Réseau de cellules intégré : étude d'architectures pour des applications de CAO de VLSI

Renaud Cornu-Emieux

THESE

présentée par

Renaud CORNU-EMIEUX

pour obtenir le titre de **DOCTEUR**
de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**
(arrêté ministériel du 5 juillet 1984)

RESEAU DE CELLULES INTEGRE :

Etude d'architectures pour des applications de CAO de VLSI

Date de soutenance : 27 septembre 1988

Composition du Jury :

Messieurs	Jacques MOSSIERE	*Président*
	Guy MAZARE	
	Jean Paul SANSONNET	
	Daniel ETIEMBLE	*Rapporteurs*
	Patrice QUINTON	

Thèse préparée au sein du Laboratoire de Génie Informatique

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président: Georges LESPINARD — Année 1988

Professeurs des Universités

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

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis

Personnes ayant obtenu le diplôme d'HABILITATION A DIRIGER DES RECHERCHES

BECKER	Monique
BINDER	Zdenek
CHASSERY	Jean-Marc
CHOLLET	Jean-Pierre
COEY	John
COLINET	Catherine
COMMAULT	Christian
CORNUEJOLS	Gérard
COULOMB	Jean-Louis
DALARD	Francis
DANES	Florin
DEROO	Daniel
DIARD	Jean-Paul
DION	Jean-Michel
DUGARD	Luc
DURAND	Madeleine
DURAND	Robert
GALERIE	Alain
GAUTHIER	Jean-Paul
GENTIL	Sylviane
GHIBAUDO	Gérard
HAMAR	Sylvaine
HAMAR	Roger
LADET	Pierre
LATOMBE	Claudine
LE GORREC	Bernard
MADAR	Roland
MULLER	Jean
NGUYEN TRONG	Bernadette
PASTUREL	Alain
PLA	Fernand
ROUGER	Jean
TCHUENTE	Maurice
VINCENT	Henri

CHERCHEURS DU C.N.R.S

Directeurs de recherche 1ère Classe

CARRE	René	LANDAU	Ioan
FRUCHART	Robert	VACHAUD	Georges
HOPFINGER	Emile	VERJUS	Jean-Pierre
JORRAND	Philippe		

Directeurs de recherche 2ème Classe

ALEMANY	Antoine	KLEITZ	Michel
ALLIBERT	Colette	KOFMAN	Walter
ALLIBERT	Michel	KAMARINOS	Georges
ANSARA	Ibrahim	LEJEUNE	Gérard
ARMAND	Michel	LE PROVOST	Christian
BERNARD	Claude	MADAR	Roland
BINDER	Gilbert	MERMET	Jean
BONNET	Roland	MICHEL	Jean-Marie
BORNARD	Guy	MUNIER	Jacques
CAILLET	Marcel	PIAU	Monique
CALMET	Jacques	SENATEUR	Jean-Pierre
COURTOIS	Bernard	SIFAKIS	Joseph
DAVID	René	SIMON	Jean-Paul
DRIOLE	Jean	SUERY	Michel
ESCUDIER	Pierre	TEODOSIU	Christian
EUSTATHOPOULOS	Nicolas	VAUCLIN	Michel
GUELIN	Pierre	WACK	Bernard
JOUD	Jean-Charles		

Personnalités agréées à titre permanent à diriger des travaux de recherche (décision du conseil scientifique)

ENSEEG

CHATILLON	Christian	SARRAZIN	Pierre
HAMMOU	Abdelkader	SIMON	Jean-Paul
MARTIN GARIN	Régina		

ENSERG

BOREL	Joseph

ENSIEG

DESCHIZEAUX	Pierre	PERARD	Jacques
GLANGEAUD	François	REINISCH	Raymond

ENSHMG

ROWE	Alain

ENSIMAG

COURTIN	Jacques

EFP

CHARUEL	Robert

C.E.N.G

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

Laboratoires extérieurs :

C.N.E.T

DEVINE	Rodericq	MERCKEL	Gérard
GERBER	Roland	PAULEAU	Yves

****

****

***

UNIVERSITE SCIENTIFIQUE TECHNOLOGIQUE ET MEDICALE
DE GRENOBLE

Président de l'Université :

M. PAYAN Jean Jacques

ANNEE UNIVERSITAIRE 1987 - 1988

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ERE CLASSE

ARNAUD Paul	Chimie Organique
ARVIEU Robert	Physique Nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AURIAULT Jean Louis	Mécanique
AYANT Yves	Physique Approfondie
BARBIER Marie Jeanne	Electrochimie
BARJON Robert	Physique Nucléaire I.S.N.
BARNOUD Fernand	Biochimie Macromoléculaire Végétale
BARRA Jean René	Statistiques - Mathématiques Appliquées
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BELORISKY Elie	Physique
BENZAKEN Claude	Mathématiques Pures
BERARD Pierre	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
CHARDON Michel	Géographie
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEGRANGE Charles	Zoologie
DEMAILLY Jean Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPORTES Charles	Chimie Minérale
DOLIQUE Jean Michel	Physique des Plasmas
DOUCE Roland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FONTAINE Jean Marc	Mathématiques Pures
GAGNAIRE Didier	Chimie Physique
GERMAIN Jean Pierre	Mécanique
GIRAUD Pierre	Géologie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean René	Mathématiques Pure

KAHANE André, détaché	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées
LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre Jean	Mathématiques Appliquées
LEBRETON Alain	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LHOMME Jean	Chimie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean Marie	Sciences Nucléaires I.S.N.
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PAYAN Jean Jacques	Mathématiques Pures
PEBAY PEYROULA Jean Claude	Physique
PERRIER Guy	Géophysique
PIERRARD Jean Marie	Mécanique
PIERRE Jean Louis	Chimie Organique
RENARD Michel	Thermodynamique
RINAUDO Marguerite	Chimie C.E.R.M.A.V.
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
SOUTIF Michel	Physique
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VALENTIN Jacques	Physique Nucléaire I.S.N.
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2EME CLASSE

ADIBA Michel	Mathématiques Pures
ANTOINE Pierre	Géologie
ARMAND Gilbert	Géographie
BARET Paul	Chimie
BLANCHI Jean Pierre	S.T.A.P.S.
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BRUANDET Jean François	Physique
BRUGAL Gérard	Biologie
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
COURT Jean	Chimie
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaires
GILLARD Roland	Mathématiques Pures
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GUIGO Maryse	Géographie
GUMUCHIAN Hervé	Géographie
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
JARDON Pierre	Chimie
JOSELEAU Jean Paul	Biochimie
KERCKHOVE Claude	Géologie
LONGEQUEUE Nicole	Sciences Nucléaires I.S.N
LUCAS Robert	Physique
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
NEMOZ Alain	Thermodynamique C.N.R.S. C.R.T.B.T.
OUDET Bruno	Mathématiques Appliquées
PECHER Arnaud	Géologie
PELMONT Jean	Biochimie
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Jean Claude	Physique du Solide
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
RICHARD Jean Marc	Physique
RIEDTMANN Christine	Mathématiques Pures
ROBERT Gilles	Mathématiques Pures
ROBERT Jean Bernard	Chimie Physique
SARROT REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L'I.U.T. 1

PROFESSEURS DE 1ERE CLASSE

BUISSON Roger	Physique I.U.T. 1
DODU Jacques	Mécanique Appliquée I.U.T. 1
NEGRE Robert	Génie Civil I.U.T. 1
NOUGARET Marcel	Automatique I.U.T. 1
PERARD Jacques	E.E.A. I.U.T. 1

PROFESSEURS DE 2EME CLASSE

BOUTHINON Michel	E.E.A. I.U.T. 1
CHAMBON René	Génie Mécanique I.U.T. 1.
CHEHIKIAN Alain	E.E.A. I.U.T. 1
CHENAVAS Jean	Physique I.U.T. 1
CHOUTEAU Gérard	Physique I.U.T. 1
CONTE René	Physique I.U.T. 1
GOSSE Jean Pierre	E.E.A. I.U.T. 1
GROS Yves	Physique I.U.T. 1
KUHN Gérard, détaché	Physique I.U.T. 1
MAZUER Jean	Physique I.U.T. 1
MICHOULIER Jean	Physique I.U.T. 1
MONLLOR Christian	E.E.A. I.U.T. 1
PEFFEN René	Métallurgie I.U.T. 1
PERRAUD Robert	Chimie I.U.T. 1
PIERRE Gérard	Chimie I.U.T. 1
TERRIEZ Jean Michel	Génie Mécanique I.U.T. 1
TOUZAIN Philippe	Chimie I.U.T. 1
VINCENDON Marc	Chimie I.U.T. 1

PROFESSEURS DE PHARMACIE

AGNIUS DELORD Claudine	Physique	Faculté la Tronche
ALARY Josette	Chimie Analytique	Faculté la Tronche
BERIEL Hélène	Physiologie et Pharmacologie	Faculté la Tronche
CUSSAC Max	Chimie Therapeutique	Faculté la Tronche
DEMENGE Pierre	Pharmacodynamie	Faculté la Tronche
FAVIER Alain	Biochimie	C.H.R.G.
JEANNIN Charles	Pharmacie Galenique	Faculté Meylan
LATURAZE Jean	Biochimie	Faculté la Tronche
LUU DUC Cuong	Chimie Générale	Faculté la Tronche
MARIOTTE Anne-Marie	Pharmacognosie	Faculté la Tronche
MARZIN Daniel	Toxicologie	Faculté Meylan
RENAUDET Jacqueline	Bactériologie	Faculté la Tronche
ROCHAT Jacques	Hygiène et Hydrologie	Faculté la Tronche
SEIGLE MURANDI Françoise	Botanique et Cryptogamie	Faculté Meylan
VERAIN Alice (Chargée)	Pharmacie galénique	Faculté Meylan

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXCEPTIONNELLE ET 1ERE CLASSE

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatre - Puériculture	C.H.R.G.
BEZES Henri	Orthopédie - Traumatologie	Hopital Sud
BONNET Jean-Louis	Ophtalmologie	C.H.R.G.
BOUCHET Yves	Anatomie	Faculté la Merci
	Chirurgie Générale et Digestive	C.H.R.G.
BUTEL Jean	Orthopédie - Traumatologie	C.H.R.G.
CHAMBAZ Edmond	Biochimie	C.H.R.G.
CHAMPETIER Jean	Anatomie Topographique et Appliquée	C.H.R.G.
CHARACHON Robert	O.R.L.	C.H.R.G.
COLOMB Maurice	Immunologie	Hopital Sud
COUDERC Pierre	Anatomie Pathologique	C.H.R.G.
DELORMAS Pierre	Pneumophtisiologie	C.H.R.G.
DENIS Bernard	Cardiologie	C.H.R.G.
GAVEND Michel	Pharmacologie	Faculté la Merci
HOLLARD Daniel	Hématologie	C.H.R.G.
LATREILLE René	Chirurgie Thoracique/Cardiovasculaire	C.H.R.G.
LE NOC Pierre	Bactériologie - Virologie	Faculté la Merci
MALINAS Yves	Gynécologie et Obstétrique	C.H.R.G.
MALLION Jean Michel	Médecine du Travail	C.H.R.G.
MICOUD Max	Clinique Médicale/Maladies Infectieuses	C.H.R.G.
MOURIQUAND Claude	Histologie	Faculté la Merci
PARAMELLE Bernard	Pneumologie	C.H.R.G.
PERRET Jean	Neurologie	C.H.R.G.
RACHAIL Michel	Hépato-Gastro-Entérologie	C.H.R.G.
DE ROUGEMONT Jacques	Neurochirurgie	C.H.R.G.
SARRAZIN Roger	Clinique Chirurgicale	C.H.R.G.
STIEGLITZ Paul	Anesthésiologie	C.H.R.G.
TANCHE Maurice	Physiologie	Faculté la Merci
VIGNAIS Pierre	Biochimie	Faculté la Merci

PROFESSEURS 2EME CLASSE

BACHELOT Yvan	Endocrinologie	C.H.R.G.
BARGE Michel	Neurochirurgie	C.H.R.G.
BENABID Alim Louis	Biophysique	[illegible]
BENSA Jean Claude	Immunologie	Hopital Sud
BERNARD Pierre	Gynécologie - Obstétrique	C.H.R.G.
BESSARD Germain	Pharmacologie	Abidjan
BOLLA Michel	Radiothérapie	C.H.R.G.
BOST Michel	Pédiatrie	C.H.R.G.
BOUCHARLAT Jacques	Psychiatrie Adultes	Hopital Sud
BRAMBILLA Christian	Pneumologie	C.H.R.G.
CHIROSSEL Jean Paul	Anatomie - Neurochirurgie	C.H.R.G.
COMET Michel	Biophysique	Faculté la Merci
CONTAMIN Charles	Chirurgie Thoracique/Cardiovasculaire	C.H.R.G.
CORDONNIER Daniel	Nephrologie	C.H.R.G.
COULOMB Max	Radiologie	C.H.R.G.
CROUZET Guy	Radiologie	C.H.R.G.
DEBRU Jean Luc	Médecine Interne et Toxicologie	C.H.R.G.
DEMONGEOT Jacques	Biostatistiques/Informatique Médicale	Faculté la Merci
DUPRE Alain	Chirurgie Générale	C.H.R.G.
DYON Jean François	Chirurgie Infantile	C.H.R.G.
ETERRADOSSI Jacqueline	Physiologie	Faculté la Merci
FAURE Claude	Anatomie et Organogénèse	C.H.R.G.
FAURE Gilbert	Urologie	C.H.R.G.
FOURNET Jacques	Hépato-Gastro-Entérologie	C.H.R.G.
FRANCO ALain	Médecine Interne	C.H.R.G.
GIRARDET Pierre	Anesthésiologie	C.H.R.G.
GUIDICELLI Henri	Chirurgie Générale et Vasculaire	C.H.R.G.
GUIGNIER Michel	Thérapeutique et Réanimation Médicale	C.H.R.G.
HADJIAN Arthur	Biochimie	Faculté la Merci
HALIMI Serge	Endocrinologie et Maladies Métaboliques	C.H.R.G.
HOSTEIN Jean	Hépato-Gastro-Entérologie	C.H.R.G.
HUGONOT Robert	Médecine Interne	C.H.R.G.
JALBERT Pierre	Histologie - Cytogénétique	Faculté la Merci
JUNIEN-LAVILLAUROY Claude	O.R.L.	C.H.R.G.
KOLODIE Lucien	Hématologie Biologique	C.H.R.G.
LETOUBLON Christian	Chirugie Générale	C.H.R.G.
MACHECOURT Jacques	Cardiologie et Maladies Vasculaires	C.H.R.G.
MAGNIN Robert	Hygiène	C.H.R.G.
MASSOT Christan	Médecine Interne	C.H.R.G.
MOUILLON Michel	Ophtalmologie	C.H.R.G.
PELLAT Jacques	Neurologie	C.H.R.G.
PHELIP Xavier	Rhumatologie	C.H.R.G.
RACINET Claude	Gynécologie-Obstétrique	Hopital Sud
RAMBAUD Pierre	Pédiatrie	C.H.R.G.
RAPHAEL Bernard	Stomatologie	C.H.R.G.
SCHAERER René	Cancérologie	C.H.R.G.
SEIGNEURIN Jean Marie	Bactériologie - Virologie	Faculté la Merci
SELE Bernard	Cytogénétique	Faculté la Merci
SOTTO Jean Jacques	Hématologie	C.H.R.G.
STOEBNER Pierre	Anatomie Pathologique	C.H.R.G.
VROUSOS Constantin	Radiothérapie	C.H.R.G.

Ce travail a été réalisé grâce au soutien du groupement C³ du CNRS et avec l'aide du Centre Norbert Segard du CNET.

Je tiens à remercier

Monsieur **Jacques MOSSIERE**, Directeur de l'ENSIMAG et du LGI, de l'honneur qu'il m'a fait en acceptant de présider le jury de cette thèse.

Monsieur **Guy MAZARE**, Professeur à l'ENSIMAG, de m'avoir accueilli dans son équipe et d'avoir assuré l'encadrement de cette thèse.

Messieurs **Daniel ETIEMBLE**, Professeur, Directeur de l'UFR d'Informatique à l'Université Pierre et Marie Curie (Paris VI) et **Patrice QUINTON**, Directeur de Recherche à l'IRISA, Responsable du Pôle Architecture du GRECO-PRC C^3, qui ont accepté d'être rapporteurs de cette thèse et qui m'ont aidé par leurs conseils.

Monsieur **Jean Paul SANSONNET**, Ingénieur à la CGE, Responsable du projet MAIA , Directeur du PRC "Architecture de Machines Nouvelles", qui a accepté de faire parti du jury de cette thèse.

Mes remerciements vont aussi à

Jean Luc RAINARD, Ingénieur au CNET/CNS de Meylan, pour l'aide qu'il m'a apporté et les conseils judicieux qu'il m'a donnés lors de la conception et de la réalisation des circuits intégrés présentés dans cette thèse.

Résumé :

Le développement des techniques d'intégration permet de réaliser des circuits de 10^5 à 10^6 transistors et dans un futur proche des circuits encore plus complexes. Les problèmes de CAO deviennent donc de plus en plus ardus comme la simulation logique ou le placement. Cette même évolution nous autorise à réaliser des machines parallèles très puissantes pour résoudre ces problèmes.

Nous proposons l'architecture d'un réseau cellulaire asynchrone. Ce réseau, composé de NxN cellules dont chacune est physiquement connectée à ses quatre voisines, dispose d'un mécanisme de communication permettant l'acheminement de messages d'une cellule quelconque à n'importe quelle autre. Un circuit intégré incluant une réseau de 2x2 cellules dédié à la simulation logique a été réalisé.

Utilisant cette architecture cellulaire, nous avons développé un placeur, qui à partir d'une configuration initiale, minimise la longueur des connexions par échange de paires. Nous avons abordé la manière dont le placement pourrait être amélioré par la méthode du recuit simulé.

Ces deux applications différentes de l'architecture cellulaire nous permettent de constater que beaucoup de parties sont communes aux deux circuits. Nous énonçons certaines règles de façon à rendre la conception plus rapide et plus sûre.

Summary :

Development of integration techniques has increased today's Integrated Circuit complexity up to 10^5 to 10^6 transistors and let us assume that ICs will become more powerful in the near future. That is, some important steps of the CAD process such as logical simulation and placement are getting critical and hard to manage.

On the other hand this evolution has leaded to the realization of highly parallel machines able to handle such time consuming tasks. We propose a new asynchronous cellular architecture based on a regular array of NxN automata each of which is physically connected to its four neighbours a distributed transmission message mechanism allowing communications between any two cells of the array. We have designed a 2x2 cell array prototype IC dedicated to logical simulation.

We have also developped a placement engine by modifying cell algorithms. It acts from an initial placement map and minimizes connection lengths by using a distributed pairwise exchange algorithm. We present how a simulated annealing approach can improved its performances.

These two different applications of the cellular architecture allow us to note that a lot of parts are the same in the two circuits. We state some rulls to make the conception faster and safer.

A mes Parents

Introduction

INTRODUCTION

Les structures et les algorithmes hautement parallèles font, à l'heure actuelle, l'objet de beaucoup de recherches. La possibilité d'intégrer de plus en plus de transistors par unité de surface joue le rôle de cause et d'effet dans ce phénomène ! Un rôle de cause : l'intégration augmentant, les temps de calcul pour simuler les nouveaux circuits à intégrer, augmentent jusqu'à atteindre les limites de l'acceptable si on utilise des moyens de calcul classiques ; un rôle d'effet : l'intégration augmentant, il est devenu possible de réaliser, non seulement des logiciels parallèles, mais aussi des architectures parallèles puissantes et performantes.

Notre étude d'une architecture cellulaire destinée, entre autre, à accélérer de manière câblée les différentes phases de la conception des circuits intégrés, se place dans ce cadre de développement de l'intégration qui nous permettra dans un futur assez proche de réaliser des réseaux cellulaires comportant un grand nombre de cellules intégrées dans un circuit, ou sur tranche, ou encore réalisés en montant de nombreux circuits identiques sur une plaque.

Cette étude a été commencée par J.P. Bernard [BER85] qui a défini les grandes lignes de l'architecture de la machine cellulaire. Les simulations fonctionnelles de cette machine cellulaire ont été réalisées par Y.Ansade [ANS88]. L'aiguilleur de cette machine et l'interface avec l'ordinateur hôte ont été définis et dessinés par P. Objois [OBJ88]. La conception et la réalisation de la partie traitement de la cellule spécialisée pour la simulation logique, celles des interfaces du réseau, ainsi que l'étude de la faisabilité du placeur m'étaient dévolues et font l'objet de cette thèse. D'autres applications sont en cours d'étude par diverses personnes de l'équipe circuit du LGI, leur énumération sera faite dans le chapitre 2.

Le chapitre 1, après un bref rappel de ce qu'est l'intégration aujourd'hui, énumère les différents types d'accélérateurs câblés existants, en présentant quelques machines parmi les plus significatives. Les réseaux cellulaires y sont aussi présentés succinctement.

Le chapitre 2 présente les grandes fonctionalités et l'architecture du réseau cellulaire que nous avons conçu, en insistant sur le mécanisme original de communication inter-cellulaire.

Le chapitre 3 expose les étapes qui nous ont menés jusqu'à la réalisation d'un prototype de réseau composé d'une matrice de 2 x 2 cellules interconnectées, dédié à la simulation logique. L'architecture interne d'une cellule y est décrite de manière complète.

Le chapitre 4 pose les bases d'une autre réalisation dans le domaine de l'aide à la conception de circuits intégrés, puisque l'architecture d'un placeur reprenant le principe du réseau cellulaire décrit au chapitre 2, y est présentée.

Le chapitre 5 essaie d'ouvrir la voie vers une méthodologie de conception de circuits cellulaires de ce type, en faisant le point sur les différentes parties du circuit réalisées qui pourront être réutilisées dans d'autres applications.

Enfin, la conclusion tire les enseignements du cheminement que nous avons suivi de la spécification du réseau cellulaire à sa réalisation, dresse quelques perspectives pour l'amélioration de l'application placement du réseau et montre l'intérêt d'une telle architecture.

Chapitre 1

Pourquoi des accélérateurs matériels ?

POURQUOI DES ACCELERATEURS MATERIELS ?

1.1. LES TECHNOLOGIES AUJOURD'HUI

Depuis la réalisation des premiers circuits intégrés MOS, au début des années 1960, le niveau d'intégration des circuits a doublé tous les ans. Cette tendance marque un peu le pas, à l'heure actuelle. Aujourd'hui, il est néanmoins possible d'intégrer 10^5 à 10^6 transistors pour un même circuit. L'objectif en VLSI étant d'atteindre 10^7 transistors dans les années à venir. D'autres techniques (WSI) permettront à moyen terme de réaliser des circuits plus importants encore [**CAN**85].

Cette évolution a permis la réalisation de circuits intégrés de plus en plus complexes. La vérification de ces circuits par simulation nécessite des outils de plus en plus puissants. Pour un simulateur, si n est le nombre de portes du circuit à simuler, le temps de calcul nécessaire varie de $O(n^2)$ à $O(n^3)$ [**AMB**87].

Plusieurs solutions ont été apportées pour remédier à ces temps de calcul prohibitifs. Une solution consiste à améliorer les machines classiques (de type Von Neuman) en utilisant des technologies de plus en plus performantes.

L'autre solution est basée sur le principe du "diviser pour régner". Ce principe consiste en une réalisation simultanée de plusieurs tâches simples au lieu de réaliser une seule tâche complexe. Sa réalisation peut différer :

- selon le nombre de tâches que l'on veut exécuter en parallèle : l'éventail allant de 2 taches un peu moins complexes à un nombre très élevé de processus élémentaires,

- selon le nombre de processeurs physiques sur lesquels ces tâches vont être distribuées.

Ces outils sont généralement groupés en deux familles : les algorithmes tournant sur machines générales et les accélérateurs câblés. Il existe une très grande dépendance entre le temps de calcul et la flexibilité de ces outils : une machine qui a une très grande flexibilité (possibilité de programmer beaucoup d'applications) est forcément plus lente qu'une machine dédiée à une application particulière [**BLA**84].

1.2. LOGICIEL DE CONCEPTION

La conception de circuits intégrés VLSI nécessite l'utilisation de plusieurs outils :

- simulateurs (fonctionnels, logiques, électriques, ...)
- éditeurs (logique, électrique, layout, ...)
- placeurs, routeurs
- générateurs automatiques (de PLA, de RAM, ...)
- vérificateurs (DRC, ERC, extracteurs, ...)

Certains d'entre eux sont très coûteux en temps de calcul. Ce sont pour ces outils que la plupart des accélérateurs ont été réalisés. Les logiciels de conception comportent généralement 4 phases :

- **Préparation** : les données sont traduites de la forme intelligible sous laquelle l'utilisateur les a communiquées à la machine, en une forme compréhensible par celle-ci.

- **Exécution** : l'algorithme est déroulé.

- **Edition des résultats** : les résultats subissent la transformation inverse de la phase de préparation.

- **Mise au point du circuit** : modification des entrées et du circuit à simuler.

Les phases de préparation et d'acquisition des résultats varient à peu près linéairement avec la taille du circuit. Par contre l'exécution peut varier de l'ordre de n^3 par rapport à la taille du circuit [SHE87]. Un accélérateur câblé doit donc au moins accélérer l'exécution, on parle dans ce cas d'**accélérateur ponctuel**. Si les 4 phases sont accélérées il s'agit d'un **accélérateur global**.

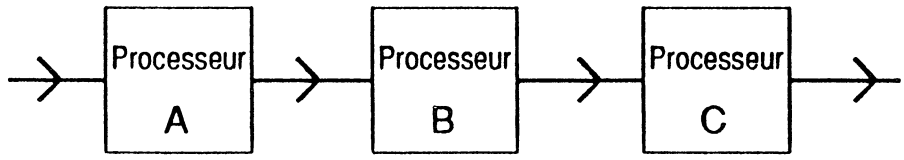
1.3. LES MACHINES DE C.A.O. VLSI

Nous avons vu que la solution apportée à l'accroissement des temps de calcul est l'utilisation du parallélisme. Ce parallélisme peut se présenter sous deux formes.

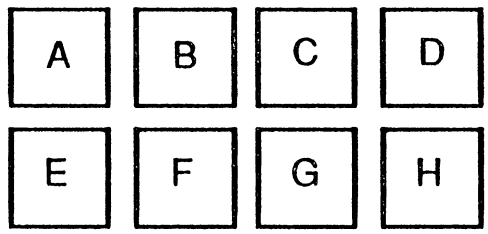
La première consiste à faire tourner sur plusieurs processeurs, un algorithme séquentiel, en réalisant en même temps des séquences de tâches successives. L'organisation pipeline est un exemple d'une telle architecture. Dans ce cas, la mémoire peut soit être partagée entre les différents processeurs, soit être distribuée.

La deuxième solution consiste, pour la simulation logique par exemple, à faire simuler par chaque processeur une porte ou un ensemble de portes. On parle dans ce cas d'architectures purement multiprocesseurs sur lesquelles les algorithmes sont distribués.

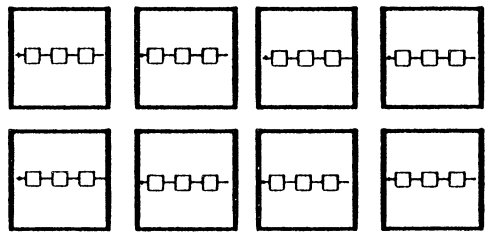
Il existe des architectures qui combinent ces deux types de parallélisme en partitionnant les circuits à simuler et en réalisant une simulation pipeline de chaque sous-circuit.



Architecture pipeline



Architecture "purement" multiprocesseur sans mémoire partagée



Architecture mixte

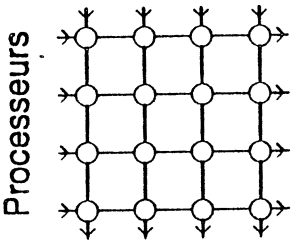
Figure 1.2. Les différentes architectures d'accélérateurs

Les architectures de ces deux derniers types nécessitent des mécanismes de communication entre les processeurs. On peut en distinguer deux grandes familles :

- celles qui utilisent un bus, dans ce cas, le bus étant une ressource critique, une seule communication entre processeurs peut se dérouler à un temps t ce qui peut retarder le fonctionnement de la machine.

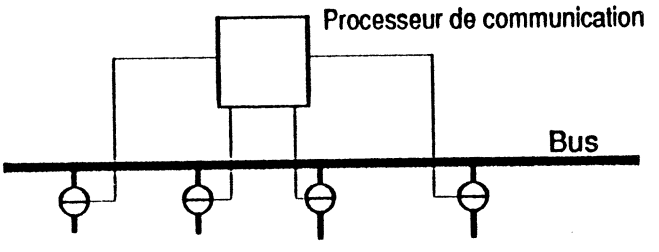
- celles qui utilisent un réseau d'interconnexion.

Il est évident que dans le cas de ces deux architectures, la mémoire doit être distribuée sur les n processeurs pour que les performances soient maximales.

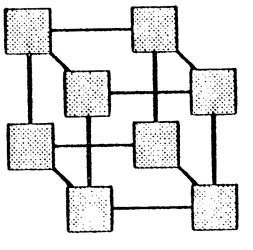


Les cercles représentent les cellules ;
le réseau peut travailler en acheminement de messages
ou en connexion de circuits (crossbar)

Connexion par matrice



Connexion par un bus



Chaque carré représente un processeur dans lequel
on trouve un dispositif de communication.

Connexion de type hypercube (n=3)

Figure 1.3. connexions entre processeurs

Des accélérateurs câblés sont, aujourd'hui, commercialisés par une douzaine de compagnies dans le monde : Aida, Cadnetix, Daisy, HHB systems, Ikos, LSI Logic, Mentor, Silicon Solutions, Simulog, Teradyne, Valid, Xcat et bien sûr ZYCAD [MIL87]. Simultanément, de nouvelles architectures et de nouveaux algorithmes sont en cours de développement dans les laboratoires de recherche.

Les premières machines ont été Zycad, YSE et HAL. Il est intéressant de noter que Boeing avait commencé des recherches dans ce sens, dans les années 1960, mais les coûts avaient été jugés exorbitants et le projet enterré ...

- **Des machines purement pipeline :**

L'accélérateur Realfast de Valid [TRA86] est constitué de deux processeurs spécialisés, l'un gérant les temps de simulation, l'autre évaluant les événements. Ces deux processeurs ont leur mémoire spécifique. L'accélérateur destiné à traiter des circuits de plus de 2,5 millions de porte est commercialisé et il est destiné à fonctionner sur des stations de travail.

La station de travail Megalogician de Daisy [GIN83] comporte elle aussi un accélérateur pipeline composé de trois processeurs spécialisés connectés à l'hôte. Ces trois processeurs sont connectés entre eux en anneau, ils communiquent par des FIFO rapides, insérés dans cet anneau. Le premier processeur joue le rôle d'échéancier ; le second possède en mémoire les informations sur la connectique du réseau à simuler, de plus il stocke les états et les forces des nœuds ; enfin le troisième effectue les évaluations. Ces trois processeurs disposent de leur mémoire locale.

Un accélérateur développé à l'université de Stanford [VLA87] permet de faire tourner SPICE2. Cet accélérateur est composé de 2

processeurs : un processeur "entier" et un processeur virgule flottante. La mémoire est partagée entre les deux processeurs.

Plusieurs machines de routage ont été réalisées avec cette architecture : un routeur basé sur l'algorithme de Lee a été décrit et réalisé à l'Université de Manchester [SPI87], cet accélérateur est composé de 4 processeurs dont un sert d'interface avec l'hôte, qui communiquent par un bus ; un autre, réalisé à l'Université du Minnesota [WON87], composé de trois pipelines de trois processeurs chacun, réalise un routage basé sur le même principe.

- **Des machines "purement" multiprocesseurs :**

Les réseaux cellulaires seraient à classer dans ce type d'architectures, mais ils seront traités dans un paragraphe suivant. Il existe peu de machines purement multiprocesseurs. En général, les processeurs travaillant sur une partie du circuit ont une architecture pipeline, on retrouve donc ces machines classées dans le sous paragraphe suivant. Notons quand même la machine de Bell [LEV82] qui permet une simulation au niveau porte et au niveau fonctionnel.

- **Des machines "mixtes" :**

Ce type d'architecture est très développé à l'heure actuelle. Zycad fut un des pionniers dans la commercialisation de ces produits. Aujourd'hui, Zycad produit une famille de simulateurs logiques à échéanciers. L'architecture du simulateur LE1000 [ZYC84] est une combinaison des deux architectures présentées précédemment. Le circuit à simuler est réparti sur 1 à 16 modules de simulation, selon la configuration adoptée. A l'intérieur de ces modules l'algorithme se déroule sur 5 processeurs fonctionnant en pipeline, la communication entre les modules étant réalisée par un bus. La machine exécute une simulation logique 12 états avec retard variable, des simulations temporelles et des simulations de panne.

Un système complet de simulation et de test appelé Zilos **[HAN87]** tourne sur un accélérateur d'un type semblable.

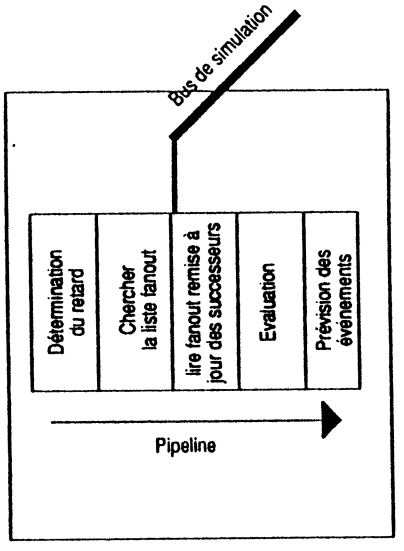


Figure 1.4. Architecture du LE1000

La machine de Yorktown **[DEN82]** est une machine de simulation logique qui est composée de 256 processeurs interconnectés grâce à un processeur de communication. Comme pour toutes les machines de cette famille, le circuit à simuler est décomposé en blocs, chacun étant affecté à l'un de ces processeurs. Chaque processeur peut simuler jusqu'à 4096 fonctions logiques à 4 entrées et 1 sortie. La simulation comporte 4 états et elle est à délai nul ou unitaire. Les performances annoncées sont de 3 milliards d'évaluations par seconde. Ce chiffre semble très optimiste : chaque processeur effectue une évaluation en 80 ns, les 256 processeurs travaillant en parallèles effectueraient 3,2 Milliards d'évaluation par seconde s'ils n'avaient pas à communiquer entre eux, ce qui semble une hypothèse très forte. La machine est composée de deux types de processeurs : les processeurs logiques composés de 8 processeurs en pipeline, qui peuvent évaluer 16 fonctions logiques différentes et les processeurs "array " pour la

simulation des RAM et des ROM. YSE permet des vérifications au niveau logique et au niveau interrupteur (switch).

MARS [**AGR**87], [**NOR**87] (Microprogrammable Accelerator for Rapid Simulation) développé par AT&T Bell Labs est un autre accélérateur fonctionnant avec les deux types d'architectures. Il est composé de plusieurs "Cluster" identiques (jusqu'à 256) interconnectés entre eux grâce à un réseau de type 8-cube. Chaque nœud du réseau est en fait une partie d'un "cluster". Ceux-ci fonctionnent en pipeline (possibilité d'avoir jusqu'à 14 tâches en parallèle). Chaque cluster peut exécuter des simulations logiques à délais multiples. Les communications entre cluster sont réalisées de manière asynchrone. Chaque canal entre cluster est composé de 16 bits.

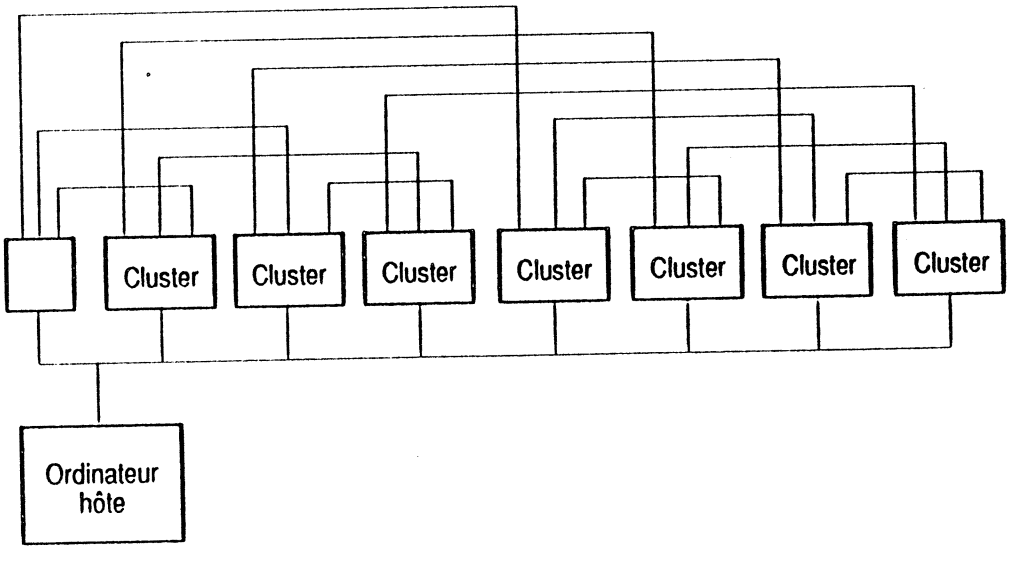


Figure 1.5. L'architecture de MARS

La machine HAL [**SAS**83] et aujourd'hui HAL2 [**TAK**86] sont des machines spécialisées pour la simulation de gros systèmes logiques. HAL2 est composé de 29 processeurs logiques qui comportent chacun un pipeline de 2 processeurs, 2 processeurs de mémoire servant à simuler des ROM et des RAM et un processeur

de contrôle. Ils communiquent entre eux par une matrice de communication.

Beaucoup d'architectures de ce type sont étudiées dans les laboratoires de recherches et dans les grandes compagnies. Citons : l'accélérateur TEGAS [BAR80] ; le Mach1000 de Silicon Solution [MOT85] ; Le Proteus-1 d'accelerated solution Corp. [WOO87] une architecture développée au LAM de Montpellier [LAN86] destinée à la simulation de panne ; une machine japonaise de simulation [HIR87] ; la machine "Awsim-2" [LEW86] de l'Université de Toronto destinée à la simulation au niveau interrupteur.

Nous pouvons aussi classer dans cette catégorie, la machine de simulation logique Luckylog [LUC87] développée chez Aptor à base de Transputer [INM87]. Il s'agit d'un simulateur logique à échéancier réparti sur 1 à 128 unités de simulation comportant chacune 3 transputers.

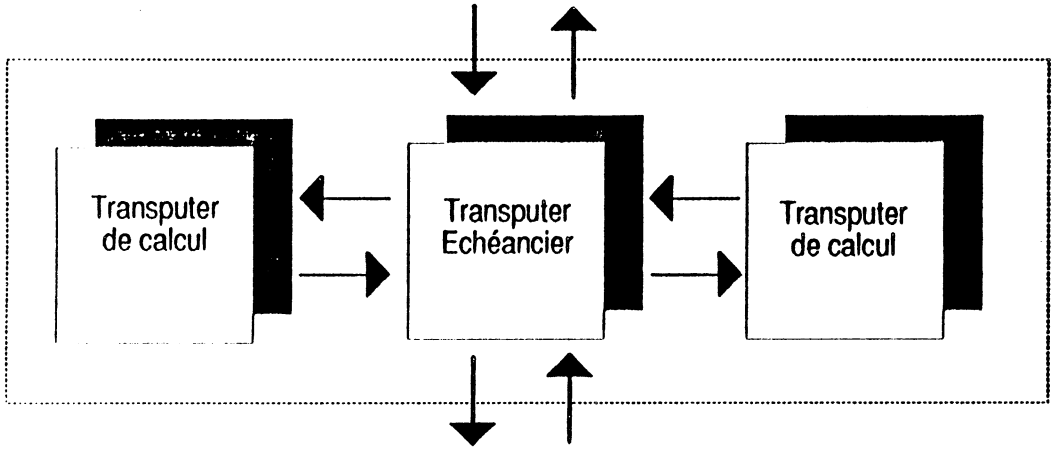


Figure 1.6. Une unité de simulation de Luckylog

1.4. LES ACCELERATEURS CELLULAIRES

Les machines cellulaires sont composées d'un grand nombre de processeurs élémentaires comportant chacun leur mémoire locale. Ces machines sont destinées à des problèmes du type réseau de nœuds interconnectés. Le fonctionnement est basé sur le principe de l'affectation d'un nœud à un processeur au moins. Il va de soit que le traitement ne doit nécessiter qu'une information locale.

Les premières applications qui ont été étudiées, concernent le traitement d'image : un pixel d'une image est associé à un processeur. Il faut donc disposer d'un réseau comportant au moins autant de processeurs que de pixels. Unger [**UNG**58] a présenté une machine dans laquelle chaque cellule était connectée à ses 8 voisines. Ce type d'application est en cours d'étude pour notre réseau cellulaire [**LAT**88b].

NTT a développé un réseau cellulaire appelé Adaptative Array Processor, composé de 256 x 256 processeurs 1 bit sur lesquelles différentes applications ont été implémentées [**WAT**86].

Ce type d'architecture semble être intéressante pour certains problèmes de C.A.O. de circuits VLSI : placement, routage, simulations... car beaucoup d'entre eux se présentent sous la forme d'un réseau de nœuds interconnectés.

Plusieurs machines existent pour ce type d'applications. Blank a proposé plusieurs architectures pour ces problèmes [**BLA**82] : un "Bit Map Processor" réseau composé de 1024 x 1024 cellules et un "Virtual Bit Map Processor" de 32 x 32 cellules avec une mémoire plus grande de façon à replier une grille plusieurs fois sur le réseau. Plusieurs applications ont été développées sur ces réseaux : DRC et Routage (algorithme de Lee).

Les réseaux systoliques [**KUN**82], [**MEL**86], sont un autre type d'architecture cellulaire. Bien que peu utilisés comme accélérateurs câblés dans les applications de CAO, ceux-ci ont la faculté de réaliser les fonctions d'analyse numérique nécessitées par les simulateurs électriques.

1.5. LES PERFORMANCES

Les différentes machines présentées annoncent de 100 000 à 1 milliard d'événements par seconde. Il est très difficile de comparer entre elles ces machines. La diversité des architectures et des algorithmes implémentés rend ceci très délicat. On peut simplement noter que bien souvent les performances annoncées sont sujettes à caution : le fait de mettre 16 modules en pipeline ne multiplie certainement pas les performances par 16 !

Beaucoup d'évaluations de performances des accélérateurs, ne donnent pas de précisions sur les temps de communication entre ceux-ci et l'ordinateur hôte qui les accueille. Il semble pourtant qu'il s'agit là d'un point critique.

Dans le même ordre d'idée, la phase de préparation des données et de partitionnement du circuit est bien souvent passée sous silence.

Voici quelques performances parmi les simulateurs présentés :

- MARS : 1 million d'événements par seconde
- MEGALOGICIAN : 100 000 événements par seconde
- ZILOS : jusqu'à 16 millions d'événements par seconde
- HAL2 : environ 20 millions d'événements par seconde
- AWSIM-2 : 2 millions d'événements par seconde
- ZYCAD : jusqu'à 16 millions d'événements par seconde
- YSE : un chiffre de plus de 1 milliard est annoncé.

Chapitre 2

Architecture de la machine cellulaire

ARCHITECTURE DE LA MACHINE CELLULAIRE

2.1. OBJET DE L'ETUDE

L'architecture de la machine que nous étudions entre dans la catégorie des architectures cellulaires non systoliques. La cellule de base de notre machine ne comporte que quelques milliers de transistors et dans les technologies à venir il sera possible de disposer de circuits de 16x16 cellules ou davantage.

Des communications générales

En général, les architectures cellulaires ou systoliques ne permettent aux cellules que de communiquer avec leurs 4 ou 8 plus proches voisines. Cela est adapté à un bon nombre de traitements très réguliers, du type : travail sur les matrices ou traitement du signal. Si l'algorithme implémenté le nécessite, il faut, pour transmettre des informations au-delà de la cellule voisine, que celle-ci la reçoive puis la réémette; ce travail est réalisé par la programmation de la cellule, donc par logiciel et fait partie de l'algorithme parallèle.

Il nous a semblé que de nombreux algorithmes ou applications n'avaient pas la belle régularité d'un traitement matriciel et nécessitaient donc que chaque cellule puisse communiquer avec un certain nombre de cellules, autres que leurs 4 voisines tout en n'étant physiquement connecté qu'à celles-ci.

L'originalité de notre étude est de permettre de traiter les problèmes de ce type dans lesquels le nombre des cellules destinataires est variable d'une cellule à l'autre, mais reste en moyenne assez faible (de 2 à 16) et pour lesquels le graphe des communications entre cellules peut être quelconque.

Asynchronisme

Un autre aspect original de notre étude est que ces cellules peuvent fonctionner de façon asynchrone les unes par rapport aux autres. Ainsi, en tolérant l'asynchronisme des comportements, on peut augmenter le parallélisme des traitements. De plus, les futures technologies nous permettrons d'intégrer de très grand réseaux, y distribuer une horloge unique semble déraisonnable d'où l'intérêt de considérer les cellules asynchrones. A la limite, elles pourraient avoir chacune leur oscillateur local, donc leur horloge interne. Même si cela n'est pas forcément très réaliste, nous avons tenu à ce que tous les mécanismes soient corrects, y compris dans le cas où tous les comportements seraient tout à fait asynchrones (hypothèses traditionnelles dans les systèmes de processus asynchrones). Ainsi, nous avons développé un système de communication par échanges de messages sur la base d'un rendez-vous "câblé" très simple de type producteur-consommateur.

