MOSAIC : une méthodologie de conception pour les circuits système VLSI

José Manuel Costa Alves Marques

▶ **To cite this version:**

José Manuel Costa Alves Marques. MOSAIC : une méthodologie de conception pour les circuits système VLSI. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1980. Français. NNT : . tel-00292681

HAL Id: tel-00292681

https://theses.hal.science/tel-00292681

Submitted on 2 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR INGENIEUR
" Informatique "

par

José Manuel
COSTA ALVES MARQUES

MOSAIC :
UNE METHODOLOGIE DE CONCEPTION POUR LES CIRCUITS SYSTEME VLSI

Thèse soutenue le 24 septembre 1980 devant la Commission d'Examen :

Monsieur	L. BOLLIET	Président
Messieurs	F. ANCEAU J. BOREL V. CORDONNIER P. JESPERS	Examinateurs

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

Vice-Présidents : M. Georges LESPINARD

M. René PAUTHENET

PROFESSEURS DES UNIVERSITES

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

.../...

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

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

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

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

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

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

.../...

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

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

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

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

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

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

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

E.N.S.H.G.

M. DELHAYE Jean-Marc

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

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

Je tiens à remercier,

Monsieur L.BOLLIET, Professeur à l'Université de Grenoble, pour l'honneur qu'il me fait en acceptant de présider le jury de cette thèse ;

Monsieur F.ANCEAU, Professeur à l'Institut National Polytechnique de Grenoble, pour ses conseils et ses encouragements tout au long de cette étude ;

Monsieur J.BOREL, Directeur du Département de Recherche et Développement de EFCIS, pour l'attention et le soutien qu'il a accordés à ce travail ;

Monsieur V.CORDONNIER, Professeur à l'Université des Sciences et Techniques de Lille, dont les remarques pertinentes m'ont aidé à améliorer la rédaction du document ;

Monsieur P.JESPERS, Professeur à l'Université Catholoqie de Louvain-la-Neuve, qui a voulu bien juger ce travail ;

les membres de l'équipe d'Architecture des ordinateurs de l'ENSIMAG avec qui j'ai pu discuter nombre de points de cette étude ;

Madame DIAZ et Madame CHALAND pour la dactylographie rapide et soignée ;

et enfin le Service de reprographie de l'IMAG qui a assuré le tirage de ce document.

SOMMAIRE

Chapitre 1 - INTRODUCTION ---------- 1

1.1. Projet MOSAIC ---------- 4

1.1.1. Objectifs ---------- 4

1.1.2. Principaux problèmes ---------- 5

1.1.3. Démarche utilisée ---------- 16

1.2. Architecture de base ---------- 19

Chapitre 2 - FORMALISME SYSTEME ---------- 25

2.1. Choix d'un formalisme ---------- 25

2.2. Système MOSAIC ---------- 27

2.2.1. Modules ---------- 27

2.2.2. Moniteurs externes ---------- 29

2.3. Découpage fonctionnel d'une application ---------- 33

2.4. Langage de spécification ---------- 41

2.5. Moniteurs externes ayant des structures plus complexes ------ 48

Chapitre 3 - ARCHITECTURE GLOBALE

3.1. Critères de choix dans une approche descendante ---------- 57

3.2. Hypothèses d'architecture ---------- 60

3.2.1. Architecture monoprocesseur ---------- 60

3.2.2. Multiprocesseur à mémoire commune ---------- 62

3.2.3. Processeur noyau ---------- 65

3.2.4. Hiérarchie des variables de communication ---------- 70

3.2.5. Solution distribuée ---------- 71

3.3. Architecture MOSAIC ---------- 74

3.4. Evaluation des performances ---------- 76

Chapître 4 - PROTOCOLE D'UTILISATION DU MECANISME DE COMMUNICATION ------- 78
4.1. Allocation ---------- 83
4.2. Appel d'une procédure d'un moniteur externe ---------- 89
4.3. Exécution des procédures ---------- 90
4.4. Mécanismes de synchronisation ---------- 94
4.5. Exceptions et erreurs ---------- 98
4.6. Multiplexage temporel ---------- 101

Chapître 5 - PROCESSEUR NOYAU ---------- 103
5.1. Représentation interne des moniteurs ---------- 104
5.1.1. Descripteur ---------- 104
5.1.2. Catalogue ---------- 106
5.2. Processeur ---------- 107
5.2.1. Architecture globale ---------- 107
5.2.2. Partie opérative ---------- 112
5.2.3. Partie contrôle ---------- 132
5.2.4. Adaptation pour une gestion en barillet ---------- 152

Chapître 6 - EVALUATION DE LA SURFACE ET PERFORMANCES DU PROCESSEUR NOYAU- 155
6.1. Evaluation de la surface ---------- 156
6.1.1. Méthode utilisée ---------- 157
6.1.2. Mémoire de communication ---------- 157
6.1.3. Partie opérative ---------- 158
6.1.4. Partie contrôle ---------- 163
6.1.5. Evaluation de la surface en fonction de la technologie ---------- 167
6.2. Performances ---------- 168
6.2.1. Temps de cycle du processeur noyau ---------- 168
6.2.2. Temps de propagation bus ---------- 169
6.3. Modèles de simulation ---------- 170

Chapitre 7 - HIERARCHIE DES VARIABLES DE COMMUNICATION ---------- 180
7.1. Décomposition d'une application ---------- 181
7.2. Extension du langage de spécification ---------- 184
7.3. Implémentation des moniteurs globaux ---------- 184
7.4. Protocole d'utilisation des bus ---------- 186
7.5. Limitations de cette approche ---------- 190

Chapitre 8 - SOLUTION DISTRIBUEE ---------- 192
8.1. Exécution locale des moniteurs ---------- 193
8.2. Protocole d'utilisation du bus ---------- 195
8.3. Structure interne des automates ---------- 198
8.4. Modèle de simulation ---------- 207

Chapitre 9 - MODULES ---------- 210
9.1. Coeur des modules ---------- 212
9.2. Interface bus ---------- 213
9.3. Entrées/Sorties ---------- 229
9.4. Topologie des modules ---------- 231
9.5. Module de test ---------- 232

Chapitre 10 - CONCLUSION ---------- 236
10.1. Etapes de la méthode ---------- 237
10.2. Remarques finales ---------- 240

ANNEXE 1 - Exemple de moniteurs externes
ANNEXE 2 - Langage intermédiaire
ANNEXE 3 - Evaluation de surface
ANNEXE 4 - Simulation de performances
ANNEXE 5 - Exemples de modules
ANNEXE 6 - Exemples d'applications

REFERENCES & BIBLIOGRAPHIE

[ADA.79] CII.HONEYWELL BULL
Rationale for the design of ADA Programming language
March 79.

[ALE.78] N.ALEXANDRIDIS
Bit sliced microprocessor architecture
Computer, June 78

[ALT.78] L.ALTMAN
Five technologies seequizing more performance from LSI chips
Electronics, 18/8/79

[ANC.79] F.ANCEAU
L'évolution des microprocesseurs et leur influence sur
l'évolution des architectures des systèmes
Real time data, Octobre 1979

[ANC.80] F.ANCEAU, R.ETIENNE
Conception des parties contrôle des microprocesseurs
Rapport interne IMAG.

[BRA.78] J.P.BRAUN
Conception et réalisation d'un circuit intégré à très
grande échelle d'intégration en technologie CMOS/SOS
Thèse CNAM, Juin 1978

[BOR.76] J.BOREL
Advanced MOSFET technologies: a review
Proceedings of the European solid state circuits Conf., Toulouse 1976

[BOR.80] J.BOREL
L'évolution des circuits intégrés
Bulletin de liaison INRIA N°60, 1980.

[BUS.77] D.W.BUSTARD
An introduction to PASCAL PLUS
The Queen's University of Belfast, 1977

[CAM.74] R.H.CAMPBELL, A.N.HABERMAN
The specification of process synchronization by path expressions
Lecture Notes in Computer science, vol.16, Springer Verlag

[CAP.78] R.P.CAPECE
Tackling the very large scale problems of VLSI
Electronics, November 1978

[CAP.79] R.P.CAPECE
Older processes revamped as new arrivals extend performance limits
Electronics, 13/9/79.

[COU.78] B.COURTOIS
Mécanisme de panne, hypothèse de panne et test analytique de circuits LSI.NMOS
Rapport de recherche n°196 IMAG

[CRO.78] Y.CROUZET
Conception de circuits à large échelle d'intégration totalement autotestable
Thèse de docteur-ingénieur, INP Toulouse, novembre 1978

[DIJ.68] E.W.DIJKSTRA
The structure of the "THE" multiprogramming system
CACM Mai 1968

[DIJ.75] E.W.DIJKSTRA
Guarded commands, nondterminacy and formal derivations of programs
CACM, août 1975

[FAG.79] F.FAGGIN
Trends in microcomputers
Euromicro Journal, n°5, 1979.

[FEL.76] J.A.FELDMAN
A programming methodology for distributed computing
TR9 University of Rochester, November 1976

[FIS.80] J.L.FISCHER
Programmable components: the shape of the VLSI to come
Electronics, 5/6/1980

[GAL.78] J.GALIAY
Conception de circuits à large échelle d'intégration facilement testables
Thèse de 3° cycle, Université Paul Sabatier, Toulouse, Nov. 1978

[GRA.78] EFCIS - Département d'informatique 1978
Manuel de référence GRANIT

[GUI.79] J.M.GUILLOT
Proposition pour un langage d'écriture de programmes répartis. Expression du contrôle et de la communication entre processus distribués
Thèse de 3° cycle, INP Grenoble, décembre 1979

[HAR.79] B.HARTMAN
16 bits 68000 microcomputer camps on 32 bits frontier
Electronics 11/10/1979.

[HAN.73] P.BRINCH HANSEN
Structured multiprogramming
CACM, July 1972

[HAN.75] P.BRINCH HANSEN
The programming language concurrent PASCAL
IEEE Trans. on software engineering, June 1975

[HAN.77] P.BRINCH HANSEN
The architecture of concurrent programs
Prentice Hall 1977.

[HAN.78] P.BRINCH HANSEN
Distributed processes: a concurrent programming concept
CACM, November 1978

[HEN.70] HENRY, CRINON, LACOUR
Conception de circuits intégrés à transistors MOS
par la méthode de l'espalier
Col. Int. sur la microélectronique avancée, Paris, avril 1970

[HEN.78] M.HENRY
Spécification pour l'unité de visualisation
Rapport interne EFCIS, octobre 1978

[HEN.79] M.HENRY
Spécification du sélecteur de voie
Rapport interne EFCIS, janvier 1979

[HOA.74] C.A.R. HOARE
Monitors, an operating system structure concept
CACM, october 1974

[HOA.78] C.A.R. HOARE
Communicating sequencial processes
CACM, August 1978

[JAM.77] A.J. JAMMEL, H.G.STIEGLER
Managers versus monitors
IFIP 1977

[JEC.79] R.M.JECMEN, A.EBEL, R.SMITH, V.KYNETT, C.HUI, R.PARSHLEY
A 25ns static RAM
Proceedings of Int. Solid state circuits Conf., 1979

[KEE.78] J.L.KEEDY
On structuring operating systems with monitors
Australian Computer journal, February 1978

[KES.77] J.L.KESSELS
An alternative to event queues for synchronization in monitors
CACM, July 1977

[LAM.80] B.W.LAMPSON, D.D.REDELL
Experience with processes and monitors in MESA
CACM, February 1980

[LAU.78] H.LAUER, R.N.NEEDHAM
On the duality of operating systems
2° Int.Symp. on operating systems, IRIA, octobre 1978

[LAS.79] C.LASSEN
Wanted: a new interconnection technology
Electronics, 27/9/79

[LAT.79] B.LATTIN
VLSI design methodology , the problem of the 80's for microcomputer design
16th Digital automation Conf. 1979

[LIS.76] A.LISTER, K.MAYNARD
An implementation of monitors
Software Practice and experience, vol. 6.76

[LIS.77] A.LISTER, P.Y.SAYER
Hierarchical monitors
Software practice and experience, vol. 7.77

[LEE.79] J.LEE, J.BREIVAGEL, R.KUNITA, C.WEBB
A 80ns 5V only dynamic RAM
Proceedings of the Int. Solid state circuit Conf., 1979

[LOU.79] K.Y.LOOSEMORE
I.C. Layout the automatic approach
European solid state circuit Conf., London, 1979

[MAR.78] M.MARINESCU
Mécanisme de communication par bus série pour les réseaux informatiques locaux
Thèse INPG, 1978

[MAR.80] M.MARINESCU
Test fonctionnel de mémoire vive à large couverture de pannes
RR. n°191 IMAG

[MAR.78] J.M.C. ALVES MARQUES
Conception d'un microordinateur CMOS/SOS
Rapport de DEA, IMAG, juin 1978

[MAR.80] J.M.C. ALVES MARQUES
A multiprocessor architecture adapted to VLSI custom design
Euromicro Symp., London 1980

[MAR.75] B.MARTI, M.MAUDUIT
ANTIOPE, service de teletexte
Revue de radiodiffusion télévision n°40, 1975

[MEA.79] C.MEAD, L.CONWAY
Introduction to VLSI systems
Addison Wesley, 1979

[MER.78] D.MERLE, M.VERAN
A tool for computer system performance analysis
Int. Conf. on performance of comp. installations, 1978

[MOO.79] G.MOORE
Are we ready for VLSI ?
Proceedings Int. Solid state circuit Conf. 1979

[NOI.75] Y.NOIREL
Un système expérimental de diffusion de données par paquet
Revue de radiodiffusion et télévision, n°40, 1975

[OAK.79] M.F.OAKES
The complete VLSI design system
Proceedings of the 16th DAC, 1979

[PAR.72] D.L.PARNAS
On the criteria to be used in decomposing systems into modules
CACM, July 1972

[PER.80] T.PEREZ SEGOVIA
L'optimisation en surface des PLAs
Rapport de DEA, ENSIMAG 1980

[PEU.79] B.L.PEUTO
Architecture of a new microprocessor
Computer, February 1979

[PRE.78] B.T.PREAS, C.W.GWYN
Methods for hierarchical automatic layout of custom
VLSI circuit masks
Proceedings 15th DAC, 1978

[QUE.79] D.QUEYNAC
Projecting VLSI impact on microprocessors
IEEE Spectrum, May 1979

[ROQ.79] B.ROQUE, J.ALVES MARQUES
ETude d'une architecture multiprocesseur adaptée à la VLSI
à la demande
EFCIS, contrat DGRST n°78 72 103

[STA.77] D.STAUM, D.BUDDLE, B.MORGAN
A single chip, highly integrated, user programmable microcomputer
Proceedings of the Int. Solid state circuits Conf., 1977

[SCH.77] J.P.SCHOELLKOPF
Machine PASCHLL: définition d'une architecture pipe-line pour une
unité de contrôle adaptée au langage PASCAL
Thèse de 3° cycle INPG, 1977

[SCH.77] P.SCHROEDER, R.PROEBSTING
A 16 x 1 bit dynamic RAM
Int. Solid state circuit Conf. 1977

[SHI.79] M.SHIMA
Demystifying microprocessor design
IEEE spectrum, July 1979

[STO.75] STONE
Introduction to computer architecture
Science research associates, 1975

[STR.79] E.STREITTER, T.GUNTER
A microprocessor architecture for a changing world: the MOTOROLA 68000
Computer, February 1979

[SUT.79] I.SUTHERLAND, C.MOLNAR, R.SPROULL, C.MUDGE
The trimobus
Proceedings of the Caltech. Conf. on the VLSI, 1979

[SUZ.79] A.SUZIM
Parties opératives à éléments modulaires
Rapport de DEA, ENSIMAG 1979

[WAK.79] Y.F.WAKELY
The INTEL MCS.48 microcomputer family: a critique
Computer, February 1979

[WIL.77] J.D.WILLIAMS
Sticks a new approach to LSI design
Thèse de Msc Massachusetts Institute of technology, June 1977

[WIR.77] N.WIRTH
MODULA, a programming language for multiprogramming
Software practice and experience, January 1977

CHAPITRE 1

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

La microélectronique a été un des domaines de la technologie où l'évolution a été une des plus spectaculaires pendant la dernière décennie.

Les progrés enregistrés dans la technologie de fabrication des circuits intégrés ont permis d'augmenter d'une façon presque linéaire les densités d'intégration.

La structure interne des circuits est donc devenue de plus en plus complexe, passant d'un ensemble de quelques transistors implémentant une fonction booléenne à des structures équivalentes à celles des unités centrales des miniordinateurs du début des années 70.

Les circuits intégrés digitaux se sont donc progressivement transformés en des machines informatiques puissantes, ce qui oblige à élargir le spectre des connaissances requises pour leur conception à des domaines tels que l'architecture, les systèmes d'exploitation et les langages qui étaient avant réservés à la conception des ordinateurs classiques.

Tous les indices industriels prévoient que l'accroissement des densités d'intégration et la diminution des temps de commutation des portes élémentaires se poursuivra d'une façon continue au moins jusqu'à la fin des années 80 [QUE.79].

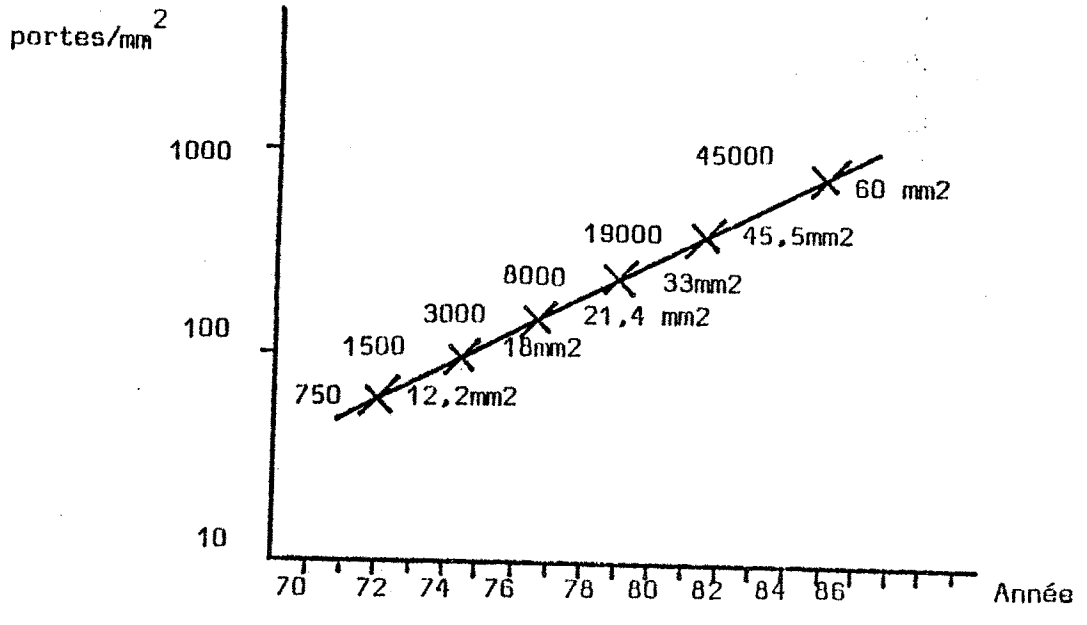


Fig. 1.1. - Evolution de la complexité des circuits
[FAG.79]

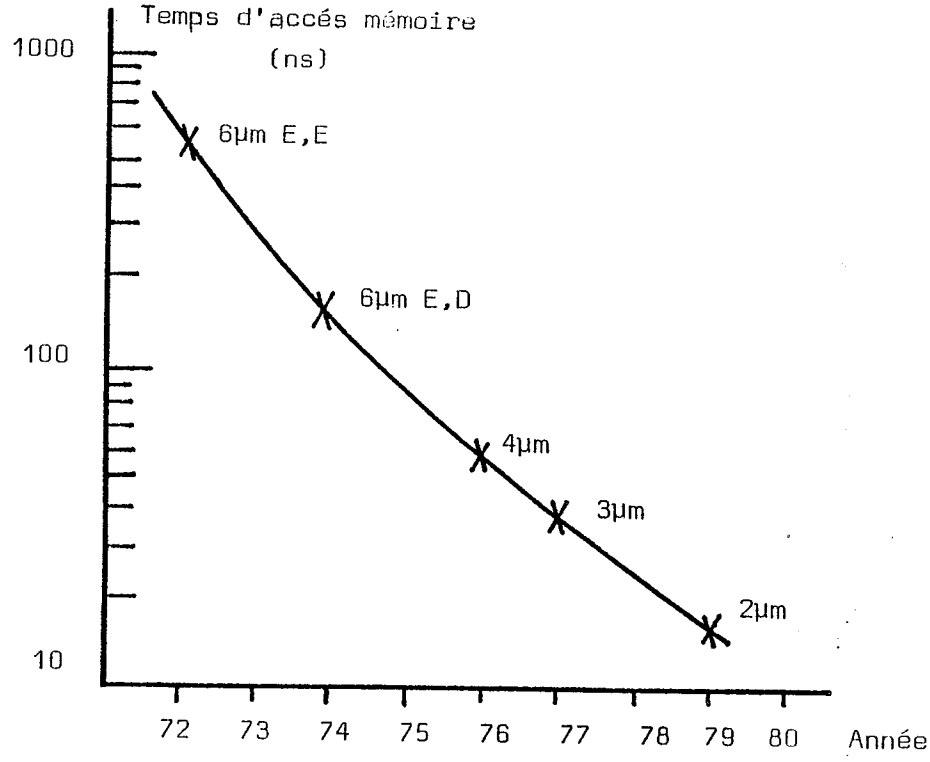


Fig. 1.2 - Evolution des temps de réponse des RAM 1Kbit [JEC.79]

L'évolution vers des produits plus performants et plus complexes permettra d'intégrer dans un seul circuit des microordinateurs de plus en plus puissants et même des systèmes informatiques complets.

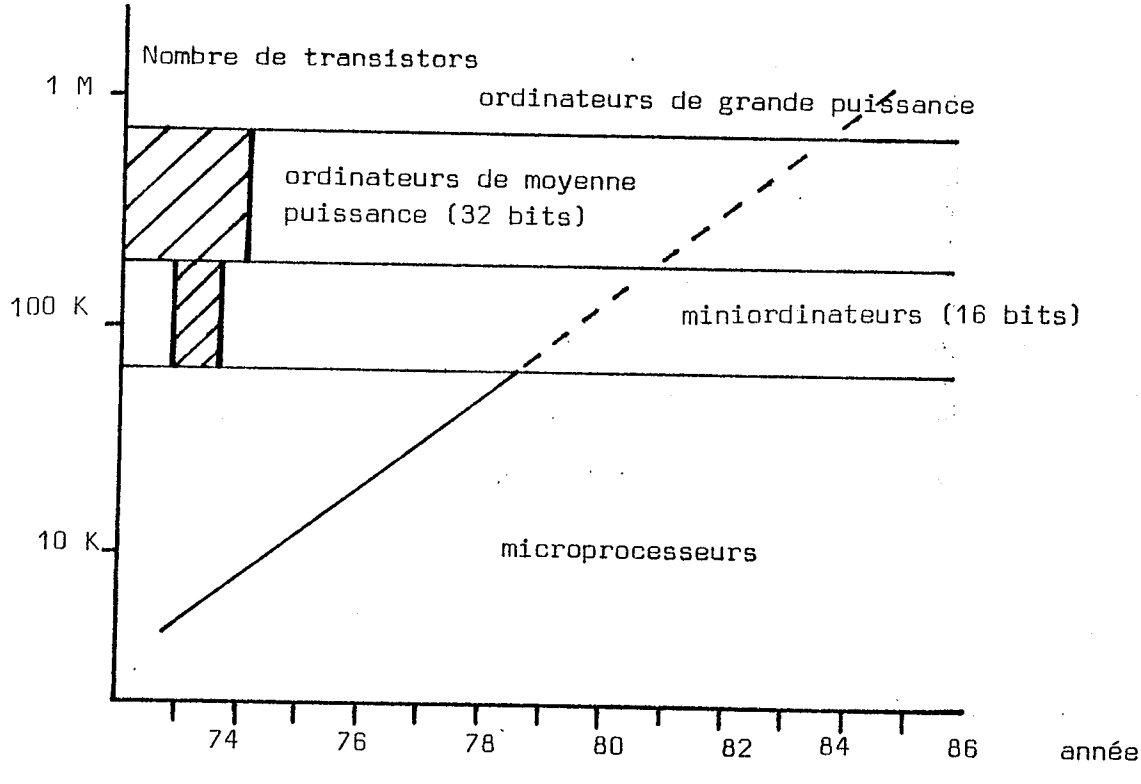


Fig.1.3. - Réalisation monolithique [AN.79]

Il est évident que cette évolution est limitée par les lois physiques qui sont le support même de la théorie de fonctionnement du transistor MOS [MEA.79]. Cependant ce ne sont pas les limites physiques, encore lointaines, qui posent des problèmes aux fabricants de circuits intégrés mais une autre contrainte beaucoup plus tangible, la complexité des circuits à réaliser avec la technologie déjà (ou très bientôt) disponible.

L'accroissement des densités d'intégration de presque 1,6 par an pose de nombreux problèmes à tous les stades de la conception et tend à augmenter d'une façon extrêmement importante la durée du développement d'un circuit.

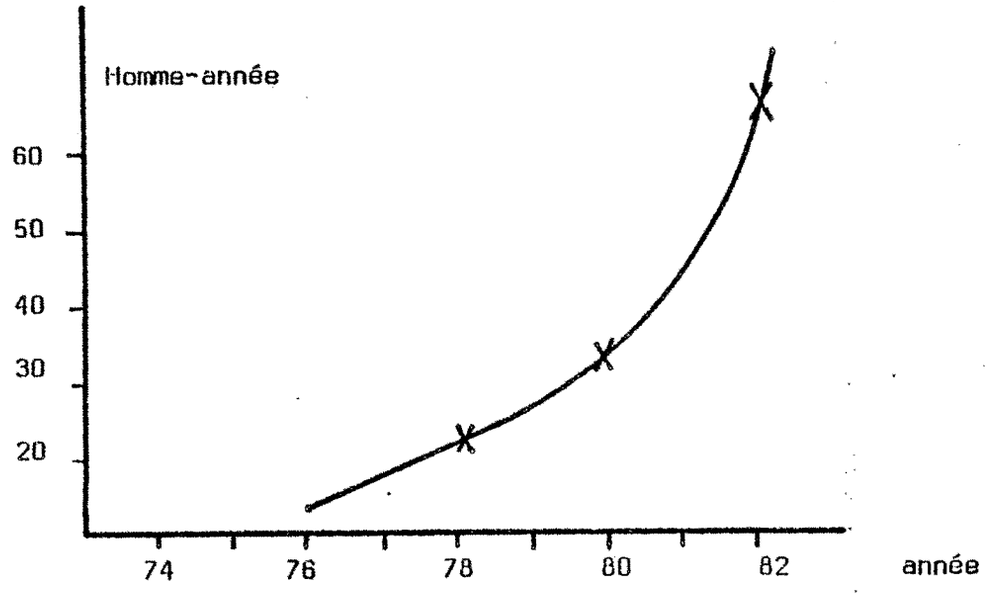


Fig. 1.4 - Années x homme nécessaires pour concevoir des circuits en logique non-répétitive [LAT.79]

Pour pouvoir exploiter les potentialités de l'évolution technologique il est donc nécessaire de définir de nouvelles méthodes de conception adaptées à la complexité des circuits à produire et tenant compte des contraintes économiques propres à l'industrie des semi-conducteurs.

1.1. PROJET MOSAIC

1.1.1. Objectifs

L'objectif du projet MOSAIC (Modular System Architecture for Integrated Circuits) est la définition d'une méthodologie de conception pour une famille particulière de circuits que nous avons appelés "circuits systèmes".

Circuit système

La désignation "système" correspond pour nous à l'intégration dans un seul circuit d'un ensemble de fonctions de calcul et de fonctions d'E/S permettant d'implémenter un système informatique complet dans un seul boîtier.

Ces circuits, dont la complexité dépassera les 100 000 MOS, peuvent être considérés comme l'étape suivante de l'évolution après les microordinateurs monoboîtiers qui sont déjà des systèmes informatiques quoique encore rudimentaires car ils ne disposent ni de fonctions d'E/S évoluées, ni de fonctions de calcul spécialisées.

Le qualificatif de circuit système peut donc s'appliquer à une large famille de circuits, composés d'une ou de plusieurs unités de traitement (avec leurs ROM et RAM respectives) et par des unités spécialisées soit dans le contrôle d'E/S, soit dans l'exécution de fonctions très particulières (ex.: calcul mathématique, conversion analogique/digitale).

Le marché potentiel de ces circuits est de se substituer aux petits systèmes informatiques sur une carte (système à base de microprocesseurs ayant une dizaine de boîtiers) utilisés dans des équipements produits dans des séries importantes comme par exemple des controleurs des organes périphériques, des controleurs de machines outils, des systèmes de télécommunication, etc...

Les avantages pour l'utilisateur d'une solution mono-boîtier du point de vue

facilité d'emploi et fiabilité pourront amortir le coût du développement de ces systèmes dans une version intégrée, si des méthodes de conception moins onéreuses que les actuelles sont utilisées.

1.1.2. Principaux problèmes

Parmi les différents problèmes qui se posent à la définition d'une méthode de conception, pour des circuits complexes, il nous semble qu'il y en a trois qui sont particulièrement prioritaires :

- la réduction du temps de conception,
- l'adaptation d'un circuit à une application,
- le fonctionnement asynchrone des différents éléments du circuit dû aux déphasages introduits par les lignes de distribution de l'horloge.

1. Réduction du temps de conception

La réduction du temps de conception des circuits très complexes (VLSI) n'est pas un problème spécifique des circuits systèmes.

L'accroissement continu des densités d'intégration tend à augmenter d'une façon presque exponentielle le temps de conception et de mise au point des circuits.

Cette augmentation résulte d'une intensification des problèmes à résoudre dans toutes les étapes du processus de conception et de test :

a/ Définition du circuit et de l'architecture:

La complexité des circuits qui peuvent être produits avec la technologie déjà disponible ($\sim$ 50 000 transistors) oblige à prendre en compte, dans leur définition, les caractéristiques des systèmes dans lesquels ils seront insérés.

La phase initiale de la conception devient donc très importante car d'une part elle conditionne fortement le succés commercial du circuit et d'autre part l'éventail de connaissances requises est de plus en plus vaste.

Les derniers microprocesseurs 16 bits [PEU.78, STE.78] sont un exemple de ce qui vient d'être dit. L'architecture de ces systèmes montre bien que l'objectif a été de concevoir des produits ayant des spécifications complètes dès le départ, même si certaines de leurs possibilités ne sont intégrables [HAR.79] que dans une phase ultérieure de l'évolution de la technologie.

Il est évident que même pour les circuits actuellement en développement un seul concepteur ou une petite équipe, ne peuvent pas étudier globalement une application et ils sont obligés d'effectuer un découpage du problème en blocs plus simples de façon à pouvoir appréhender la totalité des fonctions à exécuter par le circuit.

Sur ce dernier point l'analogie a été souvent faite avec ce qui s'est passé dans le domaine des grands programmes au début des années 70.

De la même façon, une structuration et si possible des méthodes de conception modulaires doivent être mises en place de façon à simplifier la définition de l'architecture et les étapes suivantes d'implantation et de test.

Cependant l'augmentation de la densité d'intégration comporte aussi des aspects favorables au niveau de la définition de l'architecture.

La contrainte imposée par la surface totale du circuit, qui conditionnait fortement les solutions d'architecture, est considérablement atténuée.

Le concepteur a donc plus de liberté dans la définition interne du circuit et il peut notamment utiliser des techniques de conception plus systématiques, quelque fois bien connues dans le domaine de l'architecture des machines informatiques discrètes et dont l'usage n'était pas intéressant dans

la conception de circuits intégrés à cause de l'augmentation de surface qu'elles entraîneraient par rapport à l'utilisation de logique non-répétitive (ex.: microprogrammation, UAL style "bit-slice" avec des registres à double accés, etc...).

b/ Implantation

L'implantation a été considérée depuis le début de la fabrication des circuits intégrés comme l'étape, du processus de conception, la plus lente et la moins fiable. Ceci a poussé les fabricants de circuits à développer des systèmes de CAO graphiques de façon à diminuer le temps de dessin des différents niveaux de masques du circuit. La plupart de ces systèmes sont des éditeurs graphiques puissants qui permettent de coder une figure, précédemment dessinée, en utilisant un langage graphique ou un digitaliseur.

Normallement on dispose aussi d'un ensemble de primitives qui permettent d'effectuer des manipulations graphiques sur les figures déjà codifiées (rotation, symétrie, matriçage).

Ces systèmes ne simplifient donc que le codage et l'édition et n'éliminent pas la phase de dessin manuel des masques.

La tâche de détecter les nombreuses erreurs introduites dans les phases de dessin et de codification est laissée à des programmes de vérifications de gardes qui permettent de signaler certaines classes de violations des règles de dessin.

La pénalisation en temps de conception et la faible fiabilité du dessin manuel rendent cette méthode d'implantation inexploitable dans le domaine des VLSI.

L'objectif des méthodes d'implantation symboliques [HEN.70, WIL.77] était d'éliminer la phase de dessin manuel des masques.

Ces méthodes utilisent normallement une grille sur laquelle le concepteur dispose les transistors, les contacts et les différents niveaux d'interconnexion (diffusion, polycristallin, aluminium) qui sont tracés sur le réticule utilisant un code couleur (sticks [WIL]) ou un code graphique (espalier [HEN]).

L'inconvénient de ces méthodes est une diminution de la densité d'intégration car la rigidité imposée par une grille avec des dimensions fixes ne permet pas d'exploiter les gardes minimales utilisables dans un dessin manuel.

Les technologies symétriques comme le CMOS [MAR.78] s'adaptent beaucoup mieux à ce type d'implantation que la logique NMOS où il y a une dissymétrie dans le nombre et dans les dimensions des éléments d'une porte.

La faible densité obtenue avec des méthodes d'implantation symboliques ne permet son utilisation que pour des circuits de moyenne complexité.

L'interconnexion de briques standard a été une autre approche très utilisée qui a donné origine à de nombreux systèmes de CAO [GRA.77, LOO.79].

Une brique standard est normallement un ensemble de transistors qui implémente une fonction logique (ex.: portes NOR, bascules, etc...). Des entrées/sorties disposées à des endroits fixes sur la périphérie du bloc permettent de le connecter aux autres éléments du système.

L'implantation du circuit se fait en disposant les blocs aux endroits convénients et en laissant à un programme la tâche de dessiner automatiquement les connexions.

Cette méthode permet de réduire considérablement le temps d'implantation et de garantir une meilleure fiabilité, elle présente cependant plusieurs désavantages :

1 - les connexions automatiques entraînent une diminution très sensible de la densité du circuit ;
2 - la rigidité des portes d'E/S oblige à redessiner, partiellement ou complètement, une brique chaque fois que les connexions ne peuvent pas se faire de façon standard ;
3 - pour que la méthode soit rentable, il faut disposer d'une bibliothèque de briques relativement importante qui doit être redessinée à chaque évolution de la technologie.

L'augmentation de la complexité des circuits oblige à définir des briques plus complexes. Des méthodes de définition hiérarchique ont été proposées [PRE.79] de façon à construire des briques de plus en plus importantes en assemblant celles d'un niveau de complexité inférieure.

Nous pensons que l'idée de base, de cette méthode, de fractionner la complexité globale du circuit en utilisant des blocs de structure plus simple est très intéressante, mais son efficacité est considérablement diminuée par l'utilisation d'une méthode d'interconnexion automatique.

Il nous semble que la recherche de nouvelles méthodes d'implantation utilisant des briques standard doit donc se faire simultanément selon deux axes :

- définition des blocs de base de plus en plus complexes à utiliser directement par le concepteur ;
- définition de techniques systématiques d'assemblage de ces blocs, c'est-à-dire que la définition des briques doit être faite simultanément avec la définition des méthodes d'assemblage les mieux adaptées aux utilisations envisagées pour ces briques.

c/ Test

Le test des circuits intégrés très complexes pose de nombreux problèmes car les circuits sont composés d'un nombre très élevé d'éléments difficilement accessibles de l'extérieur.

Des travaux dans ce domaine [GAL.78, CRO.78, COU.80] ont essayé de proposer des méthodes pour tester certains organes des circuits intégrés.

L'augmentation de la complexité va nécessairement obliger à prendre en compte le problème du test au niveau de la définition de l'architecture et à introduire, si nécessaire, des organes supplémentaires permettant de faire dérouler plus facilement les séquences de test prévues.

2. Adaptation d'un circuit à une application

Une des caractéristiques du processus de fabrication dans l'industrie des semi-conducteurs est la possibilité de produire un très grand nombre de circuits avec un prix unitaire faible, ce qui permet d'amortir les investissements considérables nécessaires pour le développement d'un nouveau produit.

Cependant, au fur et à mesure que les circuits deviennent plus complexes, le nombre d'applications pour lesquelles un même circuit est utilisable, se restreint, ce qui conduit à une réduction des séries de production.

Dans le cas des circuits systèmes ce problème se pose d'une façon encore plus aigue car par sa définition même le circuit ne serait adapté qu'à l'exécution d'un système particulier et ne pourrait donc pas être utilisé efficacement pour une autre application.

Au niveau du LSI, la programmation a été introduite comme une solution permettant d'adapter une structure matérielle complexe à des applications de natures différentes.

Cependant, pour les circuits systèmes, la programmation ne peut pas être utilisée exclusivement car la spécificité de certaines fonctions d'E/S oblige à les implémenter par des structures matérielles spécialisées.

D'autre part, il nous semble impossible de concevoir un circuit standard car il serait très difficile à définir, a priori, les fonctions qu'il faudrait y inclure de façon à proposer un composant standard susceptible d'intéresser un grand marché potentiel.

D'ailleurs ce problème ne se pose pas seulement pour les circuits systèmes car dans la plupart des articles concernant les circuits VLSI [MOO.79,CAP.78] les auteurs sont à peu près unanimes pour considérer que les circuits à la demande représentent une tranche importante du marché VLSI.

Cependant, il est évident qu'il faut utiliser des nouvelles méthodes de conception car les méthodes traditionnelles impliqueraient d'énormes temps de conception.

Il nous a semblé qu'une méthode utilisant comme blocs de base des processeurs fonctionnels, ayant une complexité équivalente à celle des processeurs périphériques des systèmes de microprocesseurs (5000 et 10 000 transistors), assemblés autour d'un mécanisme de communication standard serait bien adapté au type de circuit à produire.

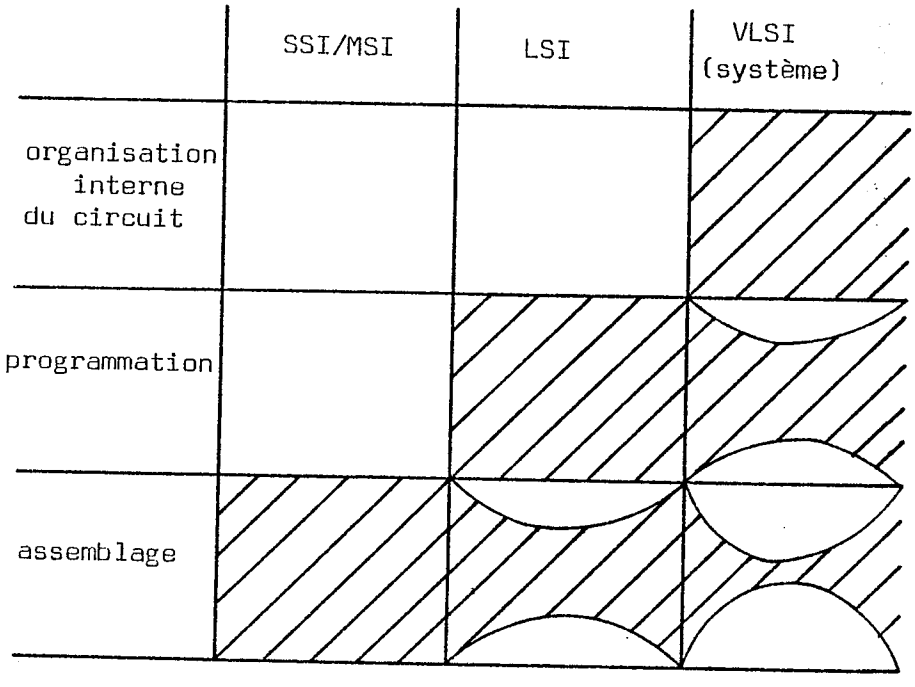
Cette démarche correspond à une évolution dans les possibilités d'adaptation des circuits à une application.

Dans le domaine du SSI/MSI l'adaptation était faite en assemblant des circuits de faible complexité sur des plaques de circuit imprimé. Les fabricants de semi-conducteurs proposaient un ensemble relativement important de circuits différents qui étaient cependant produits en de larges quantités car pour implémenter une application on devait en utiliser beaucoup.

L'adaptation des circuits intégrés à l'application était faite par l'utilisateur qui concevait un circuit logique en utilisant les composants standard.

Pour le LSI l'assemblage des circuits est devenu standard (dans un système évolué à base de microprocesseur les connexions sont pratiquement prédéfinies). La partie la plus importante de l'adaptation a donc passé du niveau de l'assemblage au niveau de la programmation.

Dans les circuits systèmes, où une approche purement programmable n'est pas utilisable, on propose d'introduire une nouvelle forme d'adaptation permettant à l'utilisateur de choisir les grands blocs matériels qui constitueront le circuit et qui seront spécialisés dans l'exécution de certaines fonctions spécifiques.



Pour réduire le coût de conception, une bibliothèque contiendrait les blocs les plus fréquemment utilisés de façon à ce que seulement de nouvelles fonctions obligeraient à dessiner un nouveau bloc.

3. Déphasage introduit par les retards dans les lignes de distribution de l'horloge

Dans les technologies actuelles, le temps de propagation intrinsèque d'un signal dans une ligne d'aluminium non chargée est négligeable par rapport au temps de commutation des portes élémentaires.

Remarque :
Dans ce texte nous appelons "temps de propagation intrinsèque" le temps de propagation d'un signal dans une ligne qui n'est connectée à aucune grille. Ce temps n'est donc fonction que de la vitesse de propagation des ondes électromagnétiques et de la capacité intrinsèque de la ligne.

Cependant, le maigrissement de la technologie fait varier inversement le temps de commutation des portes et le temps de propagation intrinsèque. Tandis que le premier se réduit proportionnellement au facteur de réduction géométrique de la technologie, le second augmente car la capacité des lignes se maintient constante, mais leur résistance croît (diminution de l'épaisseur et de la largeur des fils) [MEA.79].

K = facteur géométrique de réduction de la technologie
(ex.: NMOS 6µm → NMOS 2µm- K := 3)

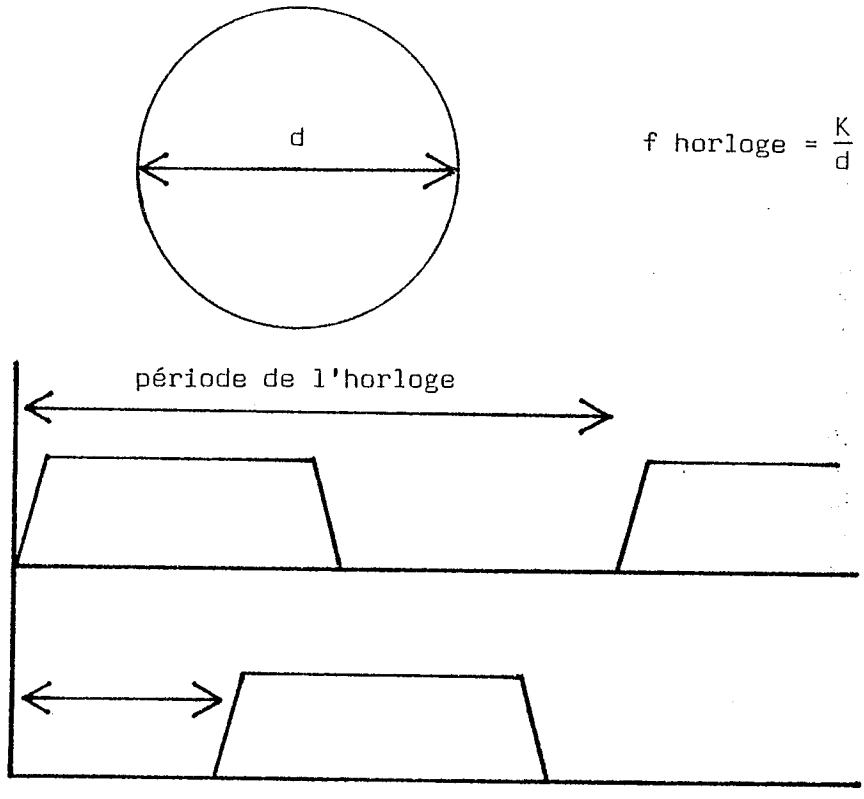
Temps de commutation d'une porte élémentaire	=	Temps de commutation d'une porte élémentaire	/ K
Temps de propagation intrinsèque	=	Temps de propagation intrinsèque	x K^2

La continuation de l'évolution technologique rendra donc les retards dans la les lignes de distribution de l'horloge de plus en plus importants par rapport à la vitesse de commutation des portes. A partir d'une certaine limite le fonctionnement synchrone de tout le circuit ne pourra plus être assuré.

Cependant, à l'intérieur des zones de taille plus petite, les retards pourront continuer à être considérés comme négligeables et donc un fonctionnement synchrone pourra continuer à être assuré.

Nous avons appelé ces regions "zones isochrones".

L'importance relative des retards introduits par le temps de propagation intrinsèque des lignes de distribution de l'horloge est évidemment dépendante de la fréquence de l'horloge utilisée, par conséquent le diamètre d'une zone isochrone est inversement proportionnel à la fréquence de l'horloge qui synchronise les actions exécutées à l'intérieur de la zone.



retard introduit par une ligne de longueur d

retard = fonction (du diamètre du module)

retard << période de l'horloge.

Le problème important au niveau de l'architecture devient la définition du protocole permettant de faire communiquer les différentes zones isochrones.

Il faut donc que ce problème soit pris en compte dès la phase de définition de l'architecture car il risque d'avoir une importance considérable dans les technologies envisageables à moyen terme ($\underset{\sim}{\sim}$ 1µm).

1.1.3. Démarche utilisée

En partant de l'idée initiale d'assembler un circuit en utilisant des processeurs fonctionnels comme des blocs de base, deux approches pouvaient être utilisées pour définir l'architecture :

- reprendre l'architecture des systèmes microprocesseurs et chercher à l'intégrer directement ; les blocs élémentaires seraient soit des microprocesseurs avec des ROM ou des RAM, soit l'équivalent des circuits périphériques des systèmes microprocesseurs discrets.
- définir une autre architecture mieux adaptée à un environnement circuit intégré monoboîtier où l'on ne partirait pas de la structure monoprocesseur classique implicite dans les systèmes à base de microprocesseurs.

La première approche a l'important avantage de permettre de réutiliser tout le support logiciel existant pour les systèmes discrets. La structure des blocs de base pourrait aussi être facilement obtenue à partir de celle des réalisations discrètes déjà existantes.

Par contre cette solution possède plusieurs désavantages en relation avec les aspects physiques et économiques propres aux circuits VLSI que l'on vient de décrire dans la section précédente :

1/ Temps de conception:

L'assemblage d'un circuit utilisant des briques complexes aseemblées autour d'un mécanisme standard de communication semble bien adapté à la complexité envisagée pour les circuits.

Cette méthode permet de fractionner la complexité globale du problème de façon à pouvoir isoler les différents niveaux du problème.

Cependant, pour que cette approche soit efficace au niveau de l'implantation il faut que les blocs puissent s'assembler d'une façon modulaire et si possible automatisable.

Une solution basée sur l'utilisation d'une architecture du type microprocesseur n'aurait pas ces deux propriétés.

Modularité :

Les circuits périphériques des systèmes à base de microprocesseur n'ont aucune propriété de modularité qui puisse permettre de les assembler d'une façon régulière à l'intérieur d'un circuit intégré.

Le découpage fonctionnel qui a conduit à la définition de ces circuits a été très influencé par des problèmes comme le nombre de broches du boîtier et la compatibilité avec les autres éléments de la famille déjà existants.

Il y a donc des blocs qui seraient très mal adaptés pour être utilisés comme des blocs de base à l'intérieur d'un circuit, comme par exemple les PLA ou les canaux de DMA qui sont groupés par blocs de quatre tandis qu'un controleur de disque souple qui a toujours besoin de cette fonction, ne la contient pas.

Il serait donc, de toute façon, nécessaire de redéfinir les éléments à utiliser.

Implantation non automatisable:

La complexité au niveau de l'implantation d'un circuit provient beaucoup plus du nombre d'interconnexions que du nombre de transistors qu'il contient.

Dans les systèmes à base de microprocesseurs, le bus est composé d'un nombre élevé de lignes (bus de données + bus d'adresses + bus de contrôle) en plus d'un certain nombre de lignes non standardisées comme les lignes de demande/ acquittement des canaux de DMA.

Par conséquent un système d'interconnexion composé d'un nombre important de lignes et dont la topologie ne serait pas fixe car les blocs de base ne peuvent pas être assemblés d'une façon modulaire.

2/ Isochronisme :

Les systèmes classiques de microordinateurs ont une architecture monoprocesseur dont la performance dépend très fortement du débit du bus mémoire/CPU.

Il faut donc que ce bus, qui doit être distribué à tous les éléments du système, soit rapide. Ceci est en contradiction avec ce qui a été dit sur les problèmes d'isochronie. Un bus rapide a une zone isochrone petite et ne peut donc pas être distribué à tous les éléments du circuit.

3/ Performances :

Pour essayer de résoudre le problème de l'isochronisme du bus il faut utiliser une fréquence de transfert faible de façon à ce que la zone isochrone du mécanisme de communication puisse englober tous les éléments du circuit.

L'hypothèse de concevoir le circuit comme une machine de Von Neuman à programme mémorisé est donc exclue car la performance du circuit serait limitée par le débit du bus système (mémoire/UAL).

Il faut donc utiliser une architecture permettant de diminuer le flux d'informations véhiculée par le bus commun. Ceci peut être obtenu en faisant de chacun des organes utilisateurs du bus un élément intelligent qui puisse exécuter localement un nombre important d'actions de façon à échanger moins d'information avec les autres éléments du système.

Si chacun des éléments utilisateurs du bus a une certaine puissance de calcul, les données transférées peuvent avoir une forme "codée" qui permet de transférer la même information avec un flux de données plus faible.

1.2. ARCHITECTURE DE BASE

Modules

Chaque bloc de base du système peut être considéré comme un processeur fonctionnel ayant une complexité de 5000 à 10 000 transistors.

On emploie la désignation fonctionnelle car chacun de ces blocs doit être spécialisé dans l'exécution d'une fonction spécifique, il ne s'agit donc pas d'une architecture où les processeurs sont banalisés pouvant exécuter n'importe quelle tâche dans le système.

Dans le reste du texte on préfère cependant appeler les blocs de base modules pour éviter la confusion courante de considérer qu'un processeur a un seul point d'exécution qui doit obligatoirement être multiplexé si plusieurs processus sont actifs à leur intérieur. Cette vision logiciel n'est pas forcément vraie au niveau matériel où un processus peut être facilement implémenté par un automate matériel dédié.

Dans les blocs de base de MOSAIC il y a souvent plusieurs processus qui sont exécutés en parallèle par des organes matériels différents, pour cette raison on a préféré utiliser la désignation plus anodine de module à celle de processeur.

Chaque module est une zone isochrone. Cette approximation est parfaitement admissible car les modules ont une complexité équivalente à celle des processeurs actuels où les problèmes d'isochronisme ne se posent pas. En choisissant des fréquences d'horloge appropriées, les modules peuvent être considérés comme des circuits totalement synchrones.

Système d'interconnexion

Même s'il s'agit de modules spécialisés, l'architecture de base de MOSAIC est une architecture multiprocesseurs.

Le problème le plus important de l'architecture devient donc la définition de la façon dont communiquent les processus implémentés dans des modules différents.

Deux approches sont envisageables :
1 - distribuer les mécanismes de communication et utiliser un protocole de communication du style réseau ;
2 - centraliser la communication dans une mémoire commune.

La première hypothèse, bien qu'intéressante, suppose l'utilisation de mécanismes matériels très complexes pour implémenter les interfaces de communication.

La surface occupée par le mécanisme de communication rend donc cette première hypothèse inexploitable au moins pour les technologies envisageables à moyen terme. D'autre part, à l'intérieur d'un circuit intégré, un mécanisme de communication basé sur l'utilisation d'une mémoire commune nous semble bien adapté, non seulement parcequ'il permet d'employer une stratégie de transfert asynchrone plus facile à mettre en oeuvre au niveau matériel, mais aussi parcequ'une mémoire, dans la technologie des circuits intégrés, est une structure régulière facilement implantable et donc facile à adapter aux besoins de l'application.

Etant donnée l'importance de minimiser le nombre de connexions et de les disposer le plus régulièrement possible, les structures du type multibus ont été a priori exclues.

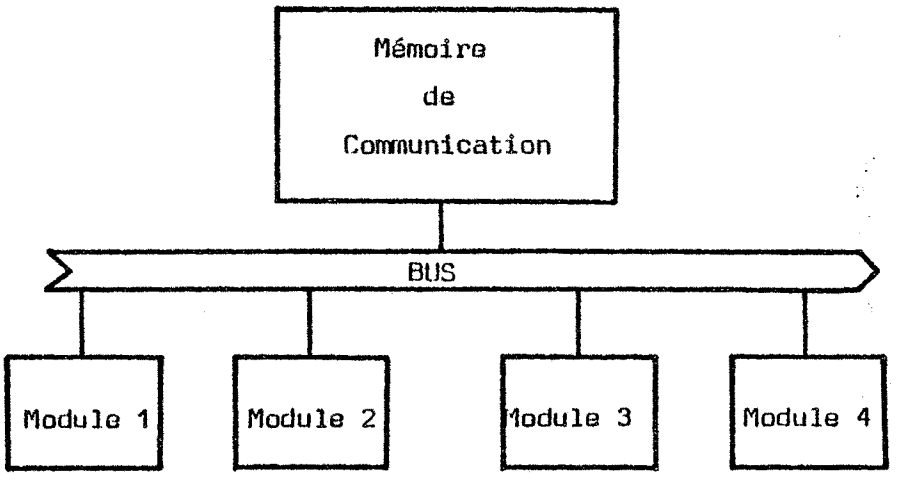


Fig. 1.5. - Structure de base.

Il est cependant évident qu'une mémoire et un bus commun sont des goulots d'étranglement potentiels du système, il faut donc chercher à minimiser les temps d'utilisation du mécanisme de communication.

La méthode classique pour améliorer les temps de réponse de ce type de système est d'utiliser des mécanismes d'accés direct à la mémoire (DMA) que l'on pourrait soit distribuer parmi les modules du système, soit les concentrer dans un seul organe.

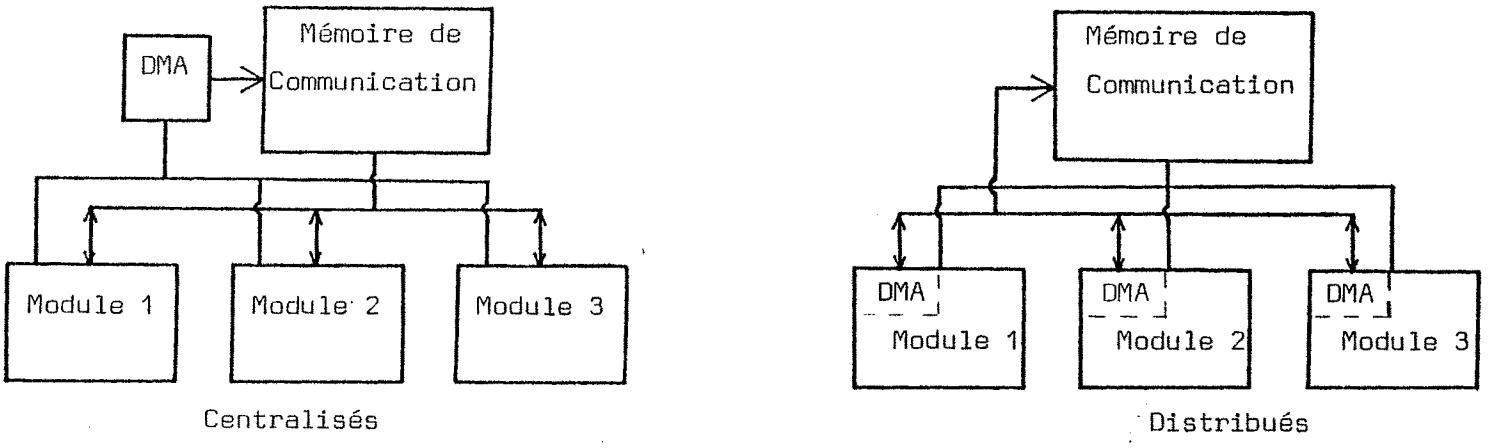


Fig. 1.6 - Mécanisme d'accés à la mémoire.

Des mécanismes de ce type permettent de diminuer le temps d'utilisation du bus commun mais ne contribuent pas à réduire le flux d'informations sur le bus commun et donc la performance continuerait à être limitée par les problèmes d'isochronisme.

D'autre part, une analyse des structures de données utilisées pour la communication entre les modules permet de remarquer que des structures telles que des FIFOs, des piles, etc... sont souvent utilisées.

Les mécanismes d'accés des DMA sont des mécanismes séquenciels qui ne sont pas bien adaptés pour implémenter ce genre de structures de données.

Une solution permettant de diminuer le débit et d'implémenter des structures de données plus élaborées est de définir une mémoire intelligente en couplant à la mémoire commune un processeur d'adressage.

La synchronisation entre les processus des différents modules est souvent associée à l'état des variables de communication. Le processeur d'adressage

connaît l'état de ces variables et peut donc exécuter facilement la synchronisation qui leur est associée.

Dans la description qui vient d'être faite on a associé à une structure de données la façon dont elle est utilisée et la synchronisation des processus qui l'utilisent. Dans le domaine des systèmes d'exploitation des variables de communication ayant les caractéristiques décrites correspondent à des concepts bien connus comme les moniteurs [HOA.74] , les managers [JAM.77] ou les D-Modules [GUI.79].

Il nous a donc paru important depuis le début de l'étude de formaliser cette couche système d'exploitation qui apparaît à l'intérieur d'un circuit ayant ce niveau de complexité.

Dans le prochain chapître on définit un formalisme système permettant de spécifier les tâches exécutées par les différents modules et la façon dont ces tâches communiquent entre elles.

Plutôt que de définir un nouveau formalisme on a préféré choisir parmi les diverses propositions existantes celles qui nous semblent les mieux adaptées.

Un langage de haut niveau qui incorpore les éléments du formalisme système est proposé comme le langage de spécification du fonctionnement du circuit.

Dans le troisième chapître on reprend le problème de la définition de l'architecture du circuit en regardant plus en détail les solutions possibles pour implémenter le formalisme système décrit dans le chapître 2.

Parmi les hypothèses analysées, celle qui nous semble présenter le plus d'intérêt est une solution basée sur l'utilisation d'un processeur noyau qui contrôlerait les communications à l'intérieur du circuit.

Cette architecture reste cependant une architecture à mémoire commune où existe toujours une limite maximale du débit.

Deux autres solutions sont donc envisagées pour des applications où des processus d'Entrées/sorties ayant des contraintes temps réel très strictes imposent des vitesse de transfert élevées.

Les trois solutions sont définies de façon à être permutables ce qui permet de proposer une architecture souple et adaptable aux besoins de différents types d'applications.

Dans le quatrième chapître le protocole de gestion du bus commun est défini et dans les chapîtres 5, 6, 7 et 8 sont proposées et évaluées les solutions matérielles correspondant aux architectures choisies dans le chapître 3.

Les deux derniers chapîtres sont dédiés à la définition des modules et à la présentation de quelques exemples d'applications qui ont été étudiées.

CHAPITRE 2

F O R M A L I S M E S Y S T E M E

2.1. CHOIX D'UN FORMALISME

Indépendamment de la réalisation matérielle des variables de communication, il nous a semblé important d'avoir un formalisme système permettant de décrire les primitives de synchronisation et les fonctions d'accés qui leur sont associées.

On pense aussi qu'un formalisme système de haut niveau constitue un outil bien adapté pour décrire et spécifier le fonctionnement global de circuits du type MOSAIC, car il permettrait au concepteur d'étudier les fonctions nécessaires pour l'exécution d'une application spécifique sans être submergé par les détails de leur implémentation.

Cependant, pour qu'une spécification de ce genre puisse être exploitable il faut qu'il existe une correspondance biunivoque entre les éléments décrits et les structures matérielles qui les implémentent. Dans le cas contraire le concepteur perdrait le contrôle sur la réalisation physique du circuit et risquerait de définir des systèmes sans aucune viabilité économique notamment au point de vue surface.

Les modules, considérés précédemment comme les blocs de base du sysème, peuvent être représentés au niveau du formalisme comme un ensemble de processus pouvant s'exécuter en parallèle.

Pour collaborer dans l'exécution d'une tâche commune les processus d'un module et ceux définis dans les autres modules doivent communiquer et se synchroniser.

Dans le domaine des systèmes d'exploitation plusieurs propositions ont été faites sur la façon d'exprimer la synchronisation entre processus : sémaphores [DIJ.68], moniteurs [HAN.73], expression de chemins [CAM.74], commandes gardées [DIJ.75].

L'objectif commun de toutes ces propositions est la définition d'un formalisme permettant aux programmeurs, non familiarisés avec les aspects système, d'exprimer sous une forme structurée et fiable la façon dont un ensemble de processus parallèles communiquent entre eux. Les mécanismes de synchronisation se sont donc progressivement éloignés du niveau de l'implémentation laissant au compilateur la tâche de construire les couches intermédiaires nécessaires.

D'autres propositions plus récentes [HOA.78, HAN.78] essaient de proposer des primitives de communication qui sont indépendantes du support matériel qui les implémente (réseau, mémoire commune, etc...).

Comme on l'a déjà mentionné notre objectif est relativement différent car bien que l'on souhaite disposer d'une description structurée et fiable, il nous intéresse surtout d'avoir un outil de spécification qui, par la nature même des applications, ne soit pas trop éloigné du niveau matériel.

Il faut aussi remarquer que le système s'adresse surtout aux concepteurs de circuits qui sont normallement plus familiarisés avec les notions de synchronisation directe (démarrage et blocage des processus) qu'avec certaines notions plus abstraites du type "commandes gardées" ou "rendez-vous" [ADA.79].

Le concept de moniteur [HAN.73, HOA.74] nous a semblé présenter une certaine structuration tout en maintenant des primitives de synchronisation d'un niveau assez proche de l'implémentation.

Les notions de type abstrait et d'exclusion mutuelle dans les accés qui sont sousjacentes dans la définition du concept de moniteur, le rendent plus fiable et plus commode à manipuler que les sémaphores. Les primitives de synchronisation signal/wait (Hoare) et delay/continue (Hansen) peuvent être traduites au niveau matériel par des actions de blocage et démarrage des processus.

Finalement, l'idée unificatrice des moniteurs (structure de données + fonctions d'accés + primitives de synchronisation) est tout à fait dans l'exprit de l'ébauche de l'architecture présentée au chapître 1.

L'existence de plusieurs langages de haut niveau qui implémentent le concept de moniteur a été un autre argument qui nous a influencé dans le choix de ce formalisme. A partir de ces langages, un outil de spécification du circuit peut être facilement défini.

2.2. Système MOSAIC

2.2.1. Modules

Chaque module exécute un ensemble de tâches fixes et bien définies.

Dans le type d'applications envisagées qui se situent très près du contrôle d'organes matériels, la nature des tâches à exécuter par chaque module peut d'une manière générale être facilement définie.

Chaque tâche est modélisée par un processus cyclique qui exécute un certain algorithme et qui peut communiquer soit avec des processus appartenant au même module, soit avec des processus définis dans d'autres modules.

Le concept de module nous oblige donc à séparer les moniteurs en deux classes:

- Moniteurs internes :

  définis à l'intérieur d'un module, ils ne sont accessibles que par les processus appartenant au même module.

- Moniteurs externes :

  définis à l'extérieur des modules, ils peuvent être accédés par des processus appartenant à des modules différents.

Considérons par exemple un module qui correspond à une interface de téléinformatique asynchrone :

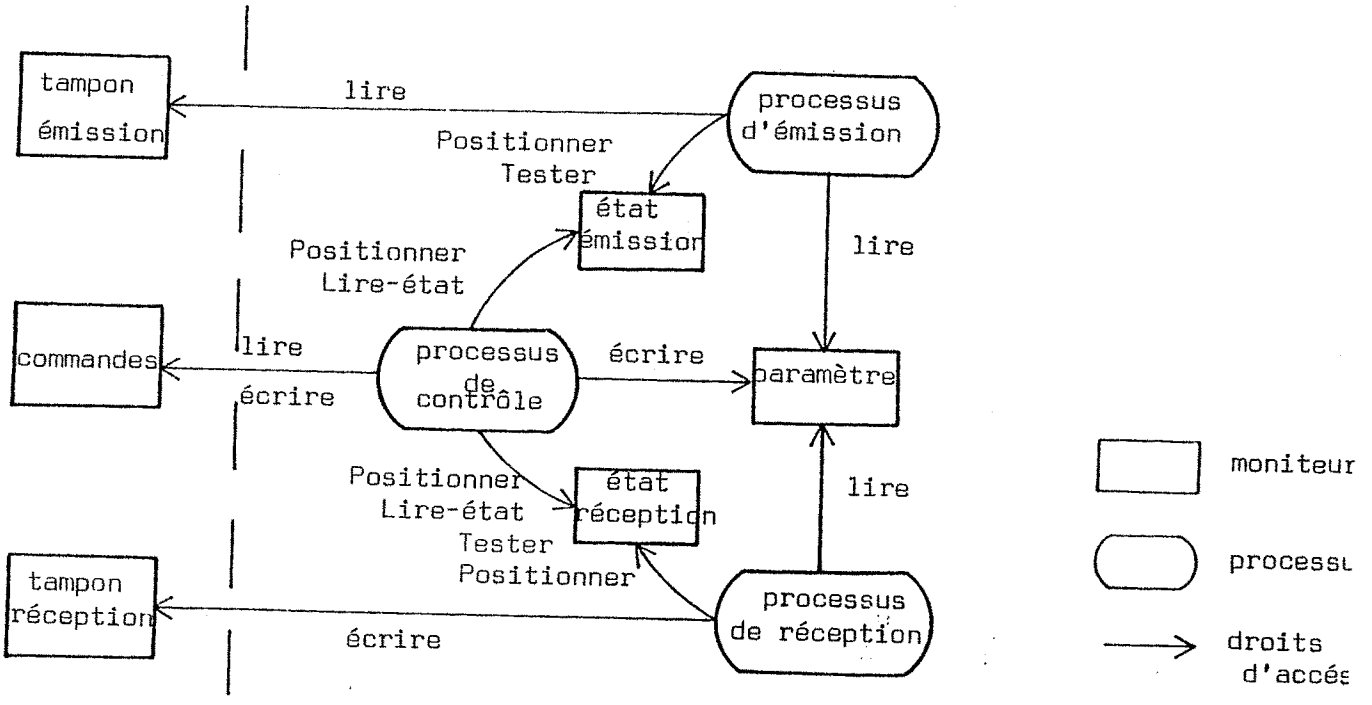


Fig. 2.1 - Module interface asynchrone téléinformatique.

Les processus de réception et d'émission exécutent l'algorithme correspondant à la réception et à l'envoi des caractères dans un protocole de transmission asynchrone.

Le processus de contrôle reçoit des ordres (ex.: démarrer/arrêter émission, démarrer/arrêter réception, état émission?, état réception?, etc...) écrites par les autres processus dans le moniteur externe "commandes".

Les paramètres pour les processus de transmission sont écrits dans le moniteur "commandes". Les moniteurs état_émission et état_réception permettent au processus de contrôle d'arrêter et de démarrer l'exécution des processus qui les utilisent ainsi que de tester leurs états respectifs.

2.2.2. Moniteurs externes

Un moniteur est défini par une structure de données et par les opérations permises sur cette structure, donc commune au type abstrait car le reste du système ne connait que les opérations qu'il peut exécuter sur les données mais pas les détails de leur exécution.

Deux niveaux de synchronisation sont associés au concept de moniteur, le premier, implicite, impose qu'un seul processus puisse être actif à l'intérieur du moniteur de façon à garantir l'exclusion mutuelle dans la manipulation de la structure de données.

Le deuxième niveau est défini par l'utilisateur à l'intérieur du moniteur et permet de suspendre l'exécution d'un processus utilisateur en fonction d'une condition logique évaluée sur les variables internes du moniteur ou sur les paramètres passés dans l'appel procédural.

Les primitives utilisées pour exprimer les opérations de synchronisation diffèrent selon les auteurs :

Hoare : wait/signal exécuté sur une variable de type "condition" (file d'attente) [HOA.74].

Hansen : Delay/continue exécuté sur une variable du type "queue" [HAN.75].

Les deux primitives sont utilisées pour signaler explicitement le blocage (wait, delay) et le démarrage (signal, continue) d'un processus.

L'utilisation d'un mécanisme direct pour signaler le démarrage d'un processus en attente pose le problème de savoir quel est l'état du processus qui exécute la primitive "signal" (ou "continue") , car au moment où il réveille le processus bloqué, deux processus seront actifs simultanément à l'intérieur du moniteur.

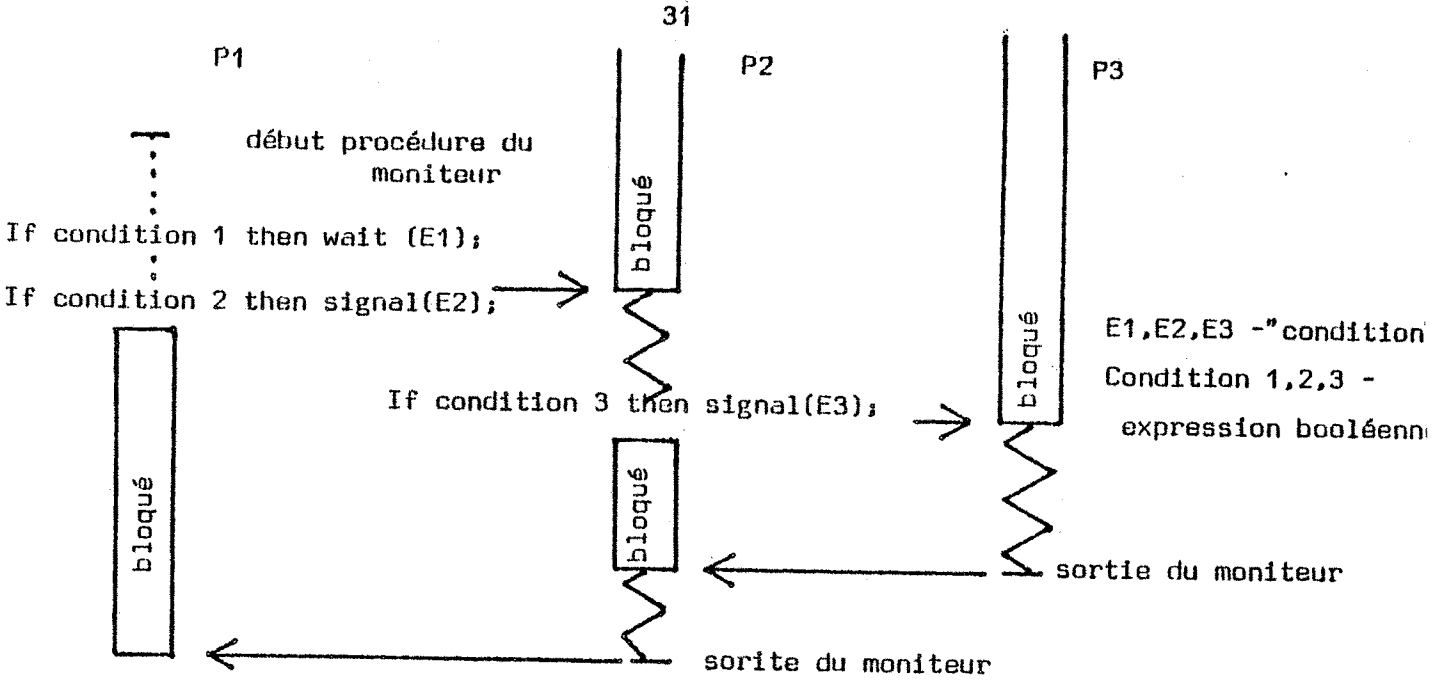
Selon Hoare, un processus qui exécute un "signal" se bloque de façon à permettre au processus réactivé de finir l'exécution de la procédure.

Hansen considère qu'après un "continue" le processus abandonne le moniteur et continue son exécution, ce qui implique que l'instruction "continue" doit être unique et terminale dans une procédure.

La proposition d'Hansen est donc très restrictive et celle d'Hoare peut conduire à des incohérences dans l'utilisation du moniteur [KES.77, KEE.77] ce que l'on peut décrire par l'exemple suivant: (voir schéma page suivante)

> Un processus P1 exécutant une procédure du moniteur modifie les conditions internes de celui-ci de façon à permettre de débloquer un autre processus en attente P2.
> P1 se bloque et P2 redémarre et continue à exécuter le moniteur jusqu'à: a/ la fin de procédure, b/ l'exécution d'un wait, c/ l'exécution d'un signal.
>
> Seulement quand P2 ou les autres processus qu'il a pu avoir débloqués (en exécutant des "signals") ont tous fini l'exécution de la procédure, P1 pourra être réveillé. Rien ne garantit donc, qu'à ce moment, les conditions qui ont permis à P1 d'exéucter le moniteur (ex.: condition 1) soient encore vérifiées quand il est réveillé.

D'autres problèmes se posent aussi comme par exemple ce qui arrive quand un processus fait un "signal" sur une file vide.



Pour éviter les problèmes décrits on préfère utiliser une structure plus rigide du type de celle proposée par Kessels [KES.77].

On considère que les conditions à tester doivent être déclarées dans les variables du moniteur :

<identificateur> : condition (expression booléenne)

Pour tester cette condition dans le corps de la procédure, on utilise une instruction :

wait (identificateur condition)

qui a pour effet de bloquer le processus si la valeur de l'expression booléenne n'est pas vraie.

Quand un processus abandonne le moniteur, à la fin de l'exécution de la procédure, ou quand il est bloqué dans un "wait", les conditions susceptibles d'avoir été modifiées par l'exécution des instructions précédentes sont réévaluées.

On laisse au compilateur la tâche de déterminer pour chaque procédure les conditions qui doivent être réévaluées.

Cette proposition suppose que les variables intervenant dans une condition sont des variables internes du moniteur (les paramètres des procédures sont donc exclus). Ceci permet de réévaluer les conditions sans avoir besoin de réveiller les processus en attente car les conditions ne dépendent pas des variables internes des processus.

On verra au chapître 4 que cette structure rigide des mécanismes de synchronisation est plus facile à implémenter par des mécanismes matériels que les propositions précédentes.

Cependant certains algorithmes obligent à faire intervenir les paramètres pour décider de l'exécution d'une procédure. La programmation peut donc devenir plus compliquée pour pouvoir tenir compte des paramètres dans les conditions sans les utiliser directement.

Exemple :

Un moniteur contrôle une file de messages composée de Nmax octets. Quand un processus veut envoyer un message de N octets, il doit le signaler dans les paramètres de la procédure.

Etant donné que la taille du message est un paramètre de la procédure, on ne peut pas l'utiliser directement dans la définition de la condition. La structure suivante serait donc nécessaire pour garantir le bon fonctionnement de la file.

Ex.:

```
FILE_MESSAGE = monitor
        .
        .
     ECRITURE_AUTHORISEE condition (OCTETS_LIVRE>0)
     PLEIN : condition NBLOCAGE
        .
        .
procedure  ECRIRE (NOCTETS_MESSAGE:integer ; ...)
begin
          wait (PLEIN)
          NBLOCAGE := false ;
          OCTETS_LIVRE := OCTET_LIVRE-NOCTETS_MESSAGE
          wait (ECRITURE_AUTHORISEE)
          NBLOCAGE := true ;
             .
             .
             .

end ;
begin
     OCTET_LIVRE := Nmax ;
     NBLOCAGE := true ;
end ;
```

2.3. DECOUPAGE FONCTIONNEL D'UNE APPLICATION

Tout le système est statique, les processus et les structures de données existeront toujours après l'initialisation du système.

Le gain d'une gestion dynamique est dans les machines classiques associé à la possibilité d'éliminer physiquement les processus de la mémoire et donc d'augmenter les ressources du système. Dans notre cas, le gain serait très faible, ou même nul, car presque tous les processus ont une existence matérielle (ROM ou PLA).

D'autre part, la plupart des tâches n'évoluent pas pendant la vie du système car elles sont associées d'une forme fixe aux fonctions d'E/S controlées par les modules.

Découpage fonctionnel

Dans le cas général il est difficile de proposer une méthode pour effectuer un découpage modulaire d'un système à partir de sa spécification car il n'y a pas de critères standard et fiables pour effectuer cette décomposition [PAR.72].

Dans MOSAIC ce découpage est cependant considérablement plus simple car les applications envisagées se situent très prés du contrôle d'organes matériels (disque souple, organes mécaniques, connexion de téléinformatique) et à ce niveau, il est relativement facile de définir les tâches qui doivent être inclues dans un module de façon à ce qu'il constitue un bloc de base cohérent.

Une des applications étudiées, un récepteur pour le système ANTIOPE [MAR.75] nous permet d'explifier ce qui vient d'être dit.

Très brièvement, les fonctions à exécuter par le circuit sont :
Recevoir à partir d'un canal vidéo des informations numériques transmises dans un protocole de transmission de données par paquets [NOI.75].

Le langage du protocole est interprété et génère un langage graphique qui sera visualisé sur un téléviseur commercial.

L'utilisateur peut en utilisant un clavier choisir la trame à visualiser et le mode de visualisation.

Si l'on analyse la description du système, on peut a priori isoler les modules suivants :

- module de réception ligne TV,
- module d'interprétation et génération du langage graphique,
- module de visualisation,
- module d'encodage du clavier.

Analyse des relations entre les processus

Le processus de réception fait la désérialisation des caractères et exécute l'algorithme du protocole de transmission (reconnaissance du mot de synchronisation, identification de la trame, décodage du mot de format).

La vitesse de transmission est très élevée (fréquence bit maximale dans la gamme 3,5 à 7 Mhz) et donc le processus d'interprétation ne pourra traiter les données à la même cadence.

Un tampon géré en FIFO semble bien adapté pour établir la communication entre les processus de réception et d'interprétation.

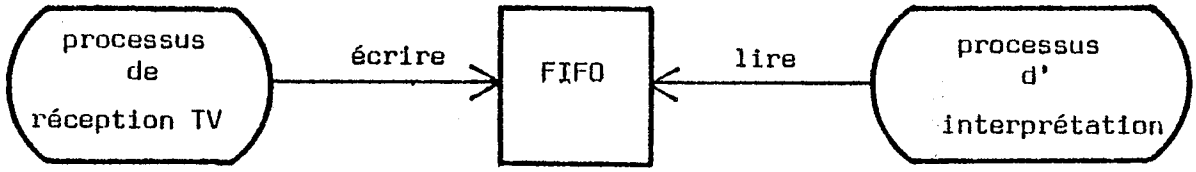


Fig. 2.3. - Processus de réception et d'interprétation

Le processus d'interprétation essaie toujours de lire le FIFO. Si celui-ci devient vide, le processus sera bloqué jusqu'au moment où un nouveau caractère y sera écrit.

Les codes du langage graphique produits par le processus d'interprétation sont stockés dans une mémoire de rafraîchissement de l'écran.

Aucune synchronisation explicite n'est associée à ce moniteur car le processus de visualisation n'est pas commandé par la présence de données dans sa mémoire de rafraîchissement.

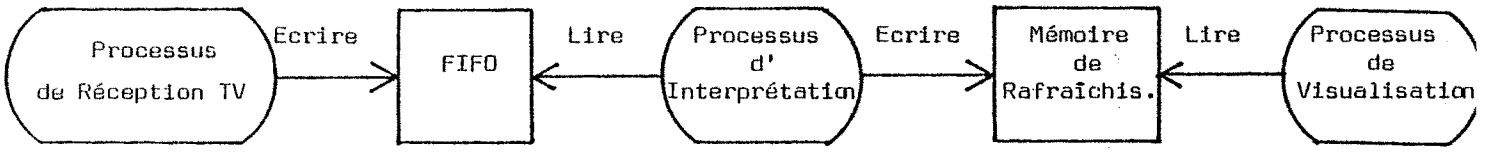


Fig. 2.4 - Réception et visualisation des caractères

Le processus clavier fait le balayage et l'encodage d'un clavier. Les caractères reçus peuvent correspondre soit à une paramétrisation du mode de visualisation, soit à la sélection d'une nouvelle page à recevoir. Pour exécuter la paramétrisation des modules de réception et de visualisation, deux solutions sont possibles :

1 - Le processus clavier dépose les caractères reçus dans un moniteur "paramètre" . Le processus de réception (ou de visualisation) lit au début de son cycle ce moniteur, et s'il détecte qu'il s'agit d'une paramétrisation qui lui est adressée, modifie ses variables internes.

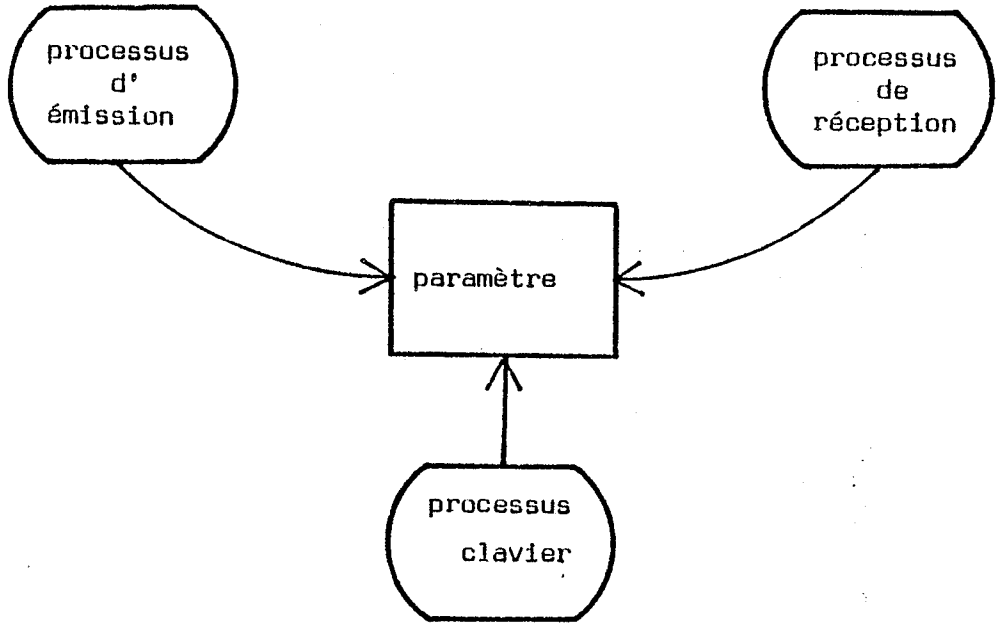


Fig. 2.5 - Système sans processus de contrôle

Cette solution a cependant deux désavantages importants au niveau des performances du système :

a/ les fréquences de cycle des processus de réception et de visualisation sont élevées. L'accés aux moniteurs externes représentera dans n'importe quelle implémentation une action plus lente que l'exécution locale.

b/ la paramétrisation est une action peu fréquente. Obliger le processus de réception et de visualisation à tester à chaque cycle la présence de paramètres, se transforme presque en une attente active.

2 - Considérer deux processus de contrôle, un dans chaque module qui se chargent exclusivement de la paramétrisation.

Les processus de réception et de visualisation continuent à tester si une paramétrisation est en cours mais ils le font localement (à l'intérieur du module) ce qui permet de réduire considérablement le temps du test.

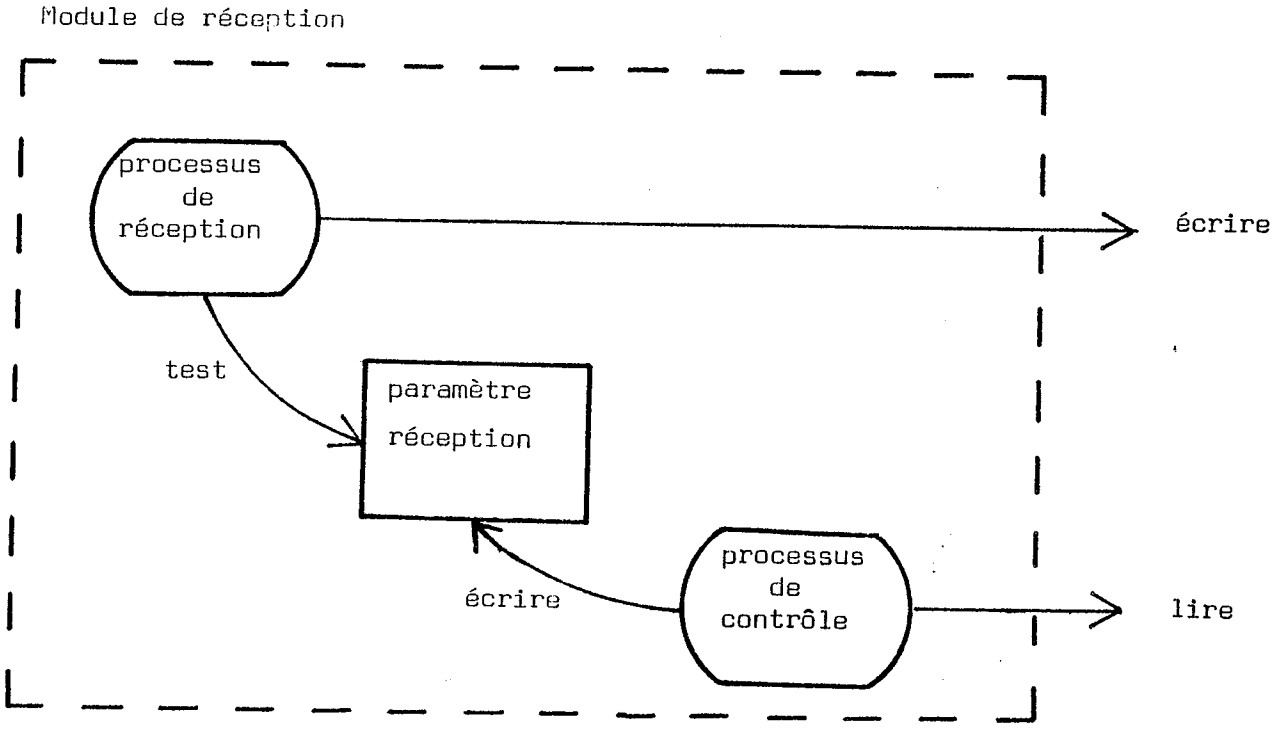


Fig. 2.6 - Module Réception

Les processus de contrôle lisent le moniteur "paramètre" et se bloquent quand il n'y a pas de message.

Cette dernière solution est donc plus performante car elle évite l'attente active qui est implicite dans la première.

Exemple : Processus de réception.

