Titre: Title:	Etude de la réalisation en circuits intégrés de l'algorithme de viterbi pour combattre le phénomène d'interférence entre symboles
Auteur: Author:	Jean-Didier Allegrucci
Date:	1989
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Allegrucci, J.-D. (1989). Etude de la réalisation en circuits intégrés de l'algorithme de viterbi pour combattre le phénomène d'interférence entre symboles [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/57916/

UNIVERSITÉ DE MONTRÉAL

ÉTUDE DE LA RÉALISATION EN CIRCUITS INTÉGRÉS DE L'ALGORITHME DE VITERBI POUR COMBATTRE LE PHÉNOMÈNE D'INTERFÉRENCE ENTRE SYMBOLES

par

Jean-Didier ALLEGRUCCI
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU GRADE DE MAITRE ÈS SCIENCES APPLIQUÉES (M. Sc. A.)

Mars 1989

ISBN 0-315-52695-5

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE

Ce mémoire intitulé:

ÉTUDE DE LA RÉALISATION EN CIRCUITS INTÉGRÉS DE L'ALGORITHME DE VITERBI POUR COMBATTRE LE PHÉNOMÈNE D'INTERFÉRENCE ENTRE SYMBOLES

présenté par: Jean-Didier Allegrucci

en vue de l'obtention du grade de: MAITRE ES SCIENCES APPLIQUÉES

a été dûment accepté par le jury d'examen constitué de:

M. David Haccoun, Ph.D., Président

M. Jean Conan, Ph.D.

M. Yvon Savaria, Ph.D.

SOMMAIRE

La récente évolution de la micro-électronique a rendu l'intégration de circuits accessible à tous les milieux scientifiques, industriels autant qu'académiques. Dans tous les domaines, on met à profit les qualités de cette nouvelle technologie (faibles coûts, performances accrues, ...). Il est cependant nécessaire de s'adapter à cette nouvelle philosophie de conception qui n'a pas les mêmes contraintes que la conception utilisant des composants discrets. Dans ce contexte, cette étude vise à analyser les possibilités d'appliquer l'intégration à grande échelle de circuits à l'algorithme de Viterbi utilisé pour palier au phénomène d'interférence entre symboles, phénomène présent lors de transmissions numériques à travers un canal à bande de fréquence limitée. Les principes classiques d'architectures, tels le parallélisme et le traitement pipeline de données, sont appliqués à l'algorithme. Les différents processeurs ainsi conçus, jumelés à des tables de métriques et à une logique spéciale pour la mise à jour de la mémoire des chemins survivants, forment un système complet pour combattre l'interférence entre symboles. Une puce relative à un des différents modules du détecteur de Viterbi a également été conçue et fabriquée. Elle a été spécialement optimisée pour s'occuper de la mise à jour des chemins survivants, une opération critique de l'algorithme de Viterbi.

ABSTRACT

The recent evolution of micro-electronics has made application specific integrated circuits accessible to the whole scientific community both in industry and in academic institutions. Like in every other fields, one always tries to take advantage of the characteristics of a new technology such as low costs, increased performances, etc... However, it turns out that it requires adaptation to a new design philosophy,quite different from the one used in the design of printed circuit. In this perspective, this study analyzes the possible VLSI implementation of the Viterbi algorithm as applied to the correction of intersymbol interference. It is well known that this phenomenon is present in digital communication over bandwidth limited time-dispersive channels. Classical architectures, like the parallel or the pipeline approaches, are applied to the algorithm. The corresponding required processors, coupled with external ROMs and a special device for storing the memory paths, can be used together to form a complete system used to fight intersymbol interference. Furthermore, a custom chip, which is part of the Viterbi detector, has also been designed and fabricated. It has been optimized for the management of the memory paths, a critical operation in the implementation of the Viterbi algorithm.

REMERCIEMENTS

J'aimerais avant tout remercier le Dr Jean Conan qui m'a donné la chance de travailler sur un projet relié aux domaines de la micro-électronique et des communications. Son apport aux niveaux technique et financier a été d'un grand recours tout au long de ma recherche.

Je désirerais présenter des remerciements très spéciaux à M. Gilles-André Morin pour son support moral ainsi que pour son aide énorme pour l'élaboration de la mise en page et l'impression de ce mémoire. Je tiens également à remercier ma famille et Stéphanie pour m'avoir supporté tout au long de ma maîtrise.

Finalement, je remercie l'École Polytechnique de Montréal, la Société Canadienne de Micro-électronique, ainsi que la Corporation Mentor Graphics pour les outils qu'ils ont mis à ma disposition pour la réalisation de cette recherche.

TABLE DES MATIÈRES

		Page
SOMMAIRE		iv
ABSTRACT		v
REMERCIEMENTS		vi
TABLE DES MATIÈRES		vii
LISTE DES FIGURES		x
LISTE DES TABLEAUX		xii
CHAPITRE I :	INTRODUCTION	1
CHAPITRE II :	INTERFÉRENCE ENTRE SYMBOLES ET ALGORITHME DE VITERBI	6
2.1	Interférence entre symboles	6
2.1.1	Modèle de transmission du canal PAM	6
2.1.2	Détection optimale à maximum de vraisemblance	8
2.1.3	Définition du treillis de décodage	12
2.2	Description de l'algorithme de Viterbi	14
2.3	Modifications nécessaires pour une réalisation pratique	18
2.3.1	Quantification des métriques	20
2.3.2	Initialisation du treillis	20
2.3.3	Mémoire des chemins et décodage en temps réel	21
2.3.4	Renormalisation des métriques	22
CHAPITRE III :	ARCHITECTURE VLSI DE L'ALGORITHME DE VITERBI	25
3.1	Opérations de l'algorithme	25
3.2	Procédure de simulation et environnement	27
3.3	Architecture parallèle	30
3.3.1	Principe du parallélisme	30
3.3.2	Structure du détecteur parallèle	35
3.3.3	Simulations	40

	3.3.4	Performances et discussion	42
	3.4	Architecture semi-parallèle	48
	3.4.1	Principe du semi-parallélisme	48
	3.4.2	Structure du détecteur semi-parallèle	51
	3.4.3	Performances et discussion	54
	3.5	Architecture pipeline	56
	3.5.1	Principe de l'approche pipeline	56
	3.5.2	Structure du processeur pipeline	58
	3.5.3	Simulations	63
	3.5.4	Performances et discussion	67
	3.6	Comparaison des approches	68
CHAPITRE IV:	MÉMOIRE DES CHEMINS		72
	4.1	Problème de mise à jour	72
	4.2	Cellule de base	73
	4.3	Fonctionnement d'une matrice	76
	4.4	Conception de la puce	81
	4.4.1	Description globale	81
	4.4.2	Conception des composants de la puce	83
	4.4.2.1	Cellule de mise à jour de l'historique	83
	4.4.2.2	Registre des signaux de sélection	87
	4.4.2.3	Circuits tampon	89
	4.4.3	Organisation de la puce	90
CHAPITRE V :	CONCLUSION ET RECOMMANDATIONS		95
	5.1	Conclusions	95
	5.2	Recommandations sur la logique d'historique	96
	5.3	Algorithme de Viterbi et mémoire du canal	96
BIBLIOGRAPHIE			99
ANNEXE A			102
ANNEXE B			112

ANNEXE C 114

ANNEXE D 115

ANNEXE E 118

LISTE DES FIGURES

Figure		Page
1.1	Caractéristiques d'amplitude et de délai du réseau téléphonique	2
1.2	Détecteur à maximum de vraisemblance	3
2.1	Modèle du système de transmission PAM à travers un canal à bande de fréquence limitée	7
2.2	Modèle discret du canal	11
2.3	Choix des chemins survivants	17
2.4	Organigramme de l'algorithme de Viterbi	19
3.1	Opérations de base de l'algorithme	26
3.2	Etape de conception de circuit	28
3.3	Un niveau de treillis avec recirculation	32
3.4	Décomposition en sous-treillis	33
3.5	Un niveau de sous-treillis avec recirculation	34
3.6	Détecteur parallèle pour mémoire égale à 2	36
3.7	Processeur ACS parallèle	37
3.8	Diagramme de synchronisation de l'architecture parallèle	43
3.9	Treillis (mémoire = 3) modifié pour l'architecture semi-parallèle	49
3.10	Treillis (mémoire = 4) modifié selon le principe du semi-parallélisme	50
3.11	Détecteur semi-parallèle (mémoire = 3)	52
3.12	Processeur ACS semi-parallèle représentant 2 sous-treillis	53
3.13	Principe de l'approche pipeline appliqué à l'algorithme de Viterbi	57
3.14	Détecteur pipeline	59

3.15	Processeur pipeline (mémoire = 3)	60
3.16	Etages pipelines	61
3.17	Synchronisation des étages pipelines	61
3.18	Canal (mémoire = 3)	64
3.19	Treillis correspondant au canal ci-dessus	64
3.20	Diagramme de synchronisation du détecteur pipeline	65
3.21	Détecteur pipeline de mémoire égale à 9	71
4.1	Connexions des sous-treillis	74
4.2	Cellule de base pour la mise à jour de la mémoire des chemins	74
4.3	Matrice 3x2 représentant la mémoire des chemins	77
4.4	Circuit de mise à jour pour un récepteur de mémoire égale à 2	79
4.5	Schéma au niveau transistor de la cellule de base	84
4.6	Agencement des signaux de commande et d'alimentation de la cellule de base	86
4.7	Schéma bloc d'une cellule de registre	88
4.8	Configuration des transistors d'une cellule de registre	89
4.9	Organisation interne de la puce	92
4.10	Dessins des masques de la puce	94
5.1	Alternative pour le problème de mise à jour de la mémoire des chemins	97

LISTE DES TABLEAUX

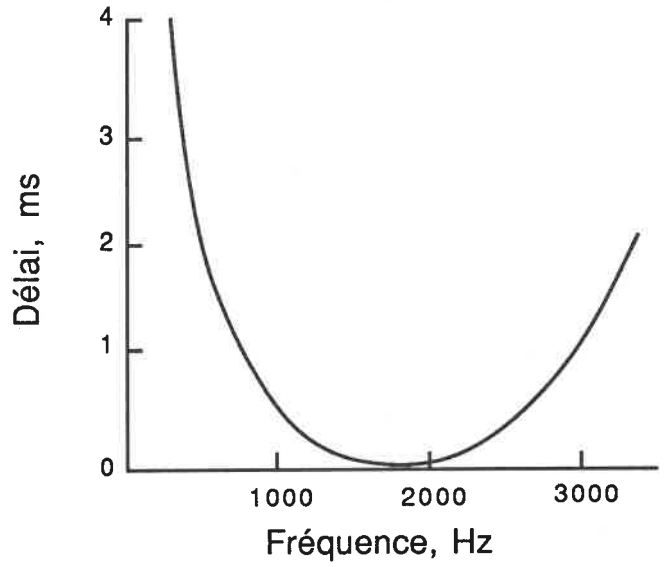
Tableau		Page
3.1	Contenu des tables de métriques de branches	41
3.2	Résultats de la simulation de l'architecture parallèle	44
3.3	Estimation de la séquence transmise	45
3.4	Résultats partiels de la simulation de l'architecture pipeline	66
3.5	Comparaison des approches	69

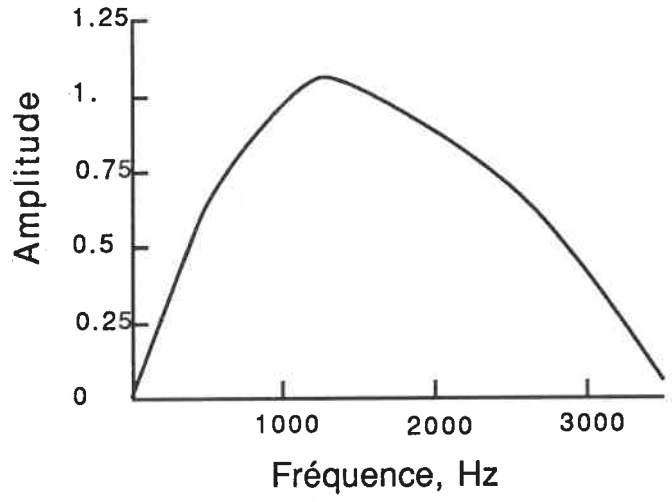
CHAPITRE I

INTRODUCTION

Le but premier de tout système de communication numérique est d'acheminer l'information vers le destinataire avec un maximum de fiabilité. En réalité l'information fournie à la sortie du canal de transmission peut être entachée d'erreurs. Ces anomalies sont dues au bruit intrinsèque au canal qui vient corrompre le signal. Parmi les canaux disponibles en communications numériques, le réseau téléphonique constitue celui qui est le plus utilisé. Ce type de canal particulier utilise une bande de fréquence non-idéale, dont la largeur est limitée. Ceci a pour conséquence que, pour un système de transmission numérique à modulation PCM ou PAM, une série d'impulsions traversant un tel canal n'est plus reconnaissable à la sortie du fait des distorsions d'amplitude et de délai engendrées par la réponse en fréquence du canal (figure 1.1). De fait les symboles transmis s'étalent dans le temps et recouvrent plusieurs symboles adjacents, causant ainsi de l'interférence entre symboles. L'abréviation d'ISI (de l'anglais "Intersymbol Interference") sera utilisée tout au long de ce mémoire pour désigner ce phénomène.

Une bonne technique de détection des signaux transmis consiste alors à tenir compte des effets de l'ISI en plus du bruit venant s'ajouter au signal d'information. Dans un système de transmission numérique à modulation PAM perturbé simultanément par le phénomène d'ISI ainsi que par l'ajout d'un bruit blanc gaussien, le détecteur à maximum de vraisemblance se compose d'un filtre adapté à bruit blanc suivi d'un processeur non-linéaire que l'on peut réaliser en utilisant l'algorithme de Viterbi (figure 1.2). L'application de cet algorithme pour la détection des symboles en présence d'ISI à été introduite par Forney [1]. Il

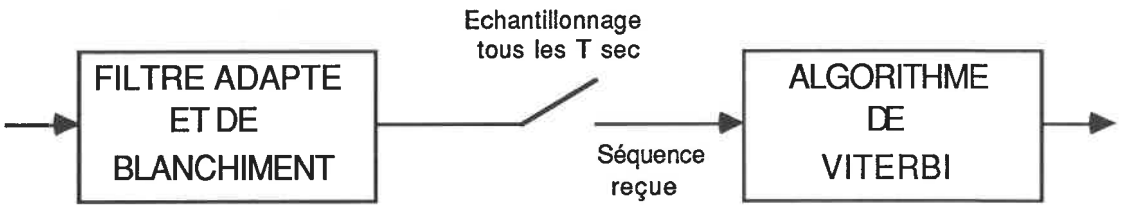




Caractéristiques d'amplitude et de délai

du réseau téléphonique

Figure 1.1



Détecteur à maximum de vraisemblance

Figure 1.2

a démontré en particulier que sa performance, quelle qu'en soit le critère, est au moins aussi bonne que pour n'importe quelle autre technique d'estimation. Dans beaucoup de cas pratiques, son efficacité est telle que le système global se comporte comme si le phénomène d'ISI n'était pas présent.

De façon générale, l'algorithme de Viterbi possède une structure simple et très régulière; ceci en fait un candidat de prédilection pour une implantation pratique en circuits intégrés. Une telle intégration présenterait des avantages certains si on tient compte des limitations de performances importantes rencontrées dans un système de microprocesseurs. En plus d'être compact et de posséder une vitesse appréciable, un tel système offrirait le potentiel d'un entretien facile, et d'un coût minimum, tout en gardant partiellement la flexibilité d'un système programmable.

Le projet présenté dans ce mémoire consiste à étudier différentes architectures pour une réalisation pratique en circuits intégrés de l'algorithme de Viterbi appliqué à la correction des erreurs de transmission numérique causées par le phénomène d'interférence entre symboles. Des simulations logiques ont servi à valider et analyser les diverses approches proposées. L'étude expérimentale a été consacrée à la réalisation ITGE (Intégration à Très Grande Echelle), en technologie CMOS 3 microns, d'une logique pour la mise à jour de la mémoire nécessaire au bon déroulement des opérations de corrections. Ce circuit constitue un des différents modules requis pour réaliser un détecteur de Viterbi.

L'organisation du mémoire se résume comme suit. Le chapitre deux place le lecteur dans le contexte auquel s'appliquera l'algorithme de Viterbi. Une description théorique du

phénomène d'ISI ainsi que de l'algorithme optimal de détection est nécessaire avant d'aborder les structures du détecteur qui sont présentées au chapitre trois. La conception même ainsi que la réalisation pratique du circuit considéré par ce projet sont exposées au chapitre quatre. Finalement nous tirons quelques conclusions et discutons des extensions possibles dans le dernier chapitre.

CHAPITRE II

INTERFERENCE ENTRE SYMBOLES ET ALGORITHME DE VITERBI

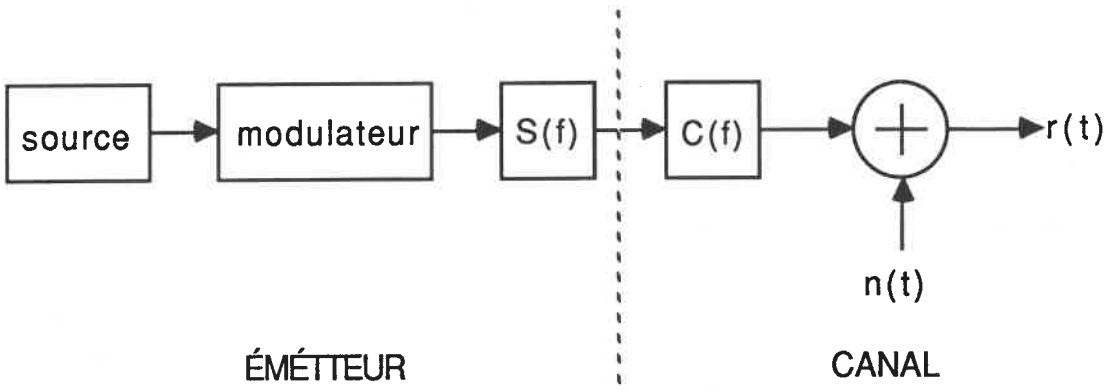
2.1 Interférence entre symboles

2.1.1 Modèle de transmission du canal PAM

Le modèle de transmission PAM (de l'anglais "Pulse Amplitude Modulation") utilisé dans cette recherche est illustré par la figure 2.1. Une source génère des symboles d'information choisis équiprobablement parmi les éléments d'un alphabet binaire {-1,+1}. Toute la théorie de ce chapitre considère, sans perte de généralité, le cas d'une source binaire. $\{u_N\}$ représente la séquence d'information générée par la source dirigée vers le modulateur PAM. Ce dernier peut être modélisé à partir d'un modulateur PAM idéal de réponse impulsionnelle $\partial(t)$, $\partial(t)$ étant une impulsion idéale, suivi d'un filtre s(t) de réponse en fréquence S(f). La transmission s'effectue à un taux de 1/T symboles par seconde. La séquence possède une longueur finie N. Nous supposons également que N est assez grand de manière à ce que les effets de début et de fin de séquence soient négligeables. Le signal appliqué à l'entrée du canal, résultat du modulateur, est alors:

$$\sum_{i=0}^{N-1} u_i \, . \, s(t - iT) \, . \qquad (2.1)$$

Le canal est représenté par un filtre de réponse impulsionnelle c(t) et de réponse



Modèle du système de transmission PAM à travers

un canal à bande de fréquence limitée

Figure 2.1

en fréquence C(f). Du bruit blanc gaussien de moyenne nulle et de variance $N_0/2$ vient s'ajouter à la sortie du filtre. Il en résulte le signal suivant:

$$r(t) = \sum_{i=0}^{N-1} u_i \, . \, h(t - iT) + n(t) \qquad (2.2)$$

$$\text{où } h(t) = s(t) * c(t). \qquad (2.3)$$

h(t) représente la sortie du filtre linéaire c(t) en réponse au signal s(t).

A partir de ce modèle, une technique de détection basée sur le critère du maximum de vraisemblance est élaborée. Voyons d'abord en quoi consiste la détection à maximum de vraisemblance.

2.1.2 Détection optimale à maximum de vraisemblance

L'objectif du système de réception est de minimiser la probabilité d'erreur pour une séquence transmise. Une règle de détection qui suit ce principe est dite optimale [2]. Si toutes les séquences d'entrée sont équiprobables, c'est-à-dire si

$$P[u_i=-1] = P[u_i=+1] = 0.5,$$

alors la technique d'estimation basée sur le critère du maximum de vraisemblance est

optimale. La règle du maximum de vraisemblance établit l'estimation $\{\hat{u}_N\}$ à partir d'une séquence reçue $\{y_N\}$ si la probabilité conditionnelle d'obtenir la séquence $\{y_N\}$ sachant que la séquence $\{\hat{u}_N\}$ a été envoyée, que nous écrirons $P[\{y_N\} \mid \{\hat{u}_N\}]$, est supérieure à n'importe quelle autre probabilité conditionnelle $P[\{y_N\} \mid \{u_N\}]$, pour toutes les séquences possibles $\{u_N\}$. Il faut donc maximiser la probabilité à posteriori $P[\{y_N\} \mid \{u_N\}]$ sur l'ensemble des séquences $\{u_N\}$ pour espérer avoir la meilleure estimation. Il peut être démontré [3] que la maximisation de cette probabilité, dans un environnement affecté de bruit blanc gaussien de moyenne nulle et de variance $N_0/2$, se ramène à la maximisation du logarithme du rapport de vraisemblance qui est défini de la manière suivante:

$$J(\{u_N\}) = (2/N_0)\int v(t)r(t)\,dt \; - \; (1/N_0)\int v(t)^2\,dt\,, \tag{2.4}$$

où v(t) représente le signal reçu non corrompu par le bruit,

$$v(t) = \sum_{i=0}^{N-1} u_i \,.\, h(t - iT)\;. \tag{2.5}$$

Utilisant (2.5), on peut réécrire la relation (2.4) sous la forme

$$J(\{u_N\}) = \quad (2/N0) \sum_{i=0}^{N-1} u_i \int h(t - iT)r(t)\,dt \; -$$

$$(1/N0) \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} u_i\, u_j \int h(t - iT)h(t - jT)\,dt. \tag{2.6}$$

Par raison de clarté et simplicité, on définit les notations suivantes:

$$Z_i = \int h(t - iT)r(t)\, dt, \qquad (2.7)$$

$$\text{et} \quad f_{i-j} = \int h(t - iT)h(t - jT)\, dt. \qquad (2.8)$$

En utilisant (2.7) et (2.8) dans (2.6), et en multipliant $J(\{u_N\})$ par la constante $-1/N_0$, maximiser $J(\{u_N\})$ revient donc à minimiser $I(\{u_N\})$, où

$$I(\{u_N\}) = -2 \sum_{i=0}^{N-1} u_i Z_i + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} u_i u_j f_{i-j}. \qquad (2.9)$$

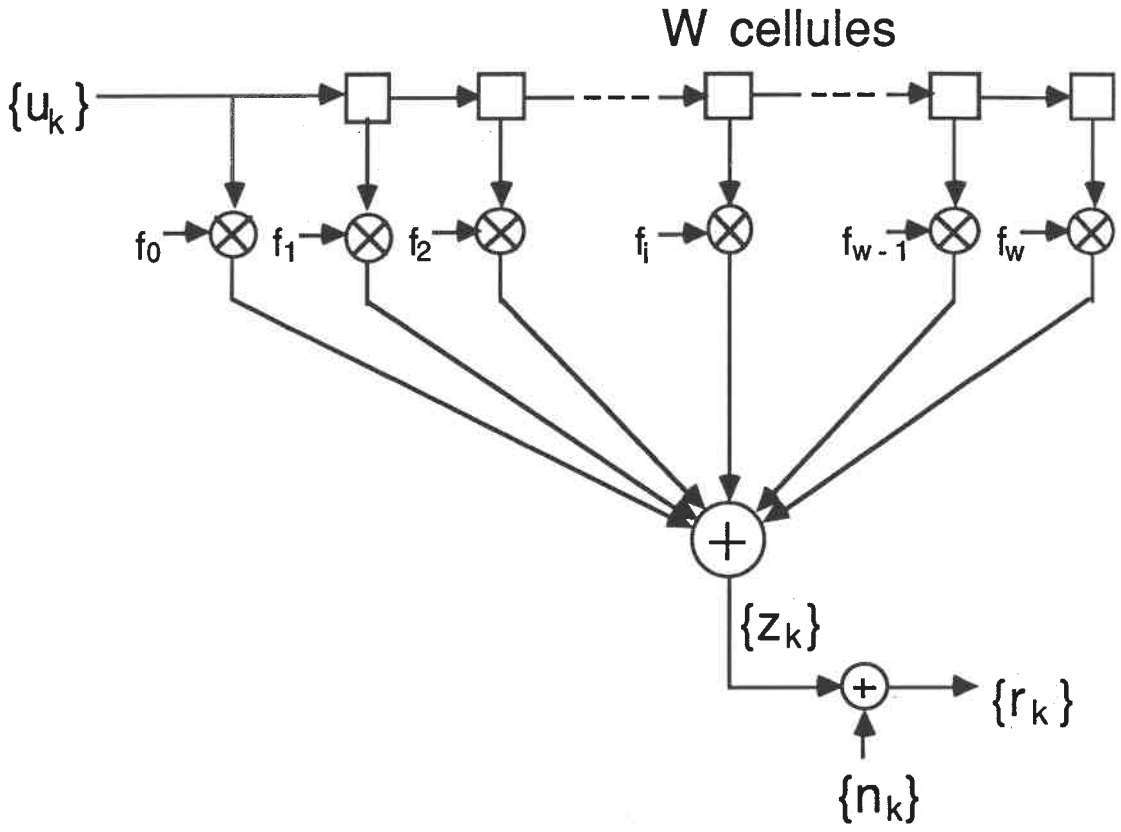
Il est maintenant nécessaire de clarifier quelques points. Le signal h(t) possède une durée finie. En dehors de l'intervalle [0,(W+1)T], la valeur du signal p(t) est nulle. On définit W comme étant la mémoire du canal. De la relation (2.8), nous avons alors

$$f_i = 0 \quad \text{pour } |i| > W. \qquad (2.10)$$

Les coefficients f_i peuvent donc prendre leur valeur sur un ensemble fini. Tous ces coefficients peuvent être calculés à partir de la fonction h(t) qui est connue. Ils possèdent également la symétrie suivante :

$$f_{-i} = f_i, \quad \text{pour tous } i. \qquad (2.11)$$

On peut donc définir un modèle discret du canal illustré par la figure 2.2. De leur coté, les coefficients Z_i sont les sorties d'un filtre adapté au signal h(t) à un taux de 1/T observations



Modèle discret du canal

Figure 2.2

par seconde. L'information fournit par ces échantillons est suffisante pour nous permettre le calcul de $I(\{u_N\})$. Les échantillons Z_i forment donc un ensemble de statistiques suffisantes pour une estimation optimale à maximum de vraisemblance.

L'estimation suivant le critère du maximum de vraisemblance nécessite le calcul de $I(\{u_N\})$ pour toutes les séquences $\{u_N\}$ possibles. Le filtre adapté du récepteur doit être suivi d'un détecteur qui va trouver la séquence $\{\hat{u}_N\}$ qui maximise $I(\{u_N\})$. Ce calcul exhaustif sur l'ensemble des séquences est irréaliste. Néanmoins, il est possible d'utiliser à la réception un algorithme récursif, soit l'algorithme de Viterbi, permettant de faire un choix de manière simple et pratique. Voyons d'abord comment déduire de la théorie décrite ci-dessus une procédure qui faciliterait l'application d'un tel algorithme.

2.1.3 Définition du treillis de décodage

Il est possible de décomposer $I(\{u_N\})$ en utilisant (2.11) sous la forme suivante :

$$I(\{u_N\}) = \{-2 \sum_{i=0}^{N-2} u_i Z_i + \sum_{i=0}^{N-2} \sum_{j=0}^{N-2} u_i u_j f_{i-j}\}$$

$$+ \{ -2\, u_{N-1} Z_{N-1} + 2u_{N-1} \sum_{i=N-1-W}^{N-2} u_i f_{N-1-i} + (u_{N-1})^2 f_0 \}. \qquad (2.12)$$

Le premier terme possède une forme équivalente à celle de (2.9). La différence provient du fait que l'effet du dernier symbole reçu u_{N-1} a été reporté dans le second terme. On remarque

que ce deuxième terme ne dépend pas de la séquence $\{u_N\}$ complète mais seulement des W+1 derniers symboles. On définit d'abord la variable vectorielle

$$S_i = (u_{i-1}, \ldots , u_{i-W}) \quad \text{pour } i = W, \ldots, N-1 , \tag{2.13}$$

ainsi que les quantités suivantes :

$$\mu_k(S_W, \ldots, S_k) = -2 \sum_{i=0}^{k-1} u_i Z_i + \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} u_i u_j f_{i-j} \quad \text{et} \tag{2.14}$$

$$m_{k+1}(S_k,S_{k+1}) = -2 u_k Z_k + 2 \sum_{i=k-W}^{k-1} u_i f_{k-i} + (u_i)^2 f_0 \ . \tag{2.15}$$

Si on réécrit la relation (2.12) en utilisant ces notations, on obtient

$$I(\{u_N\}) = \mu_N(S_W, \ldots, S_N) = \mu_{N-1}(S_W, \ldots, S_{N-1}) + m_N(S_{N-1},S_N). \tag{2.16}$$

En répétant cette décomposition jusqu'au bout, la relation précédente s'écrit alors

$$I(\{u_N\}) = \mu_W(S_W) + m_{W+1}(S_W,S_{W+1}) + m_{W+2}(S_{W+1},S_{W+2}) + \ldots$$
$$+ m_{N-1}(S_{N-2},N\text{-}1_1) + m_N(S_{N-1},S_N) . \tag{2.17}$$

On remarque que le signal reçu r(t), pour un instant donné t, est fonction d'un ensemble fini de symboles $\{u_t, u_{t-1}, \ldots, u_{t-W}\}$ de longueur W+1. Les W symboles précédant u_t ont été définis plus

tôt par le symbole S_i. S_i est appelé l'état du canal. à chaque transmission d'un symbole, le canal passe d'un état S_i à un autre état S_{i+1}. La transmission d'une séquence de symboles peut être vu comme une séquence d'états parmi lesquels le canal transite. L'estimation de la séquence la plus vraisemblable est équivalente à une estimation de la séquence d'états la plus vraisemblable.

