



**HAL**  
open science

# Un processeur pour la reconnaissance de la parole

Patrice Frison

► **To cite this version:**

Patrice Frison. Un processeur pour la reconnaissance de la parole. [Rapport de recherche] RR-0215, INRIA. 1983. inria-00076343

**HAL Id: inria-00076343**

**<https://inria.hal.science/inria-00076343>**

Submitted on 24 May 2006

**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.

**IRIA**

**CENTRE DE RENNES**

**IRISA**

Institut National  
de Recherche  
en Informatique  
et en Automatique

Domaine de Voluceau  
Rocquencourt  
BP 105  
78153 Le Chesnay Cedex  
France  
Tél.: (3) 954 90 20

Rapports de Recherche

N° 215

**UN PROCESSEUR  
POUR LA RECONNAISSANCE  
DE LA PAROLE**

**Patrice FRISON**

**Juillet 1983**

**IRIA**

**CENTRE DE RENNES  
IRISA**

Institut National  
de Recherche  
en Informatique  
et en Automatique

Domaine de Voluceau  
Rocquencourt  
B.P. 105  
78153 Le Chesnay Cedex  
France  
Tél. (3) 954 90 20

Rapports de Recherche

N° 215

**UN PROCESSEUR  
POUR LA RECONNAISSANCE  
DE LA PAROLE**

Patrice FRISON

Juillet 1983

UN PROCESSEUR INTEGRE POUR LA  
RECONNAISSANCE DE LA PAROLE

Patrice FRISON

RESUME : Les travaux exposés dans ce rapport concernent la mise en oeuvre d'un algorithme parallèle de détection de mots dans la parole continue sur une architecture systolique intégrée. Le multiprocesseur est composé d'un réseau de processeurs élémentaires connectés de manière régulière et locale. On décrit la structure de chaque processeur et sa réalisation en circuit intégré VLSI. Le circuit comporte environ 12000 transistors. La machine complète nécessite 89 circuits et devrait permettre la détection de vocabulaires de près de 2000 mots en temps réel.

SUMMARY : The work reported here is concerned with the design of an integrated systolic architecture of a parallel word spotting algorithm for continuous speech. The multiprocessor is composed of a network of elementary processors, regularly and locally connected. We describe each processor structure and its design in VLSI. Each circuit is made out of about 12000 transistors. The whole machine uses 89 chips and should be able to achieve real time word detection in a 2000 words vocabulary.

## Introduction

Les progrès considérables réalisés ces derniers temps dans le domaine de l'intégration à très grande échelle de circuits électroniques (VLSI) sont en train de modifier les rapports entre les concepteurs d'architectures et les concepteurs de circuits. En effet, il est désormais possible pour les premiers de réaliser des systèmes informatiques sur un seul circuit intégré comportant plusieurs dizaines de milliers de transistors.

Cette révolution technologique est intéressante à plusieurs titres. D'une part, l'architecte n'est plus tributaire des composants standards existants, et il lui est permis de concevoir son circuit intégré sur mesure. En particulier, il détient tous les atouts pour définir son circuit en fonction de ses besoins (fonctionnalité du circuit, vitesse, consommation, nombre de pattes du boîtier etc...). D'autre part, les circuits VLSI ont un rôle important à jouer dans le domaine des architectures parallèles. Des applications peu réalistes, il y a encore quelques années, pour des raisons diverses (coût, encombrement, temps de conception), sont maintenant dans le domaine du possible.

L'une des structures parallèles les plus directement concernées par les circuits VLSI, est classée sous le terme d'architecture systolique [1]. Une machine systolique est composée d'un réseau de processeurs élémentaires, connectés régulièrement et localement, à travers lequel circulent des flots de données qui interagissent lors de leur rencontre. On connaît actuellement de nombreux problèmes susceptibles d'être résolus sur une architecture de ce type.

Depuis plusieurs années, nous nous sommes intéressés au problème du parallélisme à travers une application spécifique: la reconnaissance de la parole continue. En effet, les derniers développements dans ce domaine ont montré qu'on est désormais capable de concevoir des systèmes de reconnaissances performants (plus de 95% de taux de reconnaissance). Cependant, ces systèmes sont en général peu exploitables en raison de leur temps d'exécution sur calculateurs classiques. Dans le système KEAL [2], par exemple, l'une des phases de la reconnaissance qui nécessite beaucoup de temps de calculs, concerne la détection des mots. Nos travaux [3] ont porté sur une modification de l'algorithme initial qui a permis de concevoir un algorithme systolique particulièrement performant. En effet, on estime pouvoir effectuer en temps réel la détection de mots appartenant à un vocabulaire d'un millier de mots.

Les travaux exposés dans ce rapport concernent la mise en oeuvre de l'algorithme parallèle de détection de mots sur une architecture systolique intégrée. Le processeur élémentaire du multiprocesseur a été conçu et les masques des circuits permettant la fabrication de la puce de silicium ont été dessinés.

Ce rapport comporte 6 parties. Le problème de la détection de mots dans un système de reconnaissance de parole est abordé dans la première partie et l'algorithme de détection est présenté. La seconde partie traite de la version systolique de l'algorithme et nous montrons comment fonctionne le réseau de processeurs. La troisième partie est dédiée à la mise en oeuvre de l'algorithme sur un multiprocesseur. L'architecture globale de cette machine est décrite. On s'intéresse dans une quatrième partie à l'architecture d'un processeur élémentaire. Les différents modules constituant le processeur sont présentés. La cinquième partie nous entraîne dans la structure interne des différents modules du processeurs. Enfin, nous montrons dans une sixième partie, comment on peut programmer la machine. Les programmes envisagés nous permettent de prévoir les performances de la machine.

## 1-L'algorithmme de detection de mots

### 1.1-Le probleme

La reconnaissance de la parole comprend en général plusieurs étapes. La première consiste à digitaliser le signal de parole et à en extraire ces paramètres acoustiques. Dans une seconde étape, appelée analyse phonétique, les sons élémentaires du langage sont extraits et identifiés. À l'issue de cette analyse, la phrase prononcée est transformée en une suite de  $N$  segments phonétiques. Notons  $(X(1), \dots, X(N))$  cette séquence. Chaque segment  $X(i)$  est lui-même constitué d'un ensemble de symboles phonétiques, qui représentent les phonèmes les probables pour ce segment. Dans une troisième étape, tous les mots du vocabulaire du langage choisi sont comparés à la phrase codée, chaque mot étant comparé à chaque sous-suite de la phrase codée. Cette étape, appelée détection de mots, est en général très longue.

La détection de mot consiste à rechercher dans la phrase codée, toutes les occurrences d'un mot donné sous sa forme phonétique. Soit  $(Y(1), \dots, Y(M))$  la suite de phonèmes représentant ce mot. Le problème se résume à comparer la suite  $Y$  avec toute sous-suite  $(X(k+1), \dots, X(k+P))$  extraite de  $X$ , et à évaluer la qualité de cette ressemblance.

Pour évaluer le degré de ressemblance entre un mot et une partie de la phrase codée, il doit être tenu compte des erreurs éventuellement commises par l'analyseur phonétique. Les erreurs considérées ici sont de trois types: des confusions, des insertions et des omissions. Une confusion apparaît lorsqu'un phonème prononcé  $z$  est traduit par un segment contenant plusieurs phonèmes et dans lequel  $z$  peut ne pas se trouver. Une insertion se produit lorsque l'analyseur phonétique détermine des segments en surnombre. Enfin un phonème peut être omis lorsque l'analyseur ne détecte pas le phonème et ne construit donc pas le segment correspondant.

### 1.2-Un exemple

La figure 1 donne un exemple de transcription phonétique d'une phrase prononcée, en l'occurrence: "liste des connecteurs". À la suite de l'analyse phonétique, la phrase a été scindée en 13 segments. On note que le segment  $X(1)$  a été inséré en début de la phrase. Parmi les omissions, on note le phonème  $t$  du mot "liste", le second phonème  $k$  du mot "connecteur", et enfin le phonème  $r$  de ce même mot en fin de phrase.

Dans cette phrase codée, le détecteur de mots trouvera par exemple, le mot "liste" entre les segments  $X(1)$  et  $X(5)$ , le mot "des" entre  $X(6)$  et  $X(7)$ , et le mot "connecteur" entre  $X(8)$  et  $X(13)$ . Mais il trouvera également, par exemple, les mots "piste" entre  $X(1)$  et  $X(5)$ , "vecteur" entre  $X(10)$  et  $X(13)$ .

### 1.3-L'algorithmme

Le principe de l'algorithmme de détection de mot est basé sur un modèle probabiliste de l'analyseur phonétique. Soit  $Y=(y(1), \dots, y(M))$  le mot à comparer à la phrase  $X=(X(1), \dots, X(P))$ . Supposons que le

Comportement:	I	C	C	C	O	C	C	C	C	C	C	C	O	C	C	O
Phrase prononcee:		l	i	s	t	æ	d	e	k	o	n	ɛ	k	t	æ	r
X(i):	1	2	3	4	5	6	7	8	9	10	11		12	13		
Transcription phonetique :	p t k	l j	i e y	s f t k	o æ ɛ	b d g	e y ɛ	p t k	o æ ɛ	n v ʌ	ɛ ɛ e		p t k	o æ		

Figure 1: Exemple de transcription phonetique



dernier phoneme  $y(M)$  soit un symbol special  $]$  appele marqueur de fin de mot. Au mot  $Y$  est associe un automate d'etat fini probabiliste (note AEFPP dans la suite du texte) qui modelise le comportement de l'analyseur phonetique lorsqu'il traite  $Y$ . La figure 2 decrit l'AEFP associe a un mot de 2 phonemes  $(y(1), y(2), ])$ . Cet AEFPP contient 4 etats,  $S(0), \dots, S(3)$ . L'etat  $S(i)$  decrit le comportement de l'analyseur phonetique lorsqu'il traite le phoneme  $y(i+1)$ . Trois possibilites s'offrent a l'analyseur:

- rester dans l'etat  $S(i)$  avec une probabilite  $P_i(y(i+1))$  et produire un phoneme  $x$  avec une probabilite conditionnelle  $q_i(x/y(i+1))$ : ce cas modelise l'insertion;
- entrer dans l'etat  $S(i+1)$  avec la probabilite  $P_c(y(i+1))$  et produire un phoneme  $x$  avec une probabilite conditionnelle  $q_c(x/y(i+1))$ : c'est le cas de la confusion;
- entrer dans l'etat  $S(i+1)$  avec la probabilite  $P_o(y(i+1))$ , sans produire de phoneme: c'est le cas de l'omission.

Il est clair que, dans le cas du marqueur de fin de mot,  $P_i(])=P_c(])=0$ , afin d'eviter que l'AEFP ne produise le marqueur. Supposons que chaque segment  $X(j)$  ne contient qu'un seul phoneme  $x(j)$ , il a ete montre [4] que la probabilite que  $Y$  ait ete prononce est egale a la probabilite que l'AEFP associe a  $Y$  produise la suite de phonemes  $x(1), \dots, x(P)$  en transitant de  $S(0)$  a  $S(M+1)$ . Soit  $L(i, j)$  la probabilite pour l'AEFP d'entrer dans l'etat  $S(i)$  apres avoir produit  $x(1), \dots, x(j)$ , alors  $L$  est caracterise par les formules suivantes:

$$(1) \quad L(i, j) = L_c(i, j) + L_i(i, j) + L_o(i, j)$$

avec

$$(2) \quad L_c(i, j) = L(i-1, j-1) \times P_c(y(i)) \times q_c(x(j)/y(i))$$

$$(3) \quad L_i(i, j) = L(i, j-1) \times P_i(y(i+1)) \times q_i(x(j)/y(i+1))$$

$$(4) \quad L_o(i, j) = L(i-1, j) \times P_o(y(i))$$

$L_c(i, j)$ ,  $L_i(i, j)$  et  $L_o(i, j)$  denotent respectivement les probabilites pour l'AEFP d'entrer dans l'etat  $S(i)$  apres confusion de  $x(j)$  et  $y(i)$ , insertion de  $x(j)$  ou omission de  $y(i)$ . Les equations (1) a (4) sont valides pour  $1 \leq i \leq M$  et  $1 \leq j \leq P$ , avec les conditions initiales suivantes:

$$(2.1) \quad (\forall i)(1 \leq i \leq M) \quad L_c(i, 0) = 0,$$

$$(\forall j)(1 \leq j \leq P) \quad L_c(0, j) = 0,$$

$$L_c(0, 0) = 1$$

$$(3.1) \quad (\forall i)(0 \leq i \leq M) \quad L_i(i, 0) = 0$$

$$(4.1) \quad (\forall j)(0 \leq j \leq P) \quad L_o(0, j) = 0$$

et finalement, pour completer le schema de recurrence:

$$(5) \quad (\forall j)(0 \leq j \leq P) \quad L(M+1, j) = L_o(M+1, j) = L(M, j)$$

qui derive du fait que  $P_c(])=0$ ,  $P_o(])=1$  et que  $S(M+1)$  n'a pas de transition d'insertion.

Dans le cas de segments contenant plusieurs noms de phonemes, les equations (2) et (3) doivent etre modifiees. Si l'on suppose que chaque segment  $X(j)$  contient  $n$  phonemes notes  $x(j, 1), \dots, x(j, n)$  avec

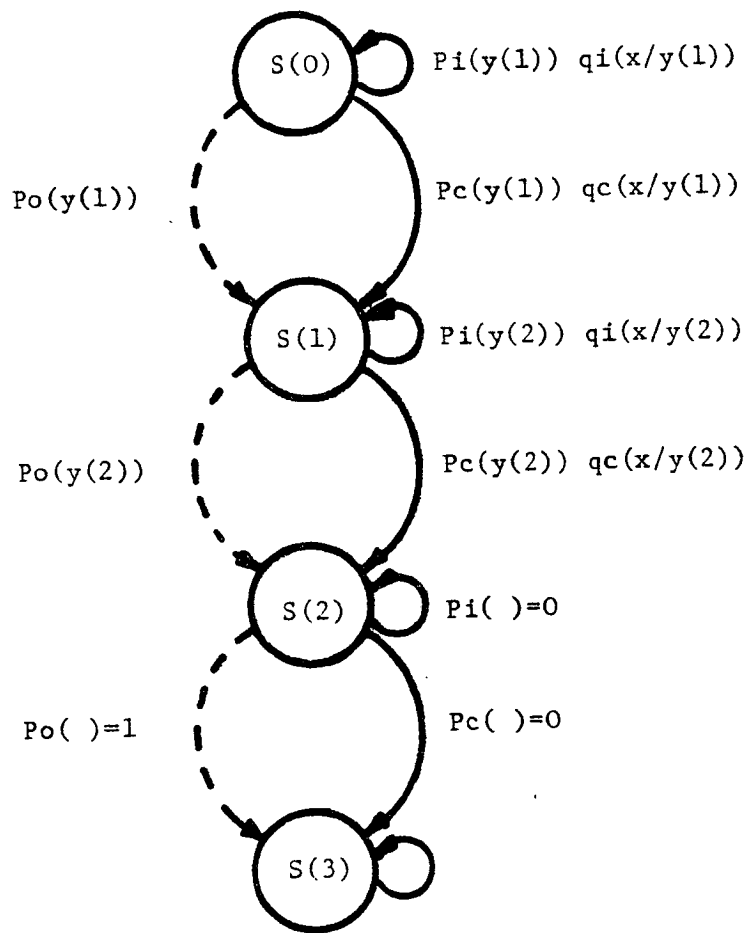


Figure 2: 1'AEPF associe au mot  $y(1).y(2)$

des probabilités respectives  $p(j,1), \dots, p(j,n)$ , les équations (2) et (3) sont modifiées en remplaçant les termes  $q_c(x/y)$  et  $q_i(x/y)$  par leur valeur moyenne sur  $X(j)$ , soient:

$$(6) \quad L_c(i,j) = L(i-1,j-1) \times P_c(y(i)) \times \prod_{k=1}^n p(j,k) \times q_c(x(j,k)/y(i))$$

$$(7) \quad L_i(i,j) = L(i,j-1) \times P_i(y(i+1)) \times \prod_{k=1}^n p(j,k) \times q_i(x(j,k)/y(i+1))$$

#### 1.4-Application à la détection de mots dans une phrase

Soit  $V = \{Y(1), \dots, Y(d)\}$  un vocabulaire de  $d$  mots à reconnaître et soit  $X = (X(1), \dots, X(N))$  la transcription phonétique d'une phrase prononcée. Le problème est de comparer chaque mot  $Y$  de  $V$  à chaque sous-suite  $(X(k+1), \dots, X(k+p))$  de  $X$  ( $0 \leq k \leq k+p \leq N$ ). Chacune de ces comparaisons fournit une vraisemblance  $L$ . Si  $L$  est une valeur suffisamment bonne (par exemple, si  $L$  dépasse un certain seuil), on dit que  $Y$  est détecté entre les segments  $X(k+1)$  et  $X(k+p)$ .

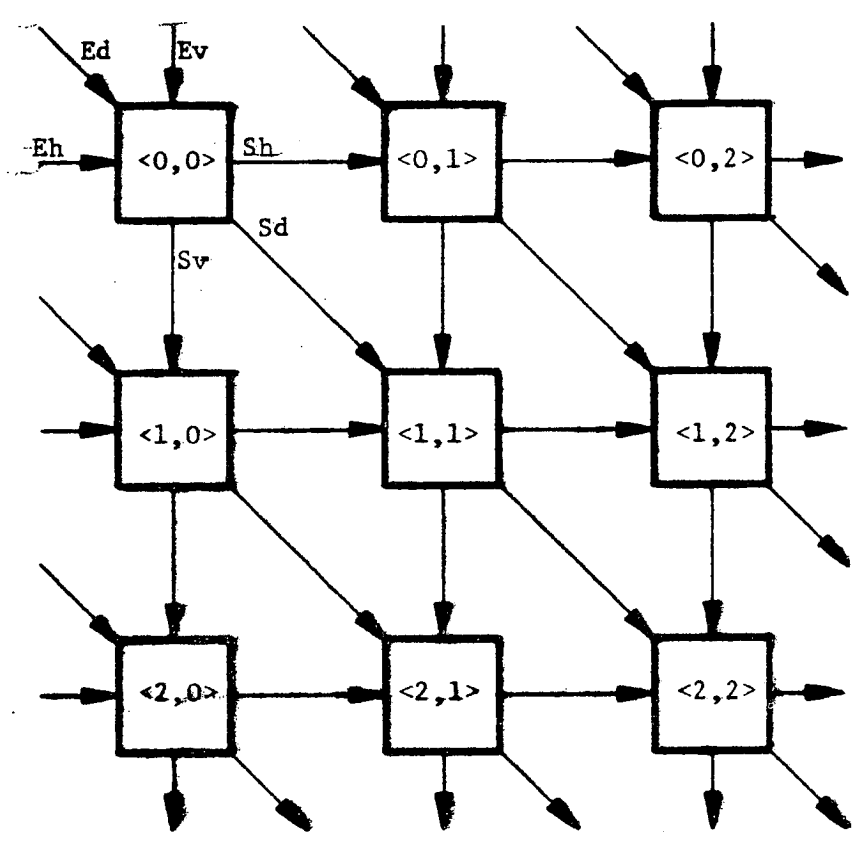


Figure 3: structure du reseau

## 2-Description du reseau de processeurs

### 2.1-Le reseau

L'idee de base consiste a attacher un processeur a chaque point  $(i,j)$ . Notons  $\langle i,j \rangle$  le processeur associe a  $(i,j)$ .  $\langle i,j \rangle$  recoit les valeurs  $L_c(i,j)$ ,  $L_o(i,j)$  et  $L_i(i,j)$ . Il calcule d'abord  $L(i,j)$  en appliquant l'equation (1) puis il calcule  $L_c(i+1,j+1)$ ,  $L_i(i,j+1)$  et  $L_o(i+1,j)$  en appliquant respectivement les equations (6), (7) et (4). Pour effectuer ces calculs,  $\langle i,j \rangle$  doit recevoir egalement les valeurs  $X(j+1)$  et  $y(i+1)$ . En consequence, le reseau est forme de  $(M+1) \times (P+1)$  processeurs  $\langle i,j \rangle$  avec  $0 \leq i \leq M$  et  $0 \leq j \leq P$ .  $\langle i,j \rangle$  doit etre connecte a  $\langle i-1,j-1 \rangle$ ,  $\langle i-1,j \rangle$  et  $\langle i,j-1 \rangle$  pour pouvoir calculer  $L(i,j)$ . La structure du reseau est representee sur la figure 3.

Cette structure montre que chaque processeur possede 6 ports d'entrees/sorties (3 ports d'entrees et 3 ports de sorties). Les lignes de connexion entre processeurs sont unidirectionnelles (leur direction est indiquee par les fleches). Pour chaque processeur les ports d'entrees/sorties sont notes comme suit:  $E_d$  (entree diagonale),  $E_v$  (entree verticale),  $E_h$  (entree horizontale) pour les entrees et  $S_d$  (sortie diagonale),  $S_v$  (sortie verticale),  $S_h$  (sortie horizontale) pour les sorties.

### 2.2-Fonctionnement du reseau

On examine tout d'abord le cycle de base d'un processeur, puis on montre comment le reseau peut etre utilise pour detecter un mot  $Y$  dans la suite de segments  $X$ . Enfin on decrit deux modes de fonctionnement pipeline du reseau qui permettent de resoudre completement le probleme.

#### 2.2.1-Cycle de base d'un processeur

Considerons le processeur  $\langle i,j \rangle$  et supposons que les donnees suivantes se trouvent disponibles sur ces ports d'entrees:

- sur  $E_v$ , successivement  $X(j+1)$  et  $L_o(i,j)$
- sur  $E_d$ ,  $L_c(i,j)$
- sur  $E_h$ , successivement  $Y(i+1)$  et  $L_i(i,j)$

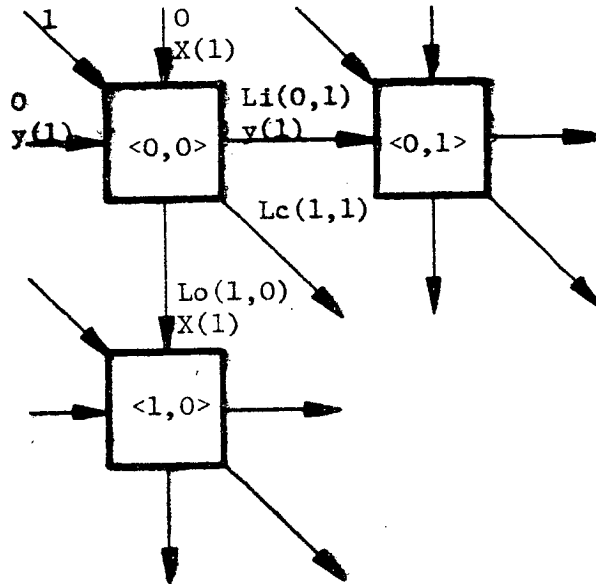
Pendant son cycle de base, le processeur calcule  $L(i,j)$  par application de l'equation (1) puis calcule  $L_i(i,j+1)$ ,  $L_c(i+1,j+1)$ , et  $L_o(i+1,j)$  en appliquant respectivement les equations (7), (6) et (4). Pour se faire, le processeur memorise d'une facon permanente les distributions de probabilite  $q_c(x/y)$ ,  $P_i(y)$  et  $P_o(y)$ . Le processeur  $\langle i,j \rangle$  emet ensuite les valeurs suivantes:

- $y(i+1)$  et  $L(i,j+1)$  sur  $S_h$ ;
- $L_c(i+1,j+1)$  sur  $S_d$ ;
- $X(j+1)$  et  $L_o(i+1,j)$  sur  $S_v$ .

Ce traitement est valide pour les processeurs interieurs i.e. les processeurs  $\langle i,j \rangle$  satisfaisant  $1 \leq i \leq M$  et  $1 \leq j \leq P$ . Les remarques suivantes montrent que le traitement decrit precedemment est egalement valide pour les processeurs des bords.

- les equations (2.1) et (3.1) definissent les valeurs initiales a envoyer aux processeurs  $\langle i,j \rangle$  avec  $i=0$  et  $1 \leq j \leq P-1$  ou  $j=0$  et  $1 \leq i \leq M$ .

ETAPE 0



ETAPE 1

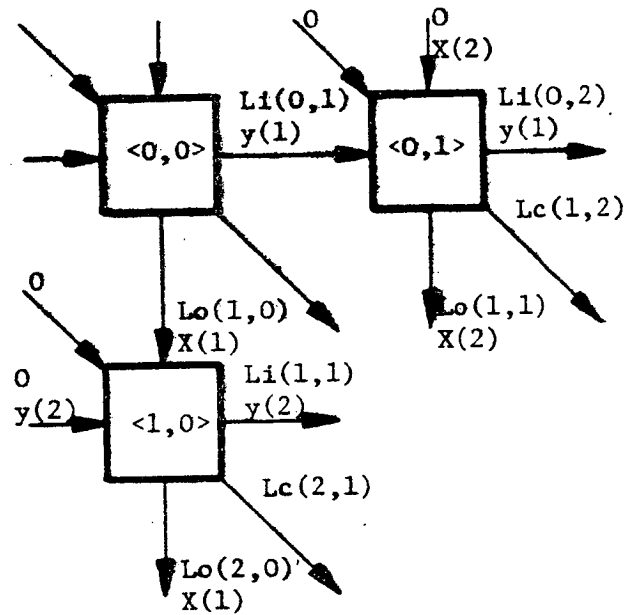


Figure 4: fonctionnement du reseau

- les processeurs  $\langle M, j \rangle$  avec  $1 \leq j \leq P-1$ , produisent  $L(M+1, j)$  sur leur port Sv, puisque  $Pc(\cdot) = \emptyset$ .
- enfin, considerons les processeurs  $\langle i, P \rangle$  ou  $0 \leq i \leq M$ . Afin que les processeurs se comportent exactement comme les autres, il est pratique d'introduire un segment fictif de fin de phrase  $X(P+1)$  contenant uniquement le marqueur de fin de mot  $\cdot$ . Si l'on pose:

$$(8) (\forall y) qc(\cdot/y) = qi(\cdot/y) = \emptyset$$

on montre que  $\langle i, P \rangle$  delivre  $Lo(i+1, P)$  sur son port Sv et donc que  $\langle M, P \rangle$  delivre  $L(M+1, P) = Lo(M+1, P)$  sur son port Sv.

