UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
THIERRY DUFOUR

ÉGALISEURS ADAPTATIFS PIPELINÉS PAR LA TECHNIQUE DE L'ANTICIPATION RELAXÉE

JANVIER 2002

Résumé

Ce mémoire propose une méthode de transformation d'algorithme et une intégration à très grande échelle (*ITGE – VLSI*) d'un égaliseur à base de réseaux de neurones afin d'améliorer la qualité des systèmes de communications numériques. Lors d'une transmission numérique sur un canal, le signal porteur d'information subit des dégradations importantes comme l'interférence inter-symbole, les non linéarités et le bruit. Ces dégradations dépendent du canal de communication utilisé (liaison hertzienne, paire de fil torsadé, guide d'onde, etc.) et le message reçu est souvent porteur d'erreurs. Pour corriger les effets du canal de communication, nous avons recours à la technique d'égalisation des canaux qui permet d'éliminer les erreurs sur le signal reçu et retrouver la séquence de données originales. La technique d'égalisation des canaux consiste à utiliser la sortie du canal et les informations du canal pour estimer le message original.

Puisque les paramètres du canal de communication sont souvent inconnus, on doit faire appel à des algorithmes adaptatifs pour permettre la reconstitution du signal transmis. Dans ce travail, l'adaptation de l'algorithme se fait à partir d'une séquence de données connues a priori du récepteur (type supervisé). De plus, la prise en compte des non linéarités du canal rend inefficace les méthodes adaptatives classiques tels les filtres transverses linéaires adaptés par l'algorithme *LMS* ou *RLS*. Nous avons donc recours à un

égaliseur à base de réseau de neurones artificiels (*RNA*) qui permet de traiter des canaux non linéaires.

Les algorithmes d'adaptation du *RNA* sont très souvent de nature récursive et leur application à de hauts débits de calculs est presque impossible. Nous proposons donc une technique de transformation d'algorithme permettant de modifier un algorithme d'adaptation du *RNA*. Cette transformation permet alors de pipeliner l'algorithme d'adaptation et ainsi opérer à de forts débits de calculs.

Des simulations à l'aide de données synthétiques ont été réalisées dans *Matlab*™. Ces simulations ont permis de démontrer les effets de la transformation de l'algorithme et nous ont permis de comparer la solution proposée à d'autres méthodes classiques. L'application à des canaux non linéaires est aussi réalisée. L'architecture de calcul proposée est hautement parallèle et la profondeur de pipeline est simple. Des simulations sur l'architecture ont été réalisées à l'aide de l'outil *Mentor Graphics*™ et les résultats sont comparés à ceux obtenus par *Matlab*™ afin de valider le fonctionnement. Une intégration de l'architecture en technologie CMOS 0.18µm dans *Synopsys*™ a permis d'évaluer les performances de l'architecture pipelinée.

Enfin, ce projet permet d'utiliser des algorithmes complexes fonctionnant à hauts débits dans la problématique d'égalisation des canaux. La réalisation de l'algorithme dans le domaine numérique rend son application fiable et peu coûteuse. De plus, la technique de transformation d'algorithme utilisée permet de modifier les caractéristiques de l'algorithme afin de traiter des cas spécifiques.

Remerciements

Je tiens à remercier en premier lieu mon directeur de recherche le professeur Daniel Massicotte. Son support et son expérience m'ont permis d'acquérir des connaissances inestimables en recherche. Sa grande compréhension et son côté humain m'ont par ailleurs permis de me réaliser pleinement sur tous les plans de ma vie tout au long de ce travail.

J'aimerais aussi remercier mes parents qui m'ont toujours encouragé et qui m'ont offert la possibilité de poursuivre mes études. Leur support moral a toujours été très apprécié et a grandement contribué à la réalisation de ce travail.

Enfin, je remercie mes collègues du Laboratoire des Signaux et Systèmes Intégrés (LSSI) pour l'aide qu'ils m'ont accordée. Les bons moments de discussion et de détente m'ont permis d'effectuer ce travail dans une ambiance motivante.

Table des matières

Liste des figures

Liste des tableaux

Liste des symboles et abréviations

Symboles

e	Erreur de correction
$\mathbf{q}$	Vecteur des poids du neurone de sortie
s	Symbole transmis
$\mathbf{W}$	Matrice des poids de la couche cachée
x	Signal analogique de la sortie du réseau
y	Signal à la sortie du canal
μ	Pas d'apprentissage
η	Bruit additif du canal
D_1	Relaxation du délai
D_2	Profondeur de pipeline de la boucle récursive
LA	Relaxation de somme
$\tilde{\bullet}$	Signal bruité
$\hat{\bullet}$	Estimé de

Abréviations

BER	Bit Error Rate
DFE	Decision Feedback Equalizer
ITGE -VLSI	Intégration à Très Grande Échelle – *Very Large Scale Integration*
LMS	Least Mean Squares

MSE	*Mean Square Error* – Erreur quadratique moyenne
PAM	Pulse Amplitude Modulation
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phased Shift Keying
RLS	Recursive Mean Squares
SNR	*Signal to Noise Ratio* – Rapport signal sur bruit
RNA	Réseau de neurones artificiels

Chapitre 1

Introduction

De nos jours, la communication de l'information prend de plus en plus de place et d'importance dans notre quotidien. Que ce soit la téléphonie, la télévision ou l'Internet, il est évident que nous sommes constamment entourés par des systèmes de communications. Cette popularité des communications nous pousse d'ailleurs à continuellement augmenter les capacités des systèmes de transmissions d'information et c'est cette quête d'augmenter le débit de données qui est la motivation principale des recherches effectuées en communications.

Dans les systèmes modernes, on utilise de plus en plus les circuits numériques pour la réalisation pratique. Cette approche nous permet d'atteindre de meilleures performances au niveau de la fiabilité, de la flexibilité, de la simplicité et de la qualité de transmission.

Dans les systèmes de communications, on retrouve le maillon le plus faible au niveau du canal de transmission. Ce canal peut entre autre être une paire de fil torsadé, une

fibre optique, une liaison hertzienne, etc. Puisque nous ne pouvons pas interagir sur les paramètres de ce bloc, le développement de plusieurs techniques de traitement des signaux ont été adoptées pour minimiser son effet. Une méthode bien connue de nos jours est appelée l'égalisation des canaux de transmission [HAY96]. Cette égalisation devient absolument nécessaire lorsque nous désirons transmettre des données à fort débit dans un environnement très hostile (bruit, distorsion de phase et d'amplitude, etc.).

L'égaliseur nous permet alors de corriger les effets du canal de transmission et de retrouver le signal original. De plus, l'algorithme de l'égaliseur doit être de nature adaptative dans le sens où les paramètres de calcul doivent pouvoir être modifiés dans le temps. Cette particularité permet de pallier les variations des paramètres du canal à travers le temps. De plus, cette adaptation de paramètres permet de régler le filtre avec une connaissance minimale du canal de transmission.

1.1 Problématique

Pour transmettre des données de nature numérique, on a souvent recours à des techniques de modulation numérique. Ces techniques permettent d'exploiter la bande passante utile du canal de transmission avec une meilleure efficacité. Par contre, le passage des données dans le canal de transmission provoque de l'interférence entre les symboles (*ISI*: Interférence Inter-Symboles) d'une même séquence de données. Alors, pour pouvoir récupérer la séquence de données originales, il faut absolument avoir recours à un égaliseur qui corrigera l'effet de l'*ISI*.

On peut résumer le problème de l'égalisation des canaux lorsqu'on désire transmettre une séquence de données $s(n)$ sur un canal quelconque de transmission. La sortie du canal nous donne le signal bruité $\tilde{y}(n)$ qui est lié au signal $s(n)$ original. Le signal $\tilde{y}(n)$ sera alors utilisé pour reconstituer la séquence originale. Cette opération de reconstitution consiste à générer un signal $\hat{s}(n)$ qui correspond à la séquence estimée de la séquence originale $s(n)$.

Plusieurs méthodes linéaires ont été proposées comme fonctions d'estimation [HAY96]. On retrouve par exemple les méthodes du gradient stochastique (*LMS*: Least Mean Square) et des moindres carrées (*RLS*: Recursive Least Square). Ces méthodes s'avèrent efficaces seulement si le système est linéaire. Dans certains systèmes, on retrouve la présence de non linéarité ce qui exige que l'égaliseur soit apte à éliminer cette non linéarité. Des propositions d'égaliseurs non linéaires ont d'ailleurs été présentées dans [VID99].

En plus de rencontrer les critères de performance selon un canal donné, les systèmes modernes doivent aussi rencontrer des critères d'implantation matérielle. Ces critères sont: la consommation en puissance, la vitesse maximale de calcul et d'exécution et le niveau de difficulté de l'implantation de l'algorithme (surface d'utilisation et/ou ressources matérielles). Pour rencontrer les critères de performances de l'application (taux d'erreurs binaires, temps d'adaptation, etc.), nous avons recours à des algorithmes adaptatifs qui permettent d'ajuster l'égaliseur. Afin d'atteindre de forts débits, on doit appliquer la

technique du pipeline sur l'architecture. Or, la plupart des algorithmes adaptatifs sont de nature récursive ce qui limite la profondeur de pipeline car l'algorithme a besoin de l'erreur à l'instant présent pour le calcul. Alors, pour pouvoir pipeliner les architectures récursives, nous avons recours à des techniques de transformation d'algorithmes qui introduisent des délais dans la boucle du calcul de l'erreur. Ces techniques nous permettent alors d'optimiser un ou plusieurs critères afin d'atteindre l'objectif visé par une application donnée.

L'application des techniques de pipeline classiques comme l'ajout de retard direct et la resynchronisation sont bien connues pour les algorithmes non récursifs [SHA98]. Dans le cas des algorithmes récursifs, diverses techniques de transformation d'algorithmes peuvent être employées mais leur application implique plusieurs compromis. Le projet de recherche proposé ici s'inscrit donc dans cette lignée et les principaux objectifs sont présentés à la section suivante.

1.2 Objectifs

Le but du présent projet consiste à appliquer les techniques de transformations sur les algorithmes récursifs actuellement rencontrés dans l'application à l'égalisation des canaux. La transformation de l'algorithme permettra d'augmenter le débit de l'égaliseur choisi. Le choix retenu doit tenir compte de l'intégration sur silicium de l'architecture résultante pour une éventuelle application réelle de l'égaliseur. De plus, la consommation en puissance sera aussi prise en considération dans le cas des applications portables. On peut résumer les objectifs de la recherche par les points suivants:

1. Études des algorithmes récursifs appliqués à l'égalisation des canaux selon les techniques de traitement numériques des signaux : algorithme *LMS*, algorithme *RLS*, filtre de Kalman, réseau de neurones.
2. Études des techniques de transformation d'algorithmes dans le but d'augmenter le débit des architectures. Attention particulière sur l'anticipation relaxée.
3. Proposition d'une méthode d'égalisation des canaux transformée par l'anticipation relaxée.
4. Évaluation des performances de la méthode retenue.

1.3 Méthodologie

Afin de bien cerner la problématique de transformation d'algorithme appliquée à l'égalisation des canaux en communication, on doit d'abord bien définir les différentes spécifications que l'égaliseur doit satisfaire. Pour ce faire, une analyse des différents canaux de transmission sera menée. Par la suite, nous verrons les principes et les différentes classes d'égaliseurs rencontrés dans les systèmes actuels sous la forme d'une synthèse [MAC98].

On abordera ensuite l'étude des différentes méthodes de transformation d'algorithmes en s'inspirant grandement de [SHA98]. Suite à l'étude des transformations, quelques méthodes comme l'anticipation, l'anticipation relaxée et la resynchronisation seront appliquées à des algorithmes simples comme le *LMS* pour en évaluer leur efficacité. L'étude des différents paramètres de la transformation permettra d'imposer les bornes

d'application et les grandes lignes d'influence des différentes techniques. On parlera alors de région de stabilité, de vitesse de convergence et d'erreur d'ajustement.

Des simulations à l'aide de l'outil *Matlab*™ seront menées et les résultats sous forme de graphiques et de tableaux seront présentés. Dans la mesure du possible, les résultats seront comparés aux valeurs théoriques pour vérifier les approximations et les hypothèses en cause.

1.4 Organisation de ce rapport

Nous verrons au Chapitre 2 les différents principes de l'égalisation adaptative des canaux en communications numériques. La section 2.1 traitera principalement de la communication numérique en général alors que la section 2.2 présentera les concepts des égaliseurs adaptatifs des canaux. Le chapitre 2 terminera sur les différentes contraintes quant à l'intégration des égaliseurs sur silicium.

Nous verrons au chapitre 3 les différentes techniques du pipeline. Les cas classiques seront présentés en 3.1 et 3.2 et nous verrons les différentes méthodes pour les algorithmes récursifs en 3.3 et 3.4. Par la suite, à la section 3.5, il sera question de l'application des techniques appliquées à l'égalisation des canaux avec un exemple d'application sur un filtre transverse linéaire adapté par la méthode *LMS*.

Le Chapitre 4 traitera du pipeline d'un égaliseur basé sur les réseaux de neurones. L'application des réseaux de neurones pour l'égalisation des canaux sera présenté en 4.1. Un rappel sur les réseaux de neurones en général sera ensuite présenté en 4.2 et nous verrons en 4.3 l'application de l'anticipation relaxée sur l'algorithme de rétropropagation du gradient. Les résultats de simulations seront enfin présentés en 4.4.

Le Chapitre 5 sera consacré à l'intégration de l'algorithme pipeliné de l'égaliseur à base de réseau de neurones. En 5.1, nous verrons l'architecture pipelinée du réseau de neurone. Nous présenterons en 5.2 les différents éléments de calculs dont l'architecture est composée. Enfin, en 5.3, nous présenterons les résultats de l'intégration dans *Synopsys*™ ainsi que des résultats de simulations qui démontrent le bon fonctionnement du réseau.

Chapitre 2

Égalisations adaptatives de canaux en communications numériques

L'élaboration d'un système de communication comporte une multitude de volets. Pour mieux cerner le problème de l'égalisation des canaux, il est important de bien connaître les différents paramètres entourant les systèmes de communication. Nous verrons donc en premier lieu les systèmes de communications d'une façon générale en présentant les différents blocs du système. Par la suite, nous aborderons les principes de bases en communication numérique avec les techniques de modulation et de démodulation. Les différents modèles de canaux de communication seront aussi présentés. Les égaliseurs adaptatifs seront abordés en présentant leur principe et les différentes techniques utilisées. Nous verrons aussi les différentes techniques d'adaptation de ces égaliseurs.

On peut représenter un système de communication par le schéma général de la Figure 2.1.

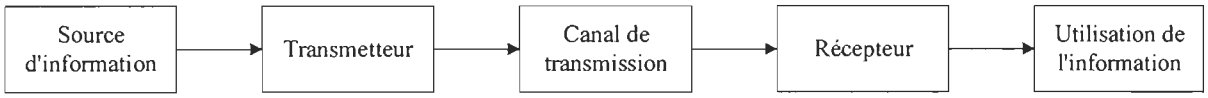


Figure 2.1: Système de communication général

Dans ce schéma général, on retrouve une source d'information qui peut être de la voix, des données, des mesures, etc. Cette information est ensuite acheminée vers un transmetteur qui transforme le signal de la source en une information qui peut être transmise sur le canal de transmission. Une fois que le signal est passé à travers le canal de transmission, il faut retrouver le message original. Pour ce faire, le bloc récepteur crée l'inverse du transmetteur et l'utilisateur ou le système peut ensuite récupérer l'information pour l'utiliser.

2.1 La communication numérique

Les avantages d'un système numérique par rapport aux systèmes analogiques ne sont plus à démontrer. Avec les systèmes numériques, il est possible d'améliorer grandement la flexibilité des communications. Nous sommes aussi en mesure d'atteindre de meilleures performances au niveau de la fiabilité, de la simplicité et de la qualité de transmission. Avec les systèmes numériques, nous avons beaucoup plus de latitude sur le contrôle de la qualité du signal. Par exemple, dans les transmissions sur longue distance, il est possible de régénérer le signal transmis par des répétitrices ce qui élimine l'effet du canal et du bruit sur le signal original. De cette façon, le signal reçu correspond exactement à ce qui a été transmis. En communication analogique, une telle régénération améliore certes la qualité du

signal reçu mais le signal régénéré est toujours de moindre qualité que le signal transmis [PRO95].

Un autre avantage se situe au niveau de la redondance dans l'information. Dans les systèmes numériques, il est possible d'éliminer cette redondance avant la modulation permettant de conserver la bande passante utile du canal de transmission au maximum. Enfin, le coût des systèmes de communications est souvent moindre que celui des systèmes de communications analogiques offrant les mêmes performances.

Le schéma général d'un système de communication numérique est présenté à la Figure 2.2 :

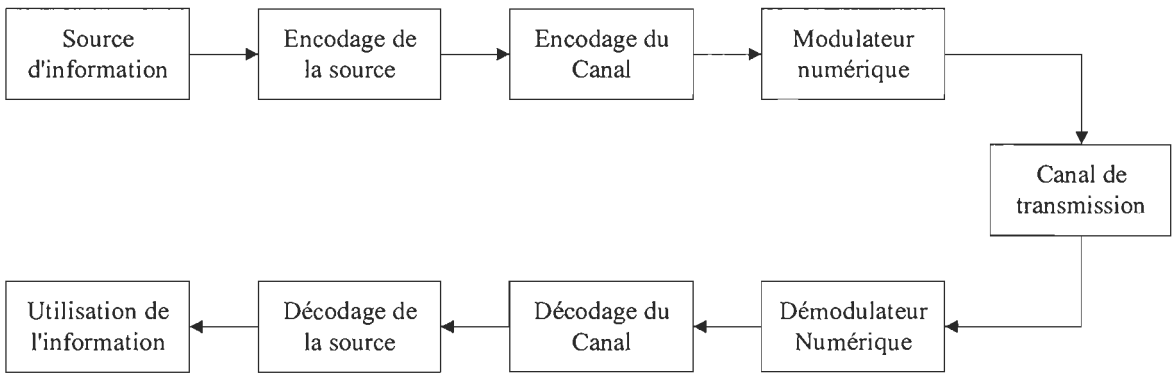


Figure 2.2: Système de communication numérique

Dans ce schéma, la source d'information peut être un signal de nature numérique ou analogique. Dans le cas d'un signal analogique, il faut le convertir en un signal numérique de dimension finie. Dans la plupart des cas, les signaux numériques sont en fait une séquence de nombres binaires. De façon idéale, il faut coder l'information sur le moins de bits possibles pour garder le débit d'information le plus bas possible. Cette fonction de

quantification est assurée par le bloc *Encodage de la source*. On peut aussi retrouver des techniques de compression des données dans ce même bloc.

Par la suite, le signal est introduit dans le bloc *Encodage du Canal* dont la fonction principale est de contrôler la redondance rencontrée dans la séquence binaire. Cette opération permet de minimiser l'effet du bruit introduit par le canal sur le signal et ainsi améliorer la qualité à la réception.

Ensuite, nous avons le modulateur numérique qui permet d'interfacer la séquence binaire au canal de transmission. Pour ce faire, le modulateur transforme la séquence en un signal électrique donné. Il est possible de transmettre les bits b un à la fois ou en les regroupant pour transmettre un groupe de bits en même temps. Cette dernière méthode est connue sous le nom de modulation *M-aire* où M représente le nombre de groupe possible ($M=2^b$). Nous verrons plus de détails sur ces techniques au cours du rapport.

Le signal est ensuite introduit dans le canal de transmission. Ce canal sera expliqué en détails plus loin et on peut résumer son effet en disant qu'il introduit une distorsion (phase, amplitude et bruit) sur le signal transmis.

À la réception, on retrouve le démodulateur numérique qui est idéalement le traitement inverse du modulateur. Cette section peut devenir très élaborée car le signal reçu est très différent de ce qui a été transmis. C'est d'ailleurs au niveau de ce bloc que la majorité des recherches s'effectuent de nos jours. Avant de prendre une décision sur le signal reçu, il

faut éliminer l'effet du canal. Cette opération est souvent effectuée par l'égaliseur. Nous verrons plus loin les différents types d'égaliseurs que nous retrouvons aujourd'hui.

2.1.1 Principes de modulation et de démodulation

Toujours dans le but de minimiser le taux de données transmises, les techniques de modulations numériques permettent de regrouper l'information dans le but de la condenser sur les différents paramètres de la porteuse. Le but ici est de maximiser l'utilisation de la bande passante disponible sur le canal de transmission. Voyons le principe de la modulation numérique dans le cas d'une liaison hertzienne.

Dans les applications de communication sans fil, il est absolument nécessaire d'utiliser une onde porteuse à haute fréquence. En effet, la longueur des antennes de transmission et de réception doit être d'environ $\lambda/4$ où λ représente la longueur d'onde de la porteuse. Cette longueur d'onde est donnée par (2.1).

$$\lambda = \frac{c}{\sqrt{\varepsilon_r} f} \tag{2.1}$$

où c représente la vitesse de la lumière, ε_r représente la permittivité relative du milieu et f est la fréquence de la porteuse.

Dans l'air, la permittivité relative est d'environ 1 et la vitesse de la lumière est fixe à $3x10^8$ m/s. Dans le cas où la porteuse serait de 1kHz, la longueur d'onde nous donnerait λ=300km. Il est évident qu'une antenne de 75km ($\lambda/4$) est physiquement très difficile à réaliser et très peu pratique. Donc, pour faciliter la réalisation pratique, des fréquences de

l'ordre des Mégahertz (*MHz*) sont utilisées pour transmettre l'information. Par exemple, certains téléphones sans fils utilisent une porteuse de 1.2GHz ce qui donne λ=25cm d'où la longueur de l'antenne est de 6.25cm. Cette longueur est donc facilement réalisable pour les applications portables.

Puisque nous utilisons des fréquences pures comme porteuse, nous pouvons écrire l'équation générale d'une onde sinusoïdale par:

$$p(t) = A\cos(\omega t + \theta) \tag{2.2}$$

Dans cette équation, nous retrouvons l'amplitude de l'onde A, la fréquence angulaire ω et la phase θ. Pour moduler l'information sur la porteuse, il s'agit de faire varier un des trois paramètres (A, θ, ω) par le signal porteur d'information (signal modulant). On peut donc réécrire (2.2) sous la forme:

$$p_M(t) = A(t)\cos(\omega(t)t + \phi + \theta(t)) \tag{2.3}$$

Où $A(t)$, $\omega(t)$ et $\theta(t)$ sont les paramètres qui sont régis par le signal modulant.

Dans les communications analogiques, on retrouve trois types de modulation: la modulation d'amplitude (*AM*) qui utilise la fonction $A(t)$, la modulation de fréquence (*FM*) qui utilise la fonction $\omega(t)$ et la modulation de phase (*PM*) qui utilise la fonction $\theta(t)$.

Dans les communications numériques, les fonctions $A(t)$, $\omega(t)$ et $\theta(t)$ sont contrôlées par l'information binaire à transmettre. L'information binaire peut être représentée par

différentes formes d'ondes. Sa forme la plus simple est certes l'onde tout ou rien tel que montré à la Figure 2.3.

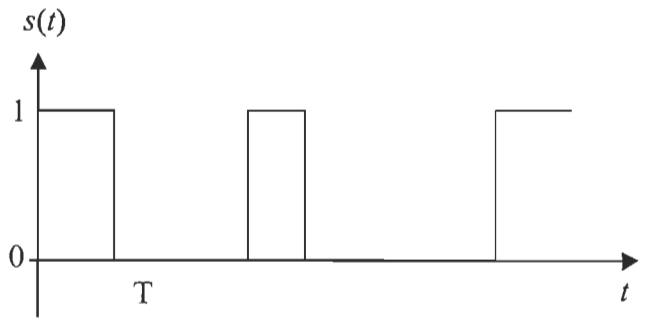


Figure 2.3: Information binaire classique

Une autre représentation des données peut se faire avec une moyenne nulle que l'on nomme non retour à zéro (*NRZ*). La Figure 2.4 montre bien la représentation de ce type de données.

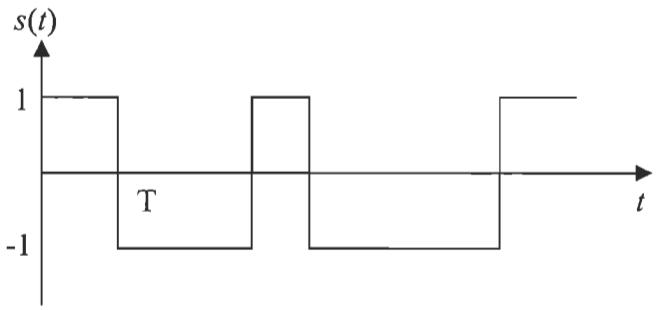


Figure 2.4: Format de données NRZ

Pour transmettre les données sur le canal, l'information binaire contrôle la valeur de l'amplitude, de la phase ou de la fréquence de l'onde porteuse. Une combinaison des paramètres est aussi possible. La modulation numérique la plus simple est la modulation d'amplitude tout ou rien (*ASK: Amplitude Shift Keying ou OOK: On Off Keying*). Le principe de cette modulation est de transmettre la porteuse seulement lorsque la valeur du bit au temps t est d'amplitude A. Si la vitesse de la porteuse est beaucoup plus grande que la

vitesse de l'information, on peut obtenir le signal modulé tel que montré à la Figure 2.5 (à titre d'exemple).

On voit dans cet exemple que la séquence transmise est 01101 (largeur des bits de 0.2 seconde) ce qui correspond aux endroits où la porteuse est présente ou non. Ce genre de modulation est utilisé dans le cas des communications à faibles débits sans contrainte dans l'utilisation spectrale du canal. Si on désire améliorer le taux d'utilisation de la bande passante, il est nécessaire de varier simultanément plusieurs paramètres de l'onde porteuse.

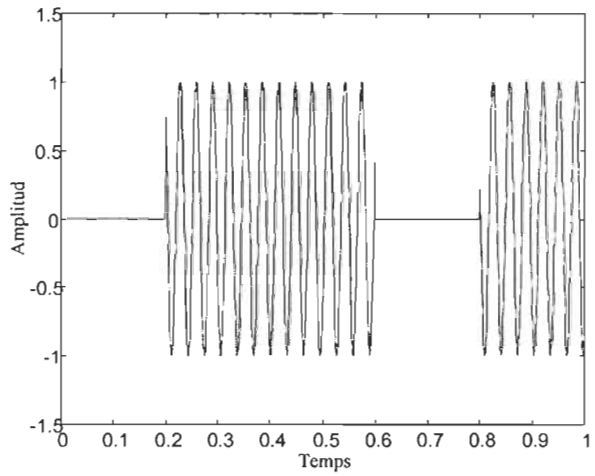


Figure 2.5: Signal ASK

Ce type de modulation n'utilise non pas un seul bit à la fois mais elle regroupe les bits en M symboles selon une règle définie. La valeur de M est définie par le nombre de bits b encodés à la fois soit $M = 2^b$. On appelle ce type de regroupement *M-aire*.

