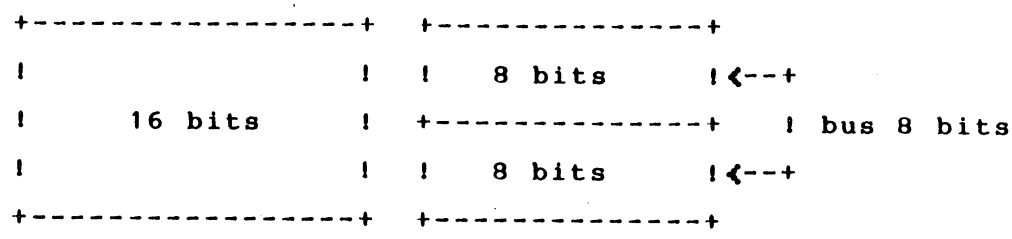
Les instructions du PCODE sont codées sur un nombre variable d'octets (au moins 1), et leurs opérandes sont de longueur variable: en particulier un opérande de type BIG occupe:
- soit 1 octet si sa valeur est inférieure à 128;
- soit 2 octets sinon.

De plus, la mémoire est organisée en mots de 16 bits: il faut donc gérer l'accès aux 2 octets contenus dans chaque mot lu, en réalisant une opération de sérialisation.

Une sous-partie opérative spécialisée a été créée pour réaliser:
- la mémorisation d'un mot de 16 bits contenant 2 octets de code,
- la sélection de l'octet suivant, contrôlée par le bit de poids faible du compteur ordinal CO,
- la construction d'un opérande cadré sur 16 bits, par masquage câblé contrôlé par le type d'opérande.

Cette réalisation est intéressante en ce sens qu'elle montre comment un chemin de donnée de hauteur 8 bits, peut être implanté dans une partie opérative de hauteur 16 bits, en coupant horizontalement le rectangle et en reliant les fils qui constituent

le bus 8 bits.



II-D- LES OPERATEURS SPECIALISES

L'exécution des instructions du PCODE requiert des opérateurs classiques, pour l'incrémentation du Compteur Ordinal, du Pointeur de Pile, et la réalisation des opérations habituelles.

II-D-1- L'ADDITIONNEUR

Une particularité apparaît cependant pour les calculs d'adresse, car il faut manipuler à la fois des adresses d'octets et des adresses de mots (2 octets): afin d'éviter des décalages, un additionneur spécial a été créé, capable de faire aussi bien +1, -1 que +2, -2, +4, -4, etc.

II-D-2- LA MULTIPLICATION ET LA DIVISION

La multiplication et la division sont réalisées classiquement par une séquence de 16 additions-décalages (ou soustractions-décalages), au moyen de 2 opérateurs:

- le 1-er est l'opérateur universel UAL ayant un décaleur en sortie, et rebouclé sur le bus B,
- le 2-ème est un simple opérateur de décalage, rebouclé sur le bus A,
- les 2 décaleurs sont liés entre eux (le bit sortant de l'un est pris comme bit entrant par l'autre).

II-D-3- LE TRAITEMENT DES OPERATIONS ENSEMBLISTES

Les ensembles (SET de PASCAL) sont codés par des bits dont la

valeur indique la présence ou l'absence d'un élément; ils occupent un certain nombre de mots.

Le test d'appartenance (fonction IN de PASCAL) est réalisé rapidement par utilisation d'un masque qui ne contient qu'un seul '1', dans la position qui correspond au numéro de l'élément.

La génération du codage d'un sous-ensemble défini par 2 valeurs I et J ([I..J] en PASCAL) est également réalisée par un générateur de masque contenant des '1' entre les positions I et J.

Le même générateur de masques est utilisé pour réaliser soit l'extraction de champs compactés, soit leur mélange avec les champs non modifiés: il facilite ainsi tous les traitements, généralement très lourds, sur les champs compactés (PACKED ARRAY ou PACKED RECORD de PASCAL).

II-E- CONCLUSION

La conception et le dessin de ce générateur de masques sont présentés dans [REF. GALLAIS84], comme l'ensemble des cellules et des mécanismes qui constituent la Partie Opérative du processeur PCODE.

III- CONCEPTION DE LA PARTIE CONTROLE

La partie contrôle du processeur PCODE est en cours de réalisation, selon les approches topologiques qui sont présentées dans les chapitres suivants.

Il a cependant été décrit, dans le langage MACSIM, (voir chapitre MACSIM) un algorithme d'interprétation du langage PCODE. Cet algorithme a été compilé (par le compilateur PASCAL), ce qui a permis de faire son exécution en pas-à-pas, pour la mise au point

de la description fonctionnelle du processeur.

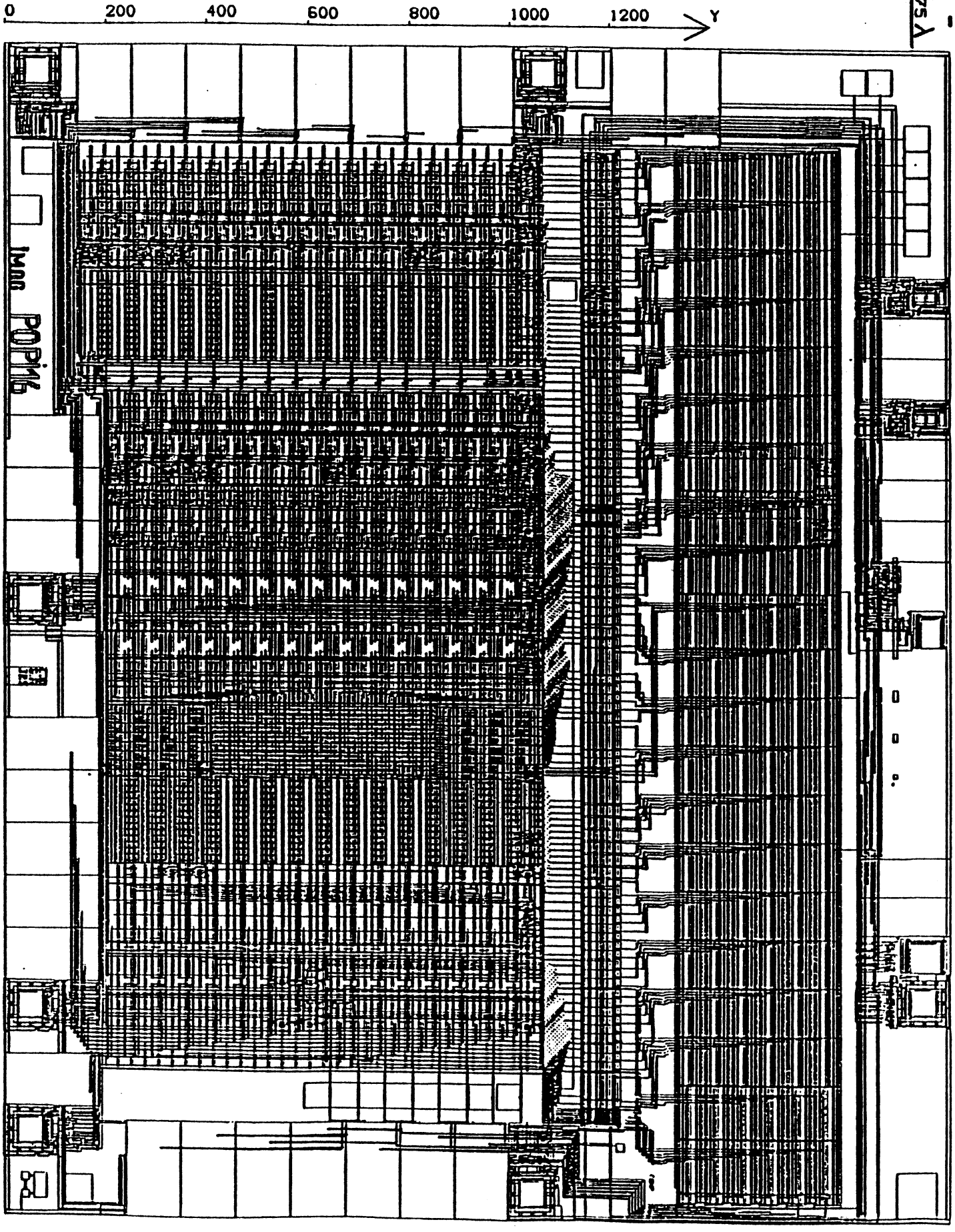
Il est maintenant possible de compiler cette description (par le compilateur de Silicium) pour en extraire la description de la Partie Contrôle, ce qui permettra d'en générer une implantation lorsque tous les outils seront opérationnels.

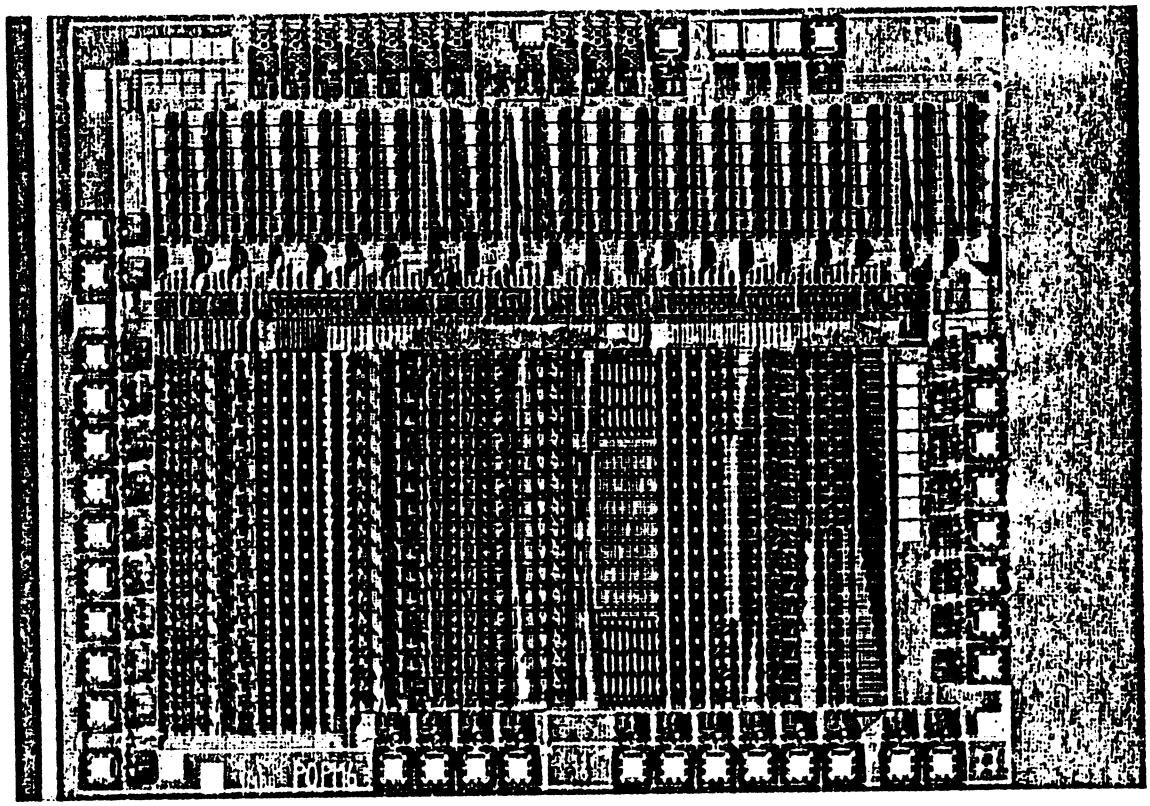
Ce travail est en cours.

PLAN DE MASSE MACHINE P.CODE

Taille circuit Echelle 1
5,13 mm
4,2 mm

$\lambda = 2,25\ mm$
$1875\ \lambda$





PHOTOGRAPHIE DU PROCESSEUR P-CODE

TROISIEME PARTIE

SOMMAIRE :

CHAPITRE 5 :

IMPLANTATION DES PARTIES CONTROLE

De la même manière que nous avons cherché à introduire un modèle pour les Parties Opératives, il nous faut définir les aspects fonctionnels, logiques, temporels et topologiques pour les Parties Contrôle, que nous appellerons aussi Séquenceurs des commandes.

Rappelons brièvement les fonctions de la Partie Contrôle (en abrégé PC):

> pour une instruction donnée du langage-machine (MACRO-instruction), la PC doit "générer" une suite ordonnée de commandes qui sont celles qu'il faut envoyer à la PO, et dans cet ordre, pour qu'elle "exécute" l'instruction en cours.

On considèrera l'accès à l'instruction suivante comme une fonction réalisée par la PO sous contrôle de la PC.

Il existe de nombreuses manières de réaliser une PC: on se reportera à la lecture de [Ref. OB-82] qui présente non seulement plusieurs organisations fonctionnelles connues et utilisées dans des processeurs LSI, mais en plus, une évaluation de la surface que leur implantation nécessiterait. Cette évaluation topologique a été faite en utilisant une approche classique qui consiste à implanter chaque fonction dans un bloc monolitique dont on sait évaluer les dimensions [Ref. REIS], puis à évaluer la surface des zones d'interconnexion nécessaires pour relier les blocs entre eux.

Aucune des organisations présentées dans [ref. OB-82] ne présente un caractère topologique suffisamment régulier pour constituer un modèle général: des tendances se dégagent cependant sur l'encombrement inhérent à telle ou telle structure; en particulier les organisations à "générateurs d'instants" semblent donner une implantation dense, au contraire des organisations "mono-PLA" qui sont consommatrices en surface. On aborde ici le problème du choix

entre la recherche d'une solution générale et automatisable qui donne dans l'ensemble de bons résultats en SURFACE et un gain en TEMPS de conception énorme, et la recherche de la meilleure solution (topologiquement parlant) pour un cas particulier.
La réponse à cette question est la suivante: lorsqu'on saura réaliser automatiquement plusieurs implantations dans plusieurs structures d'accueil, on aura besoin de choisir; ce n'est pas actuellement le cas: un peu partout dans le monde, chaque compilateur de Silicium tente une approche, qui avec des ROMS [Ref. LIPP], qui avec des PLA [Ref. BELLMAC], qui avec des réseaux de Wienberger [Ref. MacPitts MIT].

Nous allons donc présenter ici une approche personnelle, fondée d'une part sur des hypothèses et des critères topologiques globaux, et d'autre part sur la possibilité de projeter une description de haut- niveau, en la COMPILANT, sur une structure d'accueil, en définissant comme résultat de la compilation ses fonctions, le contenu de ses éléments fonctionnels, et le dessin de leurs masques.

Nous ne prétendons pas présenter ici LE modèles général des Parties Contrôle d'ordinateurs intégrés (d'ailleurs existe-t-il ?), et nous comptons sur plusieurs applications en cours pour évaluer sa généralité et son efficacité.

Nous présenterons successivement, dans cette partie:
- Les hypothèses et les critères TOPOLOGIQUES retenus,
- Deux organisations fonctionnelles implantées avec les mêmes critères topologiques:
  - leur définition fonctionnelle,
  - leur schéma temporel,
  - leur implantation topologique,
  - leur compilation.

  Ces 2 organisations sont:
  - l'architecture microprogrammée,
  - l'architecture à générateur d'instants;

CHAPITRE 1

- APPROCHE TOPOLOGIQUE GLOBALE

- APPROCHE TOPOLOGIQUE POUR LA GENERATION DES COMMANDES

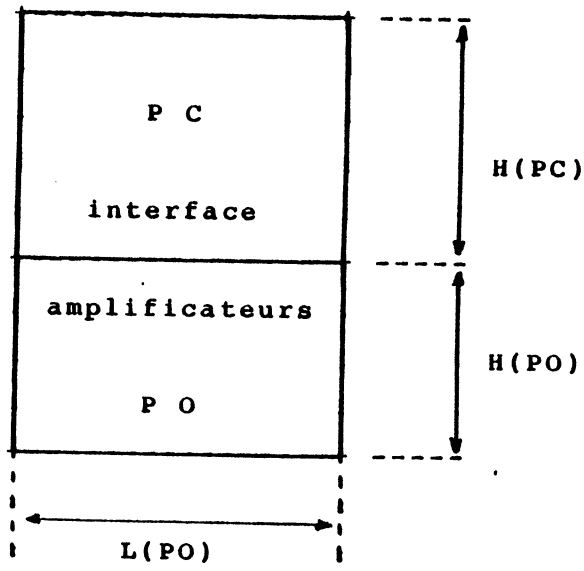
IMPLANTATION DES PARTIES CONTROLE

APPROCHE TOPOLOGIQUE GLOBALE

HYPOTHESES TOPOLOGIQUES GLOBALES

On choisit comme hypothèse de départ de restreindre la topologie globale du coeur d'un ordinateur intégré à un RECTANGLE, dans lequel s'empilent deux autres rectangles de même largeur:

- en bas la Partie Opérative et ses amplificateurs de commande, notée PO,
- au haut la Partie Contrôle et son interface, notée PC.



Pour un ordinateur donné, défini par son langage-machine, il existe de nombreuses familles de solutions qui, à technologie constante, donnent des performances et une topologie variables.

I- RELATIONS ENTRE PERFORMANCES ET SURFACE

I-a/ Performances <=> complexité de la PO:

Les performances obtenues dépendent essentiellement de la complexité de la Partie Opérative, mesurée dans deux directions:

- le nombre de bits du chemin de donnée donne la hauteur H(PO) qui

est égale à:

H(PO) = H(tranche) * NBIT + H(amplificateur)

où H(tranche) est la hauteur connue d'une "tranche" de 1 bit (BIT-SLICE),
NBIT est le nombre de bits,
et H(amplificateur) la hauteur d'un amplificateur, égale à environ 3/2 de H(tranche).

- le nombre de composants (éléments de mémorisation et opérateurs), la puissance de ces opérateurs et le degré de parallélisme (sous-parties opératives fonctionnant en parallèle) de la Partie Opérative donnent sa largeur L(PO).

Les deux quantités L(PO) et H(PO) ne varient pas d'une manière linéaire, mais suivent des courbes en escalier à marches irrégulières:
- NBIT varie par pas de 4, 8 ou 16 (ou 32 ?),
- L(PO) peut varier de quantités assez petites qui peuvent avoir une influence non négligeable sur les performances: l'addition d'un registre et d'un additionneur indépendants pour réaliser la gestion du Compteur Ordinal augmente peu L(PO), mais peut dans certains cas accroître les performances, alors que l'addition couteuse en largeur de nombreux registres peut ne pas accroître d'autant les performances.

On pourrait entrer ici dans le débat complexe propre à l'architecture des ordinateurs, sur les compromis entre la performance et le coût d'un ordinateur. On énoncera plutôt les conditions précises dans lesquelles cette étude se situe:

pour une performance fixée et une technologie donnée, on cherche à réduire à la fois
- le coût de la conception (=> structures régulières prédéfinies);
- le coût de la fabrication (=> surface minimale);
- et on devrait également prendre en compte les problèmes de testabilité, ce qui ne sera pas abordé.

Pour simplifier encore nos hypothèses et les résumer, nous raisonnerons avec un nombre de bits NBIT fixé, ce qui introduit une CONSTANTE TOPOLOGIQUE "H(PO)".

Les performances ne dépendront donc plus que de L(PO), et la Partie Contrôle PC devra bien sûr être capable de commander la PO, en exploitant le parallélisme fourni par ses ressources matérielles, et en fonctionnant à la même fréquence que cette dernière: les "temps de cycle" des 2 parties PC et PO doivent être égaux.

I-b/ Complexité de la PO <=> Hauteur de la PC

On rappelle la contrainte de départ L(PO)=L(PC).
Si L(PO) varie, et qu'on respecte la contrainte précédente, on se demande comment évolue H(PC), et donc comment évoluent:
- la forme de la puce (dx=L(PO), dy=(H(PO) + H(PC)),
- et sa surface (L(PO) * (H(PO) + H(PC))).

En d'autres termes, lorsqu'on augmente les performances de la PO, on augmente le nombre des commandes qu'elle reçoit à chaque cycle, et alors le nombre total de cycles nécessaire à l'exécution de l'algorithme (invariant) diminue:
la performance globale augmente.

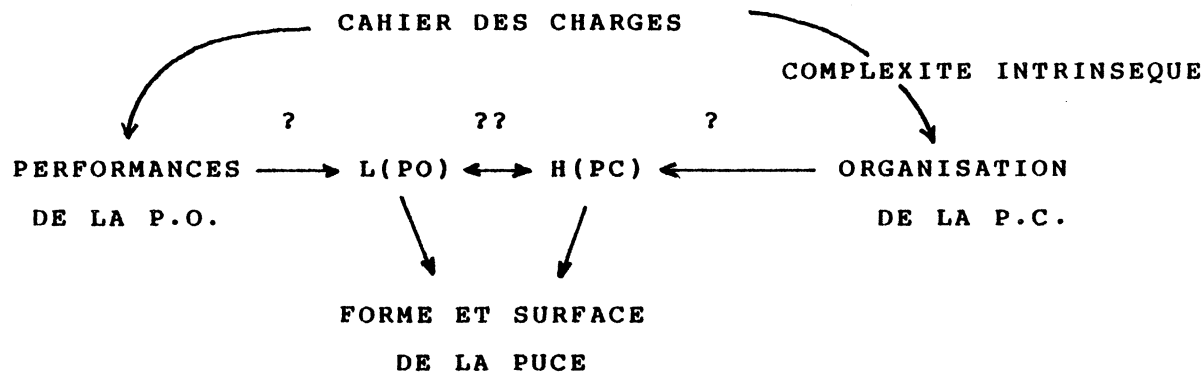
La PC a donc une complexité HORIZONTALE (nombre de commandes) qui augmente, et une complexité VERTICALE (nombre de cycles) qui diminue.

Sur le plan topologique, la complexité horizontale varie linéairement avec la largeur de la PO L(PO): on suppose en effet que la densité des fils de commande (nombre de fils par unité de longueur) est constante (ce qui peut être faux, mais les 2 valeurs évoluent toujours dans le même sens).
Par contre, on connait moins bien la variation de la hauteur de PC, qui dépend certainement dans une grande mesure de la complexité intrinsèque de l'algorithme.

I-c/ Conclusion provisoire avec la contrainte fixée

Le cahier des charges de l'ordinateur donne:

- une performance à atteindre qui impose un nombre de bits et une largeur de circuit L(PO),
- et une complexité intrinsèque de l'algorithme à mettre en oeuvre qui, combinée par une loi de variation imprécise avec L(PO), impose la hauteur du circuit (H(PO) + H(PC)).



I-d/ Modifications de la forme de la puce

Supposons qu'on obtienne, comme résultat de la conception, une puce plus haute que large. En augmentant sa largeur, donc ses performances (ce qui est intéressant), on est à peu près certain de diminuer sa hauteur, mais on n'est pas sûr de diminuer sa surface.

Si, par contre, la puce est trop large, il faut sacrifier des prformances, ce qui peut être inacceptable, en gagnant une meilleure forme. On devra donc choisir entre performance et topologie (forme + surface).

Remarque 1: une puce plus large que haute a forcément

- soit une complexité intrinsèque réduite et un nombre de bits élevé (c'est le cas de la puce RISC-II de UC Berkeley), soit une complexité intrinsèque importante et un nombre de bits réduit; la même forme de puce peut accueillir deux ordinateurs très différents (un 16 bit très complexe ou un 32 bit très simple).

Remarque 2: la puce MC68000 est plus haute que large, à cause d'une part de la complexité intrinsèque de l'algorithme et la mauvaise utilisation de la surface dans la partie basse de la PC (d'où

H(PC) grand), et d'autre part à la performance moyenne de la Partie Opérative pour certaines opérations: un multiplieur câblé augmenterait L(PO) et les performances, et permettrait vraisemblablement de diminuer H(PC). On peut aussi noter que le passage de la PO à 32 bits n'augmenterait que peu les performances du MC68000, et rendrait la puce encore plus haute que large. Ces remarques ne doivent pas être considérées comme des critiques de la qualité de cette puce, si l'on se place dans le contexte de sa conception (1979, technologie HMOS-1).

II- POSITION DU PROBLEME

La contrainte globale au niveau de la puce ayant été énoncée et discutée (L(PC) = L(PO)), le problème à résoudre se résume à:

- tenir compte de la nature et de la position des commandes de la PO supposée dessinée (puisqu'on connait L(PO)),
- compiler la description de l'algorithme pour l'implanter dans un rectangle de largeur L(PO) et de hauteur minimum (H(PC) min).
- raccorder les deux parties PC et PO en perdant le moins de surface possible pour le raccord.

Le compilateur de silicium doit, si possible, donner de bons résultats pour une gamme étendue d'algorithmes, de complexité intrinsèque variable, et pour différentes valeurs de L(PO).

II-a/ APPROCHE TOPOLOGIQUE ASCENDANTE

On part de la partie connue, c.a.d. la Partie Opérative, caractérisée par une interface rectiligne représentant le haut des amplificateurs de commandes.

Cette PO a une largeur L(PO), qu'il faut respecter globalement. Elle présente des "connecteurs" en entrée, appelés Commandes, et des "connecteurs" en sortie appelés Indicateurs (ou drapeaux, pour traduction de "FLAGS").

Une approche générale ne peut pas tenir compte des positions précises de ces connecteurs, mais seulement de leur ordre relatif, et de leur sens (attribut de type donnant un nom). On fera appel plus tard à l'assembleur de silicium (LUBRICK ou autre), pour résoudre les problèmes fins d'interconnexion, mais on tentera de créer une figure PC dont les connecteurs (SUD) sont dans le même ordre que les connecteurs (NORD) de la figure PO.

II-a-1/- Recherche d'invariants

Il est important de trouver des invariants, si possible sans dimension, quand on essaie d'aborder un problème général. Nous prendrons l'hypothèse suivante:

> "La densité moyenne des fils de commandes et des indicateurs au-dessus des amplificateurs est constante pour des règles technologiques de dessin données".

Cette propriété d'invariance, même si elle peut ne pas être vérifiée dans certains cas, peut servir de point de départ pour une approche générale qui autorise une certaine marge d'erreur (à quelques %).

Cette densité moyenne (nombre de fils par unité de longueur), ne va pas être utilisée comme telle, mais ramenée d'abord à un PAS MOYEN entre deux commandes, puis combinée à une autre constante de la technologie:

- le PAS D'UN NIVEAU DE METAL, selon une définition voisine de celle qui est donnée dans [Ref. REIS] comme invariant des formules d'évaluation de surface; ce PAS est défini comme étant celui DES LIGNES DE METAL qui servent à construire:
  - soit des matrices de PLA ou de ROM (et aussi bien les matrices ET que les matrices OU),
  - soit des tranches de parties opératives,

  en tenant compte des RAPPELS DE MASSE qui sont indispensables, et qui jouent un rôle prépondérant dans le plan de masse global.

Il existe d'ailleurs un coefficient, présenté dans [Ref. REIS], qui lie le pas minimum des fils métalliques et celui des fils "virtuels" introduit ci-dessus.

II-a-2/- Définition d'une TRAME VIRTUELLE

Ce pas de métal PM permet de couvrir tout le rectangle destiné à accueillir la Partie Contrôle avec une TRAME DE FILS VIRTUELS, espacés de PM unités, qui vont constituer la "STRUCTURE D'ACCUEIL" pour l'implantation de la PC, définissant des TRANCHES verticales, tout comme les parties opératives sont "accueillies" dans des TRANCHES horizontales (selon la convention déjà proposée pour l'orientation du circuit dans un repère orthonormé).

Le nombre de ces fils virtuels s'obtient comme une fonction du nombre N de fils de commande, du pas moyen D de ces fils de commande, et du pas PM des fils virtuels:

Nombre de fils = (N*D) / PM = (D/PM) * N = K * N

Le rapport (D/PM) est une constante K, dépendante des règles de dessin données par la technologie.

On arrive donc à un premier modèle du circuit:

> s'il y a N commandes à générer, on dispose de (K * N) fils virtuels pour implanter la Partie Contrôle.

Ces fils sont VIRTUELS en ce sens qu'ils ne seront pas forcément tous utilisés, mais qu'ils peuvent l'être, et sur toute la hauteur de la PC.
On cherchera bien sûr à les utiliser tous, puisque le rectangle de Silicium est capable de les accueillir, et on obtiendra ainsi une densité maximum pour le niveau de métalisation qui les supporte.
On chechera aussi à optimiser l'occupation de la surface qui se trouve sous ces fils, en y plaçant des composants actifs.

