Conception et réalisation d'un processeur pour une architecture cellulaire massivement parallèle intégrée

Si Mohamoud Karabernou

THESE

Présentée par

Si Mahmoud KARABERNOU

pour obtenir le grade de **DOCTEUR**

de l'**INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(Arrêté ministériel du 30 Mars 1992)

(Spécialité : Microélectronique)

CONCEPTION ET RÉALISATION D'UN PROCESSEUR POUR UNE ARCHITECTURE CELLULAIRE MASSIVEMENT PARALLÈLE INTÉGRÉE

Date de soutenance : 8 Juillet 1993

Composition du jury :

Messieurs	Yves	CHIARAMELLA	Président
	Guy	MAZARÉ	
	Gérard	MICHEL	
	Alain	GUYOT	
	Patrick	GARDA	Rapporteurs
	Bruno	ROUZEYRE	

Thèse préparée au sein du Laboratoire de Génie Informatique

UNIVERSITE
JOSEPH FOURIER
SCIENCES. TECHNOLOGIE. MEDECINE
GRENOBLE 1

Président de l'Université :

M. NEMOZ Alain

ANNEE UNIVERSITAIRE 1990 - 1991

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ERE CLASSE

ADIBA Michel	Informatique
ANTOINE Pierre	Géologie I.R.I.G.M.
ARVIEU Robert	Physique Nucléaire I.S.N.
AURIAULT Jean Louis	Mécanique
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
BLANCHI Jean Pierre	A.P.S.
BOEHLER Jean Paul	Mécanique
BOITET Christian	Informatique et Mathématiques Appliquées
BORNAREL Jean	
BRUANDET Jean François	Physique
CARLIER Georges	Biologie Végétale
CASTAING Bernard	Physique
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
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
FINKE Gerde	Informatique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GERMAIN Jean Pierre	Mécanique
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences nucléaires
GILLARD Roland	Mathématiques
GUITTON Jacques	Chimie

../ ...

HERAULT Jeanny	Physique
HICTER Pierre	Chimie
JANIN Bernard	Géographie
JOLY Jean René	Mathématiques Pures
JOSELEAU Jean Paul	Biochimie
KAHANE André	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
LOISEAUX Jean Marie	Sciences Nucléaires I.S.N.
LONGEQUEUE Nicole	Physique
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
NEMOZ Alain	Physique
OMONT Alain	Astrophysique
PELMONT Jean	Biochimie
PERRIER Guy	Géophysique
PIERRE Jean Louis	Chimie Organique
RENARD Michel	Thermodynamique
RICHARD Jean Marc	
RIEDTMANN Christine	Mathématiques
RINAUDO Marguerite	Chimie C.E.R.M.A.V.
ROBERT Jean Bernard	
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VALLADE Marcel	Physique
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie
VIDAL Micheal	

PROFESSEURS DE 2EME CLASSE

APPARU Marcel	Chimie
ARMAND Gilbert	Géographie
ARNAUD Hubert	Géologie
ARTRU Marie Christine	Physique
ATTANE Pierre	Mécanique
BARATE Robert	Sciences Nucléaires
BARET Paul	Chimie
BARGE Jean	Mathématiques
BARLET Roger	Chimie
BERTIN José	Mathématiques
BLOCK Marc	Biologie
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORRIONE Dominique	Automatique informatique
BOULON Marc	Mécanique
BOUTRON Claude	Glaciologie
BOUVET Jean	Biologie
BROSSARD Jean	Mathématiques
BRUGAL Gérard	Biologie
CAMPILLO Michel	Géophysique
CAVAILLE Jean Yves	Chimie
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
CHOLLET Jean Pierre	Mécanique
COLOMBEAU Jean François	Mathématiques (ENSL)
COTTET Georges-Henri	Modélisation, calcul scientifique, statis.
COURT Jean	Chimie
CUNIN Pierre Yves	Informatique
DAVID Jean	Géographie
DEROUARD Jacques	Physique
DHOUAILLY Danielle	Biologie
DUFRESNOY Alain	Mathématiques Pures
DUPUY Claude	Chimie
DURAND Mireille	Sciences Nucléaires
FONTECAVE Marc	Chimie
FOURNIER Jean Marc	Physique
GASPARD François	Physique
GIDON Maurice	Géologie
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GOURC Jean Pierre	Mécanique
GUIGO Maryse	Géographie
GUMUCHIAN Hervé	Géographie
HACQUES Gérard	Mathématiques Appliquées
HAMMOU Abdelkader	Chimie
HERBIN Jacky	Géographie
HERINO Roland	Physique
HERZOG Michel	Biologie
JARDON Pierre	Chimie
JUTTEN Christian	Physique
KERCKHOVE Claude	Géologie
KOSAREW Siegmund	Math. fondamentales et appliquées
KLINGER Jurgen	Glaciologie
LAURENT Christine	Mathématiques
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
MERCHEZ Fernand	Physique
MILAS Michel	Chimie
MOREL Alain	Géographie
MORIN Pierre	Physique
NGUYEN HUY Xuong-	Informatique
OUDET Bruno	Mathématiques Appliquées

../ ...

PAUTOU Guy	Biologie
PECHER Arnaud	Géologie
PELLETIER Guy	Astrophysique
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Claude	Biologie
PIBOULE Michel	Géologie
PORTESEIL Jean Louis	Physique
PUECH Laurent	Physique
RAYNAUD Hervé	Mathématiques Appliquées
REGNARD Jean René	Physique
ROBERT Claudine	Didactique des disciplines scientifiques
ROBERT Danielle	Chimie
ROBERT Gilles	Mathématiques Pures
SAJOT Gérard	Physique
SARROT REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
TEMPERVILLE André	Mécanique
TISSUT Michel	Biologie
TOURNIER Evelyne	Informatique et Mathématiques appliquées
VALLADE Marcel	Physique
VALLON Michel	Glaciologie
VICAT Jean	Physique
VINCENS Maurice	Chimie
VINCENT Gilbert	Physique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie
WITOMSKI Patrick	

PRESIDENT DE L'INSTITUT
Monsieur Maurice RENAUD

INPG

Année 1993

PROFESSEURS DES UNIVERSITES

BARIBAUD	Michel	ENSERG
BARRAUD	Alain	ENSIEG
BARTHELEMY	Alain	ENSHMG
BAUDELET	Bernard	ENSPG
BAUDIN	Gérard	UFR PGP
BEAUFILS	Jean-Pierre	ENSIEG/ILL
BOIS	Philippe	ENSHMG
BOUVIER	Gérard	ENSERG
BRINI	Jean	ENSERG
BRUNET	Yves	CUEFA
CAVAIGNAC	Jean-François	ENSPG
CHARTIER	Germain	ENSPG
CHENEVIER	Pierre	ENSERG
CHERUY	Arlette	ENSIEG
CHOVET	Alain	ENSERG
COGNET	Gérard	ENSGI
COLINET	Catherine	ENSEEG
COMMAULT	Christian	ENSIEG
CORNUT	Bruno	ENSIEG
COULOMB	Jean-Louis	ENSIEG
COUTRIS	Nicole	ENSPG
CROWLEY	James	ENSIMAG
DALARD	Francis	ENSEEG
DARVE	Félix	ENSHMG
DELLA DORA	Jean	ENSIMAG
DEPEY	Maurice	ENSERG
DEPORTES	Jacques	ENSPG
DEROO	Daniel	ENSEEG
DESRE	Pierre	ENSEEG
DIARD	Jean-Paul	ENSEEG
DOLMAZON	Jean-Marc	ENSERG
DURAND	Francis	ENSEEG
DURAND	Jean-Louis	ENSPG
FAUTRELLE	Yves	ENSHMG
FOGGIA	Albert	ENSIEG
FORAY	Pierre	ENSHMG
FOULARD	Claude	ENSIEG
GALERIE	Alain	ENSEEG
GANDINI	Alessandro	UFR/PGP
GAUBERT	Claude	ENSPG
GENTIL	Pierre	ENSERG
GENTIL	Sylviane	ENSIEG
GUERIN	Bernard	ENSERG
GUYOT	Pierre	ENSEEG
IVANES	Marcel	ENSIEG
JACQUET	Paul	ENSIMAG
JALLUT	Christian	ENSEEG
JANOT	Marie-Thérese	ENSERG

Tél. (33) 76 57 45 00 Telex 320 205 F Télécopie (33) 76 57 45 01

JAULENT	Patrick	ENSGI
JAUSSAUD	Pierre	ENSIEG
JOST	Rémy	ENSPG
JOUBERT	Jean-Claude	ENSPG
JOURDAIN	Geneviéve	ENSIEG
KUENY	Jean-Louis	ENSHMG
LACHENAL	Dominique	UFR PGP
LACOUME	Jean-Louis	ENSIEG
LADET	Pierre	ENSIEG
LE NEST	Jean-François	UFR/PGP
LESIEUR	Marcel	ENSHMG
LESPINARD	Georges	ENSHMG
LIENARD	Joël	ENSIEG
LONGEQUEUE	Jean-Pierre	ENSPG
LORET	Benjamin	ENSHMG
LOUCHET	François	ENSEEG
LUCAZEAU	Guy	ENSEEG
LUX	Augustin	ENSIMAG
MASSE	Philippe	ENSPG
MASSELOT	Christian	ENSIEG
MAZARE	Guy	ENSIMAG
MICHEL	Gérard	ENSIMAG
MOHR	Roger	ENSIMAG
MOREAU	René	ENSHMG
MORET	Roger	ENSIEG
MOSSIERE	Jacques	ENSIMAG
OBLED	Charles	ENSHMG
OZIL	Patrick	ENSEEG
PANANAKAKIS	Georges	ENSERG
PAULEAU	Yves	ENSEEG
PERRET	Robert	ENSIEG
PERRIER	Pascal	ENSERG
PIAU	Jean-Michel	ENSHMG
PIC	Etienne	ENSERG
PLATEAU	Brigitte	ENSIMAG
POUPOT	Christian	ENSERG
RAMEAU	Jean-Jacques	ENSEEG
REINISCH	Raymond	ENSPG
RENAUD	Maurice	UFR/PGP
RIMET	Roger	ENSERG
ROBERT	François	ENSIMAG
ROGNON	Jean-Pierre	ENSIEG
ROSSIGNOL	Michel	ENSPG
ROYE	Daniel	ENSIEG
SABONNADIERE	Jean-Claude	ENSIEG
SAGUET	Pierre	ENSERG
SAUCIER	Gabriéle	ENSIMAG
SCHLENKER	Claire	ENSPG
SCHLENKER	Michel	ENSPG
SILVY	Jacques	UFR/PGP
SOHM	Jean-Claude	ENSEEG
SOLER	Jean-Louis	ENSIMAG
SOUQUET	Jean-Louis	ENSEEG
TICHKIEWITCH	Serge	ENSHMG
TROMPETTE	Philippe	ENSHMG
TRYSTRAM	Denis	ENSGI
VEILLON	Gérard	ENSIMAG
VERJUS	Jean-Pierre	ENSIMAG
VINCENT	Henri	ENSPG

SITUATION PARTICULIERE

PROFESSEURS D'UNIVERSITE

DETACHEMENT

BLOCH Daniel	ENSPG
BONNET Guy	ENSPG
BRECHET Yves	ENSEEG
CAILLERIE Denis	ENSHMG
GREVEN Héléne	CUEFA
LATOMBE Jean-Claude	ENSIMAG
PIERRARD Jean-Marie	ENSHMG

DIRECTEURS DE RECHERCHE CNRS

ABELLO	Louis
ALDEBERT	Pierre
ALEMANY	Antoine
ALLIBERT	Colette
ALLIBERT	Michel
ANSARA	Ibrahim
ARMAND	Michel
AUDIER	Marc
AUGOYARD	Jean-François
AVIGNON	Michel
BERNARD	Claude
BINDER	Gilbert
BLAISING	Jean-Jacques
BONNET	Roland
BORNARD	Guy
BOUCHERLE	Jean-Xavier
CAILLET	Marcel
CARRE	René
CHASSERY	Jean-Marc
CHATILLON	Christian
CIBERT	Joël
CLERMONT	Jean-Robert
COURTOIS	Bernard
CRIQUI	Patrick
CRISTOLOVEANU	Sorin
DAVID	René
DION	Jean-Michel
DOUSSIERE	Jacques
DRIOLE	Jean
DUCHET	Pierre
DUGARD	Luc
DURAND	Robert
ESCUDIER	Pierre
EUSTATHOPOULOS	Nicolas
FINON	Dominique
FRUCHARD	Robert
GARNIER	Marcel
GIROD	Jacques
GLANGEAUD	François
GUELIN	Pierre
HOPFINGER	Emil
JORRAND	Philippe
JOUD	Jean-Charles
KAMARINOS	Georges
KLEITZ	Michel
KOFMAN	Walter
LACROIX	Claudine
LANDAU	Ioan
LAULHERE	Jean-Pierre
LEGRAND	Michel
LEJEUNE	Gérard
LEPROVOST	Christian
MADAR	Roland
MARTIN	Jean-Marie
MERMET	Jean

PERSONNES AYANT OBTENU LE DIPLOME

DE DOCTEUR D'ETAT INPG

ABDEL-RAZEK	Adel
AKSAS	Haris
ALLA	Hassane
AMER	Ahmed
ANCELLE	Bernard
ANGENIEUX	Gilbert
ATMANI	Hamid
AYEDI	Hassine Feri
A.BADR	Osman
BACHIR	Aziz
BALANZAT	Emmanuel
BALTER	Roland
BARDEL	Robert
BARRAL	Gérard
BAUDON	Yves
BAUSSAND	Patrick
BEAUX	Jacques
BEGUINOT	Jean
BELLISSENT née FUNEZ	Marie-Claire
BELLON	Catherine
BEN RAIS	Abdejettah
BERGER-SABBATEL	Gilles
BERNACHE-ASSOLANT	Didier
BEROVAL	Abderrahmane
BERTHOD	Jacques
BILLARD	Dominique
BLANC épouse FOULETIER	Mireille
BOCHU	Bernard
BOJO	Gilles
BOKSENBAUM	Claude
BOLOPION	Alain
BONNARD	Bernard
BORRIONE	Dominique
BOUCHACOURT	Michel
BRION	Bernard
CAIRE	Jean-Pierre
CAMEL	Denis
CAPERAN	Philippe
CAPLAIN	Michel
CAPOLINO	Gérard
CASPI	Paul
CHAN-TUNG	Nam
CHASSANDE	Jean-Pierre
CHATAIN	Dominique
CHEHIKIAN	Alain
CHIRAMELLA	Yves
CHILO	Jean
CHUPIN	Jean-Claude
COLONNA	Jean-François
COMITI	Jacques
CORDET	Christian
COUDURIER	Lucien
COUTAZ	Jean-Louis

CREUTIN	Jean-Dominique
DAO	Trongtich
DARONDEAU	Philippe
DAVID	Bertrand
DE LA SEN	Manuel
DELACHAUME	Jean-Claude
DENAT	André
DESCHIZEAUX née CHERUY	Marie-Noëlle
DIJON	Jean
DOREMUS	Pierre
DUPEUX	Michel
EL ADHAM	Karim
EL OMAR	Fovaz
EL-HENNAWY	Adel
ETAY	Jacqueline
FABRE	Suzanne
FAURE-BONTE	Mireille
FAVIER	Denis
FAVIER	Jean-Jacques
FELIACHI	Movloud
FERYAL	Haj Hassan
FLANDRIN	Patrick
FOREST	Bernard
FORESTIER	Michel
FOSTER	Panayolis
FRANC	Jean-Pierre
GADELLE	Patrice
GARDAN	Yvon
GENIN	Jacques
GERVASON	Georges
GILORMINI	Pierre
GINOUX	Jean-Louis
GOUMIRI	Louis
GROC	Bernard
GROSJEAN	André
GUEDON	Jean-Yves
GUERIN	Jean-Claude
GUESSOUS	Anas
GUIBOUD-RIBAUD	Serge
HALBWACHS	Nicolas
HAMMOURI	Hassan
HEDEIROS SILIVEIR	Hamilton
HERAULT	Jeanny
HONER	Claude
HUECKEL	Tomasz
IGNAT	Michel
ILIADIS	Athananios
JANIN	Gérard
JERRAYA	Ahmed Amine
JUTTEN	Christian
KAHIL	Hassan
KHUONGQUANG	Dong
KILLIS	Andreas
KONE	Ali
LABEAU	Michel
LACAZE	Alain
LACROIX	Jean-Claude
LANG	Jean-Claude
LATHUILLERE	Chantal
LATY	Pierre

LAUGIER	Christian
LE CADRE	Jean-Pierre
LE GARDEVR	René
LE THIESSE	Jean-Claude
LEMAIGNAN	Clement
LEMUET	Daniel
LEVEQUE	Jean-Luc
LONDICHE	Henry
L'HERITIER	Philippe
MAGNIN	Thierry
MAISON	François
MAMWI	Abdullah
MANTEL épouse SIEBERT	Elisabeth
MARCON	Guy
MARTINEZ	Francis
MARTIN-GARIN	Lionel
MASSE	Dominique
MAZER	Emmanuel
MERCKEL	Gérard
MEUNIER	Jean
MILI	Ali
MOALLA	Mohamed
MODE	Jean-Michel
MONLLOR	Christian
MONTELLA	Claude
MORET	Frédéric
MRAYATI	Mohammed
M'SAAD	Mohammed
M'SIRDI	Kouider Nace
NEPOMIASTCHY	Pierre
NGUYEN	Trong Khoi
NGUYEN-XUAN-DANG	Michel
ORANIER	Bernard
ORTEGA MARTINEZ	Roméo
PAIDASSI	Serge
PASSERONE	Alberto
PEGON	Pierre
PIJOLAT	Christophe
POGGI	Yves
POIGNET	Jean-Claude
PONS	Michel
POU	Tong Eck
RAFINEJAD	Paiviz
RAGAIE	Harie Fikri
RAHAL	Salah
RAMA SEABRA SANTOS	Fernando
RAVAINE	Denis
RAZBAN-HAGHIGHI	Tchanguiz
RAZZOUK	Micham
REGAZZONI	Gilles
RIQUET	Jean-Pierre
ROBACH	Chantal
ROBERT	Yves
ROGEZ	Jacques
ROHMER	Jean
ROUSSEL	Claude
SAAD	Abdallah
SAAD	Youcef
SABRY	Mohamed Nabi
SALON	Marie-Christine

SAUBAT épouse MARCUS	Bernadette
SCHMITT	Jean-Hubert
SCHOELLKOPF	Jean-Pierre
SCHOLL	Michel
SCHOLL	Pierre-Claude
SCHOULER	Edmond
SCHWARTZ	Jean-Luc
SEGUIN	Jean
SIWY	Jacques
SKALLI	Abdellatif
SKALLI HOUSSEYNI	Abdelali
SOUCHON	Alain
SUETRY	Jean
TALLAJ	Nizar
TEDJAR	Farouk
TEDJINI	Smail
TEYSSANDIER	Francis
THEVENODFOSSE	Pascale
TMAR	Mohamed
TRIOLLIER	Michel
TUFFELIT	Denis
TZIRITAS	Georges
VALLIN	Didier
VELAZCO	Raoul
VERDILLON	André
VERMANDE	Alain
VIKTOROVITCH	Pierre
VITRANT	Guy
WEISS	François
YAZAMI	Rachid

REMERCIEMENTS

Je tiens à remercier

Monsieur Yves CHIARAMELLA, Directeur du Laboratoire de Génie Informatique, de l'honneur qu'il m'a fait en acceptant de présider le jury de cette thèse.

Monsieur Guy MAZARÉ, Directeur de l'ENSIMAG, de m'avoir accueilli dans son équipe et d'avoir assuré l'encadrement de cette thèse.

Messieurs Patrick GARDA, Chargé de recherches du CNRS à Institut d'Électronique Fondamentale, Université Paris Sud, et Bruno ROUZEYRE, Maître de conférences Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier d'avoir accepté d'être rapporteurs de cette thèse.

Monsieur Gérard MICHEL, Professeur à l'ENSIMAG, Directeur et conseiller technique de APTOR qui a accepté de faire partie du jury de cette thèse.

Monsieur Alain GUYOT, Maître de conférence à l'ENSIMAG, pour l'intérêt qu'il a porté à mon travail et pour ses nombreux conseils

Je remercie également toutes les personnes que j'ai côtoyées pendant la préparation de cette thèse. Grâce à leur aide et amabilité, j'ai pu mener à bien ce travail dans de bonnes conditions.

Résumé

Cette thèse présente la conception et la réalisation en VLSI d'un processeur programmable pour une nouvelle architecture MIMD massivement parallèle, intermédiaire entre la Connection Machine et les hypercubes de processeurs 32 bits. Elle est composée d'une grille 2D de cellules asynchrones communiquant par échanges de messages. Chaque cellule intègre une partie de traitement qui consiste en un petit microprocesseur 8 bits doté d'une mémoire (données et programme), et une partie de routage permettant l'acheminement des messages. A l'issue de l'étude des différents problèmes de communication dans les machines parallèles, nous proposons un routeur original utilisant le principe du *wormhole*, et permettant d'acheminer jusqu'à cinq messages en parallèle. Nous décrivons ensuite l'architecture de la partie de traitement, en partant de la définition du jeu d'instructions, du chemin de données et de la partie contrôle jusqu'à la conception au bas niveau. Un premier prototype d'un circuit VLSI de ce processeur a été réalisé sur silicium et a permis d'obtenir les mesures des surfaces et des performances.

Mots clefs : Architecture parallèle, communication, wormhole, jeu d'instructions, chemin de données, contrôle, circuit intégré VLSI.

Abstract

This thesis presents the VLSI design of a programmable processor for a new MIMD massively parallel architecture, halfway between the Connection Machine and hypercubes based on 32-bit processors. It consists of a 2D grid of asynchronous cells which communicate by message transfers. Each cell includes a processing unit based on an 8-bit microprocessor with a small amount of memory (data and code), and a routing part. After introducing various communication problems in parallel computers, we propose an original wormhole based routing system, able to forward up to five messages in parallel. We describe also the processing unit architecture starting from the instruction set definition and the data path and control part architecture down to the low level design. A first VLSI circuit prototype realisation of the processor has been performed and measures of silicon area and performances have been obtained.

Key words : Parallel architecture, communication, wormhole, instruction set, data path, control, VLSI integrated circuit.

A ma fille Salima
A mes parents et beaux parents
A ma femme
A mes frères et sœurs
Aux lecteurs

TABLES DES MATIÈRES

INTRODUCTION

Depuis l'invention des ordinateurs électroniques dans les années 1940, le calcul parallèl est devenu sans doute la technologie informatique la plus convoitée. Les courbes d'intégration, largement publiées dans la littérature, témoignent d'un progrès technologique considérable ces dernières années en termes de complexité de mémoires et de processeurs.

En plus du besoin d'une puissance de calcul éternellement requise, la formidable avancée technologique justifie les motivations pour la réflexion sur le parallélisme. On est continuellement surpris par l'annonce du jour au lendemain du nombre de transistors pouvant être intégrés dans un même circuit pour réaliser des mémoires larges et rapides ainsi que des processeurs très puissants.

De ce fait, on ne peut pas échapper aux questions suivantes :

- Que faire des circuits contenant jusqu'à 1 milliard de transistors ?

- Quelles sont les liaisons et les connexions qui s'imposent quand un processeur énorme adresse une large mémoire ?

- Quelles sont les mesures à prendre pour garantir un conditionnement adéquat ?

- Quels seront les efforts à fournir pour mettre au point les différents logiciels nécessaires pour pouvoir faire tourner et maintenir des systèmes aussi énormes que complexes ?

La communauté scientifique de différentes disciplines s'est entraînée dans cette situation où, d'une part, il faut rentabiliser les progrès en matière de technologie et bien exploiter le savoir faire acquis dans ce sens et d'autre part répondre aux exigences de quelques applications ne donnant pas entière satisfaction sur des machines utilisant le vieux concept de Von Neumann.

En fait, il y a plusieurs types d'applications qui présentent un parallélisme suffisant pour nourrir la volonté de réfléchir sur des systèmes où plusieurs processeurs élémentaires restent actifs simultanément, en fonctionnant. Le domaine le plus exploré est certainement celui du calcul numérique. On peut distinguer deux catégories :

a) Le calcul pour la recherche de solutions parallèles à des problèmes scientifiques comme la circulation atmosphérique globale, le médical (flux du sang dans le cœur), la dynamique des molécules, astronomie (évolution des galaxies), les ressources d'énergie, le militaire, l'intelligence artificiel etc...

b) Les applications d'ingénierie comme la CAO (les simulations de circuits, le placement, le routage, la synthèse logique ...), le traitement du signal, le traitement d'image, le traitement de la parole, l'exploration sismique, la conception et simulation des voitures, avions, etc...

D'autres domaines d'applications se présentent également à l'exploitation du parallélisme : le traitement symbolique a donné lieu à de nombreux travaux; c'est aussi le cas des applications de type gestion de données (machines base de données) et bien d'autres.

Il est réconfortant de voir que le parallélisme promet une exploration plus aisée de ces domaines divers. Il reste cependant à surmonter les problèmes qu'impose une telle approche : la communication, le mode de calcul, la programmation, le contrôle, la mémoire...

Sans doute l'une des problématiques fondamentales des machines parallèles et en particulier les machines massivement parallèles est la communication. Pour accomplir un travail, il est nécessaire aux processeurs de communiquer entre eux et d'avoir accès à une mémoire.

On imagine deux manières d'échange d'informations : par le partage d'une ressource (systèmes à mémoire commune) ou par l'envoi de la copie d'une information (machines à passage de messages et cellulaires).

Le partage d'une ressource n'est pas du tout adapté au parallélisme massif. Ceci est dû aux interconnexions qui s'imposent lors de la réalisation et la gestion des accès en présence de milliers, voire de millions de processeurs élémentaires.

La copie et l'envoi d'une information est la méthode la plus utilisée dans les machines hautement parallèles commercialisées. Ce mode de

communication a exhibé de bonnes performances. Ces machines utilisent un réseau de communication pour réaliser l'échange d'information entre les différents éléments définissant le grain de parallélisme.

Plus le nombre de grains est important, plus nombreux sont les processeurs élémentaires nécessaires pour les supporter et plus le rôle du réseau de communication devient important et déterminant. C'est pourquoi la communication dans les machines parallèles est sans doute la fonction qui, dans le passé, le présent et le futur, fait l'objet de profondes réflexions et études. Elle représente une composante très critique d'une machine parallèle et la caractéristique la plus sensible dans les mesures des performances.

En général, la mesure des performances d'un réseau de communication, que ce soit pour une machine parallèle ou un système de communication quelconque, est sa rapidité et sa capacité à délivrer et véhiculer à la place adéquate l'information nécessaire avec un coût raisonnable.

Quant aux modes de calcul, les efforts de recherches convergent vers le mode MIMD (Multiple Instruction Multiple Data) avec lequel la plupart des applications pouvant être explorées.

Au sein de notre laboratoire, notre équipe étudie depuis quelques années une architecture, appelée *réseau cellulaire,* qui appartient à la classe des machines massivement parallèles. Dès le début de ce projet, la simplicité de réalisation a été le facteur le plus préoccupant puisqu'il s'agit d'une interconnexion d'éléments de calcul (cellules) identiques en tout point de vue. Les cellules sont disposées en un plan et connectées avec les voisines immédiates selon un schéma en grille orthogonale. Le mode de calcul est asynchrone et l'architecture est du type MIMD. La communication se fait par l'envoi de messages et d'une manière asynchrone.

Les premiers réseaux cellulaires étaient des machines dédiées à des applications précises (simulation logique, placement par recuit simulé, reconstruction d'image, réseaux de neuro-mimétiques). La réalisation d'un certain nombre de circuits VLSI a permit d'établir les conclusions suivantes :

- Coût de développement et réalisation important,
- Manque de généralité,
- Complexité (entre 10 000 et 20 000 transistors) comparable à celle d'un microprocesseur.

Ces trois points nous ont poussés à envisager l'étude et la réalisation d'un réseau cellulaire dont les processeurs élémentaires sont programmables.

Le travail effectué s'insère dans le cadre d'un travail d'équipe :

- Eric Payan, en plus de sa participation à la définition fonctionnelle du processeur, s'est penché sur l'implémentation du langage data-flow LUSTRE sur le réseau [Pay91].

- Pascal Rubini a étudié de près la définition fonctionnelle de l'architecture du réseau et s'est intéressé à l'écriture d'algorithmes adaptés au réseau cellulaire [Rub92].

- Youssef Latrous (thèse en préparation) étudie la gestion dynamique d'un réseau de cellules pour déterminer le modèle d'exécution correspondant à des langages d'acteurs.

- Marc Robinchon et Frédéric Boiteux [Boi91] se sont intéressés, lors de leur stage troisième année d'école d'ingénieur, à la réalisation matérielle de l'interface hôte/réseau et au chargement des programmes.

- Olivier Jacquiot a travaillé sur le problème de la simulation logique à échéancier répartie [Jac91].

Dans le cadre de la définition fonctionnelle de cette architecture qui se veut généraliste, mon travail se porte en une première phase sur la contribution à la définition du jeu d'instructions, à la définition de l'architecture de l'unité de traitement et le mécanisme de communication du point de vue matériel.

La seconde phase se porte sur les choix de conception jusqu'au bas niveau de la brique élémentaire qui est le processeur aussi bien au niveau de l'unité de traitement qu'au niveau du mécanisme de communication.

La dernière phase se résume en la réalisation sur silicium de la cellule.

Cette thèse s'inscrit donc sous le titre des processeurs massivement parallèles qui pourrait se résumer par la question suivante : "est-il possible avec les technologies actuelles de réaliser des circuits intégrant plusieurs processeurs programmables 8 bits pour en faire une machine massivement parallèle dont la

puissance de calcul serait extensible à l'infini juste par duplication de ces circuits ?"

Cette thèse étant en microélectronique et elle a été effectuée au sein d'un laboratoire de génie informatique, nous avons jugé utile, avant d'aborder la conception et la réalisation d'un processeur massivement parallèle en partant de sa définition fonctionnelle jusqu'à son implantation sur silicium, d'explorer les différentes composantes des machines parallèles. Bien que ces composantes soient interdépendantes, nous avons préféré structurer ce manuscrit en quatre chapitres. Hormis le dernier, les trois premiers explorent chacun l'une des composantes importantes (mode de calcul, réseaux d'interconnexion et architecture de l'unité de calcul) sous la forme : présentation de ce qui est existant, proposition et finalement implémentation.

Le chapitre I présente un bref aperçu sur les notions des architectures parallèles et situe notre architecture par rapport aux autres architectures fortement parallèles.

Dans le chapitre II, les différentes approches, types et les différents problèmes des réseaux d'interconnexion dans les architectures parallèles seront explorés. A l'issue de cette exploration, on présentera le routeur proposé, les algorithmes de communication ainsi que l'implémentation au niveau logique avec les choix de conception. On clôturera ce chapitre par les perspectives d'introduction de l'optique dans les machines parallèles.

Le chapitre III abordera l'unité de traitement en ce qui concerne l'élaboration du jeu d'instructions, la définition et proposition du chemin de données ainsi que l'unité de contrôle à l'issue d'une brève présentation des différents types de ces derniers. On présentera également l'implémentation logique, les choix de conception et les compromis délicats à effectuer aux différents niveaux hiérarchiques.

Le chapitre IV explicite la phase de réalisation du circuit en mettant l'accent sur la méthodologie adoptée pour la réalisation et la mise au point des différents blocs fonctionnels concernant la saisie des schématiques, les simulations logiques et l'implantation. On terminera ce chapitre par un bilan des complexités des différents blocs fonctionnels de la cellule.

On trouvera à la fin de cette thèse les annexes jugés utile à faire figurer afin de mieux évaluer le volume du travail effectué. Ces annexes incluent essentiellement le jeu d'instructions détaillé, la description détaillée de l'exécution de chaque instruction, une étude sur les différents types d'UALs, les graphes de contrôle, les listings des fichiers de test et enfin les copies d'écran des schématiques, des simulations et implantation sur silicium.

CHAPITRE I

LES ARCHITECTURES PARALLÈLES

1.1 TYPES DES ARCHITECTURES PARALLÈLES

La diversité des applications, nécessitant une puissance de calcul importante, introduit forcément une variété dans le parallélisme lui-même. Cependant, on peut définir différents types de parallélisme en fonction de ce que peut offrir une application comme source susceptible d'être exploitée pour ce parallélisme.

Il existe trois sources de parallélisme dans la majorité des applications :

• Le parallélisme de contrôle émane des applications qui contiennent un parallélisme intrinsèque. Naturellement, il existe des actions ou des tâches qui sont plus ou moins indépendantes pouvant être réalisées simultanément.

• Le parallélisme de données représente la source la plus importante des trois. Il découle des applications dont les structures de données sont très régulières (vecteurs, matrices ...) où la même action est répétée sur chaque élément de la structure.

• Le parallélisme de flux provient des applications fonctionnant selon le mode travail à la chaîne. Ces applications présentent un flux de données généralement similaires et nécessitent que des opérations soient effectuées en cascade. Ce mode de fonctionnement est aussi connu sous le nom : *pipe line.*

Également, la multiplicité des applications et de ces différentes sources ont donné naissance à deux approches du parallélisme : la pensée concurrente et la pensée parallèle. La pensée concurrente vise surtout l'exploitation du parallélisme de contrôle et la deuxième pensée concerne les applications qui par les données et leurs structures, apparaissent comme une bonne source pour le parallélisme.

Dans ces divers domaines, on retrouve également plusieurs autres termes et options qui caractérisent les différents types ou classes d'architectures des machines parallèles. Parmi les facteurs les plus dominants dans cette caractérisation, on cite : la mémoire, la granularité, la nature et les moyens d'interconnexion, les modes et les mécanismes de communication.

1.1.1 LA CLASSIFICATION DE FLYNN

A ce stade, nul ne peut se permettre d'omettre de mentionner la classification de Flynn qui fût établie en 1972 [Fly72]. Cette classification tient compte surtout du séquencement des données et des instructions dans les processeurs. Par contre la mémoire, la granularité, et le type de communication ne font pas effet sur cette classification. Malgré quelques tentatives pour présenter une meilleur classification, on reste toujours proche de celle-ci en rajoutant seulement une autre classe qui concerne le facteur de la mémoire commune/distribuée pour la classe MIMD.

1.1.1.1 LES ARCHITECTURES SISD

Le calcul SISD (Single Instruction Single Data) est le modèle traditionnel à processeur unique (machine de Von Neumann). Une application est lancée sur l'unique processeur sous un contrôle séquentiel. Ce traitement consiste à exécuter les instructions l'une après l'autre où chacune traite une donnée.

1.1.1.2 LES MACHINES MISD

Le traitement MISD (Multiple Instruction Single Data) qui est appelé aussi "pipeline" consiste à organiser au sein d'une même unité centrale plusieurs unités travaillant à la chaîne, exécutant chacune une partie de l'instruction. A un instant donné, les unités actives traitent des instructions différentes sur une même donnée. Puisque le traitement s'effectue à la chaîne, le nombre d'opérateurs fonctionnant simultanément varie au cours du temps et souvent d'une manière décroissante. Ce qui limite les performances de cette technique pour certaines applications.

1.1.1.3 LES MACHINES SIMD

La troisième combinaison de l'ensemble "multiple, unique" est l'organisation en SIMD (Single Instruction Multiple Data). Cette technique de construction des machines massivement parallèles consiste à faire exécuter une même suite d'instructions simultanément par un très grand nombre de processeurs munis chacun d'une mémoire locale.

L'architecture inclut alors une partie contrôle unique qui obtient des

instructions de la mémoire de programme. L'unité de contrôle se charge du décodage des instructions et de les diffuser sous forme de microcommandes et adresses des opérandes à tous les processeurs. Ces derniers sont des opérateurs très simples. Ils peuvent disposer tout au plus d'un registre de base et d'un bit d'inhibition qui peuvent être positionnés localement pour adresser la mémoire locale et d'exécuter sélectivement les microcommandes diffusées.

Les structures SIMD permettent d'exploiter le parallélisme des applications bien régulières pouvant employer un grand nombre de processeurs pour exécuter simultanément sous un même contrôle les mêmes opérations. Elles sont également bien adaptées pour manipuler les matrices pleines de grandes tailles.

1.1.1.4 LES MACHINES MIMD

La dernière combinaison est celle de la classe MIMD (Multiple Instruction Multiple Data). Cette architecture consiste à utiliser plusieurs processeurs pour exécuter des instructions différentes sur des données différentes. De plus, ces processeurs travaillent simultanément d'une manière autonome comme s'ils étaient des machines séparées sans contrôle central. Ils disposent donc, chacun d'une partie contrôle et exécutent un programme qui leur est propre. Le déroulement des différents programmes se fait en parallèle et d'une manière asynchrone. Cette structure de machine qui a un caractère plus général permet de traiter des applications plus variées, donc, plus nombreuses.

Pour la plupart de ces applications, lors de leurs déroulements sur de telles architectures, une coopération entre processeurs s'impose. La manière la plus usuelle d'assurer une collaboration est l'utilisation de la mémoire. Les premières structures MIMD utilisaient une mémoire centrale commune à tous les processeurs où ils trouvent leurs programmes, leurs données et leurs moyens de communication. Plusieurs solutions ont été utilisées et envisagées (mémoires multibancs, bus multiples et larges, ...) afin de favoriser et faciliter le partage de la mémoire. L'accès à la mémoire est resté un problème délicat à résoudre.

Cette organisation présente l'inconvénient d'aboutir à un goulot d'étranglement qui ne permet pas d'envisager un parallélisme massif.

1.1.2 LA CLASSE MIMD À MÉMOIRE DISTRIBUÉE

Une manière de surmonter ce problème est de doter chaque processeur d'une mémoire locale. Bien que cette approche permet de résoudre les problèmes d'embouteillage au niveau de la mémoire commune et de considérer un parallélisme massif, mais d'autres problèmes surgissent : le système de communication, ainsi que la programmation de l'ensemble des processeurs. En d'autres termes :

• Comment les processeurs vont-ils pouvoir échanger toutes les informations nécessaires à leur coopération ?

• Comment va-t-on charger les programmes de chaque processeur et quelle tâche va-t-elle être confiée à quel processeur ?

Pour de telles architectures, il faut envisager un réseau d'interconnexion utilisant le passage de messages qui doit être suffisamment efficace et rapide pour profiter de l'avantage de l'exécution parallèle [Ath88].

1.2 TOPOLOGIE DU RÉSEAU D'INTERCONNEXION

Les réseaux d'interconnexion se divisent en trois classes : hypercubes, grilles et réseaux reconfigurables. On se contente ici juste de mettre l'accent sur la délicatesse du choix parmi ces trois classes à faire vu la multitude de facteurs qui doivent être considérés. Le chapitre 2 aborde de plus en détails les réseaux d'interconnexion.

1.3 PARAMÈTRES DU PARALLÉLISME

Les machines parallèles diffèrent et se caractérisent par plusieurs paramètres où chacun a sa part d'influence dans les performances de la machine.

1.3.1 GRANULARITÉS

La granularité est une mesure du nombre d'instructions exécutées. Dans le contexte du parallélisme, l'interval de granularité varie d'une simple addition jusqu'au nombre de 1000 à 10 000 instructions [Ger89].

1.3.2 LA MÉMOIRE

La mémoire peut être soit partagée soit distribuée. L'expérience a montré qu'avec une mémoire partagée par plusieurs processeurs, on aboutit très rapidement à un goulot d'étranglement quand les accès mémoire sont intenses. Par contre, les mémoires distribuées/locales offrent la possibilité de considérer un haut niveau de parallélisme à travers un réseau d'interconnexion de milliers de processeurs. Ceci nécessite bien sûr différents types de programmation pour adopter un concept de communication à passage de messages.

D'un autre coté, la taille de la mémoire locale nécessite d'établir des compromis avec d'autres paramètres comme la granularité et les types de données. Tout dépend de la mesure de la granularité adoptée : la taille de la mémoire doit être suffisamment large pour que le processeur effectue convenablement sa tâche.

1.3.3 SYNCHRONISME / ASYNCHRONISME

L'exécution des programmes parallèles nécessite une coordination entre les tâches interdépendantes. Il y a deux méthodes générales pour accomplir et réaliser cette coordination dans les machines parallèles : la méthode Synchrone et la méthode Asynchrone.

1) La première approche consiste à concevoir un dispositif, au niveau physique de la machine, forçant toutes les opérations ou tâches à être effectuées en même temps de façon que celles-ci soient totalement indépendantes l'une de l'autre.

2) La deuxième approche de coordination, appelée Asynchrone, est la forme la plus générale du parallélisme. Elle est basée sur l'utilisation des mécanismes de contrôle de flot de données afin d'éviter tout déséquilibre entre les processeurs émetteurs et les processeurs récepteurs dû au découplage temporel. Les processeurs exécutent leur tâches sans se préoccuper de la synchronisation globale.

De ce paramètre découlent deux classes de machines parallèles comme le montre la figure 1.1. On peut classer également les architectures en fonction du contrôle et surtout de la manière dont les processeurs élémentaires échangent des messages entre eux.

Souvent, le paramètre du synchronisme dans un système implique le

partage du signal d'horloge par les différents acteurs de ce système alors qu'on peut avoir un système asynchrone où ses acteurs partagent le même signal d'horloge (tout en négligeant volontairement le problème de la puissance du signal). L'échange entre ces acteurs peut s'effectuer selon un protocole piloté par des signaux différents du signal d'horloge.

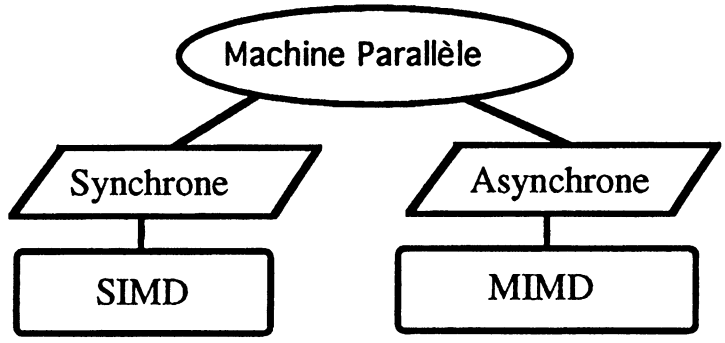


Figure 1.1 : Classification en fonction du synchronisme

1.4 LES MACHINES MASSIVEMENT PARALLÈLES

Les machines parallèles sont, elles aussi, regroupées en plusieurs classes surtout en fonction du parallélisme exploité, de la mémoire, de la structure et de la manière de coopération entre les processeurs. Les premières machines parallèles, appelées supercalculateurs, utilisaient une mémoire centrale commune. Cette classe comprend les multiprocesseurs fonctionnant en mode MIMD et les machines vectorielles qui utilisent le mode SIMD (CRAY).

1.4.1 MACHINES À PASSAGE DE MESSAGES

Vient ensuite, la classe des machines à passage de messages. Ces machines appartiennent à la classe des machines du type MIMD. Elles sont composées d'un ensemble de processeurs élémentaires qui communiquent entre eux via un réseau d'interconnexion. Chaque processeur élémentaire inclut une unité de calcul, une mémoire locale et des dispositifs constituant une interface avec le réseau d'interconnexion (iPSC/1, iPSC/2 d'Intel et le Transputer d'Inmos).

La plupart des machines à passage de messages actuellement commercialisées ont une granularité moyenne. Autrement dit, la taille de la mémoire locale est moyenne.

L'objectif principal est de construire des machines beaucoup plus rapides

intégrant un grand nombre processeurs élémentaires. L'abaissement de la granularité, la technologie des réseaux de communication, le placement des processus (statique/dynamique) et la programmation constituent les facteurs de préoccupations les plus importants dans le développement de ces types de machines.

Ces points font actuellement l'objet de travaux de recherche dans les laboratoires universitaires et industriels afin d'ouvrir des perspectives pour considérer un parallélisme beaucoup plus important.

1.4.2 MACHINES CELLULAIRES

Les machines cellulaires font partie des machines de type SIMD à mémoire distribuée. Elles exploitent le parallélisme des données et utilisent le principe de base qui consiste à affecter un processeur élémentaire par donnée. Selon le mode de traitement des données, on distingue deux types de machines cellulaires :

- les machines bit serie (GAPP, MPP) où la longueur du chemin des données des PE est d'un bit.
- les machines bit parallèle où la taille du chemin des données des processeurs élémentaires est de 32 ou 64 bits flottants.

1.4.3 MACHINES SYSTOLIQUES

Les machines systoliques sont des machines cellulaires qui à la fois exploitent le parallélisme du flux et celui des données. Elles fonctionnent sous le mode SIMD et tirent profit du concept du pipe-line. Dans les machines systoliques, les données sont introduites et traitées sous un contrôle synchrone et centralisé.

1.4.4 QUELQUES EXEMPLES DE MACHINES PARALLÈLES

1.4.4.1 LA CONNECTION MACHINE

La connection machine fait partie des architectures SIMD. La CM2 est constituée de 16 modules contenant 32 processeurs élémentaires 1 bit chacun. Chaque module est composé de deux circuits regroupant 16 processeurs élémentaires, une unité de communication, une mémoire locale de 32x256 kbits et une unité de calcul flottante (figure 1.2). Les 32 modules sont reliés à

l'ordinateur hôte par un séquenceur à travers un bus d'adresses et un bus d'instructions. Les circuits de 16 (PE) sont reliés entre eux à travers des unités de communication suivant un hypercube (de degré 12 si la configuration maximale est utilisée). Au niveau d'un même circuit de 16 (PE), les processeurs sont pleinement connectés entre eux par l'intermédiaire d'un réseau d'interconnexion complet (butterfly à 4 niveaux).

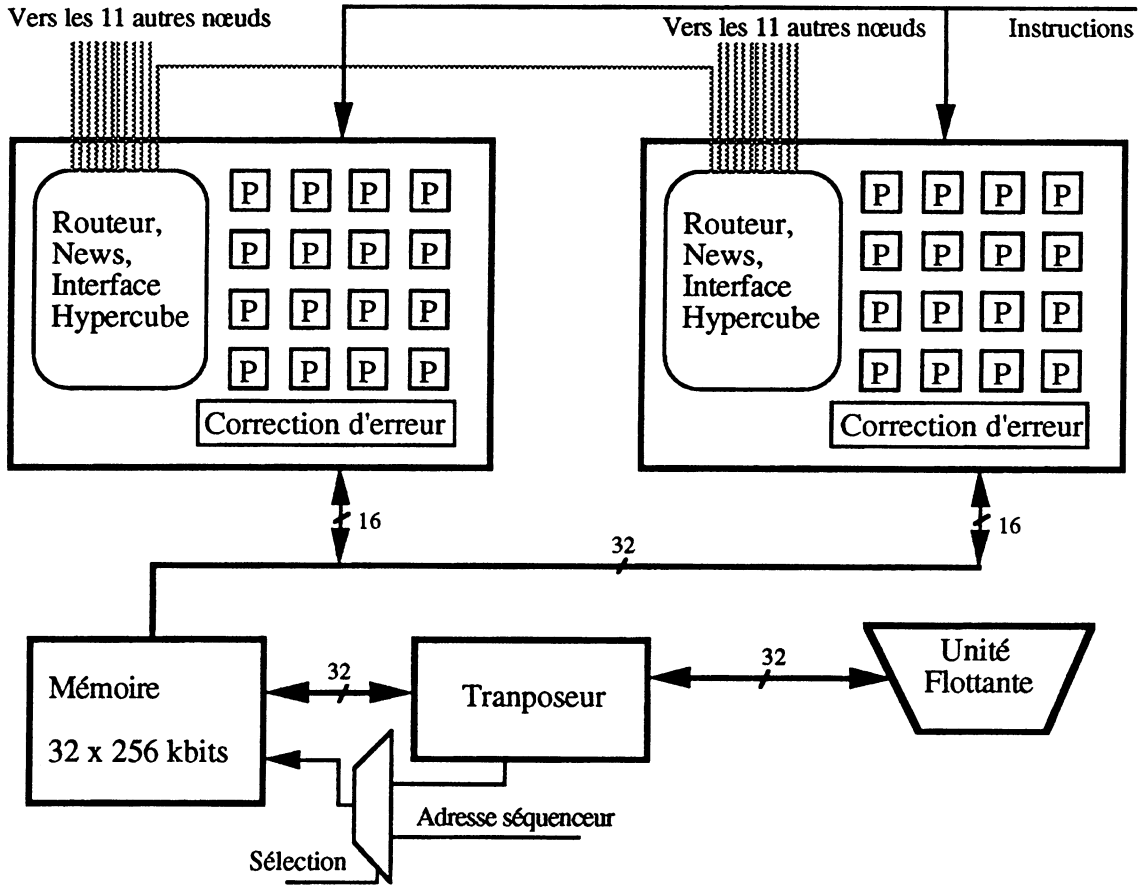


Figure 1.2 : Organisation d'un module

L'unité flottante associée à chaque module afin d'accélérer les calculs, opère sous le format standard IEEE (23 bits de mantisse et 8 bits d'exposant et le bit de signe). Une opération portant sur des nombres réels est effectuée séquentiellement par un mot de 32 bits au lieu d'être faite bit à bit en parallèle sur chacun des 32 processeurs. La nécessité de l'utilisation d'un transposeur provient du fait que l'unité flottante a besoin des 32 bits venant d'un même processeur élémentaire alors que la mémoire n'en fournit qu'un seul à chaque appel. Le transposeur est fait d'une matrice de 32 x 32 bits qui sert à stocker et arranger les opérandes de l'unité flottante.

La partie la plus intéressante dans la Connection Machine est l'algorithme de routage de message pour une topologie en hypercube. Le but du routeur de message dans la Connection Machine est de permettre une flexibilité de communication entre les processeurs élémentaires. Le réseau d'interconnexion est dynamique et la communication est du type passage de message. Si chaque processeur élémentaire de la Connection machine peut échanger des informations avec n'importe quel autre, deux processeurs quelconques ne sont pas forcément physiquement connectés.

La souplesse du réseau d'interconnexion permet l'implantation d'une variété d'applications. En plus des applications SIMD classiques à structure régulière, il est possible de faire tourner la machine pour des applications à structure irrégulière :

- les réseaux sémantiques où un processeur-cellule est associé à un concept
- les circuits VLSI où un processeur-cellule est associé à un transistor.

1.4.4.2 LE COSMIC CUBE DE CALTECH ET LES iPSC

La machine "Cosmic cube" de Caltech réalisée en 1983 par C. Seitz fut l'une des premières machines du type MIMD à passage de messages. Elle est composée de 6 à 64 processeurs élémentaires disposés en hypercube et pilotés par un ordinateur hôte. Chaque processeur élémentaire a pour base un microprocesseur (Intel 8086) entouré de son coprocesseur flottant (Intel 8087), une mémoire locale de 128 Ko, une ROM de 8 Ko et un gestionnaire de 6 canaux.

Chaque processeur élémentaire, en plus de la tâche qui lui a été confiée, possède en mémoire RAM une copie du système chargée lors de la phase d'initialisation. La communication se faisant par passage de messages, le système gère seulement les échanges entre processeurs voisins. La communication entre processeurs non directement reliés, est effectuée en utilisant le mode d'acheminement par commutation de messages "store and forward", ce qui résulte en des délais de communication importants.

1.5 QUELQUES PROCESSEURS MASSIVEMENT PARALLÈLES

Un processeur parallèle est la brique élémentaire pour construire un ordinateur parallèle. Ce dernier doit contenir plusieurs processeurs destinés à être connectés à travers un réseau d'interconnexion pour pouvoir coopérer et travailler simultanément. Il y a deux caractéristiques dominantes pour un processeur parallèle : les unités de traitement et l'unité de communication. C'est avec cette dernière qu'il prend place dans le réseau d'interconnexion.

La manière et la topologie avec laquelle les processeurs sont connectés, le mode et le mécanisme de communication est sans doute l'aspect le plus important des deux.

Il existe plusieurs processeurs utilisés dans les machines massivement parallèles sur le marché, certains n'ont pas encore franchi les frontières des laboratoires. Ils peuvent être classés en deux catégories du point de vue de la taille. Les processeurs monobit et les processeur plus puissant de 16 bits et plus. On se contentera de donner des exemples de quelques uns seulement pour illustrer les deux catégories et de mettre en évidence la position intermédiaire de notre processeur (RAP).

1.5.1 LE TRANSPUTER

Le transputer de base est un microprocesseur monochip intégrant dans un seul boîtier une CPU de 32 bits, une mémoire locale et des canaux de communication permettant une connexion point à point avec d'autres transputers (figure 1.3).

L'architecture de la CPU est du type RISC (Reduced Instruction Set Computers) et le contrôle est réalisé par microcode. L'une des particularités du transputer est sa capacité à supporter plusieurs processus concurrents. L'ordonnancement des processus étant entièrement microcodé, permet de réaliser une commutation de contexte d'une manière très rapide. Les progrès dans le domaine de la technologie et surtout en VLSI ont permis aux générations successives des transputers de subir des améliorations, comme l'intégration d'une unité de calcul en virgule flottante (T800) et l'augmentation de la taille de mémoire locale de 4 Ko à 16 Ko (T9000).

Les transputers peuvent être connectés suivant une topologie quelconque tant que le degré de la connectivité des nœuds ne dépasse pas quatre. Le langage OCCAM est le langage naturel de programmation des transputers. Il permet de spécifier explicitement le parallélisme, la séquentialité ainsi que le multiplexage

entre les actions. La sélection est basée soit sur un critère temporel comme une lecture sur un canal d'entrée ou une attente sur horloge, soit sur une condition Booléenne. Un choix aléatoire est alors entrepris si les deux conditions sont satisfaites.

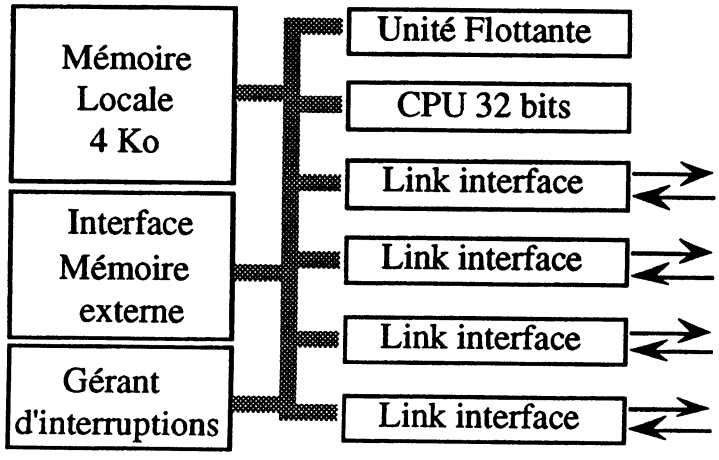


Figure 1.3 : Transputer (T800)

1.5.2 LE PROCESSEUR GAPP

Le processeur GAPP *(Geometrical Arithmetic parallel processor)* est un circuit VLSI qui fait partie des processeurs SIMD. Le GAPP contenant 72 processeurs élémentaires organisés en une matrice de 12 x 6 reliés entre eux selon une grille à deux dimensions. Des versions plus récentes du circuit en une technologie plus avancées contiennent jusqu'à 4096 (PE).

Chaque processeur élémentaire du circuit GAPP, inclue une UAL 1 bit, 1 registre de 4 bits, une mémoire RAM de 128 bits. De plus, il intègre de nombreux multiplexeurs qui permettent d'aiguiller les données en mode bit série sur les chemins internes ainsi que sur les ports d'entrée/sortie vers les quatre voisins (figure 1.4).

Les opérations arithmétiques sont effectuées en série. La puissance élementaire est donc très faible mais le processeur GAPP trouve sa puissance par le grand nombre de cellules utilisées dans une application. Les circuits GAPP peuvent être aussi organisés en une grille à deux dimensions pour atteindre des matrices plus importantes.

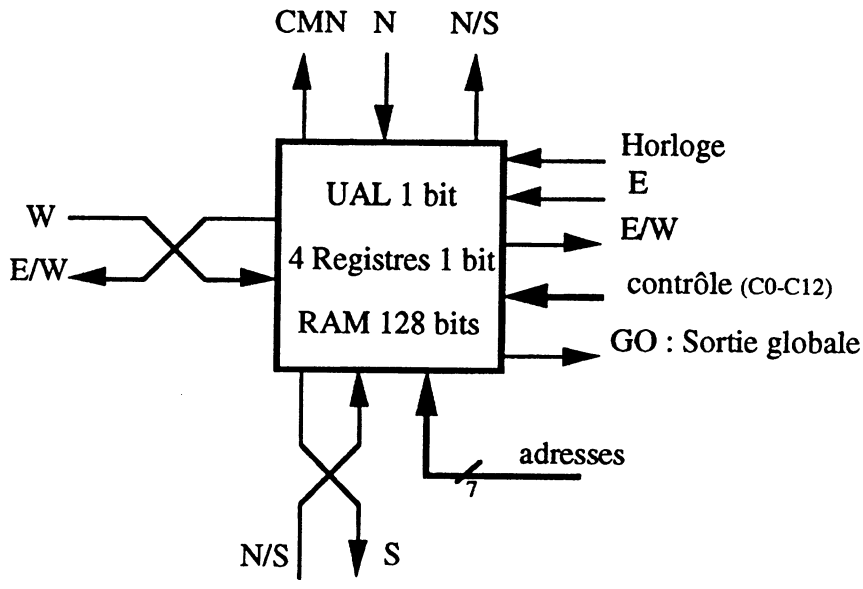


Figure 1.4 : Processeur élémentaire d'un circuit GAPP

1.5.3 LES CHIPS DE LA CM2

Chaque processeur de la "connection machine" est relativement simple à concevoir et réaliser. La puissance de la machine est due principalement à l'activité simultanée d'un grand nombre de ses processeurs. Ces derniers sont des processeurs bit série acceptant trois bits comme entrées, deux de la mémoire associée et un bit venant d'un ensemble de drapeaux ("flags") faisant partie du processeur lui-même (figure 1.5).

Au cours d'un cycle élémentaire, il peut mettre à jour l'un des deux bits de la mémoire. Il peut aussi lire l'un des quatre flags et en mettre un autre à jour (éventuellement le même).

Le processeur est capable d'effectuer 2^{16} opérations. Il peut être vu par le hôte comme une mémoire morte dans laquelle les opérations sont tabulées.

La mémorisation du résultat est conditionnée par l'état du bit appelé "context". Si ce dernier est à l'état faux, alors l'instruction n'a aucun effet. Ce mécanisme sert à ce que certaines instructions, durant un cycle, soient exécutées par certains processeurs et pas d'autres.

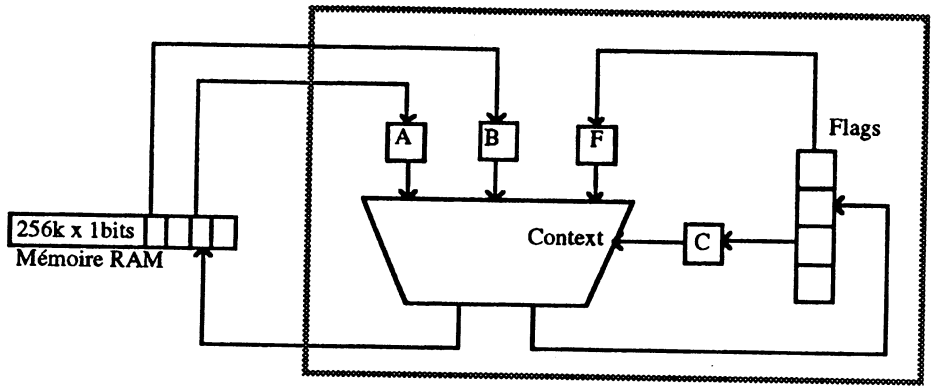


Figure 1.5 : Processeur élementaire de la connection machine

1.6 LA MACHINE CELLULAIRE RAP

Mettant en profit l'expérience et les enseignements acquis sur les machines à base de transputers et la Connection Machine notre approche est de faire des chips contenant plusieurs processeurs programmables dotés d'une mémoire locale soit en structure réseaux soit réparties en systèmes distribués. Ceci découle du fait que le transputer est un processeur puissant permettant de faire du calcul MIMD et de la multiprammation nécessitant une mémoire locale externe importante. Tout le système est donc délicat à programmer et présente des difficultés à répéter plusieurs fois le processeur élémentaire.

En ce qui concerne la connection Machine, malgré qu'elle offre un parallélisme massif important, elle passe plus de temps à gérer les unités du calcul flottant que les processeurs élémentaires. Bien que les chip de la CM 5 intègrent l'unité du calcul flottant, elle reste efficace que sur les applications à parallélisme régulier (calcul SIMD).

Pour cela, notre équipe s'intéresse depuis quelques temps à l'étude d'une architecture massivement parallèle reposant sur une architecture MIMD à mémoire distribuée dite CELLULAIRE intermédiaire entre la connection Machine et les réseaux à base de Transputers. La machine est composée d'un nombre important de petits processeurs fonctionnant d'une manière asynchrone et communiquant à travers un réseau d'interconnexion en grille bidimensionnelle (d'où le nom RAP : réseau asynchrone de processeurs programmables). Chaque processeur élémentaire comporte donc une partie traitement et une autre partie dite de routage.

La complexité d'un (PE) développé actuellement est aux alentours de 25 000 transistors (sans la mémoire). Ce qui devrait permettre la réalisation de circuits de 8 x 8 cellules avec les technologies industrielles actuelles et pourquoi

pas plus, si l'évolution dans le domaine d'intégration est prometteuse.

Au départ, l'équipe s'intéressait à des problèmes spécifiques comme le placement dans le domaine de la CAO [Cor88], la simulation logique [Cor87], la reconstruction d'image en tomographie [Lat89], le calcul neuronal [Fau90]. Seule, la partie routage était commune à ces différentes machines. Elles utilisaient le même mécanisme de communication et se différentiaient seulement par la partie traitement.

Une autre approche a été envisagée étant donnée que le développement des circuits spécifiques pour chaque application s'est avéré un travail long, délicat et coûteux. Beaucoup d'applications nécessitent une première phase d'étude très longue et une autre phase de réalisation encore plus importante en temps et en moyens.

Il est donc plus avantageux de profiter des progrès de la technologie dans le domaine de la VLSI pour se pencher sur un circuit programmable utilisable par des applications diverses. La partie traitement spécifique est donc remplacée par un petit processeur programmable. Le réseau devient alors un réseau cellulaire à usage général. La description de l'architecture est montrée dans la figure 1.6.

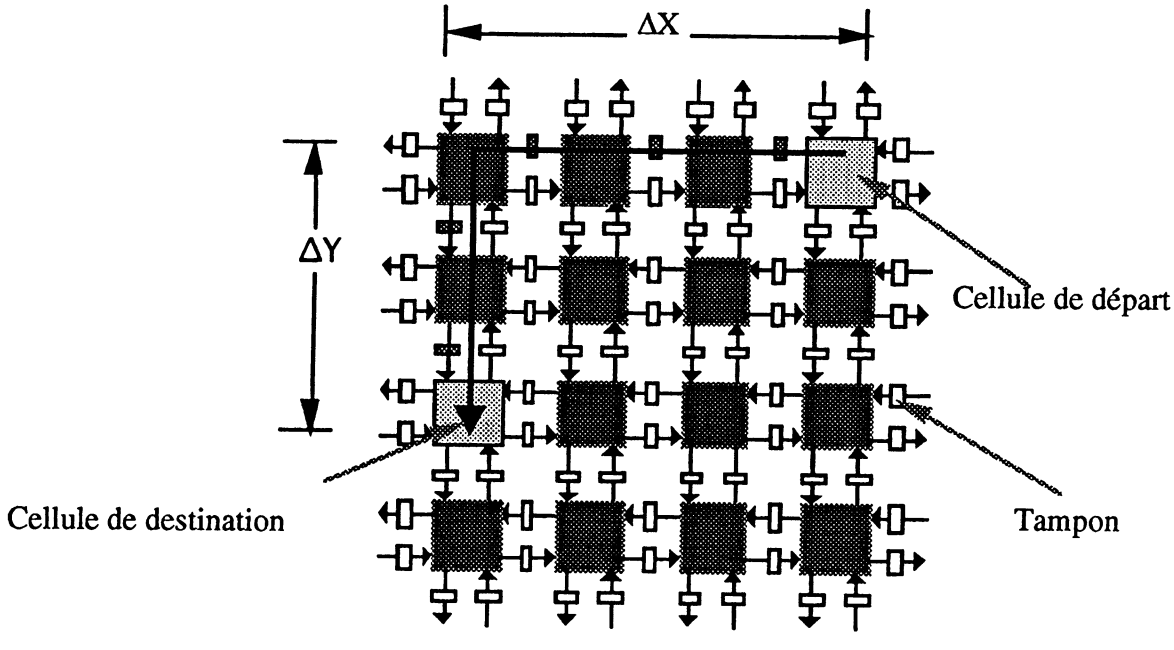


Figure 1.6 : Le réseau cellulaire et système de communication

Elle consiste en une topologie plane et un réseau d'interconnexion des cellules en grille où chaque processeur est connecté à ses quatre voisins. La

connectivité des cellules reste constante lorsque la taille du réseau augmente. Ce qui permet d'avoir des circuiteries de communication fixes et invariables. A un niveau macroscopique, puisque les technologies traditionnelles (VLSI, circuit imprimés) sont bidimensionnelles, la grille permet une connexion par simple juxtaposition des boîtiers.

En ce qui concerne la communication, les cellules partagent avec leurs quatre voisines des buffers tampons qui servent comme une boîte aux lettres pour échanger des informations. Bien sûr, la capacité d'une cellule à communiquer avec seulement ses quatre voisines n'est pas suffisante pour beaucoup d'applications. Le mécanisme d'acheminement des messages entre cellules lointaines repose sur l'utilisation des adresses relatives.

Lorsqu'une cellule veut communiquer avec une autre cellule différente de ses voisines, elle doit faire transiter son message par des cellules intermédiaires. Le message transite d'une cellule à une autre le long des arcs de la grille selon une stratégie simple, d'abord horizontalement, puis verticalement. Le message doit contenir donc non seulement les informations à transmettre mais aussi l'adresse relative de la cellule de destination. A chaque transition, l'adresse relative est mise à jour jusqu'à ce qu'elle devient nulle.

L'acheminement des messages au niveau d'une cellule est pris en charge par une circuiterie complètement autonome appelée aiguilleur. Une cellule est donc constituée de deux parties distinctes et autonomes travaillant en parallèle (figure 1.7).

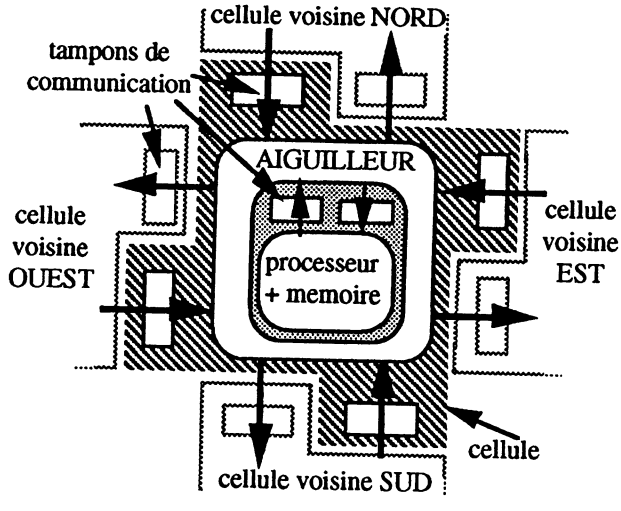


Figure 1.7: Structure de la cellule

La machine cellulaire est constituée donc d'un réseau cellulaire et d'un

ordinateur hôte relié à l'aide d'une interface. Le réseau cellulaire repose donc sur l'utilisation de plusieurs de ces cellules en une grille 2D et l'ordinateur hôte a pour rôle de préparer les données et les différents programmes des processeurs, de les faire injecter dans le réseau à travers l'interface, de piloter la phase d'initialisation et de récupérer les résultats afin d'être exploités.

1.7 CONCLUSION

L'approche d'une machine MIMD intermédiaire entre la "Connection Machine" et une machine à base de "Transputer" apparait comme une alternative intéressante. La programmation reste l'une des difficultés essentielles de ce type d'architectures. Ceci ouvre la voie au developpement des outils de programmation spécifiques. Il est difficile d'envisager une méthode de compilation donnant un programme exécutable aussi dense et rapide. Ce problème est rendu encore plus sensible dans le cas des applications irrégulières où une programmation manuelle est impossible. Le langage Lustre présente une solution intéressante pour certaines classes d'applications [Pay91].