La façon de transmettre les symboles donne lieu au type de modulation en cause. On retrouve plusieurs méthodes de modulation des symboles et les principales sont la

modulation de phase (*PSK*: *Phase Shift Keying*) et la modulation de quadrature (*QAM*: *Quadrature Amplitude Modulation*). Selon le nombre de bits codés dans un même symbole, on nomme les modulations en fonction de *M* soit *M-PSK* ou *M-QAM*. Voyons maintenant les grandes lignes de la modulation *M-QAM*.

Le principe de la modulation *QAM* est de varier la phase et l'amplitude de la porteuse pour transmettre les symboles. Nous avons alors recours à une représentation dans le plan complexe de la résultante du codage des symboles. La Figure 2.6 nous montre le plan complexe pour une modulation *32-QAM*.

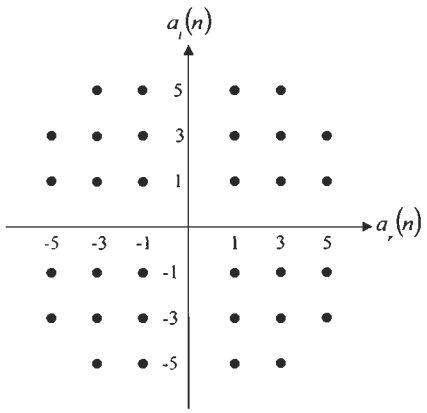


Figure 2.6: Constellation *32-QAM*

Nous avons alors deux variables soient $a_r(n)$ et $a_i(n)$ qui représentent respectivement la valeur de la composante réelle et complexe du symbole à transmettre. Les points représentent ici la valeur de l'amplitude et de la phase de la porteuse à transmettre. Dans le cas du *M-PSK*, la valeur de l'amplitude est fixe et ce n'est que la phase qui change. La Figure 2.7 nous montre le cas du *8-PSK* afin de voir la différence avec la modulation *QAM*.

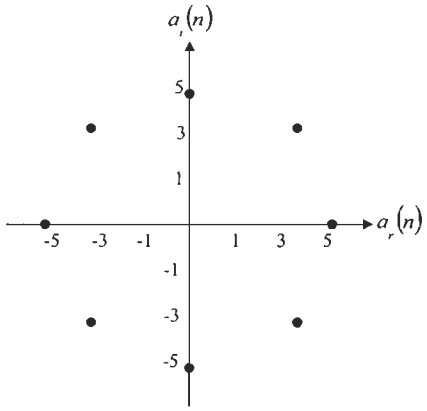


Figure 2.7: Constellation *8-PSK*

Voyons maintenant le schéma général du modulateur *QAM* que nous présentons à la Figure 2.8.

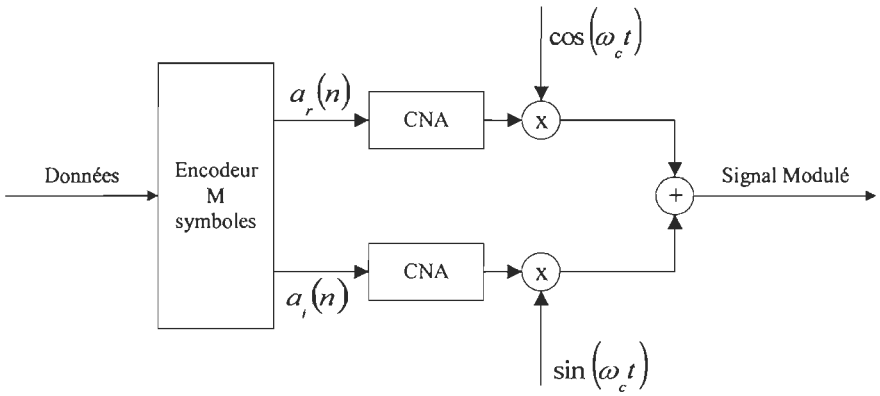


Figure 2.8: Modulateur *QAM*

On retrouve en premier lieu un encodeur qui associe les valeurs à $a_r(n)$ et $a_i(n)$. Par la suite, ces valeurs réelles et complexes sont converties en échantillons analogiques (*CNA: Convertisseur Numérique à Analogique*) et sont multipliées par les deux ondes porteuses en quadrature. La sommation des deux ondes donne alors le signal contenant l'information

complexe du symbole original. Ce signal résultant peut être modulé à son tour dans le but de le déplacer en fréquence et ainsi être transmis sur le canal de transmission.

Ce genre de modulation est aujourd'hui très populaire. On retrouve d'ailleurs, à titre d'exemple, les modems de communication sur la ligne téléphonique. Pour augmenter le débit de transmission et aussi améliorer le taux d'utilisation du canal, nous n'avons qu'à augmenter le nombre de symboles. Par contre, l'augmentation du nombre de symboles demeure un risque car le bruit du canal rend la détection difficilement réalisable de façon simple. En effet, puisque le canal de transmission et la variation de l'amplitude des symboles sont limités, la distance entre 2 symboles de la constellation devient très petite lorsque M augmente. Or, en présence d'interférence inter-symbole et de bruit, le récepteur ne pourra récupérer sans erreur les symboles transmis. Il est alors nécessaire de minimiser l'effet du canal sur le signal porteur d'information et c'est dans cette optique que la notion d'égaliseur entre en jeu.

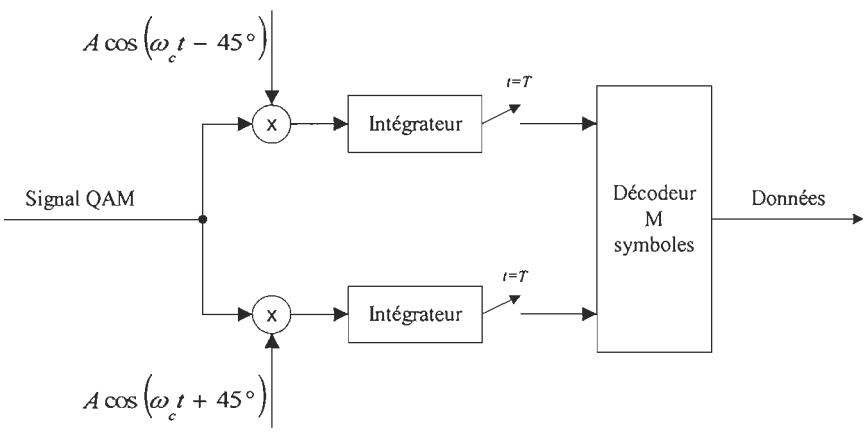


Figure 2.9: Démodulateur *QAM*

Dans le cas des récepteurs optimums, on utilise des filtres adaptés pour la démodulation et la décision des signaux reçus. On peut représenter le démodulateur *QAM* optimum par le schéma de la Figure 2.9 [STR90]. Ce genre de démodulateur est appelé détecteur de corrélation. Si le canal est plus hostile, il devient nécessaire d'utiliser un égaliseur. Avant d'aborder ce sujet, voyons les différents types de canaux de transmissions.

2.1.2 Modèles de canaux de transmission

Tel que nous l'avons mentionné précédemment, le canal de transmission constitue en grande partie la problématique d'un système de communication. Peu importe sa nature, il demeure un paramètre que nous ne pouvons pas changer. Au mieux, nous pouvons en élaborer un modèle très précis mais dans plusieurs cas, un modèle unique ne peut être clairement établi. Une analyse statistique est alors nécessaire afin d'obtenir un modèle de canal représentatif de la réalité.

Les canaux de communications sont souvent modélisés sous forme mathématique pour permettre de mesurer les performances du système dans un environnement donné. Cette modélisation est basée surtout sur des observations et représente une approximation acceptable de la réalité. Le modèle du canal est divisé en deux sections. Nous avons premièrement sa réponse impulsionnelle et ensuite le type de bruit ajouté (additif, multiplicatif, blanc Gaussien, etc.).

En pratique, on retrouve principalement 4 types de canaux de transmission que nous présentons ici :

- Canal linéaire et invariant dans le temps (*LTI*)
- Canal linéaire et variant dans le temps (*LTV*)
- Canal non linéaire et invariant dans le temps (*NLTI*)
- Canal non linéaire et variant dans le temps (*NLTV*)

Les systèmes opérant avec des canaux linéaires sont évidemment plus simples à réaliser. Le schéma de la figure suivante représente le modèle du canal linéaire et variant dans le temps.

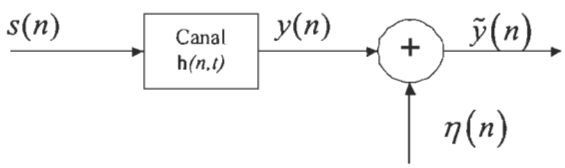


Figure 2.10: Schéma du canal linéaire et variant dans le temps

Dans le schéma de la Figure 2.10, $s(n)$ représente les données numériques originales. $\mathbf{h}(n,t)$ représente la réponse impulsionnelle du canal en cause. Le coefficient $\eta(n)$ représente un bruit blanc Gaussien de moyenne nulle. Enfin, $\tilde{y}(n)$ représente le signal à l'entrée du récepteur.

On peut montrer que le canal linéaire se comporte souvent comme un filtre passe-bas à réponse impulsionnelle finie (*FIR*) causale, dont l'équation est une simple convolution [PRO95]. Pour un canal linéaire et invariant dans le temps, l'équation du canal est :

$$\tilde{y}(n) = \left\{ \sum_{p=0}^{P-1} h(p) s(n-p) \right\} + \eta(n) \tag{2.4}$$

Dans cet exemple, $\tilde{y}(n)$ est le signal à l'entrée du récepteur, $h(p)$ représente les coefficients de la réponse impulsionnelle du canal (de longueur P), $s(n)$ représente la séquence de données transmises et $\eta(n)$ représente le bruit à l'instant n.

Un modèle de canal non linéaire est souvent représenté par une partie linéaire, une partie non linéaire et un bruit additif. Le schéma de la Figure 2.11 nous montre un tel type de canal.

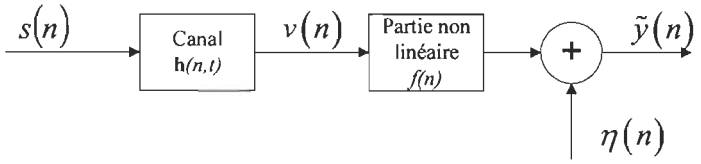


Figure 2.11: Modèle de canal non linéaire

L'équation régissant un tel type de canal peut être donnée, à titre d'exemple, par l'équation (2.5).

$$\begin{aligned} \tilde{y}(n) &= 0.5v(n) + v^3(n) + \eta(n) \\ v(n) &= 0.25s(n-2) + s(n-1) + 0.25s(n) \end{aligned} \tag{2.5}$$

On remarque dans cet exemple que la non linéarité du canal est de forme polynomiale ce qui est souvent le cas en pratique. Il est possible de créer un égaliseur non linéaire permettant de minimiser son effet. De plus, il est possible que les coefficients du canal

varient en fonction du temps (canal *NLTV*) ce qui implique une adaptation des paramètres de l'égaliseur dans le temps.

2.2 Égaliseurs adaptatifs des canaux

Puisque nous ne pouvons pas interagir sur les paramètres du canal de transmission, plusieurs techniques de traitement des signaux ont été développées pour minimiser son effet. On appelle cette méthode l'égalisation des canaux de transmission.

2.2.1 Principe de l'égalisation

Le premier rôle de l'égaliseur dans les systèmes de communication est d'annuler l'effet du canal sur le signal porteur d'information. Plusieurs méthodes peuvent être employées selon le type de canal que nous avons. Le schéma de la Figure 2.12 illustre bien la position de l'égaliseur dans un système de réception en communication numérique sans fil.

En pratique, on retrouve principalement 2 types de récepteurs. Le premier type est appelé récepteur optimal. Ce genre de récepteur est caractérisé par un égaliseur dont les paramètres de correction sont fixes dans le temps. C'est-à-dire que nous connaissons a priori les caractéristiques (réponse impulsionnelle ou réponse fréquentielle) du canal utilisé. Avec ces paramètres, la conception de l'égaliseur revient à créer l'inverse du canal de transmission.

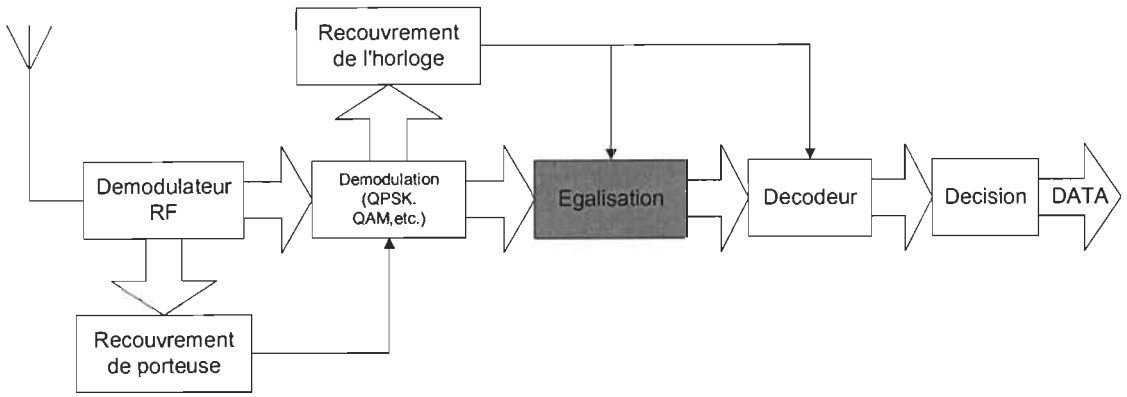


Figure 2.12: Récepteur en communications numériques sans fil

Dans certains cas, l'utilisation de ce type de récepteur permet d'obtenir des performances satisfaisantes et l'avantage important réside dans la rapidité d'exécution de ce genre d'égaliseur. Le schéma de la Figure 2.13 nous montre un tel type d'égaliseur.

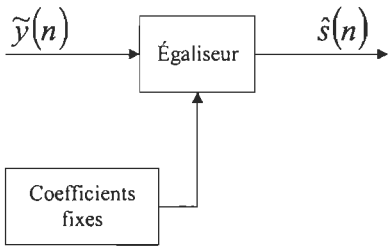


Figure 2.13: Schéma du récepteur optimum

Le deuxième type de récepteur est nommé sous optimum car la fonction de transfert de l'égaliseur est créée à partir d'une estimation des paramètres du canal. Dans ce cas, il est possible d'avoir des paramètres d'égalisation variables dans le temps. Le récepteur est donc doté d'un algorithme qui adapte les coefficients de l'égaliseur. Ces paramètres peuvent être calculés à partir d'une séquence de données connues du transmetteur et du récepteur (mode supervisé) ou de façon autodidacte par un retour statistique sur la sortie de l'égaliseur. La

Figure 2.14 montre un tel type de récepteur (à noter que les types autodidactes et supervisés sont tous deux montrés) :

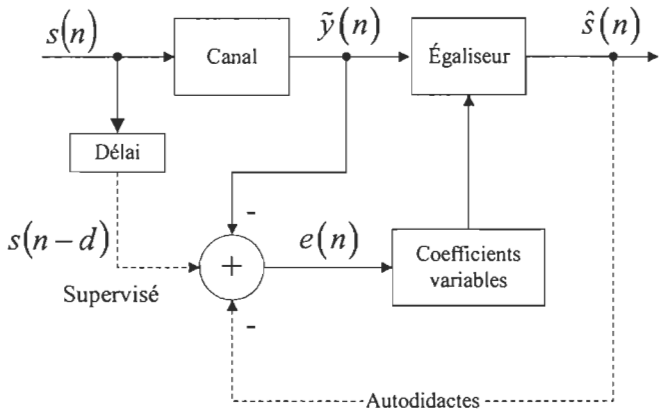


Figure 2.14: Schéma du récepteur sous optimum

2.2.2 Les techniques de l'égalisation

Les égaliseurs peuvent être divisés en trois familles principales :

- Égaliseurs transverses linéaires
- Égaliseur par retour de décision (*Decision Feedback Equalizer « DFE »*)
- Égaliseur par maximum de vraisemblance (*Maximum Likelihood Sequence Estimator « MLSE »*)

L'architecture des égaliseurs transverses linéaires est donnée par la Figure 2.15. Dans ce schéma, on suppose une égalisation de type supervisée. Pour ce type d'égaliseur, l'algorithme d'adaptation utilise l'erreur entre la sortie de l'égaliseur et la donnée originale. On qualifie cette différence d'erreur de reconstitution. Dans la famille des égaliseurs

transverses linéaires, on retrouve principalement les algorithmes d'adaptation suivants: *Least Mean Square* (*LMS*), *Recursive Least Square* (*RLS*) et les filtres de Kalman [MAC98]. Nous verrons plus loin un résumé de ces techniques.

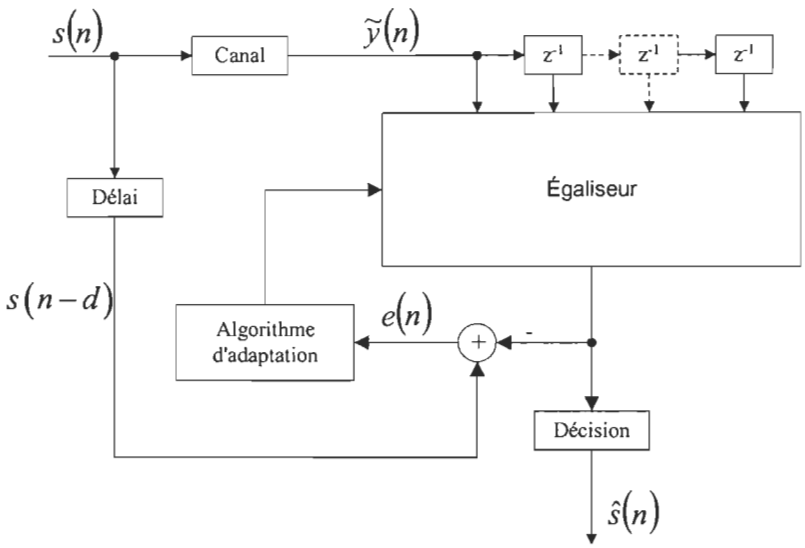


Figure 2.15: Égaliseur transverse linéaire

La Figure 2.16 nous montre le schéma de l'égaliseur *DFE*. Ce type d'égaliseur possède une nature non linéaire. Ce type d'égaliseur est caractérisé par l'ajout d'un deuxième filtre transverse dans la rétroaction. Ce filtre permet, pour l'estimation présente, d'éliminer l'interférence inter symbole causée par la détection des symboles précédents [HAY95]. L'adaptation des coefficients des filtres se fait de façon linéaire par un algorithme *LMS* ou *RLS*.

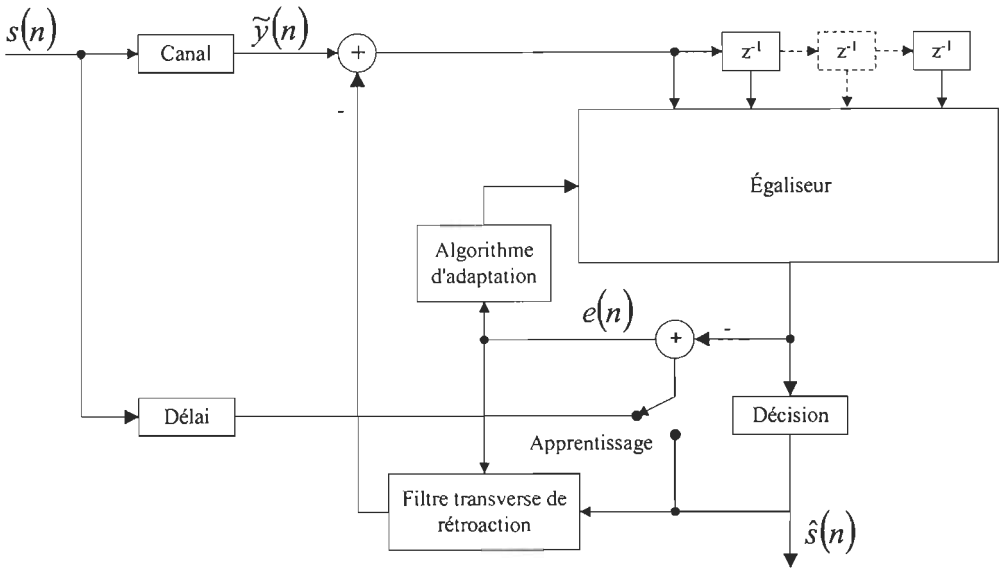


Figure 2.16: Égaliseur à retour de décision

Une autre remarque intéressante concerne le mode d'apprentissage. Selon la Figure 2.16, l'erreur $e(n)$ peut être calculée à partir de la décision ou de la séquence originale. Prenons le cas où l'on utilise la décision pour le calcul de $e(n)$. Si le canal en cause n'est pas connu et que les poids des filtres sont initialisés à 0, l'égaliseur peut mettre un temps très long avant de converger vers un minimum. Cet effet s'explique par le fait que la décision $\hat{s}(n)$ n'est pas valide pour les premiers échantillons. Il faut alors incorporer une phase d'apprentissage qui utilise la séquence originale $s(n)$ pour adapter les poids des filtres. Une fois la convergence obtenue, on peut utiliser la décision pour le calcul de l'erreur ce qui élimine la nécessité de connaître les données originales. Il est bon de souligner que les variations dans les paramètres du canal ne doivent pas être trop rapides pour conserver les performances de l'égaliseur. Enfin, si les paramètres du canal sont

connus, les poids peuvent être initialisés à des valeurs prédéfinis ce qui élimine la phase d'apprentissage.

Le troisième type d'égaliseur est appelé égaliseur du maximum de vraisemblance (*MLSE*). L'approche consiste en l'estimation de la séquence transmise la plus vraisemblable. En effet, puisque les canaux créent une corrélation entre les échantillons (convolution), il est possible de retirer cette corrélation des données en connaissant ses paramètres. Ce genre d'égaliseur nécessite donc la connaissance des paramètres du canal en cause. Si le canal en cause est variant dans le temps, il est nécessaire d'intégrer un estimateur de canal adaptatif. Ce dernier peut être basé sur le filtre de Kalman ou un filtre transverse linéaire avec adaptation par *LMS* ou *RLS*.

L'algorithme de Viterbi est le plus répandu dans la détection des codes convolutifs. Son principal désavantage réside dans sa complexité de calcul et c'est pour cette raison qu'on le retrouve très rarement en pratique. La Figure 2.17 nous montre l'application de ce genre d'égaliseur. Nous pouvons mentionner que ce type d'égaliseur est très performant. Il sert souvent de référence afin de comparer les performances des différents types d'égaliseurs [HAY96].

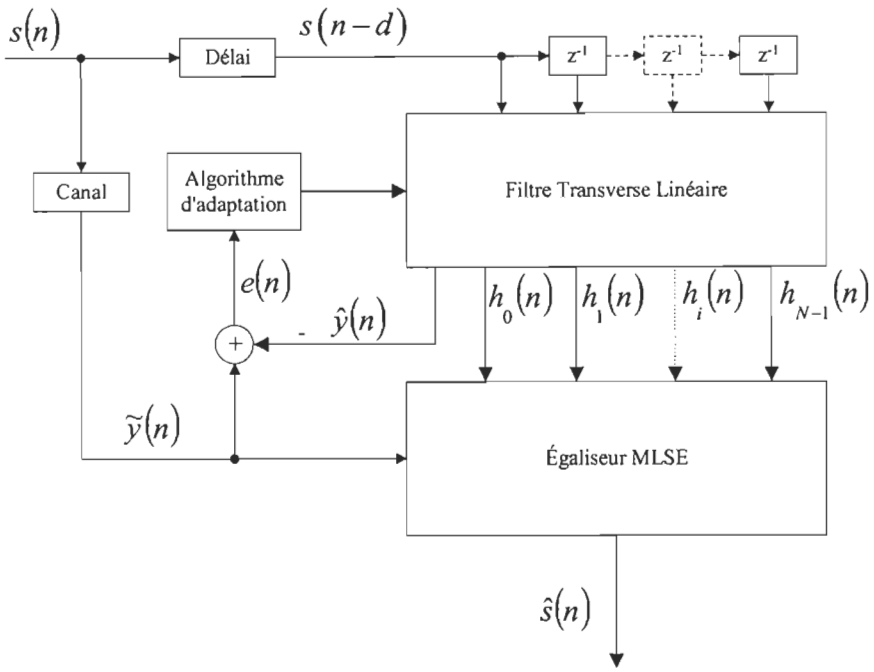


Figure 2.17: Égaliseur MLSE

2.2.3 Les techniques d'adaptation des égaliseurs

L'adaptation des coefficients des égaliseurs se fait majoritairement par des algorithmes linéaires. Ces algorithmes sont basés sur les propriétés statistiques des signaux en cause. Les algorithmes les plus souvent rencontrés sont le *LMS*, le *RLS* et tous leurs dérivés. L'avantage du *LMS* réside dans sa simplicité de calcul. En effet, lors d'une *ITGE*, cette caractéristique devient primordiale. Par contre, le choix du paramètre d'adaptation est très délicat ce qui limite la plage d'application. Un mauvais choix peut résulter en une divergence de l'algorithme ou en une convergence terriblement lente.

L'algorithme *RLS* est pour sa part assez performant au niveau de la vitesse de convergence et de la stabilité. Cependant, son application aux *ITGE* est limitée car la méthode demande une complexité de calcul assez grande. De plus, la nécessité d'effectuer une division limite beaucoup sa rapidité.

La particularité des deux algorithmes est leur nature récursive. En d'autres termes, la sortie d'un système quelconque que nous définissons $r(n)$ dépend des résultats précédents $r(n\text{-}M)$ où M est un entier positif correspondant à la profondeur de pipeline. Pour des applications à très hauts débits, un problème se pose car la vitesse de calcul est limitée par la longueur de la boucle récursive (valeur de M). Même s'il est possible de rendre parallèle le filtre transverse, il est souvent nécessaire de recourir aux techniques de pipeline à l'intérieur même de l'algorithme pour atteindre de forts débits de calculs. Plus la valeur de M est élevée, plus on pourra raffiner le pipeline de l'architecture ce qui justifie l'utilisation de techniques de transformations d'algorithmes. Ces techniques permettent d'ajuster la valeur de M et ainsi, de varier la profondeur de pipeline. On retrouve dans la littérature des techniques de transformation comme le traitement parallèle, la resynchronisation, le pliage, le déploiement ou l'anticipation (*look-ahead*) [SHA98]. Les détails de ces techniques de pipeline sont présentés au chapitre suivant.