VAL - signale la présence d'information numérique dans le canal vidéo
S - double octet de synchronisation
X,Y,Z - 3 octets qui définissent le numéro de la trame à recevoir
DESERIALISER - procédure de désérialisation de l'information
IDENTITE - fonction booléenne qui vérifie l'égalité de deux octets à distance 1 (1 bit différent est toléré).

```
RECEPTION  = process
ETAT := REPOS ;
```

```
cycle
begin
        PARAM.TEST(MODIFICATION) ;          "Test parametrage"

        if MODIFICATION then ETAT:=REPOS ;
        if VAL then
        begin
             DESERIALISER(CARACTER) ;        "Protocole de reception"
             case ETAT of
             REPOS : if IDENTITE(CARACTERS,S1) then
                        begin
                             DESERIALISER(CARACTER) ;
                             if IDENTITE(CARACTER,S2) then
                                ETAT:=ADRESSØ  ;
                                ADRESSØ : if IDENTITE(CARACTER,X) then
                                                ETAT:=ADRESS1
                                                elseETAT:=REPOS ;
                                ADRESS1 : if IDENTITE(CARACTER,Y) then
                                                ETAT:=ADRESS2
                                                else ETAT:=REPOS ;
                                ADRESS2 : if IDENTITE(CARACTER,Z) then
                                                ETAT:=FORMAT
                                                else ETAT:=REPOS ;
                                FORMAT  : begin
                                                N=DECODAGE(CARACTER) ;
                                                ETAT:=RECEP ;
                                          end ;
                                RECEP   : begin
                                                FIFO.ECRIRE(CARACTER);
                                                N:=N-1 ;
                                                if N=Ø then ETAT:=REPOS ;
                                          end
                        end
end cycle ;
```

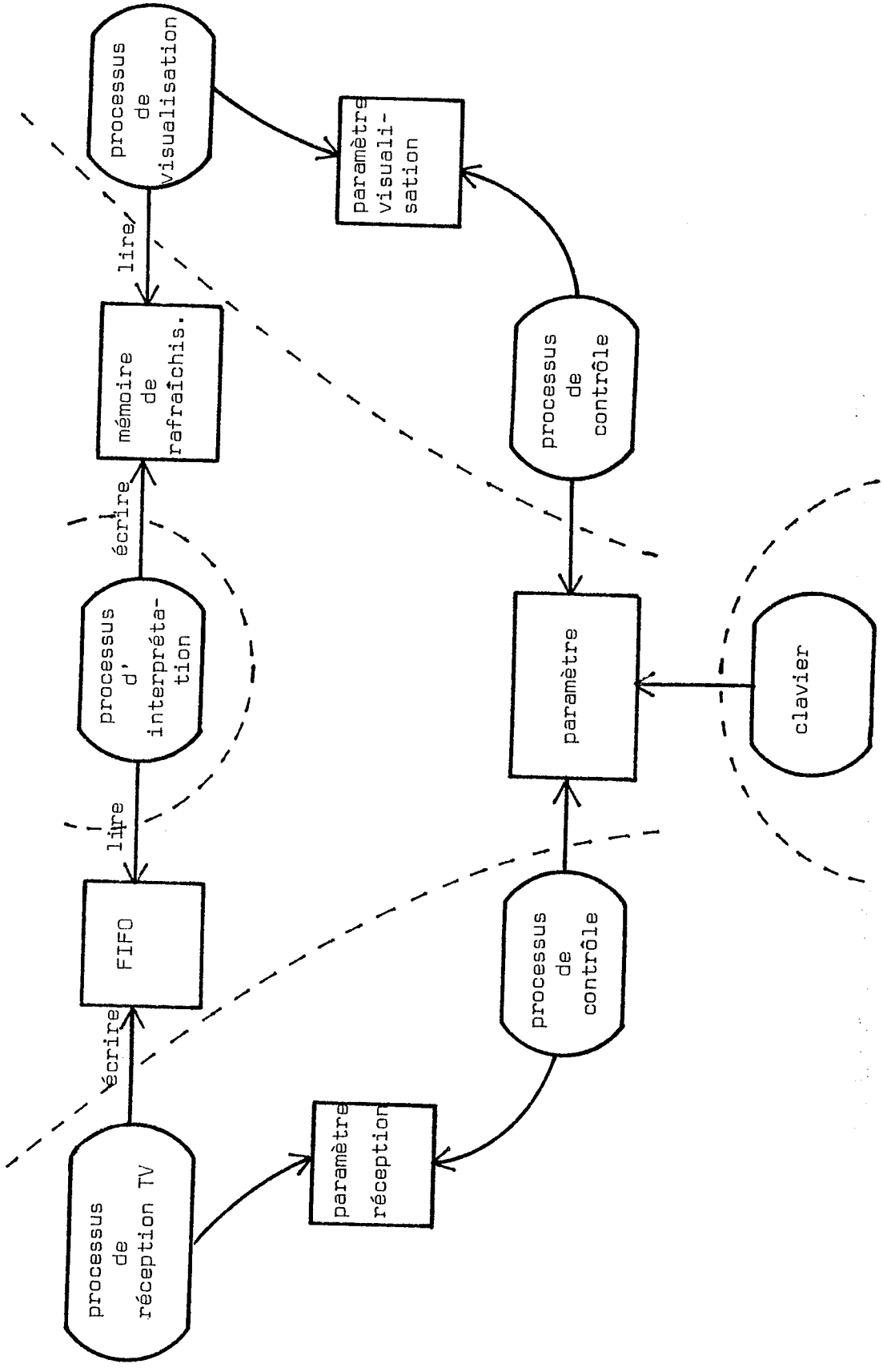


Fig. 2.7. - Système ANTIOPE

2.4. LANGAGE DE SPECIFICATION

Dans la conception de circuits très complexes les outils d'aide à la conception auront une importance prédominante.

Il nous a donc semblé important d'avoir un outil de spécification du fonctionnement global du circuit avec lequel on puisse décrire d'une manière précise les fonctions exécutées par les modules et les algorithmes des procédures des moniteurs externes.

Un langage de haut niveau qui incorporerait les primitives de synchronisation décrites précédemment nous semble parfaitement adapté à cette fonction car, d'une part les structures algorithmiques du langage permettent de décrire les tâches à exécuter, d'une façon plus commode et plus claire que les organigrammes classiques et d'autre part, l'utilisation des moniteurs permet de signaler sans ambiguité les endroits où sont faites les opérations de synchronisation.

Le "langage de spécification" proposé est inspiré de Concurrent PASCAL [HAN.75] MODULA [WIR.77] et PASCAL PLUS [BUS.77].

Quoique les notions de base soient les mêmes, il faut effectuer quelques modifications de façon à introduire les concepts nouveaux utilisés dans notre système.

Module

```
                NOM_MODULE = module
                                use (liste des procédures appartenant à des
                                     moniteurs externes, utilisées par les
                                     processus du module)
moniteurs      {  NOM_MONITEUR = monitor
 internes      {
                                end ;

processus      {  NOM_PROCESSUS = process
               {
                                end ;
                end ;
```

La liste des procédures des moniteurs externes dans l'entête constitue les paramètres formels du module. D'un point de vue fonctionnel, cette liste représente ce qui doit exister dans le reste du système de façon à ce que le module puisse s'exécuter.

Dans la partie du programme où les interconnexions du système sont décrites les paramètres formels sont substitués aux noms des procédures des moniteurs externes que les processus utilisent.

Module :

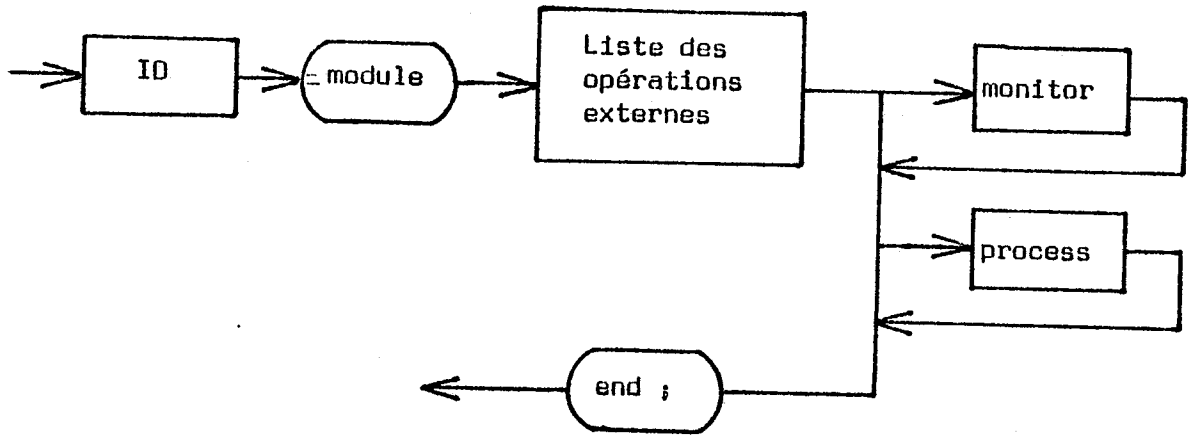


Fig. 2.8 - Module

Liste des opérations externes :

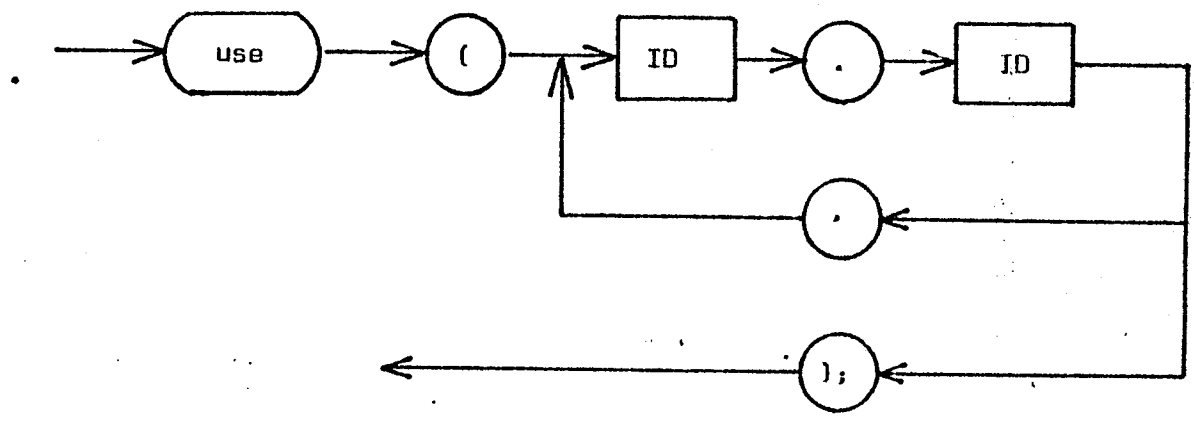


Fig. 2.9 - Liste des procédures externes

Moniteurs externes

```
NOM_MONITEUR = external monitor ("noms des procédures des autres
                                  moniteurs externes")
        {
        "Déclaration des variables et conditions"
        }
        procédure  NOM_PROCEDURE ("paramètres")
procédures
internes        begin
du moniteur
qui ne peu-
vent être       end ;
appelées de
l'extérieur
        procedure entry  NOM_PROCEDURE ("paramètres")
procédures
pouvant         begin
être
appelées
de              end ;
l'extérieur

        begin
             "initialisation"

        end ;
```

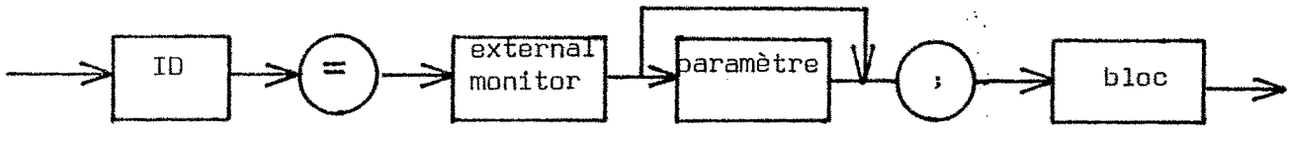


Fig. 2.10 - Moniteur externe

Remarque :

Dans cette définition comme d'ailleurs dans les précédentes, on a juste voulu montrer les différences par rapport aux propositions classiques. Dans [HAN.77] on peut trouver la définition complète des éléments du langage qui n'ont pas été définis ici comme Identificateur, Bloc, Paramètres, etc...

Dans l'annexe 1 on décrit la programmation de quelques moniteurs externes parmi les plus souvent utilisés.

Traitement des exceptions

Dans l'exécution des procédures on peut prévoir des exceptions au déroulement normal de leur algorithme.

Le traitement à faire dans ces cas (réinitialisation de certains paramètres, modification de l'état d'un processus, alarme vers l'extérieur) doit être programmé dans un moniteur spécial. Ce moniteur a une procédure unique :

Signaler_exception (identificateur)

qui peut être exécutée par tous les moniteurs externes et processus du système. L'identificateur permet de déterminer de quelle exception il s'agit et donc quel algorithme exécuter.

On verra au chapître 4 comment ce moniteur est appelé et comment il peut être utilisé.

Structure du programme

Une des idées de base de la méthode était la définition d'une bibliothèque qui contiendrait les descriptions des masques des modules les plus utilisés, de façon à pouvoir les employer à nouveau dans d'autres applications.

De la même façon, les spécifications des modules constitueront une autre bibliothèque et elles pourront être insérées dans le programme de spécification d'un nouveau système.

La déclaration d'un "module bibliothèque" se fait par :

NOM_MODULE=library module

Programme :

instance :

Déclaration des modules et moniteur externes utilisés

```
NOM_Module : library module ;

NOM_MODULE = module ... end ;

NOM_MONITEUR = external monitor ... end ;
```

system :

Substitution des paramètres formels des "use-list" des modules

```
begin
    module_A(moniteur_B.lire,
             moniteur_C.écrire)
end ;
```

On peut reprendre l'exemple ANTIOPE en supposant que les modules de visualisation et clavier existent déjà dans la bibliothèque.

```
instance :   VISU_CCITT : library module ;
             CLAVIER_T16 : library module ;

             RECEPTION_ANTIOP = module
                      use (TAMPON.ECRIRE;PARAMETRE.LIRE) ;
             RECEPTION = process ... end ;
             CONTROLE = process ... end ;
             PARAM    = monitor ... end ;
             end ;

             INTERPRETEUR = module
                      use (SOURCE.LIRE,RESULTAT.ECRIRE) ;
               INTER = process ... end ;
             end ;

             FIFO = external monitor ... end ;

             MRAFRAICH = external monitor ... end ;

             PARAMETRE = external monitor ... end ;

system :
          RECEPTION_ANTIOP (FIFO.ECRIRE, PARAM_RECEP.LIRE) ;
          INTERPRETEUR (FIFO.LIRE, MRAFRAICH.ECRIRE) ;
          VISU_CCITT (MRAFRAICH.LIRE ; PARAM_VISU.LIRE) ;
          CLAVIER_T16 (PARAM_RECEP, PARAM_VISU) ;
end ;
```

La compilation du programme permet de vérifier statiquement les droits d'accés des différents éléments et de détecter les violations des règles de cohérence du langage.

Il est aussi relativement facile de faire une simulation fonctionnelle pour tester si la description correspond aux actions réelles que l'on voulait décrire. Sur ce dernier point il faut évidemment être prudent car dans un programme où il existe plusieurs processus concurrents, on ne peut pas prétendre simuler toutes les possibilités de fonctionnement. Une simulation bien planifiée permet cependant d'éliminer les erreurs de logique dans les parties séquencielles et de détecter des erreurs de programmation dans les moniteurs.

Cahier des charges

Le programme peut constituer aussi une partie importante du cahier des charges car toutes les fonctions à exécuter par les differents éléments du système y sont décrites avec précision.

Pour compléter le cahier des charges, le concepteur doit évaluer les contraintes temporelles (logiques) des Entrées/sorties ainsi que leurs caractéristiques électriques.

2.5. MONITEURS EXTERNES AYANT DES STRUCTURES PLUS COMPLEXES

Réservation

Dans la description système qui vient d'être faite un moniteur externe peut être accédé par tous les processus qui le référencient dans leurs listes d'accés.

Ceci ne pose pas de problème si l'utilisation du moniteur peut être multiplexée dans le temps.

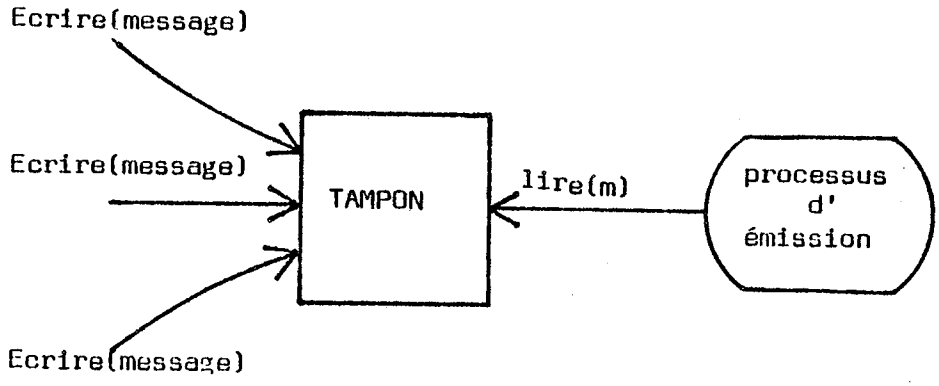


Fig. 2.11 - Processus d'émission

Dans l'exemple, présenté dans la figure 2.11 il ne faut prendre aucune précaution lorsque les messages ont une taille fixe, inférieure à celle du tampon, et les procédures lire et écrire s'exécutent dans une seule opération.

Cependant si les données du message ne sont pas transférées dans une seule opération mais dans plusieurs accés, il faut garantir qu'aucun processus ne pourra écrire dans le moniteur avant que le message soit complet.

Un sous-système controlant une imprimante est un autre exemple où les limitations physiques imposent le besoin d'un mécanisme dynamique de restriction d'accés. Supposons que ce sous-système est composé d'un moniteur où sont déposées les pages à imprimer et d'un processus qui lit ce moniteur et exécute ensuite l'algorithme de contrôle correspondant à l'impression des caractères. Etant donné que le moniteur ne peut avoir la même taille que les fichiers, il faut le réserver de façon à ce qu'un processus puisse imprimer toutes les pages d'un fichier en séquence.

Un mécanisme qui puisse permettre à un processus de reserver l'utilisation d'un moniteur, pendant un certain temps, est donc nécessaire. Ceci nous a amené à considérer la notion de "moniteur réservable" qui aurait en plus de ses procédures habituelles deux procédures "Réserver" et "Libérer" qui permettent à un processus de l'allouer et de le désallouer.

Avant d'accéder aux données d'un moniteur de ce type le processus utilisateur doit obligatoirement réserver son usage exclusif.

Dans les procédures d'accés aux données, l'identité d'un processus appelant est vérifiée et si le test est négatif, ceci signifie que ce processus a violé les règles de réservation et un appel au moniteur de traitement des exceptions est déclenché.

Interblocage

L'introduction de la possibilité d'allouer un moniteur fait apparaître le problème de l'interblocage qui peut arriver dans tous les systèmes où il y a au moins deux moniteurs réservables et deux processus qui ont besoin de les réserver simultanément pour pouvoir progresser dans leur exécution.

On a voulu traiter ce problème d'une façon très simple car dans un système comme MOSAIC qui a un nombre réduit d'éléments (processus et moniteurs) la probabilité que la situation précédente se produise, est relativement faible.

On utilise une politique de prévention en établissant un ordre partiel dans lequel les réservations doivent être faites.

Cet ordonnancement doit être établi par le concepteur en fonction des vitesses et des priorités des tâches associées aux moniteurs réservables.

Une méthode d'allocation de ce type entraîne un manque de souplesse du système car les ressources moins prioritaires risquent d'être bloquées pendant le temps où les plus prioritaires sont accédées.

Dans les exemples traités, les besoins de réservation se sont faits rarement sentir et on n'a aucune application où il y aurait un risque d'interblocage. Il nous faudrait donc des exemples concrets dont les performances puissent être évaluées avec les méthodes proposées au chapitre 3 pour que l'on puisse voir l'influence de cette politique d'allocation sur les performances du système et si d'autres méthodes (ex.: allocation par groupes de ressources) seraient nécessaires.

Un dernier aspect qui nous semble minimiser les problèmes d'interblocage est la décision initiale d'utiliser dans le contrôle des E/S des modules dont les processus sont dotés d'une certaine intelligence qui leur permet d'exécuter des requêtes plus complexes, limitant donc les besoins de réservation car certaines actions peuvent s'exécuter dans une seule opération.

Appels de moniteurs imbriqués

Pour implémenter certaines variables de communication, il est commode d'utiliser des structures hiérarchiques de moniteurs. Supposons l'exemple suivant :

- Un ensemble de trois files de messages avec des priorités croissantes :

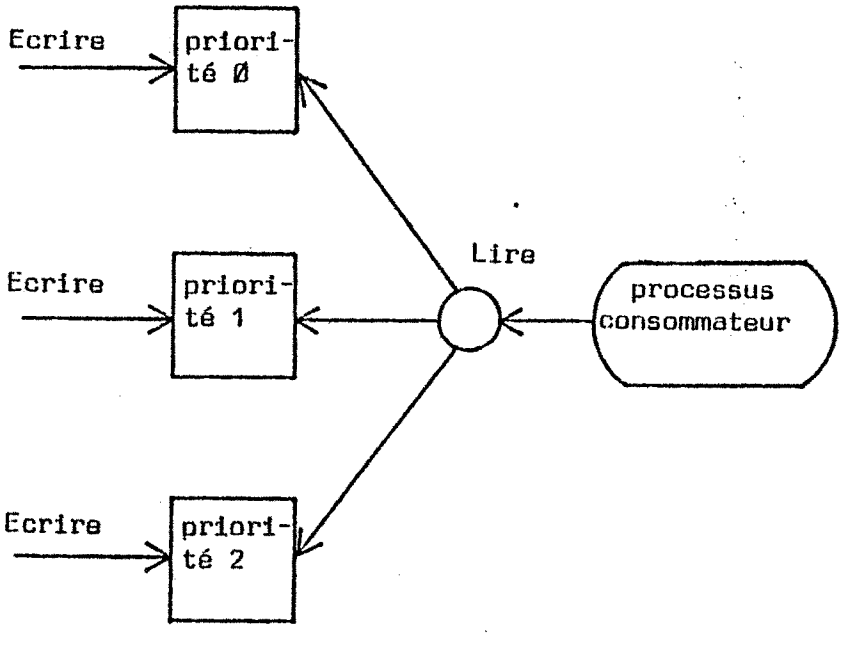


Fig. 2.12 - File de messages avec priorités

Quand un processus utilisateur déposer un message, il l'écrit dans la file correspondante à son niveau de priorité. Le problème se pose pour le processus lecteur qui doit être bloqué s'il n'existe aucun message dans aucune des files et qui doit être réveillé dès qu'un message est écrit dans n'importe quelle file.

Une solution attractive est de considérer la structure de la figure 2.13 où un moniteur "priorité" contrôle les files élémentaires.

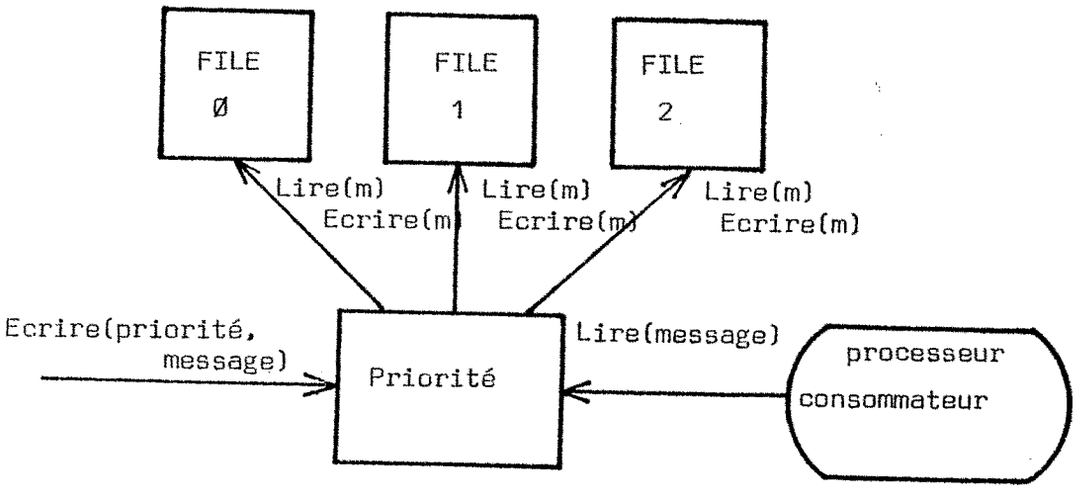


Fig. 2.13 - Hiérarchie de moniteurs

La programmation d'une telle structure pourrait être :

```
instance : FILE [0::4] : FIFO = external monitor
          ECRITURE_POSSIBLE:condition  (POINTEUR_ECR=POINTEUR_LECT∧MOD) ;
                              "déclaration-variables"
                    procedure entry ECRIRE(MESSAGE:mtype)
                    begin
                          wait (ECRITURE_POSSIBLE)
                            :
                            :
                    end ;
```

```
                procedure entry LIRE(MESSAGE:mtype,ETAT:boolean)
                begin
                        "algorithme de lecture"
                        ETAT:=(POINTEUR_LECT=POINTEUR_ECR ∧ ¬MOD)
                end ;
                begin
                        POINTEUR-LECT=POINTEUR_ECR:=0 ;
                        MOD:=false ;
                end ;
```

Moniteur priorité qui contrôle l'accés aux files élémentaires.

```
                PRIORITE = external monitor (FIFO0,FIFO1,FIFO2,FIFO3)
                begin
                        VIDE array [0::4] of boolean ;
                        IND : integer ;
                        LECTURE_POSSIBLE:condition ¬(VIDE(0)∧VIDE(1)∧ VIDE(2
                        ETAT:boolean                          ,VIDE(3)) ;
                procedure entry  LIRE(M:mtype)
                begin
                        wait LECTURE_POSSIBLE
                        IND:=0 ;
                        while VIDE(IND) do IND:=IND+1 ;
                        case IND of
                        0 :  FILE0.lire(M,ETAT) ;
                        1 :  FILE1.lire(M,ETAT) ;
                        2 :  FILE2.lire(M.ETAT) ;
                        3 :  FILE3.lire(M.ETAT) ;
                        end ;
                        VIDE(IND):=ETAT
                end ;
```

```
        procedure entry ECRIRE(M:message ; NIVEAU:integer)
        begin
                VIDE(NIVEAU):=false ;
                case NIVEAU of
                0 : FILE0.ECRIRE(M);
                1 : FILE1.ECRIRE(M);
                2 : FILE2.ECRIRE(M) ;
                3 : FILE3.ECRIRE(M) ;
        end ;
        begin
                FOR IND=0 until 3 do VIDE(IND):=true ;
        end ;
system:         PRIORITE(FILE(0),FILE(1),FILE(2),FILE(3)) ;
                        ¦
                        ¦
                        ¦
        end ;
```

Si cette structure est programmée de la façon décrite ci-dessus, un problème d'interblocage apparaitra car un processus bloqué dans la condition "ECRITURE_POSSIBLE" du moniteur libérera le verrou d'accés de "FILE(i)" mais maintiendra le verrou de "PRIORITE" ce qui empêchera d'autres processus d'entrer dans le moniteur et donc d'effectuer une lecture dans FILE(i) qui est la seule opération permettant de libérer le processus initial.

Ce problème a déjà été signalé par Lister et Manyard [LIS.76,LIS.77] et la solution proposée consiste à considérer qu'un "wait", à n'importe quel niveau d'une chaîne d'appels imbriqués à des procédures des moniteurs libère tous les verrous détenus par le processus.

Cette solution n'est facilement implémentable que si l'on considère qu'il y a une exclusion mutuelle globale pour tous les moniteurs [LIS.76].

Les structures matérielles proposées au chapître 3 pour l'implémentation des moniteurs externes permettent, avec certaines précautions, de faire des appels imbriqués de moniteurs, car la mutuelle exclusion est indirectement faite sur tous les moniteurs (§ 3.2.3).

Sur les deux derniers points il faut faire une remarque importante.

Dans cette section sont décrits les détails du formalisme des moniteurs qui ont suscité des problèmes dans la réalisation des grands systèmes d'exploitation basés sur ce concept. Il nous semble cependant que dans MOSAIC, ces problèmes ont une importance réduite car la technologie d'intégration disponible ne permettra qu'un nombre réduit d'éléments (processus, modules) à l'intérieur d'un circuit et ceux-ci auront besoin de variables de communication (moniteurs externes) peu sophistiqués.

Il nous a donc semblé qu'il fallait faire le tour des problèmes système susceptibles d'apparaître en utilisant un tel formalisme, mais que les problèmes avec une faible probabilité d'occurrence ne devaient pas être traités par des mécanismes lourds, matériels ou logiciels, qui risqueraient de pénaliser toutes les autres applications.

CHAPITRE 3

ARCHITECTURE GLOBALE

Pour rendre plus claire l'exposition du formalisme système on a présenté au chapître 1 le mécanisme de communication qui nous semblait mieux adapté à ce type de circuit.

Dans ce chapître on va traiter plus en détail le problème de la définition de l'architecture global du circuit, en essayant de justifier les solutions prises. Une démarche descendante qui part du formalisme système décrit précédemment nous permet d'analyser plusieurs hypothèses d'architecture qui sont évaluées en fonction des critères qui nous ont semblé prioritaires pour le type de circuit envisagé.

3.1. CRITERES DE CHOIX DANS UNE APPROCHE DESCENDANTE

Dans une approche descendante il est très important que les critères d'évaluation soient bien définis de façon à ce que les décisions à prendre puissent être clairement justifiées. Il est cependant habituel que, dans son ensemble, les objectifs choisis conduisent à des incompatibilités au niveau des solutions matérielles. Il faut donc, pour chaque décision à prendre, effectuer un dosage des différents critères de façon à ce que les solutions prises ne soient pas trop éloignées de l'ensemble des objectifs initiaux.

Dans MOSAIC les critères qui nous ont semblé les plus importants sont:

1 - Facilité d'implantation

Comme on l'a déjà mentionné dans les circuits très complexes, ce critère sera d'une très grande importance car le temps d'implantation représentera la plus grande partie du temps de conception. Ce paramètre doit dont être pris en compte dans l'évaluation des solutions d'architecture.

Des solutions matérielles pouvant être facilement implantées ou même implantables automatiquement doivent être préférées à des solutions obligeant à utiliser un dessin manuel.

La modularité des organes matériels est aussi très importante car il faut que d'éventuelles modifications dans une partie du système puissent être faites sans influencer le reste du circuit.

2 - Surface

Dans un circuit intégré, comme d'ailleurs dans la plupart des systèmes informatiques, un des objectifs principaux est la diminution des coûts de fabrication.

Dans les machines classiques ce problème est traditionnellement associé au nombre de composants utilisés. Par contre, dans un circuit intégré, ce paramètre n'est pas lié au nombre d'éléments qui le constituent mais à la surface de silicium occupée qui conditionne directement le rendement de production (nombre de circuits bons par waffer).

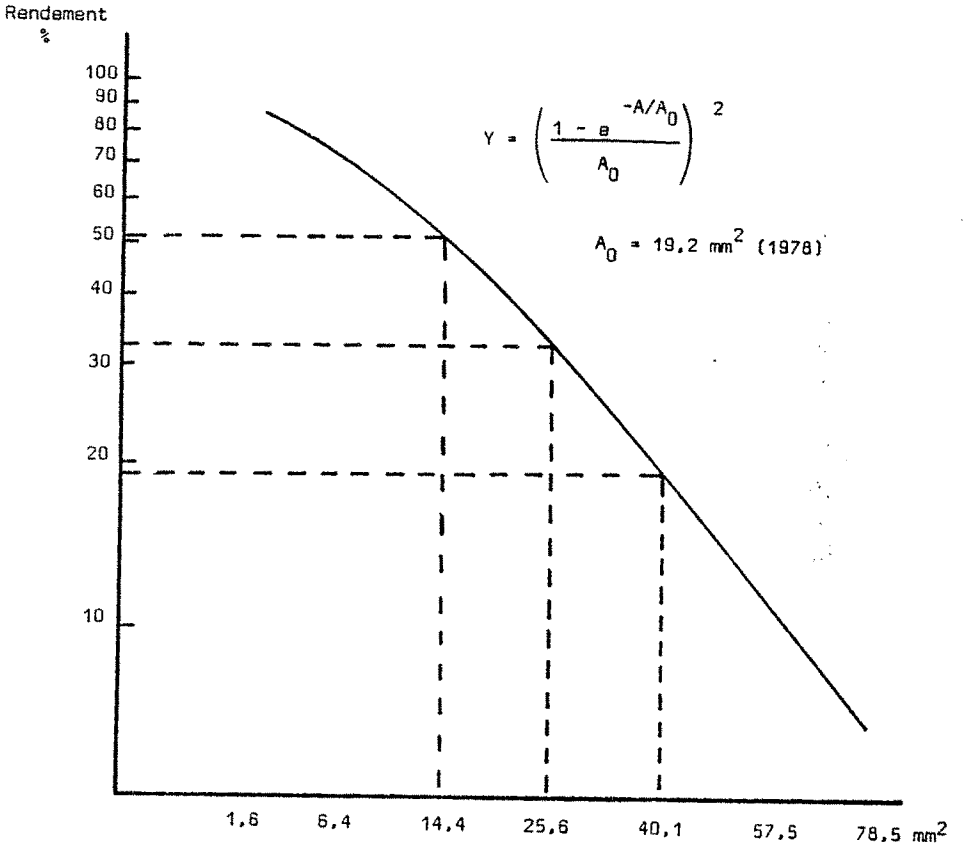


Fig. 3.1. - Rendement de production [ICE-78]

L'optimisation de la surface de silicium occupée suppose normalement l'utilisation de solutions d'architecture non régulières où chaque élément est dimensionné individuellement de façon à occuper la plus petite surface possible.

Cet objectif est donc en contradiction avec celui cité précédemment surtout parce que l'implantation de structures non régulières (logique anarchique) ne peut être faite convenablement qu'en les dessinant "à la main".

L'augmentation des densités d'intégration permet de diminuer l'influence du critère surface et il nous semble que le premier objectif cité sera dans le futur beaucoup plus prioritaire.

Il faut cependant que parmi les différents hypothèses d'architecture qui conduisent à des structures facilement implantables on choisisse celles qui occupent le moins de surface sinon on risque de proposer des solutions faciles à implanter mais sans aucune viabilité industrielle dans les technologies envisageables à court terme.

3 - Performance

L'aperçu des applications envisagées montre clairement que l'on ne cherche pas à concevoir des machines avec une grande puissance de calcul (au sens de calcul mathématique), mais des machines adaptables à un vaste ensemble d'applications ne nécessitant qu'une puissance de calcul moyenne et d'un nombre très élevé de fonctions élémentaires (associées au contrôle des E/S).

Bien que la performance ne soit pas notre critère principal, il faut cependant prendre en compte les contraintes temps réel imposées par certains organes d'E/S, surtout parcequ'elles risquent de devenir de plus en plus importantes dû à l'augmentation des performances des supports de transmission de données et des périphériques de mémoire de masse. D'autre part, certaines applications de contrôle industriel imposent aussi des fréquences d'échantillonnage très élevées.

3.2. HYPOTHESES D'ARCHITECTURE

3.2.1. Architecture monoprocesseur

Quoique a priori sans intérêt pour les raisons énoncées dans 1.2 on pouvait implémenter les mécanismes système décrits au chapître 2 en utilisant une architecture monoprocesseur. L'intérêt de la mentionner est qu'elle permet de limiter le spectre des solutions d'architecture envisageables qui seront examinées dans ce chapître.

Plusieurs exemples de systèmes multiprogrammés utilisant des moniteurs ont été décrits dans la littérature [HAN.75, LIS.76, LAM.80].

L'existence de processus concurrents est simulée par un multiplexage temporel des processeurs. Les processus sont activés par des interruptions provenant d'une horloge.

Quand un processus exécute une procédure d'un moniteur, les interruptions sont inhibées jusqu'au moment où il se bloque ou jusqu'à la fin de la procédure.

Dans MOSAIC chaque module constituerait un segment (au sens logiciel) à l'intérieur duquel seraient définis les processus et les moniteurs internes.

Chaque processus possède un espace d'adressage privé et il peut accéder soit aux moniteurs définis dans son segment, soit aux moniteurs externes définis dans un segment accessible par tous les processus.

Les désavantages d'une architecture monoprocesseur ont été amplement mentionnés dans 1.2, on ne fera donc que les rappeler :

- le débit du bus système doit être faible de façon à ce que tout le circuit puisse être compris dans la même zone isochrone,

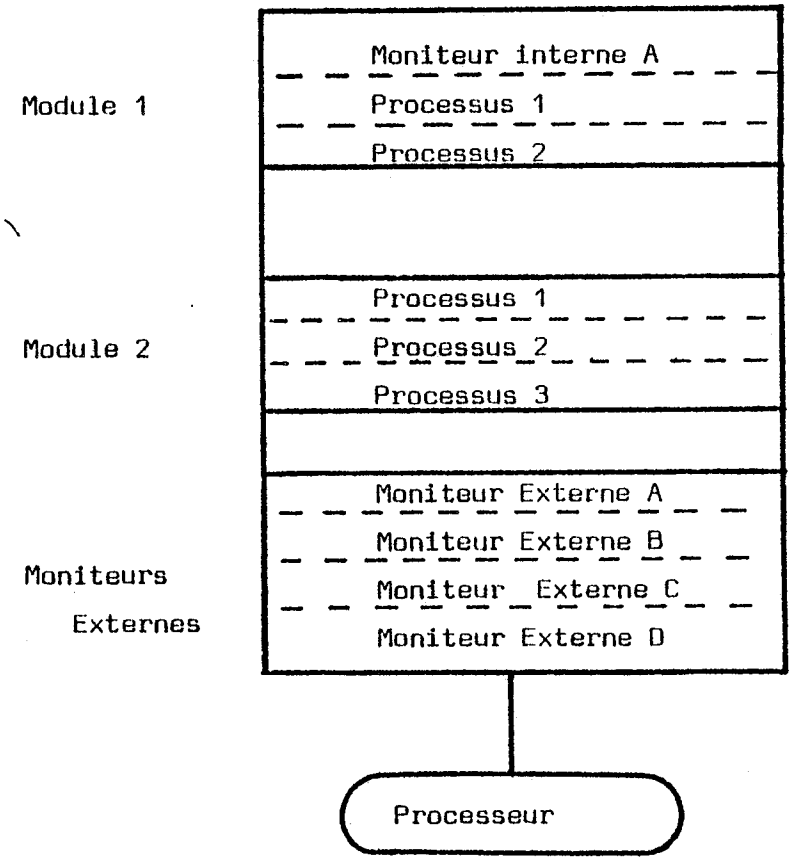


Fig. 3.2 - Schéma fonctionnel d'une architecture monoprocesseur

- performances conditionnées par le débit du bus mémoire/CPU,
- parallélisme simulé;
- système d'interconnexion non modulaire, occupant une surface importante,
- blocs de base (matériels) non modulaires difficilement assemblables.

3.2.2. Multiprocesseur à mémoire commune

Une mémoire commune est un mécanisme de communication bien adapté pour implémenter la stratégie de transfert asynchrone sous-jacente dans le concept de moniteur.

Une RAM est aussi une des structures les plus régulières dans la technologie des circuits intégrés et donc facilement implantable.

Les modules sont des blocs matériels indépendants à l'intérieur desquels s'exécutent les processus. Il n'est pas important à ce stade de la définition de connaître les détails de cette exécution, mais seulement qu'ils ont la possibilité d'allouer le mécanisme de communication (bus + mémoire commune) pour exécuter une procédure d'un moniteur externe.

Un allocateur permet aux processus qui le sollicitent d'accéder au bus commun (adresses, données, contrôle) en exclusion mutuelle.

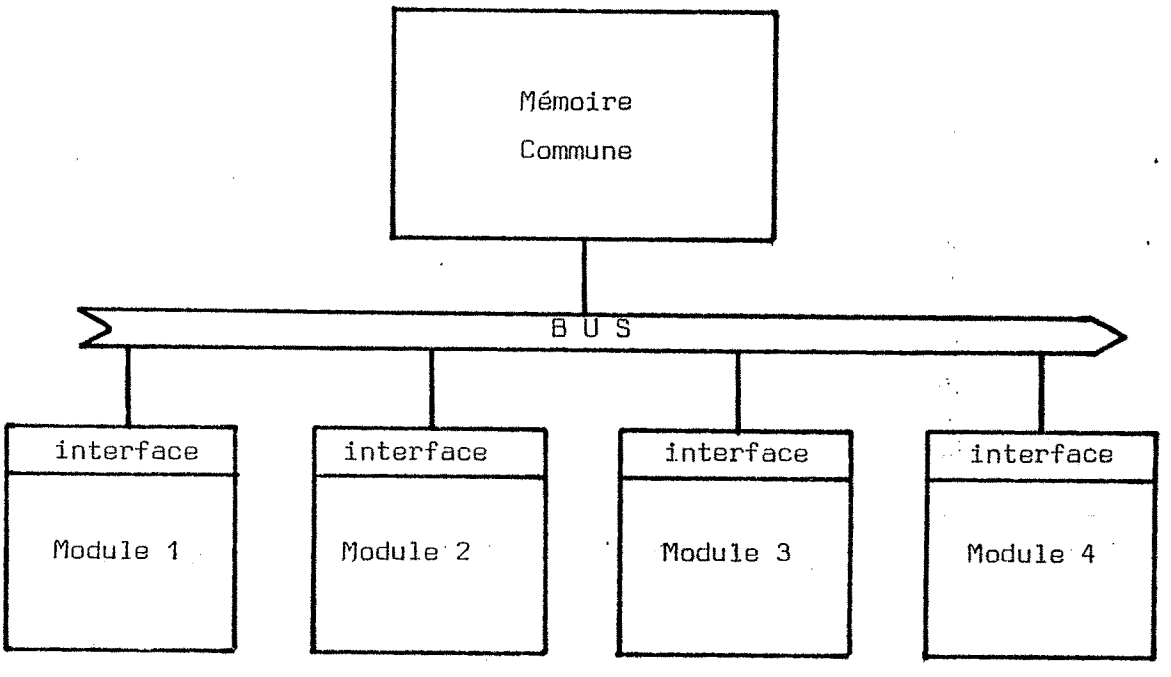


Fig. 3.3. - Schéma fonctionnel multiprocesseur à mémoire commune

Les moniteurs externes sont représentés dans la mémoire commune par des descripteurs qui réunissent toutes les informations sur leur utilisation (pointeurs, flags, compteurs, ...) et leur implémentation (adresse base, taille, ...).

Pour mémoriser les procédures et les données des moniteurs externes deux solutions peuvent être envisagées :

1 - mémoriser les données et le code des procédures dans la mémoire commune,

2 - ne mémoriser que les données dans la mémoire commune. Le code des procédures serait dupliqué dans les modules qui les utilisent.

Pour pouvoir analyser les avantages respectifs des deux solutions précédentes il faut considérer que le problème habituel de ce type d'architecture est le congestionnement du bus commun qui peut devenir le goulot d'étranglement du système.

Deux facteurs technologiques rendent ce problème plus aigu dans le cas d'un circuit intégré monoboîtier :

1 - la technologie de fabrication des processeurs et de la mémoire est la même ce qui interdit des solutions basées sur l'utilisation de processeurs lents et de mémoires rapides ;

2 - pour que la zone isochrone du bus puisse englober tous les éléments du circuit, il faut que la fréquence de transfert soit faible.

Analyse de la première solution

Un processeur qui veut exécuter une procédure d'un moniteur externe doit allouer le bus et ensuite travailler sur la mémoire commune comme sur sa mémoire locale.

Cette solution a deux désavantages importants :

1 - Les modules doivent avoir des parties opératives et des parties contrôle identiques pour pouvoir exécuter le code des procédures résidantes dans la mémoire commune.

On retrouve donc une architecture où les processeurs élémentaires sont identiques (particularisés par une mémoire programme) ce qui viole le principe initial (1.2) de considérer chaque module comme un processeur fonctionnel dont la structure interne serait optimisée (surface, temps d'exécution) dans l'exécution d'une fonction spécifique.

2 - Les accés des autres processus sont bloqués pendant le temps de lecture et d'exécution des instructions de la procédure.

Prenant en compte le nombre d'instructions nécessaires (code op. style microprocesseu 8 bits) pour exécuter les procédures des moniteurs décrits dans l'annexe 1, on a vérifié que cette solution n'est envisageable que si les processus ont dans l'ensemble une fréquence d'accés aux moniteurs externes inférieure à 1/20 de la fréquence de réponse du mécanisme de communication.

Analyse de la deuxième solution

L'exécution d'une procédure d'un moniteur se traduit par :

1ère partie
- Allocation bus
- Réservation du descripteur par un verrou (cellule de mémoire)
- Lecture du descripteur
- Libération bus

Exécution de l'algorithme de la procédure

2ème partie
- Allocation bus
- Ecriture des champs du descripteur modifiés
- Ecriture/lecture données
- Libération bus

1 - Par rapport à la solution précédente, le mécanisme de communication est utilisé pendant beaucoup moins de temps, cependant pour des cas peu favorables on peut être obligé d'effectuer 10 accés mémoire pour un seul octet transféré.

2 - Quoique les modules puissent avoir des structures internes différentes on est obligé d'introduire dans leur définition les contraintes imposées par l'exécution des procédures des moniteurs, or certains modules (controleur clavier , controleur visu) n'ont pas besoin, pour exécuter leurs fonctions spécifiques, de parties opératives et de parties contrôle classiques qui seraient nécessaires pour l'exécution des procédures des moniteurs.

D'autre part, la duplication du code des procédures entrainerait une augmentation de surface non négligeable.

Les deux solutions ne nous semblent pas correspondre aux critères initiaux car elle obligent à une duplication des organes matériels (plus important dans la deuxième solution) et imposent un "overhead" (plus important dans les première solution) considérable dans l'utilisation du bus commun.

3.2.3. Processeur noyau

Les problèmes de modularité et d'optimisation de la surface peuvent être résolus si l'on couple à la mémoire commune un processeur spécialisé dans l'exécution des moniteurs externes.

Ce processeur qui devient le noyau de communication du système doit avoir une structure interne spécialisée dans l'exécution des procédures des moniteurs.

Un processus qui veut exécuter une procédure d'un moniteur externe envoie l'identification du moniteur et de la procédure au processeur noyau qui se chargera de l'exécuter et de fournir au processus appelant toutes les informations nécessaires sur le déroulement de l'exécution.

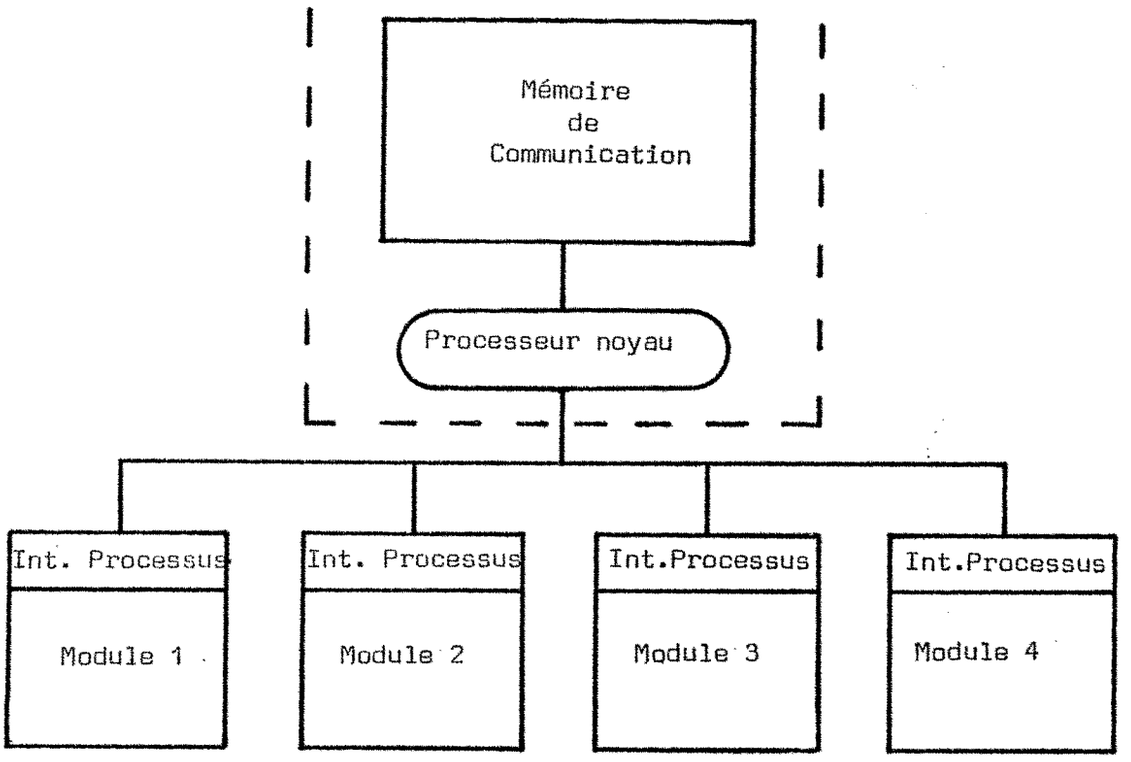


Fig. 3.4 - Schéma fonctionnel du système avec un processeur noyau

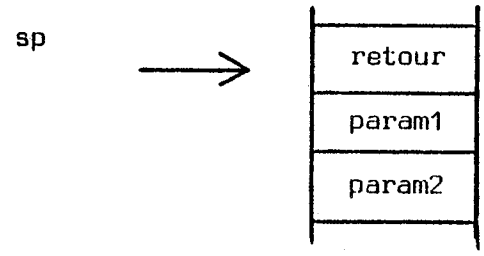
Du point de vue système cette solution correspond à avoir un processus dédié à l'exécution des moniteurs.

Dans l'implémentation classique des moniteurs, l'appel d'une primitive se traduit par un appel procédural.

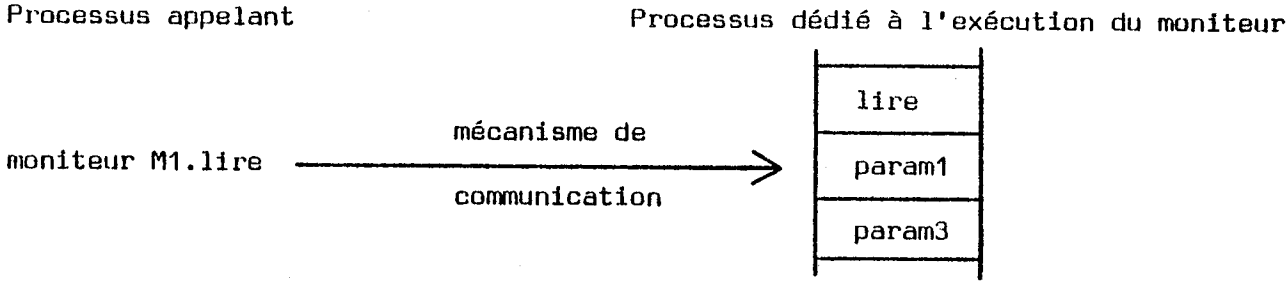
Le passage de paramètres se fait par l'empilement dans le contexte d'exécution du processus appelant de l'information de retour et des paramètres.

Contexte d'exécution du processus

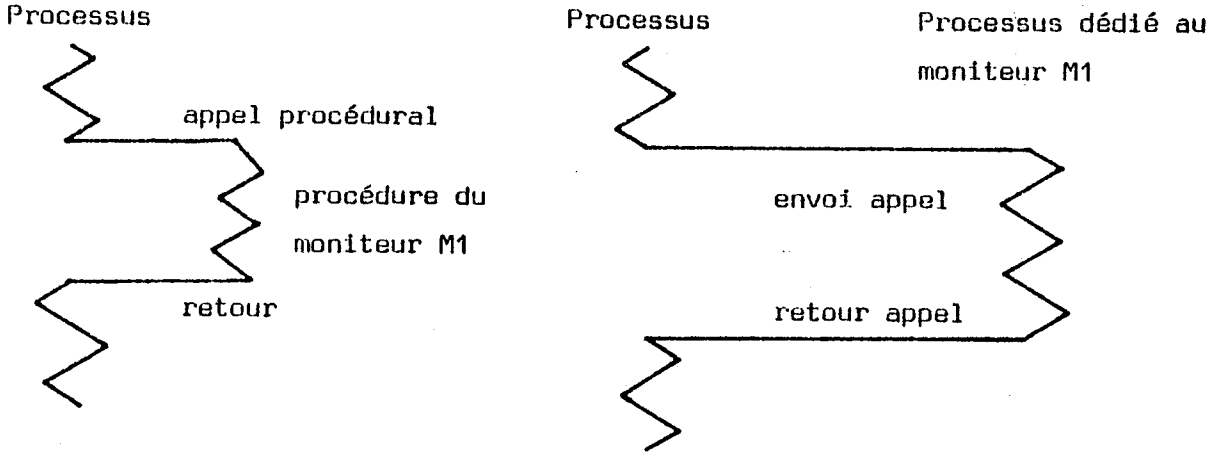
Moniteur1.lire (param1,param2)



Dans la solution proposée, les primitives des moniteurs sont exécutées par un processeur dédié. Dans ce cas le processus appelant doit envoyer sur le mécanisme de communication un message contenant l'information nécessaire pour déclencher la procédure et les paramètres, que le processus "dédié à l'exécution des moniteurs" rangera dans son contexte d'exécution.



Quoique très différentes du point de vue formel, ces deux implémentations peuvent être complètement transparentes pour l'utilisateur. Il suffit de garantir que le processus appelant reste bloqué jusqu'à ce que le processus exécutant le moniteur lui signale la fin de l'exécution de la procédure.



On est donc parti d'un mécanisme d'appel procédural au niveau du formalisme pour arriver à un mécanisme de communication par message entre processus au niveau de l'implémentation.

A ce stade de la définition on pourrait remettre en cause le choix du formalisme qui avait été fait et considérer qu'il serait préférable de définir une couche système basé sur un mécanisme de passage de messages.

On ne l'a pas fait car il nous semble que la différence entre les deux types de systèmes est plutôt une question de style qui n'a pas de conséquences réelles sur l'implémentation. Un travail de Lauer et Needharn [LAU.78] a montré d'une manière empirique que les deux types de système étaient duaux et qu'une transformation permet de passer de l'un à l'autre.

Finalement le mécanisme d'appel procédural est mieux compris et il existe plusieurs langages qui le supportent tandis que pour le mécanisme de passage de message on n'essaie que maintenant de l'introduire dans le langage [HOA.78, FEL.76].

La solution processeur noyau nous a donc semblé présenter plusieurs avantages que l'on peut systématiser selon les critères énoncés précédemment.

1/ Modularité

Les modules sont des blocs matériels indépendants du système dans lequel ils sont insérés. L'adaptation du mécanisme de communication est faite par simple programmation du processeur noyau.

Une interface standard permet aux modules de dialoguer avec le processeur noyau mais leur architecture interne n'est pas influencée par les problèmes concernant les variables de communication.

On retrouve donc au niveau de l'architecture le concept de type abstrait du domaine du logiciel.

2/ Facilité d'implantation

a) Les modules existant dans la bibliothèque peuvent être utilisés dans une nouvelle application sans être modifiés ;
b) La mémoire commune est facile à implanter et à évaluer pour une application donnée ;
c) L'architecture du processeur noyau peut être étudiée de façon à être facilement modifiable et implantable.

3/ Surface

a) Les modules n'ont pas besoin d'avoir des organes supplémentaires pour exécuter les procédures ;
b) La codification des algorithmes des procédures peut être optimisée car le code opération du processeur noyau peut être très puissant.

4/ Performances

a) Le bus commun est beaucoup moins utilisé, il n'y apas la lecture du code de la procédure ni du descripteur ;
b) L'architecture du processeur noyau peut être conçue de façon à optimiser le temps d'exécution des procédures.

Les raisons énoncées ci-dessus ont justifié une étude plus détaillée de cette solution de façon à pouvoir proposer les structures matérielles nécessaires pour son implémentation, ce qui sera l'objet des deux chapîtres suivants.

Quoique les temps d'exécution du processeur noyau soient relativement courts (6.2) cette solution reste une architecture à mémoire commune où il existe toujours une limite maximale du débit.

Bien que dans les exemples traités (annexe 6) les performances de cette solution centralisée soient suffisantes, il nous a semblé important de proposer des mécanismes plus performants qui évidemment imposeront une augmentation de la surface totale du circuit.

3.2.4. Hiérarchie des variables de communication

Certaines applications peuvent être sous-divisées en sous-systèmes faiblement couplés. Chaque sous-système sera composé d'un ensemble de modules et de moniteurs externes exécutés par un processeur noyau.

Des moniteurs globaux permettent de faire communiquer des processus appartenant à des sous-systèmes différents.

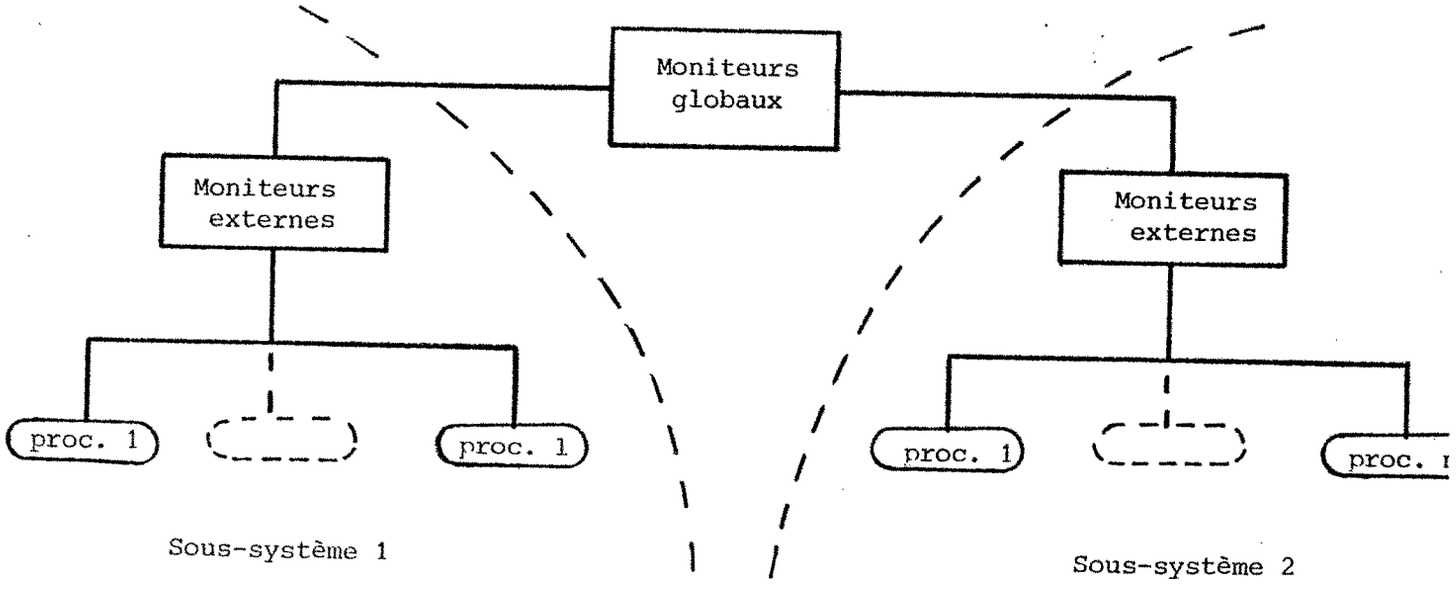


Fig. 3.5 - Schéma fonctionnel solution hiérarchique

Pour éviter d'utiliser un nouveau processeur noyau pour les moniteurs globaux ceux-ci sont exécutés par le processeur noyau de l'un des sous-systèmes. Quand les moniteurs globaux sont accédés un commutateur (voir chap.7) permet de relier les bus des deux sous-systèmes.

Cette solution permet d'utiliser les structures définies précédemment dans un contexte plus élargi où plusieurs processus peuvent travailler en parallèle.

Il est cependant évident que la duplication des processeurs noyau se traduit par une augmentation sensible de la surface totale du circuit.

L'utilisation de cette solution est fortement limitée par les deux facteurs suivants :

1 - La nature de l'application : la division en sous-systèmes doit correspondre à des propriétés de localité dans la communication inter-modules. Si les moniteurs globaux sont très utilisés, la performance de l'ensemble sera inférieure à celle d'une solution avec un seul processeur noyau.

2 - Le nombre de sous-systèmes : pour que l'implantation d'un circuit de ce type soit réalisable, il faut que le nombre de sous-systèmes soit très réduit (deux ou trois).

3.2.5. Solution distribuée

Correspond au cas limite de la fragmentation proposée dans la solution précédente.

La proposition qui a été faite en 3.2.3 de considérer un processus pour exécuter les moniteurs peut être généralisée de façon à considérer chaque moniteur comme un processus cyclique [LAU.78].

Moniteur

```
NOM-MONITEUR = monitor

A   "Déclaration variables
          et conditions"

    procedure entry PROC1 ("paramètres")
    begin

    end ;

B   procedure entry PROC2 ("paramètres"
    begin

    end ;

    begin
C        "initialisation"
    end ;

end ;
```

Processus

```
NOM-PROCESSUS = process

A   "Déclaration variables"
    M : message-type ;
    P : identification port ;
"initialisation"

C
    cycle
    begin
          [M,P] ← wait-message
          case P of
            PROC1 : begin

B                   end ;

            PROC2 : begin

                    end ;
    end ;
end ;
```

Cette généralisation du concept de moniteur permet dans le cas où ceci serait nécessaire de les distribuer et de les faire exécuter localement.

Cette solution est intéressante dans des applications où on a plusieurs processus temps réel ayant des fréquences d'accés très élevées à certains moniteurs externes.

Dans ce cas on peut implémenter ces moniteurs localement de façon à ce que les processus temps réel puissent les accéder sans utiliser le bus commun.

Les deux avantages suivants en résultent :

1 - une réponse presque immédiate est assurée aux requêtes des processus couplés aux moniteurs ;
2 - l'utilisation du bus commun est considérablement diminuée.

Pour implémenter localement les moniteurs, deux solutions sont possibles :

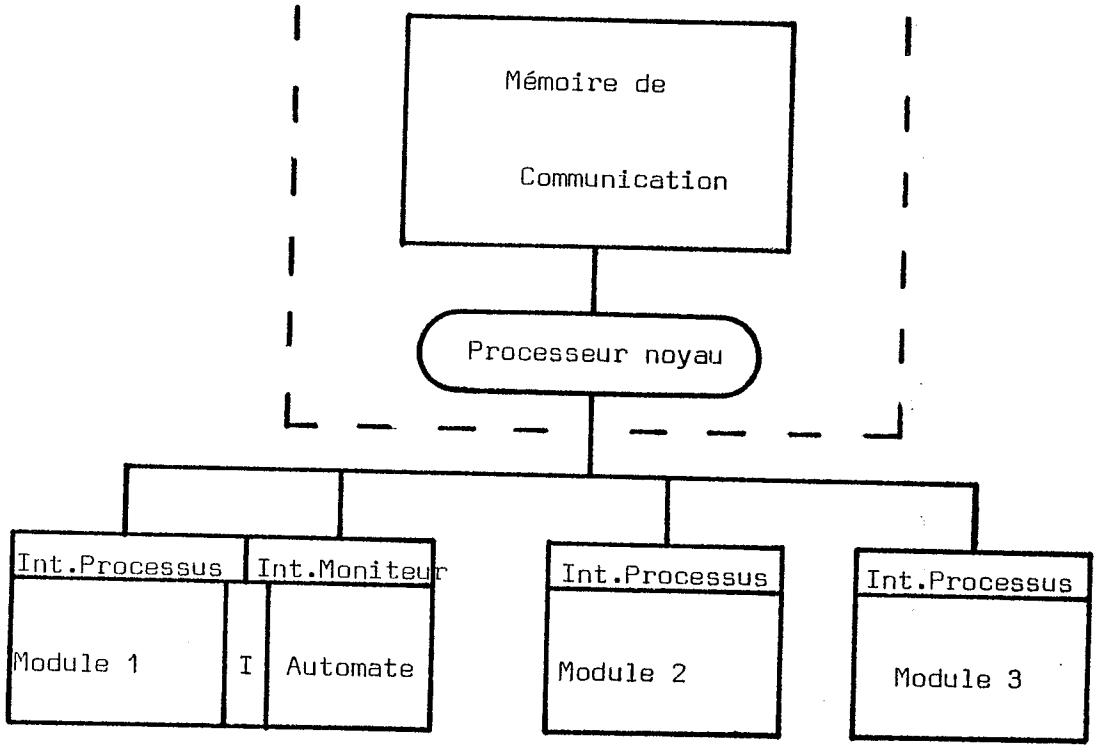
1 - faire exécuter le "processus-moniteur" dans la même structure matérielle (partie opérative + partie contrôle) où est exécuté le processus qui l'utilise le plus fréquemment ;
2 - concevoir un automate matériel qui exécuterait les algorithmes des procédures des moniteurs.

La première solution oblige à multiplexer la structure matérielle où s'exécute le processus, ce qui est en contradiction avec les conditions initiales du problème, c'est-à-dire des processus temps réel travaillant à des fréquences élevées.

Cette solution ne présente donc d'intérêt que dans des cas très particuliers, par exemple pour implémenter des moniteurs qui sont rarement utilisés, ou qui ne sont accédés qu'à des moments où les processus sont arrêtés (initialisation du système, paramètres rarement modifiés, etc...).

La deuxième solution est par contre beaucoup plus intéressante car il est relativement facile de réaliser de petits automates à base de PLA's qui permettent d'implémenter localement les procédures des moniteurs.

Le désavantage d'une telle solution est qu'évidemment la surface du circuit augmente car on duplique les organes matériels nécessaires pour l'exécution des moniteurs.



3.6. - Schéma fonctionnel solution distribuée

3.3. ARCHITECTURE MOSAIC

Les trois dernières solutions proposées ont été définies de façon à utiliser le même protocole de gestion du bus commun. Pour un processus utilisateur le fait d'accéder à un moniteur implémenté dans le processeur noyau ou par un automate local est donc transparent.

Il est cependant évident que la solution avec un seul processeur noyau est celle où la surface occupée par les mécanismes de communication est la plus petite.

L'augmentation de performances obtenue par les deux autres solutions se fait donc au prix d'une augmentation, non négligeable, de la surface du circuit.

Les caractéristiques des mécanismes de communication proposés sont résumées dans la table suivante :

implémentation des moniteurs externes	condition d'utilisation	avantages	désavantages
processeur noyau unique	Moniteurs qui ne sont pas accédés avec des fréquences très élevées.	La surface occupée est relativement faible.	La performance globale du système est imposée par la vitesse du processeur noyau.
solution hiérarchisée	Systèmes décomposables en sous-systèmes faiblement couplés.	Parallélisme entre les sous-systèmes.	Duplication des processeurs noyau.
solution distribuée par des automates distribués	Moniteurs ayant un taux d'accés très élevé.	- Réduction de l'utilisation du bus commun. - Une réponse presque immédiate est garantie au processus couplé.	Augmentation notable de la surface du circuit.

3.4. EVALUATION DES PERFORMANCES

Pour pouvoir exploiter les différentes possibilités d'architecture et trouver le meilleur rapport performances/surface il faut créer des outils permettant d'évaluer d'une part les performances globales du mécanisme de communication et d'autre part la surface occupée par chaque solution.

Pour l'évaluation des performances on a utilisé comme outil de base QNAP (Queue Network Analysis Package) [MER.78], un programme qui a été conçu pour l'évaluation des performances d'architectures d'ordinateurs. Un langage de description permet de modéliser une architecture sous forme d'un réseau de files d'attente. Des mécanismes de synchronisation permettent d'exprimer les contraintes existant entre les différents éléments de la description.

Pour chacun des mécanismes de communication proposés on a programmé en QNAP les modèles correspondants de simulation.

Le concepteur peut décrire l'architecture choisie en utilisant ces modèles de base convenablement paramétrés.

Les paramètres à utiliser sont :

1 - Fréquence logique des E/S - Ces informations font partie du cahier des charges (2.4.3) ;
2 - Temps d'exécution des procédures des moniteurs externes implémentés soit dans le processeur noyau, soit par des automates. On verra donc dans le chapitre 5 comment ces paramètres peuvent être obtenus à partir du programme de spécification.
3 - Lois statistiques et temps moyen de traitement des processus à l'intérieur des modules :
    a/ modules déjà existants (biblitohèque) - Quoique dans QNAP il ne soit pas prévu l'utilisation d'une bibliothèque de modèles prédéfinis on pourrait

y incorporer cette possibilité de façon à pouvoir réutiliser les modèles de simulation des modules déjà dessinés.

b/ modules à dessiner - C'est le cas le plus difficile à traiter car on ne peut pas avoir une idée quantitative précise avant que les structures matérielles soient définies. Cependant, dans la grande majorité des cas on peut faire des approximations qui pourront être suffisantes si les caractéristiques du module ne sont pas critiques pour la modélisation du comportement du système.

Dans les chapitres suivants on présentera simultanément les propositions d'architecture et les modèles QNAP correspondants.

CHAPITRE 4

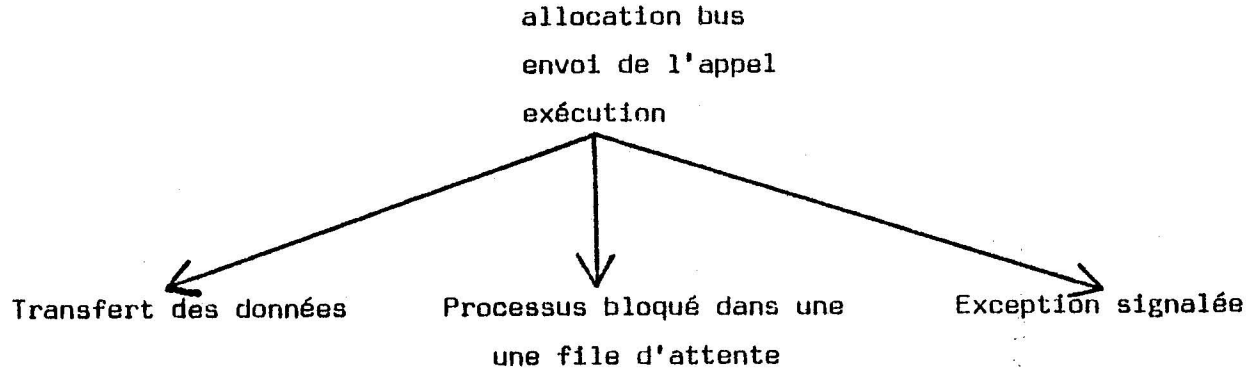
PROTOCOLE D'UTILISATION DU MECANISME DE COMMUNICATION

Les modules sont connectés au bus commun par une interface standard qui leur permet

- d'allouer le bus,
- d'envoyer/recevoir des données.

Pour simplifier la présentation on va considérer que chaque processus a sa propre interface, au chapitre 9 on verra la réalisation matérielle des interfaces et la façon de multiplexer leur utilisation parmi les processus du module.

L'exécution d'une procédure d'un moniteur par le processeur noyau ou par un automate peut se subdiviser en les étapes suivantes :



Avant de traiter chacune de ces étapes séparément, il faut analyser le problème de la distribution de l'horloge qui a été évoqué dans le chapitre 1.

Comme il a déjà été mentionné, le problème ne se pose pas à l'intérieur des modules qui peuvent être considérés comme des zones isochrones.

La définition de l'architecture interne des modules n'est donc pas influencée par ces problèmes ce qui permet d'utiliser les méthodes traditionnelles de conception des circuits synchrones.

Le problème se pose pour le dialogue entre les modules et le processeur noyau car la zone isochrone correspondant au bus commun doit être beaucoup plus large que celle des modules.

Deux solutions sont envisageables :

- un protocole de communication asynchrone :
Dans ce cas toutes les actions sont explicitement signalées par des changements dans les lignes de contrôle et aucune horloge n'est utilisée pour les séquencer.

Des protocole de communication asynchrone ont été définis pour des réseaux d'ordinateurs et pour des architectures multiprocesseur [MAR.77, SUT.79]. Cependant, de tels protocoles obligent à utiliser des organes matériels complexes et sont plus difficiles à formaliser.

- hiérarchie de zones isochrones :
Le circuit est composé par plusieurs zones isochrones (les modules, le processeur noyau), chacuneayant sa propre horloge adaptée à sa fréquence de travail (éventuellement la même).

Le bus commun et les interfaces avec les modules peuvent aussi être considérés une zone isochrone si l'horloge utilisée, pour synchroniser les différentes actions du protocole de communication, a une fréquence inférieure à la fréquence maximale admise dans les zones isochrones de taille inférieure.

Une hiérarchie de zones isochrones peut donc être établie, chacune ayant une fréquence d'horloge correspondant à sa surface (Fig. 4.1).

Il nous semble que dans le domaine du VLSI le problème des retards dans la distribution de l'horloge aura une très grande importance. Il faut cependant remarquer que pour les technologies envisageables à court terme, son influence est encore réduite.

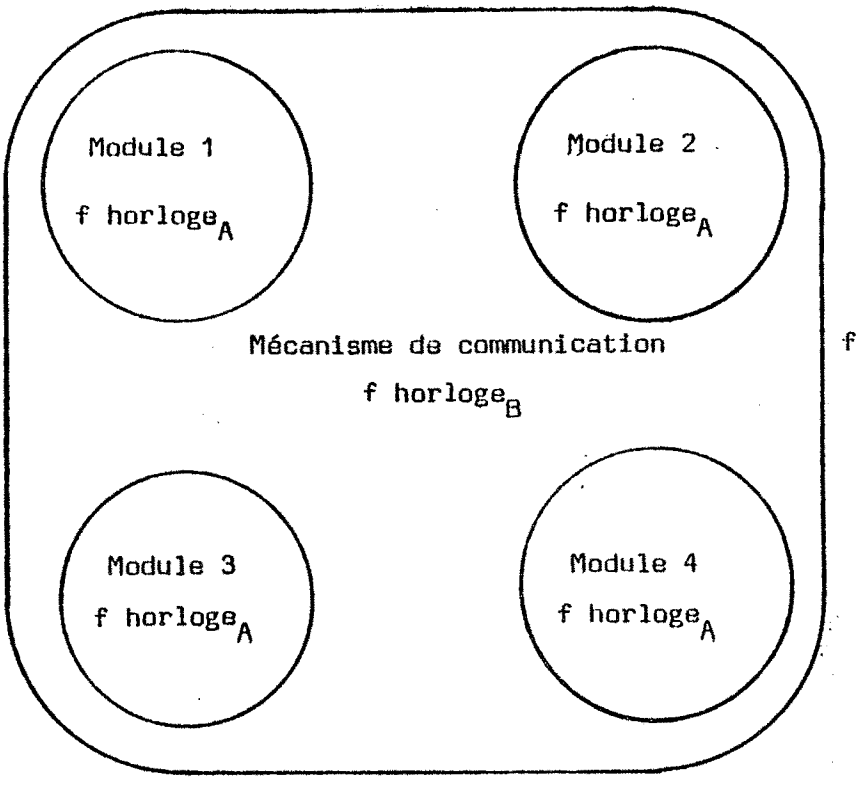


Fig.4.1 - Hiérarchie des zones isochrones

Un calcul très simple, basé sur les hypothèses de variation des paramètres avec le maigrissement de la technologie, présentée par Steiz dans [MEA.79], permet de vérifier que le problème ne se posera réellement que pour les technologies voisines de 1µm et submicroniques.

Si on considère λ le paramètre de maigrissement de la technologie, le temps de propagation dans un fil croît avec λ^2 (la capacité se maintient constante et la résistance croît avec λ^2 car la largeur et l'épaisseur du fil sont réduites).

Pour la technologie NMOS 1978 (6µm) Steiz donne un temps de propagation intrinsèque pour un fil d'aluminium de 10mm de 0,1 ns (par temps de propagation intrinsèque on entend la propagation d'un signal dans un fil qui n'attaque aucune capacité).

temps de propagation	technologie	facteur de réduction
0,1 ns	6µm	-
0,29 ns	3,5 µm	λ = 1,71
0,9 ns	2 µ	λ = 3
3,6 ns	1 µ	λ = 6

La proposition d'une hiérarchie de zones isochrones peut être utilisée pour les technologies à court et à long terme.

Une horloge rapide adaptée à la surface occupée par les modules (sensiblement la même) est envoyée à chacun d'eux. Les retards introduits par les lignes de distribution de l'horloge originent un déphasage entre les signaux reçus par chaque module.

Les actions du protocole de communication (exécutées par les interfaces) sont synchronisées par une horloge à fréquence plus basse obtenue par une division de la fréquence de l'horloge rapide envoyée aux modules (voir Fig.9.2).

Pour les technologies jusqu'à 2µm où les retards sont encore négligeables, l'horloge bus et l'horloge des modules est donc la même, pour les technologies plus poussées, il faut calculer le coefficient de division de la fréquence de façon à maintenir le temps de propagation intrinsèque des lignes de distribution de l'horloge considérablement inférieur à la période de l'horloge bus.

4.1. ALLOCATION

On peut considérer trois mécanismes d'allocation pour garantir l'exclusion mutuelle dans l'accés au noyau de communication :

1 - Allocateur réparti,
2 - Allocateur centralisé,
3 - Utilisation multiplexée dans le temps.

Allocateur réparti

a/ Priorité tournante

Un mécanisme très simple d'allocation répartie qui n'utilise que deux lignes peut être implémenté par une "Daisy chain".

Un jeton circule entre les modules donnant à chacun la possibilité d'utiliser le bus s'il en a besoin.

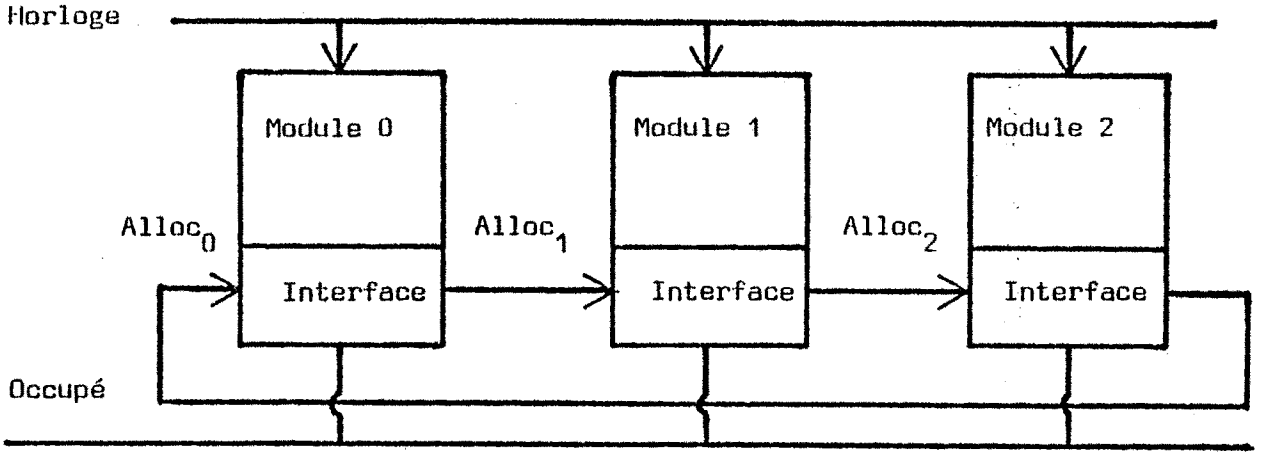


Fig. 4.2. - Allocateur réparti à priorité tournante

Cette solution,très simple,consiste à donner cycliquement à un processus la possibilité d'exécuter une procédure d'un moniteur et ne semble pas bien adaptée à certains types d'applications ayant des tâches temps réel avec des temps d'accés critiques, aux variables de communication.

Dans ce cas il faut prévoir un mécanisme permettant d'établir une liste de priorités qui donnera aux processus temps réel la possibilité d'accéder plus souvent au noyau de communication.

b/ Priorité fixe

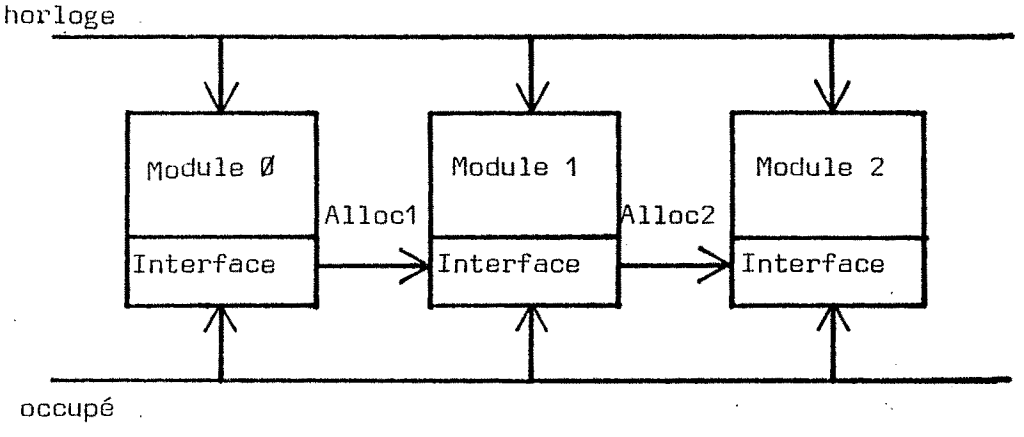


Fig. 4.3. - Allocateur réparti - Priorité fixe

La figure 4.3 décrit un schéma d'allocation à priorité fixe, la ligne "$Alloc_i$" au niveau bas indique à l'$interface_{i+1}$ qu'aucun processus de priorité plus élevée ne souhaite utiliser le bus.

La ligne "occupé" indique l'état d'utilisation du bus (niveau haut-occupé), la descente de ce signal permet de valider l'état des lignes "$Alloc_i$".

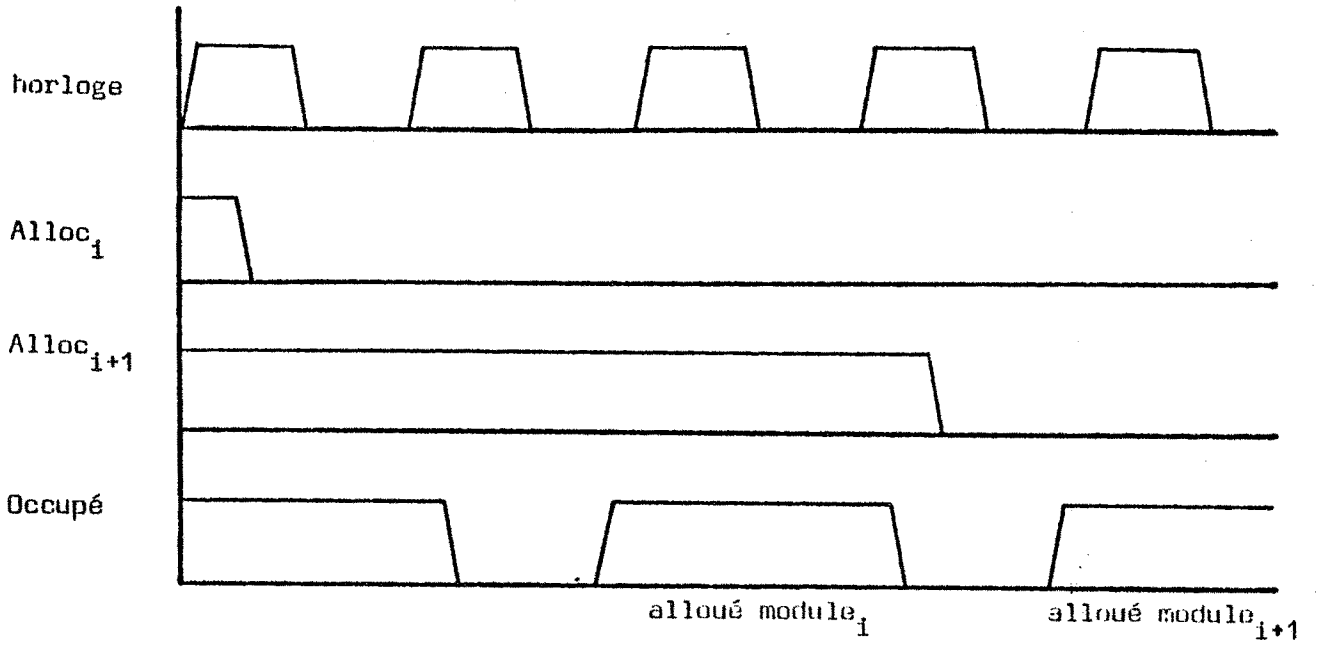


Fig. 4.4 - Chronographe de l'allocation

Le problème de ce schéma d'allocation est qu'il impose une disposition fixe des modules autour du mécanisme de communication, or on verra au chapitre 9 que la topologie du circuit est dictée par la surface des modules et donc deux modules consécutifs dans la chaîne d'allocation peuvent se retrouver éloignés dans l'implantation réelle.

Cependant, dans les exemples traités ceci n'est pas très contraignant car l'ordonnancement des priorités n'est pas une relation d'ordre stricte mais plutôt une division des modules dans deux ou trois classes de priorités différentes, ce qui permet plus de liberté dans le choix de leur positionnement dans la chaîne d'allocation.

Un deuxième désavantage de ce type d'allocation est l'utilisation d'une propagation série du signal d'allocation ce qui évidemment limite le nombre d'éléments de cette chaîne et oblige à une paramétrisation des portes commandant les lignes.

Les avantages sont :

- l'interface d'allocation est très simple (9.2),
- seulement deux lignes sont nécessaires.

Il est évident que ce schéma d'allocation ne garantit pas que tous les processus arrivent à être servis, une coalition des processus plus prioritaires peut empêcher les moins prioritaires d'accéder au bus. Il faut donc qu'il existe une certaine sous-utilisation du bus qui puisse permettre au processus moins prioritaire d'envoyer leurs requêtes.

La simulation utilisant QNAP permet de détecter ce type de problème.

Allocateur centralisé

Un allocateur centralisé présente l'avantage de pouvoir implémenter un schéma de priorités variables qui peuvent être modifiées en fonction de l'état du système.

Par contre, le nombre de lignes du bus augmente significativement (2 par interface) ainsi que la complexité du noyau car il faut y inclure l'allocateur.

La possibilité de travailler avec des priorités variables n'est pas très intéressante dans notre système où la plupart des tâches d'E/S ont des priorités bien définies imposées par les vitesses de transfert avec le milieu extérieur.

La souplesse d'utilisation qui peut permettre ce mécanisme ne nous semble pas compenser l'augmentation significative du nombre de lignes du bus (2N-2) et par conséquence de la surface totale.

Multiplexage temporel

Dans certaines applications des processus temps réel ont une fréquence d'accés aux variables de communication fixe (ou très peu variable) dans ce cas même le deuxième schéma d'allocation répartie peut ne pas être suffisant pour garantir la fréquence d'accés nécessaire car il n'y a pas de préemption du mécanisme de communication.

Un multiplexage temporel des mécanismes de communication peut se révéler une méthode intéressante pour traiter ce type d'application où l'on veut garantir une fréquence fixe d'utilisation du noyau de communication.

On considère que le temps d'utilisation du noyau de communication est alloué par tranches de temps de durée fixe. Une ou plusieurs tranches sont allouées (statiquement) à chaque processus en fonction de sa fréquence d'accés aux moniteurs et de la complexité des procédures exécutées.

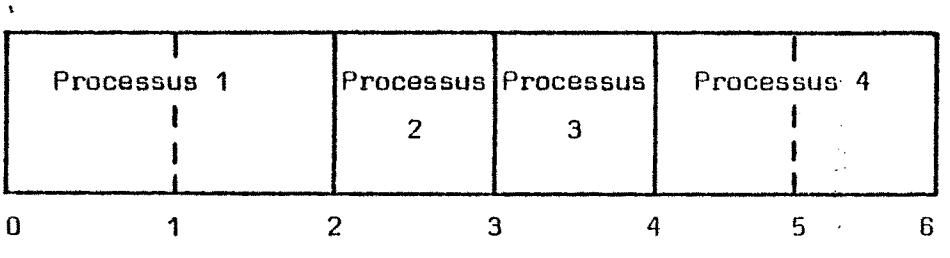


Fig. 4.5 - Allocation des tranches de temps au processus

Il est évident que cette solution n'est intéressante que si tous les processus ont des fréquences d'accés fixes, ce qui n'est pas le cas dans la majorité des applications.

Une solution mixte est beaucoup plus efficace. Des tranches de temps sont allouées statiquement aux processus temps réel avec des fréquences d'accés

stables, un ensemble de tranches est alloué aux autres processus dynamiquement par le mécanisme de "Daysy chain" décrit précédemment.

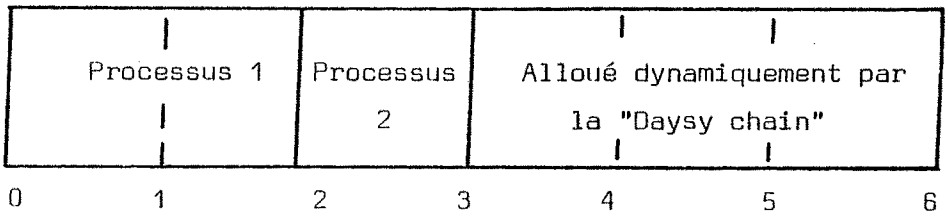


Fig. 4.6 - Schéma mixte multiplexage/allocation

Des interfaces implémentant les trois possibilités décrites ont été définies, le concepteur doit donc choisir, en fonction des caractéristiques de son application, le mécanisme d'allocation le mieux adapté.

Dans le tableau suivant on résume les caractéristiques principales des mécanismes d'allocation proposés.

	avantages	désavantages
Allocation répartie Priorité tournante	Tous les processus ont la même priorité sauf erreur il n'y a pas de famine.	Les processus temps réel n'ont pas de possibilité de travailler à des fréquences élevées.
Allocation répartie Priorité fixe	Un accés beaucoup plus rapide est garanti aux processus temps réel.	Problème de la famine.
Multiplexage temporel	On peut garantir une fréquence d'accés constante aux processeurs temps réel.	Dans les applications où les fréquences d'accés ne sont pas très stables, le temps d'utilisation du noyau de communication est mal exploité.

