République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique
DER de Génie Electrique et Informatique

Département Electronique

THESE
Présentée par

Mlle Nouma IZEBOUDJEN
Ingénieur d'état en électronique, USTHB

Pour l'obtention du

Grade de MAGISTER EN ELECTRONIQUE
Option : Télécommunications

Thème

CONCEPTION ET IMPLEMENTATION EN FPGA D'UN CLASSIFICATEUR NEURONAL DES ARYTHMIES CARDIAQUES

Soutenue-le : 06 /09 /1999 devant le jury composé de :

Mr D.BERKANI	Professeur (E.N.P)	Président
Mr A.FARAH	Professeur (E.N.P)	Rapporteur
Mr A. BELOUCHRANI	Docteur d'état (E.N.P)	Examinateur
Mr M.MEHENNI	Maître de conférences (E.N.P)	Examinateur
Mr A. CHOHRA	Docteur d'état (C.D.T.A)	Examinateur

A mes parents bien veillants

A mes frères et sœurs

A tous ceux qui me sont chers,

Je dédie ce labeur

REMERCIEMENTS

Tout d'abord, je tiens à remercier Mr H. Bessalah, directeur du CDTA, qui m'a autorisé à m'inscrire en magister à l'ENP.

Je tiens à rèmercier très vivement mon directeur de thèse Monsieur Ahcène Farah, Professeur à l'ENP, de m'avoir accepté au laboratoire Techniques Digitales et Systèmes, pour sa collaboration, pour avoir défini et orienté le sujet, pour son suivi et ses conseils qui ont permis de mener cette thèse à son aboutissement.

Je tiens à remercier, Monsieur D. Berkani, Professeur à l'ENP pour l'honneur d'avoir accepté de présider le jury de cette thèse.

Mes remerciements s'adressent aussi à : Monsieur A. Belouchrani, Docteur d'état à l'ENP ; Monsieur A. Chohra Docteur d'état au CDTA et Monsieur M. Mehenni, Maître de conférence à l'ENP, qui ont bien voulu examiner ce travail et participer au jury.

Particulièrement, Monsieur El-haffaf Youcef Ihcène ex-responsable de l'équipe ASIC du CDTA, attaché de recherche-chef du projet : Visualisation acquisition et traitement de l'ECG sous environnement PC-Windows dans lequel s'inscrit mon sujet de magister et Monsieur A. Chohra ont participé à l'origine à définir le sujet au sein du CDTA. Je tiens à les remercier très vivement pour les efforts fournis, leur esprit scientifique, leur compréhension et leurs encouragements.

Aussi je tiens à remercier Monsieur M. T. Bellaroussi, Chargé de recherche- responsable du laboratoire micro-électronique-CDTA, pour ses encouragements, sa disponibilité, son soutien moral et pour la documentation fournie.

Enfin, je tiens à remercier :

Mlles H. Boumeridja et S. Titri, chargées d'études au laboratoire Micro-électronique -CDTA, de m'avoir aider à faire la synthèse VHDL des réseaux de neurones.

Mlle F. Louiz, chargée d'études au laboratoire Micro-électronique -CDTA, de m'avoir initier au langage de programmation "C".

Ainsi que tous mes autres collègues du laboratoire micro-électronique- CDTA, qui ont contribué de près ou de loin à l'aboutissement de ce travail.

AVANT PROPOS

Les travaux présentés dans cette thèse, dirigée par le Professeur A.Farah, ont été menés au laboratoire de Micro-électronique du Centre de Développement des Technologies Avancées (CDTA), en collaboration avec le laboratoire Techniques Digitales et Systèmes (LTDS) de l'Ecole National Polytechnique (ENP).

ملخص

الهدف الأساسي من هذا البحث يتمثل في تصميم و إدماج دارة مصغرة داخل حقل شبكة منطقية مبرمجة تسمى (FPGA) موجهة لترتيب الإضطرابات القلبية و هذا باستعمال مقاربة الشبكات العصبية .

لكي نتمكن من ترتيب أكبر عدد من حالات الإضطرابات القلبية نقترح مرتب يتكون من مرتبين فرعيين متسلسلين: مرتب فرعي مرفولوجي و مرتب فرعي زمني. لتحقيق وظيفة الترتيب, نستعمل خوارزمية الإنتشار الرجعي للخطأ « retropropagation du gradient » (RPG)

ينقسم هذا العمل إلى جزئين :جزء برمجي و جزء عتادي. الجزء البرمجي وظيفته الأساسية تتمثل في تمرين المرتب. لهذا الغرض تم تطوير برنامج مكتوب بلغة البرمجة المسماة لغة C.

في الجزء العتادي, نعتبر إدماج مرحلة التعميم للخوارزمي (RPG) . نقترح هندسة متوازية و منتظمة لإدماج الشبكة العصبية. لغرض إنجازهذه الهندسة, إخترنا إستعمال منهجية تنازلية ترتكز على التركيب المنطقي باستعمال لغة الوصف VHDL.

نقترح وصف VHDL مرن ومتغير للشبكة العصبية بحيث يسهل تكييفه لتطبيقات أخرى. الوصف VHDL للمرتب تم إدماجه في دارة FPGA لعائلة XC4000-XILINX.

Abstract

The main objective of this work is the design and implementation of a FPGA integrated circuit for classification of the cardiac arrhythmia's, through the use of the neural network approach.
In order to deal with a wide variety of arrhythmia's, we propose a classifier, which is composed of two cascaded sub classifiers: a morphological sub classifier and a temporal one. To carry out the classification task, we use the back propagation algorithm (RPG). The work is divided in two parts: a software part and a hardware part. The essential task of the software part is to carry out the training of the classifier. For that, a program has been developed using the C language. In the hardware part, we consider implementation of the generalization phase of the RPG algorithm. We propose a parallel and regular architecture for the neural network implementation. In order to validate this architecture, we have opted for a top down approach based on logic synthesis and using the VHDL language. We propose a flexible and parametric VHDL description of the neural network, which can be easily adapted to other applications. The VHDL description of the classifier is mapped into the FPGA XILINX XC4000 family circuit.