On peut maintenant définir un treillis dans lequel à chacun de ses N-W niveaux se retrouvent 2^W états. Deux branches émergent de chaque état le reliant ainsi à deux autres états du niveau suivant. Les branches représentent les transitions d'états du canal. Une métrique m_k, telle que définie par la relation (2.15), est associée à chaque branche. Minimiser $I(\{u_N\})$ revient alors à minimiser la somme des termes m_k pour la séquence de symboles reçus. La détection optimale se ramène donc à la recherche de la séquence d'états la plus vraisemblable à travers un treillis, et donc du chemin de métrique minimum généré par la séquence de symboles la plus vraisemblable. Il est donc possible d'appliquer l'algorithme de Viterbi à la correction du phénomène d'interférence entre symboles. Pour chaque symbole reçu, une métrique est calculée (selon l'expression (2.15)) pour chaque transition possible d'un état vers un autre, et ceci, pour chaque état d'un niveau.

2.2 Description de l'algorithme de Viterbi

Il a été démontré par Omura [4], que l'algorithme de Viterbi peut être considéré comme une méthode de solution par programmation dynamique du problème de recherche optimale à travers un graphe, ou treillis pondéré pour lequel on associe à chaque branche un

poids ou métrique. L'algorithme met à profit la propriété de reconvergence des chemins du treillis. En effet, à chaque état du treillis deux chemins reconvergent. Il s'ensuit que l'on peut rejeter, sans perte d'optimalité, le chemin de métrique cumulative maximum (ou minimum, dépendant du choix des métriques). En effet, puisque les chemins subséquents sont identiques, il est clair que l'extension du chemin possédant la métrique la plus grande ne peut être optimale; sa métrique cumulative sera toujours plus grande que celle du chemin survivant à n'importe quel niveau supérieur, puisque la différence entre les deux métriques des extensions subséquentes sera toujours constante. En répétant cette opération pour chaque état et chaque profondeur du treillis, l'algorithme de Viterbi effectue donc bien une recherche exhaustive de tous les chemins possibles.

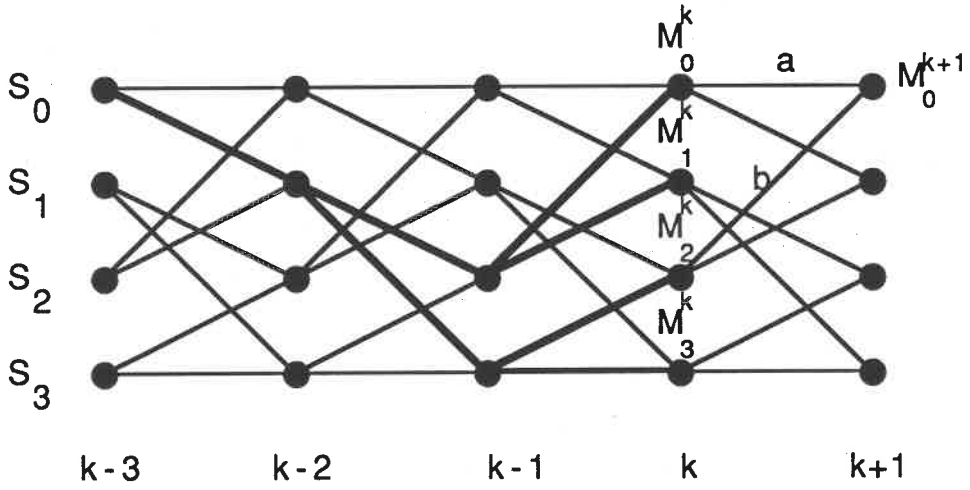
Nous décrivons maintenant de façon plus explicite l'algorithme complet dans le cas d'une transmission au travers un canal de mémoire W. L'état initial de début de transmission au travers du canal est évidemment l'état nul 0, et le récepteur débute sa recherche dans le même état. Le récepteur dispose d'une table de métriques de branche spécifiant les symboles de sorties quantifiés du canal correspondant aux digits d'entrée possibles, -1 et +1, pour chaque état.

Pour le premier symbole reçu, le récepteur le compare aux différents symboles possibles contenus dans la table correspondant à chaque branche du treillis. La différence entre le symbole et le poids de chaque branche (métriques partielles ou de branches) est ainsi calculée. Les résultats, ainsi que le bit d'entrée associé, sont ensuite conservés pour chacun des états. Chaque nouveau symbole reçu est traité pareillement, à la différence que la métrique cumulative appartenant à l'état précédent est ajoutée à la métrique partielle pour former la

métrique cumulative de cet état. Après W extensions dans le treillis, toutes les valeurs d'états ont été visitées et possèdent une métrique cumulative. C'est ainsi que chaque état a emmagasiné la distance "euclidienne" entre les symboles reçus et les symboles représentés par le chemins dans le treillis menant à chaque état depuis l'état initial nul. L'historique, correspondant aux digits successifs qu'aurait générés la source pour arriver à cet état, est également conservé.

A partir de la profondeur W+1, il s'avère que deux chemins convergent vers chaque état. A chacune des métriques de ces deux chemins est additionnée la différence entre le nouveau symbole reçu et le symbole associé à l'extension des chemins respectifs. On garde ensuite le chemin de plus petite métrique appelé survivant, et on rejette l'autre (figure 2.3). On répète cette suite d'opérations pour les W états de profondeur W+1, et par la suite la procédure est répétée pour tous les symboles reçus. En cas d'égalité entre les deux métriques de deux chemins reconvergents, le survivant peut être choisi au hasard sans perte d'optimalité.

Après la transmission complète d'une séquence de N bits, une queue formée de W "-1" est transmise au travers du canal de façon à replacer le canal dans son état initial nul. Donc à partir de la profondeur N, seuls les chemins correspondant à des bits d'entrée -1 seront prolongés. Ceci à pour conséquence que le nombre d'états utilisés dans le treillis est réduit par un facteur 2 à chaque extension. On se retrouvera donc à la fin de la queue avec un seul état qui est l'état nul et un seul survivant. La séquence d'entrée la plus vraisemblable est alors la séquence de symboles qui génère ce chemin survivant.



$$M_0^{k+1} = \text{MIN}(M_0^k + a, M_2^k + b)$$

Choix des chemins survivants

Figure 2.3

Dans la partie récursive de l'algorithme, correspondant aux profondeurs supérieures à W, l'algorithme de Viterbi (figure 2.4) se résume aux étapes suivantes [6]:

Pour chaque état et à tous les niveaux,

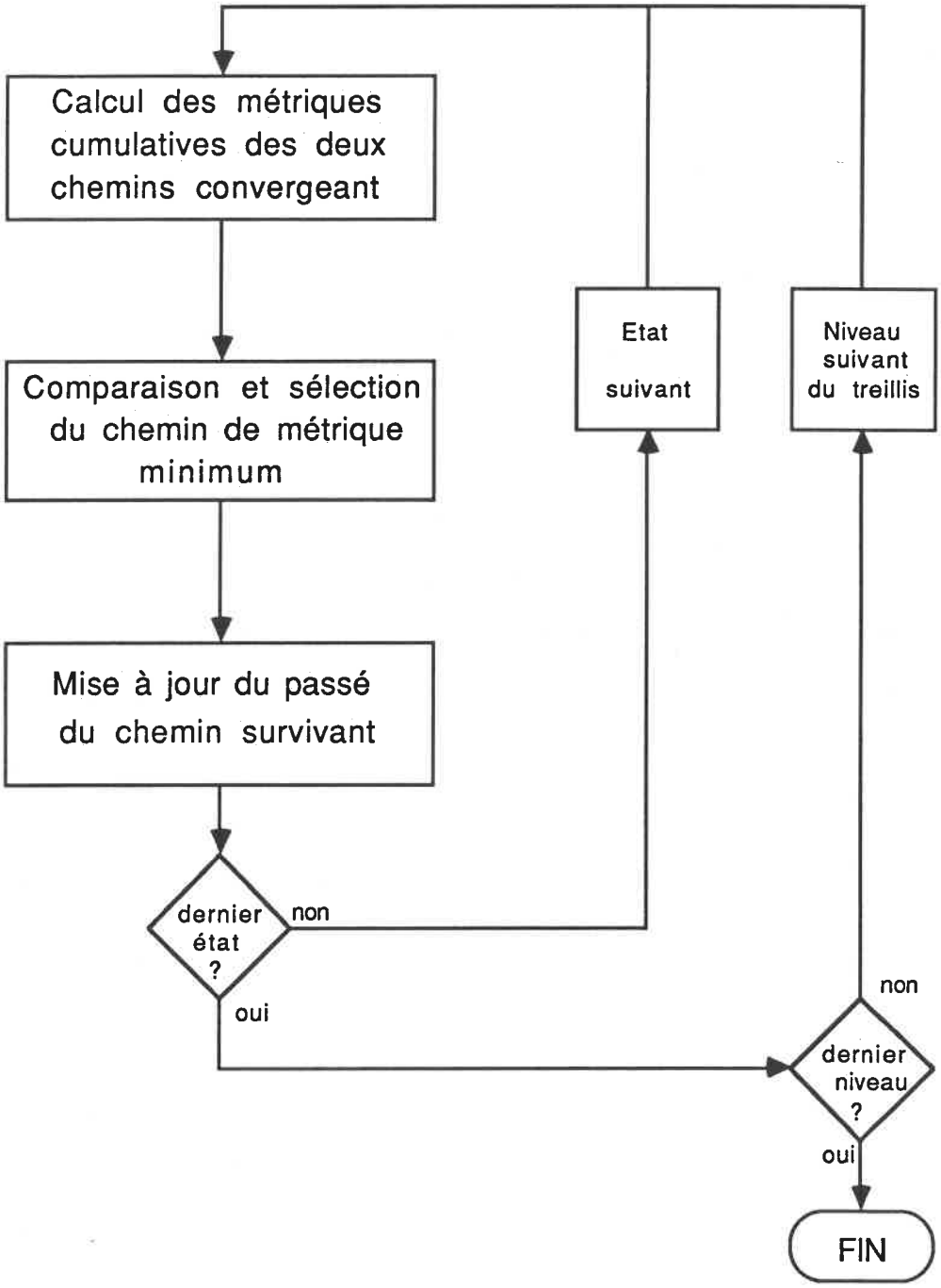
1. Calcul des métriques cumulatives des deux chemins convergeant vers cet état, en ajoutant aux métriques précédentes les métriques partielles des branches conduisant à cet état.

2. Comparaison des métriques des deux chemins et élimination du chemin de métrique maximale. Celle-ci devient alors la nouvelle métrique de l'état.

3. Mise à jour de l'historique du chemin en ajoutant à celui du chemin survivant le bit d'information correspondant à la branche de transition.

2.3 Modifications nécessaires pour une réalisation pratique

La description précédente de l'algorithme, bien que satisfaisante du point de vue théorique n'est pas adéquate pour une implantation pratique. Sans considérer l'implantation à très grande échelle, les simplifications qui suivent doivent être considérées.



Organigramme de l'algorithme de Viterbi

Figure 2.4

2.3.1 Quantification des métriques

Du point de vue pratique, l'utilisation de la valeur réelle de la métrique n'est pas satisfaisante de sorte qu'une quantification entière est nécessaire pour un traitement efficace de l'information. Cette approximation entraîne une dégradation dont l'importance dépend principalement du nombre de bits de quantification. De plus, au lieu de calculer la métrique de branche à chaque extension, il est plus simple d'utiliser une table (mémoire morte ROM) faisant la correspondance entre le symbole quantifié reçu et la métrique partielle pour un état donné.

2.3.2 Initialisation du treillis

En se référant à la section 2.2, on constate que les opérations de l'algorithme de Viterbi sont répétitives durant l'extension du treillis, à l'exception de l'initialisation et de la resynchronisation de la transmission. Si on pouvait éliminer ces deux étapes, le récepteur pourrait opérer de façon répétitive sans se soucier de la profondeur du treillis à laquelle il opère.

L'argument suivant [6] permet de court-circuiter l'opération d'initialisation. Supposons, qu'au départ on assigne une métrique nulle à l'état "0" et une métrique de valeur "infini" (en pratique une valeur très grande sera suffisante) à tous les autres états. Il s'ensuit que tous les états non nuls se trouvent biaisés par rapport à l'état "0" au début du processus d'estimation. Au bout de W extensions les chemins issus de ces états auront une métrique de valeur "infini et ne pourront survivre. En conséquence, seuls les chemins issus de l'état "0"

seront gardés par l'algorithme ce qui équivaut à la séquence d'initialisation.

2.3.3 Mémoire des chemins et décodage en temps réel

Dans l'hypothèse du décodage de séquence, le récepteur doit posséder une capacité énorme d'emmagasinage de l'information. Il doit en effet conserver les $N.2^W$ bits nécessaires pour représenter tous les chemins correspondant à une séquence transmise de N bits. De plus le récepteur doit attendre la fin de la transmission de la séquence complète avant de délivrer l'information. Ce délai peut ne pas être acceptable dans un système de communication en temps réel.

Des simulations [7] ont cependant montré qu'il n'est pas nécessaire de conserver toute cette information. De fait, les bits de l'estimation peuvent commencer à être délivrés aussitôt que la profondeur de recherche a atteint une longueur de 4 ou 5 fois la mémoire W, et ceci sans perte significative sur les performances. On doit cette propriété au fait que les chemins survivants ont tendance avec probabilité proche de 1 à être issus d'un même état commun situé à une profondeur de au plus 4 à 5 fois la mémoire. Il s'ensuit que l'on peut donc réduire considérablement la quantité de mémoire requise, et par la même occasion éliminer la nécessité d'insérer une queue à la fin de la transmission. En effet, après un nombre de cycles de décodage de l'ordre de 4 à 5 W, l'usager peut commencer à recevoir l'information sans attendre la fin de la séquence. Evidemment, dans ce cas le récepteur ne fonctionne plus de façon optimale, mais il ne nécessite plus qu'un historique de chemins réduit à 4 ou 5 W et peut fonctionner de façon continue.

Dans ce cas, l'étape correspondant à la mise à jour de la mémoire des chemins est cruciale dans le processus d'estimation. Elle sera considérée dans un chapitre subséquent où on exposera les principes d'une logique dédiée à cette opération.

2.3.4 Renormalisation des métriques

Finalement, dans le cas d'une estimation continue de l'information, une dernière modification doit être apportée à l'algorithme pour le rendre réalisable. En effet, au fur et à mesure de l'estimation, les métriques cumulatives s'accumulent et ne cessent de croître. Les capacités des registres de métriques doivent donc prendre des proportions tout à fait irréalistes pour l'estimation de séquences transmises très grandes. Cependant il s'avère qu'en pratique, l'écart des variations entre les métriques cumulatives minimum et maximum possibles est borné [8] par la valeur:

$$W.\partial_{max} \qquad (2.18)$$

où W est la mémoire du canal,

et ∂_{max} est la métrique de branche maximum.

En choisissant un nombre suffisant de bits pour la capacité des registres des métriques, on peut donc éviter tout débordement, en soustrayant de toutes les métriques cumulatives la métrique minimale à chaque pas de l'algorithme. Ceci implique cependant une certaine recherche parmi les métriques pour trouver la plus petite. En plus, la soustraction systématique ne fait que ralentir la vitesse totale du récepteur.

On peut cependant éviter ces opérations répétitives laborieuses. Une solution plus simple consiste à soustraire périodiquement de toutes les valeurs de métriques, une constante judicieusement choisie. Cette opération de renormalisation entre en exécution lorsque toutes les métriques dépassent le seuil. Grâce à ce processus, les valeurs des métriques garderont une amplitude raisonnable en évitant tout débordement de registres.

Pour déterminer la valeur du seuil de renormalisation, on peut considérer l'argument suivant. On sait que l'écart entre la métrique de valeur maximum Δ_{max} et celle de valeur minimum Δ_{min} ne dépassera jamais W. ∂_{max}. On a lors la relation suivante:

$$\Delta_{max} \leq \Delta_{min} + W.\,\partial_{max}\,. \qquad (2.19)$$

Le choix du seuil doit respecter les contraintes suivantes pour éviter un débordement. Tout d'abord, le seuil doit avoir une valeur supérieure à la valeur de métriques de branche maximum ∂_{max}. De cette façon il est possible d'empêcher une éventuelle croissance des métriques cumulatives en soustrayant plus vite que le taux de croissance maximum. Ensuite, il faut s'assurer que les registres de métriques cumulatives possèdent une capacité assez grande pour emmagasiner la somme du seuil et de l'écart maximum entre les métriques extrêmes. On peut résumer les critères de choix du seuil de renormalisation par les relations suivantes:

$$1.\ \text{seuil} \geq \partial_{max} \qquad (2.20)$$

$$2.\ \text{seuil} \leq (2^M - 1) - (W+1).\partial_{max} \qquad (2.21)$$

où M est la longueur des registres de métriques cumulatives,

$(2^M - 1)$ représente la capacité d'un registre,

$W.\ \partial_{max}$ est la valeur maximum de l'écart entre les métriques cumulatives extrêmes.

On remarque que dans le deuxième critère, une valeur supplémentaire ∂_{max} a été ajoutée. Cette valeur compense l'effet de la renormalisation un cycle en retard. La relation (2.21) peut être reformulée pour nous permettre de déterminer la longueur minimum requise pour un registre en fonction du seuil et de la métrique de branche maximum. On a alors

$$M \geq \text{Log}_2\left(\text{seuil} + (W+1).\partial_{max} + 1\right). \qquad (2.22)$$

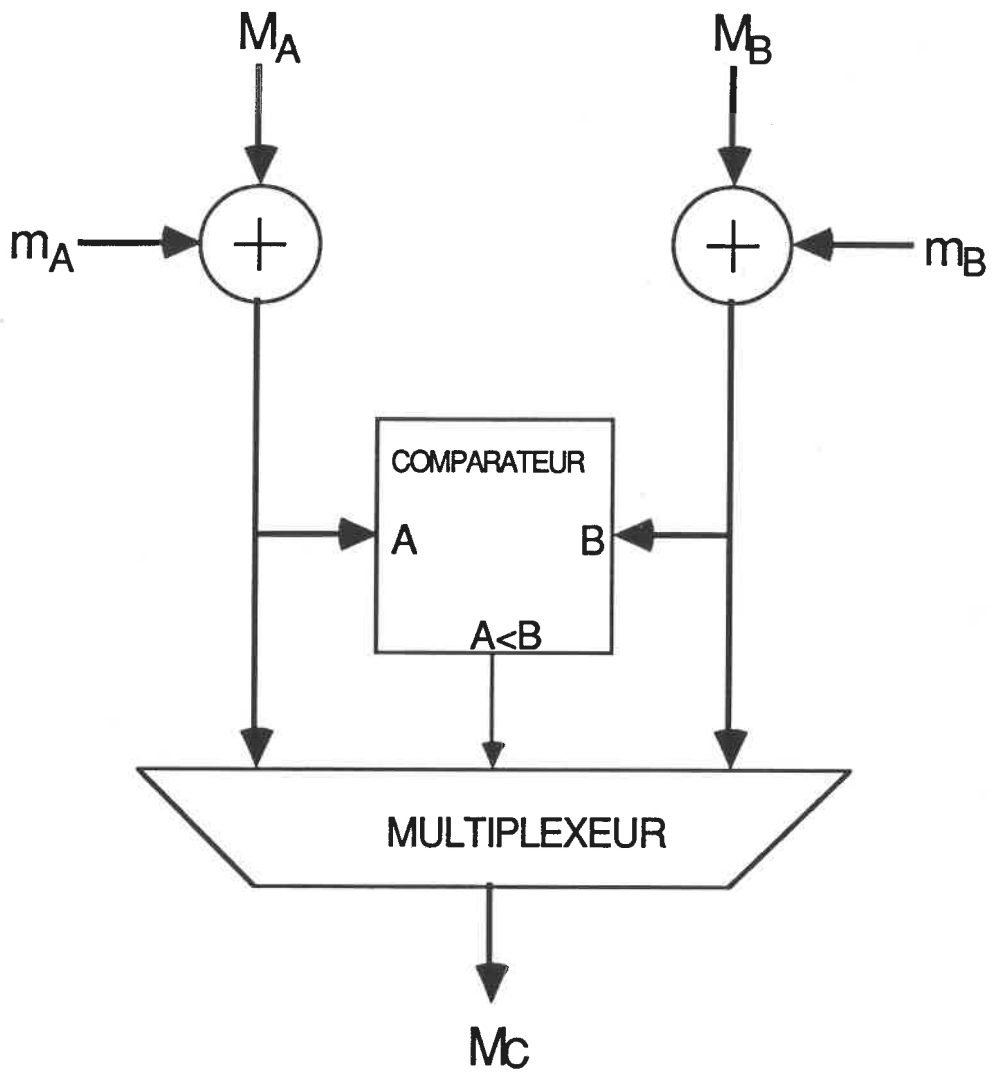
CHAPITRE III

ARCHITECTURE VLSI DE L'ALGORITHME DE VITERBI

3.1 Opérations de l'algorithme

L'algorithme de Viterbi peut être divisé en trois opérations de base, qui sont des opérations d'additions, de comparaisons et de sélections (ACS). La figure 3.1 représente le diagramme bloc des opérations de l'algorithme. Pour chaque état, les métriques partielles de branches m_a et m_b sont ajoutées aux deux métriques cumulatives M_A et M_B des états prédécesseurs. Les résultats sont comparés et la valeur la plus petite est conservée pour former M_C. Le processus est répété à chaque niveau de profondeur du treillis et pour chacun des 2^W états où W est la mémoire du treillis. Cette régularité simplifie grandement l'intégration de ces opérations.

Plusieurs concepts d'architectures d'ordinateurs peuvent être appliqués à l'algorithme de Viterbi pour exploiter ses caractéristiques. L'utilisation d'une architecture parallèle s'appliquant à chaque état permet d'avoir un maximum de vitesse. A l'autre extrême, les 2^W opérations requises par unité de temps peuvent être réalisées de façon sérielle et, dans ce cas, l'architecture pipeline est intéressante. Ces deux approches seront examinées tour à tour pour tirer profit des caractéristiques de l'algorithme. Une variante de l'approche parallèle que nous nommerons semi-parallèle sera également mentionnée. L'étude portera sur le comportement des modèles des circuits suggérés et leur réponse aux différents stimuli permettra la validation des circuits lors des simulations.



Opérations de base de l'algorithme

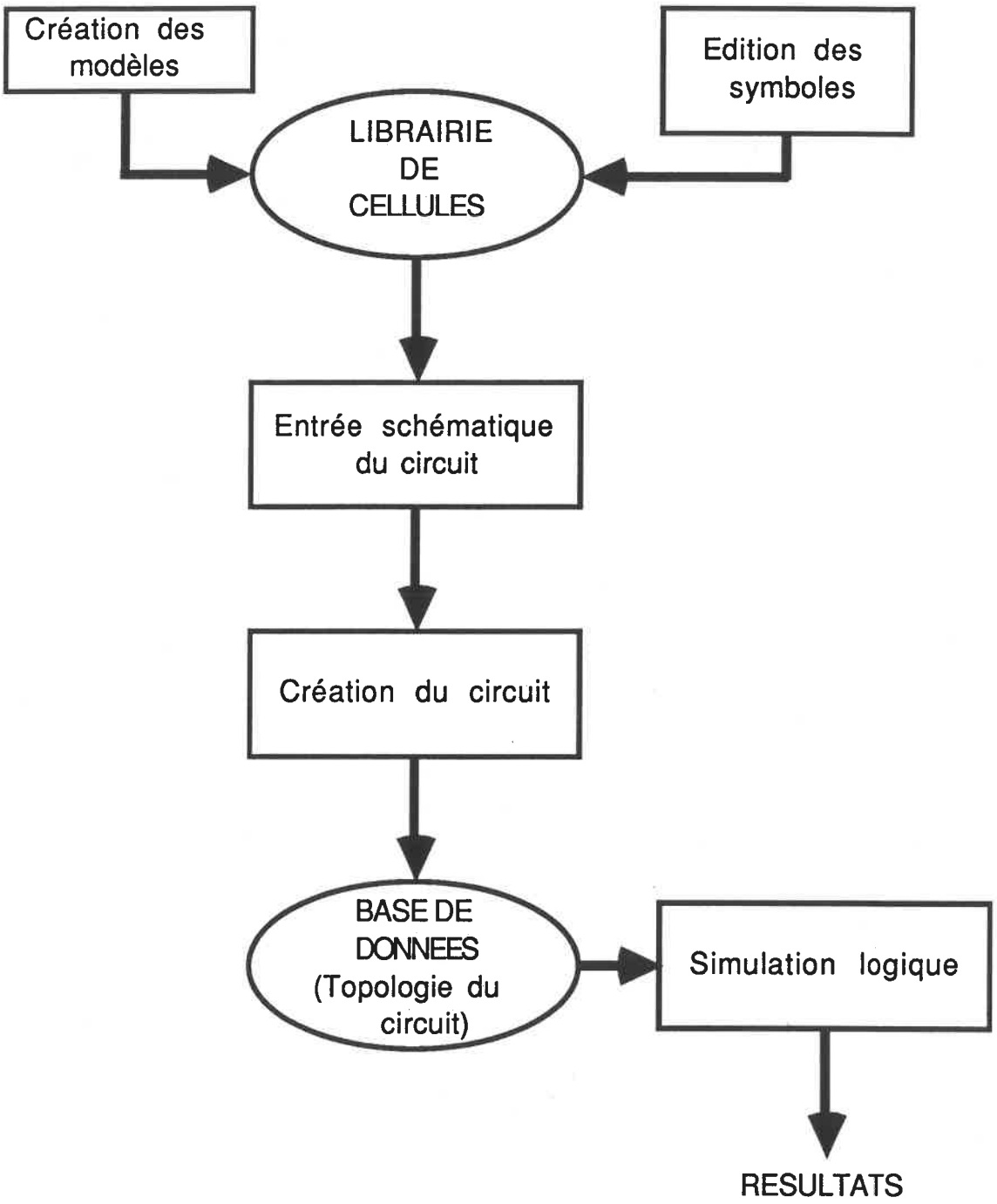
Figure 3.1

Mentionnons que la disponibilité des métriques de branches caractérisant le canal est prise pour acquise tout au long de l'étude. Elles sont supposées précalculées et contenues dans des tables (mémoires mortes) accessibles par les différents processeurs. Elles pourraient être également calculées en temps réel à la réception de chaque symbole reçu par l'intermédiaire d'un processeur de métriques. Quant à la logique de mise à jour de l'historique des chemins survivants, son étude détaillée sera considérée au chapitre suivant.

3.2 Procédure de simulation et environnement

Les circuits tels que réalisés seront analysés grâce à un système de conception de circuits assisté par ordinateur. Ce système modulaire assiste le concepteur durant toutes les étapes de la conception, depuis la description du circuit jusqu'à la validation logique et le dessin des masques utilisés pour la fabrication des circuits intégrés. Seuls les modules d'entrée schématique de circuits, de modélisation de symboles, ainsi que le simulateur logique seront utiles à cette étude. Le diagramme de la figure 3.2 résume les étapes de conception de circuits dans cet environnement de CAO (références [9] à [12]).

Suite à la conception de l'architecture, la première étape préalable aux simulations, consiste à modéliser les différents circuits [13]. La modélisation peut se faire à différents niveaux hiérarchiques, en fonction des besoins et du niveau d'abstraction visé par le concepteur. C'est ainsi qu'un circuit peut être vu comme un ensemble d'éléments de base tel des additionneurs, des registres, et des portes logiques simples, soit à l'autre extrême, comme une seule cellule dont le fonctionnement est toutefois plus complexe. Dans le dernier cas, tout regroupement de plusieurs cellules en une seule fait en sorte que la topologie du circuit est



Etape de conception de circuit

Figure 3.2

simplifiée. Il s'ensuit que le simulateur doit observer moins d'éléments, et ses performances s'en trouvent ainsi améliorées. Cette option est nécessaire dans le cas où le nombre de cellules est trop grand par rapport à la capacité du simulateur. Le détecteur pipeline sera composé d'un ensemble de composants de base, tandis que le détecteur parallèle sera composé exclusivement de cellules plus complexes représentant les processeurs.

Pour une validation logique, seul le comportement fonctionnel est pertinent; La disposition interne des transistors est de fait inutile. Dans ce cas la modélisation se situe alors a un niveau d'abstraction plus élevé. Les circuits peuvent être construits à partir d'éléments de base disponibles dans une librairie du système de CAO dont les éléments ont été préalablement modélisés pour enlever le poids de ce fardeau à l'utilisateur. Cependant, lorsque le niveau d'abstraction dépasse celui de ces cellules de base, il est possible de créer de nouveaux éléments. Un symbole représentant l'élément est d'abord défini et, par la suite, une fonction de simulation lui est associée. Cette fonction décrit le comportement logique de la cellule au moyen d'un langage de programmation de haut niveau (PASCAL ou C) tout en respectant les règles de structure interne de description de circuit propre au simulateur. Cette fonction après compilation est appelée par le simulateur chaque fois qu'un changement d'états des noeuds d'entrée de la cellule survient, entraînant une mise à jour des niveaux de ses noeuds de sortie.

Après la modélisation, il est nécessaire de spécifier à la base de données de l'ordinateur les schémas électriques des circuits. Durant cette opération d'entrée schématique, l'usager appelle les éléments du circuit et les connecte convenablement. Puis la topologie du circuit est ensuite extraite et modifiée dans un état utilisable par le module de simulation. Le modèle du circuit est maintenant utilisable pour des fins de simulations. Il suffit simplement

de lui appliquer les vecteurs de validation nécessaires à la vérification de son bon fonctionnement et de sa bonne synchronisation . A l'aide de ces simulations il est donc possible de procéder à l'analyse des approches suggérées.

3.3 Architecture parallèle

3.3.1 Principe du parallélisme

L'application simultanée des opérations de base de l'algorithme (Additions-Comparaison-Sélection) sur les 2^W états du treillis optimise la vitesse du décodeur. Idéalement une structure entièrement parallèle pourrait être réalisée en associant à chaque état du treillis complet (i.e., à tous les niveaux de profondeur) un groupe de processeurs. Tous les processeurs du niveau actuel de décodage recevraient l'information, (i.e., les métriques cumulatives et les séquences survivantes), des processeurs les précédant dans la hiérarchie. Malheureusement la complexité d'une telle architecture est proportionnelle au nombre de niveau de treillis de décodage, et donc à la longueur de la séquence transmise. Il s'ensuit donc que cette approche est irréalisable. Cependant, du fait que pour tous les niveaux du treillis la géométrie des connexions est invariante, on peut exploiter cette propriété en créant une structure similaire à un seul niveau du treillis et en insérant les connexions de retour nécessaires pour recirculer l'information nouvellement mise à jour. Ainsi nous avons un groupe de 2^W registres et 2^W processeurs. Le contenu d'un registre représente la valeur de la métrique cumulative de l'état qui lui est associée. Les 2^W processeurs servent aux calculs des nouvelles métriques cumulatives. Ils reçoivent leur information des registres de prédécesseurs pour générer un nouvel ensemble de métriques. Ces valeurs sont alors recirculées vers les registres de survivants

correspondant. La figure 3.3 illustre ce concept.