2.2. RESEAU CELLULAIRE

2.2.1. Réseau de cellules

Le cœur de la machine cellulaire est constitué d'un réseau régulier de N x M cellules asynchrones, disposées dans une matrice plane. Dans le cas, d'une bonne localité des communications, il peut être intéressant de disposer d'une matrice rectangulaire. Les cellules sont connectées à leurs deux, trois ou quatre plus proches voisines par l'intermédiaire de registres tampons (fig. 2.1).

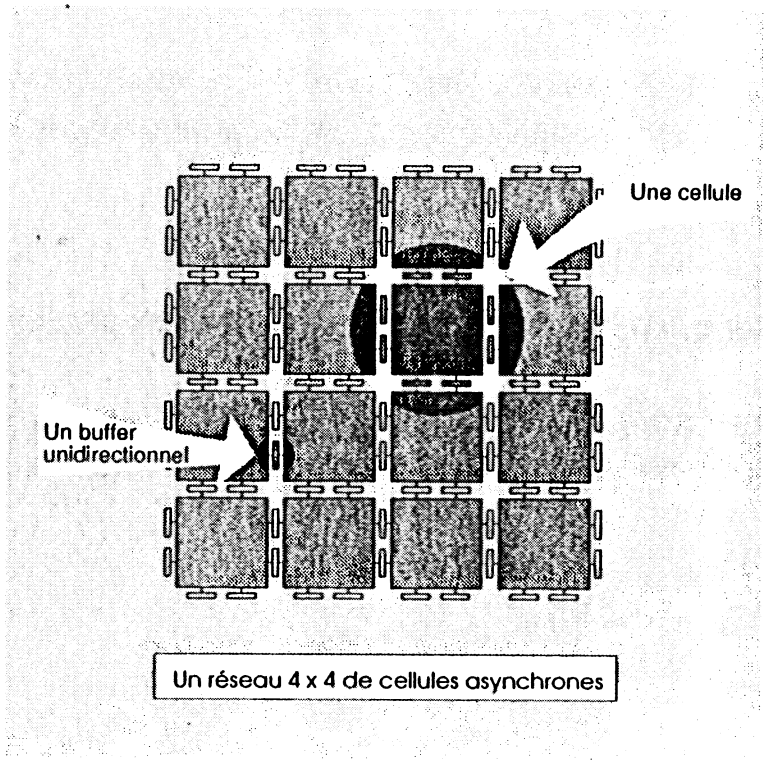


Figure 2.1. Réseau de Cellules

2.2.2. Tampons

Les tampons sont unidirectionnels et sont les seuls liens physiques entre les cellules. Chaque tampon assure la communication entre deux cellules voisines sur le modèle du producteur et du consommateur. Les messages échangés par les cellules sont de taille fixe. Le tampon ne peut contenir qu'un seul message à la fois.

Les spécifications de la communication entre les cellules sont les suivantes:

- un message donné n'est retiré qu'une seule fois, après avoir été déposé,

- les messages ne doivent pas être perdus : si un tampon contient déjà un message, on ne peut y déposer de message supplémentaire.

Pour ce faire, un tampon est composé de deux parties :

- une partie mémorisation,
- un drapeau.

2.2.3. Cellules

Les cellules sont les entités de calcul du réseau. Chaque cellule fonctionne de façon autonome. Cette cellule est elle même composée de deux parties autonomes.

L'une exécute un algorithme de transmission des messages. Elle utilise les informations fournies par les tampons d'entrée (périphériques ou internes) de la cellule. Les résultats sont stockés dans les tampons de sortie (périphériques ou internes si l'information est destinée à la cellule). La nature de l'algorithme de transmission des messages peut dépendre du type d'algorithme que l'on veut réaliser dans le réseau (simulation logique, placement parallèle, simulation neuronique, ...).

L'autre partie de la cellule, réalise la fonction pour laquelle la cellule a été conçue.

Chaque cellule comporte un tampon d'entrée et un tampon de sortie selon les quatre directions: **NORD**, **OUEST**, **SUD**, **EST**, soit **4 tampons périphériques** ainsi que **1 ou plusieurs tampons** par sens entre les deux parties qui la composent. Dans le langage

des processus asynchrones, nous dirons que les cellules jouent le rôle de producteur vis à vis de leurs tampons de sortie et le rôle de consommateur vis à vis de leurs tampons d'entrée.

Une cellule assure les fonctions suivantes :

- L'acheminement des messages provenant des tampons périphériques vers les cellules voisines pour les messages qui ne lui sont pas destinés,

- Une fonction de calcul pour les messages qui lui sont destinés.

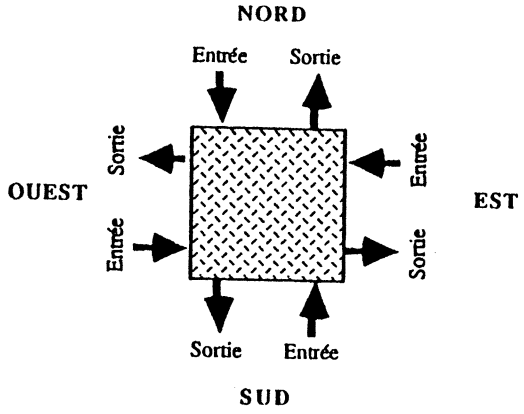


Figure 2.2. Cellule

2.3. MECANISME D'ACHEMINEMENT DE MESSAGES

L'originalité de notre machine est de permettre à chaque cellule du réseau d'échanger des messages avec n'importe quelle autre cellule du réseau. Les échanges de messages entre deux cellules peuvent se faire même s'il n'existe pas de connexions matérielles directes entre celles-ci. Les seules connexions réalisées entre les

cellules du réseau sont les connections locales: **NORD, OUEST, SUD, EST.**

Ceci est rendu possible par l'implémentation dans chaque cellule, d'un mécanisme d'acheminement de messages.

2.3.1. Messages inter-cellules

L'information qui est échangée par deux cellules est véhiculée par le réseau sous la forme d'un message, celui-ci est constitué de 2 champs (fig. 2.3) :

- Un champ **adresse**, contenant l'adresse relative de la cellule destinataire.

- Un champ **information**, contenant la valeur à transmettre.

La valeur du champ information dépend du type de problème traité par le réseau. Pour la simulation logique par exemple, ce champ contient des valeurs de simulation représentant **Vrai, Faux, Indéterminé**, ...

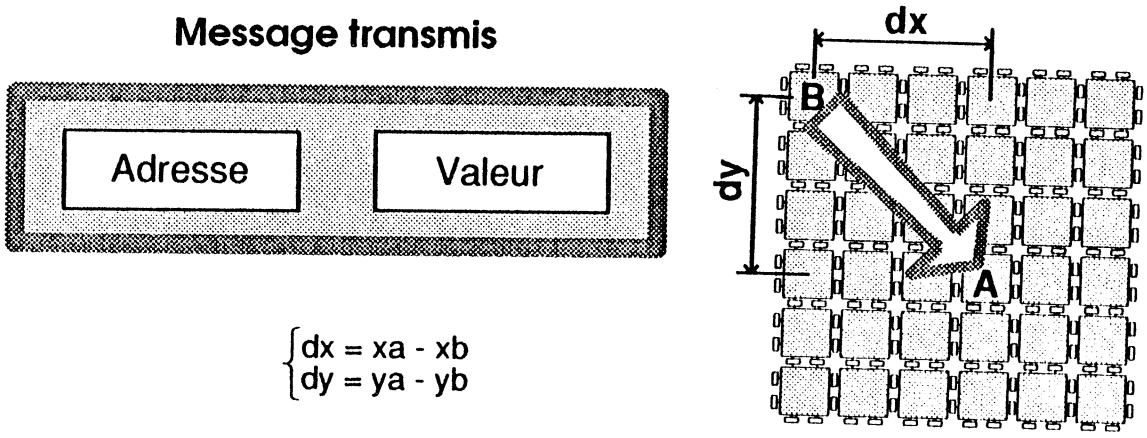


Figure 2.3. Message et système d'acheminement

Les informations contenues dans le champ adresse sont des valeurs signées. Elles représentent le déplacement relatif (dx, dy) restant à effectuer pour que le message arrive à destination. Dans certaines applications, lorsque les fonctions assurées par la cellule ne sont pas commutatives, on peut être amené à compléter l'adresse de la cellule par l'adresse locale du "point d'entrée" (port) de la cellule.

2.3.2. Cheminement des messages

Les messages sont émis par les cellules et transitent de cellule en cellule jusqu'à ce qu'ils arrivent à destination.

Le cheminement des messages est assuré par les cellules grâce à un algorithme interne fonctionnant sur le principe du déplacement relatif.

Principe du déplacement relatif:

Lorsqu'une cellule **i** veut envoyer une valeur vers une cellule **j**, elle émet un message dont le champ adresse est **(dx=(xj—xi), dy=(yj—yi))** (fig. 2.4). Les valeurs du champ adresse déterminent alors la direction vers laquelle le message doit être envoyé.

Lorsqu'une cellule reçoit un message, elle teste les valeurs dx, dy du message :

- si dx=0 et dy=0 alors le message est arrivé à destination et il est traité par la cellule.

- si dx≠0 ou dy≠0 alors la cellule transmet le message à une de ses quatre voisines après avoir modifié en conséquence la valeur du déplacement.

Plusieurs stratégies peuvent être envisagées pour cette transmission : de la plus simple (dx d'abord, dy en suite) à la plus complexe qui permet de contourner une cellule ne fonctionnant pas ; ce dernier type de stratégie s'appliquant dans le cadre de réalisation WSI.

2.4. TAMPONS : une implémentation matérielle du producteur consommateur

2.4.1. Description

Toutes les cellules du réseau fonctionnent de façon asynchrone. Les échanges de messages entre les cellules se font par l'intermédiaire de tampons de type **producteur-consommateur** sur la base d'un rendez-vous très simple.

Un tampon est composé d'un registre et d'un drapeau (fig. 2.5).

- Le **drapeau** mémorise une valeur booléenne : Vrai ou Faux qui informe les deux cellules de l'état du tampon : **Vide** ou **Plein**.

- Le **registre** mémorise un et un seul message envoyé par la cellule productrice.

Les tampons fonctionnent de façon complètement asynchrone à partir de signaux fournis par les cellules. Ces signaux représentent des valeurs booléennes et sont véhiculés par des équipotentielles. Une équipotentielle est utilisée pour chaque type de signal.

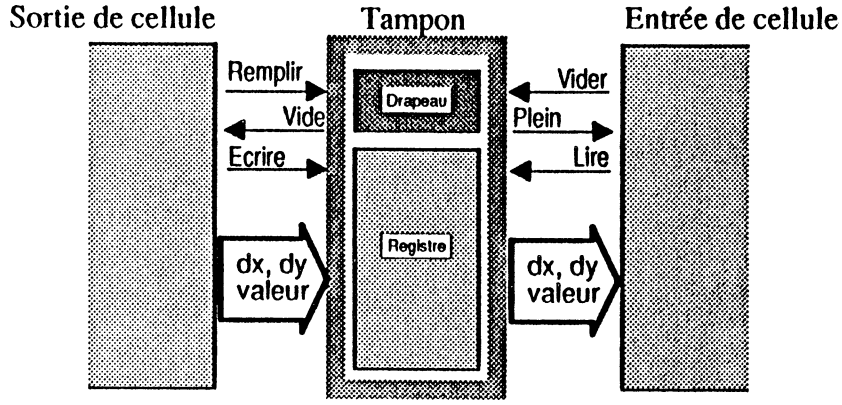


Figure 2.4. Schéma fonctionnel d'un tampon

2.4.2. Signaux utilisés

Les signaux émis par les cellules vers les tampons sont les suivants:

- **REMPLIR** est un signal envoyé au tampon par une cellule productrice pour l'avertir qu'un message a été produit et que le tampon est donc plein.

  => Drapeau := PLEIN

- **VIDER** est un signal envoyé au tampon par une cellule consommatrice pour l'avertir que le message a été consommé et donc que le tampon est vide.

  => Drapeau := VIDE.

- **ECRIRE** est un signal émis par la cellule productrice pour écrire un message dans le registre du tampon. Ce signal ne doit être émis que si le tampon est vide.

- **LIRE** est un signal qui permet à la cellule consommatrice de lire un message dans le registre du tampon. Ce signal ne doit être émis que si le tampon est plein.

Les signaux émis par un tampon vers les cellules "producteur" et "consommateur" sont les suivants :

- **VIDE** est une valeur booléenne qui informe la cellule productrice sur l'état du tampon : vide, pas vide.

- **PLEIN** est une valeur booléenne qui informe la cellule consommatrice sur l'état du tampon : plein, pas plein.

Bien que les tampons ne contiennent qu'un message à la fois, ces deux signaux ne sont pas strictement complémentaires => **VIDE** <> (NON **PLEIN**) : il peut arriver de façon transitoire que la configuration : **VIDE** = FAUX et **PLEIN** = FAUX se produise.

2.4.3. Modèle Producteur/Consommateur

Nous nous intéressons qu'au seul cas particulier d'un tampon qui ne peut contenir qu'un seul message à la fois. Le protocole habituel est décrit de la façon suivante :

PRODUCTEUR :

```
tant que vrai faire
  produire;
  tant que Drapeau ≠ VIDE faire
    attendre
  fin faire;
  ECRIRE;
  REMPLIR { Drapeau := PLEIN }
fin faire;
```

CONSOMMATEUR :

```
tant que vrai faire
  tant que Drapeau ≠ PLEIN faire
    attendre
  fin faire;
  LIRE;
  Consommer;
  VIDER { Drapeau := VIDE }
fin faire;
```

C'est de cette façon qu'est réalisé habituellement le mécanisme du producteur / consommateur entre deux processus asynchrones. Dans le cas de telles implémentations logicielles, il faut noter que :

- L'instruction "**tant que** drapeau ≠ VIDE **faire** attendre **fin faire**" est réalisée par une procédure du superviseur qui met le processus en attente.

- Les instructions **REMPLIR, VIDER, LIRE, ECRIRE** sont supposées atomiques, c'est à dire indivisibles et mutuellement exclusives. Ces conditions sont assurées

par la programmation des procédures système correspondantes.

2.4.4. Première implémentation matérielle

Les règles à respecter pour l'accès au tampon sont les suivantes (Règles du rendez-vous) :

R1: Une cellule ne peut écrire dans un tampon que s'il est **VIDE**.
ECRIRE si Drapeau = **VIDE**

R2: Une cellule ne peut lire un tampon que s'il est **PLEIN**.
LIRE si Drapeau = **PLEIN**

R3: Une cellule ne peut envoyer REMPLIR au Drapeau que lorsque le cycle d'écriture dans le registre du tampon est entièrement terminé.
REMPLIR => Drapeau := **PLEIN**

R4: Une cellule ne peut envoyer VIDER au Drapeau que lorsque le cycle de lecture du registre du tampon est entièrement terminé.
VIDER => Drapeau := **VIDE**

Ces quatre règles simples (**R1** ... **R4**), assurent l'exclusion mutuelle des cellules en lecture et en écriture sur le registre d'un tampon.

Un petit mécanisme (fig. 2.5) peut alors être réalisé à partir de ces règles.

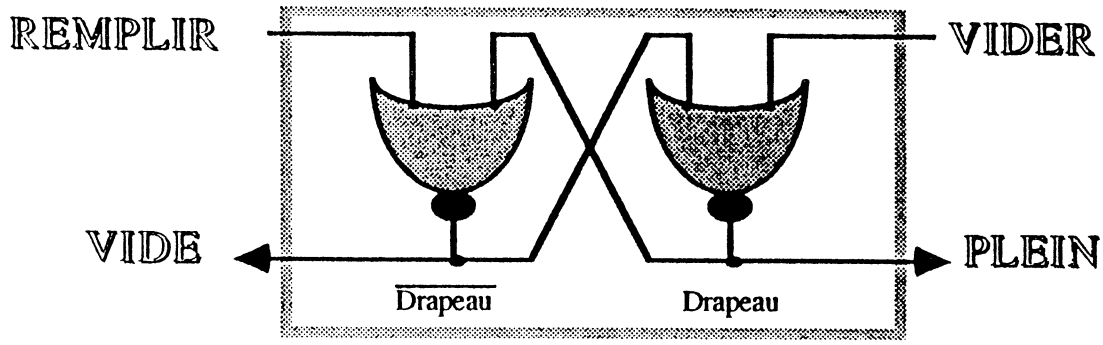


Figure 2.5. Première implémentation matérielle

2.4.5. Problème d'exclusion mutuelle

Cependant, un conflit entre les signaux **REMPLIR** et **VIDER** peut persister si les deux cellules concurrentes fonctionnent à des vitesses complètement différentes, ce qui n'est pas interdit dans notre réseau asynchrone.

En effet, **REMPLIR** et **VIDER** sont des signaux électriques, émis par les cellules, mais dont la durée d'impulsion dépend de l'horloge interne des cellules productrices en fonctionnement synchrone, et des dimensions ou des aléas technologiques en fonctionnement asynchrone ou combinatoire. Un conflit risque alors

de se produire si les 2 signaux **REMPLIR** et **VIDER** sont actifs au même instant, ce qui pourrait entraîner deux types de conflit : une perte de message ou plusieurs lectures du même message.

Exemple1: Plusieurs lectures du même message dues à un fonctionnement plus rapide du consommateur vis à vis du producteur.

Cas où : temps (**LIRE**+Consommer+**VIDER**) « temps (**REMPLIR**)

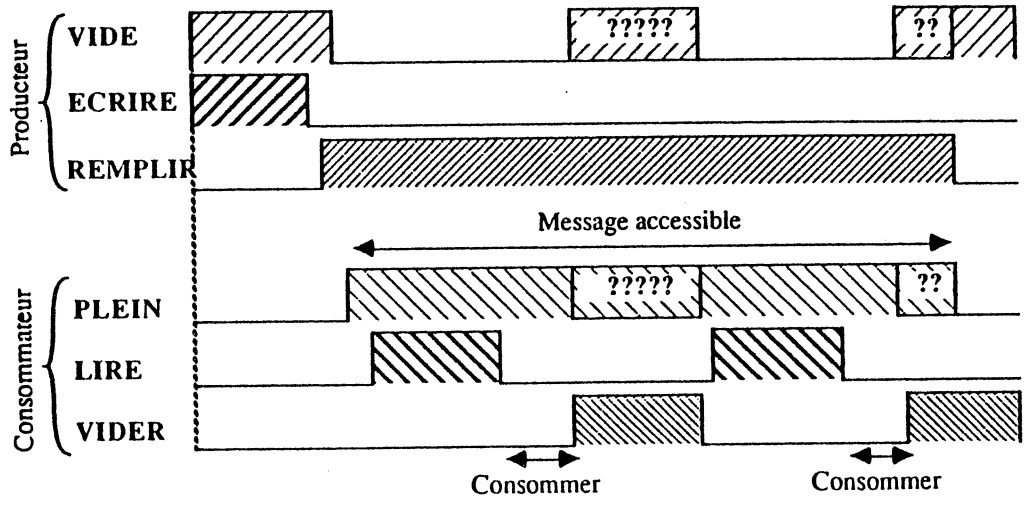


Figure 2.6. 1er conflit : plusieurs lectures d'un même message

Exemple2 : Perte de message (fig. 2.8) due à un fonctionnement plus rapide du producteur vis à vis du consommateur.

Cas où : temps (Produire + **ECRIRE** + **REMPLIR**) « temps (**VIDER**)

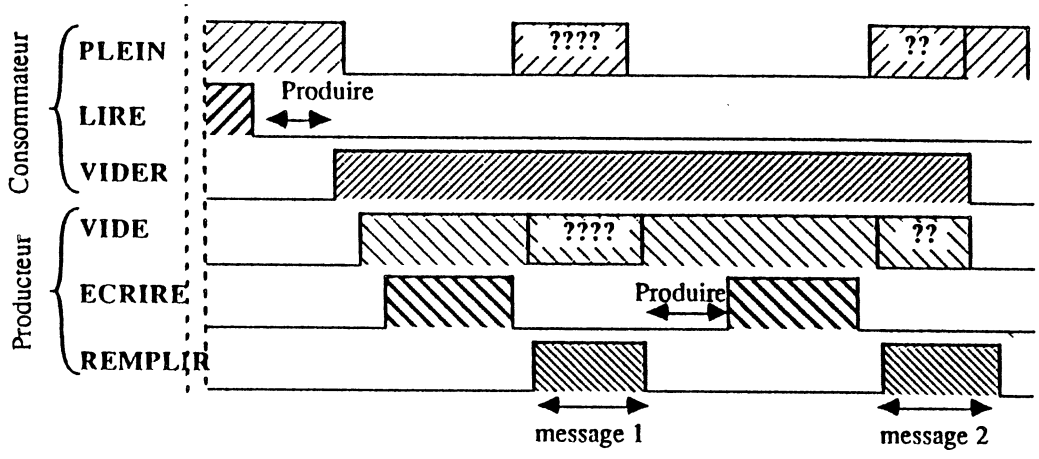


Figure 2.7. 1er conflit : pertes de messages

2.4.6. Nouvelles règles

Aussi, pour interdire aux deux signaux **REMPLIR** et **VIDER** de se recouvrir (i.e: pour ne pas perdre de message dans un tampon), il suffit d'interdire une lecture (resp. écriture) tant que le signal **REMPLIR** (resp. **VIDER**) est actif. En d'autres termes et d'après les règles **R1** ... **R4**, nous pouvons écrire les règles suivantes:

R5: L'opération **ECRIRE** ne peut être effectuée que si le tampon est plein (Drapeau=Vrai) et si le signal **REMPLIR** n'est pas actif.

R6: L'opération **LIRE** ne peut être effectuée que si le tampon est vide (Drapeau=Faux) et si le signal **VIDER** n'est pas actif.

Cela conduit à une nouvelle signification des valeurs des signaux **PLEIN** et **VIDE** en fonction de l'état du drapeau :

Logique booléenne			**Logique Complémentée**		
$\overline{\text{Drapeau}}$	=	$\overline{\text{Drapeau}}.\overline{\text{REMPLIR}}$	$\overline{\text{Drapeau}}$	=	$\overline{\text{Drapeau} + \text{REMPLIR}}$
Drapeau	=	$\text{Drapeau}.\overline{\text{VIDER}}$	Drapeau	=	$\overline{\overline{\text{Drapeau}} + \text{VIDER}}$
VIDE	=	$\overline{\text{Drapeau}} . \overline{\text{VIDER}}$	VIDE	=	$\overline{\overline{\text{Drapeau}} + \text{VIDER}}$
PLEIN	=	$\text{Drapeau} . \overline{\text{REMPLIR}}$	PLEIN	=	$\overline{\overline{\text{Drapeau}} + \text{REMPLIR}}$

2.4.7. Nouvelle implémentation matérielle

Le mécanisme de fonctionnement d'un drapeau peut alors être décrit facilement à l'aide de quatres portes **NOR**.

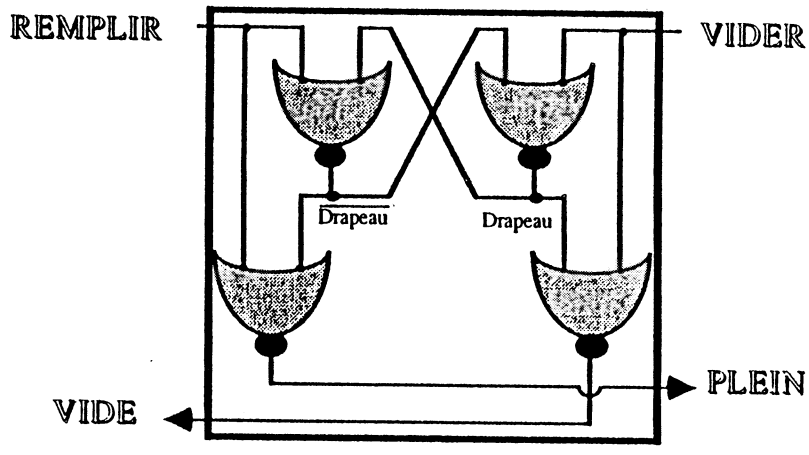


Figure 2.8. Nouvelle implémentation matérielle

2.5. CELLULE

2.5.1. Rôle

La cellule est l'entité de calcul du réseau. Elle peut exécuter plusieurs actions simultanément. Ces actions sont les suivantes:

Acheminement de message

- Assurer l'acheminement des messages qui lui arrivent par les tampons d'entrée périphériques (**Nord**, **Ouest**, **Sud** , **Est**), mais qui ne lui sont pas destinés.

- Recevoir un certain nombre de messages des cellules qui lui en envoient (i.e : des cellules qui lui sont connectées en amont).

Calcul

- Assurer le traitement des messages qui lui sont destinés.
  => exécution de la fonction de transfert associée à la cellule.

Emission de messages

- Emettre un certain nombre de messages résultats vers les cellules qui lui sont connectées en aval.

2.5.2. Description

Pour permettre un traitement plus efficace des fonctions : "**acheminement** de message", "**traitement** des messages et **émission** des résultats", une cellule est constituée de deux parties distinctes fonctionnant de façon asynchrone : "**l'Aiguilleur**" et "**l'Unité de traitement**".

Enfin, pour améliorer l'indépendance de ces deux parties et par conséquent, pour augmenter l'efficacité temporelle durant le fonctionnement du réseau, ces deux parties sont connectées par des registres tampons fonctionnant sur le même principe que les tampons inter-cellules.

Une cellule (fig. 2.10) est constituée de :

- **l'Aiguilleur** qui traite la fonction de routage des messages et la gestion des entrées/sorties de la cellule.
- **l'Unité de traitement** qui assure le calcul de la fonction associée à la cellule.
- Un ensemble de tampons : **Aiguilleur -> Unité de traitement**, qui mémorisent les messages destinés à l'Unité de traitement et désynchronisent les échanges "Aiguilleur -> Unité de traitement".

- Un ensemble de tampons: **Unité de traitement -> Aiguilleur**, qui mémorisent les messages résultats de l'Unité de traitement et désynchronisent les échanges "Unité de traitement -> Aiguilleur".

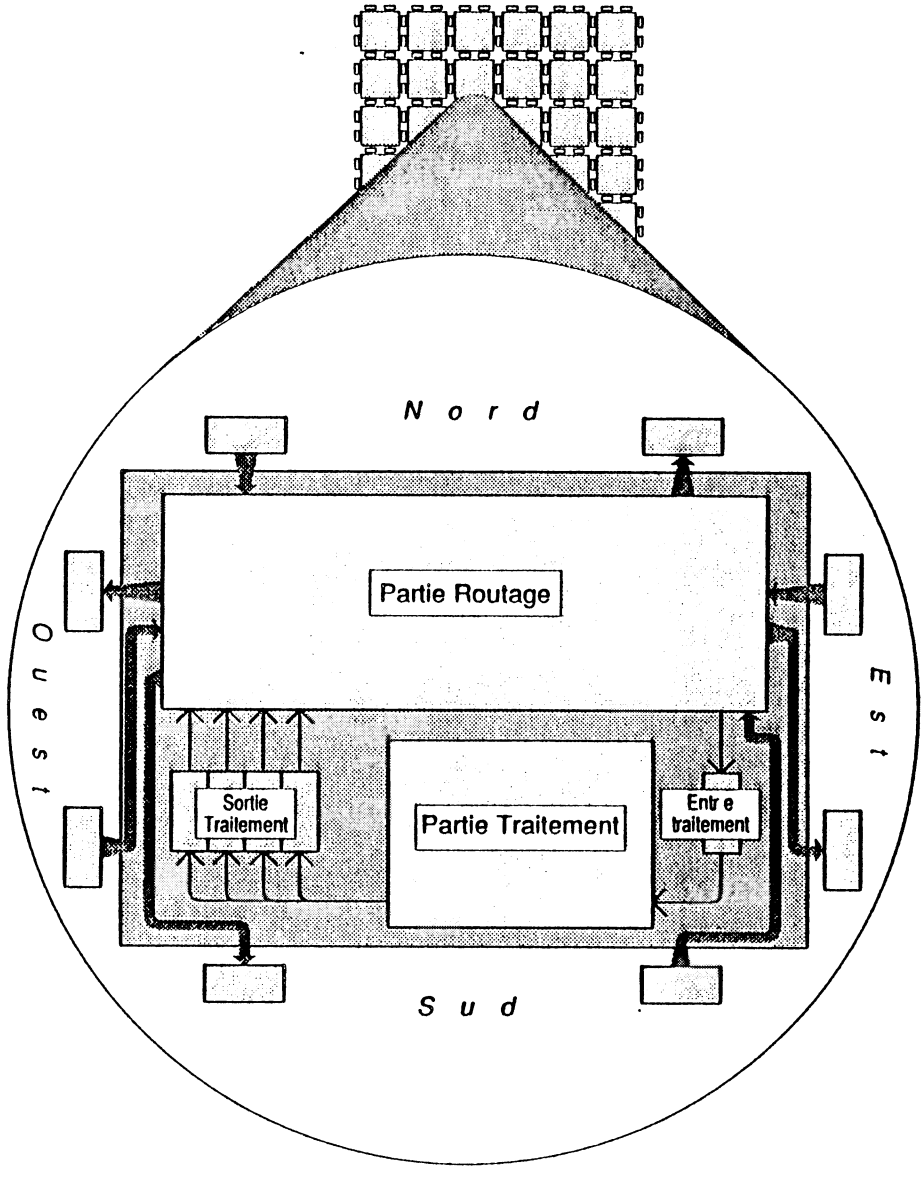


Figure 2.9. Schéma fonctionnel d'une cellule

2.6. AIGUILLEUR

2.6.1 Description

Le rôle de l'aiguilleur (fig. 2.11) est d'assurer la transmission des messages contenus dans les tampons d'entrée de l'aiguilleur vers les tampons de sortie de l'aiguilleur => **l'aiguillage des messages**.

Les tampons d'entrée de l'aiguilleur sont les tampons d'entrée périphériques de la cellule (**Nord, Ouest, Sud, Est**) et les tampons résultats de la partie traitement. Tous ces tampons contiennent un message dont le champ déplacement (**dx, dy**) indique le chemin restant à parcourir pour arriver à destination.

Les tampons de sortie de l'aiguilleur sont les tampons de sortie périphériques de la cellule (**Nord, Ouest, Sud, Est**) et les tampons d'entrée de la partie traitement. Les messages vers l'unité de traitement ne contiennent pas de champ déplacement (**dx, dy**), car ces messages sont arrivés à destination.

Le fonctionnement de l'aiguilleur est alors le suivant :

Dès qu'un message est présent sur l'un des tampons d'entrée périphérique (Nord, Ouest, Sud ,Est) ou résultat, il le lit et recherche un tampon de sortie où le réécrire dans :

- un des tampons d'entrée de l'unité de traitement si le message lui est destiné

- un des tampons de sortie périphérique si le message est destiné à une autre cellule.

Si le tampon destinataire n'est pas libre, il laissera momentanément le message en attente dans le tampon d'entrée dont l'état reste **PLEIN**. C'est seulement lorsque le tampon de sortie est disponible que le message y est transféré et que le tampon d'entrée est vidé.

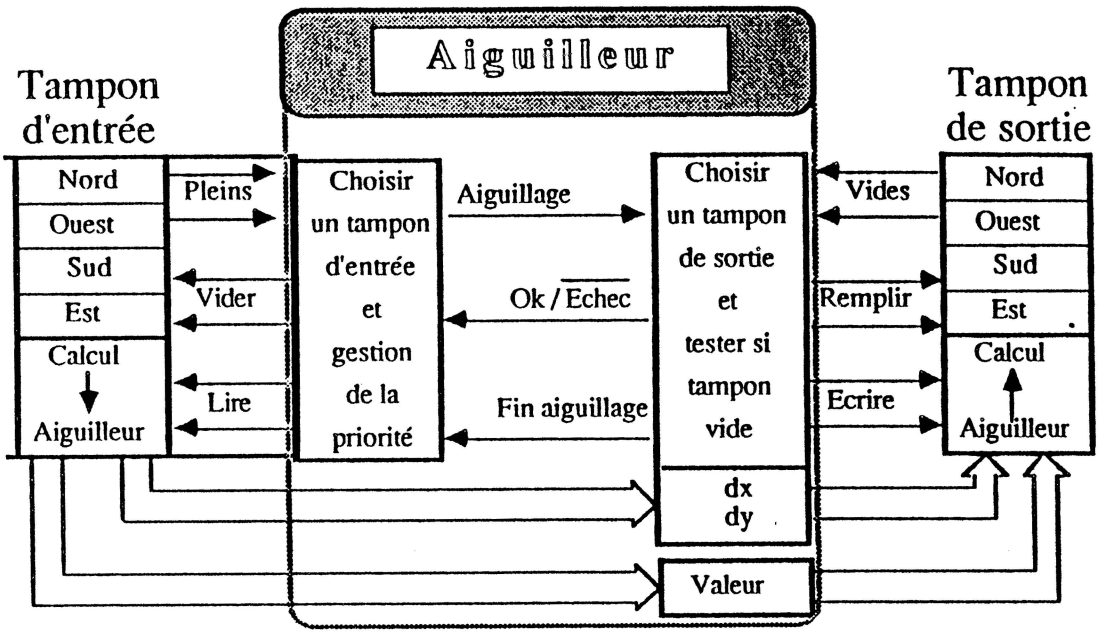


Figure 2.10. Schéma fonctionnel de l'aiguilleur

2.6.2. Politique d'acheminement

Deux cas peuvent se produire lors du choix d'un tampon de sortie : soit le message est arrivé à destination si le champ déplacement est nul (**dx=dy=0**) auquel cas le tampon de sortie sera

l'un des tampons d'entrée de la partie traitement, soit le message n'est pas encore arrivé à destination auquel cas le tampon de sortie devra être choisi parmi les quatre tampons de sortie périphériques: **"Nord, Ouest, Sud, Est"**. Dans ce dernier cas, plusieurs algorithmes ont été étudiés et testés sur des exemples..

La première classe d'algorithmes étudiés consistait à diminuer systématiquement la distance de Manhattan du message (dx, dy), de façon à rapprocher le message de sa destination. Ces algorithmes ont l'avantage d'être très simples et donc d'être facilement intégrables au niveau circuit. Nous avons retenu l'un de ces algorithmes pour la réalisation d'un circuit traitant un réseau de simulation logique.

La deuxième classe d'algorithmes plus sophistiqués permet d'envisager un routage global auto-adaptatif en cas de voie occupée et d'éviter les cellules défectueuses.

Ces algorithmes impliquent un dispositif de transmission plus complexe et pour les plus sophistiqués, une augmentation du nombre d'informations à l'intérieur d'un message. Ces algorithmes ont donné lieu à une étude pour l'intégration de gros réseaux sur **WSI** (**W**afer **S**cale **I**ntegration) [**FAU**86]. Ils sont en effet bien adaptés pour résoudre le problème de non fonctionnement de certaines cellules dans cette technologie, en permettant au message de contourner les zones ou cellules défectueuses.

2.6.3. Algorithme de l'aiguilleur

Nous avons choisi dans un premier temps, d'implémenter l'aiguilleur par une scrutation des entrées. L'algorithme, qui est donc séquentiel, est le suivant :

Répéter
  Si Reset **Alors**
    "**Vider**" les tampons d'entrée et initialiser la priorité R1
  Sinon
    Choisir un tampon d'entrée plein en fonction de la priorité R2
    Lire le message du tampon sélectionné (dx, dy, valeur)
    Choisir un tampon de sortie en fonction du déplacement (dx,dy)
    Si tampon de sortie "**VIDE**" **Alors**
      Mettre à jour le champ déplacement (dx,dy) du message R3
      "**REMPLIR**" le tampon de sortie avec le nouveau message
      "**VIDER**" le tampon d'entrée
    Sinon
      Echec => modification de la priorité R4
    Fin si
  Fin si
tant que Vrai

Remarques (R1, R2, R3, R4) sur l'algorithme de l'aiguilleur:

R1: L'initialisation du réseau qui est activée par un signal RESET, consiste à détruire tous les messages qui pourraient être en circulation. Cette initialisation est assurée par les aiguilleurs qui ont la tâche de "**Vider**" tous leurs tampons d'entrée.

R2: Une priorité permet à l'aiguilleur de lever les conflits lors de la sélection d'un tampon d'entrée lorsque plusieurs tampons

d'entrée sont pleins. Le choix d'un tampon d'entrée se fait ainsi de façon entièrement déterministe. La gestion de cette priorité, dépend directement de la nature du problème traité par le réseau. Plusieurs types de priorité ont été étudiés dans le cadre de la simulation logique.

ℝ3: La fonction de mise à jour du champ déplacement est la suivante:

```
Cas Direction de sortie au
   Nord  : dy := dy+1;
   Ouest : dx := dx+1;
   Sud   : dy := dy-1;
   Est   : dx := dx-1;
   Autre :
Fin cas;
```

ℝ4: En cas d'échec, et pour ne pas bloquer inutilement d'autres tampons d'entrée, l'aiguilleur doit obligatoirement choisir un tampon d'entrée "**PLEIN**" (s'il en existe) qui soit différent de celui avec lequel il vient d'essuyer un échec ; pour cela il modifie la priorité qu'il avait au pas précédent et recommence.

2.6.4. Efficacité de l'aiguilleur

L'algorithme exécuté par l'aiguilleur pour assurer l'aiguillage des messages est un algorithme séquentiel avec priorité.

Un seul message peut traverser l'aiguilleur durant un pas d'algorithme. Plusieurs pas d'algorithme peuvent alors être nécessaires à l'aiguilleur pour vider l'ensemble des tampons d'entrée. La vitesse du flot de messages entre les cellules dépend

donc directement du temps de traitement d'un pas d'algorithme par l'aiguilleur.

Pour des raisons d'efficacité, cet algorithme a été câblé dans l'aiguilleur, la partie séquencement étant traitée par un **PLA**. Ainsi, la version du réseau développée dans le cadre de la simulation logique, dans une technologie CMOS 2µm 1 métal, permet à l'aiguilleur d'exécuter un pas d'algorithme ou un aiguillage en moins de **300 ns**, ce qui autorise un débit théorique à l'intérieur de l'aiguilleur de **3 MégaMessages/seconde**.

2.7. PARTIE TRAITEMENT

2.7.1. Rôle

La partie traitement est le cœur de la cellule. Elle dépend de l'application que l'on veut implémenter sur le réseau.

Dans le réseau, les N x M cellules sont identiques. Mais elles peuvent comporter un petit nombre de paramètres qui sont chargés lors d'une phase d'initialisation. Ces paramètres déterminent le fonctionnement ultérieur de la cellule.

2.7.2. Fonctionnement

Les cellules étant paramétrées, le fonctionnement du réseau doit comporter deux phases :

- une phase d'**initialisation**,
- une phase de **fonctionnement**.

Durant la phase d'initialisation qui est activée par un signal RESET, les valeurs à transmettre aux différentes cellules sont injectées dans le réseau par les bords. Elles sont, ensuite, acheminées par le mécanisme de routage lui même. Une fois toutes les valeurs parvenues aux bonnes cellules, la deuxième phase débute.

La plupart des applications nécessitent un mécanisme pour synchroniser l'algorithme exécuté par chaque cellule. On peut résumer cet algorithme de la manière suivante :

Tant que vrai **faire**
  envoyer m messages
  recevoir n messages
  calculer
fin faire

L'exemple suivant montre les problèmes qui se posent s'il n'y a pas de synchronisation.

Soit une cellule A qui attend un message de B et un message de C. Imaginons la cellule B très rapides et la cellule C très lente, il se peut que B ait pu émettre deux messages vers A, avant que C n'en ait émis un. Dans ce cas A avalera les deux messages de B comme étant le message de B et celui de C d'où dysfonctionnement.

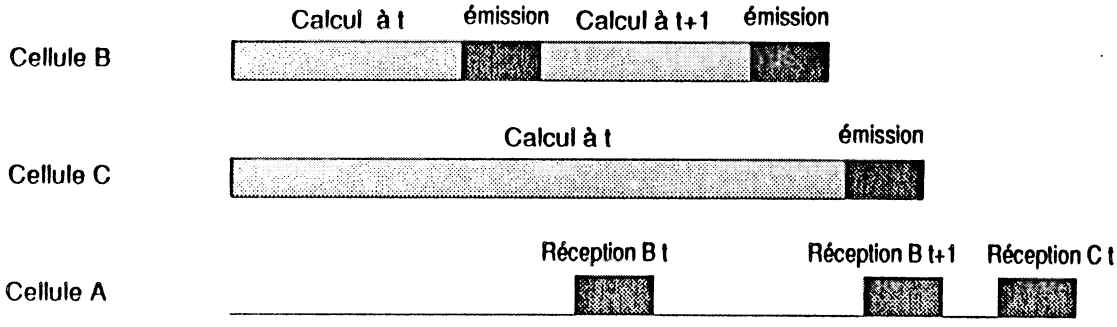
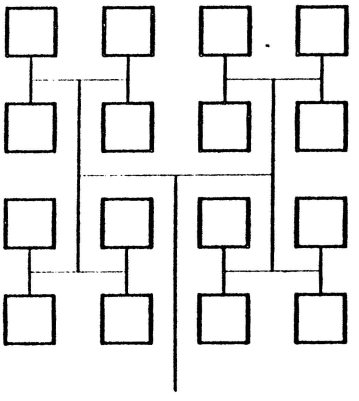


Figure 2.11. Dysfonctionnement possible

Plusieurs techniques sont envisageables pour réaliser une telle synchronisation.

La première nécessite un signal allant du réseau vers l'extérieur, ce signal appelé END est un ET du signal END qu'émet chaque cellule. A la réception du END un signal GO est émis de l'extérieur du réseau vers chaque cellule du réseau. Ces deux signaux doivent être câblés. De la même manière que le RESET, ils n'utilisent pas le mécanisme de routage.



GO, END, RESET, H,
VDD, VSS

Figure 2.12. Les signaux, GO, END et RESET

Une autre technique consiste en l'envoi d'un accusé de réception par la cellule qui reçoit un message. Si la cellule émettrice attend l'accusé de réception, cela réalise un asservissement entre toutes les cellules qui interdit les comportements aberrants. Suivant la façon dont les envoi/attente d'accusé de réception et les envoi/attente de message sont placés dans la cellule, on peut tolérer une plus ou moins grande désynchronisation des cellules et implanter des fonctionnements "demand driven" ou "data driven"

Les deux techniques comportent leurs avantages et leurs inconvénients. L'une, la première, ne nécessite pas d'information supplémentaire dans la cellule, mais oblige à réaliser le câblage de deux signaux supplémentaires. L'autre augmente la taille d'une cellule parce qu'il est nécessaire d'y stocker non seulement les adresses des cellules aval mais aussi celles des cellules amont.

2.7.3. Exemples d'applications étudiées

Cette architecture de réseau de cellules asynchrones peut être employée pour réaliser différents opérateurs spécialisés. Plusieurs applications sont ainsi étudiées :

- l'application qui fait l'objet du circuit réalisé durant ces dernières années est un simulateur logique [**MAZ87**]. Elle sera présentée dans le chapitre suivant.

- un placeur (optimisation par échange de paires) a été développé et sera présenté au chapitre 4.

- La reconstruction d'images [**LAT88a**] fait l'objet d'une étude en cours, pour laquelle une architecture est actuellement définie.

- Il semble que notre réseau soit bien adapté à des modèles neuronaux [**FAU88a**]. Pour cela, des simulations fonctionnelles en OCCAM de réseaux multicouches avec apprentissage par rétropropagation des gradients sont en cours.

- Une étude est en cours [**PAY88**] pour étudier la faisabilité sous cette forme d'une machine Data-flow qui exécuterait directement des programmes décrits dans le langage LUSTRE.

2.8. LA MACHINE CELLULAIRE

2.8.1. Le principe

Dans le paragraphe précédent, nous avons vu que des signaux (GO, END, RESET) et des messages pénètrent et sortent du réseau. C'est un ordinateur hôte qui pilotera ce dernier.

Le réseau est réalisé de tel manière qu'il est possible d'en cascader plusieurs entre eux.

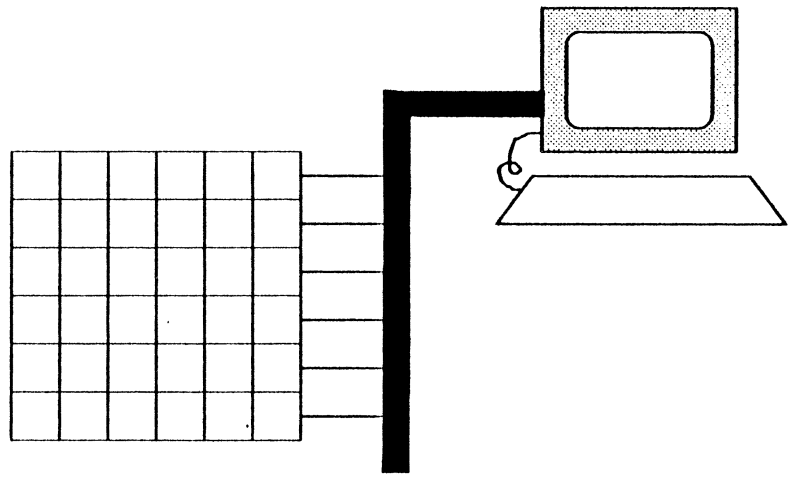


Figure 2.13. Schéma de Principe de la machine cellulaire

Une manière simple d'interfacer le réseau avec l'ordinateur hôte serait de donner une adresse pour chaque tampon périphérique et de transmettre les données en parallèle. Mais un problème de taille se pose : un message est composé de n bits, n étant supérieur à 10 dans la plupart des cas. Il est donc impératif d'utiliser un minimum de plots par tampon périphérique. Nous avons défini une cellule d'interface parallèle/série dans un sens et série/parallèle dans l'autre qui n'en utilise qu'un seul.