Résumé

L'objectif essentiel de ce travail est la conception et l'implémentation en FPGA (Field Programmable Gate Array) d'un circuit intégré dédié à la classification des arythmies cardiaques par l'approche des réseaux de neurones. Afin de pouvoir classer un grand nombre d'arythmies, nous proposons un classificateur composé de deux sous classificateurs montés en cascade : un sous classificateur de morphologie et un sous classificateur temporel. Pour réaliser la classification, nous utilisons l'apprentissage par l'algorithme de la rétropropagation du gradient (RPG). Le travail est divisé en deux parties : une partie software et une partie hardware. La partie software a pour tâche essentielle de réaliser l'apprentissage du classificateur. Pour cela un programme a été développé en utilisant le langage C. Dans la partie hardware nous considérons l'implémentation de la phase de généralisation de l'algorithme RPG. Nous proposons une architecture parallèle et régulière pour l'implémentation du réseau de neurone. Afin de valider cette architecture nous avons opté pour une approche descendante basée sur la synthèse logique et l'utilisation du langage de description VHDL. Nous proposons une description VHDL flexible et parametrée du réseau de neurone qui peut être facilement adaptée à d'autres applications. La description VHDL du classificateur est implémentée sur un circuit FPGA de la famille XC4000 de XILINX.

Mot clés : Classification, arythmies cardiaques, réseaux de neurones, implémentation FPGA

TABLE DES MATIERES

INTRODUCTION

L'évolution de l'instrumentation médicale s'accélère grâce à l'évolution des technologies de l'électronique et de l'informatique. L'avancée technologique permise par les microprocesseurs et la miniaturisation a fait apparaître des systèmes de reconnaissance automatique. Le nombre élevé de malades suivis quotidiennement et, par conséquent, la multitude de cas se présentant, rendent désormais indispensable l'utilisation de ces systèmes d'aide à la surveillance et au diagnostic. Leur utilité dans les hôpitaux, tant pour les médecins, les infirmiers et les étudiants en médecine que pour le malade lui-même, n'est plus à démontrer.

Actuellement, le succès commercial remporté par les systèmes de reconnaissance automatique est beaucoup plus lié à leur possibilité de recherche, d'extraction et de stockage de données, qu'à leur possibilité d'analyse et de diagnostic (classification). Cependant, de par le monde des efforts de recherche considérables sont consentis afin de mettre en œuvre des classificateurs fiables.

Dans cet objectif, notre travail, présenté dans cette thèse consiste en la conception en FPGA (Field Programmable Gate Array) d'un système intégré dédié à la classification des arythmies cardiaques par une nouvelle approche : les réseaux de neurones artificiels.

Historiquement, l'intérêt porté aux réseaux de neurones provient du désir de saisir les principes menant à la compréhension des fonctions du cerveau, d'une part, et de construire des tâches complexes d'autre part.

Aujourd'hui, les réseaux de neurones trouvent leur application dans divers domaines tels : l'identification, la classification, la reconnaissance des formes... etc.

Dans le domaine de la classification, et plus particulièrement la classification des signaux électrocardiographiques les travaux de recherche montrent que les classificateurs réalisés à base d'algorithmes neuronaux donnent des résultats supérieurs à tous les autres classificateurs classiques (méthodes déterministes, statististiques, syntaxiques, systèmes experts, logique flou). En effet, le signal ECG est un signal issu d'un système non linéaire (le corps humain), et présente une variation au cours du temps pour un même individu et entre plusieurs individus. Les réseaux de neurones, par leur propriété d'apprentissage et de généralisation, offrent la possibilité d'utiliser des modèles non linéaires, l'habilité de lecture et d'auto-organisation et la possibilité de lire des formes en réponse à de nouvelles formes.

Selon les performances requises de l'application, les réseaux de neurones peuvent être implémentés en software, en hardware ou en combinant ces techniques.

Dans ce contexte, notre travail consiste à prendre en charge les parties software et hardware du classificateur neuronale des arythmies cardiaques. Le classificateur, nous l'avons baptisé CNAC.

La partie software à pour tâche l'apprentissage de CNAC ; celui ci étant basé sur l'algorithme de la rétro propagation du gradient (RPG).

La partie hardware de CNAC consiste à implémenter la phase de généralisation de l'algorithme RPG en utilisant la technologie FPGA.

Pour l'implémentation de CNAC, et vu la complexité des réseaux de neurones nous avons opté pour une approche descendante de conception basée sur la synthèse de haut niveau et la description VHDL. Cette dernière est un processus automatique permettant de transformer un circuit modélisé par un langage de description de haut niveau (VHDL) en une réalisation physique du circuit.

Les réseaux de neurones étant au cœur de notre implémentation, nous avons pensé qu'il était intéressant ou tout au moins convenable, de consacrer le premier chapitre en une étude générale des réseaux de neurones, dans lequel nous présentons sommairement, les réseaux de neurones en particulier le perceptron multicouche basé sur la rétroprpagation du gradient, que nous utilisons pour notre application. Nous présentons, dans le même chapitre les implémentations software et hardware des réseaux de neurones.

Le chapitre II, est consacré à la classification automatique des arythmies cardiaques. Après un aperçu sur les principes généraux de la classification, nous présentons une étude de synthèse sur les travaux de classification de l'ECG de manière générale et en particulier les arythmies cardiaques.

A travers cette étude nous proposons, dans le chapitre III, un classificateur neuronal des arythmies cardiaques, CNAC. Le choix algorithmique étant basé sur la rétropropagation du gradient, l'apprentissage du classificateur et les résultats de la classification obtenus sont discutés dans ce chapitre.

Dans le chapitre IV, nous présentons l'implémentation digitale du CNAC en FPGA. Cette dernière nécessite des connaissances de base sur la technologie FPGA, la synthèse de haut niveau et le langage VHDL. La première partie du chapitre résume ces principes, la deuxième partie concerne l'implémentation hardware du CNAC.