Considérons maintenant la décomposition suivante du treillis. Le contenu de la mémoire du canal peut être représenté comme suit:

$$\underline{m}.d_s \qquad (3.1)$$

où d_s est le symbole qui sortira de la mémoire au prochain coup d'horloge,

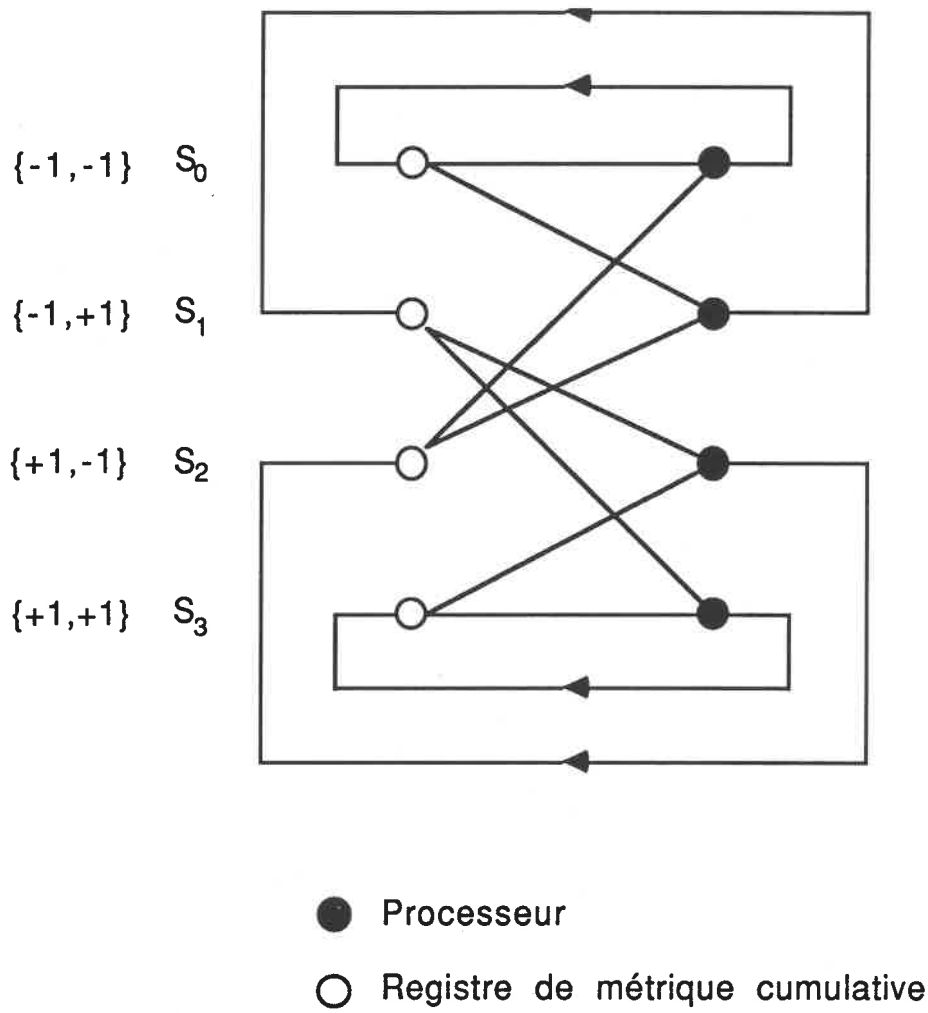
$\underline{m}$ représente le vecteur des (w-1) autres symboles de mémoire.

Tous les états du treillis, 2^W possibles, peuvent être représentés par cette notation. Lors de l'entrée d'un symbole d_e, le dernier symbole d_s est alors perdu, et l'état de la mémoire devient

$$d_e.\underline{m}. \qquad (3.2)$$

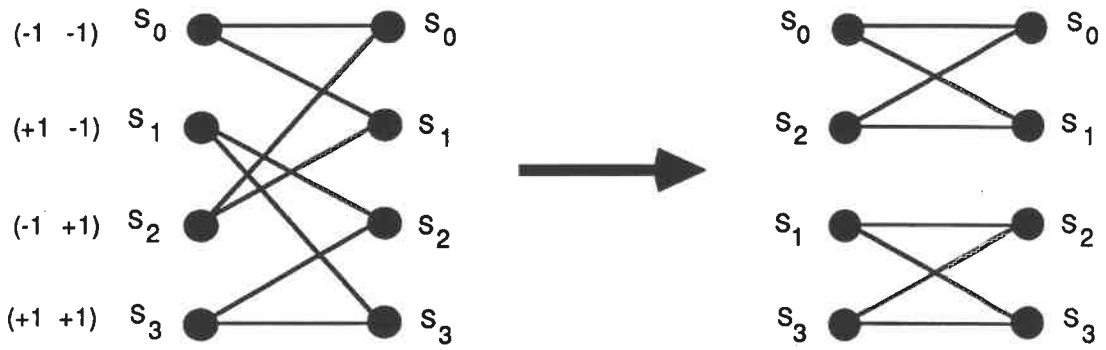
Chaque état $S=\underline{m}.d_s$ peut générer deux autres états $S_0=-1.\underline{m}$ et $S_1=+1.\underline{m}$ appelés états successeurs. Chaque état $S=d_e.\underline{m}$ possède également deux prédécesseurs, soient les états $\underline{m}.-1$ et $\underline{m}.+1$. Deux états de même valeur $\underline{m}$ ($\underline{m}.-1$ et $\underline{m}.+1$) auront les mêmes états successeurs ($-1.\underline{m}$ et $+1.\underline{m}$) et ils ne différeront que par leur premier symbole [14]. Le treillis se trouve décomposé alors en sous-treillis comme à la figure 3.4. Dans ce cas, la figure 3.5 illustre le processus d'extension avec recirculation basé sur le concept des sous-treillis. Cette propriété des sous-treillis sera exploitée pour la conception de la logique de mise à jour de l'historique des chemins que nous verrons au chapitre 4.

Notre architecture parallèle sera composée d'un processeur ACS (Addition,



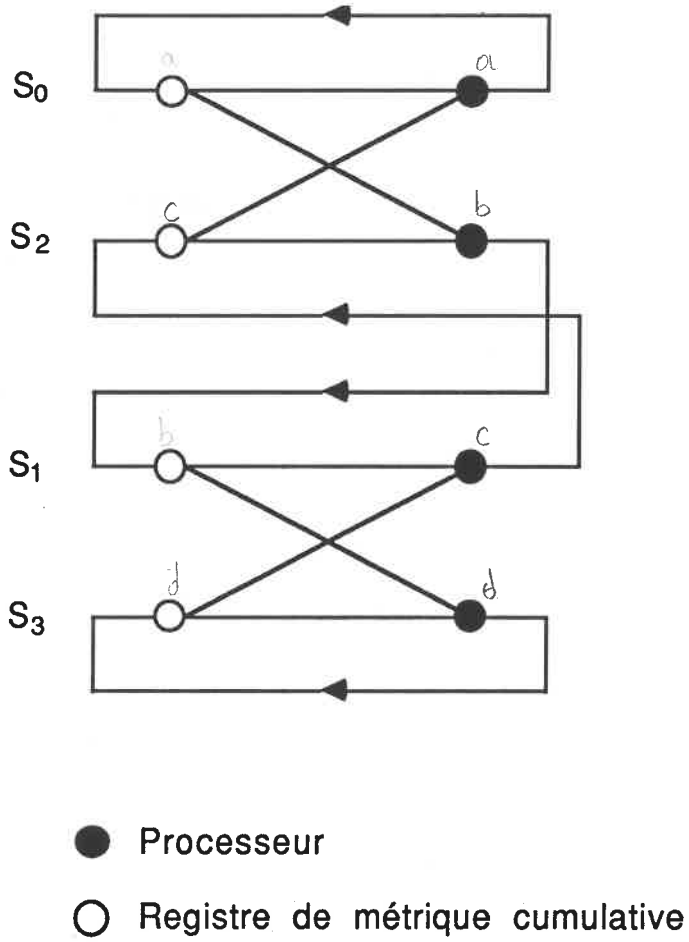
Un niveau de treillis avec recirculation

Figure 3.3



Décomposition en sous-treillis

Figure 3.4



Un niveau de sous-treillis avec recirculation

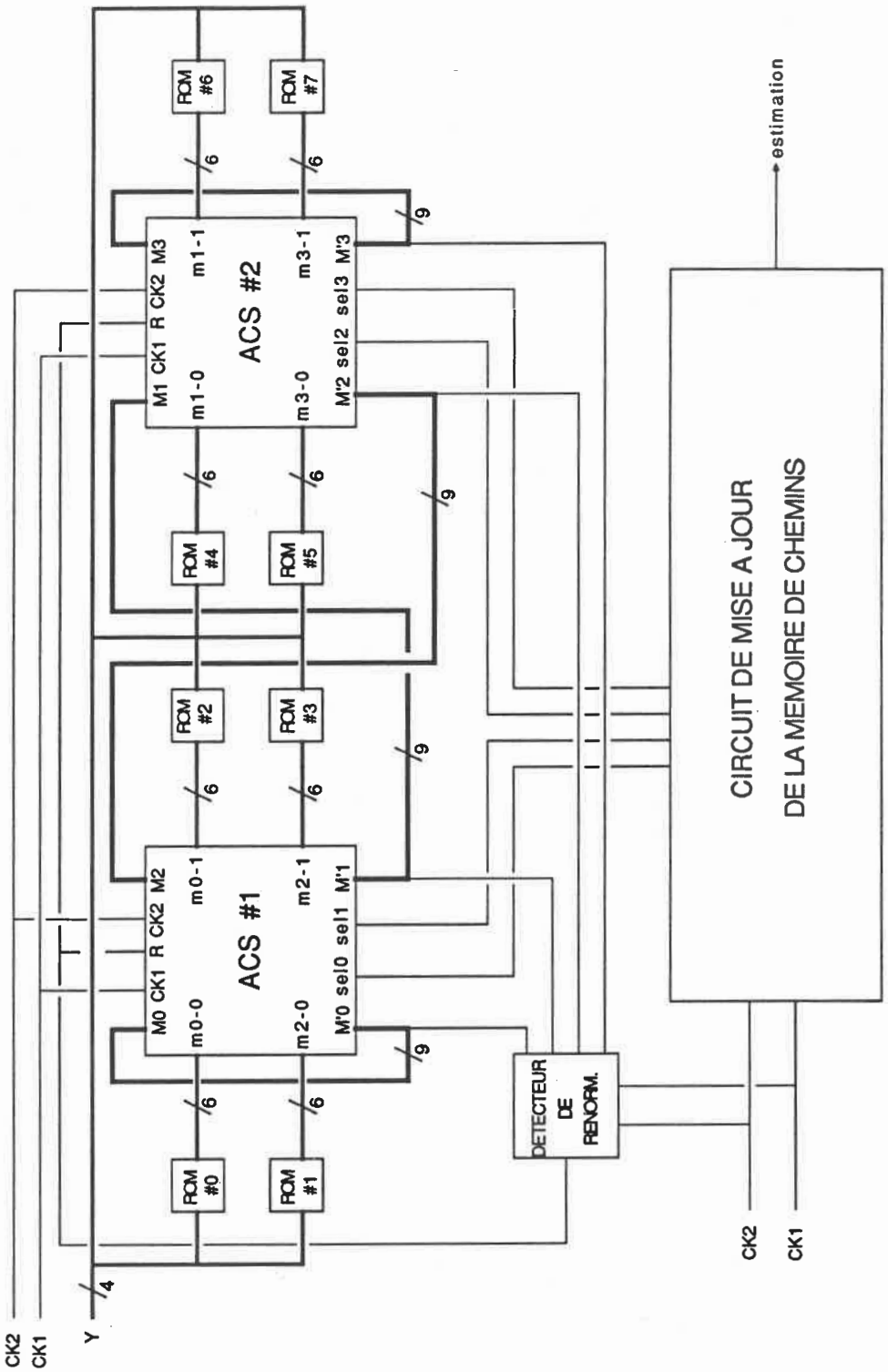
Figure 3.5

Comparaison, Sélection) par sous-treillis comme illustré à la figure 3.4. Une structure parallèle générale peut alors être imaginée pour intégrer l'algorithme de Viterbi.

3.3.2 Structure du détecteur parallèle

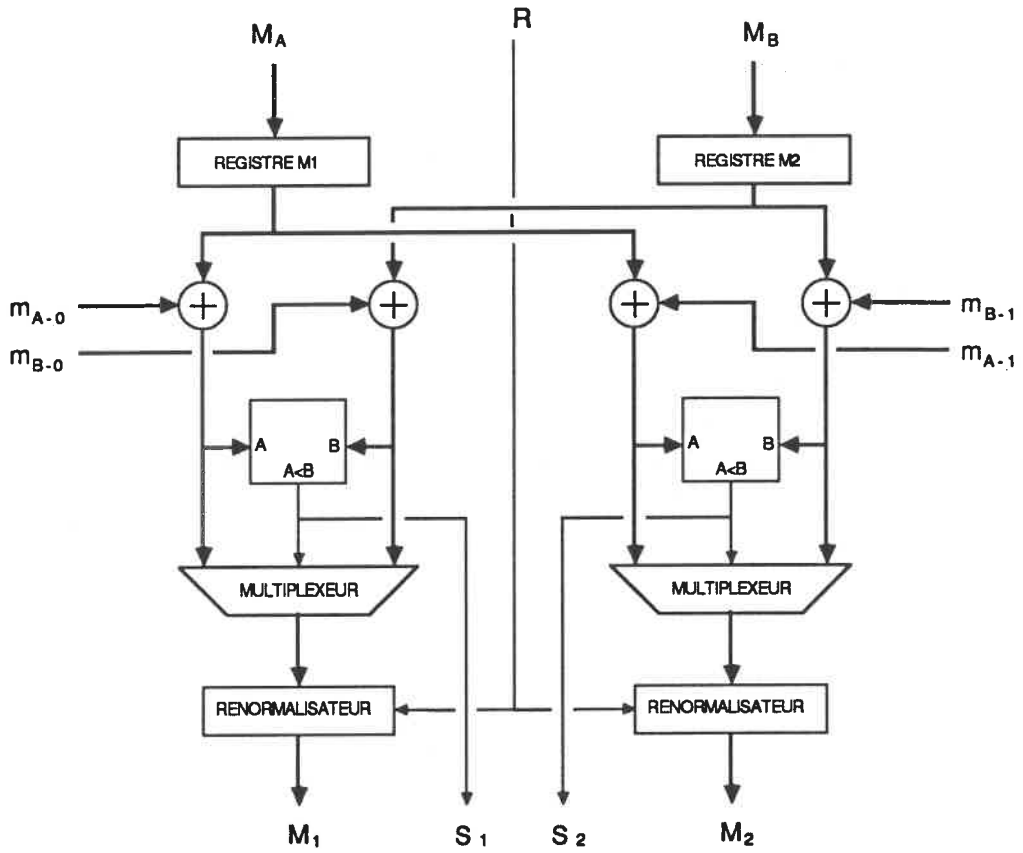
La structure appliquant le principe du parallélisme est très simple. Un processeur ACS par sous-treillis est nécessaire. D'autre part, quatre tables (mémoires mortes ou calculateurs de métriques) générant les métriques partielles de branches sont associées à chaque processeur. Finalement, une logique doit être prévue pour détecter les conditions de renormalisation des métriques cumulatives. La structure d'un détecteur parallèle complet appliqué à un canal de mémoire égale à 2 est illustrée à la figure 3.6.

Un ACS (figure 3.7) est constitué de deux registres contenant les valeurs de métriques cumulatives des deux états prédécesseurs d'un sous-treillis. Chaque additionneur possède comme entrées la valeur d'un des deux registres ainsi que la métrique partielle de branche correspondante provenant de l'extérieur de l'ACS. Les résultats des deux additions qui représentent les valeurs de métriques des deux chemins convergeant vers le même état, sont alors comparés, et la valeur la plus petite est conservée. Cette valeur est dirigée vers la sortie de l'ACS après passage au travers du circuit de renormalisation. Chacune des valeurs ainsi calculées est recirculée vers un des deux registres d'un ACS, comme indiqué par le principe de recirculation, pour servir de nouvelle métrique cumulative. Chaque comparateur génère également un signal correspondant à la sélection choisie. Ce signal de sortie de l'ACS est utilisé par le circuit de mise à jour de l'historique des chemins nécessaires dans la décision du signal décodé.



Détecteur parallèle pour mémoire égale à 2

Figure 3.6



Processeur ACS parallèle

Figure 3.7

Le circuit de renormalisation est excessivement simple puisqu'il ne comporte qu'une porte logique "ET" et une bascule. En effet, en se référant aux formules 2.20 et 2.21 de la section 2.3.4 du chapitre 2, on peut toujours simplifier l'apport matériel du circuit de renormalisation. En effet, pour le circuit simulé, on a choisi 6 bits de quantification pour les métriques de branches, et donc ∂_{max} est alors égal à 63. Alors on a la relation suivante:

$$63 \leq \text{seuil} \leq 2^M - 189 , \qquad (3.3)$$

où M est le nombre de bits assignés aux registres de métriques.

Si on choisit une valeur de seuil de 256, de la relation (2.21) on a alors $N \geq 8.8$, et donc le nombre de bits des registres de métriques cumulatives doit être d'au moins 9. Si on choisit N=9, alors la relation (3.3) devient:

$$63 \leq \text{seuil} \leq 323, \qquad (3.4)$$

de sorte qu'une valeur de seuil de 2^8=256 est acceptable. Il suffira donc simplement d'observer le bit le plus significatif de chacune des métriques calculées. Dès que tous ces bits sont égaux à 1, le seuil est alors soit atteint soit dépassé par toutes les métriques. C'est l'indication qu'il est temps de renormaliser. Le circuit détecteur de renormalisation se réduit à une porte "ET" détectant si le seuil est atteint par toutes les métriques, suivi d'un registre de 1 bit contrôlant la renormalisation le cycle suivant. De plus, il est clair que pour soustraire la valeur 256 d'un nombre entier de neuf bits supérieur ou égal à 256, il suffit de forcer le neuvième bit du registre à 0. Cette opération peut être effectuée simplement en masquant le neuvième bit des

métriques calculées pendant le cycle suivant.

Les tables contiennent les métriques partielles propres au canal. A chaque état doublé d'un symbole d'entrée 0 ou 1, est associée une mémoire morte. Cette table contient toutes les valeurs quantifiées possibles calculées d'après la relation (2.15). Le symbole reçu est également quantifié, et il suffit donc d'adresser par sa valeur la cellule correspondante de chaque table de métriques.

Finalement, la logique de mise à jour de la mémoire des chemins requiert seulement les signaux de sélection indiquant lequel parmi les deux chemins convergeant vers un état est le survivant. Il suffit donc d'un signal de sélection par état, soit 2^W au total.

La stratégie de synchronisation générale du système parallèle est relativement simple. Une horloge biphasée sans recouvrement est utilisée. Un signal d'initialisation est nécessaire pour mettre le détecteur dans un état initial valable. La seule autre entrée est la valeur quantifiée du symbole reçu. Une seule sortie suffit, soit celle fournie par le circuit d'historique qui se trouve être la décision du détecteur. A chaque coup d'horloge, un symbole reçu quantifié est disponible. Après le passage à travers l'ACS, les nouvelles métriques cumulatives ainsi que les signaux de sélection sont disponibles. Au coup d'horloge suivant, les valeurs sont emmagasinées dans les registres de métriques pour être ainsi disponibles pour de nouvelles opérations.

Examinons maintenant le fonctionnement d'un détecteur pour un canal de mémoire égale à 2 lors d'une simulation.

3.3.3 Simulation

La figure 3.6 représente le circuit simulé. Le canal auquel est appliqué ce circuit est celui de la figure 2.2. Les processeurs ACS ont été modélisés pour accélérer la simulation. Un modèle exact du comportement est décrit et se retrouve en annexe A. Les contenus des "roms" des tables de métriques correspondantes sont présentés au tableau 3.1. Le circuit a été simulé avec un signal d'horloge possédant une fréquence arbitraire de 5 MHz. De plus, les valeurs de délai des différents composants sont des approximations suffisantes pour les besoins de cette étude.

Supposons une source générant la séquence

$$\{u_k\}=\{+1,-1,+1,+1,-1,-1,-1,+1,-1,+1,+1,-1,+1,-1,+1,+1\} .$$

La séquence Y correspondant, sortie du canal (sans bruit gaussien, seulement l'interférence) est

$$\{y_k\}=\{-1,0,0,1,1,-1,-2,-1,0,0,1,1,0,0,0,1\} .$$

Comme la sortie doit être quantifiée, l'utilisation de 16 niveaux de quantification conduit à la séquence quantifiée

$$\{Y_k\}=\{3,7,7,11,11,3,0,3,7,7,11,11,7,7,7,11\} .$$

Cette séquence est envoyée au modèle du circuit. Il n'est évidemment pas nécessaire d'ajouter du bruit car la raison d'être de la simulation est simplement d'étudier le comportement du circuit durant l'estimation et non de valider l'efficacité de l'algorithme de Viterbi à combattre l'ISI. Une séquence d'initialisation précède la séquence $\{u_k\}$. Une queue de {-1} est également envoyée à la fin de la transmission pour recueillir l'estimation à la

	S_0		S_1		S_2		S_3	
y_k	- 1	+1	- 1	+1	- 1	+1	- 1	+1
0	0	3	15	35	3	15	35	63
1	1	2	11	29	2	11	29	55
2	2	1	8	24	1	8	24	48
3	3	0	5	19	0	5	19	41
4	5	0	3	15	0	3	15	35
5	8	1	2	11	1	2	11	29
6	11	2	1	8	2	1	8	24
7	15	3	0	5	3	0	5	19
8	19	5	0	3	5	0	3	15
9	24	8	1	2	8	1	2	11
10	29	11	2	1	11	2	1	8
11	35	15	3	0	15	3	0	5
12	41	19	5	0	19	5	0	3
13	48	24	8	1	24	8	1	2
14	55	29	11	2	29	11	2	1
15	63	35	15	3	35	15	3	0

Contenu des tables de métriques de branches

Tableau 3.1

sortie de la logique de mise à jour des chemins survivants. Ce délai est du au fait que ce circuit possède une mémoire de 13 et emmagasine les 13 derniers bits des chemins survivants avant de délivrer sa décision.

Le fichier des stimuli appliqués au circuit se trouve en annexe B. Les résultats de la simulation suivent: la figure 3.8 donne le diagramme de synchronisation des opérations, et les tableaux 3.2 et 3.3 donnent les résultats quantifiés des opérations par cycle d'horloge de la simulation. La première colonne de le tableau 3.2 représente les cycles d'horloge, et la deuxième colonne les symboles reçus correspondant. Les quatre colonnes suivantes permettent de suivre l'évolution des métriques cumulatives des chemins survivants durant l'estimation de la séquence reçue. Les quatre dernières colonnes donnent les valeurs des 4 signaux de sélection (un par état). Pour un état $d_e.\underline{m}$, une valeur de 0 indique que le chemin survivant provient de l'état $\underline{m}$.-1, sinon il provient de l'état $\underline{m}$.+1 pour une valeur de 1. Le tableau 3.3 résume l'estimation de la séquence (le symbole 0 est équivalent au symbole -1). Comme on peut le voir, il a été nécessaire d'attendre 14 coups d'horloge avant d'avoir le premier bit de l'estimation à la sortie de la logique d'historique. La séquence estimée $\{\hat{u}_k\}$ est donc bien égale à $\{u_k\}$. Etudions maintenant les performances physiques du circuit.

3.3.4 Performances et discussion

Les opérations se font sur tous les sous-treillis simultanément, de sorte qu'en un coup d'horloge, toutes les opérations sont réalisées. La vitesse de ces opérations est déterminée par le temps d'accès du contenu d'un "rom" ajouté au temps de propagation à travers un

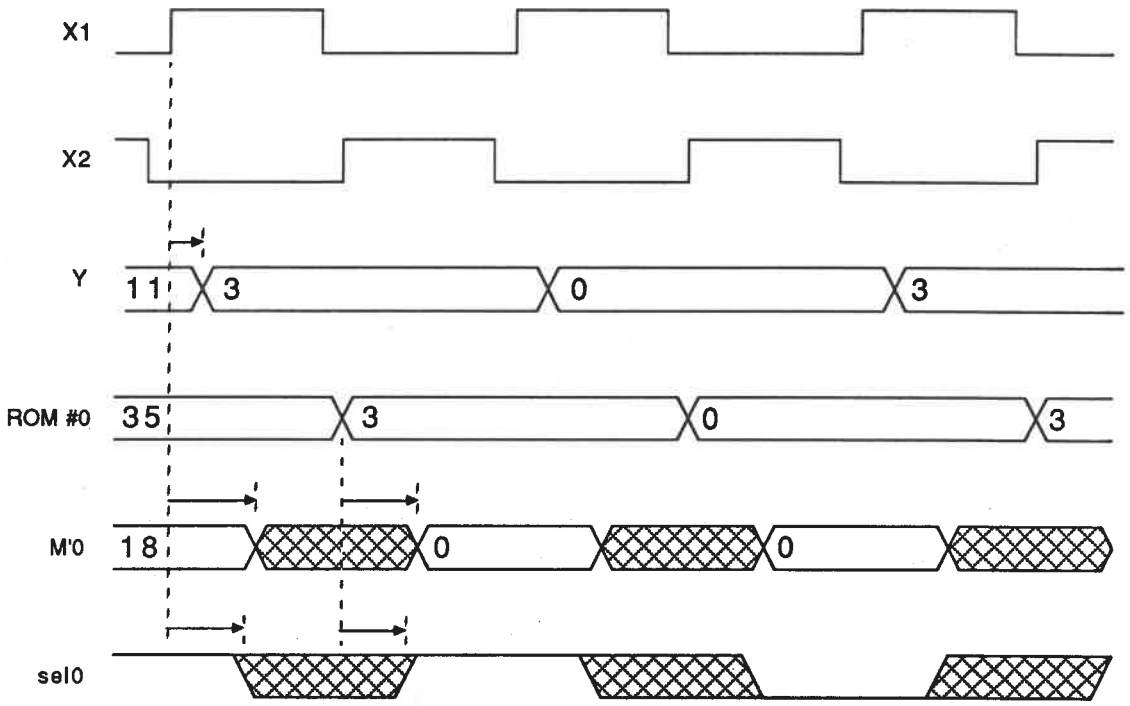


Diagramme de synchronisation de l'architecture parallèle

Figure 3.8

cycle d'horloge	Y	M'0	M'1	M'2	M'3	sel0	sel1	sel2	sel3
1	0	0	3	15	35	0	0	0	0
2	0	0	3	18	38	0	0	0	0
3	0	0	3	18	38	0	0	0	0
4	0	0	3	18	38	0	0	0	0
5	0	0	3	18	38	0	0	0	0
6	0	0	3	18	38	0	0	0	0
7	3	3	0	8	22	0	0	0	0
8	7	11	6	0	5	1	0	0	0
9	7	3	0	6	11	1	1	0	0
10	11	21	9	3	0	1	1	0	0
11	11	18	6	0	5	1	1	1	1
12	3	0	5	11	25	1	1	0	0
13	0	0	3	20	40	0	0	0	0
14	3	3	0	8	22	0	0	0	0
15	7	11	6	0	5	1	0	0	0
16	7	3	0	6	11	1	1	0	0
17	11	21	9	3	0	1	1	0	0
18	11	18	6	0	5	1	1	1	1
19	7	3	0	6	11	1	1	0	0
20	7	9	6	0	5	1	0	0	0
21	7	3	0	6	11	1	1	0	0
22	11	21	9	3	0	1	1	0	0
23	11	18	6	0	5	1	1	1	1
24	3	0	5	11	25	1	1	0	0

Résultats de la simulation de l'architecture parallèle

Tableau 3.2

cycle d'horloge	estimation
14	0
15	0
16	0
17	0
18	0
19	0
20	1
21	0
22	1
23	1
24	0
25	0
26	0
27	1
28	0
29	1
30	1
31	0
32	1
33	0
34	1
35	1

Estimation de la séquence transmise

Tableau 3.3

processeur. Il est donc facile d'imaginer des systèmes parallèles de plusieurs dizaines de mégabits/seconde, dépendant de la technologie mise à la disposition pour intégrer un pareil circuit. Les simulations ont montré l'efficacité et surtout la régularité, voire simplicité, d'une telle architecture.

Un autre attrait de cette approche en plus de sa rapidité, réside dans le fait que la vitesse est indépendante de la mémoire du canal qui fixe le nombre de processeurs utilisés. La fréquence maximale possible est approximativement

$$V_{max} = (t_r + t_p)^{-1}, \qquad (3.5)$$

où t_r est le temps d'accès aux tables de métriques partielles, et t_p le temps de propagation à travers un ACS. Par contre, t_r a tendance à augmenter avec la taille de la mémoire.

Par contre le nombre d'ACS augmente exponentiellement avec la mémoire du canal, et donc, ce qui ne semble pas affecter la vitesse globale du circuit affecte sa superficie. La complexité des processeurs et les dimensions des tables de métriques caractérisant le canal ne sont cependant pas les facteurs déterminant de cette croissance exponentielle. Bien qu'il soit préférable d'augmenter le nombre de niveaux de quantification lorsqu'on désire s'attaquer à un canal de mémoire plus grande, entraînant par le fait même un accroissement de la superficie des mémoires mortes, seuls le nombre de processeurs et de tables avec leurs interconnexions ont un rôle prépondérant dans cette approche.

Cette croissance exponentielle entraîne également une croissance de la complexité du routage global entre les processeurs caractérisée par les interconnexions entre les ACS nécessaires à la recirculation des métriques cumulatives. Des études ont été faites sur ce sujet [15]; des techniques d'optimisation de routage utilisant les techniques du "shuffle-exchange" et du "cube-connected-cycle" sont disponibles. Ces techniques supportées par la théorie n'ont pas aidé concrètement à diminuer l'impact de ce problème. Toutefois on verra qu'il est possible de sauver de la complexité dans le routage grâce à notre approche de mise à jour des chemins survivants. Chaque processeur génère simplement deux signaux de sélection (un par état) et les dirige vers le circuit de mise à jour. Comparativement, les techniques d'optimisation considèrent que le routage de chaque processeur comprend en plus de la métrique cumulative calculée, la description du chemin tronqué survivant à l'état associé au processeur (soit, dans notre exemple, 13 bits supplémentaires à faire recirculer). Notre approche réduit de façon impressionnante l'apport du routage puisque ces 13 bits se résume à un seul bit de sélection. Cependant il n'est pas possible d'éliminer les limitations spatiales intrinsèques à l'approche parallèle qui reste limitée à des mémoires inférieures à 7. La réalisation la plus impressionnante à ce jour constitue le décodeur de Viterbi conçu par une équipe de Japonais [16], intégré sur une seule puce, permettant le décodage de code convolutionnel de taux 1/2 et de longueur de contrainte 7 (équivalent à une mémoire égale à 6). La technologie utilisée est basée sur l'utilisation d'un procédé expérimental CMOS 1 micron.