L'exemple des règles n-MOS normalisés du CMP [ref. Anceau],

appliqué à la conception de la Partie Opérative du processeur P-CODE POPY16 et présentée au chapitre traitant des Parties Opératives, donne les valeurs suivantes:

- 120 commandes réparties sur une largeur de PO d'environ 2400 lambda => (D = 2400/120 = 20 lambda);
- pour un nombre RAPP de fils virtuels entre 2 rappels de masse voisin de 10, on obtient un pas de métal égal à 8 => (PM = 8 lambda).

(La formule employée est (7 + 9 / n): elle correspond aux règles de dessin utilisées par les concepteurs de l'outil PAOLA pour les règles de dessin du CMP).

On obtient pour cet exemple: K = 2,5

ce qui donne :

2,5 x 120 = 300 fils virtuels disponibles pour implanter la Partie Contrôle de POPY16.

II-b/- REMPLISSAGE DE LA TRAME VIRTUELLE

Les fils Virtuels présentés plus haut définissent une "structure d'accueil" qui consiste en (K*N) fils, répartis en bandes de RAPP fils situés entre deux "rappels de masse". On a donc défini:

((K * N) / RAPP) tranches verticales "virtuelles".

Définition: les fils virtuels verticaux sont à un pas fixe, mais pas d'une couleur fixe: ils peuvent être soit BLEUS (métal) soit ROUGES (Silicium Polycristallin), et donnent leur couleur au bloc horizontal qu'ils traversent.

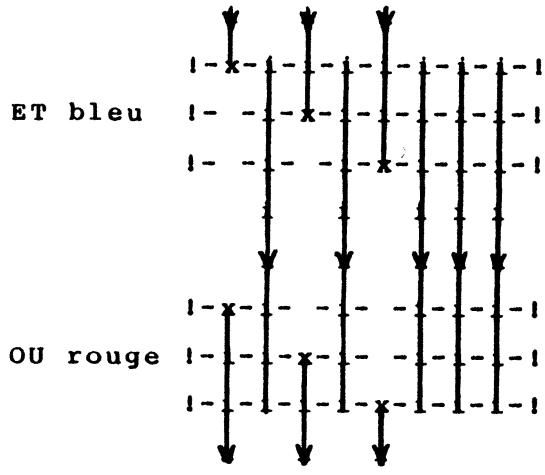
On va remplir l'espace des fils virtuels avec des blocs qui seront soit BLEUS soit ROUGES, et qui auront leurs éléments actifs (portes et transistors) EN-DESSOUS des fils virtuels.

II-b-1/- Un PLA peut être accueilli de 2 manières:

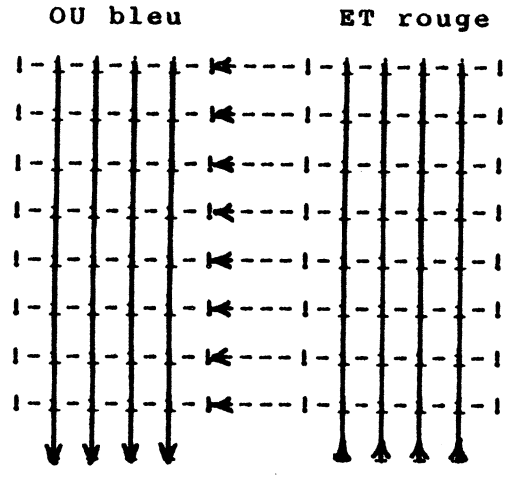
- soit en plaçant un bloc bleu (matrice ET) au-dessus d'un bloc rouge (matrice OU); le PLA peut alors avoir jusqu'à (K * N) monômes s'il occupe toute la largeur de la PC (figure (a));

- soit en plaçant un bloc bleu (matrice OU) à gauche d'un bloc rouge (matrice ET); le PLA est alors tel que (NBE + NBS) <= (K * N), si NBE et NBS sont les nombres de ses entrées et de ses sorties (figure (b)).

- soit dans un seul bloc bleu ou rouge pour un PLA MONO-MATRICE.



(a)



(b)

II-b-2/- Une mémoire morte ROM se place sous la structure, comme deux blocs bleus (matrices des points de ROM) de part et d'autre d'un bloc rouge (décodeur de l'adresse).
La ROM peut fournir NBB bits de donnée pour NBA bits d'adresse, avec: (NBB + 2*NBA) <= (K * N).
Les NBB bits fournis peuvent bien sûr être multiplexés, en sortie de la ROM, par un facteur MUX, pour donner (NBB/MUX) bits à chaque lecture; plus MUX sera grand, plus la hauteur de la ROM diminuera.

II-b-3/- Un chemin de donnée en "tranches" peut être accueilli sous la trame virtuelle:
les fils virtuels verticaux bleus peuvent alors supporter des bus et l'alimentation VCC (les lignes de MASSE sont déjà présentes). La limitation concerne le pas d'une tranche d'abord, qui doit être un sous-multiple du (ou égal au) pas des rappels de masse, et ensuite le nombre de bits qui est limité à:
((K * N) / RAPP) si les pas sont égaux (30 dans le cas de POPY).

Remarque: si le pas d'une tranche est égal au pas des rappels de masse, il y a RAPP fils virtuels qui passent au-dessus de chaque tranche de 1 bit, et l'un d'eux porte l'alimentation; il en reste donc (RAPP-1) pour des bus, ou pour des fils de commandes qui traversent le chemin de données.

II-c/ CONCLUSION

Toutes les structures régulières (ROM, PLA, Partie Opérative en TRANCHES), sont implantables SOUS la trame virtuelle, et les détails topologiques seront présentés plus loin.

Il faut noter que cette approche systématique va à l'encontre des méthodes classiques d'implantation, qui consistent à fabriquer des blocs fonctionnels topologiquement indépendants, monolithiques, puis à tenter de les placer dans la surface disponible, en utilisant des canaux de routage coûteux en surface pour les interconnecter.

Au contraire, cette approche tient compte de la contrainte topologique dès le départ, en prévoyant que toute connection doit être faite en utilisant un morceau d'un fil virtuel.

IMPLANTATION DES PARTIES CONTROLE

APPROCHE TOPOLOGIQUE POUR LA GENERATION DES COMMANDES

Ce chapitre présente l'étude de la génération des commandes de la Partie Opérative, en ne considérant que les contraintes topologiques globales énoncées au chapitre précédent. Il se décompose en 2 parties correspondant à une machine de MOORE puis à une machine de MEALY.

I - ETUDE DE LA GENERATION DES COMMANDES POUR UNE MACHINE DE MOORE

Considérons d'abord une machine (automate) de MOORE avec
- NC commandes (en sortie de l'automate),
- et NF indicateurs ou Flags (en entrée);

(on rappelle que NC+NF=N).

On dispose de K*N lignes virtuelles parmi lesquelles NC portent les commandes, NF les flags et (K-1)*N sont libres.

I-A- 1er cas: s'il n'y a aucun encodage, les NC commandes proviennent directement d'une ROM (ou d'un PLA) qui fournit NC bits de commande à chaque cycle.

La ROM est dans ce cas très large, du moins dans le cas d'un ordinateur performant, et vu le nombre de fils qui en sortent, il faut la répartir sur toute la largeur du circuit et donc on peut:
- soit la rendre transparente pour les NF fils de flags (ce qui augmente encore sa hauteur et repousse le séquenceur trés haut),
- soit disposer le séquenceur entre elle et la PO, ce qui laisse (K-1)*N lignes virtuelles pour implanter le séquenceur.

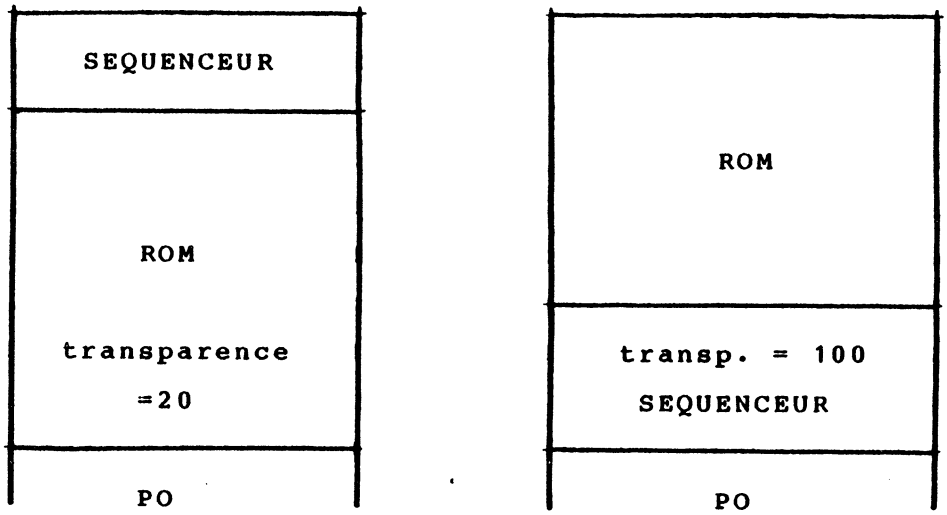
Dans le cas de POPY 16 où:

NC=100, NF=20, K=2,5

il faudrait lire 100 bits dans une ROM, et sachant qu'on dispose de 300 lignes virtuelles, on pourrait choisir parmi les implantations

suivantes:

- placer la ROM près de la PO, en la rendant transparente pour seulement 20 fils de FLAGS (NF=20) (voir Figure cas (a));
- ou placer le séquenceur entre la PO et la ROM, en le rendant transparent pour les NC=100 bits de commande: il resterait alors 300-100=200 lignes virtuelles pour l'implanter (voir Figure cas (b)).



cas (a) cas (b)

I-B- 2ème cas: afin de réduire le nombre de bits sortant de la ROM (donc sa hauteur) on définit des CHAMPS qui seront décodés pour faire la synthèse d'un groupe de commandes: on utilisera, pour chaque commande, un PLA-ET de décodage suivi d'un PLA-OU de synthèse.

On place une "couche de décodage" entre le séquenceur et la PO.

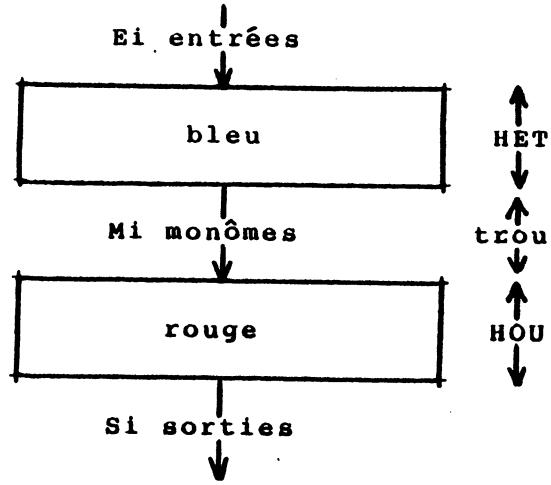
DEFINITION: chaque champ est codé sur Ei bits, donc il sort de la ROM:

SIGMA(Ei) bits.

I-B-1/ DECODAGE D'UN CHAMP DE COMMANDES

Hypothèse: les entrées et les sorties des PLA de décodage sont verticales, et elles sont donc dites "latérales" par rapport à l'orientation classique des PLAs.

- soit Ei le nombre de bits qui codent le champ i: il y a 2*Ei entrées latérales,
- soit Mi le nombre de combinaisons à calculer: il y a Mi monômes,
- soit Si le nombre de sorties synthétisées dans le PLA-OU: il y a Si fils de sorties latérales.



La hauteur du bloc formé des 2 PLA ET et OU est égale à

HET+HOU+trou (entre les 2 matrices);

HET = 2*Ei;

HOU = Si;

La largeur L est égale à

Mi+max(2*Ei,Si),

si on place sur une même ligne virtuelle verticale une entrée et une sortie.

Hi = 2*Ei + Si + trou

Li = Mi + max(2*Ei,Si)

I-B-2/ Contraintes globales:

Si on place plusieurs décodeurs de champs les uns à côté des autres, pour occuper toute la largeur de la PC, on dispose de K*N lignes virtuelles pour générer N sorties.

On doit donc avoir:

SIGMA(Si) = (N - NF) = NC

SIGMA(Li) <= K*N - NF

où NF est le nombre de commandes directes ou flags montants.

donc:

SIGMA(Li) = SIGMA(Mi) + SIGMA(max(2*Ei,Si))

Simplification:

Supposons que pour tout i , max(2*Ei,Si) = Si ,
c'est-à-dire qu'un champ codé sur Ei bit permet la synthèse de Si commandes, avec Si > 2*Ei, ce qui est un critère de définition des champs.

Par exemple:

E=2 -> S>4 4 combinaisons de S fils

E=3 -> S>6 8 " "

E=4 -> S>8 16 " "

alors on simplifie:

SIGMA(Li) = SIGMA(Mi) + SIGMA(Si)

et on obtient:

SIGMA(Mi) <= (K-1)*N

c'est-à-dire que

"toutes les lignes virtuelles libres peuvent être utilisées pour porter des monômes".

Exemple de POPY: N=120, K=2,5 => SIGMA(Mi) <= 180

Autre simplification:

Ei bits permettent de coder jusqu'à 2^{Ei} combinaisons;

On a Mi <= 2^{Ei} ($\log_2(Mi) <= Ei$)

donc

SIGMA(Mi) <= SIGMA(2^{Ei})

=> EVALUATION GLOBALE dans le cas d'un automate de MOORE:

$$SIGMA(2^{Ei}) <= (K-1)*N$$

CONSEQUENCE: le choix de l'encodage des champs se traduit par une contrainte TOPOLOGIQUE globale:

"la somme des combinaisons décodées ne peut pas dépasser le nombre de lignes virtuelles libres ((K-1)*N)".

QUELLES SONT LES VARIATIONS POSSIBLES

Depuis l'absence d'encodage (Ei=1, Mi=0 => SIGMA(Mi)=0) jusqu'à un encodage dans un seul champ de toutes les commandes (i=1, E1=n, $M1<=2^n$ => comparer 2^n avec (K-1)*N), il existe une infinité (bornée) de solutions.

Exemple de POPY: il y a 120 commandes, supposons qu'il y ait 1000 vecteurs de commandes possibles: on encode le numéro du vecteur dans un seul champ de 10 bits, et on calcule 1000 monômes (chacun fait la synthèse des 120 commandes).
On a dans ce cas besoin de 1000 monômes, ce qui est très supérieur aux 180 lignes virtuelles libres. On cherchera donc un découpage en champs qui permette d'occuper au mieux ces 180 lignes virtuelles.

I-B-3/ RAISONNEMENT SUR UN GROUPE DE COMMANDES (SOUS-ENSEMBLE)

a/ Pour un sous-ensemble donné de fils de commandes un compilateur peut calculer le nombre des combinaisons de ces fils:

nf fils -> nc combinaisons

On en déduit le codage d'un champ de e bits, où e est le plus petit entier tel que $nc <= 2^e$.

b/ Pour le même sous-ensemble de nf fils, on peut faire des découpages en deux groupes de nf1 et nf2 fils (nf1 + nf2 = nf). Le compilateur calcule deux valeurs nc1 et nc2 des combinaisons

des deux champs qui sont maintenant devenus indépendants.

Comparaison: les deux champs sont codés sur e1 et e2 fils.

Il faut comparer

- (e1+e2) avec e (pour le nombre de bits sortant de la ROM)
- et (nc1+nc2) avec nc (pour le décodage).

On a nc <= nc1*nc2,

et généralement nc1*nc2 > nc1+nc2.

Par contre (e1+e2) peut être supérieur à e.

Exemple 1:

1-er découpage avec 1 champ : nc=50 => e=6

2-nd découpage avec 2 champs: nc1=7 => e1=3

nc2=8 => e2=3

Dans cet exemple, le gain sur la ROM est nul (e1+e2=e), par contre le gain en nombre de monômes est important:

nc - (nc1+nc2) = 50-15 = 35 (=70%).

Exemple 2:

1-er découpage avec 1 champ : nc=30 => e=5

2-nd découpage avec 2 champs: nc1=14 => e1=4

nc2=10 => e2=4

Dans cet exemple, le découpage apporte une perte de 3 bits sur la ROM (e1+e2=8, e=5) et un gain en monômes assez faible:

on passe de 30 à 24 => gain=6 (=20%).

==> PREMIER COMPROMIS entre:

- le Nombre de bits sortant de la ROM et
- le Nombre de monômes à calculer.

I-B-4/ CRITERES DE CHOIX

- les lignes virtuelles libres ne coûtent rien => on cherche à maximiser le nombre total de monômes jusqu'au remplissage,
- les bits de ROM coûtent cher en hauteur de ROM et en nombre de lignes virtuelles au travers du séquenceur => on cherche à

minimiser leur nombre.

a/ Expression mathématique:
Trouver un découpage en NC champs de Ei bits décodés par Mi monômes, tel que
- SIGMA(Ei) soit le plus petit possible
- et SIGMA(Mi) soit le plus proche possible de (K-1)*N.

b/ Premier critère topologique:
La hauteur d'un décodeur de Ei entrées et Si sorties est proportionnelle à (2*Ei+Si), en l'absence de toute optimisation topologique (brisage des lignes de sortie par PAOLA). Si l'on veut un résultat topologiquement acceptable, il faut que toutes les valeurs (2Ei+Si) soient du même ordre de grandeur, pour éviter d'avoir des grosses bosses ou des grands creux.

Comme Si commandes liées dans un champ peuvent avoir au maximum 2^{Si} combinaisons, on en déduit que Ei <= Si et donc que

(2*Ei+Si) <= 3*Si.

On prendra donc le critère suivant:
le nombre de fils de commandes regroupés dans un champ sera pris dans le voisinage d'une valeur moyenne Sm, afin d'uniformiser les hauteurs des décodeurs (<=3*Si).

c/ Second critère topologique:
La définition de champs implique une concentration de la synthèse d'un groupe de commandes en une région précise: le décodeur des entrées et les monômes qui réalisent la synthèse sont regroupés dans deux PLA-ET et PLA-OU proches et denses (sauf transparence verticale bien-sûr).

Les commandes synthétisées par le PLA-OU sont disponibles sur des fils de métal horizontaux. Si toutes les sorties se font verticalement, il faut prévoir une zone de routage entre le PLA de synthèse et la destination des commandes.

On a donc une contradiction entre:
- la répartition (dispersion) des commandes (donnée par la PO),
- la centralisation de leur génération (pour optimiser l'occupation des lignes virtuelles et la taille de la ROM).

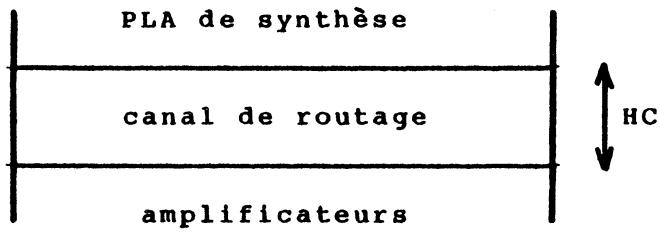
Cette contradiction ne peut pas être levée (sauf dans le cas dégénéré où chaque commande est codée par un champ de 1 bit) mais on peut proposer une approche de solution:

i/- Principe de localité:

Si plusieurs sous-parties opératives existent, pour permettre un parallélisme du traitement, elles reçoivent leurs commandes en parallèle: la synthèse de ces commandes peut être regroupée dans des zones qui se situent au-dessus de chaque sous-partie.

ii/- Placement barycentrique:

pour chaque sous-Partie Opérative, la couche des décodeurs fournit Ni commandes:



Il s'agit de choisir la position des DECODEURS, de telle sorte que la hauteur du canal de routage HC soit minimale: on calcule un placement barycentrique de chaque décodeur, de telle sorte que les commandes les plus centrales aient des chances d'être connectées directement. Si l'on réalise des PLA mono-matrice, le problème reste inchangé, le placement des entrées, des monômes et des sorties devant être fait pour minimiser le routage des sorties.

iii/- Découpage fonctionnel:

On tient compte du critère topologique de minimisation du routage dans la définition des champs.

iiii/- Commandes fortement liées:

Si on trouve un sous-ensemble de Si commandes pour lequel Si est grand et Mi petit (donc Ei petit), on peut définir un champ, bien qu'on fabrique une bosse, à condition que la partie vide créée puisse être utilisée comme une extension du canal de routage, sans pénalisation pour la forme globale.

On sortira directement les fils de commande, horizontalement, gagnant ainsi des lignes virtuelles verticales (sachant que Si>>Ei).

d/ RESUME DES CRITERES ET DU PROBLEME

Il faut trouver un découpage en champs codés sur Ei bits et donnant chacun Mi monômes, tel que:

- SIGMA(Ei) soit minimal (optimisation de la ROM),
- ((K-1)*N - Mi) soit minimal (optimisation des Lignes Virtuelles),
- la hauteur HC du canal de routage ou la hauteur maximale de PLAs mono-matrice entre les décodeurs et les amplificateurs soit minimale,
- les hauteurs (2Ei+Si) des PLA soient du même ordre de grandeur.

C'est un problème complexe à résoudre, à cause de:

- la quantité de solutions possibles
- la grandeur des valeurs de N (> 100 commandes)
- et le nombre total de combinaisons des N bits (> 1000 combinaisons).

On pourrait programmer des algorithmes de recherche exhaustive de toutes les combinaisons (les calculs seraient très longs vu le nombre de combinaisons possibles).

Il faut donc trouver des heuristiques ou des critères pour éliminer certaines combinaisons.

I-C- PROPOSITIONS

I-C-1/ Critères fonctionnels à prendre en compte:

- les différentes sources d'un bus sont mutuellement exclusives, mais les commandes sont dispersées. On a donc un CHAMP naturel fonctionnellement , mais un coût incompressible pour le routage.
- les différents registres d'un bloc de registres sont adressés par un numéro, qui définit un champ, et le coût du routage est nul si le décodeur est placé au-dessus du bloc de registres.
- un opérateur arithmétique et logique nécessite un nombre de commandes assez important, mais toutes les combinaisons ne sont pas utilisées: il y a donc peu de monômes et beaucoup de sorties. Le coût du routage peut être nul.

En appliquant seulement des critères fonctionnels sur des sous-parties opératives, on arrive à un bon résultat pour SIGMA(Ei), SIGMA(Mi), et le routage n'est coûteux que pour les sources ou destinations des bus qui peuvent être dispersées, en restant cependant localisées à une sous-partie opérative.

On peut se contenter alors d'optimisations locales à ces sous-parties, en appliquant un critère topologique qui minimise le routage:
a/ en détruisant des champs fonctionnels naturels,
b/ en mélangeant des champs indèpendants.

Exemple: mélange de deux décodeurs au-dessus d'un bloc de registres;
soit n le nombre de bits pour coder le numéro d'un registre parmi 2^n.

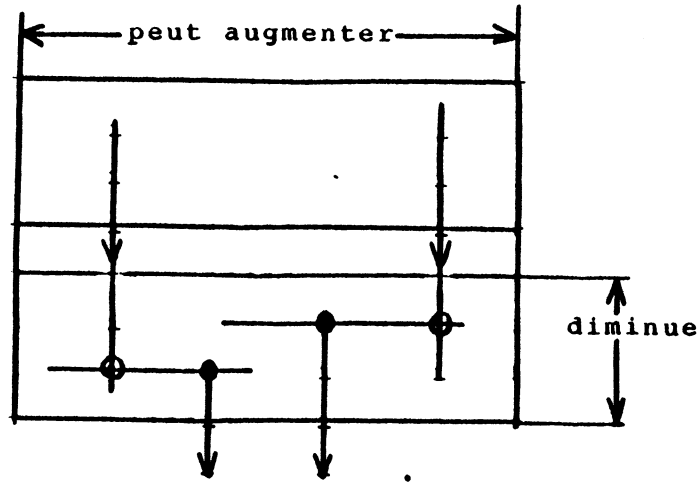
a/- 2 décodeurs côte-à-côte occupent chacun un rectangle de hauteur (2n) et de largeur ($2n+2^n$), mais un canal de routage d'au moins 2^n lignes est nécessaire.
La hauteur totale est donc: $2n + 2^n$.

b/- les 2 décodeurs sont mélangés: ils occupent un rectangle de hauteur (4n) et de largeur $2*(2n+2^n)$.
On a donc la même largeur mais une hauteur 4n au lieu de $2n+2^n$.
(On sait que $2^n > 2n$ dès que $n>2$)
Le gain en hauteur est supérieur ou égal à:

$$2^n - 2n.$$

I-C-2/ AMELIORATION: optimisation topologique des PLA de synthèse par lignes "brisées" (PAOLA).

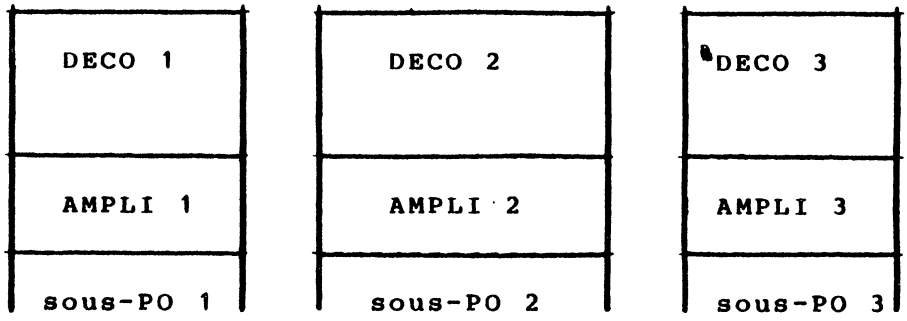
Jusqu'à ce point, la hauteur d'un PLA de synthèse a été présentée comme étant proportionnelle au nombre de ses sorties Si. Ceci devient faux si une optimisation topologique est possible, qui consiste à calculer plusieurs sorties sur la même ligne horizontale, au prix d'une duplication éventuelle de certains monômes (utilisation de lignes virtuelles supplémentaires).



La forme des 2 PLA dépend de la largeur imposée: si on étire le PLA-ET dans le sens de la largeur, on peut "dupliquer" des monômes (les calculer deux fois) et diminuer ainsi la hauteur du PLA-OU. On tient compte également des lignes virtuelles nécessaires aux entrées et aux sorties verticales.
=> en augmentant SIGMA(Mi), on réduit la hauteur des PLA-OU.

I-D- CONCLUSION PROVISOIRE:

On a défini une structure d'accueil pour les décodeurs des champs de commandes, constituée par une couche de PLAs implantée juste au-dessus de chaque sous-partie opérative: le placement barycentrique assure, pour chaque sous-partie, un canal de routage minimal, et les sorties latérales des PLA sur des fils virtuels (rouges) peuvent être directement connectés aux entrées des amplificateurs correspondants.



On découpe ainsi la complexité du problème d'implantation en se restreignant à une étude locale aux sous-parties opératives: ces dernières fonctionnent en effet en parallèle, et reçoivent donc des ensembles de champs de commandes disjoints.

Pour chaque sous-ensemble de commandes, on applique les critères proposés plus haut, en réalisant l'implantation des matrices des PLA sous les fils virtuels, définissant ainsi globalement un découpage en tranches verticales, où chacune des tranches (DECO1, DECO2, ... dans la figure) peut être optimisée par l'utilisation d'un seul grand PLA dit mono-matrice.

II - ETUDE DE LA GENERATION DES COMMANDES POUR UNE MACHINE DE MEALEY

Ce cas se distingue du précédent (machine de MOORE) par le fait que la génération d'une commande ne dépend pas uniquement de la valeur d'un champ de la microinstruction, mais également de la valeur d'un PARAMETRE (qui peut être considéré comme une ENTREE de l'automate): la valeur d'une SORTIE de l'automate ne dépend pas uniquement de son état interne.

Ce PARAMETRE est en fait une variable d'état de la MACRO-MACHINE, c'est-à-dire soit le registre contenant l'INSTRUCTION COURANTE, soit une bascule contenant un indicateur ou Flag.

Une autre définition simple est la suivante:

> "le séquenceur valide la prise en compte d'un paramètre pour le calcul d'une commande".