Enfin nous terminons par une conclusion générale.

Chapitre I

RESEAUX DE NEURONES DIGITAUX

I.1 Introduction

Ces dernières années ont vu l'émergence de nouvelles techniques regroupées sous le nom de connexionisme ou réseaux de neurones artificiels, dont le principe directeur est l'utilisation d'unités de calculs élémentaires fortement interconnectées et dont la connaissance réside précisément dans les interconnexions. L'idée sous-jacente et assez ambitieuse étant de copier l'architecture du cerveau. Ce dernier, qu'il soit humain ou animal est en effet le seul exemple de «machine» intelligente que nous connaissions. Il peut donc paraître naturel de s 'en inspirer non seulement au niveau fonctionnel, mais aussi au niveau architectural.

Le premier problème de la formalisation du fonctionnement du cerveau est celui de l'apprentissage, c'est à dire de la capacité d'acquérir de nouveaux comportements ou d'en modifier par expérience. L'enjeu, dont est actuellement l'objet de l'intelligence artificielle a renouvelé l'intérêt pour l'apprentissage. En effet, à mesure que les techniques de représentation des connaissances se font de plus en plus nombreuses et efficaces, le principal goulot d'étranglement dans la réalisation d'un système basé sur la connaissance est précisément l'acquisition de cette connaissance. La construction d'une base de connaissance nécessite la mise en œuvre d'importants moyens humains et constitue la part la plus importante des efforts de conception d'un système expert. D'où l'intérêt d'un système capable de construire sa base de connaissance à partir de données brutes.

L'origine des premières méthodes d'apprentissage date de 1943 avec les travaux de Warren MacCulloh et Walter Pitts [1] dont l'idée était d'expliquer comment un neurone très simplifié peut effectuer des inférences élémentaires et comment un réseau de ces neurones peut réaliser certaines fonctions logiques (OU inclusif, AND, NOT...). Les réseaux ainsi constitués, représentaient le model du cerveau. En 1949, D.O.Hebb présenta la théorie d'apprentissage pour la réalisation des mémoires associatives où le renforcement des connections synaptiques entre les neurones sont modifiés par des exemples d'entrées [2].

La première machine adaptative ayant connu un certain succès est le perceptron de Rosenblatt en 1960 [3]. Parallèlement, d'autres méthodes d'apprentissage étaient proposées, en particulier la célèbre procédure de Widrow et Hoff ou «Adaline » en 1960, basée sur la minimisation itérative d'un critère quadratique [4].

La recherche dans cette voix s'est poursuivie jusqu'à 1969 où deux mathématiciens, Minsky et Papert ont démontré les limites théoriques du perceptron pour résoudre les problèmes non-linéairement séparables [5]. Chercheurs et investisseurs se sont tournés vers l'approche de l'intelligence artificielle, qui semblait plus prometteuse.

Le renouveau actuel des réseaux de neurones est dû à des contributions originales, comme celles d'Hopfield en 1982 [6] qui, après avoir remarqué la similarité entre les réseaux du type proposé par Mc Culloch et Pitts avec un système élémentaire à moment magnétique ou spins, a étudié la capacité de stockage et de restauration des informations «mémoires associatives». Ce travail intéressa les physiciens en raison de l'analogie du modèle de Hopfield avec les spins. Il est intéressant de noter que ce modèle ne levait pas les limitations du perceptron et de ses variantes. Malgré cela, le percetron et les raisons de son échec semblaient oubliés. Par la suite Hinton et al. proposèrent la «machine de Boltzman», le premier modèle levant les limitations du percetron de façon satisfaisante. Les «machines de Boltzman» utilisent un ensemble de cellules dites cachées dont le rôle est de calculer les

variables intermédiaires permettant de réaliser des fonctions non-linéarirement séparables [7]. Malheureusement, la convergence de l'algorithme s'avère extrêmement longue, en raison du caractère probabiliste des éléments utilisés. Ce défaut à été corrigé par l'algorithme de la rétropropagation du gradient proposé sous diverses formes dans [8] et [9].

Plus récemment, les nouvelles découvertes en neurobiologie et l'intérêt explosif du traitement parallèle en plus du développement de la technologie VLSI ont donné une grande poussée au domaine des réseaux de neurones.

Dans ce chapitre, nous présentons un aperçu général sur les réseaux de neurones en commençant par définir les concepts de base, la construction des réseaux de neurones de manière générale, le perceptron multicouche ainsi que quelques règles d'apprentissage. Nous présentons par la suite l'implémentation des réseaux de neurones, les différents domaines d'application et enfin nous terminerons par une conclusion.

I.2 Réseaux de neurones [10], [11], [12], [13]

I.2.1 Définitions

I.2.1.1 Modèle biologique du neurone

L'élément fonctionnel de base du système nerveux est le neurone. D'une espèce animale à une autre, les neurones présentent des différences notoires ; à l'intérieur d'une même espèce, on dénombre de nombreux types de neurones. Les différences portent tant sur des aspects anatomiques que fonctionnels. Cependant des points communs sont à la base de l'archétype de la cellule nerveuse que nous présentons ici. La Figure I.1.a représente l'anatomie d'un neurone.

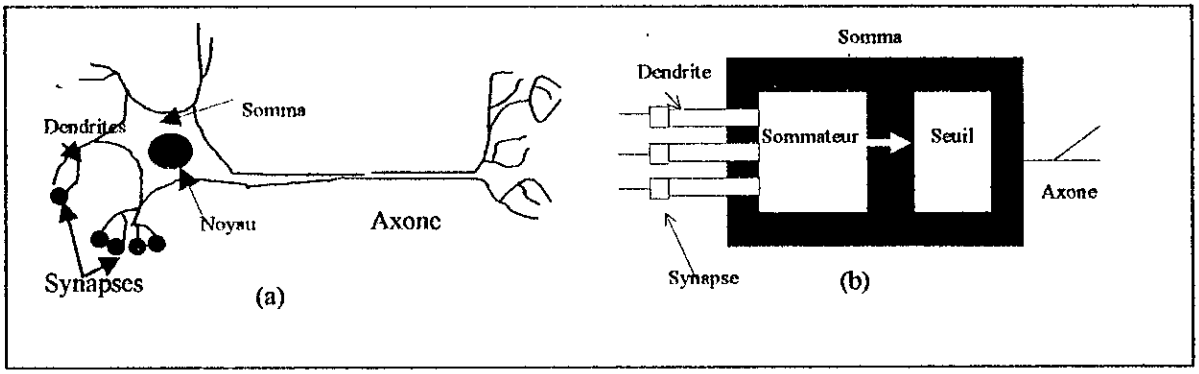


Figure I.1. (a) Anatomie d'un neurone. (b) schéma simplifié du fonctionnement du neurone.