A titre de conclusion, on peut résumer la situation du décodeur parallèle comme suit. La vitesse est optimale, mais la complexité augmente très vite de sorte que les limites sont atteintes très rapidement, même en utilisant des outils de conception et de synthèse puissants. Il y a donc un compromis à faire entre la vitesse du détecteur pour combattre l'ISI

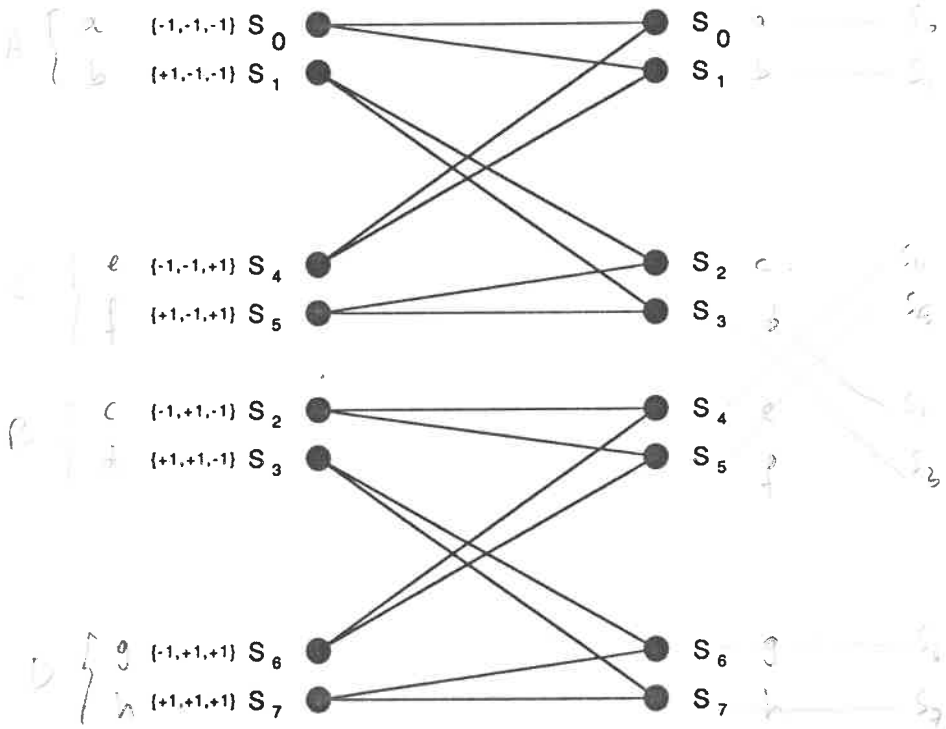
et sa complexité. C'est le principe de base de l'architecture semi-parallèle que nous envisageons maintenant.

3.4 Architecture semi-parallèle

3.4.1 Principe du semi-parallélisme

Cette approche représente un bon compromis entre la vitesse et la complexité du circuit. Comme il a été mentionné précédemment, lorsque la mémoire du canal est grande, l'architecture complètement parallèle, qui exige un processeur par sous-treillis, est pratiquement irréalisable. En utilisant un processeur pour traiter, non plus un seul sous-treillis mais plusieurs, il est possible avec une architecture fondamentalement semblable, d'effectuer les mêmes opérations en plusieurs cycles au lieu d'un seul. Le principe repose tout simplement sur le regroupement de plusieurs sous-treillis [17].

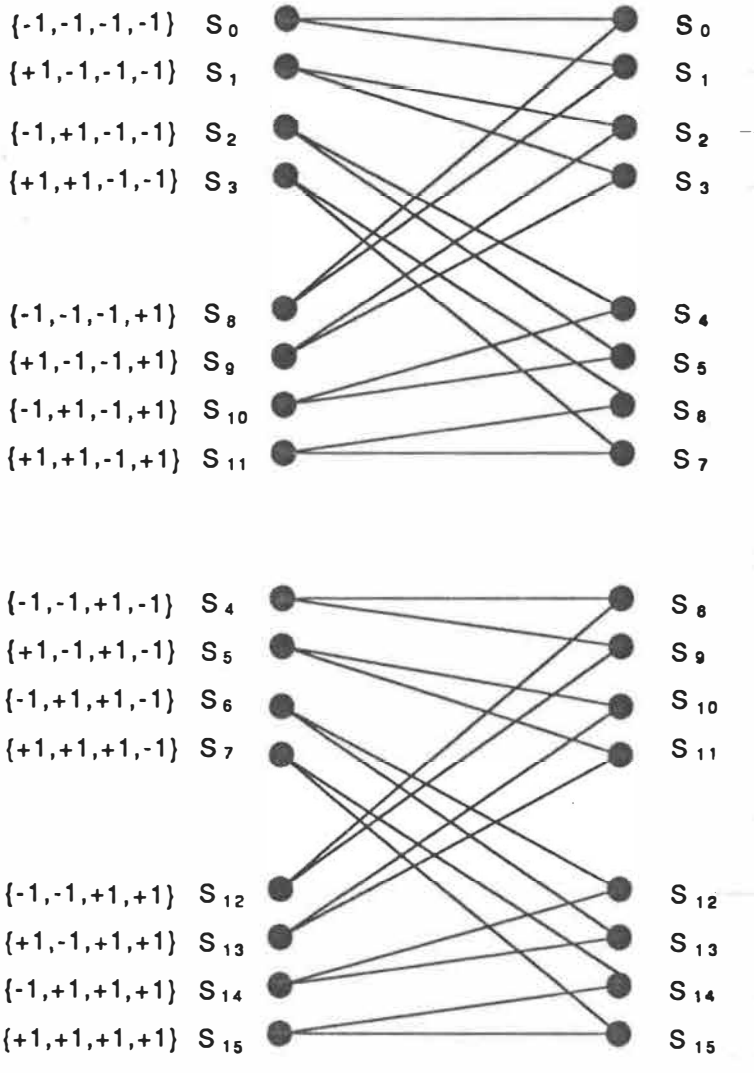
Si on reprend le treillis de la figure 3.4, nous avons 2 processeurs ACS possédant chacun deux registres de métriques. Si on augmente la mémoire à 3, le treillis peut être réarrangé comme à la figure 3.9. Dans une première étape, les états S_0, S_2, S_4 et S_6 sont traités par les processeurs; puis dans un deuxième temps c'est au tour des états S_1, S_3, S_5 et S_7. Si on augmente la mémoire , il est possible de continuer ce processus (figure 3.10). Avec P processeurs ACS, il est possible de traiter un niveau de treillis de mémoire W en $2^{W-1}/P$ coups d'horloge, où $P=2^M$. Il suffit donc de 2^{W-M-1} coups d'horloge.



Treillis (mémoire = 3) modifié

pour l'architecture semi-parallèle

Figure 3.9



Treillis (mémoire = 4) modifié selon

le principe du semi-parallélisme

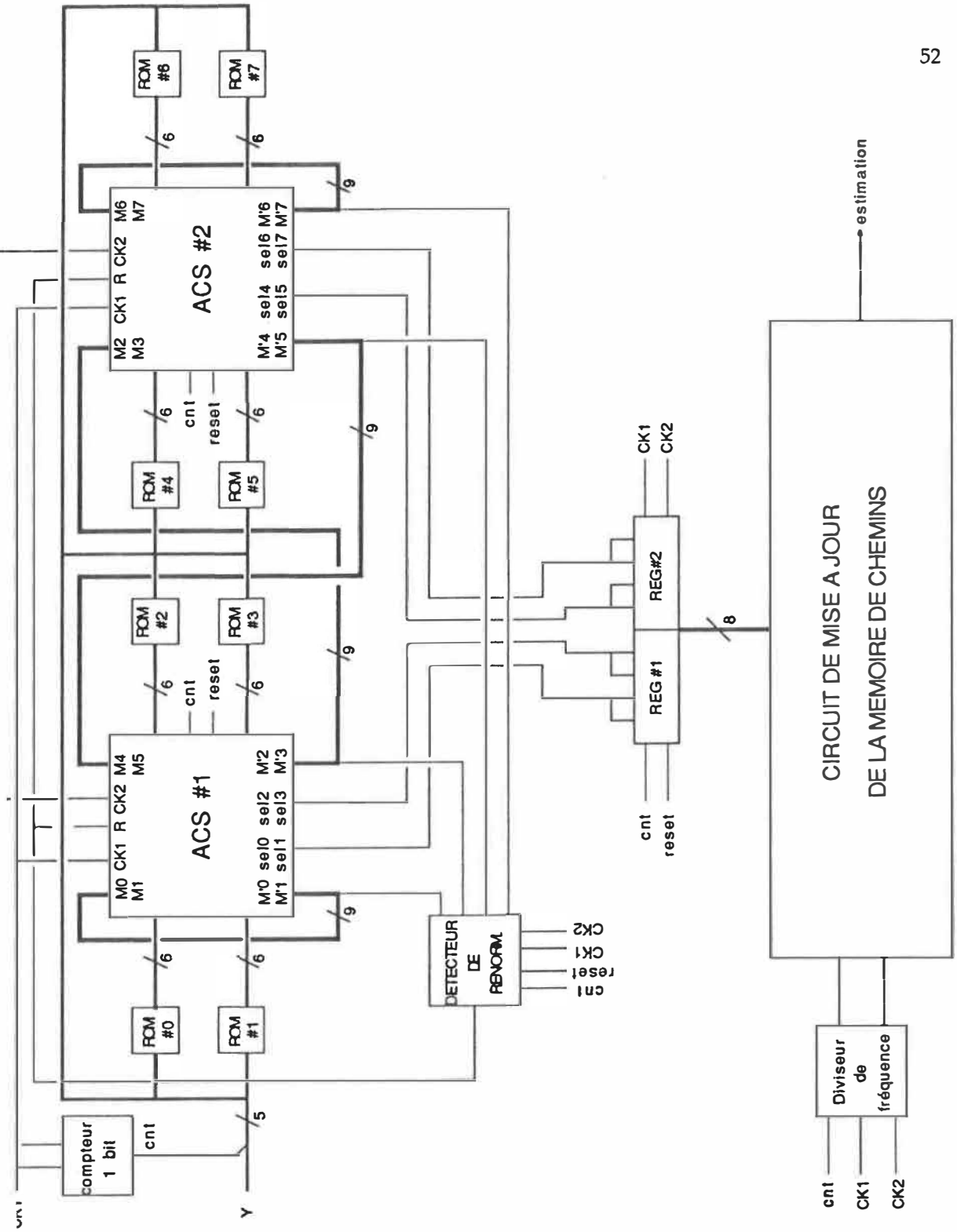
Figure 3.10

Le concept du parallélisme n'est pas affecté. Par contre chaque ACS possédera autant de registres supplémentaires qu'il faut pour représenter les métriques cumulatives des autres états traités par ce processeur, soient 2^{W-M} registres. Il faut aussi prévoir un double des registres à cause de l'architecture même de cette approche. Cette technique de décomposition du treillis peut-être faite sans grand apport de logique comme il est expliqué à la section suivante.

3.4.2 Structure du détecteur semi-parallèle

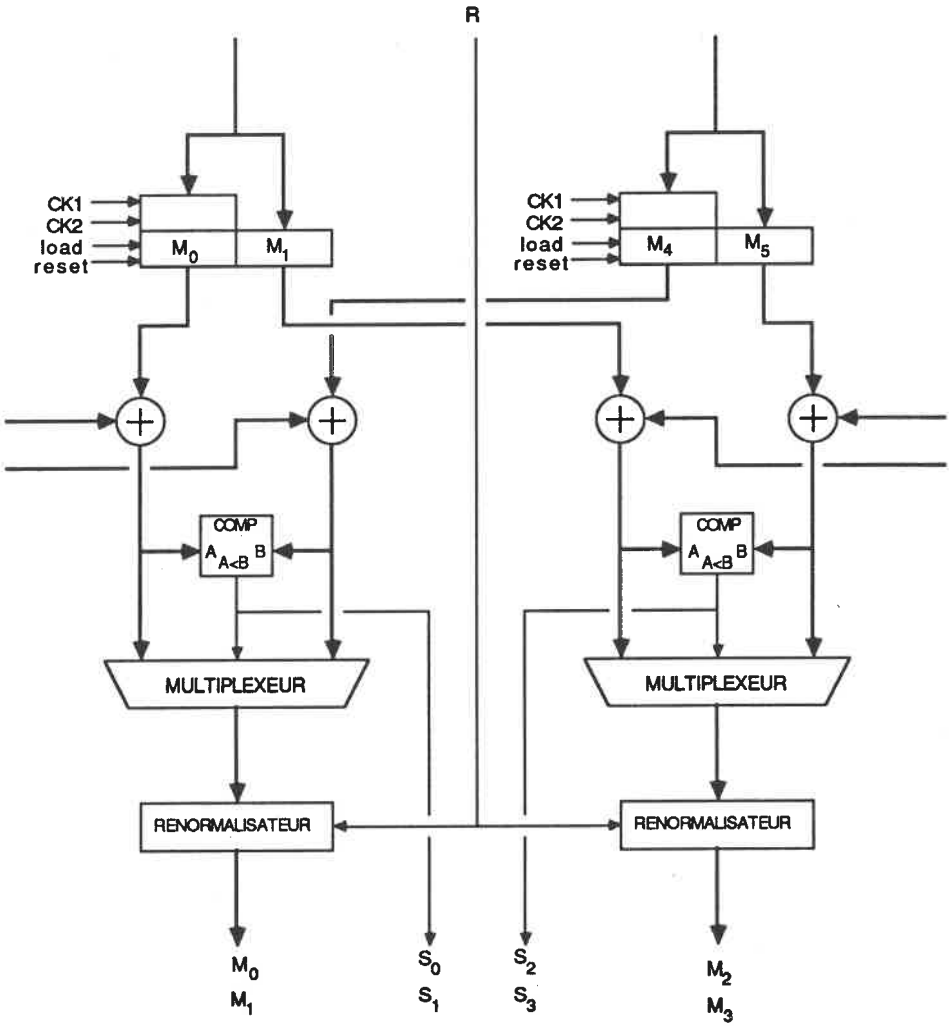
La structure générale appliquant ce principe est globalement semblable à la précédente. Les différences majeures se situent au niveau du contrôle des opérations, et d'une légère modification interne des processeurs ACS. chaque ACS est relié à quatre tables de métriques partielles puisque les tables de l'architecture parallèle de même mémoire ont été regroupées. Le détecteur semi-parallèle de mémoire 3 ainsi que la structure interne d'un ACS, sont illustrés aux figures 3.11 et 3.12 respectivement.

Les modifications internes à l'ACS touchent les registres de métrique cumulative. Le nombre de registres nécessaires à l'intérieur d'un processeur a été précédemment mentionné comme étant 2^{W-M}. Dans notre cas la mémoire du détecteur étant de 3 et M=1, 4 registres sont donc requis puisque chaque ACS traite deux sous-treillis requérant deux métriques cumulatives chacun. Un double des registres est également nécessaire pour emmagasiner temporairement les nouvelles valeurs de métriques cumulatives, et permettre au détecteur d'achever les opérations sur le présent niveau de treillis sans altérer les valeurs présentes.



Détecteur semi-parallèle (mémoire = 3)

Figure 3.11



Processeur ACS semi-parallèle représentant 2 sous-treillis

Figure 3.12

Le contrôle est d'autre part légèrement modifié puisqu'un compteur est maintenant nécessaire pour orchestrer le déroulement des opérations. Son contenu spécifie le nombre de cycles de l'horloge principale par rapport au cycle complet d'extension d'un niveau de treillis. Les sorties du compteur contrôlent les signaux d'écriture des registres de métrique cumulative et du registre des signaux de commande (signaux de sélection) requis par le circuit d'historique ainsi que la logique du détecteur de renormalisation. Elles permettent aussi de diviser le cycle d'horloge principale pour fournir à la logique de mise à jour des chemins survivants des signaux à la fréquence requise. Finalement les bits du compteur associés au symbole reçu quantifié permettent de déterminer la valeur du pointeur adressant les tables de métriques partielles.

Une horloge biphasée sans recouvrement CK est suggérée. Cette horloge représente la vitesse actuelle des processeurs. Elle est divisée par un facteur de $2^{W-1}/P$ pour donner la fréquence CK' de décodage du détecteur. Chaque cycle de CK' un symbole quantifié est reçu, et le compteur commence à compter à partir de zéro. Les métriques calculées durant le cycle CK', ainsi que les signaux de sélection sont emmagasinés. A la fin de ce cycle, le signal d'horloge de l'historique est généré, et un nouveau symbole quantifié est disponible.

Examinons les performances d'une telle approche.

3.4.3 Performances et discussion

La vitesse de cette approche est liée à deux facteurs: Le nombre P de processeurs ACS, et la mémoire W du canal. La vitesse théorique est égale à:

$$f' = f \cdot P / 2^{W-1} \qquad (3.6)$$

où f représente la fréquence d'horloge du système. Le nombre de processeurs détermine donc la vitesse pour un canal donné. Le nombre de cycles requis par niveau de treillis est inversement proportionnel au nombre de processeurs disponibles. Autrement dit, la vitesse est proportionnelle au nombre de processeurs. La vitesse maximum réalisable est atteinte lorsque toutes les opérations sur un niveau de treillis se font en un seul cycle d'horloge, i.e. si il y a autant de processeurs que de sous-treillis. Nous avons alors l'approche parallèle qui permet d'atteindre la vitesse maximale. A l'opposé, la vitesse d'estimation la plus lente est atteinte dans le cas où seulement un processeur est disponible pour les calculs. La vitesse du détecteur dépend alors uniquement de la valeur de mémoire du canal puisque les opérations se font alors de façon sérielle comme dans l'approche pipeline (voir section suivante). Au niveau des performances, l'approche semi-parallèle se situe donc entre les autres. Dépendant des nécessités (vitesse minimale, espace disponible), on peut varier le nombre de processeurs pour obtenir les performances recherchées. De façon plus générale, la vitesse du détecteur de Viterbi peut être déterminée par le nombre maximum de processeurs acceptables sur la puce, ou à l'inverse, le nombre de processeurs est déterminé par la vitesse désirée.

Le problème de disponibilité d'espace sur la puce reste le même qu'avec l'approche parallèle. Lorsque le nombre de processeurs et de mémoires mortes croît, ainsi que leur surface, il devient très difficile, voir même impossible, de les accommoder sur une même puce de silicium. Le problème des interconnexions entre les modules reste également similaire. Néanmoins, cette approche permet d'obtenir un bon compromis entre les performances et la complexité du détecteur.

Observons maintenant l'approche pipeline dont les contraintes spatiales ne sont pas aussi rigides.

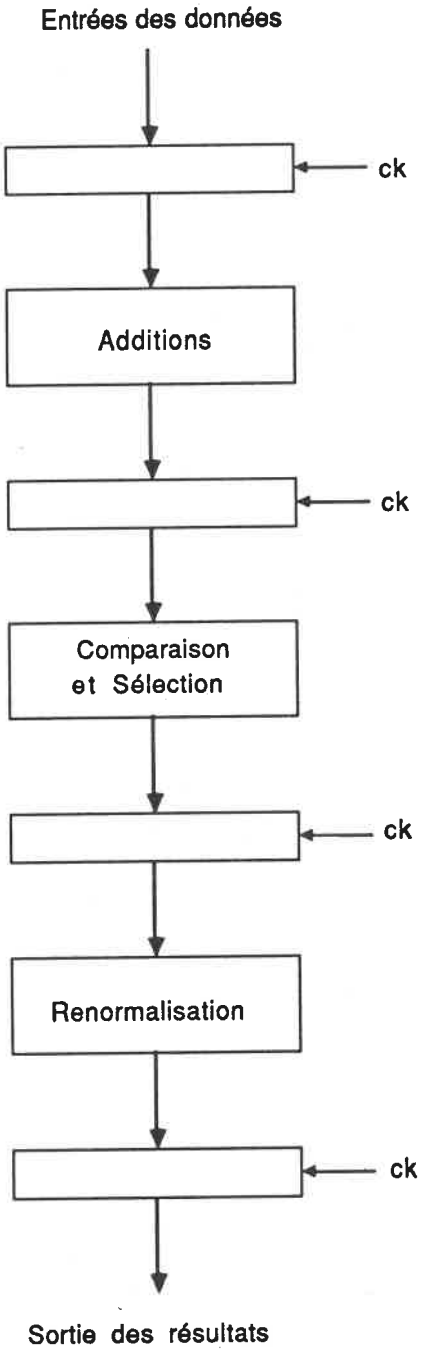
3.5 Architecture Pipeline

3.5.1 Principe de l'approche pipeline

Au lieu d'effectuer les opérations d'un niveau de treillis simultanément, il est possible de les réaliser de façon sérielle. Le principe de structure chaînée appelé pipeline est semblable à celui d'une chaîne de montage [18]. Chaque processeur traite l'information provenant du processeur précédent, puis transmet le résultat au processeur suivant. Tous les processeurs peuvent travailler simultanément mais sur des taches distinctes. Pour son application à l'algorithme de Viterbi (figure 3.13), ce principe conduit à considérer un processeur qui s'occupe des additions, un autre des comparaisons et sélections, et enfin le dernier de la renormalisation. Les différents modules sont séparés par des registres de tamponnage permettant un flot d'information régulier synchronisé par l'horloge du système.

Les calculs des métriques cumulatives des états d'un niveau du treillis sont séquentiels. Les performances d'un tel détecteur sont donc dégradées par rapport à celles de l'approche parallèle. Toutefois la vitesse des opérations peut être optimisée en répartissant la charge de travail entre les différents processeurs. Le cycle de base d'horloge est alors choisi égal au cycle du processeur le plus lent.

Dans un processeur pipeline de n registres, n cycles d'horloge sont nécessaires pour remplir les



Principe de l'approche pipeline

appliqué à l'algorithme de Viterbi

Figure 3.13

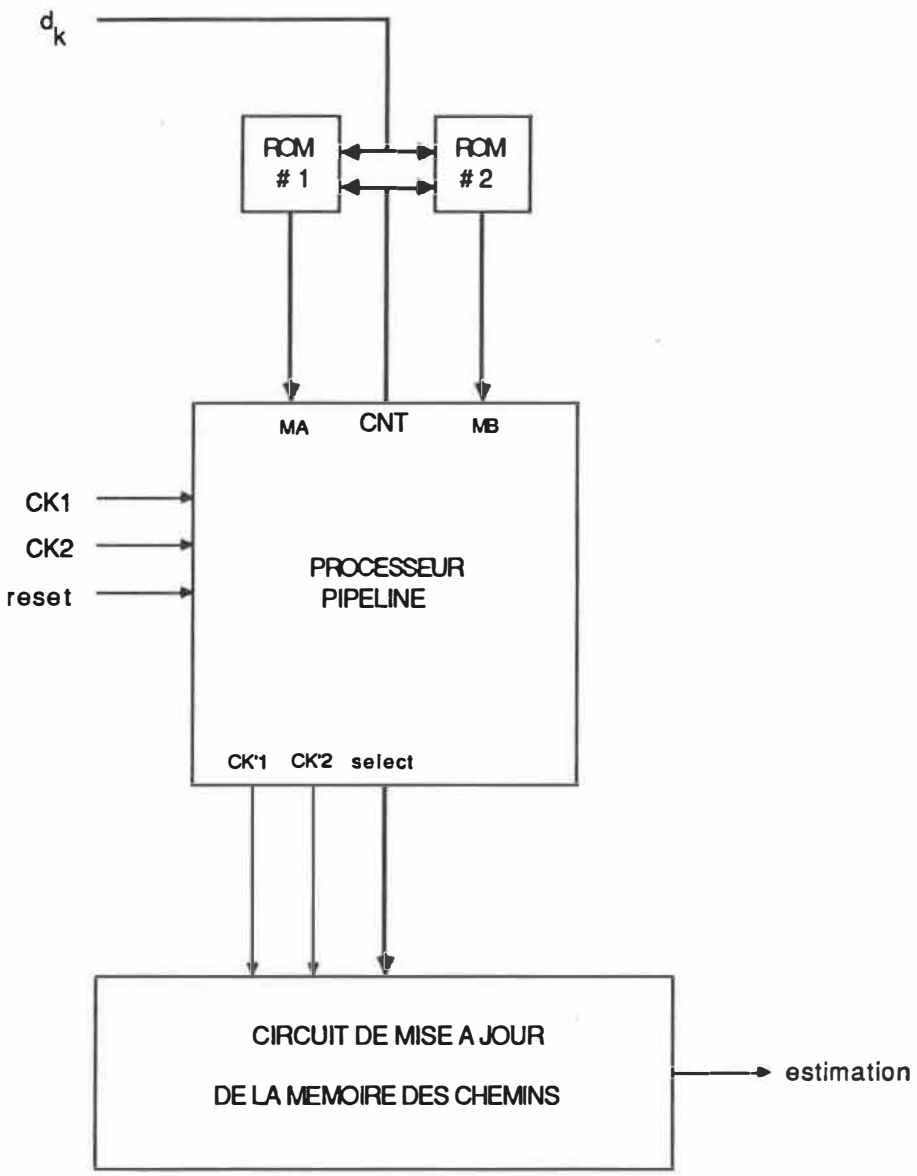
étages. Par la suite, le flot des résultats est régulier, et à chaque cycle d'horloge suivant un résultat est disponible. Evidemment, le contrôle de toutes ces opérations devient plus compliqué, mais la complexité spatiale reste grandement diminuée par rapport aux autres approches.

3.5.2 Structure du processeur pipeline

Comparativement aux architectures précédentes, la structure du détecteur pipeline possède un contrôle plus complexe. Les figure 3.14 et 3.15 représentent les schémas du détecteur de Viterbi pipeline complet pour un canal de mémoire 3. Le système est composé de trois blocs: les deux "roms" qui contiennent les valeurs de métrique de branche précalculées, le processeur pipeline, et la logique de mise à jour des chemins survivants. Avant de décrire la structure globale, examinons tout d'abord la fonction des différents étages du pipeline.

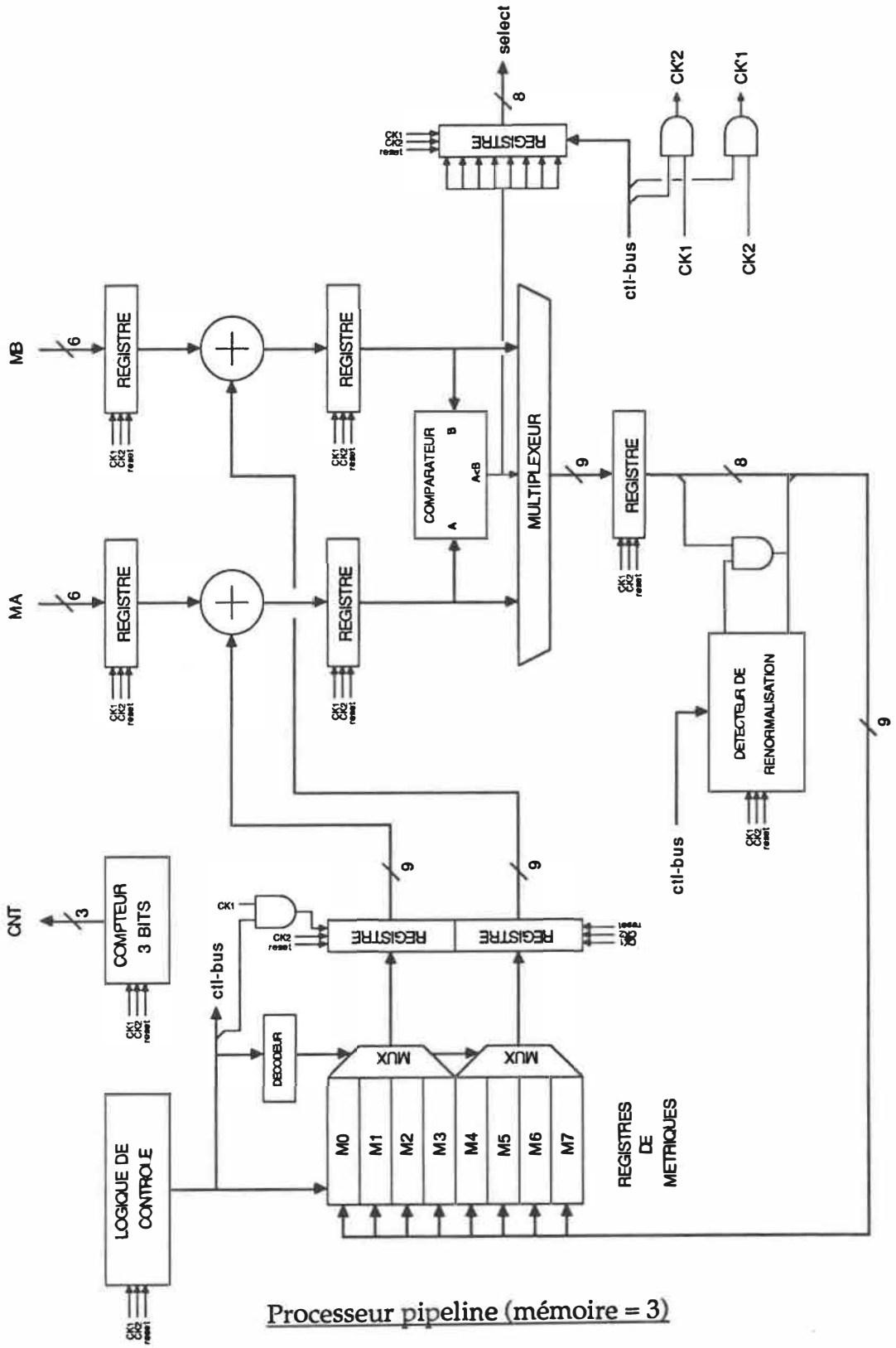
Le détecteur comporte 4 cycles (figure 3.16). Au premier cycle les métriques partielles et cumulatives sont prises des "roms" externes et des registres internes. Au cycle suivant, les additions sont effectuées. Les opérations de comparaisons et de sélections se produisent lors du troisième cycle, pour finalement aboutir au cycle de renormalisation. l'écriture du résultat dans le registre respectif s'effectue durant le premier cycle. La figure 3.17 résume les opérations de synchronisation.