II-A- DEFINITIONS DE LA PARAMETRISATION:

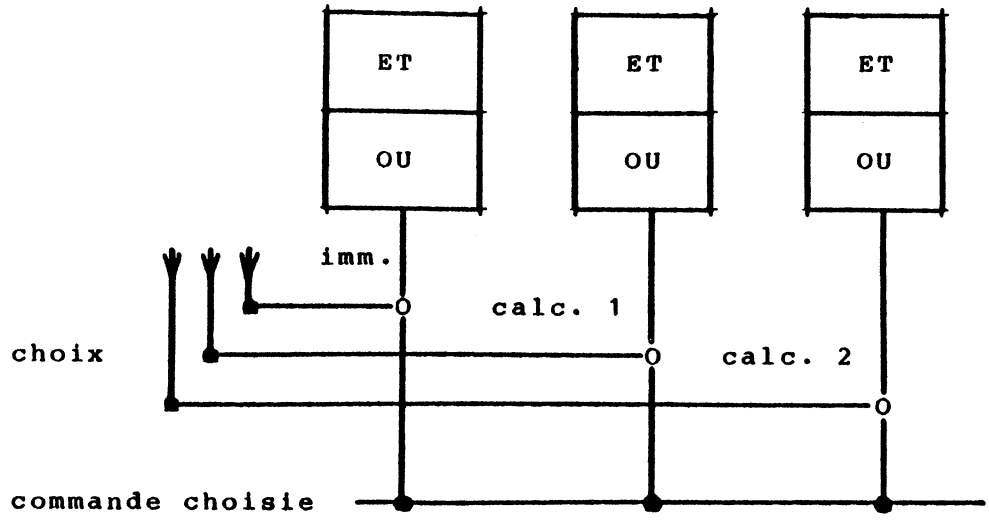
On définit plusieurs classes de paramétrisation.

II-A-1/ Définition de la paramétrisation sélective

Le séquenceur peut:

- soit fournir une commande immédiate,
- soit sélectionner une commande calculée en fonction de paramètres.

Il faut alors choisir entre les valeurs fournies par plusieurs "fonctions", évaluées par un seul ou plusieurs PLAs de synthèse de la commande.



Remarque: tous les PLA-OU de synthèse ont la même hauteur proportionnelle à Si, par contre les PLA-ET peuvent avoir des hauteurs variables dépendant

- du nombre de commandes immédiates pour l'un,
- du nombre de paramètres pour les autres.

La somme de tous les monômes à calculer donne la valeur Mi pour le champ concerné.

Cette organisation fait apparaître un nouveau type de codage pour le champ:
si le champ est codé sur Ei bits, il y a parmi les 2^{Ei} combinaisons possibles les cas de génération de commandes immédiates, plus les cas de sélection d'une commande calculée.

II-A-2/ Définition de la paramétrisation combinée

Par rapport à la définition précédente, on peut combiner dans un PLA-ET unique

- la valeur du champ provenant de la ROM,
- les valeurs des paramètres,

pour faire la synthèse des commandes dans un PLA-OU unique.

PROPOSITION:

Dans ce cas, le nombre total de monômes à calculer reste inchangé

par rapport au cas A/.

Par contre, la hauteur du PLA-ET devient égale à la somme des hauteurs des PLA-ET du cas A/, sauf optimisation due au fait que les mêmes paramètres peuvent être utilisés pour calculer plusieurs combinaisons.

D'autre part, la hauteur totale occupée diminue de Si lignes, plus un nombre de lignes égal au nombre de choix de la solution A/.

II-A-3/ **Comparaison des 2 cas:**
On peut comparer les hauteurs des cas A et B:

Cas 1: HA = max(2Ei, 2P1, 2P2,...) + 2*(HS+Si)

Cas 2: HB = 2Ei + SIGMA(2Pj) + Si
--- somme optimisée

Simplification:
HA = 2Ei + 2(HS+Si)
HB = 2Ei + 2P + Si

Différence: HA-HB = 2*HS + Si - 2*P

Discussion:
a/- Si le nombre total de paramètres P intervenant est petit (cas général), et de l'ordre de grandeur de HS (nombre de sélections différentes), le gain en hauteur est de l'ordre de Si:
(HS = P) => GAIN = Si

b/- Par contre, si P est grand, et si Hi et Si sont petits, le gain peut devenir négatif.

Remarque: la hauteur du PLA de synthèse peut être optimisée et devenir inférieure à Si, par brisure de lignes (PAOLA).

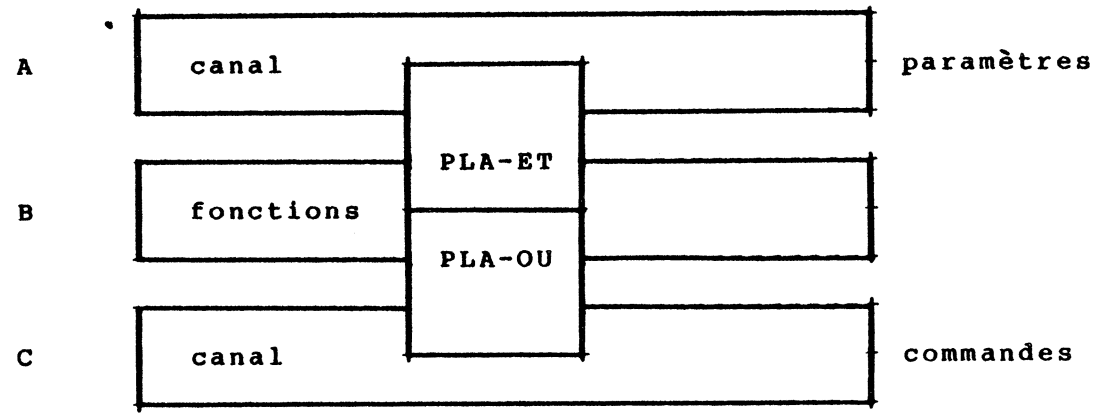
II-B- PROVENANCE DES PARAMETRES

Les paramètres proviennent de la partie opérative (Registre Instruction RI ou Indicateurs (Flags)): ils peuvent arriver verticalement depuis le bas, par une ligne virtuelle, ou horizontalement sur une ligne de routage.

On définit un NOUVEAU CANAL DE ROUTAGE pour les paramètres: c'est un canal horizontal qui amène les paramètres jusqu'aux PLA-ET où ils sont utilisés pour le calcul de monômes.

=> On cherchera à minimiser la hauteur de ce canal, en plaçant les PLA de calcul: cette minimisation peut être en contradiction avec celle du canal de routage des commandes, mais elle est cohérente avec les critères de construction de PLAs mono-matrice.

On arrive donc à la définition suivante:
A - CANAL de routage des paramètres (peut accueillir des PLA-ET)
B - ZONE de décodage et synthèse des commandes
C - CANAL de routage des commandes (peut accueillir des PLA-OU).



Une particularité du CANAL A est qu'il peut véhiculer des paramètres PERMANENTS sur toute la largeur du circuit (par exemple des bits du Registre Instruction RI), ou des paramètres LOCAUX dont l'utilisation se réduit à une petite portion (par exemple les Indicateurs (Flags)).

II-C- DISCUSSION SUR LE DEGRE DE PARAMETRISATION

La paramétrisation de type MEALY réduit la complexité VERTICALE de la Partie Contrôle: des instructions dont les algorithmes d'interprétation sont voisins peuvent être rendues équivalentes pour l'automate de MOORE, ce qui entraîne une réduction de la hauteur H(PC).

La complexité HORIZONTALE est elle limitée par la topologie: on rappelle la contrainte L(PC)=L(PO).
Cette étude montre les limites du compromis à trouver:

> on peut réduire H(PC) jusqu'à ce que L(PC) devienne égal à L(PO) qui est donné, en utilisant au maximum les lignes virtuelles disponibles.

II-C-1/ **Influence sur l'écriture de l'algorithme:**

La paramétrisation est directement issue de l'écriture de l'algorithme; elle peut être absente, ou partielle, ou totale, et il existe un très grand nombre de "formes de MEALY" pour une seule "forme de MOORE".

Dans notre cas, nous partons d'une forme algorithmique choisie par le concepteur, qui va être compilée, et le compilateur fournit comme résultat les "équations de calcul" des commandes paramétrées, en distinguant:
- les fonctions de calcul (=> remplissage de PLAs);
- les fonctions de sélection des commandes paramétrées,

soit selon une technique de sélection, ou selon une technique de combinaisons.

Ces résultats doivent ensuite être évalués en termes de coût topologique, et l'écriture de l'algorithme peut alors être modifiée jusqu'à l'obtention d'une bonne solution topologique.

II-C-2/ Conclusion

Cette approche montre les problèmes à résoudre, et met en évidence

les compromis à trouver entre la paramétrisation d'un algorithme et le coût en surface de l'implantation de la "couche" de calcul des commandes paramétrées.
Il semble important de disposer d'un système où le compilateur du langage de haut-niveau (IRENE) et un évaluateur topologique de PLAs mono-matrice soient couplés, pour trouver une solution topologiquement acceptable, en fonction des lignes virtuelles dont on dispose.

CHAPITRE 2

IMPLANTATION

DES

PARTIES CONTROLE

APPROCHE PAR
LA COMPILATION
D'UN
LANGAGE DE HAUT NIVEAU

- I : LANGAGE "IRENE"

- II: COMPILATEUR PROTOTYPE "MACSIM"

I N T R O D U C T I O N

Le langage IRENE a été défini par F.ANCEAU et par l'auteur, et présenté par K.JAHIDI et S.MARINE dans [REF. IRENE].

Ce langage de haut niveau, proche de PASCAL dont il reprend à peu près la structure de blocs et la syntaxe, a pour but de permettre l'expression de mécanismes que l'on sait réaliser par du matériel, ou par du micro-logiciel (microprogrammation). Il contient toutes les structures de contrôle que l'on sait réaliser par un séquenceur microprogrammé, et la plupart des principes du traitement de l'information que l'on sait réaliser par des portes logiques ou des interrupteurs. Il tient compte également des particularités des technologies modernes, à savoir les logiques 3-états et les logiques prioritaires. Il connait en outre les structures de base des composants modernes: le PLA est une entité du langage.

Cette définition est le fruit d'une longue expérience en conception d'ordinateurs qui a mis en évidence les besoins réels des concepteurs en langages de description, tant fonctionnels que structurels.

Cette définition a d'autre part été influencée par la nature des traitements (compilations) qui doivent être réalisés sur une description écrite en IRENE:
une description doit être compilée pour être transformée:

- en une entrée pour un simulateur fonctionnel ou structurel;
- ou en une entrée pour un assembleur de silicium.

Une description doit donc contenir toute l'information relative à ce que doit faire la machine décrite, pour qu'un compilateur génère le dessin des masques d'un circuit qui réalise les mêmes fonctions.

Ce chapitre présente quelques approches pour la compilation d'IRENE, afin d'orienter la réalisation en cours du système SIRENE, pour lequel un analyseur syntaxique fonctionne déjà (il utilise un transformateur de grammaire réalisé selon les principes présentés dans [Ref. GRIFFITHS-PELLETIER et Ref. DELAUNAY).

Ce projet de compilation d'IRENE se situe dans un cadre européen: le projet CVT.

I- COMPILATION DU LANGAGE DE DESCRIPTION DE L'ALGORITHME

I-A- INTRODUCTION

I-A-1/ Rappels sur le langage IRENE

La définition du langage IRENE comportemental (ou fonctionnel) présentée dans [ref. MARINE] fait apparaître des mécanismes de description proches d'une implémentation matérielle et distingue:

- les instructions opératives qui peuvent être conditionnelles:
  if <exp> then <action1> else <action2> endif
- les actions opératives qui peuvent elles-aussi être des transferts de données conditionnels:
  <destination>:= if <exp> then <source1> else <source2> endif

Exemple

```
DECLARATIONS:
    registre REGX;
    busa, busb;     (* variables instantannées *)
INSTRUCTIONS:
    if C1 then REGX:=if C2 then busa else busb endif
          else REGY:=if C3 then busa endif
    endif;
```

L'exécution d'une action opérative est liée à l'ETAT courant: dans cet état, on doit envoyer une commande, à la partie opérative, pour qu'une action soit exécutée.

On peut transformer la forme syntaxique IRENE, pour faire apparaître les noms de 3 commandes physiques, en :

```
"REGX:=busa"   =   (C1 and C2);
"REGX:=busb"   =   (C1 and not C2);
"REGY:=busq"   =   (not C1 and C3);
```

L'exemple précédent aurait pu s'écrire:

```
if C1 then if C2 then REGX:=busa else REGX:=busb
      else if C3 then REGY:=busa;
```

ce qui est du PASCAL, et dont l'exécution par PASCAL donnerait le même résultat.

I-A-2- Supposons maintenant que C2 soit, non pas une condition à tester, mais un paramètre égal à un bit d'état de la macromachine. Le séquenceur ignore alors la valeur de C2 et génère la commande:

(C1) = "REGX:=if C2 then busa else busb";

Interprétation matérielle:
Le registre REGX a une commande unique de chargement: la valeur chargée est calculée dans un multiplexeur contrôlé par C2.

I-A-3- Supposons que C1 soit, non pas une condition à tester, mais un paramètre. Le séquenceur ignore alors la valeur de C1 et génère la commande:

"REGX:=if C1 then if C2 then busa else busb";

=> Conséquence:
On a mis en évidence 3 formes syntaxiques correspondant à 3 réalisations différentes:

(a) if C1 then if C2 then REGX:=E1 else REGX:=E2 endif endif;
(b) if C1 then REGX:=if C2 then E1 else E2 endif endif;
(c) REGX:=if C1 then if C2 then E1 else E2 endif endif;

=> Simplification
Les trois formes précédentes peuvent être écrites:
(a') REGX:=if C1 then if C2 then E1 else E2;
(b') REGX:=if C1 then E3;
(c') REGX:=E4;

De la forme (a') on déduit: "REGX:=E1" = (C1 and C2);
"REGX:=E2" = (C1 and not C2);
De la forme (b') on déduit: "REGX:=E3" = (C1);
De la forme (c') on déduit: "REGX:=E4" = VRAI;

I-A-4- CONCLUSION

Les trois formes IRENE (a'), (b') et (c') ne sont pas plus riches que les trois formes PASCAL (a), (b) et (c) (ni moins ambigues), elles sont seulement plus agrèables à écrire et surtout elles sont facilement analysables.

I-A-5- Applications

Il s'agit de faire apparaître l'interface de commande, constituée par l'ensemble des fils de commande, et les équations de calcul de ces commandes (conditions d'activation de ces fils).

PRINCIPE SIMPLE: toute commande se décrit par une instruction d'affectation ou de connexion, selon qu'il s'agit d'un chargement de registre ou d'une connexion sur un fil (bus par exemple).
"REGX:=E1", et "REGX:=E2" sont considérés comme 2 formes syntaxiques différentes si E1 est différent de E2.

Conséquence:
1/- une recherche des parties gauches "REGX:=...;" permet de trouver toutes les parties droites.
2/- l'analyse des parties droites permet de trouver toutes les commandes et leurs équations (formes (a'), (b') et (c')).

I-B - COMPILATION DES ACTIONS OPERATIVES

On reconnaît les formes symtaxiques

"DEST:=⟨expression⟩"

et on encode les différentes valeurs des ⟨expressions⟩.

I-B-1/ EXEMPLES:

exemple 1:

"busa:=source1" => CODE=1
"busa:=source2" => CODE=2
"busa:=if C3 then source1 else source2 endif" => CODE=3
"busa:=if C4 then source2 else source3 endif" => CODE=4
"busa:=source4" => CODE=5

On en déduit la génération des commandes:

4 commandes	équations
"busa:=source1"	(CODE=1)ou(CODE=3 et C3)
"busa:=source2"	(CODE=2)ou(CODE=3 et nonC3)ou(CODE=4 et C4)
"busa:=source3"	(CODE=4 et nonC4)
"busa:=source4"	(CODE=5)

exemple 2:

"REG:=busb" => CODE=1
"REG:=if C2 then busb" => CODE=2
"REG:=if C3 then busb" => CODE=3

Une seule commande doit être générée:

"REG:=busb"	(CODE=1)ou(CODE=2 et C2)ou(CODE=3 et C3)

exemple 3:

"REG:=busb" => CODE=1
"REG:=busa" => CODE=2
"REG:=if C3 then busb" => CODE=3
"REG:=if C4 then busa else busb" => CODE=4

Deux commandes sont à générer:

"REG:=busb"	(CODE=1)ou(CODE=3 et C3)ou(CODE=4 et nonC4)
"REG:=busa"	(CODE=2)ou(CODE=4 et C4)

I-B-2/ Généralisation:

Toute forme syntaxique:

<DESTINATION> ':=' if <exp> then <SOURCE>

est encodée et la génération de la commande est obtenue par:

if (CODE et <exp>) then <DESTINATION> := <SOURCE>

La synthèse de la commande est obtenue par le OU de toutes les formes:

if ((CODE1 et <exp1>)ou(CODE2 et <exp2>)ou(...)) then
then DESTINATION := SOURCE endif;

I-B-3/ Définition de la FUSION de plusieurs codes:

exemple:

```
case RI<0> of '0:busa ; '1:busb endcase;      => CODE1
case RI<1> of '0:busb ; '1:busa endcase;      => CODE2
```

Les 2 codes CODE1 et CODE2 peuvent être fusionnés en un seul code si on ajoute un paramétre dont la valeur permettra de distinguer les anciens cas:

```
case RI of
   xxxx0 : busa;
   xxxx1 : busb;
   yyy0y : busb;       => 1 seul code pour 4 cas
   yyy1y : busa;
endcase;
```

COMPROMIS ENTRE:

- le nombre de BITS pour encoder un champ et distinguer les fonctions de calcul (=> taille de la ROM et hauteur des PLA-ET);
- le nombre de BITS de paramètres qui distinguent par eux-mêmes les fonctions de calcul (=> hauteur des PLA-ET).

I-B-4/ Proposition d'une HEURISTIQUE:

On considère que si on prend tous les bits du registre instruction comme paramètres, on obtient une distinction maximale par la valeur courante de ce registre RI.
On optimise ensuite en supprimant, dans l'ensemble des paramètres, les bits de RI qui n'interviennent pas, par une optimisation logique de tous les PLA-ET de calcul des combinaisons.

Problème:
la valeur du registre instruction RI peut ne pas suffir pour distinguer deux cas si on rencontre deux (ou plus) paramétrisations pour la même instruction:
exemple:
pour les instructions dont les codes valent 120, 121, 122 et 123, supposons qu'on ait les 2 instructions suivantes:

```
busa:= case RI of 120:source1; .... endcase;
busa:= case RI of 120:source2; .... endcase;
```

Il faut dans ce cas définir 2 codes différents, qui permettront de distinguer les deux sélections calculées en fonction de RI: on doit dans ce cas doubler le nombre des monômes dans les PLA de calcul.

I-C- CONCLUSION

En compilant la description écrite en IRENE, on peut CONSTRUIRE la couche de paramétrisation, en définissant un PLA de calcul pour chaque commande, et on obtient également le nombre de CODES nécessaires pour cette commande.

ALGORITHME PROPOSE:

1/ Recherche des mots-clés symbolisant une DESTINATION
Toutes les instructions composant la description d'un CYCLE-machine

sont concaténées dans une grande chaîne de caractères.

On recherche dans cette chaîne la présence d'un MOT-CLE ; s'il existe on trouve son "opérande" comme étant la chaîne située entre le mot-clé lui-même et le prochain ';'.

La chaîne "opérande" est alors analysée et comparée avec les valeurs des opérandes déjà rencontrées, afin de savoir s'il s'agit d'une nouvelle valeur (ou d'une ancienne).

a/ Il n'y a PAS DE PARAMETRISATION
Dans ce cas, tous les opérandes trouvés sont des expressions simples, identificateurs ou constantes.

b/ Il existe une PARAMETRISATION
La paramétrisation s'exprime par une forme syntaxique appelée expression complexe, qui peut être soit la forme IF soit la forme CASE.

- si la phrase trouvée existe déjà telle quelle comme valeur de l'opérande, il ne s'agit pas d'un nouveau cas de paramétrisation, et un "code" existe déjà.

- si elle n'existe pas, se pose le problème d'une FUSION possible avec un cas existant, par adjonction d'un paramétre;
supposons que:
"if E1 then VAL1 else VAL2 endif"
existe déjà lorsqu'on rencontre:
"if E2 then VAL1 else VAL2 endif".
On ajoute un paramètre (valeurs P1 et P2) et on fusionne les 2 cas en:
"if ((E1 et P1)ou(E2 et P2)) then VAL1 else VAL2 endif"

Le problème consiste à trouver le paramètre supplémentaire.

Solution: on associe à chaque "expression" les codes des instructions qui l'utilisent et on ajoute SYSTEMATIQUEMENT une

paramétrisation par la valeur du Code-Opération (contenu de RI) afin de faciliter les FUSIONS, sachant que l'optimisation logique pourra être faite à la fin de l'analyse pour faire éventuellement disparaître les paramètres redondants.

Exemple: si l'on trouve, à la fin de l'analyse:

```
busa:= if(C1 et (RI=V1 ou RI=V2 ou RI=V3))
       then SOURCE1 else SOURCE2 endif;
busa:= SOURCE1;
busa:= SOURCE2;
```

on peut simplifier le premier cas en:

```
busa:=if C1 then SOURCE1 else SOURCE2
```

Autre exemple: soit l'équation

```
busa:=if (C1 et (RI=V1 ou RI=V2))
         ou (C2 et (RI=V1 ou RI=V3)) then...
```

on peut simplifier en faisant disparaître (RI=V1):

```
busa:=if (C1 et RI=V2) ou (C2 et RI=V3) then...
```

CONCLUSION

Cet algorithme permet d'ajouter une paramétrisation, d'une manière automatique et systématique, ce qui a pour effet de réduire le nombre de codes pour chaque champ, en augmentant le nombre de monômes des fonctions de calcul (Ei diminue, Mi augmente). Il faudra se souvenir qu'on peut augmenter Mi en respectant la contrainte globale:

$$Mi <= (K-1)*N$$

et seul un COMPILATEUR peut faire cette vérification.

II- MACSIM: outil de description et de SIMulation de MAChines

INTRODUCTION

La définition de l'outil MACSIM répond à un double but: proposer un moyen simple de description d'une machine d'état fini du type Unité Centrale, pour en réaliser la simulation à un niveau RTL (Register-Transfer-Language = Langage de Transfert de Registres), et extraire statiquement de la description elle-même les caractéristiques du chemin de données et celles de la partie contrôle.
Le second but est de préciser le cahier des charges du système à réaliser autour du langage IRENE: MACSIM est en fait un prototype du système SIRENE, qui utilise un META-PASCAL et le compilateur PASCAL, alors qu'IRENE aura ses propres compilateurs pour la simulation et l'analyse statique.
MACSIM est donc seulement une étape de la recherche.

II-A- PRINCIPES GENERAUX

Tout langage de description et/ou de simulation amène un certain nombre de questions: que cherche-t-on à vérifier par simulation? à quel niveau de finesse doit-on décrire la machine que l'on conçoit? quel sens donne-t-on à une description et quelles informations veut-on en extraire?

On propose ici un principe général basé sur la double signification d'une description: toute description à une signification donnée par le langage de description utilisé, plus une signification donnée par le concepteur aux mécanismes qu'il décrit.
On peut en effet décrire un mécanisme unique de plusieurs manières, en obtenant des résultats de simulation identiques. La différence

entre deux descriptions, même fonctionnellement équivalentes, sera donc déterminée par l'interprétation automatique ou subjective qui en sera faite, soit par des outils d'analyse statique, soit par le concepteur.

Il est donc souhaitable d'une part de disposer de souplesse au niveau du langage de description, d'autre part de rigueur au niveau de son utilisation. Le langage doit permettre de décrire le même mécanisme avec différents niveaux de finesse, mais chaque forme décrite doit avoir une signification précise.

Exemple:
les 2 descriptions "A:=B;" et "bus:=B; A:=bus;" sont fonctionnellement équivallente, mais la 2ème est plus "riche" en ce sens qu'elle contient plus d'informations sur la réalisation matérielle choisie pour transférer la valeur de "B" dans la variable "A".

Le principe de MACSIM est l'utilisation d'un langage algorithmique puissant et répandu, le langage PASCAL, pour décrire et simuler le comportement d'une machine informatique.
Le concepteur de la machine utilise la syntaxe de PASCAL pour décrire des opérations du type "transfert de registres", et une META-SYNTAXE pour décrire les caractéristiques opérationnelles ou algorithmiques qui ne sont pas contenues dans PASCAL, afin qu'elles puissent être reconnues par un outil d'analyse STATIQUE, c'est-à-dire un autre compilateur que celui de PASCAL.

II-B- LES TECHNIQUES DE DESCRIPTION

II-B-1- Les étapes élémentaires

On se restreint à la description de machines d'état fini (qui ont un nombre fini et connu d'états).
Chaque état peut être une instruction-machine, ou un cycle-machine, ou une phase d'horloge élémentaire, selon la "finesse" choisie pour la description.

Chaque étape de traitement correspond à un état de la machine, mais seul le concepteur connaît sa finesse.

Pour chaque étape élémentaire, qui est définie dans la méta-syntaxe par l'occurrence d'un MOT-CLE, le concepteur décrit la liste des actions opératives (instructions d'affectation), qui seront exécutées séquentiellement, parce que telle est la sémantique d'exécution de PASCAL.

Sachant qu'on décrit ce qui doit se passer dans un cycle-machine, ou dans une phase élémentaire, on doit distinguer deux cas, relatifs au parallélisme possible de plusieurs actions:
a/ ou bien les actions réelles sont exécutables en parallélisme collatéral, sans aucune "dépendance" entre elles, et on peut alors les décrire dans n'importe quel ordre,
b/ ou bien l'ordre d'exécution réel est connu, parce qu'il y a par exemple une succession de phases de transferts, et les actions DOIVENT alors être décrites dans cet ordre-là.

Exemple: les 2 actions "bus:=B;" et "A:=bus;" sont réellement exécutées dans cet ordre; il faudra donc les décrire
- soit dans deux étapes successives,
- soit dans la même étape mais dans cet ordre.

Remarque: la description précédente fait apparaître la variable "bus" comme un élément de mémorisation, ce qui n'est pas toujours un "bon" modèle pour un bus, si l'on se place au niveau fin du matériel. Cependant, cette modèlisation n'introduit pas d'erreur fonctionnelle: c'est un modèle imprécis mais fonctionnellement acceptable, sachant que l'implémentation matérielle du "bus" doit être étudiée à un autre niveau, et avec d'autres outils (simulateur électrique), en assurant une conservation de la fonctionnalité entre les différents niveaux de description.

Le découpage en cycles de traitement permet de distinguer les variables de mémorisation à caractère permanent (registres et

latches), et les variables instantannées qui ne conservent pas leur valeur d'un cycle au suivant: ces dernières peuvent être systématiquement chargées avec une valeur "indéfinie" au début de chaque cycle. On peut ainsi vérifier qu'une valeur instantannée qui est "lue" dans un cycle a bien reçu une valeur non "indéfinie".
On pourrait aussi simuler des éléments de mémorisation DYNAMIQUES en définissant que leur valeur reste "définie" pendant un certain nombre de cycles élémentaires.

II-B-2/ UTILISATION DE PASCAL

a/ Implantation des variables

Une machine du type Unité Centrale comporte des registres ou latches, des bascules et des circuits combinatoires.

- Un registre peut être implanté dans une variable de type "integer" portant le même nom, si le type "integer" du PASCAL utilisé est assez grand.
- Une bascule peut être implantée dans une variable de type "boolean" portant le même nom.
- Une Mémoire Centrale peut être soit un tableau d'entiers, soit un fichier à accès direct si elle est trop grande pour être implantée dans l'espace des variables du programme PASCAL.
- Les circuits combinatoires peuvent être décrits par des "function"s de PASCAL, éventuellement paramétrées, et retournant un résultat unique de type "integer" ou "boolean", ou plusieurs résultats dans des variables globales.
  On se souviendra qu'à l'inverse d'un circuit combinatoire, une "function" de PASCAL ne calcule sa valeur que lorsqu'elle est appelée (et non pas en permanence), et qu'elle évalue ses paramètres d'entrée à ce moment-là.

b/ Equivallence ENTIER-BOOLEEN-BIT

L'extraction ou la modification des bits d'un mot n'est pas directement possible en PASCAL, alors qu'elle l'est en IRENE. On

utilisera la propriété d'équivallence de différents champs d'un "record" muni d'un aiguillage ("case"). Cette équivallence dépend bien sûr du PASCAL utilisé. Dans le cas du PASCAL UCSD sur une machine à mots de 16 bits, on a les équivallences suivantes:

```
type EQUIV16 = record case integer of
                 0: ( VINT : integer );
                 1: ( VSET : set of 0..15 );
                 2: ( VBOO : packed array[0..15] of boolean );
                 3: ( VT4  : packed array[0..3] of 0..15);
                 4: ( VT8  : packed array[0..1] of 0..255);
                 5: ( VCH  : packed array[0..1] of char)
               end;
```

La valeur d'un mot-mémoire peut ainsi être considérée soit comme un entier, soit comme un ensemble de 16 éléments binaires (bits), soit comme un tableau de 16 booléens ("false"=0, "true"=1).