4.2. APPEL D'UNE PROCEDURE D'UN MONITEUR EXTERNE

Après avoir alloué le bus, le processus doit envoyer l'appel d'une procédure d'un ME (Moniteur Externe).

Il faut donc définir le codage des informations qui définissent l'appel :

- non moniteur,
- procédure à exécuter.
- paramètres :
  ex:. non processus pour vérifier la réservation,
  . déplacement (adressage direct),
  . données.

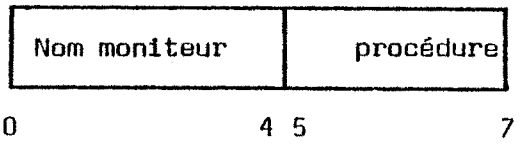
Pour définir un codage on doit imposer des limites raisonnables aux entités existantes dans le système.

Bus de données

La plupart de l'information véhiculée par le mécanisme de communication dans un tel système sont des caractères, il nous a donc semblé que le bus de données devait avoir 8 lignes, le transfert des mots de taille supérieure se décomposant en plusieurs transferts.

Cet espace d'adressage nous permet d'attribuer 5 bits pour le codage du nom moniteur et 3 pour la procédure ce qui fait un maximum de

32 moniteurs externes,
8 procédures par moniteur



Ces limites étaient largement surdimensionnées pour les applications étudiées, cependant, se le besoin se faisait sentir, des extensions peuvent facilement être faites, par exemple : bit 0 indiquant s'il s'agit d'un codage sur 1 ou 2 octets).

4.3. EXECUTION DES PROCEDURES

Après que le processus exécutant le moniteur ait reçu l'appel (non moniteur, procédure), l'exécution de la procédure peut se dérouler de manière :

- normale :
  - accés aux variables du moniteur.
- anormale :
  - non vérification d'une des conditions, ce qui oblige à bloquer le processus appelant,
  - exception (tentative de violation des droits d'accés, de l'espace d'adressage du moniteur, etc...),
  - erreur (tentative d'exécution d'un moniteur ou d'une procédure non existante dans le système).

Remarque

On sépare les deux derniers cas car ils ont des causes différentes, le premier peut avoir pour origine une faute de programmation ou être prévu pour être traité par le module de traitement des exceptions. Le deuxième correspond à une erreur matérielle qui ne peut pas provenir du programme de spécification car elle aurait été détectée par le compilateur.

On va commencer par traiter le cas du déroulement normal en supposant que le moniteur est exécuté par le processeur noyau et que son utilisation n'est pas multiplexée dans le temps. Les mécanismes matériels nécessaires pour le multiplexage temporel sont légèrement différents et ils seront analysés à la fin.

Avant de pouvoir accéder aux données du moniteur, le processeur noyau doit effectuer certains calculs (évaluation des conditions, calcul de l'adresse: base + pointeur, etc...) dont la complexité dépend de l'algorithme de procédure.

Pendant ce temps le processus appelant peut soit :

- maintenir le contrôle du bus,
- libérer le bus permettant à un autre processus d'envoyer une requête.

La deuxième hypothèse semble a priori plus performante, elle implique cependant un protocole d'utilisation du bus plus compliqué et un organe pour gérer la file d'attente dans le processeur noyau.

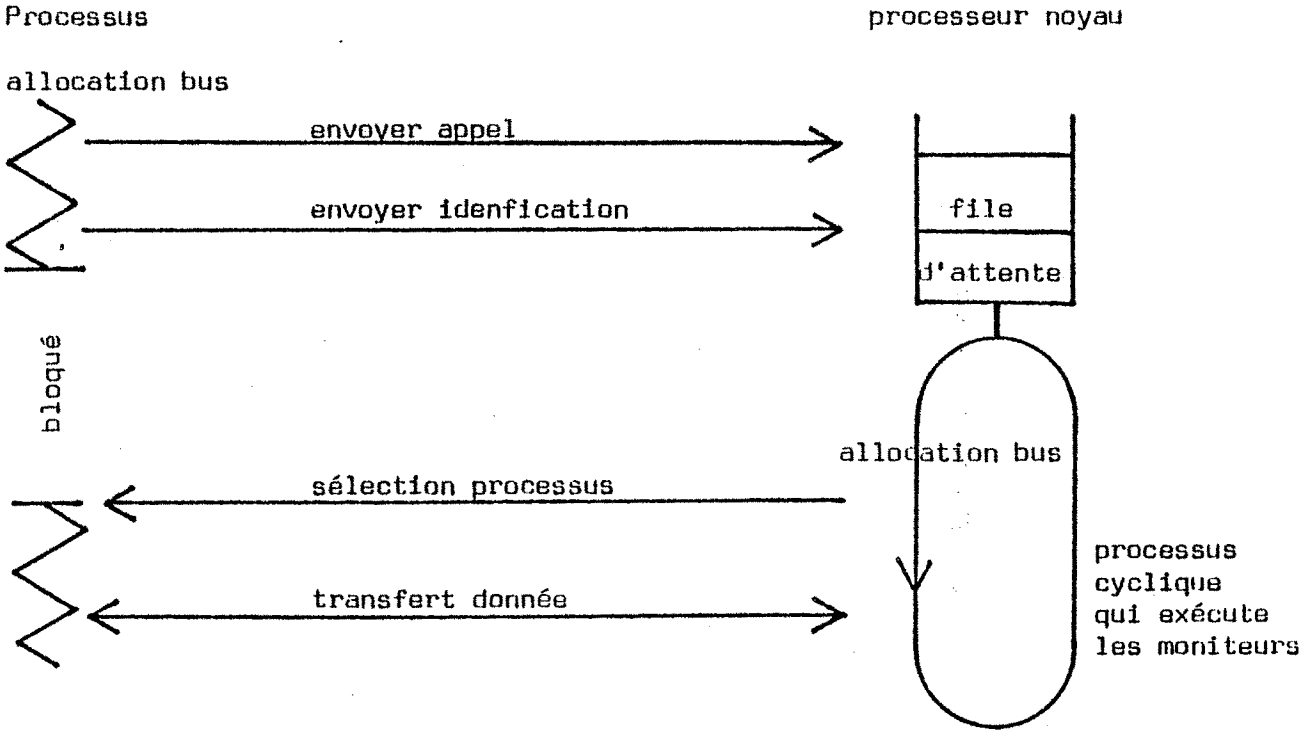


Fig. 4.7 - Libération du bus pendant le préambule de la procédure.

Après quelques tentatives de définition de l'architecture du processeur noyau on s'est rendu compte que les temps d'attente étaient relativement courts et donc qu'il ne se justifiait pas d'utiliser un mécanisme plus compliqué pour un gain de performance minimal. La première solution fut donc adoptée.

Pour que le processeur noyau puisse exécuter tout le préambule, il faut que les processus envoient avec l'appel les paramètres nécessaires tels que leur identification dans le cas d'un accés à un moniteur réservable, les déplacements dans le cas d'un adressage direct des données, etc...

Quand le processeur noyau est prêt à exécuter la partie d'accés aux données il le signale au processus par la ligne "Prêt", les transferts de données commenceront dans le cycle d'horloge suivant.

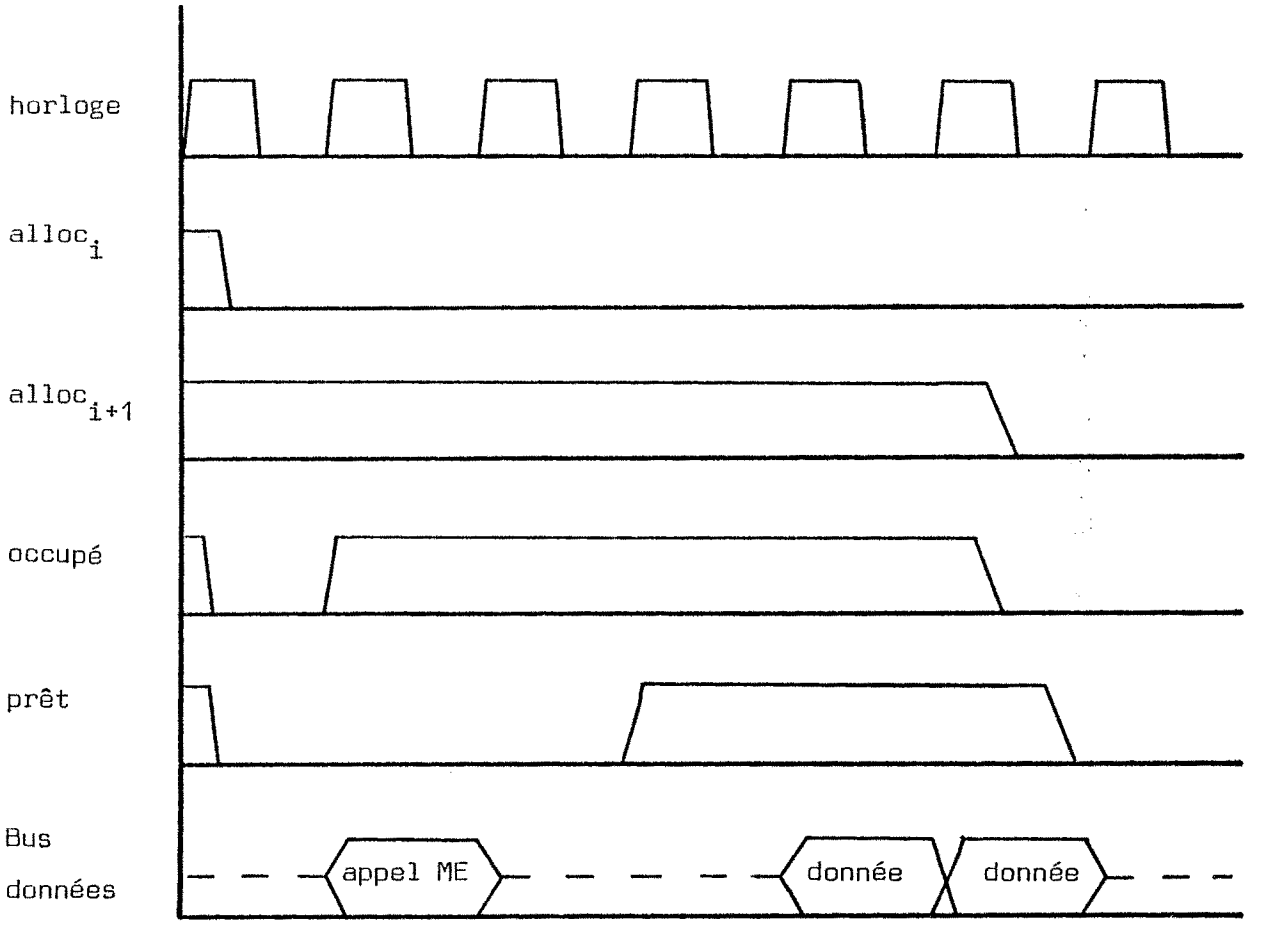


Fig. 4.8 - Chronographe du bus commun

La signification des lignes "occupé" et "prêt" est la suivante :

Occupé

bus alloué à un processus

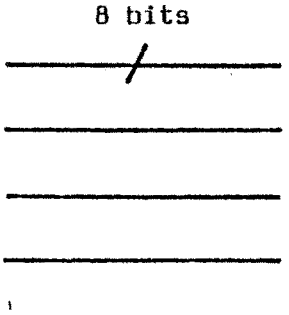
le processus libère le bus dans le cycle suivant

Prêt

le processeur noyau signale le début d'un transfert

le processeur noyau signale la fin de l'exécution de la procédure

Les lignes du bus sont donc :



Le deuxième cas peut poser certains problèmes pour le contrôle du séquencement des opérations de transfert sur le bus.

Cependant le nombre de paramètres qui interviennent dans l'évaluation des conditions est normalement réduit et de toute façon les conditions sont toujours évaluées avant d'accéder à l'information que l'on veut réellement transférer.

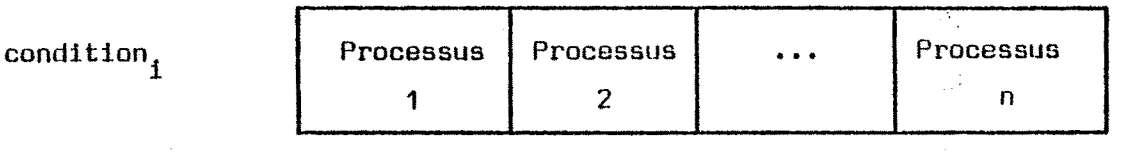
On peut donc simplifier le problème en considérant que le processus alloue le bus et envoie l'appel avec tous les paramètres nécessaires pour évaluation des conditions.

Ensuite il est soit bloqué, soit le processus noyau lui signale le transfert des données (Prêt).

Dans l'implémentation classique du mécanisme de blocage des processus, une file d'attente où sont mémorisés le nom et l'état des processus bloqués est associée à chaque condition.

File d'attente

Etant donné que le système est statique on pouvait associer à chaque condition un mot contenant un bit par processus qui indiquerait si le processus est bloqué (ou non) dans cette condition.



Cette solution présente deux désavantages :

- au point de vue surface - quoique le nombre d'entités du système ne soit pas très élevé, la mémoire nécessaire pour implémenter les files pourrait être de l'ordre de 100 octets (16 processus x 50 conditions). On verra dans 6.1 qu'une RAM de 100 octets n'est pas négligeable dans la surface totale du processeur noyau.

- au point de vue temps d'exécution - la recherche d'un processus en attente peut consommer beaucoup de temps si un test séquenciel des bits du mot condition est utilisé. La solution serait de créer un organe matériel qui détecterait le premier bit à un , mais ceci entraîne aussi une augmentation de la surface.

On a donc préféré d'utiliser une implémentation qui nous semble optimiser ces deux derniers aspects :

- les files d'attente ne sont pas maintenues dans le processeur noyau mais distribuées dans les interfaces des modules.

Quand un processus, qui a envoyé un appel, doit être bloqué on lui signale l'impossibilité d'exécution de la procédure par l'activation d'une ligne que l'on a appelée "attendre".

Simultanément, le numéro de la condition où il a été logiquement bloqué est envoyé sur les lignes du bus de données.

L'activation de la ligne "attendre" déclenche la mémorisation du numéro de la condition dans l'interface du processus qui a envoyé l'appel.

Quand les conditions internes du moniteur évoluent de façon à permettre le redémarrage du processus, une ligne "continuer" est activée et le nom de la condition envoyé sur le bus de données.

Dans l'interface une comparaison est faite, à chaque activation de la ligne "continuer", entre l'octet mémorisé et celui qui est envoyé sur le bus. Si la comparaison est vraie, le processus essaiera d'allouer le bus.

Dans le cas où plusieurs processus seraient en attente, le plus prioritaire selon le schéma d'allocation (ou le plus prêt dans la circulation du jeton) démarrerait. Les autres essaieront ensuite et selon le comportement de l'algorithme du moniteur ils seraient ou non de nouveau bloqués.

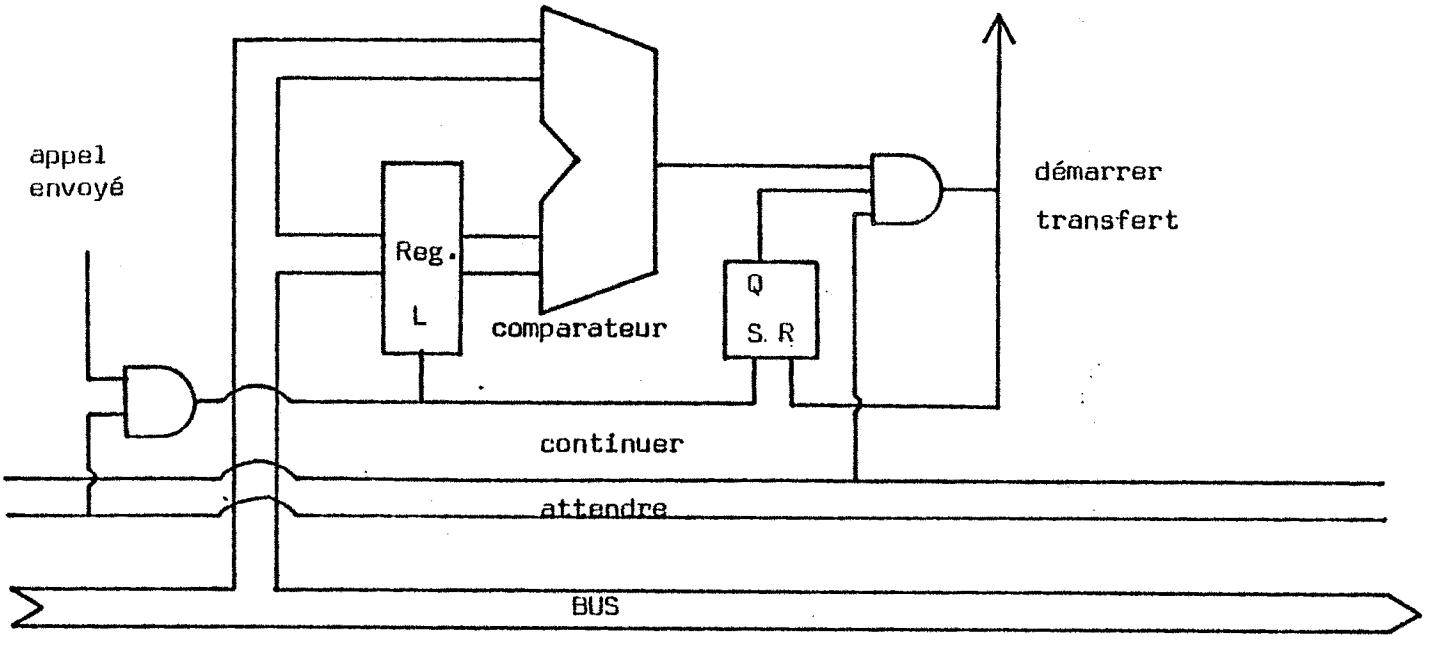


Fig. 4.9 - Files d'attente distribuées

Les avantages de cette solution sont :

1 - Rapidité - Les comparateurs des interfaces font une sorte "d'attente active" chaque fois que la ligne "continuer" est activée. Les comparaisons se déroulent donc en parallèle sans utiliser aucun autre organe du système.

2 - Surface utilisée - Il est très facile d'intégrer le comparateur dans le registre de façon à obtenir des blocs de taille réduite.

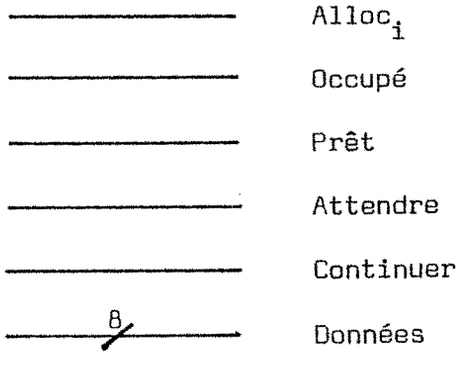
4.5. EXCEPTIONS ET ERREURS

Quand une exception est détectée par le processeur noyau il déclenche un appel à un moniteur spécial qui exécutera l'algorithme correspondant à l'exception détectée.

Les erreurs sont signalées de la même façon avec évidemment des identificateurs différents.

Pour des raisons que l'on verra au chapitre 9, ce moniteur est implémenté par un module spécial et pour l'appeler on envoie sur les lignes de contrôle une configuration qui n'était pas utilisée (occupé=1, prêt=0, continuer=1) et sur les lignes de données l'identificateur de l'exception ou de l'erreur.

Le bus est donc composé de 5 lignes de contrôle et 8 lignes de données :



Les états du mécanisme de communication sont :

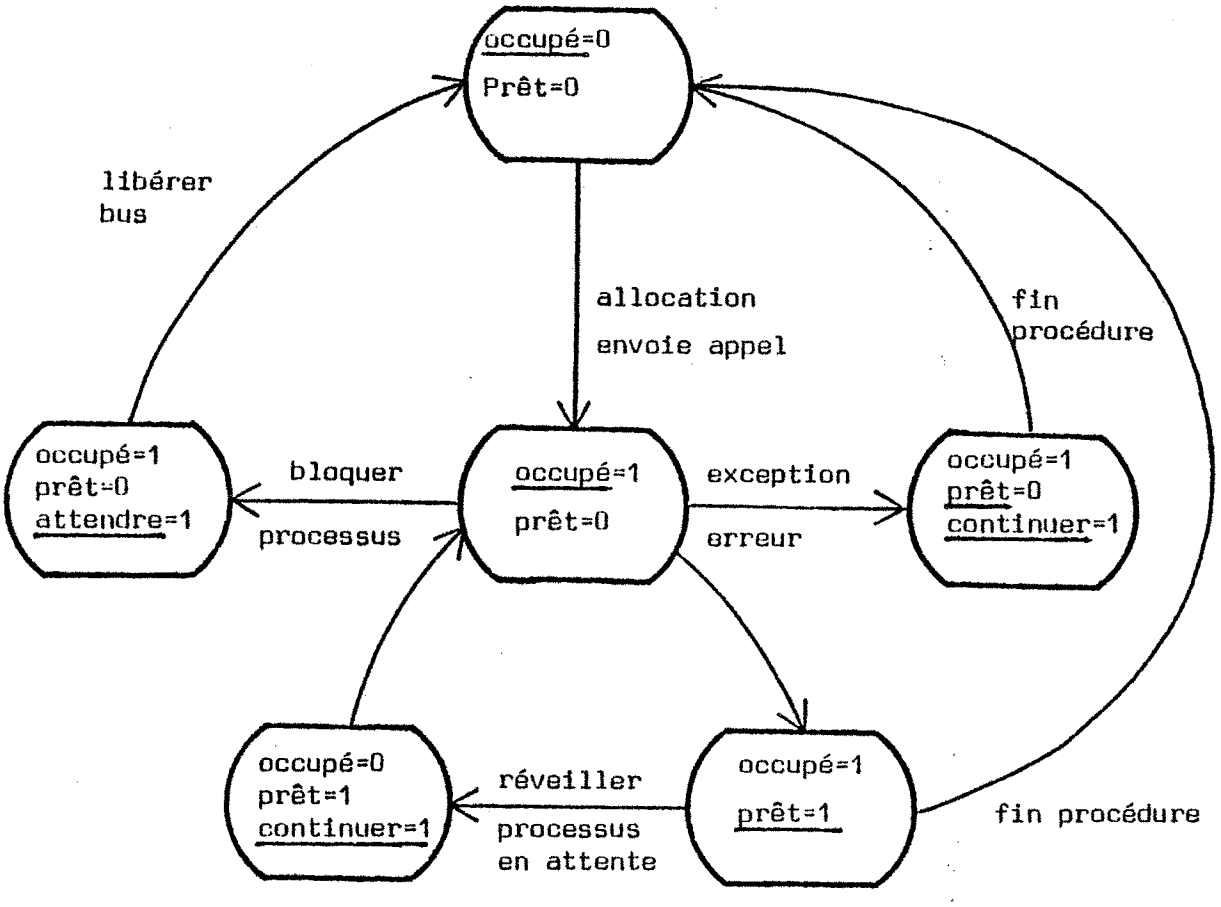


Fig. 4.10 - Etats du mécanisme de communication

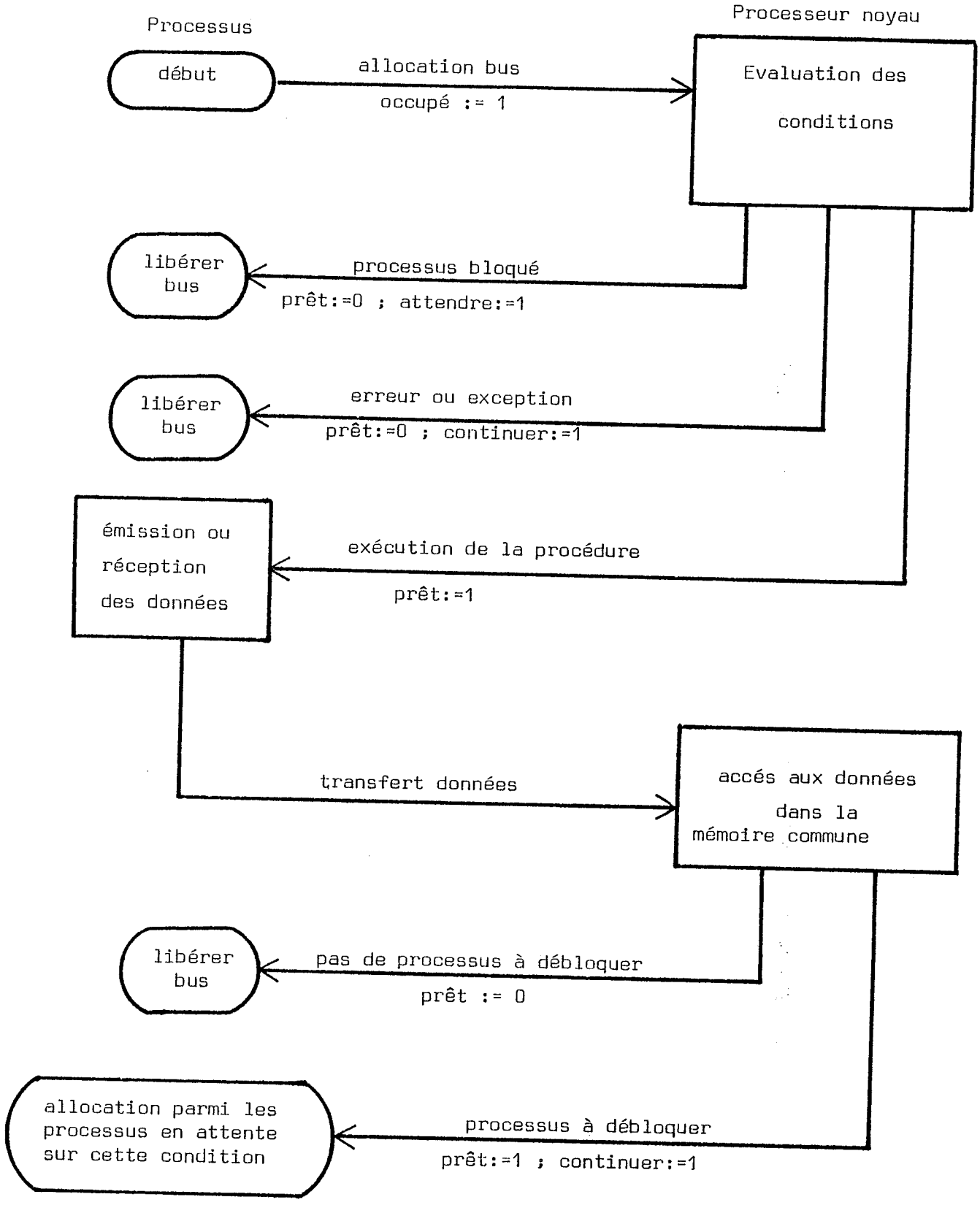


Fig. 4.11 - Dialogue processeur noyau/processus

4.6. MULTIPLEXAGE TEMPOREL

L'interface des modules possède un système de compteurs qui lui permet de détecter à quel moment du cycle le processus a le droit d'utiliser le bus commun.

Les procédures des moniteurs sont exécutées dans les tranches de temps appartenant aux processus.

Dans le processeur noyau il y aura un contexte d'exécution associé à chaque processus. Si un processus est en train d'exécuter une procédure dans le contexte correspondant sont mémorisés le nom du moniteur, la procédure et l'état d'avancement de celle-ci. Chaque fois que l'on passe des tranches de temps d'un processus à celles d'un autre, il se produit un changement de contexte.

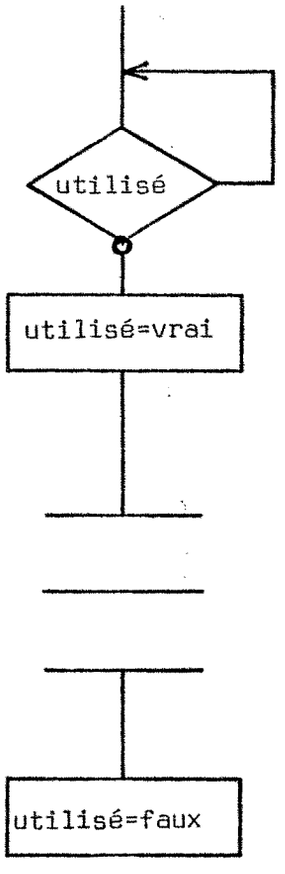
A l'exception de l'exclusion mutuelle dans l'accés aux moniteurs, qui n'est plus garantie par l'allocation du bus, tous les autres problèmes (blocage/ démarrage des processus, exceptions, erreurs) peuvent être traités de la même façon.

Pour garantir l'exclusion mutuelle il faut utiliser un mécanisme supplémentaire. Dans l'implémentation classique des moniteurs, ceci est fait en utilisant un verrou d'accés qui peut être une variable booléenne qui doit être lue et modifiée dans une seule opération (test & set).

On considère un booléen "utilisé" qui appartient au descripteur du moniteur, avant toute utilisation de ce moniteur, un processus doit tester ce booléen et le positionner dans la même tranche de temps.

obligatoirement exécuté dans
une seule tranche de temps

accés normal décomposé dans
plusieurs tranches de temps



Quand le moniteur est libéré, le processus en attente qui se trouve plus prêt dans le cycle de balayage pourra prendre le contrôle du moniteur.

Si beaucoup de conflits sont détectés dans l'accés aux moniteurs, l'efficacité du multiplexage temporel est considérablement réduite, car il y aura beaucoup de cycles inutilisés dus à l'attente des processus.

Une évaluation du système en utilisant ONAP permet de détecter facilement ce problème.

CHAPITRE 5

PROCESSEUR NOYAU

Le processeur noyau est un des éléments fondamental de l'architecture du système car il conditionne fortement

a/ **la facilité d'implantation** : les algorithmes d'accés aux variables de communication dépendent des applications. Il faut que ces modifications puissent être faites facilement et sans introduire des changements dans la topologie du circuit.

Les différents éléments du processeur noyau et de la mémoire de communication doivent être conçus de façon à être facilement imbricables.

b/ **la performance** : étant donné que les processus maintiennent le contrôle du bus,pendant l'exécution des procédures la vitesse d'exécution du processeur noyau imposera la performance globale du mécanisme de communication.

c/ **la surface** : si la surface occupée par le noyau est très importante on risque de pénaliser dès le départ la viabilité économique de la méthode.

5.1. REPRESENTATION INTERNE DES MONITEURS

5.1.1. Descripteur

Chaque moniteur est représenté par un descripteur où sont regroupées toutes les informations concernant son utilisation et son implémentation.

Dans l'annexe 1 on a programmé quelques uns des moniteurs externes les plus utilisés. Les variables décrites dans les structures de données fournissent déjà une indication relativement précise des éléments qu'il faut inclure dans le descripteur.

Les éléments présents dans le descripteur peuvent se diviser en deux groupes :

Informations sur l'implémentation :

Base : un adressage basé simplifie considérablement les algorithmes et diminue la taille des autres éléments du descripteur (pointeurs, taille).

Taille : nombre d'octets réservé pour les données du moniteur.

Unité d'information : l'analyse de la plupart des applications a montré que l'octet était l'unité d'information la plus utilisée, ce qui justifie la décision, prise dans le chapître 4, d'avoir un bus avec 8 lignes de données. Cependant, on ne peut pas forcer les processus à transférer un seul octet à chaque appel d'une procédure car ceci les obligerait à fragmenter et à réassembler l'information ainsi qu'à verrouiller les moniteurs pendant des transferts inachevés.

Dans l'appel à une procédure du moniteur, le processus doit donc pouvoir transférer l'ensemble des octets correspondant aux paramètres spécifiés dans sa description dans le "programme de spécification ".

Dans la plupart des moniteurs externes de MOSAIC, la structure de données passée en paramètre est la même pour toutes les procédures du moniteur (exemple: écrire(message) ; lire(message)). Dans le but de minimiser la taille des descripteurs on a considéré qu'un moniteur a une seule unité d'information (nombre entier d'octets) qui sera commune à toutes les procédures.

Si plusieurs unités d'information sont nécessaires, une structure plus compliquée sera utilisée (voir 5.2.2).

Informations sur l'utilisation :

Pointeurs : normalement, deux pointeurs suffisent, un pour la lecture et un autre pour l'écriture.

Flags : les flags permettent de mémoriser l'état du moniteur, elles sont particularisées pour chaque application, par exemple: Réservé, Vide, Plein.

Utilisateur_lecture, Utilisateur_écriture : dans le cas où un moniteur peut être réservé, on doit mémoriser dans le descripteur le nom de son utilisateur.

Compteurs : nécessaires pour mémoriser les processus en attente dans une condition.

5.1.2. Catalogue

Pendant l'exécution d'une procédure le descripteur du moniteur est accédé en moyenne 4 à 8 fois. Pour réduire le temps d'exécution des procédures on utilise une mémoire catalogue où tous les descripteurs des moniteurs sont regroupés.

Cette mémoire, beaucoup plus petite que la mémoire de communication aura aussi des temps d'accés plus courts.

Le noyau de communication est donc constitué par les quatre grands blocs représentés dans la figure 5.1.

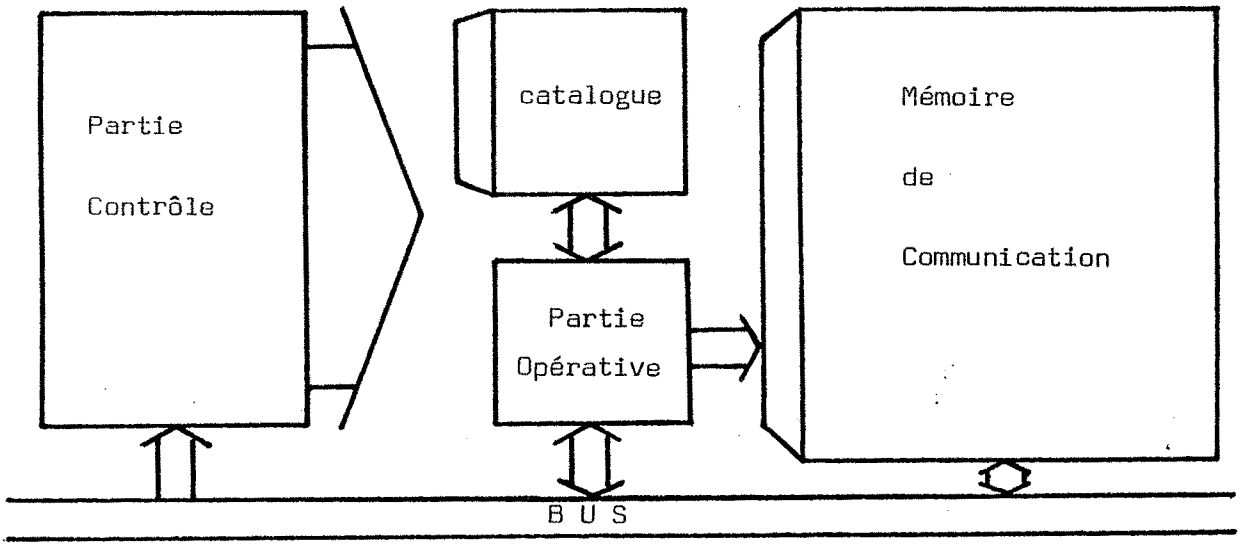


Fig. 5.1. - Schéma de blocs du noyau

5.2. Processeur

5.2.1. Architecture globale

Les appels aux procédures du moniteur (exemple: FIFOA.lire) constituent le langage d'entrée du processeur noyau. Ce langage doit subir une ou plusieurs étapes d'interprétation avant de pouvoir générer les impulsions électriques nécessaires pour activer les circuits séquenciels et combinatoires de la partie opérative.

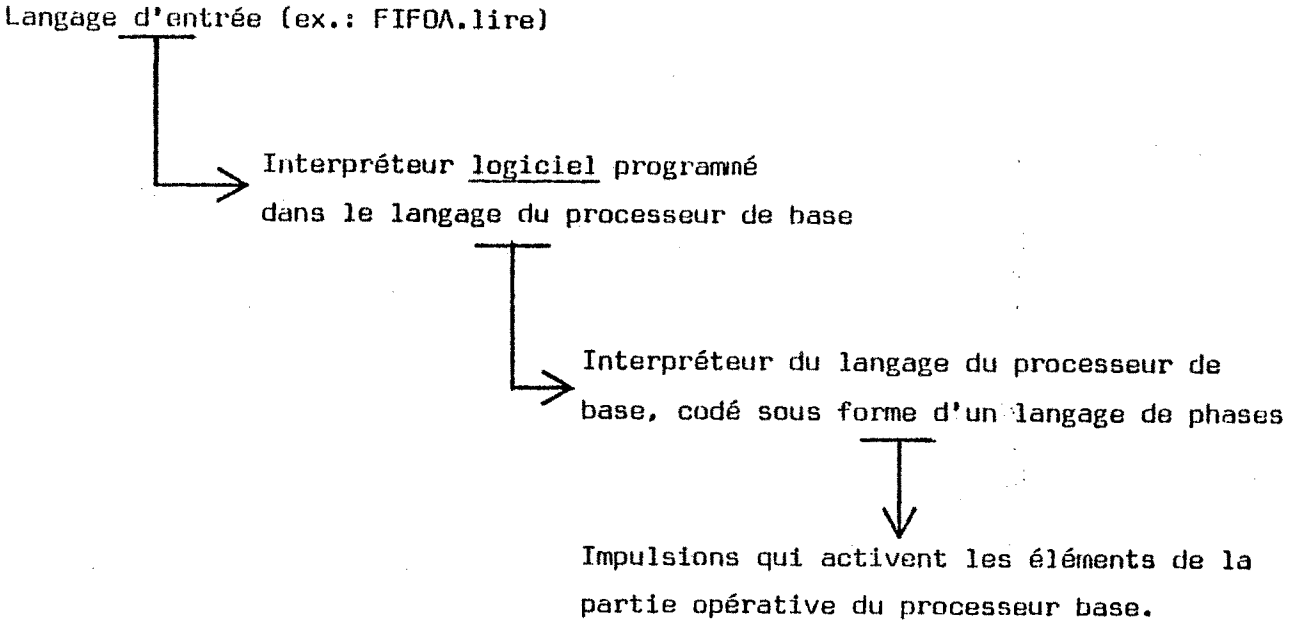
Pour la définition de l'architecture de la partie opérative et de contrôle du processeur noyau, deux structures sont envisageables :

a/ **Définir un processeur classique** (genre microprocesseur) avec un jeu d'instructions de base qui regroupe toutes les opérations nécessaires pour l'exécution des procédures (exemple: addition, soustraction de deux registres, incrémentation/décrémentation, test d'un bit dans un octet, etc...).

Le processeur serait composé d'une partie opérative classique (registre + UAL) et d'une partie contrôle relativement simple correspondant à l'interpréteur de son jeu d'instructions.

Les algorithmes des moniteurs seraient maintenus dans une ROM adressée par le processeur comme dans un système classique de microprocesseurs.

Les étapes d'interprétation pourraient se représenter de la façon suivante [SCH.77] :



Cette architecture a deux désavantages importants :

1 - le décodage logiciel est relativement lent car on est obligé de parcourir des tables d'indirection pour trouver la séquence correspondante ;

2 - des instructions très simples (style code.op.6800) exécutant un nombre réduit d'actions par instruction, ne sont pas bien adaptées pour programmer les algorithmes des moniteurs. D'une part, le code généré est très peu compact, d'autre part, le temps d'exécution est très élevé.

Pour résoudre ces deux problèmes on a donc choisi de :

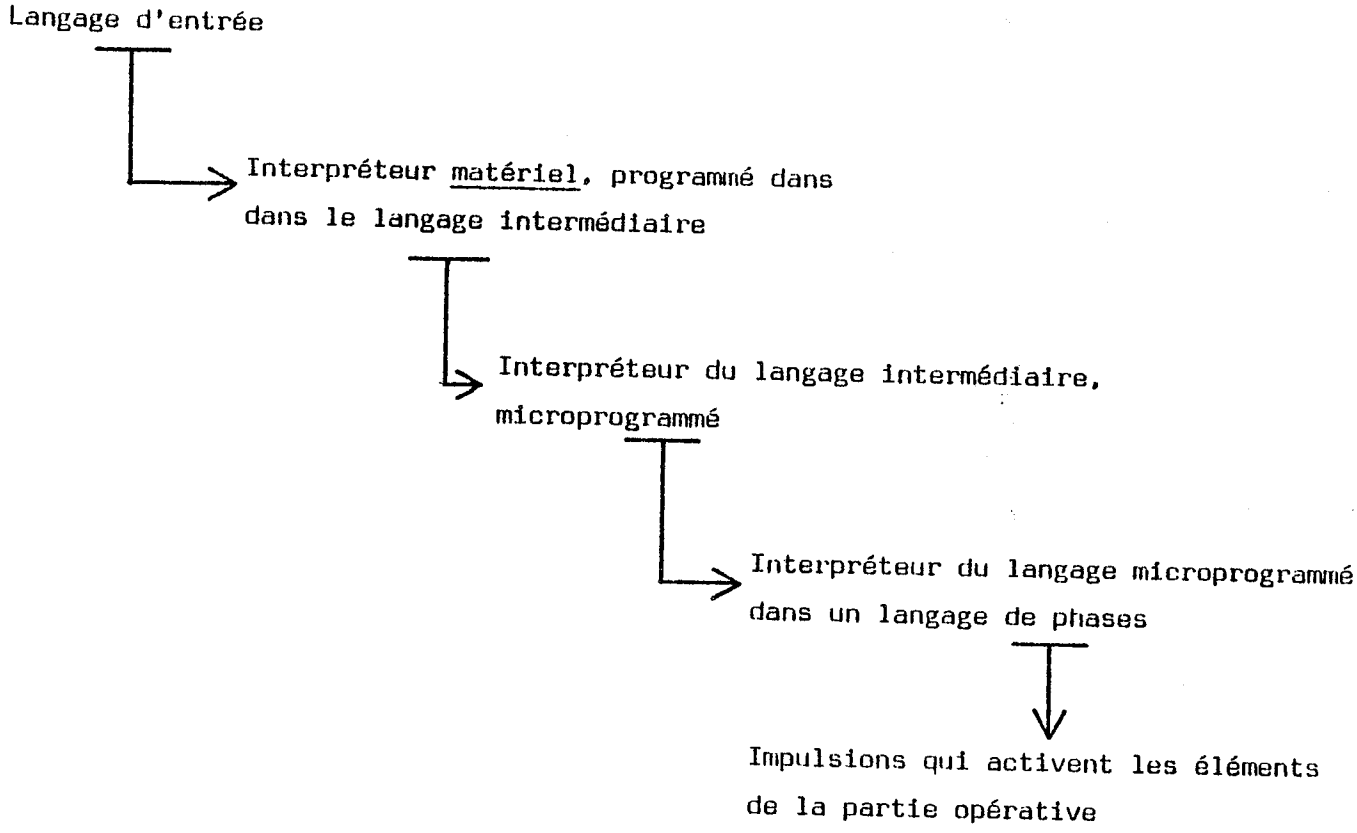
1 - définir un "langage intermédiaire" spécifique avec des instructions de base beaucoup plus puissantes.

Les instructions choisies permettent de faire exécuter plusieurs actions nécessaires dans l'exécution des procédures, par exemple : évaluer une condition ; écrire/lire N octets dans la mémoire ; calculer l'adresse suivante dans un adressage séquenciel, séquenciel bouclé ; etc... .

2 - utiliser un interpréteur matériel : un interpréteur d'un langage est un automate d'états finis qui peut être facilement implémenté en utilisant un PLA avec un mécanisme de mémorisation de l'état [MEA.79].

Les algorithmes des procédures des moniteurs sont donc écrits dans le langage intermédiaire qui à son tour est interprété par un automate matériel qui génère les microinstructions.

Ce schéma d'interprétation peut être décrit par :



Remarque :

Dans ce texte on emploie le mot microprogrammation dans un sens un peu plus large que d'habitude [STO.75, ALE.78]. Premièrement on considère que la mémoire de microprogramme peut être un PLA ce qui permet d'optimiser la codification et l'implantation du microprogramme. Deuxièmement on considère qu'un certain séquencement peut se faire sur les microordres extraits de la microinstruction, évidemment ce séquencement est rudimentaire (2 ou 4 phases) par rapport à des solutions de contrôle où tout le séquencement est fait par des réseaux de portes logiques (Z8000 [SHI.79]).

L'utilisation d'une logique anarchique est la meilleure façon d'optimiser la surface de silicium occupée par le séquenceur. Cependant un contrôle microprogrammé

bien que ne permettant pas une minimisation de la surface, possède l'important avantage de pouvoir être implanté automatiquement et d'une manière fiable.
Un autre avantage important qui découle de l'utilisation des PLA's est la possibilité de pouvoir modifier les algorithmes de contrôle sans changer la topologie globale du circuit, ce qui était fondamental dans le cas du processeur de communication de MOSAIC.

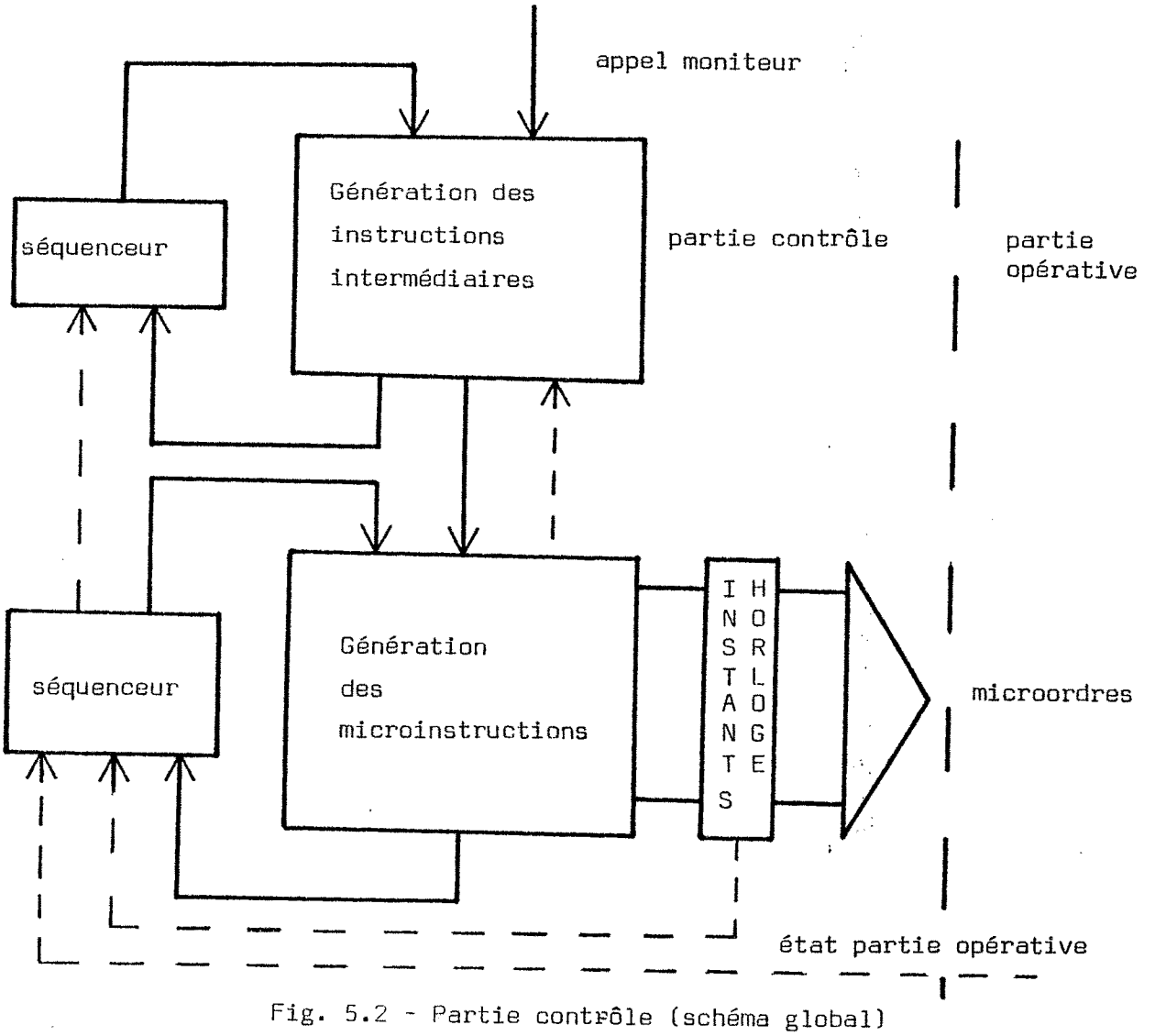


Fig. 5.2 - Partie contrôle (schéma global)

On va maintenant préciser les différents organes de la partie opérative pour pouvoir ensuite décrire plus en détail les éléments de la partie contrôle.

5.2.2. Partie opérative

Chemins de données

Le choix des types de chemins de données à utiliser est une des décisions initiales à prendre dans la définition d'une partie opérative. Une structure de machine à deux bus permet d'avoir un certain parallélisme dans l'exécution des instructions et facilite une implémentation régulière du style "bit-slice".

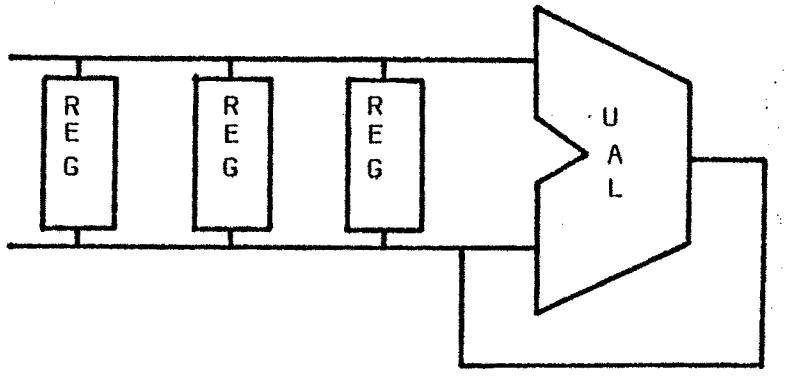


Fig. 5.3. - Machine à deux bus

Un troisième bus pour écrire le résultat permet d'augmenter les performances si l'on peut faire de l'anticipation, c'est-à-dire si l'opération suivante ne dépend pas de la valeur calculée dans l'opération précédente [MAR.78].

Dans l'exécution des procédures, il y a beaucoup d'actions conditionnelles et donc une structure de ce type ne pourrait pas être convenablement exploitée.

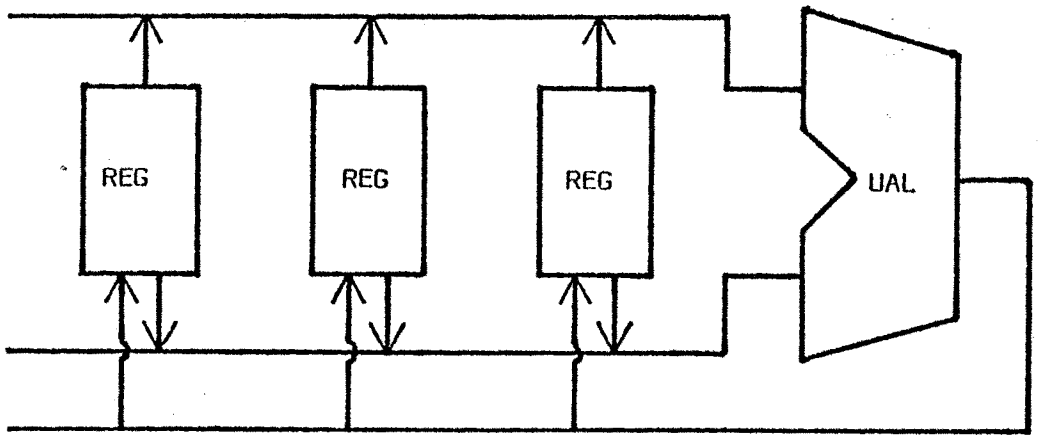


Fig. 5.4. - Machine à trois bus

D'autre part, une architecture avec deux bus nous permet d'utiliser quelques résultats d'une étude qui est actuellement développée dans l'équipe d'Architecture d'ordinateurs de l'IMAG et qui vise la définition d'une bibliothèque d'éléments utilisés dans l'assemblage d'une partie opérative [SUZ.79].

Dans ce travail, on prétend définir un ensemble de briques correspondant aux opérateurs et aux éléments de mémorisation les plus couramment utilisés dans la réalisation de parties opératives . Les connexions entre les briques sont pré-définies de façon à ce qu'elles puissent s'imbriquer directement sans utiliser des programmes de routage.

Dimensionnement des registres et des chemins de données

Dans la conception de la plupart des processeurs, la taille des variables à manipuler est une donnée fixée dès le départ, par contre dans MOSAIC elle est très dépendante des caractéristiques de l'application.
Exemple :

pointeur = $\log_2$ nombre d'octets du moniteur
nombre de flags : dépendent de l'algorithme
base = $\log_2$ taille de la mémoire

Deux solutions nous semblaient possible pour résoudre ce problème :

1 - concevoir la partie opérative comme une suite de tranches dont le nombre serait adaptable à l'application ;
2 - fixer a priori les dimensions des chemins de données et la taille des variables.

La première solution serait très souple au point de vue surface et performances du noyau. Cependant il est pratiquement impossible d'assembler une partie opérative en plaçant côte à côte des tranches d'un bit, car il y a toujours des problèmes électriques qui dépendent de la taille de la partie opérative (propagation retenue, multiplexage mémoire locale, etc...).

Obliger à simuler et à évoluer la partie opérative pour chaque application nous semble peu intéressant. On a donc décidé de fixer dès le départ la taille des bus et des variables internes.

Pour nous guider dans le choix du dimensionnement des chemins de données on a évalué les besoins de certaines applications, les données obtenues peuvent être exemplifiés par :

- la taille des données des moniteurs varie entre quelques dizaines d'octets et 2 Koctets ;
- le nombre de flags entre 0 et 6 ;
- la taille de la mémoire de communication entre 512 octets et 3 Koctets ;
- les paramètres d'une procédure entre 1 octet et 12 octets.

On a ensuite déterminé quelles sont les opérations exécutées le plus souvent et quelle est l'influence des dimensions des chemins de données dans les temps d'exécution d'une procédure.

Partant de plusieurs données de ce type on a essayé diverses solutions. Celle qui nous semble présenter le meilleur compromis surface/vitesse d'exécution est de considérer la partie opérative composées par deux sections :

- une section 8 bits chargée d'exécuter l'évaluation des conditions au début et à la fin de la procédure ;
- une section 12 bits chargée de manipuler les adresses des données dans la mémoire de communication.

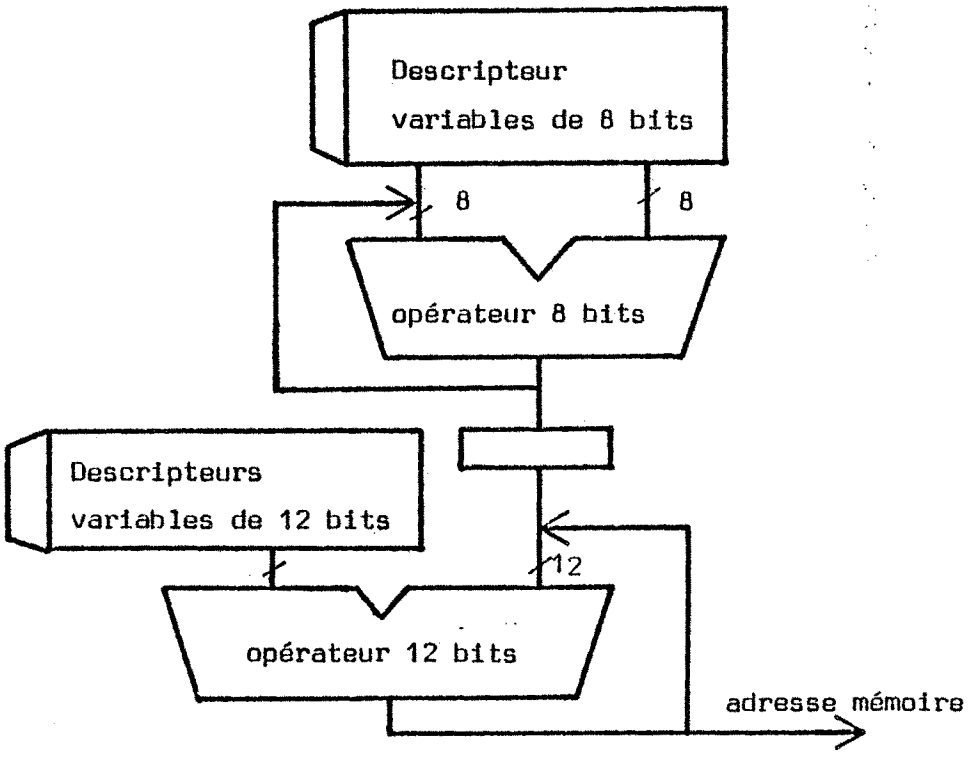


Fig.5.5 - Structure de la partie opérative.

Dimensionnement des variables

Adresse base - On a fixé la taille de la mémoire de communication à 4 Koctets ce qui nous permet de traiter tous les exemples étudiés.

Dans la section 6.1 on verra plus en détail les limitations imposées par l'implantation sur la taille de la mémoire.

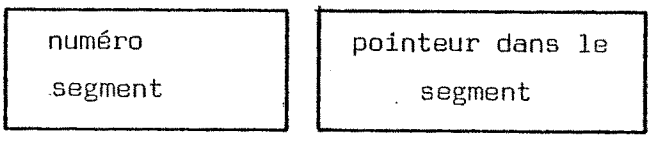
Pointeur - On a considéré deux cas en fonction du nombre d'octets occupés par les données du moniteur :

nombre d'octets < 256 - pointeur sur 1 octet

nombre d'octets > 256 - pointeur sur 2 octets.

Taille - Pour les moniteurs inférieurs à 256 octets, la taille peut être codifiée dans un seul octet tandis que pour les moniteurs supérieurs à 256 octets on aurait besoin de deux octets.

Cependant on a remarqué que les moniteurs du deuxième groupe sont, presque toujours, multiples de 256 octets, ils peuvent donc être segmentés sous la forme de blocs de 256 octets.



Quoique moins souple, ce type d'adressage permet de codifier la taille par un seul octet (nombre de segments du moniteur) et d'éviter des comparaisons inutiles dans les algorithmes d'accés.

Noctet - Nombre d'octets de l'unité d'information - On admet une valeur de 1 à 16 pour ce paramètre.
Des valeurs supérieures à 16 obligent à maintenir le mécanisme de communication bloqué pendant une période très longue ce qui peut être génant pour l'allocation du processeur noyau (4.1).

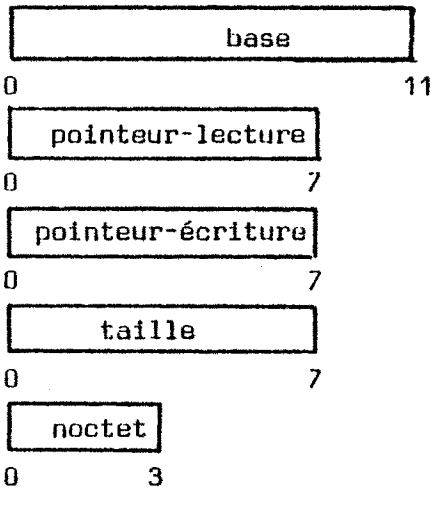
Utilisateur lecture - 1 octet
Utilisateur écriture - 1 octet

Flags - Comme on l'a déjà mentionné, le nombre de flags nécessaires dépend de l'algorithme du moniteur.

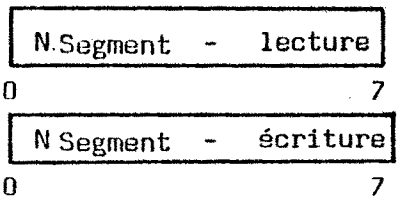
Pour simplifier la réalisation de l'opérateur matériel chargé du test et positionnement des flags, on a été amené à considérer un mot regroupant toutes les flags du descripteur. Un mot de 8 bits s'est avéré suffisant pour tous les exemples programmés.

Conditions - Chaque condition a un compteur (1 octet) pour indiquer le nombre de processus qui sont bloqués.
Exemple de quelques descripteurs :

- Moniteur - sans réservation
  (minimal) sans aucune condition
  inférieur à 256 octets



- Moniteur - supérieur à 256 octets - les éléments du descripteur précédent plus



Opérateurs -

Opérations arithmétiques nécessaires :

Addition:

sur 12 bits (base + pointeur)

sur 8 bits

Soustraction:

sur 8 bits (comparaison)

Incrémentation/décrémentation:

sur 12 bits (adresses)

sur 8 bits (pointeurs, compteurs)

Opérations logigues :

test des flags

positionnement des flags

Opérateur arithmétique sur 8 bits :

Les opérations nécessaires pour exécuter les algorithmes des procédures sont :

addition	$\alpha + \beta$
	$\alpha + \beta +$ retenue
soustraction	$\alpha - \beta$
incrémentation	$\alpha + 1$
décrémentation	$\alpha - 1$
transparence	α

Une UAL classique peut être utilisée.

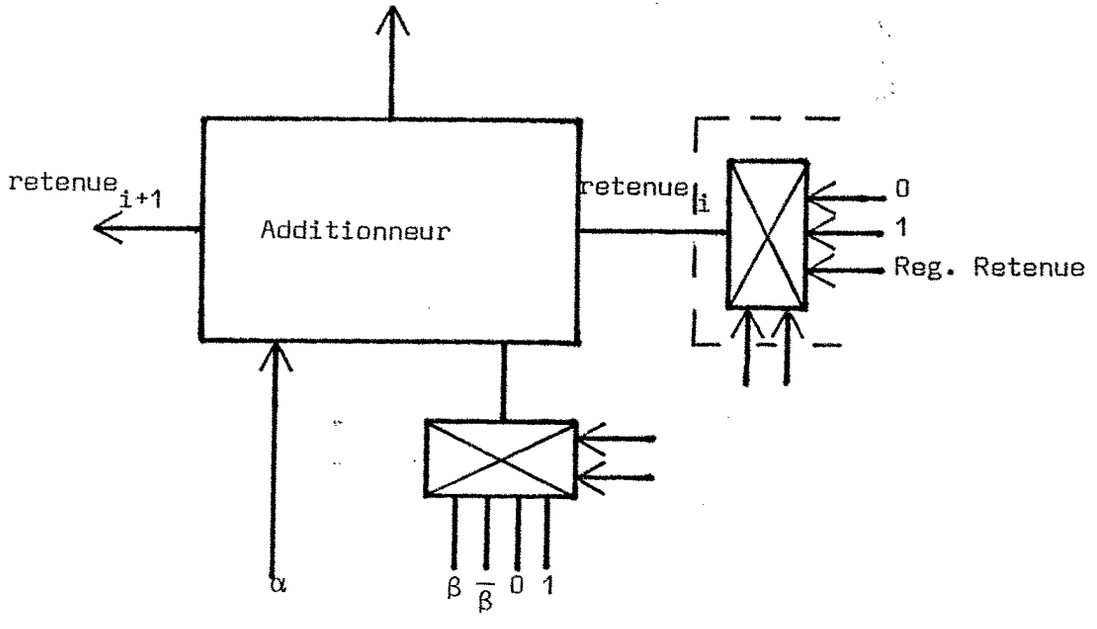


Fig.5.6 - Point UAL

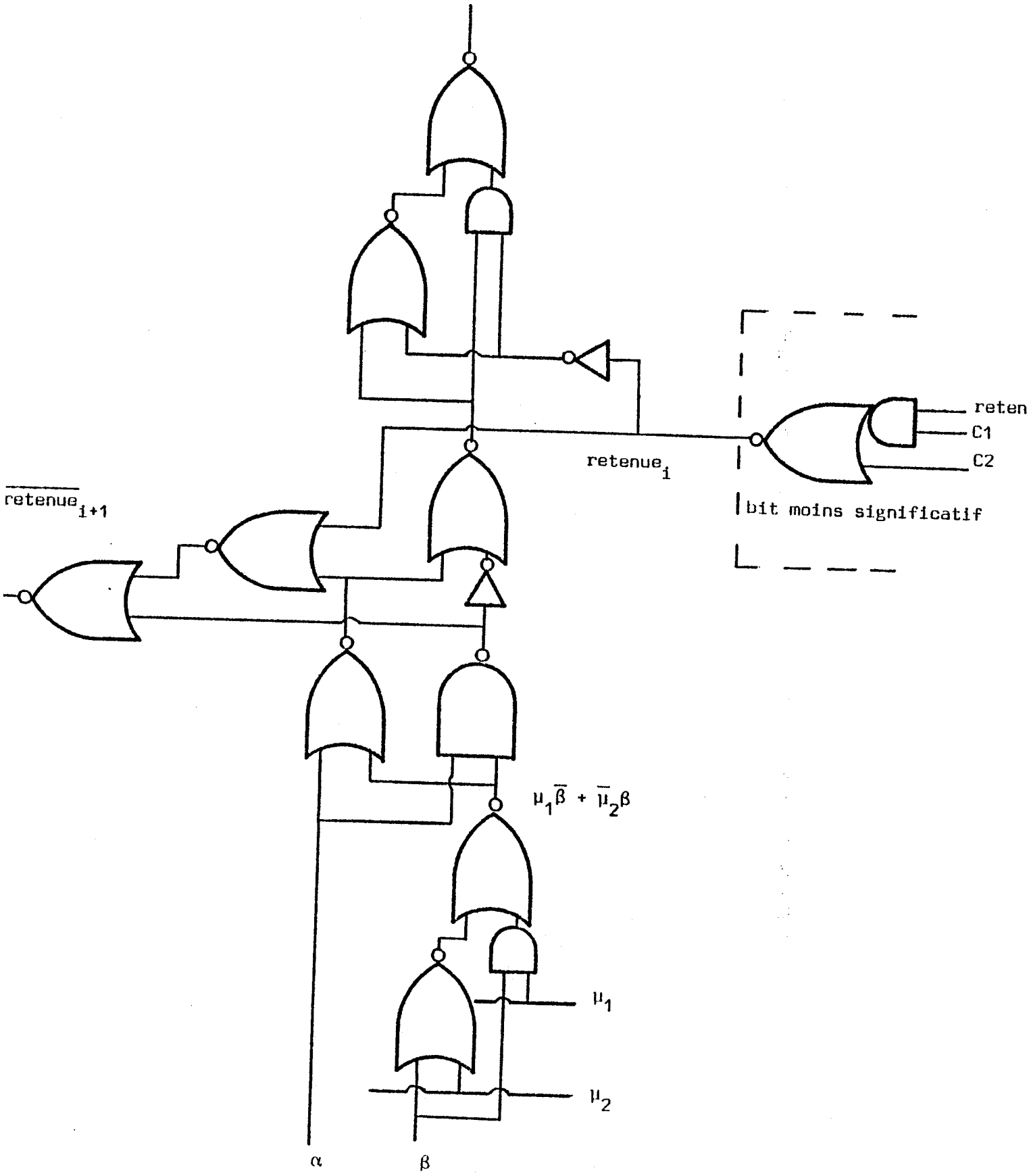


Fig.5.7 - Schéma logique de l'UAL

Tableau des commandes

	μ_1	μ_2	c_1	c_2	résultat
addition	0	0	0	0	$\alpha+\beta+0$
addition+carry	0	0	1	0	$\alpha+\beta+cy$
soustraction	0	1	0	0	$\alpha+\overline{\beta}+1$
transparence	0	1	0	0	$\alpha+0+0$
	1	0	0	1	$\alpha+FF+1$
incrémentation	0	1	0	1	$\alpha+0+1$
décrémentation	1	0	0	0	$\alpha+FF$

Opérateur logique

Le test de positionnement des flags est une opération assez courante dans les algorithmes des moniteurs. Utiliser l'UAL pour effectuer le test (opération ET avec un jeu de masques en ROM) oblige à employer deux cycles pour cette opération.

La fréquence d'utilisation des opérations de test et positionnement des flags justifie la création d'un opérateur spécial pour les exécuter.

Pour des raisons d'implémentation on utilise un registre semi-statique connecté à un multiplexeur qui permet de sélectionner le flag à tester ou à positionner.

Le multiplexeur est placé à coté de la partie opérative. Les lignes de sélection sont aussi utilisées pour le décodeur qui permet de tester et positionner les bits d'état de l'opérateur arithmétique (zéro, carry, etc...)

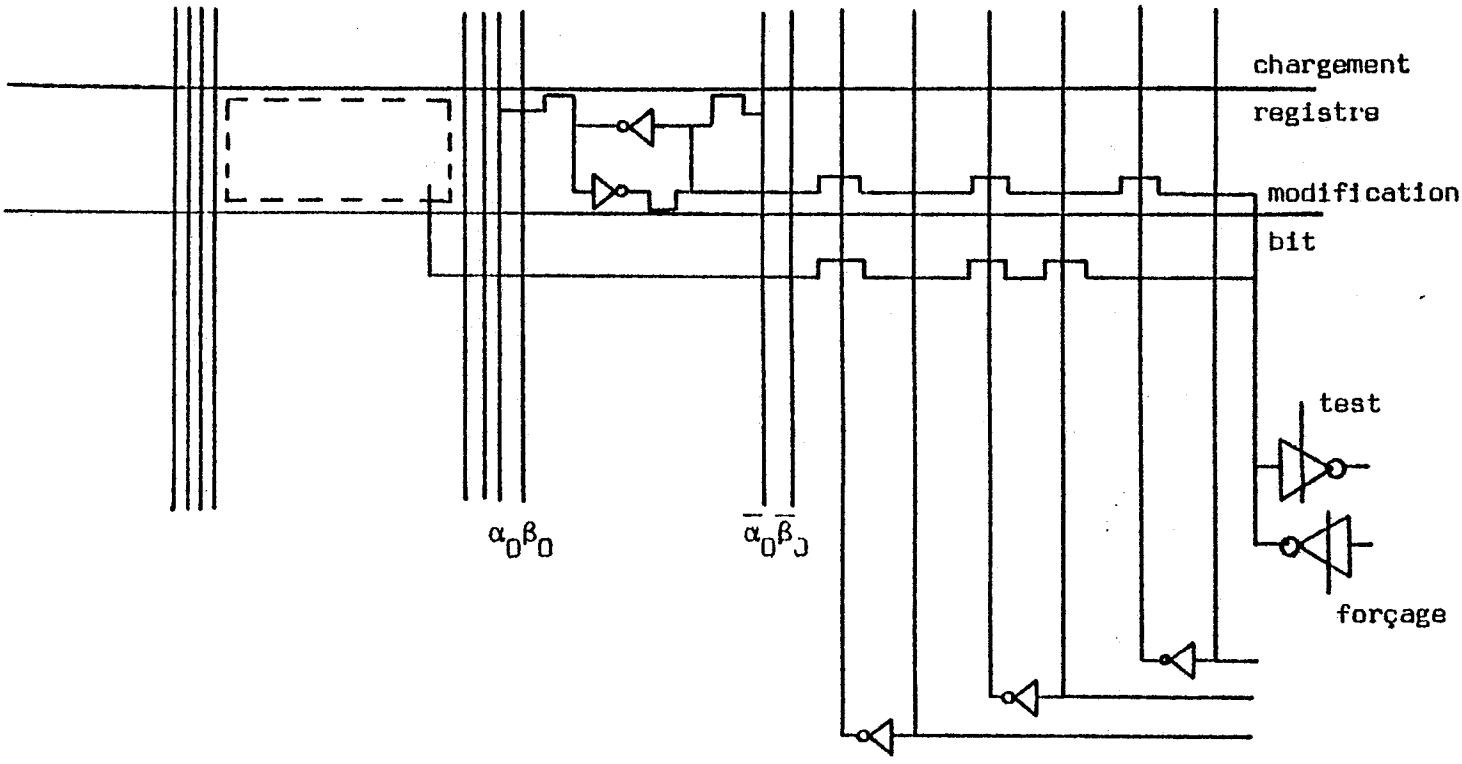


Fig. 5.8 - Opérateur binaire

Registres

Un ensemble de registres à double accés permet de mémoriser le descripteur d'un moniteur pendant son exécution.

Initialement on avait prévu de faire tout le catalogue comme une mémoire à double accés. Cependant, cette solution a du être abandonnée car le catalogue a une taille relativement importante et la surface occupée par un point mémoire à double accés est à peu près le double de celle d'un point à simple accés.

Au début de l'exécution d'une procédure le descripteur du moniteur correspondant est transféré de la mémoire catalogue à simple accés vers un ensemble de 16 registres à double accés où il est mémorisé pendant l'exécution de la procédure. Ce deuxième niveau de mémorisation a été appelé zone de travail.

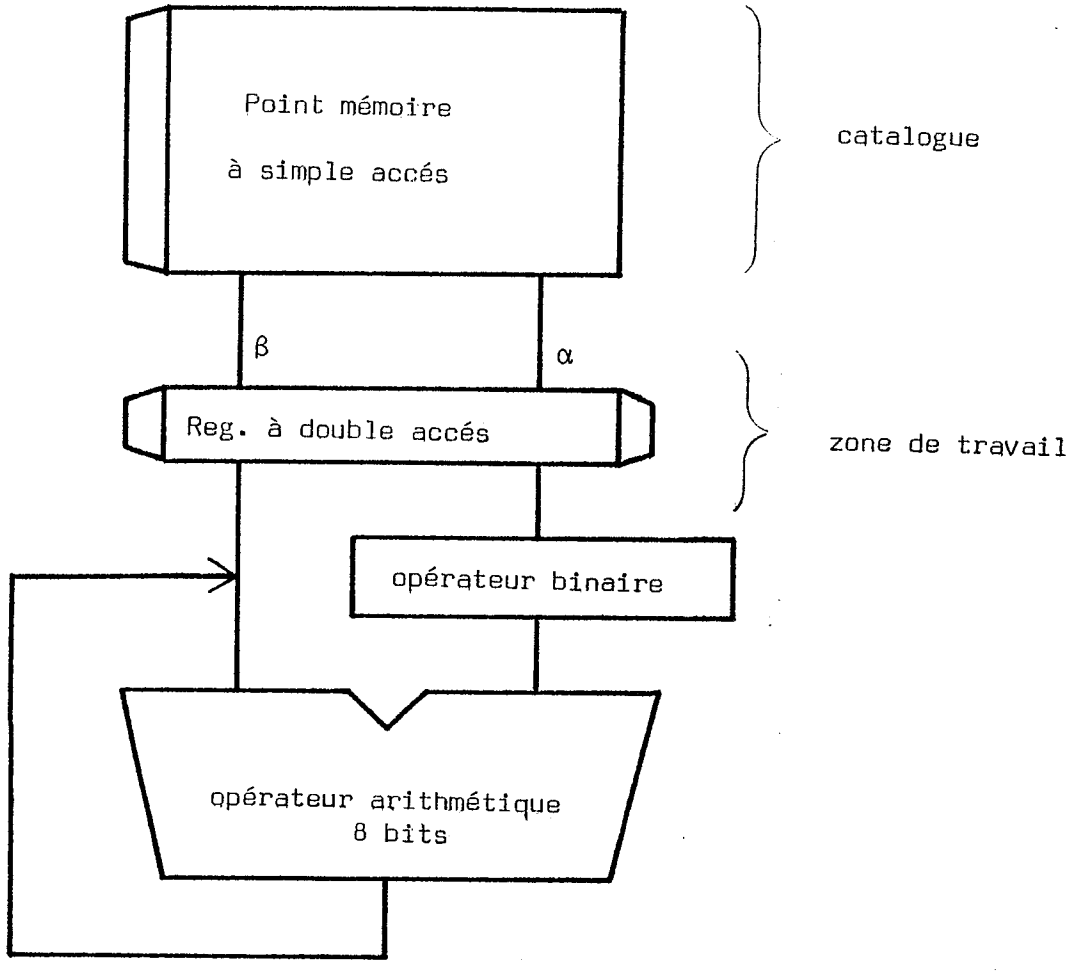


Fig. 5.9 - Section 8 bits

Implantation

Pour pouvoir implanter la partie opérative comme une suite de tranches placées côte à côte il faut que les dimensions des différentes briques soient compatibles.

Dans 6.1 on verra plus en détail les valeurs des surfaces des briques utilisées, pour le moment il faut savoir que la largeur (axe x sur la figure 5.10) d'un point à double accés est le double de celle du point à simple accés et que la largeur du point UAL est le double de celle du point à double accés.

La structure globale de l'implantation est donc la suivante :

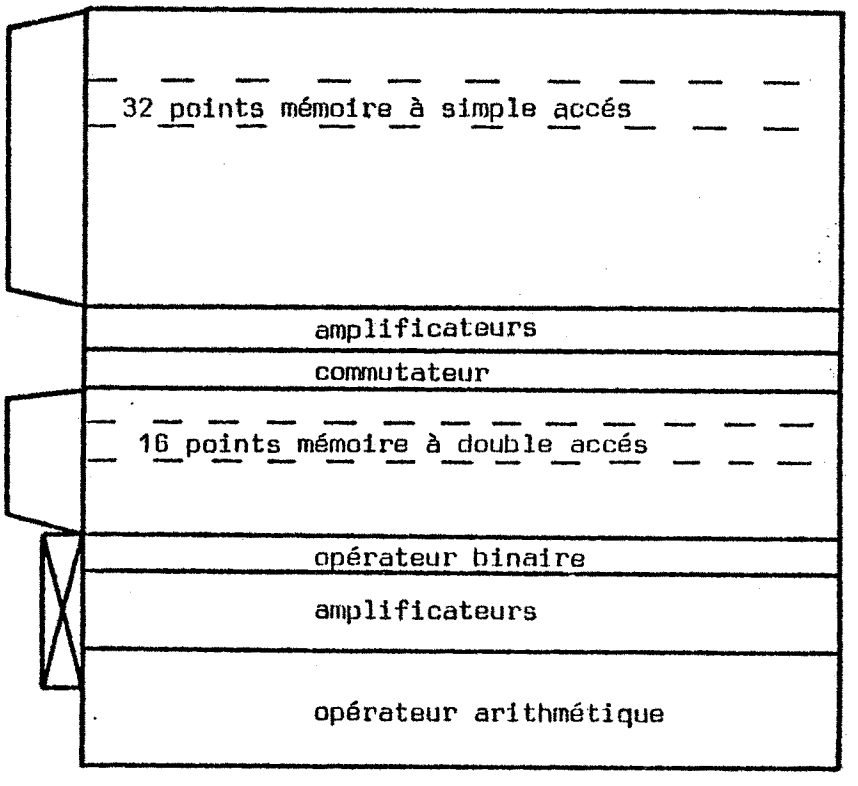


Fig. 5.10 - Structure de l'implantation

Opérateur arithmétique sur 12 bits

Justification de l'utilisation d'un opérateur 12 bits

Les procédures qui écrivent ou lisent des données dans la mémoire de communication sont les plus fréquemment utilisés et donc celles dont le temps d'exécution influence le plus la performance du noyau.

Si l'on n'avait pas introduit l'opérateur sur 12 bits l'accés aux données se traduirait par la boucle ci-après :

```
        Tampon := Noctet
ADR1 :  Reg. Adresse_Pf := Base_Pf + Pointeur ;
        Reg. Adresse_PF := Base_PF + (Nsegment) + carry ;
        Pointeur := Pointeur + 1 ;
        Tampon := Tampon - 1 ;
        Si Tampon ≠ 0 alors aller a ADR1
```

Si cette boucle était implémentée directement, le temps d'exécution serait fortement pénalisé car chaque transfert demanderait 5 cycles du processeur noyau.

Pour minimiser le temps d'adressage de la mémoire de communication on a donc utilisé un opérateur arithmétique 12 bits couplé à un registre d'adresse. On a aussi extrait du catalogue l'adresse base qui est mémorisée dans une ROM couplée à l'opérateur 12 bits.

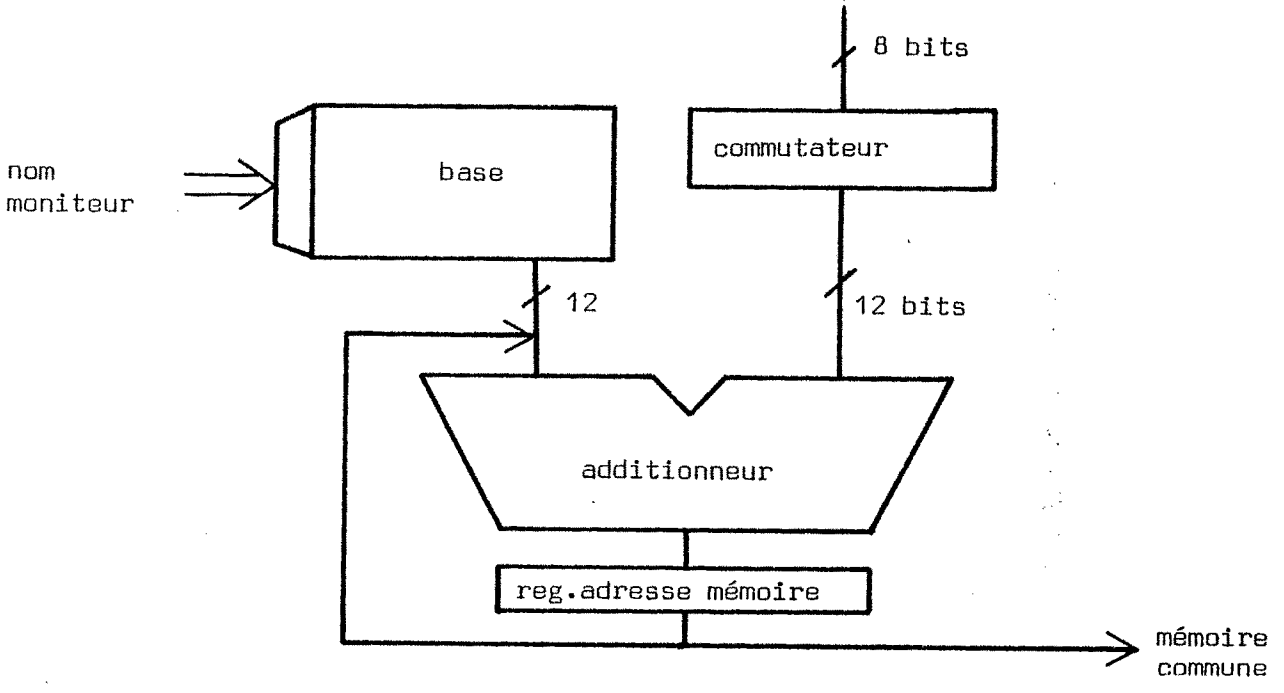


Fig. 5.11 - Section 12 bits

En profitant du paralléisme entre les deux opérateurs, la boucle précédente devient :

	Section 8 bits	Section 12 bits
	Lecture Pointeur	Lecture Ad. Base
	TAMPON := Noctet	Reg.Adresse := Base + Pointeur
ADR1 :	TAMPON := TAMPON-1	Reg.Adresse := R.Adresse + 1
	Si TAMPON=0 alors aller ADR1	Lecture mémoire de communication
	Pointeur := Pointeur+Noctet	

On peut encore optimiser cette solution en effectuant la décrémentation de Noctet dans la section 12 bits, ce qui libère la section 8 bits pour l'exécution de la réévaluation des conditions à la fin de la procédure.

La valeur initiale de $\overline{\text{Noctet+1}}$ est maintenue dans la ROM catalogue. Un registre avec un incrémenteur permettent de faire progresser la valeur de Noctet, un test à zéro est fait à chaque cycle de façon à déterminer la fin de l'accés mémoire.
Si pour un moniteur il y a plusieurs unités d'information, il faut prévoir une ligne, du catalogue ROM, pour chacune. Il y aura donc une certaine perte d'efficacité dans le remplissage de la ROM car l'adresse base sera toujours la même. Cependant ceci n'est pas très génant car l'utilisation de plusieurs unités d'information est assez rare et une ligne ROM occupe une surface très réduite (voir 6.1).

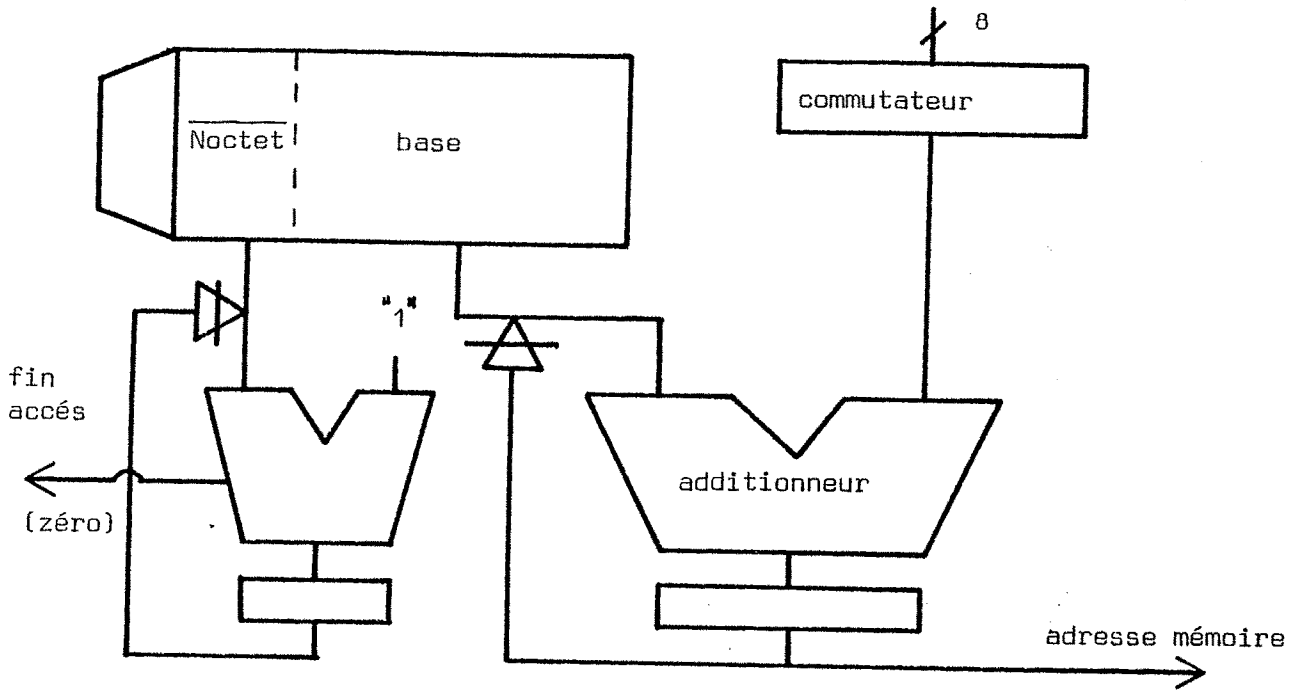


Fig. 5.12 - Décrémentation Noctet

Cette structure logique qui peut sembler un peu compliquée, s'implante très facilement car l'opérateur qui incrémente Noctet est le même que celui qui incrémente les adresses. La seule différence par rapport à un opérateur 16 bits est qu'il n'y a pas de propagation de la retenue entre le 12ème et le 13ème bits.