Le Tableau 2.1 donne une synthèse des calculs de chaque méthode *RLS* et *LMS* [VID99]. Dans ces équations, $\tilde{\mathbf{y}}$ et $\mathbf{w}$ représentent respectivement $[\tilde{y}(n)\ \tilde{y}(n-1)\ \tilde{y}(n-2)..\tilde{y}(n-N+1)]^T$ et $[w_1(n)\ w_2(n)\ w_3(n)..w_N(n)]^T$

Tableau 2.1 : Sommaire des méthodes classiques [HAY95]

LMS	
$\hat{x}(n) = \mathbf{w}^T(n-1)\tilde{\mathbf{y}}(n)$	(2.6)
$e(n) = s(n-d) - \hat{x}(n)$	(2.7)
$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu\tilde{\mathbf{y}}(n)e(n)$	(2.8)
RLS	
$\hat{x}(n) = \mathbf{w}^T(n-1)\tilde{\mathbf{y}}(n)$	(2.9)
$P(n) = \frac{1}{\lambda}P(n-1) - \frac{1}{\lambda}K(n)\tilde{\mathbf{y}}^T(n)P(n-1)$	(2.10)
$K(n) = \frac{1}{\lambda} * \frac{P(n-1)\tilde{\mathbf{y}}(n)}{1 + \frac{1}{\lambda}\tilde{\mathbf{y}}^T(n)P(n-1)\tilde{\mathbf{y}}(n)}$	(2.11)
$\mathbf{w}(n) = \mathbf{w}(n-1) + K(n)e(n)$	(2.12)
$e(n) = s(n-d) - \hat{x}(n)$	(2.13)

Chapitre 3

Techniques du pipeline

Dans le traitement numérique des signaux, il existe plusieurs méthodes de calculs. Puisque les signaux sont de nature discrète (quantifiée), nous définissons une base de temps discrète. Nous parlerons alors d'échantillons au temps n où l'instant n représente le temps actuel dans le domaine discret. Un échantillon appartenant au passé est nécessairement un échantillon du temps $(n\text{-}M)$ et un échantillon futur est du temps $(n+M)$ où M est un nombre entier positif.

Les différents algorithmes utilisent alors un ou plusieurs échantillons du domaine n pour calculer la réponse désirée. La séquence d'échantillons utilisés détermine le type d'algorithme en cause. Il existe plusieurs types d'algorithmes selon la séquence de données utilisées. Les algorithmes non récursifs ne sont fonction que des échantillons entrant dans l'architecture de calcul. Par contre, un algorithme est dit récursif si le calcul de la sortie nécessite les résultats antérieurs de la même opération. En d'autres termes, une sortie que nous définissons $r(n)$ est fonction des résultats précédents $r(n\text{-}M)$. Dans [SHA98], il est

montré que de nos jours, les différents types d'algorithmes (récursifs ou non) doivent être utilisés dans des applications ou le débit de l'information est très rapide. Pour ce faire, on doit avoir recours aux techniques de pipeline.

Le principe du pipeline est simple. Il consiste à ajouter des registres entre les étapes de calculs qui permettront soit d'augmenter la rapidité d'exécution ou de diminuer la consommation de puissance (en diminuant la fréquence d'horloge). La profondeur de pipeline (ou le raffinage) est caractérisée par le nombre de registres présents entre les différents étages de calculs . Dans le cas des algorithmes récursifs, cette profondeur est limitée par la valeur de M. Dans le cas où M est faible ou unitaire, il est nécessaire de transformer l'algorithme pour atteindre une profondeur de pipeline satisfaisante.

3.1 Méthode de resynchronisation

La méthode de resynchronisation ("*retiming*") n'est pas qualifiée de technique de pipeline proprement dite. Cette technique est appliquée pour redistribuer les registres inhérents de l'architecture ou les registres que le pipeline a introduit à travers l'architecture. Le principe est de transférer les registres d'un arc à l'autre sans altérer la séquence de sortie. Il est bon de noter que cette technique s'applique à toutes les architectures [SHA98].

Voyons un exemple d'application de la technique avec le filtre à réponse impulsionnelle finie (*FIR*: *Finite Impulse Response*) de la Figure 3.1 (les coefficients de multiplication sont inclus dans les multiplieurs):

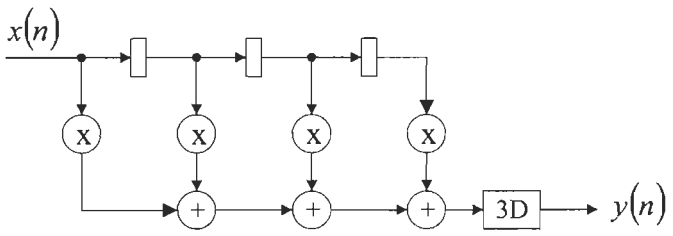


Figure 3.1: FIR avant resynchronisation

▯	Registre - délai de 1 cycle d'horloge (z^{-1})
xD	x Registres - délai de x cycles d'horloge (z^{-x})
(x)	Multiplieur
(+)	Additionneur

La vitesse maximale que ce filtre peut atteindre est délimitée par le chemin critique de calcul. Dans ce cas, le chemin critique avant que la sortie $y(n)$ soit stable est donné par:

$$\tau_{crit} = \tau_{mult} + 3\tau_{add}$$

où

τ_{mult} = Temps d'un multiplieur (3.1)

τ_{addt} = Temps d'un additionneur

Pour diminuer le chemin critique, on applique la resynchronisation sur les 3 registres (*3D*) présents à la sortie de l'architecture. Le principe est de redistribuer les registres de la branche sortante d'une opération (addition, multiplication, etc.) vers les branches entrantes. Effectuons la première passe qui consiste à redistribuer deux registres vers l'entrée du dernier additionneur. Un registre est gardé à sa sortie pour la synchronisation du système. La Figure 3.2 montre le résultat.

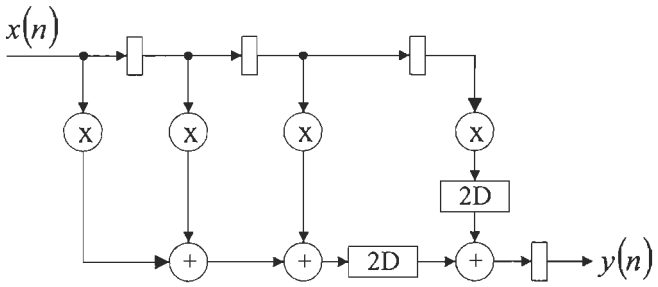


Figure 3.2: Resynchronisation du FIR étape 1

On voit qu'après la première étape, le chemin critique de la dernière branche est maintenant:

$$\tau_{crit} = \max\{\tau_{mult}, \tau_{add}\} \tag{3.2}$$

Par contre, le délai de l'architecture complète demeure grand et correspond maintenant à:

$$\tau_{crit} = \tau_{mult} + 2\tau_{add} \tag{3.3}$$

Pour diminuer ce délai, on applique une deuxième et une troisième passe de resynchronisation pour enfin arriver à l'architecture de la Figure 3.3.

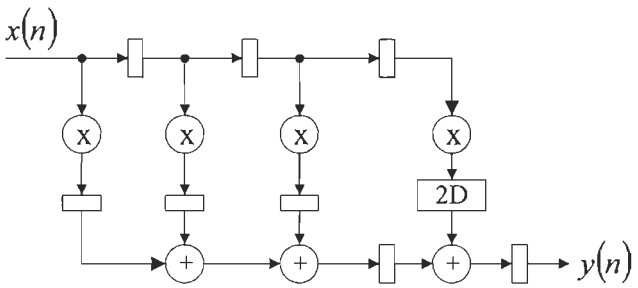


Figure 3.3: FIR après resynchronisation complète

Le chemin critique du filtre complet est alors donné par l'éq (3.4):

$$\tau_{crit} = \max\{\tau_{mult}, 2\tau_{add}\} \tag{3.4}$$

Dans le cas où le délai de deux additionneurs est plus faible que la somme du délai d'un additionneur et d'un multiplieur, il y a un gain au niveau de la vitesse maximale de calcul. De plus, on remarque que l'application de la resynchronisation n'altère pas la latence du filtre qui est de 3 cycles d'horloge dans ce cas.

3.2 Pipeline classique

Dans l'exemple précédent, la présence des 3 registres à la sortie du filtre nous permet d'appliquer la resynchronisation pour améliorer le chemin critique de l'architecture. Souvent, ces registres sont inexistants et pour pallier cet inconvénient, on applique la technique de pipeline classique [SHA98]. Notons que le pipeline classique s'applique dans le cas des algorithmes non récursifs. Il est donc possible d'ajouter des registres dans l'architecture qui pourront être redistribués par la resynchronisation. Les principes du pipeline reposent sur deux définitions que l'on retrouve dans [SHA98].

1. Une coupe dans une architecture correspond en une série de raccords qui, une fois séparées, donnent deux structures distinctes.
2. Une coupe unidirectionnelle est caractérisée par des entrées appartenant à une même coupe et des sorties appartenant à une coupe distincte.

La technique consiste à introduire M registres dans chaque élément entrant d'une même coupe. Nous verrons ici un exemple de pipeline pour mieux comprendre la méthode. Pour la démonstration, on suppose un temps d'addition τ_{add} de 10 et un temps de multiplication τ_{mult} de 20. Considérons l'architecture *FIR* de la Figure 3.4.

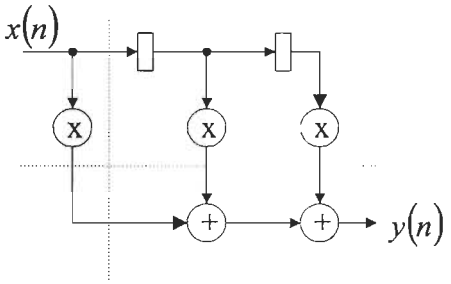


Figure 3.4: Structure FIR avant pipeline

Dans cette figure, les traits pointillés dénotent les coupes que l'on peut effectuer sur l'architecture selon les 2 définitions précédentes. Si l'on introduit un registre dans chaque coupe verticale, on obtient l'architecture de la Figure 3.5.

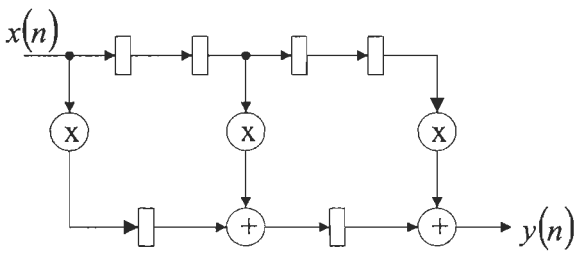


Figure 3.5: Structure FIR pipelinée #1

Dans cette architecture, on remarque que le temps critique est de 30 comparativement à 40 pour l'architecture originale. Il y a donc un gain en vitesse intéressant. Si on applique le pipeline sur la coupe horizontale, on obtient l'architecture de la Figure 3.6.

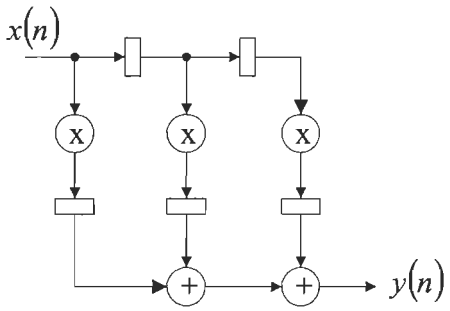


Figure 3.6: Structure FIR pipelinée #2

Dans ce cas, le chemin critique est de 20 ce qui représente la moitié du chemin critique original. On voit alors que l'application du pipeline s'avère très efficace. Par contre, l'ajout des registres implique une augmentation de la surface utilisée. Cette contrainte peut devenir importante lorsque l'ordre du filtre est très élevé.

L'ajout des registres ne se limite pas à un seul par branche. En effet, on peut ajouter M registres qui pourront être redistribués par la resynchronisation sans aucun problème. Les désavantages se situent au niveau de la surface utilisée et de la latence qui sera augmentée en fonction de M. Cette latence est souvent bien acceptée étant donné l'apport du pipeline sur la vitesse de l'architecture.

Dans le cas où l'algorithme est du type récursif, la technique classique de pipeline ne s'applique plus car la définition 2 n'est plus respectée. Si on désire pipeliner un algorithme récursif, il faut avoir recours à une technique différente que nous présentons à la section 3.3.

3.3 Technique de l'anticipation

La boucle de retour dans l'algorithme récursif limite souvent la profondeur de pipeline que l'on peut y appliquer. Par exemple, si la sortie $r(n)$ nécessite la sortie précédente $r(n-1)$, il sera impossible de pipeliner l'architecture. Pour pipeliner, on doit développer la série des équations de l'algorithme sur une longueur M.

3.3.1 Principe

La technique de l'anticipation permet d'utiliser la sortie $r(n-M)$ au lieu de la sortie $r(n-1)$ (où M est entier positif) pour le calcul de $r(n)$ ce qui permet de pipeliner l'architecture à une profondeur M. Le choix de M dépend de la profondeur de pipeline désirée. Il est important de tenir compte de l'implantation matérielle car un M trop grand peut résulter en une surface d'utilisation très grande.

Voyons maintenant les détails de la méthode. Soit l'éq. (3.5) qui représente une équation récursive de premier ordre.

$$x(n) = ax(n-1) + u(n) \tag{3.5}$$

Dans l'éq. (3.5), $x(n)$ représente la sortie du système et $u(n)$ représente l'entrée du système. On remarque la nature récursive car la valeur actuelle de $x(n)$ nécessite la valeur précédente $x(n-1)$ que l'on multiplie par un coefficient a. La borne inférieure du chemin critique est donnée par le délai d'une addition et d'une multiplication. Si on utilise les

valeurs de l'exemple précédent, le chemin critique serait de 30. Pour franchir cette limite inférieure, on doit appliquer le pipeline sur l'éq (3.5).

Pour pipeliner l'algorithme, il est nécessaire de développer les calculs de l'équation soit:

$$\begin{aligned} x(n) &= ax(n-1)+u(n) \\ x(n-1) &= ax(n-2)+u(n-1) \\ x(n) &= a[ax(n-2)+u(n-1)]+u(n) \\ x(n) &= a^2 x(n-2)+au(n-1)+u(n) \end{aligned} \tag{3.6}$$

Si on développe l'éq. (3.6) sur une longueur M, on obtient le résultat de l'éq. (3.7).

$$x(n) = a^M x(n-M)+\sum_{i=0}^{M-1} a^i u(n-i) \tag{3.7}$$

L'éq. (3.7) est très importante car elle représente la technique de base pour la majorité des transformations d'algorithmes récursifs. On remarque que la technique introduit M registres dans la boucle de récursivité ce qui permet de pipeliner l'opération de multiplication et d'addition au niveau désiré. La borne inférieure du chemin critique est alors franchie. Si la redistribution des registres est appliquée de façon uniforme sur l'addition et la multiplication, on obtient un gain en vitesse de M fois la vitesse originale [SHA98].

On remarque aussi que l'algorithme préserve la structure des données d'entrée et de sortie. C'est à dire que les performances de l'algorithme au niveau de l'erreur de reconstitution ne seront pas modifiées. Par contre, cette conservation de la performance se fait au détriment de la surface d'intégration. En effet, le calcul de l'anticipation (sommation

dans l'éq. (3.7)) est fonction de $M*N$ (N représente l'ordre du filtre) ce qui ajoute beaucoup d'éléments lorsque M et N sont élevés. Une autre observation importante concerne la nature non récursive de la boucle d'anticipation. Cette propriété permet d'appliquer la technique de pipeline classique afin d'augmenter son débit. On retrouve l'application de la technique de l'anticipation dans [HAT92] où un filtre à réponse impulsionnelle infinie (*IIR - Infinite Impulse Response*) fonctionnant à 85 *MHz* a été réalisé en technologie *VLSI*.

3.3.2 Anticipation dispersée et regroupée

Dans le cas ou la récursivité de l'algorithme est d'ordre supérieur à 1, on a recours à deux types de transformation: l'anticipation dispersée (*Scattered*), ou l'anticipation regroupée (*Clustered*). On retrouve ce genre d'algorithmes dans les filtres numériques *IIR*. Le calcul sériel de ce genre de filtre est donné par l'éq. (3.8).

$$x(n) = f_{seriel}(x(n-1), x(n-2), \ldots, x(n-N), u(n), u(n-1), \ldots, u(n-P)) \tag{3.8}$$

Dans cette équation, $x(n)$ représente la sortie du filtre, $u(n)$ représente l'entrée du filtre, N représente l'ordre du filtre et $f_{seriel}(\bullet)$ représente une fonction linéaire. L'état présent d'un tel filtre est donc fonction des états passés et des valeurs présentes et passées de l'entrée. L'application de l'anticipation regroupée sur ce type d'équation pour obtenir un pipeline d'ordre M est donnée par (3.9).

$$x(n) = f_{c,pipe}\begin{pmatrix} x(n-M), x(n-M-1), \ldots, x(n-M-N+1), \\ u(n), u(n-1), \ldots, u(n-M+1) \end{pmatrix} \tag{3.9}$$

Le calcul de $x(n)$ se fait en fonction d'un groupe de N états qui sont M échantillons dans le passé. Les détails de la fonction $f_{c,pipe}(\bullet)$ sont donnés dans [PAR89]. L'ajout matériel d'un tel type de transformation est proportionnel à M. L'anticipation dispersée est pour sa part définie par (3.10).

$$x(n) = f_{s,pipe}(x(n-M), x(n-2M), \ldots, x(n-MN), u(n), u(n-1), \ldots, u(n-NM-2)) \qquad (3.10)$$

Les détails de la fonction linéaire $f_{s,pipe}(\bullet)$ sont donnés dans [PAR89]. L'ajout matériel de (3.10) est fonction de NM et peut être réduit à $N\log_2(M)$ par décomposition [PAR89]. L'anticipation dispersée donne une architecture plus gourmande en surface que l'architecture obtenue par l'anticipation regroupée. Par contre, l'anticipation dispersée possède l'avantage de préserver la stabilité ce que l'anticipation regroupée ne peut garantir [PAR89].

3.4 Transformation de la technique de l'anticipation

3.4.1 Justification de la transformation

Comme nous venons de le voir, les différentes techniques d'anticipation permettent d'atteindre de forts débits de calcul mais il en résulte des architectures qui utilisent beaucoup de surface lorsque le niveau de pipeline M et l'ordre N du filtre sont élevés. Il est donc intéressant de développer des techniques qui minimisent cet ajout de matériel. Le principe est de transformer la technique de l'anticipation dans le but de conserver de bonnes propriétés algorithmiques et limiter l'ajout matériel. Cette transformation donne lieu à des

techniques d'approximations plutôt que des techniques de transformations car elles ne conservent pas l'exactitude de l'algorithme original. Une technique très intéressante est l'anticipation relaxée que nous présentons dans la sous-section 3.4.2.

3.4.2 Anticipation relaxée

La technique de l'anticipation relaxée permet d'atteindre de forts débits de calculs sans toutefois augmenter de façon dramatique la surface d'intégration. La technique s'applique essentiellement en deux phases:

1. Application d'une technique d'anticipation vue à la section 3.3 .
2. Tronquer la fonctionnalité de certains blocs de l'algorithme pipeliné de telle sorte que l'impact sur le comportement de la convergence soit minimal.

Alors que la phase 1 permet de créer une architecture unique, la phase 2 permet d'appliquer une vaste étendue d'approximations ce qui permet de créer une multitude d'architectures distinctes. Par contre, puisque l'étape 2 altère les caractéristiques de l'algorithme, il est important d'effectuer une analyse de stabilité sur l'architecture choisie. Pour expliquer le principe de l'anticipation relaxée, appliquons la technique sur une équation récursive du premier ordre qui varie dans le temps. Soit l'équation (3.11).

$$x(n+1) = a(n)x(n) + b(n)u(n) \tag{3.11}$$

Selon l'étape 1, on applique la technique de l'anticipation ce qui donne (3.12)

$$x(n+M)=\left[\prod_{i=0}^{M-1}a(n+i)\right]x(n)+$$
$$\sum_{i=0}^{M-1}\left\{\left[\prod_{j=1}^{i}a(n-M-j)\right]\cdot b(n+M-1-i)u(n+M-1-i)\right\} \qquad (3.12)$$

On remarque ici que la complexité de (3.12) est beaucoup plus importante que dans (3.11). Cette complexité est causée par la préservation de l'exactitude de l'algorithme sériel. On peut alors modifier ou relaxer l'équation (3.12) par différentes techniques au dépend d'une légère dégradation du comportement de la convergence. Les trois principaux types de relaxations sont: la relaxation de somme, la relaxation du produit et la relaxation du délai [SHA98].

Pour appliquer la relaxation, il est nécessaire de connaître le comportement des variables du système. La qualité des relaxations dépend de la qualité des hypothèses. Par exemple, si on considère que la valeur de $a(n)$ est voisine de 1 et que le produit $b(n)u(n)$ varie lentement sur une séquence de M termes, on peut relaxer le terme de sommation de (3.12) pour obtenir (3.13).

$$x(n+M)=\left[\prod_{i=0}^{M-1}a(n+i)\right]x(n)+Mb(n+M-1)u(n+M-1) \qquad (3.13)$$

Le résultat obtenu en (3.13) est appelé la relaxation de somme. Si la valeur de $a(n)$ est voisine de 1 et qu'elle varie lentement sur M cycles, on peut appliquer la relaxation du produit sur le premier terme de (3.12), ce qui donne (3.14).

$$x(n+M)=\left[1-M\left(1-a(n+M-1)\right)\right]x(n)+Mb(n+M-1)u(n+M-1) \qquad (3.14)$$

Enfin, si on considère que le produit $b(n)u(n)$ varie lentement sur M échantillons, on peut appliquer la relaxation du délai ce qui donne (3.15).

$$x(n+M)=\left[1-M\left(1-a(n+M-1)\right)\right]x(n)+Mb(n+M-D_1)u(n+M-D_1) \qquad (3.15)$$

On voit ici que la relaxation peut prendre différentes formes que l'on applique séparément ou en conjonction. Même si la technique de l'anticipation relaxée donne de très bons compromis sur l'implantation des algorithmes récursifs, nous retrouvons d'autres techniques que nous présentons à la section 3.4.3.

3.4.3 Autres méthodes

On retrouve dans la littérature quelques méthodes de transformation de la technique de l'anticipation qui sont différentes de l'anticipation relaxée. Il est bon de noter que ces techniques sont très souvent des dérivées ou des cas particuliers de la technique de l'anticipation relaxée.

La première technique souvent rencontrée s'appelle *Delayed LMS algorithm* (*DLMS*) [LON89]. Cette technique s'applique à l'algorithme *LMS* et elle représente un cas particulier de l'anticipation relaxée. L'application de la technique sur (2.7) et (2.8) donne le résultat des équations (3.16) et (3.17).

$$e(n-D) = s(n-d-D) - \hat{x}(n-D) \tag{3.16}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\tilde{\mathbf{y}}(n-D)e(n-D) \tag{3.17}$$

On voit ici qu'il s'agit d'une relaxation du délai et de la somme suite à l'application de l'anticipation sur les équations originales. Le grand désavantage du *DLMS* est sa lenteur de convergence. Pour améliorer la vitesse de convergence, [MAT99] présente une transformation de l'algorithme *DLMS* qui s'avère efficace. L'architecture utilise par contre une grande surface ce qui limite son champ d'application.

On présente aussi un autre type d'anticipation dans [SHA99]. La technique développée est nommée l'anticipation distribuée. Son principe repose sur l'utilisation des échantillons passés distribués de façon non uniforme. Il est alors possible de préserver la stabilité sans augmenter la surface d'utilisation. Cette technique s'applique surtout sur des algorithmes d'ordre supérieur à 1.

3.5 Application à l'égalisation adaptative des canaux

L'application de la technique de l'anticipation relaxée se prête bien à la problématique d'égalisation des canaux. En effet, pour que la réalisation matérielle soit intéressante, l'architecture ne doit pas utiliser trop de surface et fonctionner à des débits les plus élevés possible. Quelques applications reliées à l'égalisation des canaux utilisant la technique de l'anticipation relaxée ont été publiées. On retrouve d'ailleurs dans [SHA95] le cas de

réalisation d'un égaliseur *DFE* adapté par l'algorithme *LMS*. L'architecture pipelinée fonctionne à une vitesse 48 fois supérieure à celle de l'architecture sérielle.

Nous allons maintenant montrer une application de la méthode de l'anticipation relaxée sur l'architecture d'un filtre transverse linéaire adapté par l'algorithme *LMS*. Le schéma de la Figure 2.15 montre le contexte d'utilisation. Les équations du *LMS* sériel sont données par les équations (3.18) et (3.19)

$$e(n) = s(n-d) - \mathbf{w}^T \tilde{\mathbf{y}}(n) \tag{3.18}$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu \tilde{\mathbf{y}}(n) e(n) \tag{3.19}$$

La sortie du canal $\tilde{\mathbf{y}}(n)$ est définie par (3.20) (N représente le nombre d'entrées de l'égaliseur)

$$\tilde{\mathbf{y}}(n) = \left[\tilde{y}(n)\ \tilde{y}(n-1)\ \tilde{y}(n-2)..\tilde{y}(n-N+1)\right]^T \tag{3.20}$$

L'architecture de l'algorithme est donnée à la Figure 3.7. Dans cette architecture le chemin critique est donné par .(3.21)

$$\tau_{crit} = 3\tau_{mult} + (N+1)\tau_{add} \tag{3.21}$$

Pour atteindre de forts débits, on doit réduire τ_{crit}. Pour ce faire, on applique en premier lieu la technique de l'anticipation. Par la suite, nous allons limiter l'ajout matériel en appliquant la relaxation de somme. Enfin, la technique de la resynchronisation est utilisée pour distribuer les registres à travers l'architecture de calcul.