2.8.2. Les interfaces

Les informations qui transitent entre deux cellules voisines :

- le **message**,
- le signal **remplir** tampon,
- le signal **vider** tampon.

Ceci étant vrai pour les deux sens de communication.

Nous nous apercevons que, par sens, deux messages vont de la cellule émettrice vers le tampon (message et remplir) et un de la cellule réceptrice vers le tampon (vider). La solution choisie pour le protocole des interfaces est une solution synchrone avec un seul fil par sens. Ce fil servira à la fois pour émettre les signaux message et remplir du sens 1 et le signal vider du sens 2 (voir figure).

L'émetteur envoie un 0 en permanence sur le fil. Les deux types de messages qui peuvent transiter par ce fil seront différenciés par une séquence de code :

- 0 1 1 pour le signal VIDER,
- 0 1 0 pour annoncer le message, les n bits qui le composent sont ensuite émis.

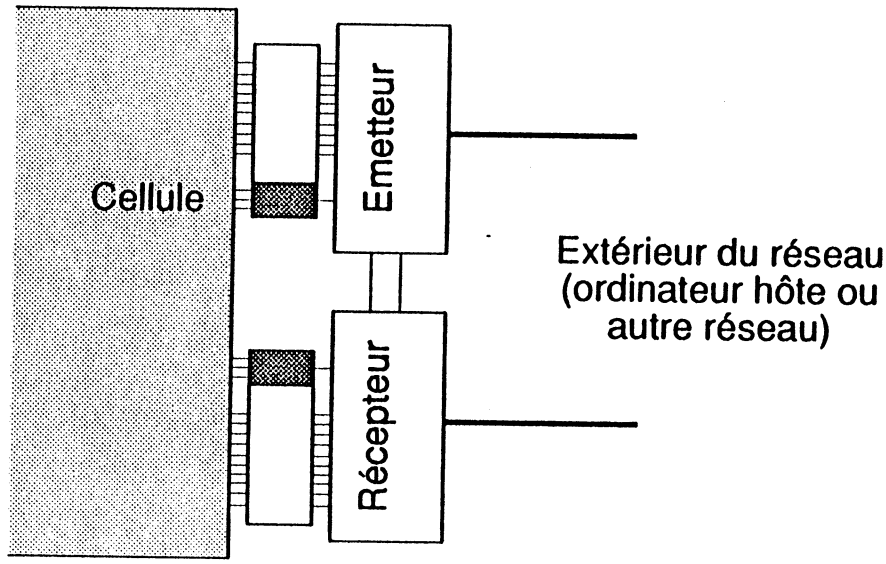


Figure 2.14. Schéma de principe des interfaces

2.8.3. La machine

La machine de simulation est composée de deux parties : un ordinateur hôte et un réseau NxM de cellules. L'ordinateur hôte a pour rôle d'enchaîner 3 opérations successives :

- préparer l'affectation d'une fonction à réaliser à chaque cellule du réseau (dans notre cas, associer chaque porte du réseau à simuler, à une cellule de notre réseau de simulation).

- initialiser le réseau en envoyant des messages à chaque cellule,

- piloter le fonctionnement du réseau en lui fournissant les entrées et en recevant les résultats.

Actuellement, nous envisageons de faire jouer le rôle de l'hôte à un PC/AT, et de réaliser la matrice de cellule sur une plaque compatible. Cela nécessite la réalisation sur la plaque d'une interface matérielle qui récupère les mots sur le bus et les envoie dans le réseau et une intelligence logicielle qui prépare les messages à envoyer aux cellules pour les faire passer, bout par bout en parallèle, sur les points d'interface des circuits. Tout cela n'a pu encore être développé.

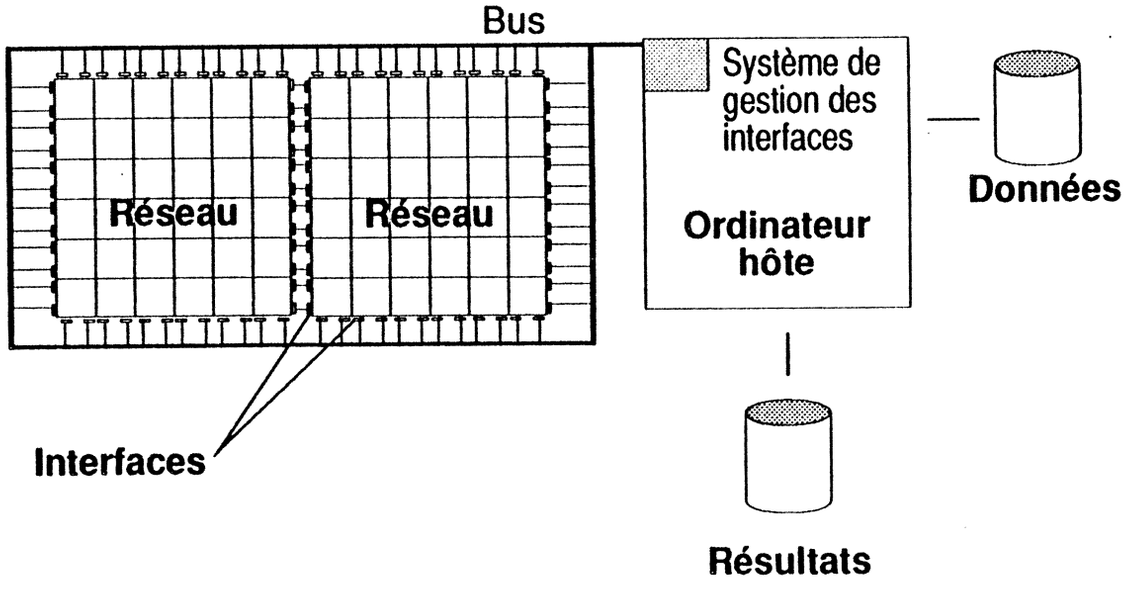


Figure 2.15. La machine cellulaire

Chapitre 3

Réalisation d'un circuit intégrant plusieurs cellules

REALISATION D'UN CIRCUIT INTEGRANT PLUSIEURS CELLULES

3.1. DESCRIPTION FONCTIONNELLE

3.1.1. L'algorithme de simulation

Comme nous l'avons vu précédemment, le but de ce circuit est de valider l'architecture. Nous avons donc choisi d'y implémenter un algorithme simple de simulation. L'algorithme séquentiel duquel nous sommes partis, nécessite :

- une variable *h* pour mémoriser le temps de simulation,
- une paire de variable : *ancienne_valeur* et *nouvelle_valeur* pour mémoriser les valeurs de chaque connexion ou entité transportant une valeur logique,
- un point d'entrée pour donner les chronogrammes d'entrée.

L'algorithme est le suivant :

```
ancienne_valeur := inconnu
Tant que h < temps_maximum_de_simulation faire
  h := h + 1
  Pour chaque porte faire
    évaluer les sorties en fonction des valeurs d'entrée
    nouvelle_valeur [sortie] := valeurs évaluées
  Fin pour
Fin tant que
```

Cet algorithme a une structure très parallèle : toutes les portes peuvent être évaluées en même temps, dès que les valeurs de leurs entrées sont connues. Nous affectons donc, une porte du réseau à

simuler à une cellule de notre réseau. Celle-ci après avoir évaluée sa valeur, l'envoie aux cellules qui sont connectées en aval.

La simulation complète d'un circuit par un réseau cellulaire nécessite des opérations successives :

- **assignation des portes aux cellules** : le schéma du circuit à simuler est compilé, un placeur assigne une cellule à une porte de façon à minimiser les connexions.

- **initialisation du réseau** : à la réception d'un signal RESET, le réseau entre dans une phase d'initialisation pendant laquelle l'ordinateur hôte envoie un nombre déterminé de messages d'initialisation aux différentes cellules.

- **Simulation** : à la fin de la phase d'initialisation, le réseau part dans une séquence de fonctionnement. Chaque cellule exécute l'algorithme de simulation suivant :

  Tant que vrai **faire**
    Pour chaque entrée **faire**
      recevoir un message
      l'enregistrer dans un registre interne
    Fin pour
    évaluer la nouvelle sortie à partir des entrées reçues
    envoyer la valeur de sortie aux cellules aval (et
      éventuellement à l'ordinateur hôte)
  Fin tant que

- **Edition des résultats** : Les cellules devant communiquer des résultats, les envoient à l'ordinateur hôte par l'intermédiaire du mécanisme de routage. L'hôte les met en forme, pour les afficher.

3.1.2. Problèmes de synchronisation

L'algorithme parallèle de simulation, précédemment décrit est trop simple et faux. Les cellules étant asynchrones et les messages ayant des longueurs variables à parcourir, des cellules peuvent recevoir leurs entrées et envoyer leurs résultats plus rapidement que d'autre. Ceci pose plusieurs problèmes.

Premièrement, il est nécessaire, sous peine de recevoir une entrée du temps t+1 avant toutes les entrées du temps t et de ne pas savoir les différencier, que toutes les cellules travaillent à la même vitesse. Une possibilité pour pallier à cela, serait de laisser attendre certains messages d'entrée dans le réseau. Mais dans ce cas là, un phénomène de "bouchon" pouvant dans certains cas aller jusqu'au deadlock, pourrait se produire : si un message est retenu aucun autre ne peut le dépasser.

Un deuxième problème est de différencier les messages en fonction de leur port d'entrée. Il se peut très bien qu'arrivent, dans une cellule, le message destiné à l'entrée E1 à l'instant t, puis à l'entrée E1 à l'instant t+1, puis à l'entrée E2 à l'instant t !

Enfin, l'ordinateur hôte doit construire des chronogrammes de sortie, il doit donc connaître le temps de calcul des valeurs qui lui sont communiquées par le réseau.

3.1.3. Les solutions pour re-synchroniser

Une solution est de synchroniser chaque exécution de la boucle principale de l'algorithme décrit, de façon à éviter tout recouvrement

d'exécution par les cellules. Ceci peut être réalisé par les signaux GO et END présentés au chapitre 2 :

```
Tant que vrai faire
  attendre le signal GO
  envoyer les sorties précédemment évaluées
  pour chaque entrée faire
    recevoir un message
    calculer le résultat partiel (opérations associatives)
  fin pour
  mémoriser le résultat final
  envoyer le signal END
fin tant que
```

L'ordinateur hôte contrôle l'exécution de la manière suivante :

```
h := temps initial
tant que h < temps_maximum_de_simulation faire
  signaler GO
  envoyer les valeurs d'entrée du temps h
  accepter un message pour chaque connexion à sortir,
    mais ne pas attendre
  attendre les signaux END
  h := h + 1
fin tant que
```

De cette manière, les cellules ont un fonctionnement asynchrone tout en étant re-synchronisées à chaque pas de simulation. Les messages sont envoyés en début de boucle de façon à ce qu'aucun message du temps t+1 ne soit dans le réseau en même temps que des messages du temps t. Les messages transitant dans le réseau, n'y sont pas laissés en attente, ainsi aucun deadlock n'est possible.

D'autres moyens de re-synchroniser et d'autres types d'algorithmes sont réalisables, ils ont été présentés dans [COR86], [OBJ87b].

3.2. LES CHOIX FONDAMENTAUX

3.2.1. La technologie

Deux possibilités nous étaient offertes pour réaliser notre circuit : le CMP et le CNET de Meylan. Pour des raisons d'outils de CAO (le CNET a mis à notre disposition un langage d'opérateurs flexibles LOF, ainsi que ses outils de vérifications), la technologie du CNET a été choisie, ceci à l'automne 1986. Ce choix n'est pas figé pour les futures réalisations. La technologie du CNET utilisée est une techno 2 µm à un seul niveau de métal.

3.2.2. La méthode de conception

Deux versions du circuit ont été réalisées. Le principe de conception de celles-ci est d'utiliser une bibliothèque de cellules de base. Ce choix implique une surface de silicium plus grande qu'un même circuit réalisé "à la main". Mais il autorise la paramétrisation de certaines parties du circuit. De plus, le temps de réalisation en est grandement réduit.

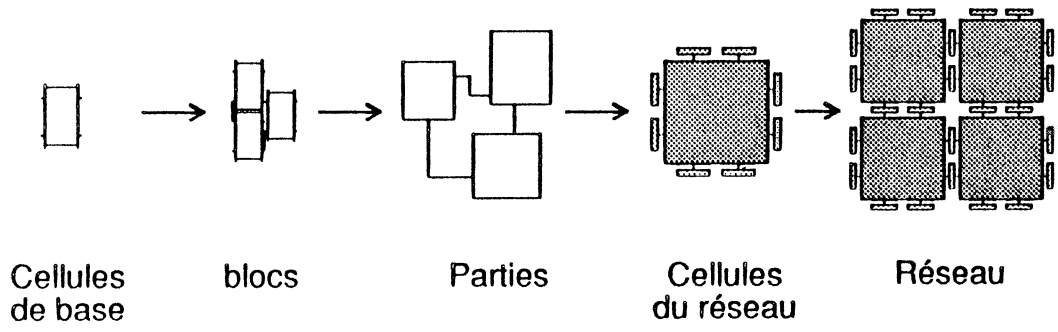


Figure 3.1. La méthode de conception

L'assemblage de cellules peut être fait manuellement, à l'aide d'éditeurs graphiques sophistiqués. Mais il peut aussi être fait par programme, la conception de "blocs" ou "parties" pouvant être rendus flexible, paramétrable. Le travail consiste alors à écrire des "générateurs" de blocs qui pourront générer toutes les configurations désirées.

Dans le cadre de notre projet, nous souhaitions pouvoir réutiliser bon nombre des parties conçues pour ce circuit, pour les employer dans d'autres applications (partie routage, par exemple). Nous avons donc choisi d'écrire des générateurs. Il était donc intéressant d'expérimenter LOF (Langage d'Opérateurs Flexibles) développé au CNET. LOF est un langage qui permet l'assemblage de plusieurs cellules entre elles, ainsi que le routage entre différents blocs. Par programme, il est possible de réaliser de grandes structures régulières (RAM,...) et ceci de manière paramétrée.

La première version ne consistait qu'en une seule cellule. Il s'est avéré que notre application ne correspondait pas à l'utilisation optimum de LOF. Comme nous allons le voir dans les paragraphes

suivants, une grande partie de la cellule ne présente pas un caractère assez régulier pour une bonne utilisation de LOF.

De cette première version, nous avons conservé, pour réaliser la deuxième (le circuit 2x2), le choix fondamental qui est l'utilisation de cellules de bases. Ces cellules de base sont : des nand, des nor, des multiplexeurs, des registres, des inverseurs, des amplis, des portes trois états.

Nous les avons toutes redessinées de façon à les rendre plus dense. Les caractéristiques de ces cellules sont les suivantes :

- largeur : 75 µm
- hauteur : multiple de 7,5 µm
- alimentations courant verticalement (VDD à gauche, VSS à droite)
- Entrées et sorties disposées sur un pas de 7,5 µm
- Entrées et sorties disponibles à droite et à gauche de la cellule et disposées sur la même horizontale pour une même entrée ou une même sortie.

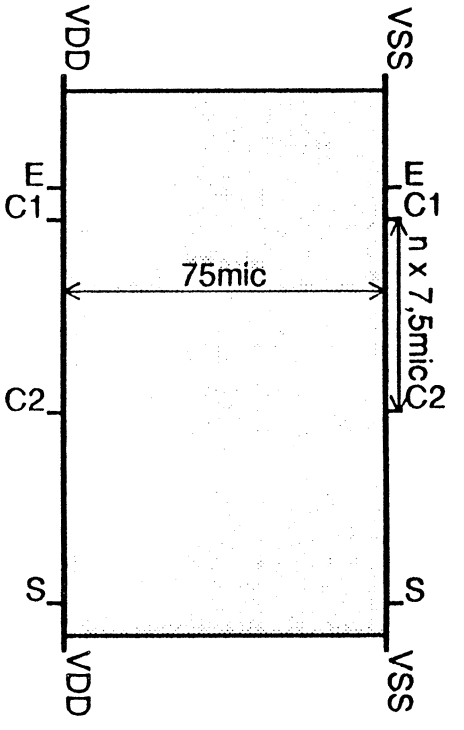


Figure 3.2. Schéma de principe d'une cellule de base

Ces cellules de bases ont été assemblées de façon à réaliser les grandes fonctionnalités de chaque partie d'une cellule du circuit. Une fois chacune des deux parties réalisée, elles ont été assemblées ensembles.

3.2.3. Les outils

La conception du circuit a été réalisée en grande partie grâce aux moyens techniques de l'Unité de Service Systèmes Intégrés (USSI) de l'INPG et grâce aussi à l'aide des moyens du CNET (pour la vérification notamment).

Le logiciel utilisé est Silvar-Lisco sur Vax station GPX, 5 Méga de RAM, comportant un disque dur de 280 Méga.

Silvar-Lisco, logiciel dont nous avons activement participé à l'implantation sur le site de Grenoble, est un logiciel de CAO ouvert comportant divers outils :

- simulateur fonctionnel
- éditeur logique, électrique (CASS)
- interface vers simulateurs logiques, électriques
- éditeur micron (PRINCESS)
- outils de vérification (DRC, ERC, extracteurs)
- générateur de masques

Le schéma suivant montre pour chaque étape de la conception le ou les outils utilisés [COR87c].

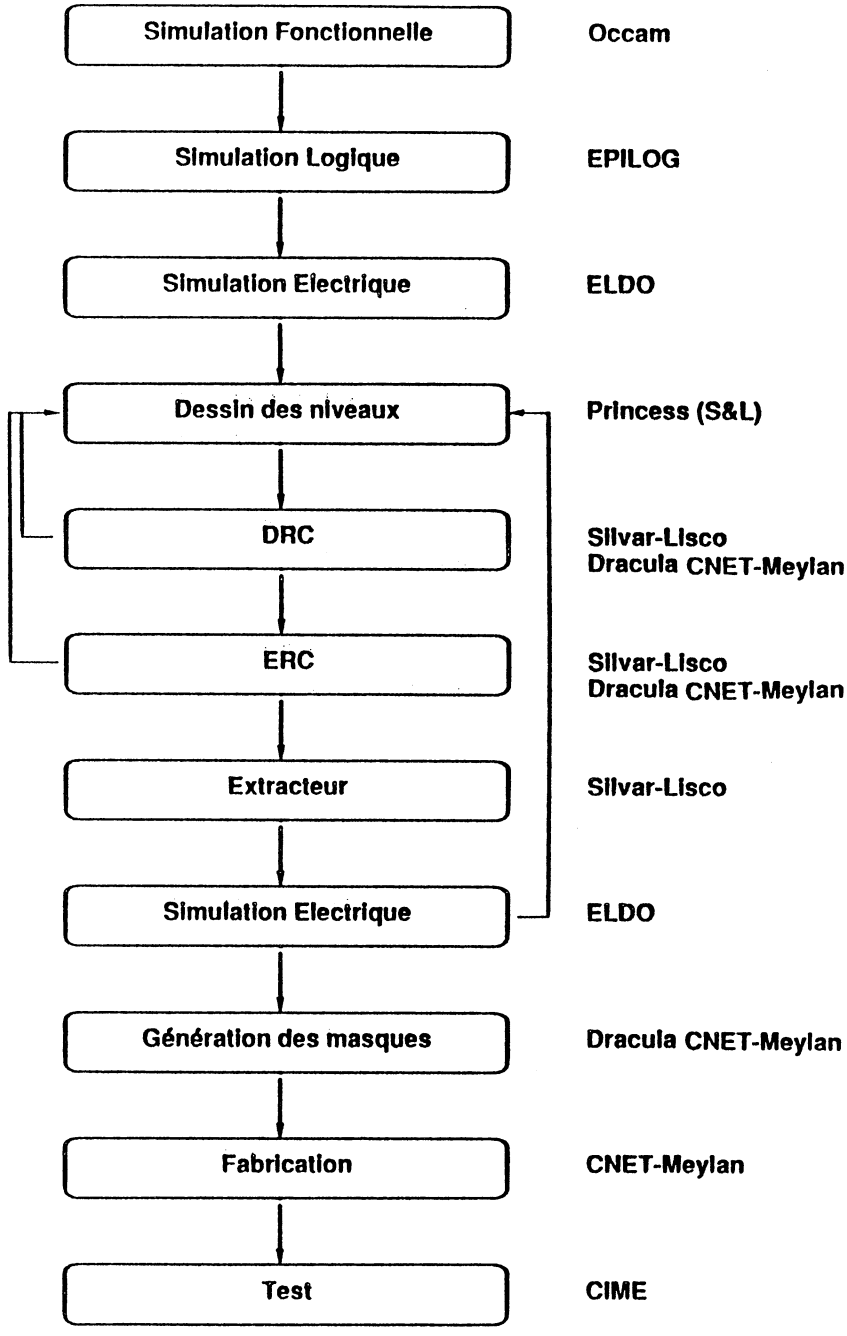


Figure 3.3. La conception, les outils

3.3. SIMULATION FONCTIONNELLE

3.3.1. Utilisation du langage OCCAM

La première étape de la conception de ce circuit a été la validation de l'architecture. Cette validation a été obtenue par une simulation fonctionnelle d'un réseau nxn avec n variant de 4 à 8.

Les recherches en cours, au LGI sur le langage OCCAM et le Transputer ainsi que la facilité avec laquelle le réseau peut être décrit dans ce langage, nous ont, naturellement, fait choisir OCCAM pour cette simulation fonctionnelle [ANS86], [COR87a].

3.3.2. La simulation du réseau

L'architecture du réseau est décrite en OCCAM, par des blocs fonctionnels communiquant. A chacun d'entre eux (cellule, tampon, aiguilleur, partie traitement,...) est associé un processus OCCAM. Les communications entre blocs sont réalisées par les canaux OCCAM [OBJ87a].

La description de la simulation est présentée dans [ANS88].

3.4. LA CELLULE

3.4.1. Deux mots sur la partie routage

Le rôle de la partie routage est d'acheminer les messages entre les tampons d'entrée et les tampons de sortie. Le choix de l'entrée est déterminé par la priorité de chacune des demandes. Il s'agit d'une priorité tournante. Le choix de la sortie est fonction de la destination finale du message (valeurs dx et dy du champ routage), l'algorithme d'acheminement implémenté dans l'aiguilleur privilégie le routage horizontal des messages.

Logiquement, l'architecture de l'aiguilleur s'organise autour de 2 blocs : l'encodeur de priorité spécialisé dans le choix du message d'entrée à transmettre et la partie traitement aiguilleur (PTA) chargée de déterminer le tampon de sortie vers lequel le message doit être acheminé. Le contrôle du transfert est séquencé par un PLA.

L'interface entre l'encodeur de priorité et la partie traitement aiguilleur est un registre interne.

La réalisation de cette partie routage est présentée dans [OBJ88].

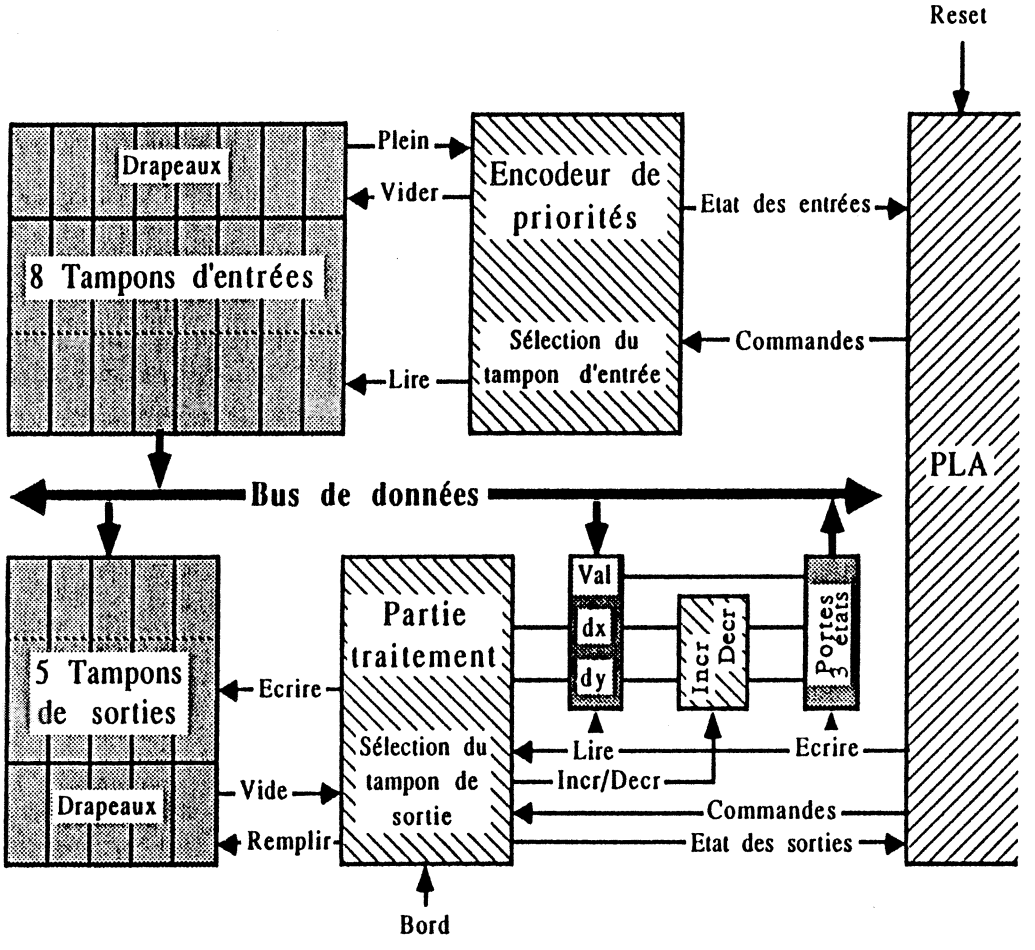


Figure 3.4. L'aiguilleur

3.4.2. Plan d'ensemble de la partie traitement

3.4.2.1. La simulation logique implémentée

La partie traitement, ici réalisée, a pour but de simuler le fonctionnement logique d'une porte. La synchronisation du déroulement du processus de simulation peut être effectué de deux manières : par l'utilisation d'accusés de réception ou par la réalisation de signaux globaux GO/END [OBJ88]. Nous avons choisi cette dernière méthode, la simulation suit un algorithme très simple :

Tant que Vrai
  Attendre le signal GO
  Emettre aux cellules aval, les valeurs précédemment évaluées
  Pour chaque entrée **faire**
    recevoir un message
    traiter le message avec le traitement de ceux déjà reçus
    stocker le résultat
  Fin Pour
  Signaler END
Fin tant que

Nous avons entrepris de réaliser un simulateur logique très simple : l'objectif était davantage de valider l'architecture que de réaliser un outil de simulation très puissant et riche en fonctionalités.

Notre simulateur accepte 4 fonctions : ET, OU, OUX (OU exclusif), COLLAGE à 0. De plus il est possible d'inverser le résultat, ce qui nous donne 4 fonctions de plus : NONET, NONOU, NONOUX et COLLAGE à 1. Le NOP peut être réalisé grâce à un OU, un ET ou un OUX à une seule entrée, l'inversion se réalisant quant à elle grâce à un NONET, un NONOU ou un NONOUX.

La simulation est réalisée sur trois états : 0, 1 ou X (état indéterminé). La longueur du champ information de chaque message à transmettre sera donc de 2 bits.

Pour des raisons de nombre de registres et toujours dans l'optique de réaliser un circuit de validation, nous avons limité la sortance de chaque porte à 4. Il est toujours possible d'augmenter cette sortance en ajoutant un NOP à n sorties sur une ou plusieurs des sorties de la porte.

Les messages d'entrée sont reçus les uns après les autres, donc quelque soit le nombre de ceux-ci, il suffit d'un seul registre d'entrée. En conséquence, nous avons pu très facilement, porter à 8 l'entrance de chaque porte. Il a suffit de mémoriser 3 bits au lieu de 2 pour déterminer cette entrance.

L'algorithme de simulation est celui d'une simulation à délai unitaire ; mais il a été possible à peu de frais de permettre des délais pouvant être 0, 1, 2 ou 3 unités de temps [COR87b].

3.4.2.2. Le matériel nécessaire pour cet algorithme

Une cellule doit donc connaître une liste de paramètres avant de pouvoir simuler la porte qui lui est associée. Ceci nécessite donc une entité de mémorisation des paramètres. Ces paramètres sont :

- Le **type** de porte (ET, OU, OUX, COLLAGE à 0) => **2 bits**
- L'**inversion** du résultat => **1 bit**
- Le **délai** (0, 1, 2 ou 3) => **2 bits**
- L'**entrance** (de 1 à 8) => **3 bits**
- La **sortance** (de 1 à 4) => **2 bits**
- l'**adresse** des cellules aval (dx, dy) chacune étant codée sur 2x4 bits, soit 8 bits par cellule, avec la possibilité d'avoir 4 cellules aval soit **32 bits**.

Il faut donc mémoriser une liste de paramètres de **42 bits**.

La deuxième entité à réaliser et l'entité de simulation qui se charge à partir des entrées de produire le résultat. Il est intéressant de noter que les fonctions logiques élémentaires associées à une

cellule (ET, OU, OUX) sont associatives. La simulation se fera donc au fur et à mesure de l'arrivée des messages d'entrée, il suffit pour cela de réaliser un NOP au premier message. Dans cette entité de simulation, les retards sont gérés par un système de registre à décalage.

Une troisième entité, beaucoup plus simple, doit gérer le positionnement des drapeaux des registres de sortie de la partie traitement. Ces registres de sortie, dans lesquels sont mémorisés dx, dy et le message, sont quant à eux éclatés : la partie dx et dy provient de l'entité mémorisation et le message d'un registre commun pour toutes les sorties et situé sur le chemin de données.

Il faut aussi compter le nombre de messages d'entrée parvenu à la cellule pour qu'elle sache si elle a fini son calcul. Pour cela il faut créer une quatrième entité qui sera un compteur ou un décompteur.

Enfin, il faut séquencer, à l'intérieur de la partie traitement, toutes ces entités entre elles ; pour cela, il faut réaliser un séquenceur.

3.4.3. Plan de masse de la cellule

Il nous a paru intéressant de séparer physiquement les deux parties (routage et traitement) qui composent la cellule. Ceci pour une raison essentielle : elles peuvent travailler avec deux horloges différentes. Ces deux parties sont entourées d'un bus qui permet à chaque tampon périphérique de communiquer avec l'aiguilleur.

Ce choix a été réalisé après avoir étudié d'autres possibilités pour réaliser cette connexion. Il s'est avéré que le bus circulaire était

celui qui prenait le moins de place dans la technologie utilisée (un seul métal).

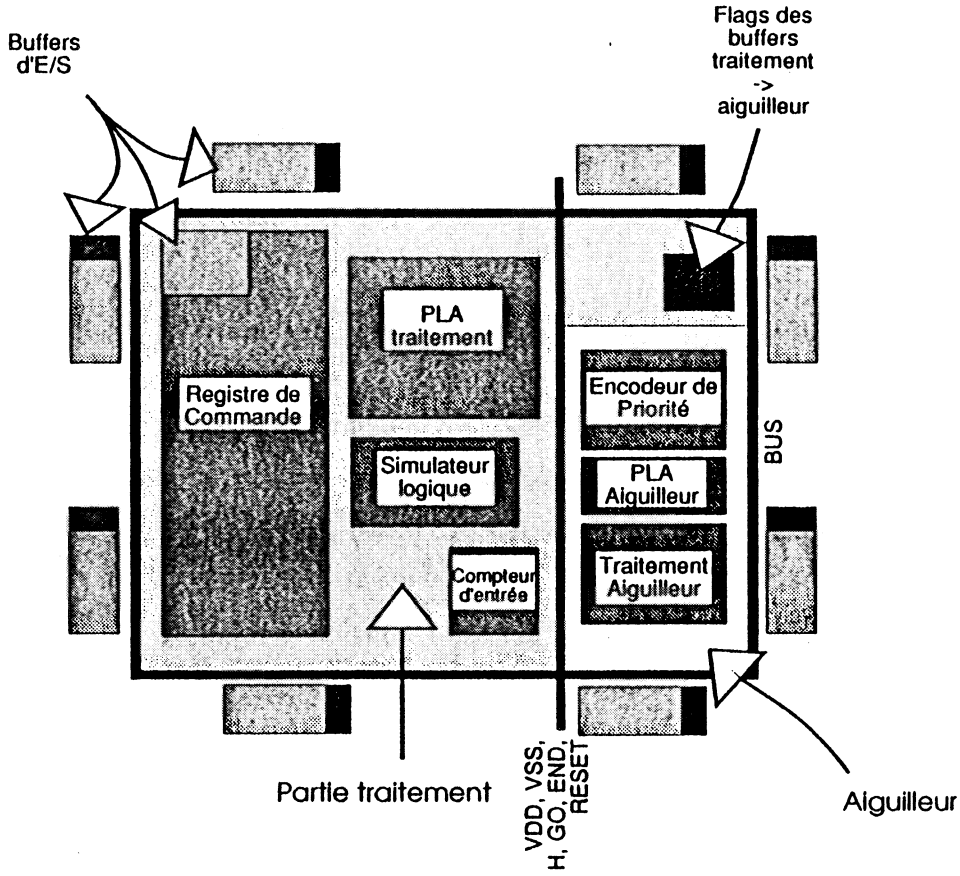


Figure 3.5. Plan de masse

3.4.4. Le stockage des données

3.4.4.1. Etude fonctionnelle

42 bits sont à mémoriser dans chaque cellule. Cette action se déroule pendant la phase d'initialisation. Ils sont reçus 2 par 2 puisque le champ information du message est de 2 bits.

Plusieurs possibilités sont envisageables pour réaliser cette entité de mémorisation. Les contraintes sont les suivantes :

- pas d'information supplémentaire dans le message,
- signal de fin de remplissage, pour indiquer la fin de la séquence d'initialisation.
- construction pouvant être réalisée de manière paramétrable.

Il s'est avéré que le système répondant le mieux à ces critères était le registre à décalage.

Le signal de fin de remplissage peut être généré sans matériel supplémentaire : à la réception du signal RESET toutes les bascules maître/esclave du registre à décalage sont initialisées à 0 sauf la première qui est initialisée à 1. A chaque décalage, le 1 avance, quand il arrive au bout du registre à décalage, ce dernier est plein.

La paramétrisation est très simple : l'ajout d'un point de mémorisation permet d'augmenter la capacité mémoire sans que le message ne soit en rien modifié et qu'il faille ajouter quoique ce soit dans la cellule.

3.4.4.2. Réalisation logique

Du fait de la longueur des messages (2 bits), l'entité de mémorisation ne sera pas composée d'un registre à décalage de 42 bits, mais de deux registres à décalage de 21 bits chacun.

La valeur que produit la cellule doit être initialisée à X au début de la simulation. Pour ce faire, 1 bit sera rajouté dans chacun des

deux registres à décalage. Ces deux bits sont donc le point de mémorisation appelé VAL dans le schéma fonctionnel de la partie traitement.

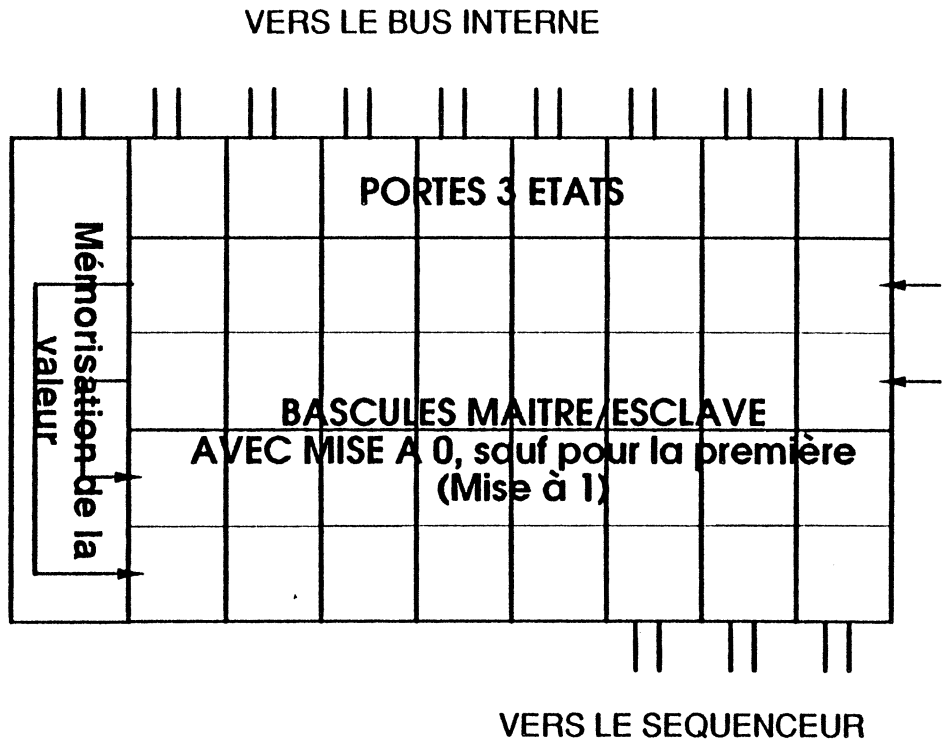


Figure 3.6. Schéma logique de l'entité de stockage des données

3.4.4.3. Implémentation

Dans le registre à décalage, on distingue trois types différents de point de mémorisation :

- ceux communiquant avec le bus (présence de porte 3-états)

- ceux communiquant avec la partie traitement
- ceux dont la valeur doit être modifiée au cour de la simulation (le résultat de simulation qui doit être initialisé, par exemple).

De ces constatations, nous avons développé trois cellules de base dont les schémas sont donnés en annexe.

3.4.5. La réalisation de la fonction logique

3.4.5.1. Etude fonctionnelle

Cette partie a plusieurs fonctions :

- réaliser la fonction logique souhaitée,
- inverser ou non le résultat,
- le stocker dans la file de délai.

La file de délai a pour but de temporiser la sortie des messages de 1, 2 ou 3 pas de simulation, de façon à amener la prise en compte par la simulation, des délais de 1, 2 ou 3 unités de temps.

Les fonctions logiques à réaliser étant toutes associatives, l'évaluation se déroule au fur et à mesure de l'arrivée des entrées. Par exemple l'évaluation d'un OU à 4 entrées se déroulera de la manière suivante :

- 1ère entrée : NOP et mémorisation dans R2
- 2ème entrée : OU de R2 et de l'entrée, mémorisation dans R2

- 3ème entrée : OU de R2 et de l'entrée, mémorisation dans R2
- 4ème entrée : OU de R2 et de l'entrée, mémorisation dans Registre de sortie.

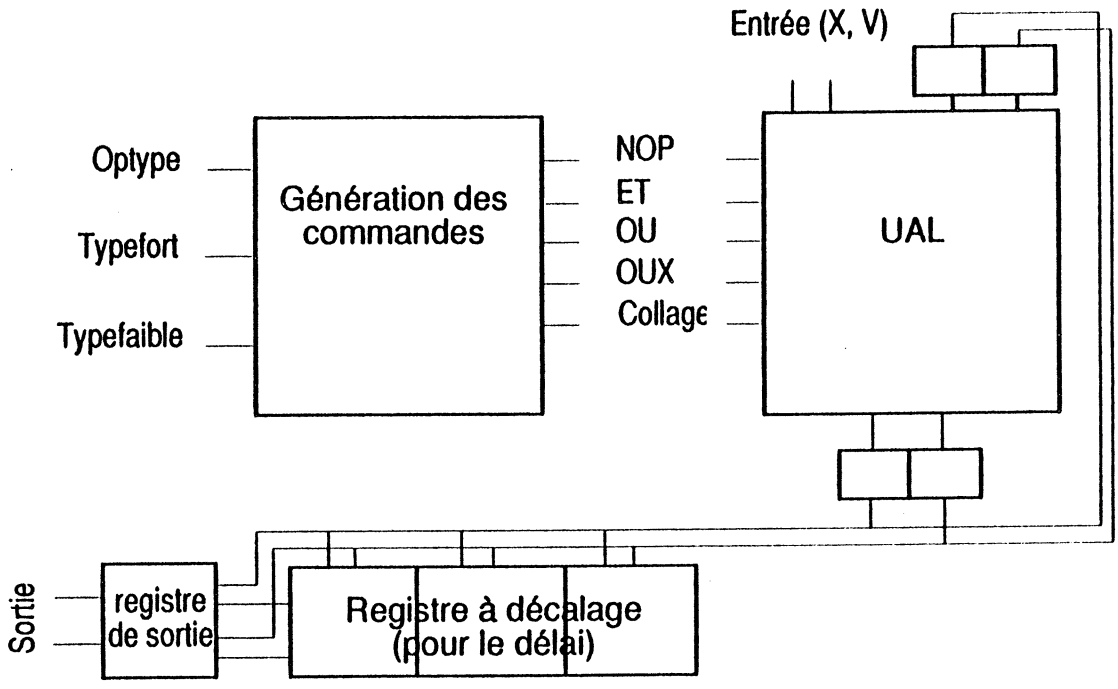


Figure.3.7. Schéma fonctionnel

3.4.5.2. Représentation logique

Nous rappelons le codage choisi pour les messages :

Fort	Faible	Résultat
0	0	0
0	1	1
1	0	X
1	1	X

Les équations logiques sont les suivantes (nous convenons de la notation X_1 pour le bit de poids fort de l'entrée 1 et VAL_1 pour le bit de poids faible de l'entrée 1) :

$$X_{et} = X_1.VAL_2 + X_1.X_2 + X_2.VAL_1$$
$$VAL_{et} = VAL_1.VAL_2$$
$$X_{ou} = X_2.\overline{VAL_1}+X_1.\overline{VAL_2}+X_1.X_2$$
$$VAL_{ou} = VAL_1 + VAL_2$$
$$X_{oux} = X_1 + X_2$$
$$VAL_{oux} = VAL_1.\overline{VAL_2}+\overline{VAL_1}.VAL_2$$
$$X_{nop} = X_1$$
$$VAL_{nop} = VAL_1$$

Pour gérer la réalisation de la fonction logique, des commandes sont nécessaires, celles-ci (ET, OU, OUX, NOP et COLLAGE) sont générées à partir de trois entrées dénommées : OPTYPE, TYPEFORT et TYPEFAIBLE.

OPTYPE est une commande qui vient du séquenceur. Ce signal a pour but d'activer la réalisation de la fonction logique. TYPEFORT et TYPEFAIBLE sont deux valeurs mémorisées au moment de l'initialisation du réseau. Ces deux valeurs correspondent à la fonction logique à réaliser.

Les cinq sorties sont, pour quatre d'entre elles, la fonction logique qui doit être réalisée et COLLAGE est un signal de remise à 0 d'un registre.

$$NOP = \overline{OPTYPE} + OPTYPE.\overline{TYPEFORT}.\overline{TYPEFAIBLE}$$
$$ET = OPTYPE.\overline{TYPEFORT}.TYPEFAIBLE$$
$$OU = OPTYPE.TYPEFORT.\overline{TYPEFAIBLE}$$
$$OUX = OPTYPE.TYPEFORT.TYPEFAIBLE$$
$$COLLAGE = OPTYPE.\overline{TYPEFORT}.\overline{TYPEFAIBLE}$$

Nous pouvons remarquer que la fonction NOP est réalisée par défaut.

L'inversion du résultat consiste en l'inversion du bit de valeur (poids faible) du résultat.

La file de délai est réalisée par un registre à décalage qui peut être chargé au niveau de chaque bascule ; le décalage se produit une seule fois dans le pas de simulation.

3.4.5.3. Implémentation

Pour les autres applications du réseau, il sera difficile de réutiliser une partie de cette entité qui est spécifique à l'application étudiée.

3.4.6. Le chargement des drapeaux de sortie

3.4.6.1. Etude fonctionnelle

Le but de cette entité est de mettre à 1 les drapeaux de sortie le nécessitant. Pour cela, elle reçoit deux entrées : un ordre de fonctionnement venant du séquenceur (CHG) et la sortance du circuit (codée sur deux bits SORTANCEFORT et SORTANCEFAIBLE) qui est mémorisée lors de l'initialisation. Sa réalisation ne pose aucun problème particulier : il s'agit d'un circuit combinatoire classique.

3.4.6.2. Réalisation logique

Le codage pour la sortance est le suivant :

SORTANCEFORT	SORTANCEFAIBLE	SORTANCE
0	0	1
0	1	2
1	0	3
1	1	4

Le boîtier reçoit 3 entrées : les 2 bits de sortance (SORTANCEFORT, SORTANCEFAIBLE) et l'ordre de chargement (CHG). Il génère 4 sorties (S1, S2, S3, S4) : une ligne par drapeau, le passage à 1 d'une ligne signifie le chargement du drapeau correspondant. Si la sortance est de 1 au moment du chargement des drapeaux ; seul le drapeau 1 sera chargé, si la sortance est de 2, les drapeaux 1 et 2 le seront ; etc...

