Réseau de cellules intégré : mécanisme de communication inter-cellulaire et application à la simulation logique

Philippe Objois

THESE

présentée par

Philippe OBJOIS

pour obtenir le titre de **DOCTEUR**

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

RESEAU DE CELLULES INTEGRE : MECANISME DE COMMUNICATION INTER-CELLULAIRE ET APPLICATION A LA SIMULATION LOGIQUE

Date de soutenance : 27 septembre 1988

Composition du jury : MM. Jacques MOSSIERE Président
Guy MAZARE
Jean-Paul SANSONNET
Daniel ETIEMBLE Rapporteurs
Patrice QUINTON

Thèse préparée au sein du Laboratoire LGI

Président: Georges LESPINARD

Année 1988

Professeurs des Universités

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

Professeur Université des Sciences Sociales (Grenoble II)

ALLIET	Louis

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

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

CHERCHEURS DU CNRS

Directeurs de recherche 1ère Classe

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

Directeurs de recherche 2ème Classe

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

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

ENSEEG

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

ENSERG

BOREL	Joseph

ENSIEG

DESCHIZEAUX	Pierre
GLANGEAUD	François
PERARD	Jacques
REINISCH	Raymond

ENSHMG

ROWE	Alain

ENSIMAG

COURTIN	Jacques

EFP

CHARUEL	Robert

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 agréees 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

UNIVERSITE SCIENTIFIQUE TECHNOLOGIQUE ET MEDICALE
DE GRENOBLE

Président de l'Université :

M. PAYAN Jean Jacques

ANNEE UNIVERSITAIRE 1987 - 1

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 Tror
ALARY Josette	Chimie Analytique	Faculté la Tror
BERIEL Hélène	Physiologie et Pharmacologie	Faculté la Tror
CUSSAC Max	Chimie Therapeutique	Faculté la Tror
DEMENGE Pierre	Pharmacodynamie	Faculté la Tror
FAVIER Alain	Biochimie	C.H.R.G.
JEANNIN Charles	Pharmacie Galenique	Faculté Meylan
LATURAZE Jean	Biochimie	Faculté la Tror
LUU DUC Cuong	Chimie Générale	Faculté la Tror
MARIOTTE Anne-Marie	Pharmacognosie	Faculté la Tror
MARZIN Daniel	Toxicologie	Faculté Meylan
RENAUDET Jacqueline	Bactériologie	Faculté la Tror
ROCHAT Jacques	Hygiène et Hydrologie	Faculté la Tror
SEIGLE MURANDI Françoise	Botanique et Cryptogamie	Faculté Meylan
VERAIN Alice (Mme Rapin)	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 Merc
	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 Merc
HOLLARD Daniel	Hématologie	C.H.R.G.
LATREILLE René	Chirurgie Thoracique/Cardiovasculaire	C.H.R.G.
LE NOC Pierre	Bactériologie - Virologie	Faculté la Merc
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 Merc
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 Merc
VIGNAIS Pierre	Biochimie	Faculté la Merc

PROFESSEURS 2EME CLASSE

BACHELOT Yvan	Endocrinologie	C.H.R.G.
BARGE Michel	Neurochirurgie	C.H.R.G.
BENABID Alim Louis	Biophysique	Faculté la Merci
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.

Je tiens à remercier

Mr. Jacques MOSSIERE, Directeur de l'ENSIMAG et du LGI, de me faire l'honneur de présider le jury de cette thèse ;

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

Mrs. Daniel ETIEMBLE, Professeur, Directeur de l'UFR d'informatique à l'Université Pierre et Marie Curie (Paris 6) 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 m'ont aidé par leurs conseils ;

Mr. Jean-Paul SANSONNET, Ingénieur à la CGE, responsable du projet MAIA et prochainement Directeur de Recherche au CNRS, Directeur du PRC "Architecture de Machines Nouvelles", qui a accepté d'être membre du jury de cette thèse.

Je tiens également à remercier

Mr. Jean-Luc RAINARD, Ingénieur au CNET/CNS de Meylan pour son aide précieuse et le temps qu'il m'a consacré.

à Véronique,
à Matthieu,
à mes parents,
à ma famille et belle famille.

Ce travail a été réalisé grâce au soutien du groupement C^3 du CNRS et avec l'aide du Centre Norbert Segard du CNET.

PLAN GENERAL

Il s'agit ici d'une table des matières sommaire, chaque chapitre ayant sa propre table des matières.

INTRODUCTION

Que ce soit pour la conception de circuits VLSI ou pour toute autre application complexe, le temps de calcul devient de plus en plus souvent trop long et le coût trop élevé. Pour cette raison, il y a actuellement un vif intérêt pour l'étude d'architectures totalement nouvelles permettant l'implémentation d'algorithmes spécifiques le plus souvent très parallèles.

Les architectures classiques du type Von Neumann sont limitées, de manière inhérente, par leur nature séquentielle. Les architectures hautement parallèles, composées d'automates ou de processeurs très simples travaillant ensemble et échangeant des données afin de produire un résultat global, semblent être plus prometteuses. Ces architectures sont d'autant plus intéressantes que l'on peut utiliser, de manière répétitive, un grand nombre de processeurs unitaires de complexité limitée.

Les réseaux systoliques [KUN82] sont un exemple bien connu des architectures parallèles. Ce sont des tableaux de processeurs simples; chacun d'eux peut communiquer avec seulement 4 ou 6 de ses voisins immédiats et exécutent simultanément le même algorithme simple. La caractéristique principale des réseaux systoliques (d'où leur nom) est leur fonctionnement synchrone au niveau global. En effet, c'est simultanément que tous les processeurs échangent des données dans un premier temps, puis calculent dans un deuxième temps. C'est pour cette raison que ces réseaux sont habituellement classés parmi les architectures SIMD (Single Instruction Multiple Data) [FLY72]. Ils nécessitent la définition d'algorithmes parallèles spécifiques (appelés algorithmes systoliques), décrits par les différents flots de données traversant le réseau pendant que les processeurs effectuent les calculs partiels.

Mais les réseaux systoliques ne peuvent apporter de solutions satisfaisantes qu'à des problèmes réguliers (par exemple analyse numérique ou traitement du signal). D'autre part, on peut trouver beaucoup d'algorithmes parallèles qui n'ont pas la régularité d'un traitement matriciel, parce qu'ils exigent des processeurs élémentaires pouvant communiquer avec plus de 4 ou 6 autres processeurs susceptibles d'être topologiquement éloignés. Les opérateurs sont

simples, mais ils n'émettent pas forcément le même nombre de messages qu'ils reçoivent, et les durées de chaque phase peuvent être très différentes. Pour exécuter de tels algorithmes, on peut les implémenter sur des architectures multiprocesseurs ou multiplexées et les programmer à l'aide de langages parallèles comme OCCAM [INM84].

Nous proposons une architecture cellulaire permettant d'exécuter ce type d'algorithmes. Ses principales caractéristiques sont :

- chaque processeur peut communiquer avec n'importe quel autre processeur du réseau. Les informations nécessaires à ces communications étant chargées dans chaque processeur au cours d'une phase d'initialisation ;
- chaque processeur fonctionne de façon indépendante, et peut à tout moment communiquer, avec le reste du réseau, de manière totalement asynchrone ;
- les processeurs que l'on appelle cellules sont très simples : ce ne sont pas des processeurs programmables, mais des automates paramétrables. Leur complexité est limitée, entre 5.000 et 10.000 transistors, ce qui nous permet d'envisager des circuits intégrant 10x10 cellules, et beaucoup plus dans le cas de machines prototypes câblées.

Il existe plusieurs techniques permettant de faire communiquer toutes les cellules du réseau entre elles, mais la plupart sont complexes et donc difficiles à intégrer en un minimum de surface. Les différents algorithmes que nous avons étudiés ont montré que la grande majorité des communications réelles ne traversent pas complètement le réseau. En effet, lors d'une phase de placement, on peut toujours organiser le réseau de façon à minimiser les distances entre des cellules échangeant des données. Il est évident que les algorithmes parallèles implémentés sur le réseau sont d'autant plus performants qu'ils possèdent une bonne localité.

Afin d'éviter l'emploi de bus horizontaux et verticaux traversant le réseau entier, les communications sont basées sur un échange de messages entre les cellules de type boîte aux lettres.

Comme première application de cette architecture, nous présentons la réalisation d'un simulateur logique basé. Nous avons choisi cette application car la modélisation d'un circuit logique s'apparente naturellement à un réseau de cellules ; de plus l'algorithme de simulation d'une entité logique (porte) est très simple donc intégrable sur une surface réduite.

Ces travaux font suite à ceux de Mr Jean Pierre Bernard qui a réalisé la première étude de cette architecture [BER85] et de Mr Yves Ansade qui a développé et simulé les algorithmes de simulation logique [ANS88]. La machine cellulaire a été réalisée conjointement avec Mr Renaud Cornu-Emieux qui s'est particulièrement attaché à l'intégration de la partie-traitement du simulateur logique [COR88]. Pour ma part, mon travail au niveau de la cellule de base a été d'étudier et d'intégrer un mécanisme de communication de message inter-cellulaire.

Dans le chapitre 1, nous présentons une architecture originale basée sur un réseau de cellules asynchrones communiquant par messages. Nous étudions l'implémentation d'algorithme de simulation logique sur cette architecture. Les problèmes de communication et de synchronisation sont discutés.

Le chapitre 2 est consacré à la réalisation d'un circuit intégrant un réseau 2x2 dédié à la simulation logique. L'implémentation de l'algorithme de communication est décrite. Les performances sont présentées.

Nous étudions au chapitre 3, une machine de simulation logique. Les protocoles et les interfaces de communication avec une machine-hôte sont décrits. Les algorithmes de la machine cellulaire sont présentés.

La présentation détaillée de la simulation logique comme outil de vérification de la conception fait l'objet d'une annexe. Les différents concepts de modélisation sont abordés, les algorithmes de simulation et leurs techniques d'accélération décrits. Enfin, nous présentons les machines de simulation les plus représentatives.

CHAPITRE 1

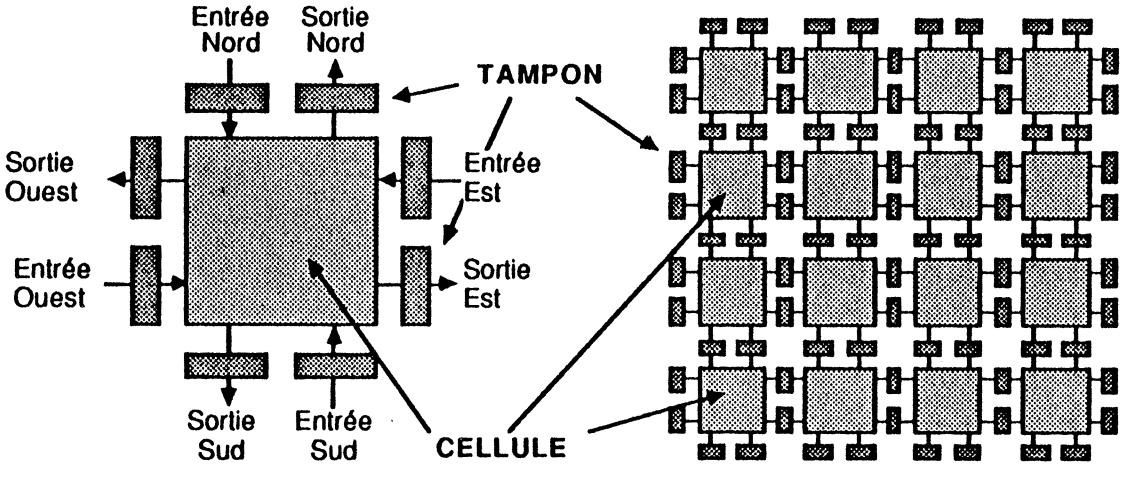
UNE MACHINE CELLULAIRE POUR LA SIMULATION LOGIQUE

TABLE DES MATIERES

1 LE RESEAU CELLULAIRE

1.1 Le réseau de cellules

L'architecture de la machine cellulaire est basée sur un réseau régulier de NxM cellules asynchrones organisées en une matrice plane. Chaque cellule exécute une tâche simple, et échange des informations avec ses 2, 3 ou 4 voisines immédiates (selon qu'elle se trouve sur un coin, un bord ou à l'intérieur du réseau). Chaque cellule est entourée de 4 tampons d'entrée, et de 4 tampons de sortie à travers lesquels s'effectuent les échanges inter-cellules. Un protocole de communication assure l'exclusion mutuelle de deux cellules sur un même tampon.

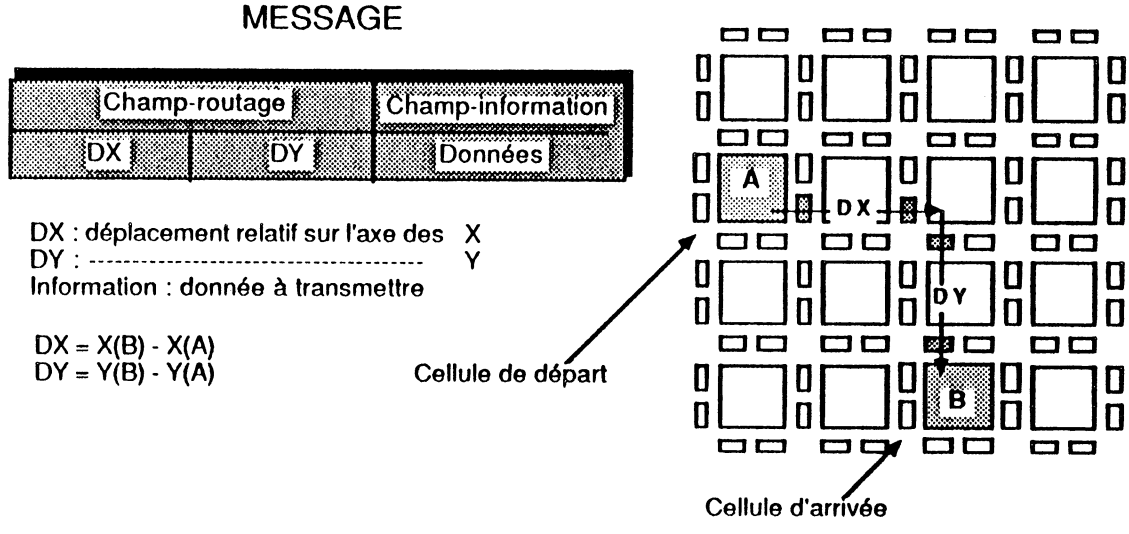


Architecture générale du réseau

1.2 Acheminement des messages

Une cellule peut communiquer avec n'importe quelle autre cellule du réseau, bien qu'elle ne soit en relation directe (physique) qu'avec ses voisines immédiates (nord, sud, est et ouest). Pour cela, chaque cellule intègre un mécanisme d'acheminement de messages. Un message est constitué de deux champs : le champ-adresse contenant la position relative de la cellule destinatrice par rapport à la position courante du message, et un champ-information contenant les données à transmettre. Les messages transitent de

cellule en cellule jusqu'à leur destination, leur champ-adresse est mis à jour à chaque traversée de cellule.



Exemple d'échange de données à travers le réseau

Si une cellule A doit envoyer une donnée à une cellule B, elle émet un message dont le champ-information contient la valeur de cette donnée, et dont le champ-routage contient la position relative de la cellule B par rapport à la cellule A :

- DX = X(B)-X(A),
- DY = Y(B)-Y(A).

Lorsqu'une cellule reçoit un message, elle le traite en fonction des valeurs du champ-routage :

→ DX = 0 et DY = 0 la donnée est arrivée à destination, et sera traitée par la cellule,

→ DX ≠ 0 ou DY ≠ 0 c'est la négation du cas précédant, le message est alors réacheminé par l'un des tampons périphériques.

Différentes stratégies d'acheminement sont possibles, nous avons notamment étudié un routage simple - acheminement en X, puis (lorsque DX=0) en Y - et plusieurs routages adaptatifs en vue d'une intégration sur tranche du réseau permettant de contourner des cellules, ou d'éviter des tampons défectueux [FAU86].

1.3 Les tampons

Les tampons sont les seuls liens entre les cellules. Ils sont unidirectionnels et ne peuvent contenir qu'un seul message de taille fixe : ils sont remplis par une cellule, et vidés par sa voisine (communication de type boîte aux lettres). Toutes les cellules du réseau fonctionnant de manière totalement asynchrone, il est nécessaire, pour éviter la perte ou la mauvaise lecture des messages, d'assurer l'exclusion mutuelle de deux cellules sur un même tampon. Cette exclusion est réalisée par un protocole de communication implémenté dans les cellules, ce protocole doit connaître, à chaque instant, l'état **vide** ou **plein** des tampons.

Un tampon sera donc composé d'une partie mémorisant le message à transmettre, et d'un drapeau indiquant son état.

1.4 La cellule

Une cellule est constituée de deux parties distinctes travaillant en parallèle de façon autonome. La partie-aiguilleur assure et gère l'acheminement des messages, la partie-traitement de la cellule exécute l'algorithme réparti implémenté sur le réseau. Ces deux parties communiquent entre elles par l'intermédiaire de tampons, selon le même principe que les communications inter-cellulaires.

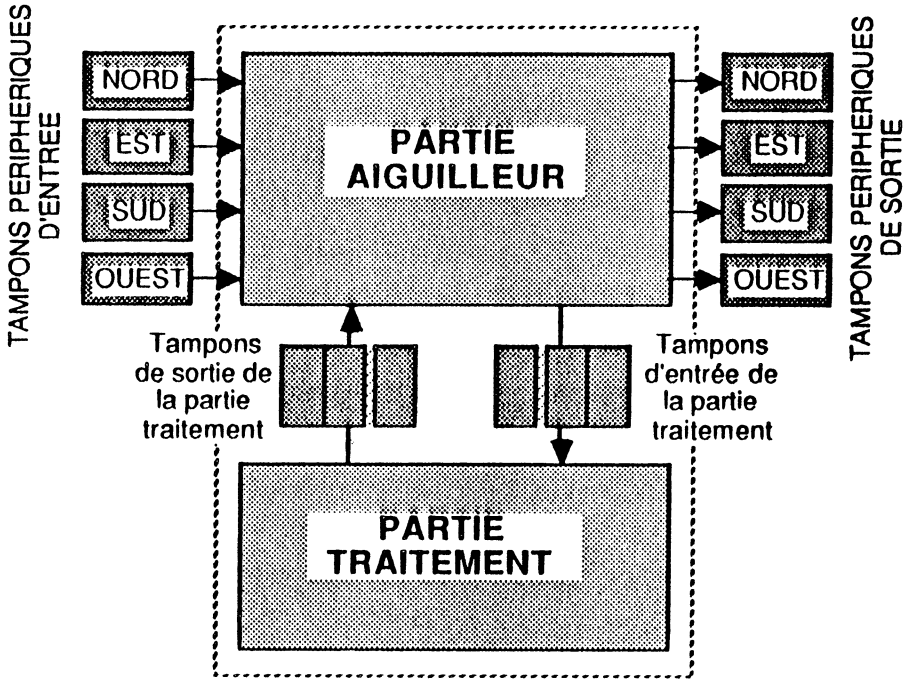


Schéma fonctionnel d'une cellule de base

Remarque : Il est possible, pour des questions de performance, de découpler la partie-aiguilleur de la partie-traitement, chacune fonctionnant alors de manière asynchrone.

1.4.1 La partie-aiguilleur

Elle est chargée de l'acheminement des messages entre les tampons d'entrée (les quatre tampons périphériques d'entrée et les tampons de sortie de la partie-traitement) et les tampons de sortie (les quatre tampons périphériques de sortie et ceux d'entrée de la partie-traitement). Dés qu'un tampon d'entrée est plein, l'aiguilleur détermine le tampon de sortie vers lequel l'acheminer. Tant que celui-ci est plein, le message est laissé en attente dans le tampon d'entrée. Si le tampon de sortie est vide (ou lorsqu'il sera vide), le message à acheminer est traité. Son champ-routage est mis à jour , et il est alors transféré dans le tampon de sortie déterminé précédemment. Le drapeau associé au tampon est mis également à jour (vide → plein). Un mécanisme de priorité permet à l'algorithme implémenté dans l'aiguilleur, de lever les éventuels conflits d'accès.

1.4.2 La partie-traitement

Alors que la partie-aiguilleur peut, dans une certaine mesure, être indépendante de l'algorithme réparti implémenté sur le réseau, la partie-traitement est spécifique à chaque application implémentée sur le réseau.

Bien que les cellules soient asynchrones, certains signaux de synchronisation sont indispensables au bon fonctionnement du réseau. C'est le cas du signal RESET dont le rôle et de rendre cohérent l'état du réseau. Cette commande est par nature câblée. Selon les algorithmes implantés, il peut être nécessaire d'initialiser les cellules du réseau à des valeurs prédéterminées lors d'une phase de pré-calcul de l'ordinateur-hôte; on pourra alors utiliser le mécanisme d'acheminement pour paramètrer chaque cellule (par exemple les adresses des cellules connectées en aval).

D'une manière générale, les algorithmes implémentés comportent une boucle infinie dans laquelle la partie-traitement attend tous ses messages d'entrée, détermine le résultat de la fonction logique qu'elle simule, et enfin diffuse ce résultat sous la forme de messages de sortie.

Selon les applications implémentées, on peut avoir besoin de synchroniser (ou de resynchroniser) les cellules entre elles. Nous avons étudié deux techniques possibles.

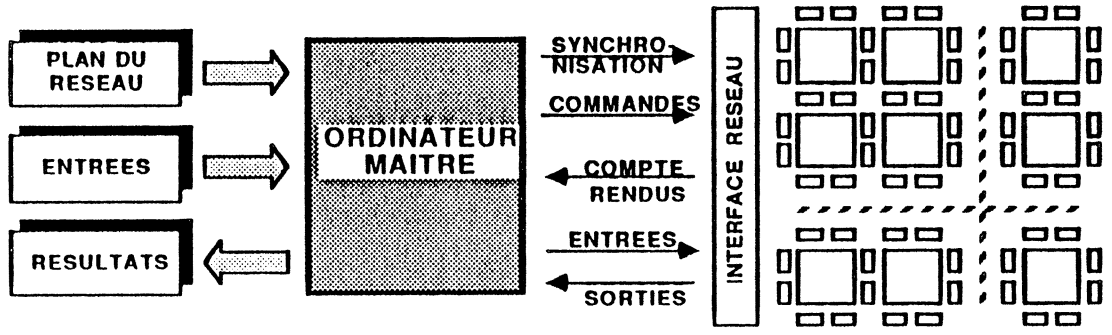
D'abord une solution "hardware", sous la forme d'un ET logique entre toutes les cellules sur un signal donné (par exemple un signal de FIN), la contrepartie sera un signal de DEBUT envoyé de façon globale par l'ordinateur-hôte à chaque cellule.

Une deuxième solution, plus décentralisée, est basée sur l'envoi de messages d'accusés de réception entre les cellules.

1.5 La machine cellulaire

Afin de pouvoir échanger des messages avec l'extérieur, chaque entrée/sortie du réseau est reliée à une interface simple. On peut ainsi assembler plusieurs circuits cellulaires pour former un réseau plus grand, et interfacer le réseau avec un ordinateur-hôte.

Dans le cas d'une machine cellulaire, l'ordinateur-hôte prend en charge la gestion du réseau. A part les commandes de synchronisation globale (reset ...) et de contrôle qui sont câblées, les échanges avec le réseau, que ce soient les entrées (données) ou les sorties (résultats), utilisent toujours le même mécanisme d'acheminement implanté dans le réseau.



Architecture de la machine cellulaire

1.5 Les applications

Une telle architecture cellulaire peut servir de support pour accélérer tout algorithme massivement parallèle nécessitant une puissance de calcul importante.

Nous étudions actuellement plusieurs applications notamment :

- un simulateur logique qui est présenté au cours du paragraphe suivant, et dont la réalisation est décrite au chapitre 2,
- un placeur (à partir d'un algorithme d'optimisation par échange de paires) [COR88],
- la reconstruction d'images [LAT88a] [LAT88b],
- un réseau neuronique multi-couches basé sur un algorithme d'apprentissage par rétro-propagation des gradients [FAU88a] [FAU88b],
- une machine à flot de données dédiée à l'exécution de programmes écrits en langage LUSTRE [PAY88].

2 LA MACHINE DE SIMULATION LOGIQUE *

Avec la complexité croissante des circuits intégrés, la puissance de calcul nécessaire à leur conception devient de plus en plus grande. De nombreux "accélérateurs de simulation" (hardware simulators) ont été conçus pour répondre à cette demande, dont le plus connu est certainement le système ZYCAD. Pour accélérer une simulation, on fait le plus souvent appel au parallélisme : on partitionne le circuit à simuler en sous-circuits, et l'on distribue la simulation de ces sous-circuits sur des processeurs travaillant en parallèle. Afin de prendre en compte les connections entre les sous-circuits, les processeurs doivent pouvoir communiquer et se synchroniser. Cette technique de parallélisation a prouvé son efficacité, malgré les ralentissements dus à ces communications et synchronisations. Ces inconvénients peuvent d'ailleurs être réduits par des algorithmes sophistiqués de partitionnement du circuit en sous-circuits [AGR86].

Nous avons appliqué cette idée à notre structure cellulaire, de façon hautement parallèle. Le principe est d'assigner chaque objet du réseau logique (par exemple une porte) à une cellule du réseau. Chaque cellule simule une porte : elle reçoit des messages des portes dont les sorties forment ses entrées, calcule à partir de ces données une nouvelle valeur de sortie (état logique de la porte), et envoie un message aux cellules connectées en aval.

2.1 Déroulement d'une simulation

La description de la simulation est fournie par l'utilisateur. Elle peut être issue d'un éditeur de schémas, ou provenir d'un fichier de description au format standardisé tel que HILO ou EPILOG. Elle doit comporter les renseignements suivants :

- la description structurelle du circuit décrivant, en terme de portes logiques et d'interconnexions, le circuit à simuler ;

(*) Une présentation complète de la simulation logique (modélisation, algorithmes, techniques d'accélération ...) est présentée dans l'annexe 1.

- la description dynamique des éléments du circuit décrivant les paramètres temporels associés aux portes et aux équipotentielles ;
- les paramètres de la simulation proprement dite : états initiaux, chronogrammes des entrées, durée de la simulation, sorties à visualiser.

A partir de ces données initiales, l'ordinateur-hôte va enchaîner et contrôler les quatre étapes de la simulation :

① Compilation des données,
② Initialisation du réseau,
③ Simulation,
④ Edition des résultats.

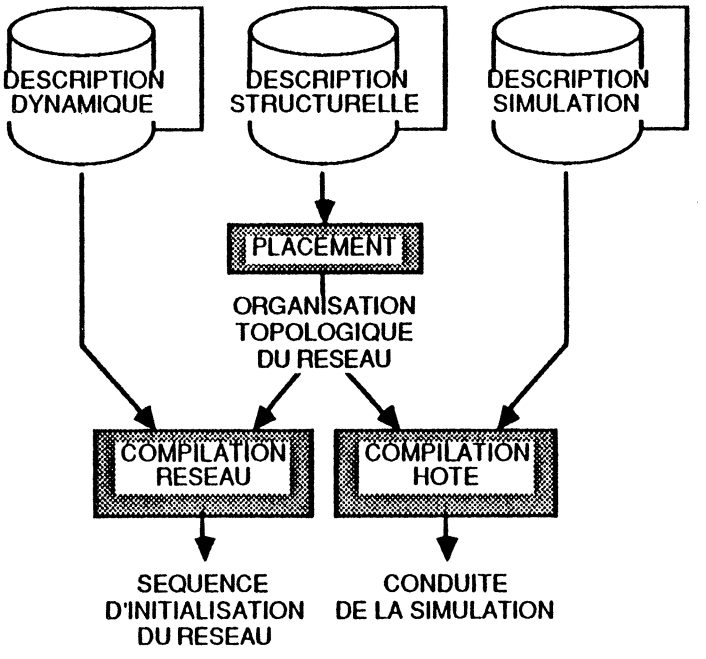
Les étapes 3 et 4 peuvent être exécutées en parallèle.

2.1.1 La compilation des données

Au cours de l'étape de compilation, l'ordinateur-hôte génère à partir des données de simulation, d'une part la séquence d'initialisation du réseau, d'autre part le fichier de conduite pas à pas de la simulation.

Cette étape qui a comme résultat intermédiaire l'organisation topologique du réseau, se décompose en trois parties :

- le placement des portes logiques du circuit sur les cellules du réseau,
- la génération des fonctions associées à chaque cellule du réseau,
- la génération du fichier de conduite pas à pas de la simulation.



La compilation des données

Le placement

Le placement consiste à affecter, à chaque cellule du réseau, une porte logique du circuit à simuler. Le placement est directement lié à l'interconnexion des portes. Cela suppose que le nombre de cellules du réseau soit supérieur ou égal au nombre de portes du circuit à simuler. Le placement est un point critique, car il conditionne la longueur (implantée physiquement) des chemins à parcourir par les messages, et donc le temps de simulation. Nous avons étudié plusieurs algorithmes de placement notamment par échange de paires [COR88].

Génération des caractéristiques des cellules

Si l'on sait maintenant où sont implantées les portes sur le réseau, il reste à préciser les caractéristiques de chaque cellule. A partir des descriptions structurelles et dynamiques du circuit à simuler et de l'organisation topologique du réseau, on détermine les caractéristiques de chaque cellule (nombre d'entrées, de sorties, adresses relatives des cellules connectées en aval ...). C'est également au cours de cette phase que l'on vérifiera la cohérence de la

description structurelle du circuit. L'ensemble de ces données forment la séquence d'initialisation du réseau.

2.1.2 L'initialisation du réseau

Au cours de cette étape, l'ordinateur-hôte va envoyer à chaque cellule du réseau, tous les paramètres nécessaires à sa programmation. La séquence d'initialisation du réseau est alors découpée en messages qui seront envoyés aux cellules par le mécanisme d'acheminement de messages du réseau.

Cette initialisation impose deux modes de fonctionnement du réseau, un mode d'initialisation et un mode de calcul (simulation). Le signal RESET (contrôlé par l'ordinateur-hôte) force chaque cellule du réseau à entrer dans le mode de fonctionnement initialisation. Après réception d'un nombre de messages prédéfinis (messages d'initialisation de la cellule), la cellule est prête à exécuter l'étape suivante. L'étape d'initialisation se termine lorsque l'ordinateur-hôte a reçu de toutes les cellules un signal de fin d'initialisation (signal câblé).

2.1.3 La simulation

Génération du fichier de conduite de la simulation

A l'aide de la description de la simulation et de la topologie du réseau, on détermine à l'avance tous les échanges entre l'ordinateur-hôte et le réseau. Ce fichier comporte notamment la séquence de stimulis d'entrée du réseau (points d'entrée, heures, cellules destinatrices, valeurs), et celle de sortie (points de sortie).

La simulation proprement dite

les différentes simulations d'un même circuit ont en commun la séquence d'initialisation du réseau.

L'ordinateur-hôte injecte les stimulis d'entrée de la simulation au réseau, les cellules exécutent l'algorithme réparti de simulation logique. Le paragraphe suivant décrit le type de simulation, et le mode de synchronisation implémenté.

2.1.4 L'édition des résultats

A chaque pas de la simulation, des messages-résultats sont envoyés par les cellules du réseau à l'ordinateur-hôte qui les mémorise dans un fichier résultat. Le traitement de celui-ci (édition à l'écran ou impression de chronogrammes sous différentes formes...) peut être immédiat ou différé.

2.2 La simulation

2.2.1 Type de simulation

Nous avons implémenté sur cette machine cellulaire, un algorithme de simulation logique à délai unitaire, à trois états (0, 1 et X). Ce type de simulation a été retenu pour des questions de taille de circuit. En effet, la puissance de calcul d'un tel réseau ne peut être vraiment intéressante que si l'on dispose d'un très grand nombre de cellules élémentaires, le but final étant son intégration sur tranche (Wafer Scale Integration). Il est donc nécessaire de concevoir des cellules unitaires les plus petites possibles. Dans cette optique, l'implantation d'un algorithme à échéancier distribué sur le réseau, ne pouvait faire l'objet de la première réalisation de machine cellulaire.

Toujours sous la même contrainte, nous avons défini les caractéristiques des portes logiques de base de notre simulateur. Le comportement d'une cellule est entièrement décrit à l'aide de cinq paramètres : la fonction logique de base, l'inversion du résultat, le délai, l'entrance et la sortance.

La fonction logique de base

Elles sont au nombre de quatre : ET, OU, OUEX et COLLAGE à 0. Ces fonctions possèdent la particularité d'être associatives.

L'inversion du résultat

Comme son nom l'indique, ce paramètre permet l'inversion logique de la sortie de la porte. Il autorise donc les fonctions logiques supplémentaires : NON-ET, NON-OU, NON-OUEX et COLLAGE à 1.

Le délai

La simulation, à délai unitaire, implémentée autorise la prise en compte de retards fixes (multiple du pas de simulation) lors de la traversée des portes. Quatre valeurs de retard sont possibles : retard nul, retard de un, deux ou trois pas.

L'entrance

On appelle entrance, le nombre d'entrées d'une porte. Elle est limitée à huit.

La sortance

C'est le nombre de sorties d'une porte. Elle est au maximum de quatre. Les cellules doivent mémoriser les adresses de toutes les cellules connectées en aval de celle-ci.

Une porte ne peut être directement reliée qu'à quatre portes en aval (sorties) et huit en amont (entrées).

La fonction logique NOP (non opération), ne fait pas partie des fonctions de base implémentées car elle peut être réalisée par les fonctions ET, OU ou OUEX à une seule entrée. Elle permet de réaliser différentes fonctions telles que :

→ retard sur une équipotentielle,
→ retard supérieur à trois pas de simulation,
→ sortance supérieure à quatre,
→ réacheminement de message.

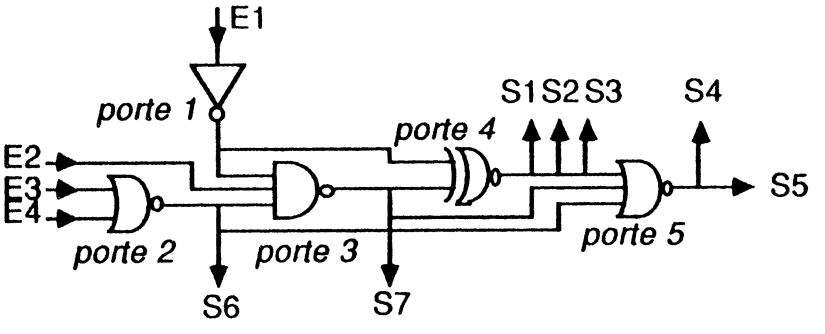
De même, la fonction INVERSE est obtenue par les fonctions ET, OU ou OUEX inversées, à une seule entrée. Ces fonctions doivent être programmées lors de la phase de compilation des données.

Lorsque le nombre de portes du circuit à simuler n'est pas exactement égal à celui des cellules du réseau, certaines cellules se retrouvent "en trop" et sont inutiles fonctionnellement : on utilisera leurs parties-aiguilleurs, mais leurs parties-traitements ne seront jamais activées. Il est cependant nécessaire qu'elles soient programmées afin de ne pas bloquer le déroulement de la simulation (voir au chapitre 2, l'algorithme de simulation implémenté dans

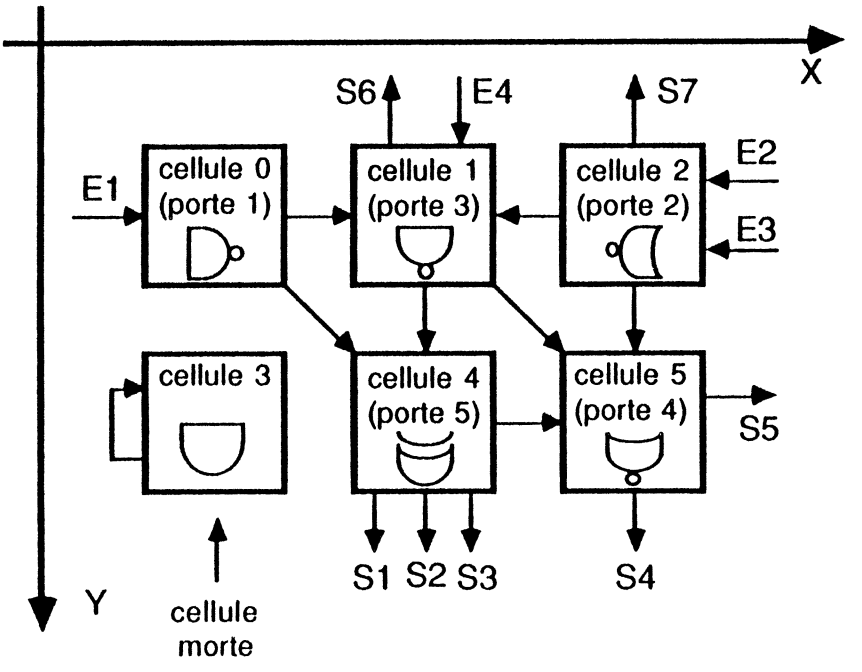
les cellules). Ces cellules sont appelées cellules mortes. Ce code-opération n'existant pas, on peut "tuer" une cellule en la programmant comme un retard nul sur une équipotentielle, ou comme un élément logique à une seule entrée bouclant sur lui-même, avec un retard non nul.

Exemple de programmation du réseau

Nous allons prendre comme exemple de programmation du réseau, la fonction combinatoire donnée par le schéma suivant :



Placement des portes sur le réseau



Caractéristiques des portes

No	Fonction	Inversion résultat	Entrance	Sortance	Retard	Adresses des sorties (dx, dy)
0	ET	oui	1	2	0≤r≤3	(1, 0), (1, 1)
1	ET	oui	3	3	0≤r≤3	(0, 1), (1, 1), (-, -)
2	OU	oui	2	3	0≤r≤3	(-1, 0), (0, 1), (-, -)
3	ET	non	1	1	r>0	(0, 0)
4	OUEX	non	2	4	0≤r≤3	(1, 0), (-, -), (-, -), (-, -)
5	OU	oui	3	2	0≤r≤3	(-, -), (-, -)

2.2.2 Implémentation de la simulation

Nous avons implanté sur le réseau, un algorithme de simulation à délai unitaire de manière très parallèle : il est distribué sur toutes les cellules du réseau. La façon la plus simple de réaliser celui-ci, est d'exécuter dans la partie-traitement de chaque cellule l'algorithme suivant :

```
tant que vrai faire
    pour toutes les entrées de la porte simulée faire
        attendre un message;
        mémoriser le message;
    fin pour
    évaluer le nouvel état de sortie de la porte simulée;
    envoyer les message de sortie vers les cellules connectées en
    aval (et éventuellement vers l'ordinateur-hôte);
fin tant que
```

Malheureusement, cet algorithme est trop simple et inexact. Les cellules fonctionnent de manière asynchrone, et les différents messages d'entrée d'une cellule ne parcourent pas tous la même distance. Certaines cellules peuvent donc recevoir leurs entrées, et émettre leurs sorties beaucoup plus vite que d'autres. Plusieurs messages successifs peuvent donc concerner la même entrée alors que les autres entrées n'ont toujours pas reçu les leurs.

Lié au mode de fonctionnement asynchrone des cellules, le problème du respect de la chronologie est capital pour un simulateur

logique. Une cellule peut recevoir au même temps t des messages émis à des dates différentes ($t+\delta_1$ et $t+\delta_2$). Il est donc nécessaire de synchroniser d'une manière ou d'une autre les différentes cellules entre elles, à chaque pas de simulation.

2.2.2.1 Algorithme synchrone

La solution la plus simple consiste à resynchroniser toutes les cellules à chaque passage dans la boucle principale de l'algorithme (c'est-à-dire à chaque fois que l'on incrémente l'horloge de la simulation). On peut réaliser cela à l'aide de deux signaux globaux de synchronisation, DEBUT et FIN, qui sont distribués à chaque cellule (de la même façon que les signaux d'alimentation, d'horloge et d'initialisation), ainsi qu'à l'ordinateur-hôte.

A la fin de chaque pas, chaque cellule émet un signal FIN. Tous ces signaux parviennent à l'ordinateur-hôte sous la forme d'un ET logique : il sera donc averti, que toutes les cellules du réseau ont terminé le même pas de simulation. Après avoir incrémenté l'heure de simulation courante, il autorisera les cellules à poursuivre l'exécution de leur algorithme par l'activation du signal DEBUT.

L'algorithme déroulé par la cellule en mode de fonctionnement simulation est maintenant le suivant :

```
tant que vrai faire
    attendre le signal DEBUT;
    envoyer le résultat évalué précédemment aux cellules
    connectées en aval;
    pour chaque entrée faire                  || Attendre un nombre
        recevoir un message;                  || de messages égal à
        mémoriser le message;                 || l'entrance de la porte
    fin pour                                  || simulée
    évaluer la nouvelle valeur de sortie;
    activer le signal FIN;                    || fin de l'évaluation
fin tant que
```

Remarque : Cet algorithme est valable pour des portes introduisant un retard temporel unitaire lors de leur fonctionnement. Dans le cas d'éléments logiques à délai nul, l'ordre de l'émission des sorties et la réception des entrées est inversé et l'algorithme devient :

tant que vrai **faire**
    attendre le signal DEBUT;
    Attendre un nombre de messages égal à l'entrance de la porte;
    évaluer la nouvelle valeur de sortie;
    envoyer le résultat aux cellules connectées en aval;
    activer le signal END; || fin de l'évaluation
fin tant que

Remarque : la seule restriction est l'impossibilité de simuler des boucles dans lesquelles tous les éléments ont un retard nul. Mais cela n'a aucune importance, car ce n'est pas réaliste.

De son côté, l'ordinateur-hôte exécute l'algorithme de contrôle suivant :

t:= heure-initiale;
tant que t ≤ heure-finale **faire**
    activer le signal DEBUT;
    envoyer les stimulis d'entrée de l'heure t;
    recevoir un nombre de messages égal à la sortance du circuit à simuler;
    attendre le signal FIN;
    t:= t+1;
fin tant que

Nous utilisons la même technique de synchronisation lors de l'initialisation du réseau. Dans ce mode de fonctionnement, l'algorithme du processeur-hôte est le suivant :

début
émettre le signal RESET;
envoyer à chaque cellule du réseau la séquence de programmation nécessaire;
attendre le signal FIN;
passer en mode de fonctionnement **simulation;**
fin

Parallèlement, les cellules déroulent l'algorithme d'initialisation suivant :

lorsque RESET **faire**
recevoir et mémoriser un nombre prédéfini de messages d'initialisation
initialiser à inconnu la sortie de la partie-traitement
envoyer le signal FIN
passer en mode de fonctionnement **simulation**
fin

La synchronisation de la simulation par les signaux DEBUT et FIN a deux principaux inconvénients. L'implantation de ces signaux sur le silicium, occupe une surface importante, et ceux-ci induisent de fortes capacités qui ralentissent le fonctionnement de la simulation. L'intégration au niveau WSI d'une telle forme de synchronisation pose également le problème d'une coupure toujours possible de ces signaux qui condamnerait toute la tranche. Un autre inconvénient, vient du fonctionnement global synchrone des cellules sur un réseau qui est par nature asynchrone. Cette utilisation non optimale du réseau, qui est inhérente à l'algorithme de simulation implémenté, se traduit également par une vitesse d'exécution moindre de la simulation.

2.2.2.2 Algorithme asynchrone

On peut limiter les inconvénients d'une synchronisation au niveau global, en synchronisant les cellules entre elles (et non au niveau global), simplement en utilisant des accusés de réception. Tous les messages acheminés à travers le réseau sont doublés,

lorsqu'ils sont arrivés à destination, d'un accusé de réception. La synchronisation se fait donc sur les échanges de messages : une cellule doit attendre les accusés de réception des messages qu'elle a envoyés à l'heure t avant d'envoyer ceux de l'heure t+1. N'ayant plus à séquencer la simulation, l'ordinateur-hôte ne fait plus qu'injecter des messages dans le réseau d'un côté, et récupérer ceux qui en sortent de l'autre.

L'algorithme implanté dans la cellule devient : soit n l'entrance, m la sortance, et r le retard de la porte simulée

```
tant que vrai faire
    si r>0
    alors
        envoyer le résultat évalué précédemment aux m
        cellules connectées en aval;
        par
            attendre les n messages d'entrée;
            évaluer la nouvelle valeur de sortie;
        et
            attendre m accusés de réception;
        fin par
    sinon
        attendre les n messages d'entrée;
        évaluer la nouvelle valeur de sortie;
        envoyer le résultat aux m cellules connectées en aval;
        attendre m accusés de réception;
    fin si
    envoyer un accusé de réception à chacune des n cellules
    connectées en amont;
fin tant que
```

Remarque : la structure **par** I1 **et** I2 **fin par** autorise l'exécution en parallèle des séquences d'instructions I1 et I2.

Les échanges de messages entre le réseau et l'ordinateur-hôte sont également synchronisés par accusé de réception. L'algorithme de l'ordinateur-hôte est le suivant :

```
t:= heure-initiale;
tant que t ≤ heure-finale faire
    | envoyer les stimulis d'entrée de l'heure courante t;
    | recevoir un nombre de messages égal à la sortance du circuit;
    | envoyer les accusés de réception des messages sortis du
    | réseau ;
    | attendre les accusés de réception de tous les stimulis émis;
    | t:= t+1;
fin tant que
```

Remarque : l'algorithme décrit ci-dessus est volontairement séquentiel, la version parallèle (si l'ordinateur-hôte le supporte) est beaucoup plus efficace : en accusant réception des messages au fur et à mesure de leur sortie du réseau, on répartit mieux les accusés de réception dans le réseau.

Lors de l'initialisation du réseau on se trouve devant le problème suivant :

> comment faire passer les cellules du mode de fonctionnement initialisation à celui de simulation, sans risque de confusion entre des messages d'initialisation (venant de l'hôte) et d'entrée (venant d'autre cellules) ?.

Une première solution consiste à typer le champ-information des messages, mais la taille des messages est un point trop critique dans l'implantation du réseau. Un autre moyen serait de faire appel à un double accusé de réception entre l'ordinateur-hôte et les cellules du réseau à la fin de la phase d'initialisation.

Mais la solution la plus simple consiste à maintenir le signal RESET haut pendant toute la phase d'initialisation. Lorsque le système-hôte est sûr que toutes les cellules du réseau sont programmées, il fait alors passer les cellules du réseau en mode de fonctionnement-simulation en faisant "retomber" le signal RESET.

Pour implémenter la synchronisation du réseau par accusés de réception, chaque cellule doit contenir les adresses relatives des cellules auxquelles elle est connectée en amont (celles qui lui envoie

un message d'entrée). Ce qui augmente la taille de la cellule de base du réseau proportionnellement à l'entrance maximum des cellules. Autre conséquence, la séquence d'initialisation du réseau est plus longue. Mais l'inconvénient majeur de la synchronisation par accusés de réception est le nombre des messages transitant dans le réseau, et leur propagation par vagues à travers le réseau à la fin des pas de simulation.

Les simulations de réseaux cellulaires, intégrant ces différents algorithmes de simulation, et les différents modes de synchronisation sont présentées dans [ANS88].

2.3 Caractéristiques générales de la machine

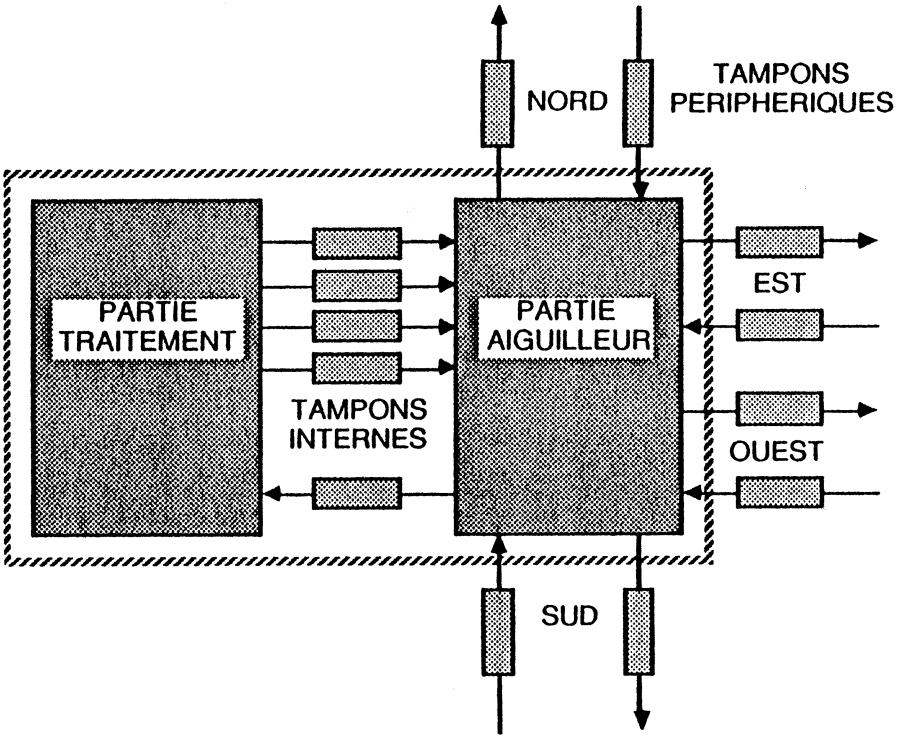
Nous avons implémenté dans la partie-traitement des cellules, l'algorithme de simulation logique à délai unitaire à trois états (0, 1 et X), la simulation étant synchronisée au niveau global par l'ordinateur-hôte gérant les signaux DEBUT et FIN.

2.3.1 Architecture de la machine

Certaines caractéristiques des portes ont des répercussions directes sur l'architecture. Les fonctions logiques implémentées sont associatives. Cela permet de réaliser la fonction logique de la porte au fur et à mesure de la réception des entrées. Il n'est donc pas nécessaire de mémoriser toutes les entrées de la porte pour calculer, en une seule opération, le résultat de sortie de la cellule. Un seul tampon d'entrée suffit donc à la partie-traitement quelle que soit l'entrance maximum de la porte simulée.

Par contre, il est nécessaire de disposer d'un nombre de tampons de sortie au moins égal à la sortance maximale des portes afin de mémoriser tous les messages de sortie à émettre.

Notre cellule aura donc cinq tampons internes : un d'entrée et 4 de sortie pour la partie-traitement.



Architecture générale de la cellule

2.3.2 Codage des messages

Codage de l'information

L'algorithme implémenté autorise une simulation logique à trois états (vrai, faux et inconnu). Ces états sont codés sur deux bits dans le champ-donnée du message. Initialement, les messages d'accusés de réception étaient codés par la quatrième valeur possible.

Codage des déplacements

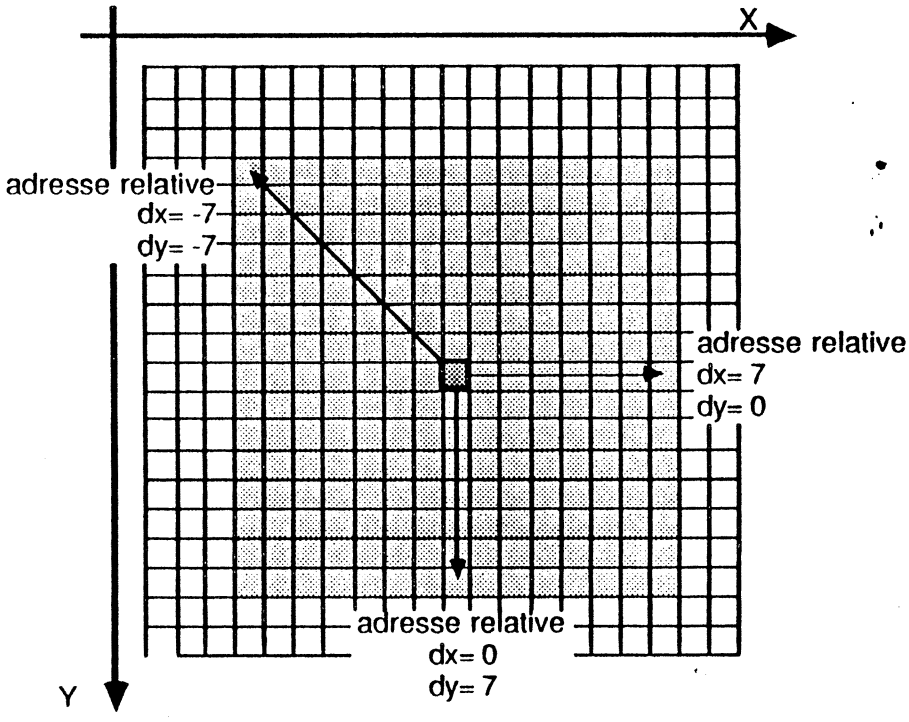
La taille du champ-routage des messages dépend directement de la taille du réseau.

Pour simplifier l'algorithme de routage, les déplacements relatifs dx et dy sont représentés dans les messages sous une forme signée : signe et valeur absolue du déplacement.

Soit n le nombre de bits servant à mémoriser le déplacement, le champ-routage des messages comportera 2.(n+1) bits (n+1 bits pour mémoriser chacun des déplacements). Une cellule pourra

alors communiquer directement avec $(2^{n+1}-1)^2$ cellules. Celles-ci sont organisées autour d'elle en un carré de 2^{n+1} -1 cellules de coté.

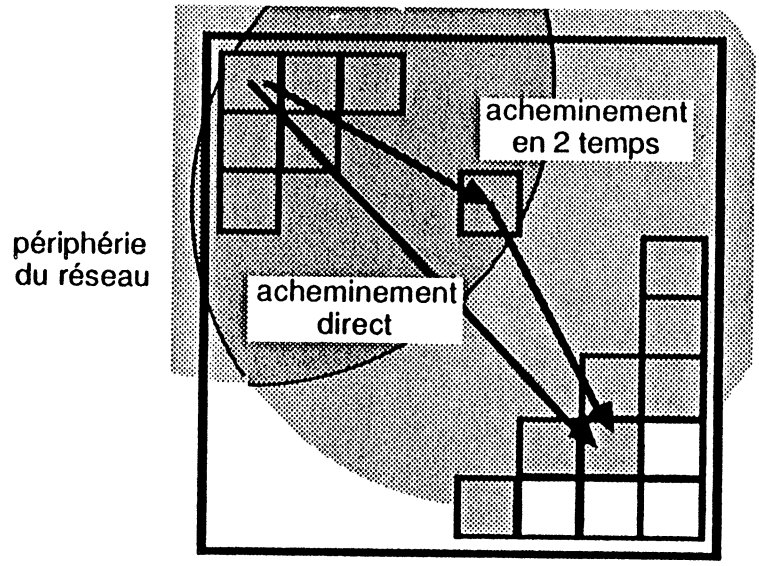
Dans l'exemple ci-dessous ou n=3, une cellule peut envoyer un message à 225 autres cellules (y compris elle-même).



Rayon d'action d' une cellule (n=3)

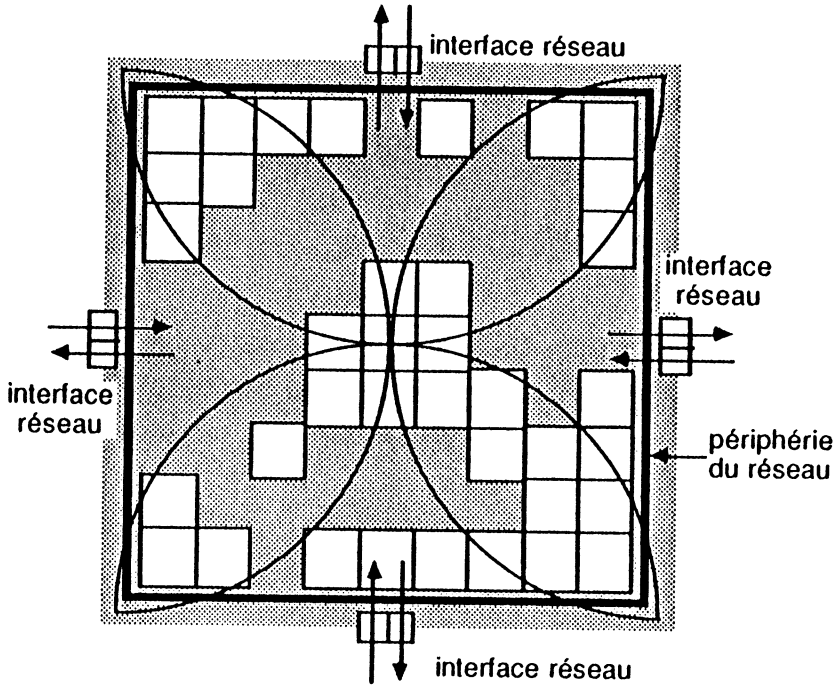
Dimension du réseau et taille des messages

D'un point de vue communication, il est nécessaire qu'une cellule puisse être initialisée, donc accessible **directement** à partir d'un tampon périphérique au réseau. Il est également indispensable pour n'importe quelle cellule de pouvoir communiquer avec n'importe quelle autre. Par contre, il n'est pas nécessaire que les informations contenues dans le champ routage des messages leur permettent de couvrir la totalité du réseau, car la localité des différents algorithmes de simulation étudiés liée à l'interconnexion des éléments logiques est bonne. Dans les cas critiques (grande distance à parcourir), une cellule intermédiaire peut être programmée pour réacheminer le message.



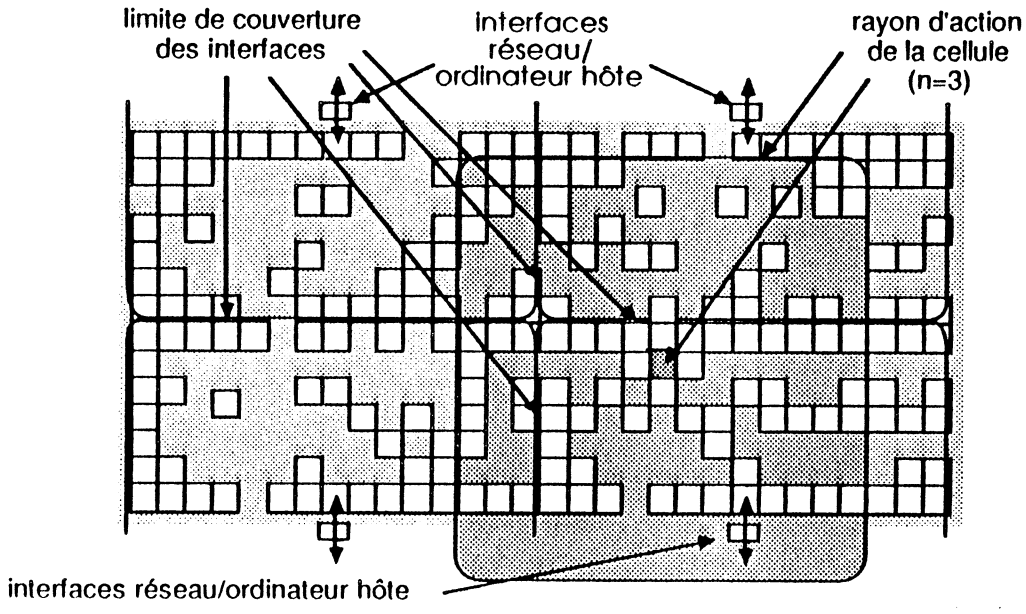
Acheminement direct et indirect

Lors de l'initialisation, on ne peut se servir de cellules relais (elles ne sont pas encore programmées), néanmoins, en répartissant plusieurs cellules d'interface à la périphérie du réseau on peut programmer des réseaux dont la taille est supérieure au rayon d'action des cellules.



Initialisation d'un réseau à partir des tampons d'interface

Dans l'exemple ci-dessous, un réseau 30x14 (420 cellules) est programmé à travers quatre tampons d'interface périphériques. Le rayon d'action des cellules est limité (valeur absolue du déplacement sur trois bits).



Réseau 30x14 programmé à travers 4 tampons d'interface

Dimensions retenues

Le premier prototype de machine cellulaire pour la simulation logique comporte une centaine de cellules (réseau 10x10). Ce prototype est réalisé à partir d'un circuit élémentaire intégrant un réseau 2x2 et des interfaces. Un tel réseau permet une bonne évaluation aussi bien des performances du simulateur (cent portes au maximum), que des performances du réseau proprement dit : mécanisme d'acheminement, phase d'initialisation... Le champ routage est codé sur huit bits, quatre pour chacun des déplacements (un bit de signe et trois bits pour la valeur absolue du déplacement relatif). La taille définitive des messages est donc de 10 bits (2x4 + 2).

Remarque : le nombre de tampons de sortie réels du réseau interfacés avec l'ordinateur-hôte limite la sortance maximum du circuit à simuler. En effet, l'ordinateur-hôte doit pouvoir identifier l'origine de

tous les messages qu'il reçoit, or cette information n'est pas contenue dans les messages. On aura donc toujours intérêt à disposer du plus grand nombre possible d'interfaces de sortie au réseau.

2.4 La cellule

2.4.1 Partie-aiguilleur

L'étude fonctionnelle, les algorithmes d'acheminement et la réalisation de cette partie sont décrits dans le chapitre 3.

2.4.2 Partie-traitement

Le rôle de la partie-traitement de la cellule est de simuler le fonctionnement d'une porte logique. C'est dans cette partie qu'est implémenté l'algorithme réparti de simulation logique.

Les parties-traitement des cellules doivent mémoriser les paramètres associés aux caractéristiques des portes, c'est-à-dire : la fonction logique, l'inversion ou non de la sortie, le retard entraîné, l'entrance, la sortance et enfin les adresses des cellules connectées en sortie.

La partie-traitement est donc architecturée autour :
- d'éléments mémorisant les paramètres de la porte,
- d'une UAL réalisant la fonction logique associée à la cellule,
- d'un élément gérant le retard de la sortie.

La gestion des retards est réalisée par un registre à décalage (FIFO), celui-ci mémorise la sortie évaluée pendant 1, 2 ou 3 pas de simulation (décalages). Le ou les messages de sortie seront alors acheminés par la partie-aiguilleur de la cellule.

La fonction logique réalisée par la cellule étant associative, l'évaluation de la sortie est effectuée au fur et à mesure de la réception des messages d'entrée. Cette opération nécessite un compteur d'entrée.

CHAPITRE 2

REALISATION DU CIRCUIT

TABLE DES MATIERES

1 METHODE DE CONCEPTION

1.1 Le cahier des charges

Il dresse les contraintes à respecter au cours de la réalisation du circuit. Elles sont de quatre ordres :

- contraintes fonctionnelles
- contraintes architecturales
- contraintes technologiques
- contraintes conceptuelles

Contrainte fonctionnelle

Il s'agit de réaliser un circuit formé de cellules asynchrones organisées en réseau et simulant le fonctionnement d'une porte logique. L'ensemble des contraintes fonctionnelles a été développé au chapitre précédent.

Contrainte architecturale

Elles sont liées à l'organisation en réseau des cellules. Celles-ci doivent pouvoir être juxtaposées sans problèmes (organisation topologique en matrice), afin de former un réseau dont la taille n'est pas encore connue puisque celle-ci dépend étroitement des contraintes technologiques et conceptuelles.

La cellule est architecturée autour de trois blocs fonctionnels : l'aiguilleur, la partie-traitement et les tampons (périphériques et internes). L'échange de données entre ces blocs est assuré par un bus.

Contraintes technologiques

Le circuit doit être réalisé en technologie CMOS, à un niveau de métal et une largeur de grille de deux microns. Il s'agit de la technologie CMOS1L2 développée au CNET de Meylan. Le choix de la technologie a des conséquences directes sur la vitesse, l'organisation topologique, la taille, la consommation et le coût du circuit.

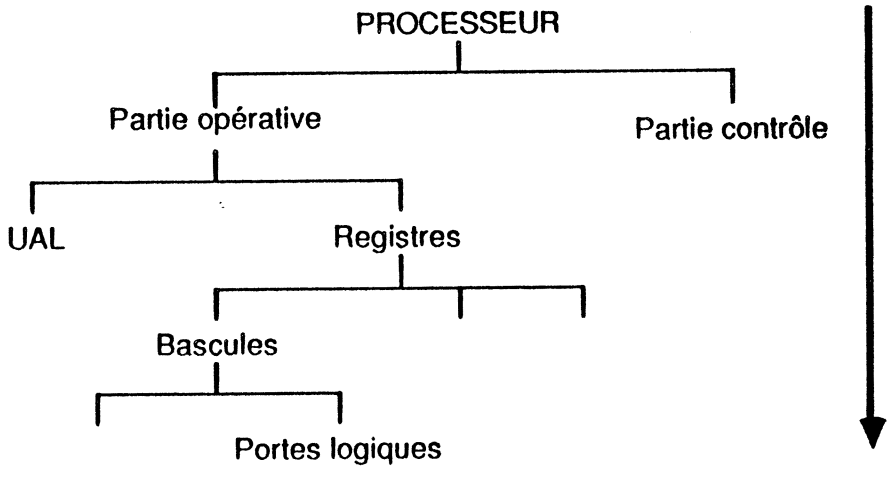
Contrainte conceptuelle

Le choix de la technologie, s'il est arrêté pour la réalisation de ce circuit, n' est pas définitif, notamment pour les développements à venir. De ce fait, la réalisation du circuit doit être la plus souple possible, et dans la mesure du possible, paramétrable.

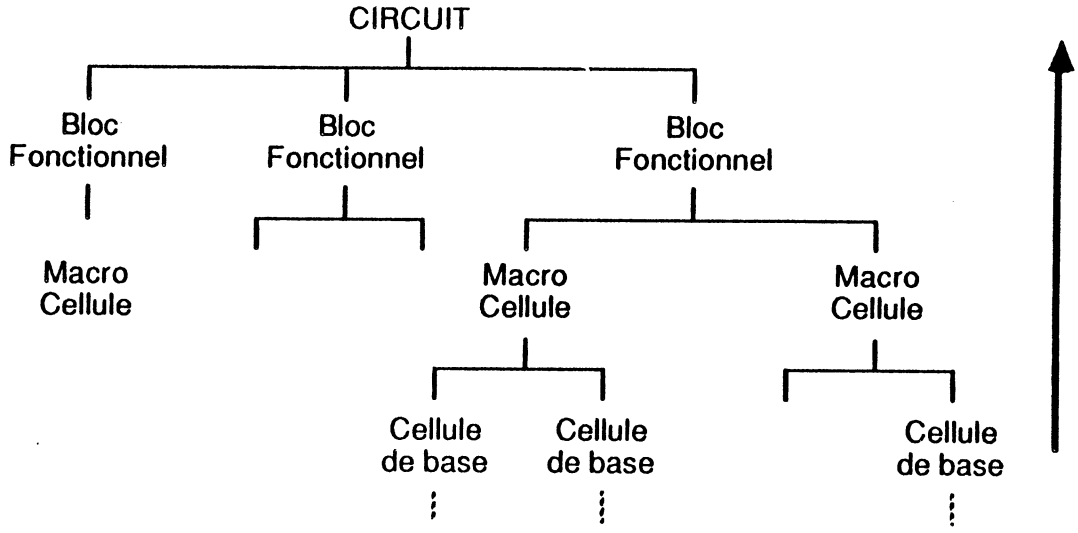
Nous avons choisi de concevoir le circuit à l'aide d'une bibliothèque de cellules. Il est évident que ce choix, s'il simplifie la conception et la réalisation de circuit VLSI, entraîne en contrepartie une densité d'intégration moindre, et donc une plus grande surface de silicium que les circuits conçus "à la main". En fait, la médiocre densité de notre circuit est surtout due aux canaux de routage, que le seul niveau de métal ne permet pas de simplifier.

1.2 Méthode de conception

L'utilisation d'une bibliothèque est intéressante, car elle permet la simplification de la conception et de la réalisation des blocs fonctionnels. La conception est descendante, alors que la réalisation est ascendante.