En ce qui concerne l'accès au réseau, il se fait à travers quelques cellules qui se trouvent sur le bord du réseau appelées cellules *d'injection* . Le choix du nombre et de la position des cellules d'injection parmi les cellules de bord constitue à lui seul une étude qui mérite d'être effectuée afin de trouver le nombre et la position optimale pour ne pas trop compliquer l'interface. D'autre part pour arriver à un débit d'échange entre le réseau et l'ordinateur assez élevé qui justifie l'utilisation d'un réseau cellulaire et notamment une architecture massivement parallèle.

CHAPITRE 2

DÉFINITION ET IMPLÉMENTATION DE LA FONCTION DE ROUTAGE

Les réflexions sur la communication dans les ordinateurs parallèles ont abouti à un ensemble assez varié de réseaux d'interconnexion. Cet ensemble est la combinaison des quelques approches entreprises par les architectes durant la construction des machines parallèles en ce qui concerne la communication aussi bien inter-processeurs que processeurs-mémoire.

Dans les paragraphes qui suivent, on se contentera seulement de présenter ces différentes approches en mettant l'accent sur leurs influences en termes de performance, faisabilité et classification.

2.1 LES APPROCHES DES RÉSEAUX D'INTERCONNEXIONS

Au moment de procéder au développement d'une machine massivement parallèle, plusieurs approches se présentent autant dans le domaine du calcul lui même qu'au niveau de la communication.

Certaines de ces approches constituent les critères de classification des réseaux et guident les choix de conception. Les approches les plus importantes se résument par les points suivants :

- La topologie,
- Le mode de communication,
- Le mécanisme de commutation,
- La stratégie de contrôle.

Il y a d'autres critères qui interviennent aussi dans la classification des réseaux des machines parallèles. Leurs influences sont d'une manière moins directe dans la classification mais plus significatives dans l'évaluation des performances. On peut citer :

- La connectivité,
- La bande passante,
- Les fonctionnalités,
- La régularité.

2.1.1 TOPOLOGIE

Comme déjà introduits dans le chapitre 1, les réseaux d'interconnexion peuvent être groupés d'une manière générale en trois classes : les hypercubes, les grilles et les réseaux reconfigurables. Ce classement découle de la combinaison des différentes approches déjà citées dans le paragraphe précédent. Un autre classement peut être déjà effectué uniquement selon la topologie. Ce classement se situe à un niveau au point où certaines des autres approches citées (mode de communication, mécanisme de commutation, stratégie de contrôle connectivité, régularité) sont mieux adaptées pour une topologie mais pas pour l'autre. On distingue deux types de réseaux du point de vue topologique : les réseaux statiques et les réseaux dynamiques.

Les réseaux statiques ont une topologie fixe. Elle est invariablement définie lors de la construction de la machine. Les processeurs s'envoient des messages où l'information est copiée ensuite routée vers la destination en passant par les dispositifs d'entrées/sorties et le réseau. Il s'agit là d'une communication basée sur la copie et l'envoi de l'information.

Les réseaux linéaires, les anneaux, les interconnexions en étoile, en arbres et pyramides, les hypercubes, les réseaux cellulaires, ... en sont des exemples. Les réseaux d'interconnexion en hypercube sont les formes les plus populaires des réseaux statiques : les familles d'iPSC d'Intel et nCUBE. Leur popularité est due à la simplicité du modèle mathématique et au fait que le nombre de liens augmente moins rapidement avec la croissance du nombre des processeurs.

Cette topologie est utilisée surtout dans les machines à passage de messages et les machines cellulaires. Les modèles de calcul utilisés avec des topologies statiques incluent les modes SIMD et MIMD mais à mémoire locale.

De bonnes performances peuvent être atteintes dans des machines utilisant cette approche et traitant des applications où la communication est faible et concentrée surtout entre les processeurs voisins [Alm89].

Dans les réseaux dynamiques, la topologie peut varier, au cours de l'exécution d'un programme parallèle, entre deux exécutions différentes ou entre deux phases de communication. Ceci permet d'envisager un placement dynamique des processus générés au cours de l'exécution. La régulation de charge due à la génération et le placement dynamique des processus est l'un des problèmes les plus délicats de ce type de réseaux ainsi que dans les systèmes distribués en général.

Ils trouvent place dans les systèmes multiprocesseurs où la mémoire est commune ou distribuée. Dans les systèmes à mémoire commune la communication est effectuée selon le partage de ressources. Les processeurs sont reliés aux bancs de la mémoire centrale à travers le réseau d'interconnexion qui s'agit généralement des bus, circuits crossbar ou des circuits multiétages. Dans les systèmes à mémoire distribuée, la communication est à passage de messages.

Contrairement aux réseaux statiques, les réseaux dynamiques nécessitent un contrôle. Il est centralisé et s'effectue à travers un contrôleur piloté par l'ordinateur hôte. La gestion dynamique du réseau est aussi un problème délicat de cette topologie.

2.1.2 CONNECTIVITÉ

La connectivité est une mesure des liaisons physiques que chaque nœud dans le réseau avec les autres nœuds. Elle définit également le degré d'un nœud. Le but d'un réseau de communication est de relier tous les éléments d'une structure avec laquelle on veut construire une machine parallèle. Cela revient à dire que tous les processeurs élémentaires seront directement connectés les uns aux autres (complète). C'est la forme la plus performante. Tous les nœuds ont un degré de N-1 et il faut N(N-1) connexions (distance 1).

Malheureusement, les solutions idéales sont les plus difficiles à réaliser. Envisager une connectivité point à point pour une machine massivement parallèle relève de l'impossible avec les techniques et les technologies actuellement utilisées dans les machines sur le marché.

Une connectivité complète ne peut être considérée que pour de très modestes valeurs de N à moins que d'autres techniques et technologies envahissent le domaine des ordinateurs parallèles pour résoudre les problèmes délicats de communication : l'optique par exemple.

Le recours aux réseaux incomplètement connectés est donc inévitable. Ceci exige la nécessité d'envisager une fonctionnalité supplémentaire qui est le routage et de maintenir un équilibre entre l'efficacité en terme de performance et la faisabilité en terme d'encombrement. Ce qui a donné naissance à plusieurs configurations où la connectivité n'est pas complète : réseaux linéaires, anneaux, hypercubes, connexion en étoile, arbres et pyramides...

2.1.3 BANDE PASSANTE

La bande passante est définie, pour un réseau statique incomplètement connecté, par le nombre de messages qui peuvent transiter par un nœud en même temps. Elle est définie par la rapidité pour un réseau dynamique. Elle doit être donc assez élevée pour subsister aux besoins de communication de l'ensemble des processeurs dans la machine.

La bande passante est dictée par la technique de détection de présence des messages utilisée dans le mécanisme de commutation. On distingue deux techniques de détection :

La détection par scrutation qui permet d'acheminer qu'un seul message (ou paquet de message) à la fois. Elle présente l'avantage d'être simple en matière de conception et de réalisation.

Cette méthode est recommandée seulement pour les machines conçues pour faire tourner des applications ayant un rapport calcul/communication supérieur à 1. Par contre, si le nombre de canaux de communication est important (> 4) et la demande en communication est importante, cette manière risque d'aboutir rapidement à une lenteur dans l'acheminement des messages qui entraîne une saturation de l'ensemble du réseau.

La deuxième technique de détection consiste à acheminer les messages (ou paquets) d'une manière parallèle en permettant leur croisement au niveau d'un nœud. La bande passante des réseaux utilisant cette technique est généralement égale au nombre de canaux de communication d'un nœud.

Elle est recommandée pour les machines où la demande en communication est forte. Elle permet de prévenir les interblocages et d'accélérer l'acheminement des messages mais présente la difficulté dans la réalisation et la mise en œuvre. En terme de surface en silicium, celle ci risque d'être importante due à la multitude des bus nécessaires et leur interconnexion si la taille des entités d'information est importante (> 8 bits).

2.1.4 FONCTIONNALITÉS

Les fonctionnalités consistent en des fonctions additionnelles effectuées par le réseau comme lors de la communication : la priorité, l'arbitrage, le traitement du message si nécessaire et éventuellement la combinaison de messages. Généralement, il est difficile de dissocier l'arbitrage de la priorité. Il est indispensable pour un réseau d'être doté d'un système d'arbitrage quand le nombre des canaux d'entrées/sorties au niveau d'un nœud est supérieur à 1.

L'arbitrage consiste à effectuer une gestion des conflits au niveau d'un canal de sortie entre les messages qui sont destinés à emprunter le même canal au même moment.

Le système d'arbitrage, qu'il soit réalisé en matériel ou par logiciel, utilise la notion de priorité pour accomplir son rôle. Il y a deux types de priorité : la priorité fixe et la priorité tournante.

• L'arbitrage avec une priorité fixe consiste à acheminer les messages selon un ordre fixe. Cet ordre peut être selon une numérotation quelconque des canaux, un ordre des directions selon lesquelles les messages sont reçus. Cette technique peut facilement aboutir à des cas de famine si les messages sont plus fréquents sur les canaux prioritaires diminuant ainsi les avantages du parallélisme.

• L'arbitrage avec une priorité tournante a pour objet de prévenir les cas de famine en distribuant une priorité égale à tous les canaux. Pour cela, on utilise des dispositifs de mémorisation des états antérieurs et présents des canaux d'entrées et on n'autorise que l'acheminement des messages qui n'ont pas encore eu droit.

2.1.5 RÉGULARITÉ

La régularité d'un réseau d'interconnexion est sa capacité de supporter les profiles de communication des applications parallèles qui sont la régularité temporelle et la régularité spatiale.

La régularité temporelle est le rythme avec lequel les communications sont effectuées. Elle peut être régulière ou irrégulière (quand la communication doit elle se produire ?).

La régularité au niveau spatiale concerne le graphe de communication (qui communique avec qui ?). Il y a des graphes de communication réguliers et des graphes de communication irréguliers.

La régularité spatiale peut être régulière comme elle peut être irrégulière. Dans les graphes irréguliers les communications ne sont pas prédicables alors que dans les graphes réguliers, toutes les communications sont prédites lors de la compilation.

On distingue deux types de profil de communication régulière : les profils N --> N et les profils N --> 1 (ou l'inverse 1 --> N).

Les profils N --> N est le traitement de N communications-sources en même temps. Ils sont très coûteux en matériel et à la mise en œuvre. Ils sont généralement supportés surtout par les réseaux dynamiques où la connectivité est complète.

Les profils N --> 1 (ou 1 --> N) est la communication de N processeurs avec un seul ou dans le cas inverse un seul avec tous les processeurs comme pour effectuer une diffusion.

2.1.6 MÉCANISMES DE COMMUTATION

La mécanique de commutation est la manière physique de transfert de l'information dans le réseau à travers les nœuds. Il y a plusieurs mécanismes de commutation. Certains sont utilisés uniquement dans les réseaux dynamiques comme les bus, les cross bar et les circuits de commutation multiétages [Alm89] et d'autres sont réservés pour les réseaux statiques comme le "store and forward" et le "wormhole".

2.1.6.1 BUS

La commutation par bus est bien évidemment le concept le plus simple à réaliser. Cette simplicité est payée par les limitations au niveau de la bande passante et la faisabilité. La bande passante est définie par le produit de la fréquence d'horloge et de la taille du chemin de données des processeurs. Elle est inversement proportionnelle au nombre de processeurs N. Ceci non seulement limite la taille du chemin de données mais aussi le nombre des processeurs. Ce qui décourage la volonté de considérer des processeurs plus puissants pour des machines hautement parallèles.

2.1.6.2 CROSS BAR

Un réseau de commutation en crossbar est composé de commutateurs arrangés en une matrice. Chaque commutateur établit une liaison entre deux points permettant d'avoir un chemin entre les processeurs et les bancs mémoire ou entre les processeurs. Le crossbar présente des avantages intéressants :

- Une connectivité complète.
- Le nombre de liaisons croît en 2*N. Ceci est impraticable pour le parallélisme massif.

- Le temps de commutation d'un crossbar correspond à un délai d'une porte logique. Le délai de commutation globale est constant.
- Le faible coût de réalisation en VLSI d'un commutateur.
- La simplicité de la commande du réseau.
- La bande passante est élevée par rapport au bus (10 fois plus) [Ted92].

L'inconvénient majeur de cette technique est la taille de la matrice où le nombre des commutateurs augmente en N^2.

2.1.6.3 CIRCUITS DE COMMUTATION MULTI-ÉTAGES

Les circuits de commutation multi-étages constituent une technique de commutation intermédiaire entre les interconnexions à bus et les circuits crossbar. Ils sont composés d'un ensemble de petits crossbar de 2x2 ou plus larges organisés en rangées interconnectées, constituant un circuit de commutation multi-étages.

Le but d'utiliser plusieurs étages est de réduire le nombre de commutateurs de N^2 à $N\log_2 N$ tout en maintenant une connectivité complète. Ceci est payé par l'augmentation du délai de commutation par un facteur de $\log_2 N$ et le nombre des liaisons qui passe de N à $N\log_2 N$ à cause des étages supplémentaires entre les points communicants [Alm89].

Les réseaux oméga, Benes et Banyan en sont des exemples. Ces réseaux sont tous capables de relier n'importe quel point d'entrée à n'importe quel point de sortie du réseau de circuits crossbar. Ils diffèrent dans le mode d'opération, la stratégie de contrôle et la nature de la connexion à l'intérieur du circuit connexion élémentaire (le crossbar). Si l'utilisation des circuits de commutation à multi-étages réduit le nombre de commutateurs, le risque d'aboutir à des conflits de connexion au niveau des étages intermédiaires n'est pas écarté dans les architectures parallèles et les systèmes distribués. Dans ce contexte, un conflit de connexion se produit dans une situation où deux chemins de sources et destinatinations différentes doivent emprunter le même lien de communication entre étages au même moment [Woe89].

Les circuits de commutation à multi-étages offrent une bande passante intermédiaire entre le bus et les circuits cross bar. Elle est 7 fois plus importante que celle du bus [Ted92]

2.1.6.4 STORE AND FORWARD

Le *store and forward* fait partie des techniques utilisées dans les machines où la communication est à passage de messages. C'est une technique qui consiste à stocker en premier la totalité du message au niveau d'un nœud ensuite de l'acheminer vers un canal de sortie selon l'adresse de destination.

A tout instant, le message est localisé dans un seul nœud mais les inconvénients majeurs de cette technique est le délai important d'acheminement et l'espace mémoire nécessaire à la sauvegarde du message avant d'être acheminé.

Le temps d'acheminement varie selon l'expression : n.m où n est le nombre de nœuds à traverser et m la taille du message

La sauvegarde du message pose aussi un problème de coût des éléments de mémorisation au niveau de chaque nœud si le message est important du point de vue taille.

L'avantage de cette technique réside dans le fait que les messages sont, à chaque étape de leur progression, retirés du réseau évitant ainsi des cas d'interblocage. Le *store and forward* reste une solution mal adaptée pour faire des circuits destinés aux architectures massivement parallèles intégrées.

2.1.6.5 WORMHOLE

Le *wormhole* est une technique introduite par [Sei84] qui a pour but d'éviter les inconvénients du *store and forward* tout en exploitant la possibilité de recouvrement entre les opérations de réception et de réemission lors d'un transfert par paquets sérialisé. Contrairement à la technique précédante, le message n'est pas localisé dans un seul nœud, mais distribué sur m (nombre de paquets) cellules successives. La tête du message ouvre un chemin dans le réseau et la queue le referme.

Le délai d'acheminement est nettement amélioré (n+m) par rapport à celui du *store and forward* (n.m) et la mémorisation n'est plus indispensable.

Ayant des messages traînant partout sous forme de vers (d'où le nom du *wormhole*) à travers le réseau, trivialement, peut présenter des cas d'interblocage surtout si le degré de connectivité d'un nœud est faible. Cet handicap peut être surmonté en augmentant la bande passante par la permission de croisement des messages au niveau d'un nœud.

2.1.6.6 VIRTUAL CUT-THROUGH

Le mécanisme du *virtual cut-through* est une autre technique de communication à passage de messages qui est très proche du *wormhole* . Elle consiste à retirer temporairement un message bloqué du réseau et à le mémoriser jusqu'à ce que le canal soit libre à nouveau.

Le retrait et la mémorisation de la tête du message ou un autre paquet n'empêche pas la queue d'avancer. Ce qui rend la modélisation du temps d'acheminement un peu probabilistique.

La détermination des éléments de mémorisation nécessaires exige de considérer le pire cas. C'est à dire prévoir un espace mémoire relatif à longueur maximale du message. Ce qui est sanctionné par l'augmentation du coût en matériel.

2.1.7 MODES DE COMMUNICATION

Comme la plupart des approches déjà examinées, les modes de communication sont liés aux applications et aux modes de calcul considérés. Le mot mode est utilisé dans plusieurs domaines où il prend des significations différentes. Par mode de communication, on désigne les points suivants :

- Stratégie de contrôle : communication à contrôle centralisé ou distribué.
- Communication synchrone ou asynchrone.

2.1.7.1 STRATÉGIE DE CONTRÔLE

La stratégie de contrôle concerne le pilotage du déroulement de la communication au niveau de chaque nœud. Le contrôle est très lié à la topologie et aux mécanismes de commutation utilisés.

Le contrôle de la communication centralisé est effectué au niveau d'un ordinateur hôte ou d'un système rattaché. Ceci ne peut être possible que pour des systèmes limités en nombre de processeurs. Le parallélisme massif est donc exclu.

Envisager un parallélisme important nécessite la distribution du contrôle des communications. Il faut aussi doter les processeurs élémentaires de fonctionnalités supplémentaires pour pouvoir gérer leur propre échange au

niveau local, d'autant plus que la communication dans les architectures massivement parallèles se fait par la copie et l'envoi des messages.

Comme certains mécanismes de commutation sont mieux adaptés pour les réseaux à topologie dynamique, la stratégie de contrôle centralisée est, elle aussi, convenable pour cette catégorie de réseaux car :

- La communication se fait par partage de ressources,
- Le nombre de processeurs n'est pas important,
- La connectivité est généralement complète,
- Les dispositifs de commutation sont simples à commander.

2.1.7.2 SYNCHRONE/ASYNCHRONE

Dans les réseaux synchrones, les échanges d'information sont effectués au rythme des pas du signal d'horloge. Au niveau électrique, ceci se traduit par le partage du même signal par plusieurs dispositifs.

Au niveau d'un circuit VLSI, le recours à une source du signal d'horloge puissante et l'utilisation de larges pistes du métal pour le véhiculer à travers tout le circuit conduit à une capacité importante limitant la fréquence de fonctionnement. Le recours à l'amplification introduit des retards provocant une desynchronisation entre émetteurs et récepteurs.

Ce mode est adapté aux systèmes destinés à traiter des applications dont le profil de communication est régulier (surtout la régularité temporelle).

Contrairement aux réseaux synchrones, les communications dans un réseau asynchrone sont effectuées d'une manière irrégulière. Les échanges ne se déroulent pas selon la cadence de l'horloge mais conformément à un protocole. Au niveau électrique, le protocole est généralement du type *handshake* .

Chaque processeur envoi et reçoit selon son propre rythme. Chaque nœud doit être doté donc d'un système de détection de présence de message et doit être capable d'accuser réception.

Les réseaux asynchrones s'adaptent bien aux types d'applications dont la régularité spatiale et temporelle des communications sont irrégulières. Le fait d'accomplir l'échange des messages selon un protocole rend les réseaux asynchrones moins rapides que les réseaux synchrones.

2.2 CRITÈRES DE PERFORMANCES

Les critères de performance est l'ensemble de mesures qui permettent de classer un réseau d'interconnexion en matière de performances parmi d'autres. Ces critères constituent également un ensemble de paramètres utiles à l'étude et à la modélisation du réseau, et d'une manière générale de la machine. Ces mesures résultent généralement des approches et des choix de conception ainsi que des facteurs technologiques. Chaque approche présente des avantages et des inconvénients. L'un des rôles des architectes est de faire des compromis en fonction des contraintes prioritaires. En générale, les réseaux d'interconnexion en termes de performance se distinguent par les points suivants : le débit, le temps d'acheminement et la complexité.

2.2.1 DÉBIT ET TEMPS D'ACHEMINEMENT

Le temps d'acheminement constitue très souvent la première des performances à consulter. C'est une mesure de la rapidité avec laquelle un message est transféré d'un nœud à un autre. Le débit est la quantité d'information transférée pendant ce temps. Il peut aussi être défini comme étant la largeur d'un canal en terme du nombre de bits transférés d'un seul coup.
Le temps d'acheminement est très lié au débit. Les transferts en bits series sont rapides mais pénalisés par le temps de reconstitution des messages et le contrôle nécessaire surtout dans les machines utilisant des processeurs différents des mono bits.

Un débit réduit peut constituer une solution pour résoudre des problèmes de coût en matériel au niveau de la surface en silicium et aussi au niveau des brochages des circuits. Cela représente un autre point où un compromis est à faire surtout dans les machines à communication intense.

2.2.2 COMPLEXITÉ

Ce critère reflète le pourcentage du coût en matériel du réseau par rapport au coût total de la machine. Au niveau des circuits VLSI, cette mesure représente le coût en surface de silicium. Ceci présente une mesure importante pour les circuits intégrant des architectures massivement parallèles.
La complexité est en fonction des choix des différentes approches et des compromis effectués généralement entre performances et coût.

Il y a d'autres approches qui interviennent également dans le développement des réseaux de communication et leur mise au point surtout du point de vue matérielle où les performances en sont très sensibles. La technologie, les méthodes de réalisation : *full custom, standard cell, ...* , les choix de conception : séquentielle, combinatoire, constituent un autre ensemble d'approches pour aborder l'élaboration d'un réseau d'interconnexion.

2.3 RÉSEAU D'INTERCONNEXION PROPOSÉ

Notre objectif est de faire un processeur massivement parallèle pour une architecture cellulaire intégrée. Le tour des principales approches des réseaux d'interconnexion étant fait pour répondre aux deux approches des modes de calcul (MIMD; graphe des communications irrégulier) architecturales (cellulaire) et le grain moyen, intermédiaire entre la Connection Machine et le transputer, nous proposons un routeur avec les caractéristiques suivantes :

- Un routeur pour une topologie statique en grille 2D,
- Incomplètement connecté,
- Bande passante de 5 messages avec une priorité tournante,
- Communication asynchrone à contrôle distribué,
- Commutation par wormhole, transfert en paquet sériel.

Au sein de la même équipe, d'autres routeurs ont été proposés et réalisés. Parmi ces routeurs il est intéressant de citer les deux suivants qui diffèrent uniquement par le mode et le débit du transfert ainsi que la méthode d'implémentation :

1) Routeur réalisé pour processeur dédié à la simulation logique [Obj88] :
   - transfert par scrutation en bit série,
   - circuit séquentiel.

Ce travail a montré qu'il fallait quelques centaines de nano secondes (300 ns) pour acheminer un message de 10 bits pour une surface de 2 mm^2 .

2) Routeur réalisé pour un processeur destiné à simuler des réseaux de neurones [Kar89] :
   - transfert parallèl en bit parallèl,
   - circuit combinatoire.

Ce travail a montré qu'il fallait moins de 40 nano secondes pour acheminer un message de 10 bits dans une surface de 13,5 mm^2 . L'aspect temporel s'avère très peu sensible à la taille du message. Une amélioration de la partie mise à jour des adresses par décrémenteur à retenue anticipée réduit considérablement l'effet du nombre de bits dans le message sur le temps d'acheminement.

Bien que ce résultat en comparaison avec celui de routeur séquentiel donne de bonnes performances, beaucoup d'autres améliorations peuvent être

conduites sur le système de priorité tournante afin de réduire encore le temps de traversée d'un message jusqu'à environ 20 nano secondes.

Contrairement à l'aspect temporel, la surface est fortement sensible à la taille du message. La taille des tampons, les buffers, les bus et les dispositifs de mise à jour en sont les causes.

Une modélisation de l'influence de la taille du message sur la complexité (en termes de transistors) a été effectuée pour prédire la surface. La figure 2.1 montre dans un plan 3D les variations de la complexité en fonction du nombre de bits dans le champ adresse et dans le champ des données [Kar89].

complexité = [2182 + 830*x + 266*y,{x,4,8},{y,2,8}] où x est le nombre de bits dans le champ adresse, y est le nombre de bits dans le champ de données. x varie de 4 à 8 bits, y de 2 à 8 bits.

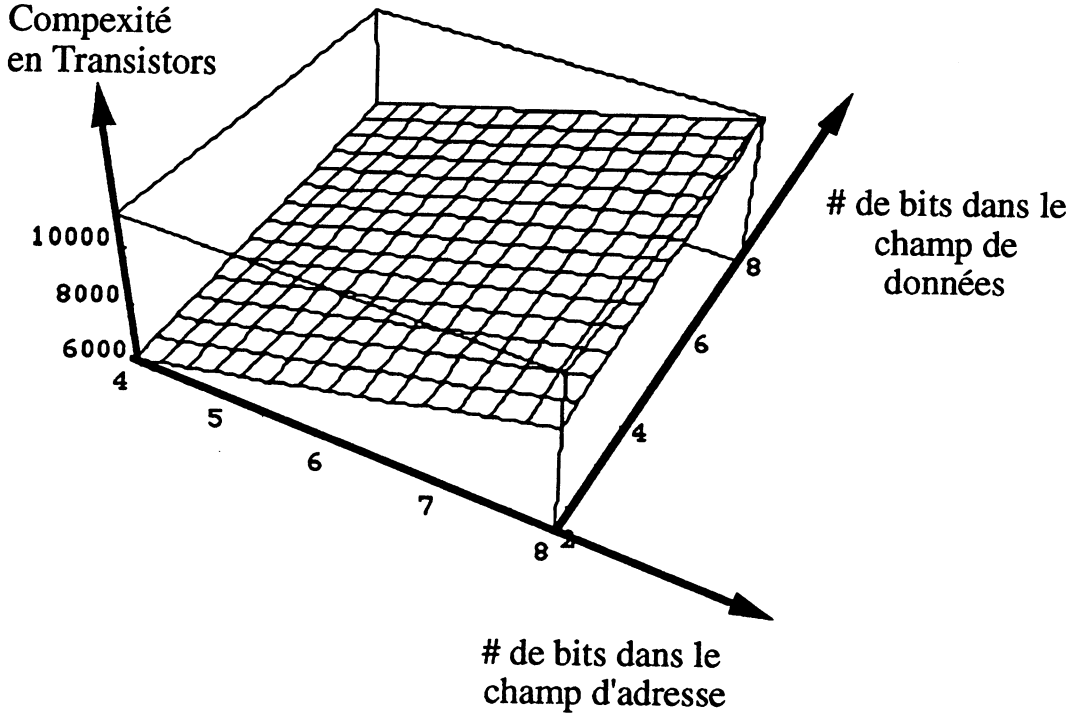


Figure 2.1 : Influence de la taille du message sur la complexité d'un routeur

La différence entre, ce dernier routeur et celui que nous proposons, réside donc dans la structure, la taille du message et le mode transfert.

2.4 IMPLÉMENTATION

Concernant la conception et l'implémentation de la cellule, la méthode descendante a été utilisée. A partir des fonctionnalités principales de la cellule, aiguillage et traitement, les blocs fonctionnels de chacune sont étudiés et des choix d'implémentation sont effectués. Ces choix constituent la tâche la plus délicate lors de la conception de la cellule.

Généralement, les critères des choix de conception de circuits VLSI se résument en terme de surface et de performances. Il faut ajouter parfois, ce qui est notre cas, les moyens de réalisation, tant au niveau matériel que humain. On peut toujours faire un choix mais on peut aussi se heurter à des problèmes dus aux outils utilisés telle que la CAO pour la réalisation.

Notre objectif est d'intégrer un maximum de processeurs dans un même boîtier. Au départ, seule la surface faisait l'objet du critère de choix principal sans trop se soucier des performances et des moyens de réalisation. L'idée étant d'intégrer plusieurs processeurs programmables du type 6809 dans un même circuit. Ceci était très ambitieux. Parallèlement avec l'étude de l'architecture de la machine, le critère de performance fût pris en considération surtout dans la partie routage. Celui-ci doit être au moins deux fois plus rapide par rapport à la partie traitement [Rub92].

Tout le long de ce qui suit, nous adoptons une description descendante de l'implémentation en partant des principaux blocs fonctionnels jusqu'aux portes logiques. La cellule étant composée de deux parties distinctes et autonomes, on commencera d'abord par la mise sous microscope de la partie routage pour passer ensuite à l'unité de traitement dans le chapitre suivant.

La partie routage du processeur élémentaire, comme il a été spécifié auparavant, prend en charge la fonction de communication et d'aiguillage des paquets des messages. Elle doit être capable de détecter les messages incidents sur chaque côté de la cellule y compris les messages émis par l'unité de calcul (processeur élémentaire). Elle doit également traiter l'entête du message si nécessaire et décider sur quel canal de sortie, les paquets de messages doivent être routés.

La partie routage est constituée de trois parties qu'on appellera : bloc d'entrée, bloc de sortie et les tampons. Une paire de tampons, un en entrée et un en sortie, constitue un canal bidirectionnel.

Le coté original du routeur proposé réside au niveau du parallélisme dans l'acheminement des messages. Autrement dit, la bande passante de l'aiguilleur est de cinq messages. Les messages transitant et émis ou reçus par la cellule, peuvent se croiser à tout moment (figure 2.2). Ceci permet d'accélérer le routage des messages et en même temps évite tout embouteillage ou interblocage de messages dans l'ensemble du réseau.

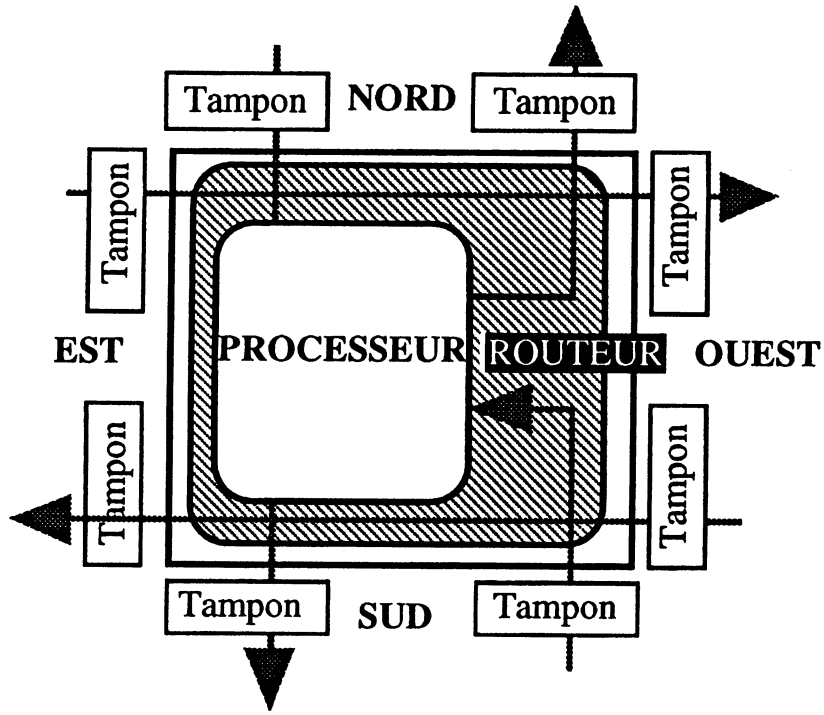


Figure 2.2 : Croisement des messages

En partant de ce concept, la structure du routeur repose sur une combinaison d'interconnexion de la duplication à cinq exemplaires de chacun des deux blocs et les tampons comme c'est montré dans la figure 2.3.

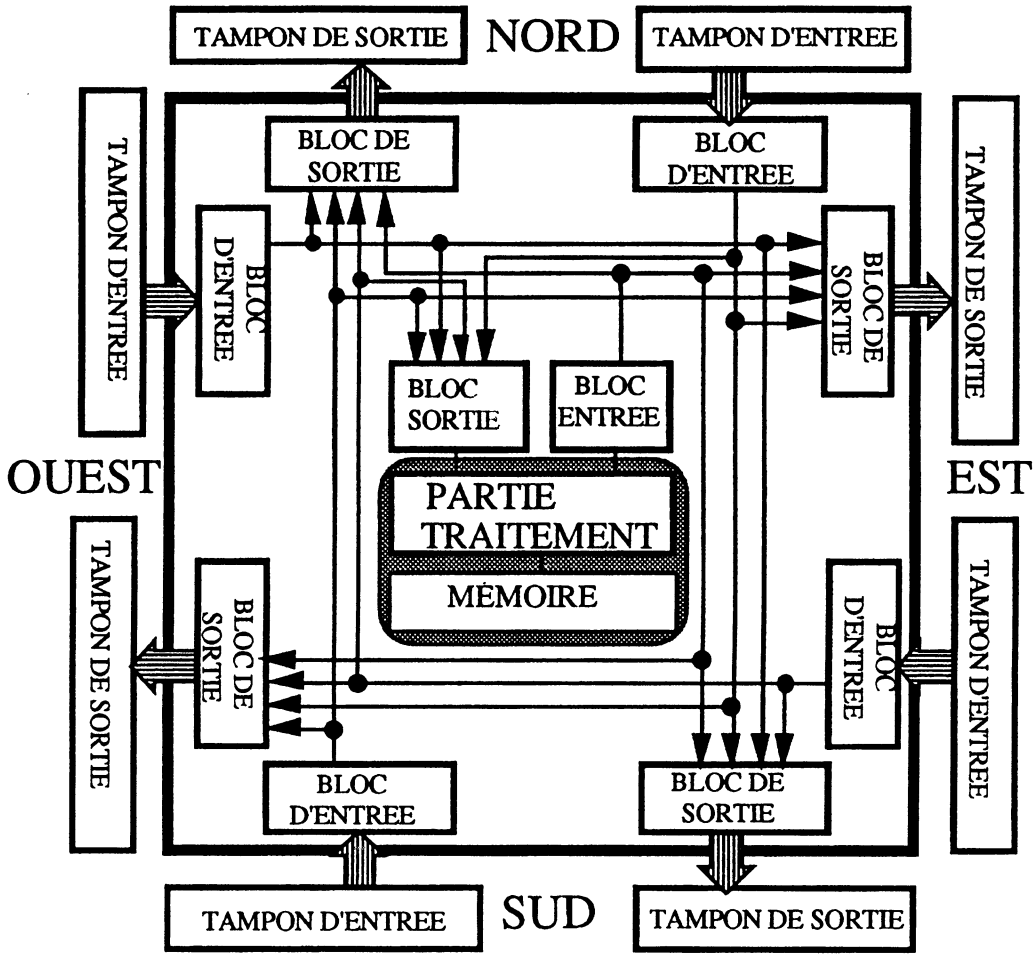


Figure 2.3 : Structure du routeur tout parallèle

2.4.1 STRUCTURE DU MESSAGE

Il est inutile de rappeler à ce niveau, l'importance du choix de la structure du message. Elle constitue l'un des paramètres sur lequel l'efficacité d'un système de communication dépend d'une manière considérable.

Après avoir comparé des réalisations de différents routeurs; routeur séquentiel à scrutation [Obj88], routeur tout parallèle combinatoire [Kar89], notre équipe, par la suite, a été inspiré du wormhole de Caltech [Fla87] où la taille du message est variable dans le sens où le message est divisé en petits paquets de bits et envoyés en une suite d'entités.

La structure utilisée est généralisée utilisant un codage préfixe pour coder l'adresse relative en plusieurs entités ("flits" ou paquets, figure 2.4). Le message est divisé en champs séparés par des délimiteurs permettant une flexibilité dans la taille même des champs des adresses relatives. En conséquence, ceci d'une part offre la possibilité de considérer des grilles de processeurs n x n et d'autre part, nécessite l'utilisation des comparateurs et des compteurs très gourmands en transistors. En réalité trois bits seulement sont utilisés pour véhiculer l'information, 2 bits pour les données et le troisième pour le contrôle marquant la différence entre une donnée (état 0) et l'un des codes de la structure du message tel que +, -, ., T.

+ : contrôle de direction +
- : contrôle de direction -
. : séparateur indiquant un zéro à gauche
T : contrôle de queue de message
..3 : adresse relative en base 4
M : entité de données, contenant l'un des symboles 0 .. 3

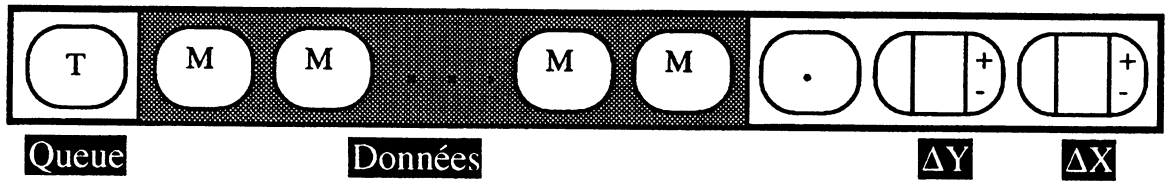


Figure 2.4 : Structure du message du Wormhole de Caltech

Au fur et à mesure que le message avance dans le réseau, sa longueur diminue. Les entités d'en-tête devenant inutiles, sont absorbées. L'exemple suivant explique le wormhole de Caltech.

Source	T M M M M . 3 + 1 2 +
Nœuds 1	T M M M M . 3 + 1 1
Nœuds 2	T M M M M . 3 + 1 0
Nœuds 3	T M M M M .3 + . 3
Nœuds 4	T M M M M . 3 + . 2
Nœuds 5	T M M M M . 3 + . 1
Nœuds 6	T M M M M . 3 + . .
Nœuds 7	T M M M M . 2
Nœuds 8	T M M M M . 1
Nœuds 9	T M M M M . .
Destination	M M M M

Nous avons été amenés à proposer une structure de message plus simple tout en exploitant les avantages du concept du wormhole pour les raisons suivantes :

• L'approche intermédiaire entre celle de la Connection Machine et celle du Transputer.

• Les applications dont les algorithmes pouvant être facilement implémentés sur un réseau comme le nôtre [Rub92].

• l'objectif de réaliser un circuit intégrant le maximum possible de processeurs.

La structure du message proposée inclue trois champs. Chaque champ a une longueur d'un octet. Un message est donc composé de trois octets :

- Le premier représente l'adresse relative en X et en Y.
- Le deuxième octet correspond à l'adresse réelle où la donnée doit être rangée dans la mémoire locale du processeur de destination (TAG).
- Le troisième octet contient la donnée à échanger entre les deux cellules communicantes (figure 2.5).

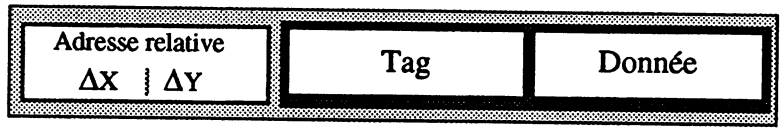


Figure 2.5 : Structure du message proposée

Cependant, il a été jugé utile de comparer, en termes de surface et performance des implémentations de routeurs du type wormhole à taille variable. La comparaison a porté sur la longueur du message, d'une part, et d'autre part du point de vue de la taille des paquets. Des routeurs permettant l'acheminement des paquets de message de tailles de 2, 4, et 8 bits ont été réalisées [Dje92]. Le tableau suivant résume les résultats des réalisations menées.

	Wormhole taille fixe 8 bits	Wormhole taille variable 8 bits	Wormhole taille variable 4 bits
Surfaces en mm^2	6,5	8,59	6,55
Acheminement d'un paquet (F=25Mhz)	4 cycles 160 ns	3 cycles 120 ns	3 cycles 120 ns

Tableau 2.1 : Comparaison des surfaces et performances

Ces réalisations ont été menées à l'aide de l'outil de conception *Cadence.* Les résultats en surface ont été obtenus en se servant du placement et routage automatique en utilisant les valeurs des largeurs des métaux par défaut (donc minimales) pour tous les signaux dans le circuit. Malgré cela, ces résultats restent significatifs et permettent de faire et de justifier les choix de conception et réalisation.

Comme il sera montré dans le chapitre 4, pour avoir les surfaces réelles avec les dimensions des largeurs convenables, il faut multiplier les surfaces par un facteur de deux. Pour les trois réalisations, les performances sont comparables mais les tailles ont des différences d'1 mm^2. La troisième réalisation utilisant une entité d'une taille de 4 bits peut déjà faire l'objet d'un rejet vu le faible débit de transfert de l'information et la surface occupée.

2.4.2 LES TAMPONS D'ENTRÉES/SORTIES

Les tampons sont des éléments de mémorisation qui sont partagés par les cellules adjacentes verticalement et horizontalement. Ils servent de moyen d'échange des messages entre deux cellules voisines. Ils assurent également l'implémentation du protocole communication entre les deux cellules voisines ainsi que l'asynchronisme dans l'échange des paquets.

Le tampon est constitué d'un registre et d'un drapeau. La taille du registre dépend uniquement de la taille du paquet. Pour notre cas, la taille du paquet est de huit bits. Quatre bits pour chacune des adresses relatives ΔX et ΔY où un bit est réservé pour le sens de déplacement (signe) EST/OUEST pour X et NORD/SUD pour Y et les autres trois bits pour le nombre de cellules à transiter (8 cellules au maximum 2^3).

Le drapeau est constitué d'une bascule RS avec une combinaison de portes NOR (figure 2.6) pour assurer le non recouvrement des signaux de l'état du registre. Il indique à tout moment l'état du registre plein ou vide et assure l'exclusion mutuelle de ces deux signaux. En plus, il prend en charge les commandes de remplissage (remplir) du registre et d'acquittement (vider) afin de mettre à jour ses états *plein* ou *vide*.

Les règles constituant le protocole de communication entre deux cellules voisines sont :

- L'état d'un tampon doit pouvoir être interrogé à tout moment,
- Une cellule ne peut écrire dans un tampon que s'il est *vide*,
- Une cellule ne peut lire un tampon que s'il est *plein*,
- Une cellule ne peut envoyer un signal *vider* qu'à la fin de l'écriture sur le tampon de sortie (fin d'écriture sur la cellule suivante).

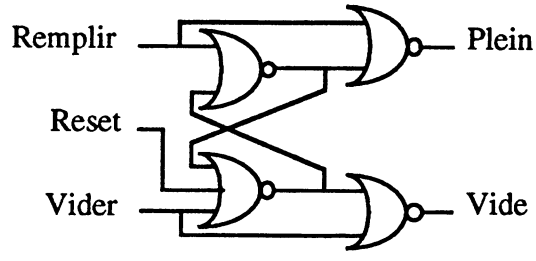


Figure 2.6 : Drapeau

Le drapeau fonctionne selon le chronogramme montré dans la figure 2.7. Le drapeau n'est vide qu'à la fin de la commande *Remplir*, de même, il n'est plein qu'à la fin de la commande *Vider*. Au cours de ces deux opérations, il est considérée comme ni plein ni vide.

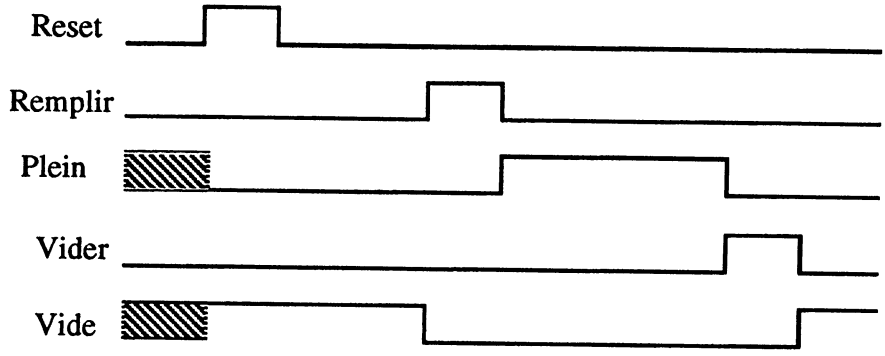


Figure 2.7 : Chronogramme Drapeau

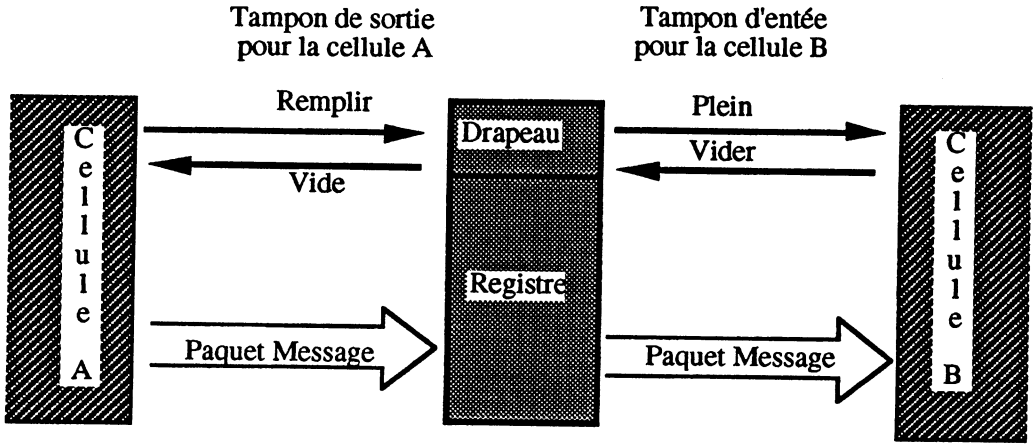


Figure 2.8 : Protocole de communication entre deux cellules voisines

2.4.3 BLOC D'ENTRÉE

Le bloc d'entrée est un circuit du type pc-po. Son rôle principal est d'assurer les fonctionnalités suivantes :

- La détection de présence de messages dans le tampon d'entrée.
- La détection du mode transparence (voir paragraphe suivant).
- L'analyse de l'adresse relative et calcul de la direction de sortie.
- La mise à jour de l'adresse relative.
- L'acheminement des paquets de message (multiplexage).
- L'attente et la réception des acquittements venant des blocs de sortie indiquant la fin du transfert d'un paquet de message.

La synoptique globale du bloc d'entrée est donnée dans la figure 2.9. Dans les paragraphes suivants, on présente une description détaillée de chaque fonctionnalité du bloc d'entrée. Le graphe de l'automate de contrôle est montré à la fin de la description.

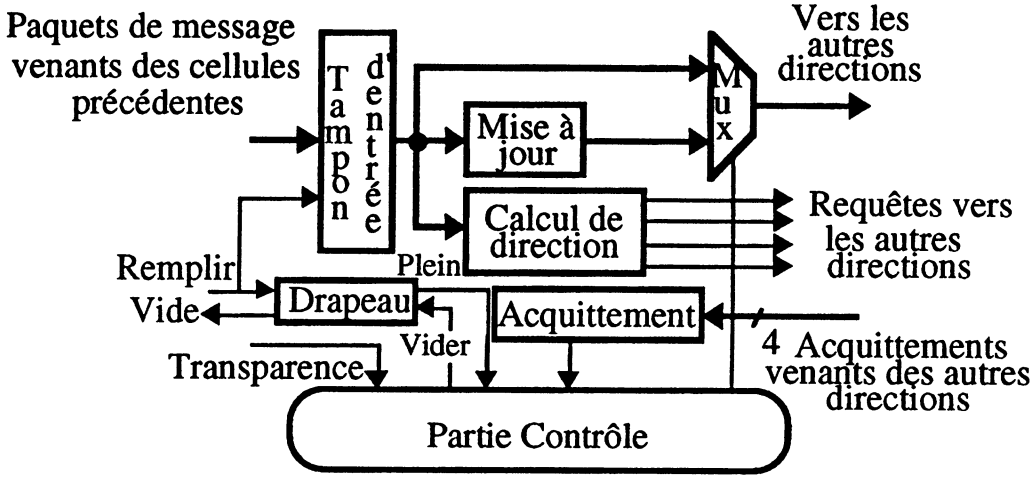


Figure 2.9 Synoptique du bloc d'entrée

Au niveau de chaque tampon d'entrée, on retrouve une copie de ce circuit. On peut déjà constater que selon la stratégie d'acheminement (selon X d'abord ensuite selon Y), on ne peut pas avoir des messages qui prennent un virage Y-X. Ces cas de figure sont montrés en figure 2.10.

Ceci se traduit par quelques simplifications au niveau des automates de contrôle des blocs d'entrée et de sortie, du circuit de calcul des destinations dans le bloc d'entrée et les interconnexions entre blocs, de manière générale, surtout des bus comme le montr la figure 2.11.

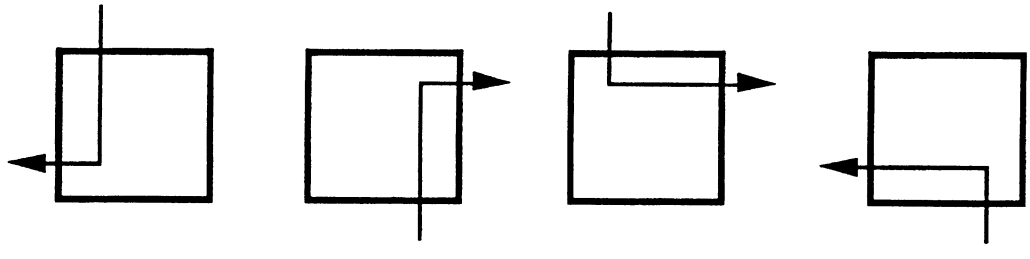


Figure 2.10 : Cas de figures des virages Y-X.

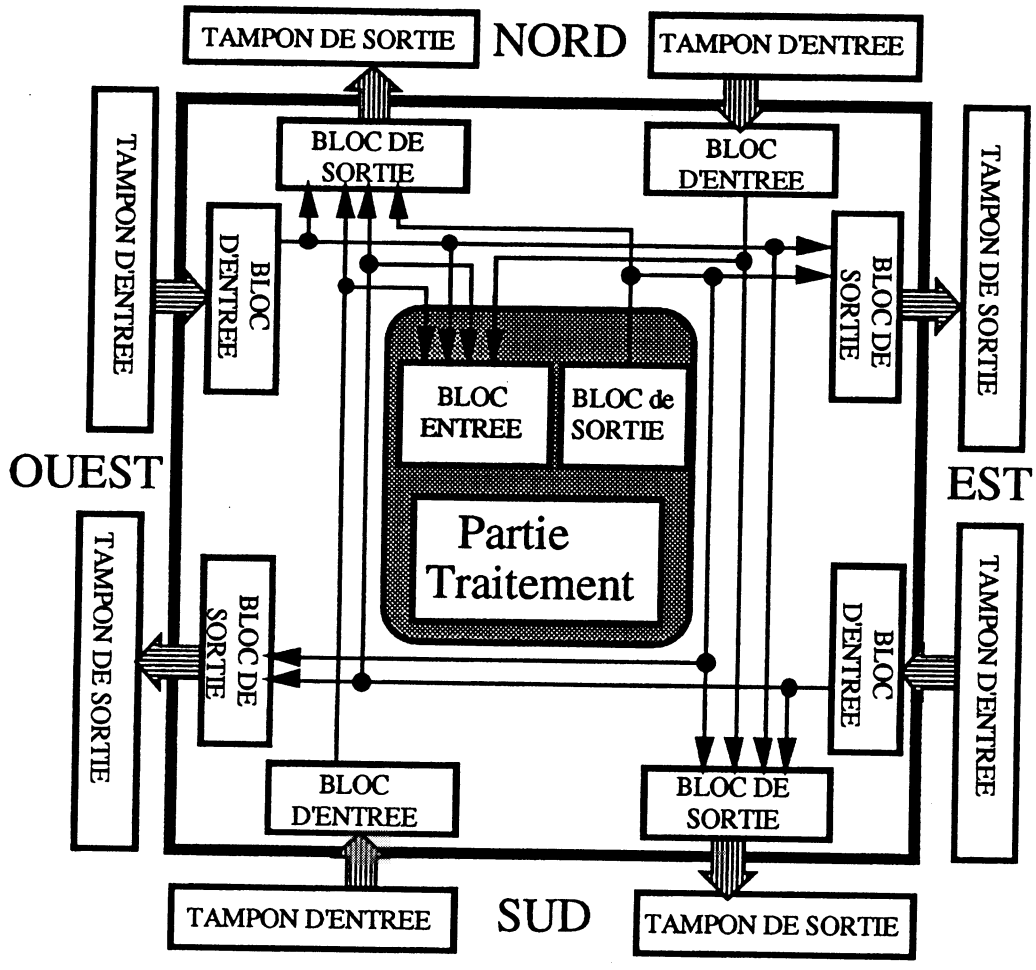


Figure 2.11 : Routeur tout parallèle simplifié

Cependant, ce gain ne compense pas l'avantage de garder la symétrie du circuit et d'augmenter la répététivité réduisant ainsi le temps, le coût et les risques d'erreurs de conception.

2.4.3.1 LA NOTION DE TRANSPARENCE

La notion de transparence est introduite, d'une part, pour faciliter le chargement du réseau quand la taille de celui-ci est importante (n > 8) et, d'autre part, pour assurer une synchronisation du début d'exécution des programmes au niveau des unités de traitement des processeurs élémentaires.

La taille de l'adresse relative étant limitée à trois bits, on ne peut adresser qu'une portion de 8x8 cellules à partir de la cellule d'injection. La figure 2.12 illustre les limitations d'adressage des cellules. Mais d'un autre côté, on gagne sur le temps d'acheminement ainsi que sur le coût en silicium

Physiquement, la transparence consiste en deux signaux nommés transparence horizontale et transparence verticale (Th et Tv) qui sont communs à chaque sous réseau de 8x8 cellules (figure 2.13), contrôlés par l'ordinateur hôte. Dans le contexte du parallélisme massif, on dispose d'un réseau contenant des milliers de processeurs. Seulement, quelques cellules peuvent être utilisées par l'ordinateur hôte comme cellules d'injection au réseau.

Le système de transparence permet l'envoi des messages à des processeurs élémentaires lointains par rapport aux cellules d'injections lors du chargement du réseau. En mode transparence, le dispositif de mise à jour de l'adresse relative dans la partie de routage d'un sous réseau est inhibé. Cependant, les systèmes de calcul de destination, multiplexage, acquittement et arbitrage restent opérationnels.

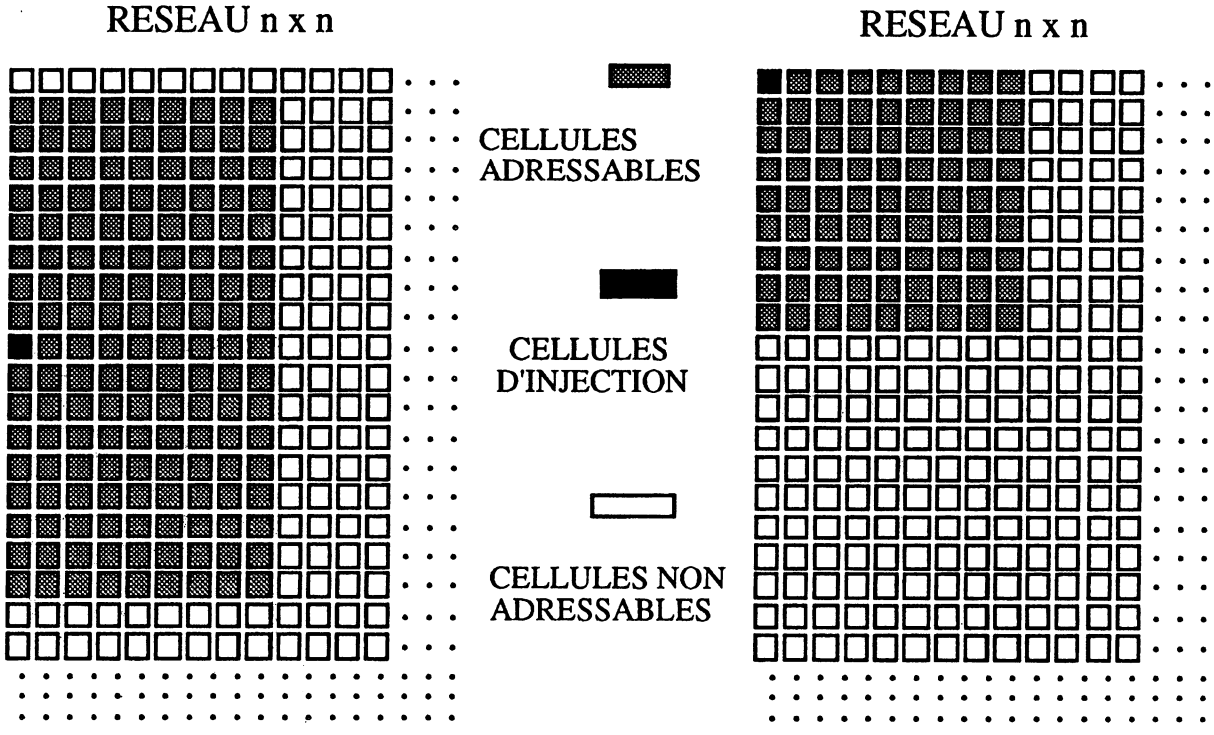


Figure 2.12: Limitation d'adressage

L'algorithme d'acheminement en fonction de la transparence est le suivant :

Tant que *plein* = **vrai**
  Si Tv = **vrai**
    Alors
      Si Th = **vrai**
      Alors Adresse = Adresse
      Sinon Adresse = Adresse -1 (Sur X seulement)
    Sinon (Tv = **faux**)
      Si Th = **vrai**
      Alors Adresse = Adresse - 1 (Sur Y seulement)
      Sinon Adresse = Adresse - 1 (Sur X ou Y)
Fin tant que

La figure 2.13 illustre la notion de transparence.

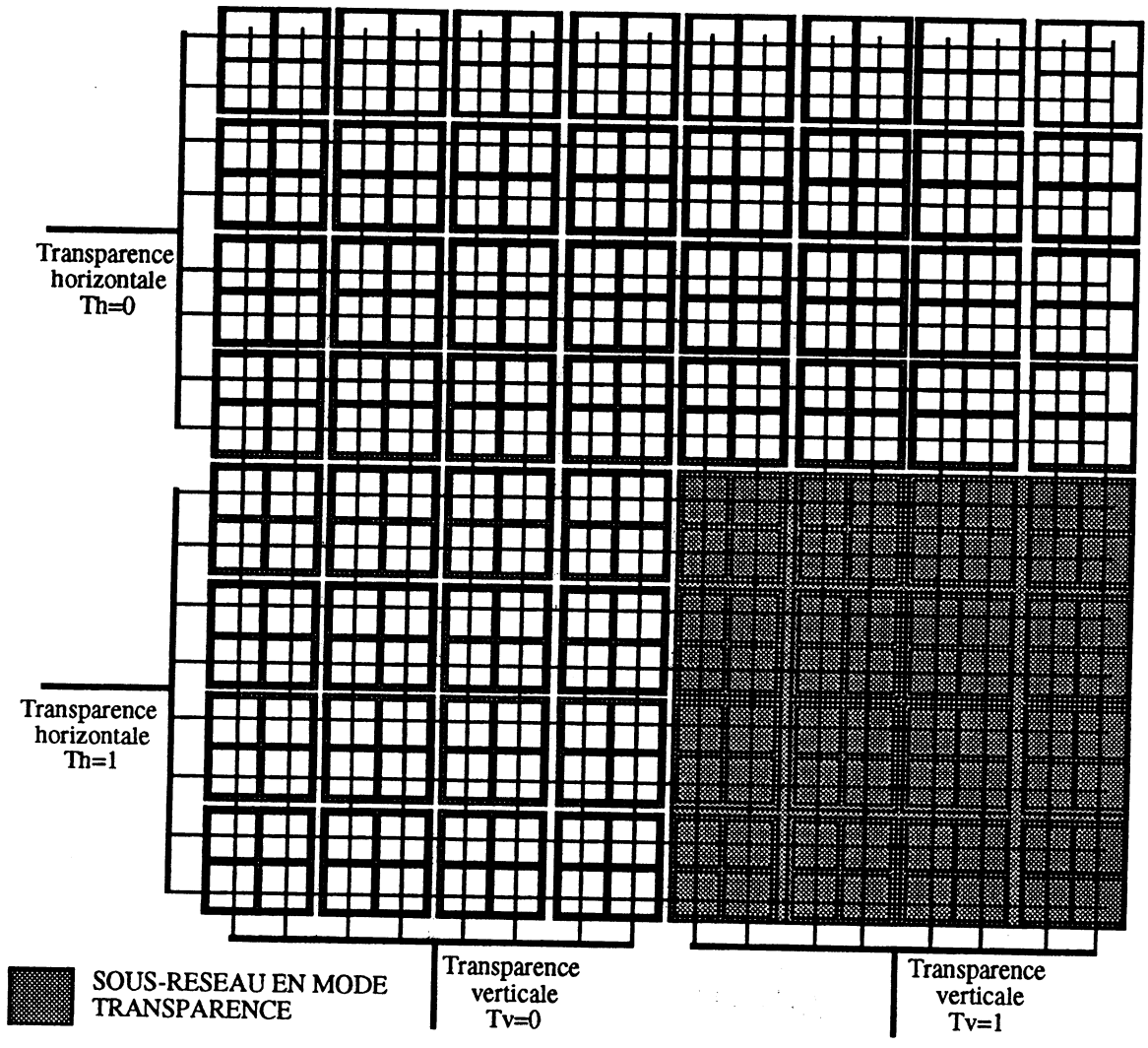


Figure 2.13: La notion de transparence

Au niveau de l'unité traitement, les deux signaux Th et Tv assurent également un moyen matériel pour séparer la phase de chargement de celle d'exécution. Ils sont donc utilisés pour indiquer aux processeurs la fin de la phase du chargement de leur code respectif. Egalement ils signalent aux processeurs à quel moment peuvent-ils commencer son exécution.

Ce système est à ne pas confondre avec un mécanisme de synchronisation des processeurs dans une machine parallèle.

L'introduction de la notion de transparence se traduit bien sûr par deux signaux supplémentaires au niveau des deux automates de contrôle (automate de contrôle de l'unité de traitement et l'automate de la partie entrée du routeur) et de deux plots d'entrées au niveau du brochage du circuit.

2.4.3.2 ANALYSE DE L'ADRESSE RELATIVE

Le calcul de la direction de sortie nécessite d'abord une analyse de l'adresse relative. Elle est effectuée suivant les propriétés arithmétiques de la tête du message. Elle dépend principalement de la magnitude et du signe de chacun des ΔX et ΔY. En ce qui concerne la magnitude, son analyse consiste en la détection d'une adresse relative nulle à l'aide d'une porte OU à trois entrées (figure 2.14). Une porte est affectée à chacun des ΔX et ΔY. Le signe étant présent dans l'adresse, il influe d'une manière considérable sur la décision d'acheminement du message.

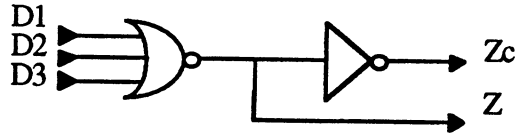


Figure 2.14 : Analyse de l'adresse relative

2.4.3.3 MISE À JOUR DE L'ADRESSE RELATIVE

La mise à jour de l'adresse est une simple décrémentation par un. Un décrémenteur de trois bits est associé également à chacun des ΔX et ΔY. Leurs sorties sont connectées à un multiplexeur 3x2-->1. Puisque notre stratégie d'acheminement des messages est : selon X d'abord puis selon Y, la sélection dépend uniquement du résultat d'analyse de ΔX. Les équations logiques gérant un décrémenteur d'un bit sont :

$$S=D \oplus Ri \quad \text{et} \quad R=!D.Ri$$

Le schéma logique d'un décrémenteur trois bits est montré dans la figure 2.15.

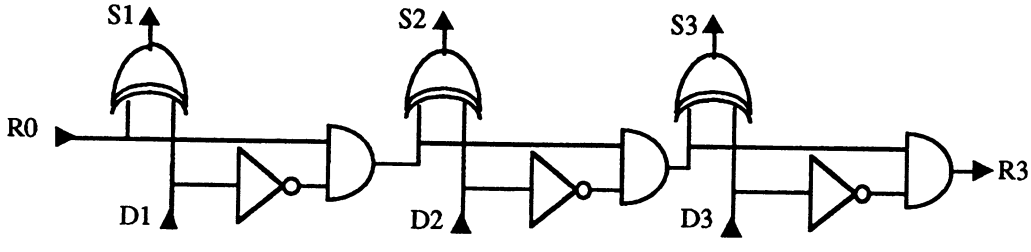


Figure 2.15 : Décrémenteur trois bits

Il est utile de noter que pour un décrémenteur, la retenue entrante du premier bit étant toujours à 1. N'ayant pas besoin de la retenue sortante, on aura donc à en générer seulement pour le deuxième et le troisième bit. Les équations logiques sont :

$$
\begin{aligned}
Ri1 &= 1 \\
S1 &= !D1 \\
S1 &= D1 \oplus Ri1 = D1 \oplus D0 \\
Ri1 &= !D0 \\
Ri2 &= !D1.Ri1 = !(D1 + D0) \\
S2 &= D2 \oplus Ri2
\end{aligned}
$$

(! = Complément, . = "ET" logique)
(≈ = ou exclusif, + = "OU" logique)

Le schéma simplifié est montré dans la figure 2.16. La porte XOR de la bibliothèque ESII contient 11 transistors. Avec cette simplification, on a un gain de 50% sur le nombre de transistors.

On peut imaginer un autre moyen de détection d'une adresse relative nulle en utilisant la retenue sortante du décrémenteur. Cette solution n'a pas été retenue à cause du temps de propagation de la retenue. La solution proposée est nettement meilleure car non seulement, on gagne au niveau de la complexité, mais la détection de l'adresse nulle est réalisée plus rapidement se fait en parallèle avec la décrémentation. Ce qui accélère considérablement la mise à jour de l'adresse relative ainsi que le calcul du canal de sortie.

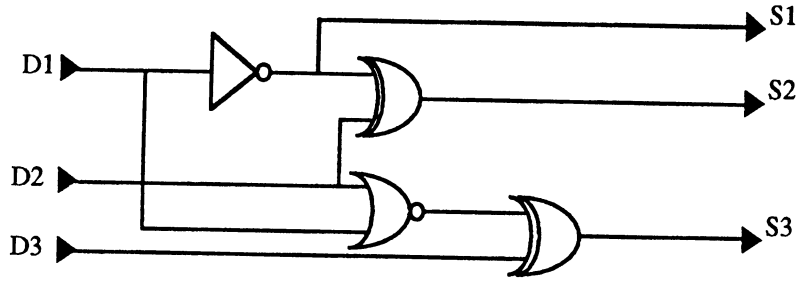


Figure 2.16 : Décrémenteur trois bits simplifié

2.4.3.4 CALCUL DE DESTINATION

La destination de sortie d'un message est calculée en fonction des magnitudes et des signes de chacune de ΔX et ΔY. Disposant des bits de signes Sx et Sy de ΔX et ΔY (respectivement), récupérables directement du tampon d'entrée avec leurs compléments et de Zx et Zy calculé par le dispositif d'analyse de l'adresse, indiquant si oui ou non les magnitudes de ΔX et ΔY sont nulles. Le tampon de sortie est déterminé selon l'algorithme suivant :

```
Tant que Plein=vrai
    Si DX ≠ 0
    |   Alors
    |   Si DX > 0
    |   Alors sortie = EST
    |   Sinon sortie = OUEST
    Sinon (DX = 0)
    |   Si DY = 0
    |   Alors sortie = TRAITEMENT_CELLULE
    |   Sinon
    |   |   Si DY > 0
    |   |   Alors sortie = SUD
    |   |   Sinon sortie = NORD
Fin Tant que
```

Les équations logiques implémentant cet algorithme sont:

EST	= ! ZX . !SX
OUEST	= ! ZX . SX
TRAITEMENT	= ZX . ZY
SUD	= ZX . !ZY . !SY
NORD	= ZX . !ZY . SY

Le schéma logique est montré dans la figure 2.17. Le routeur ne fait la mise à jour de l'adresse du message transitant que sur ΔX **ou** ΔY. Afin d'exploiter le temps de propagation de la retenue dans le décrémenteur, l'analyse de l'adresse relative se fait en parallèle avec la décrémentation. De la même manière, le calcul de la destination de sortie s'effectue en même moment que le multiplexage réalisant ainsi un parallélisme interne.

