Un Réseau systolique intégré pour la correction de fautes de frappe

Dominique Lavenier

UNITÉ DE RECHERCHE
INRIA-RENNES

Rapports de Recherche

1992

25ème anniversaire

N° 1755

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

UN RÉSEAU SYSTOLIQUE INTÉGRÉ POUR LA CORRECTION DE FAUTES DE FRAPPE

Dominique LAVENIER

Octobre 1992

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

RR-1755

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tel. : 99 84 71 00 - Télex . UNIRISA 950 473 F
Télécopie : 99 38 38 32

Un réseau systolique intégré pour la correction de fautes de frappe

An integrated systolic array for spelling correction

Dominique LAVENIER

Publication Interne n° 670 - 120 pages - Programme 1

Juillet 1992

résumé

Ce rapport présente la réalisation d'un circuit VLSI spécialisé pour la correction de fautes de frappe. L'architecture du circuit est basée sur une structure régulière, un réseau systolique bidimensionnel de 69 processeurs. La méthodologie suivie pendant la conception du circuit tire profit de cette régularité, notamment pendant les phases de validation.

abstract

This report describes the realization of a VLSI circuit for spelling correction. The circuit is highly regular, with a structure based on a 2D systolic array of 69 processors. The design methodology presented takes advantage of this regularity, especially during the validation phase.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE (U.R.A 227) UNIVERSITE DE RENNES I INSA DE RENNES
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (UNITE DE RECHERCHE DE RENNES)

Introduction

Ce rapport présente la mise en œuvre d'un algorithme parallèle pour la correction de fautes de frappe sur une architecture systolique intégré. Le réseau est composé de 69 processeurs connectés suivant une topologie bidimensionnelle incomplète (le réseau est une matrice 2 D de 15 × 15 processeurs sur laquelle 5 diagonales ont été conservées).

Le rapport est organisé en 10 chapitres traitant respectivement de l'architecture du circuit, de l'implémentation VLSI, de la méthode de conception et de la réalisation.

Le premier chapitre est cependant consacré à une description générale du circuit. L'algorithme de correction est présenté ainsi que la structure globale du circuit qui résulte de la parallélisation. Ce tour d'horizon rapide permet de mieux situer, par la suite, les différents éléments qui composent le circuit.

Les chapitres 2 à 6 traitent chacun d'une partie distincte du circuit. En effet, celui-ci peut être découpé en 5 modules ayant chacun un rôle particulier. L'organisation de ces 5 chapitres est identique : une description fonctionnelle du module est d'abord donnée, puis l'architecture est détaillée. L'aspect VLSI est ensuite abordé en s'attachant, à chaque fois, à montrer le coté régulier de la mise en œuvre sur silicium. La description du plan de masse termine la description de chaque chapitre.

Le chapitre 7 rassemble ces différentes parties et montre comment une structure VLSI régulière peut être obtenue à partir des différents éléments décrits aux chapitres précédents.

Le chapitre 8 aborde la méthodologie de conception suivie pour réaliser la puce. Elle tient compte de la très grande régularité de l'implémentation physique du circuit et est essentiellement basée sur l'étude d'une version réduite : *le noyau.*

L'avant dernier chapitre est consacré à la programmation. Pour illustrer les possibilités du circuit, les différentes étapes d'un algorithme de correction simplifié sont prises comme exemple.

Le dernier chapitre présente la réalisation, non pas du circuit complet, mais d'une version plus réduite, comprenant 34 processeurs. Ce circuit, baptisé API34, a permis de valider notre méthodologie de conception et dispose de ressources suffisantes pour être utilisé dans le cadre de la correction de fautes de frappe interactive.

Table des matières

Chapitre 1

Présentation du circuit

1.1 Introduction

L'apparition des circuits à très haute intégration (VLSI) à la fin des années 70 a relancé l'intérêt des chercheurs pour les structures cellulaires. En particulier, H. T. Kung a introduit le modèle de calcul systolique pour réaliser sur des VLSI des applications nécessitant énormément de calculs élémentaires [6]. De nombreuses recherches ont été effectuées dans les années 80 et ont permis de découvrir de nombreux algorithmes applicables à des structures cellulaires dans des domaines très variés (traitement du signal, de l'image, traitement et reconnaissance de la parole, etc.). Parallèlement, de nombreux travaux ont été entrepris pour mettre au point des méthodes automatiques de mise en œuvre d'algorithmes sur des réseaux de cellules [10]. Jusqu'à présent, peu de produits industriels ont émergé de ces recherches, excepté le circuit systolique GaPP contenant un réseau de 6 x 12 processeurs 1 bit [4]. Cela est dû principalement au fait que la technologie VLSI des années 80 (quelques centaines de milliers de transistors par puce chez les grands constructeurs) ne permettait pas une intégration suffisante pour réaliser, dans un seul boîtier, un réseau systolique pour une application réelle. Toutes les conditions sont désormais réunies pour que les années 90 voient apparaître des circuits systoliques de plusieurs millions de transistors réalisant des algorithmes complexes.

Dans ce rapport, nous présentons la réalisation d'un circuit systolique intégré pour la correction de fautes de frappe. Ce circuit comporte, dans sa version de base, 69 processeurs 8 bits, organisé en un réseau matriciel tronqué de 5 diagonales. La technologie actuelle permet d'intégrer ce réseau d'environ 300 000 transistors sur une seule puce.

La première partie de ce chapitre est consacrée à la description du problème à résoudre. Elle est suivie par une présentation de l'algorithme permettant un traitement efficace du problème sur une structure de type systolique. Nous abordons ensuite la mise en œuvre sur silicium en précisant les choix architecturaux qui ont été faits. Nous terminons en décrivant brièvement la méthode suivie avant d'indiquer les performances du circuit.

1.2 Problème et algorithme

L'utilisation croissante de l'informatique a multiplié les moyens de communication homme-machine. Cependant, le médium de communication le plus répandu reste le clavier. Malheureusement la saisie au clavier n'est pas très fiable et l'opérateur, généralement peu expert en dactylographie, commet souvent de nombreuses fautes de frappe. L'objectif de notre étude est de pouvoir corriger en temps réel des textes ayant un vocabulaire non limité par la taille des dictionnaires (le dictionnaire français étendu à toutes ses formes fléchies comporte environ 700.000 mots!). Ce problème est loin d'être trivial même si la plupart des traitements de texte du marché proposent un mode de correction. En effet, la plupart du temps, la méthode de correction est rudimentaire pour des raisons d'efficacité et n'apporte donc pas une correction satisfaisante.

Le circuit que nous présentons a pour but d'accélérer la correction d'un mot erroné. La détection d'un mot erroné est faite par consultation d'un dictionnaire : si un mot du texte n'est pas trouvé dans le dictionnaire il est considéré comme erroné. Le principe consiste ensuite à *comparer* ce mot avec tous les mots du dictionnaire (ou, plus généralement, avec ceux qui ont une longueur similaire) pour trouver celui qui lui *ressemble* le plus. Toute la difficulté réside dans la méthode de comparaison.

Une des meilleures méthodes consiste à calculer une *distance* entre le mot erroné et tous les mots du dictionnaire ayant une longueur similaire (par exemple $\pm$ 2 caractères) afin d'extraire la distance minimale qui permet de donner la bonne correction [5] [8] [7]. Malheureusement la complexité de l'algorithme calculant cette distance est trop grande pour envisager une mise en œuvre en temps réel sur une machine classique (PC ou station de travail). Le concept de réseau systolique, appliqué à un réseau 2D de processeurs, fournit une mise en œuvre efficace pour ce type de calcul [1] [2] [3].

1.2.1 Méthode de calcul

Wagner et Fisher [11] proposent de comparer deux chaînes de caractères à l'aide d'une distance, appelée **distance d'édition**, qui représente le coût minimal pour passer d'une chaîne à l'autre au moyen d'un certain nombre d'opérations élémentaires : les **opérations d'édition**. On distingue trois opérations d'édition élémentaires :

- **les substitutions** : un caractère est remplacé par un autre : $t \rightarrow r$
  exemple : systolique $\rightarrow$ sysrolique,
- **les insertions** : un caractère est ajouté : $\wedge \rightarrow t$
  exemple : systolique $\rightarrow$ systtolique,
- **les omissions** : un caractère est supprimé : $t \rightarrow \wedge$
  exemple : systolique $\rightarrow$ sysolique,

$\wedge$ représente l'absence de caractère.

Un mot erroné peut contenir une ou plusieurs de ces fautes. Ce dernier cas rend les méthodes de correction traditionnelles plus complexes.

A chaque opération d'édition est associé un coût : $d(a, b)$ pour l'erreur de substitution, $d(\wedge, a)$ pour l'insertion et $d(a, \wedge)$ pour l'omission.

L'algorithme proposé par Wagner et Fisher pour comparer deux séquences de caractères est basé sur le concept de programmation dynamique. Soient R et T deux chaînes de caractères de longueur respective m et n. On note r_i le caractère de la chaîne R à la position i et t_j le caractère de la chaine T à la position j. Soit $D(i, j)$, la distance entre les i premiers caractères de la chaîne R et les j premiers caractères de la chaîne T. La distance $D(m, n)$ entre R et T est donnée par la relation de récurrence suivante :

$$D(i,j) = Min \begin{cases} D(i-1, j-1) + d(r_i, t_j) \\ D(i-1, j) + d(\wedge, t_j) \\ D(i, j-1) + d(r_i, \wedge) \end{cases} \tag{1.1}$$

avec les conditions initiales suivantes :

$$\begin{aligned} &D(0,0) = 0 \\ &D(i,0) = D(i-1,0) + d(r_i, \wedge) \quad 1 \leq i \leq m \\ &D(0,j) = D(0,j-1) + d(\wedge, t_j) \quad 1 \leq j \leq n \end{aligned}$$

Les opération d'édition proposées par Wagner et Fisher sont le reflet exact des coquilles typographiques relevées communément dans les textes. Lowrance et Wagner proposent une extension [8] à cette définition en incluant une quatrième opération consistant en l'interversion de deux caractères adjacents et désignée par le terme permutation.

Cette dernière opération intéresse principalement la correction des textes où ce type de faute, d'ordre typographique, est souvent commis. L'opération de permutation peut être modélisée de deux manières, la première consiste à considérer que cette opération est divisible en deux opérations d'édition élémentaires dont la séquence est composée d'une insertion suivie d'une omission (ou l'inverse). La permutation n'est alors pas considérée comme une opération d'édition à part entière et sa pondération, par un cœfficient qui lui est propre, n'est pas possible. L'autre solution est de rajouter à l'équation précédente un terme représentant cette opération particulière :

$$D(i,j) = Min \begin{cases} D(i-1, j-1) + d(r_i, t_j) \\ D(i-1, j) + d(\wedge, t_j) \\ D(i, j-1) + d(r_i, \wedge) \\ D(i-2, j-2) + \gamma_p(r_{i-1}, r_i, t_{j-1}, t_j) \end{cases} \tag{1.2}$$

où γ_p est une fonction définie par :

$$\gamma_p(r_{i-1}, r_i, t_{j-1}, t_j) = \begin{cases} k_p \; si \; r_{i-1} = t_i \; et \; r_i = t_{i-1} \\ \infty \; si \; r_{i-1} \neq t_i \; ou \; r_i \neq t_{i-1} \end{cases} \tag{1.3}$$

k_p représente le coût affecté aux permutations.

1.2.2 Coût des erreurs

Substitution :

le coût associé à la substitution de deux caractères est directement déterminé par la topologie du clavier. Suivant la proximité des touches les unes par rapport aux autres, leur probabilité de confusion est plus ou moins élevée. On affecte donc une pénalisation inversement proportionnelle à la probabilité de substitution : deux touches proches ont un coût de substitution relativement faible, deux touches éloignées possèdent un coût élevé.

A chaque touche du clavier est associé une table de correspondance avec toutes les autres touches du clavier. Le coût de substitution est donc donné par simple consultation de cette table, que l'on appelle table de substitution.

Ainsi, le mot erroné **progesseur** qui compte une lettre de différence avec les mots **professeur** et **processeur** est corrigé, de préférence, en **professeur** car la touche **G** sur un clavier de type *QWERTY* est plus proche de la touche **F** que de la touche **C** et possède donc une meilleur probabilité de confusion.

Omission :

de la même manière, il est possible d'associer des tables correspondant aux coûts des omissions de chaque caractère. En pratique, la probabilité d'omettre un caractère est identique, quel que soit le caractère considéré. De plus, il ne semble pas que l'omission d'un caractère soit associée à la disposition des touches d'un clavier. Le coût affecté aux omissions est donc une constante (notée Ko).

Insertion :

ce sont des erreurs qui dépendent à la fois de la nature et de la topologie du clavier. Un mauvais clavier génère des rebonds qui peuvent être pris pour la saisie d'un second caractère. La frappe d'une touche peut également s'accompagner de la frappe malencontreuse d'une touche voisine à cause d'un mauvais positionnement des doigts ou des mains.

Les coûts d'insertion sont donc de même nature que les coûts affectés aux substitutions : plus deux touches sont proches l'une de l'autre, plus le coût affecté à l'insertion est faible. En pratique on peut considérer, comme pour les erreurs d'omission, que ce coût est assimilable à une constante. L'incidence sur les résultats n'est pas sans effets mais le modèle s'en trouve notablement simplifié. On note Ki ce coût.

Permutations :

l'interversion de deux caractères pendant la saisie d'un texte est une erreur relativement fréquente. Son coût est donné par γ_p où γ_p est, rappellons le, une fonction définie par :

$$\gamma_p(r_{i-1}r_i, t_{j-1}t_j) = \begin{cases} k_p & si\ r_{i-1} = t_j\ et\ r_i = t_{j-1} \\ \infty & si\ r_{i-1} \neq t_j\ ou\ r_i \neq t_{j-1} \end{cases} \tag{1.4}$$

k_p est un cœfficient qui ne tient pas compte de la topologie du clavier. En pratique, il arrive qu'une substitution se combine avec une permutation. Aussi, sachant que $d(x, x) = 0$ (coût de substitution d'un caractère avec lui-même), γ_p peut être redéfinie de la manière suivante :

$$\gamma_p(r_{i-1}r_i, t_{j-1}t_j) = d(r_{i-1}, t_j) + d(r_i, t_{j-1}) + Kp$$

S'il y a permutation sans confusion, le coût est réduit à Kp. Dans le cas contraire, la pénalisation est proportionnelle à la proximité des touches.

1.2.3 Complexité de l'algorithme

La comparaison d'un mot test avec un seul mot de référence comportant respectivement n et m caractères nécessite $n \times m$ calculs élémentaires $D(i, j)$. Un calcul $D(i, j)$ représente six additions et trois minimisations. Considérant l'addition et la minimisation comme des opérations élémentaires (de même complexité), la comparaison d'un mot test avec μ mots du dictionnaire de référence nécessite $n \times m \times 9 \times \mu$ opérations.

A titre d'exemple, la comparaison de mots d'une dizaine de lettres avec un vocabulaire de 200.000 mots en 0,1 secondes nécessite 1.8 Gops (giga opérations par seconde).

1.2.4 Parallélisation de l'algorithme

La figure 1.1 est une représentation graphique du calcul d'une équation simplifiée (les fautes de permutations ne sont pas prises en compte) où la distance entre les deux chaînes est donnée par la relation de récurrence suivante :

$$D(i, j) = Min \begin{cases} D(i-1, j-1) + d(r_i, t_j) \\ D(i-1, j) + 1 \\ D(i, j-1) + 1 \end{cases} \tag{1.5}$$

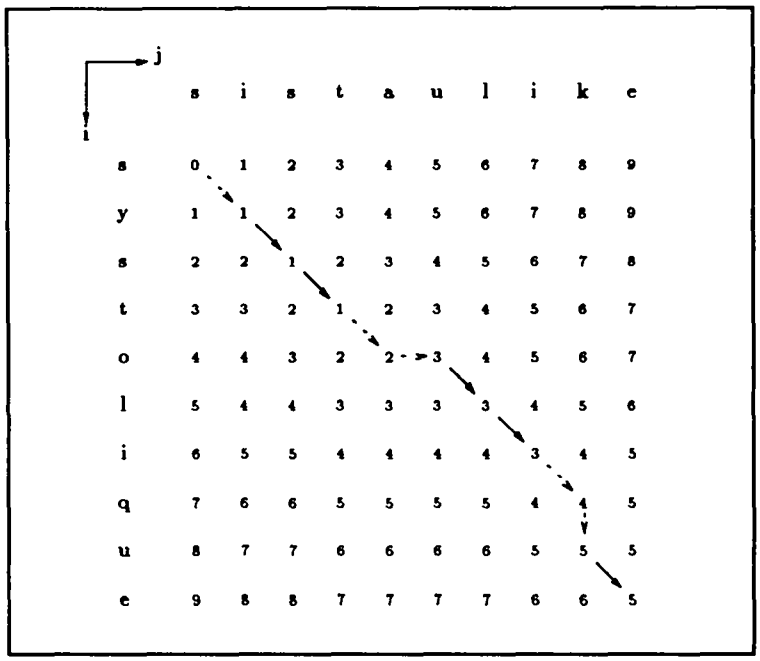


Figure 1.1 : comparaison de deux chaînes de caractères

où $d(r_i, t_j)$ vaut 0 si $r_i = t_j$ et 1 dans le cas contraire.

Le parcours représenté par les flèches montre un des chemins possibles conduisant à la solution. Les flèches en pointillé indiquent que la distance est incrémentée d'une unité suite à l'une des trois erreurs prises en considération (substitution, insertion, omission).

Une distance $D(i,j)$ est l'intersection d'une ligne et d'une colonne d'une matrice $(n \times m)$ où m s'identifie à la longueur du mot référence et n à la longueur du mot test. Le calcul de cette distance nécessite que les distances $D(i-1,j)$, $D(i-1,j-1)$ et $D(i,j-1)$ soient valides, c'est-à-dire qu'elles aient été calculées auparavant.

Supposons que le calcul d'une distance corresponde à une unité de temps Δt et qu'à l'instant t_1 le processus de calcul commence. La seule distance qui puisse être évaluée est $D(1,1)$, en supposant que les initialisations soient telles que :

$$D(0,0) = 0$$
$$D(i,0) = D(i-1,0) + 1 \quad \text{pour} \quad 1 \leq i \leq n$$
$$D(0,j) = D(0,j-1) + 1 \quad \text{pour} \quad 1 \leq j \leq m$$

En t_2, les distances $D(1,2)$ et $D(2,1)$ peuvent être calculées simultanément puisque toutes les deux requièrent la valeur de $D(1,1)$. De la même manière, $D(1,3)$, $D(2,2)$ et $D(3,1)$ peuvent être évaluées à partir de l'instant t_3.

Plus généralement, au temps t_k les distances $D(i,j)$, telles que $k = i + j - 1$, sont

évaluables. On en déduit aisément qu'il faut $n+m-1$ cycles de calcul pour parvenir au résultat final qui correspond à la distance $D(n,m)$ recherchée.

Avec une machine parallèle disposant d'au moins $n+m-1$ processeurs (nombre maximum de distances pouvant être calculées en même temps), et en reprenant l'hypothèse que l'évaluation d'une distance élémentaire s'effectue en un temps Δt, le gain obtenu, par rapport à l'exécution séquentielle, est de :

$$\Gamma = \frac{n \times m}{n+m-1}$$

1.3 Mise en œuvre sur une architecture systolique 2D

1.3.1 Principe de base

Le concept de machine systolique, dû à Kung [6], consiste à obtenir, pour résoudre un problème donné, une architecture régulière, faite de processeurs simples et identiques, connectés suivant une topologie régulière et locale (pas de liens éloignés entre processeurs) où les données circulent à vitesse constante. L'ensemble fonctionne par battements d'où le nom donné à ce type de machine.

L'idée de base est d'associer un processeur au calcul de chaque distance $D(i,j)$, comme le montre le schéma de la figure 1.2. On obtient une matrice de processeurs élémentaires connectés par l'intermédiaire de trois ports d'entrée et de trois ports de sortie. Ainsi, un processeur $P(i,j)$ reçoit respectivement les distances $D(i-1,j)$, $D(i-1,j-1)$ et $D(i,j-1)$ des processeurs $P(i-1,j)$, $P(i-1,j-1)$ et $P(i,j-1)$. Un cycle systolique correspond au temps nécessaire à un processeur pour lire les valeurs sur ses ports d'entrée, effectuer le calcul de sa distance et la propager aux processeurs voisins.

La comparaison de deux chaînes de caractères s'effectue de la manière suivante : un caractère d'indice k appartenant à la chaîne **test** (chaîne erronée) est diffusé verticalement aux processeurs $P(i,k)$, $(1 \leq i \leq m)$. De la même manière, les caractères de la chaîne de référence sont diffusés horizontalement à travers le tableau. Les valeurs présentes sur les ports d'entrée des processeurs périphériques correspondent aux valeurs d'initialisation énoncées au paragraphe 1.2.4.

Le calcul d'une comparaison suit exactement le même séquencement que celui présenté au paragraphe précédent : au temps t_1, le processeur $P(1,1)$ débute le calcul, au temps t_2, il a propagé ses résultats, ce qui autorise les processeurs $P(2,1)$ et $P(1,2)$ à commencer leur calcul de distance à cet instant. Au temps t_3 les processeurs $P(3,1)$, $P(2,2)$ et $P(1,3)$ prennent la relève et le calcul se poursuit jusqu'à ce que le processeur $P(m,n)$ ait été en mesure d'évaluer sa distance. Notons qu'à un instant donné t_k, tous les processeurs actifs se situent sur une même diagonale telle que $P(i,j)$ est actif si $k = i+j-1$.

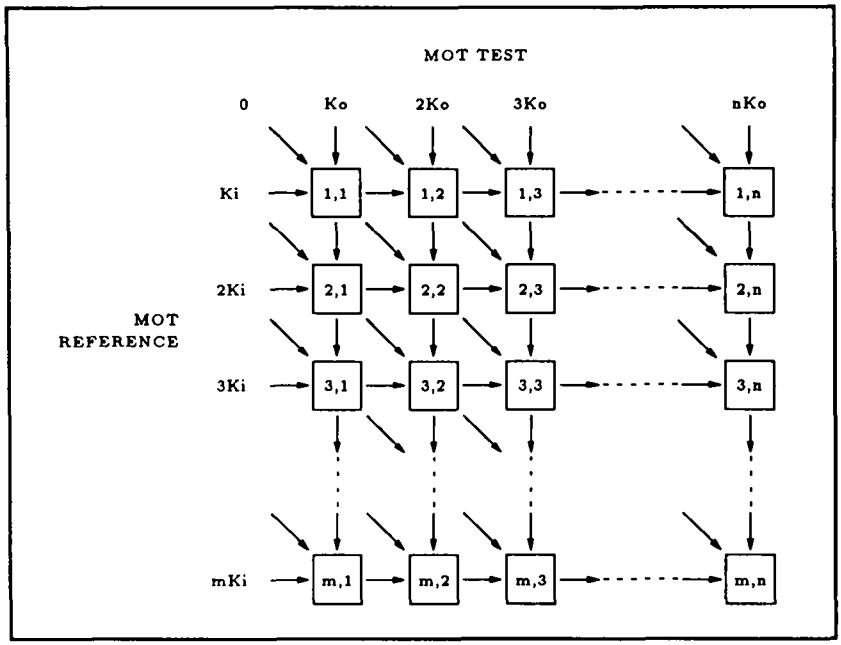


Figure 1.2 : tableau de processeurs

Dans cette approche, il faut $n + m - 1$ cycles de calcul pour obtenir le résultat d'une comparaison. Si ce chiffre correspond à la valeur théorique du gain que l'on peut espérer atteindre, $n \times m$ processeurs sont néanmoins nécessaires. L'utilisation du réseau est loin d'être optimale.

1.3.2 Pipeline des calculs

Le domaine des applications qui nous intéresse suppose qu'une chaîne test (mot erroné) puisse être comparée à un grand nombre de références de façon à évaluer celle qui est la "plus proche" ou la "plus ressemblante". D'autre part, le calcul doit être effectué le plus rapidement possible. La mise en œuvre sur une architecture systolique permet d'accélérer le processus de comparaison par enchaînement des calculs.

Partant du fait qu'une comparaison dure $n + m - 1$ cycles systoliques et qu'un processeur y participe pendant un cycle seulement, il est, par conséquent, disponible le reste du temps pour d'autres tâches. De plus, le calcul d'une comparaison se propage suivant une diagonale formée par les processeurs $P(i, j)$ vérifiant la relation $k = i+j-1$ à l'instant t_k. A chaque cycle systolique, il est donc possible d'entamer une nouvelle comparaison.

Le séquencement des données à envoyer au réseau est représenté sur la figure 1.3. Un processeur mémorise toujours le caractère de la chaîne test qui correspond à sa colonne. Par contre, les caractères de la chaîne référence transitent au rythme des

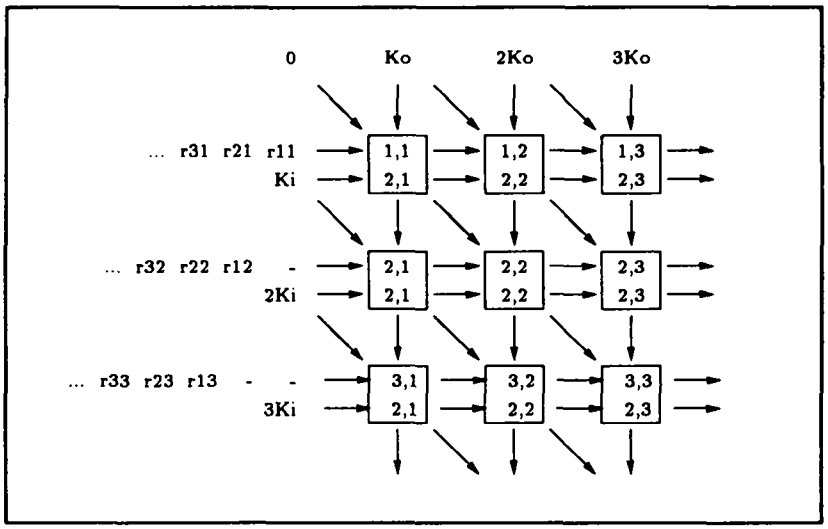


Figure 1.3 : séquencement des données sur un réseau 2-D

cycles systoliques. Ceci implique une connexion horizontale supplémentaire.

L'introduction du pipeline accroît considérablement l'efficacité du réseau, puisqu'une fois amorcé, le résultat d'une comparaison entre une chaîne test et une référence est produit à chaque cycle systolique. Pour les applications considérées (comparaison d'une instance avec quelques milliers de références), ce temps d'amorçage est négligeable. Le nombre exact de cycles systoliques requis pour la comparaison d'une chaîne de longueur n avec un ensemble de p références de longueurs identiques est de $p+2n-2$ cycles. ??

Cependant, la réalisation d'un tel réseau, aussi efficace soit-il, peut être reconsidérée devant le nombre de processeurs exigé.

1.3.3 Réduction de la taille du réseau

La taille du réseau a été fixée à 15 lettres. En effet, les mots de taille supérieure à 15 lettres sont relativement peu nombreux et leur traitement sur une machine ordinaire est tout à fait envisageable en temps réel. Cette limite peut être repoussée vers des valeurs plus grandes (20 à 25) dans les limites de la technologie disponible. Cependant, une valeur de 15 entraîne un grand nombre de processeurs (225), nombre trop important pour une intégration complète du réseau sur une seule puce. Il est donc important de bien vérifier la nécessité d'intégrer un tel réseau. Les études antérieures nous ont montré qu'en réalité, il était inutile de calculer toutes les valeurs $D(i,j)$ du tableau à deux dimensions. En effet, seules les valeurs proches de la diagonale ($i = j$) contribuent au résultat $D(m,n)$. En effet, il est clair que seuls les mots ayant des longueurs voisines peuvent obtenir une distance faible. En particulier, nous avons montré que pour un mot

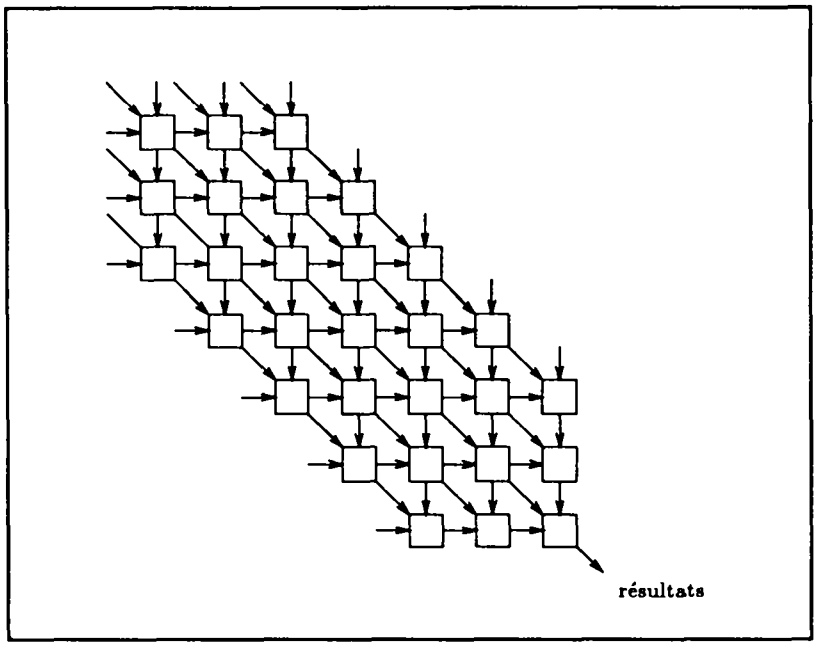


Figure 1.4 : réseau systolique tronqué

de m caractères, il s'agit de n'étudier que les mots dont la longueur est comprise entre $m-l$ et $m+l$. L'entier l est appelé largeur de recherche, et est fixé, dans le cas présent, à 2 caractères.

Cette valeur a été déterminée par expérience et signifie qu'au plus, deux erreurs consécutives de type *insertion* ou *omission* sont prises en compte pendant la correction. La succession de deux erreurs d'insertion, par exemple, *éloigne* le chemin optimum menant au résultat de deux indices par rapport à la diagonale. Dans la réalité, cette double d'erreur apparait de temps à autre, ce qui justifie la présence de 5 diagonales $(2l+1)$ de processeurs dans notre réseau. Les tests effectués avec une largeur de recherche supérieure n'ont pas changé de manière probante le taux de correction; une largeur de recherche de 2 semble même être un optimum.

Cette réduction de la largeur de recherche a deux conséquences importantes. D'une part, pour un mot test donné, le nombre de mots du dictionnaire à parcourir est fortement réduit (de plus de la moitié). D'autre part, le calcul des valeurs $D(i,j)$ est limité aux valeurs autour de la diagonale (5 diagonales pour $l=2$). Ainsi, le réseau de processeurs que nous proposons comporte 69 processeurs $(15+2\times 14+2\times 13)$. Il est représenté par la figure 1.4.

1.3.4 Propagation des résultats

Dans le cas d'un réseau de taille fixe (N lignes et N colonnes), les longueurs n et m sont souvent inférieures à N. Le processeur (m, n) qui termine le calcul de la distance est dans ce cas différent du processeur (N, N). Il est donc nécessaire de propager le résultat du processeur (m, n) au processeur (N, N). Cette opération est effectuée de façon très simple au niveau de l'algorithme. L'idée est de compléter les chaînes T et R par des caractères spéciaux $\perp$ (pour obtenir une chaîne test et un mot de référence de longueur N) et de donner des coûts d'insertion et d'omission particuliers à certains processeurs contenant les caractères $\perp$.

La figure 1.5 montre comment le résulat d'une comparaison d'un mot test de longueur 3 avec une référence de longueur 4 peut se propager sur un réseau de taille 6x6. Les processeurs notés α mémorisent un coût d'omission nul et un coût d'insertion égal à ∞. De plus, le coût de substitution du caractère $\perp$ avec tous les autres caractère de l'alphabet (y compris lui-même) doit être égal à ∞.

De cette manière, les processeurs α rendent en résultat la valeur acquise sur leur port vertical. En effet, ces processeurs effectuent le calcul suivant :

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + d(\perp, t_j) \\ D(i-1,j) + 0 \\ D(i,j-1) + \infty \end{cases} = D(i-1,j)$$

De la même façon, les processeurs notés β mémorisent un cout d'omission égal à ∞ et un coût d'insertion nul pour propager la valeur acquise sur leur port horizontal.

1.4 Intégration

Les machines systoliques sont caractérisées par un réseau de processeurs simples connectés de façon régulière et locale [6]. Il est clair qu'une telle définition se prête bien à une intégration VLSI où le maître mot est *régularité*. Cependant, la régularité des réseaux systoliques peut cacher quelques problèmes non triviaux, notamment des problèmes d'alimentation, d'initialisation ou de récupération de résultats. On pourrait ici faire le parallèle avec la conception de programmes dans laquelle il est simple de trouver les boucles mais plus compliqué de gérer, par exemple, les initialisations et les cas de terminaison.

Par ailleurs, même si la densité d'intégration permet, dans une certaine mesure, de s'affranchir du nombre de transistors, il n'en reste pas moins que dans le réseau de processeurs que nous décrivons un gain d'une centaine de transistors sur chaque tranche 1 bit d'un processeur 8 bits se traduit en dizaines de milliers (60.000) sur l'architecture complète. Il convient donc d'implémenter ce qui est juste nécessaire.

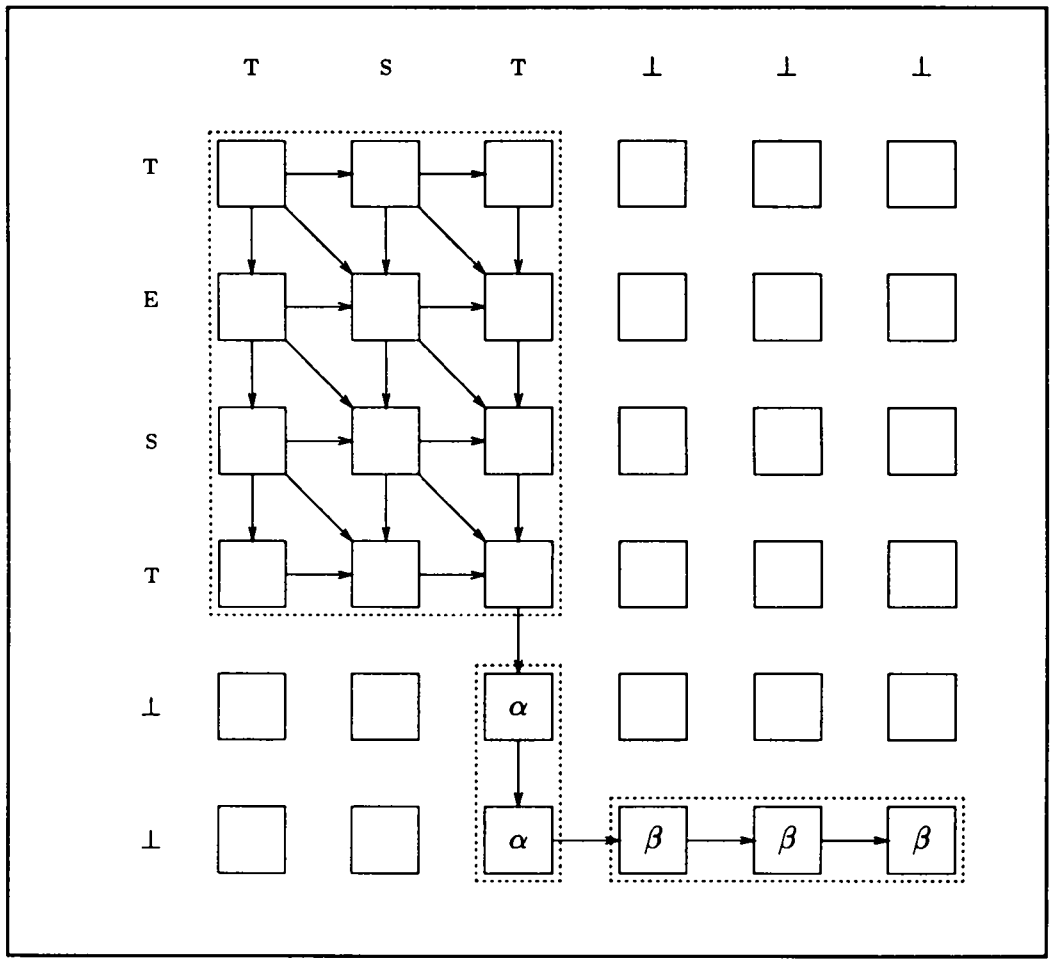


Figure 1.5 : propagation du résultat

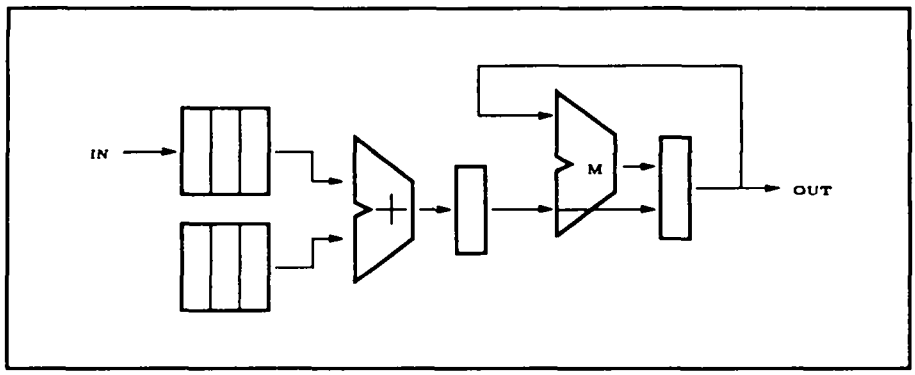


Figure 1.6 : processeur élémentaire

1.4.1 Choix architecturaux

Mémoire

Le calcul de l'équation récurrente de base demande, sur un processeur P(i,j), l'évaluation du coût de substitution des deux caractères r_i et t_j, soit $d(r_i, t_j)$. Ce coût est obtenu par consultation d'une table. Il est clair qu'il est difficilement concevable d'intégrer une mémoire dans chaque processeur permettant de contenir cette table. En effet, si l'alphabet comprend 100 caractères (majuscules, minuscules, signe de ponctuation, chiffre, ...), il faut une mémoire de 10 Koctets soit environ 500.000 transistors pour l'ensemble du réseau ! Une étude plus détaillée de l'algorithme et de l'application a attiré notre intention sur deux points. D'une part, la chaîne test étant constante pendant toute la durée d'un traitement, une seule colonne de la table de substitution est nécessaire par processeur (t_j est constant sur chaque processeur). D'autre part, la table de substitution est relativement creuse, et il serait souhaitable de ne mémoriser que les valeurs significatives. En fait, les valeurs de substitutions intéressantes pour un caractère donné concernent les caractères dont les touches sont proches. Les touches éloignées ont une probabilité de confusion identique.