Conception descendante



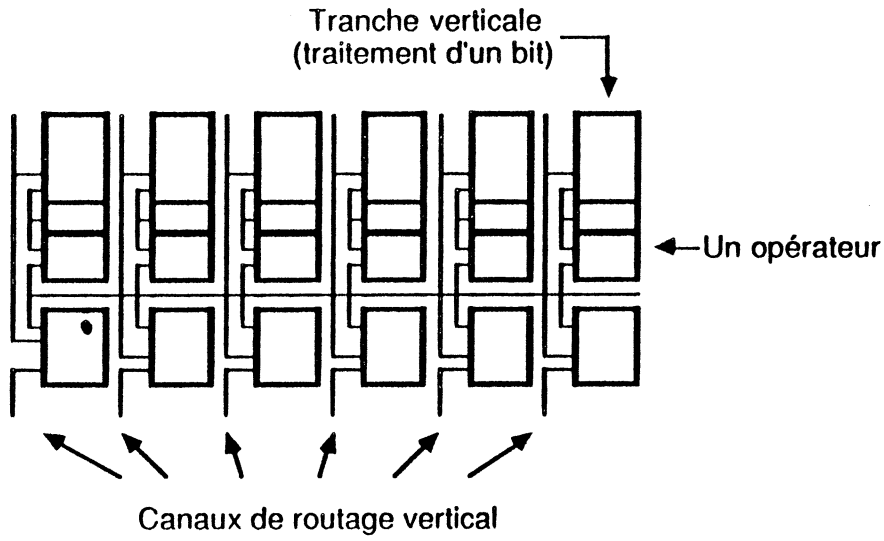
Réalisation ascendante du circuit

Mais la contrepartie de l'utilisation d'une bibliothèque de cellules est un ensemble de règles à respecter dans la conception des cellules de base, ainsi que dans l'implantation de l'architecture des blocs fonctionnels.

1.2.1 Les structures régulières

Les fonctions à implanter doivent être pensées en termes de tranches de bits. Les cellules élémentaires, ayant la même largeur, peuvent s'empiler afin de réaliser les différents opérateurs d'une même tranche de bit. De part et d'autre, des canaux de routage assurent l'interconnexion des cellules.

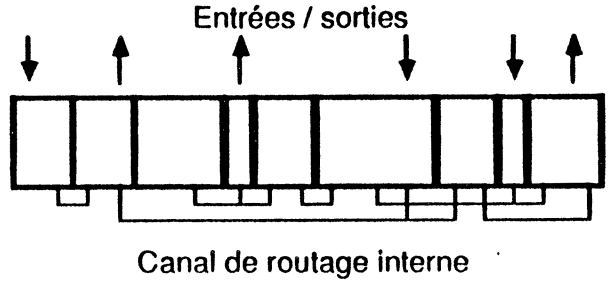
Les alimentations et les entrées/sorties du flot de données transitent verticalement en métal, les commandes sont acheminées horizontalement en métal dans la mesure du possible sinon en silicium polycristallin lors de la traversée des canaux de routage.



Exemple d'implantation de structures régulières

1.2.2 Les structures irrégulières

Ce sont les blocs de logique aléatoire. Ils doivent être implantés "en colonne" (ou en ligne). Tous les éléments logiques sont empilés sur une colonne (ou deux mais rarement plus), et le canal de routage occupe l'un des deux côtés, l'autre étant réservé au canal de routage des entrées/sorties du bloc.



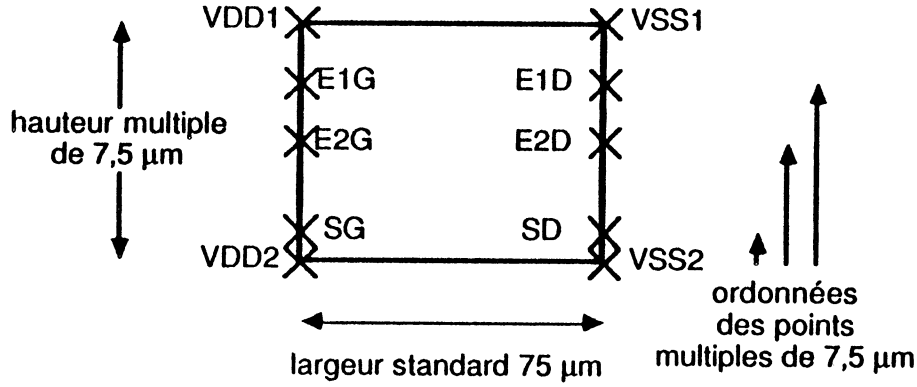
Exemple d'implantation de logique aléatoire

1.2.3 La bibliothèque de cellules

Elle comporte une vingtaine d'éléments de base dont la réalisation est liée à la technologie. La conception de ces cellules respecte des contraintes propres à la bibliothèque.

Les cellules ont une largeur standard de 75 µm. L'enveloppe de ces cellules est un rectangle de largeur standard et de hauteur

quelconque mais multiple de 7,5 µm. Les points particuliers d'entrée/sortie sont, dans la mesure du possible, accessibles sur les côtés droit et gauche (frontières est et ouest) de l'enveloppe, aux mêmes ordonnées, toujours sur un pas de 7,5 µm. On doit toujours pouvoir faire passer un fil à un pas de l'enveloppe sans violer de règles de garde. Les alimentations sont disponibles dans les coins de l'enveloppe supérieur et inférieur gauche pour VDD, et droit pour VSS. Les cellules doivent pouvoir être empilées, sans violer de règles de garde tout en assurant la continuité électrique des alimentations.



Contraintes sur les cellules

Les cellules de la bibliothèque et leurs caractéristiques électriques sont données en annexe.

1.3 Les outils de CAO

1.3.1 La première version du circuit

Au cours de l'année 1987, a été conçue, fabriquée et testée une première version de ce circuit qui ne comprenait qu'une seule cellule. Cette cellule a été presque entièrement réalisée à l'aide d'outils de conception développés au CNET de Meylan : Cassiopé (V2.1) [LEC82] pour le système de CAO, et LOF (V3) [BER84] pour l'assemblage des cellules.

LOF (pour Langage d'Opérateurs Flexibles) est un ensemble de logiciels permettant d'écrire des générateurs de circuits intégrés.

Il s'est avéré que la réalisation de notre cellule ne correspondait pas à un usage optimum de LOF. Comme on le verra plus loin dans ce chapitre, une grande partie de la cellule n'est pas régulière, et sa réalisation en un seul exemplaire ne permettait pas d'utiliser toute la puissance de ce logiciel.

Le dessin des masques de ce circuit est donné en annexe.

1.3.2 La version actuelle du circuit

Nous avons conçu et réalisé le circuit à l'aide du logiciel Silvar-Lisco, installé sur une station de travail Vax-Station GPX II. La configuration matérielle de celle-ci comportait 5 méga-octets de RAM, et 2 disques durs d'un total de 450 méga-octets.

Silvar Lisco est formé d'un ensemble ouvert d'outils regroupant tous les aspects de la CAO.
Nous avons notamment utilisé :

- → les systèmes de saisie et de gestion de schémas SDS (Structured Design System) et de dessin de masques PRINCESS
- → le simulateur logique BIMOS
- → une partie des logiciels de vérification de layout du système DVS (Design Verification System).

Les simulations électriques ont été réalisées avec ELDO [HEN87], logiciel développé par le CNET de Meylan. La génération finale automatique des masques du circuit a été faite au CNET par le logiciel Dracula.

LIEU	ETAPES	LOGICIEL
USSI/INPG	Validation fonctionnelle Saisie des schémas Validation logique Validation électrique Dessin des masques Vérification des règles de garde Vérification des régles électriques Validation électrique du layout (extraction netslist + simulation)	Langage parallèle OCCAM Silvar Lisco - SDS Silvar Lisco - Bimos Eldo Silvar Lisco - Princess Silvar Lisco - UDRC Silvar Lisco - UERC Silvar Lisco - USNG + Eldo
CNET Meylan	Validation finale du layout Génération des masques	Dracula - DRC/ERC Dracula
CNET Meylan	Fabrication	
CIME/INPG	Tests	

Organigramme de la conception du circuit

1.4 Les simulations

1.4.1 Validation fonctionnelle

Il s'agit de la validation de l'architecture générale du réseau asynchrone communiquant par messages. La nature hautement parallèle et asynchrone de cette architecture, nous a conduits à décrire et à simuler son fonctionnement à l'aide du langage de programmation parallèle OCCAM [INM84].

Cette simulation a notamment permis de tester et de mettre au point plusieurs algorithmes d'acheminement de messages, ainsi que les différents modes de synchronisation envisagés. L'architecture est décrite de façon très simple en OCCAM. A chaque bloc fonctionnel (réseau, cellule, partie-aiguilleur, partie-traitement, tampons) est associé un processus OCCAM, respectant ainsi le caractère asynchrone de ceux-ci [OBJ87a]. Les différentes interconnexions sont naturellement représentées par les canaux de communication OCCAM. Ces simulations sont présentées dans [ANS88].

1.4.2 Validation logique

La première version du circuit a été, d'un point de vue logique, décrite et validée à l'aide du logiciel EPILOG [EFC86]. Pour la deuxième version nous n'avons re-simulé avec BIMOS [BIM85] que les fonctions logiques ayant été modifiées.

1.4.3 Validation électrique

L'utilisation d'une bibliothèque de cellules aux caractéristiques électriques connues permet de simplifier la validation électrique des circuits synchrones, en limitant les simulations aux problèmes temporels critiques. C'est ce que nous avons fait pour la première version du circuit. Pour la seconde, nous avons pu, avec Silvar-Lisco, simuler électriquement non pas une représentation schématique, mais le circuit réel tel qu'il est implanté sur le silicium. Cet outil (USNG) - simulation électrique (Eldo) d'une description électrique (netlist) extrait du dessin (layout) - permet d'appréhender de manière très précise les problèmes électriques réels.

Nous nous sommes rendu compte que l'horloge de la cellule, déterminée en fonction des contraintes plus strictes de la partie-traitement, ralentissait inutilement la partie-aiguilleur. Nous avons donc pu rendre complètement indépendantes ces 2 parties par le choix de 2 horloges distinctes et adaptées à leurs contraintes respectives. Les simulations sont présentées en annexe.

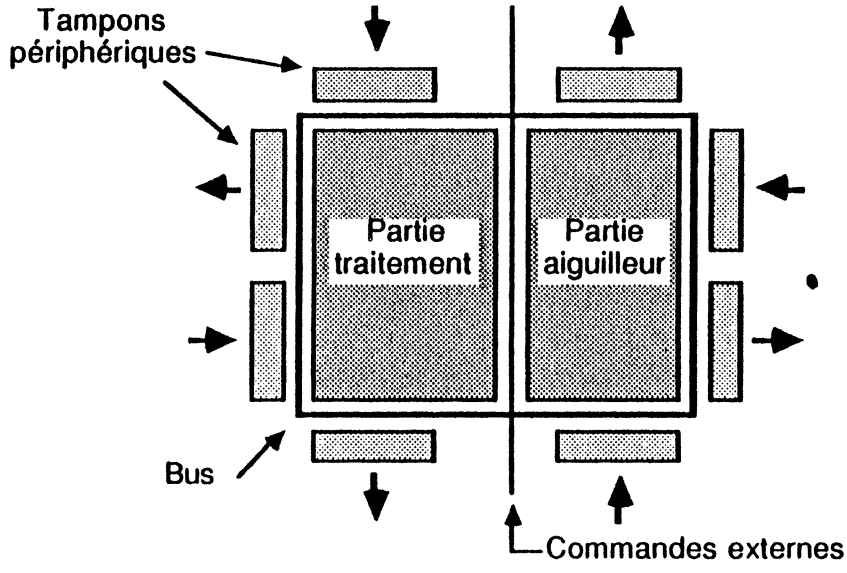
2 LA CELLULE

2.1 Etude fonctionnelle

Une cellule est composée de trois parties fonctionnellement distinctes : la partie-aiguilleur, la partie-traitement et les tampons.

Les parties-aiguilleur et traitement déroulent un algorithme synchrone, mais de manière indépendante. Les tampons fonctionnent également de façon synchrone, mais selon leur position, ils sont séquencés par deux aiguilleurs différents (tampons périphériques), ou par l'aiguilleur et la partie-traitement d'une même cellule (tampons internes).

Les communications se font toujours à travers les tampons : aiguilleur ↔ tampons et tampons ↔ partie-traitement. Les données (dix bits de message) sont échangées à travers un bus. Différentes implantations de celui-ci ont été étudiées, mais l'organisation en réseau des cellules imposant des tampons périphériques sur les quatre frontières de la cellule, nous avons implanté un bus périphérique.



Organisation générale de la cellule

2.2 Mémorisation des messages

Le rôle des tampons est d'assurer la mémorisation des messages (données et routage) lors de leur acheminement. Les tampons sont donc les points de communication obligatoires entre deux cellules voisines ainsi qu'entre les parties-aiguilleur et traitement d'une même cellule.

2.2.1 Etude fonctionnelle des tampons

Fonctionnellement, un tampon est une interface unidirectionnelle entre un producteur et un consommateur. La difficulté, ici, vient du fonctionnement totalement asynchrone de ces deux parties. Le transfert de l'information contenue dans un message doit donc se faire en deux temps :

① écriture dans le tampon (émetteur → tampon)
② lecture du tampon (tampon → récepteur)

Les protocoles de communication reposent sur plusieurs remarques :

- ✎ l'état d'un tampon doit pouvoir être interrogé à tout moment.
- ✎ on ne peut remplir un tampon (écriture) que s'il est vide.
- ✎ on ne peut vider un tampon (lecture avec acquisition) que s'il est plein.
- ✎ pour éviter qu'un tampon ne soit prématurément lu (resp. réécrit) un tampon ne doit être marqué "plein" (resp. "vide"), qu'une fois achevée l'opération d'écriture (resp. lecture avec acquisition).
- ✎ on peut lire un message dans un tampon sans l'acquérir (dans le cas d'une simple consultation).
- ✎ par contre, suite à une écriture, un tampon doit être marqué "plein" (il y a modification du contenu du tampon).

Afin d'éviter toute erreur lors des communications émetteur/tampon et tampon/récepteur, il est nécessaire d'assurer l'exclusion mutuelle du tampon entre l'émetteur et le récepteur.

Ces remarques mettent en évidence les signaux de contrôles nécessaires au transfert des messages :

- ❑ **vide** c'est l'état du tampon adressé à l'émetteur.
- ❑ **plein** c'est l'état du tampon adressé au récepteur.
- ❑ **remplir** cette commande envoyée par l'émetteur, immédiatement après la commande d'écriture, mettra à jour l'état du tampon (état := plein).
- ❑ **vider** cette commande envoyée par le récepteur, immédiatement après la commande de lecture (avec acquisition de l'information), mettra à jour l'état du tampon (état := vide).
- ❑ **écrire** ce signal contrôlé par l'émetteur chargera effectivement le message dans le tampon.
- ❑ **lire** ce signal contrôlé par le récepteur lui permet de lire le message du tampon. La lecture peut être une interrogation ou une acquisition des informations contenues dans le tampon. Dans ce dernier cas, la commande de 'lecture' sera suivie de la commande 'remplir'.

Un tampon est donc constitué de deux parties distinctes. L'une assure la mémorisation des informations proprement dites, l'autre de son état :

'plein' s'il contient un message,
'vide' s'il n'en contient pas.

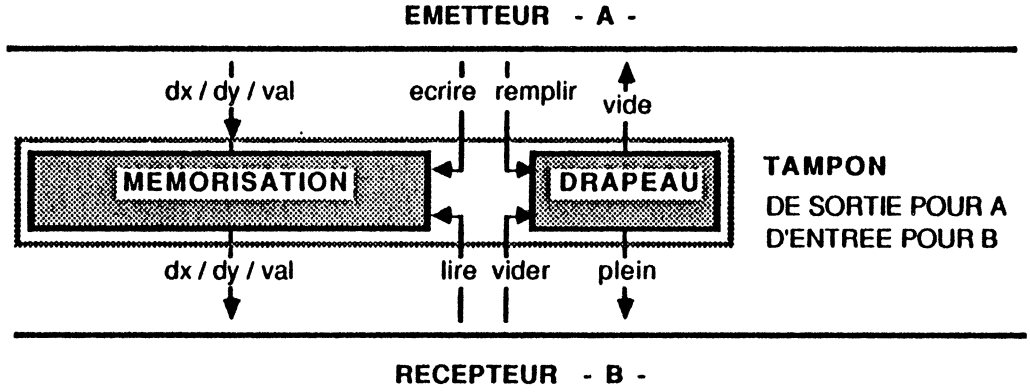


Schéma fonctionnel d'un tampon

2.2.2 Réalisation logique

2.2.2.1 La partie-mémorisation

Le tampon prend et rend ses informations sur les bus périphériques des cellules (ou d'une même cellule). Ce bus étant une ressource commune aux autres tampons, il doit pouvoir être mis par eux en état de haute impédance, lorsqu'ils n'y accèdent pas. La mémorisation d'un bit d'information est assurée par un registre. Le signal en entrée de celui-ci est régénéré par un inverseur et sa sortie reliée au bus à travers une porte trois états. C'est donc la valeur inversée qui est mémorisée.

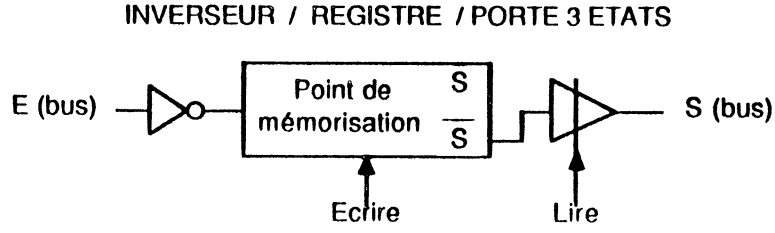


Schéma logique d'un élément de mémorisation

2.2.2.2 Le drapeau

L'exclusion mutuelle sur le tampon implique le non recouvrement des signaux 'plein' et 'vide'. La mémorisation du drapeau est basée sur une bascule RS. Un mécanisme combinatoire en porte NOR assure non seulement le non-recouvrement des signaux d'état, mais également leur invalidation lors des opérations de mise à jour. Le drapeau n'est plein qu'à la fin de la commande 'remplir', de même, il n'est vide qu'à la fin de la commande 'vider'. Au cours de ces deux opérations, il est considéré comme ni plein (plein=faux) et ni vide (vide=faux) - voir également le paragraphe sur les simulations -.

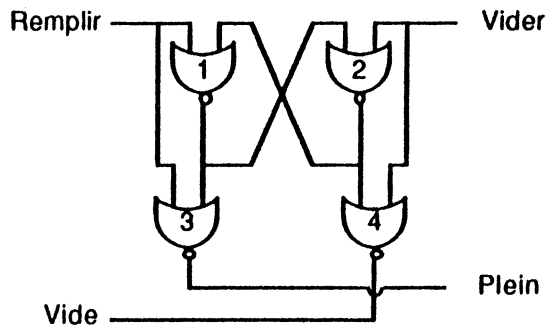
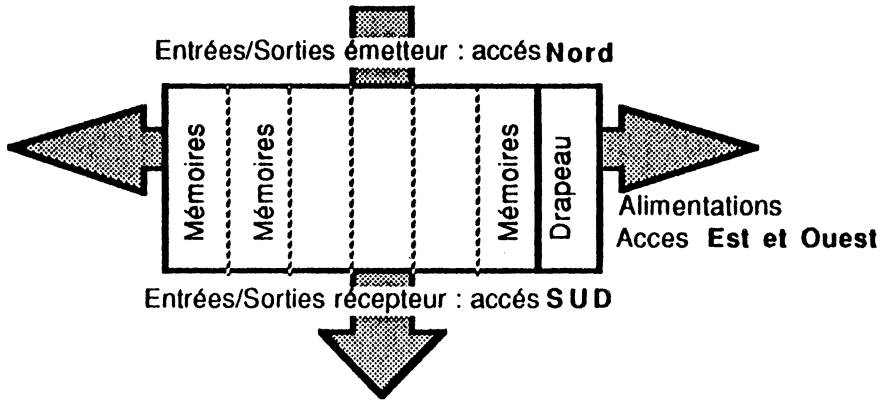


Schéma logique d'un drapeau

2.2.3 Dessin des masques des tampons

2.2.3.1 Les tampons périphériques

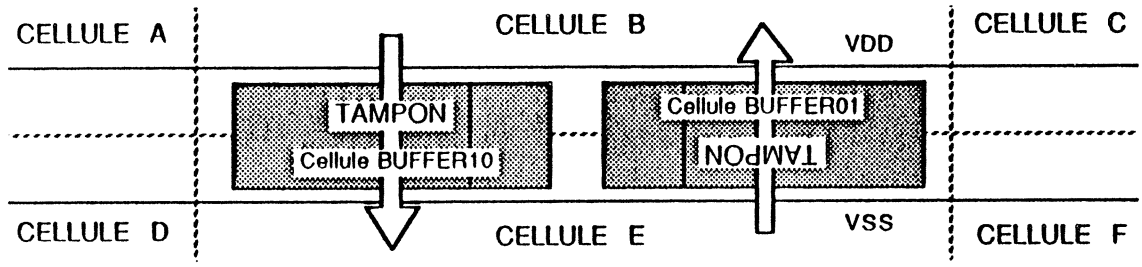
Les tampons périphériques mémorisent des messages de dix bits (huit pour le routage et deux pour les données). Ils sont formés de dix éléments de mémorisation en parallèle et d'un drapeau. La fonctionnalité des tampons ainsi que les contraintes de taille sur la cellule conduisent à une implantation régulière de l'architecture (type en tranche). Les entrées des éléments de mémorisation, les signaux 'remplir' et 'vide' sont accessibles au nord (côté émetteur; les sorties, les signaux 'vider' et 'plein' le sont au sud (côté récepteur), enfin les peignes des alimentations VDD et VSS (nord/sud) traversent le tampon d'est en ouest.



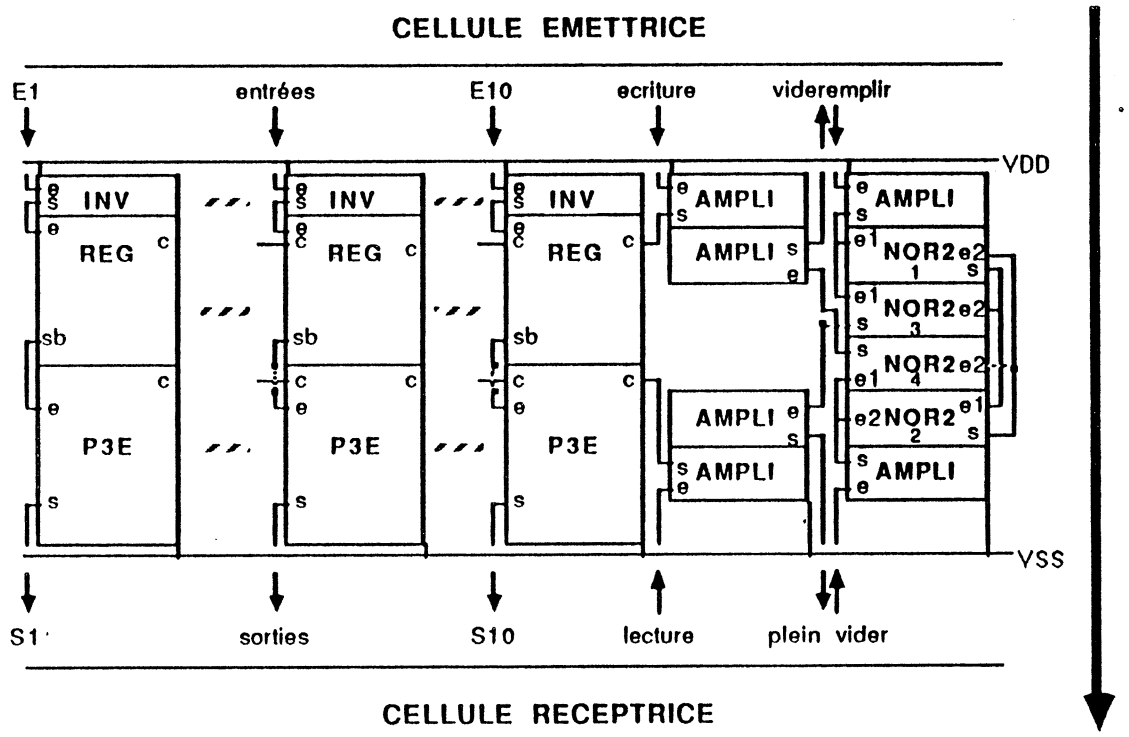
Architecture générale d'un tampon périphérique

Nous avons réalisé, en fait, deux cellules différentes pour des problèmes d'alimentation du réseau. En effet les deux tampons périphériques d'une même frontière, qui sont identiques, sont symétriques en X et en Y; leurs alimentations sont donc inversées.

La cellule BUFFER10 regroupe sur sa frontière nord VDD, et au sud la masse; inversement, la cellule BUFFER01 est encadrée au nord par VSS et au sud par VSS.



Remarques : les entrées/sorties du drapeau sont amplifiées car reliées aux parties-aiguilleur des cellules topologiquement éloignées.



Implantation des tampons périphériques

Densité

Le tampon comporte 280 transistors, son enveloppe est un rectangle de 1102 µm par 248 µm soit 0,274 mm². La densité d'intégration est donc de 1021 tr/mm².

2.2.3.2 Les tampons internes

Leur implantation est identique aux tampons périphériques. Mais le nombre de bits à mémoriser diffère selon qu'il s'agit des tampons d'entrée ou de sortie de la partie-traitement de la cellule. Dans le premier cas, le message étant arrivé à destination, on ne stocke que sa donnée (deux bits), dans le dernier cas, on ne mémorise qu'une seule fois le résultat de la fonction logique réalisée par la partie-traitement de la cellule pour les quatre tampons de sortie. Pour des problèmes de connectique les parties-mémorisation et les drapeaux des quatre tampons de sortie ont été implantés d'une manière éclatée dans la cellule.

2.3 La partie-aiguilleur

Le rôle de l'aiguilleur est d'acheminer les messages entre :

- **8 entrées**
  - → 4 internes (tampons de sortie de la partie-traitement)
  - → 4 externes (tampons périphériques d'entrée de la cellule)
- **5 sorties**
  - → 1 interne (tampon d'entrée de la partie-traitement)
  - → 4 externes (tampons périphériques de sortie de la cellule)

Le choix de l'entrée prise en compte par l'algorithme 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.

2.3.1 Etude fonctionnelle

L'organigramme simplifié de l'aiguilleur se résume à :

① choisir un message à transmettre (parmi ceux en attente)
② déterminer le tampon de sortie de celui-ci (selon une politique d'acheminement prédéterminée)
③ dans le cas où ce dernier est vide, transférer effectivement le message
④ recommencer (retour en ①)

Pour être opérationnel cet algorithme doit tenir compte des points suivants :

- toutes les demandes en entrée de l'aiguilleur doivent être traitées en un temps fini (il ne doit pas y avoir de problème de famine) ;
- l'aiguilleur ne doit jamais être bloqué par l'attente de la libération d'un tampon de sortie ;
- on ne peut choisir qu'une seule entrée à la fois, que l'on acheminera vers un tampon de sortie et un seul.

2.3.1.1 Architecture de l'aiguilleur

Logiquement, l'architecture de l'aiguilleur va s'organiser autour de deux blocs :
- **l'encodeur de priorité** spécialisé dans le choix du message d'entrée à transmettre,
- **la partie-traitement-aiguilleur** chargée de déterminer le tampon de sortie vers lequel l'acheminer.

Le contrôle du transfert est séquencé par un PLA (Programmable Logic Array). Une autre façon de faire aurait été de développer un aiguilleur entièrement combinatoire. Ce choix, qui a fait l'objet d'une étude particulière [ELK87], n'a pas été retenu en raison de son côté non reconfigurable - donc non paramétrable - en vue d'une implantation automatisée.

L'interface entre l'encodeur de priorité et la partie-traitement de l'aiguilleur est un registre interne servant à mémoriser le message en cours de transfert.

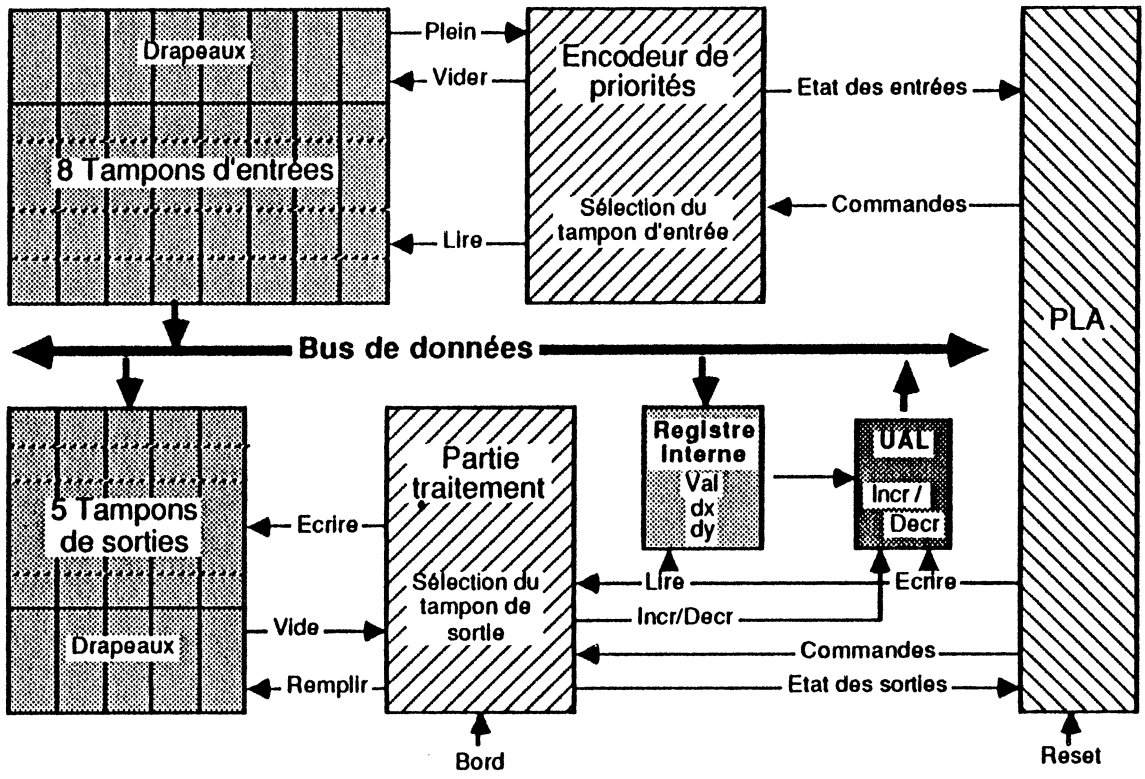


Schéma fonctionnel de l'aiguilleur

2.3.1.2 L'encodeur de priorité

Le problème de famine risque de se poser lorsqu'une entrée, plus prioritaire que les autres et ayant sans arrêt des messages à transmettre, empêche l'aiguilleur de traiter les demandes moins prioritaires qui sont donc bloquées.

Pour éviter ce blocage, nous avons développé un mécanisme de priorités tournantes :
lorsque l'on a sélectionné une entrée pour en étudier la demande d'acheminement, on s'interdit de reprendre cette entrée en compte, tant que toutes les éventuelles demandes de priorités inférieures n'ont pas été envisagées, que l'entrée en question ait été acheminée ou pas. L'aiguilleur sélectionne toutes les entrées après échantillonnage, à travers un filtre de prise en compte.

Formalisation

On appelle plein(i), les valeurs échantillonnées des signaux d'état des tampons d'entrée et pc(i) leur code de prise en compte :

- plein(i)=vrai si le tampon de priorité i demande à transmettre un message, sinon la variable est à faux,
- pc(j)=vrai si l'on autorise la demande de priorité j à être prise en compte, sinon pc(j)=faux.

Algorithme de sélection des entrées

```
initialiser les prises en compte
échantillonner les entrées
tant que vrai faire
    i:= 1;
    répéter
        i:= i+1;
    jusqu'à [plein(i-1)= vrai) et pc(i-1)= vrai] ou [i>8]
    si [plein(i-1)= vrai) et pc(i-1)= vrai]
    alors
        pc(i-1):= faux;
        si acheminement possible
        alors
            transférer le contenu du tampon i dans
            le registre interne de l'aiguilleur;
        fin si
    fin si
    si i>8
    alors
        initialiser les prises en compte;
    fin si
    échantillonner les entrées;
fin tant que
```

En plus du choix de l'entrée, l'encodeur de priorité se charge de la mise à jour du drapeau de cette entrée lorsque l'acheminement est effectué.

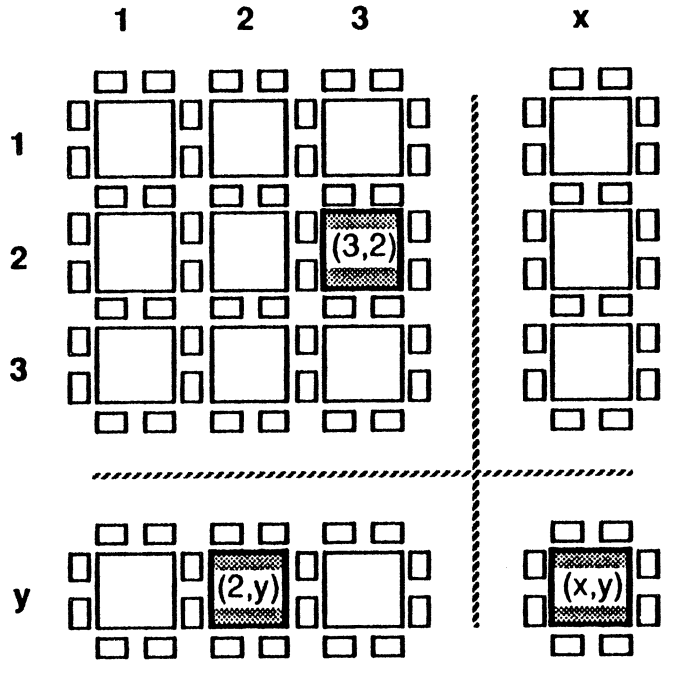
2.3.1.2 La partie-traitement de l'aiguilleur

Le message d'entrée sélectionné ayant été transféré dans le registre interne de l'aiguilleur, il s'agit maintenant de déterminer le

tampon de sortie vers lequel l'acheminer, et s'il est vide d'y transférer le message, après l'avoir mis à jour.

Codage des cellules sur le réseau

Les cellules sont repérées sur le réseau par leurs coordonnées absolues, et les messages à acheminer caractérisés par leur déplacement relatif.



Repérage des cellules sur le réseau

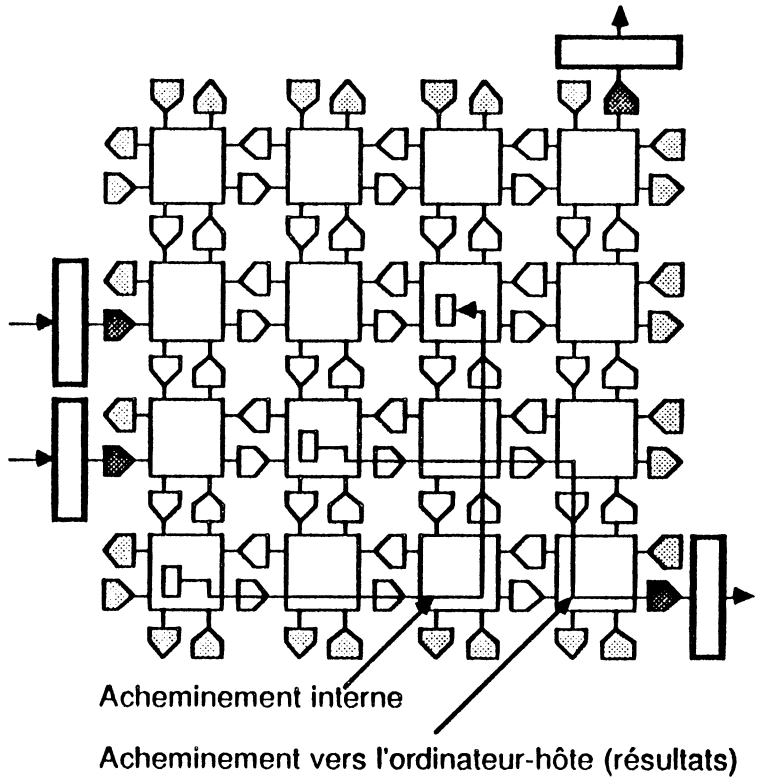
Algorithmes de la partie-traitement-aiguilleur

L'algorithme de routage implanté achemine d'abord les messages selon l'horizontale (axe des x), puis selon la verticale (axe des y).

Deux cas particuliers se posent lorsque le message arrive sur les frontières est ou ouest du réseau (cas d'une sortie du réseau). Seuls, certains tampons périphériques du réseau sont câblés pour communiquer avec l'ordinateur-hôte; il faut donc absolument passer par eux pour sortir du réseau, sinon le message sera acheminé dans un tampon d'où il ne pourra jamais sortir (cul-de-sac).

Pour gérer ces sorties, on dispose dans chaque cellule des signaux **'bord-est'** et **'bord-ouest'** indiquant que celle-ci se

trouve sur la première ou la dernière colonne de la matrice du réseau.



Exemples d'acheminements dans le réseau

Remarque : il n'y a pas de 'bord-nord', ni de 'bord-sud', l'algorithme privilégiant l'acheminement horizontal.

Formalisation : soit DX et DY les valeurs du champ-routage, et sortie la variable résultat contenant le tampon de sortie :

Algorithme d'acheminement

La description séquentielle suivante ne reflète pas la façon dont est implémenté cet algorithme, celle-ci se rapprochant plus d'une table de décision.

```
si DX>0
alors
    si bord-est = vrai (frontière est du réseau)
    alors
        si DY>0
        alors sortie = sud
        sinon
            si DY<0
            alors sortie = nord                    | on est ici devant un
            sinon (DY=0) sortie = est    <=        | un tampon d'interface
    sinon sortie= est                              | de sortie du réseau
sinon
    si DX<0
    alors
        si bord-ouest = vrai (frontière ouest du réseau)
        alors
            si DY>0
            alors sortie = sud
            sinon
                si DY<0                            | on est ici devant
                alors sortie = nord                | un tampon
                sinon (DY=0) sortie = ouest <=     | d'interface de
        sinon sortie = ouest                       | sortie du réseau
    sinon (DX=0)
        si DY>0
        alors sortie = sud
        sinon
            si DY<0
            alors sortie = nord
            sinon sortie = partie-traitement-cellule
```

La mise à jour du champ-routage est directement fonction du tampon de sortie. Elle consiste toujours en la décrémentation de la

coordonnée relative au déplacement, sauf dans le cas où le message est arrivé à destination (DX=DY=0). Dans ce cas, il concerne donc la partie-traitement de la cellule et seul le champ-information du message est transmis.

Algorithme de mise à jour du champ-routage

cas sortie **de**
| nord, sud : DY:= DY-1
| est, ouest : DX:= DX-1
| partie-traitement cellule :
fin cas

Tous ces algorithmes ne prennent leurs sens qu'une fois insérés dans le séquencement de l'aiguilleur.

2.3.1.3 Séquencement de l'aiguilleur

L'encodeur de priorité et la partie-traitement travaillent en parallèle et émettent vers le séquenceur les comptes-rendus de leurs activités :

- **entrée-pleine** indique qu'une entrée a été sélectionnée par l'encodeur
- **sortie-vide** signale que le transfert est possible (tampon de sortie libre)

Algorithme de l'aiguilleur

```
initialiser les drapeaux d'entrée à vide;   (initialisation de l'aiguilleur)
tant que vrai faire
    répéter
        initialiser les prises en compte;
        échantillonner les entrées;
        sélectionner une demande;
    jusqu'à entrée pleine
    répéter
        lire l'entrée sélectionnée; (transfert vers registre interne)
        étudier son acheminement; (détermination du tampon de
        sortie)
        mettre à jour les prises en compte; (pour la prochaine
        sélection des entrées)
        échantillonner entrées; (pour l'acheminement des entrées
        suivantes)
        si sortie vide
            alors
                calculer DX' et DY';   (mise à jour champ-routage)
                acheminer le message;   (écriture sortie)
                mettre à jour le drapeau du tampon d'entrée;
                mettre à jour le drapeau du tampon de sortie;
        fin si
        sélectionner une demande d'entrée;
    jusqu'à entrée pleine
fin tant que
```

Cet algorithme est séquentiel, et ne reflète pas exactement celui implémenté. En effet, pour accroître les performances de l'aiguilleur on utilise au maximum le parallélisme entre l'encodeur de priorité et la partie-traitement de l'aiguilleur, la partie-contrôle gérant le recouvrement. Mais les caractéristiques propres du PLA que nous avons implanté (mémorisation des entrées / sorties) ne permettent pas une optimisation du fonctionnement de l'aiguilleur (voir la réalisation du PLA).

On déduit de cet algorithme, les signaux de contrôle de l'aiguilleur :

- ❑ **tout-vider** (encodeur) : initialisation à vide des drapeaux des tampons d'entrée de l'encodeur;
- ❑ **init-pec** (encodeur) : initialisation (mise à 1) des prises en compte associées aux tampons d'entrée de l'encodeur;
- ❑ **incr-pec** (encodeur) : incrémentation des prises en compte (annihilation - mise à 0 - de la prise en compte du tampon d'entrée sélectionné);
- ❑ **latcher-entrée** (encodeur) : échantillonnage de l'état des tampons d'entrée;
- ❑ **lecture-entrée** (encodeur + pta) : commande le transfert de message entre le tampon d'entrée et le registre interne de l'aiguilleur;
- ❑ **écriture-sortie** (encodeur + pta) : commande le transfert de message entre le registre interne de l'aiguilleur et le tampon de sortie.

2.3.3 Réalisation de l'encodeur de priorité

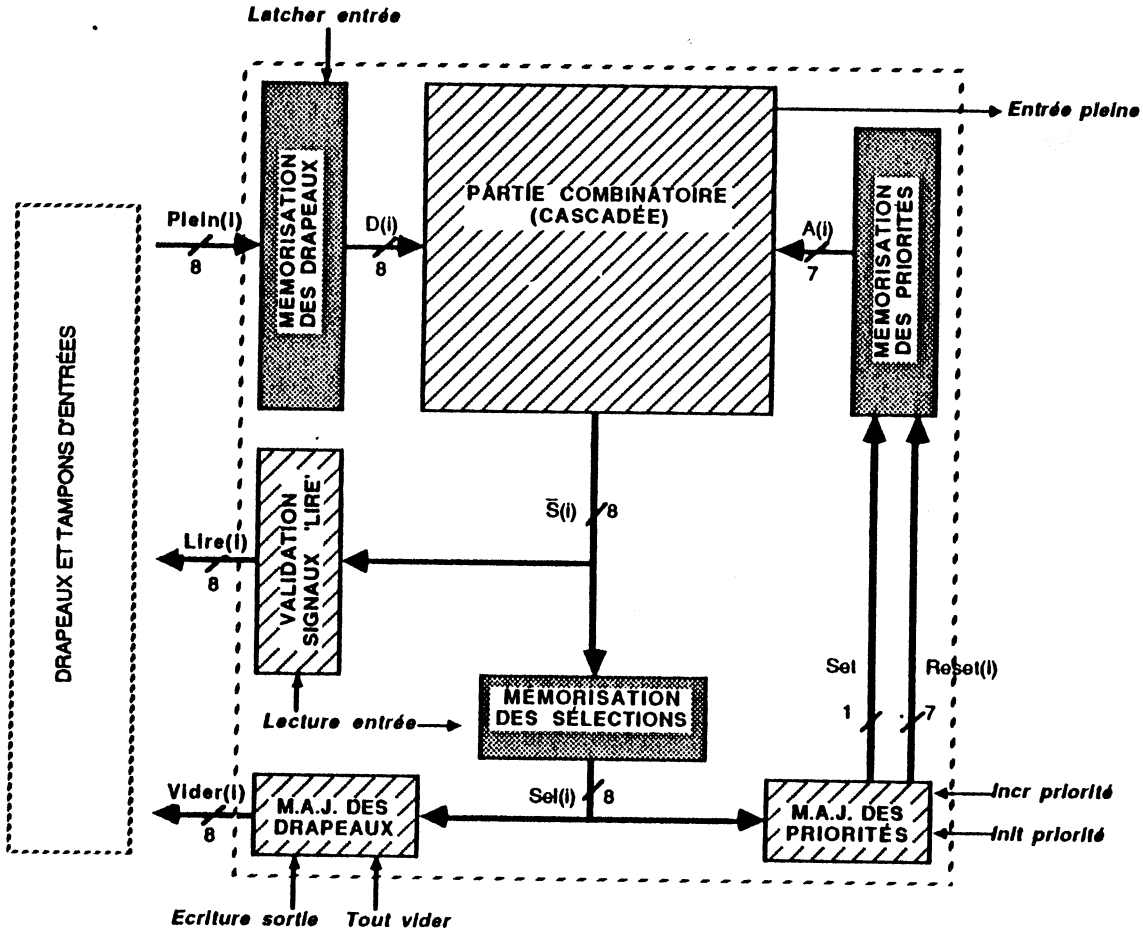
Le rôle de l'encodeur de priorité est de sélectionner la plus prioritaire des entrées en attente à un moment donné et de transférer cette demande dans un tampon interne à l'aiguilleur. L'encodeur de priorité intègre la gestion de ces priorités (prises en compte), ainsi que la mise à jour des drapeaux des tampons d'entrée.

2.3.3.1 Etude fonctionnelle

Les différents composants fonctionnels de l'encodeur se répartissent d'un côté, en éléments de mémorisation, de l'autre, en blocs combinatoires.

Nous avons besoin de mémoriser :
- l'état (plein ou vide) des tampons d'entrée,
- les prises en compte associées à chacune des ces entrées,
- la sélection (ou la non sélection) de celles-ci.

Nous avons également besoin de structures combinatoires pour :
- déterminer la sélection ou la non-sélection des tampons d'entrée,
- mettre à jour les priorités (prises en compte) des entrées à chaque pas de l'aiguilleur,
- mettre à jour les drapeaux des tampons d'entrée en cas d'acheminement,
- transférer le message sélectionné vers le registre interne.



Architecture de l'encodeur de priorité

Les signaux d'entrée nécessaires à l'encodeur viennent :
- du PLA : 'tout-vider', 'init-pec', 'incr-pec', 'latcher-entrée', 'lecture-entrée' et 'écriture-sortie'
- des drapeaux des tampons d'entrée : 'plein'

Les signaux de sortie de l'encodeur sont émis en direction :
- du PLA : 'entrée-pleine'

- des tampons d'entrée : 'lire' et 'vider'

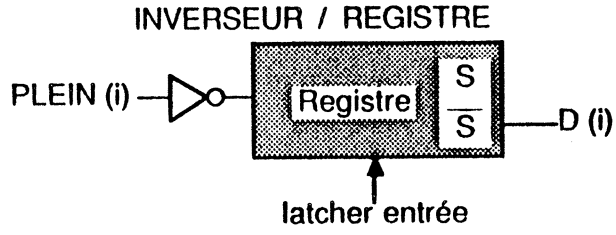
2.3.3.2 Etude logique

Les points de mémorisation

Mémorisation des drapeaux

La mémorisation du signal 'plein' des huit drapeaux des tampons d'entrée est réalisée par de simples registres (cellule REG). Le signal d(i) à la sortie de ces points-mémoire vaut :
- 1 si le tampon de priorité i est plein, donc s'il a un message à transmettre,
- 0 sinon.

L'écriture dans ces registres est commandée par le signal 'latcher-entrée' du PLA. Le signal d'entrée est régénéré par inverseur, on lit donc la sortie inversée du registre.



Mémorisation des prises en compte

Il s'agit ici de l'autorisation de traiter les entrées, et non de la notion de priorité qui, elle, dépend de l'ordre (câblé) dans lequel on traite les demandes.

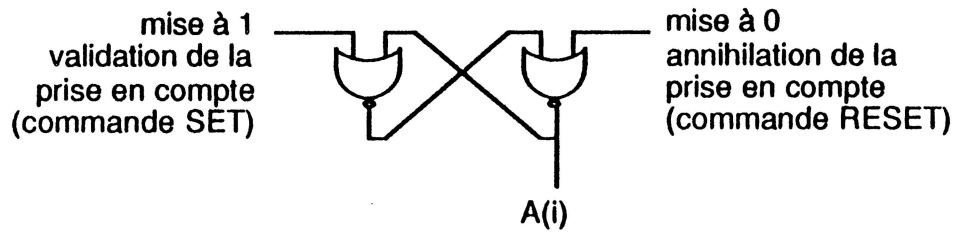
La valeur mémorisée vaut 1 si l'on autorise la prise en compte de l'entrée correspondante (il n'y a éventuellement pas de demande en attente), et 0 sinon (l'entrée a déjà été traitée - avec ou sans succès -).

L'information mémorisée doit pouvoir être mise à jour par une mise à 1 (cas de l'initialisation de la prise en compte) et une mise à 0 (suppression de la prise en compte).

Nous n'avons besoin ici que de sept éléments de mémorisation. En effet, l'entrée la moins prioritaire (priorité 8) est toujours prise en compte, car elle est toujours la dernière à être étudiée. Sa sélection

entraîne une ré-initialisation à 1 de toutes les prises en compte des entrées.

Nous avons utilisé des bascules RS pour réaliser ces points-mémoire.



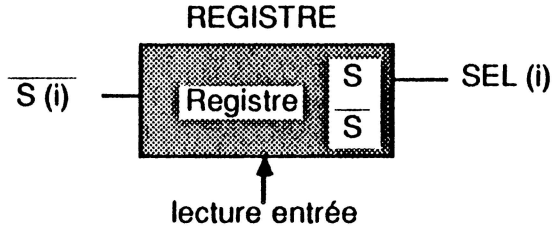
Les commandes, SET et RESET, ne sont pas celles directement générées par le PLA ('init-pec', 'incr-pec').

Mémorisation des sélections

Pour chacun des huit tampons d'entrée, on mémorise ici le fait qu'il est (ou qu'il n'est pas) sélectionné par l'encodeur de priorité. Nous avons donc en sortie de ces points mémoires les signaux stabilisés Sel(i).

Ces signaux valent 1, si le tampon d'entrée de priorité i est sélectionné afin d'étudier l'acheminement de son message, 0 sinon. Un seul de ces signaux Sel(i) ne peut être valide (=1) à la fois.

Comme pour l'échantillonnage des drapeaux, nous utilisons ici des registres simples. A noter que l'on mémorise la valeur inversée de la sélection. L'écriture, dans ces registres, est commandée par le signal 'lecture-entrée' du PLA qui marque la fin de la phase de sélection de l'encodeur.



La partie combinatoire de l'encodeur

C'est elle qui va déterminer la demande d'entrée choisie pour être acheminée éventuellement.

En entrée de ce bloc combinatoire, nous avons :
- les demandes d'acheminement D(1→8)
- leurs prises en compte respectives A(1→7), A(8)=1

et en sortie
-les signaux de sélection de chacune des entrées : S(1→8)
- le compte-rendu de sélection 'entrée pleine', destiné au PLA, indiquant que l'on a un message à transmettre (succès de la sélection)

Equations

A l'origine, on a l'équation :

$$S(i)=D(i) \cdot A(i)$$

Mais une seule entrée doit être sélectionnée à la fois. On va donc faire intervenir un signal d'état indiquant qu'une demande a (ou n'a pas) été déjà retenue. Ce signal va agir à la façon d'une prise en compte, et se propager à travers l'encodeur (d'où le terme de partie combinatoire cascadée).
Etat = 0 : on a déjà sélectionné une entrée, il y a donc blocage.
Etat = 1 : on n'a pas encore sélectionné d'entrée, on peut continuer la recherche.

On en déduit les équations générales de l'encodeur :

$S(i) = D(i) \cdot A(i) \cdot \overline{Etat(i-1)}$ avec $Etat(0) = 1$
$Etat = Etat(i-1) \cdot \overline{S(i)}$ et $A(8) = 1$

On sélectionne l'entrée i, :
- si elle en fait la demande : D(i) = 1,
- si l'encodeur est autorisé à prendre cette demande en compte : A(i)=1,
- si aucune entrée n'a déjà été sélectionnée.

On propage un nouvel état directement fonction de :
- l'état précédent,
- et de la sélection courante.

Nous avons donc implémenté les fonctions suivantes :

$S(1) = D(1) \cdot A(1)$ $\quad Etat(1) = \overline{S(1)}$
$S(i) = Etat(i-1) \cdot D(i) \cdot A(i)$ $\quad Etat(i) = Etat(i-1) \cdot \overline{S(i)}$

$S(8) = D(8) . Etat(7)$ $\qquad$ $Etat(8) = Etat(7) . \overline{S(8)}$

Remarque : entrée-pleine = $\overline{Etat(8)}$

Réalisation en porte logique

$\overline{S(1)} = NAND2 [D(1), A(1)]$

$\overline{S(i)} = NAND3 [Etat(i-1), D(i), A(i)]$

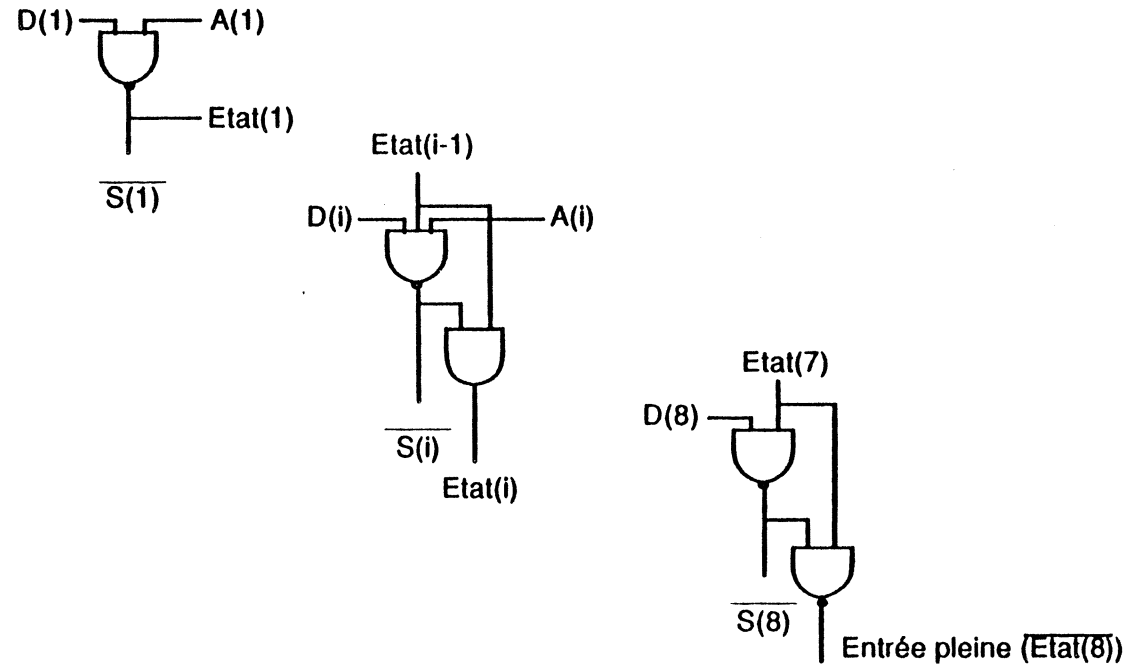
$\overline{S(8)} = NAND2 [D(8), Etat(7)]$

$Etat(1) = \overline{S(1)}$

$Etat(i) = AND2 [Etat(i-1), \overline{S(i)}]$

Entrée-pleine = NAND2 [Etat(7), S(8))]

Schéma logique de la partie combinatoire de l'encodeur



Remarque : on génère $\overline{S(i)}$ et non S(i)

Problème temporel

Le temps de stabilisation de la partie combinatoire de l'encodeur pose un des plus graves problèmes temporels de l'aiguilleur. En effet, il est nécessaire, avant de commencer la phase suivante (état P3 du séquenceur : échantillonnage des sélections mais surtout, lecture de l'entrée sélectionnée), que les signaux

$\overline{S(1 \to 8)}$ soient stabilisés. Le calcul de la sélection ne peut pas 'commencer' sur l'état P1, P5 ou P7, mais au début de l'état suivant (P2) après avoir échantillonné les entrées.

La mise à jour des drapeaux

Cette séquence combinatoire permet, sous contrôle du PLA, de vider les drapeaux des tampons d'entrée de l'aiguilleur. On est amené à vider :

- tous ces drapeaux lors de l'initialisation de l'aiguilleur,
- le drapeau du tampon dont on vient d'acheminer le message.

Dans le premier cas, c'est le signal 'tout-vider' du séquenceur qui commande l'opération, dans le deuxième, le signal 'écriture-sortie' ne doit s'appliquer que sur l'entrée sélectée.

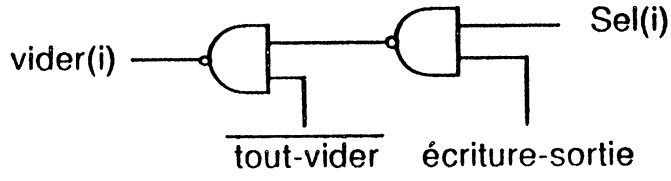
Ces contraintes se formalisent sous l'équation :

vider(i) = tout-vider + Sel(i) . écriture-sortie

On en déduit l'équation en porte NAND :

vider(i) = NAND2 [$\overline{\text{tout-vider}}$, NAND2 [Sel(i), écriture-sortie]]

Réalisation en porte logique



Problèmes temporels

Les tampons périphériques étant parfois très éloignés de l'aiguilleur, les commandes vider(i) doivent charger des lignes assez capacitives.

La mise à jour des prises en compte

Ce mécanisme combinatoire agit sur les prises en compte associées aux entrées (bascules RS), à deux niveaux :

- mise à 1 (on autorise la prise en compte),

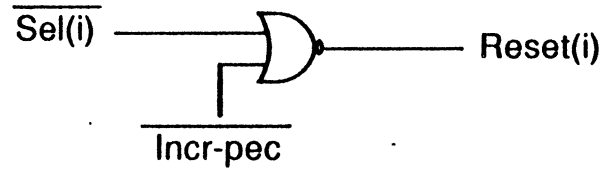
- mise à 0 (on annihile celle-ci).

La mise à 0 de prises en compte (commande RESET)

Lorsqu'une entrée de priorité i a été sélectionnée, qu'elle soit acheminée (tampon de sortie vide) ou pas, on s'interdit de la re-sélectionner jusqu'à une nouvelle ré-initialisation des prises en compte. On annihile donc cette entrée en agissant sur la commande RESET de sa bascule de prise en compte.

Equation Reset(i) = Sel(i) . incr-pec

Réalisation en porte logique



La mise à 1 de prises en compte (commande SET)

L'encodeur est amené à mettre à 1 les prises en compte des entrées à deux reprises :

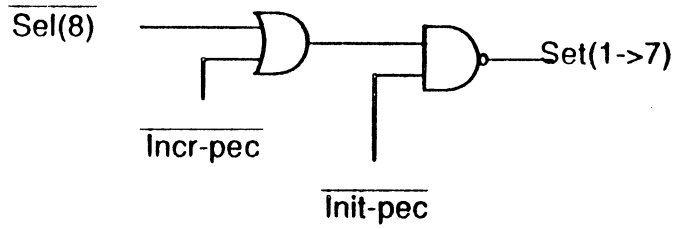
- lors de l'initialisation de celles-ci par l'aiguilleur,
- lors de la sélection de l'entrée la moins prioritaire (priorité 8); en effet, si celle-ci a été choisie, c'est que toutes les demandes plus prioritaires ont déjà été envisagées et leurs prises en compte annihilées.

Nous pouvons formaliser les contraintes des signaux Set(i) par l'équation : Set(i)= init-pec + S(8) . incr-pec avec 0<i<8

Soit en porte logique

Set(1→7)= NAND2 [$\overline{\text{init-pec}}$, OR2 [$\overline{\text{S(8)}}$, $\overline{\text{incr-pec}}$]]

Schéma logique



Problème temporel

Pas de problème temporel lors de la mise à jour des prises en compte puisque celle-ci intervient toujours avant la période de calcul des sélections (état P1, P5 ou P7 → état P2). Les prises en compte sont donc stables au moment où commence celle-ci.

Validation des signaux 'lire'

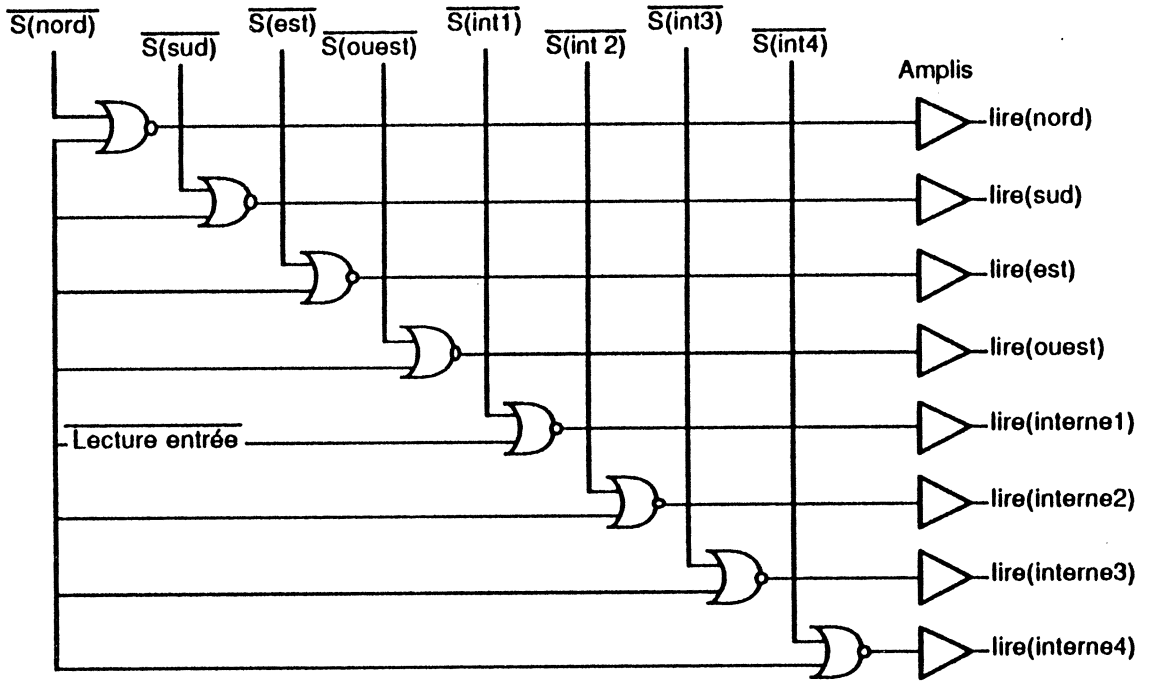
Une fois déterminé le tampon d'entrée, on transfère le contenu de son message dans le registre interne de l'aiguilleur afin d'étudier son acheminement. Le transfert s'effectue par le bus périphérique de la cellule (dix bits), on lit d'un côté pendant que l'on écrit de l'autre. Ceci est réalisé par le démultiplexage (1→8) du signal 'lecture-entrée' vers les signaux 'lire' (nord, sud, est, ouest et calcul) des tampons d'entrée par les commandes de sélection $\overline{S(nord)}$, $\overline{S(sud)}$, $\overline{S(est)}$, $\overline{S(ouest)}$, $\overline{S(int1)}$, $\overline{S(int2)}$, $\overline{S(int3)}$ et $\overline{S(int4)}$.

Nous avons ici le problème temporel le plus important de l'aiguilleur. En effet, on introduit un premier retard sur le signal 'lire' (capacité de la ligne) puis un deuxième retard (beaucoup plus important) lors du chargement du bus. Il faut être sûr lorsque retombe le signal 'lecture-entrée' que les données ont bien été dupliquées.

Equation Lire(i) = S(i) . lecture-entrée

d'où Lire(i)= NOR2 [$\overline{S(i)}$, $\overline{\text{lecture-entrée}}$]

Schéma logique



Remarque : les signaux de sortie 'lire' sont amplifiés

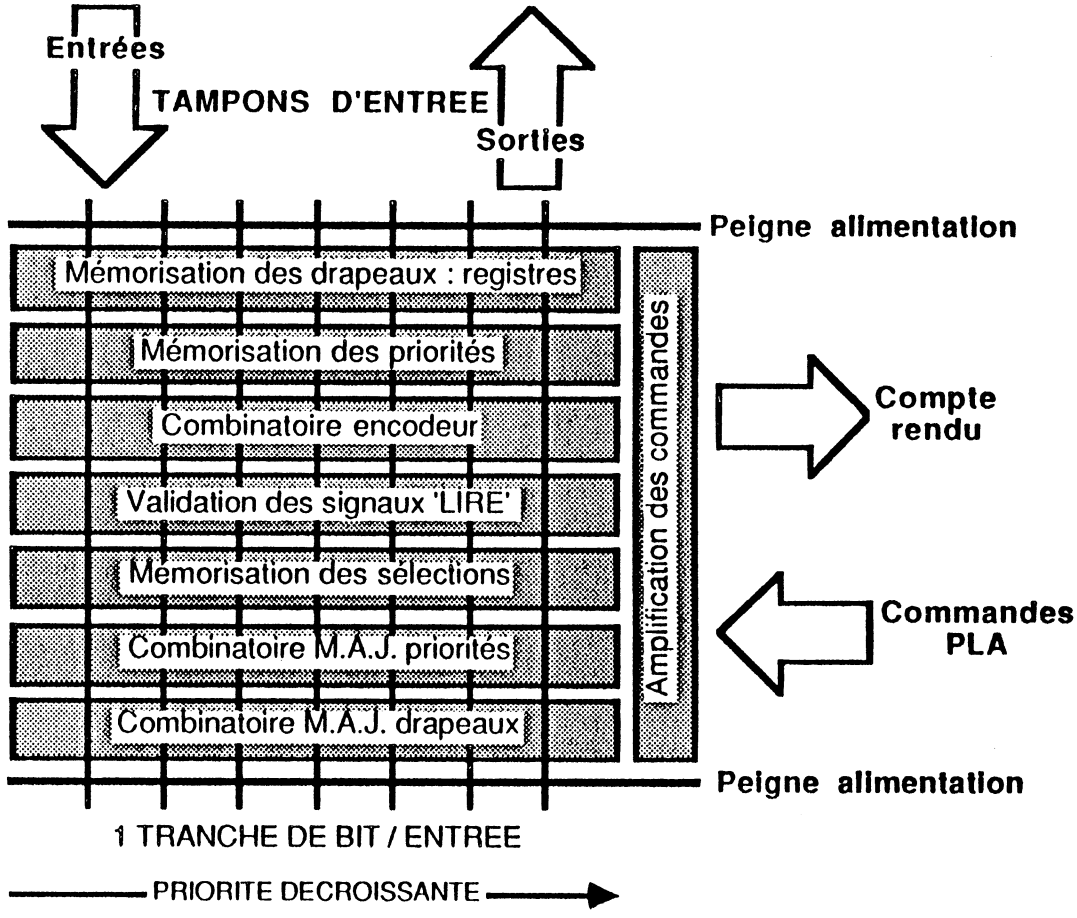
2.3.3.3 Implantation de l'architecture

De part sa fonctionnalité, l'encodeur de priorité se prête bien à une implantation régulière en tranches (bit slice) de son architecture. Pour chaque entrée, on doit effectuer à la chaîne les opérations suivantes : soit i la priorité courante

- mémorisation de la demande - A(i)
- mémorisation de la prise en compte - PC(i)
- détermination de la sélection - S(i)
- validation du signal de lecture du tampon i - lire(i)
- mémorisation de la sélection - Sel(i)
- mise à jour de la prise en compte - Reset(i)
- mise à jour du tampon d'entrée i - vider(i)

Les signaux du PLA - 'latch-entrée', 'init-pec', 'incr-pec', 'tout-vider', 'lecture-entrée', 'écriture-sortie' - et le compte-rendu 'entrée-pleine' sont propagés à travers l'encodeur par simple juxtaposition des tranches de bit. Les priorités sont définies par la position respectives des tranches entre elles. Les tampons d'entrée ont donc

comme priorité, le rang de la tranche de bit à laquelle ils sont physiquement connectés (signaux 'plein', 'lire', 'vider').



Plan de masse de l'encodeur de priorité

De part sa conception, on pourrait facilement implanter un encodeur gérant plus (ou moins) de huit entrées.

Toutes les entrées/sorties de l'encodeur sont amplifiées. Les signaux en provenance ou à destination du PLA sont amplifiés par une "tranche de bit" dédiée, alors que l'amplification des signaux en relation avec les tampons est intégrée dans les différents étages de la tranche.