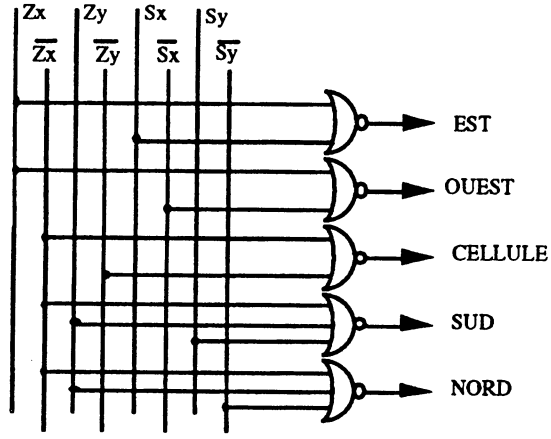


Figure 2.17: Calcul de destination

La fusion des deux algorithmes précédents (transparence et calcul de sortie) nous donne le suivant :

```
Tant que Plein = vrai
   Si DX ≠ 0
      Alors
         Si DX > 0
         Alors
            Si Th = vrai    (transparence horizontale)
            Alors Sortie = EST , sans mise à jour
            Sinon Sortie = EST , mise à jour sur X
         Sinon (DX < 0)
            Si Th = vrai
            Alors Sortie = OUEST , sans mise à jour
            Sinon Sortie = OUEST , mise à jour sur X
      Sinon (DX = 0)
         Si DY = 0
         Alors Sortie = TRAITEMENT_CELLULE
         Sinon
            Si DY > 0
            Alors
               Si Tv = vrai (transparence verticale)
               Alors Sortie = SUD , sans mise à jour
               Sinon Sortie = SUD , mise à jour sur Y
            Sinon
                  Si Tv = vrai
                  Alors Sortie = NORD , sans mise à jour
                  Sinon Sortie = NORD , mise à jour sur Y
Fin Tant que
```

2.4.3.5 MULTIPLEXAGE

En réalité on a deux types de multiplexage. Le premier se situe au niveau de la tête du message (contenant l'adresse relative) et s'effectue entre les adresses mises à jour et celles non mises à jour. Autrement dit entre ΔX et (ΔX - 1) dans le cas où $\Delta X \neq 0$ et entre ΔY et (ΔY -1) dans le cas où $\Delta X = 0$. Le deuxième type de multiplexage se situe au niveau du message même (le corps et la queue) et s'effectue entre les paquets du message transitant par le dispositif d'analyse et de mise à jour de l'adresse relative et ceux qui ne font que transiter par le routeur, comme le Tag et la Donnée. Puisque les bits de signe Sx et Sy ne subissent aucun traitement, quatre multiplexeurs du type 3x2 ->1 sont utilisés.

La synoptique du dispositif de traitement de l'adresse relative dans le circuit bloc d'entrée est donnée dans la figure 2.18. Une sauvegarde de la direction de sortie s'est avérée nécessaire d'une part pour implémenter correctement le concept du wormhole (la tête devant creuse le chemin) et d'autre part pour ne pas perdre ce chemin pendant l'acheminement du deuxième et du troisième paquet du message puisque le dispositif de calcul de direction est un circuit purement combinatoire. La commande de sauvegarde est activée par l'automate de contrôle à la fin d'acheminement du paquet de l'adresse relative.

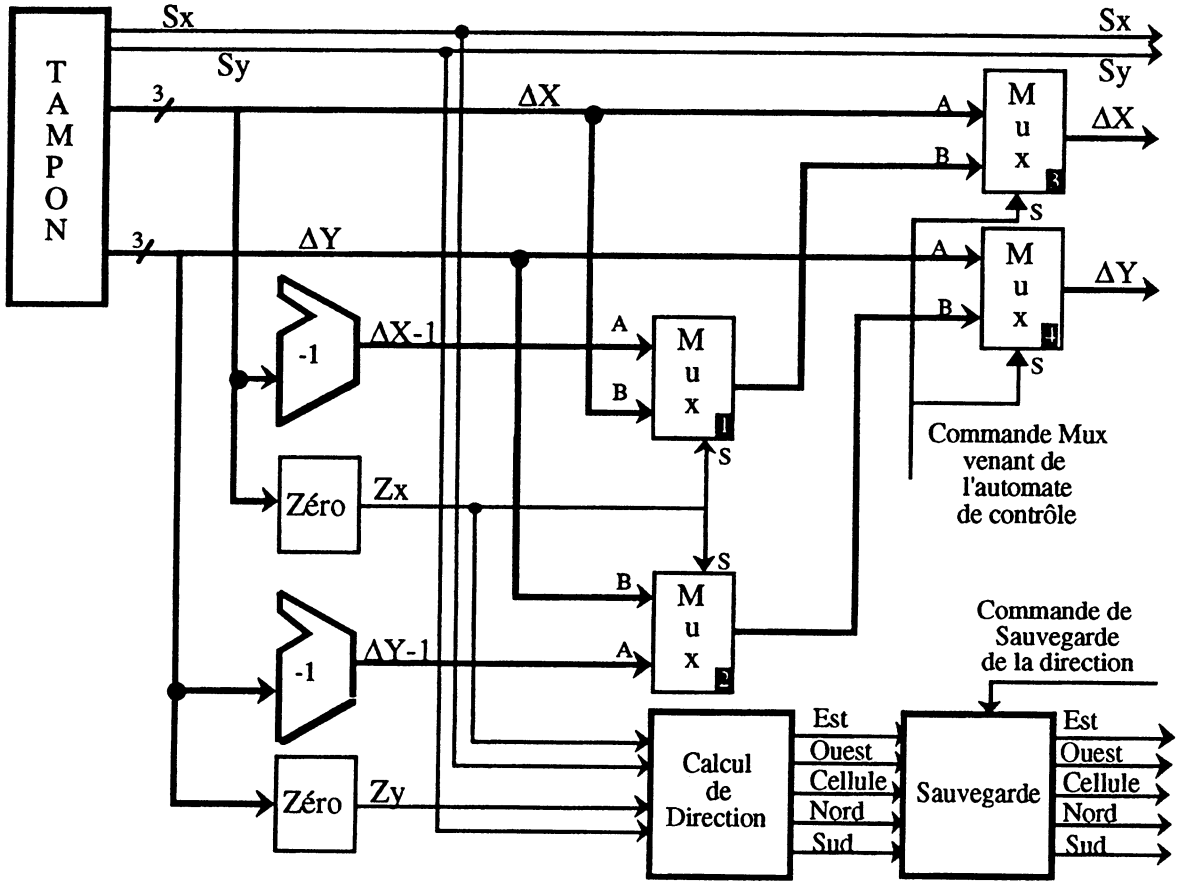


Figure 2.18 Multiplexage

Les signaux de direction EST, OUEST, CELLULE, NORD et SUD ne sont actifs que pendant que le signal PLEIN est vrai. C'est à dire chaque signal indiquant une direction, passe à travers une porte "ET" avec le signal PLEIN, sachant que ce dernier retombe à zéro après chaque acquittement.

Ce dispositif est placé après la sauvegarde d'une part, pour implémenter correctement l'algorithme d'acheminement et d'autre part, pour utiliser le même

moyen matériel pour signaler au bloc de sortie l'arrivée des paquets suivants, à savoir le Tag et la donnée.

2.4.3.6 ACQUITTEMENT

Avant de considérer un nouveau paquet de message, on doit s'assurer que le dernier paquet a bien été écrit sur le tampon de sortie calculé. Le routeur étant complètement parallèle, on dispose donc, au niveau de chaque tampon d'entrée, de quatre signaux d'acquittement venant des quatre autres blocs de sortie. On doit s'assurer également de la source du signal d'acquittement. Logiquement, au niveau d'un bloc d'entrée, on n'attend un acquittement que de la direction qu'on vient de calculer pour un message donné. Pour cela, on fait passer la demande D_i et le signal d'acquittement A_i à travers une porte ET et les quatre sorties des portes ET à travers une porte OU réalisant ainsi l'équation logique :

$$Ack = (D1.A1)+(D2.A2)+(D3.A3)+(D4.A4)$$

Le circuit logique équivalent est montré dans la figure suivante :

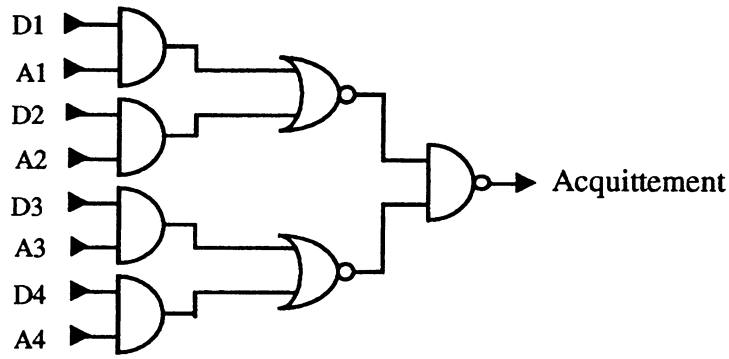


Figure 2.19 : Circuit d'acquittement

2.4.3.7 AUTOMATE DE CONTRÔLE

L'automate de contrôle du bloc d'entrée est montré dans la figure 2.20. Il est constitué de 7 états. En moyenne, deux états sont nécessaires pour chaque paquet de message. Le premier état est relatif à la détection et l'acheminement d'un paquet. Le deuxième est destiné à l'attente de l'acquittement venant de l'un des blocs de sortie.

A part l'analyse de l'adresse relative, la mise à jour et le multiplexage interne, l'ensemble des fonctionnalités du bloc d'entrée sont prises en charge par l'automate de contrôle. Sa partie combinatoire est réalisée à base d'un petit PLA.

Il est évident que devant chaque implémentation d'un bout du circuit, des choix de conception se présentent. Pour un tel automate, le nombre d'états peut facilement être réduit par utilisation d'un compteur modulo-3 (2 bascules). Il faut au minimum, quatre bascules en plus d'une entrée et d'une sortie supplémentaire au PLA.

Il est donc plus avantageux du point de vue surface d'éviter le compteur et d'augmenter le nombre d'états puisque seulement trois bascules sont nécessaires. Afin d'obtenir un codage compact et optimal des états et un nombre minimal des monômes du PLA, on a utilisé un outils de synthèse automatique.

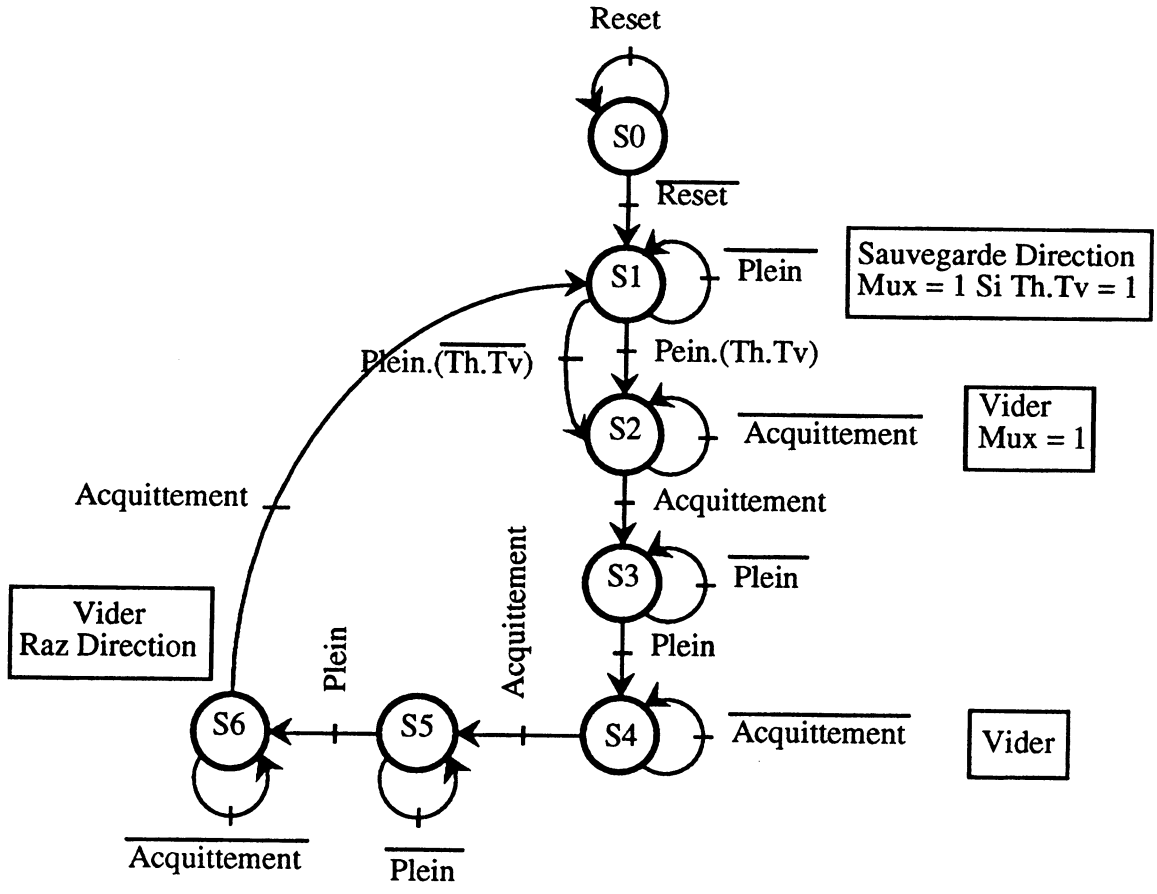


Figure 2.20: Graphe de l'automate de contrôle du Bloc d'entrée

2.4.4 BLOC DE SORTIE

Cette partie est également constituée d'un circuit pc-po dont le rôle principal est d'assurer les fonctionnalités suivantes :

- La détection des requêtes venants des quatre autres blocs d'entrée.
- L'arbitrage entre les messages acheminés vers le tampon de sortie considéré.
- La distribution d'une priorité égale aux quatre requêtes (priorité tournante).
- L'envoi des accusés de réception vers les blocs d'entrée (acquittement).

Au niveau de chaque tampon de sortie, on retrouve une copie du bloc de sortie. Ce circuit peut recevoir des demandes d'accès au tampon de sortie correspondant des quatre autres blocs d'entrée (y compris la partie traitement). Donc, les liaisons physiques entre un bloc d'entrée et un bloc de sortie sont constituées d'un signal de demande d'accès (direction calculée), d'un bus de données de huit bits et un signal d'accusé de réception (acquittement). La figure suivante résume la liaison physique entre un bloc d'entrée et un bloc de sortie.

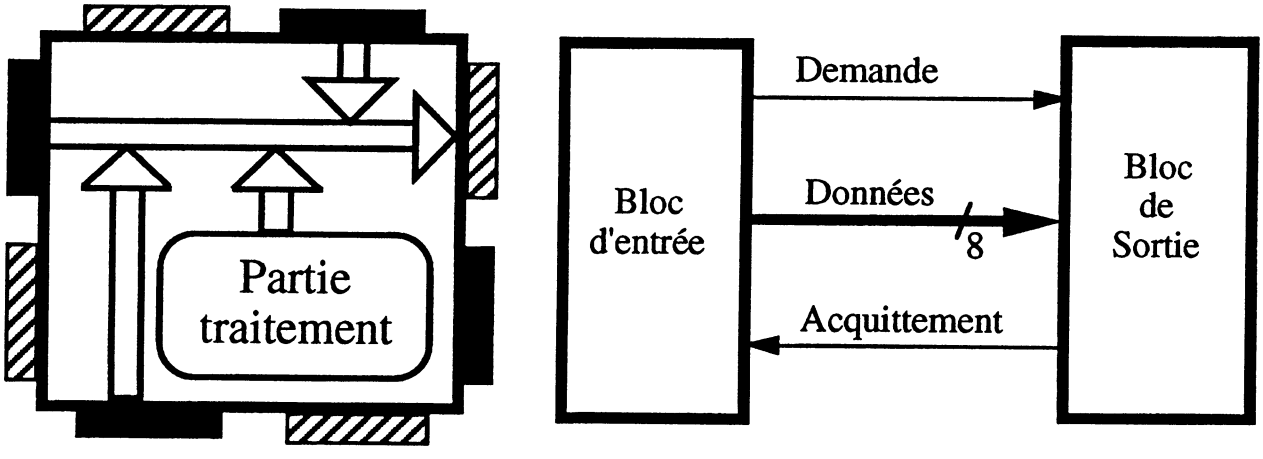


Figure 2.21: Liaison physique entre un bloc d'entrée et un bloc de sortie

La synoptique globale d'un bloc de sortie est montrée dans la figure 2.22.

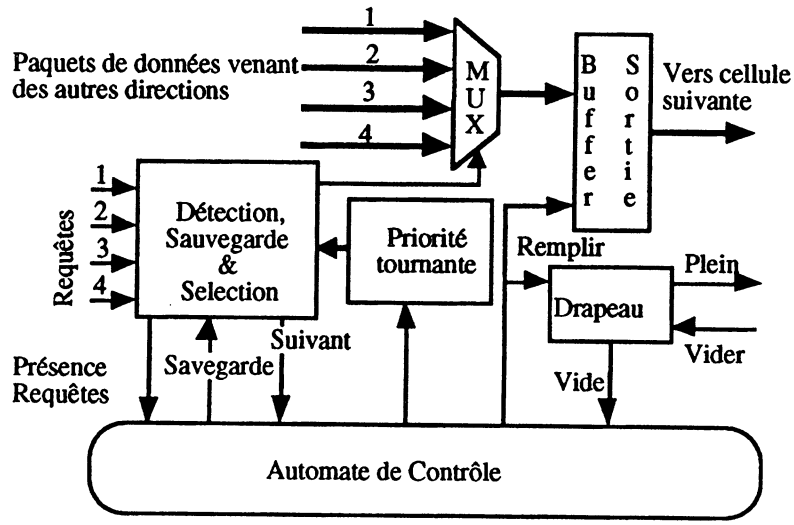


figure 2.22 : Synoptique du bloc de sortie

2.4.4.1 ÉCHANTILLONNAGE DES REQUÊTES

Au niveau de chaque tampon de sortie, on a la possibilité de recevoir 4 requêtes. Un registre de 4 bits mémorise les signaux de demandes d'accès venant de chacun des autres tampons d'entrée. Afin de minimiser le nombre des signaux d'entrée de l'automate de contrôle, une porte OU à quatre entrées nous procure un seul signal au lieu de quatre pour détecter la présence des requêtes.

La minimisation de ces signaux permet de réduire le temps de propagation du PLA de l'automate de contrôle et par la même occasion de minimiser sa surface.

Un signal appelé "Suivant" est introduit afin de différencier entre un premier paquet de message et les paquets suivants.

Au niveau du bloc de sortie et afin d'éviter une perturbation au niveau de l'arbitrage et de l'acheminement de la suite des paquets du message, la mémorisation des requêtes n'est effectuée que dans les trois cas suivants :

- Détection de présence de requêtes après un Reset.
- A la fin d'acheminement d'un message avec présence d'autres requêtes en attente.
- Détection de présence des requêtes après absence de requêtes.

2.4.4.2 ARBITRAGE ET PRIORITÉ TOURNANTE

La priorité tournante garantit, en premier lieu, la prévention des cas de famine. Avec une priorité fixe, une requête plus fréquente sur un canal risque de monopoliser le service du routage privant ainsi les requêtes sur les autres canaux.

L'arbitrage et la priorité tournante repose sur le principe d'interdiction de servir plusieurs fois une même requête d'une manière consécutive.

Son implémentation consiste en quatre bascules D, chacune est affectée à une requête. Elles mémorisent les requêtes qui viennent juste d'être servies. On appellera les sorties de ces bascules les prises en comptes P_i. En plus des bascules, on utilise un circuit purement combinatoire effectuant une sélection entre les quatre requêtes.

Le circuit de sélection (figure 2.24) ayant pour entrées les prises en compte ainsi que les requêtes sauvegardées, sert à établir le lien entre les demandes et les prises en compte, implémentant ainsi une priorité tournante tout en effectuant l'arbitrage.

Au départ, ou après un Reset, les quatre bascules de prise en compte sont initialisées à 1. Après l'asservissement d'une requête, la bascule correspondante est remise à zéro.

Au prochain coup (échantillonnage des requêtes), si la même requête est présente, elle ne peut être prise en compte même si elle possède une priorité importante par rapport aux autres car sa bascule correspondante est à l'état faux. En cas de présence des requêtes et toutes leurs prises en compte correspondantes sont à l'état faux, une réinitialisation de tout le registre est effectuée. Le principe de la priorité tournante est montré dans la figure 2.23.

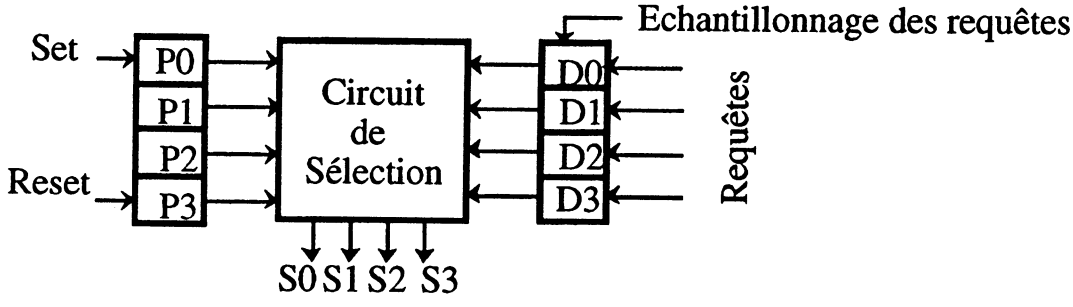


Figure 2.23: Priorité tournante

Les sorties du circuit de sélection servent de commandes pour les buffers trois états réalisant ainsi un arbitrage au niveau de l'accès au tampon de sortie.

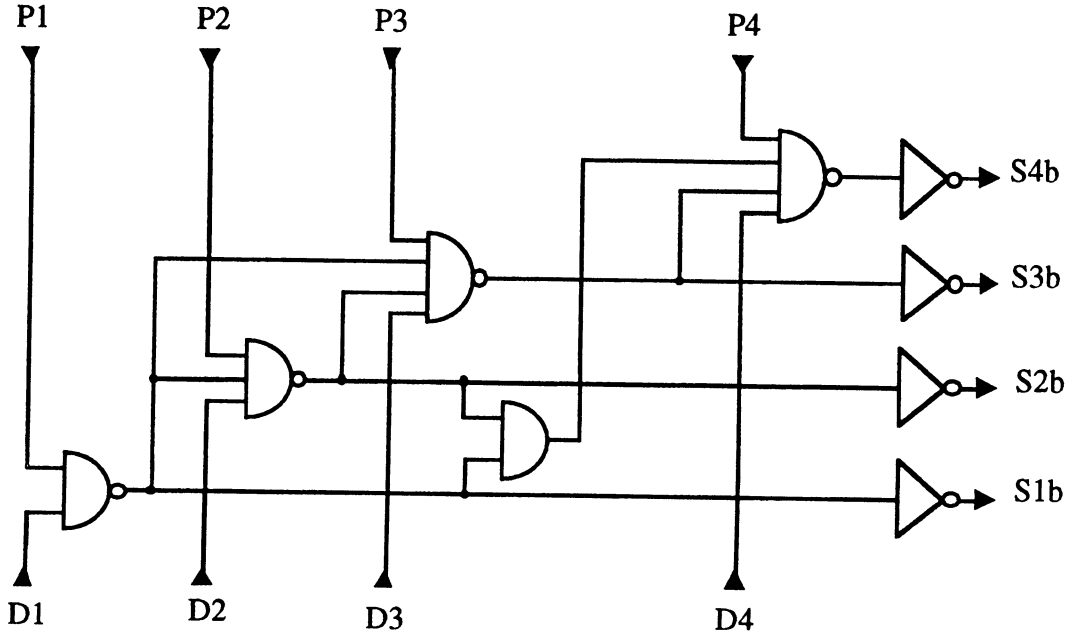


figure 2.24: Circuit de sélection

2.4.4.3 ACQUITTEMENT

Un tampon d'entrée ne doit être vidé qu'après avoir écrit sur le tampon de sortie choisi. L'envoi de l'acquittement sert à signaler au bloc d'entrée qu'on a terminé avec le paquet de message en cours et on peut passer au paquet suivant. Physiquement, l'acquittement n'est autre que le signal *Remplir* qui agit sur le drapeau du Tampon de sortie. Ce même tampon sert de Tampon d'entrée pour la cellule voisine.

2.4.4.4 AUTOMATE DE CONTRÔLE

L'automate de contrôle du bloc de sortie est montré dans la figure 2.25. De la même manière que l'automate du bloc fonctionnel d'entrée, deux états sont nécessaires pour acheminer un paquet. Soit quatre cycles au total pour acheminer un paquet de message d'un tampon d'entrée à un tampon de sortie.

Un signal nommé Z, est introduit afin de déterminer le moment de la remise à zéro les bascules de prise en compte pour la priorité tournante. Ceci étant dans le cas où il y a présence de messages mais les prises en compte

associées ne sont pas à leur état vrai. Bien sûr, un cycle sera perdu pour la réinitialisation des bascules de prise en compte.

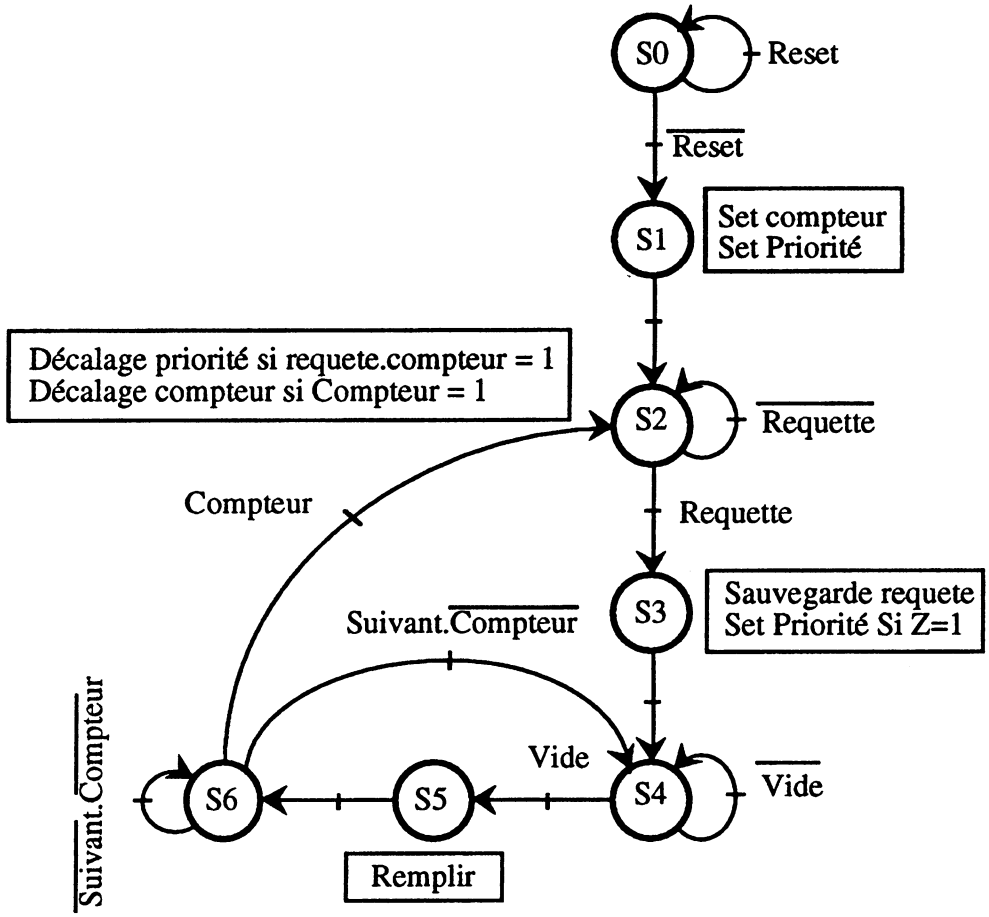


Figure 2.25: Graphe de l'automate de contrôle du bloc de sortie

Le synoptique détaillé du circuit global du bloc de sortie est montré dans la figure 2.26.

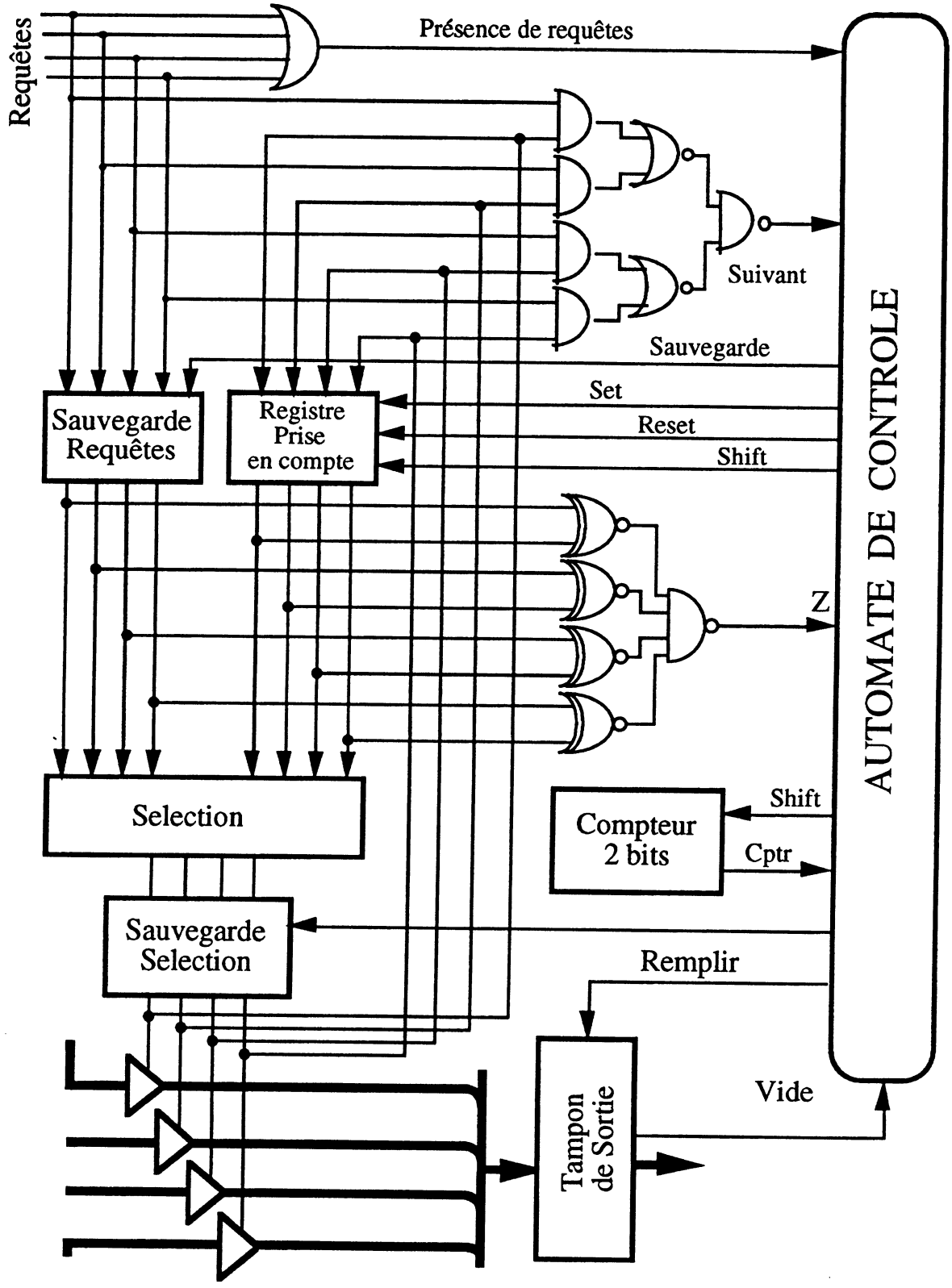
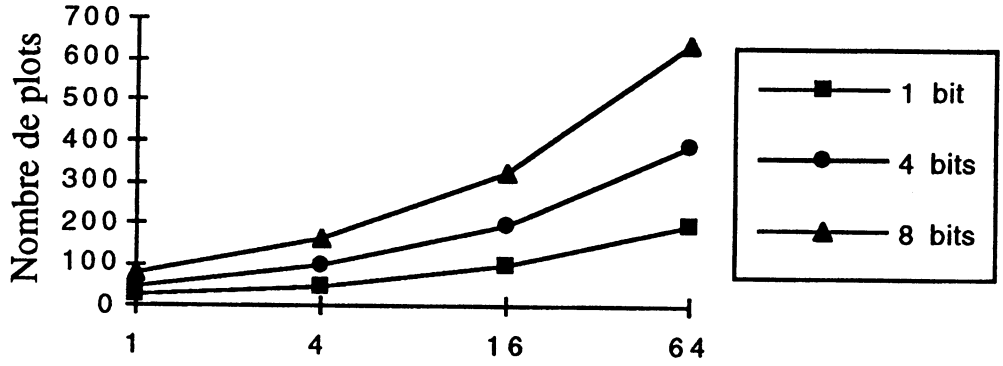


Figure 2.26: Synoptique détaillé du bloc de sortie

2.4.5 MULTIPLEXAGE DES PLOTS

Intégrer plusieurs processeurs 8 bits dans un même circuit VLSI pose le problème des plots qui augmentent en nombre avec celui des processeurs. Pour un circuit contenant un seul processeur, 20 plots sont nécessaires sur chacun des cotés en plus des signaux d'alimentation et de contrôle. La figure 2.27 montre les courbes d'accroissement des plots en fonction du nombre de processeurs intégrés dans un même boîtier (seul les plots concernant les alimentations, horloges et transparences sont supposés constants).

Croissance des plots



Nombre de processeurs

Envisager plusieurs processeurs, cela exige le recours au multiplexage des plots des signaux qui concernent la communication. Ceci pose le problème du contrôle du multiplexage qui peut se faire par les trois manières suivantes :

- Un contrôle externe à la cellule, au niveau de la réalisation d'un circuit, on perd la notion de la répététivité par juste duplication et juxtaposition des cellules sur silicium à l'intérieur du boîtier.

- Si on envisage de faire le contrôle du multiplexage à l'extérieur du circuit par l'ordinateur hôte, une partie de la gestion de la communication sur l'ensemble devra être centralisée. Cela augmente la complexité de l'interface entre le réseau et l'ordinateur hôte tant en matériel qu'au niveau logiciel.

• Le contrôle peut se faire par un circuit spécifique. Cela aussi nous fait perdre, au niveau des cartes, la notion de connexion entre circuits par juste juxtaposition des circuits.

• On peut imaginer un contrôle extra-cellule mais à l'intérieur d'un circuit en exploitant les signaux de protocole *remplir* et *vide* (solution retenue pour l'instant). Ceci entraîne un ralentissement de la communication entre les cellules de bord dû au multiplexage et la nécessité de câbler un mécanisme de gestion d'une file d'attente au bord d'un circuit.

Les figures 2.28, 2.29 et 2.30 montrent le multiplexage des signaux des données au niveau des tampons du bord.

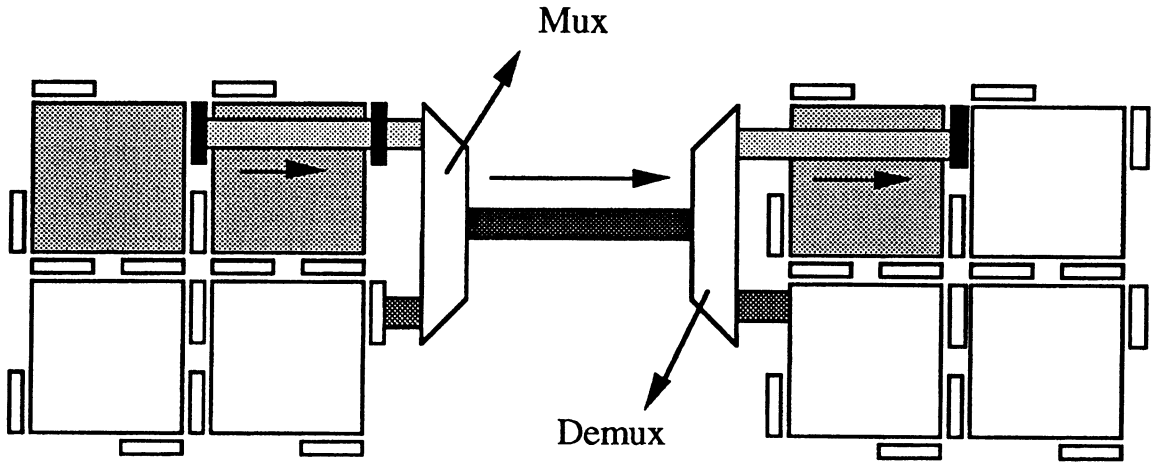


Figure 2.28 : Multiplexage et démultiplexage entre deux circuits

La sélection du multiplexage est pilotée par les deux signaux *remplir* et *vide* tampon de bord. Elle est basée sur le premier arrivé qui est le premier servis. Dès que l'un des deux *remplir* est actif, le signal *vide* de l'autre cellule est forcé à l'état 0 pour bloquer d'éventuels messages de l'autre cellule. La mise à l'état *plein* du tampon de la deuxième cellule est située seulement au niveau du mécanisme du multipexage et pas au niveau du drapeau. Le forçage dure jusqu'à ce que les trois paquets de message sont copiés sur les tampons d'entrée de la cellule suivante. La figure 2.29 montre la synoptique du dispositif de démultiplexage. Le registre à décalage est utilisé comme un compteur modulo 3 pour savoir la fin d'un message. Le rôle des deux bascules (D2 et D3) est de forcer l'état vide de l'un des deux tampons. La bascule D1 sert à mémoriser la

sélection du démultiplexage lors du transfert car un signal *remplir* n'est actif pendant qu'un cycle d'horloge pour un paquet.

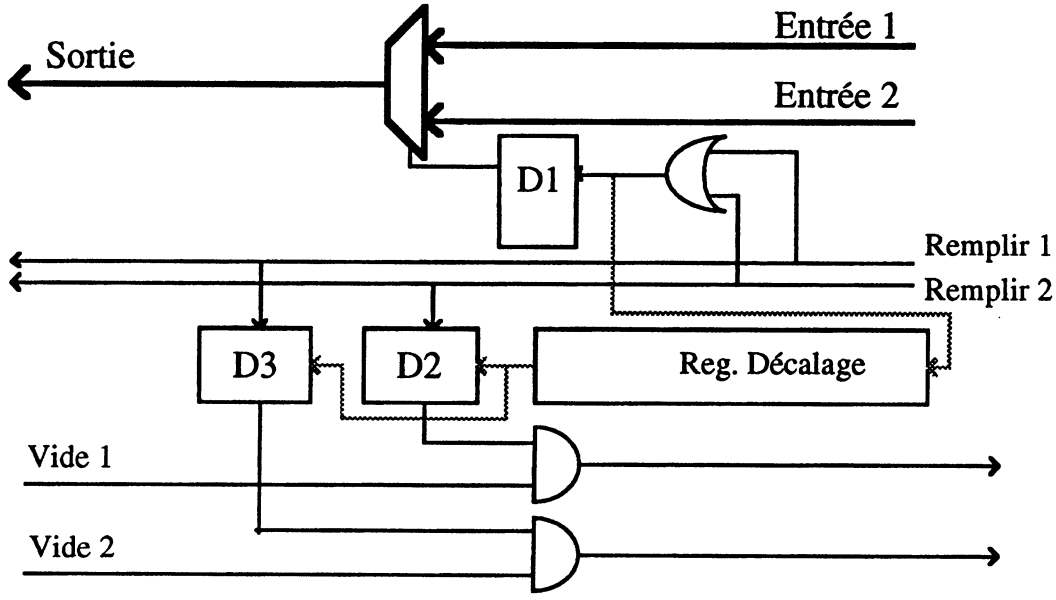


Figure 2.29 : Multiplexage

Le démultiplexage est aussi contrôlé par les deux signaux *remplir* et *plein* et repose sur le même principe que le multiplexage. La seule différence est que le démultiplexeur remplace le multiplexeur comme c'est montré dans la figure 2.30.

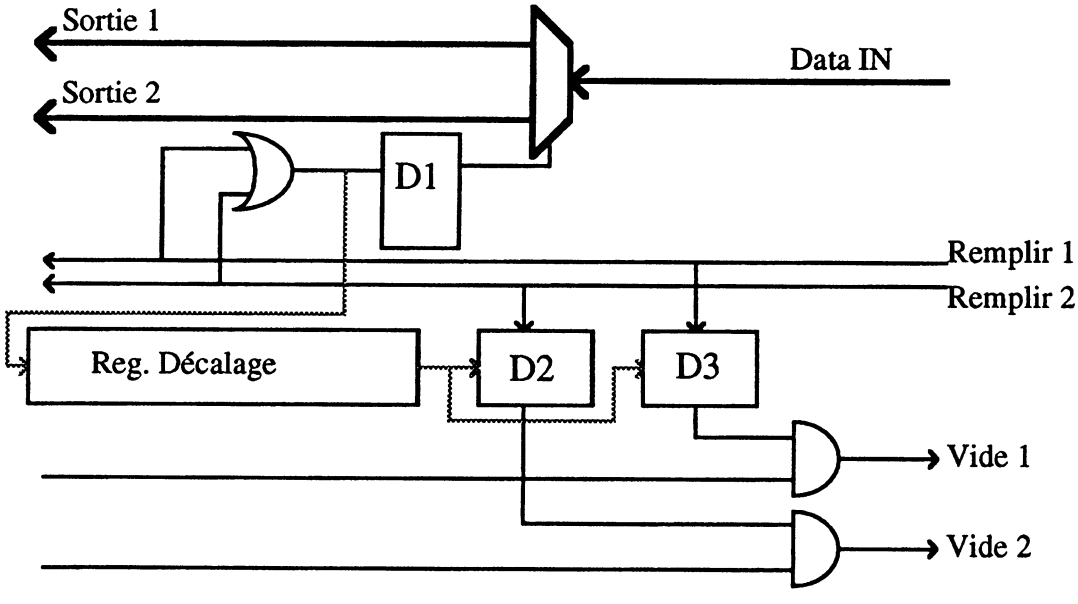


Figure 2.30 Démultiplexage

2.5 CONCLUSIONS ET PERSPECTIVES

L'utilisation du *wormhole* avec un transfert parallèl a permis d'améliorer le débit par rapport au routeur combinatoire. Cependant, on se heurte à la réduction des performances au niveau du temps d'acheminement d'un paquet de message à cause de la séquentialité dans la conception.

Il serait intéressant de surmonter cet handicape en remplaçant le routeur actuel par un routeur combinatoire tout en maintenant les précédentes approches, modes de communication et le mécanisme de commutation.

Bien que les simulations du réseau aient montré que le routeur est assez performant, les difficultés persistent toujours à cause de la faible connectivité et au multiplexage des plots quand le nombre de cellules augmente au sein d'un même circuit.

En exploitant le principe de multiplexage et de démultiplexage, ces deux dispositifs permettent d'introduire des modifications dans la stratégie générale d'acheminement des messages. En leur ajoutant des fonctionnalités supplémentaires, il serait possible alors de faire passer des messages en XY. Ceci, d'une part, augmente la connectivité et, d'autre part dans certains cas, réduit le temps d'acheminement globale d'un message.

Les technologies et techniques actuellement utilisées posent des limitations pour considérer des topologies à connectivité complète destinées à des architectures massivement parallèles. L'orientation vers des systèmes de communication des ordinateurs parallèles utilisant les fibres optiques ou des supports non filaires peuvent faire l'objet de lointaines perspectives.

On pourrai envisager des circuits avec une technologie mixte où la partie traitement est en silicium et la partie routage utilisant des diodes laser en technologie AsGa avec des fibres optiques. Des signaux de 20 Ghz sur des distances de 16 000 Km peuvent être obtenus sans amplification. Les limitations théoriques où les électrons réalisent de meilleurs performances est de l'ordre de 1 mm.

On pourrai imaginer 4 types d'interconnexions optiques : entre des unités optiques ou électroniques, entre les cartes, entre les circuits d'une même carte et entre les unités d'un même circuit.

Les interconnexions entre les cartes seraient réalisées par des fibres optiques où des amplificateurs seront alignés sur les cartes afin de compenser les "fanout" et les pertes [Als92].

En ce qui concerne les interconnexions entre les circuits VLSI, ces derniers seraient couplés à des fibres optiques à l'intérieur par des coupes en V sur la surface du circuit dans laquelle la fibre est étalée. Les circuits seront dotés de microlasers et de détecteurs. L'extrémité de la fibre optique reflète les rayons de lumière sur les détecteurs et les microlasers transmettent les rayons hors du circuit.

L'interconnexion par des fibres optiques à l'intérieur du circuit, seraient une solution très intéressante, car la lumière arrive à destination plus rapidement que les électrons à cause de l'absence des capacités à charger. Mais, cela nécessite une technologie mixte utilisant des substrats et des couches d'AsGa supplémentaires. Techniquement, ceci exige des moyens énormes et coûteux. Ce qui augmente la difficulté de réalisation au niveau technologique, d'une part, et, d'autre part, réduit le taux d'intégration [Als92].

En attendant des progrès en ce sens pour surmonter les différentes difficultés, on peut d'ores et déjà réfléchir à des architectures massivement parallèles, au niveau logiciel, avec une connectivité complète et une topologie variable.

CHAPITRE 3

ARCHITECTURE ET IMPLÉMENTATION DE LA PARTIE TRAITEMENT

3.1 ÉLABORATION ET ÉVALUATION DU JEU D'INSTRUCTIONS

Un jeu d'instructions découle généralement de la nature du traitement qu'on veut effectuer sur des données. Le concept du circuit programmable impose, lui même, déjà une large variété d'applications qu'on cherche à implémenter sur le réseau, donc plusieurs traitements. Ceci implique la nécessité de concevoir un jeu d'instructions.

Un programme est constitué d'une suite d'instructions. Chaque instruction inclut deux types d'information : d'une part l'opération à effectuer, et d'autre part les données et leur adresse. Cependant, il existe un ensemble de critères pour les différents choix intervenant dans la conception d'un jeu d'instructions d'une machine. Cet ensemble inclut principalement les points suivants :

- *Stockage des opérandes :* où peuvent-ils être rangés ailleurs qu'en mémoire ?

- *Nombre d'opérandes explicites cités par instruction :*combien d'opérandes sont nommés explicitement dans une instruction typique ?

- *Localisation de l'opérande :* où se trouve l'opérande, en mémoire ou ailleurs et comment est indiqué son emplacement ?

- *Opérations :* quelles sont les opérations disponibles dans le jeu d'instructions ?

- *Type et taille des opérandes :* quels sont le type et la taille de chaque opérande et comment sont-ils spécifiés ?

Les jeux d'instructions peuvent être donc classés selon ces cinq aspects. Cet ordre correspond à l'importance du rôle que joue chacun de ces critères dans les différents jeux d'instructions.

3.1.1 STOCKAGE DES OPÉRANDES

Le stockage interne est le point de différenciation le plus caractéristique : utilisation d'une pile, d'un accumulateur ou d'un ensemble de registres. De ce fait, les opérandes peuvent être adressés implicitement dans le cas d'une pile ou un accumulateur et explicitement dans le cas d'utilisation d'une suite de registres.

Les premières machines utilisaient des architectures à piles ou à accumulateurs. Les machines les plus récentes utilisent une architecture à registres généraux. Ce fait tient compte, d'une part, de la rapidité de cette forme de stockage par rapport à celle en mémoire et, d'autre part, la facilité et l'efficacité d'utilisation des registres lors d'une compilation [Joh92].

Bien sûr chacun de ces trois types de rangement présente des avantages et des inconvénients qui sont résumés dans le tableau suivant :

Stockage	Avantages	Inconvénients
Pile	Simplicité d'évaluation des expressions. Code compact.	Difficulté d'accès d'une manière aléatoire. Difficulté d'implantation efficace et possibilité d'aboutir à un goulet d'étranglement.
Accumulateur	État interne de la machine minimal. Code compact. Rapidité moyenne.	Trafic mémoire accumulateur dense.
Registre	Génération du code plus simple. Rapidité.	Instructions plus longues. Surface importante pour les réalisations en VLSI.

Tableau 3.1 : Comparaison des trois classes de rangement

3.1.2 NOMBRE ET LOCALISATION DES OPÉRANDES.

En ce qui concerne le deuxième et le troisième critères cités ci-dessus, ils touchent essentiellement aux modes d'adressage.

Les modes d'adressage sont des outils utilisés, par un programmeur travaillant avec des langages de haut niveau, pour effectuer une projection de la mémoire, sur la mémoire réelle de la machine. Ils permettent au programmeur de construire et d'utiliser des structures de données complexes et

multidimentionnelles alors que le matériel physique de la machine autorise la manipulation que des éléments linéaires et à taille fixe.

Bien sûr, plus le champ d'adresse est complexe et riche, moins sera le nombre d'instructions nécessaires pour extraire un élément d'une structure de données complexe. Néanmoins, un mode d'adressage complexe a tendance à augmenter la taille de l'instruction. Cela influe sur l'accroissement du temps d'exécution et complique l'unité de contrôle (sauf si son utilisation est très fréquente).

Il existe un large spectre de modes d'adressage utilisé dans les différents processeurs. Tous ces modes sont issus d'une combinaison de quelques objets et fonctions de base. Un objet peut être un registre ou un déplacement alors qu'une fonction peut être une addition, une indirection et/ou un décalage.

Le tableau suivant présente les modes d'adressage les plus utilisés dans les machines connues :

MODE D'ADRESSAGE	EXEMPLE	SIGNIFICATION	CAS D'UTILISATION
Registre	Add R4, R3	R4 <-- R4+R3	La valeur est dans un registre.
Immédiat	Add R4, #3	R4 <-- R4+3	Pour les constantes.
Déplacement	Add R4, 100 (R1)	R4 <-- R4+M[100+R1]	Accès aux variables locales.
Indirect par registre	Add R4, (R1)	R4 <-- R4+M[R1]	Accès utilisant un pointeur.
Indexé	Add R3 + (R1+R2)	R3 <-- R3+M[R1+R2]	Adressage des Tableaux. (R1= base du tableau, R2= valeur d'index)
Direct ou absolu	Add R1, (1001)	R1 <-- R1 + M[1001]	Accès variables statiques, Valeurs d'adresse grandes.
Indirect via mémoire	Add R1 , @(R3)	R1 <-- R1 + M[M[R3]]	L'adr. du pointeur p d'un tableau est en mémoire. Adresse = *p
Auto-incrémenté	Add R1, (R2)+	R1 <-- R1 + M[R2] R2 <-- R2 + d	Utile pour parcourir des tableaux dans une boucle (ordre croissant).
Auto-décrémenté	Add R1, -(R2)	R2 <-- R2 - d R1 <-- R1 + M[R2]	Utile pour parcourir des tableaux dans une boucle (ordre décroissant)
Indexé étendu	Add R1, 100(R2)[R3]	R1 <-- R1 + M[100 + R2 + R3 * d]	Pour indexer des tableaux.

Tableau 3.2 : Modes d'adressage

Bien que les processeurs fournissent un grand nombre de modes d'adressage, on n'en exploite le plus souvent qu'un sous-ensemble.

Dans le cas du réseau cellulaire, les exigences et les besoins en mode d'adressage sont différentes par rapport aux processeurs à usage général. La taille de la mémoire, locale ou partagée, doit être optimale pour répondre aux contraintes de l'efficacité de la communication et de la surface de silicium à ne pas gaspiller. La taille de la mémoire locale (256 octets) ne permet pas l'emploi des modes d'adressage résultant en un mot d'instruction long.

En ce qui concerne la communication, il nous faut non seulement des instructions nouvelles adaptées à l'architecture du réseau mais aussi avec des modes d'adressage permettant d'obtenir des instructions courtes en nombre d'octets et en nombre de cycles d'exécution.

Pour ce qui est de la surface, moins les modes d'adressage sont complexes, moins on aura besoin de rajouter des registres tampons et des opérateurs de calcul. Il faut noter également que la partie adresse d'une instruction influe considérablement sur l'algorithme d'interprétation d'une instruction et du même coup sur la simplicité du séquenceur du processeur.

Il faut donc retenir un ensemble de modes d'adressage le plus simple possible d'une part et, d'autre part, le plus efficace possible. Les modes retenus sont les suivants :

- Implicite.
- Absolu court.
- Immédiat.
- Absolu.
- Indirect.
- indirect post-incrémenté.

Cet ensemble de modes, découle aussi de l'étude et du raffinement du jeu d'instructions décrits dans les paragraphes suivants. La présence des modes inhérent, immédiat et absolu apparaît nécessaire dans les jeux d'instructions des processeurs moins spécialisés. Le mode indirect a fait valoir sa place à cause de la nature des structures de données manipulées par les cellules. Parmi les différentes applications programmées, 58% d'entre elles utilisent des tables de données comme entrées et des valeurs intermédiaires et/ou comme sorties [Rub92]. Les raisons de l'introduction de la forme courte du mode absolu sont présentées en détail dans le paragraphe suivant car il est difficile de dissocier les choix des opérations à effectuer de leurs modes d'adressage.

3.1.3 LES OPÉRATIONS

Le quatrième critère important dans la conception des jeux d'instructions concerne les opérations. Dans la plupart des jeux d'instructions, les opérateurs disponibles peuvent eux aussi bénéficier d'un classement :

- Arithmétique et logique.
- Transfert de données.
- Contrôle.
- Chaînes.
- Système.
- Flottant.

Cependant, dans le cas d'un processeur massivement parallèle, et pour mieux se situer dans le contexte de l'architecture originale de la machine, il faut inclure d'autres fonctionnalités en plus de celles déjà existantes dans les processeurs classiques :

• La première concerne la synchronisation entre cellules. L'approche du parallélisme massif et la taille du grain impliquent très souvent la décomposition des processus de base en sous-processus, d'une part, à cause de la limitation de la taille de la mémoire, et d'autre part, afin d'éviter les interblocages au niveau logiciel. Pour répondre à cela, il faut inclure des instructions du type *"envoyer un message"* et *"recevoir un message"*.

• La seconde fonctionnalité intéressante dans ce contexte est celle de la multiprogrammation au niveau d'une cellule. Elle a pour but de permettre d'abord une prévention des interblocages liés à la synchronisation des cellules et l'affectation de plusieurs processus à chaque cellule assurant ainsi un équilibrage de charge des cellules. Elle permet également la possibilité du réordonnancement au sein d'un même processus des actions d'une manière dynamique en fonction des conditions relatives à leur exécution.

• Et enfin, la programmation des communications nécessite l'élaboration des fonctionnalités suivantes :

- La réception : (canal) → valeur
  Tant que l'indicateur de présence est nul : attendre
  Remettre à zéro l'indicateur de présence
  Renvoyer le contenu du canal

- L'émission : (adresse cellule, canal, valeur)
  Tant que le tampon de sortie est plein attendre
  Générer un message [adresse, canal, valeur]

- Le test : (canal) → booléen
  Renvoyer l'indicateur de présence du canal.

Cette dernière fonction permet de réaliser une multiprogrammation par scrutation circulaire de la calculabilité des processus. Un processus est dit calculable si ses données sont présentes et si ses résultats sont demandés.

Pour être proche d'un processeur à usage général et s'adapter à l'architecture massivement parallèle, il nous faut des instructions qui permettent de réaliser :

- La communication inter-cellules,
- Des opérations arithmétiques et logiques,
- Des transferts de données,
- Du contrôle.

Mise à part les instructions de communication, l'équipe a été, dans un premier temps, inspirée du jeu d'instructions du MC 6800 et du MC 6802. Le jeu d'instructions initial (tableau 3.3) regroupait seulement les modes d'adressage "accumulateur pour les opérations unaires, immédiat et absolu pour les opérations binaires".

Les instructions de calcul binaires sont celles présentes dans le 6800. Elles impliquent toutes l'accumulateur comme l'un des opérandes et comme destinataire du résultat. Les instructions unaires regroupent les décalages du 6802 plus une instruction de complément à 1, qui portent toutes sur l'accumulateur.

Ces opérations positionnent les indicateurs du registre d'état qui sont exploités par les instructions de branchement *Bcc* et par l'instruction de positionnement conditionnel *Scc*. Les conditions de branchement sont celles définies dans le 68000 (tableau 3.4).

mnémo	instruction	modes	opération	flags
LDA	Chargement de l'accumulateur	abs,imm	A:=D	nz
STA	Rangement de l'accumulateur en mémoire	absolu	M[adr]:=A	
ADD	Addition	abs,imm	A:=A+D	nvzc
ADC	Addition avec retenue initiale	abs,imm	A:=A+D+c	nvzc
SUB	Soustraction	abs,imm	A:=A-D	nvzc
SBC	Soustraction avec emprunt	abs,imm	A:=A-D-c	nvzc
CMP	Comparaison	abs,imm	A-D	nvzc
AND	Et bit à bit	abs,imm	A:=A&D	nz
OR	Ou bit à bit	abs,imm	A:=A ! D	nz
XOR	Ou exclusif bit à bit	abs,imm	A:= A ⊕ D	nz
NOT	Complément à 1	acc	A:= !A	nz
ASL	Décalage arithmétique à gauche	acc		nvzc
LSR	Décalage à droite	acc		nzc
ROL	Rotation gauche avec la retenue	acc		nzc
ROR	Rotation droite avec la retenue	acc		nzc
CLC	Mise de la retenue à 0	implicite	c:=0	c
SEC	Mise de la retenue à 1	implicite	c:=1	c
SEND	Envoi de message	abs	tant que s attendre OUT:=M[adr ...adr+2] s:=1	s
GET	Réception bloquante	abs	Tq ¬ présent attendre A:= M[adr] présent:=0	
PUT	Envoi d'un message interne	abs	M[adr]:=A, présent := 1	
TRY	Test d'un canal	abs	PC:=M[0] si msg absent	
EXIT	Positionnement du vecteur d'échec de TRY	abs	M[0]:= adr	
Bcc	Branchement conditionnel	relatif	PC:=PC+depl si cc=vrai	
Scc	Positionnement conditionnel	acc	A:= si cc alors -1 sinon 0	
JMP	Branchement	abs	PC:=adr	
JSR	Appel de sous-programme	abs	M[1]:=PC+2, PC=adr	
RTS	Retour de sous-programme	implicite	PC:=M[1]	

Tableau 3.3 : Jeu d'instructions initial

Ce jeu d'instructions initial a été utilisé comme plate forme de base pour étudier et concevoir un autre jeu d'instructions adapté à l'architecture et aux différentes applications lourdes à tourner sur des machines séquentielles. L'étude a été portée sur deux aspects :

• Le premier concerne la programmation des différentes applications (utilité de chaque instruction et de ses modes d'adressage) susceptibles d'être implémentées sur un tel réseau cellulaire. Cette étude a été faite et présentée par Pascal Rubini dans sa thèse [Rub92] en se servant d'un simulateur de sa propre conception.

• Le deuxième aspect de l'étude concerne le coût en matériel du raffinement du jeu d'instructions. Cette étude consistait à évaluer les surfaces en silicium et le nombre de cycles nécessaires à l'exécution (complexité de l'algorithme d'interprétation).
Ces évaluations de surface ont été portées notamment sur la partie contrôle (généralement plus de 50% de la taille d'un processeur). Ces évaluations dépendent des rajouts et éliminations de quelques instructions en se servant d'un outil de génération automatique des parties contrôle ("state machine compiler") intégré dans le système VTI (appellation actuelle Compass).
Cet outil offre la possibilité de décrire, d'une manière textuelle, le graphe de contrôle de l'automate de séquencement. A partir d'une bibliothèque de cellules pré-caractérisées, la surface et le chemin critique sont déduits.

Les instructions dont le coût en matériel est important complexité élevée et le taux d'occurrence dans les programmes des différentes applications étudiées faible devaient être éliminées du tableau du jeu d'instructions.

cc	nom	Après CMP #val	décomposition
BCC	Carry Clear	A ≥ val (non signé)	C=0
BCS	Carry Set	A < val (non signé)	C=1
BEQ	Equal	A = val	Z=1
BNE	Not Equal	A ≠ val	N=0
BGE	Greater or Equal	A ≥ val (signé)	N ⊕ V =0
BGT	Greater than	A > val (signé)	Z+(N ⊕ V)=0
BHI	Higher	A > val (non signé)	N ⊕ V =1
BLE	Less or Equal	A ≤ val (signé)	Z+(N ⊕ V)=1
BLS	Less or same	A ≤ val (non signé)	C+Z=1
BLT	Less than	A < val (signé)	N ⊕ V =1
BMI	Minus		N=1
BPL	Plus		N=0
BVC	Overflow clear		V=0
BVS	Overflow set		V=1
SE	Send if buffer empty		S=0
SF	Send if buffer full		S=1

Tableau 3.4 : Instructions et conditions de branchement initiales

Dans le même sens, il fallait enrichir, si possible, le jeu d'instructions par celles utilisant le même matériel existant dans le chemin de données. Il s'agissait de chercher des instructions qui ne compliquent pas trop l'algorithme d'interprétation et qui permettent d'obtenir un jeu d'instructions plus riche et plus compact (comme dans le cas de la multiplication). Ceci étant pour répondre à la contrainte de la faible taille de la mémoire locale.

Il fallait aussi se demander si toutes les fonctionnalités classiques sont nécessaires. Avec le parallélisme à grain fin, on a tendance à travailler avec des processus de petites tailles contenant très rarement des structures internes complexes comme les appels de sous-programmes, routine multiplication, etc.

L'étude menée par [Rub92] sur les différents codes des applications programmées qui s'exécutent sur le réseau a montré que les positions mémoires référencées dans les adressages absolus sont souvent les mêmes. Elles correspondent principalement aux :

- canaux d'entrée,
- structures de message pour SEND,
- variables d'état,
- variables temporaires,
- compteurs de boucle.

Une instruction dans un adressage absolu prend plus d'une position mémoire. Le programme occupe donc une large partie de l'espace mémoire.

En se basant sur le jeu d'instructions initial, les codes instructions (en terme de codes hexadécimaux) disponibles ne sont pas tous exploités. Et afin d'optimiser l'accès aux variables en espace mémoire et en temps, un adressage absolu "court", dans lequel l'argument serait inclus dans le code instruction, a été introduit (figure 3.1).

a) instructions à adressage absolu

b) instruction à adressage absolu court

Figure 3.1 : Format des instructions

Le gain réalisé par l'utilisation du mode absolu court est considérable du point de vue espace mémoire et en temps d'exécution puisqu'on gagne un cycle pour le "fetch address". Pour cela, il faut se référer au tableau 3.5 qui résume le taux d'utilisation de la forme courte dans les différentes applications. Le mécanisme d'exploitation du champ adresse de l'instruction pendant le cycle "fetch instruction" est expliqué dans le paragraphe 3.4.1.2 (Implémentation : Partie adressage).

	absolu court	absolu
LDA	97%	3%
STA	87%	13%
GET	79%	21%
PUT	100%	-
TRY	97%	3%

Tableau 3.5 : taux de la forme courte.

Chaque remplacement d'une instruction dans le mode absolu par la forme courte se traduit en terme de performances par le gain d'un cycle. Le décodage de 16 codes, pour chacune des instructions citées dans le tableau 3.5, prévue pour toute une page, se traduit également par l'ajout de quelques monômes supplémentaires dans les PLAs dans la partie contrôle.

La forme courte peut être envisagée pour d'autres instructions comme les branchements relatifs et celles en mode immédiat. Il faut analyser donc la rentabilité des codes d'instructions, d'une part, et, d'autre part, leur disponibilité. Pour attribuer la forme courte à une instruction, il faut prévoir au moins 16 codes instructions.

En ce qui concerne les branchements relatifs, la portée ne dépasse pas les 16 positions mémoire [-8,+7]. En plus de cette limitation, le taux d'occurrence des branchements est faible dans les programmes (tableau 3.6).

Le mode immédiat court, suggéré pour les instructions LDA, ADD et SUB a fait l'objet d'un rejet vu leur taux d'occurrence dans les différents programmes [Rub92].

instruction	% d'occurrences
STA	17%
LDA	14%
SEND	9%
GET	6%
DEC	2,8%
INC	2,3%
CMP	2,3%
TRY	1,8%
autres	< 1,5%

Tableau 3.6 : taux d'occurrence des instructions

L'adresse relative dans les branchements conditionnels est aussi encombrante qu'une adresse absolue. En plus, elle nécessite un opérateur de calcul (additionneur) auprès du compteur ordinal. L'une des utilités des branchements relatifs est d'écrire des codes transportables. Cette caractéristique n'est d'aucun intérêt dans notre contexte. Les branchements relatifs sont transformés en branchements absolus.

Le faible intérêt que présente la notion de sous-programmes dans le contexte du réseau cellulaire a conduit à la suppression des instructions JSR et RTS qui font partie des instructions complexes à implémenter. Pour ne pas se priver de quelques petites routines, ces deux instructions on été remplacées par TPCA (transfert du compteur ordinal dans l'accumulateur) et TAPC (même opération dans le sens inverse) qui sont simples à implémenter.

En ce qui concerne les opérations de calcul arithmétique et logique, seules les opérations unaires ont fait l'objet de quelques modifications :

- Les décalages : l'instruction LSR (décalage à droite avec introduction d'un zéro) a été remplacée par ASR (décalage à droite avec conservation du bit de signe). Cette instruction, peut réaliser une division (signée) par deux .

- Les instructions DEC, INC et CLR (liées à l'absence du mode immédiat court) ont été introduites.

• Les modifications introduites dans le chemin de données ont conduit à doter le bloc de calcul d'un registre supplémentaire et à rectifier les interconnexions UAL-registres-Bus (figure 3.2).

a) Bloc de calcul initial *b) Bloc de calcul final*

Figure 3.2 : Bloc de calcul

Ceci a nourri la volonté d'augmenter les cellules en capacités de calcul. Avec la nouvelle structure du bloc de calcul, il est facile de réaliser une multiplication micro-codée (8x8→16) basée sur l'algorithme "additions/décalages". L'instruction multiplication est de la forme :

Mul adr1, adr2 A,B:=M[adr1] * M[adr2]

Ceci nécessite la préparation des deux opérandes en mémoire au préalable car la multiplication est dotée d'un mode d'adressage unique : l'absolu.

• Enfin, pour mettre pleinement à profit la présence de l'accumulateur B, il était préférable de doter la cellule d'une capacité de calcul 16 bits plutôt que de s'en tenir à un load/store B. En concaténant les accumulateurs A et B, les instructions de calcul de 16 bits sont :

- ADCW addition avec retenue de AB avec une données de 16 bits,
- SBCW soustraction avec emprunt de AB avec une de données de 16 bits,
- LDAW chargement de AB avec une données de 16 bits,
- STAW stockage de AB dans deux positions mémoires consécutives.

Ces opérations sont exécutées en deux phases. Chaque phase traite 8 bits vu la taille des bus de données et d'adresses. Grâce à ces 4 instructions, ainsi qu'à la multiplication, il est donc possible d'effectuer des multiplications 16x16 bits.

3.1.4 TYPE ET TAILLE DES OPÉRANDES

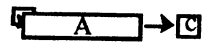
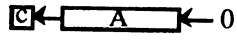
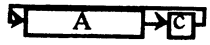
Généralement, le type d'un opérande - par exemple, entier, flottant simple précision, caractère - donne effectivement sa taille. Les types courants d'opérandes incluent le bit, le caractère (octet), le demi-mot (16 bits), le mot (32 bits), le flottant simple précision (un mot) et le flottant double précision (deux mots).

Dans les machines puissantes actuelles, l'accès aux types de données mot ou double est le plus dominant. L'approche intermédiaire entre la "Connection Machine" et le "Transputer", en plus de nos inspirations des processeurs classiques comme le 6800 ou 6502, la taille de l'octet apparaît comme un compromis bien adapté. Seul, l'espace d'adressage peut être limité par ce choix.

Avec un bus d'adresse de 8 bits, seulement 256 positions mémoires peuvent être utilisées pour constituer la mémoire locale. Avec une taille du bus d'adresse de 16 bits, on pourra se permettre d'avoir une mémoire de 64K octets qui résoudrait pas mal de problèmes dans les modes d'adressage et la compacité du jeu d'instructions. Ceci nous priverait de maintenir l'homogénéité de l'architecture (8 bits) et la satisfaction de voir plusieurs processeurs s'intégrer dans un même boîtier.