Dans le cas où la machine à simuler possède des mots de plus de 16 bits, on peut utiliser une équivallence sur 32 bits, selon la déclaration suivante:

```
type EQUIV32 = record case integer of
                 0: ( VREA : real );
                 1: ( VSET : set of 0..31 );
                 2: ( VBOO : packed array[0..31] of boolean );
                 3: ( VIH, VIL : integer );
                 4: ( VT4  : packed array[0..7] of 0..15 );
                 5: ( VT8  : packed array[0..3] of 0..255);
                 6: ( VCH  : packed array[0..3] of char)
               end;
```

b/ PROGRAMMATION DES FONCTIONS

Accès aux bits d'un mot:

On déclare:

```
BIT : boolean;        N : integer;        EQU : EQUIV16;
```

La lecture du bit numéro "j" de la variable entière N est décrite par:

```
EQU.VINT:=N;    BIT:= ( j in EQU.VSET );
```

Le forçage du bit numéro "j" avec la valeur de BIT est décrit par:

```
EQU.VINT:=N;
if BIT then EQU.VSET := EQU.VSET + [j]
       else EQU.VSET := EQU.VSET - [j];
N:=EQU.VINT;
```

Exemple du calcul des flags d'une UAL 16 bits:

Après une addition réalisée par "RESUL:=A+B;" on peut calculer:

```
        Zual := ( RESUL = 0 );  (* flag Zero *)
        Nual := ( RESUL < 0 );  (* flag Negatif *)

        (* flag oVerflow = Vual *)
    if (A>0) and (B>0) then Vual := ( 32767-A  < B );
    if (A<0) and (B<0) then Vual := ( -32767-A > B+1 );
```

Exemple de description de l'addition dans une UAL 8 bits

Les 2 opérandes sont compris entre 0 et 255 pour PASCAL, bien qu'ils représentent des entiers signés compris entre -128 et +127.

```
    RESUL := (A+B) mod 256;

        Zual := ( RESUL = 0 );
        Nual := ( RESUL > 127 );
        Cual := ( A+B > 255 );

   if (A<=127) and (B<=127) then Vual := ( RESUL > 127 );
   if (A>127)  and (B>127)  then Vual := ( RESUL <= 127 );
```

Concaténation de deux variables

On a souvent besoin de considérer la concaténation des bits de 2 variables (ou plus) comme une valeur entière; si WL est une variable de 8 bits, et WH une variable de 4 bits, on peut calculer:

```
I := 256*WH + WL;
```

et la variable I peut être utilisée pour adresser un tableau, par exemple.

Extraction de chaînes de bits

Les i bits de poids faible d'un entier sont donnés par la fonction MODULO 2^i.

Les bits de poids fort, à partir du rang j, par la fonction DIV (division entière) par 2^j.

En combinant les deux opérations DIV et MOD, on obtient les k bits allant du rang i au rang j par:

$((V \text{ DIV } 2^i) \text{ MOD } 2^{j-i})$

II-C- INTERFACE DE DIALOGUE POUR LA SIMULATION

II-C-1/ Marquage des cycles de simulation

Pour effectuer la mise au point d'une description fonctionnelle, le concepteur a besoin de connaître la progression des traitements, afin de vérifier que cette progression est correcte d'une part, et afin de savoir en quel point précis se trouvent d'éventuelles erreurs. Il programme en MACSIM l'appel d'une procédure CYCLE(n:integer), au début de chacune des étapes qu'il considère comme étant des pas de simulation. Le contrôle lui sera alors donné (arrêt de la simulation), et il pourra obtenir un état courant des variables de la machine simulée, et éventuellement modifier des valeurs de variables.

L'affichage de la valeur du paramètre "n" de la procédure CYCLE permet de connaître le numéro de l'étape qui est la prochaine à simuler.

II-C-2/ Interface de dialogue

Le dialogue entre le concepteur et le simulateur se fait en donnant le nom symbolique des variables de la machine: ce nom est le même

que celui qui est déclaré en PASCAL. Une table de correspondance "nom-alphanumérique"<->"nom-interne" doit pour cela être remplie à la génération du système.

D'autre part, les variables dont on veut surveiller les modifications doivent être doublées: on déclare "A" et "A%", de même type, et le noyau de simulation teste la modification de "A", à chaque pas de simulation, par:

```
if (A<>A%) then begin A%:=A; write('A=',A) end;
```

ce qui permet de suivre l'évolution de toutes les variables "surveillées".

II-D- ANALYSE STATIQUE D'UNE DESCRIPTION

Une description a une sémantique dynamique si elle est exécutée pour simuler la machine décrite; elle a également une sémantique statique que l'on peut mettre en évidence par l'analyse statique de sa structure, pour une méta-syntaxe donnée. Cette sémantique donne les caractéristiques statiques de la machine décrite: toutes les actions décrites, ainsi que le contrôle de leur séquencement, y sont exprimés et sont en bijection avec une structure d'accueil qui les réalise.

On extrait donc, statiquement, par une compilation du texte constituant la description, deux types de renseignements, selon deux classes de fonctions:
- les fonctions opératives (actions),
- les fonctions de contrôle (séquencement), que l'on encode pour les exploiter ensuite.

Une telle analyse statique ne vaut la peine d'être faite que

lorsque la description a un comportement correct, conforme au cahier des charges.

II-D-1/ Définition de mot-clés

Comme on l'a déjà écrit, il n'y a pas d'analyseur pour la méta-syntaxe, au sens de l'analyse syntaxique. Ce ne sera pas le cas pour le système IRENE, dans lequel la forme syntaxique elle-même est suffisamment riche pour faire apparaître les propriétés statiques.
On définit ici deux classes de mot-clés, chaque mot-clé étant une chaîne limitée de caractères, et l'analyse sera faite par recherche de ces mot-clés dans le texte.

a/ mot-clés de séquencement:
Ce sont des chaînes de caractères dont l'occurrence définit la structure de contrôle de l'algorithme, soit au niveau le plus fin des cycles d'exécution (chaque cycle-machine ou chaque état de l'automate), soit au niveau de l'enchaînement des cycles ou des états, avec des structurations éventuelles (branchements conditionnels, boucles, appels de sous-programmes, ...).
Il faut définir au minimum un mot-clé qui indique le début du cycle le plus fin, et ce peut être tout simplement le nom de la procedure CYCLE qui est appelée au début de chaque cycle pendant la simulation.

b/ mot-clés opératifs:
Chaque mot-clé opératif correspond à la commande d'une action opérative, ou à un champ pour une commande codée.
Exemple: si "busa:=" est déclaré comme mot-clé, l'analyse statique fera apparaître tous les opérandes trouvés entre ce mot-clé et le ";" suivant.
le texte "busa:=SUAL;" donnera "SUAL" comme opérande de "busa:=".

II-D-2/ PRINCIPE DE L'ANALYSE STATIQUE

Avant d'entreprendre une recherche des mot-clés, on construit une

grande chaîne de caractères contenant toutes les lignes de description comprises entre deux mot-clés de début de cycle.
Le compilateur d'IRENE fera ce travail facilement en parcourant une règle de sa syntaxe:
c'est la règle ‹liste d'instructions collatérales›.
On obtient ainsi le texte complet de la description d'un cycle, et on recherche alors dans ce texte la présence des mot-clés opératifs et la valeur de leur opérande.

Le programme d'analyse donne comme résultat un encodage de chaque description de cycle, que l'on peut appeler une microinstruction automatiquement codée par champs:
- à chaque mot-clé est alloué un champ,
- les opérandes sont codés à partir de la valeur 1, dans l'ordre où ils apparaissent.

Si un mot-clé opératif n'est pas trouvé, on affecte au champ correspondant la valeur 0.
Après la recherche des mot-clés, tous les champs ont une valeur, et la microinstruction est codée sous la forme d'une chaîne de "n" caractères, si "n" champs ont été définis.

a/ Recherche séquentielle:
La recherche des mot-clés est faite de gauche à droite, donc séquentiellement dans la chaîne: on ne trouve ainsi que la première occurrence d'un mot-clé.
Ceci impose des restrictions sur la valeur des chaînes-mot-clés: elles doivent avoir des terminaisons différentes, ou, si elles se terminent semblablement, elles doivent apparaître dans un ordre précis.
Exemple: soit les mot-clés "busa:=" et "a:=";
si l'instruction "a:=SUAL;" est écrite avant l'instruction "busa:=ram[3];" on trouvera "SUAL" comme comme opérande de "a:=". Dans le cas contraire, on trouvera "ram[3]" comme opérande de "a:=", ce qui est faux.

On adoptera donc une méta-règle pour le choix des noms des mot-clés:

"les identificateurs qui sont les noms des mot-clés ont des terminaisons différentes".

b/ Choix de la finesse de description:
Selon que l'on souhaite faire apparaître des commandes plus ou moins fines, on adoptera des styles d'écriture différents.

Exemple: pour faire apparaître le champ "adresse de RAM", on définit une variable ADRAM, à laquelle on affecte un "numéro de registre", sous la forme "ADRAM:=numéro", et on référence la RAM d'une manière unique par "busa:=RAM[ADRAM]".
Ce style d'écriture permet d'analyser le mot-clé "ADRAM:=", et d'en déduire l'existence d'un décodeur qui décode tous les "numéros" trouvés comme opérande de "ADRAM:=".

Un autre style consiste à écrire "busa:=RAM[numéro];". On trouvera alors autant d'opérandes différents pour le mot-clé "busa:=", qu'il y a de "numéros" différents, et on ne fera pas apparaître dans ce cas l'entrée d'un décodeur, mais plutôt sa sortie.

Chaque style d'écriture permet donc de reconnaître certains mot-clés, et est en bijection avec un certain type de réalisation de l'interface de commande.

c/ Levée des ambiguités apportées par PASCAL:
Toute description doit être exécutable par PASCAL. Il faut donc tout décrire en PASCAL, avec sa syntaxe. Dans le cas où cette syntaxe est unique pour exprimer deux mécanismes sémantiquement différents, il y a ambiguité:
une analyse statique ne permettra pas de distinguer ces deux mécanismes.
On propose donc de lever l'ambiguité en utilisant la syntaxe d'IRENE, pour spécifier, sous forme de commentaires PASCAL, les différentes formes de description.

La principale ambiguité syntaxique est apportée par le "if ... then ... else ...", qui a une forme unique en PASCAL pour exprimer :

- soit les actions conditionnelles:

    "if COND then A:=exp1 else A:=exp2;"

- soit les branchements conditionnels:

    "if COND then goto L1 else goto L2;"

On lèvera l'ambiguité en utilisant, entre commentaires, les formes IRENE:

    "(* A:=if COND then exp1 else exp2 endif; *)"

    "(* goto if COND then L1 else L2 endif; *)"

et on pourra ainsi reconnaître que:

- "if COND then exp1 else exp2" est un opérande de "A:=",
- "if COND then L1 else L2" est un opérande de "goto".

II-E- CONCLUSION SUR MACSIM

La définition des principes de MACSIM a été faite pour répondre à un double besoin:

- un besoin de simulation fonctionnelle pour valider la conception des algorithmes de contrôle au cours de la définition d'un ordinateur;
- un besoin de compilation de cette description pour extraire les caractéristiques du matériel qui doit permettre de réaliser l'ensemble des opérations décrites.

MACSIM n'est qu'un prototype, d'ailleurs opérationnel et utilisé pour décrire 2 processeurs (le 80C48 [Ref. BERTRAND] et le P-CODE [Ref. GALLAIS]), et sa mise en oeuvre a permis de définir plus précisément le cahier des charges des outils logiciels à construire autour du langage IRENE.

CHAPITRE 3

IMPLANTATION

DES PARTIES CONTROLE

UN EXEMPLE DE COMPILATION

I - STRUCTURE D'ACCEUIL =
"LE GENERATEUR D'INSTANTS"

II- COMPILATEUR PROTOTYPE =
"MACSIM"

I N T R O D U C T I O N

Ce chapitre présente un exemple de compilation d'une Partie Contrôle: une "structure d'accueil", constituée par le modèle du "générateur d'instants" présenté dans [REF. OB-82] et implanté dans plusieurs microprocesseurs, en particulier dans le INTEL 8048 dont l'algorithme d'interprétation est spécialement adapté (toutes les instructions s'exécutent soit en 10 instants soit en 20 instants).

La méthode suivie pour la définition de ce "compilateur" est conforme à l'approche globale présentée dans cette thèse: une structure d'accueil est inventée pour accueillir les blocs fonctionnels et réaliser les mécanismes de traitement. Elle est ensuite étudiée en termes de topologie, en tenant compte des contraintes globales.

Lorsque cette structure a été domestiquée et quand ses éléments sont définis, il est possible de mettre en oeuvre un compilateur qui extrait les caractéristiques de l'algorithme à implanter pour les projeter dans les éléments fonctionnels de la structure d'accueil.

Ainsi, pour cette structure, il suffit de réaliser un compilateur qui sait reconnaître les différents "instants" ou "cycles" d'exécution d'une instruction, puis d'associer les informations (valeurs des commandes, valeur d'un instant, valeur du Code-Opération de l'instruction) entre-elles.

On ne présentera ici que la génération des commandes pour une machine de type MOORE, sachant que la couche de paramétrisation (pour une machine de type MEALY) est traitée de la même manière pour toutes les structures.

I- LE PRINCIPE DU GENERATEUR D'INSTANTS

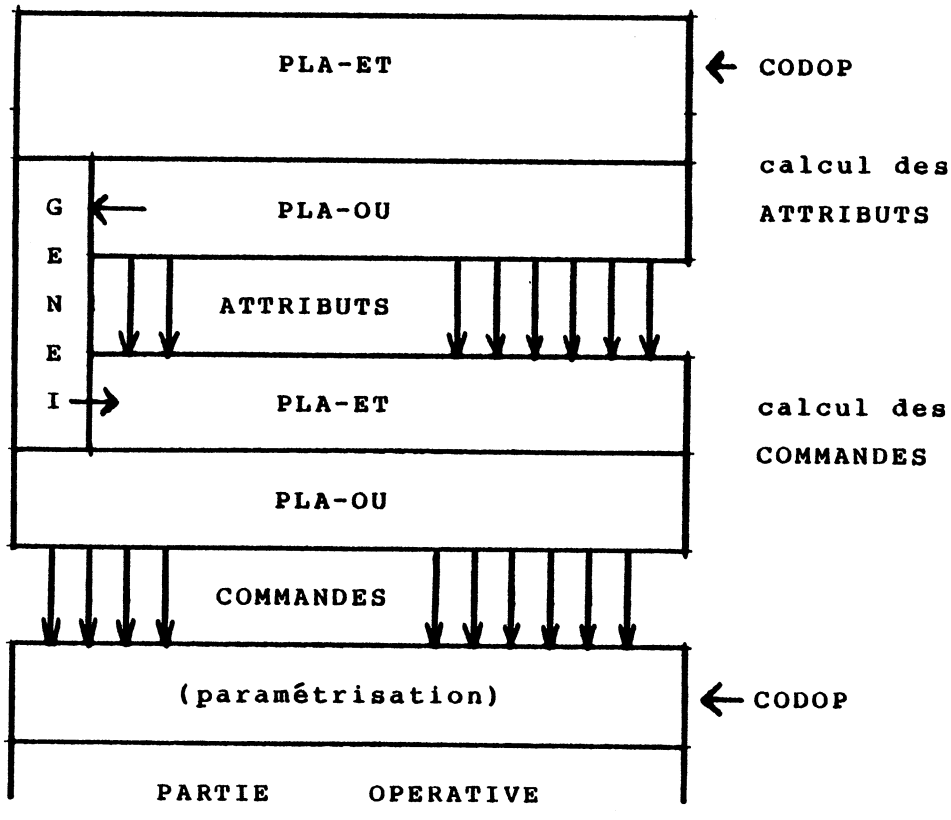
Une instruction-machine requiert un nombre variable de cycles, appelés "instants", pour son interprétation. Ces "instants" peuvent être numérotés, par exemple à partir de Zéro, et la progression du code de l'instant "contrôle" le déroulement de l'exécution de l'instruction, jusqu'à l'instant final.

Le principe consiste à calculer les commandes en fonction:

a/ du code de l'instant courant,

b/ des "attributs" caractéristiques de l'instruction, que l'on peut aussi appeler des "propriétés", que nous définirons plus loin. Toutes les instructions ne requièrent pas forcément le même nombre d'instants pour s'exécuter, et ce nombre doit également faire partie des attributs de chaque instruction (ou famille d'instructions en cas de paramétrisation).

Le schéma global est donc:



I-A- DEFINITION D'UN ATTRIBUT:

"Un groupe d'instructions possède l'attribut Ai si un ensemble de commandes est activé à l'instant Tj pour toutes les instructions de ce groupe. Tout attribut peut ainsi être codé par un nombre binaire, et porté par un fil unique."

Synthèse des commandes

Une commande est active lorsque la somme de produits d'un instant par un attribut est vraie.

Exemple: la commande

"busa:=SOURCE1" est égale à:

(A1 et T1) ou (A5 et T4) ou (A8 et T6) ou (A80 et T13)

Elle est activée:

à l'instant T1 pour les instructions qui ont l'attribut A1,
" T4 " A5,
" T6 " A8,
" T13 " A80.

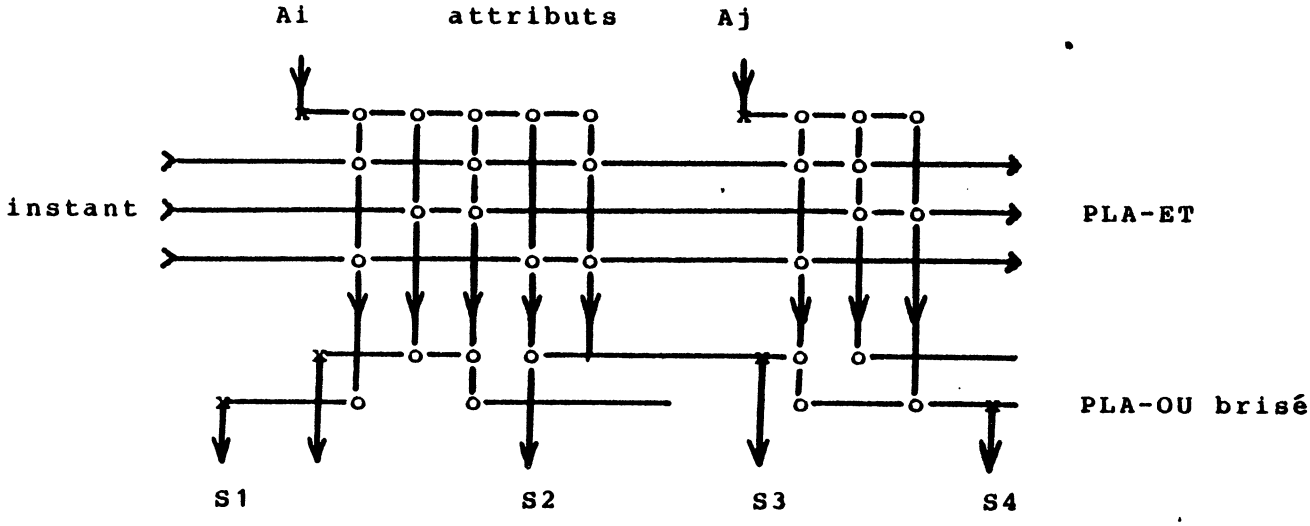
Remarques:

- une instruction a généralement plusieurs attributs,
- si toutes les instructions sont différentes à chaque temps, il y a autant d'attributs que d'instructions.

I-B- OPTIMISATION TOPOLOGIQUE

Le code de l'instant courant doit être combiné avec tous les attributs, pour donner un monôme de génération: on peut optimiser la matrice ET qui calcule les combinaisons du code de l'instant

avec tous les attributs en entrant les "fils" qui portent ces derniers "latéralement", c'est-à-dire "verticalement", et on peut optimiser la matrice OU qui calcule la synthèse des commandes en brisant ses lignes de sortie:



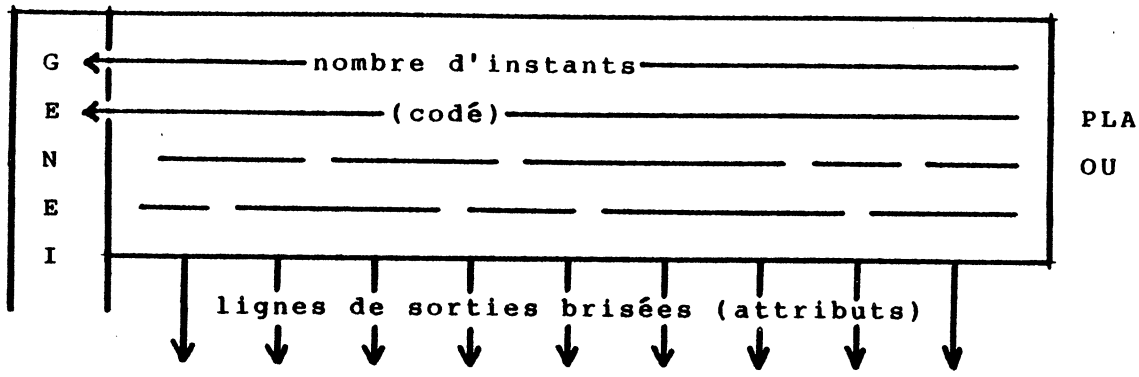
La matrice ET est automatiquement optimisée en hauteur: sa hauteur est donnée par le nombre de bits qui codent l'instant PLUS une ligne pour introduire un attribut.
Sa largeur est égale au nombre de MONOMES M1 à calculer, plus un passage pour entrer latéralement chaque attribut.

Sa largeur est donc: M1 + NA
si M1 est le nombre de monômes et NA le nombre d'attributs.

Les entrées des attributs Ai dans la matrice ET ne sont pas régulièrement espacées: leur espacement dépend du nombre de monômes qui font intervenir les fils Ai.
On a donc intérêt à réaliser le calcul des attributs par une matrice OU à sorties brisées, et à étirer le PLA de attributs jusqu'à la largeur du PLA de synthèse.

Remarque: on ne pourra vraisemblablement pas briser les lignes de calcul du nombre d'instants caractéristiques de chaque instruction, mais comme leur nombre est petit, elles occuperont toute la largeur

du circuit pour entrer directement dans le Générateur d'instants (GENEI).



I-C- LE GENERATEUR D'INSTANTS:

On cherche à l'implanter verticalement, à coté du PLA-OU de synthèse des attributs; il reçoit:
- des horloges marquant le cycle machine,
- une valeur indiquant le nombre d'instants pour 1 instruction,
- un signal de mise à zero,

et il fournit le code de l'instant suivant.

OPTIONS: si on numérote les instants à partir de 0, et on peut choisir:
- un CODAGE BINAIRE:

  n bits => 2*n fils => 2^{n+} instants
- un CODAGE 1 parmi p:

  p bits => p fils => p instants

I-C-1/ Le codage binaire:

On envisage ici le cas le plus général où les instructions peuvent nécessiter un nombre variable d'instants: on associe à chaque instruction un nombre maximum MAXi d'instants fourni par le PLA de synthèse des attributs.

On définit un compteur d'instants N qui est soit incrémenté, soit

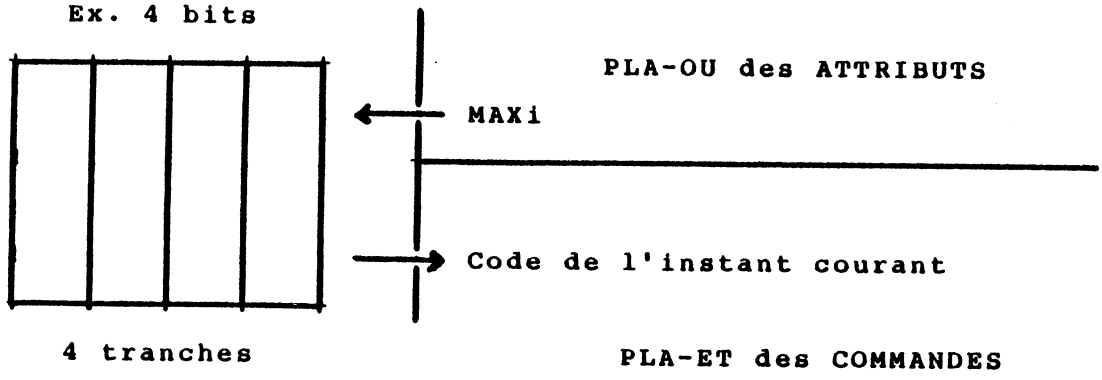
forcé à '0' après qu'il soit devenu égal à MAXi.

I-C-2/ STRUCTURE pseudo-dynamique

Un registre semi-statique peut être rafraîchi sur une phase (PHI1), et chargé, sur l'autre phase (PHI2), soit avec la sortie d'un incrémenteur, soit avec la valeur '0', dans le cas où sa valeur précédente était égale à MAXi:

ce principe requiert une petite Partie Opérative composée d'un opérateur d'incrémentation, avec ses 2 tampons d'entrée et de sortie, et un comparateur (OU exclusif).

Cette Partie Opérative peut être implantée dans des "tranches de bits" placées VERTICALEMENT, sous les lignes virtuelles définies dans le chapitre qui traite de l'approche topologique.



CONSEQUENCE: la valeur de l'attribut MAXi doit changer pendant la phase contrôlée par PHI2, et à un instant inférieur à sa valeur minimale (si le minimum des valeurs MAXi vaut 5, on a jusqu'à l'instant Numéro 5 pour le changer).

REMARQUE 1: Un signal RESET doit permettre de forcer le démarrage à l'instant 0, mais avec quelle instruction? Il faut définir cette dernière (INSTRUCTION RESET) avec des attributs qui correspondent à la séquence d'initialisation.

REMARQUE 2: Quand on passe d'une instruction à la suivante, les attributs de l'instruction suivante ne sont pas disponibles

instantanément:

- il faut donc soit changer d'instruction suffisamment tôt dans un cycle pour que les nouveaux attributs soient calculés,
  => (1/2 cycle pour le calcul)
- soit définir des attributs initiaux communs à toutes les instructions: par exemple toutes les instructions ont l'attribut A0 qui définit la même séquence pour toutes les instructions et pour, par exemple, les 4 premiers instants. On dispose ainsi de 4 cycles pour calculer les attributs de l'instruction courante:
  => le PLA des attributs peut dans ce cas être très lent,
  => une taille minimale peut être choisie pour ses transistors.

I-D- LE SEQUENCEMENT CONDITIONNEL:

Avec la notion de générateur d'instant, qui est en fait équivalente à celle de compteur ordinal relatif au début d'une séquence, on peut appliquer toutes les techniques des branchements conditionnels, en avant ou en arrière: il faut alors réaliser un SEQUENCEUR selon le principe du séquenceur de microinstructions , avec quelques restrictions:

- la notion de sous-programme existe déjà dans celle d'attributs: si deux sous-programmes sont "appelés" à l'instant Ti et s'exécutent jusqu'à l'instant Tj par deux instructions, ces dernières ont les mêmes attributs entre les deux instants Ti et Tj: elles s'exécutent d'une manière identique entre ces deux instants.
- si toutes les instructions exécutent la même microinstruction au même "instant", elles ont l'attribut le plus général (A0).
- le branchement conditionnel vers l'avant permet de sauter des instants et donc de raccourcir l'exécution d'une instruction.
- le branchement conditionnel vers l'arrière permet de revenir quelques instants en arrière, et donc de réaliser des boucles.