Organisation des entrées / sorties

Afin de simplifier la connectique, déjà complexe de la cellule, nous avons regroupé, lors de l'implantation de l'encodeur, les signaux communiquant avec les mêmes entités fonctionnelles.

Communication vers les tampons d'entrée

Les 8x3 signaux ('plein, lire, vider (1→8)') sont répartis le long de la frontière nord.

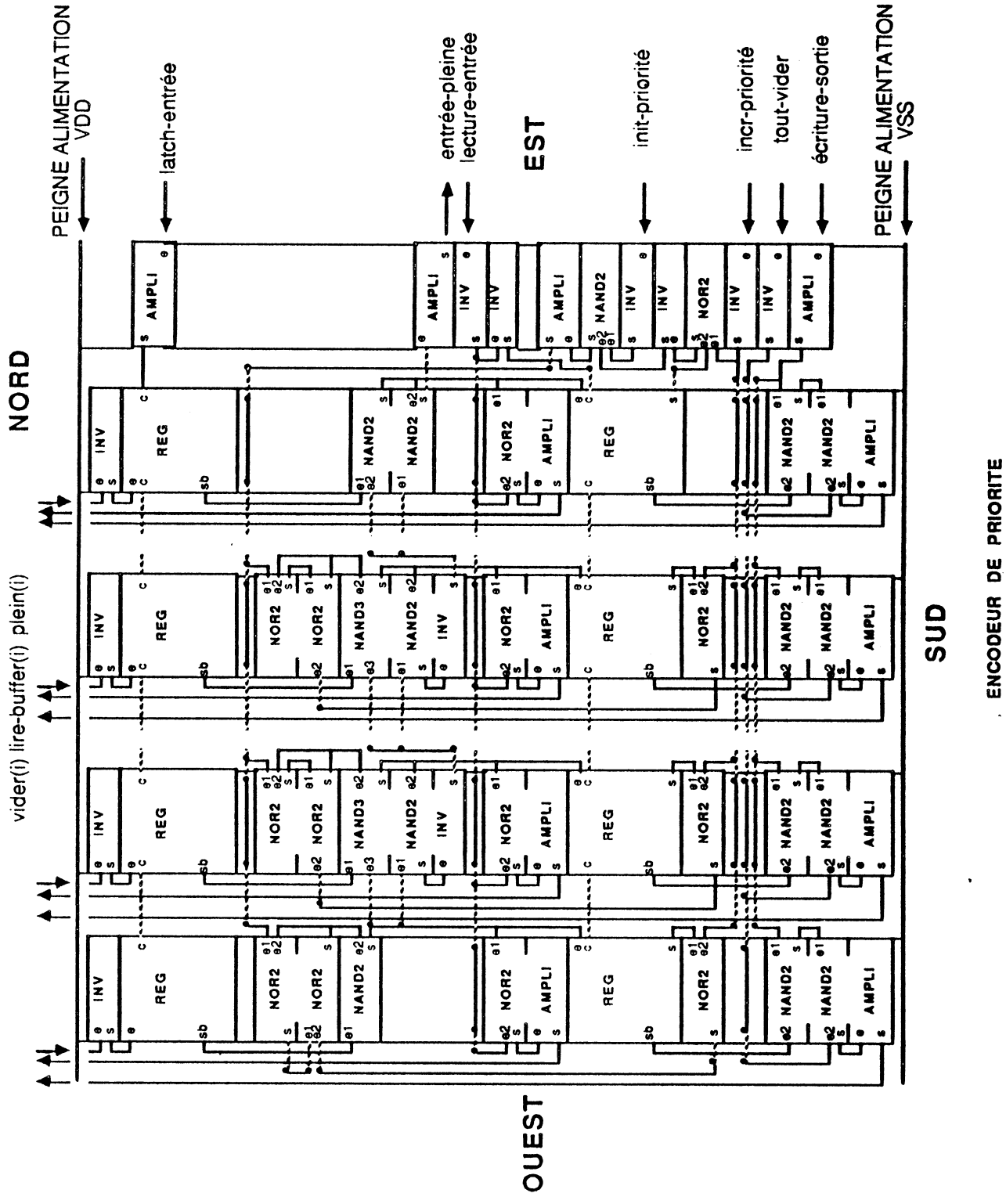
Communication avec le PLA

Les 6 commandes ('latch-entrée', 'init-pec', 'incr-pec', 'tout-vider', 'lecture-entrée', 'écriture-sortie') et le compte-rendu ('entrée-pleine') sont accessibles sur la frontière est.

Alimentations

On alimente l'encodeur de priorité par des rails orientés est/ouest, le long des frontières nord et sud. Les alimentations sont ensuite redistribuées par des peignes nord/sud.

Implantation détaillée



Densité

L'encodeur de priorité est formé de 508 transistors, son enveloppe est un rectangle de 1112 µm par 623 µm soit 0,692 mm². La densité d'intégration est donc de 734 tr/mm².

2.3.4 Détermination du tampon de sortie

Le rôle de la partie-traitement de l'aiguilleur est de déterminer le tampon de sortie de l'entrée sélectionnée par l'encodeur. De plus, elle intègre la mise à jour, des messages à acheminer (champ-routage), et des tampons de sortie.

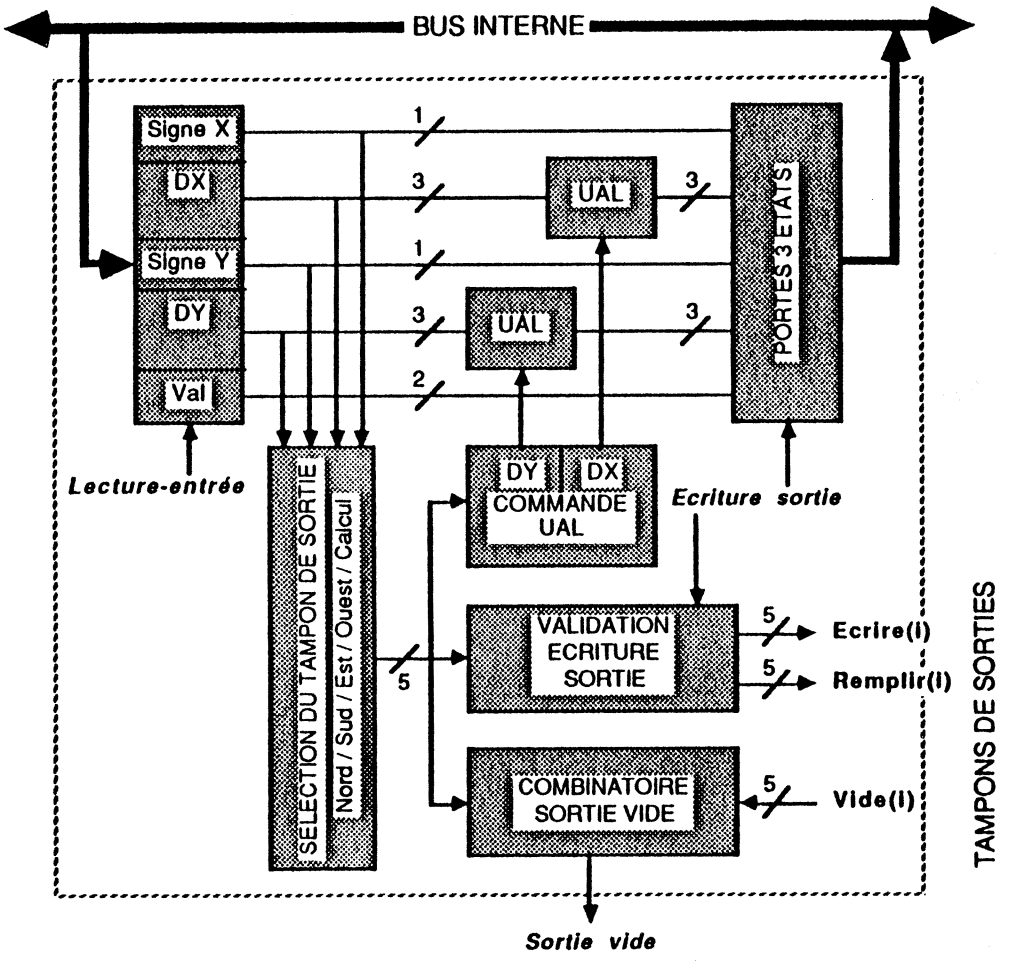
2.3.4.1 Etude fonctionnelle

La partie-traitement de l'aiguilleur est structurée autour :

- du registre interne de l'aiguilleur;
- d'une mini UAL et de son bloc de commande;
- d'une partie combinatoire déterminant le tampon de sortie;
- de 2 blocs combinatoires;
  - pour la mise à jour des tampons de sortie,
  - pour le calcul du compte-rendu.

Bien que fonctionnellement indépendant, le registre interne de l'aiguilleur a été complètement intégré à la partie-traitement de l'aiguilleur lors de l'implantation. En effet, l'interconnexion de ces deux blocs est telle qu'il est impossible de les dissocier.

La partie-traitement de l'aiguilleur prend et rend sur le bus périphérique (commandes 'lecture-entrée', 'écriture-sortie' du PLA) les dix bits de données des messages à transférer. Seuls les huit bits du champ routage (Sx, Dx2, Dx1, Dx0, Sy, Dy2, Dy1 et Dy0) servent aux calculs. A l'inverse de l'encodeur de priorités, on utilise ici les commandes 'vide' des drapeaux des tampons de sortie, et l'on met à jour ceux-ci par les signaux 'écrire' et 'remplir'. Enfin, la partie-traitement de l'aiguilleur émet vers le PLA le compte-rendu de son activité sous la forme du signal 'sortie-vide'.



Architecture de la partie-traitement de l'aiguilleur

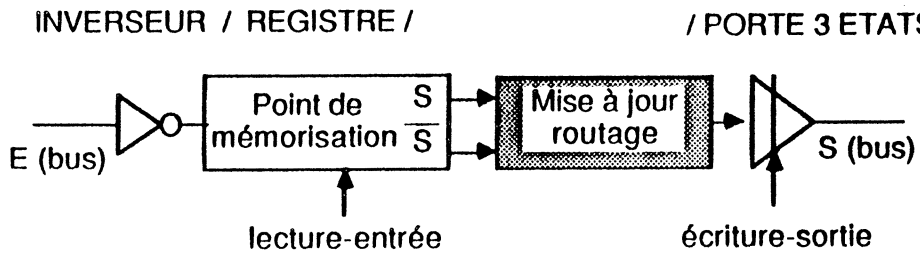
2.3.4.2 Etude logique

Le registre interne

Le registre interne mémorise les 10 bits du message courant :
- 4 bits pour le déplacement relatif horizontal (1 bit de signe Sx et 3 bits pour la valeur absolue du déplacement Dx2, Dx1 et Dx0),
- 4 bits pour le déplacement relatif vertical (1 bit de signe Sy et 3 bits pour la valeur absolue du déplacement Dy2, Dy1 et Dy0),
- 2 bits de données (Val1 et Val0).

Le registre interne a la même architecture que les tampons, mais il n'a pas de drapeaux, il est formé, pour chacun des bits à mémoriser, d'un registre simple et d'une porte trois états. L'écriture dans ces registres est commandée par le signal 'lecture-entrée'. La

lecture de ceux-ci (écriture sur le bus) est contrôlée par la commande 'écriture-sortie'. Le signal d'entrée du registre est inversé (régénération du signal).



Remarque : le champ-routage est mis à jour par la partie-traitement de l'aiguilleur.

Sélection du tampon de sortie

Il s'agit ici, en fonction de la valeur du champ-routage - DX=Sx, dx2, dx1, dx0 et DY=Sx, dy2, dy1, dy0 - et en fonction des signaux 'bord-est' et 'bord-ouest' de déterminer selon l'algorithme d'acheminement le tampon de sortie du message. Les résultats de ce traitement combinatoire seront les 5 signaux de sélection des tampons de sortie (nord, sud, est, ouest et calcul (partie-traitement de la cellule)).

Exemple : nord = 1 si le message courant doit être acheminé vers le tampon de sortie périphérique nord de la cellule, 0 sinon.

Le calcul se fait en 2 étapes, avec la génération intermédiaire des signaux :

- dx=0 vrai (=1) si DX=0 (déplacement relatif horizontal), faux (=0) sinon;
- dx>0 vrai si DX strictement positif, faux sinon;
- dx<0 vrai si DX strictement négatif, faux sinon.

De la même façon, on calculera les signaux dy=0, dy>0 et dy<0.
Les signaux di=0, di>0 et di<0 sont exclusifs.

Equations

Rappel : les bits de signe des déplacements (Sx, Sy) valent 0 si ceux-ci sont positifs et 1 sinon.

On a les équations suivantes :

$(dx=0) = \overline{dx2} \cdot \overline{dx1} \cdot \overline{dx0}$

$(dx>0) = \overline{(dx=0)} \cdot \overline{Sx}$

$(dx<0) = \overline{(dx=0)} \cdot Sx$

Ces équations sont les mêmes pour DY.

On en déduit les équations des signaux de sortie :

$ouest = (dx<0) \cdot ((dy=0) + \overline{bord\text{-}ouest})$

$est = (dx>0) \cdot ((dy=0) + \overline{bord\text{-}est})$

$nord = (dy<0) \cdot ((dx=0) + (dx>0).bord\text{-}est + (dx<0).bord\text{-}ouest))$

$sud = (dy>0) \cdot ((dx=0) + (dx>0).bord\text{-}est + (dx<0).bord\text{-}ouest))$

$calcul = (dx=0) \cdot (dy=0)$

Equations en porte logique

$(dx=0) = NOR3 [dx2, dx1, dx0]$

$(dx>0) = NOR2 [(dx=0), sx]$

$(dx<0) = NOR2 [(dx=0), \overline{Sx}]$

$inter = NAND3 [\overline{(dx=0)}, NAND2[(dx>0), bord\text{-}est], NAND2[(dx<0), bord\text{-}ouest]]$

$\overline{nord} = NAND2 [(dy<0), inter]$

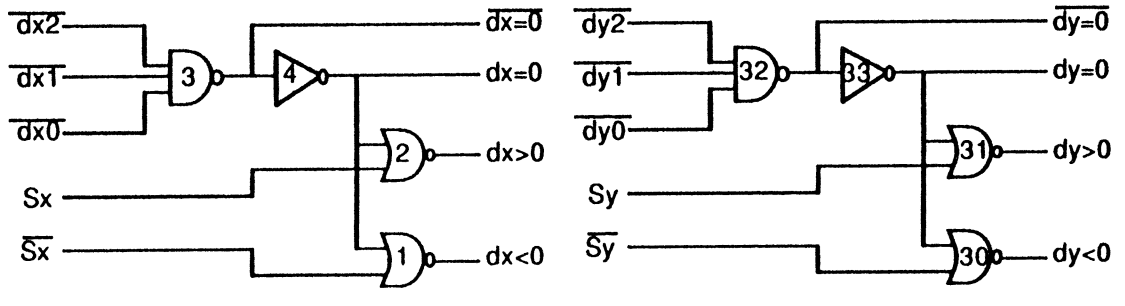
$\overline{sud} = NAND2 [(dy>0), inter]$

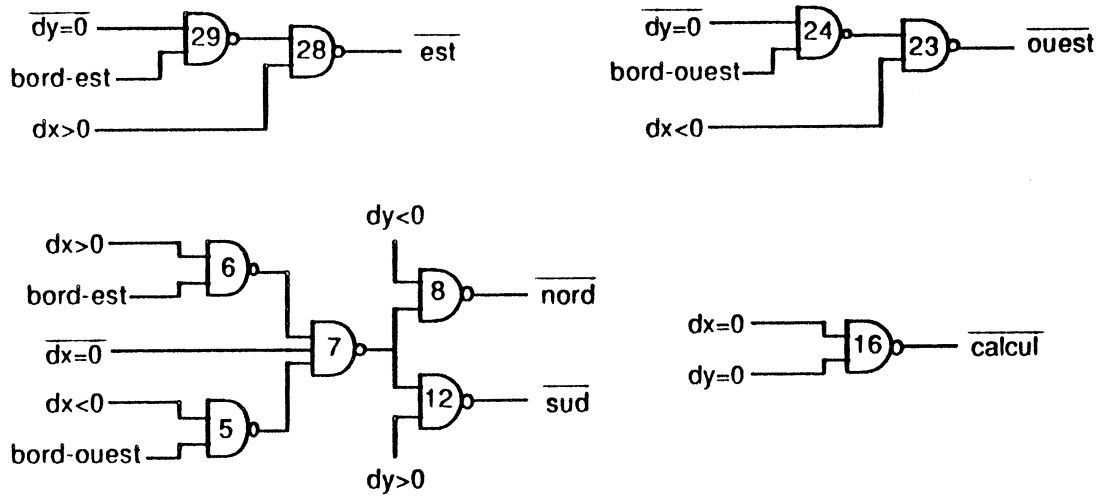
$\overline{est} = NAND2 [(dx>0), NAND2[\overline{(dy=0)}, bord\text{-}est]]$

$\overline{ouest} = NAND2 [(dx<0), NAND2[\overline{(dy=0)}, bord\text{-}ouest]]$

$\overline{calcul} = NAND2 [(dx=0), (dy=0)]$

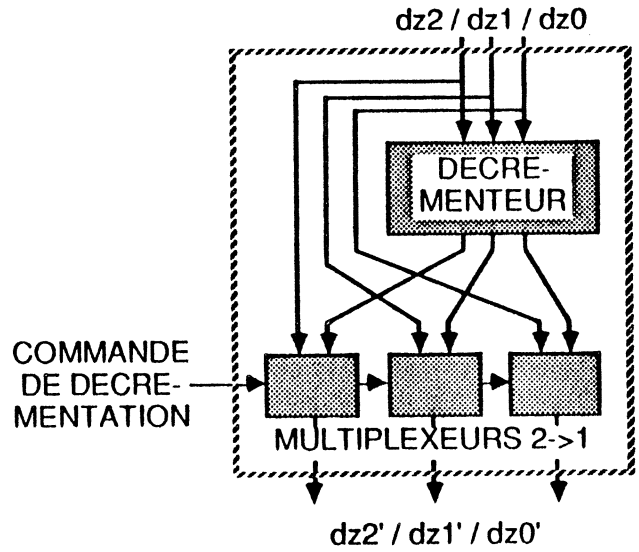
Schéma logique





La mise à jour du champ-routage

La mise à jour du champ-routage du message consiste en la décrémentation d'une des deux coordonnées (voir l'algorithme d'acheminement). Deux minis UAL (une pour chaque coordonnée) sont affectées à cet usage, elles sont formées d'un décrémenteur (trois bits) et de trois multiplexeurs (2 voies vers 1).



Architecture de l'UAL

Les commandes de ces UAL - 'decrX' et 'decrY' - agissent sur des multiplexeurs 2→1 pour valider ou non la sortie du décrémenteur.

Le décrémenteur

L'UAL proprement dite est un décrémenteur cascadé sur trois bits, du bit de poids le plus faible, vers le plus fort ($b_0 \rightarrow b_2$), avec propagation de la retenue éventuelle ($r_0 \rightarrow r_2$).

Equations $b_i' = \overline{b_i}.r_i + b_i.\overline{r_i}$

$r_{i+1} = \overline{b_i}.r_i$

Equations en porte logique $b_i' = OUEX[b_i, r_i]$

$r_{i+1} = AND2[\overline{b_i}, r_i]$

On peut simplifier ces équations dans le cas du premier et du dernier bit (b_0 et b_2). En effet la retenue initiale (r_0) est toujours à 1, et l'on a pas à générer de retenue sortante (r_3). On a donc implémenté les relations suivantes :

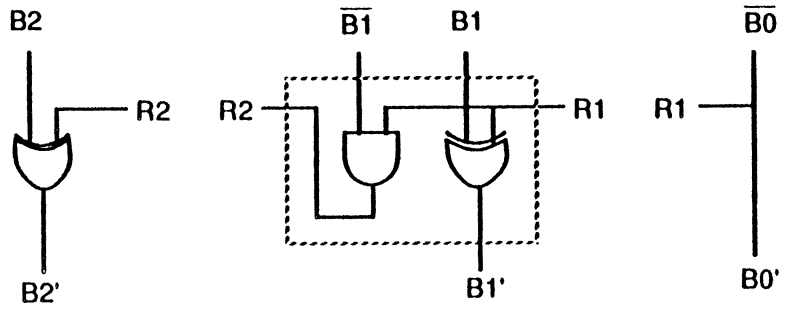
$b_0' = \overline{b_0}$

$b_1 = OUEX[b_1, \overline{b_0}]$

$r_2 = AND2[\overline{b_1}, r_1]$

$b_2 = OUEX[b_2, r_2]$

Schéma logique



Commandes UAL

Elles agissent sur les deux UALs afin de valider ou non la décrémentation des coordonnées. Elles dépendent du tampon de sortie vers lequel sera acheminé le message. On décrémente abcisse si le message est acheminé horizontalement (est ou ouest), et l'ordonnée si celui-ci est acheminé verticalement (nord ou sud).

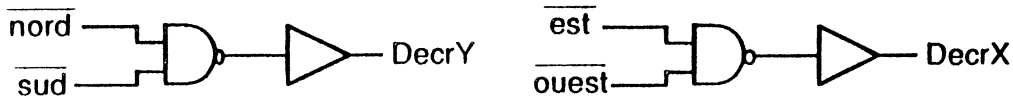
Enfin, si le message est dirigé vers la partie-traitement de la cellule, on ne modifie pas le champ-routage.

Equation DecrY = nord + sud
DecrX = est + ouest

D'où DecrY = NAND2 [$\overline{\text{nord}}$, $\overline{\text{sud}}$]
DecrX = NAND2 [$\overline{\text{est}}$, $\overline{\text{ouest}}$]

decrZ = 1 on valide la décrémentation (dz')
decrZ = 0 on ne modifie pas la valeur de la coordonnée relative

Schéma logique



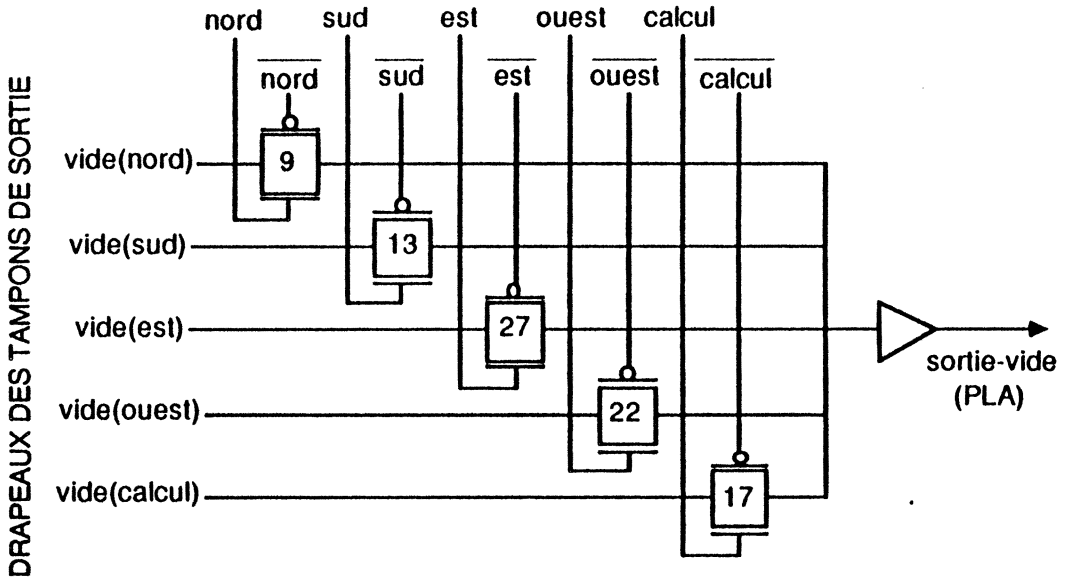
Remarques : les signaux 'decrX' et 'decrY' sont amplifiés, car ils doivent commander chacun 3 multiplexeurs.

Mise à jour du signal 'sortie-vide'

Une fois déterminé le tampon de sortie, il s'agit de savoir si celui-ci est libre (vide) ou non. Ceci est réalisé par le multiplexage (5→1) des signaux vide(i), par les commandes 'nord', 'sud', 'est', 'ouest' et 'calcul'. Le signal 'sortie-vide' à destination du PLA vaut 1 si le tampon de sortie envisagé est vide (l'acheminement est alors possible), 0 sinon.

Equation
sortie-vide = vide(nord).nord + vide(sud).sud + vide(est).est +
vide(ouest.ouest + vide(calcul).calcul

Schéma logique



Remarque : Nous avons utilisé pour implanter les interrupteurs duaux CMOS, la cellule Inter21combar.
Le signal de sortie 'sortie-vide' est amplifié.

Mise à jour des drapeaux de sortie

Une fois le tampon de sortie déterminé, s'il est vide, il faut transférer le message contenu dans le tampon interne de l'aiguilleur vers ce tampon de sortie. La commande 'écriture-sortie' du PLA contrôle ce transfert ainsi que la mise à jour des drapeaux des tampons concernés. Ceci est réalisé par le démultiplexage (1→5) du signal 'écriture-sortie' vers les signaux 'écrire' et 'remplir' (nord,sud, est, est, ouest et calcul) par les commandes de sélection $\overline{\text{nord}}$, $\overline{\text{sud}}$, $\overline{\text{est}}$, $\overline{\text{ouest}}$ et $\overline{\text{calcul}}$. On peut en effet ici écrire dans le tampon de sortie, et en même temps remplir son drapeau car celui-ci ne sera plein qu'à la fin du signal remplir.

Equations

écrire/remplir(nord) = écriture-sortie . nord
écrire/remplir(sud) = écriture-sortie . sud
écrire/remplir(est) = écriture-sortie . est
écrire/remplir(ouest) = écriture-sortie . ouest
écrire/remplir(calcul) = écriture-sortie . calcul

Equation en portes logiques

écrire/remplir(nord) = NOR2 [$\overline{\text{écriture-sortie}}$. $\overline{\text{nord}}$]

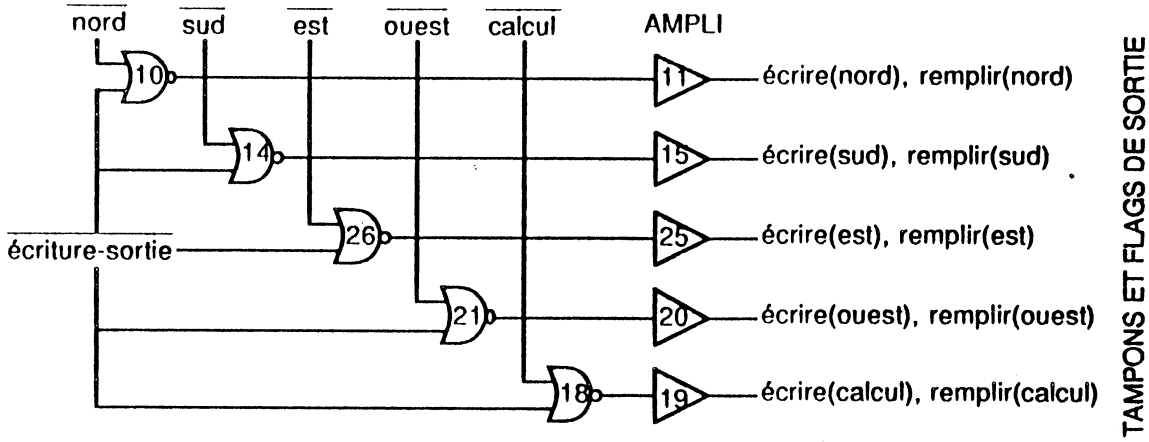
écrire/remplir(sud) = NOR2 [$\overline{\text{écriture-sortie}}$. $\overline{\text{sud}}$]

écrire/remplir(est) = NOR2 [$\overline{\text{écriture-sortie}}$. $\overline{\text{est}}$]

écrire/remplir(ouest) = NOR2 [$\overline{\text{écriture-sortie}}$. $\overline{\text{ouest}}$]

écrire/remplir(calcul) = NOR2 [$\overline{\text{écriture-sortie}}$. $\overline{\text{calcul}}$]

Schéma logique



Remarque : les commandes de sortie écrire/remplir sont amplifiées

Problème temporel

Le signal 'écriture-sortie' (généré par le PLA) est à l'origine de l'écriture sur le bus périphérique d'une information (message à transférer) et à l'origine de la lecture du bus par un tampon (tampon périphérique de sortie).

Pour des questions d'ordre topologique (voir implantation de la cellule), le registre interne de l'aiguilleur est intégré à la partie-traitement de l'aiguilleur. Or celle-ci est beaucoup plus près du PLA que ne le sont les tampons périphériques; pendant quelques nano-secondes (début de la place P7), le bus est donc en état de haute impédance alors que la commande 'lire/remplir(sortie)' est active.

2.3.4.3 Implantation de l'architecture

L'architecture de la partie-traitement de l'aiguilleur se décompose en deux blocs structurellement différents. D'un côté, le registre interne -régulier-; de l'autre, les parties combinatoires par nature irrégulières. Ces deux blocs sont disposés - le bloc registre au nord, le bloc combinatoire au sud - de part et d'autre d'un canal de routage de façon à réduire la longueur de leurs interconnections.

Implantation du registre interne

Comme pour les tampons périphériques, l'implantation du registre interne de l'aiguilleur est régulière (découpe en tranches des bits). les deux UALs (décrémenteurs et multiplexeurs) ont été implantés en tranches avec les bits du registre (bit slice). Afin de minimiser la surface de silicium de la partie-traitement de l'aiguilleur, nous avons également intégré au registre interne la génération des commandes decrX et decrY des UALs.

Les dix bits mémorisés sont (frontière ouest → est) :
Sx, dx2, dx1, dx0, Sy, dy2, dy1, dy0, Val2, Val1.

Les entrées et les sorties (reliées au bus) sont accessibles au nord, les signaux intermédiaires destinés au bloc combinatoire le sont au sud. Les commandes ('lecture-entrée' et 'écriture-sortie') arrivent par la frontière est, elles sont amplifiées et propagées d'est en ouest par juxtaposition des tranches. Leur amplification a été intégrée à la tranche de bit 'val1'.

Comme pour les tampons périphériques, les alimentations sont distribuées nord/sud par deux peignes (VDD au nord, VSS au sud) orientés est/ouest.

Implantation de la logique aléatoire

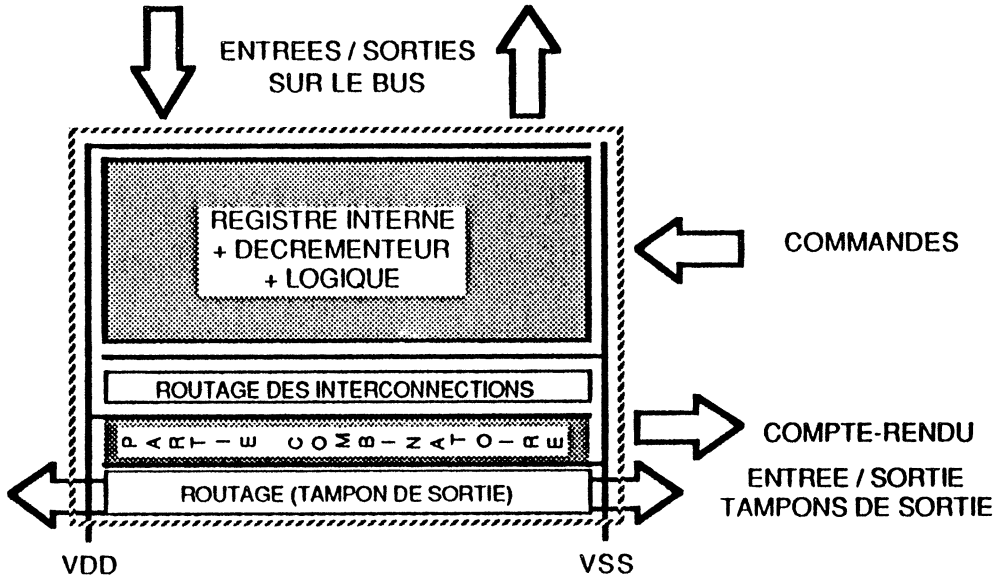
Les seules règles suivies lors de l'implantation des blocs combinatoires de la partie-traitement de l'aiguilleur ont été la simplification des connections et la minimisation de la taille de la

partie-traitement de l'aiguilleur. Elles ont conduit à une implantation "en ligne".

Les portes logiques de base, juxtaposées les unes aux autres, sont regroupées par fonctions à réaliser assurant ainsi la continuité électrique des alimentations. De part et d'autre de la ligne, deux canaux de routage assurent les interconnections. Au nord, nous avons regroupé les connections internes aux différentes fonctions combinatoires, ainsi que celles avec le registre interne; au sud, le canal de routage permet de rassembler sur les frontières est et ouest les connections vers les tampons de sorties. Pour une question de surface, une partie de cette logique aléatoire (commandes UALs) a dû être implantée avec le registre interne.

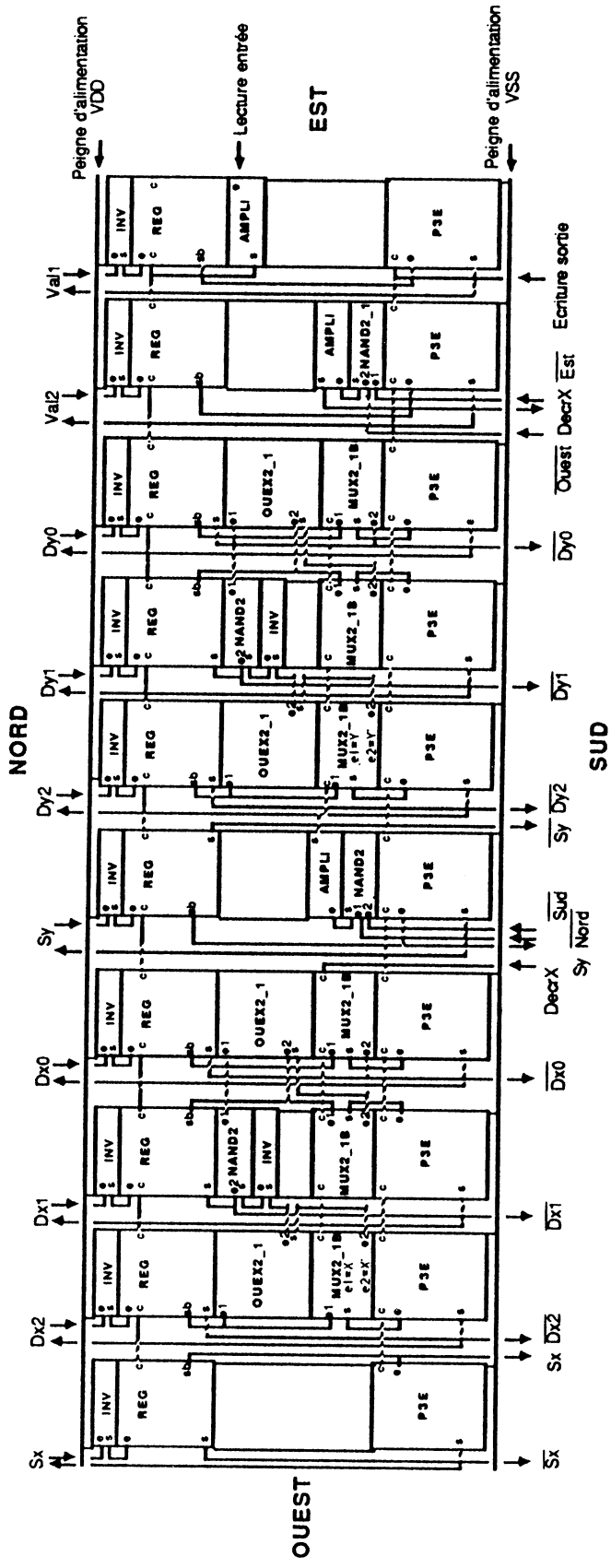
Organisation des entrées/sorties de la partie-traitement de l'aiguilleur

Les entrées/sorties du registre interne (communications avec le bus) sont regroupées au nord. Les signaux échangés avec le PLA (commandes, compte-rendu) sont accessibles sur la frontière est, ainsi que ceux en relation avec les tampons de sortie. Ces derniers sont également disponibles sur la frontière ouest.

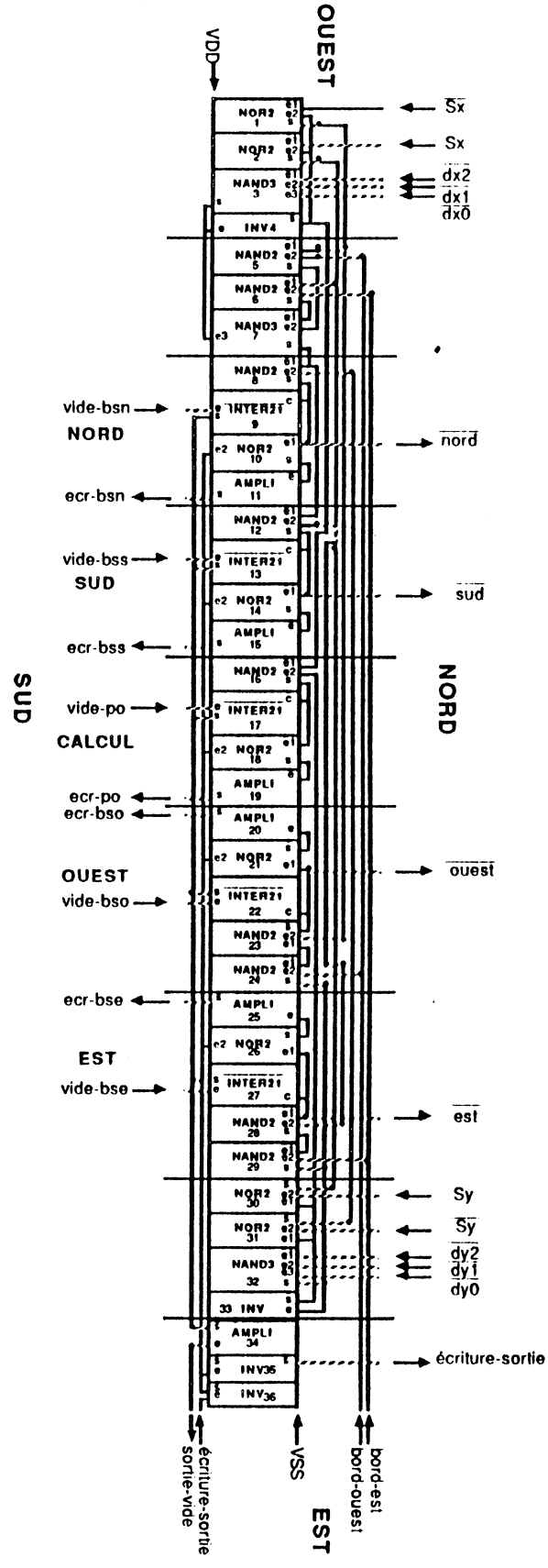


Plan de masse de la partie-traitement de l'aiguilleur

Implantation détaillée



Implantation du registre interne (avec les UALs et leurs commandes)



Implantation de la logique aléatoire

Densité

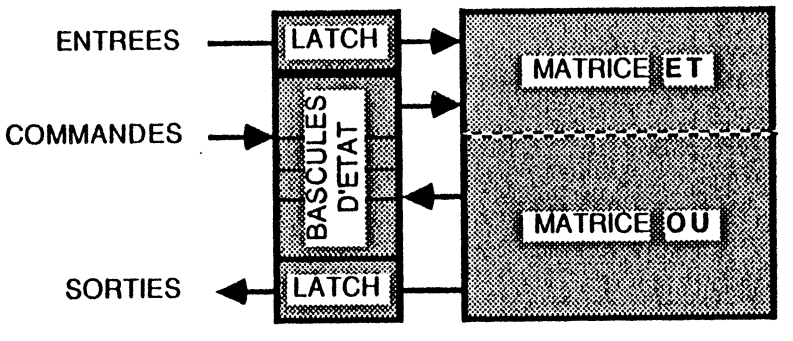
La partie-traitement contient 498 transistors, son enveloppe est un rectangle de 1155 µm par 655 µm soit 0,756 mm². La densité d'intégration est donc de 658 tr/mm².

2.3.5. Le séquenceur de l'aiguilleur

La partie-contrôle de l'aiguilleur a été réalisée à l'aide d'un PLA dont les matrices ET et OU ont été générées automatiquement par le logiciel **GASP** [FLA84]. Les performances de l'aiguilleur dépendent directement des caractéristiques de celui-ci.

2.3.5.1 Caractéristiques du PLA

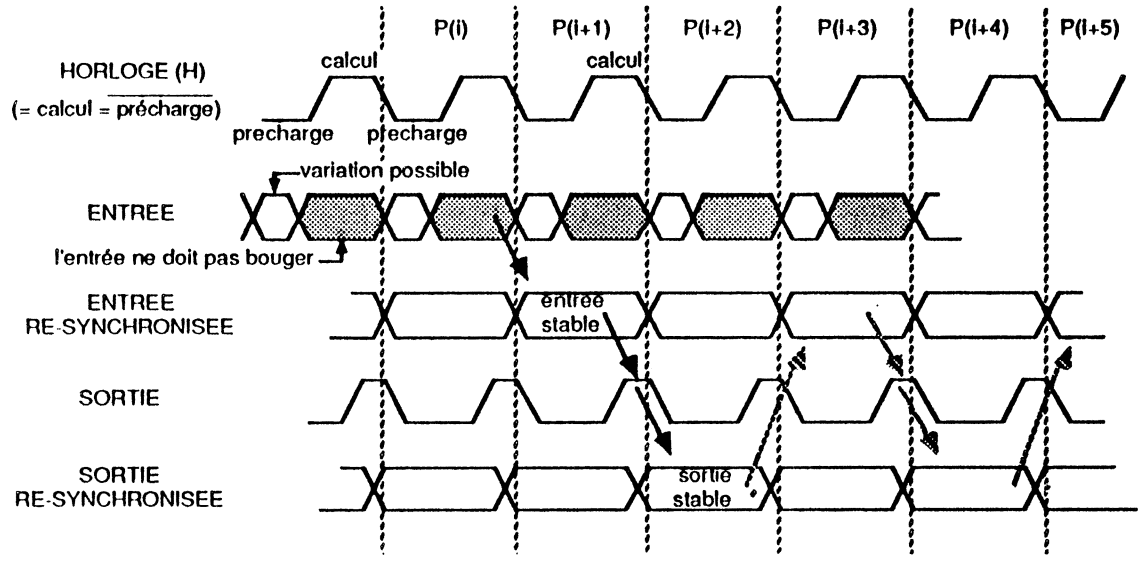
Un PLA (Programmable Logic Array) est constitué de deux matrices dites matrice ET et matrice OU. Une sortie du PLA correspond à une somme de produits (la matrice ET calcule les termes produits et la matrice OU calcule la somme de ces termes). A partir d'un graphe de contrôle établi au format réseau de Pétri, GASP génère un PLA dynamique à précharge (type DOMINO). Les entrées et sorties sont rendues statiques par mémorisation dans des registres maître/esclave.



La phase de calcul du PLA se fait sur le signal d'horloge H, et la précharge sur le signal inversé $\overline{H}$. La synchronisation des signaux d'entrée et de sortie sera donc faite sur H.

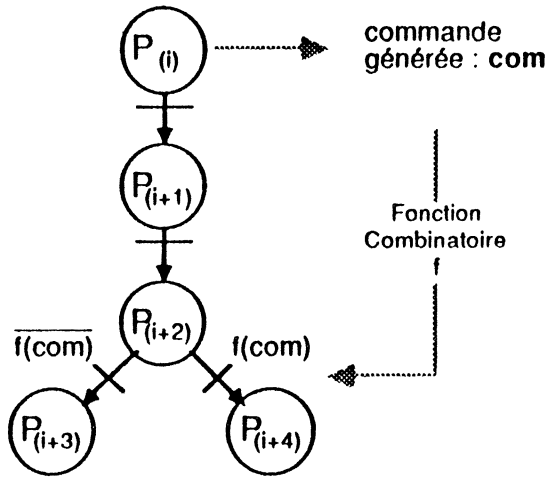
La synchronisation des entrées et des sorties à chaque changement d'état a pour conséquence :

- de retarder d'une période d'horloge la génération effective d'une commande par rapport à l'état qui l'a induit; une commande générée au cours de l'état Pi ne sera effective que sur l'état P(i+1) suivant,
- pour qu'une entrée soit utilisable sur un état, elle doit être stable et sera donc mémorisée au cours de l'état précèdent.



Chronogrammes du PLA

Il en résulte qu'une commande de sortie générée au cours de l'état Pi ne pourra être utilisée comme condition de transition que 2 états plus loin (d'un point de vue temporel).



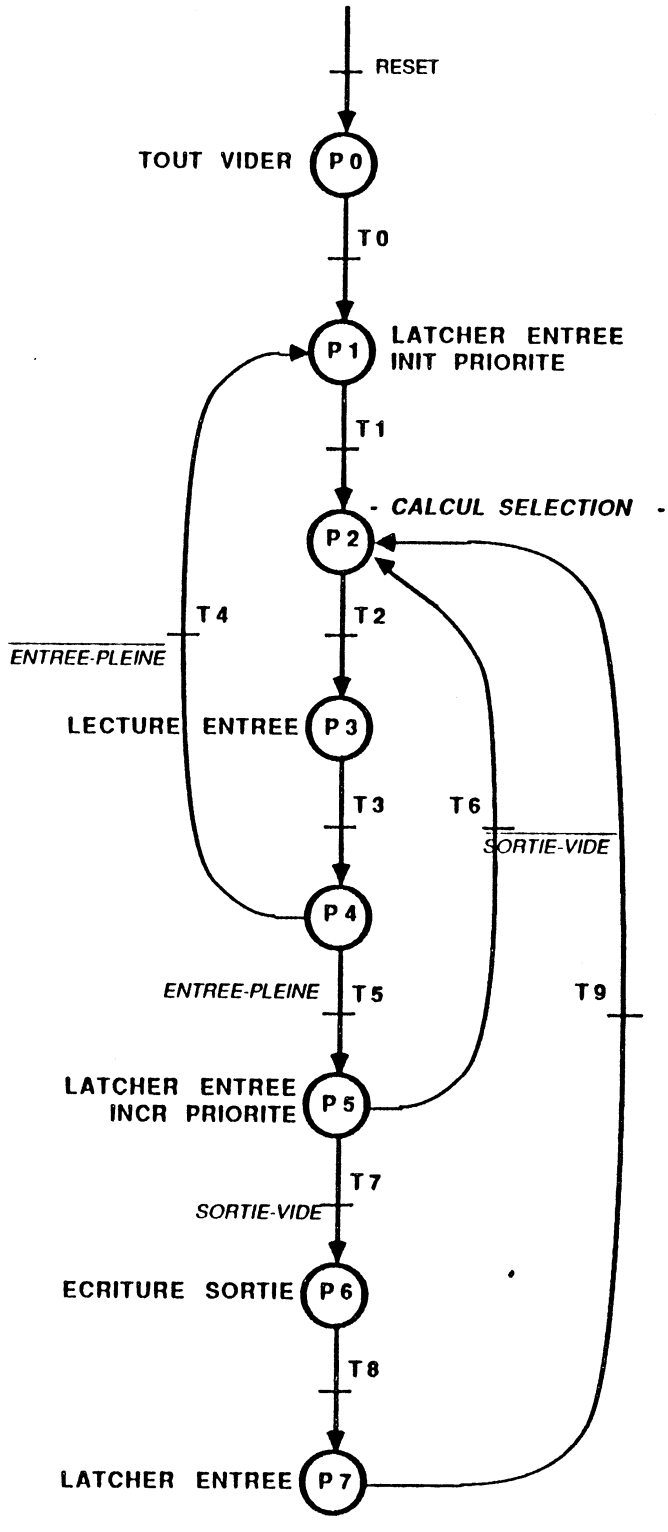
2.3.5.2 Etude du séquenceur

On voit dans l'algorithme de l'aiguilleur (paragraphe 2.3.1.3), que le schéma temporel de séquencement est organisé autour de trois boucles :

- → attente d'une demande de transfert (entrée-pleine)
- → transfert qui a pu être mené à bien (sortie-vide)
- → transfert qui a échoué ($\overline{\text{sortie-vide}}$)

Les 2 boucles de transfert servent également de boucle d'attente d'une demande d'acheminement.

Nous avons conçu le séquenceur de l'aiguilleur en réduisant la longueur temporelle des boucles (pour les performances) et en minimisant le nombre d'états (pour la surface occupée sur le silicium).



La description sous forme de réseau de Pétri du séquenceur comporte :

- 8 places,
- 10 transitions,

- 6 commandes de sortie (tout-vider, init-priorité, incr-priorité, latcher-entrée, lecture-entrée et écriture-sortie),
- 2 signaux d'entrée (entrée-pleine et sortie-vide).

2.3.5.3 Réalisation du PLA

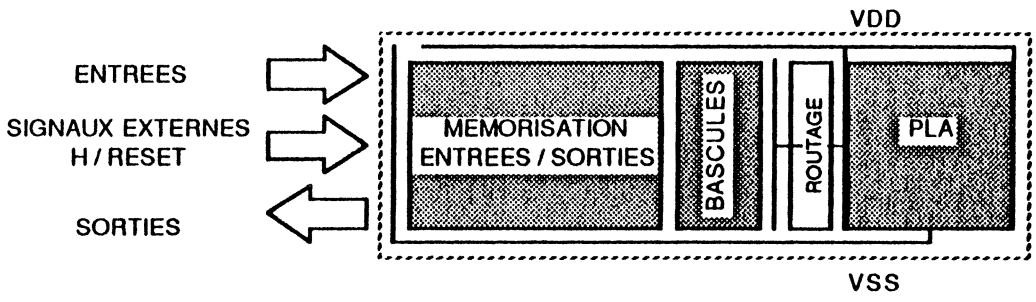
La génération automatique de ce séquenceur par GASP a donné un PLA de neuf monômes, cinq entrées et neuf sorties. Le codage des huit états se fait sur trois bits. Les équations des commandes et des états ainsi que le codage de ceux-ci sont donnés en annexe.

2.3.5.4 Implantation micron

Le bloc PLA de l'aiguilleur est formé :
- du PLA proprement dit (matrices ET et OU),
- des registres maître/esclave pour la mémorisation des états (3),
- des entrées / sorties (8).

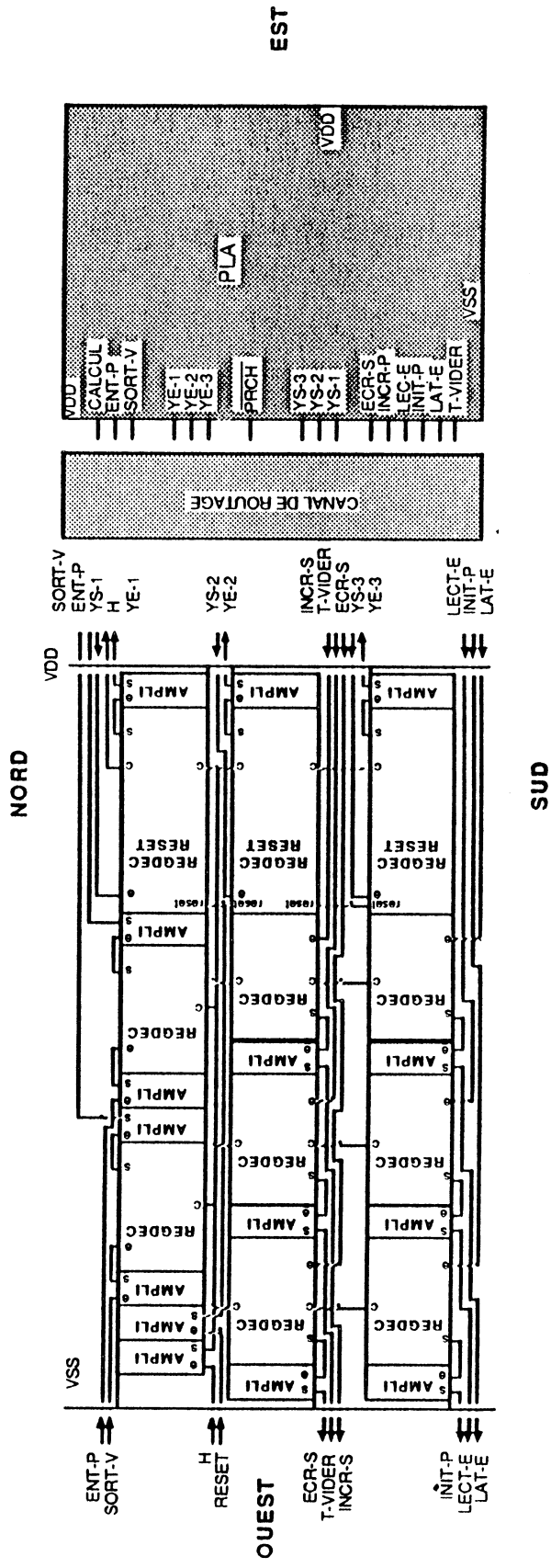
Organisation des entrées/sorties

De façon à simplifier les interconnections (avec l'encodeur de priorité et la partie-traitement de l'aiguilleur), nous avons regroupé toutes les entrées ('entre-pleine', 'sortie-vide', ainsi que les signaux d'horloge et de reset), et les sorties ('tout-vider', 'init-priorité', 'incr-priorité', 'latcher-entrée', 'lecture-entrée' et 'écriture-sortie') le long de la frontière ouest.



Plan de masse du PLA

Implantation du PLA

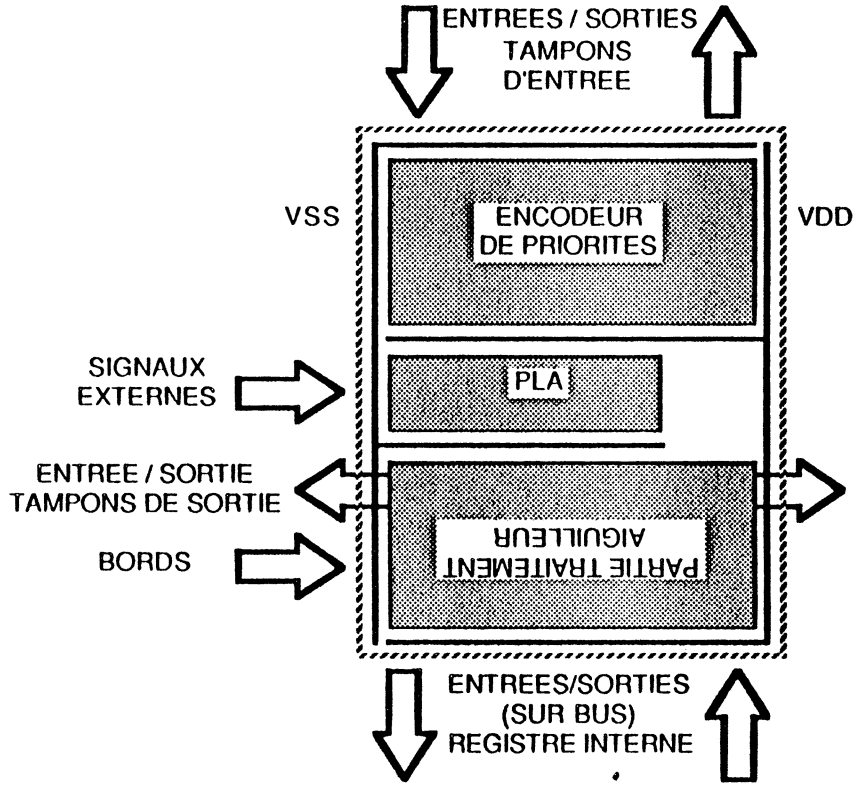


Densité

La partie PLA contient 427 transistors, son enveloppe est un rectangle de 990 µm par 390 µm soit 0,386 mm². La densité d'intégration est donc de 1106 T/mm².

2.3.5.5 Assemblage de l'aiguilleur

L'encodeur de priorité, le bloc PLA et la partie-traitement de l'aiguilleur ont été conçus pour être superposés. L'assemblage de ces 3 blocs forme l'aiguilleur qui occupe la moitié droite de la cellule.



Plan de masse de l'aiguilleur

Densité

L'aiguilleur contient 1433 transistors, son enveloppe est un rectangle de 1210 µm par 1672 µm soit 2,023 mm². La densité d'intégration est donc de l'ordre de 708 tr/mm².

2.4 La partie traitement de la cellule

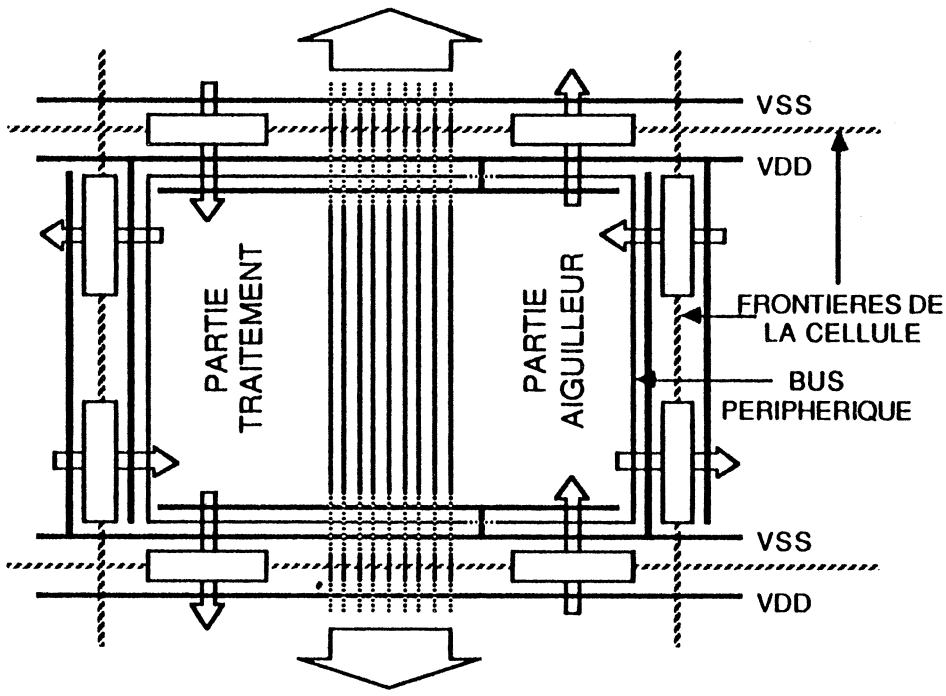
Cette partie est décrite dans [COR88].

2.5 Assemblage de la cellule

Les tampons internes de la cellule ont été intégrés à la partie traitement, exceptés les drapeaux des quatre tampons de sortie de

la partie-traitement. Pour des questions de topologie et de surface, ces drapeaux ont été implantés au-dessus de l'aiguilleur. Les cellules reçoivent de l'ordinateur-hôte les signaux d'horloge H1 et H2, d'initialisation RESET1 et RESET2 de synchronisation DEBUT, et émettent chacune un signal FIN. Ces signaux traversent les cellules verticalement et délimitent, avec le bus, les parties aiguilleur et traitement de la cellule.

La distribution des alimentations est fonction de l'organisation en réseau des cellules. Dans l'ordre des priorités des signaux, nous avons choisi d'abord les alimentations, puis les horloges et les commandes-maîtres du réseau, le bus et enfin les commandes internes.



SIGNAUX GENERAUX DE SIMULATION
GO/END/END/H1/RESET1/RESET2/H2/B.EST/B.OUEST

Plan de masse des alimentations de la cellule

Remarque : le seul niveau de métal ne permet pas de densifier les canaux d'interconnexions, qui occupent une grande surface.

Densité

La cellule comporte 7541 transistors (y compris les huit tampons périphériques, 6421 si l'on en compte que quatre), son enveloppe est un rectangle de 4109 µm par 3186 µm soit 13,09 mm^2. La densité d'intégration est donc de 576 tr/mm^2.

3 LE CIRCUIT COMPLET

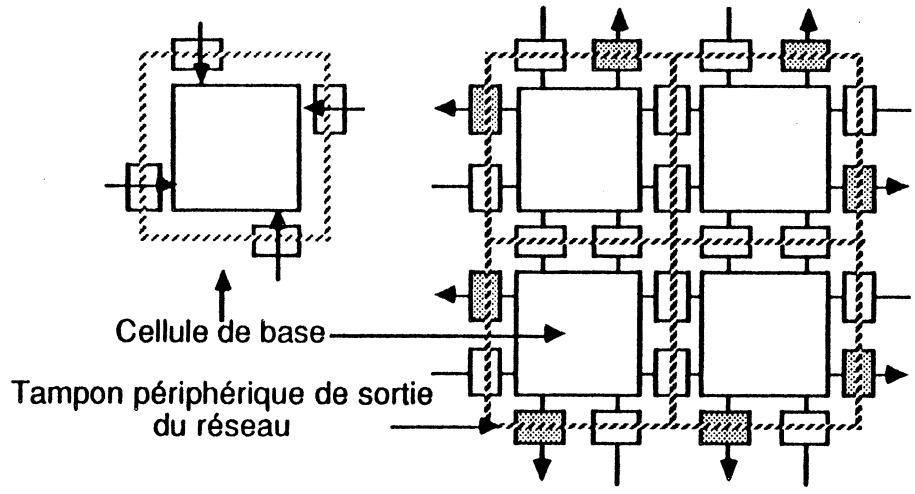
Il intègre un réseau 2x2, et huit interfaces de communication.

3.1 Le réseau 2x2

Quatre cellules ont pu être implantées en un réseau 2x2. Cette limite nous était imposée par la taille maximale du circuit.

Assemblage

Les cellules de base intègrent leurs quatre tampons périphériques d'entrée. Leur organisation en matrice se fait par juxtaposition des enveloppes. A cet assemblage, il faut encore ajouter les huit tampons périphériques de sortie du réseau, ainsi que le regroupement des signaux FIN émis par les cellules en un ET logique.



Assemblage des cellules en réseau

A ce stade, le circuit comporte trop d'entrées et de sorties pour être broché.

3.2 Les interfaces

Pour réduire le nombre de "pattes" du réseau et pour lui permettre de communiquer facilement et efficacement avec le monde extérieur, un moyen simple est d'utiliser des interfaces de communication (interface parallèle → série pour les tampons de

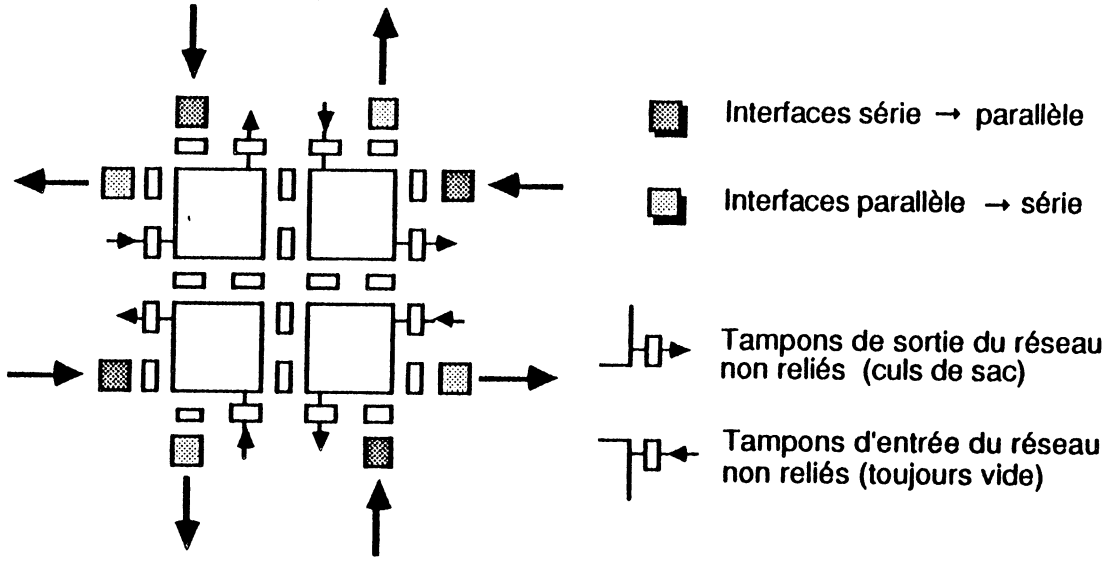
sortie du réseau, et série → parallèle pour ceux d'entrée). Ces interfaces en fait fonctionnent par couples (parallèle/série et série/parallèle) en relation avec deux tampons, un d'entrée et un de sortie. La description de ces interfaces est présentée dans [COR88].

Il est évident que ces interfaces ralentissent le flot des données. En effet, la vitesse de transfert est liée à l'horloge commune de ces deux interfaces, ainsi qu'au plus lent des deux intervenants (c'est notamment le cas lors de communication réseau/ordinateur-hôte).

Pour des raisons de taille de circuit imposée par le multiplexage des tranches, nous n'avons pu intégrer que 8 de ces interfaces, sur les 16 tampons périphériques du réseau. Afin de pouvoir programmer des réseaux plus grands, nous avons réparti ces interfaces aux 4 coins de notre réseau 2x2. Nous avons implanté 4 interfaces série/parallèle sur les tampons d'entrée, et 4 interfaces parallèle/série sur les tampons de sortie.

Ainsi, en tenant compte des tampons "culs-de-sac" (non reliés à une interface), lors du placement des portes, on peut interconnecter sans problème de communication, ces réseaux 2x2.

Parmi les 8 tampons périphériques restant, les quatre de sortie n'ont pas été câblés et aboutissent sur des pavés métalliques où l'on pourra tester sous pointes leur état logique. Quant aux tampons d'entrée, nous avons pré-câblé leurs entrées afin de servir de vecteurs de test.



