UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

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

COMME EXIGENCE PARTIELLE
DE LA MAÎTRICE EN GÉNIE ÉLECTRIQUE

PAR
MARTIN VIDAL

ARCHITECTURE SYSTOLIQUE POUR UN ALGORITHME BASÉ SUR LES RÉSEAUX DE NEURONES POUR L'ÉGALISATION DE CANAUX

22 SEPTEMBRE 1999

« Aucune machine ne remplacera jamais ce qui est l'essence même de l'homme : l'esprit, la compassion, l'amour et la compréhension. »

- Louis V. Gerstner, président de IBM

Résumé

Ce mémoire propose un algorithme et architecture intégrée à très grande échelle (ITGE - *VLSI*) pour l'égalisation de canaux de communication numérique afin d'augmenter la qualité et la vitesse des communications.

Dans les communications numériques, les signaux sont envoyés sous forme de symboles par l'émetteur. La propagation des signaux sur un canal (ex: un réseau LAN, le milieu atmosphérique, le milieu acoustique sous-marine, etc.) engendre des interférences entre les symboles, faussant ainsi l'information contenue dans le message à la réception. Ces erreurs introduites par le canal de communication sont corrigées à l'aide d'une technique nommée égalisation de canaux qui consiste à utiliser la sortie du canal et des informations *a priori* sur le canal afin de produire une estimation du symbole transmis.

Comme les caractéristiques du canal sont souvent inconnues, un algorithme adaptatif de reconstitution des données transmises est nécessaire. Dans le cadre de ce travail, nous considérons l'adaptation dite supervisée pour laquelle l'algorithme d'égalisation utilise des séquences de données connues du récepteur durant la phase d'adaptation. De plus, nous considérons des non linéarités dans le canal de transmission provenant par exemple du milieu de propagation, des amplificateurs ou des convertisseurs. Ces non linéarités rendent

inefficaces les méthodes adaptatives classiques telles que le filtre transversal utilisant les méthodes d'adaptation LMS et RLS.

Afin de réduire le temps d'adaptation tout en considérant les non-linéarités possibles du canal, nous proposons donc un algorithme basé sur un réseau de neurones artificiels utilisant une topologie multicouche avec une fonction de décision de type canonique linéaire par morceaux que nous nommons PL-MNN (*Piecewise Linear Multilayer Neural Network*). Des simulations avec des données synthétiques ont été réalisées dans l'environnement Matlab® pour des communications utilisant des modulations en amplitude (PAM) et par déphasage (QPSK). Ceci nous a permis de présenter une étude comparative complète de la méthode proposée par rapport à d'autres méthodes.

L'architecture proposée est de type systolique, profitant ainsi du parallélisme à grain fin, de la modularité, de la régularité, etc. Afin de maximiser les performances, nous proposons un multiplier/accumulateur à pipeline synchrone. La validation de l'architecture est faite grâce à un modèle VHDL dans l'environnement Mentor Graphics®. L'évaluation des performances en terme de temps de calcul (latence et débit) et surface (nombre de transistors) a été possible grâce à l'outil Synopsys® et à la technologie CMOS de 0.5 µm de la compagnie HP (CMOSIS5), tous deux fournis par la société canadienne de microélectronique (SCM - *CMC*).

Finalement, ce projet apporte une solution intéressante pour la problématique d'intégration sur silicium d'égaliseur de canaux numérique à haut débit. Son introduction dans les systèmes de communication numérique assure une transmission de données plus fiable et plus rapide. Il faut ajouter que l'architecture proposée peut être développée pour

obtenir une version pour l'égalisation dite non supervisée (*Blind equalization*) et une adaptation dirigée par la décision.

Remerciements

Je tiens tout d'abord à remercier le professeur Daniel Massicotte, le directeur de ce travail, dont le temps consacré à ses étudiants n'est jamais compté. Son jugement et ses connaissances m'ont permis d'acquérir des qualités inestimables en recherche, soit la rigueur scientifique, la ténacité, le perfectionnisme et le sens critique. Je tiens aussi à lui rendre hommage sous un aspect plus humain pour son soutien moral et sa confiance qu'il m'a toujours accordée.

La seconde mention ira à mes parents qui m'ont offert toutes les chances qui leur étaient possibles de me donner afin de réussir ce travail. L'éducation transmise, la patience, la compréhension et le dévouement sont tous des éléments honorables dont ils ont fait preuve depuis ma naissance et qui contribuent à mes études.

Je ne saurais passer sous le silence la contribution de ma copine Annie Carignan qui a été d'une compréhension exemplaire. Bien que ma présence auprès d'elle soit bien souvent inexistante, elle a cru en moi et en nous, elle mérite donc toute ma reconnaissance. Son support et ses conseils ont souvent été d'un grand secours.

Tables des Matières

Liste des figures

Liste des tableaux

Liste des abréviations et des symboles

Symboles

c	Erreur pondérée de correction
e	Erreur de correction
K	Gain d'adaptation
o	Sortie de la couche cachée
P	Matrice de corrélation
q	Poids du neurone de sortie
s	Symbole transmis
W	Coefficients à adapter
w	Poids de la couche cachée

x	Signal analogique transmis
y	Sortie du canal
z	Résultat linéaire du neurone de la couche cachée
σ^2	Variance du bruit additif
μ	Pas d'apprentissage
λ	Facteur d'oubli
η	Bruit additif du canal
δ	Erreur rétropropagée
$\tilde{\bullet}$	Signal bruité
$\bar{\bullet}$	Signal déformé linéairement
$\hat{\bullet}$	Estimé de
$\boldsymbol{x}$	Vecteur formé des élément $[x_1, x_2, x_3, ..., x_n]$ sont en caractères gras minuscules

Abbréviations

ADSL	Asymetric Digital Subscriber Line
ASIC	Application Specific Integrated Circuit

ATM	Asynchronous Transfer Mode
BER	Bit Error Rate
BLC	Bloc de Logique Combinatoire
CAO	Conception Assistée par Ordinateur
CD-ROM	Compact Disc – Read Only Memory
CSA	Carry Save Adder
DFE	Decision Feedback Equalizer
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
ITGE	Intégration à Très Grande Échelle
LAN	Local Area Network
LMS	Least Mean Squares
MAC	Multiplieur-Accumulateur
PAM	Pulse Amplitude Modulation
PCM	Pulse-Coded Modulation
PE	Processeur Élémentaire
PLL	Phase Locked Loop
QAM	Quadrature Amplitude Modulation

QPSK	Quadrature Phased Shift Keying
RCA	Ripple Carry Adder
RLS	Recurcive Mean Squares
RNA	Réseau de neurones artificiels
RNIS	Réseau Numérique d'Intégration de Service
VCO	Voltage Controled Oscillator
WSI	Wafer Scale Integration

Chapitre 1

Introduction

Depuis quelques années, les communications numériques occupent une place de choix, certains parlent même de l'ère des communications. Avec l'importance que prend ce champ technologique, la nécessité d'avoir des liaisons fiables et rapides n'est plus à démontrer. Qu'on parle de l'Internet ou des téléphones cellulaires, les transmissions doivent être sûres ce qui demande des débits de transmission élevés et sans erreur. Or, il vient un temps où on se frappe aux limites physiques des médiums de communication se traduisant, par exemple, par leur bande passante caractéristique.

Plusieurs techniques de compression ont été développées afin de pallier ces limites physiques. Les déphasages et les variations d'amplitude par rapport à une porteuse permettent de combiner plusieurs bits dans un seul symbole, compressant ainsi l'information afin d'augmenter la capacité du canal. Cependant, ces techniques deviennent plus sensibles au canal de communication et aux bruits environnants. Ainsi, elles peuvent s'avérer insuffisantes d'où la volonté d'augmenter directement le débit de transmission de

symbole, c'est alors que l'on fait appel à l'égalisation de canaux de communication [PRO95].

L'égalisation de canaux permet de corriger les effets du canal à partir de la sortie de celui-ci. Les algorithmes de traitements numériques des signaux qui sont aptes à répondre aux exigences de l'application doivent être adaptatifs. Ainsi, la correction des effets du canal peut être faite avec des connaissances minimales sur celui-ci. L'égalisation de canaux est aussi utilisée dans les systèmes de mesure, par exemple elle permet d'augmenter la densité de stockage sur les disques et bandes magnétiques et la vitesse de lecture des disques optiques (CD-ROM). Elle permet d'augmenter les vitesses de transfert dans les réseaux informatiques locaux (LAN) et d'augmenter la capacité de transmission des satellites, qui se traduit par des économies se chiffrant en millions de dollars. Les applications de l'égalisation de canaux sont nombreuses et elles se retrouvent dans plusieurs domaines [AYA98], [CHA95], [CHE95], [NAI97], [VEC99].

1.1 Problématique

Lors de la transmission de données numériques à des débits équivalents à la capacité idéale du canal de communication, des erreurs de transmission peuvent être introduites dues à des interférences inter-symboles. Lorsque la technique de modulation utilisée n'agit que sur l'amplitude de l'onde transmise (PAM : "Pulse Amplitude Modulation"), la technique (positif ou négatif) du seuillage pour classifier les symboles à la réception devient insuffisante à des débits élevés, alors une correction devient nécessaire. Si la technique de modulation agit en plus sur la phase de l'onde transmise, alors les erreurs de déphasage

doivent être traitées. Les techniques de corrections d'erreurs sont connues sous l'appellation d'égalisation de canaux.

Le problème d'égalisation de canaux se pose lors de la transmission d'un signal s, qui n'est pas directement accessible. Ce problème est fondamental en télécommunication et il est rencontré fréquemment. La reconstitution du signal transmis consiste à traiter la sortie du canal $\tilde{y}$, lié au signal transmis s de façon causative, afin d'évaluer ce dernier. Le signal résultant de cette opération, noté $\hat{s}$, correspond à l'estimation du signal transmis s et est appelé signal reconstitué. Des méthodes classiques linéaires tel que le filtre de Kalman ou l'algorithme RLS peuvent être appliquées pour calculer le signal $\hat{s}$ [HAY96]. Cependant, les non-linéarités introduites dans le canal rendent moins efficaces ces méthodes. Ces non-linéarités sont principalement dues aux saturations des convertisseurs et des amplificateurs. Ces effets sont d'autant plus perceptibles dans les communications satellites où les amplificateurs d'ondes fonctionnent en saturation fréquemment. Aussi, ces algorithmes sont lourds à implanter et offrent une faible robustesse face à la quantification des données.

Le fait de vouloir transmettre à des débits élevés amène des problèmes qui dépassent les algorithmes d'égalisation de canaux. La quantité de calculs que demande ces algorithmes fait en sorte qu'il est impossible de les implanter dans un processeur de traitement numérique de signaux (DSP : "Digital Signal Processor") commercial qui n'offre pas les performances nécessaires. Les débits de transmission dépassent souvent les 100 Mbps alors que les DSP ont une fréquence de fonctionnement de 40 MHz. La parallélisation sur plusieurs processeurs paraît une solution mais la recherche de miniaturisation des appareils

fait en sorte que ce type de système ne répond pas aux besoins. De plus, l'autonomie des appareils étant importante, la consommation des circuits doit être prise en compte. Ce qui amène à avoir de petits circuits performants : les processeurs dédiés (ASIC : "Application Specific Integrated Circuit").

1.2 Objectifs

Ce projet a pour principal objectif l'intégration à très grande échelle d'un égaliseur de canaux non linéaires. Autant que possible, l'algorithme doit accepter les techniques de pipeline pour obtenir un haut débit de calcul et il doit s'intégrer dans une faible surface pour avoir un design économique et une faible consommation. Pour la résolution du problème, ces objectifs peuvent être divisés en sous-objectifs :

1. Étude d'algorithmes d'égalisation de canaux selon les méthodes de traitement numérique des signaux : algorithme LMS, algorithme RLS, réseaux de neurones tous à base d'équations récurrentes;

2. Étude des architectures dédiées à l'égalisation de canaux plus spécifiquement celles basées sur les réseaux de neurones;

3. Développement et proposition d'un algorithme d'égalisation de canaux basée sur les réseaux de neurones en vue d'une intégration à très grande échelle (ITGE);

4. Développement et proposition d'une architecture ITGE pour l'égalisation de canaux.

1.3 Méthodologie

Afin de répondre aux objectifs fixés précédemment, le développement de l'algorithme doit se faire conjointement avec le développement de l'architecture [MAS95], pour ainsi obtenir des performances optimales lors de l'intégration en technologie ITGE. Pour ce faire, les réseaux de neurones artificiels (RNA) semblent la solution toute indiquée. Avec leur non-linéarité intrinsèque qui réside dans leur fonction d'activation, ils sont aptes à résoudre les problèmes non linéaires. Nous les utilisons souvent pour des cas de classifications qui sont similaires à la situation qui nous intéresse.

Dans la plupart des publications, la fonction non linéaire est de forme tangente hyperbolique ou sigmoïde [KEC94]. Cette forme s'implante facilement dans une architecture analogique puisqu'elle ressemble à la fonction de transfert d'un transistor. Toutefois, les circuits intégrés analogiques entraînent une diminution de la précision, une vulnérabilité face au bruit extérieur et un risque d'instabilité. Aussi, la conception est plus complexe et plus longue que pour les circuits numériques. Pour contrer ces inconvénients, ce projet de recherche se dirige plutôt vers les architectures numériques qui sont mieux adaptées à l'usager qui manipule des données binaires. Ce type d'implantation est stable, robuste au bruit et la précision peut être contrôlée par la longueur des mots binaires. Toutefois, l'implantation numérique de la fonction tangente hyperbolique demande soit une surface d'intégration élevée, soit un temps de calcul considérable [NOR95]. Puisqu'en

égalisation de canaux le résultat de correction est binaire, que ce soit pour la partie réelle ou imaginaire, l'application d'une régression linéaire sur la fonction non linéaire des neurones peut être réalisée sans affecter la qualité de reconstitution [LIU98]. Ainsi, une version numérique est possible, incluant une méthode d'apprentissage en ligne efficace rendant adaptative la correction.

Cependant, avec le nombre croissant d'applications pour ce type de traitement numérique, il serait inopportun de ne proposer qu'un seul circuit. Les RNA récurrents offre la meilleure qualité de reconstitution comme le propose [PAR97]. Cependant, ces récurrences rendent impossible l'application des techniques du pipeline afin d'augmenter les performances. Alors il peut être difficile de répondre aux besoins d'une application qui demanderait un grand débit de données. La proposition d'un RNA multicouche peut être faite pour éliminer ce problème de dépendances de données. Une architecture hautement parallèle (systolique) peut être proposée afin de réduire les délais dus aux interconnections. De plus, ce type d'algorithme et d'architecture permettra l'application des techniques de pipeline afin d'augmenter le débit de façon significative sans pour autant augmenter la surface.

Les études algorithmiques et architecturales seront réalisées pour des communications utilisant les modulations PAM et QPSK.

1.4 Organisation de ce mémoire

Tout d'abord, les bases de l'égalisation de canaux seront traitées au Chapitre 2. Plus spécifiquement en discutant des communications numériques à la section 2.1, des types

d'égalisations de canaux à la section 2.2, des algorithmes d'égalisation de canaux linéaires et non linéaires à la section 2.3 et enfin, des architectures dédiées à cette problématique à la section 2.4.

Ensuite, les RNA seront abordés au chapitre 3 en commençant par leurs fondements à la section 3.1. Puis, il sera question des RNA applicables à l'égalisation de canaux à la section 3.2 pour passer ensuite à l'algorithme proposé et un de ses cas particuliers à la section 3.3. Des résultats de simulations de numériques et des comparaisons de performances de différents algorithmes sont présentés à la section 3.4. Et enfin, la complexité de calcul des algorithmes comparés est évaluée à la section 3.5.

Le chapitre 4 porte sur les architectures parallèles. Tout d'abord, il est question des architectures parallèles sous différentes formes. L'architecture proposée, qui constitue la base de ce mémoire, est présentée à la section 4.2. La section 4.3 traite des unités arithmétiques nécessaires à l'architecture. Enfin, une évaluation des performances de l'architecture est présentée à la section 4.4 un utilisant une technologie CMOS de 0.5 μm.

Chapitre 2

Égalisation des canaux de communication numérique

Les communications numériques forment un domaine complexe. Ce projet se concentre sur un élément de ce vaste domaine qui est l'égalisation des canaux de communications. Cependant, les principes et systèmes de communication numérique doivent être connus afin d'arriver avec une méthode efficace et réaliste. Ce chapitre se veut donc une introduction aux communications numériques (section 2.1) en passant par les techniques de modulation, de démodulation, les modèles de canaux et la synchronisation entre l'émetteur et le récepteur pour être apte d'extraire l'information des signaux reçus. Par la suite, le problème d'égalisation de canaux sera présenté à la section 2.2.

L'égalisation de canaux, possède plusieurs volets. Tout d'abord, l'adaptation des égaliseurs de canaux se fait principalement de deux façons; soit en connaissant une séquence de données transmises ou soit en auto-égalisation. Aussi, plusieurs algorithmes

ont été proposés pour résoudre cette problématique en présence de canaux linéaires, non-linéaires et selon la modulation utilisée, alors les données transmises peuvent être modélisées en nombres complexes ou réelles. Quelques algorithmes sont présentés à la section 2.3.

Au niveau des réalisations en circuits ITGE, les propositions sont plus rares. Il est cependant important de connaître quelques architectures qui ont été présentées et qui peuvent servir de base de comparaison pour les circuits proposés. Pour terminer, un tour des applications possibles sera présenté à la section 2.4 en caractérisant chacune d'entre elles.

2.1 Communication numérique

Les systèmes de communications numériques sont formés généralement des mêmes éléments comme le montre la Figure 2.1. L'information à transmettre dépend de l'application. Que ce soit la voix ou un signal vidéo, le signal analogique doit être converti numériquement. Cette conversion analogique-numérique peut également contenir une certaine forme de codage comme utilise les modems PCM (Pulse-Code Modulation) et le réseau public de téléphone [AYA98] dont les convertisseurs emploient une règle mu. Ensuite le signal est codé afin de sécuriser la transmission pour être apte à détecter les erreurs de transmission à la réception. Ces encodages peuvent utiliser différentes techniques telle la redondance et la parité afin de déterminer avec exactitude la nature du signal transmis.

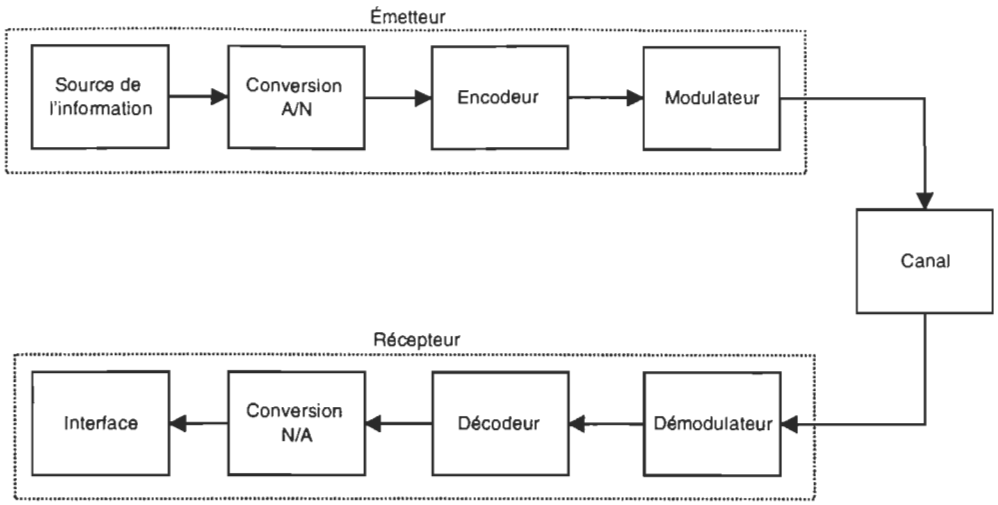


Figure 2.1 : Éléments de base d'un système de communication numérique.[PRO95]

Le modulateur convertit le signal numérique en signal analogique pour la transmission selon différentes techniques de modulation (section2.1.1). Le signal transmis contient des symboles représentés en amplitude et en phase dont chacun contient 2^m bits. On parle alors de transmission M-aire, par analogie avec les transmissions binaires.

Le canal constitue le médium de communication. Il peut s'agir d'un câble de cuivre, d'une fibre optique, de l'air ou de l'eau. Souvent on inclut les imperfections de l'émetteur et du récepteur dans la modélisation du canal ce qui peut entraîner une modélisation non-linéaire.

À la réception, le signal est démodulé afin d'extraire les bits contenus dans les symboles. Aussi, les effets du canal sont annulés le plus souvent après la démodulation. Cette correction est communément appelée l'égalisation de canaux. Par la suite, l'information est décodée et convertit analogiquement si l'application le demande. L'interface peut être alors un haut-parleur, une télévision etc.

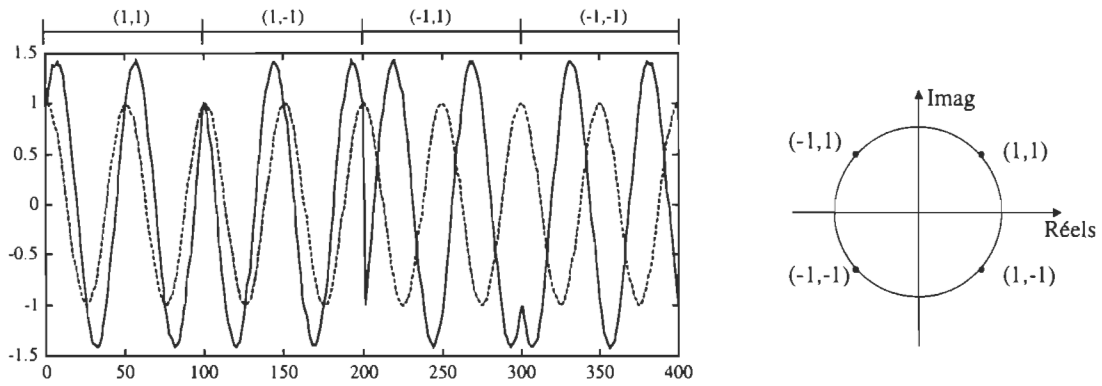
2.1.1 Techniques de modulation

Les différentes techniques de modulations permettent de compresser l'information à transmettre. Au lieu de transmettre l'information bit par bit, des symboles sont transférés représentant un nombre de bits 2^m. Le principe est d'attribuer une amplitude et une phase à chacun des symboles, ce qui lui attribue une position particulière dans le plan complexe. Une des modulations couramment utilisée est la modulation de phase en quadrature (QPSK : Quadrature Phase Shift Keying). Cette modulation n'utilise une amplitude et quatre phases pour former des symboles qui comportent deux bits. La Figure 2.2a montre le signal analogique transmis avec le signal de synchronisation alors que la Figure 2.2b représente les symboles dans le plan complexe.

Lorsque l'on considère plusieurs amplitudes et plusieurs phases, alors on parle de modulation d'amplitude en quadrature (QAM : Quadrature Amplitude Modulation). La nomenclature dépend du nombre de points de la constellation dans le plan complexe. Donc pour une constellation de 32 points, on parle de 32-QAM comme le montre la Figure 2.3 et chaque symbole représente 5 bits.

Les topologies de base des différents modulateurs M-aire sont semblables. Le schéma général est présenté à la Figure 2.4. Les bits à transmettre sont accumulés dans une pile et transmis simultanément au bloc de contrôle. Celui-ci détermine les amplitudes des parties imaginaires et réelles et les fournit aux convertisseurs numérique-analogique (N/A). Les tensions sont alors modulées avec la porteuse en phase et en quadrature pour donner naissance aux deux parties du nombre complexe à transmettre. Alors les deux parties peuvent être sommées pour former le signal à transmettre.

Cependant, le modulateur peut se compliquer lorsqu'on veut transmettre à haut débit tout en respectant la largeur de bande allouée. Aussi, le système peut être apte à transmettre avec plusieurs fréquences de porteuse [MYE98]. L'impédance de ligne doit également être adaptée, ce qui entraînerait des réflexions de l'onde à transmettre dans le circuit et des atténuations du signal transmis à cause de la différence d'impédance caractéristique.



a) b)

Figure 2.2 : Représentation (a) analogique avec le signal de synchronisation (en pointillé) et (b)dans le plan complexe des symboles QPSK.

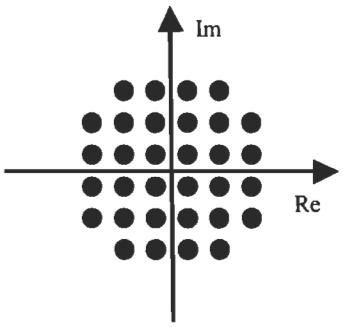


Figure 2.3 : Constellation 32-QAM.

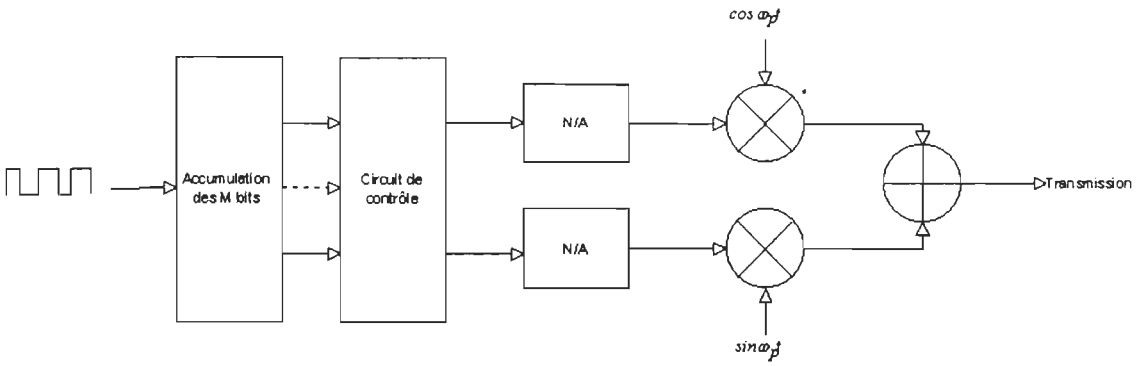


Figure 2.4 : Topologie générale d'un modulateur

2.1.2 Modèles de canaux de communication

Le canal de communication est ce que constitue le médium de transmission. Il peut s'agir de l'air, de l'eau, d'un câble coaxial, d'une paire de fils torsadés, d'une fibre optique ou d'un stockage de données (bande magnétique, disque magnétique ou optique). Chacun de ces canaux possède une largeur de bande dans laquelle la transmission peut être faite sans déformation majeure. Au-delà de cette largeur de bande, alors des erreurs peuvent survenir à la réception soit par erreur d'amplitude, de phase ou de polarité. Chaque canal peut cependant être représenté par le même schéma bloc, tel que montré à la Figure 2.5.

Les canaux de communications possèdent une partie linéaire qui reflète la bande passante du canal. Mathématiquement, ce bloc est représenté par un filtre passe-bas dont l'ordre et les coefficients $h(t)$ dépendent des caractéristiques fréquentielles du canal. Le signal transmis est donc convolué avec ce filtre causant des déformations et des pertes d'informations si aucune correction n'est faite à la réception.

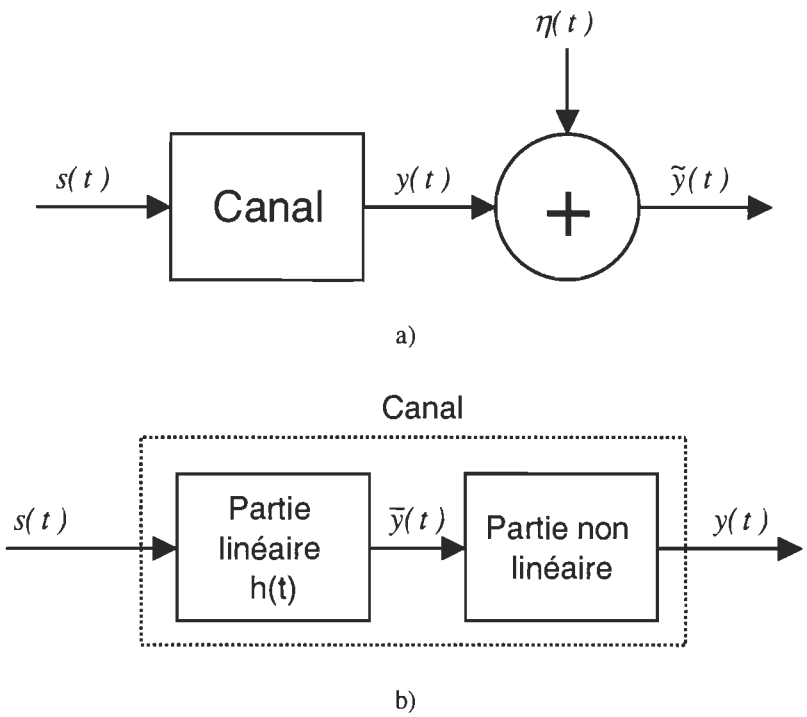


Figure 2.5 : Schéma bloc d'un canal de communication.

$$\bar{y}(t) = s(t) * h(t) \tag{2.1}$$

où * représente le produit de convolution

Dans la plupart des cas, ce filtre est quasi-stationnaire durant la transmission comme pour les fibres optiques ou les câbles. Cependant, le canal devient variant lorsque l'émetteur ou le récepteur sont en mouvement (ex: la communication mobile) ou lors des communications sous-marines [PRO95].

Cependant, les caractéristiques réelles des émetteurs et des récepteurs font qu'une partie non-linéaire vient s'ajouter comme le montre la Figure 2.5b. Dans le cas de

communications numériques, ces non-linéarités sont dues aux saturations fréquentes des amplificateurs et des convertisseurs, ainsi qu'à leur fonction de transferts. Pour le stockage de données, elles apparaissent lorsque la densité de stockage augmente [NAI97]. Lors de communication satellite, la situation est un peu différente. L'émetteur doit transmettre des signaux de grande puissance, ce qui a pour effet que les amplificateurs de l'émetteur saturent et introduisent des non-linéarités [CHA95]. On arrive donc à une expression de la forme :

$$y(t) = f(\bar{y}(t)) \tag{2.2}$$

où $f(\bullet)$ est la fonction non-linéaire.

Le bruit additif $\eta(t)$ qui est introduit est de nature aléatoire. Il peut être créé par le récepteur ou par le canal lui-même comme dans le cas de transmissions radio. Les composants électroniques sont souvent bruyants. Ce bruit est généralement modélisé comme un bruit thermique suivant une distribution gaussienne de moyenne nulle. Il vient donc l'équation globale du canal :

$$\tilde{y}(t) = f(\bar{y}(t)) + \eta(t) \tag{2.3}$$

2.1.3 Principes de la démodulation

Lors de la réception du signal, la première étape est de le démoduler pour en extraire l'information. Cette partie du récepteur comprend également l'égaliseur qui vient corriger les effets néfastes du canal. La Figure 2.6 montre de schéma bloc d'un démodulateur. Le

signal sortant du canal est multiplié par la porteuse et la quadrature de la porteuse, tous deux ayant une vitesse angulaire ω_p, afin d'en extraire la partie réelle et

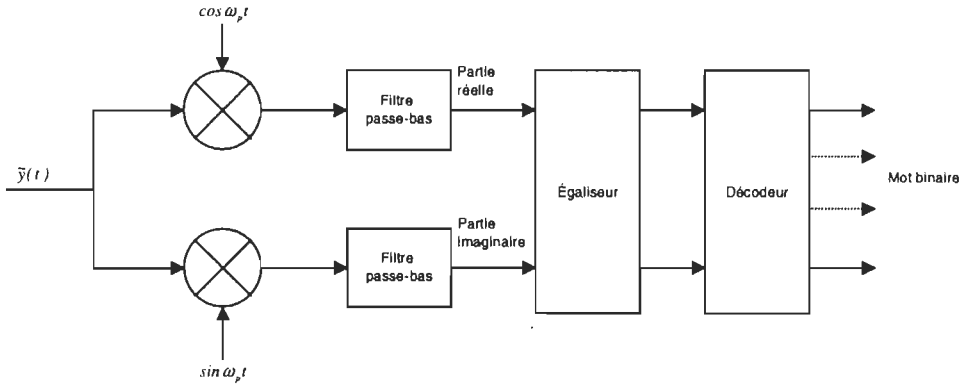


Figure 2.6 : Schéma bloc de la démodulation

imaginaire. Ensuite, les deux signaux sont filtrés afin d'éliminer les images en haute fréquence du signal. Les filtres passe-bas présente une difficulté de design quant à leur pente et à leur fréquence de coupure.

L'égalisation est un domaine de recherche en soi et sera approfondi dans la section 2.2. L'égaliseur peut être intégré dans un circuit numérique ou analogique. Dans le cas d'un égaliseur intégré numériquement, des convertisseurs analogique-numérique synchronisés sont nécessaires pour chacune des parties du nombre complexe. Le décodeur extrait l'information des amplitudes fournies par l'égaliseur. En fait, il est composé de deux détecteurs de niveaux.

2.1.4 Synchronisation émetteur-récepteur

À la réception, la porteuse, qu'on appelle également signal de synchronisation, doit être extraite pour être apte à démoduler le signal reçu. De façon générale, il est possible

d'extraire la porteuse du signal [PRO95] mais nous considérerons le cas où le signal de synchronisation est transmis avant la transmission des données. Pour effectuer cette synchronisation, des méthodes analogiques et numériques sont possibles.

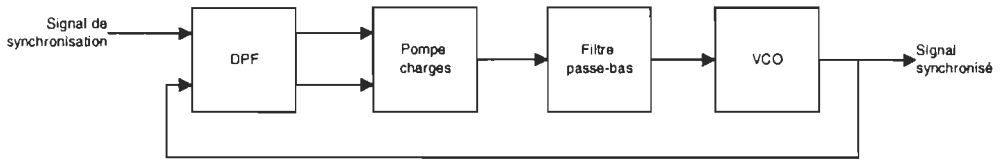
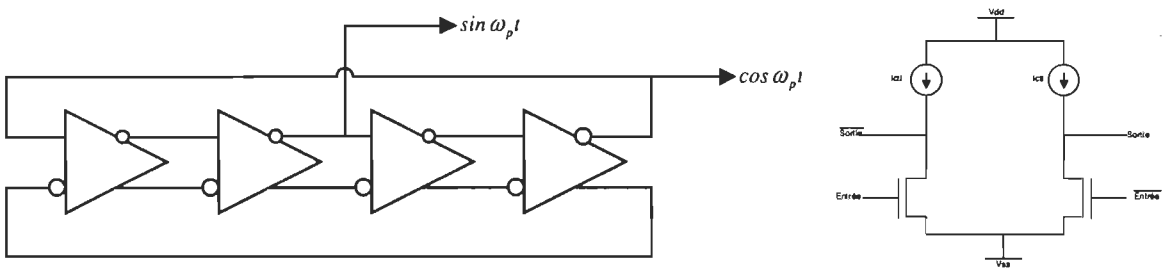


Figure 2.7 : Modèle du PLL.



a) b)

Figure 2.8 : Schéma du VCO différentiel (a) et de l'inverseur différentiel (b).

De façon analogique, la synchronisation peut se faire à l'aide d'une boucle à verrouillage de phase (PLL : Phase Locked Loop) telle que représentée à la Figure 2.7.

Pour bien comprendre le PLL, commençons par l'oscillateur contrôlé en tension (VCO : "Voltage Controled Oscillator"). Celui-ci produit une oscillation proportionnelle à la tension qui lui est appliquée à l'entrée. Le VCO est composé d'une série d'inverseur bouclée [JOH97]. La fréquence d'oscillation est contrôlée par le courant qui traverse les inverseurs. Si des inverseurs standards sont utilisés, alors le nombre d'inverseurs doit être impair. Cependant, si des inverseurs différentiels sont employés, alors il est possible d'avoir accès directement à la porteuse en quadrature en prenant un nombre pair

d'inverseur comme le montre la Figure 2.8a), alors que la Figure 2.8b) représente l'inverseur différentiel.

La tension de contrôle est fournie par le filtre passe-bas auquel on insert ou on retire des charges à l'aide du pompe-charge pour respectivement augmenter ou diminuer cette tension. Le détecteur de phase et de fréquence (DPF) permet de contrôler les charges et les décharges du filtre. Il peut s'agir du multiplieur analogique, d'une porte XOR, d'un circuit digital basé sur des bascules. La porte XOR, bien qu'il s'agisse de la plus simple des méthodes, a le désavantage de pouvoir s'accrocher à des harmoniques du signal de synchronisation.

Ce type de synchronisation est l'une des plus simples à mettre en œuvre. Une fois que le signal est synchronisé, c'est-à-dire lorsque la tension de contrôle ne varie plus, alors la sortie du pompe-charge tombe au troisième état pour ouvrir la boucle. Alors, la tension de contrôle ne varie plus. Cependant, l'inconvénient majeur de cette technique provient des résistances de fuite pouvant décharger lentement le filtre passe-bas, ce qui implique un rafraîchissement.

Une autre technique de synchronisation est la génération d'horloge par synthèse numérique directe (DDS : "Direct Digital Synthesis") [YIH96]. Cette méthode permet la génération de signaux d'horloge dont la phase et la fréquence sont très précises. Le principe est d'utiliser un phaseur dont on varie la vitesse angulaire comme le montre la Figure 2.9. Le contrôle s'effectue sur l'incrément de l'angle du phaseur. Après chaque incrémentation, la valeur du sinus de la phase calculé est évalué à partir d'une table de correspondance. Le signal d'horloge est obtenu à partir du signe du sinus. Il s'agit de la

version numérique du VCO dont la tension de contrôle a été remplacée par l'incrément du phaseur. Une fois le signal accroché sur le signal de synchronisation, la fréquence et la phase ne peuvent plus varier puisqu'elles sont basées sur un cristal dont la fréquence est d'au moins deux fois supérieure à la fréquence du signal de synchronisation.

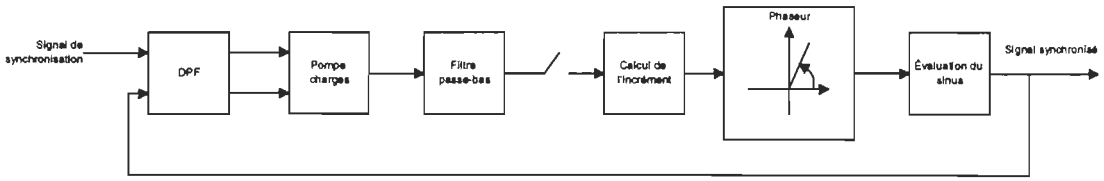


Figure 2.9 : PLL avec utilisant la génération d'horloge par synthèse numérique directe.