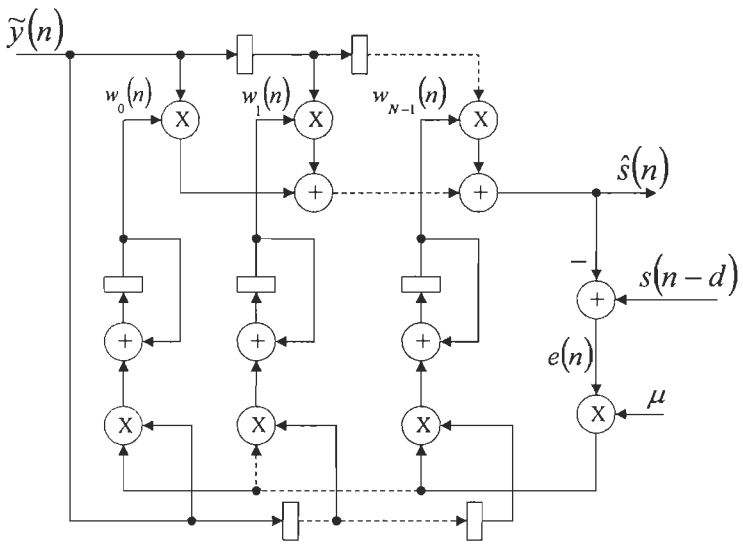


Figure 3.7 Architecture sérielle du filtre transverse

L'application de la technique de l'anticipation sur l'algorithme de mise à jour des poids donne (3.22)

$$\mathbf{w}(n) = \mathbf{w}(n - D_2) + \mu \sum_{i=0}^{D_2-1} \tilde{\mathbf{y}}(n-i) e(n-i) \tag{3.22}$$

On applique ensuite la relaxation du délai sur (3.22) ce qui donne (3.23).

$$\mathbf{w}(n) = \mathbf{w}(n - D_2) + \mu \sum_{i=0}^{D_2-1} \tilde{\mathbf{y}}(n - D_1 - i) e(n - D_1 - i) \tag{3.23}$$

Enfin, on applique la relaxation de la somme en limitant la longueur de la somme à *LA* ($LA \leq D_2$). On obtient alors l'algorithme d'adaptation des poids tel que montré en (3.24)

$$\mathbf{w}(n) = \mathbf{w}(n - D_2) + \frac{\mu D_2}{LA} \sum_{i=0}^{LA-1} \tilde{\mathbf{y}}(n - D_1 - i) e(n - D_1 - i) \tag{3.24}$$

Le calcul de l'erreur se fait par l'équation (3.25).

$$\begin{aligned} e(n) &= s(n-d) - \mathbf{w}^T(n-1)\tilde{\mathbf{y}}(n) \\ e(n) &= s(n-d) - \\ &\left[\mathbf{w}(n-D_2-1) + \frac{\mu D_2}{LA}\sum_{i=0}^{LA-1} e(n-D_1-i-1)\tilde{\mathbf{y}}(n-D_1-i-1)\right]^T \tilde{\mathbf{y}}(n) \end{aligned} \tag{3.25}$$

Or, comme la valeur de μ est faible et en remplaçant $\mathbf{w}(n-D_2-1)$ par $\mathbf{w}(n-D_2)$, nous obtenons (3.26).

$$e(n) = s(n-d) - \mathbf{w}^T(n-D_2)\tilde{\mathbf{y}}(n) \tag{3.26}$$

L'architecture de l'algorithme est donné à la Figure 3.8. On remarque la position et l'importance de D_1 qui permet de pipeliner l'étage du calcul de l'erreur. On voit aussi que D_2 permet de pipeliner l'étage de mise à jour des poids. Enfin, on voit que si LA est grand, la surface d'utilisation augmente considérablement. Cette valeur sera donc mise à la valeur minimum acceptable qui sera déterminée par simulation dans le but de satisfaire les spécifications de l'application visée.

Le chemin critique de cette architecture dépend des valeurs de D_1, D_2 et de la façon dont la resynchronisation est appliqué. Prenons l'exemple suivant où l'ordre du filtre (N) est 5, $\tau_{mult} = 40$ et $\tau_{add} = 20$. Le chemin critique est alors de 240 (selon (3.21)). Pour atteindre un gain de vitesse de 48, on a besoin d'un chemin critique de 5. Or ce chemin critique peut être atteint si chaque multiplieur est pipeliné par 8 registres et si chaque additionneur est

pipeliné par 4 registres. On peut alors, par inspection de l'architecture sérielle, fixer D_2 à 4 et D_1 à 44 pour la resynchronisation vers les multiplieurs.

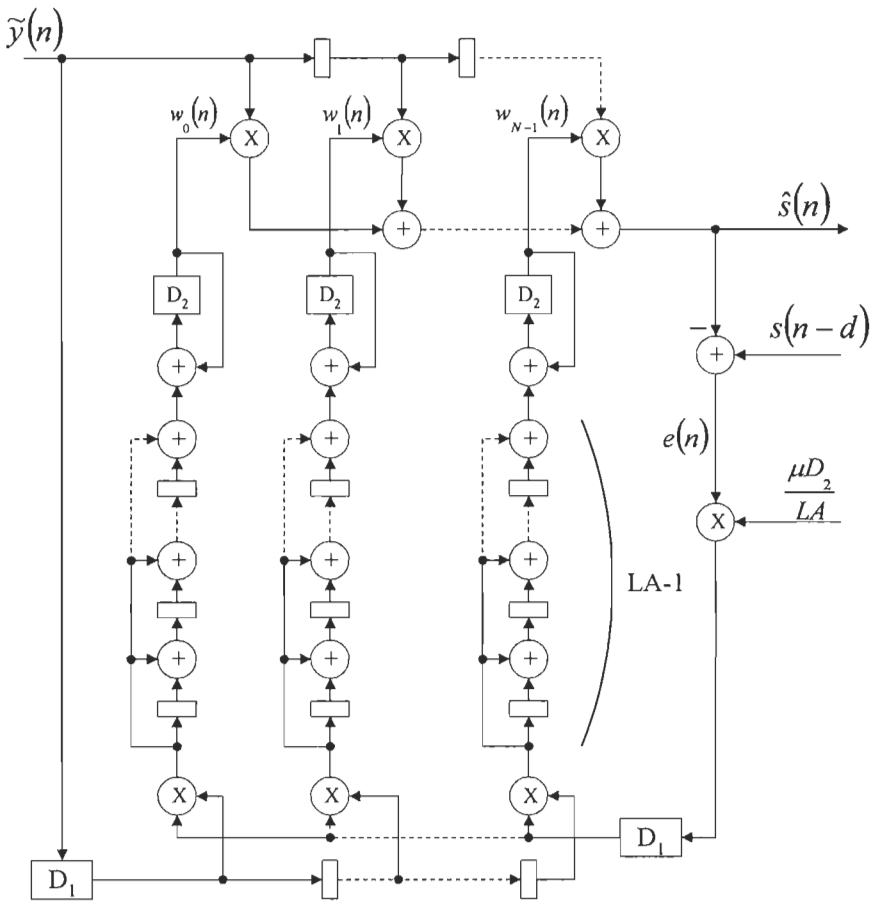


Figure 3.8: Architecture pipelinée du filtre transverse

La Figure 3.9 nous montre le résultat final de l'architecture après la resynchronisation [SHA93]. A titre comparatif, si on avait utilisé le *DLMS*, le chemin critique minimum aurait été de 20 ce qui limite beaucoup l'apport en vitesse [SHA93]. Enfin, si on désire obtenir une meilleure performance dans la convergence, on peut recourir à l'ajustement de LA qui est à 1 dans ce cas.

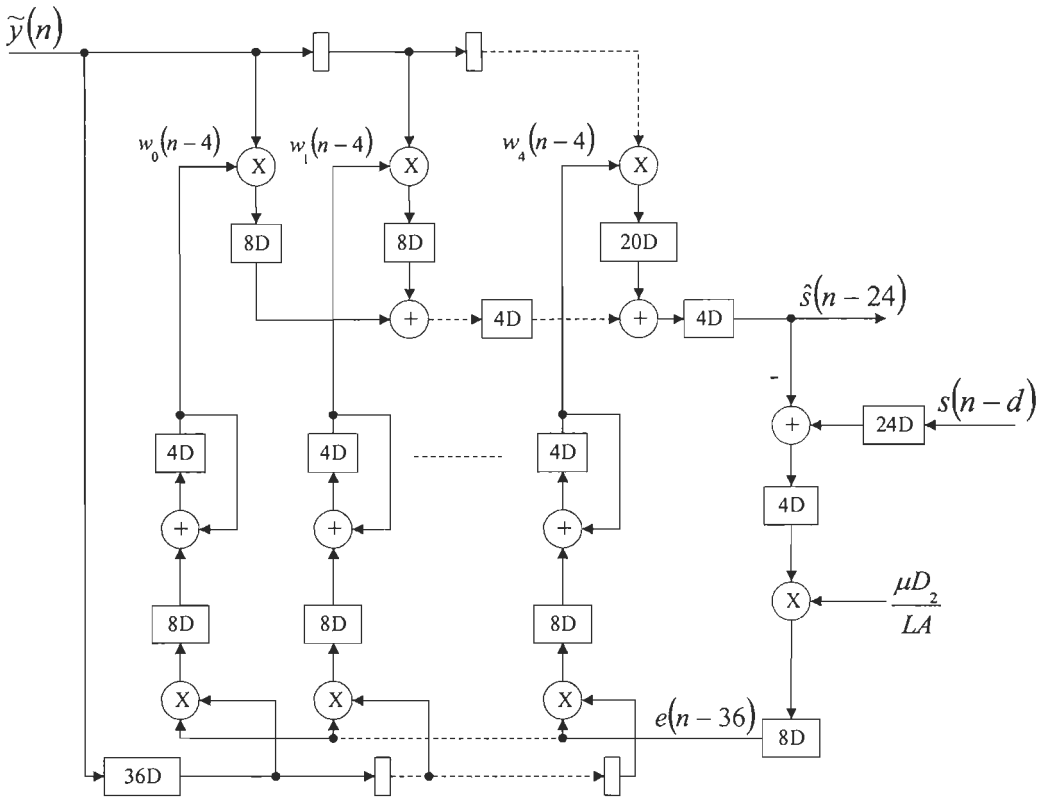


Figure 3.9: Filtre transverse pipeliné avec gain en vitesse de 48

3.6 Conclusion

Dans ce chapitre, nous avons présenté les différentes méthodes utilisées pour pipeliner les algorithmes de type récursifs et non récursifs. Nous avons vu que la méthode de la resynchronisation permet de distribuer les registres introduits par le pipeline à travers la structure de calcul dans le but d'augmenter son débit.

Il a été montré que le pipeline des algorithmes récursifs peut s'avérer assez complexe. Les méthodes de l'anticipation et ses dérivées donnent une multitude d'architectures aux caractéristiques différentes. La technique de l'anticipation relaxée propose un bon compromis entre la surface d'utilisation et le gain en vitesse obtenu mais les

caractéristiques de convergence et de stabilité sont altérées. D'autres méthodes pour pipeliner les algorithmes récursifs ont été présentées et leurs caractéristiques ont été énumérées.

Nous avons aussi présenté un exemple de la technique de l'anticipation relaxée. La méthode a été appliquée sur un filtre transverse adapté par *LMS* dans le cadre de l'égalisation des canaux. Enfin, il existe dans la littérature d'autres applications utilisant la technique de l'anticipation relaxée. On retrouve entre autre un encodeur vidéo de type *ADPCM* (*Adaptative Differential Pulse Code Modulation*) fonctionnant à 100*MHz* [SHA93] et un système de réseau local asynchrone (*ATM LAN*) opérant à un débit de 51.84*Mbps* [SHA98a].

Chapitre 4

Pipeline d'égaliseurs à base de réseaux de neurones

Dans ce chapitre, nous verrons en premier lieu l'application des réseaux de neurones artificiels (*RNA*) à l'égalisation des canaux. Nous verrons ensuite les principes de base entourant les *RNA* de type perceptron multicouches. Nous aborderons ensuite l'algorithme d'apprentissage par la rétropropagation du gradient et nous verrons comment le transformer par la technique de l'anticipation relaxée pour en arriver à un algorithme pipeliné. Enfin, les résultats des simulations seront présentés pour vérifier l'effet du pipeline sur les performances de convergence de l'algorithme de rétropropagation du gradient et sur les performances en égalisation du *RNA*.

4.1 Réseaux de neurones pour l'égalisation des canaux

L'utilisation des *RNA* à titre d'égaliseurs a pour principal avantage de pouvoir traiter des canaux non linéaires. Par contre, l'implantation matérielle d'un *RNA* demande une

bonne quantité de calcul et son utilisation pour de très hauts débits devient difficilement réalisable. Grâce à la technique de l'anticipation relaxée, nous verrons qu'il est possible de transformer l'algorithme de rétropropagation du gradient et d'obtenir un algorithme pipeliné qui pourra fonctionner à un débit très élevé.

Des travaux sur le pipeline d'égaliseurs à base de *RNA* ont été réalisés dans [VID99]. L'auteur n'a pas utilisé de transformations d'algorithme mais a plutôt choisi une approche par architecture systolique permettant d'effectuer les calculs du réseau par séquences (mise à jour des poids différée) et ainsi appliquer un pipeline direct dans les éléments de calculs.

Le contexte d'utilisation d'un *RNA* à titre d'égaliseur est donné à la Figure 4.1. Il est bon de noter que l'adaptation du *RNA* est faite de façon supervisée par l'algorithme de la rétropropagation du gradient. L'entrée du réseau correspond à un vecteur de données $\tilde{y}(n)$ décalées dans le temps par des registres. Cette configuration permet d'éliminer l'interférence inter-symboles (*ISI: Intersymbol Interference*) présente dans $\tilde{y}(n)$. Le réseau s'adapte donc dans le but de retrouver les valeurs originales exactes $s(n)$. Le calcul de l'erreur $e(n)$ est ici effectué avant la décision sur les symboles permettant ainsi une adaptation plus raffinée du réseau.

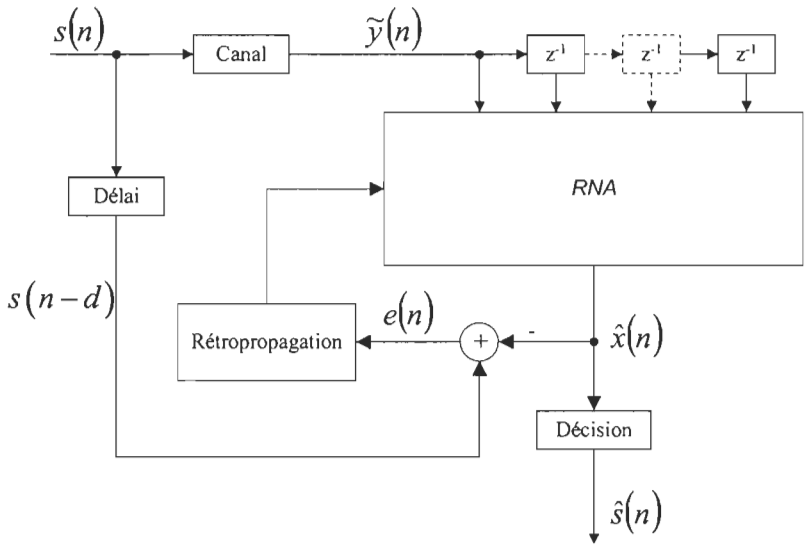


Figure 4.1: *RNA* pour l'égalisation des canaux

4.2 Les réseaux de neurones perceptron multicouches

Le schéma de la Figure 4.2 nous montre la topologie générale d'un réseau de neurones multicouches entièrement connecté. Chaque cercle dans le schéma représente un neurone. Dans ce schéma, nous avons les notations suivantes:

- l = indice de la couche du réseau
- L = nombre de couches du réseau.
- N^l = nombre de neurones sur la couche l

Le détail d'un neurone est donné par la Figure 4.3. La fonction f est appelée la fonction d'activation et est généralement de nature non linéaire.

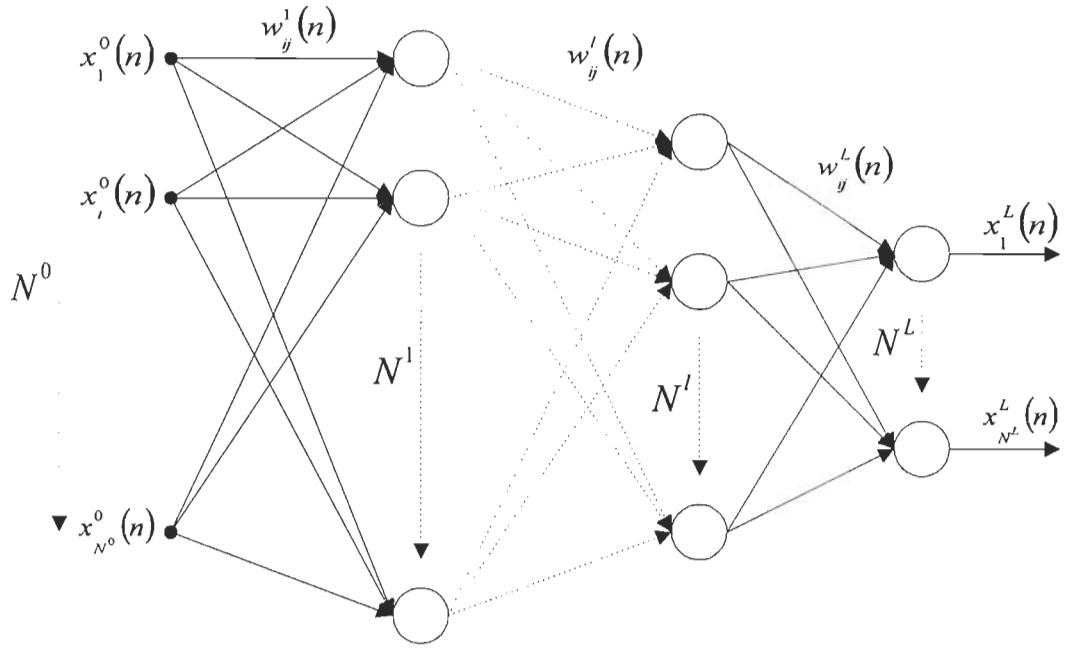


Figure 4.2: Réseau de neurones multicouches

Pour pouvoir appliquer l'algorithme de rétropropagation du gradient, la fonction d'activation doit être dérivable en tout point [HER94]. Dans la littérature, on retrouve principalement des fonctions du type tangente hyperbolique, sigmoïde, linéaire ou fonction saturation (dérivable par partie) à titre de fonction d'activation [HER94].

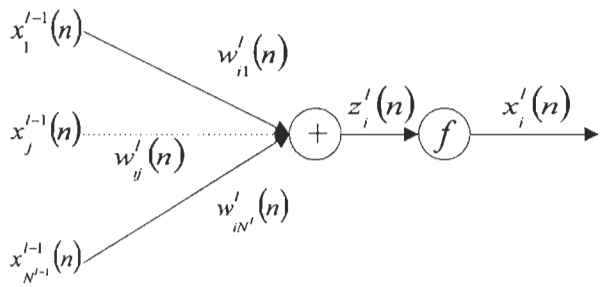


Figure 4.3: Détail d'un neurone

L'algorithme de rétropropagation du gradient pour la mise à jour des poids est donné par les équations (4.1) à (4.5) [HER94] [VID99].

$$z_i^l(n) = \sum_{j=1}^{N^{l-1}} w_{ij}^l(n-1)x_j^{l-1}(n) \tag{4.1}$$

$$x_i^l(n) = f\left(z_i^l(n)\right) \tag{4.2}$$

$$e_i^L(n) = \left[s_i(n-d) - x_i^L(n)\right] \tag{4.3}$$

$$e_i^l(n) = f'\left(z_i^l(n)\right)\sum_{j=1}^{N^{l-1}} w_{ji}^{l+1}(n-1)e_j^{l+1}(n) \tag{4.4}$$

$$\mathbf{w}_i^l(n) = \mathbf{w}_i^l(n-1) + \mu e_i^l(n)x_i^{l-1}(n) \tag{4.5}$$

$$l = 2,....,L-1$$

$$i = 1,2,....,N^l$$

où f est la fonction d'activation et f' est sa dérivée.

4.3 Anticipation relaxée sur la rétropropagation du gradient

On remarque dans l'équation (4.5) de l'algorithme que la mise à jour des poids est de nature récursive. Nous pouvons donc appliquer la technique de l'anticipation relaxée sur l'algorithme de rétropropagation. La relaxation du délai (D_1) et de la somme (D_2) sont ici utilisées. Le résultat est donné par les équations (4.6) à (4.10).

$$z_i^l(n) = \sum_{j=1}^{N^{l-1}} w_{ij}^l\left(n - D_2\right)x_j^{l-1}(n) \tag{4.6}$$

$$x_i^l(n) = f\left(z_i^l(n)\right) \tag{4.7}$$

$$e_i^L(n) = \left[s_i(n-d) - x_i^L(n)\right] \tag{4.8}$$

$$e_i^l(n) = f'\left(z_i^l(n)\right)\sum_{j=1}^{N^{l-1}} w_{ji}^{l+1}\left(n - D_2\right)e_j^{l+1}(n) \tag{4.9}$$

$$\mathbf{w}_i^l(n) = \mathbf{w}_i^l\left(n - D_2\right) + \frac{\mu}{LA}\sum_{k=0}^{LA-1} e_i^l\left(n - D_1 - k\right)x_i^{l-1}\left(n - D_1 - k\right) \tag{4.10}$$

$$i = 1,2,....,N^l$$

$$l = 2,....,L-1$$

$$1 \le D_2$$

$$1 \le LA \le D_2$$

$$0 \le D_1$$

Nous pouvons réécrire les équations sous une forme matricielle, simplifiant ainsi la notation. Le résultat est présenté par les équations (4.11) à (4.15).

$$\mathbf{z}^l(n) = \mathbf{W}^l\left(n - D_2\right)\mathbf{x}^{l-1}(n) \tag{4.11}$$

$$\mathbf{x}^l(n) = f\left(\mathbf{z}^l(n)\right) \tag{4.12}$$

$$\mathbf{e}^L(n) = \left[\mathbf{s}(n-d) - \mathbf{x}^L(n)\right] \tag{4.13}$$

$$\mathbf{e}^l(n) = f'\left(\mathbf{z}^l(n)\right)\cdot\left(\mathbf{W}^{l+1}\left(n - D_2\right)\right)^T\mathbf{e}^{l+1}(n) \tag{4.14}$$

$$\mathbf{W}^l(n) = \mathbf{W}^l\left(n - D_2\right) + \frac{\mu}{LA}\sum_{k=0}^{LA-1}\mathbf{e}^l\left(n - D_1 - k\right)\left(\mathbf{x}^{l-1}\left(n - D_1 - k\right)\right)^T \tag{4.15}$$

Les dimensions des vecteurs et des matrices en causes sont donnés par:

$$\dim\left[\mathbf{W}^l\right] = \left[N^l \times N^{l-1}\right]$$

$$\dim\left[\mathbf{z}^l\right] = \left[N^l \times 1\right]$$

$$\dim\left[\mathbf{x}^l\right] = \left[N^l \times 1\right]$$

$$\dim\left[\mathbf{e}^l\right] = \left[N^l \times 1\right]$$

Les équations précédentes représentent le cas général avec N^0 entrées, N^L sorties et L couches. Pour l'application à l'égalisation des canaux, il a été montré dans [DAH00] et [VID99] que l'utilisation d'une seule couche cachée s'avère suffisante. De plus, nous n'avons besoin que d'une seule sortie car nous traitons le cas de données binaires. Les équations (4.11) à (4.15) peuvent donc se simplifier pour en arriver aux équations (4.16) à (4.22). À noter que le changement de notation pour certaines variables permet d'éviter la confusion. Les notations suivantes seront dorénavant retenues pour le reste du rapport.

$$\mathbf{z}(n) = \mathbf{W}\left(n - D_2\right)\tilde{\mathbf{y}}(n) \tag{4.16}$$

$$\hat{\mathbf{z}}(n) = f\left(\mathbf{z}(n)\right) \tag{4.17}$$

$$x(n) = \mathbf{q}\hat{\mathbf{z}}(n) \tag{4.18}$$

$$\hat{x}(n) = f(x(n)) \tag{4.19}$$

$$e(n) = [s(n-d) - \hat{x}(n)] \tag{4.20}$$

$$\mathbf{q}(n) = \mathbf{q}(n - D_2) + \frac{\mu}{LA} \sum_{k=0}^{LA-1} e(n - D_1 - k)\left(\hat{\mathbf{z}}(n - D_1 - k)\right)^T \tag{4.21}$$

$$\begin{aligned} \mathbf{W}(n) &= \mathbf{W}(n - D_2) + \frac{\mu}{LA} \sum_{k=0}^{LA-1} e(n - D_1 - k) \bullet \\ &\left[\left(\mathbf{q}(n - D_1 - D_2 - k)^T \cdot f'\left(\mathbf{z}(n - D_1 - k)\right)\right)\tilde{\mathbf{y}}(n - D_1 - k)^T\right] \end{aligned} \tag{4.22}$$

Dans (4.22), nous avons développé la sommation pour la mise à jour des poids **W** afin de l'exprimer en fonction de l'erreur $e(n)$. Nous pouvons donc utiliser le même terme d'erreur pour la mise à jour des poids **W** et **q**. Il est bon de noter que le vecteur d'entrée $\tilde{\mathbf{y}}(n)$ correspond à $[\tilde{y}(n)\ \tilde{y}(n-1)\ \tilde{y}(n-2)..\tilde{y}(n-N+1)]^T$. Les dimensions des matrices et vecteurs sont alors:

$$\dim[\mathbf{W}] = [M \times N]$$

$$\dim[\mathbf{z}] = [M \times 1]$$

$$\dim[\mathbf{q}] = [1 \times M]$$

$$\dim[\tilde{\mathbf{y}}] = [N \times 1]$$

où M représente le nombre de neurones sur la couche cachée et N représente le nombre d'entrées. On peut donc représenter le réseau que nous venons de décrire par la Figure 4.4.

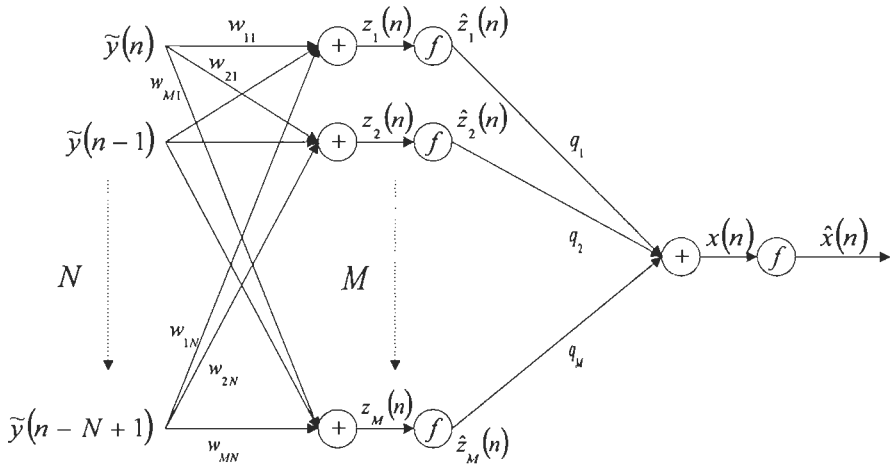


Figure 4.4: Réseau de *M* neurones à 2 couches, *N* entrées et une sortie

4.4 Résultats de simulations

Voyons maintenant l'application du réseau de neurone à l'égalisation de canaux. Dans notre cas, nous désirons transmettre des données de type *NRZ* ($\pm A$) tel que montré par la Figure 2.4. Notons que l'amplitude A est normalisée à 1. Les paramètres de simulations sont :

- 10 000 points et 20 répétitions pour les courbes du taux d'erreurs binaires (*BER – Bit Error Rate*) en fonction du rapport signal sur bruit (*SNR – Signal to Noise Ratio*)
- 5 000 points, 200 répétitions et un *SNR* de 35dB pour les courbes de l'erreur quadratique moyenne (*MSE – Mean Square Error*).
- Bruit additif de type blanc Gaussien.