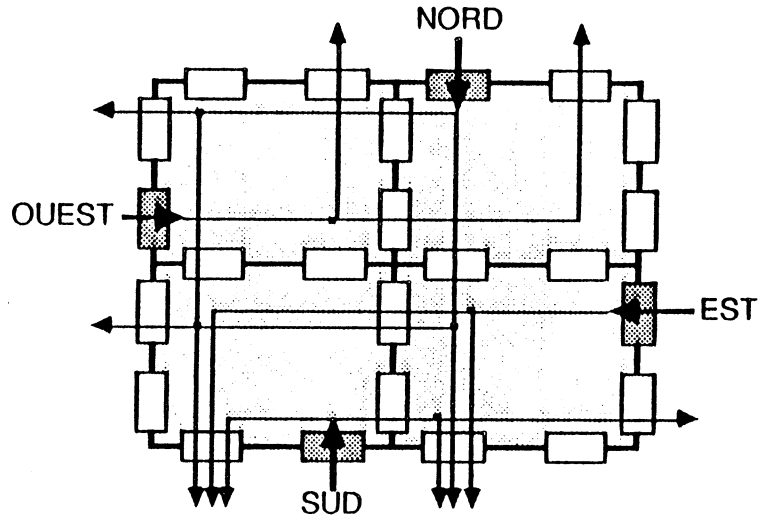
Organisation topologique des interfaces

3.3 Les vecteurs de test pré-câblés

Afin de pouvoir tester fonctionnellement les cellules du réseau 2x2, nous avons câblé, en entrée du réseau, les messages suivants :

TAMPONS D'ENTREE	MESSAGE			BITS DE MESSAGE									
	DX	DY	VAL	SX	DX2	DX1	DX0	SY	DY2	DY1	DY0	Val2	Val1
NORD	-5	2	0	1	1	1	0	0	0	1	0	0	0
SUD	2	5	1	0	0	1	0	0	1	1	0	0	1
EST	-1	1	1	1	0	0	1	0	0	0	1	0	1
OUEST	1	-1	2	0	0	0	1	1	0	0	1	1	0

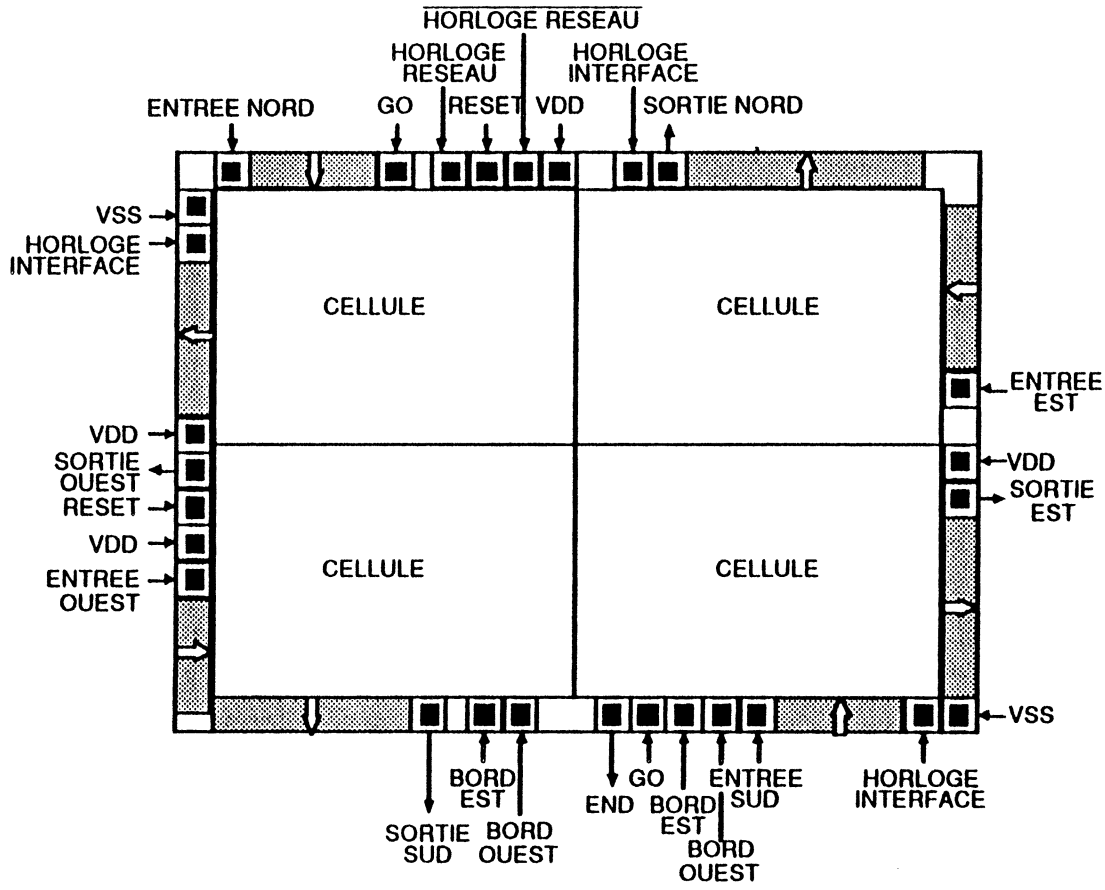
En jouant sur les signaux BORD-EST et BORD-OUEST, on peut lors d'un test sous pointes, vérifier le bon fonctionnement de l'aiguilleur de chacune des cellules du réseau.



Les différents cheminements possibles des messages de test

3.4 Le brochage du circuit

Le circuit complet comporte 28 plots.



Brochage du circuit

3.5 Densité et performances

Le circuit complet comporte 32.639 transistors, son enveloppe est un rectangle de 8852,5 µm par 7067,5 µm soit 62,5 mm². La densité d'intégration est donc de 522 tr/mm².

Le densité finale d'intégration est faible (522). Cela représente environ la moitié de ce que l'on pourrait attendre pour un circuit CMOS. Plusieurs facteurs ont contribué à ce résultat :

- la nature même du circuit dans lequel les communications ont une grande importance,
- la technologie utilisée, avec ce deuxième niveau de métal qui nous a fait défaut.

Enfin, la conception à partir d'une bibliothèque de cellules n'a jamais donné d'aussi bons résultats (en terme d'intégration) que la méthode manuelle.

Performances

De par sa conception, les performances du réseau sont :
- proportionnelles au nombre de cellules qu'il intègre,
- dépendantes de la localité des cellules (distance maximale des interconnections entre cellules),
- dépendantes également du temps de calcul de la partie-traitement de la cellule.

Les messages étant acheminés en parallèle et l'évaluation des portes étant faite au fur et à mesure de la réception des entrées, la durée d'un pas de simulation est donc majorée :
- par le temps d'acheminement de message le plus long -$T_{ache\text{-}max}$,
- augmenté du temps de traitement du message transmis- T_{cal}.

Dans un réseau NxN, l'acheminement d'un message passe par un maximum de 2.(N-1) cellules (pire cas). En supposant que le message traverse immédiatement la moitié de ces cellules, et attend un cycle avant de traverser l'autre moitié, on a donc :

$$T_{ache\text{-}max} = (N-1).T_{trav\text{-}im} + (N-1).T_{att\text{-}trav}$$

On obtient donc la durée d'un pas de simulation :

$$T_{pas} = (N-1).T_{trav\text{-}im} + (N-1).(T_{trav\text{-}im} + T_{att\text{-}trav}) + T_{cal}$$

avec
$T_{trav\text{-}im} = 300ns$
$T_{att\text{-}trav} = 250ns$
$T_{cal} = 150ns$

Remarque : ces temps ont été déterminés par les simulations et sont directement liés à la technologie utilisée. Des évaluations de performances indépendantes de la technologie sont données plus loin.

on a donc $T_{pas} = 850.(N-1) + 150$ ns où N est la taille du réseau

Pour N=100, T_{pas} = 84300ns. On voit par ce simple calcul, que la phase de placement des portes sur le réseau est très importante.

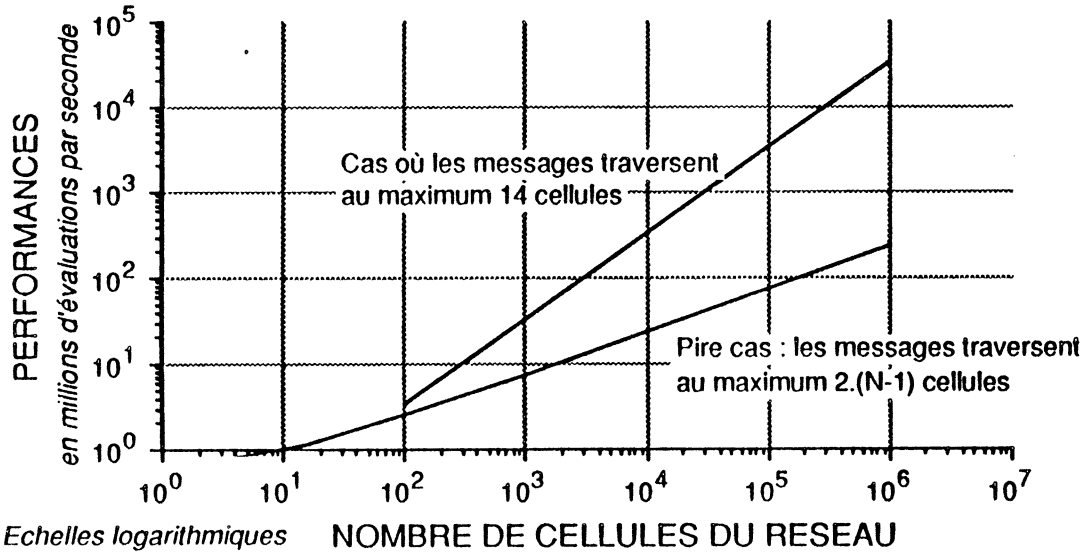
Dans l'hypothèse où, quelle que soit la taille du réseau, l'acheminement d'un message est au maximum de 8, c'est-à-dire que la zone couverte par une cellule est un "cercle" de rayon 7 (voir chap. 2, §3.3.2), le nombre maximum de cellules traversées par un message au cours d'un pas de simulation est alors de 14 (7 horizontalement et 7 verticalement). La durée d'un pas de simulation est alors de :

$$T_{pas} = 7.T_{trav\text{-}im} + 7.(T_{trav\text{-}im} + T_{att\text{-}trav}) + T_{cal}$$

d'où T_{pas} = 6100ns

Un réseau 100x100 simule donc 10^4 portes entre 6,1 et 84,3 ms ce qui donne une vitesse théorique de simulation comprise entre 118.10^6 et $1{,}6.10^4$ éval/sec. Compte tenu d'une activité du circuit à simuler de l'ordre de 20%, on obtient des performances de simulation comprises entre 23 et 327 millions d'évaluations par seconde.

Le graphique ci-dessous montre les performances du simulateur logique cellulaire (activité moyenne de 20%) dans les cas où le nombre de cellules traversées par les messages est :
- au maximum de 14,
- au maximum égal à 2.(N-1), où N est la taille du réseau (carré).



Performances du simulateur logique cellulaire (activité de 20%)

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). Il faut néanmoins remarquer que les performances du réseau cellulaire peuvent être dégradées par les communications avec le système-hôte.

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-2) et 1 micron (CMOS-1). Dans le cas de notre circuit, on estime le gain supplémentaire de 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 2 et 1 microns du CNET et les estimations de la densité d'intégration de notre circuit sur cette dernière.

	Commutation inverseur (ns)	Densité de la techno (mos/mm2)	Surface du circuit (mm2)	Densité du circuit
CMOS-2 (1 métal 2µ)	0,5	1000	62,5	522
CMOS-1 (2 métaux 1µ)	0,25	3000	16,7	1954
CMOS-1 (industrielle)	0,25	4000	8,2	4000

Comparaison des performances entre les technologies
CMOS-2 et CMOS-1 du CNET

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, une technologie à 1 micron atteint une densité d'intégration de 4000 mos/mm^2. Avec une telle densité, la taille de notre circuit serait de 8,2 mm^2 et l'on pourrait intégrer un réseau 6x6 sur une surface comparable à celle qu'il occupe actuellement.

CHAPITRE 3

INTEGRATION DANS UN SYSTEME HOTE

TABLE DES MATIERES

Pour être opérationnel, le circuit cellulaire doit être relié à un ordinateur-hôte chargé de piloter la simulation. Le choix de celui-ci est important, car ses caractéristiques vont intervenir sur les performances globales de la machine de simulation logique. En effet, le circuit cellulaire peut ne pas être exploité de façon optimale lorsque les entrées/sorties du réseau ne sont pas réalisées suffisamment vite (écriture des stimulis de simulation dans les tampons d'entrées et lecture des résultats de la simulation en cours dans les tampons de sortie).

Ce chapitre décrit l'interface entre le circuit cellulaire et un ordinateur-hôte à travers un bus de données. Les problèmes de synchronisation sont présentés et les algorithmes de communication assurant les transferts d'informations (ordinateur-hôte ↔ interfaces-réseau) sont donnés. Le choix de l'ordinateur-hôte est discuté et la réalisation de la carte présentée. Enfin, les performances de la machine cellulaire de simulation logique sont étudiées.

1 L'INTERFACE BUS

Le circuit cellulaire étant autonome, le problème de son intégration sur une carte d'extension d'un ordinateur-hôte est celui de la communication entre les interfaces périphériques du réseau (tampons d'entrée et sortie) et l'ordinateur-hôte.

Ces communications sont établies à travers les signaux du connecteur d'extension disponible "en fond de panier" de l'ordinateur-hôte. Ces signaux (pour l'essentiel, ceux du microprocesseur) sont regroupés en trois catégories :

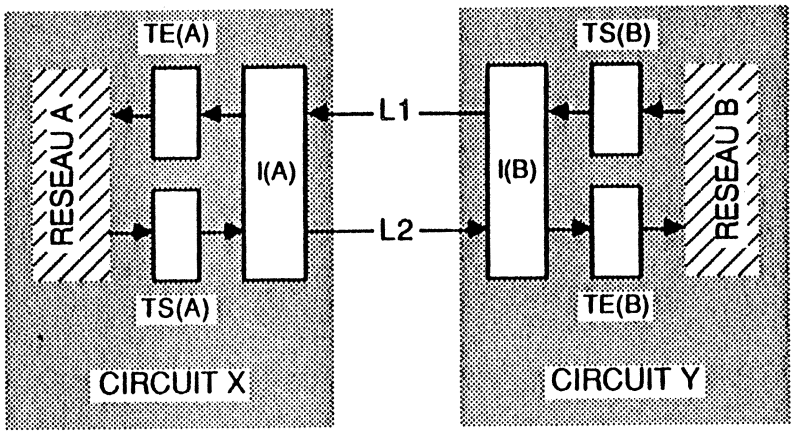
- les lignes bidirectionnelles de données,
- les lignes d'adresses,
- les lignes de commandes et de contrôles et particulièrement :
  - → les commandes mémoires (lecture, écriture),
  - → les horloges,
  - → les alimentations,
  - → les lignes d'interruptions,
  - → les canaux de DMA.

La difficulté est ici de gérer des communications entre deux systèmes totalement asynchrones.

Les interfaces parallèle ↔ série du réseau

Chaque circuit intégré du réseau cellulaire est muni d'interfaces qui permettent de les connecter directement à des circuits de même type et qui doivent ainsi permettre la connexion avec le système-hôte. Ces interfaces sont détaillées dans [COR88]. Rappelons en rapidement le principe :

Les informations contenues en parallèle dans les tampons de sortie sont transférées en série entre un émetteur (convertisseur parallèle → série) et un récepteur (convertisseur série → parallèle). Ces interfaces fonctionnent toujours par paires de couples (émetteur/récepteur d'un côté, émetteur/récepteur de l'autre). Ce couple d'interfaces gère deux lignes de communication distinctes véhiculant aussi bien des données que des commandes.



I(A)	: Interface d'entrée/sortie du réseau A
TE(A)	: Tampon d'entrée du réseau A
TS(A)	: Tampon de sortie du réseau A
I(B)	: Interface d'entrée/sortie du réseau B
TE(B)	: Tampon d'entrée du réseau B
TS(B)	: Tampon de sortie du réseau B

Architecture des interfaces réseau

Toutes ces communications sont synchronisées par une même horloge selon le protocole suivant :

→ par défaut, les lignes L1 et L2 sont au niveau bas (0),

→ la séquence 0.1.1, envoyée par l'interface I(A) sur L2, est interprétée par I(B) comme l'écriture dans TS(A) d'un message (remplissage). Les 10 bits de données de celui-ci sont alors envoyés immédiatement après la séquence de synchronisation. Le transfert se termine par la mise au niveau bas de la ligne L2,

→ la séquence 0.1.0 sur la même ligne L2 sera interprétée comme la lecture par le réseau A du message contenu dans le tampon TE(A). La séquence est également suivie par la mise au niveau bas de la ligne L2.

Le même protocole est appliqué aux transferts de messages du réseau B vers le réseau A entre I(B) et I(A) sur L1.

PROTOCOLE DE COMMUNICATION
sur L1 (I(B) -> I(A)) : - 011 TE(B) se vide - 010 TS(B) se remplit sur L2 (I(A) -> I(B)) : - 011 TE(A) se vide - 010 TS(A) se remplit

Protocole de communication entre interfaces-réseau

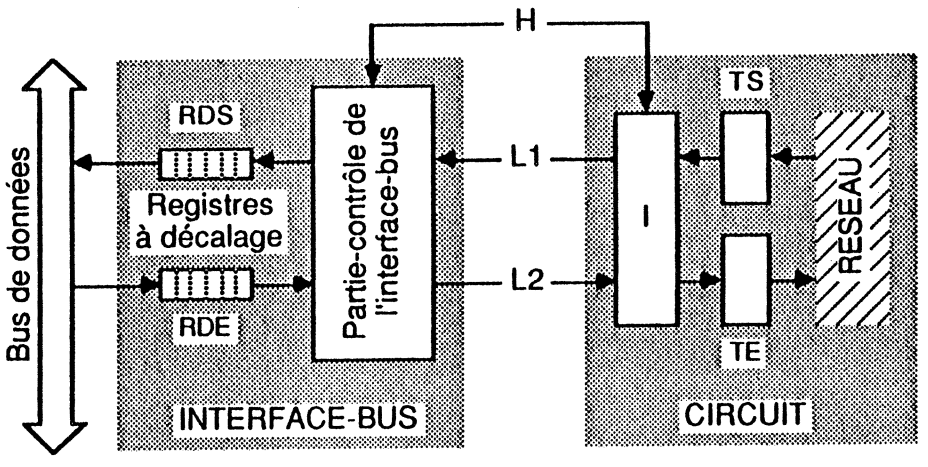
Remarque : l'horloge H doit être d'une fréquence inférieure à celle du réseau.

1.1 Organisation générale de l'interface-bus

1.1.1 Un transfert en deux temps

Les interfaces-réseau ont été étudiées pour des transferts entre systèmes parfaitement synchrones (l'horloge est commune aux deux interfaces). Ceci n'est plus vrai dans le cas de communication réseau ↔ ordinateur-hôte car ici, l'émetteur ne fonctionne plus à la même vitesse que le récepteur.

Pour synchroniser le transfert, la technique la plus simple consiste à mémoriser le message à transmettre dans un registre à décalage intermédiaire de dix bits. Ce registre est chargé à une certaine fréquence et déchargé à une autre. Dans le cas de la sortie d'un message du réseau (transfert réseau → ordinateur-hôte), la première partie du transfert (interface-réseau → registre à décalage de sortie) est contrôlée par l'interface-bus. La seconde partie du transfert (registre-à-décalage → bus-de-données) est contrôlée par l'ordinateur-hôte. Les rôles sont inversés dans le cas d'un message d'entrée du réseau.



I	: Interface d'entrée/sortie du réseau
TE	: Tampon d'entrée du réseau
TS	: Tampon de sortie du réseau
RDE	: Registre à décalage d'entrée
DRS	: Registre à décalage de sortie

PROTOCOLE DE COMMUNICATION

sur L1 (I(A) -> I(BUS)) :
- 011 TE(A) se vide
- 010 TS(A) se remplit

sur L2 (I(BUS) -> I(A)) :
- 011 RDS est vidé
- 010 RDE est rempli

Architecture générale de l'interface-bus

1.1.2 La partie-contrôle de l'interface-bus

La partie-contrôle de l'interface doit gérer les entrées et les sorties du réseau, qui sont à priori aléatoires surtout dans le cas de sortie, selon le protocole des interfaces inter-réseau. Nous avons donc été amené à différencier la partie-contrôle des entrées de la partie-contrôle des sorties.

1.2 Description de l'interface-bus

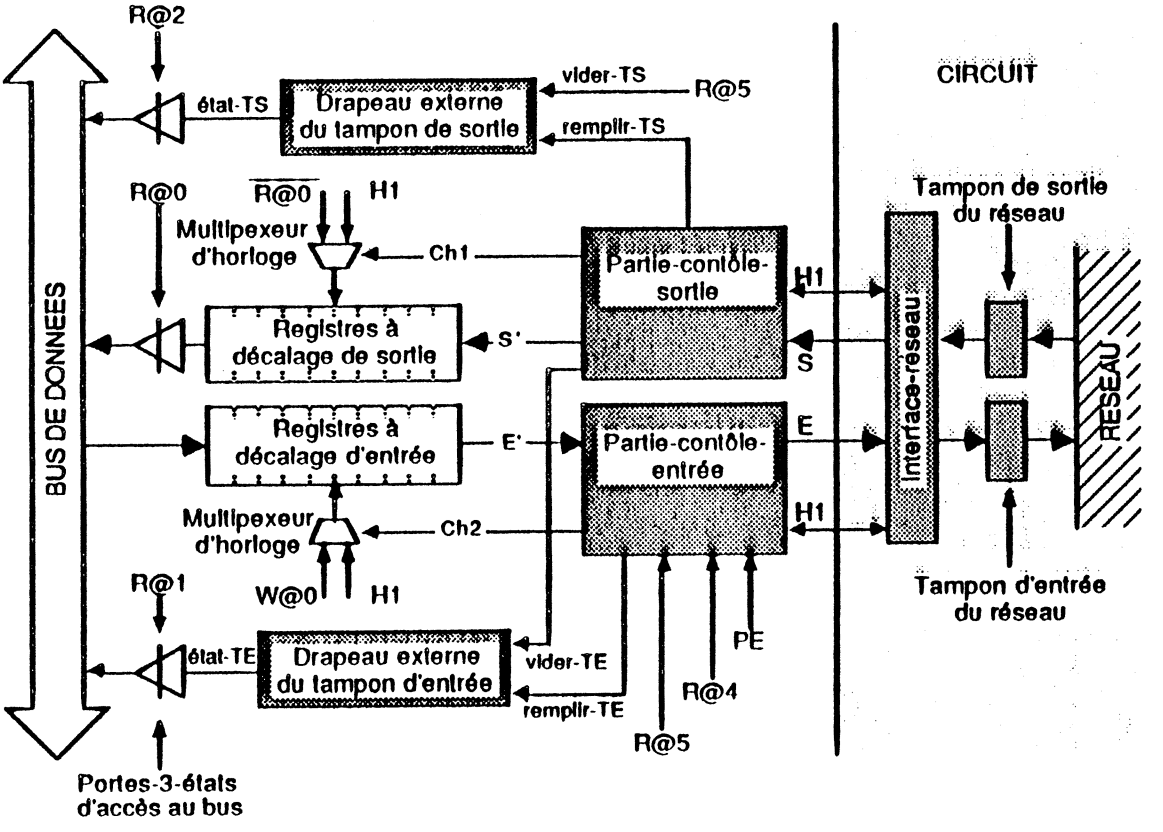
Pour un couple d'entrée/sortie du réseau (une interface-réseau), l'interface-bus complète est formée de :

→ une partie contrôle-entrée,

→ une partie contrôle-sortie,

→ 2 registres-à-décalage de 10 bits (1 pour l'entrée et 1 pour la sortie),

→ 2 bascules RS mémorisant l'état des tampons d'entrée et de sortie du réseau (drapeaux externes).



Architecture de l'interface-bus

1.2.1 Les adresses-mémoires

L'ordinateur-hôte programme et contrôle le réseau à partir de huit adresses accessibles en lecture ou en écriture.

Adresse @0

C'est l'adresse commune de tous les registres à décalage d'entrée et de sortie des interfaces-bus. Ces registres sont lus et écrits en parallèle, chaque ligne du bus de données étant affectée à un couple d'entrée/sortie d'une même interface-réseau. La lecture de cette adresse (signal R@0) concerne les registres à décalage de sortie. L'écriture à cette adresse (signal W@0) concerne elle, les registres à décalage d'entrée.

L'intérêt de cette technique est son faible coût en adresse-mémoire, mais limite le nombre des entrées et des sorties du réseau au nombre de bits du bus de données.

Adresse @1

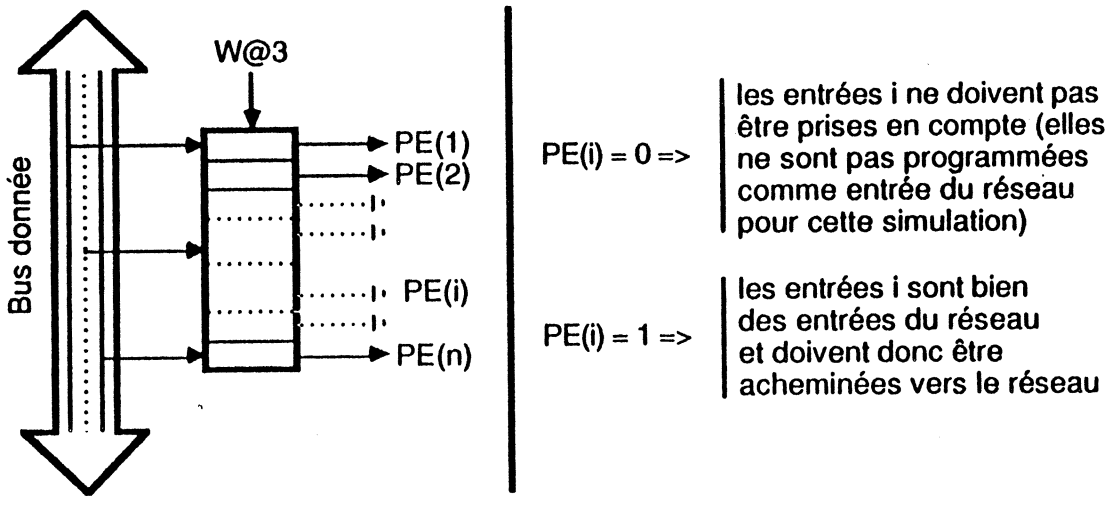
L'ordinateur-hôte peut interroger à cette adresse l'état des tampons d'entrée du réseau. Derrière cette adresse, on trouve donc les bascules RS. Son accès se fait donc uniquement en lecture.

Adresse @2

Elle joue le même rôle que l'adresse précédente mais pour les tampons de sortie.

Adresse @3

A l'adresse @3 se trouve un tampon mémorisant les prises en compte des tampons d'entrée (PE). Les registres à décalage d'entrée étant tous chargés en parallèle, il est nécessaire que les parties contrôle-entrée de chaque interface sachent si elles doivent traiter les données contenues dans ces registres comme une entrée (et donc les transmettre au réseau) ou non.



Mécanisme des prises en compte des entrées

Les prises en compte sont initialisées à 1 (vrai) avant la phase de programmation du réseau, puis ré-initialisées, à la fin de celle-ci, en fonction de l'entrance du circuit qui va être simulé.

Le tampon mémorisant les prises en compte (adresse @3) n'est adressé qu'en écriture (signal W@3).

Remarque : il n'y a pas de prise en compte des tampons de sortie, car les messages de sortie sont traités par l'ordinateur-hôte.

Adresse @4

Cette adresse est un interrupteur logiciel. Le signal de lecture de l'adresse @4 (R@4) est interprété par toutes les parties contrôle-entrée comme la fin du remplissage des registres à décalage d'entrée. L'envoi des stimulis d'entrée au réseau se fait en trois temps :

→ l'ordinateur-hôte initialise les registres à décalage d'entrée,

→ l'ordinateur-hôte indique aux parties contrôle-entrée que les registres à décalage d'entrée sont pleins,

→ les parties contrôle-entrée, des entrées programmées comme telles (PE=1), transfèrent vers les tampons d'entrée du réseau les informations des registres à décalage.

Adresse @5

Cette adresse a le même rôle, avec les sorties, que l'adresse précédente avec les entrées. Par un accès en lecture à cette adresse, l'ordinateur-hôte indique aux parties contrôle-entrée qu'il vient de lire les registres à décalage de sortie. Ceux-ci devront alors répercuter cette lecture :

→ en vidant les tampons de sortie du réseau (séquence 0.1.1),

→ en vidant les drapeaux externes de ces mêmes tampons.

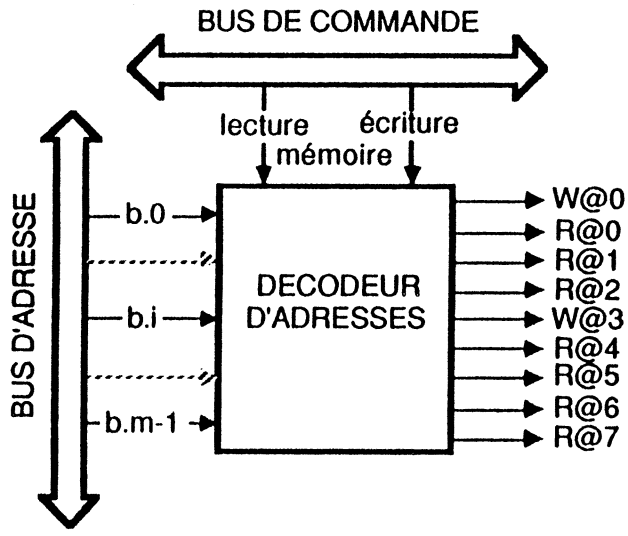
Adresse @6

L'accès en lecture à cette adresse (R@6) génère le signal RESET d'initialisation du réseau (RESET=R@6). Ce même signal initialise également les différentes parties-contrôle des interfaces-bus. Il s'agit là aussi d'un interrupteur logiciel.

Adresse @7

L'accès en lecture à cette adresse (R@7) génère le signal DEBUT qui est distribué dans le réseau (DEBUT=R@7).

Remarque : il n'y a pas d'adresse réservée à la lecture du signal FIN émis par le réseau. Pour éviter la scrutation par l'ordinateur-hôte d'une adresse mémoire, le signal FIN est câblé comme interruption.



Décodage du bus d'adresses et de commandes

1.2.2 Les registres à décalage

Ils sont en fait, des tampons d'entrée et de sortie externes.

Registre à décalage d'entrée

L'ordinateur-hôte peut physiquement les remplir à tout moment, sauf lorsque les parties contrôle-entrée transfèrent leurs contenus vers les tampons d'entrée du réseau. En fait, l'ordinateur-hôte les remplit au début de chaque cycle de simulation avec les stimulis d'entrée.

Chaque ligne du bus de données est affectée à une entrée/sortie du réseau. Le chargement de données (effectué en parallèle pour tous les registres à décalage) est séquencé par le signal W@0 d'écriture à l'adresse de ces registres. Le déchargement de ces données vers l'interface-réseau est séquencé par la partie contrôle-entrée. L'interface-réseau partage avec la partie contrôle-entrée la même horloge (H1) commune à toutes les interfaces-bus.

Les registres à décalage de sortie

Ils sont remplis par le réseau, c'est-à-dire n'importe quand après la lecture par celui-ci des messages d'entrée. Leurs chargements sont contrôlés par les parties contrôle-sortie de la même façon que les déchargements des registres à décalage d'entrée

sont contrôlés par les parties contrôle-entrée. Leurs déchargements sont réalisés par l'ordinateur-hôte et séquencés par le signal $\overline{R@0}$ de lecture à l'adresse de ces registres.

Les registres à décalage de sortie sont connectés au bus de données, à travers des portes-trois-états commandées par le signal R@0.

1.2.3 Le multiplexage des horloges (registres à décalage)

Ces boîtiers servent à basculer les commandes d'horloge des registres à décalage. Les registres à décalage d'entrée sont commandés par :

→ W@0 lors du chargement,

→ H1 lors du déchargement afin de respecter le protocole de communication synchrone avec les interfaces-réseau.

Les registres à décalage de sortie sont commandés par :

→ H1 lors du chargement (même remarque que précédemment),

→ $\overline{R@0}$ lors du déchargement.

Une fois chargées, les données sont bloquées dans les registres à décalage par une horloge au niveau bas (R@0 et $\overline{W@0}$).

Le multiplexeur d'horloge du registre à décalage d'entrée (resp. de sortie) est commandé par le signal CH3 (resp. CH2) de la partie contrôle-entrée (resp. -sortie).

1.2.4 La partie contrôle-sortie

Elle contrôle :

→ le transfert de message entre l'interface-réseau et le registre à décalage de sortie,

→ la mise à jour du drapeau externe du tampon de sortie (signal remplir) et celui du tampon d'entrée (signal vider).

Cette partie-contrôle reçoit la même horloge que l'interface-réseau et la partie contrôle-entrée, sa seule entrée est la ligne de communication L1 avec cette même interface. Elle commande :

→ la ligne de communication L1' vers le registre à décalage de sortie

→ l'horloge H2 de chargement/déchargement du même registre à décalage de sortie à travers un multiplexeur (commande CH2),

→ le signal VIDER du drapeau (externe) associé au tampon d'entrée,

→ le signal REMPLIR du drapeau (externe) associé au tampon de sortie.

1.2.5 La partie contrôle-entrée

Elle contrôle :

→ le transfert de message entre le registre à décalage d'entrée et l'interface-réseau,

→ la mise à jour du drapeau (externe) associé au tampon d'entrée.

La partie contrôle-entrée travaille en entrée sur la ligne de communication L2', sur les commandes de lecture d'adresses R@4 et R@5 ainsi que les prises en compte des tampons d'entrée PE.

En sortie, la partie contrôle-entrée :

→ gère la ligne de communication L2 vers l'interface réseau,

→ commande le signal REMPLIR du drapeau (externe) associé au tampon d'entrée.

1.2.6 Les drapeaux externes des tampons

Ces drapeaux indiquent l'état des tampons d'entrée et de sortie de chaque interface-réseau. Leur mise à jour est en phase avec celle des drapeaux internes du réseau. Ils sont interrogés par l'ordinateur-hôte afin de contrôler la simulation.

Les drapeaux externes des tampons d'entrée sont vidés par les parties contrôle-sortie (sur ordre des interfaces-réseau (suite à la lecture du tampon d'entrée par le réseau)) et remplis par la partie contrôle-entrée.

Les drapeaux externes des tampons de sortie sont vidés directement par l'ordinateur-hôte (après lecture des registres à décalage de sortie) et remplis par la partie contrôle-sortie.

Remarque : les sorties des bascules mémorisant les drapeaux sont connectées au bus de données via une porte-trois-états. Ces sorties sont commandées pour les drapeaux associés aux tampons d'entrée (resp. de sortie) par le signal R@1 (resp. R@2).

2 LES ALGORITHMES

Les parties contrôle-entrée et les parties contrôle-sortie déroulent en parallèle un algorithme séquentiel. L'ordinateur-hôte pilote la simulation en remplissant et vidant les registres à décalage, à l'aide des drapeaux externes des tampons d'entrée/sortie et des signaux de contrôle RESET, DEBUT et FIN.

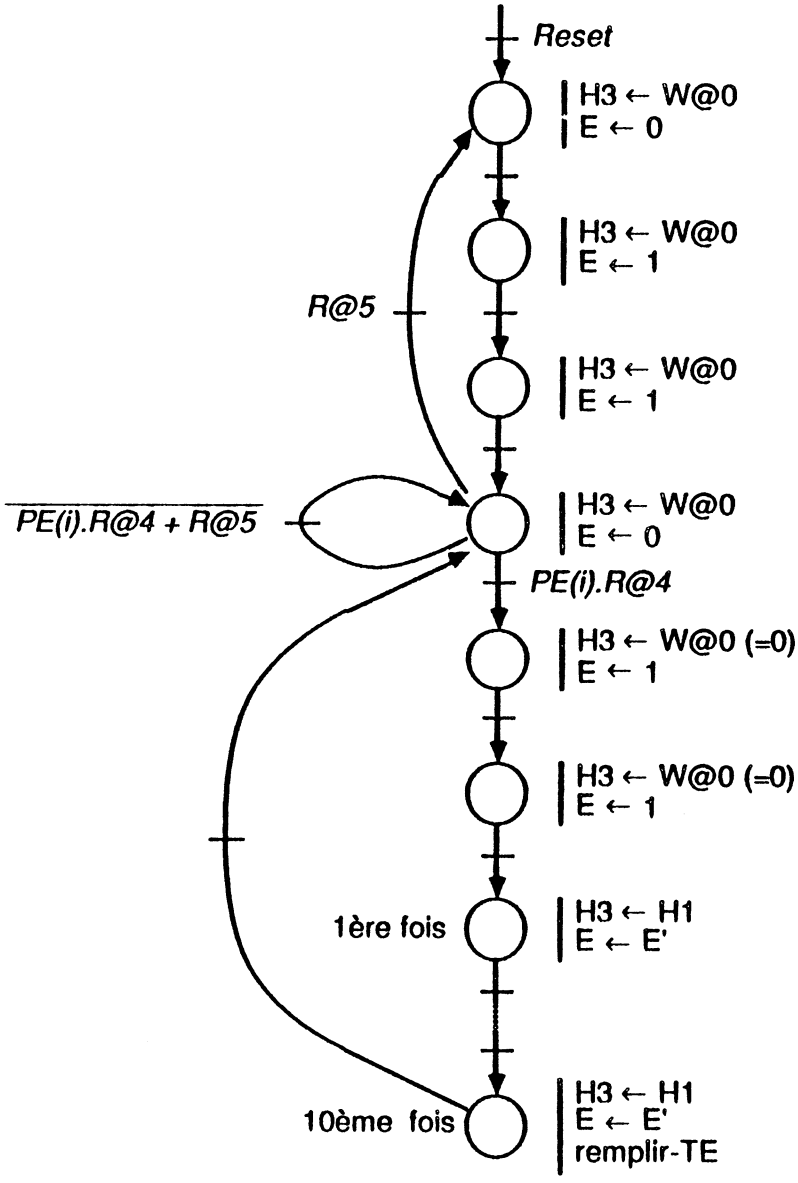
2.1 La partie contrôle-entrée

Chaque partie contrôle-entrée déroule l'algorithme suivant :

```
vider le drapeau interne associé au tampon de sortie;
commande registre à décalage entrée ← W@0;
tant que vrai faire
    tant que l'ordinateur hôte n'a pas rempli
    les registres à décalage d'entrée faire
        si l'ordinateur-hôte a lu les registres à décalage de sortie
        alors
            vider le drapeau associé au tampon de sortie (0.1.1);
        fin si
    fin tant que
    si prise en compte vrai
    alors
        envoyer sur L2 la séquence 0.1.0;
        commande registre à décalage entrée ← H1;
        pour i=1 à 10 faire
            échantillonner la ligne L2' dans a;
            écrire a sur L2;
        fin pour
        envoyer 0 sur L2;
        commande registre à décalage entrée ← W@0;
        remplir le drapeau externe associé au tampon d'entrée;
    fin si
fin tant que
```

Le point critique dans l'implémentation de cet algorithme est celui du changement de la commande d'horloge du registre à

décalage d'entrée (W@0 → H1 qui est en fait un passage de 0 → H1). Ce changement doit être parfaitement en phase avec H1.

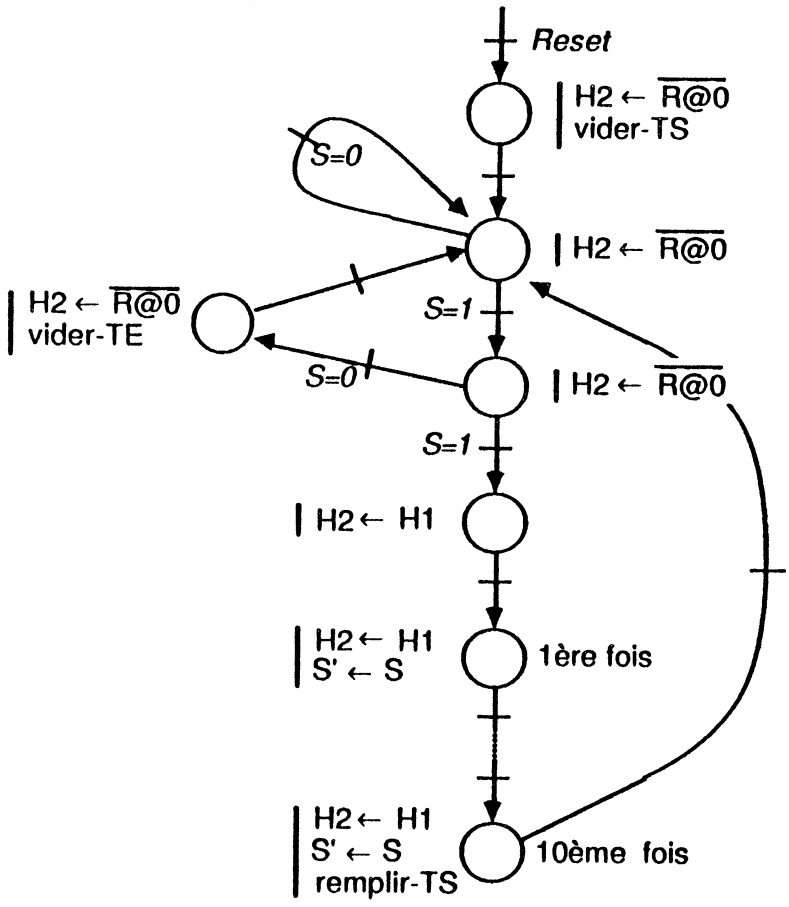


Le séquencement de la partie contrôle-entrée

2.2 La partie contrôle-sortie

L'algorithme de la partie contrôle-sortie est le suivant :

vider le drapeau externe associé au tampon de sortie;
commande registre à décalage de sortie ← $\overline{R@0}$;
tant que vrai **faire**
    échantillonner la ligne L1 dans a;
    tant que a=0 **faire**
        échantillonner la ligne L1 dans a;
    fin tant que
    échantillonner la ligne L1 dans a;
    si a=1
    alors
        vider le drapeau externe associé au tampon d'entrée;
    sinon (il s'agit d'un message de sortie)
        commande registre à décalage de sortie ← H1;
        pour i=1 à 10 **faire**
            échantillonner la ligne L1 dans a;
            écrire a sur L1';
        fin pour
        commande registre à décalage de sortie ← $\overline{R@0}$;
        remplir le drapeau externe associé au tampon d'entrée;
    fin si
fin tant que



Le séquencement de la partie contrôle-sortie

2.3 L'ordinateur-hôte

Le pré-traitement de la simulation, à partir des descriptions structurelle et temporelle du circuit à simuler et de la description de la simulation (compilation des données), conduit à la génération du fichier d'initialisation du réseau et celui de simulation.

La structure de ces fichiers est présentée au paragraphe suivant ainsi que l'initialisation des cellules du réseau (programmation).

L'algorithme de contrôle de la simulation séquence le déroulement de la simulation, à partir des données du fichier de simulation.

Algorithme de contrôle de la simulation

```
initialiser les prises en compte des entrées; ([@3] ← PE)
tant que h<h-max faire
    h := h+1;
    initialiser les registres à décalage des entrées;
    envoyer le signal DEBUT; (lecture de l'adresse @7)
    valider les entrées; (lecture de l'adresse @4)
    répéter
        lire les drapeaux des sorties; (lire @2)
    jusqu'à ce que tous les (bons) tampons de sortie soient pleins
    lire les registres à décalage de sortie;
    vider les drapeaux externes des sorties; (lecture @5)
fin tant que
```

Cet algorithme est directement lié au chargement parallèle (resp. déchargement) des registres à décalage d'entrée (resp. de sortie). Ce choix limite le nombre d'entrées/sorties du réseau au nombre de bits du bus de données. Mais ce choix permet surtout d'effectuer les entrées/sorties du réseau en 10 cycles d'accès mémoire (nombre de bits d'un message).

Avec cet algorithme, la simulation est synchronisée non pas par les signaux DEBUT et FIN, mais par le signal DEBUT et par le fait que tous les tampons de sortie sont pleins (ou tout du moins ceux qui doivent l'être). Le signal FIN ne sert alors à rien.

Il serait alors intéressant de câbler une interruption de programme sur le fait que tous les tampons de sortie qui doivent recevoir un message sont pleins. Si l'ordinateur-hôte est suffisamment rapide, il pourra, pendant que le réseau travaille, initialiser les registres à décalage d'entrée du pas de simulation suivant.

La gestion de l'interruption va consister en une sorte de relais autour de la variable booléenne drapeau-FIN.

L'algorithme principal est alors le suivant :

```
drapeau-FIN := vrai;
tant que h<h-max faire
    h := h+1;
    initialiser les registres à décalage;
    tant que drapeau-FIN faux faire rien
    drapeau-FIN := vrai;
    envoyer le signal DEBUT; (lecture @7)
    valider les entrées; (lecture @4)
fin tant que
```

Le programme d'interruption suivant est déroulé lorsque tous les tampons de sortie du circuit à simuler sont pleins :

```
    h:=h+1;
    lire les registres à décalage des sorties;
    vider les drapeaux externes des sorties; (lecture @5)
    drapeau-FIN := faux;
    fin-interruption
```

Quel que soit l'endroit dans le programme principal où interviendra cette interruption, la chronologie de la simulation sera respectée.

Remarques :

→ cette façon de faire, qui peut être intéressante pour des machines-hôtes assez rapides en regard de la durée d'un pas de simulation du réseau, ne fait gagner en fait que la moitié d'un pas de simulation.

→ Si les performances du système-hôte le permettent ou si l'utilisateur l'autorise, la boucle d'attente du programme principal peut être utilisée pour traiter les résultats de la simulation en cours (affichage de chronogrammes ...). On peut gérer ce sous-programme par une deuxième interruption qui inhiberait momentanément la première. Ces résultats, qui sont normalement traiter en fin de simulation, peuvent ainsi être exploités en temps réel par l'utilisateur.

On peut facilement paralléliser l'algorithme de contrôle de la simulation. En effet les traitements des entrées et des sorties du réseau sont indépendants. Les algorithmes sont les suivants :

Algorithme des entrées

tant que h<h-max **faire**
    h := h+1;
    initialiser les registres à décalage d'entrée;
    attendre le signal FIN;
    envoyer DEBUT;
    valider les entrées;
fin tant que

Algorithme de sortie

tant que h<h-max **faire**
    h := h+1;
    attendre FIN;
    scruter les drapeaux de sorties **jusqu'à** ce qu'ils soient pleins;
    lire les sorties;
    vider les drapeaux externes des sorties;
fin tant que

Remarque : cette version parallèle de l'algorithme n'a de sens que sur un système-hôte multi-processeurs autorisant le vrai parallélisme et non le simple multi-tâches.

2.4 Le moniteur d'entrées/sorties du réseau

Le réseau est normalement beaucoup plus rapide que l'ordinateur-hôte, il est donc très important que celui-ci ne perde pas de temps dans le contrôle de la simulation, notamment dans les opérations d'entrée/sortie. Il est donc intéressant, surtout dans le cas des entrées, que les données manipulées permettent l'optimisation de l'opération "initialiser les registres à décalage d'entrée".

La compilation des données de la simulation génère les fichiers d'initialisation et de simulation.

Le fichier d'initialisation du réseau contient, pour chaque entrée du réseau, la liste (éventuellement vide) des messages à transmettre afin de programmer les cellules du réseau.

Le fichier de simulation contient, pour chaque entrée du réseau, la liste (éventuellement vide) des messages à transmettre afin d'initialiser correctement, à chaque pas de simulation, les équipotentielles d'entrée du circuit à simuler.

Afin de ne pas ralentir le déroulement de la simulation lors de l'initialisation des registres à décalage (entrées), ces fichiers sont convertis en listes de mots binaires. Ces informations seront envoyées telles quelles par séquences de 10 mots à l'adresse des registres à décalage.

Les sorties du réseau (résultat de la simulation) suivent un processus inverse. Les mots binaires, lus par le système-hôte à l'adresse des registres à décalage, sont mémorisés tels quels à la fin de chaque pas de simulation. A la fin de la simulation, le traitement des résultats commence par le pré-traitement de ces valeurs binaires. Il s'agit ici d'organiser, en listes ordonnées chronologiquement, les états des équipotentielles de chaque sortie du circuit. Ces listes serviront de base au traitement proprement dit des résultats (édition de chronogrammes, statistiques ...).

La conversion de ces données (listes structurées ↔ mots binaires) et la lecture/écriture des registres à décalage est réalisée par le moniteur d'entrées/sorties du réseau.

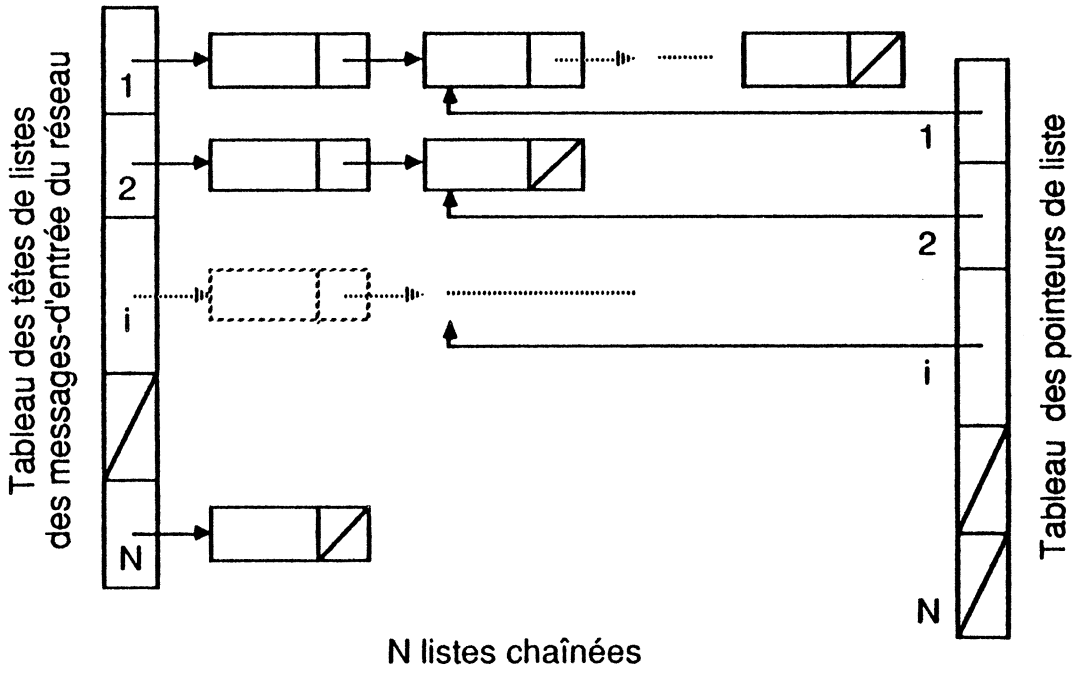
2.4.1 La partie entrée du moniteur

Structure de données des messages d'entrée

Les éléments des listes sont des enregistrements :

```
message-d'entrée = enregistrement de
    DX        : entier;
    DY        : entier;
    Val       : type état-logique;
    suivant   : pointeur de message-d'entrée;
fin enregistrement
```

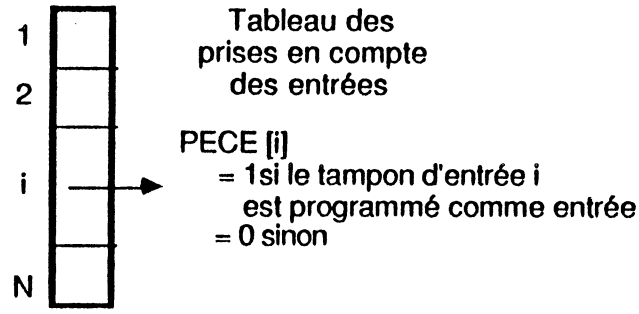
Soit N le nombre de bits du bus de données du système-hôte.



N listes chaînées

La conversion des données, liste -> mots binaires, est réalisée à l'aide d'un tableau de dimension 2, appelé STIMULIS, qui est rempli "en série" et lu "en parallèle". Ce tableau de Nx10 éléments mémorise pour chaque pas d'initialisation ou de simulation, la représentation binaire des messages devant être envoyée au réseau par chacun des tampons d'entrée. Associé à ces informations, le vecteur des prises en compte des entrées (PECE) est un tableau de dimension 1 indiquant la validation (binaire) de chacune des entrées.

	STIMULIS	1 SX	DX2	DX1	DX0	SY	DY2	DY1	DY0	Val2	10 Val1
1	Entrée 1	0	1	1	0	1	0	0	1	0	1
	Entrée 2										
	Entrée i										
N	Entrée N										



Algorithme de conversion listes → mots binaires (entrées)

Soit L la longueur (en nombre de messages) de la liste d'entrée la plus longue, l'algorithme de conversion est alors le suivant :

initialiser les pointeurs de listes;
pour k=1 à L **faire**
  pour i=1 à N **faire**
    initialiser le vecteur STIMULIS [i, 1..10];
    initialiser la variable PECE [i];
    avancer le pointeur de la liste i;
  fin pour
  mémoriser le mot binaire PECE [1..N];
  pour i=1 à 10 **faire**
    mémoriser le mot binaire STIMULIS [1..N, i];
  fin pour
fin pour

Le fichier binaire d'initialisation du réseau comporte exactement (1+10) x L mots de N bits. Lors de l'initialisation, ce fichier est lu par séquences de 11 mots. Le premier mot est la valeur des prises en

compte des entrées, les dix suivants sont les mots d'initialisation (bits à bits) de tous les registres à décalage d'entrée.

Algorithme d'initialisation des cellules du réseau

Le fichier binaire d'initialisation est chargé, dans la mesure du possible, intégralement en mémoire vive du système-hôte par exemple à partir de l'adresse *adr-ent*.

L'algorithme d'initialisation est alors le suivant :

```
envoyer le signal RESET (lecture de l'adresse @4);
pour i=0 à L-1 faire
    (initialiser les prises en compte des entrées du réseau)
    écrire à l'adr @3 le contenu de l'adresse (adr-ent + 11.i);
    (initialiser les registres à décalage des entrées)
    pour j=1 à 10 faire
        écrire à l'adr @0 le contenu de l'adresse  (adr-ent + 11.i + j);
    fin pour
fin pour
attendre le signal FIN;
```

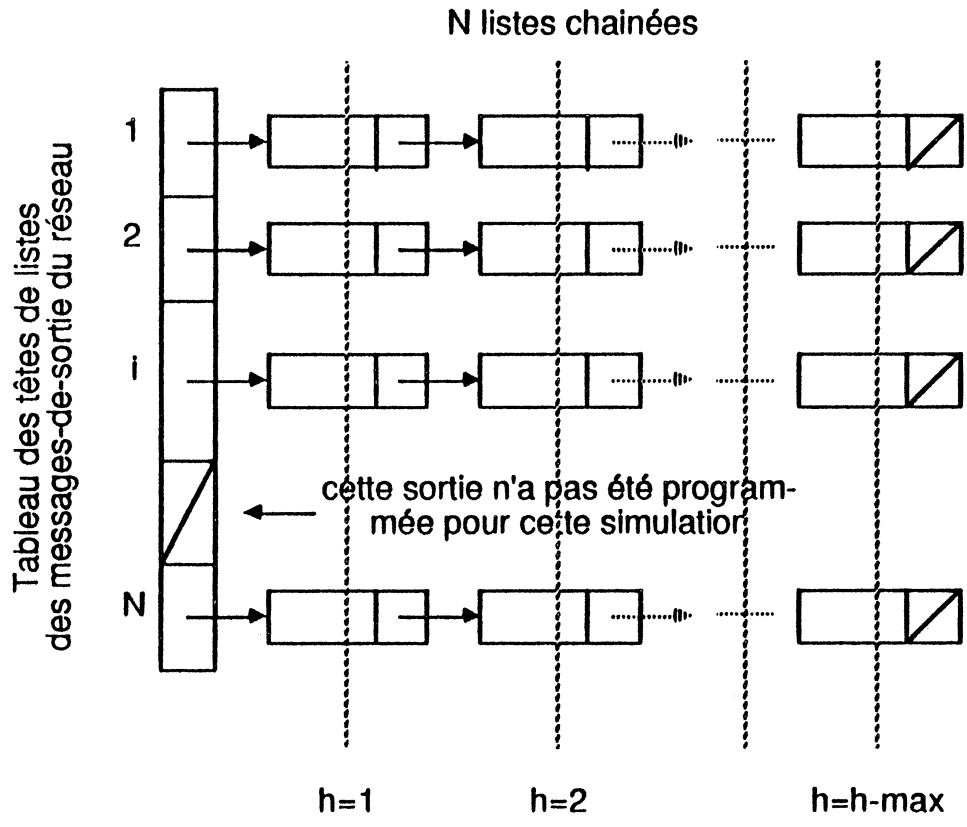
On peut optimiser cet algorithme en respectant lors de la phase de compilation la règle suivante :
toutes les entrées du réseau (programmées comme telles pour l'initialisation) doivent acheminer un même nombre constant de messages.

Dans ce cas, on peut initialiser les prises en compte de manière unique et définitive en une seule opération avant l'envoi des séquences d'initialisation des cellules du réseau.

Les entrées, lors de la simulation proprement dite, peuvent être initialisées de cette façon, car elles sont alors parfaitement définies et ne peuvent changer en cours de simulation.

2.4.2 La partie sortie du moniteur

Pour la sortie du réseau, la structure de données est la même sauf pour le champ-routage des messages que l'on n'a pas à mémoriser et à traiter ici.



Pendant la simulation, seuls les deux bits du champ-donnée des messages de sorties sont mémorisés. Le résultat de chaque pas de simulation occupe donc 2 mots-mémoire.

Algorithme de conversion mots binaires → listes (sortie)

A la fin de la simulation, l'algorithme du moniteur de sortie est alors le suivant :

```
message-de-sortie : enregistrement de
    | donnée  : type état-logique;
    | suivant  : pointeur de message-de-sortie;
fin enregistrement
```

```
initialiser les tête de listes des messages de sorties à NIL;
pour h=1 à h-max-simulation faire
    pour i=0 à 1 faire
        initialiser le mot binaire STIMULIS [1..N, i] avec le contenu
        de l'adresse (adr-sort + i);
    fin pour
    pour chaque sortie programmée du réseau faire
        pour i=1 à N faire
            créer un nouveau message-de-sortie MS;
            MS.donnée := STIMULIS [i, 9..10];
            insérer l'enregistrement MS en queue de la liste des
            messages de sortie i;
        fin pour
    fin pour
fin pour
```

Les listes ainsi générées ont toutes la même longueur égale au nombre de pas de la simulation, c'est-à-dire h-max-simulation. Les messages des listes sont ordonnés chronologiquement.

Remarque : la correspondance entre les bits des messages et les sorties est établie lors de la phase de compilation des données.

3 REALISATION DE LA CARTE

Pour des questions d'ordre prâtique, nous avons choisi d'utiliser comme système-hôte un ordinateur IBM-PC/AT. L'architecture ouverte et le succès commercial de cette machine (environnement logiciel et matériel) en ont fait un système bien adapté au développement et à l'intégration de cartes d'extension spécialisées.

3.1 Le fond de panier de l'IBM PC/AT

L'IBM-PC/AT [IBM84] est architecturé autour du microprocesseur 16 bits 80286 d'INTEL. La carte-mère du système possède 8 connecteurs d'extension à travers lesquels on accède aux 92 signaux du fond de panier. Ces signaux sont pour la plupart ceux du microprocesseur, les plus importants sont :

- 16 bits de données,
- 24 bits d'adresse,
- 16 niveaux d'interruption,
- 7 canaux de DMA,
- les signaux de lecture/écriture mémoire (I/OW et I/OR),
- 2 horloges, celle du microprocesseur (6 Mhz) et une autre plus rapide (14,31818 Mhz),
- des alimentations,
- divers signaux de contrôle notamment :
  - un signal reset (mise sous tension de la machine),
  - le signal AEN -Adress ENable- (autorisation).

Le microprocesseur fonctionne à une fréquence de 6 Mhz, ce qui donne un temps de cycle de 167 ns. Un cycle d'accès au bus nécessite 3 cycles d'horloge soit 500 ns. La mémoire vive (RAM) peut être étendue jusqu'à 16 Méga-octets.

Remarque : certains ordinateurs compatibles avec l'IBM-PC/AT, comme le COMPACQ, utilisent une horloge plus rapide (14 Mhz).

3.2 La technologie de l'interface

Les composants de l'IBM-PC sont issus de la famille logique TTL-LS ou lui sont compatibles; le circuit cellulaire est, lui, réalisé en

CMOS. Le problème entre ces deux technologies est celui de la compatibilité entre les niveaux des signaux électriques (tensions de seuil). Les niveaux de sorties CMOS peuvent commander des entrées TTL-LS, mais l'inverse est impossible (ou pour le moins hasardeux !).

Une solution, pour permettre le dialogue entre le réseau cellulaire et l'IBM-PC, est de développer la carte d'interface en logique HCMOS [ELE84].

3.3 Les composants

La famille HCMOS offre une gamme complète de composants logiques permettant la réalisation des interfaces de la carte, mais le nombre de boîtiers sera élevé. En effet, la carte d'extension supportant le réseau cellulaire est composée :

→ des 16 interfaces-bus du réseau (chacune composée de deux registres à décalage de 10 bits et des parties-contrôle d'entrée et de sortie),
→ des éléments de mémorisation des drapeaux associés aux tampons d'entrée et de sortie, et des prises en compte des entrées du réseau,
→ du décodeur des lignes d'adresse et de contrôle.

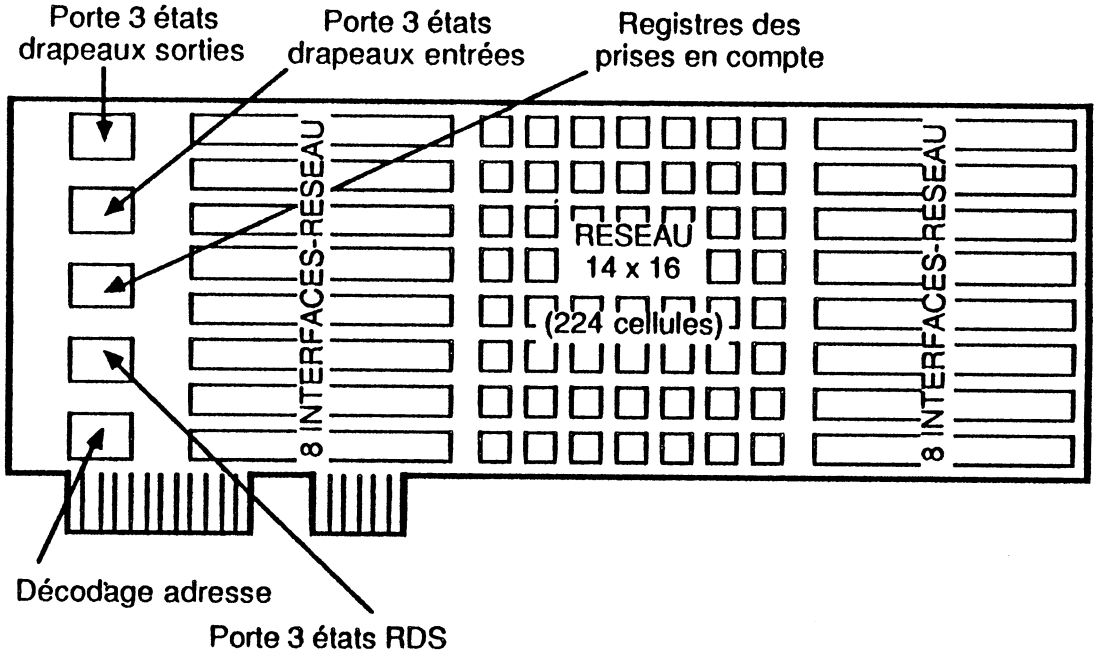
Il est donc indispensable de réduire le nombre des composants en les regroupant par fonctionnalité et en utilisant des boîtiers programmables de type PAL (Programmable Array Logic) notamment pour les parties-contrôle et le décodage des adresses.

3.4 Les caractéristiques de la machine de simulation logique

La machine de simulation intègre un réseau dont les dimensions sont déterminées par :

- le champ-routage des messages
  (3 bits par coordonnée pour la valeur absolue du déplacement),
- la conception du circuit cellulaire 2x2
  (4 entrées interfacées sur 8 et 4 sorties interfacées sur 8),
- le bus de données du système-hôte (16 bits).

Ces paramètres permettent la réalisation d'une machine de simulation prototype, intégrant un réseau 14x16, avec 16 entrées et 16 sorties. Cette machine pourra donc simuler des circuits logiques d'un maximum de 224 portes dont l'entrance et la sortance sont au maximum de 16.



Plan de masse de la carte d'extension intégrant
la machine de simulation cellulaire

CONCLUSION

Cette thèse a présenté tout d'abord l'étude d'une architecture cellulaire basée sur un réseau de cellules asynchrones communiquant par messages. Nous avons exposé ensuite une première application de cette architecture : la réalisation d'un circuit VLSI spécialisé pour la simulation logique.

L'architecture parallèle proposée ici fait partie du domaine des architectures massivement parallèles, domaine fondamentalement opposé à celui des architectures classiques du type Von Neumann.

Les architectures du type Von Neumann sont limitées par leur nature séquentielle. Il est certain que le recours à des architectures parallèles est nécessaire pour atteindre les performances dont les applications scientifiques ont maintenant besoin. Parmi ces architectures, les réseaux cellulaires sont composés d'un grand nombre de processeurs simples. A la différence des autres architectures parallèles régulières, notamment systoliques, le domaine d'application des réseaux cellulaires asynchrones n'est pas limité par les communications locales entre les processeurs.

La réalisation d'un accélérateur de simulation basé sur cette architecture nous a permis d'approfondir les contraintes liées au fonctionnement asynchrone et hautement parallèle d'un nombre important de cellules. Les problèmes soulevés sont de différents ordres :

Partitionnement de l'algorithme implémenté

C'est la "parallélisation" de l'algorithme à répartir sur les processeurs du réseau.

Placement des cellules

C'est un élément très important car il permet d'optimiser de manière globale les distances entre micro-processeurs communiquants, ce qui accroît les performances du réseau.

Communications inter-cellulaires et avec le système-hôte

C'est l'aspect le plus original de cette architecture : une cellule peut communiquer avec n'importe quelle autre cellule du réseau grâce à un échange de messages. Notre travail a permis

de mieux appréhender la complexité et les performances de la partie acheminement des cellules. Les seules limitations à ce type de communication sont d'ordre physique, lorsque la taille des messages est trop importante ou temporelle pour des distances à parcourir trop longues.

Les performances du réseau sont liées, pour une grande part, à la durée d'acheminement des messages (temps de traversée des cellules et surtout nombre de cellules traversées). On peut réduire les distances parcourues par les messages en augmentant le nombre de liens directs entre les cellules : une cellule donnée pourrait alors communiquer directement avec 6 ou 8 cellules voisines au lieu de 4 seulement.

Synchronisation des processeurs

Il est très vite apparu au cours de notre étude qu'il était souvent nécessaire à un moment ou à un autre de synchroniser les processeurs du réseau. Nous avons étudié 2 formes de synchronisation : par signaux globaux et par accusés de réception.

Nous avons choisi, pour la première réalisation du simulateur logique cellulaire, le premier type de synchronisation, celui-ci étant le plus facile à implémenter. Mais l'intervention du système-hôte ralentit le déroulement de l'algorithme réparti sur le réseau. De plus son intégration au niveau tranche n'est pas envisageable.

Quant à la synchronisation par accusés de réception, elle a pour principal inconvénient de doubler le nombre de messages acheminés dans le réseau.

L'algorithme de simulation logique que nous avons implanté sur cette architecture cellulaire resynchronise à chaque pas tous les processeurs du réseau. Cette application ne tire donc pas le meilleur parti du fonctionnement asynchrone du réseau et ne permet pas une activité maximale de celui-ci.

Cependant le travail réalisé a démontré indubitablement la faisabilité de cette architecture dont les performances de l'intégration sur tranche permettront bientôt d'exploiter l'énorme puissance de calcul potentielle. Notre étude corrobore d'autres études qui ont montré que ce type d'architecture est très bien adapté à la réalisation d'autres opérateurs spécifiques [FAU88b] [LAT88a], voir même de machines programmables [PAY88].

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[AGR 86] **P. Agrawal**
"Concurrency and communication in hardware simulators",
IEEE Transactions on Computer Aided Design, 5,4 Oct 1986.

[AGR 87] **P. Agrawal et al**
"Architecture and Design of the MARS Hardware accelerator",
Procedings of the 24th ACM/IEEE Design Automation Conference, 1987.

[AMB 87] **A.P. Ambler, N. Colemen, R.L. Manning, N. Muhammed**
"The application of Hardware accelerators in VLSI Design",
Procedings of the 1st IEEE International Conference on Computer Technology, Systems and Applications, (COMPEURO 87), mai 1987.

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

[ANS 88] **Y. Ansade**
" "
Thèse INPG de microélectronique, 1988.

[BER 84] **J.M. Berger et al**
"LOF : A building tool for flexibel blocks librairies",
Proceedings of the IEEE International Conference on Computer Design (ICCD 84), 1984, pp 851-856.

[BER 85] **J.P. Bernard**
"Etude d'une machine cellulaire pour la simulation logique de circuits intégrés",
Thèse INPG de microélectronique, Grenoble juillet 1985.

[BIM 85] **Bimos 2.1**
"The logic simulation system - Reference manual",
Document n° M-022-4, Silvar-Lisco, octobre 1985.

[BLA 84] **T. Blank**
"A survey of hardware accelerators used in computer aided design",
IEEE Design and Test, vol 1, pp. 21-39, August 1984.

[BRE 72] **M. Breuer**
"Design automation of digital systems",
Prentice Hall, 1972.

[BRY 86] **R.E. Bryant**
"Compiled simulation of MOS circuits",
Proceedings of the Canadian Conference on Very Large Scale Integration,
Montréal 27-28 oct 1986 pp.217-19,
Publi : Mc Gill University.

[BRY 87] **R.E. Bryant**
"A survey of switch-level algorithms",
IEEE Design and Test, vol. 4, n°4, 1987.

[CAN 85] **M. Cand, E. Demoulin, J.L. Lardy, P. Senn**
"Conception des circuits intégrés MOS",
Editions Eyrolles, 1985.