### 2.2.2-Detection d'un mot

Supposons que le reseau fonctionne de facon synchrone. Aux instants successifs  $0, 1, \dots, t, \dots$  chaque processeur execute le calcul precedent. On veut utiliser le reseau pour detecter le mot  $Y = (y(1), \dots, y(M))$  dans la sequence  $X = (X(1) \dots X(P))$ . La figure 4 illustre le fonctionnement du reseau

Au temps  $t=0$ , le processeur  $\langle 0, 0 \rangle$  recoit  $X(1)$  et  $\emptyset$  sur Ev, 1 sur Ed et  $y(1)$  et  $\emptyset$  sur Eh.  $\langle 0, 0 \rangle$  execute son programme puis emet  $X(1)$  et  $Lo(1, 0)$  sur Sv,  $Lc(1, 1)$  sur Sd et  $y(1)$  et  $Li(0, 1)$  sur Sh.

Au temps  $t=1$ ,  $\langle 1, 0 \rangle$  recoit les valeurs envoyees par  $\langle 0, 0 \rangle$  sur Ev,  $\emptyset$  sur Ed,  $y(2)$  et  $\emptyset$  sur Eh. Il envoie ensuite  $X(1)$  et  $Lo(2, 0)$  sur Sv,  $Lc(2, 1)$  sur Sd et  $y(2)$  et  $Li(1, 1)$  sur Sh. Au meme moment,  $\langle 0, 1 \rangle$  recoit  $X(2)$  et  $\emptyset$  sur Ev, et  $\emptyset$  sur Ed et  $y(1)$  et  $Li(0, 1)$  sur Eh. Il produit ensuite  $X(2)$  et  $Lo(1, 1)$  sur Sv,  $Lc(1, 2)$  sur Sd,  $Y(1)$  et  $Li(0, 2)$  sur Sh.

Plus generalement, au temps  $t$ ,  $X(t+1)$  et  $\emptyset$  sont disponibles sur l'entree Ev du processeur  $\langle 0, t \rangle$  et  $\emptyset$  sur l'entree de  $\langle 0, t \rangle$ . De meme,  $Y(t+1)$  et  $\emptyset$  sont envoyes sur Eh de  $\langle t, 0 \rangle$  et  $\emptyset$  sur Ed de  $\langle t, 0 \rangle$ . La diagonale de processeurs  $\langle i, j \rangle$  avec  $i+j=t$ , delivre ensuite  $X(i+1)$  et  $Lo(i+1, j)$  sur Sv,  $Lc(i+1, j+1)$  sur Sd et enfin  $y(i+1)$  et  $Li(i, j+1)$  sur Sh.

Au temps  $t=M+P$ , on voit que la probabilite  $L(M+1, P)$  de correspondance entre X et Y apparait sur le processeur  $\langle M, P \rangle$ . On peut noter que le reseau fonctionne egalement pour la detection de mots de longueur l inferieure a M, a condition que ces mots soient completes par M-1 marqueurs de fin de mot. De meme, si la phrase a une longueur l inferieure a P, la transcription phonetique de cette phrase doit etre completee par P-1 marqueurs de fin de phrase.

### 2.2.3-Fonctionnement du reseau en mode pipeline

Supposons tout d'abord que les mots du vocabulaire ont une longueur inferieure ou egale a M. Nous avons vu qu'a l'instant t, seuls les processeurs  $\langle i, j \rangle$  avec  $i+j=t$  sont actifs. Ceci suggere une meilleure methode d'introduction des donnees dans le reseau afin d'effectuer des traitements a la chaine (ou pipeline). Deux modes de traitement pipeline sont possibles : le pipeline sur les entrees  $y(i)$  (appele aussi pipeline sur les mots) et le pipeline sur les entrees  $X(j)$  (appele aussi pipeline sur la phrase).

### 2.2.3.1-Pipe-line sur les mots

En fixant sur les entrees verticales du reseau les valeurs  $X(1), \dots, X(P)$  et en presentant successivement les mots  $Y(0), \dots, Y(d)$  sur les entrees horizontales, il est possible de comparer consecutivement chaque mot du vocabulaire  $V$ , avec la meme sequence  $X(1), \dots, X(P)$ . Ce mode de fonctionnement est illustre sur la figure 5.

A chaque cycle, le premier segment  $Y(1,1)$  d'un nouveau mot  $Y(1)$  est introduit dans le reseau vers le processeur  $\langle 0,0 \rangle$ . Les cycles suivants, les segments  $Y(1,2), Y(1,3), \dots, Y(1,M+1)$  sont envoyes respectivement sur  $\langle 1,0 \rangle, \langle 2,0 \rangle, \dots, \langle M,0 \rangle$  a raison de 1 par cycle processeur.

### 2.2.3.2-Pipe-line sur la phrase

En fixant le mot  $Y=(y(1), \dots, Y(M))$  sur les entrees horizontales et en decalant les segments de la phrase de gauche a droite sur les entrees verticales, il est possible de comparer successivement chaque sous-sequence de  $X(1), \dots, X(t)$  avec  $Y$ . La figure 6 illustre ce principe.

### 2.2.3.3-Combinaison des deux modes de pipeline

En combinant ces deux modes de pipeline, il est possible de detecter tous les mots de  $V$  dans une phrase  $X(1), \dots, X(N)$ . L'une des formes les plus naturelles consiste a comparer tous les mots du vocabulaire aux sous-suites commençant a  $X(1)$ , puis aux sous-suites commençant a  $X(2)$  et ainsi de suite. En effet, si l'on desire reconnaitre de la parole en temps reel, ce mode de combinaison des deux pipelines, permet le lancement du processus de detection des que les premiers segments de  $X$  sont produits par l'analyseur phonetique, le traitement s'operant au fur et a mesure de leur production. C'est ce mode de fonctionnement que l'on considere dans la suite du texte.



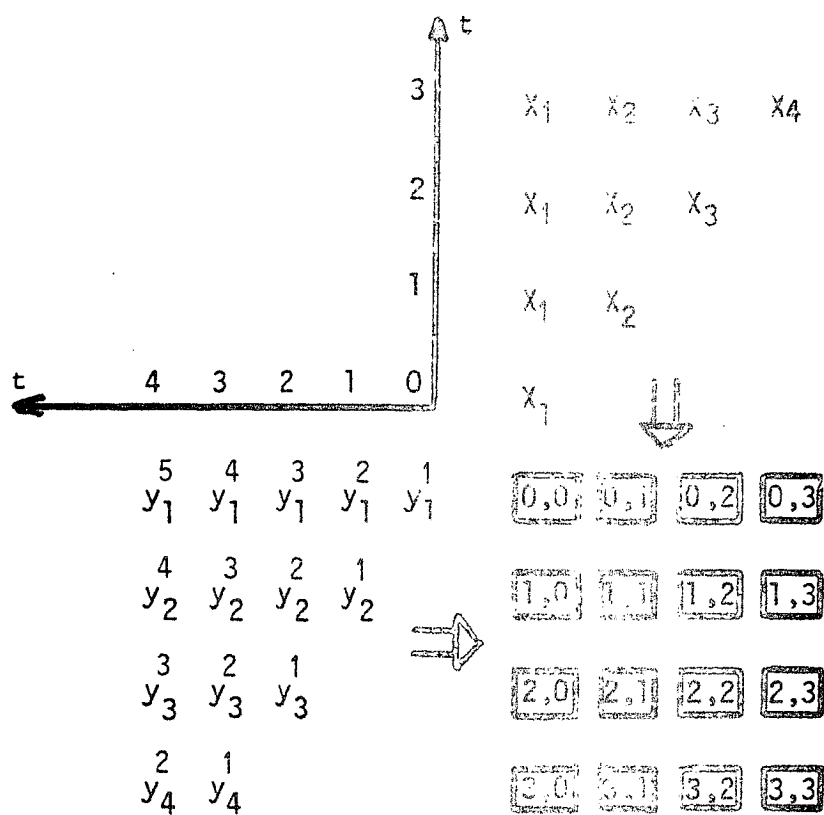


Figure 5: fonctionnement du réseau en mode pipeline sur les mots

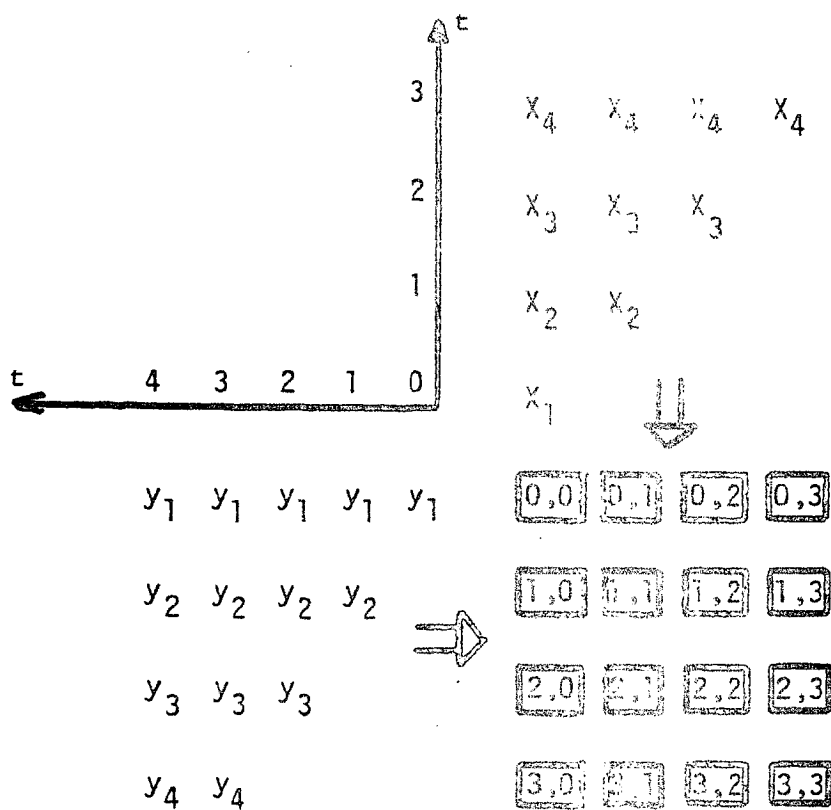


Figure 6: fonctionnement du réseau en mode pipeline sur la phrase

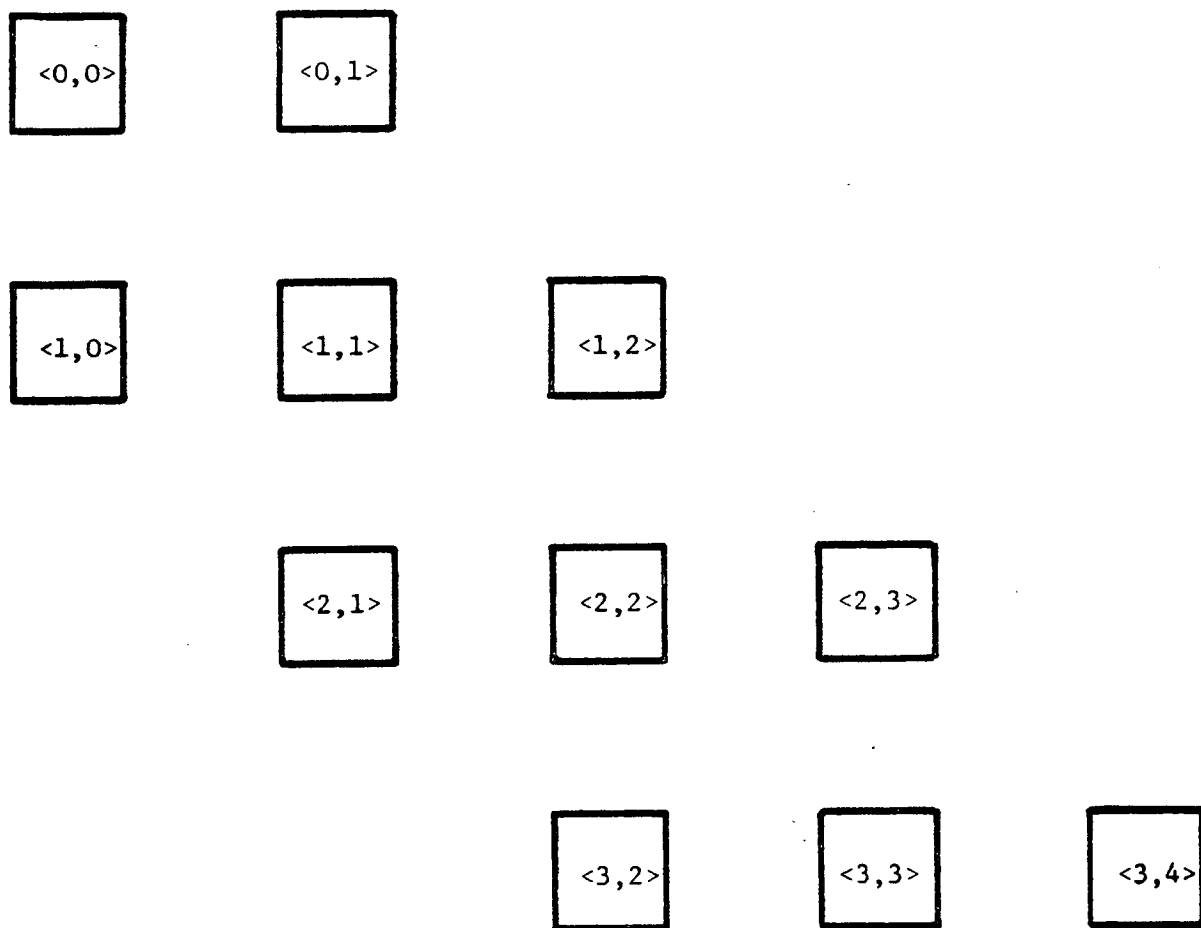


Figure 7: forme du reseau pour  $M=3$  et  $\delta=1$

### 3-Mise en oeuvre et architecture du reseau

Le but de ce paragraphe est de decrire l'architecture de processeurs capable de supporter la structure multiprocesseur presentee dans le paragraphe precedant. Trois points seront successivement abordes: le nombre de processeurs, le controle du systeme et les connexions entre processeurs.

#### 3.1-Taille du reseau

Si l'on suppose que toutes les correspondances entre un mot de longueur M et une phrase de longueur N sont vraisemblables, le nombre de processeurs est  $(M+1) \times (N+1)$ . Dans la pratique, on constate que les seules valeurs  $L(i,j)$  "proches" de la diagonale  $i=j$  sont significatives. Il est habituel de ne considerer que les valeurs  $(i,j)$  pour lesquelles:

$$\text{abs}(i-j) \leq \delta$$

ou  $\delta$  est une constante. Cela conduit a donner au reseau une forme telle que celle de la figure 7.

Les valeurs  $X(j)$  et  $Y(i)$  peuvent etre entrees directement sur les processeurs des bords de la diagonale a condition de leur appliquer un retard approprie. Les formules (3.1) fournissant les valeurs initiales des termes  $L_i$  et  $L_o$  doivent etre modifiees de la maniere suivante:

$$(3.2) \quad (\forall i), (\forall j), (i-j \geq \delta) L_i(i,j) = 0$$

$$(\forall i), (\forall j), (j-i \geq \delta) L_o(i,j) = 0$$

Un calcul simple montre alors que le nombre de processeurs du reseau est:

$$N_p = (M+1) \times (2\delta + 1) - \delta \times (\delta + 1) / 2$$

A titre indicatif, M peut etre fixe a 10 et  $\delta$  a 4 ce qui donne  $N_p = 89$  processeurs.

#### 3.2-Controle du systeme

Comme il a ete vu precedemment, le cycle de base d'un processeur se decompose en 3 parties: reception des donnees sur les ports d'entrees, calcul des valeurs  $L_c$ ,  $L_o$ ,  $L_i$  et envoi des donnees sur les ports de sorties. Les processeurs peuvent fonctionner independemment, en supposant par exemple que les communications interprocesseurs se font suivant un mecanisme de producteur-consommateur. Ceci entraine une structure de machine tres generale de type MIMD.

Or, il a ete montre que lors du traitement en mode pipeline sur les mots, les processeurs d'une meme diagonale traitent un meme mot et de plus, ils effectuent le meme programme (cycle de base). Cette remarque suggere le controle des processeurs d'une meme diagonale par un controleur unique. Celui-ci envoie un seul flot d'instructions executees simultanement par tous les processeurs d'une meme diagonale (mode de fonctionnement SIMD).

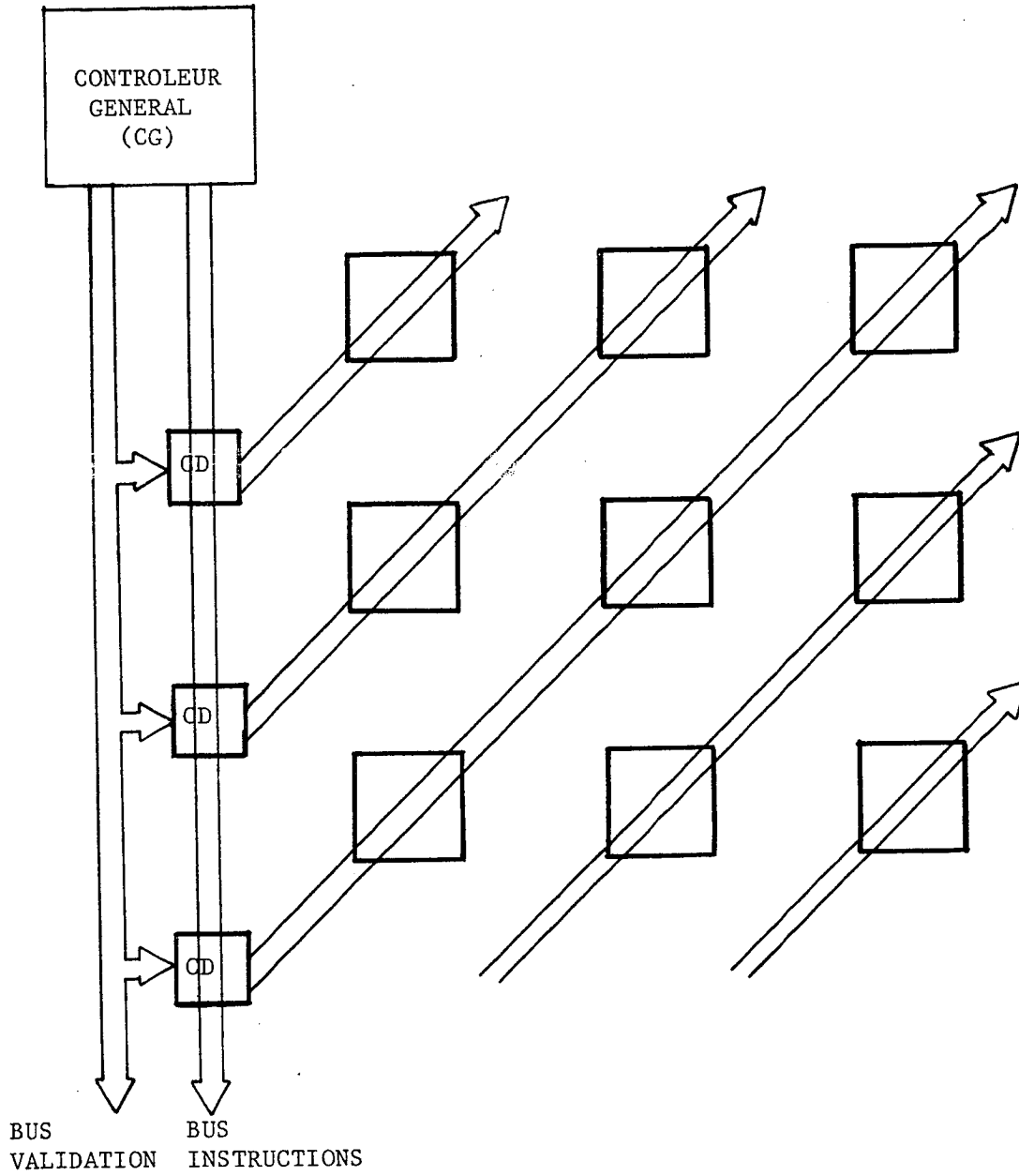


Figure 8: Organisation du controle de la machine

Par ailleurs, il est clair que les processeurs de deux diagonales successives executent le meme programme avec un certain decalage. En fixant ce decalage a un cycle processeur exactement, il est alors possible de controler les deux diagonales avec le meme controleur. Par consequent, on en deduit qu'un seul controleur (appele controleur general et note CG) est utilisable pour tout le reseau, tous les processeurs executant la meme instruction simultanement avec leur donnees propres. Cependant, il est necessaire de pouvoir inhiber certaines diagonales afin que les processeurs qui les composent soient inactifs, par exemple, lors du debut ou de la fin du pipeline sur les mots.

Dans ce but, a chaque diagonale est associee un controleur de diagonale (note CD). Ce controleur peut se trouver dans deux etats. Dans l'etat actif, il transmet aux processeurs de sa diagonale, les instructions qui lui sont envoyees par le controleur general. Dans l'etat inactif, il inhibe l'instruction du controleur, de telle sorte que les processeurs de la diagonale n'effectuent aucune operation. La figure 8 montre l'organisation du reseau de processeurs et de ces controleurs.

Deux bus sont controles par le controleur general: le bus d'instructions, sur lequel est codee l'instruction a executer et le bus de validation qui sert a activer ou desactiver un controleur de diagonale.

### 3.3-Connexions interprocesseurs

La structure du reseau necessite theoriquement que chaque processeur possede 6 ports d'entrees-sorties, (3 d'entrees et 3 de sorties). En realite, il est suffisant de ne mettre en oeuvre que 3 connecteurs d'entrees-sorties par processeur. Deux connecteurs d'entrees et un connecteur de sortie. La figure 9, montre comment les processeurs sont interconnectes physiquement.

Chaque processeur possede deux ports d'entrees notes V (entree verticale) et H (entree horizontale), et un port de sortie note S. Nous allons montrer comment cette organisation permet de realiser les differents transferts de donnees interprocesseurs necessites par l'algorithme.

#### 3.3.1-Transferts horizontaux et verticaux

Lorsqu'un processeur lit une donnee de type t, sur son port d'entree verticale (resp horizontale), le fonctionnement synchrone du reseau entraine que tous les processeurs lisent simultanement sur leur port V (resp port H) une donnee de type t et que, par consequent, tous les processeurs ont affiche sur leur port de sortie la valeur de type t qu'ils viennent d'utiliser ou de calculer.

#### 3.3.2-Transferts diagonaux

Dans le cas d'un transfert logique diagonal, on constate que l'information recue par un processeur doit etre memorisee pendant un cycle processeur avant de pouvoir etre utilisee. En effet, si un processeur emetteur appartient a la diagonale d, le processeur receveur

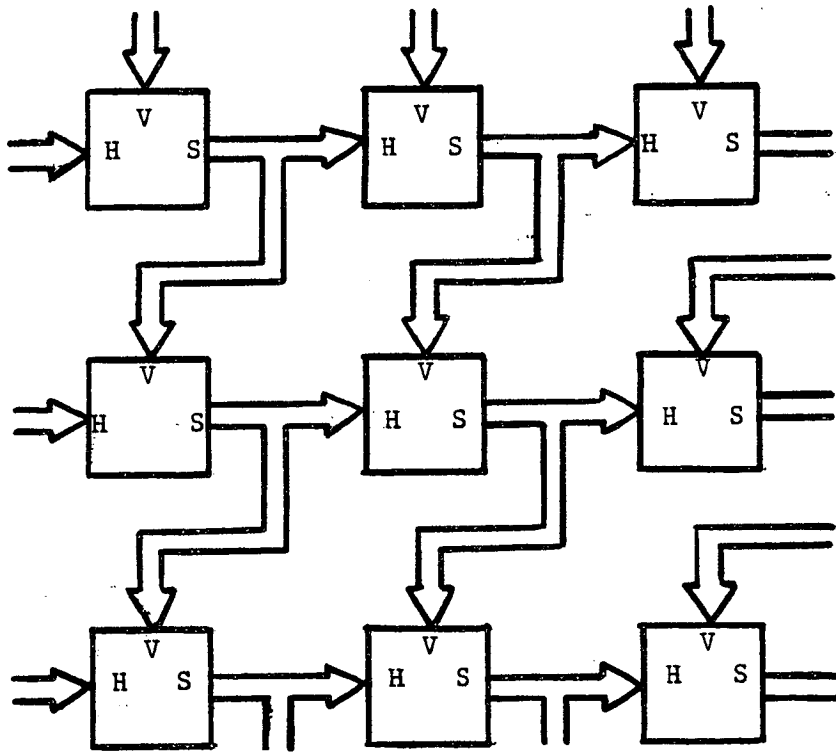


Figure 9: connexions interprocesseurs

appartient a la diagonale  $d+2$  (voir figure 3). Un transfert logique diagonal sera remplacé par deux transferts physiques: un transfert vertical, puis un transfert horizontal.

#### 4-Architecture d'un processeur

Dans le travail de conception de l'architecture d'un processeur, en vue de son integration, il faut essayer de calquer l'architecture du processeur sur l'algorithme qu'il doit effectuer, en ce sens que seules les fonctions necessaires doivent etre mises en oeuvre. Par ailleurs le processeur doit rester aussi simple que possible. Dans cette partie, on introduit les differentes parties du processeur. Dans un premier paragraphe nous reprenons en detail le programme a effectuer par un processeur, puis l'architecture globale qui en resulte est decrite dans un second paragraphe. Enfin, les differents modules du processeur sont detailles dans un dernier paragraphe.