Un neurone est une cellule. Autour du noyau, on trouve le corps cellulaire (somma). Celui-ci se prolonge par un axone unique et comporte de nombreuses dendrites qui constituent son organe «d'entrée ». L'influx nerveux est assimilable à un signale électrique, se propageant dans les neurones de la manière suivante :

- Les dendrites reçoivent l'influx nerveux d'autres neurones.
- Le neurone évalue alors l'ensemble de la stimulation qu'il reçoit (c'est à dire sa dépolarisation par rapport à l'extérieur).

- En fonction de cette stimulation (si la dépolarisation est suffisante > -50mV par exemple), le neurone transmet ou non un signal du type «tout ou rien » le long de son axone. On dira alors que le neurone est ou non excité.
- L'excitation du neurone est propagée le long de l'axone jusqu'aux autres neurones ou fibres musculaires qui y sont connectés via les synapses. La Figure I.1.b est une interprétation schématique du fonctionnement du neurone de la Figure I.1.a.

I.2.1.2 Modèles mathématiques des neurones

Il existe deux grands modèles le modèle discret et le modèle continu.

Modèle discret

La plupart des algorithmes connexionistes utilisés en ce moment sont de ce modèle. Le neurone est modélisé par deux fonctions (Figure I.2) :

1. Une fonction de sommation qui élabore un potentiel P, égal à la somme pondérée des entrées x_n du neurone.

$$p = \sum_n W_n x_n \tag{1.1}$$

Où, W_n représentent les poids synaptiques.

2. Une fonction seuil ou d'activation, *f*, qui calcule l'état de sortie S du neurone en fonction de son potentiel :

$$S = f(p) \tag{1.2}$$

La fonction d'activation peut être linéaire ou non : fonction linéaire, fonction sigmoïde, fonction seuil ou autres (Figure I.3).

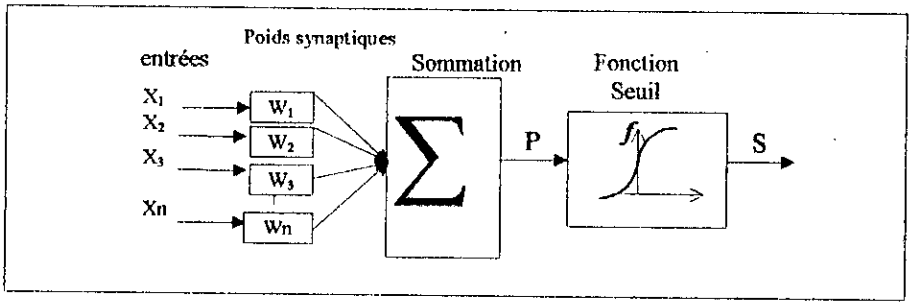


Figure I.2. model mathématique du neurone

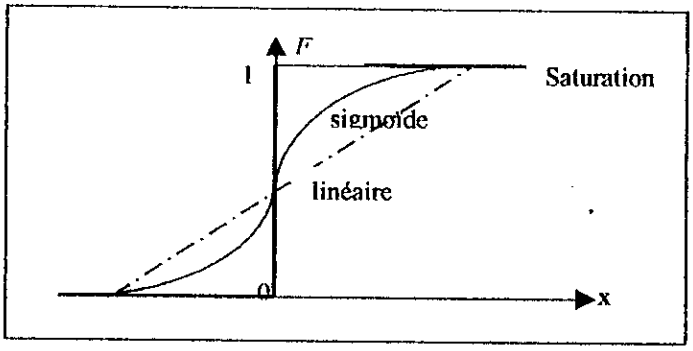


Figure I.3. Différentes formes de fonction d'activation

Modèle continue

Le modèle décrit par l'équation (1.1) est implicitement à temps discret, on suppose un signal constant à l'entrée jusqu'à ce que la sortie soit calculée. C'est à dire que la bande passante de du neurone est supposée très grande par rapport à celles des signaux d'entrée. Ainsi la caractéristique du neurone est atemporelle.
Dans le modèle continu, l'équation (1.1) est remplacée par l'équation différentielle :

$$\partial p/\partial t = \sum_n W_n x_n(t) \tag{1.3}$$

et $$S = f(t) * p(t) \tag{1.4}$$

Où * est l'opérateur de convolution. La fonction $f(t)$ représente la réponse impulsionnelle du neurone ; la sortie S dépend alors de l'activité passée de la cellule.

I.2.2 Construction des réseaux de neurones

Un réseau de neurones peut être représenté par un graphe direct composé d'un ensemble de nœuds ou éléments processeurs, fortement interconnectés par des liens orientés ou connexions. Les réseaux de neurones peuvent se distinguer par :

- l'architecture
- le mode d'apprentissage

I.2.2.1 Architecture des réseaux de neurones

Du point de vue architectural (Figure I.4), les réseaux de neurones peuvent être regroupés en deux catégories :

- Les réseaux à circulation de l'information vers l'avant « feed forward networks » dans lesquels les neurones sont organisés en couches. Dans cette catégorie, on distingue les réseaux possédant une seule couche (exp. perceptron) et les réseaux multicouches possédant une couche d'entrée, une couche de sortie et une ou plusieurs couches cachées.
- Les réseaux récurrents ou bouclés «recurrent /feedback networks » dans lesquels sont classés les réseaux compétitifs, le réseau de Kohonen, le réseau de Hopfield et les modèles ART "théorie de la résonance artificielle".