2.2 Égalisation de canaux

Il y a deux grandes catégories de récepteur : les optimums et les sous-optimums [PRO95]. La première catégorie nécessite à priori les caractéristiques du canal telles que la réponse impulsionnelle ou la réponse en fréquence. À l'aide de ces renseignements, le modèle inverse du canal est créé pour en annuler l'effet. Ces récepteurs sont très performants mais dans la grande majorité des cas, ces informations ne sont pas disponibles et de plus, les canaux varient dans le temps dans certains cas particuliers.

La seconde catégorie, quant à elle, estime le modèle inverse du canal. A l'aide de données transmises connues ou de leurs caractéristiques, l'algorithme d'égalisation s'adapte automatiquement au canal pour corriger les erreurs de transmission.

Dans cette section, il sera question des types d'adaptation, soit en connaissant des données transmises ou en auto-égalisation ("Blind Equalization"). Par la suite, nous traiterons des algorithmes linéaires et non-linéaires proposés dans la littérature pour passer

ensuite aux réalisations pratiques. Nous verrons finalement les applications qui ont déjà été traitées.

2.2.1 Principes de l'adaptation supervisée

L'adaptation se fait à partir de données transmises connues, on parle alors d'adaptation supervisée. À partir de l'erreur de reconstitution, les paramètres du filtre sont optimisés pour tendre vers un critère d'optimisation minimum. La Figure 2.10 montre le schéma d'adaptation.

Une période d'adaptation sera donc requise pour que les paramètres de l'égaliseur puissent s'adapter au canal. Par la suite, s'il y a une variation brusque du canal, une nouvelle adaptation sera nécessaire. La robustesse de l'égaliseur face aux changements de canal dépend de l'algorithme utilisé. Ces changements peuvent être dus aux changements de la réponse impulsionnelle ou du bruit additif.

La sortie de l'égaliseur $\hat{s}(t-d)$ est obtenu à partir de la sortie du filtre $\hat{x}(t)$ suivant la relation:

$$\hat{s}(t-d) = D(\hat{x}(t)) \qquad (2.4)$$

Où $D(\bullet)$ est la fonction de décision, telle qu'une fonction de Heaviside dans le cas de signaux binaires.

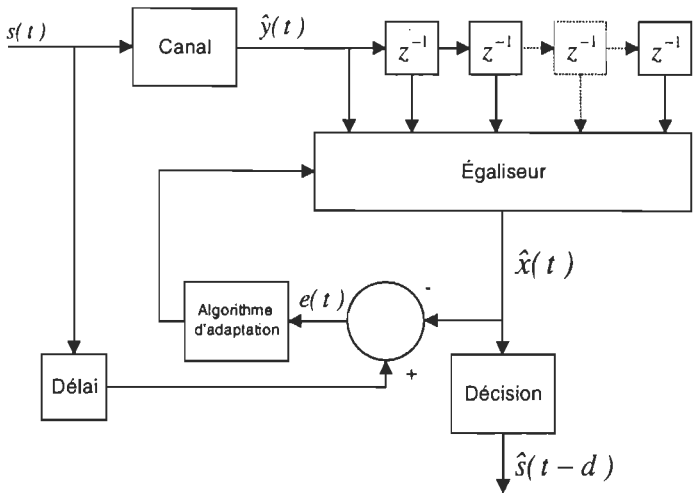


Figure 2.10 : Schéma d'adaptation d'un égaliseur de canaux.

2.2.2 *Principes de l'adaptation non supervisée*

L'auto-égalisation ("blind equalisation") permet d'adapter les paramètres du filtre sans avoir recours à une séquence connue par le récepteur. Il y a trois grandes catégories d'algorithmes pour l'auto-égalisation [PRO95]: le maximum de similitude ("maximum likehood" (ML)), le gradient stochastique et les statistiques d'ordre deux (ou quatre).

La plus populaire est probablement la méthode du gradient stochastique qui utilise un algorithme d'adaptation standard. La donnée de référence est fournie par une fonction non-linéaire comme le montre la Figure 2.11. Plusieurs fonctions non-linéaires ont été proposées mais les principales sont les algorithmes de Godard (ou CMA), de Sato, de Benveniste-Goursat et Stop-and-Go [PRO95].

Les méthodes statistiques d'ordre supérieur à deux cherche à estimer la réponse impulsionnelle du canal [PRO95]. Ceci est possible en utilisant l'auto-corrélation qui, face à un signal périodique, donne des informations sur l'amplitude et la phase du canal. Donc

en séparant l'égaliseur de l'estimateur du canal, on arrive à un auto-égaliseur performant qui demande cependant une grande quantité de calculs.

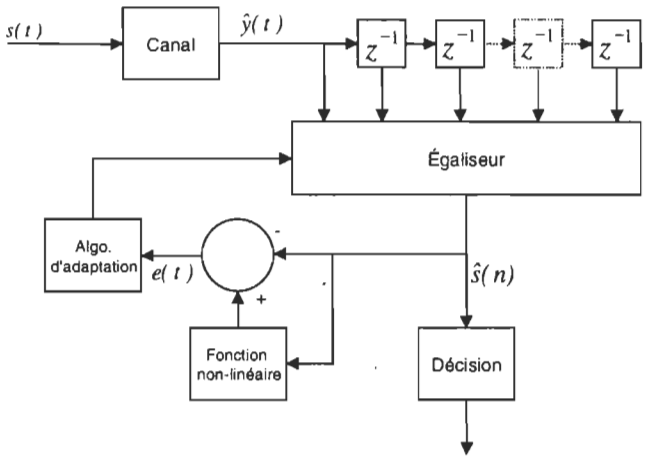


Figure 2.11: Schéma bloc de l'auto-égalisation basé sur le gradient stochastique.

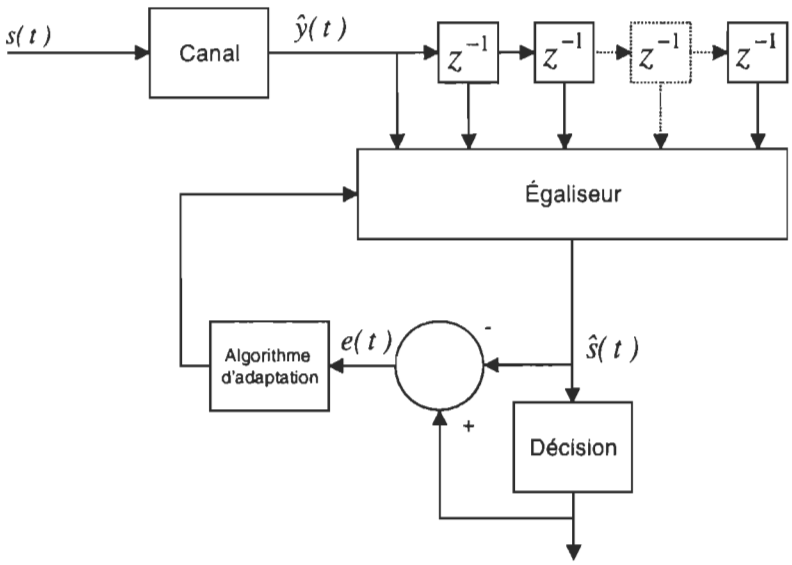


Figure 2.12 : L'adaptation dirigée par la décision.

2.2.3 Principe de l'adaptation dirigée par la décision

Lorsque l'adaptation du filtre a été complétée, il est possible de suivre la variation du canal à l'aide de l'adaptation dirigée par la décision. Cette technique consiste à utiliser la décision comme signal de référence pour continuer l'adaptation du filtre. Ainsi, lorsque l'erreur de filtre commence à augmenter, les coefficients sont mis à jour automatiquement. La Figure 2.12 montre cette technique.

2.3 Algorithmes pour l'égalisation de canaux

Les algorithmes pour l'égalisation de canaux sont nombreux. De façon générale, tous les filtres adaptatifs sont aptes à égaliser un canal de manière plus ou moins efficace. Ces algorithmes ne sont cependant pas tous aptes à égaliser les canaux non-linéaires. Un canal est linéaire lorsque $y(t) = \overline{y}(t)$ dans (2.2).

Dans cette section, nous ferons donc un survol des algorithmes les plus populaires pour l'égalisation de canaux linéaires et non-linéaires.

2.3.1 Égaliseur transversal

Les principaux algorithmes d'adaptation des filtres transversaux linéaires sont les méthodes LMS et RLS. Leur utilisation pour l'égalisation de canaux a été bien définie dans [HAY93]. Un sommaire de chacune de ces méthodes est contenu dans le Tableau 2.1.

Tableau 2.1 : Sommaire des méthodes classiques [HAY93]

LMS	
$\hat{x}(n) = \boldsymbol{w}^T \tilde{\boldsymbol{y}}(n)$	(2.5)
$e(n) = s(n-d) - \hat{x}(n)$	(2.6)
$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu \tilde{\boldsymbol{y}}(n) e(n)$	(2.7)
RLS	
$\hat{x}(n) = \boldsymbol{w}^T \tilde{\boldsymbol{y}}(n)$	(2.8)
$P(n) = \frac{1}{\lambda} P(n-1) - \frac{1}{\lambda} K(n) \tilde{\boldsymbol{y}}^T(n) P(n-1)$	(2.9)
$K(n) = \frac{1}{\lambda} * \frac{P(n-1)\tilde{\boldsymbol{y}}(n)}{1 + \frac{1}{\lambda} \tilde{\boldsymbol{y}}^T(n) P(n-1) \tilde{\boldsymbol{y}}(n)}$	(2.10)
$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + K(n) e(n)$	(2.11)
$e(n+1) = s(n-d) - \hat{x}(n)$	(2.12)

Les filtres transversaux sont reconnus pour leur efficacité sur des canaux linéaires. Cependant, ils deviennent facilement instables numériquement et ils ont un facteur à ajuster qui dépendent du canal à égaliser. L'algorithme LMS offre une faible complexité de calcul mais un choix mal adapté du pas d'apprentissage μ peut mener à une convergence très lente s'il est trop faible et une divergence de l'algorithme s'il est trop élevé. L'algorithme RLS, quant à lui, demande une grande complexité de calcul et nécessite une division, ce qui peut atténuer de façon considérable l'efficacité de l'intégration ITGE. Cependant, il converge rapidement et le facteur d'oubli λ a peu d'influence sur la convergence [HAY93].

2.3.2 *Égaliseurs non-linéaires*

Le principal égaliseur non-linéaire est basé sur l'algorithme à décision rétroactive (DFE : "Decision Feedback Equalizer")[PRO95]. L'optimisation des paramètres se fait avec les algorithmes linéaires tel RLS et LMS. Nous considérerons le DFE basé sur l'algorithme LMS.

La grande différence est au niveau de l'évaluation de la sortie de l'égaliseur.

$$\hat{x}(n) = \sum_{k=1}^{K} w_k v_k \tag{2.13}$$

$$\hat{s}(n) = D(\hat{x}(n)) \tag{2.14}$$

Où w sont les K coefficients du filtre et v, les entrées. La fonction D est une fonction non-linéaire de décision. Il peut s'agir par exemple de la fonction d'Heaviside dans le cas d'une transmission PAM. Le vecteur v est le vecteur d'entrée et est formé des éléments :

$$\begin{aligned} \boldsymbol{v}(n) &= [v_1, v_2, \ldots, v_K]^T \\ &= [\tilde{y}(n), \tilde{y}(n-1), \ldots, \tilde{y}(n-M), \hat{s}(n-1), \hat{s}(n-2), \ldots, \hat{s}(n-L)]^T \end{aligned} \tag{2.15}$$

$$e(n) = s(n) - \hat{x}(n) \tag{2.16}$$

Alors la mise à jour des coefficients peut se faire à partir de l'équation :

$$W(n+1) = W(n) + \mu \boldsymbol{v}(n) e(n) \tag{2.17}$$

2.4 Architecture pour l'égalisation de canaux

La plupart des intégrations d'égaliseurs de canaux a été fait dans une structure analogique. Une des architectures numériques qui a été repérée se retrouve dans [KEC91]. Celle-ci est basée sur un RNA entièrement connecté et peut accepter l'apprentissage en ligne. Elle est basée sur une structure en anneau. La fonction d'activation est de forme sigmoïde et son intégration est difficile. Une deuxième est proposée dans [CHE95] pour obtenir un débit de transmission de 125 Mbits/s sur une paire de fils torsadés sans gaine métallique.

Du côté analogique, un récepteur QPSK a été réalisé en utilisant les RNA pour l'égalisation de canaux et la démodulation. Dans [CHO93], un égaliseur de canaux basé sur un RNA multicouche composé de trois (3) couches cachées a été fabriqué dans une technologie de 2µm. Également, pour les communications à débit de transmission élevé, un égaliseur de canaux en mode courant est proposé dans [PAR95]. On y estime le débit de transmission à environ 100 méga-symboles par seconde.

Une architecture triangulaire pour le filtre STAR-RLS a été implantée dans une structure FPGA [BRA97]. Basé sur les rotations de Givens, le STAR-RLS est moins sensible aux effets de quantification que le RLS conventionnel. L'inconvénient de cette architecture de type systolique est que les coefficients sont calculés dans l'architecture et ils doivent être évacués, ce qui allonge le temps de calcul. Aussi, la complexité de cette architecture est considérable.

Pour des applications plus spécifiques d'égaliseurs dédiés, nous retrouvons les propositions suivantes. [SHA98] propose un circuit pour les réseaux ATM (Asynchronous Transfert Mode) et les LAN (Local Area Networks) permettant un débit de 51.4 Mbits/s. Une architecture utilisant la technique du pipeline est proposée pour un filtre linéaire employant un algorithme LMS. Dans [WAN95], on propose un circuit pour la transmission ADSL ("Asymmetric Digital Subscriber Line"). Ces deux derniers circuits utilisent une modulation en phase et amplitude sans porteuse (CAP : "Carrierless Amplitude and Phase) qui est la modulation utilisée pour les services ATM, RNIS (Réseau numérique d'intégration de service) et ADSL.

Un cas d'un circuit commercialisé par la compagnie Hitachi [HIT98] est un circuit complet effectuant la démodulation, l'égalisation et le décodage. Il a été conçu pour les modems-câble et offre un débit de transmission de 40 Mbits/s sur un canal dont la largeur de bande est 6 MHz en utilisant la modulation 256-QAM. L'égaliseur interne est cependant pour les canaux linéaires.

2.5 Conclusion

Les communications numériques comportent plusieurs aspects complexes dont la connaissance est nécessaire pour l'élaboration des éléments qui composent un système de transmission complet. Un de ces éléments est l'égalisation de canaux qui est nécessaire pour annuler les effets néfastes du médium de communication qui souvent est non-linéaire comme dans le cas de communications par satellite comme le montre les Figure 3.17 et 4.14. Les déformations créées se situent aussi bien au niveau de la phase, de l'amplitude que de la polarité.

L'égalisation de canaux ne se limite pas aux systèmes de communication mais peut également être utilisée pour améliorer les systèmes de stockage de données aussi bien optiques que magnétiques.

Les mises en œuvres pratiques ne sont pas courantes et elles ne couvrent souvent que des applications spécifiques pour des canaux linéaires. L'intégration numérique d'un égaliseur adaptatif pour les canaux non-linéaire est donc original et peut convenir à plusieurs domaines d'applications.

Chapitre 3

Réseaux de neurones pour l'égalisation de canaux

Depuis quelques années, les recherches sur l'égalisation de canaux se sont multipliées. Différentes méthodes avancées de traitements numériques ont été utilisées afin de proposer des solutions à cette problématique. Une technique qui est de plus en plus populaire consiste à tirer profit des algorithmes basés sur les réseaux de neurones artificiels (RNA). Les RNA offrent des caractéristiques intéressantes autant au niveau algorithmique qu'au niveau de leur mise en œuvre dans une technologie ITGE.

Dans ce chapitre, nous ferons tout d'abord un survol des fondements des RNA à la section 3.1 en citant les principaux types de neurones et de topologie de RNA. Ensuite, nous discuterons des propriétés des RNA qui les rendent intéressant pour résoudre cette problématique. Puis, les RNA qui ont déjà été proposés pour l'égalisation de canaux seront traités à la section 3.2. Enfin, à la section 3.3, nous passerons à la pièce maîtresse de ce mémoire, la structure linéaire par morceaux communément appelée « Piecewise Linear».

Nous en ferons la définition et la description. Finalement, nous en évaluerons les performances tant au niveau de la convergence que de la robustesse.

3.1 Principes de bases des RNA

Les RNA s'inspirent du cerveau humain. Les neurones qui constituent les centres de décisions sont reliés de façon à accomplir un travail bien défini. Chaque neurone reçoit des stimulus qui entraînent la prise d'une décision en relation avec le processus qui lui est propre. Les RNA constituent une méthode de programmation qui consiste à établir les paramètres de chaque neurone ainsi que les relations entre chacune d'elles. En établissant les paramètres optimums, nous pouvons faire l'approximation d'une fonction de façon précise. Les réseaux de neurones représentent une base algorithmique puissante pour le traitement de signaux grâce à ses propriétés (section 3.2), principalement la non-linéarité et le parallélisme intrinsèque à la méthode.

3.1.1 Le neurone formel

Tous les types de neurones ont le même principe de base. On associe chaque entrée à un poids w, c'est-à-dire un facteur multiplicatif. Pour garder l'analogie avec le cerveau humain, l'adaptation des poids du RNA est appelée apprentissage. Les produits sont ensuite sommés pour constituer la base du processus de décision. C'est ce que nous appellerons la somme des produits. Par la suite, le résultat passe dans une fonction de décision $g(\bullet)$ afin d'en évaluer sa valeur de sortie. La Figure 3.1 montre le neurone formel.

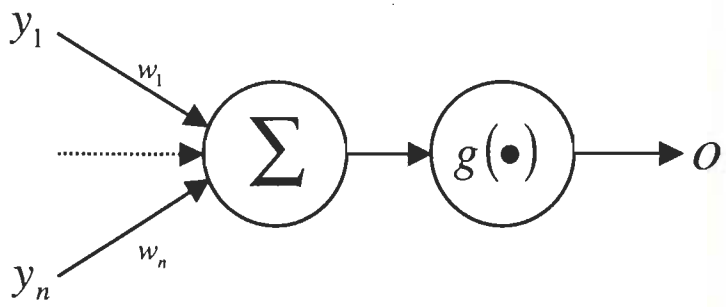


Figure 3.1 Le neurone formel

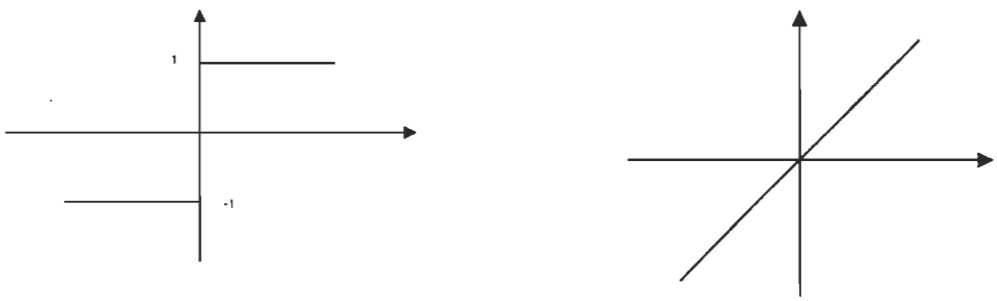
3.1.2 Topologies de RNA

Les RNA sont un ensemble de neurones reliés, maillés. Ils sont divisés en couches qui sont constituées de neurones de mêmes types. Des couches de différents types peuvent constituer le RNA pour plus de flexibilité afin de répondre à un plus grand nombre d'applications. La schématisation d'un RNA à une couche est représentée à la Figure 3.5a. Le nombre de couches change selon la complexité des données à apprendre, le nombre d'entrées et le nombre de sorties du RNA. La couche de transmission est celle qui transmet les entrées au RNA. Elle peut ou non avoir une influence sur la valeur de sortie. Il y a un neurone par entrée. La couche de sortie est celle qui produit la sortie du RNA. Il y a un neurone par sortie. Les couches cachées se situent entre la couche de transmission et la couche de sortie. On les appelle ainsi puisqu'elles sont invisibles de l'entrée et de la sortie du RNA.

Le premier modèle est appelé neurone à seuil. Ce neurone est le plus simple, car sa sortie est constituée d'un seul bit. La fonction de décision est la fonction unitaire. Si l'entrée de la fonction est inférieure à 0, la sortie prend la valeur « 0 ». Dans le cas contraire, la sortie prend la valeur « 1 » tel que montré à la Figure 3.2a.

Ce type de neurone est souvent utilisé pour des variables linéairement séparables. Des variables sont linéairement séparables si dans le plan, les « 0 » peuvent être séparés des « 1 » à l'aide d'une ou plusieurs droites. A la Figure 3.3, on retrouve sur les axes les deux entrées alors que les « o » et les « + » représentent la sortie (0 ou 1). On peut voir que les variables sont séparées à l'aide d'une seule droite donc le réseau a seulement un neurone à seuil. À cause de cette limitation, on a introduit les réseaux multicouches qui sont aptes à séparer des variables à l'aide de plusieurs droites.

Le second neurone est du type « Adaline » (ADAptive Linear Neuron) ou plus simplement neurone linéaire dont la fonction d'activation est présenté à la Figure 3.2b. La sortie du neurone est directement la somme des produits. La sortie est évidemment un nombre réel. Ce neurone est souvent retrouvé à la sortie d'un réseau Perceptron multicouche et utilise des neurones de forme sigmoïde sur la couche cachée.



a) b)

Figure 3.2 : La fonction d'activation du neurone à seuil (a) et du neurone linéaire (b)

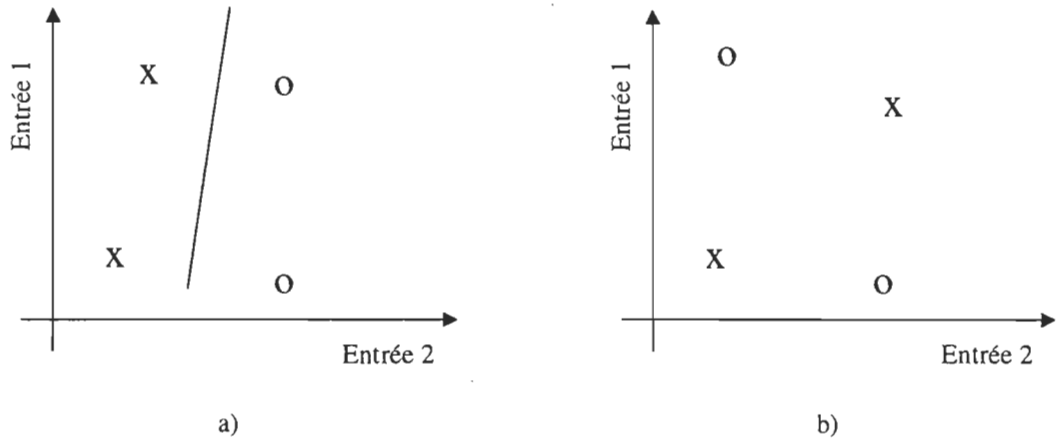


Figure 3.3 : Variables linéairement séparables (a) et linéairement non séparables

Le neurone de sigmoïde est le plus complexe de tous les neurones vus jusqu'à maintenant mais il permet l'apprentissage d'un éventail de fonction plus élaboré. La fonction de décision de celle-ci est la fonction tangente hyperbolique. Comme les entrées, les poids et le seuil sont des nombres réels, la sortie sera également un nombre réel qui varie entre 0 et 1.

La fonction de décision standard, sigmoïde ou tangente hyperbolique, est difficilement intégrable. Elle engendre soit une perte de précision ou une surface d'intégration non négligeable [NOR95]. La solution la plus simple est de créer une table de correspondance pour toutes les possibilités d'entrées. Il s'agit donc d'incorporer dans le processeur une mémoire qui contient toute l'information nécessaire pour faire le travail des couches de neurones contenant les fonctions d'activation non-linéaire de type sigmoïde. Évidemment, lorsque le nombre de bits augmente, la taille de la mémoire s'accroît rapidement ce qui fait que la mémoire occupe une surface significative.

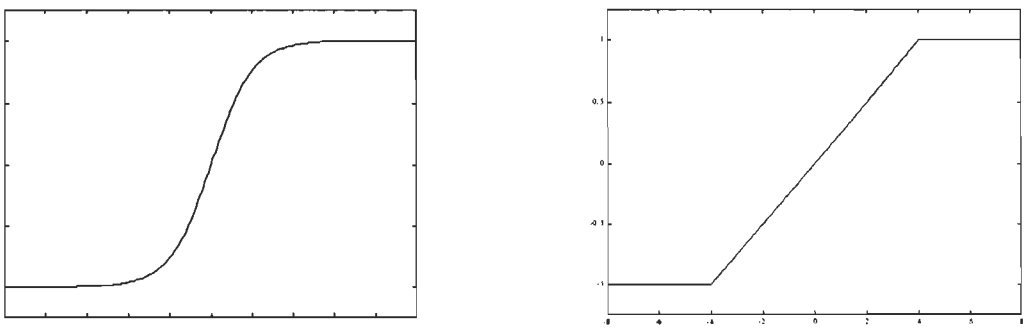
L'application d'une fonction linéaire par morceau permet d'implanter des approximations des fonctions sigmoïdes de façon compacte. La Figure 3.4 montre un exemple de deux fonctions d'activation.

Malgré que ces fonctions ont l'air bien différentes, une similitude fait que leur dynamique est équivalente [LIU98] pour le problème d'égalisation de canaux tel que l'effet de saturation dans l'intervalle $|x| > 4$.

Un type de fonctions d'activation qui est appliquée à l'égalisation de canaux est la « radial basis fonction ». Elle est une gaussienne mais les RNA utilisant cette fonction s'avèrent volumineux et complexe. Des versions sont à l'étude en vue d'une implantation en technologie ITGE.

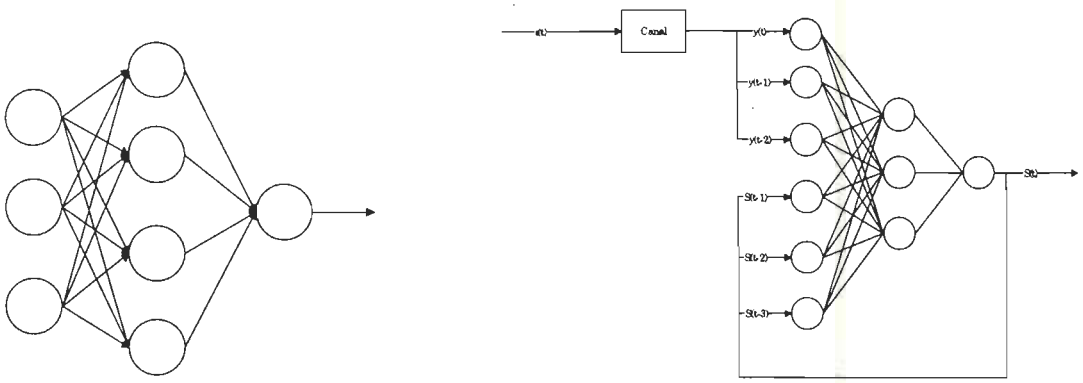
3.1.3 Propriétés des RNA

Les RNA ont deux propriétés qui les rendent très attrayants pour diverses applications. Tout d'abord, ils ont la capacité de généralisation, c'est-à-dire qu'ils peuvent interpoler entre les points de l'apprentissage pour fournir une réponse avec une grande précision. L'autre



a) b)

Figure 3.4 : Les fonctions d'activation, a) la sigmoïde et b) la fonction « piecewise linear»



a) b)

Figure 3.5 : Schématisation d'un RNA multicouche (a) et d'un RNA récursif (b) (NARX).

propriété provient de la non-linéarité intrinsèque due aux fonctions de décision. Il est donc avantageux d'utiliser des RN pour traiter les problèmes non-linéaires.

Une topologie de RNA qui tient compte de la dynamique des données est le modèle NARX tel que montré à la Figure 3.5b. Le RNA NARX s'inspire du RNA multicouche, présenté à la Figure 3.5a. Une rétroaction de la sortie du réseau est introduite ainsi qu'un nombre déterminé de retard sur celle-ci et sur les valeurs en entrée. Ainsi, on a la possibilité d'utiliser le modèle ARMA bien connue sous forme de RN. L'avantage que le RN a face à toutes les autres méthodes de ce type, c'est qu'il n'a aucune valeur à syntoniser mise à part le pas d'apprentissage. La connaissance du procédé n'est pas requise également.

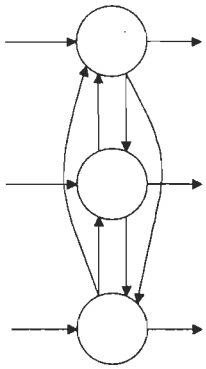


Figure 3.6 : Le réseau d'Hopfield.

Les RNA entièrement connectés tel le réseau d'Hopfield ont prouvé leur valeur face à un problème de classification. Les récurrences dans le réseau permettent une interpolation efficace. Par exemple, ce type de réseau peut servir de banque d'images. Lorsqu'on lui fournit une image incomplète, il est apte à retrouver l'image originale Il est à noter qu'il n'est pas nécessaire d'avoir une entrée et une sortie par neurone. Lorsque la fonction d'activation utilisée est une fonction seuil, alors il s'agit du réseau d'Hopfield tel que montré à la Figure 3.6.

La flexibilité obtenue avec de tels modèles permet une grande variété d'applications. Le réseau peut rester intact alors que nous varions l'apprentissage pour une autre affectation, que ce soit en traitement numérique des signaux, contrôle ou autre, la même méthodologie s'applique. Ici, nous évaluerons s'il est possible d'appliquer les RNA au problème d'égalisation de canaux.

3.2 RNA pour l'égalisation de canaux

Les topologies de RNA pour l'égalisation de canaux sont vraiment variées. Les connections, le nombre de couches cachées, les fonctions d'activation, toutes ces

modifications peuvent être apportées pour obtenir un nouveau RNA qui, selon les personnes qui l'ont proposé, est plus performant que toutes les autres propositions. Principalement, nous pouvons les classer en deux grandes catégories: les RNA multicouches et les RNA récurrents. La Figure 3.5 montre un exemple dans chacune des deux catégories.

3.2.1 Propriété des RNA pour l'égalisation de canaux

Le principal attrait des réseaux de neurones pour l'égalisation de canaux se trouve dans les propriétés de la fonction d'activation. Cette fonction est non-linéaire, ce qui agrandit le champ d'application des réseaux de neurones pour l'égalisation de canaux non-linéaires. La fonction d'activation a aussi une autre caractéristique fort intéressante : elle sature à –1 et +1. Alors, pour reconstituer un signal binaire formé de –1 et de +1, pour la partie réelle ou imaginaire, les réseaux de neurones sont particulièrement bien adaptés.

De plus, en vue d'une intégration, les RNA sont récursifs, implicitement parallèles et utilisent des communications locales et régulières, ce qui mène facilement vers une architecture systolique pour arriver à un haut débit de calcul.

3.2.2 RNA récursifs

Les RNA récurrents semblent avoir un intérêt plus particulier. Comme le mentionne [PAR97], les RNA récurrents offrent de meilleures performances que les RNA multicouches lorsque le canal comporte des annulations spectrales profondes. De plus cette équipe propose de n'appliquer que des récurrences sur la couche cachée du RNA, ce qui réduit de façon importante les dépendances de données comparativement à un RNA

entièrement connecté. D'autres équipes ont cependant opté pour un RNA entièrement connecté comme [LIU98] et [KEC94].

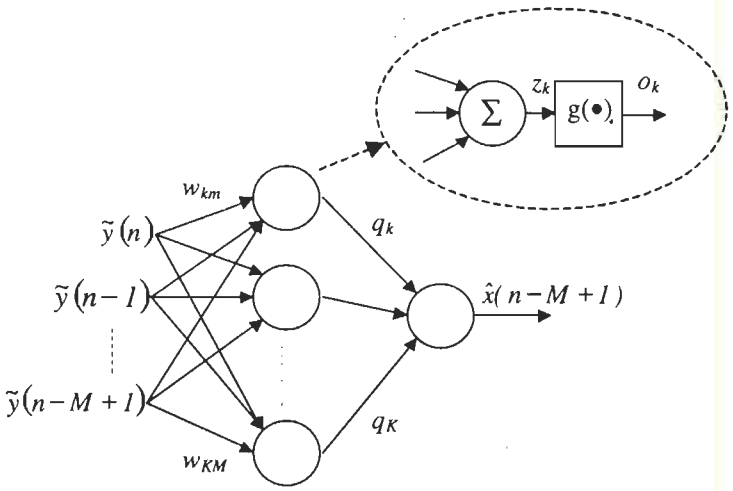
L'utilisation de RNA NARX a aussi été proposée dans [CHA95] et [CHA94] pour des applications spécifiques telles l'égalisation des canaux intérieurs de communication radio et pour les communications satellites qui sont non-linéaires. Ces RNA sont performants et convergent rapidement mais les dépendances dans les données des algorithmes réduisent l'efficacité de leur mise en œuvre dans une technologie ITGE.

3.2.3 RNA multicouches

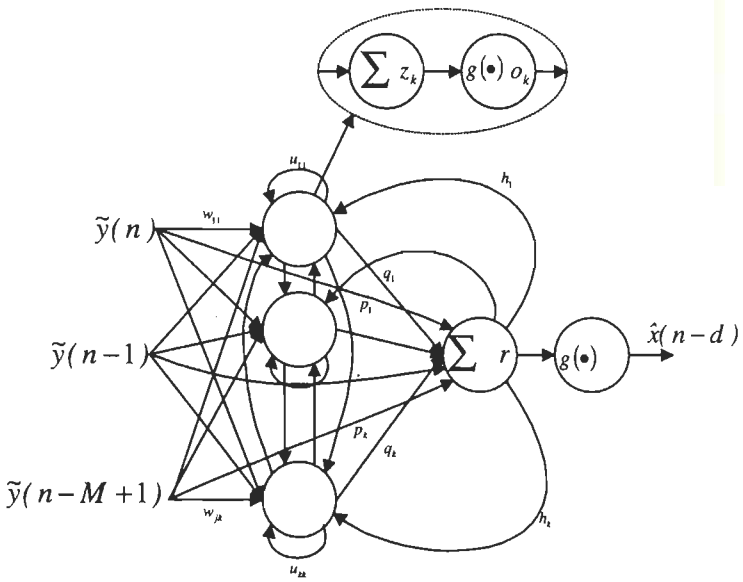
Plusieurs propositions de RNA multicouches ont été faites au cours des années. Étant une évolution du neurone formel de base, les RNA multicouches sont venus régler bien des problèmes de classifications et d'estimations. On a même dit qu'ils étaient des estimateurs universels. Donc plusieurs auteurs les ont repris afin de résoudre le problème d'égalisation de canaux.

Dans [SWE98], la comparaison de deux RNA multicouches est faite. Le premier comporte des connections régulières alors que le deuxième a des connections irrégulières et on démontre qu'en ayant des connaissances à priori sur le système, le deuxième obtient de meilleurs résultats.

Cependant, l'absence de récurrence rend ce type de RNA moins robuste mais ils sont très attrayants pour une intégration à très grande échelle puisqu'il n'y a pas de dépendance dans le réseau, alors comme nous le verrons à la section 4.2, les différentes techniques du pipeline peuvent être appliquées.



a)



b)

Figure 3.7 : Les réseaux de neurones PL-MNN (a) et PL-RNN (b).

3.3 Proposition d'un algorithme

Le développement de l'algorithme dédié à l'égalisation de canaux a été réalisé en tenant compte des critères et contraintes au niveau de l'intégration à très grande échelle. Partant des travaux proposés par [LIU98] utilisant un RNA entièrement connecté nommé PL-RNN

("Piecewise Linear Recurrent Neural Network"), nous proposons une version simplifiée afin de respecter les contraintes architecturales ITGE. La simplification consiste en l'élimination des récurrences inter-couches afin de proposer un RNA multicouche que nous nommons PL-MNN ("Piecewise Linear Recurrent Neural Network"). La Figure 3.7 montre les deux versions.

Dans cette section, nous définirons cet algorithme basé sur les RNA. Tout, d'abord nous le ferons pour les nombres complexes, c'est-à-dire en utilisant une modulation QPSK et ensuite, nous ferons un cas particulier de l'algorithme utilisant les nombres réels tels que rencontrés en modulation PAM.

3.3.1 PL-MNN pour les nombres complexes

L'algorithme PL-MNN complexe est la forme la plus générale de l'algorithme que l'on puisse rencontrer. Il permet de corriger autant la phase d'un signal QPSK que la polarité d'un signal PAM bien que dans ce deuxième cas, des simplifications peuvent être apportées comme il est démontré à la section 3.3.2. L'entrée du RNA est un vecteur formé des éléments $[\tilde{y}(n), \tilde{y}(n-1),..., \tilde{y}(n-M+1),]$ afin de créer un modèle à moyenne mobile (MA : "Moving Average").

L'algorithme d'apprentissage est basé sur la rétropropagation de l'erreur proposé tout d'abord par [LEU91] dont une version simplifiée a été proposée dans [YOU98]. L'algorithme se présente donc en deux parties: la propagation et l'apprentissage, qui se décrivent comme suit:

- la propagation:

$$\hat{x}(n-M+1) = g\left(\sum_{k=1}^{K} q_k g(z_k)\right) \tag{3.1}$$

$$z_k = \sum_{m=1}^{M} w_{km}\tilde{y}(n-m+1) \textbf{ pour } k = 1,2,\ldots,K \tag{3.2}$$

Où g(z) est donné par:

$$g(z) = \left.\frac{|z+4|-|z-4|}{8}\right|_{z_{Re}} + j\left.\frac{|z+4|-|z-4|}{8}\right|_{z_{Im}} \tag{3.3}$$

- l'apprentissage

$$e(n) = s(n) - \hat{x}(n) \tag{3.4}$$

$$e(n) = \begin{cases} 0 & \text{if } |e(n)| < 0.3 \\ e(n) & \text{partout ailleurs} \end{cases} \tag{3.5}$$

$$\delta_k = g'(z(n)_{Re})\left[e(n)q_k^*\right]_{Re} + j\, g'(z(n)_{\mathrm{Im}})\left[e(n)q_k^*\right]_{\mathrm{Im}} \tag{3.6}$$

$$w_{km}(n+1) = w_{km}(n) + \mu\delta_k y_m^*(n) \tag{3.7}$$

$$q_k(n+1) = q_k(n) + \mu e(n)o_k^*(n) \tag{3.8}$$

Où k=1,2,...,L et m=1,2,...,M, l'astérisque (*) correspond au complexe conjugué, μ est le pas d'apprentissage, $s(n)$ est le signal transmis connu et $g'(z)$ est la dérivée de la fonction d'activation évaluée au point z calculée comme suit:

$$g'(z) = \begin{cases} 0.25 & si\ -4<z<4 \\ \mathbf{0} & partout\ ailleurs \end{cases} \tag{3.9}$$

L'équation (3.5) est défini comme étant le confinement de l'erreur et a été proposé dans [NAI97]. Celle-ci permet une meilleure stabilité et accélère la convergence. Le principe est qu'une erreur minime (de l'ordre de 0.3 évalué empiriquement) n'entraîne pas une erreur de symbole, donc la mise à jour des poids du RNA n'est pas nécessaire. De plus, cette technique permet de diminuer le temps de simulation en réduisant le nombre de mise à jour.