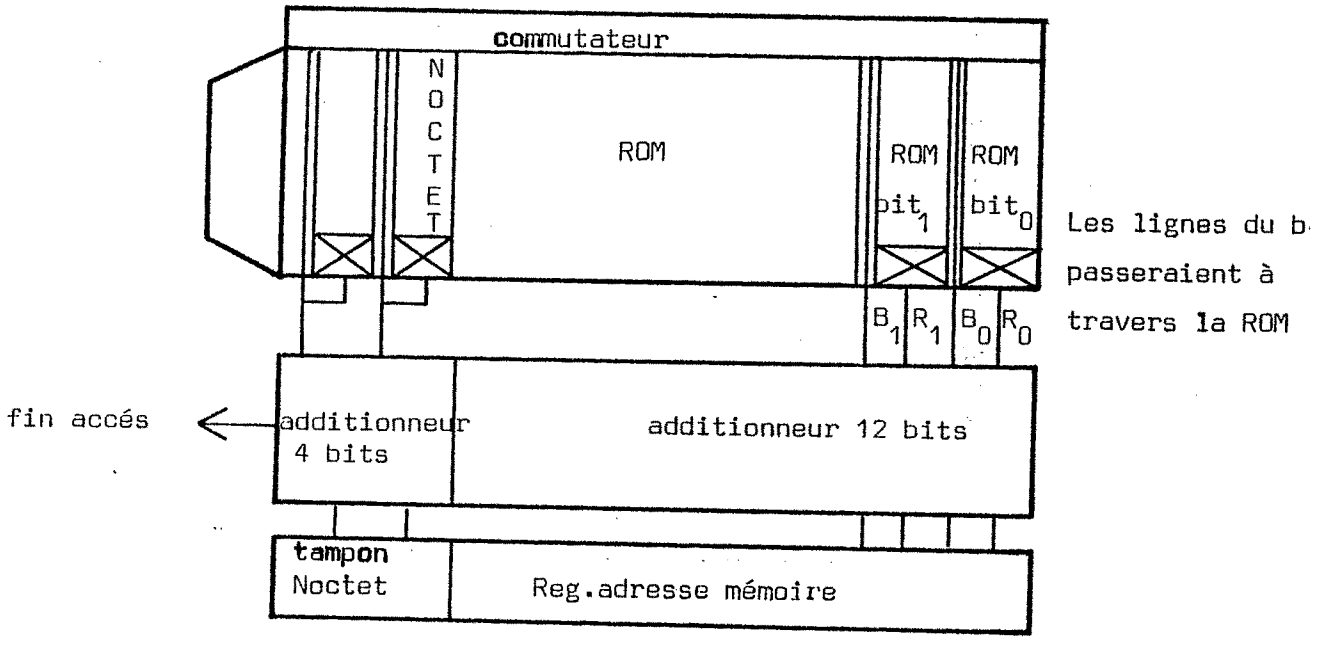


Fig.5.13 - Schéma d'implantation

Commutateur entre les deux sections opératives

L'organe qui connecte les deux sections opératives de la machine doit exécuter trois opérations :

- transfert d'un octet de la partie 8 bits → la partie 12 Bits poids faible
- transfert d'un octet de la partie 8 bits → la partie 12 bits poids fort
- transfert de Noctet de la partie 12 bits → la partie 8 bits

Si on utilise la notation suivante :

b - bus de la partie 8 bits

B - bus de la partie 12 bits

les transferts peuvent être décrits par:

1/ $B_0 \leftarrow b_0 ; \ldots ; B_7 \leftarrow b_7$

2/ $B_8 \leftarrow b_0 ; \ldots ; B_{11} \leftarrow b_3$

3/ $b_0 \leftarrow B_{12} ; \ldots ; b_3 \leftarrow B_{15}$

Le commutateur est implémenté par un multiplexeur bidirectionnel commandé par trois lignes.

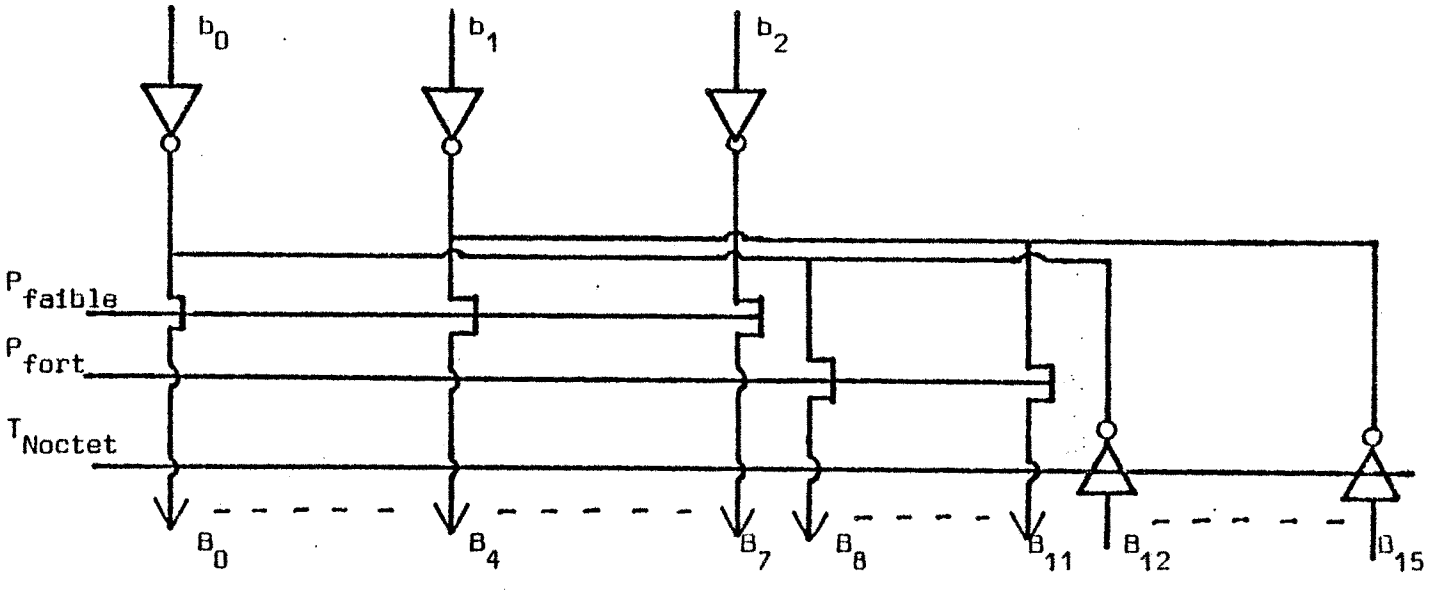


Fig. 5.14 - Commutateur entre les partie 8 et 12 bits

Séquencement de la partie opérative

Un aspect très important dans la définition du séquencement de la partie opérative est l'utilisation d'un bus avec ou sans précharge.

L'utilisation de la précharge des bus est liée au problème de la lecture destructive des points de mémoire.

Pour des raisons d'économie de surface, la géométrie des transistors des points mémoire est minimal. L'impédance des MOS de charge des inverseurs est donc élevée de façon à permettre d'utiliser des MOS de commande minimaux. Le bus qui a une charge capacitive considérable peut donc modifier l'information mémorisée dans le point si aucune précaution n'est prise.

Trois solutions sont possibles :

1 - sélectionner lentement le point mémoire de façon à réduire progressivement la résistance du transistor interrupteur, ce qui entraîne une augmentation significative des temps d'accés ;
2 - utiliser des amplificateurs pour attaquer les bus, ce qui augmente d'une forme importante la taille des points de mémorisation ;
3 - précharger les bus Q et $\overline{Q}$ à un niveau de tension intermédiaire.

La dernière solution permet d'utiliser des points de taille minimale et évite les aléas de séquencement de la première solution. Par contre, elle impose l'utilisation d'un temps mort dans l'utilisation des bus.

Cependant, si on isole, pendant la précharge, l'UAL des bus, tout le cycle peut être utilisé pour exécuter une opération car l'UAL peut finir son calcul pendant le temps de précharge suivant.

Dans la méthode de conception des parties opératives [SUZ.78] il est prévu d'utiliser des bus à précharge et de faire traverser l'UAL par les bus Q et $\overline{Q}$ provenant des éléments de mémorisation.

Des amplificateurs différenciels permettent d'accélérer la commutation des bus.

Les amplificateurs sont déclenchés par une différence de tension d'environ 1V entre leurs entrées. Les sorties sont à haute impédance tant que ce seuil n'est pas franchi (une commande permet de les désactiver pendant la précharge).

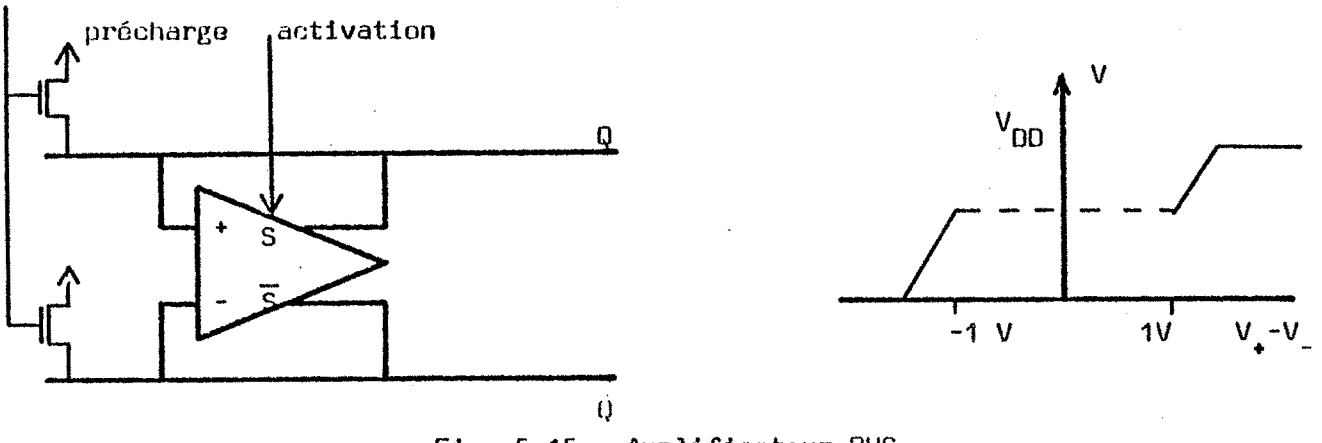


Fig. 5.15 - Amplificateur BUS

Le séquencement des opérations dans la partie opérative est donc le suivant :

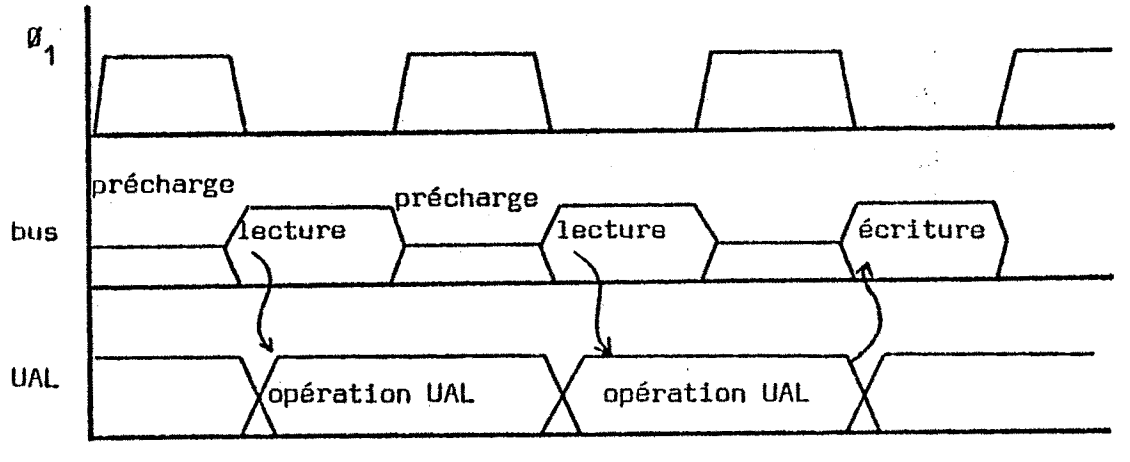


Fig. 5.16 - Chronographe de la partie opérative

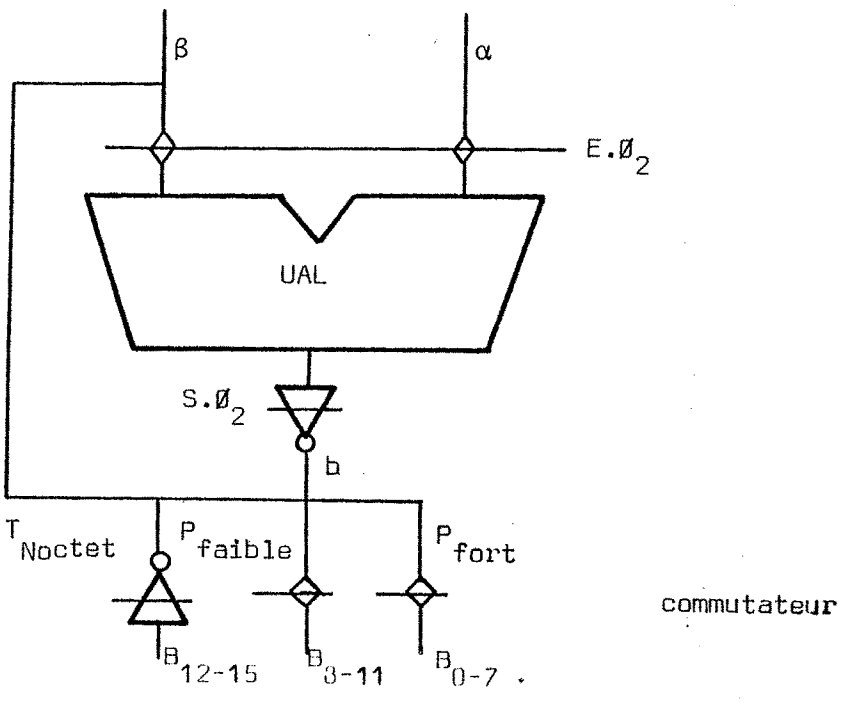


Fig. 5.17 - Opérateur 8 bits et commutateur

Communication entre le catalogue et la zone de travail

Le commutateur entre les deux types d'éléments de mémorisation a la forme suivante :

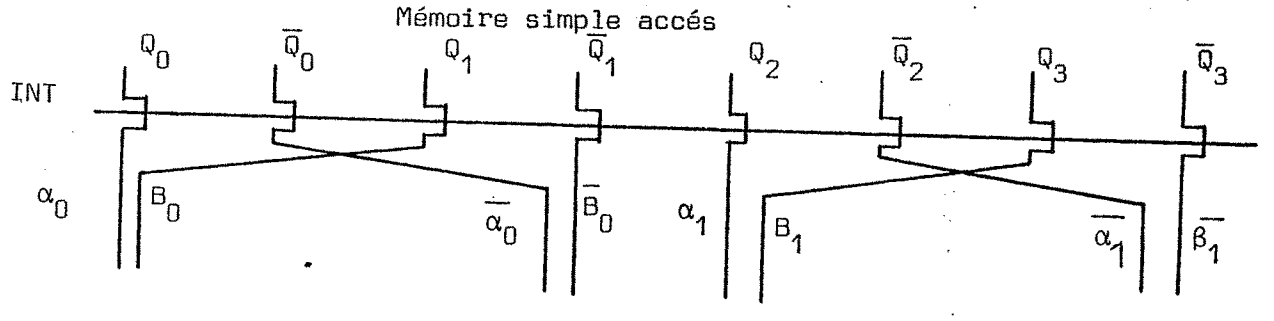


Fig. 5.18 - Commutateur entre les deux zones mémoire

La liaison entre les deux bus permet de charger deux registres de 16 bits à double accés, à partir d'un mot de 32 bits de la mémoire catalogue.

Pour effectuer ce transfert dans un seul cycle, il faut que les mémoires aient un timing différent et que les précharges des bus soient commandées.

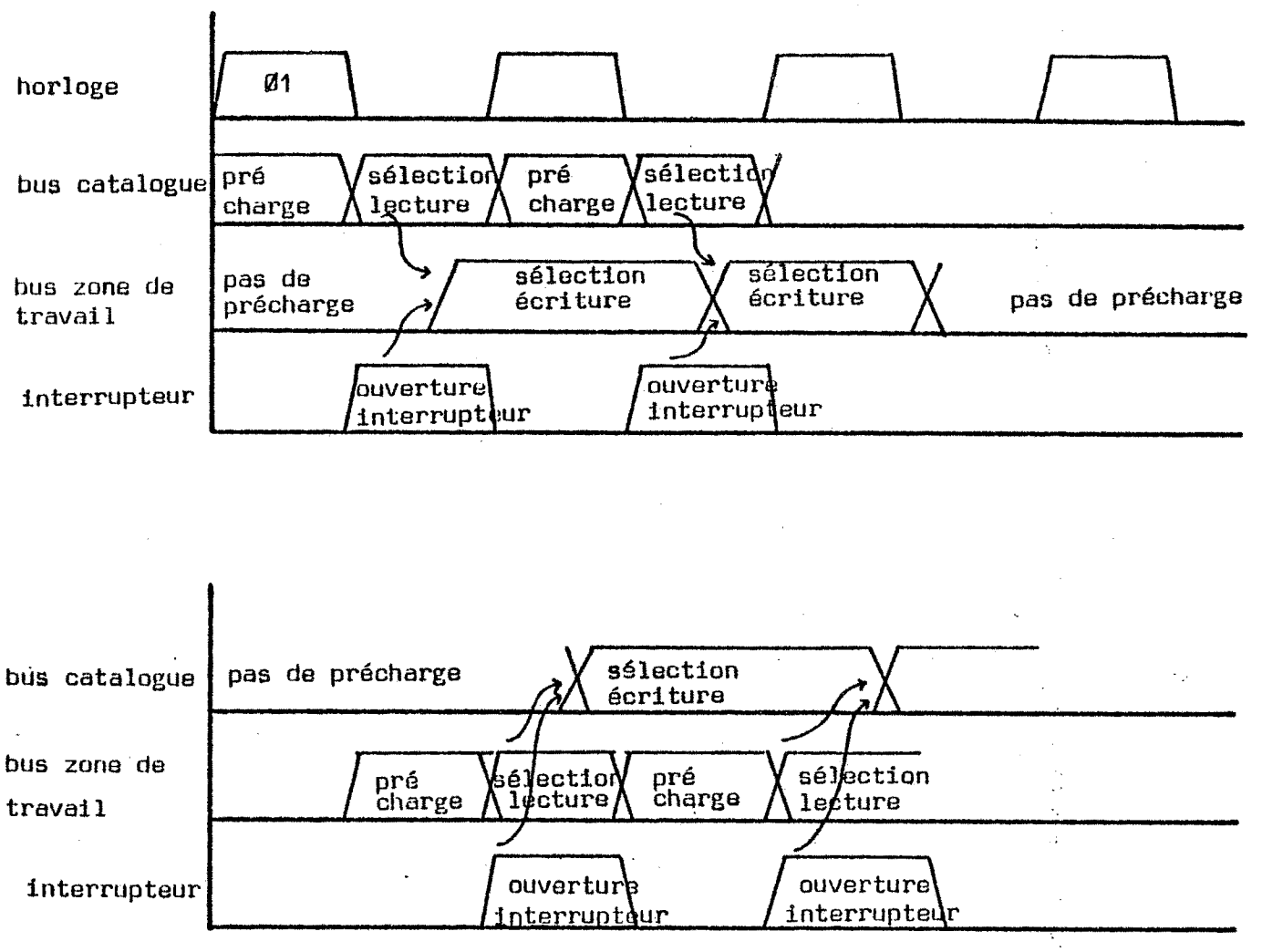
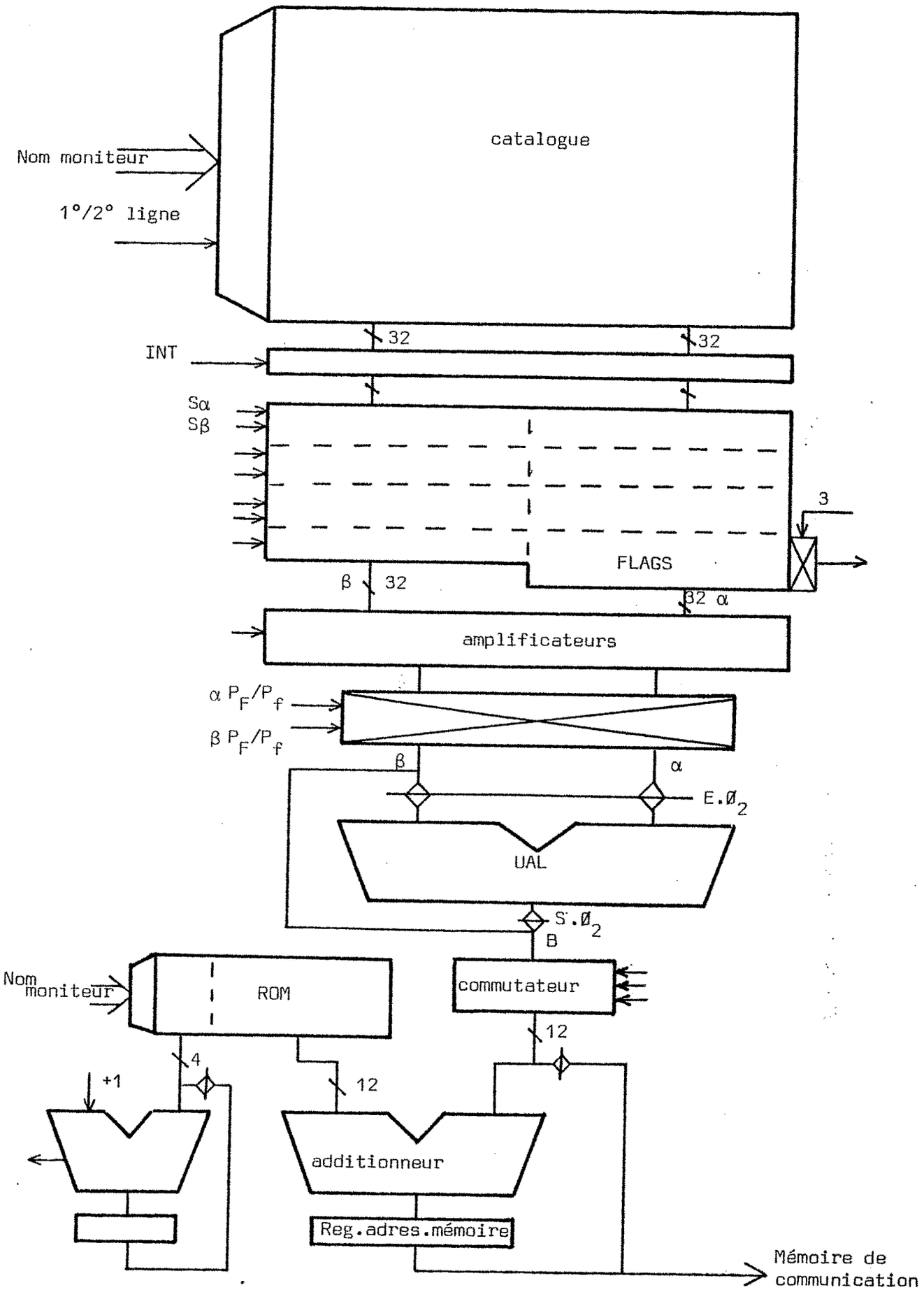


Fig. 5.19 - Chronographe des transferts entre les deux zones mémoire



5.20 - Partie opérative

5.2.3. Partie Contrôle

La partie contrôle est constituée par deux interpréteurs matériels qui traduisent le langage d'entrée du processeur en une séquence d'impulsions électriques permettant d'activer les chemins de données et les opérateurs de la partie opérative. Dans cette section on commence par décrire plus en détail chacun des langages qui interviennent dans le processus d'interprétation pour pouvoir ensuite spécifier les organes utilisés dans leur interprétation.

- Langage d'entrée

Les instructions du langage d'entrée sont un ensemble de paires ordonnées:

(nom-moniteur, procédure)

où nom-moniteur appartient à l'ensemble des identificateurs du langage de spécification du type "external monitor" et procédure appartient à l'ensemble des identificateurs qui ont été déclarés comme "procedure entry" à l'intérieur du moniteur désigné par nom-moniteur.

- Langage intermédiaire

Le langage intermédiaire se compose d'instructions qui correspondent à l'exécution de plusieurs actions élémentaires. Ces actions pouvant être parfois exécutées en parallèle en profitant de l'existence des deux opérateurs de la partie opérative. Le langage intermédiaire n'a pas été complétement défini, on s'est limité à établir la liste des instructions nécessaires (annexe 2) pour implémenter les moniteurs décrits dans l'annexe 1.

Pour l'exécution de nouvelles opérations, l'utilisateur peut en définir d'autres instructions intermédiaires.

On a subdivisé les instructions en quatre grands groupes :

- Evaluation des conditions d'entrée
  ex.: CE-Fifo-écriture, CE-Fifo-lecture
- Accés aux données
  ex.: séquenciel-bouclé-lecture, séquenciel décroissant-écriture

- Evaluation des conditions de sortie
  ex.: CS-Fifo-écriture, CS-Fifo-lecture
- Instructions spéciales
  ex.: attente, continuer, initialisation.

La puissance des instructions intermédiaires et la relative simplicité des moniteurs utilisés dans MOSAIC font que les structures de séquencement nécessaires dans le langage intermédiaire sont très simples.

Deux possibilités sont prévues :

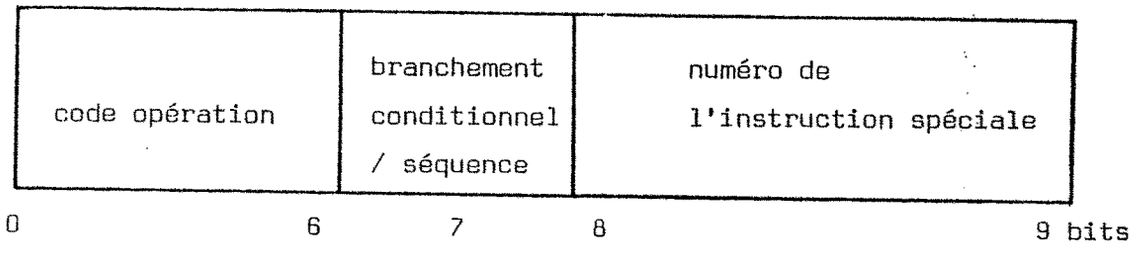
- aller en séquence,
- branchement conditionnel à une instruction spéciale.

Dans l'instruction il y a deux champs, l'un indiquant s'il s'agit ou non d'une instruction de test, l'autre indiquant le numéro de l'instruction spéciale à exécuter s'il y a un branchement.

L'exécution d'une instruction spéciale marque la fin de la procédure et donc l'interpréteur revient au début de son cycle.

Le test est toujours fonction d'une ligne appelée "test" qui est positionnée par le deuxième niveau d'interprétation.

Le format des instructions intermédiaires est le suivante :



Exemple :

procédure Fifo écrire

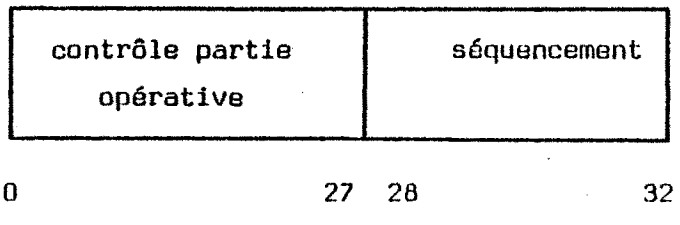
"chargement de la zone de travail"	Initialisation	seq	∅
	CE-Fifo-écriture	branch	attente
	Accés-séquenciel-bouclé	seq	∅
	CS-Fifo-écriture	branch	continuer
"sauvegarde de la zone de travail"	Sauvegarde		

Il est évident que ce langage ne sera pas manipulé par le concepteur et qu'il doit être généré automatiquement par le compilateur du langage de spécification.

Le concepteur peut par contre augmenter les instructions du langage intermédiaire en donnant leur interprétation dans le langage de microinstructions. Le compilateur doit donc avoir la possibilité d'être enrichi par de nouvelles instructions.

- Langage de microprogrammation

Les microinstructions sont du type horizontal et les champs peuvent se diviser en deux groupes.



- Contrôle de la partie opérative

Les champs nécessaires sont :

opérateur arithmétique 8 bits	code opération UAL	4 bits
	sortie UAL	1 bit
adresse charg.descripteur	adresse reg α	4 bits
	adresse reg β	4 bits
opérateur booléen	sélection flag	3 bits
	bit d'état/reg flag	1 bit
	opération test/positionnement	1 bit
catalogue	interrupteur catalogue	1 bit
	lecture/écriture catalogue	1 bit
section 12 bits	contrôle commutateur	3 bits
	addition/incrémentation	1 bit
	sortie reg.adresse mémoire	1 bit
	entrée reg.adresse mémoire	1 bit
mémoire de communication	échantillonnage adresse	1 bit
	lire/écrire	1 bit
		28 bits

- Séquencement

La définition du séquencement du microprogramme est très importante car il peut obliger soit à coder un grand nombre d'informations dans la microinstruction, soit à utiliser des organes matériels supplémentaires (compteur, pile, multiplexeur).

On a essayé de minimiser les informations codées dans la microinstruction et en même temps d'utiliser très peu de matériel pour implémenter le séquencement.

Les microinstructions ne contiennent pas l'adresse de la microinstruction suivante et on offre 8 possibilités fixes de séquencement à l'intérieur du microprogramme.

Séquencement direct :

0 - passage à la microinstruction suivante

1 - appel de l'instruction intermédiaire suivante,

2 - appel de l'instruction intermédiaire de sauvegarde (fin de la procédure).

Séquencement conditionnel :

Le test à effectuer sur les flags ou sur les bits d'état de l'opérateur arithmétique sont définis par les deux champs "sélection flag" et "bit d'état/flag" mentionnés dans le contrôle de la partie opérative (on notera par ligne-test le résultat du test).

3 - si ligne-test alors incrémentation
sinon passage à l'instruction intermédiaire suivante

4 - si ligne-test alors instruction intermédiaire spéciale
sinon instruction intermédiaire suivante

5 - si ligne-test alors sauter la microinstruction suivante
sinon passage à la microinstruction suivante

6 - si ligne-test alors boucler
sinon instruction intermédiaire suivante

7 - si ligne-test alors instruction intermédiaire suivante
sinon microinstruction suivante

Remarque : Le passage à l'instruction intermédiaire suivante signifie qu'il n'y a pas de déroutement au niveau du programme du premier niveau d'interprétation (test = vrai).

Passage à l'instruction intermédiaire spéciale signifie qu'il y a un déroutement dans le premier niveau (test = faux).

- Langage de phases

Les actions qu'il est possible d'exécuter dans un seul cycle d'horloge sont imposées par le parallélisme existant entre les différents organes matériels.

Le nombre de phases nécessaires à l'exécution d'une microinstruction est donc fixé par des choix effectués au niveau de la partie opérative (précharge bus, structure à deux bus, organe séparé pour le test des flags, deux sections pouvant travailler en parallèle, etc...).

Du point de vue de l'interpréteur du langage de microinstruction on a deux types de microinstructions :

a/ exécuté sur deux cycles (4 phases) - exemple :

		bus	opérateur 8 bits	opérateur binaire
P_1	ϕ_1	précharge	—	
				positionnement d'un flag
	ϕ_2	lecture opérandes	début opération	—
P_2	ϕ_1	précharge	fin opération	—
	ϕ_2	lecture opérandes	—	test sur les bits d'état

b/ exécuté sur un cycle (deux phases) - exemple :

P_1 ϕ_1 précharge - test ou positionnement flag

ϕ_2 transfert catalogue → registre ou registre → catalogue
section 12 bits → section 8 bits
ou
section 8 bits → section 12 bits

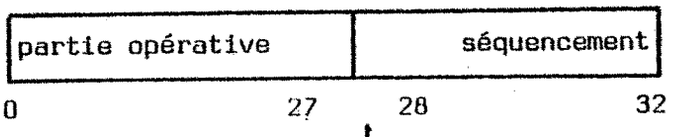
L'imbrication des langages correspondant aux quatre niveaux d'interprétation peut être schématisé par :

langage d'entrée - Fifo - lire

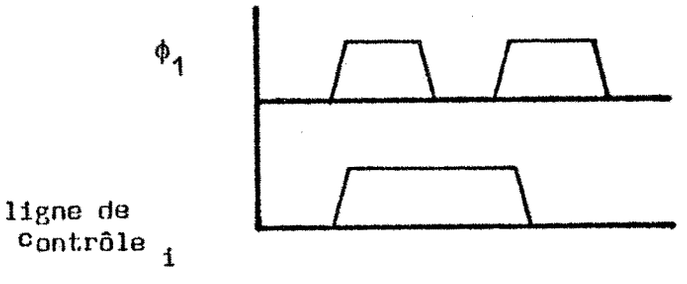
langage intermédaire -

Accés-séquenciel-bouclé | seq | ∅

langage de microinstruction



langage de phases



On va maintenant décrire les mécanismes matériels nécessaires pour établir cette chaîne d'interprétation.

Interprétation du langage d'entrée/langage intermédiaire

Les taches exécutées par cet interpréteur sont :

- contrôler le bus commun de façon à exécuter le protocole de gestion du bus.
- fournir au deuxième niveau d'interprétation les instructions intermédiaires nécessaires pour exécuter une procédure.

- contrôle du bus

Les lignes de contrôle à positionner ou à tester sont celles décrites dans le chapître 4 :

en entrée - Occupé

en sortie - Prêt, Attente, Continuer.

- génération des instructions intermédiaires

Quand le deuxième niveau d'interprétation le sollicite, le premier niveau doit fournir les instructions intermédiaires correspondant à l'interprétation de l'appel de la procédure.

Le séquencement prévu dans le langage intermédiaire est très simple :

1 - aller en séquence

2 - branchement conditionnel d'une instruction spéciale (Attente, Continuer, Exception, Initialisation).

Pour exécuter le séquencement au niveau matériel, on utilise un compteur ordinal et dans l'instruction on codifie s'il s'agit d'une instruction susceptible de provoquer un déroutement et, dans le cas affirmatif, quelle instruction spéciale exécuter.

Les informations qui permettent de calculer l'état suivant de l'automate du premier niveau d'interprétation sont :

- l'appel de la procédure (nom moniteur, procédure),
- l'état d'avancement de la procédure,(compteur)
- la ligne test/séquence,
- le numéro de l'instruction intermédiaire spéciale
- la ligne test.

Entrées du PLA

Appel	nom moniteur	5
	procédure	3
Bus - occupé		1
Compteur ordinal - adresse		4
Instruction intermédiaire	type de segment	1
	numéro d'instruction intermédiaire spéciale	2
Test		1
		17 lignes

Sorties du PLA

Instruction intermédiaire	10
Incrémenter/reset compteur	1
Prêt	1
Attente	1
Continuer	1
Charger registre appel moniteur	1
	15 lignes

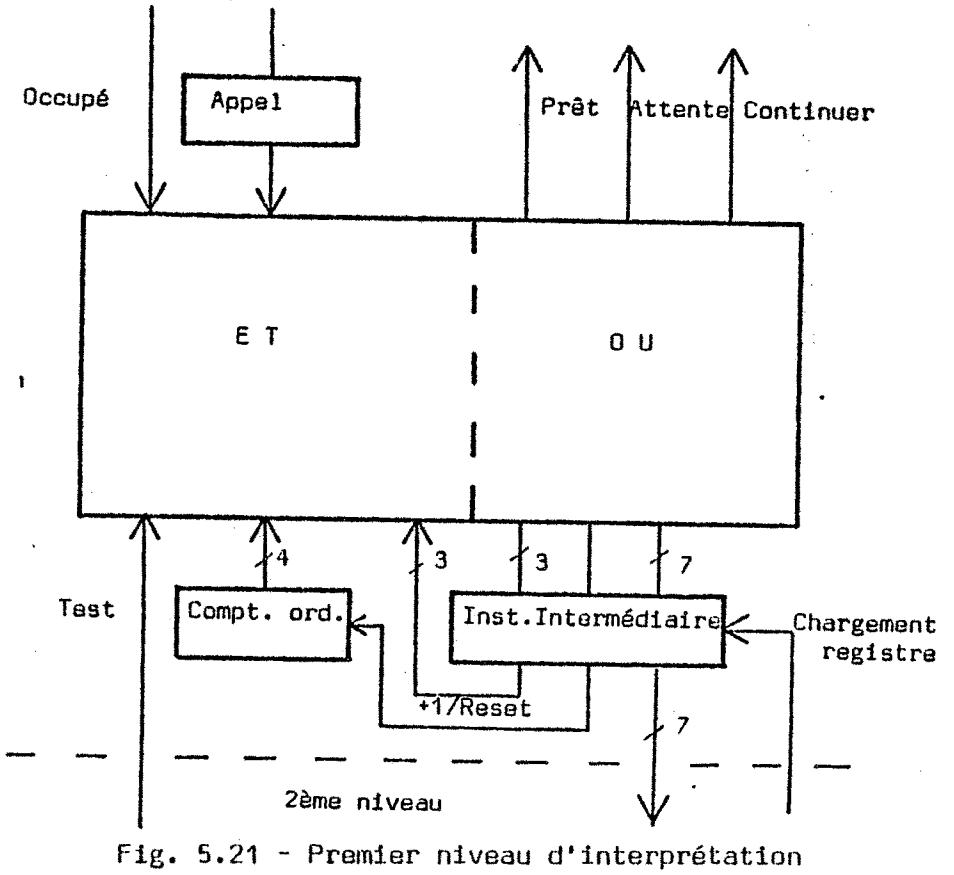


Fig. 5.21 - Premier niveau d'interprétation

Interpréteur langage intermédiaire / langage de microinstruction

Le deuxième niveau génère les microordres et exécute le séquencement à l'intérieur d'une suite de microinstructions qui interprète l'instruction intermédiaire.

Etant donné le nombre de microinstructions et la complexité du séquenceur, une solution utilisant deux PLA's, un chargé de la génération des microordres et un autre d'effectuer le séquencement, permet d'optimiser la surface.

Séquencement

En plus du PLA on utilise un compteur polynominal à 4 étages qui fonctionne comme un microcompteur ordinal et permet donc d'adresser un maximum de 15 microinstructions pour chaque instruction intermédiaire.

Les entrées du PLAseq (séquencement) sont :

Information de séquenement de la microinstruction	3
Ligne de test en provenance de la partie opérative	1
	4 lignes

sorties

Incrémenter microcompteur ordinal	1
Inhibition des sorties du PLA de génération des commandes	1
Reset microcompteur	1
Chargement du registre d'instruction intermédiaire	1
Test (vers le premier niveau)	1
	5 lignes

Les cinq possibilités de séquencement incluses dans les huit modes proposés précédemment sont :

séquence	incrémenter	reset	inhibition PLA-3	chargement reg.instruc	test
	1	0	0	0	0
appel de l'instruction intermédiaire suivante	0	1	0	1	0
appel d'uns instruction int.spéciale	0	1	0	1	1*
saut d'une micro-instruction	1	0	1	0	0*
boucle sur une microinstruction	0	0	0	0	0

Les deux possibilités de séquencement signalées par une étoile posent des problèmes de "timing".

- Le saut d'une microinstruction doit se dérouler dans deux cycles. Dans le premier la microinstruction courante est inhibée et le compteur polynomial incrémenté, dans le deuxième cycle on exécute la microinstruction.

- L'appel d'une instruction intermédiaire spéciale oblige le PLA du niveau 1 à se positionner en fonction de la ligne test . Il faut donc prévoir un cycle supplémentaire avant de charger le registre d'instruction intermédiaire.

La solution utilisée pour ces deux cas est de reboucler les lignes condition et inhibition ce qui permet au PLA d'exécuter les séquences décrites.

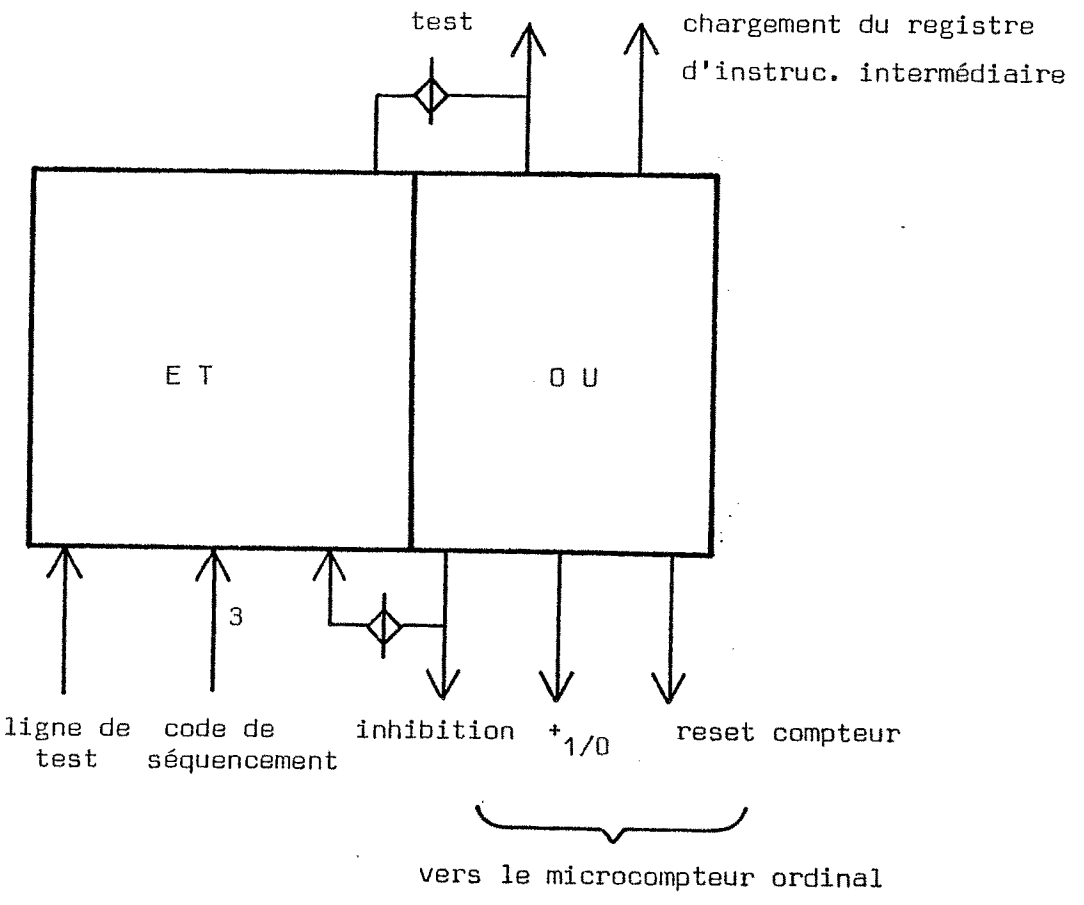


Fig. 5.22 - Séquencement deuxième niveau

PLA de génération des microordres

Le PLA de génération des microordres (PLAgc) est le plus important de la machine (en surface).

Pour l'implanter on pense utiliser une méthode de simplification basée sur l'utilisation d'une matrice "ou" en ligne brisée.

Cette méthode n'est applicable qu'à des PLA's avec une deuxième matrice relativement creusée. Dans les PLA's de génération de commandes pour des

parties opératives, chaque commande utilise un nombre restreint de monômes ce qui permet de regrouper dans la matrice "ET" les monômes intervenant dans une sortie et d'utiliser une ligne "OU" (deuxième matrice) beaucoup plus courte.

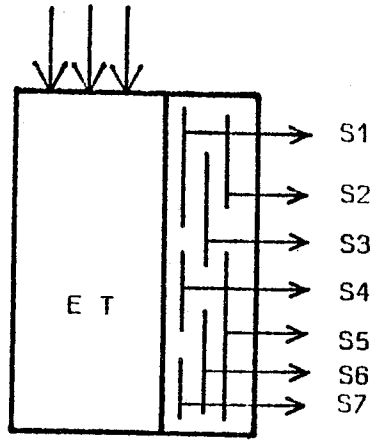


Fig. 5.23 - PLA à ligne brisée

Cette technique peut obliger à repeter quelques monômes dans la matrice "ET" quand ils sont utilisés par plusieurs sorties disjointes (ex.: S1 et S7 dans la figure 5.23). Un tel programme de minimisation est actuellement étudié [PER.80]. Les résultats déjà obtenus semblent indiquer que pour des PLA's d'une dimension semblable à celui que l'on étudie, on peut obtenir un facteur de réduction de la taille de la deuxième matrice de 2 ou 3 par rapport à la matrice "OU" classique.

Entrées du PLAgc (génération des commandes)

Instruction intermédiaire	6
Compteur polynomial	4
Inhibition	1
	11 lignes

Sorties

Microinstruction

. vers la partie opérative	26
. vers la partie séquencement	4
	30 lignes

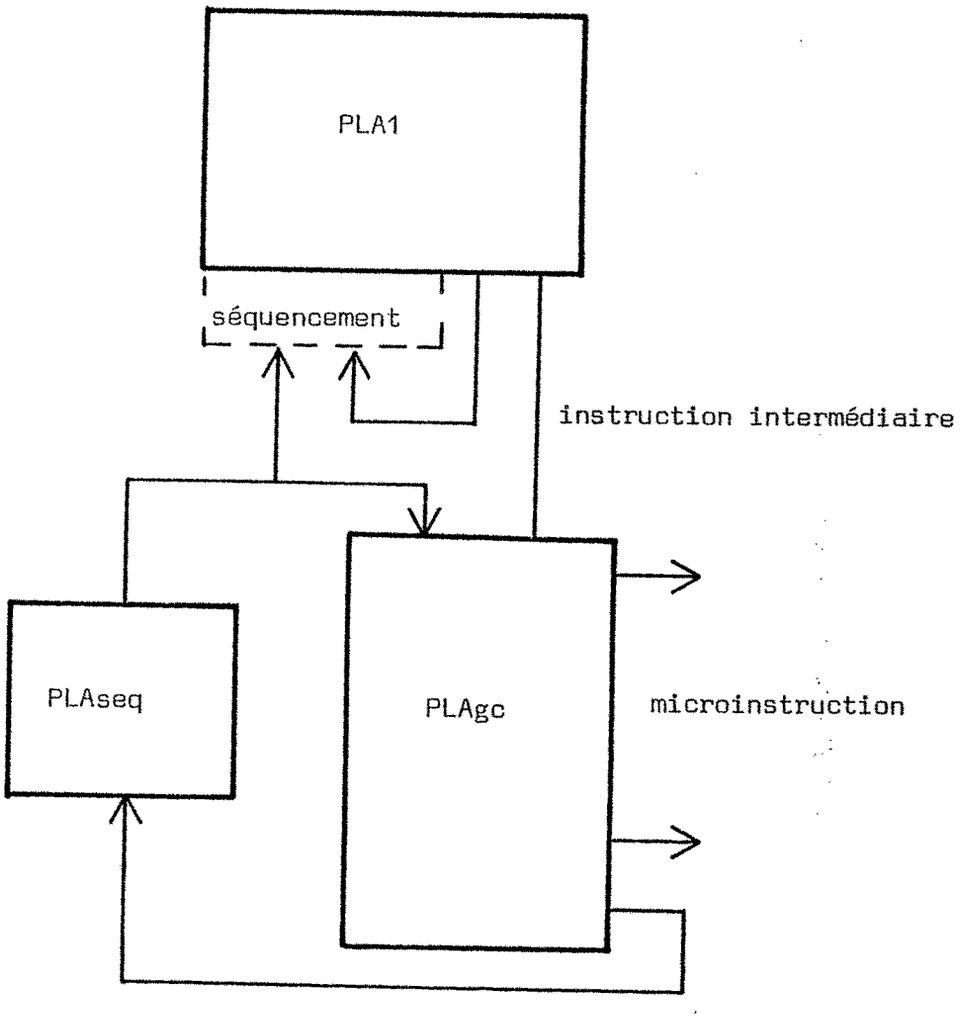


Fig. 5.24 - Schéma simplifié du contrôle

Problème de la reprise de l'exécution d'une procédure

On a laissé pour la fin le cas du démarrage d'un processus bloqué parce qu'il sort du schéma habituel d'interprétation qui vient d'être décrit.

Quand à la fin de l'exécution d'une procédure une condition, avec des processus en attente, est devenue vraie, il faut exécuter les actions suivantes pour faire redémarrer un processus en attente :

1 - alerter les processus bloqués sur la condition ; (4.4)
2 - charger le nom de la procédure qui reprend son exécution
3 - charger l'adresse de l'instruction intermédiaire où l'exécution se poursuit.

Remarque : Il ne faut pas charger le nom du moniteur ni le descripteur correspondant car un "signal" (2.2.2) ne peut se faire qu'entre les procédures du même moniteur.

Pour une question de simplification, on a considéré que dans une procédure il y a un maximum de quatre conditions ayant chacune un compteur dans le descripteur.

L'adresse de ce compteur identifie biunivoquement la condition qui est en train d'être évaluée et qui peut déclencher la reprise d'une autre procédure.

On a donc pensé d'envoyer cette adresse vers le PLA1 de façon à ce qu'elle puisse constituer un pointeur vers une table où seraient codés sur 7 bits le nom de la procédure à reprendre et l'adresse de l'instruction intermédiaire qui doit être exécutée ensuite.

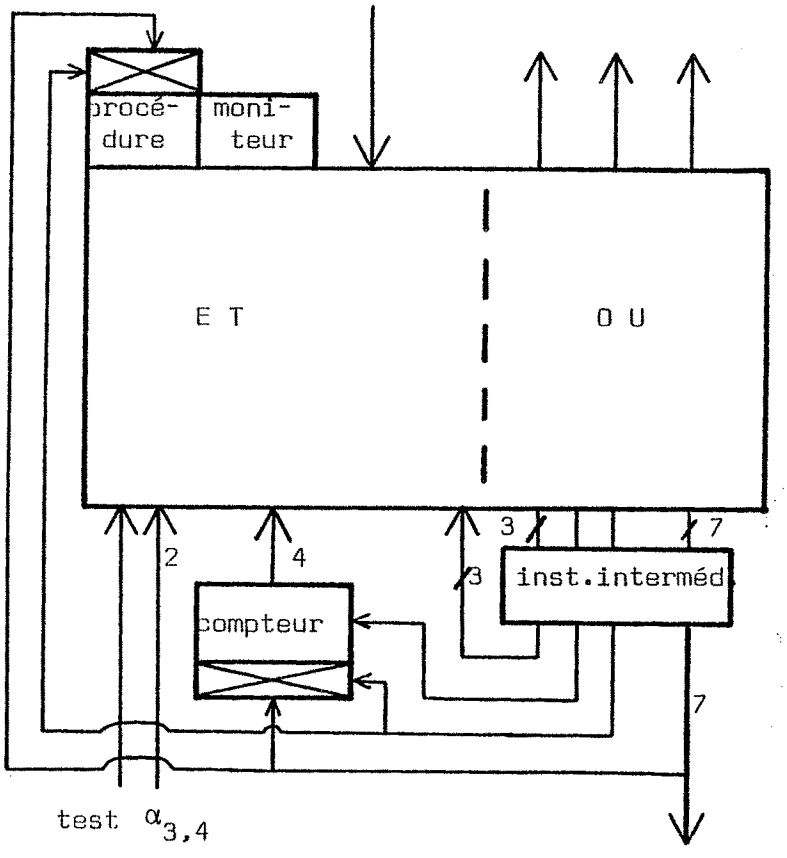


Fig. 5.25 - Modification du premier niveau d'interprétation pour permettre la reprise d'une procédure.

Le mot (procédure, condition) multiplexé sur les sorties de l'instruction intermédiaire servirait aussi d'identificateur de la condition pour les opérations Attente et Continuer qui sont exécutées sur le bus.

Interpréteur du langage de microinstruction / langage de phases

Les microinstructions contiennent un champ d'un bit "cycle" qui indique si l'exécution s'effectue sur 1 ou 2 cycles.

Les commandes sont échantillonnées à la fin de $\emptyset_1$ et peuvent être actives pendant :

1 - ϕ_2

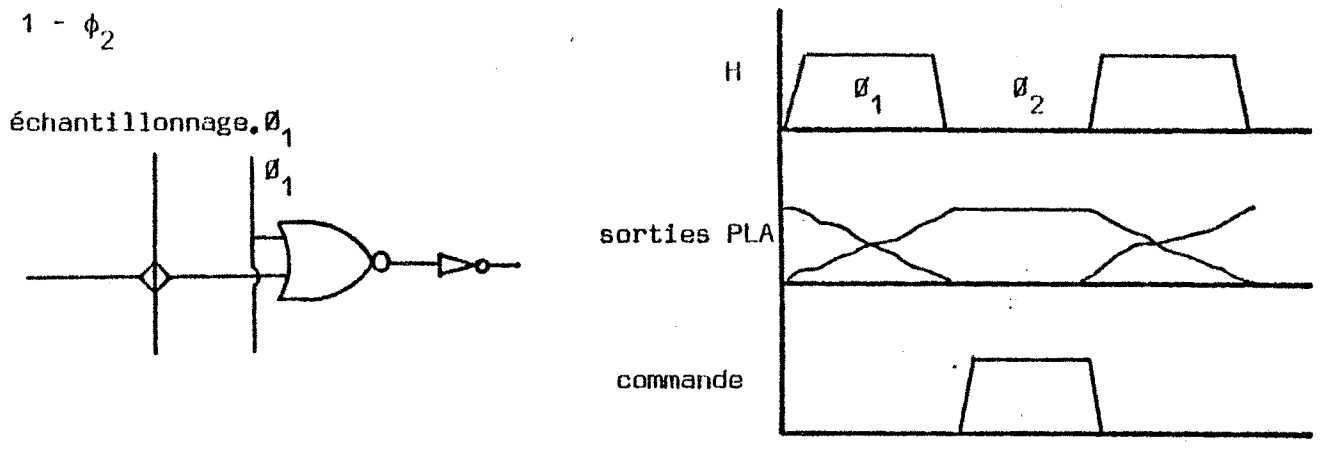


Fig. 5.26

Exemple : Commandes de lecture des éléments de mémorisation, test et positionnement des flags, commutateurs.

2 - Pendant ϕ_2 et ϕ_1

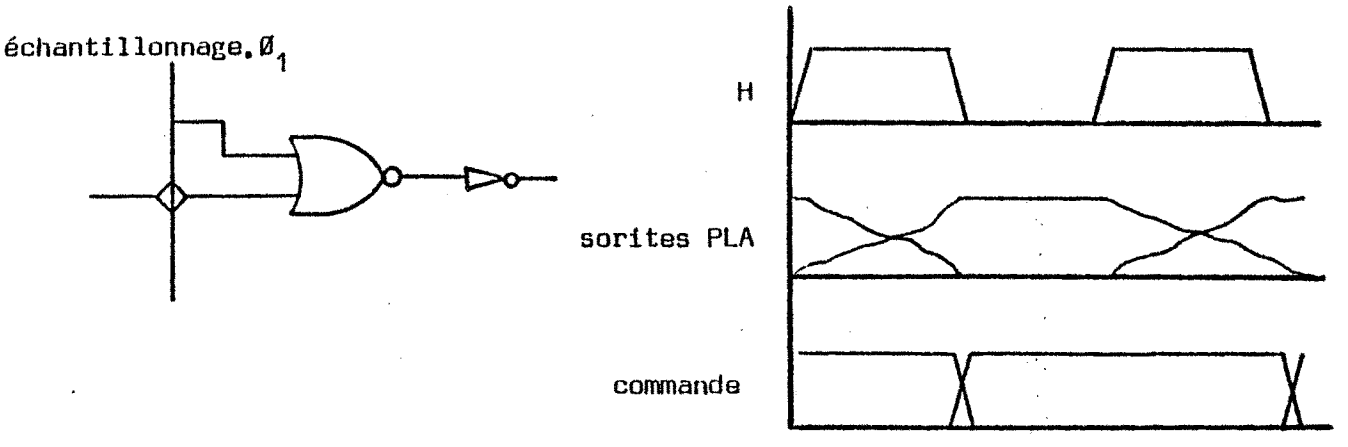


Fig. 5.27

Exemple : Commandes de l'UAL, de l'additionneur 12 bits.

3 - Actives pendant $\emptyset_2$ du deuxième cycle.

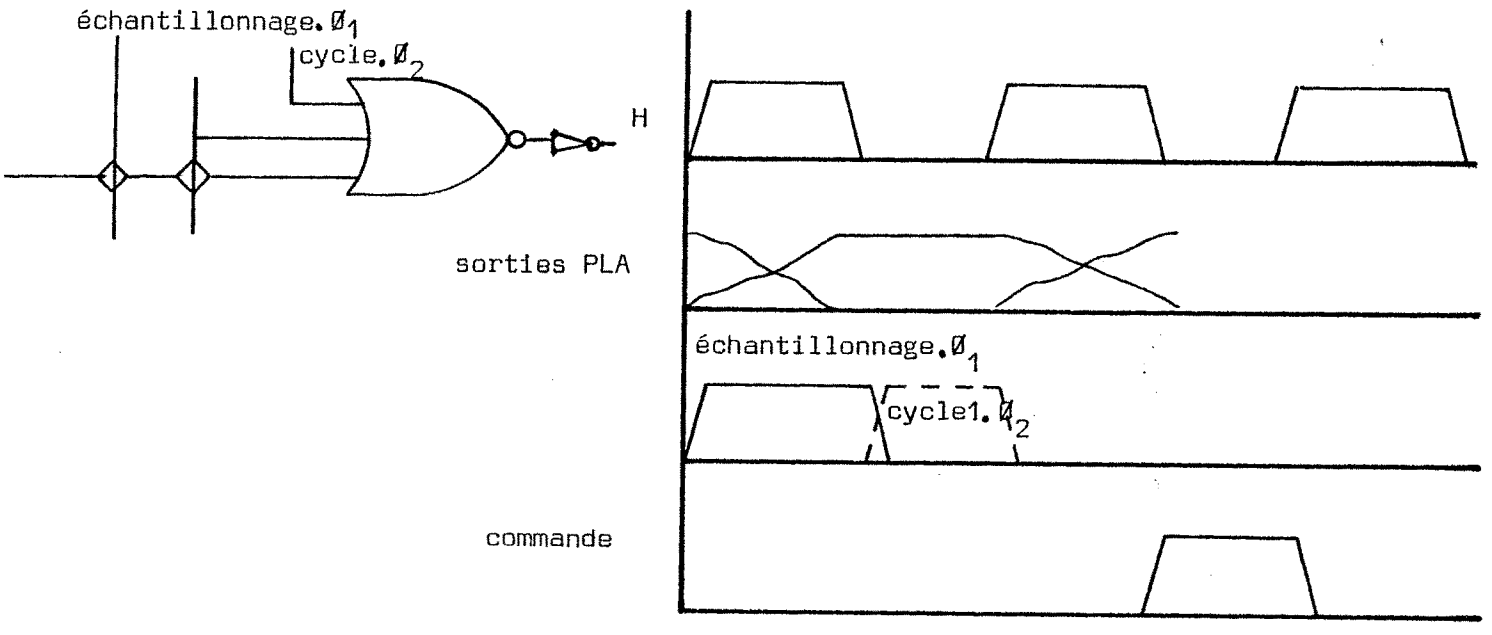


Fig. 5.28

La commande d'échantillonnage et cycle 1 sont générée à partir du champ cycle de la microinstruction par un circuit logique du type de celui décrit dans la figure 5.29.

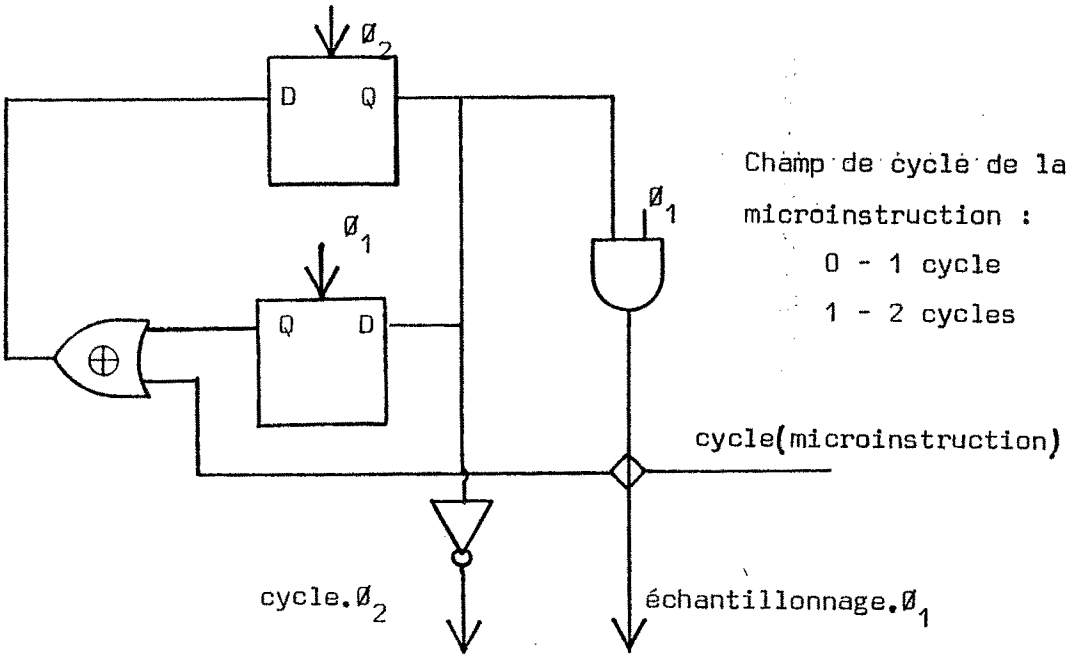


Fig. 5.29 - Commande échantillonnage

Séquencement des PLA's

Les sorties du PLAgc sont échantillonnées pendant ϕ_1 et envoyées au PLAseq. Pour calculer l'adresse de la microinstruction suivante, ce PLA a aussi besoin de l'état de l'opérateur qui est échantillonné à la fin de ϕ_1.

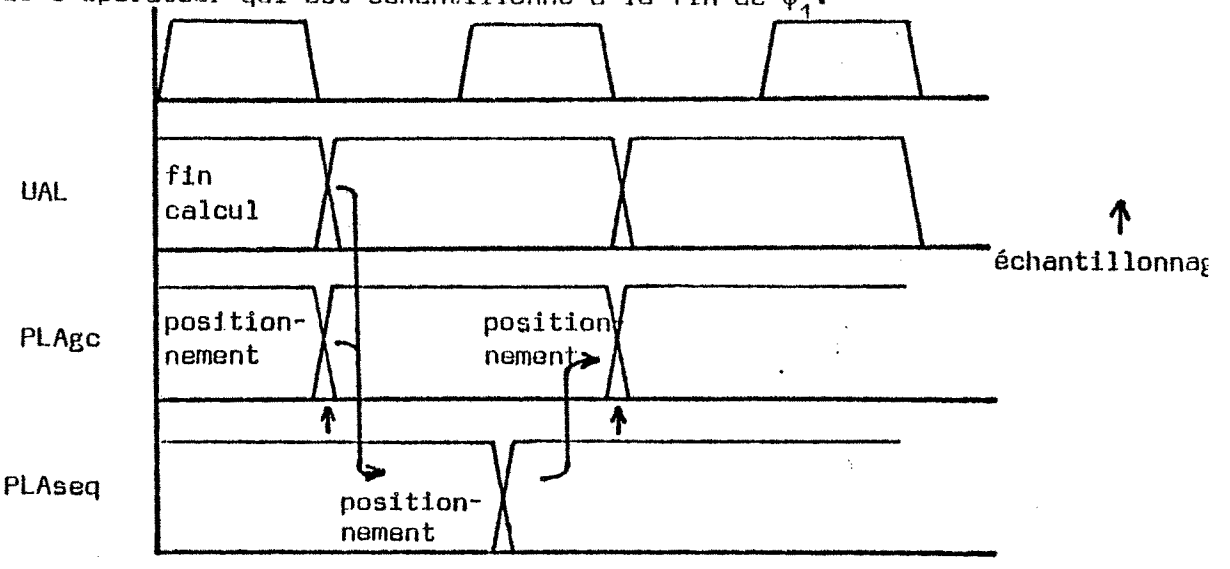


Fig. 5.30 - Séquencement PLAgc et PLAseq

Comme on verra dans 6.2, le point critique du séquencement du processeur sera le temps du positionnement des deux PLAS's qui doit être d'un demi-cycle.

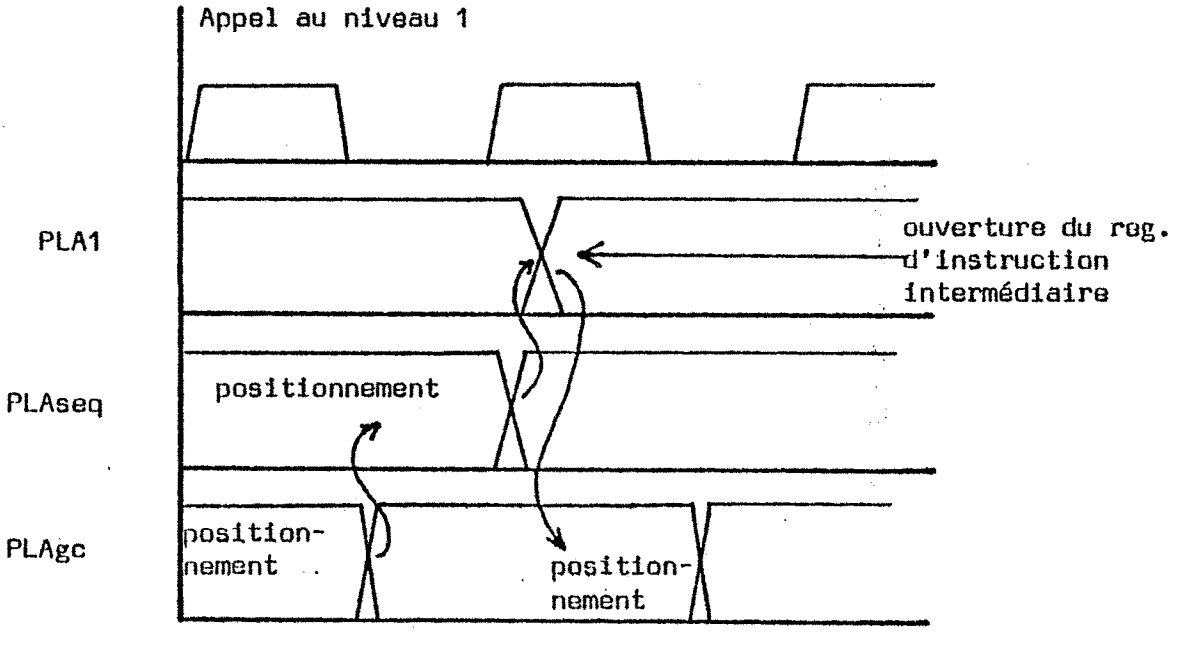


Fig. 5.31 - Appel du premier niveau

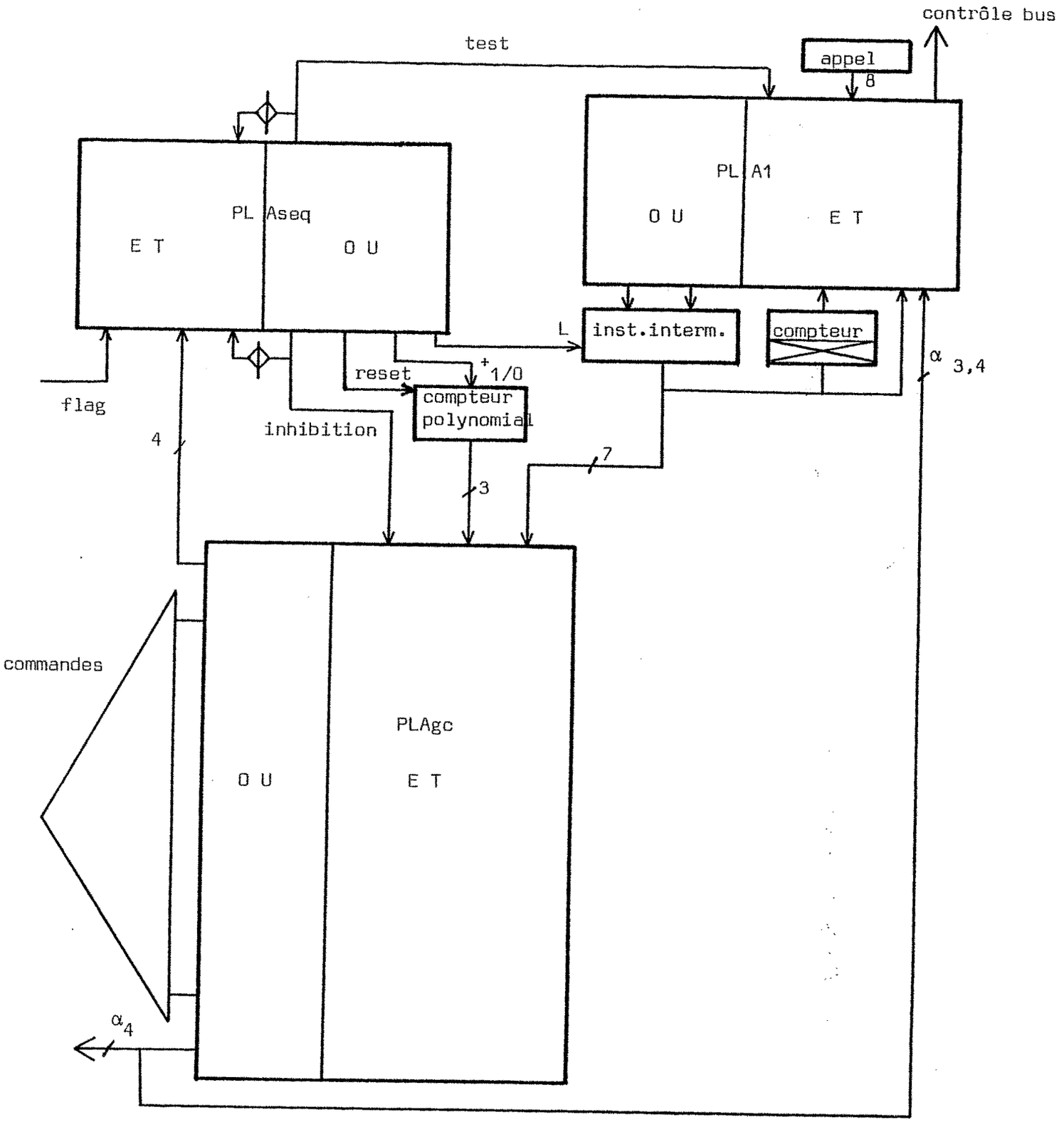


Fig. 5.32 - Partie contrôle simplifiée.

5.2.4. Adaptation du processeur de communication pour une utilisation en barillet

On voulait maintenir la plupart des structures matérielles définies précédemment en les adoptant à ce type d'allocation.

La sauvegarde des paramètres qui définissent l'état du processeur est le problème le plus important à résoudre.

Les informations à sauvegarder sont :

1 - Etat de la partie contrôle
   - appel de la procédure (nom moniteur, procédure),
   - état d'avancement de l'exécution de la procédure ;

2 - Etat de la partie opérative
   - mot d'état ;

3 - Adresse de la position de mémoire à accéder dans le cas des moniteurs ayant des unités d'informations supérieures à 1.

Partie contrôle

La façon qui nous semble plus simple de modifier la partie contrôle est de considérer que dans chaque tranche de temps allouée à un processus, il exécute une instruction intermédiaire.

Il faut donc que le temps nécessaire à l'exécution des instructions intermédiaires soit identique ou au moins qu'il ne dépasse pas une certaine limite.

D'après l'analyse des microinstructions nécessaires pour exécuter les instructions intermédiaires, il nous semble que fixer les tranches de temps à quatre cycles permettrait d'optimiser le temps d'utilisation du processeur noyau.

Le langage intermédiaire à utiliser dans une allocation en barillet doit donc être modifié de façon à ce que les instructions qui exigent plus de quatre cycles soient subdivisées.

Les instructions intermédiaires ayant besoin de moins de cycles seront maintenues en ajoutant des microinstructions vides (NOP) aux microséquences qui les interprètent.

La mémorisation de l'état d'avancement de la procédure se résume donc à:

- appel procédure,
- numéro de l'instruction intermédiaire,
- valeur de la ligne test.

Un Fifo matériel (ensemble de registres à décalage) peut être utilisé pour mémoriser les différents contextes.

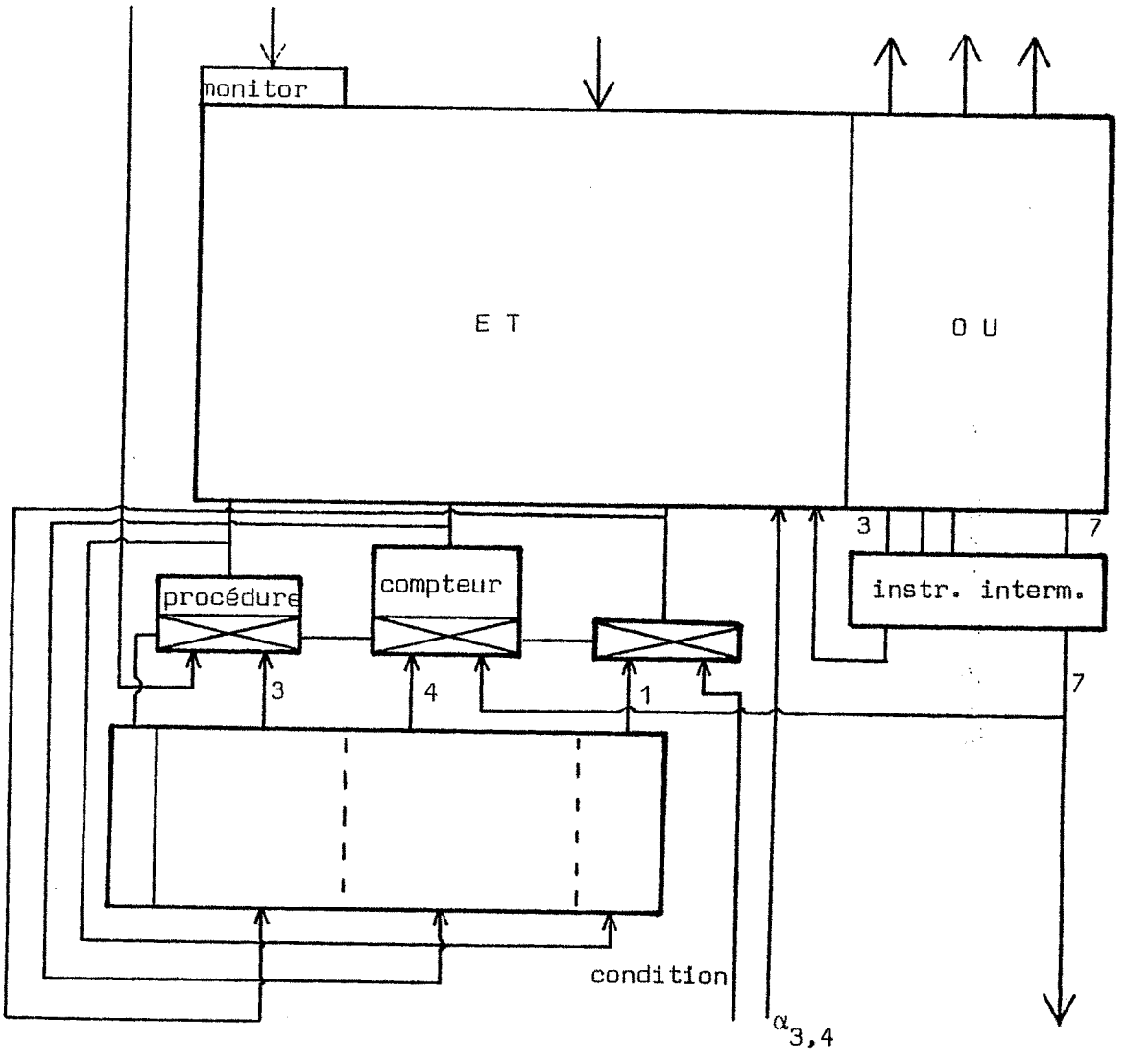


Fig.5.33 - Modification de la partie contrôle pour une utilisation en barillet.

Partie opérative :

Mot d'état - Première solution: mémoriser le mot d'état d'une façon semblable. Deuxième solution: dans les séquences des microprogrammes qui interprètent les instructions intermédiaires standard, l'état de la partie opérative ne conditionne que la microinstruction suivante.

En choisissant un découpage adéquat des instructions intermédiaires, on peut éviter de mémoriser l'état de la partie opérative, la ligne test(5.2.3) résumera les opérations effectuées dans la microséquence précédente.

Adresse mémoire -
Première solution: ajouter à la partie opérative une pile de registres de 12 bits ou pour chaque moniteur dont Noctet soit supérieur à 1.
Deuxième solution: découper l'instruction intermédiaire d'accés (ex.: accés séquenciel (Noctet>1) en plusieurs instructions, chacune permettant d'accéder à 1 octet.

Il est évident que la deuxième solution implique une diminution des performances du mécanisme de communication, il faut cependant voir que le multiplexage temporel du noyau ne visait pas à garantir une augmentation de performances, mais des fréquences d'accés fixes.

Les solutions n'imposant pas d'augmentation significative de matériel, nous semblent donc préférables même si elles entraînent une diminution des performances pures du processeur de communication.

CHAPITRE 6

EVALUATION DE LA SURFACE ET PERFORMANCES DU PROCESSEUR NOYAU

6.1. EVALUATION DE LA SURFACE

Pour choisir l'emplacement des organes du processeur noyau, il fallait d'abord avoir une idée de la taille des grands blocs qui le constituent (Partie opérative, PLA1, PLAseq, PLAgc).

Deux idées concernant l'implantation étaient cependant déjà définies depuis le début :

- La partie opérative doit être assemblée comme un ensemble de tranches, les bus seront disposés verticalement, en aluminium, et les commandes horizontalement, en polycristallin. Les alimentations seront distribuées parallèlement aux bus, en aluminium.
- Le PLAgc chargé de la génération des microordres doit être placé à coté de la partie opérative de façon à minimiser la surface occupée par les interconnexions entre la partie opérative et la partie contrôle.

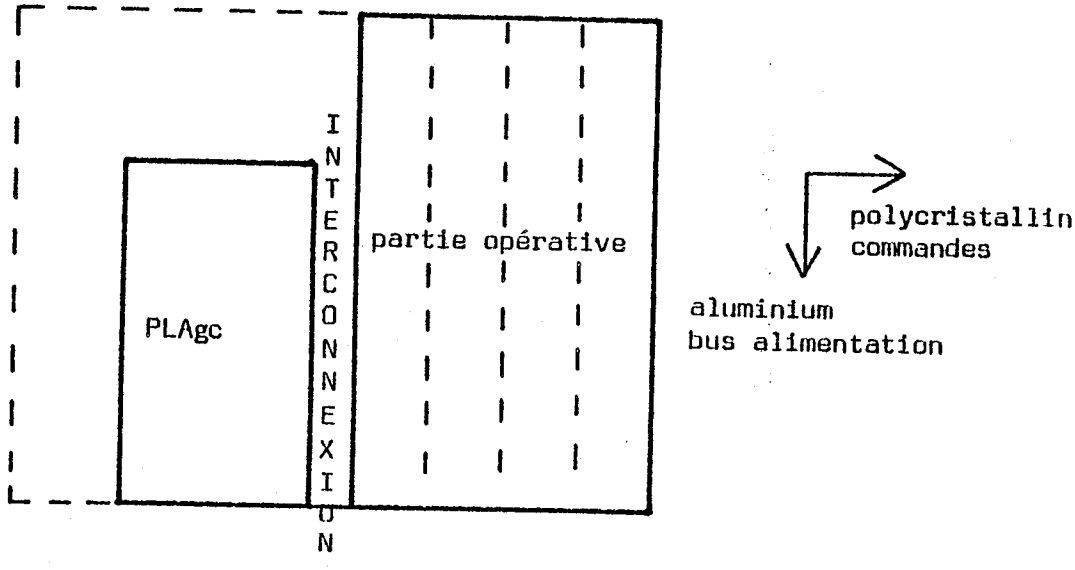


Fig. 6.1 - Plan général du noyau

6.1.1. Méthode utilisée

Au départ de ce projet on visait des densités d'intégration très élevées qui ne seront atteintes qu'avec une technologie NMOS à 2 µm.

Ceci pose un problème pour l'évaluation de la surface car les fabricants diffusent très peu d'information (règles de dessin, densités moyennes, etc...) sur les technologies de pointe.

Pour contourner cette difficulté on a préféré faire les évaluations de surface dans une technologie connue, le NMOS-Locos 6µ (SESCOSEM, Grenoble), et ensuite déterminer des coefficients de réduction par type de fonction de façon à transposer les résultats dans les technologies plus poussées.

Dans l'annexe 3, on présente la méthode suivie et les coefficients qui ont été calculés pour ramener la surface de certains organes, des microprocesseurs plus courants, dans une technologie 6 µm.

6.1.2. Mémoire de communication

La mémoire de communication représente plus de la moitié de la surface du noyau.

Pour minimiser sa surface, on pense utiliser une mémoire dynamique dont le point de mémorisation élémentaire est à peu près six fois plus petit (annexe 3) que le point statique. L'utilisation d'une mémoire dynamique pose cependant quelques problèmes:

1 - Le fonctionnement de tout le circuit doit être dynamique. Ceci ne nous semble pas très important car on avait déjà prévu des organes dynamiques dans la partie opérative.

Essayer de concevoir un circuit complètement statique implique une telle augmentation de la taille de la RAM que les applications envisageables seraient fortement limitées.

2 - Le rafraîchissement du plan mémoire. Ce problème peut être facilement résolu car dans le séquencement du processeur noyau il y a deux cycles non utilisés correspondant à l'allocation du bus et à l'envoi de l'appel.

Il suffit de rajouter un registre "Adresse rafraîchissement" à la section 12 bits de la partie opérative et de faire progresser les adresses de rafraîchissement en utilisant l'additionneur, pendant les deux cycles inoccupés.

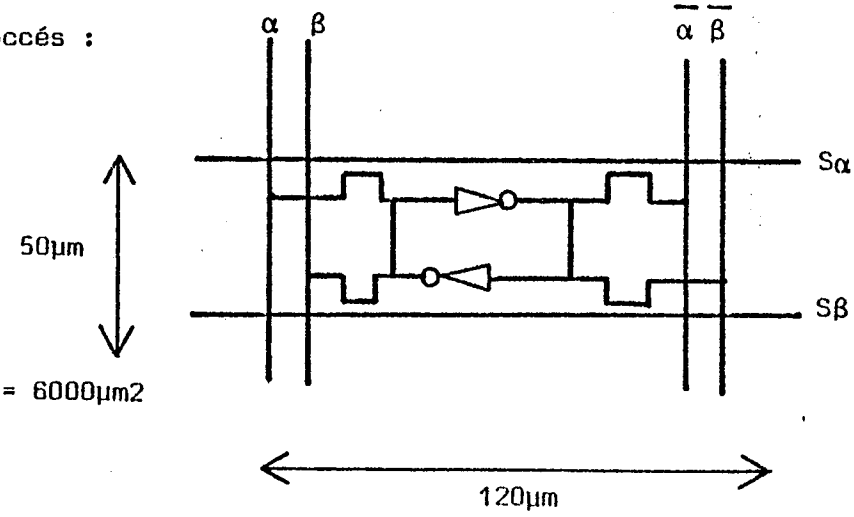
Pour évaluer la taille de la mémoire dynamique, on a pris comme référence le circuit 4116 de MOSTEK fabriqué avec des règles de dessin de 6µm et double polycristallin [SCH.77].

Le point mémoire fait - 435µm^2.
Les amplificateurs et logique périphérique (16 Kbits) - 1,98 mm^2.

6.1.3. Partie opérative

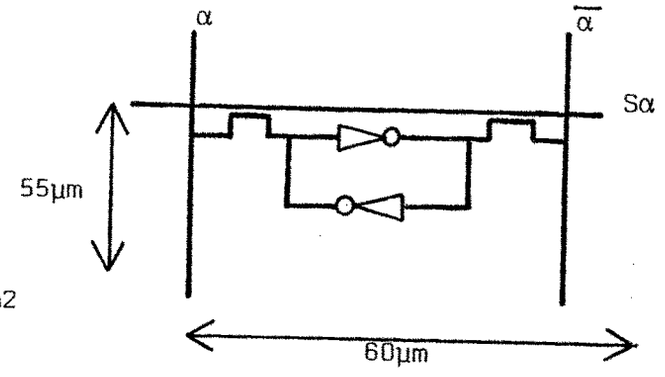
Les briques qui ont été définies en premier furent les éléments de mémorisation du circuit. Leurs largeurs ont été prédéfinies de façon à ce qu'ils puissent être facilement imbricables.

Point à double accés :



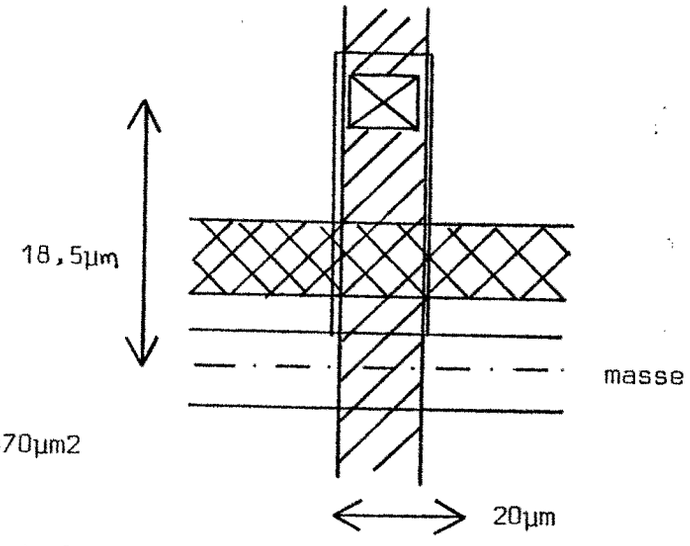
S = 120 x 50 = 6000µm2

Point à simple accés :



S = 60 x 55 = 3300µm2

Point ROM :



S = 20 x 18,5 = 370µm2

La mémoire de travail de la partie 8 bits a donc la taille suivante :

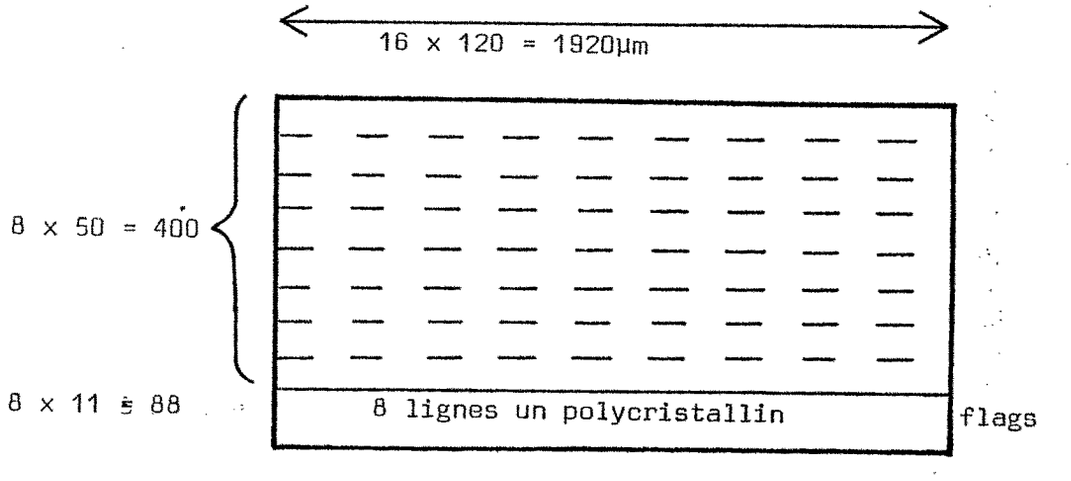


Fig. 6.2 - Mémoire de travail

Amplificateurs des bus :

On prend comme référence les amplificateurs bus du M68000 qui mesurent 90µm x 46,3µm. Comme il s'agit d'un produit HMOS (3,5µm) il faut utiliser le coefficient de normalisation 1,96 (annexe 3) pour le ramener à une technologie 6 µm:

$$90\ \mu m \times 46{,}3\ \mu m \times 1{,}96 = 8167\ \mu m^2$$

Comme on utilisera 32 amplificateurs (2 bus de 16 lignes)

$$8167\ \mu m^2 / 60 \mu m = 136 \mu m$$

Chaque amplificateur fait donc

$$136 \mu m \times 60 \mu m$$

UAL :

On utilise comme référence l'UAL du M6800 qui est presque identique à celle décrite dans 5.2.1.

$$135 \mu m \times 384 \mu m = 51840\ \mu m^2.$$

Comme on a 8 points UAL dans une tranche

$$51840 \mu m^2 / 240 \mu m = 216 \mu m$$

Surface du point UAL : $240 \times 216\ \mu m^2$.

Catalogue (section 8 bits) :

La taille du catalogue dépend du nombre et du type des moniteurs existant dans le système.

Etant donné que chaque ligne du catalogue a 32 points de mémoire, on peut calculer combien de lignes sont nécessaires pour chaque descripteur.

Exemple :

Moniteur	Descripteur (octets)	lignes
I - <256 octets sans réservation sans conditions	4	1
II - <256 octets avec réservation 2 conditions	8	2
III - >256 octets avec réservation 2 conditions	10	2 1/2

$$\text{Surface catalogue} = 1920 \times 50 \sum_{i=0}^{NM} NCDi/4 \quad \mu m^2$$

NCD = nombre de champs du descripteur

NM = nombre de moniteurs

Pour une première évaluation on a considéré que 20 lignes sont largement suffisantes.

Catalogue (section 12 bits)

La partie du descripteur codifié dans la ROM fait 16 bits (adresse base + Noctet).

La largeur de 1920µm de la partie opérative permet d'avoir 96 points de ROM par ligne, donc 6 moniteurs.