4.4.1 Canal C1 *linéaire*

L'équation du canal *C1* linéaire est donné par (2.4) et les coefficients sont donnés en (4.23).

$$\begin{bmatrix} h_0 & h_1 & h_2 \end{bmatrix}^T = \begin{bmatrix} 0.3482 & 0.8704 & 0.3482 \end{bmatrix}^T \tag{4.23}$$

Nous présenterons les résultats de simulations des équations (4.16) à (4.22). Pour bien définir l'effet du pipeline sur les performances de convergence de l'algorithme, nous verrons les résultats pour une plage de variation des paramètres D_1, D_2, LA. Les simulations ont par ailleurs démontrés que le paramètre LA (relaxation de somme) doit être égal à D_2. Si on ne respecte pas cette condition, il n'y a pas de convergence du réseau.

Le Tableau 4.1 montre les différents paramètres de simulation pour le réseau de neurone. Dans tous les cas, le nombre d'entrées N est de 3 et le nombre de neurones M sur la couche cachée est de 3. Une étude complète sur ce type de *RNA* sans pipeline a été réalisée dans [DAH00] et [VID99]. Il a été démontré que cette taille de réseau est appropriée pour les canaux linéaires et non linéaires que nous traiterons dans les simulations. Il est bon de noter que le paramètre μ, pour une valeur de D_1 donnée, est plus élevé dans les cas où LA est 4. Ce fait est normal car dans les équations (4.21) et (4.22), μ est divisé par LA ce qui se traduit par un pas d'adaptation global qui est environ le même pour une valeur de D_1 donnée. Les résultats pour le réseau de neurone seront aussi comparés à ceux obtenus par l'algorithme de Viterbi (*MLSE* avec connaissance a priori des coefficients du canal) pour les performances en égalisation et par un filtre transverse linéaire adapté par l'algorithme *LMS*.

Tableau 4.1: Paramètres de simulation pour C1 linéaire

Nom	*D1*	*D2*	*LA*	μ
RNA_1	0	1	1	0.3
RNA_2	20	1	1	0.1
RNA_3	20	4	4	0.3
RNA_4	40	1	1	0.04
RNA_5	40	4	4	0.125
RNA_6	60	1	1	0.03
RNA_7	60	4	4	0.125

Les paramètres du filtre transverse *LMS* pour le cas *C1* linéaire sont:

- Nombre de coefficients, $N = 7$
- Pas d'adaptation, $\mu = 0.1$
- Délai sur les données de supervision, $d = 4$

Les résultats pour les caractéristiques de convergence du réseau sont présentés de la Figure 4.5 à la Figure 4.7. On voit bien que l'ajout du délai D_1 retarde le début de l'adaptation des poids. On remarque aussi que le paramètre D_2 ne change pas trop la valeur de l'erreur quadratique moyenne (*MSE*) résiduelle pour une valeur de D_1 donnée. Ceci s'explique par le fait que le paramètre *LA* est maintenu à D_2 pour conserver la convergence ce qui garde les propriétés algorithmiques. Les courbes présentées ne montrent pas très bien cette dernière remarque mais elle a été confirmée par une simulation de 10 000 données pour l'adaptation. De plus, l'erreur résiduelle est plus élevée ce qui s'explique par le fait que

l'algorithme pipeliné est en fait une approximation de l'algorithme sériel. Cette observation est vraie lorsque le bruit est faible (*SNR* de 35dB). Lorsque le bruit est plus important, l'erreur résiduelle devient la même pour toutes les profondeurs de pipeline. Cette remarque résulte de la Figure 4.8 avec un *SNR* de 17dB (le *BER* est non nul dans ce cas). Dans tous les cas, le *MSE* résiduel des *RNAx* (x = 1, 2, …,7) est toujours plus faible que celui obtenu par *SLMS* qui, pour sa part, se stabilise à une valeur fixe. Par contre, le nombre d'échantillons requis pour la convergence des *RNAx* est plus élevé que le cas *SLMS* dès que la valeur de D_1 dépasse 40.

Pour les performances en égalisation, nous voyons à la Figure 4.9 que tous les *RNAx* possèdent la même performance. Les performances en égalisation du *SLMS* sont inférieures au réseau de neurones pour un SNR en deçà de 16 dB. Par contre, pour le cas du canal *C1* étudié, il devient plus performant à partir d'un SNR de 17dB et il permet de retrouver la séquence originale sans erreur à partir d'un SNR de 20 dB contre 23dB pour les *RNAx*. Il est bon de noter que nous ne pouvons généraliser cette dernière remarque à tous les canaux linéaires. D'autres études [VID99], [DAH00] ont montré des résultats différents et il a été constaté que dans le cas d'un canal linéaire, les deux méthodes (*SLMS* et *RNA*) montrent des résultats équivalents. Enfin, les résultats de l'algorithme de Viterbi, méthode considérée comme optimale, montre sa supériorité en terme de taux d'erreurs binaires (*BER*) en égalisation du canal *C1* linéaire.

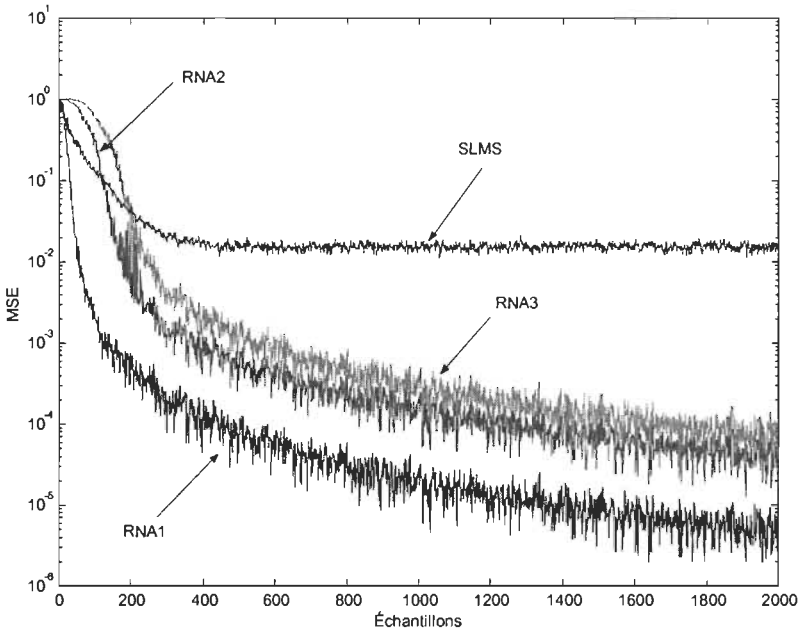


Figure 4.5: Convergence pour RNA_1, RNA_2, RNA_3 et SLMS (C1 lin)

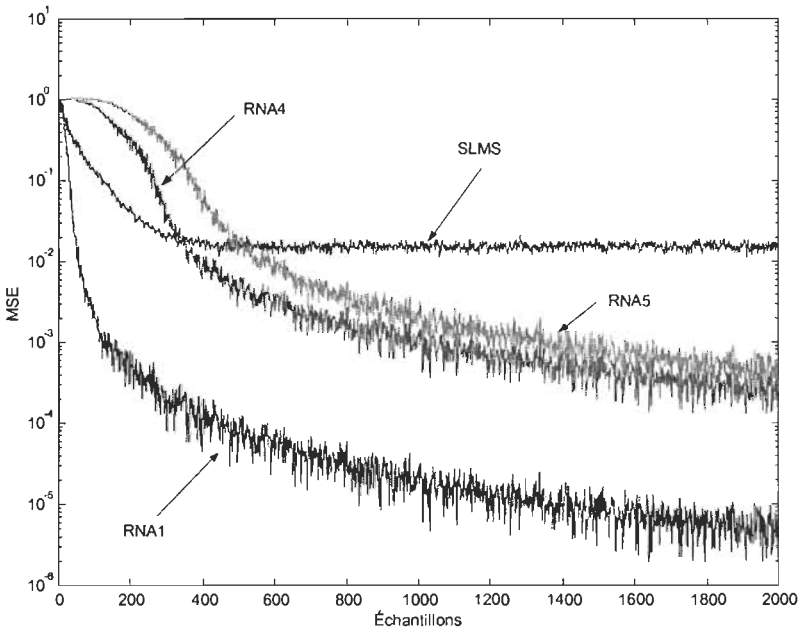


Figure 4.6:Convergence pour RNA_1, RNA_4, RNA_5 et SLMS (C1 lin)

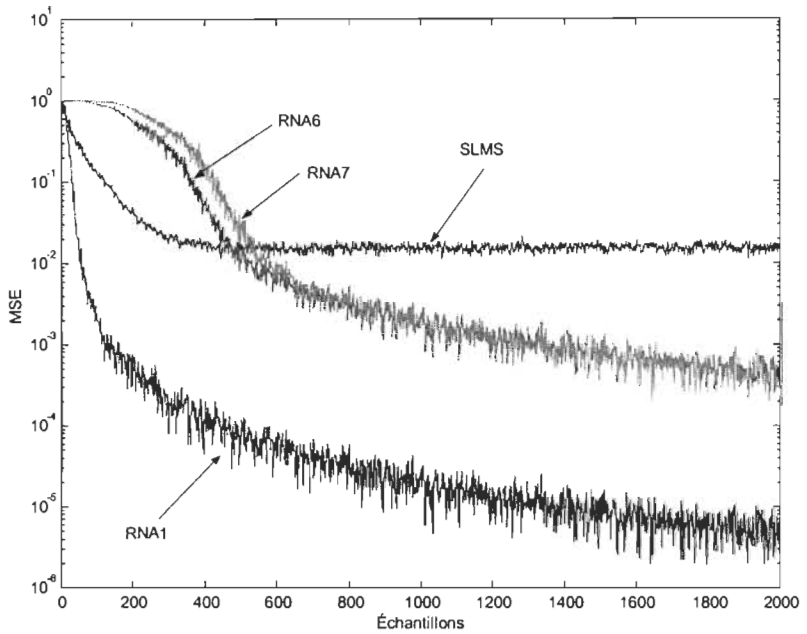


Figure 4.7:Convergence pour RNA_1, RNA_6, RNA_7 et SLMS (C1 lin)

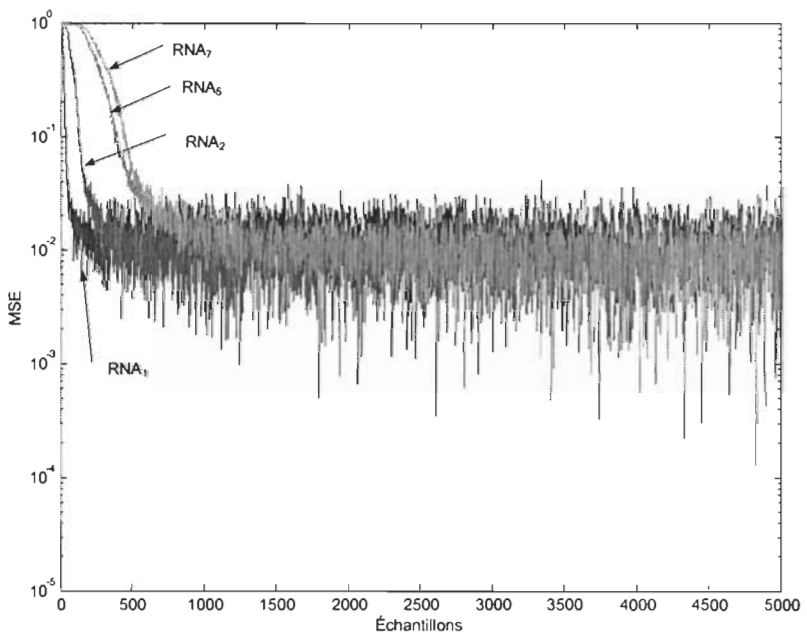


Figure 4.8: Convergence de RNA_1, RNA_2, RNA_5 et RNA_7 pour *SNR* de 17dB

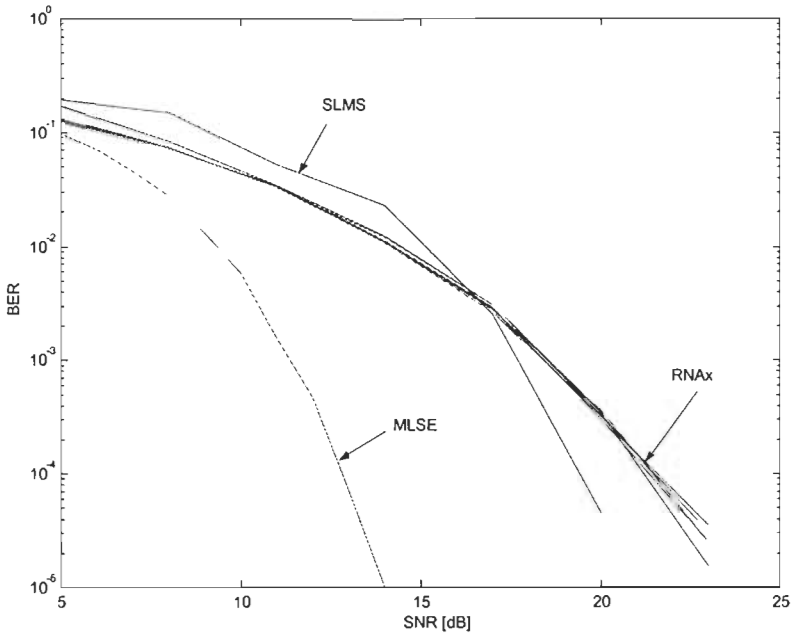


Figure 4.9: Performance en égalisation pour C1 linéaire

4.4.2 Canal C1 *non linéaire*

L'équation du canal *C1* non linéaire est donnée par (4.24) (avec les mêmes coefficients **h** donnés en (4.23)):

$$\begin{aligned} \tilde{y}(n) &= 0.5v(n) + v(n)^3 + \eta(n) \\ v(n) &= \sum_{m=0}^{M-1} h(m)s(n-m) \end{aligned} \tag{4.24}$$

Les paramètres du *LMS* pour le cas *C1* non linéaire sont:

- Nombre de coefficients, $N = 23$
- Pas d'adaptation, $\mu = 0.005$
- Délai sur les données de supervision, $d = 12$

Le Tableau 4.2 montre les différents paramètres de simulations pour le *RNA*. Tel que mentionné et justifié dans la sous section 4.4.1, le nombre d'entrées N est de 3 et le nombre de neurones M sur la couche cachée est de 3.

Tableau 4.2: Paramètres du réseau pour C1 non linéaire

Nom	*D1*	*D2*	*LA*	μ
RNA_1	0	1	1	0.12
RNA_2	20	1	1	0.04
RNA_3	20	4	4	0.12
RNA_4	40	1	1	0.02
RNA_5	40	4	4	0.11
RNA_6	60	1	1	0.02
RNA_7	60	4	4	0.1

Les caractéristiques de convergence pour un *SNR* de 40dB sont montrées de la Figure 4.10 à la Figure 4.12. On remarque que le *LMS* converge vers une valeur de *MSE* élevée (autour de 1) ce qui dénote son incapacité à égaliser les données en présence de non linéarités dans le canal de communication. Dans la Figure 4.11 et la Figure 4.12, on remarque que, pour la même valeur de D_1, la relaxation de somme (lorsque $D_2 = 4$) donne un *MSE* résiduel plus faible comparativement au cas original ($D_2 = 1$). On remarque encore que l'effet du pipeline résulte en un *MSE* résiduel plus élevé mais aucun effet significatif est noté sur le *BER*. De plus, les courbes de convergences présentent quelques ondulations causées par la nature non linéaire du canal et par le fait que le gradient de l'erreur n'est pas assez constant sur une période de D_1 échantillons. Enfin, la Figure 4.13 montre que le *SLMS* est incapable d'égaliser les données et que le réseau de neurones n'est pas vraiment influencé par le pipeline car toutes les courbes *RNAx* ont la même caractéristique.

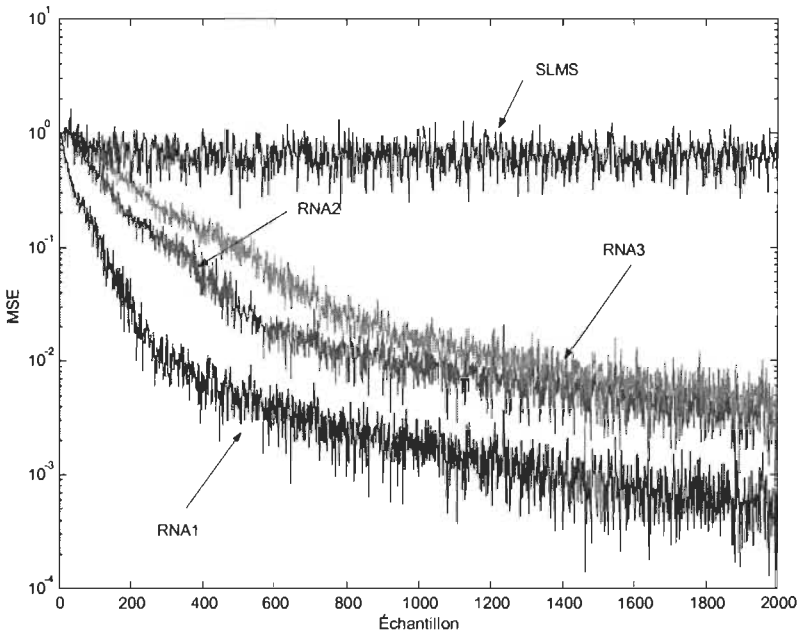


Figure 4.10: Convergence pour RNA_1, RNA_2, RNA_3 et SLMS (C1 nlin)

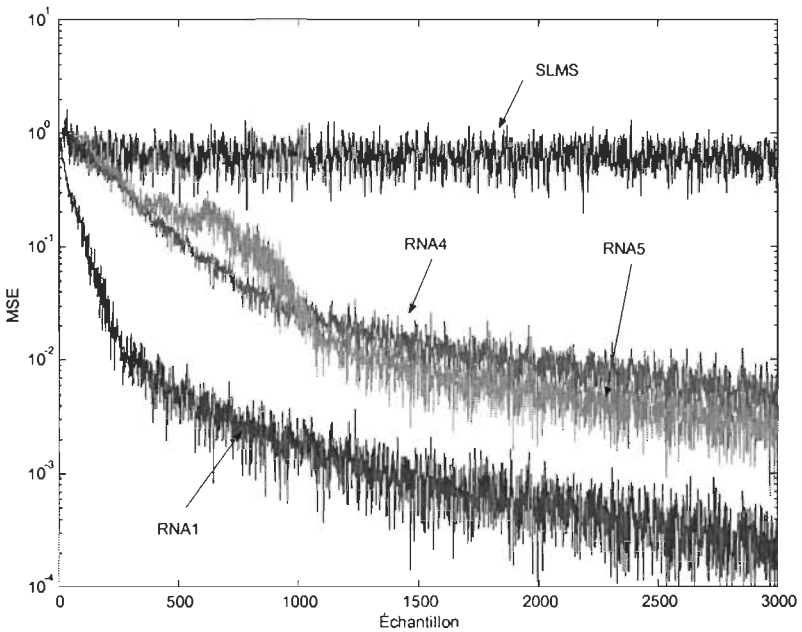


Figure 4.11: Convergence pour RNA_1, RNA_4, RNA_5 et SLMS (C1 nlin)

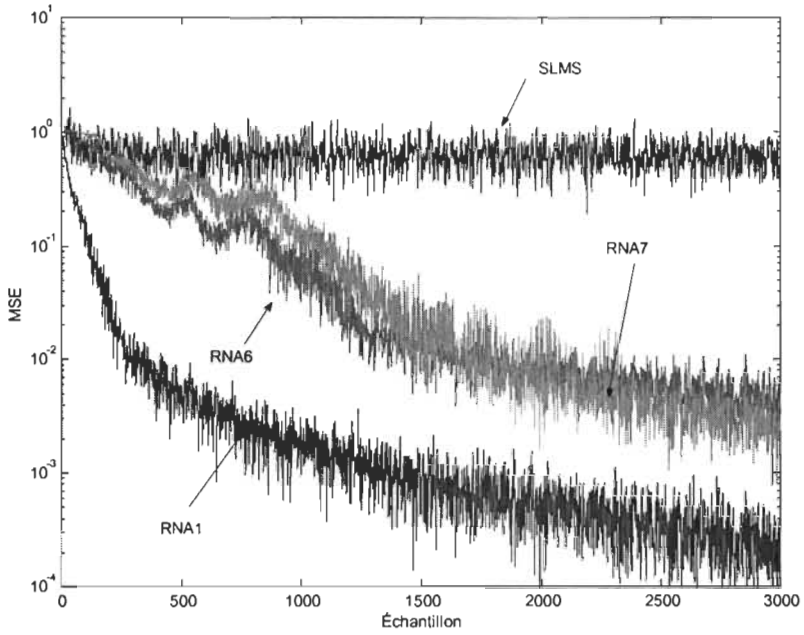


Figure 4.12: Convergence pour RNA_1, RNA_6, RNA_7 et SLMS (C1 nlin)

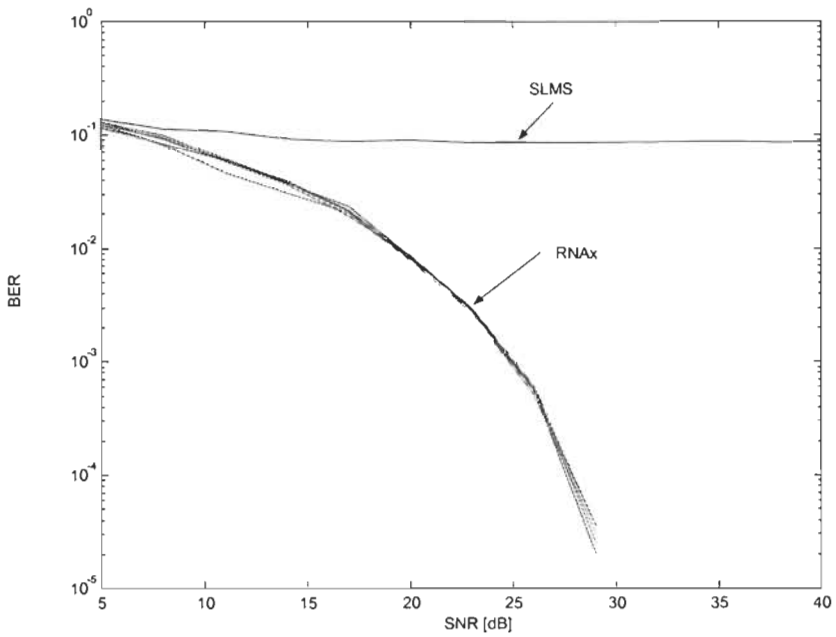


Figure 4.13: BER vs SNR pour C1 non linéaire

4.5 Conclusion

Dans ce chapitre, nous avons vu l'utilisation d'un réseau de neurones pour l'application à l'égalisation des canaux. Un rappel sur la théorie des réseaux de neurones multicouches a ensuite été exposé. L'algorithme de la rétropropagation du gradient a été présenté et les caractéristiques des équations ont montré que la technique de pipeline par la technique de l'anticipation relaxée pouvait être appliquée afin de pipeliner l'algorithme. Les équations correspondantes ont alors été établies.

Par la suite, nous avons vu le cas où le réseau possède une seule couche cachée et un seul neurone de sortie. Les équations associées à ce réseau ont été présentées et nous avons ensuite exposé des résultats de simulations pour montrer l'effet du pipeline sur la convergence du réseau. Un canal linéaire et un canal non linéaire ont été utilisés. Il a été noté que plus le pipeline est profond, plus la convergence du réseau est affectée et que les performances en égalisation demeurent inchangées. Une comparaison entre le *RNA*, *MLSE* et le *LMS* a montré que le *RNA* est assez performant et qu'il permet de traiter des problèmes non linéaires.

Chapitre 5

Intégration sur silicium de l'égaliseur adaptatif pipeliné

Dans ce chapitre, nous aborderons le problème d'intégration en technologie *VLSI* d'un réseau de neurones multicouches pipeliné par la technique de l'anticipation relaxée. Nous présenterons les différentes étapes pour en arriver à l'architecture pipelinée et nous expliquerons les différents éléments de calcul dont l'architecture est composée. Enfin, nous présenterons les résultats des simulations et d'intégration qui permettent de comparer les résultats obtenus en virgule flottante (*Matlab*™) et en virgule fixe (*VHDL*) pour un canal linéaire et non linéaire.

5.1 Architectures de calcul

Afin de réaliser l'architecture pipelinée en *VHDL*, on doit fixer les variables du réseau et de l'algorithme à des grandeurs fixes. Nous avons donc choisi de réaliser un réseau adapté

par la rétropropagation du gradient avec une fonction d'activation de type tangente hyperbolique.

Les paramètres de l'intégration sont:

- Nombre d'entrées, $N = 3$
- Nombre de neurones de la couche cachée, $M=3$
- Relaxation du délai, $D_1 = 13$ (justifié par l'architecture de calcul)
- Relaxation de somme, $D_2 = LA=1$
- Pas d'apprentissage, $\mu = 0.125$
- Quantification sur 16 bits signés en virgule fixe (1 bit de signe, 3 bits avant la virgule, 12 bits en décimales) pour toutes les variables et paramètres.

Le choix des dimensions du *RNA* se base sur les résultats obtenus au Chapitre 3. Il a été établi que la grandeur du réseau à 3 entrées et 3 neurones sur la couche cachée permet de traiter plusieurs cas intéressants. La profondeur de pipeline est choisie de sorte que chacune des opérations du réseau (addition, soustraction, fonction d'activation, etc.) soit séparée par un registre. De cette façon, le chemin critique de l'architecture correspondra au délai de l'élément le plus lent. Ce choix de profondeur de pipeline permet d'augmenter considérablement la vitesse de calcul avec peu d'effets sur les propriétés de convergence. La quantification, pour sa part, se base sur les grandeurs que prennent les différentes variables à l'intérieur du réseau. Ces grandeurs ont été vérifiées par simulation et le choix de 12 bits en décimale permet de minimiser l'effet de quantification, permettant ainsi de

garder une bonne précision [DUF01]. Cette valeur de quantification peut être réduite mais les effets néfastes d'une quantification plus grossière sont vite notés du côté de la fonction d'activation.

Avant de pouvoir en arriver à l'architecture pipelinée, nous devons tout d'abord établir l'architecture de base réalisant les équations de l'algorithme de rétropropagation sans le pipeline. Cette architecture est appelée architecture sérielle que nous nommerons *SRNA – Serial RNA*. Les différents éléments dont les différentes architectures sont composées sont résumés dans le Tableau 5.1.