La sortie du RNA $\hat{s}(n-d)$ est obtenue par la propagation à partir de (3.1) du vecteur $[\tilde{y}(n), \tilde{y}(n-1),..., \tilde{y}(n-M+1),]$. Il apparaît donc que ce filtre crée un décalage temporel qui s'ajoute à celui introduit par le canal pour donner un retard *d*.

3.3.2 PL-MNN pour les nombres réels

Lorsqu'une modulation PAM est utilisée pour la transmission, des simplifications dans l'algorithme peuvent être apportées rendant son implantation en technologie ITGE plus simple puisque des unités arithmétiques pour nombres réels peuvent alors être employés. L'algorithme se divise toujours en deux parties: la propagation et l'apprentissage et ils sont définis comme suit:

- la propagation:

$$\hat{s}(n-d) = g\left(\sum_{k=1}^{L} q_k\, g\left(\sum_{j=1}^{M} w_{kj}\, \tilde{y}(n-j+1)\right)\right) \tag{3.10}$$

où $g(z)$ est la fonction canonique linéaire par morceaux définie par :

$$g(z)=\frac{|z+4|-|z-4|}{8} \tag{3.11}$$

- l'apprentissage:

$$c(n) = \mu(s(n)- \hat{x}(n)) \tag{3.12}$$

$$w_{kj}(n+1)=w_{kj}(n)+c(n)q_k(n)g'(z(n))\tilde{y}_j(n) \tag{3.13}$$

$$q_k(n+1)=q_k(n)+c(n)o_k(n) \tag{3.14}$$

pour j=1...M et k=1...L, où μ est le pas d'apprentissage, $s(n)$ est le symbole transmis et $g'(z)$ est la dérivée de la fonction d'activation calculée de la manière suivante:

$$g'(x)=\begin{cases} 0.25 & si\ -4 < x < 4 \\ 0\ partout\ ailleurs \end{cases} \tag{3.15}$$

La valeur de M est fonction de la réponse impulsionnelle du canal et le nombre de neurones sur la couche cachée L est à optimiser empiriquement.

La sortie du RNA $\hat{s}(n-d)$ est obtenue par la propagation à partir du vecteur formé par $\tilde{y}(n)$ jusqu'à $\tilde{y}(n-M)$. Il apparaît donc que ce filtre crée un décalage temporel.

3.4 Résultats de simulations et comparaison de performances

Cette section consiste à comparer différents algorithmes d'égalisation de canaux. Ce qui caractérise les différents algorithmes est leur robustesse au bruit $\eta(n)$, leur vitesse de convergence en phase d'adaptation et la capacité de reconstituer dans le cas d'un canal non

stationnaire ou variant. L'évaluation quantitative de performance sera donnée par le calcul du BER.

Le BER est la proportion de bits erronés lors d'une transmission binaire. Il s'agit du rapport entre le nombre de bits erronés et le nombre total de bit transmis comme le montre l'équation suivante :

$$BER = \frac{Nombre\ de\ bits\ erronnés}{Nombre\ de\ bits\ transmis} \tag{3.16}$$

Pour qu'il soit le plus représentatif possible, le nombre de données d'évaluation doit être assez grand pour que les plus petites proportions soient représentées.

3.4.1 PL-MNN pour les nombres réels

Le choix des algorithmes de comparaison sont les algorithmes LMS et RLS bien décrits dans [HAY93]. De plus, leur convergence ne nécessite pas une estimation de la réponse impulsionnelle du canal contrairement à des méthodes comme Kalman et Viterbi. Une période d'adaptation est nécessaire avec des données connues. Nous allons aussi comparer la méthode PL-MNN (à l'aide du programme présenté à l'annexe D) avec la méthode PL-RNN afin de connaître les effets de l'élimination des récurrences dans le RNA.

Pour toutes les études, un filtre d'ordre 11 a été utilisé pour les méthodes LMS et RLS alors que pour les algorithmes PL-MNN et PL-RNN, le RNA possède une seule couche cachée avec trois (L=3) neurones sur cette couche et trois valeurs d'entrée (M=3). Le pas d'apprentissage μ pour l'algorithme LMS est fixé à 0.0625, soit la valeur la plus élevée qui assure la convergence sur le canal linéaire, le facteur d'oubli λ de l'algorithme RLS est fixé

à 0.98 et le pas d'apprentissage μ pour les algorithmes basés sur les RNA est fixé à 0.5 pour le canal linéaire et 0.25 pour le canal non-linéaire.

La génération des données transmises se fait de façon pseudo-aléatoire avec des valeurs de l'intervalles [-1,1] qui sont équiprobables.

- Définition des canaux

Le canal linéaire utilisé est formé d'une réponse impulsionnelle à trois (3) points [HAY91] suivant l'équation :

$$\tilde{y}(n) = h_1 s(n) + h_2 s(n-1) + h_3 s(n-2) + \eta(n) \tag{3.17}$$

où

$$h_n = \frac{1}{2}\left[1 + cos\left(\frac{2\pi}{W}(n-2)\right)\right]_{n=1,2,3} \tag{3.18}$$

Ce qui rend le problème mal conditionné.

Le paramètre W fait varier l'étendue de la réponse impulsionnelle, déformant plus ou moins le signal transmis. Plus W est grand, plus le signal est déformé. Ensuite, un bruit blanc $\eta(n)$ de variance σ_n^2 est introduit.

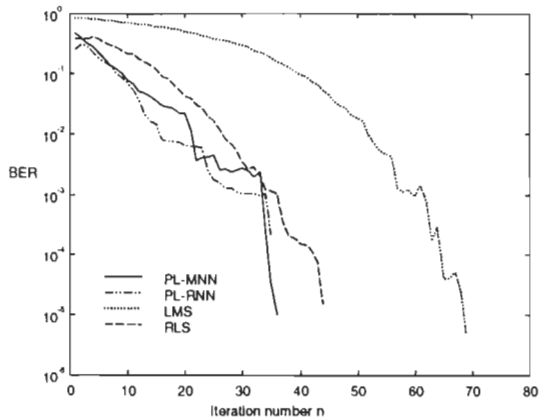


Figure 3.8 : Vitesse de convergence sur un canal linéaire avec un SNR de 20 dB.

Le canal non-linéaire est semblable au canal linéaire. Cependant, des non-linéarités sont introduites entre la réponse impulsionnelle et l'incorporation du bruit blanc. Ainsi, les données qui traversent le canal sont déformées de la façon suivante :

$$y_n = 0.5x + x^2 + x^3 + \eta(n) \tag{3.19}$$

$$x = 0.25s(n-2) + s(n-1) + 0.25s(n) \tag{3.20}$$

Où s correspond aux données transmises.

- Vitesse de convergence

Pour l'étude de la vitesse de convergence, il s'agit de calculer le BER après chaque itération d'apprentissage. Le nombre de données de validation était de 10000, et 20 répétitions ont été effectuées pour donner un aperçu des courbes de convergence des différents algorithmes. Pour chacun des types de canal, la vitesse de convergence a été évaluée pour un niveau de bruit assurant la convergence; soit 20 dB pour le canal linéaire,

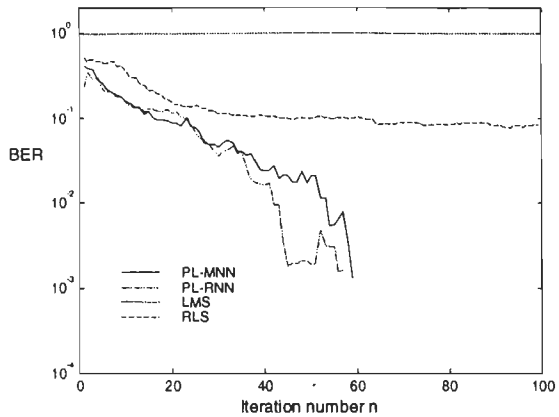


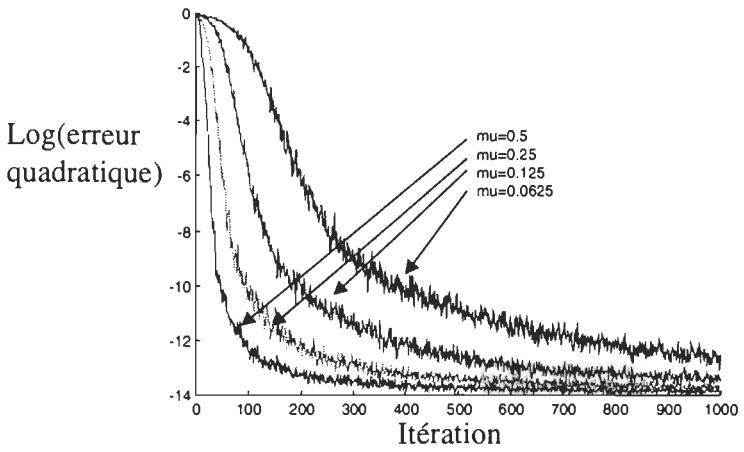
Figure 3.9 : Vitesse de convergence sur un canal non-linéaire avec un SNR de 25 dB.

alors que 25 dB ont été utilisé pour le canal non-linéaire. La Figure 3.8 et la Figure 3.9 présentent les valeurs moyennes des 20 répétitions pour chacun des types de canal.

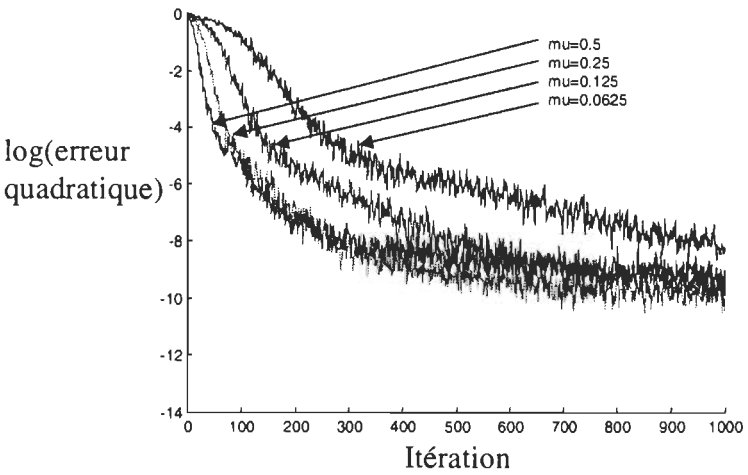
Les algorithmes PL-RNN et PL-MNN ont une vitesse de convergence presque deux fois plus rapide et l'algorithme RLS, reconnu pour sa vitesse de convergence.

Il est intéressant de remarquer que l'algorithme RLS minimise l'erreur mais sans jamais l'annulée sur le canal non-linéaire alors que l'algorithme LMS diverge tout simplement et ce peu importe le pas d'apprentissage. Pour les réseaux de neurones, les erreurs de transmission sont éliminées dans chacun des cas. Toutefois, la convergence du PL-RNN est moins assurée que celle du PL-RNN puisqu'il s'est avéré que le pas d'apprentissage μ ainsi que les poids initiaux ont beaucoup d'influence sur la convergence.

La vitesse de convergence est cependant influencée par le pas d'apprentissage μ. La Figure 3.10 présente les résultats de convergence sur les canaux linéaires et non-linéaires pour 500 réalisations différentes. La moyenne de l'erreur quadratique de chacune des itérations est présentée. On peut voir que l'erreur minimale est atteinte sur le canal linéaire



a)



b)

Figure 3.10 : Influence de la variation du pas d'apprentissage μ sur la convergence sur un canal (a) linéaire et (b) non-linéaire.

avec tous les μ utilisés alors que sur le canal non-linéaire, l'erreur minimale atteinte dépend de ce paramètre.

- Robustesse au niveau de bruit additif du canal

La robustesse au niveau de bruit additif du canal consiste à évaluer l'effet de la variance du bruit aléatoire sur l'efficacité des algorithmes de correction. Pour faire cette analyse, chacune des méthodes de correction ont eu 200 données d'apprentissage pour chacun des niveaux de bruit. La validation, pour évaluer le BER, s'est faite sur 10000 données. Pour certifier les résultats obtenus, 20 répétitions ont été faites et la moyenne de ces répétitions a été mise en graphique pour chacun des canaux à la Figure 3.11 et à la Figure 3.12.

La robustesse face au bruit des différents algorithmes sont semblables. Les algorithmes RLS et PL-RNN offrent les meilleures performances sur un canal linéaire. L'algorithme LMS et RLS devraient normalement être identique, mais les 200 données d'apprentissage n'ont pas été suffisantes pour certain niveau de bruit afin de minimiser l'erreur de reconstitution. Pour ce qui est de la méthode PL-MNN, ses performances sont plus que respectables compte tenu de la simplicité de l'algorithme. Le rapport signal à bruit (SNR : Signal to Noise Ratio) est défini pas l'équation suivante :

$$SNR = \frac{\|s(n)\|_2}{\|\eta(t)\|_2} \tag{3.21}$$

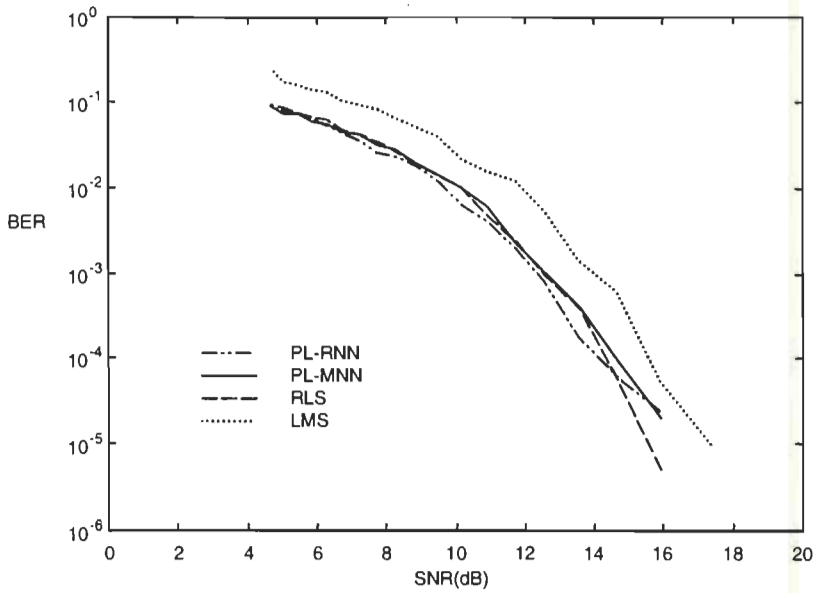


Figure 3.11 : Robustesse au bruit sur un canal linéaire

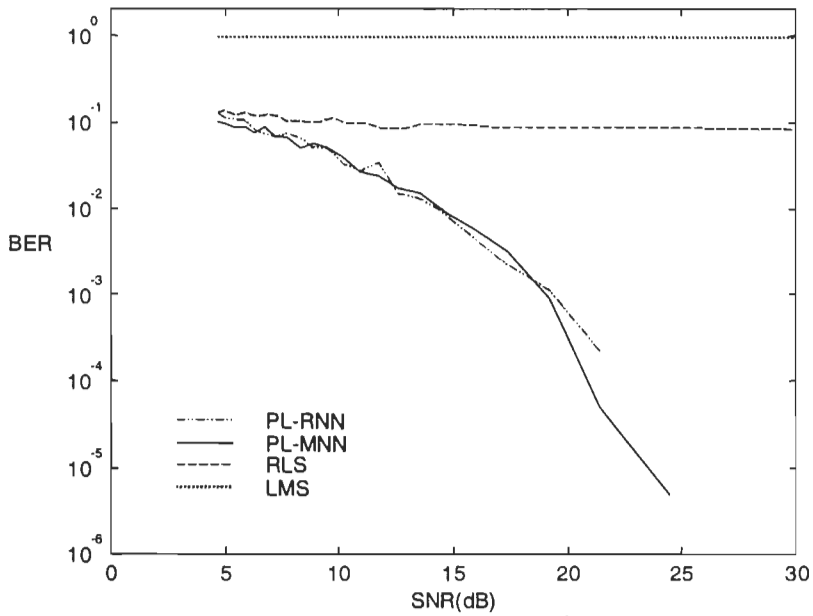
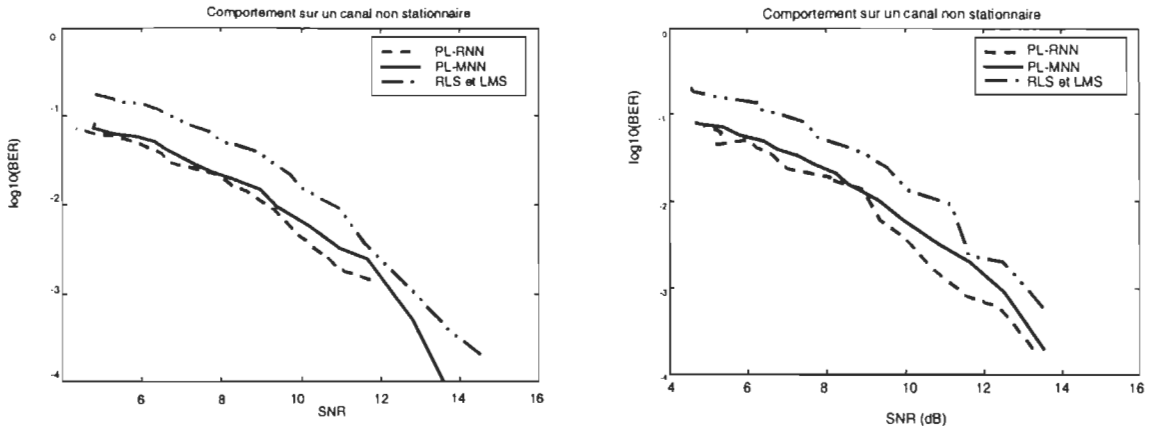


Figure 3.12 : Robustesse au bruit sur un canal non-linéaire.



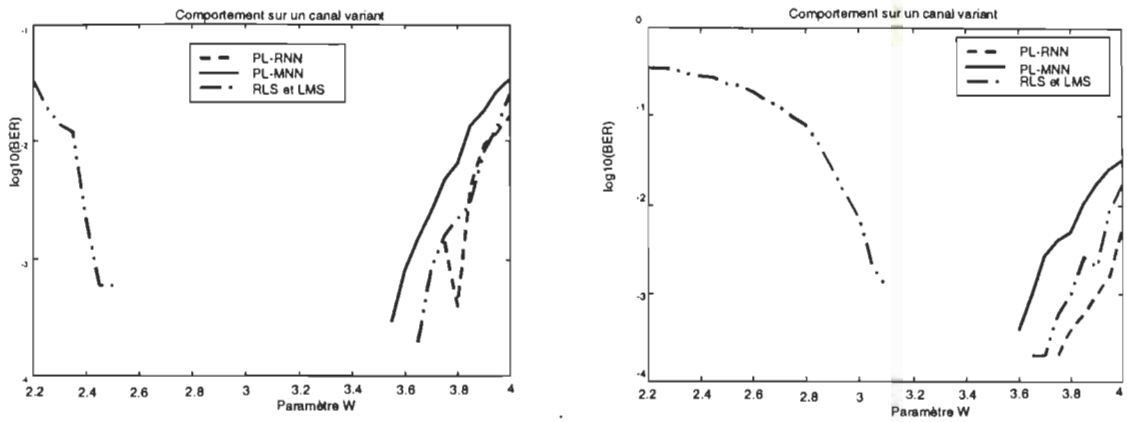
a) b)

Figure 3.13 : Comportement sur un canal linéaire non stationnaire après l'apprentissage effectué avec des données bruitées à 20 dB (a) et 30 dB (b).

Étant donné que les algorithmes RLS et LMS sont des méthodes linéaires, il est normal qu'il leur soit plus difficile de reconstituer avec un canal non-linéaire. Cependant, il est étonnant de constater que le réseau PL-RNN offre une moins grande robustesse au bruit aléatoire que le réseau PL-MNN.

- Robustesse face à un canal non-stationnaire

Ensuite, il faut évaluer la capacité de chacun des algorithmes à travailler avec un canal non-stationnaire c'est à dire que la variance du bruit varie dans le temps. Pour cette analyse, un apprentissage avec 200 données a été effectué. Le rapport signal sur bruit a été fixé à 20 dB et 30 dB. Ensuite, un balayage avec 2000 données dont le rapport signal sur bruit varie graduellement de 40 dB à 6 dB, a été effectué en calculant le BER pour chacun des niveaux de bruit. 5 répétitions ont été effectuées en évaluant la moyenne des répétitions.



a) b)

Figure 3.14 : Comportement sur un canal variant avec l'apprentissage à a) W=3.1 et b) W=3.5.

Selon les tendances montrées par la Figure 3.13, l'algorithme PL-RNN offre les meilleurs résultats alors que les résultats semblables ont été obtenus pour les algorithmes RLS et PL-MNN. Il est intéressant de constater qu'avec un apprentissage par des données dont le SNR est de 30 dB, le BER est inférieur à l'inverse du nombre de bits transmis à partir de 14 dB.

- Robustesse face à un canal variant

Enfin, nous évaluerons la capacité de chacun des algorithmes à travailler avec un canal variant c'est-à-dire quand la réponse impulsionnelle varie dans la temps. Pour cette étude, un apprentissage avec 200 données a été effectué. L'apprentissage a été effectué avec des données déformées par un canal linéaire ayant un paramètre W de 3.1 et 3.5. Le rapport signal sur bruit a été fixé à 20 dB. Ensuite, un balayage, avec 2000 données pour différentes valeurs de W variant graduellement entre 2.2 et 4, a été effectué en calculant le BER pour différentes valeurs de W. Cinq (5) répétitions ont été effectuées en évaluant la moyenne des répétitions.

Les tendances montrées par la Figure 3.14 indique que les algorithmes PL-RNN et PL-MNN reconstituent parfaitement lorsque la déformation est plus faible (W plus faible), alors que les algorithmes RLS et LMS sont limités à ce niveau. Il est possible d'égaliser des canaux comportants des déformations plus importantes que l'apprentissage. Les meilleurs résultats ont été obtenus avec le réseau PL-RNN. L'algorithme RLS, lorsque la déformation est plus importante, obtient des résultats supérieurs que le réseau de neurones PL-MNN.

3.4.2 PL-MNN pour les nombres complexes

L'étude de performance l'algorithme PL-MNN pour les communications QPSK (à l'aide du programme présenté à l'annexe E) sera moins exhaustive que celle pour les nombres réels, le but étant de démontrer le fonctionnement de l'algorithme face à transmissions utilisant une modulation QPSK.

Pour les analyses nécessitant une comparaison, les algorithmes LMS et RLS complexes ont été employés. Des filtres de 11 coefficients ont été utilisés. Nous allons aussi comparer la méthode PL-MNN avec un algorithme basé sur un RNA multicouche utilisant une fonction d'activation de type sigmoïde pour connaître l'effet de l'approximation de la fonction d'activation par la fonction "Piecewise Linear". Les RNA utilisés ont cinq neurones sur la couche cachée (L=5) et ils possèdent quatre entrées (M=4).

La génération des données transmises se fait de façon pseudo-aléatoire avec des valeurs de l'intervalles [-1,1] qui sont équiprobables et ce autant pour la partie réelle qu'imaginaire du symbole.

-Description du canal

La canal linéaire utilisé pour les simulations numériques possède une réponse impulsionnelle à 7 points décrit par:

$$\begin{aligned}\tilde{y}(n) &= (-0.005 - 0.004j)s(n) + (0.009 + 0.030j)s(n-1) + (-0.024 - 0.104j)s(n-2) \\ &+ (0.854 + 0.520j)s(n-3) + (-0.218 + 0.273j)s(n-4) + (0.049 - 0.074j)s(n-5) \\ &+ (-0.016 + 0.020j)s(n-6) + \eta(n)\end{aligned} \tag{3.22}$$

Ce canal n'est pas sévère puisque ses zéros sont assez loin du cercle complexe unitaire mais il est s'agit d'un canal à phase mixte puisqu'il possède des zéros à l'intérieur et à l'extérieur du cercle unitaire comme le montre la Figure 3.15. Ce type de canal est le plus général [MAC98].

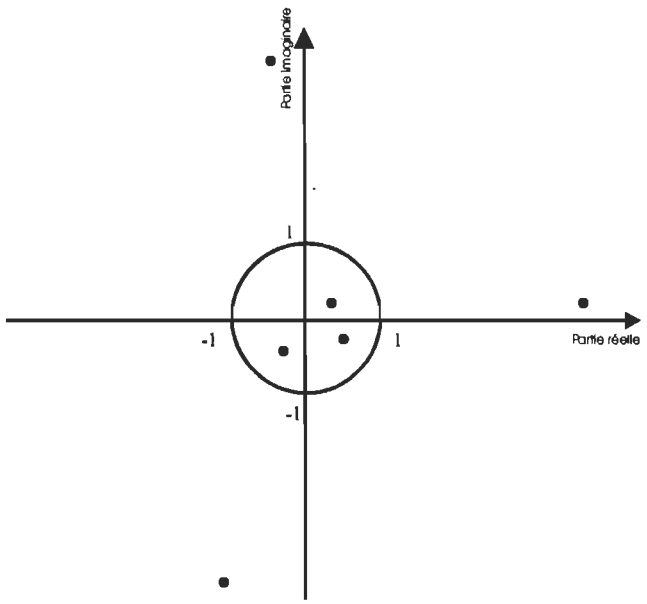


Figure 3.15 : Position des zéros dans le plan complexe

Le canal non-linéaire, quant à lui, utilise la même réponse impulsionnelle, mais une non-linéarité est introduite autant sur la phase que sur l'amplitude:

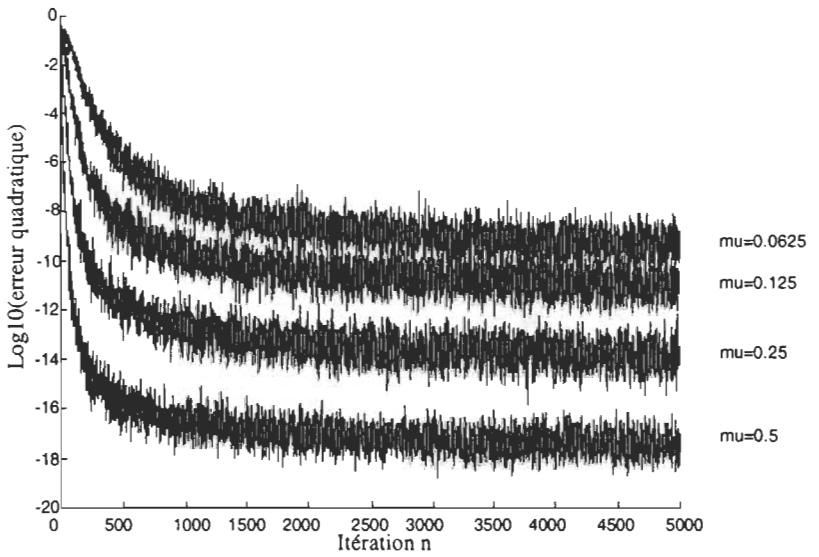
$$\begin{aligned} v(n) &= (\textbf{-0.005 - 0.004j})s(n) + (\textbf{0.009 + 0.030j})s(n-1) + (\textbf{-0.024 - 0.104j})s(n-2) \\ &+ (\textbf{0.854 + 0.520j})s(n-3) + (\textbf{-0.218 + 0.273j})s(n-4) + (\textbf{0.049 - 0.074j})s(n-5) \\ &+ (\textbf{-0.016 + 0.020j})s(n-6) \end{aligned} \tag{3.23}$$

$$|\tilde{y}(n)| = 0.5|v(n)| + 0.25|v(n)|^2 + 0.5|v(n)|^3 + \eta_1(n) \tag{3.24}$$

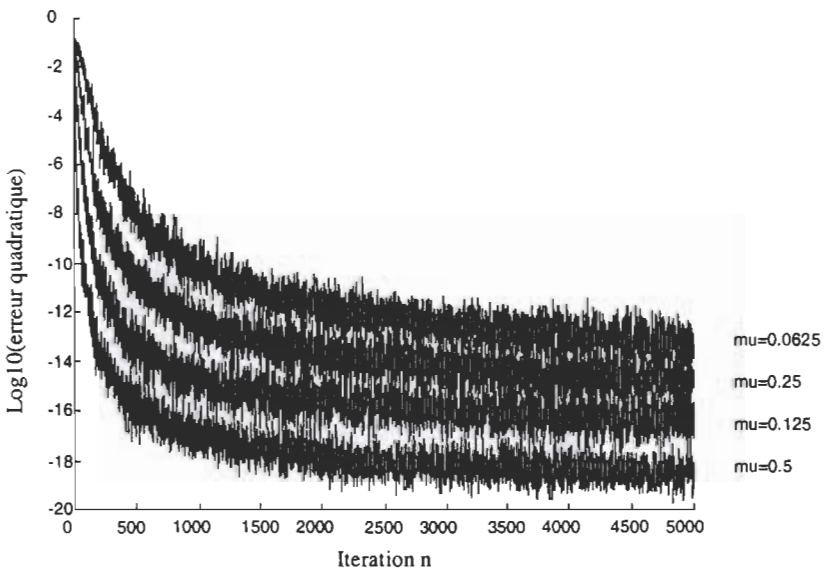
$$arg(\tilde{y}(n)) = arg(v(n)) + |\tilde{y}(n)| \tag{3.25}$$

-Vitesse de convergence

Lors de l'égalisation d'un canal de communication utilisant la modulation QPSK, le pas d'apprentissage μ influence la convergence beaucoup plus sur un canal linéaire que non-linéaire comme le montre la Figure 3.16 qui présente la moyenne de l'amplitude de l'erreur quadratique pour 500 réalisations différentes. En effet, pour de faibles pas d'apprentissage, des erreurs inférieures ont été atteintes sur le canal non-linéaire.



a)



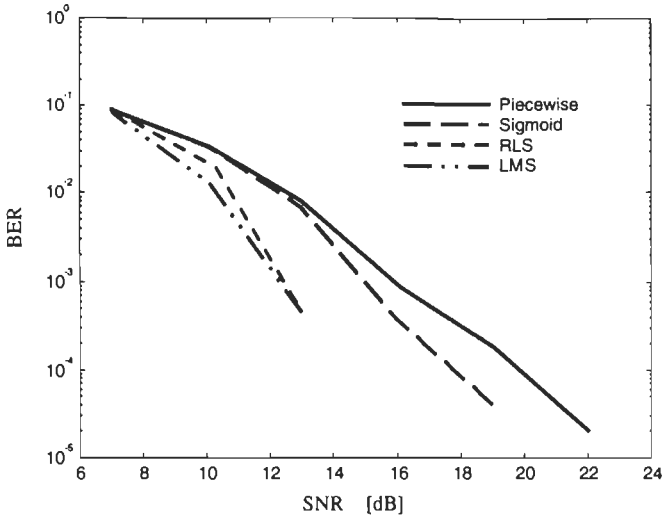
b)

Figure 3.16 : Influence du paramètre μ sur la convergence pour un canal linéaire (a) et non-linéaire (b) utilisant une modulation QPSK.

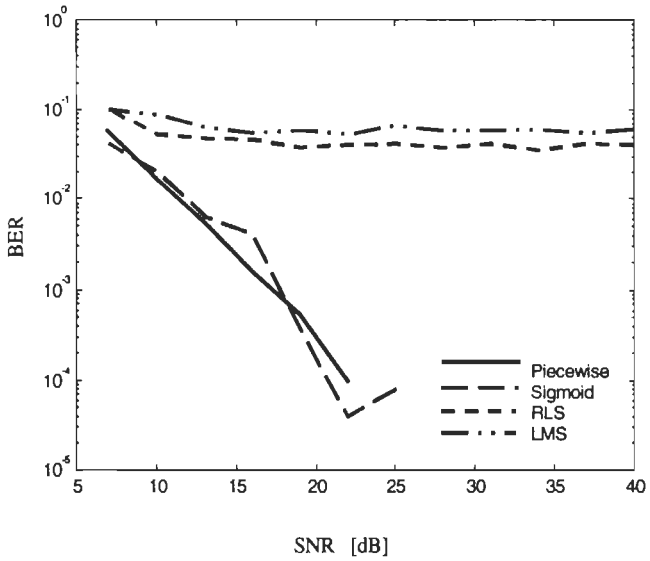
-Robustesse face au bruit additif du canal

Pour faire cette analyse, chacune des méthodes de correction ont eu 5000 données d'apprentissage pour différents niveaux de bruit variant entre 40 dB et 6 dB. La validation, pour évaluer le BER, s'est faite sur 10000 données. Pour certifier les résultats obtenus, 5 répétitions ont été faites et la moyenne de ces répétitions a été mise en graphique pour chacun des canaux à la Figure 3.11 et à la Figure 3.12 afin de connaître la tendance de cette robustesse.

Dans le cas de canaux linéaires, les algorithmes LMS et RLS offrent une meilleure robustesse face au bruit additif que les méthodes basées sur les RNA. Cependant, lorsque des non-linéarités sont introduites par le canal, les méthodes PL-MNN et l'algorithme utilisant la fonction d'activation de type sigmoïde sont plus performants. La Figure 3.18 montre des exemples de reconstitutions sur un canal non-linéaire avec les algorithmes RLS et PL-MNN. On peut voir que la méthode RLS est incapable d'égaliser correctement alors que la méthode PL-MNN réussit sans erreur pour le nombre de données utilisé dans les simulations.

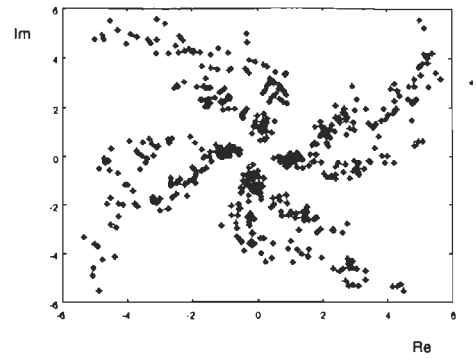


a)

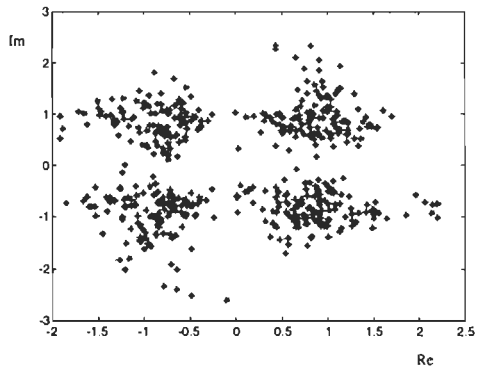


b)

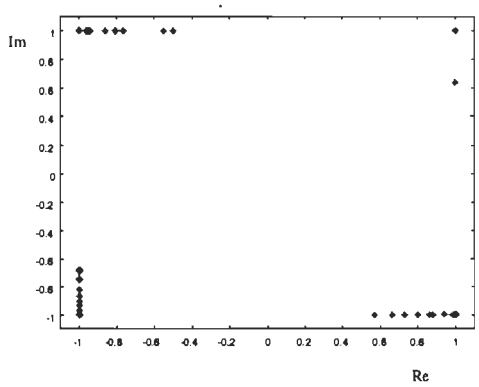
Figure 3.17 : Robustesse face au bruit additif pour un canal linéaire (a) et non-linéaire (b) pour des transmissions QPSK.



a)



b)



c)

Figure 3.18 : Exemple d'égalisation à partir de la sortie d'un canal non-linéaire (a) avec les algorithmes RLS (b) et PL-MNN (c).

3.5 Complexité de calcul

Les différentes opérations vectorielles demandent un certain nombre de calculs élémentaires. Les résultats pour chacun des algorithmes comparé sont présentés au Tableau 3.1.

Tableau 3.1 : Opérations élémentaires des différents filtres

Filtre	#additions/soustraction	# division	# multiplication	Total
LMS	2N		2N+1	4N+1
RLS	$2N^2+2N$	1	$3N^2+4N+1$	$5N^2+6N+2$
PL-RNN	$M^2+ K^2+2MK+M+3K$		$2K^2+4MK+M+6K$	$M^2+ 3K^2+6MK+2M+9K$
PL-MNN	MK+3K-1		4MK+2K +1	5MK+5K

3.6 Conclusion

Les performances de reconstitution montrent bien les forces de chacun des algorithmes. Au niveau de la convergence, les réseaux de neurones ont démontré qu'ils s'adaptaient jusqu'à deux fois plus rapidement que les filtres transversaux s'adaptant avec les algorithmes RLS ou LMS lors de transmissions utilisant la modulation PAM. Étant donné qu'aucune connaissance du canal n'est requise, une période d'adaptation est nécessaire ce qui empêche la transmission pendant cette période. Il est donc évident qu'il est très important que cette période soit la plus courte possible. Les réseaux de neurones ont eu un autre avantage indéniable, celui de pouvoir s'adapter sur un canal non-linéaire. Ce type de

canal se rapproche beaucoup plus de la réalité avec la saturation des amplificateurs et les convertisseurs dans les circuits de réceptions et de transmission.