Ces constatations nous ont amenés a définir une mémoire associative contenant seulement les caractères produisant des valeurs significatives et delivrant un coût par défaut pour les caractères non mémorisés dans la table (caractères éloignés).

Processeur

Le traitement de base affecté à chaque processeur nécessite 6 additions et 3 minimisations, soit 9 opérations élémentaires. Les performances escomptées du circuit sont telles qu'il n'est pas nécessaire de chercher à effectuer ce calcul le plus rapidement possible. Une solution séquentielle est envisageable en déroulant ce calcul sur plusieurs cycles d'horloges.

La figure 1.6 indique le principe de l'architecture d'un processeur élémentaire du réseau 2D. Un processeur est composé de plusieurs registres, d'un additionneur et d'un minimiseur, ce dernier opérateur étant directement connecté à la sortie du premier. En utilisant ce schéma d'interconnexion, le calcul peut être effectué en 6 cycles machines. Nous montrerons au paragraphe 1.6 que ce choix est tout à fait valide puisque les performances obtenues par cette structure correspondent aux contraintes temps réel de l'application de correction de fautes de frappe.

Partage du matériel implanté

Lors de la réalisation d'un circuit spécialisé il suffit d'implanter ce qui est juste nécessaire. Une mémoire par processeur est-elle utile ? En observant le traitement effectué par chaque processeur, il apparaît que l'accès à la table de substitution n'intervient qu'un cycle sur les six du cycle systolique. Les tables d'une même colonne de processeurs étant identiques, il est possible de mettre en œuvre une seule mémoire associative pour une colonne de 5 processeurs. De ce fait, l'architecture globale du réseau est divisée en deux parties fonctionnant en parallèle (cf figure 1.7), l'une pour faire circuler les données, l'autre pour effectuer les calculs. Ces 2 parties sont connectées via les mémoires associatives et sont référencées respectivement par *réseau de données* et *réseau de calcul.*

Le réseau de données est composé d'une matrice de registres contenant les caractères *référence* du dictionnaire et émule le flot de données à travers le réseau systolique. Les registres d'une même colonne sont utilisés séquentiellement pour adresser le bloc de mémoire correspondant et obtenir ainsi le coût de substitution approprié.

Le réseau de calcul contient les processeurs élémentaires qui déterminent chacun une distance partielle en fonction de leur situation dans le réseau. Les coût d'insertion et d'omission sont des constantes disponibles dans des registres internes, tandis que les coûts de substitution sont remis à jour à chaque nouveau cycle systolique. Un cycle systolique dure au moins 6 cycles machine. Pendant cet interval de temps, 5 accès à la mémoire sont réalisés séquentiellement de manière à fournir aux processeurs les valeurs des coûts de substitution pour le cycle systolique suivant.

1.4.2 Organisation de la puce

Le plan de masse du circuit est représenté sur la figure 1.8. On y retrouve les différents éléments décrits précédemment à savoir, le réseau de calcul, la mémoire et le réseau de données. Deux modules supplémentaires ont été introduits, un module de contrôle permettant de gérer le fonctionnement de l'ensemble et un module particulier dont les principales fonctions sont d'initialiser et de tester le circuit. L'ensemble de ces modules sont décrits succintement dans les paragraphes suivants.

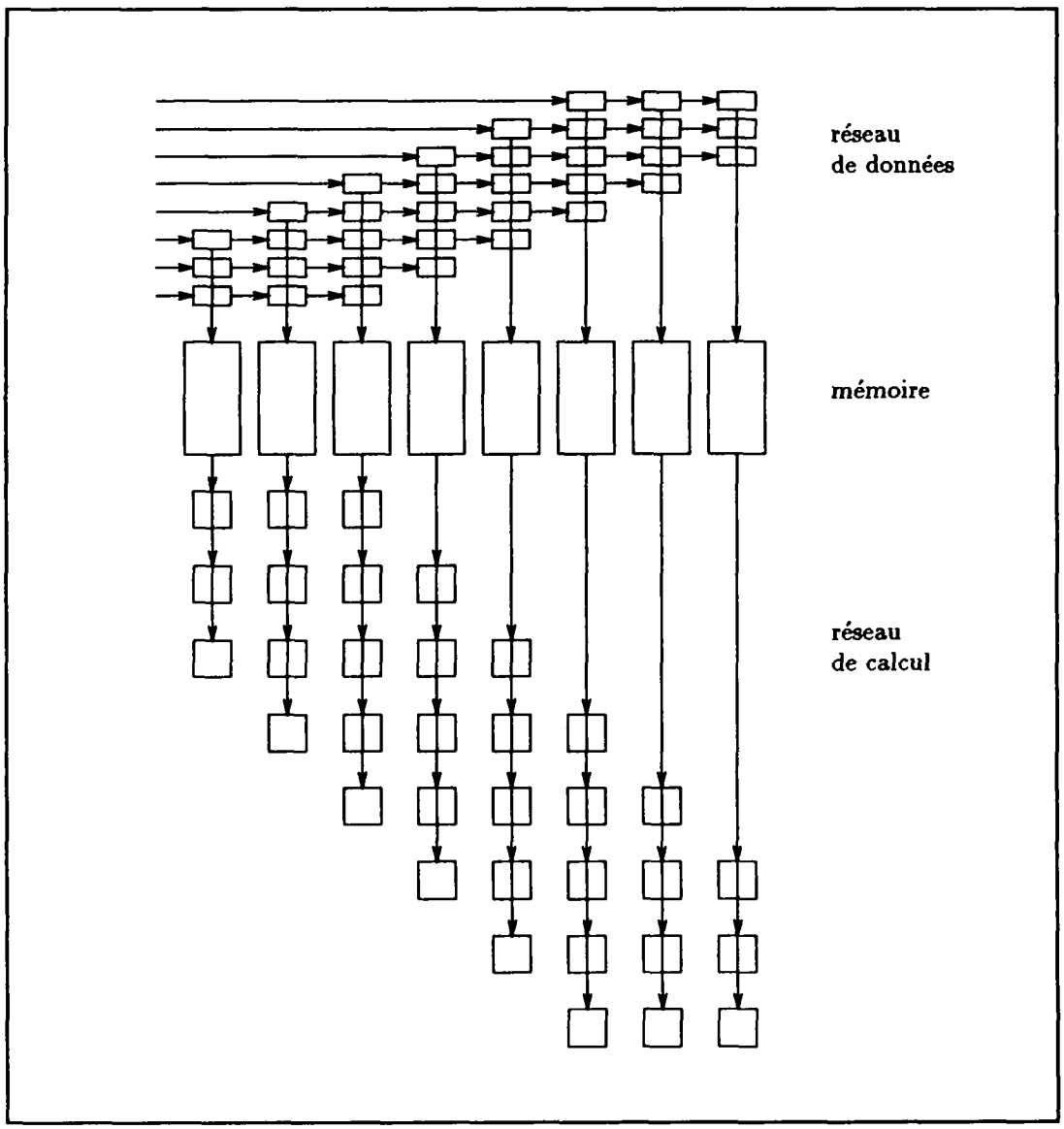


Figure 1.7 : division du réseau en 2 parties

Réseau de calcul

Deux remarques méritent d'être faites concernant le réseau de calcul. D'une part, le réseau a été *rectangularisé* afin d'obtenir une forme plus convenable pour un VLSI. D'autre part, toujours dans un souci de régularité, des processeurs (indiqués en gris sur la figure) ont été ajoutés dans le coin supérieur gauche ainsi que dans le coin inférieur droit. Ils effectuent le même calcul que les autres, mais les résultats obtenus sont inutilisés. Leur rôle essentiel est d'assurer les connexions avec le reste de la puce. Les processeurs du coin supérieur gauche assurent la continuité des connexions entre les mémoires et les processeurs. Les processeurs du coin inférieur droit n'ont aucun rôle de calcul. Ils assurent la continuité des connexions entre les processeurs et les commandes.

Mémoire associative

La mémoire associative est chargée en début de traitement de chaque mot test. Ceci s'effectue grâce à un registre de configuration, registre à décalage de 15 étages (cf figure 1.8). Chaque mémoire associative (une par colonne de processeurs) contient 10 couples (caractère, coût). Le temps de chargement de ces mémoires est négligeable par rapport au temps de traitement d'un mot test.

Réseau de données

Le réseau de données a, lui aussi, subit une *rectangularisation.* De plus, afin de favoriser les entrées de données, le réseau est *renversé* par rapport au réseau de calcul. Cela permet d'avoir les registres d'entrées du réseau le long du bord supérieur du circuit, ce qui facilite son intégration. Un mécanisme d'acquisition des données en provenance de l'extérieur complète ce module.

Registre de configuration

Le registre de configuration permet de charger les mémoires associatives, mais aussi de tester le circuit. En effet, grâce à cet élément, il est possible de charger n'importe quel registre de chaque processeur et lire le registre accumulateur de n'importe quel processeur. De ce fait, des tests systématiques de tous les chemins de données et des divers composants de la puce peuvent être appliqués pour vérifier, module par module, le fonctionnement de l'ensemble.

Ce module à enfin pour tâche de récupérer le résultat des comparaisons produit par le processeur situé sur la dernière colonne de la rangée du milieu et de le propager en sortie.

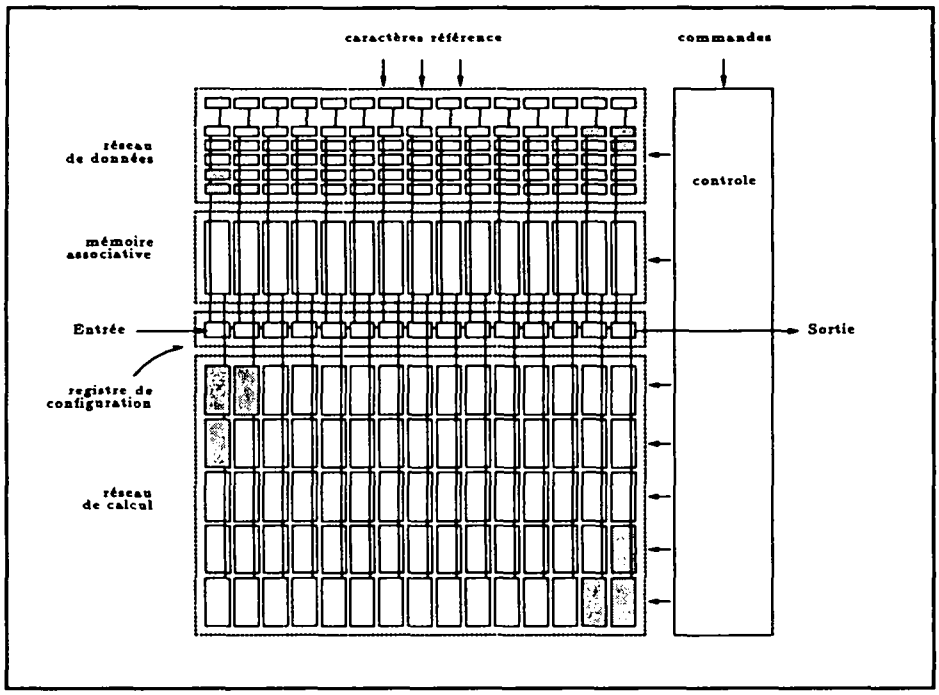


Figure 1.8 : Plan de masse de la puce

Contrôle

L'ensemble des éléments est commandé par un module de contrôle générant toutes les micro-commandes. C'est un simple décodeur qui, en fonction d'une micro-instruction, active et amplifie les signaux de commandes. La micro-instruction est découpée en champs correspondant aux différentes unités de manière à ce qu'elles puissent être activées en parallèle.

Cette structure permet de conserver une certaine *programmabilité* du circuit : en modifiant la suite de micro-instructions soumise au circuit API69 des programmes sensiblement différents peuvent être mis en œuvre. Ainsi, des applications proche de celle pour laquelle le circuit a été étudié peuvent utiliser avantageusement cette puce. Ajoutons enfin que cette *programmabilité* facilite considérablement le test.

Horloge

Le circuit fonctionne sur la base d'une horloge à deux phases non recouvrantes notées respectivement phi1 et phi2.

1.5 Méthode de Conception

Bien qu'ayant un intérêt applicatif indéniable, ce circuit se présente également comme un support particulièrement riche pour élaborer de nouvelles stratégies d'intégration d'architectures régulières. La très haute régularité du circuit autorise une méthode de conception à la *Mead et Conway* [9], c'est à dire, basée sur l'aboutement de cellules. Aucun routage n'est alors nécessaire, d'ou une densité d'intégration particulièrement satisfaisante.

La stratégie développée pendant la conception du circuit API69 consiste à travailler sur une version réduite du circuit : l'idée de base repose sur une construction automatique à partir de paramètres tels que la largeur des chemins de données, le nombre de processeurs, le nombre de mots mémoire, etc.

Grâce à un générateur on réalise automatiquement un *noyau* parfaitement représentatif du circuit final, tant du coté schématique que topologique, et qui sert à la validation et à la mise au point.

Le *noyau* doit, par exemple, posséder toutes les cellules intervenant dans le circuit et, d'un point de vue dessin de masque, reproduire toutes les combinaisons possibles d'aboutement entre cellules. De cette manière, les temps de simulation, de vérification du dessin de masque, de comparaison, etc, sont réduits au minimum et permet, en ne travaillant que sur une base de quelques milliers de transistors, de concevoir des circuits de plusieurs centaines de milliers, voire quelques millions de transistors.

1.6 Performances

Le circuit permet de traiter des chaînes de caractères de longueur inférieure ou égale à 15. Pour des tailles supérieures, un circuit dédié ne se justifie plus par rapport à un traitement algorithmique approprié sur des machines séquentielles conventionnelles. En effet, le nombre de mots supérieurs à 15 caractères étant relativement restreints, le temps de traitement devient suffisamment court (< 0.1s sur de gros vocabulaires) pour ne pas faire appel à un matériel spécialisé.

Le temps de traitement dépend uniquement des tailles des dictionnaires mis en jeu. La taille du mot erroné n'intervient pas puisque le réseau produit un résultat tous les cycles systoliques, quelle que soit la longueur du mot en cours de comparaison. Le temps de traitement peut donc être prédit de manière exacte en fonction du nombre d'éléments du dictionnaire sélectionné.

Le temps de cycle est également un élément déterminant des performances du circuit. La fréquence d'horloge externe calculée est de 25 MHz. Sachant que l'algorithme qui prend en compte les quatre erreurs d'insertion, d'omission, de substitution et de permutation peut être mis en œuvre avec un cycle systolique égal à 6 cycles machine, soit 480ns (un cycle machine équivaut à 2 phases PHI1 / PHI2), le nombre de références traitées en une seconde dépasse les 2 millions. En puissance de crête, (lorsque tous les

processeurs du réseau sont actifs) on atteint alors 1.24 Giga operations/seconde sur cet algorithme particulier.

Cette capacité est largement suffisante pour l'édition de texte car les dictionnaires d'usage courant ont une taille relativement modeste (quelques 10^4 mots, voire 10^5, formes fléchies incluses). Un utilisateur ayant un doute sur l'orthographe d'un mot peut alors se voir proposer quelques corrections en moins d'un dixième de seconde! Par contre, lorsqu'il s'agit d'une correction temps réel sur de gros volumes de données cette puissance de calcul se révèle indispensable.

Dans cette optique, une étude concernant la correction des adresses postales après lecture optique à montré que le même algorithme, réduit à la correction des erreurs d'insertion, d'omission et de substitution, se révèle également efficace. Son implémentation sur le réseau systolique est directe et requiert un cycle systolique de 4 cycles machine. Le nombre de références traitées en une seconde dépasse, dans ce cas, les 3 millions.

Une version séquentielle de l'algorithme a été développée et testée sur une station de travail SUN-4c/60. Elle apporte des éléments de comparaison intéressants par rapport à la mise en œuvre systolique. Précisons que cette version intègre un certain nombre d'optimisations non possibles sur une structure parallèle telles que l'arrêt des calculs si la distance dépasse un certain seuil ou le parcours limité du dictionnaire en fonction des résultats déjà produits. Sur un vocabulaire de 135 000 mots, le temps moyen de correction d'un mot de 10 lettres est de l'ordre de 2 secondes. C'est pratiquement le cas le plus défavorable car le nombre de références du dictionnaire intervenant dans le processus de comparaison est le plus élevé. De plus, ce temps de correction n'est pas constant et dépend énormément des données (dictionnaire et mot en cours de comparaison). Il peut varier de moins d'une seconde à 4 ou 5 secondes.

Dans le pire des cas, la correction d'un mot de 10 lettres avec notre circuit nécessite la comparaison de 58 000 références sur les 135 000 appartenant au dictionnaire (références ayant entre 8 et 12 caractères), soit un temps de réponse de 28 ms. Le gain obtenu par rapport à une station de travail récente est donc de l'ordre de 70. Mais il est clair que ce circuit s'adresse également aux parcs des PC où les logiciels de traitement de texte sont très utilisés. La méthode de correction logicielle ne peut être mise en œuvre en temps réel à cause du temps de réponse (entre 20 et 40 secondes pour une correction!). L'utilisation d'un circuit spécialisé est alors pleinement justifié.

Chapitre 2

Le réseau de Processeurs

2.1 Introduction

Le réseau de processeurs du circuit API69 est une matrice de 15 × 15 cellules où ne sont conservées que 5 diagonales, soit 69 processeurs. Les processeurs sont connectés horizontalement et verticalement entre eux (cf figure 2.1). L'émulation d'une connexion diagonale, indispensable pour mettre en œuvre l'algorithme de programmation dynamique, est réalisée par un transfert vertical suivi d'un transfert horizontal. Ceci implique l'ajout d'une sixième diagonale de *processeurs* pour assurer cette liaison. En fait, cette diagonale (en pointillé sur la figure) doit simplement posséder des cellules réduites, au niveau matériel, à un dispositif minimum assurant cette fonction.

L'alimentation en données est faite par 15 bus de diffusion (noté DBUS), un pour chaque colonne de processeurs. Ces bus permettent d'acheminer des valeurs d'initialisation ou des données issues de la mémoire. Tous les processeurs d'une colonne i ont accès au i^{eme} bus DBUS.

Pour des raisons d'intégration VLSI évidentes, la topologie du réseau représentée sur la figure 2.1 ne peut pas être implémentée telle quelle sur silicium. Elle nécessite un réarrangement plus *régulier*, par *rectangularisation.* En plaçant horizontalement les processeurs situés sur les diagonales on obtient une matrice de 5 × 15 processeurs comme le montre la figure 2.2. Les connexions verticales restent identiques tandis que les connexions qui étaient auparavant horizontales deviennent diagonales.

Toujours dans un souci de régularité, le réseau initial est complété, dans les *coins*, par des processeurs représentés en grisé sur la figure 2.2. Ils ne jouent aucun rôle mais leur présence physique permet de préserver la régularité et d'assurer la propagation des signaux de commandes et des alimentations. Afin de ne pas perturber le fonctionnement de l'ensemble, un mécanisme doit permettre leur inhibition (cf paragraphe 2.4.2).

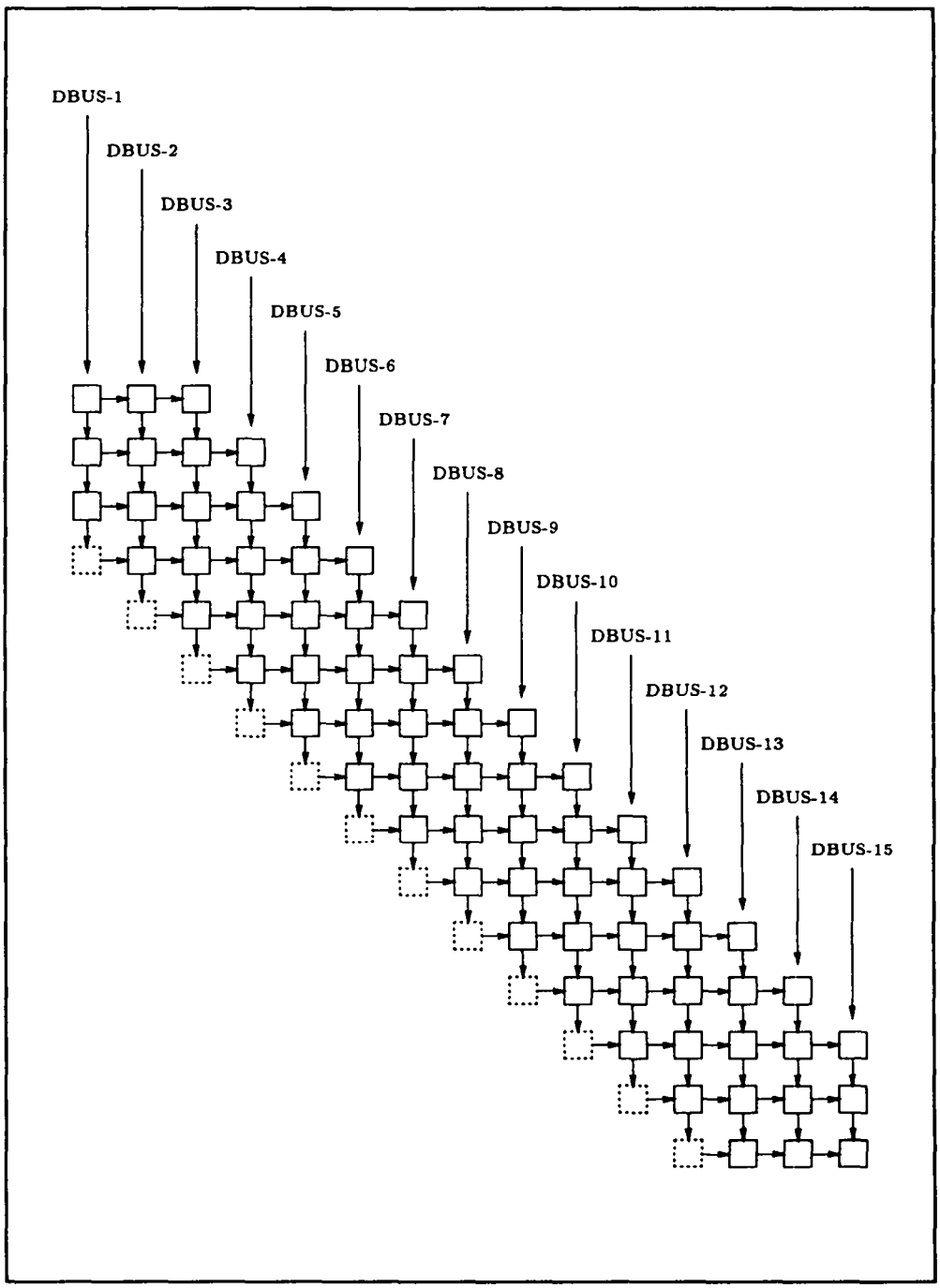


Figure 2.1 : réseau API69

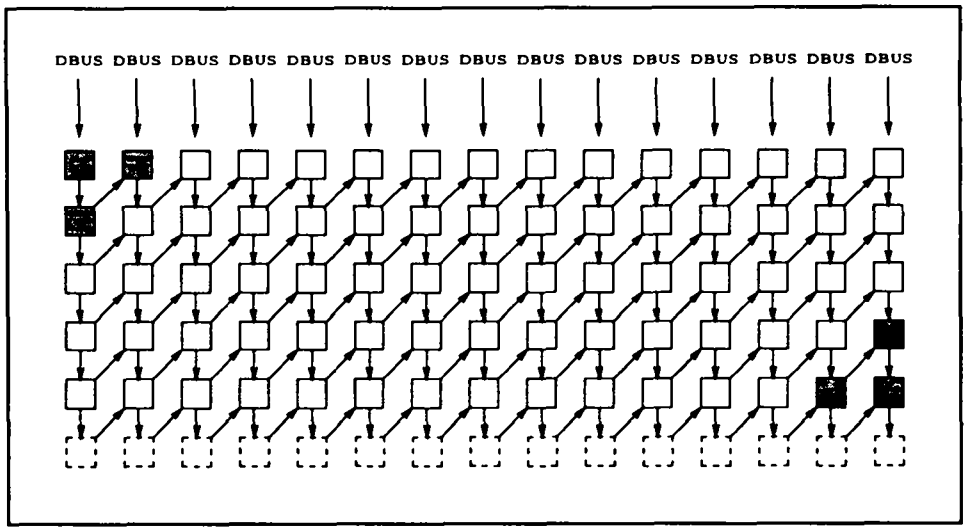


Figure 2.2 : matrice régulière de processeurs

2.2 Description fonctionnelle

Le réseau est composé de 3 types de macro-cellules (cf figure 2.3) :

- les cellules **processeur** :, éléments actifs du réseau (notées **P** sur la figure),
- les cellules **aiguillage** : qui assurent soit une liaison verticale, soit une liaison diagonale entre les processeurs (notées **A** sur la figure),
- les cellules **connexion** : qui permettent l'émulation de la liaison logique de la diagonale inférieure (notées **C** sur la figure).

Les entrées des cellules d'aiguillage de la première rangée sont reliées au bus de diffusion. Le bus InP des processeurs situés sur la première ligne peut donc être connecté au bus DBUS, ce qui permet d'initialiser les registres d'entrées/sorties de ces processeurs (cf paragraphe 2.5).

2.2.1 Cellule processeur

D'un point de vue externe, un processeur se caractérise par un bus d'entrée (InP), un bus de sortie (OutP), un bus de diffusion (DBUS) et 12 signaux de commande. Le bus d'entrée InP permet d'acquérir les résultats des processeurs voisins. Le bus OutP délivre le résultat du calcul aux processeurs voisins. Le bus DBUS permet d'acheminer vers les processeurs des données externes telles que les valeurs d'initialisation ou les coefficients de substitution stockés en mémoire.

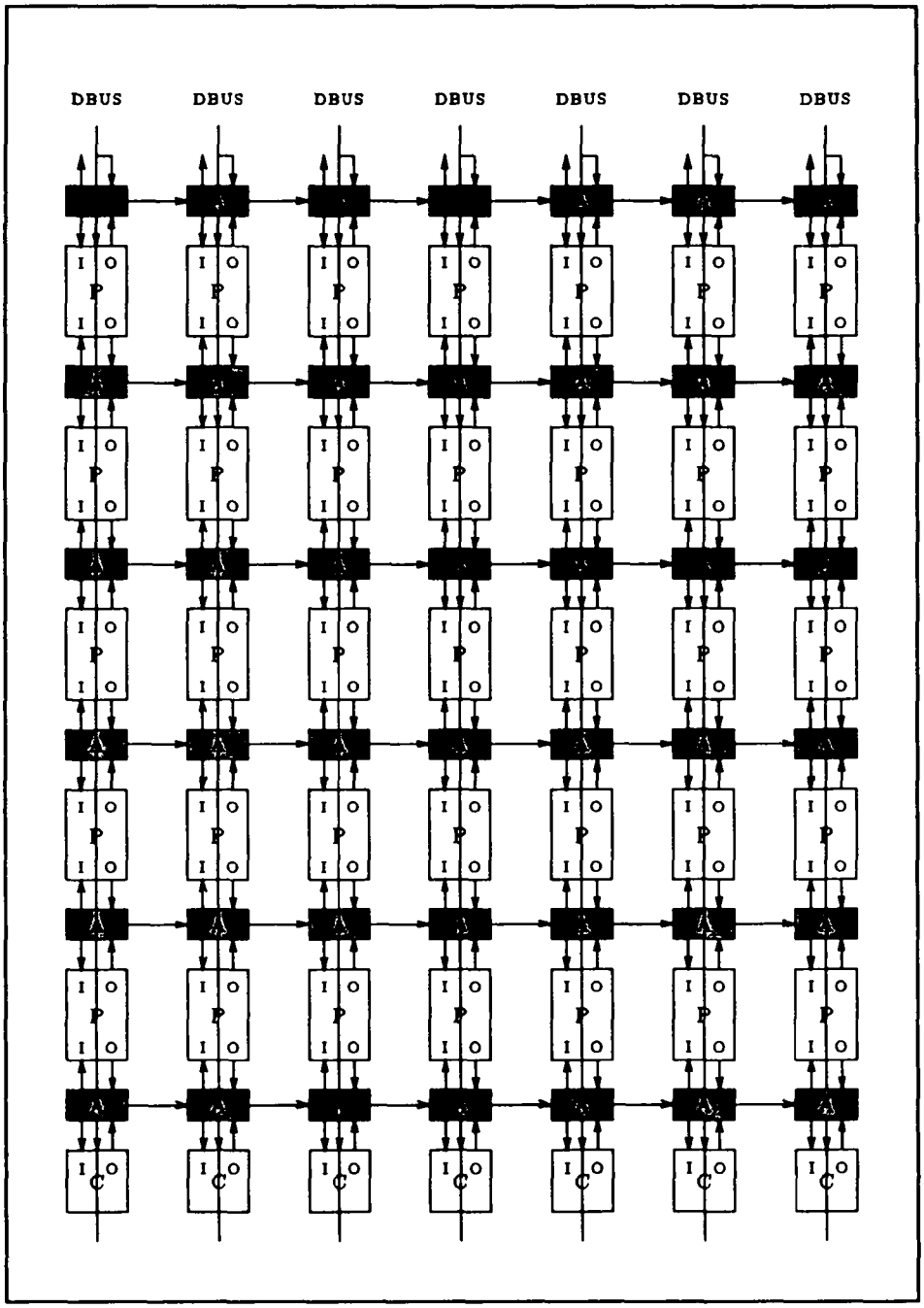
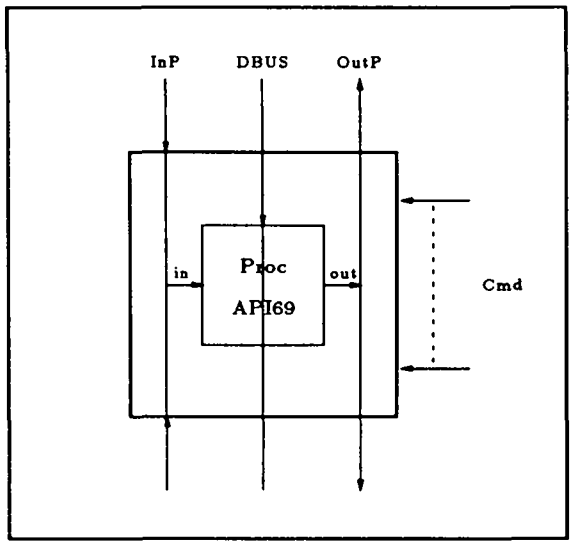


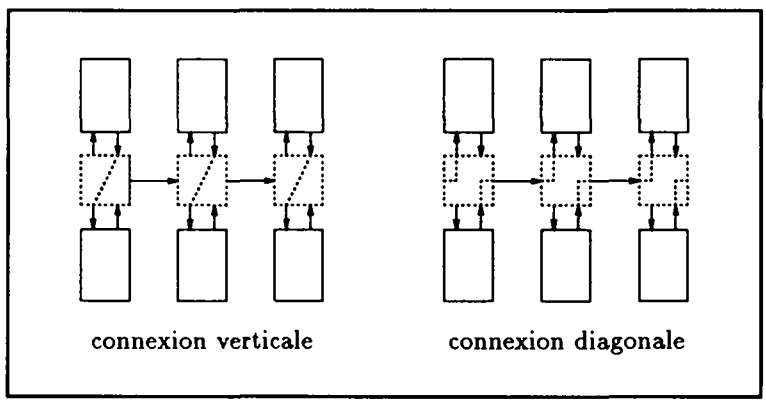
Figure 2.3 : connexion des processeurs



Les commandes sont décrites en détail au paragraphe 2.4.6. Elles indiquent les actions que doit effectuer un processeur.

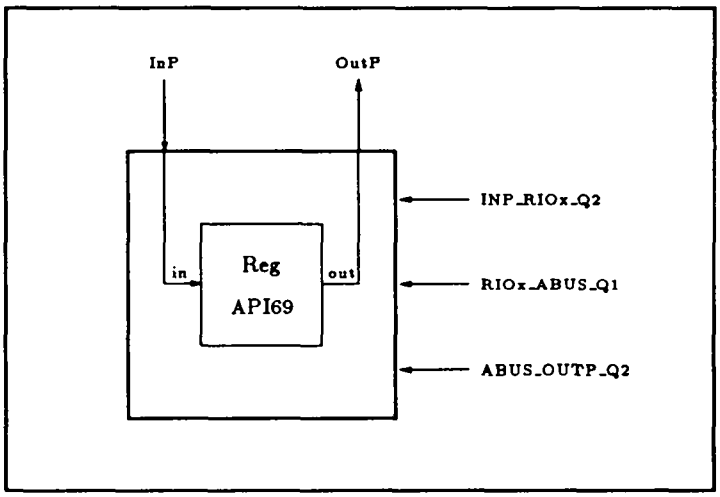
2.2.2 Cellule aiguillage

L'interconnexion des processeurs se fait à l'aide d'un dispositif permettant deux types de connexion : une connexion verticale et une connexion diagonale. Ce dispositif est commandé respectivement par les signaux LINK_UPDW_Q2 et LINK_DIAG_Q2. Précisons que lorsqu'on évoque le terme *connexion diagonale* il s'agit de la connexion physique, après *rectangularisation* de la matrice originale.



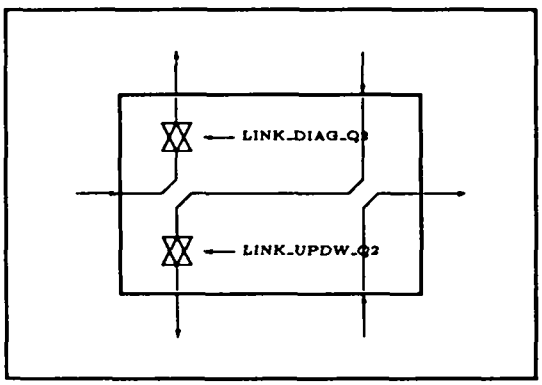
2.2.3 Cellule connexion

Ce module est un sous ensemble d'un processeur. Seules les parties matérielles réalisant la mémorisation temporaire des données échangées entre processeurs sont conservées. Les commandes qui contrôlent ce module sont donc également un sous ensemble des commandes du processeur.



2.3 Architecture d'une cellule d'aiguillage

Une cellule d'aiguillage permet de réaliser 2 liaisons différentes entre les processeurs, une liaison verticale et une liaison diagonale. La figure ci-dessous donne le principe de fonctionnement de ce dispositif purement combinatoire.



2.4 Architecture d'un processeur

2.4.1 Description globale

La figure 2.4 donne le schéma d'un processeur du circuit API69. Il comprend :

- **un bloc de registres d'entrées/sorties** (RIO)
  les données proviennent du bus d'entrée InP et sont émises sur le bus ABUS. Le nombre de registre est de 8.