3.1.5 JEU D'INSTRUCTIONS FINAL

Le jeu d'instructions final est montré dans les tableaux 3.7 et 3.8. Les tables des instructions en termes de code, du nombre de cycles et du nombre de positions mémoire occupées sont données en annexe 1. La description détaillée de l'exécution de chaque instruction en évoquant le matériel physique impliqué (registres, opérations, bus, cycles et modes d'adressage) est également donnée en annexe 2.

mnémo	instruction	modes	opération	flags
LDA	Chargement de l'accumulateur	abs,imm, page 0, (I), (I)+	A:=D	nz
STA	Rangement de l'accumulateur en mémoire	abs,pg 0, (I), (I)+	M[adr]:=A	
ADD	Addition	abs,imm,(I),(I)+	A:=A+D	nvzc
ADC	Addition avec retenue initiale	abs,imm,(I),(I)+	A:=A+D+c	nvzc
SUB	Soustraction	abs,imm,(I),(I)+	A:=A-D	nvzc
SBC	Soustraction avec emprunt	abs,imm,(I),(I)+	A:=A-D-c	nvzc
CMP	Comparaison	abs,imm,(I),(I)+	A-D	nvzc
AND	Et bit à bit	abs,imm,(I),(I)+	A:=A&D	nz
OR	Ou bit à bit	abs,imm,(I),(I)+	A:=A | D	nz
XOR	Ou exclusif bit à bit	abs,imm,(I),(I)+	A:= A ⊕ D	nz
NOT	Complément à 1	abs,acc,(I),(I)+	D:= !D	nz
ASR	Décalage arithmétique à droite	abs,acc,(I),(I)+		nvzc
ASL	Décalage à gauche	abs,acc,(I),(I)+		nzc
ROL	Rotation gauche avec la retenue	abs,acc,(I),(I)+		nzc
ROR	Rotation droite avec la retenue	abs,acc,(I),(I)+		nzc
CLR	Remise à zéro	abs,acc,(I),(I)+	D:=0	z
DEC	Décrémentation	abs,acc,(I),(I)+	D:=D-1	nvzc
INC	Incrémentation	abs,acc,(I),(I)+	D:=D+1	nvzc
TST	Comparaison avec 0	abs,acc,(I),(I)+	A,D-0	znc
TAI	Transfert Acc → I	implicite	I:= A	
TIA	Transfert I → Acc	implicite	A:= I	
LDI	Chargement de I	abs,imm,(I)	I:= D	
CLC	Mise de la retenue à 0	implicite	c:=0	c
SEC	Mise de la retenue à 1	implicite	c:=1	c
Bcc	Branchement conditionnel	relatif	PC:=PC+depl si cc=vrai	
TAPC	Transfert Acc → PC	implicite	PC:= Acc	
TPCA	Transfert PC → Acc	implicite	Acc:= PC	
SEND	Envoi de message	abs,(I),(I)+	tant que s attendre OUT:=M[adr ...adr+2] s:=1	
GET	Réception bloquante	abs,pg F,(I),(I)+	Tq ¬ présent attendre A:= M[adr] présent:=0	
PUT	Envoi d'un message interne	abs,pg F,(I),(I)+	M[adr]:=A, présent := 1	
TRY	Test d'un canal	abs,pg F,(I),(I)+	PC:=M[0] si msg absent	

MUL	Multiplication. 8x8 → 16	abs (adr1,adr2)	Acc * MQ → Acc, MQ	
LDAW	Chargement 16 bits	abs,imm,(I),(I)+	Acc, MQ := D16	zn
STAW	Sauvegarde 16 bits	abs,(I),(I)+	D16 := Acc, MQ	
ADCW	Addition 16 bits	abs,imm,(I),(I)+	Acc,MQ+ D16 → Acc,MQ	
SBCW	Soustraction 16 bits	abs,imm,(I),(I)+	Acc,MQ- D16 → Acc,MQ	

Tableau 3.7 : Jeu d'instructions final

cc	nom	Après CMP #val	décomposition
BRA	Branch		
BCC	Carry Clear	A ≥ val (non signé)	C=0
BCS	Carry Set	A < val (non signé)	C=1
BEQ	Equal	A = val	Z=1
BNE	Not Equal	A ≠ val	N=0
BGE	Greater or Equal	A ≥ val (signé)	N ⊕ V =0
BGT	Greater than	A > val (signé)	Z+(N ⊕ V)=0
BHI	Higher	A > val (non signé)	N ⊕ V =1
BLE	Less or Equal	A ≤ val (signé)	Z+(N ⊕ V)=1
BLS	Less or same	A ≤ val (non signé)	C+Z=1
BLT	Less than	A < val (signé)	N ⊕ V =1
BMI	Minus		N=1
BPL	Plus		N=0
BVC	Overflow clear		V=0
BVS	Overflow set		V=1

Tableau 3.8: Instructions et conditions de branchement

3.2 ARCHITECTURE DE LA PARTIE OPÉRATIVE

Les chemins de données dans les microprocesseurs sont classés en fonction des types et organisations des registres internes, de la mémoire locale (si elle existe) et de la manière de leur interconnexion. On distingue cinq types d'organisation classiques des parties opératives :

- Structure à registres,
- Structure à accès unique à la mémoire locale,
- Structure à micro-accumulateur,
- Structure à mémoire double accès,
- Structure à éléments standards.

Nous commencerons par présenter très brièvement chacune de ces structures en mettant l'accent sur les avantages et les inconvénients de chacune d'elles. On présentera par la suite la structure proposée.

3.2.1 STRUCTURE À REGISTRES :

Cette structure consiste à utiliser un ensemble de registres connectés à l'aide de deux bus aux deux ports de l'unité arithmétique et logique. Une opération peut être donc effectuée sur des opérandes stockés dans n'importe quels registres. La forme de la micro-instruction est du type :

$$R_k \leftarrow R_i \ OP \ R_j$$

La sortie de l'UAL est rebouclée sur l'un des bus afin de pouvoir ranger le résultat dans l'un des registres contenant les opérandes ou l'un des registres restants. La figure 3.3 illustre ce premier type de chemin de données.

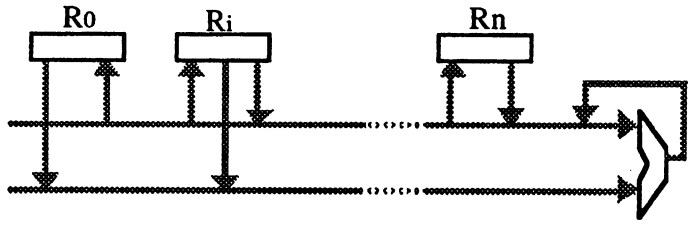


Figure 3.3: Structure à registres

L'avantage de cet arrangement réside dans la flexibilité de la manipulation des registres. Cette structure réduit la longueur du code exécutable puisque les opérations impliquant les données dans les registres ne nécessitent pas d'adresse. Un deuxième avantage est la rapidité d'exécution des opérations puisque les registres ont un accès direct à l'UAL. Il est donc possible d'accélérer l'exécution d'un code en maintenant les données les plus fréquentes dans les registres. Ces derniers doivent être sauvegardés en cas d'interruption ou de commutation de contexte dans un système multiprogrammé ou multiprocessus. Un nombre important de registres dans une partie opérative aboutit à un temps de chargement de contexte et de prise en charge d'interruption considérable.

3.2.2 STRUCTURE À ACCÈS UNIQUE À LA MÉMOIRE LOCALE

Cette structure utilise une mémoire locale au lieu d'une suite de registres. Deux registres tampon sont utilisés à l'entrée de l'UAL. La sortie de cette dernière est rebouclée sur le bus afin de pouvoir ranger le résultat dans la mémoire ou dans l'un des deux registres. Cette structure est illustrée en la figure 3.4.

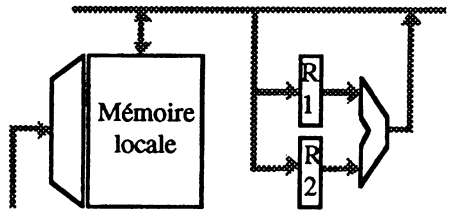


Figure 3.4: Structure mémoire locale à accès unique

3.2.3 STRUCTURE À MICRO-ACCUMULATEUR

Une autre manière de véhiculer les données, est d'utiliser une structure à micro-accumulateurs avec un simple ou double bus (figure 3.5). Cette organisation est très adaptée pour l'implémentation des opérations telles que la multiplication et la division. Les accumulateurs sont simplement des registres contenant implicitement des opérandes pour des instructions spécifiant l'adresse du deuxième opérande s'il s'agit d'une opération binaire (sauf si elle concerne un deuxième accumulateur).

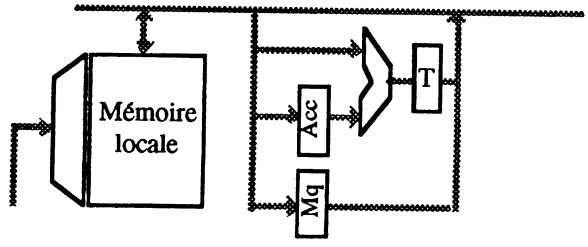


Figure 3.5 Structure à base de micro-accumulateur

3.2.4 STRUCTURE À MÉMOIRE DOUBLE ACCÈS

L'avantage d'utiliser une mémoire à double accès est de réduire les phases d'initialisation des registres avant d'effectuer une opération binaire impliquant l'UAL sur des mots mémoire (figure 3.6). Elle permet également de paralléliser et de faciliter les transferts des opérandes aux opérateurs. Cependant, elle présente l'inconvénient d'être encombrante à cause du double décodage de l'adresse et de l'arbitrage pour un éventuel accès à la même position mémoire, sans oublier un contrôle plus pointilleux.

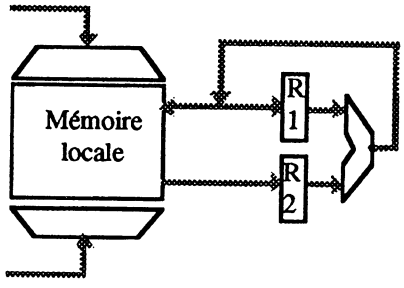


Figure 3.6 Structure à base d'une mémoire à double accès

3.2.5 STRUCTURE À ÉLÉMENTS STANDARDS

La structure à base d'éléments standards a été utilisée dans le MC 68000, dérivée de celle à mémoire à double accès. Elle consiste à se servir d'éléments standards comme une RAM à double accès, une ROM, deux registres et une UAL connectés à travers deux ou trois bus (figure 3.7). Cette structure a montré ses preuves dans le MC 68000 en termes d'efficacité, mais toutefois reste moins intéressante du point de vue surface pour un réseau cellulaire.

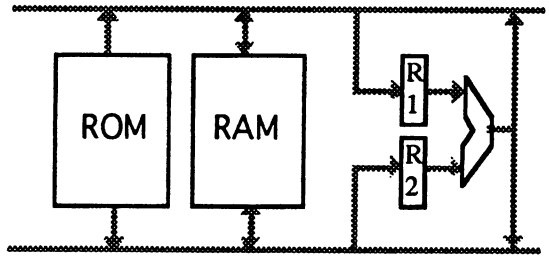


Figure 3.7: Structure à éléments standards

3.2.6 L'ARCHITECTURE DE LA PARTIE OPÉRATIVE PROPOSÉE

Le chemin des données proposé pour notre processeur présente une structure à deux bus (un bus d'adresse et un bus de données). Cette structure est très proche de celle à accès unique à la mémoire locale et de celle à base de micro-accumulateur. L'utilisation de la mémoire locale découle de l'approche architecturale globale. Le choix de sa taille (256 octets seulement) provient des contraintes de surface et de la volonté de garder une homogénéité de la taille des bus.

La structure à micro-accumulateur répond convenablement aux exigences du jeu d'instructions défini (surtout en ce qui concerne la multiplication). L'idée d'utiliser une mémoire à double accès n'a pas été retenue vu sa consommation en surface et la complexité de son contrôle.

L'idée initiale était de partir d'un chemin de données simple et classique, construit autour d'un bus de données et d'un bus d'adresse, comme celui du 6800 ou du 6502 (figure 3.8). Parallèlement au raffinement du jeu d'instructions, le chemin des données a lui aussi subit des modifications : révision du nombre de registres, ajout et suppression de quelques opérateurs, rectifications d'interconnexion entre registres. Pour chaque modification, nous avons effectué

des estimations des coûts induits par l'algorithme d'interprétation et la surface occupée par la partie contrôle.

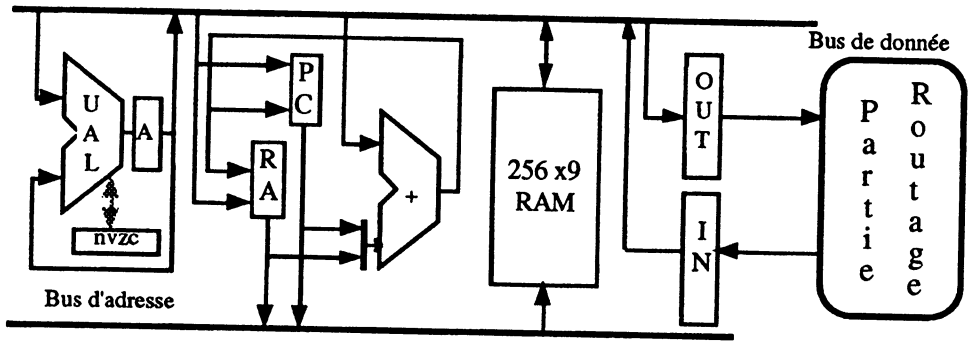


Figure 3.8 Chemin de données initial.

La partie opérative proposée est constituée de 4 blocs fonctionnels : le bloc de calcul, le bloc d'adressage, le bloc mémoire et le bloc d'interface routage. Hormis le bloc mémoire, les trois autres blocs ont subis des changements pour aboutir à un chemin de données qui selon les estimations et évaluations effectuées autant sur le plan programmation [Rub92] que celui physique, nous paraît capable de supporter les différentes contraintes d'un processeur massivement parallèle au sens capacité de traitement. La figure 3.9 montre la version finale de la partie opérative.

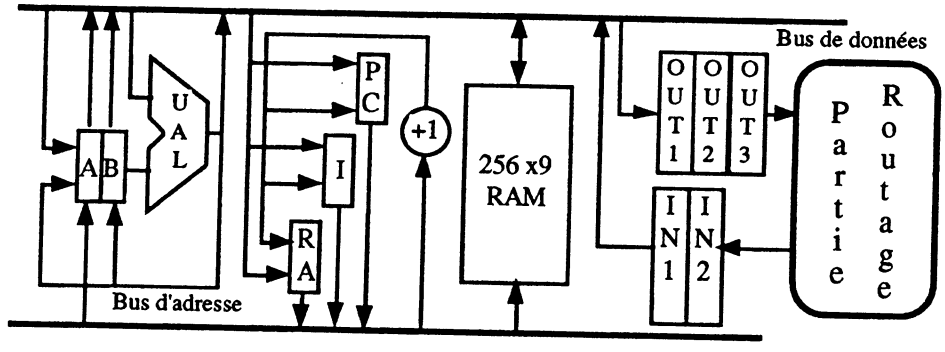


Figure : 3.9 Chemin de données final

3.2.7 L'UNITÉ ARITHMÉTIQUE ET LOGIQUE

Les UALs sont classées en fonction des types d'additionneurs qu'elles utilisent. Ces derniers peuvent être regroupés en trois catégories : les additionneurs séquentiels à propagation de retenue (qui ont un temps d'exécution de l'ordre du nombre de bits à additionner : O(n)), les additionneurs

à retenue bondissante ($O(\sqrt{n})$) et les additionneurs à anticipation de la retenue ($O(\log_2 n)$) [Mul89]. Il est intéressant de dresser un tableau comparatif des principaux types d'UAL étudiées (tableau 3.9). Cette étude comparative a été menée en utilisant une technologies CMOS 2.0 µm [Sou89].

	Type d'UAL	Surface	Rapidité	Remarques
Séquentielles	Ripple Carry	S	F	
	Domino (NMOS)	- 15*S%	2 x F	+ Horloge
	Manchester (NMOS)	- 9S*%	3 x F	+ Horloge
Retenue Bondissante	Carry Sélect	+ 57%*S	3 x F	+Commande
	Carry skip	+16%*S	3 x F	+Commande
Retenue Anticipée	Full Carry Look Ahead	+ 34%*S	4 x F	
	Ripple carry within group CLA between group	+20%*S	3 x F	
	CLA within group Ripple Carry between group	+25%*S	3,5 x F	
	Boent &Kung	+29%*S	2,5 x F	

Tableau 3.9: Tableau comparatif des UALs

(CLA = Carry Look Ahead, F= fréquence. Afin de faciliter la lecture, l'étude théorique et les schémas descriptifs de chaque type d'UAL présentés dans ce tableau, sont détaillés dans l'annexe 3).

Les résultats du tableau montre qu'une UAL à base d'un additionneur à propagation de retenue (ripple carry) est la plus compacte. Si on prend la contrainte de surface comme l'exigence la plus dominante, on n'aura pas d'autre choix à faire qu'une UAL à base d'additionneur "Ripple Carry", en plus de sa simplicité de conception et de réalisation.

3.3 ARCHITECTURE DE LA PARTIE CONTRÔLE

En général, le rôle de la partie contrôle d'un microprocesseur consiste, d'une part en la lecture, le décodage et l'exécution des instructions et d'autre part, à assurer la communication avec l'extérieur à travers les broches de contrôles. Le fonctionnement normal de la partie contrôle est l'exécution de l'algorithme d'interprétation des instructions.

Au niveau matériel, cela se traduit par la nécessité de disposer d'un registre d'instruction (IR) et d'un ensemble de circuits permettant de décoder les opérations et de déterminer l'état courant de l'algorithme afin de calculer les commandes appropriées pour la partie opérative. Il existe bien sûr une large variété de structures de cet ensemble de circuits. Les parties contrôle des microprocesseurs sont classées selon ces structures. Il est intéressant de faire un tour rapide de ces classes afin de situer l'architecture de la partie contrôle proposée pour notre processeur.

3.3.1 PARTIES CONTRÔLE À UN SEUL NIVEAU D'INTERPRÉTATION

Les parties contrôle à un seul niveau d'interprétation sont obtenues à partir de l'écriture de l'algorithme d'interprétation des instructions sous une forme directement traduisible en matériel, de la décomposition en étapes élémentaires correspondant aux états de la machine et le recensement de l'ensemble des commandes de la partie opérative à activer au cours de cette étape ainsi que la prise en compte de l'ensemble des informations permettant de déterminer l'étape suivante.

L'algorithme décrivant le niveau matériel est généralement obtenu lors de la conception de la partie opérative.

3.3.1.1 PARTIES CONTRÔLE CÂBLÉES

Les parties contrôle câblées constituent la structure de contrôle utilisée dans les tout premiers circuits VLSI tel que 4004 et 8008 d'Intel ou bien le 6800 de Motorola. Ce type de contrôle est donc obtenu par une traduction directe de l'algorithme d'interprétation en matériel. Un élément de mémorisation comme une bascule D ou un registre maître-esclave est associé à chaque état de l'algorithme.

Chaque commande de la partie opérative est calculée à l'aide d'un circuit purement combinatoire à base de portes NAND et NOR (figure 3.10). Les entrées du circuit combinatoire sont constituées des sorties des bascules correspondant aux états dans lesquels cette commande doit être activées ainsi que des conditions d'activation et comptes rendus venant de la partie opérative.

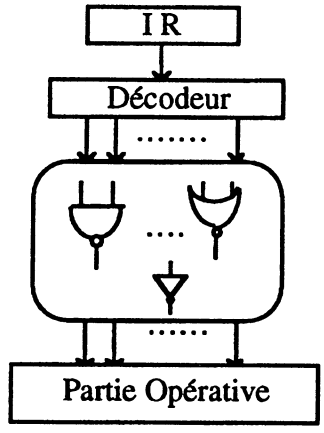


Figure 3.10 : Partie contrôle câblée

Il est important de noter que dans ce type de contrôle, puisque les commandes sont générées par un circuit combinatoire, il est nécessaire de les séparer par des délais convenables, d'une part, pour éviter les perturbations dues aux aléas et, d'autre part, pour assurer une synchronisation entre la partie contrôle et les différentes éléments de la partie opérative. Pour cela, des distributeurs de phases sont utilisés qui, à partir des tops d'horloge, fournissent des signaux d'impulsion à des niveaux différents du cycle.

3.3.1.2 PARTIES CONTRÔLE À MICROPROGRAMMATION

Le contrôle par microprogrammation est un concept qui date depuis 1951, mais il n'a été utilisé qu'à la fin des années 70 par Fairchild dans le 52752 fast et par Motorola dans le 68000 [Obr82].

L'algorithme d'interprétation est décrit sous forme d'un microprogramme placé dans une ROM. A chaque étape de l'algorithme d'interprétation est associée une ou plusieurs micro-instructions. Chaque micro-instruction fournit directement les commandes pour la partie opérative et l'adresse de la micro-instruction suivante.

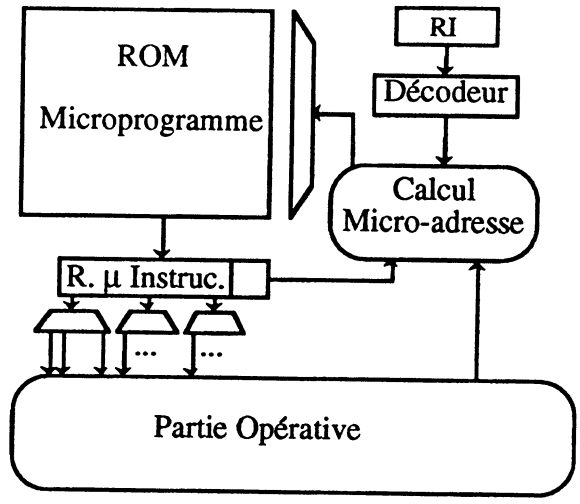


Figure 3.11 : Partie contrôle microprogrammée

Une partie contrôle utilisant ce concept est constituée généralement de la ROM contenant le microprogramme, d'un micro-compteur ordinal contenant l'adresse du mot sélectionné, d'un registre de micro instruction et d'un circuit permettant le calcul de l'adresse de la micro instruction suivante (figure 3.11). On distingue deux types de contrôle par microprogrammation :

- la microprogrammation horizontale,
- la microprogrammation verticale.

a) MICROPROGRAMMATION HORIZONTALE

Dans ce type de microprogrammation, chaque état de l'algorithme d'interprétation est décrit par une micro instruction. Cette dernière doit non seulement pouvoir contenir suffisamment d'information concernant un état mais aussi fournir des informations directement exploitables donc les moins codées possible.

La définition du format de la micro instruction est alors une tâche délicate. Généralement, on retrouve au moins deux champs : Le champ de séquencement et le champ des commandes. Chaque bit du champ des commandes correspond à une commande de la partie opérative. Vu la nécessité de prévoir suffisamment de bits pour couvrir l'ensemble des commandes dans chaque phase, le champ des commandes de la micro instruction risque d'être très long et, en conséquence, la taille de la ROM devient importante.

b) MICROPROGRAMMATION VERTICALE

Dans la microprogrammation verticale, on retrouve essentiellement des techniques ayant pour but de réduire la longueur de la micro instruction et plus précisément le champ des commandes, diminuant ainsi la taille de la ROM. Pour cela plusieurs méthodes peuvent être utilisées :

• La définition d'une partition de la micro instruction en un champ codant les commandes compatibles de la partie opérative. Autrement dit, les commandes qui ne sont jamais activées simultanément.

• La définition des champs de la micro-instruction en relation avec les différents blocs fonctionnels constituant la partie opérative. Un champ code spécifie alors tous les profils des commandes nécessaires pour un bloc donné.

• L'utilisation des micro-instructions à format variable. Dans cette méthode, de nouveaux formats sont définis dans lesquels les commandes actives au même moment sont regroupées tout en essayant de garder le même nombre de bits. Un champ spécial est donc ajouté indiquant le type de découpage et la signification de chaque champ dans la micro-instruction.

A titre de comparaison, et grâce à l'utilisation de ces techniques, les micro-instructions sont plus courtes dans la microprogrammation verticale par rapport à celles dans la microprogrammation horizontale. Par conséquent, la taille de la ROM peut être considérablement réduite.

Il faut noter également que la surface consommée par les décodeurs et les interconnexions n'est pas négligeable. La microprogrammation verticale est efficace dans les cas de microprogrammes longs où le gain en surface de la ROM contrebalance le coût des décodeurs.

Il est important de signaler pour cette structure de contrôle les problèmes d'implantation posés par la ROM. Il s'agit de faire le compromis entre :

• L'utilisation d'une ROM la plus compact possible afin d'économiser du silicium

• La surface consommée par les interconnexions entre les sorties de la ROM et les points d'action des commandes qui sont dispersés sur toute la partie opérative.

Pour y remédier, on peut modifier le facteur de forme de la ROM en divisant le plan de la matrice de mémorisation par le décodeur en le plaçant au milieu ou bien mettre plusieurs mots sur une ligne et utiliser des démultiplexeurs.

3.3.1.3 PARTIES CONTRÔLE UTILISANT LES PLA

Les parties contrôles utilisant des PLA reposent sur le principe de réaliser la partie contrôle en partant de la description des commandes sous forme d'équations booléennes. En analysant chaque état ou étape de l'algorithme d'interprétation, on obtient les équations des commandes qui doivent être activées pendant cet état tout en tenant compte éventuellement de l'ensemble des conditions d'activation associées.

Une commande C_i est exprimée par une équation booléenne sous forme de sommes de produits. Une commande est donc une fonction logique qui a pour variables les bits du code opération, les signaux de contrôle externes, les flags de la parties opérative et les bits codant l'état de l'automate de contrôle.

Rappelons brièvement qu'un PLA (Programming Logic Array) est constitué de deux matrices de transistors disposés de telle manière que la première matrice réalise la fonction "ET" et la deuxième matrice réalise la fonction "OU". La sortie est donc la somme des produits des entrées. Dans une organisation classique d'un PLA, on s'aperçoit que les sorties sont parallèles aux entrées. La série des entrées et des sorties constitue la longueur du PLA. La largeur est déterminée par les monômes. La taille d'un PLA est donc directement proportionnelle au nombre des entrées, au nombre des sorties et au nombre de monômes.

Avec les PLA, les mêmes problèmes sont rencontrés que ceux avec la ROM, à savoir la taille et l'implantation. En ce qui concerne la taille, il est nécessaire de dessiner les buffers d'entrées et de sorties les plus étroits possible afin de réduire le pas au minimum. Pour l'implantation, des méthodes d'optimisation topologique ont été développées [Per80] et [Chu82] permettant la réorganisation de la structure des PLAs.

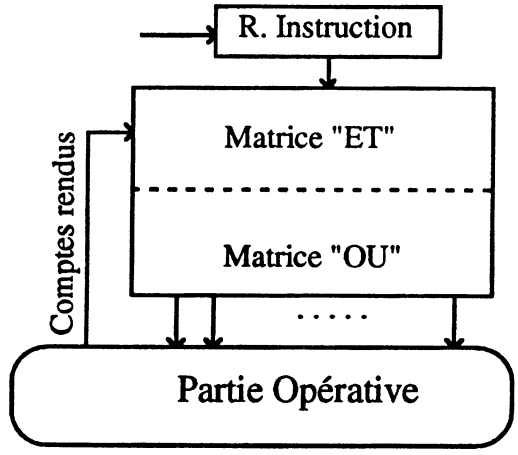


Figure 3.12: Partie contrôle à PLA unique

On distingue deux types de structure de contrôle utilisant des PLA : Les parties contrôle à PLA unique et les parties contrôle à PLA multiple.

Dans le premier type, l'ensemble des commandes de la partie opérative sont générées à l'aide d'un seul PLA (figure 3.12). Le choix d'une telle méthode peut se justifier seulement pour les machines relativement simples. Le risque d'aboutir à un PLA long en temps de réponse et gros en terme de surface ne peut être évité pour des processeurs ayant des jeux d'instruction riches et complexes.

Le deuxième type de contrôle à base des PLAs a pour principe d'utiliser plusieurs PLA (figure 3.13). Ceci s'inscrit bien sûr dans l'axe de rechercher tout les moyens et méthodes possibles afin d'apporter une minimisation et utilisation efficace de la surface des circuits et une amélioration sur les performances.

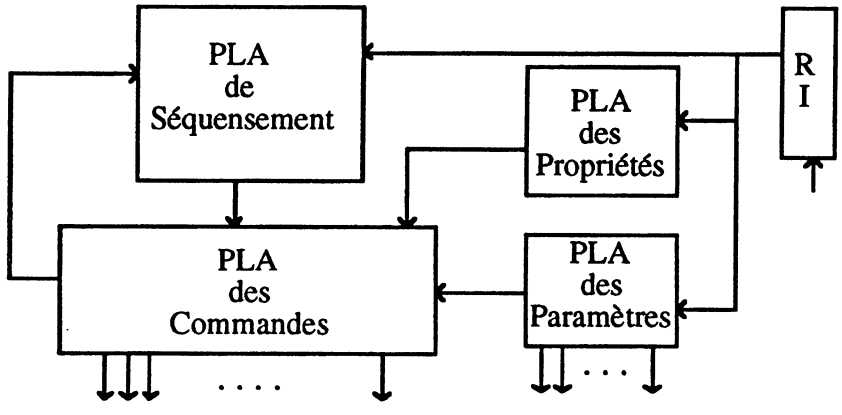


Figure 3.13 : Partie contrôle à PLA multiple

L'inconvénient des parties contrôle à PLA unique réside, d'une part, dans la taille et, d'autre part, dans la nécessité de présence, pour chaque état, de toutes les informations décrivant l'algorithme et de la machine. Ce qui affecte le temps de réponse. A chaque fois, on a besoin de décoder presque toutes ces informations pour la détermination de l'état suivant et la génération des commandes pour la partie opérative.

Afin d'éviter de procéder à un décodage à chaque cycle, il est plus intéressant d'extraire les informations décrivant l'instruction directement à partir du code opération, dès sa lecture, et seulement les informations utiles seront utilisées. Pour ce faire, l'algorithme d'interprétation est transcrit sous une forme permettant la visualisation des propriétés des instructions qui sont prises en compte dans chaque état de l'algorithme. Cela conduit à une transformation de l'algorithme sous une forme équivalente de l'algorithme de Mealy. A ce stade, le séquencement n'est plus fonction de certaines propriétés. Par contre, ces dernières influent seulement sur la génération des commandes.

En partant de la nouvelle description de l'algorithme d'interprétation, la définition et le calcul du PLA de séquencement devient plus simple. Le PLA doit reconnaître maintenant que des groupes ou des familles d'instructions seulement et un nombre réduit d'états. Le codage des états nécessite donc moins de bits. Ce qui diminue la taille globale du PLA de séquencement.

L'information codée et générée par le PLA de séquencement est exploitée avec le code et les caractéristiques de l'instruction afin de générer les commandes pour le chemin des données. Ces caractéristiques qui restent fixes tout au long du déroulement de l'exécution, sont extraites du code opération en une seule fois lors du décodage. Elles peuvent indiquer par exemple : le type d'opération à effectuer; lecture ou écriture, le type d'opérande associé; mémoire ou registre interne, etc. ...

L'avantage de l'utilisation des PLA multiples réside dans le remplacement de la logique aléatoire utilisée dans les réalisations câblées par une structure plus régulière du point de vue implantation matérielle offrant plus de flexibilité pour le concepteur.

La décomposition de l'algorithme découle d'une façon naturelle de la classification hiérarchique des informations en distinguant les informations globales (propriétés), locales (paramètres) et temporelles (séquencement), permettant ainsi de passer de l'automate de Moore à l'automate de Mealy.

L'approche de l'utilisation des PLA multiples est également avantageuse du point de vue mise au point et développement du circuit.

A titre de comparaison avec le concept de la microprogrammation, du point de vue souplesse, les deux principes utilisent actuellement des générateurs automatiques. Par contre du point de vue surface, l'existence du champ adresse dans la micro-instruction est la cause d'une perte de surface.

3.3.1.4 PARTIES CONTRÔLE UTILISANT UN GÉNÉRATEUR DE TEMPS

Le principe du contrôle à base générateurs de temps découle du fait que chaque commande est caractérisée par l'instant durant l'exécution de son activation. Le déroulement d'une instruction est donc décrit par deux ensembles : les commandes à activer et les instants de leur activation.

Ce type de contrôle inclut un dispositif d'élaboration des commandes à activer, un moyen de générer les instants et un système de validation des commandes (voir figure 3.14).

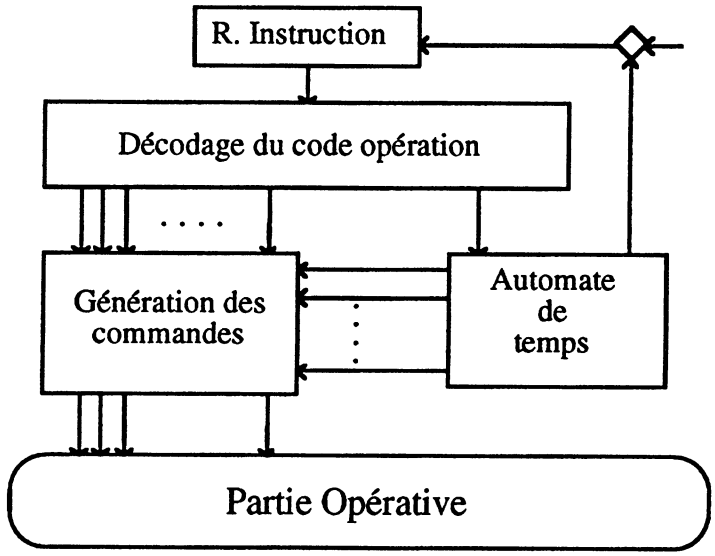


Figure 3.14 : Contrôle utilisant un générateur de temps

Les commandes à activer peuvent être obtenues directement à partir du code opération à l'aide d'un PLA. Les instants peuvent être générés par un automate qui assure la progression dans le déroulement de l'exécution ainsi que l'identification de chaque instant par les signaux de séquencement.

Les signaux de décodage (propriétés) et les signaux de séquencement sont combinés dans un PLA ou un circuit de logique combinatoire pour donner

finalement les commandes séquencées prêtes à être exploitées par la partie opérative.

A la première pensée, l'automate peut être simple et efficace par rapport à l'algorithme d'interprétation. Les jeux d'instructions des microprocesseurs incluent, pour marquer leur originalité, quelques instructions qui nécessitent un traitement spécial en ce qui concerne le séquencement.

Le graphe de l'automate peut rester homogène grâce à l'utilisation des automates auxiliaires ou des compteurs adaptés.

Bien qu'elle soit réalisable, cette forme de partie contrôle n'est pas utilisée dans les microprocesseurs connus.

3.3.2 PARTIES CONTRÔLE À PLUSIEURS NIVEAUX D'INTERPRÉTATION

Le principe de ce type de contrôle consiste à empiler des parties contrôles mono-niveau de telle façon à ce que les commandes générées d'un niveau supérieur servent d'instructions au niveau juste inférieur. Un niveau d'interprétation *i* doit signaler au niveau *i-1*, la fin de l'interprétation correspondante pour que ce dernier puisse envoyer l'instruction suivante.

La difficulté de cette structure réside dans la détermination de la part d'interprétation de chaque niveau. Intuitivement, la démarche à suivre consiste à décomposer chaque instruction en une suite d'étapes où chacune est chargée d'accomplir une tâche précise. Chaque étape peut être décomposée en plusieurs pas et chaque pas peut être encore décomposé en plusieurs tâches élémentaires.

Le décodage doit être effectué de telle manière qu'il y ait à chaque niveau, seulement les informations nécessaires concernant le niveau. Il faut donc pouvoir disposer des informations décrivant le type de l'étape, le niveau de pas et son type de pas, etc. Chaque information subit une interprétation plus poussée au niveau inférieur. La figure 3.15 illustre ce type de contrôle.

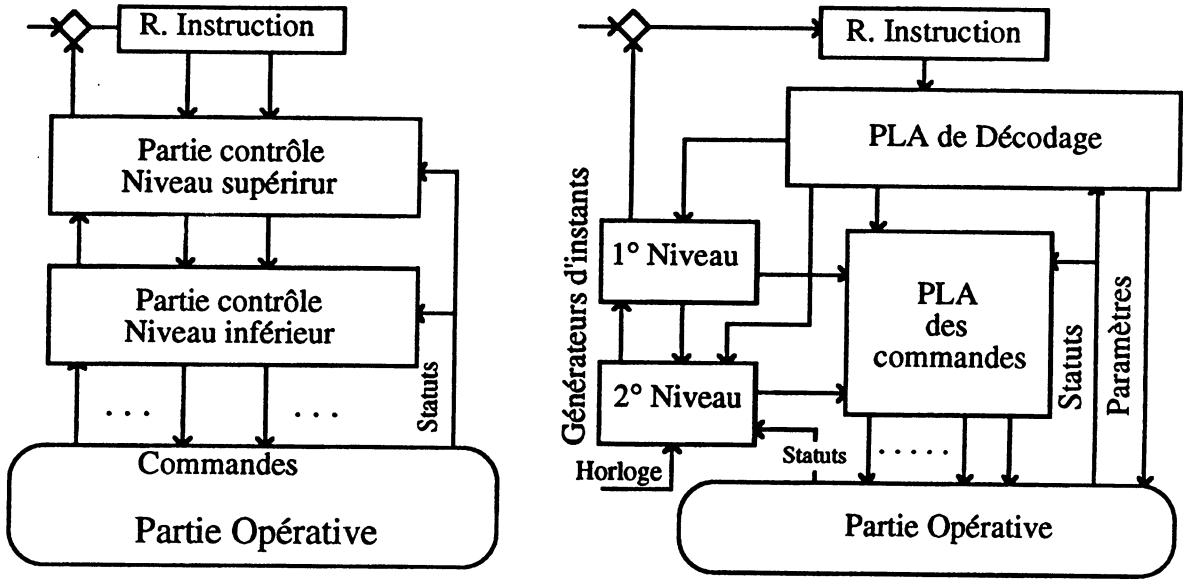


Figure 3.15 : Partie Contrôle à plusieurs niveaux d'interprétation

Au moment de la réalisation matérielle, chaque niveau peut être conçu en utilisant l'une des méthodes déjà présentées.

Notons que le contrôle à plusieurs niveaux d'interprétation a trouvé application dans les microprocesseurs qui ont marqué les années 1970 comme le fameux I 8080 (1973), I 8085 (1977), Z 80 (1976) et le Z 8000 (1978) [Obr82].

3.3.3 ARCHITECTURE DE LA PARTIE CONTRÔLE PROPOSÉE

L'architecture de la partie contrôle proposée pour notre processeur est une combinaison de plusieurs types de contrôle qu'on vient de présenter. Elle est du type de deux niveaux d'interprétation utilisant de multiples PLAs avec un générateur de temps (figure 3.16). Le choix de cette structure découle de la tâche de respecter la contrainte de surface puisque la partie contrôle prend, en général, plus de 50% de la surface des processeurs classiques [Anc85]. En plus de la surface, ce choix repose également sur la nature du jeu d'instructions relativement riche, les moyens de réalisation mis à disposition et les méthodes de mise au point envisagées.

Le PLA Groupe constitue le premier niveau d'interprétation. Douze groupes sont décodés (voir tableau 3.10) en fonction de leurs propriétés, leurs modes d'adressage et les éléments de la partie opérative qui sont impliqués. Le décodage prend en compte également du chemin ou de la trajectoire suivie dans le graphe de contrôle de l'automate lors de l'exécution de l'instruction.

L'automate joue le rôle du générateur de temps au même moment qu'il contrôle le déroulement de l'exécution des instructions.

Les PLA constituent le deuxième niveau d'interprétation et en même temps génèrent les commandes pour la partie opérative.

Au départ, un seul PLA a été prévu. Par la suite, il est devenu nécessaire de le découper en plusieurs petits PLA, d'une part, pour diminuer le temps de réponse et, d'autres part, pour faciliter le calcul des monômes. Ce découpage a pour but d'introduire une modularité dans ce bout de circuit pour permettre une aisance dans la mise au point des versions futures du processeur. Donc, pour chacune des quatre unités de la partie opérative, un PLA lui est associée. En plus de cette division par module de traitement, un deuxième découpage s'est avéré nécessaire au sein d'un même PLA modulaire lors de la phase de réalisation afin d'homogénéiser les temps de réponse des différents PLA et de faciliter la synchronisation entre la partie contrôle et la partie opérative.

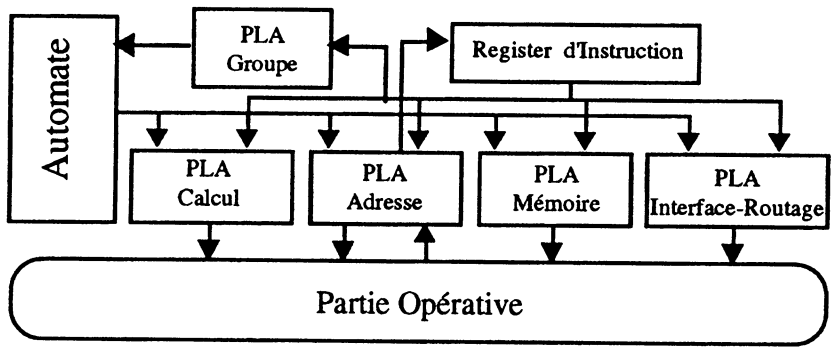


Figure 3.16: Structure de la partie contrôle proposée

Groupes		Instructions	Modes d'adressage	# de Monômes
Fetch ((instr)). --> Ecrit. Mém.	iw	STA $XX	Abs Court	4
		PUT $XX	Abs court	
		STA (I)	Indirect	
		STA (I)+	Indirect P. Incr.	
		PUT (I)	Indirect	
		PUT (I)+	Indirect P. Incr.	
Fetch (Opér). -->Exéc.	oe	TRY $XX	Abs Court	6
		GET $XX	Abs Court	
		Opérat Unaires	Abs, (I), (I)+	
		Opérat Binaires	Abs, (I), (I)+	
Exéc. -->Fetch (Opér).	eo	ADCW	Abs	3
		SBCW	Abs	
Exéc. -->Ecrit. Mém	ew	Opérat Unaires	Abs, (I), (I)+	3
Fetch Adr. --> Fetch (Opér).	ao	Toutes	Abs	1
Fetch ((instr)). --> Fetch (Opér).	io	TRY $XX	Abs Court	1
		GET $XX	Abs Court	
		LDA $XX	Abs Court	
		Toutes	Immédiat., (I), (I)+	
Fetch ((instr)). -->Exéc.	ie	Branchement.	Inhérent	1
		Opérat sur Acc	Inhérent	
		Opérat sur Retenue	Inhérent	
		Acc --> PC	Inhérent	
		Acc --> (I)	Inhérent	
Fetch (opér). --> Fetch Opér.	oo	Multiplication	Abs	4
		LDAW	Abs, imméd, (I), (I)+	
		SEND	Abs, (I), (I)+	
MUL	mul	MUL	Abs	1
GET	get	GET	Abs, (I), (I)+	1
Ecrit. Mém.--> Ecrit. Mém.	ww	STAW	Abs, (I), (I)+	3
SEND	send	SEND	Abs, (I), (I)+	2

Tableau 3.10: Groupes des Instructions

3.4 IMPLÉMENTATION

3.4.1 PARTIE OPÉRATIVE

La partie opérative comprend quatre blocs distincts. Un bloc de calcul constitué de l'unité arithmétique et logique ainsi que de deux accumulateurs et les éléments nécessaires au calcul des bits d'état. Un bloc d'adressage composé des éléments d'adressage tel que le compteur ordinal (pc), le registre d'index (I) et un registre auxiliaire (RA). Un troisième bloc contenant la mémoire et une bascule pour gérer le 9ième bit. Et enfin, un quatrième bloc servant à implémenter les deux fonctions d'émission et de réception des message qu'on appellera la partie interface routage (figure 3.17).

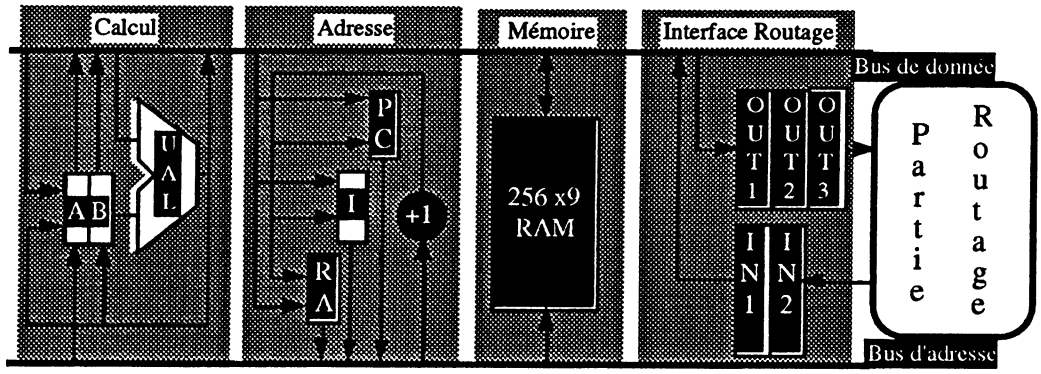


Figure 3.17 : Structure de la partie opérative

Dans ce qui suit, nous allons détailler l'implémentation de chacun de ces quatre blocs.

3.4.1.1 PARTIE CALCUL

Comme son nom l'indique, la partie calcul constitue le cœur de la partie opérative du processeur où les opérations arithmétiques et logiques sont exécutées.

La structure de la partie calcul représente généralement un moyen de classification des parties opératives et même des processeurs.

La partie calcul présentée dans la figure 3.18 a une structure double bus à base d'éléments standards tel que les accumulateurs et l'unité arithmétique, [Anc85]. Le choix a été porté sur cette structure pour sa simplicité et sa compatibilité avec le jeu d'instructions défini. Le chemin de donnée doit non

seulement permettre l'exécution du jeu d'instructions d'une manière efficace mais aussi répondre à des impératifs de surface très sévères. Plus on se rapproche des structures classiques plus on satisfait les contraintes de surface. La structure de la partie calcul proposée diffère par rapport à celle de base par le rajout de quelques éléments simples afin de mieux l'adapter au jeu d'instructions et de quelques modifications en terme d'interconnexion entre registres.

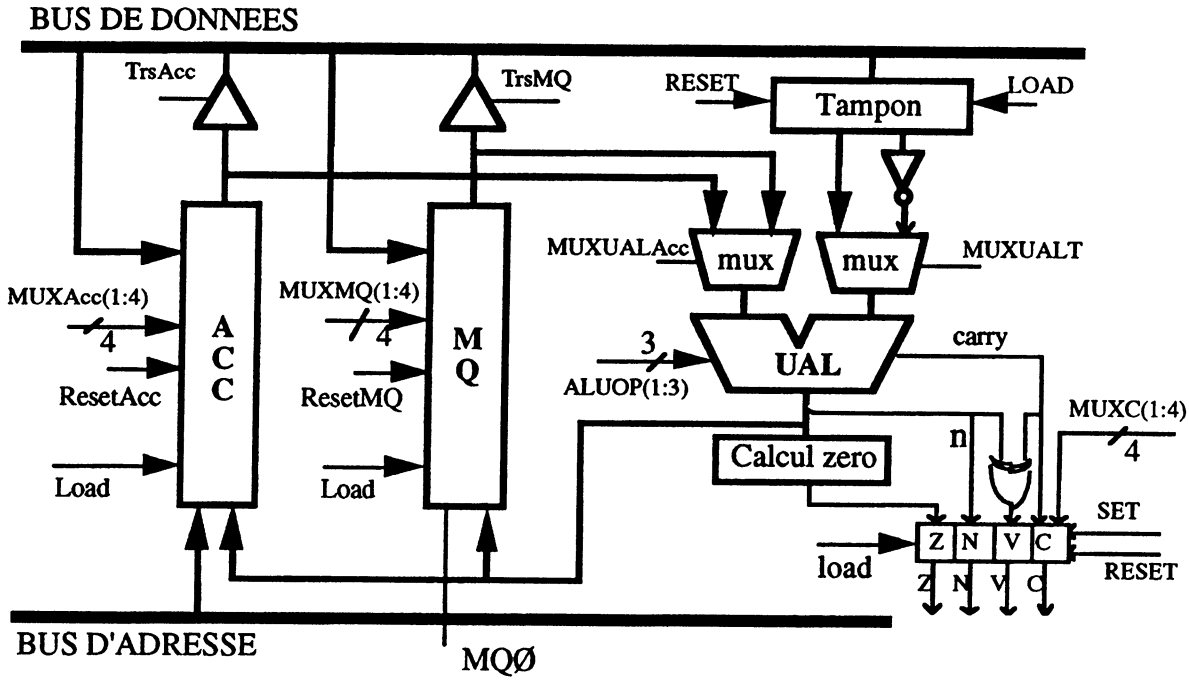


Figure 3.18 : Schéma de la partie calcul

3.4.1.1.1 LES ACCUMULATEURS

Les accumulateurs (Accumulateur A et le registre MQ) sont des registres à décalage bidirectionnels connectés à travers un multiplexeur à la même entrée de l'UAL. Ils sont accessibles à partir du bus de données et de l'UAL. L'accumulateur A a une entrée supplémentaire à partir du bus d'adresse afin d'exécuter l'instruction TPCA (transfert PC dans Acc) d'une manière très rapide puisque le registre PC (compteur ordinal) ne peut pas écrire sur le bus de données.

Pour certaines instructions, le résultat de l'opération est sauvegardé en fin du cycle dans l'accumulateur A comme indiqué dans la table du jeu d'instructions et la description d'exécution des instructions en annexe 1 et 2.

Les instructions de décalage arithmétique et logique sont exécutées au niveau des accumulateurs. On retrouve donc à l'entrée de chaque bit des accumulateurs, des multiplexeurs. Pour l'accumulateur A, il nous faut un multiplexeur de sept entrées tandis que pour le registre MQ, on n'a besoin que d'un multiplexeur à six entrées. Ces entrées sont :

- bus de données,
- sortie UAL,
- bit - 1 (pour les décalages à gauche),
- bit + 1 (pour les décalages à droite),
- carry (pour le bit 0 et bit 7 seulement),
- 0 (pour le bit 0 et bit 7 seulement),
- bus d'adresse (pour accumulateur A seulement).

3.4.1.1.2 L'UNITÉ ARITHMÉTIQUE ET LOGIQUE

L'unité arithmétique et logique utilise un additionneur du type "Ripple Carry" pour sa simplicité, sa surface par rapport aux autres types et son temps de propagation suffisant pour être utilisé dans un processeur massivement parallèle (O(n)). L'UAL possède 5 opérations : 4 opérations logiques et 1 opération arithmétique (addition). L'addition est basée sur des portes OU exclusifs. Ceci a été fait dans le but d'exploiter la porte OU exclusif prévue pour effectuer l'opération logique XOR qui figure dans le jeu d'instructions du processeur. Le schéma général d'un bit d'UAL est montré dans la figure 3.19.

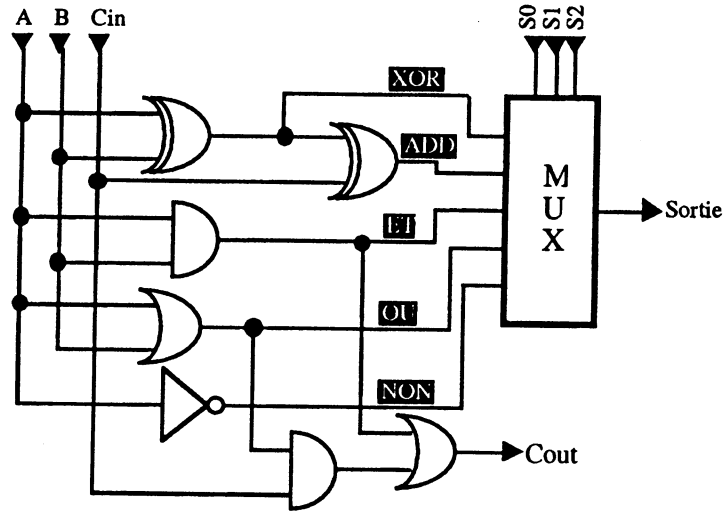


Figure 3.19 : Schéma général d'un bit d'UAL

Le multiplexeur de sélection des opérations possède des commandes encodées au lieu d'utiliser une commande pour chaque opération afin de réduire le nombre de commandes dans la partie contrôle gérant l'UAL.

La soustraction est effectuée en additionnant le premier opérande avec le complément du deuxième, après avoir initialisé la retenue entrante de l'UAL à 1 [Mul89]. Les opérations logiques sur des mots mémoire sont effectuées au niveau du registre MQ afin de préserver le contenu de l'accumulateur A.

3.4.1.1.3 LA MULTIPLICATION

L'algorithme de multiplication employé découle de la méthode classique basée sur l'addition et décalages. Cette technique demande un matériel élémentaire (un additionneur et des registres à décalage qui existent déjà) dont le temps de calcul est de l'ordre du nombre de bits des résultats qui est de 16 bits dans notre cas [Bae80].

L'instruction MUL possède un mode d'adressage unique (absolu). Cette instruction nécessite au préalable une préparation par le programmeur des multiplicandes et le multiplieur en mémoire en deux positions consécutives. Par contre, à la fin de l'exécution, le résultat est maintenu dans les deux accumulateurs et c'est au programmeur de l'exploiter.

3.4.1.1.4 LES BITS D'ÉTATS

Avec la partie calcul, est associé un mécanisme permettant de calculer à tout moment les bits d'états du processeur ainsi que les conditions de branchement. La figure 3.20 montre le schéma logique du circuit qui sert à gérer ces bits et à calculer les conditions de branchement. Le circuit contient des éléments de mémorisation et de logique combinatoire.

A la fin de chaque cycle processeur (sauf pour le "fetch" instruction), les bits d'état sont calculés et mémorisés.

- Le plus simple à calculer est le bit indiquant le signe (N) de la donnée en cours de traitement : c'est simplement le bit de poids fort de l'octet.
- Le second bit, dans l'ordre de simplicité, est la retenue (C). La retenue est prise de l'UAL après une opération arithmétique ou bien de l'un des accumulateurs après une opération de décalage. Un multiplexeur à 5 entrées est utilisé à l'entrée de la bascule mémorisant la retenue.

• Le troisième bit est celui du débordement. Le calcul de ce bit lors des opérations arithmétiques se fait en fonction du bit de signe (N), du résultat et de la retenue sortante de l'UAL (C) selon l'équation logique suivante [Anc85]:

$$V = N \oplus C$$

• Le quatrième et dernier bit d'état est le bit Z indiquant si oui ou non la donnée sous traitement est nulle. Le bit Z est calculé après un chargement de l'un des accumulateurs, après une instruction de décalage ou après une opération arithmétique. Ceci nécessite un multiplexage entre ces trois sources et un moyen qui permet de détecter que les huit bits de la donnée sous traitement sont tous à l'état logique zéro. Pour cela, on utilise deux portes NOR à quatre entrées combinées avec une porte ET. Il a été préférable d'éviter l'utilisation d'une porte NOR à huit entrées pour, en premier lieu, éviter le risque d'avoir un temps de propagation de la porte important et, en second lieu, à cause de son absence de la bibliothèque des cellules utilisées.

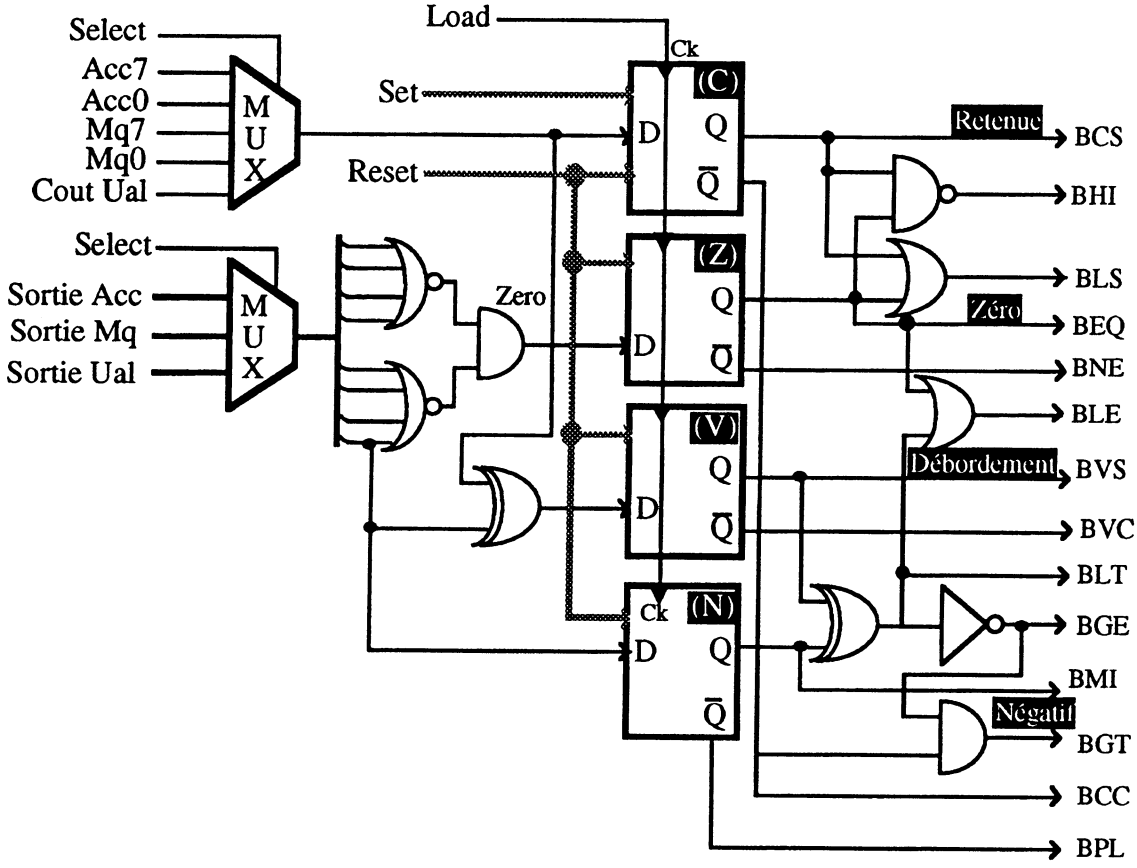


Figure 3.20 : Circuit de calcul des bits d'états

3.4.1.2 PARTIE ADRESSAGE

La partie adressage du chemin de données est constituée essentiellement de trois registres 8 bits (PC, I, RA), un incrémenteur et un compteur. Le registre PC (compteur ordinal) contient à tout moment l'adresse de l'instruction suivante à exécuter. Le registre I est le registre d'index qui sert de registre pointeur d'adresse pour les instructions de mode d'adressage indirect avec ou sans post incrémentation. Ces deux registres sont identiques du point de vue implémentation. Ils peuvent être chargés à partir du bus de données ou de l'incrémenteur.

Au niveau des entrées des deux registres, il devient nécessaire d'utiliser des multiplexeurs 2-->1. Ces deux registres peuvent être lus à partir du bus d'adresse à travers des buffers trois états. La figure 3.21 montre la structure de la partie adressage.

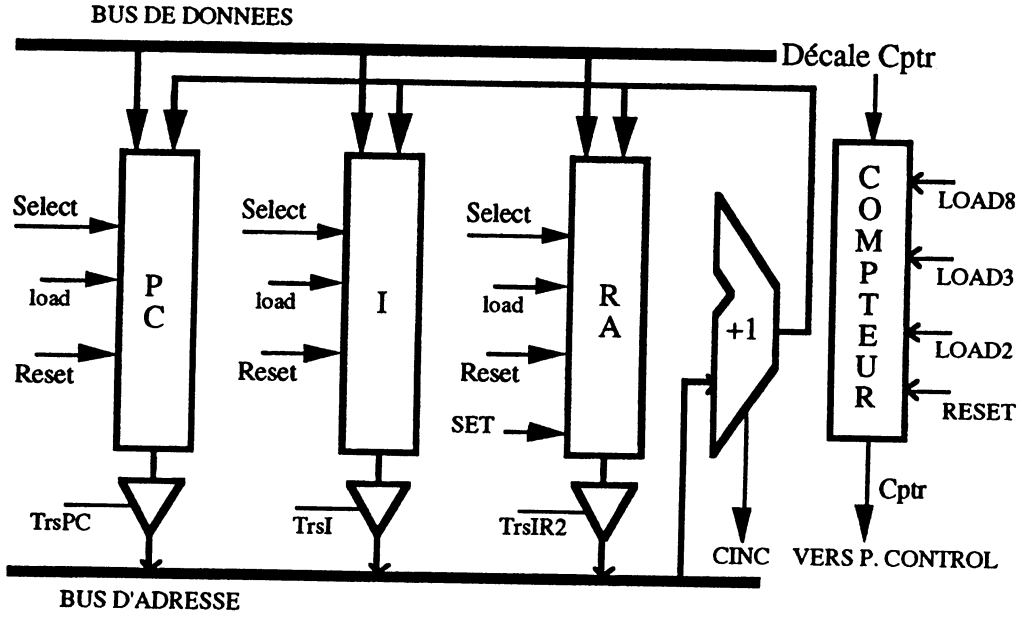


Figure 3.21 : Structure de la partie d'adresse

Le registre auxiliaire RA est utilisé comme un registre temporaire principalement pendant l'exécution des instructions dont le mode d'adressage est :

- absolu court
- absolu
- indirect (sans post-incrémentation) pour les instruction : SEND, LDAW, STAW, ADCW, SBCW

A la fin d'un cycle fetch instruction, le code de l'instruction est chargé dans le registre d'instruction (IR inclue dans la partie contrôle) et le registre auxiliaire RA. Ceci est fait dans le but d'accélérer l'exécution des instructions dont l'adressage est l'absolu court.

Le code d'instruction étant de huit bits, les quatre bits du poids fort désignent le code de l'instruction et les autres quatre bits pointent sur l'une des 16 positions mémoires soit dans la page $00-0F ou bien $F0-FF. Juste après le décodage de l'instruction, les quatre bits de poids fort sont remplacés par des 1 ou des 0 selon l'instruction. Le registre auxiliaire (RA), est utilisé comme pointeur d'adresse pour le reste de l'exécution de l'instruction.

Le remplacement des bits doit être fait rapidement afin de laisser suffisamment de temps pour la suite de l'exécution. Trois manières d'implémentation peuvent être considérées :

- Utilisation des bascules avec Preset pour les 4 bits de poids fort seulement (Figure 3.22-a).
- Utilisation des multiplexeurs à l'entrée des 4 bits de poids fort seulement (Figure 3.22-b).
- Utilisation des bascules à Reset avec des multiplexeurs et les sorties complémentées des 4 bits de poids fort seulement (Figure 3.22-c).

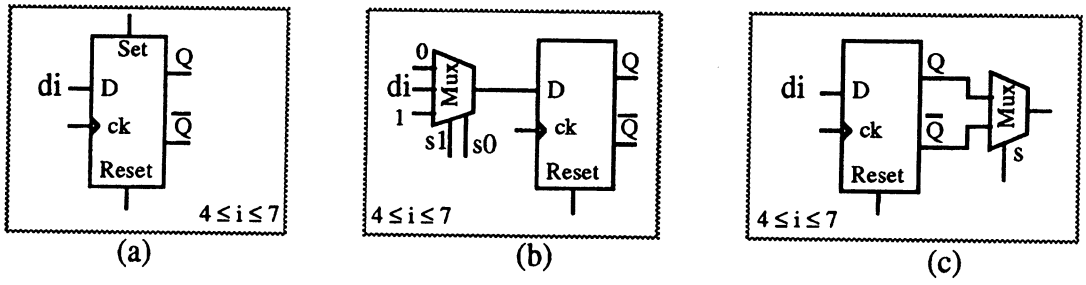


Figure 3.22 : Solutions pour le registre AR

Le choix entre ces trois alternatives est dicté par les contraintes du temps et de la surface bien que ce mécanisme n'a pas un facteur important de répététivité dans l'ensemble du processeur. Puisque dans certains cas, le registre RA est utilisé comme pointeur d'adresse. Lui aussi peut être chargé à partir du bus de données et de l'incémenteur. Donc, on retrouve aussi aux entrées des multiplexeurs 2-->1. Sachant que l'on veut surtout accélérer l'exécution des instructions du mode adressage rapide, il faut donc prendre la solution la plus

rapide, décrite dans la figure 4.31-a. La solution (b) nous aurait conduit à l'utilisation des multiplexeurs 4-->1 qui est plus long par rapport à la solution (a). Quant à la solution (c), on se retrouverait avec une bascule ayant un multiplexeur 2-->1 à l'entrée et à la sortie également plus lente que la solution (a).

3.4.1.2.1 L'INCRÉMENTEUR

L'implémentation de l'incrémenteur est très classique. Elle utilise des portes XOR à deux entrées et des portes ET pour propager la retenue (voir figure 3.23). Il effectue donc la mise à jour des trois registres PC, I et RA. Pour un cycle quelconque durant l'exécution d'une instruction, seulement l'un des trois registres est impliqué. Il est donc inutile d'avoir un incrémenteur pour chaque registre. Ces registres ont accès au bus d'adresse à travers des buffers trois états. Puisque l'incrémenteur est un circuit purement combinatoire et pour éviter l'utilisation d'un multiplexeur, on peut continuellement incrémenter le bus d'adresse et commander l'écriture et la sélection du registre impliqué au moment voulu.

Le compteur possède une deuxième utilité qui s'avère très importante lors de la phase d'initialisation du processeur. Après la mise sous tension ou un signal Reset, il est nécessaire d'initialiser les 9 ièmes bits de la mémoire avant de procéder à l'attente des paquets de messages constituant le programme à exécuter. La mémoire est donc parcourue depuis l'adresse 00 jusqu'à l'adresse $FF par une écriture d'un 0. La retenue sortante *Inc* de l'incrémenteur constitue le moyen de la fin d'initialisation de la mémoire.

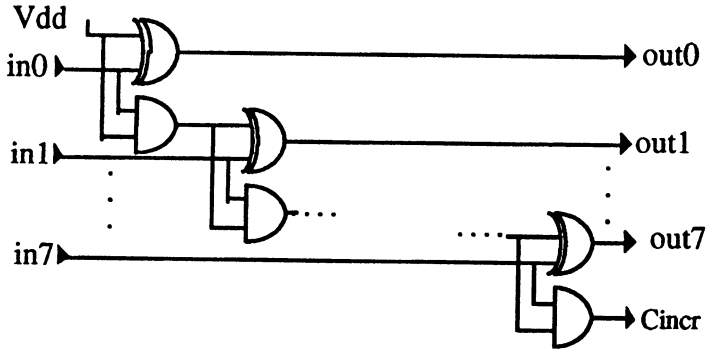


Figure 3.23: Incrémenteur

3.4.1.2.2 LE COMPTEUR

Le compteur utilisé dans le bloc de calcul est assez spécial. C'est un compteur à base de registres à décalage dont l'état logique de la retenue nous indique la fin du comptage (figure 3.24). Il est sollicité lors de l'exécution des instructions un peu spéciales comme la multiplication (MUL), l'envoi des messages (SEND) et les instructions à 16 bits (LDAW, STAW, ADCW, SBCW). Il possède trois modulo :

a) Modulo 2 : impliqué lors d'exécution des instructions à 16 bits
b) Modulo 3 : impliqué lors d'exécution de l'instruction SEND
c) Modulo 8 : impliqué lors d'exécution de l'instruction MUL

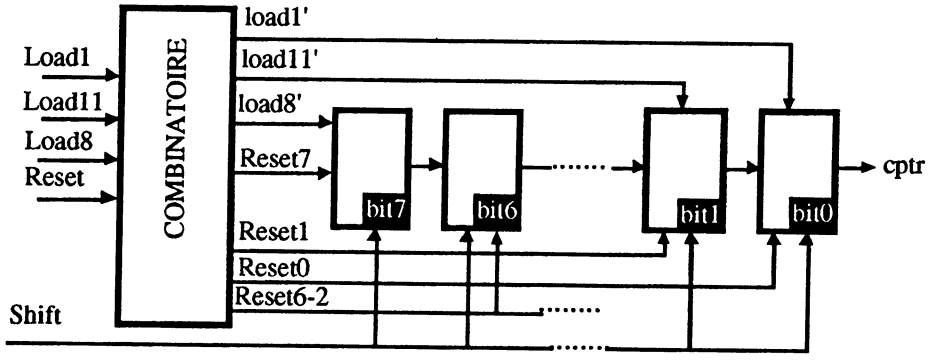


Figure 3.24 : Le compteur

La partie combinatoire du compteur sert à générer les commandes locales pour le compteur en fonction des commandes globales générées par la partie contrôle.

En ce qui concerne les commandes d'initialisation ou de chargement du compteur (load8, load11 et load1), leur passage par le circuit combinatoire sert simplement à les inverser car les commandes de preset des bascules utilisées sont actives à l'état logique bas. Par contre, ce qui est généré en plus par rapport aux commandes globales, ce sont les commandes de remise à zéro des différentes bascules. Ceci peut paraître additionnel, mais cela a été fait par mesure de sécurité car la sortie du compteur "cptr" est utilisée par l'automate de contrôle au niveau des conditions de transition d'un état à un autre.

Afin de ne pas laisser des bits à l'état 1 traîner après l'exécution d'une instruction quelconque qui a sollicité le compteur, il est préférable lors de chaque initialisation de s'assurer du contenu de chaque bascule afin d'éviter toute erreur de parcours dans le graphe de l'automate pendant l'exécution. Le

schéma logique de cette petite partie combinatoire est montré dans la figure 3.24.