Tableau 5.1: Éléments des architectures de calcul

$\square$	Registre - délai de 1 cycle d'horloge (z^{-1})
$\boxed{x}$	x Registres - délai de x cycles d'horloge (z^{-x})
$\otimes$	Multiplieur
$\oplus$	Additionneur
f	Fonction d'activation
f'	Dérivée de la fonction d'activation

L'architecture *SRNA* est présentée à la Figure 5.1. La réalisation de l'architecture s'inspire de [SHA98]. Les différentes variables des équations (4.16) à (4.22) (en posant D_1=0 et D_2=LA=1) sont indiquées à travers l'architecture afin de bien suivre le flot des données. On remarque ici que les neurones d'entrées et le neurone de sortie sont en fait des filtres *FIR* avec mise à jour des coefficients. Le reste des éléments sont nécessaires pour le calcul de la mise à jour des poids qui se fait à partir de l'erreur $e(n)$ tant pour les poids d'entrée $w_{ij}(n)$ que pour les poids de sortie $q_i(n)$.

L'ajout du pipeline à l'architecture *SLMS* donne lieu à l'architecture de la Figure 5.2. Nous appelons cette architecture *PIPRNA*. Dans cette architecture, nous avons seulement ajouté les registres aux bons endroits dans l'architecture sérielle afin de correspondre aux équations (4.16) à (4.22). La sommation présente dans les équations (4.21) et (4.22) est illustrée par la série d'additionneur – registre et la longueur est notée *LA*-1. Cette configuration est tirée de [SHA98].

Puisque nous voulons avoir un seul registre entre les différents éléments de calcul, nous pouvons dès maintenant poser D_2 à 1. La valeur de D_2 correspond en fait à la profondeur de pipeline d'un additionneur. Pour fixer une valeur à D_1, on doit procéder à la resynchronisation de l'architecture *PIPRNA* en supposant que D_1 est suffisamment grand. Le résultat de la resynchronisation, que nous appellerons *PIPRNA1*, est donné à la Figure 5.3. L'idée est de redistribuer les D_1 registres de la sortie (après $e(n)$) vers l'entrée en laissant un registre entre chaque élément de calcul. Après la resynchronisation, nous trouvons que $D_1 = 13$. Il est bon de noter que deux registres ont été laissés après la dérivée de la fonction d'activation. Nous verrons à la section 5.2 ce que justifie cette action.

Pour écrire le code *VHDL* du réseau, nous pouvons utiliser deux approches. La première consiste à définir le réseau de façon comportementale et écrire le code *VHDL* des équations directement. Cette approche possède le désavantage de ne pas avoir de contrôle sur les éléments qui réaliseront nos équations lors de l'intégration sur silicium. Par contre elle donne un code plus compact et plus facile à lire.

L'autre approche consiste à définir une architecture de calcul basée sur des éléments de calculs simples (addition, soustraction, multiplication, etc.). Cette approche structurelle (*RTL : "Register Transfert Logic"*) permet d'imposer des éléments définis à l'avance pour l'intégration sur silicium. Par exemple, on peut créer des cellules d'additions au niveau transistor et dessin de masque et ensuite définir un code *VHDL* qui imposera l'utilisation de ces cellules lors de l'intégration physique. Nous avons retenu cette deuxième approche pour la réalisation en *VHDL* car elle offre une plus grande flexibilité lors de l'intégration.

Pour faciliter l'écriture du code *VHDL*, nous avons groupé certains éléments de *PIPRNA1* pour former une architecture modulaire nommée *PIPRNA2*. Le résultat est donné à la Figure 5.4.

Les éléments $neurone_i$ et $neurone_o$ (Figure 5.5 et Figure 5.6) sont des entités que nous avons créées et utilisées dans l'architecture de *PIPRNA2*. En fait, dans l'élaboration d'une architecture, il est très important de respecter la synchronisation des données. Un délai soit manquant ou en trop peut entraîner un disfonctionnement total de l'algorithme. Les différentes étapes que nous venons de franchir sont donc très importantes car elles permettent de minimiser les erreurs de synchronisation.

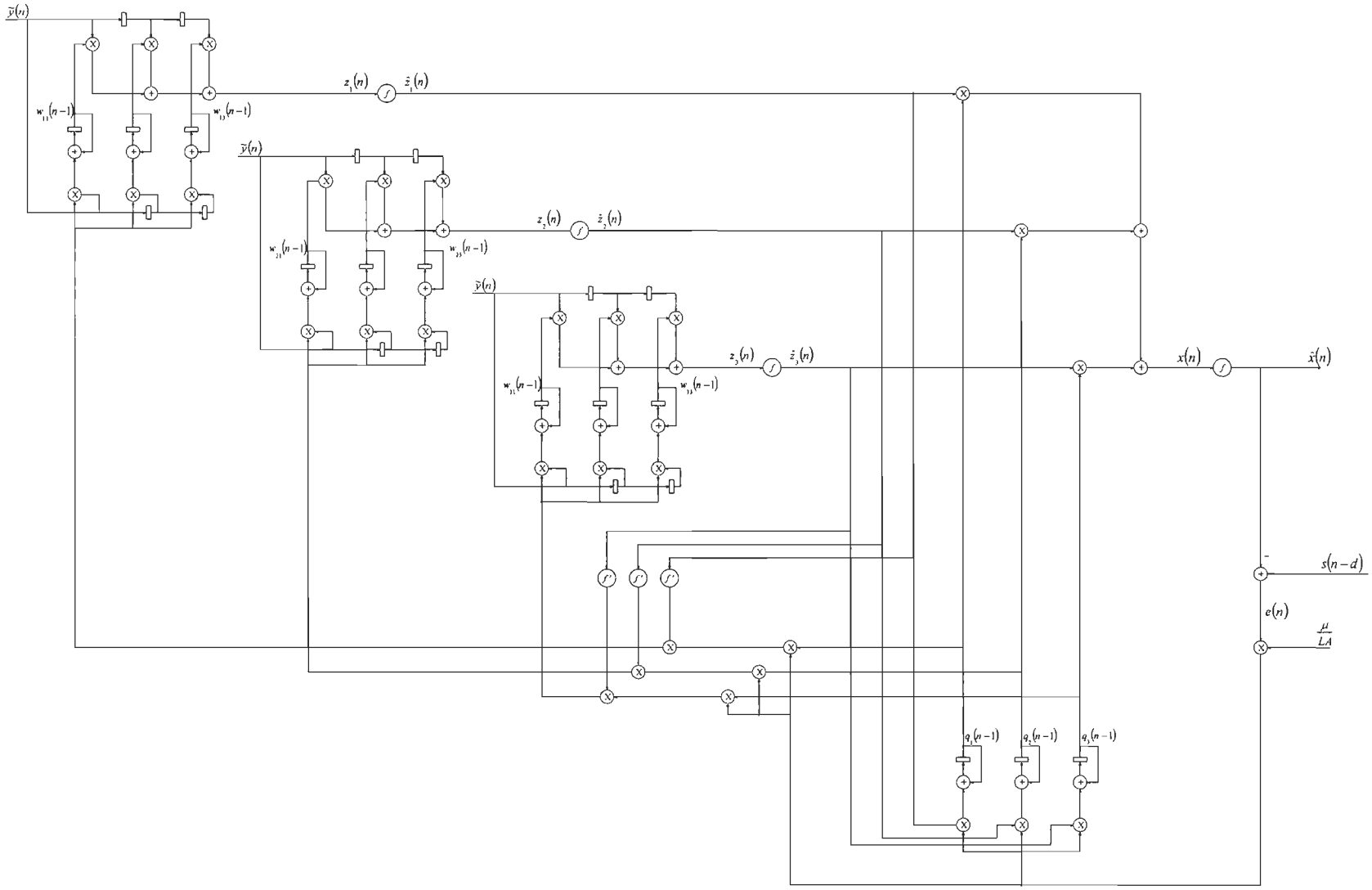


Figure 5.1: Architecture sérielle (*SRNA*)

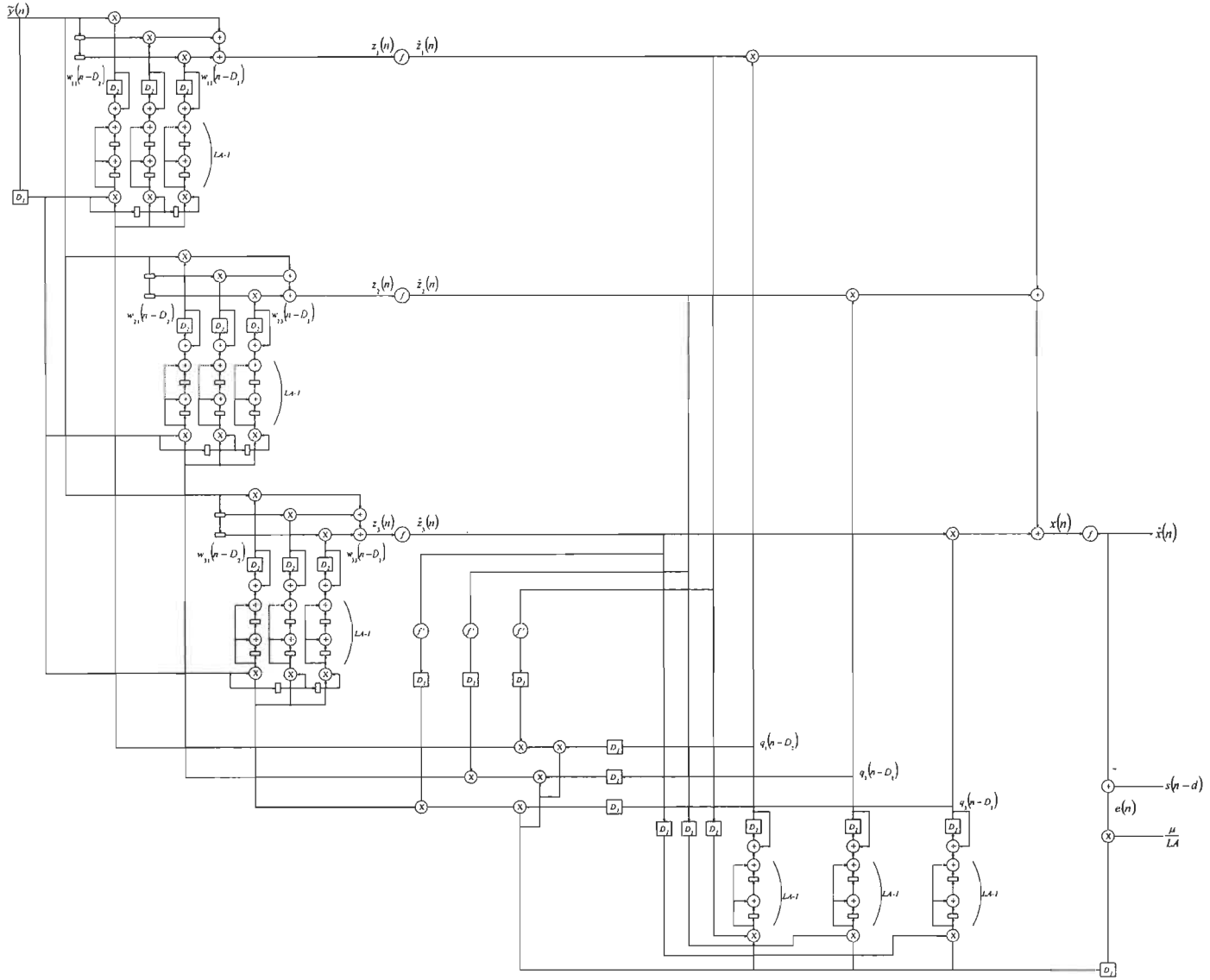


Figure 5.2: Architecture pipelinée avant resynchronisation (*PIPRNA*)

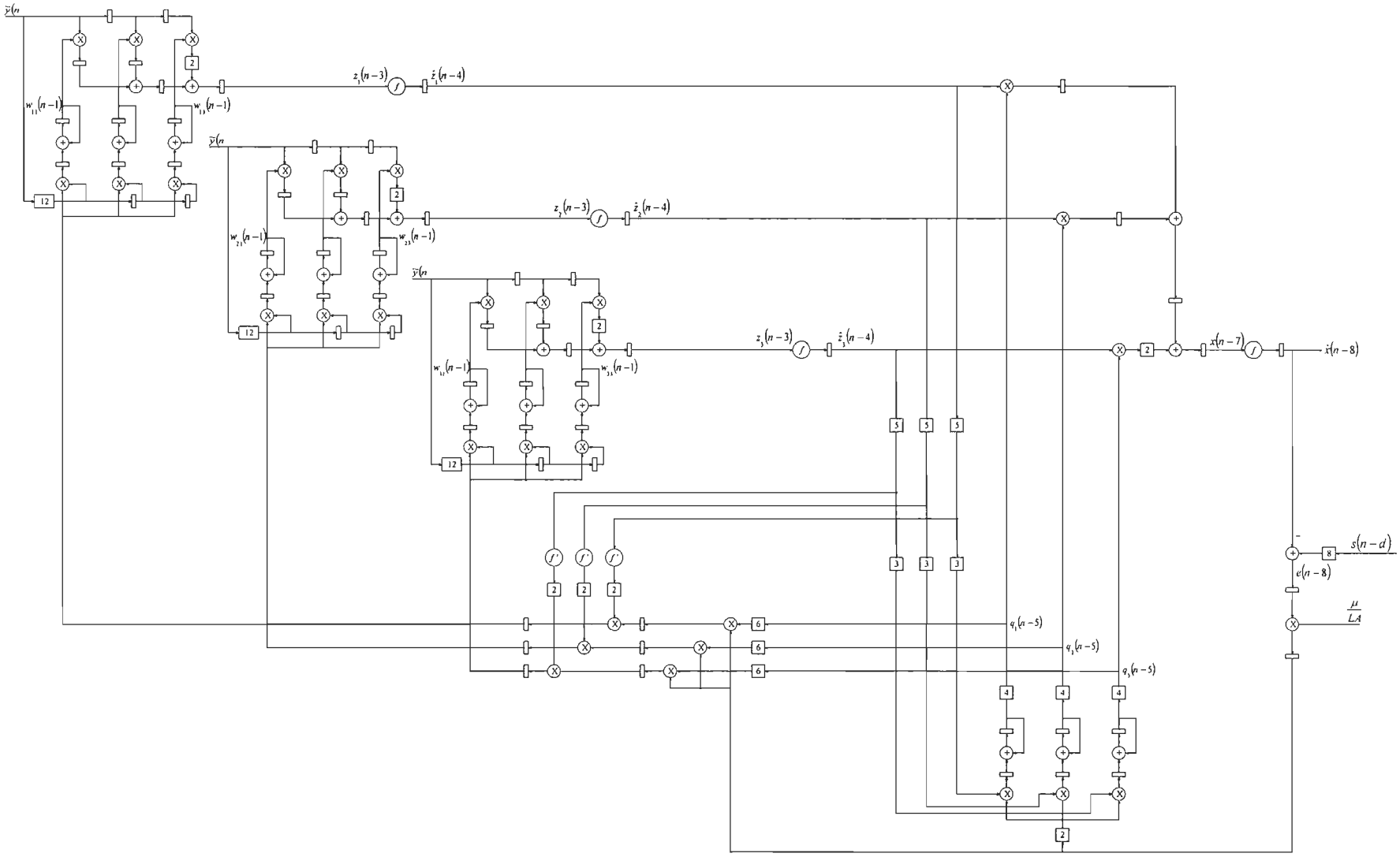


Figure 5.3: Architecture pipelinée après resynchronisation (*PIPRNA1*)

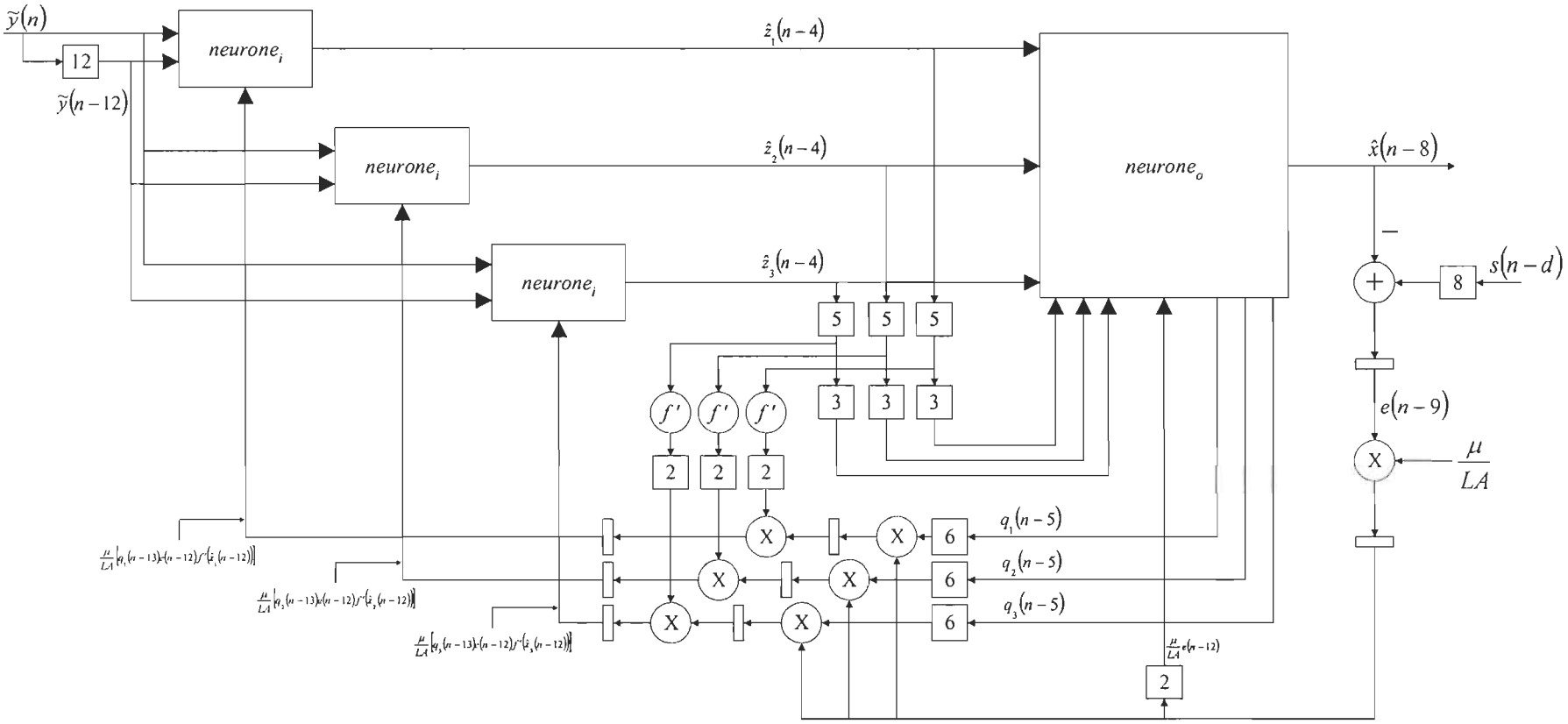


Figure 5.4: Architecture pipelinée réalisée en *VHDL* (*PIPRNA2*)

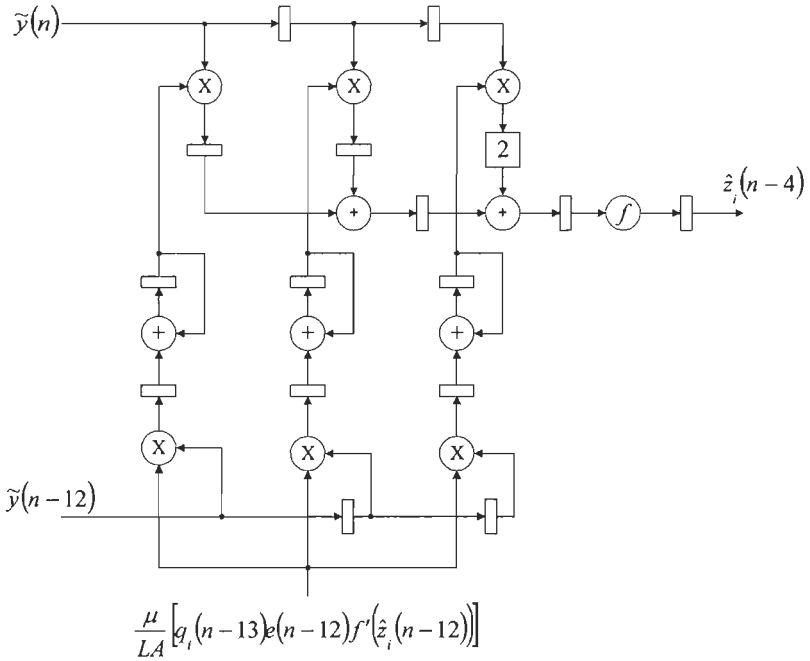


Figure 5.5: Schéma de *neurone*$_i$

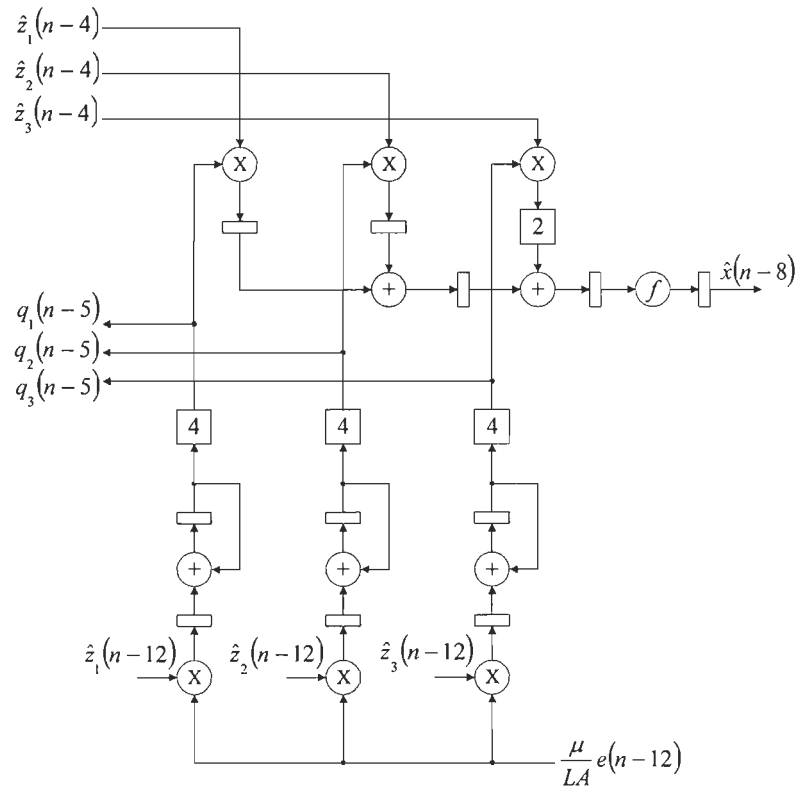


Figure 5.6: Schéma de *neurone*$_o$

5.2 Éléments de calculs

Le fonctionnement de l'architecture repose évidemment sur ses éléments de calculs. Pour cette application, nous avons choisi de réaliser les éléments de calculs de façon comportementale étant donné qu'il s'agit des opérations de base. L'outil de synthèse *Synopsys*™ les réalisera avec des composants de sa librairie qui est déjà optimisée. Le code *VHDL* de toute l'architecture et des composants est donnée dans [DUF01].

5.2.1 Addition et soustraction

L'addition et la soustraction sont des opérations assez simples et nous ne s'y attarderons pas. La seule chose à noter est que nous avons défini au niveau comportemental une entité *adder* en *VHDL* qui correspond à un additionneur sans registre. Ensuite, nous avons défini une entité *addreg* qui correspond à un additionneur suivi d'un registre. Cette approche permet de simplifier grandement le code *VHDL* de l'architecture globale. Nous l'avons d'ailleurs appliquée à toutes les composantes de l'architecture.

5.2.2 Multiplication

La multiplication est pour sa part un peu plus complexe car nous devons tronquer le résultat afin de demeurer sur des grandeurs de 16 bits. En effet, en multipliant deux mots de 16 bits, nous obtenons un mot de 32 bits. Or nous devons sélectionner les bons bits du résultat pour garder la quantification au même niveau.

Voyons un exemple présenté à la Figure 5.7 a) en multipliant 1 par 2 (*S* correspond au bit de signe et *V* à la position de la virgule).

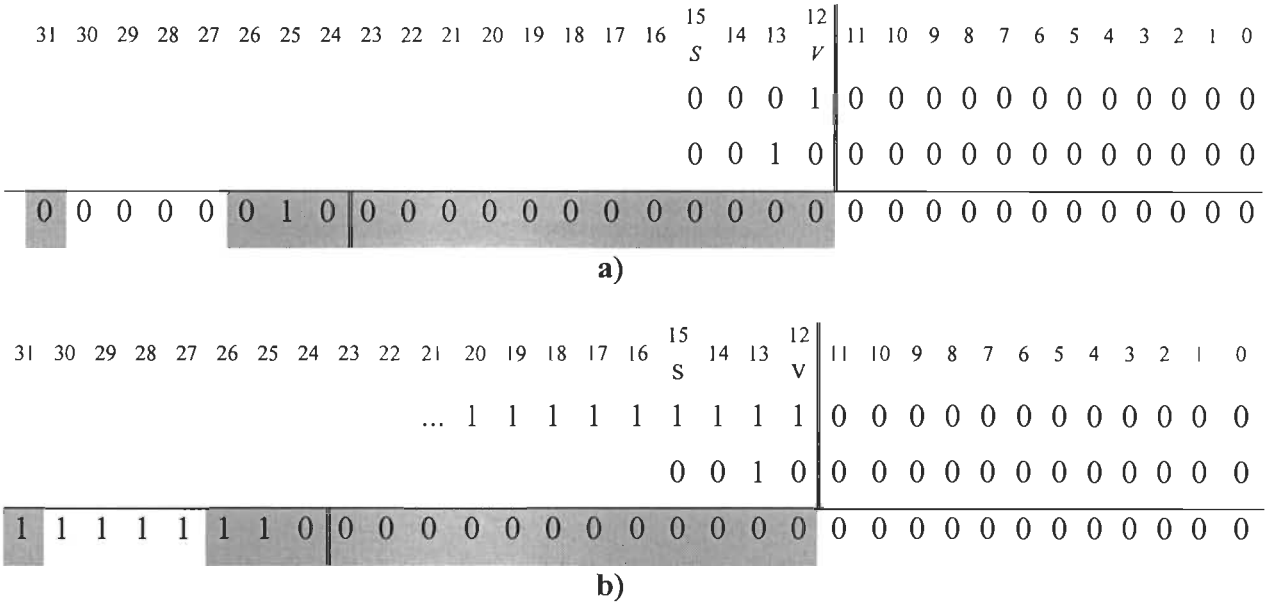


Figure 5.7: Exemple de multiplications a) 1 par 2 b) 1 par -1

On doit donc garder le bit 31 qui constitue le bit de signe (toujours 0 en résultat positif), ensuite on conserve les bits 26 à 12 pour le reste du résultat. La section ombragée de la Figure 5.7 dénote les bits qui sont conservés pour le résultat.