S1 = CHG
S2 = (SORTANCEFORT + SORTANCEFAIBLE).CHG
S3 = SORTANCEFORT.CHG
S4 = SORTANCEFORT.SORTANCEFAIBLE.CHG

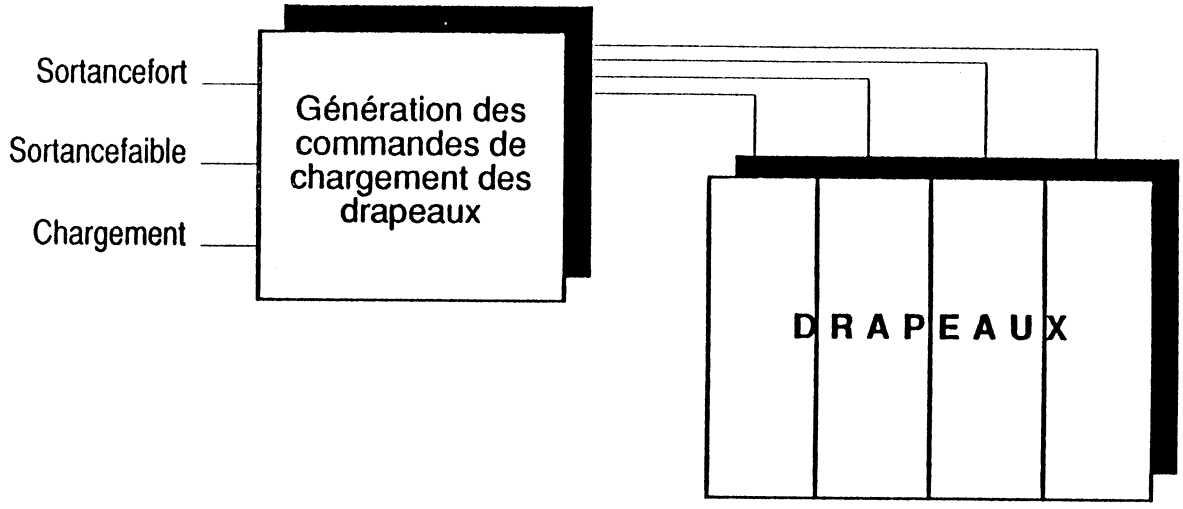


Figure 3.8. : chargement des drapeaux

3.4.7. Le calcul du nombre d'entrées réalisées

3.4.7.1. Etude fonctionnelle

Les messages arrivant, dans la cellule, les uns après les autres et étant traités immédiatement, il est nécessaire de compter leur nombre de façon à connaître l'instant auquel ils sont tous arrivés. A ce moment là, la cellule a terminé son pas de simulation puisqu'elle a effectuée les opérations au fur et à mesure de l'arrivée des messages.

Pour ce faire plusieurs solutions ont été envisagées :

- compteur,
- décompteur,
- registre à décalage, à jeton (registre à décalage comportant un seul point de mémorisation à 1, tous les autres étant à 0).

Le choix s'est porté vers le registre à décalage à jeton. Cette solution nécessite plus de transistors que les deux autres mais elle a un fonctionnement très simple : à chaque entrée, un ordre de décalage est émis et quand le jeton arrive en bout de registre le pas de simulation est terminé. De plus la modification de l'entrance ne nécessite que la modification d'un décodeur 1 parmi N et l'ajout de points de mémorisation dans le registre à décalage. Ce qui peut facilement se réaliser de manière automatique.

3.4.7.2. Réalisation logique

Un décodeur 1 parmi N génère 8 sorties dont une seule est à 1 et les 7 autres à 0, à partir de l'entrance (codée sur 3 bits). A la réception du signal "remise à 0 du compteur", ces 8 valeurs sont chargées à la bonne place dans un registre à décalage. A chaque entrée celui-ci reçoit un ordre de décalage. Quand la valeur 1 arrive au bout, le pas de simulation est terminé.

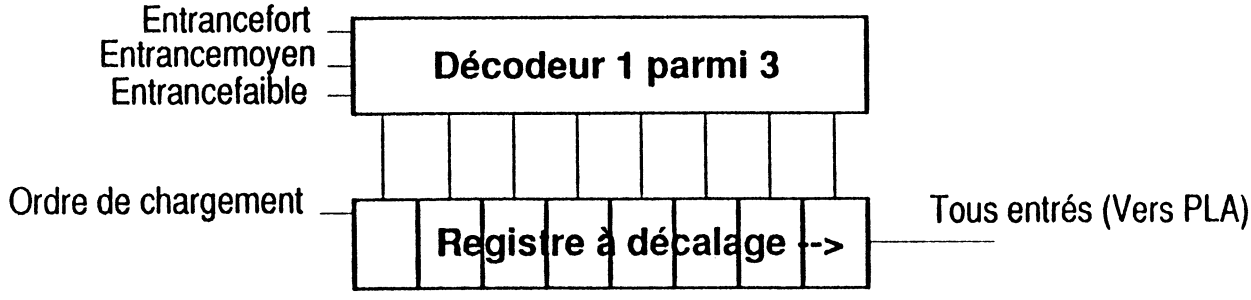


Figure 3.9.: Réalisation logique

3.4.8. Le séquenceur

Le séquencement de ces diverses entitées entre elles, est réalisé par un PLA qui a été généré automatiquement par le logiciel GASP.

Le fonctionnement du séquenceur doit être défini sous forme de grafcet. A partir de celui-ci et d'une bibliothèque de cellules de base, GASP génère un PLA avec ses bascules ainsi qu'un routage entre eux.

3.4.8.1. Caractéristiques du PLA

Le PLA est constitué de deux matrices : une matrice ET qui calcule les termes produit et une matrice OU qui calcule la somme

des termes produits. GASP génère un PLA dynamique à précharge de type DOMINO. Nous avons rendu les entrées statiques par échantillonnage.

La précharge du PLA intervient sur $\overline{H}$ et le calcul sur H.

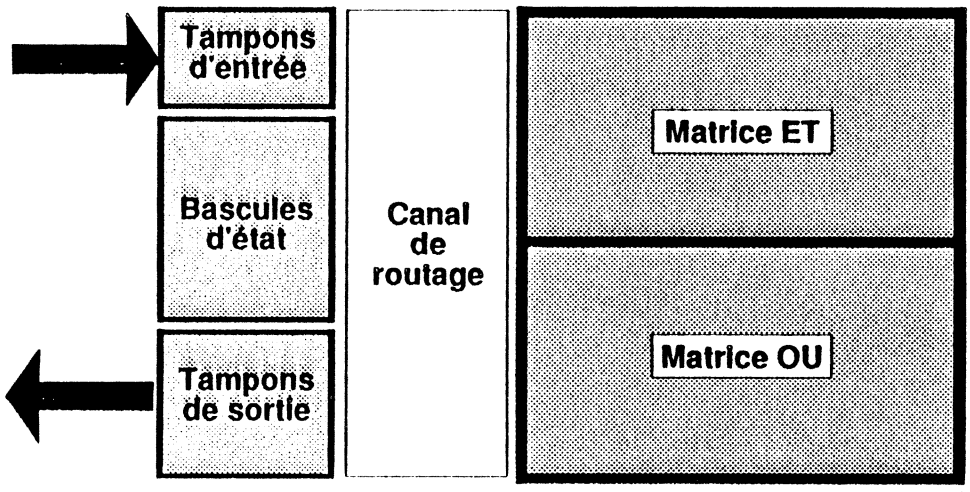


Figure 3.10. Schéma fonctionnel du séquenceur

3.4.8.2. Le séquencement

Le fonctionnement de la partie traitement comporte deux phases successives : l'initialisation et la simulation.

La phase d'initialisation est une boucle, activée par le signal RESET, de laquelle le séquenceur sort à la réception du signal FINREMPLIR. Ce signal indique la fin du remplissage du registre à décalage qui comporte les données nécessaires à la simulation.

L'initialisation se déroule grâce à N pas successifs, l'ordinateur hôte pilotant l'initialisation :

Tant que non tous émis
émettre GO
émettre le nième message des NxM cellules par la meilleur entrée possible
attendre END
Fin tant que

Lors de l'initialisation, l'ordinateur hôte envoie des valeurs dans le réseau à partir de cellules périphériques. Les messages qui arrivent aux cellules ne comportent pour toute information, que leur contenu. Il semble clair, qu'il faut que ces messages arrivent dans la cellule, dans le même ordre que lors de leur départ de l'ordinateur hôte. Une solution simple serait d'entrer tous les messages destinés à une même cellule par la même cellule périphérique.

Il se peut pour diverses raisons (cellules plus lentes, cheminement plus long, ...) qu'à un moment, un "bouchon" se forme entre l'ordinateur hôte et une cellule périphérique, les autres entrées du réseau étant libres. L'idée est de permettre, dans ce cas, à l'ordinateur hôte d'utiliser les autres entrées du réseau. Mais on peut, alors, faire varier l'ordre des arrivées dans une cellule. Pour remédier à cela, on découpe l'initialisation en pas commençant par l'émission d'un GO et se terminant par la réception de tous les END émis par les cellules ; ainsi, puisque chaque cellule ne recevra qu'un message par pas, l'ordinateur hôte peut choisir (en indiquant le bon dx et dy, bien évidemment) l'entrée qu'il veut pour n'importe quel message.

Comme nous l'avons vu au paragraphe 3.1., la phase de simulation débute par l'émission des résultats (si le délai n'est pas nul), suivie de la réception des n entrées et du calcul qui s'effectue à chaque réception.

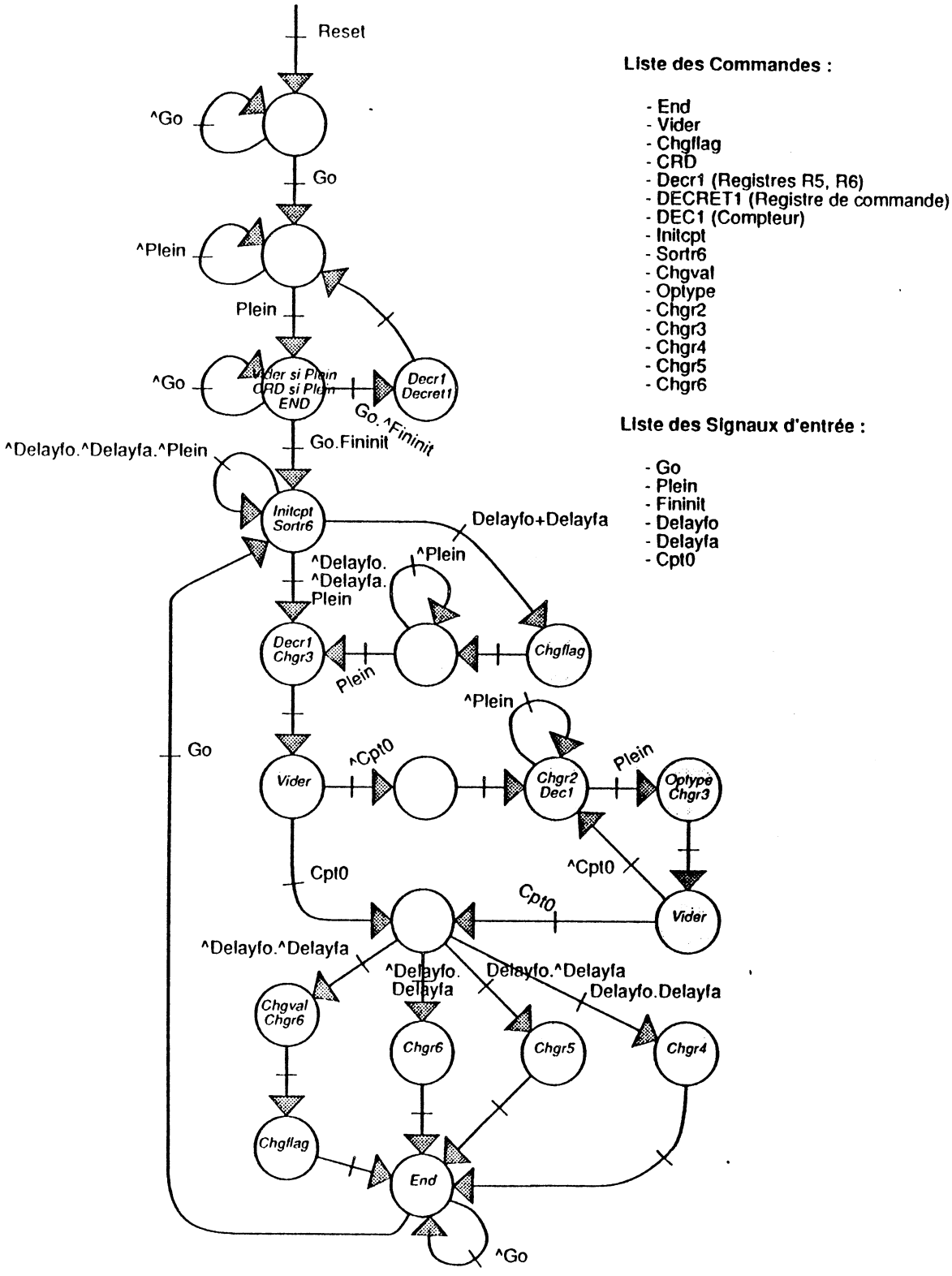


Figure 3.11. Le graphe du séquenceur

3.4.8.3 Réalisation du PLA

La génération automatique de ce PLA par GASP donne les caractéristiques suivantes :

- 6 signaux d'entrée
- 16 signaux de sortie

3.4.8.4. Implantation

L'entité de séquencement comporte donc le PLA, les bascules d'état et les registres Maître/Esclave d'échantillonage des entrées sorties. L'implantation est donnée en figure 3.12.

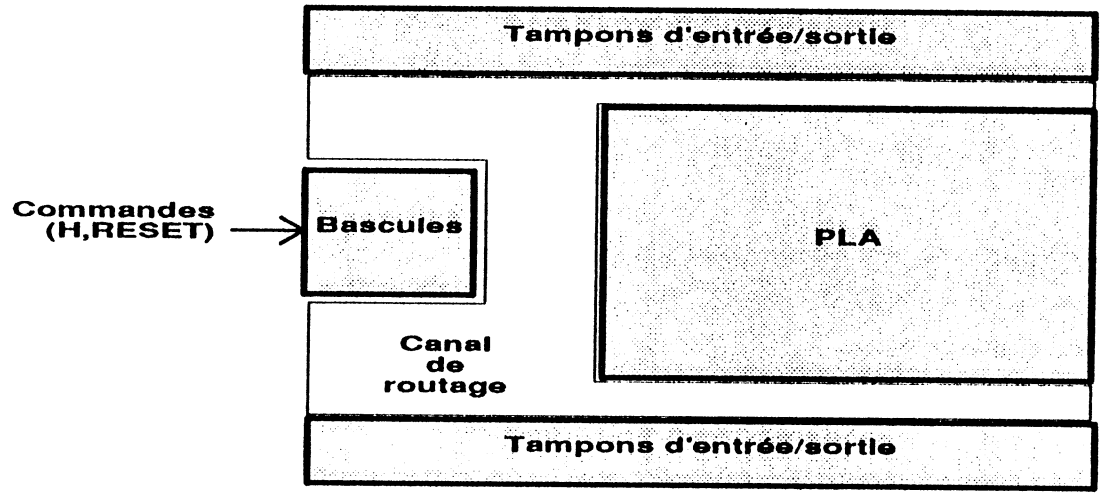


Figure 3.12. Implantation du séquenceur

3.4.9. Première réalisation

Un premier circuit a été réalisé en 1987. Ce circuit consistait en une seule cellule autour de laquelle avait été réalisé un système de multiplexage pour pouvoir entrer en parallèle dans le réseau les 10 bits d'un message.

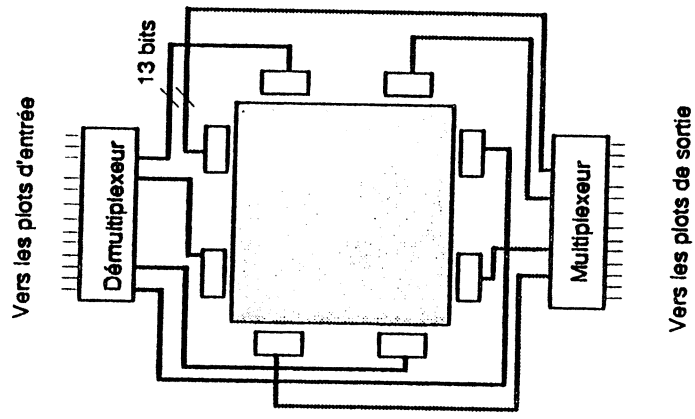


Figure 3.13. le système de multiplexage

Cette première version a été presqu'entièrement réalisée avec le logiciel LOF du CNET. Le circuit a été entièrement décrit sous forme textuel. Nous ne disposions, que de Cassiopée (Base de donnée développée au CNET, comportant des éditeurs graphiques) dont la version, de l'époque, ne permettait pas l'affichage de plus d'une centaine de transistors. Dans ces conditions, la superficie du circuit était gigantesque (avec des outils plus performants le circuit d'un réseau 2 x 2 que nous présentons dans le paragraphe suivant occupe presque la même superficie que la cellule unique !). L'assemblage final a du être réalisé au CNET sur CALMA. Nous y avons alors fait tourner des DRC et des ERC.

Ce circuit a été testé. Le mécanisme de multiplexage fonctionnait ainsi que les tampons d'entrée et de sortie. Mais les PLA générés automatiquement ne marchaient pas. Nous n'avons donc pu tester que des morceaux de l'aiguilleur et de la partie traitement.

Depuis, le logiciel Silvar-Lisco et tous ses outils de vérifications ont été installés à l'USSI, ce qui nous a permis de réaliser la deuxième version du circuit dans de bien meilleures conditions. Certains points ont été améliorés, notamment la disposition du registre à décalage. La bibliothèque de cellules de base a été entièrement redessinée de façon à obtenir des cellules plus denses que celles qui nous avaient été fournies.

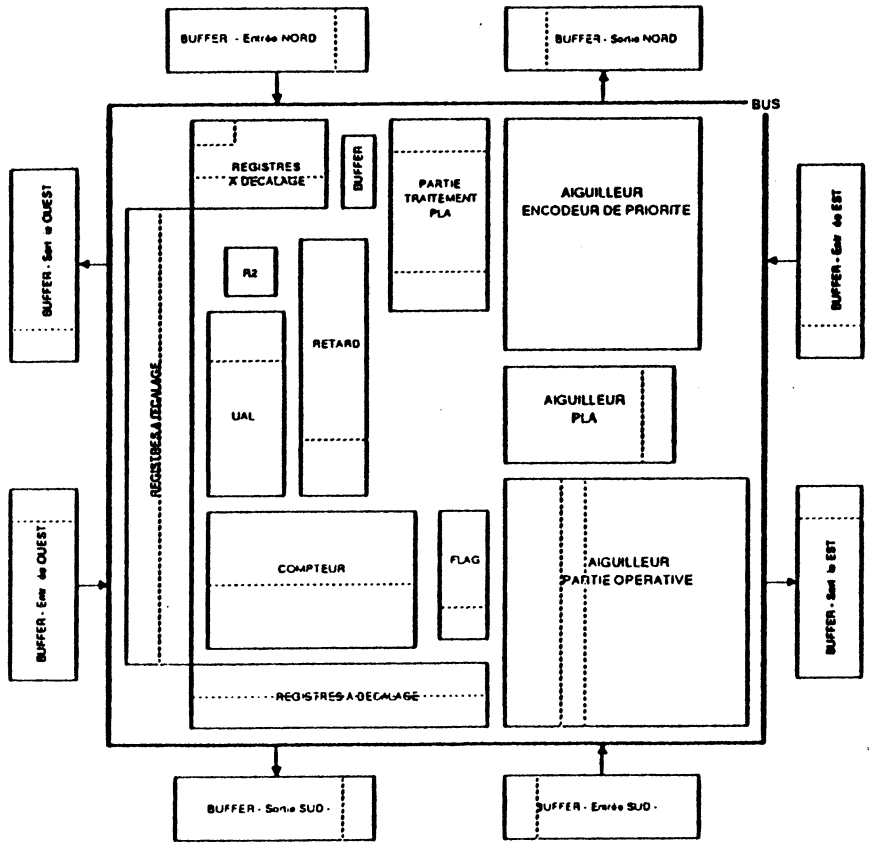


Figure 3.14. Plan de masse de la première cellule

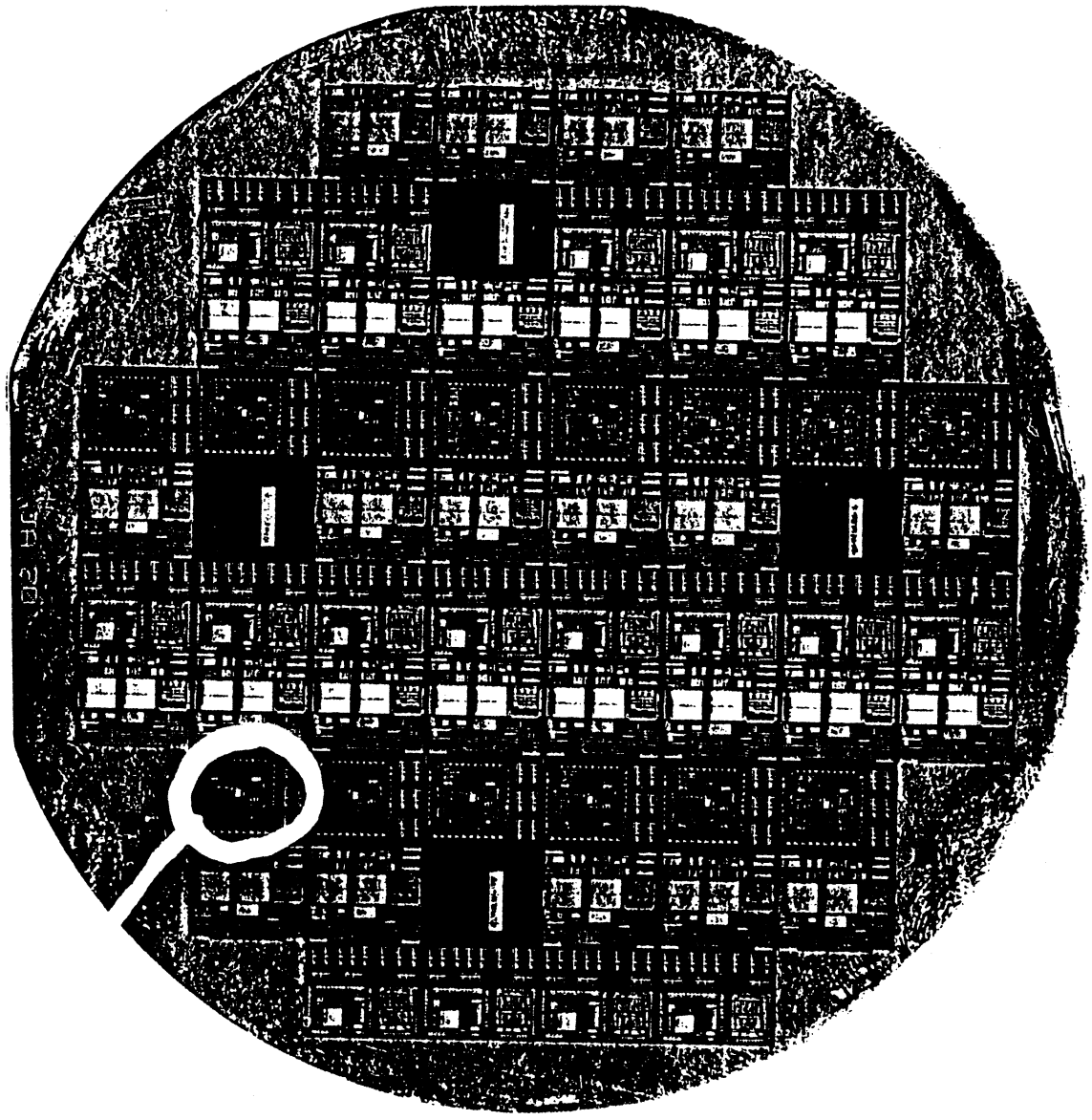


Figure 3.15. Photo du wafer contenant la première cellule

3.5. LE CIRCUIT

Le circuit réalisé, dont le test débute actuellement est composé de 2 x 2 cellules. Des interfaces série/parallèle et parallèle/série sont

disposées entre les cellules périphériques (elles le sont toutes dans ce cas !) et l'extérieur du réseau.

3.5.1. L'assemblage des cellules

Une fois, les cellules réalisées, elles sont assemblées sous forme d'une matrice. Les tampons mis en commun sont superposés. Les signaux END sortant de chaque cellule sont regroupés, un ET logique de ces signaux est réalisé de façon à générer le signal qui est communiqué à l'ordinateur hôte.

La communication entre le circuit et l'extérieur est réalisée par l'intermédiaire de plots d'entrée/sortie conçus au CNET Meylan. Ces plots sont regroupés dans une couronne. Vu leur faible nombre (28) il nous a semblé intéressant d'inclure dans cette couronne les cellules d'interface.

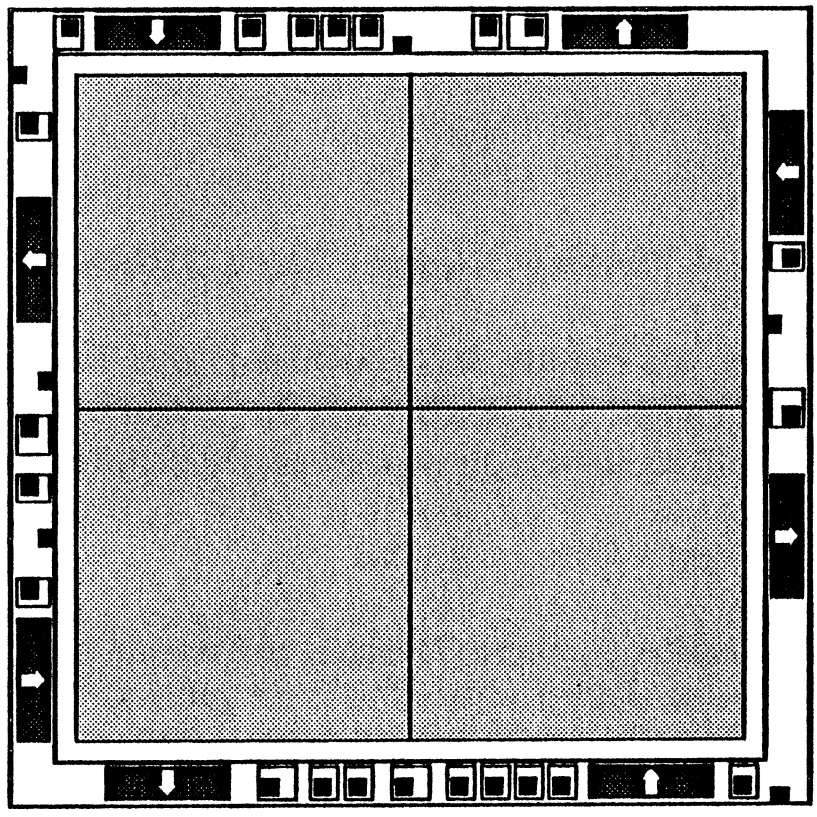


Figure 3.16. L'assemblage du réseau

La taille du circuit et les limites de la technologie nous ont obligés à ne pas affecter une cellule d'interface par tampon périphérique, nous aurions alors dépassé la côte maximum autorisée. Cette limitation permet tout de même un fonctionnement quasi normal du réseau. Par contre, nous avons pu câbler des vecteurs de test sur les tampons non pourvus d'interface. Il est important de souligner le caractère de prototype de ce circuit : des pavés de métal sont judicieusement placés de façon à faciliter le test (pose des pointes).

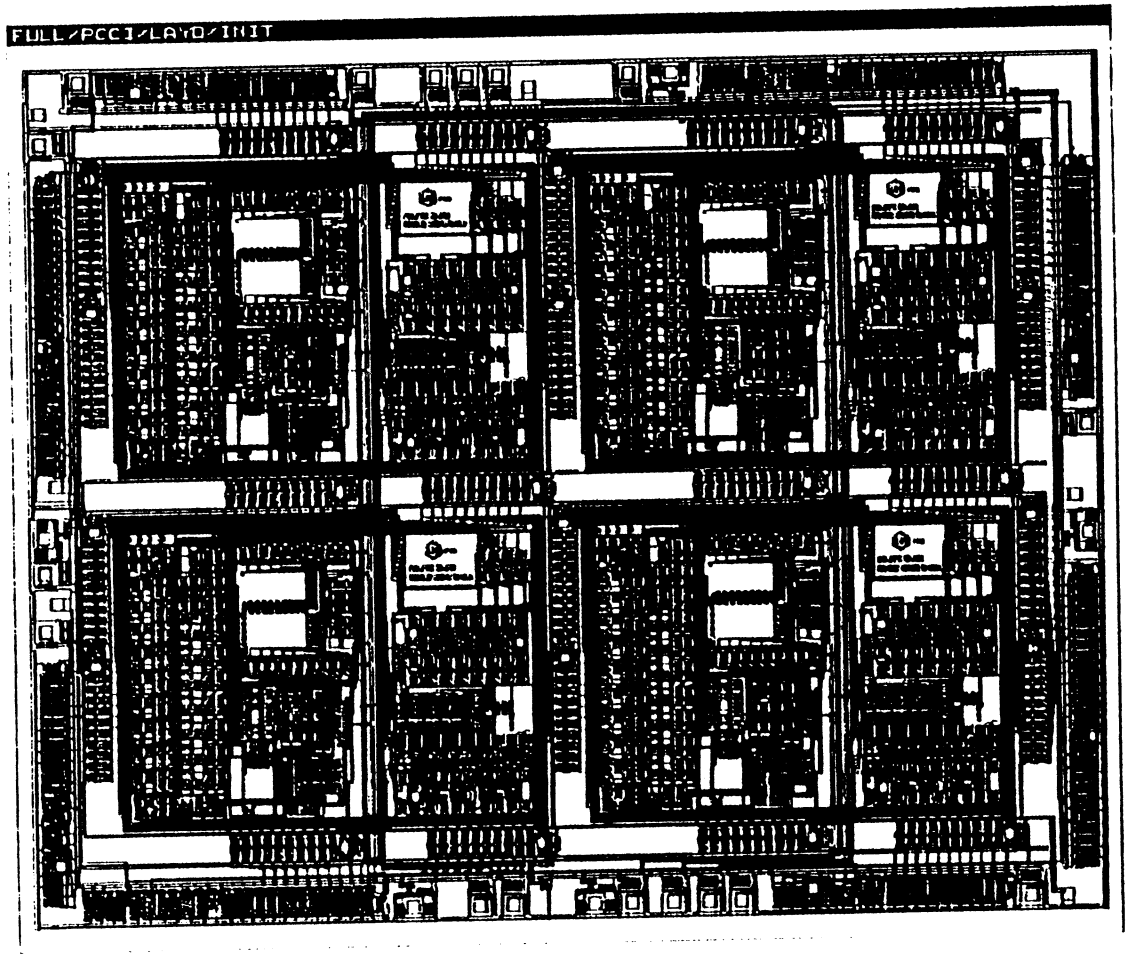


Figure 3.17. Le réseau 2x2

3.5.2 Les interfaces

But

La puissance de notre réseau augmente avec son nombre de cellules. Plus le nombre de cellules intégrées sera grand, plus le réseau sera puissant. L'intégration ayant ses limites, le réseau doit être construit de manière à être cascadable. La solution la plus simple et la plus rapide serait de connecter "pin" à "pin" chaque bascule de chaque tampon périphérique. Mais chaque tampon transmet, dans notre application, 10 bits. Dans le réseau 2x2, on dénombre 16 tampons périphériques soit 16x10 = 160 plots à réaliser ! Pour un réseau 8x8 il faudrait 640 plots. Ces chiffres ne sont pas réalistes. Le but des interfaces est de permettre la communication entre le réseau et l'ordinateur hôte ou entre deux réseaux, ceci en utilisant le moins de liens physiques possibles de façon à minimiser le nombre de plots, mais en conservant la possibilité de connexion "pin" à "pin".

Principe

Les tampons de sortie se trouvant à la périphérie du réseau émettent, vers l'extérieur, le message qu'ils contiennent et ils reçoivent un ordre VIDER qui est validé quand l'extérieur a lu le message.

Les tampons d'entrée quant à eux, reçoivent un message et ils émettent un signal VIDER lorsqu'ils ont été vidés par la cellule voisine, pour indiquer cela à l'extérieur.

Pour cela, les 2 tampons d'une même frontière de la cellule (un de sortie et un d'entrée) utilisent une seule interface émettrice et la

même interface réceptrice. Ainsi, les communications s'effectuent comme représenté sur la figure 3.24.

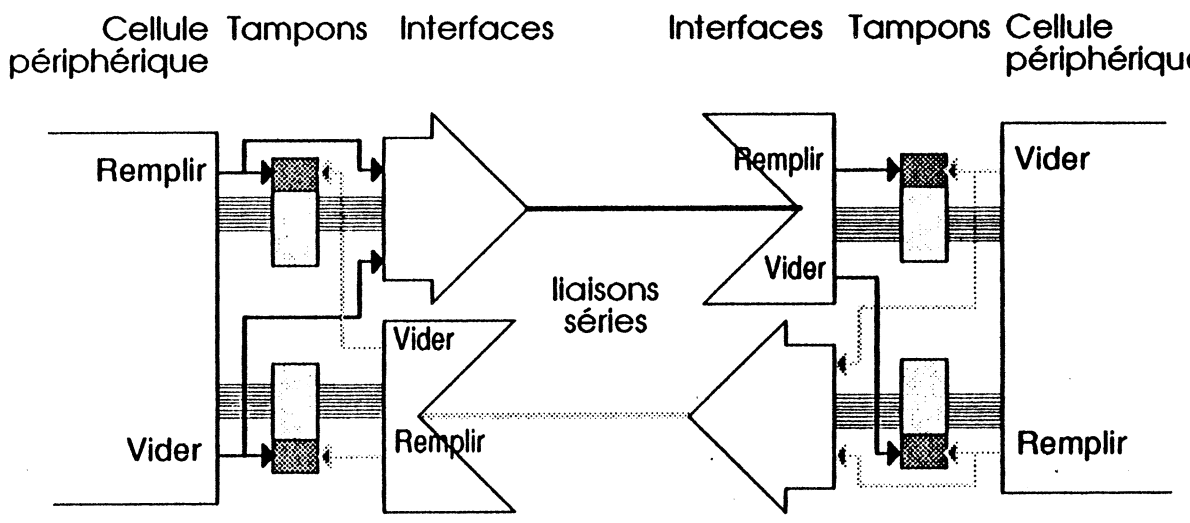


Figure 3.18. Principe des interfaces

La liaison entre un émetteur et un récepteur est une liaison synchrone (c'est à dire que l'émetteur et le récepteur bien qu'étant sur deux puces différentes, doivent recevoir la même horloge $TH_{interface} >> Th_{réseau}$) n'utilisant qu'un seul fil. Deux types de signaux transitent par ce fil, ils sont différenciés par un code :

- 0 1 1 indique l'ordre VIDER
- 0 1 0 indique un message

3.5.2.1 L'émetteur

L'émetteur est composé de deux registres à décalage à jeton :

- l'un commande l'émission du message :
  0 1 0 puis les n bits du message (n étant fixé, il n'est pas besoin d'ordre de fin de message)

- l'autre commande l'ordre vider :
  0 1 1

Au repos, quand les jetons des deux registres sont dans leur case repos, la valeur 0 est émise.

L'émetteur reçoit, en entrée, les signaux REMPLIR, venant du tampon de sortie, et VIDER, venant du tampon d'entrée. Une fois le signal REMPLIR reçu, le message contenu dans le tampon de sortie est prêt à être émis. Ces deux signaux attaquent chacun une bascule R-S. Tous les tops d'horloge $H_{interface}$ ($H_{interface} >> h_{réseau}$), ces bascules sont échantillonnées. Le programme suivant est alors exécuté :

Appelons Mém(VIDER), respectivement Mém(REMPLIR), la sortie de la bascule R-S de mémorisation de VIDER, respectivement la sortie de la bascule R-S de mémorisation de REMPLIR.

```
Si une séquence est en cours alors ne rien faire
Sinon
  Si Mém(VIDER) = 1 alors
    lancer la séquence d'exécution de VIDER
  Sinon
    Si Mém(REMPLIR) = 1 alors
      lancer la séquence d'exécution de REMPLIR
    Fin de si
  Fin de si
Fin de si
```

On peut remarquer que la priorité est donnée à l'émission de la séquence la plus courte : si Mém(VIDER) et Mém(REMPLIR) sont à 1 tous les deux, c'est VIDER qui passe en premier (3 bits). Ainsi, on évite les problèmes de famine. En effet, une fois qu'un ordre VIDER est émis, un autre ne pourra l'être que si et seulement si le tampon d'entrée qui vient de recevoir l'ordre VIDER a été re-rempli. Pour ce faire il faudra que l'interface émettrice de sens inverse émette un message, ce qui nécessite au moins (n + 3) tops d'horloge, n étant le nombre de bits de ce message. Pendant ce temps, la séquence d'émission d'un message correspondant au signal REMPLIR qui a été reçu, peut commencer dans quel cas, elle ne sera plus interrompue.

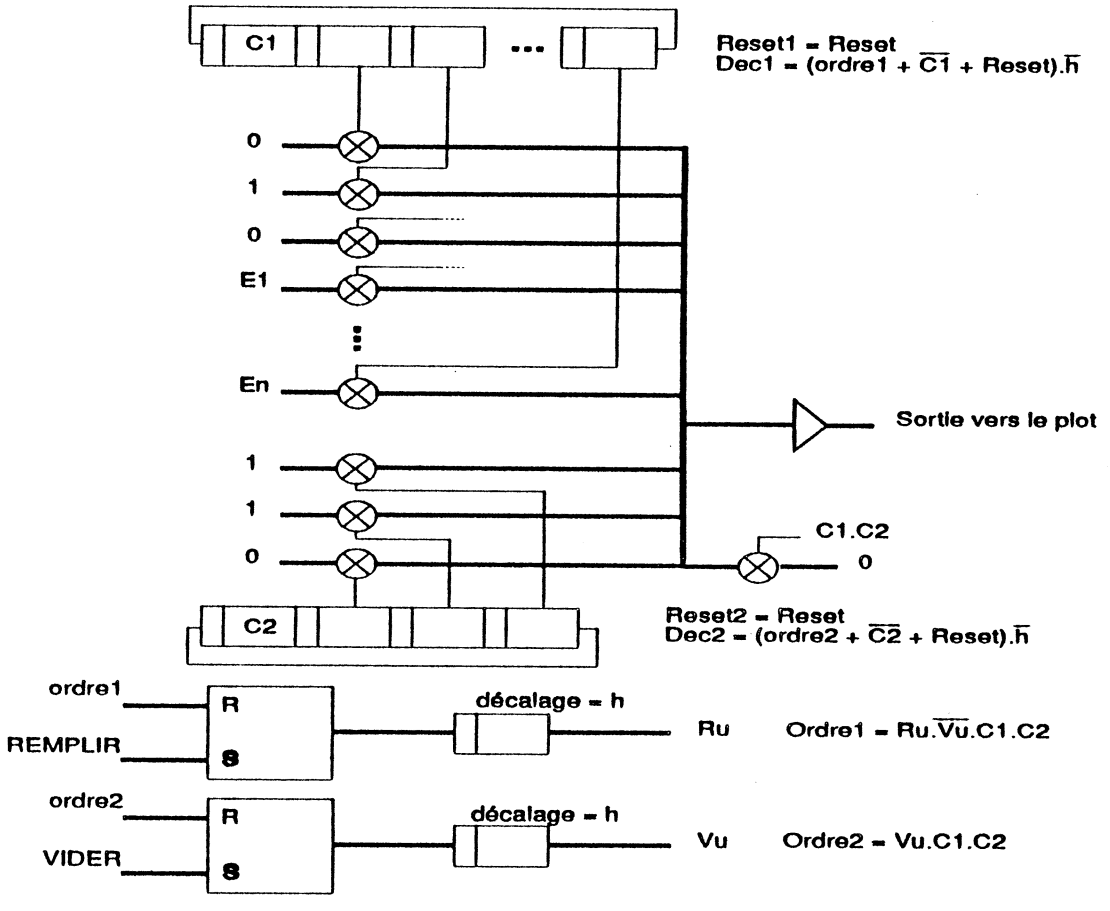


Figure 3.19. Schéma logique de l'émetteur

3.5.2.2. Le récepteur

Le récepteur, au repos, reçoit les entrées venant de l'émetteur associé, dans un registre à décalage, dit de réception, conservant les trois dernières valeurs. Quand la séquence de VIDER est

reconnue, un ordre VIDER est émis vers le tampon de sortie. Quand la séquence "début de message" est reconnue, les valeurs arrivant sont aiguillées vers un autre registre à décalage qui va conserver les n bits du message de façon à remplir le tampon d'entrée d'un seul coup. Un ordre REMPLIR sera ensuite émis vers ce tampon. Le registre à décalage de réception est remis à 0 pour éviter la reconnaissance d'une fausse séquence.

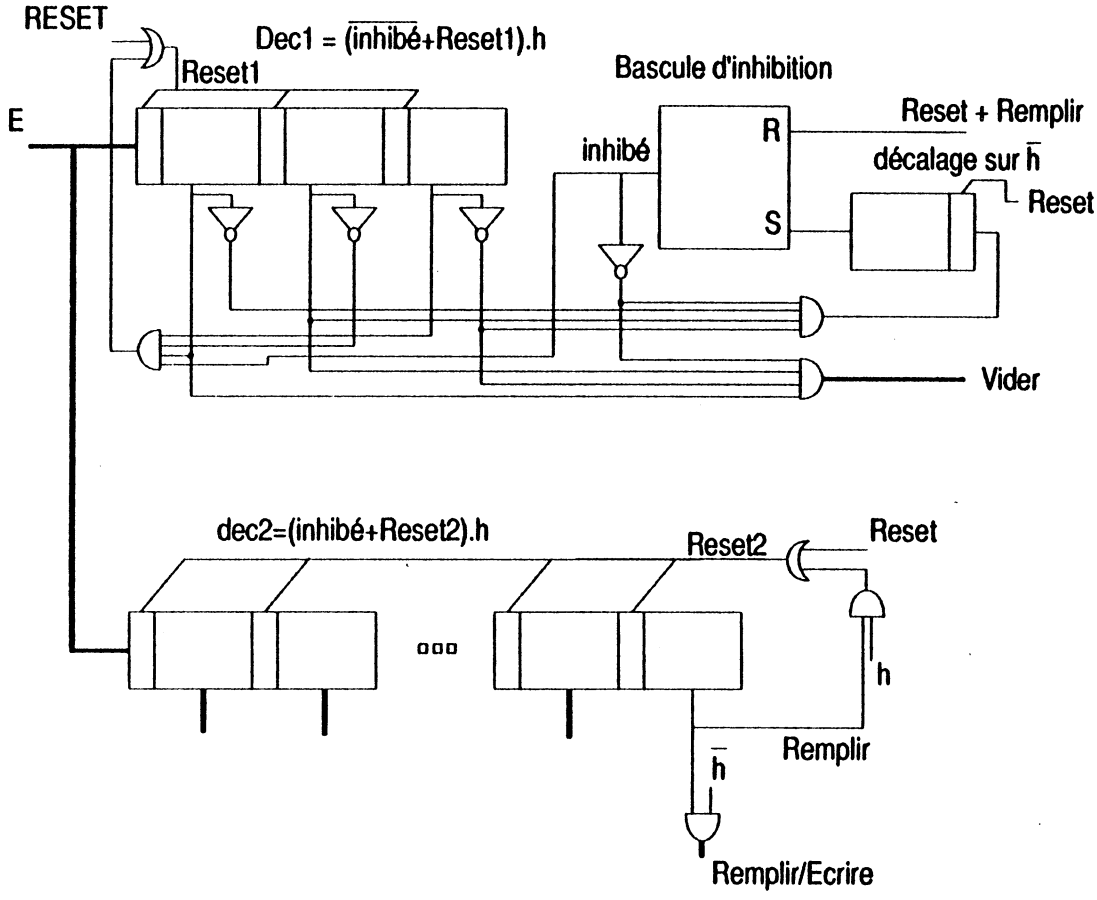


Figure 3.20. Schéma logique du récepteur

3.6. REALISATION ET TEST

Cette deuxième version a été réalisée, à partir de cellules de base que nous avons dessinées et caractérisées. Entre la Version 1 et la version 2, tout n'était pas à reprendre : le fonctionnement et la logique n'ont pas été changés. Par contre, le changement de bibliothèque nous a obligés à reprendre les simulations électriques.

Le circuit, ainsi que la bibliothèque, ont été décrits sous l'éditeur graphique de Silvar-Lisco (Princess). Pour chaque bloc, nous avons fait tourner des vérificateurs de dessin (DRC), électrique (ERC) et des extracteurs ce qui nous permettait de faire tourner des simulations électriques (ELDO) sur le schéma extrait. Sur le site de Grenoble, nous avons joué le rôle de précurseur puisque nous avons été une des deux premières équipes à utiliser cette chaîne de CAO.

Nous avons pu suivre ce plan de travail jusqu'au niveau cellule. Pour réaliser ces vérifications au niveau du circuit entier, nous avons utilisé les moyens techniques (Dracula) que le CNET a mis à notre disposition, les microvax ne pouvant traiter d'aussi gros problèmes.