Le processeur pipeline est constitué de deux additionneurs, d'un comparateur et d'un multiplexeur pour effectuer les opérations de base de l'algorithme. Les métriques sont cumulées dans un groupe de registres. Un registre de 8 bits contenant les signaux de sélection



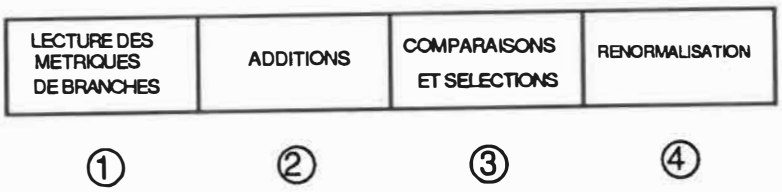
Détecteur pipeline

Figure 3.14



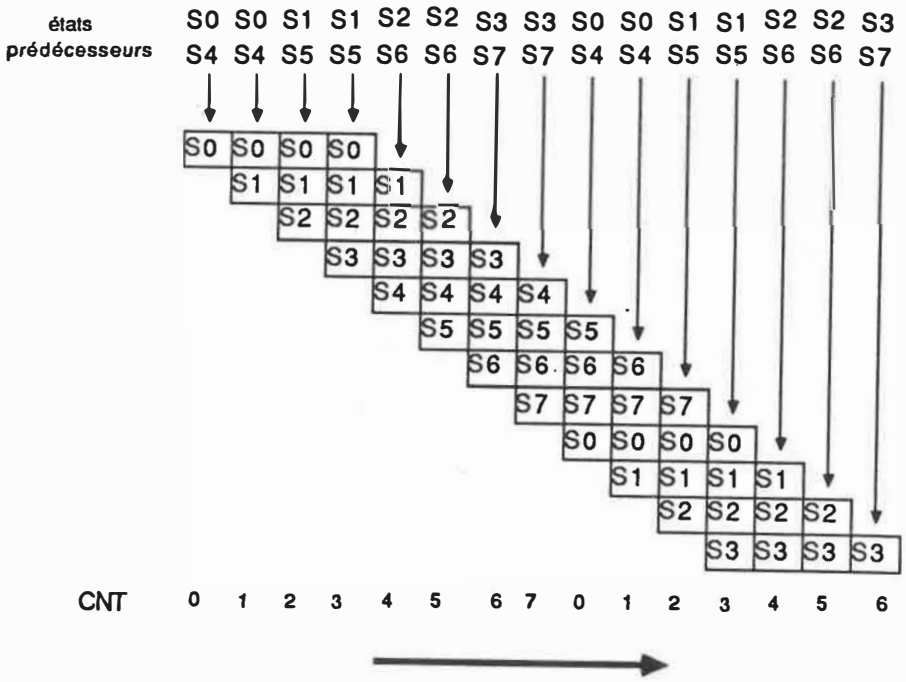
Processeur pipeline (mémoire = 3)

Figure 3.15



Etages pipelines

figure 3.16



Synchronisation des étages pipelines

Figure 3.17

est relié à la logique d'historique. Le tout est séparé par des registres dynamiques partitionnant le circuit. Le circuit de renormalisation fonctionne suivant le même principe que pour l'approche parallèle.

La logique de contrôle est composée d'un compteur de 3 bits et d'un registre à décalage qui orchestrent les opérations et les accès aux différents registres. La sortie du compteur est dirigée vers l'extérieur pour former, avec le symbole reçu, l'adresse des "roms" fournissant les métriques partielles. Les signaux d'horloge requis par la logique de mise à jour des chemins survivants sont générés par le processeur pipeline. Ils doivent être générés au rythme d'un niveau de treillis, soit à une fréquence 2^W (pour un canal de mémoire W) fois plus petite que celle de l'horloge dirigeant le processeur, le temps de mettre à jour le registre de sélection. L'arrivée des symboles reçus doit se faire également à cette vitesse.

A l'initialisation, le registre à décalage est forcé à 1, c'est-à-dire un "1" logique est inséré dans sa première cellule, tandis que le reste des registres sont mis à "0". Un signal d'initialisation est prévu pour amener le processeur dans un état connu où tous les registres internes sont remis à 0.

Normalement, l'accumulation des métriques requiert deux mémoires distinctes. Du fait que pendant que le processus se déroule sur un niveau de treillis, une mémoire doit fournir l'information de source nécessaire aux calculs, tandis que l'autre sert de destination. Au niveau de treillis suivant, leurs rôles sont permutés. Dans notre exemple, le fait d'avoir un détecteur de mémoire 3 possédant 4 étages pipeline permet d'éviter d'avoir une copie des registres. Les simulations vont montrer le déroulement et la synchronisation causés par ces

décisions.

3.5.3 Simulation

Le modèle du canal utilisé pour cette simulation, ainsi que le treillis associé sont illustrés par les figures 3.18 et 3.19. La séquence utilisée pour la validation du circuit est la même que celle considérée lors de la simulation de l'approche parallèle, soit

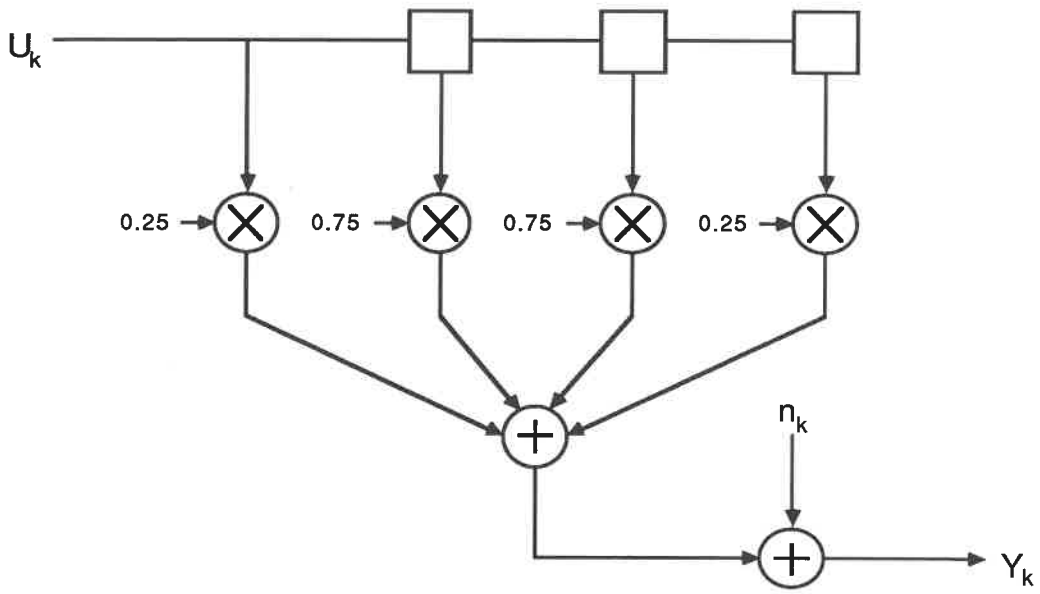
$$\{u_k\}=\{+1,-1,+1,+1,-1,-1,-1,+1,-1,+1,+1,-1,+1,-1,+1,+1\}.$$

Après transmission au travers du canal, puis quantification (16 niveaux), la séquence reçue par le détecteur devient:

$$\{y_k\}=\{1,5,7,9,11,7,11,5,7,9,11,9,7,7,9\}.$$

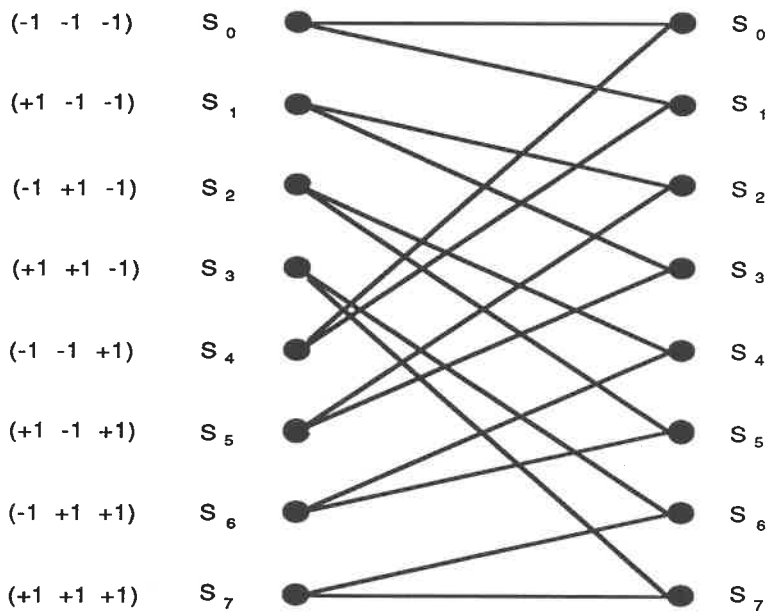
Une séquence d'initialisation, ainsi qu'une queue de 3 symboles ont été placées aux extrémités de $\{u_k\}$ pour former la séquence complète envoyée au modèle.

Le fichier de stimuli appliqués au circuit est disponible en annexe C. Les résultats partiels seulement, sont exprimés par le diagramme de synchronisation de la figure 3.20 ainsi que par le tableau 3.4. Comme dans l'approche parallèle les résultats sont exprimés par rapport au cycle d'horloge correspondant. Le tableau 3.3 possède une colonne représentant les symboles reçus, une colonne pour les signaux de sélection (utilisant la même convention que dans l'approche parallèle), et finalement une colonne pour la sortie de la logique de mise à jour de l'historique des chemins.



Canal (mémoire = 3)

Figure 3.18



Treillis correspondant au canal ci-dessus

Figure 3.19

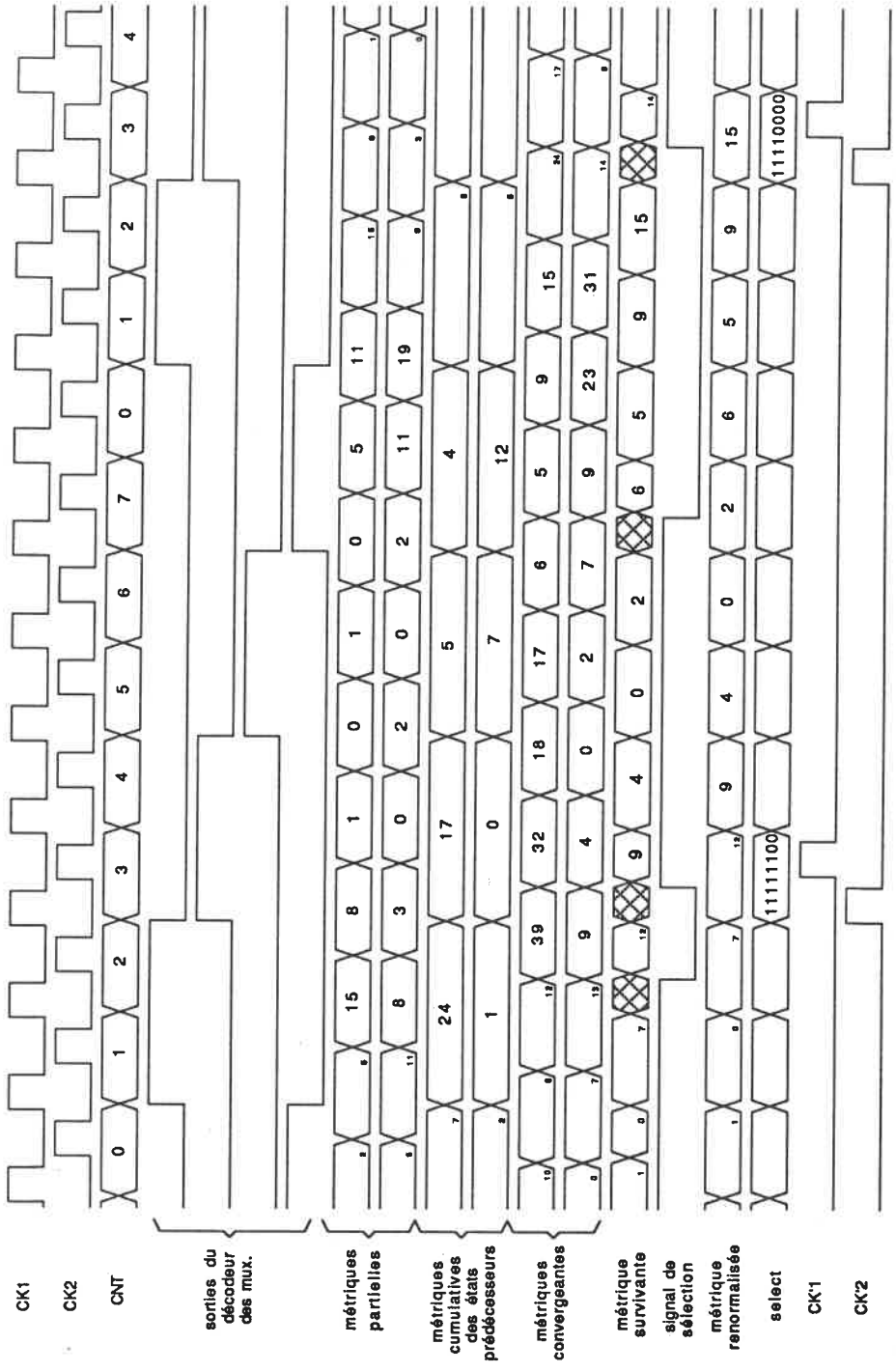


Diagramme de synchronisation du détecteur pipeline

Figure 3.20

cycle d'horloge	symbole reçu
1	0
9	0
17	0
25	0
33	0
41	0
49	0
57	0
65	0
73	1
81	5
89	7
97	9
105	11
113	7
121	1
129	1
137	5
145	7
153	9
161	11
169	9
177	7
185	7
193	9
201	11
209	7
217	1

cycle d'horloge	select
11	00000000
19	00000000
27	00000000
35	00000000
43	00000000
51	00000000
59	00000000
67	00000000
75	00000000
83	00000000
91	00000000
99	11000000
107	11110000
115	11110000
123	11111100
131	11110000
139	00000000
147	00000000
155	11000000
163	11110000
171	11110000
179	11111100
187	11110000
195	11000000
203	11110000
211	11110000
219	11111100
227	11110000

cycle d'horloge	estimation
115	0
123	0
131	0
139	0
147	0
155	0
163	0
171	0
179	0
187	1
195	0
203	1
211	1
219	0
227	0
235	0
243	1
251	0
259	1
267	1
275	0
283	1
291	0
299	1
307	1
315	0
323	0
331	0

Résultats partiels de la simulation

de l'architecture pipeline

Tableau 3.4

3.5.4 Performances et discussion

Dans cette approche, la vitesse est inversement proportionnelle à la longueur de la mémoire du canal du fait que les calculs à chaque cycle ne se font que sur un seul état. La fréquence actuelle du détecteur est donc:

$$f' = f \cdot 2^{-W}, \qquad (3.5)$$

où f est la fréquence d'opération du détecteur et 2^W le nombre d'états du canal.

Les résultats de la simulation indiquent bien le délai de huit cycles de base requis pour traiter un niveau de treillis. Tous les huit cycles un nouveau symbole est reçu et les nouveaux signaux de sélection sont envoyés à l'historique. On peut observer sur le tableau 3.4 que l'ensemble des signaux de sélection est disponible 10 cycles après la réception du symbole reçu. Ce nombre provient des huit cycles requis par les huit états du treillis et des cycles supplémentaires provenant des deux premiers étages pipelines. La vitesse est nettement plus faible que dans les cas étudiés précédemment puisque le nombre d'états du canal influence directement les performances du détecteur de Viterbi.

Du coté de la complexité du circuit, on constate que la structure de flot de données et de contrôle nécessaires est presque indépendante de la valeur de mémoire du canal. Ceci est un avantage certain pour les applications nécessitant une grande mémoire. En effet, leur croissance est négligeable par rapport à la croissance exponentielle des tailles des registres de métriques cumulatives et des tables de métriques partielles requises. Pour une mémoire de 9

états, c'est-à-dire pour un canal possédant 512 états, la taille des deux tables internes de métriques cumulatives doit être de 512 par 9. De la même manière, les deux "roms" externes devraient avoir une taille de 8k bits par 6 (cette valeur de 8k provient de la dimension de l'adresse accédant les tables de 13 bits, soit 9 bits pour les états et 4 bits de quantification). Ces dimensions ne constituent pas une limite technologique. Notons aussi le point important suivant. Les deux tables externes donnent également une grande flexibilité à l'approche pipeline. Il suffit de changer leur contenu pour pouvoir utiliser le détecteur sur un autre canal de transmission. Une modification du contenu contrôlée en temps réel pourrait être la base d'un système à auto-égalisation.

3.6 Comparaison des approches

Le tableau 3.5 résume les caractéristiques des trois approches.

Le phénomène d'interférences entre symboles est présent sur les canaux à bande de fréquence limitée, tel le canal de voix téléphonique. Pour une transmission à vitesse maximum sur cette voie, il est admis que le détecteur doit être capable de supporter une vitesse d'environ 19 Kbits/sec (ce qui correspond à la capacité théorique d'une voie). Cette application requiert également un récepteur de grande mémoire.

Une vitesse de 19 Kbits/sec est facilement réalisable, quelle que soit l'approche utilisée. Cependant pour l'approche pipeline, la vitesse dépend de la mémoire du récepteur. Pour un récepteur de mémoire 9, c'est-à-dire 512 états possibles, opérant à une vitesse de base de 10 MHz, la vitesse d'estimation est alors

approche	vitesse	complexité
parallèle	f	croissance exponentielle
semi-parallèle P processeurs	$f. P / 2^{W}$	croissance exponentielle
pipeline	$f / 2^{W-1}$	croissance exponentielle des mémoires de métriques

f : fréquence d'opération du récepteur

W : mémoire du canal

Comparaison des approches

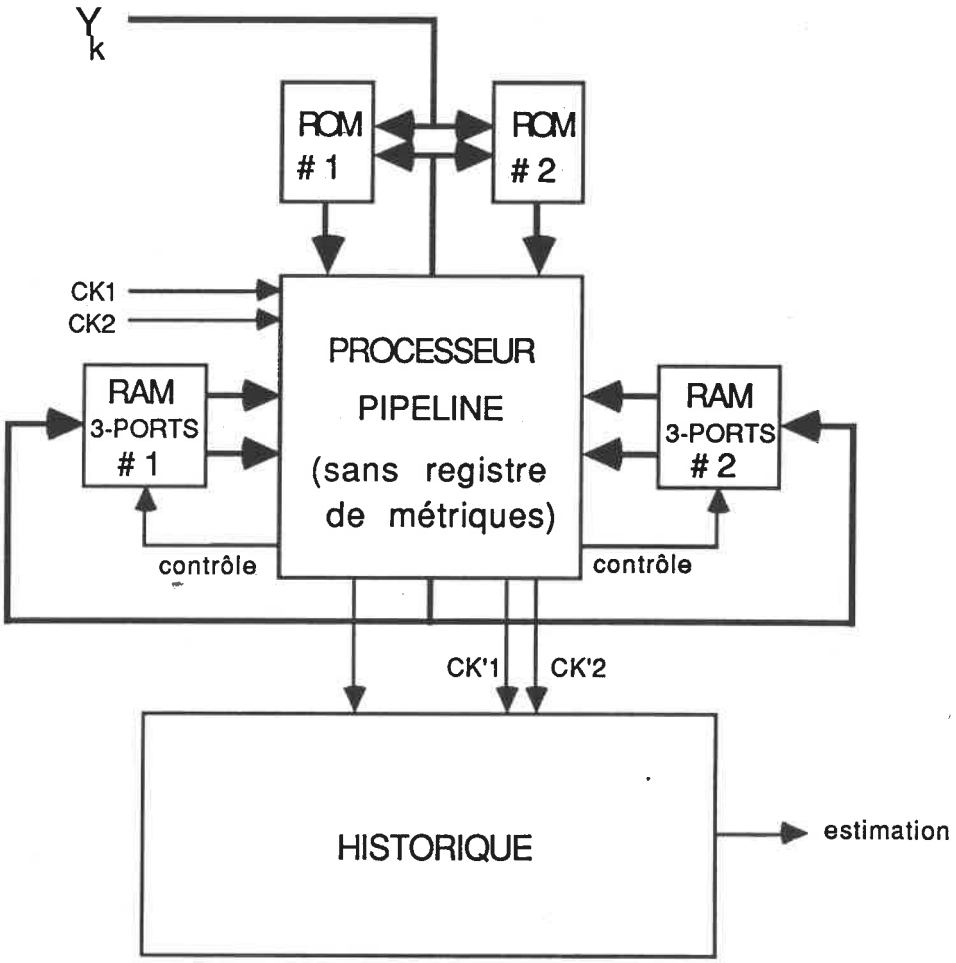
Tableau 3.5

$$10\,\text{MHz} \,/\, 512 = 19.5\,\text{KHz}.$$

Il s'ensuit qu'une valeur de mémoire de 9 semble être la limite du récepteur pipeline au point de vue pratique. Dans la section précédente, nous avons vu que deux mémoires supplémentaires de 512x9 bits contenant les métriques cumulatives sont requises pour la réalisation d'un récepteur de mémoire 9. Des "rams" internes sont utilisables. Des "rams" à trois ports (deux ports de lecture et un port d'écriture) serait l'idéal. Si la technologie ne permet pas cette intégration, il est possible d'utiliser des "rams" commerciales externes.

Pour atteindre une mémoire de cette taille en utilisant l'approche parallèle, il faudrait 256 (512/2) ACS et aussi 1024 (256x4) tables de métriques partielles. Il n'est pas pensable dans les limites de la technologie actuelle d'intégrer tous ces processeurs sur une seule puce. L'approche semi-parallèle offre un compromis intéressant bien qu'à la limite, pour intégrer une mémoire de 9, le récepteur semi-parallèle tendra vers la solution pipeline.

L'architecture pipeline semble donc être la plus apte à une intégration pour combattre l'ISI sur voie téléphonique. La figure 3.21 illustre le schéma de base d'un détecteur pipeline général, avec ses deux mémoire "ping-pong" à trois ports.



Détecteur pipeline de mémoire égale à 9

Figure 3.21

CHAPITRE IV

Circuit de mise à jour de la mémoire des chemins

4.1 Problème de mise à jour

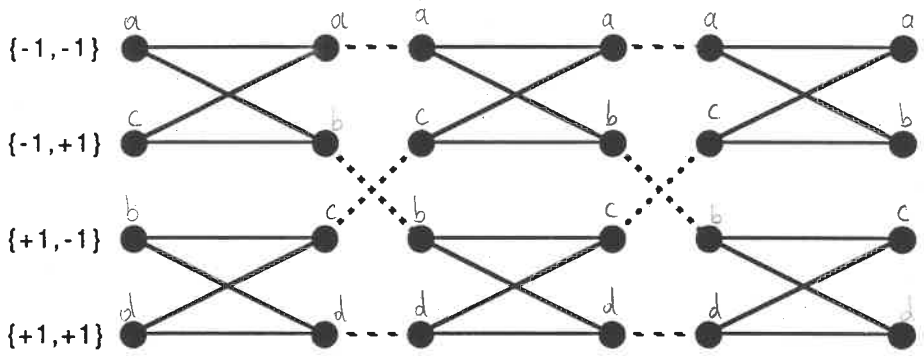
Un des obstacles majeurs limitant la performance d'un processeur de Viterbi relativement à une intégration en circuit intégré est lié au problème de la mise à jour de la mémoire des chemins survivants. En effet, le volume d'information lors de l'échange de données entre les différents processeurs du système se trouve être alourdi par la taille imposante des données de représentation du passé de chacun des survivants de chaque état. Comme il a été démontré au chapitre 2, il n'est cependant pas nécessaire de garder la totalité des chemins survivants, puisque le passé immédiat s'étendant sur 4 à 5 fois la mémoire est suffisant en pratique. Malgré cette simplification, un récepteur de mémoire 3, et possédant donc 8 états, doit mémoriser pour chacun de ces états au moins les 20 dernières branches des survivants pour éviter une dégradation trop forte due à la troncation de l'historique.

Pour cet exemple spécifique, les approches conventionnelles consistent soit d'utiliser un bus parallèle de données de 20 bits pour chacun des états, soit d'effectuer le transfert par un bus série utilisé 20 fois consécutives. Dans le premier cas, il est bien évident que l'espace de silicium occupé par la circuiterie des bus à l'intérieur de la puce est considérable. Tandis que dans le second cas, la vitesse des échanges de données est ralentie par un facteur proportionnel à la longueur de l'historique.

La solution proposée dans cette étude exploite au mieux les possibilités de l'intégration micro-électronique à grande échelle en élaborant une architecture régulière permettant une réalisation efficace du circuit de gestion de la mémoire des chemins. Sur la puce, la mémoire des chemins est représentée par une matrice de surface égale au nombre d'états du treillis par la longueur des chemins. Les cellules des chemins composant cette matrice sont reliées de la même manière que les états dans le treillis, de sorte qu'un seul signal de sélection par état est nécessaire pour commander les opérations. Pour un circuit de mémoire 3, 8 signaux sont donc connectés à la matrice, et le problème n'est plus celui de relier 8 bus de 20 bits vers la puce, mais seulement un bus de 8 bits vers la mémoire en treillis. Non seulement ce système est dense et régulier mais il permet aussi d'effectuer les opérations de mise à jour en un seul coup d'horloge. La cellule de base utilisée dans cette approche est décrite à la section suivante.

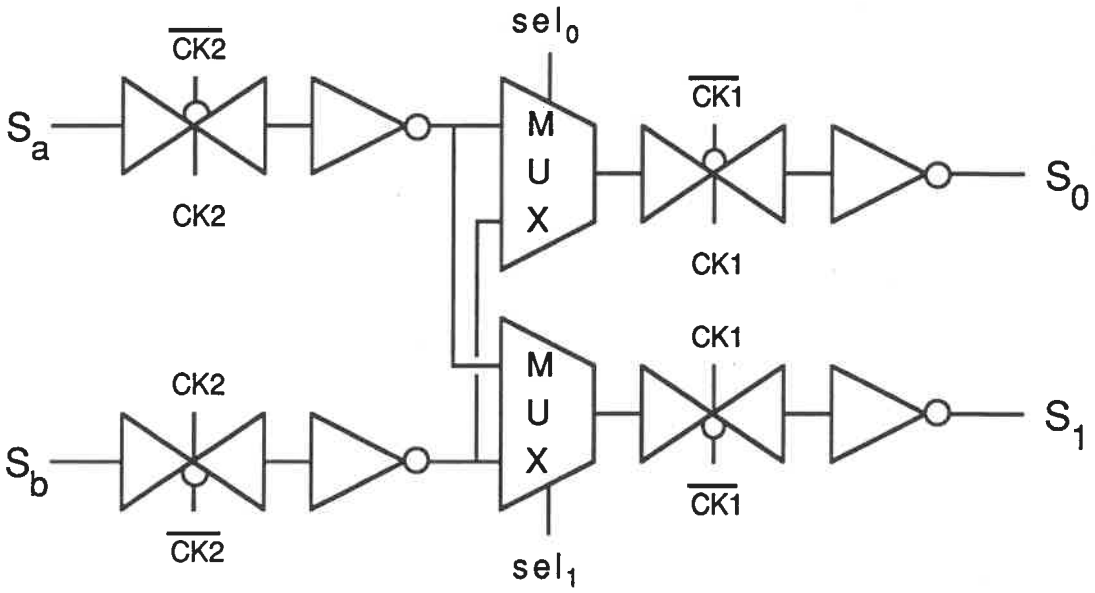
4.2 Cellule de base

Le principe des sous-treillis tel que décrit dans la description de l'approche parallèle de la section 3.3 constitue l'élément majeur permettant l'élaboration d'une logique de mise à jour de l'historique des chemins. A l'image du sous-treillis (figure 4.1), chaque cellule, représentant un symbole du chemin survivant de l'état associé, reçoit deux symboles des cellules associées aux deux états prédécesseurs, et transmet les symboles sélectionnés aux deux cellules associées aux états successeurs. Pour effectuer convenablement la mise à jour, une cellule doit donc être capable de choisir les symboles survivants parmi les deux symboles d'entrée. De plus elle doit pouvoir router les sélections vers les deux cellules des états successeurs.



Connections des sous-treillis

Figure 4.1



Cellule de base pour la mise à

jour de la mémoire des chemins

Figure 4.2

Le schéma de la cellule de base est illustré à la figure 4.2. Pour permettre un flot continu de données vers la sortie de la logique, les cellules sont commandées par une horloge biphasée sans recouvrement. La fréquence de cette horloge varie selon l'approche utilisée. Par exemple, dans l'approche parallèle, elle est la même que celle du système de détermination des survivants. Par contre, dans l'approche pipeline, on devra être capable de générer les deux signaux à une fréquence égale à une fraction de la fréquence de base du système. Le fonctionnement en deux temps permet le décalage vers les cellules suivantes sans perte d'information. La sélection des deux symboles survivants correspondant aux états successeurs du sous-treillis est effectuée grâce à deux multiplexeurs à deux entrées. Evidemment les signaux de commande des multiplexeurs proviennent toujours du processeur de Viterbi, et ceci, quelle que soit l'architecture utilisée.

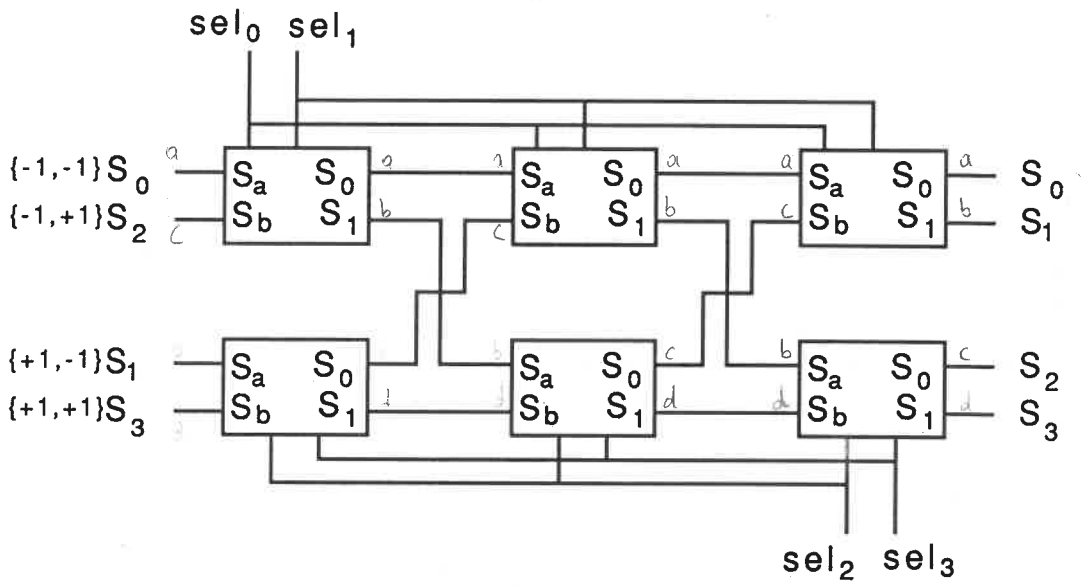
Le fonctionnement d'une cellule se fait en deux temps. A la phase CK2, les symboles des chemins des états prédécesseurs pénètrent à l'intérieur de la cellule au moyen d'interrupteurs électroniques. Les symboles sont alors inversés, et disponibles à l'entrée des deux multiplexeurs. A cet instant, les sélections ne sont cependant pas correctes puisque les signaux de sélection ne sont disponibles qu'à la phase suivante. A la fin de la phase, les interrupteurs d'entrée se retrouvent bloqués, de sorte que les valeurs des symboles sont alors conservées grâce aux capacités d'entrées des inverseurs. A la phase suivante CK1, les signaux de commande des multiplexeurs sont disponibles et les symboles survivants se retrouvent donc aux sorties des multiplexeurs. Les sorties sont de nouveau inversées après passage au travers d'une autre paire d'interrupteurs commandés par le signal CK1. Ces interrupteurs s'ouvrent à la fin de la phase et les valeurs des symboles sont alors isolées du reste de la cellule pendant la phase CK2 prêtes à être transférées vers la cellule suivante.