##### 4.1-Le programme de base du processeur

Afin de definir les modules constituant le processeur, il est necessaire de preciser les fonctions qu'il doit realiser et donc definir precisement le programme qu'il doit effectuer. Pendant son cycle de base, le processeur  $\langle i, j \rangle$  calcule tout d'abord  $L(i, j)$  par application de l'equation (1) c'est a dire:

$$(1) \quad L(i, j) = L_c(i, j) + L_i(i, j) + L_o(i, j)$$

puis calcule  $L_i(i, j+1)$ ,  $L_c(i+1, j+1)$  et  $L_o(i+1, j)$  par application des equations (7), (6) et (4) soit:

$$(9) \quad L_i(i, j+1) = L(i, j) \times \prod_{k=1}^n p(j+1, k) \times q_i(x(j+1, k) / y(i+1))$$

$$(10) \quad L_c(i+1, j+1) = L(i, j) \times \prod_{k=1}^n p(j+1, k) \times q_c(x(j+1, k) / y(i+1))$$

$$(11) \quad L_o(i+1, j) = L(i, j) \times P_o(y(i+1))$$

Avant de pouvoir decrirer completement le programme du processeur, il nous faut definir la valeur de  $n$  ( $n$  indique le nombre de phonemes engendres par l'analyseur phonetique pour chaque segment de parole  $X(i)$ ). La valeur 3 a ete choisie pour  $n$  car elle correspond a la valeur generalement choisie.

Supposons que le processeur ait recu, de ses voisins, toutes les informations necessaires, et que ces valeurs soient rangees dans les variables suivantes:

PX1, PX2, PX3 :  $p(j+1, 1), p(j+1, 2), p(j+1, 3)$   
 Yi, Yc, Yo :  $P_i(y(i+1)), P_c(y(i+1)), P_o(y(i+1))$   
 X1, X2, X3 : nom des phonemes  $x(j+1, 1), x(j+1, 2), x(j+1, 3)$   
 Y : nom du phoneme  $y(i+1)$   
 Qc, Qi : tableaux contenant respectivement les probabilites de confusions et d'insertions  $q_c(x/y)$  et  $q_i(x/y)$

Par ailleurs, en supposant que les valeurs  $L, L_c, L_i, L_o$  sont calculees dans des variables de meme nom, le programme 1 suivant se deduit des equations (1), (9), (10) et (11).



```
debut
  L:=Lc+Li+Lo;
  Li:=LxYix(PX1xQi[X1,Y]+PX2xQi[X2,Y]+PX3xQi[X3,Y]);
  Lc:=LxYcx(PX1xQc[X1,Y]+PX2xQc[X2,Y]+PX3xQc[X3,Y]);
  Lo:=LxYo
fin
```

#### Programme 1

Afin de simplifier l'architecture du processeur, deux modifications ont été apportées au programme 1. L'une concerne la taille des tableaux à mémoriser dans le processeur, la seconde est liée à la représentation des données.

##### 4.1.1- Réduction de la taille des tableaux

Le programme 1 indique que 2 tableaux Qc et Qi doivent être mémorisés en permanence dans le processeur car ils contiennent des informations propres au système de reconnaissance et sont donc utilisables pendant tout le traitement d'une phrase. Soit F le nombre total de phonèmes du langage, exception faite du marqueur ], la taille de Qc et Qi est donnée par  $(F+1) \times (F+1)$ . Pour fixer les idées, en posant  $F=32$  et le nombre de bits par mot égal à 10, la taille cumulée des 2 tableaux vaut plus de 21k bits. Si l'on fait l'hypothèse que ces mémoires sont des mémoires vives, chargées dynamiquement en fonction du langage choisi ou du locuteur par exemple, cette taille de mémoire vive est trop importante pour que l'on puisse envisager l'intégration d'un processeur complet sur une puce de silicium, dans la technologie visée (NMOS 5 microns).

Cependant, dans le cas du traitement en mode pipeline sur les mots illustre par la figure 5, il apparaît que les processeurs d'une même colonne j reçoivent le même segment  $X(j+1)$  pendant que tous les mots du vocabulaire défilent sur les entrées horizontales du réseau. Cette remarque suggère de ne mémoriser dans les processeurs que les valeurs  $q_c(x/y)$  et  $q_i(x/y)$  pour toutes les valeurs de y mais pour 3 valeurs de x seulement, puisqu'un segment  $X(j)$  contient 3 noms de phonèmes ( $n=3$ ).

Par ailleurs, nous avons fait l'hypothèse que les probabilités conditionnelles  $q_i(x/y)$  étaient indépendantes de y. Finalement, le tableau Qc se réduit à un tableau de  $3 \times (F+1)$  éléments et le tableau Qi ne contient plus que 3 éléments seulement.

##### 4.1.2- Représentation des valeurs

Les calculs effectués par le processeur étant, en général, des produits de probabilités, les résultats s'étendent sur une large échelle entre 0 et 1. Par conséquent, l'utilisation d'opérations en virgule fixe est peu réaliste. Par ailleurs, l'utilisation d'opérations en virgule flottante risque d'entraîner de nombreux problèmes. La complexité des circuits, la taille du microprocesseur, la vitesse d'exécution des opérations en sont quelques exemples. L'utilisation des logarithmes permet de résoudre, en partie, ces problèmes (dans la suite du texte, les logarithmes à base 2 sont considérés).

Les multiplications, operations les plus utilisees, sont remplacees par des additions, qui ont l'avantage de s'effectuer rapidement et d'etre faciles a mettre en oeuvre.

Les additions, par contre, sont plus difficiles a mettre en oeuvre. La methode utilisee pour calculer  $\text{Log}(a+b)$  en fonction de  $\text{Log}(a)$  et  $\text{Log}(b)$  est la suivante. Sachant que:

$$(12) \quad \begin{aligned} \text{Log}(a+b) &= \text{Log}(a) + \text{Log}(1+b/a) \\ &= \text{Log}(b) + \text{Log}(1+a/b) \end{aligned}$$

le probleme se reduit a calculer la valeur  $\text{Log}(1+u)$  connaissant  $\text{Log}(u)$ . Posons Z la fonction telle que:

$$(13) \quad Z: t \rightarrow Z(t) = \text{Log}(1+2^t)$$

l'equation (12) devient:

$$\begin{aligned} (14.1) \quad \text{Log}(a+b) &= \text{Log}(a) + Z(\text{Log}(b) - \text{Log}(a)) \\ (14.2) \quad &= \text{Log}(b) + Z(\text{Log}(a) - \text{Log}(b)) \\ (14.3) \quad &= \max(\text{Log}(a), \text{Log}(b)) + Z(-\text{abs}(\text{Log}(a) - \text{Log}(b))) \end{aligned}$$

Les equations (14.1) et (14.2) montrent que  $\text{Log}(a+b)$  peut etre calcule de 2 facons differentes. La formule (14.3) tire profit de ce resultat et montre, en particulier, que l'espace de depart de la fonction Z peut etre reduit a l'ensemble des valeurs negatives. Cette remarque est tres importante, car comme nous le verrons par la suite, la fonction Z sera tabulee.

Le programme 2 suivant, est la version modifiee et simplifiee du programme 1. Les variables T1, T2 et T3 ont ete introduites pour memoriser certains resultats intermediaires. La fonction SOMLOG, recoit deux parametres en entree: P1= $\text{Log}(a)$  et P2= $\text{Log}(b)$  puis calcule  $\text{Log}(a+b)$ .

```

debut
{ calcul de L }
  L:=SOMLOG(SOMLOG(Li,Lc),Lo);
{ calcul de Li }
  T1:=PX1+Qi[1];
  T2:=PX2+Qi[2];
  T3:=PX3+Qi[3];
  Li:=L+Yi+SOMLOG(SOMLOG(T1,T2),T3);
{ calcul de Lc }
  T1:=PX1+Qc[1,Y];
  T2:=PX2+Qc[2,Y];
  T3:=PX3+Qc[3,Y];
  Lc:=L+Yc+SOMLOG(SOMLOG(T1,T2),T3);
{ calcul de Li }
  Lo:=L+Yo;
fin

```

Programme 2

#### 4.1.3-Codage des valeurs

Il a ete choisi de coder les logarithmes des probabilites sur 16 bits en complement a 2, afin d'une part, de conserver une bonne

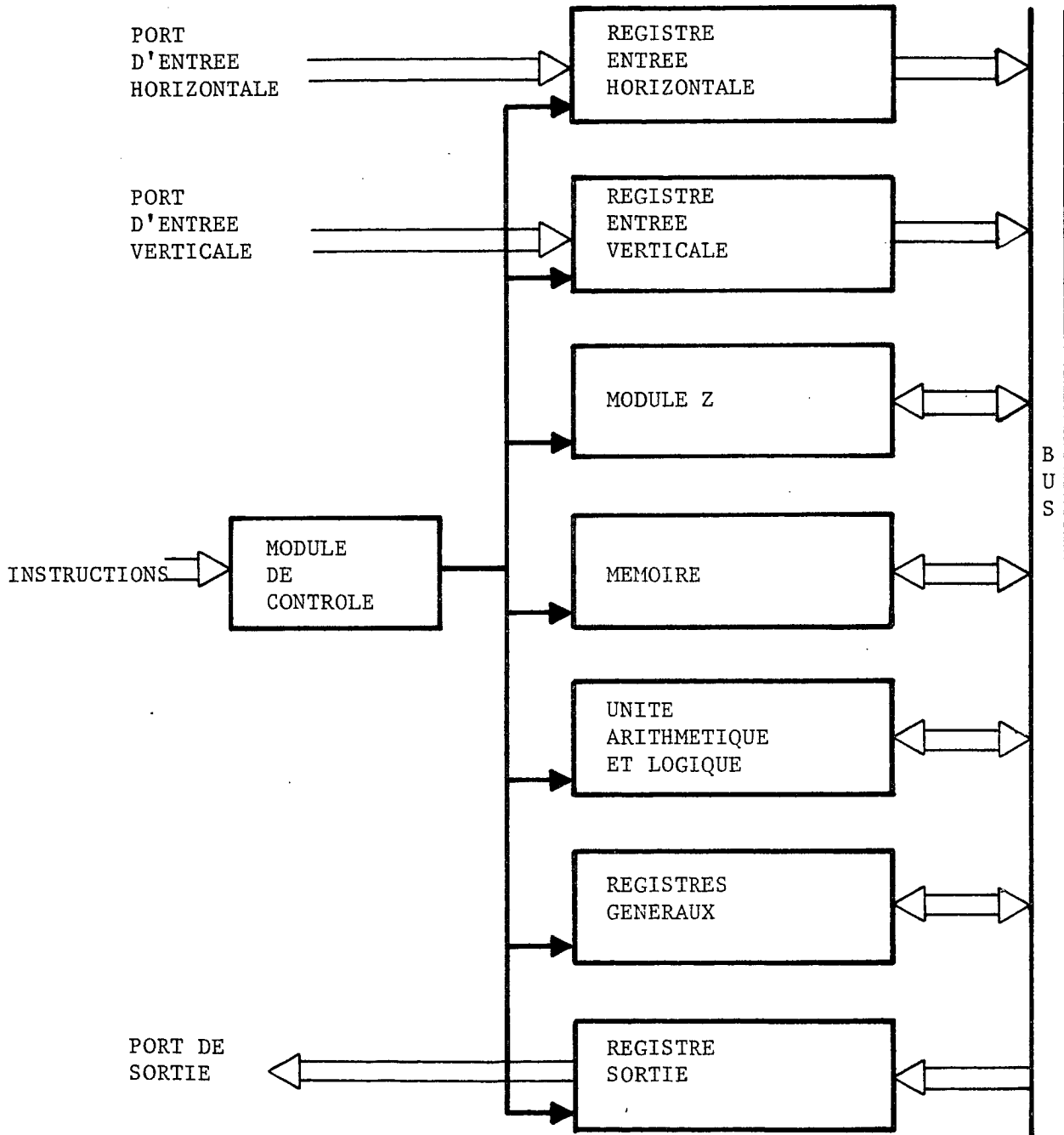


Figure 10: architecture d'un processeur

precision sur les calculs, et d'autre part, de coder les valeurs sur une large etendue. Un mot de 16 bits est compose de 2 parties. Les 9 bits de poids forts constituent la partie entiere de la valeur; les 7 bits de poids faibles forment la partie fractionnaire. Ceci permet le codage des probabilites avec une bonne precision puisque 2 valeurs codees consecutives  $u$  et  $v$  ( $u < v$ ) representent 2 probabilites  $p_u$  et  $p_v$  dont le rapport  $p_u/p_v$  est constant et vaut 0.9946. A noter que la valeur de probabillite 0 n'est pas codable, elle est cependant approchee par la valeur notee Min qui vaut  $2^{*-256}$ , soit environ  $10^{*-77}$ .

#### 4.2-Structure interne du processeur

Le processeur est constitue de 4 modules principaux:

- un module memoire ou sont stockees les valeurs de probabilites de confusion  $Q_c(x/y)$ ;
- une unite arithmetique et logique capable d'effectuer des operations elementaires telles que soustraction, addition, incrementation;
- un tableau de registres generaux servant de memoire de travail;
- un module, note Z, utilise pour tabuler la fonction Z.

Ces modules sont organises autour d'un bus unique comme l'illustre la figure 10. Egalement connectes au bus, se trouvent 3 registres specialises, notes RS (registre sortie), RH (registre entree horizontale), RV (registre entree verticale). Ils permettent au processeur de communiquer avec l'exterieur.

Un module de controle fait egalement partie du processeur. Son role est de recevoir les instructions envoyees par le controleur CG, de les decoder et d'envoyer les commandes ainsi engendrees vers les differents modules.

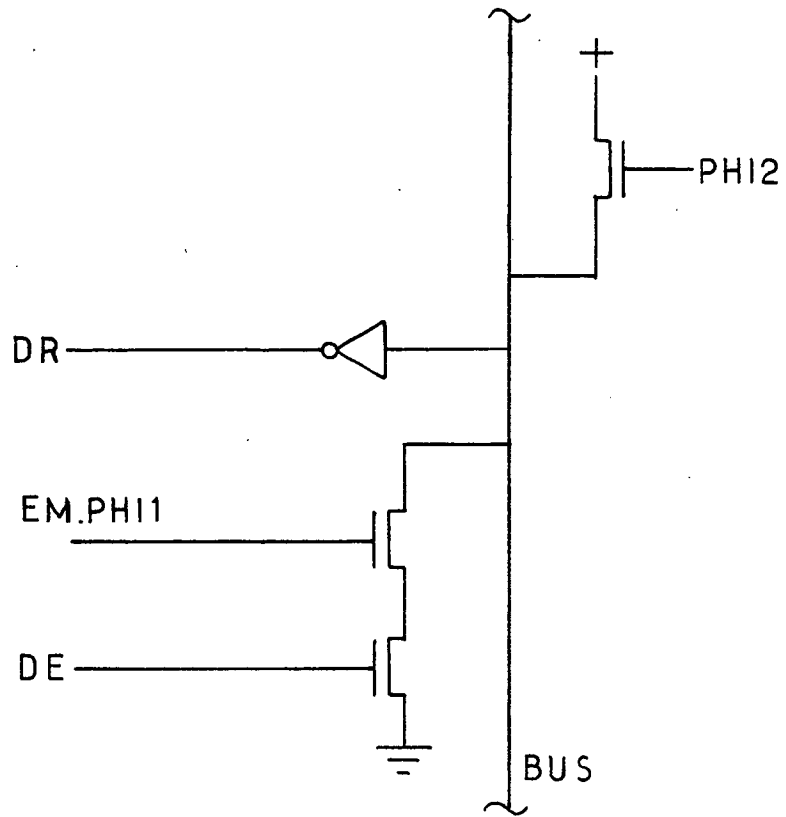


Figure 11: principe de fonctionnement du bus

## 5-Structure interne des modules du processeur

Dans cette partie, les différents modules constituant le processeur sont étudiés en détail. La technologie choisie est la technologie NMOS. Les circuits sont synchronisés par une horloge. L'horloge est composée de deux phases non recouvrantes notées PHI1 et PHI2. Pour des raisons de typographie, nous noterons Xbar le complément d'un signal X.

On étudie dans un premier paragraphe, la structure du bus sur lequel sont connectés la plupart des modules. Dans les paragraphes suivants, on étudie successivement la mémoire, le module Z, l'unité arithmétique et logique, le tableau de registres et enfin, le module de contrôle.

### 5.1-Le bus

Le bus permet le transfert d'information d'une unité à l'autre. Deux critères ont présidé à sa conception. D'une part, il doit être rapide et d'autre part, il doit pouvoir fonctionner avec un nombre quelconque de modules. Son fonctionnement repose sur la technique du préchargement: pendant la phase PHI2, le bus est préchargé, puis pendant la phase PHI1 suivante, l'information à transmettre est envoyée sur le bus. Le principe de fonctionnement du bus est illustré sur la figure 11.

Pendant PHI2, le bus, agissant comme une capacité, est chargé. Puis pendant PHI1, si la commande d'émission (notée EM) est active, la donnée à émettre (notée DE) produira l'un des effets suivants. Si DE vaut 1, le bus est déchargé; si DE vaut 0, le bus reste chargé. Il apparaît donc que le bus contient l'information DE complétée ( $BUS=DEbar$ ). Par conséquent, la donnée reçue par un module (notée DR), doit être égale à la donnée du bus complétée ( $DR=BUSbar$ ).

- Cette structure de bus répond aux spécifications citées plus haut:
- la rapidité est assurée. En effet, dans le cas le plus défavorable ou l'information du bus doit être 0, cette valeur est valide très peu de temps après le début de PHI1 car le déchargement du bus est une opération très rapide.
  - il est clair que cette organisation permet la connexion d'un nombre quelconque de modules sur le bus.

### 5.2-La mémoire

Elle contient les valeurs de probabilités de confusion  $q_c(x/y)$  pour 3 valeurs de x et F (nombre de phonèmes) valeurs de y. Cependant, on remarque que le tableau  $Q_c$  est relativement creux puisque  $Q_c[i,j]=0$  si i et j ne sont pas des phonèmes de même type (consonne ou voyelle). Ceci suggère une organisation particulière de la mémoire afin d'exploiter ce phénomène. La mémoire est adressée par un triplet (A,B,C) défini de la façon suivante:

- A prend ces valeurs dans l'intervalle  $[0,19]$  et indique le numéro d'un phonème y dans son type. Les phonèmes se répartissent en 12 voyelles et 20 consonnes dans la langue Française.
- B prend ces valeurs dans l'intervalle  $[0,2]$  et désigne l'un des trois

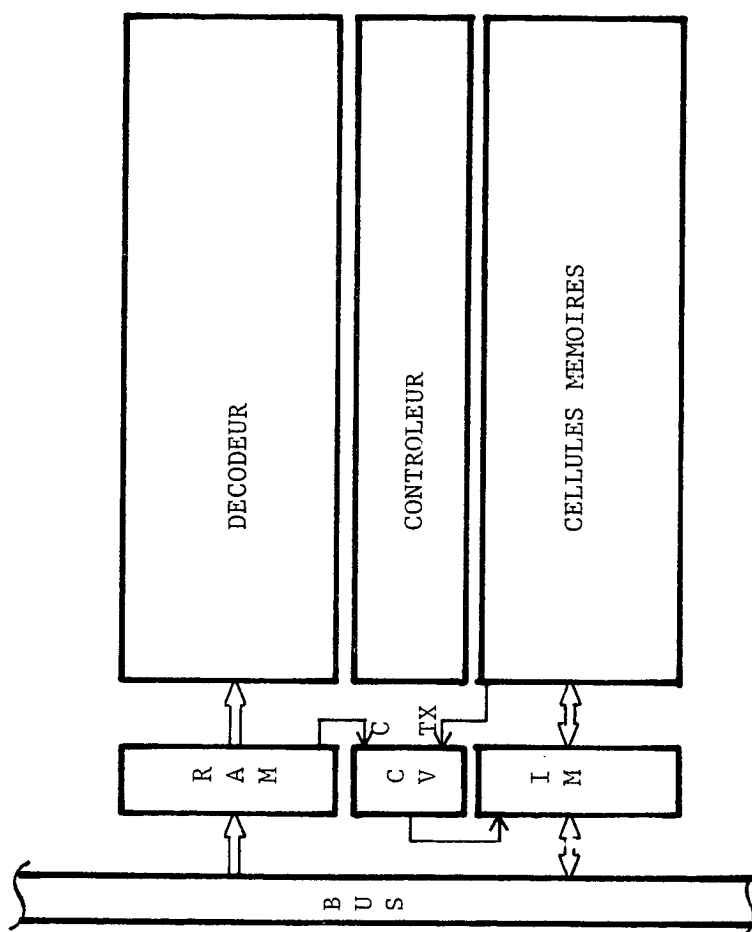


Figure 12: architecture de la memoire

phonemes x;  
- C vaut 0 ou 1 selon que y est une voyelle ou bien une Consonne.

La memoire est divisee en 3 blocs de 20 mots chacuns. La valeur B sert a designer l'un des 3 Blocs, la valeur A indique l'Adresse d'un mot a l'interieur d'un bloc. Chaque mot contient 2 informations. La premiere, notee Q, represente une valeur de confusion  $q_c(x/y)$ . Afin de reduire la taille de la memoire, Q est code sur 11 bits, 4 bits pour la partie entiere et 7 bits pour la partie fractionnaire, ce qui donne une precision suffisante pour le codage des probabilites de confusion. Lorsqu'une valeur memoire est transferee sur le bus, elle est completee par 5 uns sur les poids forts. La seconde, codee sur 1 bit, note TX, donne le type du phoneme x.

Lorsqu'une lecture memoire est executee avec le triplet (A,B,C), le bit TX du mot lu est compare au bit C. Si TX et C sont egaux, la lecture est valide. Si par contre, TX et C sont differents, la valeur lue est invalide, et doit etre remplacee par la valeur Min.

La memoire est constituee de 5 parties principales: un tableau de cellules memoires (note CM), un registre adresse (note RAM), un decodeur (note DC), un controleur (note CT), une interface memoire (note IM). L'architecture de la memoire est schematisee sur la figure 12.

Les differentes parties constituant la memoire sont detailles dans les paragraphes suivants.

#### 5.2.1-Le tableau de cellules memoires

Les cellules memoires sont organisees en un tableau de 12 lignes par 60 colonnes. Chaque colonne represente un mot memoire. Le circuit d'une cellule memoire est represente sur la figure 13.

Cette cellule est une cellule memoire dynamique a 3 transistors (notes T1, T2, T3). La capacite associee a la grille du transistor T2 sert a memoriser une information elementaire. Un seul bus est utilise pour transfere les informations de ou vers la cellule memoire. Le bus est precharge pendant PHI1, les operations de tranfert s'effectuent pendant PHI2. Les commandes de lecture et d'ecriture associees a la cellule, doivent etre des signaux d'horloge synchronises avec PHI2.

L'ecriture d'une valeur dans une cellule est effectuee de la facon suivante. L'information a stocker est presentee sur le bus, et la commande d'ecriture (notee ECR) est activee. Le transistor T1 devient passant et charge la capacite associee a la grille du transistor T2, avec la tension se trouvant sur le bus. Par consequent, la cellule memorise l'information se trouvant sur le bus. A noter que cette methode peut donner naissance a un probleme de partage de charge (charge sharing) entre le bus et la capacite associee au bit memoire. En effet, si l'une seulement des capacites est chargee, l'operation d'ecriture provoque un partage de charge entre les deux capacites. Cependant, le bon fonctionnement du systeme est assure a condition que la capacite associee au bus soit tres superieure a celle de la grille du transistor T2. Nous avons pu verifier que cette condition etait valide, dans la mise en oeuvre qui a ete faite.



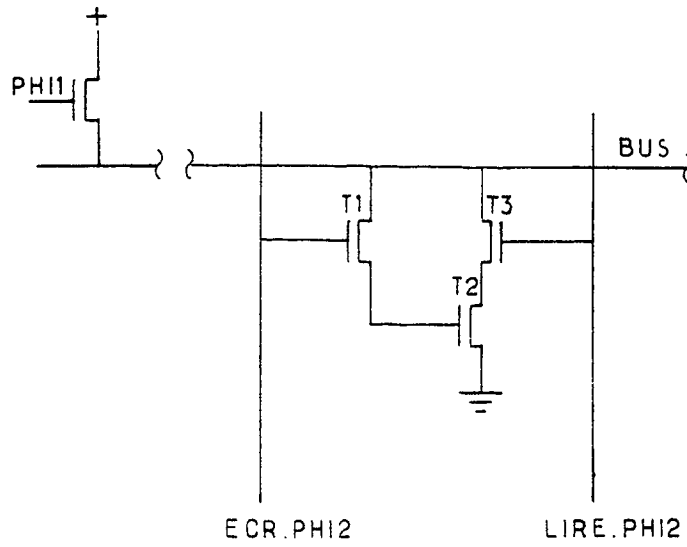


Figure 13: circuit d'une cellule memoire

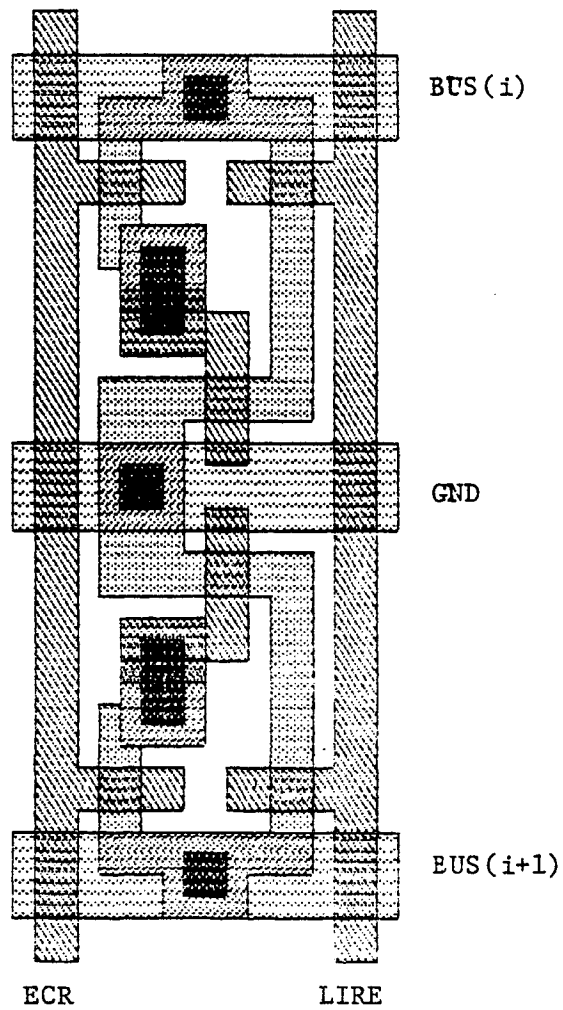


Figure 14: dessin d'une double cellule memoire

La lecture s'opere en activant pendant PHI2, la commande LIRE. Le transistor T3 devient passant. Deux cas sont a considerer. D'une part, si la valeur stockee sur la grille de T2 vaut 1, le bus est alors relie a la masse a travers les transistors T2 et T3. Si la valeur stockee sur la grille de T2 vaut 0, par contre, le bus reste charge. Par consequent, on constate que la valeur lue est le complement de la valeur ecrite.