Le circuit total comporte 32639 transistors. Chaque cellule (on ne compte que 4 tampons par cellule) est composée de 6421 transistors le partage se faisant comme suit :

- 280 transistors par tampon
- 3868 pour la partie traitement
- 1433 pour l'aiguilleur.

Le circuit global occupe une superficie de 8,8 x 7 mm^2.

Le circuit est en cours de fabrication à ce jour (fin juillet 1988). Il devrait arriver à la mi-août, il sera alors testé. Nous utiliserons pour cela le matériel du CIME. Le test de la version précédente a été réalisé au CNET de façon à bénéficier de l'expérience des équipes travaillant dans ce centre.

3.7. PERFORMANCES DU RESEAU

Le réseau est tel qu'une augmentation du nombre de cellules augmente ses performances. Nous avons défini une architecture qui permettra dans un futur assez proche de disposer d'outils très performants et modulaires : ces performances augmentent au fur et à mesure des progrès de l'intégration sans que l'architecture ne soit modifiée.

Pour évaluer les performances du réseau, il faut déterminer la durée d'un pas de simulation. Celles-ci sont entièrement dépendantes de l'acheminement des messages dans le réseau. La phase de placement est donc importante. Nous supposons que le réseau a une bonne localité. Le temps d'un pas de simulation est donc égal soit au temps de parcours du plus long chemin s'il n'y a pas d'encombrements, soit égale au temps de parcours du chemin sur lequel il y a le plus d'encombrements.

La simulation fonctionnelle nous a permis de constater que la durée du pas de simulation est proportionnelle à la taille du chemin le plus long. Ceci est en grande part du au fait de la faible connectivité moyenne des circuits à simuler. Le chemin le plus long dans un réseau N x N est de 2N - 2. Nous supposons que le message passe directement dans la moitié des cellules et a 1

attente dans l'autre moitié (hypothèse plausible d'après les résultats des simulations fonctionnelles). Nous obtenons alors T_{max} :

$$T_{max} = (N - 1).T_{trav} + (N - 1).(T_{trav} + T_{anul}) + T_{cal}$$

T_{trav} = 300ns, T_{anul} = 250ns, T_{cal} = 150 ns

donc si N = 8, T_{max} = 6100 ns
Si N = 100, T_{max} = 84300 ns

si N = 8 : 64 portes sont simulées en 6,10 microsecondes, soit une vitesse de simulation théorique proche de 10 millions de portes par seconde, ce qui donne avec une activité moyenne de 20 %, une vitesse de simulation supérieure à 2 millions de portes par seconde.

si N = 100 : 10000 portes sont simulées en 84,30 microsecondes, soit une vitesse de simulation théorique de 118 millions de portes par seconde, ce qui donne avec une activité moyenne de 20 %, une vitesse de simulation supérieure à 23,7 millions de portes par seconde.

En partant d'une hypothèse plus forte disant que quelque soit la taille du réseau, le chemin le plus long sera toujours de 8, on obtient les résultats suivants :

$$T_{max} = 7.T_{trav} + 7.(T_{trav} + T_{anul}) + T_{cal}$$

T_{trav} = 300ns, T_{anul} = 250ns, T_{cal} = 150 ns

donc, quelque soit N, T_{max} = 6100 ns.

Ce qui signifie en effectuant les même calcul que précédemment, que si N = 8 la vitesse de simulation sera supérieur

à 2 millions d'événements par seconde et si N = 100, la vitesse sera supérieure à 327 millions d'événements par seconde.

La courbe suivante montre la zone des gains en performances, en fonction de l'augmentation du nombre de cellules.

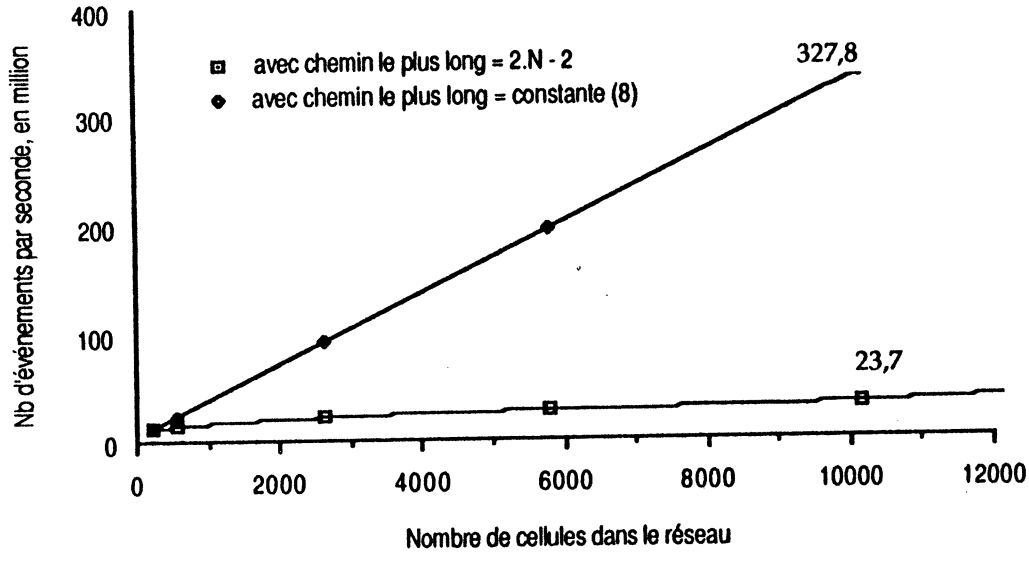


Figure 3.21. les gains en performances

On peut considérer que l'augmentation des performances en fonction du nombre de cellules dans le réseau est compris entre ces deux courbes.

On peut tout de même noter deux faits importants, c'est que même dégradées par les communications entre le réseau et l'ordinateur hôte, ces performances sont bien au-delà de celles qu'offrent les accélérateurs classiques et surtout, les performances augmentent de manière significative avec la taille du réseau.

Ces performances montrent l'efficacité de l'architecture cellulaire du réseau. A titre de comparaison, la vitesse maximale des machines de simulation logique actuelles est de l'ordre de quelques milliards d'événements par seconde (the Yorktown Simulation Engine d'IBM, the System Development Engine de Zycad).

On peut tirer de ces résultats une estimation des performances de la cellule indépendamment de la technologie.

Performance de l'architecture

C'est une estimation en terme de nombre de portes traversées et plus précisément, en terme d'équivalent inverseurs traversés. La caractérisation d'un inverseur de taille minimal commandant un autre inverseur, également de taille minimal, dans la technologie utilisée (CMOS 1 métal 2 microns du CNET), donne un temps de commutation de 0,5ns.

La complexité en temps de l'aiguilleur équivaut alors à la traversée de 300 x 0,5 = 600 inverseurs cascadés.

On peut, à partir de cette mesure, qualibrer les résultats précédents en fonction d'une technologie donnée. Si l'on compare la technologie utilisée par rapport à une autre plus récente du CNET (2 métaux 1 micron), cette dernière est 2 fois plus rapide : on obtient alors un temps de traversée de l'aiguilleur de : 600 x 0,25 = 150ns.

Performances en terme de surface

On peut réduire la surface d'intégration du circuit en changeant de technologie. Deux points contribuent à ce gain :

- réduction de la largeur minimale de grille : par exemple les technologie 1 micron,

- augmentation du nombre des niveaux métalliques : par exemple 2 métaux.

Les circuits réalisés au CNET montre que l'intégration augmente d'un facteur 3 entre les technologies 2 microns (CMOS-1) et 1 micron (CMOS-2). Dans le cas de notre circuit, on estime le gain supplémentaire en surface, apporté par un 2ème niveau de métal, à 20%. En effet, la faible densité du circuit s'explique, en partie, par une forte connectique qui occupe une surface importante.

Le tableau ci-après résume les caractéristiques principales entre les technologies CMOS-1 et CMOS-2 du CNET et les estimations de la densité d'intégration de notre circuit sur cette dernière.

	CMOS2 (circuit réalisé)	CMOS1 (circuit réalisable sans modifications de fond sur la technique de conception)	CMOS1 (moyenne des circuits du CNET, réalisable en dessinant des cellules de bases plus importante)	CMOS1 (techniques industrielles de conception)
Densité (en tr/mm2)	522	2000	3000	4000
Surface du circuit (en mm2)	62,5	16,7	10,9	8,2

Figure 3.22. Comparaison de la taille du circuit en fonction des technologies et de la méthode de conception

Réalisé de manière industrielle, c'est-à-dire en densifiant au maximun le dessin des masques donc en adoptant des techniques de conception autres que celles que nous avons utilisé, avec une technologie à 1 micron les circuits atteignent une densité

d'intégration de 4000 mos/mm². Avec une telle densité, la taille de notre circuit serait de 8,2 mm² et l'on pourrait intégrer un réseau 6x6 sur une surface comparable à celle qu'il occupe actuellement.

Chapitre 4

Etude d'un placeur

ETUDE D'UN PLACEUR

4.1. L'ALGORITHME DE PLACEMENT

4.1.1. But de l'étude

Le réseau présenté précédemment a eu comme première algorithme implanté un algorithme de simulation logique. Il est rapidement apparu que le point fort du réseau était son mécanisme d'acheminement des messages. Partant de ces constatations, plusieurs types d'applications ont été étudiées, de façon à déterminer si elles étaient implémentables sur le réseau.

L'étude qui va être présentée dans ce chapitre, est destinée à montrer la faisabilité d'un réseau dédié au placement. Pour cela, les mêmes outils que pour la simulation logique ont été utilisés : le réseau avec sa nouvelle application a été simulé en OCCAM. Les résultats ayant semblé intéressants, une architecture qui réutilise un maximum d'éléments de la machine de simulation a été proposée.

4.1.2. Algorithme de placement

L'algorithme choisi est un algorithme d'amélioration de placement par échange de paires [IOS83] : à partir d'un placement initial quelconque, nous réalisons des paires de cellules et nous échangeons de place, les cellules à l'intérieur de ces paires, si un gain est obtenu. La formation des paires, les calculs de gain et les éventuels échanges se réalisent en parallèle pour chaque paire.

Définissons, à ce stade, une convention de vocabulaire : le réseau dont on doit réaliser le placement est composé de "portes", tandis que notre réseau physique est composé de cellules.

A chaque cellule nous associons une porte.

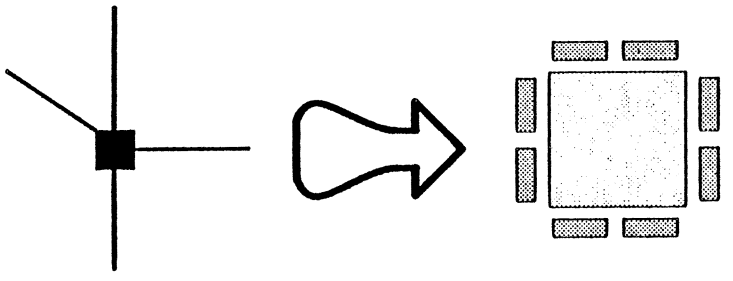


Figure 4.1. Affectation d'une porte à une cellule

Cet algorithme se décompose en trois phases devant être réalisées l'une après l'autre, le travail de ces trois phases pouvant être réalisé en parallèle :

- phase n°1 : **formation** des paires
- phase n°2 : **calcul** pour la décision de l'**échange** et échange éventuel
- phase n°3 : reconstitution de la **cohérence** du réseau.

• **<u>phase n° 1</u>** :

Les paires sont constituées par un mécanisme programmable interne aux cellules. A la fin de cette phase chaque porte doit se trouver dans l'un des états suivants :

- **supérieur** : la cellule jouera lors de la phase suivante, le rôle prépondérant dans la paire (calcul de la décision de l'échange).

- **inférieur** : la cellule transmettra à sa cellule appariée, les données nécessaires au calcul de la décision d'échange et elle recevra le cas échéant l'ordre d'échange.

- **nul** : La cellule concernée, n'appartient à aucune paire et n'est pas concernée par le cycle d'échange en cours.

Le mécanisme qui détermine l'état d'une cellule est constitué d'une table et d'un pointeur. Dans cette table, on trouve les états successifs de la cellule et l'adresse relative de sa cellule appariée. En jouant sur le nombre d'états mémorisés, on peut jouer sur la politique de formation des paires.

Cette politique peut s'avérer capitale pour la convergence rapide du placement. Par exemple : le parallélisme engendre des phénomènes d'oscillation qui ne peuvent s'éliminer qu'en formant les paires de manière originale [CHY83]. En ne travaillant qu'avec des paires que nous appellerons de voisinage (paires dans lesquelles les deux portes sont au contact l'une de l'autre, verticalement ou horizontalement) le risque d'obtenir un optimum local est élevé. Pour le réduire, il semble intéressant de former de temps à autre des paires "trouées" (les deux portes pouvant se trouver relativement éloignées l'une de l'autre).

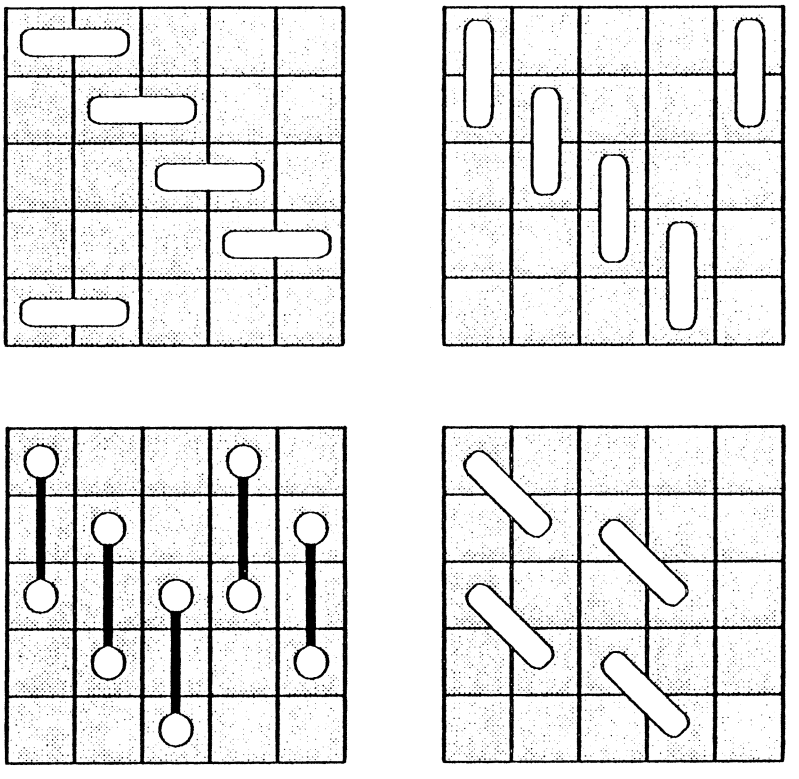


Figure 4.2. Une politique originale de réalisation des paires

• **<u>phase n° 2</u>** :

La décision de l'échange est effectuée localement. Il en découle que chaque paire considère qu'elle est seule à échanger ses portes. (phénomène d'oscillation).

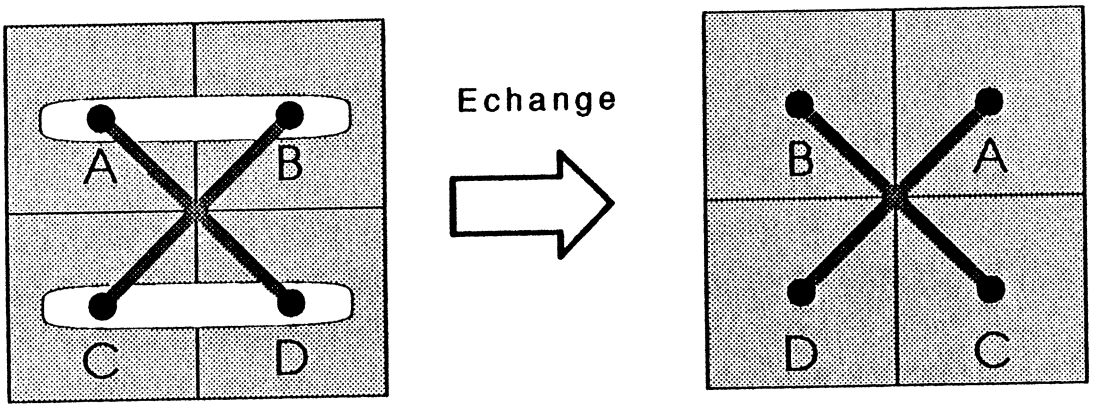
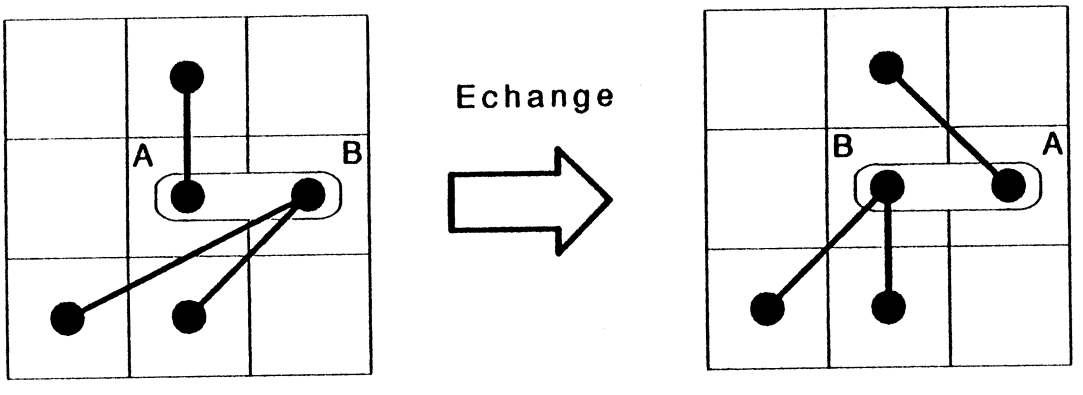


Figure 4.3. le phénomène d'oscillation

Cette décision se déroule de la manière suivante : la cellule supérieur calcule la longueur des connexions qui partent de sa porte associée et de son appariée et elle réalise le même calcul dans le cas où les deux portes auraient permuté. Si le gain est positif ou nul l'échange est effectué.



Longueur des connexions partant de A : 1
Longueur des connexions partant de B : 5
Total : 6

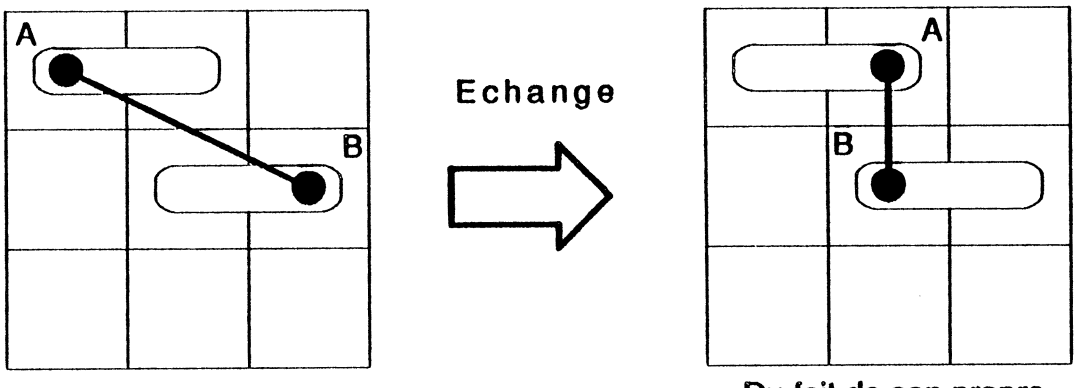
Longueur des connexions partant de A, si changement : 2
Longueur des connexions partant de B, si changement : 3
Total, si changement : 5

5 < 6, donc échange

Figure 4.4. la décision de l'échange

L'échange peut être décidé même dans le cas d'un gain nul, de manière à éviter d'aboutir à certains optimums locaux.

Il est important de noter que lors d'un échange, les cellules auxquelles sont associées les deux portes modifient l'adresse des cellules auxquelles elles sont connectées. Mais ceci ne suffit pas pour assurer la cohérence du réseau : les cellules auxquelles elles sont connectées ont très bien pu changer de place, elles aussi !



déplacement connu par A, pour aller à B :

dx = 2
dy = 1

Du fait de son propre changement de place, A suppose que B se trouve à :

dx = 1
dy = 1

La cellule A n'a connaissance que de son propre changement, les valeurs dx et dy sont modifiées en conséquence.

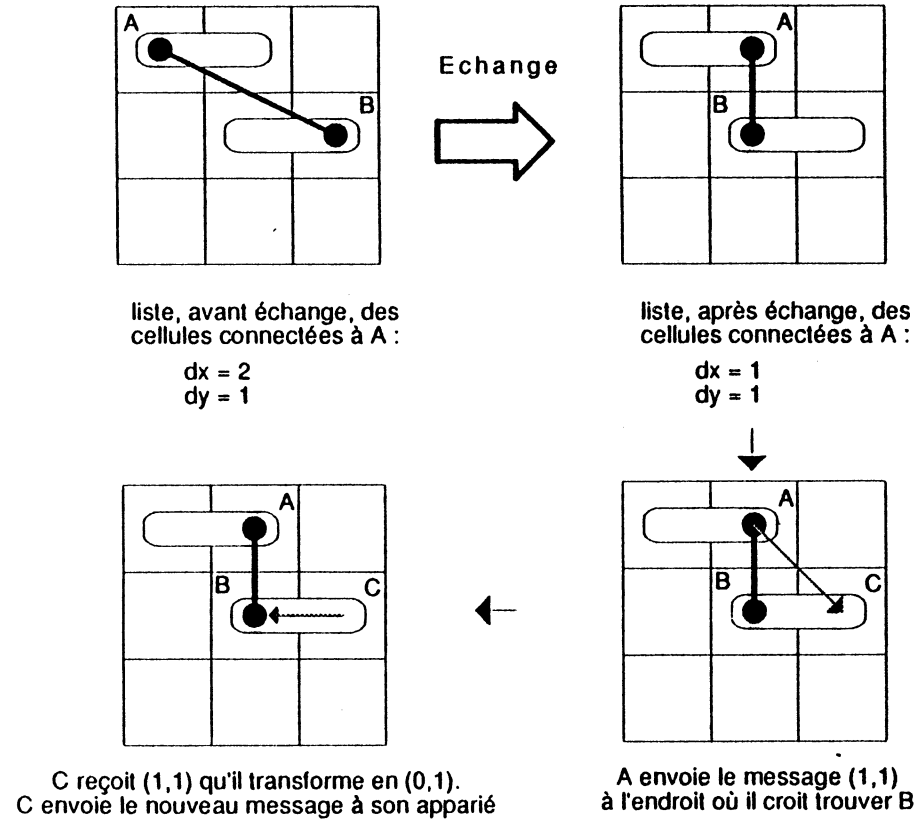
Un rétablissement de la cohérence est nécessaire .

Figure 4.5. réseau dans un état incohérent

• **phase n° 3** :

La cohérence du réseau est rétablie lors de cette troisième phase : Les cellules envoient un message à l'endroit où elles croient trouver leurs partenaires (porte à laquelle la

porte associée à la cellule émettrice, est connectée). Ce message comporte le déplacement effectué pour arriver à destination, le code de la porte à laquelle le message est destiné, le code de la cellule émettrice. Chaque message arrive soit au partenaire, soit à son apparié qui fait suivre après avoir modifié le message. Le partenaire incorpore alors le déplacement opposé à celui contenu dans le message dans la liste de ses déplacements.



En prenant les valeurs opposées à celles contenues
dans le message, c'est à dire (0,-1),
le point B sait qu'il est connecté à une porte placée à dx=0 et dy=-1.
Pendant le même temps, la même manœuvre s'est déroulée
pour rétablir la cohérence de A.

Figure 4.6. rétablissement de la cohérence

Ces trois phases doivent être synchronisées. Si la machine de placement est réalisée sur le même principe que la machine de simulation, les signaux GO et END peuvent être utilisés pour cette synchronisation. L'algorithme réalisé au niveau de chaque cellule est alors :

```
Tant que vrai
  Attendre le signal GO
  phase n°1
  émettre le signal END
  Attendre le signal GO
  Si état ≠ NUL alors
    phase n° 2
  Fin de si
  émettre le signal END
  Attendre le signal GO
  Si état ≠ NUL alors
    phase n° 3
  Fin de si
  émettre le signal END
Fin tant que
```

L'ordinateur hôte quant à lui, déroule l'algorithme suivant :

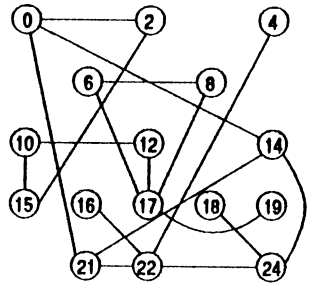
```
Initialiser le réseau
Fini := Faux
Tant que non fini
  émettre GO
  attendre les END
  émettre GO
  attendre les END
  émettre GO
  attendre les END
  Tous les n pas calculer la longueur totale des connexions
  Si depuis m x n pas la longueur n'a pas diminué alors
    Fini := Vrai
  Fin de si
Fin tant que
```

La fin du placement peut être déterminé lorsque deux relevés successifs donne la même longueur. Mais pour pouvoir sortir d'un minimum local, il peut être intéressant de donner à m une valeur plus grande.

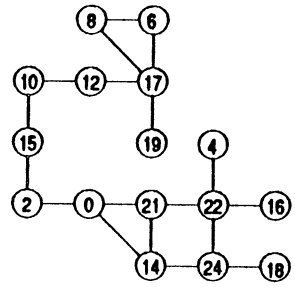
4.1.3. Simulation fonctionnelle

L'algorithme présenté précédemment a été simulé en OCCAM. Les schémas suivant montrent les résultats obtenus sur des exemples pris au hasard et sur des exemples de la littérature.

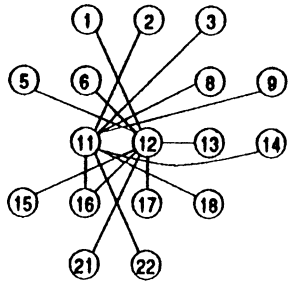
Les résultats obtenus sont satisfaisants. Avec des paires formées comme figure 4.2. (dans le sens horizontal et vertical) subissant un décalage de 1 cellule à chaque cycle nous avons vu que les résultats obtenus sont équivalents voir meilleurs que ceux d'autres machines employant des méthodes classiques. Par contre, ils sont moins bons que par une méthode de recuit simulé.



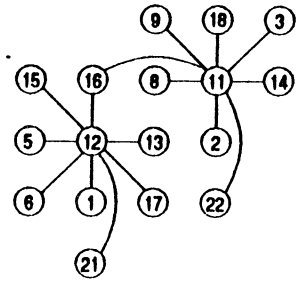
Placement initial (D=52)



Résultats (D=21)

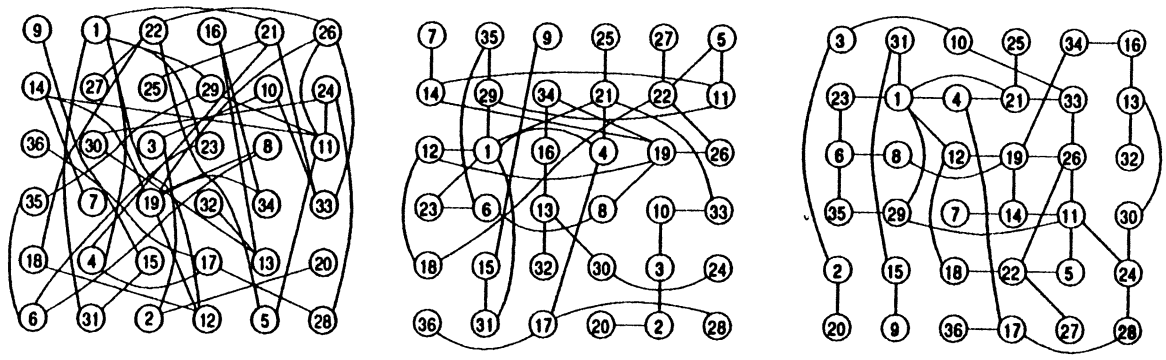


Placement initial (D=42)



Résultats (D=24)

Figure 4.7. Exemples de simulation



Placement initial

Placement final (L=97) obtenu par [UED83]

Placement obtenu par notre algorithme (L=73)

Figure 4.8. Comparaison avec un placement obtenu avec une machine classique

4.2. L'ARCHITECTURE DE LA MACHINE DE PLACEMENT

4.2.1. Premières constatations

De l'algorithme lui même, nous pouvons tout de suite déduire que la partie traitement de la machine de placement sera plus complexe que la partie traitement du simulateur logique. Cet algorithme nécessitant beaucoup de variables, le nombre de points de mémorisation à implémenter sera important. De plus il est divisé en trois phases bien distinctes, donc le séquenceur et l'UAL seront certainement plus importants que pour le simulateur logique.

La partie aiguilleur n'a pas à être modifiée. Par contre, les tampons périphériques voient leur taille augmentée. Lors de l'échange des cellules, les valeurs dx et dy doivent être échangées. Si on limite à 4 le nombre de bits de dx et de dy, il faut néanmoins 8 bits pour le champ valeur du message. Dans la phase 3, le message est encore plus long puisqu'il faut transmettre en plus le code de la porte à laquelle on envoie le message (pour un réseau 8x8, soit 64 portes, il faut 6 bits pour ce codage) et le code de la cellule émettrice, soit un message de 20 bits plus 2x4 bits pour le champs déplacement.

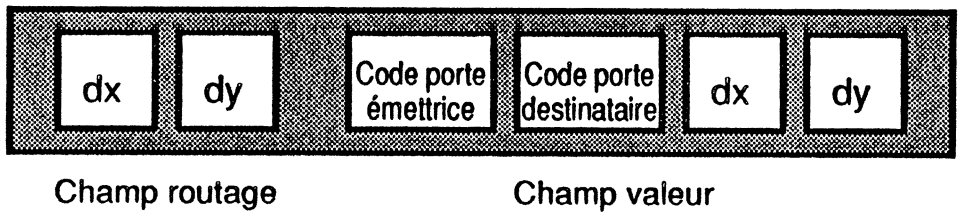


Figure 4.9. format du message

4.2.2. La mémorisation des données

Les données forment deux catégories :

- **données** destinées à la **formation des paires**. Elles sont associées à une cellule déterminée, elles ne bougeront donc pas lors de l'exécution de l'algorithme.

- **données** indiquant la **connectique du réseau** à placer. Elles sont associées à une porte, donc elles changeront de cellules lors d'éventuels échanges de portes. Des données de ce type seront échangées lors du rétablissement de la cohérence du réseau.

Plus la politique de formation des paires est complexe, plus il faut stocker d'informations. Celles-ci se présentent sous la forme suivante : 2 bits pour coder l'état de la cellule + 8 bits pour indiquer l'adresse (en déplacement relatif) de la cellule appariée, soit 10 bits par paire différente à former. On s'aperçoit que le nombre de bits à mémoriser devient vite important ! Avec la politique simple de formation des paires qui a été simulée, une cellule retombe dans le même état avec le même apparié tous les 22 cycles. Il faut donc 22 x 10 = 220 bits de mémorisation dans ce cas là. Ces chiffres montrent que le type de point de mémorisation ne peut être qu'une RAM.

Les informations de contrôle (entrance, sortance, adresses des cellules aval et amont) peuvent être mémorisées de la même manière que pour la simulation logique, c'est à dire dans un registre à décalage.

4.2.3. Les moyens de calcul

Examinons les trois phases successives et déterminons les outils nécessaires aux calculs qui sont effectués :

- **phase n°1 : formation des paires :**
  - incrémenter le pointeur de la RAM.

- **phase n°2 : décision de l'échange.et échange éventuel :**
  - il faut donc faire une somme, puis déterminer le plus grand nombre parmi deux, pour les cellules dans l'état "supérieur".

- le nombre de messages transmis entre deux cellules lors de cette phase n'étant pas uniforme, le dernier message de l'échange comporte un code réservé pour indiquer la fin de l'échange, à la cellule réceptrice.

- **phase n°3 : rétablissement de la cohérence :**
  - il se peut qu'une cellule ait à reconnaître si un message lui est destiné (identité de deux nombres) ; à ajouter deux nombres ; à inverser deux nombres (changer un bit de signe).
  - Chaque cellule doit compter le nombre de messages lui étant destiné pour déterminer la fin de son fonctionnement.

Le placeur nécessite donc une UAL beaucoup plus importante que le simulateur logique. Cette UAL doit donc pouvoir réaliser :

- une addition
- une comparaison (déterminer le plus grand de deux nombres)
- une autre comparaison (déterminer l'égalité de deux nombres)
- une inversion.

Comme pour le simulateur logique, il est nécessaire de disposer du dispositif pour compter le nombre d'entrées effectivement arrivées dans la cellule et du dispositif pour mettre à 1 le bon nombre de drapeaux de sortie.

Par contre, les messages émis en sortie sont nombreux :

- la longueur des connexions de la cellule,
- les dx, dy des cellules amont et aval et le numéro de la porte, ceci lors des changements,
- les dx, dy des cellules aval, le numéro de la porte destinataire et le numéro de la porte émettrice, lors du rétablissement de la cohérence.

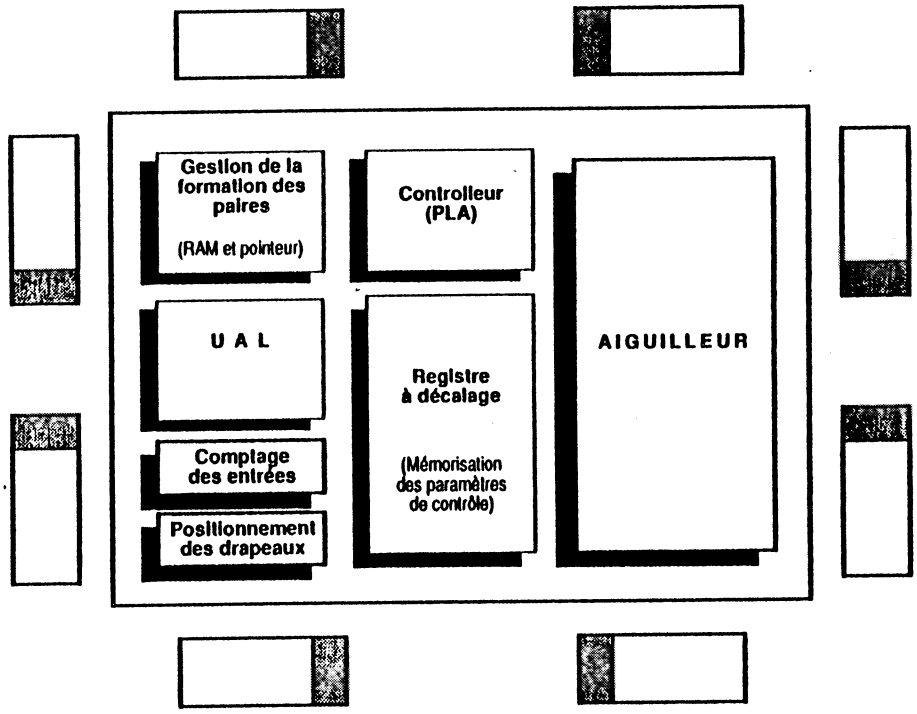


Figure 4.10. Schéma fonctionnel de la cellule du routeur

4.3. LE RECUIT SIMULE

La méthode d'optimisation par recuit simulé est une méthode non déterministe qui utilise une analogie avec les concepts de la mécanique statistique. Son but est de pouvoir sortir de minimums locaux de façon à converger vers l'optimum.

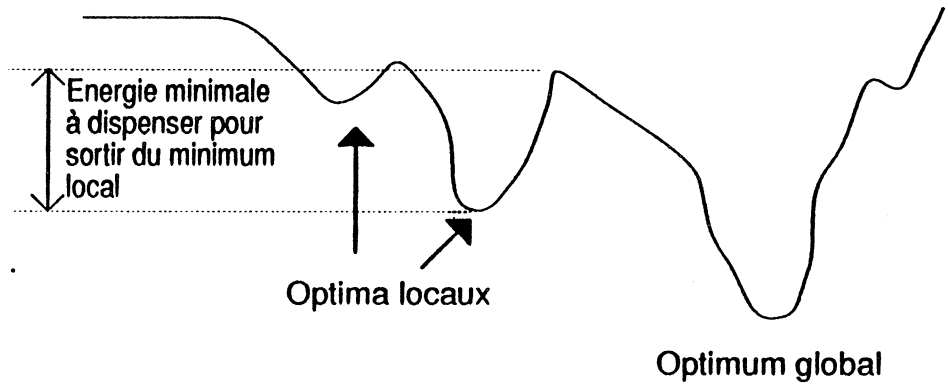


Figure 4.11. Convergence vers l'optimum

Ce paragraphe présente la méthodologie de construction d'algorithmes qui a été tirée de cette analogie, la description détaillée de cette analogie peut être trouvée dans [**SIA**85].

4.3.1. L'algorithme de Métropolis

La méthode pour simuler le processus de recuit simulé a été présentée dans les années 50 par Metropolis, Rosenbluth et Teller et se comporte comme suit :

à partir d'une configuration k, de complexité w_k, pour trouver la configuration k + 1, le processus suivant est exécuté :

tant que non fini
  perturber légèrement la configuration k
  calculer la nouvelle complexité w_{k+1}
  Si la nouvelle complexité est inférieur ou égale à l'ancienne **alors**
    accepter cette nouvelle configuration
    incrémenter l'indice k
  Sinon
    $E := e^{-\beta(w_{k+1} - w_k)}$
    tirer au hasard un nombre R compris entre 0 et 1
    Si $R \leq E$ **alors**
      accepter cette nouvelle configuration
      incrémenter l'indice k
    Fin de si
  Fin de si
  déterminer si on a fini ou pas
Fin de tant que

ß joue un rôle équivalent à celui de l'inverse de la température T. [SIA85] montre que l'évolution de cette température peut être la suivante :

$T_{k+1} := D.T_k$ avec D = 0,9

Cette valeur s'explique en faisant une analogie avec la physique : à partir d'un état à température élevée, si on baisse cette dernière lentement, on arrivera à un état cristallin ordonné qui présente une forte régularité, tandis qu'une diminution brutale

donne un état polycristallin ayant une structure totalement irrégulière.

La détermination de la température initiale T_0 est importante : il ne faut pas la prendre trop grande sous peine de converger très lentement, ni trop faible sous peine de s'enfermer dans un minimum local.

Cet algorithme peut s'appliquer aux problèmes de placement : une configuration est un placement donné et la complexité est la longueur des fils.

4.3.2. Implémentation de cet algorithme

Les résultats obtenus par cet algorithme sont généralement très bons, mais son exécution est très gourmande en temps CPU. De plus il n'est pas facile à mettre en œuvre : il ne faut pas démarrer trop chaud (temps CPU inutile) ou trop froid (mauvais résultats), il ne faut pas baisser la température trop vite (mauvais résultats) ou trop lentement (temps CPU inutile).

Pour cela, son exécution de manière parallèle semble intéressante. Une réalisation de type pipeline est en cours d'étude à Carnegie-Mellon University [KRA87].

Une manière d'implémenter cet algorithme dans le réseau est de réaliser un algorithme de recuit réparti dans chaque cellule :

comme entrée chaque cellule doit recevoir :

- l'adresse des cellules auxquelles elle est connectée,
- la température initiale,

- le facteur de diminution de la température,
- le nombre de pas à réaliser.

en cours d'exécution chaque cellule doit :

- suivre une certaine politique, prédéfinie, de formation des paires,
- calculer la longueur des interconnexions,
- décider de l'échange, pour cela il faut tirer un nombre aléatoire, calculer une exponentielle et modifier la température.

Les résultats doivent être communiqués à l'ordinateur hôte à la fin de l'exécution du nombre désiré de pas.

Connaissant la qualité des résultats de la méthode, cette solution sera très intéressante à implémenter parce que son temps d'exécution sera bon, aussi : les seuls échanges de messages entre le réseau et l'ordinateur hôte se feront au début et à la fin du placement donc le fonctionnement ne sera pas ralenti par des entrées/sorties.

Mais plusieurs problèmes délicats restent à résoudre : comment réaliser très simplement un générateur de nombre aléatoire indépendant dans chaque cellule ? comment effectuer à peu de frais un calcul équivalent à celui de l'exponentielle ? Ces problèmes, très intéressants, n'ont pas encore pu être traités ; mais un prolongement de l'étude de la machine à placer devra les aborder.

Chapitre 5

Vers un compilateur de machines cellulaires

VERS UN COMPILATEUR DE MACHINES CELLULAIRES

5.1. INTERETS

La conception d'un circuit intégré est un processus long et coûteux. Quand cela est possible il est très intéressant de pouvoir sauter quelques étapes. Nous avons proposé une architecture pour le réseau de cellules qui pourra être reprise dans bon nombre d'applications.

De façon à minimiser les temps de conception, il parait donc intéressant d'utiliser un maximum de parties communes entre les différentes applications que nous désirons implémenter sur le réseau. Avec une bibliothèque de blocs, on peut s'affranchir des différentes simulations, du dessin des masques et des vérifications de ces blocs puisque ceci a déjà été fait.

Ces blocs devront être "normalisés" de façon à pouvoir les assembler à partir d'un langage d'opérateurs flexibles. Ce qui permettrait de réduire encore plus les temps de conception et de rendre cette dernière beaucoup plus sûre.

5.2. L'AIGUILLEUR

A priori, l'aiguilleur semble réutilisable tel quel si l'on change d'application. Plusieurs réserves doivent cependant, être émises.

Tout d'abord, les différentes applications ne nécessitent pas toutes le même nombre de bits de valeur. Ceci ne pose pas de problèmes particuliers : Il faut simplement modifier le nombre de points de mémorisation dans les tampons périphériques et dans le

tampon interne de l'aiguilleur, ainsi que le nombre de fils dans le bus qui fait communiquer entre eux les tampons.

Par contre, le nombre de bits de dx et de dy peut varier (certaines applications ont une meilleur localité que certaines autres). Ceci pose quelques problèmes : la partie traitement de l'aiguilleur qui calcule le registre de sortie, est réalisée sous forme combinatoire. Il faut donc modifier les fonctions à réaliser par l'adjonction de quelques NONET. Le logiciel GASP que nous avons utilisé pour réaliser les PLA est tout a fait capable de résoudre ce genre de problèmes.

Quelque soit la taille du message et le nombre de bits pour dx et dy, le séquenceur de l'aiguilleur reste inchangé. L'aiguilleur peut donc facilement être réutilisé ou modifié selon le cas.

5.3. LA PARTIE TRAITEMENT

Par définition, la partie traitement est modifiée entre deux réseaux dédiés à deux applications différentes. Malgré cette différence plusieurs blocs sont communs entre applications. Nous différencions deux types d'entités dans la partie traitement :

- les entités de calcul,

- les entités que nous appelons de "contrôle".

5.3.1. Les entités de calcul

Dans deux réseaux réalisant des fonctions totalement différentes (placement et simulation logique, par exemple), il semble clair que les entités de calcul ne sont pas les mêmes ! Donc dans ce cas, elles ne sont pas réutilisables en l'état d'une application sur l'autre. L'effort de conception pour une application nouvelle doit se faire à ce niveau.

Malgré tout, certains blocs peuvent être communs à plusieurs applications : un additionneur peut s'utiliser dans des réseaux différents par exemple. L'utilisation de blocs existant en bibliothèque, se fait au coup par coup dans les entités de calcul. Pour les applications nécessitant des opérations un peu plus complexes que la simulation logique, l'évolution se fera vers un chemin de données "data path" analogue (en plus léger) à celui des microprocesseurs.

5.3.2. Les entités dîtes de "contrôle"

Dans les entités que nous appelons de contrôle, nous incluons le séquenceur, ainsi que les points de mémorisation des paramètres de la cellule et les processus de contrôle des entrée/sortie (positionnement des drapeaux de sortie, contrôle du nombre d'entrées effectivement parvenues à la cellule).

• **le séquenceur** :

Le séquenceur, bien évidemment, ne peut pas être réutilisé d'une application sur l'autre. Son temps de réalisation est quand même assez faible puisqu'il est généré automatiquement par GASP.

• **le positionnement des drapeaux de sortie**:

Ceci est réalisé de manière purement combinatoire. Si le nombre de tampons de sortie de la partie traitement est identique entre deux applications, on peut réutiliser exactement le même dessin. Par contre si ce nombre est modifié, il faut redessiner le contrôle du positionnement des drapeaux de sortie.

Quelque soit l'application, le nombre de drapeaux de sortie ne varie pas dans de grandes proportions : 4, 8 et 16 semblent des nombres de sorties raisonnables. On peut donc réaliser 3 contrôles du positionnement des drapeaux de sortie que l'on utilisera en fonction des besoins.

Une solution pour les applications ne nécessitant que 4 tampons de sortie serait d'utiliser un boîtier de contrôle de 16 auquel on ne pourrait passer en paramètre qu'une sortance de 4 au maximum. Cette solution, bien que réalisable, demande beaucoup de place inutile.