Il faut cependant considérer que la ROM est traversée par un bus en aluminium qui permet de faire communiquer les deux sections de la partie opérative (5.2.1).

1920µm - (16 x 15µm) = 1680µm

Chaque ligne peut donc contenir 5 moniteurs ce qui nous obligerait d'avoir un

multiplexeur en sortie de ROM avec trois commandes de sélection. Etant donné que chaque ligne de ROM occupe la moitié d'une commande multiplexeur, on a préféré n'avoir que quatre moniteurs par ligne.

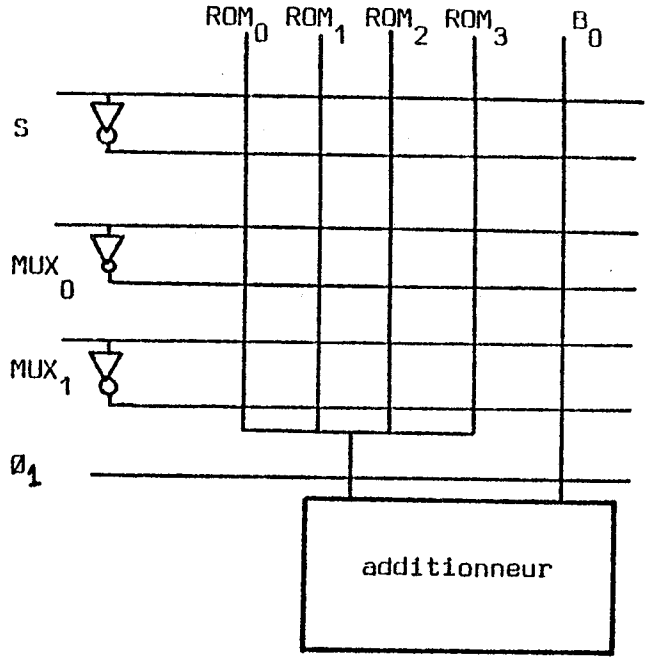
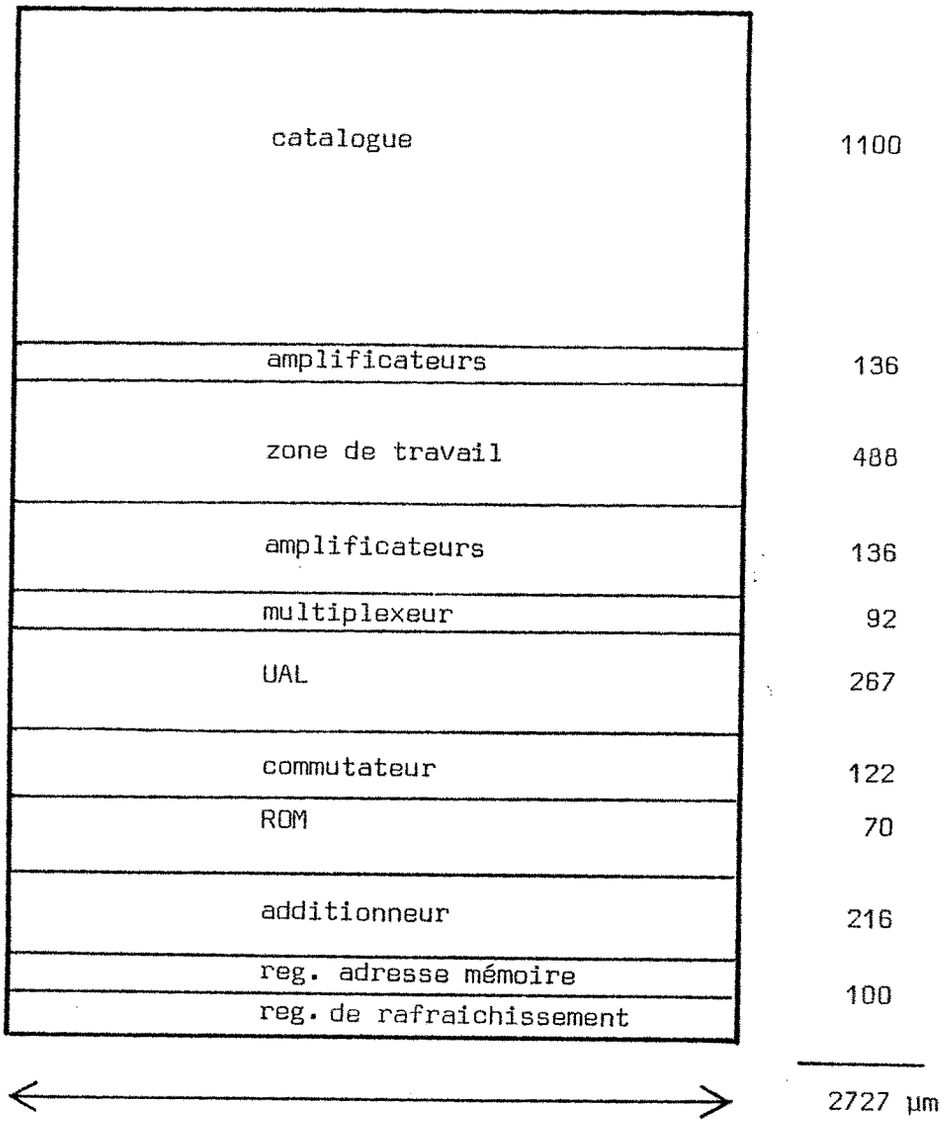


Fig. 6.3. - Descripteur ROM

Les autres éléments ont été évalués avec les méthodes utilisant des dessins symboliques décrites dans l'annexe 3.

Considérant donc un catalogue (RAM) composé de 20 lignes et un total de 16 moniteurs, on obtient la taille suivante pour la partie opérative en technologie NMOS-6µm.



6.4. - Topologie de la partie opérative.

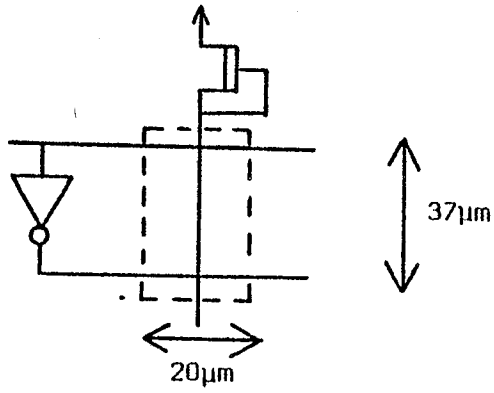
6.1.4. Partie contrôle

Pour l'évaluation des tailles des PLA's on s'est basé sur les microprogrammes des moniteurs standard dont certains exemples sont décrits dans l'annexe 2.

Le PLA1et le PLAseq sont des PLA's classiques composés de deux matrices "ET" et "OU". Dans l'annexe 3 on donne une formule permettant de calculer la taille

d'un PLA en fonction des dimensions de la cellule élémentaire, du nombre d'entrées, de sorties et de monômes.

En technologie NMOS-LOCOS 6µmun point de PLA a les dimensions suivantes :



$$S = [Ex37+(S + \lceil \frac{S}{8} \rceil)x20] \times [M + \lceil \frac{M}{8} x20] \ \mu m^2$$

E - entrées
M - numéro de monômes
S - sorties

Les estimations basées sur la taille des programmes écrits en langage intermédiaire correspondant aux moniteurs standard (annexe 1) nous ont permis de calculer que $M \underset{\sim}{<} 40$ pour le PLA1. (E = 19, S = 15 - Fig. 5.24).

$$S_{PLA1} = 1043 \times 900 \ \mu m^2$$

Pour le PLAseq, les différentes possibilités de séquencement donnent $M \underset{\sim}{<} 22$ (E = 6, S = 5 - Fig. 5.22).

$$S_{PLAseq} = 322 \times 460 \ \mu m^2$$

Le PLAgc est considérablement différent à cause de l'utilisation d'une deuxième matrice en ligne brisée, on a donc calculé séparément la taille des deux matrices

et considéré qu'un facteur de réduction de deux serait envisageable pour la matrice "OU".

Le nombre de monômes a été obtenu à partir des microprogrammes du type de ceux présentés dans l'annexe 2 . M = 58 .

Surface matrice "ET" :

$$S = (Ex37) \times [(M + [\frac{M}{8})x20]$$

Entrées - 16

Monômes - 58

$$S = 370 \times 1300\ \mu m^2$$

Surface matrice "OU" :

$$S = [(M + [\frac{M}{8})x20] \times \frac{[(S + [\frac{S}{8})x20]}{\text{facteur de réduction}}$$

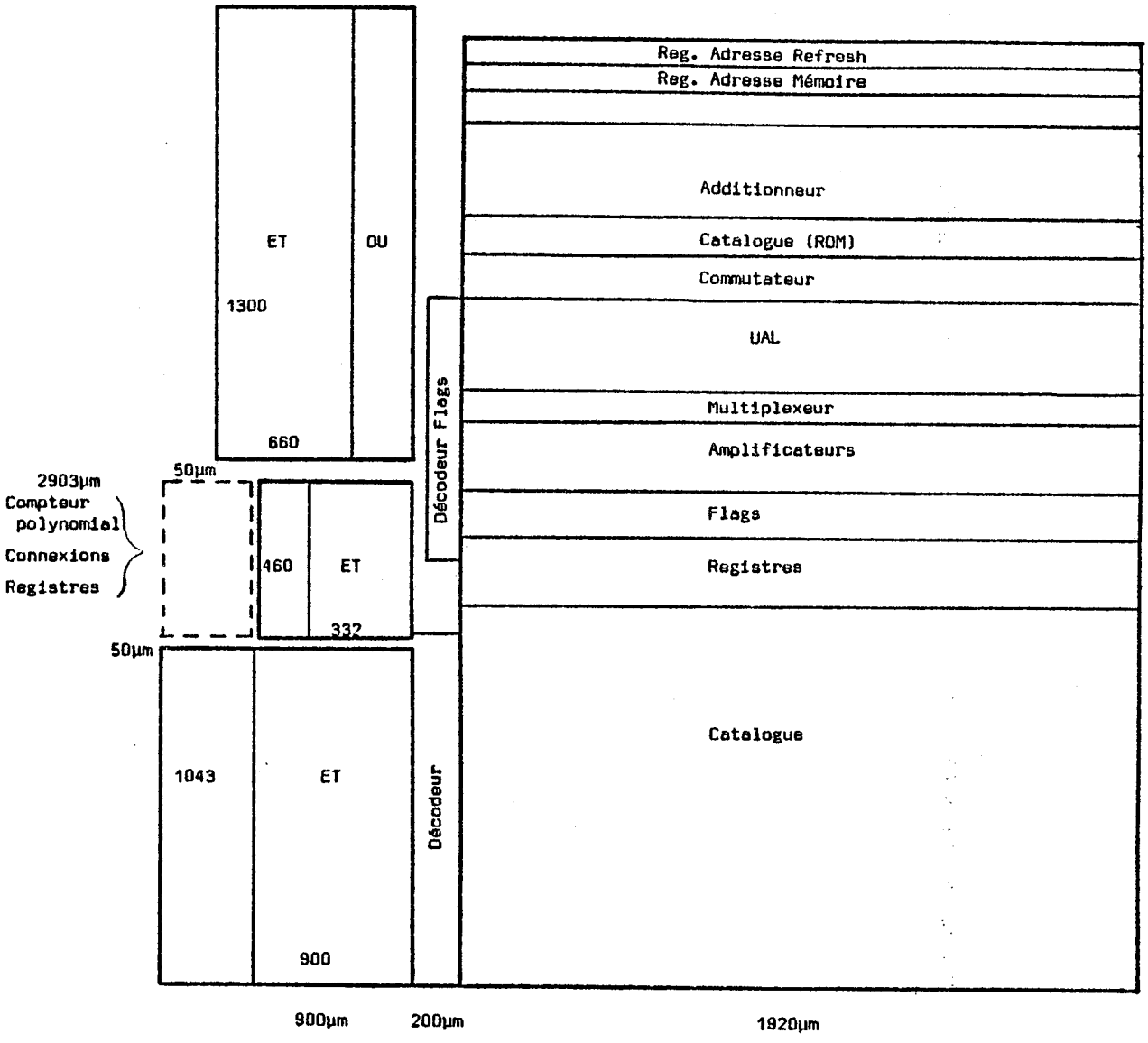
Sorties - 32

$$S = 1300 \times \frac{720}{2} = 1300 \times 360\mu m^2$$

Surface du $PLAg_C = 1300 \times 730\ \mu m^2$

Pour choisir l'emplacement de la partie contrôle on a essayé de trouver une topologie qui nous permette de minimiser les surfaces occupées par les interconnexions et de respecter la longueur imposée par la partie opérative.

Les interconnexions entre PLA (registres, portes de transmissions) ont été évaluées en utilisant des dessins symboliques ("stick diagram" [WIL.77]).



6.5. - Topologie du processeur noyau

Mémoire de communication

La mémoire de communication doit s'adapter aux contraintes imposées par la taille de la partie opérative.

Si on suppose une mémoire de communication avec 2 Koctets, on obtient :

processeur noyau	8,76 mm2
mémoire de communication	14,23 mm2
	22,99 mm2

6.1.5. Evolution de la surface en fonction de la technologie

Dans l'annexe 3 on décrit les coefficients de réduction de la surface de différents types de fonctions pour des technologies plus poussées que le NMOS 6µ.

On peut donc calculer la taille du noyau que l'on a pris comme exemple dans HMOSI (largeur grille: 3,5µm) et dans une technologie homogène (voir annexe 3) à 2µm.

On commence par diviser le noyau dans les différents types de fonctions pour lesquelles des coefficients ont été calculés :

RAM statique	2,73mm2
RAM dynamique	8,76mm2
ROM ou PLA	2,06mm2
Logique aléatoire	9,44mm2

HMOSI :

RAM statique	2,73/1,55	= 1,76
RAM dynamique	8,76/2,19	= 4
ROM ou PLA	2,06/3,08	= 0,67
Logique aléatoire	9,44/1,96	= 4,82
		11,25mm2

Technologie 2µm :

RAM statique	2,73/2,2	= 1,24
RAM dynamique	8,76/4	= 2,19
ROM ou PLA	2,06/5,44	= 0,38
Logique aléatoire	9,44/3,8	= 2,48
		6,29mm2

6.2. PERFORMANCES

6.2.1. Temps de cycle du processeur noyau

Dans le chapitre précédent on a fait quelques hypothèses sur les temps de réponse des différents éléments du processeur noyau :

1 - temps de cycle du processeur = temps d'accés à la mémoire de communication ;

2 - temps d'accés à la mémoire catalogue ou aux registres de la zone de travail inférieur à la moitié du temps de cycle.

Comme il a été déjà mentionné, la technologie à partir de laquelle il est envisageable de réaliser des circuits du type MOSAIC est le NMOS 2µ.

Il est cependant difficile de quantifier le temps de cycle d'un processeur dans une technologie 2µm car, à l'exception de quelques mémoires, très peu de circuits ont été encore produits.

Pour avoir une référence sur laquelle baser les évaluations de performances on a pris délibérément une option pessimiste en considérant un temps de cycle de 100 ns.

On peut analyser les hypothèses précédentes en fonction de cette valeur de temps de cycle :

1/ Temps d'accés à la mémoire de communication = 100 ns.
Dans des produits existants (MK4516, I2118) fabriqués avec des règles de dessin de 3,5µm [LEE.79] on obtient des temps d'accés de 80ns. Dans MOSAIC ces temps sont encore plus favorables car on n'a pas le retard imposé par les amplificateurs E/S compatibles avec des charges TTL.

2/ Temps de cycle du processeur = 100ns.
Le 68000 fabriqué avec des règles 3,5µm a un temps de cycle de 125ns.

3/ Temps d'accés mémoire catalogue <50ns.
La courbe présentée dans la figure 1.2 [JEC.79] Montre que pour des mémoires RAM statiques de 1 Kbits les temps d'accés avec des règles de 2µm sera de l'ordre de 20ns.

Le seul point critique à l'intérieur du processeur noyau devient le temps de positionnement des PLA's du deuxième niveau d'interprétation qui doit être de 50ns. Des simulations basées sur le modèle de transistor HMOS permettent d'espérer obtenir des temps de cet ordre de grandeur si l'on utilise des PLA's dynamiques avec une précharge commandée.

6.2.2. Temps de propagation du bus

Un point critique dans la définition du séquencement du circuit est la relation entre le séquencement du processeur et le temps de propagation des données dans le bus commun.

Deux problèmes d'origine différente interviennent dans la définition du séquencement du bus :

a/ Problème de l'isochronie.

Une solution globale au niveau de l'architecture a été proposée dans le chapître 4 pour résoudre le problème du retard dans la distribution de l'horloge.

Dans le même chapître on a aussi montré que les problèmes d'isochronie ne commencent à devenir vraiment importants que dans les technologies inférieures à 1µm.

Les évaluations de surface et temporelles que l'on fait dans ce chapître sont, pour des raisons évidentes, basées sur les caractéristiques des technologies utilisables à court terme.

Le retard introduit dans la distribution de l'horloge est donc négligeable par rapport au temps total de propagation du signal (1 ns pour un fil de 10mm en aluminium).

b/ Charges capacitives des portes attaquées par le bus.

La charge capacitive du bus est relativement importante, on a donc considéré un temps de propagation d'environ 100ns car, d'une part cette valeur peut être facilement obtenue en utilisant des amplificateurs de bus adaptés, et, d'autre part, un temps de propagation identique au temps d'accés à la mémoire commune simplifie considérablement le séquencement.

La figure 6.6 montre un chronographe de séquencement de la machine basé sur les hypothèses qui viennent d'être faites.

6.3. MODELES DE SIMULATION

On utilise QNAP qui est un programme d'évaluation des performances d'architecture d'ordinateurs.

Une description de ce programme peut être trouvée dans le manuel d'utilisation [CII.80], on se limitera donc à faire remarquer les notions qui nous semblent importantes pour notre système.

Les entités existant dans les modèles QNAP sont :

1/ Les stations composées d'une file d'attente et d'un ou plusieurs serveurs.

Les serveurs exécutent un certain service sur les clients et les envoient dans d'autres stations ou les font sortir du système.

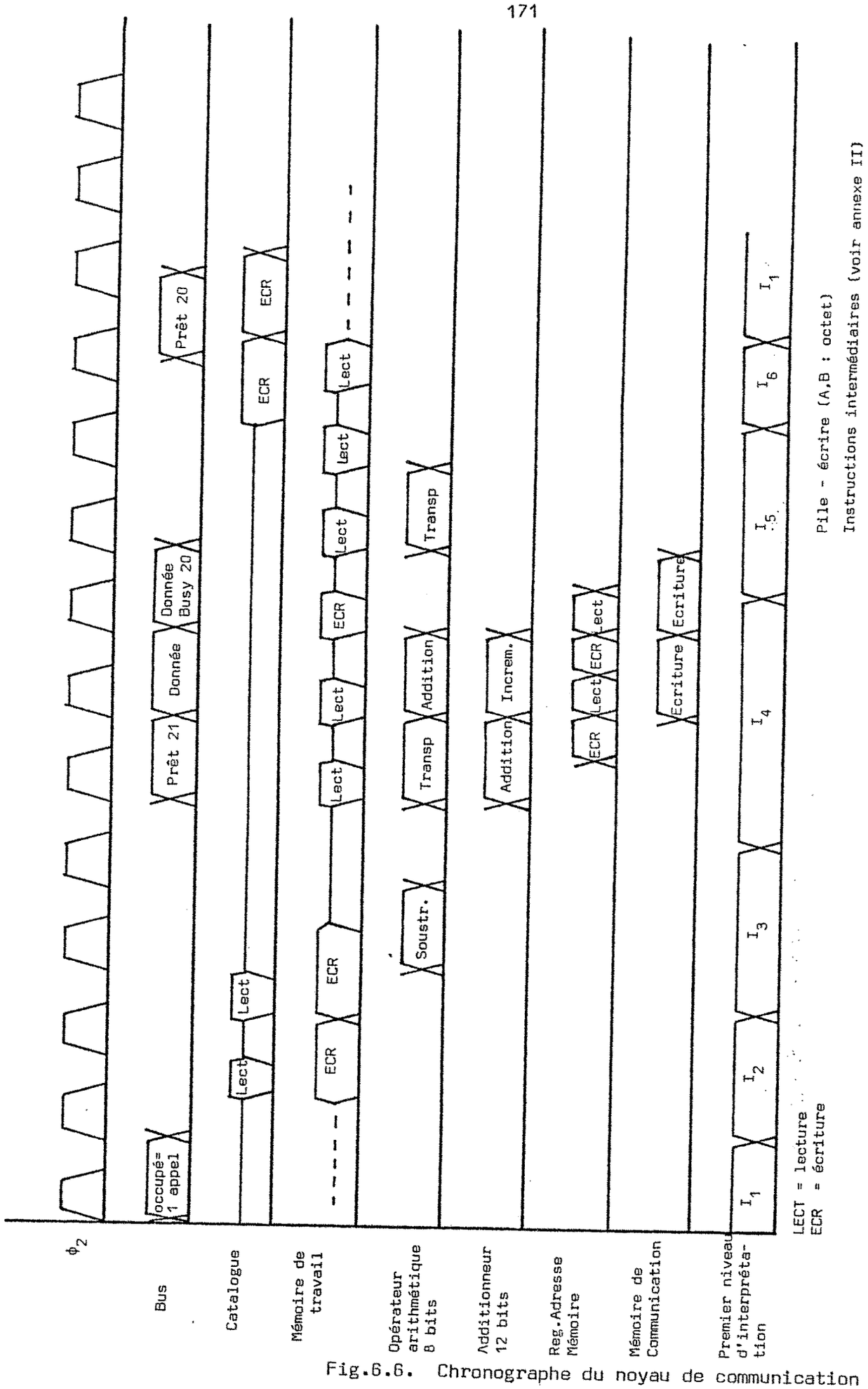


Fig.6.6. Chronographe du noyau de communication

Dans QNAP il y a plusieurs types de stations, celles qui sont utilisées dans notre modèle sont :

- les stations génératrices de clients,
- les stations à simple serveur,
- les stations de synchronisation.

2/ Les clients qui sont produits par les stations génératrices et sont ensuite envoyés dans les files d'attente d'autres stations ou retirés du système.

Une autre caractéristique importante de QNAP est la possibilité d'exprimer des contraintes de synchronisation entre les clients des différents services.

Des stations déclarées comme "ressources" ou "sémaphores" peuvent être reservées en exclusion mutuelle par les clients. Les opérations de synchronisation ont la même notation et la même signification que celles qui ont été définies pour les sémaphores [DIJ.68].

Modèle MOSAIC

On veut modéliser le dialogue existant sur le bus commun et déterminer quelle est l'influence du noyau de communication sur les performances des processus existants dans les modules.

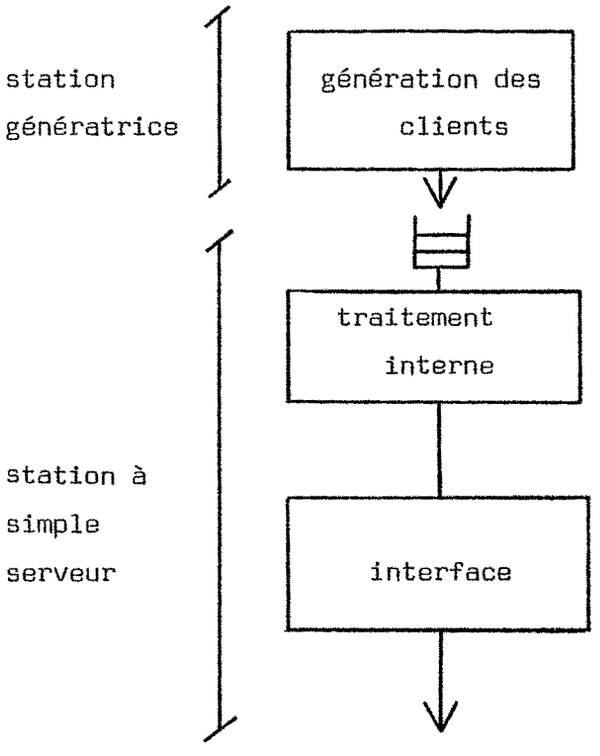
Les clients de notre modèle sont donc les données transférées qui déclenchent des appels aux moniteurs externes.

Le bus est déclaré comme une "ressource" que les clients (transferts) doivent posséder avant de continuer l'exécution.

Les processus résidant dans les modules sont représentés par deux modèles.

1/ Processus temps réel avec une fréquence de traitement imposée par le milieu extérieur.

Exemple: Processus devant échantillonner une entrée avec une certaine fréquence.



Loi statistique (ex.: Poisson, constante etc...) correspondant à la génération des données traités par le processus.

Loi statistique correspondant au traitement réalisé par le processus.

Synchronisation avec bus P(Bus)
Temps d'exécution de la procédure par le processeur noyau
Libération du bus V(Bus).

Remarques :

1/ La station qui génère les clients peut être considérée comme le milieu extérieur qui impose une certaine fréquence de génération. Le nombre de clients dans la file d'attente, entre la station génératrice et la station qui modélise le traitement interne du processus permet de savoir si celui-ci suit la fréquence imposée par le milieu extérieur.

2/ La loi statistique de génération et les paramètres correspondant doivent être déterminés par le concepteur à partir du cahier des charges.

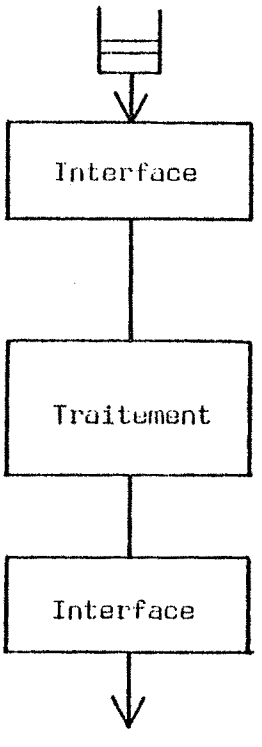
3/ La loi statistique correspondant au traitement interne doit être établie en fonction des caractéristiques du module utilisé.

4/ Le temps d'exécution de la procédure peut être obtenu en comptant les cycles nécessaires pour l'exécution des instructions intermédiaires correspondantes dans la procédure.

Dans un système d'aide à la conception des circuits MOSAIC cette valeur serait obtenue directement à partir de la compilation du programme de spécification.

Un des résultats de la compilation mentionnée dans 2.4 serait l'indication du nombre de cycles nécessaires pour l'exécution de chaque procédure des moniteurs externes.

2/ Processus sans fréquence d'entrée imposée par le milieu extérieur.



P(Bus)
Lecture des données (temps d'exécution de la procédure de lecture)
V(Bus)

Loi statistique correspondant au traitement réalisé par le processus

P(Bus)
Ecriture des données (temps d'exécution de la procédure d'écriture)
V(Bus)

Client envoyé vers la file d'attente d'un autre processus ou retiré du système

Les remarques 3 et 4, faites dans le modèle précédent, s'appliquent à nouveau.

Comme exemple on reprend l'application ANTIOPE décrite dans le chapitre 2.

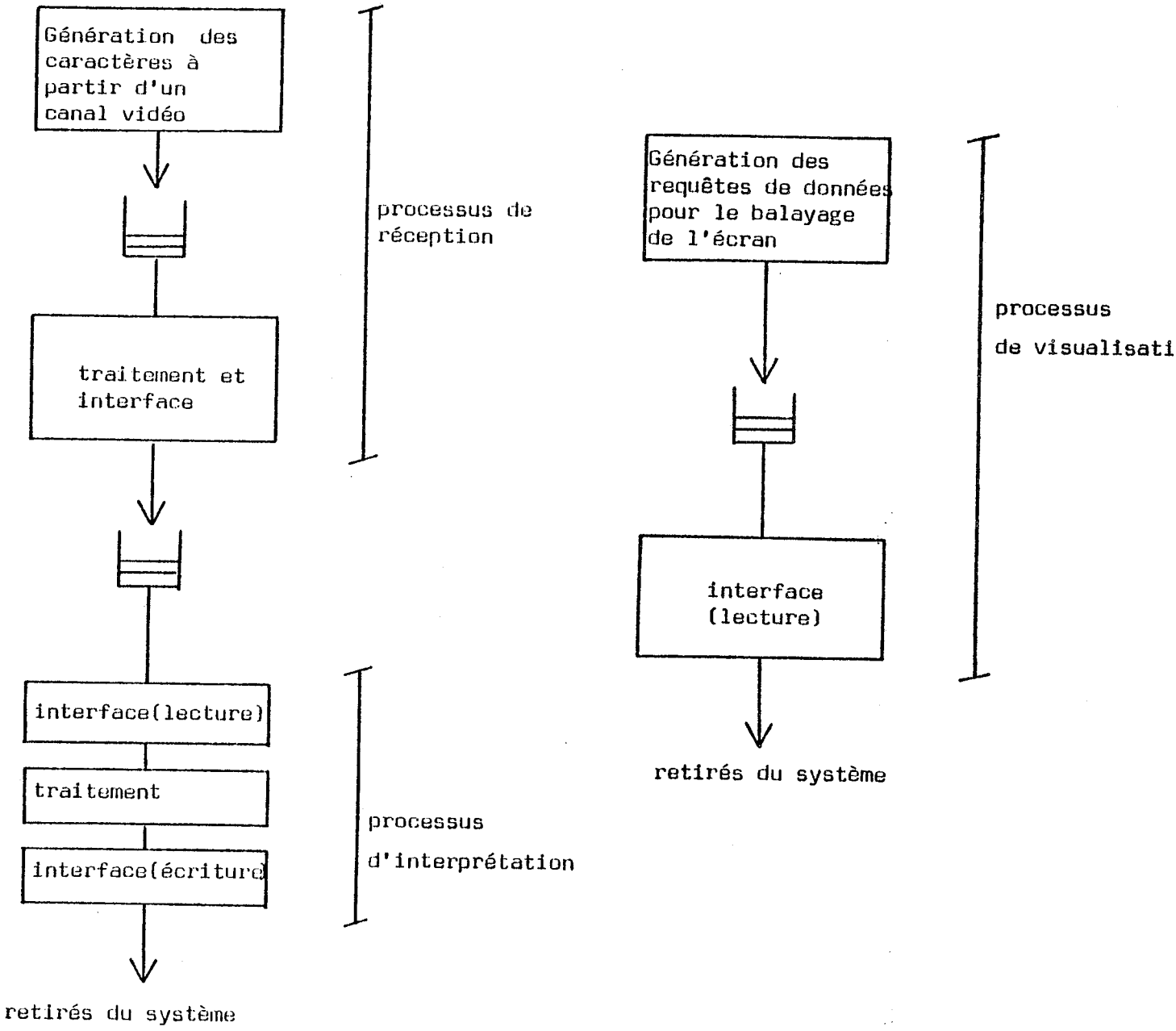
Cette application constitue un bon test des performances du système car elle présente des contraintes temps réel très strictes (fréquence de réception des caractères et fréquence de balayage de l'écran).

Le processus de réception et le processus de traitement sont facilement représentables en utilisant respectivement les modèles 1 et 2. Le problème se pose pour le processus de visualisation qui n'est pas synchronisé avec les autres et est obligé de travailler à une cadence fixe imposée par le balayage de l'écran TV.

On a choisi de le représenter en utilisant le modèle 1 et de faire sortir du système les clients générés par les deux chaînes.

Dans l'allocation du bus on considère le schéma de priorités fixes (4.1.1), dans le modèle d'ANTIOPE le plus prioritaire est le processus de réception suivi du processus de visualisation.

Programme et résultats (annexe IV).

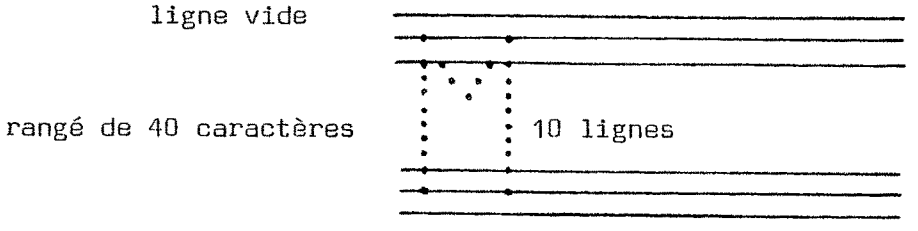


Simulations effectuées

1 - Le premier système simulé peut être considéré comme le pire des cas :

a/ le processus de réception transfère un caractère à chaque appel de la procédure et on suppose que la fréquence de réception est le maximum admissible (fréquence bit 6,2Mhz - fréquence caractère 0,775 Moctet/s).

b/ on considère l'algorithme de balayage exécuté sans qu'aucun tampon ne soit utilisé dans le module de visualisation.
L'algorithme de rafraîchissement est le suivant :



temps affichage caractère= 1,185µs

Pendant 10 lignes, le processus de visualisation lira la même rangée de 40 caractères.

Remarque : Pour avoir une idée des performances à exiger du processeur noyau on a fait ces simulations en variant le temps de cycle. On commence à 75ns et on continue jusqu'à 150ns par incréments de 25ns.

Résultats (annexe IV) :

- temps de cycle de 75ns.

Les résultats montrent que le système arrive à suivre la fréquence imposée.

- temps de cycle de 100ns.

Dans ce cas le processus de réception et de visualisation n'arrivent pas à tenir les cadences imposées par le milieu extérieur.

Les résultats prouvent donc que la stratégie de transfert choisie n'est pas bien adaptée et qu'il faut considérer des solutions où le flux d'information sur le bus soit plus réduit.

2 - On suppose qu'il existe un tampon dans le module de réception de façon à permettre qu'à chaque appel de la procédure d'écriture dans le FIFO on puisse envoyer deux caractères :

Les simulations montrent que le système suit les cadences imposées par les entrées jusqu'à un temps de cycle de 125ns.

3 - On pourrait aussi modifier la partie de visualisation en maintenant la partie de réception du modèle 2 :

On considère que dans le module de visualisation il y a un tampon de 40 caractères ce qui permet d'éviter de relire la même ligne 10 fois.

Le remplissage du tampon peut se faire pendant la ligne vide, le retour ligne et la première ligne affichée

$$t_{total} = 2 \times (40 \times \text{temps affichage caractère} + \text{temps retour ligne})$$

$$t_{caractère} = t_{total}/40 = 2{,}67\mu s$$

Les simulations montrent que le mécanisme de communication suit les fréquences jusqu'à un temps de cycle de 175ns.

	75 ns	100 ns	125 ns	150 ns	175 ns
modèle 1	oui	non			
modèle 2	oui	oui	limite	non	
modèle 3	oui	oui	oui	oui	limite

Les simulations que l'on vient de présenter visaient deux objectifs :

1/ vérifier si un temps de cycle de 100 ns était très pénalisant pour des applications où il faut travailler à des vitesses élevées ;
2/ montrer les possibilités qu'aurait un concepteur de modifier l'architecture de son système de façon à l'adapter aux besoins de l'application.

En modifiant deux ou trois paramètres dans le modèle QNAP, il peut obtenir un ensemble de résultats qui lui permettent d'effectuer des choix plus précis.

CHAPITRE 7

HIERARCHIE DES VARIABLES DE COMMUNICATION

7.1. DECOMPOSITION D'UNE APPLICATION

L'architecture décrite dans le chapître précédent est une architecture à mémoire commune et donc la performance globale du système est limitée par la bande passante du mécanisme de communication.

Si l'on suppose que Ncycle représente le nombre moyen de cycles, du processeur noyau, nécessaires pour exécuter une procédure d'un moniteur externe, la fréquence maximale d'utilisation du noyau est :

$$f\ max = \frac{1}{Ncycle \times tcycle}$$

Pour des applications qui n'utilisent que des moniteurs simples, comme ceux programmés dans l'annexe 1, la valeur de Ncycle est approximativement $\simeq$ 12.

$$f\ max = \frac{1}{12 \times 100\ ns} = 833 \times 10^3\ appel/s$$

A propos de la valeur de cette fréquence, il faut faire une remarque importante:

Si l'on considère que le temps d'accés à une mémoire dynamique est de l'ordre de 100ns, on pourrait penser qu'une solution où les modules accéderaient directement à la mémoire (3.2.2) serait beaucoup plus performante (100ns + temps d'allocation < 200ns => f = 5Mhz).

Il faut cependant remarquer que les deux chiffres ne sont pas comparables car dans cette dernière solution les processus n'accédent qu'aux données dans la mémoire et doivent exécuter localement les algorithmes des variables de communication.

Dans le temps moyen de l'appel il y a donc un pourcentage significatif ($\simeq$ 70%) qui correspond à l'exécution de l'algorithme des procédures ce qui évidemment conduit à une diminution du travail local des modules et à une optimisation du nombre d'allocations et d'accés à la mémoire (3.2.2).

Dans certaines applications où les processus ont des fréquences de transfert élevées imposées par le milieu extérieur, une situation de saturation du processeur noyau peut donc être atteinte.

Dans ce chapître on étudie la possibilité de diviser la charge du mécanisme de communication en faisant une partition de l'application en sous-systèmes faiblement couplés.

Pour que cette solution puisse améliorer les performances globales de l'ensemble il faut que les caractéristiques de l'application permettent de faire une sous-division réelle de l'application et donc que les sous-systèmes créés échangent peu d'information entre eux.

L'exemple suivant permet d'illustrer ce qui vient d'être dit :

Supposons un petit terminal de bureautique composé par un système de traitement de texte et par un coupleur téléinformatique avec un réseau local.

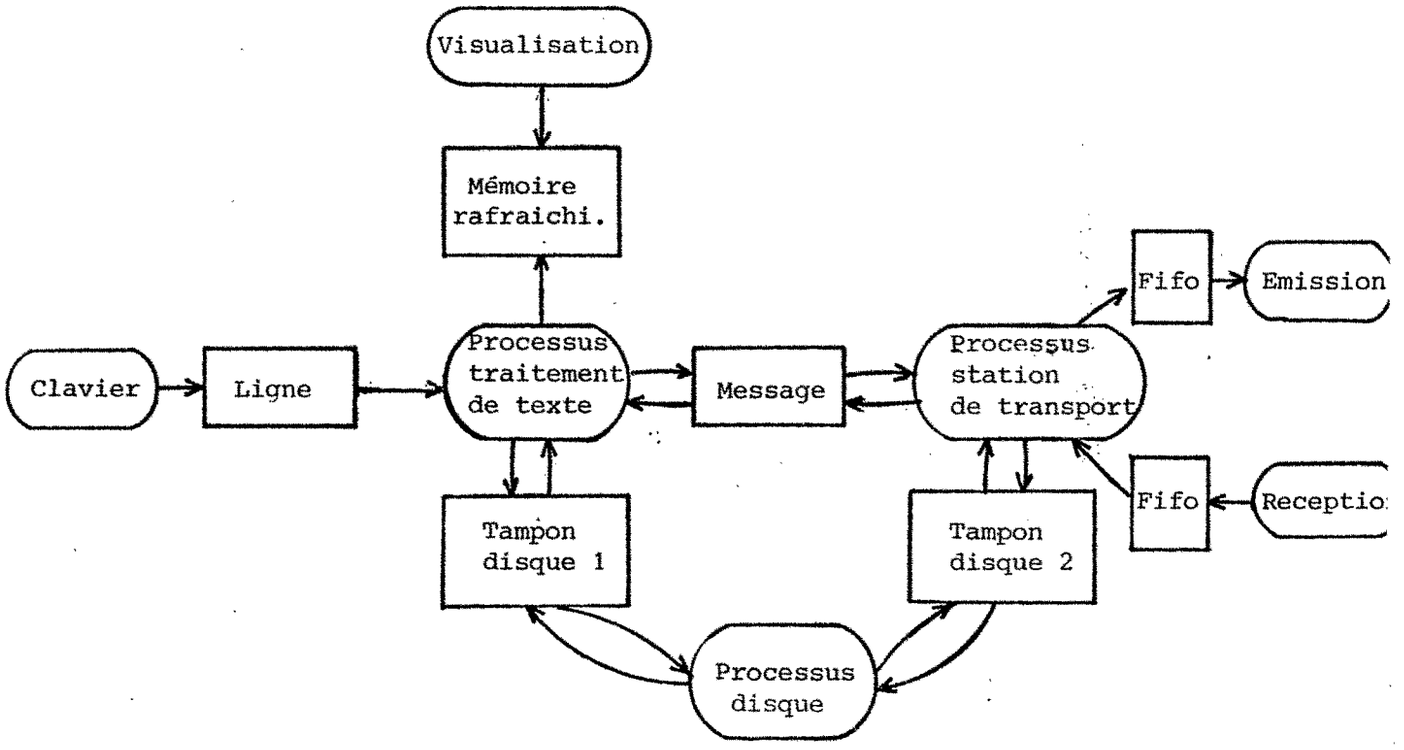


Fig.7.1 - Terminal de bureautique

Le coupleur assure le routage de l'information sur le réseau et n'accéde au tampon-disque 2 que lorsqu'il reçoit un message à destination du poste de travail.

De son côté, la partie de traitement de texte travaille localement et n'accéde au coupleur de téléinformatique que s'il y a un message à envoyer ou à recevoir.

Si l'on utilise un seul processeur noyau, des problèmes de débit peuvent se poser car il y a quatre processus (visu, disque, réception téléinformatique, émission téléinformatique) qui ont des fréquences de travail relativement élevées.

Cette application peut facilement être découpée en deux sous-systèmes, la partie de traitement de texte et la partie de téléinformatique. Les deux sous-systèmes travaillent en parallèle à l'exception des moments où des messages à destination de ce poste de travail sont reçus ou envoyés.

Pour permettre aux deux sous-systèmes de communiquer, on a créé une nouvelle entité, les moniteurs globaux, qui permettent aux processus définis dans des sous-systèmes différents de communiquer entre eux.

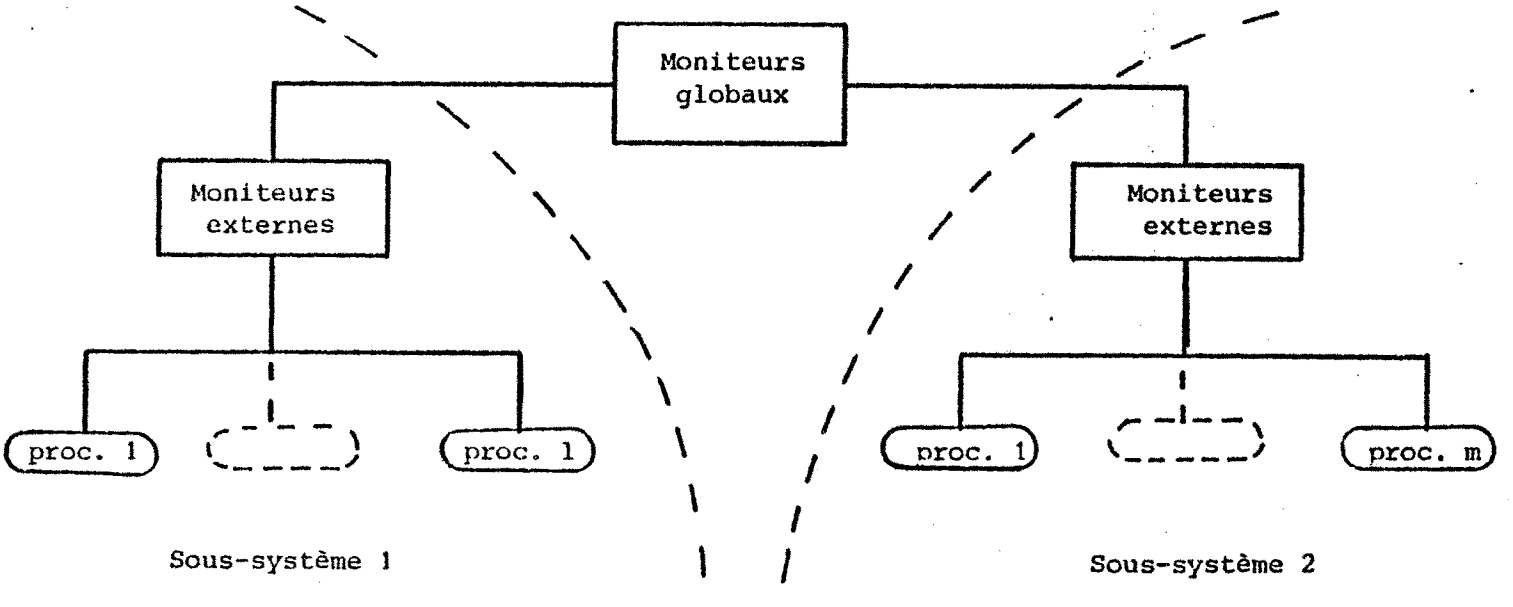


Fig. 7.2 - Niveaux de moniteurs.

7.2. EXTENSION DU LANGAGE DE SPECIFICATION

L'extension du langage de spécification est très simple. On crée deux nouveaux types : Subsystem et Global monitor qui permettent de représenter les deux entités qui viennent d'être introduites.

```
NOM_SOUSSYSTEME = subsystem
                      use (liste des procédures des moniteurs globaux
                           utilisés par les modules du sous-système)
                  "Définition des modules"
end ;
NOM = global monitor
              définition classique d'un moniteur
      end ;
```

7.3. IMPLEMENTATION DES MONITEURS GLOBAUX

Deux solutions sont possibles :

1 - Dédier un processeur noyau à l'exécution des moniteurs globaux. Cette solution serait extrêmement pénalisante du point de vue surface et le processeur noyau serait très sous-employé car, par la définition même du problème, les moniteurs globaux sont peu utilisés.

2 - Faire exécuter les moniteurs globaux par un des processeurs noyau qu'implémente un des sous-systèmes. Cette solution est beaucoup plus économique du point de vue surface mais elle impose une pénalisation du temps d'exécution des procédures des moniteurs globaux. Cependant si les sous-systèmes sont faiblement couplés, cette augmentation du temps d'exécution aura peu d'influence sur les performances de l'ensemble.

La façon la plus simple pour faire communiquer les deux processeurs noyau est de connecter les deux bus communs quand les moniteurs globaux sont accédés.

D'autres solutions plus performantes (ex.: bus dédié entre les deux processeurs, partage de mémoire) auraient pu être utilisées mais elles impliquent toutes l'utilisation d'organes matériels plus complexes. Etant donné l'augmentation de surface qui représente déjà un processeur noyau, il nous semble que des solutions plus complexes au niveau matériel risqueraient de n'avoir aucune viabilité dans les technologies envisageables à moyen terme.

L'exemple précédent peut s'implémenter de la façon suivante :

Moniteurs globaux { Message
Tampon-disque 2

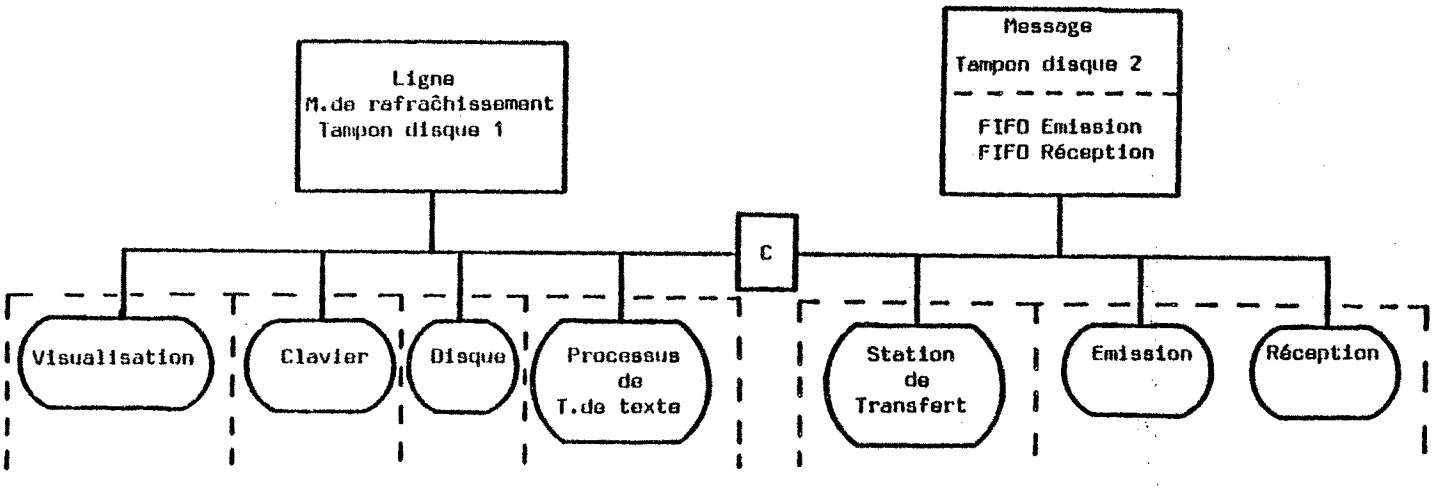


Fig. 6.3 - Implémentation du terminal de bureautique

Le commutateur "C" permet de connecter les deux bus quand un moniteur global est accédé.

7.4. PROTOCOLE D'UTILISATION DES BUS

Pour faciliter l'exposition du protocole on considère le schéma suivant qui correspond à une simplification de la solution précédente :

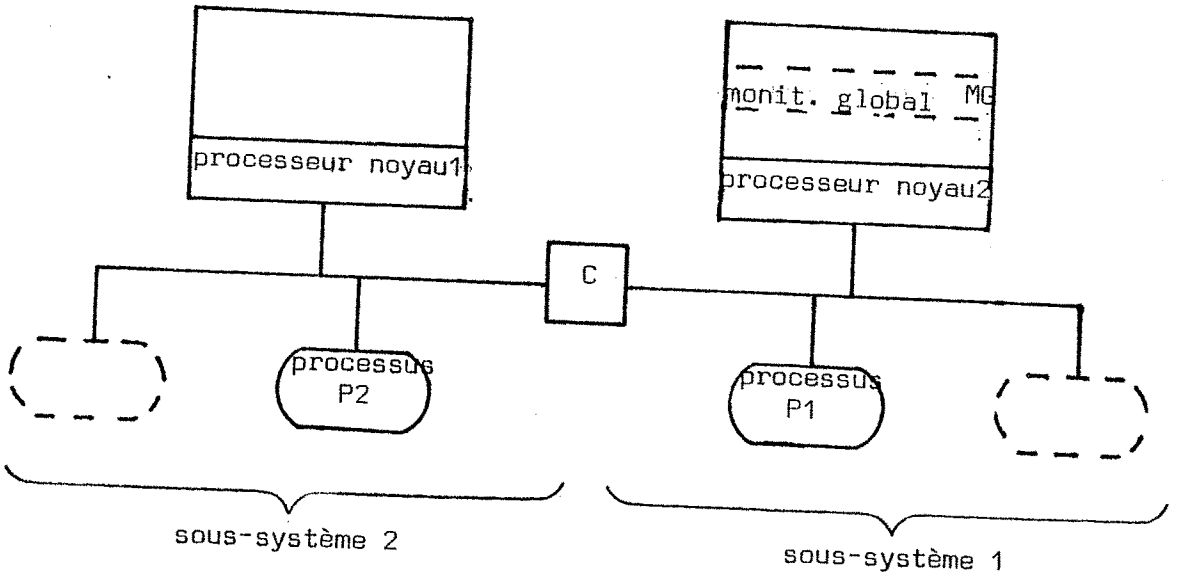


Fig. 6.4 - Accés à un moniteur global.

1/ Le processus P_2 veut exécuter une procédure de Moniteur global MG :

- P_2 alloue le bus B2,
- Envoie l'appel à une procédure de MG,
- Le processeur noyau 2 reconnaît un appel à un moniteur global et envoie une demande de connexion au commutateur,
- Le commutateur doit être considéré par le sous-système 1 comme un processus (il faut que la priorité de ce processus soit élevée car il bloque tout le sous-système 2),

- Quand la connexion est établie, le commutateur notifie le processeur noyau 2 et celui-ci repetera l'appel,

- Exécution normale de la procédure comme si un seul processeur noyau était présent.

2/ Le processus P_1 exécute une procédure d'un moniteur global MG :

L'exécution de la procédure se déroule de la façon standard décrite dans 4.3.

Le seul problème nouveau est la possibilité que l'état du moniteur global soit modifié de façon à ce qu'un "signal" sur une condition doive être déclenché.

Dans ce cas il faut le transmettre au sous-système 2 car il peut concerner un processus de ce sous-système. Pour le faire, le commutateur doit allouer le bus 2 et mettre les deux bus en communication de façon à ce que la valeur de la ligne "continuer1" se propage à la ligne "continuer2".

Cependant, cette implémentation peut se révéler peu performante car le "signal" peut ne pas concerner les processus du sous-système 2 et donc la mise en communication des deux bus est une perte de temps inutile.

Ce problème peut se résoudre très facilement si l'on considère deux types de "signal", un global qui peut concerner tous les processus du système et un autre local, qui ne s'adresse qu'aux processus du sous-système où les moniteurs globaux sont implémentés.

Pour pouvoir générer automatiquement, par compilation, ces deux types de "signal" on va modifier légèrement le langage de spécifications.

Les sous-systèmes sont représentés de la même façon, mais les moniteurs globaux sont décrits dans le sous-système où ils seront exécutés.

```
S1 = subsystem
        use (MG.lire) ;

     end ;
S2 = subsystem
        MG = global monitor

             end ;
        end ;
```

La liste des procédures utilisées par le sous-système S1 permet au compilateur de détecter quelles sont les conditions qui peuvent produire un "signal" global et celles qui ne pourront donner origine qu'à un "signal" local.

Commutateur :

Dans la description des procédures on a déjà spécifié les fonctions qui doivent être exécutées par le commutateur.

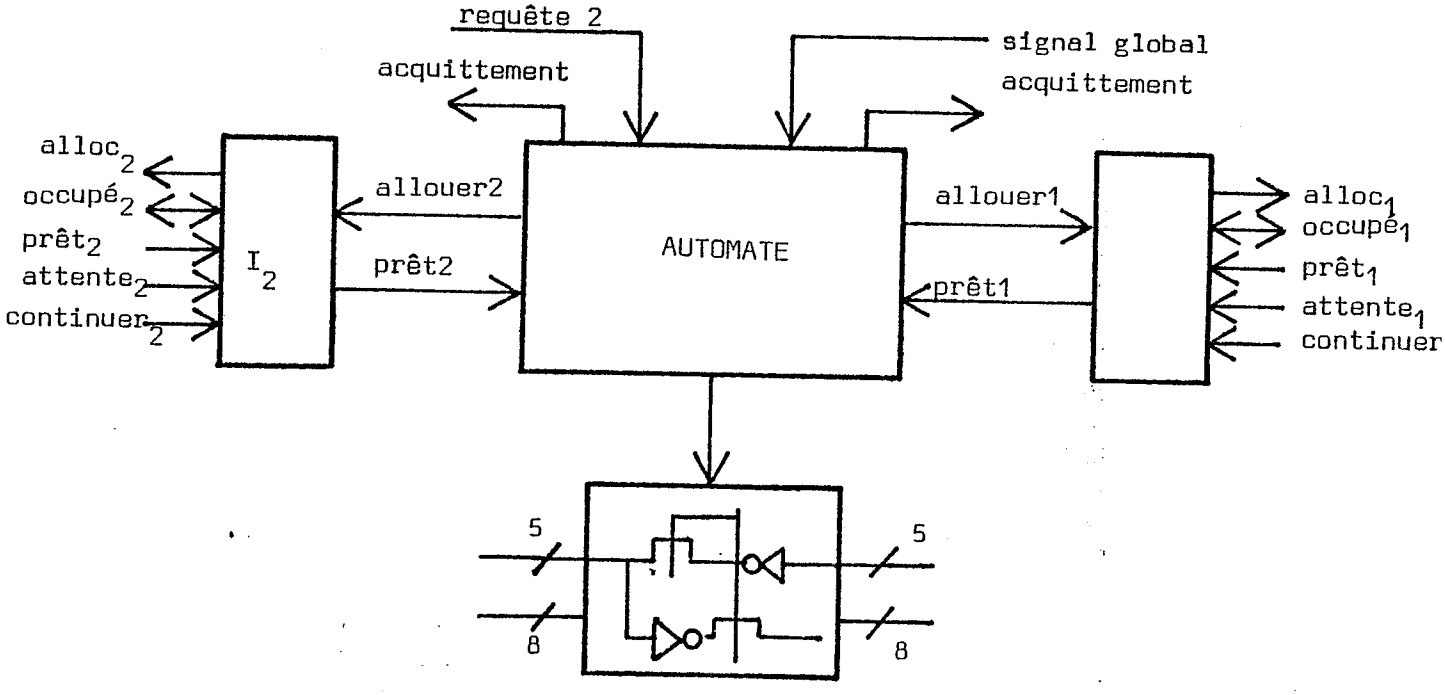


Fig. 7.6 - Commutateur

I_1 et I_2 sont des interfaces identiques à celles des processus qui seront décrites dans 9.2.

Temps d'exécution des procédures :

Sous-sytème 1 : seulement le "signal global" peut retarder l'exécution des procédures. Si l'on suppose que ces actions de synchronisation sont très rares par rapport à l'utilisation normale, on peut considérer que le sous-système 1 n'est pratiquement pas affecté par l'utilisation des moniteurs globaux.

Sous-système 2 : le temps d'exécution d'une procédure d'un moniteur global est considérablement augmenté :

- temps d'allocation bus 1,
- envoyer appel,
- détection d'un appel à un moniteur global*,
- attente connexion*,
- exécution de la procédure.

Les deux actions signalées sont additionnées au temps normal d'exécution.

Si le sous-sytème 1 est très chargé on aura une attente de connexion approximativement égal à la moitié de la moyenne du temps d'exécution des procédures du sous-système 1.

Si la fréquence d'accés aux moniteurs globaux est élevée, la performance du sous-système 2 est donc nettement inférieur à celle d'une solution avec un seul processeur noyau.

Modèle QNAP :

Dans l'évaluation des performances, il nous intéresse surtout de modéliser les appels aux moniteurs globaux à partir du sous-système 2.

Le modèle de simulation du système est composé de deux modèles indépendants représentant les deux sous-systèmes.

L'interface des processus du sous-système 2 qui accédent aux moniteurs globaux est modifié de la façon suivante :

```
P(BUS2) ;
CST ("temps de détection par le processeur noyau 2
      qu'il s'agit d'un appel à un moniteur global") ;
  P(BUS1) ;
  CST ("temps d'exécution de la procédure") ;
  V(BUS1) ;
V(BUS2) ;            CST: loi constante
```

7.5. LIMITATIONS DE CETTE APPROCHE

Surface :

La surface occupée est évidemment la limitation majeure de cette solution car d'une part la division en sous-systèmes suppose une application complexe, donc avec beaucoup de modules et, d'autre part, la surface occupée par les deux processeurs noyau (8,2mm2 en technologie 2µm) est déjà importante.

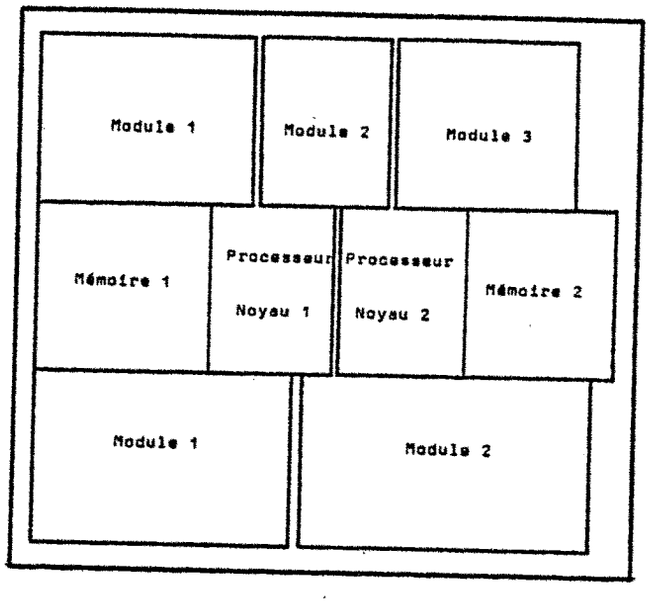


Fig.6.6 - Schéma d'implantation

Implantation :

L'implantation d'un circuit avec plus de deux processeurs noyau devient très difficile car il serait nécessaire de rajouter des lignes de commandes des bus et la disposition de la mémoire et des modules ne peut pas avoir la même régularité.

Possibilité de découpage d'une application :

Cette limitation provient de la nature même des applications car il faut que le découpage corresponde à des propriétés réelles de localité dans la communication entre les modules. Dans le cas contraire, les performances de l'ensemble peuvent être inférieures à celles de la solution avec un seul processeur.

CHAPITRE 8

SOLUTION DISTRIBUEE

8.1. EXECUTION LOCALE DES MONITEURS

La solution hiérarchique ne peut être appliquée qu'à un certain nombre réduit d'applications où une décompostion en deux sous-systèmes faiblement couplés peut être faite.

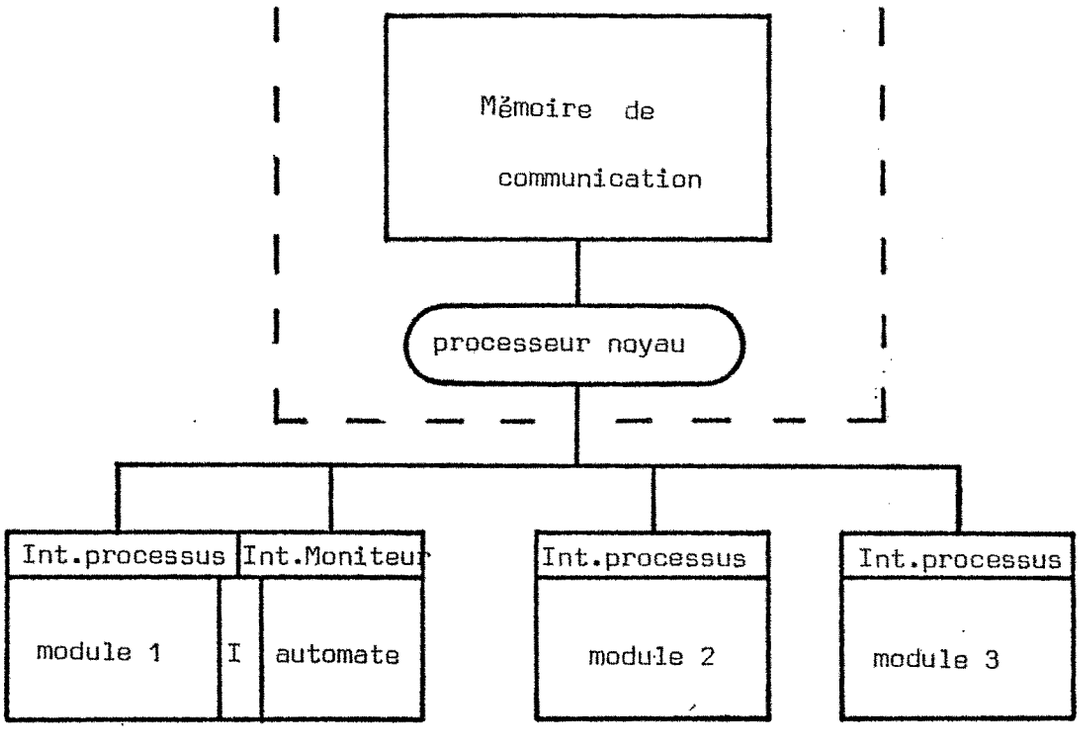
Dans la grande majorité des cas les problèmes de saturation du noyau de communication arrivent quand il y a un, ou plusieurs, processus qui ont des fréquences très élevées d'accés à des moniteurs externes, pendant une certaine période de fonctionnement du système (ex.: un processus ayant une fréquence d'échantillonnage très élevée à certains moments de son algorithme).

Pour éviter que ces processus temps réel ne bloquent le mécanisme de communication pendant une période importante, on peut implémenter localement les moniteurs externes qu'ils accédent avec des fréquences élevées. De cette façon on assure, au processus couplé, une réponse presque immédiate du mécanisme de communication et celui-ci ne supporte pas les pointes d'activité du processus temps réel.

Deux solutions étaient possibles pour réaliser cette implémentation locale d'un moniteur :

1 - faire exécuter le moniteur dans les parties opérative et contrôle où s'exécutent les processus ;
2 - concevoir un automate qui serait chargé de l'exécution des moniteurs et qui serait placé à côté du module pouvant communiquer directement avec celui-ci sans utiliser le bus.

Dans la première solution on doit arrêter le processus pendant les accés au moniteur. Ceci est en contradiction avec notre hypothèse initiale de considérer que le processus temps réel avait une fréquence de travail très élevée.



8.1. - Schéma fonctionnel

Cette implémentation ne peut être utilisée que dans des situations où les processus couplés au moniteur travaillent très lentement, ou sont même arrêtés, comme par exemple pendant l'initialisation du système ou quand le processus est en attente d'une paramétrisation.

La deuxième solution est par contre beaucoup plus intéressante car elle permet au moniteur et au processus de s'exécuter en parallèle et de communiquer sans utiliser le bus commun.

Il est cependant évident que l'implémentation locale, par un automate, d'un moniteur ne peut être justifiée que si les données qu'il véhicule représentent

un flux d'information important.

On va étudier la deuxième solution plus en détail car la première correspond uniquement à une modification de l'interface bus du module décrite dans 9.2.

8.2. PROTOCOLE D'UTILISATION DU BUS

a/ allocation

On continue à utiliser le même protocole de gestion du bus défini au chapître 4.

La seule différence étant que le multiplexage temporel du mécanisme de communication ne nous semble pas présenter d'intérêt si l'on distribue l'exécution des moniteurs.

Le multiplexage temporel et la distribution de l'exécution des moniteurs visent à résoudre deux problèmes de caractéristiques opposées. Dans le multiplexage temporel on voulait assurer une réponse continue du mécanisme de communication à des processus ayant des fréquences très régulières d'accés mais pas très élevées. La solution distribuée vise par contre à retirer du système de communication les requêtes des processus qui ont épisodiquement des fréquences d'accés très élevées.

L'exécution des moniteurs distribués ne peut donc pas être multiplexée et par conséquent ne multiplexer que l'utilisation du processeur noyau imposerait un séquencement de l'utilisation du bus et des automates très compliqué.

b/ exécution des procédures

Dans le chapître 4 on a considéré que le processus qui envoie un appel à un moniteur externe maintient le contrôle du bus pendant tout le temps de l'exécution de la procédure.

Cette solution a été justifiée par le fait que le noyau ne pouvait pas exécuter plus d'une requête à la fois. Libérer le bus pendant la première partie de l'exécution des procédures ne pouvait qu'augmenter très faiblement la performance globale, et correspondait par contre à une sensible augmentation de la complexité matérielle.

Si l'on distribue l'exécution des moniteurs par des automates indépendants on a plusieurs points d'exécution et la première restriction est donc éliminée.

Il faut donc évaluer à nouveau quel est le gain de temps d'exécution et simultanément quel accroîssement de complexité impose un protocole de gestion du bus où le processus le libère après l'envoi de l'appel au moniteur externe.

On peut utiliser comme exemple de base la lecture de deux octets dans un FIFO

Solution sans libération du bus	Exécutée par :	
	noyau	automate
allocation par le processus	1	1
envoi de l'appel	1	1
initialisation	2	1
test condition d'entrée	2	2
transfert	5	4
réévaluation des conditions	2	2
sauvegarde	1*	0
	14 cycles	11 cycles

*La sauvegarde prend deux cycles mais le deuxième peut être utilisé pour l'allocation de l'appel suivant.

Solution avec libération du bus :

Phase 1 -

Le processus envoie l'appel et il est obligé de fournir son identification :

	cycles noyau	cycles automate
allocation par le processus	1	1
envoi appel	1	1
envoi non processus	1	1
libération bus / allocation	0	0
	3	3

Quand la procédure peut être exécutée, le processeur noyau ou l'automate, invitent le processus à recevoir ou à émettre les données.

	cycles noyau	cycles automate
allocation par le noyau ou les automates	1	1
identification processus	1	1
transfert	4	3
réévaluation des conditions	2	2
sauvegarde	1	0
	9	7

Dans l'exemple considéré, le gain est pratiquement nul et il faut aussi considérer que le nombre d'éléments en compétition pour l'allocation du bus est plus élevé dans la deuxième solution et donc les temps d'attente d'allocation risquent d'être plus longs.

En contrepartie, l'augmentation matérielle est significative :

- les interfaces processus doivent avoir la logique nécessaire pour reconnaître les invitations à émettre et à recevoir ;
- le processeur noyau et les automates ont une interface leur permettant d'allouer le bus commun.

  Les automates susceptibles d'être appelés par plusieurs processus doivent avoir aussi des FIFOs pour mémoriser les appels en attente de traitement.

Il nous semble donc que le gain en performance ne justifie pas l'augmentation de complexité du matériel et par conséquent à continuer à utiliser le protocole sans libération du bus.

8.3. STRUCTURE INTERNE DES AUTOMATES

Les automates sont constitués par une partie opérative simple, une partie contrôle qui leur permet de faire exécuter les procédures des moniteurs et de contrôler le bus, et une mémoire où sont stockées les données :

a/ partie opérative

Variables :

- pointeurs,
- taille,
- compteurs pour les conditions,
- éventuellement le nom de l'utilisateur.

La partie opérative a une structure à deux bus de façon à pouvoir être implantée en utilisant la bibliothèque de briques mentionnée dans 5.2.2.

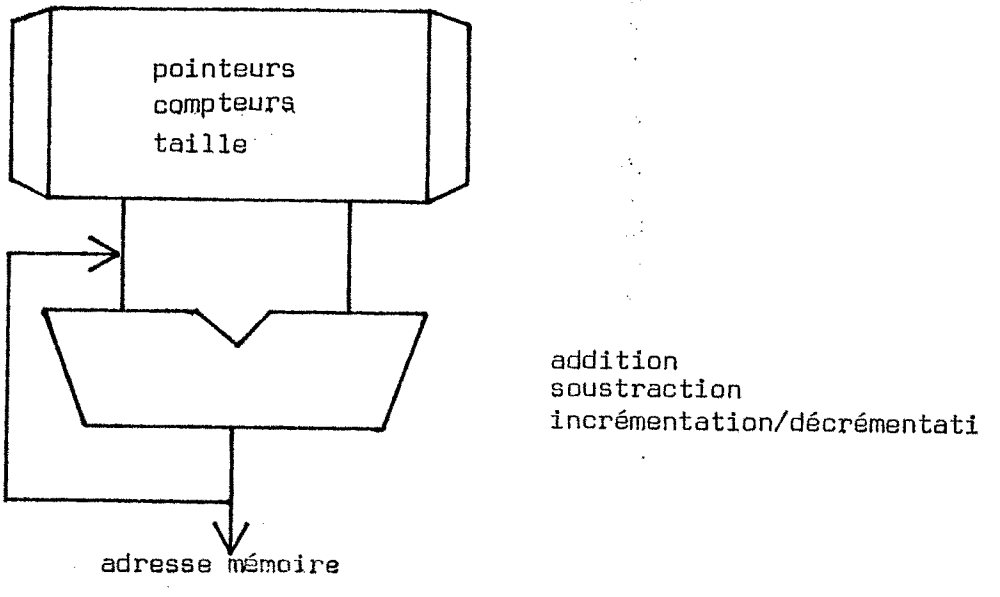


Fig. 8.2. - Schéma logique

Le dimensionnement des variables et des chemins de données dépend de l'application. Cependant, étant donné la facilité d'implantation offerte par l'utilisation de la bibliothèque de briques et la taille relativement importante de la mémoire tampon des moniteurs distribués (1/2K, 1K, 2K) on peut adapter la partie opérative à chaque application.

On aura donc 9, 10, 11 tranches de 1 bit selon la taille du moniteur.

Surface

Considérons l'exemple suivant :

FIFO - 1 Koctets - non réservable.

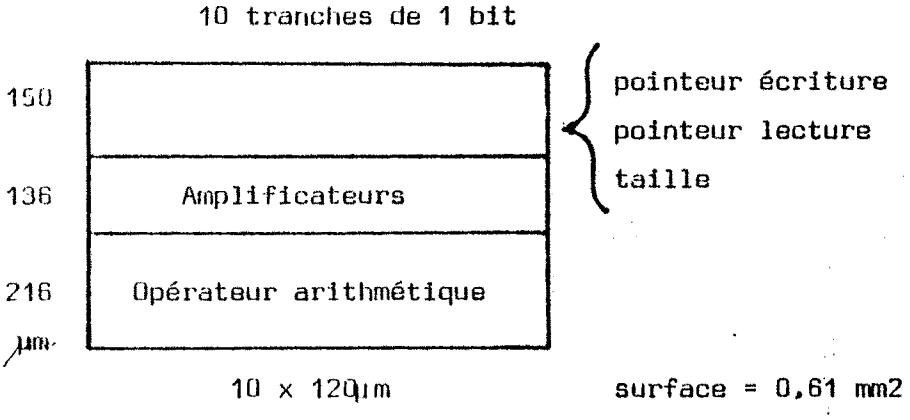


Fig. 8.3 - Implantation partie opérative

b/ partie contrôle

La partie contrôle doit assurer deux fonctions :

- exécuter l'algorithme des procédures,
- contrôler l'interface avec le module qui lui est couplé et l'interface avec le bus commun.

La structure de la partie contrôle de tous les automates doit être identique de façon à permettre la définition d'un langage commun de microprogrammation. Le remplissage du PLA, qui constitue le coeur de la partie contrôle, peut donc être obtenu par une compilation du programme de spécification dans ce langage de microprogrammation.

On va commencer par analyser les fonctions des deux interfaces :

Interface avec le bus

Fonctions :

a/ Détecter sur les lignes de données un appel à un moniteur :

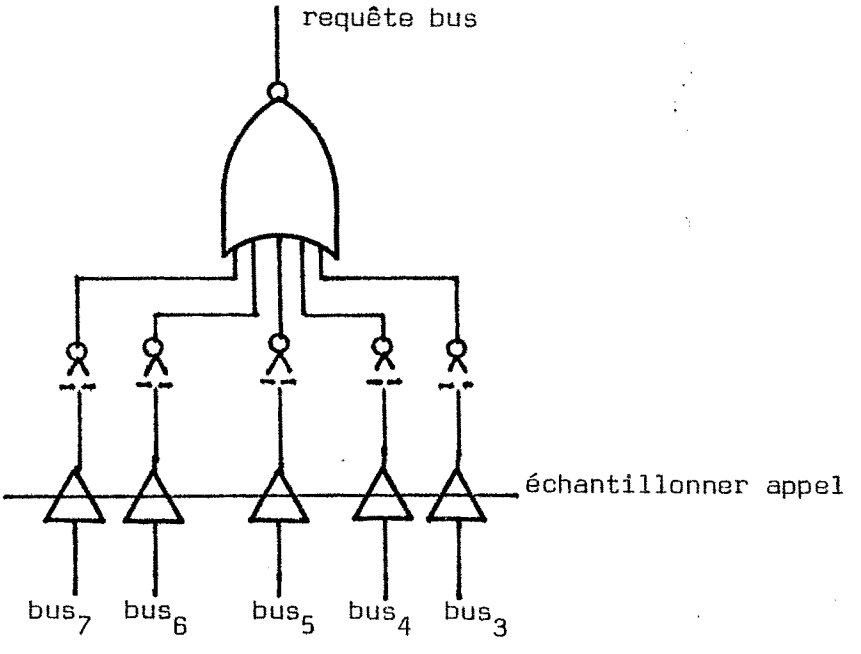


Fig.8.4 - Détection d'un appel

b/ Nom procédure :

La ligne "échantillonner appel" permet aussi d'échantillonner les trois bits qui codifient la procédure.

c/ Echantillonner les données en provenance du bus :

1) les processus qui accèdent au moniteur par le bus écrivent dans le moniteur -

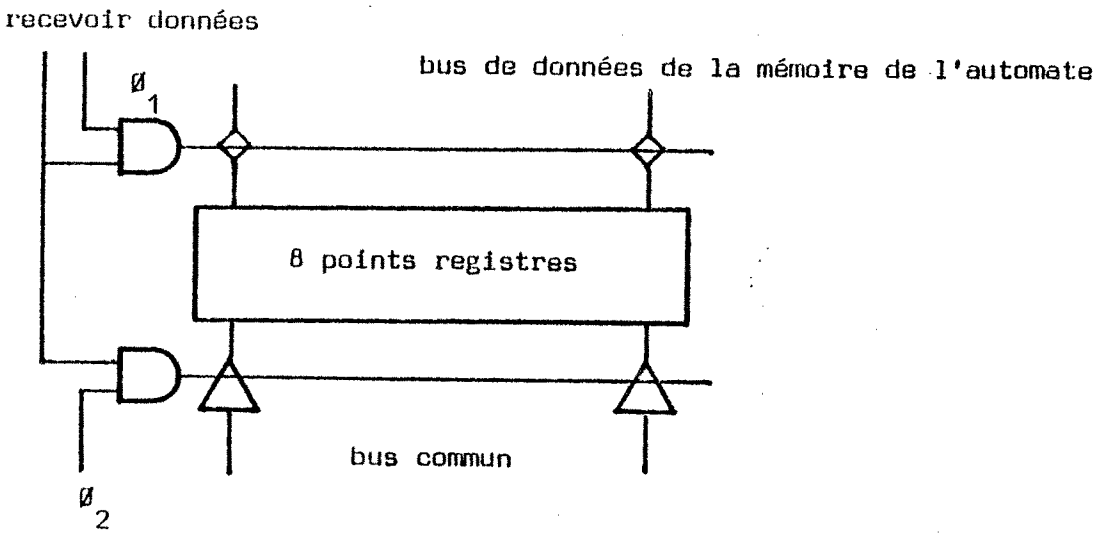


Fig. 8.5. - Réception données

2) les processus qui accèdent au moniteur par le bus lisent le moniteur -

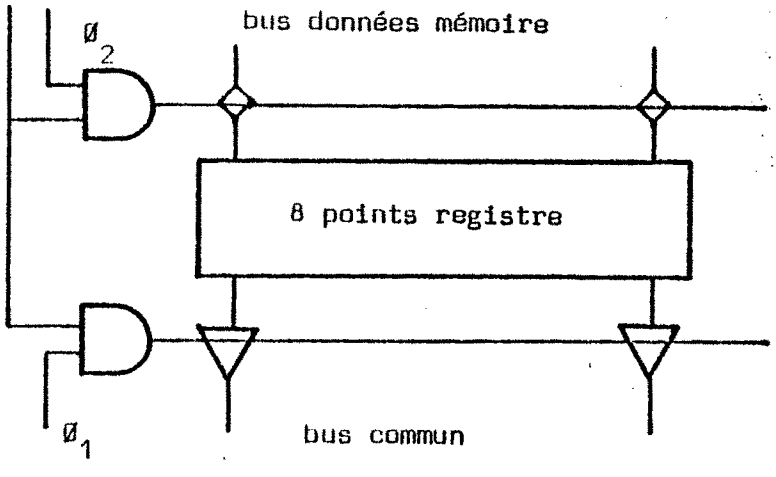


Fig. 8.6. - Envoi données

d/ Positionnement et test des lignes qui définissent l'état du bus :

lignes :

Occupé

Prêt

Attente

Continuer.

Interface avec le module

Fonctions :

a/ Signaler à la partie contrôle que le processus veut faire un accés :

ligne :

Requête - processus .

b/ Procédure à exécuter :

Une ligne permet de charger un registre avec le nom de la procédure envoyé par le module :

ligne :

Echantillonnage - module.

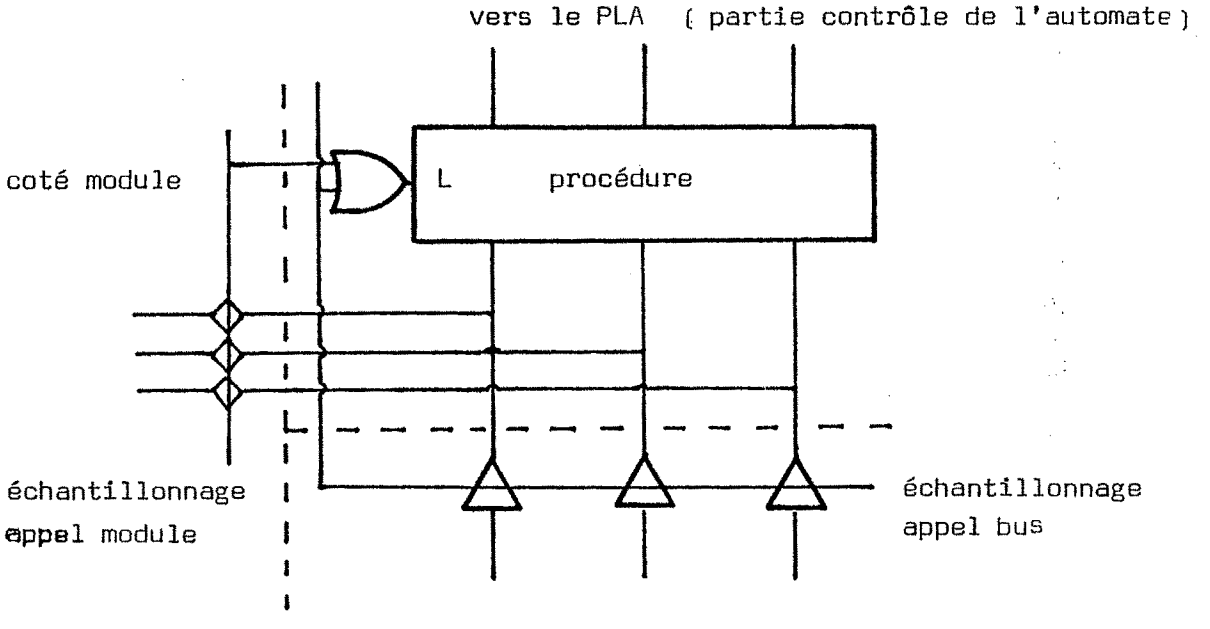


Fig. 8.7. - Nom procédure

c/ Mise en communication du bus de données de la mémoire et du bus de données du module :

ligne :

Connexion - bus.

Algorihtmes des procédures :

Le microprogramme mémorisé dans un PLA correspond à trois ou quatre séquences différentes et représentent les procédures exécutées par le moniteur.

L'information de séquencement provient de quatre sources différentes :

a/ une adresse codée dans la microinstruction : ce champ a quatre bits car les séquences de microprogramme n'ont pas plus de 16 microinstructions.

b/ un incrémenteur qui est utilisé pour compter le nombre d'accés aux données déja effectués.

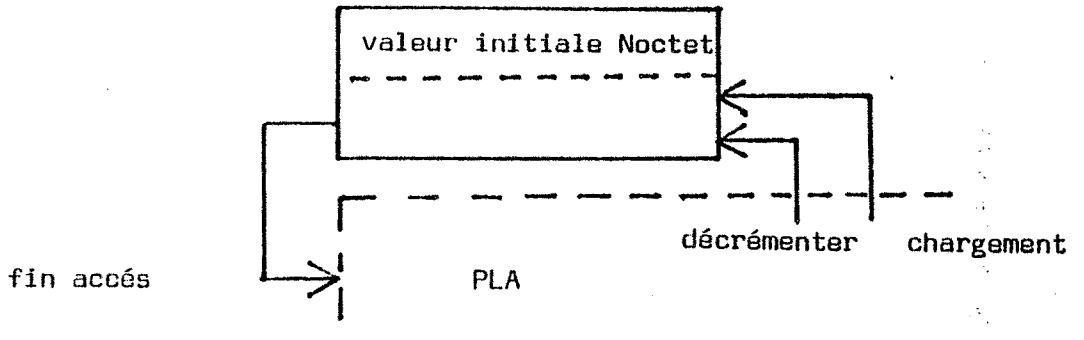


Fig. 8.8. - Comptage du nombre d'accés aux données.

Réaliser le comptage des accés par un organe matériel séparé permet d'utiliser la partie opérative pour incrémenter les adresses en parallèle et donc de diminuer le temps d'exécution des procédures.

c/ flags : Les flags sont utilisés pour mémoriser certaines informations sur l'état du moniteur. Le positionnement et le test des flags peut être fait directement par le microprogramme.
Etant donné que leur nombre est réduit on considère que chaque flag est mémorisé par une bascule T.

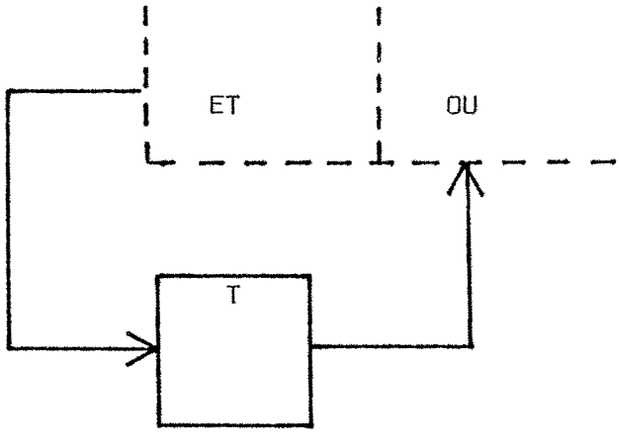


Fig. 8.9. - Flag

Normallement on aura de 0 à 4 flags par moniteur

d/ bits d'états de la partie opérative : Les bits d'état zéro et signe nous semblent suffisants.

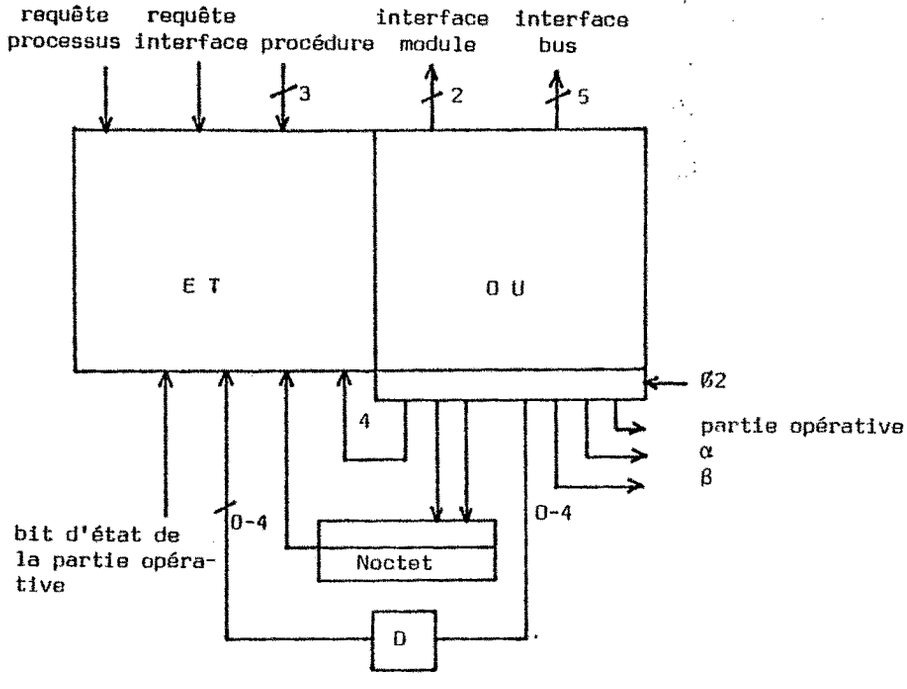


Fig. 8.10 - Partie contrôle de l'automate

Surface :

PLA - Entrées :

Interfaces	1	requête d'accés du processus couplé
	1	requête d'accés des processus extérieurs
	3	procédure
	1	bus-occupé
Informations d'état de l'automate	0 - 4	flags (0 - 4)
	2	bits d'états
	4	adresse microinstruction
	1	compteur
	13 - 17	

Sorties :

4	- commande partie opérative
2-3	- adresse α
2-3	- adresse β
2	- registre Noctet (charger, décrémenter)
0-4	- flag
2	- interface module (échantillonnage-module, connexion bus)
5	- interface bus (échantillonnage-appel, envoyer/recevoir données, prêt, attendre, continuer)
1	- écrire/lire mémoire
18 - 24	

Le nombre de microinstructions nécessaires, par exemple, pour implémenter un FIFO est de 23 microinstrctions :

$$\text{surface} = [13\times37+(15+\lceil\frac{15}{8}\rceil)\times20] \times [(23+\lceil\frac{23}{8}\rceil)\times20)] \ \mu m2$$

$$= 801 \times 500 = 0,40 mm2$$

Si on suppose que les interconnexions autour représenteront 50% de la surface du PLA, on a :

$$S_{contrôle} = 1,5 \times 0,40 = 0,6 \ mm2$$

Si on considère un FIFO avec une taille de 1 Koctets, la surface de l'automate est de :

mémoire dynamique	=	9192 x 420 µm2
	=	3,44 mm2
partie opérative	=	0,61 mm2
partie contrôle	=	0,60 mm2
		4,65 mm2

Dans une technologie 2 µm.

mémoire dynamique	=	0,86 mm2
partie opérative	=	0,157mm2
partie contrôle	=	0,155mm2
		1,172mm2

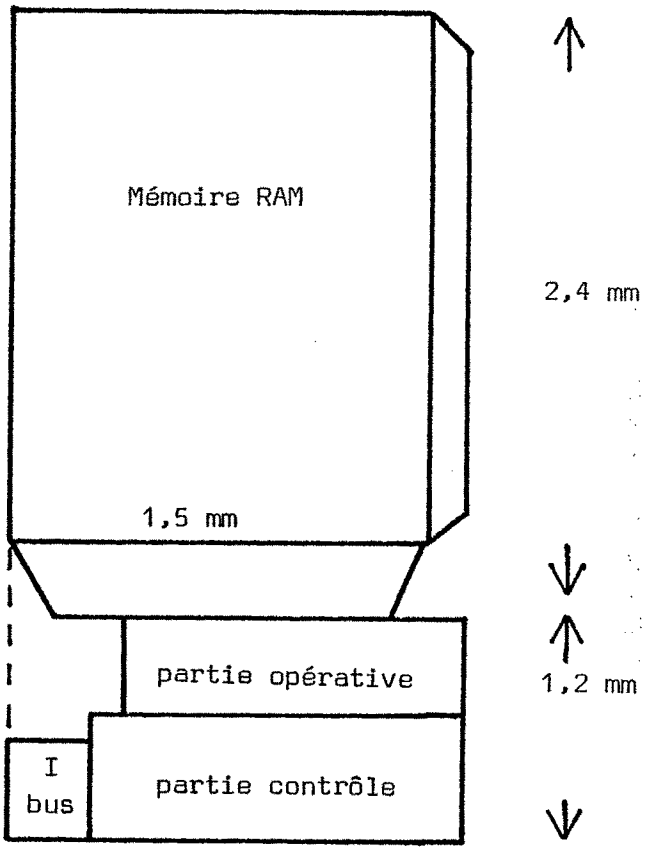


Fig. 8.11 - Implantation de l'automate

8.4. MODELE DE SIMULATION

Le schéma de simulation proposé dans 6.3 peut être facilement adapté pour décrire une architecture ayant des moniteurs implémentés localement.