I.2.2.2 Apprentissage des réseaux de neurones

L'apprentissage permet au réseau de neurones de modifier sa structure interne (poids synaptique) pour s'adapter à l'environnement de l'application.
Il existe trois modes d'apprentissage : **supervisé, renforcé** et **non supervisé**

Dans l'apprentissage **supervisé**, un professeur qui connaît parfaitement la sortie désirée ou correcte, guide le réseau en lui apprenant à chaque étape le bon résultat. Donc l'apprentissage ici, consiste à comparer le résultat obtenu avec le résultat désiré, puis à ajuster les poids des connexions pour minimiser la différence entre les deux. C'est par exemple le cas du réseau multicouche à rétropropagation du gradient ou encore le cas des fonctions à base radiale.

Dans l'apprentissage **non supervisé**, le réseau modifie ses paramètres en tenant compte seulement des informations locales. Ces méthodes n'ont pas besoin de sorties désirées préétablies. C'est par exemple, le cas des cartes auto organisatrices de Kohonen.

Dans l'apprentissage **renforcé**, un professeur est supposé présent, mais la réponse exacte n'est pas présentée au réseau. Cependant, on indique au réseau s'il a calculé la bonne réponse ou non. Le réseau doit donc utiliser cette information pour améliorer ses performances. Typiquement, les poids synaptiques dont les unités possèdent la bonne réponse sont renforcés et ceux dont les unités possèdent la mauvaise réponse sont rejetés. C'est le cas par exemple des systèmes ART.

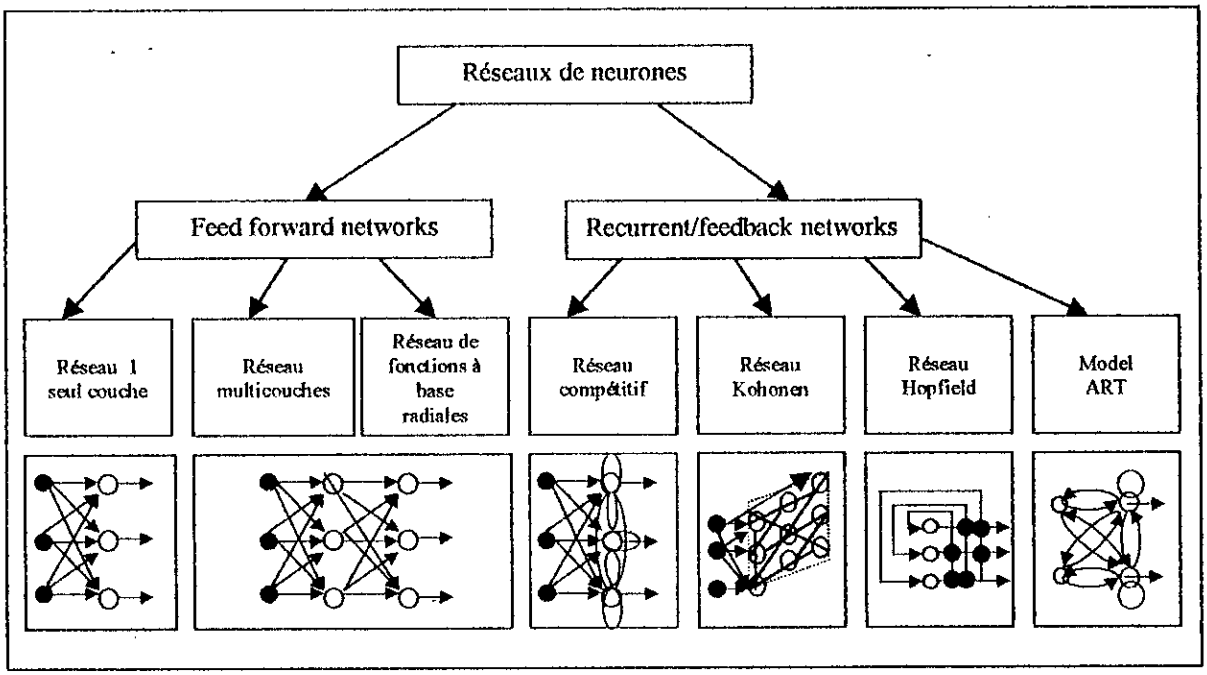


Figure I.4. Différentes architectures des réseaux de neurones.

La méthode d'ajustement des poids synaptiques pendant l'apprentissage du réseau peut être choisie dans une gamme variée de règles d'apprentissage dont les plus connues sont citées ci dessous :

1. La règle de Hebb (ou règle de corrélation)

L'apprentissage de Hebb est le premier mécanisme d'évaluation des synapses, proposée par D.O Hebb en 1949. Une interprétation de cette règle pour les réseaux de neurones est la suivante : Si deux neurones connectés entre eux sont activés en même temps, la connexion qui les relie doit être renforcée, dans le cas contraire elle n'est pas modifiée. Sa formulation est la suivante :

$$W_{ij}(t+1) = W_{ij}(t) + \eta S_i S_j \qquad (1.5)$$

Où, $W_{ij}(t)$ est le poids de la connexion reliant les neurones S_i et S_j.

η :est un nombre compris entre 0 et 1, représentant le taux d'apprentissage.

t: représente l'étape d'apprentissage.

2. La règle du perceptron

Dans le cas du perceptron, La fonction d'activation est une fonction discrète. Les sorties prennent des valeurs binaires (0 ou 1). La règle d'apprentissage du perceptron est la suivante :

$$\begin{cases} W_{ij}(t+1) = W_{ij}(t) + \eta x_i & \text{Si la sortie actuelle est égale à 0 et doit être égale à 1} \\ W_{ij}(t+1) = W_{ij}(t) - \eta x_i & \text{Si la sortie actuelle est égale à 1 et doit être égale à 0} \\ W_{ij}(t+1) = W_{ij}(t) & \text{Si la sortie est correcte} \end{cases} \quad (1.6)$$

Où, $\eta > 0$ représente le coefficient d'apprentissage,

t : l'étape d'apprentissage

x_i : l'entrée du neurone i.