La cellule decrite precedemment est une cellule memoire dynamique car elle ne peut stocker une information indefiniment, a cause de courants de fuite au niveau du transistor T2 qui dechargent la capacite de la grille de T2. Un rafraichissement des cellules memoires sera donc necessaire periodiquement (quelques millisecondes). Deux raisons principales ont motive le choix de cellules memoires dynamiques: un encombrement reduit sur le silicium et une consommation d'energie pratiquement nulle.

#### Mise en oeuvre

Le dessin a ete fait de telle sorte que deux cellules voisines puissent partager la meme masse. La figure 14 donne le dessin des masques d'une double cellule memoire. Deux cellules sont placees l'une sur l'autre. La seconde cellule etant symetrique a la premiere par rapport a la masse.

#### 5.2.2-Le registre adresse

Le registre adresse est utilise pour memoriser l'adresse a laquelle la prochaine operation memoire doit etre effectuee. Il est constitue de 8 bits dont la signification est la suivante:

- bits 0-1: codage de B, adresse de Bloc;
- bits 2-6: codage de A, Adresse dans le bloc;
- bit 7 : codage de C, type de phoneme.

Le schema d'un bit du registre est donne sur la figure 15. La valeur provenant du bus est tout d'abord inversee (comme le preconise le fonctionnement du bus), puis charge dans un registre dynamique dont la commande de chargement (note CH.PHI1) est synchronisee sur PHI1.

L'utilisation d'un registre dynamique se justifie tres bien puisque le registre adresse memorise l'adresse de la prochaine operation memoire. Par consequent, l'adresse ne doit etre memorisee que pendant quelques cycles d'horloge. Le bit d'information memorise dans le registre est amplifie et fourni, sous ses formes normale et complementee, au decodeur.

#### Mise en oeuvre

Les cellules registres sont empilees les unes sur les autres. Des cellules amplificatrices appelees ADRSPB (amplification sans inversion) et ADRISP (amplification avec inversion) sont egalement placees les unes sur les autres. Cependant, une fois sur deux, les cellules ADRSPB et ADRISP sont placees tete en bas. La figure 16 montre un empilement de 3 cellules de registres avec les amplificateurs associes.

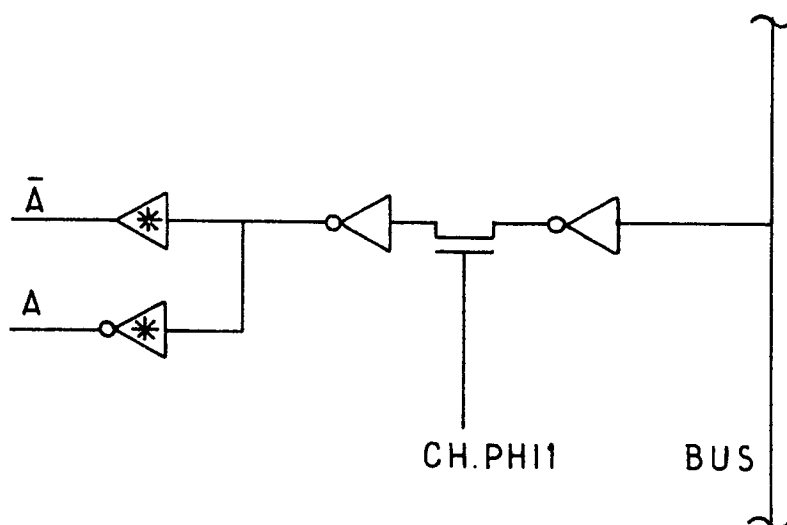


Figure 15: cellule de registre adresse memoire

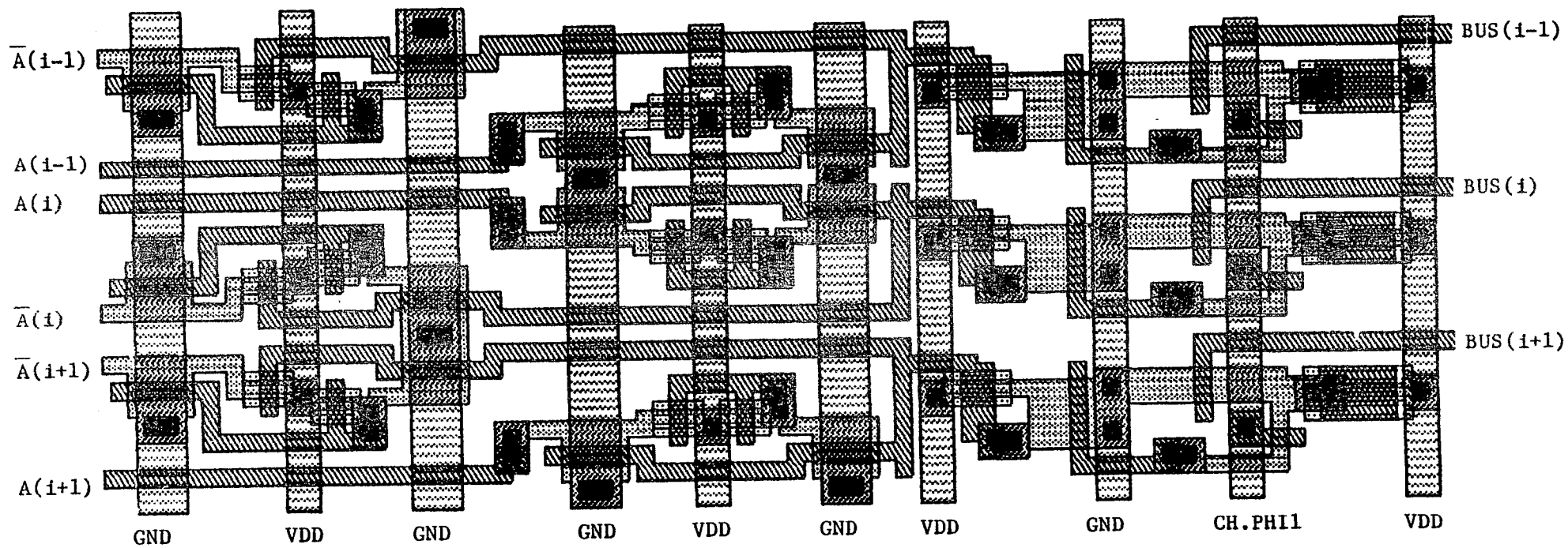


Figure 16: dessin de 3 cellules du registre adresse memoire

### 5.2.3-Le decodeur

Lorsqu'une commande d'accès mémoire a été émise, le rôle du decodeur est d'activer la commande LIRE ou ECR associée aux cellules mémoires du mot adresse. Le decodeur est représenté sur la figure 17. Le decodeur reçoit 2 types de signaux d'entrée:

- des signaux d'adresse;
- des signaux de lecture et d'écriture (LIRE et ECR).

et fournit, en sortie, 2 signaux Lire(i) et Ecr(i) pour chaque mot mémoire i. Chaque signal d'entrée provenant du registre adresse est amplifié sous ses 2 formes: normale et complétement. Chaque signal de sortie est engendré par un circuit NOR à entrées multiples.

### 5.2.4-Le contrôleur

Le rôle du contrôleur est double. Il doit, d'une part, mémoriser les commandes de lecture et d'écriture provenant du decodeur pendant la phase PHI2, puis, d'autre part, émettre des signaux de commande amplifiés pendant PHI1. La figure 18 représente le circuit du contrôleur. Pendant PHI2, la commande LIRE ou ECR provenant du decodeur est mémorisée sur la grille du transistor T1. Pendant PHI1, la valeur de commande est amplifiée et envoyée vers les cellules mémoires.

### 5.2.5-L'interface mémoire

Ce module sert d'interface entre le bus mémoire (noté BM) et le bus général (BG). Lorsqu'une opération d'écriture mémoire est effectuée, l'information à écrire, en provenance de BG est tout d'abord stockée dans un registre appelé REC avant d'être transférée sur BM pour une écriture effective. Lorsqu'une opération de lecture est demandée, un processus similaire se déroule. La donnée provenant de BM est mémorisée dans un registre appelé RLE, après avoir subi certaines transformations, puis est envoyée sur BG. Ces transformations sont de deux ordres. D'une part, du à la structure d'une cellule mémoire à 3 transistors, une valeur lue doit être complétement. D'autre part, l'organisation de la mémoire, décrite plus haut dans le paragraphe 5.2, nécessite un contrôle de validité d'un mot lu. En particulier, dans le cas d'un mot invalide, la valeur  $\emptyset$  doit être rangée dans le registre RLE.

Le circuit représenté sur la figure 19a engendre un signal noté DV (Donnée Valide), qui vaut  $\emptyset$  lorsque le mot lu est valide et réciproquement vaut 1 lorsqu'il est invalide. DV est déterminé à partir de trois signaux d'entrée: TXbar et Cbar décrits précédemment, et un signal R (pour Rafraichissement). La commande R est utilisée pour préciser qu'un contrôle de validité d'une donnée lue en mémoire est nécessaire. En effet, il n'est pas toujours nécessaire d'effectuer de contrôle. La mémoire étant dynamique, elle doit être rafraîchie de temps en temps. Dans ce cas, la valeur lue en mémoire (y compris le bit TX) est réécrite sans aucune modification. La figure 19c donne de façon plus détaillée le circuit qui a été effectivement mis en oeuvre.

La figure 19b donne la table de vérité associée au circuit. Lorsque R vaut 1, cas où aucun contrôle de validité n'est demandé, DV vaut  $\emptyset$  (les données sont valides) quelles que soient les valeurs de

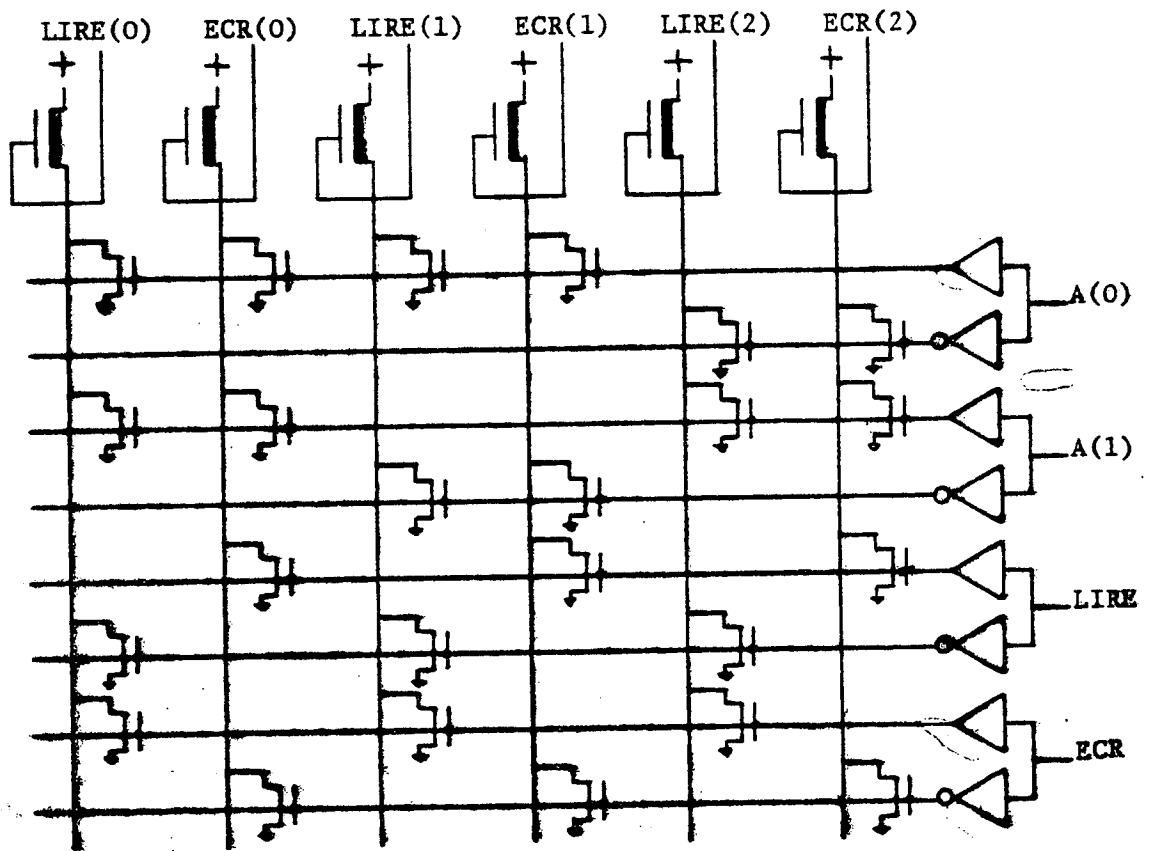


Figure 17: decodeur memoire

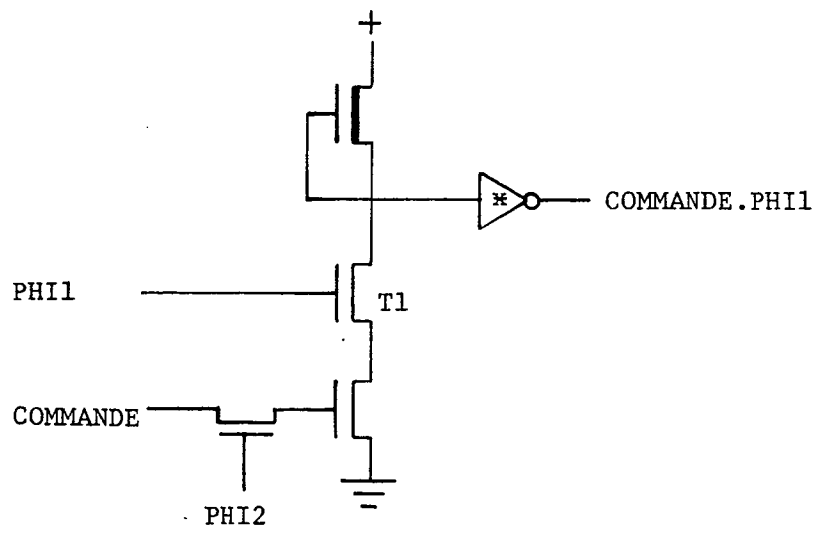


Figure 18: circuit de controle d'operations memoire

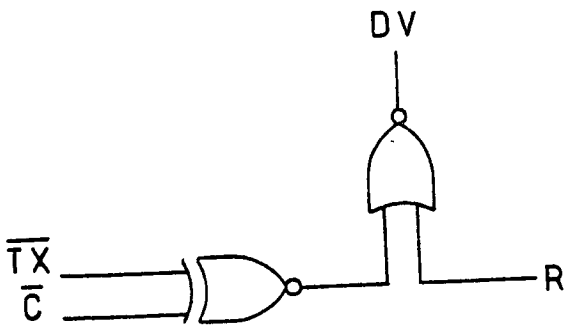


Figure 19a: circuit de controle de validite

$\overline{TX}$	$\overline{C}$	R	DV
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 19b: table de verite du controle de validite

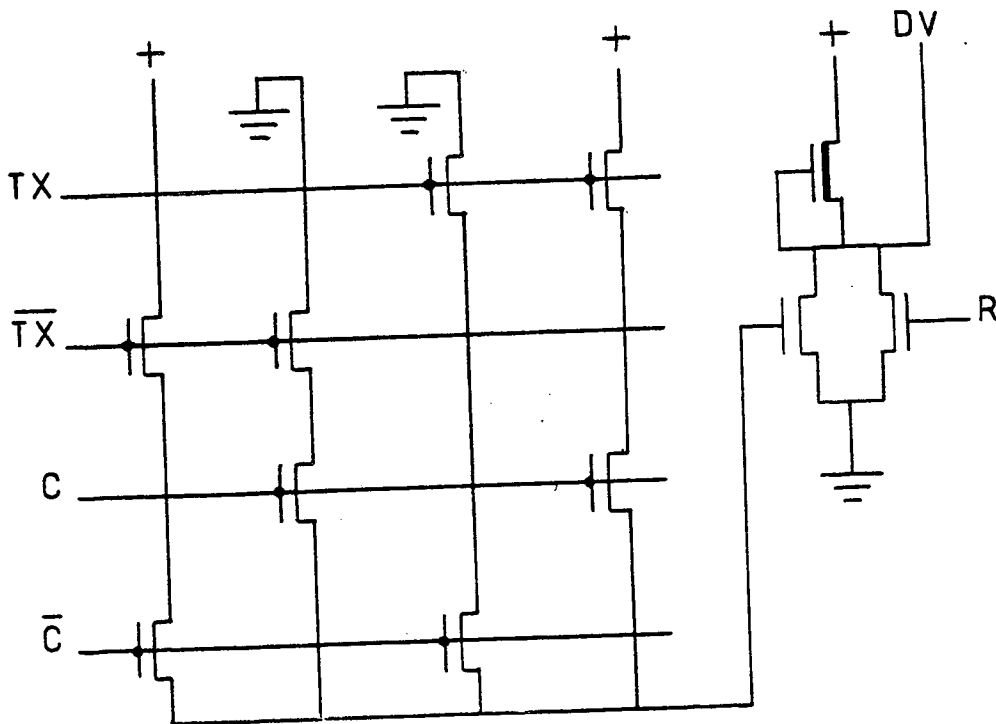


Figure 19c: mise en oeuvre du circuit de controle



TXbar et Cbar. Lorsque R vaut 1, par contre, les donnees ne sont valides ( $DV=0$ ) que si TXbar et Cbar sont egaux.

Le circuit associe aux registres REC et RLE est represente sur la figure 20. Le registre REC est represente sur la figure 20a. La donnee provenant du bus BG est tout d'abord inversee (a cause du principe de fonctionnement du bus), puis memorisee dans un registre dynamique grace a la commande CH.PHI1. La commande ECR.PHI2 permet de tranferer sur le bus BM, la donnee memorisee (qui a subi 2 inversions successives).

Le registre RLE (figure 20b) est similaire a REC. La commande LIRE.PHI2 permet la memorisation de la donnee lue. La donnee enregistree est envoyee sur le bus BG grace a la commande EMET.PHI1.

Sur la figure 20c, est represente le circuit d'interface entre le bus BM et le registre RLE. La table de verite associee a ce circuit (figure 20d) indique d'une part que si la donnee est valide ( $DV=0$ ), la donnee lue en memoire (DL) est complementee (voir fonctionnement d'une cellule memoire), et que, d'autre part, si  $DV=1$  la valeur 0 est fournie en entree du registre RLE.

#### Cas particulier

Le module IM est compose de 12 cellules elementaires decrites precedemment. Cependant la cellule associee au bit TX est legerement differente des autres. En effet son fonctionnement est le suivant:

- lorsqu'un rafraichissement est demande ( $R=1$ ), la valeur lue en memoire est envoyee sur le bus BG (apres complementation);
- lorsque le rafraichissement n'est pas demande, la valeur 1 est envoyee sur le bus BG, quelle que soit la valeur lue.

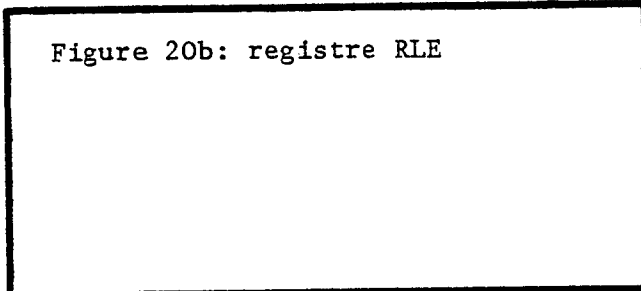
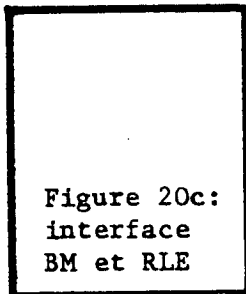
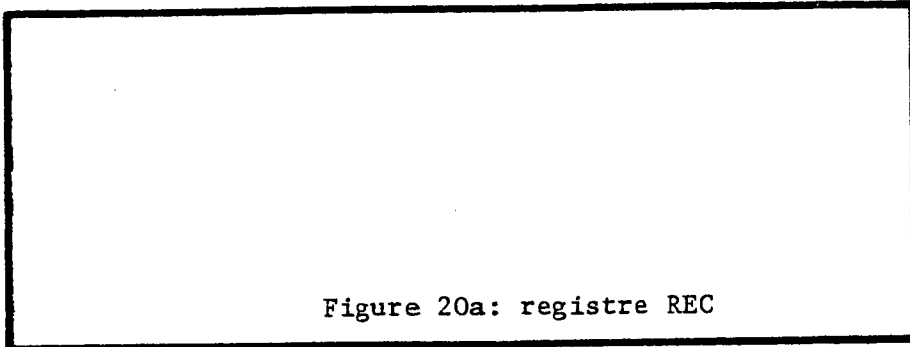
Le circuit correspondant a la cellule TX est represente sur la figure 21. L'interface entre le bus BM et le registre RLE differe de celle decrite figure 20c. La table de verite correspondant a ce circuit est donnee sur la figure 21d: si  $R=1$  (rafraichissement), la donnee lue (DL) est complementee; par contre si  $R=0$ , la valeur 1 est engendree.

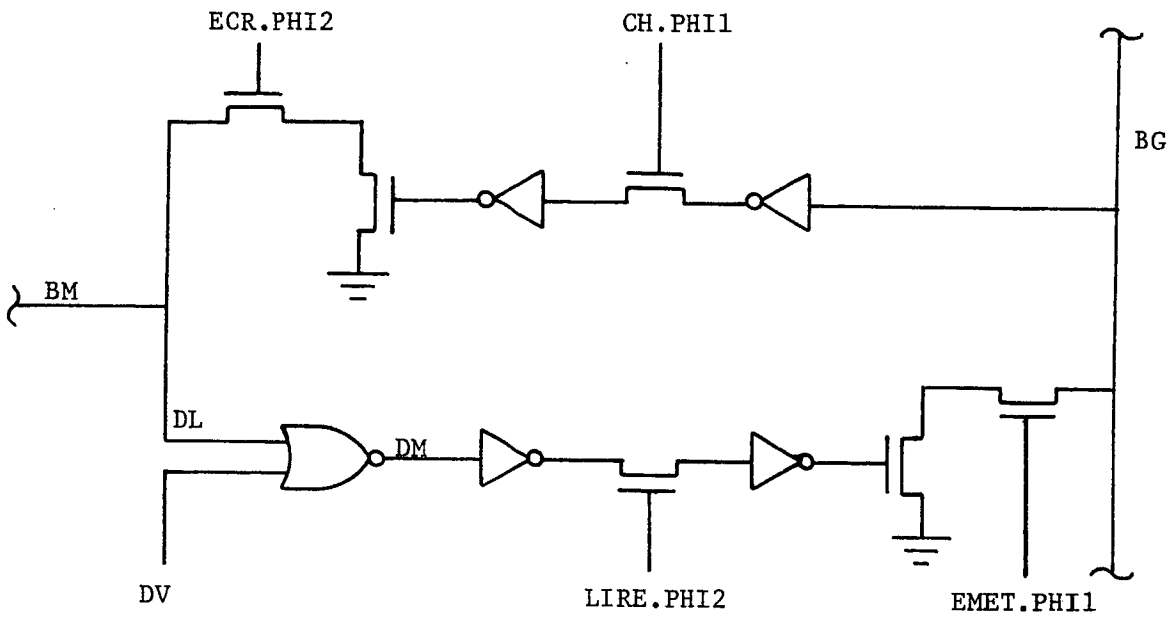
#### Mise en oeuvre

Les cellules de IM sont empilees les une sur les autres permettant ainsi aux commandes associees de se propager verticalement. La figure 22 represente le dessin de l'empilement de 2 cellules dont l'une est la cellule TX.

#### 5.2.5-Remarque

La memoire contenant des mots de 12 bits, 4 bits supplementaires doivent etre fournis pour completer a 16 bits. Les donnees en memoire sont toujours negatives, donc 4 bits 1 sont envoyes sur les poids forts du bus.