[CHI 86] **M. Chiang, R. Palkovic**
"LCC simulators speed development of synchronous hardware",
Computer Design, Mars 86, pp. 87-92.

[COR 86] **R. Cornu-Emieux et al**
"Some highly parallel algorithms processed by a regular network of asynchronous cells ",
Proceedings of the International Workshop on Parallel Algorithms Architectures, Luminy, North Holland 1986.

[COR 87a] **R. Cornu-Emieux, Ph. Objois**
"OCCAM pour la simulation fonctionnelle d'une architecture hautement parallèle",
Séminaire "Programmation Parallèle, Transputer et OCCAM",
Talloires, mai 1987.

[COR 87b] **R. Cornu-Emieux, D. Lattard, G. Mazaré, Ph.Objois**
"Réseau de cellules asynchrones dédié à la simulation logique : conception et réalisation",
Journées Architectures de C^3, Sophia Antipolis, juillet 1987,
Bigre et Globule n°56 novembre 1987.

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

[COR 88] **R. Cornu-Emieux**
"Réseau de cellules intégré : Etude d'architectures pour des applications de CAO",
Thèse INPG de microélectronique, Grenoble 1988.

[DAI 87] **Daisy**
"Manuel de présentation, Megalogician",
DAISY System, 1984.

[DAL 87] **P. Agrawal, W.J. Dally, R. Tutundjian**
"Logic simulation algorithms for pipelined hardware architectures",
Proceedings of the International Workshop on Hardware Accelerators,
Oxford, octobre 1987.

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

[DUN 84] **L.N. Dunn**
"IBM's Engineering Design System Support for VLSI Design and verification",
IEEE Design & Test of Computers, vol. 1, n° 1, fevrier 1984, pp. 30-40.

[DYS 87] **C.M. Dyson, A.H. Gray**
"Mixed-mode simulation, on transputers",
Proceedings of the International Workshop on Hardware Accelerators,
Oxford, octobre 1987.

[EFC 86] **EFCIS**
"Epilog : Manuel d'utilisation",
EFCIS, 1986.

[ELE 84] **Elektor**
"Exogamie logique",
Elektor, février 1984.

[ELK 87] **M. Elkatrani**
"Etude d'un aiguilleur combinatoire",
Rapport de DEA microélectronique - INPG, Grenoble, juin 1987.

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

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

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

[FAZ 87] **W. Fazakerly**
"Hardware simulator models for logic devices",
VLSI System Design, vol 18, n°12 Novembre 1987.

[FLA 81] **P.L. Flake et al**
"HILO mark 2 Hardware description language"
Proceeding of 5th International symposium on Computer Hardware
Description and their application (CHDL 81), 1981, pp 95-108.

[FLA 84] **E. Flamand**
"A complete and automatic system for sequencer design",
Proceeding of IEEE International Conference on Computer Design (ICCD 84), 1984.

[FLY 72] **M.J. Flynn**
"Some computer organizations and their effectiveness",
IEEE Transactions and Computers, C 21, 9, Sept. 1972.

[FRA 87] **M.A. Franklin, K. Wong**
"load and communications balancing on multiprocessor logic simulation engines",
Proceedings of the International Workshop on Hardware Accelerators, Oxford, octobre 1987.

[GIN 83] **L. Gindraux, G. Catlin**
"CAE Station's simulators tackle 1 million gates",
Electronic Design, 10 nov. 1983, pp. 127.

[GRA 87] **C.M. Dyson, A.H. Gray**
"A pilpelined event-driven mixed-mode simulator",
Publication IEEE CH2469, mai 1987.

[HEN 87] **B. Hennion, P. Senn**
"Eldo : A general purpose third generation circuit simulator based on the OSR method", ECCTD 1987.

[HIR 87] **F. Hirose et al**
"Simulation Processor SP",
Publication CH2469, IEEE mai 1987.

[HOW 83] **J.K. Howard, L. Malm, L.M. Warren**
"Introduction to the IBM Los Gatos Logic Simulation Machine",
Proceedings of the IEEE International Conference on Computer Design : VLSI in Computer, octobre 1983, pp. 584-587.

[HWA 87] **P. Hwang, J. Zasio**
"A low-cost high performance levelized-compiled-code simulation accelerator",
Proceeding of the International Workshop on Hardware Accelerators, 30 Sept-2O ct 1987.

[IBM 84] **IBM**
"Technical Reference Personal Computer AT",
IBM, 1984.

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

[KAT 83] **S. Kato, T. Sasaki**
"FDL : A structural behavior description langage",
Proceedings of the 6th Tnternationnal Symposium CHDL, mai 1983, pp. 137-152.

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

[LAT 88a] **D. Lattard, G. Mazaré**
"Parallel image reconstruction by using a dedicated asynchronous cellular array",
Procedings of the Parallel Processing for Computer Vision and Display Conference, Leeds, janvier 1988.

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

[LEC 82] **J. Lecourvoisier et al**
"A design methodology based upon symbolic layout and CAD tools",
19th Design Automation Conference, Las Vegas, juin 1982.

[LLT 87] **Aptor**
"LL3T : User's guide",
Aptor, 1987.

[MAH 87] **M. McMahon**
"Accelerators for faster logic simulation : the zycad approach",
Procedings of the 1st IEEE International Conference on Computer Technology, Systems and Applications, (COMPEURO 87), mai 1987.

[MAZ 87] **G. Mazaré, R. Cornu-Emieux, Ph. Objois**
"A VLSI asynchronous cellular array to accelerate logical simulation",
Procedings of the 30th Midwest International Symposium on Circuits and Systems, Syracuse (NY), août 1987.

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

[MIS 86] **J. Misra**
"Distributed discrete-event simulation",
ACM Computing surveys, vol 18, n°1, mars 1986, pp. 39-65.

[MOT 85] **G. Mott, R. Hall**
"The utility of hardware accelerators in the Design Environment",
VLSI Design, octobre 1985.

[NOR 87] **P. Agrawal, L.W. Noronha**
"Modeling Circuits in the MARS Hardware Accelerator",
Publication IEEE CH2469, mai 1987.

[OBJ 87a] **Ph. Objois et al**
"Simulation fonctionnelle d'une architecture parallèle en OCCAM",
7th OCCAM User Group Meeting & International Workshop on Parallel Programing of Transputer Based Machines, Grenoble, septembre 1987.

[OBJ 87b] **Ph. Objois, Y. Ansade, R. Cornu-Emieux, G. Mazaré**
"Highly parallel logic simulation accelerators based upon distributed discrete-event simulation",
Proceedings of the International Workshop on Hardware Accelerators, Oxford, octobre 1987.

[PAY 88] **E. Payant**
"Etude de la réalisation d'architecture sequencée par les données sous forme d'un réseau de cellulaire",
Rapport de DEA informatique - INPG, Grenoble, juin 1988.

[PFI 82] **G.F. Pfister**
"The Yorktown Simulation Engine",
Procedings of the 19th ACM/IEEE Design Automation Conference, juin 1982, pp. 51-54.

[SAS 83] **T. Sasaki et al**
"HAL II : a block level hardware logic simulator",
Procedings of the 20th ACM/IEEE Design Automation Conference, juin 1983, pp150-156.

[SMI 86] **R.J. Smith**
"Fundamentals of parallel logic simulation",
Procedings of the 23rd Design Automation Conference, juin 1986.

[TAK 86] **S. Takasaki, T. Sasaki and al**
"HAL II : a mixed level hardware logic simulation system"
Procedings of the 23rd Design Automation Conference, 1986.

[THO 80] **E. Thompson et al**
"Tegas Design Langage (TDL) - A system for modular description, and top-down / bottom-up verification of LSI and VLSI",
IEEE International Symposium on Circuits and System, 1980, pp 300-303.

[TRA 86] **P. Traynar**
"Utilisation des accélérateurs matériels pour la simulation logique en IAO/CAO",
Electronique Industrielle, N°110, juin 1986.

[TUR 85] **L. Turner Smith, D.A. Gross**
"Preparing large networks for a simulation accelerator",
Proceedings of the IEEE International Symposium Circuits And System, (ISCAS 85), 1985.

[VLS 88a] **VLSI Systems Design Staff**
"1988 Survey of Logic Simulators",
VLSI Systems Design, février 1988.

[VLS 88b] **VLSI Systems Design Staff**
"Hardware Accelerators",
VLSI Systems Design, août 1988.

[WAN 87] **L.T. Wang et al**
"SSIM : A software Levelized Compiled-Code simulation",
Proceedings of the 24th IEEE Design Automation Conference, 1987.

[WHA 86] **D.J. Wharton**
"Behavioral modeling in logic simulation",
VLSI Design, août 1986.

[WON 86] **K.F. Wong, M.A. Franklin, R.D. Chamberlain, B.L. Shing**
"Statistics on logic simulation",
Proceedings of the 23rd IEEE Design Automation Conference, 1986.

[ZYC 84] **Zycad**
"LE1000 reference manual",
Zycad, 1984.

ANNEXE 1

LA SIMULATION LOGIQUE

TABLE DES MATIERES

L'avènement des circuits à haute densité d'intégration a entraîné une évolution dans la conception des circuits. Evolution sur la forme, avec des technologies permettant la réalisation de circuits entre cent mille et un million de transistors, et évolution sur le fond, avec la participation directe du concepteur à tous les stades de la conception, du cahier des charges jusqu'au dessin des masques.

De plus, le concepteur n'a plus le droit à l'erreur, les coûts de fabrication sont tels à ce degré d'intégration que la moindre faute entraîne non seulement un sur-coût, mais également des retards dans la conception. D'ailleurs, la Conception Assistée par Ordinateur (CAO) évolue vers une automatisation de plus en plus poussée.

Depuis longtemps, la conception de circuits utilise la puissance de calcul des ordinateurs, et les outils logiciels de conception ont dû eux aussi évoluer pour rester efficaces. Au cours de la conception, trois types de logiciels exigent aujourd'hui des moyens de calculs très importants : il s'agit des simulateurs, des vérificateurs de dessin et des compilateurs de silicium (placeurs, routeurs ...).

La complexité grandissante des circuits intégrés a entraîné le développement de machines spécialisées pour ces outils afin que le temps de calcul reste du domaine de l'acceptable. Parmi ces gros consommateurs de temps CPU (Central Process Unit), les simulateurs ont fait l'objet de recherches particulièrement poussées, car la simulation est le pivot de la CAO des circuits à haute densité d'intégration (VLSI - Very Large Scale Integration).

La méthodologie descendante couramment utilisée en conception permet la décomposition de l'étude des circuits en plusieurs étapes :

- étude comportementale
- étude logique
- étude électrique

Ces étapes correspondent à des degrés d'abstraction de plus en plus proches du comportement physique des circuits intégrés.

Chaque étape fait appel à un ou plusieurs niveaux de description, modélisant de façon de plus en plus fine le circuit réel.

Il ne viendrait jamais à l'esprit d'un concepteur d'envoyer à la fabrication un circuit qu'il n'a pas entièrement simulé. L'automatisation complète de la chaîne de CAO lui permettra certainement un jour de se passer de cette étape, mais en attendant les compilateurs de silicium, le concepteur doit intervenir à toutes les étapes de la conception et les valider.

L'étape finale, c'est bien sûr le niveau électrique, mais la simulation d'un circuit complet à ce niveau est trop complexe pour être envisagée tout de suite, car plus on se rapproche du niveau électrique, plus la simulation est longue.

On distingue habituellement trois niveaux d'abstraction dans l'étude logique des circuits :

- ❒ niveau fonctionnel,
- ❒ niveau porte,
- ❒ niveau interrupteur.

Certains simulateurs savent traiter plusieurs niveaux en même temps, on parle alors de simulation multi-niveau (multi-level simulator).

Au niveau logique, la précision de la simulation repose sur la modélisation :

- → des équipotentielles du circuit,
- → du temps.

Une simulation au niveau logique est utile, bien sûr, à la validation logique du circuit simulé, mais également à sa validation temporelle représentée par le séquencement des éléments logiques dans le temps, c'est-à-dire l'ordre des événements.

1 FORMALISATION DU PROBLEME

Le rôle d'un simulateur logique est de déterminer l'évolution temporelle des signaux logiques à l'intérieur d'un circuit, en fonction d'une séquence d'entrée donnée. Les éléments manipulés par le simulateur sont de deux types : les éléments logiques qui composent le circuit et les signaux véhiculés par les interconnections de ces éléments. Le temps est également une donnée très importante. En effet, grâce à l'horloge de la simulation, on peut ordonner chronologiquement tous les événements d'une simulation.

1.1 Modélisation des circuits à simuler

Le circuit à simuler est toujours modélisé comme une interconnexion d'éléments logiques. La description des composants élémentaires est fonction du niveau d'interprétation du simulateur. Cette description est très souvent hiérarchisée.

Niveau fonctionnel

Au niveau fonctionnel, les éléments logiques sont décrits, le plus souvent, à l'aide d'un langage spécifique de haut niveau, par des algorithmes caractérisant leur fonctionnement. Ces éléments sont prédéfinis (bascule, latch ...), ou créés par le concepteur.

Niveau porte (gate-level)

Au niveau logique, le concepteur décrit le circuit à simuler comme un réseau de portes logiques élémentaires interconnectées. La logique booléenne permet la représentation de n'importe quelle fonction logique, à l'aide de trois opérateurs de base : l'inverseur, le ET et le OU. Néanmoins, tous les simulateurs proposent une bibliothèque d'éléments logiques déjà caractérisés.

Parmi les portes logiques les plus fréquemment utilisées, on trouve l'inverseur, le ET le OU ainsi que le NON-ET, le NON-OU, le OUEX (ou exclusif), le NON-OUEX . Certains simulateurs proposent également des opérateurs plus complexes (bascule, registre ...), on parle alors de simulation logique au niveau registre.

Si l'on se limite aux portes de base énumérées ci-dessus, une porte est entièrement définie au niveau fonctionnel par :
- son nombre d'entrées (entrance),

- la fonction logique qu'elle réalise.
Il reste alors à connaître l'interconnexion de ces portes pour obtenir la description structurelle du circuit.

Niveau interrupteur (switch-level)

Au niveau interrupteur, l'élément de base est le transistor MOS considéré comme un interrupteur qui peut être ouvert ou fermé. Le circuit est alors décrit comme un réseau d'interrupteurs.

1.2 Modélisation des signaux

D'un point de vue électrique, les équipotentielles sont caractérisées par leurs tensions et les intensités des courants qu'elles véhiculent. En simulation logique, ces valeurs sont discrétisées en un nombre fini d'états logiques. Ces états sont représentés par un couple (niveau logique, force du signal). Le nombre d'états disponibles pour la simulation est fonction de la précision du simulateur.

1.2.1 Modélisation des niveaux logiques

La logique booléenne ne reconnaît que deux valeurs logiques :
- → vrai - codé par 1,
- → faux - codé par 0.

Cette logique est trop restrictive, et ne permet pas de modéliser le comportement logique des circuits avec le niveau de détail souhaité. Pour affiner la modélisation, d'autres valeurs sont nécessaires, à commencer par la valeur indéterminée X.

La valeur indéterminée X

On rencontre souvent cette valeur au début d'une simulation. Dans ce cas, tous les niveaux logiques des équipotentielles qui n'ont pas été initialisées sont, à priori, inconnus. On peut alors étudier le temps de stabilisation d'un circuit lors de son initialisation.

Le niveau indéterminé X peut aussi résulter du conflit entre deux niveaux différents de puissances identiques.

Les valeurs "transitoires"

Afin d'étudier les périodes transitoires entre les niveaux stables, on fait appel à des valeurs supplémentaires. Certains simulateurs utilisent la transition indéterminée souvent notée Ø, alors que d'autres distinguent :

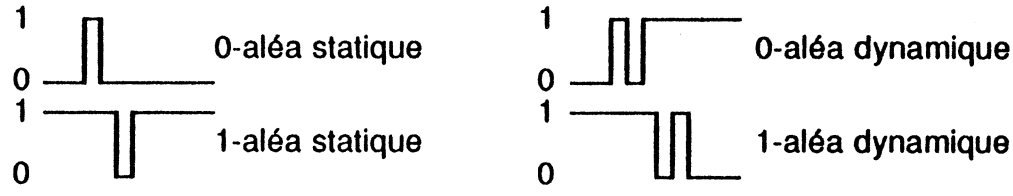
- la transition montante (passage de 0→1) notée $Ø_{01}$,
- la transition descendante (passage de 1→0) notée $Ø_{10}$.

La transition montante est également représentée par le symbole U (pour UP), et la transition descendante D (DOWN).

Les aléas de fonctionnement

Toujours dans les régimes transitoires, un aléa de fonctionnement représente la brève variation de niveau d'un signal. On différencie les aléas statiques - le signal est stable -, des aléas dynamiques - le signal est dans une période transitoire (montante ou descendante) -.

Les aléas sont modélisés par 4 niveaux :



Modélisation des aléas

Peu de simulateurs logiques intègrent ces phénomènes et leurs représentations symboliques ne sont pas standardisées.

1.2.2 Modélisation des forces

Malgré la diversité des niveaux logiques, certains conflits ne peuvent être réglés que grossièrement : par exemple, un ET câblé entre un signal de niveau logique 0 et un autre à 1. En l'absence d'autre information, le simulateur ne pourra calculer le niveau logique résultant et générera un niveau indéterminé X.

La puissance d'un signal est représenté par la force de l'état logique qui lui est associé. Cette force est due à l'impédance de sortie de l'élément logique générant le signal. Les principales forces utilisées sont dans l'ordre décroissant :

→ E*(xternal)* représente un signal externe très puissant;
→ D*(riven)* représente un signal interne régénéré. Par exemple en sortie d'un transistor signal;
→ R*(esistive)* représente un signal dégradé. Par exemple en sortie d'un élément résistif;
→ Z*(large high impedance)* représente la force d'un signal mémorisé sur une grosse capacité (bus ...);
→ z*(high impedance)* représente la force d'un signal mémorisé sur une petite capacité (point capacitif ...);
→ U *(unknow)* représente une force inconnue.

Comme pour les niveaux logiques, ces modèles peuvent être complétés par des forces de transition.

En cas de conflit, le signal ayant la force la plus élevée, l'emportera sur les autres. Lorsque les signaux en présence ont des forces égales et des niveaux différents, le niveau du signal résultant ne peut être prédit et sera indéterminé. En plus de ces forces, certains simulateurs utilisent une ou plusieurs forces inconnues.

Certains anciens simulateurs utilisaient la force Z comme un niveau logique à part entière. Ces simulateurs, en général, ne géraient pas les forces, et les états logiques n'étaient donc représentés que par les niveaux logiques.

Le nombre d'états proposés par un simulateur est fonction du nombre de niveaux logiques et de forces. Par exemple, le système Mach 1000 de Silicon Solution travaille avec quatre niveaux et huit forces soit trente deux états logiques.

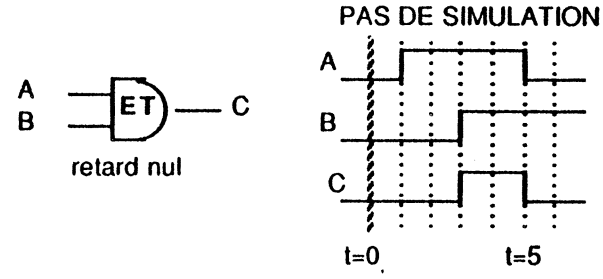
1.3 Modélisation du temps

La traversée d'une porte par un signal n'est jamais réellement instantanée. L'étude temporelle (dynamique) d'un élément logique

conduit à déterminer le retard induit par cet élément, c'est-à-dire l'établissement et la stabilisation, dans le temps, de sa sortie. Comme pour la modélisation logique des signaux, les simulateurs, selon leur précision, travaillent sur des modèles de retards plus ou moins fins. Les retards s'expriment le plus souvent en fonction de l'horloge de simulation. En général, les retards s'appliquent aux portes, mais certains simulateurs les appliquent sur leurs signaux de sortie [EFC86].

1.3.1 Retard nul

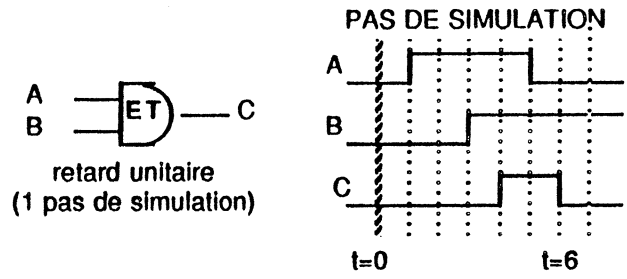
On suppose dans ce modèle simpliste que le temps de traversée d'une porte est nul. Une telle modélisation des retards n'est pas réaliste. Elle permet de vérifier le bon fonctionnement logique des circuits, mais ne donne aucune information d'ordre temporel, puisque les sorties sont valides dès que les entrées le sont.



Exemple de simulation à retard nul

1.3.2 Retard unitaire

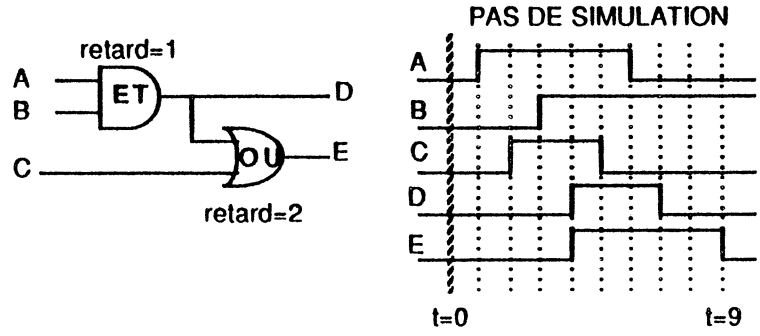
Toutes les portes du circuit à simuler ont le même temps de traversée égal à un pas de l'horloge de simulation. On peut ainsi appréhender les problèmes temporels des circuits (temps de stabilisation).



Exemple de simulation à retard unitaire

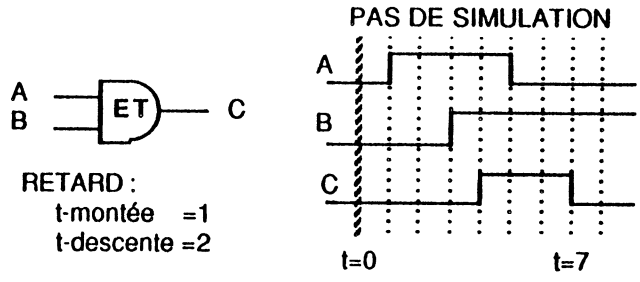
1.3.3 Retard variable

Ici, toutes les portes n'ont plus le même retard constant. Chacune introduit, dans l'établissement de son signal de sortie, un retard qui lui est propre et qui est un multiple entier du pas de simulation.



Exemple de simulation à retard variable

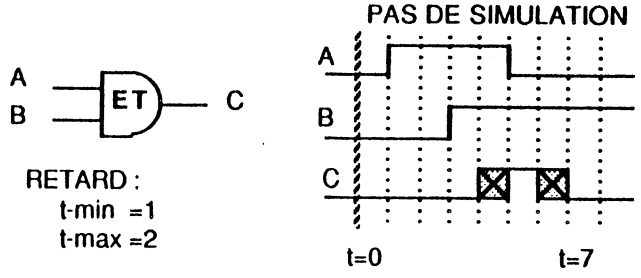
Ce modèle peut être affiné, en différenciant le retard à la montée (passage de la sortie de 0→1), du retard à la descente (passage de la sortie de 1→0). Le retard est alors défini par un couple de valeurs (t-m, t-d).



Exemple de simulation avec retards de montée et de descente différents

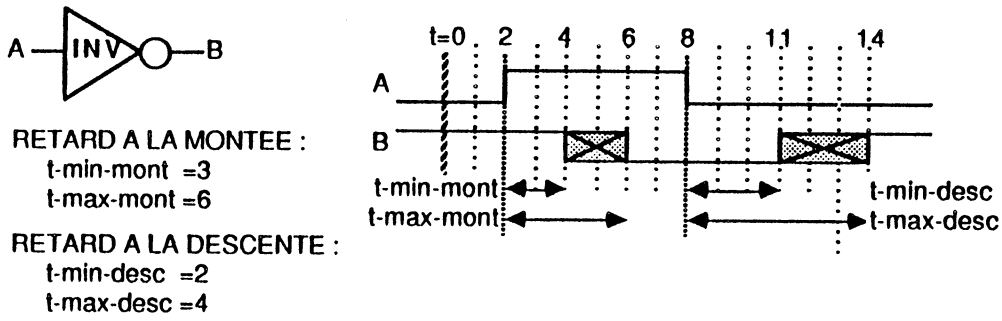
1.3.4 Retard ambigu (min/max)

Lorsque le temps de traversée n'est pas exactement connu, on définit le retard comme un intervalle borné par les valeurs de délais minimales (t-min) et maximales (t-max) de propagation. Avant t-min, la sortie n'a pas bougé, après t-max, la sortie est valide. Entre ces deux extrêmes, la sortie est dans une phase transitoire montante ou descendante.



Exemple de simulation avec retards minimaux et maximaux

Comme pour le retard variable, on peut affiner le modèle du retard ambigu, en différenciant le retard à la montée, du retard à la descente. Le retard est alors défini par deux couples de valeurs, l'un caractérisant le retard à la montée (t-min-montée, t-max-montée), l'autre le retard à la descente (t-min-descente, t-max-descente).



Exemple de simulation avec prise en compte de retards minimaux/maximaux, à la montée et à la descente

1.3.5 Retard inertiel

On appelle retard inertiel le temps minimal de réaction d'une porte, c'est-à-dire le temps minimum pendant lequel les entrées d'une porte doivent être stables pour que celles-ci puissent déterminer correctement sa sortie. Dans le cas d'un retard inertiel (ri) non nul, les variations d'entrées de fréquence trop élevée (>1/ri) ne sont pas prises en compte.

2 LA SIMULATION LOGIQUE

Le but d'un simulateur logique est de déterminer l'évolution temporelle des signaux logiques à l'intérieur d'un circuit, en fonction d'une séquence d'entrée donnée.

Une simulation est caractérisée par :

→ les états logiques sur lesquels elle travaille,
→ les types de retards qu'elle sait gérer,
→ son algorithme de simulation.

2.1 La forme : les types de simulation

Nous avons vu au paragraphe 2 la modélisation du temps par les retards et celle des tensions par les états logiques. Selon leur précision, les simulateurs travaillent sur un nombre plus ou moins limité d'états.

Logique à 2 états : (0, 1)

Elle est souvent associée à un retard nul ou unitaire. Elle n'autorise que la validation du fonctionnement logique du circuit simulé.

Logique à 3 états : (0, 1, X)

Elle est souvent associée à des retards unitaires ou variables. Elle permet d'étudier correctement les problèmes temporels.

Logique à 4 états : (0, 1, X, Ø ou Z)

Elle reprend les caractéristiques de la logique 3 états, plus l'état transitoire qui permet de détecter les aléas de fonctionnement. Certains simulateurs proposent au contraire l'état de haute impédance Z, autorisant notamment la simulation de portes de transfert.

Logique à 5 états : (0, 1, X, Ø, Z) ou (0, 1, X, $Ø_{01}$, $Ø_{10}$)

De plus en plus fine, la logique cinq états permet, selon les simulateurs, de prendre en compte les états transitoires montants et descendants ou l'état de haute impédance. Plusieurs types de retards sont alors disponibles.

Les logiques supérieures à cinq états font toujours intervenir des forces, notamment les forces D, R, Z et U. Le type de simulation le plus fréquent est une simulation à douze états (trois niveaux logiques 0, 1, X et quatre forces D,R,Z, U) [DAI84].

2.2 Le fond : les algorithmes

Il existe 2 algorithmes fondamentaux de simulation logique, à délai unitaire ou compilé (compiled code) et à échéancier (event-driven).

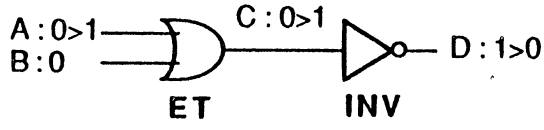
Définitions

Un événement

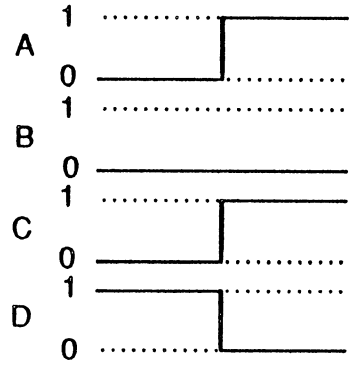
Un événement est le changement d'état d'une équipotentielle. Il est produit par une évaluation. Un événement peut entraîner une évaluation.

Une évaluation

On appelle évaluation, le calcul par un élément logique de sa sortie en fonction de ses entrées.



3 événements (équipotentielles A, B et C)
2 évaluations (portes ET et INV)



Un pas de simulation

Un pas de simulation représente l'intervalle de temps élémentaire géré par la simulation. Tous les événements se produisant au cours d'un même pas de simulation sont donc considérés comme simultanés.

Un cycle de simulation

On appelle cycle de simulation, l'ensemble des pas de simulation compris entre deux séquences d'entrées.

L'activité d'un circuit

Un élément logique est dit actif au cours d'un pas de simulation, si sa sortie change d'état, c'est-à-dire s'il génère un événement. Le pourcentage moyen d'éléments actifs, au cours d'un pas de simulation, représente l'activité du circuit à simuler. Cette activité des circuits intervient dans le calcul théorique de performance d'un simulateur. Son estimation varie de 10% à 20% [CHI86] [TRAY86].

2.2.1 La simulation compilée

Dans ce type de simulation, tous les éléments logiques sont évalués à chaque pas de simulation. L'algorithme est ici très simple, et nécessite peu de variables :

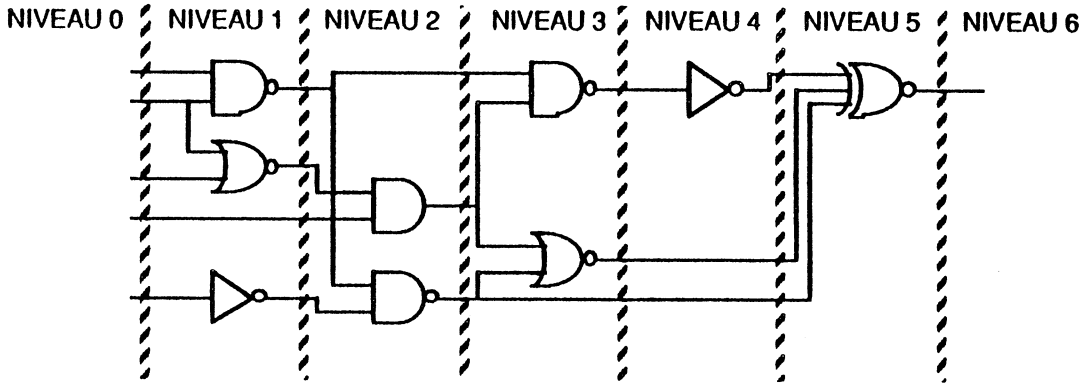
- l'heure courante de simulation : heure-simulation,
- pour chaque équipotentielle, la valeur de ses états courants et précédents par rapport à l'heure de simulation.

Principe de l'algorithme

```
initialiser tous les états précédants à inconnu
initialiser à 0 heure-simulation
tant que heure-simulation < hmax-simulation faire
    heure-simulation := heure-simulation + 1
    initialiser les états-courants des entrées du circuit avec les
    stimulis de l'heure-simulation
    pour chaque porte du circuit à simuler faire
        évaluer la sortie en fonction de l'état précédent des entrées
        état-courant de la sortie := la nouvelle valeur évaluée
    fin pour
    pour toutes les équipotentielles faire
        état-précédent := état-courant
    fin faire
fin tant que
```

Ce type d'algorithme autorise une simulation à délai nul ou unitaire.

Il présente l'avantage très important de pouvoir être compilé avec ses données. Ce type de simulateur travaille directement sur la description structurelle du circuit à simuler en traduisant cette description en code machine exécutable. Durant l'exécution, les instructions sont exécutées en séquence, sans test ni branchement. Le temps de simulation est déterminé une fois la phase de compilation terminée. Le code produit lors de la compilation respecte la hiérarchie des éléments logiques du circuit, ceux-ci sont donc évalués par couches, suivant l'ordre de propagation des signaux.



Décomposition du circuit en couches hiérarchisées

Dans le cas de simulation de circuits asynchrones, le circuit global est découpé en sous-systèmes de fonctionnement synchrone regroupant des composants dont la vitesse est homogène. L'horloge est alors donnée par le sous-système le plus lent, ce qui permet d'éviter des pas de simulation inutiles aux sous-systèmes les plus rapides.

Dans la littérature, ce type de simulation est appelé LCC simulation pour Levelized Compiled Code.

Le code généré étant particulièrement efficace, les simulateurs LCC affichent des performances impressionnantes (la machine de simulation d'IBM, cf. §4.4 et le simulateur AIDA , cf. §4.6).

2.2.2 La simulation à échéancier

C'est la technique de simulation la plus employée. A la différence des simulateurs LCC, la simulation à échéancier n'évalue que les éléments logiques dont les entrées ont changé [BRE72].

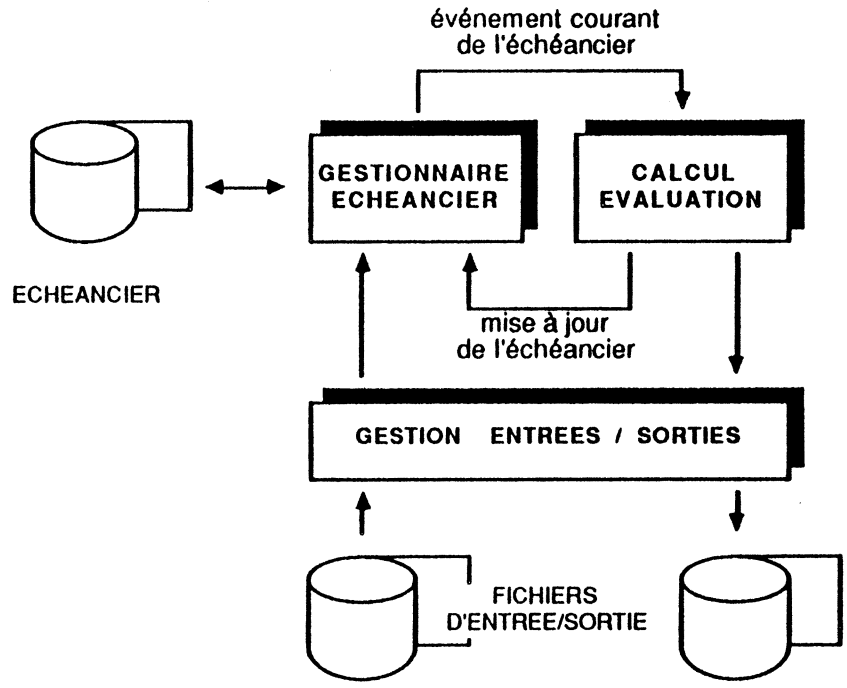
Chaque événement est identifié par trois paramètres :
- l'heure à laquelle il s'est produit,
- l'équipotentielle qui a changé d'état,
- le nouvel état qu'a pris celle-ci.

Tous les éléments sont mémorisés dans un échéancier (event-list) selon l'ordre chronologique. Le traitement d'un événement (on prend toujours le plus récent) consiste à déterminer tous les événements qu'il peut induire à travers des évaluations et à les insérer dans l'échéancier. Le traitement se termine par la suppression de l'événement de l'échéancier. La simulation s'achève lorsqu'il n'y a plus d'événement à traiter, ou lorsque l'heure courante de simulation est supérieure à l'heure limite imposée par l'utilisateur. Le principe de l'algorithme est le suivant :

```
événement : enregistrement de
    temps          : entier
    équipotentielle : entier
    nouvel-état    : ensemble des états de la simulation
```

```
tant que heure-simulation < hmax-simulation faire
    evt-courant := le premier événement de l'échéancier
    effacer cet événement de l'échéancier
    heure-simulation := evt-courant.temps
    pour chaque porte G reliée à evt-courant.équipotentielle faire
        déterminer les nouveaux états de sortie et les délais
        pour chaque sortie I de G faire
            si la sortie évaluée est différente de l'ancienne valeur
            alors
                créer un nouvel événement E
                E.temps := heure-simulation + délai
                E.équipotentielle := I
                E.nouvel-état := la valeur évaluée
                insérer l'événement E dans l'échéancier
            fin si
        fin pour
    fin pour
fin tant que
```

Dans un simulateur à échéancier, les deux opérations de base sont
- l'évaluation des portes,
- la gestion de l'échéancier.



Structure fonctionnelle d'un simulateur à échéancier

La gestion de l'échéancier est le point critique du temps de calcul de ce type de simulateurs. En effet, si quelques instructions-machines suffisent à évaluer une porte, la gestion des événements (interrogation, mise à jour) prend beaucoup plus de temps, surtout pour des circuits complexes dont la sortance est élevée.

2.2.3 Comparaison des algorithmes

Dans les deux cas, l'ordre de grandeur des algorithmes est en N (N étant le nombre de éléments logiques du circuit), et celui de la simulation en N^2 [AMB87]. En fait l'ordre de grandeur est basé sur l'activité A du circuit, dans le cas de :

- l'échéancier $O(n) = A.N$
- compilé $O(n) = N$

Par rapport à la simulation LCC, la simulation à échéancier doit gérer une liste d'événements, ce qui peut lui fait perdre beaucoup de temps. Le tableau ci-dessous [HWA87] permet de comparer l'efficacité des deux algorithmes exécutés sur des ordinateurs classiques. L'échelle est donnée par le nombre d'instructions nécessaires à l'évaluation d'une porte.

SIMULATION LOGICIELLE	LCC	ECHEANCIER
NOMBRE D'INSTRUCTION POUR EVALUER UNE PORTE	25	600

Le domaine privilégié de la simulation compilée est celui des circuits synchrones complexes, surtout si leurs activités sont élevées, mais la vérification temporelle qu'elle permet est limitée (retard nul ou unitaire). La prise en compte de modèles temporels plus fins serait trop coûteuse en terme d'efficacité.

La simulation à échéancier est bien adaptée à tous les types de circuits - synchrones et asynchrones - et surtout, elle autorise une modélisation du temps sans restriction. La simulation à échéancier est particulièrement intéressante lorsque l'activité du circuit à simuler est faible, la barre étant fixée aux alentours des 20% [CHI86].

2.2.4 Les simulateurs logiques logiciels

Les simulateurs logiciels de type LCC sont assez rares, car cette technique est très facile à intégrer matériellement. Néanmoins, on peut citer les simulateurs SSIM [WAN87] et COSMOS [BRY86].

Par contre tous les logiciels "classiques" de simulation sont à échéanciers. Les plus connus sont HILO [FLA81], TEGAS [THO80], BIMOS [BIM85] et EPILOG [EFC86].

15 ETATS LOGIQUES	
3 niveaux	5 forces
0 1 X	E external D driven R resistive L large high impedance Z high impedance

BIMOS

5 ETATS LOGIQUES	
états	description
0	0 à 0 (statique)
1	1 à 1 (statique)
Up	0 à 1 (transition montante)
Down	1 à 0 (transition descendante)
E	transition avec aléa

TEGAS

5 ETATS LOGIQUES	
états	description
0 1 Ø X Z	Ø : transition montante ou descendante

EPILOG

4 ETATS LOGIQUES	
états	description
0 1 X Z	Simulateur logico-fonctionnel

HILO

Comparaison des simulateurs logiciels à échéancier

Ces simulateurs sont le plus souvent du type logico-fonctionnel, permettant également la simulation de défauts (HILO) ou la génération de vecteurs de test (EPILOG, TEGAS).

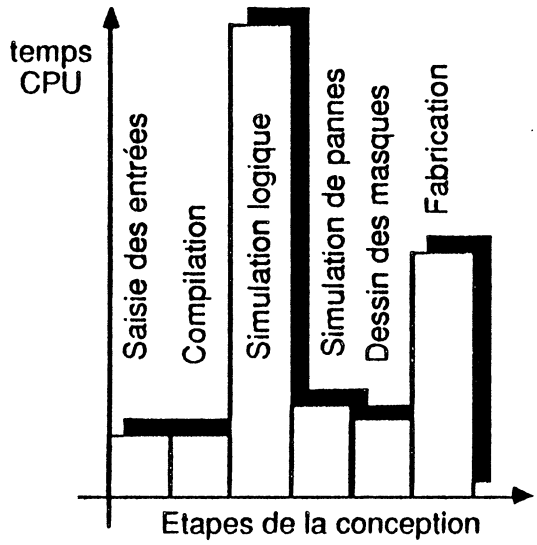
3 TECHNIQUES D'ACCELERATION

Pourquoi accélérer les simulations ?

La réponse est que sur des machines traditionnelles, les temps de simulation de circuits complexes sont beaucoup trop longs pour être exploitables [MOT85].

Etudions par exemple le temps que prendrait la simulation du fonctionnement réel pendant 10ms d'un circuit de 10^5 portes à une fréquence de 20 Mhz. Si l'on prend une activité moyenne de 10%, il nous faudra évaluer 10^5 x 10% x 20.10^6 x 10^{-2} portes soit 2.10^9 évaluations. Sur une machine conventionnelle, la simulation (à échéancier) d'une évaluation prend environ 500 instructions machines, les performances habituellement retenues sont donc de l'ordre de 2.10^3 évl/sec et par mips. La simulation de notre circuit prendra donc entre ... 11 et 12 jours sur un ordinateur classique de type VAX 11/780.

Ce calcul simpliste met en évidence le goulot d'étranglement que représente la simulation logique dans le temps de conception d'un circuit (the logic simulation bottleneck). Le graphique ci-dessous permet d'appréhender les temps de calcul des différentes étapes de la conception de circuits VLSI.



Répartition des temps de calcul en CAO
(source ZYCAD)

Pour accélérer ces algorithmes (comme pour tout logiciel), le moyen le plus efficace est de développer une machine câblée dédiée à ce type d'application; on appelle ces machines des accélérateurs matériels (hardware accelerators). Ces machines sont apparues vers les années 80, d'abord connectées à des stations de travail, puis vers 1982-83 sous la forme de machines autonomes accélérant, de manière ponctuelle ou globale, les étapes de la CAO [BLA84]. Leur succès a été d'autant plus grand qu'elles correspondaient exactement à une demande d'outils efficaces pour la conception de circuits complexes (cent mille à un million de transistors MOS), particulièrement dans le domaine de la simulation logique.

3.1 Techniques générales

Les techniques d'accélération résultent de deux approches différentes :

- → spécialisation des tâches à effectuer, par développement de processeurs spécialisés,
- → exécution simultanée de tâches sous forme d'architecture parallèle ou d'architecture pipeline.

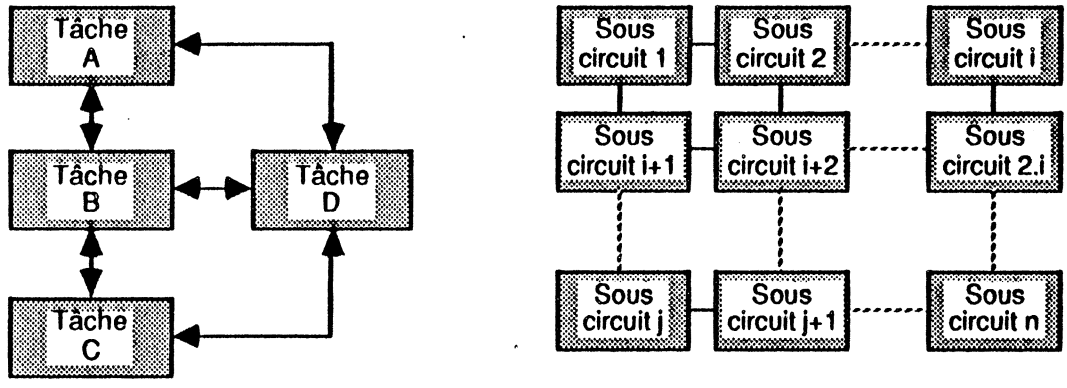
La première approche est simple à mettre en œuvre. A partir de l'algorithme de base, certaines opérations "longues" (gestion de l'échéancier, calcul des évaluations ...) peuvent être intégrées dans un processeur spécialisé afin d'en réduire le temps d'exécution. Ces processeurs peuvent être de type micro-programmés [AGR87] ou faire appel à des architectures RISC (Reduced Instruction Set Computer), comme le simulateur développé par AIDA (cf. §4.4). La nature séquentielle de l'algorithme est maintenue, il ne faut plus alors que quelques instructions pour réaliser la fonction.

La deuxième approche (exécution simultanée de tâches) fait appel au parallélisme à 2 niveaux :

- → inhérent à l'algorithme de simulation
- → inhérent au circuit simulé

Le parallélisme algorithmique est basé sur le partitionnement fonctionnel des tâches de l'algorithme, chaque processeur exécute alors une partie de l'algorithme initial, et communique au processeur aval le résultat de ses calculs (traitement pipeline des données). Par exemple, la gestion de l'échéancier peut être décomposée en plusieurs opérations élémentaires telles que la lecture d'un événement, sa suppression de l'échéancier, ou l'insertion dans celui-ci de nouveaux événements ...

Le parallélisme physique repose sur le partitionnement du circuit à simuler, la simulation des sous-circuits est alors réalisée par des processeurs indépendants exécutant en parallèle le même algorithme. Le partitionnement entraîne l'échange, entre les différentes unités de simulations, de certains événements, et surtout d'informations permettant d'assurer la synchronisation des différentes horloges de simulation.



TRAITEMENT PIPELINE D'UN ALGORITHME SEQUENTIEL

ALGORITHME DISTRIBUE SUR UNE ARCHITECTURE PARALLELE

Architecture pipeline et parallèle

Ces architectures sont complémentaires et peuvent être combinées, la plupart des accélérateurs matériels utilisent d'ailleurs ces deux techniques (ZYCAD).

Il faut remarquer qu'un parallélisme massif du circuit réduit l'intérêt du traitement pipeline de l'algorithme de simulation logique.

Robert J. Smith [SMI86] donne une classification des ces techniques selon leurs performances. Au bas de l'échelle, les

ordinateurs tous usages, qui prennent entre 200 et 10^3 instructions pour évaluer une porte, affichent des performances de l'ordre de 10^3 à 5.10^3 éval/mips. Ces performances peuvent être améliorées par :

→ utilisation d'un langage spécialisé, optimisé pour les algorithmes de simulation permettant l'évaluation d'une porte en 100 ou 500 instructions. Les performances peuvent alors atteindre 2.10^3 à 10^5 éval/mips, toujours sur une machine générale;

→ utilisation d'un processeur dédié à la simulation logique intégrant l'algorithme de façon micro-programmé, 30 à 60 instructions suffisent à évaluer une porte. Les performances peuvent alors atteindre 10^5 à 3.10^5 éval/sec;

→ introduction du traitement pipeline dans le modèle précédant (gestion événement, évaluation ...). On passe alors à 10 instructions/porte, ce qui porte la vitesse de simulation autour de 10^6 éval/sec;

→ amélioration du modèle précédent en optimisant les accès mémoires, on double la vitesse : 2.10^6 éval/sec;

→ enfin, la répartition de l'algorithme de simulation sur plusieurs processeurs permet d'atteindre des vitesses de l'ordre de 5.10^7 éval/sec.

Ces résultats sont valables pour l'accélération de la simulation à échéancier. Dans le cas d'accélérateurs intégrant une simulation compilée, les performances peuvent atteindre 10^7 éval/sec [PFI82] en utilisant une architecture pipeline où les processeurs spécialisés travaillent avec une mémoire propre.

3.2 Techniques propres aux simulateurs LCC

Le grand intérêt de la simulation compilée, par rapport à la simulation à échéancier, est la compilation globale de l'algorithme avec ses données. D'autre part, il n'y a pas à proprement parler d'horloge de simulation : c'est la cadence de l'exécution qui gère l'heure de simulation.

A partir de là, la technique la plus simple, pour accélérer ce type de simulation, est de développer des processeurs dédiés à

l'exécution de ces programmes. D'autre part, les instructions de bases sont ici très simples et réduites. On se trouve alors dans le domaine d'application des processeurs à architectures RISC. Les simulateurs logiques développés par Aida (cf. §4.4), Mentor Graphics (Computer Engine) [MIL87] entrent dans cette catégorie.

3.3 Techniques propres aux simulateurs à échéancier

Il est difficile de paralléliser l'algorithme de simulation à échéancier, car l'échéancier est une ressource globale qui doit être accessible à tous les processeurs; néanmoins, la technique la plus couramment utilisée consiste à implémenter, sur une architecture parallèle, un algorithme à échéancier distribué [MIS86] [WON86]. Le circuit à simuler est partitionné en sous-circuit sur lequel chaque processeur exécute le même algorithme de simulation à échéancier. Les processeurs gèrent un échéancier local et peuvent échanger des événements entre eux, lorsqu'une évaluation a engendré un événement concernant une porte d'un autre sous-circuit.

Le problème est alors de gérer les communications entre sous-circuits, afin d'éviter qu'un processeur ne reçoive un événement antérieur à son heure locale de simulation. Différents protocoles sont présentés dans [MIS86].

Un autre problème de la simulation à échéancier réparti est le partitionnement même du circuit, surtout lorsque celui-ci est complexe : en effet, il faut éviter qu'à certains pas de la simulation certains processeurs soient complètement inactifs et d'autres surchargés [AGR86] [FRA87] [TUR85].

La technique du pipelining ne pose en revanche aucun problème pour être adaptée à la simulation logique à échéancier. Elle est rarement employée seule, mais très souvent en complément d'un échéancier réparti. Selon les simulateurs, l'ensemble des opérations nécessaires au traitement d'un événement est décomposé entre trois et onze étapes. Le traitement de ces étapes est alors intégré soit dans un processeur pipeline (ZYCAD, cf. §4.1), soit plusieurs processeurs communiquant par un bus rapide (DAISY, cf.

§4.2). Certains simulateurs utilisent des processeurs pipeline micro-programmables (MARS, cf. §4.5).

Enfin, il faut citer l'implantation pipeline de simulation à échéancier sur transputer (HILAS [DYS87] [GRA87], LL3T [LLT87]).

4 LES MACHINES ACTUELLES

Chaque année, le magazine VLSI Systems Design fait le point sur tous les simulateurs logiques commercialisés [VLS88a]. Cette année, près d'une cinquantaine sont ainsi répertoriés (dont la majorité sont des simulateurs logiciels tournant sur de petites machines) prouvant ainsi la demande importante pour ce type de simulateurs. Cette revue a également publié une comparaison sur les accélérateurs matériels [VLS88b]. Parmi toutes ces machines et celles qui ne sont pas commercialisées (usage interne, en cours d'étude ou prototype), nous en avons retenu 7 qui représentent les principales formes d'intégration .

Ces machines utilisent toutes les techniques d'accélération exposées précédemment et souvent en combinent plusieurs. Il est difficile de comparer ces différentes réalisations, pour plusieurs raisons :

→ pour des questions commerciales, les auteurs des articles cités en références, ne présentent pas toujours en détail leurs machines. Toutes les techniques étant connues, l'intégration de différentes architectures tient plus du savoir-faire que de la science,

→ très peu d'auteurs sont réellement objectifs dans les performances qu'ils annoncent. Très souvent ils ne prennent pas en compte les temps de calculs nécessaires à la compilation des données (qui peuvent être aussi longs que la simulation elle-même !), et passent sous silence le ralentissement dû aux communications entre processeurs.

De même, les performances annoncées en termes d'événements ou d'évaluations par seconde, ou encore d'évaluations en équivalent porte à 2 entrées par seconde, peuvent prêter à confusion, d'autant plus que les évaluations de performances (benchmark) ne sont pas toujours neutres.

Les 7 machines présentées sont :

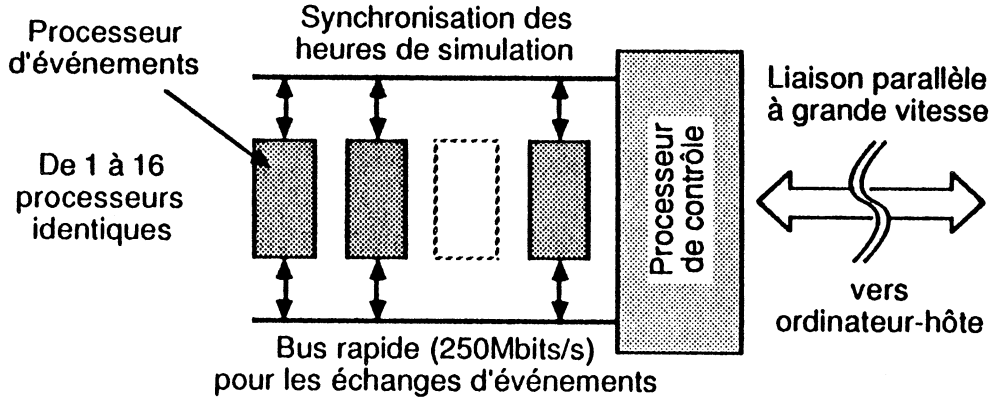
- Logique Evaluator série 1000 (LE1000) de ZYCAD
- Megalogician de DAISY SYSTEM
- Hal II de NEC

- AIDA simulator
- MARS de AT&T BELL Laboratories
- Yorktown Simulation Engine d'IBM
- Realfast accelerator de VALID

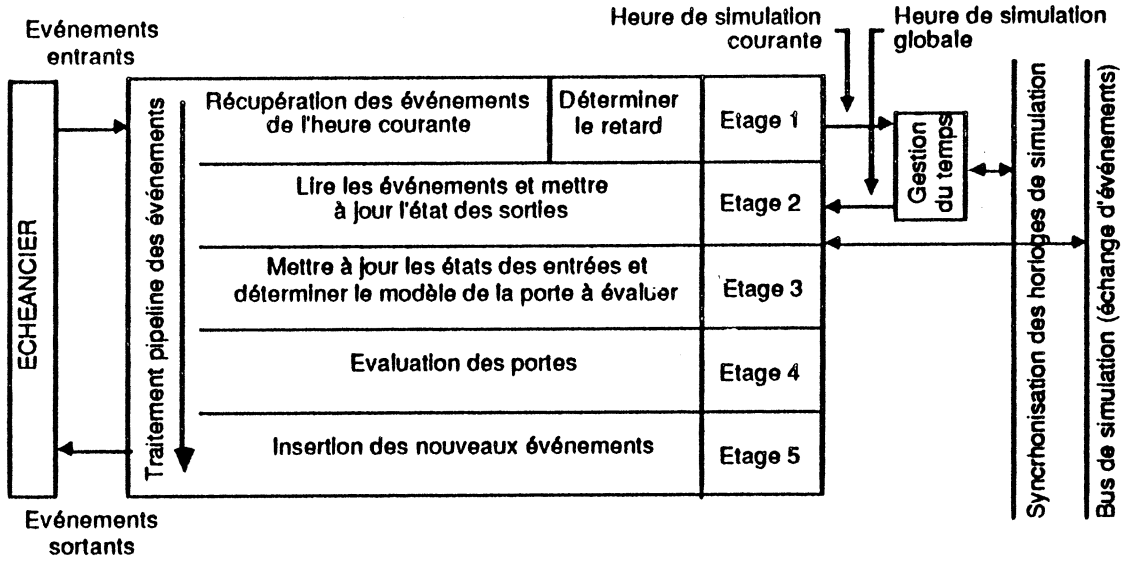
4.1 Logic Evaluator série 1000 (LE1000)

Zycad est certainement la marque la plus connue dans le domaine des simulateurs. Zycad commercialise sous la série LE1000 [ZYC84] 8 simulateurs logiques (LE1001-A → LE1032).

L'architecture est la même pour tous les modèles, elle est basée sur une simulation à échéancier distribuée sur un maximum de 16 processeurs. Chaque processeur exécute un algorithme câblé, organisé de façon pipeline sur 5 étages. Les processeurs synchronisent leurs simulations sur une horloge-maître en utilisant une "roue de synchronisation" (time wheel), et échangent des événements par l'intermédiaire d'un bus très rapide (250Mbits/s).



Architecture générale du Logic Evaluator de Zycad



Architecture d'un processeur d'événements

Le système LE1000 exécute une simulation logique à douze états (trois niveaux logiques : 0, 1 et X et 4 forces : haute impédance, résistive, forte et externe). 3 délais sont possibles (connections, porte et entrée), 3 modèles standards d'éléments sont disponibles : unidirectionnel, bidirectionnel et porte câblée. Les tables de vérité sont redéfinissables par l'utilisateur. Ce simulateur détecte les oscillations de boucles à délai nul (fin de simulation). Enfin, le LE1000 permet la simulation de mémoire (ROM/RAM), ainsi que celle de panne.

Le système LE1000 se connecte à un ordinateur-hôte du type VAX, IBM, APOLLO ou autre via un lien.

Zycad annonce pour le modèle LE1002 une vitesse de l'ordre de 10^6 évé^t/sec sur un maximum de 64.10^3 éléments. La configuration maximale (LE1032, 16 processeurs) peut traiter des circuits de plus de 10^6 d'éléments à la vitesse de 16.10^6 évé^t/sec.

Remarque : la vitesse maximale de 16.10^6 évé^t/sec est très optimiste, car il s'agit d'une vitesse théorique ne tenant pas compte des communications entre processeurs.

Le système LE1000 est commercialisé à un prix compris entre $150K et $2.1M.

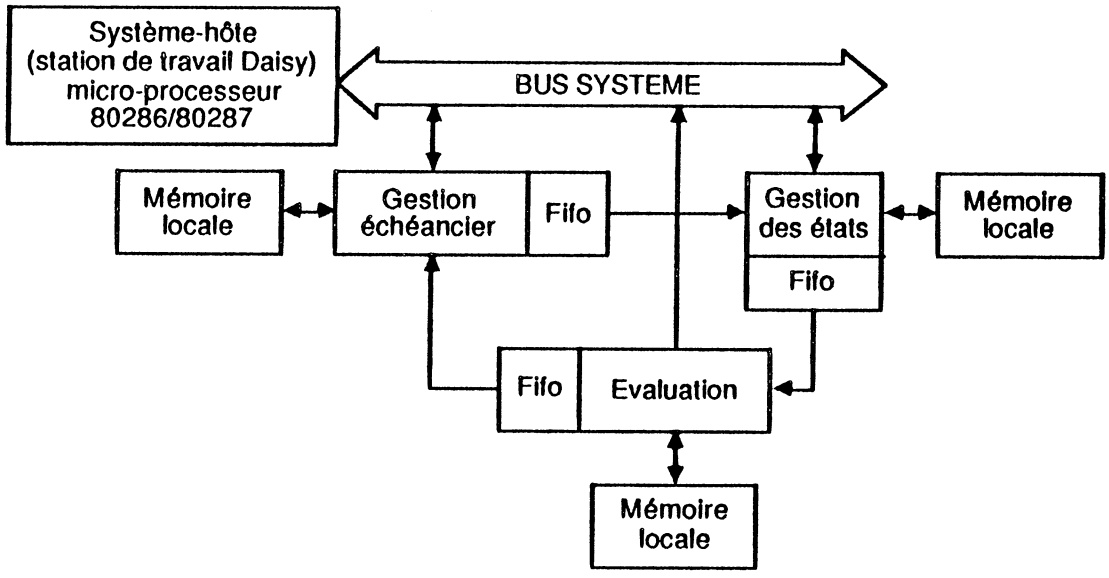
Zycad commercialise également plusieurs machines de simulation à échéancier entièrement câblé :
→ The SPRINTOR : carte d'extension pour IBM/PC-AT ou Micro-VAX autorisant la simulation de circuit comportant moins de seize mille éléments (modèle 30) à la vitesse de 2.10^5 évé!/sec,
→ The EXPEDITOR : accélérateur matériel indépendant devant être relié à un ordinateur-hôte (station de travail ...) permettant de simuler des circuits de moins de 32.10^3 éléments (modèle 200) à une vitesse de 10^6 évé!/sec.

Enfin, Zycad commercialise The System Development Engine (SDE) permettant la mise au point de système complet (matériel et logiciel). Ce système, qui comporte 2 modèles (8016 et 8032), intègre un simulateur logique à échéancier entièrement câblé. Les performances annoncées sont impressionnantes : simulation d'un maximum de plus de 10^6 éléments à la vitesse de 10^9 évé!/sec.

4.2 Megalogician

Megalogician [GIN83] est l'accélérateur matériel dédié aux stations de travail DAISY.

Megalogician utilise la technique de la simulation à échéancier, le traitement pipeline des données est réparti sur 3 processeurs (traitement pipeline en anneau) :
- Queue Unit : un processeur d'événements (gestion échéancier),
- State Unit : un processeur d'états (gestion des états des nœuds du circuit),
- Evaluation Unit : un processeur d'évaluation (évaluation de la sortie des éléments logiques du circuit).



Architecture du Megalogician

Les 3 processeurs sont programmables. Le domaine d'application du Megalogician est donc très large et les simulations de toutes catégories sont donc possibles. Daisy a développé son propre langage de simulation (DSL : Daisy Simulation Langage) travaillant sur 3 états et 4 forces et autorisant des délais de montée et de descente distincts. Les éléments disponibles vont des simples portes booléennes de base jusqu'à la modélisation sous formes d'équations de fonctions complexes.

L'accélérateur de simulation Megalogician permet de multiplier par mille les performances de la station de travail dans ce domaine, les vitesses annoncées sont de l'ordre de 10^5 évé^t/sec pour 256.10^3 portes logiques.

Le système Megalogician est commercialisé au prix de $62K.

Le Megalogician s'insère dans la gamme des simulateurs Daisy entre :
- le Personal Megalogician (10^5 éval/s sur 64.10^3 portes) qui nécessite comme le Megalogician d'être intégré à un système-hôte,
- et le Gigalogician (3.10^6 éval/s sur des circuits d'au plus 256.10^3 portes logiques dans la configuration maximale). A la différence des 2 autres simulateurs de la gamme, le Gigalogician est autonome.

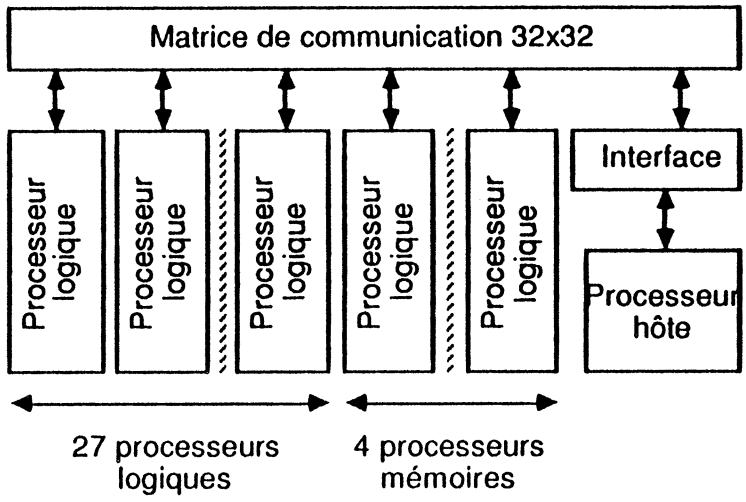
4.3 HAL II

HAL II [TAK86] est un simulateur logique développé par NEC qui reprend les fonctionnalités de la première version de HAL [SAS83]. HAL II est un simulateur logique multi-niveau supportant les simulations au niveau porte, bloc et fonctionnel (format d'entrée FDL [KAT83]), destiné surtout à la simulation de gros circuits.

La notion de bloc est fondamentale dans la modélisation du circuit. Un bloc-logique représente un ensemble plus ou moins complexe de portes logiques représentant une entité fonctionnelle. De même, un bloc-mémoire sera composé uniquement d'éléments-mémoire (RAM ...). Evidemment, le partitionnement du circuit à simuler respectera cette découpe en blocs, définie par le concepteur, afin de minimiser les communications de bloc à bloc.

La simulation est une simulation à 2 états (0 et 1), à délai nul.

La simulation logique à échéancier implémentée sur les processeurs logiques est traitée de manière pipeline. Les processeurs sont formés d'un processeur de bloc (contrôle de la simulation et gestion de l'échéancier) et d'un simulateur de bloc-logique (simulation de la fonction du bloc-logique ou du bloc-mémoire).



Architecture générale de la machine HAL II

Les performances annoncées par les auteurs portent sur différentes simulations de circuits de 10^5 portes. Elles ont montré dans le cas d'une simulation au niveau bloc, un temps de cycle de une milli-seconde, 10 milli-secondes pour le niveau porte, ce qui donne des vitesses de l'ordre de 5.10^5 et 5.10^6 éval/sec. Les temps de compilation (génération des modèles) étaient respectivement d'environ 95 et 780 secondes.

Ces performances montrent que la simulation au niveau bloc est 10 fois plus rapide que la simulation au niveau porte.

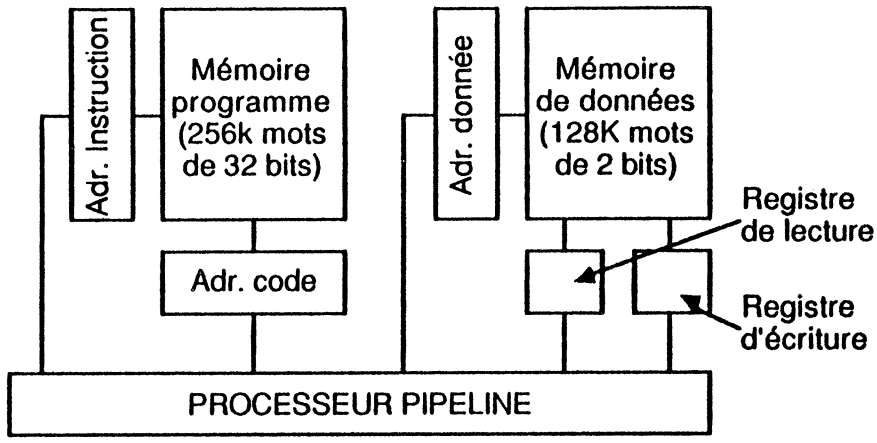
4.4 AIDA Simulator

Le simulateur logique développé par AIDA est un simulateur du type LCC [CHI86]. Il est implanté sur les stations de travail 32 bits de AIDA, mais ses performances peuvent être sensiblement améliorées, par l'adjonction d'un co-processeur spécialisé à architecture RISC [HWA87].

AIDA autorise des simulations à 4 états : 0, 1, X et Z. Le coprocesseur de simulation travaille sur 10 primitives logiques :

- six primitives booléennes :
  AND, NAND, OR, NOR, XOR, XNOR;
- quatre fonctions logiques :
  Transmission gate, Dot, Weak driver, Tri-state driver.

L'accélérateur de simulation AIDA est composé d'une mémoire d'instructions (programme), d'une mémoire de données (états) et d'un processeur pipeline à architecture RISC.



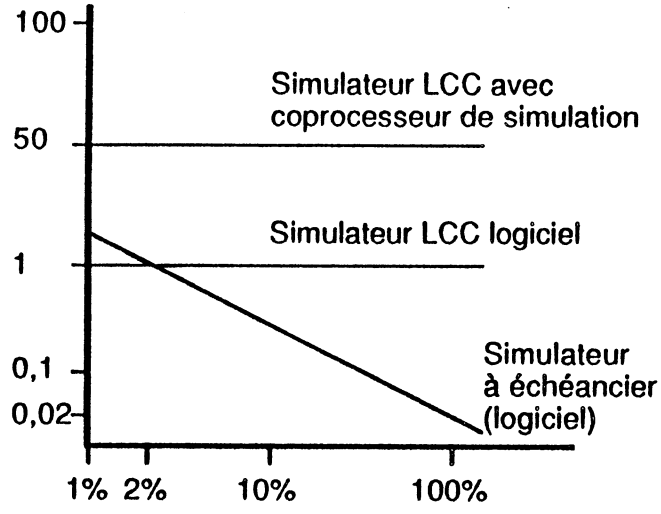
Architecture de l'accélérateur de simulation AIDA

Le jeu d'instructions (réduit) du processeur de simulation est composé de 16 codes-opération :

LA load accumulator
AN and
NA nand
OR or
NO nor
XO exclusive-or
XN exclusive-nor
TR tri-state
TX transmission gate
DO dot
WD Weak driver
CO compare
NI non operation instruction
ST store accumulator
LS load store control register
HA halt

Les performances annoncées par les auteurs sont de l'ordre de 5.10^6 évé^{l}/sec.