Observons maintenant le comportement d'une matrice complète.

4.3 Fonctionnement d'une matrice

La mémoire des chemins survivants et le circuit de mise à jour de ces chemins sont réalisés à partir d'une matrice composée exclusivement de la cellule de base considérée à la section précédente. Les dimensions de cette matrice sont égales au produit de la longueur de troncation des chemins par le nombre d'états du récepteur. Ces cellules sont reliées à la façon d'un treillis comme le montrent les figures 4.1 et 4.3 pour un treillis de profondeur 3 et de mémoire 2. Les rangées de la matrice représentent le chemin survivant associé à l'état de la rangée correspondante du treillis. Les colonnes représentent la profondeur de mémoire arrière du treillis pour tous les états.

Le fonctionnement global d'une matrice est le suivant. A chaque comparaison des deux métriques cumulatives des états reconvergeant, un signal de sélection est généré. Fig. 3.12 Tous les signaux (un par états) pour un cycle donné sont alors dirigés vers le circuit de mise à jour des chemins. Les deux chemins correspondant aux deux états prédécesseurs d'un état quelconque sont alors décalés, et un seul (le chemin survivant) remplace l'ancien chemin de l'état en question. A chaque coup d'horloge les chemins sont remaniés de la sorte, et chaque élément binaire d'un chemin se trouve décalé vers la sortie petit à petit. On voit donc que les mémoires des chemins subissent simultanément un décalage et un réarrangement de direction. Entre l'entrée d'un symbole et sa sortie de la matrice, un nombre de cycles d'horloge correspondant à la longueur de la matrice se sera écoulé. Les symboles mémorisés dans la dernière colonne de



Matrice 3x2 représentant

la mémoire des chemins

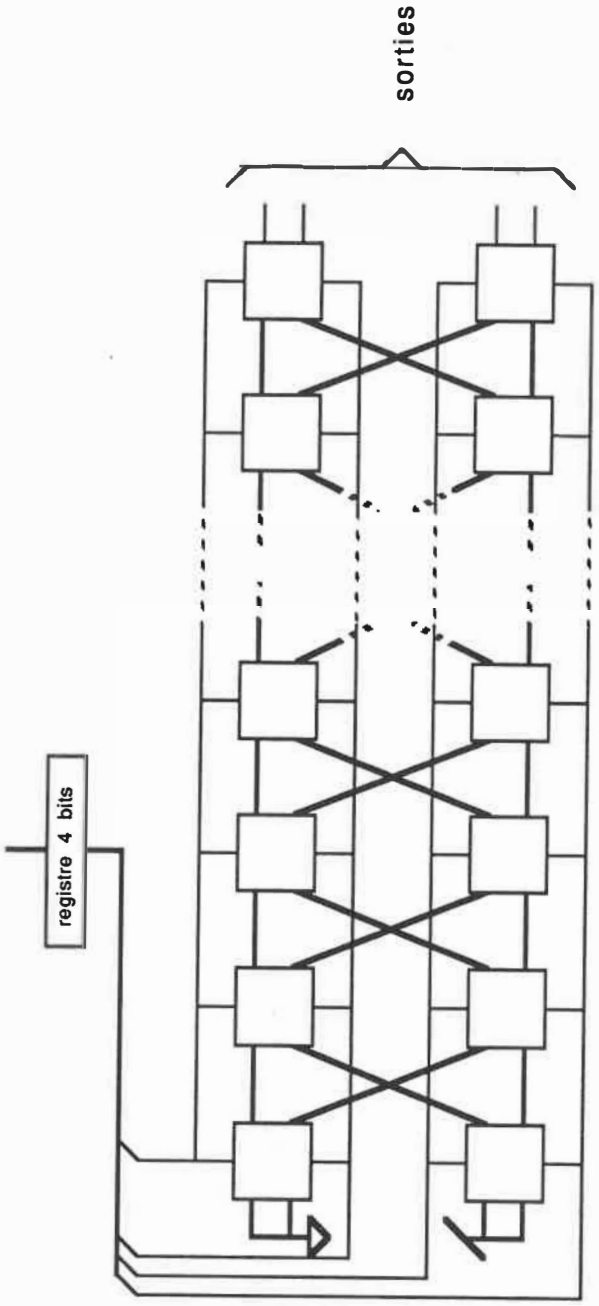
Figure 4.3

la matrice caractérisent l'estimation de l'algorithme; soit une décision majoritaire sur les états, soit un choix correspondant à l'état de métrique maximale donne l'estimation finale de sortie. Par contre, à l'entrée de la première colonne, des valeurs fixes sont injectées puisque pour un état donné, les W bits les plus récents, W étant la mémoire du récepteur, représente l'état.

Pour mieux comprendre le principe, un système complet de mise à jour a été décrit et analysé par le biais d'une simulation logique en se basant sur l'exemple du processeur parallèle étudié à la section 3.3. La figure 4.4 illustre le circuit de mise à jour des chemins correspondants.

Un registre dynamique capte les signaux de sélection pour les fournir aux cellules de la matrice avec un cycle de retard. La matrice est composée de 4 rangées de 12 cellules chacune. Pour accélérer le temps de simulation, un modèle d'une cellule de base a été écrit. La série des signaux de commande générée par le récepteur à la suite de la réception d'une séquence erronée est appliquée au circuit. Les résultats obtenus sont contenus dans le tableau 4.1 et montrent l'évolution du contenu de la matrice tout au long du déroulement des opérations du détecteur pour chaque cycle d'horloge.

La première estimation sort de la matrice 13 coups d'horloge plus tard puisqu'une attente de 12 cycles est nécessaire pour permettre à un symbole donné de traverser la matrice au complet, et un cycle supplémentaire de retard est ajouté par le registre intermédiaire entre le processeur et la matrice. La séquence à la sortie est donc bien équivalente à celle générée par la source. On peut s'apercevoir, comme prévu, que plus on s'enfonce dans le treillis, plus



Circuit de mise à jour pour un

récepteur de mémoire égale à 2

Figure 4.4

k	k+1	k+2	k+3
0000000000000	0010000000000	0011000000000	0001100000000
0100000000000	0110000000000	0101000000000	0110100000000
1000000000000	1010000000000	1011000000000	1001100000000
1100000000000	1110000000000	1101000000000	1101100000000
k+4	**k+5**	**k+6**	**k+7**
0011010000000	0001101000000	0000110100000	0011011010000
0100110000000	0101101000000	0110110100000	0100011010000
1011010000000	1001101000000	1000110100000	1011011010000
1100110000000	1101101000000	1110110100000	1100011010000
k+8	**k+9**	**k+10**	**k+11**
0010001101000	0001000110100	0011011011010	0011010001101
0101101101000	0110110110100	0110100011010	0101101101101
1010001101000	1001000110100	1011011011010	1011010001101
0101101101000	0101000110100	0110100011010	0111010001101
k+12	**k+13**	**k+14**	**k+15**
0001101000110	0010110100011	0001011010001	0010101101000
0101101000110	0100110100011	0101011010001	0110101101000
1001101000110	1010110100011	1001011010001	1010101101000
0111101000110	0100110100011	0101011010001	0110101101000
k+16	**k+17**	**k+18**	**k+19**
0011010110100	0001101011010	0000110101101	0000011010110
0111010110100	0101101011010	0100110101101	0100011010110
1011010110100	1001101011010	1000110101101	1000011010110
0111010110100	0101101011010	0100110101101	0100011010110

Résultats de la simulation du fonctionnement

d'une matrice de mémoire des chemins

Tableau 4.1

les chemins ont tendance à coïncider du fait que les chemins survivants du treillis proviennent de la même racine.

Les simulations ont donc montré que l'architecture préconisée pour la mise à jour de l'historique fonctionne avec régularité, simplicité, et rapidité. Le système est d'autre part adaptable à n'importe quelle architecture du processeur de Viterbi puisqu'il n'en dépend pas. Il suffit simplement de lui transmettre les signaux de sélection et les signaux d'horloge à la bonne fréquence.

4.4 Conception de la puce

4.4.1 Description globale

La puce contenant la logique de mise à jour des chemins survivants constitue un des différents modules du détecteur de Viterbi. Elle a été conçue de manière à être indépendante de l'architecture utilisée pour réaliser le reste des opérations de l'algorithme. Il suffit de connecter les signaux de sélection générés par le processeur de Viterbi aux entrées correspondantes de cette puce et de lui fournir les deux signaux de l'horloge biphasée et sans recouvrement à la bonne fréquence. Le prototype pourra être ainsi utilisé pour valider les processeurs des autres approches.

La logique de mise à jour est composée principalement d'une matrice de cellules de base reliées entre elles à la façon d'un treillis. Puisque la grande majorité de l'espace

disponible sur la puce est occupé par cette matrice, il devient important d'optimiser la cellule de base pour en réduire la superficie. Les dimensions vont fixer le nombre maximum d'états possibles ainsi que la profondeur du treillis. La taille de la matrice est déterminée par la taille de la puce mise à notre disposition par la SMC ("Société Canadienne de Micro-électronique"). La puce disponible, de taille B, possède des dimensions de 6.743 mm par 2.966 mm, soit à peu près 20 mm^2; par contre, les plots de connexions prennent une partie de la superficie. La technologie CMOS 3 microns ainsi que les dimensions de la cellule d'historique ont finalement imposé la taille finale de la matrice. Contraint par ces dimensions, il a été possible de faire entrer dans ces limites une matrice de 8 sous-treillis (16 états) par une profondeur de treillis de 28 correspondant à un détecteur de Viterbi de mémoire égale à 4.

Un registre dynamique permettant l'écriture à chaque cycle d'horloge d'un mot de 16 bits représentant les 16 signaux de sélection est également nécessaire. Ces signaux proviennent du processeur de Viterbi utilisé. Chacun de ces signaux (un par état) et son inverse commandent une rangée complète de la matrice. Or une rangée possède 28 cellules de base, de sorte que les capacités de grilles ajoutées aux capacités parasites des connexions forment une capacité globale trop forte pour pouvoir être commandé par un seul transistor de taille unitaire. Il s'ensuit que le registre ne pourra pas fournir la commande dans un temps raisonnable (i.e. avant le cycle suivant) s'il n'est pas muni d'inverseurs de tailles adéquates. Des inverseurs tampon ont donc été prévus pour permettre de commander chaque rangée de cellules. Il y va de même pour les quatre signaux d'horloge (les deux phases et leurs inverses). Chacun de ces signaux contrôle chaque cellule de la matrice, soit 448 (16x28) cellules en plus des 16 bits du registre de sélection. Des cellules tampon plus grosses ont donc été prévues dans la conception. Il est à noter que les signaux d'horloge inversés nécessaires au bon

fonctionnement des portes de transmission des cellules de base doivent commander des transistors P. Du fait que ces transistors possèdent une taille double des transistors N, il a été nécessaire de concevoir les inverseurs tampon en conséquence.

Les entrées requises par le prototype sont les 16 signaux de sélection et les deux signaux d'horloge. Aucun signal d'initialisation n'est nécessaire puisque la matrice s'initialise par elle-même au bout de 28 cycles d'horloge. Le premier étage étant connecté directement à des valeurs fixes (niveaux +1 ou -1 représentés respectivement par les tensions 5 et 0 volts), il est clair que, de par le principe même de la matrice, ces valeurs puissent se propager jusqu'à l'autre extrémité. La sortie de la cellule de l'état {-1,-1,-1,-1} du dernier étage de la matrice forme l'unique sortie, soit l'estimation. Il apparaît que ce choix arbitraire est aussi bon [19] qu'une décision majoritaire ou que celle basée sur le choix du chemin de plus grande métrique.

4.4.2 Conception des composants de la puce

4.4.2.1 Cellule de mise à jour de l'historique

La figure 4.5 représente le schéma effectif de la cellule au niveau des transistors. Chaque inverseur, ainsi que chaque interrupteur, requiert deux transistors. D'autre part, les multiplexeurs sont réalisés au moyen de deux interrupteurs. Un total de 24 transistors est donc requis pour réaliser la fonction de mise à jour des chemins survivants par sous-treillis. Les canaux des transistors P possèdent une taille double de celle des transistors N pour compenser la faible vitesse des trous par rapport à celle des électrons. Ainsi, les trous se déplaçant à peu

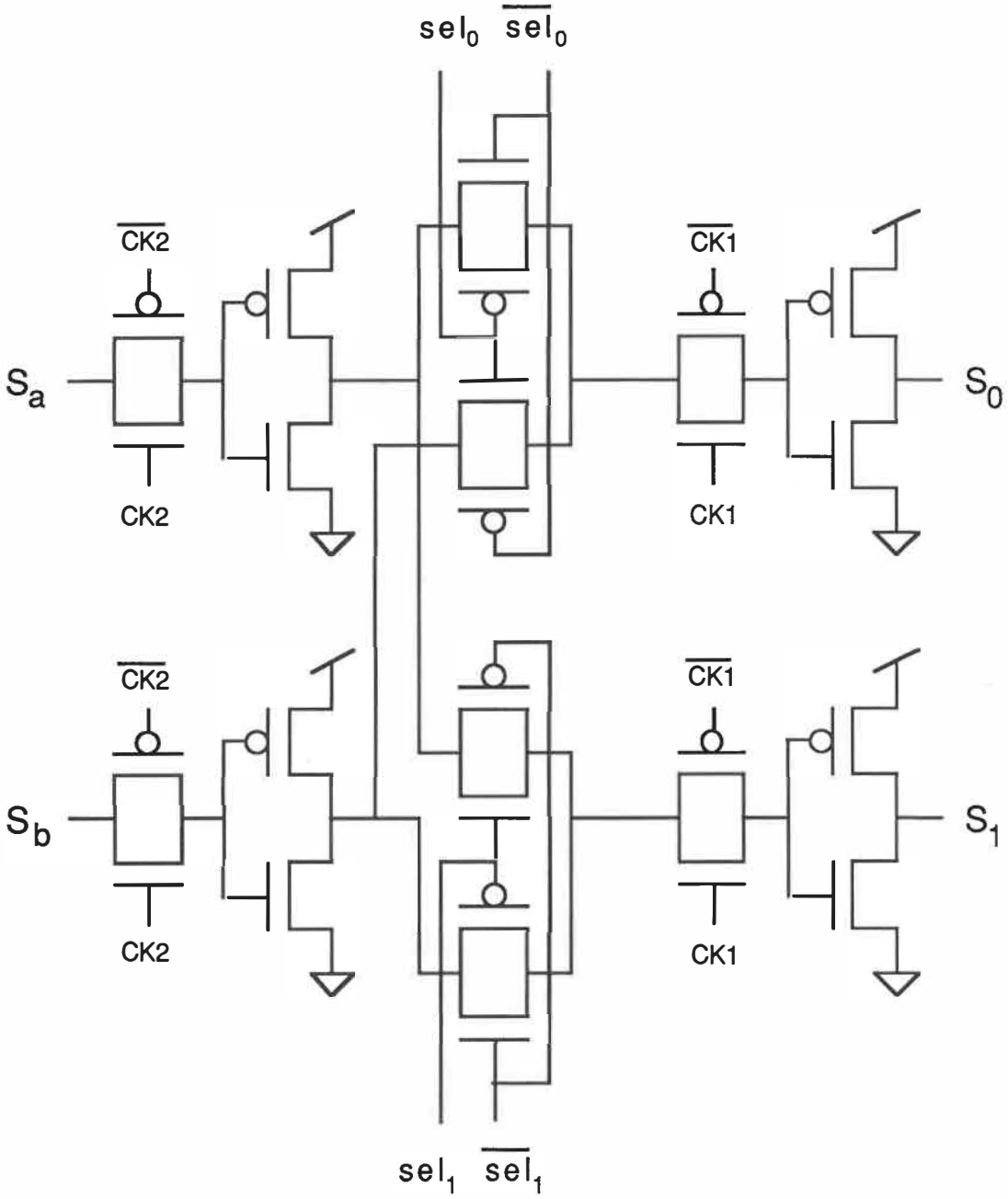


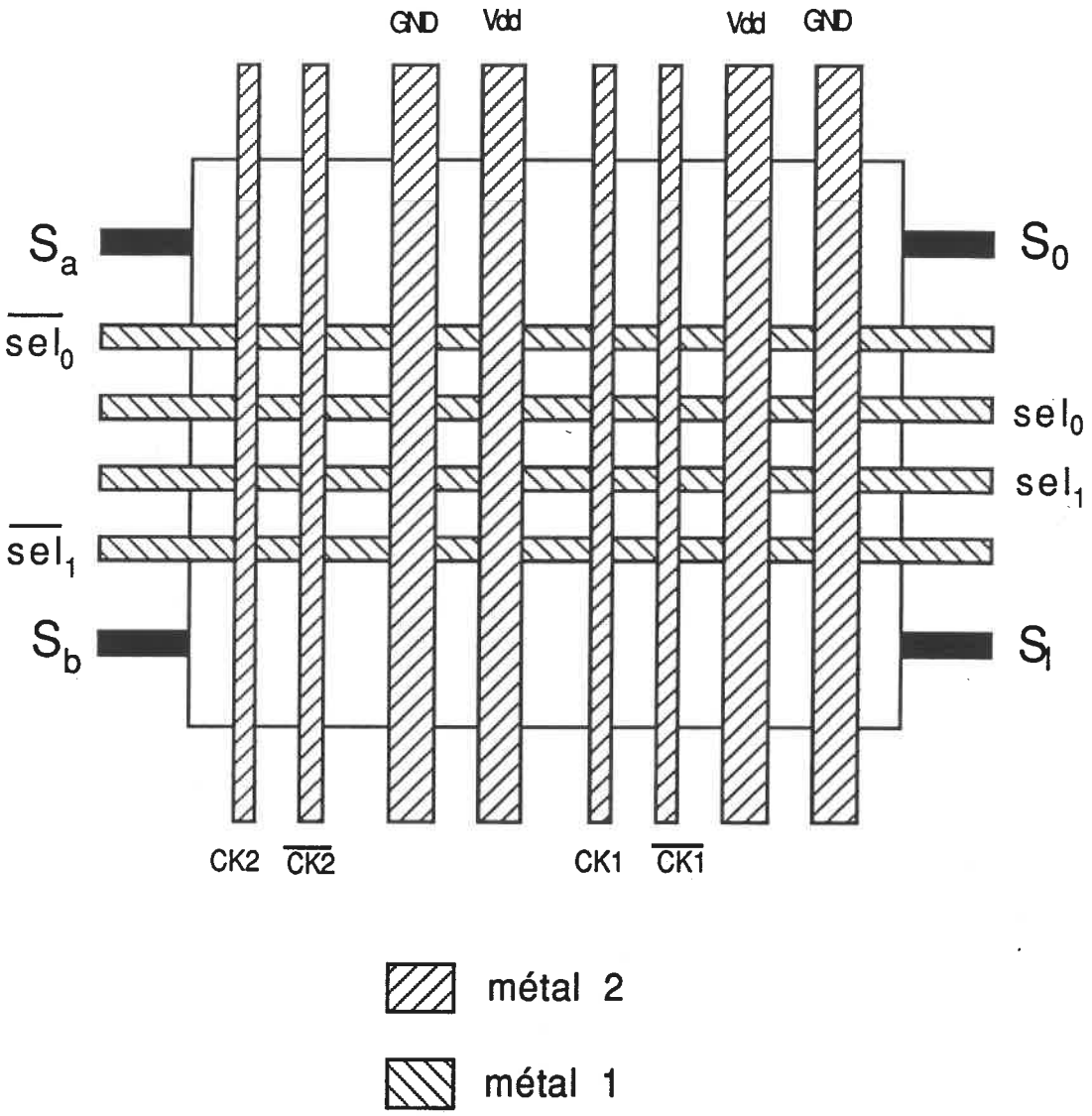
Schéma au niveau transistor de la cellule de base

Figure 4.5

près deux fois moins vite que les électrons, cela permet d'égaliser les temps de transition vers les niveaux haut (+5 volts) et bas (0 volts) des éléments logiques.

Les signaux de commande fournis à chaque cellule sont les 2 phases d'horloge et leurs inverses, ainsi que les 2 signaux de sélection munis également de leur inverse. Chaque signal de sélection commande une rangée de la matrice. Pour une optimisation efficace du routage de chacun de ces signaux et de leur inverse, leur propagation se fait horizontalement à travers les cellules de base sur la première couche de métal. Par contre les signaux d'horloge se propagent verticalement sur la deuxième couche de métal reliant ainsi entre elles toutes les cellules de même profondeur. Les signaux peuvent donc se croiser sans danger. Les deux lignes d'alimentation (+5 volts et la référence) sont également disponibles verticalement en utilisant la deuxième couche de métallisation. Ainsi lors de la formation de la matrice, les cellules se connectent directement partageant ainsi leurs signaux communs sans problème de connexions. La figure 4.6 présente l'agencement des signaux de commande et d'alimentation de la cellule de base tel que décrit plus haut. La cellule comporte également deux entrées de données provenant de l'étage précédent, et deux sorties qui sont dirigées vers les cellules suivantes correspondantes.

Les simulations analogiques ont permis de déterminer les temps de propagation internes des données dans la cellule de base par rapport aux signaux d'horloge. L'annexe D contient le schéma des simulations au niveau des transistors. Les simulations ont tenté de reproduire le comportement de la cellule dans ses conditions normales d'opérations. Le temps de propagation à travers la première partie de la cellule de base calculés à partir de la transition montante de la phase CK2 est de 3.75 nsec pour une transition montante et de 2.81



Agencement des signaux de commande et

d'alimentation de la cellule de base

figure 4.6

nsec pour une transition descendante. La première partie est composée de la porte de transmission commandée par la phase CK2, de l'inverseur et de la logique de sélection. Le délai des sorties de la cellule de base calculé à partir de la transition montante de la phase CK1 est d'environ 2 nsec. Le délai maximum est donc d'environ 4 nsec. La cellule pourrait donc théoriquement être opérée à une fréquence de 250 MHz. Cette fréquence correspond à une période de 4 nsec, soit celle du délai maximum. Cependant il ne faut pas oublier les capacités parasite provenant de la disposition des transistors et de leurs liens à l'intérieur des cellules ainsi que des interconnexions entre les différentes cellules. Celles-ci ont pour effet de ralentir la propagation des signaux internes. Il faut également que l'estimation sortant d'une cellule parvienne à l'extérieur de la puce par l'intermédiaire du plot de sortie. Tous ces éléments ont pour effet de retarder la propagation du signal, et ainsi de diminuer sa fréquence effective d'utilisation. Il faut également tenir compte du temps de commutation des différents inverseurs tampon des signaux de commande. Néanmoins, une fréquence de 10 MHz est facilement réalisable car elle implique une période de 100 nsec.

4.4.2.2. Registre des signaux de sélection

Le registre de 16 bits utilisé dans la puce est un registre dynamique. "Dynamique" signifie que le registre peut perdre sa valeur si elle n'est pas régénérée à des intervalles réguliers. Cependant, il n'y a pas de risque de perte d'information étant donné que le registre fonctionne à la fréquence de la logique de mise à jour de l'historique. Les schémas bloc et transistors sont illustrés respectivement sur les figures 4.7 et 4.8.

Son fonctionnement est très simple. Pendant la phase CK2, les données présentes à

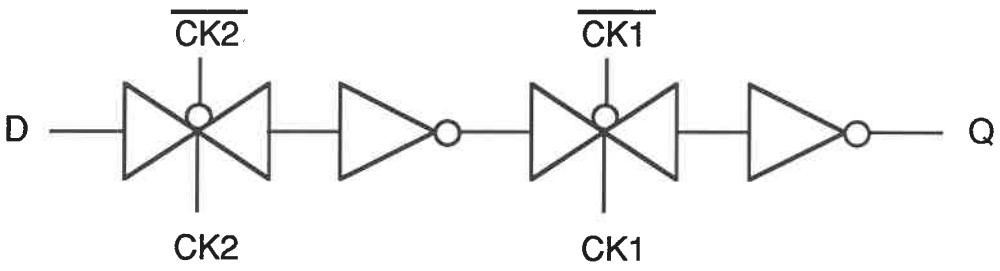
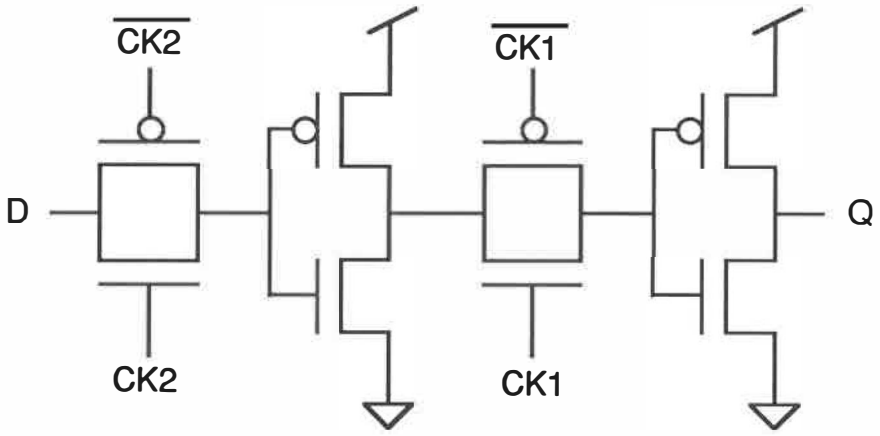


Schéma bloc d'une cellule de registre

figure 4.7



Configuration des transistors

d'une cellule de registre

figure 4.8

l'entrée du registre se propagent vers l'intérieur du registre à travers une porte de transmission et un inverseur jusqu'à l'entrée d'une autre porte de transmission. Celle-ci est commandée par la phase CK1. Une fois la phase CK2 terminée, les données sont ainsi emprisonnées entre les deux interrupteurs électroniques. La phase CK1 laisse alors glisser les données vers la sortie du registre et les cellules de base de la matrice.

La disposition des lignes de commande d'une cellule de registre de un bit est semblable à celle de la cellule de base de l'historique. Les signaux d'horloge se propagent verticalement en utilisant la deuxième couche de métallisation. Les lignes d'alimentation sont ainsi routées de façon identique. L' entrée et la sortie de chaque cellule se fait horizontalement à partir de la première couche de métallisation.

4.4.2.3 Circuits tampon

L'attaque de grosses charges capacitives est effectuée par une chaîne d'inverseurs de taille variée [20]. En fragmentant le délai à travers des inverseurs de taille croissante, il est possible d'atteindre un temps de propagation optimal. Le rapport de la taille entre les étages successifs offre un rendement optimal pour une valeur d'environ 3. Un rapport variant de 2 à 10 est aussi acceptable. Par le biais de simulations analogiques, les étages tampon nécessaires pour commander les cellules de la puce ont été conçus en utilisant cette règle pour déterminer leur taille. Le calcul de leur taille ainsi que les résultats des simulation se trouvent en annexe E.

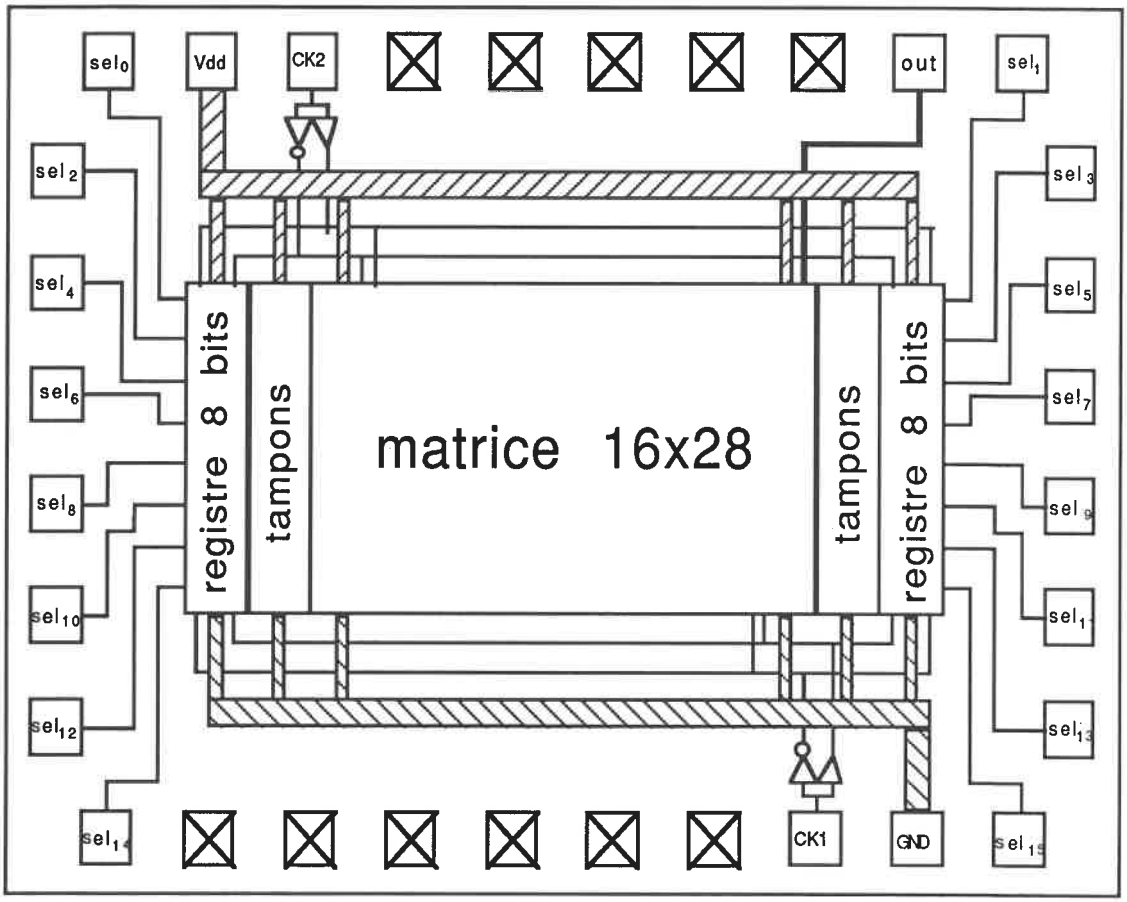
L'amplificateur pour chacun des signaux de sélection est composé d'un inverseur de

taille 3 suivi d'un autre inverseur de taille 9. Les simulations ont permis d'évaluer une valeur de délai d'environ 2.5 nsec pour commander une rangée complète de cellules de base (équivalent à 28 capacités d'un inverseur de taille unitaire). L'amplificateur des deux phases CK1 et CK2 de l'horloge possède deux inverseurs successifs de taille 5 et 30 respectivement. Le délai est approximativement de 3.5 nsec. Tandis que les amplificateurs des signaux inverses des deux phases demandent un délai de 4 nsec pour commander toutes les cellules. Les tailles respectives des deux étages qui les composent sont de 7 et 42. Les simulations n'ont cependant pas tenu compte des capacités parasites des liaisons ainsi que des lignes de métallisation utilisées pour propager les signaux. Il faut donc utiliser ces résultats avec un certain recul. On remarque également que chaque signal d'horloge répond plus rapidement que son inverse. La différence n'est cependant pas très grande, et elle n'aura aucun effet néfaste si le temps mort entre les phases CK1 et CK2 lui est supérieur.