- **un registre de configuration des E/S** (Rmsk)
  ce registre permet d'inhiber l'écriture des registres d'entrées/sorties. On peut ainsi gérer de manière relativement souple les problèmes d'acquisition de valeurs en bordure du réseau.

- **un bloc de registres de constantes** (RCTE)
  les données proviennent du bus de diffusion DBUS et peuvent être émises sur le bus BBUS. C'est dans ce bloc de registres que sont stockées les constantes d'insertion, d'omission, etc. Le coefficient de substitution est également mémorisé dans ce bloc de registres car la mémoire permet d'acheminer le résultat d'une lecture via le bus DBUS. Il y a 8 registres de constante.

- **un additionneur**
  il prend ses opérandes sur les bus ABUS et BBUS, et envoie le résultat de l'addition dans le registre RADD.

- **un minimiseur**
  ses opérandes proviennent des registres RADD et ACC. Le résultat peut être stocké soit dans le registre ACC, soit dans le registre RD.

- **3 registres de travail**

  - le registre RADD
    à chaque cycle d'horloge, il mémorise le résultat de l'additionneur.
  - le registre ACC
    il peut être chargé soit par le contenu du registre RADD, soit par le résultat de l'opération de minimisation.
  - le registre RD
    il contient le résultat du calcul.

  Ces registres chargent sur PHI1 une valeur élaborée sur PHI2.

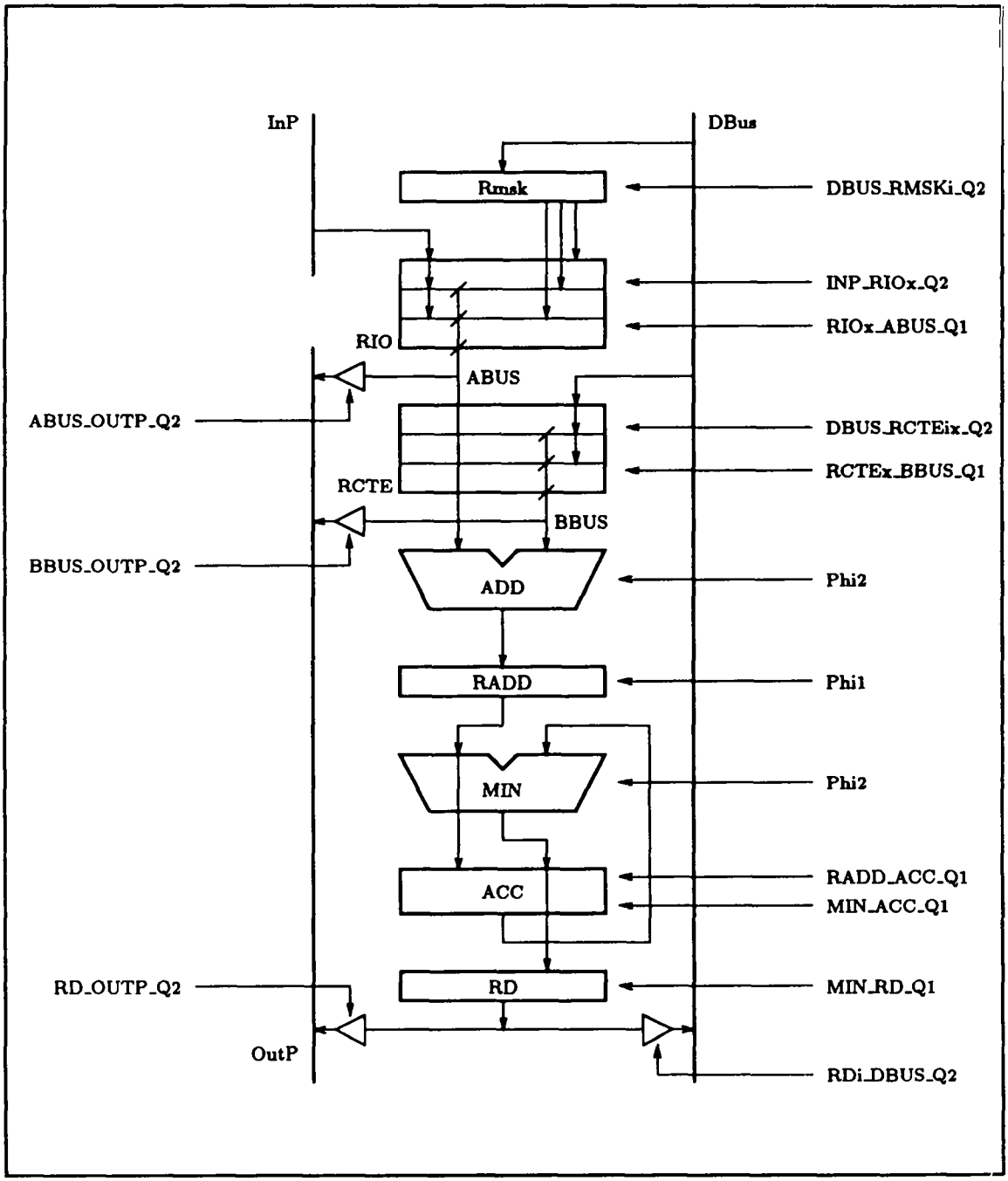


Figure 2.4 : architecture d'un processeur API69

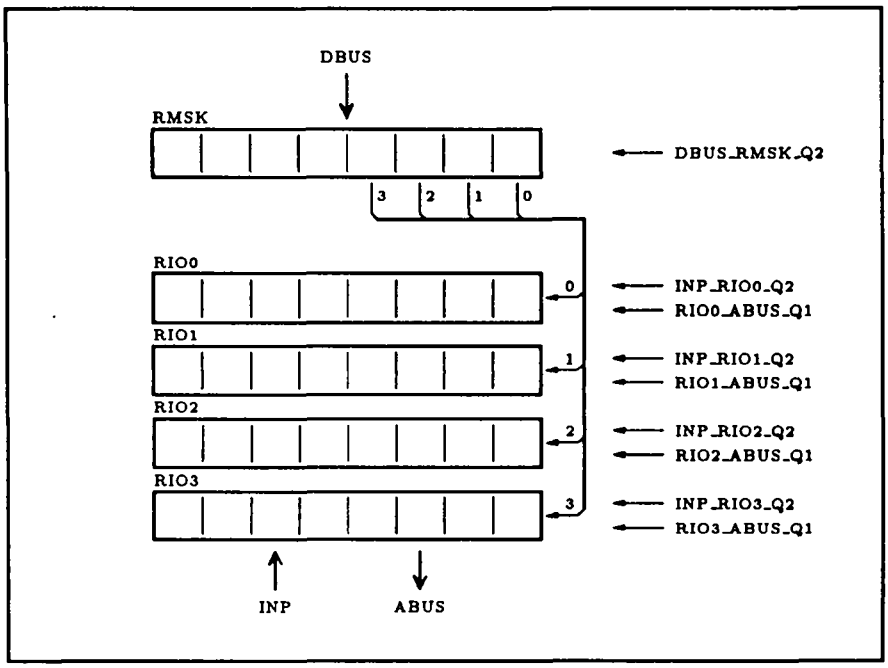


Figure 2.5 : dispositif d'entrés/sorties

Le bus OutP peut transférer des données provenant des registres d'entrées/sorties et des registres de constantes. Cette fonction est utile lors des phases d'initialisation (les registres d'entrées/sorties sont chargés en pipeline par une connexion verticale entre processeurs).

Le bus de diffusion (DBUS) peut transmettre le contenu du registre RD, c'est à dire le résultat d'un calcul.

Les principaux sous-ensembles du processeur sont décrits plus en détail dans les paragraphes suivantes.

2.4.2 Dispositif d'entrées/sorties

Le dispositif d'entrées/sorties se compose d'un bloc de registres banalisés (RIOx) et d'un registre de masquage (Rmsk).

Les registres d'entrées/sorties (registres RIOx) enregistrent (pendant la phase PHI2) le contenu du bus InP lorsque la commande INP_RIOx_Q2est active. Cette commande peut être localement inhibée par le bit x du registre de masquage (inhibée s'il vaut 0).

Ce dispositif permet de gérer l'alimentation en données des processeurs situés en bordure du réseau : dans le cas de la programmation dynamique, par exemple, les processeurs périphériques utilisent des valeurs constantes alors que les autres processeurs

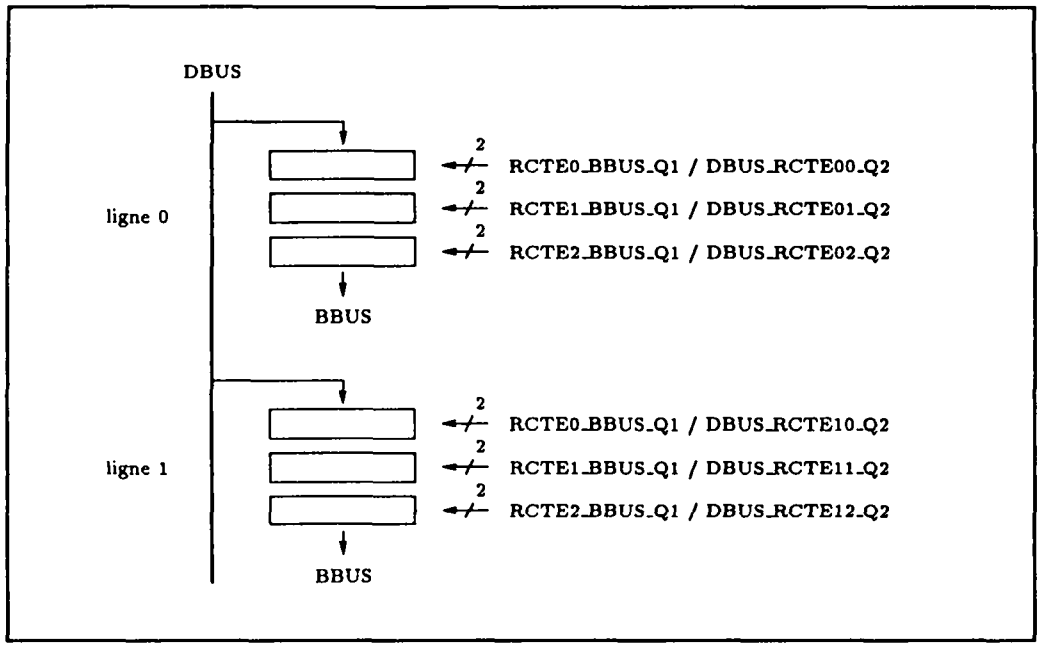


Figure 2.6 : registres de constantes

du réseau acquièrent des données calculées par leurs voisins. Ces valeurs sont acquises via les registres d'entrées/sorties. D'autre part, comme les processeurs en bordure du réseau peuvent n'être connectés à aucune source émettrice significative, la commande INP_RIOx_Q2 aurait pour effet de mémoriser une valeur non désirée dans ces registres.

Pour initialiser les processeurs périphériques, on valide toutes les commandes et on initialise les registres d'entrées/sorties par décalage vertical (la i^{eme} ligne peut être connectée au DBUS, les autre communiquent par ABUS/INP). Il suffit ensuite de reconfigurer correctement les registres de masquages (DBUS_RMSKi_Q2, i = no de ligne).

La commande RIOx_ABUS_Q1, active sur phi1, émet le contenu du registre RIOx sur le bus ABUS.

2.4.3 Registres de constantes

Les registres de constantes sont chargés à partir du bus de diffusion DBUS par la commande DBUS_RCTEix_Q2 (i = no de ligne, x = no de registre). Leur contenu peut être émis sur le bus BBUS sous l'action de la commande RCTEx_BBUS_Q1.

Chaque processeur pouvant mémoriser des valeurs différentes dans ses registres de constantes, la commande DBUS_RCTEix_Q2 ne peut être activée simultanément pour l'ensemble des processeurs du réseau. Cela signifierait que tous les processeurs d'une même colonne chargent, par exemple, le registre RCTE0 avec la même valeur présente sur le bus de diffusion. Cette commande peut donc être activée seulement sur une ligne

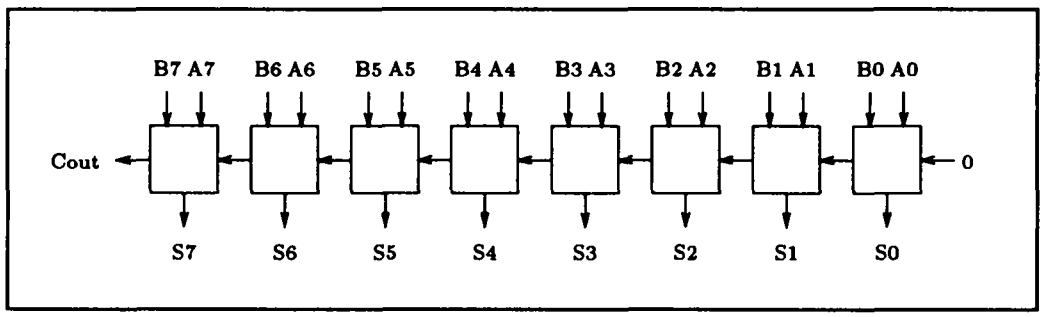
de processeurs (cf figure 2.6).

Dans l'application de fautes de frappe, on mettra, par exemple, ki dans RCTE2, ko dans RCTE3 et le coût de substitution dans RCTE1.

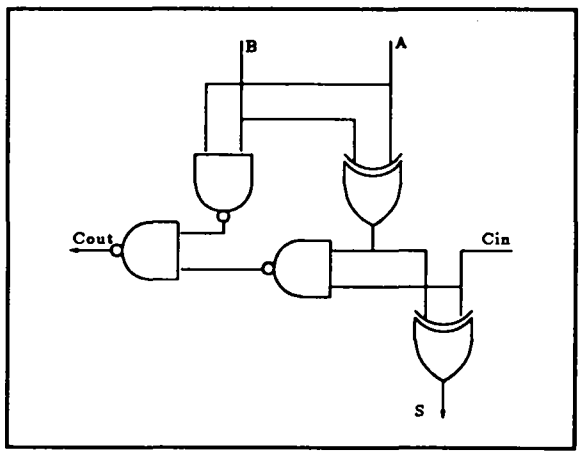
Le coefficient de substitution représentant la *distance* entre deux caractères, et requis à chaque cycle systolique, transite par les registres constante. En effet, la mémoire émet sur ce bus lorsqu'elle est en mode lecture, ce qui permet, en parallèle avec le calcul en cours, de configurer convenablement un des registres de constantes.

2.4.4 Additionneur

L'additionneur est composé de 8 cellules additionneur 1 bit connectées en cascade : la retenue sortante de la cellule i est connectée à la cellule entrante de la cellule $i+1$:



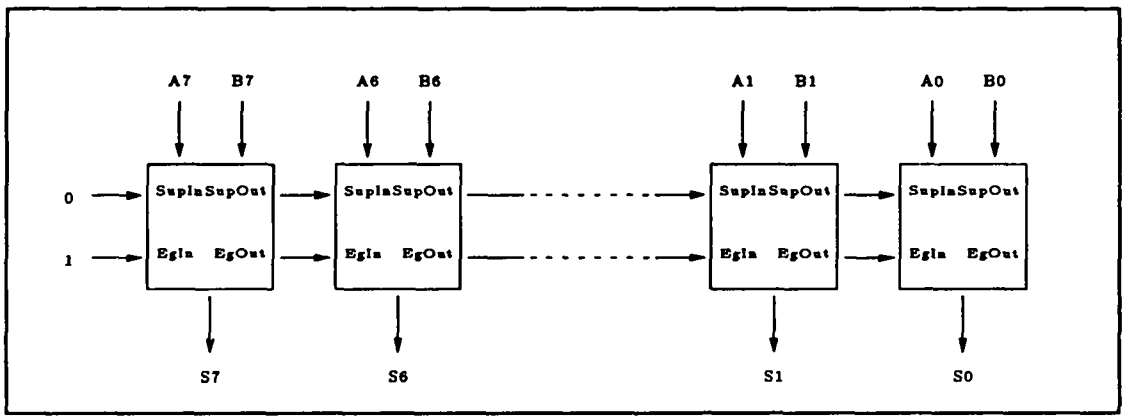
Chaque cellule possède 3 entrées (A, B et Cin) et 2 sorties (S et Cout). La figure ci-dessous donne le schéma interne d'une cellule.



L'additionneur est purement combinatoire et ne nécessite pas de commandes particulières. Si δ est le temps de traversée d'une porte NAND et 3δ le temps de traversée d'une porte XOR, le délai d'une addition sur 8 bits est de $(3+7\times 2+3)\delta = 20\delta$. Le résultat s'élabore pendant la phase PHI2.

2.4.5 Minimiseur

Le principe de fonctionnement du minimiseur est également basé sur un découpage modulaire en cellules de 1 bit. La figure ci-dessous représente la connexion de 8 cellules identiques.



Au niveau de chaque cellule, les entrées SupIn et EgIn représentent l'état de la comparaison des bits de poids supérieurs. Pour la cellule i, par exemple, les états suivants signifient :

SupIn	EgIn	signification
0	0	$A_7..A_{i+1} < B_7..B_{i+1}$
0	1	$A_7..A_{i+1} = B_7..B_{i+1}$
1	0	$A_7..A_{i+1} > B_7..B_{i+1}$
1	1	impossible

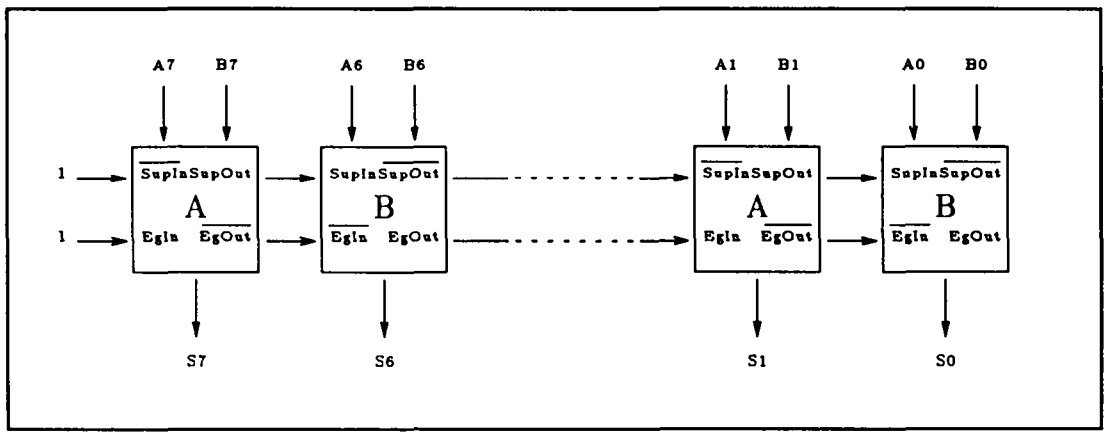
En fonctions de ces signaux et des entrées A et B, chaque cellule réalise les fonctions suivantes :

$$\begin{aligned} \text{SupOut} &= \text{SupIn} + \text{A}.\overline{\text{B}}.\mathit{EgIn} \\ \text{EgOut} &= \text{EgIn} \,.\, \text{A} \oplus \text{B} \\ \text{S} &= \text{A}.\overline{\text{SupOut}} + \text{B}.\text{SupOut} \end{aligned}$$

La réalisation pratique est un peu différente. Elle cherche à minimiser le nombre de portes logiques nécessaires pour cabler ces fonctions.

Deux types de cellules sont utilisées, des cellules de type A et des cellules de type B. Extérieurement, elles diffèrent par la signification des niveaux logiques, en entrée comme en sortie. En interne, leurs fonctions sont différentes et ne font intervenir, au

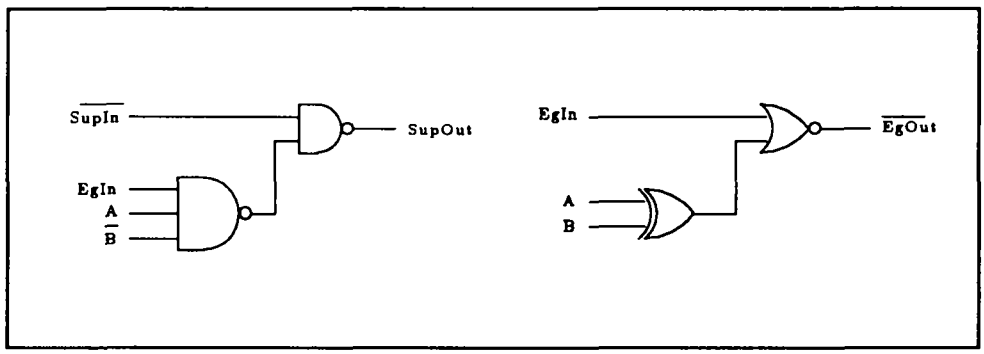
niveau de la propagation des signaux Sup et Eg, que des portes logiques NAND et NOR. D'un point de vue VLSI de telles portes sont plus faciles à intégrer que des portes logiques AND ou OR: elles prennent moins de place et sont plus rapides. La figure suivante indique le schéma retenu:



cellule de type A

$$\overline{\text{SupOut}} = \overline{\text{SupIn} + \text{EgIn}.\text{A}.\overline{\text{B}}} = \overline{\overline{\text{SupIn}}.\overline{\text{EgIn}}.\text{A}.\overline{\overline{\text{B}}}}$$

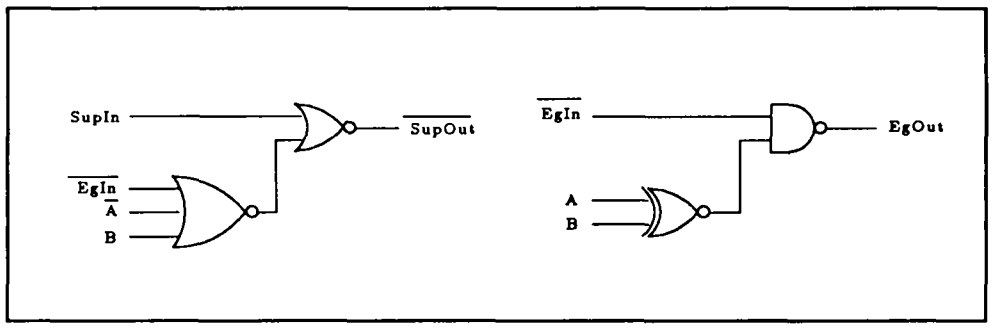
$$\overline{\text{EgOut}} = \overline{\text{EgIn} + (\text{A} \oplus \text{B})}$$



cellule de type B

$$\overline{\text{SupOut}} = \overline{\text{SupIn} + \text{EgIn}.\text{A}.\overline{\text{B}}} = \overline{\text{SupIn} + \overline{\overline{\text{EgIn}} + \overline{\text{A}} + \text{B}}}$$

$$EgOut = EgIn + A \oplus B = \overline{\overline{EgIn} \cdot \overline{A \oplus B}}$$



2.4.6 Commandes

Cette section récapitule les commandes utilisées par un processeur. Les commandes suffixées par "_Q1" sont actives pendant la phase phi1. De même, les commandes suffixées par "_Q2" le sont pendant la phase phi2. En règle général, une commande décrit un transfert d'une source émettrice vers un récepteur. La commande :

Src_Dst_Qx

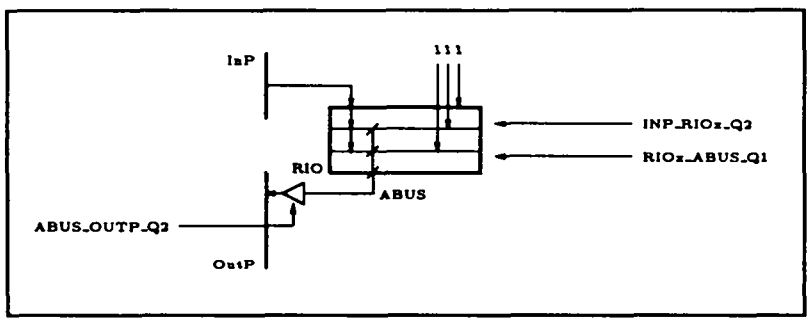
désigne un transfert de *Src* vers *Dst* pendant la phase x.

- INP_RIOx_Q2 : chargement du registre d'entrées/sorties x avec le contenu du bus InP.
- DBUS_RCTEix_Q2 : chargement du registre constante x avec le contenu du bus DBUS sur la ligne de processeur i.
- RIOx_ABUS_Q1 : chargement du bus ABUS avec le contenu du registre d'entrées/sorties RIOx (x=1,2,...).
- RCTEx_BBUS_Q1 : chargement du bus BBUS avec le contenu du registre constante RCTEx (x=1,2,...).
- RADD_ACC_Q1 : chargement du registre ACC avec le contenu du registre RADD.
- MIN_ACC_Q1 : chargement du registre ACC avec le résultat de l'opérateur MIN.
- MIN_RD_Q1 : chargement du registre RD avec le résultat de l'opérateur MIN.
- ABUS_OUTP_Q2 : chargement du bus OutP avec le contenu du bus ABUS.

- BBUS_OUTP_Q2 : chargement du bus OutP avec le contenu du bus BBUS.
- RDi_DBUS_Q2 : chargement du bus DBUS avec le contenu du registre RD de la ligne de processeur *i*.
- RD_OUTP_Q2 : chargement du bus OutP avec le contenu du registre RD.
- DBUS_RMSKi_Q2 : chargement du registre RMSK avec le contenu du bus DBUS sur la ligne de processeurs *i*.

2.5 Architecture d'une cellule connexion

Ce module assure l'émulation d'un transfert de données sur la diagonale inférieure du réseau. Cette fonction étant également assurée par les processeurs, les même éléments matériels ont été repris.



2.6 Mise en œuvre VLSI

2.6.1 Plan de masse d'un processeur

La figure 2.7 donne le plan de masse d'un processeur (exemple d'un processeur 4 bits) du circuit API69. Il est constitué de tranches de 1 bit aboutées horizontalement, chaque tranche étant composée de cellules de nature différente et traversée verticalement par 3 bus, le bus de diffusion (DBUS), le bus d'entrée (InP) et le bus de sortie (OutP). Les bus internes (ABUS, BBUS, ...) ne sont pas représentés sur la figure.

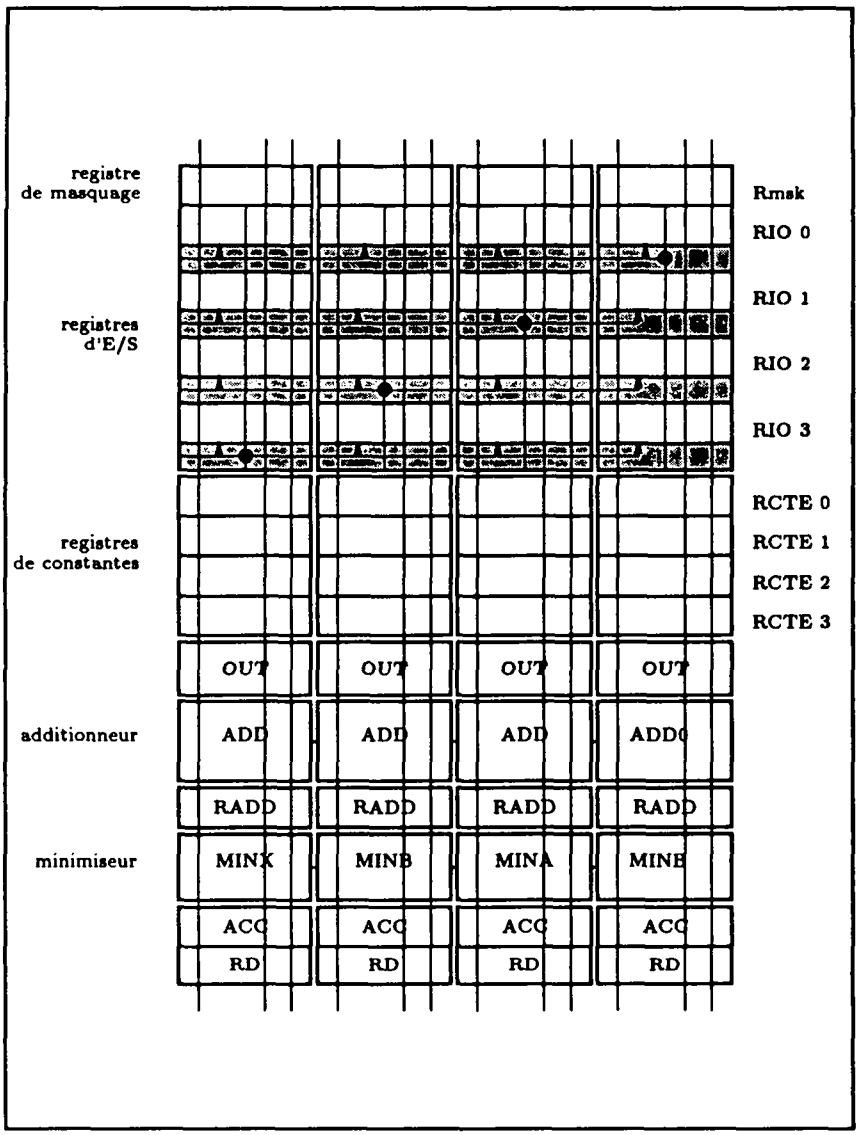


Figure 2.7 : plan de masse d'un processeur

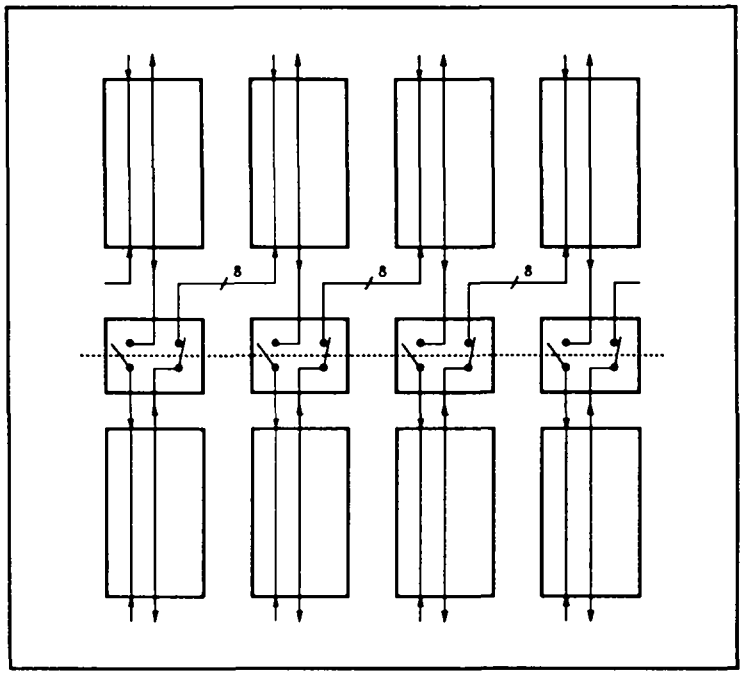


Figure 2.8 : principe de connexion entre processeurs

La connexion du registre de masquage aux registres d'entrées/sorties appelle quelques commentaires : plusieurs cellules de routage sont nécessaires pour assurer cette connexion *presque* régulière (cf figure 2.5). Les bits du registre de masquage traversent systématiquement l'ensemble des registres d'entrées/sorties. Deux types de cellule de routage permettent soit d'effectuer une connexion entre un bit du registre de masquage et l'entrée *Msk* d'un registre d'entrées/sorties, soit de propager le bit sans connexion. En fait, les cellules de routage situées à droite doivent être légèrement différentes de manière à ne pas connecter les commandes de deux processeurs adjacents.

Pour la même raison, la cellule de poids faible de l'additionneur (ADD0) est également différente des autres cellules qui constituent cet opérateur. On doit, de plus, positionner la retenue entrante à zéro.

Le minimiseur est composé de 2 types de cellules différentes comme expliqué au paragraphe 2.4.5. La cellule de gauche (MINX) est un peu différente puisqu'elle initialise la chaîne de propagation du minimiseur.

La cellule référencée **OUT** sur le schéma de la figure 2.7 a pour rôle d'émettre sur le bus OutP le contenu des bus ABUS ou BBUS en fonction des commandes ABUS_OUTP_Q2 et BBUS_OUTP_Q2.

Un processeur est ainsi formé de 8 tranches de 1 bit, légèrement différentes les unes des autres suivant leur position, mais respectant toutes un certain nombre de contraintes pour permettre une propagation horizontale des alimentations et des signaux de commande.

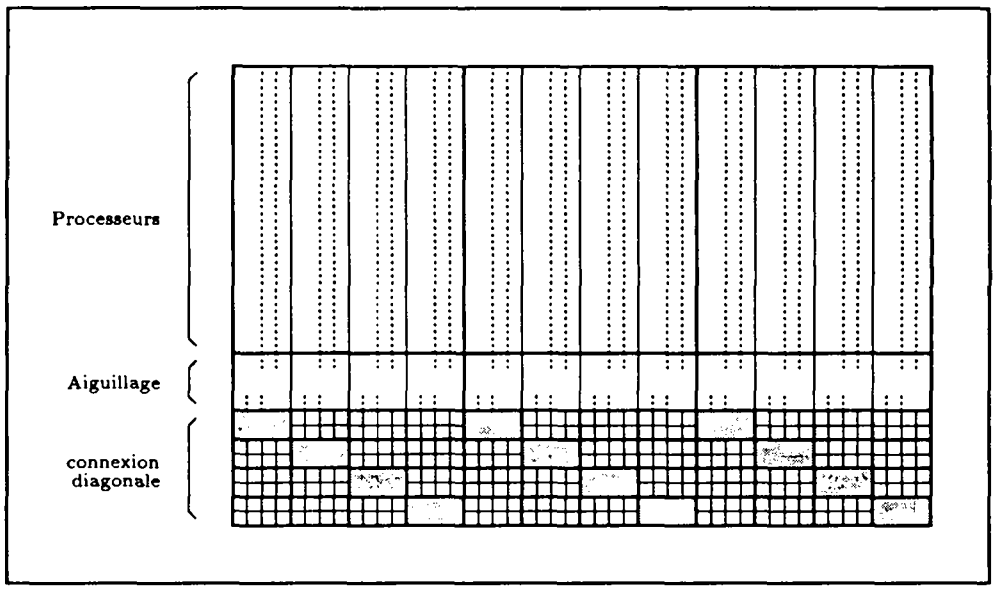


Figure 2.9 : connexion entre processeurs

2.6.2 Connexion entre processeurs

La figure 2.8 montre les connexions à réaliser entre processeurs. Deux connexions doivent pouvoir être effectuées, une connexion verticale (du haut vers le bas) et une connexion diagonale (du bas-gauche vers le haut-droite). La cellule d'aiguillage permet d'avoir l'une ou l'autre connexion; elle est en aboutement sur le bord inférieur des processeurs.

La connexion diagonale demande cependant un routage particulier pour acheminer la sortie d'un processeur vers son voisin Nord-Est. Deux types de cellule de routage sont alors nécessaires comme l'indique la figure 2.9.

2.6.3 Plan de masse du réseau

Le réseau de processeurs est constitué de 15 colonnes identiques de 5 processeurs (8 tranches de 1 bit), toutes les colonnes étant en aboutement direct les unes avec les autres.

La figure 2.10 représente le plan de masse. On y retrouve les 3 types de macro-cellules mentionnés précédemment : processeurs (rectangles blanc), aiguillages (carrés gris) et connexions (dernière ligne, cf section 2.1). Les parties hachurées représentent les zones de routage.

Chaque macro-cellule est traversée horizontalement par ses signaux de commandes et ses fils d'alimentation. Un signal de commande est ainsi partagé par toutes les cellules situées sur le même axe horizontal.