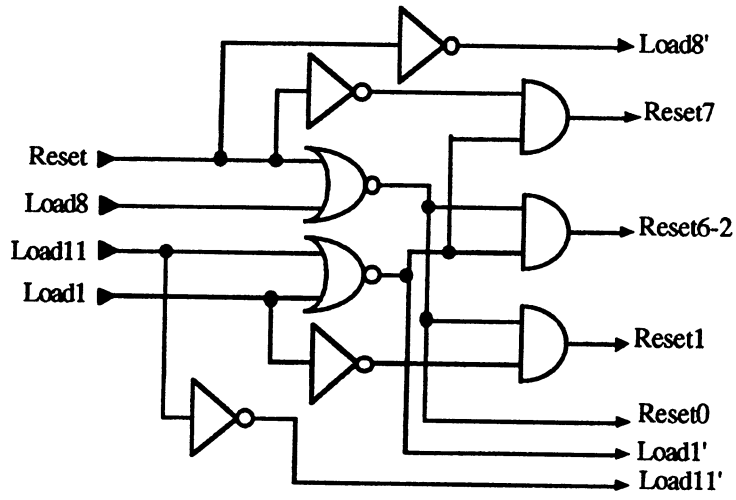


Figure 3.24 : Partie combinatoire du compteur

3.4.1.3 LA MÉMOIRE

La partie mémoire du chemin de données est constituée d'un bloc mémoire de 256 mots de 9 bits et une bascule D (figure 3.25). La taille de la mémoire (2^8 = 256 octets) a été choisie de telle façon à aboutir, d'une part, à un processeur 8 bits caractérisant notre approche intermédiaire entre la Connection Machine et le Transputer et, d'autre part, à un circuit pas trop gros du point de vue surface afin de pouvoir intégrer un maximum de processeurs dans un même boîtier.

Le 9ième bit n'est pas utilisé pour la parité comme c'est généralement le cas dans les mémoires de cette forme, mais principalement pour assurer une synchronisation de communication entre les processus du réseau distants ou locaux.

Lors de la phase d'initialisation, tous les 9ièmes bits sont initialisés à zéro. Après le rangement d'un message incident (dans une position mémoire définie par le tag), le 9ième bit correspondant est initialisé à 1 indiquant la présence d'un message à cette position. Après la lecture du message, le 9ième bit est remis à zéro.

La bascule D est utilisée afin de prendre en charge ce 9ième bit soit en lecture pour tester la présence d'un message soit en écriture en le mettant à 1 ou

0 pour marquer la présence ou le retrait d'un message. Une autre raison pour l'utilisation de cette bascule est de préserver l'homogénéité du bus de données à 8 bits.

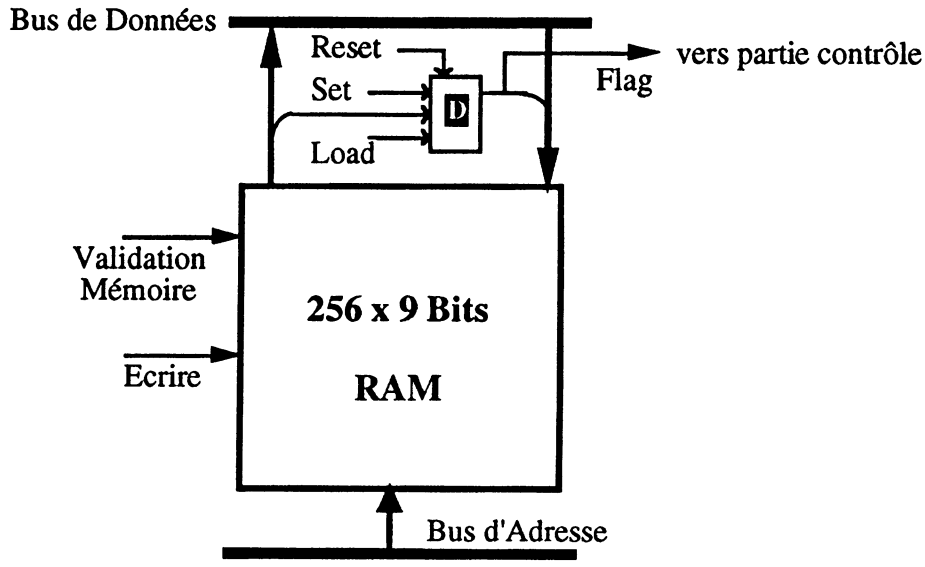


Figure 3.25 : La partie mémoire

3.4.1.4 INTERFACE ROUTAGE

L'interface routage fait l'objet d'une contrainte sévère sur la conception du processeur afin de garantir une absence d'interblocage dans l'acheminement des messages. Ces derniers doivent être retirés du réseau dans un délai ne dépendant pas d'une condition du réseau : les messages doivent être extraits du réseau dans l'ordre où ils arrivent. Le processeur ne doit pas avoir à attendre de pouvoir émettre un message avant de retirer le message incident. Il doit pouvoir donc exécuter deux processus concurrents : un processus de réception et un processus d'envoi de messages.

Gérer ceux-ci explicitement comme deux processus (au sens des systèmes d'exploitation) nécessite un noyau de commutation de processus coûteux en silicium et en temps.

En plus de la stratégie d'acheminement selon X d'abord ensuite sur Y, du mode de communication wormhole et du système tout parallèle, câbler un pseudo-processus dont le rôle est de ranger les messages incidents au processeur en mémoire, à une adresse définie par le champ TAG présent dans chaque message, résout définitivement le problème d'interblocage [Fau91], [Kar91a].

Matériellement, l'interface routage est basée sur de petites mémoires FIFO dans les deux sens : traitement-routage et routage traitement.

L'arrivée des messages est détectée à l'aide d'un signal *plein* qui a pour fonction de provoquer une interruption du processeur. Le signal d'interruption n'est pris en considération ou échantillonné que dans les trois cas suivants :

• Au début d'un cycle de la phase d'initialisation du processeur. Au cours de laquelle, le processeur ne fait qu'attendre son programme à exécuter sous forme de paquets de messages tout en vérifiant l'état des signaux de transparence.

• Au début d'un cycle fetch d'instruction.

• Au début d'un cycle d'exécution d'une instruction (voir automate de contrôle en figure 4.39). Il a été permis au contrôleur de considérer l'arrivée d'un message au début de ce cycle d'exécution pour éviter que l'instruction GET soit bloquante. Il se peut qu'on se retrouve dans une situation où le processeur exécute une boucle d'attente d'un message à travers l'instruction GET.

Pour la majorité des instructions, entre le fetch et l'exécution, il y a les accès mémoire en écriture ou en lecture. Un message, qui arrive pendant un accès mémoire en lecture ou en écriture, n'est pris en compte que pendant le cycle suivant.

La sauvegarde du contexte du processeur avant le rangement du message en mémoire n'est pas nécessaire puisque aucun des registres ou accumulateurs n'est utilisé. Le rangement est accompli en chargeant le bus d'adresse par le "tag" et le bus de données par la "valeur" avant d'activer les commandes pour un cycle d'écriture de la mémoire.

3.4.1.4.1 INTERFACE ROUTAGE-TRAITEMENT

L'interface routage-traitement constitue le pseudo-processus câblé afin de garantir l'absence d'interblocage des messages. L'interface routage-traitement est construite autour d'une mémoire FIFO de 2 x 8 bits (figure 3.26). Une fois la destination du message est atteinte, le champ d'adresse devient inutile. Seulement les deux octets "étiquette" et "donnée" restent significatifs et doivent être sauvegardés.

Le protocole d'échange de message entre la partie routage et traitement reste identique à celui entre deux cellules adjacentes car la partie traitement est vue par l'aiguilleur exactement comme une cellule voisine.

Les trois bascules connectées en cascade implémentent le principe de fonctionnement d'une mémoire FIFO. La troisième bascule signale au processeur l'incidence d'un message (signal plein) et en même temps indique au routeur l'état vide ou plein du buffer (mémoire FIFO) d'interface. Le signal remplir fait avancer les bits d'états dans les bascules et les paquets de message dans les registres constituant la petite mémoire FIFO.

Le signal *lire message* est généré par la partie contrôle du processeur tout au début d'un cycle écriture de la mémoire locale en réponse à l'interruption provoquée par le signal *plein* permettant ainsi la validation des buffers trois états..

La quatrième bascule a été ajoutée à des fins de synchronisation.

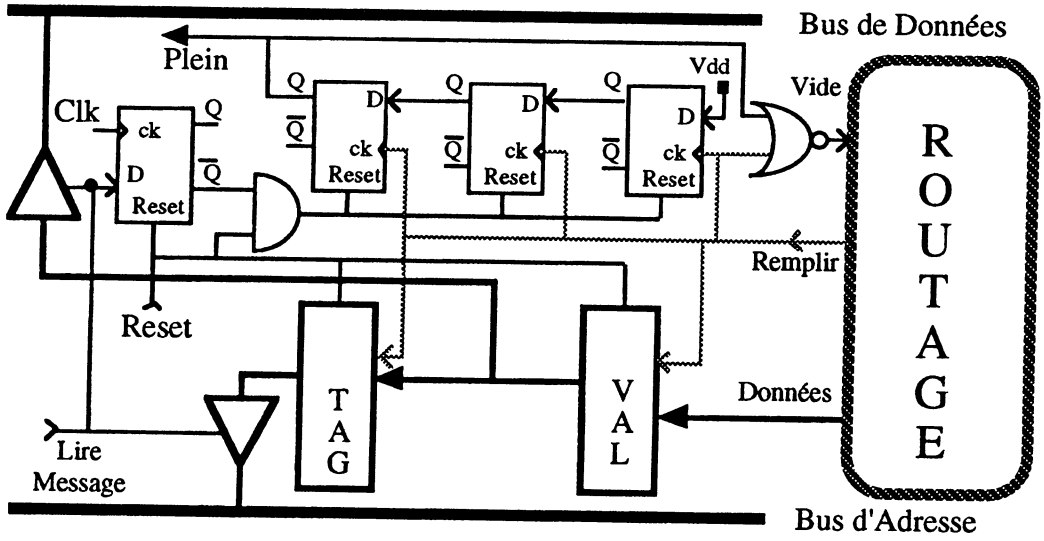


Figure 3.26: Interface routage traitement

3.4.1.4.2 INTERFACE TRAITEMENT-ROUTAGE

L'interface traitement-routage consiste essentiellement à exécuter l'instruction de communication SEND. Il s'agit de transférer les trois octets, constituant le message, stockés au préalable en mémoire vers la partie routage. De même que l'interface routage-traitement, elle est bâtis autour d'une mémoire FIFO de 3 x 8 bits jouant le rôle d'un buffer tampon entre la partie traitement et l'aiguilleur.

L'avantage d'utiliser un buffer est de ne pas monopoliser le processeur. Pendant l'envoi des messages paquet par paquet, il s'agit d'attendre et envoyer les signaux nécessaires au protocole d'échange. Rappelons que la partie traitement est vue par l'aiguilleur exactement comme un cinquième tampon d'entrée-sortie.

La détermination de la taille du buffer découle de l'étude et de la définition de l'architecture de la machine, menée parallèlement [Rub92]. La conclusion était que pour les applications étudiées sur ce réseau, un buffer de trois octets était largement suffisant.

Avant de procéder à l'envoi d'un message, il faut s'assurer que le buffer est vide. La porte ET à trois entrées permet de signaler à la partie contrôle l'état du buffer. Les bascules D permettent à la fois d'implémenter le principe des FIFO et d'indiquer à tout moment l'état de chaque position mémoire de la FIFO. A chaque transfert d'un paquet de message de la mémoire locale vers le buffer, le signal *Send* est activé, déclenchant le mécanisme de décalage des paquets dans le buffer.

Pendant le retrait du message par l'aiguilleur, c'est le signal *Vider* qui cadence le pas de décalage. Il faut préciser à ce niveau que le principe du tampon FIFO repose sur le temps de propagation des registres qui pour cette fois ci est considéré comme un avantage au lieu d'un inconvénient dans la plupart des cas. Le multiplexeur permet de faire passer un 1 logique pendant le remplissage du buffer et un 0 logique lors du retrait du message par l'aiguilleur. La figure 3.27 montre le schéma fonctionnel de cette partie d'interface.

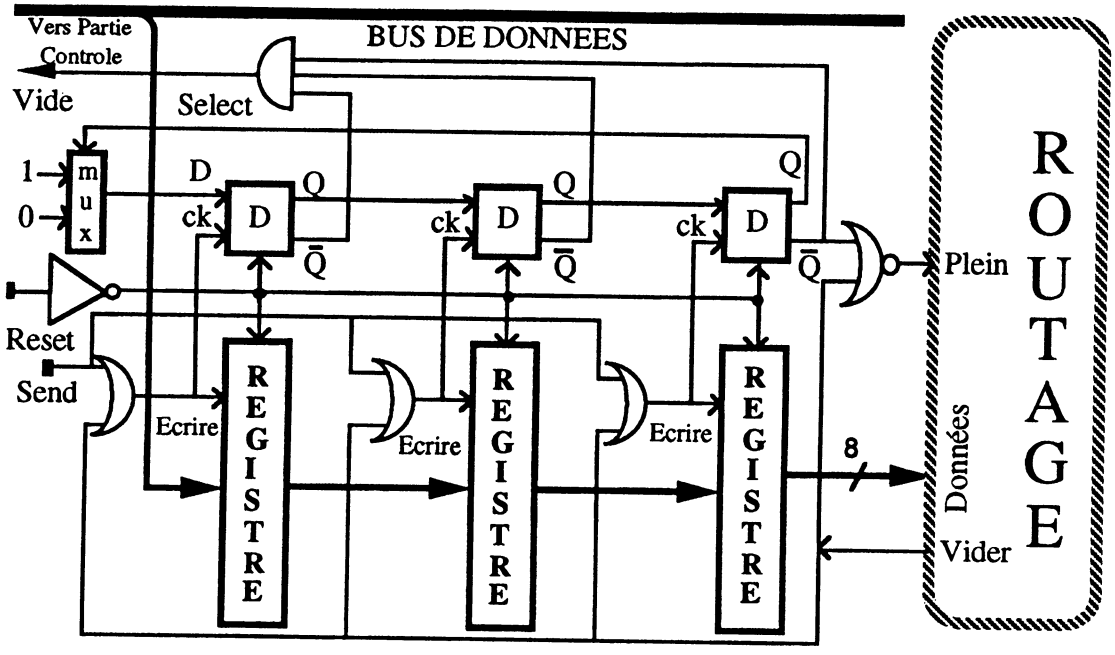


Figure 3.27 Interface traitement routage

3.4.2 PARTIE CONTRÔLE

La partie contrôle est du type deux niveaux d'interprétation, utilisant un générateur de temps. Elle est constituée du registre d'instructions, d'un petit automate et de multiple PLAs (figure 3.16). Le premier niveau d'interprétation est effectué à l'aide d'un PLA de décodage (groupe) qui sert à classer les instructions en groupes selon leurs propriétés. Le deuxième niveau est entrepris par un ensemble de PLAs où chacun gère une unité de la partie opérative et par l'automate qui sert comme générateur de temps. La figure 3.28 montre la structure en blocs fonctionnels de la partie contrôle.

Les groupes d'instructions sont calculés à partir du registre d'instructions. Ces dernières sont classées non seulement en fonction de leurs propriétés mais aussi du chemin parcouru dans le graphe de l'automate (figure 3.29). Les instructions qui parcourent le même chemin dans le graphe de l'automate lors de leurs exécutions, appartiennent au même groupe.

Après la phase d'initialisation, l'état suivant de l'automate est calculé en fonction de l'état présent, des propriétés de l'instruction (groupe) et éventuellement des signaux de compte rendu venant de la partie opérative comme le compteur cptr, les codes condition de branchement, Flag, vide, plein, ...

3.4.2.1 GRAPHE DE CONTRÔLE

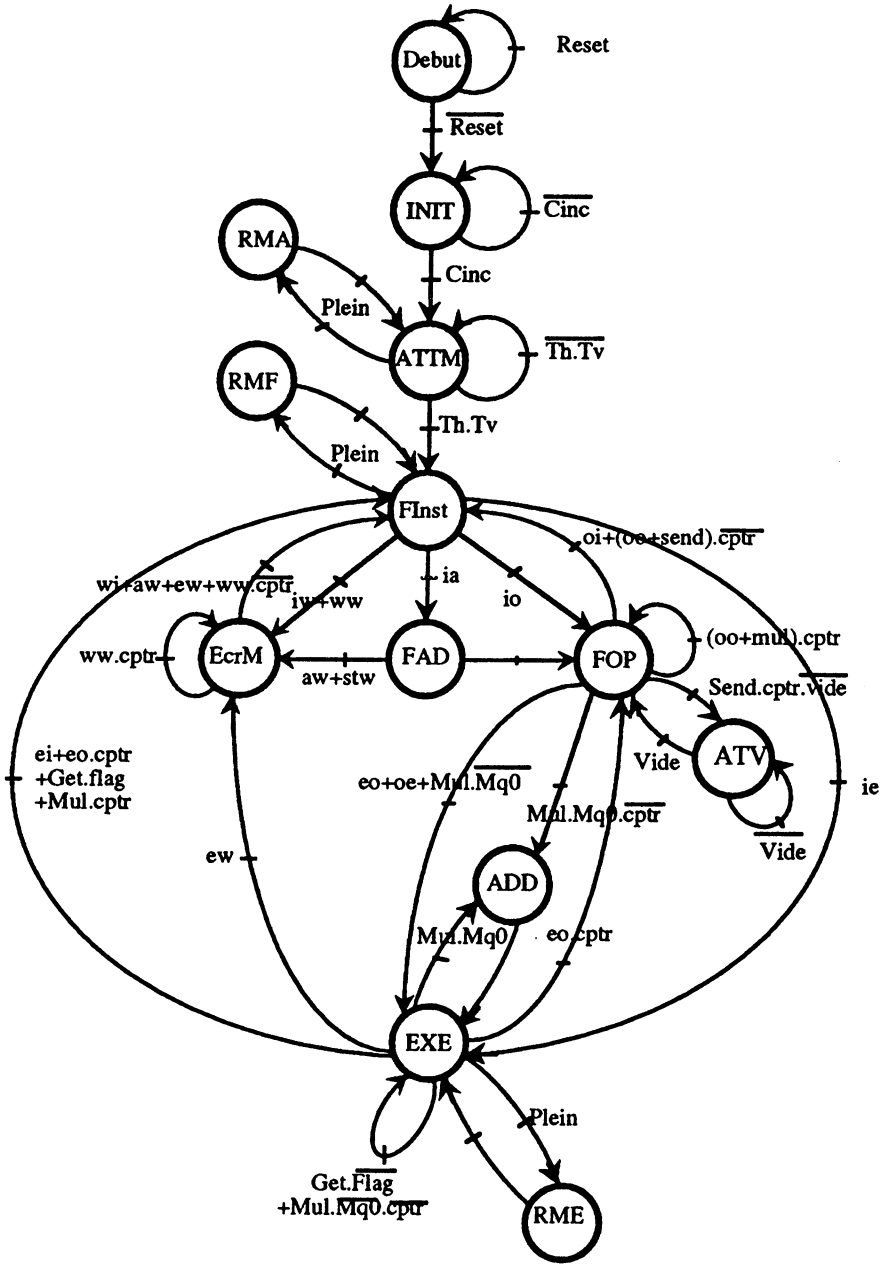


Figure 3.29 : Graphe de l'automate de contrôle

(Le graphe de contrôle avec la légende complète est donné en annexe 4)

Les commandes pour la partie opérative sont générées par des PLA et sont calculées en fonction du code instruction, du code des états de l'automate, des signaux de compte rendu et des signaux de contrôle externes.

A la fin d'un cycle fetch, le code instruction est stocké dans le registre instruction IR et le registre auxiliaire AR.

Avant le début du cycle qui est le fetch, les signaux des groupes sont déjà prêts puisque le registre IR est directement connecté au PLA de décodage des groupes sans aucune commande de lecture pour le registre ou début de calcul pour le PLA.

Après l'échantillonnage des signaux des comptes rendus, l'état suivant de l'automate est calculé. Le code généré est exploité pour calculer les commandes pour la partie opérative pendant le premier quart du cycle.

La figure 3.30 montre la structure détaillée de la partie contrôle et la figure 3.31 montre le chronogramme de fetch, et du décodage d'une instruction.

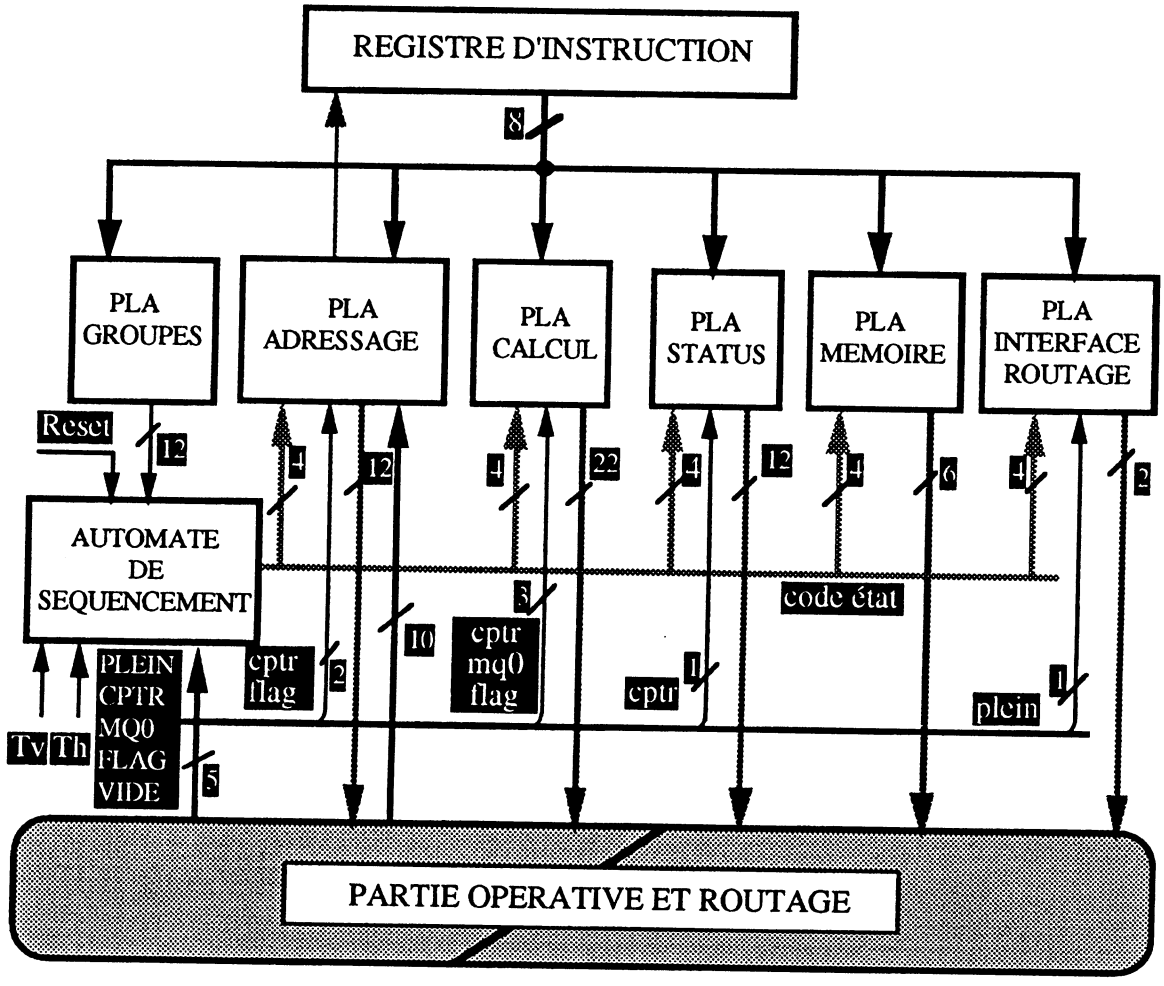


Figure 3.30 : Structure détaillée de la partie contrôle

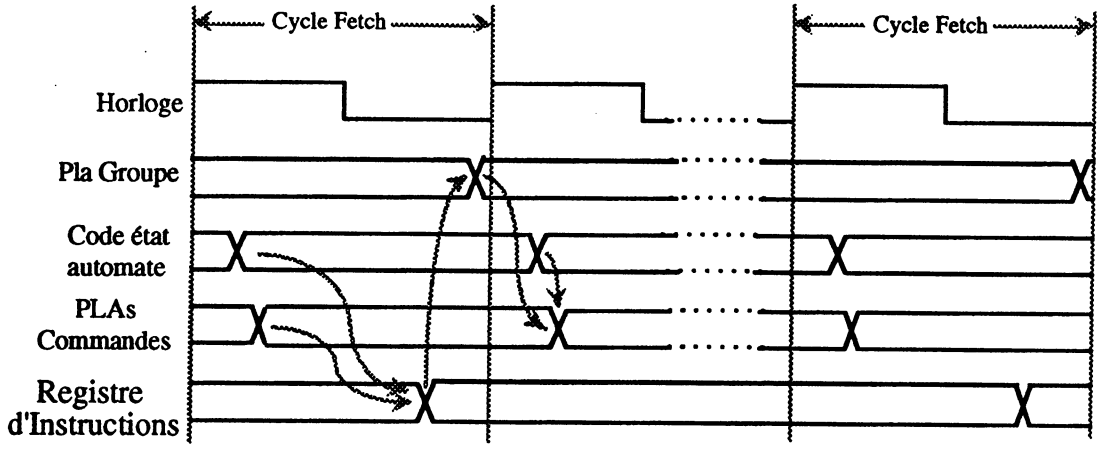


Figure 3.31 : Chronogramme de partie contrôle

La fréquence de fonctionnement de la partie traitement dépend d'une façon très profonde de la technologie utilisée, des moyens, de la méthode et de la manière de réalisation. Elle est déterminée en fonction du temps nécessaire au décodage des instructions (PLA Groupe), de la capacité de la partie contrôle à générer les commandes rapidement (PLA le plus lent) et du temps d'exécution de l'opération la plus lente au niveau de la partie opérative. Il est évident que l'opération d'addition consomme le plus de temps. Les autres cycles consistent, pour la plupart, à des accès mémoire où leurs temps ne dépassent pas les 20 ns (en lecture et incluant les temps d'établissement des signaux nécessaires à un accès mémoire.

L'addition étant basée sur les portes ou exclusif, le temps de calcul est proportionnel à la taille des opérandes (8 bits). Une addition est effectuée en 40 ns car les portes XOR ont un temps de propagation moyen de 2,5 ns. En se basant sur le tableau des temps de calcul des différents PLA (tableau 3.11), on peut déterminer le temps nécessaire pour un cycle d'exécution d'une addition comme suit :

- Le temps nécessaire au PLA de l'automate pour le code de l'état présent,
- Le temps de traversée des bascules de l'automate,
- Le temps nécessaire au PLA calcul pour générer les commandes et la traversée des mux de l'entrée de l'UAL,
- Le temps de l'additionneur,
- Le temps de traversée des mux de sélection des opérations et le chargement des accumulateurs en fin de cycle (Le temps du PLA Groupe intervient seulement pendant les cycles fetch).

Éléments	Temps de propagation moyen
PLA Groupe	10 ns
PLA Automate	9,2 ns
PLA Calcul	19 ns
PLA status	17 ns
PLA Adresse	12 ns
PLA mémoire	9,7 ns
PLA interface Routage	7,1 ns

Tableau 3.11: Temps de calcul des PLAs

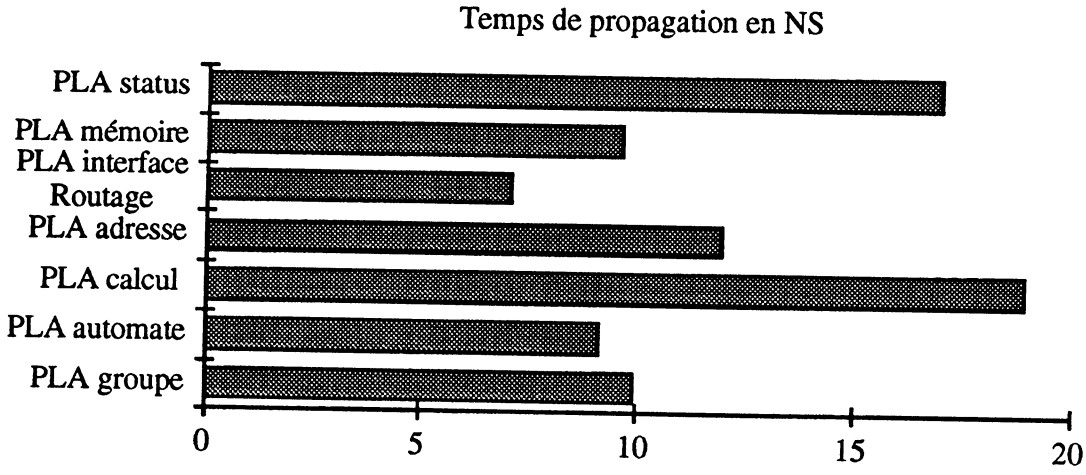


Figure 3.32 : Temps de réponse des Pla

La somme des délais est de l'ordre de 80 ns permettant l'utilisation d'une fréquence de 12,5 MHz. La fréquence de fonctionnement du routeur étant de 25 MHz, un seul signal d'horloge est à générer et peut être utilisé pour les deux parties constituant la cellule en passant par un diviseur par deux. Les effets du retard causé par la division ne gênent pas le bon fonctionnement de la cellule puisque les échanges entre les deux parties sont effectués d'une manière asynchrone.

3.4.2.2 SYNCHRONISATION PARTIE OPÉRATIVE ET PARTIE CONTRÔLE

La synchronisation dans les microprocesseurs entre la partie opérative et la partie contrôle appartient à l'ensemble des problèmes majeurs qui sont rencontrés lors de leur mise au point. En effet, Il faut savoir à quel moment chaque commande doit être activée au niveau d'un cycle et au niveau de l'algorithme d'interprétation. Les commandes ont été classées en trois groupes :

Groupe 1 : Commandes qui peuvent être activées tout au début du cycle comme les contrôles des buffers trois états, des registres d'adresse ou bien pour charger le bus de données à partir du registre d'interface routage-traitement ou l'un des accumulateurs.

Groupe 2 : Commandes de lecture ou écriture de la mémoire qui doivent être activées au milieu du cycle après que le bus d'adresse et/ou de données soient chargés et stables.

Groupe 3 : Commandes qui doivent être activées en fin de cycle qui concernent surtout les commandes de chargement des différents registres ou l'un des accumulateurs après une lecture mémoire ou transfert inter-registres.

Pour établir une synchronisation entre la partie opérative et la partie contrôle, deux signaux supplémentaires (sg2 et sg3) ont été générés à partir du signal d'horloge (figure 3.33). Les commandes des groupes 2 et 3 sont validées par les deux signaux sg2 et sg3 respectivement à travers des portes ET (figure 3.34).

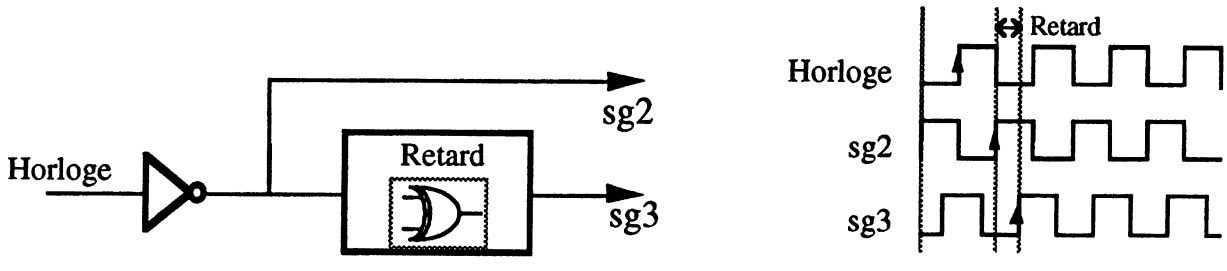


Figure 3.33 : Synchronisation

La durée du retard est déterminée essentiellement par le temps d'accès de la mémoire locale car les commandes du groupe 2 sont principalement celles

qui concernent la mémoire. Dans notre cas, la mémoire a été générée à l'aide d'un outil automatique intégré dans Cadence.

Le temps d'accès de cette mémoire est de l'ordre de 15 ns à partir de l'activation des commandes de validation de celle-ci et du signal de lecture/écriture.

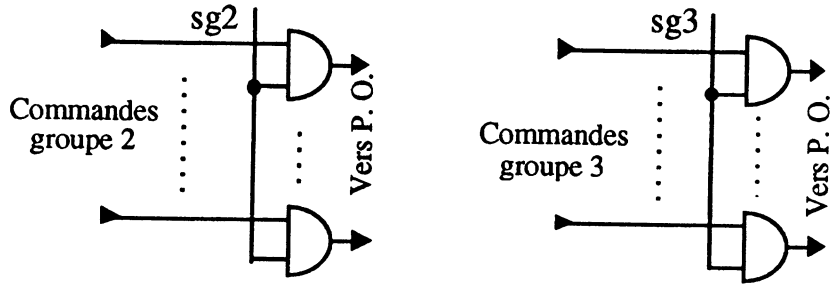


Figure 3.34: Validation des commandes

3.5 CONCLUSION

Pour une première version du circuit, la structure de la partie contôle par ses deux niveaux d'interprétation et sa modularité peut offrir une entière satisfaction. Ce choix a largement facilité le calcul des équations des monômes et la mise au point lors de la réalisation. Par ailleurs, des améliorations restent à envisager par des techniques de pipeline dans le décodage des instructions, ainsi que pour le fetch des opérandes et des adresses. Ce qui permettrait d'améliorer la fréquence de fonctionnement de la partie traitement.

En ce qui concerne le jeu d'instructions, on peut conclure qu'il est assez riche et justifié pour la plupart des applications susceptibles d'être implémentées sur le réseau.

CHAPITRE 4

RÉALISATION DU CIRCUIT

Selon le processus général de la conception des circuits intégrés, la première étape consiste à procéder à la validation de l'approche architecturale par des outils de description et de simulation à un niveau fonctionnelle. Pour cela le langage de description matériel de haut niveau VHDL a été utilisé (actuellement, il est le langage universel pour cette tâche).

Sa puissance réside dans le fait qu'il soit un langage de haut niveau d'abstraction et indépendant de la technologie. Il possède également un large spectre de domaines d'utilisation et s'élève bien au dessus du cadre des circuits intégrés pour s'appliquer aux cartes de composants et même à des systèmes entiers comprenant à la fois des parties matérielles et logicielles.

Les étapes suivantes consiste à se servir des outils de CAO pour valider la conception du circuit d'une manière plus fine en utilisant des simulateurs logiques et procéder à la réalisation du circuit moyennant le reste des outils de placement , du routage, de la vérification et de la conversion de base des données.

Dans les paragraphes qui suivent, nous présentons la démarche entreprise pour valider la conception et aboutir à la réalisation finale d'une première version d'un circuit contenant une cellule.

4.1 DESCRIPTION VHDL

La description VHDL du circuit effectuée par [Had92] a pour but non seulement de valider l'architecture globale du circuit mais aussi de permettre son utilisation dans les tâches suivantes :

- Simulations fonctionnelles du réseau cellulaire avec d'éventuels différents types de routeurs.

- Étude de la testabilité du circuit et de la machine.
- Entrée pour une interface avec des compilateurs de silicium pour directement obtenir un fichier exploitable par les fondeurs de circuits et de réaliser d'autres versions du circuit. La différence des versions du circuit peut se porter aussi bien dans une technologie nouvelle que dans d'éventuelles améliorations.

LES SIMULATIONS VHDL

Les simulations en VHDL ont été portées sur la partie routage et la partie traitement. Les simulations VHDL de la partie routage ont permis de vérifier l'algorithme de routage ainsi que les différentes fonctionnalités qui lui ont été associées [Had92] :

- le parallélisme dans l'acheminement (la bande passante).
- le calcul de destination.
- la mise à jour de l'adresse relative.
- le transfert par paquet sériel.
- le rangement des paquets de message dans la mémoire locale du processeur.
- le rôle des signaux de transparence dans la partie routage.

En ce qui concerne la partie traitement, les simulations VHDL ont porté sur des petits programmes rangés auparavant dans la mémoire (en utilisant le routeur bien sûr) composés de quelques instructions simples de calcul comme LDA, LDI, TAI, TIA, ADD, MUL, ADCW, LDAW, INC, XOR avec différents modes d'adressage et d'instructions de communication comme SEND et GET.

Ceci a permis de vérifier le fonctionnement du processeur du point de vue calcul ainsi que du point de vue communication [Had92].

4.2 RÉALISATION

Comme pour les étapes précédentes, il y a aussi plusieurs approches pour aborder la réalisation d'un circuit. Ces approches touchent au choix de la technologie et les méthodes de réalisation.

En ce qui concerne la technologie, le CMOS actuellement apparaît le meilleur en termes du taux d'intégration et consommation [Can86].

Pour ce qui est des méthodes de réalisation, au départ, on avait opté pour le *full custom* car il permet d'obtenir la meilleure performance en ce qui concerne la compacité. Du point de vue fabrication, plus la taille du circuit est réduite plus la probabilité d'avoir des circuits bon pour le fonctionnement est élevée. Cette méthode résulte en un coût important autant en effectif humain qu'en temps de réalisation.

Le *full custom* reste actuellement utilisé surtout pour des circuits à complexité réduite, dans des petites parties pointues d'un circuit complexe et par les fournisseurs d'outils CAO. Des bibliothèques assez riches de standard cells allant d'une simple porte logique d'un inverseur jusqu'à un processeur sont apparues sur le marché. L'utilisation de ces cellules caractérisées permettent de simplifier les différentes phases de réalisation en ce qui concerne les simulations, le placement et le routage, les vérifications nécessaires et par la même occasion le coût de réalisation.

En plus des bibliothèques de cellules, sont fournis également des générateurs de blocs de circuit qui présentent une régularité du point de vue fonctionnement ou implantation comme les mémoires, les PLAs et les multiplieurs.

Un autre argument important qui nous a orienté pour réaliser le circuit avec des cellules standard est la rapidité avec laquelle les technologies progressent. Pour changer de technologie, en *full custom* , il faut refaire tout le dessin des cellules de base et le routage à la main. Ce qui n'est envisageable pour un circuit contenant des dizaines de milliers de transistors avec une faible régularité.

Malgré l'existence d'un procédé qui consiste à faire passer le dessin des masques par une opération de *shrink* (réduction des dimensions par facteur lié au rapport géométrique des deux technologies), la fiabilité de cette méthode reste faible.

Avec l'approche du standard cell, il est plus facile de passer d'une technologie à une autre plus intéressante sur le plan des dimensions et/ou des performances juste par un simple changement de la bibliothèque en terme de chemins dans les fichiers de paramètres associés aux outils CAO.

Bien que les outils de CAO automatiques produisent des circuits moins denses, ils garantissent une plus grande sûreté de conception. Ce qui est essentiel pour un circuit de recherche.

Dans un premier temps, notre objectif était de réaliser un circuit contenant 4 cellules. Par la suite, nous avons préféré de réaliser un circuit contenant une seule cellule pour faciliter le test après fabrication. Dans une deuxième étape, il sera plus facile de faire un second circuit intégrant plus de cellules juste par la duplication de l'implantation d'une même cellule dans un boîtier.

Contrairement à la phase de conception, la réalisation du circuit a été effectuée en utilisant la méthode ascendante en utilisant les cellules de la bibliothèque de ES2 (European Silicon Structures) de Eurochip, associée avec *SOLO 2030* en une technologie CMOS 1.2 µm à deux niveaux de métal.

Il s'agissait de dessiner le circuit sous forme de schémas logiques à partir des bibliothèques fournies par le fondeur en commençant par les cellules de base (registres, multiplexeurs, buffers trois états, bouts de circuits combinatoires etc.) jusqu'aux différents blocs fonctionnels du processeur ensuite au circuit final. A partir du deuxième niveau hiérarchique, les représentations symboliques des cellules des niveaux précédents sont utilisés [Sol90]. A chaque étape, les simulations logiques sont effectuées (voir paragraphe 4.3).

4.2.1 PARTIE ROUTAGE

La réalisation de la partie routage consistait à décrire un seul bloc d'entrée au niveau d'un tampon d'entrée et un seul bloc de sortie et dupliquer la même description pour le reste des tampons d'une cellule. Le concept du routeur tout parallèl mérite que les interconnexions (signaux de contrôle et de transfert de données) entre les bloc d'être soigneusement effectuées afin d'éviter les risques d'erreurs.

LES PARTIES CONTRÔLE

L'outil SOLO 2030 ne possède pas un moyen de générer une partie contrôle à partir d'une description d'un graphe. Pour cette raison, les parties contrôle des deux blocs de base constituant la partie routage, en l'occurrence bloc d'entrée et bloc de sortie, ont été réalisées à l'aide d'un outil de synthèse GASP développé par APTOR. La simplicité des deux parties de contrôle ne présentait pas de difficultés à les faire à la main. Mais l'utilisation de GASP avait deux buts :

- obtenir un codage compact optimal des codes d'état des automates,
- obtenir une matrice binaire optimisée et exploitable par le générateur automatique PLA intégré dans SOLO 2030.

4.2.2 PARTIE TRAITEMENT

La réalisation de traitement a été faite en deux phases. Il fallait d'abord commencer par la partie opérative afin de recenser toutes les commandes nécessaires, ensuite passer à la partie contrôle.

Une fois le jeu d'instructions et l'architecture du chemin de données définis, nous avons procédé dans une première étape au dessin des schématiques des cellules de base communes aux quatre parties de la partie opérative (calcul, adresse, mémoire et interface routage) :

- registres 8 bits,
- multiplexeurs,
- buffers trois états.

Dans une deuxième étape, le dessin en schématique des quatre parties du chemin de données s'est fait séparément puisqu'elles sont liées que par les bus de données et d'adresse (voir chapitre 3 paragraphe 3.2.6).

4.2.2.1 PARTIE CONTRÔLE

Le calcul des monômes de la partie contrôle a constitué l'une des tâches les plus délicates lors de la phase pré-réalisation. A défaut d'outils destinés à la génération d'une partie contrôle d'un microprocesseur à partir d'un jeu d'instructions ou d'un graphe d'interprétation de l'algorithme intégré dans SOLO 2030, ils ont été calculés à la main selon le principe suivant :

Pour chaque commande de la partie opérative, noter toutes les instructions, les signaux externes et de compte rendu qui nécessitent l'activation de cette commande.

Les monômes des différents PLA sont en fonction des variables suivantes :

- les codes instructions,
- les codes états du graphe de contrôle,
- les signaux de compte rendu des différents blocs de la partie opérative,
- les signaux de contrôle externes tel que *plein* (interface routage), th, tv, et reset.

Dans un premier temps, les monômes ont été calculés, d'une manière exhaustive, et optimisés par la suite en utilisant l'outil *expresso* afin de réduire le nombre des monômes. Rappelons que la taille d'un PLA dépend du nombre des entrées et des sorties et du nombre des monômes. Les tables en binaire exploitables par des générateurs automatiques de PLA sont données en annexe 5. Le nombre totale des monômes est de 334 pour 52 commandes. La répartition des monômes et des commandes sur les quatre parties du chemin de données est montrée dans la figure 4.1.

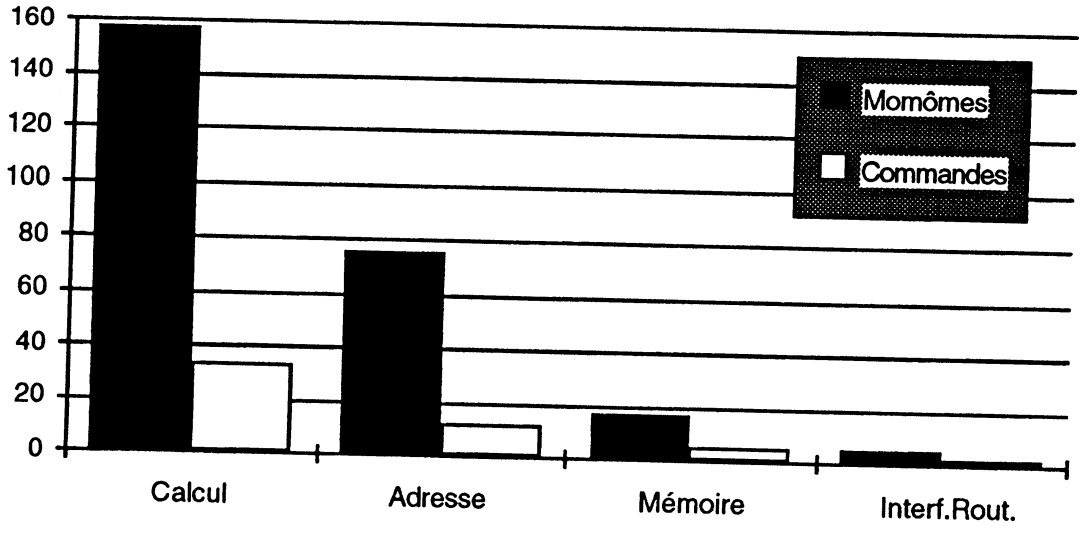


Figure 4.1 : Répartition des monômes et commandes

4.2.2.2 L'AUTOMATE

Pour les mêmes raisons que celles des parties contrôle du routeur (codes des états du graphe compacts optimisés et table du PLA), l'outil GASP a été encore une fois utilisé. A partir d'une description textuelle du graphe de l'automate, GASP génère les fichiers contenant les codes et une matrice binaire qui peut être utilisée aussi bien pour la synthèse logique (standard cells) que pour un générateur automatique de PLA.

4.2.2.3 TEST DES PLAS

Une fois les PLA générés et optimisés par *expresso* nous avons procédé à la vérification logique des différents PLA vu le rôle important qu'ils jouent dans la partie traitement du processeur. Des simulations logiques sur chaque PLA ont été effectuées. Les vecteurs de test ont été générés à l'aide de programmes écrits en langage C. Selon le nombre des entrées du PLA, les programmes permettent de générer toutes les combinaisons possibles (codes d'instructions, signaux de compte rendu et externes). Les vérifications consistaient en l'observation des commandes activées en fonction du code instructions avec la combinaison des signaux de compte rendu et externes. Les listings des fichiers de test sont donnés en annexe 5.

Les programmes en langage C génèrent des fichiers textes contenant deux colonnes. La première colonne contient des valeurs de temps (un pas de 100 ns) et la deuxième contient toutes les combinaisons binaires et hexadécimales possibles selon le nombre de bits spécifié. Ces fichiers sont destinés à être insérés dans les fichiers d'entrées de simulation logique des différents PLA constituant les parties de contôle du routeur et la partie de traitement. Le nombre de bits spécifié dans chaque fichier correspond au nombre des entrées de chaque PLA.

4.3 LES SIMULATIONS LOGIQUES

Les simulations logiques avaient pour but de valider la conception des différentes parties du circuit d'une manière plus fine en commençant par les niveaux hiérarchiques les plus bas jusqu'au niveau le plus haut.

Comme c'est connu dans les circuits électroniques, encore plus avec des circuits réels :

Si un circuit marche correctement pour la première fois, cela veut dire qu'il y a une erreur quelque part.

Les problèmes de synchronisation, d'initialisation et d'adaptation des *fanin* et *fanout* surgissent au moment de cascader les différents étages et niveaux hiérarchiques d'un circuit complexe. Parallèlement avec les simulations logiques, les valeurs temporelles sont mesurées pour la détermination de la fréquence de fonctionnement du circuit final.

Dans les paragraphes qui suivent, nous présentons la méthode entreprise pour effectuer les simulations logiques. Cette méthode consiste à spécifier les phases et les points d'observation essentiels dans les différentes parties du circuit.

4.3.1 PARTIE ROUTAGE

Les simulations logiques sur la partie routage ont été effectuées après la saisie de schématiques de chaque niveau hiérarchique en commençant par les cellules de base communes aux différents blocs comme les registres et le "drapeau" jusqu'au routeur complet. Ceci a permis d'apporter rapidement les corrections qui s'imposaient et les améliorations qui s'afficher nécessaires pour respecter l'algorithme de routage d'une part et augmenter les performances d'autre part.

Au niveau d'un tampon d'entrée il fallait observer les fonctionnalités suivantes :

- Le protocole de communication entre deux cellules voisines,
- La détection de présence de message,
- Le calcul de la direction,
- La mise à jour de l'adresse,
- Le multiplexage (la non mise à jour des paquets de Tag et de données),
- L'attente des acquittements avant de vider le tampon.

En ce qui concerne le bloc de sortie, les points importants à observer :

- La détection de présence de requêtes,
- L'échantillonnage des requêtes,
- L'arbitrage et priorité tournante,
- L'envoi de l'acquittement vers le bloc d'entrée.

L'étape suivante consistait à simuler un tampon avec un bloc d'entrée et un bloc de sortie. Une suite de messages est injectée dans le tampon pour être récupérée à la sortie du bloc de sortie. Cette étape a permis de mesurer le temps d'acheminement d'un paquet de message et de procéder à des ajustements pour résoudre les problèmes de synchronisation qui se présentaient.

La dernière étape avait pour but de simuler le routeur complet. Ces simulations ont permis de valider le concept du routeur tout parallèle où le croisement des messages était le point essentiel qui méritait d'être observé.

4.3.2 PARTIE TRAITEMENT

De la même manière, les simulations logiques de la partie traitement ont été effectuées parallèlement avec la saisie des schématiques des différents niveaux hiérarchiques en commençant par les cellules de base communes aux différentes parties du chemin des données (calcul, adresse, mémoire et interface routage). Ensuite chaque partie est simulée individuellement. Les simulations de la partie opérative globale sans la partie contrôle n'ont pu être faites à cause du nombre important des commandes. Ce qui nécessite un fichier d'entrée de simulation coûteux en temps de mise au point.

Dans la partie calcul les points d'observation sont :

- Les cinq opérations de l'UAL,
- La sélection des opérations de l'UAL,
- Le multiplexage au niveau de l'UAL (Accumulateur A ou registres MQ),
- Le multiplexage et décalages au niveau des accumulateurs,
- Les bits du registre d'état,
  - La détection de la retenue,
  - Le multiplexage au niveau de l'entrée de la bascule de la retenue,
  - La mise à 1 et à 0 de la retenue,
  - La détection d'un zéro,
  - Le calcul du débordement,
  - La détection d'une valeur négative.
- Le calcul des conditions de branchement,

Les simulations de la partie adresse ont permis de voir les points suivants :

- Le chargement et le multiplexage au niveau des registre PC, I, RA (du bus de données ou de l'incrémenteur),
- L'incrémenteur,
- Le compteur,
- Le dispositif du remplacement des bits de poids fort du registre AR pour l'implémentation de l'adressage absolu court,

Les points d'observation dans la partie qui inclut la mémoire sont :

- Le fonctionnement globale de la mémoire,
- La gestion du 9 ième bit par la bascule D.

En ce qui concerne l'interface avec le routeur, les simulations ont vérifié les fonctionnalités suivantes :

- La détection de présence d'un paquet de message,
- L'absorption de l'entête du message (adresse relative),
- Le fonctionnement du principe d'une mémoire FIFO,
- La préparation des signaux de chargement des bus pour le rangement du message dans la mémoire (le Tag sur le bus d'adresse et la Donnée sur le bus de données.

PARTIE CONTRÔLE

Les simulations de la partie contrôle ont été réalisées en deux phases : les simulations de l'automate et les simulations de la partie contrôle complète (automate et les PLA).

Les simulations de l'automate avait pour but de vérifier le bon fonctionnement du PLA *groupe* et le PLA de calcul des états suivants en fonction des différents types d'instructions, les signaux de compte rendu et des signaux externes. Il s'agissait de charger le registre d'instructions par l'un des codes de la table d'instruction et observer les états par lesquels l'automate passe dans le graphe.

4.3.3 SIMULATIONS DU CIRCUIT COMPLET

Les simulations du circuit complet ont permis d'apporter les dernières retouches en ce qui concerne la conception du processeur ainsi que le routeur. Elles consistaient essentiellement d'ajuster les synchronisations suivantes :

- Entre le routeur et la partie traitement (interface routage) : Émission et réception des messages,
- Entre la partie opérative et la partie contrôle,
- Entre les différentes parties du chemin de données.

D'autre part, elles ont également permis de vérifier un ensemble de fonctions du circuit qui concernent surtout la partie traitement difficile à examiner avec les simulations précédentes :

- Le fonctionnement globale de la partie contrôle :
  - l'initialisation de la mémoire,
  - la détection de présence des messages pendant
    - l'initialisation,
    - le fetch d'une instruction,
    - l'exécution d'une instruction.

Le simulateur logique *Silos* utilisé tout au long de ces simulations, fournit un fichier *si.llog* contenant les estimations des capacités au niveau de chaque nœud en termes de *fanin et fanout*. Les nœuds sur lesquels il y avait un excés de charge ont été amplifiés avant de procéder à d'autres simulations.

Les simulations du circuit complet consistaient à injecter, en se servant du routeur, une instruction (en code machine) dans la mémoire locale et à commander le début d'exécution (à partir de l'adresse 0) à l'aide des signaux TH et TV (voir chapitre 2 paragraphe 2.4.3.1). Les bus et les points d'observation déjà cités dans les paragraphes précédents sont examinés et les modifications nécessaires sont apportées en cas de mauvais fonctionnement aussi bien sur le plan de synchronisation que sur le plan opérationnel. Cette procédure est répétée pour chaque type d'instructions.

En phase final, des petits programmes d'une taille d'une dizaine d'instructions (boucles, routines de calcul, remplissage et parcourt d'un tableau) sont injectés en mémoire pour conclure la conception du circuit avant de procéder à l'implantation.

4.4 IMPLANTATION

L'implantation du circuit a été réalisée en se servant du placement et routage automatique intégré dans *Cadence* à partir de la description schématique du circuit. Les éléments physiques constituant le circuit se divisent en trois types :

- Les *Standard Cells* : cellules de la bibliothèque de ES2.
- Les *Macro Cells* : blocs générés par des outils automatiques comme la

mémoire et les PLAs.
- Les connexions.

Les *Standard Cells* sont placées automatiquement dans des zones paramètrées (surface et nombre de rangées de cellules) auparavant spécifiés.

La version de cadence utilisée ne supportait pas le placement automatique des *Macro Cells* , il fallait donc le faire manuellement. Ceci nous a contraint à effectuer plusieurs placements différents des *Macro Cells* avec différentes zones de *Standard Cells* afin d'obtenir une implantation plus optimale.

Le circuit complet avec la couronne des plots occupe une surface de 32,8 mm^2 (5,33 mm x 6,15 mm) pour 33861 transistors (sans la mémoire). Ce qui donne une densité d'intégration de 1033 transistors par mm^2. Cette densité est donc calculée sans tenir compte des transistors de la mémoire. Une donnée difficile à obrenir auprès des représentants de CMP et EUROCHIP. La seule précision qu'on pouvait obtenir est le nombre de transistors par cellule mémoire (6 transistors). La mémoire étant de 256 mots de 9 bits, on a donc 13824 transistors sans le décodage et multiplexage. Après consultations, nous avons estimé la compléxité de la mémoire à environ 15 000 transistors. Ceci nous donne une compléxité totale du circuit d'environ 47 685 avec une densité d'intégration de 1454 transistors par mm^2.

4.4.1 IMPLANTATION DES PLOTS

En ce qui concerne le placement des plots d'entrées/sorties, il n'était pas envisageable de procéder à un placement automatique. Ceci nous aurait fait perdre l'avantage de développer des cartes par simple juxtaposition des circuits en une matrice. Il fallait donc choisir l'option de placement des plots *manuel* dans le *Place & Route* de SOLO 2030 afin d'avoir les sorties d'un circuit en face des entrées du circuit voisin.

Le circuit comporte 92 plots dont 43 plots d'entrées, 40 plots de sorties et 7 plots d'alimentation. Dans les plots d'alimentation, figurent 4 plots de 5v dont deux pour alimenter la couronne et 3 plots de masse. La figure 4.1 montre le plan de brochage et la liste des plots.

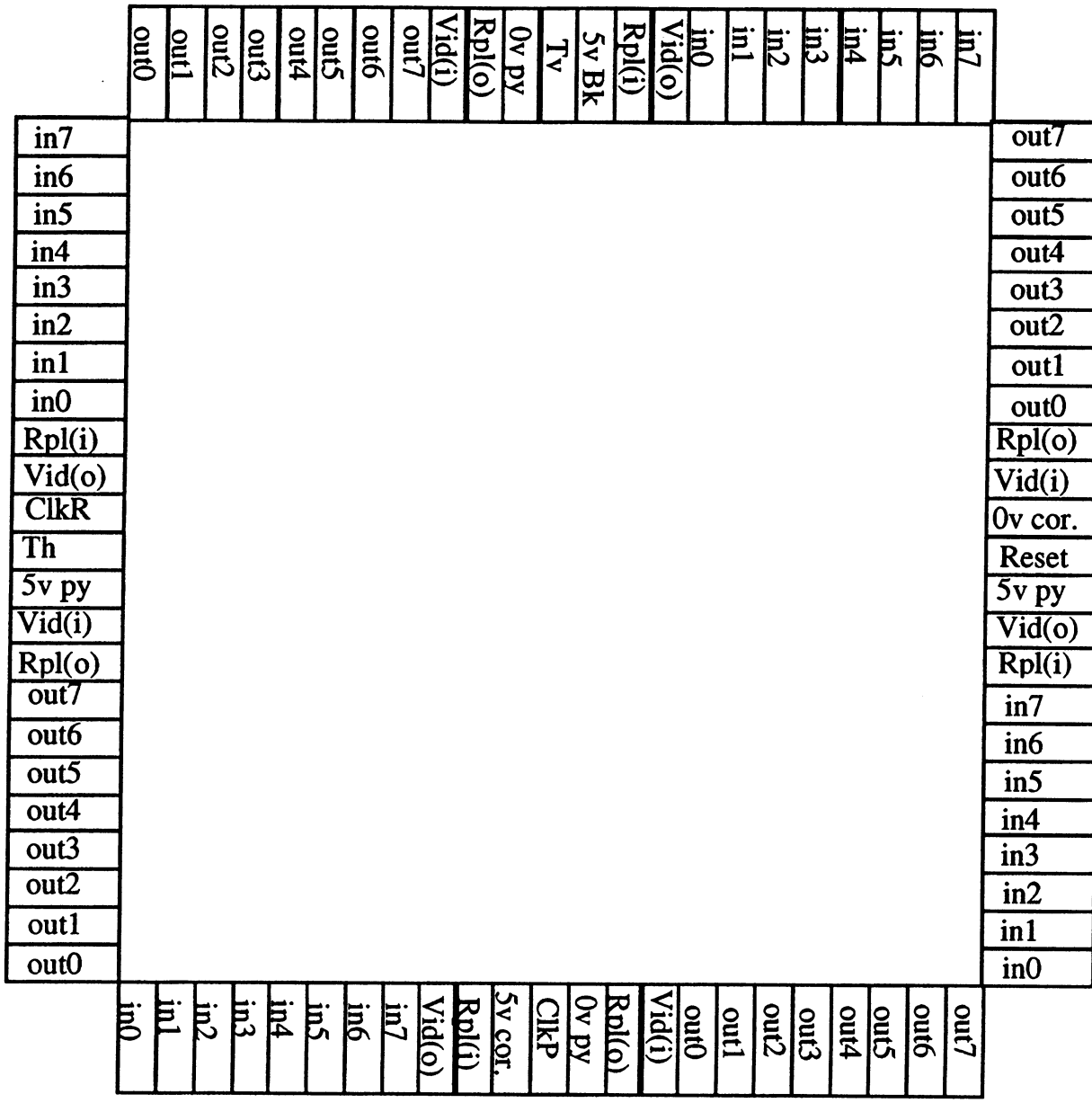


Figure 4.1 : Plan de brochage

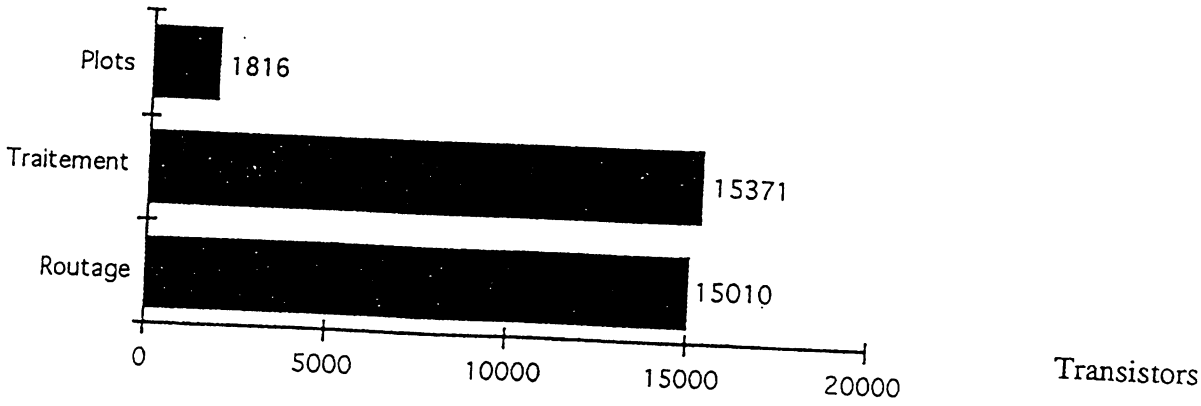
On remarque qu'il y a deux plots pour l'horloge, un pour le routeur (ClkR) qui fonctionne à 25 Mhz et un deuxième pour le processeur (ClkP) dont la fréquence maximale est de 12,5 Mhz.

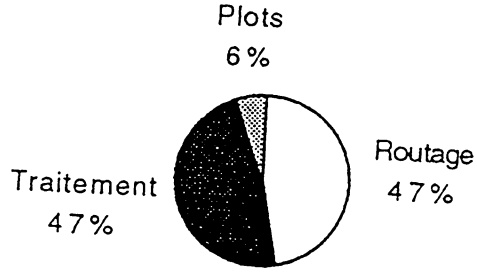
4.4.2 BILAN DES DES COMPLEXITÉS

Il est fort intéressant de dresser un bilan des complexités dans chaque bloc du circuit. Le résultat le plus intéressant de cette réalisation est le rapport des complexités entre le routeur et la partie traitement qui sont comparables. Les

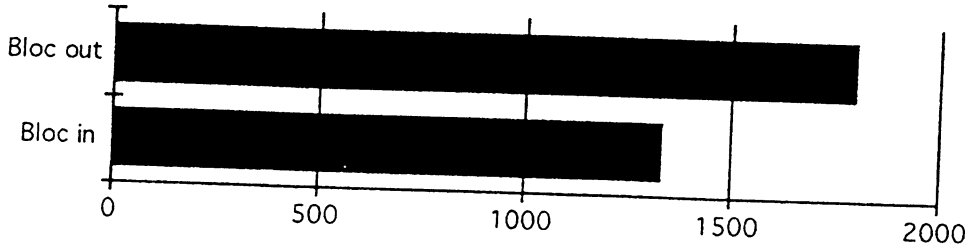
figures suivantes montrent les rapports des compléxités à différents niveaux hiérarchiqus du circuit.

CELLULE (ROUTAGE - TRAITEMENT - PLOTS)

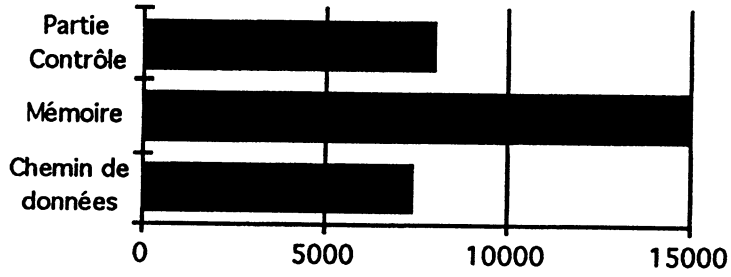


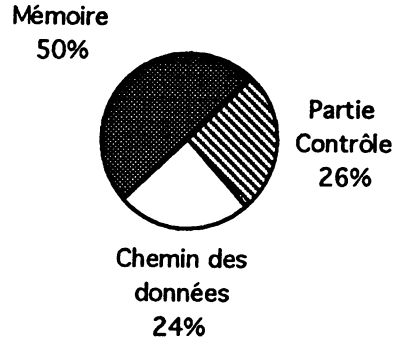


ROUTAGE : BLOC IN - BLOC OUT

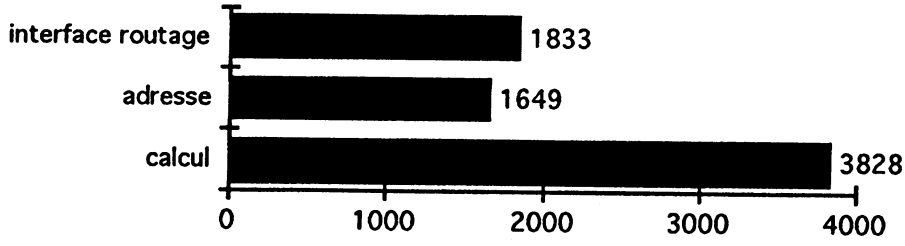


TRAITEMENT : (CHEMIN DE DONNÉES - MÉMOIRE - CONTRÔLE)

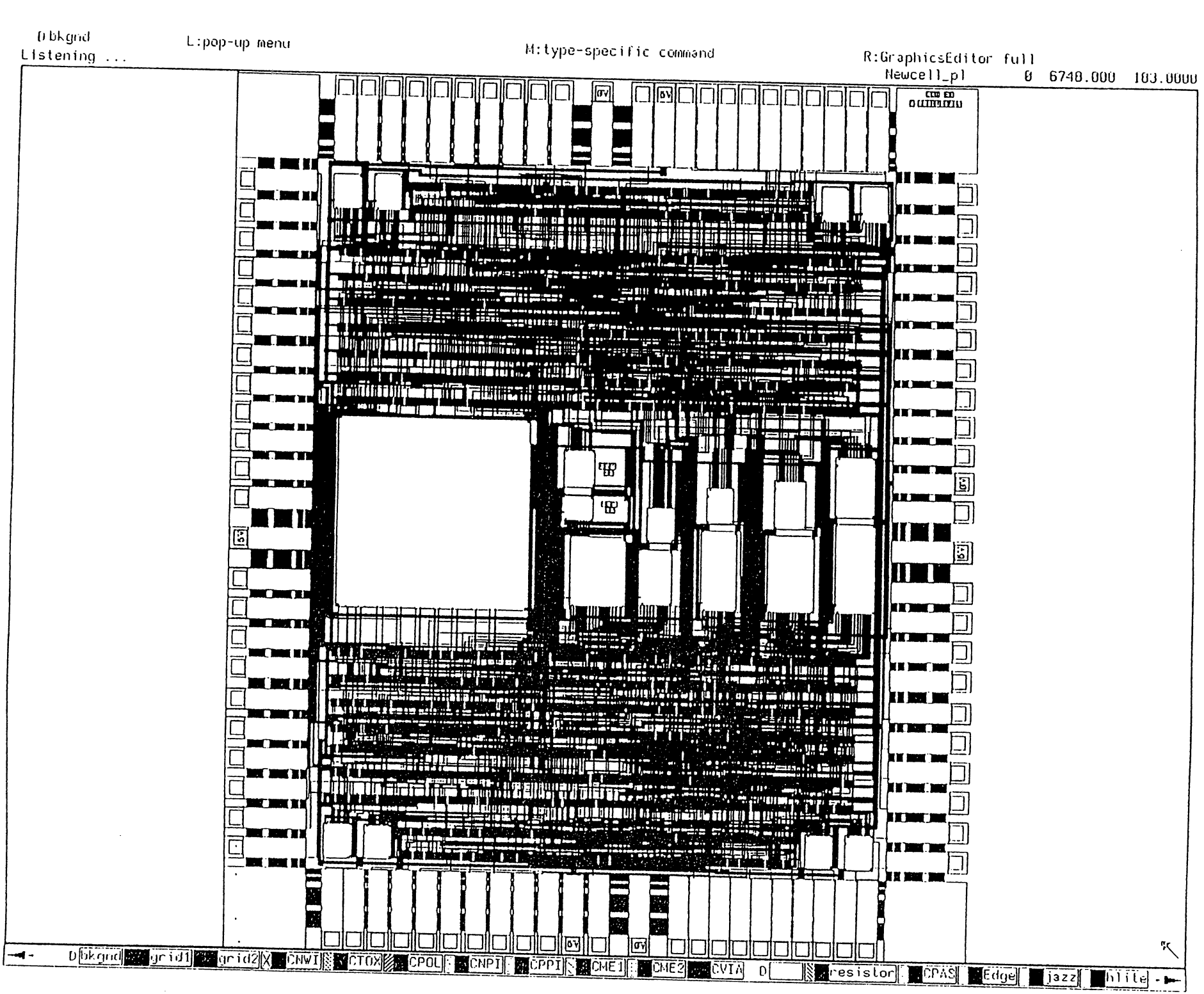




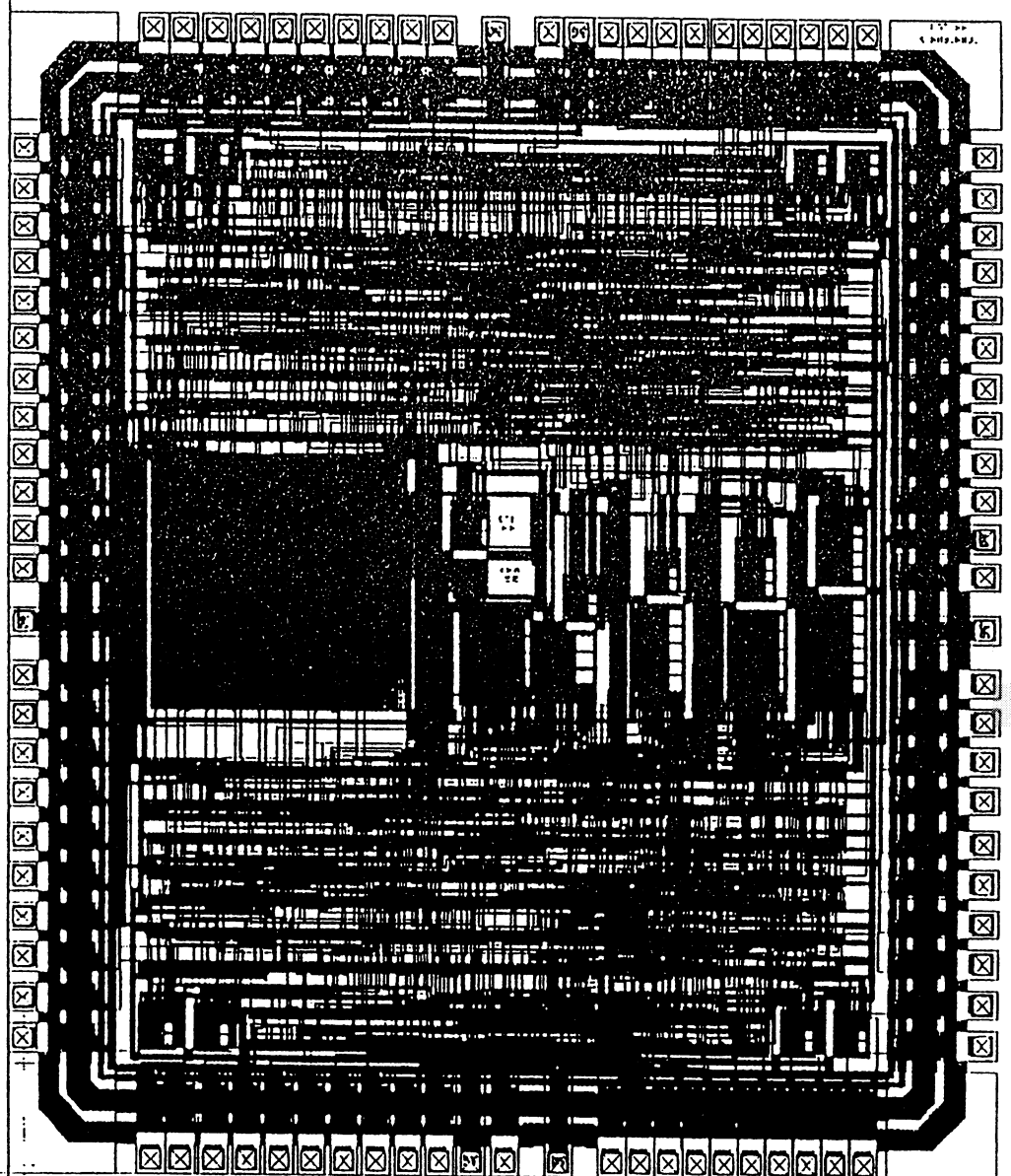
CHEMIN DE DONNÉES (CALCUL - ADRESSE - INTERFACE ROUTAGE)



La figure suivante montre le layout du circuit complet. Il est composé en trois zones : deux contiennent les rangées des *standard cells* et la zone centrale contient la mémoire et les PLA de l'unité de traitement. Les PLA de la partie routage sont disposés aux quatre coins de la cellule. Il est difficile de dresser un bilan des surfaces car les *standard cells* ont été placées d'une manière automatique. Ceci rend encore plus difficile l'extration de la surface consommée par les bus du routeur qui en compte quatre.