Afin de permettre tous les modes de simulation possibles, AIDA intègre également un simulateur à échéancier. Le tableau ci-dessous permet de comparer les performances des simulateurs LCC (logiciel et matériel) et à échéancier.



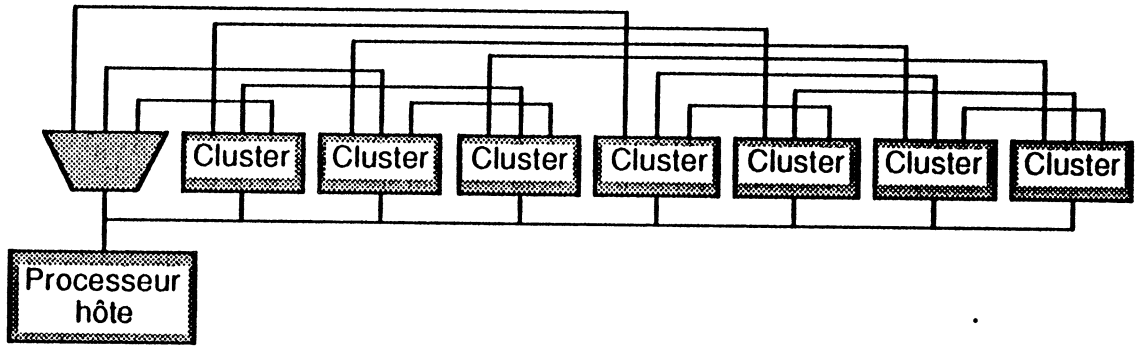
Performances comparées des différents simulateurs

Le simulateur AIDA est, selon les versions, commercialisé à un prix compris entre $10K et $200K.

4.5 M.A.R.S.

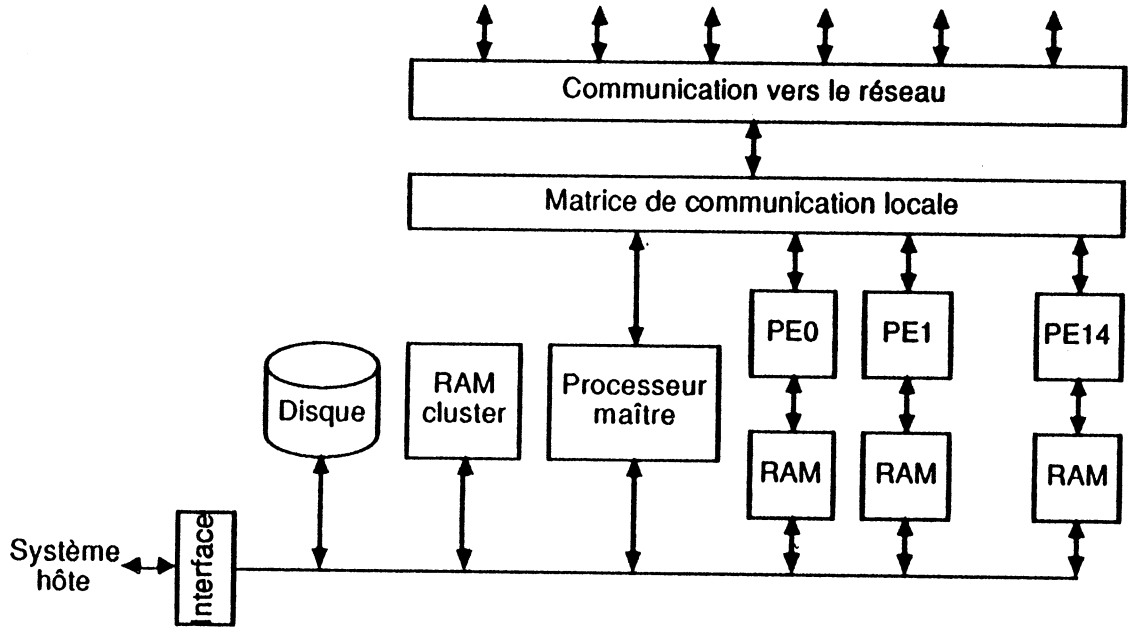
M.A.R.S [AGR87] (Microprogrammable Accelerator for Rapid Simulations) est un accélérateur matériel multi-processeurs de simulation, développé par les laboratoires AT&T Bell. Cette machine permet d'implémenter de manière efficace des complexes de différentes familles. L'architecture de MARS est en particulier bien adaptée à l'accélération de simulations à échéancier.

L'architecture de MARS est hiérarchisée. Au plus haut niveau, MARS est constitué d'unités de traitement (clusters) organisées en réseau et communiquant par des nœuds.



Architecture générale de MARS
(ici 7 clusters intégrés dans un réseau cubique de dimension 3)

Le processeur-hôte est chargé de l'interface utilisateur. Dans chaque cluster, le traitement des données est réalisé de manière pipeline sur un ensemble d'au plus 14 processeurs micro-programmables. Chacun de ces processeurs élémentaires gère une mémoire (64K mots de 16 bits) qui lui est propre.



Architecture des clusters

MARS à l'origine a été conçu pour la simulation de circuit en particulier, et pour servir de support aux logiciels de CAO en général. La simulation logique a fait l'objet d'études particulières notamment sur les algorithmes [DAL87] et sur un compilateur de description

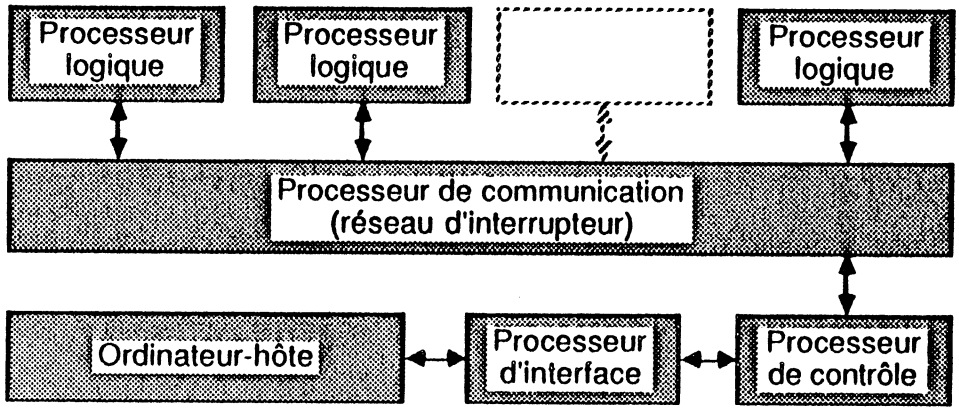
[NOR87]. Plusieurs algorithmes de simulation ont été testés sur MARS : simulation à délais variables en 1 ou 2 phases et simulation à délai unitaire.

Les performances de ces différents algorithmes, sur un système composé d'un unique cluster sont respectivement de l'ordre de 8.10^5, 6.10^5 et $1{,}5.10^6$ éval/sec. Les auteurs espèrent atteindre des vitesses de l'ordre de 10^9 éval/sec sur des configurations maximales de 256 clusters.

4.6 The Yorktown Simulation Engine

La compagnie IBM, pour ses besoins internes, a développé 3 générations de machines de simulation :
- the Logic Simulation Machine (LSM) [HOW83]
- the Yorktown Simulation Engine (YSE) [PFI82] [DEN82]
- the Engineering Verification Engine (EVE) [DUN84]

La machine YSE, comme les autres, est architecturée autour d'un ensemble de processeurs. Les échanges d'informations sont pilotés par un processeur de communication gérant une matrice d'interrupteurs (crossbar switch). Les communications vers l'ordinateur-hôte sont de leur côté, prises en charge par un processeur d'interface.



Architecture du Yorktown Simulation Engine

Les 256 processeurs de la machine YSE (resp. 64 et 256 pour LSM et EVE) exécutent en parallèle un algorithme de simulation

compilé. Le traitement est de type pipeline à 8 étages permettant l'évaluation de 4096 fonctions logiques différentes à 4 entrées. La simulation est une simulation à 4 états à délai unitaire ou nul. Certains processeurs de type ARRAY sont spécialisés dans la simulation d'éléments mémoires (RAM, ROM ...) évitant ainsi de saturer les processeurs logiques, s'ils avaient à remplir cette tâche.

Le partitionnement du circuit à simuler fait appel à plusieurs techniques destinées à minimiser les communications inter-processeurs :
- remplissage des processeurs selon la liste de connectivité,
- même remplissage que précédemment, mais on intègre, dans le même processeur, les prédécesseurs de chaque porte,
- placement aléatoire et amélioration par relaxation.

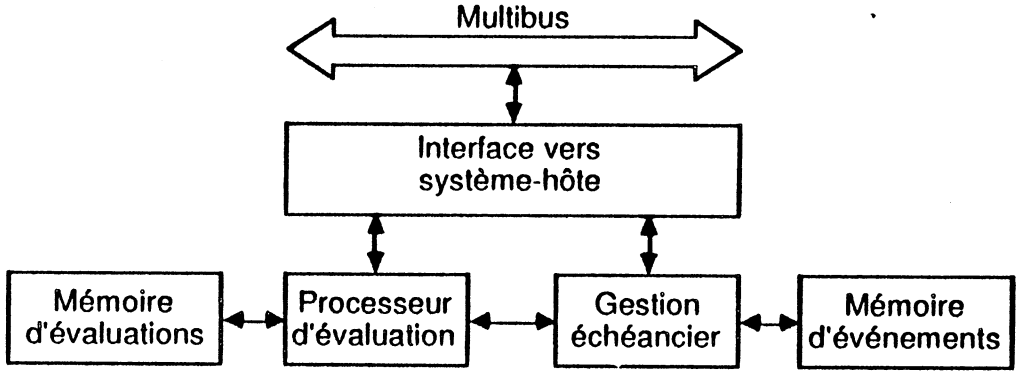
Selon les auteurs, ces techniques donnent de bons résultats.

Quant aux performances, les processeurs évaluent une porte en 80ns, ce qui donne une vitesse théorique de $3{,}2.10^9$ éval/sec, pour un circuit limité à 10^6 portes (resp. 64.10^3 et 2.10^6 pour LSM et EVE). Les auteurs qui annoncent des performances autour de 3.10^9 éval/sec sont donc optimistes.

4.7 Realfast

Realfast [TRA86] est un accélérateur matériel de simulation destiné aux stations de travail Scald de VALID.

L'architecture de Realfast repose sur le traitement pipeline de la simulation à échéancier. 2 processeurs micro-codés sont dédiés à la gestion de l'échéancier d'une part et à celle de l'évaluation des éléments logiques d'autre part.



Architecture de l'accélérateur de simulation Realfast

Chaque processeur possède une mémoire locale de 32 Méga-octets. Le processeur d'événement est un processeur en tranche série AM 2900 de 4 Mips, son programme micro-codé est en PROM. Le processeur d'évaluation est formé de processeurs AM 2901 32 bits à pipeline micro-programmables et d'une ALU spécialisée pour les évaluations logiques.

Les performances du simulateur Realfast sont de l'ordre de 5.10^5 éval/sec (soit 100 à 500 fois supérieure aux simulations purement logicielles) pour une capacité d'environ 3.10^6 portes.

Comme pour le Megalogician, Realfast, initialement conçu pour accélérer la simulation de circuits, peut être programmé pour accélérer les diverses parties de la chaîne de CAO.

Le simulateur Realfast est commercialisé à un prix compris entre $35K et $100K.

4.8 Remarques sur les accélérateurs matériels de simulation

Le tableau récapitulatif des principales caractéristiques de simulateurs présentés amène plusieurs remarques :

→ le traitement pipeline de données étant beaucoup plus facile à intégrer, aucune machine ne présente une architecture simplement parallèle;

→ excepté la machine SDE de Zycad, il semble que les simulateurs à échéancier aient beaucoup de mal à dépasser le cap du milliard d'évaluations par seconde. En effet, cette vitesse ne pourrait être approchée qu'en utilisant un parallélisme massif. Or la technique de simulation par échéancier distribué atteint ici ses limites, les temps de communication et de synchronisation entre les processeurs devenant trop importants;
→ une architecture mixte (parallèle et pipeline) semble être nécessaire pour atteindre des vitesses élevées de simulations;
→ excepté MARS dont les performances restent à prouver, les performances des simulateurs matériels d'usage général ne peuvent égaler celles des simulateurs "purs";
→ les simulateurs logiques les plus performants atteignent actuellement des vitesses de l'ordre du milliard d'événements par seconde.

Machine	Architecture (parallèle/ pipeline)	Algorithme (échéancier/ LCC)	Performances (maximales)	Utilisation (simulation/ général)
LE 1000	mixte	échéancier	16.10^6	simulation
MEGALOGICIAN	pipeline	échéancier	100.10^3	général
HAL II	mixte	échéancier	5.10^6	simulation
AIDA	pipeline	LCC	5.10^6	simulation
MARS	mixte	échéancier	10^9	général
YSE	mixte	LCC	10^9	simulation
REALFAST	pipeline	échéancier	500.10^3	général

Comparaison des caractéristiques principales des simulateurs présentés

ANNEXE 2

EXEMPLE DE PROGRAMME LOF

Il s'agit d'un programme assemblant automatiquement un tampon dont les nombres de bits d'information et de routage sont passés en paramètres.

```
MODULE BUFFER;
\COMPILE;
EXTERNAL PAS, FLAG, AMPHORN, REGISTRE, PORTE_3E, AMPLI;

     \* APPEL ET PASSAGE DE PARAMETRES                *\
     \* RESULTAT:= NBIT_INFO _BUFFER NBIT_ROUTAGE;  *\

NBIT_INFO:= ;
NBIT_ROUTAGE:= ;
NBIT:= NBIT_INFO+NBIT_ROUTAGE;

     \* BLOC GAUCHE DE L'ASSEMBLAGE DU BUFFER : LE FLAG *\

WITH FLAG, ALL DO;
  PIN (0, HEIGHT(FLAG)+6*PAS                ; 'VDD0G', VDD, ALU);
  PIN (0, HEIGHT(FLAG)+4*PAS+HEIGHT(AMPLI); 'VSS0G', VSS, ALU);
  PIN (0, HEIGHT(FLAG)+6*PAS                ; 'VDD0D', VDD, ALU);
  PIN (0, HEIGHT(FLAG)+4*PAS+HEIGHT(AMPLI); 'VSS0D', VSS, ALU);
  FLAG:= WHATWITH;
ENDWITH;
FLAG:= OVERSIZE (FLAG,
  {HEIGHT(AMPLI)+4*PAS, 4*PAS, HEIGHT(PORTE_3E)-HEIGHT(REGISTRE), 2*PAS});

     \* BLOC CENTRAL DE L'ASSEMBLAGE DU BUFFER : LES AMPLIS DES COMMANDES *

WITH AMPHORN DO;
  WIRE ('EH'; O, PAS, ALU; O, 0, POLY);
  TDK:= WIREEDGE;
  WIREEND ('LECT1', ENTREE);
  PIN(TDK; 'LECT2', ENTREE);
  NAME 'VSSG'  AS 'VSS2';
  NAME 'VDD2D' AS 'VDD4';
  NAME 'VDD1D' AS 'VDD3';
  DISCARD 'EH', 'H', 'HB', 'VSSD';
  PIN ( LENGTH(AMPHORN), HEIGHT(AMPHORN)+PAS; 'VSS_BI', VDD, ALU);
  AMP_INF:= WHATWITH;
ENDWITH;
WITH AMPHORN DO;
  WIRE ('EH'; O, 2*PAS, ALU; O, 0, POLY);
  TDK:= WIREEDGE;
  WIREEND ('ECR1', ENTREE);
  PIN(TDK; 'ECR2', ENTREE);
  NAME 'VSSG'  AS 'VSS1';
  NAME 'VDD2D' AS 'VDD2';
  NAME 'VDD1D' AS 'VDD1';
  DISCARD 'EH', 'H', 'HB', 'VSSD';
  PIN ( LENGTH(AMPHORN), HEIGHT(AMPHORN)+PAS                ; 'VSS_TE', VDD, A
  PIN ( LENGTH(AMPHORN), HEIGHT(AMPHORN)+3*PAS              ; 'VDD0', VDD, ALU
  PIN ( LENGTH(AMPHORN), HEIGHT(AMPHORN)+HEIGHT(AMPLI)+PAS; 'VSS0', VSS, ALU
  AMP_SUP:= WHATWITH;
ENDWITH;
AMPHORN:= AMP_INF V(2*PAS) AMP_SUP;
AMPHORN:= OVERSIZE
  (AMPHORN, {HEIGHT(AMPLI), 0, HEIGHT(PORTE_3E)-HEIGHT(REGISTRE), 2*PAS});
```

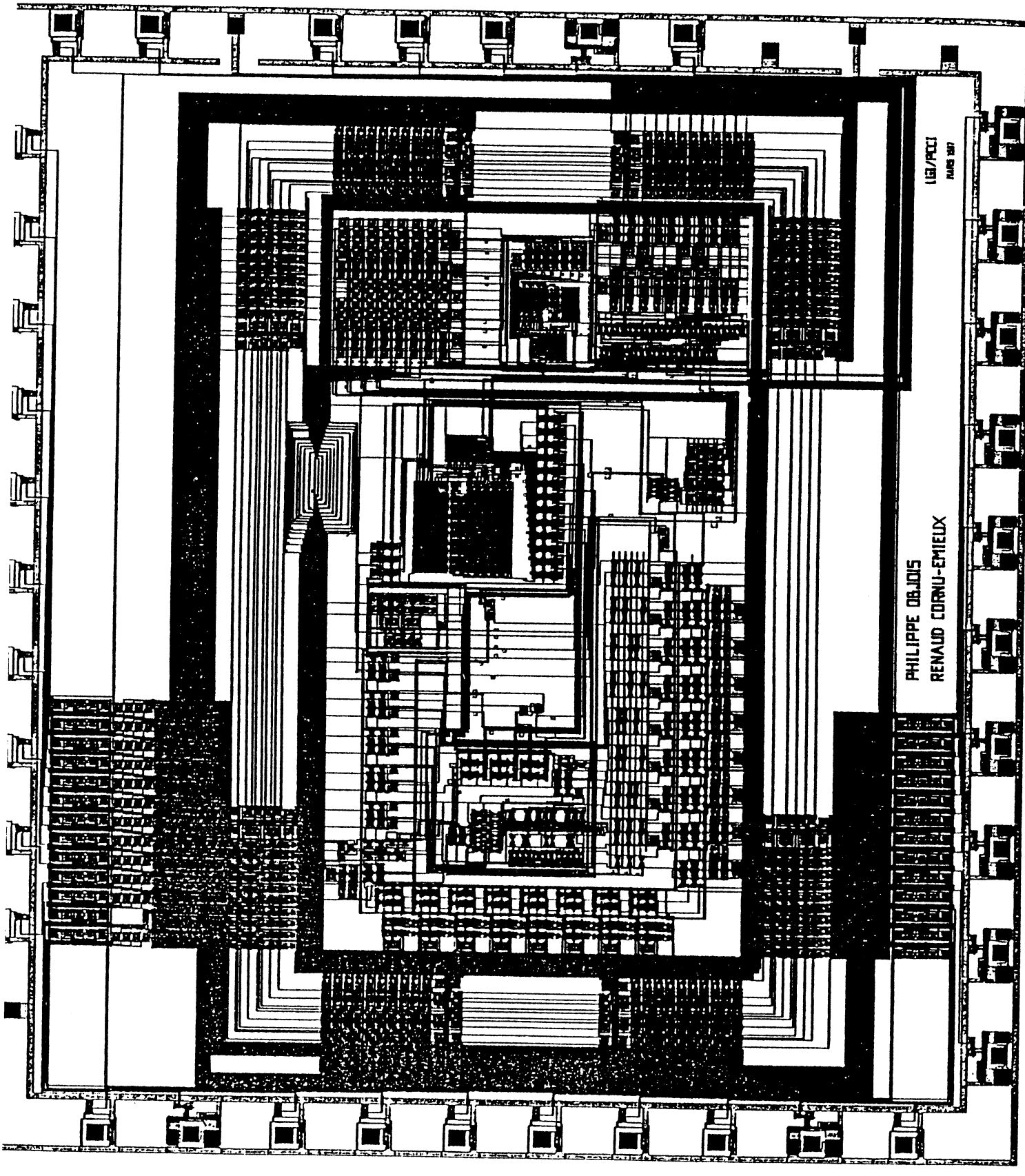
```
      \* BLOC DROIT DE L'ASSEMBLAGE DU BUFFER : LE TAMPON *\
      \* TAMPON = NBIT FOIS UNE TRANCHE_DE_BIT             *\

WITH PORTE_3E DO;
  WIRE (-3*PAS, HEIGHT(PORTE_3E)+PAS; E, LENGTH(PORTE_3E)+3*PAS, ALU);
  WIREEND('VSS_TER', VSS);
  WIRE ('EG'   ; N, 2*PAS, ALU, POLY; N, 0, ALU);
  WIRE ('VDD1G'; O, 3*PAS, ALU);
  WIRE ('HG'   ; O, 3*PAS, ALU);
  WIRE ('VSSG' ; O, 3*PAS, ALU);
  WIRE ('HBG'  ; O, 3*PAS, ALU);
  WIRE ('VDD2G'; O, 3*PAS, ALU);
  WIRE ('VSS2G'; O, 3*PAS, ALU);
  POS_VSS5G:= WIREEDGE;
  WIRE ('SG'   ; O, PAS, ALU; O, 0, POLY);
  TDK:= WIREEDGE;
  WIREEND('SH', SORTIE);
  PIN(TDK; 'SB', SORTIE);
  NAME 'VDD1D' AS 'VDD3D';
  NAME 'VSS2D' AS 'VSS5D';
  NAME 'VSSD'  AS 'VSS2D';
  NAME 'VDD2D' AS 'VDD4D';
  PORTE_3E:= WHATWITH;
ENDWITH;
WITH REGISTRE DO;
  WIRE (-3*PAS, HEIGHT(REGISTRE)+PAS; E, LENGTH(REGISTRE)+3*PAS, ALU);
  WIREEND('VSS_BIS', VSS);
  WIRE ('E1'  ; N, 2*PAS, ALU, POLY; N, 0, ALU);
  WIRE ('VDD1'; O, 3*PAS, ALU);
  WIRE ('H1'  ; O, 3*PAS, ALU);
  WIRE ('VSS1'; O, 3*PAS, ALU);
  WIRE ('NH1' ; O, 3*PAS, ALU);
  WIRE ('VDD3'; O, 3*PAS, ALU);
  NAME 'VDD'  AS 'VDD1D';
  NAME 'VSS'  AS 'VSS1D';
  NAME 'VDD2' AS 'VDD2D';
  REGISTRE:= WHATWITH;
ENDWITH;
WITH AMPLI DO;
  WIRE ('VDDG'; O, 3*PAS, ALU);
  WIRE ('VSSG'; O, 3*PAS, ALU);
  WIRE ('EG'  ; O, 2*PAS, ALU; O, 0, POLY); TDK:= WIREEDGE;
  WIREEND('EH', ENTREE);
  PIN(TDK; 'EB', ENTREE);
  NAME 'VSSD' AS 'VSS0D';
  NAME 'VDDD' AS 'VDD0D';
  AMPLI:= WHATWITH;
ENDWITH;
TAMPON:= PORTE_3E V(2*PAS) REGISTRE V(2*PAS) AMPLI;
TAMPON:= OVERSIZE(TAMPON, {0, 0, 0, 3*PAS});
TAMPON:= SPIDER(TAMPON, {'EH', 'SH'}, N);
TAMPON:= SPIDER(TAMPON, {'EB', 'SB'}, S);
WITH TAMPON, ALL DO
  NAME 'EH' AS 'EH1';
  NAME 'EB' AS 'EB1';
  NAME 'SH' AS 'SH1';
  NAME 'SB' AS 'SB1';
  PIN(POS_VSS5G[0]+3*PAS, POS_VSS5G[1]; 'VSS5G', ACCES, ALU);
  BUFFER:= WHATWITH;
ENDWITH;
```

```
LOOP N:= 2 UPTO NBIT;
  WITH BUFFER, ALL DO;
    DISCARD 'VSS0D', 'VSS_BIS', 'VSS1D', 'VSS_TER', 'VSS2D', 'VSS5D',
            'VDD0D', 'VDD1D', 'VDD2D', 'VDD3D', 'VDD4D';
    BUFFER:= WHATWITH;
  ENDWITH;
  WITH TAMPON, ALL DO;
    NAME 'EH' AS 'EH'&N;
    NAME 'SH' AS 'SH'&N;
    NAME 'EB' AS 'EB'&N;
    NAME 'SB' AS 'SB'&N;
    TRANCHE_DE_BIT:= WHATWITH;
  ENDWITH;
  BUFFER:= BUFFER HE TRANCHE_DE_BIT;
ENDLOOP;

     \* ASSEMBLAGE DU BUFFER PROPREMENT DIT *\

BUFFER:= AMPHORN HE BUFFER;
TAB_ROUT:= { {{'VSS0D'},{'VSS0'}},
             {{'VDD0D'},{'VDD0'}},
             {{'VSS1D'},{'VSS_TE'}},
             {{'VDD1D'},{'VDD1'}},
             {{'VSS2D'},{'VSS1'}},
             {{'VDD2D'},{'VDD2'}},
             {{'VSS3D'},{'VSS_BI'}},
             {{'VDD3D'},{'VDD3'}},
             {{'VSS4D'},{'VSS2'}},
             {{'VDD4D'},{'VDD4'}} };

BUFFER:= FLAG HR(TAB_ROUT, PAS, ALU, ALU) BUFFER;
BUFFER:= SPIDER(BUFFER,
   {'VSS0G', 'VDD0G', 'VSS1G', 'VDD1G', 'VSS2G', 'VDD2G',
    'VSS3G', 'VDD3G', 'VSS4G', 'VDD4G', 'VSS5G'}, O);
BUFFER:= SPIDER (BUFFER,
  {'REMPLIR1', 'VIDER1', 'PLEIN1', 'VIDE1', 'LECT1', 'ECR1'}, N);
BUFFER:= SPIDER (BUFFER,
  {'REMPLIR2', 'VIDER2', 'PLEIN2', 'VIDE2', 'LECT2', 'ECR2'}, S);

     \* FIN DU MODULE *\

:= BUFFER;
ENDMODULE;
\LIB 'O BUFFER O E BUFFER BUFFER Q';
```

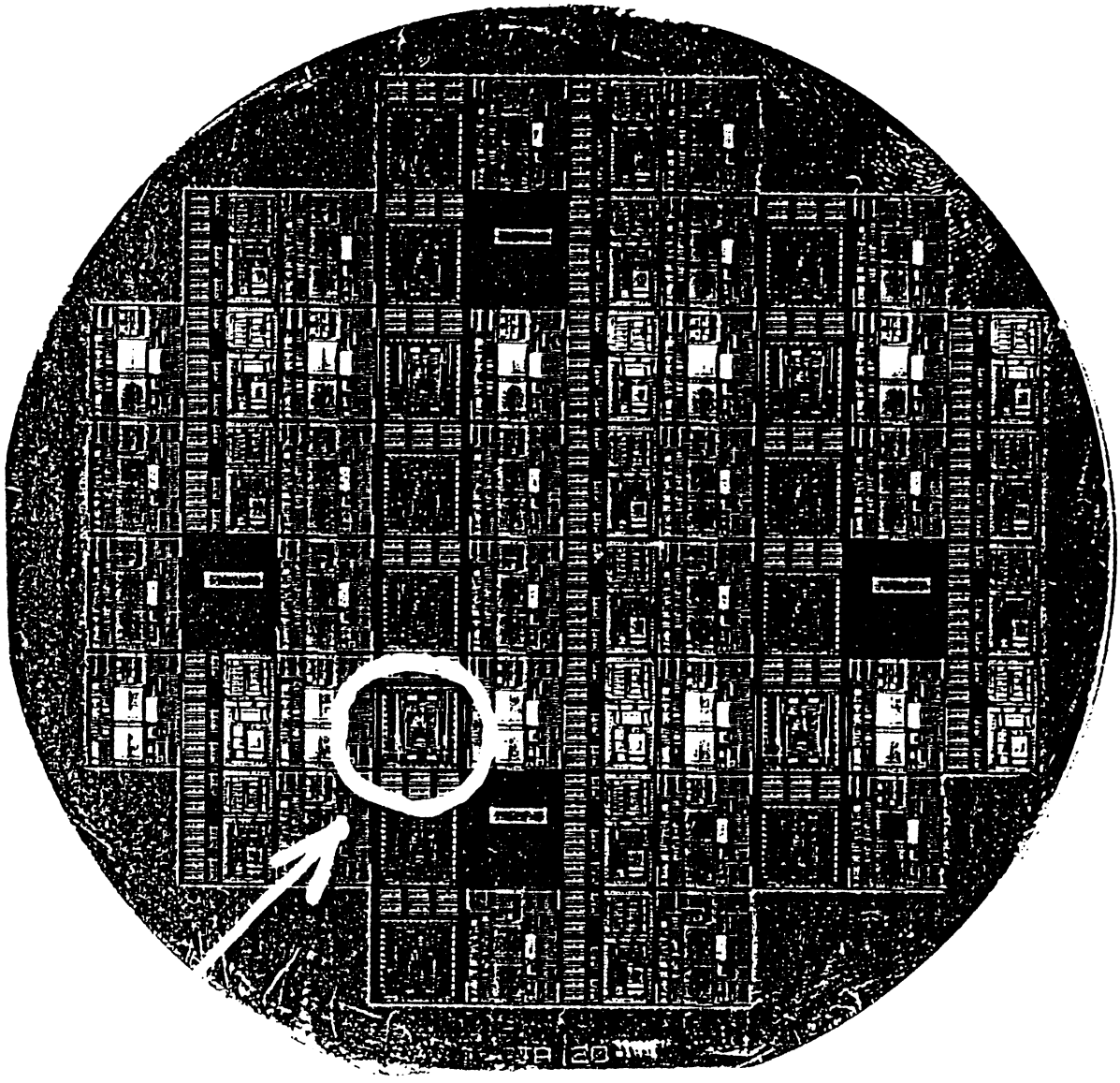
ANNEXE 3

PREMIERE VERSION DE LA CELLULE

Dessin des masques de la cellule.



Agrandissement de la tranche contenant la 1ère version de la cellule



ANNEXE 4

LA BIBLIOTHEQUE 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

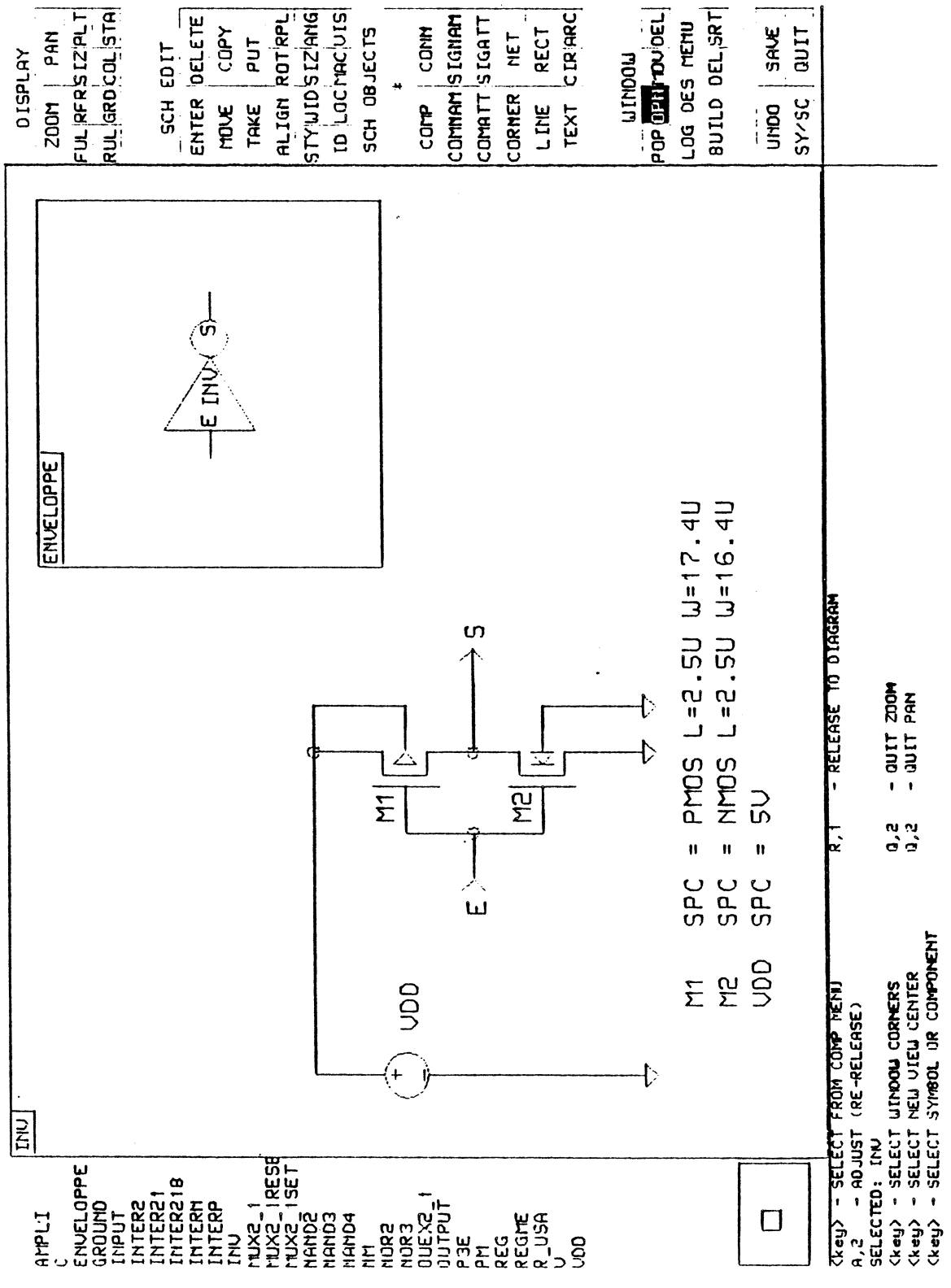
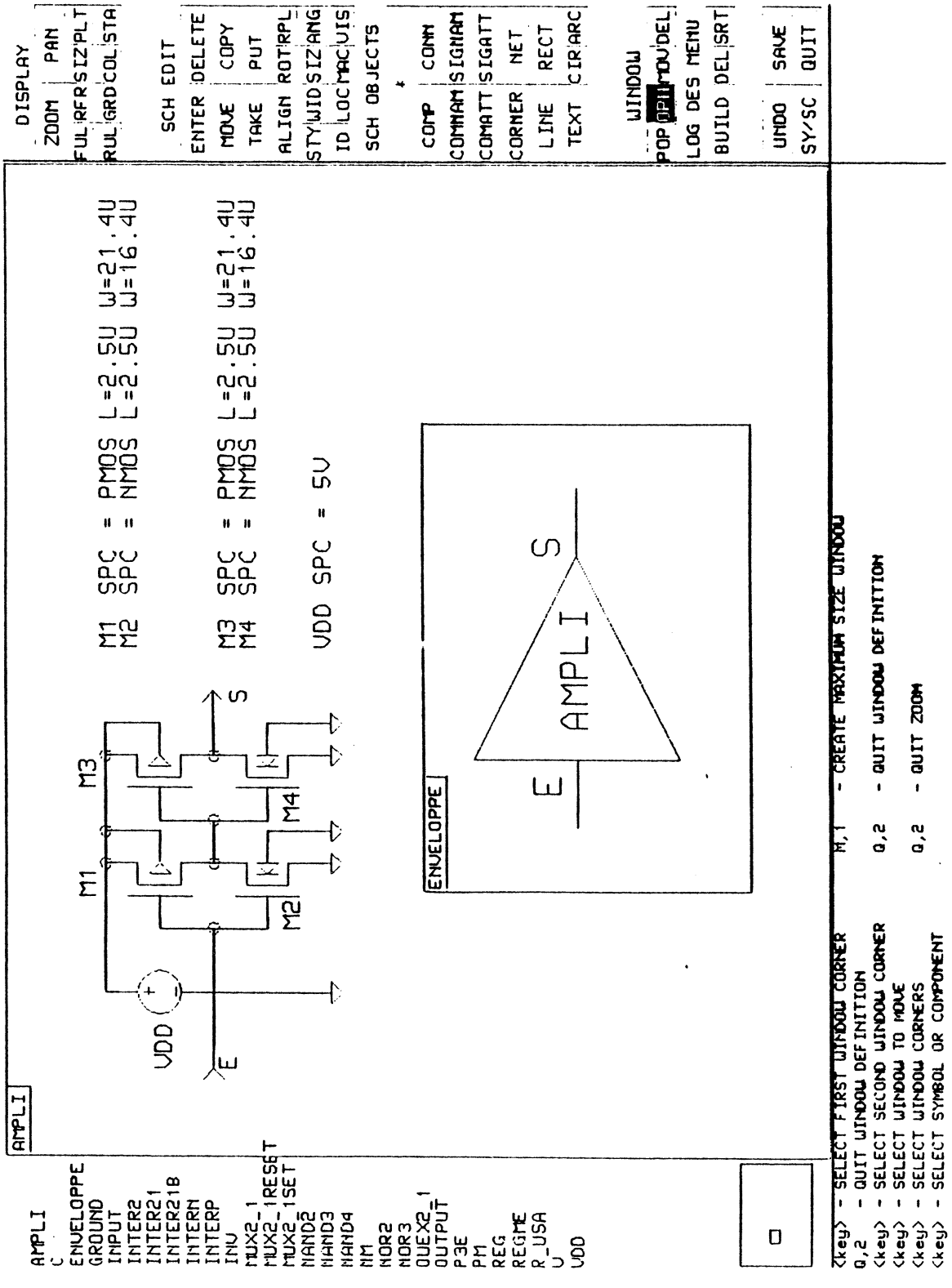
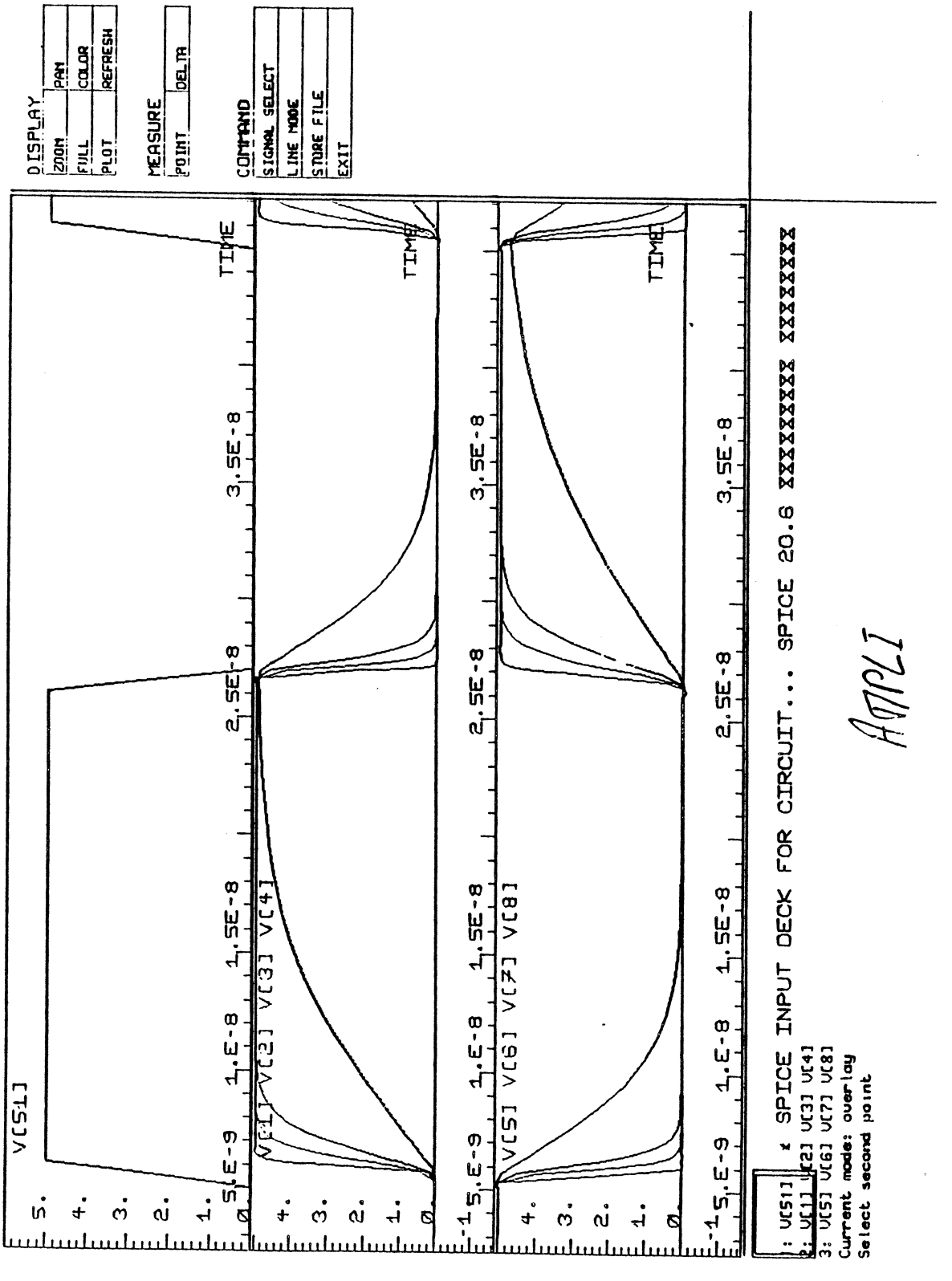


Schéma électrique de l'amplificateur



Simulation électrique de l'inverseur et de l'ampli



E

S-Ampli

S-Inv

AMPLI

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

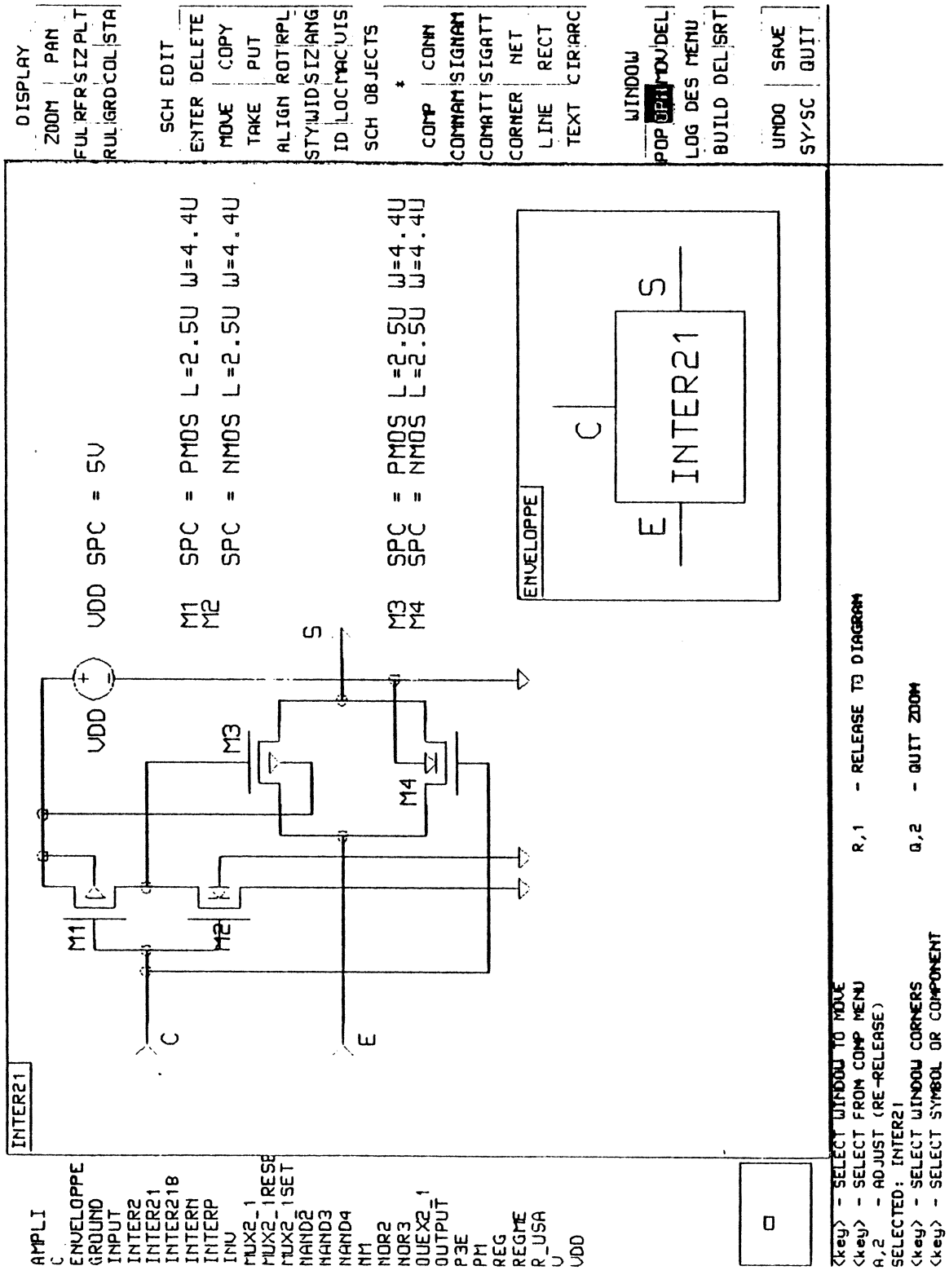


Schéma électrique d'un interrupteur - 1 commande (inversée) -

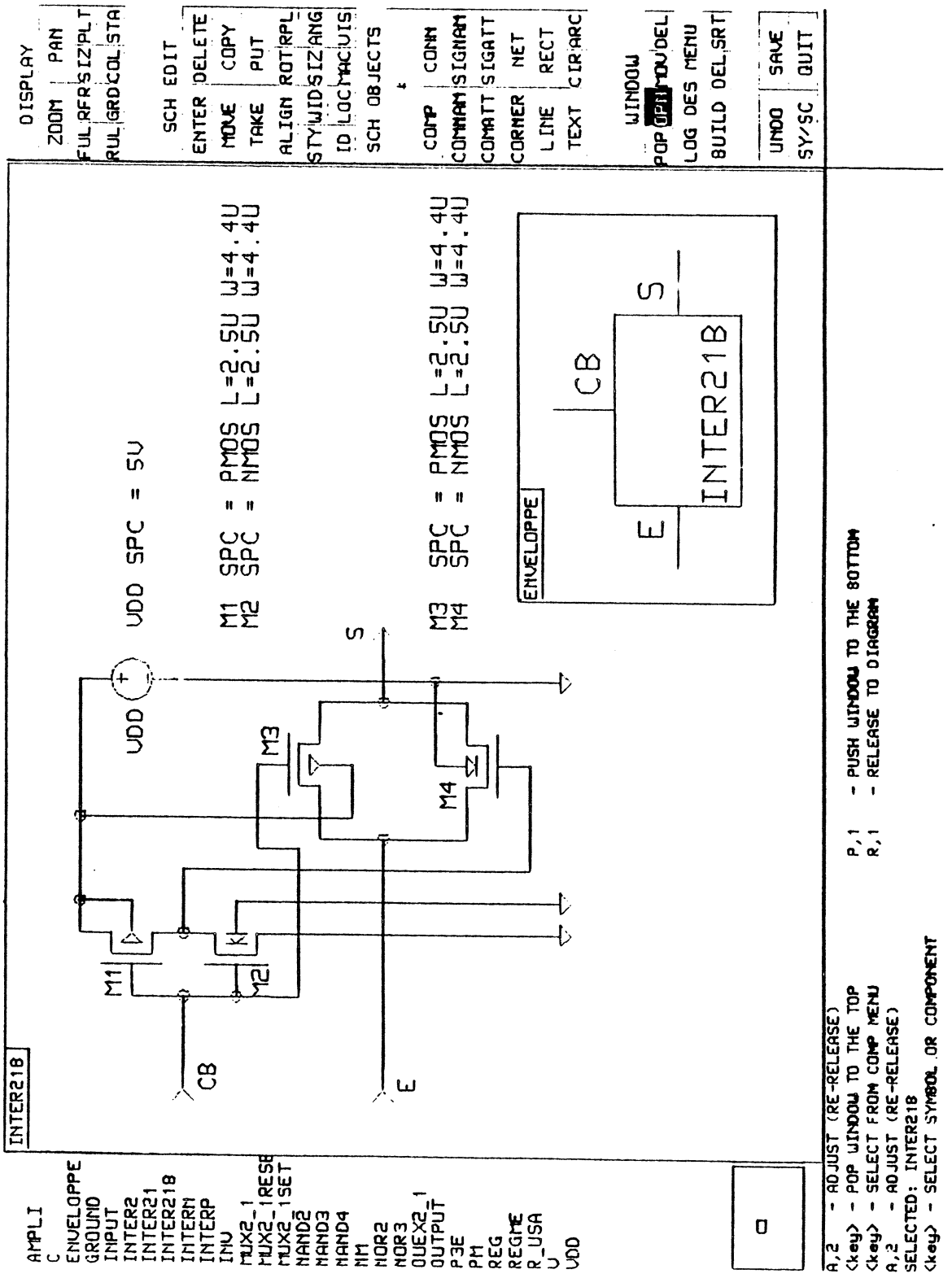


Schéma électrique d'un interrupteur - 2 commandes (complémentaires) -

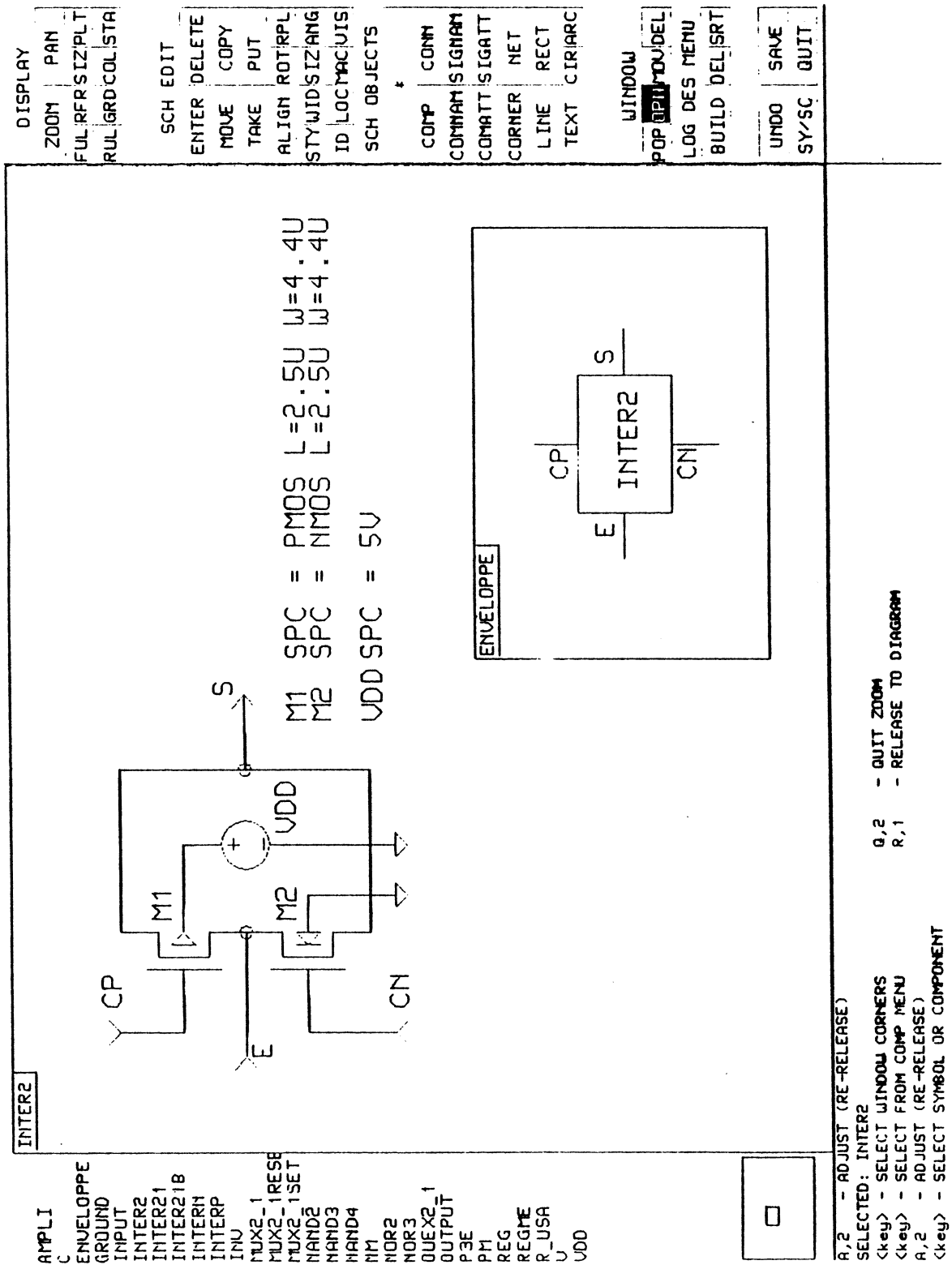
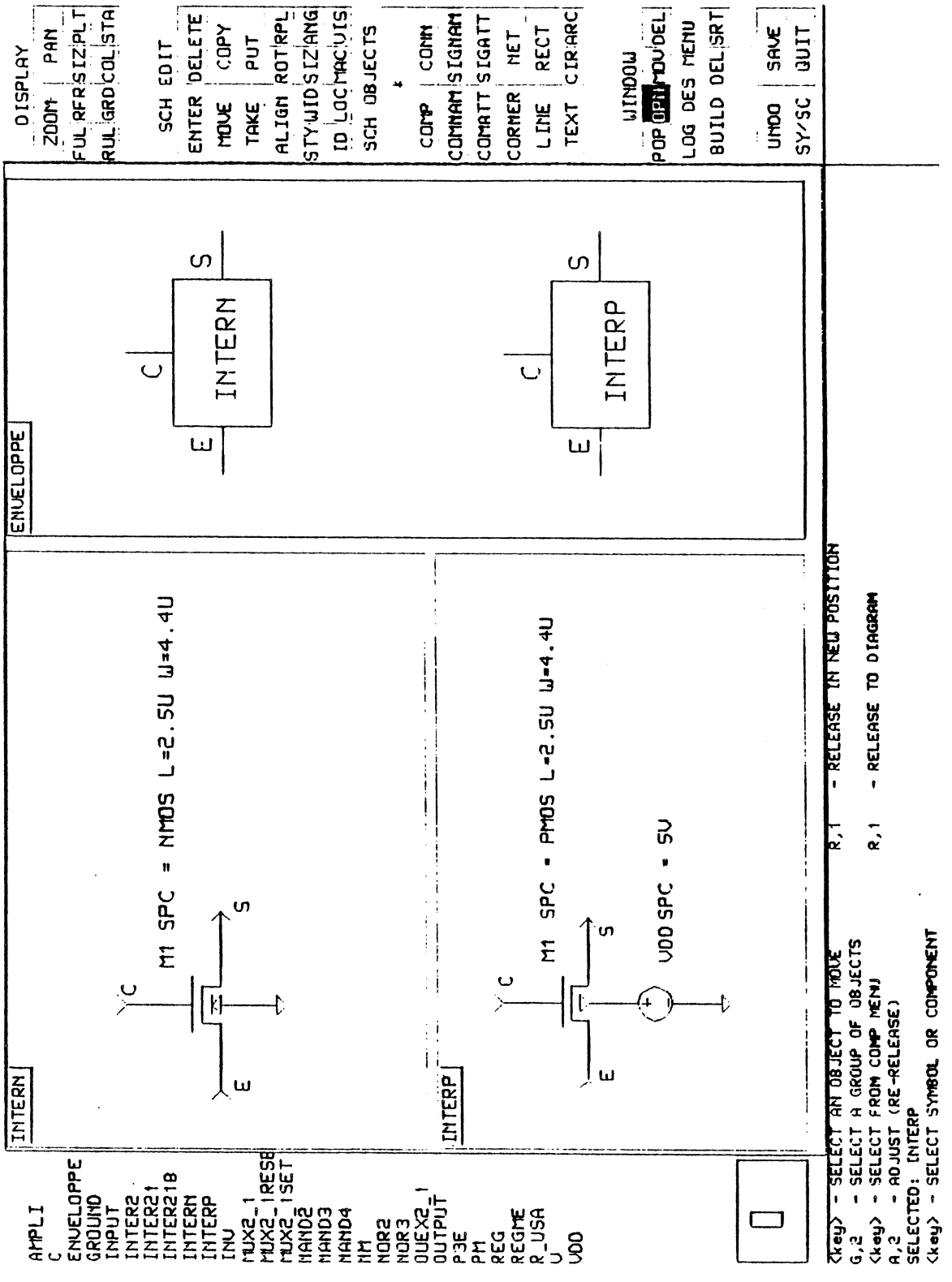


Schéma électrique d'interrupteurs simples



Simulation électrique des interrupteurs

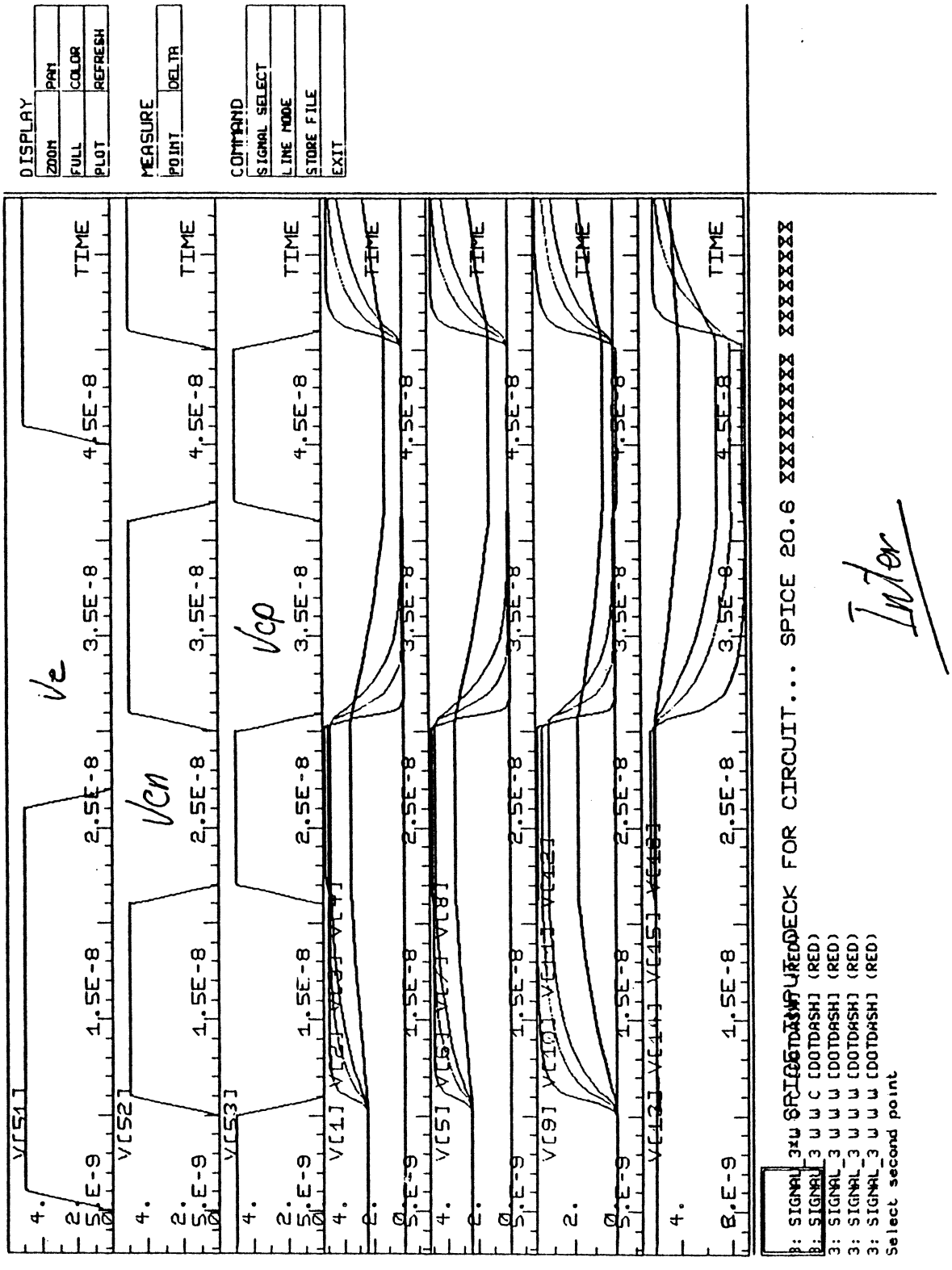


Schéma électrique du NAND 2 entrées

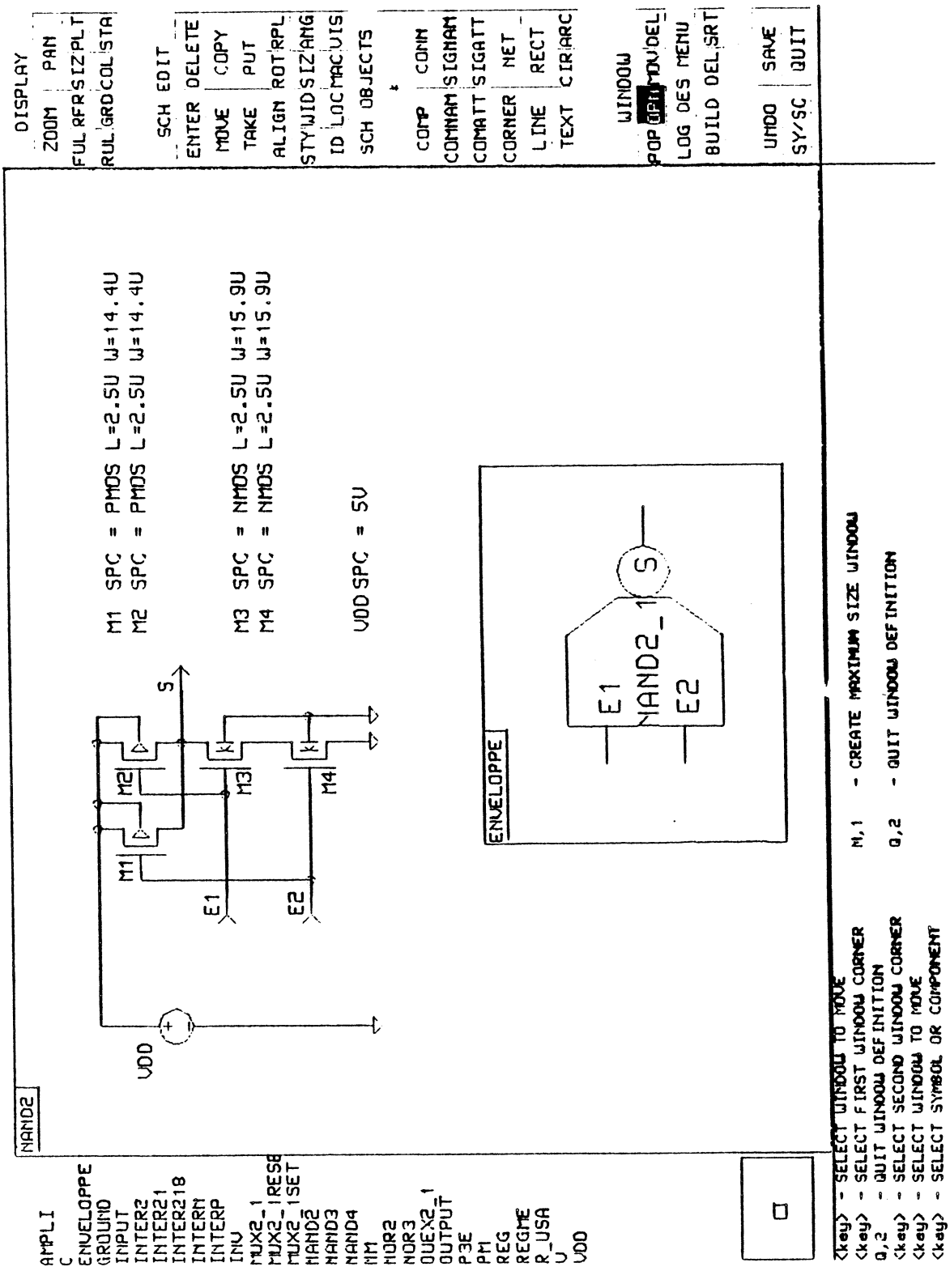


Schéma électrique du NAND 3 entrées

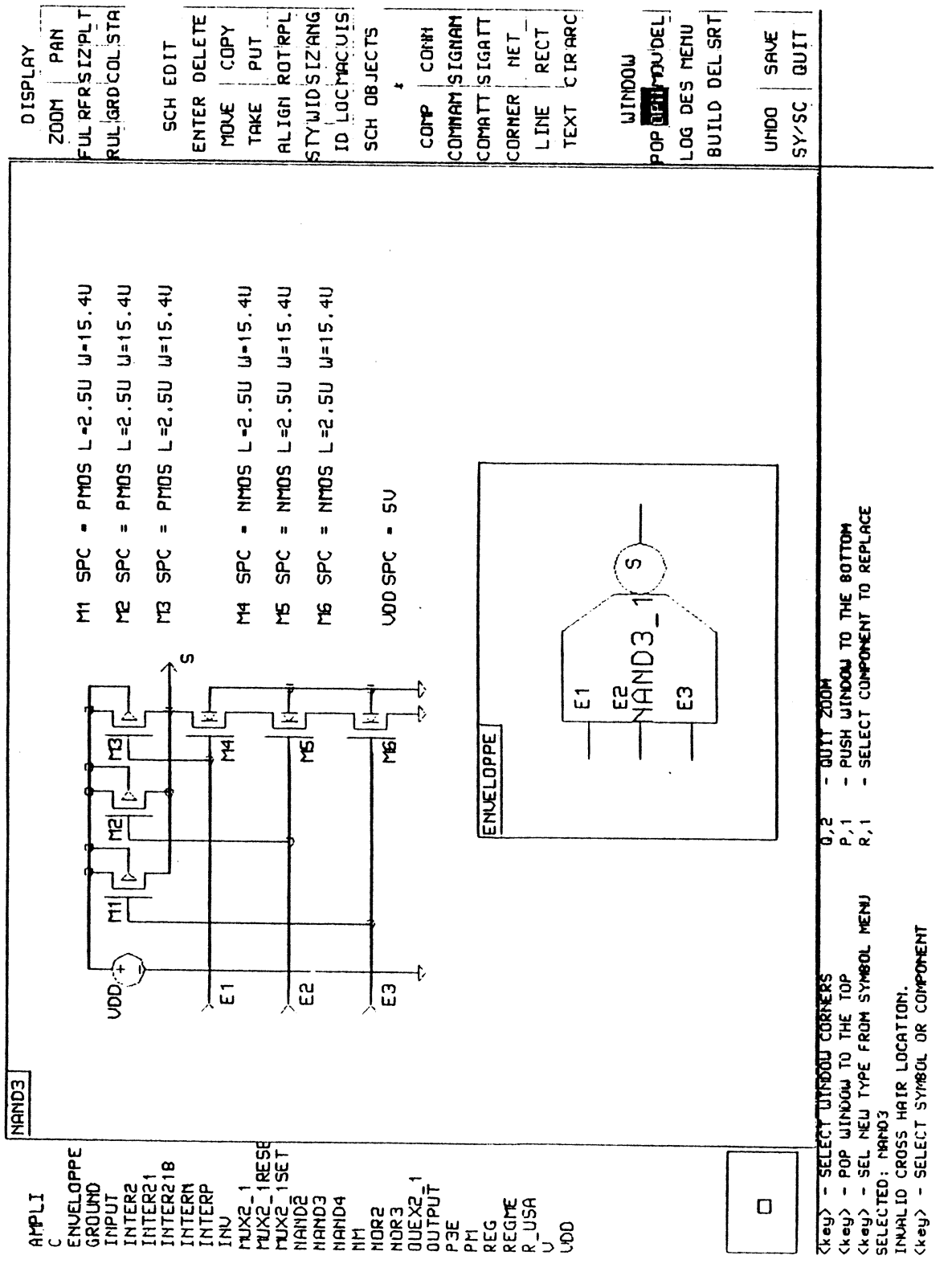
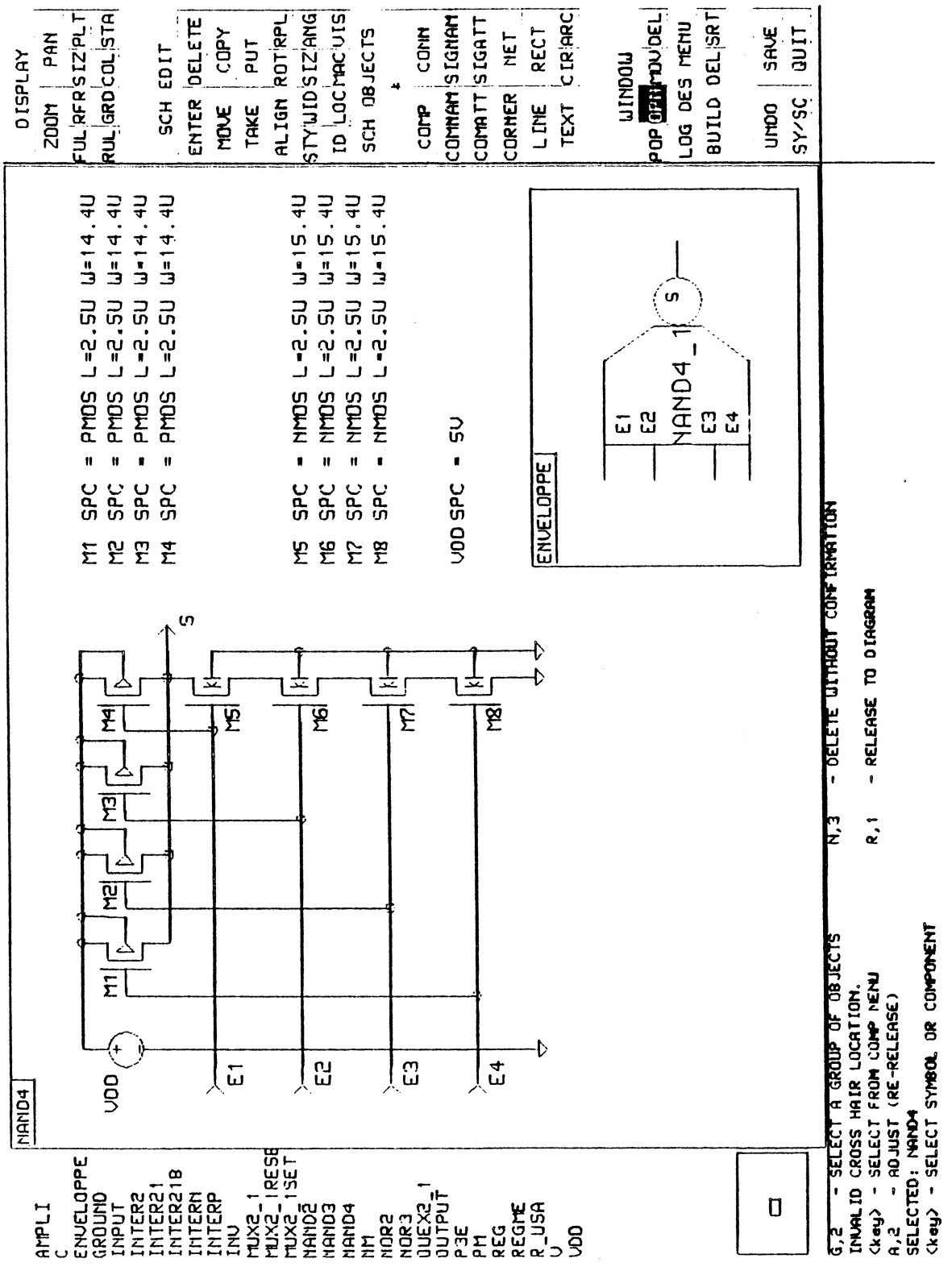
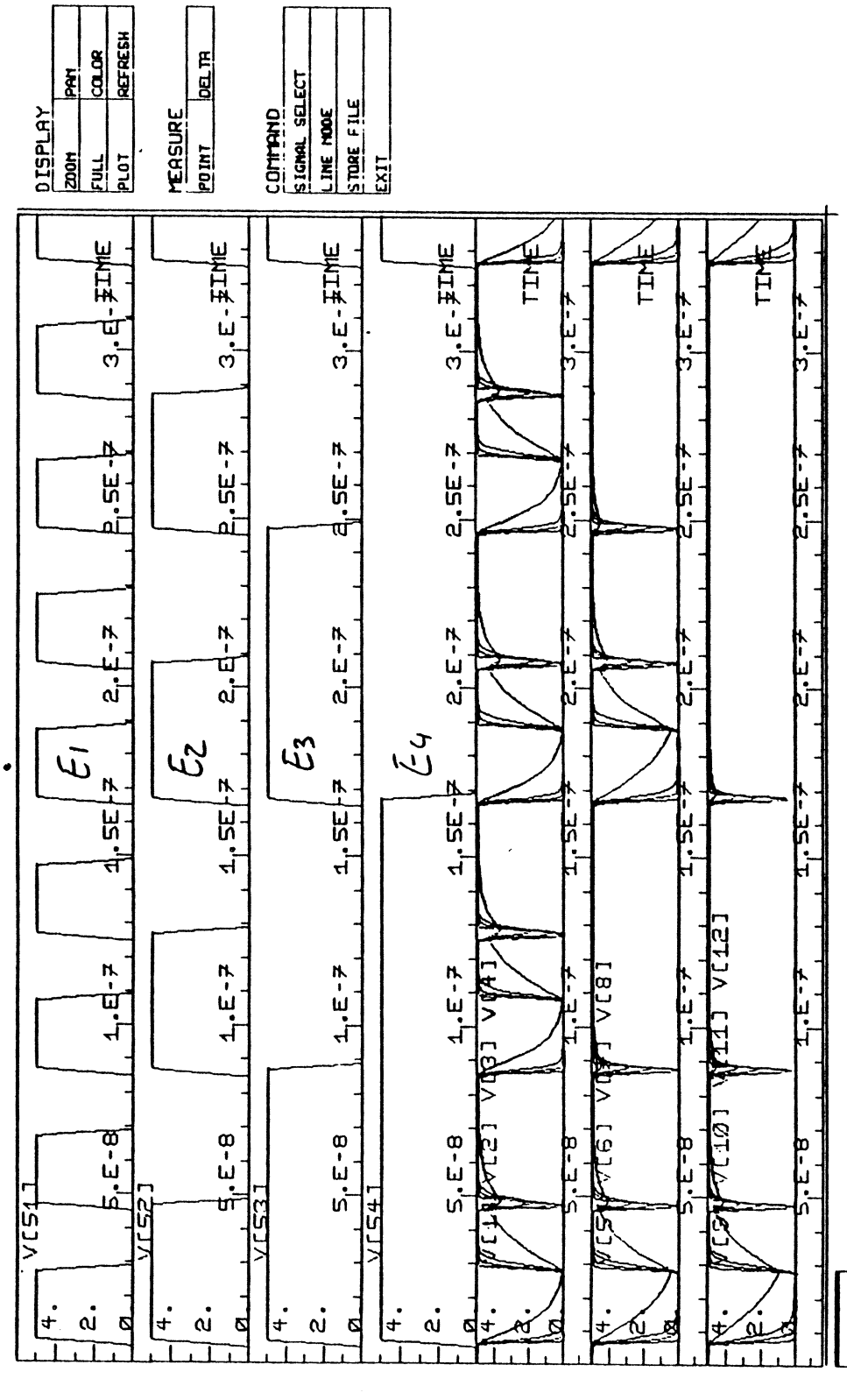