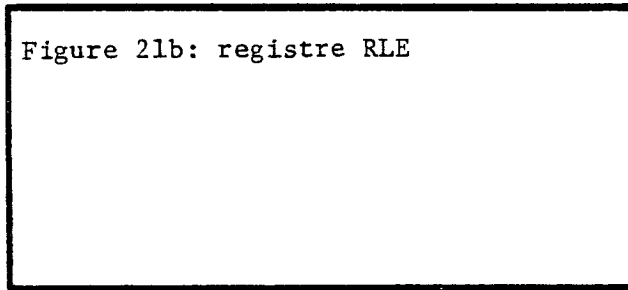
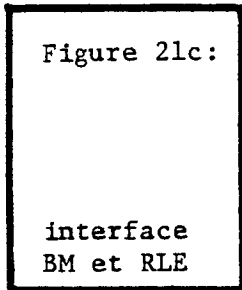
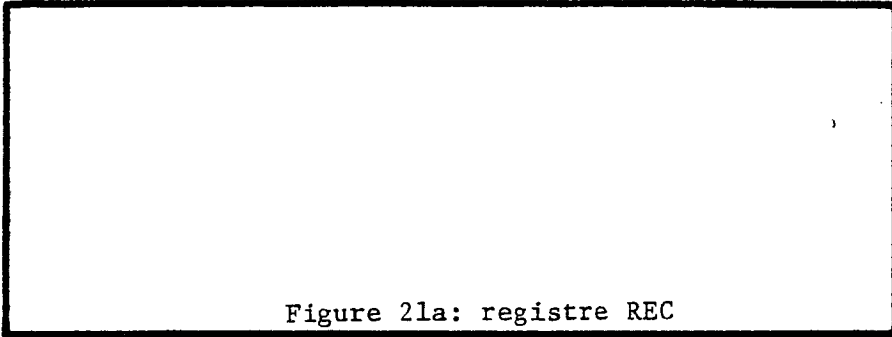


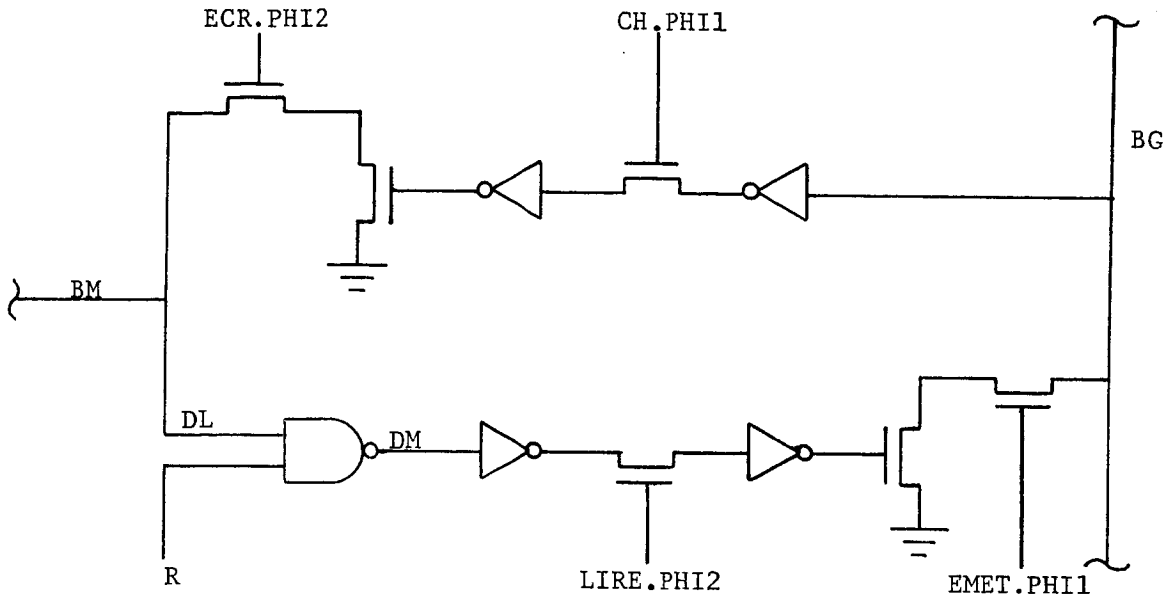


DV	DL	DM
0	0	1
0	1	0
1	0	0
1	1	0

Figure 20d: table de verite associee au circuit  
interfacant le bus BM et le registre RLE

Figure 20: interface memoire





DL	R	DM
0	0	1
0	1	1
1	0	1
1	1	0

Figure 21d: table de verite du circuit interfacant le bus BM et le registre RLE

Figure 21: cellule associee au bit TX

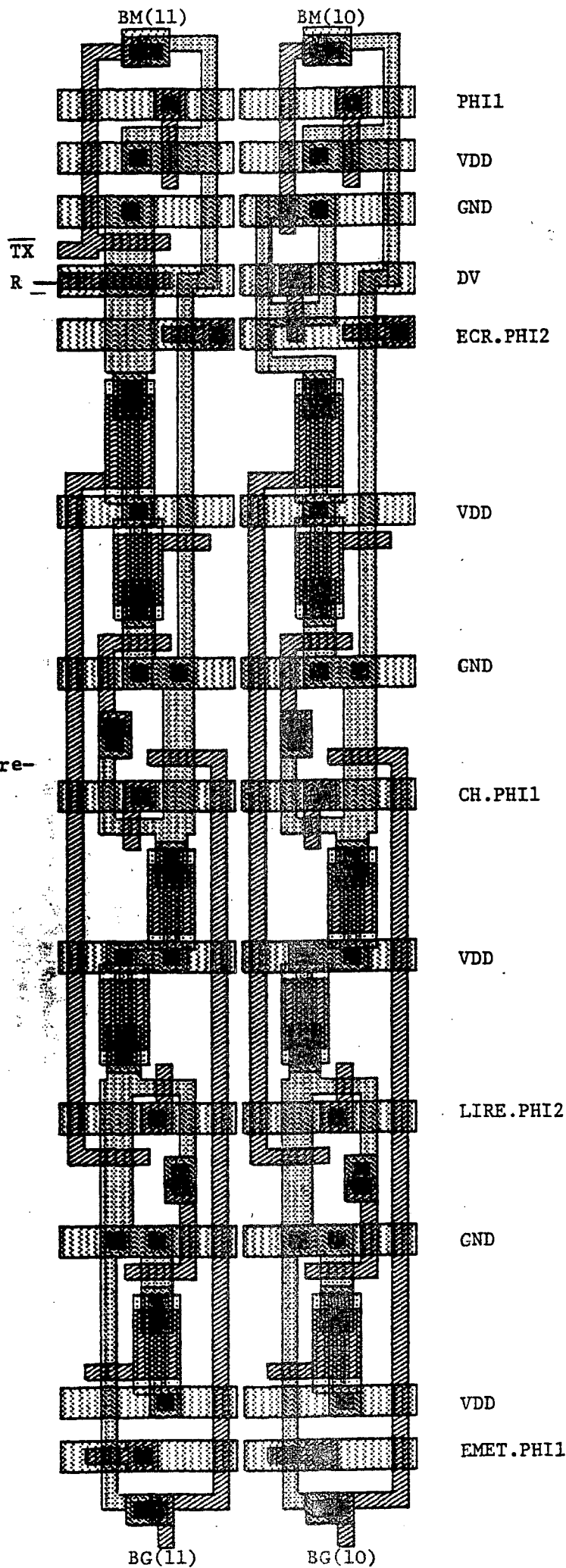


Figure 22: dessin de 2 cellules de IM. Le dessin de gauche represente la cellule TX

### 5.3-Le module Z

Une maniere generale de mettre en oeuvre une fonction logique combinatoire ayant  $n$  entrees et  $m$  sorties est d'utiliser une memoire de  $2^{*n}$  elements de  $m$  bits chacuns. Les  $n$  entrees sont utilisees pour adresser la memoire, et le mot accede fournit les  $m$  valeurs de sorties. Une memoire a acces en lecture seulement, appelee memoire morte, est generalement utilisee. Le probleme majeur associe a cette technique est du a la taille de la memoire ainsi obtenue. En effet cette methode revient a coder, exhaustivement, la table de verite de la fonction  $F$ , sans se soucier de simplifications possibles.

Le module Z sert a tabuler la fonction Z definie dans le paragraphe 4 de la facon suivante:

$$(13) \quad Z:t \rightarrow Z(t)=\text{Log}(1+2^t)$$

Pour mettre en oeuvre cette fonction, la methode la plus simple consiste a tabuler cette fonction a l'aide d'une memoire morte. Cependant les valeurs de  $\text{Log}(u)$  etant codees sur 16 bits, une memoire morte de 64K mots est necessaire. Vue la taille importante de cette memoire, cette solution a ete rejetee. Comme nous allons le montrer plus loin, l'utilisation d'un PLA (Programmable Logic Array) pour tabuler Z est une solution interessante.

Ce paragraphe se compose de 3 parties. Une breve definition d'un PLA est donnee dans une premiere partie. On montre, dans une seconde partie, comment un PLA peut etre utilise pour tabuler la fonction Z. La troisieme partie concerne la mise en oeuvre complete du module Z.

#### 5.3.1-Definition d'un PLA

Notons  $B_p$  l'ensemble des vecteurs de  $p$  composantes booleennes. Soit  $F$  une fonction booleenne definie de  $B_n$  vers  $B_m$ . Soit  $F_i$ , ( $1 \leq i \leq m$ ) la composante de  $F$ , definie de  $B_n$  vers  $B$ , telle que si  $Y$  appartient a  $B_m$ ,  $X$  appartient a  $B_n$  et  $Y=F(X)$  alors  $F_i(X)=Y(i)$ . Chaque fonction  $F_i$ , peut s'exprimer d'une facon simplifiee, sous forme d'une somme de monomes. Soit  $M$  l'ensemble des monomes necessaires a la definition de toutes les fonctions  $F_i$ , la technique utilisee dans un PLA consiste a decoder les seuls monomes appartenant a  $M$ . Le PLA est constitue de 2 parties appelees matrice ET et matrice OU. La matrice ET est formee de  $2^n$  colonnes et  $\text{Card}(M)$  lignes. Les colonnes representent les entrees sous leurs 2 formes: forme normale et forme complementee. Chaque ligne de la matrice ET decode un monome de  $M$ , en ce sens qu'elle effectue le ET logique (d'ou le nom de la matrice) des termes constituant le monome. La matrice OU est formee de  $m$  colonnes et  $\text{Card}(M)$  lignes. Chaque ligne represente un monome de  $M$ . Chaque colonne est associee a une fonction  $F_i$ , puisqu'elle effectue le OU logique des monomes constituant  $F_i$ .

#### 5.3.2-Utilisation d'un PLA pour tabuler Z

L'idee d'utiliser un PLA pour tabuler Z est nee de la remarque suivante. Soit  $u$  une valeur definie dans l'intervalle  $]0,1[$ . Il est clair que  $\text{Log}(u)$  prends ses valeurs dans l'intervalle  $]-\text{inf},0[$  et

$\log(1+u)$  dans l'intervalle  $]0,1]$ . Si l'on considère que les nombres sont codés de la manière décrite dans le paragraphe 4.1.3 (codage sur 16 bits, dont les 9 bits de poids fort représentent la partie entière et les 7 bits de poids faible, la partie fractionnaire), on constate que l'intervalle  $]0,1]$  est codable sur 7 bits et ne contient que 128 valeurs (on notera  $E$  cet ensemble de valeurs). Par contre, l'intervalle  $]-\infty,0]$  nécessite un codage sur au moins 15 bits, et contient  $32K$  valeurs (soit  $S$  cet ensemble).

Soit  $Z'$  la fonction approchée de  $Z$ , définie de  $E$  vers  $S$ . Les remarques précédentes montrent que  $Z'$  est surjective. Par ailleurs, la fonction  $Z'$  étant monotone croissante, on en déduit que les valeurs de l'espace de départ donnant la même valeur dans l'espace d'arrivée, sont consécutives. Soient  $v(i,1), \dots, v(i,k(i))$ ,  $1 \leq k(i) \leq n$ , les valeurs de  $B_n$  qui produisent la valeur  $V(i)$  de  $B_m$  par application de  $Z'$ , la remarque précédente indique que les valeurs  $v(i,1), \dots, v(i,k(i))$  ne diffèrent que par leurs bits de poids faibles. L'utilisation d'un PLA suggère le regroupement des monômes  $M(i,1), \dots, M(i,k(i))$  associés à  $v(i,1), \dots, v(i,k(i))$  en un seul monôme  $M(i)$  tel que:

$$M(i) = \bigvee_{j=1}^{k(i)} M(i,j) \quad (\text{sommation booléenne})$$

Malheureusement, dans la plupart des cas, cette condition n'est pas réalisable. Une solution à ce problème consiste à approcher  $Z'$  par une fonction  $Z''$  qui permet de réaliser la condition précédente.

Soit  $M'(i)$  un monôme qui se décompose de manière canonique, sous forme d'une somme de monômes  $M'(i,j)$  de la façon suivante:

$$M'(i) = \bigvee_{j=1}^{k'(i)} M'(i,j)$$

Posons  $v'(i,j)$ , la valeur associée à  $M'(i,j)$ . La résolution du problème consiste à trouver un ensemble de monômes, noté  $M$ , dont les éléments sont notés  $M'(i)$ , qui satisfait les conditions suivantes:

$$(C1) \quad (\forall i, \forall j, i \neq j) \quad M'(i) + M'(j) \neq M'(i); \\ \text{et } M'(i) + M'(j) \neq M'(j)$$

$$(C2) \quad (\forall V(i), V(i) \text{ appartenant à } S), \\ \text{il existe } v'(i,j) \text{ associé à } M'(i,j) \text{ de } M'(i) \\ \text{tq } Z'(v'(i,j)) = V(i)$$

$$(C3) \quad (\forall v, v \text{ appartenant à } E), \\ \text{il existe } M'(i) \text{ englobant le monôme } Mv \text{ associé à } v$$

$$(C4) \quad \text{il existe une bijection entre } M \text{ et } S$$

La condition (C1) exprime que les monômes de  $M$  sont disjoints, c'est à dire qu'ils couvrent des valeurs différentes. La condition (C2) exprime que toute valeur de l'espace d'arrivée  $S$  de la fonction  $Z'$  est représentée dans un monôme. Par symétrie, la condition (C3) nécessite que tout monôme associé à une valeur de l'espace de départ



soit couvert par un monome  $M'(i)$ . Enfin, la condition (C4) evite qu'un meme monome puisse etre utilise pour deux valeurs de l'ensemble d'arrivee. On peut montrer que plusieurs ensembles  $M$  satisfont les conditions precedentes. La fonction  $Z''$  cherchee, approchee de la fonction  $Z'$ , est la fonction definie de  $E$  vers  $S$ , telle qu'a tout element  $v'(i,j)$  de  $E$  associe a  $M'(i,j)$  on affecte la valeur  $V(i)$  associee a  $M'(i)$ , au sens de la condition (C2).

L'algorithme suivant a ete utilise pour determiner un ensemble  $M$ . Posons  $R$  la fonction reciproque de  $Z$  et  $R'$  la fonction approchee de  $R$  definie de  $S$  vers  $E$ . La fonction  $Z'$  etant surjective, il est clair qu'il existe des elements de  $E$  n'ayant pas d'antecedant. Posons  $E'$ , le sous-ensemble de  $E$ , tel que tout element de  $E'$  possede un antecedant par  $R'$ . Soient  $a(i-1)$ ,  $a(i)$ ,  $a(i+1)$  trois elements consecutifs de  $E'$ . L'algorithme permettant de determiner les monomes  $M'(i)$  est le suivant:

```

pour tout triplet (a(i-1),a(i),a(i+1))
faire
    k:=a(i-1); l:=a(i); m:=a(i+1);
    tant que (k/2 # l/2) et (l/2 # m/2)
    faire
        k:=k/2; l:=l/2; m:=m/2
    fait;
    M(i):= monome associe a l
fait

```

Soit  $m(i)$  le monome associe a  $a(i)$ . Le principe de l'algorithme est de determiner le plus grand monome  $M(i)$  englobant  $m(i)$  verifiant la condition (C1). La methode utilisee consiste a supprimer successivement les bits de poids faible (division par 2) des valeurs  $a(i-1)$ ,  $a(i)$  et  $a(i+1)$ . Les monomes associes aux nouvelles valeurs englobent ainsi les monomes associes aux precedentes valeurs. Le traitement se poursuit tant que les valeurs ainsi obtenues sont differentes. En effet, si deux valeurs deviennent egales, cela signifie que les monomes associes sont egaux et donc que la condition (C1) n'est pas verifiee. On montre que le monome obtenu  $M(i)$  verifie les conditions (C1), (C2) et (C4). La condition (C1) est verifiee par construction meme de l'algorithme.  $M(i)$  verifie la condition (C2), puisque par construction,  $a(i)$  appartient a  $E'$ . Il est clair que la condition (C4) est egalement realisee par construction, puisque pour chaque element de  $E'$ , l'algorithme determine un monome. La condition (C3), par contre, n'a pu etre verifiee que par l'experience.

### 5.3.3-Mise en oeuvre du module Z

Le module Z est forme de 3 parties comme le montre la figure 23: un registre d'entree note RPLA, un PLA et un controleur de sortie.

#### 5.3.3.1-Le registre d'entree

RPLA est un registre de 11 bits. En effet, la table de verite du PLA produite par l'algorithme decrit dans le paragraphe 5.3.2, montre que les bits 10 a 14, n'interviennent que lorsqu'ils valent tous 1. Il est donc interessant de remplacer ces 5 bits par leur ET logique. Par ailleurs, la table de verite montre que le bit 0 n'est jamais utilise. Les 11 bits du RPLA ont la signification suivante:

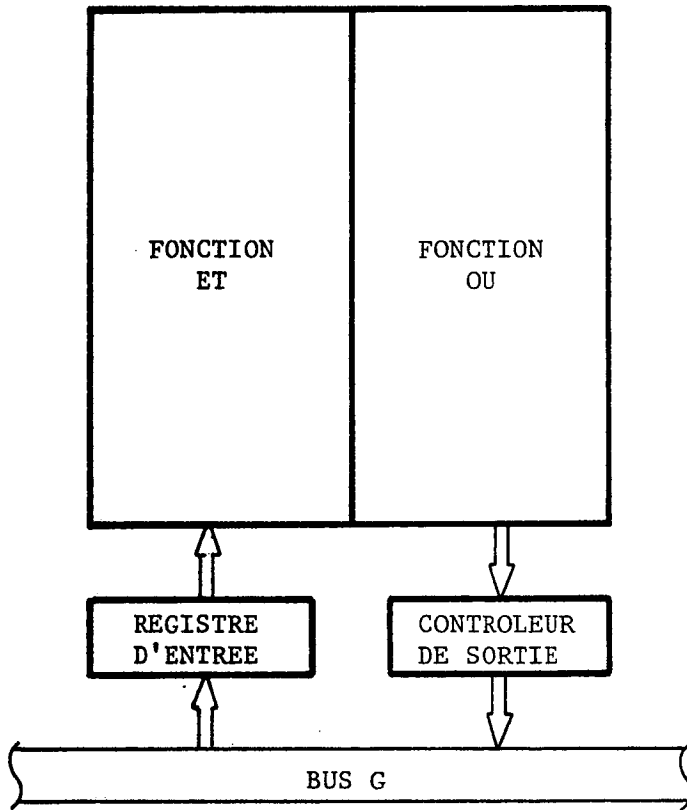


Figure 23: architecture du module Z

- les bits 0 a 8 sont connectes aux bits 1 a 9 du bus;
- le bit 9 est connecte au bit 15 du bus (bit de signe)
- le bit 10 represente le ET logique des bits 10 a 14 du bus.

Pour les bits 0 a 9, le registre d'entree a la structure representee sur la figure 24.

La figure 25 represente le circuit associe au bit 10 du PLA. Le NOR a 5 entrees est utilise pour calculer la fonction suivante:

$$B_{14}.B_{13}.B_{12}.B_{11}.B_{10} = \overline{B_{14}+B_{13}+B_{12}+B_{11}+B_{10}}$$

### 5.3.3.2-Le controleur de sortie

Le PLA fournit des valeurs positives entre 0 et 1 et donc 8 bits sont necessaires pour leur codage: 1 bit avant la virgule et 7 bits apres. Lorsque le resultat du PLA est envoye sur le bus, les 8 bits de sortie sont envoyes sur les poids faible du bus, et 8 valeurs 0 sont envoyees sur les poids forts. La figure 26 montre le circuit associe a une cellule du controleur de sortie.

### 5.3.3.3-Le PLA

Le PLA est forme de 11 entrees, 8 sorties et 128 termes produits. Il a ete engendre automatiquement par un programme appele MKPLA [5]. La mise en oeuvre de ce PLA est decrite en detail dans Mead et Conway [6] et Hon et Sequin [7].

## 5.4-L'unite arithmetique et logique (UAL)

Elle est destinee a effectuer les operations necessitees par les differents programmes. Ces operations sont au nombre de 6 et necessitent 1, 2 ou pas d'operandes. Certaines operations positionnent un indicateur de condition (note COND); d'autres operations s'effectuent conditionnellement a la valeur contenue dans cet indicateur. Si l'on note A et B les operandes et R le resultat de l'operation, les operations de l'UAL sont les suivantes:

- |                                     |       |                                   |
|-------------------------------------|-------|-----------------------------------|
| - addition                          | A + B | --> R                             |
| - soustraction                      | A - B | --> R                             |
| - complementation<br>conditionnelle | * B   | --> R vaut - B si COND<br>B sinon |
| - incrementation                    | B + 1 | --> R                             |
| - soustraction<br>conditionnelle    | A * B | --> R vaut A-B si COND<br>A sinon |
| - zero                              | 0     | --> R                             |

L'unite arithmetique et logique utilisee est celle decrite dans Mead et Conway [6]. Cette UAL est formee principalement de trois modules:

- un module note KILL charge d'arreter la propagation de la retenue;
- un module note PROP qui propage la retenue;
- un module, note RESU, dont le but est de fournir le resultat d'une operation.

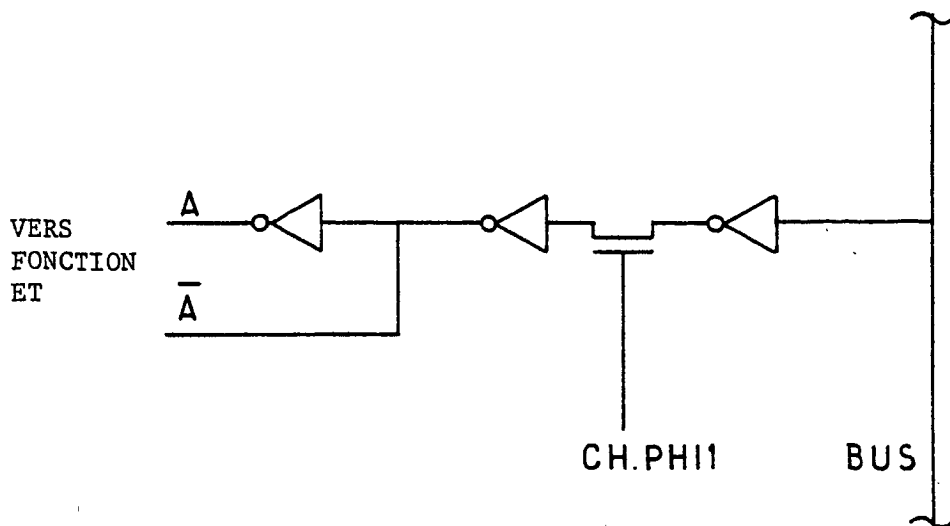


Figure 24: registre entree PLA

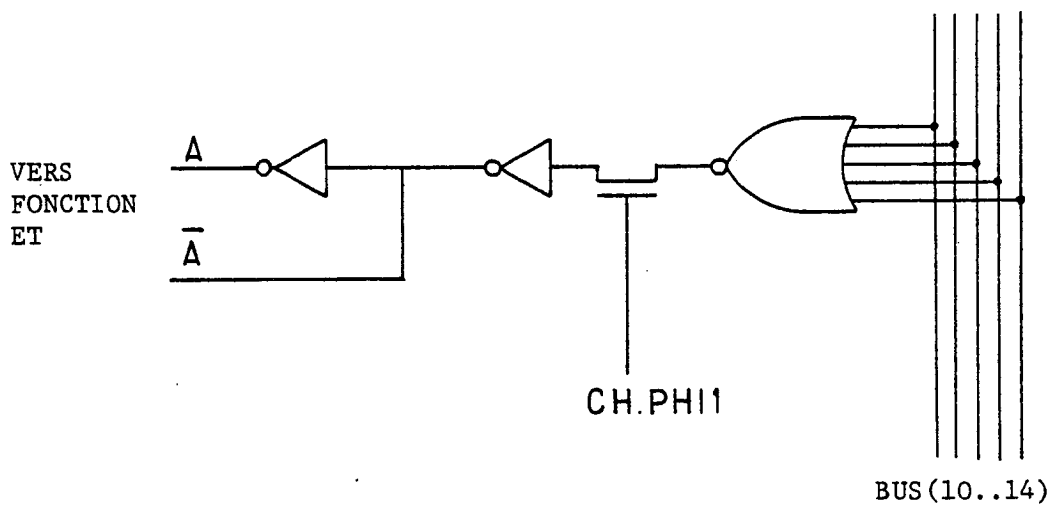


Figure 25: bit 10 du registre entree PLA

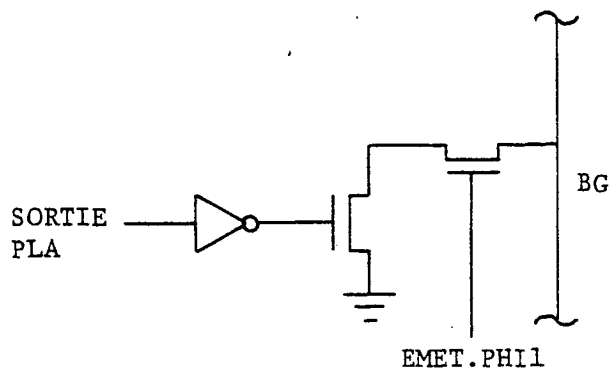


Figure 26: cellule de sortie PLA

Les modules KILL, PROP, RESU sont controles par trois groupes de signaux de 4 bits chacun, notes respectivement K, P, R. Par ailleurs, l'UAL recoit un signal, note Cin, de retenue entrante. Cette UAL est tres generale puisqu'a chaque combinaison de K, P, R et Cin est associee une operation arithmetique ou logique (exemples: addition, soustraction, incrementation, et logique, ou logique, decalage). Le tableau suivant donne les valeurs de K, P, R, Cin pour effectuer les operations decrites precedemment en fonction de la valeur de la condition.