Les modifications à introduire concernent les interfaces des processus (interface au sens modèle d'un processus 6.3 qui modélisent l'accés aux moniteurs externes).

Les deux nouvelles situations à modéliser sont :

a/ Processus couplé à l'automate :

Il n'y a pas de synchronisation avec le bus. Le processus ne peut être retardé que si l'automate est en train d'exécuter une procédure pour un processus implémenté dans un autre module.

L'automate est représenté par un sémaphore et l'interface du processus aura la forme :

Interface processus couplé {
P (Automate) ;
CST (Temps d'exécution de la procédure) ;
V (Automate) ;

b/ Processus implémenté dans d'autres modules, et utilisant l'automate :

Pour ces processus il y a une double synchronisation. D'abord ils doivent obtenir le bus et ensuite l'automate qui peut être en train d'exécuter une procédure pour le processus couplé.

L'interface devient donc :

```
P (Bus)
P (Automate)
CST ("temps d'exécution de la procédure)
V (Automate)
V (Bus)
```

Pour exemplifier ces modifications on peut reprendre l'exemple ANTIOPE en considérant que la mémoire de rafraîchissement est implémentée par un automate.

Processus de réception : identique

Processus d'interprétation : la partie de lecture du tampon de réception est la même.

La mémoire de rafraîchissement est représentée par un sémaphore (MRAF).

L'interface qui écrit les caractères dans la mémoire de rafraîchissement est donc :

```
P (Bus) ;
P (MRAF) ;
CST ("temps d'exécution procédure") ;
V (MRAF) ;
V (Bus) ;
```

Processus de visualisation :

L'interface devient :

```
P (MRAF) ;
CST ("temps d'exécution de la procédure") ;
V (MRAF) ;
```

Les résultats et les programmes sont décrits dans l'annexe 4.

On peut remarquer que le processus de visualisation suit la fréquence jusqu'à un temps de cycle de 150 ns.

Le processus de réception à 100ns arrive très difficilement à suivre la fréquence de réception tandis que dans la simulation sans le moniteur implémenté localement (6.3) la fréquence limite était atteinte à 75ns.

CHAPITRE 9

M O D U L E S

Dans le système MOSAIC un module correspond à une généralisation de la notion classique de processeur fonctionnel, c'est-à-dire un organe doté d'une certaine "intelligence" qui lui permet d'exécuter ses fonctions spécifiques et de communiquer avec les autres éléments du système.

A l'intérieur d'un module, un ou plusieurs processus peuvent être actifs. Aucune hypothèse n'a été faite sur leur implémentation, ils peuvent être exécutés par une seule partie opérative et une seule partie contrôle, par plusieurs ou sous forme d'un montage électronique particulier.

Cependant un module doit être toujours composé des sections suivantes :

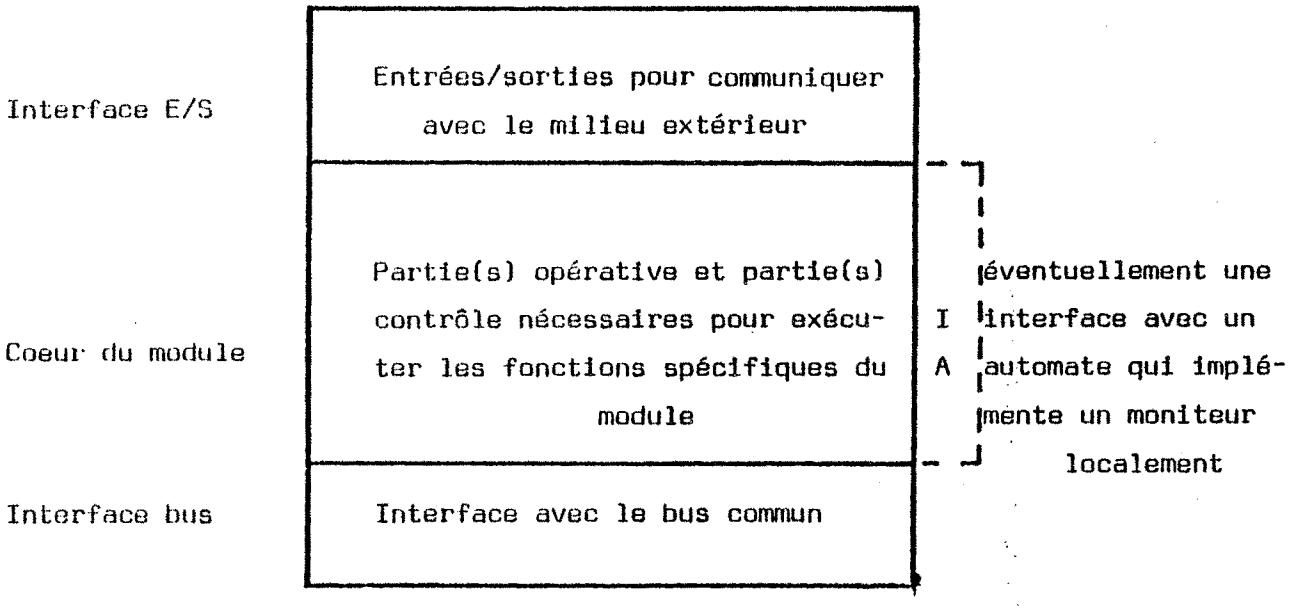


Fig. 9.1. - Sections d'un module

La structure matérielle de la partie centrale du module n'est pas traitée dans ce travail. Elle correspond à la conception et à l'implantation d'un processeur fonctionnel classique, qui doit cependant obéir à certaines contraintes topologiques imposées par les deux autres sections.

Dans ce chapitre, on analysera l'implémentation des deux interfaces mais on ne précisera que les aspects macrocospiques relatifs au coeur du module sans proposer une implémentation.

Cependant, pour pouvoir évaluer la surface des circuits implémentant certaines applications étudiées, on présente dans l'annexe 5 quelques exemples de modules qui reprennent la structure matérielle de circuits commerciaux existant.

9.1. COEUR DES MODULES

Dans le coeur du module sont implémentés sur les partie(s) opérative et partie(s) contrôle nécessaires, les processus qui caractérisent le module dans le système.

Les modules peuvent être divisés en deux classes qui correspondent à des implémentations différentes :

Modules dédiés - Module exécutant une fonction très spécifique avec peu de possibilités de paramétrisation, ex.: contrôleur E/S, circuit de calcul mathématique, circuit de traitement de signal, etc...

L'architecture interne de ce type de module est assez semblable à celle des processeurs périphériques dévelopés pour des systèmes microprocesseurs.

La structure matérielle de ces modules est spécialisée pour l'exécution d'une fonction et donc il ne s'agit que d'un processeur banalisé adaptable par programmation.

Modules programmables - Modules constitués par des processeurs d'usage général dotés d'une mémoire programme et d'une mémoire de travail.

Les fonctions exécutées par ce type de module sont très semblables à celles des microordinateurs existants (ex.: I8048, M6801). Les modules programmables seront utilisés pour implémenter des tâches à caractère logiciel comme par exemple l'interprétation d'un langage de commandes, l'édition de texte, etc...

Il est évident que le principal avantage des modules programmables est la facilité avec laquelle ils peuvent être adaptés à une application mais leur surface est nettement plus grande que celle des modules dédiés.

Des méthodes de conception de parties opératives du type de celui décrit dans 5.2.2. [SUZ.78]et de parties contrôle [ANC.79] peuvent être utilisées pour concevoir des modules programmables ayant une structure de base permettant d'optimiser leur temps d'exécution et leur surface.

Pour les modules dédiés il est cependant beaucoup plus difficile de proposer des méthodes de conception rapides et performantes. Des approches du type "building block"[GAZ.77] peuvent économiser du temps mais la surface ne sera pas optimisée.

9.2. INTERFACE BUS

Dans la définition de l'interface bus apparaît à nouveau le problème de la définition des zones isochrones à l'intérieur du circuit.

La méthode proposée au chapître 4, de création d'une hiérarchie de zones isochrones correspond à fournir à chaque module et au processeur noyau, une horloge rapide et à synchroniser les actions sur le bus par une horloge à fréquence plus basse correspondant à une zone isochrone plus large.

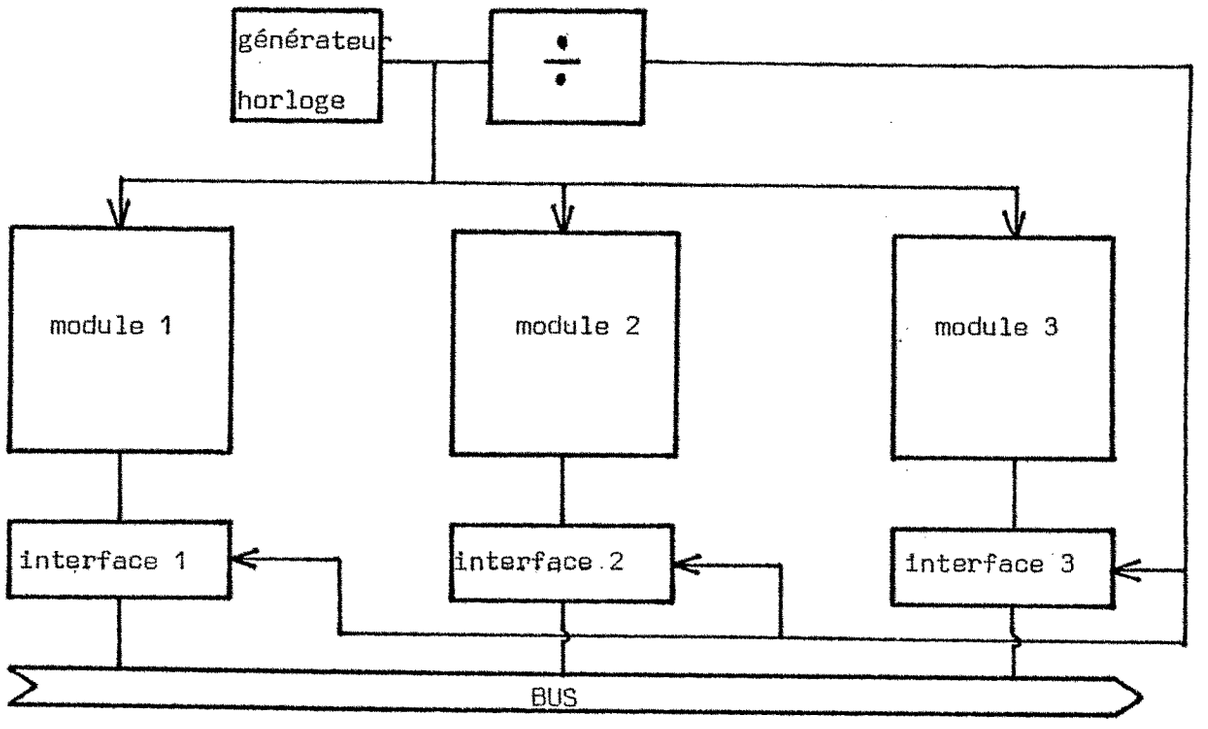


Fig. 9.2. - Distribution de l'horloge

Comme les retards introduits par les lignes de distribution de l'horloge sont différents, les horloges internes des modules ne seront pas en phase avec l'horloge bus.

Les interfaces ont donc un fonctionnement asynchrone par rapport au coeur du module.

Etant donné que l'horloge interne des modules est beaucoup plus rapide que celle du bus, on peut utiliser un dialogue très simple (module, interface) avec une bascule qui indique si le tampon de l'interface est vide ou plein.

Cette solution peut provoquer des problèmes de métastabilité [MEA.79] mais il serait possible de les résoudre en faisant varier la phase de l'horloge interne des modules de façon à éviter la zone critique où les phénomènes de métastabilité peuvent apparaître.

Dans la description de l'interface qui suit l'horloge utilisée est donc implicitement l'horloge bus.

Remarques: - Comme il a déjà été mentionné, les retards dans la distribution de l'horloge sont encore négligeables pour des technologies à 2µm. L'horloge utilisée dans cette technologie serait donc la même.

- Pour simplifier l'exposition on continue à considérer un seul processus actif dans le module. A la fin de cette section, on analysera les problèmes posés par l'existence de plusieurs processus.

Les actions que doit exécuter l'interface sont :

- Envoyer l'appel :
  - . allouer bus
  - . interpréter les signaux de contrôle de l'exécution : - Prêt
    - Attendre
    - Continuer.
- Envoyer/Recevoir des données,
- Détecter un "signal" pour la condition où le processus est bloqué.

a/ Envoi de l'appel

Le processus doit envoyer à l'interface une requête de "début-appel". Celle-ci essaiera ensuite d'allouer le bus et de détecter si l'appel est exécutable.

Le schéma logique décrit dans la Fig. 9.3 permet de détecter si le bus est libre et de l'allouer suivant le schéma de priorité fixe.

La ligne "Envoyer appel" signale au processeur que l'appel (nom moniteur, procédure) doit être envoyé dans le cycle suivante (séquencement bus, Fig. 4.7).

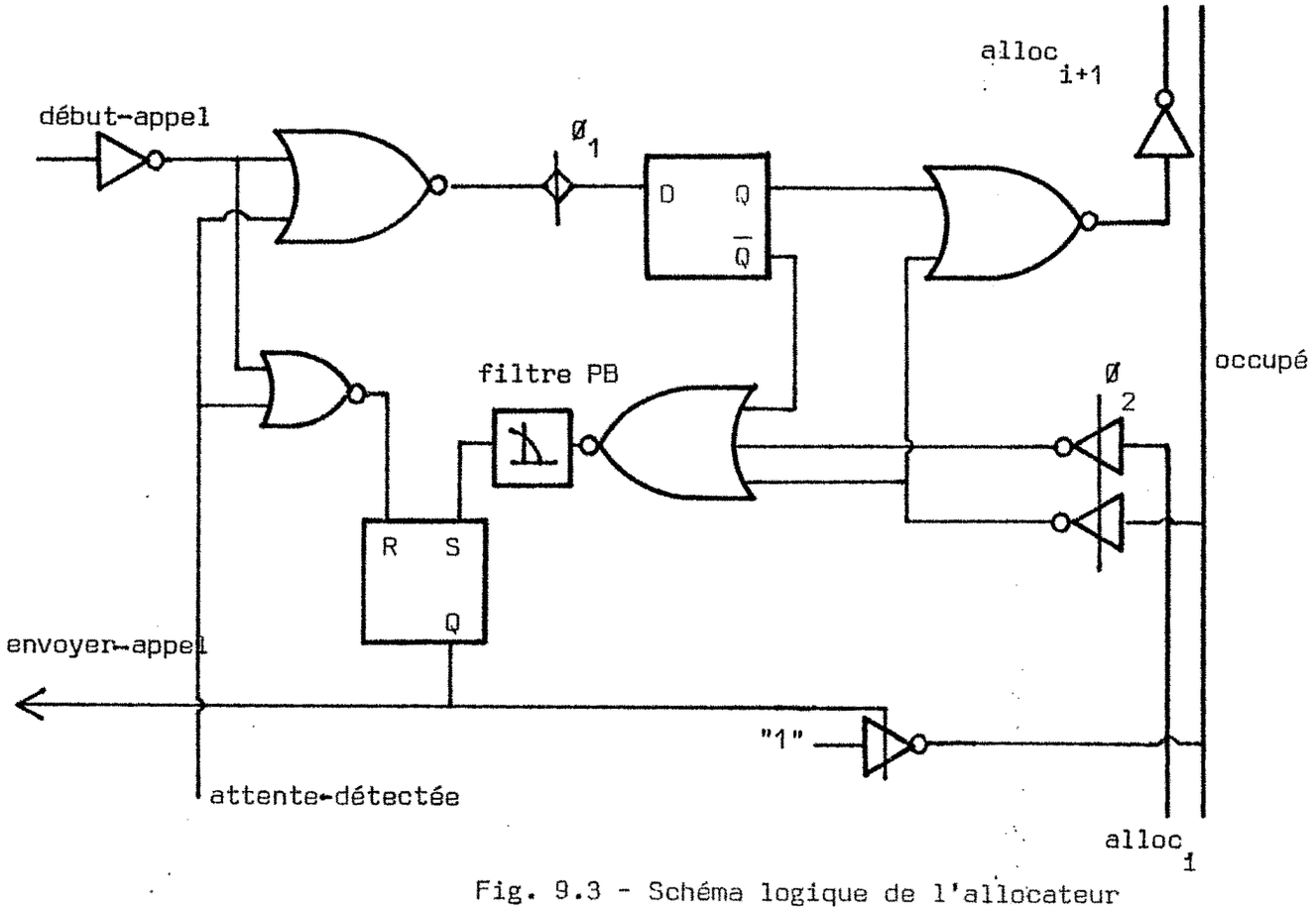


Fig. 9.3 - Schéma logique de l'allocateur

La ligne Attente-détectée sera positionnée par la section de l'interface qui traite les primitives de synchronisation. Son activation oblige à libérer le bus permettant à un autre processus d'envoyer son appel.

Dans l'hypothèse d'une méthode d'allocation avec des priorités fixes (4.1) le temps de propagation de la chaîne d'allocation peut devenir critique.

Dans le pire des cas, le signal d'impossibilité d'allocation (niveau haut) doit se propager par toute la chaîne de portes NOR et inverseur dans tcycle/2 ($\sim$ 50ns).

Le filtre à implémenter par un monostable permet d'éliminer les impulsions de durée trop courte de façon à éviter de possibles aléas de synchronisation.

Multiplexage temporel

Dans le multiplexage temporel il faut fournir à l'interface un signal qui indique quelle est la tranche de temps allouée au processus.

Exemple :

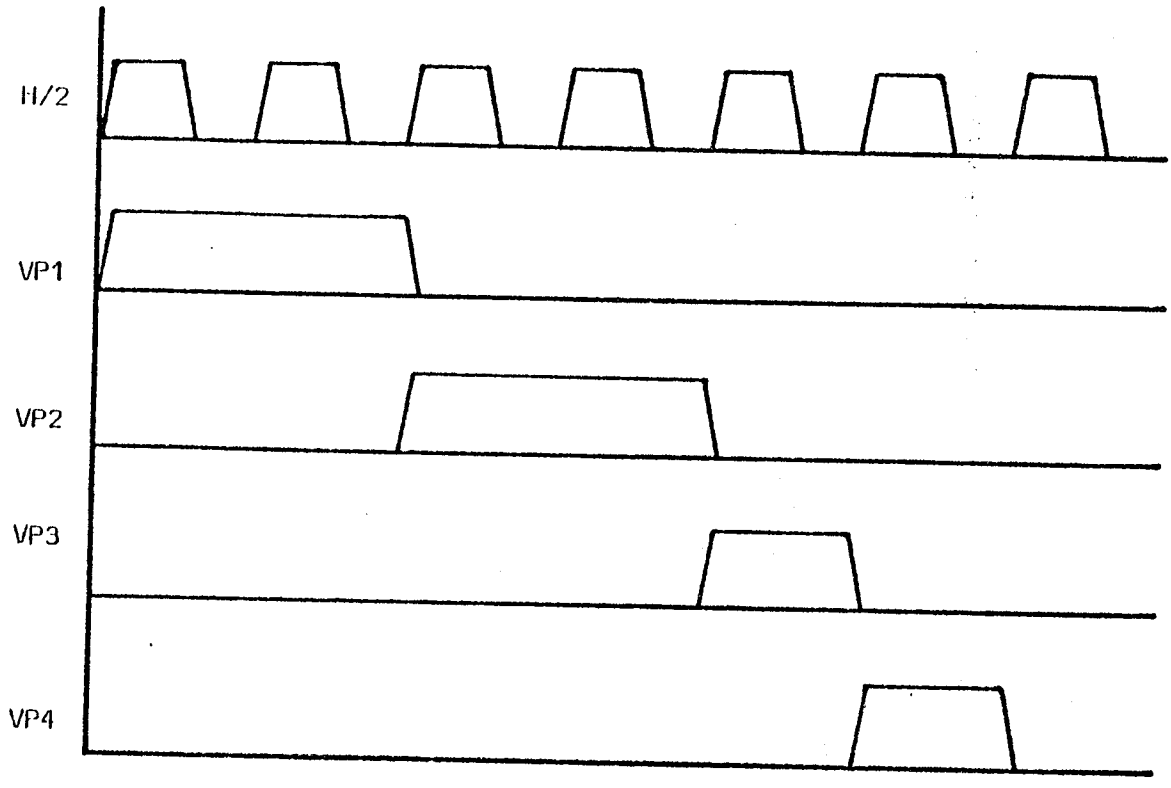


Fig. 9.4 - Séquencement dans le multiplexage temporel.

Ces informations temporelles peuvent être facilement obtenues par deux monostables qui peuvent être réalisés par des compteurs.

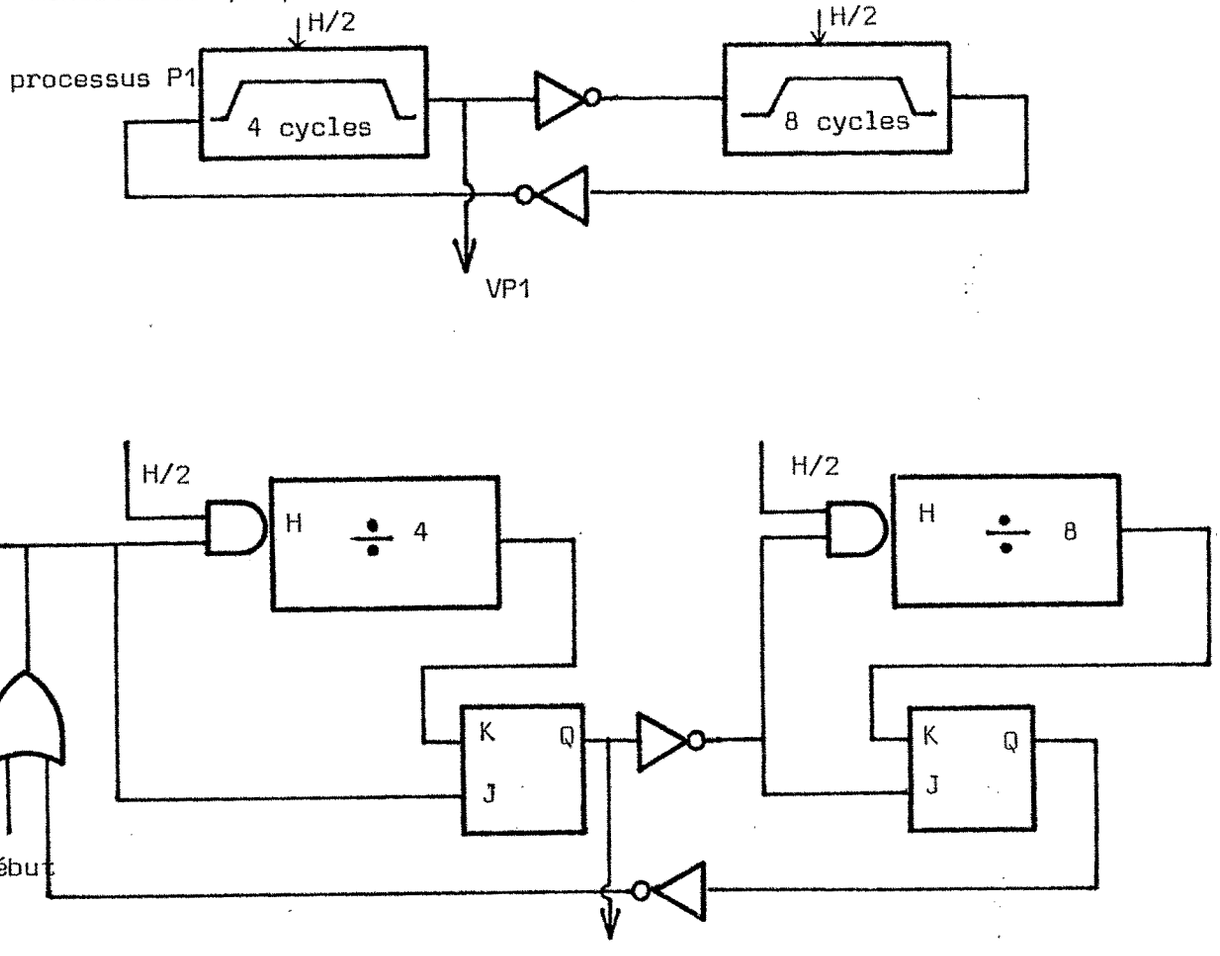


Fig. 9.5 Gé nération du séquencement des interfaces.

b/ Envoi/réception de données

Après l'allocation du bus, le processeur doit envoyer l'appel au moniteur externe.

Le schéma logique de la Fig. 9.6 permet de mémoriser l'état "Appel envoyé"

et de détecter ensuite s'il a été ou non accepté par le processeur noyau ou l'automate qui l'exécute.

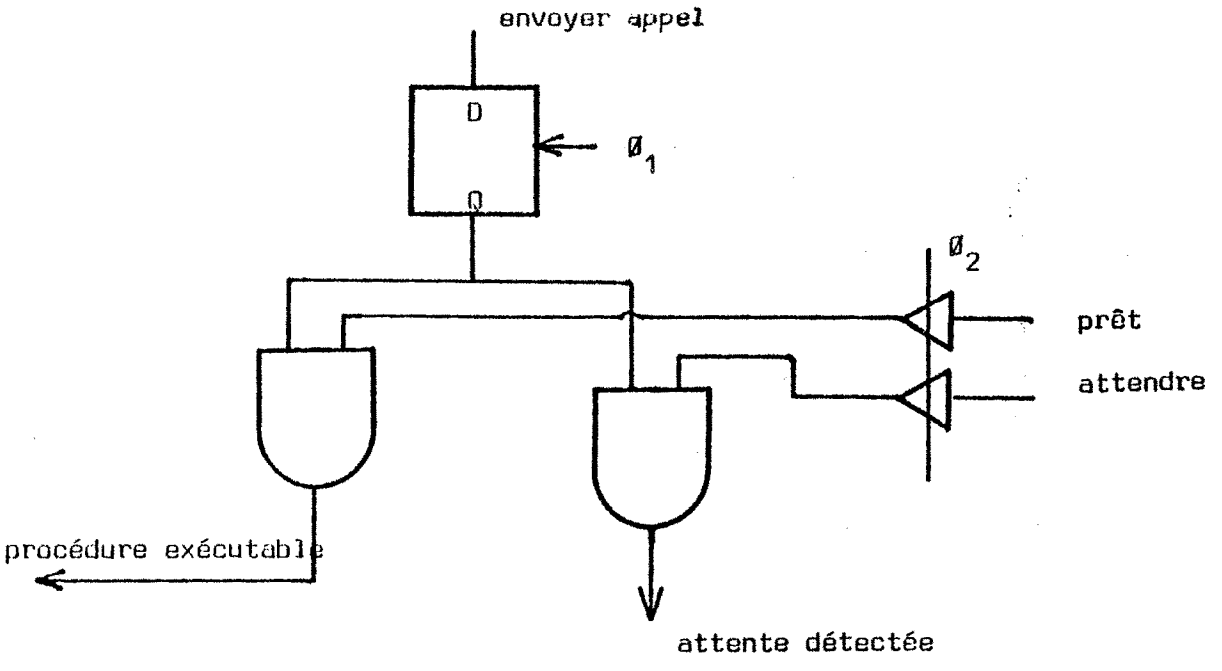


Fig. 9.8 - Reponse du processeur noyau.

L'interface a un registre tampon qui lui permet de mémoriser, pendant un demi-cycle, l'octet reçu ou à transmettre.

La ligne Emission/Réception permet au processus de contrôler le sens du transfert.

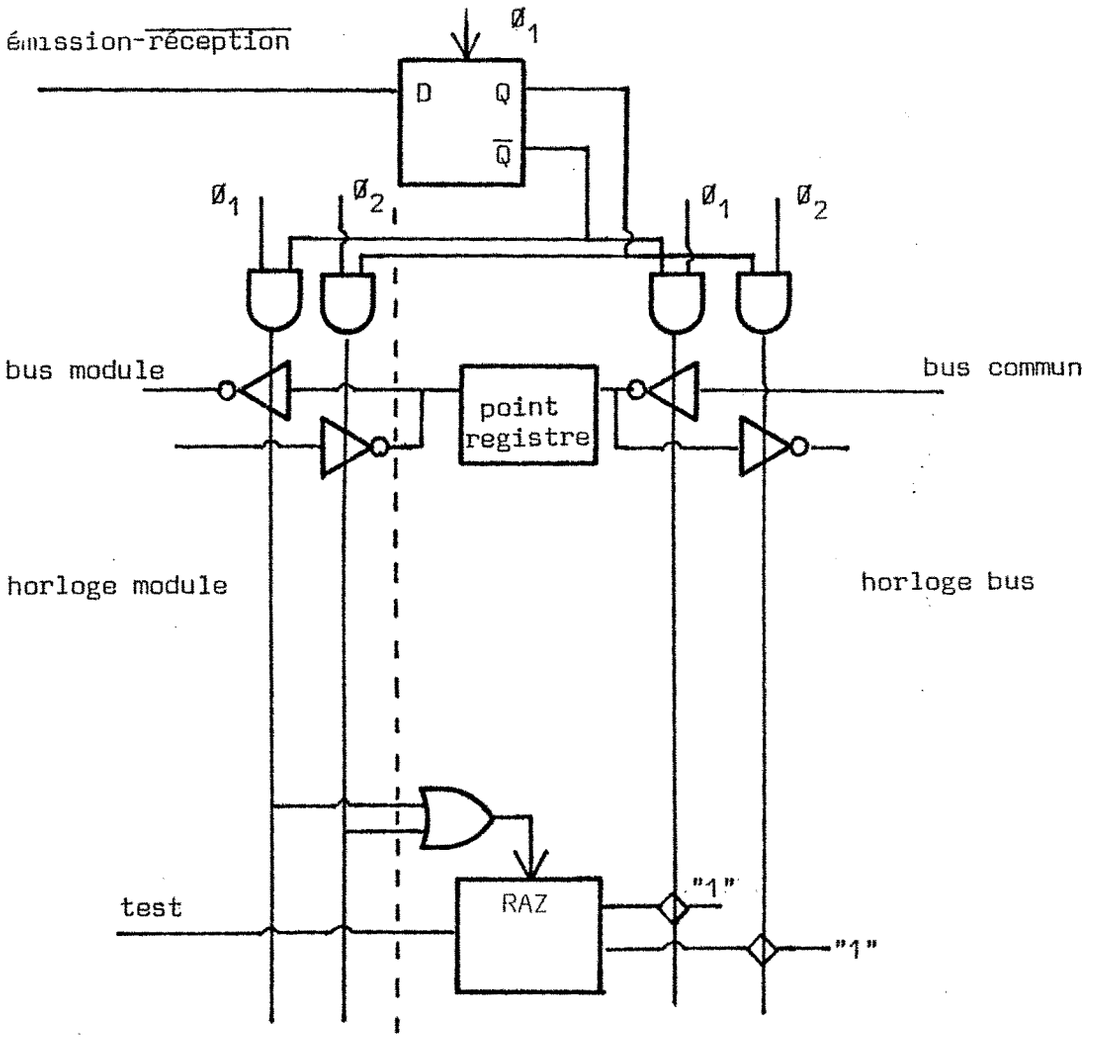


Fig. 9.7 - Registre de données de l'interface.

Pour envoyer l'octet qui constitue l'appel à un moniteur et ensuite recevoir ou émettre les données, la partie contrôle du module doit positionner les lignes de contrôle de l'interface de la façon suivante :

Envoi appel

Requête - Appel = 1
Réception/$\overline{\text{Emission}}$ = 0
Bus = Appel

Réception données

cycle 1	cycle 2
Requête - Appel = 1	Requête - Appel = 1
Réception/$\overline{\text{Emission}}$ = 1	Requête/$\overline{\text{Emission}}$ = 1
Bus = 0	Bus := Donnée 1

Emission données

cycle 1	cycle 2
Requête - Appel = 1	Requête
Réception/$\overline{\text{Emission}}$	Réception/$\overline{\text{Emission}}$ = 1
Bus := Donnée 1	Bus := Donnée 2

Le séquencement requis pour le transfert de plusieurs octets doit aussi être assuré par les structures matérielles qui implémentent le processus (lire/écrire un caractère toutes les 100ns). Si les caractéristiques internes du module rendaient cette opération impossible, il faudrait prévoir,dans sa conception, un FIFO permettant de rendre indépendant le séquencement interne du module et celui du bus.

Remarque : Pour les modules programmables, on peut inclure dans leur code opération des instructions comme :

- Envoyer (caractère) - envoi de l'appel ou d'un seul carcatère.
- Envoyer (adresseI, Noctet) - envoi d'un ensemble de Noctet mémorisé à partir de adresseI
- Recevoir (adresseI, Noctet).

L'interpréteur du langage machine de ces modules doit ensuite se charger de générer les signaux nécessaires pour contrôler l'interface.

c/ Opérations de synchronisation

Les deux sections de l'interface qui viennent d'être décrites ne concernent que les appels exécutables. La troisième partie va être chargée de la synchronisation explicite dans l'exécution d'un moniteur externe (4.4).

Un registre "condition" permet de mémoriser le nom de la condition où le processus a été logiquement bloqué. Le chargement de ce registre est déclenché par l'activation de la ligne "Attente détectée". Après que le registre ait été chargé on compare la valeur mémorisée avec celle du bus de données à chaque excitation de la ligne "continuer".

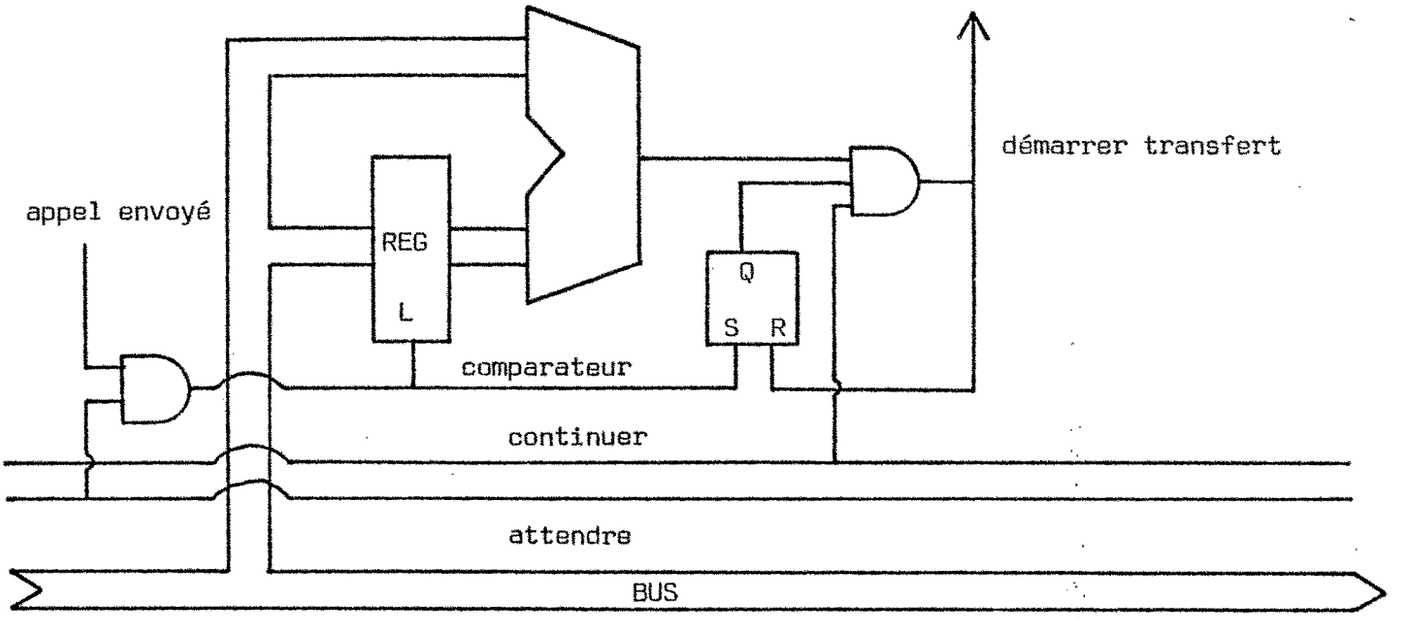


Fig. 9.8 - Détection du "signal"

Le schéma électrique correspondant est considérablement plus simple.

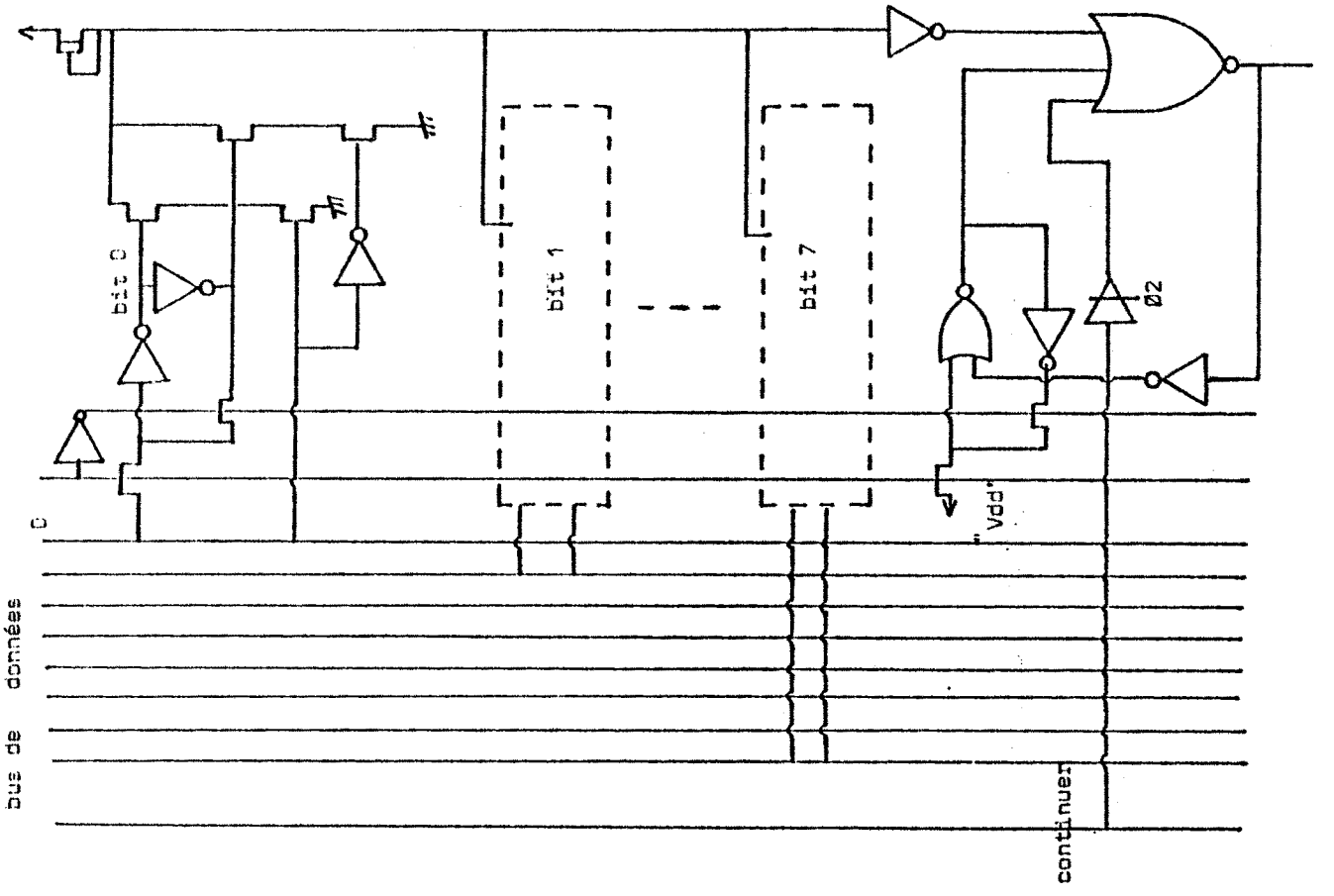


Fig. 9.9 - Détection "signal" - Schéma électrique

Surface

La logique anarchique qui constitue l'interface ne permet pas de faire une évaluation très précise de sa surface.

Pour avoir un ordre de grandeur on a compté les transistors de tous les éléments de l'interface et on a divisé le résultat par la densité moyenne de la logique anarchique en 6µm. La valeur de densité utilisée était 700 MOS/ m2 (annexe 3).

Partie allocation	30
Envoi/réception donnée	50
Détection exécutable	15
Synchronisation externe	60
	155

155/700 = 0,22 mm2

A cette surface, il faut rajouter les deux registres utilisés pour les données et pour la condition :

17 points registres x 400 = 0,068mm2
surface interface $\underline{\sim}$ 0,22 + 0,068 $\underline{\sim}$ 0,29mm2.

Plusieurs processus dans le module

Dans la définition classique des moniteurs un processus bloqué dans une condition à l'intérieur d'un moniteur, cesse son activité jusqu'à l'arrivée d'un signal sur cette condition.

L'algorithme exécuté par l'interface correspond à ce schéma car le processus après avoir envoyé l'appel sera en attente de l'activation de la ligne "Procédure-exécutable" et donc pour la partie contrôle du processus un "wait" correspond à une attente au milieu d'une instruction d'appel à un moniteur externe.

Cette implémentation peut être utilisée s'il n'existe qu'un seul processus à l'intérieur du module.

Cette restriction initiale qui avait été prise pour simplifier l'exposition doit donc être levée de façon à ce que l'on puisse traiter des situations plus générales où plusieurs processus sont implémentés dans le coeur du module.

On va étudier trois cas qui nous semblent les plus représentatifs :

a/ **Le coeur du module est composé d'un processeur multiplexé entre plusieurs processus** - Après qu'un processus ait été bloqué dans un moniteur externe les autres processus doivent continuer à s'exécuter mais ils n'accèdent pas aux moniteurs externes.

Exemple :

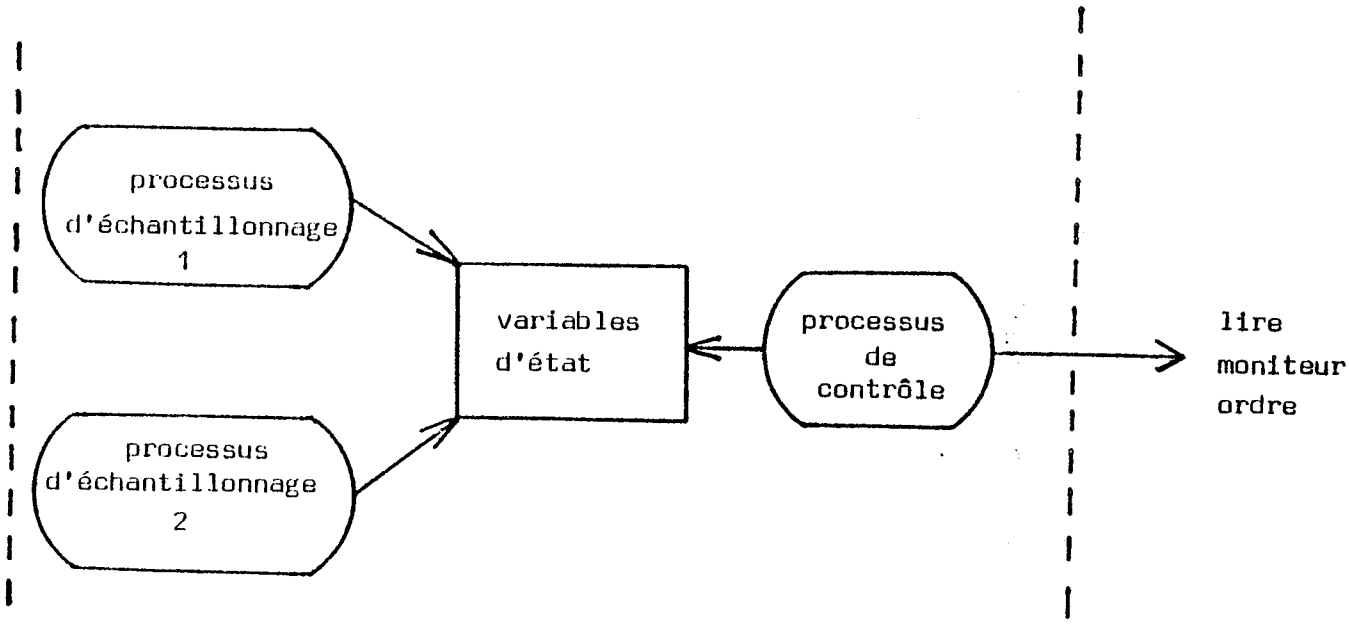


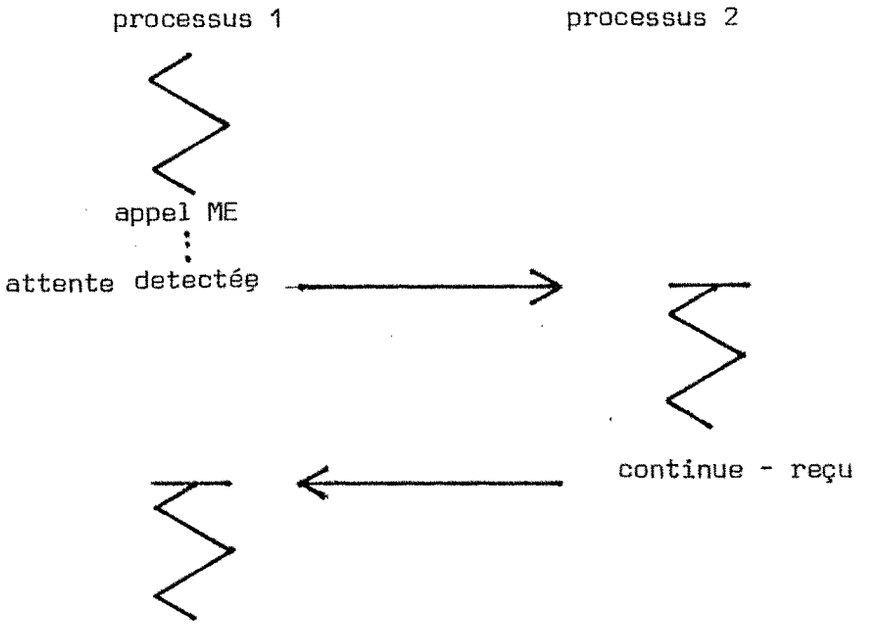
Fig. 9.10 -

Les processus qui gèrent les entrées temps réel ne peuvent pas être arrêtés pendant une période indéterminée quand le processus de contrôle est bloqué à l'intérieur du moniteur.

Une solution très simple est d'envoyer vers le contrôle du processeur les lignes d' "Attente-détectée" et "Continuer-reçu" qui doivent être prises en compte par l'algorithme de contrôle de la façon suivante :

1 - après avoir déclenché l'envoi de l'appel, la ligne d' "Attente détectée" doit être testée. Si elle est positionnée, le processus appelant est endormi et le contrôle est transféré à un processus en attente d'exécution.

2 - la ligne "Continuer-reçu" aura la même fonction qu'une interruption dans un processeur normal. Elle permettra de changer le contexte d'exécution du processeur de façon à ce que le processus en attente puisse continuer son exécution.



b/ Le coeur du module est constitué par un processeur multiplexé mais tous les processus peuvent avoir besoin d'accéder aux moniteurs externes :

Ce cas est plus difficile à résoudre car il conduit à une augmentation du matériel de l'interface.

Deux solutions pouvaient être utilisées:

1 - une interface par processus.

Avantages :

. aucun problème au niveau logique, les processus sont indépendants vis-à-vis du mécanisme de communication.

. l'interface est la même pour tous les processus .

Désavantages :

. la surface occupée par 4 ou 5 interfaces peut devenir importante.

. le contrôle du processeur doit être modifié de façon à avoir la possibilité de choisir l'interface à utiliser. La partie contrôle doit aussi gérer les différentes lignes "Continue-reçu" qui constituent une sorte de vecteur d'interruption pour le processeur.

2 - une interface unique.

Dans ce cas, il faut modifier la partie de détection des "signaux" de façon à pouvoir mémoriser les différentes conditions où il y a des processus en attente.

Une solution est l'utilisation d'une mémoire associative capable de reconnaître en parallèle si la condition envoyée concerne un des processus bloqués dans le module. La mémoire associative génère un vecteur d'interruption avec un bit par processus qui permettra à la partie contrôle de redémarrer le plus prioritaire (s'il y en a plusieurs en attente de la même condition).

La deuxième solution est du point de vue surface la plus performante, cependant elle est moins souple car elle oblige à que tous les processus du module aient la même priorité dans la chaîne d'allocation.

c/ Dans le coeur du module il existe plusieurs points d'exécution :

Chacun pouvant être multiplexé parmi plusieurs processus.

Le coeur du module est constitué par plusieurs points d'exécution et on retrouve les deux situations décrites précédemment multipliées par le nombre de processeurs existant.

Il est évident que ce cas correspond à une maximisation des fonctions du module et on peut considérer qu'il sera assez rare d'avoir des modules composés par plusieurs processeurs travaillent en parallèle.

Les deux solutions possibles sont :

1 - une interface par point d'exécution.

2 - multiplexer l'utilisation d'une interface parmi quelques uns des points d'exécution. Cette solution est fortement limitée par le débit des processeurs et par la complexité de l'organe d'allocation de l'interface. Elle ne peut donc être utilisée que pour des cas très simples (deux ou trois processeurs partageant une interface).

Un exemple qu'illustre la complémentarité des deux solutions précédentes est celui d'un processeur de téléinformatique pouvant travailler en full-duplex (Fig. 9.11).

Le schéma global de fonctionnement peut être le suivant :

1 - le processus de contrôle utilise l'interface 1 pour lire le moniteur où sont déposées les commandes ;

2 - il est bloqué jusqu'au moment où un processus du système y écrit une commande. Il doit ensuite paramétrer les processus de réception et/ou d'émission.

Le(s) processus de transmission travaillent ensuite en utilisant leurs interfaces respectives.

Les exceptions au déroulement normal des opérations de transmission ou la fin d'une opération sont signalées au processus de contrôle. Celui-ci soit écrit dans un moniteur le résultat de l'opération soit lit une nouvelle commande.

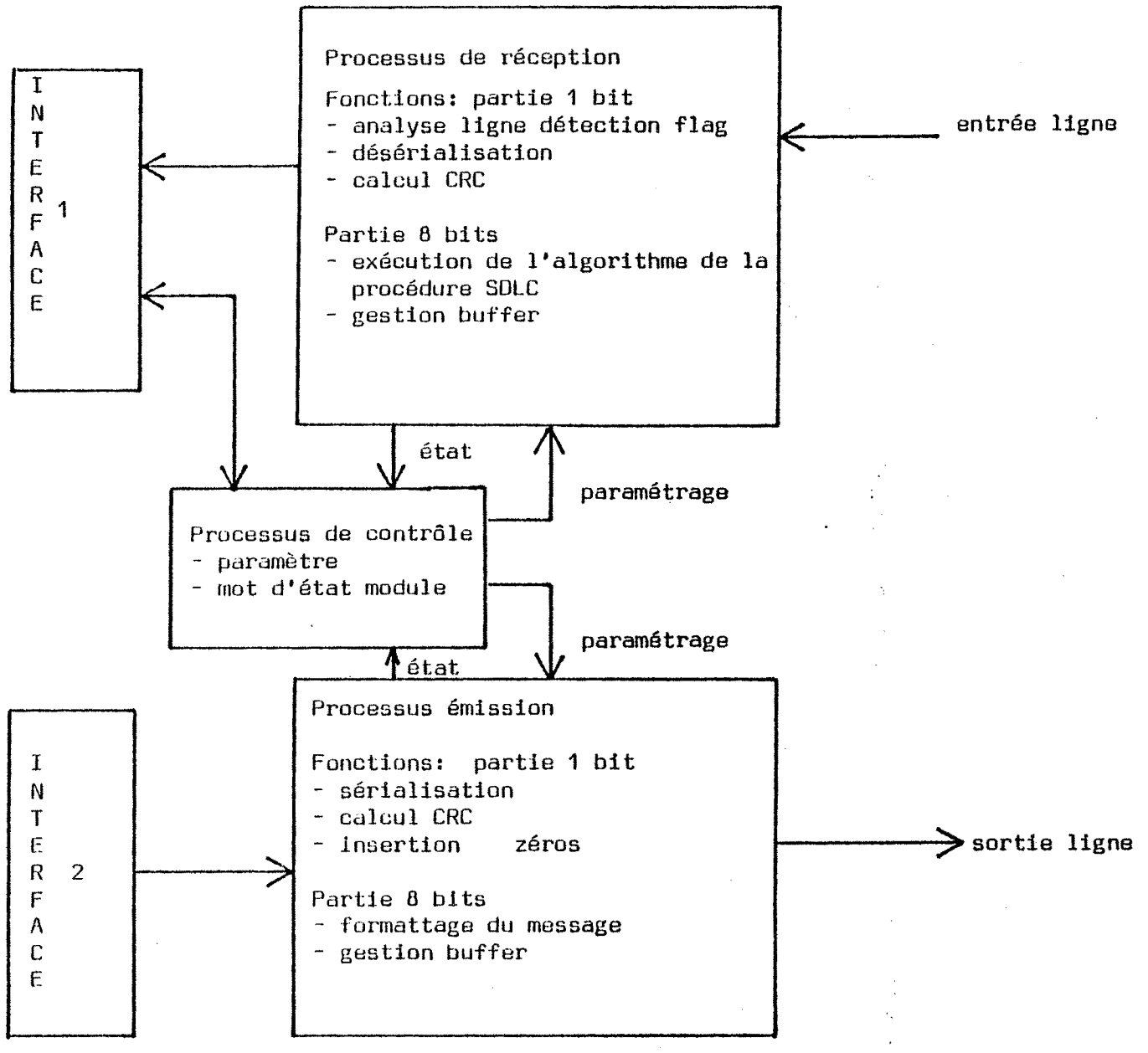


Fig. 9.11 - Interface full-duplex.

9.3. ENTREES/SORTIES

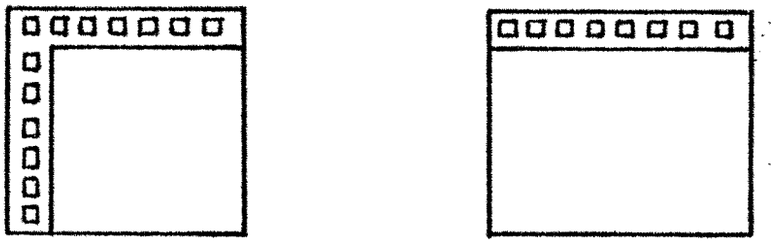
Les entrées/sorties posent deux problèmes importants. Le premier au niveau global du circuit, concerne le nombre debroches total et le deuxième au niveau des modules, concerne la disposition des plots et des amplifications d'E/S.

Le premier problème dépend de la technologie de fabrication des boîtiers

et sera traité dans les conclusions globales, on n'analysera pour le moment que le deuxième.

L'idée de base de maintenir les dessins des modules dans une bibliothèque empêche de disposer les plots d'E/S autour du module d'une façon fixe.

Exemple :



Module utilisé dans un coin Module utilisé au milieu

Fig. 9.12

Pour ne pas imposer une contrainte topologique supplémentaire (modules "coin", modules "milieu") et ne pas pénaliser la surface du module par le nombre des plots d'E/S, on a préféré ne pas les lier au dessin du module.

Une bande d'interconnexions composée uniquement de lignes d'aluminium qui seront reliées aux plots est définie dans le côté du module opposé à l'interface bus.

Les amplificateurs et plots d'E/S sont disposés sur la périphérie du circuit en dehors des modules. Un programme de routage permet ensuite de connecter les modules aux amplificateurs et aux plots de façon à minimiser la surface occupée par la périphérie du circuit.

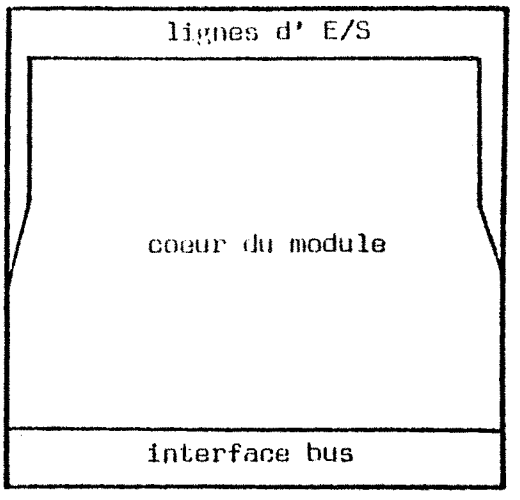


Fig. 9.13 - Topologie des modules

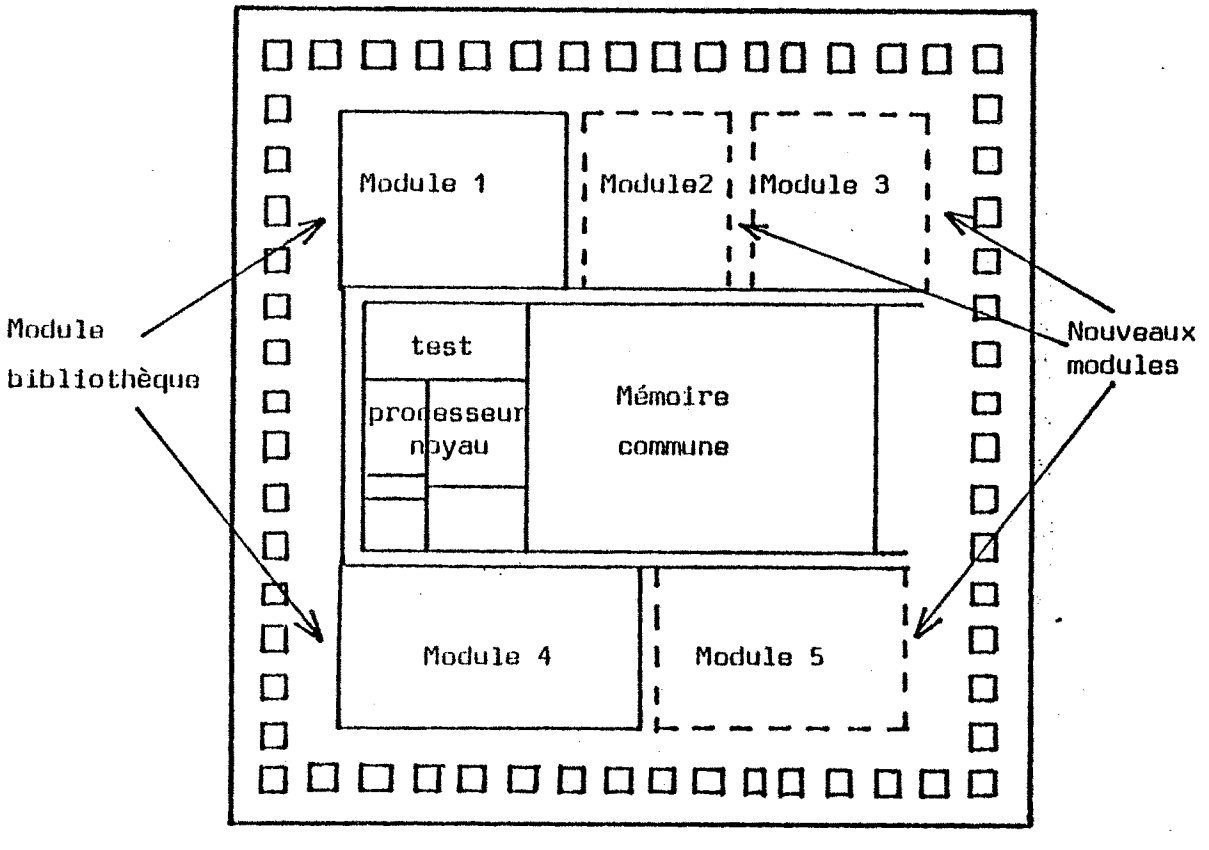


Fig. 9.14 - Schema de l'implantation global du circuit

9.4. TOPOLOGIE DES MODULES

Pour que l'utilisation de la bibliothèque de modules soit commode, il faut que les modules aient des tailles compatibles.

Nous avons considéré que les modules seraient dessinés de façon à maintenir fixe la dimension Y (bande de connexion - interface).

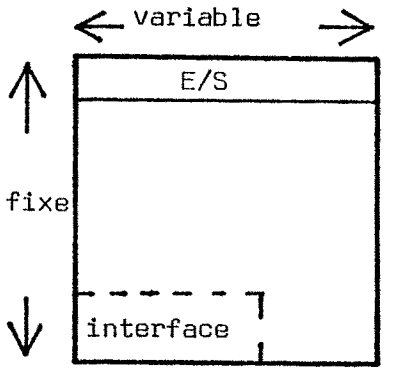


Fig. 9.15 - Topologie des modules

Cependant, l'analyse de certains exemples de modules a montré qu'il y a de grandes différences de taille entre eux. Il est donc impossible de choisir une dimension fixe, car elle ne serait adaptée qu'à certains modules et obligerait les autres à avoir des formes rectangulaires très prononcées.

On a donc été amené à considérer deux dimensions qui, comme on le verra dans la section suivante, s'adaptent bien aux surfaces des modules envisagés.

Dans une technologie 6µm les dimensions choisies sont :

4,5mm ce qui donne pour un module carré - 20,25mm2

2,25mm ce qui donne pour un module carré - 5,04mm2

Placement des modules :

Un outil très simple, permettant de choisir automatiquement le meilleur placement pour les modules, doit être créé pour faciliter cette phase de la méthode.

Le programme doit avoir comme données d'entrée les surfaces ou les estimations de surface et le nombre d'E/S des différents modules. Il calcule ensuite le placement optimal des modules autour du noyau de communication, de façon à minimiser la surface du circuit.

Les contraintes de placement pourraient être définies de façon à permettre au concepteur d'indiquer au programme la localisation d'un module particulier.

9.5. MODULE DE TEST

Le problème essentiel dans la mise au point et dans le test de production est d'accéder aux éléments internes du circuit. Dans MOSAIC, ce problème est encore plus important car le noyau de communication du système (processeur noyau et bus commun)ne sont pas accessibles à partir des E/S du circuit. Il nous a donc semblé important d'introduire dans tous les circuits un organe permettant de faciliter le test.

L'utilisation d'une communication série avec l'extérieur permet d'économiser des broches qui, comme on le verra dans le chapître 10, sont une des contraintes dans la réalisation des circuits systèmes.

Cependant, une communication série pose un problème car le débit interne sur le bus peut être très important (pour tcy = 100ns le débit max = 10Moctets) ce qui rend impossible d'envoyer vers l'extérieur toute l'information véhiculée par le bus.

Une solution envisageable est de rendre le module test intelligent de façon à ce qu'il puisse exécuter des séquences de test complètes ne signalant à l'extérieur que les évènements significatifs.

Cette solution pose cependant deux problèmes :
a/ comment tester le module de test ?
b/ le module de test risque d'occuper une surface importante.

On a donc préféré définir un module de test beaucoup plus simple qui n'exécute que deux fonctions de base avec lesquelles l'utilisateur pourra programmer à l'extérieur des séquences de test permettant de vérifier le fonctionnement
- du processeur noyau,
- de la mémoire de communication,
- des modules.

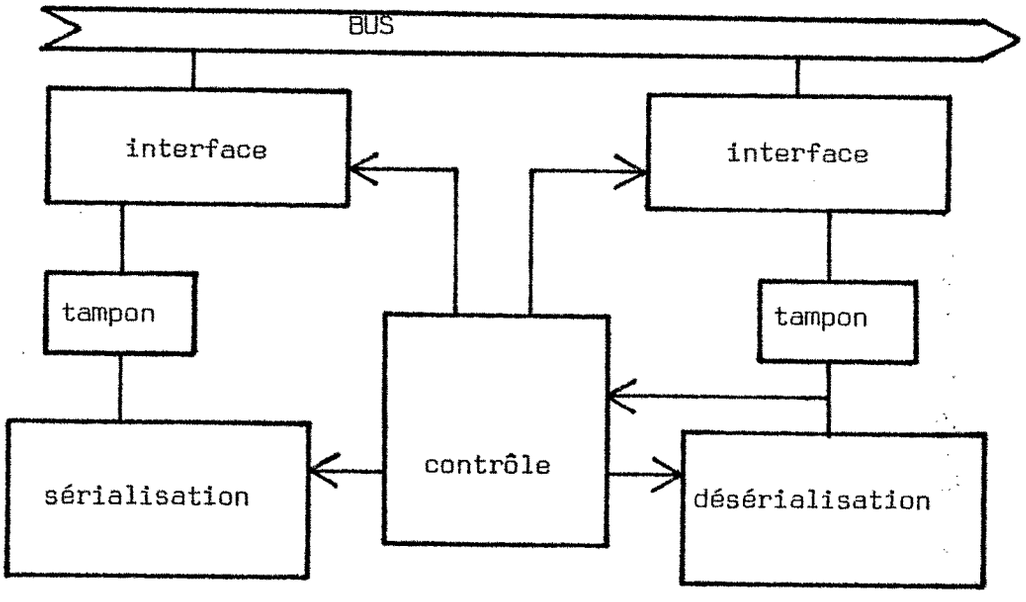


Fig. 9.16 - Module test

Les actions exécutées par le module test sont :

- activer les lignes du bus de façon à envoyer une requête à un moniteur externe (l'appel et les paramètres doivent être fournis de l'extérieur) ;
- envoyer vers l'extérieur les caractères qui suivent l'activation des lignes :
  - . Prêt
  - . Continuer
  - . Attente
- décoder toutes les configurations anormales des lignes du bus.

Les séquences de test à programmer à l'extérieur peuvent être exemplifiées par :

- pour tester le processeur noyau et la mémoire de communication :
  - . envoyer un appel - Ecrire-Moniteur A
  - . envoyer un appel - Lire-Moniteur A .
- pour tester le mécanisme de synchronisation des files d'attente :
  - . envoyer un appel à un moniteur de façon à faire débloquer un processus d'attente. Sur le bus on doit voir le dialogue correspondant au déblocage d'un processus en attente.

Pendant le fonctionnement normal, le module de test peut être utilisé pour décoder en permanence les lignes du bus de façon à détecter les configurations correspondant à des situations anormales.

Les exceptions qui ont été mentionnées dans le chapître 2 peuvent être détectées par ce mécanisme.

Il est évident que, pour le traitement des exceptions, on n'envisage d'exécuter que des actions très simples, du type paramétrisation d'un module, remise à zéro des pointeurs d'un moniteur externe ou envoi d'une alarme vers l'extérieur.

Un petit PLA peut être chargé de générer les requêtes au processeur noyau nécessaires pour exécuter ces actions.

Quoique potentiellement possibles, des actions plus complexes, du type détection de panne d'un module ou même des tentatives pour la résoudre, représenteraient une charge très importante pour le module de test qui deviendrait un processeur occupant une surface appréciable.

Bien qu' exécutant des actions très simples, un module de test avec ces caractéristiques, nous semble bien adapté pour le test en production et pour faciliter la mise au point. Cependant, pour évaluer son efficacité réelle il faudra définir les séquences de test complètes permettant de vérifier le bon fonctionnement du noyau de communication et les modules.

CHAPITRE 10

C O N C L U S I O N

10.1. ETAPES DE LA METHODE

Les étapes, les plus importantes, de la méthode qui vient d'être décrite dans les chapîtres précédents peuvent être résumés par :

Analyse fonctionnelle :

1/ Analyse et décomposition de l'application dans les modules et moniteurs externes nécessaires :

Dans cette décomposition il faut tenir compte des spécifications des modules déjà existants dans la bibliothèque et qui pourraient être utilisées à nouveau.

Définition des fonctions exécutées par les modules et des algorithmes des moniteurs externes : cette étape est une des plus importantes et des plus délicates de la méthode car les choix effectués à ce niveau influenceront fortement la performance et la surface du circuit.

2/ Programmer dans le langage de spécification les algorithmes des nouveaux modules et des moniteurs externes nécessaires.

Un langage de programmation nous semble un outil bien adapté pour formaliser les actions à exécuter par le circuit car il oblige à définir, dès le début, tous les détails du fonctionnement logique du système.

3/ Utiliser le compilateur du langage de spécification pour :
a/ vérifier la cohérence du programme,
b/ faire une simulation fonctionnelle,
c/ calculer les temps des exécutions des procédures des moniteurs externes.

4/ Etablir le cahier des charges du circuit :
Le programme de spécification constitue déjà une partie importante du cahier des charges car il contient une description très précise du fonctionnement du circuit.

A cette description fonctionnelle il faut rajouter les spécifications concernant les E/S :

- fréquences logiques,
- caractéristiques électriques.

Définition de l'architecture :

1/ Choisir l'implémentation des moniteurs externes (automate, processeur noyau) et le type d'allocation du bus commun utilisé.

2/ Simuler les performances du système en utilisant les modèles QNAP convenablement paramétrés.

Implantation :

1/ Obtenir par une compilation du programme de spécification dans le langage intermédiaire du processeur noyau, ou dans le langage de microprogrammation des automates, le remplissage de leurs PLAs respectifs.

2/ Calculer la surface occupée par ces mécanismes de communication (surface mémoire de communication, surface de la partie contrôle du processeur noyau).

3/ Définir la topologie du circuit et le placement des modules. Après avoir évalué les surfaces des nouveaux modules, le concepteur doit fournir au programme de placement automatique, les surfaces et le nombre d'E/S de tous les modules.

4/ Dessiner les nouveaux modules qui doivent s'adapter le mieux possible aux canevas d'implantation définies précédemment.

Dans le chapître 9 on a établi une séparation entre les modules en fonction de leur structure interne.

Les "modules programmables" sont adaptés à leur fonction par le programme contenu dans leurs mémoires mortes. Il suffit donc de compiler les processus PASCAL, du programme de spécification, dans le langage machine des modules. Eventuellement, il peut être nécessaire d'adapter la taille de la mémoire programme ou de données.

L'adaptation des modules programmables est donc très facile, cependant on a aussi fait remarquer que les modules programmables occupent une surface importante et que certaines fonctions ne peuvent être implémentées qu'en utilisant une structure matérielle spécialisée.

La définition et le dessin des masques des "modules dédiés" représentera donc la partie la plus longue du processus de conception d'un circuit MOSAIC.

Dans l'annexe 6 sont décrites deux applications qui ont été étudiées.

Dans ces exemples on a surtout cherché à évaluer la surface des circuits permettant d'implémenter deux applications d'une complexité déjà appréciable.

La technologie sur laquelle les évaluations ont été basées est le NMOS 2µm, les résultats obtenus montrent qu'avec cette technologie des circuits système d'une complexité équivalente à ceux qui ont été évalués peuvent être produits avec des rendements satisfaisants.

10.2. REMARQUES FINALES

Le but principal de MOSAIC était de proposer une méthode qui puisse permettre de concevoir dans un temps raisonnable des circuits très complexes permettant d'implémenter des petits systèmes informatiques.

Adaptation des circuits VLSI

Le premier problème à résoudre a été de choisir la façon dont un circuit très complexe peut être adapté à une application.

Dans l'étape précédente de l'évolution de la complexité des circuits intégrés, la programmation a été proposée comme une nouvelle méthode d'adaptation d'un circuit à une application. L'utilisation de la programmation a permis d'offrir à l'utilisateur, une machine virtuelle facilement programmable implémentée par une structure matérielle complexe mais dont l'utilisateur n'avait même pas connaissance.

Une approche programmable n'était pas utilisable pour les circuits systèmes car certaines fonctions ne peuvent être implémentées que par des structures matérielles spécialisées.

On a donc choisi d'adapter le circuit en modifiant sa structure interne de façon à pouvoir utiliser dans sa réalisation des organes matériels spécialisés dans l'exécution d'une fonction spécifique.

Cette conception à la demande devait cependant respecter le deuxième objectif initial qui était d'obtenir "un temps de conception raisonnable". Il fallait donc définir une méthode de conception adaptée à la complexité des circuits à produire.

Dans la méthode proposée dans MOSAIC, le circuit est conçu comme l'assemblage de blocs d'une complexité importante autour d'un mécanisme de communication standard.

L'adaptation du circuit se fait donc en utilisant une nouvelle méthode basée sur le choix d'un ensemble de processeurs fonctionnels et dans la définition des algorithmes utilisés pour les faire communiquer.

Il nous semble que l'approche proposée permet avec un coût de conception faible d'obtenir des systèmes informatiques bien adaptés et performants dans l'exécution d'une application. Il est cependant évident que cette méthode n'est utilisable que pour un type particulier de circuits et que par exemple elle ne serait pas adaptée pour concevoir des circuits de haute performance du type microprocesseur 32 bits.

Méthode de conception

Temps de conception

La diminution du temps de conception a été une de nos préoccupations fondamentales.

L'idée de base de la méthode proposée dans MOSAIC est de diviser la complexité globale du circuit en fractionnant la structure matérielle en des modules indépendants.

Cet objectif est le même que celui des méthodes de définition hiérarchique de briques, où l'on construit une hiérarchie de briques de plus en plus complexe en partant d'un jeu initial de briques standard.

L'approche du problème choisie dans cette méthode, considérée dans la littérature

comme une des solutions les plus intéressantes pour le VLSI [OAK.79, PRE.79] très différente de celle que nous avons utilisée.

La définition hiérarchique est une méthode qui dépend fortement des outils de CAO (routage, placement) et où il n'y a aucune analyse du type de circuits à produire.

Dans MOSAIC l'approche a été l'inverse car on est parti de l'analyse des applications pour définir une méthode de conception qui leur soit adaptée. Cette approche est donc moins générale que la précédente mais ce manque de généralité est à notre avis largement compensé par l'optimisation en surface que l'on peut obtenir par rapport à des méthodes d'interconnexion automatiques.

Le temps de conception d'un nouveau circuit se résume à :

- analyse et programmation du système,
- choix de la solution d'architecture,
- dessin de nouveaux modules.

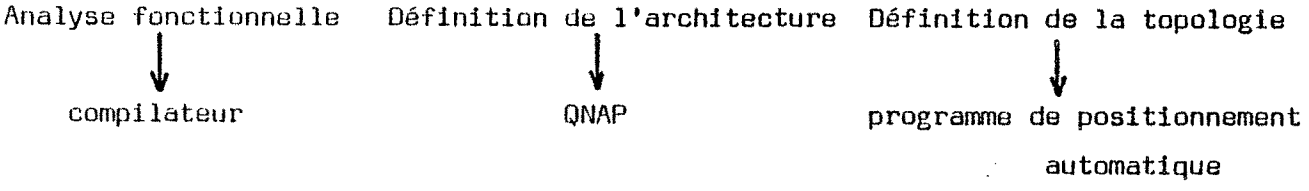
Il est évident que le dernier point représentera dans certaines applications le pourcentage le plus important du temps de conception. Cependant l'expérience acquise en analysant un certain nombre d'applications montre que dans ce type de circuits certains modules sont réutilisés plusieurs fois car la plupart des applications en ont besoin, ex.: clavier, visu, liaison asynchrone.

L'investissement en temps de conception des premiers circuits peut donc être amorti dans la réalisation des circuits suivants.

Outils de CAO

La méthode proposée ne nécessite pas d'outils de CAO très sophistiqués. On a voulu définir des outils simples (compilateurs, QNAP, positionnement automatique),

qui ne constituent pas un système global de CAO, mais plutôt un ensemble d'outils ponctuels qui permettent de valider et d'automatiser certaines phases de la conception :



Pour que la chaîne soit complète il manque un outil permettant de définir la réalisation matérielle d'un module à partir de sa description en PASCAL dans le programme de spécification.

Pour des modules ayant une structure du type informatique (partie opérative, partie contrôle) il y a actuellement des travaux de recherche [SUZ.79, ANC.79] qui sont développés dans ce domaine. Il faut cependant être très prudent car il nous semble impossible de définir un outil de CAO totalement automatique qui puisse être simultanément général et performant.

Démarche descendante

On a voulu que, dans la démarche utilisée, les décisions concernant le dessin réel du circuit ne soient prises qu'à la fin, après que la structure globale soit complètement définie.

La méthode proposée est donc une démarche descendante où l'utilisateur étudie et simule le système et seulement après avoir défini complètement la topologie et les mécanismes de communication dessine les nouveaux modules.

Ce type d'approche permet au concepteur de concentrer initialement ses efforts sur la définition des algorithmes et des fonctions à exécuter par le système au lieu

de commencer à figer immédiatement certaines parties matérielles du circuit sans que la totalité des fonctions à réaliser soient définies et comprises.

Les outils de CAO définis précédemment permettent d'expérimenter facilement plusieurs solutions et d'en comparer les avantages avant de commencer la phase d'implantation.

Isochronisme

Dans le chapître 1 on a mentionné le problème de l'isochronisme comme un des problèmes nouveaux qui se posent dans la conception de circuits VLSI.

Quoique ce problème ne soit pas encore important dans une technologie 2µm on a voulu le traiter car il sera certainement très important à moyen terme.

Considérer le circuit comme une hiérarchie de zones isochrones permet de résoudre le problème d'une façon plus simple que l'utilisation de protocole asynchrones qui impliquent une augmentation de la complexité matérielle et qui sont plus difficiles à formaliser.

Le problème de la métastabilité n'a pas été abordé dans ce texte mais dans la solution hiérarchique proposée il peut être résolu en utilisant un mécanisme d'asservissement de la phase de l'horloge interne par rapport à la phase de l'horloge bus, de façon à éviter la zone critique de métastabilité.

Une étude serait nécessaire pour dessiner et simuler un circuit simple pour réaliser cet asservissement.

Nombre de broches des boîtiers

Le nombre de broches des boîtiers est un problème qui peut limiter fortement le domaine d'application des circuits systèmes.

Vis-à-vis des utilisateurs, le grand avantage d'intégrer tout un système informatique dans un boîtier est de réduire pratiquement à un seul circuit toute l'électronique d'une application (à l'exception de quelques circuits d'adaptation d'impédance).

Si on essaie d'implémenter des applications très proches du contrôle d'organes matériels, on s'aperçoit que le nombre de E/S nécessaires croît rapidement.

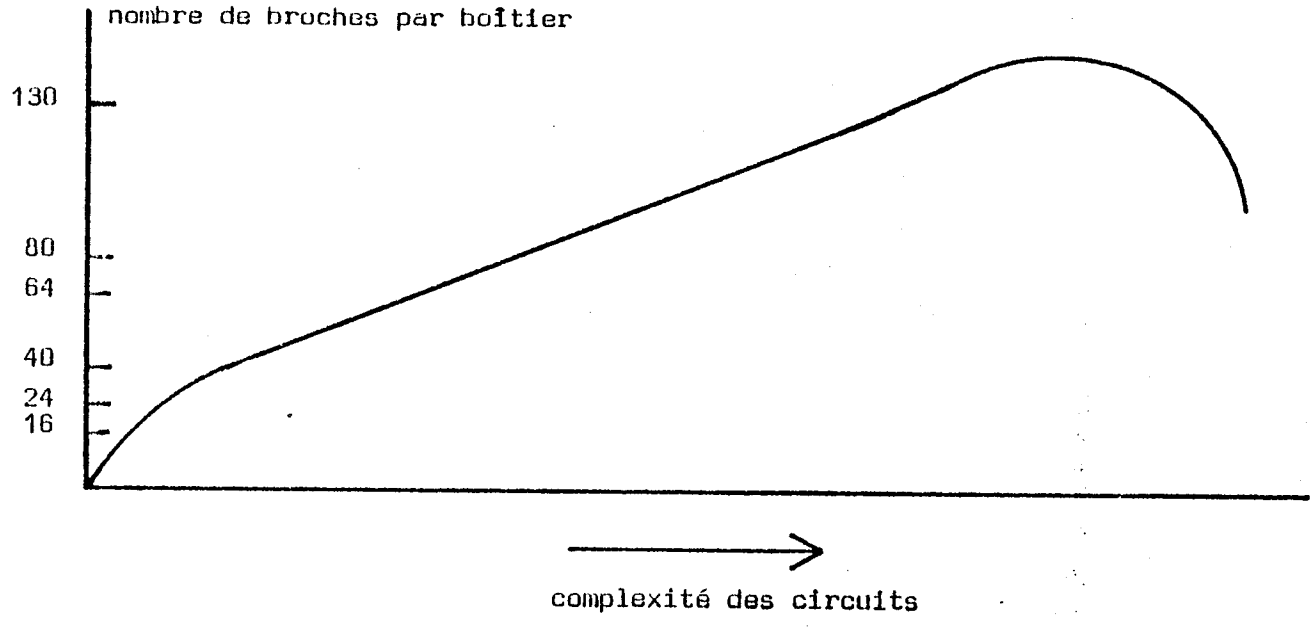


Fig.10.3 - Nombre de broches des boîtiers en fonction de la complexité[LAS.79

Pour exploiter correctement une solution monolithique dans ce type d'application il faut donc pouvoir disposer de boîtiers ayant un nombre de pattes de l'ordre d'une centaine.

Si la technologie de fabrication des boîtiers et des substrats (plaque de circuits imprimés,hybrides) ne permettra pas d'avoir un tel nombre de broches, l'évolution de la technologie d'intégration ne fera pas diminuer le nombre de boîtiers de ces systèmes informatiques.

Les applications seront implémentées par un certain nombre de circuits résultant d'une décomposition fonctionnelle qui découle du nombre de broches maximal des boîtiers. Le maigrissement de la technologie ne fera que diminuer la surface des circuits mais le nombre de boîtiers utilisés restera inchangé.

Consommation et tensions d'alimentation

Dans tout le texte on a utilisé comme référence la technologie NMOS avec les niveaux de tension actuels. Ce choix était le seul qui nous permettait de faire des évaluations temporelles et de surface basées sur des paramètres réels de la technologie.

Cependant on est bien conscient de la limitation imposée par la puissance dissipée dans des circuits de 40mm2 et ayant plus de 100 000 MOS.

Le maigrissement de la technologie doit donc être obligatoirement accompagné d'une diminution des niveaux de tension actuels. Ce problème qui se pose pour tous les types de circuit doit conduire à l'établissement de nouveaux standards pour l'industrie.

La consommation peut être aussi minimisée par l'utilisation intensive de logique dynamique ou par l'utilisation d'une technologie CMOS.

Il faut remarquer que les choix de base au niveau de l'architecture du circuit ne sont pas dépendants de la technologie et qu'à part les nécessaires redéfinitions des organes matériels, la méthode de conception proposée pouvait aussi être utilisée avec une technologie CMOS.

Dans MOSAIC on a voulu proposer une solution pour la conception d'une famille de circuits très complexes qui doit représenter un pourcentage important du marché VLSI des années à venir.

L'approche choisie prétend résoudre les problèmes qui à notre avis étaient les plus importants dans la conception de ce type de circuits :

- le temps de conception,
- l'adaptation d'un circuit à l'application,
- les problèmes d'isochronisme.

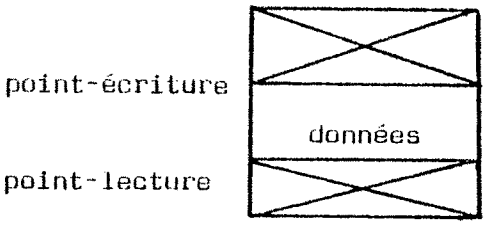
Les évaluations de surface et temporelles qui ont été faites semblent montrer que la solution proposée permet d'avoir des circuits performants sans sacrifier la surface de silicium occupée. Il serait maintenant indispensable d'essayer de traiter un plus grand nombre d'applications et de simuler en logique discrète le fonctionnement du noyau de communication.

A N N E X E 1

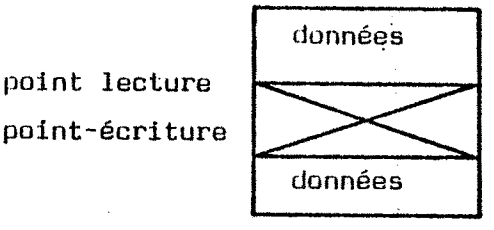
EXEMPLES DE MONITEURS EXTERNES

Les moniteurs externes utilisés dans MOSAIC ont généralement des structures très simples (fifo, pile, tampon). Dans cette annexe, nous présentons quatre exemples correspondant à des structures de données souvent utilisées. Ces mêmes exemples seront aussi utilisés dans l'annexe 2 pour montrer comment s'enchaînent les niveaux de contrôle dans le processeur noyau.

1. FIFO



Mod = false



mod = vrai

MOD indique si les pointeurs d'écriture et de lecture se trouvent dans la même boucle.

```
FIFO = external monitor

structure de données
        var : TABLEAU : array (0 :: N) of info-type ;
              POINT-ECRITURE, POINT-LECTURE : integer ;
              TAILLE : integer ;
              MOD : boolean ;
              LIRE-POSSIBLE : condition (POINT-LECTURE = POINT-ECRITURE ∧ MOD) ;
              ECRI-POSSIBLE : condition (POINT-LECTURE = POINT-ECRITURE ∧ ¬ MOD) ;

procédure
        procedure entry ECRIRE (DONNEE : info-type) ;
        begin
              wait (ECRI-POSSIBLE) ;
              TABLEAU (POINT-ECRITURE) := DONNEE ;
              POINT-ECRITURE := POINT-ECRITURE + 1 ;
              if POINT-ECRITURE > TAILLE then
                                    begin POINT-ECRITURE := 0 ;
                                          MOD := true ;
                                    end ;
        end ;
```

```
                    procedure entry LIRE (DONNEE : info-type)
                    begin
                          wait (LIRE-POSSIBLE) ;
                          DONNEE := TABLEAU (POINT-LECTURE) ;
        procédure         POINT-LECTURE := POINT-LECTURE + 1 ;
                          if POINT-LECTURE > TAILLE then
                                                begin POINT-LECTURE := 0 ;
                                                      MOD := false ;
                                                end ;
                    end ;
initialisation
de la structure     begin
de donnée                 POINT-LECTURE := POINT-ECRITURE := 0 ; MOD = false
                    end ;
    end ;
```

2. PILE

```
PILE = external monitor
                    var : TABLEAU : array (0 :: N) of info-type ;
        structure         POINT-LECTURE, POINT-ECRI : integer ;
        de données        TAILLE : integer ;
                          LIRE-POSSIBLE : condition (POINT-LECTURE > 0) ;
                          ECRI-POSSIBLE : condition (POINT-ECRITURE < TAILLE) ;
                    procedure entry LIRE (DONNEE : type-info)
                    begin
                          wait (LIRE-POSSIBLE) ;
                          DONNEE := TABLEAU (POINT-LECTURE) ;
                          POINT-LECTURE := POINT-LECTURE - 1 ;
                    end ;
                    procedure entry ECRIRE (DONNEE : type-info)
        procédures  begin
                          wait (ECRI-POSSIBLE) ;
                          TABLEAU (POINT-ECRITURE) := DONNEE ;
                          POINT-ECRITURE := POINT ECRITURE + 1 ;
                    end ;
```

```
initialisation
        begin
            POINT-ECRITURE := POINT-LECTURE := 0 ;
        end ;
```

3. TAMPON

Dans cet exemple, nous considérons un tampon rempli par un processus écrivain. Quand le tampon est plein, le processus lecteur est réveillé et vide le tampon.

```
TAMPON : external monitor

structure de données
        var : TABLEAU : array (0 :: N) of info-type ;
              POINTEUR, TAILLE : integer ;
              VIDE : boolean ;
              LIRE-POSSIBLE : condition (¬ VIDE) ;
              ECRI-POSSIBLE : condition ( VIDE) ;

procédures
        procedure entry LIRE (DONNEE : type-info)
        begin
              wait (LIRE-POSSIBLE) ;
              DONNEE := TABLEAU (POINTEUR) ;
              POINTEUR := POINTEUR + 1 ;
              if POINTEUR = TAILLE then
                                  begin VIDE := true ;
                                        POINTEUR := 0
                                  end ;
        end ;
        procedure entry ECRIRE (DONNEE : type-info)
        begin
              wait (ECRI-POSSIBLE) ;
              TABLEAU (POINTEUR) := DONNEE ;
              POINTEUR := POINTEUR + 1 ;
              if POINTEUR = TAILLE then
                                  begin VIDE := false ;
                                        POINTEUR := 0
                                  end ;
        end ;

initialisation
        begin
              VIDE := true ;
              POINTEUR := 0
        end ;
```

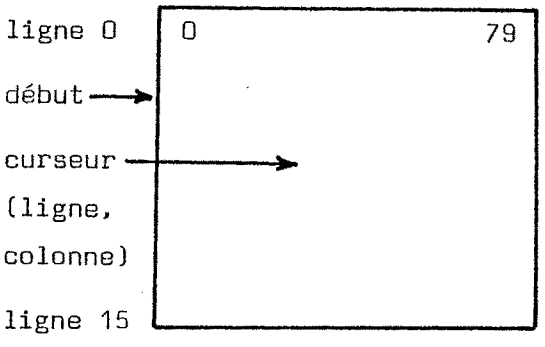
4. Mémoire de rafraichissement d'un écran de télévision

Une structure un peu moins classique peut être représentée par une mémoire de rafraichissement d'un écran de télévision.