Vérifions maintenant les nombres négatifs. L'exemple de la Figure 5.7 b) montre la multiplication de 1 par –1. Le résultat est –1 et dans la représentation en complément à 2, le nombre –1 doit avoir 1 au lieu de 0 dans les bits 16 jusqu'à l'infini. Le résultat contiendra non pas des 0 dans les bits 28 à 31 mais bel et bien 1 ce qui respecte la représentation en complément à 2. L'opération de multiplication signée de la librairie *IEEE* en *VHDL* tient compte de ce comportement et nous n'avons pas à s'en soucier dans l'écriture du code

VHDL. Nous avons donc créé une entité *multiplier* pour le comportement de la multiplication et une entité *multreg* qui contient le registre de synchronisation.

5.2.3 Fonction d'activation ($f_{PL}(x)$)

La fonction d'activation est l'élément de calcul le plus complexe à réaliser car on désire une implantation qui soit rapide sans utiliser beaucoup de surface. Plusieurs méthodes sont proposées dans [MUR92] et nous avons retenu la méthode de Amin, Curtis et Hayes-Gill [AMI97]. Le principe de cette méthode consiste à réaliser la fonction d'activation en utilisant les morceaux de droites (équations linéaires de la forme $y = mx + b$) dont les pentes sont des multiples de 2 (simplifie grandement la multiplication) et les ordonnées à l'origine sont des multiples de 2^n où n est un entier positif ou négatif. Cette méthode est appelée "*Piecewise Linear Approximation of Non linear function - (PLAN)*" [AMI97] que nous noterons $f_{PL}(x)$.

Avant d'appliquer la méthode, on doit choisir les équations des droites qui reflète le mieux la fonction d'activation. Il faut aussi faire un compromis dans le choix des pentes et des ordonnés à l'origine. Puisque nous avons retenu la tangente hyperbolique comme fonction d'activation, les pentes m sont comprises entre 1 et 0. Pour l'ordonnée à l'origine b, nous avons une valeur toujours inférieure à 1. Le choix de la valeur de l'ordonnée à l'origine ne doit pas avoir une grande longueur décimale. C'est-à-dire que si nous voulons utiliser par exemple b = 0.5009765625, nous aurons un circuit plus complexe que si nous utilisons b = 0.5. Cette affirmation sera justifiée au cours du développement de la méthode. Les équations choisies pour les droites ont été déterminées de façon empirique en respectant les

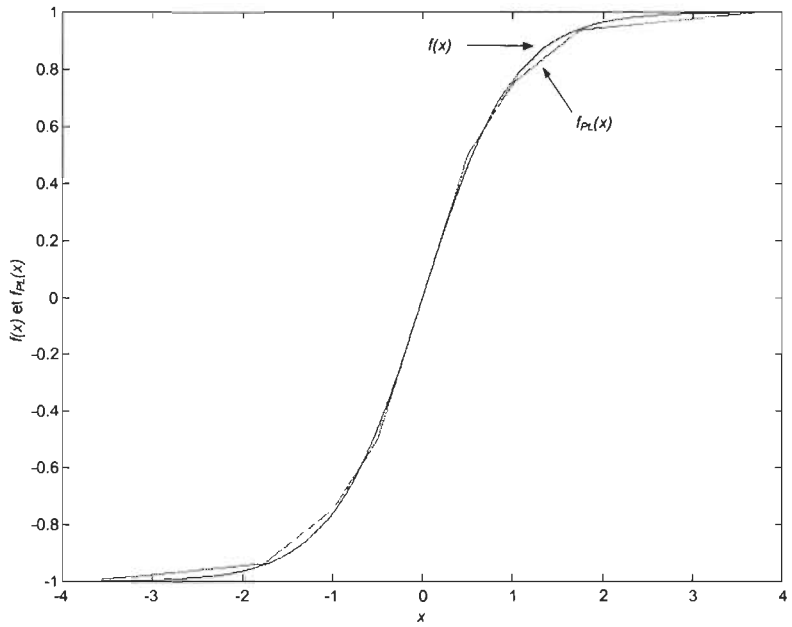
points de changement de sorte à n'avoir aucune discontinuité. Nous avons donc établi les équations du Tableau 5.2.

Tableau 5.2: Équations de la fonction d'activation $f_{PL}(x)$

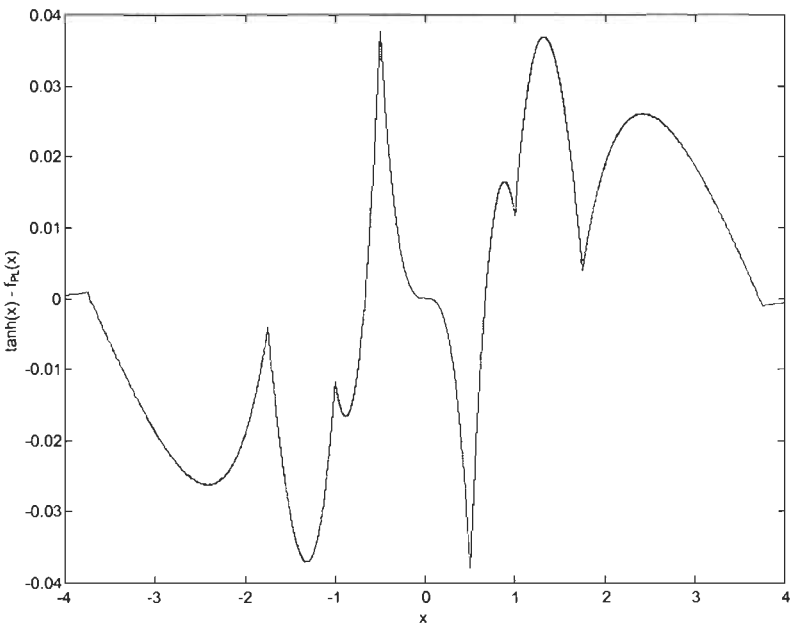
Équation	Intervalle
$f_{PL}(x) = \|x\|$	$z1 : 0 \leq \|x\| < 0.5$
$f_{PL}(x) = 0.5\|x\| + 0.25$	$z2 : 0.5 \leq \|x\| < 1$
$f_{PL}(x) = 0.25\|x\| + 0.5$	$z3 : 1 \leq \|x\| < 1.75$
$f_{PL}(x) = 0.03125\|x\| + 0.8828125$	$z4 : 1.75 \leq \|x\| < 3.75$
$f_{PL}(x) = 1$	$z5 : 3.75 \leq \|x\|$

Comme la fonction d'activation est impaire (symétrique en diagonale à 0), on peut utiliser cette propriété et dire que lorsque la valeur de x est négative, la valeur de $f_{PL}(x)$ est égale à $-\tanh(|x|)$. De cette façon, nous n'avons qu'à réaliser la fonction pour la valeur absolue de x et adapter le signe du résultat $f_{PL}(|x|)$ en fonction du signe de x. La Figure 5.8 a) montre le résultat de l'approximation (résultat du code *VHDL* de $f_{PL}(x)$) et de la courbe réelle $f(x)=tanh(x)$. L'erreur de l'approximation est montrée à la Figure 5.8 b).

On remarque que l'approximation $f_{PL}(x)$ est toujours en dessous de la fonction $tanh(x)$ quand la valeur absolue de x est supérieure à 1. Cette propriété à été décrite dans [AMI97] comme étant plus sûre car les chances de divergence causée par l'approximation sont réduites mais la vitesse de convergence est un peu plus lente.



a)



b)

Figure 5.8: Fonction d'activation a) Approximation b) Erreur d'approximation

La fonction $f_{PL}(x)$ est réalisée par le schéma de la Figure 5.9.

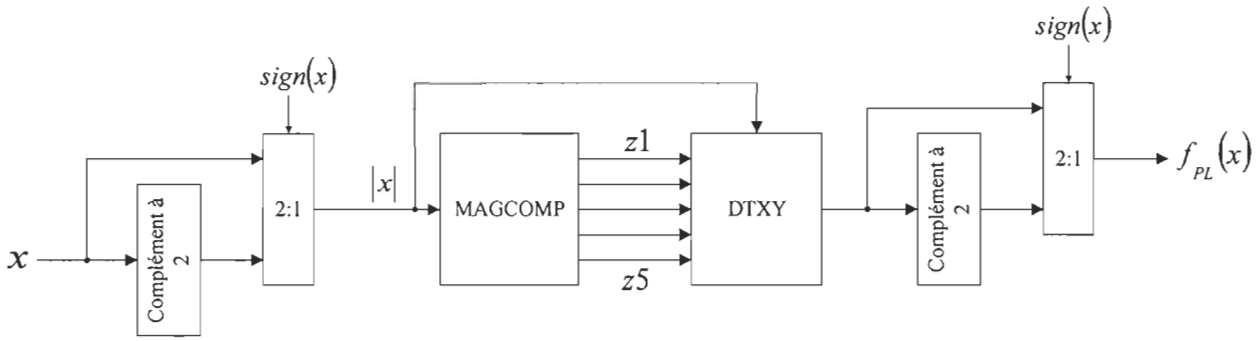


Figure 5.9: Réalisation de la fonction $f_{PL}(x)$

L'entrée x est d'abord convertie en valeur absolue par le complément à 2 (réalisé avec un additionneur) et le multiplexeur 2 à 1. Ensuite, la valeur absolue de x entre dans *MAGCOMP* qui est un comparateur de magnitude. Les signaux *z1* à *z5* générés correspondent aux intervalles définis dans le Tableau 5.2. Ensuite, le bloc *DTXY* permet un transfert direct de $|x|$ à $f_{PL}(|x|)$ en utilisant la technique d'orientation des bits ("*bit mapping*"). Enfin, la sortie $f_{PL}(|x|)$ est dirigée par le signe de x (grâce à la symétrie de la fonction d'activation).

Voyons en premier la réalisation du module *MAGCOMP*. Le vecteur d'entrée $|x|$ est toujours positif (valeur absolue) sur 16 bits. On doit maintenant définir des équations logiques pour *z1* à *z5* selon le vecteur d'entrée. On retrouve dans le Tableau 5.3 les différents intervalles de $|x|$ pour lesquelles les valeurs de $z1$ à $z5$ passent à l'état 1 logique. Il est à noter qu'il ne peut y avoir qu'un seul des signaux $z1$ à $z5$ à l'état 1 logique à la fois.

Les lettres A à P (A est le bit le plus significatif) correspondent aux 16 bits dont $|x|$ est constitué.

Tableau 5.3: Table de correspondance du comparateur de magnitude

	$\|x\|$	A Signe	B	C	D Virgule	E	F	G	H	I	J	K	L	M	N	O	P
$z1=1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0.499..	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
$z2=1$	0.5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	0.999..	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
$z3=1$	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	1.7499..	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1
$z4=1$	1.75	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
	3.7499..	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1
$z5=1$	3.75	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	7.999..	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Par inspection du Tableau 5.3, on trouve les expressions en (5.1) pour les variables $z1$ à $z5$.

$$\begin{aligned} z1 &= \overline{B + C + D + E} \\ z2 &= \overline{B + C + D + \overline{E}} \\ z3 &= \overline{B}\,\overline{C}D\left(\overline{E} + \overline{F}\right) \\ z4 &= \overline{z1 + z2 + z3 + z5} \\ z5 &= B + CDEF \end{aligned} \tag{5.1}$$

Puisque le bit A représente le bit de signe et est toujours 0, il n'apparaît pas dans (5.1). On remarque aussi que l'implémentation physique de *MAGCOMP* implique seulement de simples portes logiques. Ce circuit est donc simple à réaliser, ne prends pas beaucoup

d'espace et est très rapide si on le compare à un multiplieur de 16 bits ou même à un additionneur de 16 bits.

Le cœur de la fonction d'activation repose sur la réalisation du module *DTXY*. L'équation qui relie $|x|$ à $f_{PL}(|x|)$ est de la forme $f_{PL}(x) = m|x| + b$. Les bits de $|x|$ seront notés de A à P avec A comme étant le plus significatif et les bits de $f_{PL}(|x|)$ seront notés de A' à P' avec A' comme étant le bit le plus significatif. L'idée de la méthode est d'orienté les bits de $|x|$ vers les bits de $f_{PL}(|x|)$ par de simples équations logiques.

La valeur de l'ordonnée à l'origine b peut être exprimé par l'équation (5.2) où a_i représente la valeur (0 ou 1) du bit en position décimale i et T représente le nombre de bits en décimales (12 dans ce cas).

$$b = \sum_{i=1}^{T} a_i 2^{-i} \tag{5.2}$$

La valeur de la pente m est un multiple de 2^{-n} où n est un entier négatif ce qui correspond à faire un décalage des bits vers la droite. La méthode veut que s'il n'existe pas de i pour lequel la valeur non nulle de $a_i 2^{-i}$ de l'équation (5.2) est inférieure ou égale à l'opération $m|x|$, les bits de $f_{PL}(|x|)$ prennent la même valeur que les bits de $|x|$ mais décalés selon la valeur de m. Nous noterons cette condition comme étant la condition de transfert direct.

Voyons un exemple par le Tableau 5.4 a) où $m = 0.25$ et $b = 0.5$. La valeur de $|x|$ peut varier de 0 à 1.999..., tel que montré par des k (k vaut soit 0 ou 1) dans les bits D à P. On remarque dans ce premier cas que la condition de transfert direct est respectée car il n'y a pas de i pour lequel la valeur non nulle de $a_i 2^{-i}$ de l'équation (5.2) est inférieure ou égale à l'opération $m|x|$. En d'autres mots, il n'y a pas de 1 logique dans la valeur de b vis-à-vis un bit k de la ligne $m|x|$ du Tableau 5.4 a). On peut alors écrire que les bits de sortie A' à D' sont tous 0. Le bit E' est toujours 1 logique et les bits F' à P' prennent respectivement les valeurs des bits D à N (décalage de 2 positions). Le calcul de $f_{PL}(|x|)$ est donc très simple.

Tableau 5.4: Exemple d'orientation des bits a) $b = 0.5$ b) $b = 0.125$

	A Signe	B	C	D Virgule	E	F	G	H	I	J	K	L	M	N	O	P
$\|x\|$	0	0	0	k	k	k	k	k	k	k	k	k	k	k	k	k
$m\|x\|$	0	0	0	0	0	k	k	k	k	k	k	k	k	k	k	k
b	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$m\|x\| + b$	0	0	0	0	1	k	k	k	k	k	k	k	k	k	k	k
	A' Signe	B'	C'	D' Virgule	E'	F'	G'	H'	I'	J'	K'	L'	M'	N'	O'	P'

a)

	A Signe	B	C	D Virgule	E	F	G	H	I	J	K	L	M	N	O	P
$\|x\|$	0	0	0	k	k	k	k	k	k	k	k	k	k	k	k	k
$m\|x\|$	0	0	0	0	0	k	k	k	k	k	k	k	k	k	k	k
b	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
$m\|x\| + b$	0	0	0	0	k'	k'	k'	k	k	k	k	k	k	k	k	k
	A' Signe	B'	C'	D' Virgule	E'	F'	G'	H'	I'	J'	K'	L'	M'	N'	O'	P'

b)

Changeons b pour 0.125. Le Tableau 5.4 b) montre les étapes pour obtenir $f_{PL}(|x|)$. Dans ce cas, la condition de transfert direct n'est pas respectée pour $i = 3$ (il y a un bit de b à 1 vis-à-vis un bit k de l'opération $m|x|$). Il faut donc écrire une table de vérité pour déterminer les valeurs de E', F' et G' (montré par k' au lieu de k dans leur cases respectives) car lorsque le bit E de $|x|$ sera à 1, il y aura génération d'une retenue lors de l'addition de b et $m|x|$ et nous devons en tenir compte. Puisque $|x|$ varie entre 0 et 1.999..., les variables d'entrées de la table de vérité, présentée au Tableau 5.5, sont D et E et les variables de sortie sont E', F' et G'. On peut alors utiliser les tables de Karnaugh pour la simplification et obtenir les équations booléennes de l'orientation des bits entre $|x|$ et $f_{PL}(|x|)$. Pour les valeurs de H' à P', la condition de transfert direct est respectée et nous pouvons directement transférer les valeurs de F à N (décalage donné par m) aux bits H' à P'. Nous obtenons, pour cet exemple, les équations présentés en (5.3).

Tableau 5.5: Table de vérité pour l'exemple d'orientation des bits

D	E	E'	F'	G'
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	0	0

$$
\begin{aligned}
&A' = B' = C' = D' = 0 \\
&E' = DE \\
&F' = D \ xor \ E \\
&G' = \overline{E} \\
&H' = F \\
&I' = G,...
\end{aligned}
\tag{5.3}
$$

Nous sommes maintenant prêts à établir les équations qui réaliseront le bloc *DTXY*. Débutons par l'intervalle $z1 : 0 \leq |x| < 0.5$. Le Tableau 5.6 présente les différentes étapes pour obtenir $f_{PL}(|x|)$. Pour cet intervalle, la condition de transfert direct est respectée. Nous pouvons donc déterminer immédiatement la valeur des bits A' à P' que nous présentons en (5.4).

$$A' = 0, B' = 0, C' = 0, D' = 0, E' = 0, F' = F, G' = G, H' = H, \ldots \tag{5.4}$$

Voyons maintenant le cas de l'intervalle $z2 : 0.5 \leq |x| < 1$. Le Tableau 5.7 montre les détails pour cet intervalle. Si on inspecte le Tableau 5.7, la condition de transfert direct n'est pas respectée pour $i = 2$. Par contre, la valeur de $|x|$ en $i = 2$ est toujours 1 alors le résultat ne dépend pas des variables A à P. Le transfert entre $|x|$ et $f_{PL}(|x|)$ peut donc être direct et le résultat est présenté en (5.5).

$$A' = 0, B' = 0, C' = 0, D' = 0, E' = 1, F' = 0, G' = F, H' = G, \ldots \tag{5.5}$$

Tableau 5.6: *DTXY* pour $z1 = 1$

	A Signe	B	C	D Virgule	E	F	G	H	I	J	K	L	M	N	O	P
$\|x\|$	0	0	0	0	0	k	k	k	k	k	k	k	k	k	k	k
$m\|x\|$	0	0	0	0	0	k	k	k	k	k	k	k	k	k	k	k
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$m\|x\| + b$	0	0	0	0	0	k	k	k	k	k	k	k	k	k	k	k
	A' Signe	B'	C'	D' Virgule	E'	F'	G'	H'	I'	J'	K'	L'	M'	N'	O'	P'

Tableau 5.7: *DTXY* pour $z2$ =1

	A Signe	*B*	*C*	*D* Virgule	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*
$\|x\|$	0	0	0	0	1	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
$m\|x\|$	0	0	0	0	0	1	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
b	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
$m\|x\| + b$	0	0	0	0	1	0	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
	A' Signe	*B'*	*C'*	*D'* Virgule	*E'*	*F'*	*G'*	*H'*	*I'*	*J'*	*K'*	*L'*	*M'*	*N'*	*O'*	*P'*

Pour le cas $z3 : 1 \leq |x| < 1.75$, nous avons les étapes présentées au Tableau 5.8. La condition de transfert direct est respectée pour cet intervalle et la valeur des bits *A'* à *P'* est présenté en (5.6).

Tableau 5.8: *DTXY* pour $z3$ =1

	A Signe	*B*	*C*	*D* Virgule	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*
$\|x\|$	0	0	0	1	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
$m\|x\|$	0	0	0	0	0	1	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
b	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$m\|x\| + b$	0	0	0	0	1	1	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
	A' Signe	*B'*	*C'*	*D'* Virgule	*E'*	*F'*	*G'*	*H'*	*I'*	*J'*	*K'*	*L'*	*M'*	*N'*	*O'*	*P'*

$$A' = 0, B' = 0, C' = 0, D' = 0, E' = 1, F' = 1, G' = E, H' = F, I' = G, \ldots \quad (5.6)$$

Le Tableau 5.9 montre les étapes de l'intervalle $z4 : 1.75 \leq |x| < 3.75$. La condition de transfert direct n'est pas respectée en $i = 7$ et nous devons établir une table de vérité (présentée au Tableau 5.10) pour obtenir les variables *H'*, *I'*, *J'* et *K'*. Un élément important à remarquer ici est qu'il est impossible que les bits *C*, *D*, *E* et *F* soient tous à 1 en même

temps à cause de la plage de variation de $|x|$. Ceci à pour effet que la retenue pouvant être générée lors de l'addition de b en position K' ne peut se propager jusqu'en position G'. On peut dès lors écrire que E', F' et G' sont tous à 1 peu importe $|x|$.

Tableau 5.9: *DTXY* pour $z4$ =1

	A Signe	*B*	*C*	*D* Virgule	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*
$\|x\|$	0	0	*k*	*K*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
$m\|x\|$	0	0	0	0	0	0	0	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
b	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0
$m\|x\| + b$	0	0	0	0	1	1	1	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*	*k*
	A' Signe	*B'*	*C'*	*D'* Virgule	*E'*	*F'*	*G'*	*H'*	*I'*	*J'*	*K'*	*L'*	*M'*	*N'*	*O'*	*P'*

Tableau 5.10: Table de vérité pour *H'*, *I'*, *J'* et *K'* en $z4$=1

C	*D*	*E*	*F*	*H'*	*I'*	*J'*	*K'*
0	0	0	0	*dc*	*dc*	*dc*	*dc*
0	0	0	1	*dc*	*dc*	*dc*	*dc*
0	0	1	0	*dc*	*dc*	*dc*	*dc*
0	0	1	1	*dc*	*dc*	*dc*	*dc*
0	1	0	0	*dc*	*dc*	*dc*	*dc*
0	1	0	1	*dc*	*dc*	*dc*	*dc*
0	1	1	0	*dc*	*dc*	*dc*	*dc*
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	*dc*	*dc*	*dc*	*dc*

Après simplification de la table de vérité du Tableau 5.10 où les éléments "*dc*" sont des conditions sans importance (*don't care*), nous obtenons les valeurs de A' à P' par (5.7).

$$A' = 0,\ B' = 0,\ C' = 0,\ D' = 0,\ E' = 1,\ F' = 1,\ G' = 1,\ H' = 1,\ I' = C(EF+D),\\ J' = E\ xor\ F,\ K' = \overline{F},\ L' = G,\ M' = H,\ N' = I,\ \ldots \tag{5.7}$$

Pour l'intervalle $z5 : 3.75 \leq |x|$, seul le bit D' est à 1 (partie de saturation de $f_{PL}(x)$). Les valeurs de A' à P' sont alors données par (5.8).

$$A' = 0,\ B' = 0,\ C' = 0,\ D' = 1,\ E' = 0,\ F' = 0,\ G' = 0,\ H' = 0,\ldots \tag{5.8}$$

En combinat les résultats obtenus dans les équations (5.4) à (5.8) avec les variables $z1$ à $z5$, nous obtenons l'orientation des bits en (5.9) qui correspond au bloc *DTXY*. Encore une fois, on remarque que la réalisation pratique se résume à des portes logiques de base ce qui donne une intégration qui est rapide et ne consomme pas beaucoup de surface.

$$\begin{aligned}
A' &= B' = C' = 0\\
D' &= z5\\
E' &= \overline{z1 + z5}\\
F' &= z3 + z4 + Fz1\\
G' &= Gz1 + Fz2 + Ez3 + z4\\
H' &= Hz1 + Gz2 + Fz3 + z4\\
I' &= Iz1 + Hz2 + Gz3 + Cz4(EF + D)\\
J' &= Jz1 + Iz2 + Hz3 + z4(E \oplus F)\\
K' &= Kz1 + Jz2 + Iz3 + \overline{F}z4\\
L' &= Lz1 + Kz2 + Jz3 + Gz4\\
M' &= Mz1 + Lz2 + Kz3 + Hz4\\
N' &= Nz1 + Mz2 + Lz3 + Iz4\\
O' &= Oz1 + Nz2 + Mz3 + Kz4\\
P' &= Pz1 + Oz2 + Nz3 + Lz4
\end{aligned} \tag{5.9}$$

5.2.4 Dérivée de la fonction d'activation ($f_{PL}'(x)$)

La dérivée de $tanh(x)$ est donnée par l'équation (5.10). Si on considère que cette approximation $f_{PL}(x)$ est bonne, alors sa dérivée peut être calculée tel que montré par l'équation (5.11).

$$\tanh'(x) = 1 - \tanh^2(x) \tag{5.10}$$

$$f_{PL}'(x) \cong 1 - f_{PL}^2(x) \tag{5.11}$$

Pour réaliser cette dérivée, nous avons utilisé un multiplieur et un additionneur comme nous le montre la Figure 5.10.

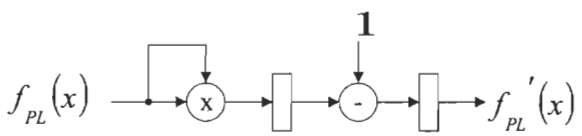


Figure 5.10: Réalisation de la dérivée de $f_{PL}(x)$

Il est bon de noter ici que la dérivée présente un délai de calcul de 2 cycles d'horloge ce qui a été pris en compte dans l'architecture globale de la Figure 5.4. En effet, les deux registres présents à la sortie du calcul de la dérivée sont redistribués pour réaliser la dérivée de la Figure 5.10. Enfin, la Figure 5.11 nous montre le résultat du code *VHDL* de $f_{PL}'(x)$ et $tanh'(x)$ alors que la Figure 5.12 nous montre l'erreur d'approximation.

La courbe de $f_{PL}'(x)$ donne une bonne approximation pour x inférieur à 1. Par contre, lorsque x grandit, nous avons une perte de précision car l'erreur est plus grande. Il aurait été intéressant d'appliquer la technique utilisée pour réaliser $f_{PL}(x)$ sur $f_{PL}'(x)$ afin de mieux contrôler les paramètres de la dérivée. Dans la présente application, nous n'en avons pas jugé la nécessité étant donné la présence du saturateur à la sortie du réseau. Si les valeurs en amplitude des échantillons étaient très importantes, alors il serait justifié d'appliquer la méthode sur $f_{PL}'(x)$.