Pour la robustesse au bruit, aucun algorithmes ne s'est démarqué sur le canal linéaire pour la modulation PAM alors que pour les communications utilisant la modulation QPSK, les méthodes LMS et RLS ont un net avantage. Au niveau des réseaux de neurones, sur le canal non-linéaire, les résultats sont comparables. Sur un canal non stationnaire, il a été intéressant de voir à quel point les algorithmes sont robustes. Le point d'opération est large et ne dépend presque pas des données d'apprentissage puisque les résultats ont été semblable pour les apprentissages avec des données ayant un rapport signal sur bruit de 20 dB et 30 dB. Les algorithmes se sont également comportés de façon similaire devant la variation de la réponse impulsionnelle. Cependant, les filtres tranversaux perdent leur capacités de reconstitution lorsque W diminue trop alors que ce n'est pas le cas pour les réseaux de neurones.

Une étude sur le nombre de neurones sur la couche cachée a été aussi réalisée. Les résultats sur la robustesse et la vitesse de convergence ont montré que ce paramètre a peu d'influence puisque les courbes étaient superposées dans la plupart des cas.

Ce qui différencie le plus les algorithmes, c'est la complexité de calcul qu'ils impliquent. L'algorithme LMS a une faible complexité de calcul mais sa vitesse de convergence est si lente, qu'elle le rend inapplicable pour résoudre le problème d'égalisation de canaux. L'algorithme RLS a une grande complexité de calcul qui n'est pas justifiée en regard des performances obtenues par les réseaux de neurones qui ont une plus faible complexité de calculs. Le réseaux PL-MNN, qui se compare avantageusement à la

méthode PL-RNN et au RNA utilisant une fonction d'activation de type sigmoïde pour ses performances, a une complexité de calculs très faible, dont l'ordre n'est fonction d'aucun carré contrairement au réseau PL-RNN. Il faut dire que plus les annulations spectrales sont profondes, plus le réseau de neurones récursifs se démarquera [PAR97]. C'est-à-dire que les zéros de la fonction de transfert du canal sont près du cercle unitaire, créant une forte atténuation du signal.

Chapitre 4

Architectures systoliques pour l'égalisation de canaux

Les architectures numériques prennent une place de choix dans les circuits modernes. Le développement des outils de conception assistée par ordinateur (CAO) s'est surtout orienté vers le design numérique, rendant du même coup ce type de circuit plus simple à réaliser. Les circuits analogiques ont été longtemps les plus utilisés. La diminution de la taille des transistors et l'augmentation de leur densité d'intégration permet de faire des circuits numériques complexes tout en occupant une surface d'intégration raisonnable. On parle maintenant même de circuits réalisés à la grandeur d'une tranche de silicium (WSI : "Wafer Scale Integration").

Cependant, les circuits analogiques ont certains inconvénients que les circuits numériques n'ont pas tels que la stabilité des circuits. Les circuits analogiques sont affectés par différents bruits qui peuvent conduire à un mauvais fonctionnement du circuit. De

l'extérieur, les champs électriques et magnétiques influencent le circuit tout comme les variations de la tension d'alimentation. De l'interne, les bruits thermiques et quantiques influencent le fonctionnement du circuit analogique. Aussi, la précision de ces circuits est limitée. La variation des paramètres telle la largeur des canaux introduit des erreurs dans les gains des transistors, et de ce fait, le comportement peut différer d'un circuit à l'autre. Également, le vieillissement des composants influence le bon fonctionnement ce qui limite la durée de vie du circuit. Cependant, les circuits analogiques ont des attraits indéniables. Ils peuvent fonctionner à des fréquences très élevées, ils consomment peu et occupent une faible surface d'intégration. Les circuits modernes fonctionnant en mode courant sont très performants et de plus en plus répandus.

D'autre part, les circuits numériques, quant à eux, sont stables. Le vieillissement les affecte peu, ils ont donc une bonne durée de vie. Les transistors qui les composent fonctionnent en saturation, la variation du gain de ceux-ci ne les affecte donc pas, rendant le fonctionnement presque indépendant des fluctuations du procédé de fabrication. De plus, la précision du circuit est contrôlée par la longueur des mots binaires employée et par le type d'arithmétique utilisé : virgule flottante ou virgule fixe. Cependant, même avec un effort constant pour diminuer la taille des transistors, la surface d'intégration des circuits numériques est très importante dans bien des cas. Les fréquences de fonctionnement dépendent de l'architecture des circuits mais sont souvent inférieurs à celles des circuits analogiques, et la consommation est aussi plus importante due aux charges et décharges rapides des capacités parasites. L'égalisation de canaux étant souvent utilisée pour un environnement bruité et la consommation de puissance du circuit n'étant pas un facteur

prédominant, les circuits numériques ont été choisi pour implanter en technologie ITGE l'algorithme PL-MNN.

Dans ce chapitre, il sera question à la section 4.1 des architectures parallèles dédiées en traitant des techniques de parallélisme des algorithmes pour par la suite converger vers les architectures systoliques en parlant de leur origine, des propriétés de ces architectures et de la systolisation des algorithmes. Une autre technique de parallélisation qui sera abordée en détail est le pipeline. Ensuite, à la section 4.2, une architecture systolique sera proposée pour l'algorithme PL-MNN. A la section 4.3 nous ferons le tour des unités arithmétiques nécessaires pour le fonctionnement de l'architecture PL-MNN autant réelle que complexe en portant une attention particulière au multiplieur-accumulateur (MAC), qui constitue le goulot d'étranglement de l'architecture. Enfin, la section 4.4 présente des résultats de performance de l'architecture lors d'une implantation ITGE dans une technologie de 0.5 µm.

4.1 Architectures parallèles dédiées

Pour répondre aux exigences des transmissions numériques, les circuits doivent pouvoir exécuter un grand débit de calcul. Pour arriver à cette finalité, la parallélisation de l'algorithme est nécessaire. Le principe général du parallélisme est de gagner de la vitesse en effectuant des calculs simples de façon simultanée. Des processeurs élémentaires sont reliés pour faire un calcul global. Les architectures multiprocesseurs sont synchronisées et offrent un excellent rendement. Il s'agit donc d'avoir un bon contrôle des opérations pour rendre efficace l'architecture dans son ensemble.

4.1.1 Techniques du parallélisme

Les techniques du parallélisme sont nombreuses, que ce soit au niveau logiciel ou matériel. Dans cette section, il ne sera question que des techniques du parallélisme matérielles, plus précisément les systèmes multiprocesseurs, les architectures systoliques et les différentes techniques du pipeline.

La première technique utilisée est la multiprocesseurs. Cette technique consiste à utiliser plusieurs processeurs d'applications générales et de distribuer les tâches de calcul entre les différents processeurs. Cette méthode donne naissance à un système dédié parallèle et non à un processeur dédié parallèle. Cette technique utilise souvent une topologie maître-esclave c'est-à-dire qu'un processeur effectue le contrôle des données et des tâches des autres processeurs qui lui sont rattachés comme le montre la Figure 4.1. Les entrées-sorties de l'architecture se font à partir du maître et tous les esclaves ne peuvent communiquer qu'avec le maître et un seul à fois.

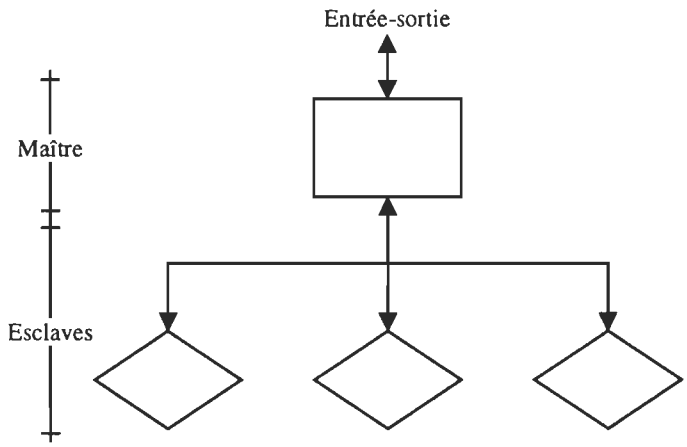


Figure 4.1 : Représentation de la topologie maître-esclave.

Un tel système demeure flexible dû au fait que les processeurs sont d'applications générales. Cependant, ce type de système occupe un espace non négligeable et peut s'avérer fort dispendieux. Aussi, le fait que les processeurs esclaves ne peuvent communiquer qu'un seul à la fois avec le maître peut s'avérer un désavantage quand les communications sont fréquentes. Pour un tel système, les tâches de chaque processeur doivent demeurer assez longues pour éviter les communications. Une telle topologie est souvent pratique pour les simulations en temps réel sur des processeurs possédant assez de mémoire pour stocker les données afin d'équilibrer les temps de communication et de calcul.

Une autre technique consiste à diviser non seulement les tâches de calculs mais aussi les contrôles en appliquant la philosophie « diviser pour régner ». Cette technique donne naissance à des architectures de type systolique, proposées pour la première fois en 1982 par H.T. Kung dans [KUN82]. Ces architectures sont constituées de processeurs élémentaires (PE) dédiés qui communique localement entre eux. En utilisant cette topologie, on arrive à des processeurs entiers dédiés à une classe d'applications. Cependant, tout algorithme n'est pas systolisable, certaines conditions sont à respecter comme nous allons le voir à la section 4.1.4.

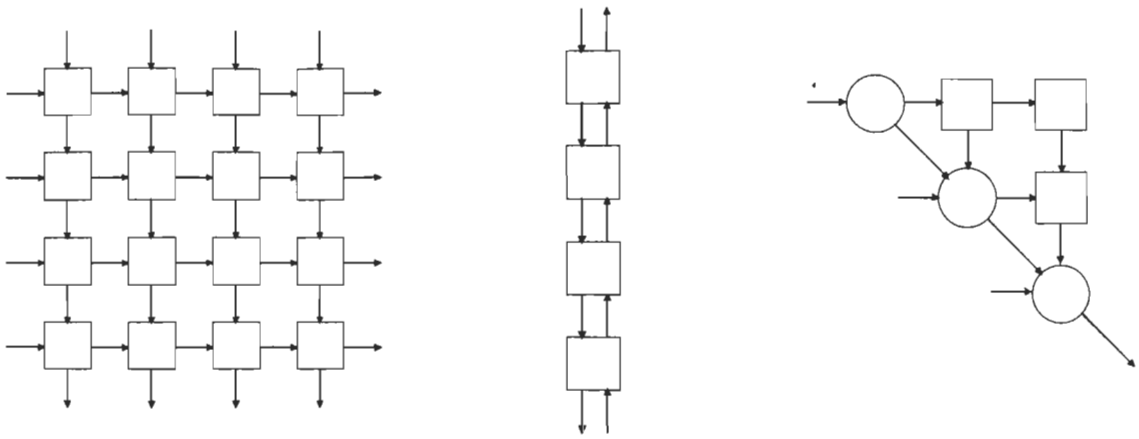
En troisième lieu, il est possible d'appliquer la technique du pipeline qui se situe à un niveau plus bas. Le principe est de diviser les blocs de logiques combinatoires (BLC) afin d'en augmenter le débit. Dans le pipeline conventionnel, chaque sous-bloc combinatoire est séparé du suivant par un registre comme le montre la Figure 4.2. De cette façon, le débit peut être augmenté mais souvent au dépend de la latence du circuit.

Figure 4.2 : Représentation d'un bloc combinatoire normal (a)et pipeliné (b).

4.1.2 Architectures systoliques

Les architectures systoliques sont en fait une extension du pipeline synchrone conventionnel. Basés sur le principe « diviser pour régner », les éléments qui les composent sont de plus ou moins faible complexité. Ses éléments ou processeurs élémentaires (PE) peuvent prendre différentes formes dans la représentation graphique des architectures systoliques comme le montre la Figure 4.3.

Le travail de ces cellules est synchronisé par une horloge. À chaque cycle, les PE font le travail qui leur est assigné et transmettent les résultats à leurs proches voisins. Donc, les données se propagent de manière pulsée tel le sang dans les vaisseaux sanguins, d'où le terme systolique.



a) b) c)

Figure 4.3 : Exemples d'architectures systoliques

Cependant, ce type d'architecture mène à des applications dédiées à cause de leur grande spécificité. Aussi, ces architectures ne constituent que la partie arithmétique du processeur, elles nécessitent donc un processeur hôte. En intégration, si la dimension du problème devient grande, la réalisation pratique est irréalisable à cause d'une trop grande surface d'intégration rendant impossible la distribution d'une horloge unique.

4.1.3 Propriétés des architectures systoliques

Les architectures systoliques offrent des propriétés intéressantes pour la mise en œuvre d'un circuit intégré dédié à une application spécifique (ASIC : "Application Specific Integrated Circuits"). Les principales sont la régularité, la modularité, le parallélisme et les communications locales.

La régularité et la modularité des architectures systoliques facilitent la réalisation de celles-ci. La régularité vient du fait que les PE sont pour la plupart identiques. Donc, seule la réalisation d'un petit nombre de PE est nécessaire pour ensuite générer l'architecture en

entier. Une optimisation locale sur les PE peut être réalisée plutôt qu'une optimisation globale de l'architecture afin d'obtenir un meilleur compromis entre la surface et le temps de calcul. Cette régularité simplifie également le placement des divers éléments sur le silicium.

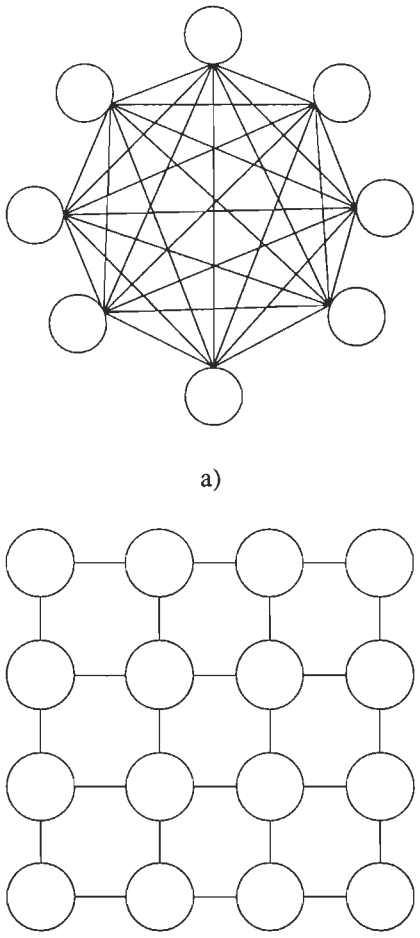
La modularité des architectures systoliques permet d'adapter facilement l'architecture pour différentes dimensions du problème. Par exemple, si la taille des matrices ou des vecteurs augmente, il s'agit seulement de grossir le réseau systolique tout en gardant la même topologie pour adapter l'architecture.

Le principe général du parallélisme est de gagner de la vitesse en effectuant des calculs simples de façon simultanée. Dans les architectures systoliques, le fait que tous les PE fonctionnent en même temps synchronisé sur l'horloge les rend hautement parallèles. Cependant, la surface d'intégration peut réduire le nombre d'unités arithmétiques qui sont implantées et par le fait même, réduire le degré de parallélisme de l'architecture. Aussi, le fait que l'horloge commande un nombre de commutations, des délais de connections peuvent entraîner un décalage temporel de celle-ci et nuire à la synchronisation du circuit intégré.

De manière réaliste, il est impossible physiquement que tous les processeurs soient connectés les uns aux autres. Aussi, il s'agit de ne pas ralentir le circuit à cause de connections trop longues. Les longues connections entraînent des capacités et des résistances parasites dans le circuits qui introduisent des délais temporels qui peuvent devenir non négligeables face au temps de calculs. C'est ici qu'intervient le principe de localité. C'est-à-dire que chaque processeur élémentaire ne communique qu'avec ses

voisins immédiats amenant des chemins de longueurs minimales comme le met en évidence la Figure 4.4.

En suivant le même principe, le rapport entrée-sortie doit suivre la même règle. Il faut donc désigner des processeurs élémentaires hôtes, en périphérie du réseau, qui communiqueront avec l'extérieur.



a)

b)

Figure 4.4 : Réseau de processeurs élémentaires a) sans communications locales et b) avec communications locales

Si on prend les technologies actuelles qui fabriquent des circuits en deux dimensions, il est impossible de communiquer efficacement qu'avec ses voisins en évitant dans bien des cas de changer de couche d'intégration. Il est donc profitable d'utiliser des communications locales.

4.1.4 Systolisation d'un algorithme basé sur les RNA

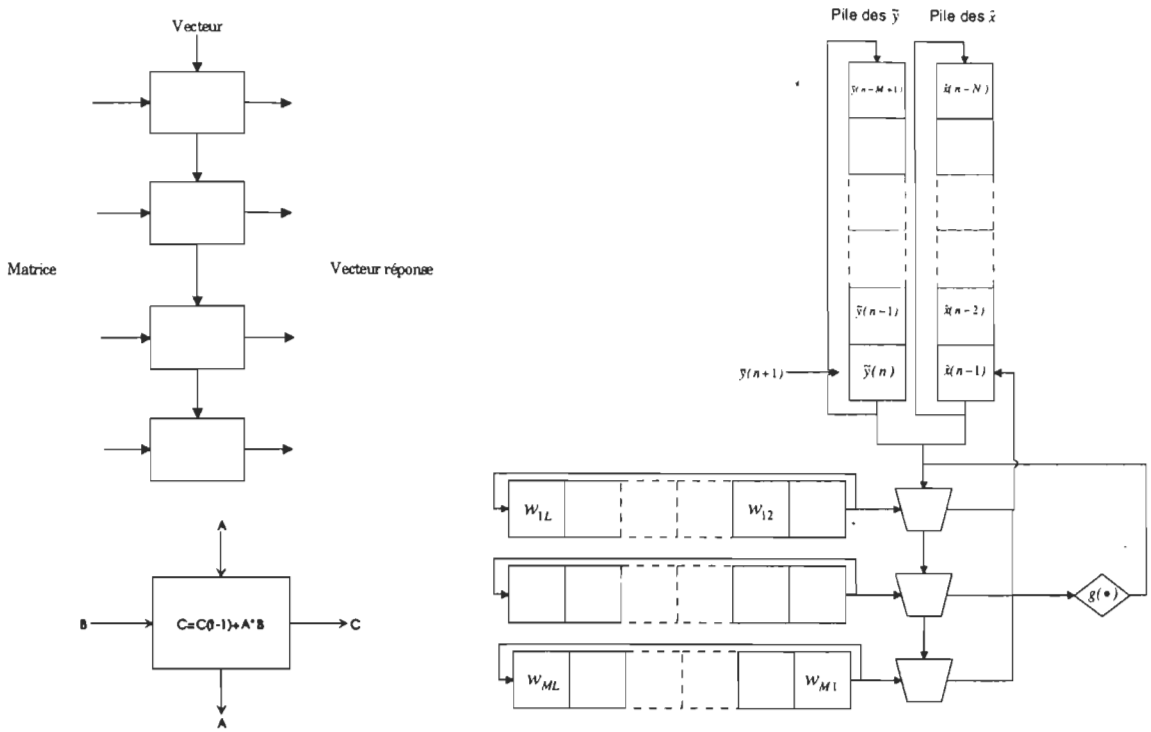
Afin d'optimiser au maximum les performances de notre architecture, il est important de bien comprendre les opérations mathématiques du RNA. Des architectures systoliques efficaces mettant l'accent sur différents critères de conception peuvent alors être considérées [VID98].

Un des éléments communs à toutes les couches dont nous avons déjà discuté est la somme des produits. A chaque neurone, chacune des entrées est multipliée par un poids, pour par la suite, sommer tous ces produits. Supposons une couche de quatre neurones qui possède deux entrées. Il y aura donc quatre valeurs en sortie de cette couche. Les deux entrées peuvent former un vecteur colonne à deux éléments alors que les poids forment une matrice quatre par deux comme il a été montré au chapitre précédent. La sortie de la couche, qui est un vecteur colonne à quatre éléments, sera donc le produit matrice-vecteur de la matrice des poids et du vecteur des entrées.

$$\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \\ W_{41} & W_{42} \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} = \begin{bmatrix} W_{11}E_1 + W_{12}E_2 \\ W_{21}E_1 + W_{22}E_2 \\ W_{31}E_1 + W_{32}E_2 \\ W_{41}E_1 + W_{42}E_2 \end{bmatrix} \tag{4.1}$$

Ce sont les éléments de ce nouveau vecteur qui seront affectés par la fonction de décision déterminée par le type de neurones constituant la couche. Lorsque chaque élément a été transformé par la fonction, le vecteur colonne sert d'entrée à la prochaine couche.

Le produit matrice vecteur est souvent effectué à partir d'une architecture systolique linéaire comme le propose [QUI89]. On dit linéaire puisqu'elle ne contient qu'une colonne de processeur élémentaire, comme le montre la Figure 4.5a. A partir de cette architectures linéaire, il est possible d'implanter un RNA NARX (voir section 3.1.3) comme le montre la Figure 4.5b [VID98B]. Le nombre de PE correspond au nombre de neurone sur la couche cachée. La récursivité des algorithmes à base de RNA peut être intégrée à l'aide de piles circulaires. Comme les éléments du vecteur réponse sont obtenues de façon séquentielle, une seule fonction d'activation est nécessaire dans l'architecture, ce qui constitue un net avantage quand on sait que dans la plupart des cas, la fonction d'activation est contenue dans une mémoire qui nécessite une grande surface d'intégration [NOR95].



a) b)

Figure 4.5 : Architectures systoliques du produit matrice-vecteur (a) et d'un RNA NARX (a).

Chaque processeur élémentaire doit effectuer l'opération :

$$c = c(t-1) + ab \tag{4.2}$$

De plus, la valeur c doit pouvoir être réinitialisée à zéro lorsqu'un nouveau produit matrice-vecteur doit être effectué. Également, un registre supplémentaire doit y être incorporé pour garder en mémoire la valeur de c à l'instant t-1.

Ce type d'architecture systolique est apte à effectuer des calculs nécessaires à la plupart des algorithmes basés sur les RNA et il offre un excellent compromis entre la surface d'intégration et le temps de calcul.

4.1.5 Techniques du pipeline

Dans cette section, les fondements du pipeline seront traités. Aussi, un survol du deux types de pipeline sera faite soit le pipeline conventionnel et le pipeline par vague. Nous verrons également les conditions qui sont nécessaires pour que les techniques du pipeline puissent être appliquées.

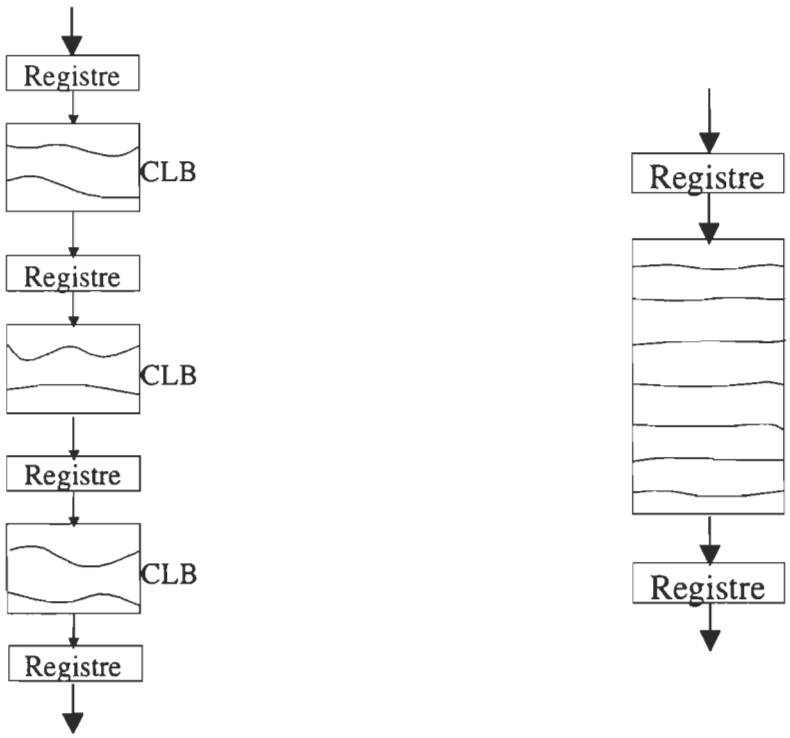
Le pipeline est une technique de parallélisation qui consiste à divisé un BLC pour en augmenter le débit. Il en résulte des BLC plus courts comme le montre la Figure 4.2. Cependant, la latence du circuit dépendra du BLC le plus long puisqu'il fixera la fréquence de fonctionnement.

La technique du pipeline conventionnelle consiste à séparer les BLC avec des registres afin d'assurer la synchronisation. Évidemment, le débit est augmenté au dépend de la surface d'intégration à cause de l'insertion de registres. Cependant, si les chemins de données sont équilibrés, c'est-à-dire que les temps de propagation de toutes les données sont tous égaux, et ce peu importe leur valeur, alors les registres peuvent être retirés pour donner naissance au pipeline par vague. De nouvelles données sont introduites dans le BLC avant même de connaître le résultat de l'entrée précédente pour créer des vagues de données dans le BLC.

De façon générale, le pipeline par vague est beaucoup plus performant. Il fonctionne à des fréquences plus élevées en évitant les problèmes de propagation d'horloge. Aussi, il nécessite habituellement une plus faible surface d'intégration que le pipeline conventionnel [BUR98]. Cependant, la conception est beaucoup plus ardue. Le fait de devoir équilibrer les chemins de données nécessite le design d'un circuit custom, rendant inutilisable les

outils de synthèse automatique. La stabilité des circuits utilisant le pipeline par vague est vulnérable aux variations de températures, de tensions d'alimentation et aux tolérances du procédé de fabrication [GHO95].

Pour ce projet, la technique du pipeline conventionnelle sera utilisée puisque les appareils de communications sont souvent contraints de fonctionner dans des situations non idéales.



a) b)

Figure 4.6 : Comparaison entre le pipeline conventionnel (a) et le pipeline par vague.

4.2 Proposition d'une architecture pour l'égalisation de canaux

Comme il a été mentionné au chapitre 3, les RNA possèdent toutes les caractéristiques afin d'obtenir une architecture systolique en vue d'une intégration ITGE. Cette architecture doit offrir la possibilité d'un apprentissage en ligne afin que le circuit puisse s'adapter au canal. Pour ce faire, l'architecture doit être en mesure d'effectuer les équations (3.1) à (3.8) pour la version pour les nombres complexes de l'algorithme PL-MNN et les équations 3.10 à 3.14 dans le cas de communications utilisant la modulation PAM.

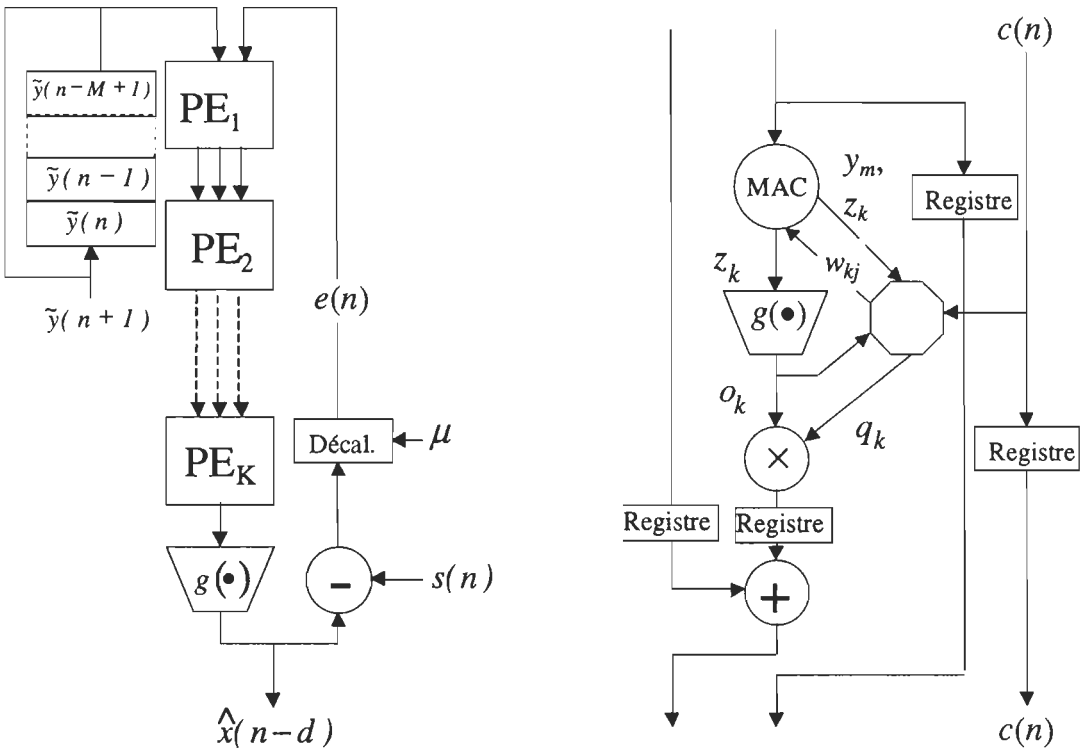
Dans chacun des algorithmes, la même architecture est proposée, en différenciant seulement les unités arithmétiques. Cette architecture est représentée à la Figure 4.7a alors que la Figure 4.7b montre la structure d'un PE. Le nombre de PE dépend du nombre de neurones sur la couche cachée puisque nous avons un PE par neurone.

Chacun des PE possède son contrôle dans la cellule octogonale que nous nommerons cellule d'apprentissage. Lors de l'apprentissage en ligne, la cellule d'apprentissage met à jour les poids, les fournit aux unités arithmétiques et les synchronise. Lors de l'apprentissage hors ligne, alors elle ne sert qu'à fournir et synchroniser les poids. Pour bien comprendre l'architecture, nous décrirons de fonctionnement de l'architecture pour les nombres réels, celle pour les nombres complexes se comportant de la même manière.

Le vecteur d'entrée $\tilde{y}(n)$, composé des éléments $[\tilde{y}(n),\ldots,\tilde{y}(n-M)]$, est contenu dans une pile circulaire à l'entrée de l'architecture. Celle-ci transmet séquentiellement les éléments de $\tilde{y}(n)$, en commençant par $\tilde{y}(n-M)$. Lorsque toutes les valeurs du vecteur

d'entrée ont été livrés à l'architecture, l'élément $\tilde{y}(n-M)$ est expulsé de la pile et élément $\tilde{y}(n+1)$ est introduit. Le réseau de PE et la pile circulaire qui contient le vecteur d'entrée donne une modularité à l'architecture permettant d'implanter des RNA possédant n'importe quel nombre d'entrées et de neurones sur la couche cachée.

Le vecteur d'entrée $y(n)$ se propage dans l'architecture. Les éléments circulent un à un à l'entrée des PE. Les valeurs sont d'abord multipliées par leur poids w_{jk} correspondant et accumulées jusqu'à ce que tous les éléments du vecteur $\tilde{y}(n)$ y aient circulé. Ces deux opérations effectuent la partie linéaire des neurones de la couche cachée, c'est-à-dire la somme des produits. Lorsque l'accumulation est terminée, le résultat se dirige à deux



a) b)

Figure 4.7 : Schéma de l'architecture proposée (a) et d'un processeur élémentaire (b).

endroits. Tout d'abord, il est transmis à la cellule d'apprentissage afin qu'elle soit en mesure d'effectuer l'équation 3.14. Aussi, ce résultat est acheminé vers la fonction d'activation canonique linéaire par morceaux.

Une fois la fonction d'activation calculée, le résultat est transmis à la cellule d'apprentissage pour calculer l'équation (3.15). Également, la sortie de la fonction d'activation est dirigée vers un multiplieur pour effectuer le produit avec le poids q_k et additionnée par la suite avec la somme partielle du PE précédent. Ces deux dernières opérations effectuent une partie des calculs de la partie linéaire du neurone de sortie, le résultat global étant fourni par PE_L. La fonction d'activation est ensuite opérée sur le résultat global pour obtenir la valeur corrigée $\hat{x}(n-d)$.

Pour compléter l'apprentissage, le résultat de correction $\hat{x}(n-d)$ est soustrait de la valeur transmise $s(n-d)$ pour obtenir l'erreur de reconstitution $e(n)$. Un registre à décalage pondère l'erreur de correction $e(n)$ par le facteur μ, le pas d'apprentissage, ce qui implique que μ doit être une puissance négative de 2. L'erreur pondérée $c(n)$ est ensuite propagée dans l'architecture et fournie aux cellules d'apprentissage pour finir la mise à jour des poids du RNA.

Le fait que les éléments nécessaires à l'apprentissage soient calculés séquentiellement par la phase de propagation de l'algorithme permet l'utilisation d'une cellule d'apprentissage de faible complexité comme le montre la Figure 4.8.

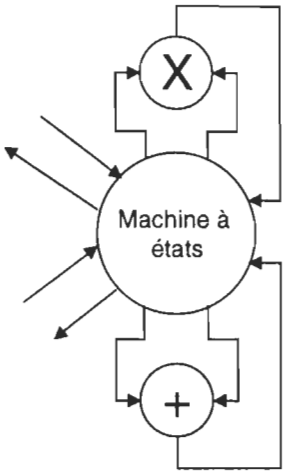


Figure 4.8 : Structure de la cellule d'apprentissage.

L'algorithme d'apprentissage ne demande pas un grand nombre de calculs. L'affectation de la dérivée de la fonction d'activation n'est est fait qu'un décalage puisqu'il s'agit une multiplication par 0.25 ou 0. Certaines valeurs sont même connues avant même que la propagation commence tel $\tilde{y}_j(n)$ et $q_k(n)$. Il est donc facile de réaliser les opérations nécessaires durant la phase de propagation de l'algorithme et ainsi ne pas nuire au débit de l'architecture au dépend d'une faible surface d'intégration.

4.3 Unités arithmétiques pour l'architecture proposée

Comme nous avons pu le constater à la section 4.2, les unités arithmétiques utilisé sont la fonction d'activation, le multiplieur-accumulateur (MAC), un multiplieur et un additionneur. Dans cette section, il ne sera question que du MAC, le multiplieur et l'additionneur étant des parties intégrantes de cette unité arithmétique. Une version pipelinée du MAC sera aussi proposée.

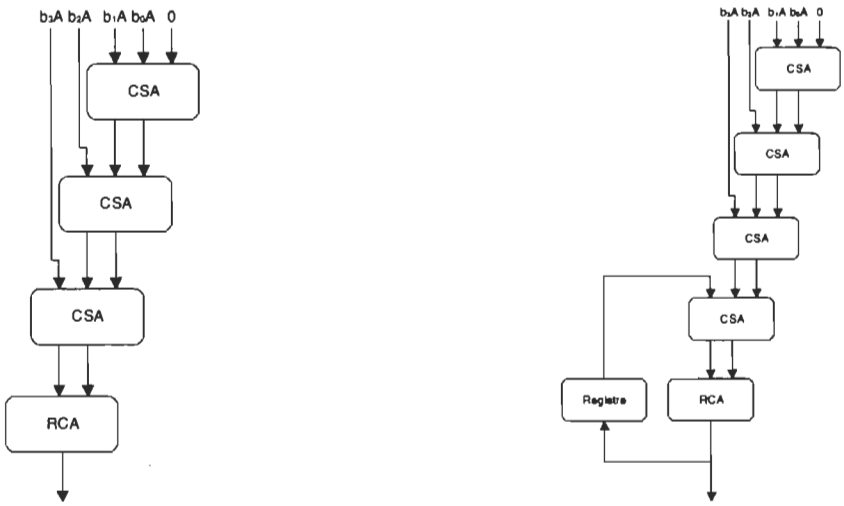
4.3.1 Le multiplieur-accumulateur

Le MAC est en fait un multiplieur cellulaire [MAD95]qui possède un étage additionneur à conservation de retenue (CSA : "Carry Save Adder") de plus qu'un multiplieur conventionnel de ce type. Le multiplieur cellulaire est basé sur la cellule additionneur complet (FA : "Full Adder") qui effectue les opérations :

$$u = a \oplus b \oplus c \tag{4.3}$$

$$v = ab + bc + ac \tag{4.4}$$

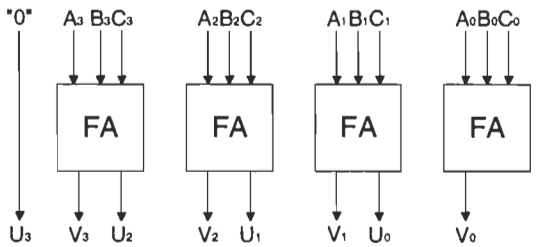
C'est à partir de cette cellule que tout le MAC est réalisé. De façon plus concrète, u est en fait la somme binaire et v est la retenue. Le principe du CSA est de transformer une addition complète de 3 nombres en une addition complète de 2 nombres. Comme la multiplication est une succession d'additions, cette structure a permis de donner naissance au multiplieur cellulaire. Une version 4 bits est présentée à la Figure 4.9a.



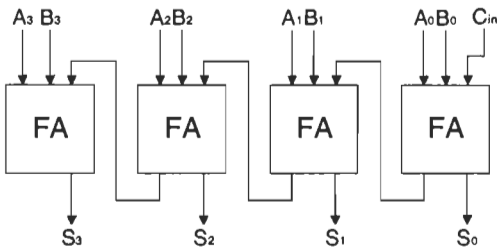
a) b)

Figure 4.9 : Multiplieur cellulaire 4 bits (a) et du MAC 4 bits (b).

Mis à part les FA, le multiplieur cellulaire utilise des portes ET entre les bits du multiplicande et un des bits du multiplicateur. Il s'agit d'une structure simple et régulière. L'addition complète à la fin de la multiplication se fait à l'aide d'un additionneur à propagation de retenue (RCA : "Ripple Carry Adder"). Alors on peut la modifier pour obtenir le MAC, comme montré à la Figure 4.9b. La différence entre le CSA et le RCA se situe au niveau des connections des cellules FA. La Figure 4.10 montre ces deux unités arithmétiques. Cependant, une telle structure n'accepte pas les nombres signés. Cette lacune peut être contrée comme nous allons le voir dans la section suivante.



a)



b)

Figure 4.10 : Schémas du CSA (a) et du RCA (b) 4bits.

4.3.2 Multiplieur-accumulateur pipeliné

La structure du MAC présenté à la section 4.3.1 se prête bien pour le pipeline. En effet, les blocs combinatoires CSA peuvent être séparé par des registres pour augmenter le débit de l'architecture. Le MAC proposé tient compte du signe avec une représentation signe-module pour les nombres entrants et une représentation complément à 2 pour le résultat. Il se divise en quatre blocs : le multiplieur cellulaire, la temporisation, le complément à 2 et l'accumulation tel que présenté à la Figure 4.11.