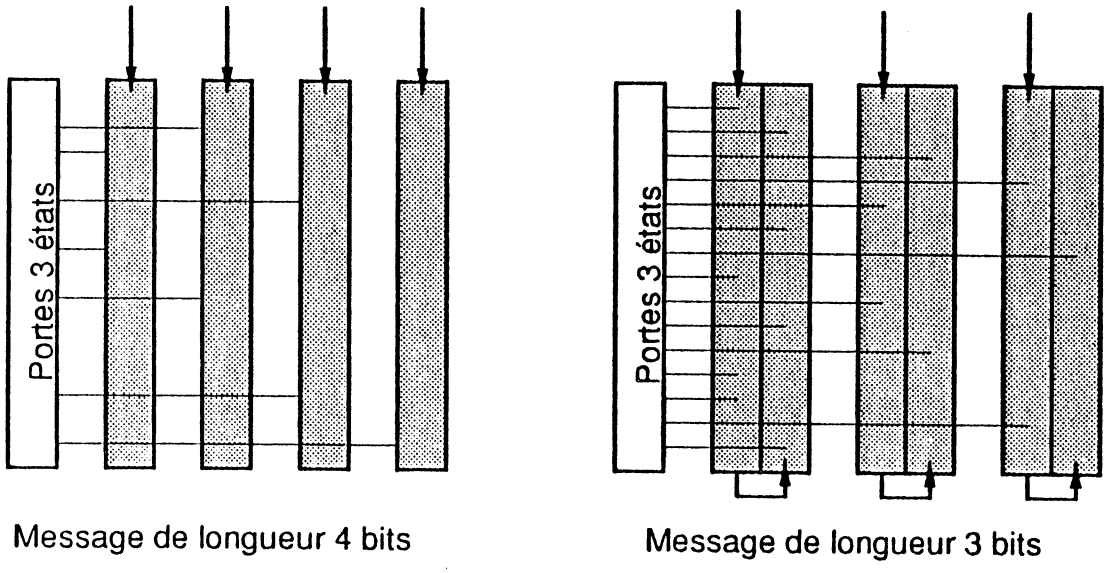
• **le contrôle du nombre d'entrées effectivement parvenues à la cellule** :

Nous avons vu au chapitre 3 que ce contrôle est réalisé par un registre à décalage dans lequel, au début du comptage, un jeton est positionné à la bonne place pour arriver en bout de registre quand le nombre d'entrées reçu est le bon.

Ce choix nous permet de paramètrer très facilement ce contrôle : pour charger le registre à décalage on utilise un décodeur 1 parmi n, ce genre de décodeur a une structure très régulière donc sa réalisation est aisée ; pour modifier l'entrance, il suffit de modifier le nombre de bascules maître/esclave du registre à décalage.

- **mémorisation des paramètres de la cellule** :

Cette mémorisation est réalisée dans m registres à décalage, m étant la longueur du champ valeur des messages (pour la simulation logique m = 2). En fonction de l'application, on réalise plus ou moins de registres à décalage, ceux-ci étant plus ou moins longs.



Message de longueur 4 bits

Message de longueur 3 bits

Figure 5.1. Registres à décalage pour mémoriser les paramètres

5.4. DES REALISATIONS DE PLUS EN PLUS RAPIDES

La méthodologie de conception que nous avons employée : *cellules de base* → *blocs* → *parties* → *cellules* → *réseau*, nous permettra de diminuer les temps de conception pour les applications futures. En effet, au fur et à mesure des différentes réalisations, nous pourrons réutiliser ce qui a déjà était dessiné et ce qui sera dessiné en plus viendra s'inscrire dans la bibliothèque pour d'autres réalisations futures.

A terme, il semble possible de pouvoir utiliser un langage d'opérateurs flexible du genre LOF (développé au CNET) pour implémenter une application nouvelle sur le réseau.

Conclusion

CONCLUSION

Le but de cette thèse était d'étudier une architecture parallèle et différentes applications pouvant y être implémentées, de mener à bien la réalisation du circuit intégré du réseau prototype de simulation logique et d'ouvrir la voie de l'automatisation de la conception de circuits cellulaires de ce type.

Ceci nous a amené à étudier les problèmes liés à la simulation logique et au placement.

Pour la simulation logique, nous avons précisé l'architecture du réseau et étudié la manière d'implémenter la partie traitement d'une cellule de ce réseau. Deux versions ont été réalisées, la première consistait en une cellule unique. Cette première version, nous a permis d'acquérir une certaine expérience dans la réalisation de circuits intégrés et de vérifier que certains choix architecturaux (bus circulaire, dispositions des alimentations) étaient les bons. La deuxième version a nécessité une étude complémentaire : il s'agissait de réaliser un réseau 2x2 pour lequel des interfaces parallèle/série et série/parallèle originales ont été définies. Cette nouvelle version, comme la première, a été fabriquée par le CNET mais nous l'avons conçue à l'USSI avec des outils totalement différents et beaucoup plus puissants que précédemment.

Le réseau que nous avons réalisé montre la faisabilité d'un tel circuit, mais pour une réalisation que nous appelerons "opérationnelle", nous devrons utiliser une technologie submicronique avec un minimum de 2 niveaux de métal pour résoudre facilement et sans perte de place les conflits lors du croisement des fils d'horloges, d'alimentations, de commandes et les bus.

Une autre question peut se poser : doit-on utiliser des cellules de base de la bibliothèque, aussi élémentaires que l'inverseur, par exemple, ou doit-on, pour gagner de la place mais en perdant du temps dans la conception, dessiner des cellules plus complexes comme un étage du registre à décalage ou un boîtier de commande de décompteur ? Le caractère plus complexe des nouvelles applications envisagées (placement, réseau de neurones, reconstruction d'images, ...) nous amènera certainement vers cette nouvelle solution [FAU88b].

Pour le placement, nous avons implémenté sur le réseau un algorithme original qui se base sur le principe d'échange de paires. Cette architecture a été simulée fonctionnellement en OCCAM. Les résultats obtenus sont prometteurs. Bien que n'atteignant pas la qualité de placement des algorithmes se basant sur le principe du recuit simulé, les résultats que donne l'algorithme original que nous avons implémenté sont meilleurs que ceux d'autres algorithmes classiques d'échange de paires.

L'amélioration de l'algorithme de placement doit maintenant, se poursuivre dans deux directions : étudier si les résultats ne seraient pas améliorés avec une nouvelle politique de formation de paires permettant de créer des paires moins régulières et étudier la manière dont l'algorithme peut être modifié pour simuler le recuit. Le traitement aléatoire peut en effet être réalisé, soit dans chaque cellule, soit au niveau de l'ordinateur hôte.

Une architecture originale

L'architecture cellulaire présentée comporte plusieurs aspects originaux : fonctionnement asynchrone, communication par échange de messages. Chaque cellule est composé d'un processeur de traitement et d'un processeur de communication. Bien que physiquement connectée qu'à ces quatre voisines, chaque

cellule peut communiquer avec n'importe qu'elle autre cellule du réseau. Ceci est réalisé par un mécanisme réparti : un message est composé de deux champs (un champ valeur et un champ déplacement), il est transporté de proche en proche jusqu'à la cellule destinataire, grâce aux indications du champ déplacement. Dans ce type d'architecture, deux phénomènes peuvent diminuer le rendement : la communication entre le réseau et le monde extérieur et la nécessité, pour certains algorithmes, de se resynchroniser , c'est à dire attendre que toutes les cellules aient terminé une action avant d'en recommencer une autre.

Un réseau très performant

Bien que les temps de communication entre le réseau et le monde extérieur soient assez importants, les estimations de performances du réseau montrent qu'elles sont comparables et bien souvent meilleures que celles d'autres architectures. Les meilleures applications à implémenter sont celles dont le nombre de paramètres est faible, ceci afin de minimiser le temps d'initialisation et dont le nombre d'entrées et de sorties en cours de fonctionnement est limité. Les autres n'utiliseront pas de manière optimale la puissance du réseau mais resteront très performantes. De ce point de vu, la simulation logique semble être une application correcte : peu de paramètres et, en régle générale, peu d'entrées-sorties.

La resynchronisation dont certains algorithmes ont besoin ralentit le réseau. De ce point de vue, la simulation logique est une mauvaise application : le temps d'exécution d'un pas de simulation est déterminé par la cellule la plus lente et par l'acheminement le plus long. Il est important de remarquer que le ralentissement dû à la cellule la moins rapide, tout en pénalisant le fonctionnement du réseau, n'est pas catastrophique. Le réseau reste malgré tout très performant malgré le fait qu'il ne soit pas utilisé de manière optimale.

Une architecture évolutive

Le point le plus important à souligner est le fait que nous avons défini une architecture dont les performances s'accroîtront dans le temps sans que nous ayons à la modifier en quoi que ce soit. Au fur et à mesure de l'évolution des techniques d'intégration, nous serons capables de multiplier les performances du réseau par le simple fait d'y ajouter des cellules.

En conclusion, il existe une voie nouvelle différente du schéma de calcul Von Neuman : celle du parallélisme massif. Cette voie nouvelle passe par des architectures cellulaires composées d'un grand nombre de cellules qui savent travailler indépendamment les unes et autres et qui savent communiquer. Cela est rendu, aujourd'hui, possible par les performances de l'intégration. Notre travail a consisté à explorer une petite partie des possibilités qu'offre cette voie nouvelle, en approfondissant les contraintes de réalisation et les résultats prévisibles de l'option que constitue le Réseau de Cellules Asynchrones.

Références bibliographiques

Références bibliographiques

[**AGR**87] "*Architecture and Design of the MARS Hardware accelerator*"
P. Agrawal, W.J. Dally, A.K. Ezzat, W.C. Fischer, H.V. Jagadish, A.S. Krishnakumar
24th ACM/IEEE Design Automation Conference

[**AMB**87] "*The application of Hardware accelerators in VLSI Design*"
A.P. Ambler, N. Colemen, R.L. Manning, N. Muhammed
Procedings of ULSI and computers, first international conference on computer technology, systems and applications.
COMPEURO 87, mai 1987

[**ANS**86] "*OCCAM for functional simulation of highly parallel architectures*"
Y. Ansade, R.Cornu-Emieux, B. Faure
ESPRIT Project Meeting, St Pierre de Chartreuse, Février 1986

[**ANS**88] Y. Ansade
Thése INPG de microélectronique

[**BAR**80] "*A computer architecture for digital logic simulation*"
R. Barto, S. Szygarda, E. Thompson
Electronic Engineering, septembre 1980

[**BER**85] "*Etude d'une machine cellulaire pour la simulation logique de circuits intégrés*"
J.P. Bernard
Thése de Docteur Ingénieur, juillet 1985

[**BLA**82] "*A bit map architecture and algorithms for design automation*"
T. Blank
PhD thesis, University of Stanford, septembre 1982

[BLA84] "*A survey of hardware accelerators used in computer-aided design*"
T. Blank
IEEE Design & Test, août 1984

[CAN85] "*Conception des circuits intégrés MOS*"
M. Cand, E. Demoulin, J.L. Lardy, P. Senn
Editions Eyrolles, 1985

[CHY83] "*A placement algorithm for Array processors*"
D.J. Chyan, M.A. Breuer
Procedings of the 20th Design Automation Conference, juin 1983

[COR86] "*Some highly parallel algorithms processed by a regular network of asynchronous cells*"
R. Cornu-Emieux, Y. Ansade, B. Faure, G. Mazaré, P. Objois
North Holland 1986, Procedings of the International Workshop on Parallel Algorithms Architectures, Luminy, 1986

[COR87a] "*OCCAM pour la simulation fonctionnelle d'une architecture hautement parallèle*"
R. Cornu-Emieux, P. Objois
Séminaire "Programmation Parallèle, Transputer et OCCAM", Talloires, mai 1987

[COR87b] "*Réseau de cellules asynchrones dédié à la simulation logique : conception et réalisation*"
R. Cornu-Emieux, D. Lattard, G. Mazaré, P.Objois
Journées Architectures de C3, Sophia Antipolis, juillet 1987.
Bigre et Globule n°56 novembre 1987

[**COR**87c] "*An integrated highly parallel architecture to accelerate logical simulation*"
R. Cornu-Emieux, G. Mazaré, P. Objois
ISELDECS-87, International Symposium on Electronic Devices, Circuits and Systems, Kharagpur (Inde), décembre 1987

[**DEN**82] "*The Yorktown simulation engine*"
M. Denneau
Procedings of the 19th Design Automation Conference, juin 1982

[**FAU**86] "*WSI asynchronous cells network*"
B. Faure, Y. Ansade, R. Cornu-Emieux, G. Mazaré
IFIP Workshop on WSI, Grenoble, mars 1986

[**FAU**88a] "*A VLSI asynchronous cellular architecture dedicated to multilayered neural network*"
B. Faure, G. Mazaré
procedings of nEuro'88, Paris, juin 1988

[**FAU**88b] "*A VLSI implementation of multilayered neural network*"
B. Faure, G. Mazaré
International workshop on VLSI for Artificial Intelligence, Oxford juillet 1988

[**GIN**83] "*CAE station's simulators tackle 1 Million gates*"
L. Gindraux, G. Catlin
Electronic Design, novembre 1983

[**HAN**87] "*Integration of a simulation accelerator for design and test*"
S.R. Hansen
Procedings of ULSI and computers, first international conference on computer technology, systems and applications.
COMPEURO 87, mai 1987

[**HIR**87] "*Simulation Processor SP*"
F. Hirose et all
publication CH2469, IEEE 1987

[**INM**84] "*Occam programming manual*"
INMOS Limited
Prentice Hall, 1984

[**IOS**83] "*A module interchange placement machine*"
A. Iosupovici, C. King, M.A. Breuer
Procedings of the 20th Design Automation Conference, juin 1983

[**KRA**87] "*Placement by simulated annealing on a multiprocessor*"
S.A. Kravitz, R.A. Rutenbar
Research report n° CMUCAD-87-33, Université Carnegie-Mellon, juillet 1987

[**KUN**82] "*Why systolic architectures*"
H.T. Kung
IEEE Computer 15, 1 (1982) pp 37-46

[**LAN**86] "*Design specifications of a hardware concurrent fault simulation accelerator*"
C. Landrault, D. Marcon, P. Pinede
Silicon Design Conference, 1986

[**LAT**88a] "*Parallel image reconstruction by using a dedicated asynchronous cellular array*"
D. Lattard, G. Mazaré
Procedings of the parallel processing for computer vision and display conference, Leeds, janvier 1988

[LAT88b] "*Une nouvelle architecture cellulaire pour la reconstruction d'images*"
D. Lattard, G. Mazaré
PIXIM 88, octobre 1988, Paris

[LEV82] "*Special-purpose computer for logic simulation*"
Y. Levendel, P. Menon, S. Patel
The BELL system technical journal, décembre 1982

[LEW86] "*A high performance hardware accelerator for circuit level simulation of VLSI circuits*"
D.M. Lewis
publication CH2353, IEEE 1986

[LUC87] "*LL3T*"
User Guide
APTOR

[MAZ87] "*A VLSI asynchronous cellular array to accelerate logical simulation*"
G. Mazaré, R. Cornu-Emieux, P. Objois
30th Midwest International Symposium on Circuits and Systems, Syracuse (NY), Août 1987

[MEL86] "*Algebraic and combinatorial aspects of systolic algorithms for some linear problems*"
L. Melkemi, M. Tchuente
North Holland 1986, Procedings of the International Workshop on Parallel Algorithms Architectures, Luminy, 1986

[MIL87] "*Put the pedal to the metal with simulation accelerators*"
B. Milne
Electronic Design, septembre 1987

[MOT85] *"The utility of hardware accelerators in the Design Environment"*
G. Mott, R. Hall
VLSI Design, octobre 1985

[NOR87] *"Modeling Circuits in the MARS Hardware Accelerator"*
P. Agrawal, L.W. Noronha
24th ACM/IEEE Design Automation Conference

[OBJ87a] *"Simulation fonctionnelle d'une architecture parallèle en OCCAM"*
P. Objois, Y. Ansade, R. Cornu-Emieux, D. Lattard, G. Mazaré
7th OCCAM User Group Meeting & International workshop on Parallel programing of transputer based machines, Grenoble, septembre 1987

[OBJ87b] *"Highly parallel logic simulation accelerators based upon distributed discrete-event simulation"*
P. Objois, Y. Ansade, R. Cornu-Emieux, G. Mazaré
International Workshop on Hardware Accelerators, Oxford, octobre 1987

[OBJ88] P. Objois
Thèse INPG de microélectronique

[PAY88] Rapport de DEA d'informatique
E. Payant
juin 1988

[SAS83] *"HAL : a block level hardware logic simulator"*
T. Sasaki et all
Procedings of the 20th Design Automation Conference, juin 1983

Références bibliographiques

[SHE87] "*The analysis of simulation acceleration concepts and the associated trade-offs in the CAE environment* "
C. Sheldon
International Workshop on Hardware Accelerators, Oxford, octobre 1987

[SIA85] "*Optimisation du placement de blocs par la méthode thermodynamique : application à la conception du plan de masse d'un circuit* "
P. Siarry, L. Bergonzi, G. Dreyfus
1er colloque national sur les circuits à la demande, Grenoble 1985

[SPI87] "*A high performance routing engine* "
T.D. Spiers, D.A. Edwards
24th ACM/IEEE Design Automation Conference

[TAK86] "*HAL II : a mixed level hardware logic simulation system* "
S. Takasaki, T. Sasaki and all
Procedings of the 23rd Design Automation Conference, 1986

[TRA86] "*Utilisation des accélérateurs matériels pour la simulation logique en IAO/CAO* "
P. Traynar
Electronique Industrielle, N°110, juin 1986

[UED83] "*A parallel processing approach for logic Module placement* "
K. Ueda, T. Komatsubara, T. Hosaka
IEEE Transactions on computer aided design, vol 1, janvier 1983

[UNG58] "*A computer oriented toward spatial problems* "
S. Unger
Procedings of the IRE, octobre 1958

[VLA87] *"A vector hardware accelerator with circuit simulation emphasis"*
A. Vladimirescu et all
24th ACM/IEEE Design automation conference

[WAT86] *"A new routing algorithm and its hardware implementation"*
T. Watanabe, Y. Sugiyama
Procedings of the 23rd Design Automation Conference, 1986

[WON87] *"A hardware accelerator for maze routing"*
Y. Won, S. Sahni, Y. El-Ziq
24th ACM/IEEE Design Automation Conference

[WOO87] *"Architecture of a general accelerator for CAD : the proteus 1"*
S.P. Smith, B. Wood
Procedings of the International Workshop on Hardware Accelerators, Oxford, octobre 1987

[ZYC84] *"LE1000 reference manual"*
Zycad
1984

ANNEXES

Annexe 1

La bibliothèque de cellules

Pour réaliser le circuit cellulaire, nous avons développé en technologie CMOS 1 métal 2 microns (CNET), une bibliothèque de cellules élémentaires . Cette bibliothèque est simple et ne comprend qu'une vingtaine de cellules de base :

un inverseur,
un amplificateur,
divers interrupteurs - portes de transferts -,
des NANDs à 2, 3 et 4 entrées,
des NORs à 2 et 3 entrées,
divers multiplexeurs (2→1),
un registre simple (latch) et un registre maître/esclave,
une porte-3-états,
un OU exclusif à 2 entrées.

Nous présentons dans cette annexe les caractéristiques électriques de ces composants. Nous les avons simulés en mettant à chaque fois, 4 capacités de charges différentes à leurs sorties (0.1, 0.5, 1 et 5 PF).

Ces simulations ont été réalisées à l'aide du logiciel SPICE.

Les régles de dessin et d'assemblage de ces cellules sont données au paragraphe 1.2 du chapitre 2

Schéma électrique de l'inverseur

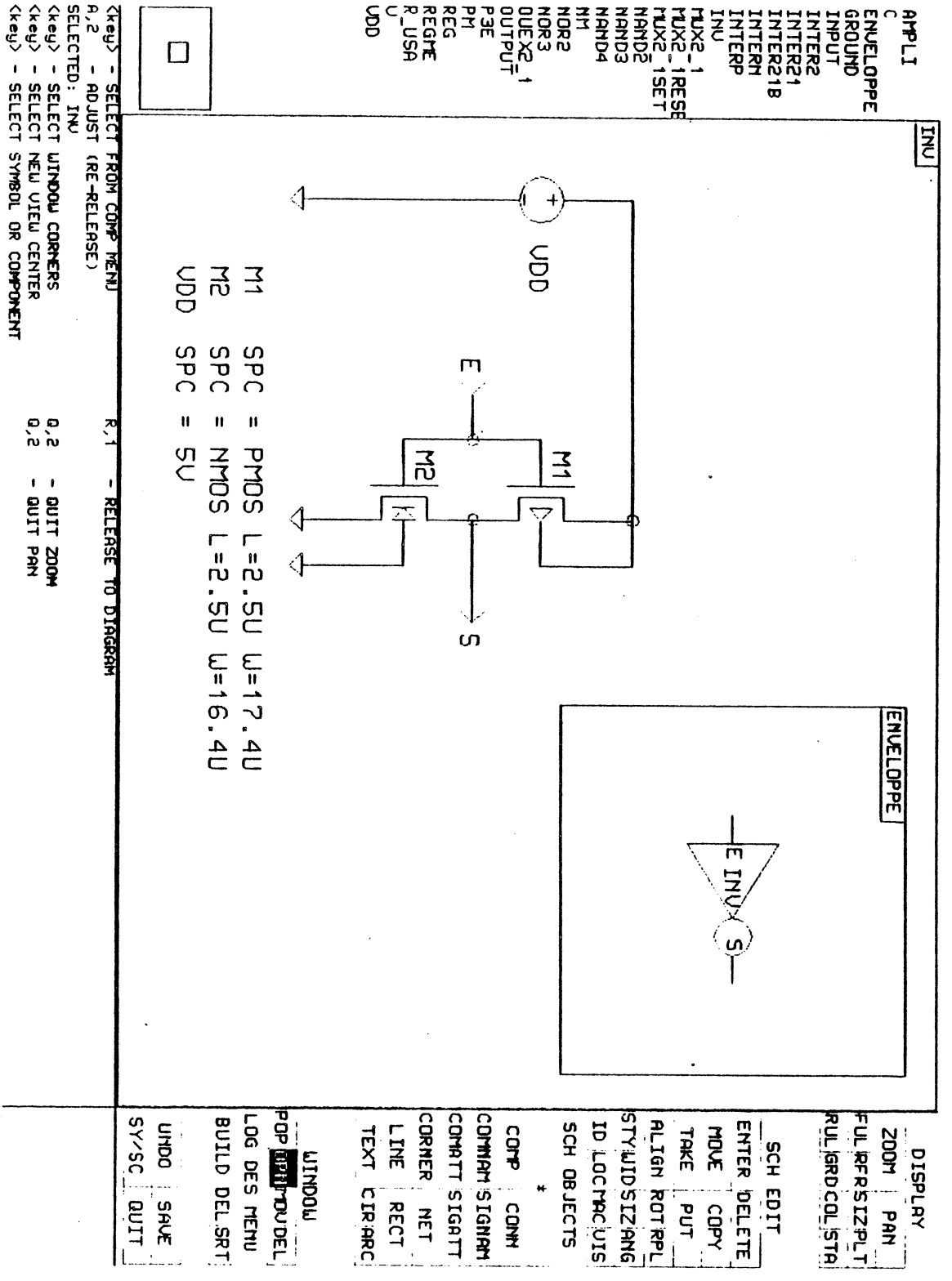
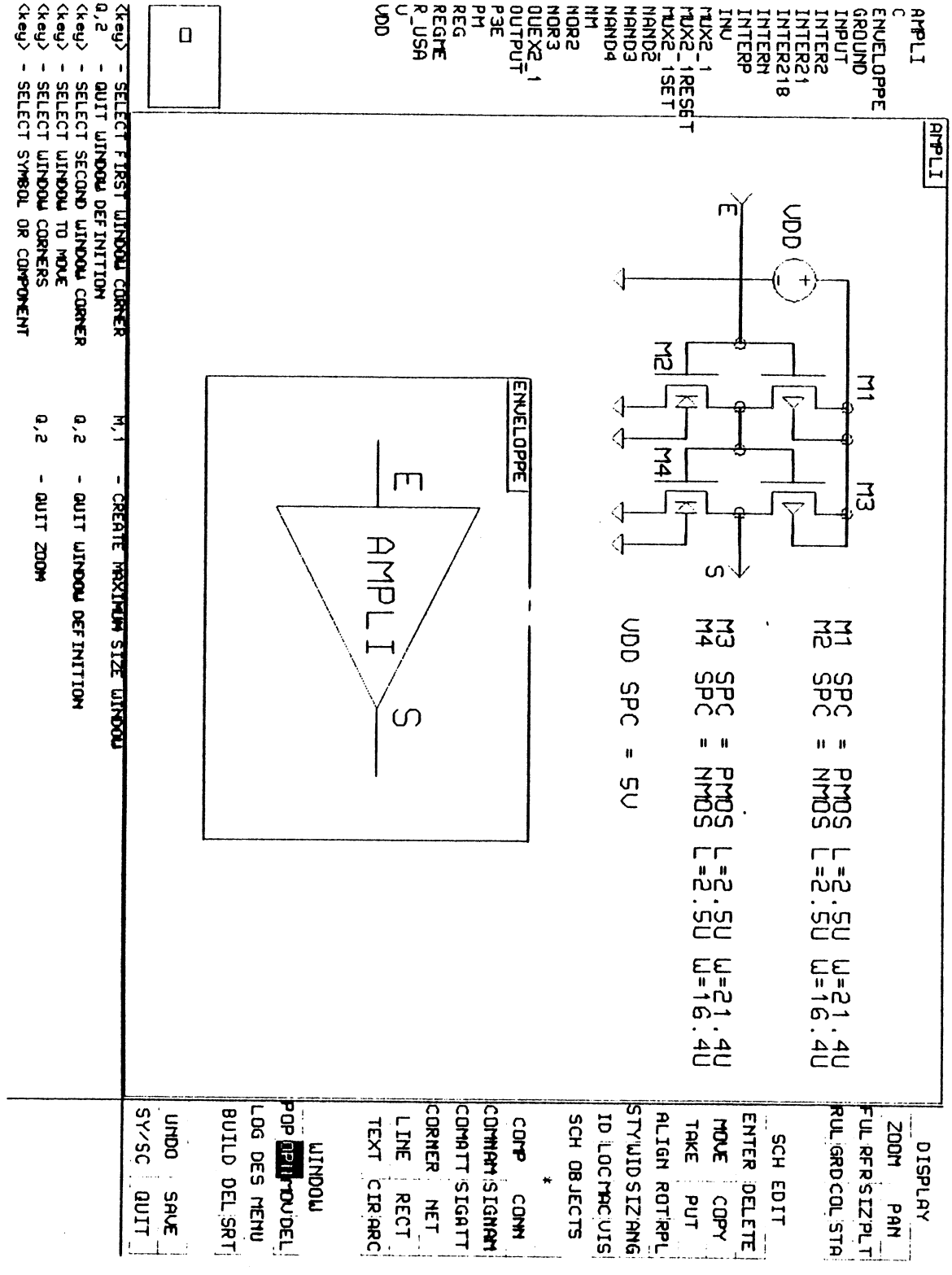


Schéma électrique de l'amplificateur



Simulation électrique de l'inverseur et de l'ampli

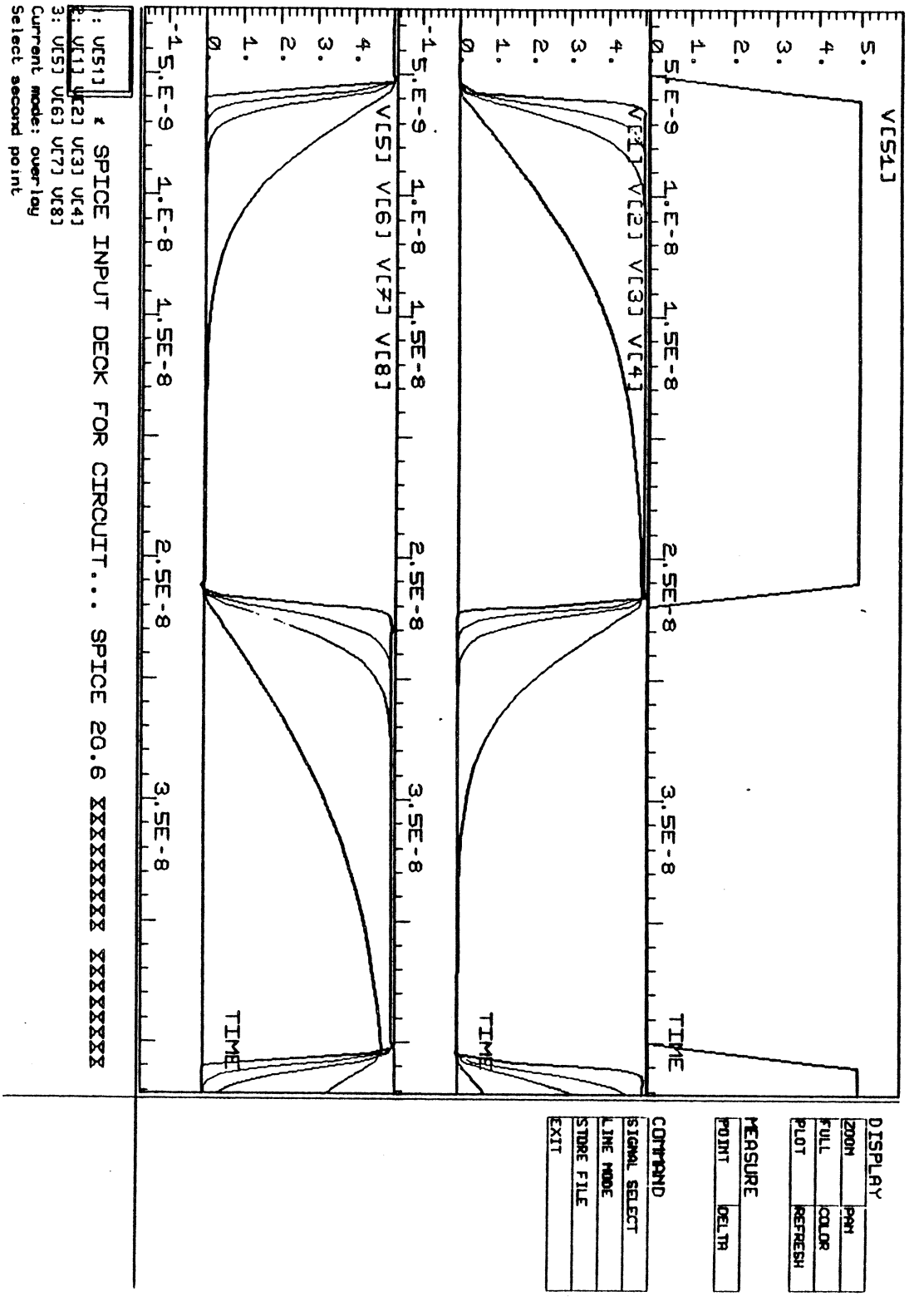
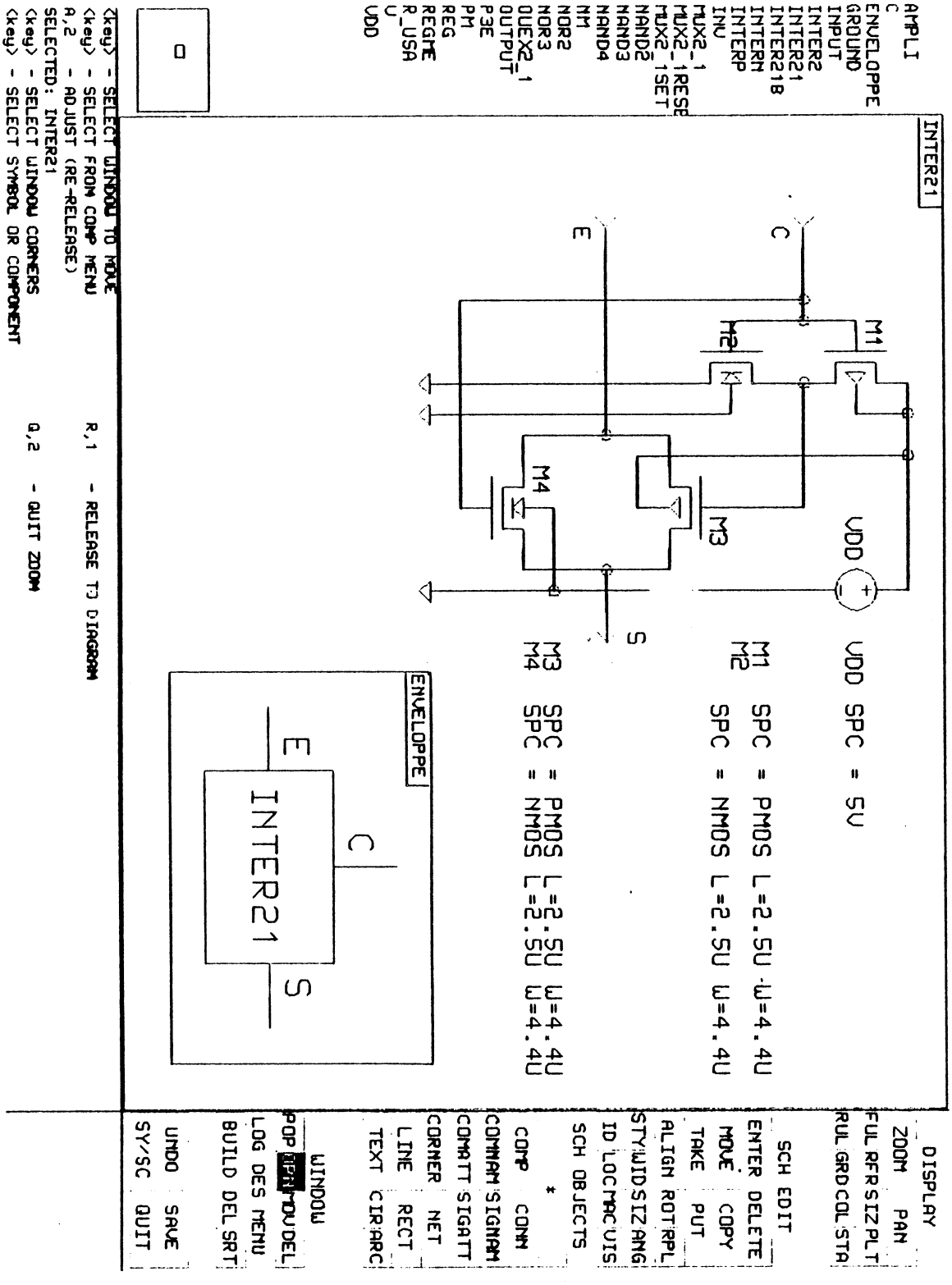


Schéma électrique d'un interrupteur - 1 commande (non inversée) -



Annexe 1 : La bibliothèque de cellules

Schéma électrique d'un interrupteur - 1 commande (inversée) -

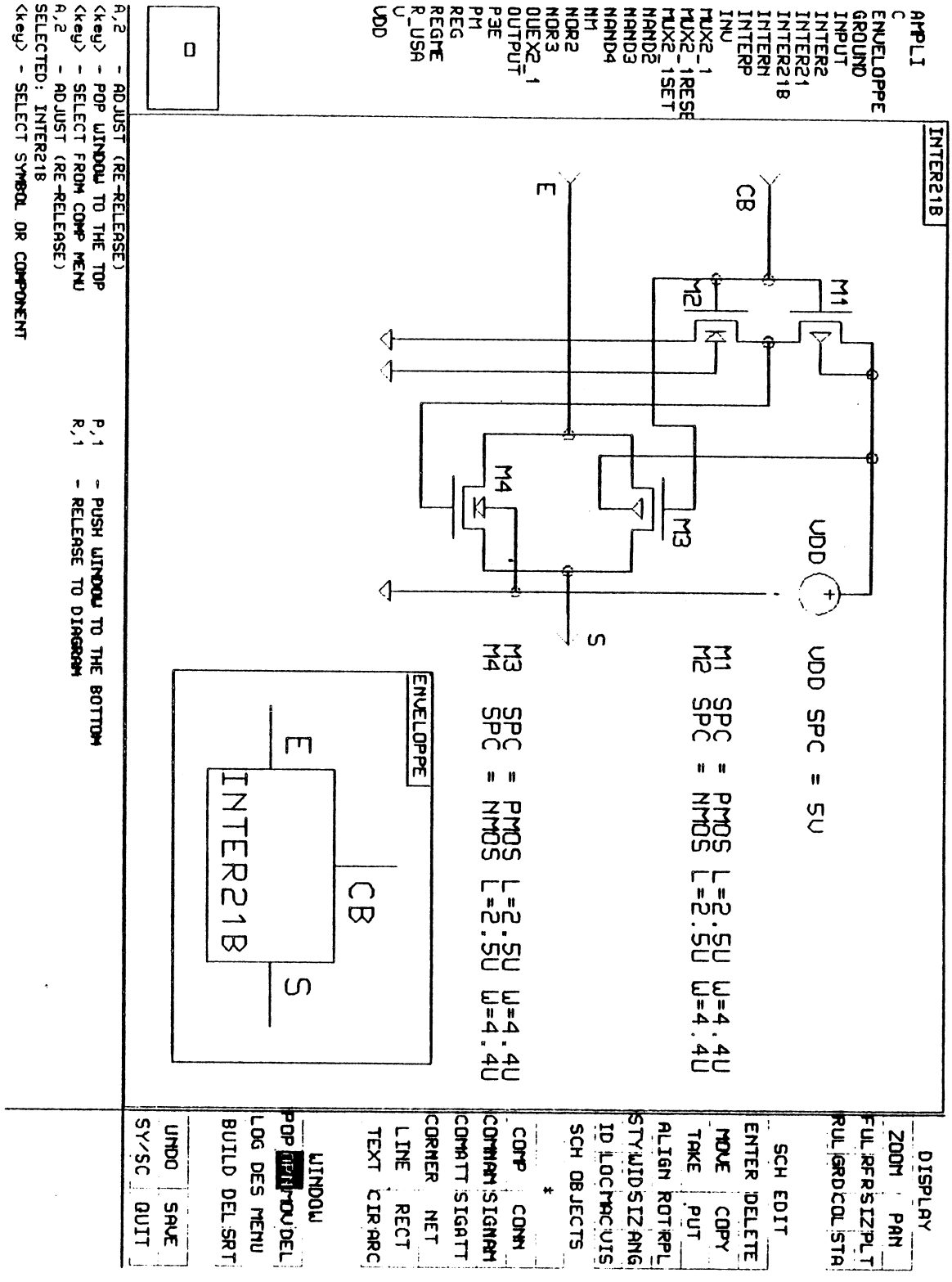


Schéma électrique d'un interrupteur - 2 commandes (complémentaires) -

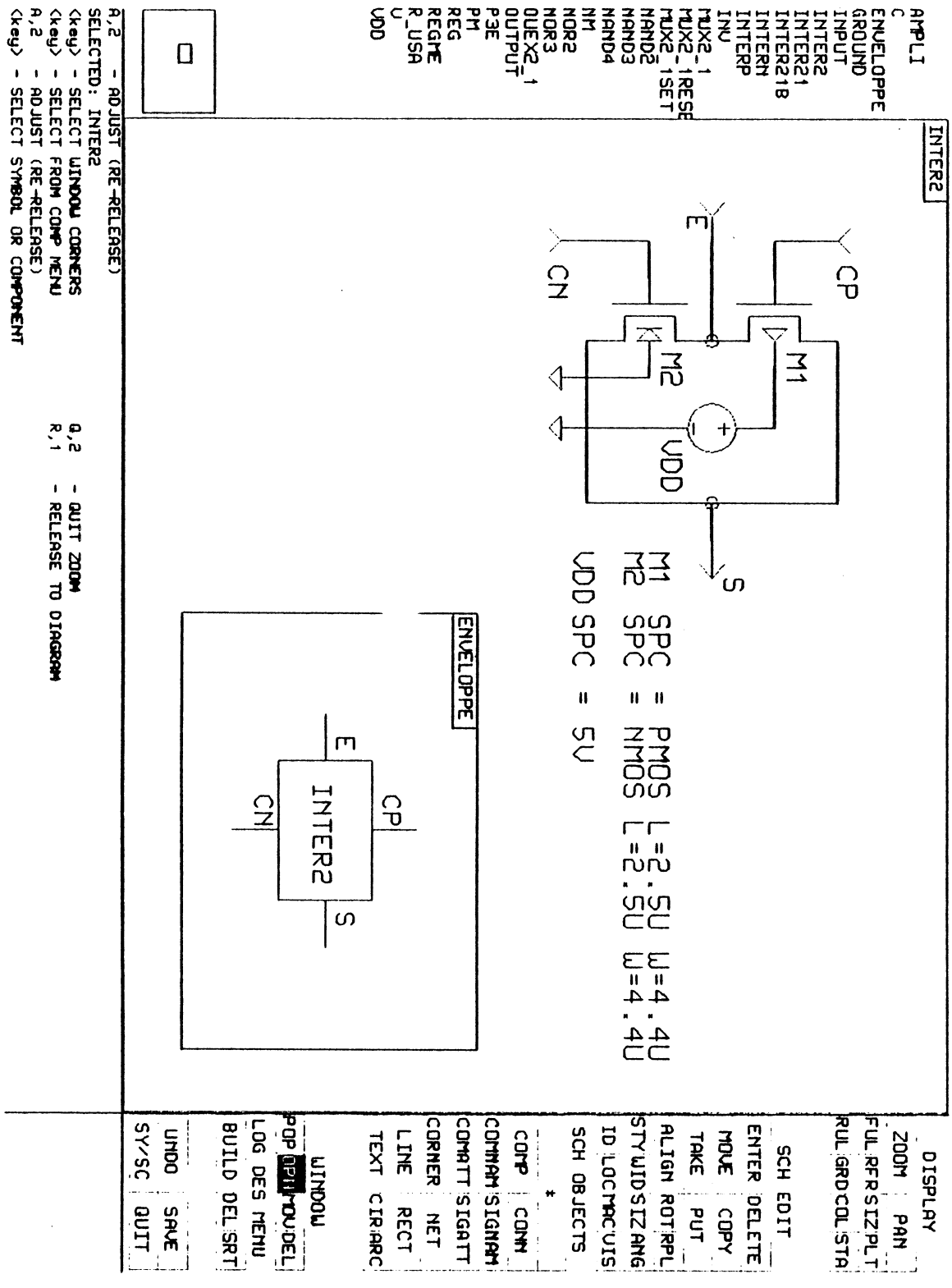
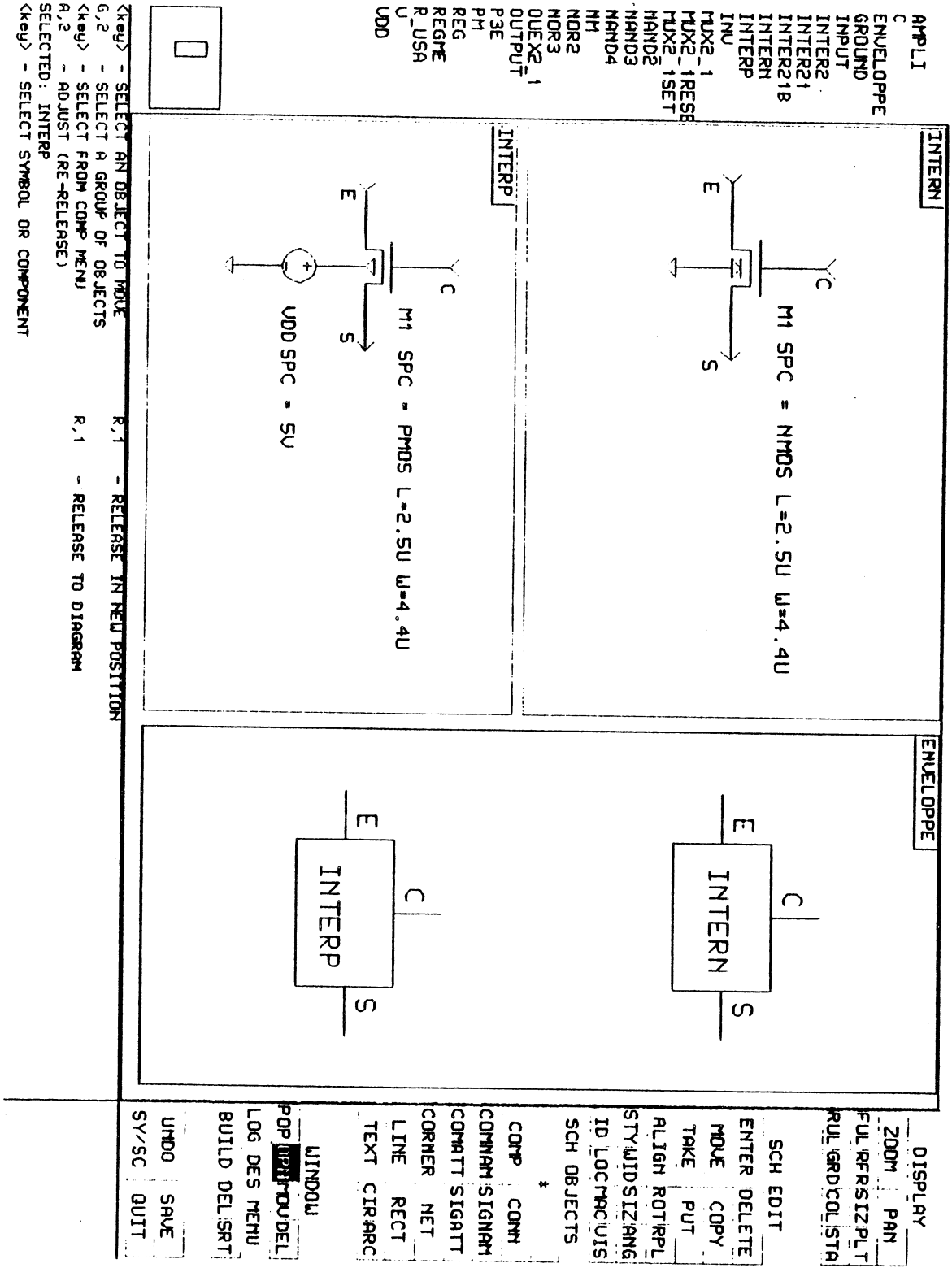