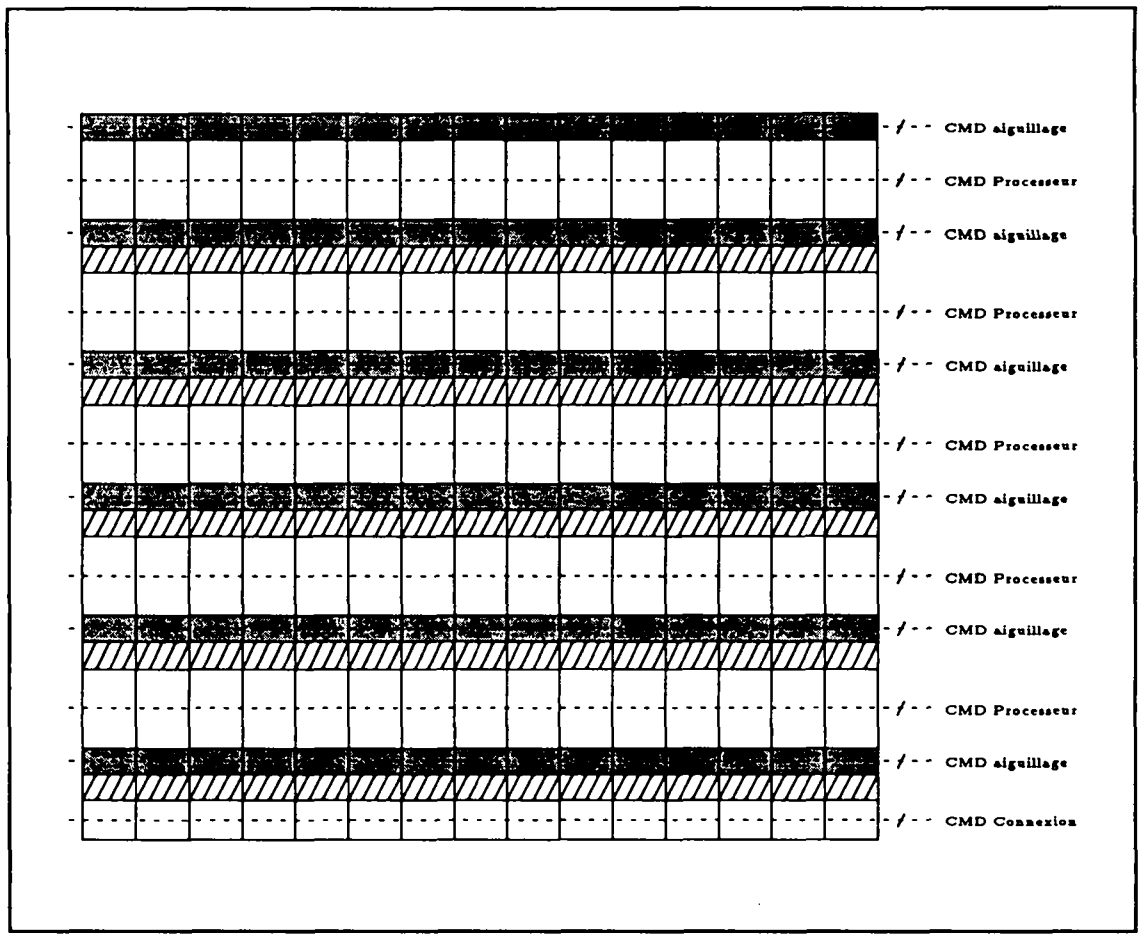


Figure 2.10 : plan de masse du réseau API69

2.6.4 Génération du dessin de masque

Le réseau de processeurs est donc une structure extrêmement régulière (cf figure 2.10) puisqu'il s'agit, en fait, d'un tableau de cellules interconnectées directement les unes avec les autres. Aucun routage supplémentaire n'est requis.

Sa construction peut être facilement automatisée avec pour paramètres :

- le nombre de colonnes de processeurs,
- le nombre de lignes (ou nombre de diagonales) de processeurs,
- la largeur du chemin de données,
- le nombre de registres d'E/S par processeur,
- le nombre de registres de constantes par processeur.

La plupart des outils de CAO VLSI offrent la possibilité de générer des tableaux de cellules en prenant comme paramètres d'entrée un fichier où sont décrit les dispositions des cellules les unes par rapport aux autres. Le logiciel de CADENCE intègre un tel utilitaire référencé *Make Array*.

Un générateur du tableau de processeurs consiste donc en un programme qui prend comme paramètres d'entrée les 5 paramètres énumérés précédemment et rend en résultat un fichier décrivant la structure topologique du tableau. A partir de ce fichier, *Make Array* établit les correspondance avec les dessins de masque de chacune des cellules et produit le dessin de masque final.

Le générateur du tableau de processeurs, écrit en C, est référencé par *Gen_Proc*.

Chapitre 3

La mémoire

3.1 Introduction

La mémoire du circuit API69 contient les cœfficients de substitution représentant la proximité des touches d'un clavier les unes par rapport aux autres. Un clavier classique possède environ 50 touches dédiées aux caractères intervenant dans la composition d'un texte (lettres, chiffres, caractères de ponctuation, caractères divers tels que $, %, etc). Par conséquent, l'information relative à l'éloignement de deux touches d'un clavier peut être représentée par un tableau bidimensionnel :

$$T_{sub}[t][r]$$

qui délivre le coût de substitution du caractère t par le caractère r. La taille du tableau correspondant est donc de l'ordre de 2.5 Koctets ($T_{sub}[t][r]$ peut être différent de $T_{sub}[r][t]$).

Au cours d'un cycle systolique, tous les processeurs doivent consulter cette table. Un cycle comprenant seulement quelques phases d'horloge, il est inenvisageable de partager l'accès de cette table entre les 69 processeurs. La solution la plus immédiate consiste à ce que chaque processeur en possède une copie. Cette approche un peu brutale se traduit directement au niveau mise en œuvre VLSI par une taille de mémoire conséquente (supérieure à 1 Mbit). De plus, chaque processeur doit posséder un système d'adressage mémoire complet. Cette solution est évidemment peu intéressante et difficilement viable en terme de densité d'intégration.

Plusieurs constatations conduisent, cependant, à réduire très notablement cette capacité de stockage. La première repose sur une analyse plus fine du problème en observant, à un instant donné, les besoins d'un processeur : la comparaison d'un mot test (mot erroné) avec un ensemble très important de références (le dictionnaire) fait que celui-ci effectue pendant toute la durée du processus un calcul où intervient seulement un et un seul caractère du mot test. On peut donc considérer ce dernier comme une constante pour toute la durée du processus de comparaison ce qui permet de réduire

à une dimension la table T_{sub}. Le processeur ne mémorise, en effet, que la portion de table qui l'intéresse à savoir, $T_{sub}[t][r_j]$ où t représente le caractère du mot test impliqué dans le calcul et r_j, l'ensemble des autres caractères de l'alphabet.

Cette réduction implique cependant qu'à chaque nouvelle comparaison les tables soient remises à jour dans chaque processeur (le mot erroné change!). Cela suppose, et c'est le cas ici, que le temps de cette opération soit négligeable par rapport au reste du calcul.

La seconde constatation montre que les cinq processeurs d'une même colonne utilisent également, et de la même manière, le même caractère du mot test. Les mémoires de ces processeurs possèdent donc des contenus identiques. De plus, rappelons qu'un cycle systolique comprend au moins cinq cycles machines dont un seul nécessite un accès à la mémoire. Le partage de la mémoire est donc tout à fait possible entre les processeurs. Il faut simplement garantir qu'il n'y ait pas d'accès simultanés. Le fonctionnement pipeline et synchrone du réseau assure que de tels événements ne se produisent pas. Le contrôle global du circuit est un peu plus complexe puisqu'il faut gérer dans le temps cet accès réparti de la mémoire.

La dernière remarque est d'ordre algorithmique et a été validée à l'issu de nombreux tests. Elle consiste à ne prendre en considération que le coût de substitution de deux caractères pour lesquels les touches associées sont proches l'une de l'autre. Cela signifie, par exemple, que le coût de substitution du caractère **z** par le caractère **k** (sur un clavier de type qwerty) sera identique au coût de substitution du caractère **z** par le caractère **p** car les caractères **k** et **p** sont *éloignés* du caractère **z** et que, par conséquent, les probabilités de confusion entre les touches (**z**,**k**) et (**z**,**p**) sont faibles et quasiment identiques.

En pratique, une dizaine de valeurs significatives par touche sont nécessaires pour obtenir de bons résultats. D'un point de vue architecturale, cette réduction au niveau de la mémorisation des coûts de substitution peut être mise en œuvre par l'utilisation d'une mémoire associative. Celle-ci stocke uniquement les touches significatives et leurs valeurs de substitution associées, soit une dizaine de couple (ref,coût). Lorsqu'aucune correspondance n'est trouvée, une valeur par défaut doit être délivrée (elle correspond au coût de deux touches considérées comme éloignées).

Ces optimisations successives conduisent à une mémoire de taille d'environ 2.5 Kbits, soit un gain de 400 par rapport à la proposition initiale.

3.2 Description fonctionnelle

L'ensemble de la mémoire du circuit API69 se divise en 15 modules identiques, un par colonne de processeurs. Chaque module possède une capacité de mémorisation égale à 21 mots repartis en 10 couples (référence,coût) plus une valeur représentant le coût par défaut lorsqu'aucune correspondance n'a été trouvée. Les références sont stockées sur 8 bits (possibilité de 256 caractères différents) et les coûts sur 7 bits seulement.

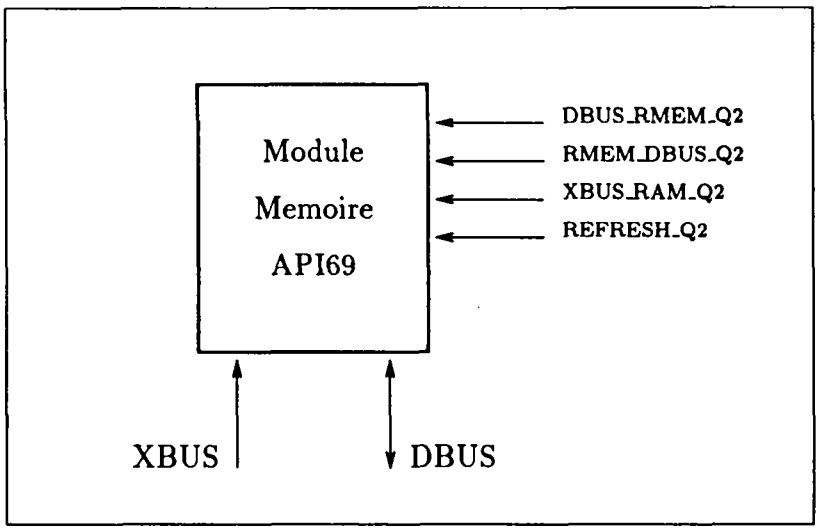


Figure 3.1 : mémoire API69

La raison pour laquelle les coûts ne sont représentés que sur 7 bits est explicitée au paragraphe 3.4.

Les 15 modules sont identiques et fonctionnent tous de manière synchrone. Aussi, pour simplifier, la description fonctionnelle de la mémoire ne porte que sur un module.

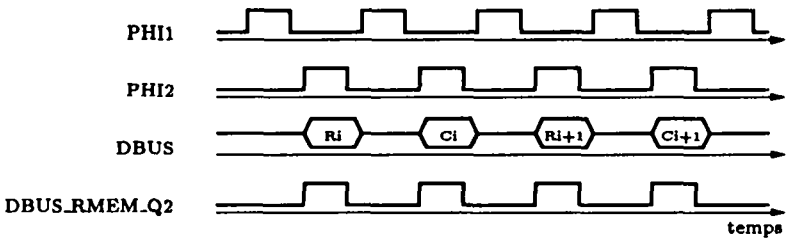
La figure 3.1 décrit l'aspect externe d'un module. Deux bus (XBUS et DBUS) permettent d'y accéder et quatre commandes gèrent son fonctionnement.

3.2.1 Chargement de la mémoire

Le chargement d'un module mémoire peut être vu comme le chargement d'un registre à décalage de 21 mots. Les mots à mémoriser sont présentés successivement sur DBUS (pendant phi2) en activant simultanément le signal DBUS_RMEM_Q2.

L'ordre dans lequel les données doivent être présentées correspond aux dix couples (référence-coût) plus une dernière valeur représentant le coût émis par défaut lorsqu'aucune référence n'est sollicitée.

Le chronogramme ci-après résume le protocole de chargement pour les couples (référence-coût) R_iC_i et $R_{i+1}C_{i+1}$:



Lorsque la mémoire n'est pas en mode de chargement (signal DBUS_RMEM_Q2 inactif), elle doit être rafraîchie. Le signal REFRESH_Q2 assure cette fonction.

3.2.2 Lecture de la mémoire

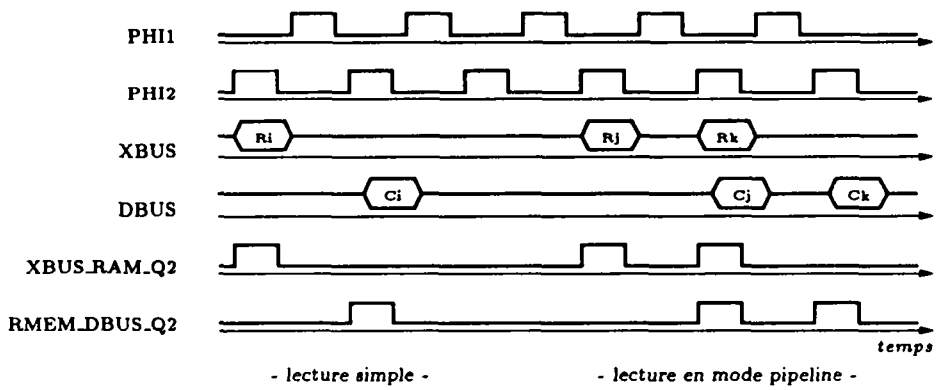
lecture simple

La première étape dans le processus de lecture consiste à présenter (sur phi2) une valeur dite de référence sur XBUS tout en activant la commande XBUS_RAM_Q2. Cette action a pour effet de sélectionner en interne le coût associé à cette référence (cette sélection s'effectue au cours de la phase phi1 suivante).

En activant ensuite le signal RMEM_DBUS_Q2, DBUS est chargé par le coût relatif à la référence précédemment mémorisée. Si la référence n'existe pas en mémoire c'est la valeur par défaut qui est délivrée.

lecture en mode pipeline

Le processus est identique à la lecture simple. Cependant, à chaque phase phi2, une référence est présentée sur XBUS. De cette manière, avec une phase phi2 de décalage, les valeurs correspondantes sont lues. Le chronogramme ci-dessous résume les deux modes de lecture de la mémoire:



La commande RMEM_DBUS_Q2 peut être activée plusieurs fois de suite sans introduction préalable d'une nouvelle référence. Dans ce cas, la mémoire délivre le coût

associé à la dernière référence mémorisée.

3.3 Architecture

La figure 3.2 représente l'architecture interne d'un module mémoire du circuit API69. Le schéma montre seulement la mémorisation de quatre couples de valeur (le principe reste le même avec 10 couples). Les registres P (*pattern*) et V (*value*) sont les principaux composants de la mémoire. Les premiers mémorisent les références (sur 8 bits) et les seconds, les coûts associés (sur 7 bits).

L'initialisation de ces registres se fait par décalages successifs : le registre Vdef est chargé par la valeur présente sur le bus DBUS, le registre V_0 par l'ancienne valeur de Vdef, les registres V_i par la valeur des registres P_{i-1} et, enfin, les registres P_i par la valeur des registres V_i. Ce décalage s'effectue chaque fois que la commande DBUS_RMEM_Q2 est activée.

Le registre d'entrée RAM permet la mémorisation d'une référence. Celle-ci est propagée à l'ensemble des registres P. La commande XBUS_RAM_Q2 charge le registre RAM avec le contenu de XBUS. Ce registre d'entrée est dynamique.

Un registre P a également le rôle de déterminer si la valeur présente sur REFBUS est identique à la valeur qu'il mémorise. En cas de succès, le signal M est activé. Celui-ci commande le registre V associé qui délivre alors sur MEMBUS la valeur du coût correspondant à la référence. L'ensemble de ces opérations s'effectuent pendant la phase phi1. Il est clair que 2 registres P mémorisant une même référence mais avec des coûts différents produiraient un fonctionnement anormal.

En fin de phase (phi1) cette valeur est mémorisée dans le registre RMEM. La commande RMEM_DBUS_Q2 permet d'émettre son contenu sur DBUS.

Si aucun registre P_i n'a activé son signal M_i alors le signal Md devient actif et, par conséquent, indique au registre Vdef d'émettre son contenu sur MEMBUS.

3.4 Mise en œuvre VLSI

3.4.1 Plan de masse d'un module mémoire

La figure 3.3 donne l'implantation VLSI d'un module mémoire de 4 couples (référence,valeur) de 4 bits. Ce module est découpé en 4 tranches de 1 bit, chaque bit étant ensuite subdivisé en cellules élémentaires. La tranche de poids fort est différente des autres tranches, elle gère en plus la propagation du signal Md (cf figure 3.2).

Un bit est composé de cellules CP et cellules CV mémorisant respectivement les références (*pattern*) et les valeurs associées. La première cellule CV (celle du bas) est connectée au bus DBUS. Toutes les autres cellules CV et CP sont connectées en registre à décalage.

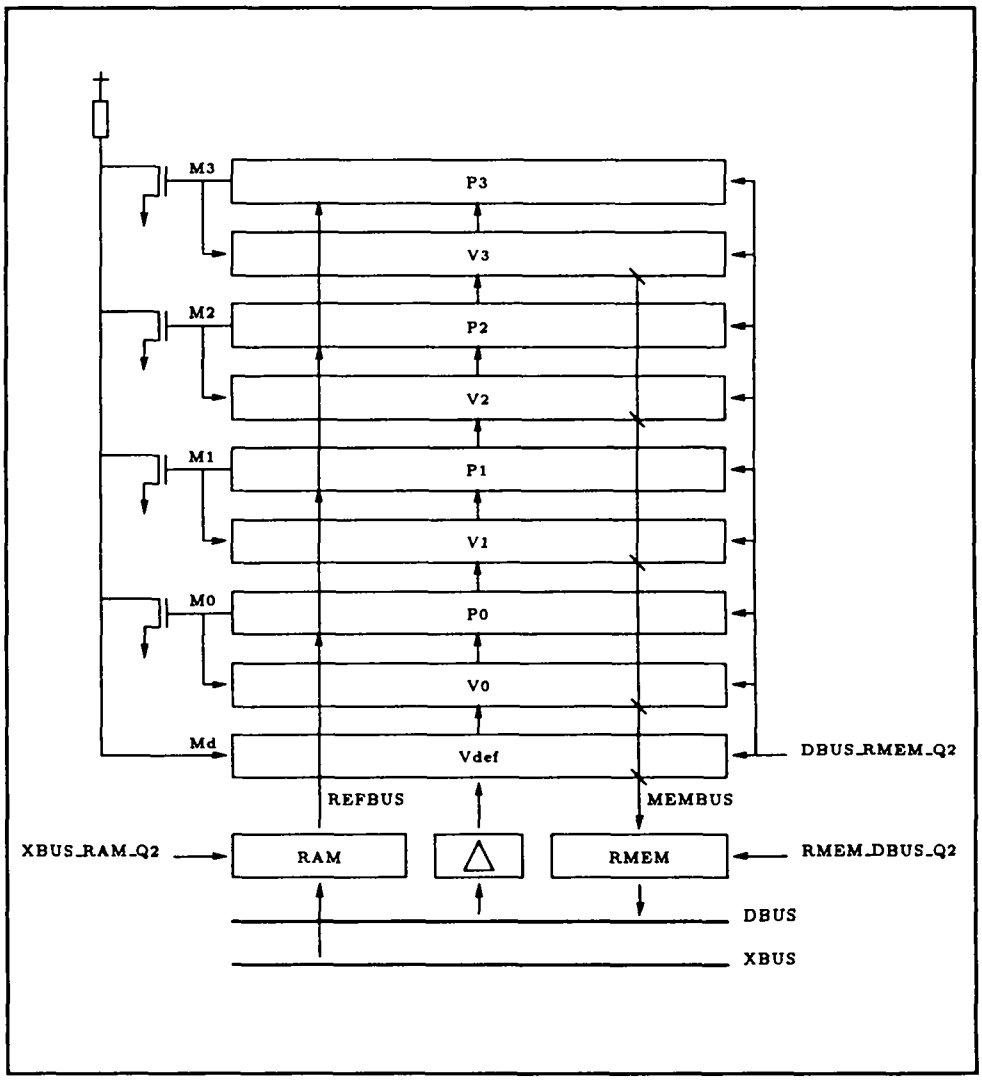


Figure 3.2 : architecture d'un module mémoire

Les registres d'entrée et de sortie RAM et RMEM sont placés l'un au dessus de l'autre. Les bus DBUS et XBUS traversent verticalement l'ensemble des cellules et sont, de cette manière, accessibles aux registres RAM, RMEM et Vdef. Les commandes transitent horizontalement via les cellules où elles sont utilisées. L'ensemble de la mémoire peut ainsi être constitué par l'aboutement horizontal des 15 modules mémoire, les fils de commande étant communs à tous les modules.

Le signal Md est produit par la cellule CPV correspondant au bit de poids fort de chaque registre V. Ces cellules contiennent la logique nécessaire pour réaliser cette tâche. Elles sont donc différentes des cellules CV et ne possèdent pas toutes les fonctionnalités désirées, en particulier celle qui consiste à émettre vers le registre RMEM (manque de place!). Ceci a pour conséquence de ne pouvoir mémoriser qu'une valeur sur 7 bits, le bit de poids fort étant systématiquement positionné à zéro. Par contre, elles sont capables de faire transiter une valeur sur 8 bits pour être en mesure de stocker (par décalage successifs) une référence codée sur 8 bits.

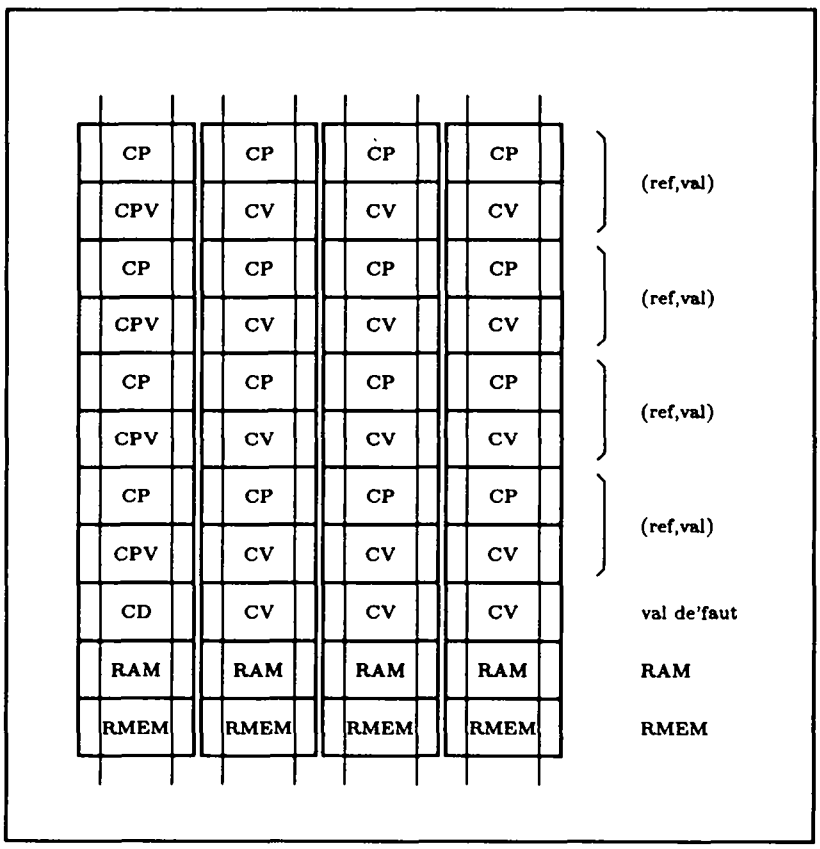


Figure 3.3 : plan de masse d'un module mémoire

3.4.2 Génération du dessin de masque

La mémoire complète consiste en l'aboutement horizontal de 15 modules mémoire identiques. Chaque module étant également réalisé par aboutement de cellules, le dessin de masque de la mémoire (tout comme celui du réseau de processeurs) est équivalent à un tableau de cellules. Là, encore, aucun routage supplémentaire n'est nécessaire pour construire ce module.

Les paramètres requis pour générer ce module de manière automatique sont :

- le nombre de couple (référence,coût),
- la largeur du chemin de données,
- le nombre de colonnes de processeurs.

Le générateur de mémoire est référencé par *Gen_Mem*.

Chapitre 4

Le réseau d'alimentation

4.1 Introduction

Le réseau d'alimentation a pour fonction d'alimenter les mémoires de substitution avec les caractères des mots du dictionnaire : au cours d'un cycle systolique, chaque processeur est pourvu d'une nouvelle donnée, le coût de substitution d'un caractère référence avec le caractère de la chaîne test commun à tous les processeurs d'une colonne.

Afin de minimiser le matériel implanté sur le silicium (cf section 1.4.1) les caractères des mots du dictionnaire ne transitent pas par le réseau de processeurs mais par le réseau d'alimentation (les deux réseaux ont des structures équivalentes). Ce réseau adresse directement des mémoires qui produisent des données pour les processeurs. Ce partitionnement est dû aux constatations suivantes :

- tous les processeurs d'une même colonne utilisent les mêmes coûts de substitution,
- un cycle systolique durant 6 cycles machines et un processeur n'accédant qu'une seule fois à la mémoire au cours d'un tel cycle, la mémoire peut être partagée entre les 5 processeurs d'une même colonne.

Le flot de données est en avance d'un cycle systolique par rapport au calcul effectué sur le réseau de processeurs, ceci pour permettre l'accès séquentiel aux mémoires. Pendant le premier cycle machine, les caractères pour le prochain cycle systolique sont chargés parallèlement dans le réseau d'alimentation, avec décalage global dans ce réseau. Dans ce même cycle (machine), les processeurs consomment les coûts qui avaient été envoyés au cycle systolique précédent. Pendant les cinq cycles machine suivants, l'accès aux mémoires se fait séquentiellement, diagonale par diagonale, et les données qu'elles délivrent sont transmises aux processeurs.

4.2 Description fonctionnelle

Le réseau, tel qu'il a été défini, demande un nouveau mot référence à chaque cycle systolique, soit l'équivalent de 15 caractères. Un cycle systolique durant 6 cycles machines, l'acquisition d'un nouveau mot test peut être multiplexé dans le temps. Le meilleur découpage consiste à l'acquérir en 5 cycles machines, ce qui réduit l'interface avec le monde extérieur à trois entrées.

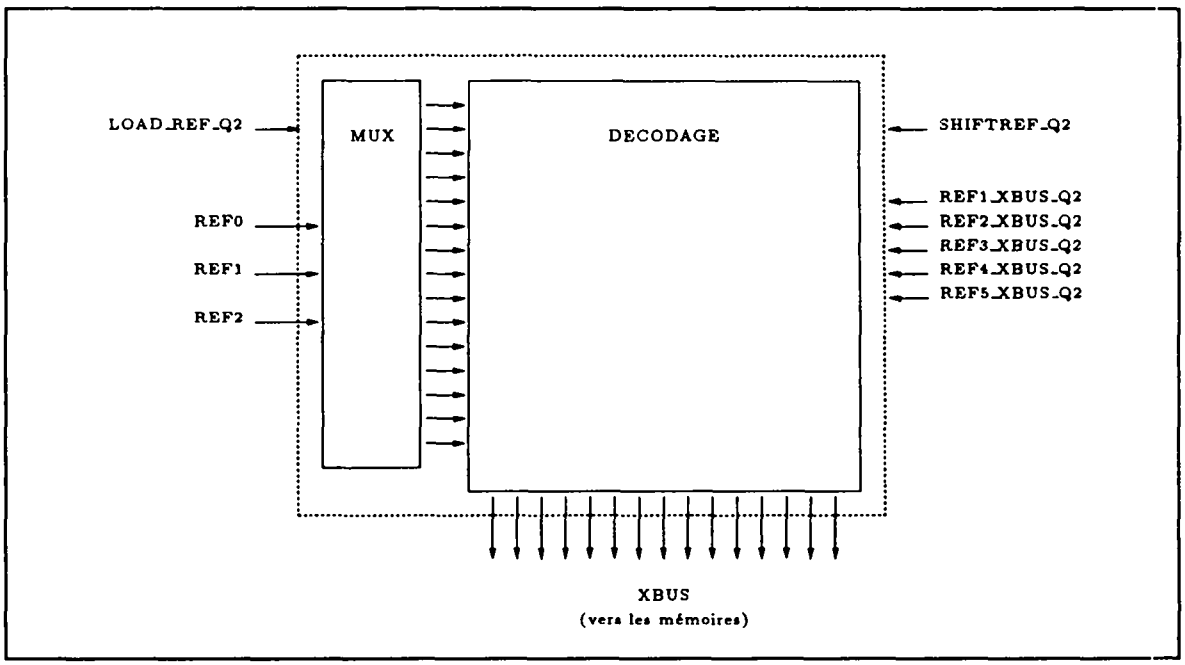


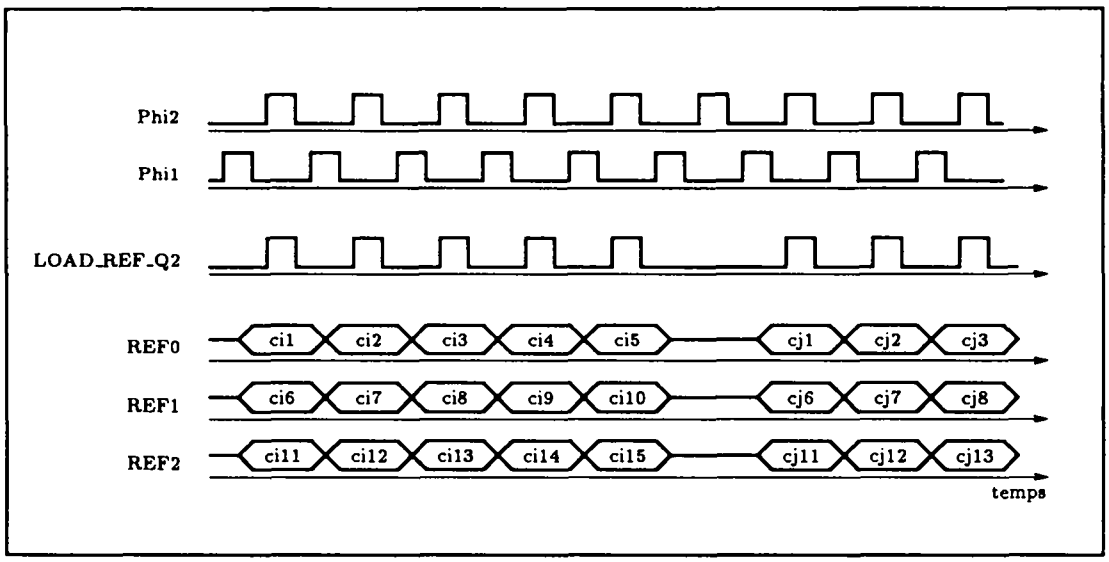
Figure 4.1 : réseau alimentation

D'un point de vue fonctionnel, le réseau d'alimentation peut être divisé en deux parties, une partie acquisition proprement dite et une partie décalage. Ces deux parties seront référencées, dans la suite du texte, par module de multiplexage et module de décalage.

4.2.1 Le module de multiplexage

Trois entrées, notées REF0, REF1 et REF2, permettent d'acquérir les caractères d'un mot référence sous l'action de la commande LOADREF_Q2 (cf figure 4.1). L'entrée REF0 reçoit séquentiellement les 5 premiers caractères, l'entrée REF1, les 5 suivants, et l'entrée REF2, les 5 derniers. Il ne s'agit pas, en fait, des caractères du même mot référence mais des caractères à présenter, à un instant donné, au réseau (il faut tenir

compte du décalage dû au fonctionnement systolique et à la topologie du réseau). Le chronogramme ci-après précise comment sont acquis les caractères des mots du dictionnaire.



L'activation de la commande LOADREF_Q2 provoque la lecture des entrées REF0, REF1 et REF2. En activant 5 fois cette commande, l'ensemble des 15 caractères constituant les données à fournir au réseau est acquis. A l'issu de la cinquième activation de la commande, les données sont présentes en entrée du réseau et donc prêtes à être utilisées.

4.2.2 Le module de décalage

La commande SHIFTREF_Q2 décale d'une position vers la droite les références du réseau. Les registres situés à gauche reçoivent les caractères acquis par le module de multiplexage. Les commandes REF1_XBUS_Q2 à REF5_XBUS_Q2 émettent respectivement le contenu des diagonales sur le bus XBUS de chaque mémoire. Rappelons que ce bus est connecté à l'entrée *pattern* des mémoires.

4.3 Architecture

4.3.1 Le module de multiplexage

Le module de multiplexage est représenté par le schéma de la figure 4.2. Il est constitué de trois registres à décalage de 5 éléments chacun. Le décalage est commandé par le signal LOADREF_Q2.

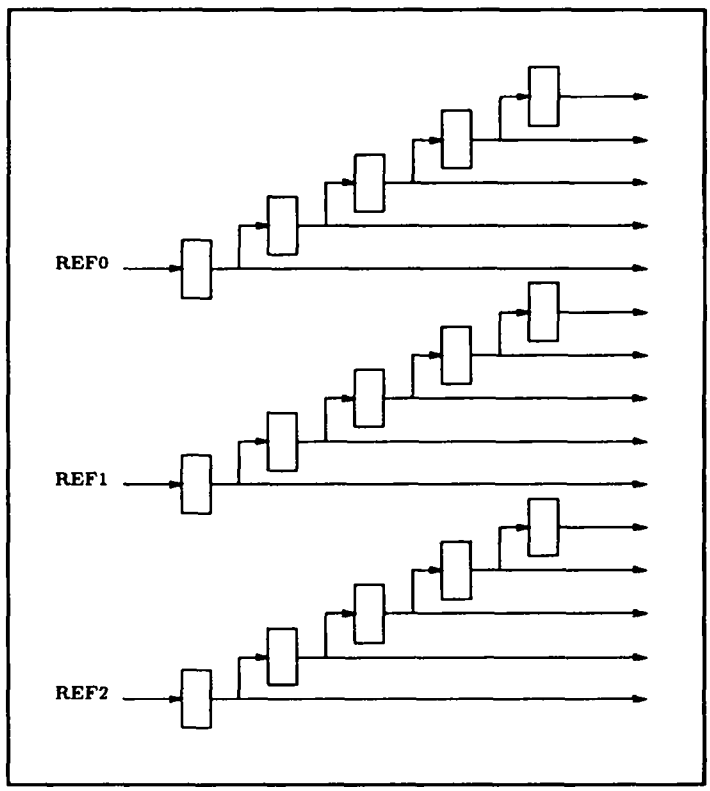


Figure 4.2 : module de multiplexage

4.3.2 Le module de décalage

Ce module comprend 15 registres à décalage composés de 5 registres chacun, à l'exception des deux premiers et des deux derniers. Les registres ont une sortie trois états pour permettre l'émission d'une donnée sur les bus XBUS des mémoires. Le décalage est commandé par le signal SHIFTREF_Q2 et l'émission des données sur les bus par les signaux REF1_XBUS_Q2 à REF5_XBUS_Q2.