D bkgnd
Listening ...
L:pop-up menu
M:type-specific command
R:GraphicsEditor full
Newcell_pl
0 6748.000 103.0000
D bkgnd grid1 grid2 X CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite

CCON L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



Dbkgnd grid1 grid2 CNWI CTOX CNPI CPPI CPOL CME1 CME2 CCON DCPAS CVIA BlackBox Dummy EuroLab

4.5 CONCLUSION

Le fait de bâtir les parties contrôle à l'aide des PLA dans une première version du circuit, a énormément facilité la phase de validation de la conception lors des simulations logiques. Dans les cas d'un mauvais fonctionnement, il suffisait juste d'apporter quelques modifications dans la table du PLA concerné et refaire la génération en moins d'une dizaine de secondes pour que tout rentre dans l'ordre.

Cette facilité de mise au point du circuit a été payée par le coût de la surface consommée par les interconnexions entre les PLA et le reste du circuit dû à leur nombre et à leur taille différente.

Bien que la surface totale du circuit semble élevée par rapport à la technologie utilisée et sa complexité de 33800 transistors (sans la mémoire), il est possible de l'optimiser par l'un des moyens suivants :

- Réaliser le circuit en *Full Custom* ,
- Réaliser le circuit avec une technologie plus fine comme la 1.0 µm ou moins tout en utilisant les nouvelles possibilités récemment introduites dans Solo 2030 suivantes :
  - placement des macro cellules automatique,
  - placement par blocs fonctionnels,
- Remplacer les PLA par des standard cells en utilisant l'outil de synthèse logique et les tables de génération.
- Réaliser le circuit en milieu industriel.

CONCLUSION GÉNÉRALE

Par ce travail, nous avons contribué à la définition fonctionnelle de l'architecture et abordé la conception et la réalisation sur silicium de la brique élément d'un réseau cellulaire de seconde génération. Les réseaux cellulaires de la première génération étaient dédiées à des applications spécifiques. C'est en constatant que le coût de developpement et de fabrication de machines spécifiques est important que nous nous sommes orientés vers une machine à caractère général. D'autre part, il s'est avéré que la complexité des cellules restait comparable à celle d'un petit processeur programmable.

Au depart, nos objectifs consistaient à l'étude d'un réseau cellulaire plus général dans tous ses points de vue : architecturale, algorithmique, programmation, conception et réalisation. Nous avons rapidement débouché sur des solutions de partie traitement à base de processeurs 8 bits dotés d'une mémoire de quelques centaines d'octets. Ce qui nous fait situer dans une position originale entre les machines SIMD 1 bits et les hypercubes à base de processeurs 32 bits MIMD.

Les enseignements que l'on peut tirer sont que le réseau cellulaire reste exploitable en ce qui concerne les aspects purement informatiques (architecture, programmation et algorithme) et présente des performances intéressantes par rapport aux autres types de machines [Lat89], [Pay90] et [Rub92].

Sur le plan d'intégration, il est possible en milieu universitaire de faire des circuits contenant 4 x 4 cellules. On peut être plus ambitieux en milieu industriel aujourdhui où il est possible de faire des circuits 8 x 8 cellules.

Sur le plan réalisation, nous avons réalisé un circuit contenant une cellule. Sur trois circuits testés, on a eu la satisfaction d'en avoir un qui marche à 75% et il reste encore d'autres à tester.

Vu l'importance de la communication dans les architecture parallèles, nous avons, par ce travail, aporté une contribution à la réalisation d'un routeur le

plus performant possible du point de vue rapidité, fonctionnalités et absence d'interblocages. Le routeur utilise un transfert et un contrôle parallèl à base du *wormhole* à taille fixe. Nous avons également conçu les dispositifs nécessaires afin d'augmenter la connectivité des cellules en exploitant les multiplexages et démultipléxages au niveau entrées/sorties des cellules du bord pour permettre la réduction des plots des circuits contenant 4 cellules. En conséquence, la stratégie et les algorithmes d'acheminement des messages doivent subir des modifications. En ce qui concerne les performance, avec la technologie utilisée, nous avons atteint un débit de 6,25 M octets par lien et par cellule. On a donc un débit de31,25 M octets par cellule. Mais le débit global reste dépendant de la topologie.

En ce qui concerne l'unité de traitement, du point de vue architecture, elle permet une programmation assez souple et efficace. Néanmoins, la fréquence de fonctionnement est à améliorer par l'utilisation d'une UAL plus rapide et une optimisation de la partie contrôle.

Maintenant qu'on dispose d'une description détaillée du processeur jusqu'au bas niveau, on reste toujours optimiste de pouvoir présenter un circuit, dans les prochaines versions, plus compact et performant tout en mettant à profit des moyens plus importants aussi bien en matériel qu'humain.

BIBLIOGRAPHIE

[Alm89] G. S. ALMASI ET A. GOTTLIEB, "Highly Parallel Computing", The Benjamin/Cummings Publishing company, Inc. 1989.

[Als92] ALASTAIR D. MCAULAY, "Researchers look to optics to move computer technology forward", Computer in physics, vol. 6, No. 6, Nov/Dec. 1992.

[Anc85] FRANÇOIS ANCEAU, "The Architecture of Microprocessors", ADDITSON WESLEY, 1985 .

[Ath88] ATHAS (W.C.), SEITZ (C.L.). "Multicomputer : message passing concurrent computers", *Computer* (Aug 1988) 21, 8.

[Bae80] J. L. Baer, "Computer systems architecture", Computer Science Press, 1980, pp 76-131.

[Boi91] F. BOITEUX, M. ROBICHON, "Réalisation d'une interface matérielle et logicielle entre un ordinateur hôte Machintosh II et un réseau parallèle cellulaire", rapport de stage de fin d'étude, ENSIMAG-ENSERG, section Architecture des systèmes numériques, juin 1991.

[Can86] M. CAND, E. DEMOULIN, J-L. LARDY, P. SENN, "Conception des Circuits Intégés Mos", 1986 Eyrolles

[Chu82] S. CHUQUILLANQUI, PEREZ SEGOVIA, "PAOLA : a tool for topological optimization of large PLAs", 19th Design Automation Conference, Las Vegas, 14-16 juin 1992

[Cor87] R. CORNU-EMIEUX, G. MAZARE, Ph. OBJOIS, "An integrated highly parallel architecture to accelerate logical simulation", proceedings of IEEE ISELDECS 87, décember 1987.

[Cor88] R. CORNU-EMIEUX, "Réseau de cellules intégré : Etude d'architecture pour des applications de CAO de VLSI", thèse en microélectronique, I.N.P. Grenoble, septembre 1988.

[Dal86] WILLIAM J. DALLY, CHARLES L. SEITZ, "The Torus Routing Chip", Distributed Computing, Vol 1 No 4 October 1986, pp 187-196.

[Dal86c] WILLIAM J. DALLY, "A VLSI Architecture for concurrent data structures",Thèse Mars 1986, California Institut of Technology.

[Dje92] DJEMAL RIDHA, "Implémentation d'un routage de type *wormhole* entre les processeurs élementaires d'un réseau cllulaire", Rapport DEA microélectronique à l'Université Joseph Fourier, Juin 1992

[Fau90] B. FAURE, G. MAZARE, "Neural networks : a cellular architecture", proceedings of the International Conference "Neural Networks, Biological Computers or Electronic Brains", AFCET-Entretiens de Lyon, Mars 1990.

[Fau91] B. FAURE, S. M. KARABERNOU, G. MAZARE, E. PAYAN, P. RUBINI, "Une architecture parallèle intégrée", Annales des Télécommunications, 46, n° 1-2, 1991

[Fay89] FAYEZ El GUIBALY,"Design Analysis of Arbitration Protocols", IEEE Transactions on Computers Vol.38 No2 February 1989.

[Fla87] CHARLES M. FLAIG, "VLSI Mesh Routing Systems", May 1987 Technical Report 5241:TR:87, Computer Science Department, California Institut of Technology.

[Fly72] M.J. FLYNN, "Some computer organizations and their effectiveness", IEEE Transaction on Computers, 21-9,Sept 1972.

[Ger91] C. GERMAIN-RENAUD, J. P. SANSONNET "Les ordinateurs massivement parallèles", Edition Armand Colin, 1991

[Had92] HADJOUJA ABDELKADER, "Modélisation et simulation d'un réseau cellulaire", Rapport DEA microélectronique à l'Université Joseph Fourier, Juin 1992

[Jac91] O. JACQUIOT,"Mise en oeuvre d'une interface et étude d'algorithmes de simulations distribuées pour un eéseau cellulaire", Rapport DEA informatique, ENSIMAG, Juin 1991.

[Joh92] JOHN L. HENNESSY, DAVID A. PATTERSON,"Architecture des ordinateurs, une approche quantitative", Edition McGraw-Hill 1992

[Kar89] S. M. KARABERNOU,"Etude et Réalisation d'un Mécanisme d'Acheminement de Messages dans un Réseau Cellulaire", Rapport DEA microélectronique à l'Université Joseph Fourier, Juin 1989.

[Kar90a] S. M. KARABERNOU, G. MAZARÉ, E. PAYAN, P. RUBINI , "A network with small general units for fine grain parallelism", Proceedings of the international Workshop on Algorithms and parallel

Architecures, Pont à Mousson, Juin 1990, pp 197-200.

[Kar90b] S. M. KARABERNOU, E. PAYAN, P. RUBINI, "Une machine générale massivement parallèle pour le parallèlisme à grain fin", Actes du deuxième Symposium Architecture Nouvelle de Machines, Toulouse, Septembre 1990, pp 17-38.

[Kar90c] S. M. KARABERNOU, E. PAYAN, P. RUBINI ,"Une Architecture Massivement parallèle programmable : La Machine RAP", 3 ième rencontres sur les Algorithms et Architectures Massivement Parallèles, Luminy, Octobre 1990.

[Kar91a] S. M. KARABERNOU, G. MAZARE, E. PAYAN, P. RUBINI, "VLSI Design of a Massivelly Parallel Processor", Proceeding. of the 34th Midwest Symposium on Circuits and Systems, Monterey, California May 1991.

[Kar91b] S. M. KARABERNOU, C. AKTOUF, G. MAZARÉ, P. RUBINI, " VLSI design and testing of a massivelly parallel precessor ", Proc. of the 1991 International Conference on Microelectronics, Décembre 1991, le Caire, Egypte.

[Kra82] R. H. KRAMBECK, C. M. LEE and H.-F. S. LAW, "High-Speed Compact Circuits with CMOS", IEEE Journ. of Solid-State Circuits, Vol. SC-17, No 3, June 1982, PP. 614-619.

[Lat89] D. LATTARD, G. MAZARE, "Image reconstruction using an original asynchronous cellular array", proceedings de IEEE ISCAS89, may 1989, pp. 13-16.

[Mul89] J. M. MULLER, "Arithmétique des Ordinateurs, Opérateurs et Fonctions Elémentaires", 1989 Edition MASSON.

[Nei85] NEIL H. E. WEST, KARMAN ESHRAGHIAN, "Principles of VLSI Design, A Systems Perspective", 1985 Addisson Wesley.

[Nga87] JOHN Y. NGAI and CHARLES L. SEITZ, "A Framework for adaptive routing", 16 jully 1987 Technical Report 5246:TR:87, Computer Science department, California Institut of Technology.

[Obj88] P. OBJOIS, "Réseau de cellule intégré : mécanisme de communication inter-cellulaireet application à la simulation logique", thèse en microélectronique, I.N.P. Grenoble, septembre 88.

[Obr82] MONIKA OBREBSKA, "Etude Comparative de Différentes Parties de Contrôles des Microprocesseurs", Thèse en microélectronique I.N.P.Grenoble Juin 1982.

[Pay91] ERIC PAYAN, "Etude d'une architecture cellulaire programmable : Définition fonctionnelle et méthodologie de programmation", thèse en microélectronique, I.N.P. Grenoble juin 1991

[Per80] T. PEREZ SEGOVIA, "Optimization en surface des PLAs", Rapport DEA, ENSIMAG, Juin 1980.

[Rub92] P. RUBINI, "Définition fonctionnelle : évaluation et programmation d'une architecture massivement parallèle", Thèse Doctorat INPG, Juin 1992.

[San91] J. P. SANSONNET, "Concepts d'Architectures Avancées", Cours DEA Informatique Paris XI. 1990-1991.

[Sei84] SEITZ CHARLE L., "Concurrent VLSI Architectures", IEEE Transaction on Computers vol C-33 no 12 pp 1247-1265, December 1984.

[Shu87] A.S. SHUBAT, J.A. PERTORIUS, and C.A.T. SALAMA, "Expandable Arithmetic Block Macrocell", Integraion Workshop, The VLSI Journal, May 1987, PP. 47-71.

[Sol90] ES2 SOLO 2030 FAMILY, "Solo 2030 User Guide", Edition 2, August 1990.

[Sou89] MOHAMMAD SOUEIDAN, "Conception d'un Microprocesseur Reconfigurable", Thèse INPG Avril 1989.

[Ted92] TED G. LEWIS, HESHAM EL-REWINI, "Introduction to parallel computing", Editions Prentice-Hall International, 1992

[Woe89] WOEI LIN, TSANG-LING SHEN CHITA R. DAS. TSE-YOU FUNG, CHUAN-LIN WU, "A Conflit-free Routing Scheme on Multistage Interconnection Networks", IEEE Transactions on Computers Vol 38, No 8, August 1989.

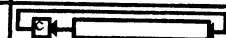
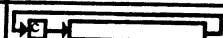
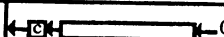
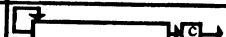
ANNEXES

ANNEXE 1: JEU D'INSTRUCTIONS ET TABLE DES CODES

A.1.1 CONDITIONS DES INSTRUCTIONS DE BRANCHEMENT

MNEMONICS		CONDITIONS	CODE	BITS D'ÉTATS
1	**BRA**	Branch	$05	-
2	**BHI**	Branch if higher	$15	$\overline{C} + \overline{Z}$
3	**BLS**	Branch if lower or same	$25	$C + Z$
4	**BCC**	Branch if carry clear	$35	$\overline{C}$
5	**BCS**	Branch if carry set	$45	C
6	**BNE**	Branch if not equal to zero	$55	$\overline{Z}$
7	**BEQ**	Branch if equal to zero	$65	Z
8	**BVC**	Branch if overflow clear	$75	$\overline{V}$
9	**BVS**	Branch if overflow set	$85	V
10	**BPL**	Branch if plus	$95	$\overline{N}$
11	**BMI**	Branch if minus	$A5	N
12	**BGE**	Branch if greater or equal	$C5	$N . V + \overline{N} . \overline{V}$
13	**BLT**	Branch if less than	$D5	$N . \overline{V} + \overline{N} . V$
14	**BGT**	Branch if greater than	$E5	$N . V . \overline{Z} + \overline{N} . \overline{V} . \overline{Z}$
15	**BLE**	Branch if less or equal	$F5	$Z + N . \overline{V} + \overline{N} . V$

A.1.2 JEU D'INSTRUCTIONS

	INSTRUCTIONS / ADDRESSING MODES	INHERENT ~ # H	IMMEDIAT ~ # H	ABSOLUTE ~ # H	ABSOLUTE QUICK ~ # H	INDIRECT ~ # H	INDIRECT WITH POST INCREMENT ~ # H	DESCRIPTION	STATUS
1	LDA		2 2 AA	3 2 9A	2 1 4X	2 1 BA	2 1 EA	M ⟶ A	z,n
2	LDI		2 2 AB	3 2 9B		2 1 BB		M ⟶ I	
3	STA			3 2 9C	2 1 0X	2 1 BC	2 1 EC	A ⟶ M	
4	ADD		3 2 A0	4 2 90		3 1 B0	3 1 E0	A+M ⟶ A	z,n,v,c
5	SUB		3 2 A1	4 2 91		3 1 B1	3 1 E1	A-M ⟶ A	z,n,v,c
6	ADC		3 2 A2	4 2 92		3 1 B2	3 1 E2	A+C+M ⟶ A	z,n,v,c
7	SBC		3 2 A3	4 2 93		3 1 B3	3 1 E3	A+C-M ⟶ A	z,n,v,c
8	XOR		3 2 A4	4 2 94		3 1 B4	3 1 E4	A⊕M ⟶ A	z,n
9	OR		3 2 A5	4 2 95		3 1 B5	3 1 E5	A+M ⟶ A	z,n
10	AND		3 2 A6	4 2 96		3 1 B6	3 1 E6	A.M ⟶ A	z,n
11	CMP		3 2 A8	4 2 98		3 1 B8	3 1 E8	A-M	z,n
12	INC	2 1 64		5 2 84		4 1 C4	4 1 F4	(A,M)+1 ⟶ (A,M)	z,n,c
13	DEC	2 1 65		5 2 85		4 1 C5	4 1 F5	(A,M)-1 ⟶ (A,M)	z,n,c
14	ROL	2 1 60		5 2 80		4 1 C0	4 1 F0		z,n,c
15	ROR	2 1 62		5 2 82		4 1 C2	4 1 F2		z,n,c
16	ASL	2 1 61		5 2 81		4 1 C1	4 1 F1		z,n,c
17	ASR	2 1 63		5 2 83		4 1 C3	4 1 F3		z,n,c
18	CLR	2 1 67		5 2 87		4 1 C7	4 1 F7	0 ⟶ A,M	z
19	TST	2 1 6I		4 2 88		3 1 C8	3 1 F8	A,M - 00	z,n,c
20	TAI	2 1 6B						A ⟶ I	
21	TIA	2 1 6D						I ⟶ A	z,n
22	TPCA	2 1 6E						PC ⟶ A	z,n
23	TAPC	2 1 6C						A ⟶ PC	
24	BCC			2 2 7X				IF C=True Then Adr ⟶ PC Else Next Instruction	
25	TRY			4 2 9F	3 1 2X	3 1 CF	3 1 EF	If F=0 then Acc ⟶ PC Else next instruction	
26	PUT			3 2 9D	2 1 1X	2 1 CD	2 1 ED	A ⟶ M , 1 ⟶ Flag	
27	GET			5+ 2 9E	3+ 1 3X	3+ 1 CE	3+ 1 EE	M ⟶ A , 0 ⟶ Flag	z,n
28	SEND			5 2 99		4 1 B9	4 1 E9	M1,M2,M3 ⟶ tag,val,adr	
29	CLC	2 1 69						0 ⟶ C	
30	SEC	2 1 6A						1 ⟶ C	
31	MUL			5+ 1 89				A*MQ ⟶ A,MQ	
32	LDAW		3 3 A9	4 2 8A		3 1 CA	3 1 FA	HB ⟶ MQ , LB ⟶ Acc	z,n
33	STAW			4 2 8C		3 1 CC	3 1 FC	MQ ⟶ M , Acc ⟶ M+1	
34	ADCW		5 3 AE	6 2 8E		5 1 BE	5 1 FE	Acc(LB) + M ⟶ Acc MQ(HB) + (M+1) ⟶ MQ	z,n,v,c
35	SBCW		5 3 AF	6 2 8F		5 1 BF	5 1 FF	Acc(LB) - M ⟶ Acc MQ(HB) - (M+1) ⟶ MQ	z,n,v,c
	TOTAL	15	13	28	5	27	26	114	

A.1.3 TABLES DES CODES DES INSTRUCTIONS

0000	STA $00	PUT $F0	TRY $F0	GET $F0	LDA $00		ROL A	BRA	ROL abs	ADD abs	ADD #	ADD (I)	ROL (I)		ADD (I)+	ROL (I)+
0001	STA $01	PUT $F1	TRY $F1	GET $F1	LDA $01		ASL A	BHI	ASL abs	SUB abs	SUB #	SUB (I)	ASL (I)		SUB (I)+	ASL (I)+
0010	STA $02	PUT $F2	TRY $F2	GET $F2	LDA $02		ROR A	BLS	ROR abs	ADC abs	ADC #	ADC (I)	ROR (I)		ADC (I)+	ROR (I)+
0011	STA $03	PUT $F3	TRY $F3	GET $F3	LDA $03		ASR A	BCC	ASR abs	SBC abs	SBC #	SBC (I)	ASR (I)		SBC (I)+	ASR (I)+
0100	STA $04	PUT $F4	TRY $F4	GET $F4	LDA $04		INC A	BCS	INC abs	XOR abs	XOR #	XOR (I)	INC (I)		XOR (I)+	INC (I)+
0101	STA $05	PUT $F5	TRY $F5	GET $F5	LDA $05		DEC A	BNE	DEC abs	OR abs	OR #	OR (I)	DEC (I)		OR (I)+	DEC (I)+
0110	STA $06	PUT $F6	TRY $F6	GET $F6	LDA $06		NOT A	BEQ	NOT abs	AND abs	AND #	AND (I)	NOT (I)		AND (I)+	NOT (I)+
0111	STA $07	PUT $F7	TRY $F7	GET $F7	LDA $07		CLR A	BVC	CLR abs				CLR (I)			CLR (I)+
1000	STA $08	PUT $F8	TRY $F8	GET $F8	LDA $08		TST A	BVS	TST abs	CMP abs	CMP #	CMP (I)	TST (I)		CMP (I)+	TST (I)+
1001	STA $09	PUT $F9	TRY $F9	GET $F9	LDA $09		CLC	BPL	MUL abs	SEND abs	LDAW #	SEND(I)			SEND(I)+	
1010	STA $0A	PUT $FA	TRY $FA	GET $FA	LDA $0A		SEC	BMI	LDAW abs	LDA abs	LDA #	LDA (I)	LDAW(I)		LDA (I)+	LDAW(I)+
1011	STA $0B	PUT $FB	TRY $FB	GET $FB	LDA $0B		TAI	BGE		LDI abs	LDI #	LDI (I)				
1100	STA $0C	PUT $FC	TRY $FC	GET $FC	LDA $0C		TAPC	BLT	STAW abs	STA abs		STA (I)	STAW(I)		STA (I)+	STAW(I)+
1101	STA $0D	PUT $FD	TRY $FD	GET $FD	LDA $0D		TIA	BGT		PUT abs			PUT (I)		PUT (I)+	
1110	STA $0E	PUT $FE	TRY $FE	GET $FE	LDA $0E		TPCA	BLE	ADCW abs	GET abs	ADCW #	ADCW(I)	GET (I)		GET (I)+	ADCW(I)+
1111	STA $0F	PUT $FF	TRY $FF	GET $FF	LDA $0F				SBCW abs	TRY abs	SBCW #	SBCW(I)	TRY (I)		TRY (I)+	SBCW(I)+
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

ANNEXE 2 : DESCRIPTION D'EXECUTION DES INSTRUCTIONS

fi : Fetch instruction
fo : Fetch opérande
fa : Fetch adresse
exe : Execution
wm : Ecriture mémoire
ad : Adition

FETCH

fi : PC ⟶ BUSA, M[BusA] ⟶ BusD, BusD ⟶ IR and IR2 //PC +1 ⟶ PC*
* sauf l'instruction GET

ABSOLU COURT

*** LDA**
fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc

*** STA**
wm : IR2 ⟶ BusA // Acc ⟶ BusD , BusD ⟶ M [BusA]

*** GET**
fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc
exe : SI FLAG(M[BusA]) =0 ALORS **GET**
wm : SINON FLAG(M[BusA]) =1 // Ac ⟶ BusD, BusD ⟶ M [BusA],
PC+1 ⟶ PC

*** PUT**
wm : Acc ⟶ BusD // IR2 ⟶ BusA //1 ⟶ FLAG , BusD ⟶ M[BusA]

*** TRY**
fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD
exe : SI FLAG(M[BusA]) =0 ALORS Acc ⟶ BusD , BusD ⟶ PC
SINON Fetch instruction suivante

IMPLICITE

*** BCC**

exe : SI C=VRAI ALORS PC ⟶ BusA, M [PC] ⟶ BusD, BusD ⟶ PC
SINON PC + 1 ⟶ PC

*** CLC *, SEC *, CLRA , ROLA , RORA , ASLA , ASRA**

exe : COMMANDES // [Set Carry//Reset Carry]* ,Acc ⟶ Acc

*** TAI , TAPC**

exe : Acc ⟶ BusD , BusD ⟶ I / PC

*** TIA , TPCA**

exe : I / PC ⟶ BusA , BusA ⟶ Acc

*** NOTA**

exe : **OP** (Acc) , ALU ⟶ Acc

*** TSTA**

exe : Reset carry // Reset LATCH // Acc **OP** LATCH , ALU ⟶ Acc

*** INCA , DECA**

exe : Reset LATCH // Set CARRY // Acc **OP** LATCH , ALU ⟶ Acc

IMMEDIAT

*** XOR , OR , AND , ADD* , ADC , SUB* , SBC**

fo : PC ⟶ BusA//Reset carry*, M[BusA] ⟶BusD, BusD⟶ LATCH
//PC + 1⟶ PC

exe : Acc **OP** LATCH , ALU ⟶ Acc

*** CMP**

fo : PC ⟶ BusA//Reset carry, M [BusA] ⟶BusD, BusD ⟶ LATCH
//PC + 1⟶ PC

exe : Acc **OP** LATCH

*** LDA , LDI**

fo : PC ⟶ BusA, M[BusA] ⟶BusD , BusD ⟶ I, Acc // PC + 1 ⟶ PC

*** LDAW**

fo : PC⟶ BusA, M[BusA] ⟶BusD, BusD ⟶ MQ // PC + 1⟶ PC
//Set Cptr(1)

fo : PC ⟶ BusA, M[BusA] ⟶BusD, BusD⟶ Acc // PC + 1 ⟶ PC
//Shift Cptr

*** ADCW , SBCW**

fo : PC ⟶ BusA, M[BusA] ⟶BusD, BusD ⟶ LATCH //PC + 1⟶ PC
//Set Cptr(1)

exe : MQ **OP** LATCH , ALU ⟶ MQ

fo : PC ⟶ BusA, M [BusA] ⟶BusD, BusD ⟶ LATCH //PC + 1 ⟶ PC
//Shift Cptr

exe : Acc **OP** LATCH , ALU ⟶ Acc

ABSOLUT

*** ROL , ROR , ASL , ASR, NOT, CLR* : (mq)**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ MQ

exe : Commandes // Reset MQ*, MQ ⟶ MQ

wm : MQ ⟶ BusD // IR2 ⟶ BusA , BusD ⟶ M [BusA] , Δ

*** INC , DEC**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD ,PC + 1 ⟶ PC // BusD ⟶ IR2

fo :IR2⟶ BusA//Set carry//ResetLATCH, M[BusA]⟶BusD,BusD⟶ MQ

exe : MQ **OP** LATCH , ALU ⟶MQ

wm : IR2 ⟶ BusA // MQ ⟶ BusD , BusD ⟶ M [BusA]

*** CMP**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

fo : IR2 ⟶ BusA //Reset carry , M [BusA] ⟶BusD , BusD ⟶ LATCH

exe : Acc **OP** LATCH

*** TST**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

fo :IR2⟶BusA//Reset carry//Reset LATCH, M[BusA]⟶BusD,BusD⟶ MQ

exe : MQ **OP** LATCH

*** LDA, LDI**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

fo : IR2 ⟶ BusA , M [BusA] ⟶BusD , BusD ⟶ A, I

*** STA**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

wm : IR2 ⟶ BusA // Acc ⟶ BusD , BusD ⟶ M [BusA]

*** ADD* , SUB* , ADC , SBC , XOR , OR , AND**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC// BusD ⟶ IR2

fo : IR2 ⟶ BusA // Reset carry*, M [BusA]⟶BusD, BusD ⟶ LATCH

exe : Acc **OP** LATCH , ALU ⟶Acc

*** TRY**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD

exe : SI FLAG(M[BusA]) =0 ALORS Acc ⟶ BusD , BusD ⟶ PC
SINON Fetch Instruction Suivante

*** PUT**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

wm : Acc ⟶ BusD // IR2 ⟶ BusA //1 ⟶ FLAG, BusD ⟶ M[BusA]

*** GET**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2

fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc

exe : SI FLAG(M[BusA]) =0 ALORS **GET**

wm : SINON FLAG(M[BusA]) = 0 //IR2 ⟶ BusA//Acc ⟶ BusD, BusD ⟶ M [BusA]

*** SEND**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2
SI Vide = faux alors attendre
SINON :

fo : IR2⟶BusA//Set Cptr(11),M [BusA]⟶ BusD,IR2+1 ⟶IR2
// BusD⟶ OUTREG

fo : IR2⟶BusA//Shift Cptr, M [BusA]⟶ BusD , IR2+1 ⟶IR2
// BusD⟶ OUTREG

fo : IR2⟶BusA//Shift Cptr, M [BusA]⟶ BusD , IR2+1 ⟶IR2
// BusD⟶ OUTREG

*** MUL**

fa : PC ⟶ BusA, M [BusA] ⟶ BusD , PC+1 ⟶ PC// BusD ⟶ IR2
fo : IR2 ⟶ BusA//ResetAcc, M[BusA] ⟶ BusD, BusD ⟶ MQ
// IR2+1 ⟶ I R2
//Set Cptr(01)
fo : IR2 ⟶ BusA//Reset carry, M[BusA] ⟶ BusD, BusD ⟶ LATCH
//IR2+1 ⟶ IR2
//Set Cptr(8)
ad : SI MQ0 = 1 ALORS MQ **OP** LATCH , ALU ⟶ Acc
exe : SINON SHIFT Acc , MQ and Cptr

*** LDAW**

fa : PC ⟶ BusA , M[BusA] ⟶ BusD ,PC + 1 ⟶ PC // BusD ⟶ IR2
fo : IR2 ⟶ BusA M[BusA] ⟶ BusD, IR2 + 1 ⟶ IR2//BusD ⟶ MQ
//Set Cptr(01)
fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc// Shift Cptr

*** STAW**

fa : PC ⟶ BusA , M [BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2
wm : IR2 ⟶ BusA // MQ ⟶ BusD BusD ⟶ M[BusA] , IR2+1 ⟶ IR2
// Set Cptr(01)
wm : IR2 ⟶ BusA //Acc ⟶ BusD, BusD ⟶ M[BusA] // Shift Cptr

*** ADCW , SBCW**

fa : PC ⟶ BusA , M[BusA] ⟶ BusD , PC + 1 ⟶ PC // BusD ⟶ IR2
fo : IR2 ⟶ BusA,M[BusA] ⟶ BusD, BusD ⟶ LATCH//IR2+1 ⟶ IR2
//Set Cptr(01)
exe : MQ **OP** LATCH, ALU ⟶ MQ
fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ LATCH// Shift Cptr
exe : Acc **OP** LATCH, ALU ⟶ Acc

INDIRECT

*** ROL , ROR , ASL , ASR, CLR*, NOT**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ MQ
exe : **OP** MQ / Reset MQ*
wm : I ⟶ BusA, MQ ⟶ BusD, BusD ⟶ M[BusA]

*** INC , DEC**

fo :I ⟶ BusA// Reset LATCH// Set carry, M[BusA] ⟶ BusD, BusD⟶MQ
exe : MQ **OP** LATCH , ALU ⟶ MQ
wm : I ⟶ BusA // MQ ⟶ BusD, BusD⟶ M[BusA]

*** CMP**

fo : I ⟶ BusA // Reset carry, M [BusA] ⟶ BusD , BusD ⟶ LATCH
exe : Acc **OP** LATCH

*** TST**

fo :I ⟶BusA// Reset carry// Reset LATCH, M[BusA]⟶BusD, BusD⟶ MQ
exe : MQ **OP** LATCH

*** LDA , LDI**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ I / A

*** STA**

wm : Acc ⟶ BusD // I ⟶ BusA, BusD⟶ M[BusA]

*** ADD* , ADC , SUB* , SBC , XOR , OR , AND**

fo : I ⟶ BusA // Reset carry* , M [BusA] ⟶ BusD , BusD ⟶ LATCH
exe : Acc **OP** LATCH, ALU ⟶ Acc

*** TRY**

fo : I ⟶ BusA , M [BusA] ⟶ BusD
exe : SI FLAG(M[BusA]) =0 ALORS Acc ⟶ BusD , BusD ⟶ PC
SINON Fetch instruction Suivante

*** PUT**

wm : I ⟶ BusA //1 ⟶ FLAG// Acc ⟶ BusD , BusD ⟶ M[BusA]

*** GET**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc
exe : SI FLAG(M[BusA]) =0 ALORS **GET**
wm : SINON FLAG(M[BusA]) =0 // I⟶ BusA// Acc ⟶ BusD,
BusD⟶ M[BusA]

*** SEND**

fo : I ⟶ BusA, M [BusA] ⟶ BusD, I + 1 ⟶ IR2
//BusD ⟶ OUTREG//Set Cptr(11)

fo : IR2 ⟶ BusA, M [BusA] ⟶ BusD,IR2+1 ⟶ IR2
//BusD ⟶ OUTREG//Shift Cptr

fo : IR2 ⟶ BusA, M [BusA] ⟶BusD,IR2+1 ⟶ IR2
//BusD ⟶ OUTREG//Shift Cptr

*** LDAW**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , I + 1 ⟶ IR2 // BusD ⟶ MQ
// Set Cptr(01)

fo : IR2 ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc//Shift Cptr

*** STAW**

wm : I⟶BusA// MQ ⟶ BusD, BusD⟶ M[BusA] // I+1 ⟶IR2
// Set Cptr(01)

wm : IR2 ⟶BusA// Acc ⟶ BusD, BusD⟶ M[BusA] , Shift Cptr

*** ADCW , SBCW**

fo : I ⟶ BusA , M [BusA] ⟶BusD , BusD ⟶ LATCH //I+1 ⟶IR2
// Set Cptr (01)

exe : MQ **OP** LATCH, ALU ⟶ MQ

fo : IR2 ⟶ BusA , M [BusA] ⟶BusD , BusD ⟶ LATCH//Shift Cptr

exe : Acc **OP** LATCH , ALU ⟶ Acc

INDIRECT POST INCREMENTE

*** ROL , ROR , ASL , ASR, CLR*, NOT**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ MQ

exe : **OP** MQ / Reset MQ*

wm : MQ ⟶ BusD // I ⟶ BusA , BusD⟶ M[BusA] // I+1 ⟶ I

*** INC , DEC**

fo : I ⟶ BusA//Set carry//Reset LATCH, M [BusA]⟶ BusD, BusD⟶ MQ

exe : MQ **OP** LATCH , ALU ⟶ MQ

wm : I ⟶ BusA // MQ ⟶ BusD, BusD⟶ M[BusA] , I+1 ⟶ I

*** CMP**

fo : I ⟶ BusA // Reset carry , M [BusA] ⟶ BusD, BusD ⟶ LATCH
// I+1 ⟶ I

exe : Acc **OP** LATCH

*** TST**

fo : I ⟶ BusA // Reset LATCH, M [BusA] ⟶ BusD , BusD ⟶ MQ
// I+1 ⟶ I

exe : MQ **OP** LATCH

*** LDA**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc // I+1 ⟶ I

*** STA**

wm : Acc ⟶ BusD// I ⟶ BusA ⟶ BusD, BusD⟶ M[BusA], I+1 ⟶ I

*** ADD* , ADC , SUB*, SBC , XOR , OR , AND**

fo : I ⟶ BusA // Reset carry*, M [BusA] ⟶ BusD, BusD ⟶ LATCH
// I+1 ⟶ I

exe : Acc **OP** LATCH, ALU ⟶ Acc

*** TRY**

fo : I ⟶ BusA , M [BusA] ⟶ BusD // I+1 ⟶ I

exe : SI FLAG(M[BusA]) =0 ALORS Acc ⟶ BusD , BusD ⟶ PC
SINON Fetch Instruction Suivante

*** PUT**

wm : I ⟶ BusA //Acc ⟶ BusD //1 ⟶FLAG, BusD ⟶ M[BusA]
// I+1 ⟶ I

*** GET**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , BusD ⟶ Acc

exe : SI FLAG(M[BusA]) =0 ALORS **GET**
SINON :

wm : I ⟶ BusA//0 ⟶ FLAG // Acc ⟶ BusD, BusD ⟶ M[BusA]
//I+1 ⟶ I

*** SEND**

fo : I ⟶ BusA, M [BusA] ⟶ BusD, I + 1 ⟶ I // BusD ⟶ OUTREG
//Set Cptr(11)

fo : I ⟶ BusA, M [BusA] ⟶ BusD, I + 1 ⟶ I // BusD ⟶ OUTREG
//Shift Cptr

fo : I ⟶ BusA , M [BusA] ⟶ BusD, I + 1 ⟶ I // BusD ⟶ OUTREG
//Shift Cptr

*** LDAW**

fo : I ⟶ BusA , M [BusA] ⟶ BusD , I + 1 ⟶ I // BusD ⟶ MQ
// Set Cptr(01)

fo : I ⟶ BusA , M [BusA] ⟶ BusD , I+1 ⟶ I // BusD ⟶ Acc
// Shift Cptr

*** STAW**

wm :I⟶BusA// MQ⟶ BusD, BusD⟶ M[BusA], I+1 ⟶I // Set Cptr(01)

wm : I ⟶BusA// Acc⟶ BusD, BusD⟶ M[BusA], I+1 ⟶ I // Shift Cptr

*** ADCW , SBCW**

fo : I ⟶ BusA , M [BusA] ⟶BusD , BusD ⟶ LATCH //I+1 ⟶I
// Set Cptr(01)

exe : MQ **OP** LATCH, ALU ⟶ MQ

fo : I ⟶ BusA , M [BusA] ⟶BusD , BusD ⟶ LATCH // I+1 ⟶ I
// Shift Cptr

exe : Acc **OP** LATCH, ALU ⟶ Acc

ANNEXE 3 : LES DIFFÉRENTS TYPES D'UALS

A.1.UAL A BASE D'ADDITIONNEURS SÉQUENTIELS

A.1.1 UAL TYPE RIPPLE CARRY

L'unité arithmétique et logique du type "Ripple carry" est une UAL à chemin de retenue simple utilisant comme additionneur le "ripple carry adder". L'UAL est définie comme suit :

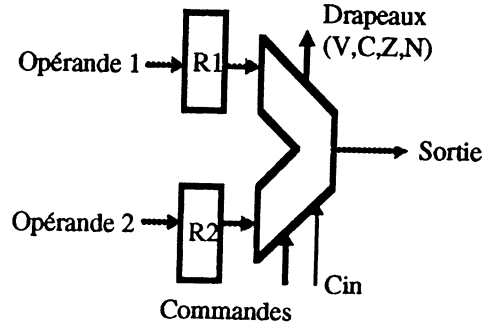


Figure A.1 : Schéma général de l'UAL

Une tranche d'UAL est constituée, d'une part, d'un bloc combinatoire sélectionnant l'opération désirée en fonction des commandes, et d'autre part, d'un opérateur effectuant l'opération. La figure A.2 montre le schéma général d'un bit d'UAL.

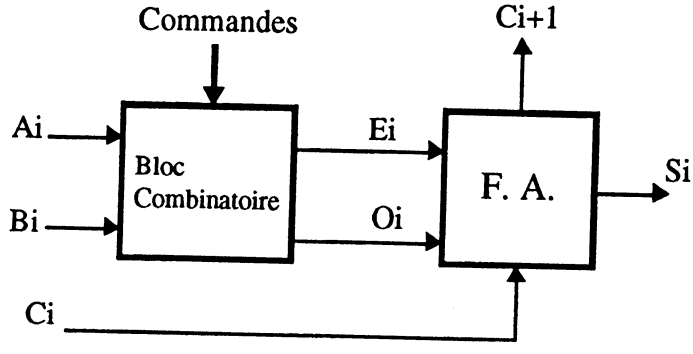


Figure : A.2: Schéma général d'1 bit de l'UAL

Tout le long de cet aperçu sur les différents types d'UAL, il sera pris comme modèle de reference, l'UAL utilisant un additionneur "Ripple Carry" afin de pouvoir comparer en surface sur silicium et en performances avec les autres type d'UAL.

A.1.2 UAL TYPE DOMINO

Une porte logique de type CMOS domino est constituée d'une porte dynamique suivie d'un buffer CMOS (figure A.3). Un transistor de maintien est fortement recommandé en parallèle avec le transistor de précharge P pour les opérations statiques et dans les systèmes à faibles fréquences.

Ce type de circuit regroupe les avantages des circuits dynamiques, par rapport aux circuits NMOS, (faible consommation, surface comparable avec la structure NMOS et rapidité) et celles des circuits CMOS (stabilité et facilité d'utilisation des circuits statiques.

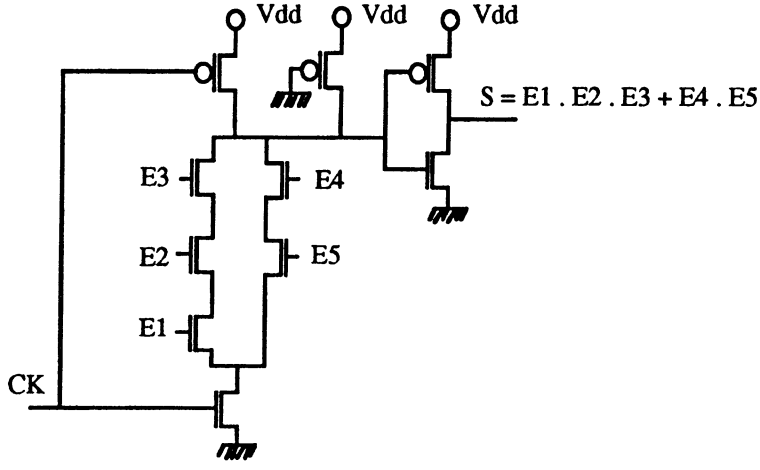


Figure A.3: Structure CMOS Domino

Les circuits en structure domino CMOS ont des surfaces comparables avec les circuits en technologie NMOS et pseudo-NMOS, mais avec une surface de 1,5 à 2 fois plus grande [KRA 82]. Cependant, le gain de vitesse d'une UAL utilisant la technique domino par rapport à une UAL de type CMOS est de 30% [SHU 87] si on prend en terme de surface et performance l'UAL type Ripple Carry comme reference.

A.1.3 UAL TYPE MANCHESTER

Il est évident que dans les UAL, la propagation de la retenue qui fait l'objet des préoccupations des concepteurs. L'UAL de type MANCHESTER utilise les principes de génération et propagation de la retenue. Le principe de génération est basé sur une constatation très simple: lors de l'addition de deux nombres $A = a_{n-1}\, a_{n-2} \ldots a_i \ldots a_1 a_0$ et $B = b_{n-1}\, b_{n-2} \ldots b_i \ldots b_1 b_0$, si $a_i = b_i$, il n'est pas nécessaire de connaître la retenue entrante c_i pour calculer la retenue sortante c_{i+1}. En effet :

si $a_i = b_i = 0$, alors $c_{i+1} = 0$,
si $a_i = b_i = 1$, alors $c_{i+1} = 1$.

Le principe de propagation complète le principe de génération. La retenue est propagée dans le cas où $a_i \neq b_i$, nous avons alors $c_{i+1} = c_i$. En partant de ces deux principes, on peut définir déjà deux variables P_i et G_i afin d'accélérer la retenue :

$P_i = a_i \oplus b_i$ (aptitude du $i^{ème}$ rang à propager la retenue)
et $G_i = a_i.b_i$ (aptitude du $i^{ème}$ rang à générer la retenue)

Une UAL du type MANCHESTER utilise bien évidement un additionneur du type MANCHESTER. Le schéma de principe de la chaîne de MANCHESTER est donné dans la figure A.4.

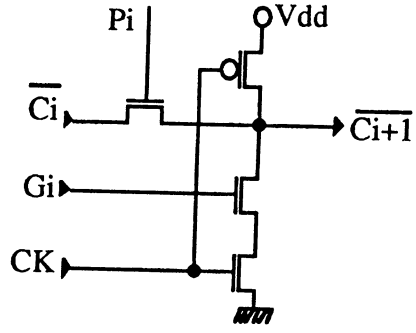


Figure A.4 : Chaîne de retenue Manchester

Lorsque CK passe à 0, la chaîne de retenue est préchargée à 1 et lorsque CK remonte à 1, la phase d'évaluation est commencée. Un schéma simplifié d'un additionneur du type MANCHESTER est donné en la figure suivante:

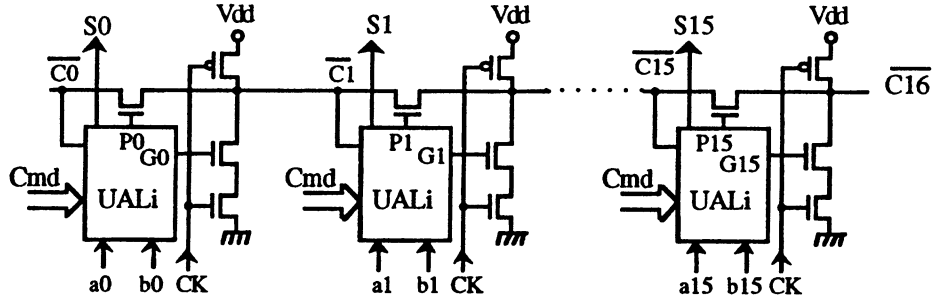


Figure A.5 : UAL du type Manchester

La décharge complète de la ligne de la retenue correspond à $P_i=1$ pour tout i avec $C_0= 1$. Ceci est montré dans la figure A.6.

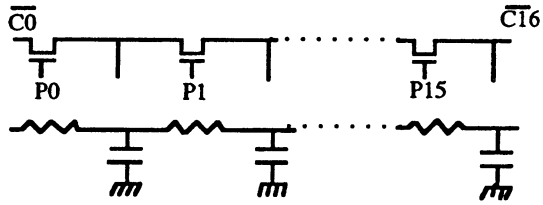


Figure A.6 :Circuit RC equivalent : décharge de la retenue

La vitesse de l'UAL dépendra principalement de la facilité à décharger la ligne de retenue. Dans le but de faciliter la décharge, deux technique peuvent être utilisées : la première consiste à calculer $\prod P_i$ pour 4 tranches. Lorsque $\prod P_i= 1$, la retenue entrante saute ces tranches.

La seconde technique consiste à utiliser un circuit de décharge inclus dans chaque tranche de l'UAL.

En comparaison avec l'UAL de référence, l'UAL MANCHESTER est trois fois plus rapide et offre une une diminution de surface de 10%. L'inconvénient de ce type est la nécessité d'introduire une horloge de précharge surtout qu'elle doit être distribuée pour tous les bits introduisant ainsi des problèmes de synchronisation.

A.2. UAL À BASE D'ADDITIONNEURS À RETENUE BONDISSANTE

A.2.1 UAL TYPE CARRY SÉLECT

Le principe de la méthode d'anticipation de la retenue, consiste à diviser les tranches d'un additionneur en groupes de 4 tranches. Dans le cas d'un additionneurs 16 bits, les 16 tranches de bits sont divisées en 4 groupes de 4 tranches. Chacun des groupes de 4 bits est dupliqué; l'un effectue l'opération avec une retenue entrante nulle, tandis que l'autre effectue l'opération avec une retenue entrante égale à 1. Les huit blocs travaillent en parallèl. L'utilisation des multiplexeurs, commandés par des blocs de selection, permettent de choisir les bon blocs et en conséquence le bon résultat de l'opération (figure A.7).

La commande de sélection du premier groupe est évidemment la retenue entrante de l'UAL (C0). La sélection du second groupe est effectuée par la retenue sortante exacte du premier bloc (C4), donnée par l'équation logique suivante :

$$C4 = C_4^0 + C0C_4^1$$

où $C^0{}_4$ est la retenue sortante du premier groupe recevant une retenue entrante égale à 0 et $C^1{}_4$ est la retenue sortante du premier groupe recevant une retenue égale à 1. La retenue exacte du 2ème groupe C8 sélectionne le résultat du calcul du 3ème groupe.

$$C8 = C_8^0 + C_4^0C_8^1 + C0C_4^1C_8^1$$

où C_8^0 est la retenue sortante du deuxième groupe recevant une retenue entrante égale à 1.

Le dernier groupe est sélectionné par :

$$C12 = C_{12}^0 + C_8^0C_{12}^1 + C_4^0C_8^1C_{12}^1 + C0C_8^1C_{12}^1$$

où C_{12}^0 est la retenue sortante du troisième groupe recevant une retenue entrante égale à 0 et C_{12}^1 est la retenue sortante du troisième groupe recevant une retenue entrante égale à 1.

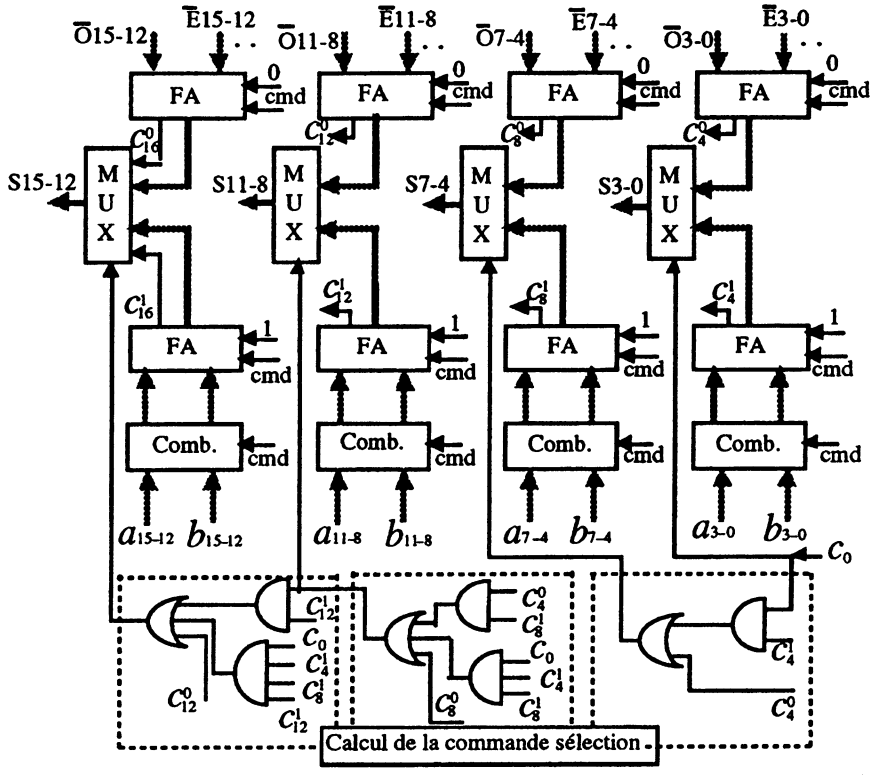


Figure A.7 : UAL de type "Carry Sélect Adder"

Cette méthode offre la possibilité de faire des UAL trois plus rapides que l'UAL ripple carry mais cet avantage est payé par une augmentation de surface de 60%.

A.2.2 UAL TYPE CARRY SKIP

Les UALs "Carry-Skip" utilise les principes de génération et de propagation de la retenue. L'idée générale de la méthode consiste à diviser l'additionneur en blocs. Chacun de ces blocs est constitué d'un additionneur séquentiel et d'un circuit spécial permettant de détecter s'il y a propagation de la retenue à travers le bloc ($\prod_i P_i = 1$) (figure A.8).

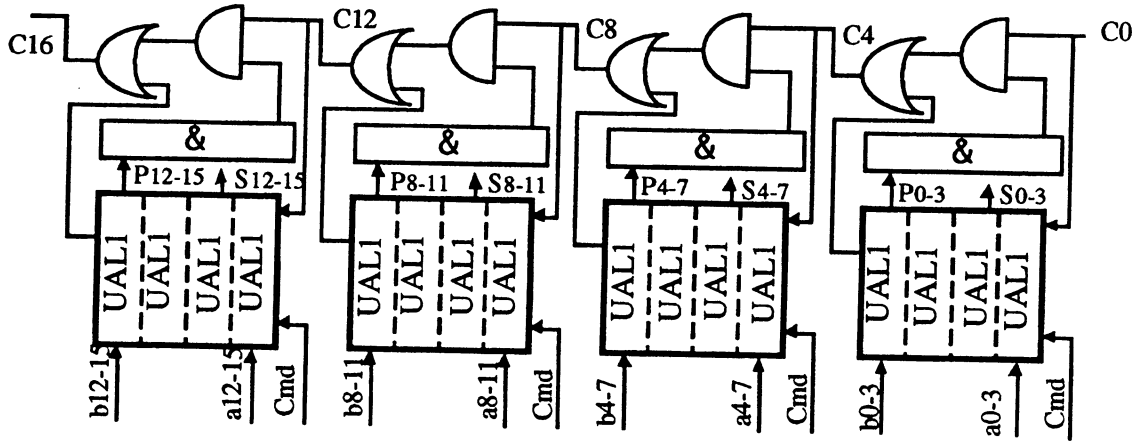


Figure A.8 : UAL du type "Carry Skip"

Si, dans l'un des blocs, on a $P_i = a_i \oplus b_i = 0$, alors une retenue est générée et toute la partie "aval" est ignorée. On note P, le nombre de tranches de l'UAL dans un bloc, m le nombre de blocs, T1 le temps de propagation dans une tranche de l'UAL et T2 le temps nécessaire pour que la retenue traverse un bloc. Dans le cas de blocs de tailles égales, on cherche à minimiser le plus long temps de propagation de la retenue. Le chemin le plus long qui détermine le temps le plus long pour une retenue qui commence au début du premier bloc, traverse les m-2 blocs suivants jusqu'au dernier bloc. Le temps total est donc

$T(m)=2*T_1*P(m)+(m-2)*T_2$, avec P=n / m; n étant le nombre de bits traités.

La première question posée dans cette méthode est la suivante : quel est le nombre optimal des blocs ?

Pour cela, il suffit de trouver la valeur de m qui minimise T en utilisant la première dérivée de T(m).

$$T_{min} = 2(\sqrt{2*n*T_1*T_2} - T_2)$$

Le temps de propagation de la retenue dans toute l'UAL est proportionnel à $\sqrt{n}$.

A titre de comparaison avec l'UAL initiale, ce type d'UAL est deux fois plus rapide pour une légère augmentation en surface de 16%.

A.3. UALs À BASE D'ADDITIONNEURS ANTICIPANTS LA RETENUE

Les UALs anticipant la retenue utilise le principe de calculer la retenue à l'avance plutôt que de la propager. Le calcul de la retenue est effectué selon l'algorithme suivant :

- l'étage i reçoit une retenue entrante égale à 1 si et seulement si :
  - l'étage i-1 a généré une retenue à 1 ($G_{i-1} = 1$) ou
  - l'étage i-1 a propagé une retenue à 1 générée par l'étage i-2 ($P_{i-1}=1$ et $G_{i-2}=1$ ou
  - .... ou
  - les étages i-1, i-2, ..., 1, 0 ont propagé une retenue entrante dans l'additionneur ($P_{i-1} = P_{i-2} = ... = P_1 = P_0 = 1$ et $C_0 = 1$.

On obtient donc :

$$C_i=G_{i-1} + G_{i-2}P_{i-1} + G_{i-3}P_{i-2}P_{i-1} + P_{i-1}\,P_{i-2} \ldots P_1P_0C_0$$

Il est évident que cette méthode devienne très lourde à utiliser dès que la longueur de l'additionneur dépasse 4 ou 5 étages. Il reste toujours possible de regrouper les bits de l'additionneur en petits groupes.

Dans le reste de cette étude, quelques types d'UAL utilisant ce principe sont présentés en commençant par l'UAL du type "Full Carry Look Ahead". Cette dernière sera étudiée en détails. Par contre, pour les autres types juste le principe de fonctionnement et la comparaison en performance et surface seront présentés.

A.3.1 UAL FULL CARRY LOOK AHEAD

Le principe de base de ce type d'UAL consiste à calculer parallèlement toutes les retenues à partir des entrées a_i et b_i où $0 \leq i \leq 15$. Ce qui implique un temps de calcul indépendant de la longueur des opérandes (A et B).

La partie additionneur est donc décomposée en 4 groupes de 4 bits et chacun de ces groupes est constitué de trois blocs (voir figure A.9).

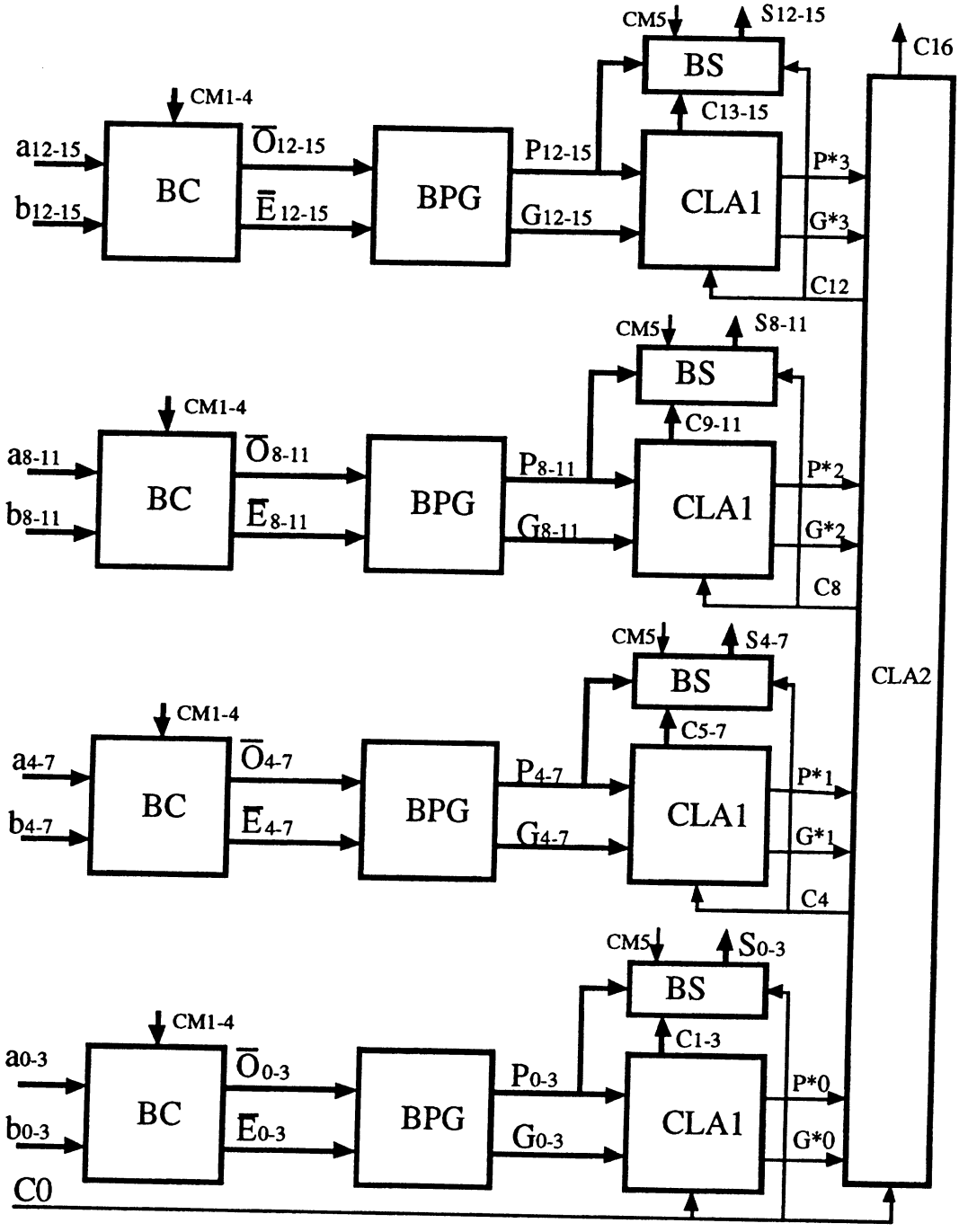


Figure A.9 : UAL du type "Full Carry Look Ahead Adder"

Le bloc BPG qui calcule les variables P_i et G_i où $0 \leq i \leq 3$ et

$$P_i = O_i \oplus E_i = a_i \oplus b_i \text{ (pour l'addition)}$$
$$G_i = O_i + E_i = a_i.b_i \text{ (pour la retenue)}$$

Le schéma logique implémentant ces équations est montré dans la figure suivante :

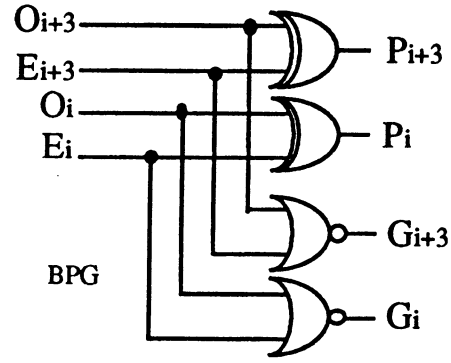


Figure A.10 : schéma logique du bloc BPG

Le bloc CLA1 qui calcule 3 retenues pour un groupe de 4 bits (a_i a_{i+1} a_{i+2} a_{i+3}, b_i b_{i+1} b_{i+2} b_{i+3}) et deux variables Pi* et Gi* alimentant le bloc CLA2 qui sont définies par :

$Pi^* = Pi\ Pi+1\ Pi+2\ Pi+3$

$Gi^* = Gi+3 + Gi+2\ Pi+3 + Gi+1\ Pi+2\ Pi+3 + Gi\ Pi+1\ Pi+2\ Pi + 3$

Le shéma logique du bloc CLA1 est montré en figure A.11.

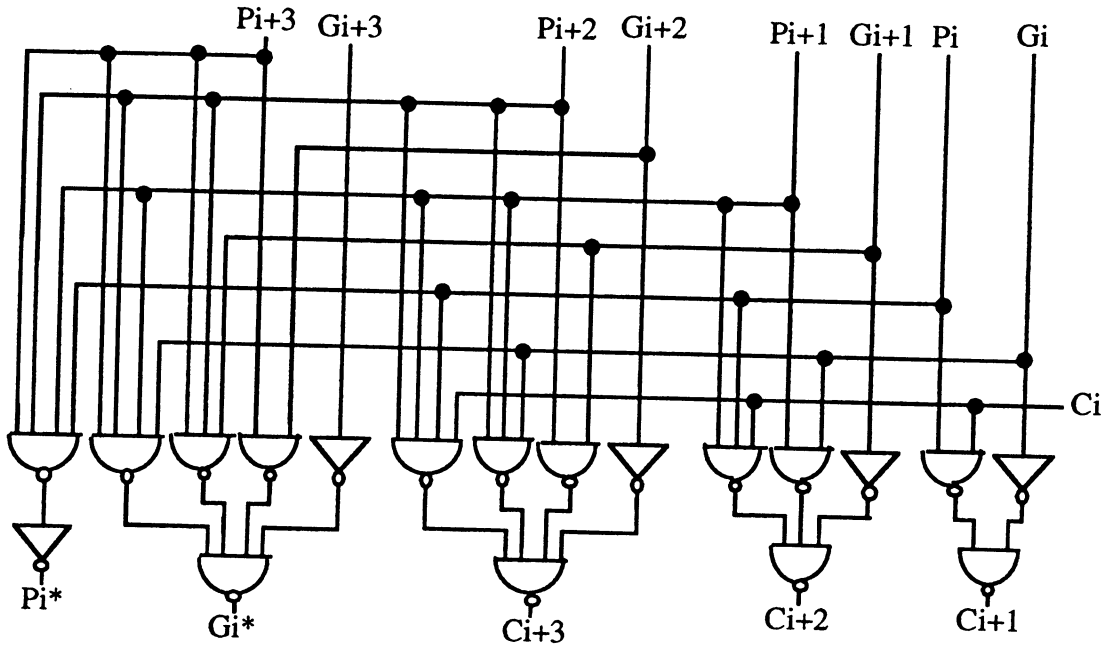


Figure A.11 : Schéma logique de bloc CLA1

Le bloc CLA2 calcule les retenues C4 C8 C12 C16 à partir des variables P0* P1* P2* P3* G0* G1* G2* G3* et son schéma logique est montré dans la figure A.12.