Schéma électrique d'interrupteurs simples



Simulation électrique des interrupteurs

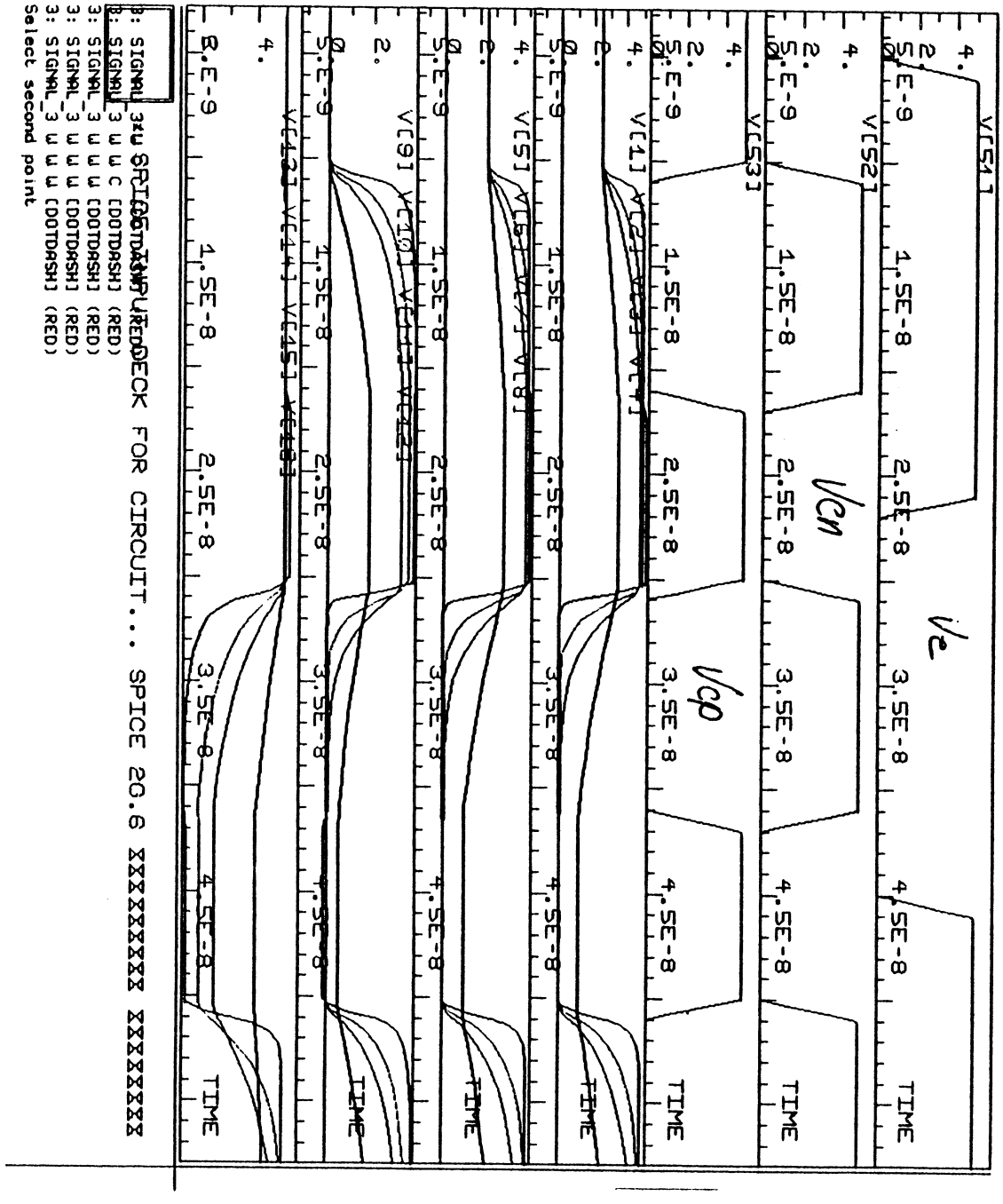


Schéma électrique du NAND 2 entrées

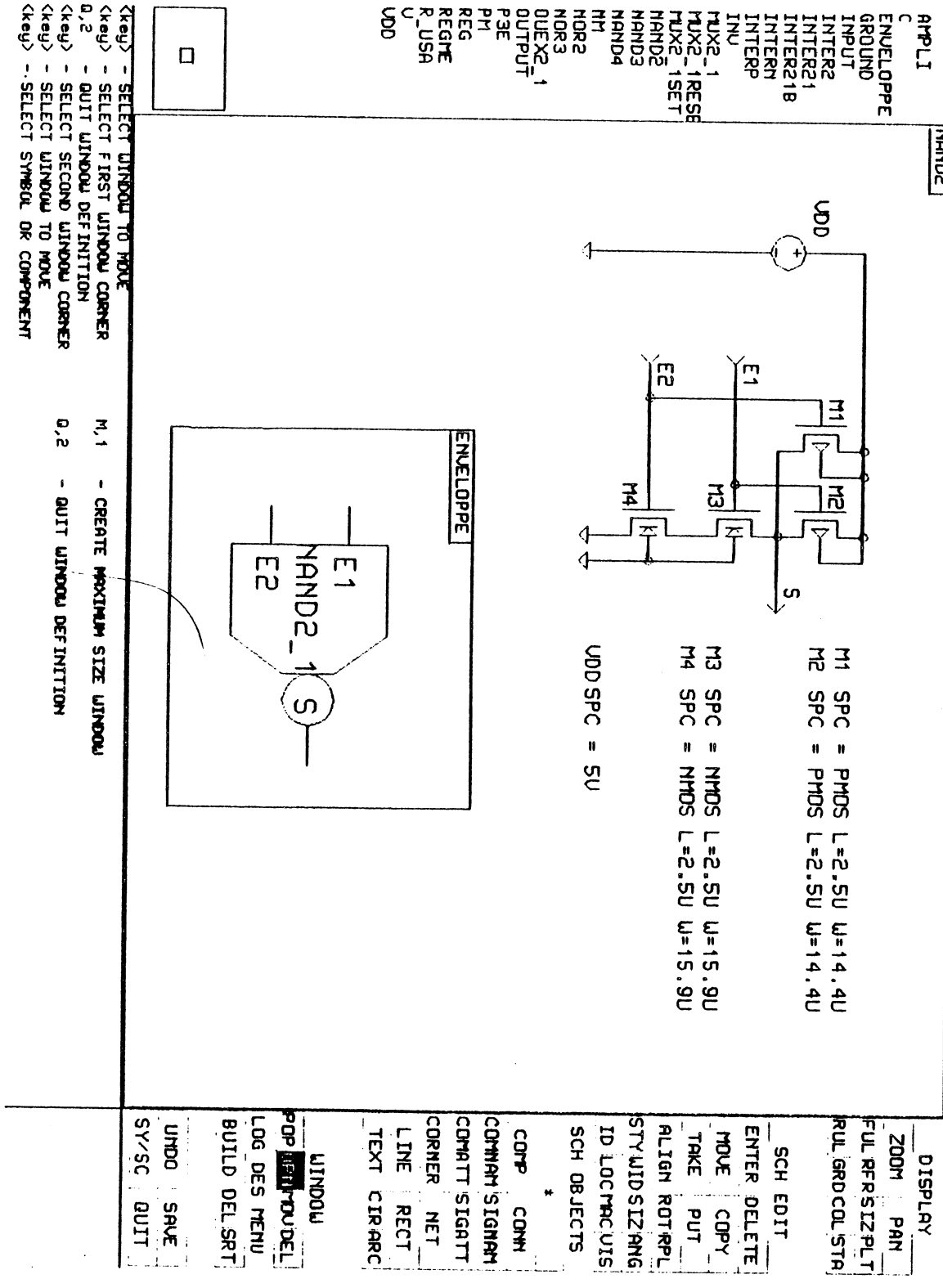


Schéma électrique du NAND 3 entrées

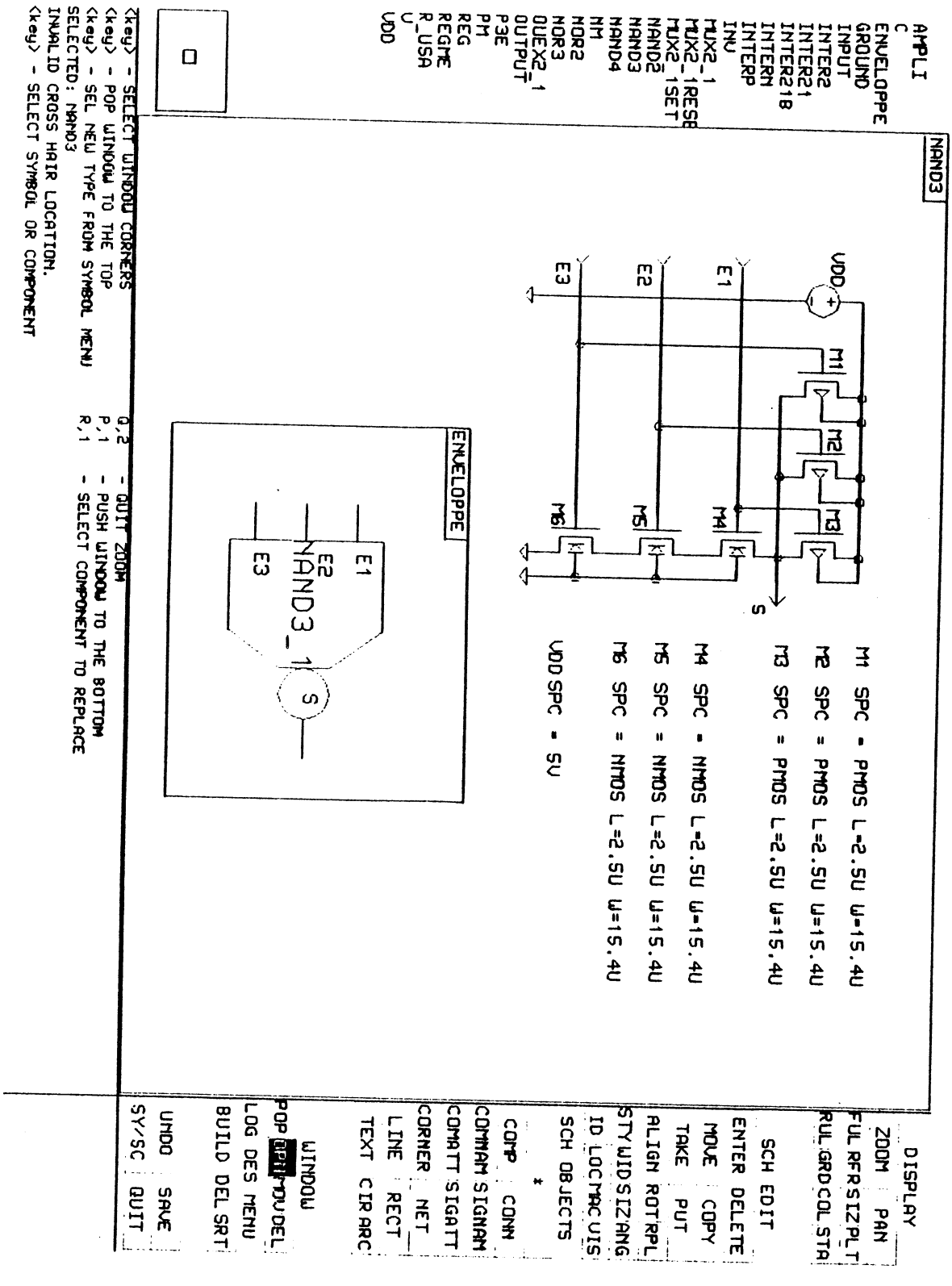
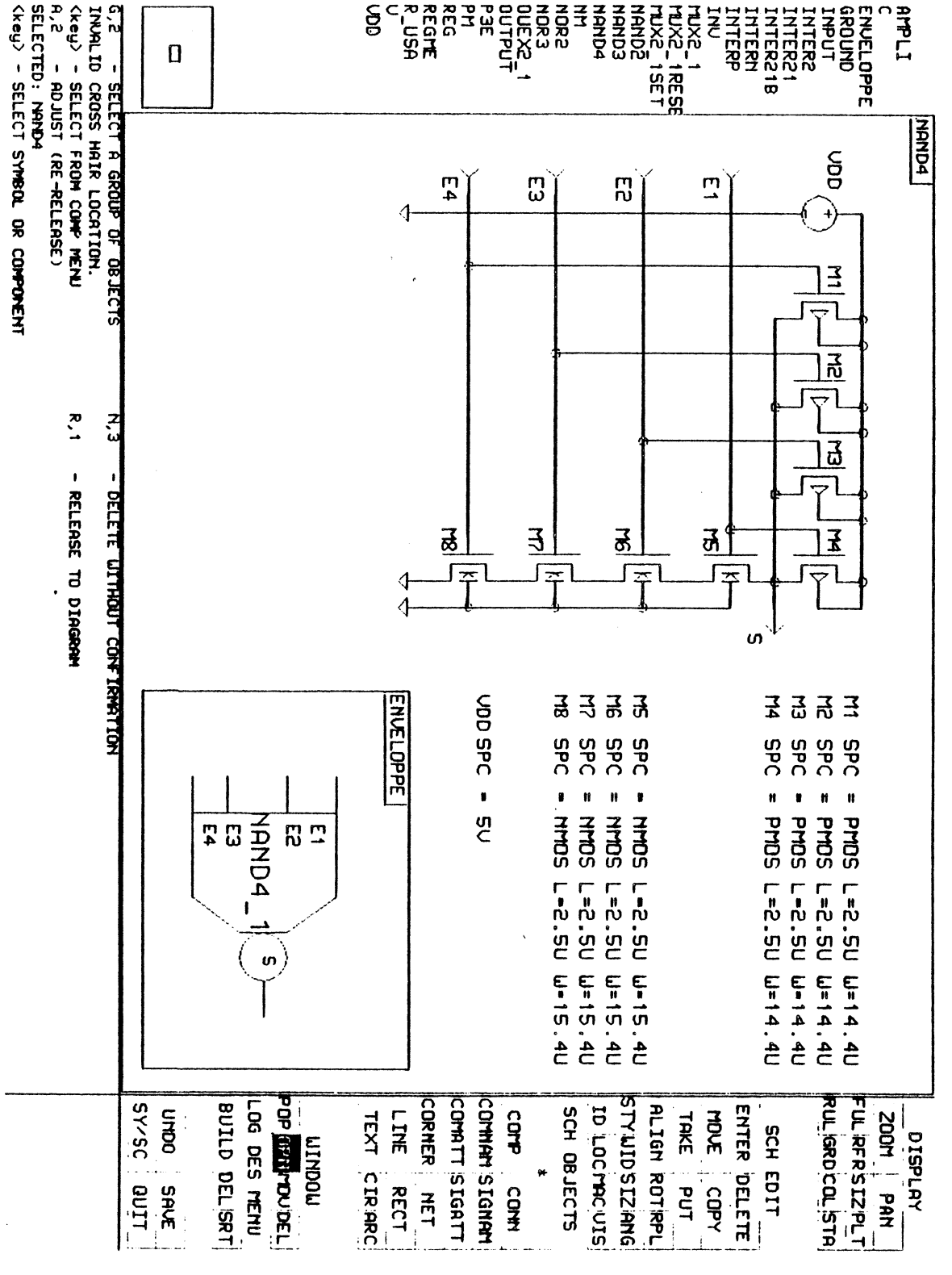
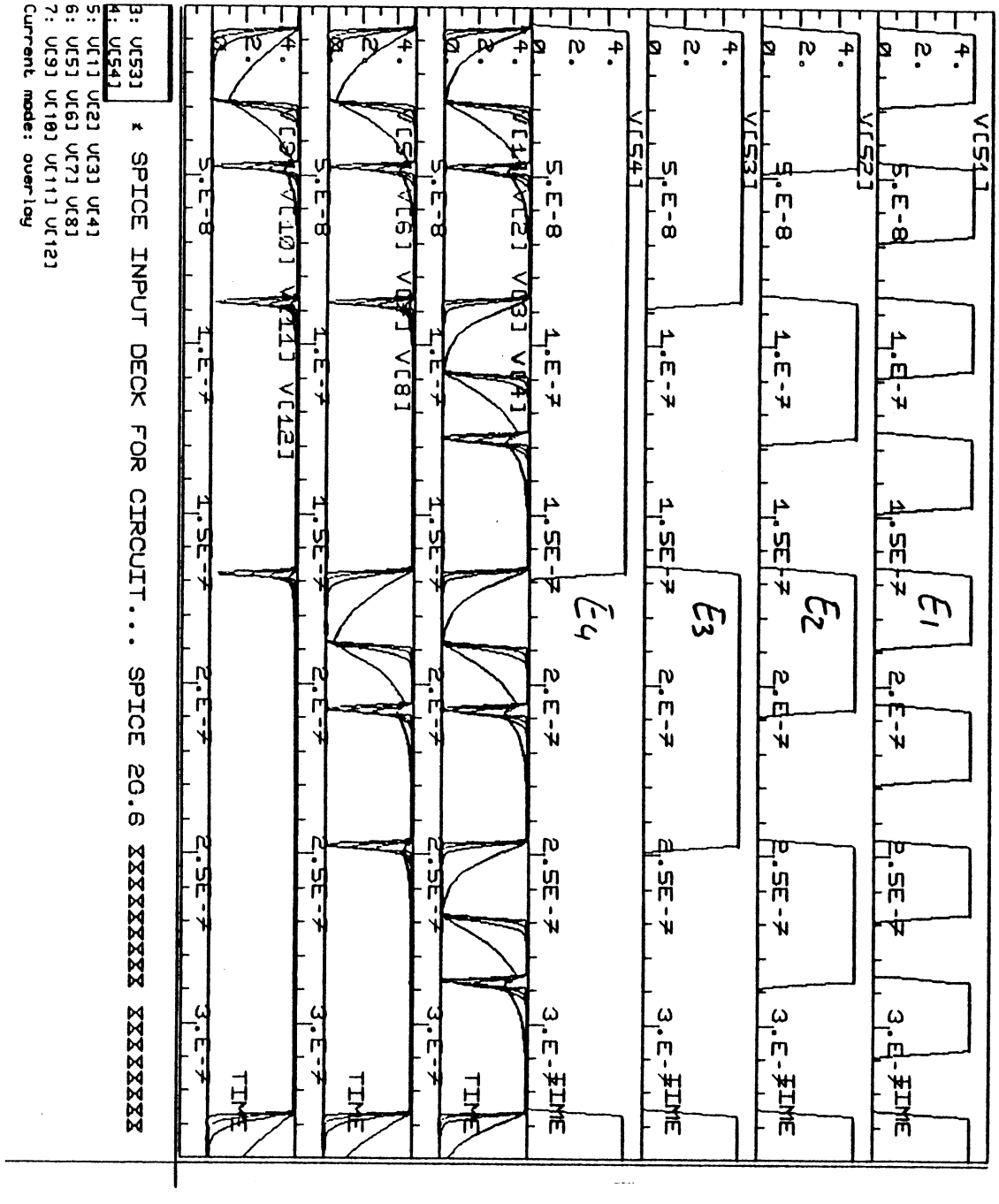


Schéma électrique du NAND 4 entrées



Simulation électrique des NANDs



Annexe 1 : La bibliothèque de cellules

Schéma électrique d'un multiplexeur (2→1)

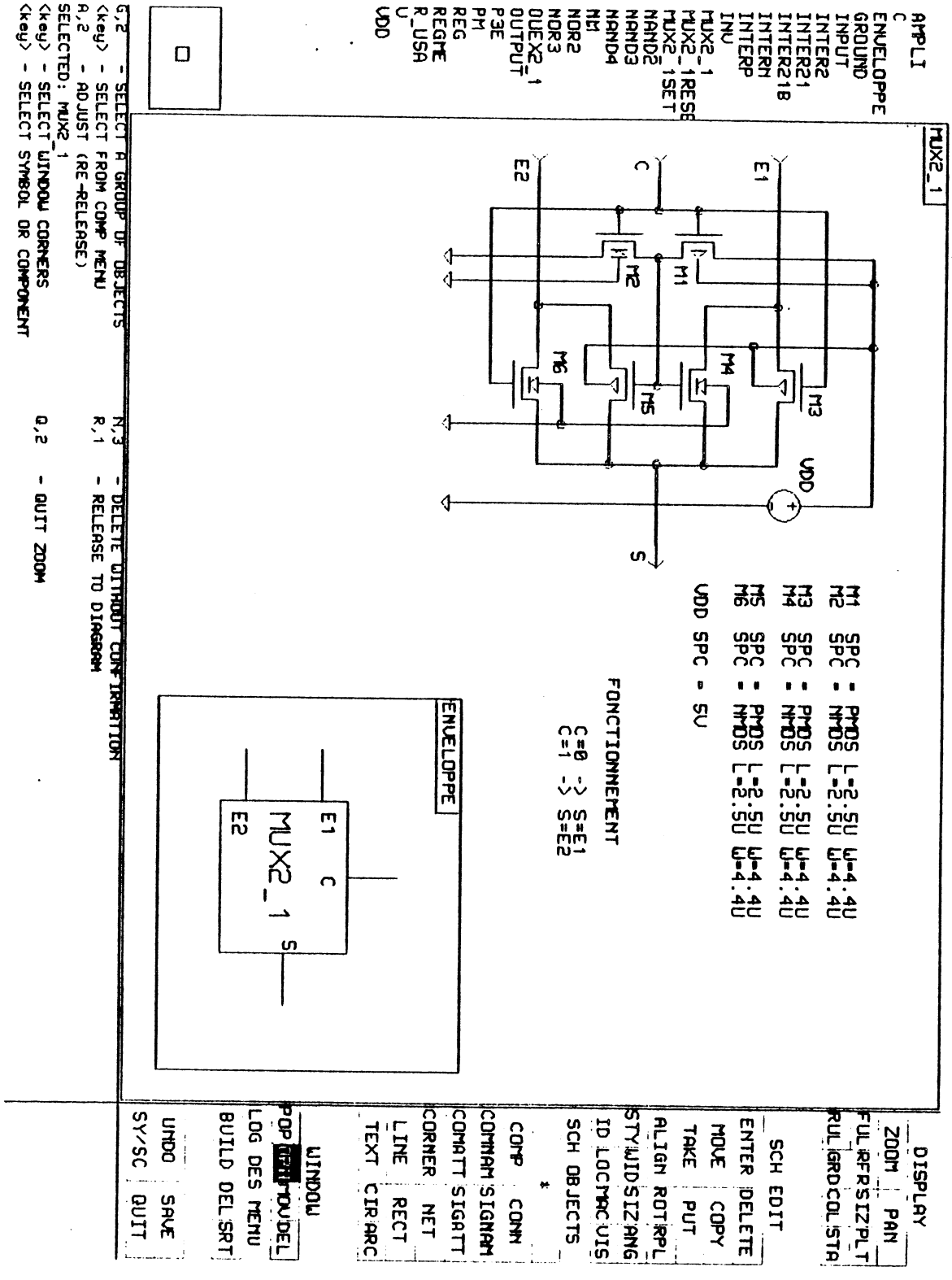


Schéma électrique d'un multiplexeur (2→1) avec mise à 1

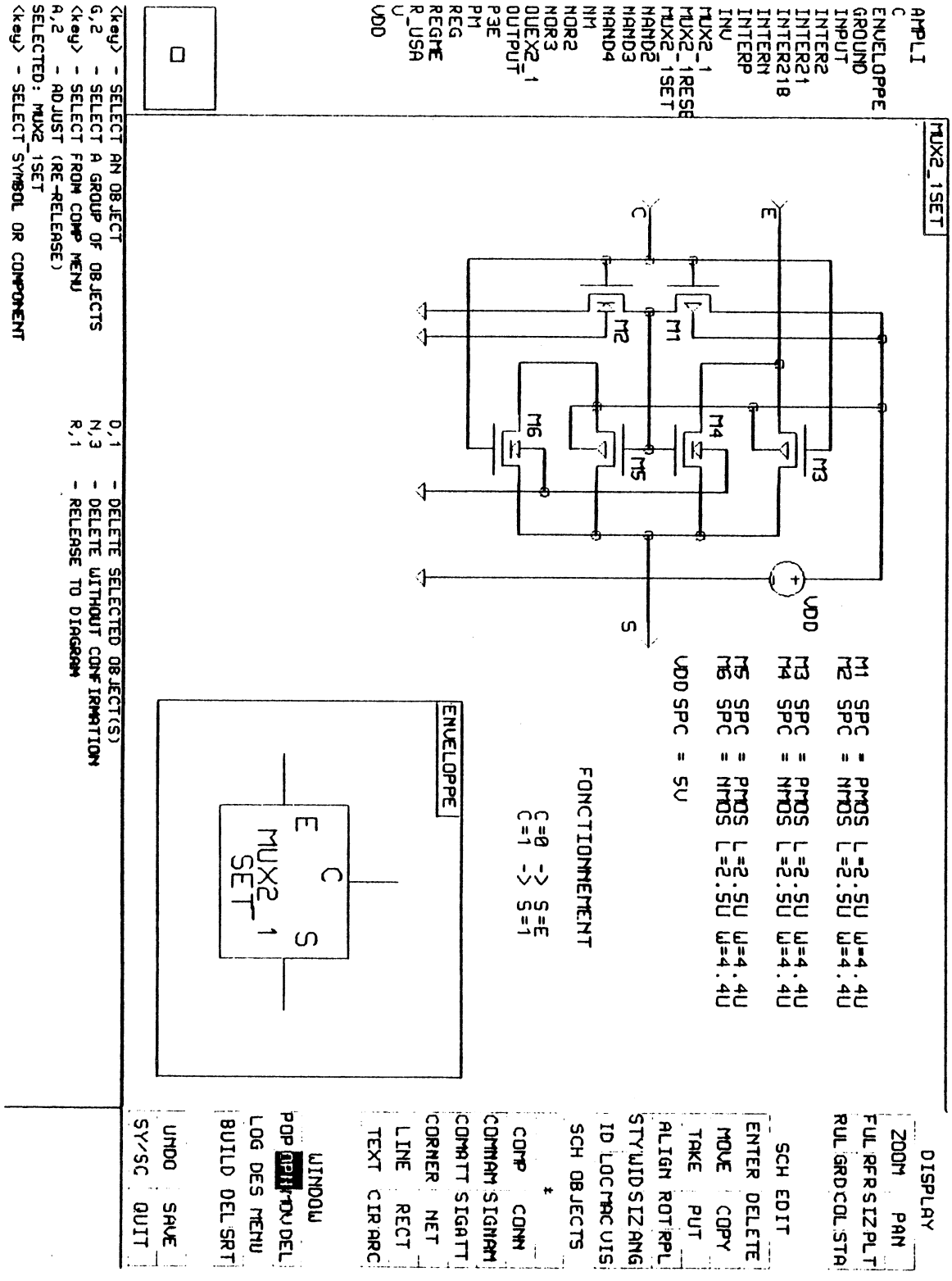
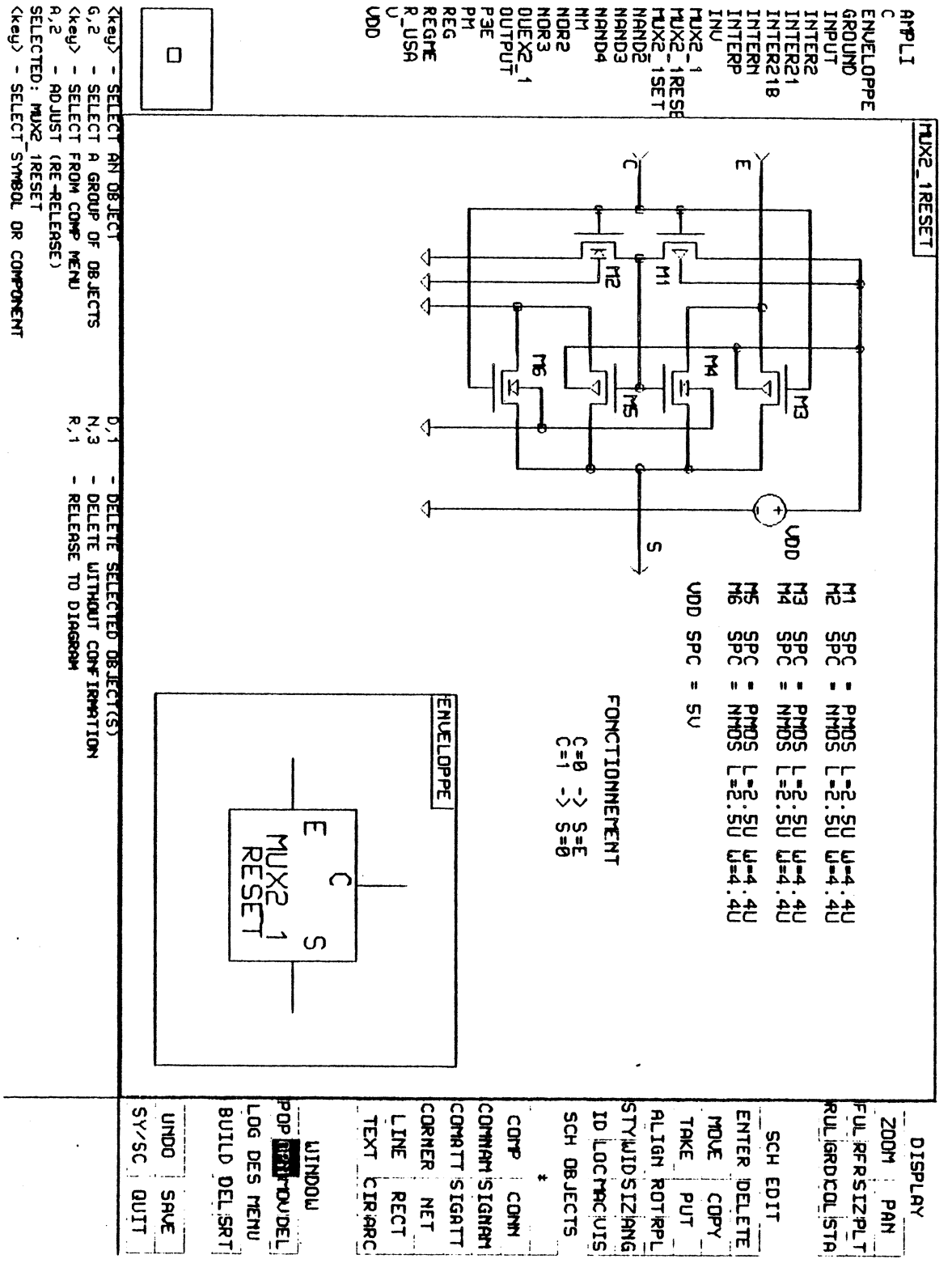


Schéma électrique d'un multiplexeur (2→1) avec mise à 0



Simulation électrique des multiplexeurs

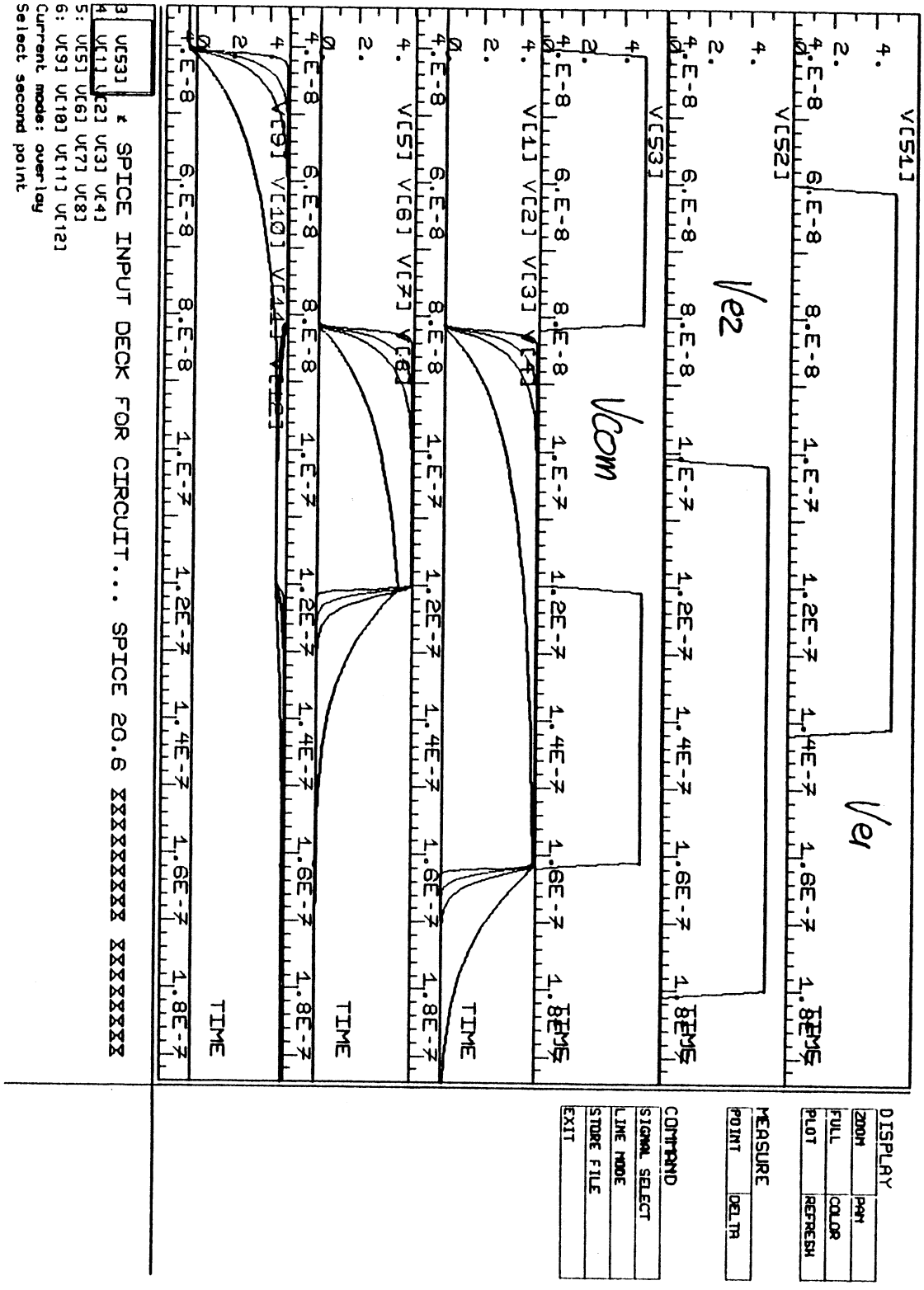


Schéma électrique du NOR 2 entrées

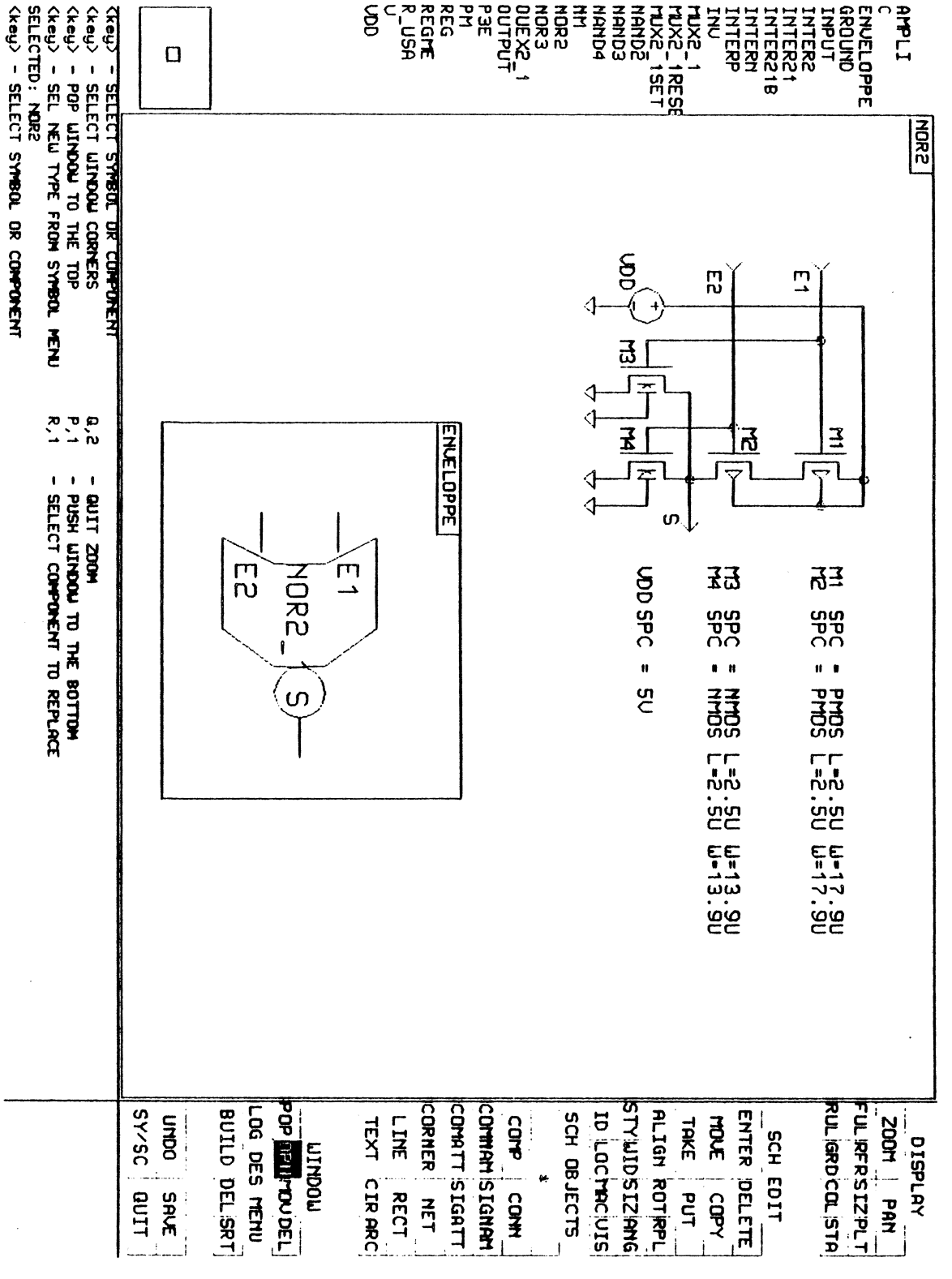
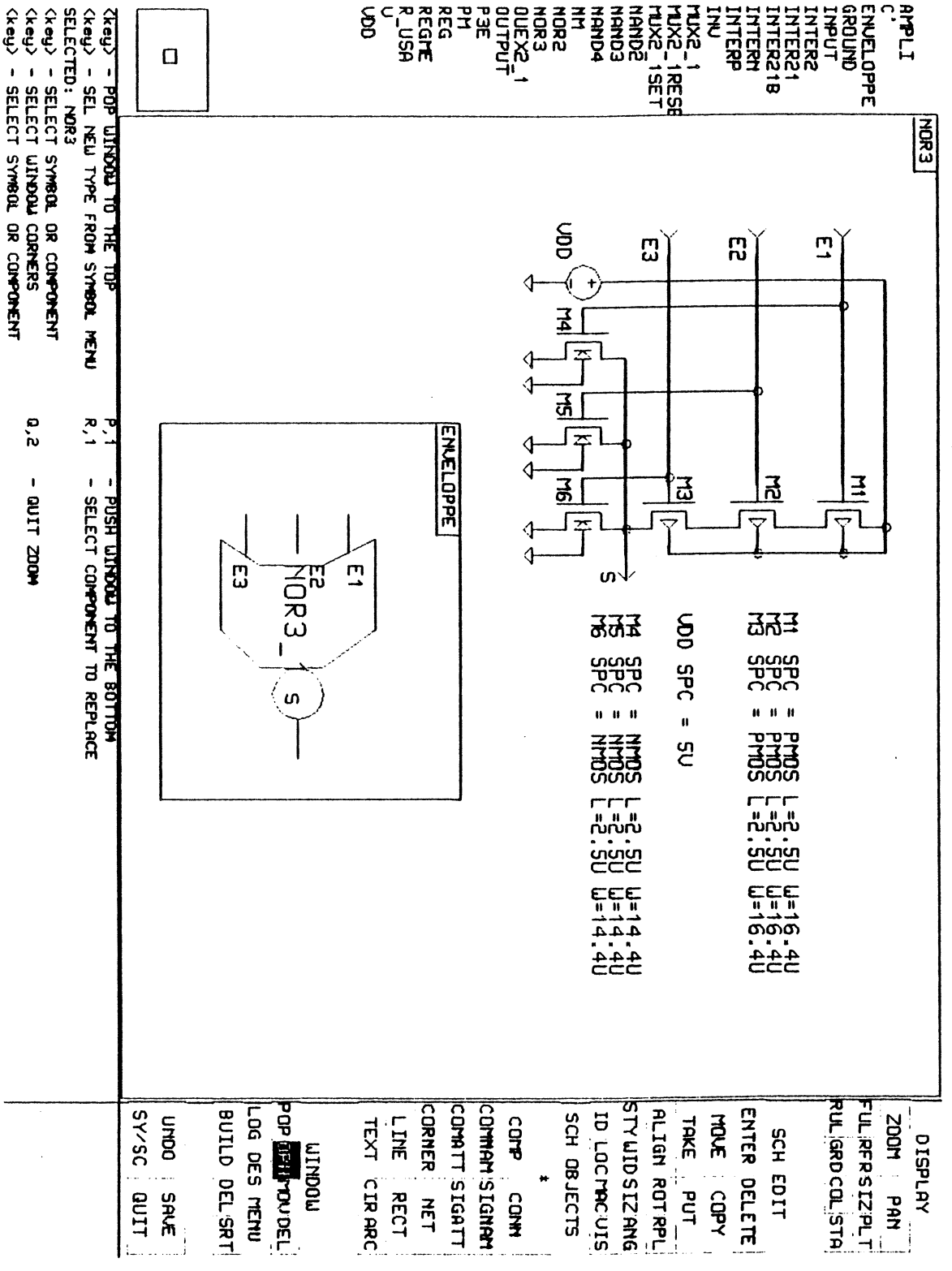


Schéma électrique du NOR 3 entrées



Simulation électrique des NORs

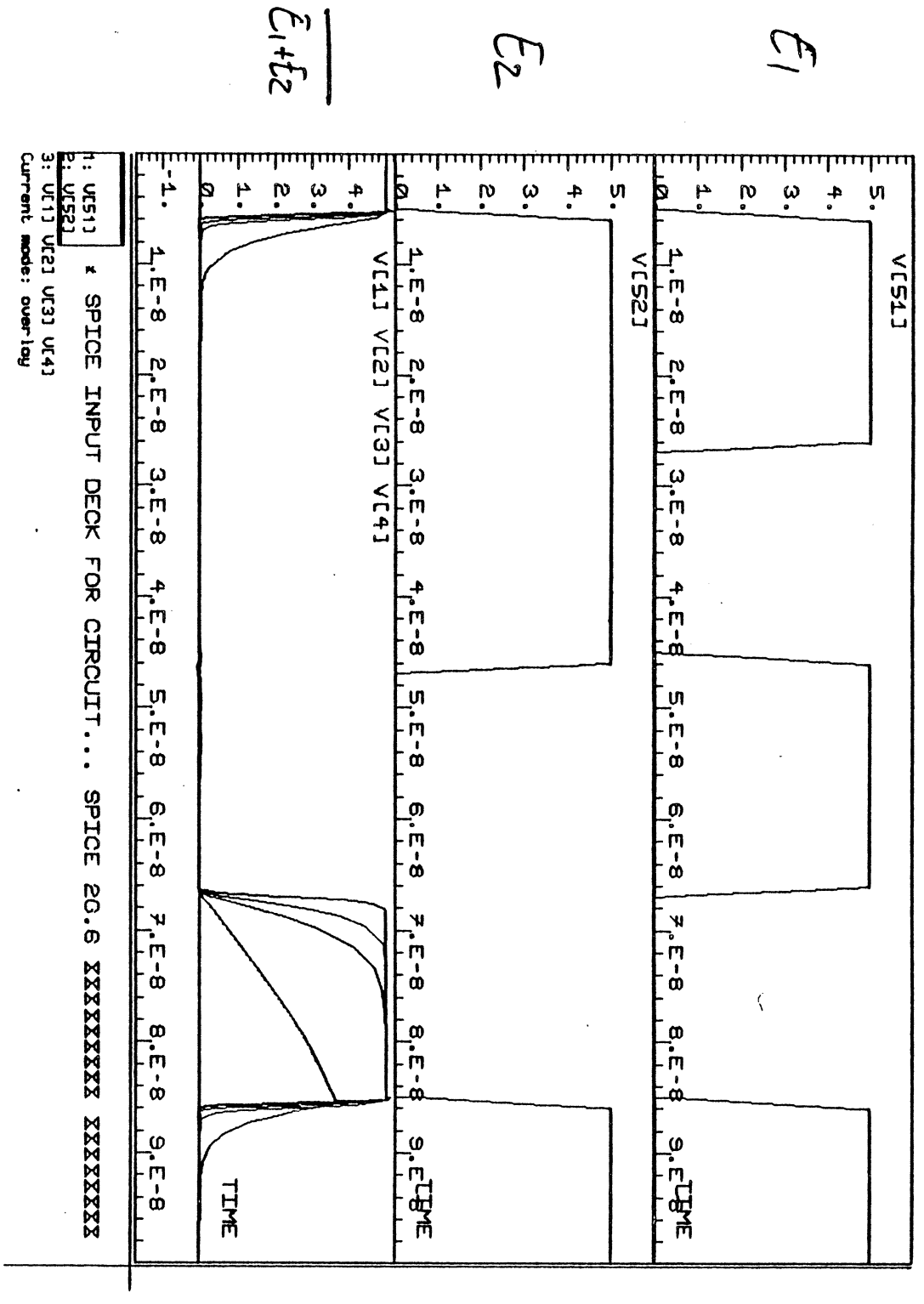


Schéma électrique du registre simple (latch)