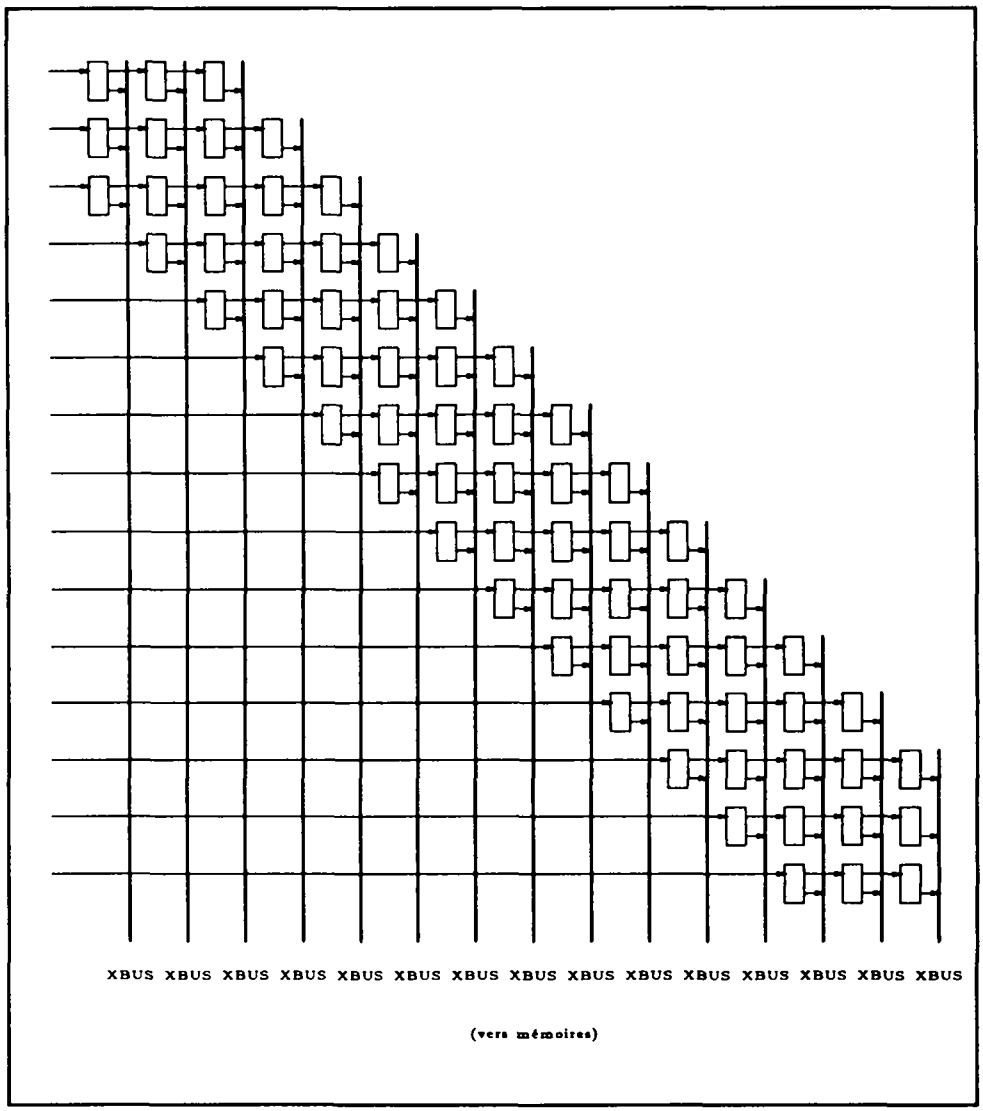


Figure 4.3 : module de décalage

4.4 Mise en œuvre VLSI

L'implémentation VLSI d'un module vise, en général, à optimiser la surface du silicium. Dans le cas du réseau d'alimentation cela consiste à le *rectangulariser*, c'est à dire à le déformer de manière à obtenir une structure régulière telle que celle qui est représentée sur le schéma de la figure 4.4. Cette transformation physique est identique à celle qui a été effectuée sur le réseau de processeurs

La structure topologique obtenue, pour le module de décalage, est une matrice dont les lignes représentent les diagonales du réseau. La matrice est complétée dans les coins supérieurs gauches et inférieurs droits par des registres supplémentaires dont le but est de préserver la régularité de l'ensemble. De cette manière, un routage particulier pour acheminer les signaux de commande ou les fils d'alimentation n'est pas nécessaire. D'un point de vue fonctionnel, ces registres ne jouent aucun rôle.

L'implantation physique du module de multiplexage cherche également la régularité en disposant sur une seule ligne les registres à décalage (cf figure 4.4).

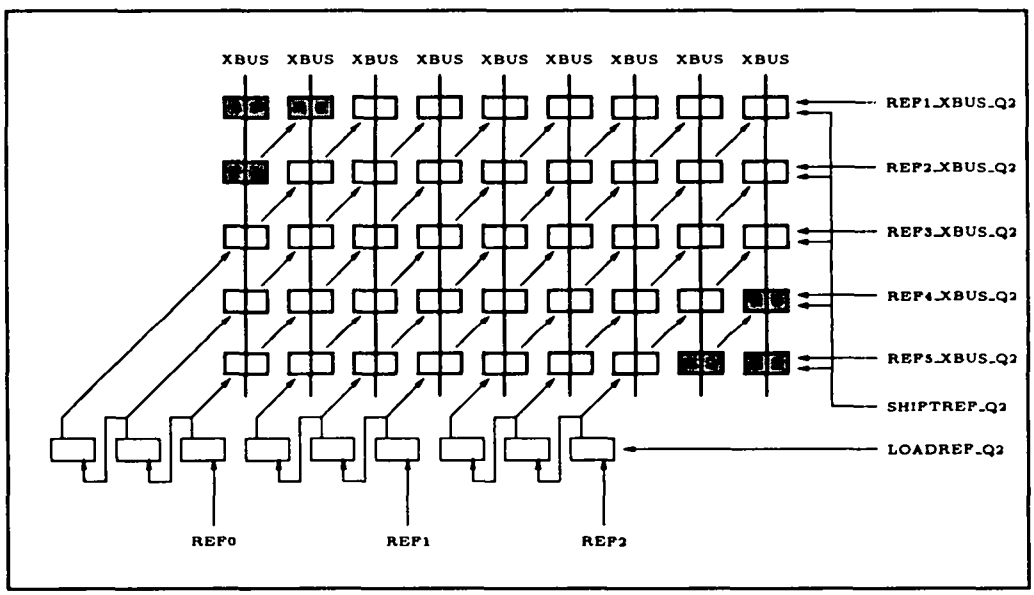


Figure 4.4 : *rectangularisation* du réseau

L'étape finale est d'agencer l'interconnexion de ces deux modules entre eux afin de disposer d'un module complètement régulier. Le schéma de principe du plan de masse est donné par la figure 4.5. Par rapport au dessin de la figure 4.4, le schéma a été renversé (miroir vertical) de manière à ce qu'il s'intègre directement dans le plan de masse de l'ensemble du circuit.

La régularité a été obtenue en diffusant à l'intérieur du module de décalage la

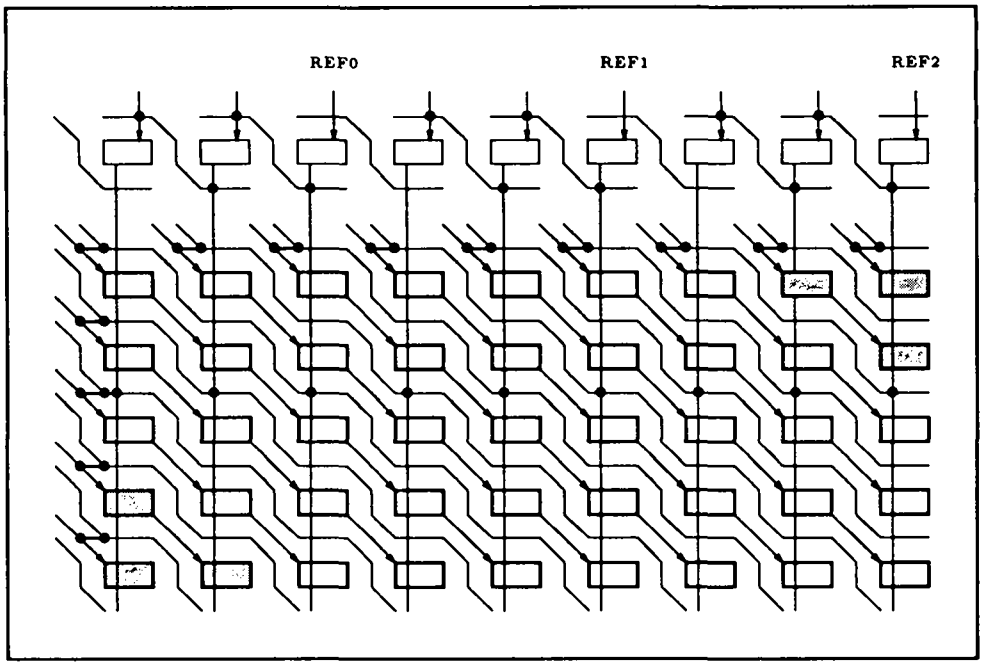


Figure 4.5 : principe du plan de masse

connectique nécessaire au cablage: toutes les cellules du module de décalage sont traversées verticalement par les sorties du registre à décalage du module de multiplexage. Celles-ci sont ensuite propagées en diagonale et connectées aux cellules périphériques gauches et aux cellules périphériques supérieures. Les points de connexion apparaissent par des points noirs sur la figure 4.5.

4.4.1 Génération du dessin de masque

Le module d'alimentation est un assemblage régulier de cellules : les cellules constituant la partie multiplexage et celles constituant la partie décalage. L'interconnexion entre ces cellules est réalisable à partir de cellules de routage de manière à ce que l'ensemble forme une structure régulière paramétrable. La figure 4.6 donne le plan de masse de l'ensemble et indique les différentes cellules de routage à mettre en place.

Les paramètres requis pour générer ce module de manière automatique sont :

- le nombre de colonnes de processeurs,
- le nombre de diagonales de processeurs,
- le nombre d'entrées référence

Le générateur du module d'alimentation est référencé par *Gen_Alim*.

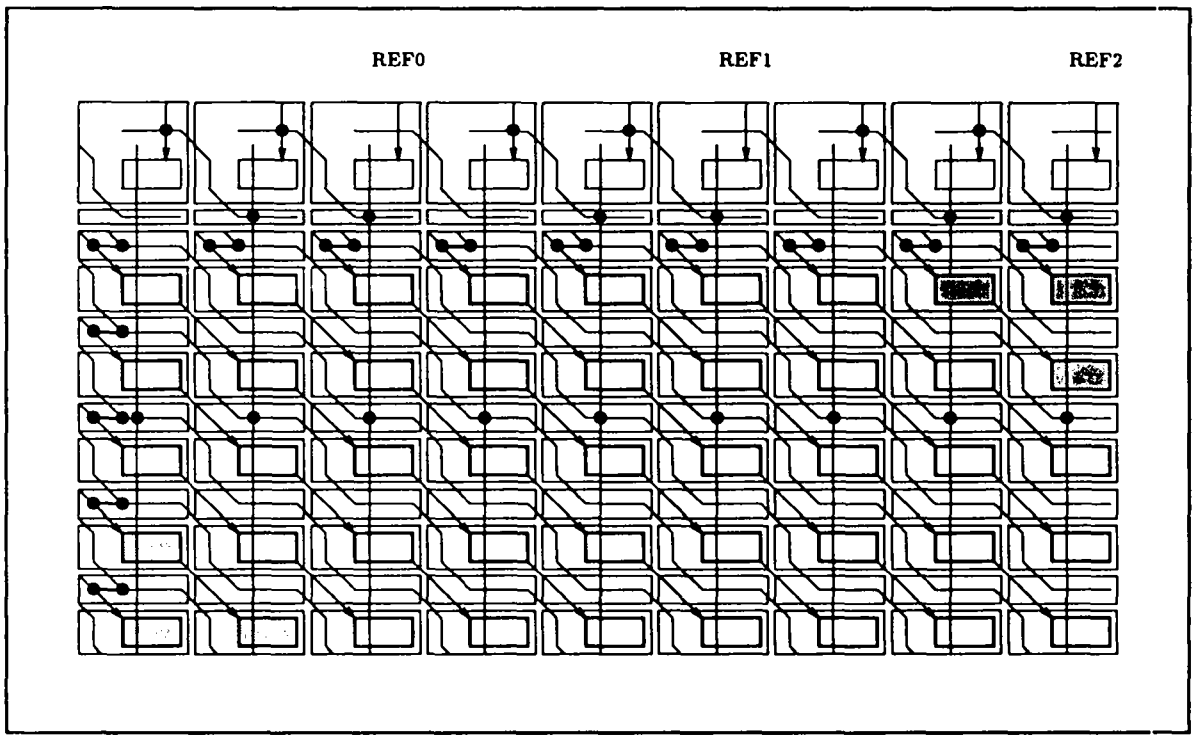


Figure 4.6 : plan de masse

Chapitre 5

Le registre de configuration

5.1 Introduction

Le registre de configuration n'intervient pas directement pendant un calcul. Il a deux rôles :

- **initialiser** les mémoires avec les tables de substitution correspondant au mot à traiter,
- **tester** le circuit, c'est à dire les mémoires et l'ensemble des processeurs.

Le principe est de pouvoir accéder aisément aux bus internes du circuit, notamment aux bus mémoire (XBUS) et aux bus de diffusion (DBUS) présents sur chaque colonne de processeurs. L'accès à ces bus autorise l'accès à toutes les mémoires et à tous les registres de sortie des processeurs puisque ceux-ci peuvent émettre leurs résultats sur le bus de diffusion de la colonne à laquelle ils appartiennent.

La figure 5.1 rappelle l'interconnexion des différents modules du circuit API69 via les deux principaux bus XBUS et DBUS. Le premier est dédié à l'alimentation des mémoires. Le second présente un aspect beaucoup plus général puisqu'il permet de véhiculer des données en provenance de la mémoire ou du réseau de processeurs.

Toutefois, l'accès aux bus n'est pas direct et s'effectue à l'aide d'un registre à décalage, le registre de configuration, dont chaque étage est en liaison avec une colonne de processeurs/mémoire. Par exemple, pour obtenir la valeur du registre de sortie d'un processeur, il faut opérer en deux temps : émettre vers le registre de configuration toute la ligne correspondante, puis décaler le nombre de fois nécessaire pour avoir la valeur de la colonne voulue.

De même, l'écriture dans une mémoire de la colonne numéro i nécessite de présenter la donnée sur l'entrée Din et de la propager par i décalages. Lorsqu'elle est arrivée dans la bonne colonne, on l'émet sur le bus de diffusion et on active la commande d'écriture mémoire.

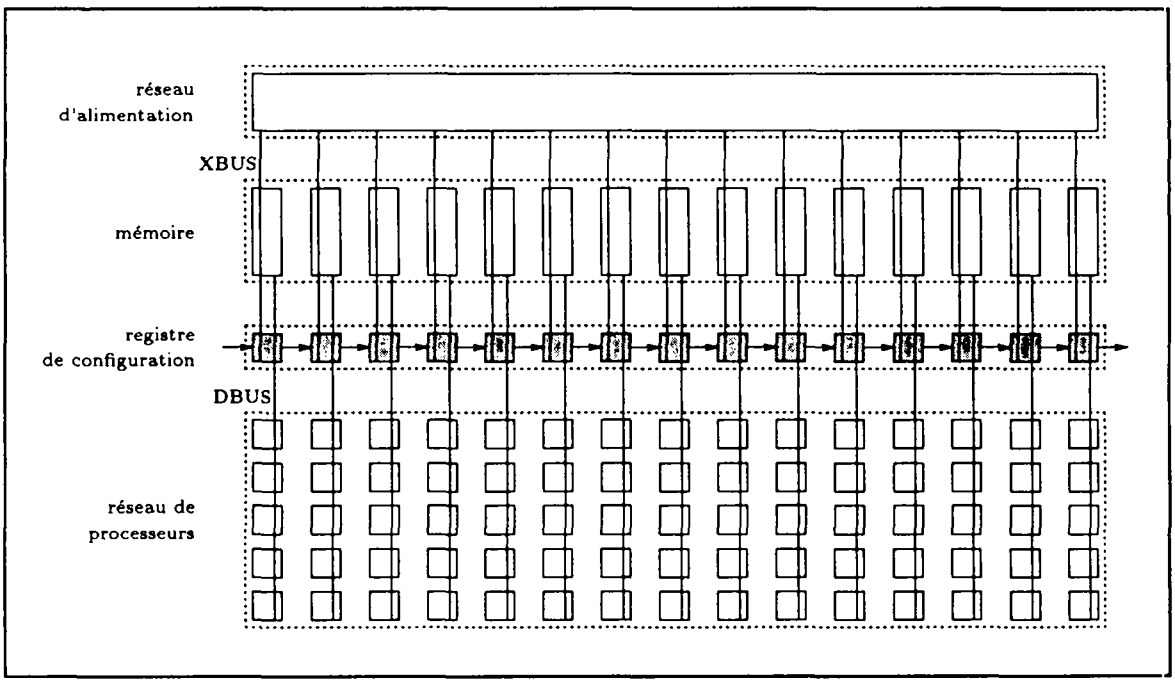


Figure 5.1 : situation du registre de configuration

On cherchera évidemment à optimiser l'usage de cette ressource sur l'ensemble des colonnes du réseau ou sur l'ensemble des mémoires.

5.2 Description fonctionnelle

Le registre de configuration est un registre à décalage de 15 étages. Chaque étage est connecté au bus mémoire (XBUS) et au bus de diffusion (DBUS) et peut lire ou écrire sur chacun de ces deux bus. La figure 5.2 donne le schéma de principe de ce module. Le registre de configuration est géré par 5 commandes :

- SHIFT_CR_Q2 : décalage vers la droite des informations contenues dans chaque étage. La cellule CR1 reçoit la donnée présente en Din et la nouvelle donnée mémorisée dans la cellule CR15 est délivrée en sortie.
- CR_XBUS_Q2 : le contenu de chaque cellule CRi est émis sur chaque bus XBUSi.
- CR_DBUS_Q2 : le contenu de chaque cellule CRi est émis sur chaque bus DBUSi.
- XBUS_CR_Q2 : le contenu de chaque bus XBUSi est mémorisé dans la cellule CRi
- DBUS_CR_Q2 : le contenu de chaque bus DBUSi est mémorisé dans la cellule CRi

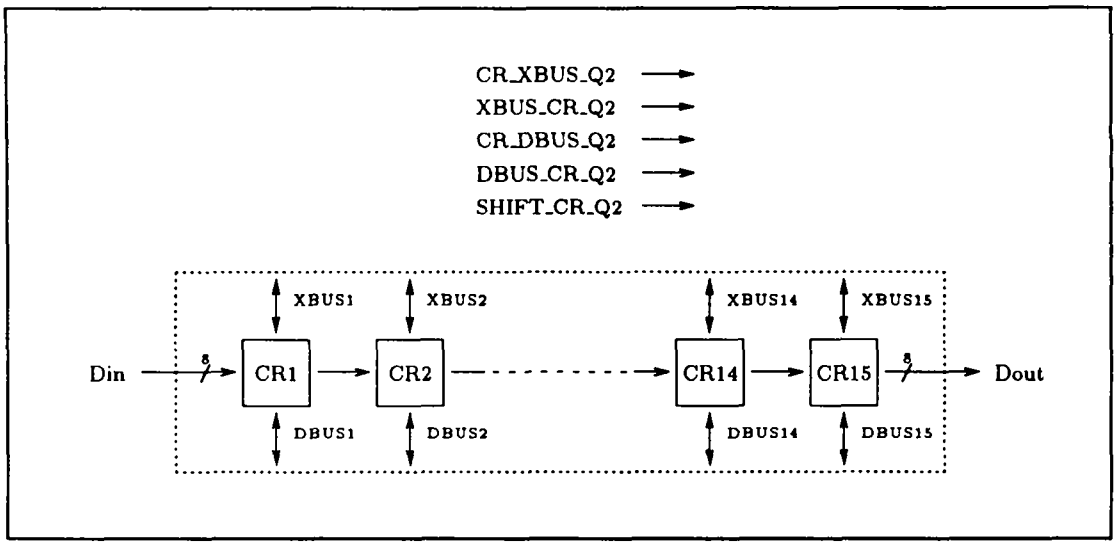


Figure 5.2 : registre de configuration

Ainsi, pour initialiser les mémoires avec les couples (référence,coûts), par exemple, on doit effectuer la suite d'opérations décrite par l'algorithme suivant :

```
for (i=0;i<MEMSIZE;i++)
  {
    for (j=0;j<15;j++) { Din := R[i][j]; SHIFT_CR; }
    CR_DBUS, DBUS_RMEM;
    for (j=0;j<15;j++) { Din := C[i][j]; SHIFT_CR; }
    CR_DBUS, DBUS_RMEM;
  }
```

Les références et les coûts sont introduits séquentiellement par paquet de 15 puis stockés dans la mémoire.

5.3 Architecture

La figure 5.3 donne le schéma de principe d'une cellule du registre de configuration. Elle est basée sur une cellule de mémorisation dynamique classique composée de portes de transfert et de deux inverseurs. Pour un bon fonctionnement de l'ensemble, les signaux de commande SHIFT_CR_Q2, XBUS_CR_Q2 et DBUS_CR_Q2 doivent être exclusifs. Par contre, les signaux de commandes CR_XBUS_Q2 et CR_DBUS_Q2 peuvent être

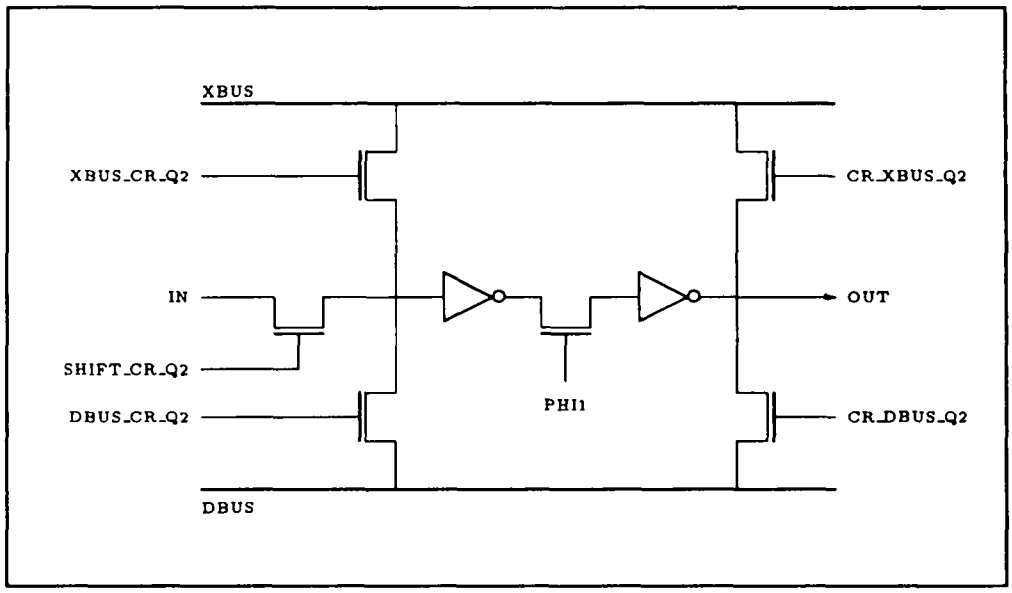


Figure 5.3 : cellule du registre de configuration

actionnés simultanément; cela a pour effet de charger en même temps les bus XBUS et DBUS avec la même valeur.

Précisons que le schéma de la figure 5.3 n'est qu'un schéma de principe et qu'en réalité, la cellule est plus complexe. Les bus sont, en effet, des bus à précharge, ce qui implique un interfaçage particulier à l'intérieur des cellules.

Remarquons, enfin, que l'utilisation d'un registre à décalage dynamique demande une consommation immédiate des données qui sont acquises. Dans le cas présent, c'est l'usage normal de ce module qui ne sert qu'à transmettre des données et qui n'a aucun rôle de mémorisation.

5.4 Mise en œuvre VLSI

La mise en œuvre VLSI consiste à abouter linéairement 8 × 15 cellules identiques et à les connecter en registre à décalage. Le schéma du haut de la figure 5.4 indique comment ces connexions doivent être réalisées (l'exemple porte sur un registre à décalage de 2 étages avec un chemin de données sur 4 bits).

Les cellules de base du registre de configuration (notées CR) sont traversées par les bus XBUS et DBUS et ont une entrée et une sortie désignées respectivement par IN et OUT. Ces entrées sont situées sur le bord inférieur de la cellule.

Le schéma du bas (figure 5.4) indique comment les connexions inter-cellules sont réalisées. Deux cellules de routage sont nécessaires. De cette manière, à chaque étage du registre à décalage (ensemble de 8 cellules CR) est associé une matrice de 8 × 8

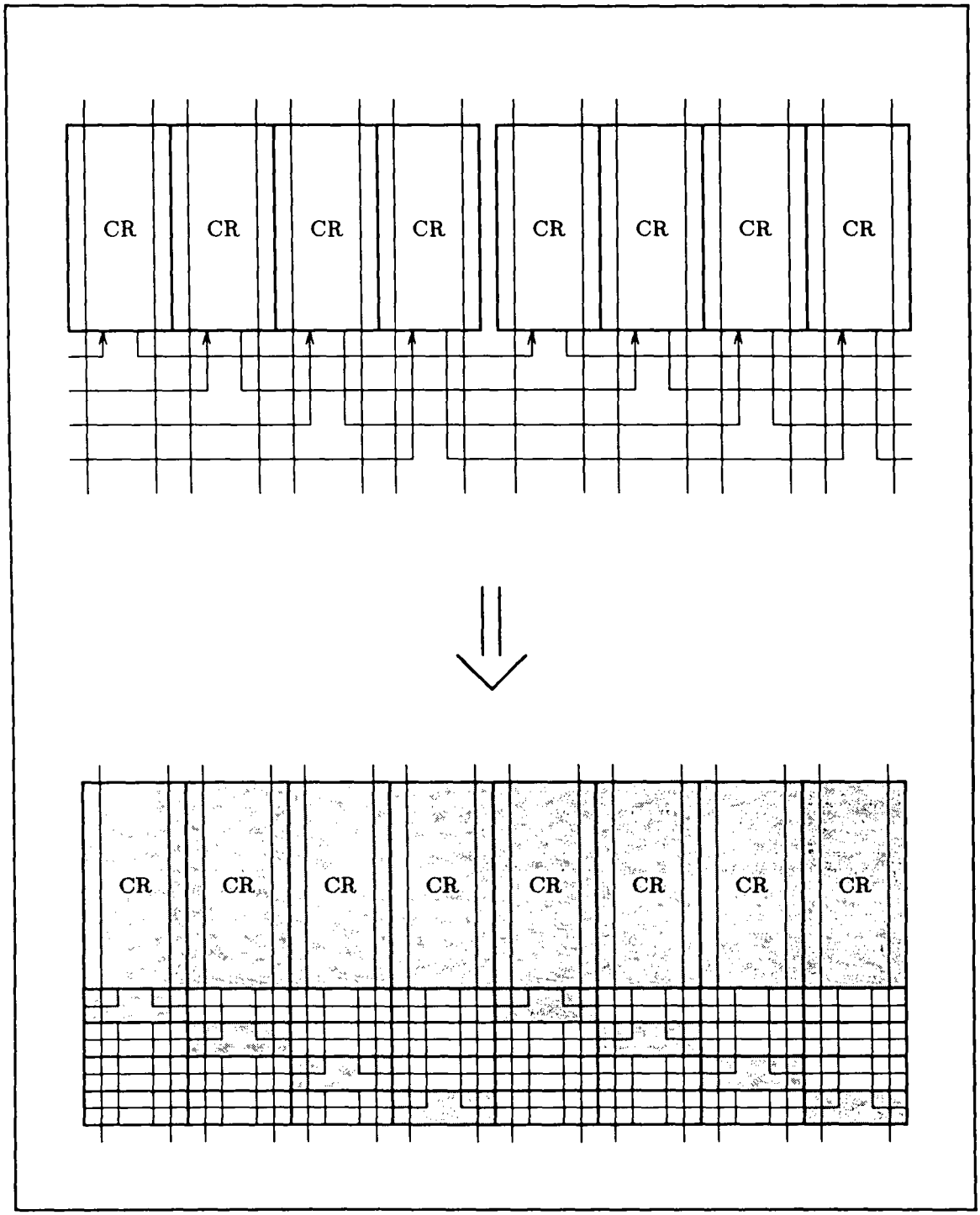


Figure 5.4 : implémentation VLSI

cellules de routage dont les cellules de la diagonale sont différentes.

5.4.1 Génération du dessin de masque

La génération du dessin de masque sous forme d'un tableau de cellules est immédiate. Les paramètres qui doivent être pris en compte pour construire automatiquement ce module sont :

- le nombre de colonnes de processeurs,
- la largeur du chemin de données.

Le générateur du registre de configuration est référencé par *Gen_RegConf*.

Chapitre 6

Le contrôle

6.1 Introduction

Les modules décrits dans les chapitres précédents ont tous en commun d'être contrôlés par des signaux de commandes spécifiques. L'ensemble fonctionnant sur la base d'une horloge à deux phases non recouvrantes (PHI1 et PHI2), ces signaux peuvent être soit actifs sur la phase PHI1, soit actifs sur la phase PHI2.

Le module de contrôle synchronise l'ensemble de ces modules (réseau d'alimentation, mémoire, réseau de processeurs, registre de configuration) en activant les signaux appropriés.

Un processus de comparaison se décompose en deux phases : initialisation des mémoires en fonction du mot à corriger puis comparaison des références avec ce mot erroné. On peut donc considérer qu'il existe deux processus distincts et bien définis auxquels sont associés, dans le temps, des instants d'activation des signaux de commande.

Une première approche consisterais à *cabler* directement sur silicium un automate réalisant ces deux tâches. L'inconvénient majeur de cette solution est que toute modification de l'algorithme (même minime) est impossible. Or, même si le circuit est fortement dédié à une application de correction, les éléments qui le composent permettent certaines variations autour de l'algorithme de programmation dynamique de base. Par exemple, le circuit peut être utilisé pour la reconnaissance d'adresses postales (après lecture optique). Dans ce cas, l'algorithme est légèrement différent car il ne fait pas intervenir les erreurs de permutation.

Notre solution est à mi-chemin entre l'intégration d'un séquenceur et le contrôle direct de tous les signaux de commande. Elle consiste à regrouper les signaux qui ne peuvent être émis simultanément, puis à affecter un codage sur ce groupe de signaux. L'entrée du module de contrôle peut donc être assimilée à une instruction comportant plusieurs champs, chaque champ étant dédié à un groupe particulier.
Le codage est réparti en 5 champs comme le montre la figure ci-dessous.

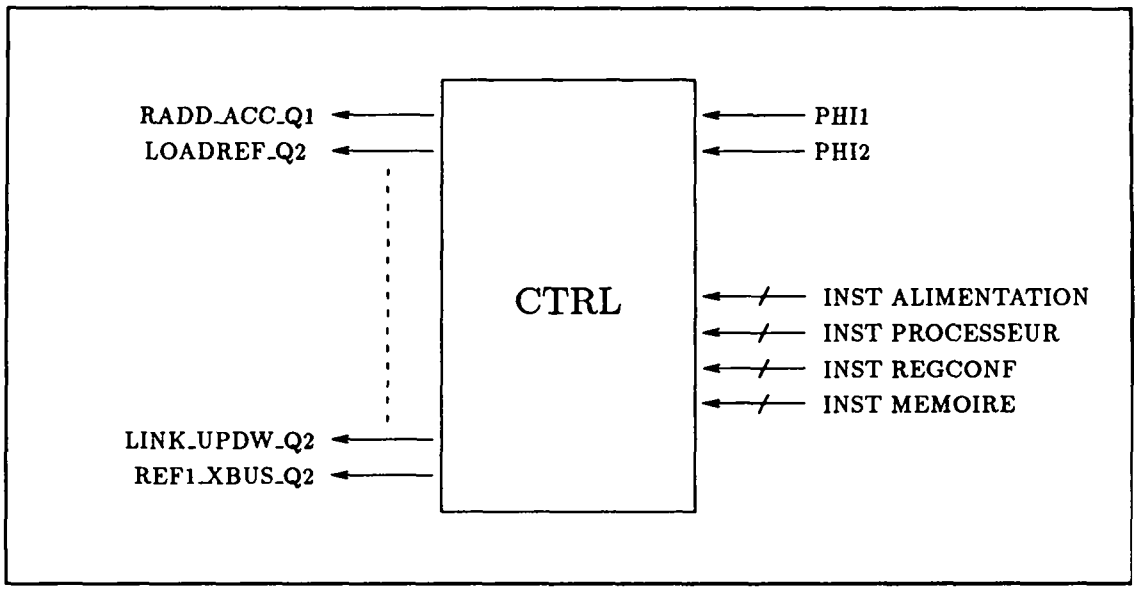


Figure 6.1 : module de contrôle

SEL_DIAG	ALIMENTATION	PROCESSEUR	MEMOIRE	REG_CONF

Ce codage autorise l'utilisation en parallèle des différents modules, ce qui permet de gérer simultanément le calcul sur le réseau de processeurs, l'alimentation de la mémoire, l'accès à la mémoire, etc.

6.2 Description fonctionnelle

Le module de contrôle est, en fait, un simple décodeur qui reçoit en entrée des instructions et qui génère, en sortie, les signaux de commande correspondants. Il a également la charge de les qualifier en fonction de la phase où ils sont actifs puis de les amplifier.

Le circuit fonctionne avec une horloge à deux phases non recouvrantes : une instruction est acquise à chaque phase et décodée pendant cette phase ; elle est ensuite exécutée à la phase suivante. Ainsi une instruction notée *INST_Q2*, par exemple, est acquise pendant la phase PHI1 puis exécutée pendant la phase PHI2.

Le codage des différents champs est le suivant :

REG_CONF :	SHIFTCR_Q2	0	0	1								
	CR_XBUS_Q2	0	1	0								
	CR_DBUS_Q2	0	1	1								
	XBUS_CR_Q2	1	0	0								
	DBUS_CR_Q2	1	0	1								
MEMOIRE :	DBUS_RMEM_Q2	1	X	X								
	XBUS_RAM_Q2	X	1	X								
	RMEM_DBUS_Q2	X	X	1								
	REFRESH_Q2	0	X	X								
PROCESSEUR :	DBUS_RCTEk_Q2	X	X	X	X	X	X	X	X	X	k	k
	IN_RIOk_Q2	X	X	X	X	X	X	k	k	k	X	X
	ABUS_OUTP_Q2	X	X	X	X	0	1	X	X	X	X	X
	BBUS_OUTP_Q2	X	X	X	X	1	0	X	X	X	X	X
	RD_OUTP_Q2	X	X	X	X	1	1	X	X	X	X	X
	RD_DBUS_Q2	X	X	X	1	X	X	X	X	X	X	X
	LINK_UPDW_Q2	X	0	1	X	X	X	X	X	X	X	X
	LINK_DIAG_Q2	X	1	0	X	X	X	X	X	X	X	X
	DBUS_RMSK_Q2	1	X	X	X	X	X	X	X	X	X	X
	RCTEk_BBUS_Q1	X	X	X	X	X	X	X	X	X	k	k
	RIOk_ABUS_Q1	X	X	X	X	X	X	k	k	k	X	X
	RADD_ACC_Q1	X	X	X	X	X	1	X	X	X	X	X
	MIN_ACC_Q1	X	X	X	X	1	X	X	X	X	X	X
	MIN_RD_Q1	X	X	X	1	X	X	X	X	X	X	X
ALIMENTATION :	LOADREF_Q2	X	X	X	X	1						
	SHIFTREF_Q2	X	X	X	1	X						
	REFk_XBUS_Q2	k	k	k	X	X						

Le champ SEL_DIAG, quant à lui, permet de sélectionner une diagonale (ou ligne) de processeur parmi les 5. En effet, certaines instructions (DBUS_RCTEik_Q2, DBUS_RMSKi_Q2, RDi_DBUS_Q2) demandent de préciser la ligne de processeurs où cette opération doit s'exécuter.

Deux signaux actifs sur des phases différentes peuvent se partager le même codage, ce qui permet de réduire la largeur (en nombre de bits) de l'instruction.

6.3 Architecture

La figure 6.3 indique l'architecture du module de contrôle. Il est basé sur une structure classique de décodeur : les signaux à décoder (lignes verticales I0 à I8 sur le schéma) croisent les signaux de commandes (lignes horizontales) connectés à une de leurs extrémités au niveau logique 1. A l'intersection de ces lignes, une cellule de type 0 ou de type 1 peut modifier l'état de la ligne horizontale en fonction des niveaux appliqués sur les entrées. De cette manière, une sortie est active lorsque son niveau logique est à 1.

Chaque sortie du décodeur est connectée à un qualifieur/amplificateur qui, suivant la phase en cours, prend en compte le décodage effectué pour activer ou non le signal de commande pendant la phase suivante. La figure 6.3 donne le schéma d'un qualifieur/amplificateur PHI2. Le schéma est identique pour un qualifieur de type PHI1, il

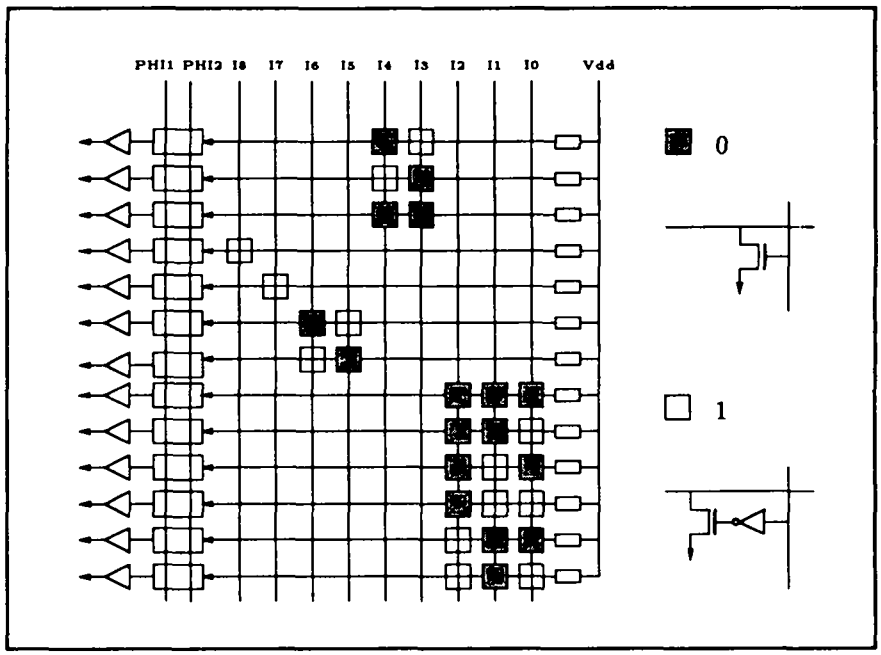


Figure 6.2 : architecture du module de contrôle

suffit d'inverser PHI1 et PHI2.