L'adresse relative de branchement ne peut être déduite que du code de l'instruction: on rajoute donc des attributs caractéristiques de

chaque instruction, donc des lignes horizontales dans la matrice OU, augmentant ainsi la hauteur de la PC. On doit d'autre part réaliser un mécanisme de test d'une condition qui nécessite un numéro de condition, et le numéro de l'instant où doit être effectué le test: toutes ces informations sont également des attributs de l'instruction en cours.

La matrice OU de synthèse des attributs augmente ainsi à chaque fois qu'un mécanisme NON SYSTEMATIQUE est ajouté:

=> cette ARCHITECTURE n'est COMPACTE que POUR UN SEQUENCEMENT SYSTEMATIQUE

Exemple: Chaque instruction de 8048 s'exécute en 10 ou 20 instants et il n'y a pas de séquencement conditionnel.

II - COMPILATION D'UNE DESCRIPTION ALGORITHMIQUE

Il suffit d'un mot-clé ('FININST') pour reconnaître la fin d'une séquence décrivant une instruction, si, par convention, le premier cycle décrit la première microinstruction, c'est-à-dire le premier instant, de la première instruction.

II-A- LA META-SYNTAXE MACSIM:

La réalisation des FUSIONS des commandes paramétrées et la détermination des ATTRIBUTS requièrent la connaissance du ou des code-opération(s) de l'instruction (ou du groupe d'instructions) en cours: on définit donc un autre mot-clé ('CODOP') qui doit être trouvé dans le premier cycle de chaque instruction.

La méta-syntaxe MACSIM est:

```
CYCLE(N1); (* CODOP=...; *)
           description du premier cycle;
CYCLE(N2); description du deuxième cycle;
  "              "
  "              "
CYCLE(Np); description du dernier cycle ;
           FININST ;
```

L'opérande du mot-clé 'CODOP' est une chaîne de caractères contenant des '0', des '1' ou des 'x'. Si une seule expression ne suffit pas, on utilise d'autres mots-clés de prolongation CODOP2, CODOP3, CODOP4,...

La forme générale aurait pu être une liste de chaînes, par exemple:

```
CODOP=(1110xxxx, 101101xx);
```

II-B- **ANALYSE DE CHAQUE CYCLE:**

Entre deux mots-clés 'CYCLE' on pratique une recherche systématique de tous les mots-clés distinctifs des CHAMPS de commandes. Un analyseur syntaxique (celui d'IRENE par exemple) pourra suivre la syntaxe, reconnaître les identificateurs en position de mot-clé et chercher dans une table.

N'ayant pas encore cet analyseur, nous effectuons une recherche systématique d'une SOUS-CHAINE dans une CHAINE (fonction POS de PASCAL-UCSD) qui indique la présence ou l'absence de cette sous-chaîne dans la microinstruction.

Si la sous-chaîne 'MOT-CLE' existe, son opérande est trouvé entre elle-même et le prochain ';'. Son opérande est donc une chaîne de caractères qui est à son tour analysée par recherche de 'if' ou de 'case' pour savoir si la commande courante est paramétrée (voir FUSIONS, chapître MACSIM).

La "valeur" de l'opérande est comparée avec la liste des "valeurs" déjà rencontrées pour ce champ. Cette liste est gérée dynamiquement dans le TAS de PASCAL, par utilisation de pointeurs et d'une allocation dynamique (NEW(p)).

On peut ainsi affecter un CODE à chaque CHAMP de la microinstruction; le CODE 'INACTIF' est affecté au champ si le mot-clé qui le caractérise n'est pas trouvé dans le cycle.

II-C- **MEMORISATION DES RESULTATS:**

Si NC est le nombre des champs, chaque cycle est décrit par une chaîne de NC caractères.

La valeur 'INACTIVE' est codée '.', les valeurs rencontrées sont codées de '1' à '9' , puis de 'A' à 'Z', dans l'ordre où elles sont rencontrées.

On obtient ainsi une forme codée textuelle pour chaque cycle, et

pour chaque instruction (suite d'instants).

* commentaire = NOMS des instructions-machine
<NOMBRE DE CODOP> <CODOP1> <CODOP2> ...
<CYCLE1> = chaîne de NC caractères .
"
"
<CYCLEn>= " "

Remarque: cette forme codée est beaucoup plus dense que la forme originale (PASCAL-MACSIM) et elle peut être utilisée comme forme descriptive d'un SIMULATEUR qui l'interprète (voir plus loin).

II-D- **DEUXIEME FORME DE L'INTERPRETEUR:**
Le fichier résultat précédent (textuel) est mis en mémoire: chaque microinstruction occupe NC octets (pour 100 instructions ayant en moyenne 15 cycles, il faut une mémoire de 100x15xNC octets, soit 32 Koctets si NC = 20).

On définit d'autre part à partir des codes-opération une table des points d'entrée (qui associe à la valeur d'un CODOP une entrée dans la table) et une table des longueurs qui donne le nombre de cycles de chaque instruction.

TABLE des points d'entrée = 256x2 = 512 octets (pour 8 bits)
TABLE des longueurs = 256x1 = 256 octets (pour 8 bits)

Déclarations

```
const NMAX = nombre total de cycles décrits
      NC   = nombre de champs
      NSMAX = nombre de cycles max (de la séquence la plus longue)
var
        TM = packed array [0...NMAX] of string [NC];
        TE = packed array [0...255] of integer;
        TL = packed array [0...255] of 0..NSMAX;
```

Initialisation

En parcourant le fichier textuel, on remplit TE (table des points

d'entrée), TM (table des microinstructions), et TL (table des longueurs de séquence).

Exécution

Pour interprèter l'instruction dont le CODOP est contenu dans RI, on obtient:

- le point d'entrée dans TE[RI] que l'on met dams une variable i,
- on accède aux TL[RI] microinstructions qui se suivent à partir de la position i.

Interprétation

On charge la microinstruction dans MUI:

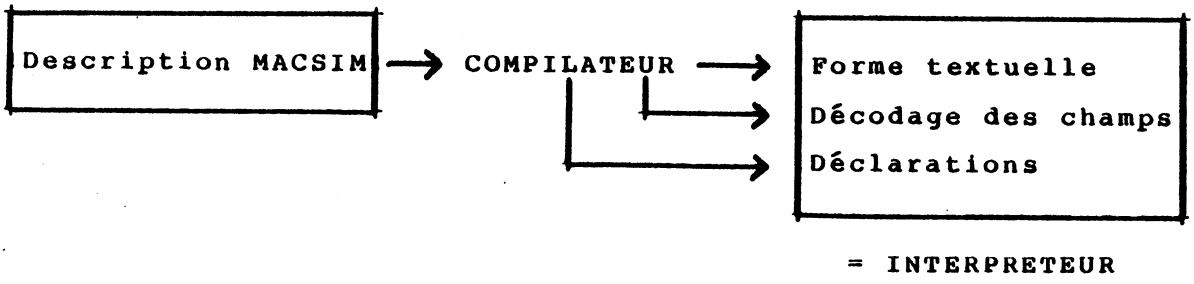
MUI:=TM[i]

puis on interprète les champs entre MUI[1] et MUI[NC]. Chaque champ contient une valeur codée, soit '.' pour la valeur INACTIVE, soit une valeur allant de '1' à '9' puis de 'A' à 'Z'.

Il faut donc, pour chaque champ j exécuter un décodage de la forme:

```
case MUI[j]
   '1' : <action1>;
   '2' : <action2>;
    "
end;
```

ce qui est justement la forme de l'"interpréteur" généré lors du premier traitement.



AVANTAGES DE LA METHODE

-> la compilation d'une description fonctionnelle de haut niveau pour une machine complexe est une opération longue et coûteuse. Une petite modification dans un seul cycle d'une seule instruction oblige à recompiler tout un module.

-> la solution proposée permet d'avoir, sous forme textuelle, une représentation compacte, et somme toute lisible, de chaque instruction, dans laquelle une modification est aisée: il suffit alors de "recharger" le simulateur-interpréteur avec la description textuelle compacte modifiée.

-> une solution interactive est également possible: la description étant "en mémoire", il est possible de modifier dynamiquement la description d'une instruction. Il faut pour cela:

- avoir accès aux tables internes par l'intermédiaire d'un éditeur conversationnel,
- pouvoir sauvegarder sur un nouveau fichier le nouveau contenu des tables internes.

GENERATION automatique d'un SIMULATEUR fonctionnel

Une description MACSIM est avant tout un programme écrit en PASCAL: la pauvreté (relative) des formes syntaxiques de PASCAL est compensée par l'adjonction de formes syntaxiques d'IRENE placées entre COMMENTAIRES.

Une telle description doit être compilable par le compilateur PASCAL, et doit donc contenir toutes les déclarations nécessaires.

On peut "récupérer" ces déclarations de variables ainsi que celles des fonctions ou procédures utilisées pour décrire les actions à effectuer.

Le programme d'analyse fournit, quant à lui, le texte du décodage de chaque champ.

Conclusion

En réunissant:

- les déclarations,
- les descriptions des fonctions,
- les décodages de chaque champ,

on obtient automatiquement un programme PASCAL qui représente une nouvelle forme du même SIMULATEUR, avec cette fois une INTERPRETATION de chaque champ de chaque MICROINSTRUCTION, à la place d'une COMPILATION suivie d'une EXECUTION.

II-D- TRACE DE LA SIMULATION

L'utilité de la simulation, pour la mise au point, réside dans la possibilité d'un contrôle, par l'opérateur, des résultats de chaque cycle, ou de chaque instruction. Il faut donc prévoir un mécanisme:

- d'exécution en pas-à-pas (cycle-à-cycle),
- d'exécution instruction par instruction,
- d'affichage des valeurs des registres qui ont changé,
- de forçage de nouvelles valeurs, en particulier si l'initialisation n'est pas automatique.

a/- En analysant les déclarations de variables, on génère un nouvel ensemble de déclarations en doublant chaque variable.

déclaration initiale	déclaration générée
A:integer;	A,A%:integer;

b/- On génère une procédure de surveillance de l'évolution de chaque variable, qui sera appelée à chaque fin de cycle:

```
procédure SURVEILLE;
begin
     if (A<>A%) then
     begin WRITE('A=',A)  ; A%:=A end;
       x...
end;
```

c/- On génère une procédure de modification de la valeur des variables déclarées:

```
procédure MODIFIE (s:string);
begin
      if s='A' then READLN(A)
      else....
end;
```

d/- On génère une procédure d'examen de la valeur d'une variable:

```
procédure EXAMINE(s:string);
begin
      if S='A' then WRITELN(A)
      else...
end;
```

Remarque: dans le cas des tableaux, la valeur d'un indice doit également être spécifiée et on ne peut pas "surveiller" tous les éléments d'un tableau.

Exemple:

```
if S='MEM' then begin WRITE('INDICE?') ; READ(i);
                      WRITELN(MEM[i]);
                end;
```

III - EXPANSION DES CHAMPS

Le résultat de la phase (II) d'ANALYSE (COMPILATION de l'algorithme écrit en PASCAL-MACSIM ou en IRENE) est un microprogramme horizontal codé:
- pour chaque famille d'instructions définie par un ensemble de codes opérations, on possède une matrice de valeurs des champs pour chaque "instant", sous la forme:

CHAMPS / INSTANTS	CHAMP 1	CHAMP 2	- - - -	CHAMP NC
instant 1	v11	v21		vnc1
instant 2	v12	v22		vnc2
" "				
instant p	v1p	v2p		vncp

L'expansion des champs consiste à faire un découpage VERTICAL, pour fournir les informations suivantes:
"La valeur j du champ i doit être générée
- à l'instant k1 pour telles instructions,
- à l'instant k2 pour telles autres instructions, etc..."

On doit donc parcourir le fichier résultat de l'analyse, dans le sens vertical, et autant de fois qu'il y a de champs.

III-A- PREMIER PARCOURS

Le premier parcours effectué sur le fichier permet de faire l'EXPANSION des codes opération de chaque famille d'instruction: le codage alphanumérique avec des '0', '1' et 'x' est étendu à une représentation ensembliste sur 256 bits (pour un CODOP sur 8 bits).

Cette représentation va permettre de réaliser des UNIONS entre toutes les familles d'instructions qui doivent générer la même valeur d'un champ au même instant.

On profite également de ce premier parcours du fichier pour construire un tableau d'ensembles indiquant si un champ a la valeur par défaut à tous les instants pour une instruction donnée: on évite ainsi d'analyser une instruction qui n'utilise pas ce champ, et de plus, on construit un champ qui encode le nombre d'instants pour chaque instruction.

III-B- PARCOURS DU FICHIER POUR CHAQUE CHAMP

Partant du premier champ, on parcourt toutes les instructions, sauf celles qui sont marquées comme n'utilisant pas ce champ, pour construire autant d'ensembles qu'il y a de valeurs possibles pour ce champ et on obtient un nouveau fichier de la forme suivante:

Numéro du CHAMP, VALEUR DU CHAMP

Numéro d'instant, ENSEMBLE d'instructions
Numéro d'instant, ENSEMBLE d'instructions
"
"

Ce fichier est stocké sous une forme binaire, et chaque ensemble, déclaré comme SET OF 0..255, n'occupe que 32 octets sur disque.

IV - SYNTHESE DES PLAS

Tous les ensembles d'instructions trouvés dans le résultat de l'expansion (phase III), constituent des CARACTERISTIQUES, qui peuvent être numérotées, dans l'ordre où elles apparaissent dans le fichier.

IV-A- Numérotation des CARACTERISTIQUES

Une table des caractéristiques, pouvant contenir des ensembles (SET OF 0..255) est initialisée comme vide.

Pour chaque caractéristique trouvée dans le fichier, on recherche dans la table si elle existe déjà, ou si elle est nouvelle: elle est dans ce cas introduite dans la table. L'indice dans la table est appelé un NUMERO d'ATTRIBUT.
Cette numérotation permet:

1/- D'exprimer chaque valeur d'un champ comme une SOMME DE PRODUITS d'un INSTANT par un ATTRIBUT.

(CHAMPi = vj) = (Tk1.An1)+(Tk2.An2)+...

indique que le champ i doit avoir la valeur j,
à l'instant Tk1 pour les instructions qui ont l'attribut An1,
à l'instant Tk2 pour les instructopns qui ont l'attribut An2,...

2/- De générer la description du PLA des ATTRIBUTS:
En effet, chaque caractéristique étant décrite par un ensemble de 256 bits, la valeur '1' du bit de rang i indique que l'instruction dont le code-opération vaut i possède l'attribut correspondant: il faut donc faire l'expansion des bits à '1' pour en déduire les valeurs des codes opération à décoder.

D'autre part, chaque instruction possède plusieurs attributs (le bit de rang i est à '1' dans plusieurs ensembles caractéristiques) et son code-opération ne doit être décodé qu'une seule fois, donnant naissance à un seul monôme qui active plusieurs sorties

(fils qui portent les attributs).

On renverse donc les données pour associer à chaque code-opération l'ensemble des attributs qu'il possède, de manière à obtenir le résultat sous la forme:

<CODE-OPERATION1> <ENSEMBLE D'ATTRIBUTS1>
<CODE OPERATION2> <ENSEMBLE D'ATTRIBUTS2>
" "

ce qui donne la description de la matrice ET de décodage du Code opération et celle de la matrice OU de génération des attributs.

Exemple:

00010011 01010101.....

L'instruction dont le code-opération vaut '00010011' possède toutes les propriétés impaires (1,3,5,...).

IV-B- OPTIMISATION LOGIQUE

Au cours du traitement, on a réalisé l'expansion de la forme contenant des '0', '1' et 'x' vers la forme ensembliste, afin de réaliser des unions, et on a ainsi perdu la connaissance des familles d'instructions décrites avec des 'x'.

Les retrouver consiste à classer les codes-opérations qui produisent les mêmes sorties, puis à rechercher des 'x' dans les classes obtenues: on réalise ainsi une optimisation logique localisée aux classes d'instructions, qui n'a rien à voir avec une optimisation logique globale, mais permet au moins de retrouver les familles données au départ.

Algorithme d'optimisation pour une classe

L'optimisation, dans une classe, est possible si deux profils d'entrée ne sont différents que par 1 bits,
ou si 4 profils ne sont distincts que par 2 bits,

ou si 8 profils ne sont distincts que par 3 bits.
Il faut donc que 2^n profils existent dans la classe, distincts de n mêmes bits , afin d'introduire n 'x' et gagner ainsi (2^n-1) monômes.

Soit une classe de N profils (au départ).

Comparaisons et réductions sur 1 bit
Partant de l'élément i, on le compare avec les autres éléments à partir du rang (i+1).

Si une réduction de 1 bit se produit entre l'élément i et l'élément j, on modifie l'élément i, en introduisant un 'x', et on élimine l'élément j.

On prend ensuite l'élément de rang (i+1) et on recommence jusqu'à arriver au dernier élément.

S'il y a au moins une réduction dans l'étape précédente, on recommence une itération.

Comparaison de deux éléments
Deux éléments se réduisent si, et seulement si, tous les chiffres qui les composent sont égaux deux à deux, sauf pour un chiffre qui est égal à '0' pour l'un et '1' pour l'autre.

FINALEMENT:
on fournit au système PAOLA les descriptions:
- du PLA ET, sous la forme d'un fichier textuel contenant des chaînes de 8 caractères composées des '0', '1' et 'x'.
- du PLA OU de génération des propriétés, sous la forme d'un fichier textuel contenant des chaînes de NA caractères composées de '0' et de '1', qui codent les valeurs des NA attributs.

Chacun des deux fichiers contient M1 éléments où M1 est le nombre de monômes générés par le PLA ET.

IV-C- Génération du PLA de commande

L'étape d'expansion (phase III) a permis d'ajouter un CHAMP qui code le nombre d'instants associé à chaque famille d'instructions.

Ce nombre d'instants est un attribut supplémentaire qui est fourni par le PLA des attributs, au générateur qui code des instants.

IV-C-1/ Codage des instants

On peut adopter un codage binaire (2n fils pour 2^n instants), ou un codage en 1 parmi p (p bits pour p instants), ou un codage mixte, par exemple:

1 bit de poids faible doublé (-> 2 cas),
+ 1 bit de poids fort doublé (-> 2 cas),
+ 5 bits codés en 1 parmi 5,

permettant de coder 5x2x2=20 instants avec 9 fils qui entrent dans le PLA ET, alors qu'un codage binaire nécessiterait 5 bits, soit 10 fils, et un incrémenteur sur 5 bits. On peut par contre se contenter d'un registre à décalage sur 5 bits (+1) et utiliser un signal d'horloge pour le bit de poids faible.

Le choix du codage peut donc:
- soit être libre et il faut alors que le concepteur indique le codage choisi,
- soit être par défaut un codage binaire pour les séquences longues, ou un codage en 1 parmi p pour les séquences courtes. Le générateur d'instant pourra dans ce cas être dessiné automatiquement , selon un modèle prédéfini.

IV-C-2/ Supposons le codage d'un instant choisi

Le PLA de génération des champs peut alors être calculé, la valeur i du champ j étant définie comme active par une somme de produits d'un INSTANT par un attribut.

Codage des valeurs des champs

a/- la valeur par défaut de chaque champ n'apparaît pas dans les équations: il n'y a aucune combinaison qui la génère. On considérera donc que la sortie du PLA de commande vaut par défaut '1' et que seul un monôme peut mettre cette valeur sortie à '0'.
Conséquence: si un champ peut avoir 3 valeurs, les valeurs actives seront codées '00', '01' et '10', et la valeur par défaut sera '11'. Cette dernière valeur ne sera jamais décodée puisqu'elle ne donne lieu à aucune commande.

b/- si un champ n'est pas encodé, il faut générer autant de fois qu'il y a de valeurs actives possibles, ce qui augmente la hauteur du PLA OU, même s'il est brisé.

CONCLUSION
On réalise un encodage systématique des valeurs actives des champs, la valeur '1' étant par définition INACTIVE.

OPTIMISATION DU CODAGE DES VALEURS
Certains champs peuvent avoir une valeur par défaut quelconque: on peut alors récupérer le code 'INACTIF' (des '1' partout) pour encoder une valeur active.
Par exemple: on peut toujours mettre une source sur un bus, ou on peut toujours demander une "addition" à un opérateur.

SOLUTION: on demande au concepteur, dans le cas où une optimisation est possible, c'est-à-dire lorsqu'il y a exactement 2^n valeurs actives, si la valeur (2^n-1) peut être utilisée pour coder une valeur active. Si oui, n bits suffisent, si NON, il en faudra 1 de plus.

FINALEMENT: on fournit au système PAOLA les descriptions
- du PLA ET qui calcule les combinaisons d'un INSTANT codé sur n fils avec 1 attribut (sur 1 fil) et génère NM monômes dans l'ordre où les combinaisons sont données,
- du PLA OU qui génère les valeurs des champs.

Ces deux PLAs se présentent sous une forme simplifiée, car le PLA ET présente cette particularité qu'une propriété peut être combinée avec plusieurs instants. On adoptera une représentation simplifiée:

(<NUMERO D'ATTRIBUT>, <CODE INSTANT>) -> <VECTEUR DE SORTIE>

IV-D- OPTIMISATION TOPOLOGIQUE GLOBALE

Le PLA OU des attributs peut être brisé: ses sorties se font latéralement, c'est-à-dire vers le bas: chaque attribut (fil) rentre ainsi latéralement, c'est-à-dire par le haut, dans le PLA ET de combinaisons (<instant>, <attribut>).

Tous les monômes générés pour le même attribut et pour des instants différents se trouvent donc VOISINS: on constitue ainsi des grappes de monômes:

((Ai.T1), (Ai.T2), (Ai.T3),...,(Ai.Tn)) = 1 grappe

Etant donné que l'on peut avoir un représentant de chaque instant, ces monômes ont de fortes chances de devoir générer une valeur pour tous les champs, donc le PLA OU a peu de chance d'être BRISABLE.

=> il faut faire un choix entre

a/- la BRISURE du PLA OU des attributs qui peut être très efficace (quelques lignes pour 100 attributs),
- l'entrée latérale dans le PLA ET de génération qui est très efficace (une ligne pour 100 attributs),
- la hauteur du PLA OU de génération qui est proportionnelle au nombre de sorties.

b/- la brisure du PLA OU de génération suppose un choix de répartition et des duplications de monômes: le même attribut Ai doit donc être amené en plusieurs points du PLA ET de

génération qui peut à son tour devenir très haut (jusqu'à 100 lignes).

On choisira donc la solution a/, d'autant plus que si on introduit NP attributs latéralement dans le PLA OU, il existe NP lignes verticales pour sortir les commandes latéralement du PLA OU (en général NP>NC, si NC est le nombre de bits nécessaires pour l'encodage des champs de commande).

IV-E- OPTIMISATION LOGIQUE

Une valeur d'un champ est générée par:

(Ai.T1) + (Ai.T2) + ... + (Ai.Tn)

On peut factoriser Ai et on obtient:

Ai.(T1+T2+...+Tn)

pour mettre en évidence une optimisation de la somme (T1+T2+...Tn), et réduire ainsi le nombre de monômes, donc la largeur du PLA ET.

IV-E- ETUDE DES LIMITES TOPOLOGIQUES

Etant données les possibilités non calculables d'optimisations logiques, les limites topologiques ne peuvent pas être calculées, mais on les fera apparaître au cours des phases du traitement.

1/- le PLA des attributs

- le PLA ET a une hauteur proportionnelle au double du nombre de bits du registre instruction NRI (=> hauteur FIXE).
- le PLA OU a une largeur proportionnelle à la somme du nombre des monônes à calculer (NM1) et du nombre d'ATTRIBUTS (NA) (après optimisation logique).

Donc l'implantation topologique est limitée par:

(NM1 + NA + NRI) < K*N

- la hauteur du PLA OU dépend des BRISURES possibles.

2/- le PLA de génération

Le générateur d'instants empiète sur le PLA ET d'une largeur LGI.

Ce PLA ET calcule NM2 monômes (après optimisation) et les NP attributs entrent latéralement.

2ème limite:

(LGI + NM2 + NP + NRI) < K*N

Si la première limite est satisfaite, on voit que:

LGI + NM2 < NM1

donc le nombre de monômes de la deuxième matrice doit être inférieur de LGI à celui de la première matrice, ce qui est généralement le cas, car les équations de génération des commandes sont simples par rapport à celles qui génèrent les attributs.

V- CONCLUSION:

On trouvera dans [Ref. BERTRAND] les résultats de la compilation de l'algorithme du microprocesseur 8048.

Ce chapitre a montré un exemple de mise en oeuvre d'une compilation de Partie Contrôle: les programmes de traitement des informations contenus dans la description de l'algorithme sont simplifiés par le fait que la syntaxe du langage de description (PASCAL-MACSIM ou IRENE) contient les mécanismes implantables dans la structure d'accueil.

Les limites topologiques peuvent être étudiées, au cours des phases de compilation, et les seules alternatives possibles concernent les transformations du type MOORE<-->MEALY qui, reportées dans la description, produisent des modifications topologiques automatiquement "calculées" par le compilateur de Silicium.

Cet exemple de compilateur est en cours d'utilisation, pour la génération de la Partie Contrôle d'un microprocesseur dont le code est celui du INTEL 8048, et réalisé en C-MOS [REF. SAHBATOU, BERTRAND]; les dessins des masques ne sont pas terminés, car il faut encore résoudre les problèmes d'interface informatique entre:
- l'optimiseur Logique [REF. KRASICKY],
- l'optimiseur Topologique [REF. PAOLA].

Le système MACSIM prototype, écrit en PASCAL, fonctionne sur un microordinateur Pascal MicroEngine, avec 64 Koctets de Mémoire. Les 3 phases du traitement (analyse, synthèse et expansion) sont exécutées séquentiellement par 3 petits programmes (d'environ 200 lignes) qui communiquent par des fichiers: la phase de synthèse est particulièrement longue (3 heures pour le 8048), car le même fichier résultat de la 1-ère phase doit être parcouru autant de fois qu'il y a de champs de commande (environ 50 fois); le traitement sur une machine mieux fournie en mémoire (256 Koctets) devrait être de l'ordre de quelques minutes.

CHAPITRE 4

IMPLANTATION DES PARTIES CONTROLE STRUCTURES MICROPROGRAMMEES : UNE APPROCHE DE SOLUTION

I N T R O D U C T I O N

La microprogrammation, introduite par WILKES dans les années 1960, a fait couler beaucoup d'encre, suscité de nombreux développements, donné un coeur à quantité de mini et méga-ordinateurs, et elle a même donné un avant-goût de la compilation du Silicium dans [Ref GUYOT-75] bien que le terme employé ait été alors "compilation de microprogrammes".

Les expériences de l'auteur dans ce domaine concernent l'utilisation d'une famille de circuits TTL (spécialement la famille AMD 2900), qui sert à bâtir le coeur de machines microprogrammées: 5 processeurs microprogrammés ont été conçus pour l'ordinateur PASC-HLL défini dans [REF. SCH-77], ils sont présentés en détail dans [REF. BAI-83].

Cette technique est simple à comprendre: elle sert de base à la pédagogie pour les travaux pratiques d'architecture interne des ordinateurs, nécessitant seulement un simulateur de ROM comme celui utilisé à l'ENSIMAG et présenté dans [Ref SCH-79], du nom de MADAM, pour la mise au point de microprogrammes qui contrôlent des chemins de donnée simples.

Elle consiste à enregistrer un programme, appelé microprogramme, dans une mémoire morte ou vive (on peut alors charger des microprogrammes différents et obtenir des machines aux langages différents [Ref. B1700]), ou proposer la même machine avec des microprogrammes différents dans des ROMs (ex. le LSI11 et le Pascal-MicroEngine Ref. Western Digital).
Ce microprogramme contient des microinstructions, de taille fixe (typiquement de 16 bits à une centaine de bits) qui sont lues dans la mémoire dans un ordre "calculé" par un "séquenceur": cet ordre dépend de l'instruction en cours d'exécution (d'interprétation), et de certaines conditions provenant essentiellement des résultats des

calculs précédents effectués par la PO.

Chaque microinstruction lue est exécutée dans un "cycle-machine": elle contient soit des commandes directes qui sont simplement validées par des signaux d'horloge, soit des commandes encodées, formant des "champs", qui sont décodées et éventuellement combinées avec des "paramètres", pour finalement produire des commandes pour la PO.