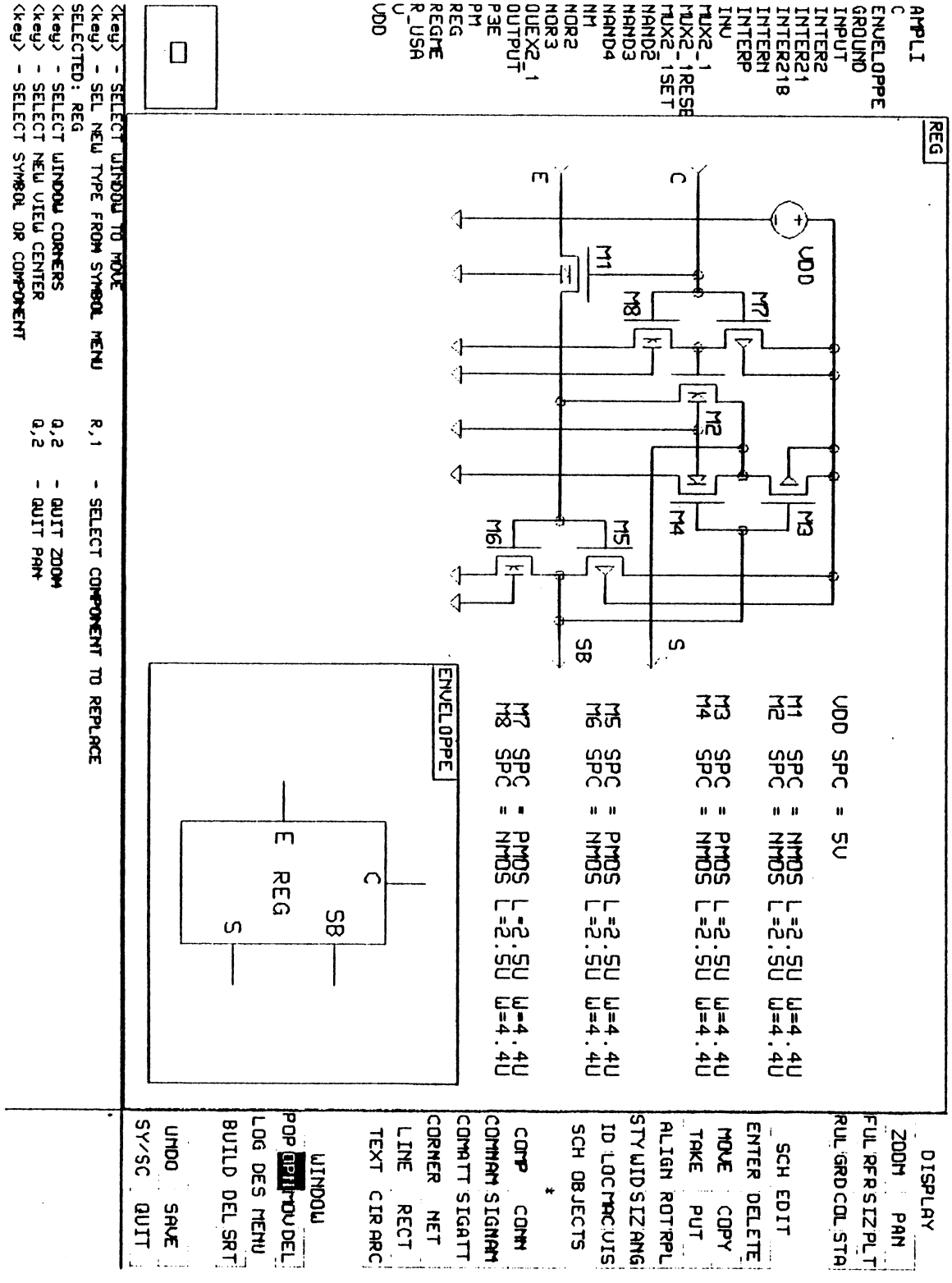
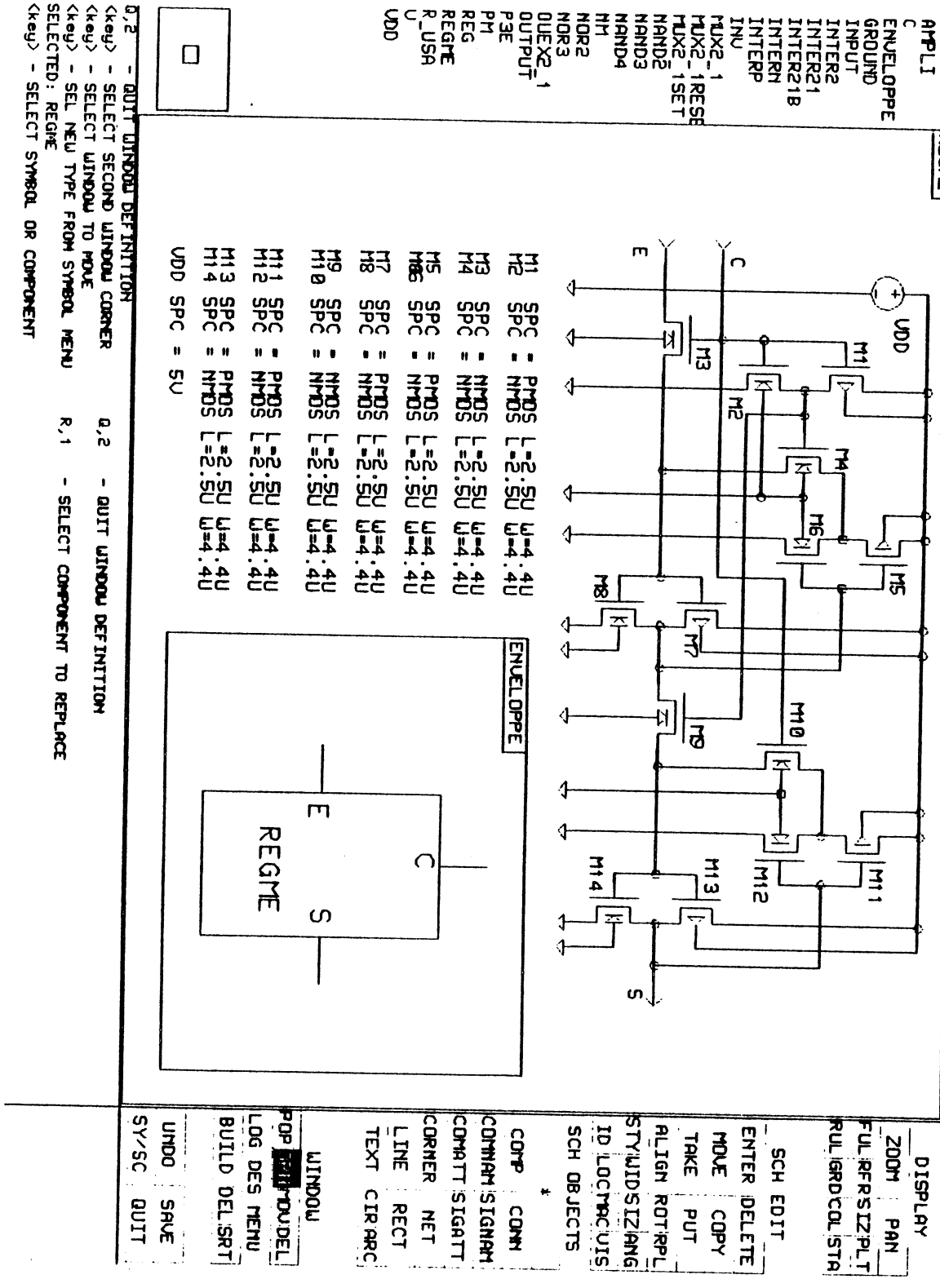


Schéma électrique du registre maître/esclave



Simulation électrique des registres

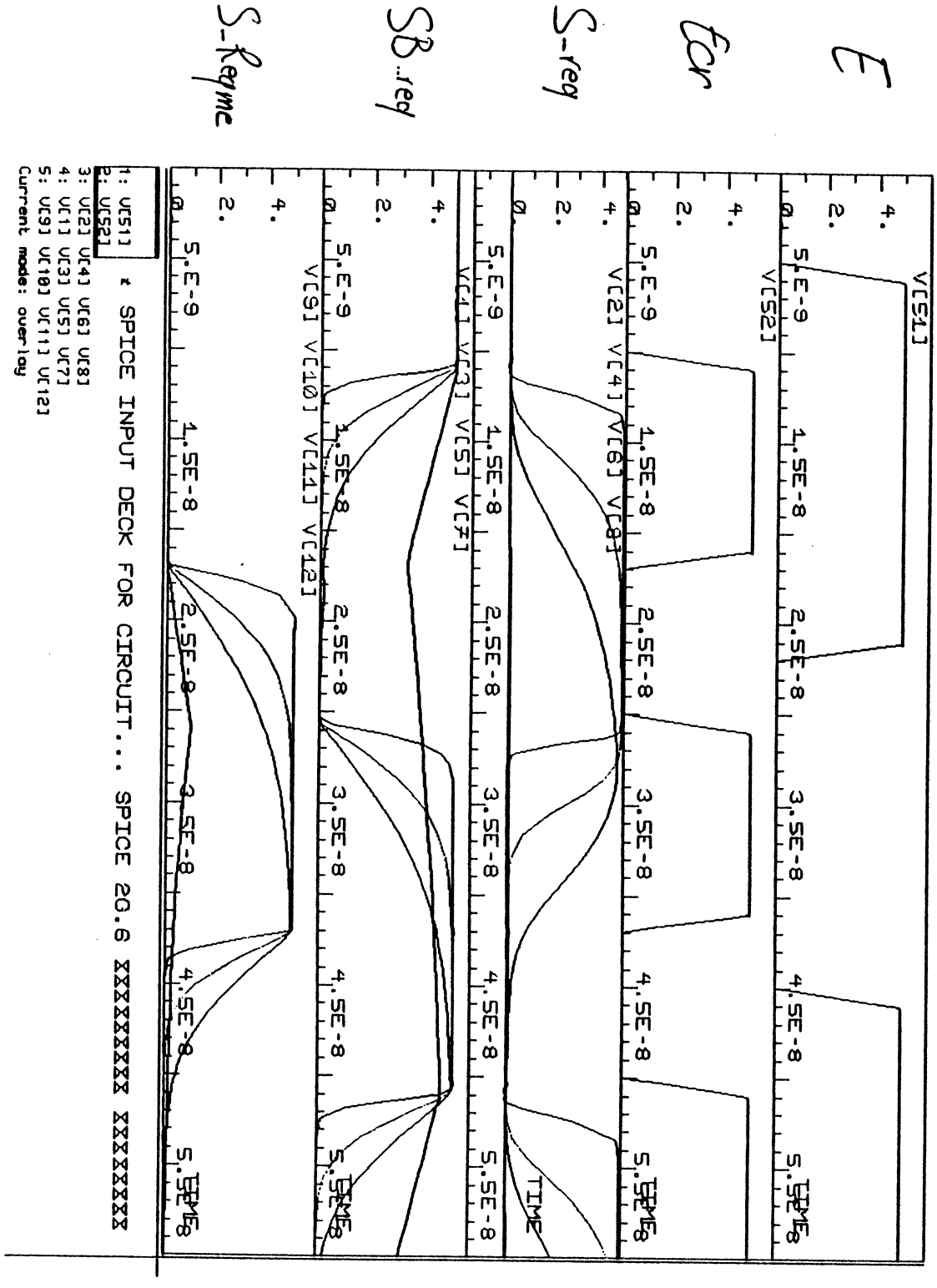
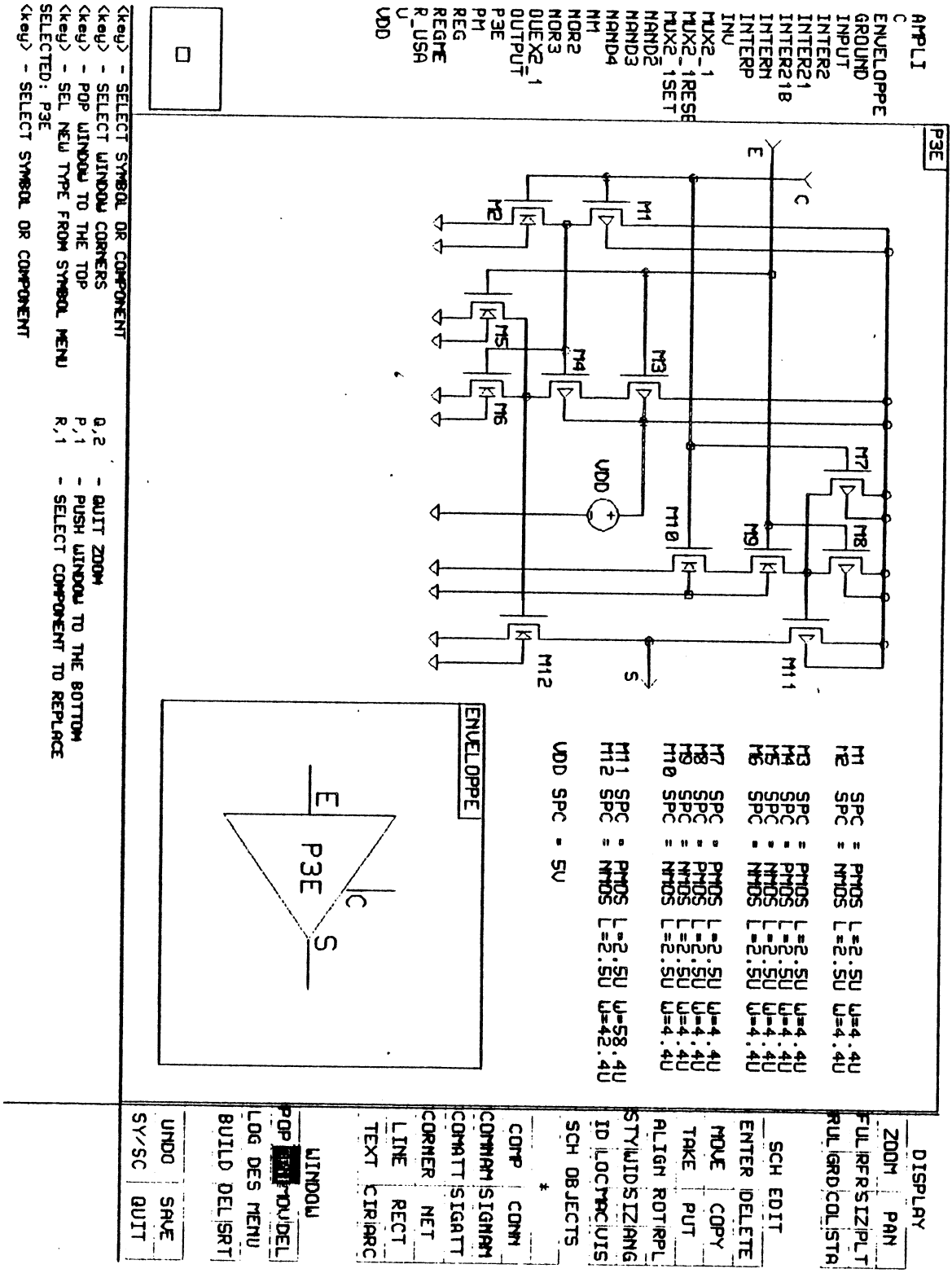


Schéma électrique de la porte-3-états



Simulation électrique de la porte-3-états

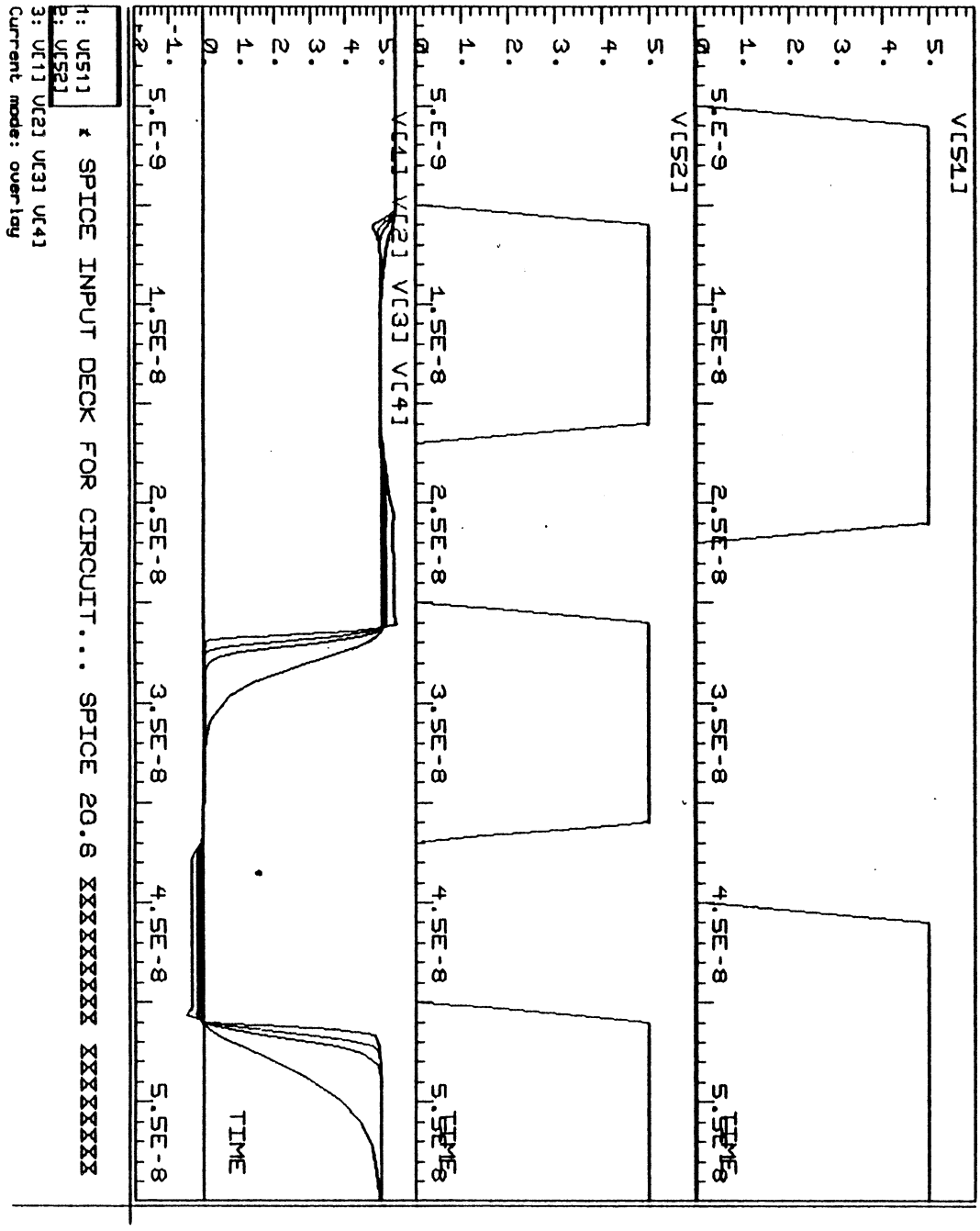
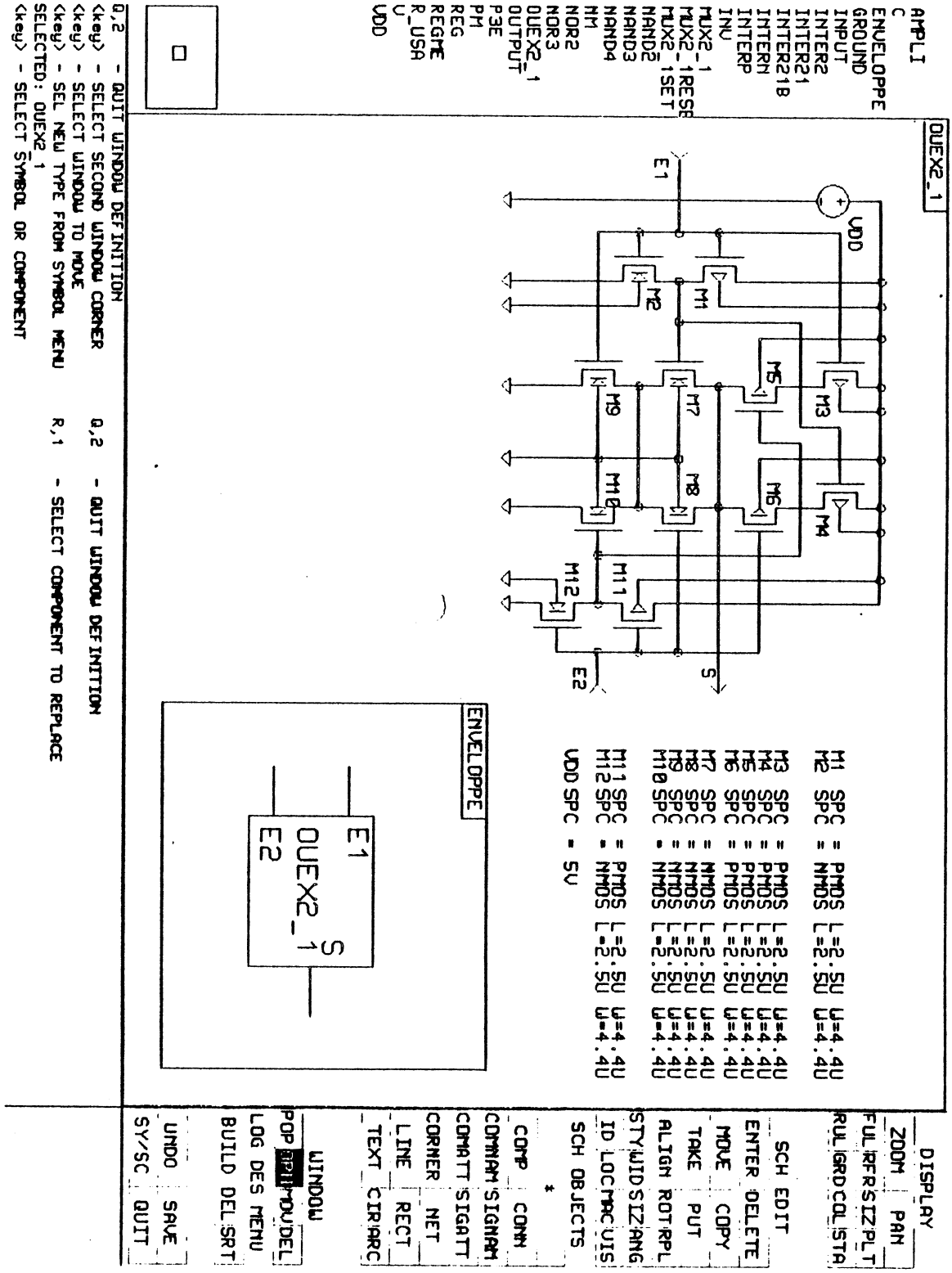
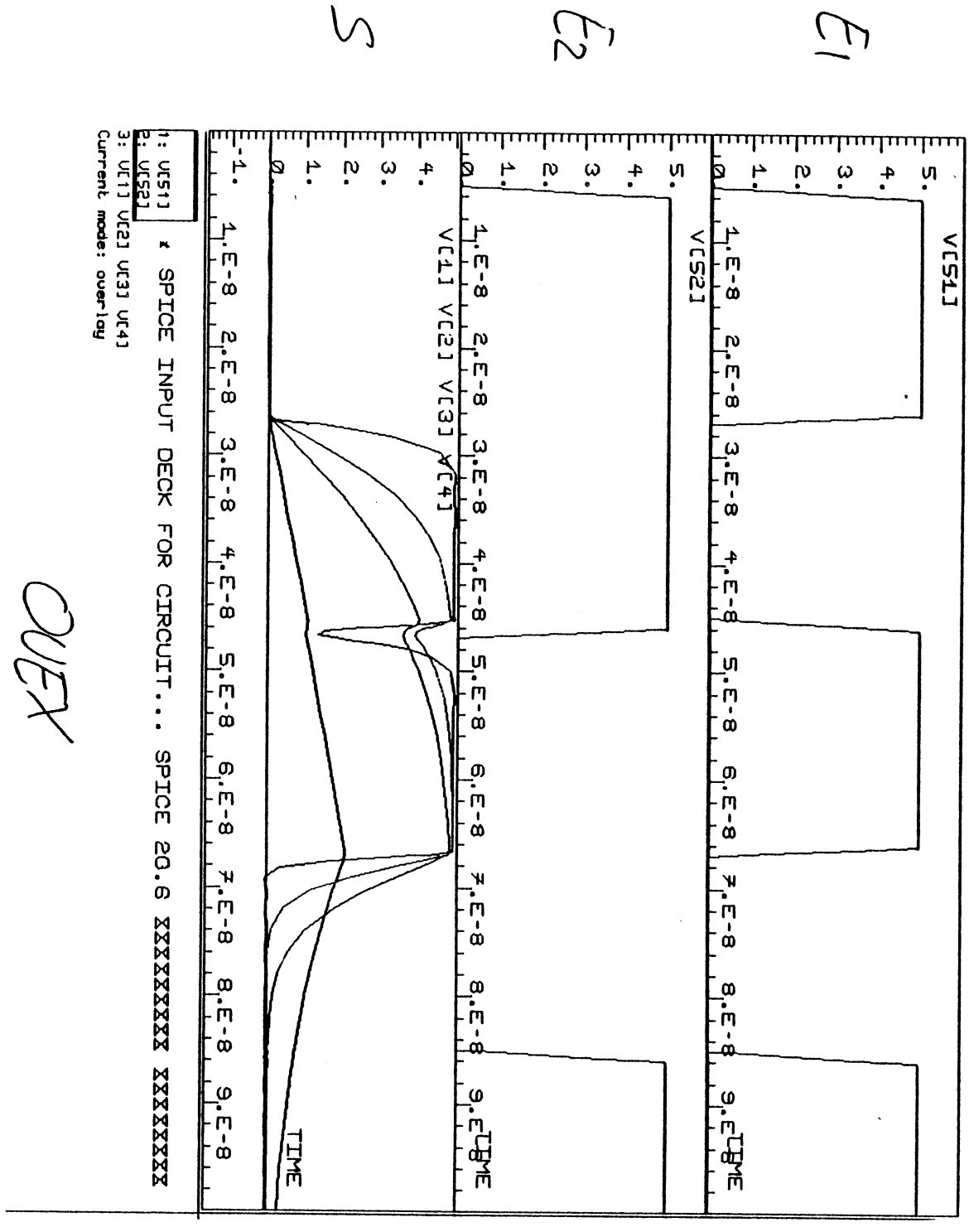


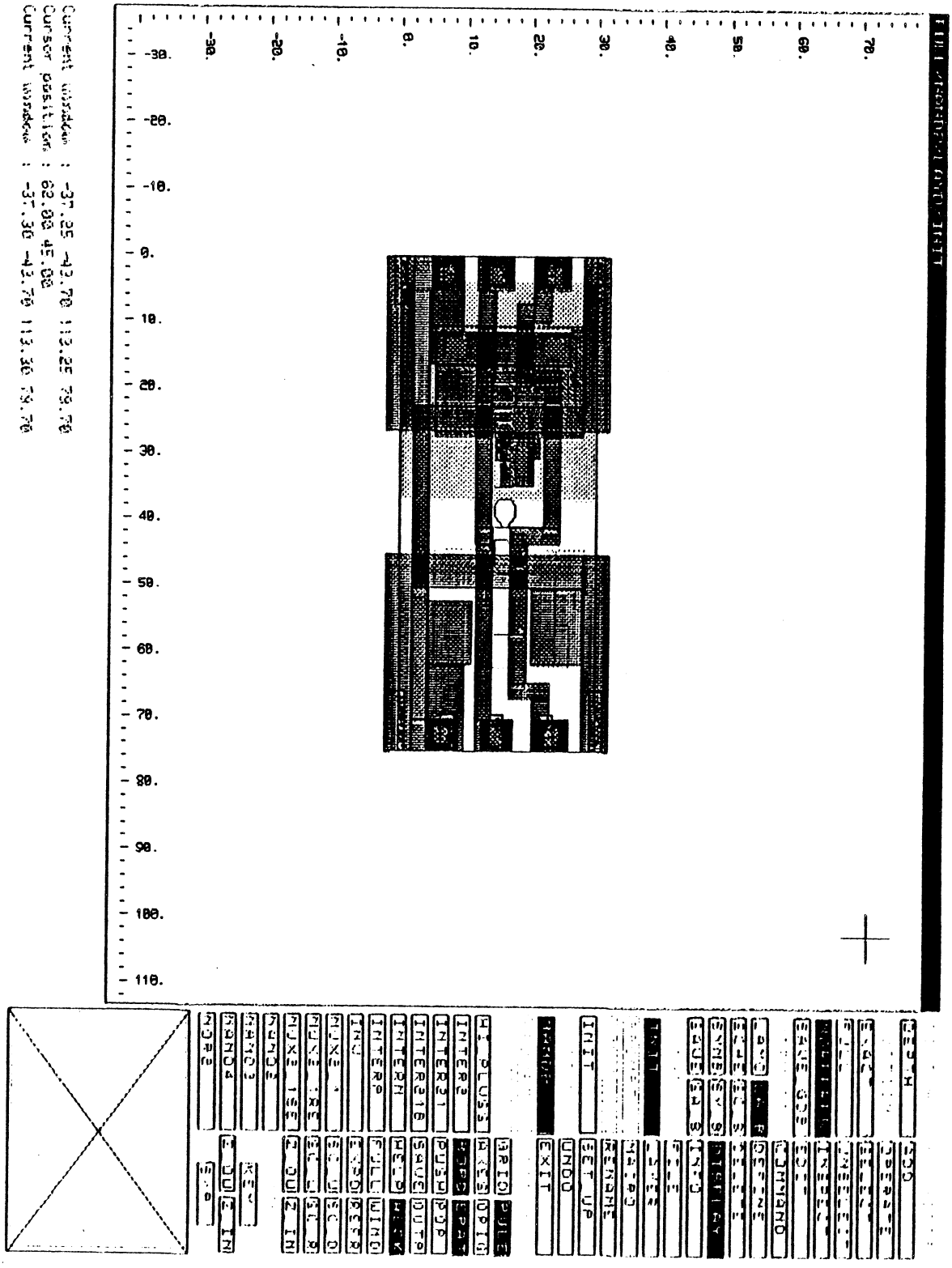
Schéma électrique du OU exclusif 2 entrées



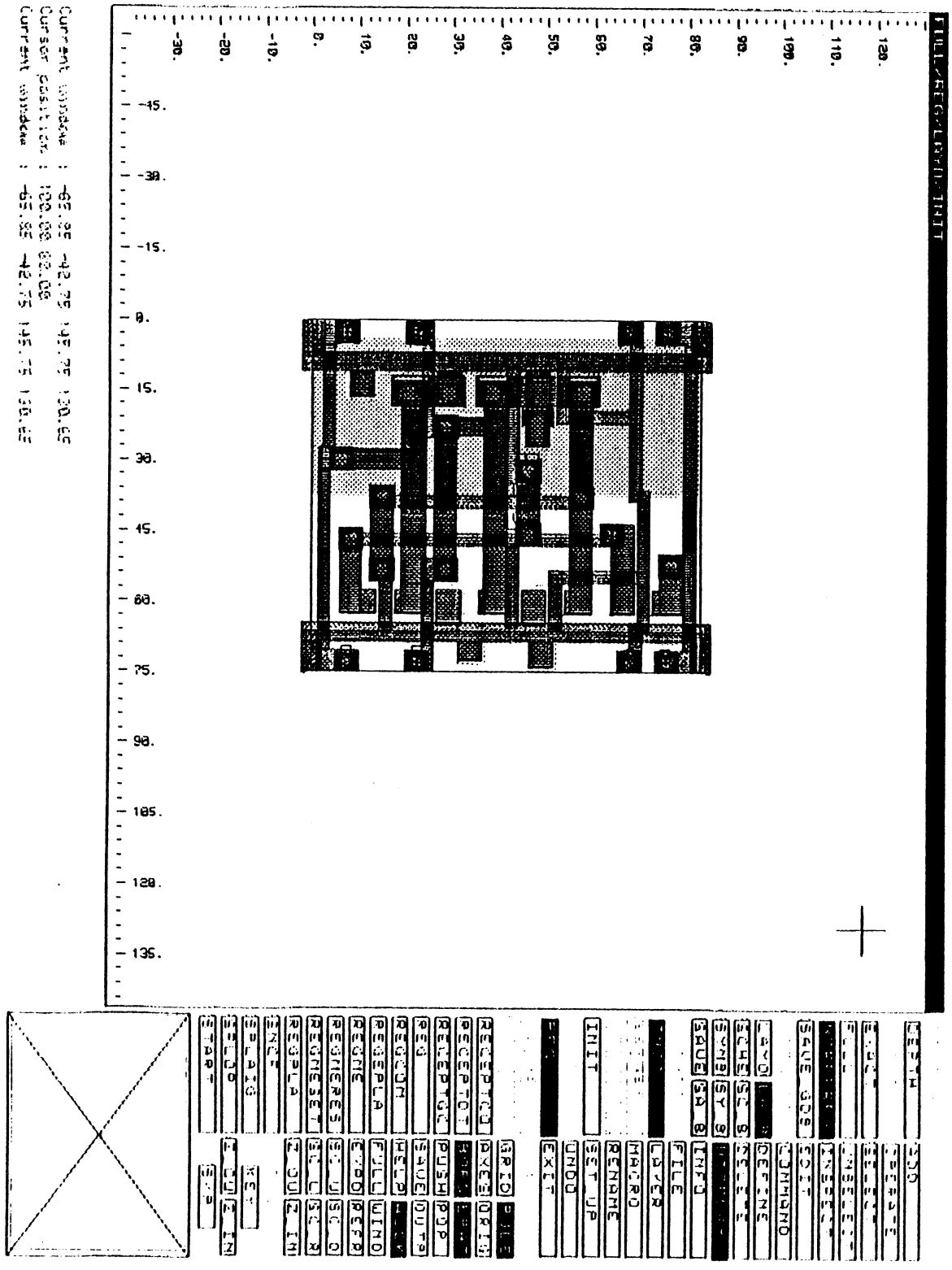
Simulation électrique du OU exclusif



Dessin des masques du NAND 2 entrées



Dessin des masques du latch



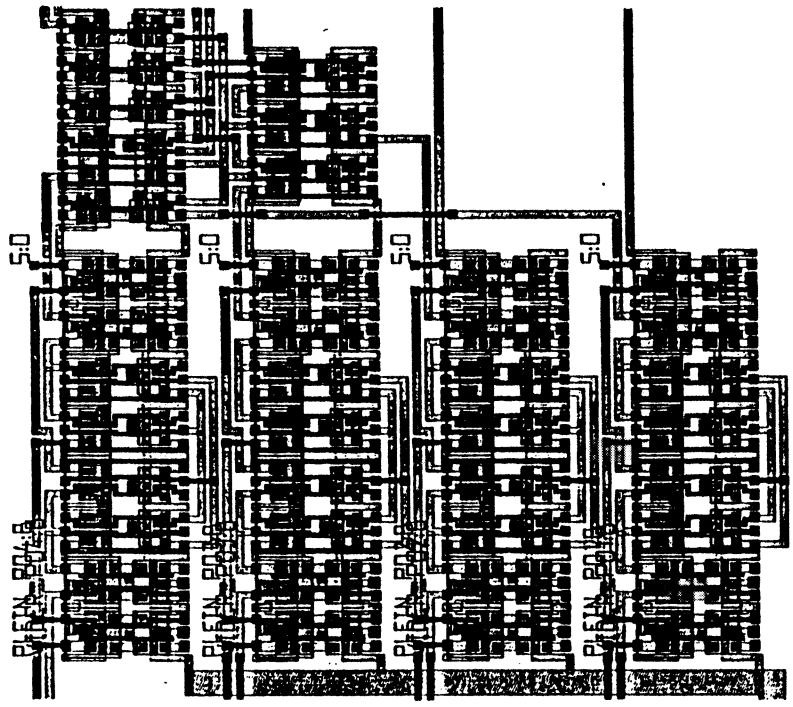
Annexe 2

Dessin des masques du circuit

Dans cette annexe, nous présentons certaines parties significatives du circuit.

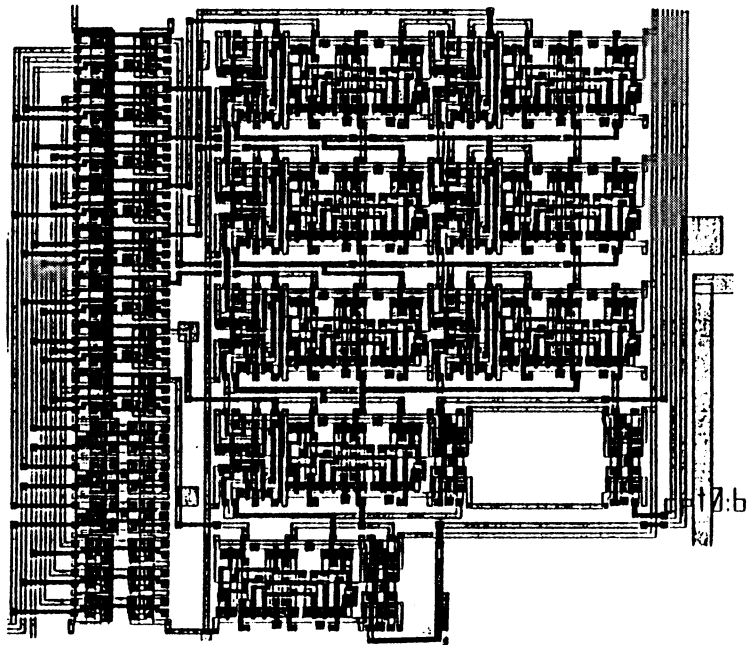
Schéma de la partie opérative

Schéma de l'entité de chargement des drapeaux



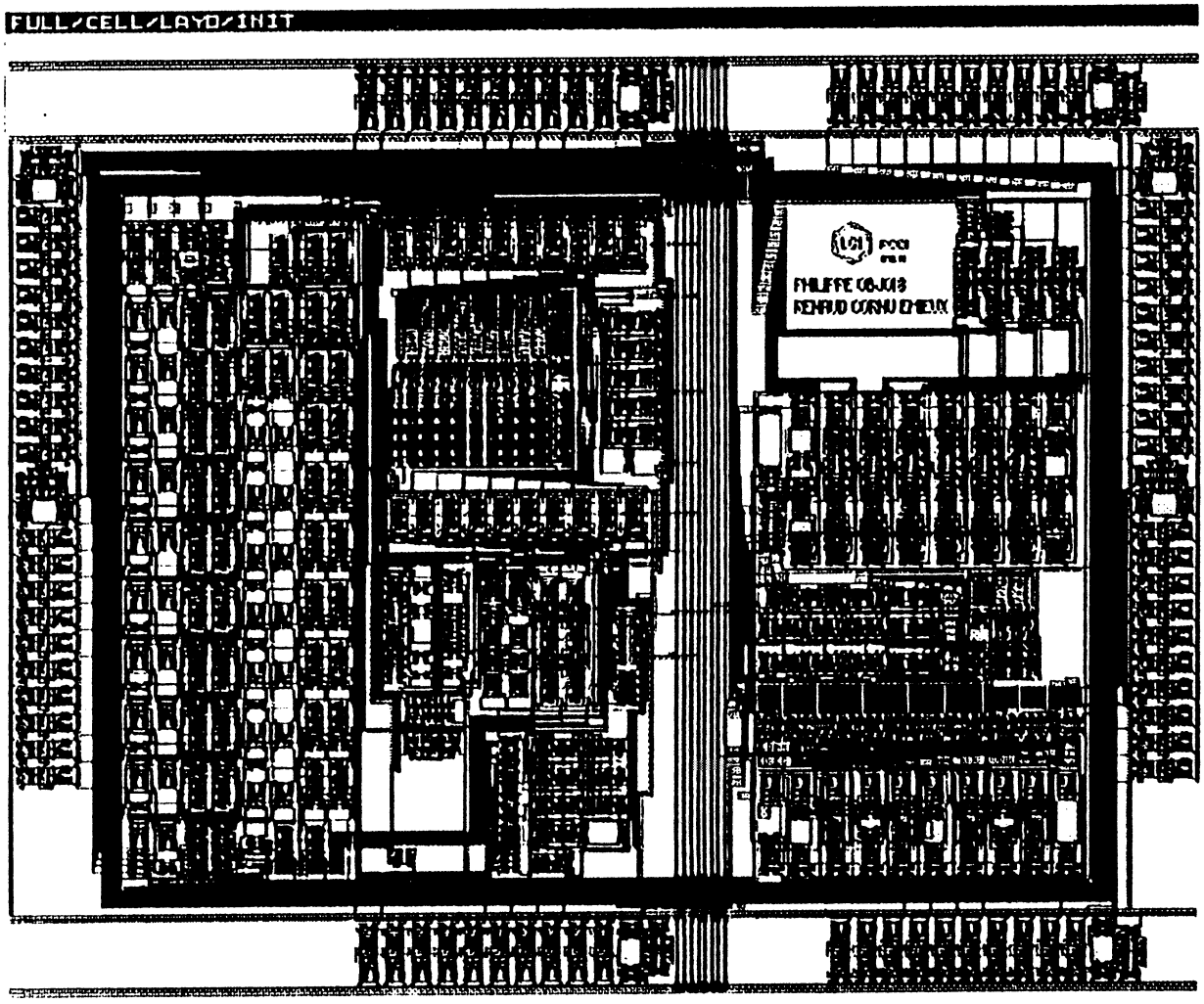
Annexe 2 : Dessin des masques du circuit

Schéma de l'entité de comptage des entrées



Annexe 2 : Dessin des masques du circuit

Schéma de la cellule



PROCESSEUR DE TRAITEMENT PROCESSEUR DE COMMUNICATION

TABLE DES MATIERES

Résumé :

Le développement des techniques d'intégration permet de réaliser des circuits de 10^5 à 10^6 transistors et dans un futur proche des circuits encore plus complexes. Les problèmes de CAO deviennent donc de plus en plus ardus comme la simulation logique ou le placement. Cette même évolution nous autorise à réaliser des machines parallèles très puissantes pour résoudre ces problèmes.

Nous proposons l'architecture d'un réseau cellulaire asynchrone. Ce réseau, composé de NxN cellules dont chacune est physiquement connectée à ses quatre voisines, dispose d'un mécanisme de communication permettant l'acheminement de messages d'une cellule quelconque à n'importe quelle autre. Un circuit intégré incluant une réseau de 2x2 cellules dédié à la simulation logique a été réalisé.

Utilisant cette architecture cellulaire, nous avons développé un placeur, qui à partir d'une configuration initiale, minimise la longueur des connexions par échange de paires. Nous avons abordé la manière dont le placement pourrait être amélioré par la méthode du recuit simulé.

Ces deux applications différentes de l'architecture cellulaire nous permettent de constater que beaucoup de parties sont communes aux deux circuits. Nous énonçons certaines règles de façon à rendre la conception plus rapide et plus sûre.

Development of integration techniques has increased today's Integrated Circuit complexity up to 10^5 to 10^6 transistors and let us assume that ICs will become more powerful in the near future. That is, some important steps of the CAD process such as logical simulation and placement are getting critical and hard to manage.

On the other hand this evolution has leaded to the realization of highly parallel machines able to handle such time consuming tasks. We propose a new asynchronous cellular architecture based on a regular array of NxN automata each of which is physically connected to its four neighbours a distributed transmission message mechanism allowing communications between any two cells of the array. We have designed a 2x2 cell array prototype IC dedicated to logical simulation.

We have also developped a placement engine by modifying cell algorithms. It acts from an initial placement map and minimizes connection lengths by using a distributed pairwise exchange algorithm. We present how a simulated annealing approach can improved its performances.

These two different applications of the cellular architecture allow us to note that a lot of parts are the same in the two circuits. We state some rulls to make the conception faster and safer.

Mots clefs :

Accélérateur Câblé, CAO de VLSI, Circuit Intégré, Parallélisme, Placement, Recuit Simulé, Réseau Cellulaire Asynchrone, Simulation Logique.

Asynchronous Cellular Network, Hardware Accelerator, Integrated Circuit, Logical Simulation, Parallelism, Placement, Simulated Annealing, VLSI CAD.