W_{ij} : le poids synaptique ou connexion entre neurone i et le neurone j

L'inconvénient majeur du perceptron est qu'il ne peut s'appliquer que lorsque les classes sont linéairement séparables.

3. La règle de Widrow-Hoff ou règle delta

Nous avons vu, que le perceptron est limité à des sorties binaires, widrow et Hoff ont étudié l'algorithme d'apprentissage du perceptron en considérant une fonction d'activation continue et différentielle. Cette règle est aussi connue sous le nom de la méthode des moindres carrées ou en anglais "least mean square (LMS)" ou encore règle delta. Le principe est le suivant :

- Calculer l'erreur quadratique selon la formule :

$$E = \sum_{j=1}^{n} (d_j - y_j)^2 \quad (1.7)$$

Avec $y_j = \sum_{i=1}^{m} x_i W_{ji}$ (1.8)

- Minimiser cette erreur en modifiant les poids de chaque neurone suivant la règle :

$$W_{ji}(t+1) = W_{ji}(t) + \eta x_i (d_j - y_j) \quad (1.9)$$

Où,

n: le nombre de neurones à la sortie

d_j : est la sortie désirée

y_j : est la sortie calculée

x_i : l'entrée i du neurone j

m: le nombre de neurones à l'entrée.

η: coefficient d'apprentissage.

La règle de Widrow-Hoff pose le problème de l'apprentissage comme un problème de minimisation de fonction (Erreur) globale.

4. La règle delta généralisée

La règle delta généralisée ou encore appelée règle de la rétropropagation du gradient est une généralisation de la règle de Widrow Hoff et s'applique à un réseau multicouche utilisant des fonctions d'activation différentiables et un apprentissage supervisé. Nous verrons en détail (section 1.2.3) la formulation de cette règle.

5. L'apprentissage compétitif

L'apprentissage compétitif se fait par coopération et compétition entre les neurones. Les algorithmes d'apprentissage compétitif tentent de minimiser une fonction de distorsion entre un vecteur d'entrée X_i et un vecteur de poids W_{ji} . La règle de modification des poids synaptiques est donnée par l'équation suivante :

$$W_{ji}(t+1) = W_{ji}(t) + \eta(X_i - W_{ij}^{+}) \tag{1.10}$$

W_{ji}, : le poids synaptique entre le neurone *i* et neurone *j*
η : le coefficient d'apprentissage
t : le temps d'apprentissage.
W_{ij}^{+} : le poids synaptique du neurone gagnant

I.2.3 Le perceptron multicouches (MLP)

On considère que le MLP [14] est constitué de trois couches de neurones ; la première couche avec $\mathbf{n_0}$ entrées et $\mathbf{n_1}$ unités (neurones), la seconde avec $\mathbf{n_2}$ unités et la sortie avec $\mathbf{n_3}$ unités.

La Figure I.5 montre une représentation fonctionnelle de l'algorithme.

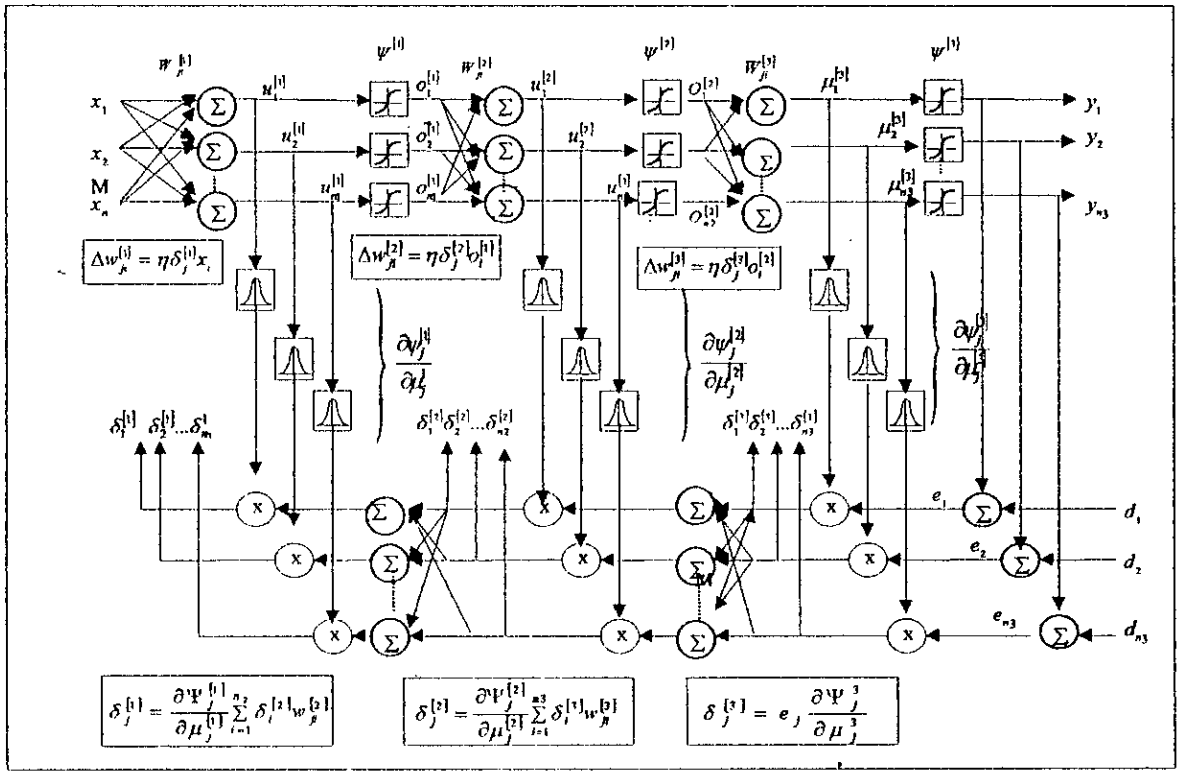


Figure I.5. Représentation fonctionnelle de l'algorithme de la rétropropagation du gradient

L'algorithme de rétropropagation du gradient est utilisé dans le perceptron multicouche. La phase importante ou ce dernier est utilisé est bien évidemment l'apprentissage : On présente au réseau des entrées et on lui demande de modifier sa pondération de telle sorte que l'on retrouve la sortie correspondante. L'algorithme consiste dans un premier temps à propager vers l'avant les entrées jusqu'à obtenir une sortie calculée par le réseau. La seconde étape compare la sortie calculée à la sortie désirée. On modifie alors les poids de telle sorte qu'à la prochaine itération, l'erreur commise entre la sortie calculée et connue soit minimisée.