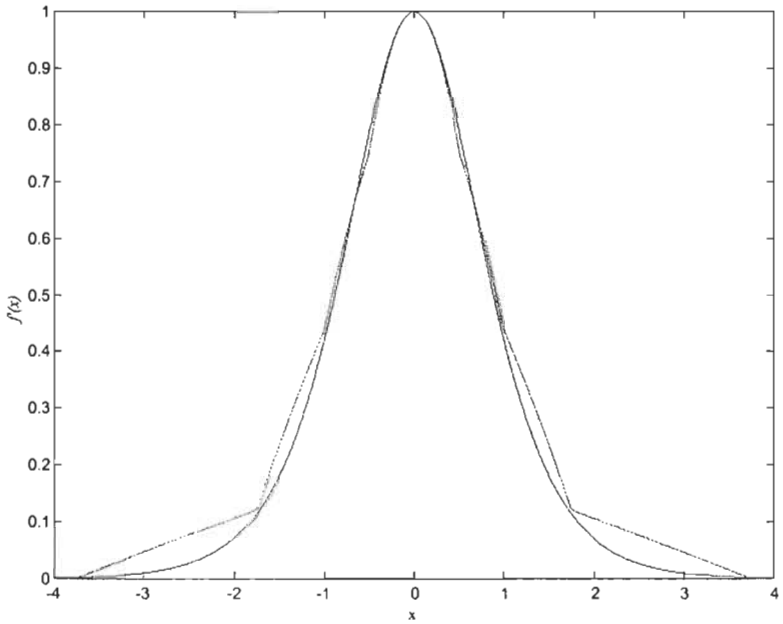


Figure 5.11: Dérivée de tanh(x) et $f_{PL}'(x)$

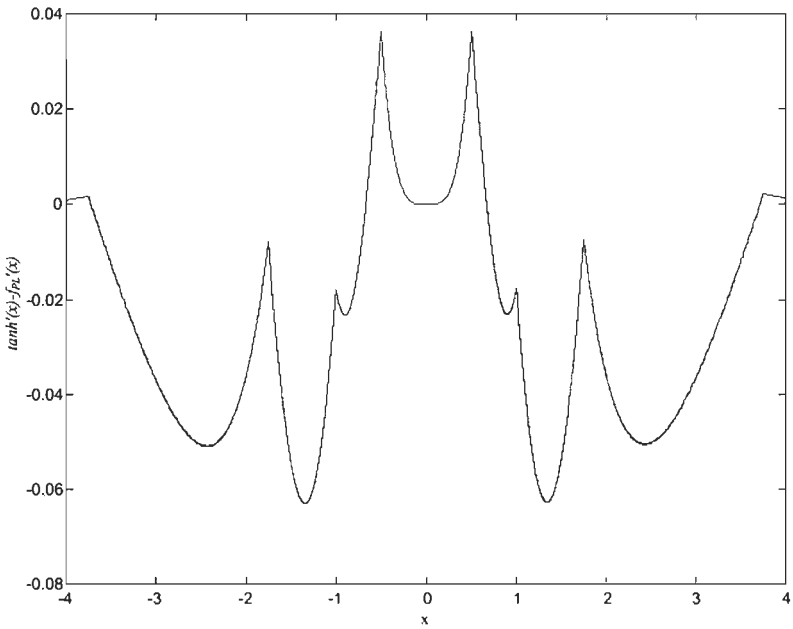


Figure 5.12: Erreur d'approximation de $f'_{PL}(x)$

Nous voyons ici que la réalisation des éléments de calculs est une étape importante pour l'intégration numérique du réseau de neurones. L'application de l'approximation linéaire par morceaux permet de réaliser la fonction d'activation avec une bonne précision, donnant un circuit simple qui sera rapide et utilisera peu d'espace.

5.3 Résultats de simulations

Dans cette section, nous verrons en premier lieu les paramètres d'intégration et de simulation pour les courbes de convergence et d'égalisation du réseau. Nous présenterons ensuite les résultats de simulation du code *VHDL* dans l'application à l'égalisation des

canaux de communications pour le canal *C1* linéaire et non linéaire. Enfin, nous verrons le résultat de la synthèse du réseau dans *Synopsys*™.

Les paramètres d'intégration, de simulations pour la convergence et d'égalisation sont :

- Technologie d'intégration CMOS de 0.18µm (CMOSP18) fourni par la *CMC*.
- Nombre de données pour l'apprentissage = 2000
- Nombre de répétition pour la moyenne du *MSE* = 200
- Rapport signal sur bruit (*SNR*) pour la moyenne du MSE = 35dB
- Nombre de données pour l'égalisation = 5000
- Nombre de répétition pour la moyenne de l'égalisation = 20
- Canal de communication de type linéaire et non linéaire (*C1*) avec bruit gaussien additif

5.3.1 Propriétés de convergence

Nous avons donc simulé le réseau complet et nous avons observé l'erreur quadratique moyenne entre les données désirées et les données estimées. La Figure 5.13 montre la convergence pour le canal *C1* linéaire (équations (2.4) et (4.23)) pour trois cas : simulation dans *Matlab*™ avec la fonction $f_{PL}(x)$, simulation dans *Matlab*™ avec la fonction $tanh(x)$ et simulation du code *VHDL* avec $f_{PL}(x)$. On remarque que les courbes sont presque toutes identiques. Il est bon de noter que la courbe où on utilise $tanh(x)$ donne un *MSE* résiduel plus faible que les autres. Ce résultat est causé par le fait que la dérivée de $tanh(x)$ n'est pas nulle pour des valeurs supérieures à 3.75 ce qui est par contre le cas pour $f_{PL}'(x)$. Alors, lorsque nous sommes près de la solution (près de la saturation), il n'y a plus de mise à jour des poids de la couche cachée avec $f_{PL}'(x)$ d'où le *MSE* se stabilise et cesse de diminuer.

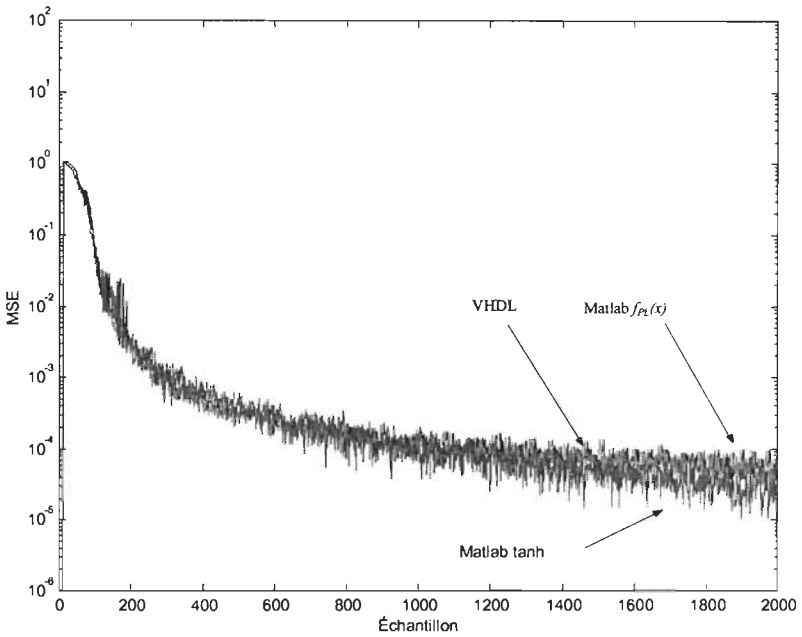


Figure 5.13: Caractéristiques de convergence du réseau *VHDL* pour *C1* linéaire

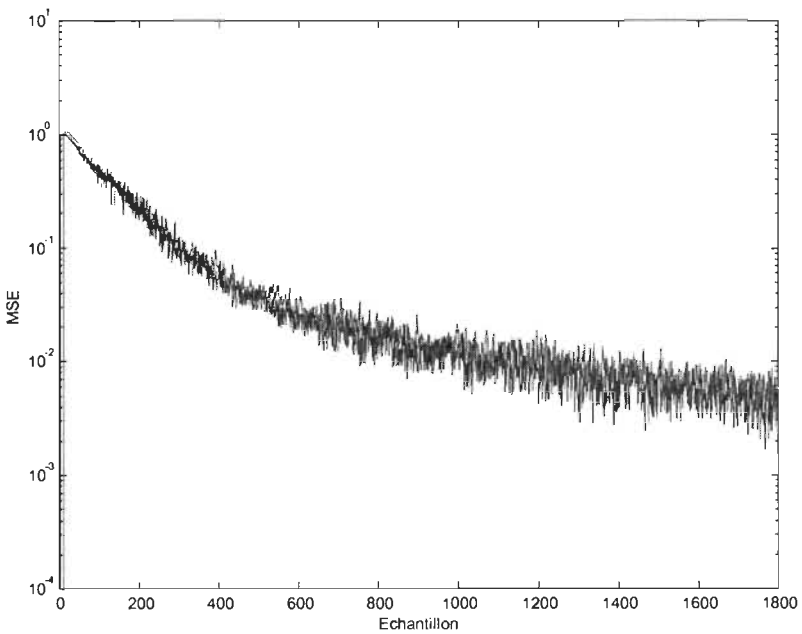


Figure 5.14: Caractéristiques de convergence du réseau *VHDL* pour *C1* non linéaire

Une solution proposée dans [AMI97] est de placer la valeur de la dérivée à une valeur ε pour (x>3.75) dans le but d'éviter de converger vers un minimum local mais plutôt vers un minimum global.

Pour le canal *C1* non linéaire (équations (4.23) et (4.24)), nous présentons le résultat à la Figure 5.14 pour les trois même cas : simulation dans *Matlab*™ avec la fonction $f_{PL}(x)$, simulation dans *Matlab*™ avec la fonction *tanh*(x) et simulation du code *VHDL* avec $f_{PL}(x)$. On remarque une convergence de l'algorithme ce qui démontre la capacité de *PIPRNA2* à traiter un canal non linéaire. Les trois courbes sont ici superposées ce qui nous montre que la réalisation en virgule fixe n'altère pas la convergence du *RNA*.

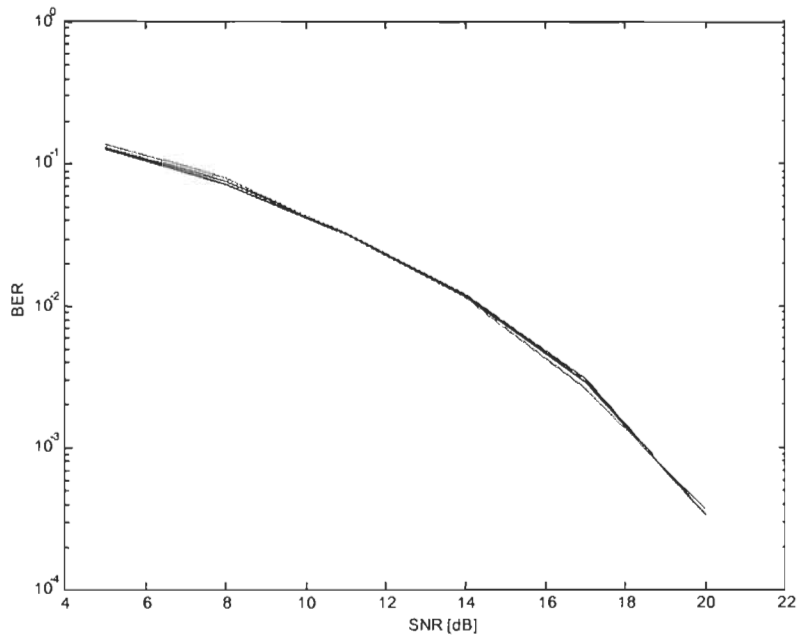


Figure 5.15: Performances en égalisation du réseau *VHDL* pour *C1* linéaire

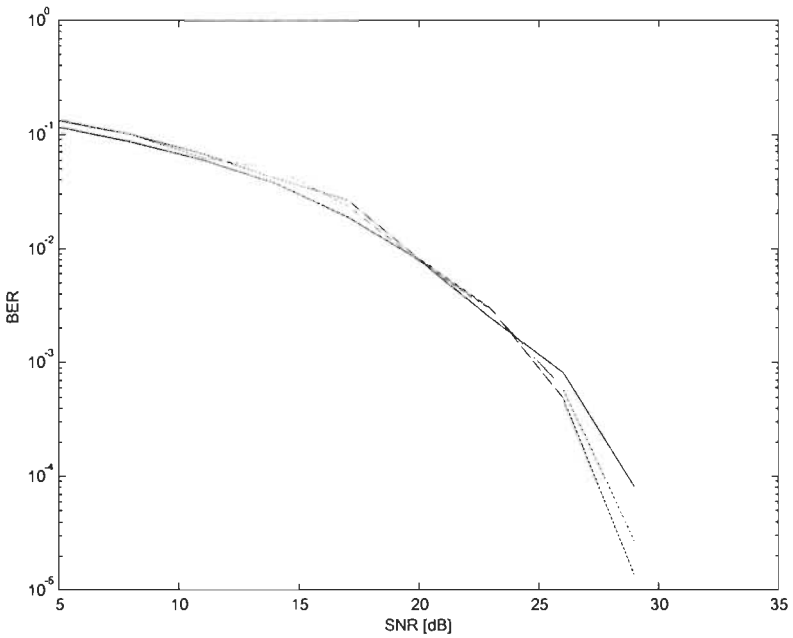


Figure 5.16: Performances en égalisation du réseau *VHDL* pour *C1* non linéaire

5.3.2 Performances en égalisation

Afin de valider le bon fonctionnement après l'adaptation des poids par le code *VHDL*, nous avons réalisé des simulations en égalisation. La Figure 5.15 montre les performances en égalisation des trois cas pour *C1* linéaire. On remarque ici que les trois cas sont identiques et que l'implantation en *VHDL* n'a pas détruit les performances en égalisation du réseau. Pour *C1* non linéaire, nous avons le résultat à la Figure 5.16 et nous remarquons que l'implantation *VHDL* a conservé ses propriétés en égalisation.

5.3.3 Résultats de synthèse

L'outil de synthèse utilisé est *Synopsys*™ et les caractéristiques qui nous intéressent sont la vitesse d'exécution (chemin critique) et la surface d'intégration. La surface donnée par l'outil a été transformée en terme de porte NAND équivalente. Le Tableau 5.11 nous donne le rapport de synthèse de *Synopsys*™.

Tableau 5.11: Résultats de la synthèse du réseau complet

Élément	Chemin critique (incluant les plots)	Surface d'intégration (équivalence en terme de porte NAND)
Porte NAND	0.29ns	1
Soustraction	5.65ns	190
Addition	5.61ns	180
Addition avec pré chargement	5.61ns	259
Multiplication	13.92ns	1 750
$f_{PL}(x)$	10.63ns	354
$f_{PL}'(x)$	13.93ns	1 940
Réseau complet	14.43ns	74 510

Dans le résultat de synthèse, on remarque que la fonction d'activation utilise un peu plus d'espace qu'un additionneur mais beaucoup moins qu'un multiplieur ce qui dénote une intégration efficace. La surface utilisée par $f_{PL}'(x)$ correspond à l'addition de la surface d'un additionneur et d'un multiplieur ce qui était attendu. Pour le réseau complet, le nombre de porte NAND équivalente est assez élevé et il serait probablement possible de concevoir un circuit de contrôle qui permettrait d'utiliser le même composant pour calculer différentes parties du réseau. Par exemple, au lieu d'avoir 3 neurones à l'entrée, on pourrait utiliser un seul neurone qui ferait le travail 3 fois avant de donner la sortie. Le désavantage de cette réutilisation des composants est que le débit de calcul sera diminué car on réduit le parallélisme.

Le chemin critique du réseau complet correspond en fait au chemin critique de l'élément le plus lent du réseau additionné au temps de propagation des plots (pour la connections des fils au boîtier). On remarque ici que le multiplieur est l'élément le plus lent. La fonction $f_{PL}(x)$ possède un temps de calcul assez long à cause de la réalisation des compléments à 2 par des additionneurs 16 bits. Ces composants pourraient être optimisés pour opérer plus rapidement et permettre un calcul plus rapide de la fonction d'activation. De son côté, la fonction $f_{PL}'(x)$ possède un temps critique équivalent à celui du multiplieur. Si nous utilisions le réseau non pipeliné, nous aurions un chemin critique donné par:

$$\tau_{crit} = 6\tau_{mult} + 6\tau_{add} + 2\tau_{plan} = 138.44ns \tag{5.12}$$

Le facteur de vitesse entre l'architecture pipelinée et non pipelinée est alors de 9.6. Ceci est très intéressant car l'augmentation de surface du réseau pipeliné par rapport au réseau sériel se résume à quelques registres supplémentaires alors que la vitesse est considérablement augmentée.

5.4 Conclusion

Dans ce chapitre, nous avons tout d'abord présenté les architectures sérielles et pipelinées d'un *RNA*. Les architectures sont définies par les valeurs que prennent les différentes variables des équations. L'objectif était de fractionner tous les éléments de calcul de l'architecture sérielle de sorte à ce que le chemin critique corresponde à l'élément

de calcul le plus lent soit la multiplication. Nous avons aussi vu que la synchronisation des données est capitale pour le bon fonctionnement du réseau. Par la suite, nous avons présenté les différents éléments de calculs et expliqué leur fonctionnement. Une des grandes difficultés a été d'appliquer la méthode de resynchronisation pour l'insertion des registres à travers l'architecture.

Nous avons aussi souligné que l'application de la technique d'approximation par morceau linéaire aurait aussi pu s'appliquer à la réalisation de $f_{PL}'(x)$ mais puisque les données traitées dans cette application sont d'amplitude discrète (± 1), il n'était pas nécessaire d'avoir une très grande précision sur l'amplitude de sortie du réseau. Enfin, nous avons vu, par des résultats de simulation, que l'intégration du réseau de neurone ne change pas ses caractéristiques au niveau de la convergence ni de ses qualités d'égaliseur et ce, même en présence de non linéarité.

Conclusion

Dans ce rapport, nous avons en premier lieu présenté le concept général des systèmes de communications. Il a été retenu que les systèmes de communications se tournent de plus en plus vers l'approche numérique. Cette approche permet d'atteindre de meilleures performances au niveau de la flexibilité, de la fiabilité et du coût.

Il a ensuite été question des différentes techniques de modulation des signaux numériques. Les différentes méthodes pour moduler l'information numérique repose sur les trois principaux paramètres qui constituent l'onde porteuse soit la phase, l'amplitude et la fréquence. Il a été montré que la modulation numérique permet de regrouper plusieurs bits du message original en différents symboles pour utiliser de façon optimale la bande passante disponible. Enfin, la nécessité d'appliquer la modulation pour transposer en haute fréquence les signaux en bande de base pour les communications sans fil a été démontrée par un exemple.

Par la suite, nous avons abordé le canal de transmission. Nous avons aussi souligné que nous ne pouvons interagir sur les paramètres de ce bloc. Les différents types de canaux de transmissions peuvent être modélisés de façon mathématique pour évaluer les performances des systèmes d'égalisation. On retrouve des canaux linéaires et non linéaires et les paramètres du canal peuvent être variants ou invariants dans le temps. Le modèle complet du canal de transmission à été présenté avec la source de bruit additif.

Suite à la présentation du canal de transmission, la problématique entourant l'égalisation des canaux à été présentée. Nous avons alors vu que pour récupérer la séquence de données originales, il est nécessaire d'utiliser un égaliseur pour éliminer l'effet du canal de transmission. Le choix du type d'égaliseur est fonction du type de canal en cause. Les deux grands types d'égaliseurs sont les optimums et les sous optimums. L'un a besoin de connaître a priori les caractéristiques et l'autre utilise une technique d'adaptation pour le calcul de ses paramètres. Ce calcul peut de plus se faire de façon supervisée ou autodidacte. Enfin, nous avons vu les trois grandes familles d'égaliseurs qui sont les filtres transverses, les *DFE* et les *MLSE*.

Pour adapter les poids des égaliseurs, il existe plusieurs algorithmes et les plus importants ont été présentés. Il est bon de noter qu'ils sont presque tous de nature récursive linéaire. On retrouve principalement dans la littérature le *LMS*, le *RLS* et le filtre de Kalman. Les différentes caractéristiques de calcul des algorithmes ont été résumées dans un tableau récapitulatif.

Nous avons ensuite vu les différentes techniques de pipeline pour augmenter le débit de calcul des architectures. La technique de la resynchronisation a été abordée pour montrer comment redistribuer les registres introduits par le pipeline. Un exemple de pipeline pour un algorithme non récursif a été présenté et il a été montré que la technique classique ne s'applique pas aux algorithmes de type récursifs. Nous avons alors présenté la technique de l'anticipation pour nous permettre de pipeliner les algorithmes de type récursif. Cette technique possède d'ailleurs quelques inconvénients qui sont l'augmentation de surface lorsque l'ordre du filtre et la profondeur de pipeline sont élevés. Pour pallier ce problème, nous avons eu recours à des techniques de transformation d'algorithme. Ces différentes techniques permettent d'approximer l'équation originale obtenue par anticipation pour réduire la surface utilisée de l'algorithme pipeliné. La technique de l'anticipation relaxée s'avère être très efficace pour ce genre de transformation d'algorithmes. Elle donne lieue à de multiples architectures que l'on peut optimiser selon les performances requises par l'application. Nous avons terminé la section du pipeline par un exemple d'application de l'anticipation relaxée appliquée à un filtre transverse adapté par l'algorithme *LMS*. Ce filtre était destiné à l'égalisation des canaux.

Nous avons ensuite présenté les réseaux de neurones à titre d'égaliseur et nous avons écrit leurs équations générales. L'attrait des réseaux de neurone est leur nature adaptative et leur capacité à traiter des problèmes non linéaires. Leur grand désavantage est de ne pouvoir opérer à des très grands débits étant donné leur complexité de calcul. En élaborant les équations de la rétropropagation du gradient, nous avons remarqué que la mise à jour des poids consiste en une série d'équations linéaires récursives qui se prête très bien au

pipeline par la technique de l'anticipation relaxée. Une série de simulations a démontré que la transformation de l'algorithme de rétropropagation du gradient pouvait être pipeliné et ainsi permettre de fonctionner à de plus grands débits. Les caractéristiques de convergence sont toutefois altérées mais les performances en tant qu'égaliseur sont demeurées les mêmes.

Par la suite, nous avons proposé une architecture de calcul hautement parallèle et pipeliné fonctionnant à un débit presque 10 fois supérieur comparativement à l'algorithme original. Cette architecture en virgule fixe a été traduite en code *VHDL* et simulée avec l'outil *Mentor Graphics*™. Les simulations ont confirmé le bon fonctionnement de l'architecture et nous avons vu que la quantification en virgule fixe n'a pas eu beaucoup d'effet sur les résultats. Une synthèse avec l'outil *Synopsys*™ a permis d'évaluer la surface d'utilisation du réseau de neurone complet. Nous avons remarqué que la surface utilisée est tout de même importante et il serait possible de modifier l'architecture pour réutiliser le même neurone pour faire tous les calculs. Par contre, cette approche diminue le débit de calcul ce qui n'est pas toujours souhaitable.

Bibliographie

[AMI97] Amin, H., Curtis, K. M., Hayes-Gill, B. R., Piecewise Linear Approximation Applied to Nonlinear Function of a Neural Network, *IEEE Proc.-Circuits Devices Syst.*, Vol. 144, No. 6, Decembre 1997.

[DUF01] Dufour, T., Intégration VLSI d'un réseau de neurone pipeline par la technique de l'anticipation relaxée, R *Laboratoire des Signaux et Systèmes*, UQTR, 2001, 101p.

[HAT92] Hatamian, M, Parhi, K. K., An 85 MHz 4^{th} order programmable IIR digital filter chip, *IEEE journal of solid state circuit*, pp. 175-183, Février 1992.

[HAY96] Haykin, S., Adaptive Filter Theory, *Prentice-Hall*, 1996.

[HER92] Herzberg, H., Haimi-Cohen, R. and Be'ery, Y., A systolic array realization of an LMS adaptive filter and the effects of delayed adaptation, *IEEE Trans. Signal Processing,* vol. 40, pp. 2799–2803, Novembre 1992.

[HER94] Hérault, J., Jutten, C., Réseaux neuronaux et traitement du signal, *Hermes*, 1994, 313p.

[MAC98] Macchi, O., L'égalisation numérique en communication, *CNRS-SUPELEC*, Ann. Télécommunications, Vol 53, No 1-2 1998.

[MAT99] Matsubara, K., Nishikawa, K., Kiya, H., Pipelined LMS adaptive filter using a new look-ahead transformation, *IEEE Trans. on Circuits and Systems*, vol 46, no 1, 1999.

[MUR92] Murtagh, P., Tsoi, A. C., Implementation Issues of Sigmoid Function and its Derivative for VLSI Digital Neural Network, *IEE Proc-E*, Vol. 139, No. 3, Mai 1997.

[LAN92] Lan, C.-P., Wen, S.-C., Jen, C.-W., Efficient synthesis and high-speed implementation of look-ahead recursive filters, *IEEE Conf. Acoust., Speech, Signal Processing,*vol. 4, pp. 305–308, 1992.

[LON89] Long, G., Ling, F., and Proakis, J. G., The LMS algorithm with delayed Coefficient adaptation, *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 37, pp. 1397–1405, Septembre 1989.

[MEY93] Meyer, M.D.; Agrawal, D.P., A high sampling rate delayed LMS filter architecture, *IEEE Trans. on Circuits and Systems – II*, vol 40, no 11, 1993.

[PAR89] Parhi, K. K., Messerschmitt, D. G., Pipeline interleaving and parallelism in recursive digital filter – parts I-II, *IEEE Trans. Acoust., Speech, Signal Processing*, vol 37, no 7, pp. 1099-1134.

[PAR91] Parhi, K. K., Finite word effects in pipelined recursive filters, *IEEE Trans. on Signal Processing*, vol 39, no 6, pp. 1450-1454, 1991

[PRO94] Proakis, J. G. Salehi, M. , Communication systems Engineering, *Prentice-Hall*, 1994.

[PRO95] Proakis, J. G., Digital Communications, *McGraw-Hill*, 1995.

[SHA93] Shanbag, N. R., Parhi, K. K., Relaxed Look Ahead Pipelined LMS Adaptative Filters and their Application to ADPCM Coder, *IEEE Trans. on Circuits and Systems*, vol. 40 no 12 1993.

[SHA95] Shanbag, N. R., Parhi, K. K., Pipelined Adaptive DFE Architectures Using Relaxed Look-Ahead, *IEEE Trans. on Signal Processing* , vol. 43 no 6 1995.

[SHA98] Shanbag, N. R., Algorithm Transformation Techniques for Low-Power Wireless VLSI Systems Design, *International Journal of Wireless Information Networks.*, vol. 5 no 2 1998.

[SHA98a] Shanbag, N. R., Im, G., VLSI Design of 51.84Mb/s Transceivers for ATM-LAN and Broadband Access., *IEEE Trans. on Signal Processing*, vol. 46 no 5 1998.

[SHA99] Shaw, A. K., Ahmed, M. I., Pipelined recursive digital filters: a general look-ahead scheme and optimal approximation, *IEEE Trans. on Circuits and Systems – II*, vol 46, no 11, 1999.

[STR90] Stremler, F. G., Introduction to communication systems, 3rd edition, *Addison-Wesley* 1990

[VID99] Vidal, M., Architecture systolique pour un algorithme basé sur les réseaux de neurones pour l'égalisation de canaux, *Mémoire de maîtrise de génie électrique*, UQTR, 1999.