Cette technique, bien que d'un certain âge, est encore utilisée, parce qu'elle conserve son avantage sur les structures plus compactes, qui est la facilité de modifier le microprogramme, durant la mise au point des prototypes, ou celle d'étendre le jeu d'instructions (exemple: le jeu d'instruction du MC 68020 est une extension de celui du MC 68000).

Il est par contre très coûteux, sinon impossible, de modifier un séquenceur câblé (si l'on veut ajouter une seule instruction au MC6800 ou au Z8000 dont la PC est câblée et très dense, il faut redessiner une grande partie du circuit).

On donne ici 2 exemples récents et significatifs dans leur catégorie: le MC68000 (PO 16-bits), et le HP9000 (PO 32-bits).

I- DEFINITION FONCTIONNELLE GENERALE

Le séquenceur d'une machine microprogrammée calcule, au début de chaque cycle, l'adresse de la microinstruction suivante, qui sera extraite de la Mémoire Morte, pour être décodée et exécutée au cycle suivant ...
Cette adresse peut provenir de différentes "sources", parmi lesquelles le séquenceur choisit, selon la "commande" qu'il reçoit.

Définition 1:

La commande reçue par le séquenceur provient d'une combinaison entre une "instruction de séquencement", qui peut être conditionnelle, et une "condition" que nous définissons comme étant "de classe 1".

Définition 2:

La source sélectionnée fournit une valeur de type adresse, qui peut dépendre de paramètres externes inconnus du séquenceur. Cette valeur peut également être modifiée par des "conditions" que nous définissons comme étant "de classe 2".

Les définitions générales précédentes recouvrent la majorité des mécanismes connus (parce qu'elles sont générales).

Il existe une infinité de variantes, selon le nombre de sources possibles, l'existence de conditions de classe 1 ou 2, etc..., et un grand nombre d'implémentations différentes de ces mécanismes généraux.

Il s'agit pour nous de proposer des solutions qui respectent les contraintes électriques, temporelles, topologiques et logiques qui proviennent des limites et des buts fixés pour cette étude: intégration et compilation.

RESTRICTIONS:

Bien que l'avenir des circuits VLSI semble être dans le C-MOS, nous nous limiterons, pour cette présentation, à une étude électrique propre au n-MOS, à des exemples de dessins avec les règles normalisées du CMP français [Ref ANCEAU-81]. Nous essaierons cependant de montrer que l'approche topologique globale reste valable, puisqu'elle est basée sur l'existence de 2 couches de fils perpendiculaires: l'existence éventuelle d'une 3ème couche de fils (en Poly2 ou en Métal2), devra bien sûr être étudiée par la suite, mais son impact semble se limiter à la simplification de la distribution des alimentations (pour le Métal2) et à la plus grande compacité des cellules de base. Une étude en cours [REF ANTO-83] permettra de préciser l'influence des règles technologiques sur la topologie des circuits.

Nous préciserons enfin que cette approche est avant tout topologique, c'est-à-dire basée sur des propriétés géométriques propres à une certaine classe de technologies: la recherche française dispose actuellement, et pour quelques années encore, via le CMP, d'un accès à de bonnes technologies (H-MOS2 ou X-MOS, ainsi que C-MOS) qui sont couvertes par cette étude, et les expérimentations se feront à court terme avec ces technologies (1 Poly et 1 Métal); l'accès à des technologies plus modernes (par exemple C-MOS double métal) doit également être envisagé, mais les grands principes topologiques ne sont pas remis en question.

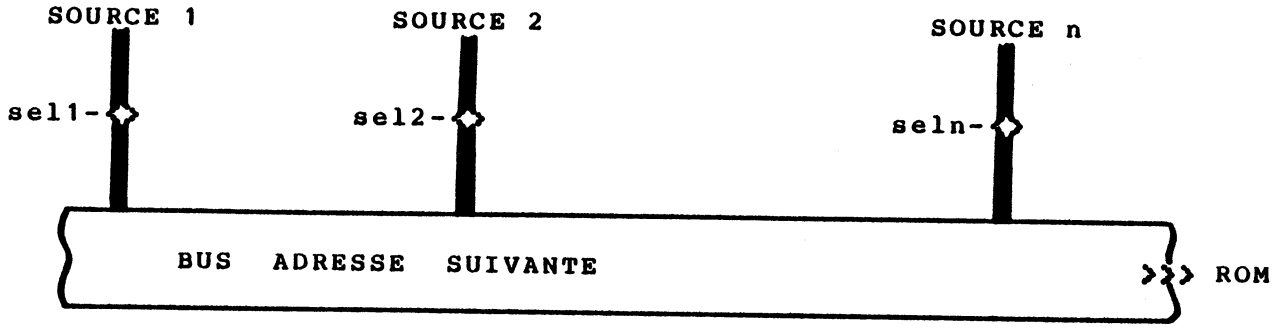
Il paraît important d'entreprendre des recherches à long terme pour influer sur les technologies du futur, afin qu'elles répondent aux besoins des concepteurs, mais il faut avant tout essayer de domestiquer les technologies du présent, pour acquérir une solide expérience !

II- HYPOTHESES ET CONTRAINTES

II-a/ Première hypothèse (topologique):

Les différentes sources d'adresse possibles, et les conditions de classe 2, sont dispersées géographiquement; elles sont donc, dans leur majorité, éloignées de leur destination qui est le décodeur de la ROM contenant le microprogramme.

On définira donc une nappe de fils, appelée "Bus Adresse Suivante" (BAS), qui relie toutes les sources entre elles: un interrupteur, placé près de la source, permettra de mettre la valeur de cette source sur le bus BAS, et seuls les signaux de sélection devront aller du séquenceur jusqu'aux différentes sources.



II-b/ Deuxième hypothèse (logique):

Les conditions de classe 2 agissant aprés la sélection, et à cause de la première hypothèse, on ne doit pas "couper" le bus avec des portes logiques, sous peine de devoir doubler le nombre de ses lignes: on utilisera donc une logique où le '0' est majoritaire, c'est-à-dire un NON-OU de ET, qui est bien adapté au n-MOS (ainsi qu'au C-MOS avec plusieurs transistors "n" et un seul transistor "p"), et permet de mélanger les fonctions de sélection et de

forçage à '0' par une condition (la sélection se ramène à autoriser un forçage éventuel à '0').

II-c/ Troisième hypothèse (électrique):

La topologie proposée devant être capable d'accueillir des applications très différentes, on doit prévoir que le bus BAS peut être long (si les sources sont éloignées) et chargé (si les sources sont nombreuses). On choisira donc une technique de précharge inconditionnelle de ce bus, réalisée dans un emplacement fixé pouvant accueillir des transistors de précharge de taille variable selon la charge du bus et la durée de la phase de précharge.

La décharge conditionnelle sera assurée par 2 interrupteurs en série entre le bus et la masse: ils seront eux-aussi placés dans des emplacements fixés, là où une source se relie au bus, et leur taille sera ajustée pour obtenir un temps de décharge minimal.

II-d/ Quatrième hypothèse (temporelle):

On reste dans le cas d'une machine à 4 phases, telles qu'elles ont été définies pour la Partie Opérative. Sur ces 4 phases, on en réserve 2 pour la lecture de la ROM, les 2 dernières phases T3 et T4: en effet la ROM peut avoir beaucoup de MOTS, donc une phase T3 est réservée au décodage de l'adresse, et la phase T4 à la lecture des bits de la ROM.
==> Il faut donc décharger le bus BAS pendant la phase T2.

Conséquence temporelle:

le calcul des commandes de sélection doit être fait pendant la phase T1, ce qui implique que les conditions de classe 1 doivent être présentes au début de cette phase.

Conséquence algorithmique:

les instructions de séquencement conditionnel ne peuvent porter que sur des conditions directes dont la valeur ne dépend que d'un état antérieur. En particulier, on ne peut pas se permettre de choisir quelle sera la condition dont la valeur permettra de choisir entre deux sources d'adresse.

Conséquence fonctionnelle:

on doit combiner dans un PLA-ET unique le code des instructions de séquencement et les conditions de classe 1.

Exemples:
Les instructions IRENE:
(IS=1): if (C1) then goto $E1 else return endif;
ou
(IS=2): if (not C1 or C3) then call $E4 else goto $TE[RI] endif;
ou
(IS=3): case RI<1:0> of
"00:goto $E0; "01:call $E0; "10:repeat; "11:return
endcase;

définissent des instructions conditionnelles et des conditions de la classe 1.

Si l'on ne dispose que d'une seule phase T1 pour calculer les commandes de sélection des sources, la solution rapide consiste à définir autant d'instructions de séquencement qu'il y a d'instructions conditionnelles portant sur des conditions de classe 1 différentes ou sur des sources sélectionnées différentes.

D'autre part, les combinaisons des conditions de classe 1, si elles ont un caractère statique (combinaisons de bits de RI) peuvent être

pré-calculées dans un double PLA de caractéristiques, sinon elles doivent être combinées au code de l'instruction de séquencement.

Exemples précédents:

```
(C1 and (IS=1))                 --> goto $
(nC1 and (IS=1))                --> return
(nC1 or C3 and (IS=2))          --> call $
(C1 and nC3 and (IS=2))         --> goto $TE[RI]
((RI<1:0>="00) and (IS=3))      --> goto $
((RI<1:0>="01) and (IS=3))      --> çall $
((RI<1:0>="10) and (IS=3))      --> repeat
((RI<1:0>="11) and (IS=3))      --> return
```

==> Leçon pour le compilateur:

La compilation de la description IRENE doit mettre en évidence toutes les instructions conditionnelles portant sur des conditions différentes ou sur des sources différentes, pour en déduire:

- un codage de ces instructions,
- le remplissage du PLA-ET de combinaisons, et éventuellement son optimisation logique,
- la liste des combinaisons des conditions entre elles, afin de noter si ces combinaisons seront utilisées dans d'autres formes syntaxiques, soit pour le calcul paramétré d'une adresse (condition de classe 2), soit pour le calcul paramétré d'une commande (condition de classe 3 à définir).

III - ARCHITECTURE POUR UN SEQUENCEUR:

Le séquenceur peut être plus ou moins riche en fonctions, selon la richesse en instructions de séquencement présentes dans l'algorithme qu'il implante; on doit cependant prévoir ici le cas le plus général qui consiste en:

- un Compteur Ordinal de N bits,
- une pile de profondeur P (P mots de N bits),
- un nombre variable de sources d'adresse (adresses immédiates ou adresses calculées).
- un registre temporaire permettant de réaliser des attentes actives en "bouclant" sur la même adresse.

Les éléments précédents vont constituer une Partie Opérative, composée d'éléments connus (registres et opérateurs) dont on sait faire le dessin automatiquement (par une fonction LUBRICK).

Une structure à deux doubles bus différentiels, placée sous des tranches VERTICALES (au nombre de N), va accueillir le séquenceur, dont la hauteur dépendra du nombre P de mots de la Pile, dont la taille peut être calculé comme étant égale au nombre d'imbrications d'appels de procédures et de boucles.

Le contrôle de l'accès et de l'adressage des registres qui constituent la Pile peut être fait par un registre à décalage dynamique. On synchronise les décalages et les accès sur le bus selon le tableau suivant:

Opération (phase)	Lecture (T2+T3)	Ecriture (T3)	Empile (T2)	Dépile (T4)
NOP	0	0	0	0
PUSH	0	1	1	0
POP	1	0	0	1
PULL	0	0	0	1
LOOP	1	0	0	0

On peut ainsi rafraîchir les registres dynamiques en T1 ou en T3, les décaler dans un sens en T2, et dans l'autre sens en T4.

III-A/ IMPLANTATION DES ADRESSES IMMEDIATES

Principe 1:

Il existe toujours au moins une adresse immédiate à générer pour chaque instruction: c'est l'adresse de la 1-ère microinstruction à exécuter pour cette instruction, appelée son "point d'entrée". La paramétrisation (de type MEALY) a pour 1-er effet bénéfique de réduire le nombre de points d'entrée différents, puisque plusieurs instructions peuvent être interprétées par un microprogramme unique.

Définition 1:

on définit un PLA (ET-OU) de "points d'entrée", qui décode le code opération de chaque (famille d') instruction(s) et fournit une adresse.

Principe 2:

certaines familles d'instructions peuvent avoir un point d'entrée identique, mais des séquences d'interprétation distinctes par au moins une adresse de branchement: c'est une situation fréquente qui correspond au principe de familles d'instructions qui ont les mêmes types d'opérandes, mais réalisent des traitements

différents. On peut dans ce cas utiliser la notion de sous-programme, mais cela ne résoud pas le problème des 2 adresses.

Définition 2:

On définit un 2-ème PLA de points d'entrée, que l'on mélange au 1-er, afin de mettre des monômes en commun et de partager les entrées qui sont dans les 2 cas les bits du code-opération.

Principe 3:

Une Pile permet de réaliser des "boucles": on range dans la pile l'adresse de l'instruction suivante, puis, en fin de boucle, on référence le sommet de pile comme source d'adresse.

Principe 4:

Toutes les instructions ne nécessitent pas de branchements à une adresse immédiate; celles qui en nécessitent peuvent également être décodées dans un autre PLA de branchement, à l'entrée duquel on peut d'ailleurs mélanger des conditions aux bits du code-opération.

L'application des 4 principes précédents, si elle résoud tous les cas, permet d'éviter la définition d'un champ "adresse immédiate" dans la microinstruction, ce qui diminue la largeur de la ROM, et donc également sa hauteur et donc celle de la PC.

Si toutefois l'utilisation de cette "adresse immédiate" s'avère indispensable, on peut:

- réduire sa taille en limitant la portée des branchements soit à l'intérieur d'une "page" courante, soit au début d'une "page";
- placer les bits de cette "adresse immédiate" en face des tranches de bits qui accueillent le séquenceur, de sorte que chaque bit emprunte une ligne de bus qui traverse chaque tranche.

III-B/ LE SEQUENCEMENT CONDITIONNEL

La Microinstruction extraite de la ROM doit contenir une

"instruction de séquencement", qui peut être conditionnelle à plusieurs niveaux:

III-B-1- Les conditions de classe 1:

Ces conditions doivent être directement commbinées, dans un PLA-ET, avec le code de l'instruction de séquencement, pour donner naissance, par un PLA-OU, aux "commandes" du séquenceur, afin que ce dernier effectue une sélection de la source de l'adresse suivante.
On peut réaliser une paramétrisation de ces conditions, c'est-à-dire calculer des combinaisons d'INDICATEURS et de bits du code-opération, dont les valeurs seront, au début du cycle, combinées avec l'instruction de séquencement.

Exemple: soit les instructions IRENE:

```
if C1 then CALL($2) else CONTINUE endif;
if C2 then CALL($2) else CONTINUE endif;
```

on paramètre C1 et C2, en définissant 2 "paramètres" P1 et P2, et on obtient une seule forme,

```
if ((C1 and P1) or (C2 and P2)) then ...
```

Par contre

```
if C3 then LOOP else RETURN endif;
```

ne peut pas être paramétrée.

III-B-2- Les conditions de classe 2:

Ces conditions peuvent agir sur la génération de points d'entrée, c'est-à-dire être combinées avec les bits du code-opération à l'entrée d'un PLA-ET.
Elles peuvent également être sélectionnées pour réaliser un forçage de certains bits du "bus adresse suivante" BAS, pour réaliser ce qui est communément appelé un ECLATEMENT, le forçage à '0' étant

prioritaire pour ce bus qui est préchargé.

Dans les 2 cas, il faut définir des combinaisons de conditions, calculées par un PLA de COMBINAISONS, dont les sorties sont validées par des commandes issues de la microinstruction.

III-C/ CONCLUSIONS POUR LA TOPOLOGIE

Un nombre variable de PLAs, de tailles et de formes variables, doit être implanté pour réaliser la génération des adresses et des combinaisons de conditions.
Ces PLA ont en commun des entrées qui sont les bits du code opération en provenance de la PO, et des conditions qui proviennent elles-aussi de la PO. Ces signaux arrivent par des Lignes Virtuelles verticales, après avoir traversé la couche de paramétrisation: il est donc raisonnable de les faire monter jusqu'à un canal Horizontal, qui va constituer un très large PLA, sur toute la largeur du circuit. Les lignes de ce canal pourront bien sûr être brisées, pour supporter
- soit des entrées,
- soit des monômes,
- soit des sorties.

Ce très large PLA va accueillir toutes les combinaisons de conditions provenant du bas, ainsi que les bits de microinstruction provenant du haut après avoir traversé successivement le Séquenceur et une couche contenant des PLAs-OU.

La limite topologique concerne le nombre des Lignes virtuelles libres, sachant que cette couche doit être transparente pour les commandes qui sont à ce niveau encodées dans NC champs de Ei bits:

il reste (K*N - SIGMA(Ei)) lignes pour réaliser cette couche.

IV- IMPLANTATION DE TRANCHES DE BITS SOUS LA ROM

Le nombre de bits à extraire de la ROM à chaque cycle (nombre de bits de la microinstruction) est donné par la compilation de la description de l'algorithme: il dépend du parallélisme désiré, des encodages réalisés pour les champs, et du degré de paramétrisation.

Ce nombre de bits NBD, comparé à la largeur du circuit, impose un facteur de multiplexage MUX pour la ROM, qui occupe une largeur proportionnelle à:

$$NBD*MUX + 2*NBA \quad (<=K*N),$$

si NBA est le nombre de bits d'adresse.

Il est souhaitable que MUX soit un multiple du nombre RAPP de fils virtuels entre 2 rappels de masse;
posons:

$$MUX = k * RAPP$$

On cherche alors une solution qui permet de faire correspondre les lignes de rappel de masse de la ROM avec celles des tranches de Partie Opérative qui accueillent le séquenceur. On donne un tableau de valeurs numériques, qui fait apparaître la hauteur d'une tranche (en Lambda) et sa transparence:

MUX	RAPP	HAUTEUR	TRANSPARENCE
4	20	54	2/5
5	20	54	1/4
6	30	62	1/5
7	14	57	0
8	16	65	1/2

V- C O N C L U S I O N

Un travail important reste à faire pour l'implantation des structures microprogrammées: deux études sont en cours, l'une concernée par la maîtrise des problèmes topologiques, l'autre par la compilation de l'algorithme écrit en IRENE.

Cette approche semble cependant prometteuse pour résoudre ce problème complexe, dont les applications sont multiples.

CHAPITRE 5

SYSTEME D'HORLOGERIE MULTI-PHASE

INTRODUCTION DE MEMORISATIONS DYNAMIQUES

I N T R O D U C T I O N

La conception de Parties Contrôle ne peut être dissociée de celle des Parties Opératives, ni sur le plan TOPOLOGIQUE, ni sur le plan TEMPOREL.

Sachant que nous avons défini un modèle pour les Parties Opératives lié à l'existence de BUS DIFFERENTIELS A PRECHARGE qui fonctionnent en 4 (ou 3) phases d'horloge, nous devons utiliser ces mêmes 4 (ou 3) phases pour le séquencement des transferts élémentaires entre les composants de la Partie Contrôle. Nous adopterons en fait un système à QUATRE phases.

D'autre part, l'approche topologique générale que nous présentons doit faire appel à des principes généraux qui permettront de résoudre les problèmes électriques et temporels pour des applications de tailles différentes:
il est donc nécessaire de définir un fonctionnement électrique général qui ne posera pas de problèmes insurmontables d'implantation. C'est pourquoi nous présentons ici le principe, et les applications possibles de l'horlogerie multi-phase, combinée avec la mémorisation dynamique, de manière à surmonter les problèmes d'amplification des signaux:
on rendra systématique la perte d'une phase (sur 4) pour précharger certaines lignes, et on alternera un mécanisme de précharge avec une amplification dynamique pour fabriquer des systèmes rapides dans lesquels la localisation et le dimensionnement des transistors d'amplification peuvent être appréhendés par le compilateur de silicium.

Les principes présentés ne sont pas nouveaux, nous tentons seulement ici un regroupement de différentes techniques existantes, pour résoudre simplement, et systématiquement, un problème délicat.

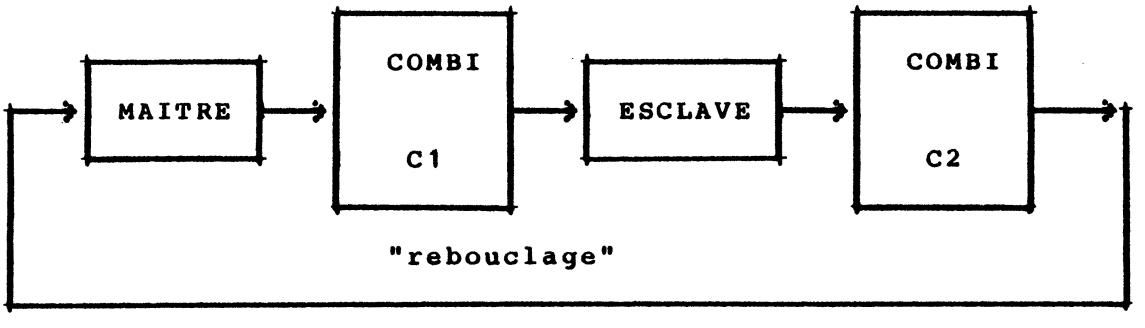
I- RAPPELS SUR LA MEMORISATION DANS UN CIRCUIT n-MOS

Contrairement aux technologies bipolaires (TTL, ECL) avec lesquelles on fait grand usage de registres ou bascules chargées sur un front d'horloge, la technologie n-MOS a rendu systématique l'usage de registres transparents, parfois statiques, ou semi-statiques, ou même dynamiques, et celui de la précharge de capacités comme principe de mémorisation.

I-A- Analogie avec les structures bipolaires

Il existe une analogie entre les structures MOS et les structures bipolaires, lorsque, pour ces dernières, on utilise un registre ESCLAVE transparent (quand l'horloge H est à son niveau haut "1"), et un registre MAITRE chargé sur le front montant de cette même horloge (quand elle passe de "0" à "1") (REF. Advanced Micro Device, circuits Am2901, Am2909,...).

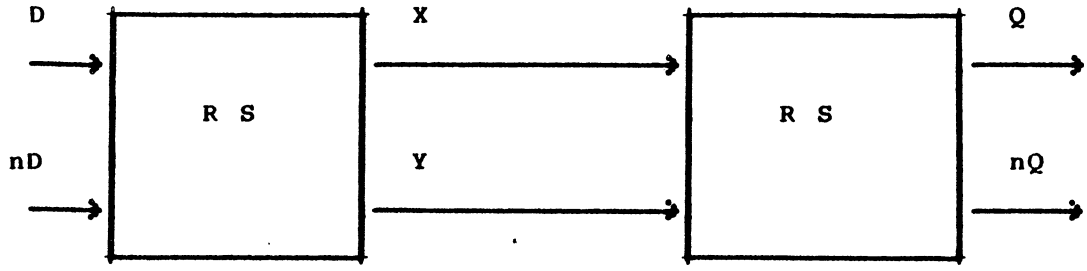
Cette dernière organisation fait apparaître deux phases, bien qu'il n'y ait qu'un seul signal d'horloge: la circulation de l'information et le "rebouclage" d'éléments de mémorisation sont possibles grâce à la dissymétrie des MAITRES et des ESCLAVES.



Les circuits combinatoires de la classe C1 (leurs entrées sont des MAITRES) doivent avoir un temps de calcul inférieur à une demi-période d'horloge, alors que les circuits de la classe C2 (leurs entrées sont des ESCLAVES transparents) ont un temps de calcul

voisin de la période de l'horloge.
Il est donc possible d'organiser une structure "pipe-line" à l'intérieur même de chaque cycle d'horloge.

Les registres chargés sur un front coûtent cher en nombre de portes et donc en surface. On en trouve très peu dans le coeur d'un CI, seulement près des plots d'entrée du circuit pour ECHANTILLONNER des entrées asynchrones:



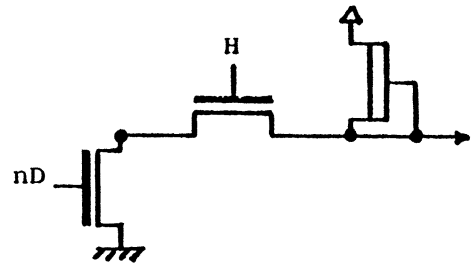
On a deux bascules RS: la première est un LATCH transparent qui se ferme sur une transition de l'horloge; l'entrée peut être considérée comme échantillonnée à cet instant. Pendant la deuxième phase, la deuxième bascule peut éventuellement être modifiée selon les valeurs de X et de Y.

APPLICATION : c'est une structure maître/esclave dans laquelle on peut introduire des circuits combinatoires entre les bascules RS de type MAITRE et celles de type ESCLAVE (transparentes).

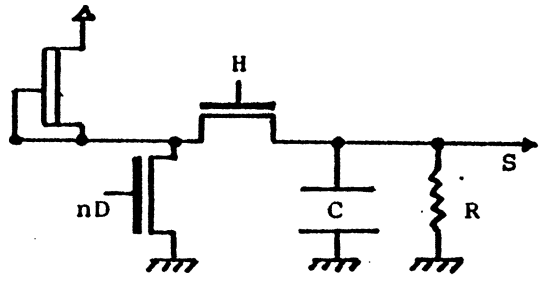
I-B- **Solution N-MOS:**

I-B-1/- Les éléments de mémorisation peuvent être simplifiés en n-MOS: on remplace une porte ET par un seul transistor-interrupteur.

ce qui est logiquement équivalent si on ajoute une charge et un chemin vers la masse (pour H=1).



I-B-2/- Deuxième simplification:
on remplace une bascule RS par une capacité, qui est un réservoir d'énergie , et on place le générateur de courant près de l'entrée:



D	nD	S
0	1	0
1	0	1

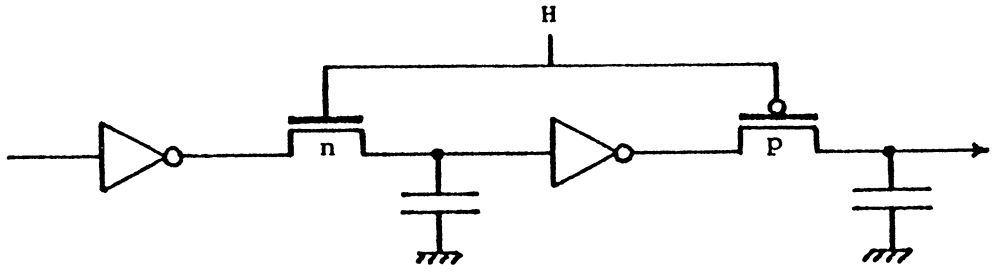
quand H devient égal à "1",
si D=0, le générateur de courant charge la capacité C,
si D=1, la capacité est déchargée à la masse.

Problème électrique: Le point S ne restera au niveau logique '1' qu'un temps limité défini par les valeurs de C et de R: il faudra donc "rafraîchir" la valeur de S avec une fréquence supérieure à une fréquence fixe, définie par la durée de la décharge de la capacité C par la résistance de fuite R vers la masse. (selon la technologie utilisée, le temps maximum qui sépare deux rafraîchissements peut aller de quelques dizaines de micro-secondes à quelques milli-secondes).

I-B-3/ Structure maître/esclave dynamique

La solution bipolaire requiert une seule horloge mais deux types de portes logiques duales: le ET et le OU.

En technologie C-MOS, on dispose de transistors N et P (complémentaires), ce qui pourrait permettre de simplifier:



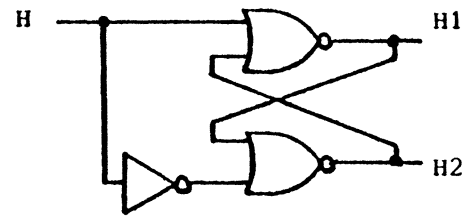
Cette structure n'est malheureusement pas utilisable, car un interrupteur unique transmet mal l'information: il dégrade le niveau électrique, et on doit utiliser des transistors des 2 sexes (N et P), commandés par des horloges complémentaires.

En technologie N-MOS, un seul transistor (N) existe et on pourrait construire nH (l'horloge complémentaire) à partir de H; mais l'inverseur qui génère nH introduit un retard critique quand H passe de 0 à 1:
pendant un court instant correspondant au retard de nH par rapport à H, les deux interrupteurs sont fermés, et on peut assister à une décharge du point S qui avait une chance sur deux d'avoir la valeur "1".
=> on n'a pas naturellement de structure maître/esclave: l'entrée du maître influe sur la valeur de l'esclave.