Fonctions à éxécuter :

- lire les caractères selon un algorithme séquentiel bouclé,
- écrire les caractères selon un algorithme séquentiel (gestion de la dernière ligne selon un algorithme de roll-up),
- écrire un retour chariot,
- déplacer le curseur (up, down, left, right).

Supposons une mémoire de rafraichissement de 16 lignes et 80 caractères par ligne. Pour simplifier l'adressage des caractères avec le système de roll-up, nous considérons que la page logique à visualiser est différente de la page physique.



début-première ligne de la page logique

Pour les opérations de positionnement du curseur du type retour chariot, up, down, left, right, il faut maintenir la notion de ligne et de colonne. Cependant, il serait peu intéressant de calculer l'adresse où écrire un caractère en faisant une multiplication, car cette opération n'existe pas dans le jeu d'instructions de base du processeur noyau. On a donc choisi d'utiliser un pointeur physique pour l'adressage et un pointeur logique (ligne, colonne) pour les autres opérations.

```
MRAFRAICH = external monitor
            var :  MEM array (0 :: 1280) of info-type ;
                   CURSEUR_LIGNE, CURSEUR_COLONNE : integer :
                   POINT_LECT, POINT_ECR : integer ;
                   DEBUT : integer
            procedure entry LIRE (DONNEE : type-info)
            begin
                   DONNEE := MEM (POINT-LECT)
                   POINT_LECT := POINT_LECT + 1 ;
                   if POINT LECT > 1280 then POINT LECT := 0 ;
            end ;
            procedure entry ECRIRE (DONNEE)
            begin
                   MEM (POINT_ECR) := DONNEE
                   POINT_ECR := POINT_ECR + 1 ;
                   if POINT_ECR = 1280 then POINT_ECR := 0 ;
                   CURSEUR_COLONNE := CURSEUR_COLONNE + 1 ;
                   if CURSEUR_COLONNE = 80 then
                                        begin CURSEUR_COLONNE := 0 ;
                                              CURSEUR_LIGNE := CURSEUR_LIGNE + 1 ;
                                              if CURSEUR_LIGNE = 16 then
                                                 begin  CURSEUR_LIGNE := 15 ;
                                                        DEBUT := DEBUT + 1 ;
                                                        if DEBUT = 16 then DEBUT := 0
                                                 end ;
                                        end ;
            end :
            procedure entry RETOUR-CHARIOT
            begin
                   for I := CURSEUR_COLONNE until 80 do
                   begin MEM (POINT_ECR) := "0" ;
                         POINT_ECR := POINT_ECR + 1 ;
                         if POINT_ECR = 1280 then POINT_ECR := 0 ;
                   end ;
                         CURSEUR_COLONNE := 0 ;
                         CURSEUR_LIGNE := CURSEUR_LIGNE + 1 ;
                         if CURSEUR_LIGNE = 16 then
```

```
                                    begin CURSEUR_LIGNE = 15 ;
                                          DEBUT := DEBUT + 1 ;
                                          if DEBUT = 16 then DEBUT = 0 ;
                                    end ;
end ;
procedure entry UP
begin
      if CURSEUR_LIGNE ≠ 0 then
      begin
            CURSEUR_LIGNE := CURSEUR_LIGNE - 1 ;
            POINT_ECR := POINT_ECR - 80 ;
      end ;
end ;
procedure entry DOWN
begin
      CURSEUR_LIGNE := CURSEUR_LIGNE - 1 ;
      if CURSEUR_LIGNE = 16 then
      begin
            DEBUT := DEBUT + 1 ;
            if DEBUT = 16 then DEBUT := 0 ;
            CURSEUR_LIGNE := 15 ;
      end ;
      POINT_ECR := POINT_ECR + 80
end ;
procedure entry LEFT
begin
      if CURSEUR_COLONNE = 0 then
      begin
            if CURSEUR_LIGNE ≠ 0 then
                                    begin CURSEUR_COLONNE = 79
                                          CURSEUR_LIGNE := CURSEUR_LIGNE - 1
                                    end ;
      end ;
      else CURSEUR_COLONNE := CURSEUR_COLONNE - 1 ;
      POINT_ECR := POINT_ECR - 1 ;
end ;
```

La procédure RIGHT est identique.

D'autres procédures seraient cependant nécessaires ; exemple : début balayage, lecture position curseur, etc ..

5. Moniteurs réservables

Dans ce cas, il faut rajouter aux moniteurs classiques (2.5.1) :

. les procédures de "réservation" et de "libération",

. dans les procédures qui effectuent les accès aux données, une section permettant de vérifier l'identité du processus qui l'exécute.

Pour représenter ces extensions, on considère le FIFO classique de l'exemple 1 réservable en écriture. On ne notera que les éléments qui doivent être rajoutés.

```
FIFO_RESER = external monitor

structure de données
        var :
              cf. exemple 1
            LIBRE : condition ( RESERVE) ;
            RESERVE : boolean ;
            UTILISATEUR : type nom processus ;

procédures d'accès aux données
        procedure entry ECRIRE (DONNEE : type-info, NOM : type nom processus)
        begin
            if NOM ≠ UTILISATEUR then SIGNALER_EXCEPTION ("VIOLATION_RESERV") ;
            .
            :
        end ;
        procedure entry LIRE (DONNEE : type info)
            .
            : cf. exemple 1
            .
        end ;
```

```
                procedure entry RESERVER (NOM : type nom processus)
                begin
procédures              wait (LIBRE) ;
de réserva-             UTILISATEUR := NOM ;
tion et de              RESERVE := true ;
libération      end ;
du              procedure entry LIBERER (NOM : type nom processus)
moniteur        begin
                        if NOM ≠ UTILISATEUR then SIGNALER_EXCEPTION ("VIOLATION_RESERV") ;
                        RESERVE := false ;
                end ;
                begin
                        RESERVE := false
initiali-               :
sation                  :
                end ;
```

6. Remarques

On a utilisé la désignation type-info sans en préciser la signification. Dans le chapitre 4, on a choisi comme unité d'information du système l'octet. Les données transférées sont donc des multiples entiers d'octets. Cependant, on a été obligé de limiter le nombre d'octets à 16 pour éviter qu'un processus ne bloque très longtemps le bus commun

type-info = (octet, octet 1, octet 2, ..., octet 16).

7. Structure de données des moniteurs

Les exemples cités sont relativement représentatifs des types de moniteurs que l'on retrouve dans les applications envisagées. Les structures de données qui ont été décrites permettent de voir quelles sont les informations nécessaires pour implémenter les moniteurs :

- 1 tableau,
- 1, 2, 4 pointeurs,
- quelques variables booléennes,

. des variables pour mémoriser le nom des utilisateurs,
. des constantes, comme par exemple taille.

Dans l'implémentation des moniteurs qui est faite au chapitre 5, le tableau correspond à la zone de mémoire réservée pour le stockage des données. Les autres éléments font partir du descripteur du moniteur où sont regroupées toutes les informations sur l'état de ce dernier.

ANNEXE 2

LANGAGE INTERMÉDIAIRE

Nous présentons ici quelques unes des instructions intermédiaires nécessaires pour faire exécuter les moniteurs décrits dans l'annexe 1. Pour quelques unes de ces instructions, nous décrivons les actions exécutées par les micro-instructions qui correspondent à leur interprétation.

Pour simplifier l'écriture des micro-instructions, nous utilisons un pseudo-Algol comme langage de description.

Les éléments référencés dans les programmes sont soit ceux mentionnés dans la description des parties opératives et de contrôle du processeur noyau (5.2), soit les éléments des descripteurs des moniteurs (5.1).

Pour décrire les micro-instructions, on utilise le format suivant qui correspond à une simplification de celui présenté au § 5.2.3. :

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement

Dans le champ séquencement est spécifié un des 8 types possibles de séquencements décrits au § 5.2.3. Pour rendre plus simple la notation, on utilise les mnémoniques suivantes :

SEQ1 - passage à la micro-instruction suivante

SEQ2 - passage à l'instruction intermédiaire suivante

SEQ3 - passage à l'instruction intermédiaire spéciale

BR1 - en fonction du test, soit passage à la micro-instruction suivante,
soit passage à l'instruction intermédiaire suivante

BR2 - en fonction du test, soit sauter la micro-instruction suivante,
soit continuer en séquence,

BR3 - en fonction du test, soit boucler, soit passage à la micro-instruction suivante,

BR4 - en fonction du test, soit boucler, soit passage à l'instruction intermédiaire suivante,

BR5 - en fonction du test, soit passage à l'instruction intermédiaire spéciale,
soit passage à l'instruction intermédiaire suivante.

Les instructions intermédiaires peuvent être divisées en quatre grands groupes :

. test des conditions d'entrée de la procédure (CE)

. accès aux données (ACCES)

. test des conditions de sortie de la procédure (CS)

. instructions spéciales (IS).

1. Test des conditions d'entrée

1.a. CEFIFO - lecture

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement
1	NOP	NOP	test F $\overline{MOD}$	NOP	BR1
2	POINT-LECT— POINT-ECR	NOP	test zéro	NOP	BR1
2	COMP-LECT:= COMP-LECT+1	NOP	NOP	NOP	SEQ3

Remarques: test F = test flag

test = test bit d'état

COMP-LECT = compteur des processus lecteurs bloqués

L'instruction spéciale qui est appelée par SEQ3 à la fin de la procédure, est évidemment "ATTENTE".

1.b. CEFIFO - écriture

Identique à CEFIFO Lecture à l'exception du test qui se fait sur MOD au lieu de $\overline{MOD}$.

1.c. CEFIFO - lecture - étendu

La désignation 'étendu' correspond à des moniteurs ayant une taille supérieure à 256 octets (§ 5.1.1).

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement
1	NOP	NOP	test F $\overline{\text{MOD}}$	NOP	BR1
2	POINT-LECT ← POINT-ECR	NOP	test zéro	NOP	BR1
2	NSEG-LECT ← NSEG-ECR	NOP	test zéro	NOP	BR1
2	COMP-LECT:= COMP-LECT+1	NOP	NOP	NOP	SEQ3

NSEG-LECT = numéro du segment

1.d. CEFIFO - écriture - étendu

1.e. CEPile - lecture

1.f. CEPile - écriture

1.g. CEPile - lecture - étendu

1.h. CEPile - écriture - étendu

1.i. CETampon - lecture

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement
1	NOP	NOP	test F VIDE	NOP	BR1
2	COMP-LECT:= COMP-LECT+1	NOP	NOP	NOP	SEQ3

Pour le tampon étendu, il n'y a aucune différence, car le test ne dépend pas des pointeurs, mais seulement du flag "VIDE".

1.j. CETampon - écriture

Pour l'évaluation des conditions d'entrée référencées, on a 28 micro-instructions. On peut aussi inclure dans les tests des conditions d'entrée la vérification de la réservation.

1.k. Test - réservé - lecture

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement
2	utili-lect-EBUS	NOP	test zéro	NOP	BR5

Remarques: EBUS= entrée directe au bus de données

BR5 permet d'appeler l'instruction intermédiaire spéciale "exception".

2. Accès aux données

2.a. Accès séquentiel lecture

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement
1	SUAL:=POINT-LECT	RADR:=SUAL	X	X	SEQ1
1	SUAL:=NOCTET + POINT-LECT	RADR:=RADR+ Base RNOCTET:= RNOCTET+1	X	X	SEQ1
1	POINT-LECT:=SUAL	RADR:=RADR+1 RNOCTET:= RNOCTET+1	test fin accès	lire	BR1
1	NOP	RADR:=RADR+1 RNOCTET:= RNOCTET+1	test fin accès	lire	BR4

Remarques: SUAL = sortie UAL

RADR = registre d'adresse mémoire

RNOCTET = registre utilisé pour l'incrémentation de Noctet

La dernière micro-instruction est exécutée par les moniteurs externes qui ont une unité d'information supérieure à 1 octet et elle permet d'implémenter une boucle contrôlée par l'incrémentation de Noctet dans la partie 12 bits.

2.b. Accès - séquentiel - écriture

2.c. Accès - séquentiel - lecture - étendu

2.d. Accès - séquentiel - écriture - étendu

2.e. Accès - séquentiel - bouclé - lecture

2.f. Accès - séquentiel - bouclé - écriture

2.g. Accès - séquentiel - bouclé - lecture - étendu

2. h. Accès - séquentiel - bouclé - écriture - étendu

2.i. Accès - séquentiel - décroissant - lecture

2.j. Accès - séquentiel - décroissant - écriture

2.k. Accès - séquentiel - décroissant - lecture - étendu

2.l. Accès - séquentiel - décroissant - écriture - étendu

3. Conditions de sortie

3.a. CS - FIFO - lecture

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement
2	COMPT-ECRI	X	test zéro	X	BR1
2	COMPT-ECRI:= COMPT-ECRI-1	X	X	X	SEQ3

Remarque: COMPT-ECRI = compteur pour les processus écrivains en attente.

Dans cette instruction, on se limite à tester s'il y a des processus en attente d'écriture. S'il y en a, on décrémente le compteur et on se branche à l'instruction spéciale "continuer".

3.b. CS - FIFO - écriture

3.c. CS - Pile - écriture

3.d. CS - Pile - lecture

3.e. CS - tampon - lecture

nombre de cycles	opérateur arithmétique 8 bits	opérateur arithmétique 12 bits	opérateur binaire	mémoire commune	séquencement
2	taille-POINT-LECT	X	test zéro	X	BR1
2	POINT-LECT:=0	X	X	X	SEQ1
2	COMPT-ECRI	X	test zéro	X	BR1
2	COMPT-ECRI:= COMPT-ECRI-1	X	X	X	SEQ3

Si le tampon a été complètement lu, on remet le pointeur de lecture à zéro et on teste s'il y a des processus écrivains en attente.

3.f. CS - tampon - écriture

4. Instructions spéciales

4.a. Attente

Cette instruction positionne la ligne "attendre" et envoie sur le bus l'identification de la condition (procédure, adresse) (§ 5.3.4).

4.b. Continuer

Cette instruction positionne la ligne "continuer" et effectue le changement de conteste décrit au § 5.3.4.

4.c. Initialisation

Dans l'instruction d'initialisation, il faut spécifier le nombre de champs du descripteur qu'il faut charger à partir du catalogue.

Etant donné que la structure de la mémoire (§ 5.2.2.) permet de transférer 4 octets (2 lignes mémoire) dans chaque lecture, on doit avoir quatre instructions différentes d'initialisation :

initialisation_1 - 2 lignes - descripteur ≤ 4 octets
initialisation_2 - 4 lignes - descripteur ≤ 8 octets
initialisation_3 - 6 lignes - descripteur ≤ 12 octets
initialisation_4 - 8 lignes - descripteur ≤ 16 octets

4.d. Sauvegarde

Le problème est le même que ci-dessus, et on est évidement obligé d'avoir quatre instructions.

5. Exemples

5.a. FIFO avec moins de 256 octets et non réservable

Descripteur :

point-lect	point-ecr
compt-lect	compt-ecr
taille	

Procédure lire :

initialisation_2
CE_FIFO_lecture
accès_séquentiel_boucle_lecture
CS_FIFO_lecture
sauvegarde_2

5.b. FIFO avec plus de 256 octets et non réservable

Descripteur :

point-lect	point-ecr
nesg-lect	nseg-ecr
compt-lect	compt-ecr
taille	

Procédure lire :

initialisation_2
CE_pile_lecture_étendu
accès_séquentiel_décroissant_lecture
CS_pile_lecture
sauvegarde_2

5.c. Pile avec plus de 256 octets et réservable en lecture

Descripteur :

point-lect	point-ecr
nseg-lect	nseg-ecr
compt-lect	compt-ecr
taille	utilisateur

Procédure lire:

initialisation_2

test-réservé-lecture

CE-pile-lecture-étendu

accès-séquentiel-décroissant-lecture

CS-pile-lecture

sauvegarde_2

4.d. Mémoire de rafraichissement d'un écran de l'annexe 1 avec plus de 256 octets

Descripteur :

point-lect	point-ecr
nseg-lect	nseg-ecr
curseur-ligne	curseur-colonne
début	taille (1280)
N.de lignes(16)	N.de colonnes (80)

Procédure lire :

initialisation_3

accès-séquentiel-bouclé-étendu-lecture

sauvegarde_3

Procédure écrire :

initialisation_3

accès-séquentiel-bouclé-étendu-écriture

actualiser-curseur

sauvegarde_3

Pour pouvoir exécuter les algorithmes des procédures de la mémoire de rafraichissement, il faut rajouter des instructions intermédiaires.

Exemple : actualiser-curseur

nombre de cycles	opérateur algorithmique 8 bits	opérateur algorithmique 12 bits	opérateur binaire	mémoire commune	séquencemen
2	CCOLONNE:= CCOLONNE+1	X	X	X	SEQ1
2	'80'-CCOLONNE	X	test zéro	X	BR1
2	CCOLONNE:=0	X	X	X	SEQ1
2	CLOGNE:=CLIGNE+1	X	X	X	SEQ1
2	'16'-CLIGNE	X	test zéro	X	BR1
2	DEBUT:=DEBUT+1	X	X	X	SEQ1
2	CLIGNE:=0	X	X	X	SEQ2

CLIGNE ≡ CURSEUR-LIGNE ; CCOLONNE ≡ CURSEUR-COLONNE

Pour les autres procédures, il faut aussi introduire quelques instructions intermédiaires nouvelles :

Procédure RETOUR CHARIOT

initialisation_3
effacer-ligne
actualiser-curseur
sauvegarde_3

Procédure UP

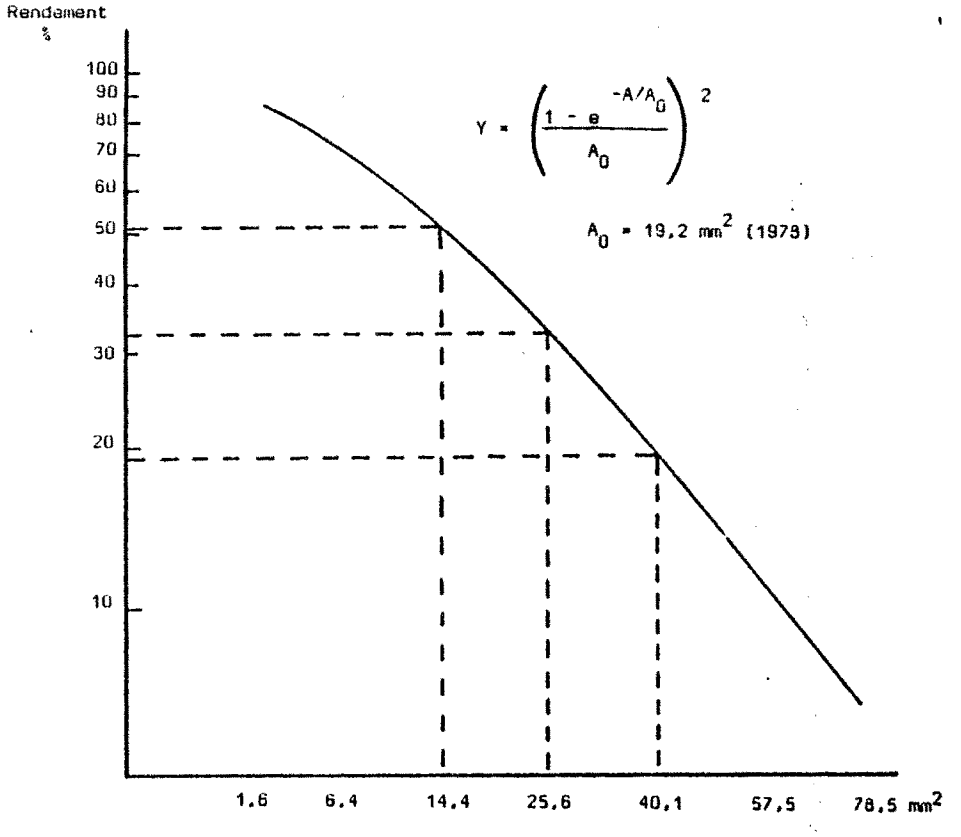
initialisation_3
décrémenter-curseur-ligne
sauvegarde_3

ANNEXE 3

EVALUATION DE SURFACE

EVALUATION DE SURFACE

L'évaluation de la surface est d'une importance primordiale à cause de son influence directe sur le rendement de production.



Pour évaluer d'une façon globale la surface d'un circuit, il existe deux méthodes classiques :

1 - Evaluer le nombre total de portes ou des transistors du circuit et le diviser par une densité moyenne obtenue par l'analyse de circuits du même type et fabriqués dans la même technologie.

Evidemment la valeur obtenue est très peu précise et ne peut être considérée que comme une indication de l'ordre de grandeur de la surface.

ex.: CMOS/SOS - SC1/SC2 - 180 MOS/mm2 (EFCIS)

Z80 $\underline{\sim}$ 122 portes/mm2 - Z8000 $\underline{\sim}$ 148 portes/mm2 - 68000 $\underline{\sim}$ 294 portes/mm2

Remarque : Le nombre de transistors ou portes annoncé par les fabricants est quelquefois sujet à des interprétations "commerciales" et donc peu précis surtout dans les circuits où il y a un pourcentage important de PLA ou de ROM.

2 - Décomposer le circuit dans les différents blocs qui le constituent et diviser le nombre de transistors ou portes de chacun d'eux par les densités moyennes par type de fonction.

La surface occupée par les interconnexions entre blocs est normalement calculée en multipliant la surface totale occupée par les blocs par un coefficient qui est fonction de la technologie

$$S_{bloc_1} = N^{o}_{transistor} / Densité_{fonction\ A}$$

$$S_{logique} = \sum_{i=1}^{n} S_{bloc_i}$$

$$S_{circuit} := S_{logique} + \underbrace{K \times S_{logique}}_{\text{interconnexion}} + \underbrace{S_{pourtour}}_{\text{plats d'E/S}}$$

K = coefficient pour les interconnexions dans la technologie utilisée

Exemple :

CMOS/SOS - SC_1 [Bra 78]

Logique répétitive (N transistors)	- N/800
Logique aléatoire (N transistors)	- N/580
RAM statique (M bits)	- M/100
ROM (M bits)	- M/1300
K (coefficient pour les interconnexions)	- 1

Cette méthode est beaucoup plus précise et fournit de bons résultats à condition de disposer des valeurs correctes des densités par type de fonction.

Le problème de l'obtention de ces densités nous oblige à séparer les circuits à évaluer en deux classes :

1 - circuits très complexes à implanter avec les technologies les plus

récentes ou quelque fois avec celle que l'on pense qui seront au point à la fin du processus de conception.

2 - Circuits de moyenne complexité réalisés dans une technologie bien connue pour laquelle les valeurs des densités par type de fonction ont été déterminés.

Le problème de l'évaluation de la première classe de circuits est assez compliqué car d'une part les fabricants diffusent très peu d'informations (règles de dessin, densités, etc...) sur les technologies de pointe et d'autre part il y a peu de circuits "observables" susceptibles de fournir des évaluations comparatives.

Il nous a semblé qu'une solution à ce problème est de faire l'évaluation de surface des blocs fonctionnels du circuit dans une technologie connue (NMOS-6µ) et ensuite de déterminer des coefficients de réduction par type de logique pour transposer les résultats dans une technologie plus poussée.

II - EVALUATION DE SURFACE EN NMOS-6µ

Pour la technologie NMOS-6µ il ne nous a pas été possible d'obtenir les tables de densités par fonction logique. Ayant cependant accès aux règle de dessin NMOS-LOCOS(SESCOSEM) pouvant mesurer la surface d'organes identiques sur des microprocesseurs fabriqués dans des technologies semblables, nous avons pu déterminer quelques paramètres de base.

On a donc utilisé une méthode d'évaluation mixte pas très rapide mais qui ne doit pas introduire de grosses erreurs dans le résultat final.

Pour évaluer la surface d'un bloc exécutant une fonction spécifique on peut utiliser trois méthodes :

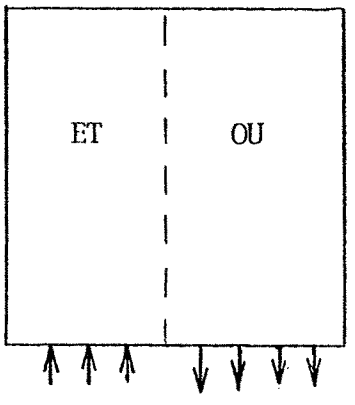
1 - Schémas à l'échelle 1000 dessinés en fonction des règles de dessin

A cause de l'investissement considérable en temps, nécessaire pour réaliser ces dessins, il n'est intéressant de les faire que pour des éléments très répétitifs, comme par exemple le point PLA, le point RAM à simple accès, le point RAM à double accès, le point registre.

Des formules très simples peuvent être établies ensuite pour calculer la taille d'un plan de PLA, ROM où RAM.

Exemple :

Un PLA classique :



avec un point

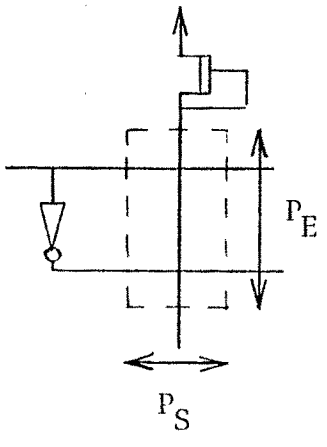


Figure 1

N_E : nombre d'entrées
N_S : nombre de sorties
N_m : nombre de monômes
$\lceil$: fonction minorant entier

$$S_{PLA} = [N_E \times P_E + (N_S + \left\lceil \frac{N_S}{8} \right\rceil) \times P_S] \times [(N_m + \left\lceil \frac{N_m}{8} \right\rceil) \times P_S]$$

Remarque : $\left\lceil \frac{N_S}{8} \right\rceil$ et $\left\lceil \frac{N_m}{8} \right\rceil$ correspondent à l'introduction des rappels de masse.

Dans une technologie 6µ, un point de PLA réalisé avec un contact 7µ × 5µ mesure $P_E = 37\mu$, $P_S = 20\mu m$

$$S = [N_E \times 37 + (N_S + \lceil \frac{N_S}{8} \rceil) \times 20] \times [(N_m + \lceil \frac{N_m}{8} \rceil) \times 20] \quad \mu m^2$$

Pour une RAM il faut tenir compte du nombre de mots, de leur taille et du degré de multiplexage.

Dans les dimensions du point RAM sont normallement déjà prises en compte les 2 bus, les alimentations et la ligne de sélection.

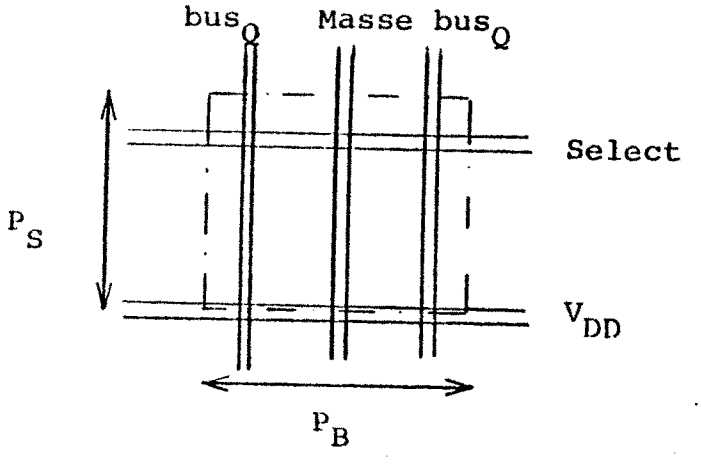


Figure 2

N_m : nombre de mots
T_m : Taille des mots
K : Degré de multiplexage

$$S = (\frac{N_m}{K} \times P_S) \times (T_m \times K \times P_B)$$

2 - Dessins symboliques

En utilisant un formalisme du genre "sticks" [WIL-77] on peut définir la structure globale de l'implantation d'une fonction logique.

Il faut ensuite déterminer les endroits critiques selon les deux dimensions et les évoluer d'une façon plus fine, en utilisant les règles de dessin.

Cette méthode donne de bons résultats surtout pour l'évolution de structures régulières (multiplexeurs, compteurs, etc...)

Exemple : Un registre de décalage dynamique avec une commande de remise à zéro.

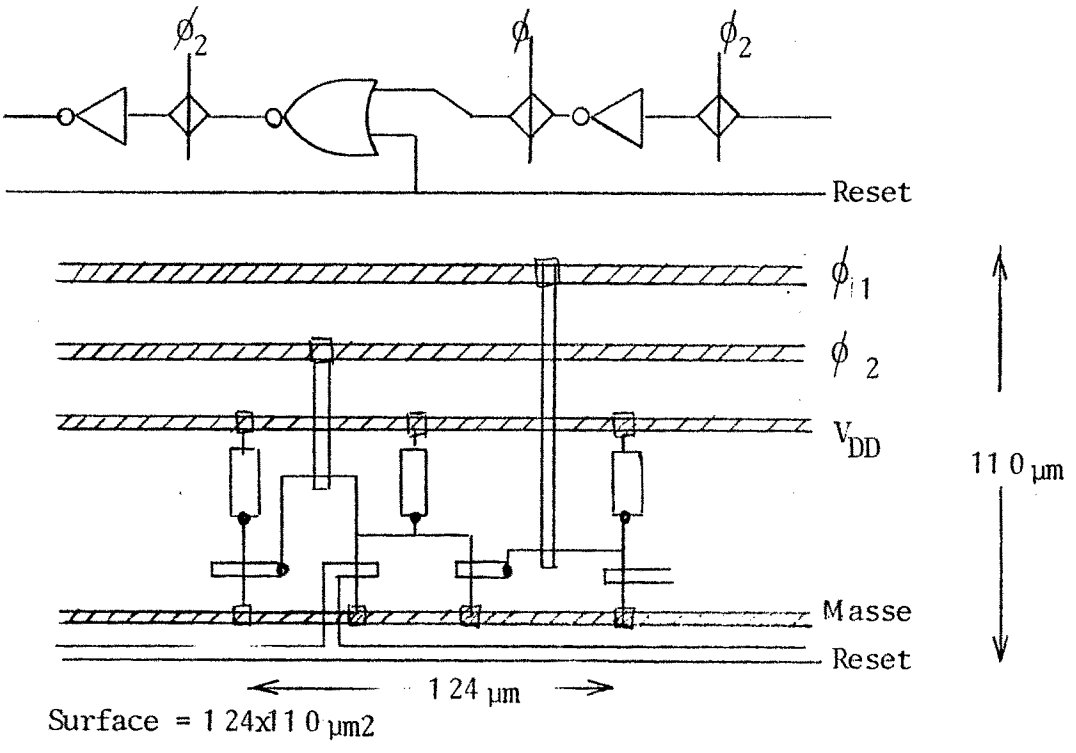


Figure 3

Remarque : Les stick diagrams" sont utilisés avec un code couleur, la limitation imposée par l'usage des photocopieuses noir et blanc nous a obligé à la modifier de la façon suivante :

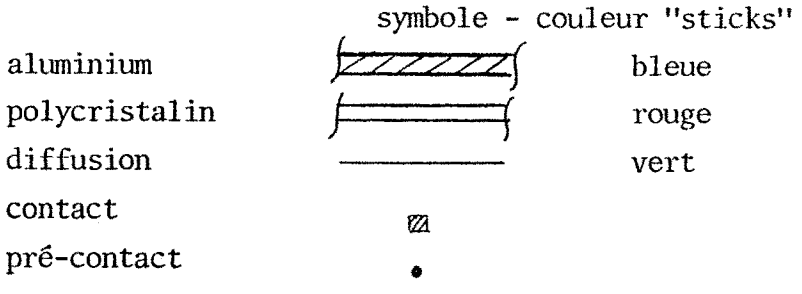


Figure 4

3) Mesure de la surface d'organes identiques existant sur des microprocesseurs fabriqués dans la même technologie ou dans des technologies semblabes.

A la fin de cette annexe on trouvera les fiches qui ont été faites pour les microprocesseurs les plus courants et des coefficients de normalisation de surface qui ont été calculés pour "ramener" chacun d'eux à la technologie 6µm.

III - COEFFICIENT DE REDUCTION DE LA SURFACE EN FONCTION DE LA TECHNOLOGIE

La valeur de ces coefficients est très dépendante du type d'échantillon sur lequel on les détermine (type d'implantation, répétitivité, etc...)

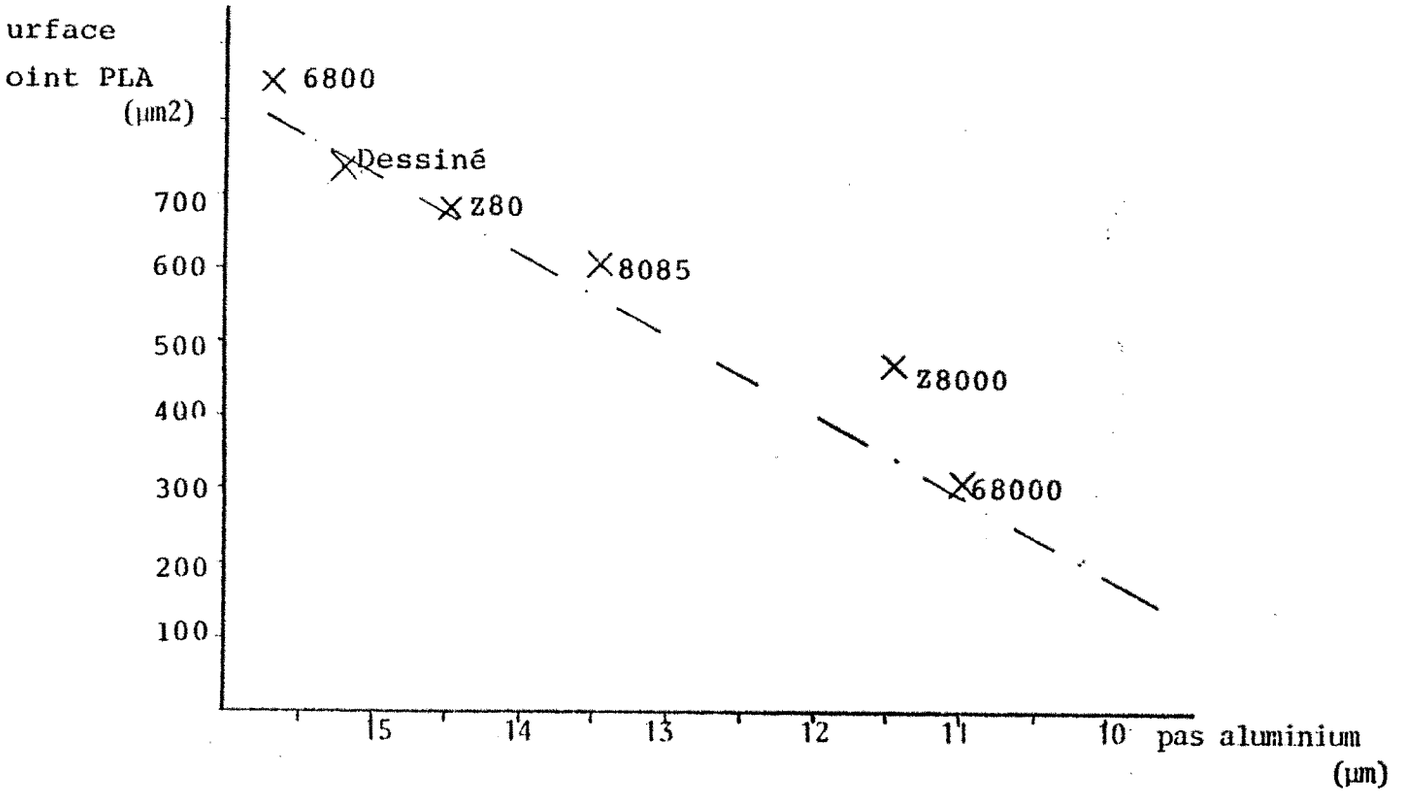
Dans les articles qui traitent ce sujet on donne normallement des densités globales [ALT-77, ANC-79, CAP-79, FAG-79] qui s'appliquent à tout le circuit.

Pour essayer d'avoir un peu plus de précision on a décidé d'établir 4 coefficients respectivement pour le point de PLA, le point de RAM statique, le point de RAM dynamique et pour la logique.

Remarque : Pour ce qui suit, les valeurs utilisées ont été obtenus par des mesures effectuées au micromètre ou publiées dans Electronics (77,78,79,80) et dans les proceedings de la Conférence ISSCC (79,80).

L'évolution de la technologie est normalement repérée par la longueur du canal. Dans le domaine des performances ceci est relativement correct car la diminution de la longueur canal entraîne une diminution presque linéaire des temps de commutation. Cependant, pour la réduction de surface, la diminution de la longueur du canal a une influence relativement restreinte si elle n'est pas accompagnée d'une réduction uniforme de tous les autres paramètres (aluminium, contacts, diffusion).

Le 8085 illustre ce qui vient d'être dit sur la réduction de surface car dans ce cas seules les grilles, ont été réduites, les autres paramètres étant à peu prêt inchangés par rapport à la technologie 6µ. D'ailleurs si l'on trace la même courbe en fonction du pas de l'aluminium (largeur Aluminium + distance entre Aluminium) les points se disposent d'une manière beaucoup plus linéaire.



La cellule du PLA est très simple et permet d'analyser finement l'influence sur la surface de la cellule de la largeur grille et de la largeur de l'Aluminium, ceci nous a permis de déduire l'équation suivante :

$$S_{\text{point PLA}} = 740 - 113 \times \Delta L_G - 98{,}6 \times \Delta L_{ALU} + 15{,}2 \times \Delta L_G \times \Delta L_{ALU}$$

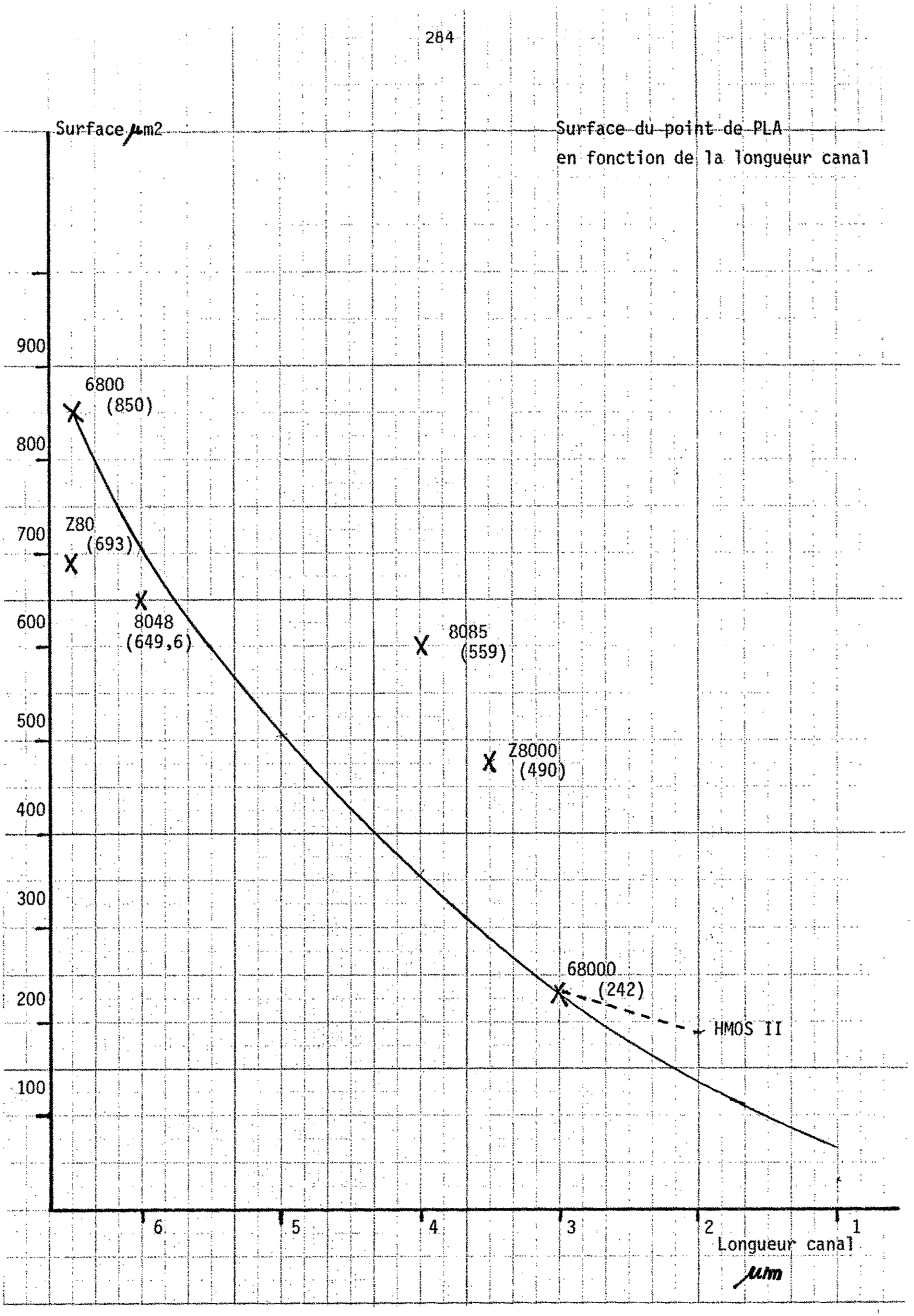
$\Delta L_G = 6 - L_G$

$\Delta L_{ALU} = 8 - L_{ALU}$

variation par rapport au 6µ.

Si on suppose que pour une technologie cohérente la variation de la largeur grille est identique à celle de l'aluminium on obtient l'équation suivante que l'on a tracée dans la figure 8.

$$S_{\text{Point PLA}} = 15{,}2\ L_G^2 + 25{,}2\ L_G + 17{,}6$$


Surface µm2
Surface du point de PLA
en fonction de la longueur canal
900
800
700
600
500
400
300
200
100
6800
(850)
Z80
(693)
8048
(649,6)
8085
(559)
Z8000
(490)
68000
(242)
HMOS II
6
5
4
3
2
1
Longueur canal
µm

Pour une technologie intermédiaire (diminution de la seule longueur du canal) on obtient :

$S = S' - 12 \times (L'_G - L_G)$

S' - surface du point de PLA dans une technologie cohérente

L'_G - largeur grille dans cette technologie

L_G - largeur grille dans la technologie intermédiaire

Les coefficients de réduction correspondent aux technologies de pointes actuelles et à celles en perspective sont :

Surface du point PLA	Technologie NMOS	Réduction par rapport au 6 µ
740 μ^2	6 µ	-
270 μ^2	HMOS	1/3,08
210 μ^2	HMOS II(2µ) pas de réduction ALU	1/3,52
136 μ^2	2µ - réduction ALU	1/5,44
62 μ^2	1µ - réduction ALU	1/12

Remarque : Dans la technologie HMOS la grille des transistors peut être réduite à 3µ, c'est le cas dans le 68000 où le point PLA à une surface de 240 μm^2.

RAM Statique

Pour les RAM statiques on présente deux courbes une pour les points de mémoire à 6 transistors et une autre pour des points de mémoire à 4 transistors avec des charges en Poly

Les courbes correspondantes à des approximations par des polynômes de 2ème degré des valeurs observées en fonction de la largeur grille.

On a calculé les facteurs de réduction par rapport à une cellule dessinée en utilisant les règles de dessin 6 µm(60µm× 55µm).

Surface point RAM statique 6T	Technologie	Coefficient de réduction par rapport au 6µ .
3300	6µ	--
2130	HMOS(3,5)	1/1,55
1902	HMOS II	1/1,74
1500	2µ	1/2,2
1048	1µ	1/3,14

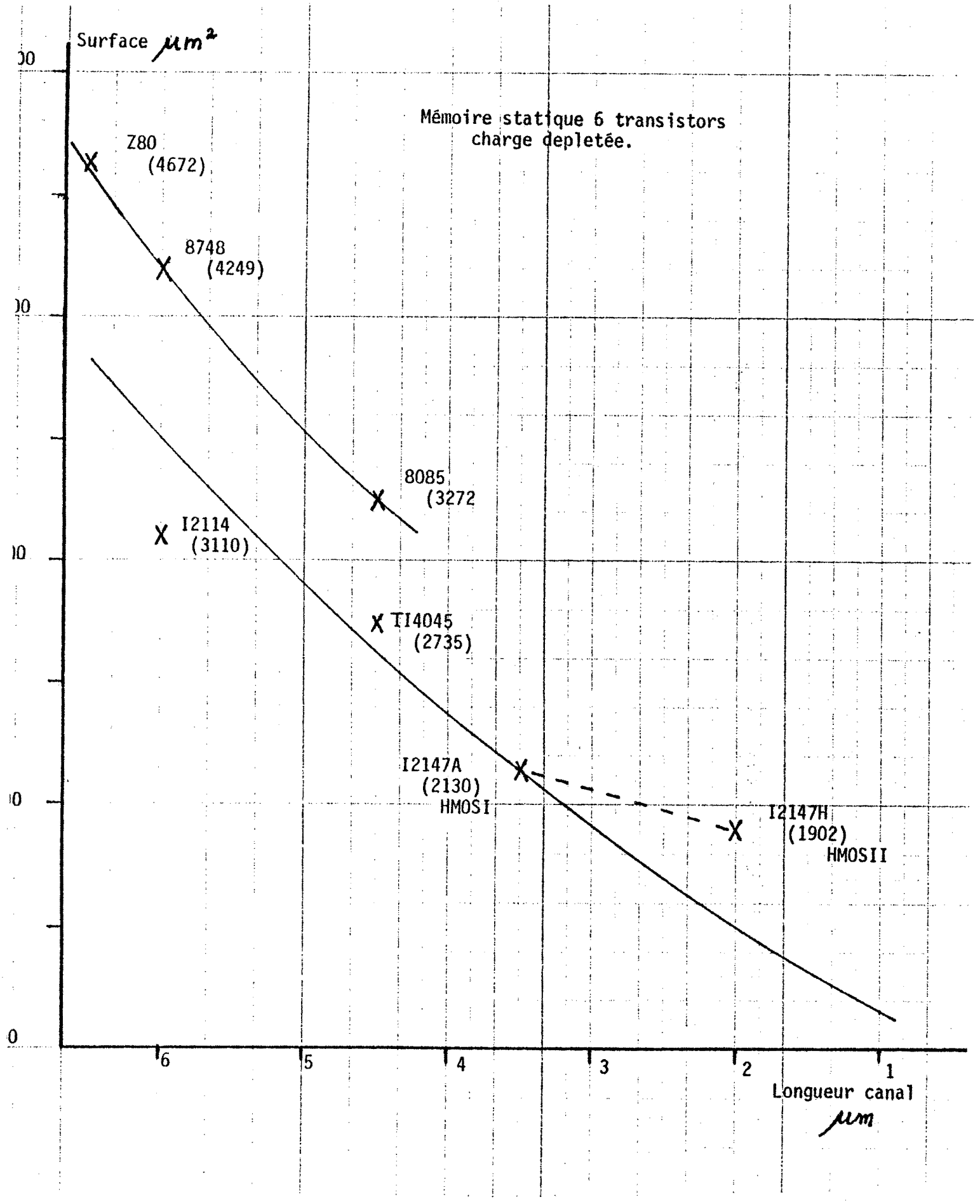
Les cellules à 4T supposent l'emploi d'une technologie avec du Poly très résistif. Jusqu'à maintenant on n'a pas constaté l'utilisation de cette technologie sur des microordinateurs standard (8048, 6801), probablement à cause du coût élevé de ces technologies compte-tenu de la faible proportion de RAM sur ces circuits. Cependant, si l'on suppose que dans les circuits futurs du type Mosaïc la RAM représentera une surface très importante l'utilisation de cette technologie sera peut-être justifiée.

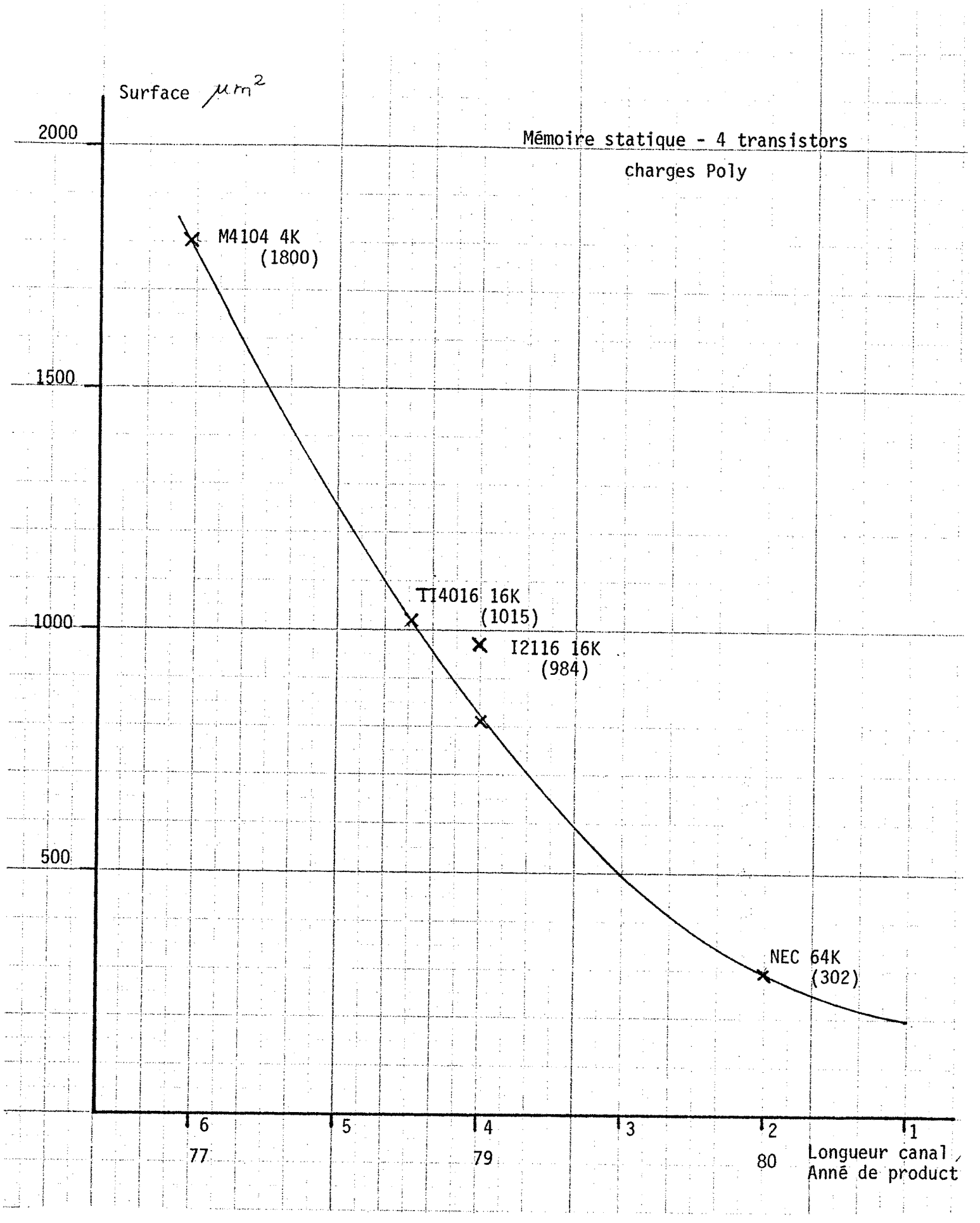
Surface Point RAM statique 4T	Technologie	Réduction par rapport au 6 µ	Réduction par rapport au 6µ à 6T
1800 μ^2	6 µ	-	1/1,83
655 μ^2	HMOS(3,5)	1/2,75	1/5,04
302 μ^2	2 µ	1/5,96	1/10,92
210 μ^2	1 µ	1/8,57	1/15,75

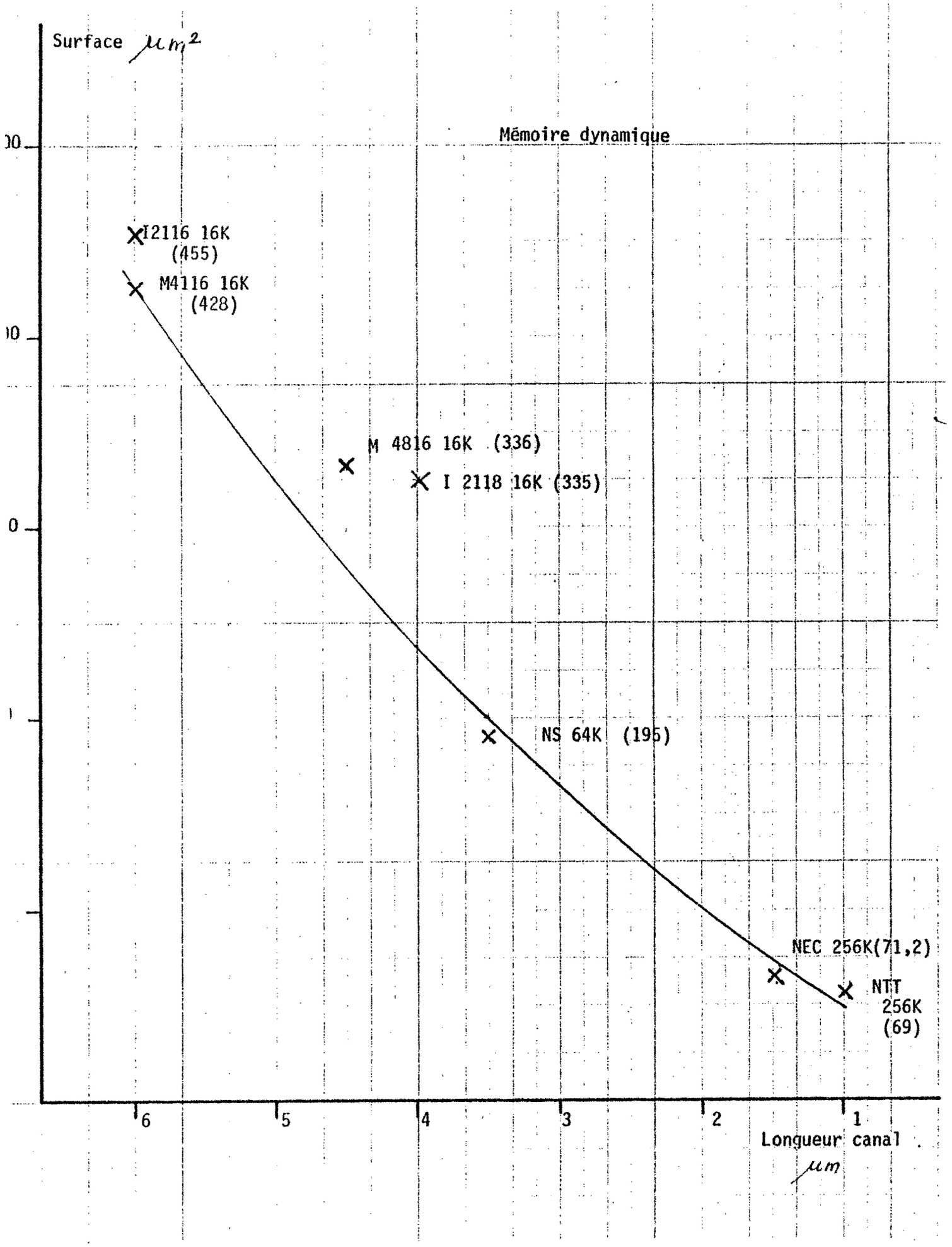
on n'a pas de données pour calculer le point correspondant à la technologie HMOS II

RAM Dynamique

On a procédé de la même façon que pour les points statiques.


Surface µm²
Mémoire statique 6 transistors
charge depletée.
Z80
(4672)
8748
(4249)
8085
(3272
I2114
(3110)
TI4045
(2735)
I2147A
(2130)
HMOSI
I2147H
(1902)
HMOSII
0
6
5
4
3
2
1
Longueur canal
µm


Surface µm²
2000
1500
1000
500
Mémoire statique - 4 transistors
charges Poly
M4104 4K
(1800)
TI4016 16K
(1015)
I2116 16K
(984)
NEC 64K
(302)
6
5
4
3
2
1
77
79
80
Longueur canal
Année de product


Surface μm^2
Mémoire dynamique
I2116 16K (455)
M4116 16K (428)
M 4816 16K (336)
I 2118 16K (335)
NS 64K (195)
NEC 256K(71,2)
NTT 256K (69)
6
5
4
3
2
1
Longueur canal μm

Surface Point RAM Dynamique	Technologie	coefficient de réduction par rapport au 6µ.
482 μ^2	6 µ	-
195 μ^2	3,5 µ	1/2,19
105 μ^2	2 µ	1/4
70 μ^2	1 µ	1/6,11

Logique

Pour les fonctions logiques (UAL, multiplexeurs, compteurs, amplificateurs bus, etc...) il est beaucoup plus difficile d'obtenir des chiffres comparables car leur implémentation est souvent différente.

Si l'on considère que la taille d'un bloc est imposée par un ensemble ordonné de rectangles (tous les niveaux de masques confondus)

$$S = \sum_{i=1}^{\gamma} a_i \times \sum_{i=1}^{\ell} b_i ;$$

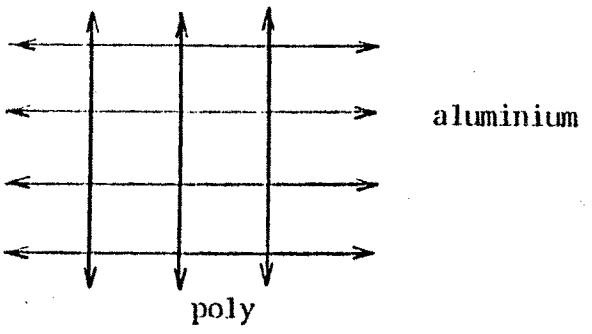
Si une réduction homogène des côtés de tous les rectangles se produisait on obtiendrait :

$$S^1 = \sum_{i=1}^{\gamma} \frac{a_i}{K} \times \sum_{i=1}^{\ell} \frac{b_i}{K} = \frac{S}{K^2}$$

Cependant il n'existe pas un coefficient homogène de réduction K pour tous les paramètres, en particulier pour le pas du polycristallin et de l'aluminium.

On peut simplifier le problème en considérant l'hypothèse suivante qui correspond à une philosophie de conception.

On considère que dans un bloc les lignes d'aluminium sont disposée perpendiculairement aux lignes de polycristallin. La surface étant déterminée par le pas minimum de l'aluminium dans une direction et par le pas minimum des diffusions et des polycristalins dans l'autre.



Dans ce cas on peut considérer deux coefficients :

K_1 : coefficient de réduction du pas de diffusion ou du polycristall
K_2 : coefficient de réduction du pas de l'aluminium.

$$S^1 = \frac{S}{K_1 \times K_2}$$

Remarque : Selon la théorie du maigrissement de la technologie la réduction du pas de diffusion et du pas du polycristallin est identique.

Pour le HMOSI les coefficients sont les suivants :

	MOS 6µ	Pas HMOS	Coefficient
Aluminium	15 µm	11 µm	K_1 = 1/1,36
Polycristallin	11 µm	7,5 µm	K_2 = 1/1,46

Donc :

$$S^1 = \frac{S}{1,96}$$

Pour valider ce coefficient on a effectué des mesures de densités sur une tranche d'un bit de l'UAL

6800 - 721 MOS/mm^2

8085 - 784 MOS/mm^2

68000 - 1512 MOS/mm^2

Le rapport trouvé est de 2,08 ce qui ne s'éloigne pas trop de la valeur calculée précédemment.

Pour calculer les coefficients pour des technologies plus poussées on a extrapolé les réductions du pas de polycristallin et de l'aluminium de la façon suivante :

Polycristallin	Technologie	largeur	distance entre 2 lignes	Pas
	6µ	6	5	11
	HMOS	3,5	4	7,5
	2µ	2	3	5
	1µ	1	2	3

Aluminium

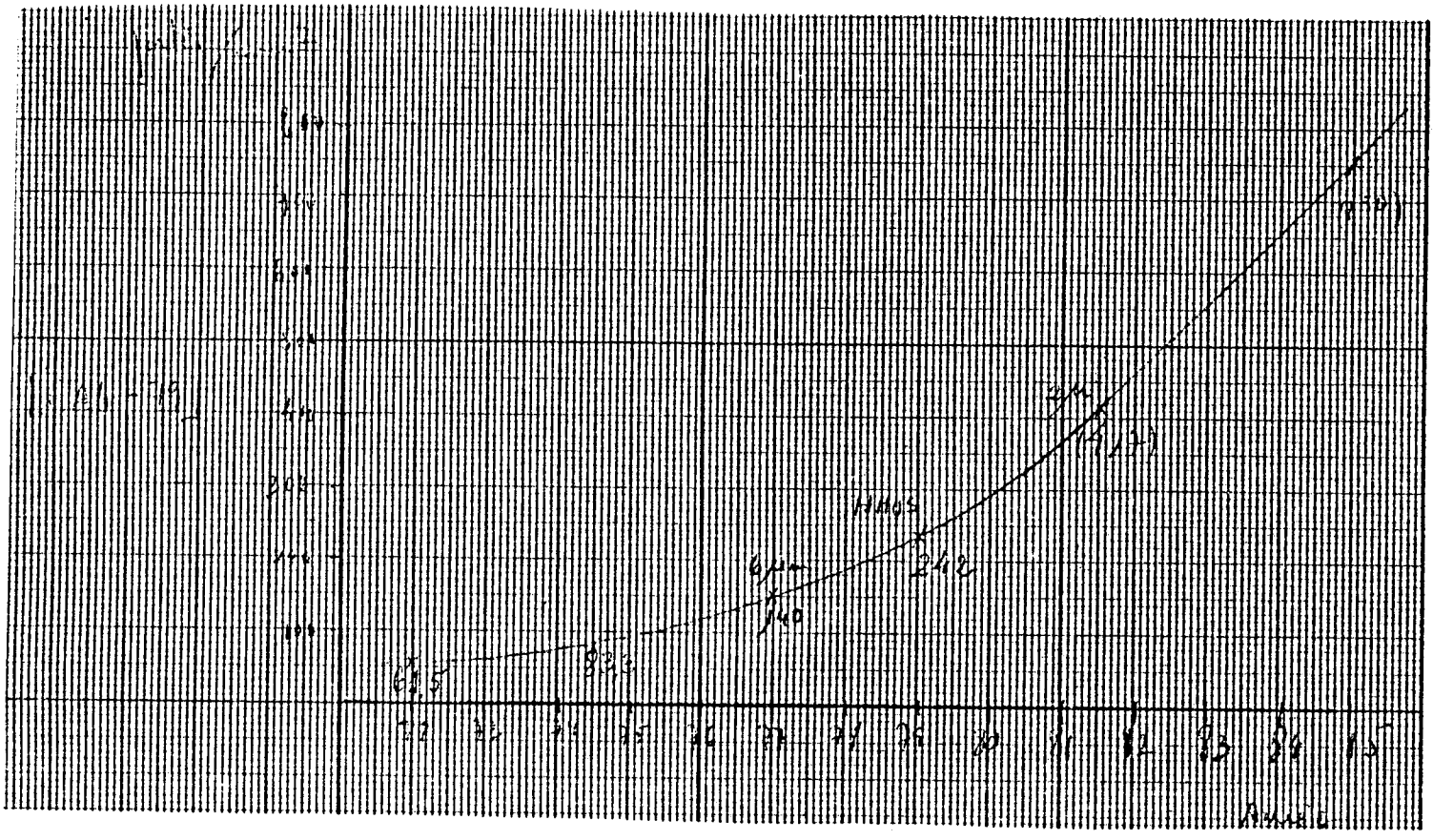
	Technologie	largeur	distance entre 2 lignes	Pas
	6µ	8	7	15
	HMOS	5	6	11
	2µ	3,5	5	8,5
	1µ	2,5	4	6,5

Coefficients de réduction

Technologies homogènes	K_1	K_2	$K_1 \times K_2$
HMOS	1,36	1,46	1,96
2µ	1,76	2,2	3,88
1µ	2,3	3,66	8,45

Il est bien évident que ces coefficients sont des estimations car d'une part le maigrissement théorique de la technologie ne se répercute pas d'une façon homogène sur tous les paramètres (en particulier les contacts) et d'autre part des problèmes électriques risquent d'invalider l'hypothèse de réduction homotétique que l'on a pris au début.

L'analyse des densités globales des microprocesseurs permet aussi de calculer des coefficients de réduction.



Entre le Z80 (1977-6) et un produit HMOS le coefficient est de 1,73
Pour un produit 2µm on obtiendrait un coefficient de 2,98.

Les valeurs obtenues sont donc inférieures à celles déterminées précédemment, ce qui peut être justifié par le fait de baser le calcul de la densité dans la surface totale du circuit y inclus les plots d'E/S et le chemin de découpe dont la surface ne se réduit pas avec l'évolution de la technologie.

INTEL 8085

Année de fabrication : 1977 - 1er trimestre
Surface : 4,17 mn × 5,64 = 23,51 mm^2
N° de transistors ≃ 6200
Densité ≃ 264 T/mm^2

Technologie : NMOS à charge deplétée

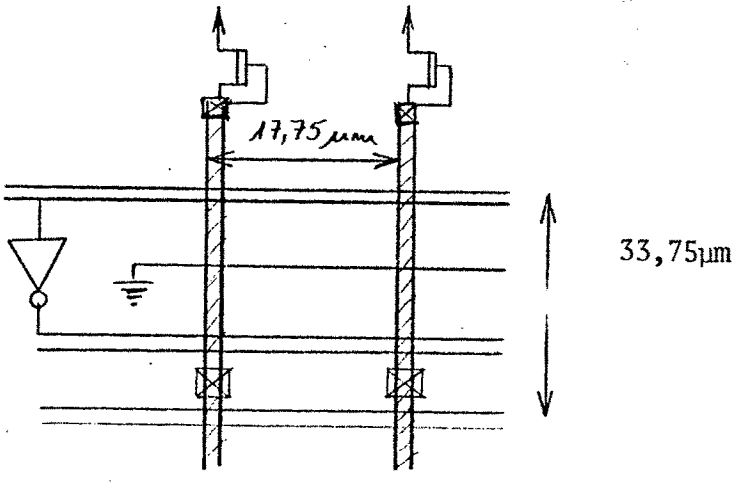
Poly utilisé en connexion bus ≃ 5,8µ
Poly grille UAL et RAM ≃ 4,25 µ
Poly grilles PLA - 3,6 à 3,8 µ
Pas aluminium - 13,5 µ
Largeur diffusion PLA ≃ 4,625 µ

Les écarts par rapport aux règles de conception 6 µ sont

Poly grilles (UAL, RAM)	≃	- 1,75 µ
Poly grille (PLA)	≃	- 2,3 µ
Pas aluminium	≃	- 1,5 µ

Il semble donc qu'il s'agisse d'une technologie intermédiaire où seulement les grilles ont été considérablement réduites. Il est aussi intéressant de remarquer que dans le PLA on trouve des grilles plus petites que dans le reste du circuit probablement pour améliorer le temps de positionnement.

Point PLA

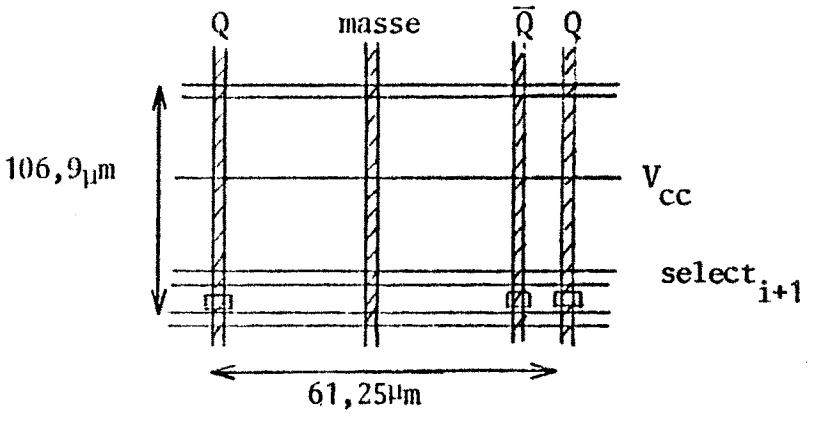


$$S = 33,75 \times 17,75 \simeq 559\ \mu^2$$

Par rapport au point dessiné avec des règles de 6µ

$$\frac{S_{8085}}{S_{6\mu}} = 1/1,32$$

Point de RAM



Par point $53,43 \times 61,25 = 3272,6\ \mu^2 \simeq 3273\ \mu^2$

Logique :

Comme il a été déjà dit la technologie du 8085 est une technologie intermédiaire, on peut cependant considérer comme étant en moyenne réalisé dans une technologie de 5µ .
(Diffusion ≃ 5µ; Pas aluminium ≃ Pas $\text{Aluminium}_{6\mu}$ - 2µ ;
Largeur grille ≃ Lgrille 6µ - 1,5).

En utilisant la formule précédante on obtient

$S_{8085}/S_{6\mu} = 1/1,3$

Ce paramètre sera utilisé pour transposer en technologie 6 µ la surface des organes du 8085.

Z80

Année de fabrication : 1976 - 3ème trimestre
Surface : 4,54 mm × 4,88 mm = 22,2 mm^2 (Z80A) / 27,4 mm^2 (Z80)
N° de transistor : 8200
Densité : 122 portes /mm^2 ou 369 T/mm^2 (Z80A)

Technologie :

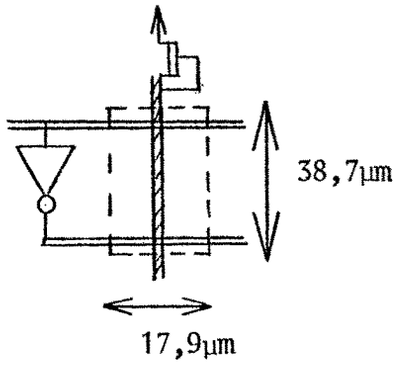
Grilles Poly ≃ 6,5 µ
Diffusion ≃ 6,5 µ
Pas aluminium ≃ 14,3 µ

Ecarts par rapport aux règles de conception 6 µ

Δ Poly ≃ + 0,5 µ
Δ Diffusion ≃ + 0,5 µ
Δ Pas aluminium ≃ - 0,7 µ

Technologie 7/6µ classique des valeurs très semblables ont été mesurées sur le 6800, le seul paramètre qui semble avoir été réduit est le pas de l'aluminium.

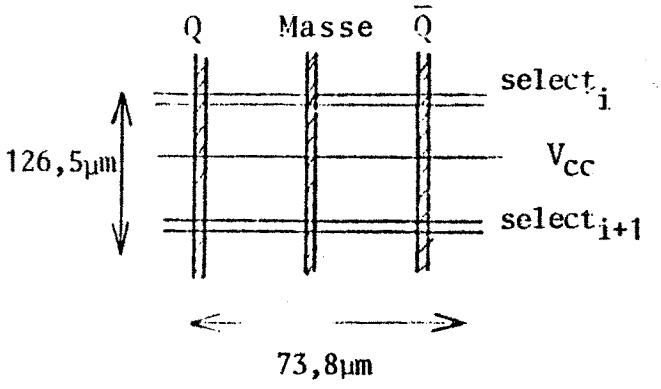
Point PLA



$S = 38,7 \times 17,9 = 692,7\ \mu^2 = 693\ \mu^2$

Coefficient de réduction par rapport au 6µ dessiné = 1/1,07

Point RAM



$$S = 63,3 \times 73,8 = 4671,5\ \mu^2 \approx 4672\ ^2$$

Logique :

La technologie étant identique à celle du 6800 on utilise le coefficient unité pour la transposition.

INTEL 8748

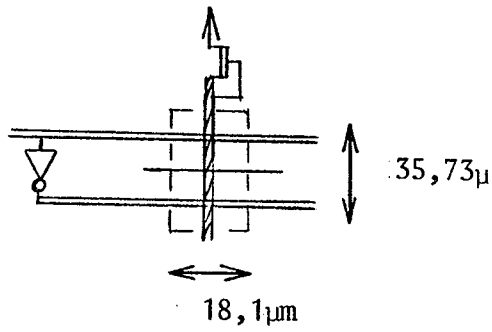
Date de fabrication : 1976, 4ème trimestre
Surface : $5,6 \times 6,6 \simeq 36,96$ mm^2
N° de transistor $\simeq$ 20 000
Densité : 541 T/mm^2 (pourcentage très élevée de ROM et RAM)

Technologie :

Pas aluminium = 13,8µ à 14,2 µ
Poly grilles $\simeq$ 6 µ

La technologie est très semblabe à celle décrite dans les règles de conception 6µ [SES-77] a l'exception du pas de l'aluminium qui semble être plus petit.

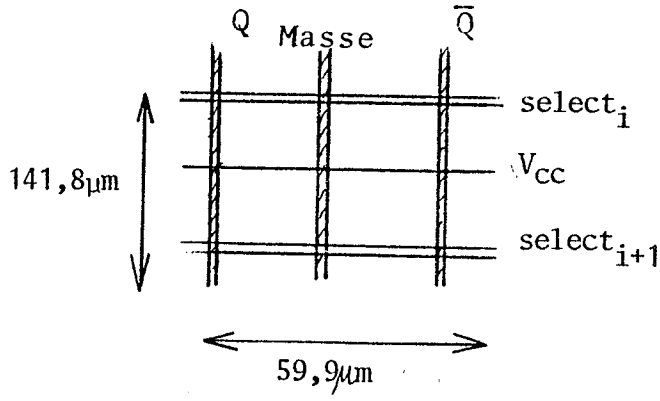
Point PLA



$S = 35,73 \times 18,11 \simeq 648\mu^2$

Réduction par rapport à la technologie 6µ = 1/1,14

RAM



$70,9 \times 59,9 \simeq 4\ 247\ \mu^2$

Z 8000

Date de fabrication : 1978, 3ème trimestre
Surface : 238 × 256 mil^2 ≃ 6,03 × 6,48 ≃ 38,9 mm^2
N° de transistors = 17 000
Densité : 148 portes/mm^2

Technologie :

Poly grille (PLA) ≃ 3,5 µ
Poly grille (RAM) ≃ 4 µ
Pas aluminium ≃ 11,4 µ

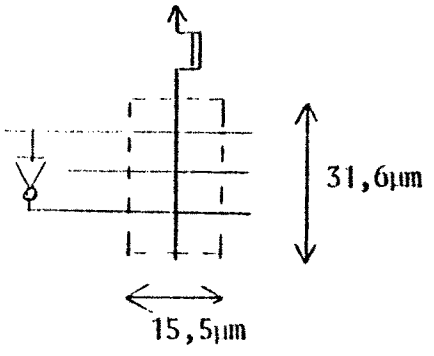
Ecarts par rapport aux règles de conception 6 µm

pas aluminium ≃ - 3,6 µ
poly grille PLA ≃ - 2 µ
poly grille RAM ≃ - 1,4 µ

Il s'agit d'une technologie où les éléments actifs ont été réduits en moyenne de 2µ à 1,5µ, on peut donc le classer dans des règles 4,5µ.

Point PLA

PLA



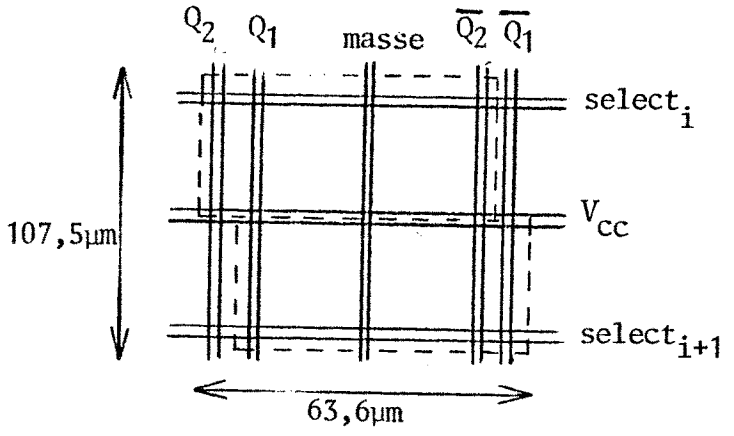
Remarque : Dans le Z8000 il n'y a pas une vraie matrice PLA. Horizontalement, il y a des lignes brisées et verticalement le pas dépend de l'impédance attaquée par chaque sortie.

$S = 15,5 \times 31,6 = 489,9\mu^2$

$Coef_{Point\ du\ PLA} = 1/1,51$

RAM

Accès simple mais avec 2 bus connectés en alternance



$S = 53,75 \times 63,6 = 3547,5\ \mu m^2$

Logique :

Si on considère le Z8000 fabriqué avec des règles 4,5µ le coefficient de réduction par rapport au 6µ est de

$$S_{6\mu}/S_{Z8000} = 1/1,6$$

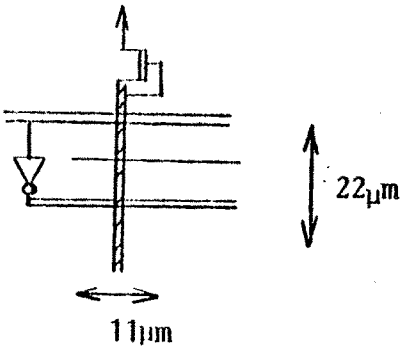
MOTOROLA 68000

Année de fabrication : 1979
Surface :6,27 mm × 7,11 mm ≈ 44,29 mm^2
Nombre de transistors : 68000 (grand pourcentage de ROM)
Nombre de portes : 13 000
Densité : 294 portes /mm^2

Technologie HMOS

Poly grille (PLA)	- 3µ à 3,2µ
(UAL)	- 3,5µ
Pas aluminium	- 11 µ

Point PLA

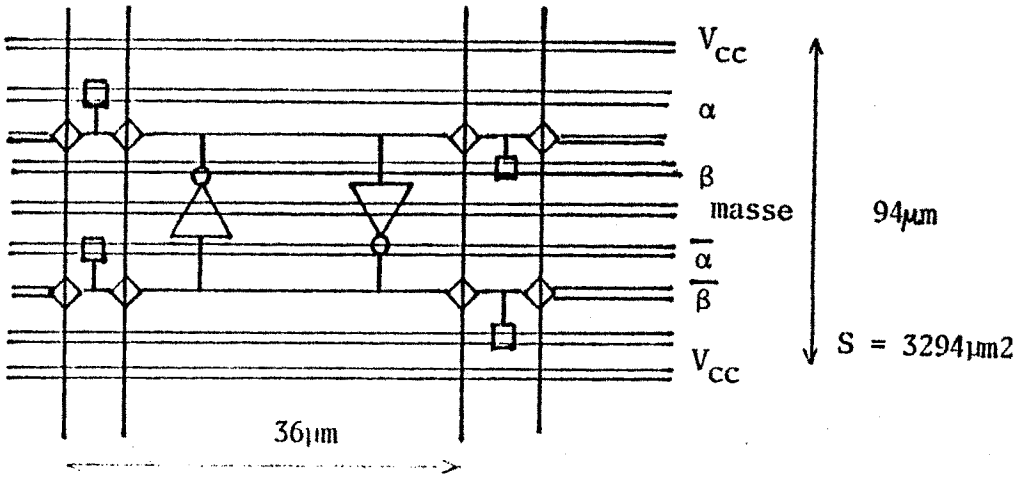


Les grilles des transistors du PLA on des tailles inférieures ≈ 3 µ que celles utilisées pour la logique.

S = 22 × 11 = 242

$Coef_{Point\ du\ PLA}$ = 1/3,16

RAM



A N N E X E 4

SIMULATION DE PERFORMANCES

Programme 1

Ce programme QNAP correspond au modèle de simulation du circuit Antiope proposé au § 6.3. Dans ce modèle, on considère qu'il n'y a aucun tampon, ni dans le processus de réception, ni dans le processus de visualisation.

Processus de réception :

station PREC - génération des caractères avec une fréquence constante

station IPR - interface avec les mécanismes de communication

Processus de visualisation :

station VISU - génération des caractères avec une fréquence constante pendant l'affichage d'une ligne

station IPV - interface avec les mécanismes de communication

Processus d'interprétation

station PTRAI - loi de traitement exponentiel de paramètre TTRAI = 50X temps de cycle.

```
* QUEUING NETWORK ANALYSIS PACKAGE (25-8-79) ***

 1     /DECLAR/ QUEUE BUS,PREC,VISU,PTRAI,IPR,IPV;
 2              INTEGER COL,TRANCHE,LIGNE,COL2,TRANCHE2,LIGNE2;
 3              REAL TLM,TLM1,TLM2,TLM3,TEM,TETR,TLTR,TREC,TCAR,TLIG,TPG,TTRAI,TI
 4     /STATION/NAME=BUS;
 5              TYPE = RESOURCE;
 6              SCHED = PRIOR;
 7            &
 8            & PROCESSUS DE RECEPTION TV
 9            &
10     /STATION/NAME=PREC;
11              TYPE = GEN;
12              SERVICE = BEGIN
13              CST(TREC);
14              END;
15              TRANSIT=IPR;
16     /STATION/NAME=IPR;
17              SERVICE =
18              BEGIN
19                   PRIOR:=2;
20                   P(BUS);CST(TETR);V(BUS);
21              END;
22              TRANSIT = PTRAI;
23            &
24            & PROCESSUS DE VISUALISATION
25            &
26     /STATION/NAME=IPV;
27              SERVICE=
28              BEGIN
29                        PRIOR:=1;
30                        P(BUS);CST(TLM);COL2:=COL2+1;
31                        IF COL2 > 40 THEN
32                        BEGIN
33                             CST(TLM1); COL2:=0;  &  CHANGEMENT TRANCHE
34                             TRANCHE2:=TRANCHE2+1;
35                             IF TRANCHE2>10 THEN
36                             BEGIN
37                                  CST(TLM2);TRANCHE2:=0; &CHANGEMENT LIGNE
38                                  LIGNE2:=LIGNE2+1;
39                                  IF LIGNE2 > 25 THEN
40                                  BEGIN
41                                       LIGNE2:=0;
42                                       CST(TLM3);
43                                  END;
44                             END;
45                        END;
46              V(BUS);
47              END;
48              TRANSIT=OUT;
```

```
/STATION/NAME=VISU;
        TYPE=GEN;
        SERVICE=BEGIN
                CST(TCAR);COL:=COL+1;
                IF COL>40 THEN
                        BEGIN
                             CST(TLIG);
                             COL := 0;
                             TRANCHE:=TRANCHE+1;
                             IF TRANCHE = 10 THEN
                             BEGIN
                                  CST(40*TCAR+TLIG);
                                  LIGNE:=LIGNE+1;
                                  TRANCHE := 0;
                                  IF LIGNE=25 THEN
                                  BEGIN
                                       LIGNE:=0;
                                       CST(TPG);
                                   END;
                             END;
                        END;
                END;
        TRANSIT = IPV;
         &
         & PROCESSUS D' INTERPRETATION
         &
/STATION/NAME=PTRAI;
        SERVICE = BEGIN
                PRIOR := 0;
                P(BUS);CST(TLTR);V(BUS);
                EXP(TTRAI);
                P(BUS);CST(TEM);V(BUS);
                  END;
        TRANSIT = OUT;
/METHOD/IMAX=400;
/EXEC/BEGIN
      PRINT (" SIMULATIONS EN FONCTION DU T. CYCLE");
      PRINT (" MODELE 1 - PAS DE TAMPON DANS LES PROCESSUS ");
      TMCY := 0.050;
       WHILE TMCY < 0.10 DO
       BEGIN
            TMCY:=TMCY+0.025;
            LIGNE:=0;COL:=0;TRANCHE:=0;
             COL2:=0;TRANCHE2:=0;LIGNE2:=0;
             TLM:= 7 * TMCY;TLM1:= 4 * TMCY;TLM2:= 4 * TMCY;TLM3:= 2 *TMCY
             TEM:= 7 * TMCY;TETR:= 8 * TMCY;TLTR:= 9 * TMCY;
           TREC:=1.3;
          TTRAI:=50*TMCY;TCAR:=1.185;TLIG:=12;TPG:=2000;
      PRINT (" TMCY = ",TMCY);
      SIMUL;
      END;
```

```
100          END;
```

SIMULATIONS EN FONCTION DU T. CYCLE
MODELE 1 - PAS DE TAMPON DANS LES PROCESSUS

TMCY = .7500E-01 µs

```
*** SIMULATION BEGINS ... ***
***TIME =       400.00
*************************************************************************
* NAME *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *SERV NB*
*************************************************************************
*      *          *          *          *          *          *      *
* BUS  * .5721    * .9625    * 2.025    * 1.204    * 1.680    *    672*
*      *          *          *          *          *          *      *
* RREC * 1.300    * 1.000    * 1.000    * 1.300    * .7675    *    307*
*      *          *          *          *          *          *      *
* VISU * 1.446    * 1.000    * 1.000    * 1.446    * .0900    *    276*
*      *          *          *          *          *          *      *
* RTKA * 8.712    * .9942    * 131.0    * 176.8    * .1125    *     45*
*      *          *          *          *          *          *      *
* IPK  * .8857    * .6785    * .6783    * .8857    * .7650    *    306*
*      *          *          *          *          *          *      *
* IPV  * 1.132    * .7797    * 1.429    * 4.075    * .0875    *    275*
*      *          *          *          *          *          *      *
*************************************************************************
SPACE USED :  38/100
*** END OF SIMULATION : TIME =       400.00
```

TMCY = .10000 µs

```
*** SIMULATION BEGINS ... ***
***TIME =       400.00
*************************************************************************
* NAME *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *SERV NB*
*************************************************************************
*      *          *          *          *          *          *      *
* BUS  * .7545    * .9970    * 2.985    * 1.506    * 1.320    *    528*
*      *          *          *          *          *          *      *
* RREC * 1.300    * 1.000    * 1.000    * 1.300    * .7675    *    307*
*      *          *          *          *          *          *      *
* VISU * 1.446    * 1.000    * 1.000    * 1.446    * .0900    *    276*
*      *          *          *          *          *          *      *
* IPK  * 1.509    * .9967    * 22.03    * 68.79    * .0600    *    264*
*      *          *          *          *          *          *      *
* IPV  * 1.505    * .9949    * 6.083    * 10.09    * .0600    *    264*
*      *          *          *          *          *          *      *
*************************************************************************
SPACE USED :  44/100
*** END OF SIMULATION : TIME =       400.00
```

Programme 2

On considère un tampon dans le processus de réception. Le modèle est le même. La seule différence est que le temps de génération des caractères dans le processus de réception est le double de celui de l'exemple précédent.