Prenons le MAC 8x16 bits comme exemple. Les bits de signes sont retirés des nombres pour être traités par une porte XOR qui déterminera le signe du résultat de multiplication. L'amplitude des nombres est introduite dans le multiplieur cellulaire. Les résultats U et V sont ensuite temporisés pour les synchroniser avec l'accumulation. Cette temporisation se fait à l'aide d'une topologie triangulaire de registre dont la hauteur est de 15 registres. Elle est nécessaire afin de synchroniser le bit le plus significatif avec la propagation de la retenue dans les RCA. Suite à la temporisation, les nombres subissent un complément à 1. Il s'agit en fait de porte XOR dont une entrée est le signe de la multiplication tel que le représente la Figure 4.12. Pour compléter le complément à 2, nous devons additionner 1 au deux résultats de multiplication, U et V. Cette addition sera faite dans l'accumulation. Comme le montre la Figure 4.13, les opérations nécessitent deux RCA: un pour compléter la multiplication et un pour l'accumulation. Pour finir le complément à deux, la retenue d'entrée (C_{in}) prend la valeur du signe. Également, le signe vient s'ajouter au nombre à la position la plus significative.

L'accumulation crée une dépendance dans le flot de données. Donc au lieu d'attendre le mot binaire en entier pour faire l'accumulation, elle se fait bit par bit. Ceci est possible en pipelinant la retenue des RCA. La temporisation des résultats de multiplication permet de tenir compte de ce pipeline dans la synchronisation des données.

Cette façon, nous obtenons un pipeline très profond. Cependant, la distribution de l'horloge doit être optimisée pour diminuer l'effet de cette propagation sur la latence du MAC.

La validation de la structure du MAC pipeliné a été réalisée à l'aide du modèle VHDL présenté à l'annexe B.

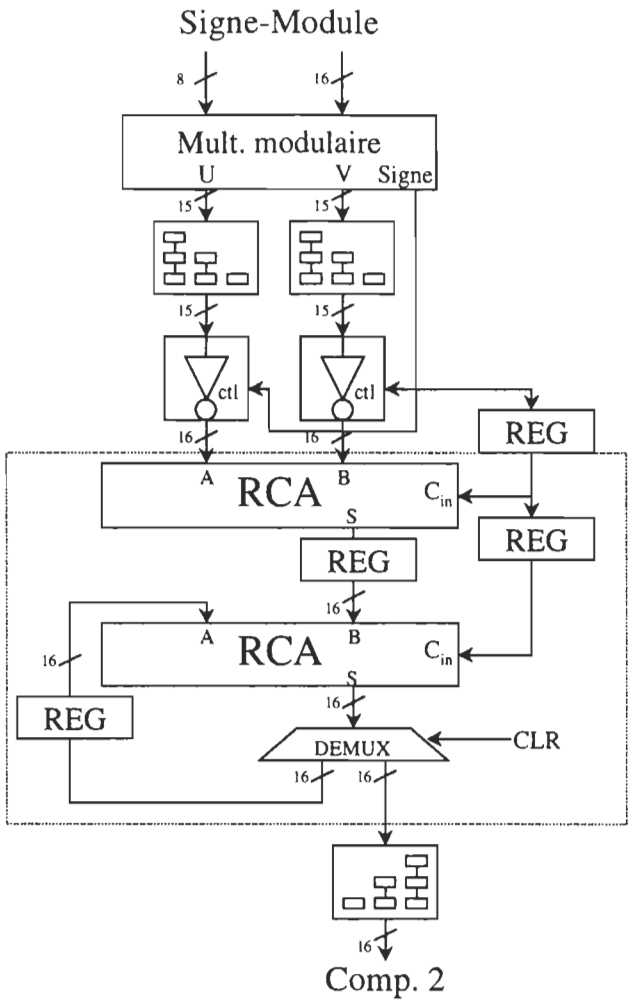


Figure 4.11 : Le MAC pipeline 8X16.

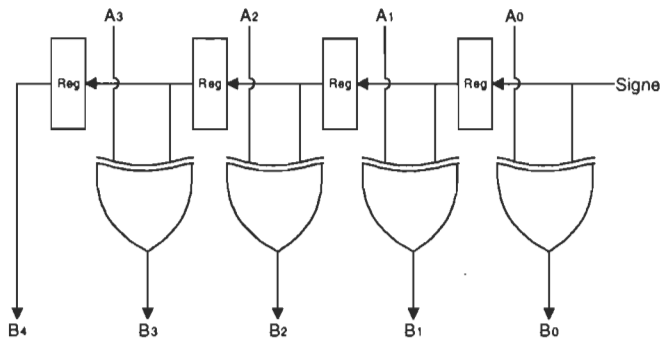


Figure 4.12 : Schéma du complément à 1.

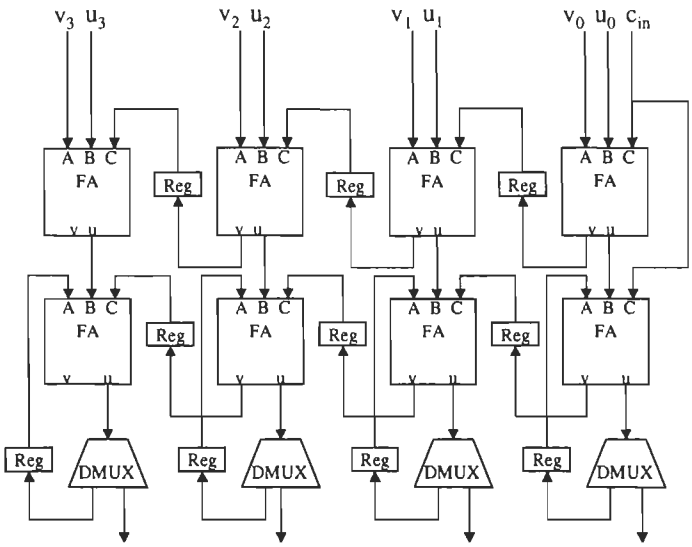


Figure 4.13 : Accumulation dans le MAC (4 bits).

4.3.3 Unités arithmétiques pour les nombres complexes

Les unités arithmétiques complexes peuvent être réalisées à partir de celles pour les nombres réels. L'addition complexe se fait à partir de l'addition des parties imaginaires et réelles, donc l'implantation peut se faire à l'aide de deux RCA.

La multiplication est un peu plus complexe. Elle correspond en fait à un produit matrice-vecteur décrit par:

$$C = AB = \begin{bmatrix} A_R & -A_I \\ A_I & A_R \end{bmatrix} \begin{bmatrix} B_R \\ B_I \end{bmatrix} = \begin{bmatrix} C_R \\ C_I \end{bmatrix} \tag{4.5}$$

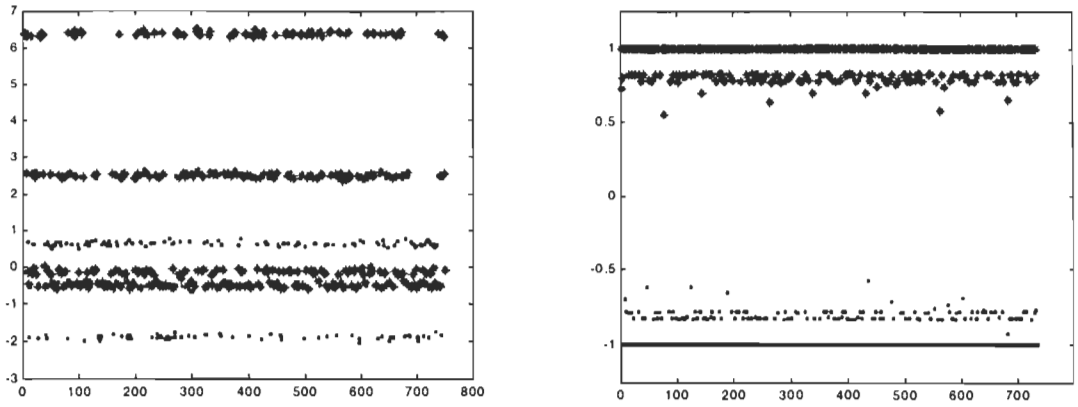
Il est donc possible de l'implanter à partir d'un MAC. Il y a des techniques plus avancées de mise en œuvre en technologie ITGE de multiplieurs complexes comme l'utilisation de l'arithmétique redondante [SHI98]. Cette technique permet une faible surface d'intégration et une faible consommation de puissance.

4.4 Performance de l'architecture proposée

Tout d'abord, une étude de quantification a été réalisée sur la convergence de l'algorithme car celle-ci est la plus affectée par cette quantification. Comme nous désirons un grand débit de transmission, le convertisseur analogique-numérique de 8 bits était une contrainte à respecter pour obtenir le temps de conversion escompté.

À partir des résultats de l'étude de quantification, un modèle VHDL (voir Annexe C) a été créé pour valider l'architecture lors d'un apprentissage hors ligne. Les poids de l'algorithme PL-MNN ont donc été calculé dans l'environnement Matlab® et insérer dans le modèle. La Figure 4.14 démontre bien que l'égalisation s'est effectué sans erreur sur un canal non linéaire dont le rapport signal-bruit a été fixé à 30 dB.

Tout d'abord, une étude de quantification a été réalisée sur la convergence de l'algorithme car celle-ci est la plus affectée par cette quantification. La Figure 4.15 montre qu'avec un convertisseur de 8 bits, une longueur de mots binaires de 16 bits offrent un bon compromis précision-suface.



a) b)

Figure 4.14 : Exemple de reconstition (b) avec un modèle VHDL de l'architecture à partir de la sortie d'un canal non linéaire (a).

Alors avec la mise en œuvre d'un MAC 8x16 bits, nous obtenons une fréquence d'horloge de 100 MHz pour la version non pipelinée et 500 MHz pour la version pipelinée. Le nombre de cycles nécessaires pour une correction avec M=2 et K=3, est de M+K+5 pour le processeur non pipeliné alors que la version pipelinée requiert M+L+35 cycle lors de l'apprentissage hors ligne. Dans la phase d'apprentissage en ligne, le nombre de cycle est majoré respectivement de 3 et 6 cycles. Pour chacune des architecture, le débit est de f/(M+1). Le résumé des performances obtenues dans une technologie CMOS de 0.5 µm de trouve au Tableau 4.1.

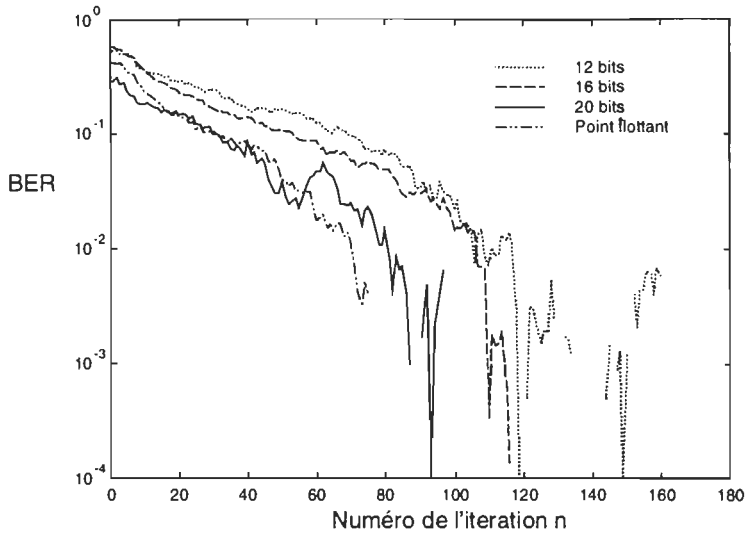


Figure 4.15 : Étude de quantification sur la convergence.

4.5 Conclusion

La proposition d'une architecture pour l'algorithme PL-MNN a été faite. Avec sa version pipelinée, elle accepte des débits de transmission jusqu'à 167 méga-bits par seconde (Mbps). Pour ce faire nous avons proposé une topologie de MAC, qui constitue le goulot d'étranglement de l'architecture. Cependant, cette estimation de débit est un peu optimiste puisque le logiciel de synthèse numérique Synopsys ® ne tient pas compte des délais dus à la propagation de l'horloge donc en réalité, la fréquence de fonctionnement devrait être autour de 400 MHz pour donner un débits de transmission de l'ordre de 133 Mbps.

De plus, la structure du MAC proposée permet l'application de la technique du pipeline par vague due à sa réduction de dépendance [MOR99]. Cette technique peut être aussi appliquée à l'ensemble de l'architecture afin d'optimiser le débit de transmission avec un apprentissage hors ligne [MOR99B], [MOR99C].

Tableau 4.1 : Performances des architectures

Architecture	Mode	Fréquence d'horloge [MHz]	Latence [µs]	Débit [MHz]
Version non pipelinée	Hors ligne	100	0.11	33
	En ligne		0.11	7
Version pipelinée	Hors ligne	500	0.08	167
	En ligne		0.08	11.3
DSP56001	Hors ligne	20	1.2	8.3
	En ligne		2.2	0.45

Enfin, avec un ajout minimal, une version pour l'auto-égalisation peut être mise en œuvre en se basant sur la méthode du gradient stochastique montré à la Figure 2.11. Aussi, l'architecture proposé peut accepté l'adaptation dirigée par la décision sans aucun ajout pour suivre les variations du canal de transmission comme le montre la figure 2.12.

Chapitre 5

Conclusion

Dans les communications numériques, les transmissions dans des bandes en fréquences limitées créent des interférences entre les symboles. Ces interférences faussent l'information reçue. Il peut s'agir d'une inversion de polarité lors de transmission utilisant la modulation PAM et un changement de phase lors de communication de signaux modulés QPSK. L'égalisation de canaux a donc pour but d'estimer la séquence transmise à partir du signal disponible à la réception. L'objectif de ce travail était de proposer une architecture parallèle d'un algorithme d'égalisation de canaux non-linéaire en vue d'une mise en œuvre ITGE. Du côté algorithmique, les méthodes basées sur les RNA ont été étudiées et choisies. L'algorithme PL-MNN (*Piecewise Linear Multilayer Neural Network*) a été développé et testé, ce qui est présenté au chapitre 3, afin de répondre aux contraintes d'intégration.

Les caractéristiques des émetteurs et des récepteurs font en sorte que les modèles de canaux sont souvent non-linéaires, rendant les méthodes conventionnelles d'égalisation

utilisant un filtre transversal linéaire avec adaptation LMS ou RLS moins efficaces. Aussi, dans la littérature, la proposition d'architectures ITGE dédiées à l'égalisation de canaux est plutôt rare alors que la demande en débit de transmission élevé s'est accrue par l'arrivée des applications comme la communication sans fils, la télévision numérique haute résolution, etc..

Le travail présenté propose une solution intéressante à la problématique de l'égalisation de canaux pour des communications à débit élevé. La nécessité de réaliser les calculs en temps réel au cours d'une transmission numérique justifie le développement d'une architecture efficace en vue du intégration en technologie ITGE. C'est pourquoi une architecture hautement parallèle (systolique) a été proposée, voir chapitre 4, tout en considérant la possibilité d'appliquer la technique du pipeline pour augmenter le débit de l'architecture.

Les objectifs de recherche fixés ont été atteints suivant une méthodologie basée sur les deux aspects suivants: tenir compte des contraintes et critères au niveau de l'ITGE durant le développement algorithmique; tenir compte des contraintes et critères au niveau algorithmique durant le développement architecturale.

Cette méthodologie nous a mené à la proposition d'une architecture ITGE dédiée à un algorithme d'égalisation de canaux non linéaire. Cet algorithme est basé sur un réseau de neurones artificiels utilisant une topologie multicouche avec une fonction de décision de type canonique linéaire par morceaux nommée PL-MNN. La méthode proposée permet l'égalisation de canaux non-linéaires contrairement aux méthodes utilisant un filtre transversal linéaire avec adaptation LMS et RLS. Même dans le cas de canaux linéaires,

nous avons démontré que la méthode d'égalisation proposée converge plus rapidement et est plus robuste au niveau du bruit entachant les données à la réception. De plus, l'algorithme PL-MNN possède une faible complexité de calcul considérant que dans la plupart des cas, cinq (5) neurones sur la couche cachée suffisent pour assurer l'adaptation. Dans l'évaluation de la complexité il faut noter que du point de vue intégration sur silicium l'algorithme d'apprentissage n'effectue aucune division. Dans le but d'augmenter le débit de l'architecture nous avons évité les récurrences au niveau de la topologie du réseau de neurones. Cette considération a eu pour conséquence un affaiblissement de la robustesse de la méthode au profit d'une augmentation significative du débit de l'architecture obtenue. Des simulations avec des données synthétiques ont été réalisées dans l'environnement Matlab® en utilisant l'outil "*Application Program Interface*" permettant d'utiliser le langage de programmation C++ pour les calculs et Matlab® pour la mise en graphique des résultats. Ceci nous a permis de présenter une étude comparative complète de la méthode PL-MNN par rapport à d'autres méthodes comme le filtre transversal linéaire avec adaptation LMS et RLS et la méthode PL-RNN (*Piecewise Linear Recurrent Neural Network*) [LIU98].

L'architecture proposée étant de type systolique, elle profite de ses avantages comme la modularité, la régularité, etc.. Le débit en apprentissage hors ligne n'est nullement dépendant du nombre de neurones sur la couche cachée ce qui peut devenir un net avantage dans le cas de canaux stationnaires et invariants. La validation de l'architecture s'est faite grâce à un modèle VHDL dans l'environnement Mentor Graphics® et l'évaluation des performances a été possible grâce à l'outil Synopsys® et à la technologie CMOS de 0.5 µm (CMOSIS5), tous deux fournis par la société canadienne de microélectronique (SCM).

Dans cette technologie, l'architecture peut atteindre un débit de 167 Mbps lorsque deux (2) retards sur la sortie du canal de transmission sont considérés et en utilisant une modulation PAM. L'application de la technique du pipeline s'est vue réalisée en proposant une nouvelle structure de multiplieur-accumulateur (MAC) qui accepte l'application d'un pipeline profond grâce à sa réduction de dépendances dans les données à accumuler.

Ce projet apporte une solution intéressante pour la problématique de mise en œuvre de l'égalisation de canaux numériques à haut débit. Son introduction dans les systèmes de communication numérique assure une transmission de données plus fiable et plus rapide et ce, à un coût raisonnable. Il faut ajouter que l'architecture proposée peut être développée pour obtenir une version non supervisée (*blind equalization*) et une adaptation dirigée par la décision.

Ce projet a permis la participation à quatre (4) conférences [VID99], [VID99B], [VID99C] et [MOR99] (voir annexe A), un symposium [MOR99C] et un article de journal a été rédigé [VID99D] pour publier la majeure partie du travail présenté. Les publications [VID99], [VID99B] et [VID99D] concernent la présentation de l'architecture simplifiée pour l'algorithme d'égalisation PL-MNN utilisé pour la communication en bande de base. Le multiplieur-accumulateur utilisant un pipeline synchrone y est aussi présenté. La publication [VID99C] présente l'algorithme d'égalisation de canaux PL-MNN et son architecture associée pour la communication avec modulation de type QPSK utilisant les nombres complexes. La possibilité d'appliquer la technique du pipeline par vagues sur l'architecture et le multiplieur-accumulateur a été démontrée dans [MOR99], [MOR99B] et [MOR99C]. De plus, la présentation [MOR99C] s'est vue décerner le prix "PMC-Sierra

High-Speed Networking & Communications Award" au symposium sur la R&D de la micro-électronique au Canada le 14 juin 1999 à Ottawa pour sa contribution au niveau des réseaux et des communications haute vitesse.

Bibliographie

[ADA97] Adali, T.; Liu, X.; Sonmez, M.K, " Conditional distribution learning with neural networks and its application to channel equalization", IEEE Transactions on Signal Processing, Avril 1997,pp. 1051-1064.

[AYA98] Ayanoglu. E. and al.,"An equalizer design technique for the PCM modem: A new modem for the digital public switched network", IEEE Transactions on communications, Juin 1998, pp. 763-774.

[BEN95] Benelli, G.; Favalli, L.; Gatta, M.; Mecocci, A., " QPSK receiver based on recurrent neural networks", European Transactions on Telecommunications, Juillet 1995, pp. 455-462.

[CHA95] Chang, P-R, Wang, B-C, "Adaptive Decision feedback equalization for digital satellite channel usinf multilayer neural networks", IEEE Journal on Selected Areas in Communications,Février 1995, pp. 316-324.

[CHE95] Cherubini, G.; Olcer, S.; Ungerboeck, G., "A quaternary partial-response class-IV transceiver for 125 Mbit/s data transmission over unshielded twisted-pair cables: principles of operation and VLSI realization", IEEE Journal on Selected Areas in Communications, Dec. 1995.

[CHI96] Chin-Teng Lin; Chia-Feng Juang," An adaptive neural fuzzy filter and its applications", Proceedings of the Fifth IEEE International Conference on Fuzzy Systems. FUZZ-IEEE '96 (Cat. No.96CH35998), Septembre 1996.

[CHO93] Choi, J.; Bang, S.H.; Sheu, B.J.,"A programmable analog VLSI neural network processor for communication receivers", IEEE Transactions on Neural Networks, Mai 1993.

[HAY96] Haykin, S., Adaptative filter theory, Prentice Hall, 1996.

[HIT98] Hitachi, HD49430F: Preliminary Specification, Avril 1998, 43 pages.

[JOH97] Johns, D.A. and Martin, J., Analog Integrated Circuit Design, John Wiley & son, 1997, 706 pages.

[KEC91] G.Kechriotis and E.S.Manolakos, "AVLSI architecture for the on-line training of recurrent neural networks", Proc. IEEE Asilomar Conference on Signals, Systems and Computers, November 1991, pp.506-510.

[KEC94] Kechtiotis, G., Zervas, E., Manolakos, E.S., "Using Recurrent Neural Networks for

Adaptive Communication Channel Equalization", IEEE Transaction on Neural Networks, Vol. 5, No. 2, Mars 1994, pp. 267-278.

[KEC94b] G.Kechriotis and E.S.Manolakos,"Training fully recurrent neural networks with complex weights", IEEE Transaction on circuits and systems II, Mars 1994, pp.235-238

[KYU98] Kyung-Wook Shin; Bang-Sup Song; Bacrania, K.,"A 200-MHz complex number multiplier using redundant binary arithmetic", IEEE Journal of Solid-State Circuits, June 1998.

[LEU91] Leung, H. Haykin, S.,"The complex backpropagation algorithm", IEEE Transactions on Signal Processing, Septembre 1991, pp. 2101-2104.

[LIU98] X. Liu, T. Adah and L. Demirekler, "A Piecewise Linear Recurrent Neural Network Structure and its Dynamics", ICCASP 98, Seattle, 1998, pp.1221-1224.

[LO92] Lo, N.W.K.; Hafez, H.M.," Neural network channel equalization", IJCNN International Joint Conference on Neural Networks (Cat. No.92CH3114-6)., Juin 1992.

[MAC98] Macchi, O., "L'égalisation numérique en communication", Ann. Télécommunications, vol.53, no. 1-2, 1998, pp.39-58.

[MOR99] F. Morin, M. Vidal et D. Massicotte, "Intégration d'un MAC en utilisant la technique du pipeline par vagues", Congrès de l'ACFAS, Université d'Ottawa, mai 1999.

[MOR99b] F. Morin, M. Vidal, et D. Massicotte, "A High Throughput Architecture for Channel Equalization Based on Neural Network Using Wave Pipeline Method", CCECE'99, Edmonton, Mai 1999.

[MOR99c] F. Morin, M. Vidal, et D. Massicotte, "A Wavepipeline Architecture For High Speed Nonlinear Channel Equalization", Texpo'99, Ottawa, 1999.

[NAI97] Nair, S.K.; Moon, J, "Data storage channel equalization using neural networks", IEEE Transactions on Neural Networks, Septembre 1997, pp. 1037-1048.

[NOR95] Tomas Nordström, Highly parallel computer for artificial neural networks, thèse de doctorat, Lulea University of Technology, division of Computer Science and Engineering, Suède, 1995.

[PAR95] Park, J.C.; Mittal, R.; Bracken, K.C.; Carley, L.R.; Allstot, D.J, "High-speed CMOS current-mode equalizers", 1995 IEEE Symposium on Circuits and Systems (Cat. No.95CH35771)., Avril 1995,

[PAR97] Parisi, R. et al., "Fast Adaptive Digital Equalisation by Recurrent Neural Networks", IEEE Trans. on Signal Processing, Vol. 45, No. 11, Nov. 1997, pp

2731-2739.

[PAT94] Patra, J. C. , Pal, R. N. ,"A functional link artificial neural network for adaptive equalization", Signal Processing, Fevrier 1995. Pp. 181-195.

[PRO95] Proakis, J.G., Digital Communications, 3e Edition, Mcgraw-Hill, 1995,928 pages.

[SHA98] Shanbhag, N.R.; Gi-Hong Im, "VLSI systems design of 51.84 Mb/s transceivers for ATM-LAN and broadband access", IEEE Transactions on Signal Processing, May 1998, pp. 1403-1416.

[SHI98] Shin,K-W, Song, B-S, Bacrania, K.,"A 200-MHz Complex Number Multiplier Using Redundant Binary Arithmetic", IEEE Journal of Solid-State Circuits, June 1998, pp. 904-909.

[SIM98] Simard, R., Jacob, A., Télécommunication, notes de cours, UQTR, 1998.

[SWE98] Sweatman, C.Z.W.H.; Mulgrew, B.; Gibson, G.J. "Two algorithms for neural-network design and training with application to channel equalization", IEEE Transactions on Neural Networks, Mai 1998,pp. 533-543.

[VEC99] Vecci L, Campolucci P, Piazza F, "Complex-valued neural networks with adaptive spline activation function for digital radio links nonlinear equalization ",IEEE Transaction on neural networks, Février 1999, pp. 505-514.

[VID98] Vidal, M, L'égalisation de canaux par un réseau de neurone: Algorithme et implantation. Rapport de recherche pour le Groupe de recherche en électronique industrielle (GREI), UQTR, 47 pages,1998.

[VID98b] Vidal, M. , Modélisation en VHDL d'un réseau de neurones artificielles. Rapport de recherche GREI UQTR, 44 pages, 1998.

[VID99] M. Vidal et D. Massicotte, "A Parallel Architecture of a Piecewise Linear Neural Network for Channel Equalization", IEEE-IMTC/99, Italie, 1999.

[VID99b] M. Vidal et D. Massicotte, "Algorithmes et architectures systoliques pour l'égalisation de canaux", Congrès de l'ACFAS, Université d'Ottawa, mai 1999.

[VID99c] M. Vidal et D. Massicotte, " QPSK Equalizer Based on Piecewise Neural Network", International Conference on Wireless Communications (Wireless'99), Calgary, Juillet, 1999.

[VID99d] M. Vidal et D. Massicotte, "A VLSI Parallel Architecture of a Piecewise Linear Neural Network for Nonlinear Channel Equalization", soumis à IEEE Transaction on Instrumentation and Measurement, en mai 1999.

[WAN95] Wang, C.-K.; Shiue, M.-T. "A CAP-based ADSL transceiver VLSI architecture design ",1995 International Symposium on Communications, Décembre 1995.

[YOU98] You, C. , Hong, D., " Nonlinear blind equalisation schemes using complex-valued multilayer feedforward neural network", IEEE Transaction on neural networks, Novembre 1998, pp. 1442-1455.

Annexe A : Publications

[MOR99b] F. Morin, M. Vidal, et D. Massicotte, "A High Throughput Architecture for Channel Equalization Based on Neural Network Using Wave Pipeline Method", CCECE'99, Edmonton, Mai 1999.

[VID99] M. Vidal et D. Massicotte, "A Parallel Architecture of a Piecewise Linear Neural Network for Channel Equalization", IEEE-IMTC/99, Italie, 1999.

[VID99c] M. Vidal et D. Massicotte, " QPSK Equalizer Based on Piecewise Neural Network", International Conference on Wireless Communications (Wireless'99), Calgary, Juillet, 1999.

A High Throughput Architecture for Channel Equalization Based on a Neural Network Using a Wave Pipeline Method

Frédéric Morin, Martin Vidal and Daniel Massicotte

Electrical Engineering Department, Université du Québec à Trois-Rivières
Research Group on Industrial Electronics
C.P. 500, Trois-Rivières, Québec, Canada, G9A 5H7
Tel.: 1-(819)-376-5071, Fax : 1-(819)-376-5219
Email : {Frederic_Morin, Martin_Vidal, Daniel_Massicotte}@uqtr.uquebec.ca

ABSTRACT

The use of a wave pipelining method for the design of a systolic architecture dedicated to channel equalization is proposed. A description is given of the piecewise linear multilayer neural network (PL-MNN) algorithm and the architecture. To improve the throughput of the architecture we propose a wave-pipelined version of the multiplier-accumulator (MAC) the presents the bottleneck of the architecture. A 16×8-bit MAC is performed using a Normal Process Complementary Pass Transistor (NPCPL) as a universal cell for the creation of conventional logic gates and is used to optimize the wave pipelined MAC. The throughput and the latency of the MAC have been evaluated at 650 MHz and 8 ns respectively. The performance has been evaluated in a 0.5 μm CMOS technology in comparison with the systolic architecture with and without a conventional pipeline and the proposed wave pipeline structure.

1. INTRODUCTION

The field of communications (e.g. satellites, modems, and cellular phones) often uses methods of compression to increase transmission speed. Nevertheless, these techniques do not satisfy increasing demands for the high throughput communication of data. Thus, channel equalization is interesting because it supplements these methods of digital communication by having a direct impact on the data flow.

On the whole, the forward model of the nonlinear channel can be applied as follows

$$\{\tilde{y}(n)\} = \Im[\{x(n)\}, \Theta] + \eta(n) \qquad (1)$$

where the channel's output $\tilde{y}(n)$ is a scalar corrupted with additive noise $\eta(n)$ and Θ represents the vector of the parameters of the operator $\Im$. The problem of nonlinear channel equalization is to solve the reconstruction problem, which consists of a regularized inversion of the operator $\Im$, i.e., an operator of reconstruction $\Re$

$$\{\hat{x}(n)\} = \Re[\{\tilde{y}(n)\}, \Theta] \qquad (2)$$

The measurement of $\tilde{y}(n)$ leads us to deduce the value of the estimated entry $x(n)$ through a corrective processor called a channel equalizer that makes it possible to obtain $\hat{x}(n)$.

Since data communication is in constant evolution, the need for fast systolic architecture is always in demands and becomes a necessity. To improve the throughput of VLSI architectures [1], [6], the wave pipeline method is proposed to bring higher transmission rate in a nonlinear channel. This method is applied on the VLSI architecture proposed in [6], using the NPCPL cell as basic logic gates [2]. The multiplier-accumulator (MAC) represents the principal combinatory logic block (CLB) unlikely it acts as a bottleneck.

In Section 2, we discuss the adaptive PL-MNN algorithm for channel equalization, and in Section 3, we describe the systolic VLSI architecture dedicated to this algorithm. In Section 4, we present the structure of the MAC and the wave-pipeline technique using the NPCPL cell for universal logic. A comparison study of the architecture both with and without a conventional pipeline and with a wave pipeline is performed in Section 5. Our conclusions are presented in the final section.

2. MULTILAYER NEURAL NETWORK ALGORITHM

There are many algorithm propositions of channel equalization in the literature. Some are for linear channel [3], and others are for a nonlinear channel [4], [7]. However, some VLSI architectures are dedicated to nonlinear channel equalization [6]. In the paper [6], a

This work was supported by the Natural Science and Engineering Research Council of Canada and by Le Fonds FCAR Québec.

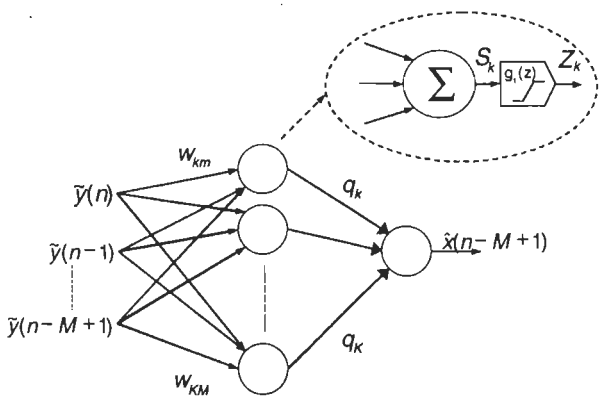


Figure 1: The structure of the neural network (PL-MNN).

systolic array architecture that uses the recurrent equations of the neural network is proposed. Therefore, these equations were found to have a regular and recursive form produces a local communication of data. Fig. 1 depicts the piecewise linear multilayer neural network (PL-MNN) representation of the algorithm of correction. In the case of an invariant and stationary channel, we need only consider the propagation of the data $\tilde{y}$ through the neural network to get the result of the correction. The following relation represents the algorithm of correction:

$$\hat{x}(n-M+1) = g\left(\sum_{k=1}^{K} q_k\, g(s_k)\right) \tag{3}$$

and

$$s_k = \sum_{m=1}^{M} w_{km}\, \tilde{y}(n-m+1) \text{ for } k = 1,2,\ldots,K \tag{4}$$

where $g(s)$ defines the piecewise linear function:

$$g(s) = \frac{|s+4| - |s-4|}{8} \tag{5}$$

The value of M depends on the frequency characteristics of the channel, and the numbers of hidden neurons K are tuned for optimized algorithm performance.

3. VLSI IMPLEMENTATION OF THE PL-MNN

To obtain a VLSI architecture for channel equalization at high throughput, we used the wave pipeline method on this architecture. The on-line learning will not be included, because of the difficulty of incorporating a recurrence equation in the architecture when using wave pipelining.

Fig. 2 shows the systolic architecture, and the elementary processor (PE_k) [6]. The architecture has an array of $\tilde{y}(n-m)$ for an input with a maximum stage of

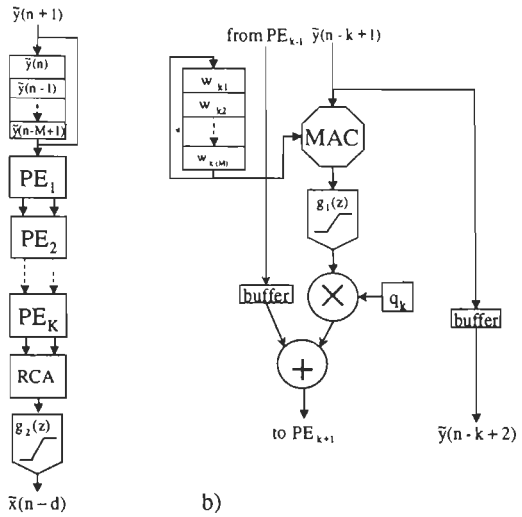


Figure 2: The systolic architecture of PL-MNN algorithm (a) and the processing element PE_k (b).

M+1 delay. The signal $\tilde{y}$ introduced inside the array is loop until the signal $\tilde{y}(n)$ has crossed the PE_1. The last step is to add up the partial products from each elementary processor, before they pass through the activation function. The combinatory logic blocks use a full adder (FA) cell for the main structure.

All PE blocks are identical, and a number of hidden neurons K is fixed to assure the convergence of the equalizer performance. Inside the PE, we must use an array containing the weight w_{km} in the hidden layer since we are using only off-line learning. These will then multiply and accumulate with the output of the channel $\tilde{y}(n)$, where the octagonal shape represents the MAC. Thereafter, s_k will pass through the activation function and give z_k, which is multiplied with the weight of the neural output q_k. Finally, the output will be partially added to the output value of the PE_{k-1}. A study of the data quantization in the scale [-1,1] demonstrates that an 8-bit analog-to-digital converter quantify the input signal $\tilde{y}(n)$ and an 8 bits to 16 bits wordlength for the constants (w_{km}, q_k) and variables $(s_k, z_k, \hat{x}_k)$ data respectively present a good tradeoff for the VLSI implementation [6].

4. THE INTEGRATION OF A WAVE PIPELINED MULTIPLIER-ACCUMULATOR

The wave pipeline, like the conventional pipeline, is an optimization method of the flow inside the CLB. The pipeline technique introduces a number N registers inside a CLB to allow the division of the structure into several small parts, which establishes the order of depth of the pipeline. The principle of the wave pipeline

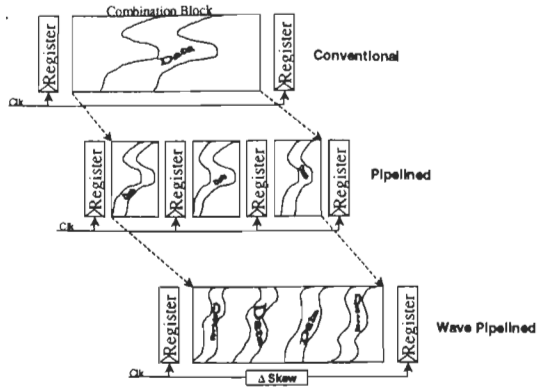


Figure 3: Concept of wave pipeline structure

introduces inside the CLB a number of waves of data superior to the propagation delay through the combinatory logic. This allows one to reduce the difference between the maximum and minimum propagation delays of the CLB without using internal registers. As result, a concrete reduction of integration surfaces is obtained if we suppose an equivalent performance and an increase in the data flows relative to the number of introduced waves [5]. However, this method of implementation remains complex and requires the use of advanced technology with respect to the delay of transistors (fine-tuning) and connections (coarse tuning). Unlike the conventional pipeline, which aims to minimize the length of path between registers, the wave pipeline aims to equalize it [1]. In other words, the shortest path must be almost equal to the longest. Fig. 3 depicts both types of pipeline methods.

One of the solutions adopted to deal with the problem generated by the transmission of logic is the use of a cell called the NPCPL (Normal Process Complementary Pass Transistor Logic), shown in Fig. 4. Indeed the cell NPCPL makes it possible to balance the delays granted the logical gates as explained in [2]. To apply the wave pipeline method, each logic gate (NAND, AND, OR, NOR, XOR, DELAY, etc) consists of one NPCPL cell. Moreover, the optimization generated by the use of the wave pipeline, unlike the conventional pipeline, makes possible a certain decrease in power consumption, because the conventional pipeline must use a synchronous combinatory logic, which implies a higher number of transitions. Of course, there is the problem of clock skew during its propagation inside the CLB. In a wave pipeline, therefore, we use an intentional clock skew, or a control skew (CS) to reduce the effect of non-desirable path delays. Nevertheless, the addition of a buffer to equalize the paths may increase the surface integration compared with a conventional pipeline.