Malgré tout, il ne faut pas oublier que l'on a une couche cachée. On rétropropage alors l'erreur commise vers l'arrière de la couche de sortie jusqu'à la couche d'entrée tout en modifiant la pondération. On répète ce processus sur tous les exemples jusqu'à temps que l'on obtienne une erreur de sortie considérée comme négligeable.

L'algorithme de la rétropropagation standard, utilise la méthode de plus forte descente pour la minimisation de la moyenne carrée de l'erreur locale, qui est donné par :

$$E_p = \frac{1}{2}\sum_{j=1}^{n_3}(d_{jp} - y_{jp})^2 = \frac{1}{2}\sum_{j=1}^{n_3} e_{jp}^2 \tag{1.11}$$

Et de l'erreur globale donnée par :

$$E = \sum_p E_p = \frac{1}{2}\sum_p \sum_j (d_{jp} - y_{jp})^2 \tag{1.12}$$

Où, d_{jp} et y_{jp} représentent la sortie désirée et la sortie actuelle du *j-ième* neurone de sortie pour le *p-ième* exemple.

Le problème de l'apprentissage peut être formulé comme suit : étant donné l'état actuel des poids synaptiques, on doit déterminer l'augmentation ou la diminution $\Delta W_{ji}^{[s]}$ (s étant le numéro de la couche : s=1,2,3) des poids afin de minimiser l'erreur globale, E.

La variation $\Delta W_{ji}^{[s]}$ s'écrit :

$$\Delta W_{ji}^{[s]} = -\eta \frac{\partial E_p}{\partial W_{ji}^{[s]}} \quad \eta > 0 \tag{1.13}$$

η est le coefficient d'apprentissage

Il est montré que si le paramètre η est suffisamment faible, cette procédure minimise l'erreur globale, $E = \sum_p E_p$

Pour la couche de sotie (s=3)

$$\Delta W_{ji}^{[3]} = -\eta \frac{\partial E_p}{\partial W_{ji}^{[3]}} = -\eta \frac{\partial E_p}{\partial \mu_j^{[3]}} \cdot \frac{\partial \mu_j^{[3]}}{\partial W_{ji}^{[3]}} \tag{1.14}$$

avec $\mu_j^{[3]} = \sum_{i=1}^{n_2} W_{ji}^{[3]} x_i^{[3]} = \sum_{i=1}^{n_2} W_{ji}^{[3]} O_i^{[2]}$ (j=1,…,n_3) (1.15)

On définit l'erreur locale à la sortie appelée delta par :

$$\delta_j^{[3]} = -\frac{\partial E_p}{\partial \mu_j^{[3]}} = -\frac{\partial E_p}{\partial e_j} \cdot \frac{\partial e_j}{\partial \mu_j^{[3]}} = e_{jr} \cdot \frac{\partial \psi_j^{[3]}}{\partial \mu_j^{[3]}} \tag{1.16}$$

où $\psi_j^{[3]}$ représente la fonction d'activation.

On obtient la formule pour la mise à jour des poids à la couche de sortie

$$\Delta W_{ji}^{[3]} = \eta \delta_j^{[3]} x_i^{[3]} = \eta \delta_j^{[3]} O_i^{[2]} \tag{1.17}$$

$$\text{Où,} \quad \delta_j^{[3]} = e_{jp}(\psi_j^{[3]})' = (d_{jp} - y_p)\frac{\partial \psi_j^{[3]}}{\partial \mu_j^{[3]}} \tag{1.18}$$

Pour la mise à jour des poids synaptiques de la seconde couche :

$$\begin{aligned} \Delta W_{ji}^{[2]} &= -\eta \frac{\partial E_p}{\partial W_j^{[2]}} = -\eta \frac{\partial E_p}{\partial \mu_j^{[2]}} \cdot \frac{\partial \mu_j^{[2]}}{\partial W_{ji}^{[2]}} \\ &= \eta \delta_j^{[2]} x_i^{[2]} = \eta \delta_j^{[2]} O_i^{[1]} \end{aligned} \tag{1.19}$$

Où l'erreur locale pour la couche cachée est définie par :

$$\begin{aligned} \delta_j^{[2]} &= -\frac{\partial E_p}{\partial \mu_j^{[2]}} \\ &= -\frac{\partial E_p}{\partial O_j^{[2]}} \cdot \frac{\partial O_j^{[2]}}{\partial \mu_j^{[2]}} \end{aligned} \tag{1.20}$$

$$\text{Avec} \quad \mu_j^{[2]} = \sum_{i=1}^{n_1} W_{ji}^{[2]} x_i^{[2]} = \sum_{i=1}^{n_1} W_{ji}^{[2]} O_i^{[1]} \qquad (j=1, \ldots n_2) \tag{1.21}$$

Sachant que

$$O_j^{[2]} = \psi_j^{[2]}(\mu_j^{[2]}) \tag{1.22}$$

On obtient

$$\delta_j^{[2]} = -\frac{\partial E_p}{\partial O_j^{[2]}} \cdot \frac{\partial \psi^{[2]}}{\partial \mu_j^{[2]}} \tag{1.23}$$