Operation a effectuer	Condition	K	P	R	Cin	Operation effectuee
A+B	-	1	6	6	0	A+B
A-B	-	2	9	6	1	A-B
*B	1	10	5	6	1	-B
*B	0	0	10	12	0	B
B+1	-	5	10	6	1	B+1
A*B	1	2	9	6	1	A-B
A*B	0	0	12	12	0	A
0	-	0	0	0	0	0

Trois registres sont attaches a l'UAL comme le decrit la figure 27a. Les registres RA et RB sont utilises pour memoriser provisoirement les 2 operandes. Le registre RR memorise le resultat de l'operation. Les registres RA et RB sont charges pendant PHI1, puis l'operation est effectuee sur l'UAL et le resultat charge dans RR pendant PHI2. La figure 27b donne les circuits des registres RA, RB et RR. On remarque que le registre RA est charge a partir du bus alors que RB ne peut etre charge qu'avec la valeur de RR.

L'indicateur de condition est memorise dans un registre dynamique de 1 bit. Il est charge avec le bit de poids fort du registre RR. Ce bit correspond, en fait, au signe du resultat de l'operation effectuee. Toutes les operations ne chargent pas l'indicateur COND. Les seules operations qui le positionnent sont les suivantes: A+B, A-B, B+1.

### 5.5-Le tableau de registres

Le programme execute par un processeur (cf programme 2 dans le paragraphe 4.1.2), indique que certaines informations doivent etre memorisees, dans des variables. Ces variables sont de deux types: d'une part, des variables contenant des informations propres a l'algorithme de detection de mots, et d'autre part, des variables de travail qui permettent de memoriser des resultats intermediaires. Les mnemoniques des 16 registres sont les suivants: L, Li, Lc, Lo, Yi, Yc, Yo, X1, X2, X3, I1, I2, I3, NY, Lc\_en\_attente, T1. Toutes ces variables ont ete regroupees dans un tableau de 16 registres de 16 bits.

Le circuit d'une cellule d'un registre est represente sur la figure 28. Le principe de fonctionnement de ce circuit est d'une part d'effectuer des operations memoires (lecture ou ecriture) pendant PHI1, et, d'autre part, de proceder a un rafraichissement de l'information stockee, pendant PHI2. Une operation d'ecriture, consiste a stocker l'information du bus sur l'entree d'un inverseur. Le second inverseur corrige l'information en sortie du premier inverseur et envoie cette

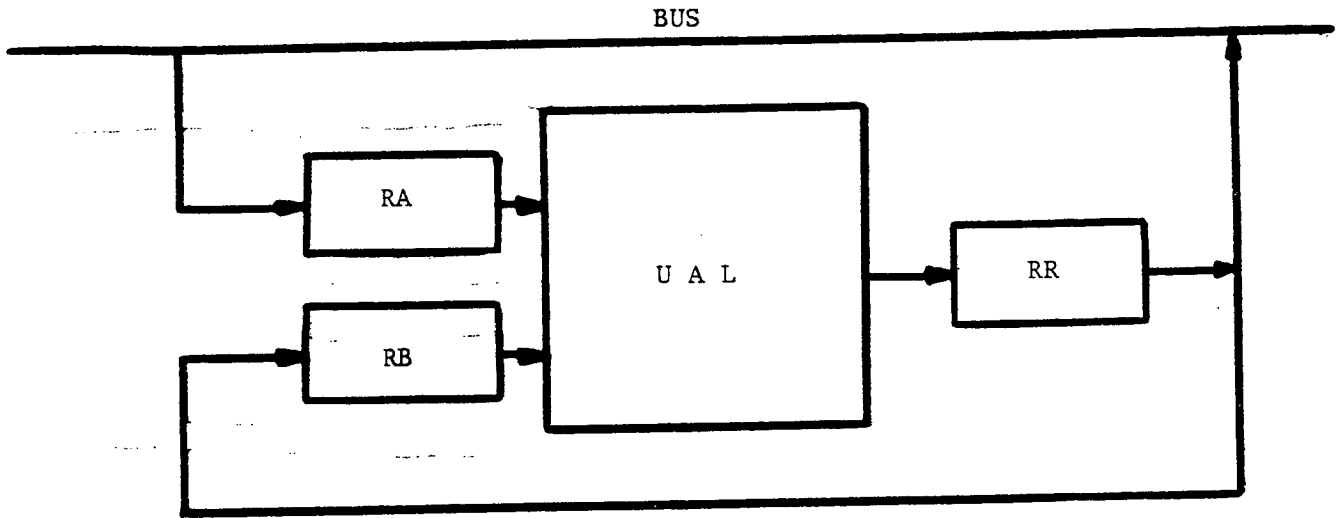


Figure 27a: l'UAL et ses registres

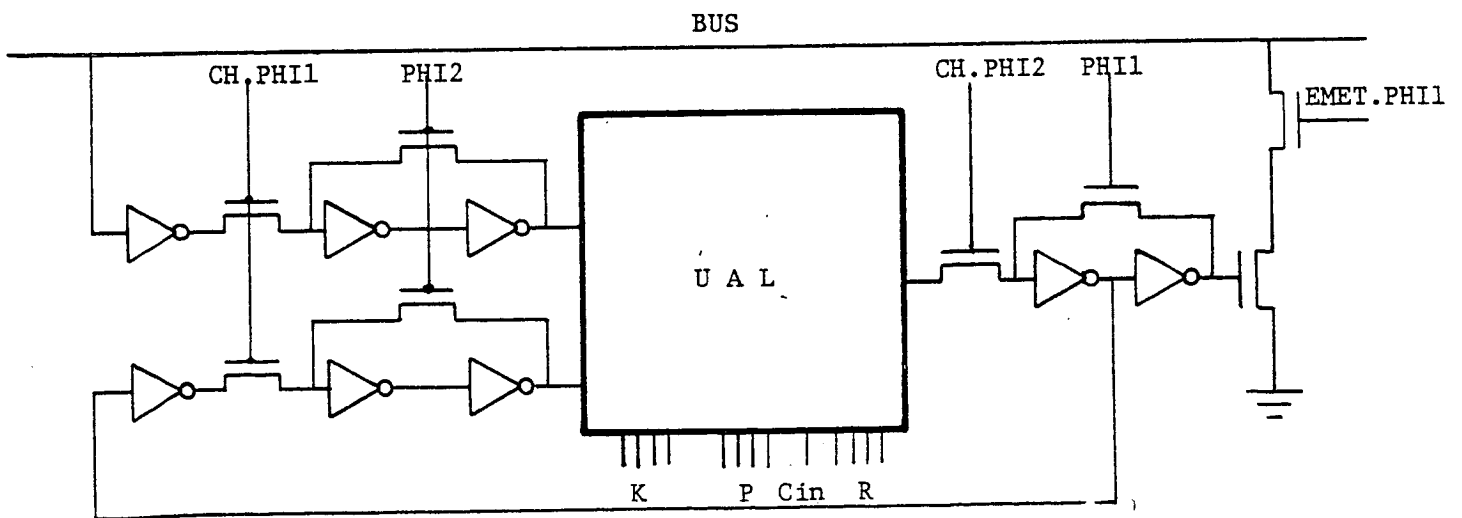


Figure 27b: description des registres de l'UAL

information sur le bus lorsqu'une lecture est effectuee. L'entree du premier inverseur et la sortie du second sont reliees a travers un transistor actif pendant PHI2, permettant ainsi de rafraichir l'information stockee.

On note que les registres ne sont pas connectes au bus BG suivant le principe decrit dans le paragraphe 5.1. En effet, l'information provenant du bus est utilisee directement, sans avoir ete complementee. Reciproquement, l'information stockee dans un registre est envoyee directement sur BG. Chaque cellule registre etant formee principalement de 2 inverseurs, adopter le principe decrit precedemment aurait necessite l'introduction de 2 inverseurs supplementaires, ce qui, pour des raisons d'encombrement, n'etait pas tres realiste. Ainsi les cellules registres recevant directement la valeur du bus, contiennent une information complementee. Lors d'une lecture, la valeur du registre est envoyee directement sur le bus. Par consequent, le bus transfere une valeur complementee comme il se doit.

Les commandes de lecture et d'ecriture, synchronisees avec PHI1, sont engendrees par un circuit de meme type que celui represente sur la figure 18. Pendant PHI2, la commande d'operation memoire est stockee sur la grille du transistor T1; le transistor T2 etant "ouvert", la valeur 0 est fournie en sortie du circuit. Lorsque PHI1 devient actif, l'information precedemment stockee sur T1 est alors disponible sur la sortie.

#### Mise en oeuvre

Le dessin d'une cellule de registre est represente sur la figure 29. Les cellules d'un meme registre, sont placees les unes sur les autres. Cependant une cellule sur deux est placee apres avoir subi une operation de symetrie sur y. Ceci permet le partage de VDD et GND entre deux cellules voisines.

### 5.6-Controle des operations internes au processeur

Dans les paragraphes precedents, les differents modules constituant la partie operative du processeur ont ete etudies. A chaque module est associe un certain nombre de commandes necessaires a son fonctionnement. Ces commandes pourraient provenir directement du controleur CG. Cette solution, qui a l'avantage de permettre l'execution de plusieurs commandes en parallele, n'a pas ete retenue a cause du trop grand nombre de commandes qui sont a fournir au chip. La solution adoptee a ete de grouper les commandes et de les coder dans une microinstruction.

Par consequent, cette microinstruction doit etre decodee a l'interieur du processeur. Ce paragraphe est compose de deux parties: le codage des differentes commandes est expose dans un premier paragraphe, les modules effectuant le decodage sont ensuite etudies dans un second paragraphe.

#### 5.6.1-Codage des commandes

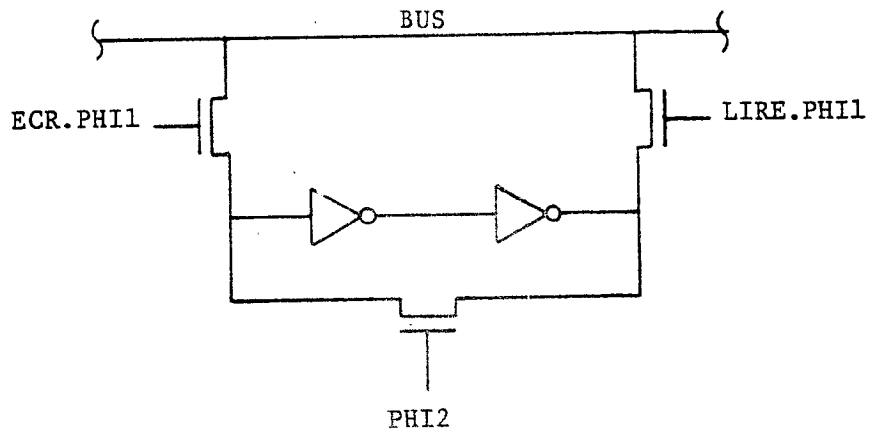


Figure 28: une cellule d'un registre general

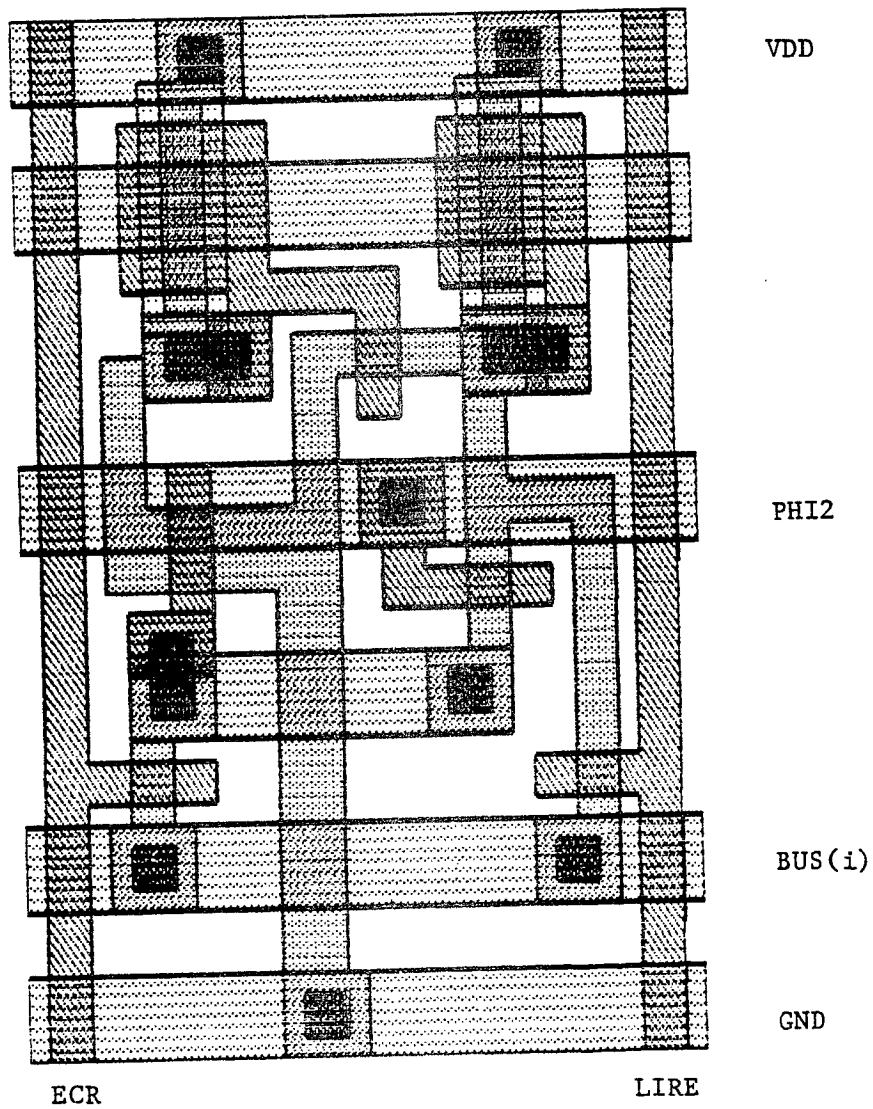


Figure 29: dessin d'une cellule registre general



Les commandes ont été divisées en deux groupes suivant qu'elles sont actives pendant PH11 ou PH12. La taille d'une microinstruction est ainsi réduite à 10 bits, notes M0 à M9.

5.6.1.1-Microinstruction active pendant PH11

Pendant PH11, des transferts sur le bus général (BG) sont organisés. La microinstruction exécutable pendant cette phase se compose de trois champs de 3, 3 et 4 bits respectivement comme l'indique la figure 30.

ORIGINE			DESTINATION			REGISTRE			
M9	M8	M7	M6	M5	M4	M3	M2	M1	M0

Figure 30 : Microinstruction active pendant PH11

Le champ ORIGINE précise d'où provient l'information à transférer sur BG. Le champ DESTINATION précise à quel endroit l'information doit parvenir. Dans le cas où l'un des registres généraux intervient dans un transfert, l'adresse de ce registre est codée dans le champ REGISTRE. À noter que cette méthode de codage interdit le transfert d'information entre registres généraux puisque l'adresse d'un seul registre peut être codée. La figure 31 donne le codage des différentes commandes.

Champ ORIGINE	codage
- registre général	M9M8M7
- registre sortie UAL (Acc)	0 0 1
- sortie PLA	0 1 0
- registre mémoire (RM)	0 1 1
- registre entrée horizontale RH	1 0 1
- registre entrée verticale RV	1 1 0
- rien	1 1 1
	0 0 0
Champ DESTINATION	M6M5M4
- registre général	0 0 1
- registre entrée UAL (RA)	0 1 0
- registre entrée PLA	0 1 1
- registre mémoire (RM)	1 0 1
- registre sortie chip RS	1 1 0
- registre adresse mémoire (RAM)	1 0 0
- rien	0 0 0
Champ REGISTRE	
- nombre entre 0 et 15	

Figure 31: codage des champs de la microinstruction active pendant phil

5.6.1.2-Microinstruction active pendant PH12

Les opérations commandées par cette microinstruction sont de trois types: opérations sur la mémoire, opérations sur l'UAL, opérations d'entrées/sorties. La microinstruction est formée de trois champs de 2, 1, et 4 bits respectivement, comme l'indique la figure 32.

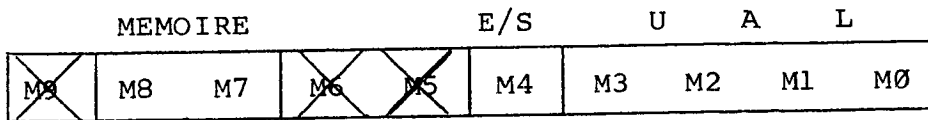


Figure 32 : microinstruction active pendant PHI2

On remarque que trois bits sont inutilisés. La figure 33 donne le codage des différents champs de la microinstruction.

	codage
Champ MEMOIRE	M8 M7
- pas d'operation	0 0
- ecriture	0 1
- lecture	1 0
- lecture pour rafraichissement	1 1
Champ entree/sortie (E/S)	M4
- pas d'operation	0
- chargement registres RV et RH	1
Champ UAL	M3 M2 M1 M0
- pas d'operation	0 - - -
- 0 --> Acc	1 0 0 0
- RA + Acc --> Acc	1 0 0 1
- RA - Acc --> Acc	1 0 1 0
- * Acc --> Acc	1 0 1 1
- Acc + 1 --> Acc	1 1 0 0
- RA * Acc --> Acc	1 1 0 1

Figure 33: codage de la microinstruction active pendant phi2

### 5.6.2-Decodage de la microinstruction

Rappelons qu'une microinstruction est composee de 10 bits, et que le chip recoit une nouvelle microinstruction a chaque phase de l'horloge. Suivant que la microinstruction s'exécute pendant PHI1 ou PHI2, les champs de bits qu'elle comporte n'ont pas la meme signification. Le role du decodeur de microinstruction est d'engendrer les commandes pour les differents modules en fonction d'une part de la phase courante de l'horloge et d'autre part, des valeurs des champs de la microinstruction.

Le decodage est effectue principalement dans 3 modules:

- Le decodeur de l'UAL
- Le decodeur d'adresse de registre
- Le decodeur des champs M4-M9

#### 5.6.2.1-Decodage des operations sur l'UAL

Pour commander K, P, R et Cin en fonction de l'operation demandee, un PLA est utilise. De plus, afin de regrouper toutes les commandes concernant l'UAL dans un meme module, ce PLA va aussi engendrer deux autres commandes: la commande de chargement de l'accumulateur, active lors d'une operation sur l'UAL, et la commande de chargement du signe du resultat qui ne s'opere que sous certaines conditions. Le PLA

reçoit 5 bits en entree: les bits M0 a M3 et la valeur du registre SIGNE. En combinant ces 5 informations, il genere les signaux K, P, R, Cin, CHbar/Sgn et CHbar/Acc. La figure 34 montre la table de verite du PLA.

M3M2M1M0	SGN	K3K2K1K0	P3P2P1P0	Cin	R3R2R1R0	CHbar /sgn	CHbar /acc
0 - - -	-	0 0 0 0	0 0 0 0	0	0 0 0 0	1	1
1 0 0 0	-	0 0 0 0	0 0 0 0	0	0 0 0 0	0	0
1 0 0 1	-	0 0 0 1	0 1 1 0	0	0 1 1 0	0	0
1 0 1 0	-	0 0 1 0	1 0 0 1	1	0 1 1 0	0	0
1 0 1 1	0	0 0 0 0	1 0 1 0	0	1 1 0 0	1	0
1 0 1 1	1	1 0 1 0	0 1 0 1	1	0 1 1 0	1	0
1 1 0 0	-	0 1 0 1	1 0 1 0	1	0 1 1 0	0	0
1 1 0 1	0	0 0 0 0	1 1 0 0	0	1 1 0 0	1	0
1 1 0 1	1	0 0 1 0	1 0 0 1	1	0 1 1 0	1	0

Figure 34: Table de verite du pla associe a l'UAL

#### Mise en oeuvre

Pour la realisation effective du PLA, quelques simplifications ont ete operees. On remarque en effet que R0 vaut toujours 0 et que K1 est egal a P0. Ceci permet une legere diminution du nombre de sorties du PLA. Par ailleurs la condition d'entree 1000- ne donnant que des "0" en sortie, il est inutile de rentrer cette valeur effectivement dans le PLA, puisqu'un PLA fournit la valeur 0 en sortie, lorsqu'aucune condition d'entree n'est verifiee. Le PLA comporte finalement: 5 entrees, 13 sorties et 8 termes produits.

#### 5.6.2.2-Decodage du champ registre

Le champ de microinstruction M0 a M3 permet le codage d'une valeur entre 0 et 15, et ainsi sert a selectionner l'un des 16 registres. Le decodeur d'operations sur un registre recoit les informations suivantes:

- 4 bits d'adresse permettant de selectionner l'un des 16 registres
- 1 bit de lecture
- 1 bit d'ecriture

La structure du decodeur est la meme que celle du decodeur memoire (cf figure 17).

#### 5.6.2.3-Decodage des champs M4-M9

Le decodage du champ M4-M9 se fait dans un module appele DCDM49. Ce module est charge de generer, memoriser et envoyer des signaux de commandes aux differents modules. Le module DCDM49 est compose de deux parties comme le montre la figure 35. Une partie decodage (notee DCD49) qui reconnait certaines configurations binaires et une partie generation de signaux de commandes.

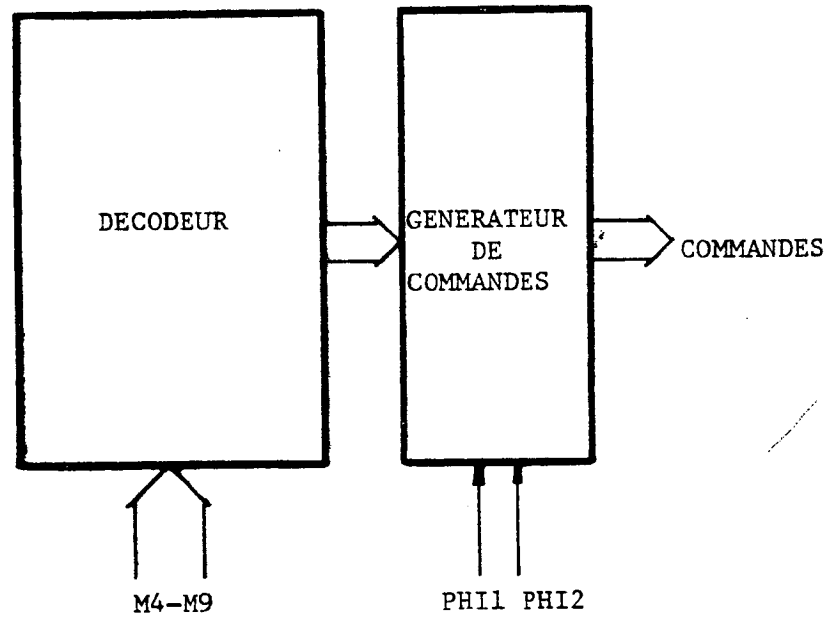


Figure 35: architecture du module DCDM49

Le decodeur est programme comme l'indique la figure 36.

M9	M8	M7	M6	M5	M4	SIGNAL ENGENDRE
0	0	1	-	-	-	LIRE REGISTRE
0	1	0	-	-	-	Acc/bus.PHI1
0	1	1	-	-	-	PLA/bus.PHI1
1	0	1	-	-	-	RM/bus.PHI1
1	1	0	-	-	-	RH/bus.PHI1
1	1	1	-	-	-	RV/bus.PHI1
-	-	-	0	0	1	ECR REGISTRE
-	-	-	0	1	0	CH/Acc.PHI1
-	-	-	0	1	1	CH/RPLA.PHI1
-	-	-	1	0	0	CH/RAM.PHI1
-	-	-	1	0	1	CH/RM.PHI1
-	-	-	1	1	0	CH/RS.PHI1
-	0	1	-	-	-	ECR MEMOIRE
-	0	1	-	-	-	ECR/MEM.PHI2
-	1	-	-	-	-	LIRE MEMOIRE
-	1	-	-	-	-	LIR/MEM.PHI2
-	1	1	-	-	-	REF/MEM.PHI2
-	-	-	-	-	1	CH/RH+RV.PHI2

Figure 36: Programmation du decodeur M49

Le generateur de signaux de commandes fournit des signaux de trois types:

- des signaux de commandes executables pendant PHI1. Ces signaux sont recus pendant la phase PHI2 et memorises pendant cette phase, puis envoyes pendant la phase PHI1 suivante (type 1).
- des signaux de commandes executables pendant PHI2. Ces signaux sont recus pendant la phase PHI1 et memorises pendant cette phase, puis envoyes pendant la phase PHI2 suivante (type 2).
- des signaux destines aux autres decodeurs. Ces signaux sont decodes et sont envoyes pendant le meme cycle (pas de memorisation) vers les decodeurs memoire et registre (type 3). Les signaux de ce troisieme type sont essentiellement les 4 signaux suivants:
  - lecture memoire
  - ecriture memoire
  - lecture registre
  - ecriture registre.

#### Mise en oeuvre

Les trois types de generateurs ont ete dessines de telle sorte qu'ils puissent s'empiler les uns sur les autres, partageant ainsi les signaux PHI1, PHI2, VDD, GND. La figure 37 represente les dessins des trois circuits et leur empilement.