For the suggested architecture, the use of a multiplier-accumulator shown in Fig. 5 makes it possible

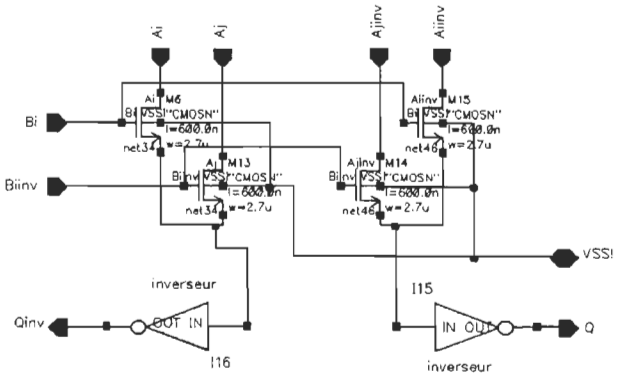


Figure 4: Structure of the NPCPL

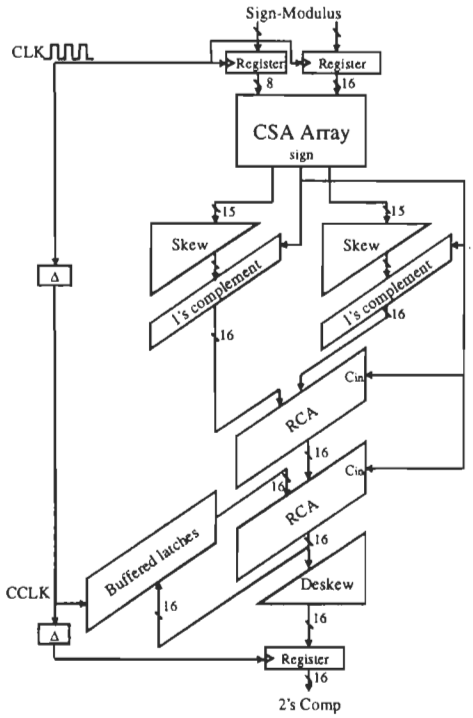


Figure 5: Structure of the MAC

to increase the data flow associated with the processing of the equalizer. The structure of the MAC permits a visualization of the flow and waves propagation inside the MAC arithmetic. To reduce both the surface area and latency we only use the 16 first significant bits (MSB) during the accumulation feedback. The arithmetic begins in sign-modulus and end in 2's complement. We used a temporal control skew and deskew before and after the arithmetic as proposed in [9]. A modified carry save adder (CSA+) structure used for this implementation is proposed in [2], where Fig. 6 represents the cell used during the multiplier array or CSA array. The regularity and low dependency of data justify the use of CSA array went applying the wave pipeline method. This CSA+ cell produces an equal path throughout the multiplier structure.

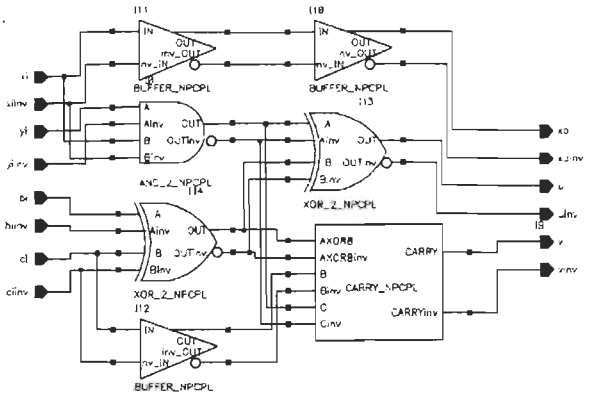


Figure 6: CSA+ structure

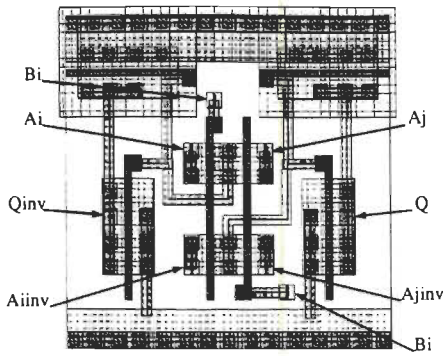


Figure 7: Layout of the NPCPL cell

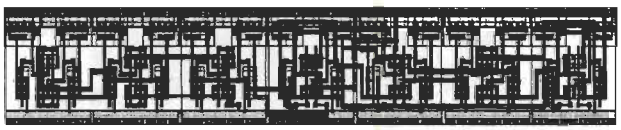


Figure 8: Layout of the CSA+

The addition at the end of the partial sum (u, v) is generally called the ripple carry adder (RCA), where the CSA inside are modified so the delay of the sign propagation is equalize. To use the ripple carry adders, we must first pass through a triangular skew made of a singular buffer to synchronize data. Since we are in sign-modulus, we use the sign (MSB) to see if it is necessary to transform the value by a 2's complement conversion if the value is supposed to be negative. The sign value linked to the carry of the RCA will then complete the necessary conversion. The output is then is deskewed by a triangular topology of buffer [9].

The MAC used to resolve Eq. (4), represents the bottleneck of this architecture; therefore, the use of a high throughput method is needed. What constitutes the advantages of this MAC is the use of the carry save adder (CSA) cell instead of the CLB. The cell is defined by the following equation [10]:

$$u = a \oplus b \oplus c \tag{6}$$

$$v = ab + bc + ac \tag{7}$$

where a,b and c are the added bits.

The modified CSA cell use mentioned earlier in [2], used the equations (6) and (7) and added a logical AND for the half adder.

The accumulation is realized one bit at a time since the bits have been delayed by the triangular skewed. Indeed, this technique helps to reduce the data dependency, and permits finer tuning when equalizing the paths for the wave pipeline. The registers used to synchronize the data in the feedback make it possible to use the wave pipelined MAC for different values of clock frequency. Since the CLB requires a certain data dependency, which means that the accumulation of data i and i+1... is necessary, the possibility of a structure without a latch (register) in the feedback is almost impossible. There is one exception where the control of data and clock must be restricted to a special frequency, but it is unacceptable to obtain a low sensitivity to fabrication process. Without data dependency, we believe that our architecture gives a good regularity and the shortest and longest paths are almost equal. So, the shorter path from wave j+1 does not meet the longer path from wave j.

5. Comparison Study and Performance Evaluation

The performance of the proposed architecture was done using a 3.3 V, single poly, triple metal 0.5 µm CMOS technology from the Canadian Microelectronics Corporation (CMC) with Cadence® tools. Simulation of the multiplier-accumulator was performed using a SpectreS™ simulator. To obtain a more accurate level of simulation, the layout of the NPCPL and CSA cells was made, and we have extracted all the parasitic and inter-layer coupling capacitance. Those layouts are represented in Figs. 7 and 8. The surface used by the NPCPL is 745 µm^2 and that for the CSA 5209 µm^2. The NPCPL cell has advantages, but is restrictive. Disadvantages are first, the necessity to give the inverse of all data being treated, and therefore to upset the path delay by one inverter (i.e. $\bar{a}+\bar{b} \Rightarrow (ab)$) from the beginning. Another problem is the limited fanout capability, where each gate cannot drive more then two other gates if we want a substantial fast slope. Those factors are less significant than the advantages of the NPCPL and may be avoided, even if more buffers are needed.

A test was made on a 4×4-bit MAC to lighted-up simulation and tuning time. The output of the MAC, s3:s0, is shown in Fig. 9. The master clock (CLK), the reset signal (RESET) and the intentional clock skew

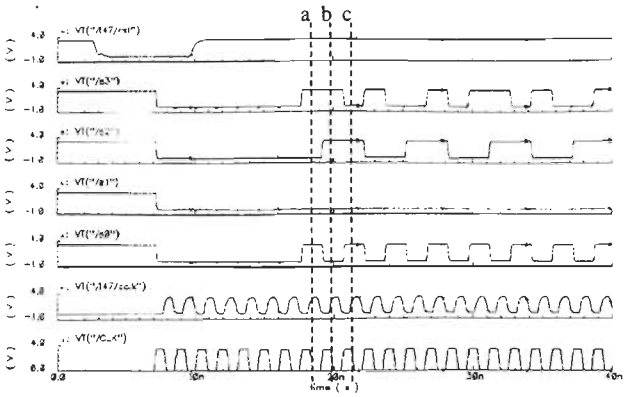


Figure 9: Results of the 4×4-bit MAC with a clock frequency of 650 MHz, where a=1101_b×1100_b+0, b=0101_b×1010_b+a, and c=1101_b×1100_b+b, etc...

(cclk) of the MAC are indicated. We observed from Fig. 9 a good quality of synchronization and shape of the output waves. The speed of the 16×8-bit MAC was simulated at 650 MHz. A comparison study was made with the non-pipeline and conventional pipeline MAC structures. Table I represent the maximum clock frequency, number of transistors needed for each method and latency.

The performance of the systolic architecture (Fig. 2) is evaluated in terms of throughput and latency. The throughput is evaluated to f/M with and without a conventional pipeline and with a wave-pipeline. The latency is evaluated to $(M+K+35)/f$ and $(M+K+5)/f$ with and without a pipelined MAC respectively, and to $M/f+(K+16)\tau_{buf}$+24 ns with the wave-pipelined version, where τ_{buf} is the delay of one NPCPL buffer (τ_{buf}=0.3 ns).

6. Conclusion

In this paper we have presented a wave-pipelined multiplier-accumulator and a systolic architecture of a piecewise linear multilayer neural network for nonlinear channel equalization using 0.5 µm CMOS technology. The properties of recursiveness, regularity, and localized communication of the systolic implementation in an off-line adaptation of the equalizer have made it possible to apply a faster method of optimization of the throughput using the wave-pipelining method. The validation of the proposed 16×8-bit multiplier-accumulator using a NPCPL cell is made using Cadence® tools. The NPCPL cell makes it possible to equalize the paths of the combination logic blocks. The maximum clock rate obtained for a 16×8 bits is 650 MHz and a latency of 8 ns. The number of waves is evaluated at 5 in the MAC structure. The proposed multiplier-accumulator and systolic architectures can be applied in high throughput architectures and communication systems.

Table I: Performance comparison of the 16×8-bits MAC.

	Max. Clock	# Transistors	Latency
Conventional	100 MHz	3500	10 ns
Pipelined*	500 MHz	15000	52 ns
Wave-pipelined	650 MHz	12000	8 ns

* Results obtained with Synopsys® tools without considering the delay of registers and the clock skew, it corresponds to an overestimate.

References

[1] W.P. Burleson, M. Ciesielski, F. Klass, W. Lui, "Wave-Pipelining : A Tutorial and Research Survey", *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Vol. 6, No. 3, September 1998, pp.464-474.

[2] D. Ghosh, S. K. Nandy, "Design and Realisation of High-Performance Wave-Pipeline 8 × 8 b Multiplier in CMOS Technology", *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Vol. 3, No. 1, March 1995, pp.36-48.

[3] S. Haykin, *Adaptive filter theory*, Prentice Hall, 1991, pp. 342-347;492-497.

[4] G. Kechtiotis, E. Zervas, E. S. Manolakos, "Using Recurrent Neural Networks for Adaptive Communication Channel Equalization", *IEEE Transaction on Neural Networks*, Vol. 5, No. 2, Mars 1994, pp. 267-278.

[5] K. J. Nowka, "High-Performance CMOS System Design Using Wave Pipelining", *Technical Report CSL-TR-95-XXX*, Stanford University, August 1995.

[6] M. Vidal and D. Massicotte, "A Parallel Architecture of a Piecewise Linear Neural Network for Channel Equalization", Accepted for publication in *Instrumentation and Measurement Technology Conference (IMTC'99)*, Venice, May 1999.

[7] X. Liu, T. Adah and L. Demirekler, "A Piecewise Linear Recurrent Neural Network Structure and its Dynamics", *International Conference on Acoustics, Speech and Signal Processing (ICASSP'98)*, Seattle, 1998, pp.1221-1224.

[8] D. Massicotte and My B. Elouafay, "A Systolic Architecture for Kalman-Filter-Based Signal Reconstruction Using the Wave Pipeline Method", *5th IEEE International Conference on Electronics, Circuits and Systems (ICECS'98)*, Portugal, Lisbon, 7-10 Sept 1998, pp.199-202.

[9] S-J. Jou, C-H. Chen, E-C. Yang, and C-C. Su, "A Pipelined Multiplier-Accumulator Using a High-Speed, Low-Power Static and Dynamic Full Adder Design", *IEEE Journal of solid-state circuits*, Vol. 32, No. 1, January 1997.

[10] V.K. Madisetti, "VLSI Digital Signal Processors: An introduction to Rapid Prototyping and Design Synthesis", *IEEE Press*, 1995.

A VLSI Parallel Architecture of a Piecewise Linear Neural Network for Nonlinear Channel Equalization

Martin VIDAL and Daniel MASSICOTTE

Department of Electrical Engineering
Université du Québec à Trois-Rivières
C.P. 500, Trois-Rivières, Québec, Canada, G9A 5H7
Phone: 1-(819)-376-5071, FAX : 1-(819)-376-5219
E-mail: {Martin_Vidal, Daniel_Massicotte}@uqtr.uquebec.ca

Abstract

This paper proposes a systolic architecture based on a multilayer neural network (MNN) to solve the problem of the nonlinear channel equalization. This architecture is based on a piecewise linear multilayer neural network (PL-MNN) algorithm derived from a recursive version (PL-RNN) [7]. In place of a sigmoid function, both algorithms use a canonical piecewise linear function, which makes the MNN more suitable for a digital VLSI implementation. The PL-MNN algorithm is more suitable for a VLSI pipelined implementation. The pipeline technique is applied to obtain a high throughput circuit which can be used in a high speed adaptive channel equalization. A performance study on both linear and nonlinear channels is presented. A comparison of results obtained with two conventional methods (LMS and RLS) and the PL-RNN algorithm is presented, and a performance evaluation of the systolic architecture is done for a 0.5 μm CMOS technology.

1. Introduction

The adaptive channel equalization shown in Fig. 1 is a very common inverse problem in various fields of application, such as telecommunications, wireless communications, modems and measurement systems (e.g. [1], [2], [14]). Both linear and nonlinear channels are considered. In the case of a linear channel, the channel equalization problem is defined by a convolution operation of a signal $x(n)$ crossing a channel represented by an impulse response function $h(n)$. The produced output $\tilde{y}(n)$ is a scalar corrupted with additive noise $\eta(n)$. The discrete form of the convolution equation is:

$$\tilde{y}(n)=\sum_{m=1}^{M} h(n-m)x(m) + \eta(n) \quad \text{for } n=1,2,3,... \qquad (1)$$

This project was made possible by the support of the Fonds FCAR Québec.

In general, the forward model of the nonlinear channel can be applied as follows

$$\{\tilde{y}(n)\} = \Im[\{x(n)\},\Theta] + \eta(n) \qquad (2)$$

where Θ is the vector of the parameters of the operator $\Im$. The reconstruction consists of a regularized inversion of the operator $\Im$, i.e., an operator of reconstruction $\Re$

$$\{\hat{x}(n)\} = \Re[\{\tilde{y}(n)\},\Theta] \qquad (3)$$

To solve the problem of channel equalization, it is necessary to estimate the signal applied to the channel, $x(n)$, defined as the transmit signal, taking into account that it is a numerically ill-conditioned rule. The adaptive channel equalization problem is rather popular [3], [4] but the VLSI implementation of the algorithms is rarely considered [5], [6], [10]. Numerical methods are suitable for solving the problem under discussion; they are based on an a priori knowledge of the impulse response of the conversion system, $h(n)$, and of the measured samples of the output, $\tilde{y}(n)$. Solutions given by the LMS and RLS algorithms were proposed in [3] and by the Kalman filter in [4]. These algorithms, however, cannot be applied on a nonlinear channel, which is well treated with a neural-network-based method of correction [7], [8], [13].

A Piecewise Linear Recurrent Neural Network (PL-RNN) was proposed in [7]. Recurrent neural networks are able to equalize channel with deep spectral nulls [11] but the data dependencies reduce the throughput of implementation, and they are not suitable to the application of pipeline techniques. We propose a Piecewise Linear Multilayer Neural Network (PL-MNN) which has no data dependency. Consequently, the application of pipeline techniques can be done to maximize the throughput.

In Section 2, we present the PL-MNN algorithm. A description of the pipelined systolic architecture is presented in Section 3. Finally, in Section 4, a comparison study with LMS, RLS, and PL-RNN algorithms is

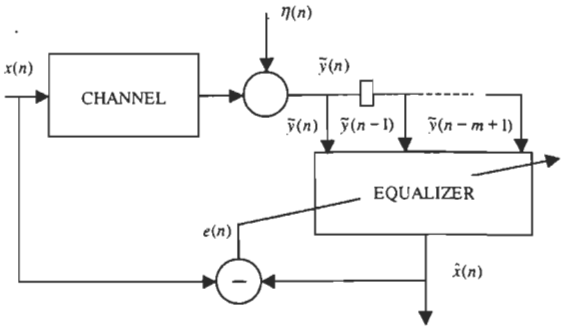


Figure 1: Adaptive channel equalization.

performed on the convergence speed, the robustness of noise, and the quantization analysis. A performance evaluation of the proposed architecture in 0.5 μm CMOS technology is also done. Section 5 gives the conclusion.

2. Description of Channel Equalization Algorithm Toward a VLSI Implementation

In many applications, the performance evaluation of the implementation is essentially characterized by computation speed, power consumption, miniaturization and quality of channel correction. The recurrent equations of the MNN formulation present good properties for VLSI implementation - such as localized communications, regularity and recursiveness - which allow for the use of a systolic approach. Several propositions for a high speed analog implementation of an MNN were published, but complete numerical devices for this application are unusual. The main difficulty is the sigmoid activation function which is hard to implement numerically [15], but which can be easily shaped with a single transistor. Otherwise, analog circuits lack precision, and outside factors easily disturb the operation of the device. A Hopfield network algorithm and parallel architecture were proposed in [8] and [6] respectively. This method uses a sigmoid function as the activation function, and the digital implementation of this function is difficult for a large number of neurons. This limitation is well defined in [9]. The piecewise function is more attractive for digital implementation to obtain a parallel architecture with a high throughput. A piecewise linear recurrent neural network (PL-RNN) was proposed in [7] to reconstruct the input signal of a linear or a nonlinear channel.

To reduce data dependencies in the data flow for the VLSI implementation, we remove all recurrences in the PL-RNN to obtain the piecewise linear multilayer neural network (PL-MNN) structure proposed in Fig. 2. The input of the PL-MNN is a vector composed by the output of the channel $\tilde{y}(n)$ and its previous values and the result of correction $\hat{x}(n)$ are obtained at the output with a delay of M samples.

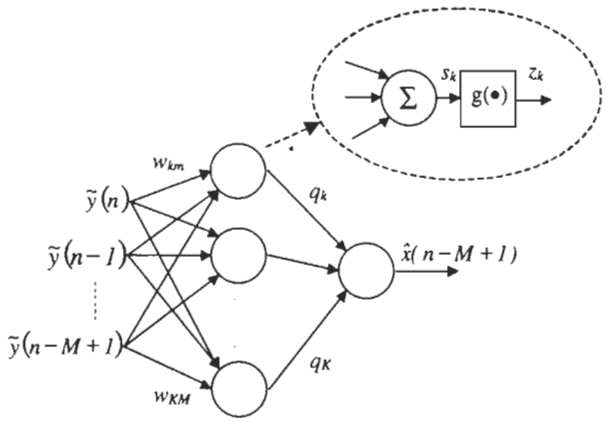


Figure 2: The structure of the neural network (PL-MNN).

The algorithm of correction consist of two steps:
- the propagation step, defined as follows

$$\hat{x}(n-M+1) = g\left(\sum_{k=1}^{K} q_k\, g(s_k)\right) \tag{4}$$

and

$$s_k = \sum_{m=1}^{M} w_{kj}\,\tilde{y}(n-m+1) \text{ for } k = 1,2,\ldots,K \tag{5}$$

where $g(s)$ defines the piecewise linear function

$$g(s) = \frac{|s+4| - |s-4|}{8} \tag{6}$$

- the learning step, based on the backpropagation of the output error

$$e(n) = \mu(x(n) - \hat{x}(n)) \tag{7}$$

$$w_{km}(n+1) = w_{km}(n) + e(n)q_k(n)g'(s_k(n))y_m(n) \tag{8}$$

$$q_k(n+1) = q_k(n) + e(n)z_k(n), \quad q_k(1) = 0 \tag{9}$$

for $m=1\ldots M$ and $k=1\ldots K$, where μ is the step-size, $x(n)$ is the desired output and $g'(s)$ is the derivative of evaluated s as follows:

$$g'(s) = \begin{cases} 0.25 & \text{if } -4 < s < 4 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The value of M depends on the frequency characteristic of the channel, and the number of hidden neurons K is set to assure the convergence of the equalizer performance.

The result of correction $\hat{x}(n-M+1)$ is obtained by the propagation step defined in Eq. (4) of the data $\tilde{y}(n)$ to $\tilde{y}(n-M+1)$ through the MNN.

3. The Proposed Systolic Architecture

The properties of Eqs. (4)-(9) - recursiveness, regularity, and localized communication - make it possible to use a

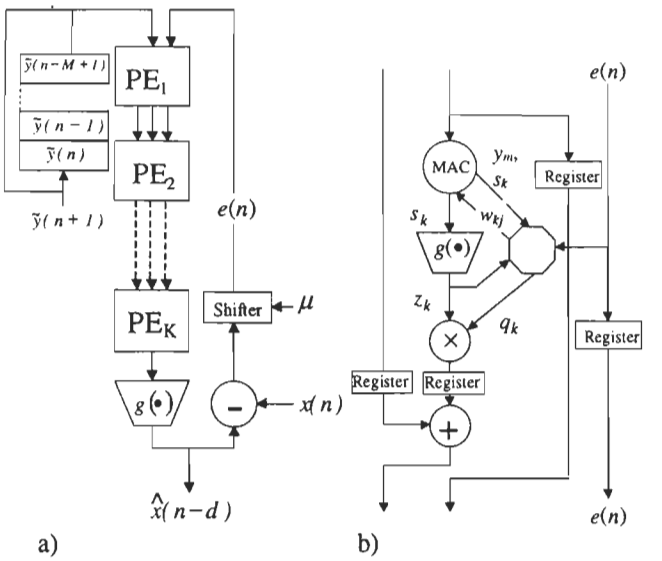


Figure 3: The systolic architecture of PL-MNN algorithm (a) and the processing elements PE_k (b).

systolic approach for the VLSI implementation [5]. These properties justify the inclusion of the on-line learning step in the systolic architecture of the proposed ANN structure to realize an adaptive equalizer. Fig. 3 shows the proposed systolic architecture of PL-MNN where the input data $\tilde{y}(n)$ through in the array of processing elements (PE). All PEs are identical. The number of neurons on the hidden layer establishes the number of PEs. Fig. 3b shows the architecture of one PE and the input-output data of each element cell. The linear part of the hidden neuron to obtain the data s_k is computed in the multiplier-accumulator (MAC) cell; the nonlinear part of the hidden neuron which gives the data z_k is computed in the trapezoid cell; the linear part of the output neuron is computed with the multiplier cell; and, for the learning step, weights w_{kj} and q_k are computed and stocked in the octagonal cell. The bi-directional link between the octagonal cell and the MAC is used to communicate the weights w_{km}, the data $\tilde{y}(n-m+1)$ and s_k. The bi-directional link between the octagonal cell and the multiplier host the data q_k and z_k. With these data, this cell can compute sequentially Eqs (8) and (9) to carry out the training step for an adaptive channel equalization. These sequential operations cannot slow the computation speed because the octagonal cell is waiting for the data $e(n)$ obtained after the propagation step. However, two multipliers and one adder are used in each octagonal cell. The needed adder can be the full adder (FA) of a multiplier.

The MAC constitutes the bottleneck of this architecture. A pipelined version of the MAC is proposed to improve the performance in terms of computation time without adding a

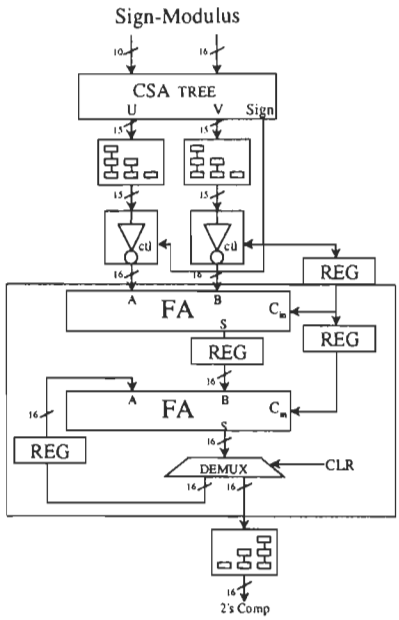


Figure 4: Pipelined version of the MAC

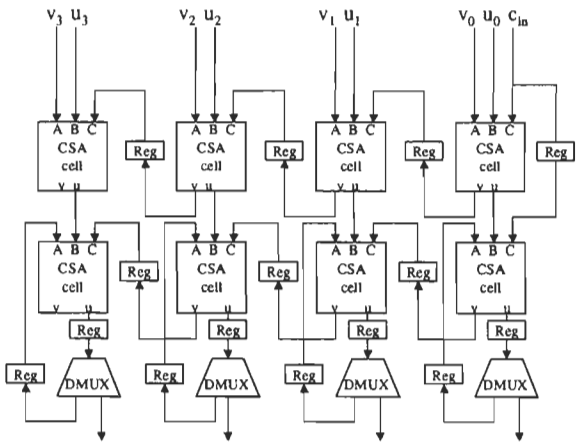


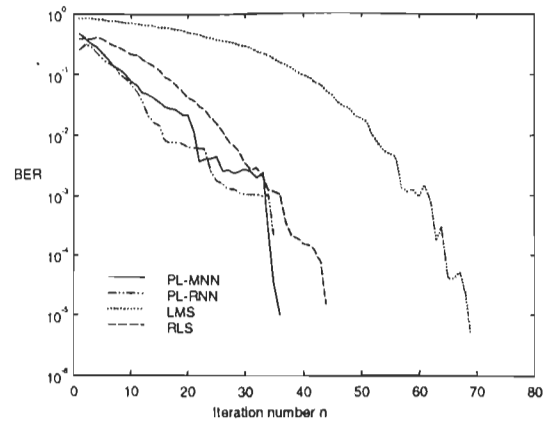
Figure 5: Four bits version of pipelined two stages FAs of the pipelined MAC.

relevant integration area. The problem of pipelining a MAC host is the feedback in the data flow. To reduce this data dependency, the idea is to accumulate the results, bit by bit, instead of accumulating all the binary words at the same time. Fig. 4 shows the pipelined MAC. The MAC is based on a cellular multiplier that uses carry save adder (CSA) cells defined by the following equations [12]:

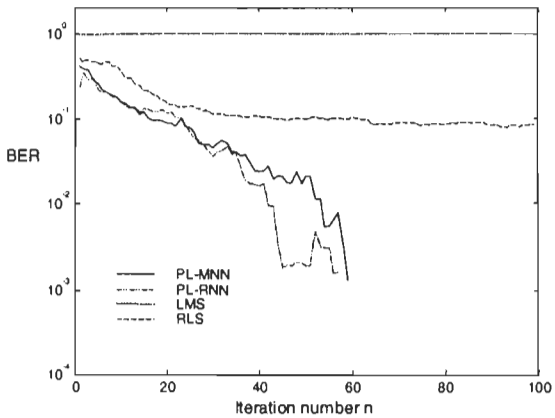
$$u = a \oplus b \oplus c \tag{11}$$

$$v = ab + bc + ac \tag{12}$$

where a, b and c are the added bits. The insertion of a register that makes the pipelining is done every stage. The values that enter in the MAC must be in a sign-modulus representation. After the multiplication, a conversion to



a)

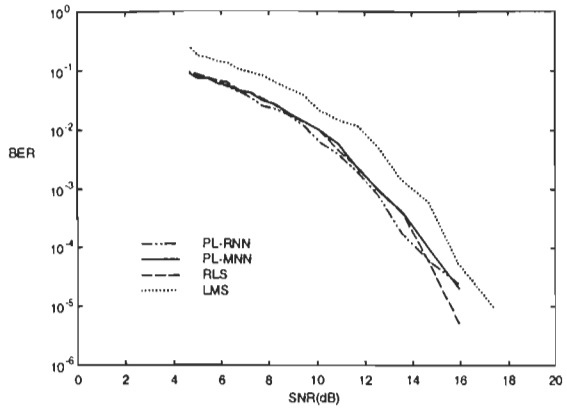


b)

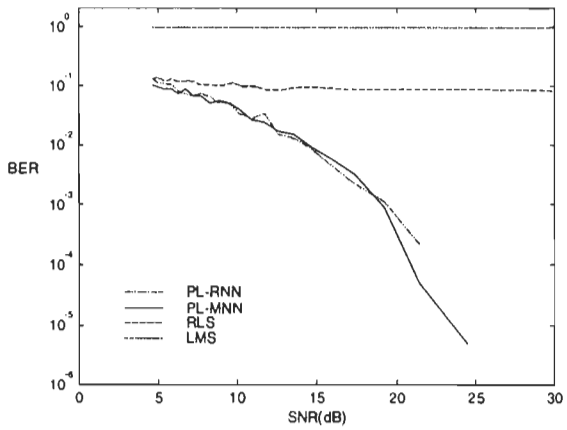
Figure 6: Convergence speed for the linear channel (a) and the nonlinear channel (b) with SNR of 20 dB and

two's complement is done perform, carry out the accumulation. The sign of the multiplication result, control the inversion of all bits of the result. Each value, u-bit and v-bit, has to be added with 1 to complete the two's complement conversion. The sign is linked to the carry-in and to the MSB of each FA to complete the conversion.

Fig. 5 shows an example of the two FA stages with a wordlength of 4-bits. The FAs have a pipelined carry, which means there is a register place between each cell. The cell used here is still the CSA cell where the carry-in is c-bit and the carry-out is a v-bit. To synchronize the most and the least significant bits of the multpliant A and the multiplicator B, a triangle topology of registers is used before the first FA (14 registers for the MSB and no register for the LSB). To synchronize the output, a reverse triangle topology of a register is introduced.



a)



b)

Figure 7: Robustness of noise for the linear channel (a) and the nonlinear channel (b) with an on-line training of 100

4. Numerical Results and Performance Evaluation

In this section, a comparative study with conventional method (LMS and RLS filters) [3], the PL-RNN structure [7] and the proposed PL-MNN structure is presented. For the comparison, an 11 taps LMS and RLS filters are used in all analysis. For the PL-MNN and PL-RNN structures, the parameters are set to M=3, K=3, which correspond to the minimum to assure the algorithm's convergence, and the step size μ is set to 0.5.

The method of correction proposed for the channel equalization has been studied using synthetic data generated according to the following formulas:

$$\tilde{y}(n) = 0.28x(n-2) + x(n-1) + 0.28x(n) + \eta(n) \qquad (13)$$

for the linear channel and

$$\tilde{y}(n) = 0.5v + v^2 + v^3 + \eta(n) \qquad (14)$$

$$v = 0.25x(n-2) + x(n-1) + 0.25x(n) \qquad (15)$$

for the nonlinear channel.

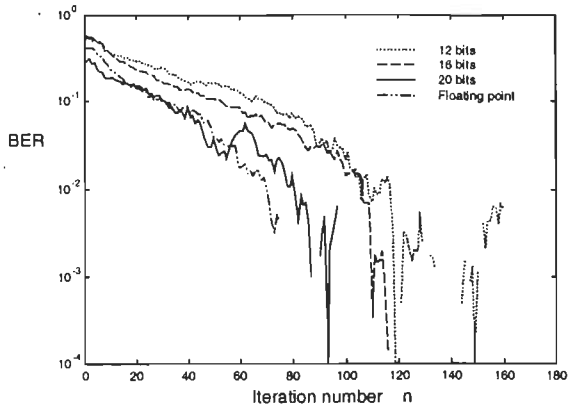


Figure 8: Effect of quantization of PL-MNN algorithm of the on-line learning for a nonlinear channel with SNR of 30 dB.

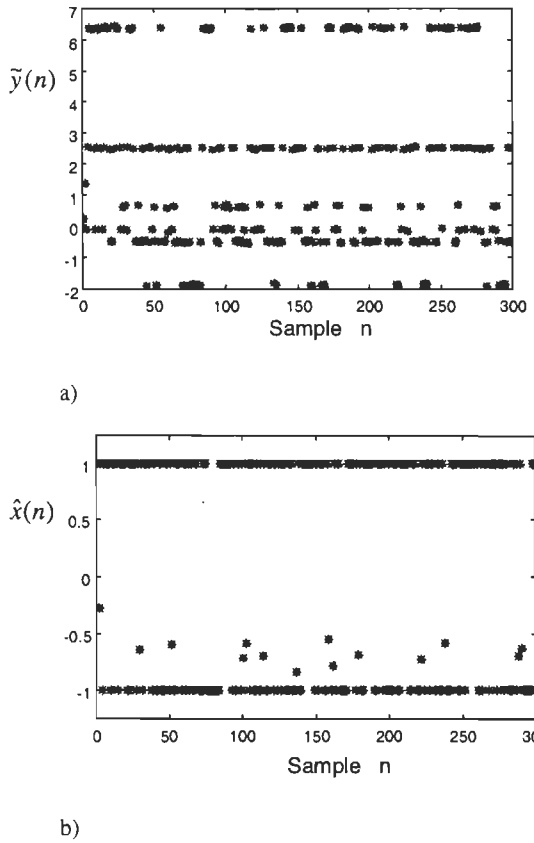


Figure 9: Test of correction for a nonlinear channel with PL-MNN for SNR= 30 dB and μ=0.25: a) the output signal $\tilde{y}(n)$ and b) the result of correction $\hat{x}(n)$.

The input signal $x(n)$ is assumed to be an independent sequence taking values from $\{-1,1\}$ with equal probability and $\eta(n)$ denotes the zero mean white Gaussian noise to obtain a signal noise ratio (SNR)

Table I: Performance evaluation of the PL-MNN algorithm with M=L=3 for 0.5-μm CMOS technology.

Architecture	Learning	Clock frequency [MHz]	Latency [μs]	Throughput [MHz]
Non pipelined version	off-line	100	0.11	33
	on-line		0.11	7
Pipelined version	off-line	500	0.08	167
	on-line		0.08	11.3
DSP56001	off-line	20	1.2	8.3
	on-line		2.2	0.45

Table II: Estimation of algorithms' complexity

Filter	# add./sub.	# div.	# mult.	Total
LMS	2N		2N+1	4N+1
RLS	$2N^2$+2N	1	$3N^2$+4N+1	$5N^2$+6N+2
PL-RNN	M^2+ K^2+2MK +M+3K		$2K^2$+4MK+ M+6K	M^2+ $3K^2$+6MK+2M +9K
PL-MNN	MK+3K-1		4MK+2K +1	5MN+5K

of 5 dB to 40 dB for the study. The quality of correction was assessed using the bit error rate (BER). For all simulations, 10000 data was generated for testing the algorithms in calculating the BER after each iteration on linear and nonlinear channels. Fig. 6 shows the convergence speed in terms of the BER. The data had a SNR of 20 dB for the linear channel and 30 dB for the nonlinear channel. The mean of 20 repetitions is plotted in Fig. 6.

Fig. 7 illustrates robustness of noise. Each algorithm used 100 iterations for the on-line step and 10000 data were used to calculate the BER for an SNR from 5 dB to 40 dB. Fig. 7 illustrates the mean of 20 repetitions.

Also, Table II shows the algorithms' complexity with the on-line learning mode. The algorithm PL-MNN needs the lowest number of computations for the filters' dimensions considered in this paper.

The quantization analysis was performed on the convergence of the PL-MNN algorithm. The effect of quantization is more perceptible an on-line than an off-line learning step, so the convergence is observed to evaluate the wordlength. The nonlinear channel used is defined in (14) and (15) with a SNR of 30 dB. An 8-bits analog to the digital converter was considered to quantify the input signal $\tilde{y}(n)$. Fig. 8 illustrates the effect of the wordlength 12, 16, 20-bits, and in floating point operation used for the data w_{km}, q_k, s_k, z_k and $\hat{x}(n)$ on the convergence in the on-line learning mode of the architecture. With regard to this figure, we can conclude that a 16-bits wordlength is a good tradeoff between the quality of the adaptive equalizer and the integration area of the proposed systolic architecture. Fig. 9 shows an example of reconstruction

using a 16-bits wordlength for a nonlinear channel, which has a SNR of 30 dB.

The performance evaluation is carried out in terms of the latency and throughput for a 16-bits wordlength. The structural model of the proposed architecture was made in VHDL using Mentor-Graphics CAD tools for register transfer level (RTL) modeling and simulation. The design was made by means of standard CAD tools available from the Canadian Microelectronics Corporation (CMC) with the Hewlett-Packard 0.5-µm CMOS technology available from MOSIS through CMC. A low-effort synthesis was made with Synopsys tools. The integration area of one PE is about 4000 gates with a pipelined MAC. The evaluation of the clock frequency, f, of the architecture with a non pipelined MAC is 100 MHz, and this increases to 500 MHz with the pipelined version. In off-line learning (e.g. a stationary channel equalizer), the throughput is evaluated to f/M with and without a pipelined version. The latency is evaluated to (M+L+35)/f and (M+L+5)/f with and without a pipelined MAC respectively. In on-line learning (e.g. a adaptive channel equalizer), the throughput is evaluated to f/(M+L+38) and f/(M+L+8) with and without a pipelined MAC respectively. The latency is the same as in off-line learning.

In comparison with Motorola's DSP56001, its clock frequency is 20 MHz and uses about 50 000 gates. Table I summarizes the performance evaluation of the proposed architecture with and without a pipelined MAC for off-line and on-line learning. A comparison with DSP56001 is shown in terms of clock frequency operation, latency and throughput.

Conclusion

In this paper we have presented a systolic architecture of a piecewise linear multilayer neural network for channel equalization. A comparative study of the proposed PL-MNN algorithm was done with the PL-RNN algorithm and two conventional methods (LMS and RLS) for both linear and nonlinear channels. Our method is based on the reduction of the data dependencies by removing all recurrences to obtain a pipelined version of a systolic architecture. The study has shown that the convergence speed is better than those of the LMS and RLS algorithms and provide a good quality of correction in the case of a nonlinear channel. For a 16-bits version of the pipelined architecture we have obtained throughputs of 8.3 MHz and 166 MHz with and without adaptation of the channel equalizer respectively. The proposed systolic architecture can by applied in wireless communications and measuring systems [1], [14].