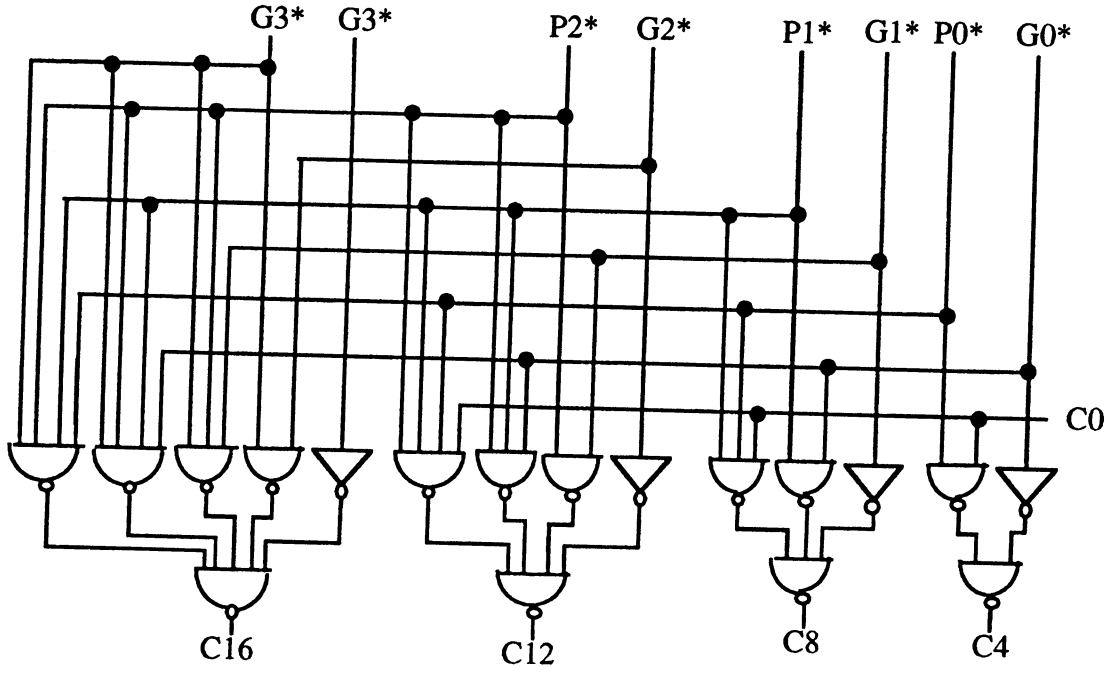


Figure A.12 : schéma logique du bloc CLA2

Et enfin le bloc BS calcule la somme Si qui est définie par les équations logiques suivantes :

$Si = Pi \oplus Ci$ pour les opérations arithmétiques
$Si = Pi \oplus 1$ pour les opérations logiques

Ce type d'UAL est bien adapté pour les circuits où la rapidité est le premier soucis car elle est presque 4 fois plus rapide par rapport à l'UAL de base. Le temps de calcul est la somme des temps de propagation de chaque bloc constituant un groupe (indépendant de la taille des opérandes).

En ce qui concerne la surface, elle est plus importante de 34% par rapport à la surface d'une UAL du type ripple carry [Sou89].

A.3.2 Ripple Within Groups, Look Ahead Between Groups Adder

L'UAL de type "Ripple Within Groups, Look Ahead Between Groups Adder utilise la même approche que Carry Look Ahead mais en modifiant son schéma de base. Cette approche consiste à calculer les retenues (C4, C8, C12, C16) entre les groupes à l'aide des blocs CLA3 (remplacant les bloc CLA1) et le bloc CLA2 puis laisser ces retenues se propager dans chacun des groupes. Les blocs CLA3 dont le schéma logique est montré en figure A.13 génère les variables Pi* et Gi* à partir des Pi et Gi.

Le bloc BS inclut seulement trois additionneus calculant la somme et la retenue et un additionneur qui calcule seulement la somme puisque la retenue de frontière est calculée par le bloc CLA3.

Ce type d'UAL est moins performant que l précédant car elle est rapide par un facteur de 3 seulement par rapport à une UAL classique. La surface suplémentaire est aussi moins importante. Seulement 20% de la surface initiale est nécessaire.

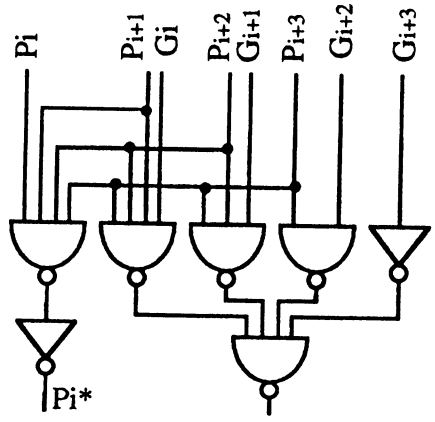


Figure A.13 : schéma logique du bloc CLA3

A.3.3 UAL CARRY LOOK AHEAD WITHIN GROUPS, RIPPLE BETWEEN GROUPS

Le principe de ce type d'UAL est dérivé aussi du "Full Carry Look Ahead Adder". Il consiste à diviser une UAL de 16 bits en quatre groupes où l'anticipation de la reteue est réalisée et propagée par la suite entre ces blocs. Cette UAL utilise donc des additionneurs calculant seulement la somme puisque les retenues dans les groupes sont calculées par les blocs CLA2. Comme dans l'UAL CLA, on retrouve dans chaque groupe les blocs BC, BPG et CLA2 et le temps de calcul est déterminé de la même manière en additionnant les delais de chaque bloc constituant un groupe. Le facteur de rapidité pour ce type d'UAL par rapport à celle de base est de 3,2 qui est payée par une surface supplémentaire de 25,5%.

A.3.4 UAL TYPE BOENT KUNG

En 1982, Brent et Kung ont présenté un type d'additionneur utilisant les mêmes principes que les CLA (Carry Look Ahead) mais beaucoup plus réaliste pour une taille raisonnable des opérandes.

Le principe repose aussi sur le calacul des retenues C_{i+1} en fonction des couples (G_i, P_i), mais au lieu de le faire en temps constant (cas des additionneurs à évaluation anticipée de la retenue, le calcul est fait en un temps proportionnel au logarithme binaire de la taille des opérandes.

Soit en A et B les deux opérandes définis comme suit :

$A = a_{n-1}\ a_{n-2} \ldots a_i \ldots a_1\ a_0$

$B = b_{n-1}\ b_{n-2} \ldots b_i \ldots b_1\ b_0$

et

$G_i = a_i\, b_i$

$P_i = a_i \oplus b_i$

$C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \ldots + P_i P_{i-2} \ldots P_i G_0 + P_i P_{i-1} \ldots P_1 P_0 C_0$

Soit un opérateur "O" est défini par :

$(X1, Y1)\ O\ (X2, Y2) = (X1 + X2Y1, Y1Y2)$

Il a été montré dans [Bre82] que cet opérateur est associatif

(Propriété N°1)

Soit en deux quantités g_i et π_i qui sont définies comme suit :

$$(\gamma_i, \pi_i) = \begin{cases} (G_0, P_0) & \text{si } i=0 \\ (G_i, P_i)\ O\ (\gamma_{i-1}, \pi_{i-1}) & \text{si } 1 \le i \le n \end{cases}$$

avec la propriété N° 2 :

$g_i = C_{i+1}$ et $\pi_i = P_i P_{i-1} \ldots P_0$

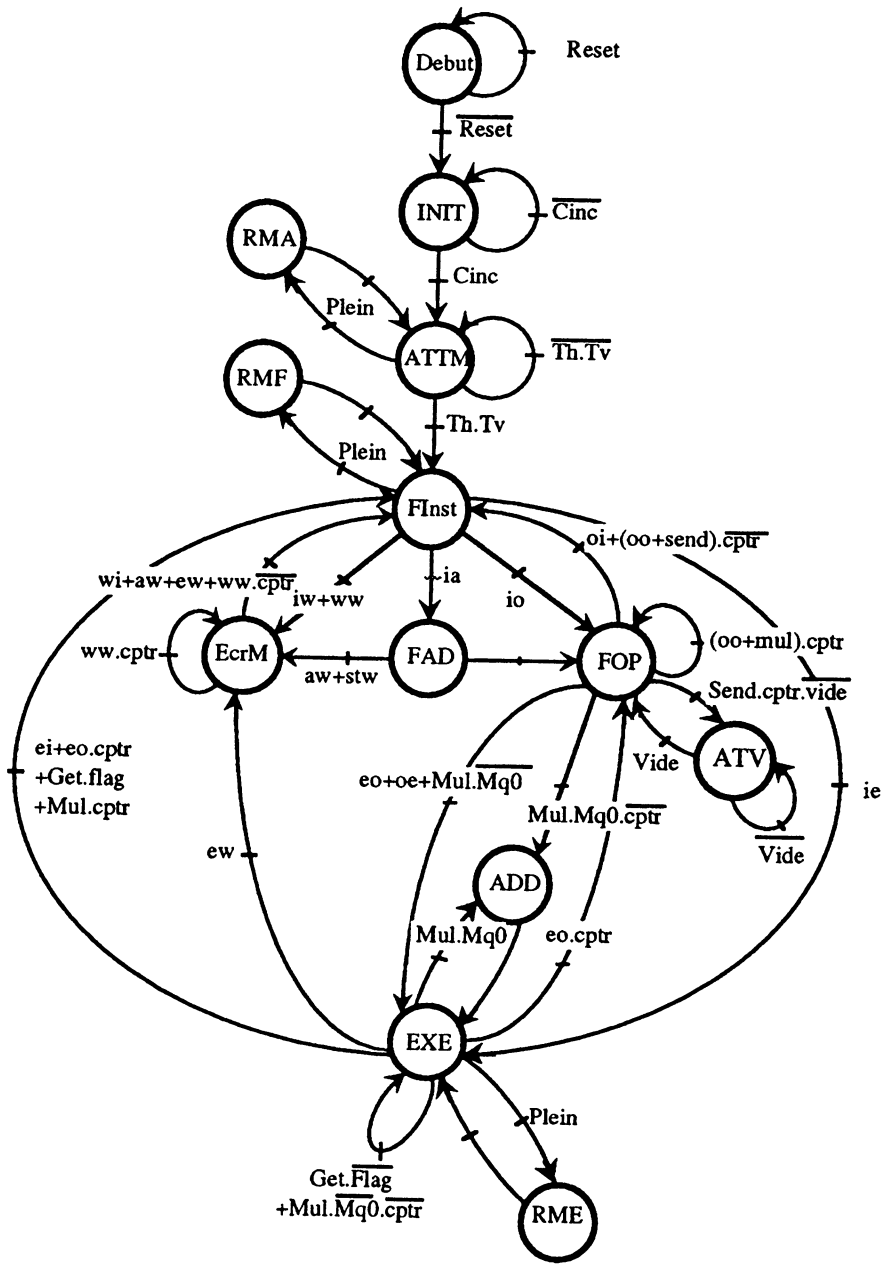
Suite à la démonstration des deux propriétés précédentes, Brent et Kent ont proposé de calculer les retenues à des blocs BBK. Le bloc (BBK) calcule les couples (g_i, π_i) en fonction des (G_i, P_i) avec $C_0 = 0$. Ceci implique que :

$g_i = C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \ldots + P_i P_{i-1} \ldots P_2 P_1 G_0$

$\pi_i = P_i P_{i-1} \ldots P_2 P_1 P_0$

En ce qui concerne les performances et la surface, elle est plus rapide d'un facteur de 2,5 et nécessite une surface supplémentaire d'un tiers par rapport à celle de base.

ANNEXE 4 : GRAPHE DE CONTRÔLE



NOMS DES ÉTATS

AT = Début
RF = Remize à zero Flag
AM = Attente Messages
FI = Fetch Instruction
FO = Fetch Opérand
FA = Fetch Adresse
AD = Adition (multiplication)
EX = Exécution
EM = Ecriture Mémoire
LMA = Lecture Message(A)
LMF = Lecture Message(F)
LME = Lecture Message(E)

CONDITIONS DES TRANSITIONS :

Reset : reset
Cinc : Retenue incrementeur
Tv : Transparence Verticale
Th : Transparence Horizontale
Cptr : Sortie compteur
Mq0 : Bit 0 du Registre Mq
Flag : 9 ième bit mémoire
Mul : Instruction Multiplication
Get : Instruction Get

GROUPES D'INSTRUCTIONS

io : Fetch(Inst.) --> Fetch(Opér)
oo : Fetch(Opér.) -->Fetch(Opér)
oi : Fetch(Opér.)--> Fetch(Inst)
ia : Fetch(Inst.) --> Fetch(Adr)
ao : Fetch(Adr) --> Fetch(Opér)
aw : Fetch(Adr)--> Ecrit. Mém.
iw : Fetch(Inst.) --> Ecrit. Mém.
wi : Ecrit. Mém. --> Fetch(Inst.)
ww : Ecrit. Mém. --> Ecrit. mém.
oe : Fetch(Opér) --> Exécution
eo : Exécution --> Fetch(Opér)
ew : Exécution --> Ecrit. Mém.
ie : Fetch(Inst.) --> Exécution
ei : Exécution --> Fetch(Inst.)

CODES ÉTATS

DÉBUT	: 0111
RF	: 1111
AM	: 0110
FI	: 0100
FO	: 0010
FA	: 1000
AD	: 0001
EX	: 0000
EM	: 0011
LMA	: 1100
LMF	: 1010
LME	: 0101

ANNEXE 5 : LISTINGS DES PROGRAMMES DE GÉNÉRATION DES VECTEURS DE TEST

```
/*****************************************************************
                        /*Bit8.c : PLA Groupe*/
*****************************************************************/

#include <stdio.h>

main()
{
FILE *ptr_file, *fopen();
int i, x, y, t;
int a[16], b[16];
     ptr_file=fopen ("bit8test","w");
     if (ptr_file==NULL) {printf("erreur");}
     t=0;
     for (i=0; i<16; ++i) {
     a[i]=i;
     b[i]=i;
     }
     for (x=0; x<16; ++x) {
     for (y=0; y<16; ++y) {
     printf("%d %x %x\n",t,a[x],b[y]);
     fprintf(ptr_file,"%d %x %x\n",t,a[x],b[y]);
     t=t+100;
     }
     }
     fclose(ptr_file);
}
```

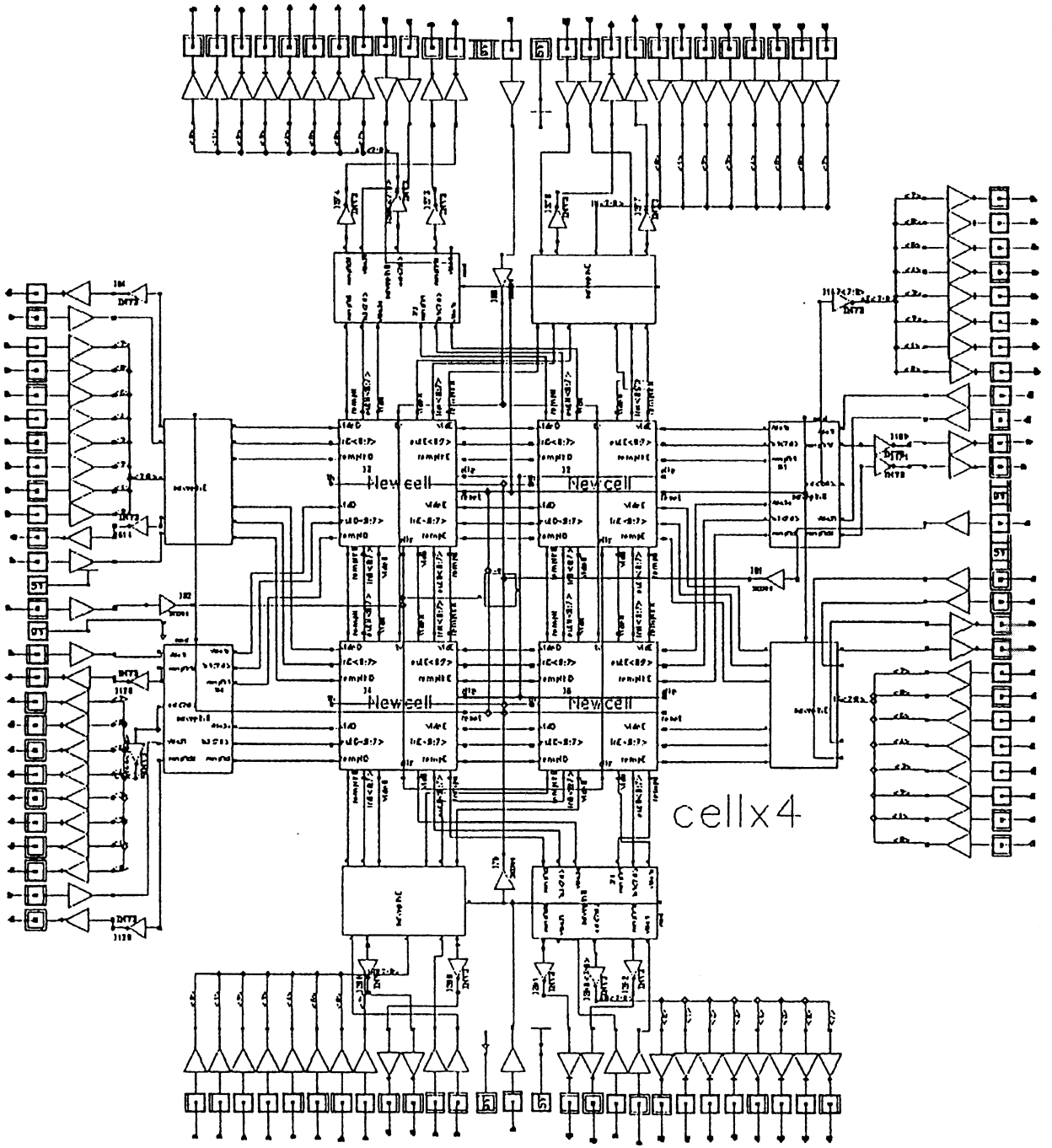
```
*****************************************************************
          /*Bit12.c : PLA AdresseA et PLA Mémoire*/
*****************************************************************
#include <stdio.h>
main()
{
FILE *ptr_file, *fopen();
int i, x, y, z;
unsigned long t;
int a[16], b[16], c[16];
     ptr_file=fopen ("bit12test","w");
     if (ptr_file==NULL) {printf("erreur");}
     t=0;
     for (i=0; i<16; ++i) {
     a[i]=i;
     b[i]=i;
     c[i]=i;
     }
     for (x=0; x<16; ++x) {
     for (y=0; y<16; ++y) {
     for (z=0; z<16; ++z) {
     /*printf("%d %x %x %x\n",t,a[x],b[y],c[z]);*/
     fprintf(ptr_file,"%lu %x %x %x\n",t,a[x],b[y],c[z]);
     t=t+100;
     }
     }
     }
     fclose(ptr_file);
}
```

```
*****************************************************************
    /*Bit13.c : PLA interface Routage et PLA status(calcul)*/
*****************************************************************

#include <stdio.h>
main()
{
FILE *ptr_file, *fopen();
int i, x, y, z, v;
unsigned long t;
int a[16], b[16], c[16];
     ptr_file=fopen ("bit13test","w");
     if (ptr_file==NULL) {printf("erreur");}
     t=0;
     v=0;
     for (i=0; i<16; ++i) {
     a[i]=i;
     b[i]=i;
     c[i]=i;
     }
     for (x=0; x<16; ++x) {
     for (y=0; y<16; ++y) {
     for (z=0; z<16; ++z) {
     /*printf("%d %x %x %x %d\n",t,a[x],b[y],c[z],v);*/
     fprintf(ptr_file,"%lu %x %x %x %d\n",t,a[x],b[y],c[z],v);
     if (v==0)
          v=1;
     else
          v=0;
     t=t+150;
     }
     }
     }
     fclose(ptr_file);
}
```

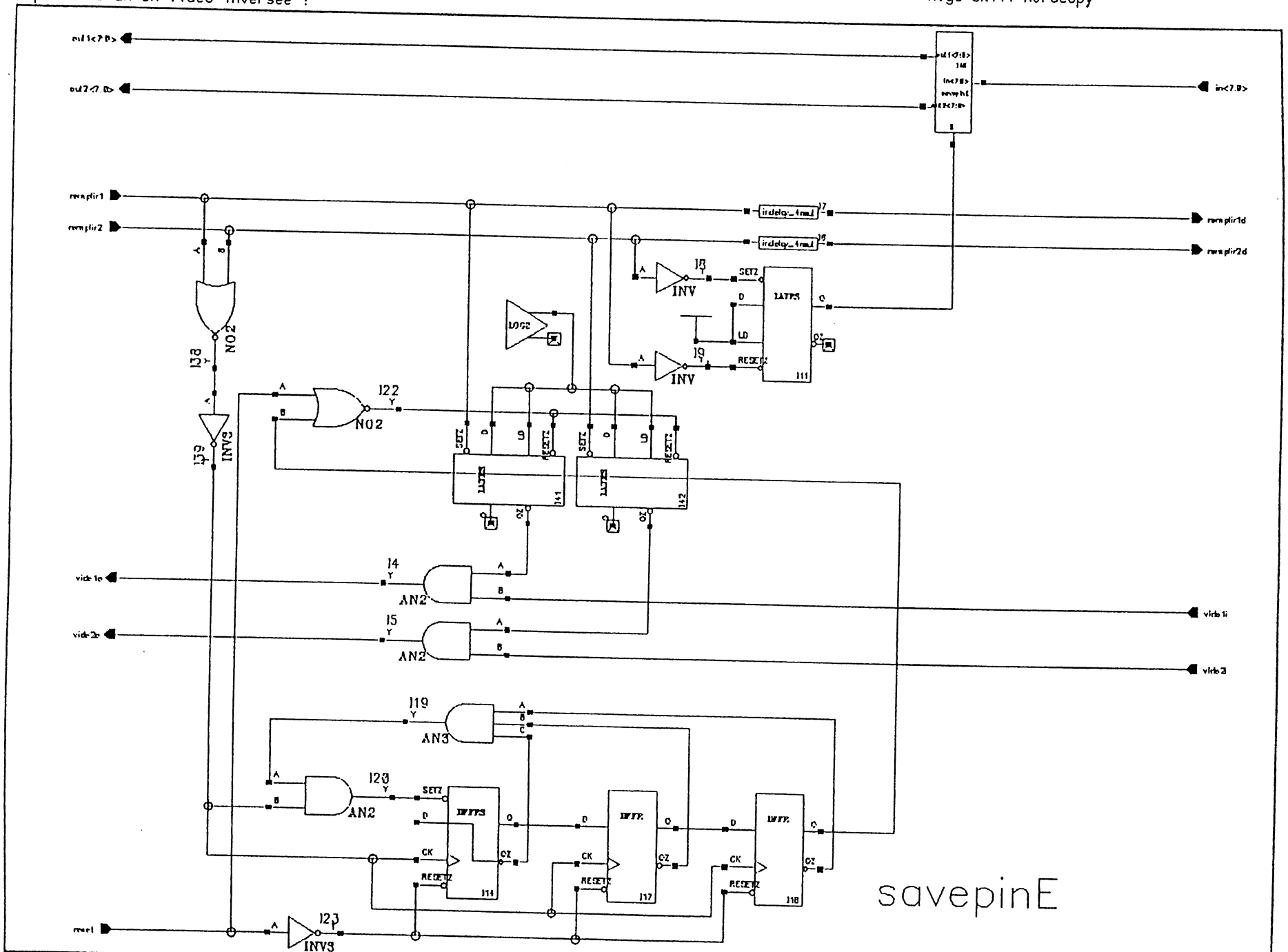
```
*****************************************************************
                    /*Bit15.c : PLA calcul*/
*****************************************************************
#include <stdio.h>
main()
{
FILE *ptr_file, *fopen();

int i, j, x, y, z, w;
unsigned long t;
int a[16], b[16], c[16], d[8];

                ptr_file=fopen ("bit15test","w");
        if (ptr_file==NULL) {printf("erreur");}

        t=0;
        for (j=0; j<8; ++j) {
        d[j]=j;
        }
        for (i=0; i<16; ++i) {
        a[i]=i;
        b[i]=i;
        c[i]=i;
        }
        for (x=0; x<16; ++x) {
        for (y=0; y<16; ++y) {
        for (z=0; z<16; ++z) {
        for (w=0; w<8; ++w) {
fprintf(ptr_file,"%lu %x %x %x %o\n",t,a[x],b[y],c[z],d[w]);
t=t+250;
        }
        }
        }
        }
        fclose(ptr_file);
}
```

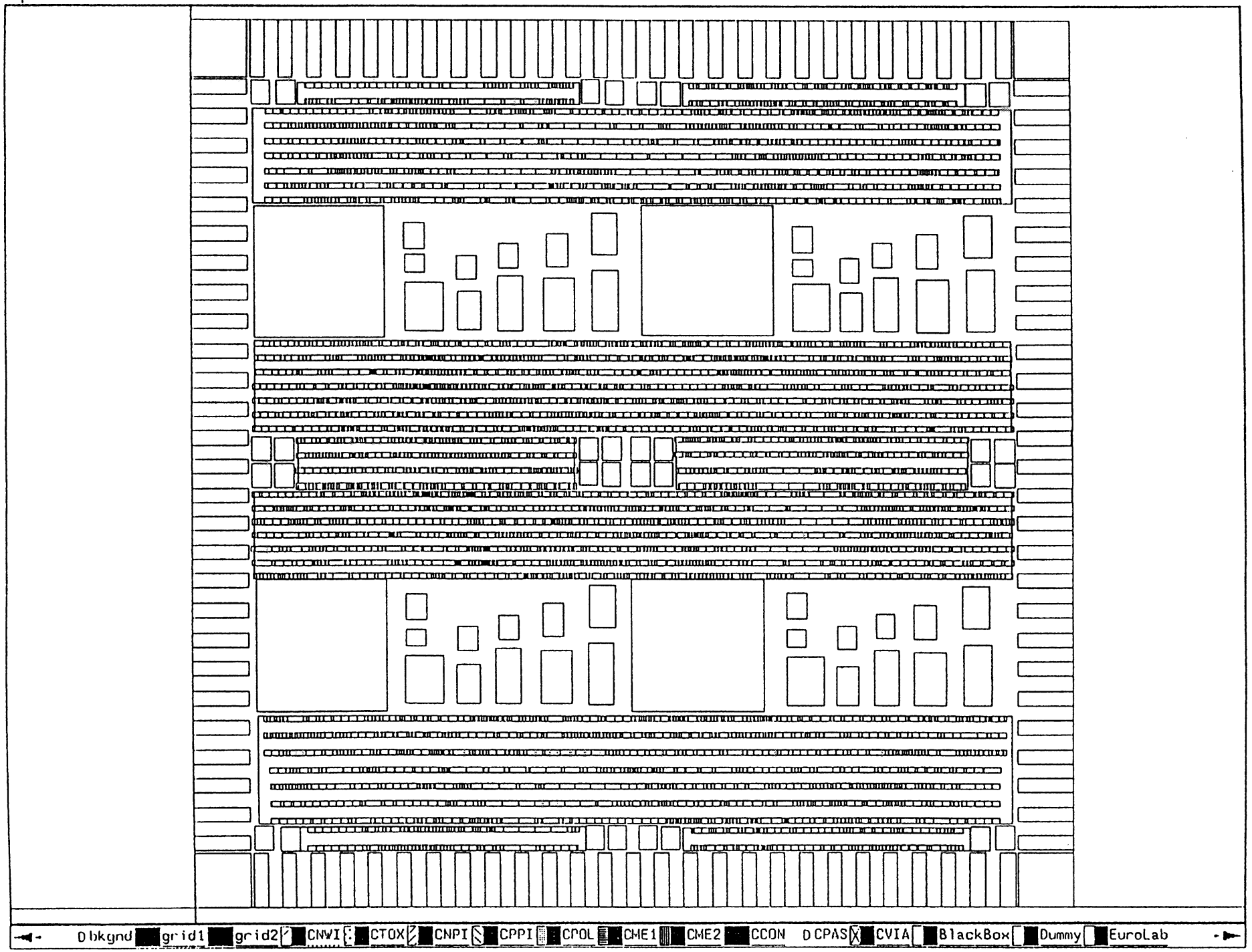
```
*****************************************************************
                    /*Bit24.c PLA AdresseA*/
*****************************************************************
#include <stdio.h>
main()
{
FILE *ptr_file, *fopen();
int i, x, y, z, w, k, l;
unsigned long t;
int a[16], b[16], c[16], e[16], f[16], g[16];
     ptr_file=fopen ("bit13test","w");
     if (ptr_file==NULL) {printf("erreur");}
          t=0;
     for (i=0; i<16; ++i) {
     a[i]=i;
     b[i]=i;
     c[i]=i;
     e[i]=i;
     f[i]=i;
     g[i]=i;
     }
     for (x=0; x<16; ++x) {
     for (y=0; y<16; ++y) {
     for (z=0; z<16; ++z) {
     for (w=0; w<16; ++w) {
     for (k=0; k<16; ++k) {
     for (l=0; l<16; ++l) {
     /*printf("%lu %x %x %x %x %x
%x\n",t,a[x],b[y],c[z],e[w],f[k],g[l]);*/
     fprintf("%lu %x %x %x %x %x
%x\n",t,a[x],b[y],c[z],e[w],f[k],g[l]);
     t=t+200;
     }
     }
     }
     }
     }
     }
     fclose(ptr_file);
}
*****************************************************************
```

ANNEXE 6 : COPIES D'ÉCRAN DES SCHÉMATIQUES, SIMULATIONS ET LAYOUT

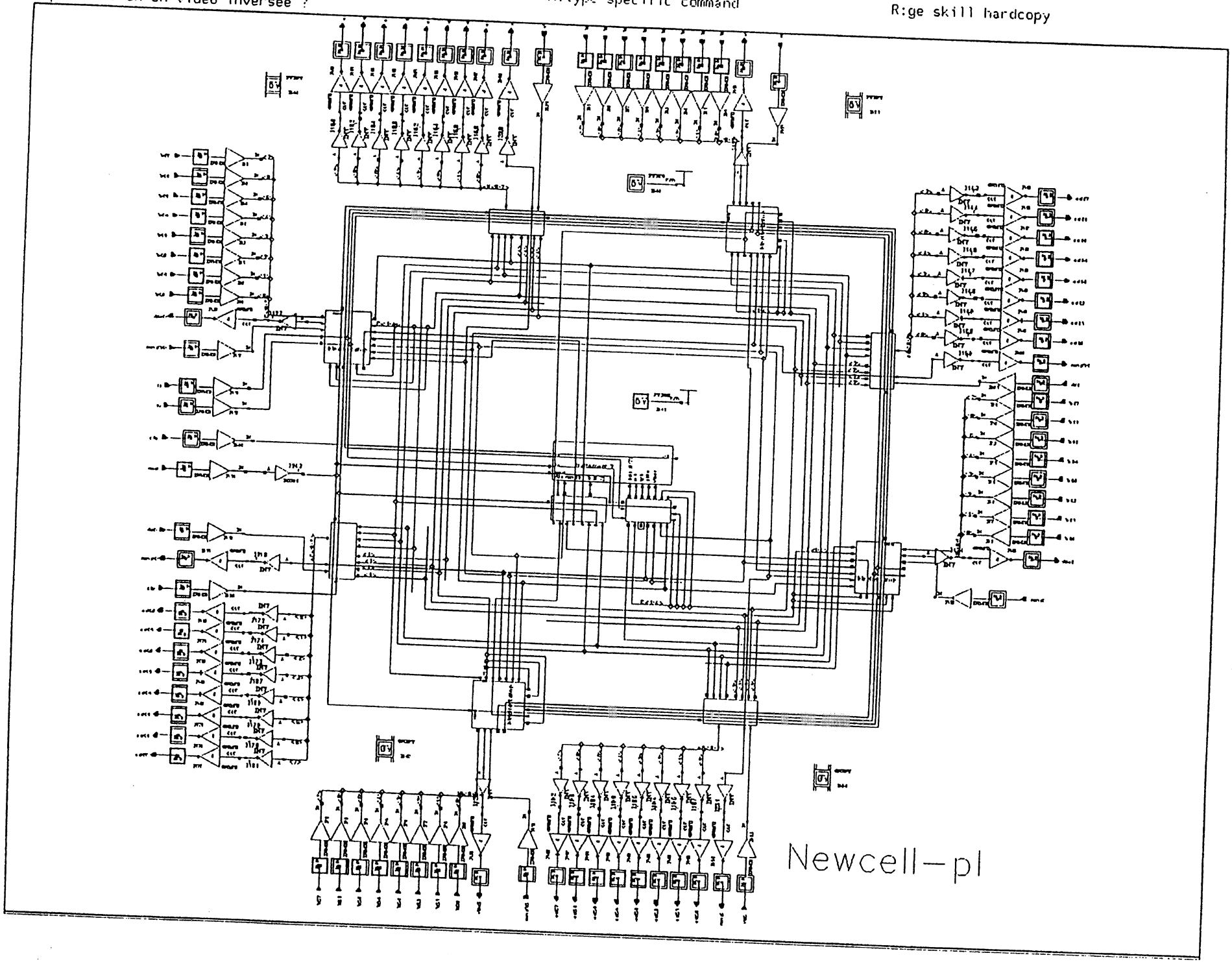

Newcell
Newcell
Newcell
Newcell
cellx4

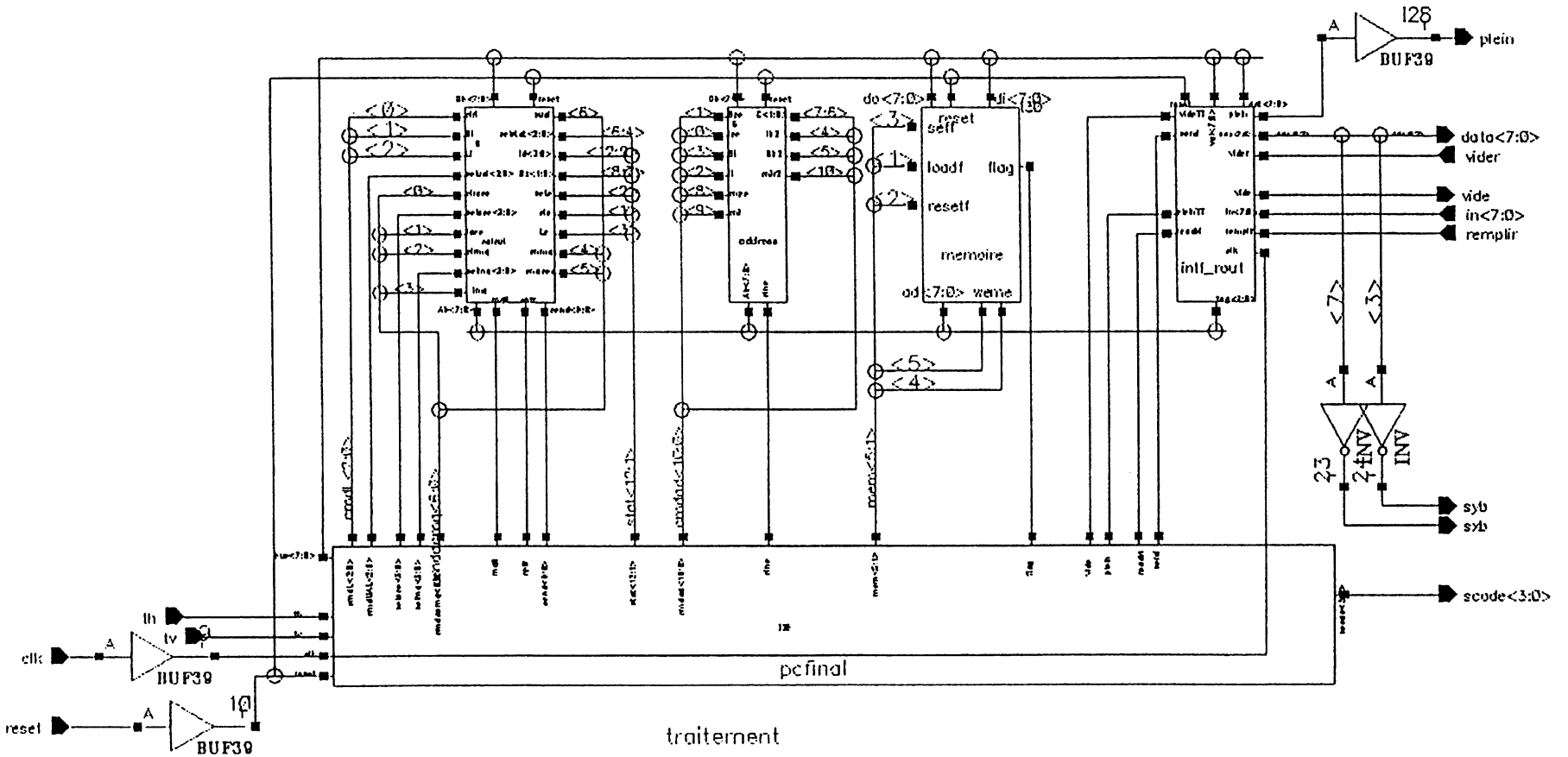
D bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



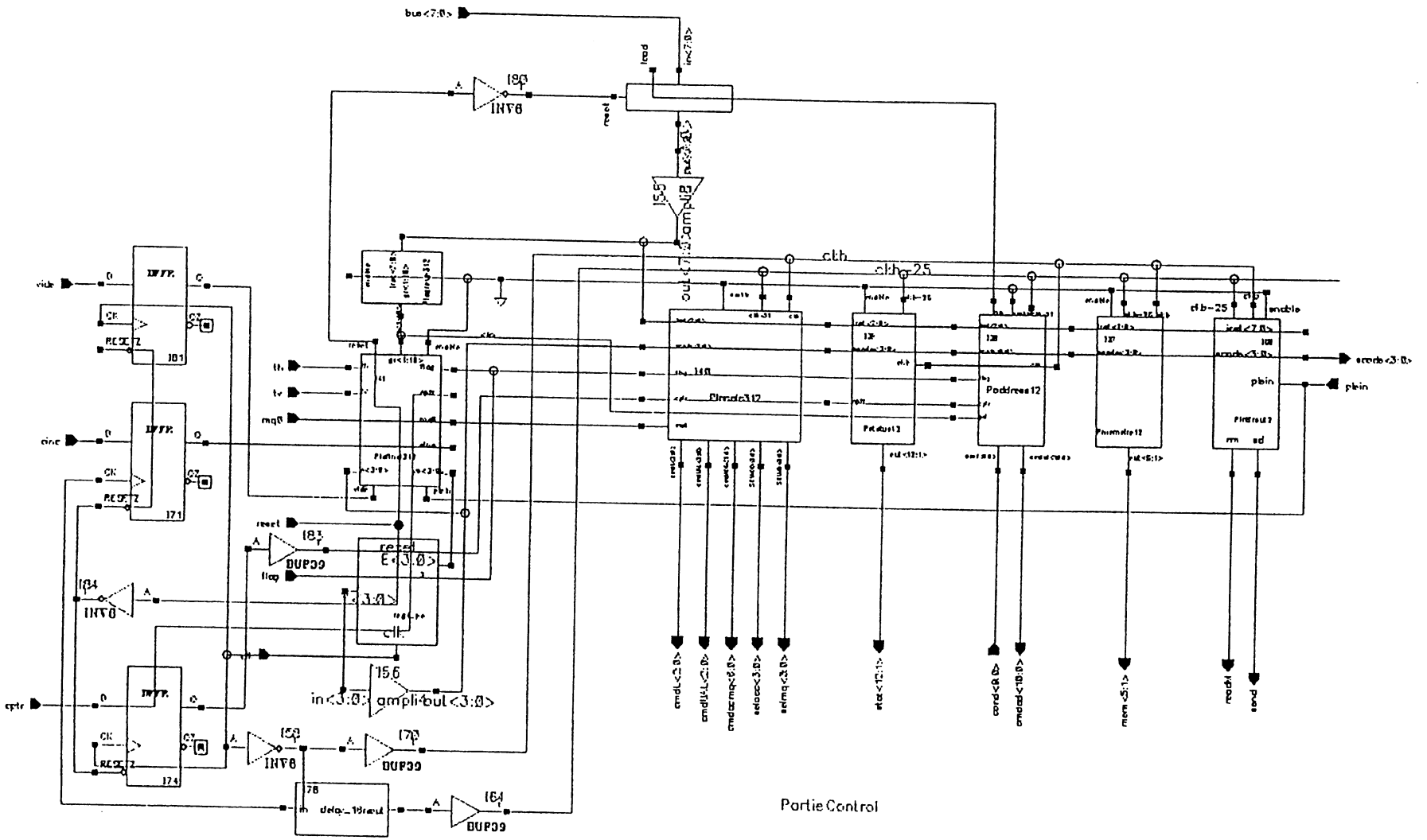

instance
L:pop-up menu
M:type-specific command
R:ge skill hardcopy
Copie d'ecran en video inversee ?
Dbkgnd
grid1
grid2
CNWI
CTOX
CNPI
CPPI
CPOL
CME1
CME2
CCON
DCPAS
CVIA
BlackBox
Dummy
EuroLab

D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



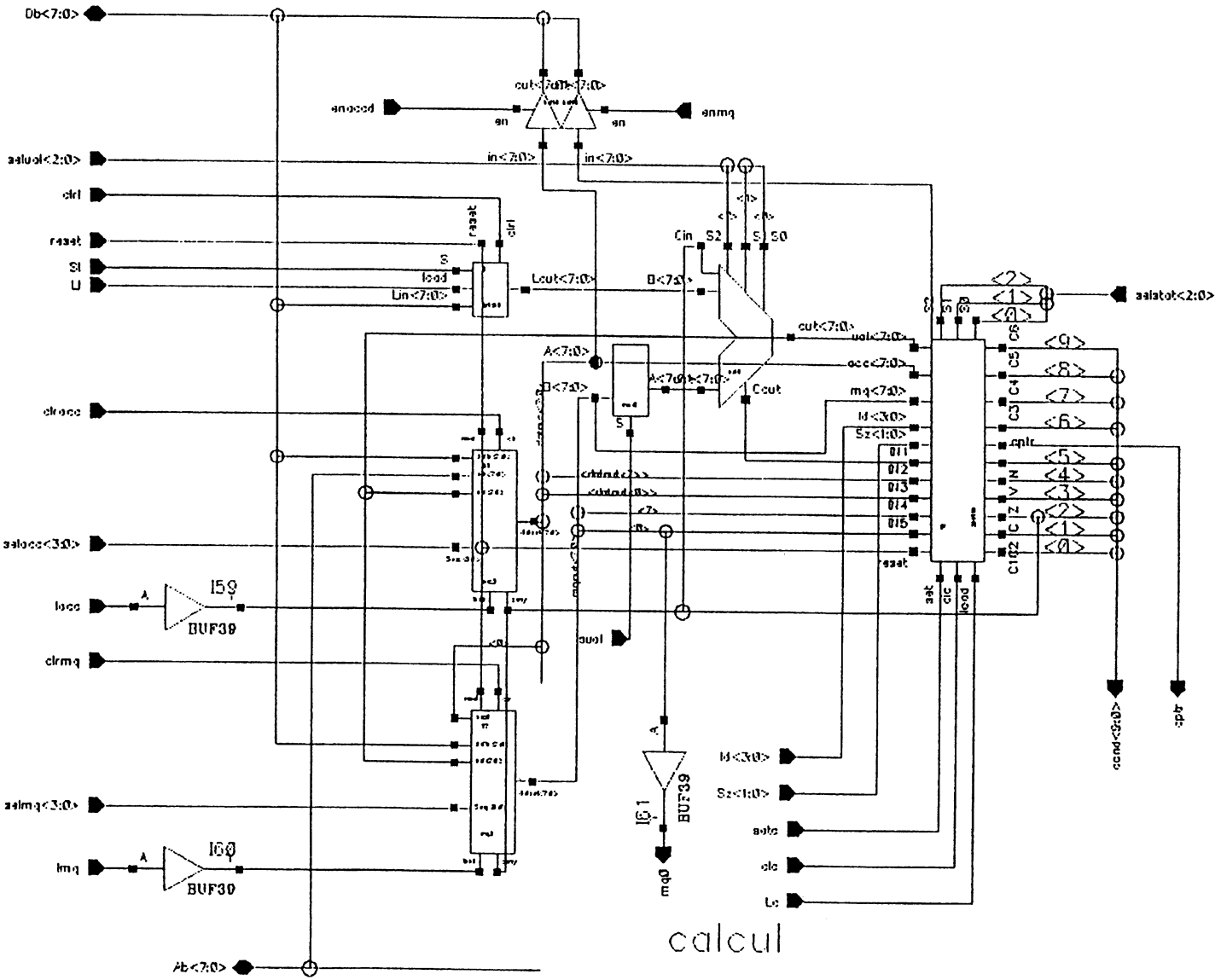

BUF39
plein
data<7:0>
vider
vide
in<7:0>
remplir
syb
syb
scode<3:0>
reset
loadf
flag
resetf
memoire
weme
intf_rout
pcfinal
traitement
clk
reset
BUF39
BUF39
INV
INV

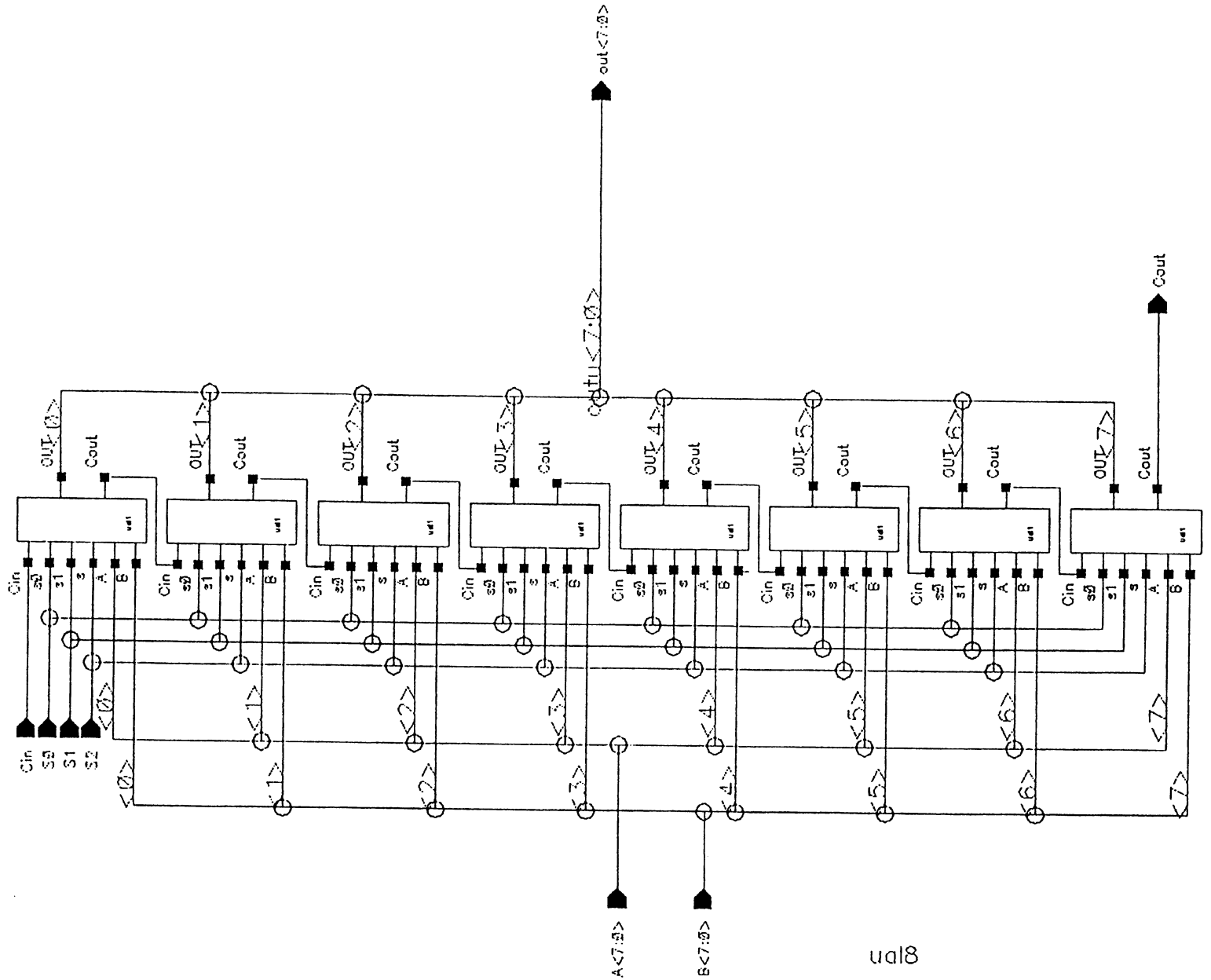
D bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



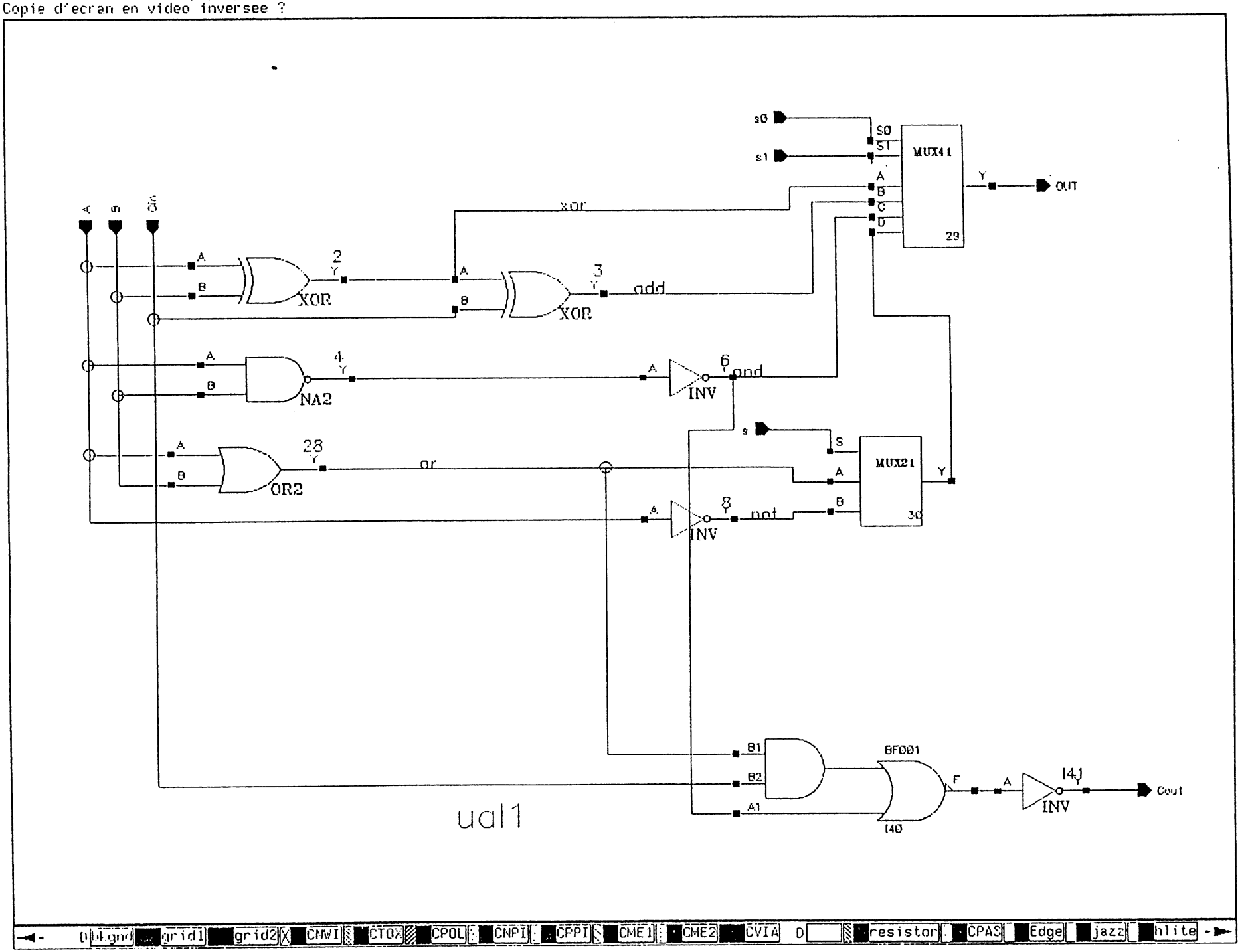
D bkgnd grid1 grid2 X CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite

D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?




out<7:0>
Cout
OUT<0>
OUT<1>
OUT<2>
OUT<3>
OUT<4>
OUT<5>
OUT<6>
OUT<7>
Cout
Cin
S0
S1
S2
A
B
A<7:0>
B<7:0>
ual8

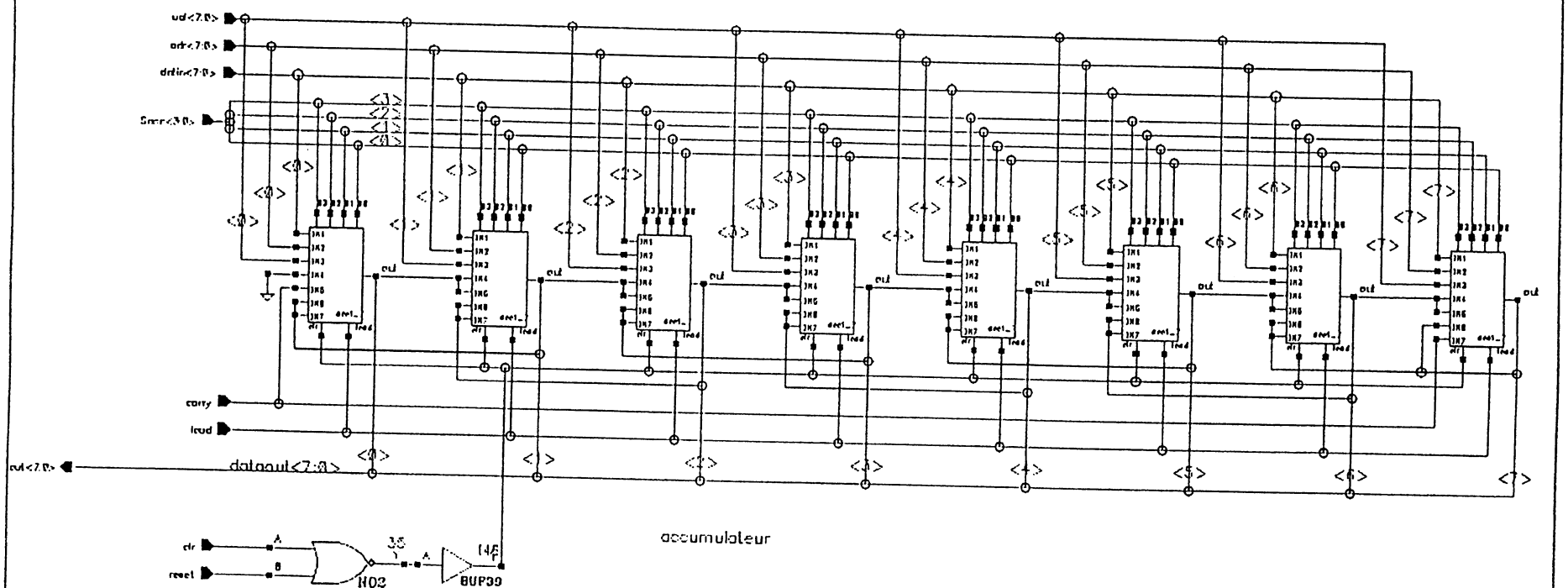
D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



D:bkgnd grid1 grid2 X CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite

D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy

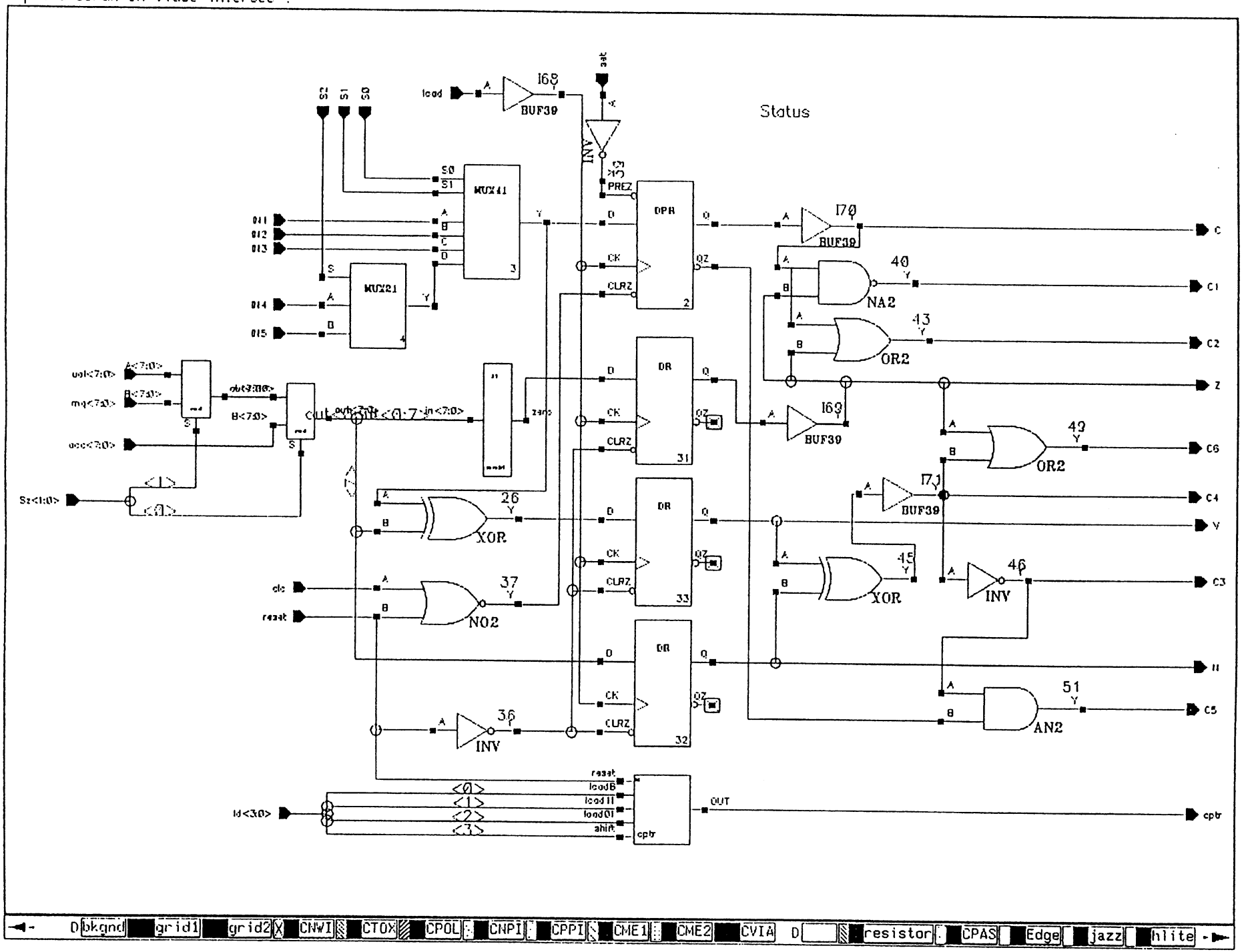
Copie d'ecran en video inversee ?

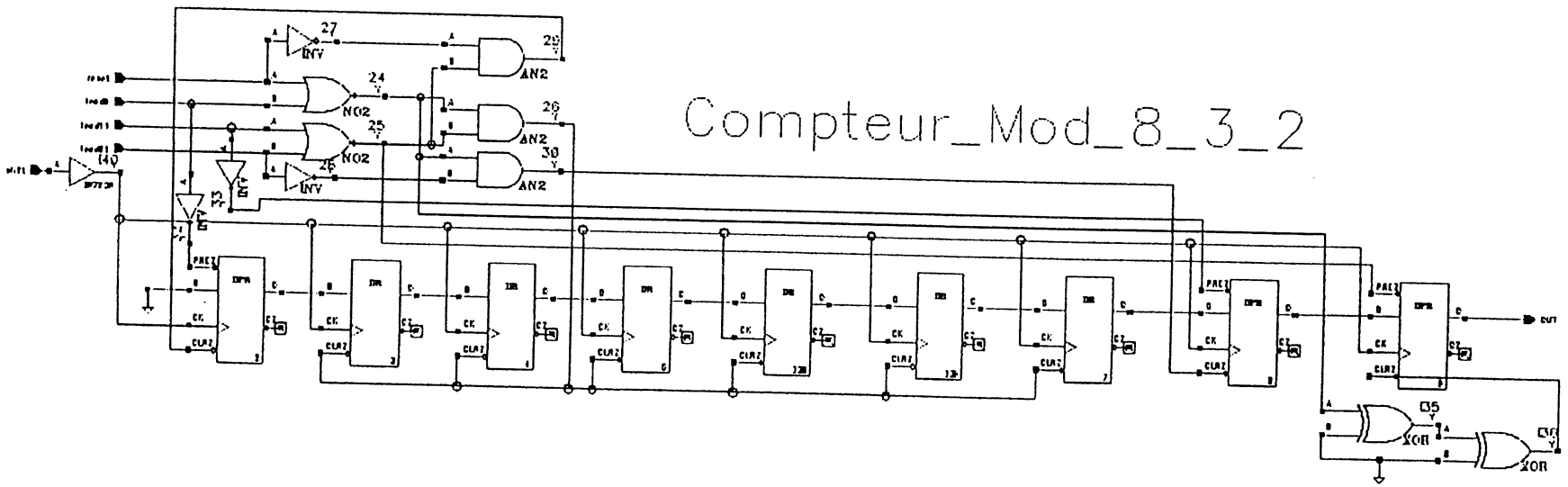


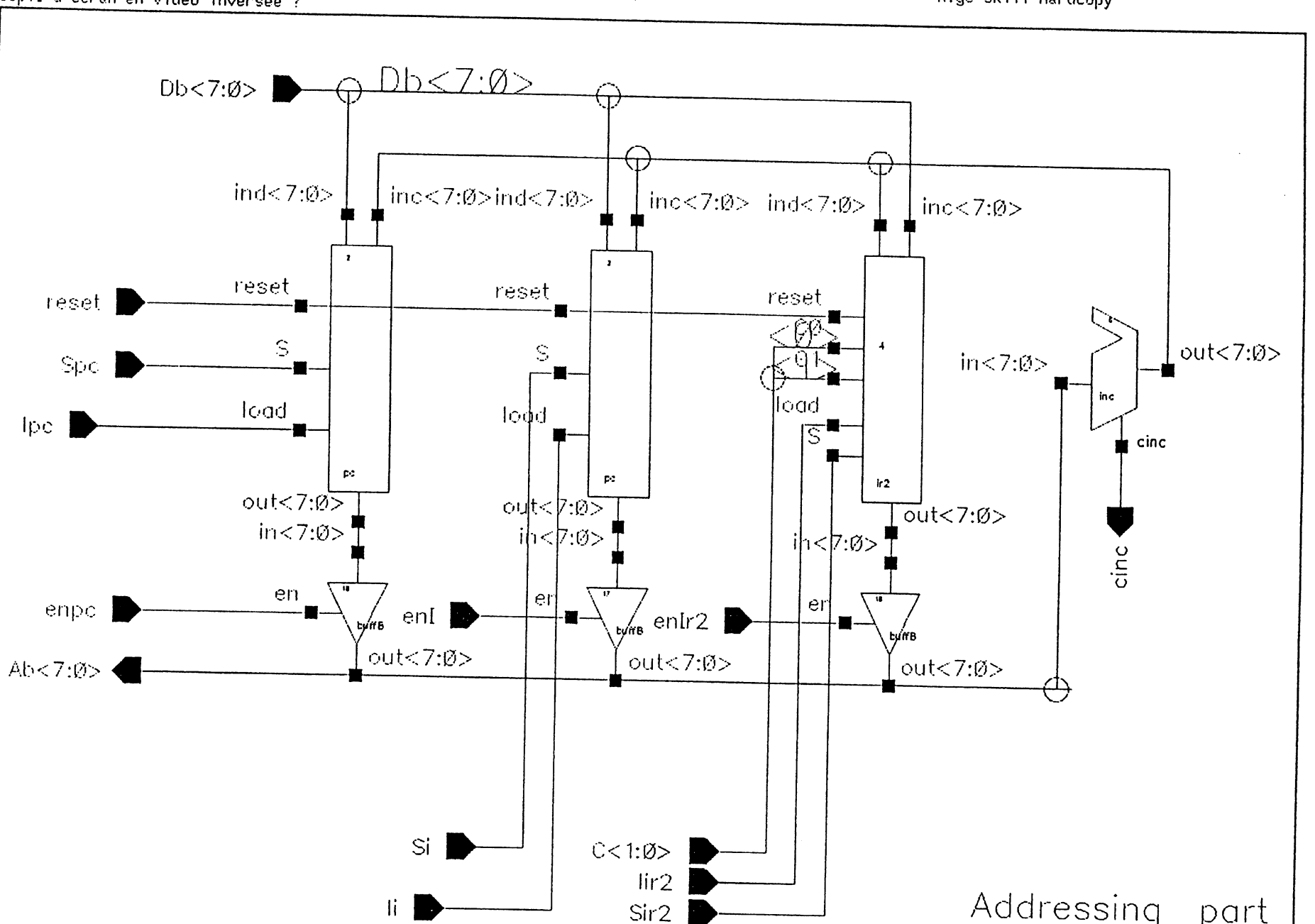
accumulateur

D bkgnd | grid1 | grid2 | X | CNWI | CTOX | CPOL | CNPI | CPPI | CME1 | CME2 | CVIA | D | resistor | CPAS | Edge | jazz | hlite

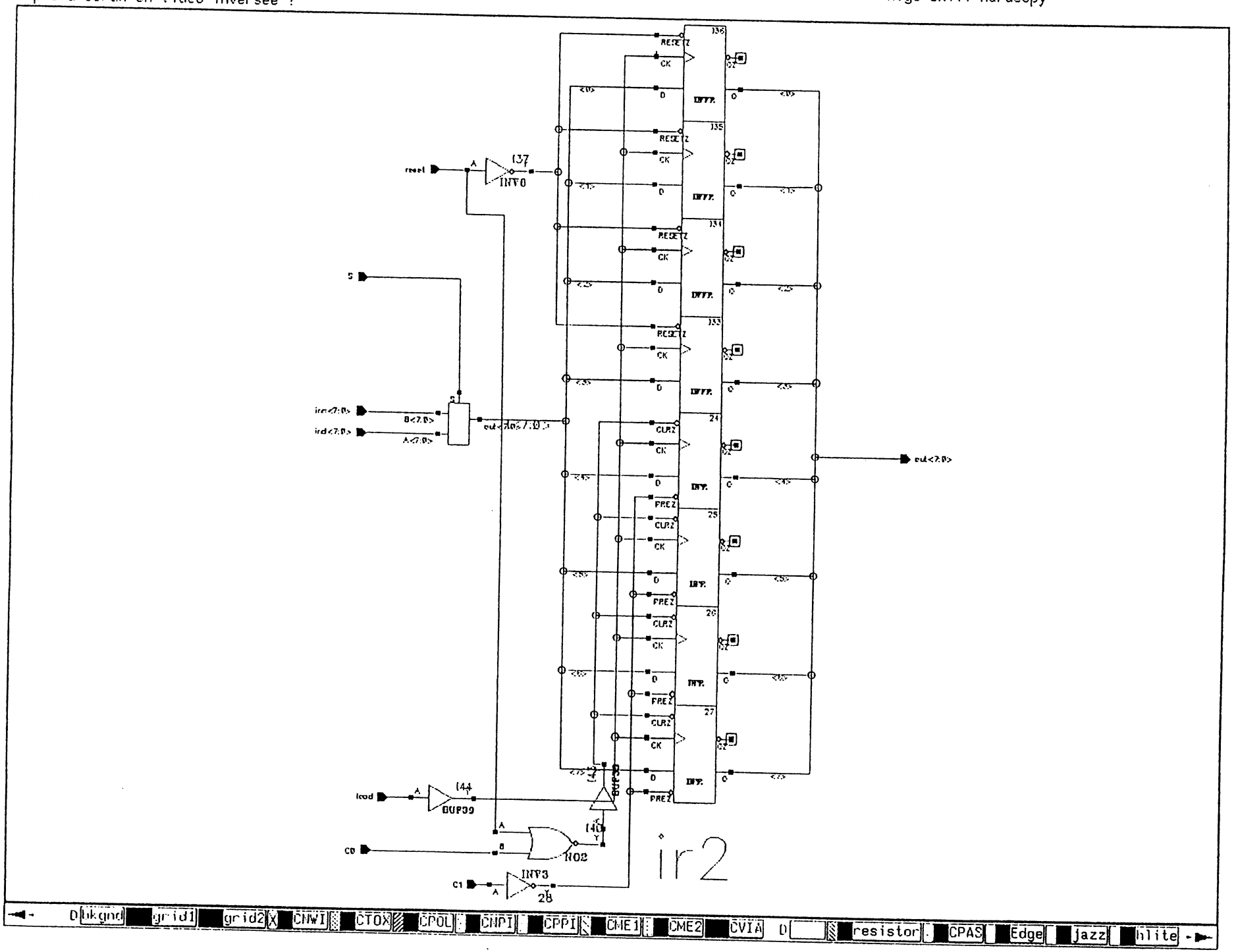
D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?