#### 5.7-Mise en place des modules sur la plaquette de silicium

L'une des phases les plus delicates dans la conception d'un microprocesseur consiste dans le placement, sur la plaquette de silicium, des modules qui le constituent. Cette phase est cruciale pour deux raisons principales. D'une part, elle determine la taille de la puce, facteur primordial actuellement pour la production de circuits

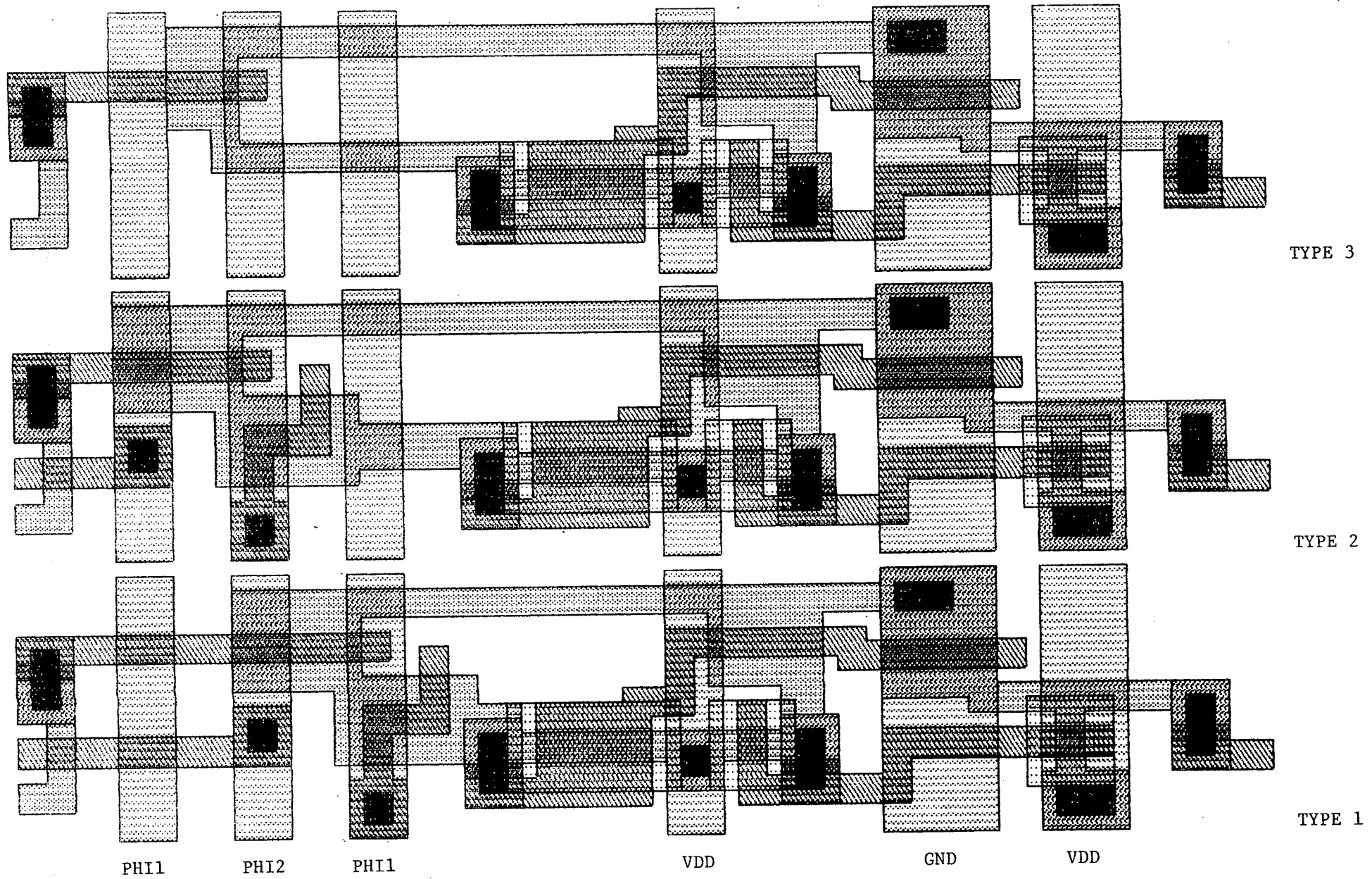


Figure 37: dessin des 3 types de generateurs de commandes

a grande echelle. En effet, plus le chip est grand, plus le rendement est faible et donc plus le circuit est cher. D'autre part, elle peut influencer les performances du processeur. En effet, une mauvaise repartition des modules entraine generalement l'utilisation de longs fils, d'un module a l'autre, d'ou une diminution des performances. Dans la technologie visee (NMOS 5 microns), la diminution de performances est peu sensible. Cependant, il faut conserver a l'esprit qu'avec les progres constants realises au niveau de l'integration, le temps de propagation des signaux sur des longs fils deviendra un facteur determinant dans les prochaines annees.

La figure 38 represente l'implantation des differents modules sur le silicium. Le chip comporte 62 plots de connexions, situes sur la peripherie de la puce, qui se repartissent de la maniere suivante:

- 3 series de 16 plots pour les entrees/sorties;
- 10 plots pour le controle des operations internes;
- 4 plots pour l'alimentation et les horloges.

Une version du chip avec seulement 18 plots de connexions etait envisageable en supposant que les entrees/sorties s'effectuent en serie. Par consequent, un plot par port d'entree/sortie etait necessaire. Plusieurs raisons ont motive notre choix pour la version parallele, notamment, la rapidite des transferts, la facilite de programmation, de controle et de test du processeur, et egalement la simplicite de realisation. Il est clair, cependant, que la version serie doit etre envisagee dans le cas de l'integration de plusieurs processeurs sur la meme puce.

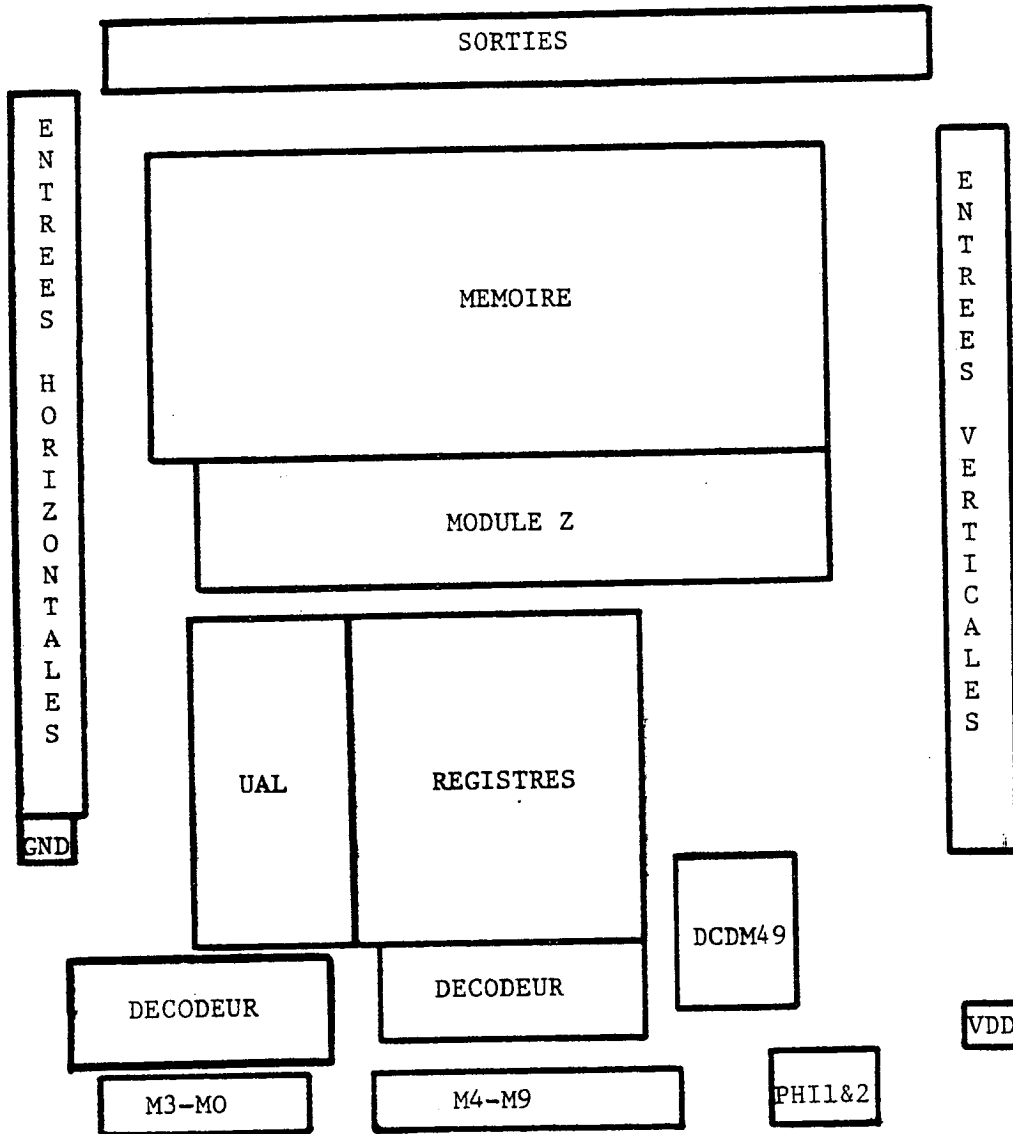
Les ports d'entrees horizontales et verticales du processeur se situent respectivement sur les cotes gauche et droit. Le port de sortie se trouve dans le haut. Le port de controle, recevant les microinstructions, ainsi que les plots des horloges PHI1 et PHI2, se situent dans le bas. La masse et l'alimentation occupent respectivement les plots inferieurs des cotes gauche et droit.

Le bus G est distribue sur 3 cotes du chip, le long des plots d'entrees/sorties du processeur. Ceci permet, d'une part, de connecter aisement les registres d'entrees/sorties (integres directement pres des plots correspondants) sur le bus et, d'autre part, de faciliter les connexions des differents modules sur le bus. Le bus G est constitue de 16 fils metalliques afin de minimiser les delais de transmissions d'informations.

Egalement dans le but de simplifier le probleme de cablage, les commandes destinees a la memoire, au module Z et aux registres d'entrees/sorties, sont regroupees sous forme d'un bus de 16 fils metalliques. Ce bus de controle s'etend sur le cote droit du chip, entre les bus G et les plots de connexions.

La memoire et le module Z occupent la partie superieure de la plaquette et sont connectes au bus G sur le cote droit. L'UAL et le tableau de registres sont places l'un a cote de l'autre, dans la partie inferieure du chip. Ils sont connectes au bus sur le cote gauche.

Les trois modules de decodage de microinstruction sont places dans le bas du circuit. Le decodeur d'operations arithmetiques est place





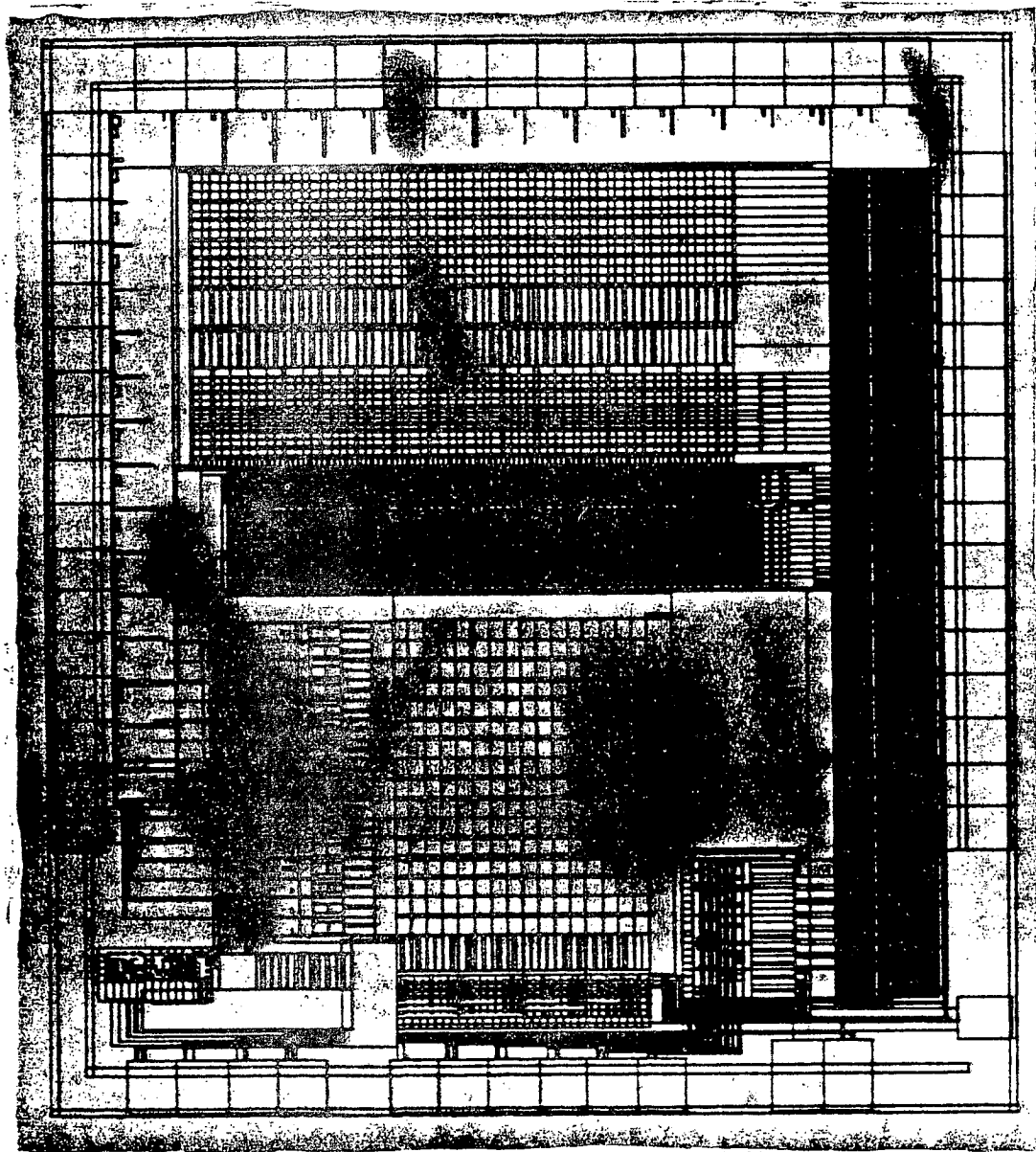


Figure 38: plan de la puce de silicium

sous l'UAL, le decodeur de registre, sous le tableau de registre. On note que les plots associes aux bits M0-M3 de la microinstruction sont proches de ces deux decodeurs, reduisant ainsi la longueur de fils de connexions. Enfin, le decodeur du champ M4-M9 est place en bas, a droite, a cote du bus de commande.

## 6-Programmation du reseau de processeur

Dans cette partie, nous nous interessons a la programmation du tableau de processeurs ainsi qu'a ces performances. Avant le lancement du traitement en pipeline sur les mots, le multiprocesseur doit etre initialise. Cette initialisation comprend deux parties: d'une part le chargement des memoires des processeurs et d'autre part, le chargement de certains registres. Par ailleurs, la memoire etant dynamique, il est necessaire de prevoir un programme de rafraichissement qui s'executera periodiquement, toutes les 2 a 3 millisecondes, par exemple.

Dans un premier paragraphe, l'initialisation des memoires des processeurs est aborde. Puis le chargement des registres est etudie dans un second paragraphe. Dans le troisieme paragraphe, nous decrivons le programme de base d'un processeur. Enfin un programme de rafraichissement de la memoire des processeurs est donne dans le quatrieme paragraphe.

Dans la suite du texte, les programmes seront ecrits dans un pseudo langage de programmation proche de Pascal. En ce qui concerne les instructions d'entrees/sorties, nous utiliserons les 2 instructions suivantes: Afficher et Lire. L'instruction Afficher(X) signifie que la valeur X est envoyee sur le port de sortie. L'instruction Lire(E,X) est utilisee pour specifier que la valeur se trouvant sur le port d'entree E (Horizontal ou Vertical) doit etre rangee dans la variable X. Pour la plupart des programmes, les microprogrammes sont donnes en detail. Pour chaque phase d'horloge, nous donnons les commandes a activer.

### 6.1-Initialisation des memoires des processeurs

Nous avons vu que dans le cas du mode pipeline sur les mots, les memoires des processeurs doivent etre prealablement chargees avec les valeurs de probabilite de confusion  $qc(x/y)$ , pour 3 valeurs de x et 20 valeurs de y (cf structure de la memoire dans le paragraphe 5.2.). Ces valeurs sont fournies sur le port d'entree verticale des processeurs de la premiere rangee. Les processeurs d'une meme colonne possedent les memes valeurs en memoire.

#### 6.1.1-Chargement d'une colonne

Le principe du chargement des memoires des processeurs d'une meme colonne consiste a transferer les valeurs de processeurs en processeurs en mode pipeline. Pour ce faire, les valeurs a enregistrer sont precedees de leur adresse. Le fonctionnement est le suivant:

- a l'etape 0, seul le processeur  $\langle 0,0 \rangle$  est actif. Il recoit successivement, sur son port d'entree verticale, les valeurs 0 et  $V(0)$  puis range  $V(0)$  a l'adresse 0.
- a l'etape 1, le processeur  $\langle 0,0 \rangle$  recoit la valeur 1 et emet la valeur 0 vers le processeur  $\langle 1,0 \rangle$ . Puis il recoit la valeur  $V(1)$  et transmet a  $\langle 1,0 \rangle$  la valeur  $V(0)$ . Le processeur  $\langle 0,0 \rangle$  range la valeur  $V(1)$  a l'adresse 1 tandis que  $\langle 1,0 \rangle$  range  $V(0)$  a l'adresse 0.
- plus generalement, a l'etape t, le processeur  $\langle j,0 \rangle$  recoit les valeurs t-j et  $V(t-j)$  et emet les valeurs t-j-1 et  $V(t-j-1)$  vers le processeur  $\langle j+1,0 \rangle$ .

### 6.1.2-Chargement de l'ensemble des processeurs

Les processeurs d'une meme diagonale etant commandes par un seul controleur, le chargement de la seconde colonne commence avec un cycle de retard par rapport a celui de la premiere colonne. Le chargement est represente sur la figure 39 et s'interprete ainsi:

- a l'etape 0, seul le processeur  $\langle 0,0 \rangle$  est actif. Le chargement des processeurs de la premiere colonne est lance.
- a l'etape 1, les processeurs  $\langle 0,0 \rangle$ ,  $\langle 1,0 \rangle$  et  $\langle 0,1 \rangle$  sont actifs. Le chargement des processeurs de la seconde colonne est lance.
- plus generalement, a l'etape t, les processeurs  $\langle i,j \rangle$  tels que  $(i+j) \leq t$  sont actifs. Le chargement des processeurs des colonnes 1 a t est en cours.

### 6.1.3-Programme de chargement

Le programme execute, a chaque etape, par un processeur actif est le suivant:

```

Debut
  Afficher (adresse);
  Lire (Vertical, adresse);
  Afficher (valeur);
  Lire (Vertical, valeur);
  memoire [adresse] := valeur

```

Fin

Programme 3: chargement des memoires

Ce programme s'explique ainsi. Afin que les processeurs se transmettent des valeurs, ils commencent par afficher la valeur a transmettre sur leur port de sortie, puis lisent la nouvelle valeur sur leur port d'entree verticale. Ce procede est repete pour la lecture de l'adresse et de la valeur. Puis le rangement memoire est effectue.

### 6.1.4-Temps de chargement des memoires

Le microprogramme correspondant au programme 3 precedent est le suivant:

```

1: REG(AD)/BUS ; CH/RS
2: CH/RV
1: RV/BUS ; CH/REG(AD)
2: -
1: REG(VAL)/BUS ; CH/RS
2: CH/RV
1: RV/BUS ; CH/REG(VAL)
2: -
1: REG(AD)/BUS ; CH/ADR
2: -
1: REG(VAL)/BUS ; CH/RM
2: ECR/MEM

```

microprogramme de chargement des memoires

Les registres notes AD et VAL sont 2 des 16 registres generaux. Le microprogramme est effectue en 6 cycles. Le temps de chargement de l'ensemble des memoires est egal au temps de chargement de la memoire

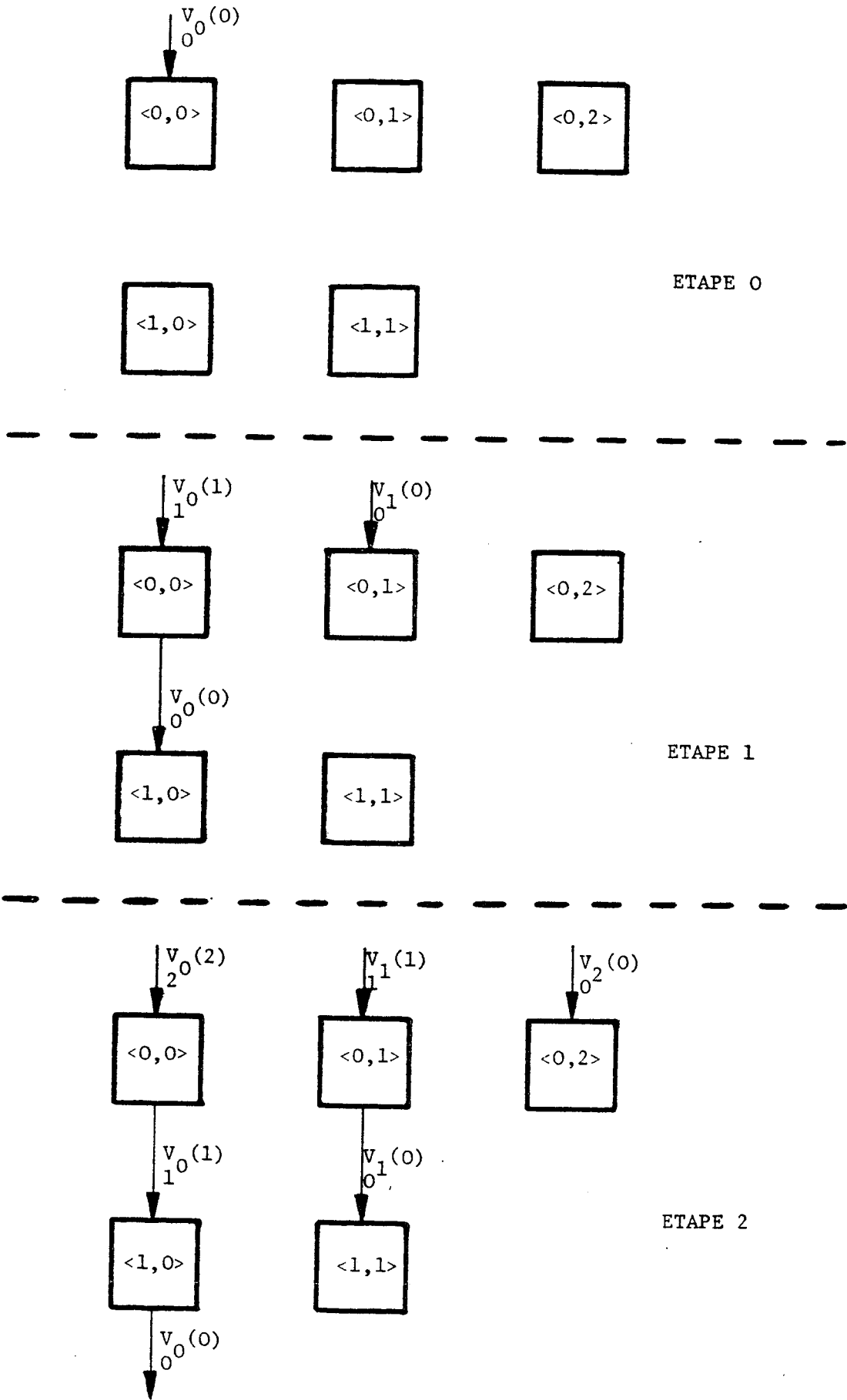


Figure 39: fonctionnement du reseau pour le chargement memoire

du processeur  $\langle 0,0 \rangle$  auquel doit être ajouté le temps de vidage du pipeline. Soient  $D$  le nombre de diagonales du système et  $E$  le temps d'exécution du microprogramme (exprime en nombre de cycles). Le nombre de valeurs à charger par mémoire étant de 60, le nombre de cycles machines (note  $C_m$ ) nécessaire au chargement de toutes les mémoires est donné par la formule:

$$C_m = 60 * E + (D-1) * E$$

ou encore

$$C_m = (D + 59) * E$$

A titre indicatif, avec  $E=6$  et  $D=25$ , on obtient  $C_m=504$  cycles.

## 6.2-Chargement des valeurs $X_i$ et $Q_i(X_i)$

Avant de lancer le traitement en pipeline sur les mots, les processeurs doivent enregistrer dans des registres des valeurs de deux types. D'une part, les valeurs de probabilités des 3 phonèmes composant un segment  $X_i$  et d'autre part, les probabilités d'insertions associées. Comme dans le cas de la mémoire, les processeurs d'une même colonne possèdent les mêmes informations. Cependant, le principe de chargement de ces valeurs se fait sur un mode différent de celui du chargement des mémoires vu dans le paragraphe précédent. Les valeurs étant à ranger dans des registres généraux, il n'est pas possible d'envoyer à un processeur une adresse, puis la valeur à ranger à cette adresse. En effet les registres ne sont adressables que par l'extérieur, notamment par les contrôleurs diagonaux (CD). Les CD fonctionnent de façon synchrone et engendrent les mêmes microinstructions ce qui signifie que les registres généraux doivent être chargés en parallèle.

Le principe de fonctionnement adopté consiste, dans un premier temps, à diffuser la valeur à charger sur tous les ports d'entrées des processeurs d'une même colonne en mode pipeline, puis dans un deuxième temps, à diffuser à tous les processeurs une instruction de transfert de la valeur du port d'entrée vers le registre adéquat.

### 6.2.1-Programme de chargement des registres

Le programme de chargement est le suivant:

```

Debut
{ Diffusion d'une valeur sur une colonne de M processeurs }
  Faire M Fois
    afficher (valeur);
    lire (vertical, valeur)
  Fait;
{ Transfert de la valeur dans le registre R }
  R:=valeur
Fin

```

programme 4: Chargement des registres

Pendant l'exécution de ce programme, tous les processeurs sont actifs. On suppose, de plus que la valeur à enregistrer est disponible pendant tout le temps de la boucle faire. De cette façon la valeur se propage de processeur en processeur. Lorsque la valeur a atteint le

dernier processeur d'une colonne, la commande de chargement dans un registre est envoyée.