References

[1] J. G. Proakis, "Equalization Techniques for High-Density Magnetic Recording", *IEEE Signal Processing Magazine*, Vol. 15, No. 4, July 1998, pp. 73-82.

[2] S. W. McLaughlin, "Schedding Light on the Future of SP for Optical Recording", *IEEE Signal Processing Magazine*, Vol. 15, No. 4, July 1998, pp. 83-94.

[3] S. Haykin, Adaptative Filter Theory, 3^e edition, Prentice Hall, 1996.

[4] D. Godard, "Channel Equalization Using a Kalman Filter for Fast Data Transmission", *IBM Journal Res. Develop.*, vol. 18, 1989, pp. 267-273.

[5] D. Massicotte, "A Systolic VLSI Implementation of Kalman-Filter-Based Algorithms for Signal Reconstruction", *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Seattle, 12-15 May 1998, pp.3029-3032.

[6] G. Kechriotis and E. S. Manolakos, "A VLSI Architecture for the On-Line Training of Recurrent Neural Networks", *Proc. IEEE Asilomar Conference on Signals, Systems and Computers*, November 1991, pp.506-510.

[7] X. Liu, T. Adah and L. Demirekler, "A Piecewise Linear Recurrent Neural Network Structure and its Dynamics", *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP/98)*, Seattle, pp.1221-1224, 1998.

[8] G. Kechtiotis, E. Zervas, E. S. Manolakos, "Using Recurrent Neural Networks for Adaptive Communication Channel Equalization", *IEEE Transaction on Neural Networks*, Vol. 5, No. 2, pp. 267-278, Mars 1994.

[9] T. Nordström, *Highly Parallel Computer for Artificial Neural Networks*, Ph.D. Thesis, Lulea University of Technology, Computer Science and Engineering, Sweden, 1995.

[10]L.-P. Brais and M. Sawan, "Adaptive Filtering Using Field Programmable Devices", *5th Canadian Workshop on FPD (FPD'98)*, Montreal, 1998, pp. 136-140.

[11]Parisi, R. et al., "Fast Adaptive Digital Equalisation by Recurrent Neural Networks", *IEEE Trans. on Signal Processing*, Vol. 45, No. 11, pp 2731-2739, Nov. 1997.

[12]V.K. Madisetti, *VLSI Digital Signal Processors: An introduction to Rapid Prototyping and Design Synthesis*, IEEE Press, 1995.

[13]D. Massicotte and B. Megner, "Neural-Network-Based Method of Correction in a Nonlinear Dynamic Measuring System", *IEEE Instr. and Meas. Technology Conf.*, Venise, 24-26 May 1999.

[14]S.K. Nair and J. Moon, "Data Storage Channel Equalization Using Neural Networks", *IEEE Trans. Neural Networks*, Vol. 8, No. 5, Sept. 1997, pp. 1037-1048.

[15] H. Amin, K.M. Curtis, and B.R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of neural network", *IEE Proc.-Circuits Devices Syst.*, Vol. 144, No. 6, Dec. 1997, pp. 313-317.

QPSK Equalizer Based on Piecewise Neural Network

Martin VIDAL and Daniel MASSICOTTE

Department of Electrical Engineering
Université du Québec à Trois-Rivières
C.P. 500, Trois-Rivières, Québec, Canada, G9A 5H7
Phone: 1-(819)-376-5071, FAX : 1-(819)-376-5219
E-mail: {Martin_Vidal, Daniel_Massicotte}@uqtr.uquebec.ca

Abstract

An equalizer based on a multilayer neural network (MNN) is proposed to solve the problem of the nonlinear channel QPSK communication. An algorithm using a complex piecewise linear multilayer neural network (CPL-MNN) algorithm is applied. The sigmoid function is replaced by a canonical piecewise linear function which makes the MNN more suitable for a digital VLSI implementation. A performance study on both linear and nonlinear channels is presented. A comparison of results obtained with two conventional methods (LMS and RLS) and two MNN methods (using both conventional sigmoid activation function and piecewise linear function) is also presented. The properties of our algorithm - recursiveness, regularity, and localized communication - make it possible to use a systolic approach to implement the equalizer in digital VLSI technology.

1. Introduction

The adaptive channel equalization shown in Fig. 1 is a very common inverse problem in various fields of application, such as telecommunications, wireless communications, modems and measurement systems (e.g. [1-3], [9]). To solve the problem of channel equalization, it is necessary to estimate the signal applied to the channel, $s(n)$, defined as the transmitted signal, taking into account that it is a numerically ill-conditioned rule. The data $s(n)$ is transmitted crossing a channel to produce the output $\tilde{y}(n)$ corrupted with additive noise $\eta(n)$. The adaptive channel equalization problem is rather popular [3], [10] but the VLSI implementation of the algorithms is rarely considered [6], [12]. Numerical methods are suitable for solving the problem under discussion; they are based on an adaptive step of the channel by the knowledge of the transmit signal $s(n)$ and of the measured samples of the output, $\tilde{y}(n)$. Solutions given by the LMS and RLS algorithms were proposed in [3] and by the Kalman filter in [4]. These algorithms, however, cannot be applied on a nonlinear channel, which is well treated with a neural-network-based method of correction [1], [2], [7], [10].

In a previous work [12], we have presented a piecewise linear multilayer neural network (PL-MNN) algorithm for PAM modulation type and a VLSI systolic digital implementation. In this paper, we present the application of the PL-MNN, defined in Fig. 2, to the QPSK communication using complex parameters (the weights w_{km} and q_k) to obtain a complex piecewise linear multilayer neural network (CPL-MNN). The QPSK modulation type uses the phase of the signal to form a

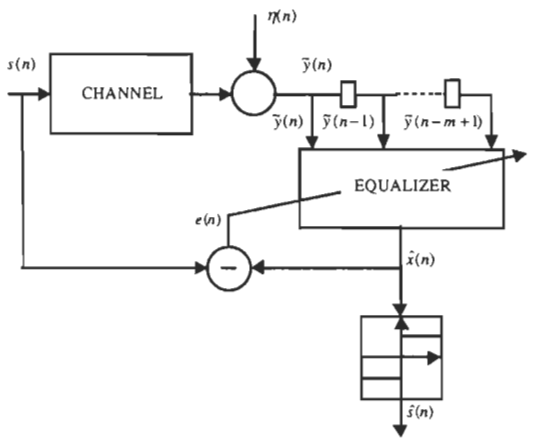


Figure 1: Adaptive channel equalization.

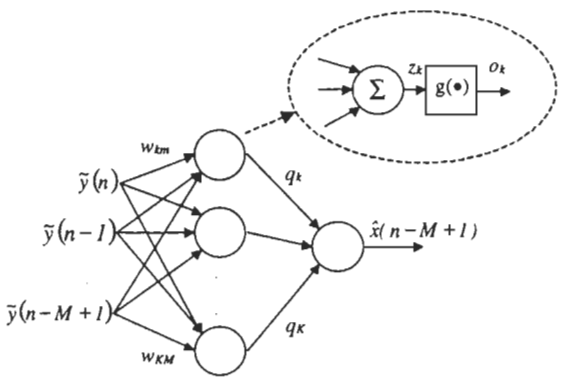


Figure 2 : The structure of the neural network (CPL-MNN).

symbol which includes two bits [9]. Both linear and nonlinear channels are considered for QPSK communication.

In Section 2, we present the CPL-MNN algorithm. A description of the systolic architecture is presented in Section 3. A comparison study of the LMS, RLS, and a conventional MNN methods is performed in Section 4. Finally, Section 5, gives the conclusion.

2. The proposed CPL-MNN Algorithm

The use of QPSK modulation implies a complex algorithm, called CPL-CNN, to make a phase correction. A learning algorithm for feedforward multilayer neural networks was proposed in [13] and was adapted to PL-MNN algorithm [12] as follows:

- the propagation step, defined by

$$\hat{x}(n-M+1) = g\left(\sum_{k=1}^{K} q_k g(z_k)\right) \tag{1}$$

$$z_k = \sum_{m=1}^{M} w_{km}\tilde{y}(n-m+1) \text{ for } k = 1,2,\ldots,K \tag{2}$$

where $g(z)$ is the complex piecewise linear function

$$g(z) = \left.\frac{|z+4|-|z-4|}{8}\right|_{z_{Re}} + j\left.\frac{|z+4|-|z-4|}{8}\right|_{z_{Im}} \tag{3}$$

- the learning step, based on the backpropagation of the output error

$$e(n) = s(n) - \hat{x}(n) \tag{4}$$

$$e(n) = \begin{cases} 0 & \text{if } |e(n)| < 0.3 \\ e(n) & \text{otherwise} \end{cases} \tag{5}$$

$$\delta_k = g'(z(n)_{Re})\left.[e(n)q_k^*]\right|_{Re} + j\, g'(z(n)_{Im})\left.[e(n)q_k^*]\right|_{Im} \tag{6}$$

$$w_{km}(n+1) = w_{km}(n) + \mu\delta_k y_m^*(n) \tag{7}$$

$$q_k(n+1) = q_k(n) + \mu e(n) o_k^*(n) \tag{8}$$

for $m=1,2,\ldots M$ and $k=1,2,\ldots K$, where the asterisk (*) denotes the complex conjugation, μ is the step-size, $x(n)$ is the desired output and $g'(z)$ is the derivative of evaluated z as follows:

$$g'(z) = \begin{cases} 0.25 & \text{if } -4 < z < 4 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The Eq. (5) is defined as the error confinement

[14] and is used to increase the stability and the speed of convergence. The value of M depends on the frequency characteristic of the channel, and the number of hidden neurons K is set to ensure the convergence of the equalizer performance.

The result of correction $\hat{x}(n-d)$ is obtained by the propagation step, defined in Eq. (1), of the data $\tilde{y}(n)$, $\tilde{y}(n-1),\ldots,\tilde{y}(n-d)$ through the MNN. Where d is the delay due the channel and the algorithm (usually $d=M-1$). The estimation of the transmitted signal $s(n)$ is computed as follows:

$$\hat{s}(n) = sign(\hat{x}(n)_{Re}) + j\, sign(\hat{x}(n)_{\mathrm{Im}}) \tag{10}$$

where *sign* is the Heaviside function.

3. VLSI Implementation of the Equalizer

The properties of Eqs. (1)-(10) - recursiveness, regularity, and localized communication - make it possible to use a systolic approach for the VLSI implementation [5]. These properties justify the inclusion of the on-line learning step in the systolic architecture of the proposed ANN structure to realize an adaptive equalizer. Fig. 3 shows the systolic architecture proposed in [12] for the PL-MNN where the input data $\tilde{y}(n)$, stocked in a circular stack, through the array of processing elements (PE). We can applied the same architecture using complex number arithmetic units. All PEs are identical. The number of neurons on the hidden layer establishes the number of PEs. The stack and the array of PE give a modularity to the architecture to allow any number of neurons on the hidden layer and any number of inputs. Fig. 4 shows the architecture of one PE and the input-output data of each element cell. The linear part of the hidden neuron to obtain the data z_k is computed in the multiplier-accumulator (MAC) complex number cell (e.g. [15]); the nonlinear part of the hidden neuron which gives the data o_k is computed in the

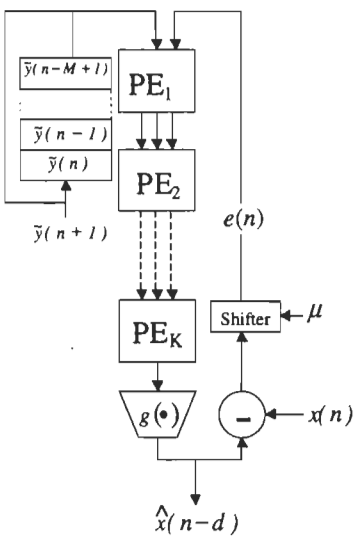


Figure 3 : The systolic architecture of PL-MNN algorithm.

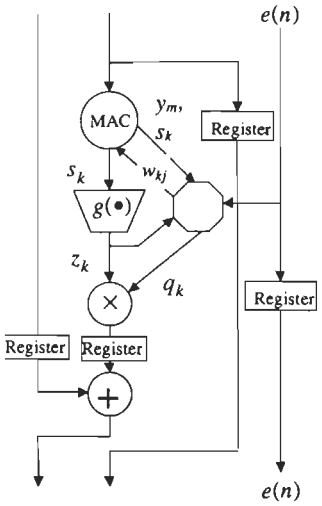


Figure 4: The processing element PE_k.

trapezoid cell; the linear part of the output neuron is computed with the complex number multiplier cell. The elements needed by the learning algorithm come out of the propagation step sequentially, that allows the centralization of all the learning steps in only one low complexity cell and the adaptation is doing during the propagation step. The weights w_{kj} and q_k in complex numbers are computed and stocked in the octagonal cell. The bi-directional link between the octagonal cell and the MAC is used to communicate the weights w_{km}, the data $\tilde{y}(n-m+1)$ and z_k. The bi-directional link between the octagonal cell and the multiplier hosts the data q_k and o_k. With these data, this cell can compute sequentially Eqs (4) to (8) to carry out the training step for adaptive channel equalization. These sequential operations cannot slow the computation speed because the octagonal cell is waiting for the data $e(n)$ obtained after the propagation step.

4. Numerical Results

The method of correction proposed for the channel equalization has been studied using synthetic data generated according to the following formulas:

- for the linear channel:

$$\begin{aligned}\tilde{y}(n) &= (-0.005-0.004\mathrm{j})s(n)+(0.009+0.030\mathrm{j})s(n-1)\\ &+(-0.024-0.104\mathrm{j})s(n-2)+(0.854+0.520\mathrm{j})s(n-3)\\ &+(-0.218+0.273\mathrm{j})s(n-4)+(0.049-0.074\mathrm{j})s(n-5)\\ &+(-0.016+0.020\mathrm{j})s(n-6)+\eta(n)\end{aligned} \quad (11)$$

- for the nonlinear channel:

$$\begin{aligned}v(n) &= (-0.005-0.004\mathrm{j})s(n)+(0.009+0.030\mathrm{j})s(n-1)\\ &+(-0.024-0.104\mathrm{j})s(n-2)+(0.854+0.520\mathrm{j})s(n-3)\\ &+(-0.218+0.273\mathrm{j})s(n-4)+(0.049-0.074\mathrm{j})s(n-5)\\ &+(-0.016+0.020\mathrm{j})s(n-6)\end{aligned} \quad (12)$$

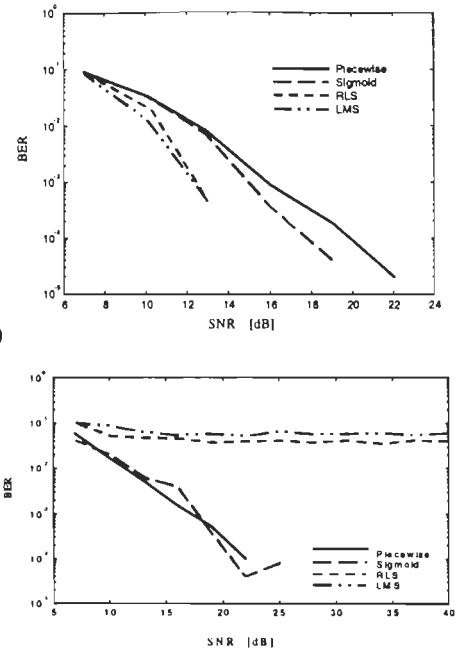


Figure 5 : Robustness of noise for the linear channel (a) and the nonlinear channel (b) with an on-line training of 5 000 iterations.

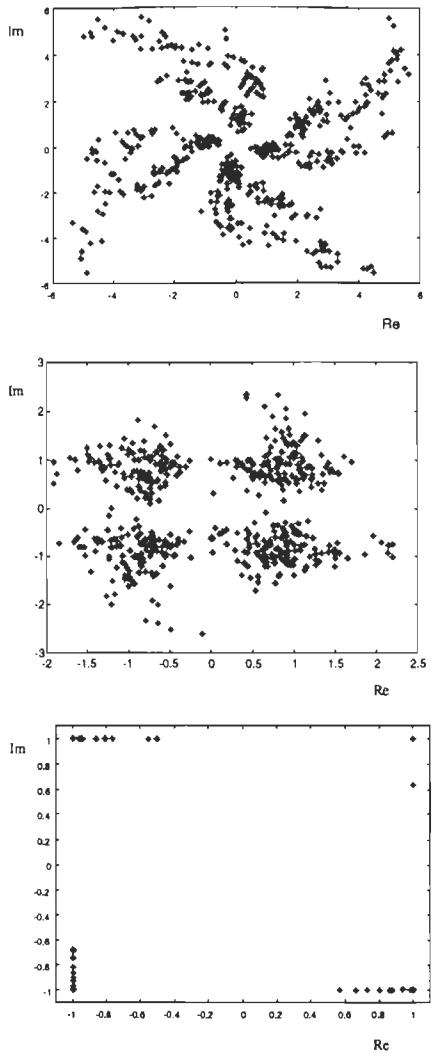
$$|\tilde{y}(n)| = 0.5|v(n)| + 0.25|v(n)|^2 + 0.5|v(n)|^3 + \eta_1(n) \quad (13)$$

$$\arg(\tilde{y}(n)) = \arg(v(n)) + |\tilde{y}(n)| \quad (14)$$

A comparative study with two conventional methods (LMS and RLS filters) [3], the conventional MNN using a sigmoid function for the neurons in the hidden layer and the proposed CPL-MNN structure has been realized for both linear and nonlinear channels. For the comparison, 11 taps LMS and RLS filters are used in all analysis. For both MNN structures, the parameters are set to M=4, K=5, which correspond to the minimum to ensure the algorithm convergence, and the step size μ is set to 0.031.

The input signal $s(n)$ is assumed to be an independent sequence taking values from {-1,1} for each complex part with equal probability and $\eta(n)$ denotes the zero mean white Gaussian noise to obtain a signal noise ratio (SNR) of 5 dB to 40 dB. The quality of correction is assessed using the bit error rate (BER). For all simulations, 10 000 data are generated to test the algorithms, the BER was calculated after each learning on linear and nonlinear channels. Fig. 5 illustrates robustness against noise. Each algorithm used 5 000 data for the adaptation of its parameters for a SNR from 5 dB to 40 dB. Fig. 5 illustrates the mean of 10 repetitions.

Fig. 6 shows an example of equalization on a nonlinear channel. It appears that the CPL-MNN algorithm succeeds in equalizing the nonlinear channel without error, contrary to the RLS algorithm.



a)

b)

c)

Figure 6 : An example of correction results $\{\hat{x}(n)\}$ for a nonlinear channel output $\{\tilde{y}(n)\}$ (a) with the RLS algorithm (b) and the proposed CPL-MNN algorithm (c).

5. Conclusion

In this paper, we have presented an equalizer based on a piecewise linear multilayer neural network (CPL-MNN). A comparative study of the proposed CPL-MNN algorithm has been done with the conventional multilayer neural network algorithm and two conventional methods (LMS and RLS filters) for both linear and nonlinear channels. The study has shown that the CPL-MNN algorithm provides a good quality of correction in both case of channels. Also, the proposed algorithm has a low complexity and is more stable in fixed point arithmetic then both RLS and LMS algorithms. Based on our previous work [12], we have proposed a VLSI systolic architecture to implement the equalizer using a complex number arithmetic units. The systolic architecture makes it possible a modular structure to be applied for any size of the hidden layer. With minimal adding, a blind equalizer can be built with a directed decision adaptation [9]. The proposed equalizer can be used in wireless communications and measurement systems.

Acknowledgement

This project was made possible by the support of the Fonds FCAR Québec and by the Natural Sciences and Engineering Research Council of Canada.

References

[1] P.-R. Chang, B.-C. Wang, "Adaptive Decision feedback equalization for digital satellite channel using multilayer neural networks", *IEEE Journal on Selected Areas in Communications*, February 1995, pp. 316-324.

[2] P.-R. Chang et al., "Adaptive Packet Equalization for Indoor Radio Channel Using Multilayer Neural Network", *IEEE Transactions on Vehicular Technology*, 1994, pp. 773-780.

[3] S. Haykin, *Adaptative Filter Theory*, Prentice Hall, 1996.

[4] D. Godard, "Channel Equalization Using a Kalman Filter for Fast Data Transmission", *IBM Journal Res. Develop.*, vol. 18, 1989, pp. 267-273.

[5] D. Massicotte, "A Systolic VLSI Implementation of Kalman-Filter-Based Algorithms for Signal Reconstruction", *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Seattle, 12-15 May 1998, pp. 3029-3032.

[6] G. Kechriotis and E. S. Manolakos, "A VLSI Architecture for the On-Line Training of Recurrent Neural Networks", *Proc. IEEE Asilomar Conference on Signals, Systems and Computers*, November 1991, pp. 506-510.

[7] X. Liu, T. Adah and L. Demirekler, "A Piecewise Linear Recurrent Neural Network Structure and its Dynamics", *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP/98)*, Seattle, pp.1221-1224, 1998.

[8] G. Kechtiotis, E. Zervas, E. S. Manolakos, "Using Recurrent Neural Networks for Adaptive Communication Channel Equalization", *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 267-278, March 1994.

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

[10]Parisi, R. et al., "Fast Adaptive Digital Equalisation by Recurrent Neural Networks", *IEEE Trans. on Signal Processing*, Vol. 45, No. 11, pp 2731-2739, Nov. 1997.

[11]H. Amin, K.M. Curtis, and B.R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of neural network", *IEE Proc.-Circuits Devices Syst.*, Vol. 144, No. 6, Dec. 1997, pp. 313-317.

[12]M. Vidal and D. Massicotte, "A Parallel Architecture of a Piecewise Linear Neural Network for Channel Equalization", *IEEE Instrumentation and Measurement Technology Conference (IMTC'99)*, Venice, May 1999.

[13]You, C., Hong, D., "Nonlinear Blind Equalization Schemes Using Complex-Valued Multilayer Feedforward Neural Networks", *IEEE Transactions on Neural Networks*, November 1998, pp.1442-1455.

[14]Nair, S.K, Moon, J., "Data Storage Channel Equalization Using Neural Networks", *IEEE Transactions on Neural Networks*, September 1997, pp.1037-1048.

[15]Shing, K-W., Song, B-S., "A 200 MHz Complex Multiplier Using Redundant Binary Arithmetic", IEEE Journal of Solid-State Circuits, vol.33, no.6,June 1998, pp.904-908.

Annexe B : Programme VHDL pour le multiplieur-accumulateur pipeliné

```
--VHDL structurel parametrise d'un multiplieur accumulateur
--pipeline pour les nombres signes. Les entrees et la sortie sont en signe-module.
--La longueur des mots binaires doit etre paire (signe inclus)

--Martin Vidal 1998
--Universite du Quebec a Trois-Rivieres


LIBRARY ieee;
USE ieee.std_logic_1164.all;


--=================================================
--PAQUETAGES DU PROGRAMME
--=================================================

--*********************************
--PAQUETAGE DES TYPES ET CONSTANTES
--*********************************

PACKAGE piece IS

--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
 CONSTANT word:INTEGER :=8;   --DETERMINE LA LONGUEUR DES MOTS BINAIRES
--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
 CONSTANT conversion:INTEGER:=8;              --DETERMINE LA LONGUEUR DES
                                              --MOTS BINAIRES du convertisseur
--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

--@@@@@@@@@@@@@@@@@@@@@
 CONSTANT S:INTEGER:=3;        --Nombre du neurone sur la couche cachee
--@@@@@@@@@@@@@@@@@@@@@

--@@@@@@@@@@@@@@@@@@@@@
 CONSTANT M:INTEGER:=2 ;        --Nombre de retards sur l'entree
--@@@@@@@@@@@@@@@@@@@@@


 SUBTYPE data IS std_logic_vector(word-1 downto 1);

 TYPE sig_vect IS ARRAY(INTEGER RANGE <>) of std_logic_vector(word-1 downto 1);


END piece;

PACKAGE BODY piece IS

END piece;

--************************
--PAQUETAGE DES COMPOSANTS
--************************
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.piece.all;


PACKAGE composants IS

--******************
--COMPOSANTS DE BASE
--******************

COMPONENT cell                                                  --Cellule CSA
PORT
     (a,b,c  :  in  std_logic;
      u,v    :  out std_logic);
END COMPONENT;
-------------------------------------------------------
COMPONENT ou_ex                 --XOR
 PORT(a,b:in std_logic;
    c:out std_logic);
end COMPONENT;
-------------------------------------------------------
COMPONENT et                    --AND
PORT(a,b:in std_logic;c:out std_logic);
END COMPONENT;
-------------------------------------------------------
COMPONENT registre
 PORT(
    a,clk,rst : in  std_logic;
    b         : out std_logic);
END COMPONENT;
-------------------------------------------------------
component buf                   --BUFFER
    port( op : buffer std_logic;  ip : in std_logic);
end component;
-------------------------------------------------------
COMPONENT out_cell              --Etage de sortie du MAC pipeline ( sortie ou accumulation )
PORT(ip,end_acc,rst,clk:IN std_logic;
   op_ret,op_fwd,ctl_fwd:OUT std_logic);
END COMPONENT;

COMPONENT inv_cell
  PORT(a,in_ctl,clk,rst :IN  std_logic;
     b,out_ctl:OUT std_logic);
END COMPONENT;

--*****************************
--ETAGES DE COMPOSANTS DE BASES
--*****************************

COMPONENT etage_mult            --Assemblage de cellule CSA
 GENERIC(nb_cell:INTEGER);
```

```
 PORT(
   a,b,c:in std_logic_vector(nb_cell downto 1);
   u,v:out std_logic_vector(nb_cell downto 1));
END COMPONENT;
-------------------------------------------------------
COMPONENT etage_et            --Assemblage de AND
 GENERIC(nb_et:INTEGER);

 PORT(a  :IN  std_logic_vector(nb_et DOWNTO 1);
    b  :IN  std_logic;
    c  :OUT std_logic_vector(nb_et DOWNTO 1));
END COMPONENT;
-------------------------------------------------------
COMPONENT etage                                    --Assemblage de cellules CSA et de registres
 GENERIC(nb_cell:INTEGER);

 PORT(
          a,b,c:IN std_logic_vector(nb_cell downto 1);
   u,v  :OUT std_logic_vector(nb_cell downto 1);
   clk,rst:in std_logic);
END COMPONENT;
-------------------------------------------------------
COMPONENT colonne                          --Registre en serie
 GENERIC(nb_reg:INTEGER);

 PORT(
   a      :in std_logic;
   b      :out std_logic;
   clk,rst  :in std_logic);

END COMPONENT;
-------------------------------------------------------
COMPONENT etage_reg                --Serie de registre en parallele

GENERIC(nb_reg:INTEGER);
PORT(
   a      :in std_logic_vector(nb_reg downto 1);
   b      :out std_logic_vector(nb_reg downto 1);
   clk,rst  :in std_logic);
END COMPONENT;
-------------------------------------------------------
COMPONENT pyramide
 generic(nb_reg:INTEGER);
 PORT( a:in std_logic_vector(nb_reg downto 1);
     b:out std_logic_vector(nb_reg downto 1);
     clk,rst:in std_logic);
END COMPONENT;
-------------------------------------------------------
COMPONENT pyramide2
 generic(nb_reg:INTEGER);
 PORT( a:in std_logic_vector(nb_reg downto 1);
     b:out std_logic_vector(nb_reg downto 1);
     clk,rst:in std_logic);
```

```
END COMPONENT;
-------------------------------------------------
COMPONENT etage_sortie
GENERIC(nb_cell:INTEGER);
PORT(ip:IN std_logic_vector(nb_cell downto 1);
   op_ret,op_fwd:OUT std_logic_vector(nb_cell downto 1);
   end_acc,clk,rst:IN std_logic);
END COMPONENT;
-------------------------------------------------
COMPONENT matrice
 GENERIC(nb_reg,nb_col:INTEGER);
 PORT( a:in std_logic_vector(nb_col downto 1);
     b:out std_logic_vector(nb_col downto 1);
     clk,rst:in std_logic);
END COMPONENT;

--********************
--UNITES ARITHMETIQUES
--********************

COMPONENT multiplieur
GENERIC(nb_bits1,nb_bits2:INTEGER);
PORT(
   a:IN std_logic_vector(nb_bits1 DOWNTO 1);
   b:IN std_logic_vector(nb_bits2 DOWNTO 1);
   u,v:OUT std_logic_vector(nb_bits2-1 DOWNTO 1);
   clk,rst,in_acc:in std_logic;
   sign_mult,out_acc:out std_logic);
END COMPONENT;
-------------------------------------------------
COMPONENT full_adder
GENERIC(nb_cell:INTEGER);
PORT(
   a,b:in std_logic_vector(nb_cell downto 1);
   u:out std_logic_vector(nb_cell downto 1);
   Cin, clk,rst:IN std_logic);
END COMPONENT;
-------------------------------------------------
COMPONENT comp_2
PORT(a:IN data;
   ctl,clk,rst:IN std_logic;
   b:OUT std_logic_vector(word downto 1));
END COMPONENT;
-------------------------------------------------
COMPONENT mac
PORT(
   a,b:in std_logic_vector(word downto 1);
   end_acc,clk,rst:in std_logic;
   c:out std_logic_vector(word downto 1));
END COMPONENT;

END composants;

--================================================
```

```
--Cellule de base pour le pipeline d'un MAC
--=================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY cell IS

 PORT
     (a,b,c  :  in  std_logic;
      u,v    :  out std_logic);
END cell;

ARCHITECTURE behav OF cell IS
BEGIN

calcul:PROCESS(a,b,c)
BEGIN
 u<=a XOR b XOR c;
 v<=(a AND b)OR(a AND c)OR(b AND c);
END PROCESS;

END behav;

--=================================================
--Registre avec reset
--=================================================
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY registre IS
PORT(
    a,clk,rst : in  std_logic;
    b         : out std_logic);
END registre;

ARCHITECTURE behav OF registre IS
BEGIN

PROCESS(clk,a,rst)
BEGIN

if rst='1' THEN
  IF clk'EVENT AND clk ='1'THEN
    b<=a;
  END IF;
ELSE
  b<='0';
END IF;

END PROCESS;

END behav;
```

```
--=====================================================
--Porte ET (2 entrees)
--=====================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY et IS

PORT(a,b:in std_logic;c:out std_logic);

END et;

ARCHITECTURE behav OF et IS
BEGIN
c<=a and b;
END behav;

--=====================================================
--Porte XOR (2 entrees)
--=====================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ou_ex IS

PORT(a,b:in std_logic;c:out std_logic);

END ou_ex;

ARCHITECTURE behav OF ou_ex IS
BEGIN
c<=a xor b;
END behav;
--=====================================================
--VHDL pour l'assemblage d'une serie de cellules de base
--=====================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;


ENTITY etage_mult IS

 GENERIC(nb_cell:INTEGER);

 PORT(
   a,b,c:in std_logic_vector(nb_cell downto 1);
   u,v:out std_logic_vector(nb_cell downto 1));

END etage_mult;
```

```
ARCHITECTURE struct OF etage_mult IS

SIGNAL bidon:std_logic;

begin

G1:FOR i IN 1 to nb_cell GENERATE
 G2:IF i=1 GENERATE
  premier_cellule:cell
    port map(a(i),b(i),c(i),bidon,v(i));
 END GENERATE;

  G3:IF i>1 and i<=nb_cell GENERATE
  cellule:cell
    port map(a(i),b(i),c(i),u(i-1),v(i));
 END GENERATE;
END GENERATE;

u(nb_cell)<='0';

END struct;

--=======================================================
--VHDL pour l'assemblage d'une serie de registres
--=======================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;

ENTITY etage_reg IS

GENERIC(nb_reg:INTEGER);

PORT(
    a       :in std_logic_vector(nb_reg downto 1);
    b       :out std_logic_vector(nb_reg downto 1);
    clk,rst  :in std_logic);


END etage_reg;

ARCHITECTURE struct OF etage_reg IS

begin

G1:FOR i IN 1 to nb_reg GENERATE
  cellule:registre
    port map(a(i),clk,rst,b(i));
END GENERATE;

END struct;
```

```
--=========================================================
--ASSEMBLAGE D'UNE SERIE DE CELLULE ET REGISTRE
--=========================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
--**
ENTITY etage IS
GENERIC(nb_cell:INTEGER);

PORT(
        a,b,c:IN std_logic_vector(nb_cell downto 1);
   u,v  :OUT std_logic_vector(nb_cell downto 1);
   clk,rst:in std_logic);
END etage;
--**
ARCHITECTURE struct OF etage IS

SIGNAL ui,vi:std_logic_vector(nb_cell downto 1);

BEGIN

cellule:etage_mult
   generic map(nb_cell)
   port map(a,b,c,ui,vi);

regist1:etage_reg
  generic map(nb_cell)
  port map(ui,u,clk,rst);

regist2:etage_reg
  generic map(nb_cell)
  port map(vi,v,clk,rst);

END struct;

--=============================================--
--VHDL pour l'assemblage de ET logique
--=============================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY etage_et IS
GENERIC(nb_et:INTEGER);
PORT(a  :IN  std_logic_vector(nb_et DOWNTO 1);
   c  :OUT std_logic_vector(nb_et DOWNTO 1);
   b  :IN std_logic);
```

```
END etage_et;

ARCHITECTURE struct OF etage_et IS

BEGIN

G1:FOR i in 1 to nb_et GENERATE
et_logic:et
PORT MAP(a(i),b,c(i));

END GENERATE;

END struct;

--=============================================================
--MATRICE DE REGISTRES
--=============================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY matrice IS
 GENERIC(nb_reg,nb_col:INTEGER);
 PORT( a:in std_logic_vector(nb_col downto 1);
      b:out std_logic_vector(nb_col downto 1);
      clk,rst:in std_logic);
END matrice;

ARCHITECTURE struct OF matrice IS

BEGIN
G1:IF nb_reg=0 GENERATE
  b<=a;
END GENERATE;

G2:IF nb_reg>0 GENERATE
 G3:FOR i in 1 to nb_col GENERATE

  col_reg:colonne
   GENERIC MAP (nb_reg)
   PORT MAP(a(i),b(i),clk,rst);

 END GENERATE;
END GENERATE;

END struct;
--=============================================================
--MULTIPLIEUR PARAMETRISE
--=============================================================

LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY multiplieur IS
GENERIC (nb_bits1,nb_bits2:INTEGER);
PORT(
   a:IN std_logic_vector(nb_bits1 DOWNTO 1);
   b:IN std_logic_vector(nb_bits2 DOWNTO 1);
   u,v:OUT std_logic_vector(nb_bits2-1 DOWNTO 1);
   clk,rst,in_acc:in std_logic;
   sign_mult,out_acc:out std_logic);
END multiplieur;

ARCHITECTURE struct OF multiplieur IS

--************************************
--DECLARATION DES SIGNAUX DE CONNECTION
--************************************

SIGNAL ai,bi,ci,temp:sig_vect(nb_bits1 DOWNTO 1);
SIGNAL zero,clk_inv:std_logic;
SIGNAL signe,acc:std_logic_vector(nb_bits1 downto 1);
SIGNAL zeros:data;

--************************
--DEBUT DE L'ARCHITECTURE
--************************

BEGIN
clk_inv<=not clk;
zero<='0';
zeros<=(others=>'0');
acc(1)<=in_acc;
G1: FOR i IN 1 TO (nb_bits1-2) GENERATE

 G2:IF i=1 GENERATE
 calcul_entree1:etage_et
   GENERIC MAP(nb_bits2-1)
   PORT MAP(a=>b(nb_bits2-1 downto 1),b=>a(i),c=>ai(i));
 calcul_entree2:etage_et
   GENERIC MAP(nb_bits2-1)
   PORT MAP(a=>b(nb_bits2-1 downto 1),b=>a(i+1),c=>bi(i));
 premier_etage:etage
   GENERIC MAP(nb_bits2-1)
   PORT MAP(ai(i),bi(i),zeros,ai(i+1),bi(i+1),clk,rst);
 signe_detector:ou_ex
   port map(a(nb_bits1),b(nb_bits2),signe(i));
 reg:registre
   PORT MAP(signe(i),clk,rst,signe(i+1));
 reg2:registre
   PORT MAP(acc(i),clk_inv,rst,acc(i+1));
 END GENERATE;
```

```
 G3:IF i > 1 GENERATE
 calcul_entree3:etage_et
   GENERIC MAP(nb_bits2-1)
   PORT MAP(b(nb_bits2-1 downto 1),a(i+1),temp(i));
 tampon:matrice
   GENERIC MAP(i-1,nb_bits2-1)
   PORT MAP(a=>temp(i),b=>ci(i),clk=>clk,rst=>rst);
 etage_paire:etage
   GENERIC MAP(nb_bits2-1)
   PORT MAP(ai(i),bi(i),ci(i),ai(i+1),bi(i+1),clk,rst);
 reg:registre
   PORT MAP(signe(i),clk,rst,signe(i+1));
 reg2:registre
   PORT MAP(acc(i),clk_inv,rst,acc(i+1));
 END GENERATE;

END GENERATE;

u<=ai(nb_bits1-1);
v<=bi(nb_bits1-1);
sign_mult<=signe(nb_bits1-1);
out_acc<=acc(nb_bits1-1);

END struct;

--=============================================================
--Serie de registre relies en colonne
--=============================================================


LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY colonne IS

 GENERIC(nb_reg:INTEGER);

 PORT(
   a:in std_logic;
   b:out std_logic;
   clk,rst:IN std_logic);


END colonne;

--************
--ARCHITECTURE
--************

ARCHITECTURE struct OF colonne IS
```

```
--**********************
--SIGNAUX DE CONNECTION
--**********************

SIGNAL inter:std_logic_vector(nb_reg downto 1);

BEGIN

G7:IF nb_reg=0 GENERATE
  b<=a;
END GENERATE;

G1:IF nb_reg=1 GENERATE
 reg_1:registre
  PORT MAP(a,clk,rst,b);
END GENERATE;

G2:IF nb_reg>1 GENERATE

 G3:FOR i in 1 to nb_reg GENERATE

  G4:IF i=1 GENERATE
   premier_reg:registre
    PORT MAP(a,clk,rst,inter(i));
  END GENERATE;

  G5:IF i=nb_reg GENERATE
   dernier_reg:registre
    PORT MAP(inter(i-1),clk,rst,b);
  END GENERATE;

  G6:IF i>1 AND i<nb_reg GENERATE
   reg_2:registre
    PORT MAP(inter(i-1),clk,rst,inter(i));
  END GENERATE;

 END GENERATE;

END GENERATE;

END struct;

--=============================================================
--Cellule d'inversion
--=============================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.piece.all;
USE piecewise.composants.all;

ENTITY inv_cell IS
```

```
 PORT(a,in_ctl,clk,rst :IN  std_logic;
    b,out_ctl:OUT std_logic);
 END;

ARCHITECTURE struct OF inv_cell IS
BEGIN

 reg:registre
  PORT MAP(a=>in_ctl,b=>out_ctl,clk=>clk,rst=>rst);

 ou_exclusif:ou_ex
  PORT MAP(a=>a,b=>in_ctl,c=>b);

END struct;

--=====================================================================
--VHDL pour effectuer le complement 2 si necessaire (mettre carry in du FA a 1)
--=====================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.piece.all;
USE piecewise.composants.all;

ENTITY comp_2 IS
PORT(a:IN std_logic_vector(WORD-1 downto 1);
   ctl,clk,rst:IN std_logic;
   b:OUT std_logic_vector(WORD downto 1));
END comp_2;

ARCHITECTURE behav OF comp_2 IS

SIGNAL propagation:std_logic_vector(WORD-1 downto 1);
BEGIN

G1:for i IN 1 TO word-1 GENERATE

 G2:IF i=1 GENERATE
  premiere_cellule:inv_cell
   PORT MAP(a=>a(i),b=>b(i),in_ctl=>ctl,out_ctl=>propagation(i),clk=>clk,rst=>rst);
 END GENERATE;

 G3:IF i>1 GENERATE
  cellule:inv_cell
   PORT MAP(a=>a(i),b=>b(i),in_ctl=>propagation(i-1),out_ctl=>propagation(i),clk=>clk,rst=>rst);
 END GENERATE;

END GENERATE;

b(word)<=propagation(word-1);

END behav;
```

```
--=============================================================
--VHDL pour le full adder a propagation de retenue pipelinee
--=============================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY full_adder IS

 GENERIC(nb_cell:INTEGER);

 PORT(
   a,b:in std_logic_vector(nb_cell downto 1);
   u:out std_logic_vector(nb_cell downto 1);
   Cin, clk,rst:IN std_logic);


END full_adder;

ARCHITECTURE struct OF full_adder IS

--**********************
--SIGNAUX DE CONNECTION
--**********************
SIGNAL ri,ro:std_logic_vector(nb_cell downto 1);

begin

G1:FOR i IN 1 to nb_cell GENERATE

 G2:IF i=1 GENERATE
 premier_cellule:cell
   port map(a(i),b(i),Cin,u(i),ri(i));
 END GENERATE;

 G3:IF i>1 GENERATE
  reg1:registre
   port map(ri(i-1),clk,rst,ro(i-1));
  cellule:cell
   port map(a(i),b(i),ro(i-1),u(i),ri(i));
 END GENERATE;


END GENERATE;

END struct;



--=============================================
--PYRAMIDE DE REGISTRES
```

```
--=======================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY pyramide IS
 generic(nb_reg:INTEGER);
 PORT( a:in std_logic_vector(nb_reg downto 1);
      b:out std_logic_vector(nb_reg downto 1);
      clk,rst:in std_logic);
END pyramide;

ARCHITECTURE struct OF pyramide IS

BEGIN

G1:FOR i in 2 to nb_reg GENERATE

 col_reg:colonne
  GENERIC MAP (i-1)
  PORT MAP(a(i),b(i),clk,rst);

END GENERATE;
b(1)<=a(1);
END struct;

--=======================================================
--PYRAMIDE DE REGISTRES (INVERSE)
--=======================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY pyramide2 IS
 generic(nb_reg:INTEGER);
 PORT( a:in std_logic_vector(nb_reg downto 1);
      b:out std_logic_vector(nb_reg downto 1);
      clk,rst:in std_logic);
END pyramide2;

ARCHITECTURE struct OF pyramide2 IS

BEGIN

G1:FOR i in 1 to nb_reg-1 GENERATE

 col_reg:colonne
  GENERIC MAP (nb_reg-i)
```

```
 PORT MAP(a(i),b(i),clk,rst);

END GENERATE;
b(nb_reg)<=a(nb_reg);
END struct;
--============================================================================
--BUFFER
--============================================================================
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY buf IS
    port( op : buffer std_logic;  ip : in std_logic);
end buf;

ARCHITECTURE behav OF buf IS
BEGIN
op<= ip AND '1';
END behav;

--============================================================================
--CELLULE ELEMENTAIRE DE L'ETAGE DE SORTIE
--============================================================================

LIBRARY ieee;
LIBRARY piecewise;
USE ieee.std_logic_1164.all;
USE piecewise.composants.all;

ENTITY out_cell IS
PORT(ip,end_acc,rst,clk:IN std_logic;
   op_ret,op_fwd,ctl_fwd:OUT std_logic);
END out_cell;

ARCHITECTURE behav OF out_cell IS

SIGNAL acc_et_clk,nacc_et_clk,inv_clk,rst_ret:std_logic;

BEGIN

acc_et_clk<=end_acc AND clk;
nacc_et_clk<= (NOT end_acc) AND clk;
inv_clk<=NOT clk;
rst_ret<=rst AND not acc_et_clk;

reg_fwd:registre
PORT MAP(a=>ip,b=>op_fwd,clk=>acc_et_clk,rst=>rst);

reg_ret:registre
PORT MAP(a=>ip,b=>op_ret,clk=>nacc_et_clk,rst=>rst_ret);

reg_ctl:registre
PORT MAP(a=>end_acc,b=>ctl_fwd,clk=>inv_clk,rst=>rst);
```

```
--FEEDBACK:PROCESS(nacc_et_clk,nacc_et_clk,rst,ip)
--reg:std_logic;
--BEGIN
-- IF rst='0' THEN
--  op_ret<='0';
-- ELSIF acc_et_clk='1' THEN
--   op_ret<='0';
-- ELSIF nacc_et_clk'EVENT and nacc_et_clk='1' THEN
--          op_ret<=ip;
-- END IF;
--END PROCESS;

--FORWARD:PROCESS(acc_et_clk,rst,ip)
--BEGIN
--  IF rst='0'THEN
--    op_fwd<='0';
--  ELSIF acc_et_clk'EVENT AND acc_et_clk='1' THEN
--   op_fwd<=ip;
--  END IF;
--END PROCESS;

--CONTROL:PROCESS(end_acc,rst,inv_clk)
--BEGIN
-- IF rst='0' THEN
--  ctl_fwd<='0';
-- ELSIF inv_clk'EVENT and inv_clk='1' THEN
--  ctl_fwd<=end_acc;
-- END IF;
--END PROCESS;

END behav;

--=============================================================================
--ETAGE DE SORTIE
--=============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;

ENTITY etage_sortie IS
GENERIC(nb_cell:INTEGER);
PORT(ip:IN std_logic_vector(nb_cell downto 1);
   op_ret,op_fwd:OUT std_logic_vector(nb_cell downto 1);
   end_acc,clk,rst:IN std_logic);
END etage_sortie;

ARCHITECTURE struct OF etage_sortie IS
SIGNAL acc:std_logic_vector(nb_cell downto 1);
BEGIN

G1:FOR i in 1 to nb_cell GENERATE
```

```
G2:IF i=1 GENERATE
premiere_cellule:out_cell
PORT
MAP(ip=>ip(i),op_ret=>op_ret(i),op_fwd=>op_fwd(i),end_acc=>end_acc,ctl_fwd=>acc(i),clk=>clk,rst=>rst)
;
END GENERATE;

G3:IF i>1 GENERATE
cellule:out_cell
PORT MAP(ip=>ip(i),op_ret=>op_ret(i),op_fwd=>op_fwd(i),end_acc=>acc(i-
1),ctl_fwd=>acc(i),clk=>clk,rst=>rst);
END GENERATE;

END GENERATE;

END struct;

--============================================================================
--MULTIPLIEUR-ACCUMULATEUR
--============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY piecewise;
USE piecewise.composants.all;
USE piecewise.piece.all;


ENTITY mac IS
PORT(
   a:in std_logic_vector(conversion downto 1);
   b:in std_logic_vector(word downto 1);
   end_acc,clk,rst:in std_logic;
   c:out std_logic_vector(word downto 1));
END mac;

ARCHITECTURE struct OF mac IS

SIGNAL m2p_u,m2p_v,p2c_u,p2c_v:data;
SIGNAL c2r_u,c2r_v,fa2r,r2fa,fa2es,es2fa,es2p,r2fa_u,r2fa_v:std_logic_vector(word downto 1);
SIGNAL signe,signe2,in_acc,out_acc,clk_inv,signe3:std_logic;


BEGIN

clk_inv<=not clk;
--*********************************
--INSERTION DE BUFFERS POUR LE SIGNE
--*********************************



buffer2:registre
```

```
 PORT MAP(signe,clk,rst,signe2);
buffer3:colonne
 GENERIC MAP(3)
 PORT MAP(in_acc,out_acc,clk_inv,rst);

mult:multiplieur
 GENERIC MAP(conversion,word)
 PORT MAP(a,b,m2p_u,m2p_v,clk,rst,end_acc,signe,in_acc);

pyramid_u:pyramide
 GENERIC MAP(WORD-1)
 PORT MAP(a=>m2p_u,b=>p2c_u,clk=>clk,rst=>rst);

pyramid_v:pyramide
 GENERIC MAP(WORD-1)
 PORT MAP(a=>m2p_v,b=>p2c_v,clk=>clk,rst=>rst);

complement_u:comp_2
 PORT MAP(p2c_u,signe,clk,rst,c2r_u);

complement_v:comp_2
 PORT MAP(p2c_v,signe,clk,rst,c2r_v);

registre_u:etage_reg
 GENERIC MAP(word)
 PORT MAP(a=>c2r_u,b=>r2fa_u,clk=>clk,rst=>rst);

registre_v:etage_reg
 GENERIC MAP(word)
 PORT MAP(a=>c2r_v,b=>r2fa_v,clk=>clk,rst=>rst);

fa1:full_adder
 GENERIC MAP(WORD)
 PORT MAP(r2fa_u,r2fa_v,fa2r,signe2,clk,rst);

pipeline:etage_reg
 GENERIC MAP(WORD)
 PORT MAP(a=>fa2r,b=>r2fa,clk=>clk,rst=>rst);

reg1:registre
 PORT MAP(signe2,clk,rst,signe3);

fa2:full_adder
 GENERIC MAP(WORD)
 PORT MAP(r2fa,es2fa,fa2es,signe3,clk,rst);

reg:etage_sortie
 GENERIC MAP(WORD)
 PORT MAP(ip=>fa2es,op_ret=>es2fa,op_fwd=>es2p,end_acc=>out_acc,clk=>clk,rst=>rst);

pyramid_s:pyramide2
 GENERIC MAP(WORD)
 PORT MAP(a=>es2p,b=>c,clk=>clk,rst=>rst);
end struct;
```