Compteur_Mod_8_3_2

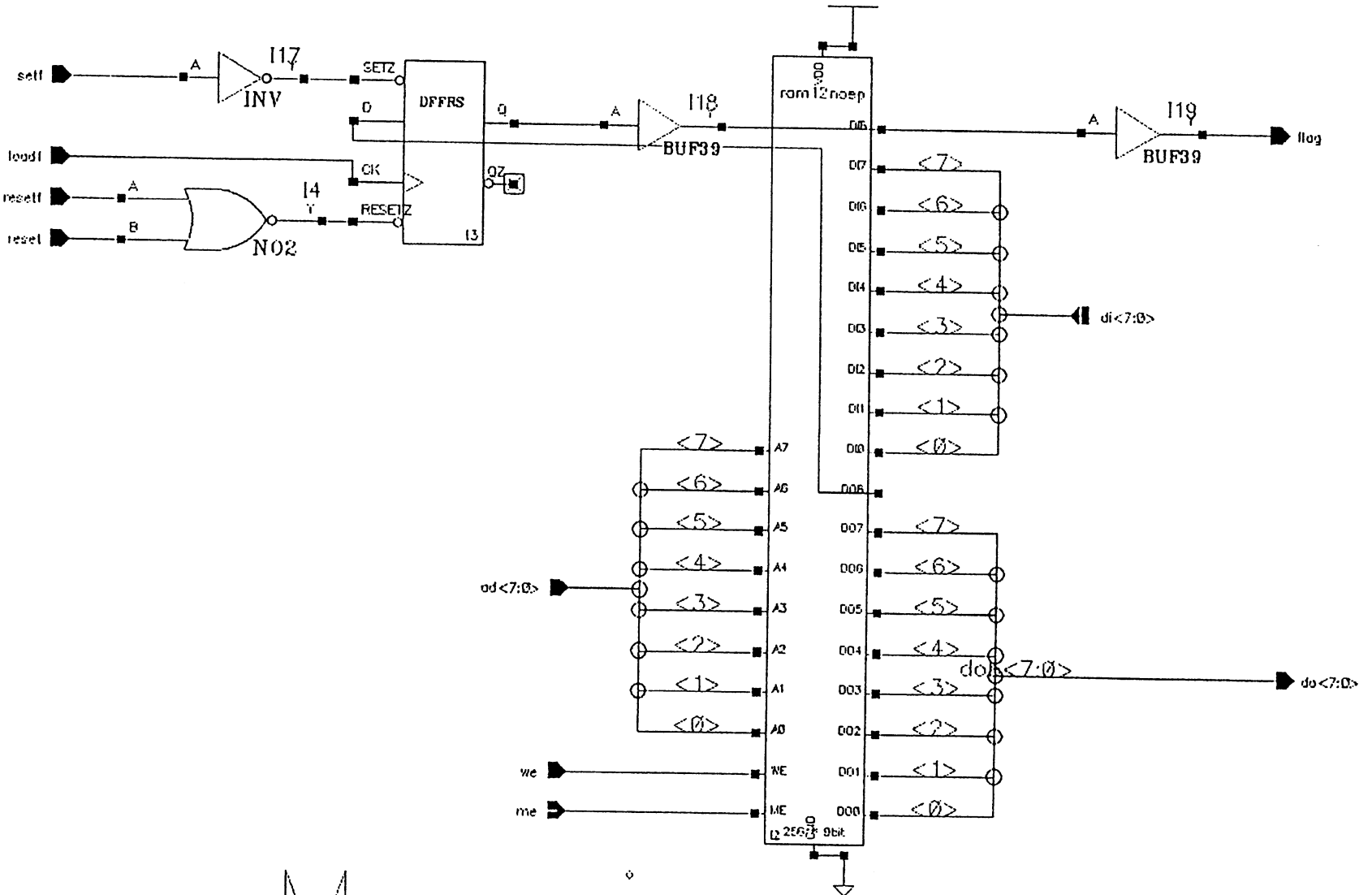


D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



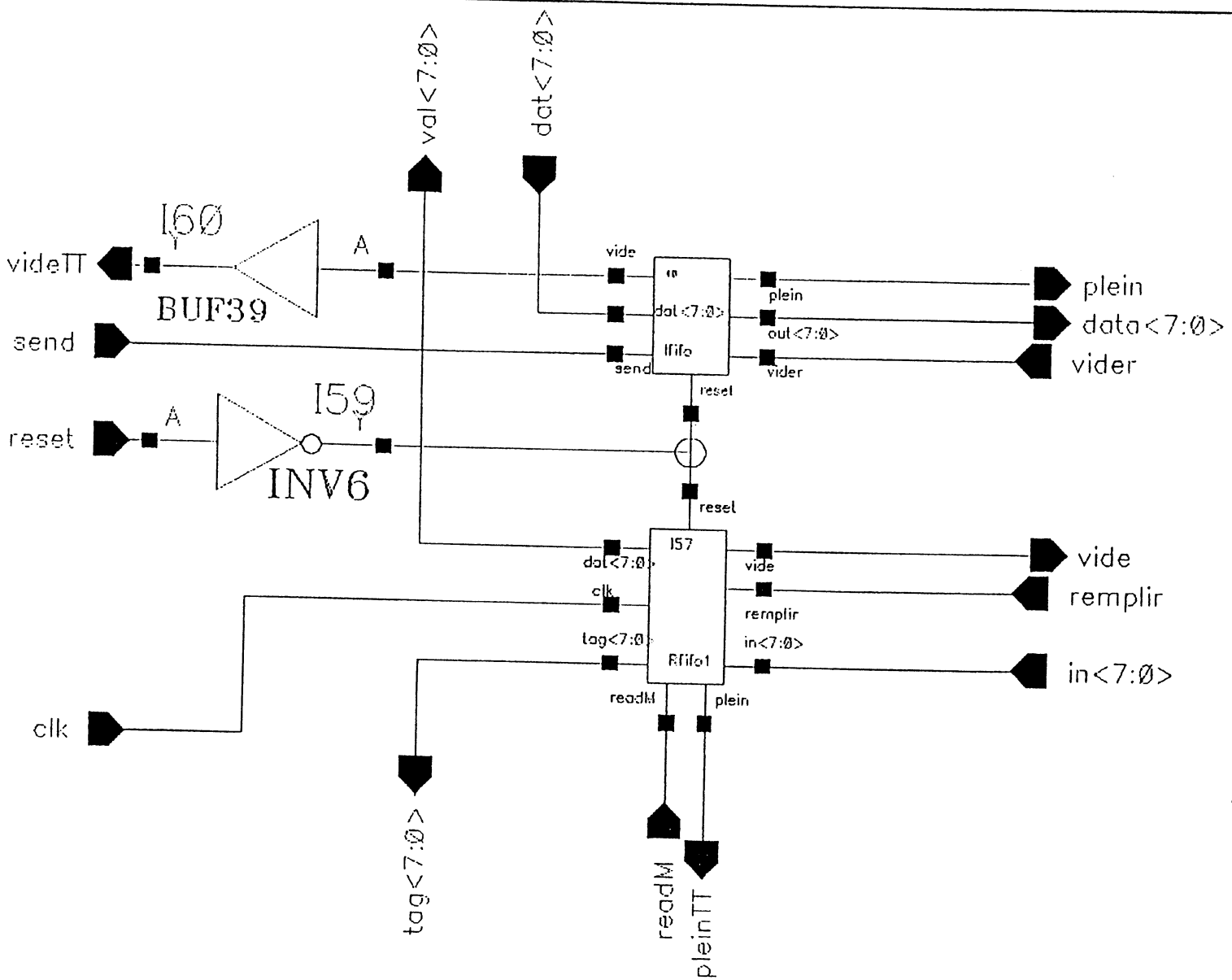
D bkgnd | grid1 | grid2 | CNWI | CTOX | CPOL | CNPI | CPPI | CME1 | CME2 | CVIA | D | resistor | CPAS | Edge | jazz | hlite

D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



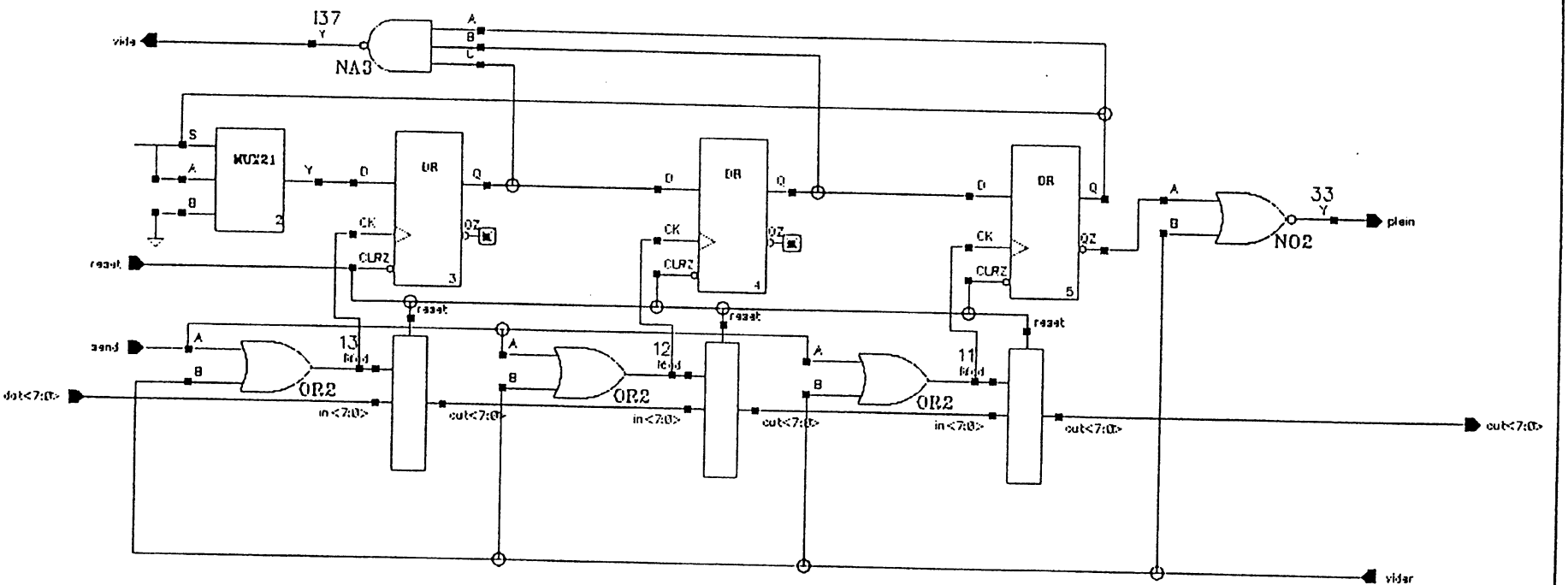
D bkgnd grid1 grid2 X CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite

D bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



Interface_Routage

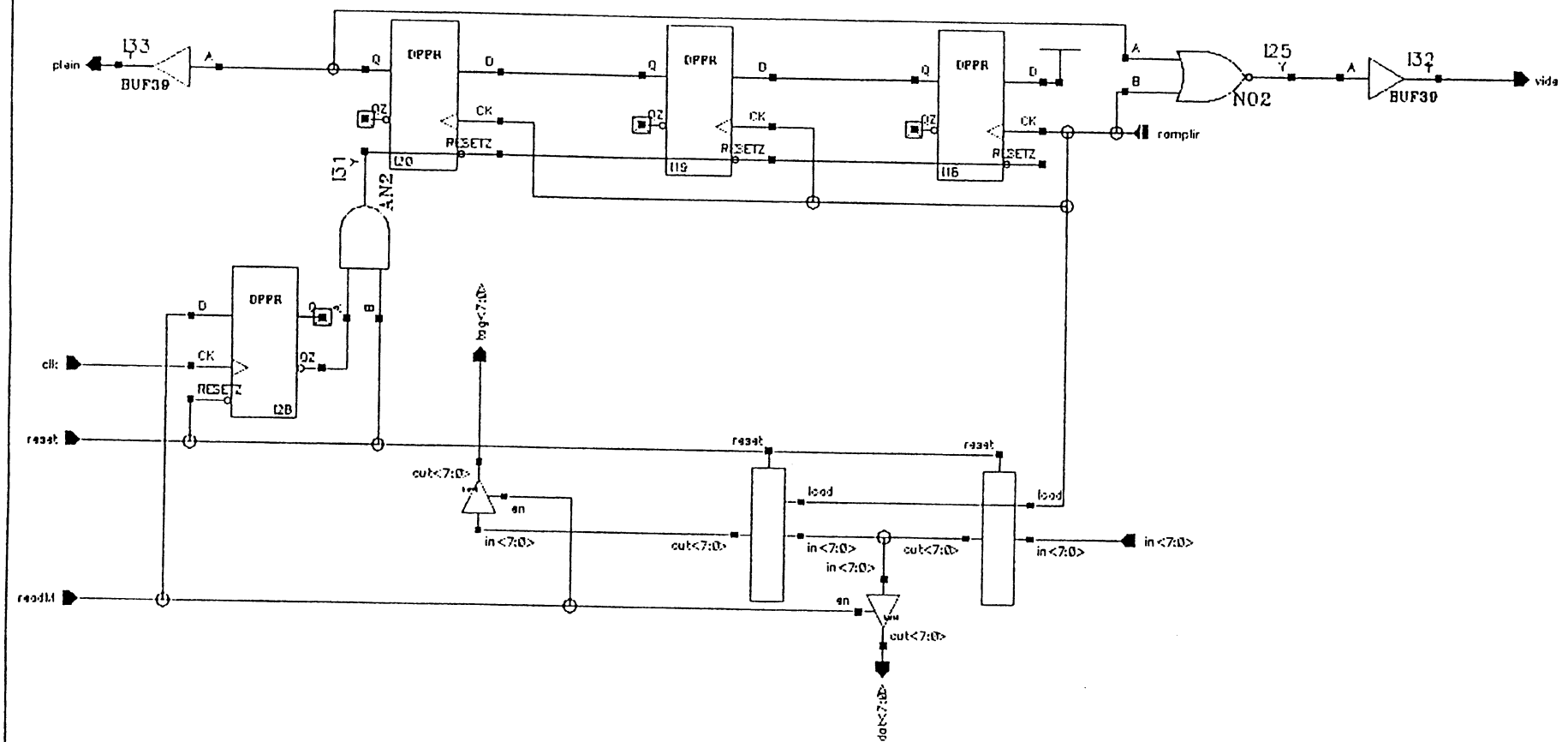
D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?



lfifo_interface_Traitement/Routage

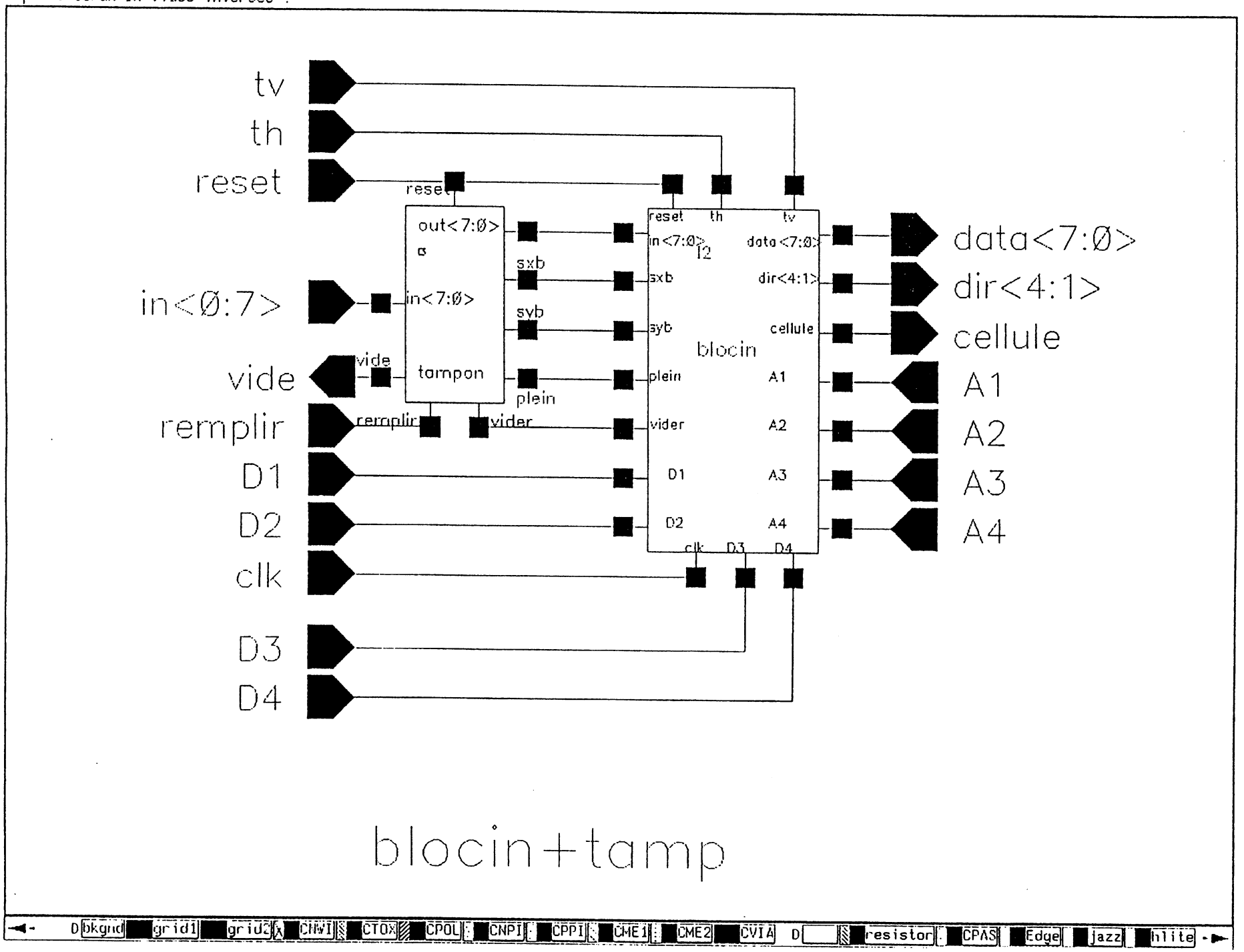
D bkgnd grid1 grid2 CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite

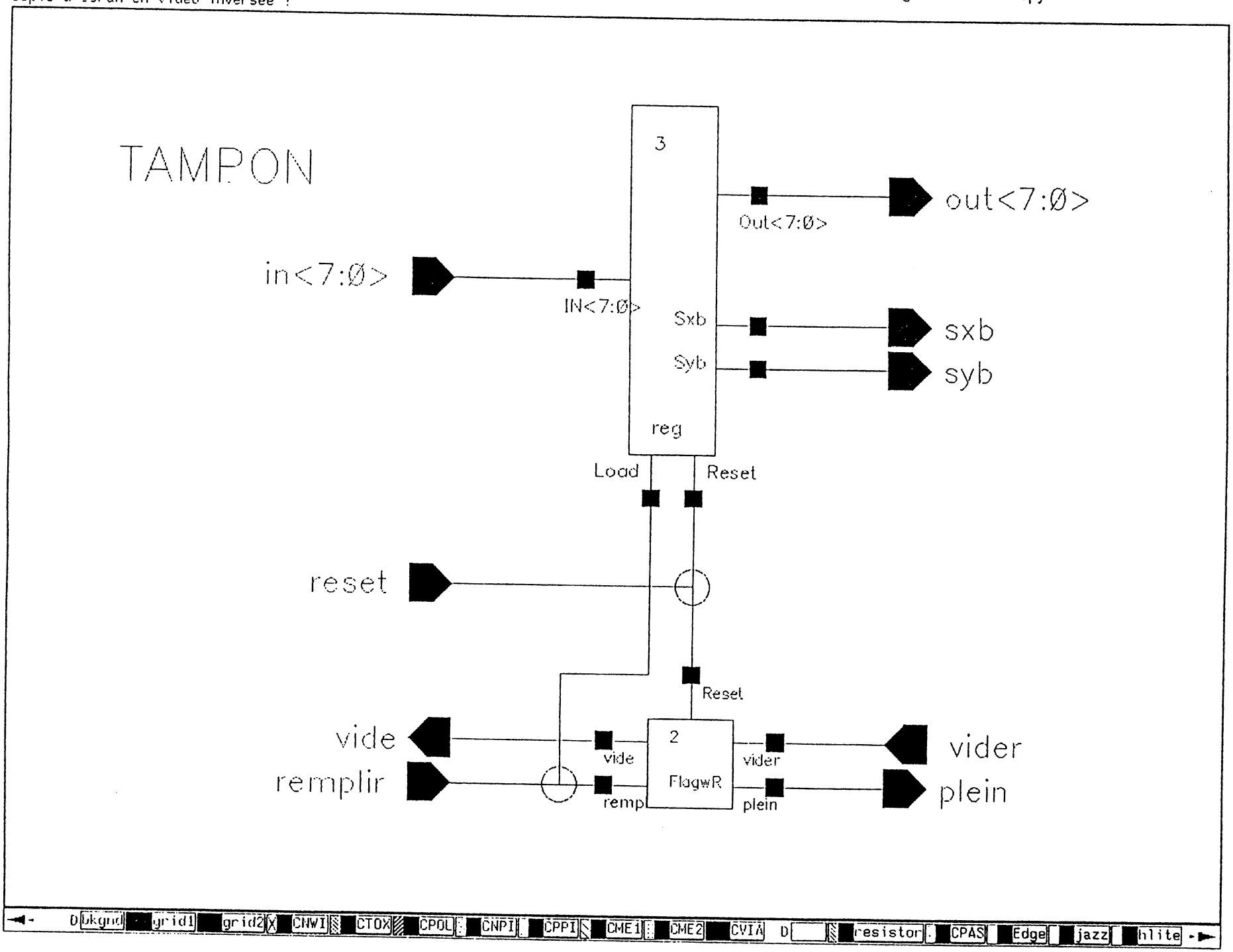
D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?

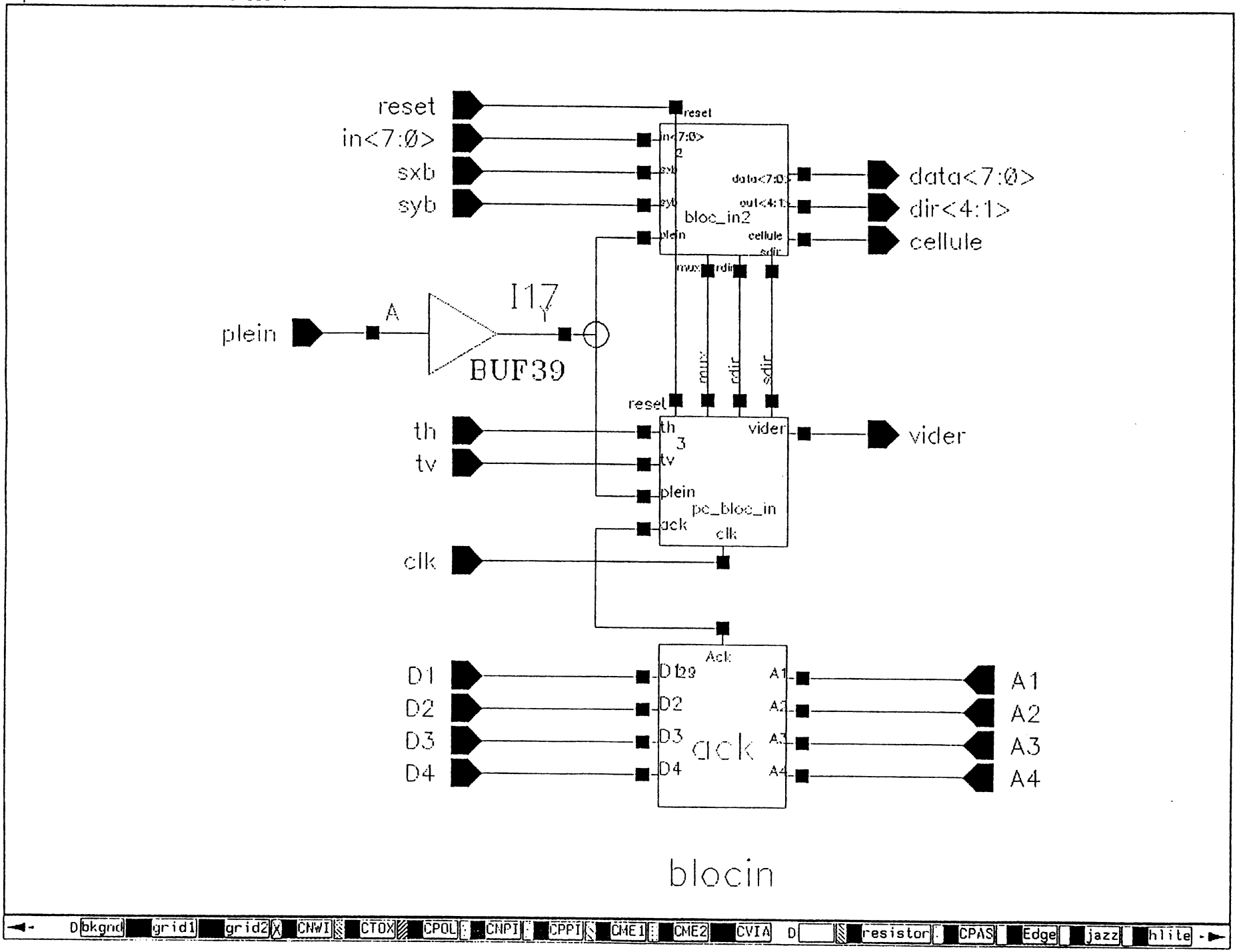


Rfifo_Interface_Routage/Traitement

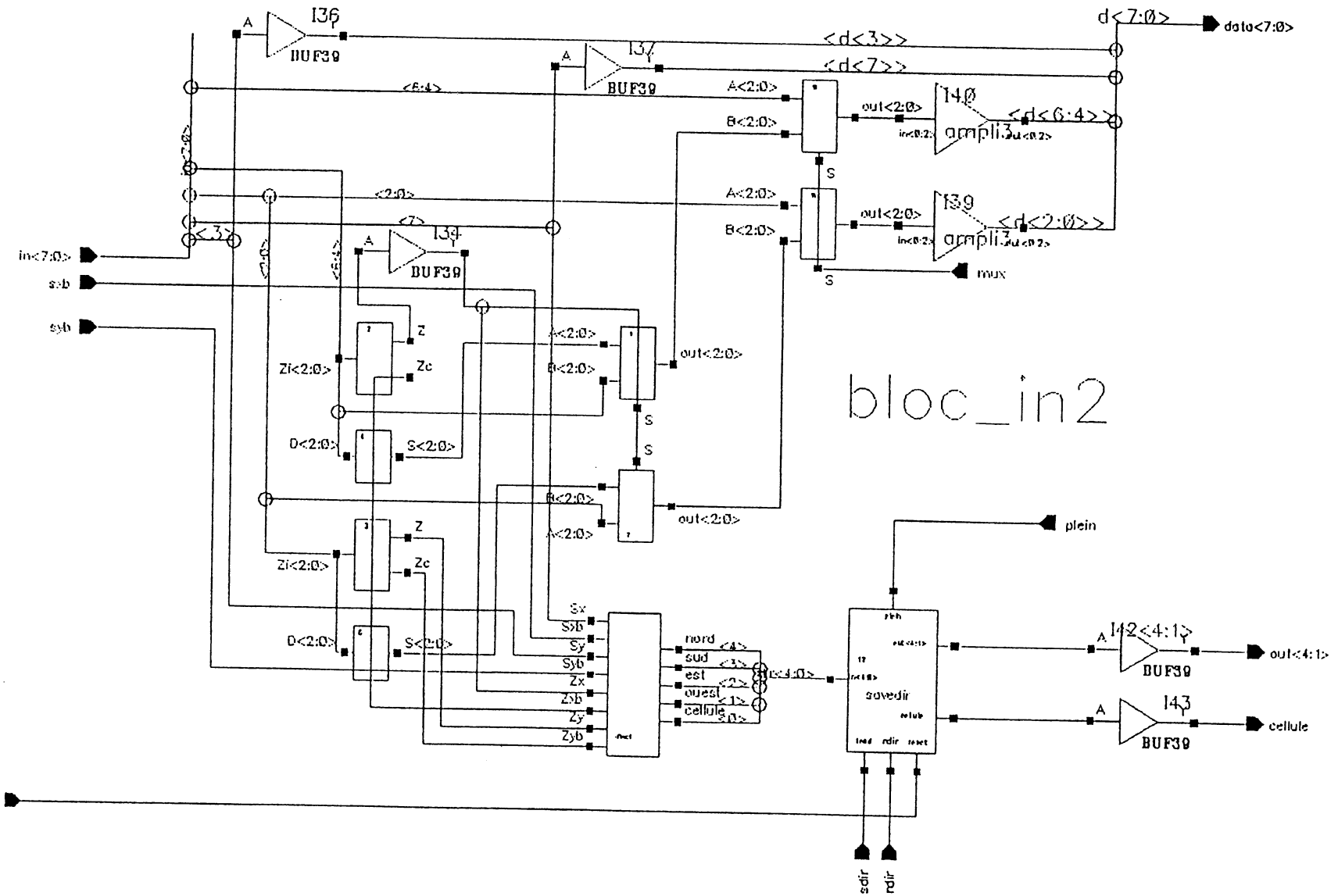
D:bkgnd grid1 grid2 X CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite

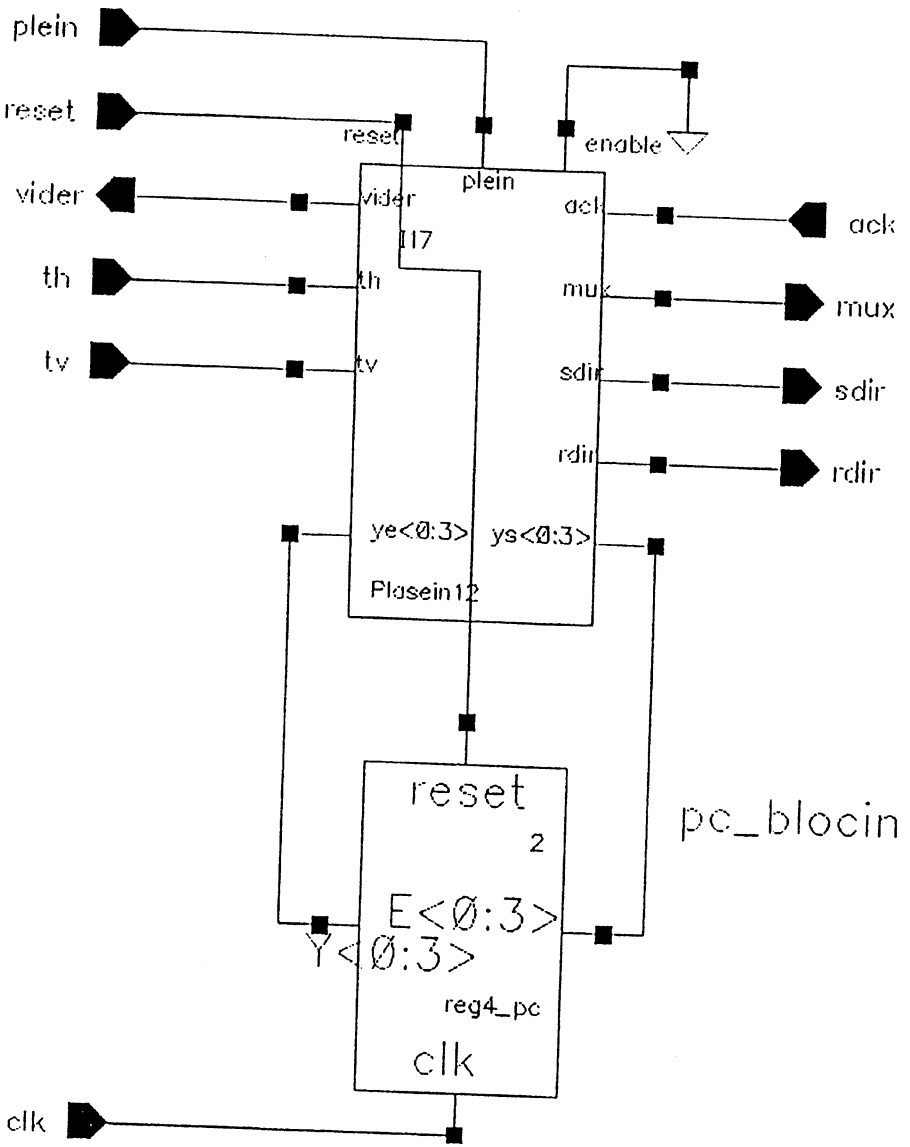


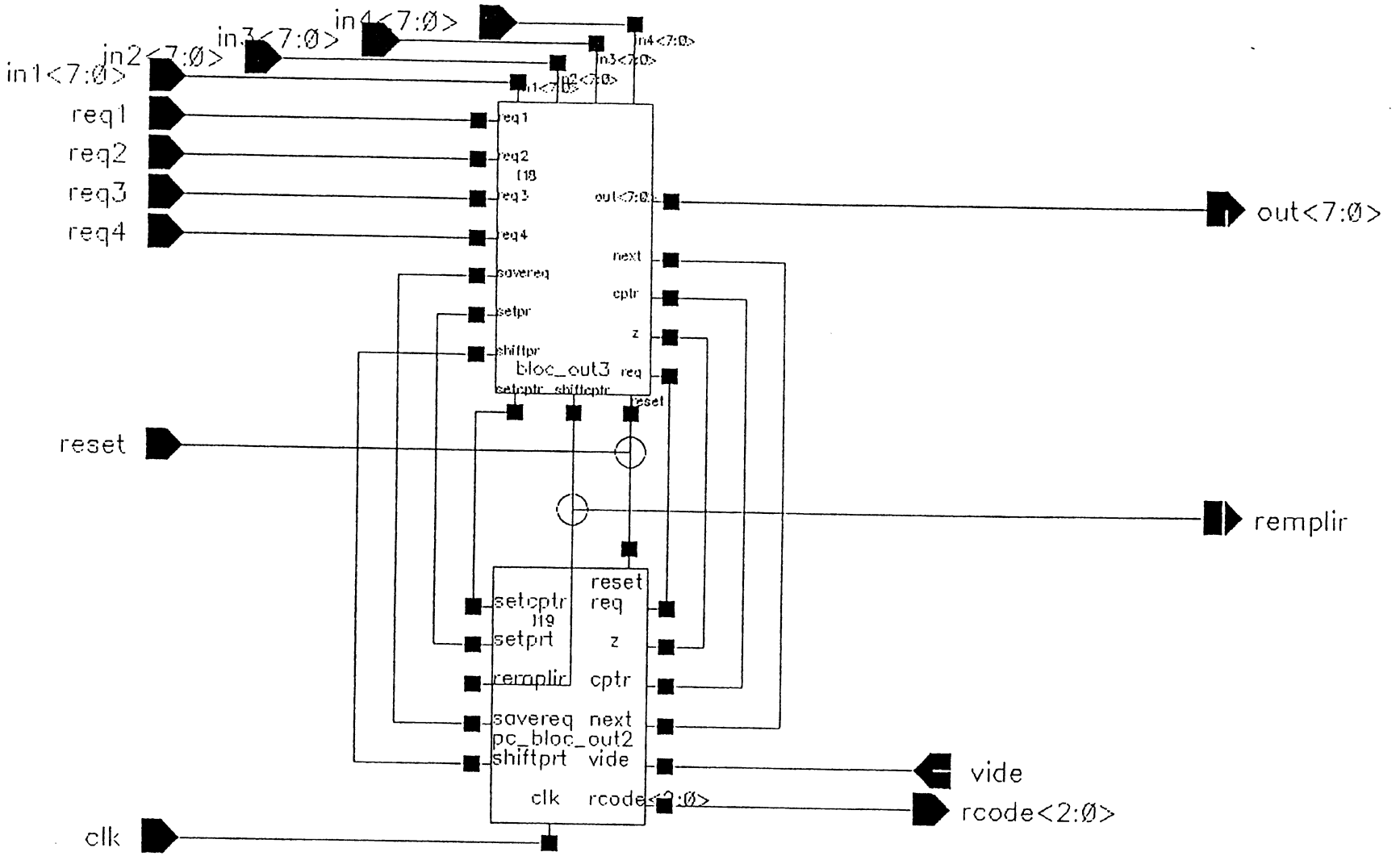

TAMPON
3
Out<7:Ø>
out<7:Ø>
in<7:Ø>
IN<7:Ø>
Sxb
sxb
Syb
syb
reg
Load
Reset
reset
Reset
vide
remplir
2
vide
remp
FlagwR
vider
plein
vider
plein


reset
in<7:0>
sxb
syb
plein
th
tv
clk
D1
D2
D3
D4
data<7:0>
dir<4:1>
cellule
vider
A1
A2
A3
A4
I17
A
Y
BUF39
bloc_in2
pc_bloc_in
ack
Ack
blocin

D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?

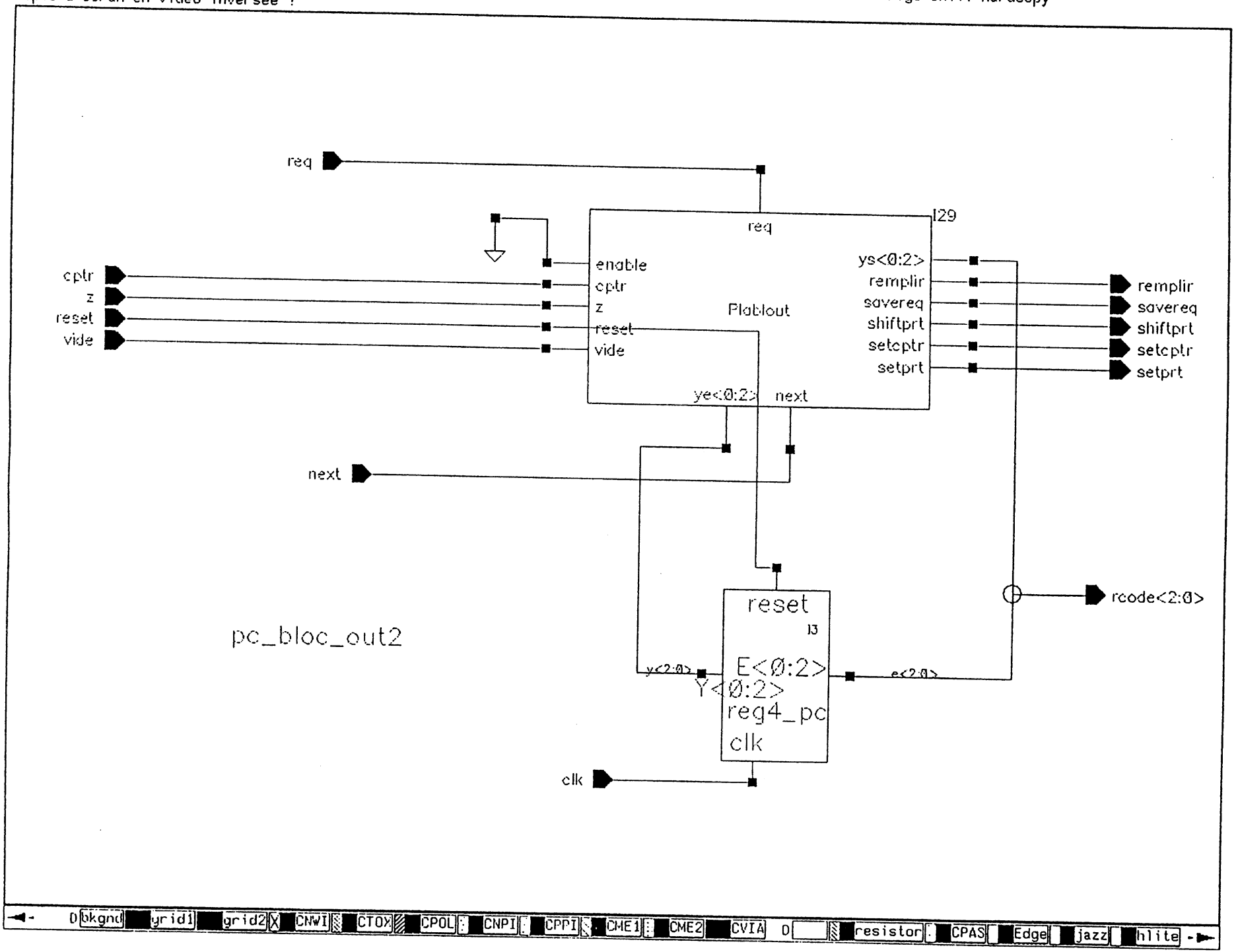





in4<7:0>
in3<7:0>
in2<7:0>
in1<7:0>
req1
req2
req3
req4
reset
clk
out<7:0>
remplir
vide
rcode<2:0>
bloc_out3
setcptr
setptr
remplir
savereq
shiftptr
clk
reset
req
z
cptr
next
vide
rcode<2:0>
pc_bloc_out2

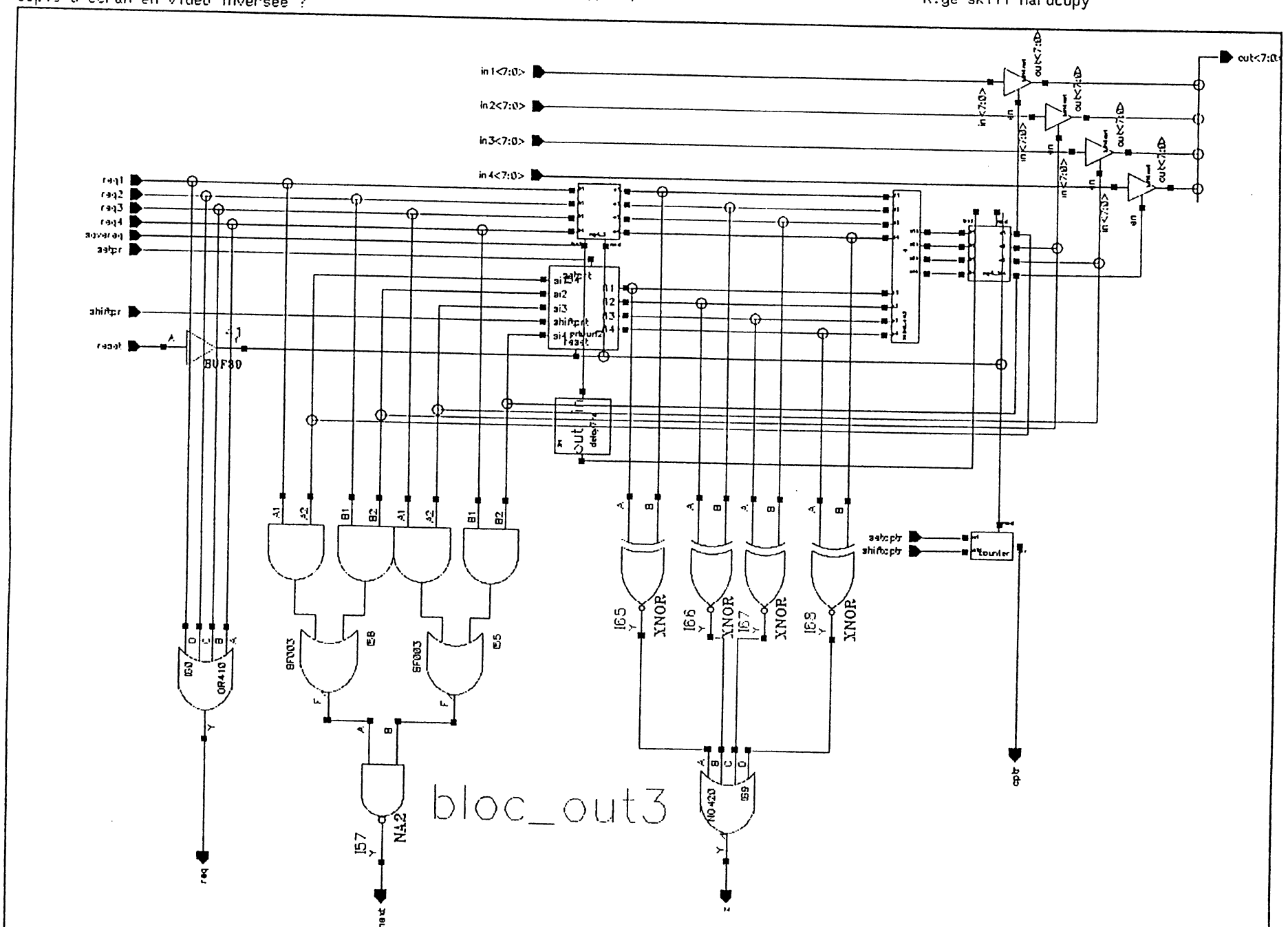
blocout2

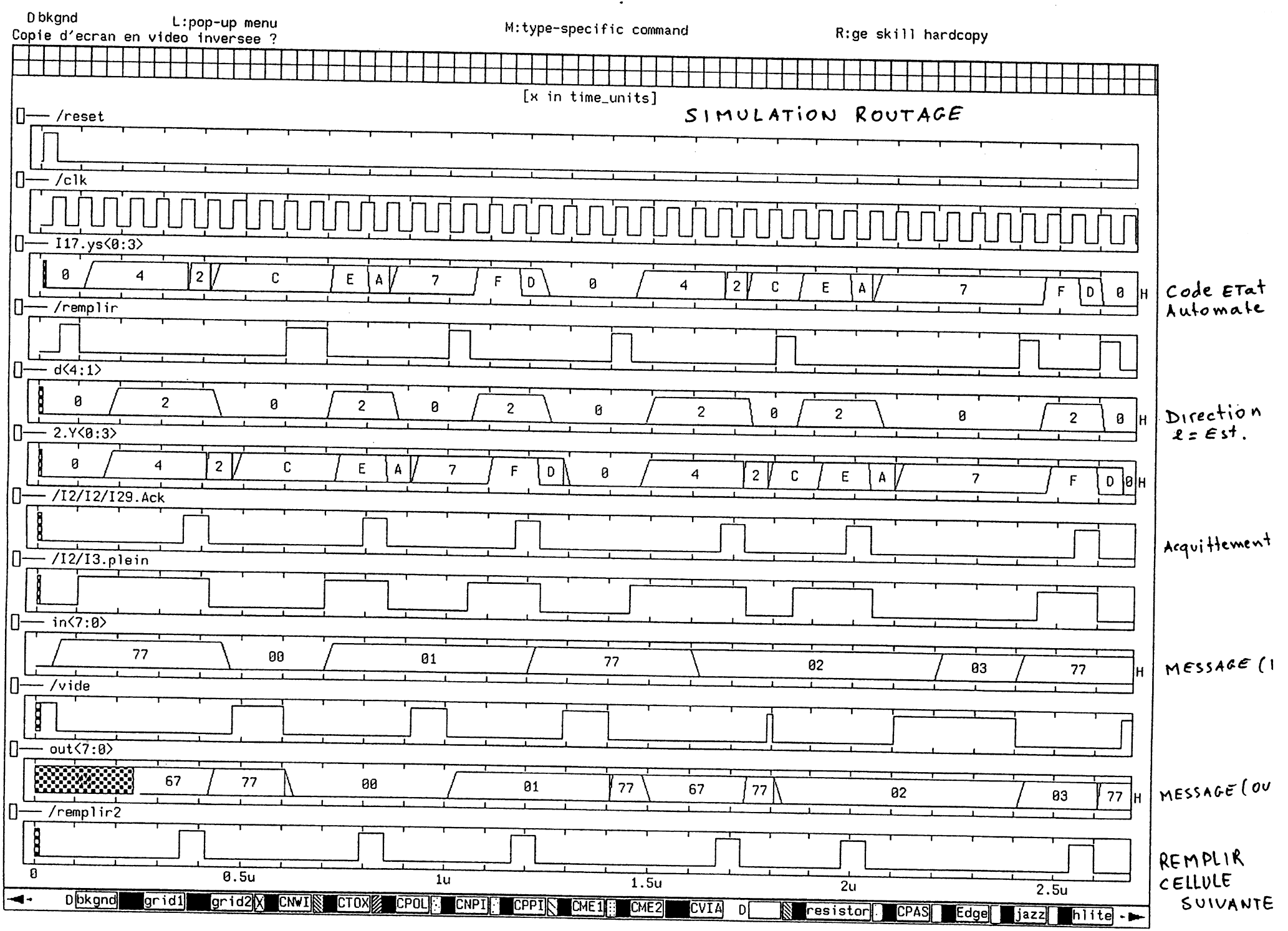
D bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?

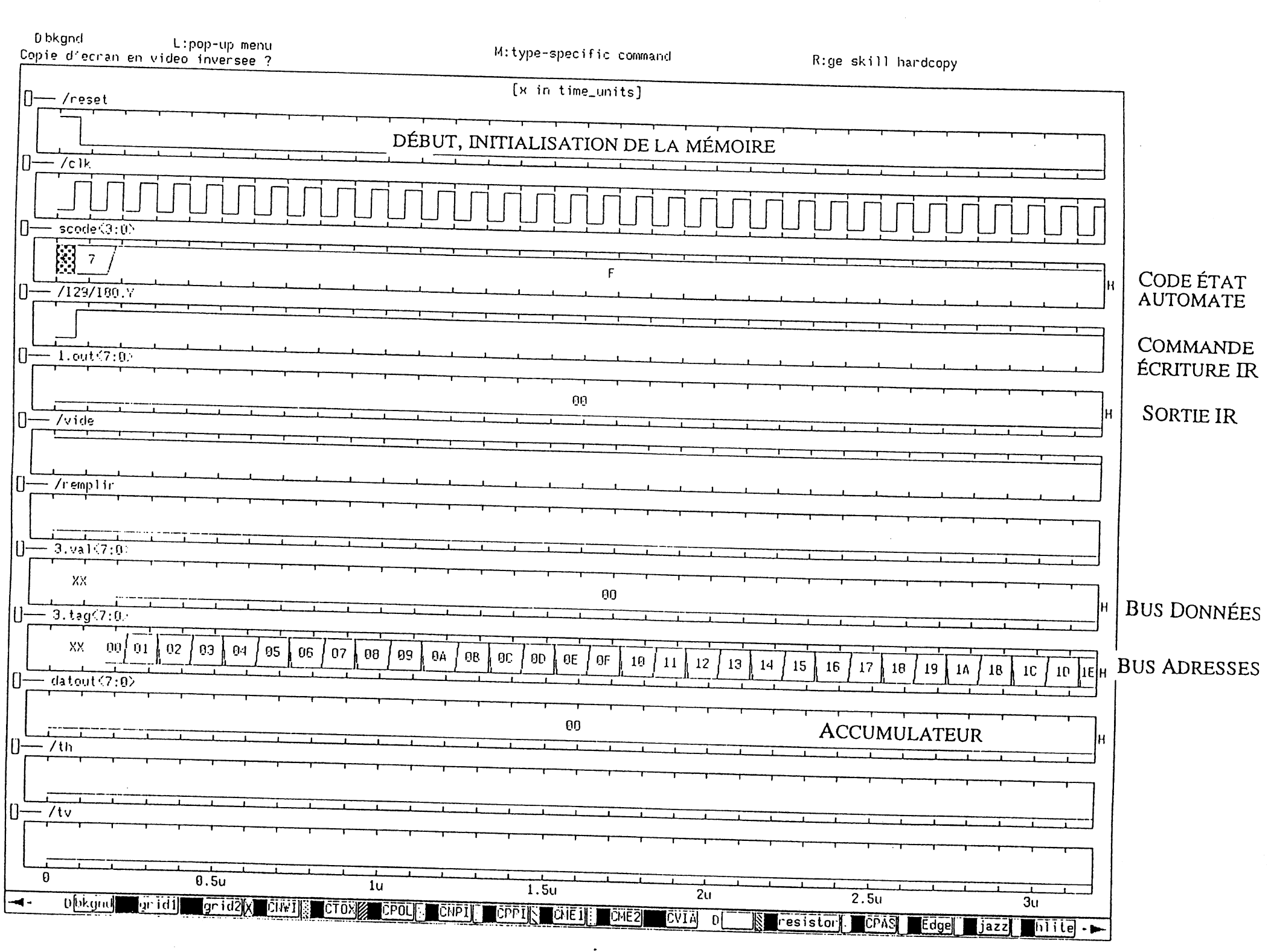


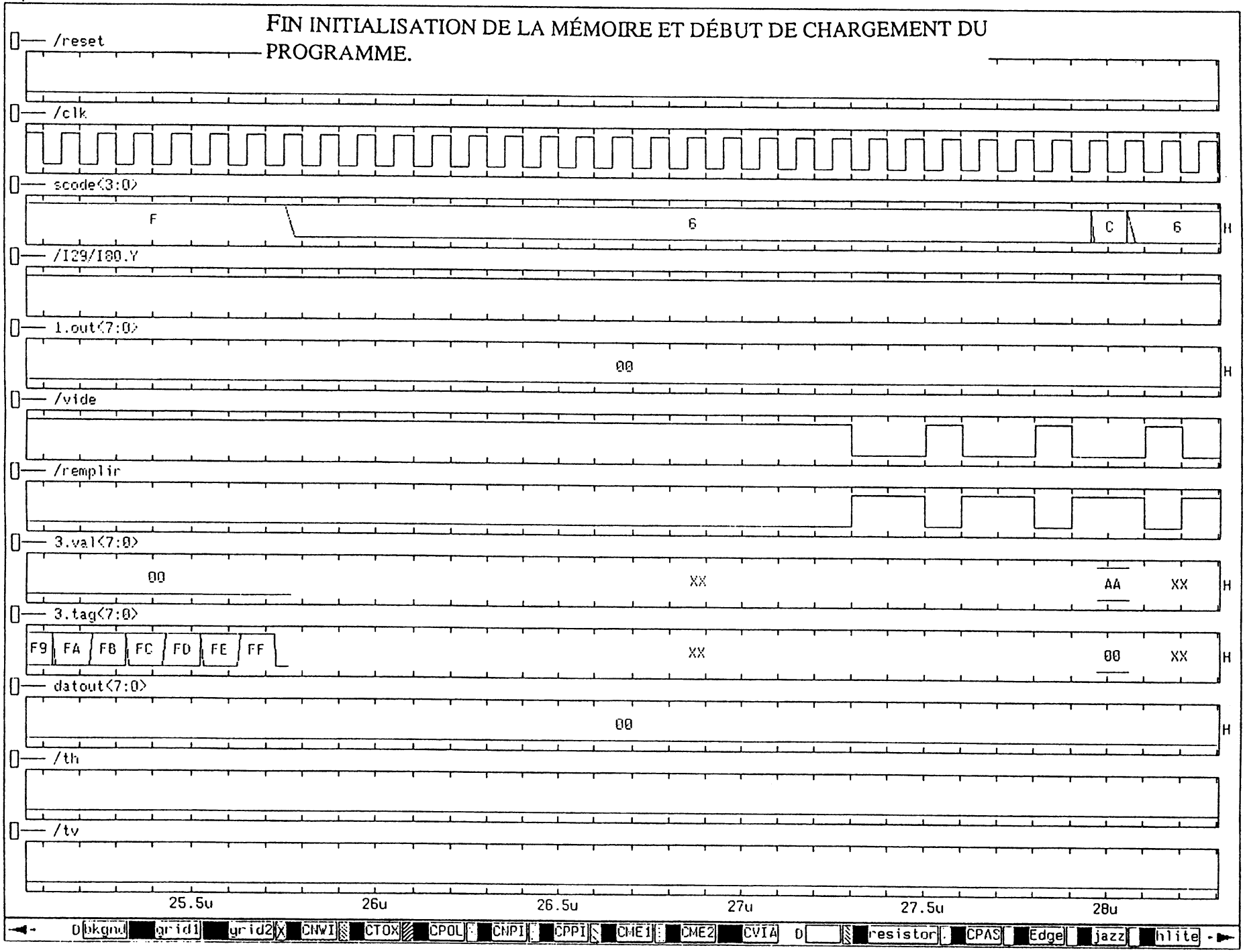
D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy

Copie d'ecran en video inversee ?

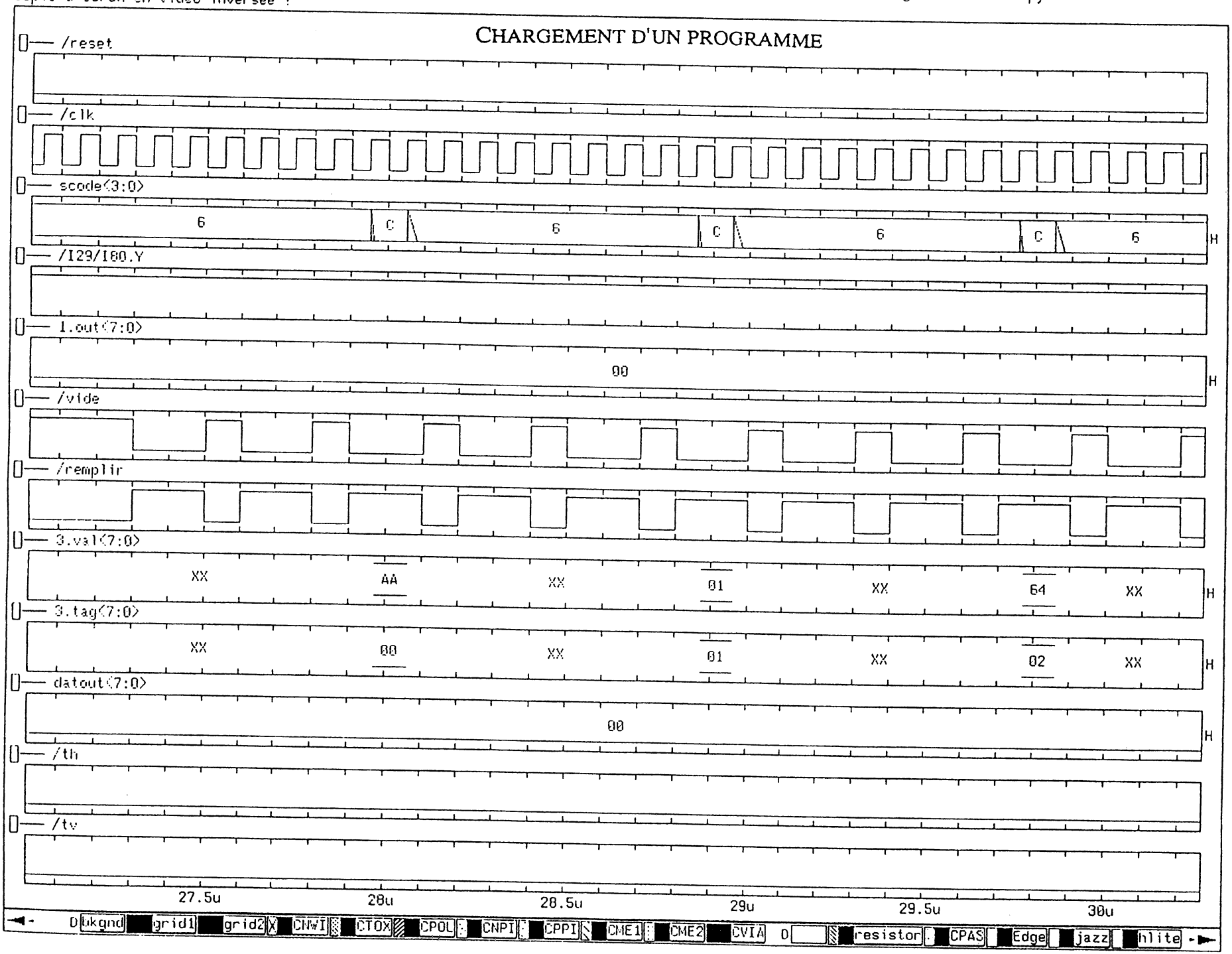


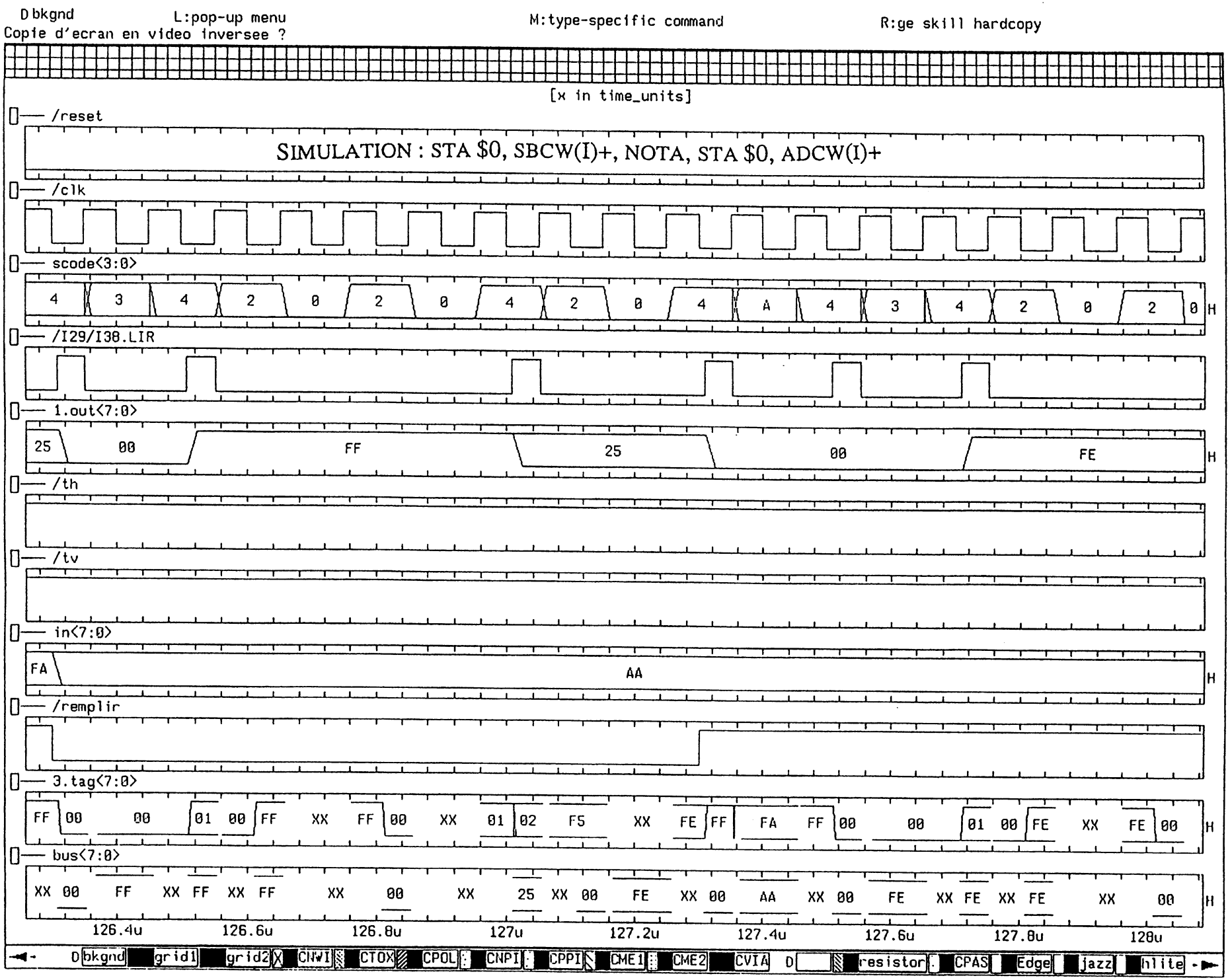

D bkgnd
L:pop-up menu
Copie d'ecran en video inversee ?
M:type-specific command
R:ge skill hardcopy
[x in time_units]
SIMULATION ROUTAGE
/reset
/clk
I17.ys<0:3>
/remplir
d<4:1>
2.Y<0:3>
/I2/I2/I29.Ack
/I2/I3.plein
in<7:0>
/vide
out<7:0>
/remplir2
Code ETat Automate
Direction e = Est.
Acquittement
MESSAGE (I
MESSAGE (OU
REMPLIR CELLULE SUIVANTE
0
0.5u
1u
1.5u
2u
2.5u
D bkgnd grid1 grid2 X CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite


D:bkgnd
L:pop-up menu
Copie d'ecran en video inversee ?
M:type-specific command
R:ge skill hardcopy
[x in time_units]
/reset
DÉBUT, INITIALISATION DE LA MÉMOIRE
/clk
scode<3:0>
7
F
/129/180.Y
1.out<7:0>
00
/vide
/remplir
3.val<7:0>
XX
00
3.tag<7:0>
XX 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E
datout<7:0>
00
ACCUMULATEUR
/th
/tv
0
0.5u
1u
1.5u
2u
2.5u
3u
D:bkgnd
grid1
grid2
CNWI
CTOX
CPOL
CNPI
CPPI
CME1
CME2
CVIA
resistor
CPAS
Edge
jazz
hlile
CODE ÉTAT AUTOMATE
COMMANDE ÉCRITURE IR
SORTIE IR
BUS DONNÉES
BUS ADRESSES



D:bkgnd L:pop-up menu M:type-specific command R:ge skill hardcopy
Copie d'ecran en video inversee ?




D bkgnd
L:pop-up menu
Copie d'ecran en video inversee ?
M:type-specific command
R:ge skill hardcopy
[x in time_units]
/reset
SIMULATION : STA $0, SBCW(I)+, NOTA, STA $0, ADCW(I)+
/clk
scode<3:0>
4 3 4 2 0 2 0 4 2 0 4 A 4 3 4 2 0 2 0 H
CODE ETATS AUTOMATE
/I29/I38.LIR
COMMANDE ECRITURE I
1.out<7:0>
25 00 FF 25 00 FE H
SORTIE I
/th
/tv
in<7:0>
FA AA H
/remplir
3.tag<7:0>
FF 00 00 01 00 FF XX FF 00 XX 01 02 F5 XX FE FF FA FF 00 00 01 00 FE XX FE 00 H
BUS ADRESS
bus<7:0>
XX 00 FF XX FF XX FF XX 00 XX 25 XX 00 FE XX 00 AA XX 00 FE XX FE XX FE XX 00 H
BUS DONNÉ
126.4u
126.6u
126.8u
127u
127.2u
127.4u
127.6u
127.8u
128u
D bkgnd grid1 grid2 X CNWI CTOX CPOL CNPI CPPI CME1 CME2 CVIA D resistor CPAS Edge jazz hlite

AUTORISATION de SOUTENANCE

VU les dispositions de l'Arrêté du 30 mars 1992 relatif aux Etudes doctorales

VU les rapports de présentation de

- Monsieur GARDA Patrick
- Monsieur ROUZEYRE Bruno

Monsieur KARABERNOU Si Mahmoud

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 23 juin 1993

/ BV.

Pour le Président de l'INPG
et par délégation
le Directeur de l'Ecole Doctorale
J.L. LACOUME

INPG

Résumé

Cette thèse présente la conception et la réalisation en VLSI d'un processeur programmable pour une nouvelle architecture MIMD massivement parallèle, intermédiaire entre la Connection Machine et les hypercubes de processeurs 32 bits. Elle est composée d'une grille 2D de cellules asynchrones communiquant par échanges de messages. Chaque cellule intègre une partie de traitement qui consiste en un petit microprocesseur 8 bits doté d'une mémoire (données et programme), et une partie de routage permettant l'acheminement des messages. A l'issue de l'étude des différents problèmes de communication dans les machines parallèles, nous proposons un routeur original utilisant le principe du *wormhole*, et permettant d'acheminer jusqu'à cinq messages en parallèle. Nous décrivons ensuite l'architecture de la partie de traitement, en partant de la définition du jeu d'instructions, du chemin de données et de la partie contrôle jusqu'à la conception au bas niveau. Un premier prototype d'un circuit VLSI de ce processeur a été réalisé sur silicium et a permis d'obtenir les mesures des surfaces et des performances.

Mots clefs : Architecture parallèle, communication, wormhole, jeu d'instructions, chemin de données, contrôle, circuit intégré VLSI.

Abstract

This thesis presents the VLSI design of a programmable processor for a new MIMD massively parallel architecture, halfway between the Connection Machine and hypercubes based on 32-bit processors. It consists of a 2D grid of asynchronous cells which communicate by message transfers. Each cell includes a processing unit based on an 8-bit microprocessor with a small amount of memory (data and code), and a routing part. After introducing various communication problems in parallel computers, we propose an original wormhole based routing system, able to forward up to five messages in parallel. We describe also the processing unit architecture starting from the instruction set definition and the data path and control part architecture down to the low level design. A first VLSI circuit prototype realisation of the processor has been performed and measures of silicon area and performances have been obtained.

Key words : Parallel architecture, communication, wormhole, instruction set, data path, control, VLSI integrated circuit.