Voyons maintenant comment sont connectés tous ces éléments.

4.4.3 organisation de la puce

L'organisation interne des différents éléments de la puce ainsi que leurs interconnexions sont illustrées à la figure 3.9. On y retrouve également la description des entrées et sorties de la puce. Son espace interne est principalement occupé par la matrice des cellules. De chaque coté de la matrice se trouve une moitié du registre de sélection, soit 8 cellules de registres, suivie des tampons requis pour commander les cellules de mémoire. Une colonne de 16 tampons (2 tampons par bit de registre) se retrouvent donc à la suite de chacune de ces partitions. Les lignes de propagation, à travers la matrice, des signaux de sélection et



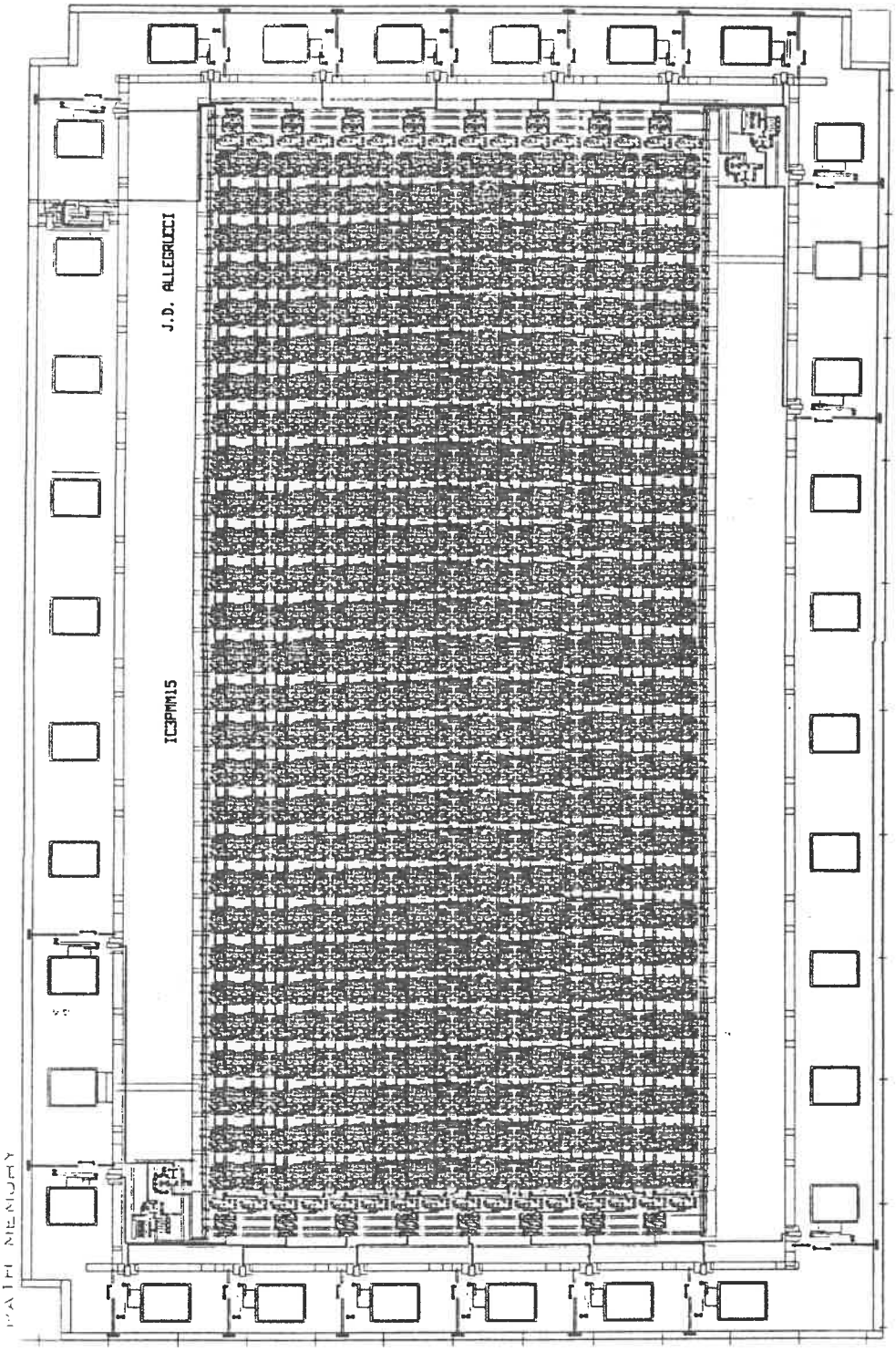
Organisation interne de la puce

figure 4.9

de leur inverse sont directement reliées à chacun de ces tampons. La disposition des lignes de commande et d'alimentation facilite les interconnexions entre les modules. Pour alimenter la matrice et les autres modules, des lignes de métal horizontales ont été placées le long des cotés haut et bas de la matrice. Ainsi il suffit de relier verticalement à ces fils les bornes d'alimentation des cellules prévues à cet effet. Le même agencement est utilisé pour les signaux d'horloge.

32 plots de connexions sont disponibles sur la puce. Seulement 21 sont utilisés; les autres restent non connectés. 16 plots servent à transférer les signaux de sélection vers le registre interne; ils sont partagés parmi les deux extrémités de la puce. Deux autres plots d'entrées sont utilisés pour les deux phase de l'horloge. Les plots d'alimentation sont situés à des extrémités opposées. Finalement, un plot de sortie est utilisé pour l'estimation.

Nous avons donc un système très dense (figure 4.10) qui utilise au maximum les ressources spatiales de la puce. La puce possède plus de 5500 transistors (5376 transistors pour la matrice) et la matrice possède une densité d'environ 450 transistors/mm^2.



Dessin des masques de la puce

figure 4.10

CHAPITRE V

Conclusions et recommandations

5.1 Conclusions

Dans ce travail, différentes architectures tirant à profit les caractéristiques de régularité de l'algorithme de Viterbi appliqué à la correction des erreurs de transmission numérique causées par le phénomène d'interférence entre symboles ont été étudiées. Les contraintes reliées à une telle application sont différentes que celles impliquées dans la réalisation d'un décodeur de Viterbi utilisé pour le décodage convolutionnel, par exemple. La vitesse du récepteur est moins importante car elle est limitée par la bande passante du canal de transmission (réseau de voix téléphonique). Alors que l'approche parallèle permet d'atteindre de grande vitesse, elle devient difficilement réalisable pour des mémoires supérieures à 4 ou 5. Même si l'architecture semi-parallèle offre un compromis intéressant, nous avons vu que l'approche pipeline est la plus prometteuse pour une application pratique.

Un module faisant partie intégrante du détecteur de Viterbi a été également réalisé avec un procédé CMOS 3 microns. L'architecture de cette logique est utilisable par toutes les architectures mentionnées dans ce mémoire. En plus de cette versatilité, son originalité provient du fait qu'elle permet d'effectuer une opération de mise à jour de la mémoire des chemins survivants en en cycle d'horloge seulement. La régularité de cette approche est également un atout puisqu'elle permet d'atteindre une densité (transistors/mm^2) élevée à l'intérieur de la puce.

Les extensions et recommandations suivantes au niveau des différentes puces d'un ensemble complet formant un détecteur de Viterbi pourraient être également envisagées.

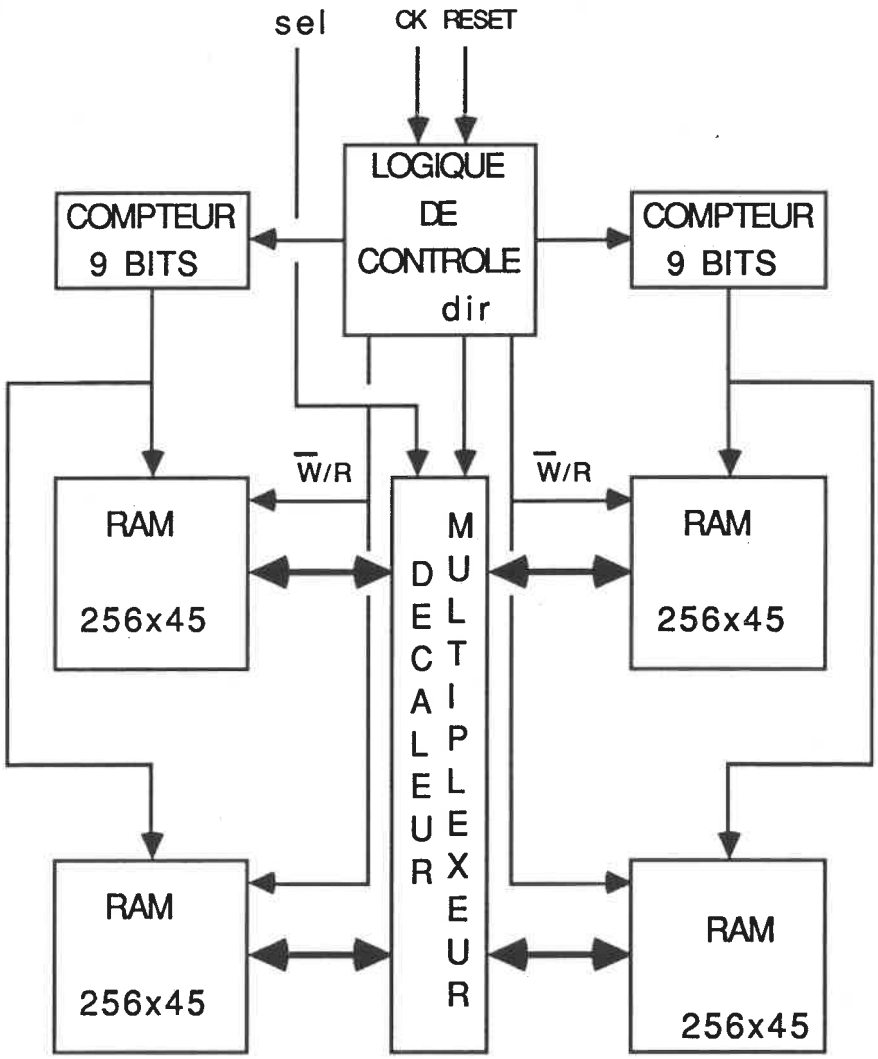
5.2 Recommandations sur la logique d'historique

Comme il a été suggéré au chapitre précédent, cette puce requiert pour chaque état un signal de sélection. Par conséquent, considérant un système de mémoire 9, 512 signaux de sélection doivent être fournis au registre de sélection. Une solution à ce problème serait d'avoir un registre à décalage à la place du registre de sélection; les valeurs des signaux de sélection seraient alors emmagasinées une par une à l'intérieur de ce registre au rythme du processeur pipeline qui produit un signal par cycle d'horloge.

Un autre problème d'envergure concerne l'interconnexion en treillis des colonnes de la matrice. En effet, la surface requise pour ces connexions par conducteur de métal devient beaucoup plus important que la surface même des cellules de mémoires. Dans ce cas particulier, il pourrait être préférable d'utiliser des mémoires "rams" internes contrôlées par le processeur pipeline. La figure 5.1 démontre une esquisse de ce que pourrait être le système résultant.

5.3 Algorithme de Viterbi et mémoire du canal

Malgré son optimalité, l'algorithme de Viterbi a toujours été vu comme une solution pratiquement inutilisable comme détecteur à grandes mémoires. Cependant, conformément à l'étude que nous avons menée, l'évolution de la micro-électronique a considérablement élargie



Alternative pour le problème de mise

mise à jour de la mémoire des chemins

Figure 5.1

le champ des possibilités puisqu'un plus grand nombre de transistors peut être placé sur la même puce, accroissant ainsi les performances d'un système. Parmi les différentes architectures étudiées, l'approche pipeline semble la plus prometteuse pour combattre l'interférence entre symboles sur un canal de mémoire élevée. Prenant pour exemple un système pipeline de mémoire égale à neuf, et en se fixant une technologie capable d'intégrer 10^5 transistors, il est alors possible d'envisager une puce comprenant le processeur pipeline et les mémoires de métriques cumulatives dont la répartition est la suivante:

- 2 mémoires de métriques cumulatives à trois ports possèdent plus de 92000 transistors (2 x 512 x 9 x 10 transistors/cellules),

- le processeur pipeline, estimé à 2000 transistors,

pour un total de plus de 94,000 transistors. Deux modules de mémoires mortes ("roms") supplémentaires pour la caractérisation du canal et la logique d'historique complètent le système.

BIBLIOGRAPHIE

1. G. D. Forney, "Maximum-Likelyhood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference", IEEE Transaction on Information Theory, pp. 363-378, may 1972.

2. S. Lin and D. J. Costello, "Error Control Coding", Chap. 1, Pretince-Hall, 1983.

3. S. Benedetto, E. Biglieri and V. Castellani, "Digital Transmission Theory", Chap. 2, p. 80-82, Prentice-Hall, 1987.

4. J. K. Omura, "On the Viterbi Decoding Algorithm", IEEE Transaction on Information Theory, It-15, pp. 177-179, january 1969.

5. V. K. Bhargava, D. Haccoun, R. Matyas and P. P. Nuspl, "Digital Communications by Satellite", Chap. 12, p. 383, wiley, 1981.

6. J. L. Massey, "Error Bounds for Tree Codes, Treillis Codes and Convolutional Codes with Encoding and Decoding Procedures", in Coding and Complexity, CISM Lecture Series No 216, Springer-Verlag, New-York, 1975.

7. J. A. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication", IEEE Trans. Commun. Techno., COM-19, pp. 835-848, october 1971.

8. A. J. Viterbi and J. K. Omura, "Principles of Digital Communication and Coding", Chap. 4, p. 260, McGraw-Hill, New-York, 1979.

9. Mentor Graphics, "Behavioural Language Model (BLM) - User's Manual".

10. Mentor Graphics, "Quicksim - User's Manual".

11. Mentor Graphics, "Idea Series - Schematic Capture - User's Manual".

12. Mentor Graphics, "Idea System - Reference Manual".

13. A. Mizco, "Digital Logic Testing and Simulation", Chap. 4, Harper & Row, 1986.

14. J. Conan, "An F8 Microprocessor-Based Breadboard for the Simulation of Communication Link Using Rate 1/2 Convolutional Codes and Viterbi Decoding", IEEE Trans. on Communications, VOL. COM-31, No 2, pp. 165-171, 1983.

15. P. Glenn Gulak, E. Shwedyk, "VLSI Structures for Viterbi Receivers: Part I - General Theory and Applications", IEEE Journal on selected areas in Communications, VOL. SAC-4, No 1, pp. 142-154, 1986.

16. T. Ishitani, K. Tansho, N. Miyahara, S. Kubota, and S. Kato, "A Scarce-State-Transition Viterbi-Decoder VLSI for bit Error-Detection", IEEE Journal on Solid-State Circuits, VOL. SC-22, No 4, pp. 575-581, august

1987.

17. J. Bérubé, "Implantation pratique d'un décodeur de Viterbi flexible", Mémoire de maitrise Es Sciences Appliquées, Ecole Polytechnique de Montréal, novembre 1986.

18. C. Mead, and L. Conway, "Introduction aux Systèmes VLSI", Version française Interéditions, chap. 8, 1983.

ANNEXE A

Modèle d'un processeur parallèle

```
/* Version BLM d'un ACS de base pour la configuration parallèle de l'algorithme          */
/* de Viterbi.                */

# systype "sys5"
#include "/idea/sys/ins/qsim.h"
# include "acs.pin.h"

#define MASK_1 1
#define MASK_REN 127

/* creation d'une region de données à l'usager. */
typedef  struct rec_t {
         short          g1[9], g2[9],        /* Métriques d'entrées. */
                        ga[9], gb[9],        /* Métriques de sorties. */
                        g1a, g2a,            /*Métriques de branches. */
                        g1b, g2b,
                        sa, sb;              /* Signaux de sélection. */
         int            met_1, met_2,        /* Valeurs des métriques internes. */
                        met_1i, met_2i,      /* Valeurs intermédiaires des métriques */
                                             /* internes. */
                        met_a, met_b;        /* Valeurs des métriques survivantes */
                                             /* avant renormalisation. */
         long occur;                         /* Instant de la dernière activation de la */
                                             /* routine "ACT_OUT". */
         int            act_mem_count,       /* Nombre d'activation en attente de la */
                                             /* routine d'entrèe. */
                        act_reg_count,       /* Nombre d'activation en attente de la */
                                             /* routine de mise à jour des registres. */
                        act_out_count;       /* Nombre d'activation en attente de la */
                                             /* routine de sortie. */
         char           *act_mem,            /* Pointeur à la routine d'entrée. */
                        *act_reg,            /* Pointeur à la routine des registres. */
                        *act_out;            /* Pointeur à la routine de sortie. */
         } rec_t, *rec_ptr_t;

static long REC_SIZE = size_of(rec_t);

static long time1, time50;

void acs_allocate()
{
     rec_ptr_t              rec_ptr;
```

```
char                    *entry_pt_1, *entry_pt_2;
short                   cntr, len7, len8, len9, len_mes;
double                  time_nsec;
boolean                 io_pin;
qsim_priority_t   priority;

qsim_allocate(&REC_SIZE,&rec_prt);
/* Initialiser les parametres internes a des valeurs inconnues. */
for   (cntr = 0; cntr < 9; cntr++)
      {
      rec_ptr->ga[cntr] = QSIM_ZERO;
      rec_ptr->gb[cntr] = QSIM_ZERO;
      rec_ptr->g1[cntr] = QSIM_ZERO;
      rec_ptr->g2[cntr] = QSIM_ZERO;
      }
rec_ptr->sa = QSIM_ZERO;
rec_ptr->sb = QSIM_ONE;
rec_ptr->met_1 = 0;
rec_ptr->met_2 = 0;
rec_ptr-> met_1i = 0;
rec_ptr-> met_2i = 0;
rec_ptr-> met_a = 0;
rec_ptr-> met_b = 0;
rec_ptr->g1a = 0;
rec_ptr->g1b = 0;
rec_ptr->g2a = 0;
rec_ptr->g2b = 0;

/* Initialiser les comptes d'activation à zéro. */
rec_ptr->act_mem_count = 0;
rec_ptr->act_reg_count = 0;
rec_ptr->act_out_count = 0;

/* Initialiser le temps d'évènements à "très longtemps. */
rec_ptr->occur = -5000;

/* Initialisation des délais par rapport au pas d'incrément du temps. */
time_nsec = 1.;
time1 = qsim_nsec_to_time_step(&time_nsec);
time_nsec = 50.;
time50 = qsim_nsec_to_timestep(&time_nsec);

/* Définition des points d'entrées des routines d'activation. */
len7 = 7;
len8 = 8;
len9 = 9;
len_mes = 25;
if    ((! qsim_define_entry_point("acs_out",%len7,&rec_ptr->act_out)) | |
```

```
        (! qsim_define_entry_point("acs_mem",%len7,&rec_ptr->act_mem)) | |
        (! qsim_define_entry_point("acs_reg",%len7,&rec_ptr->act_reg)) | |
        (! qsim_define_entry_point("acs__g1a",%len8,&entry_pt_1)) | |
        (! qsim_define_entry_point("acs__phi1",%len9,&entry_pt_2)))
        qsim_message("ACS entry point error !!!",&len_mes);

    /* Création des nouvelles routines de ports. */
    io_pin = 0;
    priority = 2;
    qsim_install_pin_handler(&qsim_instance_ptr->ACS_I_G1B, &entry_pt_1,
        &io_pin, &priority);
    qsim_install_pin_handler(&qsim_instance_ptr->ACS_I_G2A, &entry_pt_1,
        &io_pin, &priority);
    qsim_install_pin_handler(&qsim_instance_ptr->ACS_I_G2B, &entry_pt_1,
        &io_pin, &priority);
    qsim_install_pin_handler(&qsim_instance_ptr->ACS_I_G1, &entry_pt_2,
        &io_pin, &priority);
    qsim_install_pin_handler(&qsim_instance_ptr->ACS_I_G2, &entry_pt_2,
        &io_pin, &priority);

    qsim_instance_ptr->user_data_area = (char *) rec_ptr;
}

void acs_save()
{
    qsim_save(&REC_SIZE,&qsim_instance_ptr->user_data_area);
}

void acs_restore()
{
    qsim_restore(&REC_SIZE,&qsim_instance_ptr->user_data_area);
}

acs__phi2()
{
    rec_ptr_t       rec_ptr;
    short           clk1, cntr, len_mes;

    rec_ptr = (rec_ptr_t) qsim_instance_ptr->user_data_area;

    switch
    (qsim_con_value[(*(qsim_instance_ptr->ACS_I_PHI2))->bits[0]])
    {
    case QSIM_ONE:
        clk1 = qsim_con_value[(*(qsim_instance_ptr->ACS_I_PHI1))->bits[0]];
        if (clk1 != QSIM_ZERO)
        {
            len_mes = 17;
```

```
            qsim_message("clock overlap !!!",&len_mes);
        }
        for (cntr = 0; cntr < 9; cntr++)
        {
            rec_ptr->g1[cntr] =
            qsim_con_value[(*(qsim_instance_ptr->ACS_I_G1))->bits[cntr]];
            rec_ptr->g2[cntr] =
            qsim_con_value[(*(qsim_instance_ptr->ACS_I_G2))->bits[cntr]];
        }
        rec_ptr->act_mem_count++;
        qsim_activate(&rec_ptr->act_mem,&time1);
        break;
    case QSIM_UNKNOWN:
        len_mes = 27;
        qsim_message("Value of PHI_2 is unknown !",&len_mes);
        len_mes = 36;
        qsim_message("Internal registers are set to unknown.",&len_mes);
        rec_ptr->met_1 = -1;
        rec_ptr->met_2 = -1;
        break;
    case QSIM_ZERO:
        break;
    }
}

acs__phi1()
{
    rec_ptr_t       rec_ptr;
    short           clk2, cntr, len_mes;

    rec_ptr = (rec_ptr_t) qsim_instance_ptr->user_data_area;

    switch
    (qsim_con_value[(*(qsim_instance_ptr->ACS_I_PHI1))->bits[0]])
    {
    case QSIM_ONE:
        clk2 = qsim_con_value[(*(qsim_instance_ptr->ACS_I_PHI2))->bits[0]];
        if (clk2 != QSIM_ZERO)
        {
            len_mes = 17;
            qsim_message("clock overlap !!!",&len_mes);
        }
        rec_ptr->met_1 = rec_ptr->met_1i;
        rec_ptr->met_2 = rec_ptr->met_2i;
        rec_ptr->act_reg_count++;
        qsim_activate(&rec_ptr->act_reg,&time1);
        break;
    case QSIM_UNKNOWN:
```

```
            len_mes = 27;
            qsim_message("Value of PHI_1 is unknown !",&len_mes);
            len_mes = 36;
            qsim_message("Internal registers are set to unknown.",&len_mes);
            rec_ptr->met_1 = -1;
            rec_ptr->met_2 = -1;
            rec_ptr->act_reg_count++;
            qsim_activate(&rec_ptr->act_reg,&time1);
            break;
        case QSIM_ZERO:
            break;
        }
}

acs_mem()
{
        rec_ptr_t           rec_ptr;
        int                 con_bus_value();

        rec_ptr = (rec_ptr_t) qsim_instance_ptr->user_data_area;

        if  ((rec_ptr->act_mem_count > 1) ||
            (qsim_con_value[(*(qsim_instance_ptr->ACS_I_PHI2))->bits[0]]
            != QSIM_ONE))
            {
                rec_ptr->met_1i = -1;
                rec_ptr->met_2i = -1;
                rec_ptr->act_mem_count--;
            }
        else
            {
                rec_ptr->met_1i = con_bus_value(rec_ptr->g1);
                rec_ptr->met_2i = con_bus_value(rec_ptr->g2);
                rec_ptr->act_mem_count = 0;
            }
}

acs_reg()
{
        rec_ptr_t           rec_ptr;

        rec_ptr = (rec_ptr_t) qsim_instance_ptr->user_data_area;

        if  ((rec_ptr->act_reg_count > 1) ||
            (qsim_con_value[(*(qsim_instance_ptr->ACS_I_PHI1))->bits[0]]
            != QSIM_ONE))
            {
                rec_ptr->met_1 = -1;
```

```
                rec_ptr->met_2 = -1;
                rec_ptr->act_mem_count--;
            }
        else
            {
                rec_ptr->act_mem_count = 0;
            }
        rec_ptr->act_out_count++;
        qsim_activate(&rec_ptr->act_out,&time50);
}

acs_out()
{
        rec_ptr_t               rec_ptr;
        qsim_bit_string_t       out_sa, out_sb;

        rec_ptr = (rec_ptr_t) qsim_instance_ptr->user_data_area;

        if    (rec_ptr->act_out_count > 1)
            {
                rec_ptr->met_a = -1;
                rec_ptr->met_b = -1;
                rec_ptr->sa = QSIM_UNKNOWN;
                rec_ptr->sb = QSIM_UNKNOWN;
                rec_ptr->act_out_count--;
            }
        else
            {
                rec_ptr->act_out_count = 0;
                acs_read_inputs();
                acs_calc();
            }
        out_sa = qsim_con_state[rec_ptr->sa][QSIM_STRONG];
        out_sb = qsim_con_state[rec_ptr->sb][QSIM_STRONG];
        qsim_delay_output(&qsim_instance_ptr->ACS_O_SA,&out_sa);
        qsim_delay_output(&qsim_instance_ptr->ACS_O_SB,&out_sb);
        acs__r();
}

acs_calc()
{
        rec_ptr_t           rec_ptr;
        int                 x, y;

        rec_ptr = (rec_ptr_t) qsim_instance_ptr->user_data_area;

        if ((rec_ptr->met_1 < 0) || (rec_ptr->met_2 < 0))
            {
```

```
        rec_ptr->sa = QSIM_UNKNOWN;
        rec_ptr->sb = QSIM_UNKNOWN;
        rec_ptr->met_a = -1;
        rec_ptr->met_b = -1;
        }
else
        {
        if ((rec_ptr->g1a < 0) || (rec_ptr->g2a  < 0))
                {
                rec_ptr->sa = QSIM_UNKNOWN;
                rec_ptr->met_a = -1;
                }
        else
                {
                x = rec_ptr->met_1 + rec_ptr->g1a;
                y = rec_ptr->met_2 + rec_ptr->g2a;
                if ( x <= y)
                        {
                        rec_ptr->sa = QSIM_ZERO;
                        rec_ptr->met_a = x;
                        }
                else
                        {
                        rec_ptr->sa = QSIM_ONE;
                        rec_ptr->met_a = y;
                        }
                }
        if ((rec_ptr->g1b < 0) || (rec_ptr->g2b  < 0))
                {
                rec_ptr->sb = QSIM_UNKNOWN;
                rec_ptr->met_b = -1;
                }
        else
                {
                x = rec_ptr->met_1 + rec_ptr->g1b;
                y = rec_ptr->met_2 + rec_ptr->g2a;
                if ( x <= y)
                        {
                        rec_ptr->sb = QSIM_ZERO;
                        rec_ptr->met_b = x;
                        }
                else
                        {
                        rec_ptr->sb = QSIM_ONE;
                        rec_ptr->met_b = y;
                        }
                }
```

```
        }
}

acs_read_inputs()
{
    rec_ptr_t       rec_ptr;
    int             i;
    short           g1a[6], g1b[6], g2a[6], g2b[6];

    rec_ptr_t = (rec_ptr) qsim_instance_ptr->user_data_area;

    for (i = 0; i < 9; i++)
        {
        g1a[i] = qsim_con_value[(*(qsim_instance_ptr->ACS_I_G1A))->bits[i]];
        g2a[i] = qsim_con_value[(*(qsim_instance_ptr->ACS_I_G2A))->bits[i]];
        g1b[i] = qsim_con_value[(*(qsim_instance_ptr->ACS_I_G1B))->bits[i]];
        g2b[i] = qsim_con_value[(*(qsim_instance_ptr->ACS_I_G2B))->bits[i]];
        }
    rec_ptr->g1a = con_bus_value(g1a);
    rec_ptr->g2a = con_bus_value(g2a);
    rec_ptr->g1b = con_bus_value(g1b);
    rec_ptr->g2b = con_bus_value(g2b);
}

acs_r()
{
    rec_ptr_t           rec_ptr;
    int                 i;
    qsim_bit_string_t   out_ga, out_gb;

    rec_ptr_t = (rec_ptr) qsim_instance_ptr->user_data_area;

    switch
    (qsim_con_value[(*(qsim_instance_ptr->ACS_I_R))->bits[0]])
    {
    case QSIM_ONE:
        con_value_bus(rec_ptr->ga, rec_ptr->met_a);
        con_value_bus(rec_ptr->gb, rec_ptr->met_b);
        break;
    case QSIM_UNKNOWN:
        for (i = 0; i < 9; i++)
            {
            rec_ptr->ga[i] = QSIM_UNKNOWN;
            rec_ptr->gb[i] = QSIM_UNKNOWN;
            }
        break;
    case QSIM_ZERO:
        rec_ptr->met_a = rec_ptr->met_a & MASK_REN;
```

```
            rec_ptr->met_b = rec_ptr->met_b & MASK_REN;
            con_value_bus(rec_ptr->ga, rec_ptr->met_a);
            con_value_bus(rec_ptr->gb, rec_ptr->met_b);
            rec_ptr->ga[8] = QSIM_ZERO;
            rec_ptr->gb[8] = QSIM_ZERO;
            break;
      }

      for (i = 0; i < 9; i++)
            {
            out_ga[i] = qsim_con_state[rec_ptr->ga[i]][QSIM_STRONG];
            out_gb[i] = qsim_con_state[rec_ptr->gb[i]][QSIM_STRONG];
            }
      qsim_delay_output(&qsim_instance_ptr->ACS_O_GA,&out_ga);
      qsim_delay_output(&qsim_instance_ptr->ACS_O_GB,&out_gb);
}

acs_g1a()
{
      rec_ptr   _t        rec_ptr;

      rec_ptr = (rec_ptr_t) qsim_instance_ptr->user_data_area;

      if (qsim_time() > rec_ptr->occur)
            {
            rec_ptr->act_out_count++;
            qsim_activate(&(rec_ptr->act_out),&time50);
            rec_ptr_occur = qsim_time();
            }
}

/* Cette routine execute une conversion d'un nombre de huit bits en une valeur      */
/* entière.                                                                         */
int con_bus_value(pointeur)

short *pointeur);
{
      int         total;
      short       i, flag;

      flag = 1;
      for (i = 8, total = 0; (i >= 0) && flag; i--)
            if (pointeur[i] != QSIM_UNKNOWN)
                  total = 2*total + pointeur[i];
            else
                  flag = 0;
      if (flag)
            return(total);
```

```
    else
        return(-1);
}

/* Cette routine execute la conversion d'une valeur entière en un nombre          */
/* composé de 9 bits.                    */
con_value_bus(pointeur, valeur)

short    *pointeur;
int  valeur;
{
    int        i;

    if (valeur == -1)
        for (i = 0; i < 9; i++)
            pointeur[i] = QSIM_UNKNOWN;
    else
        for (i = 0; i < 9; i++, valeur = valeur >> 1)
            pointeur[i] = valeur & MASK_1;
    return(0);
}

acs_init()
{
    acs_out();
}
```