Annexe C : Programme VHDL de l'architecture

```
--Ce fichier contient l'architecture complete du ML-PNN sans l'apprentissage

LIBRARY ieee;
USE ieee.std_logic_1164.all;

--============================================================================
--PAQUETAGE POUR LES COMPOSANTS DE L'ARCHITECTURE DU RN
--============================================================================

--*********************************
--PAQUETAGE DES TYPES ET CONSTANTES
--*********************************

 PACKAGE piece2 IS

--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  CONSTANT word:INTEGER :=8;   --DETERMINE LA LONGUEUR DES MOTS BINAIRES
--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  CONSTANT convert:INTEGER :=8;       --DETERMINE LA LONGUEUR DES MOTS BINAIRES DU
                --CONVERTISSEUR
--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

--@@@@@@@@@@@@@@@@@@@@@@@@@
  CONSTANT S:INTEGER:=3;         --Nombre du neurone sur la couche cachee
--@@@@@@@@@@@@@@@@@@@@@@@@@

--@@@@@@@@@@@@@@@@@@@@@@@@@
  CONSTANT M:INTEGER:=2 ;         --Nombre de retards sur l'entree
--@@@@@@@@@@@@@@@@@@@@@@@@@

SUBTYPE data IS std_logic_vector(word-1 downto 1);

TYPE sig_vect IS ARRAY(INTEGER RANGE <>) of std_logic_vector(word-1 downto 1);


END piece2;

--*****************************************
--Les elements de l'architecture
--*****************************************

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;

PACKAGE neural IS

COMPONENT activation
PORT(a:in signed(word downto 1);b:out signed(word downto 1));
```

```
END COMPONENT;

COMPONENT multiplieur1
PORT(INa : IN signed(convert DOWNTO 1);
   INb : IN signed(word DOWNTO 1);
   output : OUT signed(word DOWNTO 1));
END COMPONENT;

COMPONENT additionneur
PORT(INa,INb:IN signed(word DOWNTO 1);output:OUT signed(word DOWNTO 1));
END COMPONENT;

COMPONENT registre
GENERIC(nb_bits:INTEGER);
PORT(
    a : in  signed(nb_bits DOWNTO 1);
    b : out signed(nb_bits DOWNTO 1);
                clk,rst:IN std_logic);
END COMPONENT;

COMPONENT mult_acc
PORT(INa : IN signed(convert DOWNTO 1);
   INb : IN signed(word DOWNTO 1);
   output : OUT signed(word DOWNTO 1);
   clk,end_acc,rst: in std_logic);
END COMPONENT ;

COMPONENT multiplieur2
PORT(INa,INb : IN signed(word DOWNTO 1);
   output : OUT signed(word DOWNTO 1));
END COMPONENT;

COMPONENT proc_ele
PORT(poidsW,poidsQ,som_partielle:IN signed(word DOWNTO 1);
   data:signed (convert DOWNTO 1);
   vers_PE:OUT signed(word DOWNTO 1);
   end_acc,clk,rst:IN std_logic);
END COMPONENT;

COMPONENT registre2
PORT(
    a,clk,rst : in  std_logic;
    b       : out std_logic);
END COMPONENT;

END neural;
```

```
--==========================================================================
--FONCTION D'ACTIVATION PIECEWISE
--==========================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;

ENTITY activation IS
PORT(a:in signed(word downto 1);b:out signed(word downto 1));
END activation;

ARCHITECTURE behav OF activation IS
BEGIN

P1:PROCESS(a)
        VARIABLE temp:std_logic;
        VARIABLE zeros:signed(word-2 DOWNTO 1);
BEGIN
        zeros:=(others=>'0');
        IF a(word)='1' AND a(word-1)='1' AND a(word-2)='1' THEN
                temp:='0';
        ELSIF a(word)='0' AND a(word-1)='0' AND a(word-2)='0'THEN
                temp:='0';
        ELSE
                temp:='1';
        END IF;

        IF temp='1'THEN
                b(word-1)<='1';
                b(word-2 downto 1)<=zeros;
        ELSE
                b(word-2 downto 2)<=a(word-3 downto 1);
                b(word-1)<=a(word);
                b(1)<='0';
        END IF;
        b(word)<=a(word);
END PROCESS;

END behav;

--==========================================================================
--Multiplieur du MAC
--==========================================================================
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;
```

```
ENTITY multiplieur1 IS
PORT(INa : IN signed(convert DOWNTO 1);
   INb : IN signed(word DOWNTO 1);
   output : OUT signed(word DOWNTO 1));
END multiplieur1;

ARCHITECTURE behav OF multiplieur1 IS
BEGIN
 mult:PROCESS(INa,INb)
  VARIABLE temp:signed(word+convert DOWNTO 1);
 BEGIN
  temp:=INa*INb;
  output(word)<=temp(word+convert);
  output(word-1 DOWNTO 1)<=temp(word+convert-2 DOWNTO convert);
 END PROCESS;
END behav;

--=============================================================================
--Additionneur
--=============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;

ENTITY additionneur IS
PORT(INa,INb:IN signed(word DOWNTO 1);output:OUT signed(word DOWNTO 1));
END additionneur;

ARCHITECTURE behav OF additionneur IS
BEGIN
 output<=INa+INb;
END behav;

--=============================================================================
--Registre
--=============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;

ENTITY registre IS
GENERIC(nb_bits:INTEGER);
PORT(
    a : in  signed(nb_bits DOWNTO 1);
    b : out signed(nb_bits DOWNTO 1);
```

```
                clk,rst:IN std_logic);
END registre;

ARCHITECTURE behav OF registre IS
BEGIN

PROCESS(clk,a,rst)
BEGIN

if rst='1' THEN
 IF clk'EVENT AND clk ='1' THEN
   b<=a;
 END IF;
ELSE
 b<=(others=>'0');
END IF;

END PROCESS;

END behav;

--=============================================================================
--Multiplieur accumulateur
--=============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;
USE piecewise.neural.all;

ENTITY mult_acc IS
PORT(INa : IN signed(convert DOWNTO 1);
   INb : IN signed(word DOWNTO 1);
   output : OUT signed(word DOWNTO 1);
   clk,end_acc,rst: in std_logic);
END mult_acc;

ARCHITECTURE struct OF mult_acc IS
 SIGNAL multB,result_mult,mult2add,accum,result_add:signed(word DOWNTO 1);
 SIGNAL multA:signed(convert DOWNTO 1);
 SIGNAL temp1,temp2:std_logic;

BEGIN
 regIN_A:registre
  GENERIC MAP(convert)
  PORT MAP(a=>INa,b=>multA,clk=>clk,rst=>rst);

 regIN_B:registre
  GENERIC MAP(word)
  PORT MAP(a=>INb,b=>multB,clk=>clk,rst=>rst);
```

```
mult:multiplieur1
 PORT MAP(INa=>multA,INb=>multB,output=>result_mult);

regMULT:registre
 GENERIC MAP(word)
 PORT MAP(a=>result_mult,b=>mult2add,clk=>clk,rst=>rst);

addi:additionneur
 PORT MAP(INa=>mult2add,INb=>accum,output=>result_add);

regACC:registre
 GENERIC MAP(word)
 PORT MAP(a=>result_add,b=>accum,clk=>clk,rst=>temp1);

regOUT:registre
 GENERIC MAP(word)
 PORT MAP(a=>result_add,b=>output,clk=>temp2,rst=>rst);

PROCESS(rst,end_acc,clk)
BEGIN
 temp1<= rst AND end_acc;
 temp2<= NOT end_acc AND clk;
END PROCESS;

END struct;

--=============================================================================
--Multiplieur du processeur elementaire
--=============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;

ENTITY multiplieur2 IS
PORT(INa,INb : IN signed(word DOWNTO 1);
   output : OUT signed(word DOWNTO 1));
END multiplieur2;

ARCHITECTURE behav OF multiplieur2 IS
BEGIN
 mult:PROCESS(INa,INb)
  VARIABLE temp:signed(2*word DOWNTO 1);
 BEGIN
  temp:=INa*INb;
  output(word)<=temp(2*word);
  output(word-1 DOWNTO 1)<=temp(2*word-2 DOWNTO word);
 END PROCESS;
END behav;

--=============================================================================
```

```
--Processeur elementaire
--=============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;
USE piecewise.neural.all;

ENTITY proc_ele IS
PORT(poidsW,poidsQ,som_partielle:IN signed(word DOWNTO 1);
   data:signed (convert DOWNTO 1);
   vers_PE:OUT signed(word DOWNTO 1);
   end_acc,clk,rst:IN std_logic);
END proc_ele;

ARCHITECTURE struct OF proc_ele IS

 SIGNAL  mac2act,act2reg,reg2mult,mult2reg,reg2add,add2reg:signed(word DOWNTO 1);

BEGIN

mac:mult_acc
 PORT MAP(INa=>data,INb=>poidsW,output=>mac2act,clk=>clk,end_acc=>end_acc,rst=>rst);

pcewis:activation
 PORT MAP(a=>mac2act,b=>act2reg);

regACT:registre
 GENERIC MAP(word)
 PORT MAP(a=>act2reg,b=>reg2mult,clk=>clk,rst=>rst);

mult:multiplieur2
 PORT MAP(INa=>reg2mult,INb=>poidsQ,output=>mult2reg);

regMULT:registre
 GENERIC MAP(word)
 PORT MAP(a=>mult2reg,b=>reg2add,clk=>clk,rst=>rst);

addition:additionneur
 PORT MAP(INa=>reg2add,INb=>som_partielle,output=>add2reg);

regOUT:registre
 GENERIC MAP(word)
 PORT MAP(a=>add2reg,b=>vers_PE,clk=>clk,rst=>rst);

END STRUCT;

--=============================================================================
--Registre de 1 bit
--=============================================================================
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;

ENTITY registre2 IS
PORT(
    a,clk,rst : in  std_logic;
    b         : out std_logic);
END registre2;

ARCHITECTURE behav OF registre2 IS
BEGIN

PROCESS(clk,a,rst)
BEGIN

if rst='1' THEN
 IF clk'EVENT AND clk ='1' THEN
   b<=a;
 END IF;
ELSE
 b<='0';
END IF;

END PROCESS;

END behav;

--=============================================================================
--Processeur avec 3 PEs sans apprentissage
--=============================================================================

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
LIBRARY piecewise;
USE piecewise.piece2.all;
USE piecewise.neural.all;

ENTITY pl_mnn IS
PORT(data:signed(convert DOWNTO 1);
   poidsW1,poidsW2,poidsW3,poidsQ1,poidsQ2,poidsQ3:signed(word DOWNTO 1);
   reconst:OUT signed(word DOWNTO 1);
   clk,end_acc,rst:IN std_logic);
END pl_mnn;

ARCHITECTURE struct OF pl_mnn IS
 SIGNAL  PE12PE2,zero,PE22PE3,PE2act:signed(word DOWNTO 1);
 SIGNAL  data2,data3:signed(convert DOWNTO 1);
 SIGNAL  end_acc2,end_acc3:std_logic;
BEGIN
zero<=(others=>'0');
 PE1:proc_ele
  PORT MAP(data=>data,poidsW=>poidsW1, poidsQ=>poidsQ1,vers_PE=>PE12PE2,
                  clk=>clk,rst=>rst,end_acc=>end_acc,som_partielle=>zero);
```

```
regDATA1:registre
 GENERIC MAP(convert)
 PORT MAP(a=>data,b=>data2,clk=>clk,rst=>rst);

regEND1:registre2
 PORT MAP(a=>end_acc,b=>end_acc2,clk=>clk,rst=>rst);

PE2:proc_ele
 PORT MAP(data=>data2,poidsW=>poidsW2,poidsQ=>poidsQ2,vers_PE=>PE22PE3,
                clk=>clk,rst=>rst,end_acc=>end_acc2,som_partielle=>PE12PE2);

regDATA2:registre
 GENERIC MAP(convert)
 PORT MAP(a=>data2,b=>data3,clk=>clk,rst=>rst);

regEND2:registre2
 PORT MAP(a=>end_acc2,b=>end_acc3,clk=>clk,rst=>rst);

PE3:proc_ele
 PORT MAP(data=>data3,poidsW=>poidsW3,poidsQ=>poidsQ3,vers_PE=>PE2act,
                clk=>clk,rst=>rst,end_acc=>end_acc3,som_partielle=>PE22PE3);

pcewis:activation
 PORT MAP(a=>PE2act,b=>reconst);
END STRUCT;
```

Annexe D : Programme C++ de l'algorithme PL-MNN à nombres réels

```
/*=====================================================
Programme pour l'adaptation du PLMNN
Martin Vidal
Université du Québec à Trois-Rivières
1999
=====================================================*/

#include <math.h>
#include "mex.h"

/*Declaration des fonctions*/

void piecewise  (
                double *z,
                double *zhat,
                int L
                );

void deriv      (
                double *z,
                double *pwderiv,
                int L
                );

void matvect    (
                double  *in1,
                double  *in2,
                double  *out,
                int     r1,
                int     r2,
                int     c2
                );

/*=================================================
Programme principal
=================================================*/

void plmnn      (
                double *s,
                double *ytild,
                int nbdata,
                double *erreur,
                double *mu,
                double *W,
                double *Q,
                int M,
                int L
                )
{

/*Declaration des variables*/

        double *z,*zhat,*x,*xhat,*c,*pwderiv,*input;
        double sum;
```

```
        int delai;
        int i,j,k,compt;

        input=(double *)malloc(M*sizeof(double));
        z=(double *)malloc(L*sizeof(double));
        zhat=(double *)malloc(L*sizeof(double));
        pwderiv=(double *)malloc(L*sizeof(double));
        x=(double *)malloc(sizeof(double));
        xhat=(double *)malloc(sizeof(double));
        c=(double *)malloc(sizeof(double));

/*PROPAGATION*/

        delai=M/2;

for(i=0;i<nbdata;i++)
        {
        input=ytild+(i*M);
        matvect(W,input,z,L,M,1);
        piecewise(z,zhat,L);
        sum=0;
        for(j=0;j<L;j++)
                {
                sum += *(Q+j)* *(zhat+j);
                }
        *x=sum;
        piecewise(x,xhat,1);

/*Détermination de l'erreur*/
        if(i<delai)
                *(erreur+i)=0;
        else
                *(erreur+i)=*(s+i-delai)-*xhat;



/*++++++++++++++++++++++++++++++++++++++++++++++++
Mise à jour des poids
++++++++++++++++++++++++++++++++++++++++++++++++*/
        if(*(erreur+i)!=0)
                {

/*Calculs des dérivés*/

                deriv(z,pwderiv,L);

                *c=*mu* *(erreur+i);

/*Entrée de la couche cachée*/

                compt=0;
                for(j=0;j<M;j++)
                        {
                        for(k=0;k<L;k++)
```

```
                                {
                                *(W+compt)+= *c * *(Q+k)* *(input+j)* *(pwderiv+k);
                                compt++;
                                }
                        }

/*Couche de sortie*/

                for(j=0;j<L;j++)
                        {*(Q+j)+= *c* *(zhat+j);}
                }
        else
                *(erreur+i)=0.00000000000001;
        }
        free(input);
        free(z);
        free(zhat);
        free(pwderiv);
        free(x);
        free(xhat);
        free(c);
}

/*==================================================================
Interface des variables
==================================================================*/

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )

{
/*Initialisation des variables*/

        double *erreur;
        int nbdata,M,L;
        double *s,*y_tild,*mu,*W,*Q;

/* Vérifie le nombre d'arguments de la fonction */

        if (nrhs != 5)
                mexErrMsgTxt("Wrong number of input arguments");
        else if (nlhs > 1)
                mexErrMsgTxt("Too many output arguments.");

/*Vérifie les dimensions des paramètres*/

        nbdata=mxGetN(prhs[1]);
        M=mxGetN(prhs[2]);
        L=mxGetM(prhs[2]);

/*Vérifie si les matrices sont compatibles*/

        if(L!=mxGetM(prhs[3]) | M!=mxGetM(prhs[1]))
                mexErrMsgTxt("Les matrices sont incompatibles");
```

```
/* Crée les matrices de sortie */

        plhs[0] = mxCreateDoubleMatrix(nbdata,1,mxREAL);

/* Assigne les pointeurs aux différents paramètres */

        erreur = mxGetPr(plhs[0]);
        s = mxGetPr(prhs[0]);
        y_tild = mxGetPr(prhs[1]);
        W = mxGetPr(prhs[2]);
        Q = mxGetPr(prhs[3]);
        mu = mxGetPr(prhs[4]);

/* Effectue le programme principal */
  plmnn(s,y_tild,nbdata,erreur,mu,W,Q,M,L);
}

/*==========================================================
SOUS-ROUTINES
==========================================================

Produit matrice*vecteur*/


void matvect(
                double*in1,
                double*in2,
                double*out,
                int             r1,
                int             r2,
                int             c2
                )
{
  int           i,j,k,count=0;
  doublesum;

  for (i=0; i< c2; i++)
        { for (j=0; j< r1; j++)
                { sum=0;
                  for (k=0; k< r2; k++)
                        { sum+=*(in1+r1*k+j) * *(in2+k+i*r2);
                        }
                  *(out+count)=sum;
            count++;
                }
        }
}


/*_____________________________________

Fonction d'activation*/
```

```
void piecewise  (
                double *z,
                double *zhat,
                int L
                )
{

/*Initialisation des variables*/

        int i;

/*Programme*/

        for(i=0;i<L;i++)
                {
                if(*(z+i)<(-4))
                        *(zhat+i)=-1;
                else if(*(z+i)>4)
                        *(zhat+i)=1;
                else
                        *(zhat+i)=0.25000* *(z+i);
                }
}

/*_____________________________________________

Dérivé de la fonction d'activation*/

void deriv      (
                double *z,
                double *pwderiv,
                int L
                )
{

/*Initialisation des variables*/

        int i;

/*Programme*/

        for(i=0;i<L;i++)
                {
                if(*(z+i)>(-4) && *(z+i)<4)
                        *(pwderiv+i)=0.25;
                else
                        *(pwderiv+i)=0;
                }
}
```

Annexe E : Programme C++ de l'algorithme PL-MNN à nombres complexes

```
/*=====================================================
Programme pour l'adaptation du PLMNN pour nombres complexes
Martin Vidal
Université du Québec à Trois-Rivières
1999
=====================================================*/


#include <math.h>
#include "mex.h"


/*Declaration des fonctions*/

void piecewise  (
                double *zr,
                double *zi,
                double *zhatr,
                double *zhati,
                int L
                );

void deriv      (
                double *zr,
                double *zi,
                double *pwderivr,
                double *pwderivi,
                int L
                );

void matvect    (
                double  *in1r,
                double  *in1i,
                double  *in2r,
                double  *in2i,
                double  *outr,
                double  *outi,
                int     r1,
                int     r2,
                int     c2
                );

void multcpx    (
                double *in1r,
                double *in1i,
                double *in2r,
                double *in2i,
                double *outr,
                double *outi
                );
```

```
/*=====================================================
Programme principal
=====================================================*/

void plmnn      (
                double *sr,
                double *si,
                double *ytildr,
                double *ytildi,
                int nbdata,
                double *erreurr,
                double *erreuri,
                double *mu,
                double *Wr,
                double *Wi,
                double *Qr,
                double *Qi,
                int M,
                int L
                )
{

/*Declaration des variables*/

        double *zr,*zi,*zhatr,*zhati,*xr,*xhatr,*pwderivr,*inputr,*xi,*xhati,*pwderivi,*inputi;
        double *dr,*di,*dWi,*dWr,*dQi,*dQr;
        int delai;
        int i,j,k,compt;

        inputr=(double *)malloc(M*sizeof(double));
        inputi=(double *)malloc(M*sizeof(double));
        zr=(double *)malloc(L*sizeof(double));
        zi=(double *)malloc(L*sizeof(double));
        zhatr=(double *)malloc(L*sizeof(double));
        zhati=(double *)malloc(L*sizeof(double));
        pwderivr=(double *)malloc(L*sizeof(double));
        pwderivi=(double *)malloc(L*sizeof(double));
        xr=(double *)malloc(sizeof(double));
        xi=(double *)malloc(sizeof(double));
        xhatr=(double *)malloc(sizeof(double));
        xhati=(double *)malloc(sizeof(double));
        dWr=(double *)malloc(sizeof(double));
        dWi=(double *)malloc(sizeof(double));
        dQr=(double *)malloc(L*sizeof(double));
        dQi=(double *)malloc(L*sizeof(double));
        dr=(double *)malloc(L*sizeof(double));
        di=(double *)malloc(L*sizeof(double));

/*PROPAGATION*/

for(i=0;i<nbdata;i++)
                {
        inputr=ytildr+(i*M);
        inputi=ytildi+(i*M);
```

```
        matvect(Wr,Wi,inputr,inputi,zr,zi,L,M,1);
        piecewise(zr,zi,zhatr,zhati,L);
        matvect(Qr,Qi,zhatr,zhati,xr,xi,1,L,1);
        piecewise(xr,xi,xhatr,xhati,1);

/*Détermination de l'erreur*/
                *(erreurr+i)=*(sr+i)-*xhatr;
                *(erreuri+i)=*(si+i)-*xhati;


/*++++++++++++++++++++++++++++++++++++++++++++++
Mise à jour des poids
++++++++++++++++++++++++++++++++++++++++++++++++*/



/*Calculs des dérivés*/
if(*(erreurr+i)>0.3 && *(erreuri+i)>0.3)
                {
                deriv(zr,zi,pwderivr,pwderivi,L);
                for(j=0;j<L;j++)
                        {
                        multcpx((erreurr+i),(erreuri+i),(Qr+j),(Qi+j),(dr+j),(di+j));
                        *(dr+j)=*(dr+j)* *(pwderivr+j);
                        *(di+j)=*(di+j)* *(pwderivi+j);
                        }


/*Entrée de la couche cachée*/

                compt=0;
                for(j=0;j<M;j++)
                        {
                        for(k=0;k<L;k++)
                                {
                                multcpx(dr+k,di+k,inputr+j,inputi+j,dWr,dWi);
                                *dWr=*dWr* *mu;
                                *dWi=*dWi* *mu;
                                *(Wr+compt)+= *(dWr);
                                *(Wi+compt)+= *(dWi);
                                compt++;
                                }
                        }

/*Couche de sortie*/

                for(j=0;j<L;j++)
                        {
                        multcpx(erreurr+i,erreuri+i,zhatr+j,zhati+j,dQr,dQi);
                        *dQr=*dQr**mu;
                        *dQi=*dQi**mu;
                        *(Qr+j)+=*dQr;
                        *(Qi+j)+=*dQi;
```

```
                        }

                }
        else
                {
                *(erreurr+i)=0.000000000001;
                *(erreurr+i)=0.000000000001;
                }
        }
        free(inputr);
        free(inputi);
        free(zr);
        free(zi);
        free(zhatr);
        free(zhati);
        free(pwderivr);
        free(pwderivi);
        free(xr);
        free(xi);
        free(xhatr);
        free(xhati);
        free(dWr);
        free(dWi);
        free(dQr);
        free(dQi);
        free(dr);
        free(di);

}

/*==========================================================================
Interface des variables avec Matlab
==========================================================================*/

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )

{
/*Initialisation des variables*/

        double *erreurr,*erreuri;
        int nbdata,M,L;
        double *sr,*y_tildr,*mu,*Wr,*Qr,*si,*y_tildi,*Wi,*Qi;

/* Vérifie le nombre d'arguments de la fonction */

        if (nrhs != 5)
                mexErrMsgTxt("Wrong number of input arguments");
        else if (nlhs > 1)
                mexErrMsgTxt("Too many output arguments.");

/*Vérifie les dimensions des paramètres*/

        nbdata=mxGetN(prhs[1]);
```

```
        M=mxGetN(prhs[2]);
        L=mxGetM(prhs[2]);

/*Vérifie si les matrices sont compatibles*/

        if(L!=mxGetN(prhs[3]) | M!=mxGetM(prhs[1]))
                mexErrMsgTxt("Les matrices sont incompatibles");

/* Crée les matrices de sortie */

        plhs[0] = mxCreateDoubleMatrix(nbdata,1,mxCOMPLEX);

/* Assigne les pointeurs aux différents paramètres */

        erreurr = mxGetPr(plhs[0]);
        erreuri = mxGetPi(plhs[0]);
        sr = mxGetPr(prhs[0]);
        si = mxGetPi(prhs[0]);
        y_tildr = mxGetPr(prhs[1]);
        y_tildi = mxGetPi(prhs[1]);
        Wr = mxGetPr(prhs[2]);
        Wi = mxGetPi(prhs[2]);
        Qr = mxGetPr(prhs[3]);
        Qi = mxGetPi(prhs[3]);
        mu = mxGetPr(prhs[4]);

/* Effectue le programme pricipal */
  plmnn(sr,si,y_tildr,y_tildi,nbdata,erreurr,erreuri,mu,Wr,Wi,Qr,Qi,M,L);
}


/*=============================================================
SOUS-ROUTINES
=============================================================

Produit matrice*vecteur*/


void matvect    (
                double  *in1r,
                double  *in1i,
                double  *in2r,
                double  *in2i,
                double  *outr,
                double  *outi,
                int     r1,
                int     r2,
                int     c2
                )

{
  int           i,j,k,count=0;
  doublesumi,sumr;
```

```
  for (i=0; i< c2; i++)
        {
                for (j=0; j< r1; j++)
                {
                        sumi=0;
                        sumr=0;

                  for (k=0; k< r2; k++)
                                {
                                sumr+=*(in1r+r1*k+j) * *(in2r+k+i*r2)-*(in1i+r1*k+j) * *(in2i+k+i*r2);
                                        sumi+=*(in1r+r1*k+j) * *(in2i+k+i*r2)+*(in1i+r1*k+j) *
*(in2r+k+i*r2);
                                }
                        *(outr+count)=sumr;
                        *(outi+count)=sumi;
                        count++;
                        }
                }
}

/*_____

Fonction d'activation*/

void piecewise  (
                double *zr,
                double *zi,
                double *zhatr,
                double *zhati,
                int L
                )
{

/*Initialisation des variables*/

        int i;

/*Programme*/

        for(i=0;i<L;i++)  /*Partie réelle*/
                {
                if(*(zr+i)<(-4))
                        *(zhatr+i)=-1;
                else if(*(zr+i)>4)
                        *(zhatr+i)=1;
                else
                        *(zhatr+i)=0.25000* *(zr+i);


                                /*Partie imaginaire*/

                if(*(zi+i)<(-4))
```

```
                *(zhati+i)=-1;
        else if(*(zi+i)>4)
                *(zhati+i)=1;
        else
                *(zhati+i)=0.25000* *(zi+i);
        }

}

/*_____________________________________________

Dérivé de la fonction d'activation*/

void deriv      (
                double *zr,
                double *zi,
                double *pwderivr,
                double *pwderivi,
                int L
                )
{

/*Initialisation des variables*/

        int i;

/*Programme*/

        for(i=0;i<L;i++)  /*Partie réelle*/
                {
                if(*(zr+i)>(-4) && *(zr+i)<4)
                        *(pwderivr+i)=0.25;
                else
                        *(pwderivr+i)=0;
                }
        for(i=0;i<L;i++)  /*Partie imaginaire*/
                {
                if(*(zi+i)>(-4) && *(zi+i)<4)
                        *(pwderivi+i)=0.25;
                else
                        *(pwderivi+i)=0;
                }

}
/*_____________________________________________
Produit par le conjugué*/

void multcpx    (
                double *in1r,
                double *in1i,
                double *in2r,
                double *in2i,
                double *outr,
                double *outi
```

```
                    )

{
*outr=*in1r**in2r+*in1i**in2i;
*outi=*in1i**in2r-*in1r**in2i;
}
```