CONSEQUENCE:
on doit définir deux signaux d'horloge H1 et H2, l'un contrôlant le chargement d'un maître (H1) et l'autre celui d'un esclave (H2); ces 2 signaux ne doivent pas être à "1" simultanément: on dit que H1 et H2 sont sans recouvrement (NON-OVERLAPPING).

On peut se reporter à [Ref. MEAD & CONWAY, chapitre 2].

On génère H1 et H2 à partir d'une horloge unique H par le circuit (a):

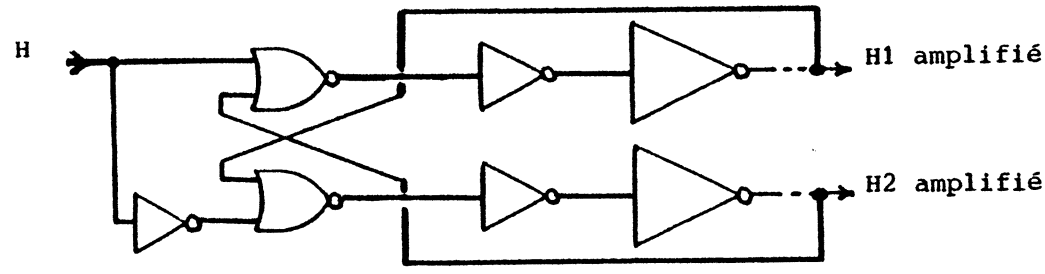


circuit (a)

Ce montage donne un exemple de circuit auto-synchronisé: la montée d'une phase d'horloge est contrôlée par la descente de l'autre phase, et vice-versa.

REMARQUE: On peut mettre un nombre PAIR d'inverseurs en série, en sortie des 2 portes NON-OU (circuit b), ce qui permet:

- d'allonger le temps de NON-recouvrement,
- de construire une chaîne d'amplificateurs pour H1 et H2.



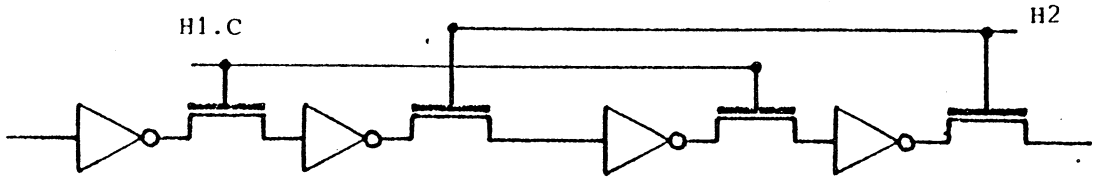
circuit (b)

I-B-4/ Applications de la structure dynamique

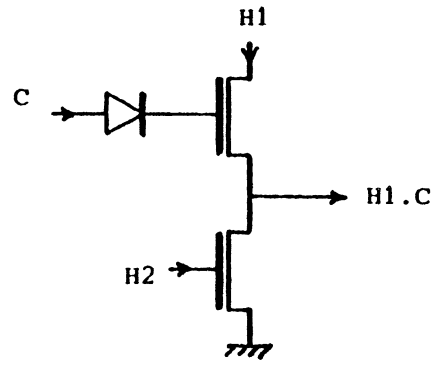
a/- Le registre à décalage infini:

Une succession de cellules dynamiques contrôlées alternativement par H1 et H2 se fait naturellement. On peut bien sûr modifier le rythme des décalages, en ne décalant que sur H1 **et** une condition C (signal H1.C), mais on risque alors de perdre des valeurs '1' si C n'est vrai que rarement (problème de fréquence de

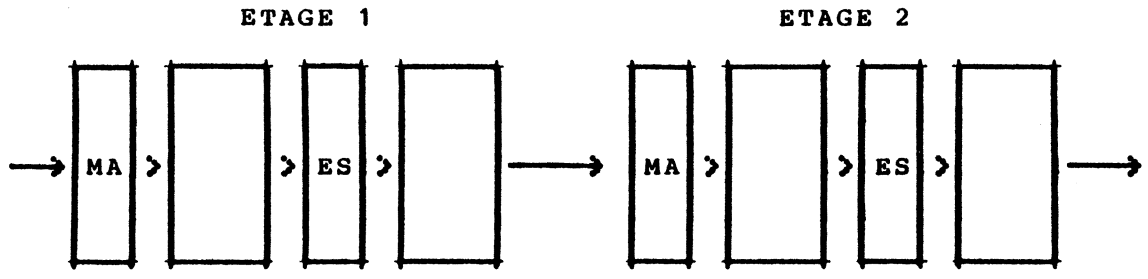
rafraîchissement).



REMARQUE: On doit faire attention à la génération de la commande H1.C à partir de H1:
si la porte qui calcule H1.C a une descente lente, sa sortie et H2 peuvent se recouvrir (valoir "1" simultanément):
La solution réside dans l'utilisation d'amplificateurs de commmandes dynamiques:
- avec un "rappel" à '0' commandé par H2,
- avec éventuellement une diode de "bootstrap" permettant une montée rapide de la commande jusqu'à un potentiel égal à VDD (si VGS>VDS+VTH).



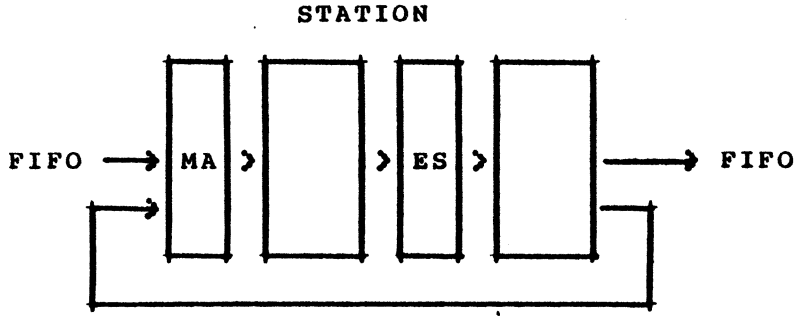
GENERALISATION:
Chaque point de mémorisation dynamique peut être remplacé par une famille de registres : on obtient une structure de machine qu'on peut appeler un PIPE- LINE dynamique:



Les deux étages ETAGE1 et ETAGE2 fonctionnent en parallélisme : le résultat du traitement effectué au cycle i par l'étage 1 est traité au cycle (i+1) par l'étage 2, etc...

REMARQUE: On peut introduire des étages qui réalisent uniquement des transferts, ce sont alors des files d'attente (FIFO = First-In-First-Out) et on construit ainsi une structure avec des stations qui:

- soit font un traitement local (rebouclage = travail pour le client courant),
- soit prennent un nouveau client dans la file d'attente située en entrée ...



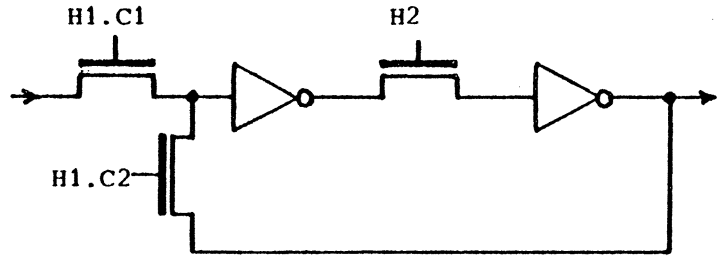
I-C - LE REGISTRE SEMI-STATIQUE

- Les registres bipolaires présentés en I-A- sont STATIQUES: une bascule RS conserve indéfiniment sa valeur. Le signal H n'est donc pas vraiment un signal d'horloge mais un signal de chargement, pouvant être actif ou non, et construit à partir d'une horloge.

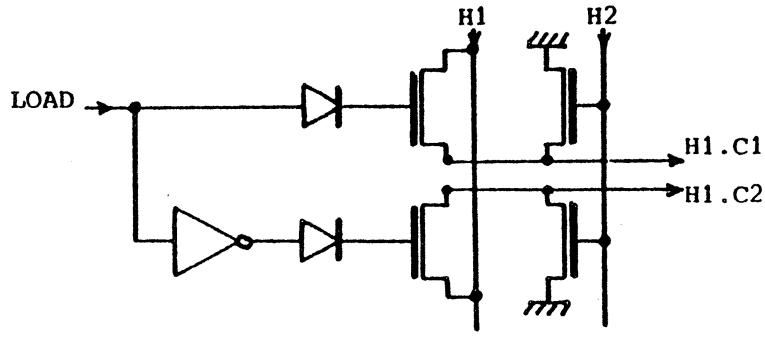
- Les registres N-MOS présentés en I-B- sont eux DYNAMIQUES: s'ils ne sont pas chargés avec une nouvelle valeur, ils peuvent perdre leur ancienne valeur.

I-C-1/ Solution (a):

La solution A consiste à recharger ces registres **systématiquement** avec une valeur qui est soit une nouvelle valeur, soit leur ancienne valeur, en utilisant deux commandes (H1.C1) et (H1.C2) disjointes (elles ont des valeurs complémentaires), et un transfert maître/esclave commandé par l'horloge H2.



La génération des commandes (H1.C1) et (H1.C2) peut être réalisée par:

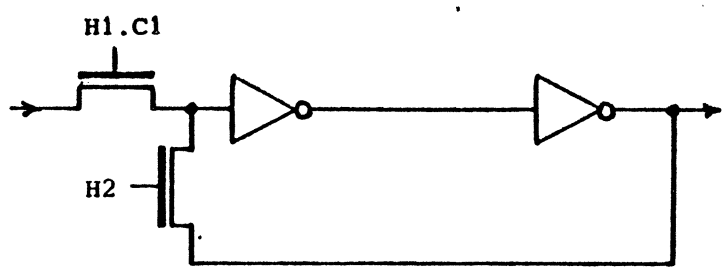


REMARQUE: La commande LOAD, activée par H1, doit être calculée pendant que H2=1 (surtout si l'on veut une génération rapide avec

un bootstrap).

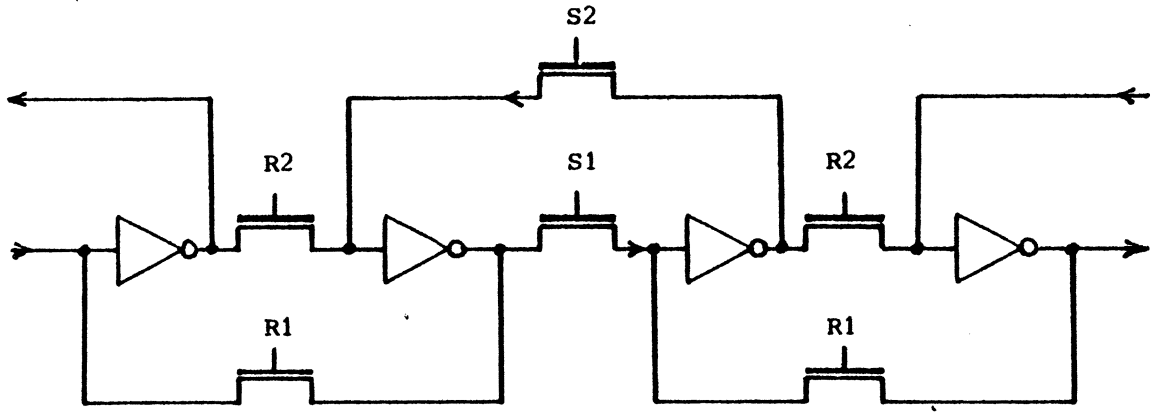
I-C-2/ **Solution (b)**:

On conserve une seule commande (H1.C), active pendant H1, et on contrôle le rebouclage par l'horloge H2: on ne peut pas dans ce cas se permettre d'arrêter l'horloge H2, mais par contre il n'y a pas de contrainte sur la fréquence des chargements d'une nouvelle valeur.

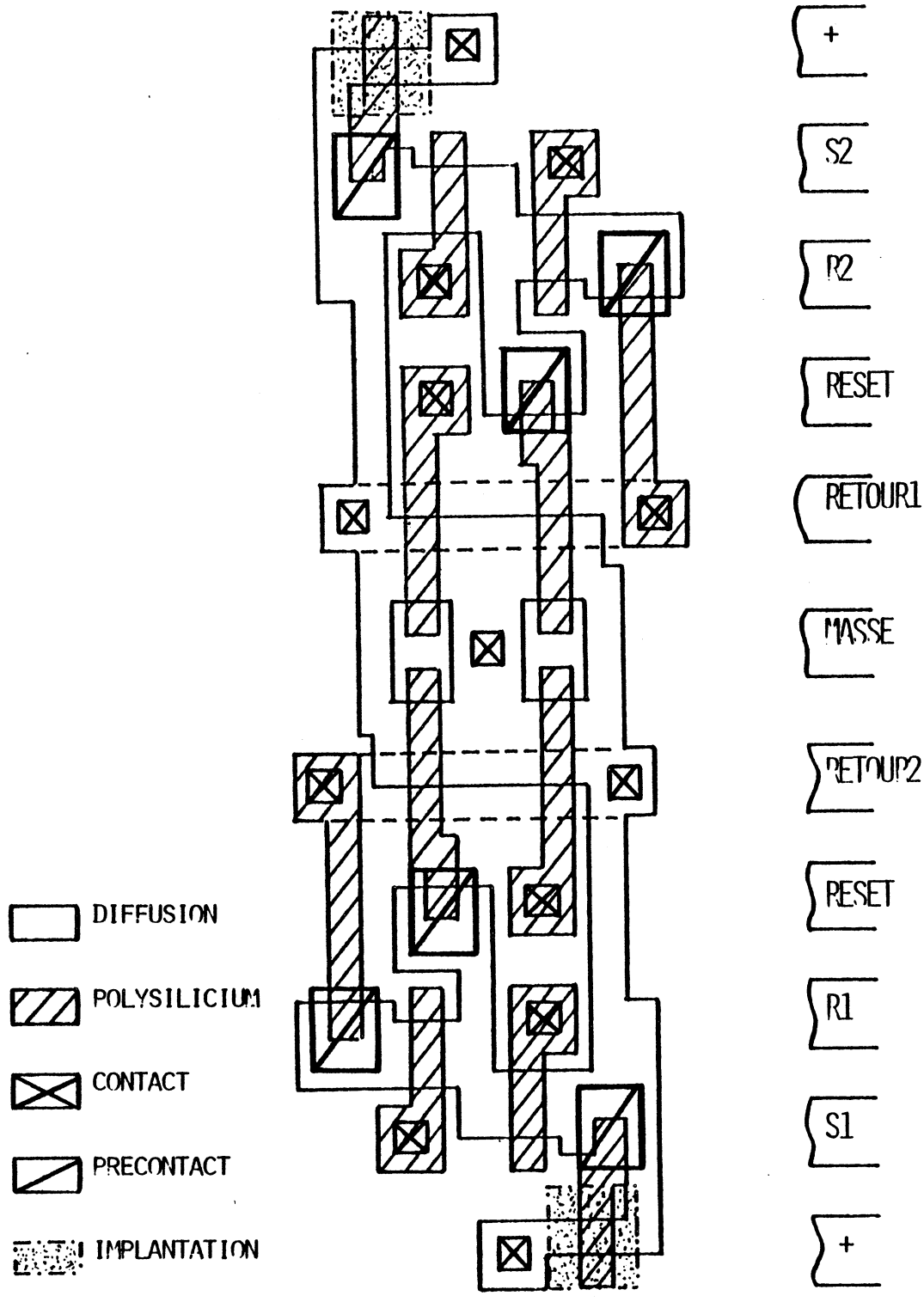


I-C-3/ **APPLICATION**:

le registre à décalage bidirectionnel, avec 2 commandes de rafraîchissement R1 et R2 (actives respectivement sur H1 et H2), et 2 commandes de décalage S1 et S2; S1, actif sur H1, contrôle le décalage vers la droite, et S2, actif sur H2, vers la gauche.



Ce registre peut s'implanter comme suit (règles n-MOS normalisées):

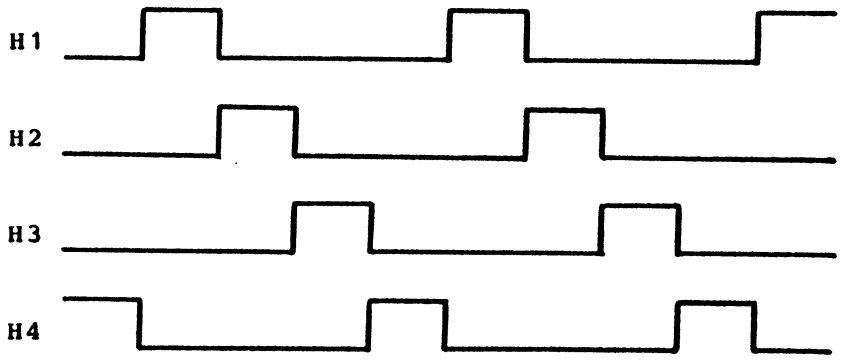


II- EXTENSION A PLUS DE DEUX PHASES ET APPLICATIONS

II-A- DEFINITIONS ET PRINCIPES

DEFINITION 1:

On définit un système à plus de deux phases, distinguées par plus de deux horloges "sans recouvrement" H1, H2, H3,...



DEFINITION 2:

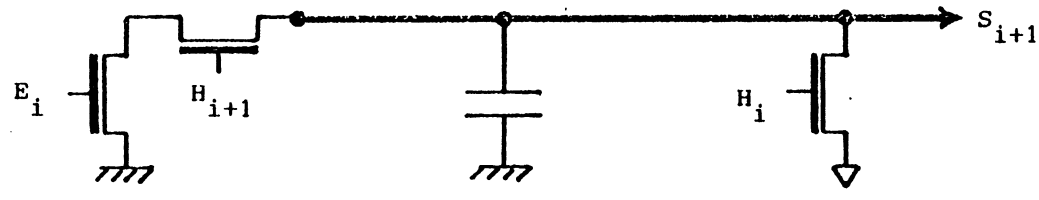
Toute ligne, dans un circuit intégré, définit une CAPACITE de mémorisation dynamique, d'autant plus importante qu'elle est longue et qu'elle est connectée à plus de grilles, drains ou sources de transistors.

PRINCIPE:

Le principe de la PRECHARGE consiste à:

1/ CHARGER la capacité par un signal d'horloge Hi (au temps Ti),

2/ la décharger ou non pendant le temps Ti+1, selon la valeur d'un signal Ei.



Si la valeur de Ei vaut "1" pendant le temps Ti+1 (lorsque l'horloge Hi+1 vaut "1"), la ligne S est déchargée et devient égale à "0", cette décharge étant **IRREVERSIBLE**.

II-B- CONSEQUENCES ET GENERALISATION

CONSEQUENCE:

Il est impératif

- que la valeur de Ei ait été calculée avant le début de Ti+1, c'est-à-dire pendant le temps Ti, autrement dit que Ei soit STABLE à la fin de Ti,
- ou alors que le passage à "1" de Ei, s'il se produit après le début de Ti+1, soit correct et non pas transitoire.

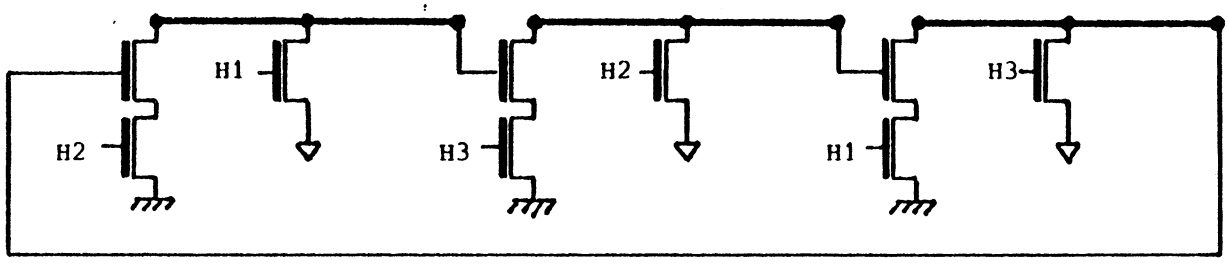
ITERATION:

Si l'on itère sur l'indice i, on voit que l'on peut appliquer le même mécanisme pour le calcul de Ei:

-1/ on charge le signal Ei au temps Ti-1,

-2/ on le décharge au temps Ti avec une valeur Ei-1 qui a été calculée au temps Ti-1.

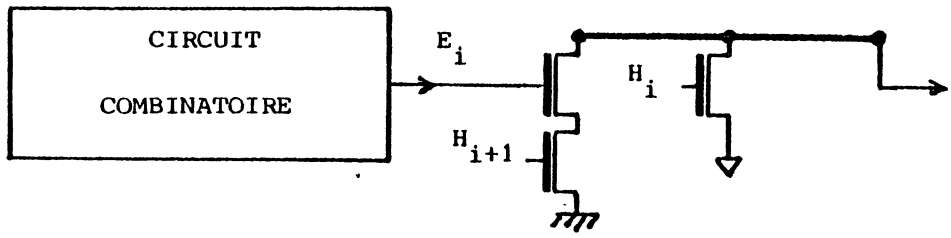
Exemple à trois phases:



REGLE a:

Une valeur Ei+1 utilisée au temps Ti+1 pour décharger une ligne préchargée au temps Ti, doit être CALCULEE pendant le temps Ti.

La décharge peut durer une PHASE complète et donc requérir la traversée de circuits combinatoires actifs pendant cette phase.



REGLE b:

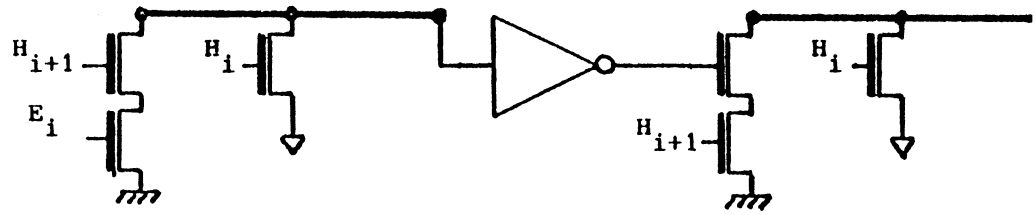
La décharge de Si+1 est irréversible, MAIS on peut accepter un RETARD dans le calcul de Ei, à condition que sa valeur soit égale à "0" au début de Ti+1.

II-C- APPLICATION IMPORTANTE: PARALLELISME dans une phase:

On peut calculer pendant le temps Ti+1 la valeur de l'entrée Ei, au lieu de le faire au temps Ti, à condition que cette entrée Ei ne passe pas à "1" d'une manière intempestive:

REGLE R1:

une solution consiste à définir Ei comme le complément d'une ligne préchargée au temps Ti, afin d'enchaîner deux opérations de décharge.



En général, il faut que l'état de REPOS de Ei soit "0" au début du temps Ti.

=> On peut placer un certain nombre de montages de ce type en série, à condition que le "temps de calcul" total, qui est la somme de tous les temps de décharges successives, ne dépasse pas la durée

de la phase.

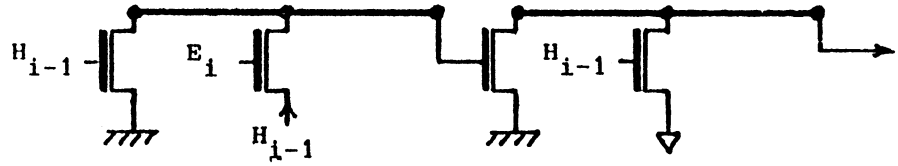
II-D- APPLICATION: couplage avec l'AMPLIFICATION DYNAMIQUE:

L'amplification dynamique consiste à utiliser un signal d'horloge Hi comme alimentation d'un amplificateur: la sortie de l'amplificateur ne vaut 1 que si son entrée vaut 1 au début du temps Ti (pour obtenir un effet de bootstrap) et seulement après le début de Ti.
Comme la capacité de la sortie peut être chargée au temps Ti, il faut la décharger au plus tard au temps Ti-1, pour que Si+1 soit égal à Ei à la fin du temps Ti.

On a donc un principe dual du précédent:
- alimentation conditionnelle (en Ti),
- décharge systématique (au plus tard en Ti-1).

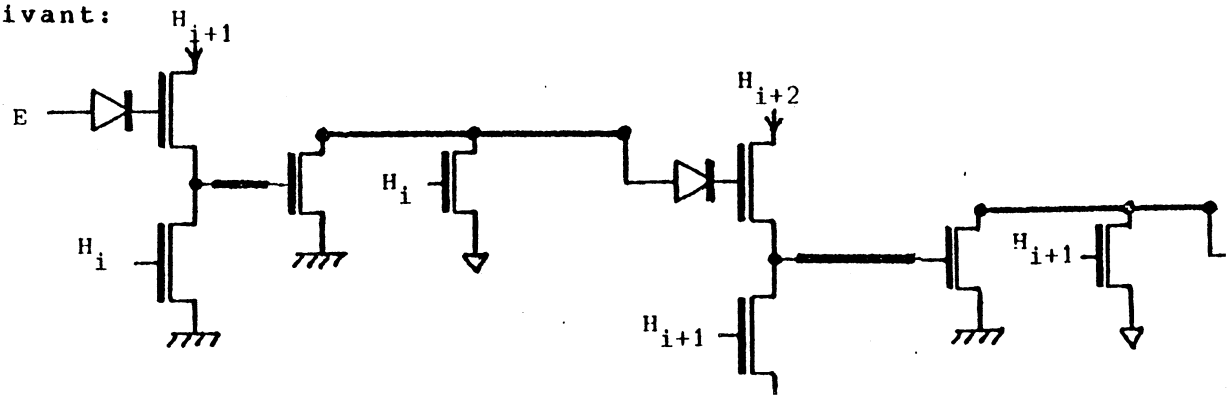
COMBINAISON des deux principes:

Une sortie d'amplificateur contrôlé par Hi peut décharger une ligne préchargée par Hi-1.



SIMPLIFICATION (S1):

Comme Ei-1 ne peut valoir "1" qu'après le début de temps Ti, on peut se passer d'un interrupteur, ce qui donne le montage suivant:



III- APPLICATIONS A UNE STRUCTURE MICROPROGRAMMEE

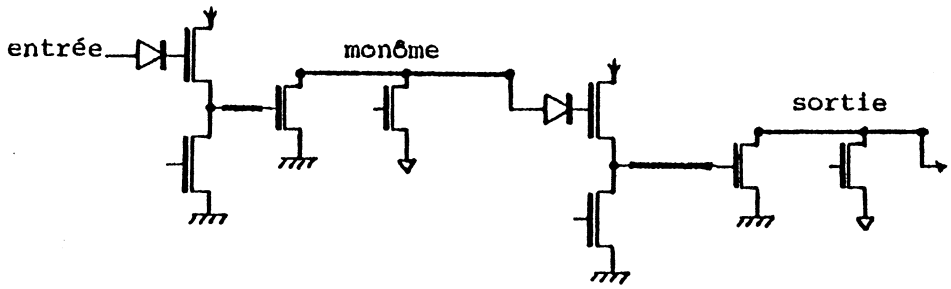
On montre ici des applications des principes précédents, afin d'anticiper sur la résolution des problèmes électriques et topologiques rencontrés dans la génération des masques d'une Partie Contrôle.

Application 1:

E est une entrée d'une matrice de PLA, le transistor contrôlé par Ei-1 est un point du PLA et S une sortie.

Application 2:

On place deux matrices de PLA en série (matrice ET et matrice OU)



Si les entrées sont calculées à la fin de Ti, on obtient une sortie durant le temps Ti+2 (voir plus loin le schéma temporel d'une ROM).

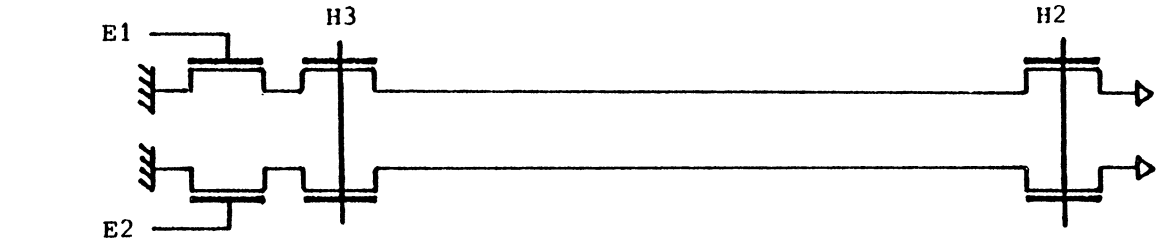
III-A- **APPLICATION A UNE MEMOIRE MORTE ou ROM:**

Soit E l'adresse de la microinstruction suivante à lire dans une ROM constituée de deux matrices de PLA, un PLA-ET servant de décodeur, un PLA-OU codant les valeurs des mots de la ROM.