Schéma électrique du NAND 4 entrées



Simulation électrique des NANDs



Nand 2

Nand 3

Nand 4

Nand

Schéma électrique d'un multiplexeur (2→1)

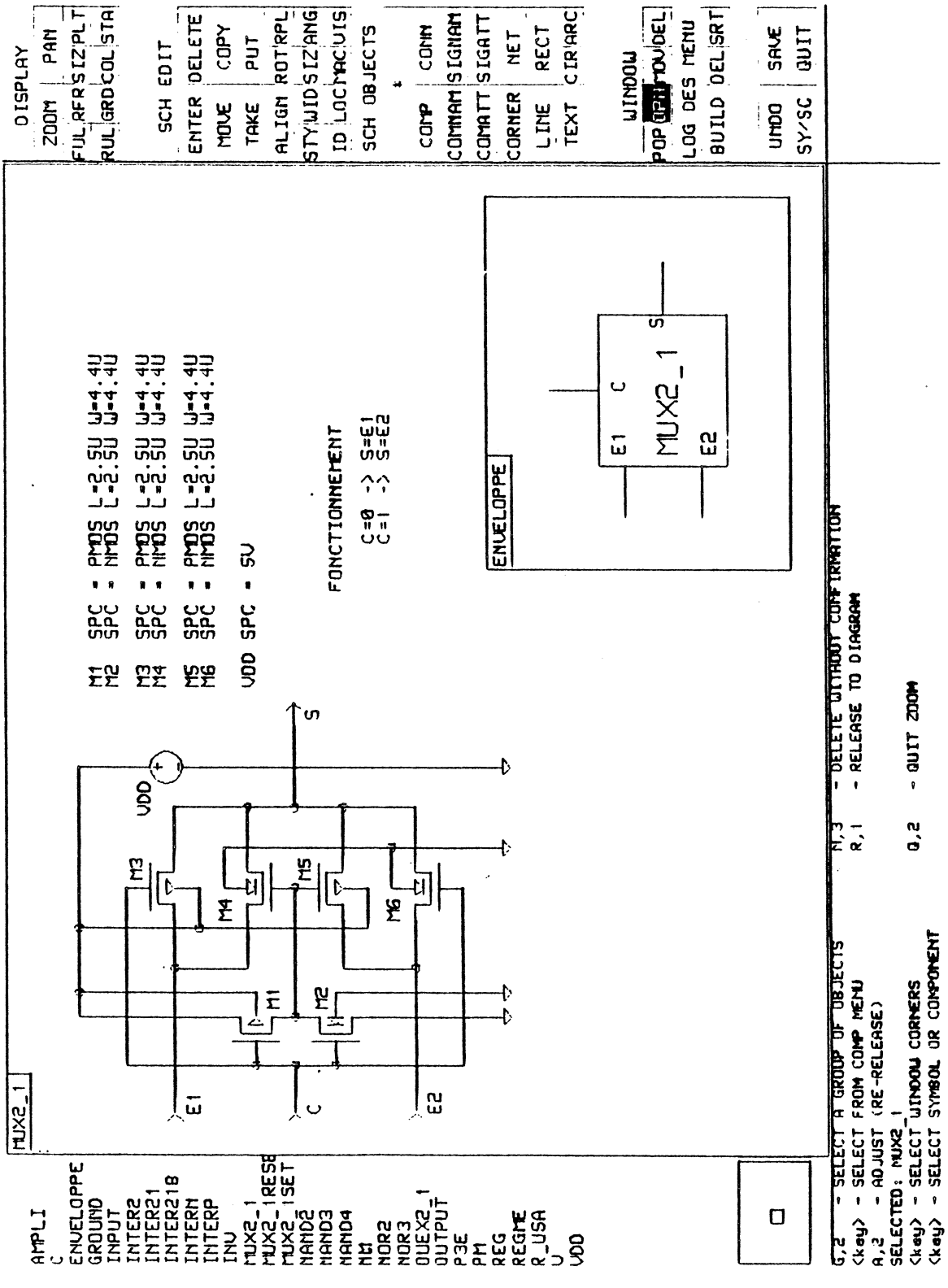


Schéma électrique d'un multiplexeur (2→1) avec mise à 1

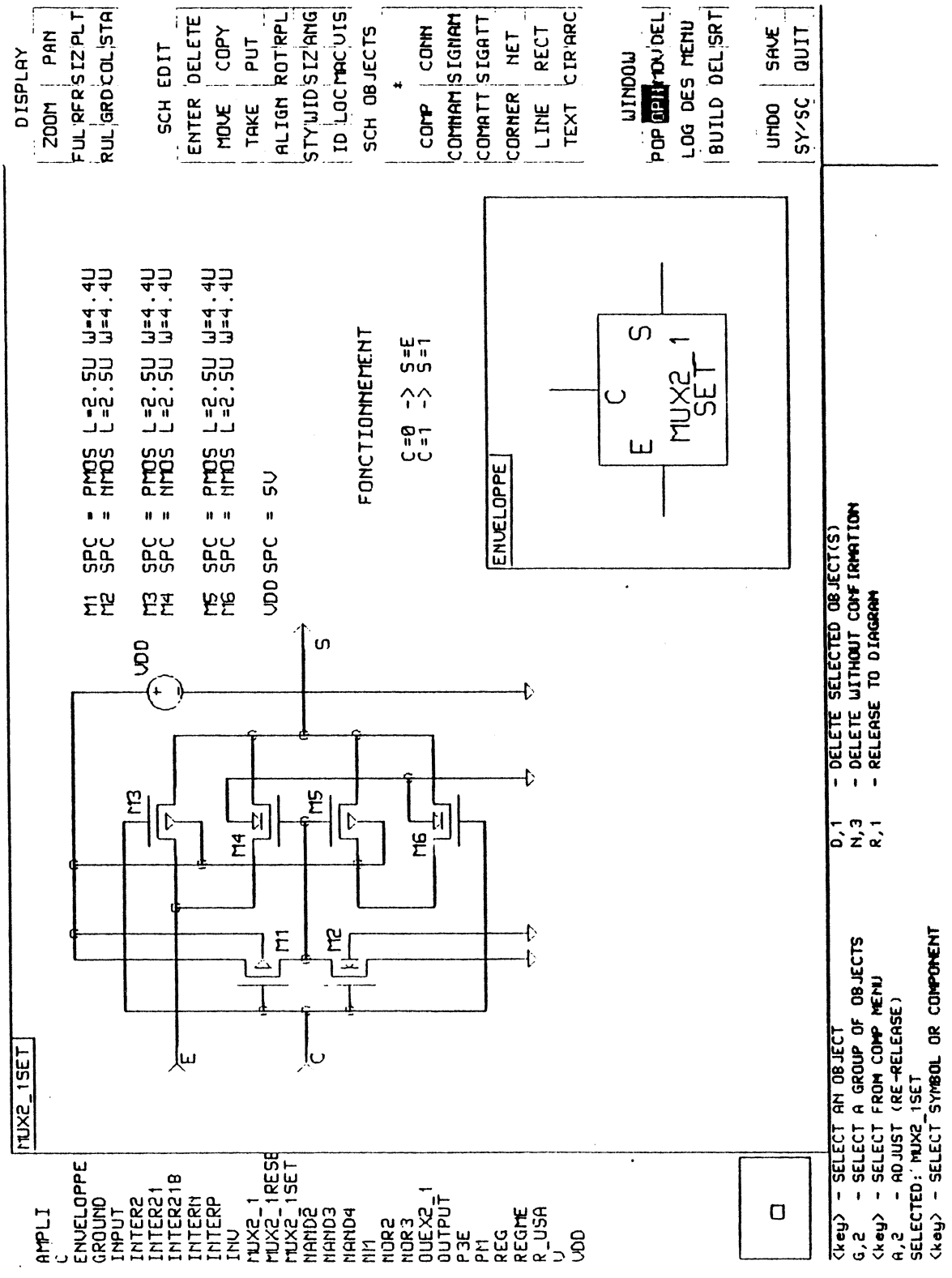
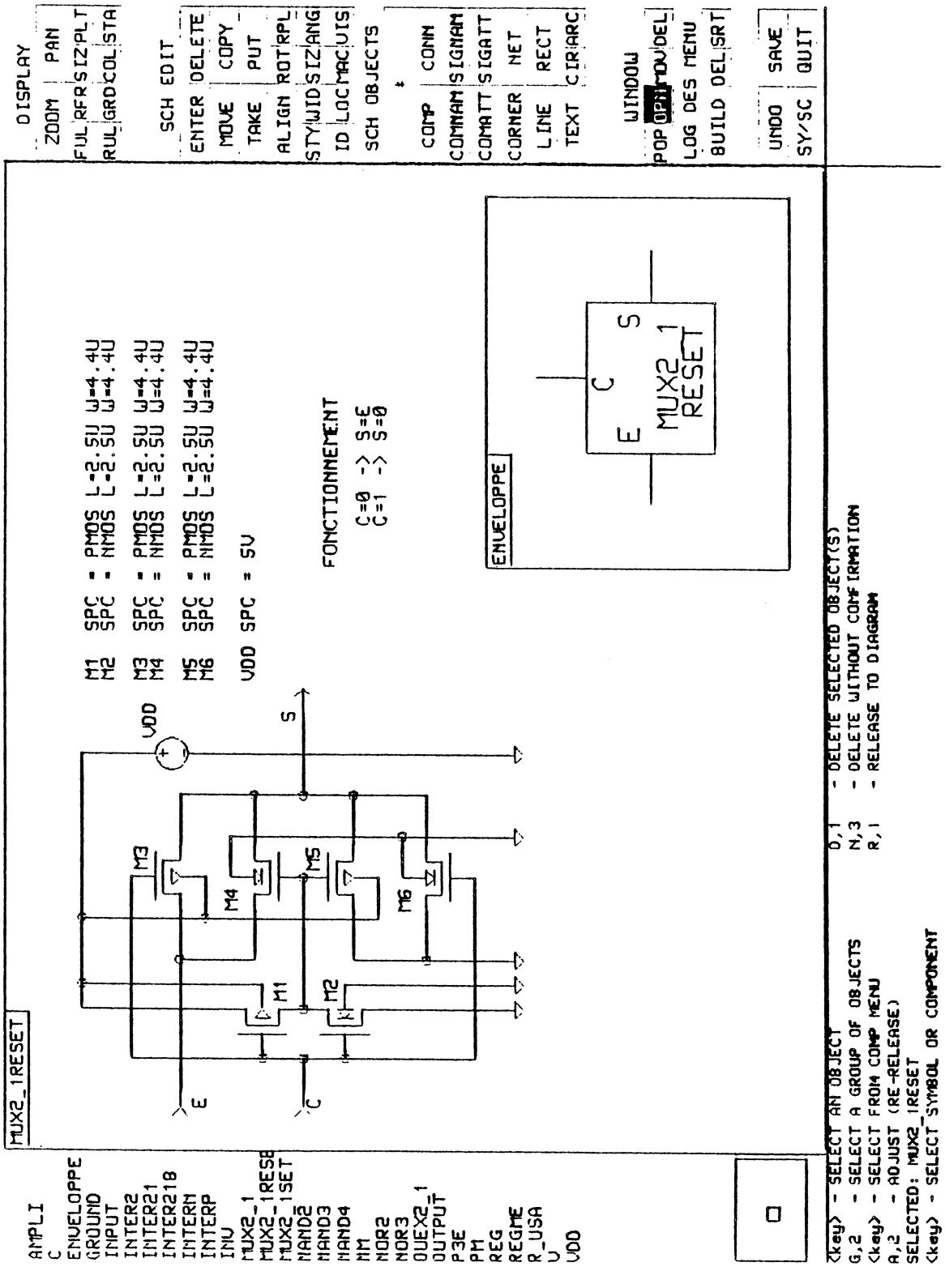


Schéma électrique d'un multiplexeur (2→1) avec mise à 0



Simulation électrique des multiplexeurs

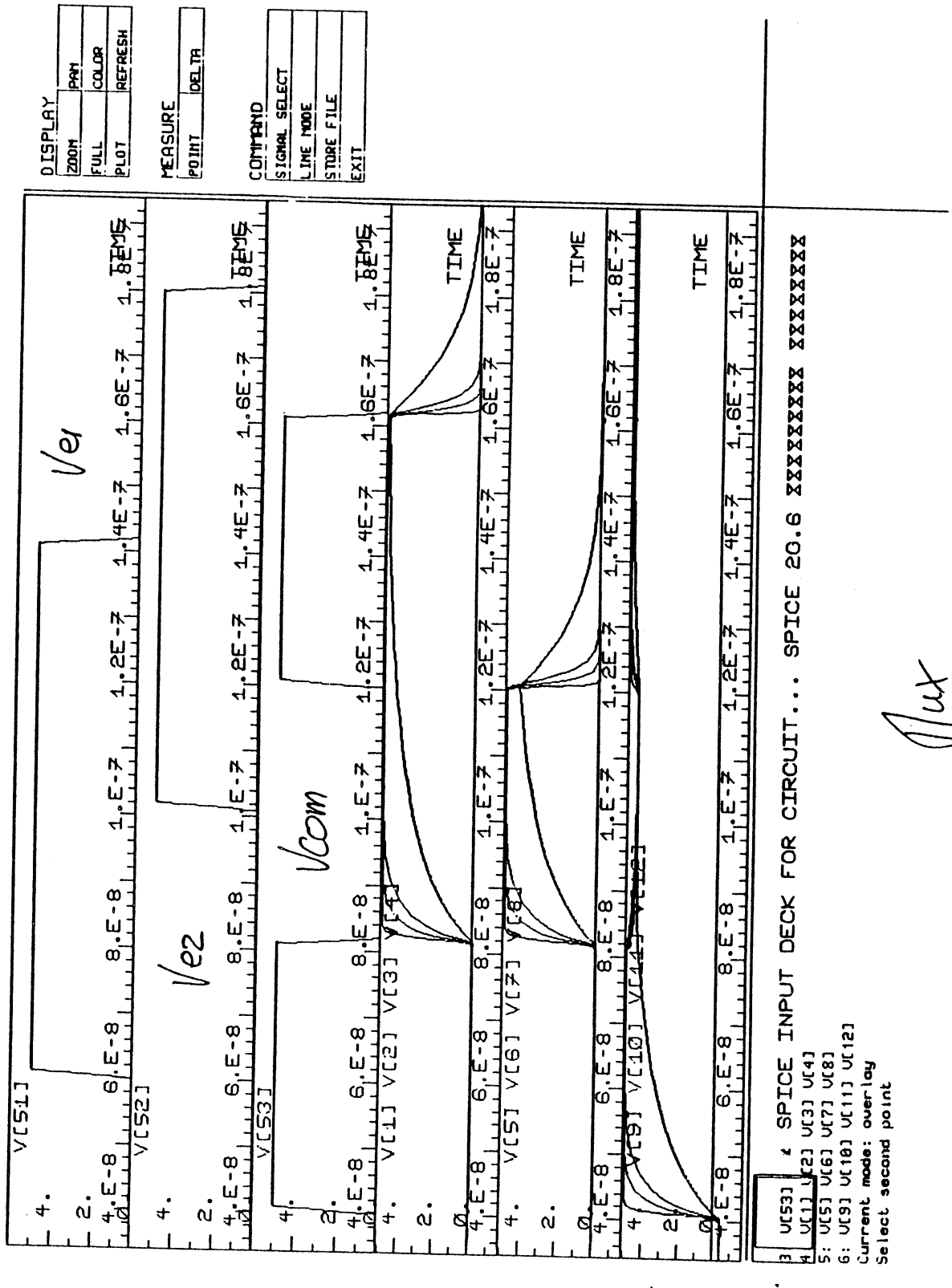


Schéma électrique du NOR 2 entrées

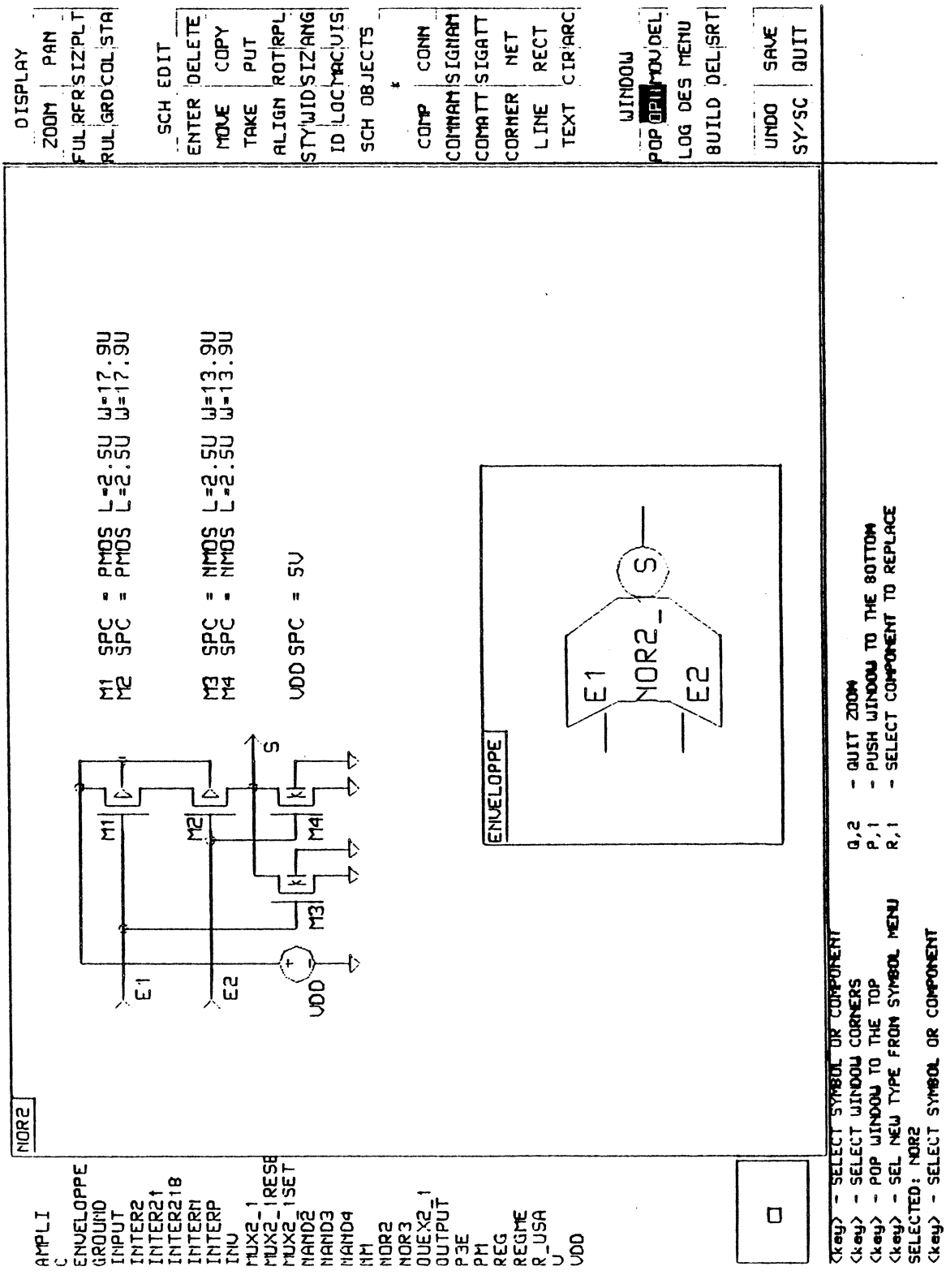
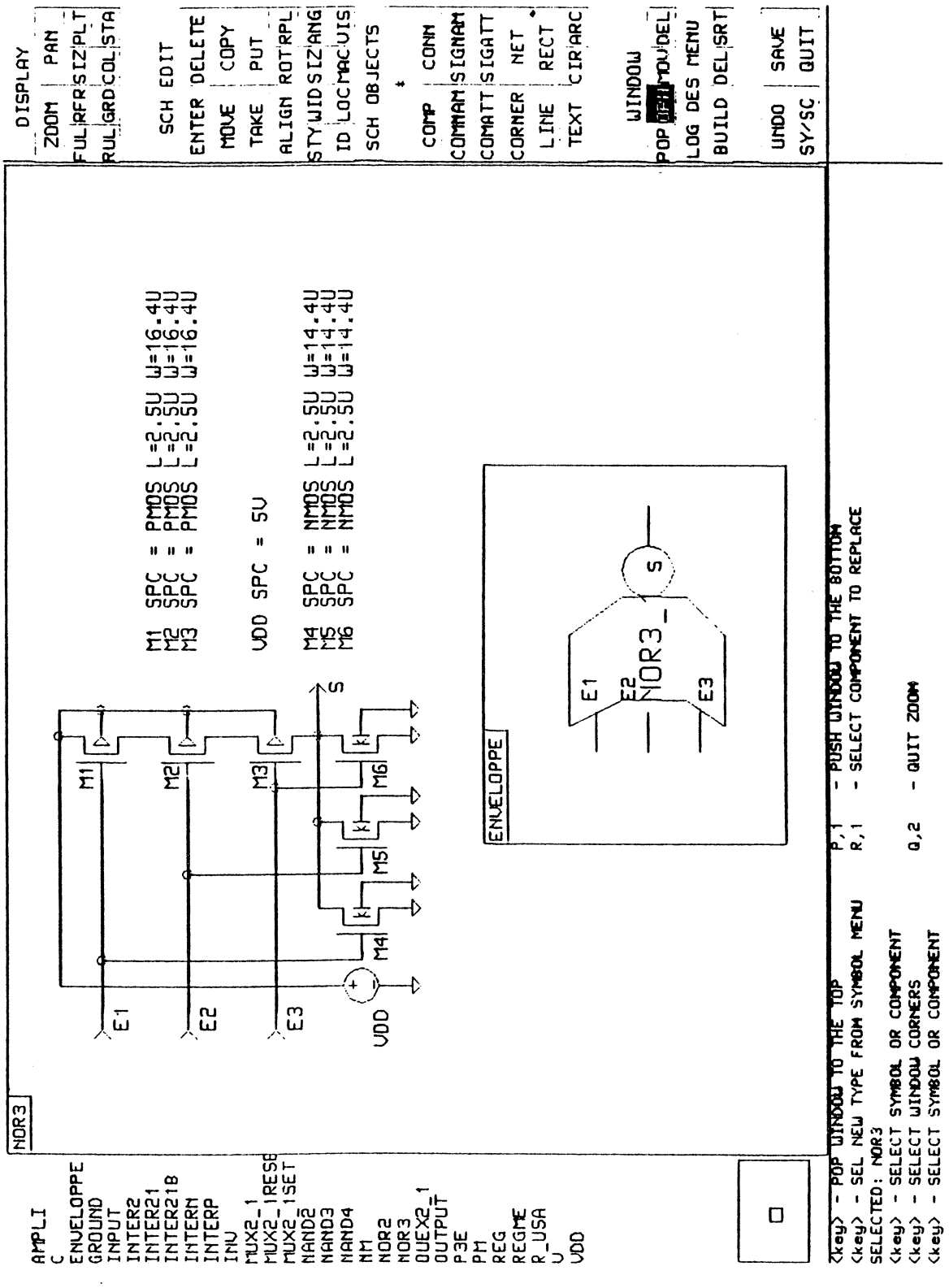
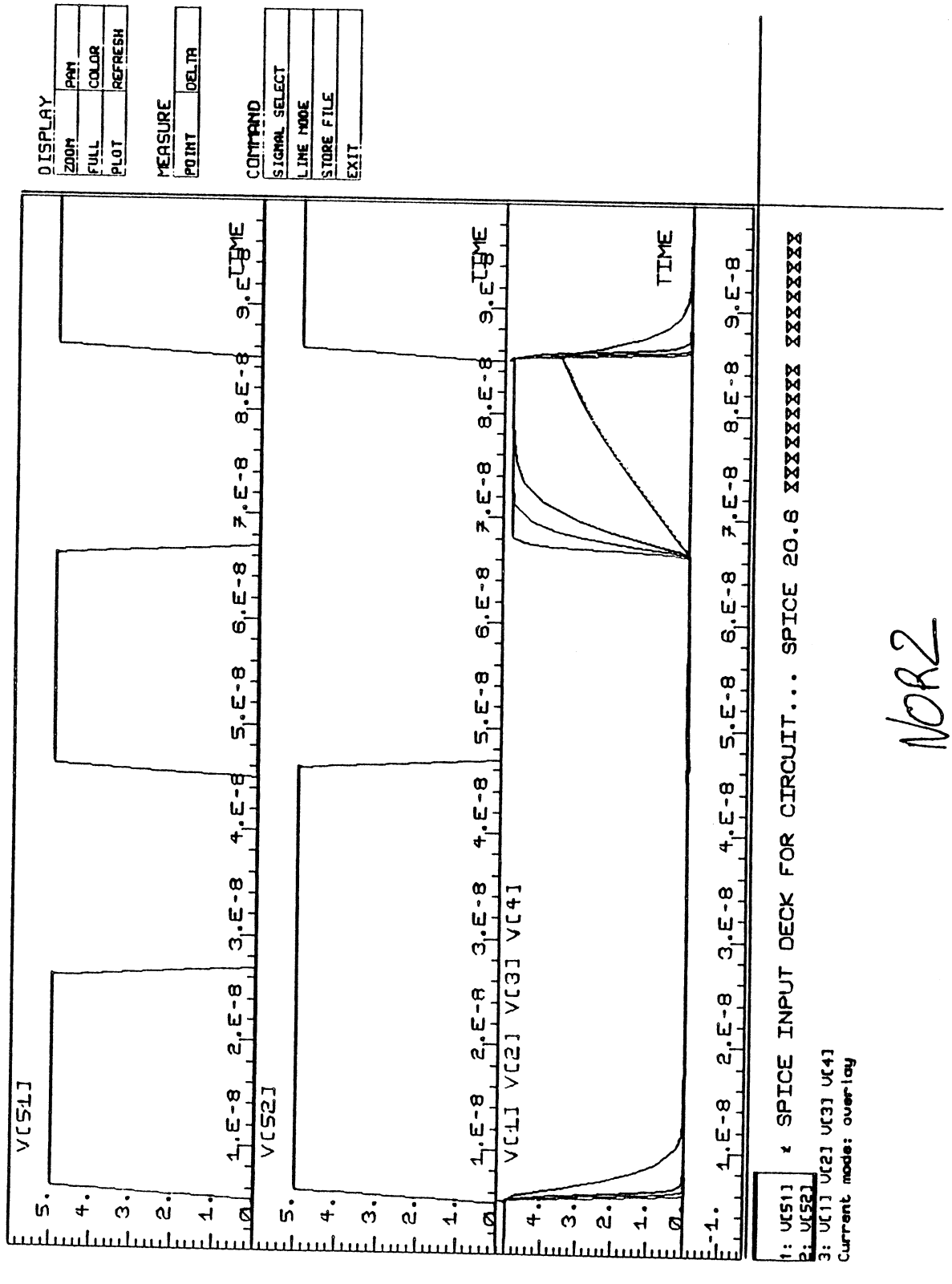


Schéma électrique du NOR 3 entrées



Simulation électrique des NORs



E1

E2

$\overline{E_1 + E_2}$

NOR2

Schéma électrique du registre simple (latch)

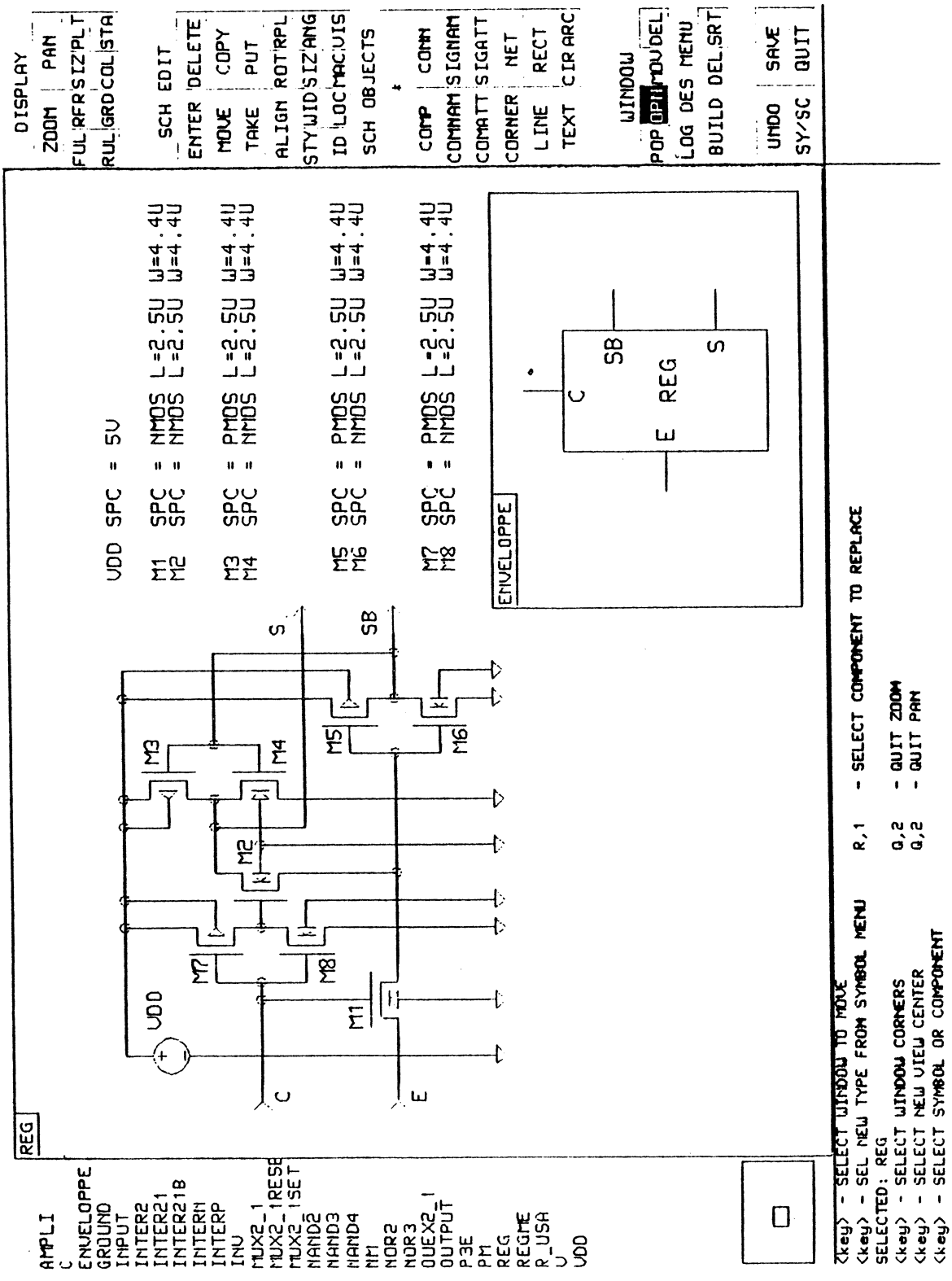
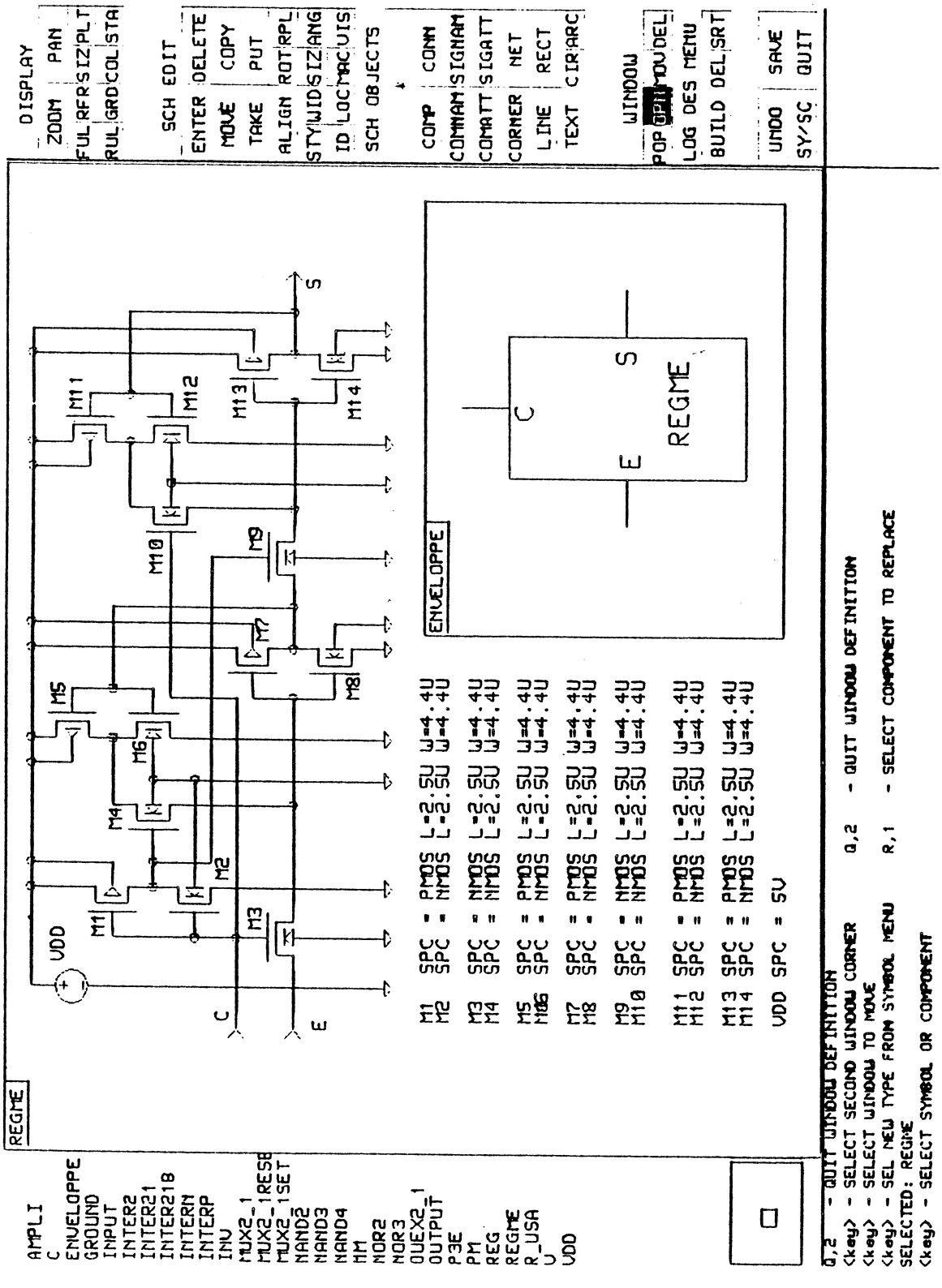


Schéma électrique du registre maître/esclave



Simulation électrique des registres

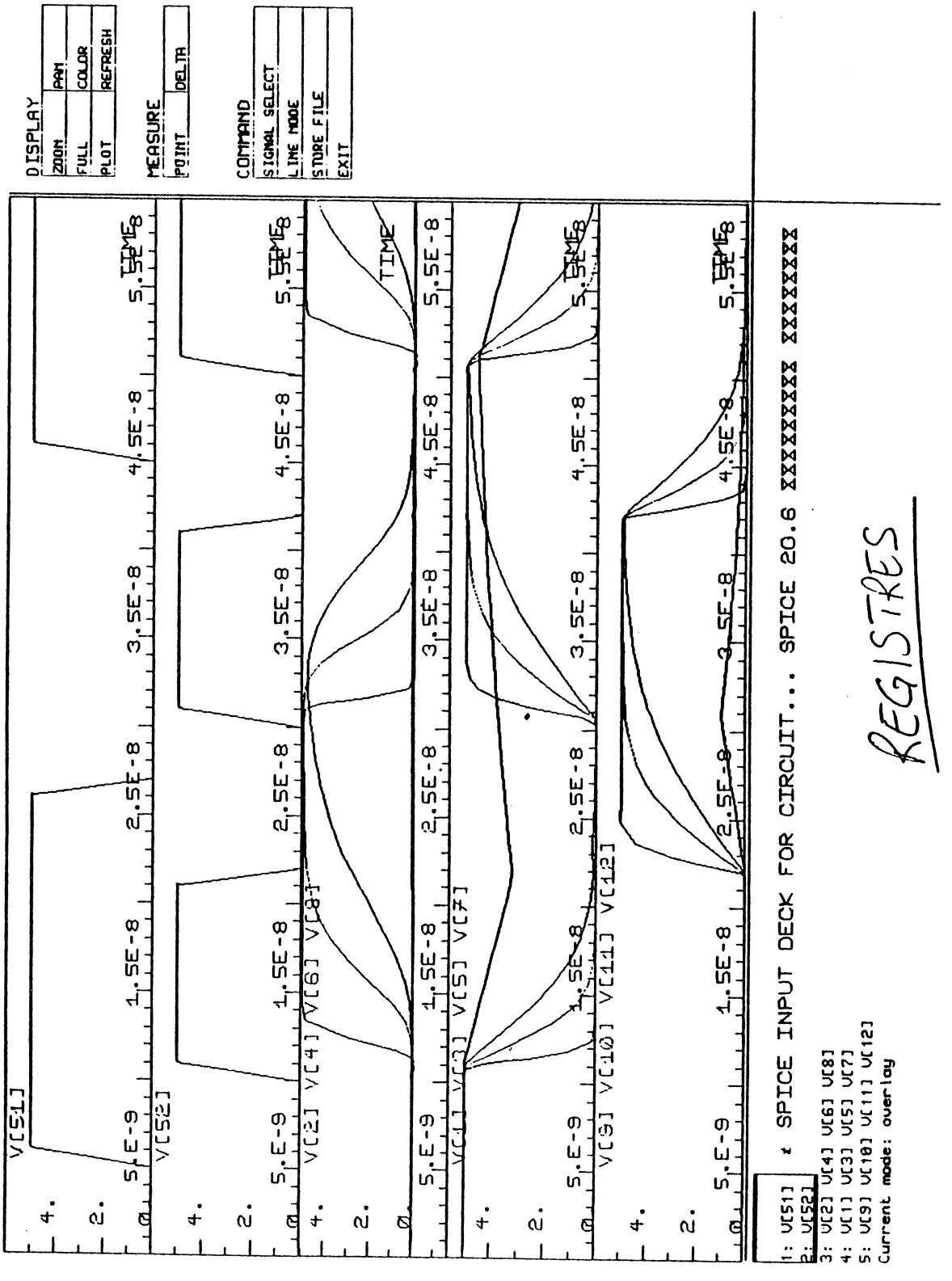
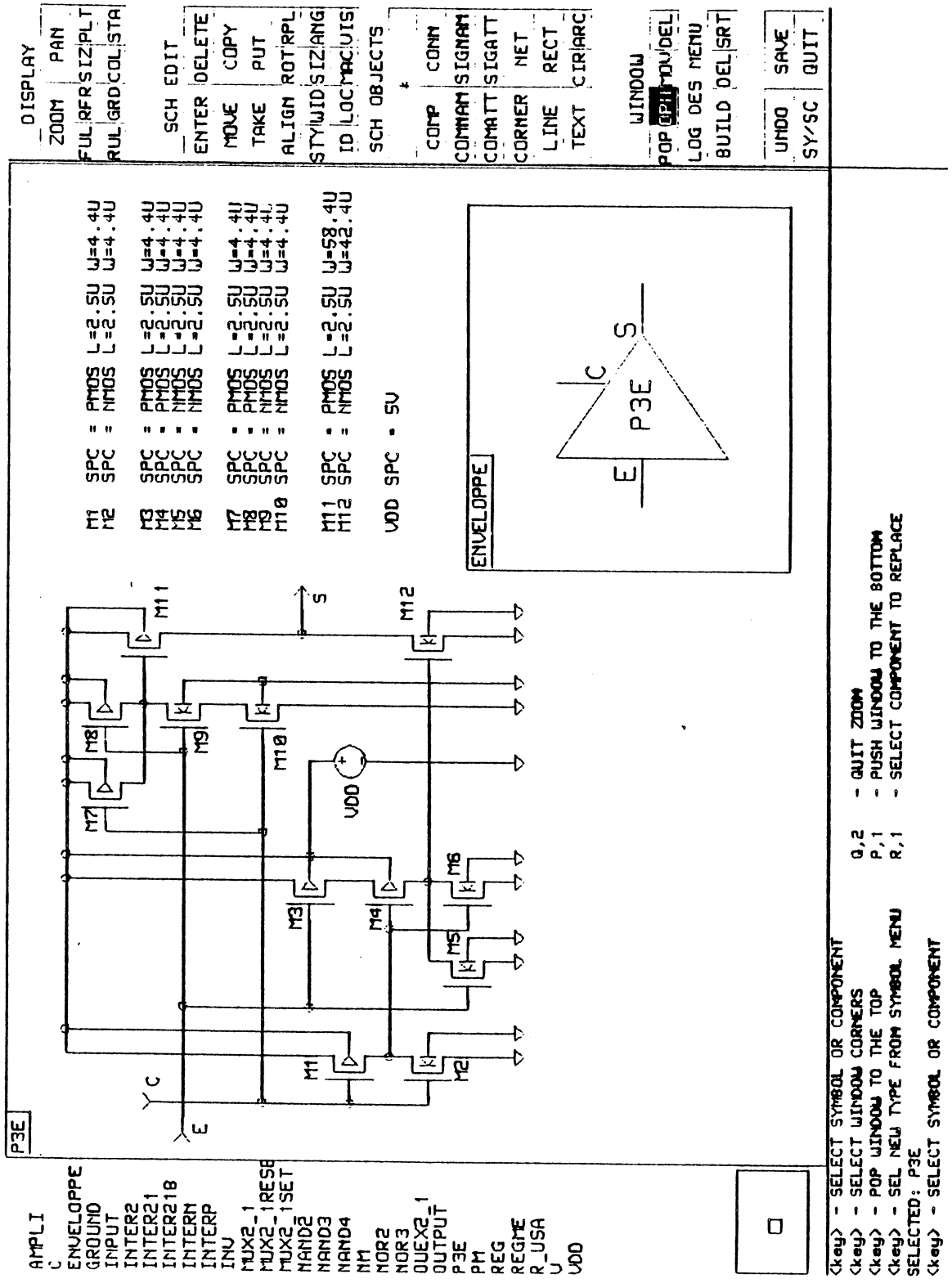
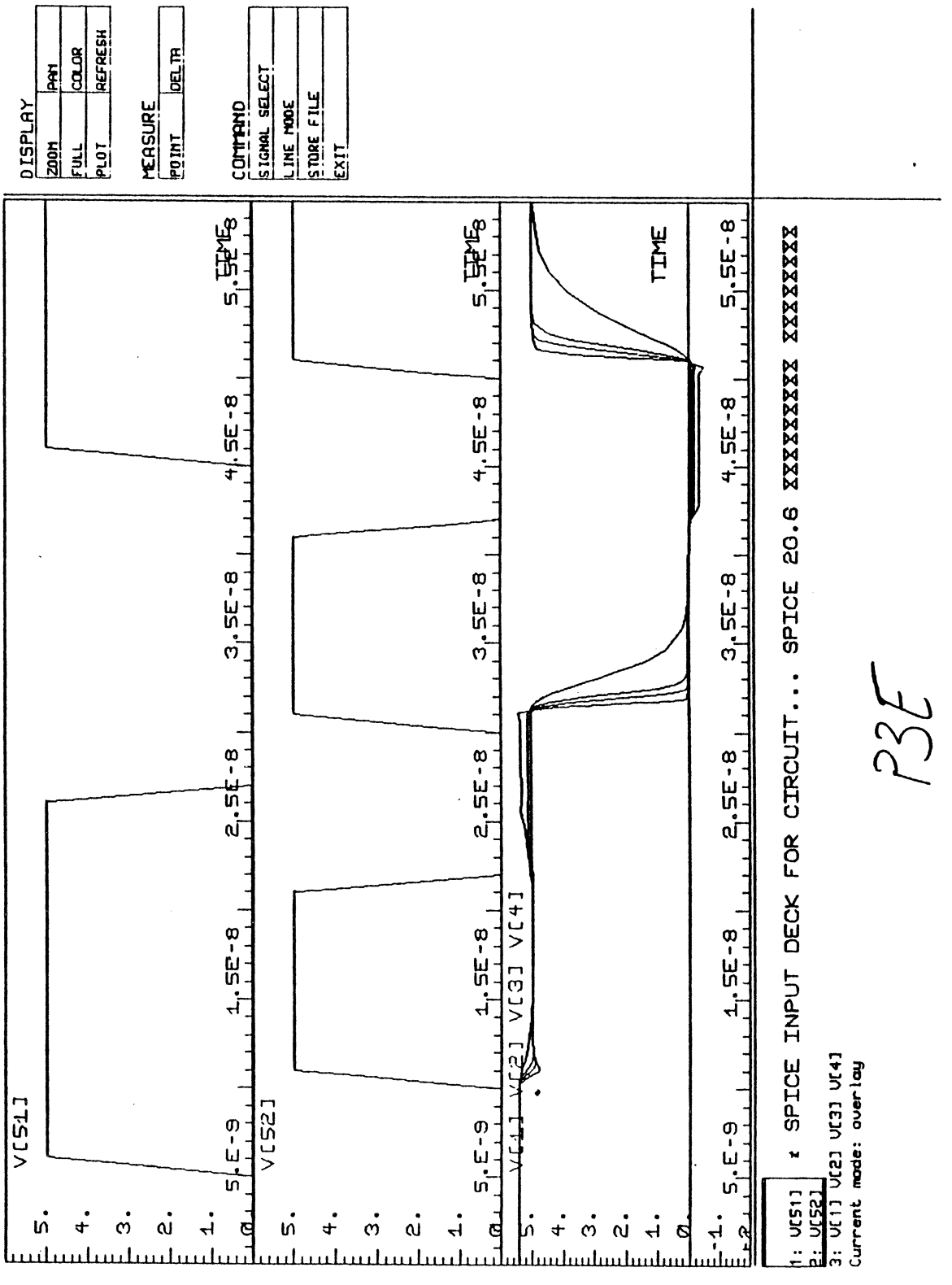