### 6.2.2-Temps de chargement des registres

Le programme 4 precedent peut etre microprogramme de la facon suivante:

```

1: RV/BUS ; CH/RS      }  M
2: CH/RV                }  fois
1: RV/BUS ; CH/REG(R)
2: -

```

microprogramme de chargement des registres

Le nombre de valeurs a enregistrer est de 6. Le temps de chargement des registres, compte en nombre de cycles machines et note Cr, est donc donne par:

$$Cr = (M+1) * 6$$

Avec M=10, Cc vaut 66 cycles.

### 6.3-Programmation d'un cycle processeur

Le programme execute par un processeur se decompose en 2 phases. D'une part, une phase de transmission d'information dans laquelle le processeur recoit des donnees de ces voisins et transmet des donnees a ces voisins. D'autre part, une phase de calculs dans laquelle le processeur determine les valeurs de Li, Lo et Lc.

#### 6.3.1-Transmission d'information

Avant d'effectuer tout calcul, le processeur doit recevoir les informations suivantes:

- NY: nom du nouveau phoneme Y a traiter;
- YI, YC, YO: probabilites associees a Y;
- LI, LC, LO: resultats de processeurs voisins.

Par ailleurs, il doit envoyer les valeurs de meme type vers ces voisins. Le programme 5 est execute par tous les processeurs actifs pendant la phase de transmission.

Debut

```

Afficher(LO); Lire(Vertical,LO);
Afficher(NY); Lire(Horizontal,NY);
Afficher(YI); Lire(Horizontal,YI);
Afficher(YC); Lire(Horizontal,YC);
Afficher(YO); Lire(Horizontal,YO);
Afficher(LI); Lire(Horizontal,LI);
Afficher(LC_EN_ATTENTE); Lire(Horizontal,NOUVEAU_LC);
Afficher(LC); Lire(Vertical,LC_EN_ATTENTE); LC:=NOUVEAU_LC

```

Fin

#### Programme 5: transfert d'informations

Pour les valeurs LO, NY, YI, YC, YO et LI les transferts s'effectuent de facon simple: l'ancienne valeur est affichee sur le

port de sortie et la nouvelle valeur est lue sur l'un des ports d'entrees. Le cas de la valeur LC est plus complexe puisqu'un transfert diagonal doit etre emule. Il a ete vu que le transfert diagonal est remplace par 2 transferts successifs, notamment, un transfert vertical suivi d'un transfert horizontal (cf paragraphe 3.3.2). Un processeur contient donc a tout moment 2 valeurs de LC: d'une part, la valeur qu'il vient de calculer et d'autre part, une valeur LC (notee LC\_EN\_ATTENTE) en cours de tranfert diagonal, c'est a dire une valeur ayant effectue seulement un transfert horizontal. Pour obtenir la nouvelle valeur LC, le processeur recoit par transfert horizontal la valeur LC\_EN\_ATTENTE. L'ancienne valeur LC, quant a elle, est envoyee par transfert vertical dans la variable LC\_EN\_ATTENTE. Pour ce faire, la variable NOUVEAU\_LC est utilisee temporairement.

### 6.3.2-Phase de calculs

Dans ce paragraphe, nous reprenons d'une facon plus precise le programme 2 decrit dans le paragraphe 4.1.2. En particulier, les instructions sont plus detaillees de facon a refleter la structure du processeur. La fonction SOMLOG, par exemple, qui calcule la valeur  $\text{Log}(a+b)$  en fonction de  $\text{Log}(a)$  et  $\text{Log}(b)$ , est remplacee par une macro et est entierement programme.



```

PROGRAMME CALCULS;
  MACRO SOMLOG(K1,K2);
  { cette macro range dans le registre K1 la valeur  }
  { suivante:      max(K1,K2) + Z(-abs(K1-K2))      }
  Debut
    K1:=K1-K2; SIGNE:=(K1>0);
    Si SIGNE
      Alors Debut
        K1:=-K1;
        K2:=K2 - K1;
      Fin;
    K1:=K2 + PLA(K1);
  Fin { Macro SOMLOG };

```

```

Debut { Programme }
  { calcul de L }
  SOMLOG(Lo,Li);
  SOMLOG(Lo,Lc);
  L:= Lo;
  { calcul de Li }
  T1:=X1+I1;
  T2:=X2+I2;
  SOMLOG(T1,T2);
  T2:=X3+I3;
  SOMLOG(T1,T2);
  Li:=L+YI+T1;
  { calcul de Lc }
  Adr:=NY;
  T1:=X1+Qc[Adr];
  Adr:=Adr+1;
  T2:=X2+Qc[Adr];
  SOMLOG(T1,T2);
  Adr:=Adr+1;
  T2:=X3+Qc[Adr];
  SOMLOG(T1,T2);
  Lc:=L+YC+T1;
  { calcul de Lo }
  Lo:=L+Yo
Fin { programme }

```

Programme 6: calculs

### 6.3.3-Temps d'execution

Les microprogrammes correspondants aux phases de transferts et de calculs sont donnes en annexe. On note que le nombre de cycles des phases de tranfert et de calcul, notes Ct et Cc respectivement, sont de 18 et de 87 cycles. Le temps total d'execution du programme de base, note Cp, est donc de 105 cycles.

### 6.4-Rafraichissement de la memoire

Periodiquement, la memoire des processeurs doit etre rafraichie. En effet, l'information est conservee sous forme de charge electrique sur la grille d'un transistor, et cette charge se dissipe apres quelques millisecondes.

#### 6.4.1-Programme de rafraichissement

Le programme de rafraichissement consiste a lire chaque mot memoire puis a le reecrire. Nous avons vu dans la paragraphe que la memoire est structuree en 3 blocs de 20 elements. Une adresse memoire est formee par un doublet (A,B) ou A represente une adresse a l'interieur d'un bloc et B, le nuero du bloc. A appartient a l'intervalle [0,19] et est code sur 5 bits. B appartient a l'intervalle [0,2] et est code sur 2 bits. Le programme de rafraichissement consiste a balayer les adresses memoires entre 0 et 78 correspondants respectivement aux couples (A,B)=(0,0) et (A,B)=(19,2). Le programme 7 suivant procede au rafraichissement.

```

Debut
  Adr:=0;
  Faire 79 fois
    Debut
      Mem[Adr]:=Mem[Adr];
      Adr:=Adr+1
    Fin
  Fin

```

Programme 7: rafraichissement des memoires

#### 6.4.2-Temps d'execution

Le microprogramme associe au programme 7 est le suivant:

```

1: -
2: 0 --> ACC
1: ACC/BUS; CH/RAM      } repeter
2: LIRE.MEM             } 79
1: RM/BUS; CH/RM;      } fois
2: ECR.MEM; ACC+1 --> ACC }

```

microprogramme de rafraichissement

Ce microprogramme est execute simultanement sur tous les processeurs a la fin d'une phase de calculs par exemple. Le nombre de cycles necessaire au rafraichissement, note Cf, est de 159 cycles.

#### 6.5-Estimation des performances du multiprocesseur

Lors du fonctionnement en mode pipeline sur les mots, le traitement a effectuer se decompose ainsi:

- chargement des memoires;
- chargement des registres;
- execution multiple du programme de base entrecoupee de rafraichissements memoire.

Ce traitement doit etre renouvele pour chaque suite de segments  $X(i), \dots, X(i+P)$ . Si l'on desire detecter les mots d'un vocabulaire de  $d$  mots en temps reel, cela signifie que ce traitement devra s'effectuer pendant la duree de prononciation moyenne d'un phoneme. Nous allons estimer la taille de vocabulaire  $d$  que l'on peut esperer traiter. Trois hypotheses sont necessaires pour ce calcul:

- la duree moyenne d'un phoneme est estime a 100 millisecondes;
- la duree d'un cycle d'horloge de la machine est estimee a environs 500 nanosecondes (estimation grossiere et pessimiste);
- nous supposerons, enfin, que la memoire dynamique des processeurs doit etre rafraichie toutes les 2 millisecondes, soit 50 fois pendant le traitement d'un phoneme.

Nous noterons  $T_m$ ,  $T_r$ ,  $T_f$ ,  $T_p$ , les temps d'executions associes aux valeurs  $C_m$ ,  $C_r$ ,  $C_f$ ,  $C_p$  respectivement. Pendant la periode de 100 millisecondes, le temps passe pour les initialisations et les rafraichissements (note  $T_i$ ) est donne par la formule:

$$T_i = T_m + T_r + 50 \times T_f$$

soit  $T_i = 4.3$  millisecondes.

Soit  $T_u$ , le temps utilisable pour les calculs.  $T_u$  vaut 95.7 millisecondes. La taille du vocabulaire est donc donnee par:

$$d = T_u / T_p$$

soit environ 1800 mots. Cette valeur est tres nettement superieure a la valeur de 1000 mots envisagee initialement.

Annexe du paragraphe 6

Dans cette annexe, nous decrivons les microprogrammes de transferts de donnees et de calculs executes pendant un cycle processeur.

Traduction du programme de transfert de donnees

```
1: REG[Lo]/Bus; CH/RS;
2: CH/RV;
1: RV/Bus; CH/REG[Lo];
2: -
1: REG[NY]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[NY];
2: -
1: REG[YI]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[YI];
2: -
1: REG[YC]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[YC];
2: -
1: REG[YO]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[YO];
2: -
1: REG[Li]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[Li];
2: -
1: REG[Lc_en_attente]/Bus; CH/RS;
2: CH/RH;
1: RH/Bus; CH/REG[nouveau_Lc];
2: -
1: REG[Lc]/Bus; CH/RS;
2: CH/RV;
1: RH/Bus; CH/REG[Lc_en_attente];
2: Ø --> Acc;
1: REG[nouveau_Lc]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Lc];
2: -
```

nombre de cycles de la phase de TRANSFERT: 18

## Traduction de la macro SOMLOG

```

2: Ø --> Acc;
1: REG[K2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[K1]/Bus; CH/RA;
2: RA - Acc --> Acc;
1: Acc/Bus; CH/REG[K1];
2: * Acc --> Acc;
1: Acc/Bus; CH/REG[K1];
2: -
1: REG[K2]/Bus; CH/RA;
2: RA * Acc --> Acc;
1: REG[K1]/Bus; CH/RPLA;
2: -
1: PLA/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[K1];

```

nombre de cycles: 8

Dans les microprogrammes qui suivent, un registre note T2 est utilise. Le nombre de registres etant limite a 16, il s'agit en fait du meme registre que celui qui contient la valeur de Lo.

## Traduction du calcul de L

```

1: -
SOMLOG(Lo,Li);
SOMLOG(Lo,Lc);
2: -
1: Acc/Bus; CH/REG[L];

```

nombre de cycles: 17

Traduction du calcul de Li

```
2: Ø --> Acc;
1: REG[X1]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[I1]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T1];
2: Ø --> Acc;
1: REG[X2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[I2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: Ø --> Acc;
1: REG[X3]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[I3]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: -
1: REG[YI]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[L]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Li];
```

nombre de cycles: 28

Traduction du calcul de Lc

Dans ce microprogramme, un registre note Adr est utilise. En fait, le nombre de registre etant limite a 16, ce registre est le meme que le registre Lc.

```
2: Ø --> Acc;
1: REG[NY]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Adr];
2: -
1: REG[Adr]/Bus; CH/RAD;
2: LIRE/MEMOIRE; Ø --> Acc;
1: RM/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[X1]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T1];
2: Ø --> Acc;
1: REG[Adr]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: -
2: Acc + 1 --> Acc;
1: Acc/Bus; CH/REG[Adr];
2: -
1: Acc/Bus; CH/RAD;
2: LIRE/MEMOIRE; Ø --> Acc;
1: RM/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[X2]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: Ø --> Acc;
1: REG[Adr]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: -
2: Acc + 1 --> Acc;
1: Acc/Bus; CH/RAD;
2: LIRE/MEMOIRE; Ø --> Acc;
1: RM/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[X3]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[T2];
SOMLOG(T1,T2);
2: -
1: REG[YC]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: REG[L]/Bus; CH/RA;
2: RA + Acc --> Acc;
1: Acc/Bus; CH/REG[Lc];
```

nombre de cycles: 38

Traduction du calcul de Lo

```
2: 0 --> Acc;  
1: REG[YO]/Bus; CH/RA;  
2: RA + Acc --> Acc;  
1: REG[L]/Bus; CH/RA;  
2: RA + Acc --> Acc;  
1: Acc/Bus; CH/REG[Lo];  
2: -
```

nombre de cycles: 4

nombre de cycles de la phase CALCULS: 87

nombre total de cycles du programme de base d'un processeur: 105



## Conclusion

Les travaux presentes dans ce rapport concernent la conception d'un circuit integre specialise pour une application de detection de mots dans la parole continue. Nous avons tout d'abord repris les travaux anterieurs et expose la methode de detection de mots puis l'algorithme parallele qui a ete imagine. Nous avons ensuite montre les differentes etapes qui ont abouti a la definition de la puce de silicium.

Dans une premiere etape, nous avons defini l'architecture du multiprocesseur capable de supporter l'algorithme parallele de detection. La machine est composee d'un tableau de processeurs interconnectes de facon reguliere et locale et fonctionnant de facon synchrone. Chaque processeur possede deux ports d'entrees et un port de sortie. Le controle de l'ensemble est realise grace a un controleur general qui diffuse un flot d'instruction vers les processeurs. Ces instructions transitent par des controleurs diagonaux qui gerent une diagonale de processeurs.

Dans une seconde etape, nous avons defini les fonctions devant etre realisees par chaque processeur. L'algorithme que chaque processeur doit derouler nous a permis d'inventorier les fonctions qu'il doit realiser. Ces fonctions sont de deux ordres: d'une part, des fonctions de calculs et d'autre part, des fonctions de memorisation. Les calculs a effectuer sont des calculs de probabilite faisant intervenir des sommes de produits. Le passage aux logarithmes a permis de simplifier les operations a effectuer qui se reduisent a des additions, soustractions et consultation de table. En ce qui concerne les fonctions de memorisation du processeur, elles ont ete reduites au minimum puisque chaque processeur contient un memoire de soixante mots et un ensemble de 16 registres.

Dans une troisieme etape, l'architecture d'un processeur a ete definie. Le processeur est constitue principalement de 4 modules: un module memoire ou sont stockees des valeurs de probabilites de confusions entre phonemes; une unite arithmetique et logique capable d'effectuer des operations elementaires telles que addition, soustraction, incrementation; un tableau de registres servant de memoire de travail; un module note Z, utilise pour tabuler une fonction.

Dans une quatrième étape, les différents modules constituant le processeur ont été abordés et les circuits associés conçus. Les masques des circuits ont été dessinés ensuite, sachant que les circuits seraient réalisés dans une technologie NMOS 5 microns. Le circuit définitif s'intègre sur une puce de silicium de 5mm sur 6mm et comporte environ 12000 transistors. Il comporte 62 plots d'entrées/sorties qui se décomposent en 3 groupes de 16 plots pour les ports d'entrées/sorties proprement dits, 10 plots recevant les microinstructions et 4 plots pour l'alimentation, la masse et les signaux d'horloge.

Dans une dernière partie, nous nous sommes intéressés aux performances du processeur. Nous avons montré comment initialiser le multiprocesseur puis comment le programmer. Les différents microprogrammes ont été donnés ce qui nous a permis d'évaluer les performances de la machine. Nous avons montré en particulier, que la machine était capable de traiter le problème de la détection de mots en temps réel sur un vocabulaire d'environ 1800 mots alors que l'objectif initial était de traiter environ 1000 mots.

#### Perspectives

La méthodologie de conception de circuits intégrés préconisée par Mead et Conway, semble prometteuse. Si l'on s'en tient à notre propre expérience, 6 semaines de cours suivis de 8 mois de travail ont permis le dessin d'un circuit intégré d'environ 12000 transistors. Les performances estimées permettent d'envisager un fonctionnement des circuits sur un vocabulaire de près de 2 mille mots, alors que nous avons tablé sur un vocabulaire d'un millier de mots.

Le circuit a été dessiné en visant une technologie NMOS à 5 microns. Cette technologie est loin d'être la plus sophistiquée puisqu'au niveau industriel une technologie de moins de 3 microns est couramment utilisée. Par ailleurs, au niveau de la recherche, on est descendu déjà au dessous du micron. Cela signifie qu'il est désormais possible d'intégrer sur une même puce de silicium plusieurs processeurs comme celui décrit dans ce rapport. A moyen terme, il est clair que le multiprocesseur complet pourra être intégré. Cependant, 2 problèmes importants concernant une telle intégration méritent d'être soulignés. D'une part, la consommation du circuit risque d'être énorme et une technologie de type CMOS, beaucoup moins consommatrice de courant, doit être considérée. D'autre part, l'intégration du multiprocesseur complet pose le problème du nombre de plots d'entrées/sorties. Une façon de limiter ce nombre consiste à effectuer des entrées/sorties en mode série.

A plus court terme, notre étude va se poursuivre dans 2 domaines principaux. D'une part, avant de fabriquer le circuit intégré, nous nous proposons d'effectuer des simulations du circuit à deux niveaux: niveau logique et niveau électrique, afin de vérifier le fonctionnement logique du circuit et d'estimer de façon assez précise, ses performances. D'autre part, nous envisageons la fabrication du circuit et son test.

## Bibliographie

1. Kung H.T., Why Systolic Architectures?, Computer vol 15, no 1, Janvier 1982, pp 37-46.
2. Mercier G., Nouhen A., Quinton P., Siroux J., The Keal Speech Understanding System, in Spoken Language Generation and Understanding, J.C. Simon Editor, Proceedings of the NATO ASI, Bonas, 1980.
3. Banatre J.P., Frison P., Quinton P., A Network for the Detection of Words in Continuous Speech, VLSI Int Conf, J.P. Gray Editor, Academic Press, 1981.
4. Bahl L.R., Jelinek F., Decoding for Channels with Insertions, Deletions, and Substitutions with Application to Speech Recognition, IEEE Trans. on Information Theory, 21, no 4, Juillet 1976, pp 404-411.
5. Landman H.A., MAKE PLA Users' Guide, ERL, University of California, Berkeley, 1981.
6. Mead C., Conway L., Introduction to VLSI Systems, Addison-Wesley, 1980.
7. Hon R.W., Sequin C.H., A Guide to LSI Implementation, Xerox PARC Technical Report SSL-79-7, Palo Alto, California, 1980.

## Liste des Publications Internes IRISA

- PI 150 **Construction automatique et évaluation d'un graphe d'«implication» issu de données binaires, dans le cadre de la didactique des mathématiques**  
H. Rostam , 112 pages : Juin 1981
- PI 151 **Réalisation d'un outil d'évaluation de mécanismes de détection de pannes]-]Projet Pilote SURF**  
B. Decouty, G. Michel, C. Wagner, Y. Crouzet , 59 pages : Juillet 1981
- PI 152 **Règle maximale**  
J. Pellaumail , 18 pages : Septembre 1981
- PI 153 **Corrélation partielle dans le cas « qualitatif »**  
I.C. Lerman , 125 pages : Octobre 1981
- PI 154 **Stability analysis of adt ptively controlled not-necessarily minimum phase systems with disturbances**  
Cl. Samson , 40 pages : Octobre 1981
- PI 155 **Analyses d'opinions d'instituteurs à l'égard de l'appropriation des nombres naturels par les élèves de cycle préparatoire**  
R. Gras , 37 pages : Octobre 1981
- PI 156 **Récursion induction principle revisited**  
G. Boudol, L. Kott , 49 pages : Décembre 1981
- PI 157 **Loi d'une variable aléatoire à valeur  $R^+$  réalisant le minimum des moments d'ordre supérieur à deux lorsque les deux premiers sont fixés**  
M.Kowalowka, R. Marie , 8 pages : Décembre 1981
- PI 158 **Réalisations stochastiques de signaux non stationnaires, et identification sur un seul échantillon**  
A. Benveniste J.J. Fuchs , 33 pages : Mars 1982
- PI 159 **Méthode d'interprétation d'une classification hiérarchique d'attributs-modalités pour l'«explication» d'une variable ; application à la recherche de seuil critique de la tension artérielle systolique et des indicateurs de risque cardiovasculaire**  
B. Tallur , 34 pages : Janvier 1982
- PI 160 **Probabilité stationnaire d'un réseau de files d'attente multiclasse à serveur central et à routages dépendant de l'état**  
L.M. Le Ny , 18 pages : Janvier 1982
- PI 161 **Détection séquentielle de changements brusques des caractéristiques spectrales d'un signal numérique**  
M. Basseville, A. Benveniste , pages : Mars 1982
- PI 162 **Actes regroupés des journées de Classification de Toulouse (Mai 1980), et de Nancy (Juin 1981)**  
I.C. Lerman , 304 pages :
- PI 163 **Modélisation et Identification des caractéristiques d'une structure vibratoire : un problème de réalisation stochastique d'un grand système non stationnaire**  
M. Prévosto, A. Benveniste, B. Barnouin , 46 pages : Mars 1982
- PI 164 **An enlarged definition and complete axiomatization of observational congruence of finite processes**  
Ph. Darondeau , 45 pages : Avril 1982
- PI 165 **Accès vidéotex à une banque de données médicales**  
A. Chauffaut, M. Dragone, R. Rivoire, J.M. Roger , 25 pages : Mai 1982
- PI 166 **Comparaison de groupes de variables définies sur le même ensemble d'individus**  
B. Escofier, J. Pages , 115 pages : Mai 1982
- PI 167 **Transport en circuits virtuels internes sur réseau local et connexion Transpac**  
M. Tournois, R. Trepos , 90 pages : Mai 1982
- PI 168 **Impact de l'intégration sur le traitement automatique de la parole**  
P. Quinton , 14 pages : Mai 1982
- PI 169 **A systolic algorithm for connected word recognition**  
J.P. Banâtre, P. Frison, P. Quinton , 13 pages : Mai 1982
- PI 170 **A network for the detection of words in continuous speech**  
J.P. Banâtre, P. Frison, P. Quinton , 24 pages : Mai 1982
- PI 171 **Le langage ADA : Etude bibliographique**  
J. André, Y. Jégou, M. Raynal , 12 pages : Juin 1982
- PI 172 **Comparaison de groupes de variables : 2ème partie : un exemple d'application**  
B. Escofier, J. Pajès , 37 pages : Juillet 1982
- PI 173 **Unfold-fold program transformations**  
L. Kott , 29 pages : Juillet 1982
- PI 174 **Remarques sur les langages de parenthèses**  
J.M. Autebert, J. Beauquier, L. Boasson, G. Senizergues , 20 pages : Juillet 1982
- PI 175 **Langages de parenthèses, langages N.T.S. et homomorphismes inverses**  
J.M. Autebert, L. Boasson, G. Senizergues , 26 pages : Juillet 1982
- PI 176 **Tris pour machines synchrones ou Baudet Stevenson revisited**  
R. Rannou , 26 pages : Juillet 1982
- PI 177 **Un nouvel algorithme de classification hiérarchique des éléments constitutifs de tableau de contingence basé sur la corrélation**  
B. Tallur , Juillet 1982 :
- PI 178 **Programmes d'analyse des re'sultats d'une classification automatique**  
I.C. Lerman et collaborateurs , 79 pages : Septembre 1982
- PI 179 **Attitude à l'égard des mathématiques des élèves de sixième**  
J.Degouys, R. Gras, M. Postic , 29 pages : Septembre 1982
- PI 180 **Traitements de textes et manipulations de documents : bibliographie analytique**  
J. André , 20 pages : Septembre 1982

- PI 181 **Algorithme assurant l'insertion dynamique d'un processeur autour d'un réseau à diffusion et garantissant la cohérence d'un système de numérotation des paquets global et réparti**  
Annick Le Coz, Hervé Le Goff, Michel Ollivier, 31 pages ; Octobre 1982
- PI 182 **Interprétation non linéaire d'un coefficient d'association entre modalités d'une juxtaposition de tables de contingence**  
Israël César Lerman, 34 pages ; Novembre 1982
- PI 183 **L'IRISA vu à travers les stages effectués par ses étudiants de DEA (1<sup>ère</sup> année de thèse)**  
Daniel Herman, 41 pages ; Novembre 1982
- PI 184 **Commande non linéaire robuste des robots manipulateurs**  
Claude Samson, 52 pages ; Janvier 1983
- PI 185 **Dialogue et représentation des informations dans un système de messagerie intelligent**  
Philippe Besnard, René Quiniou, Patrice Quinton, Patrick Saint-Dizier, Jacques Siroux, Laurent Trilling, 45 pages ; Janvier 1983
- PI 186 **Analyse classificatoire d'une correspondance multiple ; typologie et régression**  
I.C. Lerman, 54 pages ; Janvier 1983
- PI 187 **Estimation de mouvement dans une séquence d'images de télévision en vue d'un codage avec compensation de mouvement**  
Claude Labit, 132 pages ; Janvier 1983
- PI 188 **Conception et réalisation d'un logiciel de saisie et restitution de cartes élémentaires**  
Eric Sécher, 45 pages ; Janvier 1983
- PI 189 **Etude comparative d'algorithmes pour l'amélioration de dessins au trait sur surfaces point par point**  
M.A. ROY, 96 pages ; Janvier 1983
- PI 190 **Généralisation de l'analyse des correspondances à la comparaison de tableaux de fréquence**  
Brigitte Escofier, 35 pages ; Mars 1983
- PI 191 **Association entre variables qualitatives ordinales «nettes» ou «floues»**  
Israël-César Lerman, 42 pages ; Mars 1983
- PI 192 **Un processeur intégré pour la reconnaissance de la parole**  
Patrice Frison, 80 pages ; Mars 1983
- PI 193
- PI 194 **Régime stationnaire pour une file M/H/1 avec impatience**  
Raymond Marie et Jean Pellaumail, 8 pages ; Mars 1983
- PI 195 **SIGNAL : un langage pour le traitement du signal**  
Paul Le Guernic, Albert Benveniste, Thierry Gautier, 49 pages ; Mars 1983
- PI 196 **Algorithmes systoliques : de la théorie à la pratique**  
Françoise André, Patrice Frison, Patrice Quinton, 19 pages ; Mars 1983

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique