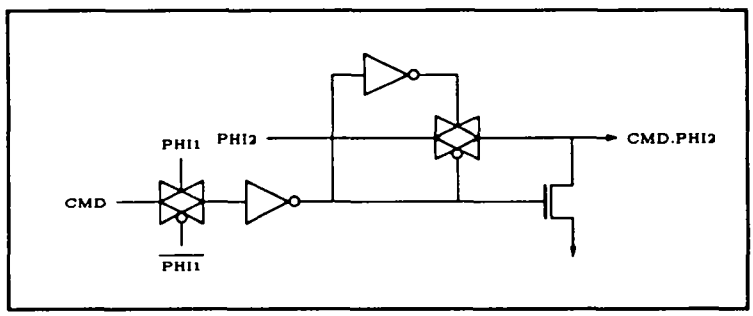


Figure 6.3 : qualifieur PHI2

6.4 Mise en œuvre VLSI

La structure régulière du module de contrôle induit directement l'implémentation VLSI. La figure 6.4 indique le plan de masse. La partie décodeur est composée d'une matrice de cellules de trois types, les cellules 0, les cellules 1 et les cellules vides. Ces dernières assurent seulement la continuité des connexions des lignes horizontales et verticales.

Le bord droit du décodeur est en aboutement avec une colonne de cellules *pull-up* maintenant, par défaut, les lignes horizontales à un niveau logique 1. Le bord gauche est connecté à une colonne de qualifieurs/amplificateurs de type PHI1 ou PHI2 suivant les signaux de commande à générer.

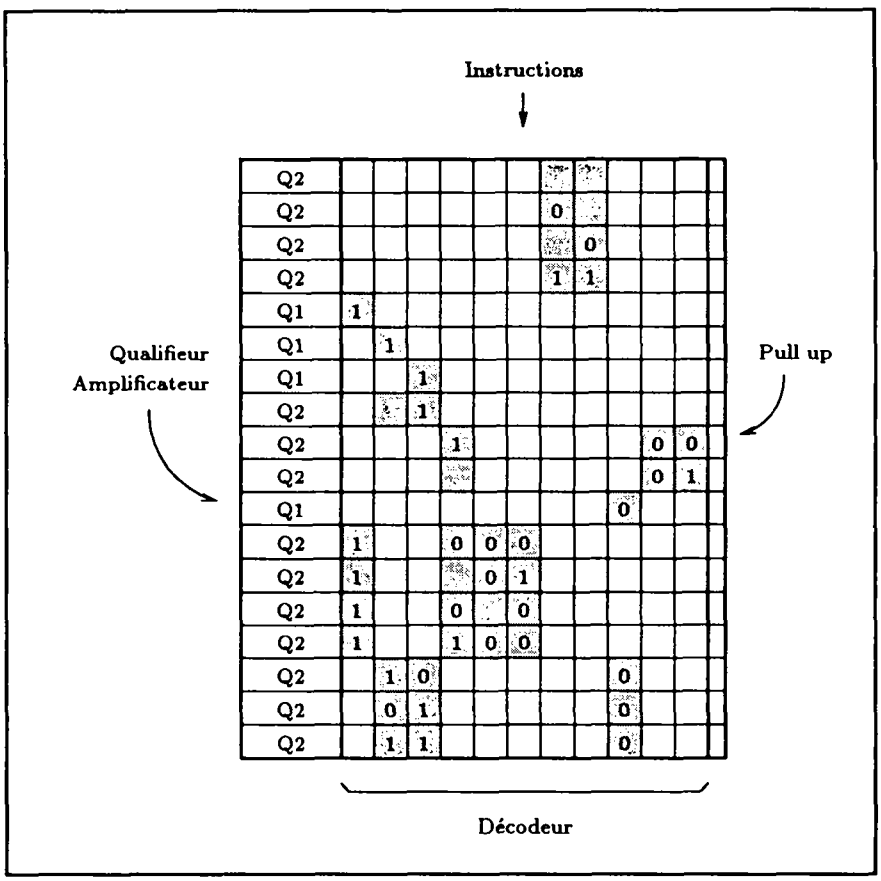


Figure 6.4 : plan de masse du module de contrôle

6.5 Génération du dessin de masque

Comme pour les précédents modules, les caractéristiques topologiques régulières de ce module permettent une génération automatique du dessin de masque sous forme de tableau de cellules. Les paramètres à prendre en compte regroupent l'ensemble des paramètres qui interviennent pour la construction des autres modules puisque le nombre et la natures des commandes à prévoir est fonction de ceux-ci, soit :

- le nombre de colonnes de processeurs,
- le nombre de lignes (ou nombre de diagonales) de processeurs,
- la largeur du chemin de données,
- le nombre de registres d'E/S par processeur,
- le nombre de registres de constantes par processeur,
- la taille de la mémoire,
- le nombre d'entrées référence.

A cet ensemble de paramètres vient s'ajouter le codage effectif de chaque commande. Ce codage est décrit explicitement dans le fichier de description (cf annexe A).

Chapitre 7

Dessin de masque

7.1 Introduction

D'un point de vue fonctionnel, le circuit API69 se compose de 5 modules distincts, le réseau de processeurs, la mémoire, le réseau d'alimentation, le registre de configuration et le module de contrôle. Chacun d'eux à un rôle bien particulier et peut être spécifié indépendamment des autres.

Cette répartition est également conservée au niveau des dessins de masque comme l'ont montré les chapitres précédents. De plus, les propriétés de régularité de chaque module permettent d'envisager leur construction à partir de structures simples, des tableaux de cellules.

Le dessin de masque du circuit complet consiste donc à assembler 5 tableaux de cellules. Mais il est clair que pour conserver une certaine *régularité* sur l'ensemble du circuit cet assemblage doit également présenter des aspects réguliers. La meilleure manière est d'abouter directement les tableaux de cellules de façon à concevoir un dessin de masque tel que celui représenté sur le schéma de la figure 7.1.

Ce schéma d'interconnexion implique cependant un certain nombre de contraintes sur chaque module et donc sur les cellules de base les constituant. La section suivante présente le squelette type d'une cellule de base et détaille les différentes contraintes qui lui sont associées.

7.2 Cellules de base

L'ensemble du circuit n'étant qu'un assemblage de cellules, celles-ci doivent respecter un certain nombre de règles topologiques de manière à ce qu'elles puissent s'abouter sans problème. De plus, on retrouve les mêmes fonctions dans plusieurs endroits du circuit et le fait de s'imposer des contraintes, lors de l'édition du dessin de masque des cellules réalisant ces fonctions, permet de les réutiliser plusieurs fois sans modification.

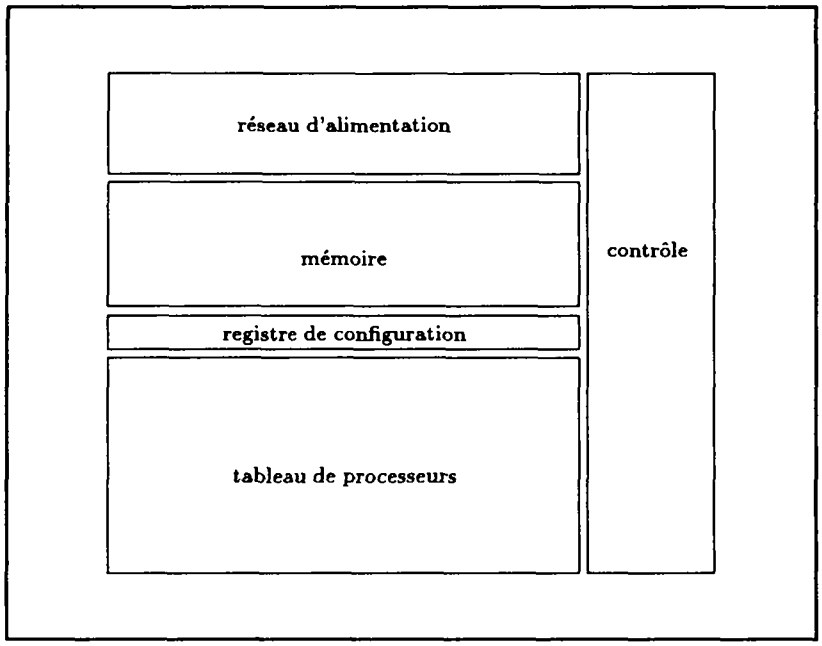


Figure 7.1 : dessin de masque (principe)

La figure 7.2 indique les contraintes qui ont été fixées dès le départ sur le dessin de masque des cellules de base. On en distingue plusieurs types : les contraintes sur les alimentations, les contraintes sur le positionnement des fils d'horloge, les contraintes sur les signaux de commandes, les contraintes sur la largeur des cellules et les contraintes sur la hauteur des cellules.

Contraintes sur les alimentations

Deux fils d'alimentation traversent horizontalement chaque cellule, un fil vehiculant la tension d'alimentation (Vdd) et l'autre la masse (Gnd). Au milieu de chacun de ces deux fils est placé un contact de polarisation. Ces fils ont une largeur de 4μm et sont en métal 1.

Le fait d'avoir des fils d'alimentation de largeur fixe et un contact positionné exactement au milieu de ces fils est une condition nécessaire pour que 2 cellules de nature différente placées l'une au dessus de l'autre puissent partager la même ligne d'alimentation.

Contraintes sur les horloges

Le circuit est piloté par une horloge à deux phases non recouvrantes, notées PHI1 et PHI2 qui sillonnent le circuit horizontalement et traversent les cellules qui en ont besoin.

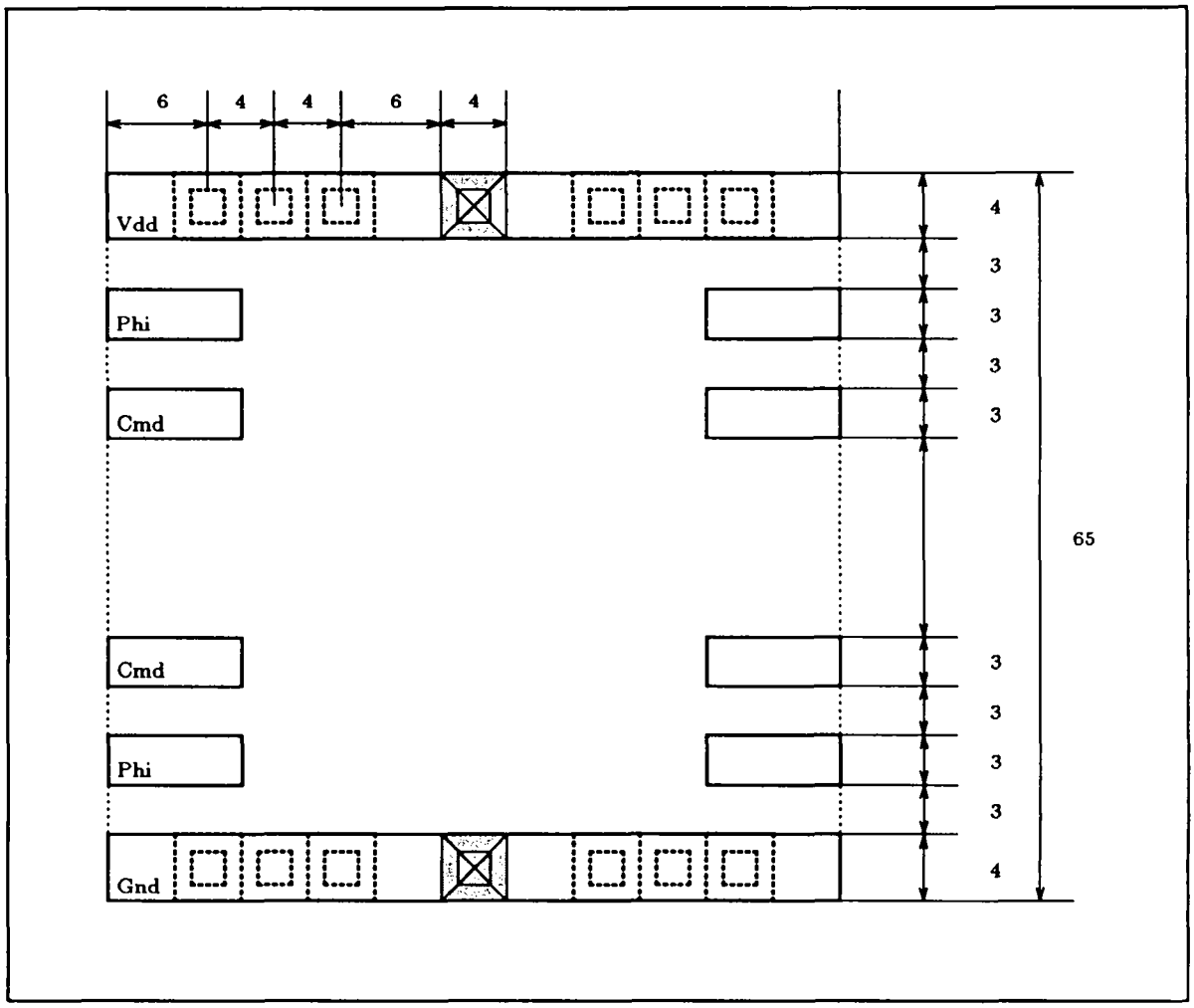


Figure 7.2 : contraintes sur le dessin de masque d'une cellule

Leur complément est également utilisé dans certaines cellules. Pour qu'une connexion automatique des horloges puisse se réaliser simplement (par aboutement de cellules de connexion en bordure des blocs), leur position, leur largeur et leur nombre ont été fixés. Ainsi, on dispose dans chaque cellule de deux signaux d'horloge qui peuvent être n'importe quelle combinaison des 4 horloges disponibles, PHI1, $\overline{\text{PHI1}}$, PHI2, $\overline{\text{PHI2}}$. Si une cellule n'utilise pas d'horloge, l'espace réservé aux fils d'horloge peut être utilisé autrement. Les fils d'horloge sont en métal 1.

Contraintes sur les commandes

Chaque cellule peut recevoir, au maximum, deux signaux de commande produit par le module de contrôle. Ils sont en métal 1, traversent la cellule horizontalement et ont une largeur et une position fixée. Cette contrainte permet de ne dessiner qu'un seul type de cellule dont la fonction est de qualifier et d'amplifier la commande après décodage. Comme pour les signaux d'horloge, si une cellule n'a pas besoin de signaux de commande, l'espace réservé peut être utilisé autrement.

Contraintes horizontales

La *largeur* de la cellule n'a pas une dimension fixée. Elle doit simplement respecter quelques règles qui consistent à pouvoir placer des contacts à des positions bien précises par rapport au centre (contact de polarisation) et par rapport aux bords de la cellule (cf figure 7.2). En relation avec les contraintes liées aux fils d'alimentation cela permet que deux cellules différentes (de même largeur) placées l'une au dessus de l'autre puissent partager le même fil d'alimentation.

Contraintes verticales

La hauteur de toutes les cellules est de 65 μm. Cette contrainte, relativement stricte, permet d'utiliser la même cellule pour générer les signaux de commande à l'ensemble du circuit.

7.3 Interconnexion des modules

En respectant les contraintes énoncées au paragraphe précédent et en fixant une largeur de cellule pour pour celles qui interviennent dans la composition du réseau d'alimentation, du réseau de processeur, du registre de configuration et de la mémoire, l'assemblage de ces derniers modules est immédiat : ces quatre modules s'empilent verticalement et doivent avoir le même leur base. Cela se traduit directement, au niveau d'une cellule de base, par une largeur identique de toutes les cellules.

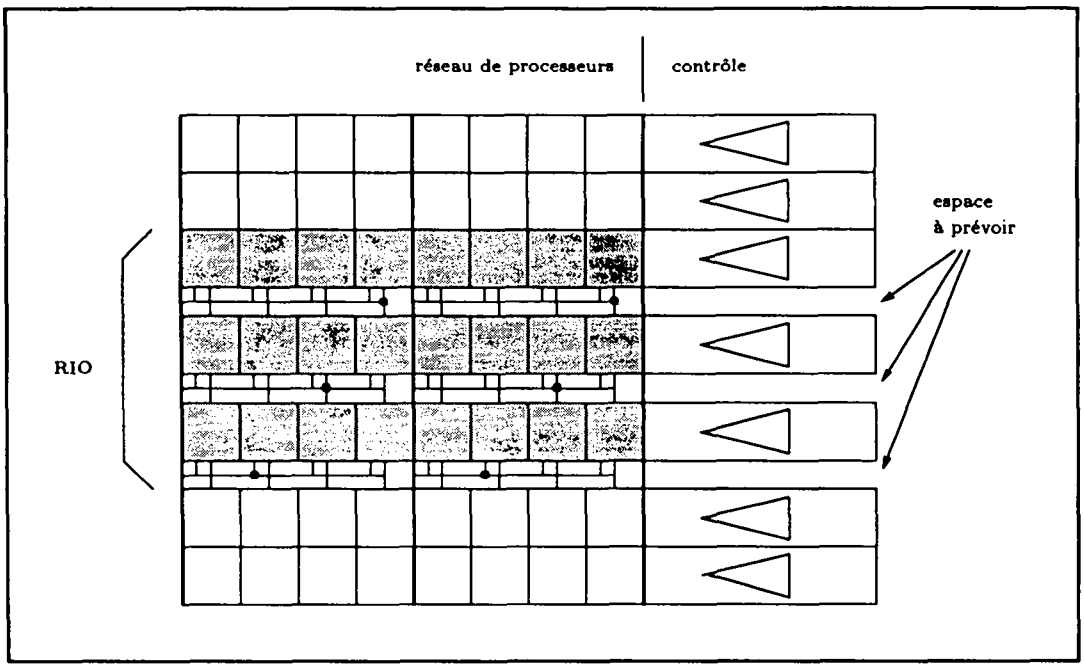


Figure 7.3 : connexion registres E/S et module de contrôle

La connexion de cet ensemble avec le module de contrôle requiert un peu plus d'attention. Plusieurs entorses à l'apparente régularité du circuit viennent contrarier la contrainte verticale imposée aux cellules de base.

La première provient des registres d'entrées/sorties des processeurs (cf section 2.4.2 page 33). D'un point de vue topologique, ces registres ne peuvent pas s'abouter directement verticalement. Un espace doit être prévu pour laisser passer un fil de métal permettant une connexion avec le registre de masquage. Ce même espace doit être conservé dans le module de contrôle (cf figure 7.3).

Une autre irrégularité sur le module de contrôle est provoquée par le registre de configuration. Celui-ci, rappelons le, nécessite un routage permettant de cascader les 15 étages qui composent le registre à décalage. Ce routage est réalisé par un assemblage de 8 cellules connectées sur le bord inférieur des cellules de base (cf figure 5.4 page 65). De plus, ces connexions doivent traverser horizontalement le module de contrôle pour récupérer, en bordure du cœur de la puce, la sortie du registre de configuration.

Le réseau d'alimentation présente également des routages de même type qui font que les cellules réalisant les différents registres à décalage ne sont pas directement aboutées verticalement les une avec les autres.

Dans tous ces cas, les cellules d'amplification du module de contrôle ainsi que toutes les cellules situées sur le même axe horizontale ne peuvent pas être aboutées directement avec leur voisine verticale. Il faut aménager un espace et prévoir les connexions

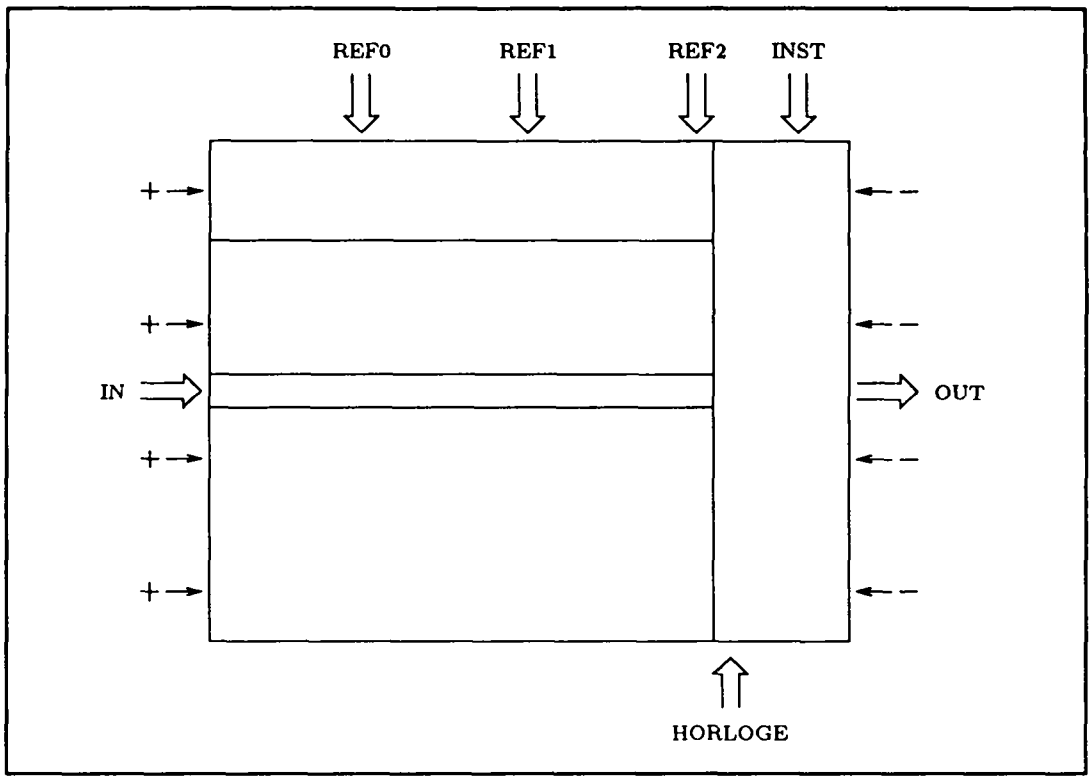


Figure 7.4 : dessin de masque

verticales qui sont normalement réalisées lorsque les deux cellules sont juxtaposées.

7.4 Plan de masse

La connexion des alimentations électriques aux cellules est immédiate : chaque cellule étant traversée horizontalement par deux fils d'alimentation (VDD et GND) et chaque cellule étant en aboutement directe avec ses voisines, l'alimentation électrique peut se faire par les bords. Ainsi toutes les cellules du bord *droit* sont connectées à GND et toutes les cellules du bord *gauche* sont connectées à VDD. Deux fils verticaux sont donc nécessaires des deux cotés du cœur du circuit.

La connexion des horloges (PHI1, $\overline{\text{PHI1}}$, PHI2, $\overline{\text{PHI2}}$) est plus délicate dans la mesure où toutes les cellules n'utilisent pas les mêmes phases (deux sont au plus disponibles par cellule). Les connexions sont réalisées dans le module de contrôle qui génère, en même temps que les commandes, les phases de l'horloge utilisées par une *ligne* de cellules.

La figure 7.4 donne le plan de masse du cœur du circuit API69 et indique les connexions relatives aux entrées/sorties.

Chapitre 8

Conception

8.1 Introduction

Le circuit API69 se compose de 5 modules principaux qui, comme nous l'avons vu dans les chapitres précédents, peuvent être implémentés sur silicium de manière extrêmement régulière.

Le but de ce chapitre est de montrer comment cette régularité est exploitée pour produire, dans les meilleurs conditions possibles, un tel circuit. En effet, le nombre de transistors contenu sur cette puce limite considérablement les approches de conception traditionnelles, d'une part à cause des outils de CAO disponibles pour réaliser des circuits *full custom* et, d'autre part, à cause des capacités des machines qui les supportent.

Le nombre de transistors du circuit API69 avoisine les 300 000. Devant ce nombre d'éléments, il n'est plus concevable, par exemple d'effectuer une comparaison entre le dessin de masque complet et le schéma électrique ayant servi à la simulation afin de s'assurer qu'il y a correspondance entre les deux. Cette étape de vérification demande beaucoup de temps et doit être souvent réitérée pendant la conception d'un circuit relativement complexe, au niveau des cellules de base, des modules ou du circuit complet. A titre d'exemple, la comparaison d'un schéma électrique d'une dizaine de milliers de transistors avec le dessin de masque correspondant peut demander quelques heures et requérir quelques dizaines de M octets d'espace mémoire.

La suite du chapitre explique quelle a été la démarche suivie pour concevoir le circuit API69. Elle fait appel à la notion de *noyau* d'un circuit représentant, en quelque sorte, une version minimale du circuit final mais conservant toutes ses propriétés. Cette notion est approfondie dans la section suivante. Les sections 8.3 et 8.4 montrent respectivement l'impact immédiat de cette approche sur la simulation et la vérification.

8.2 Noyau d'un circuit

La méthode suivie pour concevoir le circuit API69 s'appuie avant tout sur les deux constatations suivantes :

1. d'un point de vue dessin de masque, le circuit API69 n'est qu'un simple tableau de cellules,

2. le nombre de transistors interdit une conception traditionnelle, notamment pour les phases de vérification.

La première remarque suggère une construction automatique du dessin de masque car, bien que ce ne soit qu'un tableau, le nombre de cellules à assembler est important. Du second point est née l'idée de ne faire porter l'étude que sur un sous-ensemble du circuit afin de limiter le plus possible le temps de conception tout en garantissant, bien sûr, la production d'un circuit *zéro défaut.*

La notion de noyau repose à la fois sur l'aspect automatisation et sur l'aspect *modèle réduit* du circuit. L'idée est la suivante : à partir d'un générateur paramétrable, deux dessins de masque du circuit sont produits. Le premier représente le circuit final tandis que le second en est une réduction du premier respectant toutes ses propriétés. Le respect de ces dernières porte à la fois sur l'aspect topologique et sur l'aspect fonctionnel du circuit.

Les propriétés liées à la topologie du circuit concernent l'aboutement des cellules les unes par rapport aux autres. Il est clair, par exemple, que la vérification du respect des règles de dessin d'une matrice de N x N cellules ne nécessite pas une vérification complète de la matrice mais seulement une matrice réduite à 2 x 2. Sur des structures plus complexes, faisant intervenir plusieurs types de cellules, il faut s'assurer que toutes les configurations des aboutements des cellules les unes par rapport aux autres se retrouvent dans le noyau. La figure 8.1 montre un exemple de noyau pouvant être déduit d'une structure à plusieurs cellules.

D'un point de vue fonctionnel, les règles qui permettent de déduire un noyau à partir du circuit VLSI complet sont moins nettes et dépendent des vérifications envisagées. Dans le cas des mémoires associatives du circuit API69, par exemple, leur vérification fonctionnelle ne nécessite pas d'établir une structure comportant 10 couples de valeurs de 8 bits chacunes. Une mémoire de 2 ou 3 couples sur un chemin de données réduit à 2 ou 3 bits suffit pour valider fonctionnellement un tel dispositif.

Le noyau retenu pour le circuit API69 possède les caractéristiques suivantes :

- chemin de données sur 4 bits,

- mémoire associative de 3 couples,

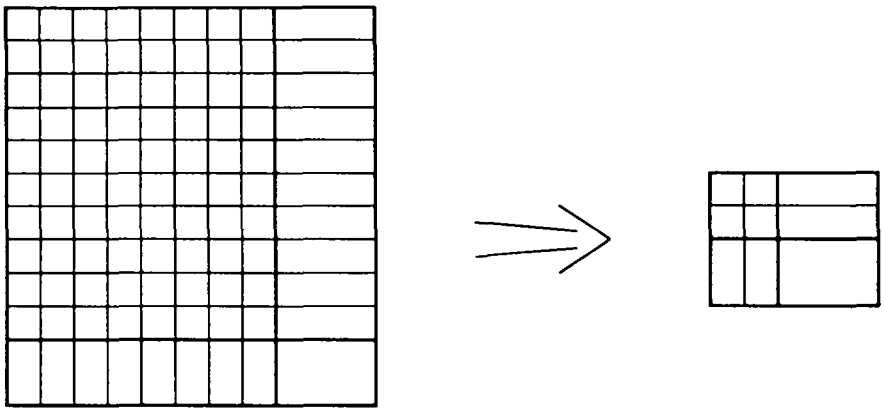


Figure 8.1 : exemple de noyau d'un circuit

- 6 colonnes de processeurs,
- 3 diagonales,
- 3 entrées référence,
- 3 registres d'entrées/sorties par processeur,
- 3 registres de constante par processeur.

Avec ces caractéristiques, l'implémentation de l'algorithme de programmation dynamique est tout à fait possible. Tous les cas de figure sont permis : comparaison de mots de longueurs différentes, propagation du résultat, etc. D'un point de vue strictement topologique, ce noyau aurait pu être encore plus réduit dans la mesure où 6 colonnes de processeurs ne sont pas indispensables pour vérifier toutes les combinaisons d'aboutement de cellules.

8.3 Simulation

La simulation a pour but de vérifier (partiellement) que l'architecture proposée est en accord avec les spécifications du circuit. Cette vérification s'effectue en plusieurs temps, d'abord au niveau fonctionnel afin de valider l'architecture générale, puis à un niveau beaucoup plus proche de la machine pour prendre en compte les détails techniques liés à la mise en œuvre. Les deux simulations doivent donner les mêmes résultats lorsqu'elles sont soumises aux mêmes stimuli d'entrée.

La figure 8.2 indique les outils développés et ceux qui ont été mis en œuvre pour simuler le circuit API69. Les objets en grisé indiquent une production automatique, soit de programmes, soit de fichiers.

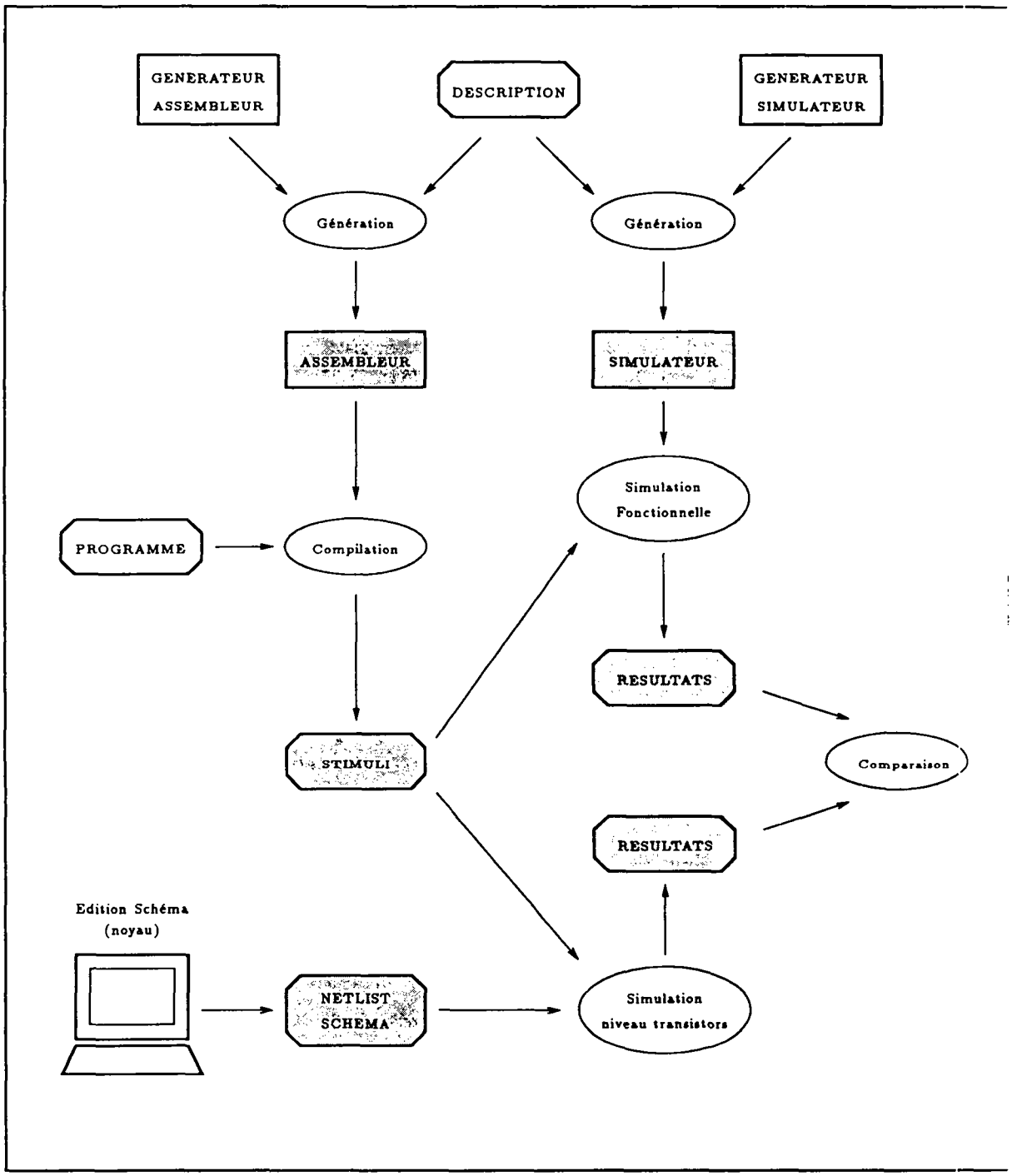


Figure 8.2 : chaîne de simulation

La simulation fonctionnelle fait appel à un simulateur écrit en langage C. Celui-ci prend en entrée un fichier de stimuli décrivant, dans le temps, les différentes instructions exécutées par le circuit. Le simulateur génère, ensuite, un fichier de données indiquant l'état de toutes les sorties du circuit.

En fait, le simulateur du circuit API69 est produit automatiquement par un *générateur de simulateur* à partir d'un fichier de description où sont précisées les caractéristiques du circuit API69 susceptible d'intervenir pour la constitution du noyau. Ainsi on trouve des paramètres tels que la largeur du chemin de données, le nombre de registres internes aux processeurs du réseau, la taille de la mémoire, la dimension du réseau, etc. Outre le fait de pouvoir créer très rapidement un simulateur (quelques minutes), on s'assure d'une correspondance fonctionnelle entre les simulateurs (on suppose que le programme *générateur* est correct!).

Les stimuli à présenter au simulateur représentent, en général, un volume de données conséquent, surtout si la simulation porte sur l'exécution complète d'un algorithme. L'écriture manuelle d'un tel fichier de stimuli est quasi impossible sans erreurs et demande, de toute façon, beaucoup trop de temps. Un outil de génération de stimuli s'avère donc indispensable. Un générateur d'assembleur a également été développé; il permet de décrire dans un *langage* (!) de plus haut niveau les opérations à effectuer par le circuit et produit le code binaire correspondant. Cet assembleur intègre, de plus, quelques macro-instructions spécifiques à l'environnement de simulation pour générer des données aux entrées du circuit. L'assembleur est, bien sûr, généré à partir du même fichier de description que le simulateur.

La première étape dans la conception du circuit API69 a donc consisté à générer le simulateur et l'assembleur correspondant à la version finale du circuit et à le tester dans un environnement adéquat.

Dans un second temps, le simulateur et l'assembleur du noyau ont été générés. Parallèlement, une représentation schématique du noyau a été saisie avec l'outil de CAO CADENCE, au niveau transistor.

Des programmes de test ont ensuite été soumis aux deux simulateurs (simulateur fonctionnel et simulateur niveau transistor du noyau) pour vérifier que les résultats étaient identiques. Pour éviter toutes erreurs de transcription au niveau des stimuli d'entrée, le simulateur fonctionnel accepte le même format d'entrée que le simulateur utilisé avec CADENCE (SILOS).

La constitution de cette chaîne de simulation a permis de réduire de façon notable les temps de simulation et d'analyse des résultats. Le gain de temps a porté sur :

- la saisie du schéma avec l'outil de CAO : en effet, la dimension réduite du noyau demande la connexion d'un nombre limité de cellules de base. Ce gain a surtout été mis en évidence lors des *retouches* du schéma, étapes inévitables pendant les phases de mise au point.

- la simulation niveau transistor: il est clair que le rapport entre le nombre de

transistors du noyau et celui du circuit complet se répercute directement sur le temps de simulation.

Le temps passé à mettre au point les générateurs de simulateurs et d'assembleurs n'est pas négligeable et a demandé un certain investissement. On peut estimer le coût (en temps) d'un générateur à deux fois celui du programme qu'il produit.

8.4 Dessin de masque

Le dessin d'un masque d'un circuit conçu de manière *full custom* s'accompagne toujours de deux importantes étapes de vérification : le respect des règles de dessin et la correspondance avec le schéma électrique. Ces deux étapes sont d'autant plus longues que le circuit est complexe (en nombre de transistors).

Notre démarche consiste à produire automatiquement le dessin de masque final et de ne faire porter les vérifications mentionnées ci-dessus que sur le noyau, construit, lui aussi, de manière automatique à partir du même générateur de dessin de masque

La figure 8.3 indique les différentes étapes nécessaires à la production du dessin de masque : un générateur produit à partir d'un fichier de description (le même que celui nécessaire aux générateurs de simulateurs et d'assembleurs) une structure intermédiaire décrivant la position relative de toutes les cellules du circuit les unes par rapport aux autres.

Ce fichier est compréhensible par un utilitaire de CADENCE, "Make-Array", qui, en fonction d'un représentation simplifiée (construite au préalable), assure la correspondance avec la base de données où sont stockées les cellules. Cette représentation topologique simplifiée est complètement indépendante de la taille du circuit et du nombre de cellules. Elle précise simplement la position relative des blocs, tels que le réseau de processeurs, la mémoire, etc. Le circuit API69 n'étant rien d'autre qu'un tableau de cellules, il peut ainsi être généré directement avec la fonction "Make-Array".

La vérification des règles de dessin consiste à générer le dessin de masque du noyau, puis à lancer le vérificateur (VRD) sur cet ensemble. S'il n'y a pas d'erreurs et que tous les aboutements de cellules présents dans le circuit final se retrouvent dans le noyau, alors le dessin de masque final (produit de la même façon que le noyau) sera sans erreurs de dessin.

De la même manière, la comparaison du dessin de masque avec le schéma électrique s'effectue sur le noyau. Deux *netlist* sont construites, une en provenance du schéma électrique du noyau et l'autre à partir du dessin de masque du noyau. Si elles concordent, alors le dessin de masque final correspondra à la description architecturale qui en a été faite.

L'utilisation de la notion de noyau a été déterminante durant cette phase de conception. Sans cette démarche, le circuit n'aurait pas pu être vérifié de manière satisfaisante. Pour fixer les idées, la comparaison d'un dessin de masque avec un schéma électrique

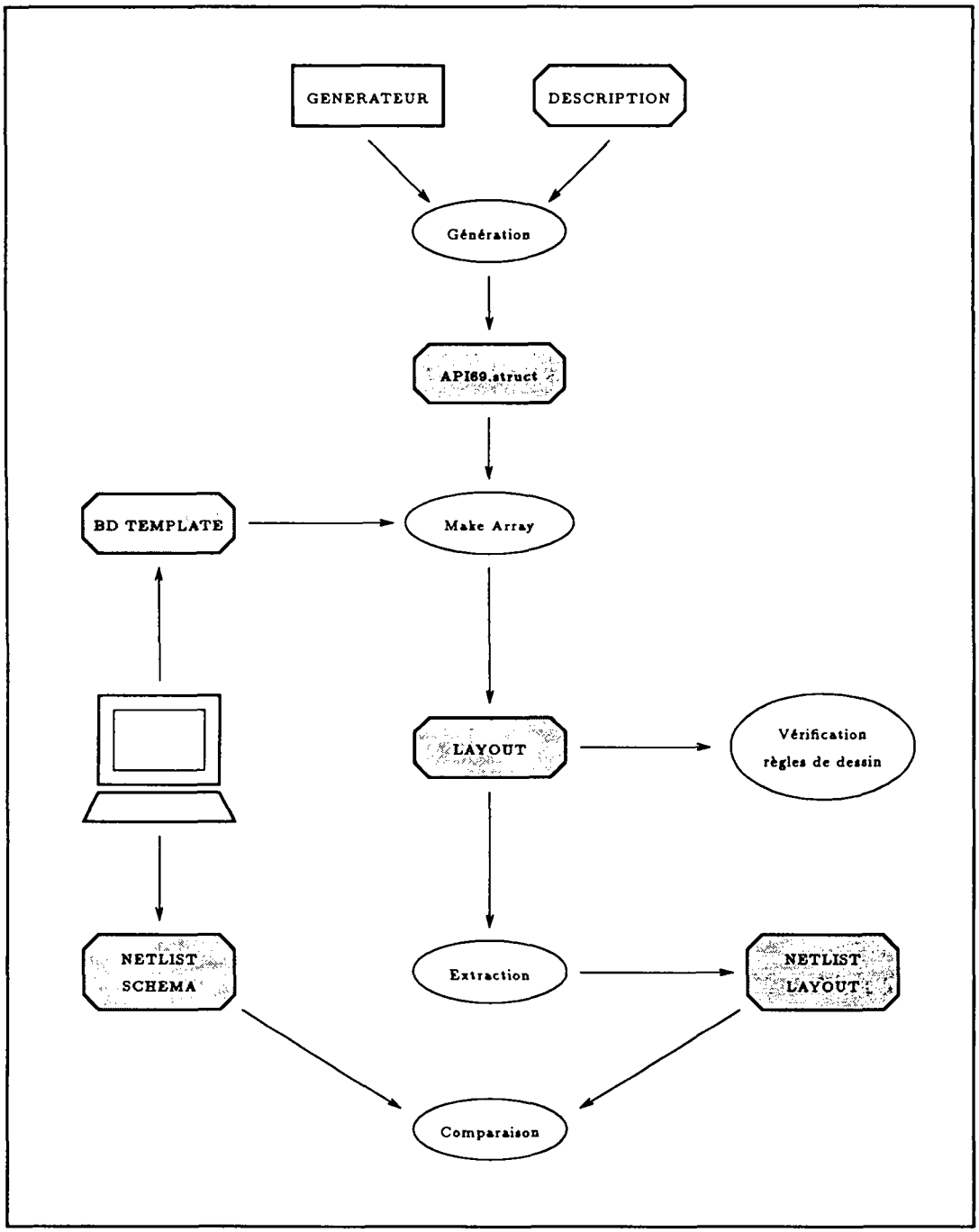


Figure 8.3 : production du dessin de masque

sur un circuit (ou un module) d'environ 10 000 transistors prend de 4 à 6 heures avec les outils de CADENCE sur une station de travail de type SUN 3.

8.5 Conclusion

La conception du circuit API69 s'est faite entièrement à partir d'un noyau, tant au niveau simulation que vérification. Cette approche a demandé la réalisation de 3 générateurs écrits en C qui, à partir d'un même fichier de description, produisent :

- un simulateur,
- un assembleur,
- un fichier d'entrée pour l'utilitaire "Make-Array".

Cette approche a également demandé l'utilisation d' un outil de CAO VLSI permettant la conception de circuits de manière *full custom*. Avec cet outil ont été constitués :

- une bibliothèque de cellules (schéma + dessin de masque)
- le schéma électrique du noyau (saisie graphique),
- le plan de masse du circuit (saisie graphique, Make-Array).

Chapitre 9

Programmation

9.1 Introduction

Ce chapitre traite de la programmation du circuit API69. Ce circuit étant dévolu au calcul de la distance entre deux chaînes de caractères nous prendrons, tout au long de ce chapitre, un exemple simplifié de la correction de fautes de frappe. L'algorithme de programmation dynamique utilisé pour cette présentation ne fait intervenir que trois types de faute, les fautes d'insertion, les fautes d'omission et les fautes de substitution. L'équation de récurrence associée est donnée par :

$$D(i,j) = Min \begin{cases} D(i-1,j-1) + Tsub(r_i, t_j) \\ D(i-1,j) + Ko \\ D(i,j-1) + Ki \end{cases} \tag{9.1}$$

avec les valeurs d'initialisation suivantes :

$$D(0,0) = 0, \quad D(i,0) = D(i-1,0) + Ki, \quad D(0,j) = D(0,j-1) + Ko$$

La comparaison d'un mot erroné avec une suite de références issues d'un dictionnaire se décompose en deux phases :

- une phase d'initialisation,
- une phase de calcul.

La phase d'initialisation consiste à charger les mémoires en fonction des caractères du mot erroné. Elle consiste également à initialiser les registres des processeurs avec les constantes Ki et Ko et à configurer correctement les registres d'entrée/sorties des processeurs périphériques.

La phase de calcul déroule $N+28$ cycles systoliques (cf page ??) si N est le nombre de références à comparer avec le mot erroné. Dans un cycle systolique, 3 actions se déroulent en parallèle, le calcul proprement dit, l'accès à la mémoire et l'acquisition des références.

Le processus de comparaison complet peut être décrit par la suite d'événements suivants :

```
begin

  /* initialisation */

  <chargement des memoires>
  <initialisation RE/S>
  <initialisation RCTE>

  <CS acquisition reference>
  <CS acquisition reference>, <CS acces memoire>

  /* calcul */

  for (i=0;i<n+28;i++)
   {
     <CS acquisition reference>, <CS acces memoire>, <CS calcul>
   }

end
```

CS indique un Cycle Systolique qui dure 6 cycles machines. En optimisant un temps soit peu la mise en œuvre de l'algorithme pris comme exemple, le nombre de cycles pourrait être réduit. Notre but, dans ce chapitre, cherche plutôt à montrer simplement comment l'ensemble d'une application peut être programmé. L'optimisation n'apporterait pas d'information supplémentaire.

Tout ce qui est sur une même ligne, dans le programme ci-dessus, est effectué en parallèle. La phase d'initialisation demande, avant d'entamer le calcul, un cycle systolique <accès mémoire> pour charger, un cycle systolique à l'avance, les coefficients de substitution dans les registres des processeurs de manière à ce qu'ils puissent l'utiliser immédiatement. Pour que cette opération puisse s'effectuer, il faut que le réseau d'alimentation ait d'abord été initialisé. C'est pourquoi, deux cycles systoliques <acquisition référence> sont nécessaires pendant la phase d'initialisation.

Les sections suivantes détaillent respectivement les phases d'initialisation et de calcul en donnant pour chacunes d'elles la suite d'instructions API69 permettant de réaliser ces fonctions.

9.2 Chargement des mémoires

Le processus d'initialisation des mémoires consiste à envoyer au circuit 21 × 15 valeurs via le port d'entrée du registre de configuration. Chaque fois que 15 valeurs ont été reçues la commande de mémorisation est actionnée. Le programme s'écrit :

```
/* ------------------------------------------------------------------- */
/* CHARGEMENT DES MEMOIRES                                             */
/* ------------------------------------------------------------------- */

for (i=0;i<10;i++)
 {
   for (j=0;j<15;j++)
    {
      SHIFTCR                          /* acq. 15 caracteres       */
    }
   CR_DBUS, DBUS_RMEM                  /* memorisation             */
   for (j=0;j<15;j++)
    {
      SHIFTCR                          /* acq. 15 couts            */
    }
   CR_DBUS, DBUS_RMEM                  /* memorisation             */
 }
for (j=0;j<15;j++)
 {
   SHIFTCR                            /* acq. 15 couts par defaut */
 }
CR_DBUS, DBUS_RMEM                    /* memorisation             */

/* ------------------------------------------------------------------- */
```

La mémoire contient 10 couples (caractère-coût) plus un coût par défaut, soit 21 mots.

9.3 Initialisation des registres d'entrées/sorties

Les registres d'entrées/sorties sont pourvus d'un mécanisme permettant d'inhiber leur fonction d'écriture, ceci pour gérer les problèmes d'acquisition de valeurs sur les processeurs situés sur la périphérie du réseau. La première étape consiste à autoriser l'écriture dans tous les registres d'entrées/sorties en positionnant à la valeur 1 tous les bits des registres de masquage. Cette opération effectuée, on initialise tous les registres d'entrées/sorties par décalages successifs, de la première rangée de processeurs vers la

dernière. La dernière phase positionne les bits du registre de masquage des processeurs en fonction de leurs situations topologiques sur le réseau.

```
/* ------------------------------------------------------------------ */
/* INITIALISATION RIO                                                 */
/* ------------------------------------------------------------------ */

for (i=0;i<15;i++)
 {
   SHIFTCR                                          /* RMSK := 0xFF  */
 }
for (i=1;i<=5;i++)
 {
   CR_DBUS, DBUS_RMSKi
 }

 for (k=0;k<8;k++)                        /* initialisation RI00 a RI07 */
 {
   for (i=0;i<5;i++)
    {
      for (j=0;j<15;j++)
       {
         SHIFTCR
       }
      CR_DBUS, LINK_UPDW, IN_RIOk, RIOk_ABUS, ABUS_OUTP
    }
 }

for (j=1;j<=5;j++)                /* positionnement des registres RMSK */
 {
   for (i=0;i<15;i++)
    {
      SHIFTCR
     }
   CR_DBUS, DBUS_RMSKj
 }

/* ------------------------------------------------------------------ */
```

9.4 Initialisation des registres de constantes

Deux constantes, Ki et Ko, doivent être chargées dans les registres de constante des processeurs avant de commencer le calcul. En fait, tous les processeurs ne mémorisent pas les mêmes constantes: Ki et Ko ont une valeur spécifique suivant la nature du

calcul effectué par les processeurs. Certains ne participent pas directement au calcul de distance mais ont simplement la charge de propager le résultat vers la périphérie du réseau (cf paragraphe 1.3.4). Le programme est le suivant :

```
/* ------------------------------------------------------------------ */
/* INITIALISATION RCTE                                                */
/* ------------------------------------------------------------------ */

for (i=1;i<=5;i++)                  /* pour chaque ligne de processeurs */
 {
   for (j=0;j<15;j++)
    {
      SHIFTCR                        /* acquisition sequentielle de Ki */
    }
   CR_DBUS, DBUS_RCTEi2                                  /* Ki -> RCTE2 */
 }

for (i=1;i<=5;i++)                  /* pour chaque ligne de processeurs */
 {
   for (j=0;j<15;j++)
    {
      SHIFTCR                        /* acquisition sequentielle de Ko */
    }
   CR_DBUS, DBUS_RCTEi3                                  /* Ko -> RCTE3 */
 }

/* ------------------------------------------------------------------ */
```

9.5 Cycle systolique de calcul

Un cycle systolique de calcul doit réaliser les opérations décrites par l'équation 9.1. En terme de programmation, cela signifie qu'un processeur doit acquérir trois valeurs, D(i-1,j-1), D(i,j-1) et D(i-1,j), et propager son résultat à trois processeurs voisins. L'algorithme s'écrit :

```
IN        (Dv,Dd,Dh)
Rv    =   Dv + Ko
Rh    =   Dh + Ki
Rd    =   Dd + Kc
D     =   (Rv < Rh) ? Rv : Rh
D     =   (D < rd) ? D : Rd
OUT       (D,D,D)
```

Dv symbolise la distance D(i-1,j), Dh la distance D(i,j-1), Dd la distance D(i-1,j-1), Ki représente le cœfficient d'insertion, Ko le cœfficient d'omission et Kc le cœfficient de substitution (on suppose que celui-ci est déterminé par l'accès à une table, en parallèle avec le calcul en cours). Le pipeline du calcul implique l'ajout d'un délai sur la propagation "diagonale" du résultat vers le processeur (i+1,j+1). Cette connexion diagonale peut être réalisée en deux temps, d'abord par une connexion verticale, puis par une connexion horizontale. L'algorithme s'écrit alors :

```
IN        (Dv,Dd,Dh)
Rv    =   Dv + Ko
Rh    =   Dh + Ki
Rd    =   Dd + Kc
D     =   (Rv<Rh) ? Rv : Rh
D     =   (D<rd) ? D : Rd
OUT       (D,Dh,D)
```

ce qui est équivalent à (cf figure 2.4 page 32) :

```
IN        (Dv,Dd,Dh)
RADD  =   Dv + Ko
ACC   =   RADD
RADD  =   Dh + Ki
ACC   =   MIN (ACC,RADD)
RADD  =   Dd + Kc
RD    =   MIN (ACC,RADD)
OUT       (RD,Dh,RD)
```

En supposant que plusieurs opérations puissent se dérouler en parallèle, la suite d'instructions se réécrit :

```
IN        (Dv,Dd,Dh)
RADD =    Dv + Ko
RADD =    Dh + Ki      |  ACC  = RADD
RADD =    Dd + Kc      |  ACC  = MIN (ACC,RADD)
                          RD   = MIN (ACC,RADD)
OUT       (RD,Dh,RD)
```

De même, en parallélisant les entrées/sorties on obtient :

```
RADD = Dd + Kc
RADD = Dv + Ko  |  ACC  = RADD
RADD = Dh + Ki  |  ACC  = MIN (ACC,RADD)  |  OUT(Dh) IN(Dd)
                   RD   = MIN (ACC,RADD)  |  OUT(MIN(ACC,RADD)) IN (Dv
                                             OUT(RD) IN(Dh)
```

En régime permanent, un cycle systolique se compose de cinq *sous-cycles* représentant 5 cycles machines. Les commandes activées pendant ce cycle systolique sont les suivantes :

```
cycle machine 1   RIO1_ABUS       ; RIO[Dd] -> ABUS
                  RCTE1_BBUS      ; RCTE[Kc] -> BBUS
                                  ; RADD := Dd + Kc

cycle machine 2   RIO2_ABUS       ; RIO[Dv] -> ABUS
                  RCTE3_BBUS      ; RCTE[Ko] -> BBUS
                                  ; RADD := Dv + Ko
                  RADD_ACC        ; ACC := RADD

cycle machine 3   RIO3_ABUS       ; RIO[Dh] -> ABUS
                  RCTE2_BBUS      ; RCTE[Ki] -> BBUS
                                  ; RADD := Dh + Ki
                  MIN_ACC         ; ACC := MIN (ACC,RADD)
                  ABUS_OUTP       ; OUT (Dh)
                  INP_RIO1        ; IN (Dd)

cycle machine 4   MIN_RD          ; RD := MIN (ACC,RADD)
                  RD_OUTP         ; OUT (MIN (ACC,RADD))
                  INP_RIO2        ; IN (Dv)

cycle machine 5   RD_OUTP         ; OUT (RD)
                  INP_RIO3        ; IN (Dh)
```

Toutes les instructions mentionnées dans un cycle machine s'effectuent en parallèle. Les résultats des comparaisons successives sont produits sur la dernière colonne de la troisième rangée (rangée du milieu). Pour récupérer ces valeurs, ce processeur émet le contenu de son registre RD sur le bus de diffusion. Cette valeur est ensuite acquise par le registre de configuration qui, par défaut, délivre sur le port de sortie la dernière valeur mémorisée. Un sixième cycle machine peut, par conséquent être rajouté au cycle systolique ou être reporté au début du cycle systolique suivant. La deuxième solution donne le programme suivant :

```
cycle machine 1   RIO1_ABUS       ; RIO[Dd] -> ABUS
                  RCTE1_BBUS      ; RCTE[Kc] -> BBUS
                                  ; RADD := Dd + Kc
                  RD3_DBUS        ; RD3 -> DBUS
                  DBUS_CR

cycle machine 2   RIO2_ABUS       ; RIO[Dv] -> ABUS
                  RCTE3_BBUS      ; RCTE[Ko] -> BBUS
                                  ; RADD := Dv + Ko
                  RADD_ACC        ; ACC := RADD

cycle machine 3   RIO3_ABUS       ; RIO[Dh] -> ABUS
                  RCTE2_BBUS      ; RCTE[Ki] -> BBUS
                                  ; RADD := Dh + Ki
                  MIN_ACC         ; ACC := MIN (ACC,RADD)
                  ABUS_OUTP       ; OUT (Dh)
                  INP_RIO1        ; IN (Dd)

cycle machine 4   MIN_RD          ; RD := MIN (ACC,RADD)
                  RD_OUTP         ; OUT (MIN (ACC,RADD))
                  INP_RIO2        ; IN (Dv)

cycle machine 5   RD_OUTP         ; OUT (RD)
                  INP_RIO3        ; IN (Dh)
```

9.6 Cycle systolique d'accès à la mémoire

Pendant un cycle systolique, les registres de constante qui mémorisent la valeur de Kc (coût de substitution) doivent être systématiquement remis à jour. Cette valeur est utilisée pendant le premier cycle machine dans le cycle systolique de calcul (cf cycle précédent) et, par conséquent, ne doit pas être modifiée pendant ce cycle machine.

L'accès à la mémoire est pipeliné, c'est à dire que pendant les 5 premiers cycles machine, 5 valeurs (caractères) sont présentées successivement et que pendant les cycles machines 2 à 6, les coûts de substitution sont délivrés sur le bus DBUS. Le programme du cycle systolique d'accès à la mémoire est le suivant :

cycle machine 1	REF1_XBUS, XBUS_RAM
cycle machine 2	REF2_XBUS, XBUS_RAM RMEM_DBUS, DBUS_RCT11
cycle machine 3	REF3_XBUS, XBUS_RAM RMEM_DBUS, DBUS_RCT21
cycle machine 4	REF4_XBUS, XBUS_RAM RMEM_DBUS, DBUS_RCT31
cycle machine 5	REF5_XBUS, XBUS_RAM RMEM_DBUS, DBUS_RCT41
cycle machine 6	RMEM_DBUS, DBUS_RCT51

L'accès à la mémoire demande donc 6 cycles machines, soit un de plus que pour le processus de calcul.

9.7 Cycle systolique d'acquisition des références

En parallèle avec le calcul et les accès à la mémoire, les caractères correspondant aux références issues d'un dictionnaire doivent être acquis. Cela consiste à lire successivement 5 données sur les 3 ports d'entrée puis à effectuer un décalage sur le réseau d'alimentation.

cycle machine 1	LOAD_REF
cycle machine 2	LOAD_REF
cycle machine 3	LOAD_REF
cycle machine 4	LOAD_REF
cycle machine 5	LOAD_REF
cycle machine 6	SHIFTREF

Rappelons que la commande LOAD_REF initialise des registres d'entrée et que la commande SHIFT_REF décale d'une position vers la droite les caractères références du réseau d'alimentation tout en initialisant ceux situés sur la bordure gauche par les caractères présents dans les registres d'entrée.

9.8 Cycle systolique complet

cycle machine 1	RIO1_ABUS	; calcul
	RCTE1_BBUS	
	RD3_DBUS	; recuperation resultat
	DBUS_CR	
	REF1_XBUS, XBUS_RAM	; acces memoire
	LOAD_REF	; acquisition des references
cycle machine 2	RIO2_ABUS	; calcul
	RCTE3_BBUS	
	RADD_ACC	
	REF2_XBUS, XBUS_RAM	; acces memoire
	RMEM_DBUS, DBUS_RCT11	
	LOAD_REF	; acquisition des references
cycle machine 3	RIO3_ABUS	; calcul
	RCTE2_BBUS	
	MIN_ACC	
	ABUS_OUTP	
	INP_RIO1	
	LINK_DIAG	
	REF3_XBUS, XBUS_RAM	; acces memoire
	RMEM_DBUS, DBUS_RCT21	
	LOAD_REF	; acquisition des references
cycle machine 4	MIN_RD	; calcul
	RD_OUTP	
	INP_RIO2	
	LINK_UPDW	
	REF4_XBUS, XBUS_RAM	; acces memoire
	RMEM_DBUS, DBUS_RCT31	
	LOAD_REF	; acquisition des references
cycle machine 5	RD_OUTP	; calcul
	INP_RIO3	
	LINK_DIAG	
	REF5_XBUS, XBUS_RAM	; acces memoire
	RMEM_DBUS, DBUS_RCT41	
	LOAD_REF	; acquisition des references
cycle machine 6	RMEM_DBUS, DBUS_RCT51	; acces memoire
	SHIFTREF	; acquisition des references

Chapitre 10

API34

10.1 Introduction

Le circuit API34 est une version réduite du circuit API69. Il possède un réseau de 34 processeurs organisé en une matrice 12 × 12 où ne sont conservées que 3 diagonales. Les principales caractéristiques du circuit sont résumées dans le tableau suivant :

paramètres	
largeur du chemin de données	8
taille de la mémoire	8
nb de colonnes de processeurs	12
nb de diagonales	3
nb d'entrées référence	3
nb de registres E/S	3
nb de registres CTE	3

La puce a été réalisée en technologie CMOS 1.5μm. Le nombre de transistors est d'environ 150 000. La surface du cœur (i.e. sans les plots) est de 7,064 mm × 7,094 mm, soit à peu près 50 mm^2. Le circuit complet mesure 8,760 mm × 8,683 mm (76 mm^2).

Les tests ont montré un fonctionnement correct dès la première fabrication. La fréquence d'horloge du circuit est cependant inférieure à celle qui était attendue, compte tenu de la technologie employée. Cette faiblesse est sans doute due à une sous estimation des dispositifs réalisant l'amplification des signaux de commande.

10.2 Description

La figure 10.2 indique les connexions externes du circuit API34. Leurs fonctionnalités sont les suivantes :

- $IN_{0,7}$: entrée du registre de configuration,
- $OUT_{0,7}$: sortie du registre de configuration (sorties complémentées),
- $INST_{0,22}$: instruction,
- $REF0_{0,7}$, $REF1_{0,7}$, $REF2_{0,7}$: entrées des références,
- H : horloges (phi1, phi2, $\overline{phi1}$, $\overline{phi2}$),
- VDD, GND : alimentation électrique.

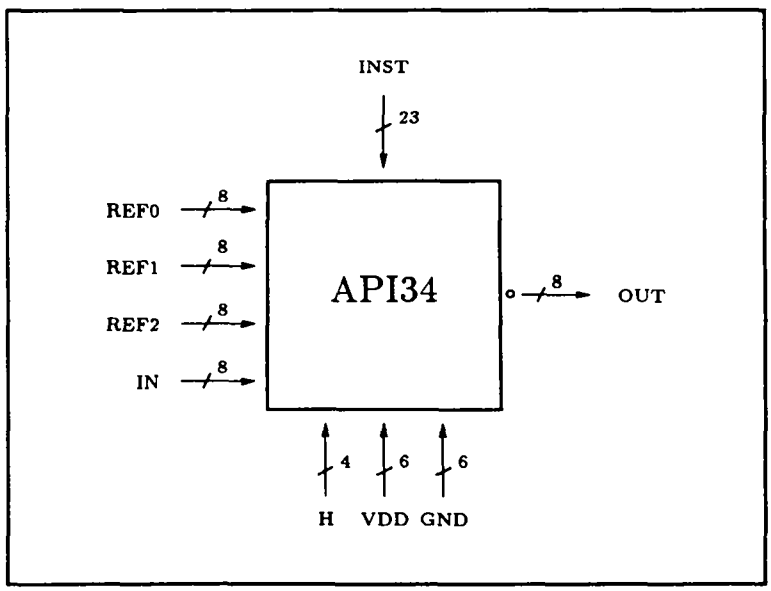


Figure 10.1 : entrées/sorties du circuit API34

10.3 Codage des instructions

Les instructions permettant de piloter le circuit sont codées sur 23 bits. Le codage est le suivant :

```
NOP                 000 0000 0000000000 000 000


Registre de Configuration
-------------------------

SHIFTCR_Q2          XXX XXXX XXXXXXXXXX XXX 001
CR_XBUS_Q2          XXX XXXX XXXXXXXXXX XXX 010
CR_DBUS_Q2          XXX XXXX XXXXXXXXXX XXX 011
XBUS_CR_Q2          XXX XXXX XXXXXXXXXX XXX 100
DBUS_CR_Q2          XXX XXXX XXXXXXXXXX XXX 101

Memoire
-------

DBUS_RMEM_Q2        XXX XXXX XXXXXXXXXX 1XX XXX
XBUS_RAM_Q2         XXX XXXX XXXXXXXXXX X1X XXX
RMEM_DBUS_Q2        XXX XXXX XXXXXXXXXX XX1 XXX
REFRESH_Q2          XXX XXXX XXXXXXXXXX 0XX XXX

DBUS --> Registres Processeur
-----------------------------

DBUS_RCTE11_Q2      XX1 XXXX XXXXXXXX01 XXX XXX
DBUS_RCTE12_Q2      XX1 XXXX XXXXXXXX10 XXX XXX
DBUS_RCTE13_Q2      XX1 XXXX XXXXXXXX11 XXX XXX
DBUS_RCTE21_Q2      X1X XXXX XXXXXXXX01 XXX XXX
DBUS_RCTE22_Q2      X1X XXXX XXXXXXXX10 XXX XXX
DBUS_RCTE23_Q2      X1X XXXX XXXXXXXX11 XXX XXX
DBUS_RCTE31_Q2      1XX XXXX XXXXXXXX01 XXX XXX
DBUS_RCTE32_Q2      1XX XXXX XXXXXXXX10 XXX XXX
DBUS_RCTE33_Q2      1XX XXXX XXXXXXXX11 XXX XXX

DBUS_RMSK1_Q2       XX1 XXXX 1XXXXXXXXX XXX XXX
DBUS_RMSK2_Q2       X1X XXXX 1XXXXXXXXX XXX XXX
DBUS_RMSK3_Q2       1XX XXXX 1XXXXXXXXX XXX XXX
```

```
Registres Processeur --> DBUS
-----------------------------

RD1_DBUS_Q2       XX1 XXXX XXX1XXXXXX XXX XXX
RD2_DBUS_Q2       X1X XXXX XXX1XXXXXX XXX XXX
RD3_DBUS_Q2       1XX XXXX XXX1XXXXXX XXX XXX

Processeur
----------

IN_RIO1_Q2        XXX XXXX XXXXXX01XX XXX XXX
IN_RIO2_Q2        XXX XXXX XXXXXX10XX XXX XXX
IN_RIO3_Q2        XXX XXXX XXXXXX11XX XXX XXX

ABUS_OUTP_Q2      XXX XXXX XXXX01XXXX XXX XXX
BBUS_OUTP_Q2      XXX XXXX XXXX10XXXX XXX XXX
RD_OUTP_Q2        XXX XXXX XXXX11XXXX XXX XXX

RCTE1_BBUS_Q1     XXX XXXX XXXXXXXX01 XXX XXX
RCTE2_BBUS_Q1     XXX XXXX XXXXXXXX10 XXX XXX
RCTE3_BBUS_Q1     XXX XXXX XXXXXXXX11 XXX XXX
RIO1_ABUS_Q1      XXX XXXX XXXXXX01XX XXX XXX
RIO2_ABUS_Q1      XXX XXXX XXXXXX10XX XXX XXX
RIO3_ABUS_Q1      XXX XXXX XXXXXX11XX XXX XXX

RADD_ACC_Q1       XXX XXXX XXXXX1XXXX XXX XXX
MIN_ACC_Q1        XXX XXXX XXXX1XXXXX XXX XXX
MIN_RD_Q1         XXX XXXX XXX1XXXXXX XXX XXX

Connexion entre Processeurs
---------------------------

LINK_UPDW_Q2      XXX XXXX X01XXXXXXX XXX XXX
LINK_DIAG_Q2      XXX XXXX X10XXXXXXX XXX XXX

Alimentation
------------

LOADREF_Q2        XXX XXX1 XXXXXXXXXX XXX XXX
SHIFTREF_Q2       XXX XX1X XXXXXXXXXX XXX XXX
REF1_XBUS_Q2      XXX 01XX XXXXXXXXXX XXX XXX
REF2_XBUS_Q2      XXX 10XX XXXXXXXXXX XXX XXX
REF3_XBUS_Q2      XXX 11XX XXXXXXXXXX XXX XXX
```

10.4 Brochage

Le circuit API34 est encapsulé dans un boîtier PGA 100. Le brochage des pins est décrit par le tableau suivant :

PAD	PIN	SIG	PAD	PIN	SIG	PAD	PIN	SIG
1	B2	NC	35	M6	IN5	69	E12	NPHI2
2	B1	NC	36	N6	IN4	70	D13	NPHI1
3	C2	GND	37	M7	IN3	71	D12	PHI1
4	C1	REF23	38	L7	NC	72	C13	GND
5	D2	REF24	39	N7	IN2	73	B13	NC
6	D1	REF25	40	N8	IN1	74	C12	NC
7	E2	REF26	41	M8	IN0	75	A13	NC
8	E1	REF27	42	L8	VDD	76	B12	NC
9	F3	REF10	43	N9	I22	77	A12	NC
10	F2	REF11	44	M9	I21	78	B11	GND
11	F1	REF12	45	N10	I20	79	A11	I4
12	G2	REF13	46	M10	I19	80	B10	I3
13	G3	NC	47	N11	I18	81	A10	I2
14	G1	REF14	48	N12	VDD	82	B9	I1
15	H1	REF15	49	M11	NC	83	A9	I0
16	H2	REF16	50	N13	NC	84	C8	GND
17	H3	REF17	51	M12	NC	85	B8	OUT0
18	N2	REF00	52	M13	NC	86	A8	OUT1
19	J2	REF01	53	L12	VDD	87	B7	OUT2
20	K1	REF02	54	L13	I17	88	C7	NC
21	K2	REF03	55	K12	I16	89	A7	OUT3
22	L1	REF04	56	K13	I15	90	A6	OUT4
23	M1	VDD	57	J12	I14	91	B6	OUT5
24	L2	NC	58	J13	I13	92	C6	OUT6
25	N1	NC	59	H11	I12	93	A5	OUT7
26	M2	NC	60	H12	I11	94	B5	GND
27	N2	NV	61	H13	I10	95	A4	REF20
28	M3	GND	62	G12	I9	96	B4	REF21
29	N3	REF05	63	G11	NC	97	A3	REF22
30	M4	REF06	64	G13	I8	98	A2	VDD
31	N4	REF07	65	F13	I7	99	B3	NC
32	M5	VDD	66	F12	I6	100	A1	NC
33	N5	IN7	67	F11	I5			
34	L6	IN6	68	E13	PHI2			

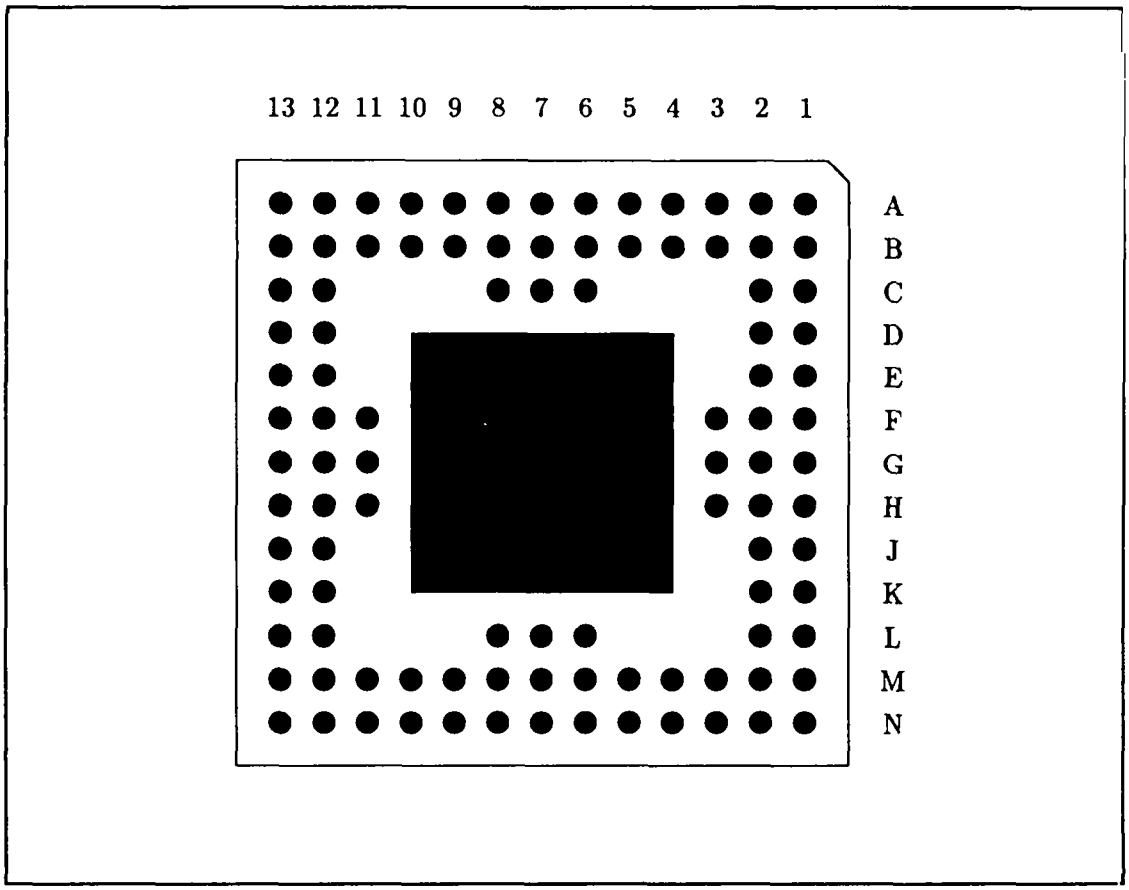
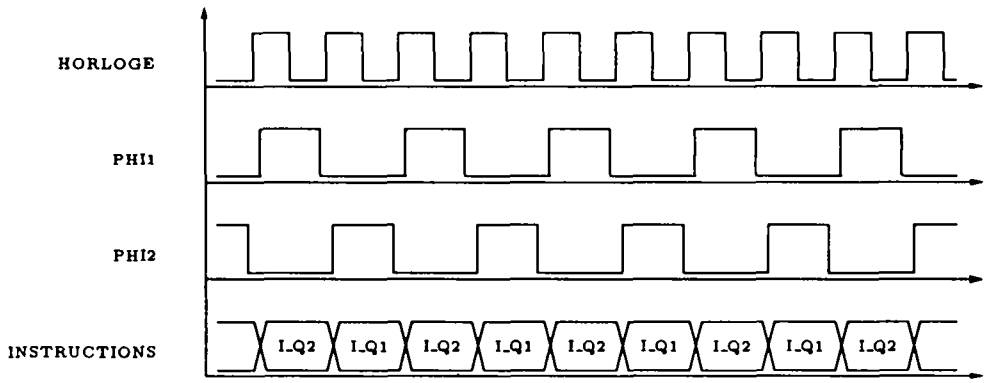


Figure 10.2 : vue coté pin

10.5 Génération des instructions

On distingue 2 types d'instructions, celles actives sur la phase PHI1 et celles actives sur la phase PHI2. Ces deux instructions doivent être émises alternativement vers le circuit API 34. Pour ces deux types d'instructions, le décodage s'effectue lors de la phase précédente. Une instruction qui s'exécute sur la phase PHI1 doit donc être présentée au circuit pendant la phase PHI2 précédente; de la même manière, une instruction de type PHI2 doit être reçue par le circuit lors d'une phase PHI1.