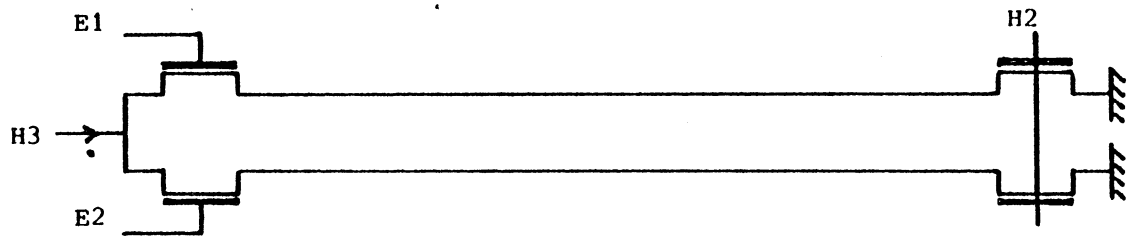
III-A-1/ **Le décodeur de l'adresse:**

on peut:

- (a) soit précharger les entrées en T2 et les décharger en T3,
- (b) soit décharger les entrées en T2 et les amplifier en T3.

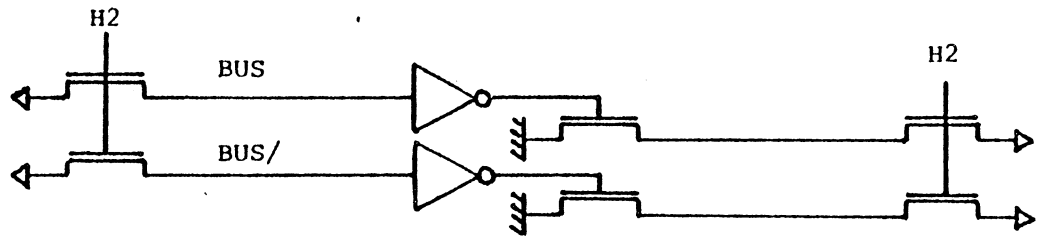


(a)



(b)

Remarque: si BA est un double bus préchargé en T2 et déchargé (d'un seul coté) en T3, on peut simplifier en utilisant la Simplification (S1), tout en respectant la Règle (R1).



La solution la moins contraignante en temps est la solution (a), c'est celle que nous retiendrons.

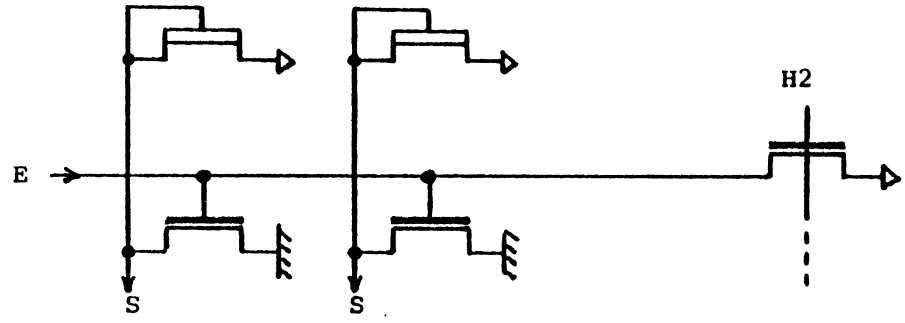
III-A-2/ **La sortie du décodeur**:

Par définition une seule sortie du décodeur est active: si on travaille en logique positive il faut précharger toutes les n sorties, pour en décharger (n-1) à chaque cycle.

Afin de diminuer la consommation, on peut passer en logique négative: toutes les sorties restent à "1" sauf une qui se décharge (puis elle-seule sera rechargée, etc...):

=> UN DECODEUR du type NAND réduit la consommation mais il n'est pas implantable, en N-MOS, au-delà de 6 à 8 bits d'adresse (6 à 8 transistors en série); cette solution pourra par contre être retenue pour le C-MOS.

=> on restera donc en logique positive, pour le n-MOS, et un simple générateur de courant (MOS de charge) sera utilisé à la place de la précharge, car, les entrées étant préchargées en T2, elles passent toutes à "1", mettant toutes les sorties à "0" par l'intermédiaire du décodeur.



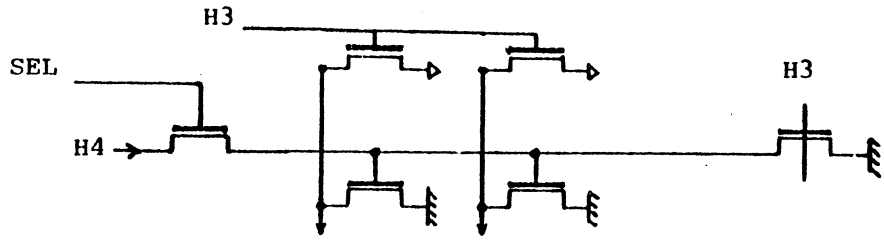
La sortie S a toute la phase T3 pour monter à "1", afin d'être amplifiée au début de la phase T4 par un amplificateur commandé par H4.

III-A-3/ **Lecture de la ROM**:

Les bits de sortie de la ROM peuvent être préchargés en T3, pour être déchargés éventuellement en T4, s'ils font parti du mot sélectionné d'une part, et si d'autre part ils "valent Zéro".

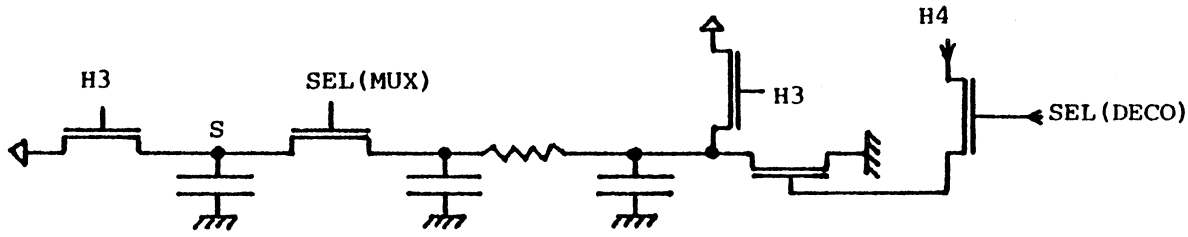
Un multiplexage est généralement effectué en sortie d'une ROM

(choix d'un bit parmi MUX); les commandes des multiplexeurs ne sont actives qu'en T4 puisqu'elles proviennent aussi du décodage de l'adresse.



Remarque 1: La sortie des multiplexeurs doit elle-aussi être préchargée en H3 (parce que toutes les commandes sont à "0" pendant H3).

Remarque 2: Le multiplexeur classique nécessite un seul transistor par ligne. On a donc le modèle suivant:



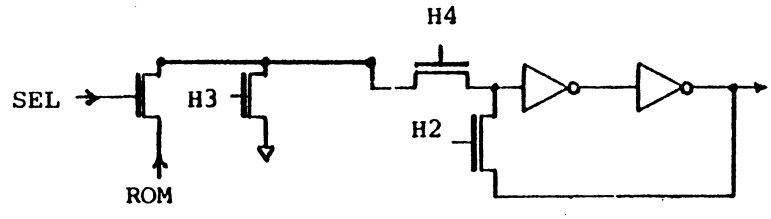
Le niveau électrique **bas** en S peut être un "mauvais Zéro", parce que trop près de VTH; le structure éléctrique est celle d'un NON-ET (NAND) avec une résistance R importante entre la sortie S et la masse (environ 10 Ohm pour 1000 lambda de métal [règles du CMP 83]).

III-A-4/ **Les sorties de la ROM**:

Un bit de sortie de la ROM, après le multiplexeur, est préchargé en T3, et sa valeur peut passer à "0", ou rester à "1" pendant le temps T4.

a/- Si l'on place un REGISTRE pour mémoriser le mot lu dans la ROM,

ce dernier peut être controlé par H4 et on doit placer un inverseur pour empêcher la décharge de la ligne de ROM.

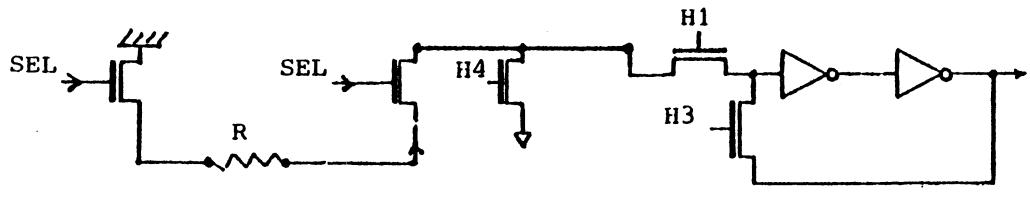


Comme ce registre est systématiquement chargé en temps T4, on peut le construire "semi-statique", et le rafraîchir en T2.

Remarque 1: On peut remplacer la précharge et l'inverseur par un générateur de courant.

Remarque 2: La lecture de la ROM étant lente, on charge systématiquement un "1" dans le registre au début de T4; ce "1" peut se transformer en "0" vers la fin de T4, si le bit lu vaut "0":
on assiste donc un passage systématique à "1" de tous les bits en sortie du registre, CE QUI PEUT ETRE GENANT. D'autre part, il faut également vider la charge accumulée systématiquement au point e, à travers trois interrupteurs et une résistance (voir III-A-3, remarque 2).

b/- Ce même registre peut n'être chargé qu'au temps T1, c'est-à-dire lorsque le bit de ROM est effectivement lu et stabilisé:
on réalise ainsi un échantillonnage au début de T1, avec la nécessité de laisser le chemin vers la masse ouvert, pour écouler la charge éventuelle accumulée dans l'entrée du registre.



Cet écoulement peut être assez long, à cause de la résistance R
=> donc le temps d'établissement du registre est long, lorsqu'il doit passer de "1" à "0".
Il faut d'autre part laisser les sélections actives pendant T1, donc commander leur amplificateur par (H4 + H1).

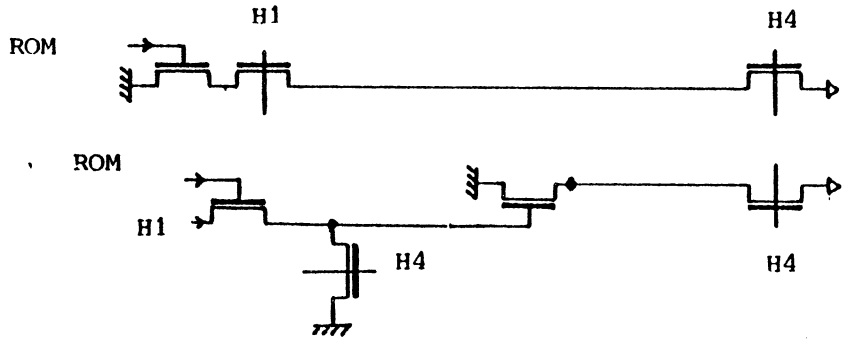
CONCLUSION:

Les solutions a/ et b/ présentent des inconvénients:
a/ ne marche que si la décharge est assez rapide dans T4,
b/ introduit un retard tel que la nouvelle valeur du bit n'est disponible qu'au "milieu" de T1.

c/- **Solution dynamique:**

Sachant que les barrières temporelles sont réalisées par la mutuelle exclusion des phases, on peut se passer de registre pour mémoriser la sortie de la ROM, en considérant que les bits sortant de la ROM peuvent directement être utilisés en entrée d'opérateurs qui les "décodent".

Nous rappelant que les sorties de la ROM sont stables à la fin de T4, elles peuvent être utilisées pour décharger, en T1, des lignes préchargées en T4.

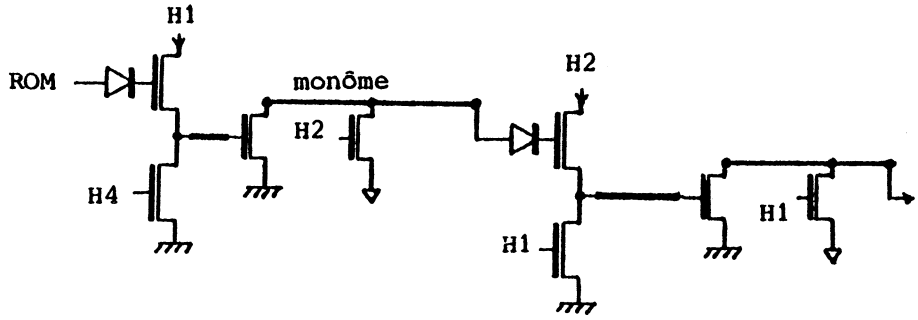


CONSEQUENCE: Les entrées des décodeurs peuvent être directement déchargées dès le début du temps T1, ce qui permet d'accélérer les calculs critiques pour l'adresse suivante: on dispose ainsi du début de T1 jusqu'à la fin de T2 pour calculer cette adresse.

III-B- CALCUL DE L'ADRESSE SUIVANTE

III-B-1/- DECODAGE DE L'INSTRUCTION DE SEQUENCEMENT

Un champ particulier de la microinstruction indique la manière de calculer l'adresse suivante: Ce champ est décodé pendant T1 et sa valeur peut être combinée, dans une matrice de PLA-ET, avec une Condition dite "de classe 1" (voir chapitre MICROPROGRAMMATION). Cette dernière Condition doit être présente en début de T1, ou être calculée pendant le début du temps T1.
Ainsi, au début de T2, une matrice de PLA-OU peut faire la synthèse des commandes qui peuvent être exécutées:
- soit dès qu'elles sont calculées (pendant T2),
- soit au début du temps T3, mais avec des restrictions.



Si les deux PLA-ET et PLA-OU sont "rapides", ils peuvent donner leurs sorties à la fin de T1, et ces dernières sont utilisables en T2 (amplifiées par H2).

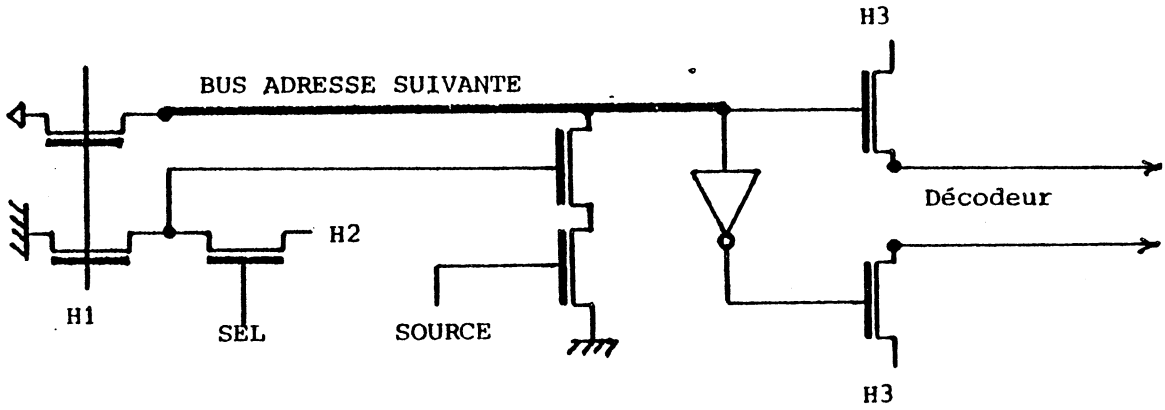
CONSEQUENCE: En supprimant un registre, on utilise la phase qui suit celle de la lecture des bits de ROM pour décoder les champs de la microinstruction.

REMARQUE: Le PLA de décodage peut se trouver loin de la ROM. Une capacité importante peut donc exister et l'utilisation d'un amplificateur commandé par H1 se justifie sur chaque bit de sortie de ROM, de manière à avoir un schéma équivallent pour tous les bits, indépendant de leur fonction et de leur "charge".

PAR CONTRE: on n'a plus besoin d'amplificateurs sur les entrées des PLA-ET: seulement un interrupteur à la masse et une précharge, ce qui est topologiquement très intéressant.

III-B-2/ Exécution des commandes:

Le BUS ADRESSE SUIVANTE (BAS) qui relie toutes les sources d'adresses possibles géographiquement éloignées, est préchargé en T1, pour être déchargé en T2 par la source sélectionnée.
On dispose ainsi du temps T2 uniquement pour positionner ce BUS, en sélectionnant une SOURCE qui est autorisée à le décharger.

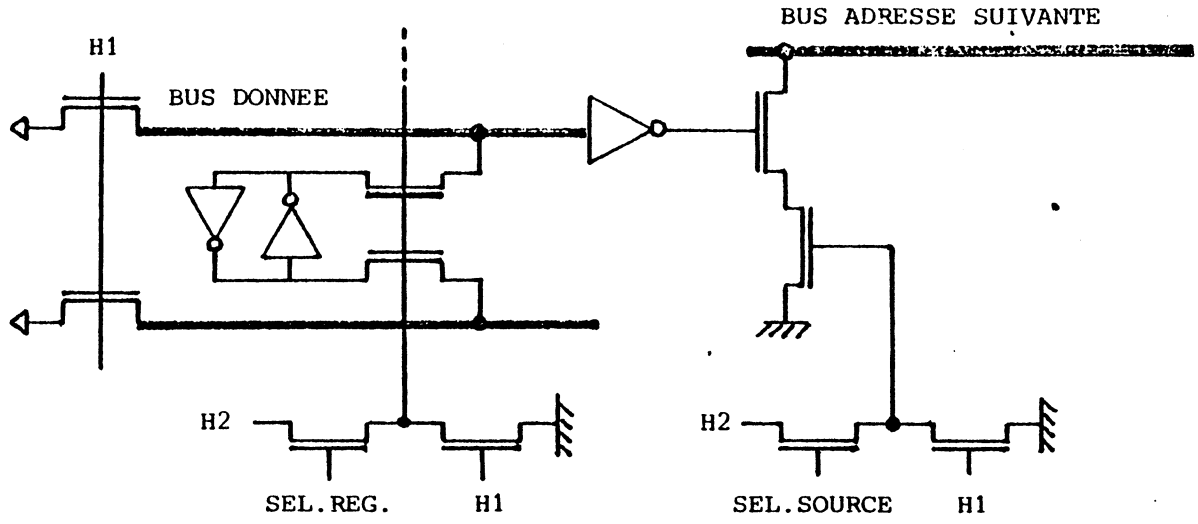


III-B-3/ LIAISON AVEC UNE PARTIE OPERATIVE DE SEQUENCEMENT

Le Micro-Compteur Ordinal et la PILE des adresses de retour sont implantés dans des points de mémorisation connectables à un double bus, qui fonctionne de la manière suivante:

- T1 = précharge
- T2 + T3 = lecture
- T3 = écriture

La connexion est donc la suivante:



IV- C O N C L U S I O N

Ce chapître a tenté de montrer l'intérêt de la mémorisation dynamique dans un contexte multi-phase: la présentation théorique précédente est en cours d'implantation, pour construire des blocs fonctionnels, assemblés par l'assembleur de silicium LUBRICK, qui seront appelés par le compilateur de silicium.

Cette approche systématique des problèmes électriques permet de simplifier la topologie de la Partie Contrôle: le BUS Adresse Suivante, qui est préchargé en un point unique du circuit, peut avoir une longueur variable et se connecter aux différentes sources de l'adresse suivante ainsi qu'aux PLA de génération des éclatements. D'autre part, la prise en compte de conditions de séquencement peut être réalisée rapidement, quel que soit le nombre de conditions à tester, car les bits qui sortent de la mémoire ROM ou du PLA de synthèse des commandes entrent directement dans les PLA de combinaisons.

CONCLUSION

CONCLUSION

Nous avons cherché à présenter plusieurs aspects des recherches menées dans le domaine complexe de la conception des Circuits Intégrés, en proposant une approche par la Compilation.

La méthodologie présentée dans l'introduction fait apparaître deux démarches:

- l'une, ascendante, part des contraintes de la topologie, et s'appuie sur un Assembleur de Silicium souple et puissant, capable de générer les dessins des masques d'un circuit dense, à partir d'une programmation de blocs fonctionnels;

- l'autre, descendante, consiste à compiler des descriptions fonctionnelles de haut-niveau, pour les projeter sur des structures matérielles.

La convergence des deux démarches ne peut se produire que si des structures d'accueil, comme celles qui sont définies dans les deuxième et troisième parties, peuvent être définies à la fois par leur topologie et par leur fonctionnalité.

Contrairement à d'autres approches, on propose ici de distinguer les problèmes de compilation et ceux de dessin, tout en soulignant l'importance des notions d'architecture d'ordinateur qui dirigent les choix fonctionnels et influent sur les choix topologiques.

La conception d'un ordinateur n'est pas chose aisée !
La tâche est encore plus complexe lorsqu'il s'agit d'intégrer cet ordinateur sur une puce de moins d'un centimètre-carré !
Les nombreux problèmes à résoudre trouveront peut-être ici une approche de solution ?

REFERENCES

[AMD] ADVANCED MICRO DEVICES, "AMD 2900 family"

[ANC-82] F.ANCEAU, J.P.SCHOELLKOPF : "CAPRI: A silicon compiler for circuits defined by their behaviour".
Workshop on Silicon Compilers, Edinburgh, July 1982.

[ANC-83] F.ANCEAU : "CAPRI: A design methodology and a silicon compiler for VLSI circuits specified by algorithms."
Proceedings of the third Caltech conference on VLSI, March 21-23, 1983, pp 15-31.

[ANC-83] F.ANCEAU, R.REIS : "Complex integrated circuit design strategy".
IEEE JSSC VOL SC17, June 1982.

[ANTO-84] C.ANTONINO : rapport Contrat LETI, à paraître

[ANVAR-75] F.ANCEAU, G.BAILLE, J.P.SCHOELLKOPF: "Machine informatique électronique destinée à l'exécution parallèle d'un langage post-fixé".
brevet ANVAR No 75 29 473, septembre 1975.

[BAILLE-83] G.BAILLE : "Machine PASC-HLL: réalisation avec des microprocesseurs en tranches d'une unité centrale multiprocesseur adaptée au langage PASCAL".
Thèse d'université, INPG, Grenoble, Octobre 1983

[BELLMAC] "MAC-32 a 32-bit microprocessor",
ISSCC Fevrier 1981

[BERTRAND-83] F.BERTRAND : "Simulation statique (comportementale) du 8048, génération des PLAS".
rapport de contrat MATRA-HARRIS, 1983.

[BURROUGHS] Burroughs Co. "B1700 Hardware Description"

[CHAILLOUX] J.CHAILLOUX· "Définition de Le Lisp"
Rapport INRIA. 1983.

[COREEN-80] Y.S.PARK : "Etude de I-PASCAL et de sa compilation"
Thèse de Docteur Ingénieur. INPG. 1980.

[DELAUNAY] M.DELAUNAY. J.F.GRABOWIECKI: "Note technique d'utilisation du transformateur de grammaire L11"
rapport de recherche IMAG No 234. septembre 1980.

[ECL] "Simulation en ONAP du multiprocesseur PASC-HLL"
Mémoire d'ingénieur. Ecole Centrale de Lyon. 1979.

[FORTIER-74] R.FORTIER : "Etude et définition du langage intermédiaire et d'une machine formelle multiprocesseur orientée vers l'exécution du langage PASCAL".
Thèse de 3ème cycle. USMG. Grenoble. Octobre 1975.

[GALLAIS-84] R.GALLAIS : " Participation à la réalisation d'un microprocesseur 16 bits interprétant le P-CODE".
Thèse d'ingénieur CNAM. GRENOBLE. mars 1984.

[GRIFFITHS-PELLETIER] "Un compilateur de compilateur"
rapport interne IMAG. 1970.

[GROSS] R.R.GROSS : "Silicon compilers : a critical survey".
Department of computer science. UNIVERSITY OF NORTH CAROLINA. CHAPEL HILL. NC 27514. MAY 31. 1983.

[GUYOT-75] A.GUYOT: "un compilateur de microprogramme"
Thèse de 3ème cycle. USMG. Grenoble. 1975.

[INRIA-82] HEINZ: "un extracteur de schéma électrique"
Thèse de 3ème cycle. INRIA. 1982.

[IRENE] S.MARINE. F.ANCEAU. K.JAHIDI "IRENE: un langage de description de circuits intégrés logiques"
rapport de recherche IMAG No 356. mars 1983.

[JERRAYA-83] A.JERRAYA: "COMFOR: Une nouvelle approche pour la vérification des masques des circuits intégrés"
Thèse de Docteur-Ingénieur. INPG. novembre 1983.

[JERRAYA-GUYOT] " LUCIE. LANGAGES ET PROGRAMMES 1981 "
Rapport interne IMAG 1981.

[KRASICKY] R.KRASICKI " Etude d'un optimiseur logique de PLA ".
Thèse d'Ingénieur CNAM. Grenoble. octobre 1983.

[MSINC] "MSINC User Manual". Un simulateur électrique.

[MACPITTS] "Generating custom high performance VLSI. designs from succinct algorithmic descriptions".
MIT conference on advanced research in VLSI. 1982.

[MARINE] S.MARINE. F.ANCEAU. K.JAHIDI : " IRENE : Un langage de description de circuits intégrés logiques".
rapport de recherche IMAG No 356. mars 1983.

[MEAD] C.MEAD. L.CONWAY: "Introduction to VLSI design"
Addison-Weslev Editeur.

[OB-82] M.OBREBSKA: " Etude comparative de différentes méthodes de conception des parties contrôle des microprocesseurs".
Thèse de Docteur-Ingénieur. INPG. Grenoble. Juin 82.

[PAOLA] S.CHUQUILLANQUI, T.PEREZ SEGOVIA " PAOLA : A tool for topological optimization of large PLAs".
Proceedings of 19th DAC, LAS VEGAS, NEVADA 1982.

[ONAP] M.VERAN, "ONAP User Manual"
Centre scientifique CII GRENOBLE.

[REIS] R.REIS: " TESS : Evaluateur topologique prédictif pour la génération automatique de plans de masse de circuits VLSI".
Thèse de Docteur-Ingénieur, INPG, janvier 1983 Grenoble.

[RISC] SHERBURNE and Co: "datapath design for RISC".
Conference on advanced research in VLSI, MIT, Janvier 1982.

[SCH-77] J.P.SCHOELLKOPF: "Machine PASC-HLL: Définition d'une architecture pipe-line pour une unité centrale adaptée au langage PASCAL".
Thèse de 3ème cycle, INPG, Grenoble, Juin 1977.

[SCH-79] G.BAILLE, J.LAURENT, J.P.SCHOELLKOPF : " Présentation du système MADAM ".
contrat IRIA No 78204, juin 1979.

[SPICE] "SPICE User Manual", un simulateur électrique

[SUZIM] A.SUZIM: "Etude des parties opératives à éléments modulaires pour processeurs monolithiques".
Thèse de Docteur-Ingénieur, INPG, Grenoble, novembre 1981.

[SYCOMORE] "Présentation du projet national Sycomore"
INRIA, THOMSON, BULL.

[WALLICH] P.WALLICH: "Technology 83: Automation (design/manufacturing)".
IEEE Spectrum 20, january 1983, pp 55-58.

[WERNER] J.WERNER: "The silicon compiler: Panacea, wishful thinking, or old hat ?".
VLSI DESIGN 3, 5, september/october 1982, pp 46-52.

[WIRTH] N.WIRTH, K.JENSEN: "Définition du langage P-CODE" (listing)

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 5 de l'arrêté du 16 avril 1974

VU les rapports de présentation de Messieurs

. F. ANCEAU, Professeur
. J. BOREL, Directeur EFCIS
. J. VUILLEMIN, Professeur

Monsieur Jean-Pierre SCHOELLKOPF

est autorisé à présenter une thèse en soutenance en vue de l'obtention du grade de DOCTEUR D'ETAT ES SCIENCES.

Fait à Grenoble, le 5 mars 1985

Le Président de l'U.S.M.G

Le Président
M. TANCHE

UNIVERSITÉ SCIENTIFIQUE ET MÉDICALE DE GRENOBLE

Le Président de l'I.N.P.-G

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

P.O. le Vice-Président,