Schéma électrique de la porte-3-états



Simulation électrique de la porte-3-états



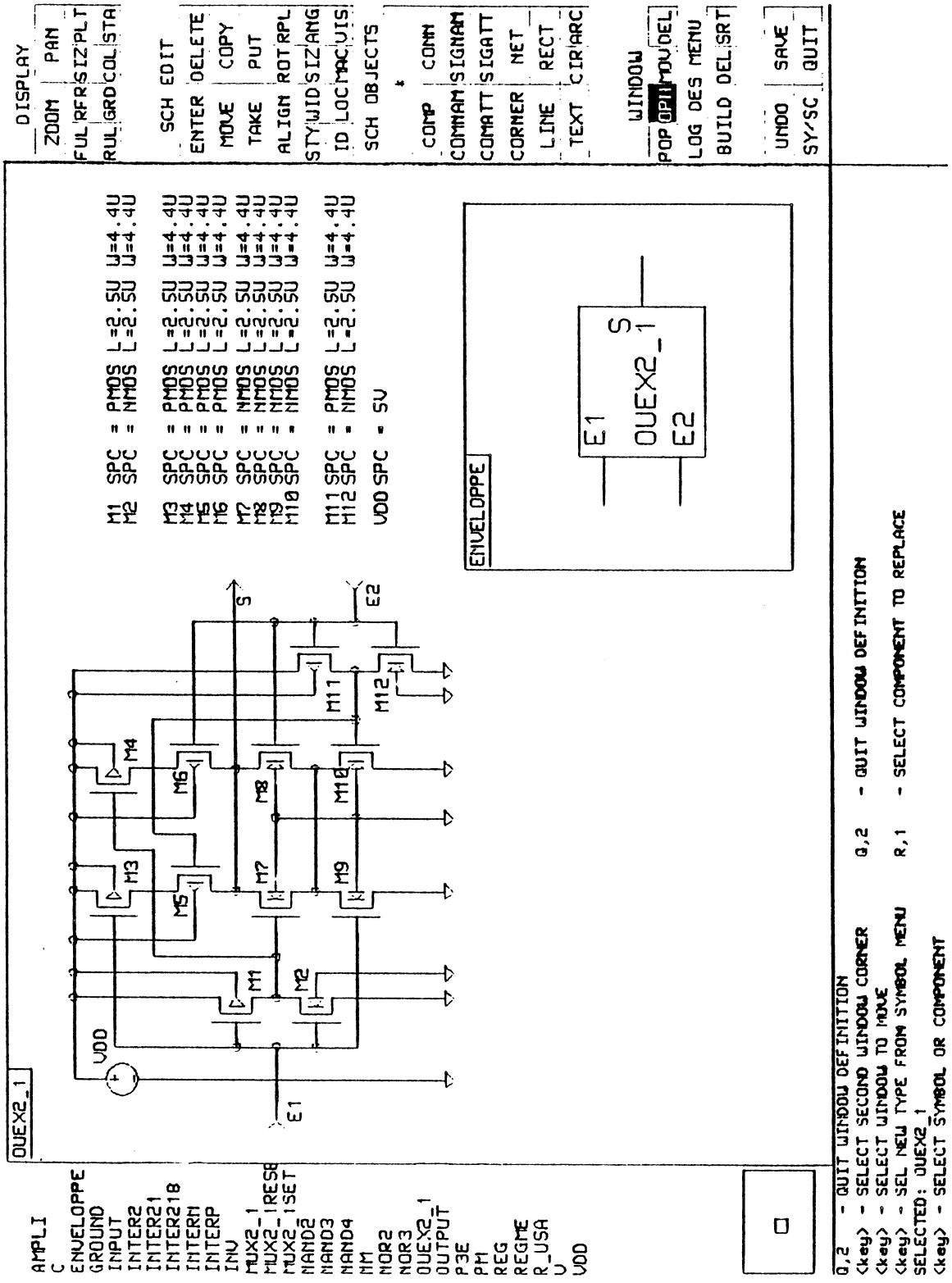
P3E

E

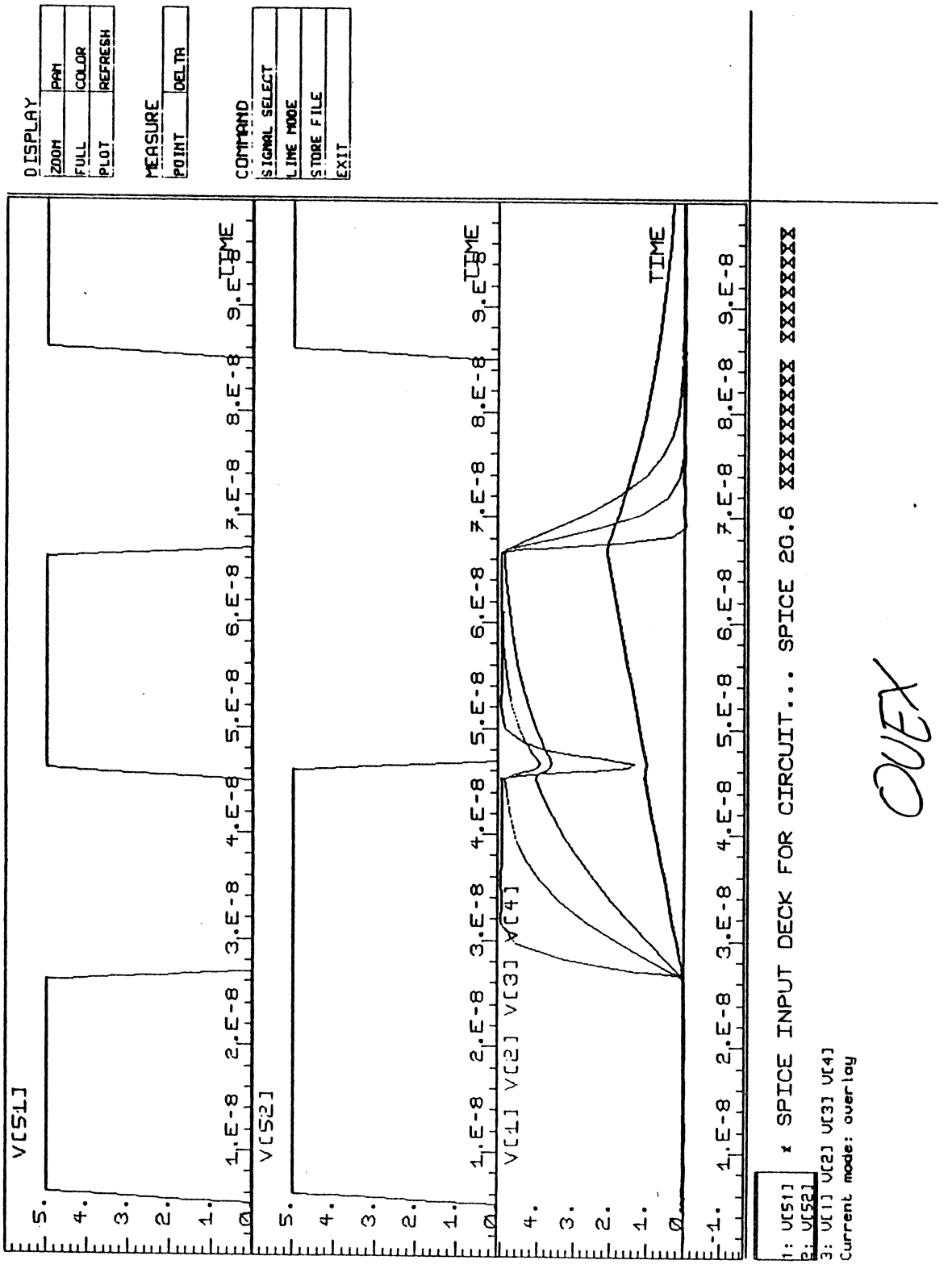
Lire

S

Schéma électrique du OU exclusif 2 entrées



Simulation électrique du OU exclusif



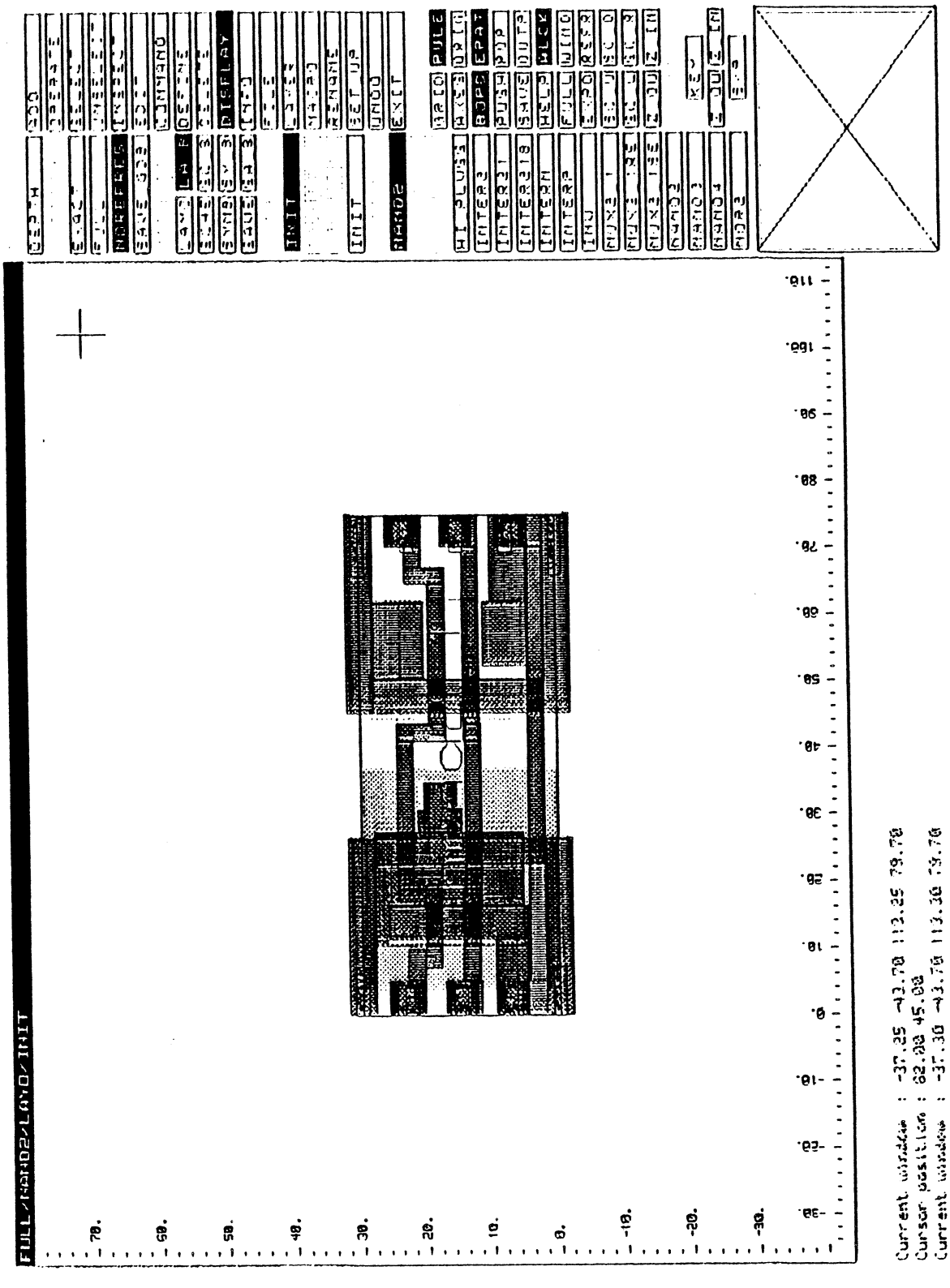
E1

E2

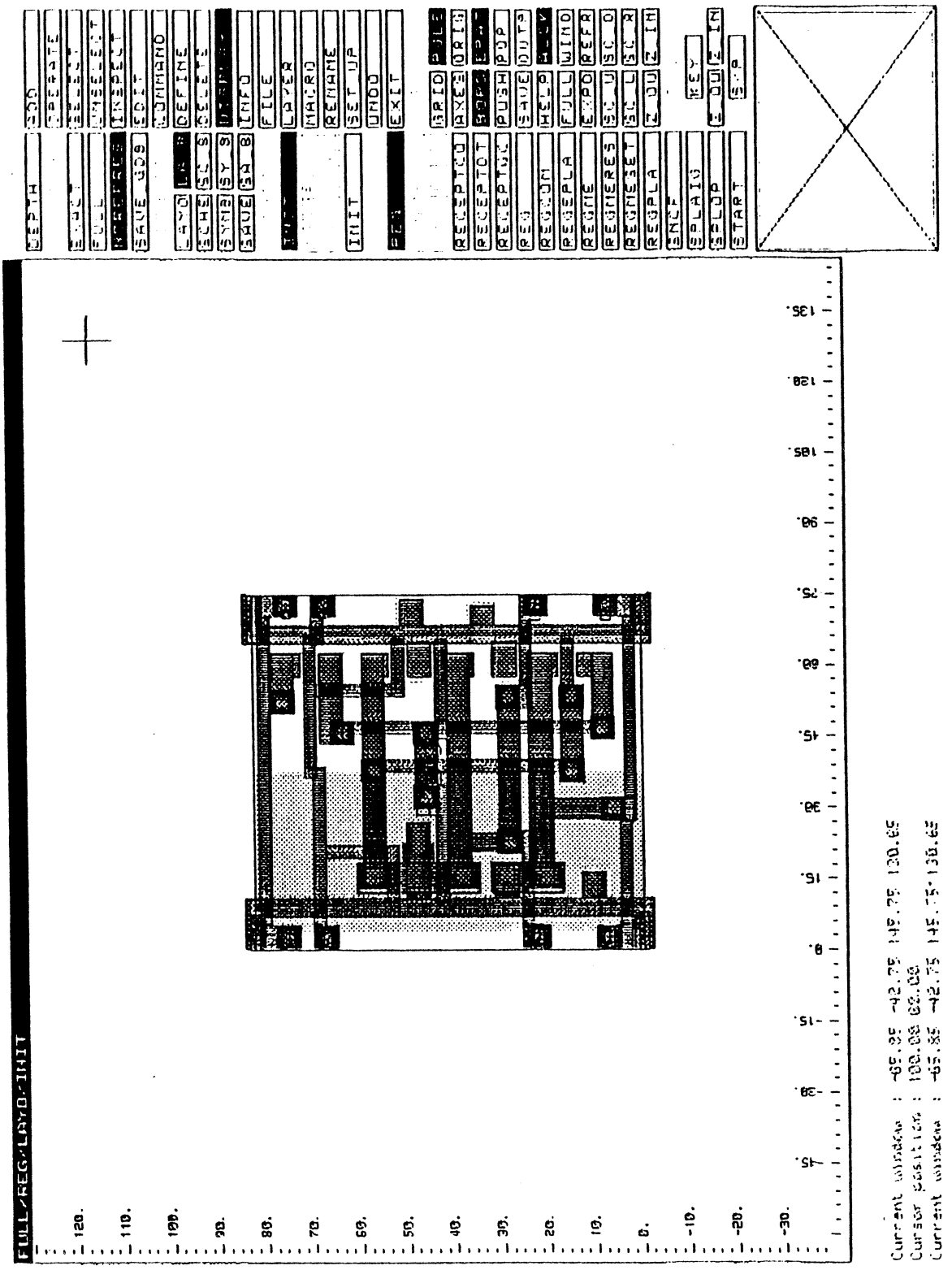
S

OUEX

Dessin des masques du NAND 2 entrées

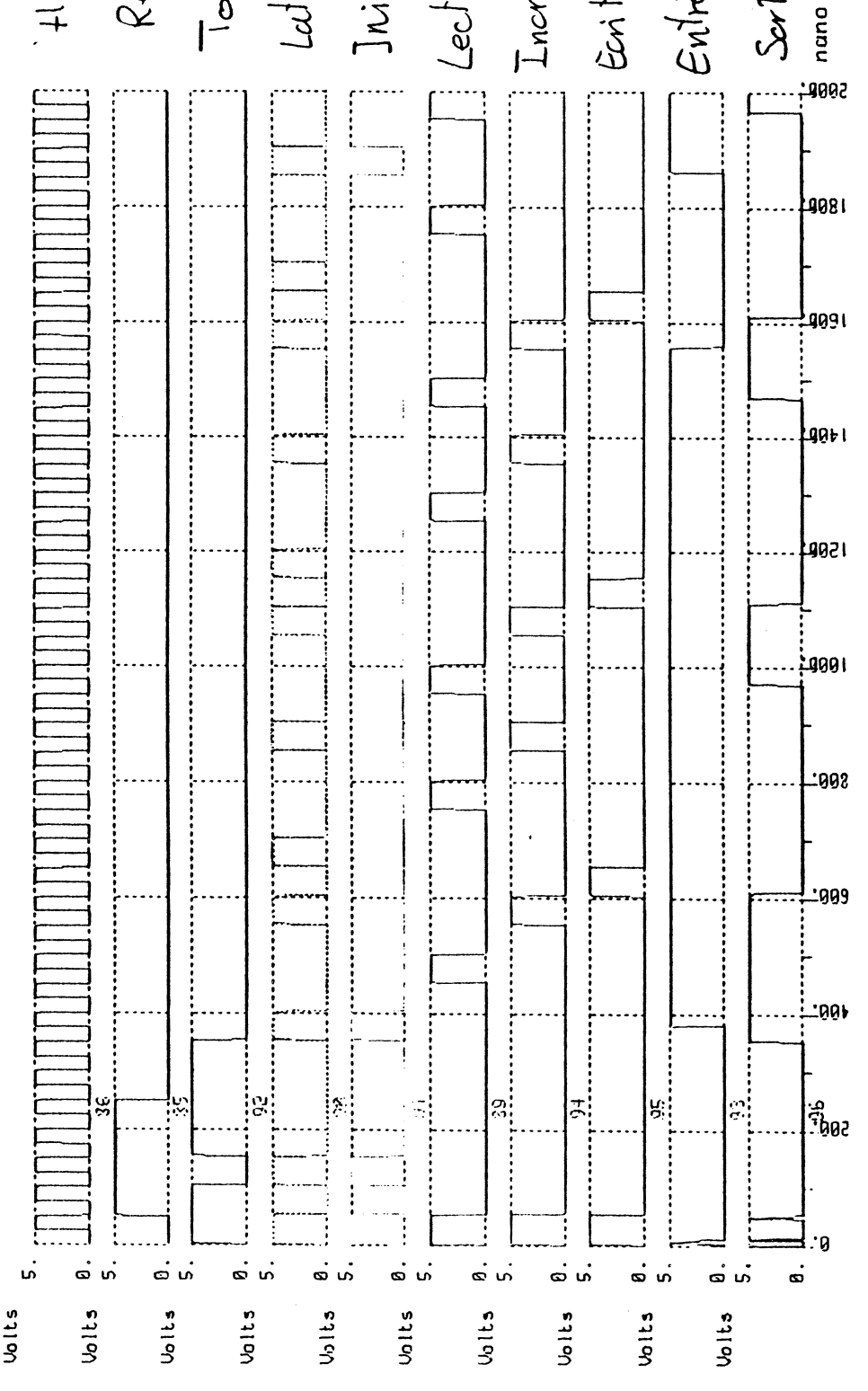


Dessin des masques du latch



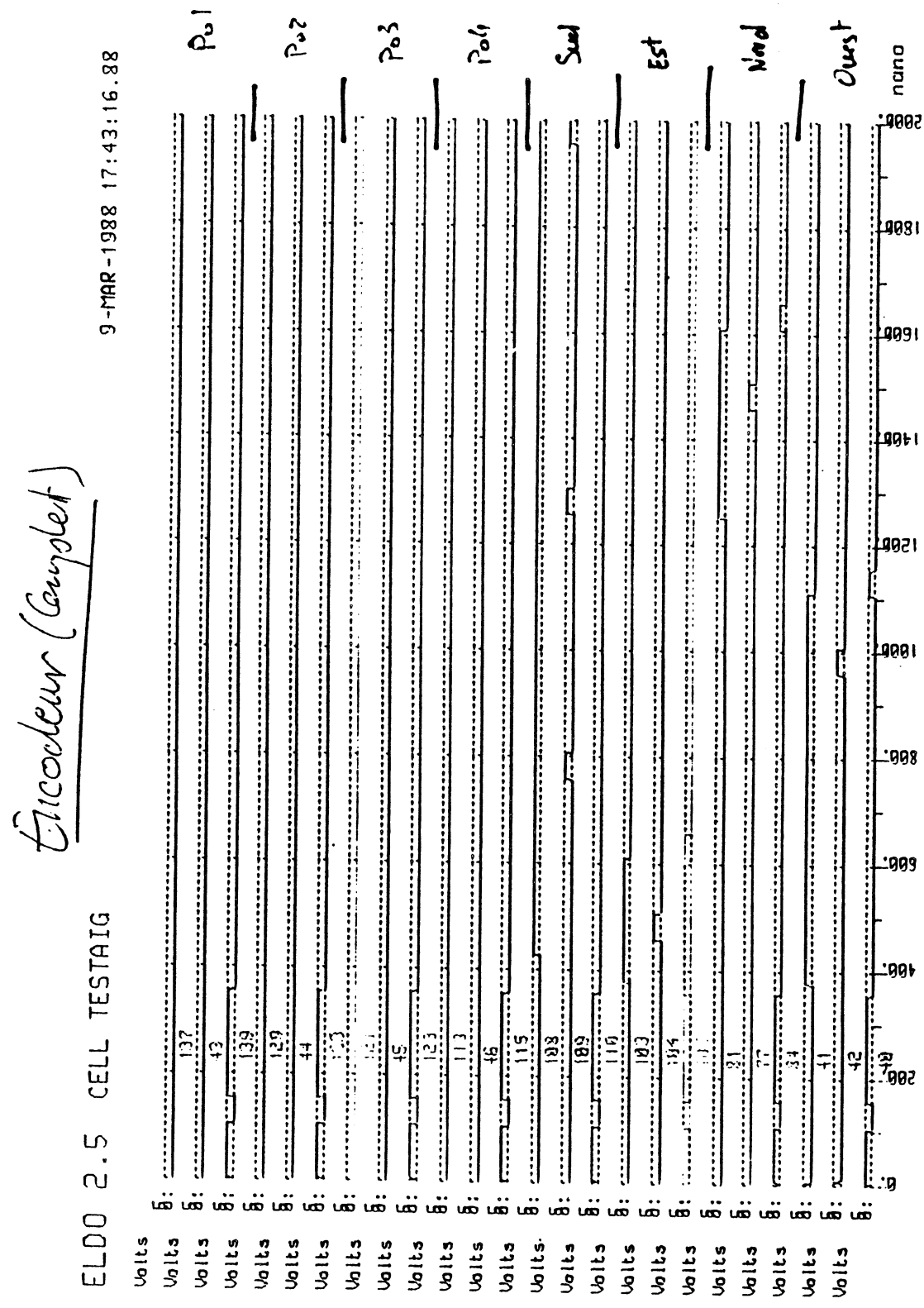
ANNEXE 5

INTEGRATION D'UN RESEAU 2x2

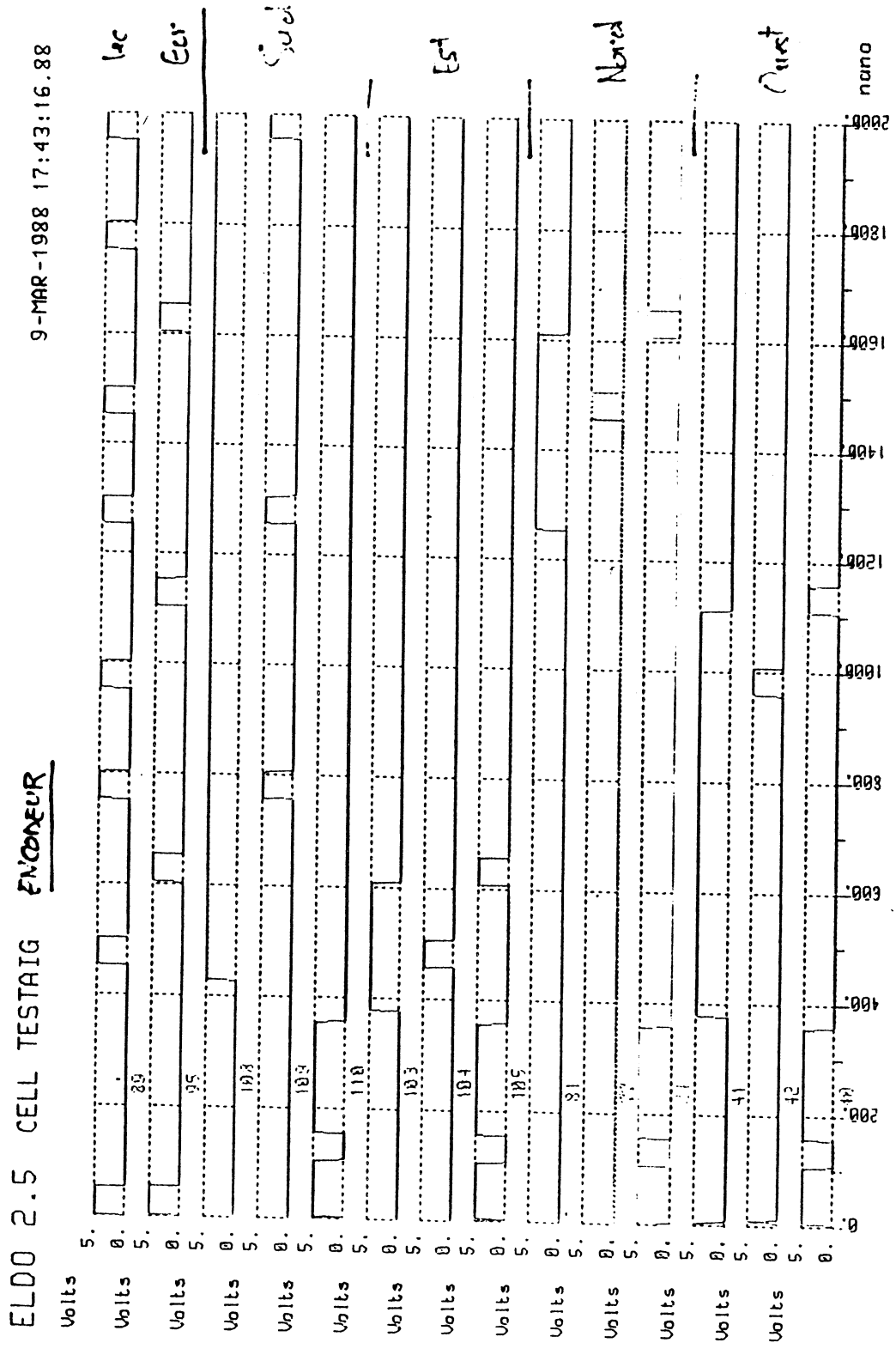


Simulation de l'aiguilleur (à partir du dessin des masques)

1 - Séquencement de l'aiguilleur (sorties du PLA après latchs)

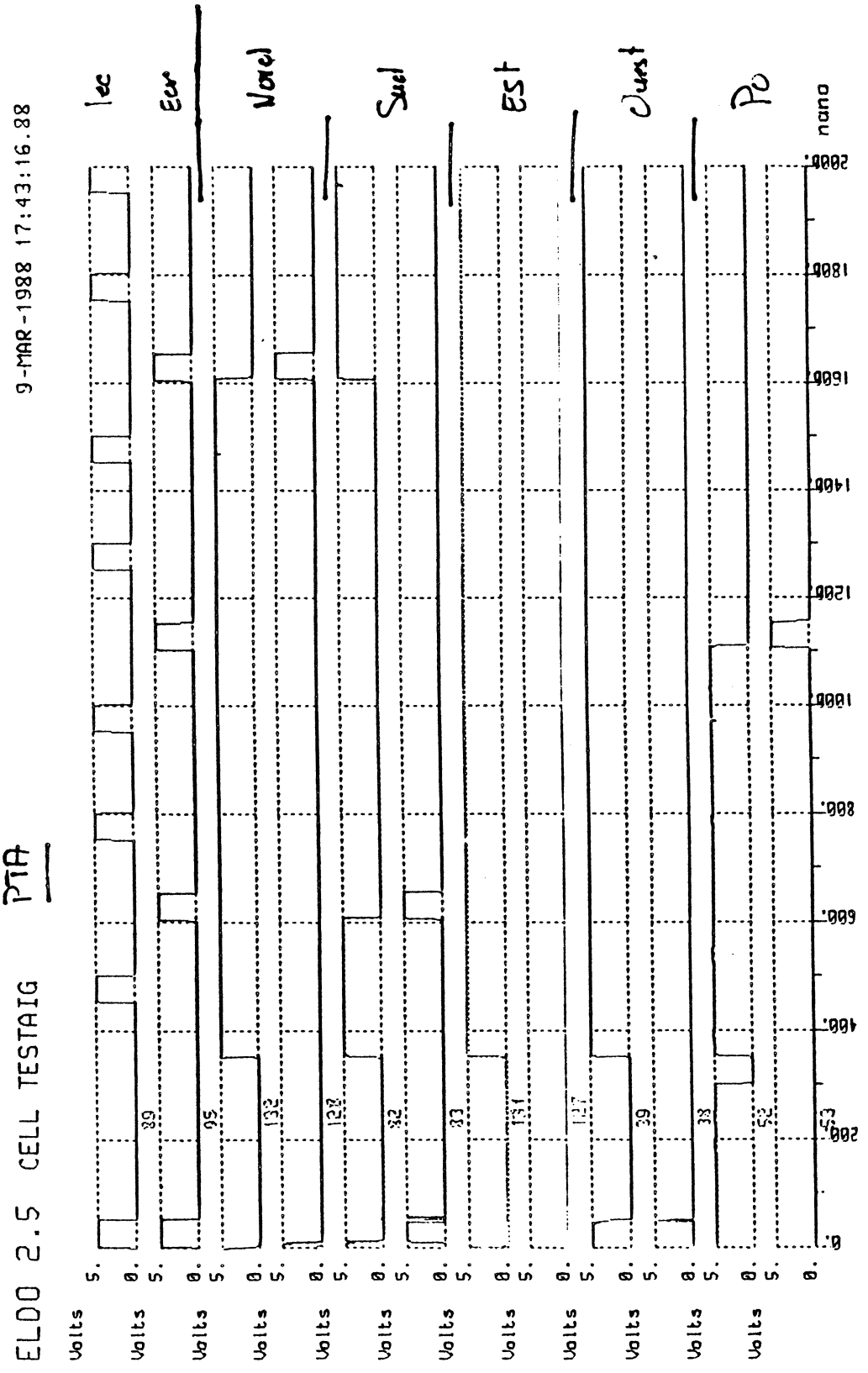


2 - Entrées / sorties de l'encodeur - tampons d'entrée -

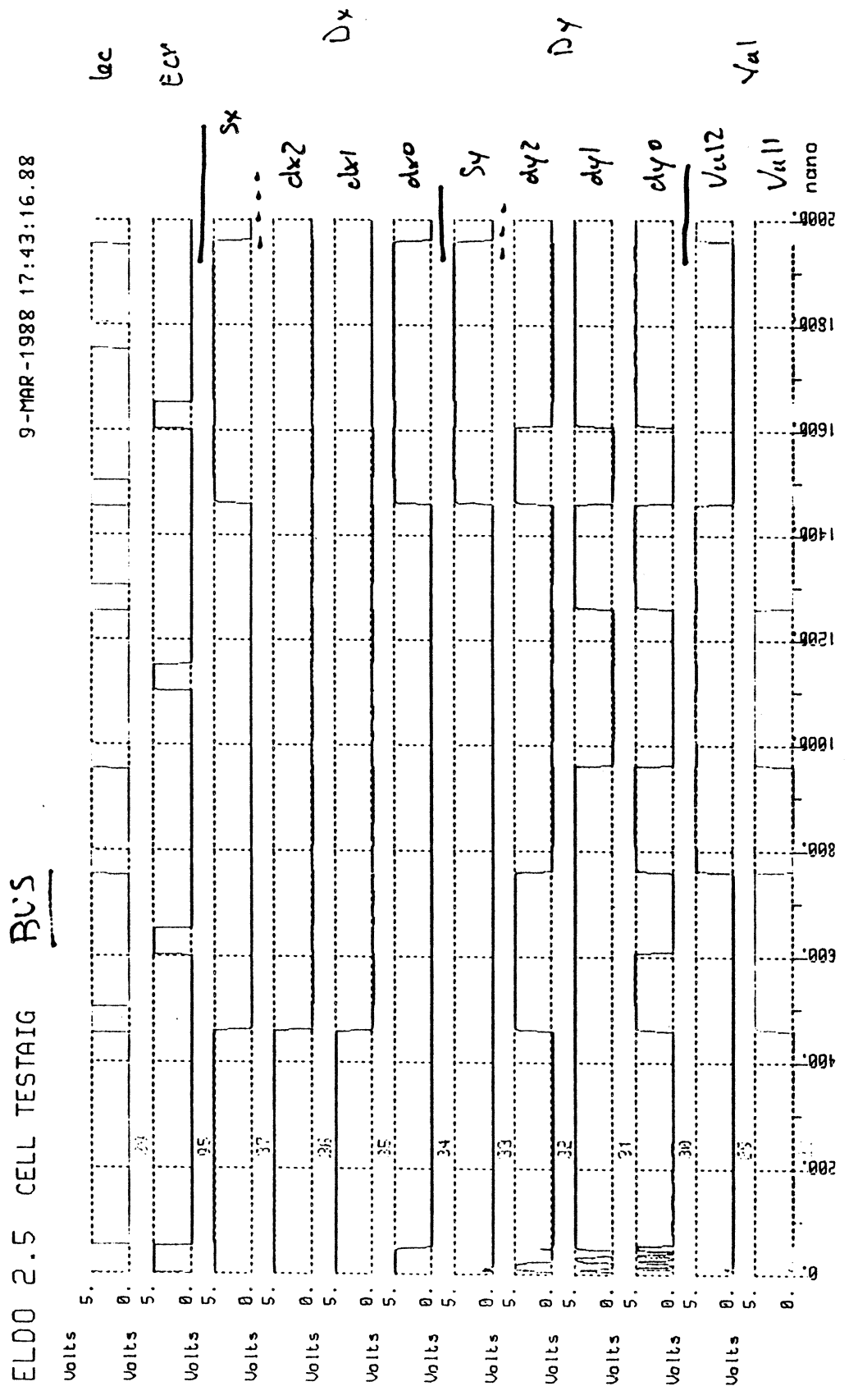


3 - Entrées / sorties de l'encodeur - tampons de sortie -

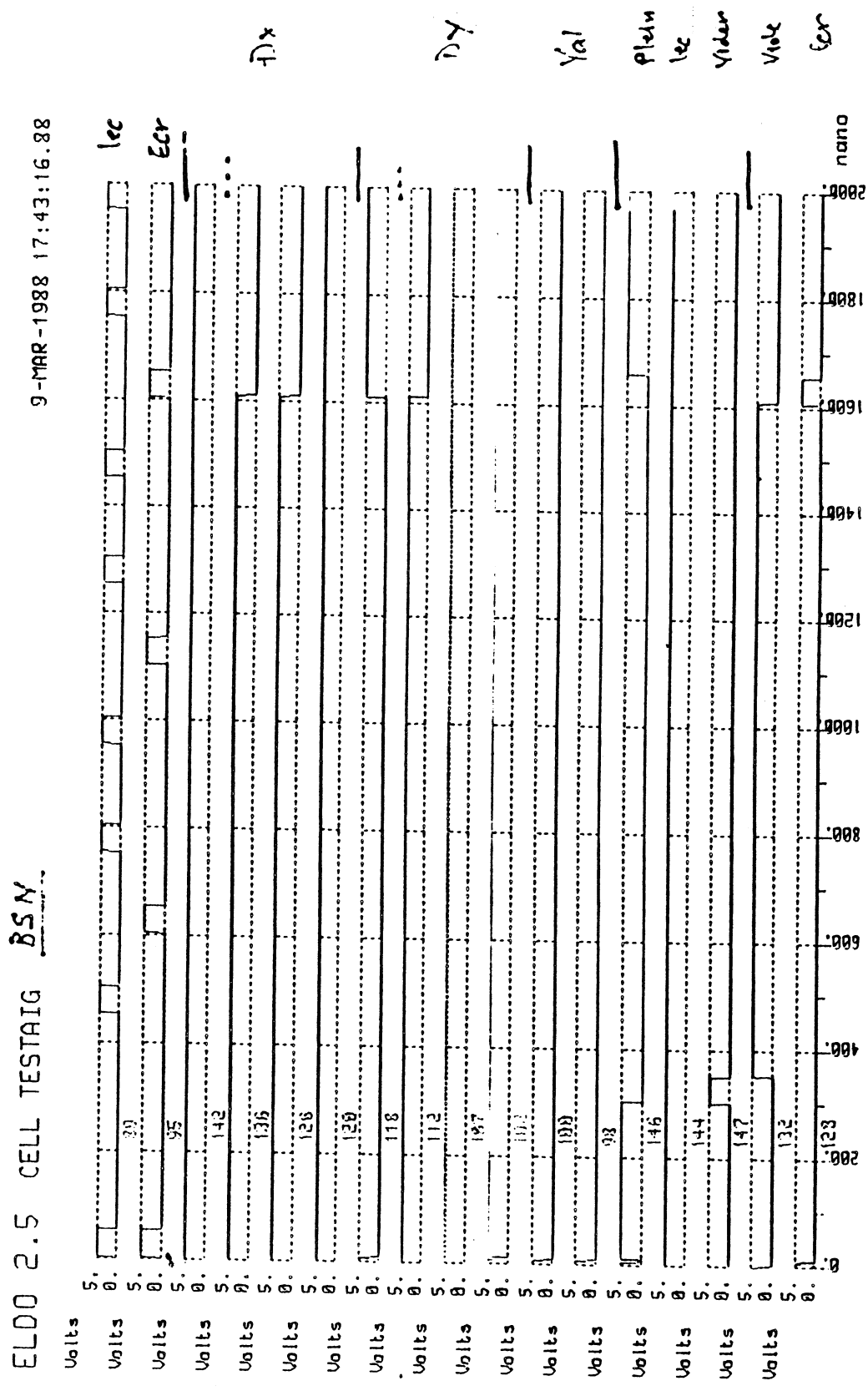
Vide
Ecrit



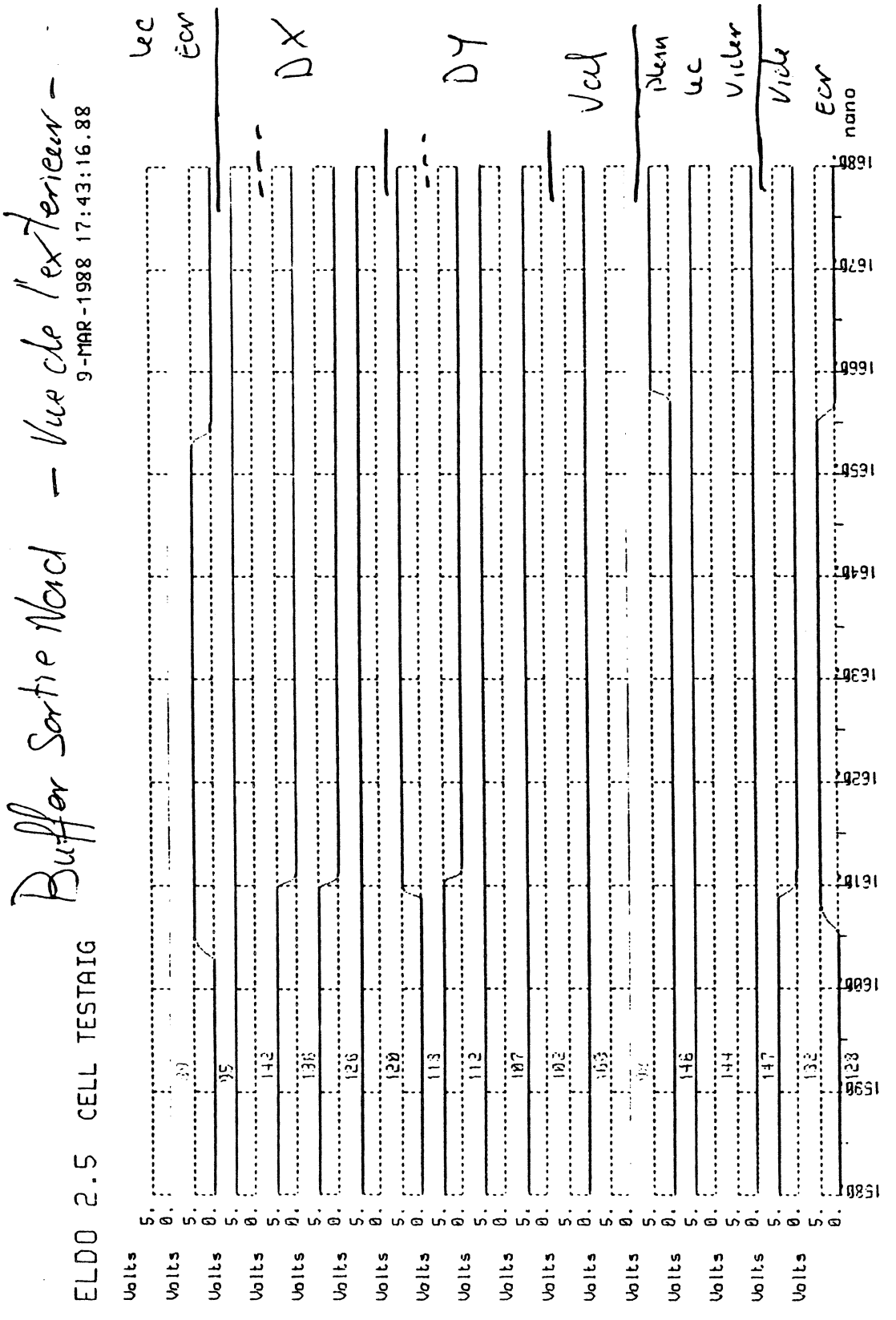
4 - Entrées / sorties de la partie traitement aiguilleur



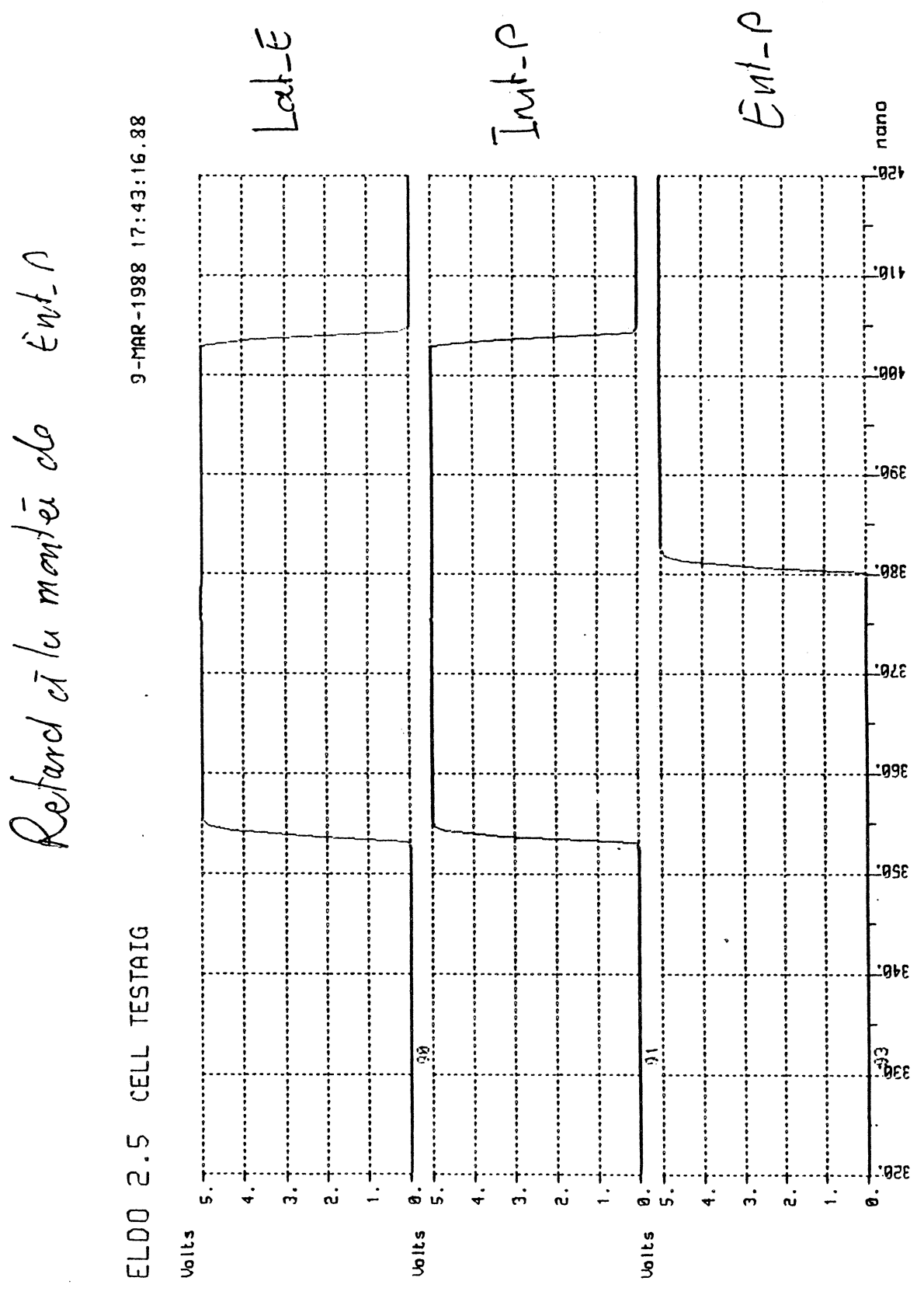
5 - Etats du bus



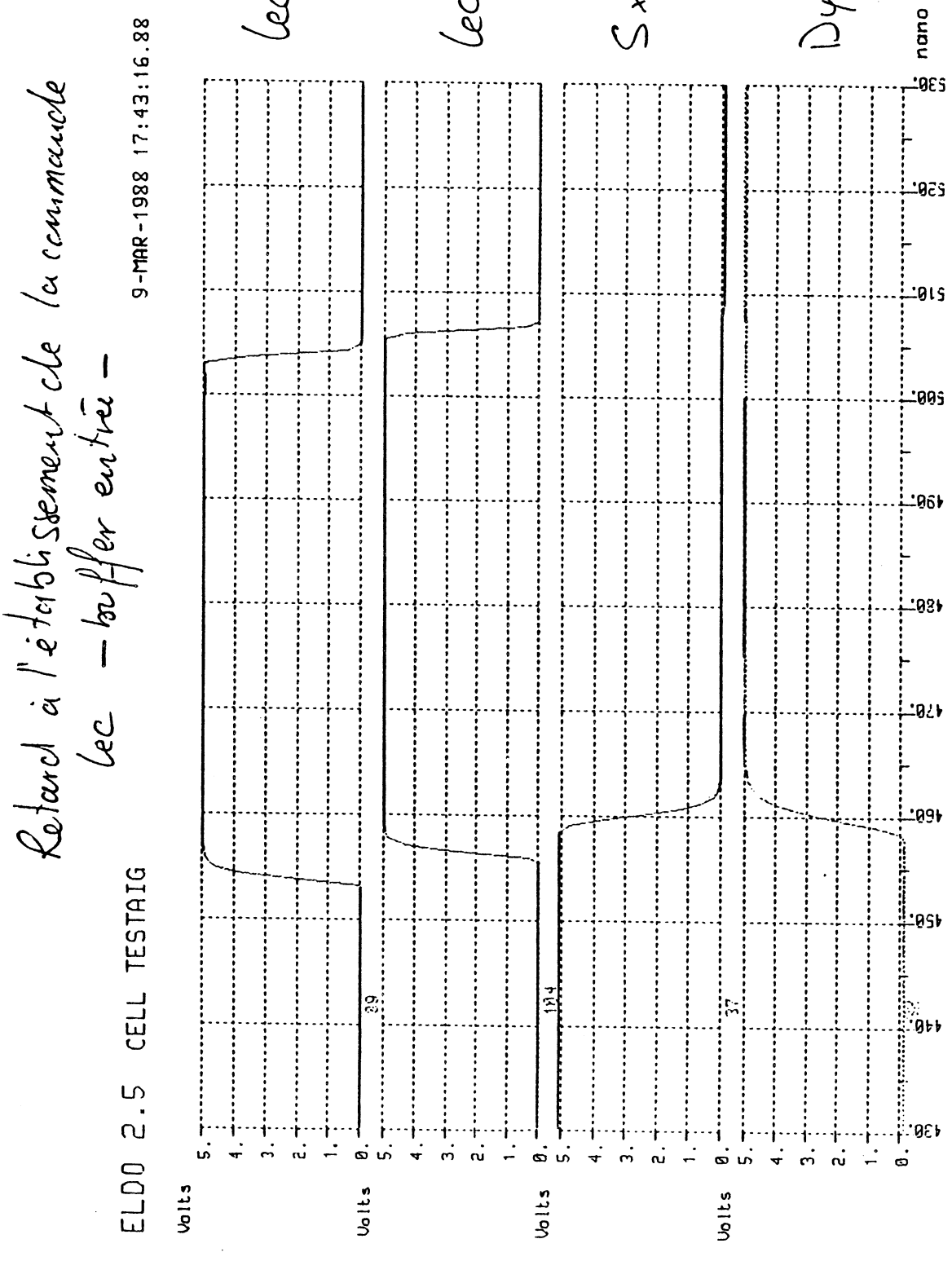
6 - Le tampon de sortie nord (vue par l'aiguilleur)



7 - Le tampon de sortie nord (vue "de l'extérieur")

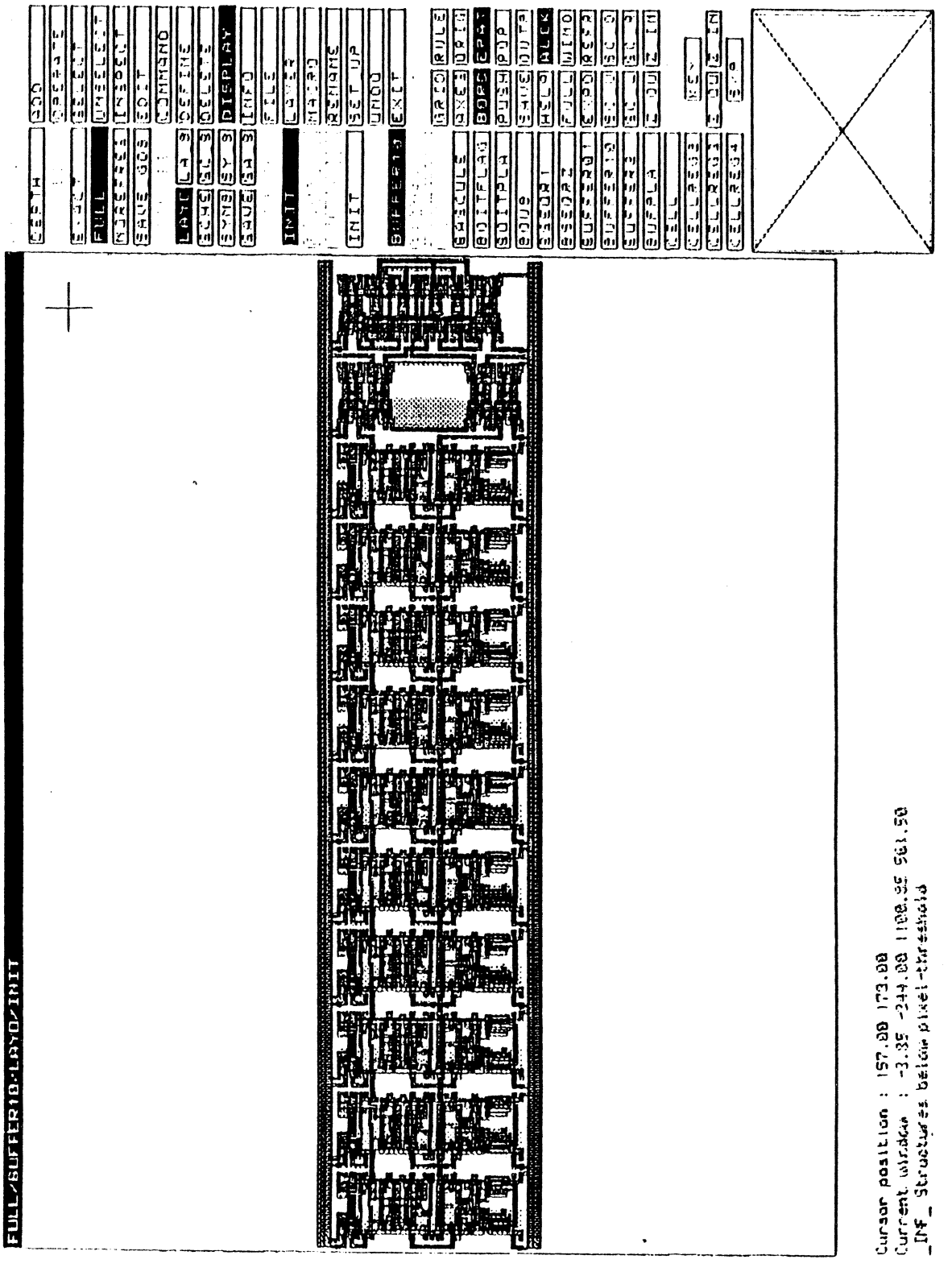


8 - Retard à la montée du signal entrée-pleine

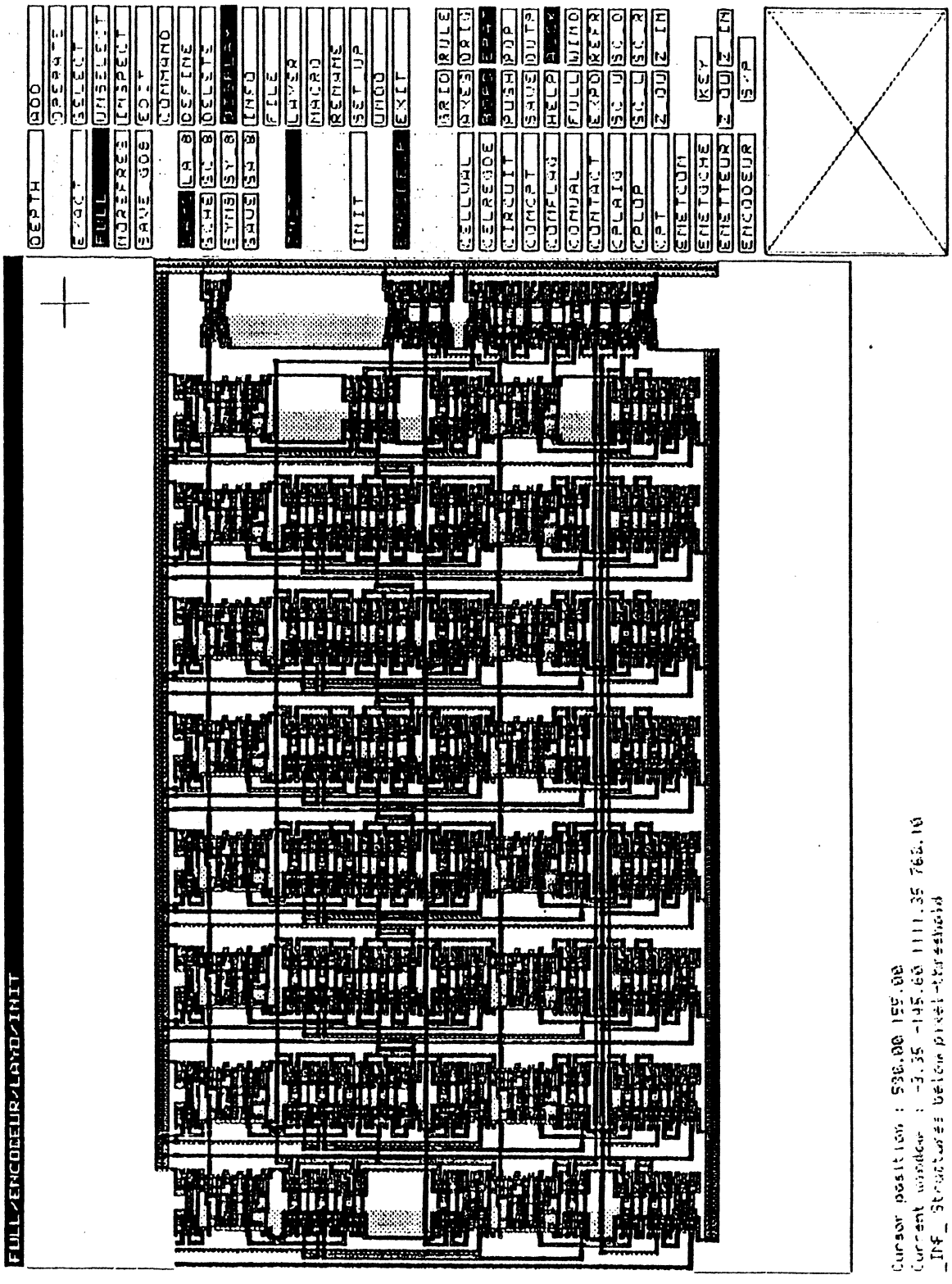


9 - Chargement du bus lors de la lecture d'un tampon d'entrée

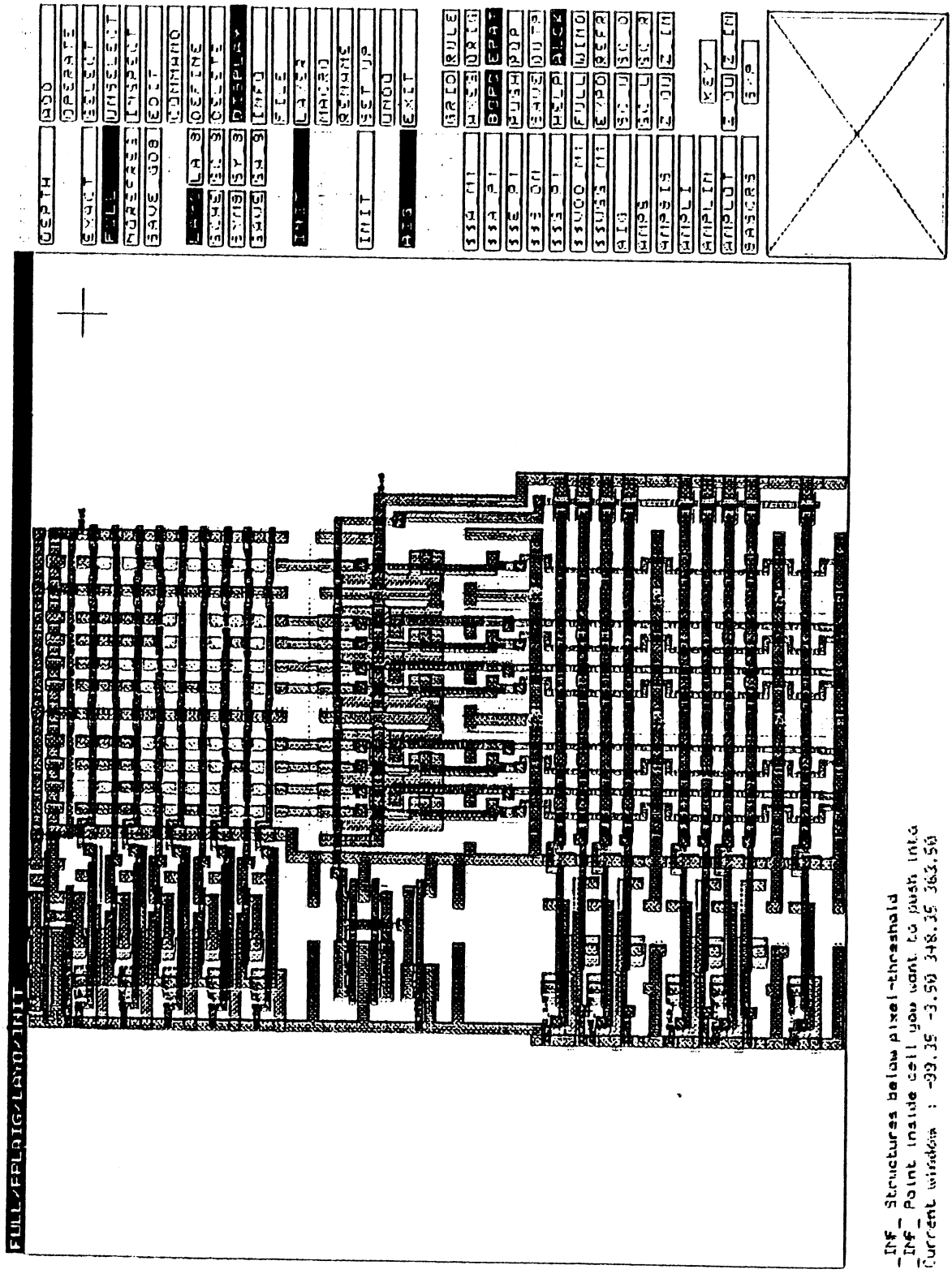
Dessin des masques d'un tampon



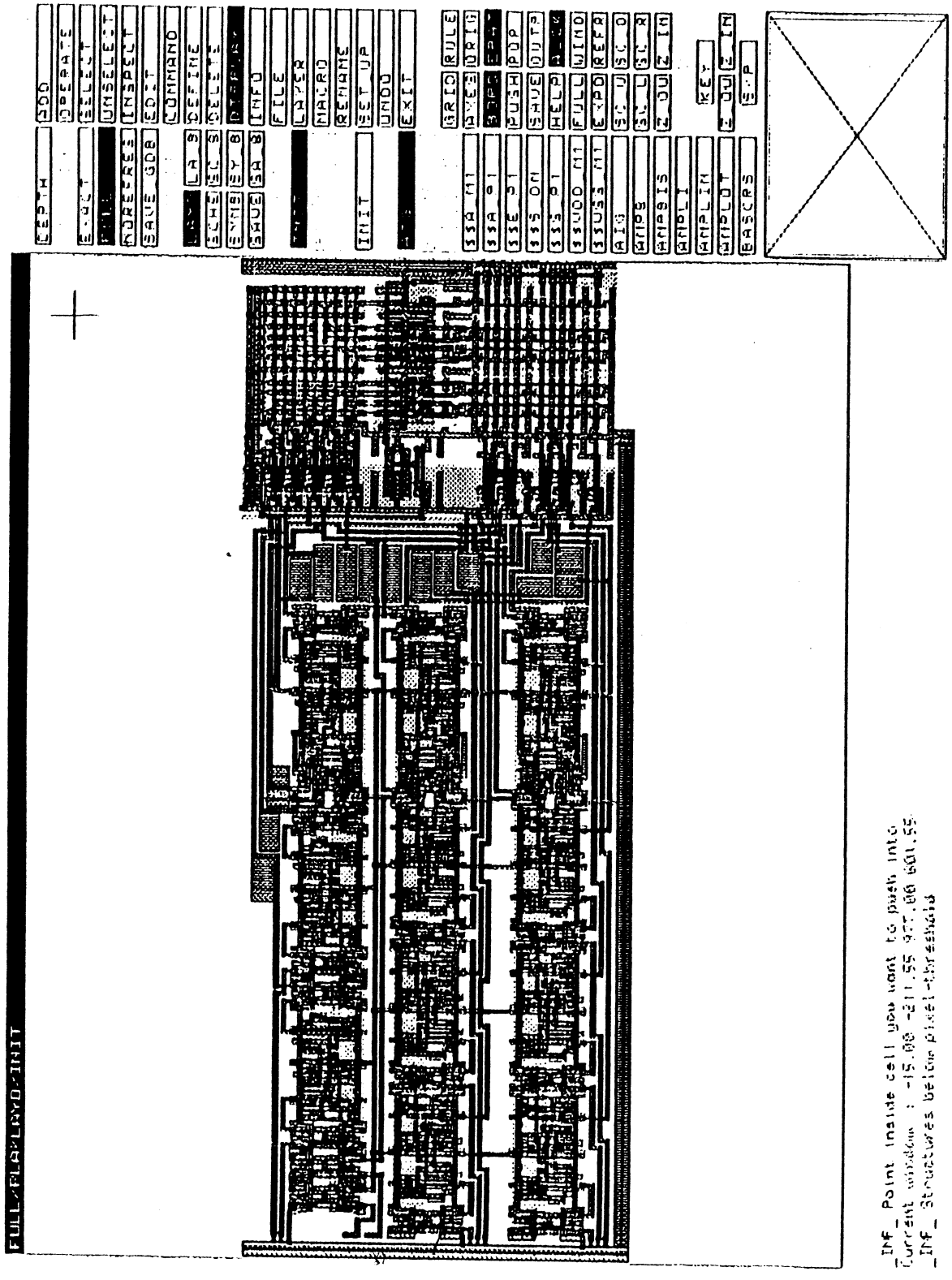
Dessin des masques de l'encodeur de priorité



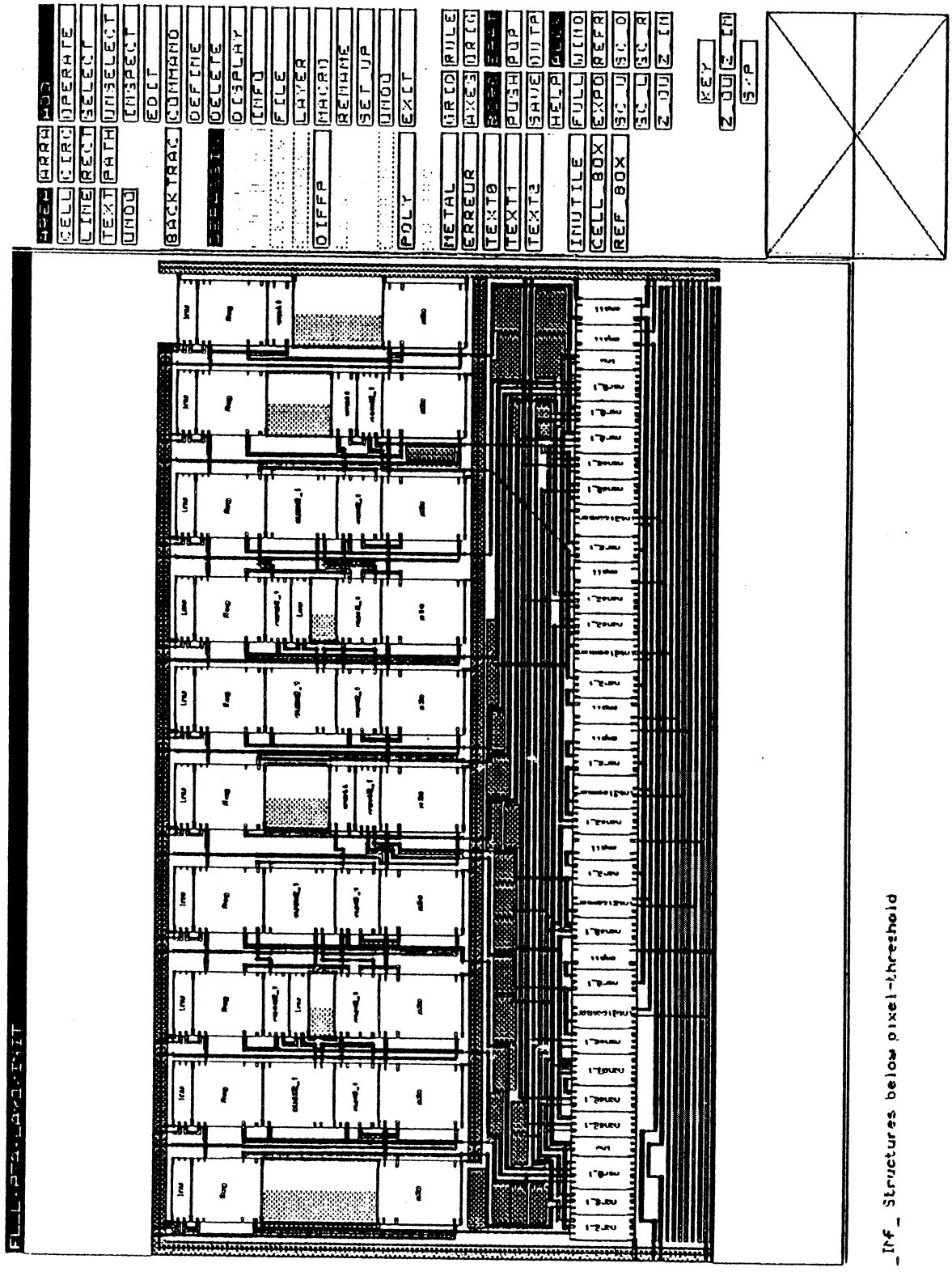
Dessin des masques des matrices du PLA



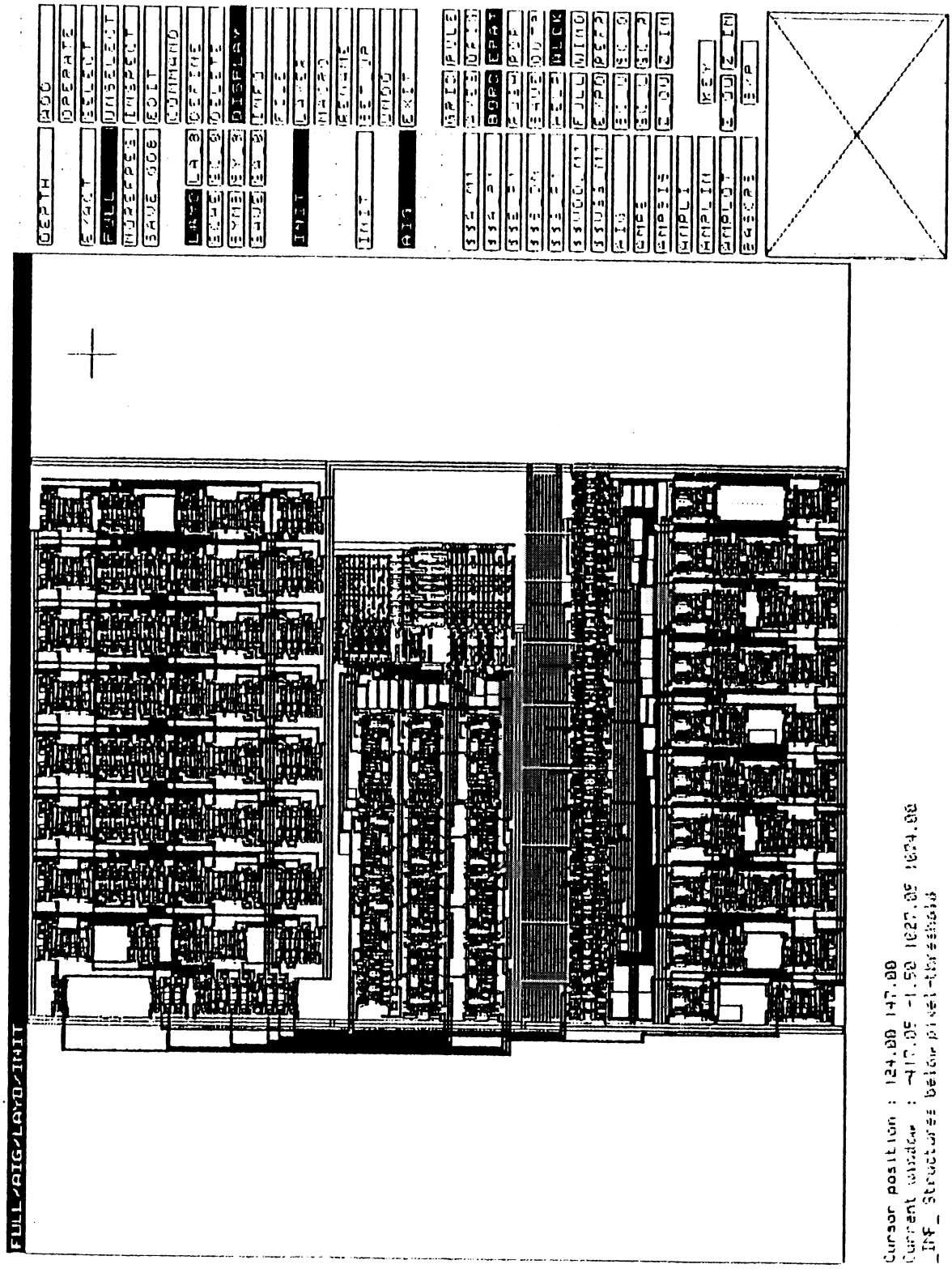
Dessin des masques du bloc PLA avec les latchs



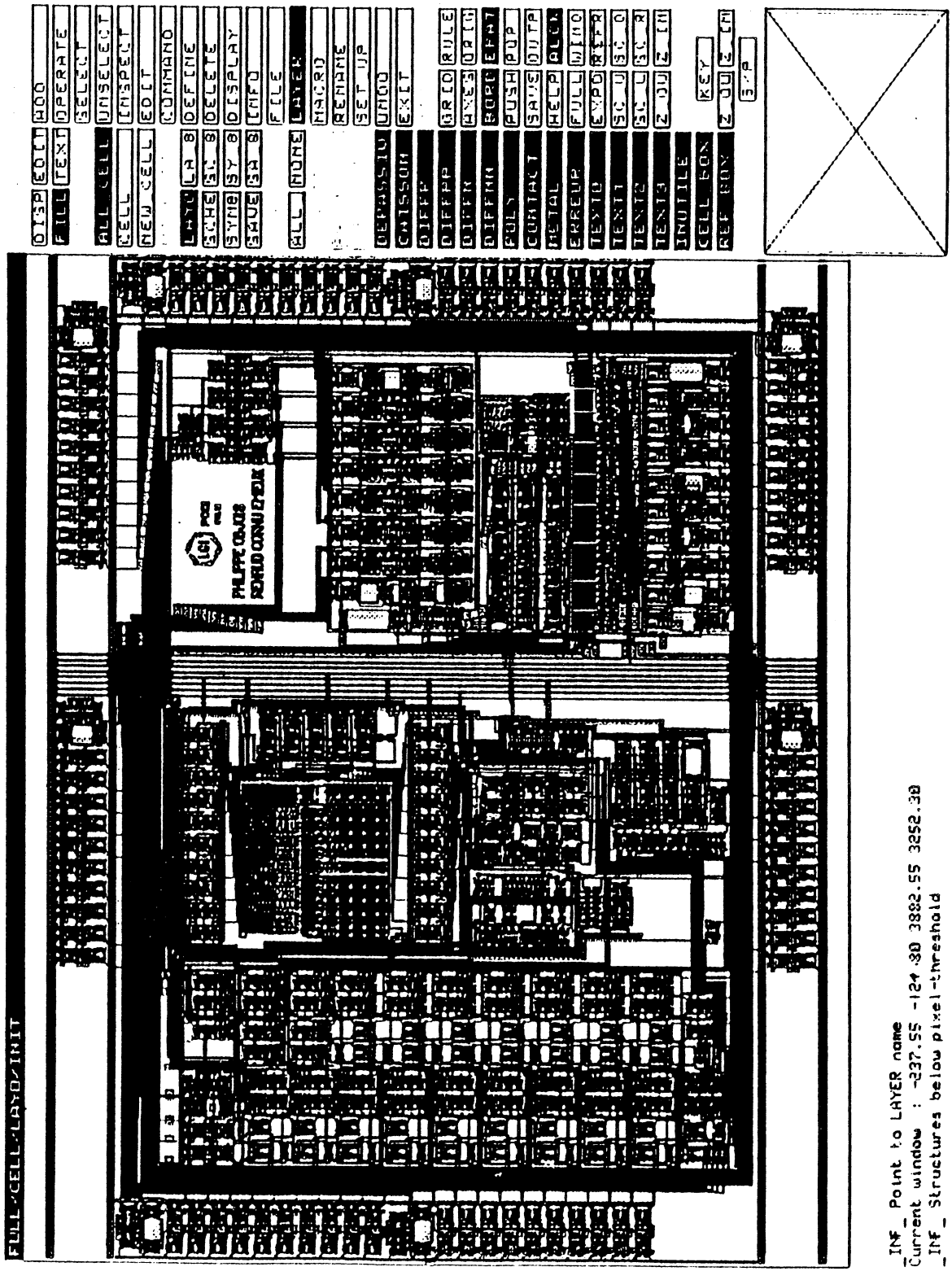
Dessin des masques de la partie-traitement de l'aiguilleur



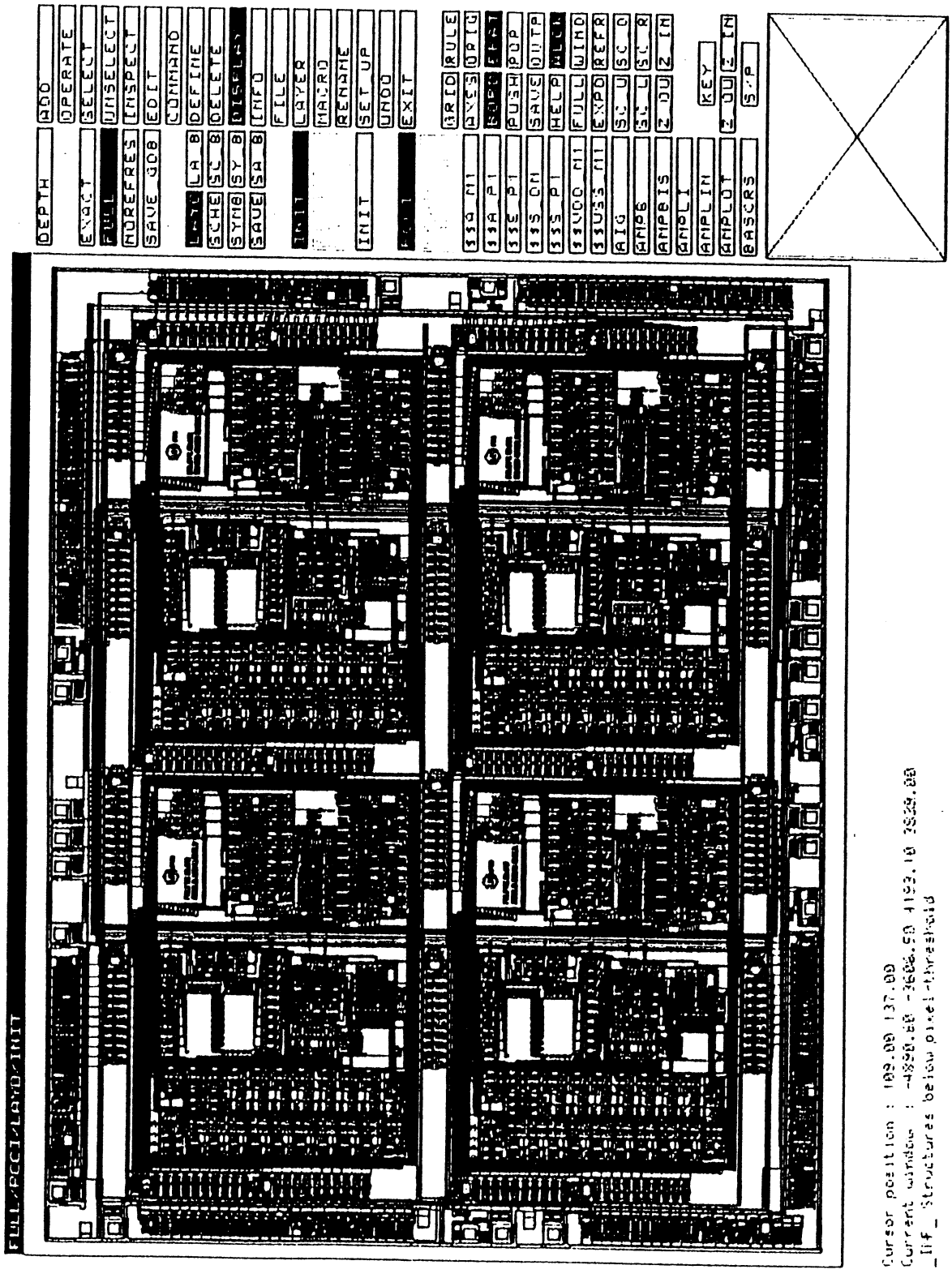
Dessin des masques de l'aiguilleur



Dessin des masques de la cellule



Dessin des masques du circuit complet



INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de Messieurs

. D. ETIEMBLE
. P. QUINTON

Monsieur OBJOIS Philippe

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité "Microélectronique"

Fait à Grenoble, le 15 Septembre 1988

Georges LESPINARD
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

Abstract

Massively parallel integrated architectures are an improved alternative to Von Neumann sequential computers for time consuming tasks. This thesis present a highly parallel architecture based on a regular network of asynchronous cells. Each cell performs a simple task and includes a message routing mechanism allowing communication between any two cells of the network. This type of architecture can be the squeleton of dedicated hardware accelerators able to process efficiently a large class of distributed highly parallel algorithms.

The particular engine we have design concerns an application of logic simulation. The customization of this engine for a given problem is as simple mapping directly the logic circuit on the network by assigning one logical entity per cell. The entire simulation, under supervision of a host computer, consists in two phases : first, customization of the network by initializing each cell, and second, performing of the simulation algorithm distributed among the cells. Different simulation algorithms and synchronization modes are presented.

We have designed an integrated circuit prototype consisting in a 2x2 cell network and communication interfaces and we present its integration on a IBM/PC board.

Key-words

Parallel architecture, asynchronous cellular network, integrated circuit, computer assisted design (CAD), logical simulation, hardware accelerator.

Résumé

Il existe un nouveau schéma de calcul, celui du parallélisme massif, différent du schéma séquentiel de Von Neumann. Nous proposons dans cette thèse une architecture régulière hautement parallèle basée sur un réseau de cellules asynchrones communiquant par messages. Chaque cellule exécute une tâche simple et intègre un mécanisme de communication lui permettant d'échanger des informations avec n'importe quelle autre cellule du réseau.

Cette architecture permet d'exécuter de manière efficace bon nombre d'algorithmes très parallèles.

Nous avons étudié un accélérateur de simulation logique basé sur cette architecture cellulaire. Le principe est d'associer, à chaque cellule du réseau, un élément logique du circuit à simuler. Contrôlée par un système-hôte, la simulation se déroule en 2 temps :

1 - initialisation des cellules du réseau ;

2 - exécution de l'algorithme réparti dans les cellules.

Plusieurs algorithmes de simulation ainsi que différents modes de synchronisation sont présentés.

Nous présentons la réalisation d'un circuit, dédié à cette application, intégrant un réseau 2x2 et ses interfaces de communication, ainsi qu'une machine prototype de simulation logique basée sur ce circuit utilisant un ordinateur IBM PC/AT comme système-hôte.

Mots-clés

Architecture parallèle, réseau cellulaire asynchrone, circuit intégré, conception assisté par ordinateur (CAO), simulation logique, accélérateur matériel.