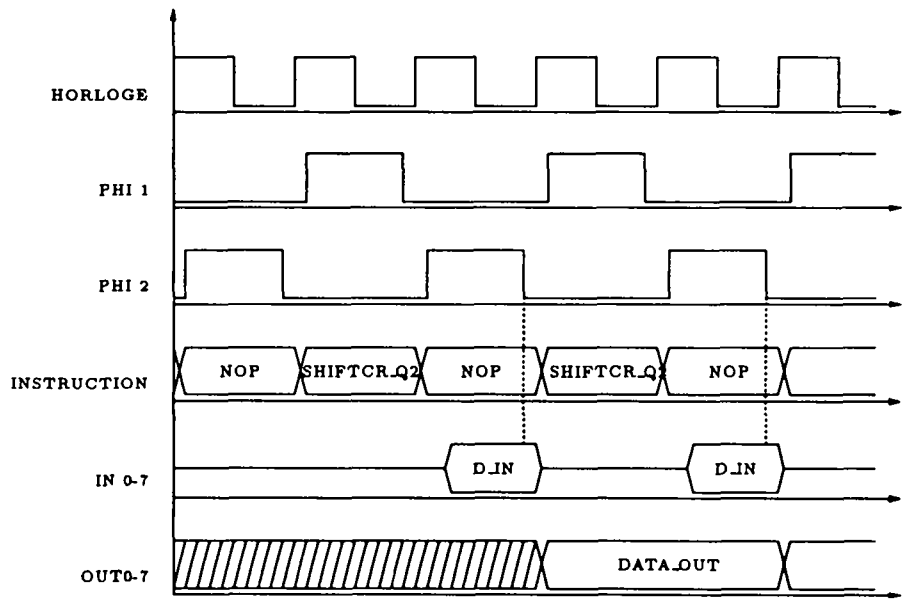
Le chronogramme ci-après indique en fonction de l'horloge et des phases PHI1-PHI2 comment les instructions doivent être présentées au circuit.



10.6 Entrées/sorties

10.6.1 Registre de configuration

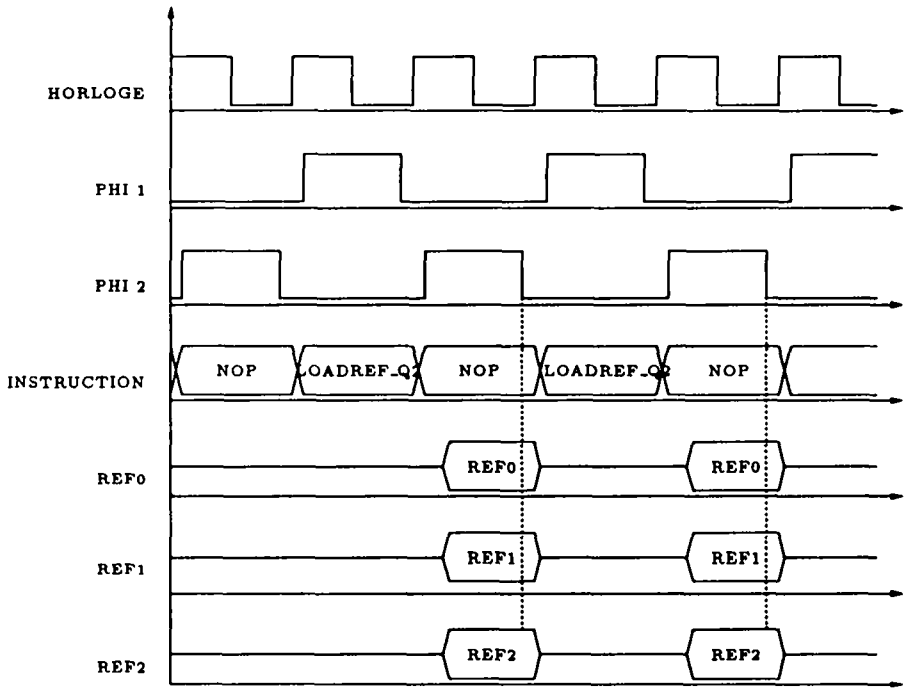
L'instruction SHIFTCR_Q2 permet d'acquérir une donnée sur le port IN et de la mémoriser dans le premier étage du registre de configuration (registre à décalage). Toutes les données stockées dans ce registre sont alors décalées vers la droite ce qui a pour effet de présenter en sortie la nouvelle valeur du dernier étage du registre de configuration. Cette valeur apparaît cependant au début de la phase PHI1 suivante ainsi que le montre le chronogramme ci-après.



Les données en entrée doivent être valides fin PHI2. Les données en sortie sont valides fin PHI1 (phase suivante) et stables PHI2.

10.6.2 Acquisition des références

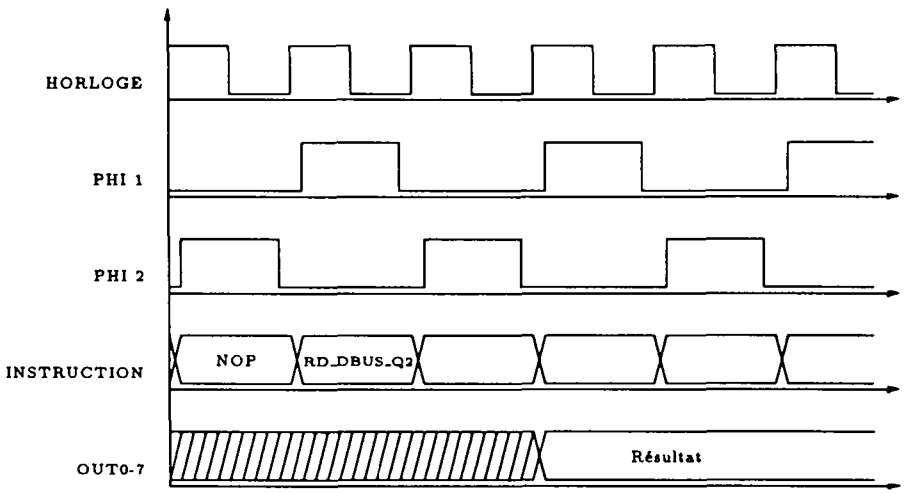
L'instruction LOADREF_Q2 commande le chargement des références (mots en provenance du dictionnaire). Les données à charger doivent être valides fin PHI2 sur les entrées REF0, REF1 et REF2.



10.6.3 Lecture du résultat

Le résultat d'une comparaison doit être prélevé sur le processeur de la rangée du milieu de la dernière colonne. Pour réaliser ceci, on active la commande RD2_DBUS_Q2 qui charge le registre de configuration avec le contenu des registres RD de tous les processeurs de la rangée du milieu. Le dernier étage du registre de configuration est donc chargé avec la valeur qui nous intéresse. Cette valeur apparaît en sortie en début de phase PHI1 suivante.

La valeur présente en sortie reste stable pendant un *certain* temps. Le registre de configuration étant dynamique (i.e. sans rafraîchissement) la mémorisation est fonction de la capacité associée à la grille des transistors réalisant la cellule mémoire. En pratique, on peut considérer qu'une donnée reste stable pendant quelques millisecondes.


HORLOGE
PHI 1
PHI 2
INSTRUCTION
NOP
RD_DBUS_Q2
OUT0-7
Résultat

Remerciements

Je remercie Patrice Frison et Joel Ristori pour leur lecture attentive de ce rapport et des nombreuses remarques qui, je l'espère, contribuent à la clarté et à la compréhension de ce texte.

Annexe A

Fichiers de Description

Les pages suivantes donnent respectivement les fichiers de descriptions du noyau, du circuit API69 et du circuit API34.

C'est à partir de ces fichiers qu'ont été construits les différents outils (simulateurs, générateurs, assembleurs, ...) permettant la réalisation et la vérification des circuits. C'est pourquoi on y retrouve toutes les informations qui caractérisent le circuit.

La syntaxe d'une ligne peut prendre une des trois configurations suivantes :

- *; commentaires*
- *D <ident> <nombre>*
  les différents identificateurs représentent des noms de constante dans les programmes C. Ainsi, à partir d'un tel fichier de description la génération automatique d'un fichier de définition de constantes (directive *define*) est immédiate.
- *C <nom> <phase> <codage> <type> <module>*
  cette ligne permet de définir complètement une commande : son mnémonique, la phase ou elle s'exécute, son codage et le module auquel elle appartient. Le champ *type* est une information propre au simulateur.

Le symbole # repère la fin de la description.

A.1 Noyau

```
; ------------------------------------------------------------------------
; *****      VERSION NOYAU : 3 diagonales et 6 colonnes              *****
; ------------------------------------------------------------------------
;
; largeur chemin de donnees
D NBB 4
;
; taille de la memoire
D TMEM 3
;
; nb de colonnes de processeur
D NBCOL 6
;
; nb de diagonales (ou lignes de processeurs)
D NBDIAG 3
;
; nb d'entrees reference
D NBENREF 3
;
; nb de registre E/S
D NBRIO 3
;
; nb de registre CTE
D NBRCTE 3
;
; commandes du circuit API69
;
; nom              phase  codage                        type       module
; ------------------------------------------------------------------------
;
C SHIFTCR_Q2       2      XXXXXXXXXXXXXXXXXXXXX001   B          REGCONF
C CR_XBUS_Q2       2      XXXXXXXXXXXXXXXXXXXXX010   A          REGCONF
C CR_DBUS_Q2       2      XXXXXXXXXXXXXXXXXXXXX011   A          REGCONF
C XBUS_CR_Q2       2      XXXXXXXXXXXXXXXXXXXXX100   B          REGCONF
C DBUS_CR_Q2       2      XXXXXXXXXXXXXXXXXXXXX101   B          REGCONF
;
C DBUS_RMEM_Q2     2      XXXXXXXXXXXXXXXXX1XXXXX   B          MEMOIRE
C XBUS_RAM_Q2      2      XXXXXXXXXXXXXXXXXX1XXXX   B          MEMOIRE
C RMEM_DBUS_Q2     2      XXXXXXXXXXXXXXXXXXX1XXX   A          MEMOIRE
C REFRESH_Q2       2      XXXXXXXXXXXXXXXXXX0XXXXX   B          MEMOIRE
;
C DBUS_RCTE11_Q2   2      XX1XXXXXXXXXXXXX01XXXXXX   B          PROCESS
C DBUS_RCTE12_Q2   2      XX1XXXXXXXXXXXXX10XXXXXX   B          PROCESS
C DBUS_RCTE13_Q2   2      XX1XXXXXXXXXXXXX11XXXXXX   B          PROCESS
C DBUS_RCTE21_Q2   2      X1XXXXXXXXXXXXXX01XXXXXX   B          PROCESS
C DBUS_RCTE22_Q2   2      X1XXXXXXXXXXXXXX10XXXXXX   B          PROCESS
```

```
C DBUS_RCTE23_Q2    2       X1XXXXXXXXXXXXX11XXXXXX    B           PROCESS
C DBUS_RCTE31_Q2    2       1XXXXXXXXXXXXXX01XXXXXX    B           PROCESS
C DBUS_RCTE32_Q2    2       1XXXXXXXXXXXXXX10XXXXXX    B           PROCESS
C DBUS_RCTE33_Q2    2       1XXXXXXXXXXXXXX11XXXXXX    B           PROCESS
;
C DBUS_RMSK1_Q2     2       XX1XXXX1XXXXXXXXXXXXXXX    B           PROCESS
C DBUS_RMSK2_Q2     2       X1XXXXX1XXXXXXXXXXXXXXX    B           PROCESS
C DBUS_RMSK3_Q2     2       1XXXXXX1XXXXXXXXXXXXXXX    B           PROCESS
C IN_RIO1_Q2        2       XXXXXXXXXXXXX01XXXXXXXX    B           PROCESS
C IN_RIO2_Q2        2       XXXXXXXXXXXXX10XXXXXXXX    B           PROCESS
C IN_RIO3_Q2        2       XXXXXXXXXXXXX11XXXXXXXX    B           PROCESS
;
C ABUS_OUTP_Q2      2       XXXXXXXXXXX01XXXXXXXXXX    A           PROCESS
C BBUS_OUTP_Q2      2       XXXXXXXXXXX10XXXXXXXXXX    A           PROCESS
C RD_OUTP_Q2        2       XXXXXXXXXXX11XXXXXXXXXX    A           PROCESS
C RD1_DBUS_Q2       2       XX1XXXXXXX1XXXXXXXXXXX    A           PROCESS
C RD2_DBUS_Q2       2       X1XXXXXXXX1XXXXXXXXXXX    A           PROCESS
C RD3_DBUS_Q2       2       1XXXXXXXXX1XXXXXXXXXXX    A           PROCESS
;
C RCTE1_BBUS_Q1     1       XXXXXXXXXXXXXXX01XXXXXX    A           PROCESS
C RCTE2_BBUS_Q1     1       XXXXXXXXXXXXXXX10XXXXXX    A           PROCESS
C RCTE3_BBUS_Q1     1       XXXXXXXXXXXXXXX11XXXXXX    A           PROCESS
C RIO1_ABUS_Q1      1       XXXXXXXXXXXXX01XXXXXXXX    A           PROCESS
C RIO2_ABUS_Q1      1       XXXXXXXXXXXXX10XXXXXXXX    A           PROCESS
C RIO3_ABUS_Q1      1       XXXXXXXXXXXXX11XXXXXXXX    A           PROCESS
;
C RADD_ACC_Q1       1       XXXXXXXXXXXX1XXXXXXXXXX    B           PROCESS
C MIN_ACC_Q1        1       XXXXXXXXXXX1XXXXXXXXXXX    B           PROCESS
C MIN_RD_Q1         1       XXXXXXXXXX1XXXXXXXXXXXX    B           PROCESS
;
C LINK_UPDW_Q2      2       XXXXXXXX01XXXXXXXXXXXX    A           PROCESS
C LINK_DIAG_Q2      2       XXXXXXXX10XXXXXXXXXXXX    A           PROCESS
;
C LOADREF_Q2        2       XXXXXX1XXXXXXXXXXXXXXXX    B           ALIMENT
C SHIFTREF_Q2       2       XXXXX1XXXXXXXXXXXXXXXX    B           ALIMENT
C REF1_XBUS_Q2      2       XXX01XXXXXXXXXXXXXXXXX    A           ALIMENT
C REF2_XBUS_Q2      2       XXX10XXXXXXXXXXXXXXXXX    A           ALIMENT
C REF3_XBUS_Q2      2       XXX11XXXXXXXXXXXXXXXXX    A           ALIMENT
;
; nb : les signaux de type A modifient la valeur des bus
;      ils sont actives en premier dans le simulateur
#
```

A.2 API69

```
; ------------------------------------------------------------------------
; *****          VERSION API69 : 5 diagonales et 15 colonnes          *****
; ------------------------------------------------------------------------
;
; largeur chemin de donnees
D NBB 8
;
; taille de la memoire
D TMEM 10
;
; nb de colonnes de processeur
D NBCOL 15
;
; nb de diagonales (ou lignes de processeurs)
D NBDIAG 5
;
; nb d'entrees reference
D NBENREF 3
;
; nb de registre E/S
D NBRIO 8
;
; nb de registre CTE
D NBRCTE 4
;
; commandes du circuit API69
;
; nom               phase  codage                          type    module
; ------------------------------------------------------------------------
;
C SHIFTCR_Q2        2      XXXXXXXXXXXXXXXXXXXXXXXX001  B       REGCONF
C CR_XBUS_Q2        2      XXXXXXXXXXXXXXXXXXXXXXXX010  A       REGCONF
C CR_DBUS_Q2        2      XXXXXXXXXXXXXXXXXXXXXXXX011  A       REGCONF
C XBUS_CR_Q2        2      XXXXXXXXXXXXXXXXXXXXXXXX100  B       REGCONF
C DBUS_CR_Q2        2      XXXXXXXXXXXXXXXXXXXXXXXX101  B       REGCONF
;
C DBUS_RMEM_Q2      2      XXXXXXXXXXXXXXXXXXXXX1XXXXX  B       MEMOIRE
C XBUS_RAM_Q2       2      XXXXXXXXXXXXXXXXXXXXX1XXXX  B       MEMOIRE
C RMEM_DBUS_Q2      2      XXXXXXXXXXXXXXXXXXXXX1XXX  A       MEMOIRE
C REFRESH_Q2        2      XXXXXXXXXXXXXXXXXXXXX0XXXX  B       MEMOIRE
;
C DBUS_RCTE10_Q2    2      XXXX1XXXXXXXXXXXXXX00XXXXX  B       PROCESS
C DBUS_RCTE11_Q2    2      XXXX1XXXXXXXXXXXXXX01XXXXX  B       PROCESS
C DBUS_RCTE12_Q2    2      XXXX1XXXXXXXXXXXXXX10XXXXX  B       PROCESS
C DBUS_RCTE13_Q2    2      XXXX1XXXXXXXXXXXXXX11XXXXX  B       PROCESS
C DBUS_RCTE20_Q2    2      XXX1XXXXXXXXXXXXXXX00XXXXX  B       PROCESS
```

```
C DBUS_RCTE21_Q2    2      XXX1XXXXXXXXXXXXXXX01XXXXXX  B      PROCESS
C DBUS_RCTE22_Q2    2      XXX1XXXXXXXXXXXXXXX10XXXXXX  B      PROCESS
C DBUS_RCTE23_Q2    2      XXX1XXXXXXXXXXXXXXX11XXXXXX  B      PROCESS
C DBUS_RCTE30_Q2    2      XX1XXXXXXXXXXXXXXXX00XXXXXX  B      PROCESS
C DBUS_RCTE31_Q2    2      XX1XXXXXXXXXXXXXXXX01XXXXXX  B      PROCESS
C DBUS_RCTE32_Q2    2      XX1XXXXXXXXXXXXXXXX10XXXXXX  B      PROCESS
C DBUS_RCTE33_Q2    2      XX1XXXXXXXXXXXXXXXX11XXXXXX  B      PROCESS
C DBUS_RCTE40_Q2    2      X1XXXXXXXXXXXXXXXXX00XXXXXX  B      PROCESS
C DBUS_RCTE41_Q2    2      X1XXXXXXXXXXXXXXXXX01XXXXXX  B      PROCESS
C DBUS_RCTE42_Q2    2      X1XXXXXXXXXXXXXXXXX10XXXXXX  B      PROCESS
C DBUS_RCTE43_Q2    2      X1XXXXXXXXXXXXXXXXX11XXXXXX  B      PROCESS
C DBUS_RCTE50_Q2    2      1XXXXXXXXXXXXXXXXXX00XXXXXX  B      PROCESS
C DBUS_RCTE51_Q2    2      1XXXXXXXXXXXXXXXXXX01XXXXXX  B      PROCESS
C DBUS_RCTE52_Q2    2      1XXXXXXXXXXXXXXXXXX10XXXXXX  B      PROCESS
C DBUS_RCTE53_Q2    2      1XXXXXXXXXXXXXXXXXX11XXXXXX  B      PROCESS
;
C IN_RIO0_Q2        2      XXXXXXXXXXXXXXXX000XXXXXXXX  B      PROCESS
C IN_RIO1_Q2        2      XXXXXXXXXXXXXXXX001XXXXXXXX  B      PROCESS
C IN_RIO2_Q2        2      XXXXXXXXXXXXXXXX010XXXXXXXX  B      PROCESS
C IN_RIO3_Q2        2      XXXXXXXXXXXXXXXX011XXXXXXXX  B      PROCESS
C IN_RIO4_Q2        2      XXXXXXXXXXXXXXXX100XXXXXXXX  B      PROCESS
C IN_RIO5_Q2        2      XXXXXXXXXXXXXXXX101XXXXXXXX  B      PROCESS
C IN_RIO7_Q2        2      XXXXXXXXXXXXXXXX110XXXXXXXX  B      PROCESS
C IN_RIO7_Q2        2      XXXXXXXXXXXXXXXX111XXXXXXXX  B      PROCESS
;
C ABUS_OUTP_Q2      2      XXXXXXXXXXXXXXX01XXXXXXXXXX  A      PROCESS
C BBUS_OUTP_Q2      2      XXXXXXXXXXXXXXX10XXXXXXXXXX  A      PROCESS
C RD_OUTP_Q2        2      XXXXXXXXXXXXXXX11XXXXXXXXXX  A      PROCESS
;
C RD1_DBUS_Q2       2      XXXX1XXXXXXXXX1XXXXXXXXXXXX  A      PROCESS
C RD2_DBUS_Q2       2      XXX1XXXXXXXXX1XXXXXXXXXXXXX  A      PROCESS
C RD3_DBUS_Q2       2      XX1XXXXXXXXXX1XXXXXXXXXXXXX  A      PROCESS
C RD4_DBUS_Q2       2      X1XXXXXXXXXXX1XXXXXXXXXXXXX  A      PROCESS
C RD5_DBUS_Q2       2      1XXXXXXXXXXXX1XXXXXXXXXXXXX  A      PROCESS
;
C LINK_UPDW_Q2      2      XXXXXXXXXXXX01XXXXXXXXXXXXX  A      PROCESS
C LINK_DIAG_Q2      2      XXXXXXXXXXXX10XXXXXXXXXXXXX  A      PROCESS
;
C DBUS_RMSK1_Q2     2      XXXX1XXXXXX1XXXXXXXXXXXXXXX  B      PROCESS
C DBUS_RMSK2_Q2     2      XXX1XXXXXXX1XXXXXXXXXXXXXXX  B      PROCESS
C DBUS_RMSK3_Q2     2      XX1XXXXXXXX1XXXXXXXXXXXXXXX  B      PROCESS
C DBUS_RMSK4_Q2     2      X1XXXXXXXXX1XXXXXXXXXXXXXXX  B      PROCESS
C DBUS_RMSK5_Q2     2      1XXXXXXXXXX1XXXXXXXXXXXXXXX  B      PROCESS
;
C RCTE0_BBUS_Q1     1      XXXXXXXXXXXXXXXXXXX00XXXXXX  A      PROCESS
C RCTE1_BBUS_Q1     1      XXXXXXXXXXXXXXXXXXX01XXXXXX  A      PROCESS
C RCTE2_BBUS_Q1     1      XXXXXXXXXXXXXXXXXXX10XXXXXX  A      PROCESS
C RCTE3_BBUS_Q1     1      XXXXXXXXXXXXXXXXXXX11XXXXXX  A      PROCESS
;
```

```
C RIO0_ABUS_Q1      1      XXXXXXXXXXXXXXXXX000XXXXXXXX  A       PROCESS
C RIO1_ABUS_Q1      1      XXXXXXXXXXXXXXXXX001XXXXXXXX  A       PROCESS
C RIO2_ABUS_Q1      1      XXXXXXXXXXXXXXXXX010XXXXXXXX  A       PROCESS
C RIO3_ABUS_Q1      1      XXXXXXXXXXXXXXXXX011XXXXXXXX  A       PROCESS
C RIO4_ABUS_Q1      1      XXXXXXXXXXXXXXXXX100XXXXXXXX  A       PROCESS
C RIO5_ABUS_Q1      1      XXXXXXXXXXXXXXXXX101XXXXXXXX  A       PROCESS
C RIO6_ABUS_Q1      1      XXXXXXXXXXXXXXXXX110XXXXXXXX  A       PROCESS
C RIO7_ABUS_Q1      1      XXXXXXXXXXXXXXXXX111XXXXXXXX  A       PROCESS
;
C RADD_ACC_Q1       1      XXXXXXXXXXXXXXXX1XXXXXXXXXXX  B       PROCESS
C MIN_ACC_Q1        1      XXXXXXXXXXXXXXX1XXXXXXXXXXXX  B       PROCESS
C MIN_RD_Q1         1      XXXXXXXXXXXXXXX1XXXXXXXXXXXX  B       PROCESS
;
C LOADREF_Q2        2      XXXXXXXXX1XXXXXXXXXXXXXXXXX  B       ALIMENT
C SHIFTREF_Q2       2      XXXXXXXX1XXXXXXXXXXXXXXXXXX  B       ALIMENT
C REF1_XBUS_Q2      2      XXXXX001XXXXXXXXXXXXXXXXXXX  A       ALIMENT
C REF2_XBUS_Q2      2      XXXXX010XXXXXXXXXXXXXXXXXXX  A       ALIMENT
C REF3_XBUS_Q2      2      XXXXX011XXXXXXXXXXXXXXXXXXX  A       ALIMENT
C REF4_XBUS_Q2      2      XXXXX100XXXXXXXXXXXXXXXXXXX  A       ALIMENT
C REF5_XBUS_Q2      2      XXXXX101XXXXXXXXXXXXXXXXXXX  A       ALIMENT
;
; nb : les signaux de type A modifient la valeur des bus
;      ils sont actives en premier dans le simulateur
#
```

A.3 API 34

```
; ------------------------------------------------------------------------
; *****          VERSION API34 : 3 diagonales et 12 colonnes          *****
; ------------------------------------------------------------------------
;
; largeur chemin de donnees
D NBB 8
;
; taille de la memoire
D TMEM 8
;
; nb de colonnes de processeur
D NBCOL 12
;
; nb de diagonales (ou lignes de processeurs)
D NBDIAG 3
;
; nb d'entrees reference
D NBENREF 3
;
; nb de registre E/S
D NBRIO 3
;
; nb de registre CTE
D NBRCTE 3
;
; commandes du circuit API69
;
; nom               phase  codage                    type   module
; ------------------------------------------------------------------
;
C SHIFTCR_Q2        2      XXXXXXXXXXXXXXXXXXX001  B      REGCONF
C CR_XBUS_Q2        2      XXXXXXXXXXXXXXXXXXX010  A      REGCONF
C CR_DBUS_Q2        2      XXXXXXXXXXXXXXXXXXX011  A      REGCONF
C XBUS_CR_Q2        2      XXXXXXXXXXXXXXXXXX100  B      REGCONF
C DBUS_CR_Q2        2      XXXXXXXXXXXXXXXXXX101  B      REGCONF
;
C DBUS_RMEM_Q2      2      XXXXXXXXXXXXXXXXX1XXXX  B      MEMOIRE
C XBUS_RAM_Q2       2      XXXXXXXXXXXXXXXXX1XXXX  B      MEMOIRE
C RMEM_DBUS_Q2      2      XXXXXXXXXXXXXXXXX1XXX  A      MEMOIRE
C REFRESH_Q2        2      XXXXXXXXXXXXXXXXX0XXXX  B      MEMOIRE
;
C DBUS_RCTE11_Q2    2      XX1XXXXXXXXXXX01XXXXX  B      PROCESS
C DBUS_RCTE12_Q2    2      XX1XXXXXXXXXXX10XXXXX  B      PROCESS
C DBUS_RCTE13_Q2    2      XX1XXXXXXXXXXX11XXXXX  B      PROCESS
C DBUS_RCTE21_Q2    2      X1XXXXXXXXXXXX01XXXXX  B      PROCESS
C DBUS_RCTE22_Q2    2      X1XXXXXXXXXXXX10XXXXX  B      PROCESS
```

```
C DBUS_RCTE23_Q2     2      X1XXXXXXXXXXXXX11XXXXXX  B      PROCESS
C DBUS_RCTE31_Q2     2      1XXXXXXXXXXXXXX01XXXXXX  B      PROCESS
C DBUS_RCTE32_Q2     2      1XXXXXXXXXXXXXX10XXXXXX  B      PROCESS
C DBUS_RCTE33_Q2     2      1XXXXXXXXXXXXXX11XXXXXX  B      PROCESS
;
C DBUS_RMSK1_Q2      2      XX1XXXX1XXXXXXXXXXXXXXX  B      PROCESS
C DBUS_RMSK2_Q2      2      X1XXXXX1XXXXXXXXXXXXXXX  B      PROCESS
C DBUS_RMSK3_Q2      2      1XXXXXX1XXXXXXXXXXXXXXX  B      PROCESS
C IN_RIO1_Q2         2      XXXXXXXXXXXXX01XXXXXXXX  B      PROCESS
C IN_RIO2_Q2         2      XXXXXXXXXXXXX10XXXXXXXX  B      PROCESS
C IN_RIO3_Q2         2      XXXXXXXXXXXXX11XXXXXXXX  B      PROCESS
;
C ABUS_OUTP_Q2       2      XXXXXXXXXXX01XXXXXXXXXX  A      PROCESS
C BBUS_OUTP_Q2       2      XXXXXXXXXXX10XXXXXXXXXX  A      PROCESS
C RD_OUTP_Q2         2      XXXXXXXXXXX11XXXXXXXXXX  A      PROCESS
C RD1_DBUS_Q2        2      XX1XXXXXXX1XXXXXXXXXXXX  A      PROCESS
C RD2_DBUS_Q2        2      X1XXXXXXXX1XXXXXXXXXXXX  A      PROCESS
C RD3_DBUS_Q2        2      1XXXXXXXXX1XXXXXXXXXXXX  A      PROCESS
;
C RCTE1_BBUS_Q1      1      XXXXXXXXXXXXXX01XXXXXX  A      PROCESS
C RCTE2_BBUS_Q1      1      XXXXXXXXXXXXXX10XXXXXX  A      PROCESS
C RCTE3_BBUS_Q1      1      XXXXXXXXXXXXXX11XXXXXX  A      PROCESS
C RIO1_ABUS_Q1       1      XXXXXXXXXXXXX01XXXXXXXX  A      PROCESS
C RIO2_ABUS_Q1       1      XXXXXXXXXXXXX10XXXXXXXX  A      PROCESS
C RIO3_ABUS_Q1       1      XXXXXXXXXXXXX11XXXXXXXX  A      PROCESS
;
C RADD_ACC_Q1        1      XXXXXXXXXXX1XXXXXXXXXX  B      PROCESS
C MIN_ACC_Q1         1      XXXXXXXXXX1XXXXXXXXXXX  B      PROCESS
C MIN_RD_Q1          1      XXXXXXXXX1XXXXXXXXXXXX  B      PROCESS
;
C LINK_UPDW_Q2       2      XXXXXXX01XXXXXXXXXXXXX  A      PROCESS
C LINK_DIAG_Q2       2      XXXXXXX10XXXXXXXXXXXXX  A      PROCESS
;
C LOADREF_Q2         2      XXXXX1XXXXXXXXXXXXXXXX  B      ALIMENT
C SHIFTREF_Q2        2      XXXX1XXXXXXXXXXXXXXXX  B      ALIMENT
C REF1_XBUS_Q2       2      XX01XXXXXXXXXXXXXXXXX  A      ALIMENT
C REF2_XBUS_Q2       2      XX10XXXXXXXXXXXXXXXXX  A      ALIMENT
C REF3_XBUS_Q2       2      XX11XXXXXXXXXXXXXXXXX  A      ALIMENT
;
; nb : les signaux de type A modifient la valeur des bus
;      ils sont actives en premier dans le simulateur
#
```

Références

[1] D. J. Burn, B. D. Ackland, N. Weste, "Array Configurations for Dynamic Time Warping," *IEEE Trans. on acoustic, speech and signal processing,* vol. ASSP-32 n° 1, pp. 119-127, feb. 1984.

[2] F. Charot, P. Frison, P. Quinton, "Systolic Architectures for Connected Speech Recognition," *IEEE Trans on ASSP,* vol. 34, n° 4, pp. 765-779, 1986.

[3] P. Frison, E. Gautrin, D. Lavenier, J.L. Scharbarg, "Reseaux specifiques a base du processeur API15C," *Deuxième Symposium : Architectures Nouvelles de Machines,* Toulouse, 12-14 sept 1990.

[4] S. Y. Kung, "NCR's GAPP Chip," in "VLSI Array Processors," *Prentice Hall Information and System Sciences Series,* pp 470, 1988.

[5] P. A. V. Hall, G. R. Dowling, "Approximate string matching," *Comput. Surv.,* vol. 12, pp. 381-402, 1980.

[6] H. T. Kung, "Why systolic architectures," *Computer,* vol. 15, pp. 37-46, Jan. 1982.

[7] D. Lavenier, " MicMacs : un réseau systolique linéaire programmable pour le traitement de chaînes de caractères," *Thèse de l'université de Rennes 1,* Juin 1989.

[8] R. Lowrance, R. A. Wagner, "An extension of the string to string correction problem," *J. Assoc. Comput. Mach.,* vol. 22, n° 2, pp. 177-183, 1975.

[9] C. Mead, L. Conway, "Introduction to VLSI systems," Addison-Wesley, 1980.

[10] P. Quinton, Y. Robert, "Algorithmes et Architectures Systoliques," *ed. Masson,* 1989.

[11] A. Wagner, J.M. Fisher, "The string to string correction problem," *J. ACM,* vol. 21, pp. 168-173, 1974.

LISTE DES DERNIERES PUBLICATIONS INTERNES PARUES A L'IRISA

PI 661 REACHABILITY ANALYSIS ON DISTRIBUTED EXECUTIONS
Claire DIEHL, Claude JARD, Jean-Xavier RAMPON
Juin 1992, 18 pages.

PI 662 RECONSTRUCTION 3D DE PRIMITIVES GEOMETRIQUES PAR VISION ACTIVE
Samia BOUKIR, François CHAUMETTE
Juin 1992, 40 pages.

PI 663 FILTRES SEMANTIQUES EN CALCUL PROPOSITIONNEL
Raymond ROLLAND
Juin 1992, 22 pages.

PI 664 REGION-BASED TRACKING IN AN IMAGE SEQUENCE
François MEYER, Patrick BOUTHEMY
Juin 1992, 50 pages.

PI 665 CORRECTNESS OF AUTOMATED DISTRIBUTION OF SEQUENTIAL PROGRAMS
Cyrille BARREAU, Benoît CAILLAUD, Claude JARD, René THORAVAL
Juin 1992, 32 pages.

PI 666 AGREGATION FAIBLE DES PROCESSUS DE MARKOV ABSORBANTS
James LEDOUX, Gerardo RUBINO, Bruno SERICOLA
Juillet 1992, 30 pages.

PI 667 MODELES D'EVALUATION DE LA FIABILITE DU LOGICIEL ET TECHNIQUES DE VALIDATION DE SYSTEMES DE PREDICTION : ETUDE BIBLIOGRAPHIQUE
James LEDOUX
Juillet 1992, 76 pages.

PI 668 TWO COMPLEMENTARY NOTES ON SKEWED-ASSOCIATIVE CACHES
André SEZNEC
Juillet 1992, 10 pages.

PI 669 PARALLELISATION D'UN ALGORITHME DE DETECTION DE MOUVEMENT SUR UNE ARCHITECTURE MIMD
Fabrice HEITZ, Sergui JUFRESA, Etienne MEMIN, Thierry PRIOL
Juillet 1992, 34 pages.

PI 670 UN RESEAU SYSTOLIQUE INTEGRE POUR LA CORRECTION DE FAUTES DE FRAPPE
Dominique LAVENIER
Juillet 1992, 120 pages.

PI 671 EARLY WARNING OF SLIGHT CHANGES IN SYSTEMS AND PLANTS WITH APPLICATION TO CONDITION BASED MAINTENANCE
Qinghua ZHANG, Michèle BASSEVILLE, Albert BENVENISTE
Juillet 1992, 32 pages.

PI 672 ORDRES REPRESENTABLES PAR DES TRANSLATIONS DE SEGMENTS DANS LE PLAN
Vincent BOUCHITTE, Roland JEGOU, Jean-Xavier RAMPON
Juillet 1992, 8 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399