ANNEXE B

Stimuli appliqués au modèle de la configuration parallèle

```
CIRCUIT VITERBI_PARALLELE;

/* Ce fichier contient les stimuli appropriés pour les simulations          */
/* fonctionnelles de l'architecture parallèle. Le langage de stimuli est    */
/* le MISL.                                                                 */

/* Définition des vecteurs */
VECTOR YC = "Y(3)", "Y(2)", "Y(1)", "Y(0)";
VECTOR MET0 = "M0(8)", "M0(7)", "M0(6)", "M0(5)", "M0(4)",
        "M0(3)", "M0(2)", "M0(1)", "M0(0)";
VECTOR MET1 = "M1(8)", "M1(7)", "M1(6)", "M1(5)", "M1(4)",
        "M1(3)", "M1(2)", "M1(1)", "M1(0)";
VECTOR MET2 = "M2(8)", "M2(7)", "M2(6)", "M2(5)", "M2(4)",
        "M2(3)", "M2(2)", "M2(1)", "M2(0)";
VECTOR MET3 = "M3(8)", "M3(7)", "M3(6)", "M3(5)", "M3(4)",
        "M3(3)", "M3(2)", "M3(1)", "M3(0)";

TIMEDEF PERIOD = 200 NS

/* Horloge sans recouvrement */
CK1, CK2 = LO;
BACKGROUND CK1 = HI:5NS, LO:+90NS .. AFTER 110NS;
BACKGROUND CK2 = HI:105NS, LO:+90NS .. AFTER 110NS;

/* Initialisation du circuit */
YC = '000' B;
MET0, MET1, MET2, MET3 = '000' H;
R = HI;

/* Envoie des données au détecteur */
DO 6 TIMES
{
YC = '0000' B:+110NS $
ALIGN;
};
YC = '0011' B:+110NS $
YC = '0111' B $
YC = '0111' B $
YC = '1011' B $
YC = '1011' B $
YC = '0011' B $
```

```
YC = '0000' B $
YC = '0011' B $
YC = '0111' B $
YC = '0111' B $
YC = '1011' B $
YC = '1011' B $
YC = '0111' B $
YC = '0111' B $
YC = '0111' B $
YC = '1011' B $
YC = '1011' B $
YC = '0011' B $

END.
```

ANNEXE C

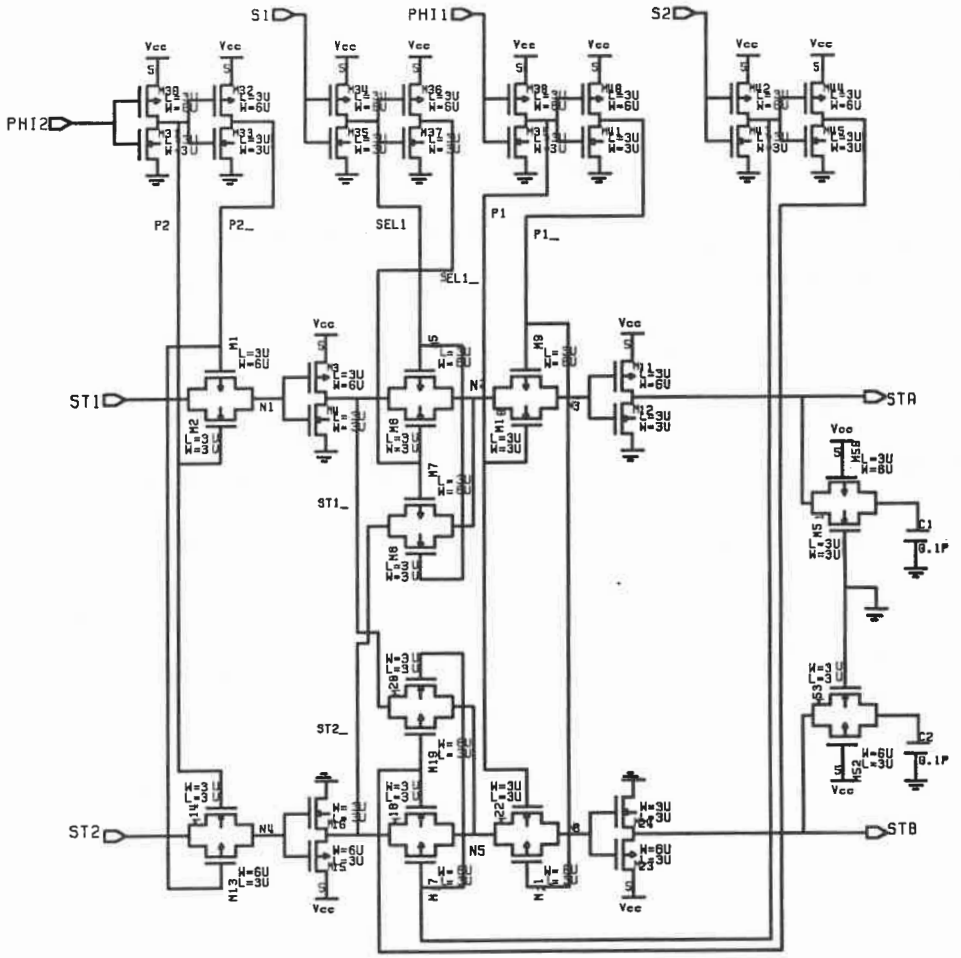
Stimuli appliqués au modèle de la configuration pipeline

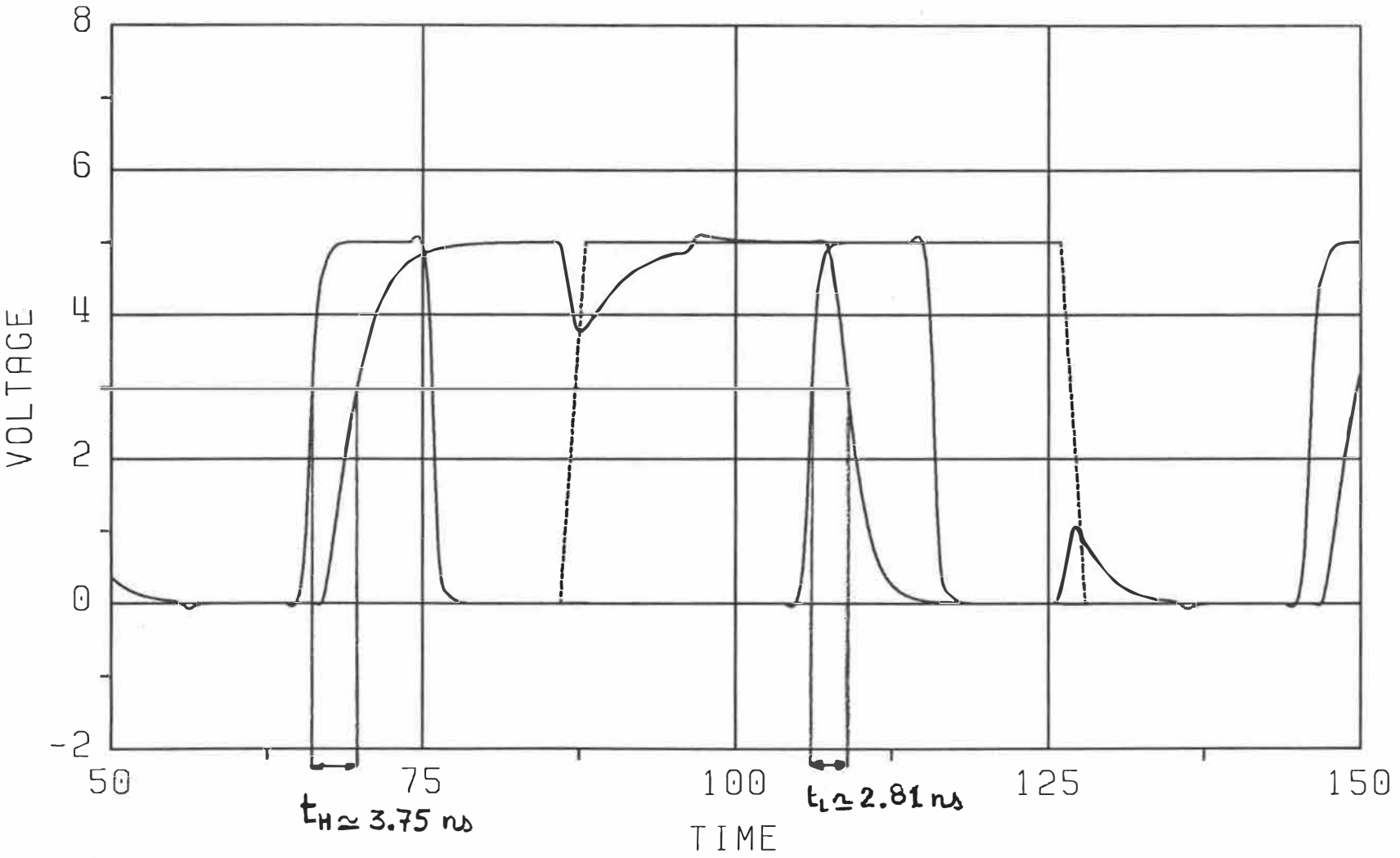
```
CIRCUIT VITERBI_PIPELINE;

/* Ce fichier contient les stimuli pour la simulation fonctionnelle de la      */
/* configuration pipeline. Le langage utilisé est le MISL.                      */

/* Définition des vecteurs */
VECTOR YC = "YC(3)", "Y(2)", "Y(1)", "Y(0)";

TIMEDEF PERIOD = 800 NS;

/* Horloge sans recouvrement */
CK1, CK2 = LO;
BACKGROUND CK1 = HI:5NS, LO:+40NS .. AFTER 60NS;
BACKGROUND CK2 = HI:55NS, LO:+40NS .. AFTER 60NS;

/* Initialisation du circuit */
YC = '000' B;
RESET = HI, LO:+60NS;

/* Envoie des données au détecteur */
DO 9 TIMES {
YC = '0000' B:+60NS $
ALIGN;
}
YC = '0001' B:+60NS $
YC = '0101' B $
YC = '0111' B $
YC = '1001' B $
YC = '1011' B $
YC = '0111' B $
YC = '1011' B $
YC = '0101' B $
YC = '1001' B $
YC = '0111' B $
YC = '1011' B $
YC = '1001' B $
YC = '0111' B $
YC = '0111 B $
YC = '1001' B $
YC = '1011 B $
YC = '0111' B $
YC = '0001' B $
END.
```

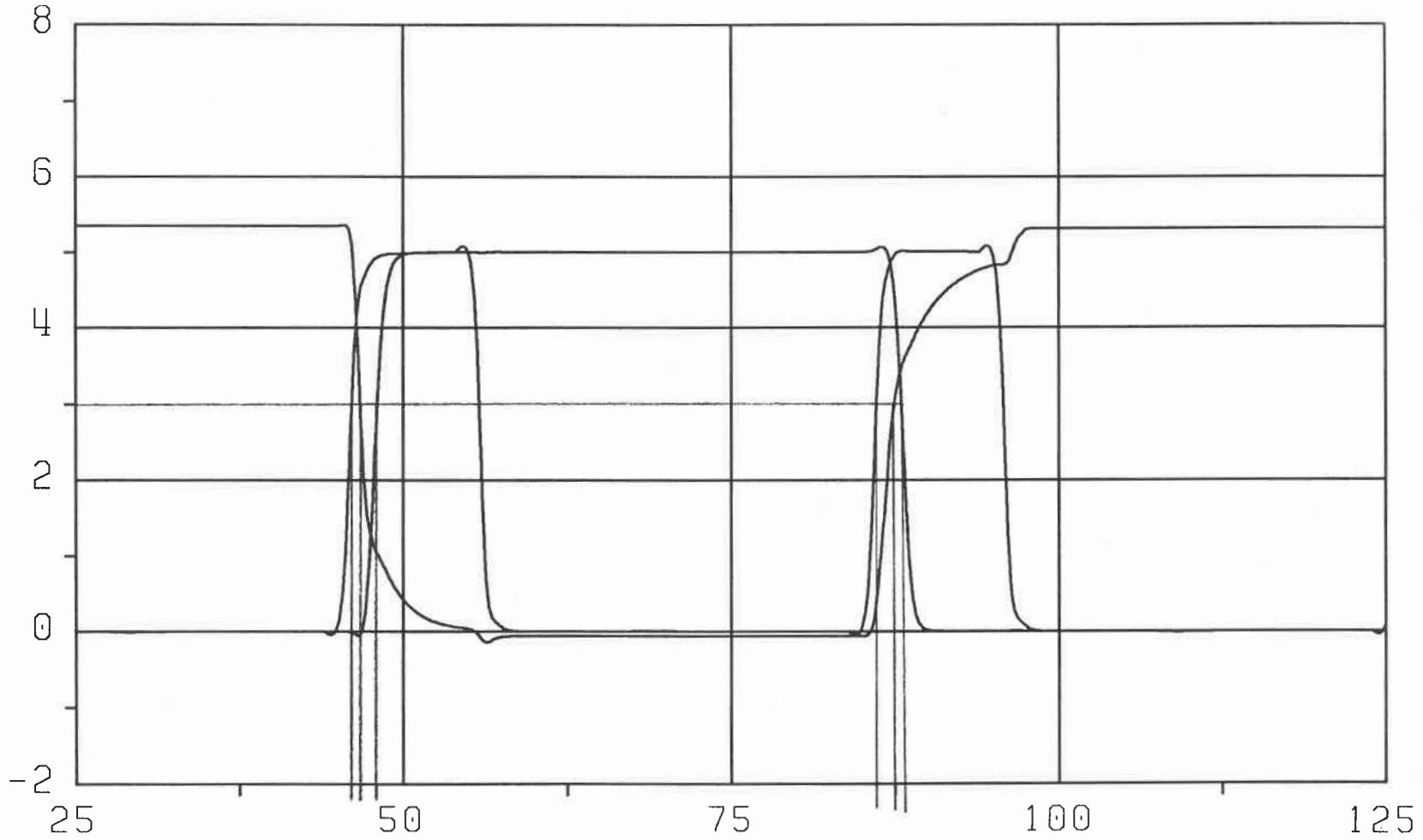
ANNEXE D

Simulation analogique de la cellule de base pour la mise à jour de la mémoire des chemins:

circuits et résultats




8
6
4
2
0
-2
VOLTAGE
50
75
100
125
150
TIME
$t_H \simeq 3.75$ ns
$t_L \simeq 2.81$ ns



$t^{L}_{N3} \simeq 0.875$ ns

$t^{H}_{STA} \simeq 1.97$ ns

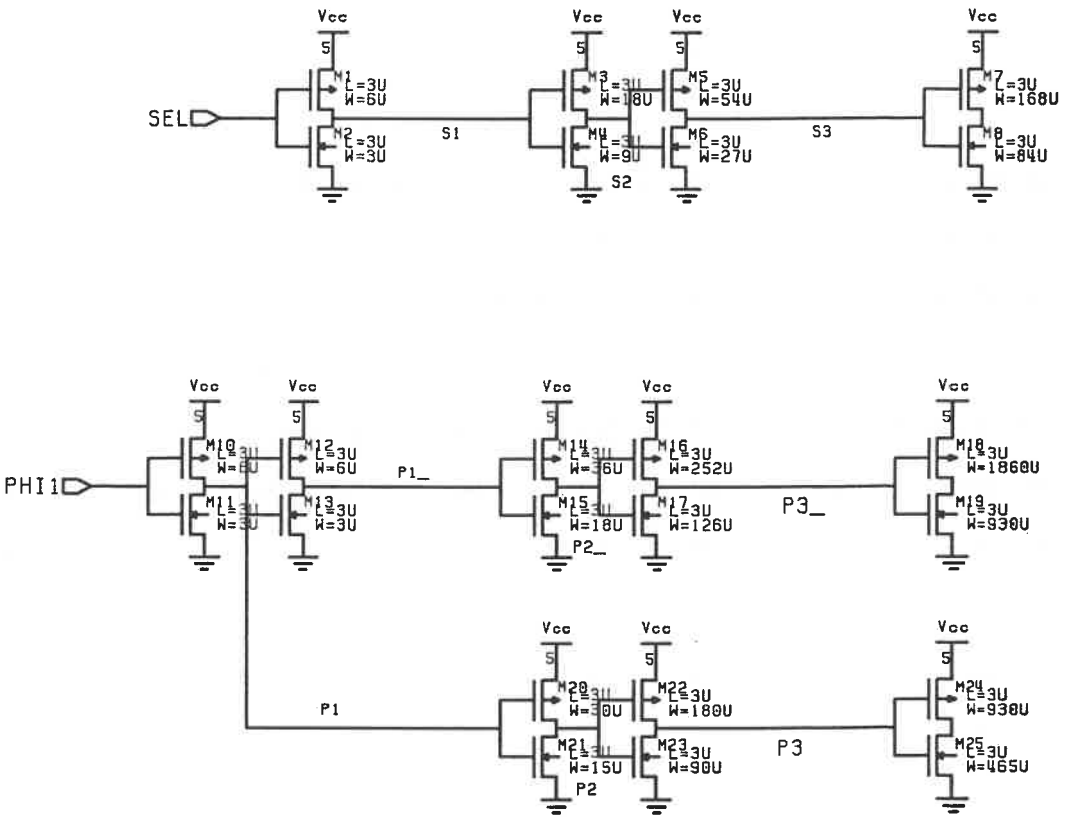
$t^{H}_{N3} \simeq 1.25$ ns

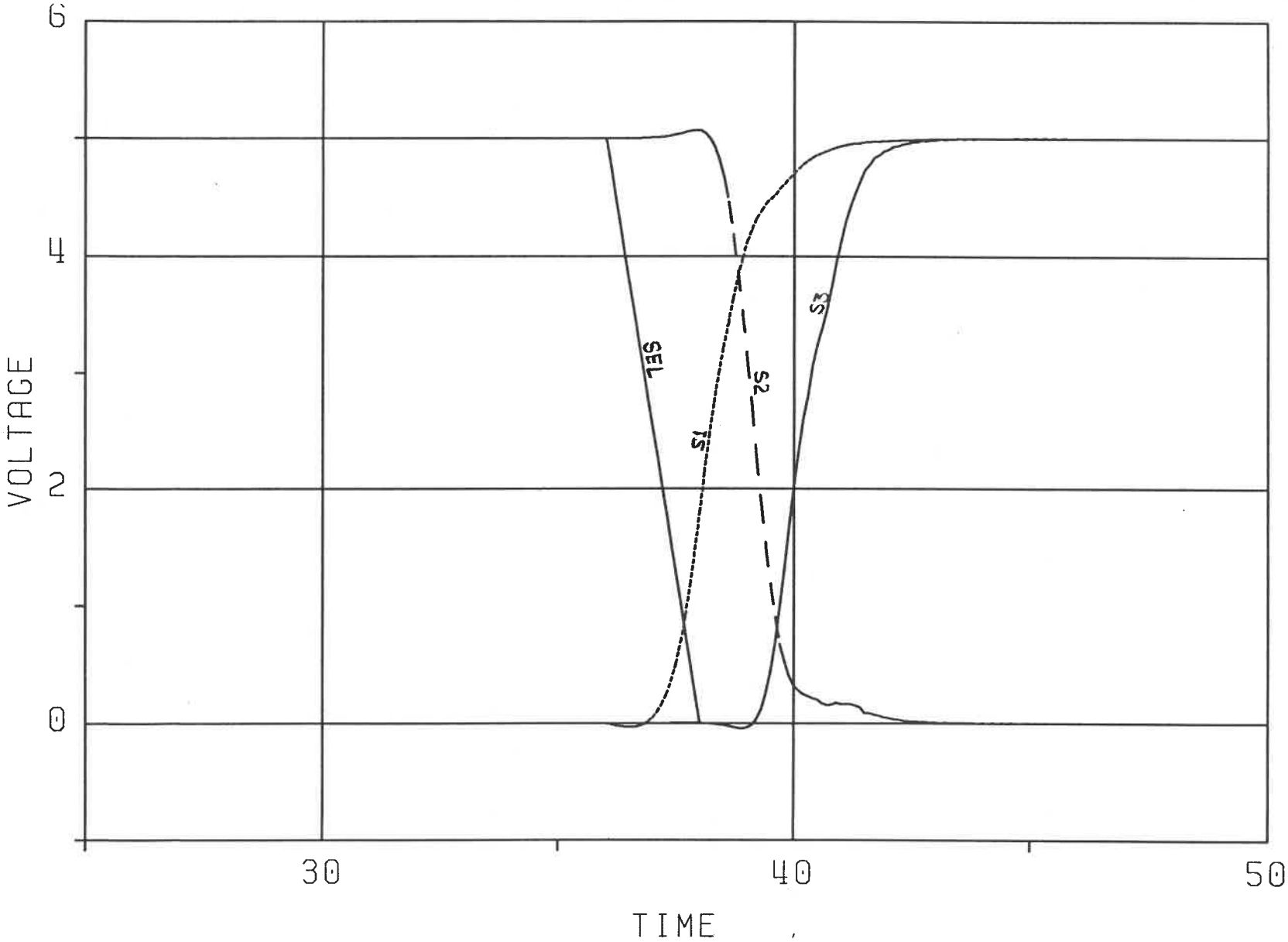
$t^{L}_{STA} \simeq 1.72$ ns

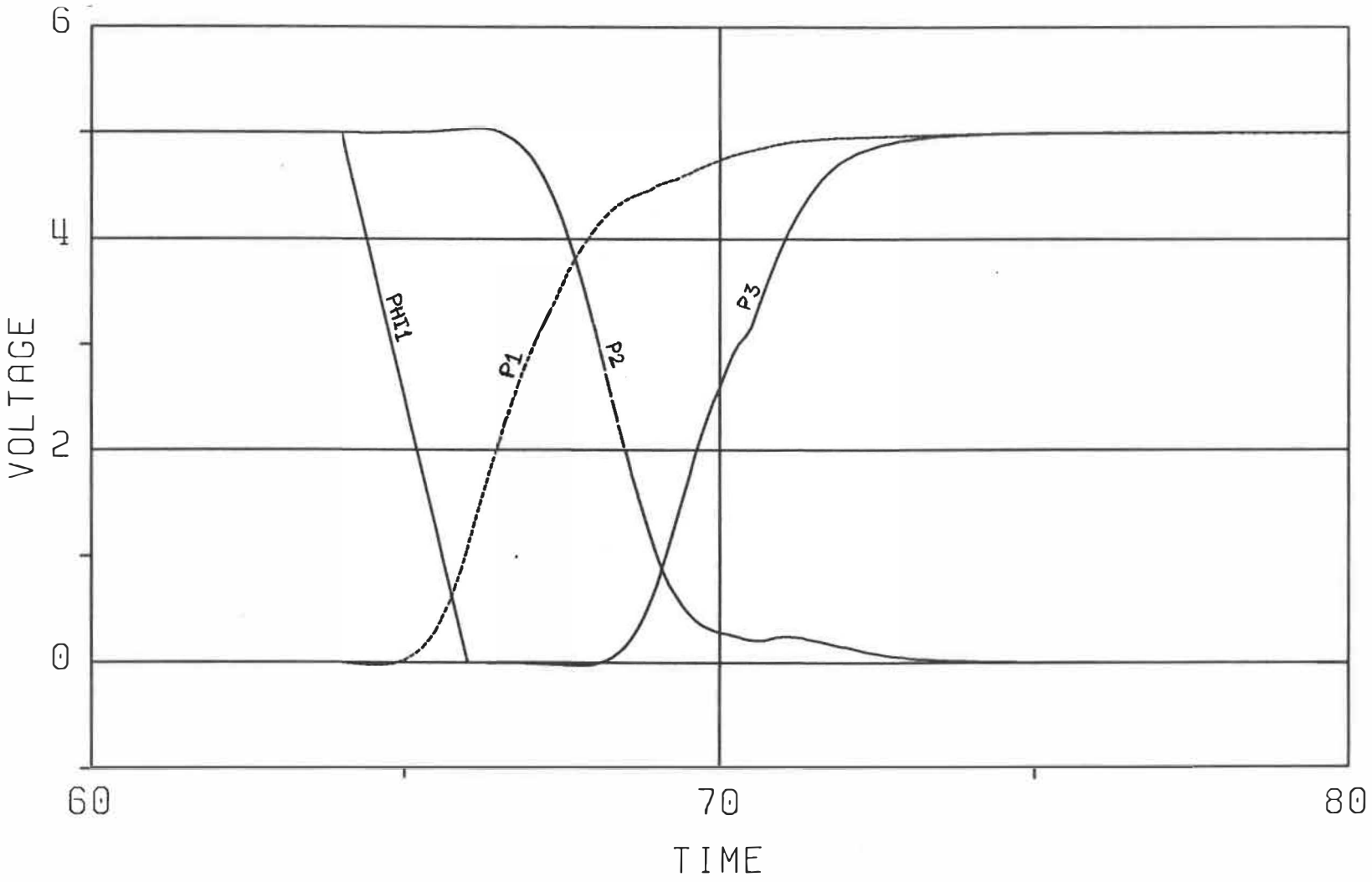
ANNEXE E

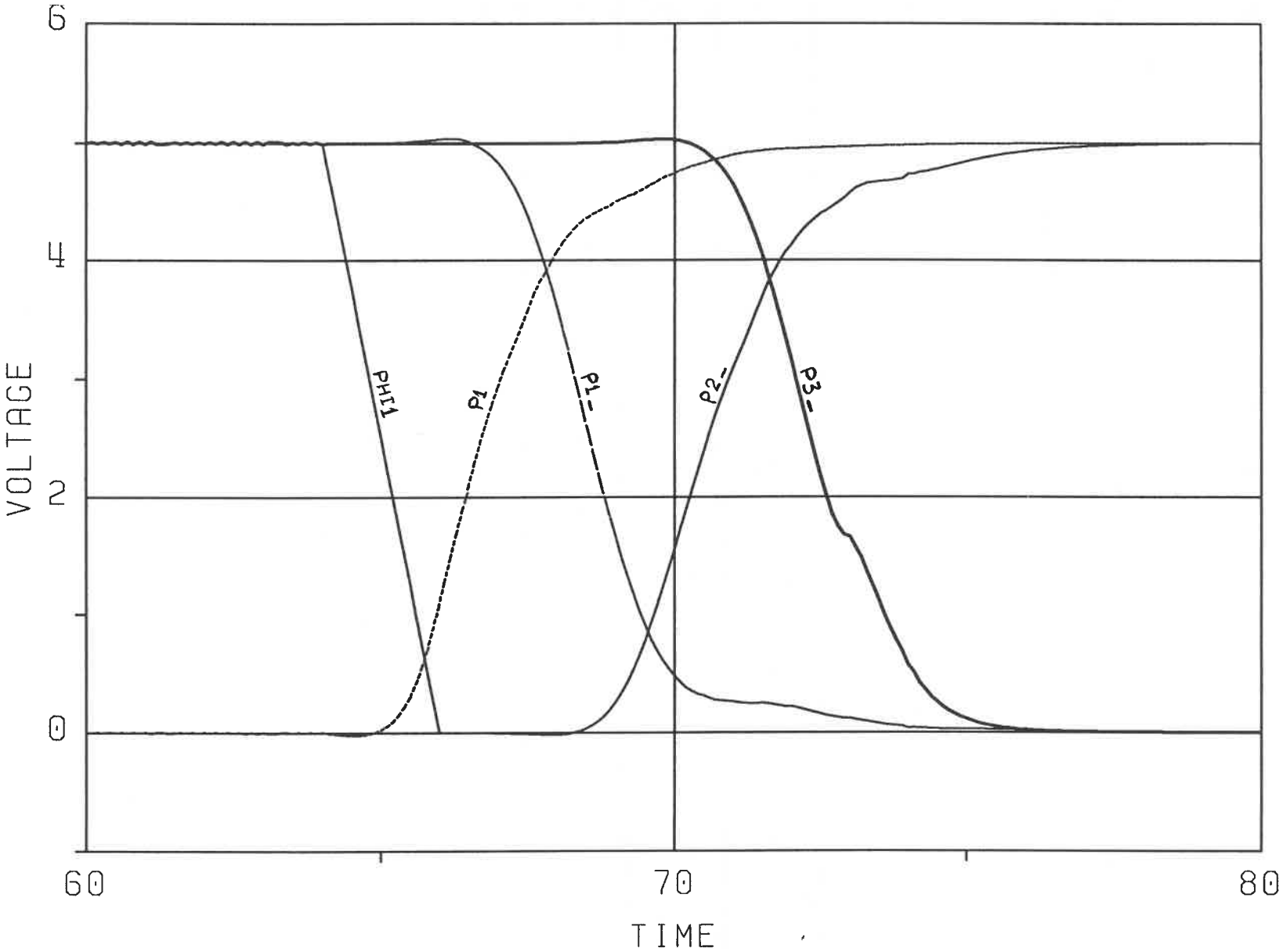
Simulations analogiques des circuits tampons:

circuits et résultats




6
4
2
0
VOLTAGE
SEL
S1
S2
S3
30
40
50
TIME


6
4
2
0
VOLTAGE
PHI1
P1
P2
P3
60
70
80
TIME


6
4
2
0
VOLTAGE
PHI1
P1
P1_
P2_
P3_
60
70
80
TIME