TEMPS DE CYCLE = 75 NS

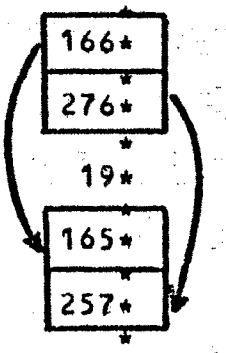
```
***TIME =       400.00
*************************************************************************
* NAME *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *SERV NB*
*************************************************************************
*      *          *          *          *          *          *       *
* BUS  * .5192    * .0866    * .9483    * .7171    * 1.322    *     529*
*      *          *          *          *          *          *       *
* RREC * 2.400    * 1.000    * 1.000    * 4.400    * .4150    *     166*
*      *          *          *          *          *          *       *
* VISU * 1.446    * 1.000    * 1.000    * 1.446    * .0900    *     276*
*      *          *          *          *          *          *       *
* RTRA * 9.056    * .9914    * 64.29    * 176.8    * .1075    *      43*
*      *          *          *          *          *          *       *
* IPR  * .7375    * .5061    * .5061    * .7375    * .4150    *     166*
*      *          *          *          *          *          *       *
* IPV  * .0345    * .4578    * .4511    * .6537    * .0900    *     276*
*      *          *          *          *          *          *       *
*************************************************************************
SPACE USED :  22/100
*** END OF SIMULATION ; TIME =       400.00
```

TMCY = .10000 µs

```
*** SIMULATION BEGINS ... ***
***TIME =       400.00
*************************************************************************
* NAME *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *SERV NB*
*************************************************************************
*      *          *          *          *          *          *       *
* BUS  * .0889    * .8578    * 1.588    * 1.112    * 1.245    *     498*
*      *          *          *          *          *          *       *
* RREC * 2.400    * 1.000    * 1.000    * 4.400    * .4150    *     166*
*      *          *          *          *          *          *       *
* VISU * 1.446    * 1.000    * 1.000    * 1.446    * .0900    *     276*
*      *          *          *          *          *          *       *
* RTRA * 15.61    * .9906    * 68.90    * 176.7    * .2506E-01*      29*
*      *          *          *          *          *          *       *
* IPR  * 1.041    * .4521    * .4521    * 1.041    * .4150    *     166*
*      *          *          *          *          *          *       *
* IPV  * .9974    * .0833    * .9895    * 1.435    * .0850    *     274*
*      *          *          *          *          *          *       *
*************************************************************************
SPACE USED :  24/100
*** END OF SIMULATION : TIME =       400.00
```

'MCY = .1250μS

```
* SIMULATION BEGINS ... ***
*TIME =       400.00
*************************************************************************
NAME *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *SERV NB*
*************************************************************************
     *          *          *          *          *          *      *
BUS  * .8622    * .9959    * 1.875    * 1.622    * 1.152    *   461*
     *          *          *          *          *          *      *
PREC * 2.400    * 1.000    * 1.000    * 2.400    * .4150    *   166*
     *          *          *          *          *          *      *
VISU * 1.440    * 1.000    * 1.000    * 1.446    * .9900    *   276*
     *          *          *          *          *          *      *
PTRA * 18.56    * .9897    * 72.14    * 162.0    * .4750E-01*    19*
     *          *          *          *          *          *      *
IPR  * 1.452    * .5947    * .5947    * 1.452    * .4125    *   165*
     *          *          *          *          *          *      *
IPV  * 1.545    * .9949    * 12.70    * 18.91    * .0425    *   257*
     *          *          *          *          *          *      *
*************************************************************************
'ACE USED :  27/100
* END OF SIMULATION : TIME =       400.00
```



Programme 3

Ce programme correspond au troisième modèle proposé au § 6.3.
Nous considérons un tampon de 40 caractères dans le processus de visualisation, ce qui permet d'éviter de lire 10 fois la même ligne.
Pour la réception, le tampon de 1 caractère a été maintenu.

```
UEUING NETWORK ANALYSIS PACKAGE (25-8-79) ***

  /DECLAR/ QUEUE BUS,PREC,VISU,PAPPLI,TPR,TPV,SVISU;
          INTEGER NC=0,LIGNE=0,RANG=0,COL,N;
          REAL TLM,TEM,TEF,TLF,TREC,TDEL,TCAR,TLIG,TPG,TEXEC,TMCY=0,025;
 /STATION/NAME=SVISU;
           TYPE=SEMAPHORE,MULTIPLE(N);
 /STATION/NAME=BUS;
          TYPE = RESOURCE;
          SCHED = PRIOR;
 /STATION/NAME=PREC;
          TYPE = GEN;
          SERVICE = BEGIN
          CST(TREC);
          END;
          TRANSIT=TPR;
 /STATION/NAME=TPR;
          SERVICE = BEGIN
          PRIOR:=2;
          P(BUS);CST(TEF);V(BUS);
          END;
          TRANSIT = PAPPLI;
 /STATION/NAME=TPV;
          TYPE=GEN;
          S RVICE=BEGIN
                  PRIOR:=1;
                  P(BUS);CST(TLM);V(BUS);
                  NC:=NC+1;
                  IF NC=40 THEN BEGIN NC:=0;P(SVISU) END;
                  END;
          TRANSIT=VISU;
 /STATION/NAME=VISU;
          SERVICE=BEGIN
          IF COL < 40 THEN BEGIN COL:=COL+1;CST(TCAR);END
                    ELSE BEGIN
                         COL:=0;
                         CST(9*(TLIG+40*TCAR));
                         V(SVISU);
                           CST(40*TCAR+2*TLIG);
                          RANG := RANG+1;
                         IF RANG=25 THEN BEGIN
                                  RANG := 0;
                                  CST(TPG)
                                  END;
                    END;
          END;
          TRANSIT = OUT;
 /STATION/NAME=PAPPLI;
          SERVICE = BEGIN
           PRIOR :=0;P(BUS);CST(TLF);V(BUS);
```

PROCESSUS DE RECEPTION

PROCESSUS DE VISUALISATION

```
49                EXP(TEXEC);P(BUS);CST(TEM);V(BUS);
50                      END;
51              TRANSIT = OUT;
52   /METHOD/TMAX=400;
53   /EXEC/BEGI"
54         PRINT ("  SIMULATIONS EN FONCTION DU T. CYCLE");
55          WHILE TMCY < 0.100 DO
56          BEGIN
57          TMCY:=TMCY+0.025;
58          N:=0;LIGNE:=0;COL:=0;NC:=0;RANG:=0;
59          TLM:=7*TMCY;TEM:=7*TMCY;
60          TEF:=8*TMCY;TLF:=9*TMCY;
61         TREC:=2.4;
62         TEXEC:=50*TMCY;TCAR:=1,185;TLAG:=12;TPG:=4000;
63         PRINT (" TMCY = ",TMCY);
64         SIMUL;
65         END;
66         END;
```

CY = .7500E-01 μs

```
 SIMULATION BEGINS ... ***
TIME =       400.00
*****************************************************************
AME *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *SERV NB*
*****************************************************************
    *          *          *          *          *          *      *
US  * .5914    * .5042    * .6203    * .7276    * .8525    *   341*
    *          *          *          *          *          *      *
REC * 2.400    * 1.000    * 1.000    * 4.400    * .4150    *   166*
    *          *          *          *          *          *      *
ISU * 1.165    * .1155    * .8453    * 8.444    * .9750E-01*    39*
    *          *          *          *          *          *      *
APP * 5.902    * .9919    * 52.12    * 136.0    * .1675    *   67*
    *          *          *          *          *          *      *
PR  * .8620    * .2750    * .2750    * .6628    * .4150    *   166*
    *          *          *          *          *          *      *
PV  * .8654    * 1.000    *1.0000    * .8654    * .9750E-01*    39*
    *          *          *          *          *          *      *
*****************************************************************
CE USED :  20 100
 END OF SIMULATION : TIME =       400.00
```

CY = .70000 μs

```
 SIMULATION BEGINS ... ***
TIME =       400.00
*****************************************************************
AME *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  THRUPUT *SERV NB*
*****************************************************************
    *          *          *          *          *          *      *
US  * .7874    * .6085    * .8149    * 1.055    * .1725    *   309*
    *          *          *          *          *          *      *
REC * 2.400    * 1.000    * 1.000    * 4.400    * .4150    *   166*
    *          *          *          *          *          *      *
ISU * 1.165    * .1155    * .1448    * 1.486    * .9750E-01*    39*
    *          *          *          *          *          *      *
APP * 7.092    * .9910    * 55.03    * 121.4    * .1275    *   51*
    *          *          *          *          *          *      *
PR  * .9170    * .3806    * .3806    * .9170    * .4150    *   166*
    *          *          *          *          *          *      *
PV  * 1.421    * 1.000    *1.0000    * 1.421    * .9750E-01*    39*
    *          *          *          *          *          *      *
*****************************************************************
CE USED :  21 100
 END OF SIMULATION : TIME =       400.00
```

Programme 4

Ce modèle correspond à l'architecture distribuée décrite au chapitre 8.
On simule le modèle correspondant au circuit Antiope, dans le cas où la mémoire de rafraichissement est implémenté par un automate local (§ 8.3.).
Les modifications importantes se situent au niveau de l'interface du processus de visualisation (IPV) et du processus d'interprétation (PTRAI).

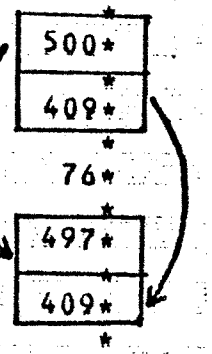
```
QUEUING NETWORK ANALYSIS PACKAGE (25-8-79) ***

    /DECLAR/ QUEUE BUS,PREC,VISU,PTRAI,IPR,IPV,MRAF;
             INTEGER COL,TRANCHE,LIGNE,COL2,TRANCHE2,LIGNE2;
             REAL TLM,TLM1,TLM2,TLM3,TEM,TETR,TLTR,TREC,TCAR,TLIG,TPG,TTRAI,TMG
    /STATION/NAME=MRAF;
    TYPE=SEMAPHORE;
    SCHED=PRIOR;
    /STATION/NAME=BUS;
             TYPE = RESOURCE;
             SCHED = PRIOR;
    /STATION/NAME=PREC;
             TYPE = GEN;
             SERVICE = BEGIN
             CST(TREC);
             END;
             TRANSIT=IPR;
    /STATION/NAME=IPR;                                  PROCESSUS DE
             SERVICE =
             BEGIN                                      RECEPTION
                  PRIOR:=2;
                  P(BUS);CST(TETR);V(BUS);
             END;
             TRANSIT = PTRAI;
    /STATION/NAME=IPV;
             SERVICE=
             BEGIN
            PRIOR := 1;
             P(MRAF);COL2:=COL2+1;CST(TLM);
                      IF COL2 > 40 THEN
                      BEGIN
                           CST(TLM1); COL2:=0;  & CHANGEMENT TRANCHE
                           TRANCHE2:=TRANCHE2+1;
                           IF TRANCHE2>10 THEN
                           BEGIN
                                 CST(TLM2);TRANCHE2:=0; &CHANGEMENT LIGNE
                                 LIGNE2:=LIGNE2+1;
                                 IF LIGNE2 > 25 THEN
                                 BEGIN
                                       LIGNE2:=0;
                                       CST(TLM3);
                                 END;
                           END;
                      END;
             V(MRAF);                                   PROCESSUS DE
             END;
             TRANSIT=OUT;                               VISUALISATION
    /STATION/NAME=VISU;
             TYPE=GEN;
             SERVICE=BEGIN
```

```
                    CST(TCAR);COL:=COL+1;
                    IF COL>40 THEN
                              BEGIN
                                   CST(TLIG);
                                   COL := 0;
                                   TRANCHE:=TRANCHE+1;
                                   IF TRANCHE = 10 THEN
                                   BEGIN
                                        CST(40*TCAR+TLIG);
                                        LIGNE:=LIGNE+1;
                                        TRANCHE := 0;
                                        IF LIGNE=25 THEN
                                        BEGIN
                                             LIGNE:=0;
                                             CST(TPG);
                                         END;
                                   END;
                         END;
                    END;
               TRANSIT = IPV;
/STATION/NAME=PTRAI;
               SERVICE = BEGIN
                         PRIOR := 0;
                         P(BUS);CST(TLTR);V(BUS);
                         EXP(TTRAI);
                         P(BUS);P(MRAF);CST(TEM);V(MRAF);V(BUS);
                           END;
               TRANSIT = OUT;
/METHOD/TMAX=600;
/EXEC/BEGIN
          PRINT (" SIMULATIONS EN FONCTION DU T. CYCLE");
          TMCY := 0.075;
           WHILE TMCY < 0.15 DO
           BEGIN
                TMCY:=TMCY+0.025;
                LIGNE:=0;COL:=0;TRANCHE:=0;
                 COL2:=0;TRANCHE2:=0;LIGNE2:=0;
                 TLM:= 7 * TMCY;TLM1:= 4 * TMCY;TLM2:= 4 * TMCY;TLM3:= 2 *TMC
                 TEM:= 7 * TMCY;TETR:= 8 * TMCY;TLTR:= 9 * TMCY;
               TREC:=1.2;
              TTRAI:=50*TMCY;TCAR:=1.18;TLIG:=12;TPG:=2000;
          PRINT (" TMCY = ",TMCY);
          SIMUL;
          END;
          END;
```

```
 SIMULATIONS EN FONCTION DU T. CYCLE
TMCY = .10000
```

```
*** SIMULATION BEGINS ... ***
***TIME =       600.00
```

NAME	SERVICE	BUSY PCT	CUST NB	RESPONSE	THRUPUT	SERV NB
BUS	.8202	.8889	1.250	1.135	1.083	650
PREC	1.200	1.000	1.000	1.200	.8333	500
VISU	1.449	1.000	1.000	1.449	.0817	409
RTKA	7.832	.9967	211.9	465.8	.1267	76
IPK	1.038	.8605	1.442	3.494	.0283	497
IPV	.7604	.5184	.5219	.7657	.0817	409
MKAF	.0000	.5691E-01	.5691E-01	.3449	.1650	99

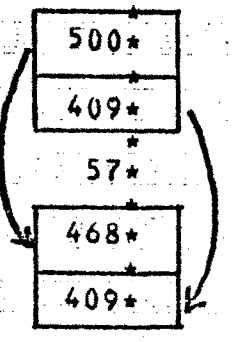



```
SPACE USED :  55/100
*** END OF SIMULATION : TIME =       600.00
```

TMCY = .1250 µs

```
*** SIMULATION BEGINS ... ***
***TIME =       600.00
```

NAME	SERVICE	BUSY PCT	CUST NB	RESPONSE	THRUPUT	SERV NB
BUS	1.025	.9964	1.559	1.396	.9717	583
PREC	1.200	1.000	1.000	1.200	.8333	500
VISU	1.449	1.000	1.000	1.449	.0817	409
RTKA	10.45	.9963	201.8	458.0	.9500E-01	57
IPK	1.274	.9957	19.03	43.13	.1800	468
IPV	.9578	.6329	.6844	1.004	.0817	409



A N N E X E 5

EXEMPLES DE MODULES

Pour évaluer la surface des modules qui sont le plus souvent utilisés dans les applications MOSAIC, on a dû se baser sur des circuits standard commerciaux.

Il est évident que cette méthode d'évaluation conduit à des résultats non optimisés car les structures internes de ces circuits ne sont pas adaptées à la topologie qu'il faudrait respecter dans une méthode de conception comme MOSAIC.

Pour optimiser la surface totale, il faut concevoir les modules en essayant de disposer les organes internes de façon à minimiser les connexions avec l'interface et la bande E/S.

Cependant, les informations disponibles sur la surface des circuits standard et quelquefois sur leur topologie interne, permettent d'avoir très rapidement des estimations de la surface des futurs modules.

1. Modules programmables

On a pris comme exemple les circuits INTEL 8048 et MOTOROLA 6801. Ces deux micro-ordinateurs correspondent à peu prêt à la définition de module programmable qui a été faite au § 9.1.

On peut résumer les caractéristiques de l'INTEL 8048 (INT 78, STA 77, WAK 79) de la manière suivante :

technologie NMOS - 6 µm

organes internes : la structure interne est construite autour d'un bus unique sur lequel viennent se connecter :

- le CPU 8 bits avec un accumulateur et des registres tampon
- la mémoire programme - 1 K octet de ROM
- la mémoire de données - 64 octets de RAM
- un timer 8 bits
- 2 ports d'E/S de 8 bits quasi-bidirectionnels
- 1 port d'E/S bidirectionnel utilisé pour des extensions de la mémoire programme ou de données, ou pour une connexion à un bus micro-processeur.

Les applications envisagées pour le module de ce type sont d'utiliser les possibilités algorithmiques de son jeu d'instructions pour programmer des petits interpréteurs qui, en fonction d'un certain langage de commande, génèrent des signaux de contrôle sur ses nombreuses E/S.

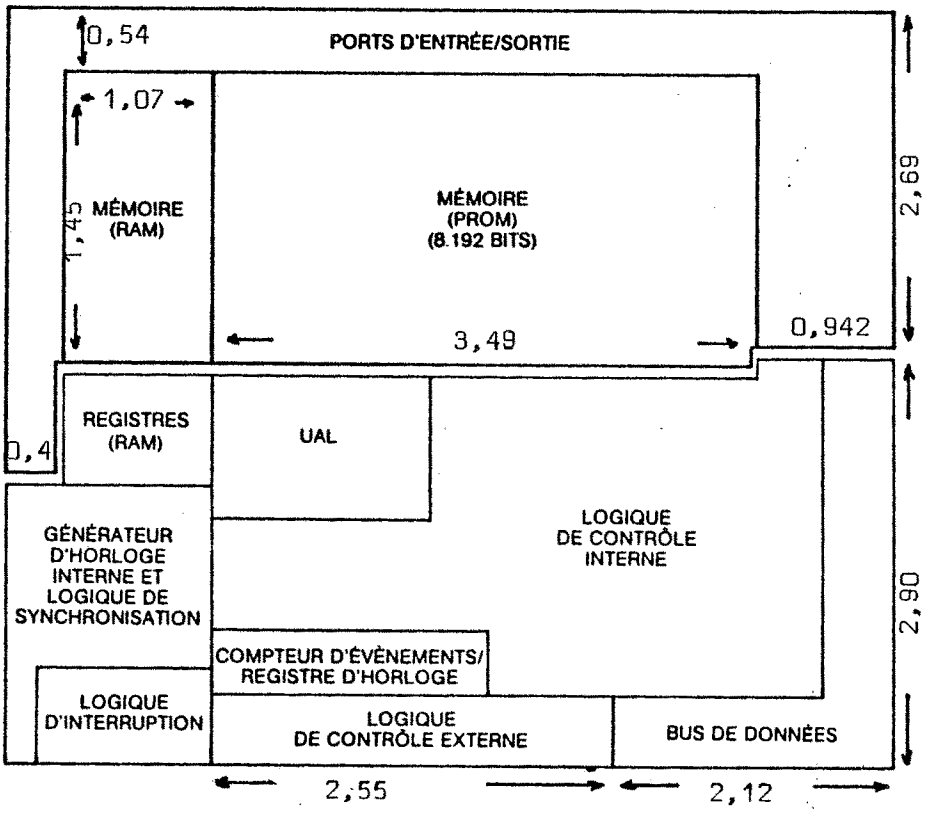


Figure 1 - Schéma d'implantation du 8748 [TOO 77]

La surface du 8048 est de 33,96 mm^2

$33,96 - 13,30\ mm^2 = 20,66\ mm^2$

Il faut ajouter la surface de l'interface

$20,66 + 0,26 = 20,92\ mm^2$

Dans l'hypothèse de la topologie du § 3 .4

$20,76\ mm^2 = 4,5\ mm \times 4,65\ mm$

Connexions vers l'extérieur

port 1 8bits	
port 2 8bits	
test 0	testés par programme
test 1	
INT	interruption
19 lignes	

Ces lignes sont présentent dans la bande d'interconnexions d'E/S du module, mais seulement celles utilisées pour l'application sont connectées aux amplificateurs d'E/S.

Evolution de la surface du module :

mémoire morte	3,19 mm^2
mémoire RAM	1,55 mm^2
logique	16,02 mm^2

En appliquant les coefficients de l'annexe 3, on obtient :

	HMOS (3,5 µm)	2 µm
mémoire morte	1,04 mm^2	0,59
mémoire RAM	1,0 mm^2	0,70
logique	8,17 mm^2	4,12
	10,21 mm^2	5,41 mm^2
	3,2 mm x 3,19 mm	2,28 mm x 2,37 mm

Caractéristiques du MOTOROLA 6801 :

Technologie NMOS - 6 µm

Organes internes : le circuit est construit autour d'un CPU 6800 auquel on a rajouté certaines instructions

- CPU 8 bits
- mémoire de données 128 octets
- mémoire programme 2 K octets
- timer 16 bits avec une logique de contrôle puissante
- interface série pouvant travailler en NRZI et biphase
- 3 ports E/S (8 bits) programmables
- 1 port 5 bits qui peut être utilisé par l'interface série ou le timer ou être défini comme un port banalisé.

Le jeu d'instructions est plus puissant que celui du 8048, ce qui permet d'utiliser le 6801 pour des applications avec un traitement informatique plus important. La taille de la mémoire interne peut cependant être contraignante et il faudrait maintenir la possibilité de travailler avec une mémoire externe.

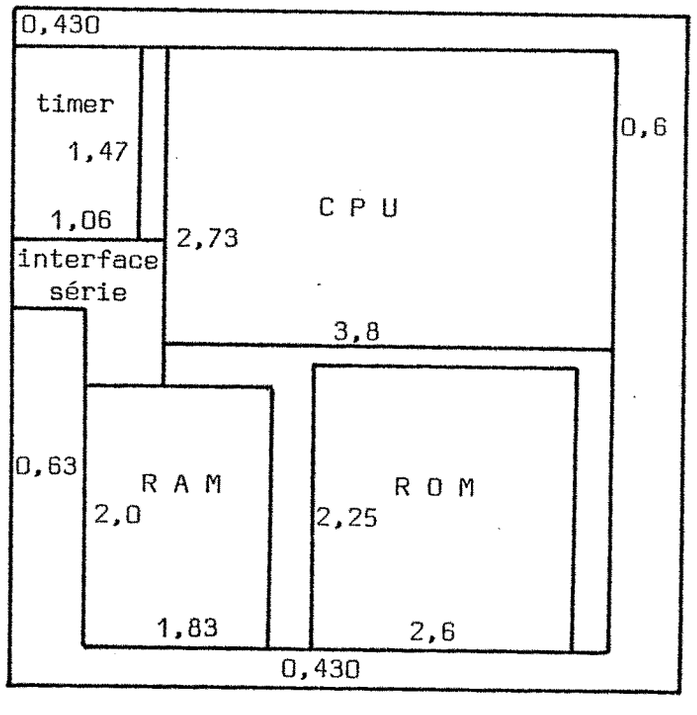


Figure 2 - Schéma d'implantation du 6801

Pour utiliser le 6801 comme module MOSAIC, il faut retirer les amplificateurs et plots d'E/S qui représentent une surface de 10,51 mm^2.
L'interface série doit être également retirée, car elle n'a pas de sens dans une méthode de conception modulaire (surface 1,36 mm^2).

La surface du 6801 étant de 39 mm^2, un module MOSAIC équivalent aura donc :

39 - 10,51 - 1,37 = 27,13 mm^2

dans la topologie qui a été définie :

4,5 mm x 6 mm

Connexions d'E/S :

3 ports d'E/S de 8 bits	24
2 interrupts	2
2 strobles qui permettent de contrôler le port 3	2
	28 lignes

Evolution de la surface :	HMOS	2 µm
mémoire morte - 5,8 mm^2	1,62	1,07
mémoire vive - 3,66 mm^2	2,16	1,525
logique 17,67 mm^2	9,02	4,55
	12,8 mm^2	6,08 mm^2
	3,2 mm x 4 mm	2,28 mm x 2,67

2. Modules dédiés

2.1. Définition

Ce sont les modules chargés d'exécuter une tâche spécifique, d'E/S ou de calcul, et n'ayant pas la possibilité d'être programmés.

Les exemples présentés se basent sur des circuits périphériques des systèmes de microprocesseur classiques. Il faut cependant souligner la différence qui existe entre la façon dont sont utilisés ces circuits dans un système de type micro-ordinateur et l'emploi des modules dédiés dans MOSAIC.

Dans un système microprocesseur, un circuit périphérique ne peut exécuter ses fonctions spécifiques que sous le contrôle d'un CPU qui se charge de le commander et de vérifier si l'opération a été bien exécutée. Dans MOSAIC, chaque module est un processeur autonome qui peut prendre l'initiative de dialoguer avec les autres composants du système.

2.2. Interface ligne série

On se base sur la description du circuit MOTOROLA 6850. Une interface série full-duplex est réprésentée, dans le système MOSAIC, par le schéma fonctionnel de la figure 2.1.

Par rapport au circuit MOTOROLA, les processus d'émission et de réception sont peu différents ; c'est surtout au niveau du processus de contrôle qu'il y a des différences importantes.

Processus de contrôle :
ce processus se charge de paramétrer l'activité des deux autres, en fonction des ordres qui sont écrits dans un moniteur externe (moniteur ordre).
Pour une interface série, on peut considérer par exemple les ordres suivants :

- paramétrer vitesse de transmission,
- paramétrer type de caractère,
- démarrer réception (nom tampon réception),
- démarrer émission (nom tampon émission).

Le processus de contrôle est donc un petit interpréteur qui décode l'ordre qui lui est envoyé et qui, en fonction de celui-ci, charge les registres et démarre ou arrête les processus d'émission ou de réception.

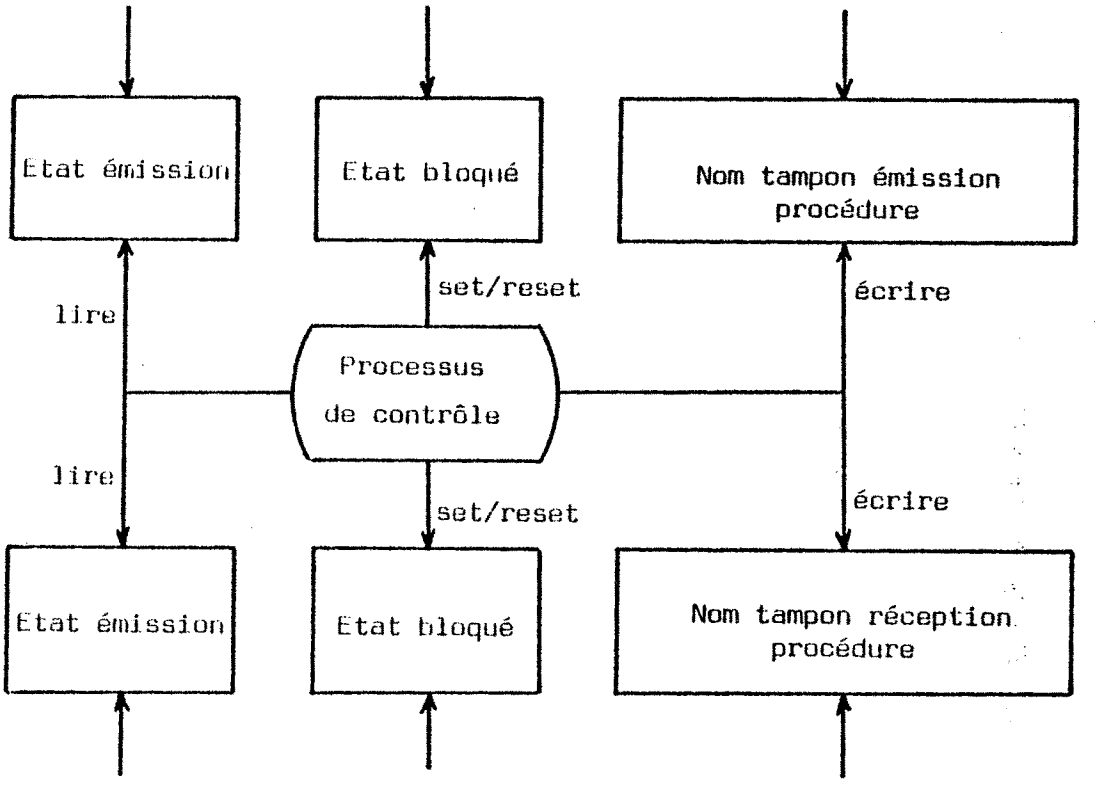


Figure 3 - Processus de contrôle

2.3. Structure matérielle

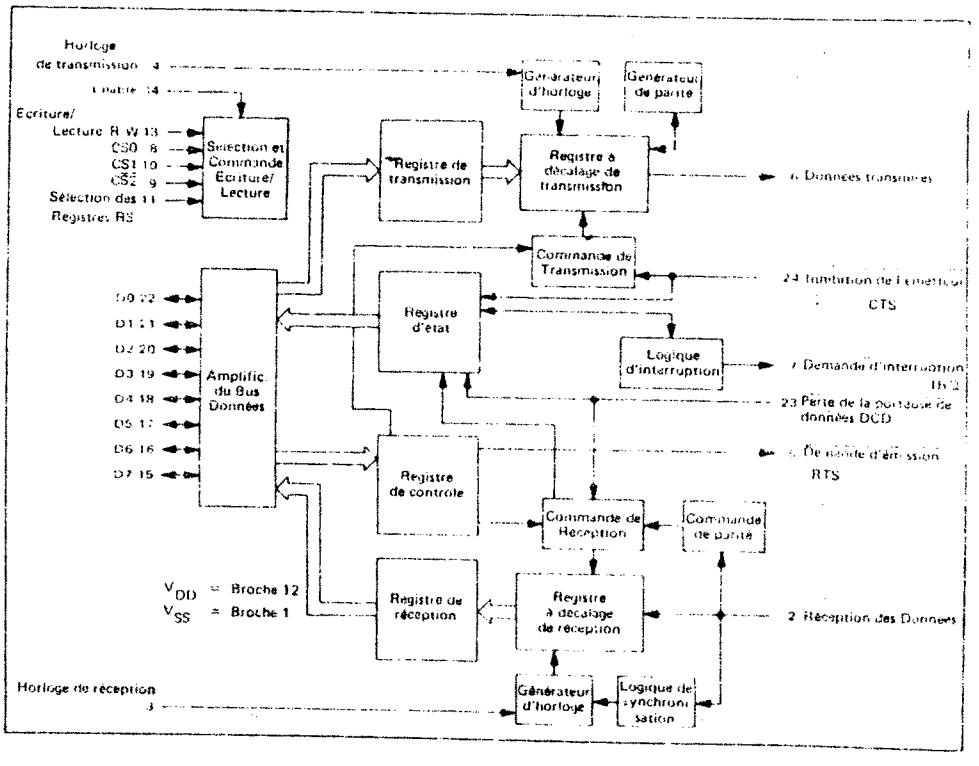


Figure 4 - Schéma fonctionnel du MOTOROLA 6850

Il faut supprimer de ce schéma :

- la sélection et la commande écriture/lecture,
- l'amplificateur du bus de données,
- la logique d'interruption,

et il faut rajouter :

- le processus de contrôle,
- l'interface avec le bus commun.

Surface : le coeur du circuit 6850 (sans plots d'E/S) fait 6,2 mm^2 en technologie 6 µm.

a) Processus de contrôle

Le processus de contrôle peut être implémenté par un petit automate à base de PLA que l'on a calculé pour avoir 10 entrées, 8 sorties et 20 états internes pour exécuter les commandes très simples que l'on a décrit précédemment

$$S_{PLA} = 587 \times 400 = 0,23 \text{ mm}^2$$

On peut donc considérer que le processus de contrôle ne représentera qu'une augmentation de l'ordre de $0,23 \times 1,5 = 0,35 \text{ mm}^2$, car les registres d'état et de contrôle existaient déjà dans le 6850.

b) Interface bus

Une seule interface multiplexée pouvait être utilisée par les trois processus ; il faut cependant tenir compte du débit maximal des processus d'émission et réception.

La vitesse de transmission maximale d'un 6850 est de 500 Kbits/s, donc 20 µs par caractère. Si on suppose une transmission en full duplex, on aurait une moyenne de 10 µs pour :

- allouer l'interface,
- transférer le caractère vers le registre de l'interface,
- allouer le bus,
- exécuter le transfert,
- signaler la fin du transfert au processus demandeur.

Avec un temps de cycle de 100 ns, l'écriture ou la lecture d'un FIFO prend 1,2 µs et les transferts à l'intérieur du module sont aussi relativement rapides ; le paramètre important est donc le temps d'allocation bus qui ne dépend que du reste du système.

Pour éviter d'imposer des contraintes sur les mécanismes de communication et pouvoir éventuellement travailler à des vitesses de transmission plus élevées, on peut utiliser deux interfaces car la place occupée est relativement petite

$$2 \times 0,25 = 0,5 \text{ mm}^2$$

Le processus de contrôle utilisera l'interface du processus d'émission pour lire des ordres et signaler des conditions d'erreur.

Surface totale :

$$6,2 + 0,5 + 0,5 = 7,2 \text{ mm}^2$$

L'interface série serait donc un module de dimension latérale de 2,25 mm

2,25 mm x 3,2 mm

en technologie 2 µm

1,14 mm x 1,62 mm

3. Module clavier

Fonctions :

balayage du clavier

encodage des touches alphanumériques

encodage et interprétation des touches fonction.

Ayant très peu d'information sur ce type de circuit, on a essayé de définir un schéma fonctionnel et ensuite d'évaluer la surface des différents blocs nécessaires.

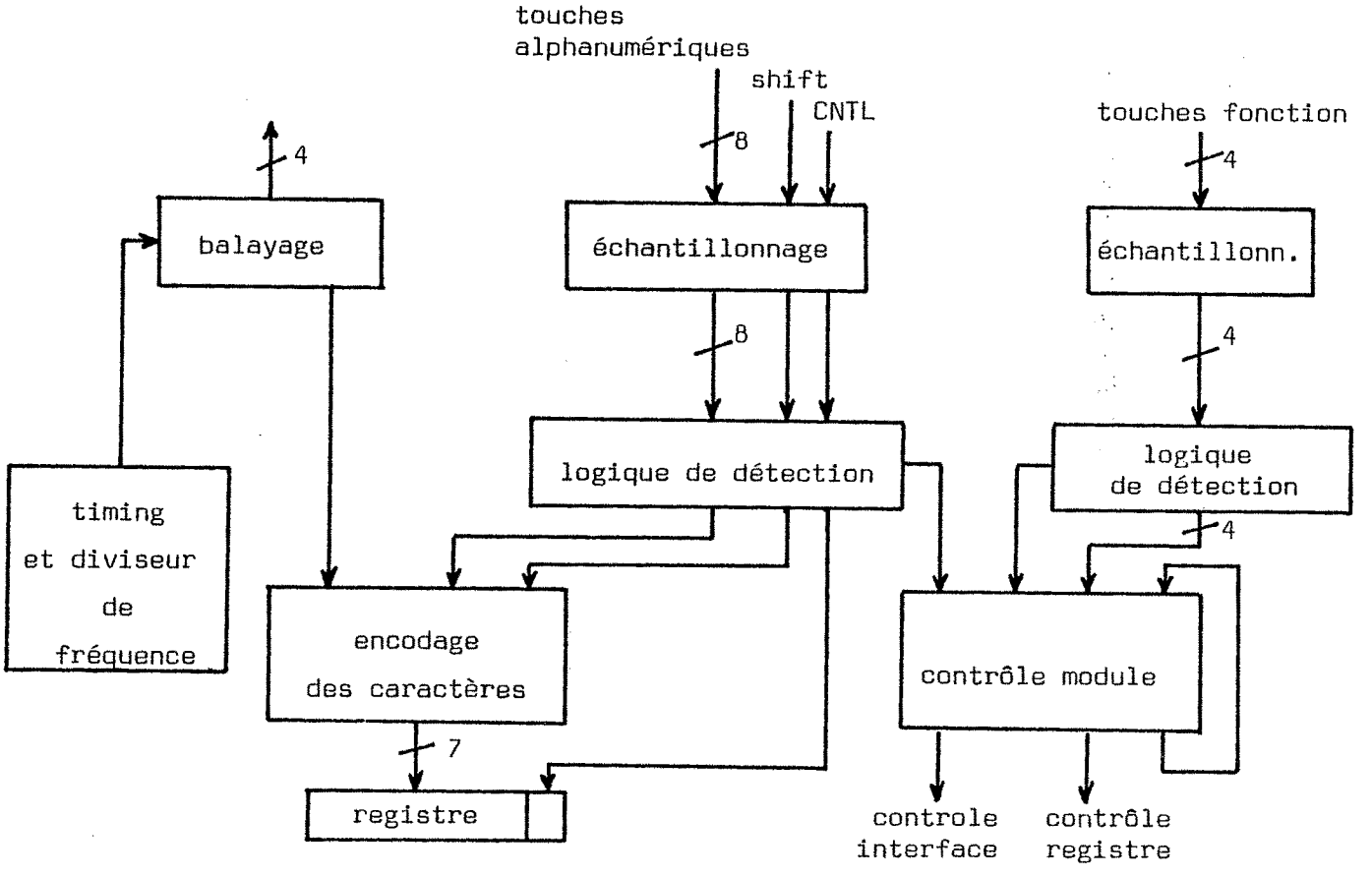
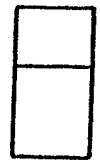


Figure 5 - Schéma fonctionnel du module clavier

Balyage : compteur 4 bits

4 registres avec un incrémenteur



220 µ x 240 µ ≈ 0,0528 mm^2

Remarque : le balayage codé implique l'utilisation d'un décodeur à l'extérieur, mais l'option non codée obligerait à utiliser 16 lignes E/S au lieu de 4. Le besoin de minimiser le nombre total de broches du circuit nous oblige donc à utiliser la solution codée.

Echantillonnage et logique de détection :

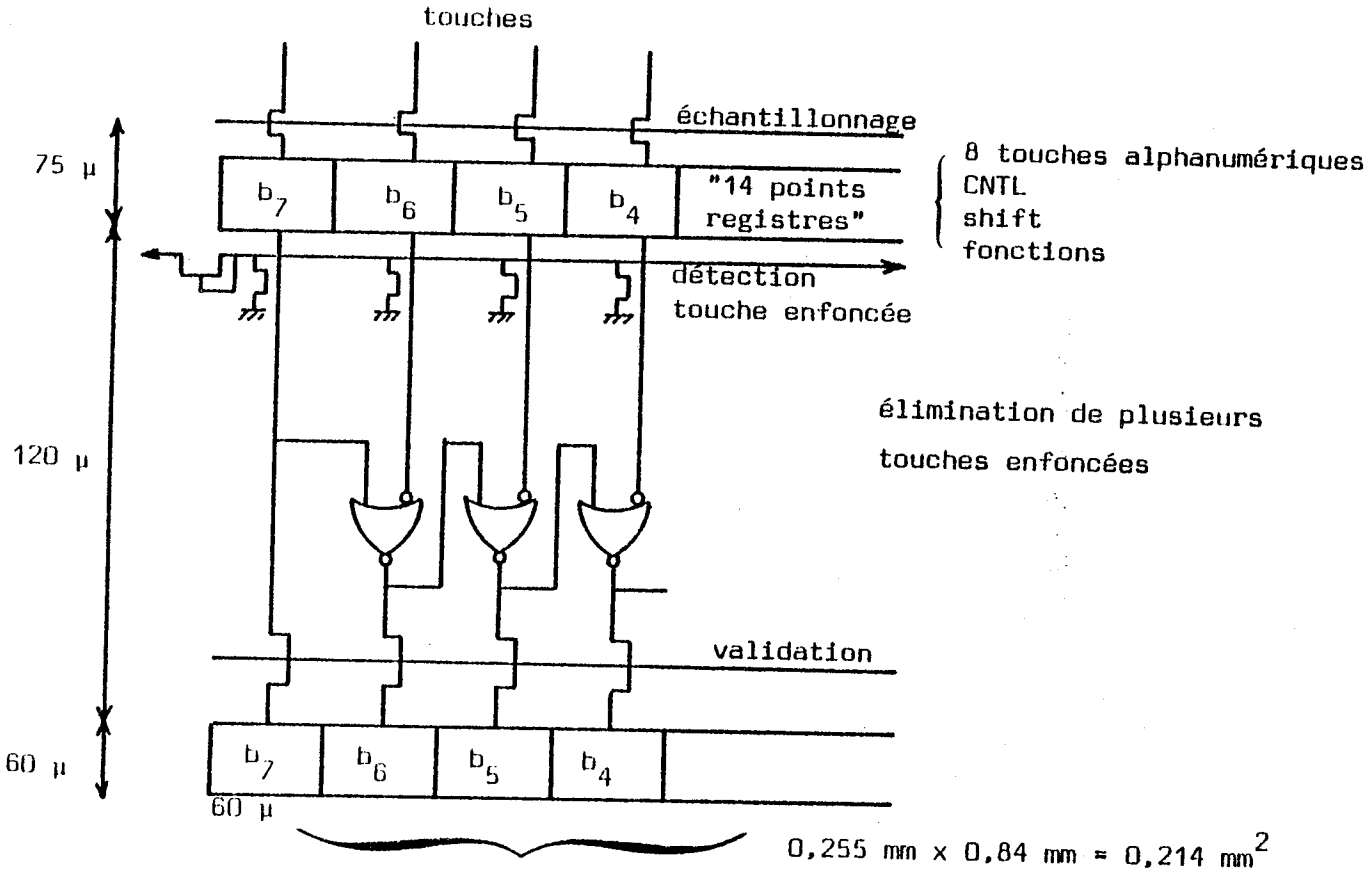
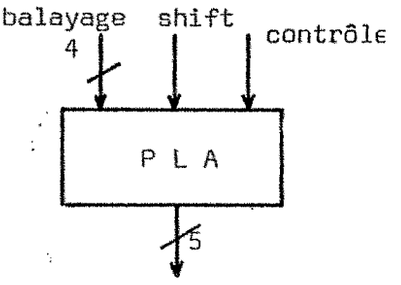


Figure 6

Encodage caractères :

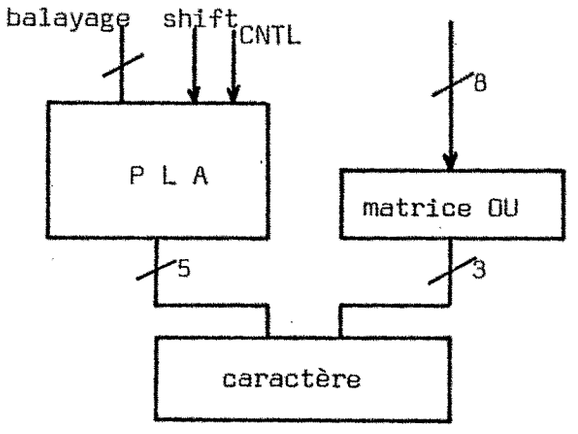
Un module clavier doit être très souple dans l'encodage des touches du clavier, de façon à pouvoir être utilisé dans des applications différentes (ASCII, clavier français, EBCDIC, ..). La solution la plus souple est de réaliser l'encodage en utilisant des PLAs dont le masque pourrait être modifié.

Etant donné que l'on a considéré un balayage code, le PLA de transcodage aurait un maximum de 32 états internes



Surface du PLA : 0,322 mm = 0,72 mm = 0,23 mm^2

L'encodage est fait de la façon suivante :



$S_{matrice\ OU}$ = 160 µ x 60 µ = 9600 μm^2

registre = 8 x 4000 μm^2 = 32000 μm^2

Figure 7

Pour l'entrée sur 8 bits, une seule ligne sera à "1" et donc une simple matrice OU sera suffisante.

Surface du bloc d'encodage : $S_{PLA} + S_{matrice\ OU}$ + registre =

0,23 + 0,0096 + 0,032 = 0,271 mm^2

Touches fonctions

La signification de ces touches est encore plus dépendante de l'application que les touches alphanumériques. On a donc décidé de les connecter directement au bloc de contrôle du module.

Contrôle du module

Pour maintenir l'idée initiale de faire de chaque module une entité indépendante dans le système, on pense que le module clavier doit aussi avoir une unité de contrôle. Cette unité doit effectuer deux fonctions :

1) contrôler les organes internes (échantillonnage des caractères, contrôle de l'interface),
2) effectuer une partie du traitement qui est normalement associé aux touches fonctions. Comme on le verra en annexe 6, les touches fonctions peuvent correspondre soit à des caractères spéciaux, soit elles fournissent l'identification du moniteur qui doit recevoir les caractères suivants.

On a donc prévu une sortie du PLA sur 8 bits qui permet de coder soit un caractère soit un appel à un moniteur externe. La programmation du PLA permettra d'adapter le module clavier à une application particulière.

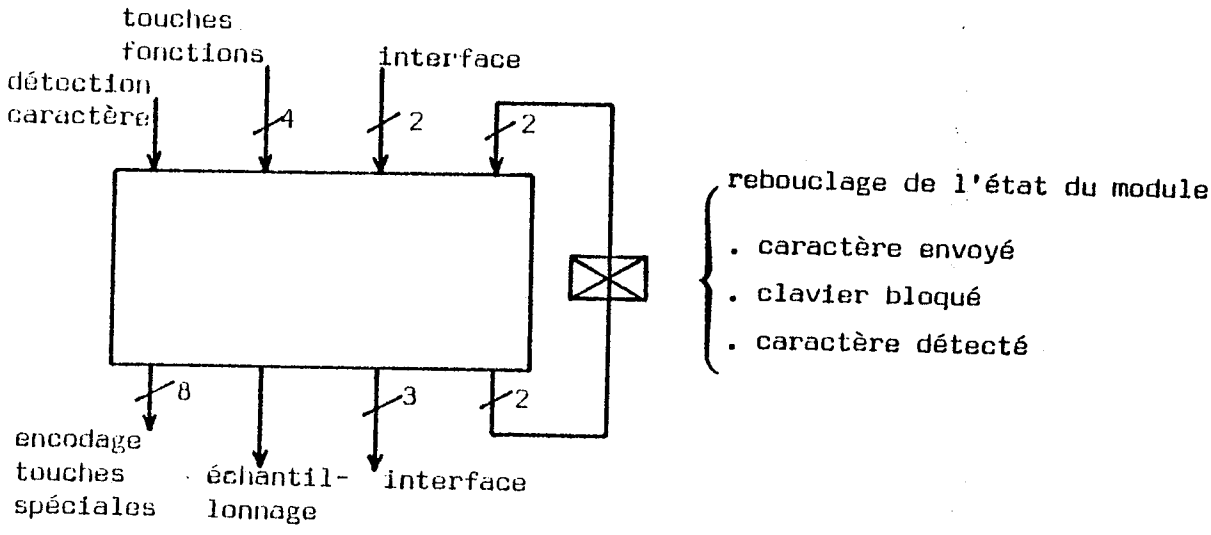


Figure 8

La surface du PLA dépend de l'application : on peut cependant supposer que 30 monômes seront suffisants : entrées 9, sorties 14, monômes 30.

S_{PLA} = 670 µm x 660 µm = 0,44 mm^2

Remargue : la conception de ce module doit, à notre avis, être étudiée soigneusement car, au niveau du clavier, on peut détecter certaines touches (contrôle du curseur, effacer caractère, effacer ligne, retour chariot), qui correspondent à des algorithmes qui peuvent être exécutés par le processeur noyau si le module clavier appelle les procédures correspondantes. Il nous semble que ce premier décodage et l'appel des procédures correspondantes pourraient être également exécutés par le module clavier.

Séquencement

. Générateur des phases : 0,250 mm^2
. Diviseur de la fréquence de base (10 Mhz) pour générer une fréquence d'échantillonnage de 200 hz. Un registre à décalage rebouclé avec 16 étages permet de faire une division par 2^{16}, ce qui donne une fréquence d'échantillonnage de 152 hz.

Surface : 0,176 mm^2

Balayage	0,053
échantillonnage	0,214
encodage	0,271
contrôle	0,44
timing	0,426
interface bus	0,25
	1,654 mm^2

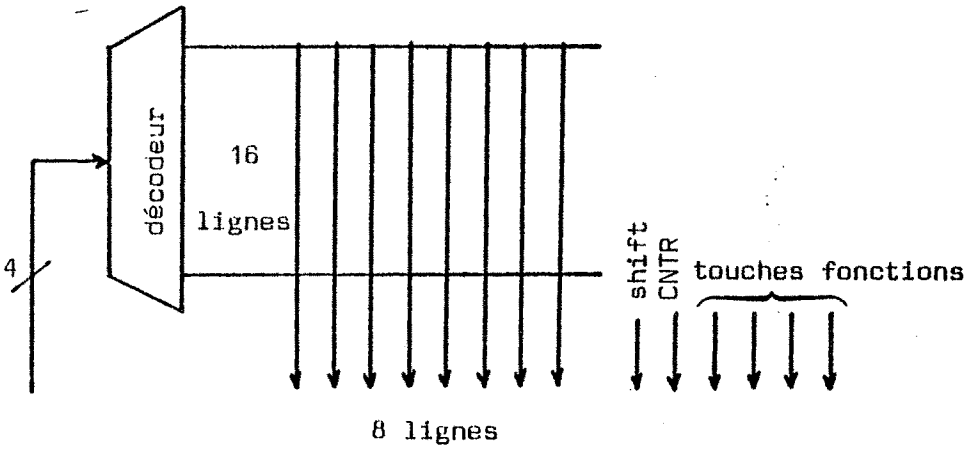
Dans ces calculs on n'a pas pris en compte les interconnexions, on va donc supposer qu'elles représentent une surface identique à celle de la logique

S = 1,654 x 2 = 3,31 mm^2

2,25 mm x 1,47 mm

nombre de lignes d'E/S : 18

Les connexions à l'extérieur du module (si toutes les lignes étaient utilisées) permettent de connecter un clavier de 32 touches sans décodage et de 128 touches avec décodage.



4. Contrôleur de visu

Dans la définition du contrôleur de visu, un des choix importants à faire est d'utiliser ou non un tampon interne pour les caractères à visualiser. Trois solutions sont possibles :

1) pas de tampon : dans ce cas, pendant l'affichage d'une ligne, le mécanisme de communication doit fournir un caractère toutes les 1,2 µs. Ceci représente une surcharge qui empêcherait pratiquement les autres modules de l'utiliser pendant l'affichage des caractères ;
2) tampon pour mémoriser la ligne à afficher : pour une ligne de 80 caractères, le tampon représente une augmentation de surface du module correspondant à une RAM de 80 x 8 = 480 bits. Le chargement du tampon doit se faire pendant que les lignes vides entre deux rangées de caractères sont affichées.Pour un codage utilisant une matrice de 7x5, il y a 5 lignes vides et donc on aura en moyenne 6 µs pour charger chaque caractère dans le tampon. Quoique la fréquence de transfert exigée du mécanisme de communication soit inférieure, elle continue à imposer une contrainte importante sur la performance du système qui ne pourrait pas avoir d'autres processus travaillant à des fréquences élevées.

3) double tampon : un tampon est rempli pendant que le contenu du deuxième est affiché. Cette solution est celle qui occupe le plus de surface, mais elle a l'avantage de n'imposer aucune contrainte importante au niveau de la performance du mécanisme de communication. On a donc préféré l'utiliser.

Evaluation de la surface

Pour l'évaluation de la surface, on se base sur le circuit 96364 de SESCOSEM. Ce circuit contient la logique nécessaire pour contrôler une unité de visualisation (compteurs, comparateurs, génération de synchronisation TV), mais il n'y a ni le double tampon ni le générateur de caractères.

Tampon

Le tampon est une mémoire RAM de 2 x 80 x 8 = 960 bits.

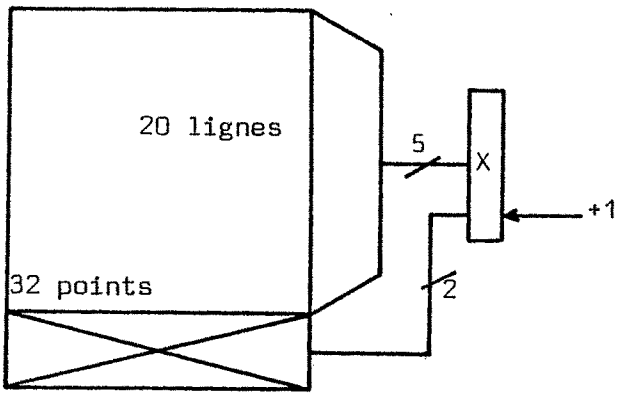


Figure 10

mémoire : (32 x 60) x (20 x 55) = 1920 = 1100 = 2,12 mm^2
décodeur : (5 x 37) x 1100 = 0,20 mm^2
multiplexeur : (4 x 20) x 1920 = 0,154 mm^2
total : 2,15 mm^2

Générateur de caractères

On peut choisir soit de le mettre à l'intérieur du module, ce qui permet d'économiser 11 lignes d'E/S mais oblige à utiliser un alphabet fixe, soit de le mettre à l'extérieur. On pense que dans une bibliothèque MOSAIC, ces deux options devraient exister, de façon à permettre de choisir en fonction du nombre d'E/S disponibles et de l'alphabet utilisé.

Dans l'option qui consiste à avoir le générateur de caractères à l'intérieur du circuit, on aurait, pour une matrice de 7x5 et un jeu de 64 caractères :

64 x 7 = 448 mots de 5 bits

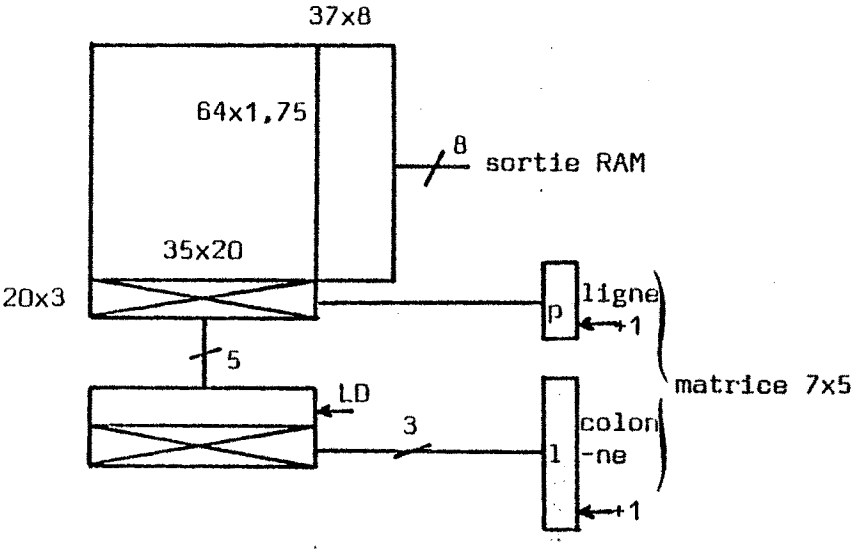


Figure 11

Surface plan mémoire : (35 x 20) x (64 x 17,5) = 700 x 1120 = 0,784 mm2

décodeur : (37 x 8) x 1120 = 0,33 mm^2

multiplexeur : (20 x 3) x 700 = 0,042 mm^2

total : 1,156 mm^2

5. Curseur

Les fonctions de manipulation du curseur ne sont pas exécutées au niveau du module, mais dans le processeur noyau où sont actualisés et maintenus les pointeurs sur la mémoire.

Au début de chaque trame, le module doit lire la valeur du pointeur d'écriture et le stocker dans un registre interne pour pouvoir effectuer le clignotement du curseur.

Les fonctions de roll-up, down, up, right, left, sont aussi effectuées par le processeur noyau, en modifiant les pointeurs associés à la mémoire de rafraichissement.

Surface :

La surface du coeur du 96364 est de 7,2 mm^2 dans une technologie 7 µ, avec un pas d'aluminium de 20 µm. En utilisant les règles de l'annexe 3, on peut calculer les facteurs de réduction pour ramener cette surface aux règles 6 µm

$$S_{7\mu} / S_{6\mu} = 1/1,93$$

La surface du coeur du 96364 serait donc de

$$7,2 / 1,93 = 3,73 \ mm^2$$

Dans le schéma fonctionnel de la figure 12, il faut éliminer la logique de contrôle du pointeur d'écriture, ce qui représente dans le schéma d'implantation (GAS 77), à peu prêt 1/6 du circuit.

Il faudrait rajouter un bloc de contrôle, le générateur de caractères et le tampon pour les deux lignes de caractères.

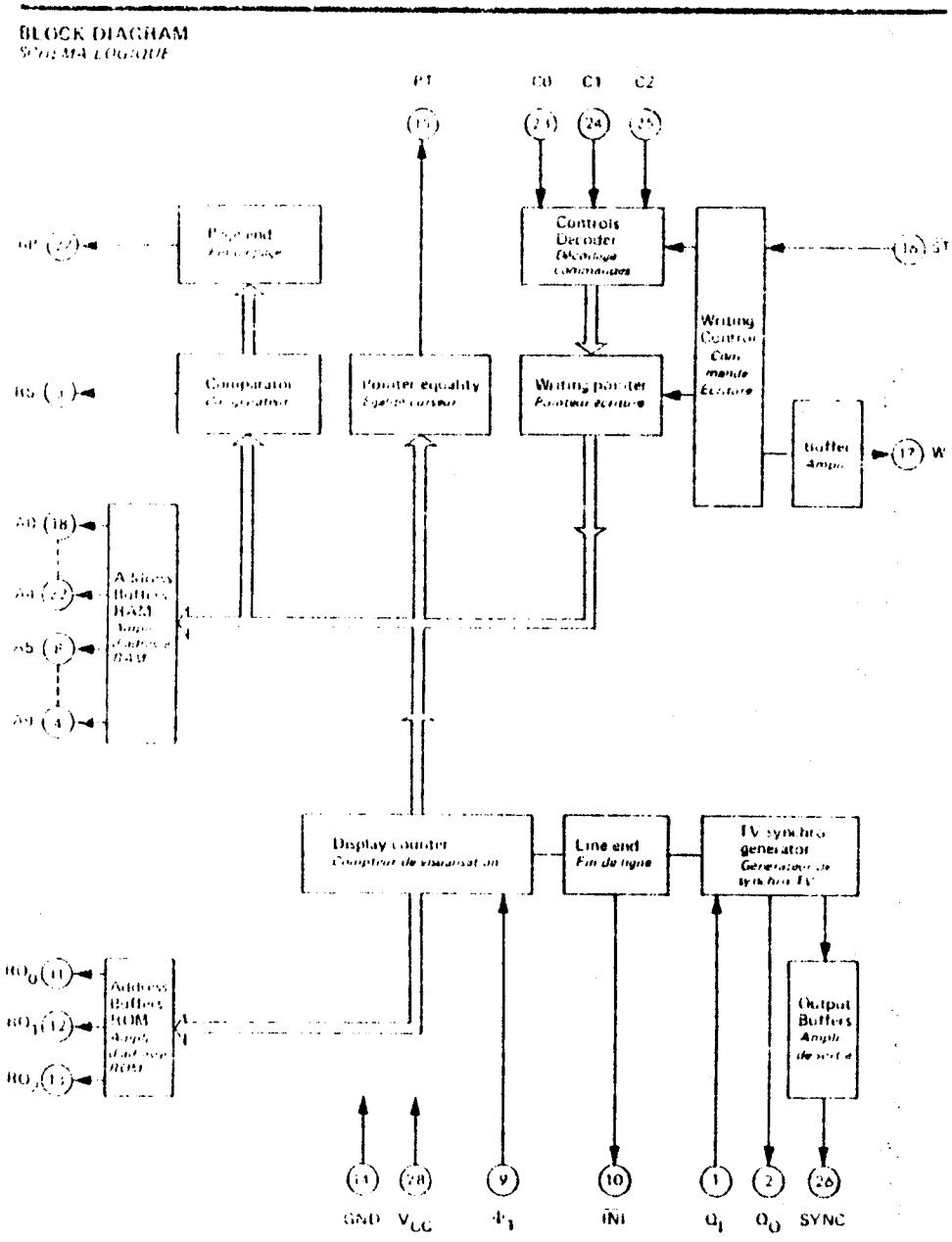


Figure 12 - Schéma fonctionnel du 96364 SESCOSEM

Surface :

$$3,73 - \frac{3,73}{6} = 3,11 \text{ mm}^2$$

	3,11 mm^2
tampon pour 2 rangées de caractères	- 4,12
générateur caractères	- 1,50
module de contrôle	- 0,5
interface bus	- 0,26
	9,49 mm^2

Dans la topologie MOSAIC :

4,5 x 2,10

Les lignes pour l'extérieur sont peu nombreuses :

1 vidéo

1 synchronisation

2 horloge

4

A N N E X E 6

EXEMPLES D'APPLICATIONS

1. Circuit pour contrôler un terminal informatique regroupant plusieurs périphériques

La fonction principale du circuit est de contrôler le flux d'information entre les périphériques qui lui sont connectés.

Le terminal est indépendant du système d'exploitation de l'ordinateur auquel il est connecté, il ne fait que le routage de l'information vers ou depuis les différents périphériques et l'ordinateur, en fonction des commandes qui lui sont données par l'utilisateur.

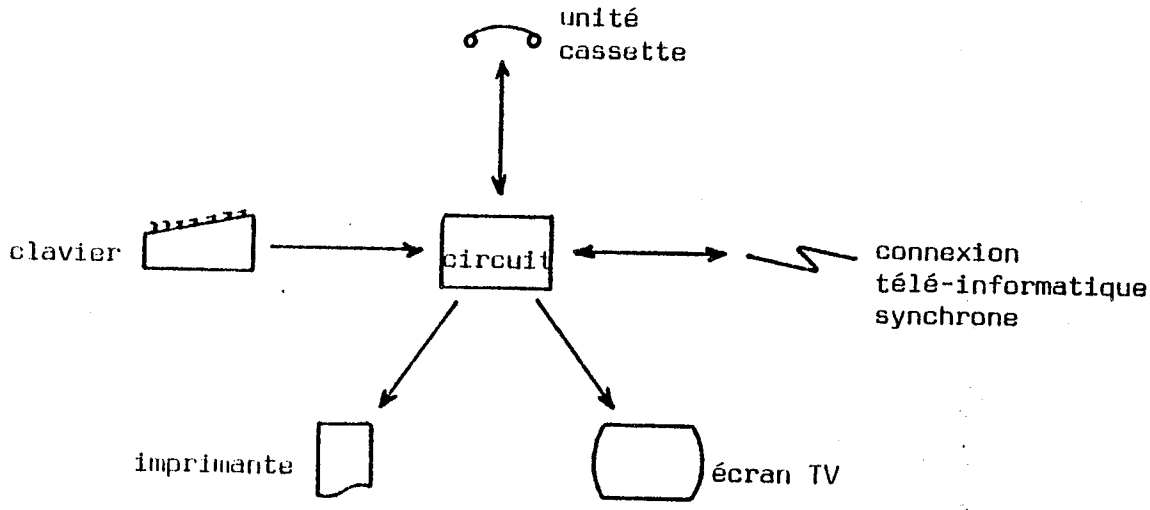


Figure 1 - Exemple étudié

Le système est composé des périphériques suivants :

Clavier : clavier alphanumérique habituel
touches fonction permettant de contrôler l'aiguillage du flux de caractères vers les autres modules.

Console_vidéo : avec une page de 16 x 80 caractères et un générateur de caractères classique de 7x5.

Imprimante : liaison série permettant d'envoyer des données ou des caractères de contrôle vers une imprimante que l'on suppose dotée de la logique de contrôle nécessaire.

Cassette : contrôle d'une unité de cassette digitale qui est utilisée comme mémoire de masse du terminal.

Liaison_de_télé-informatique_asynchrone : permet de connecter le terminal à un ordinateur central.

Fonctionnement :

Une zone de travail de 2560 octets a été définie dans la mémoire de communication du circuit. Dans le processeur noyau sont programmés les algorithmes d'accès à la zone de travail correspondant aux différents périphériques.

Entrée_texte :

En utilisant le clavier, l'utilisateur peut entrer du texte dans la mémoire de travail ; celle-ci permet un maximum de 32 lignes de 80 caractères. Les touches fonction (cassette, imprimante, ligne) sont utilisées pour sélectionner un module ; la ligne suivante est comprise comme une commande à destination du module sélectionné.

Impression :

Imprimer le contenu de la zone de travail jusqu'à la détection d'un EOT (End of Text) L'impression se fait par des blocs de 2560 caractères jusqu'à la détection d'un EOT. La synchronisation associée aux procédures d'accès à la zone de travail permet d'arrêter le processus d'impression à la fin d'un bloc et de le faire démarrer quand un nouveau bloc est prêt à être imprimé.

Remarque : on n'a pas prévu l'hypothèse d'une impression directe à partir du clavier, car avant d'imprimer, l'utilisateur fait toujours appel à l'éditeur de l'ordinateur central pour mettre son texte au propre.

L'impression peut se faire avec ou sans affichage simultané sur l'écran.

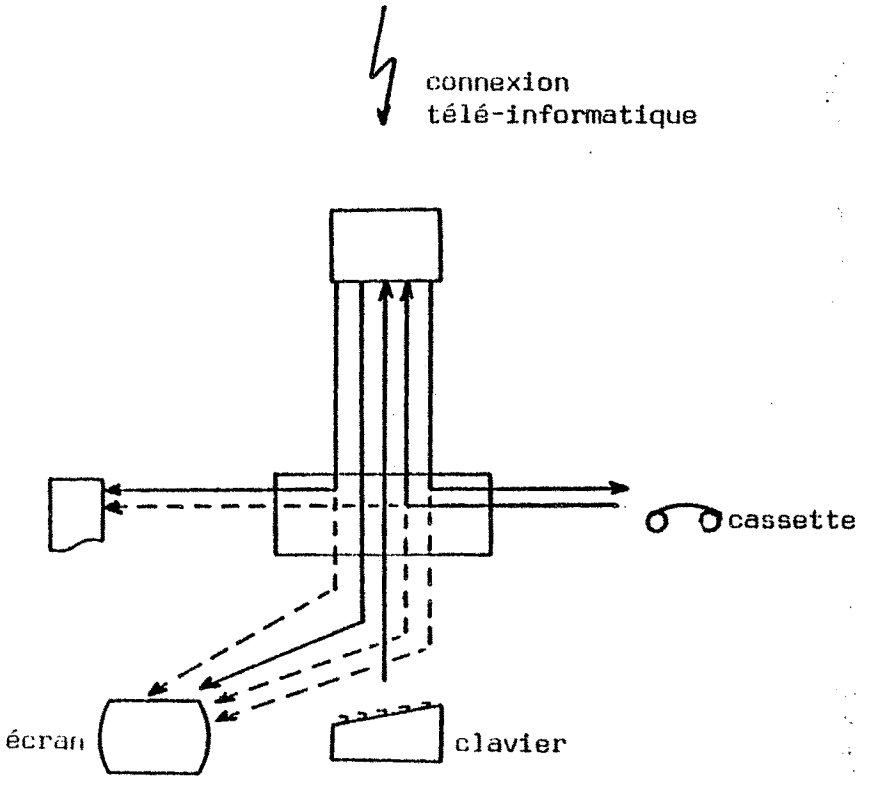


Figure 2 - Flux des données

Cassette :

Le logiciel de gestion de la cassette est très simple. On considère que la bande est divisée en blocs de taille fixe de 2560 octets. Les blocs sont référencés par le nombre correspondant à sa position dans la bande. Une recherche aléatoire peut donc être facilement faite (horloge gravée sur la bande). Dans le bloc 0 il y a un catalogue permettant de connaître les blocs utilisés. Il n'y a pas de système de fichier ; l'utilisateur doit gérer lui-même les blocs de la bande utilisés.

Les commandes du module cassette sont :

écrire/lire ; bloc n° i jusqu'à bloc n° j

lire bloc 0 et l'afficher sur l'écran

Pour les mêmes raisons que celles évoquées pour l'impression, on n'a pas jugé utile de graver directement du texte à partir du clavier.

Télé-informatique :

Transmission

- envoyer caractère à caractère
- envoyer des blocs jusqu'à la détection d'un EOT.

Réception

- recevoir des caractères et les ranger dans la zone de travail, avec ou sans affichage.

La figure 3 présente le système décrit dans le formalisme défini au chapitre 2. Les modules utilisés sont :

Module clavier : semblable à celui décrit dans l'annexe 5. Les touches fonction sont "cassette", "imprimante", "ligne". Le décodage des touches permet d'aiguiller la ligne suivante vers le moniteur correspondant.

Surface : 2,5 mm x 1,47 mm

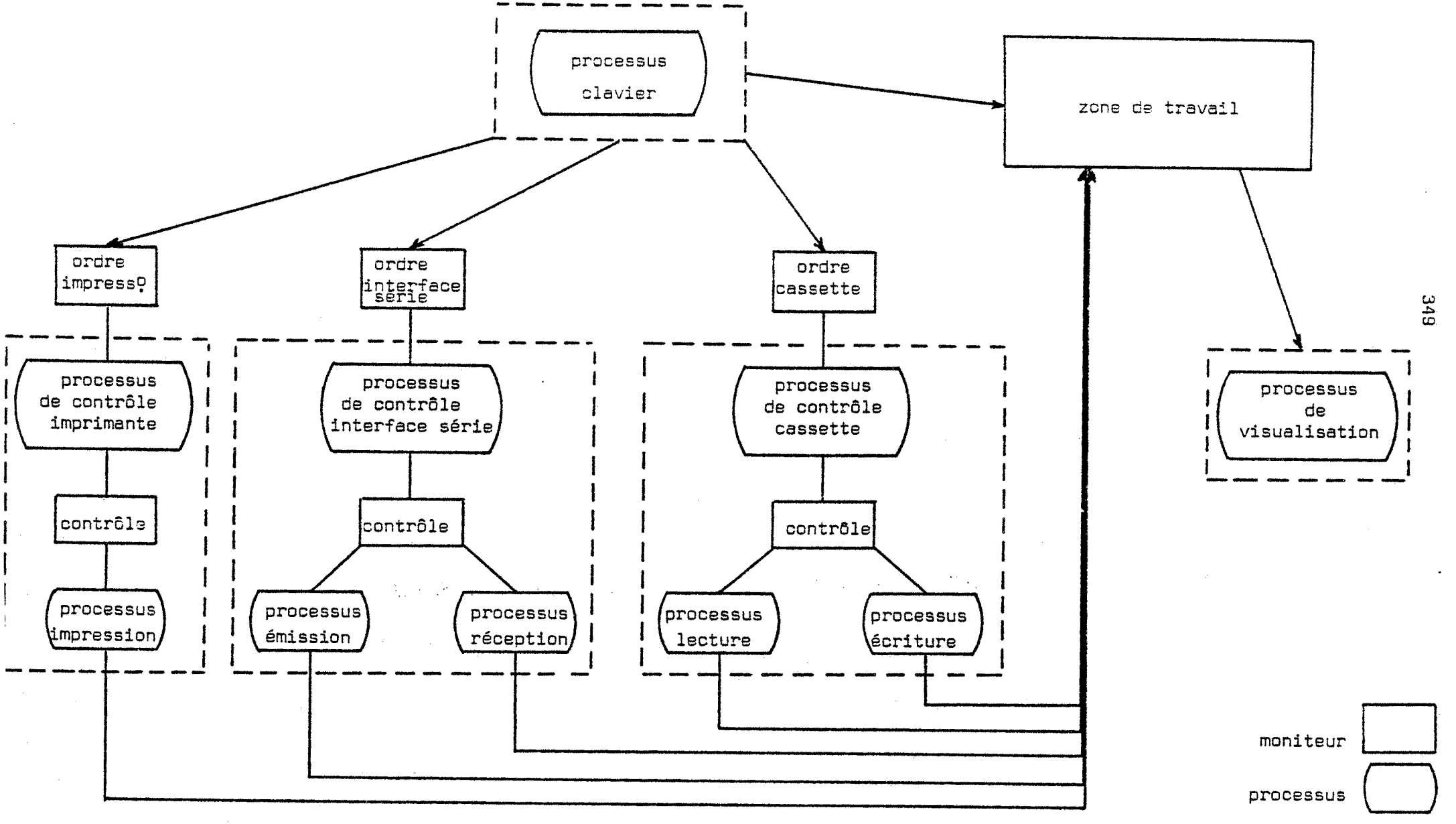
nombre de lignes E/S utilisées : 15.

Module impression : étant donné que l'on a considéré que la logique de contrôle de l'imprimante était dans le périphérique, ce module ne fait qu'établir une liaison série.

Surface : 2,5 mm x 3,2 mm

Nombre de lignes d'E/S utilisées : 4 (on n'a pas besoin des commandes pour des modems).

Figure 3 - Schéma fonctionnel de l'application



Module télé-informatique : c'est le même module qui a été décrit dans l'annexe 5.

Surface : 2,5 mm x 3,2 mm

nombre de lignes d'E/S utilisées : 7.

Module visu : on suppose l'utilisation d'un alphabet standard figé dans une ROM à l'intérieur du module, comme dans l'annexe 5.

Surface : 4,5 mm x 2,2 mm

nombre de lignes d'E/S utilisées : 4.

Module cassette : on a considéré que le module cassette contrôlait un périphérique du type de celui décrit dans la figure 4 ci-dessous [MFE-250B].

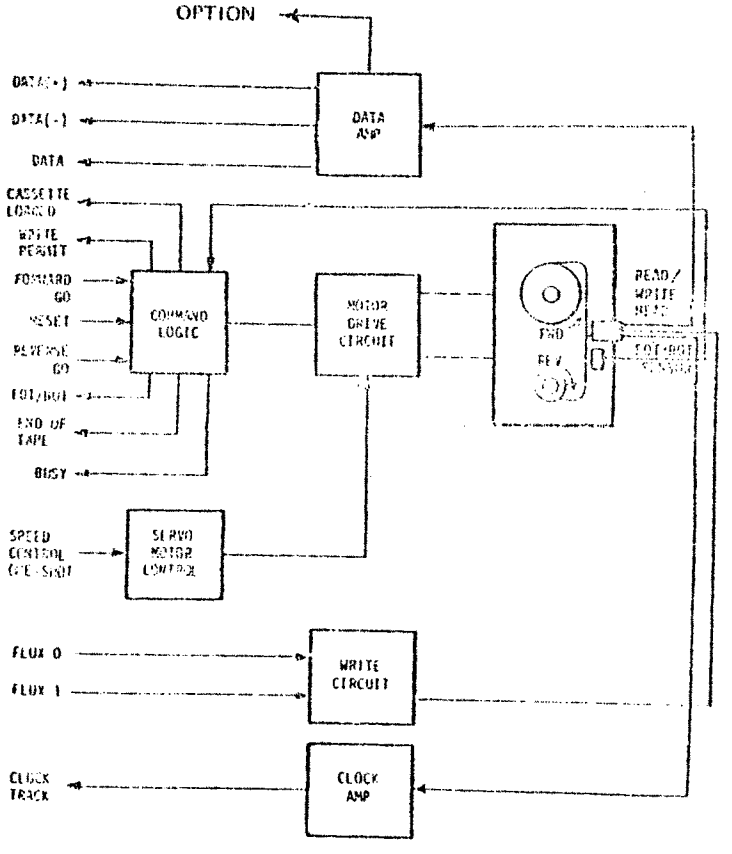


Figure 4 - Cassette

Etant donné que les algorithmes de gestion de la cassette sont très simples, on pense qu'un processeur du genre 8048 serait suffisant pour les exécuter.

Algorithmes de contrôle :

- contrôle du moteur, 4 sorties
  - forward go
  - reverse go
  - step control, one shot (contrôle de la vitesse)
  - reset
- détection de l'état de l'unité, 5 entrées
  - cassette loaded
  - write permit
  - EOT/BOT (marque optique sur la bande)
  - END of tape
  - busy
- lecture/écriture des caractères, 1 sortie ; 2 entrées
  - data (data + V data -)
  - flux (flux + V flux -)
  - clock track, (horloge)

Pour envoyer et recevoir des caractères en série, il manque au 8048 une interface série. Dans le 6801 on trouve une interface de ce type dont la surface est 1,34 mm^2.
Le module cassette aurait donc une surface de

20,76 + 1,34 = 22,1 mm^2 ≡ 4,5 mm x 4,91 mm

et 12 lignes d'E/S.

Noyau de communication :

5 moniteurs : ordre imprimante, 1 octet
ordre télé-informatique, 1 octet
ordre cassette, 4 octets (il faut indiquer le nombre de blocs à lire ou à écrire).

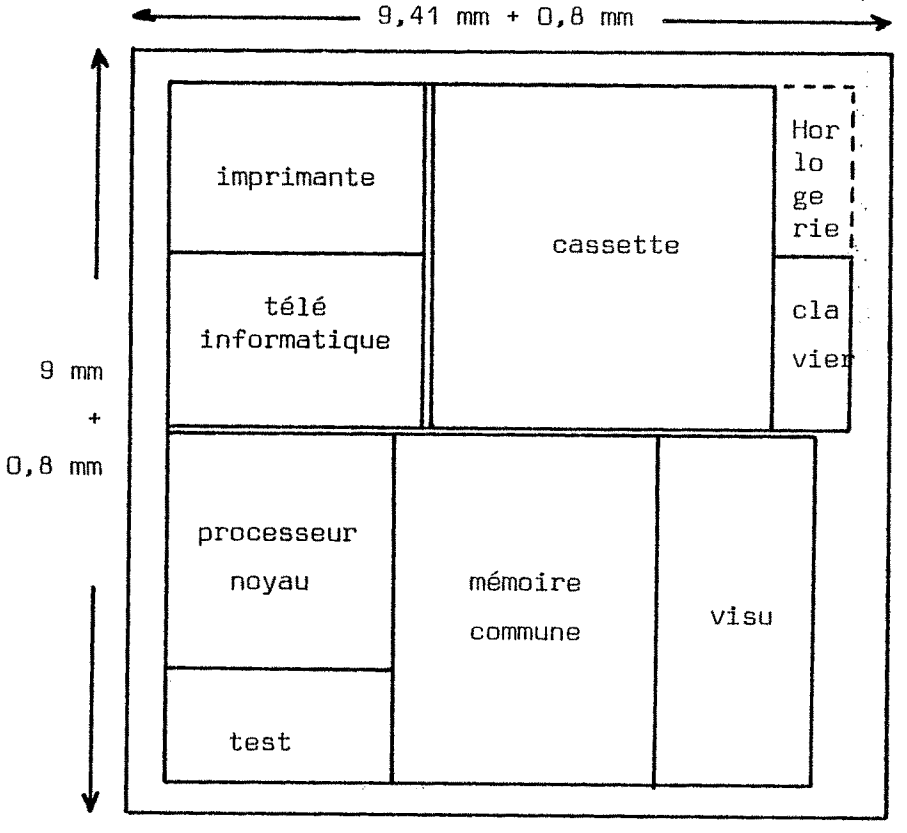
Zone de travail : ce moniteur a un très grand nombre de procédures et sa programmation constitue la partie la plus difficile de la définition du système. En résumé, les procédures utilisées sont :

- . écrire un caractère,
- . lire un caractère,
- . écrire un retour chariot ; signale au processeur noyau la fin d'une ligne de moins de 80 caractères,
- . écrire un EOT ; signale au processeur noyau la fin d'un bloc de moins de 2560 caractères,
- . lire position curseur,
- . remise à zéro des pointeurs.

Surface du noyau :

mémoire 2587 octets avec amplis = (2,5 x 1,37) x 4,5 = 3,425 x 4,5
processeur noyau + test = 3 x 4,5
total noyau = 6,43 mm x 4,5 mm

Pour la périphérie du circuit, on considère 0,4 mm de chaque côté, indépendemment de la technologie.



Etant donné que les dimensions sont imposées par les modules, on utilise les coefficients de la logique pour calculer la surface en HMOS et 2 µm.
On considère que la largeur de la bande de plots d'entrée/sortie et amplificateurs ne diminuera pas avec la technologie, et donc on rajoute 0,8 mm aux deux dimensions pour calculer la surface.

Surface en HMOS :

$$(6,4 + 0,8) \times (6,72 + 0,8) = 54,14 \text{ mm}^2$$

Surface en 2 µm

$$(4,5 + 0,8) \times (4,77 + 0,8) = 29,5 \text{ mm}^2$$

2. Circuit de réception et visualisation ANTIOPE

La description fonctionnelle de cette application a déjà été faite au § 2.3. et on l'a à nouveau utilisée pour expliciter la façon d'évaluer les performances du système au § 6.3. On ne va donc s'intéresser ici qu'à des aspects d'implantation.

Module de réception :

On ne dispose pas de l'évaluation de la surface de ce module. Dans une architecture multi-boîtiers [HEN 79] un circuit spécial est utilisé pour effectuer la réception des données. Dans cette architecture, un microprocesseur est utilisé pour faire l'interprétation du langage de la procédure de transmission. Ceci oblige le circuit spécial de réception à exécuter deux fonctions supplémentaires : la gestion d'un FIFO de 1024 caractères nécessaire pour adapter la fréquence de réception à la fréquence de travail du microprocesseur, et la gestion d'une interface de communication avec le bus microprocesseur.

Dans MOSAIC, le module de réception ne contiend que la partie qui exécute l'algorithme de réception, car le FIFO est implémenté par le processeur noyau.

Quoique l'algorithme de réception soit plus compliqué que celui d'un ACIA, surtout à cause de la détection des codes identiques à distance 1, on ne pense pas que ce module soit plus grand que le module de transmission série.

S = 2,25 mm x 3,2 mm

Module d'interprétation du langage :

Un module du type 6801 avec 2 K octets de ROM doit être suffisant pour exécuter l'algorithme d'interprétation.

S = 4,5 mm x 6 mm

Module clavier :

Identique à celui décrit au § 8.5.2.

S = 2,25 mm x 1,3 mm

Module de visualisation :

Pour l'évaluation de cette application, on a considérablement simplifié la partie de génération des caractères [HEN 80B]. Les deux simplifications les plus importantes ont été de ne pas considérer la possibilité d'utiliser un alphabet modifiable par l'utilisateur et de réduire la taille de l'alphabet de base à 128 formes alphanumériques et graphiques (au lieu de 128 formes alphanumériques + 96 formes graphiques). Dans ce cas il faut augmenter considérablement la taille du générateur de caractères du module de visualisation décrit dans l'annexe 5.

Générateur de caractères, 128 formes de 10 x 10 points

128 x 10 x 10 = 12800 points ROM

12800 x 0,00034 mm^2 = 4,35 mm^2

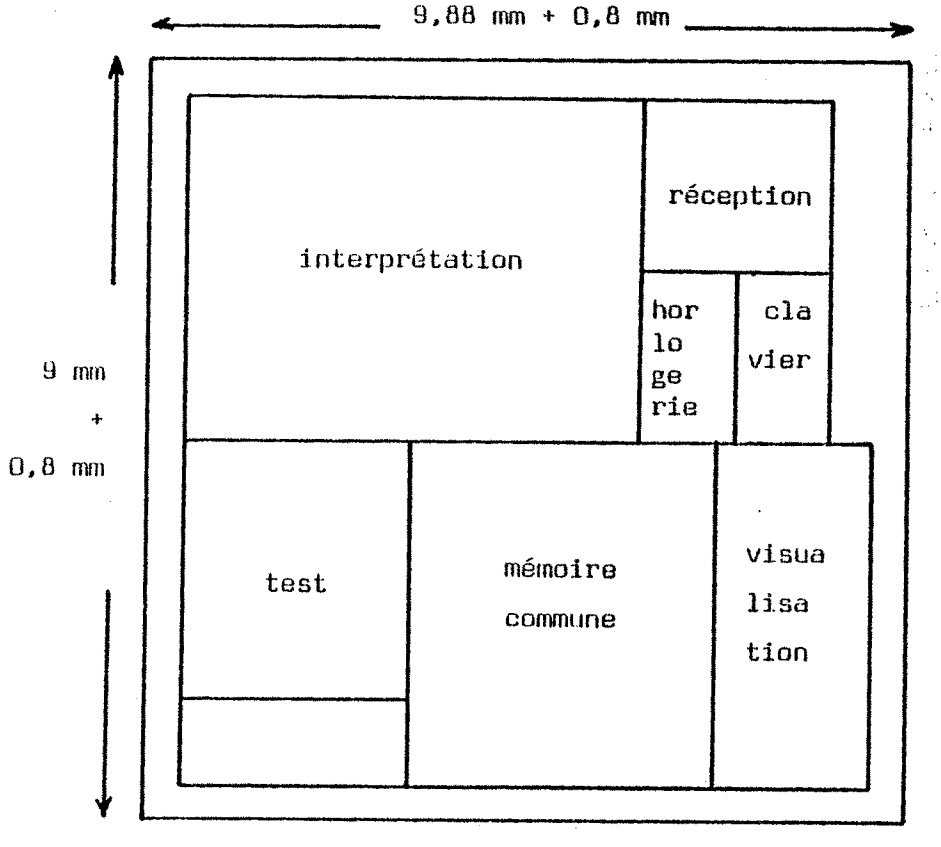
Le générateur de caractères précédent faisait 1,51 mm^2, l'augmentation de surface est donc : 4,35 mm^2 - 1,51 mm^2 = 3,02 mm^2

9,46 + 3,02 = 12,48 mm^2

4,5 x 2,77

la mémoire commune à 3 K octets = (3 x 1,37) x 4,5 = 4,11 x 4,5

avec le processeur noyau et le module de test = 7,11 x 4,5

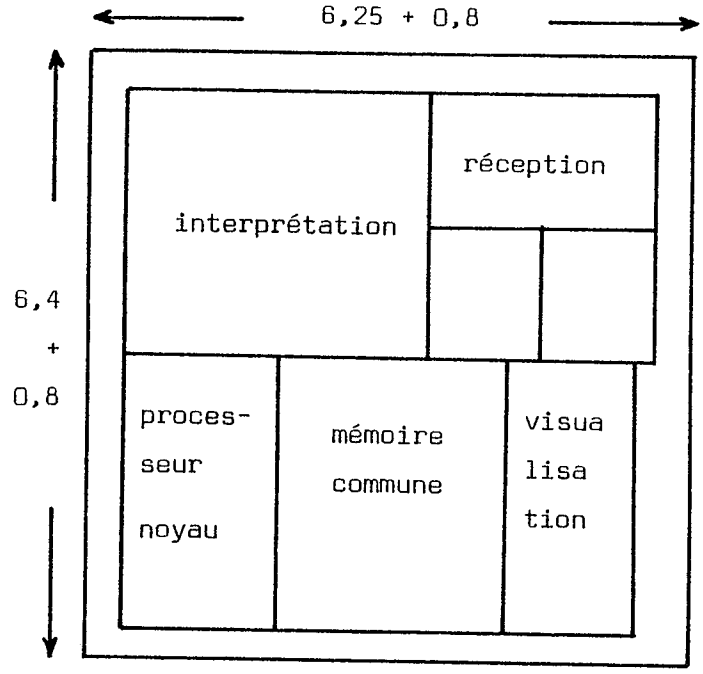


S = 9,8 x 10,68 mm^2

Réduction HMOS :

on applique les coefficients de l'annexe 3 pour la logique, mémoire dynamique et ROM. Il faut remarquer que la partie inférieure du circuit se réduit plus rapidement que la partie supérieure, car les coefficients de réduction de la mémoire dynamique et de la ROM sont plus élevés que celui de la logique .

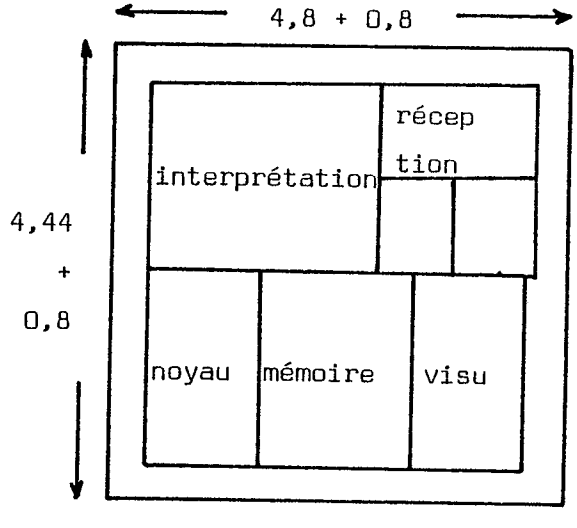
$7,05 \times 7,2 = 50,76\ mm^2$.



En 2 µm :

on applique à nouveau les coefficients de l'annexe 3 :

$5,29 \times 5,6 = 29,6\ mm^2$



dernière page de la thèse

AUTORISATION DE SOUTENANCE

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

VU les rapports de présentation de Messieurs :

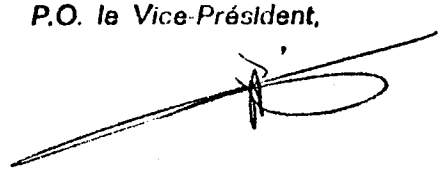
- F. ANCEAU, Professeur à l'Institut National Polytechnique de GRENOBLE
- P. JESPERS, Professeur à l'Université Catholique de LOUVAIN - BELGIQUE -
- J. BOREL, Directeur recherche et développement - EFCIS -

Monsieur José Manuel COSTA ALVES MARQUES

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

Grenoble, le 10 Septembre 1980

Le Président de l'I.N.P.G.

Ph. TRAYNARD
Président
de l'Institut National Polytechnique

P.O. le Vice-Président,