Le facteur $-\partial E_p / \partial O_j^{[2]}$ peut s'écrire :

$$\begin{aligned}\frac{-\partial E_p}{\partial O_i^{[2]}} &= -\sum_{i=1}^{n_3}\frac{\partial E_p}{\partial \mu_j^{[3]}}\cdot\frac{\partial \mu_j^{[3]}}{\partial O_i^{[2]}} \\ &= \sum_{i=1}^{n_3}(-\frac{\partial E_p}{\partial \mu_j^{[3]}})\frac{\partial}{\partial O_j^{[2]}}\left[\sum_{i=1}^{n_3} W_{ji}^{[3]} x_i^{[3]}\right] \\ &= \sum_{i=1}^{n_3}\delta_i^{[3]}\frac{\partial}{\partial O_i^{[2]}}\left[\sum_{i=1}^{n_3} W_{ji}^{[3]} O_i^{[2]}\right] \\ &= \sum_{i=1}^{n_3}\delta_i^{[3]} W_{ji}^{[3]}\end{aligned} \tag{1.24}$$

L'erreur locale dans la couche cachée peut être évaluée en utilisant la formule :

$$\delta_j^{[2]} = \frac{\partial \psi_j^{[2]}}{\partial \mu_j^{[2]}}\sum_{i=1}^{n_3}\delta_i^{[3]} W_{ji}^{[3]} \tag{1.25}$$

Par analogie, la mise à jour des poids synaptiques de la couche d'entrée s'écrit comme suit :

$$\Delta W_{ji}^{[1]} = \eta\delta_j^{[1]} x_i^{[1]} = \eta\delta_j^{[1]} O_i^{[0]} = \eta\delta_j^{[1]} x_i \tag{1.26}$$

où l'erreur locale est déterminée par :

$$\delta_j^{[1]} = \frac{\partial \psi^{[1]}}{\partial \mu_j^{[1]}}\sum_{i=1}^{n_2}\delta_j^{[2]} W_{ij}^{[2]} \tag{1.27}$$

$$\mu_j^{[1]} = \sum_{i=1}^{n_0} W_{ji}^{[1]} x_i \qquad (j=1,\ldots n_1) \tag{1.28}$$

Les étapes d'apprentissage de l'algorithme de la rétropropagation du gradient sont comme suit:

1. Initialiser les poids synaptiques $W_{ji}^{[s]}$ à des valeurs aléatoires (généralement entre -0.5/fan_in et +0.5/fan_in, où le fan_in représente le nombre de neurones contenus dans la couche directement inférieure).

2. Présenter un vecteur d'entrée et la sortie correspondante (désirée) et calculer les sorties actuelles de tous les neurones en utilisant les valeurs présentes $W_{ji}^{[s]}$ dans les équations (1.28), (1.21) et (1.15).

3. Spécifier la sortie désirée et évaluer les erreurs locales $\delta_j^{[s]}$ pour toutes les couches en utilisant les équations (1.18), (1.25) et (1.27).

4. Ajuster les poids synaptiques en accord avec la formule itérative $\Delta W_{ji}^{[s]} = \eta\delta_j^{[s]} x_i^{[s]}$ (s=3,2,1) correspondant aux équations (1.17), (1.19) et (1.26).

5. Calculer l'erreur E_p en utilisant l'équation (1.11).

6. Répéter les étapes 2 à 5 avec le même vecteur d'entrée jusqu'à ce que l'erreur E_p soit très proche de l'erreur admissible.

7. Répéter 2 à 6 pour chaque vecteur d'entrée X (pour chaque exemple d'apprentissage).

Tous les exemples d'apprentissage sont présentés périodiquement jusqu'à ce que les poids synaptiques soient stabilisés, i.e. jusqu'à ce que l'erreur pour tout l'ensemble complet soit acceptable et le réseau converge.

En pratique, une grande valeur du coefficient d'apprentissage η est préférable car elle entraîne une convergence rapide. Malheureusement, une grande valeur de η peut aboutir à une oscillation ou même à une divergence. Pour surmonter ce problème, un terme momentum est introduit dans les équations de mise à jour des poids synaptiques, telle que:

$$W_{ji}^{[s]}(t+1) = W_{ji}^{[s]}(t) + \Delta W_{ji}^{s}(t)$$
$$\Delta W_{ji}^{[s]}(t) = \eta \delta_j^{[s]} O_i^{[s-1]} + \alpha \Delta W_{ji}^{[s]}(t-1)$$
$$Avec \quad \eta > 0, \quad 0 \le \alpha \le 1$$
$$s = 1,2,3$$

I.2.4 Propriété des réseaux de neurones

L'intérêt principal porté aux réseaux de neurones tient sa justification dans les propriétés suivantes :

- **La capacité d'apprentissage**

La capacité d'apprentissage se traduit par la capacité du réseau de neurones à apprendre à résoudre des problèmes à partir d'exemples de façon similaire à l'être humain ou l'animal.

- **La capacité de généralisation**

La capacité de généralisation se traduit par la capacité d'un système à apprendre et à retrouver, à partir d'un ensemble d'exemples des règles qui permettent de résoudre un problème donné non appris.

- **Le parallélisme**

Cette notion se situe à la base de l'architecture des réseaux de neurones considérés comme un ensemble d'entités élémentaires qui travaillent simultanément. Le parallélisme permet une rapidité de calcul supérieure mais exige de penser et de poser les problèmes différemment .

I.3 Implémentation des réseaux de neurones

Selon les performances requises de l'application, les réseaux de neurones peuvent être implémentés en software sur un ordinateur conventionnel, à l'aide de processeurs DSPs "Digital signal Processing" programmables (circuits à application générale), en VLSI "Very Large Scale Integration " sur du silicium (circuits dédiés en full custom, cellules standards, FPGA), en optique ou en combinant ces techniques.

La Figure I.6 montre la vitesse de traitement (en nombre de connections/seconde) en fonction du nombre de synapses (poids synaptiques), obtenue pour les différents matériels utilisés pour l'implémentation des réseaux de neurones. Il est clair qu'à l'heure actuelle, toutes les technologies disponibles sont limitées par rapport à ce qui est recherché dans les réseaux de neurones [15].

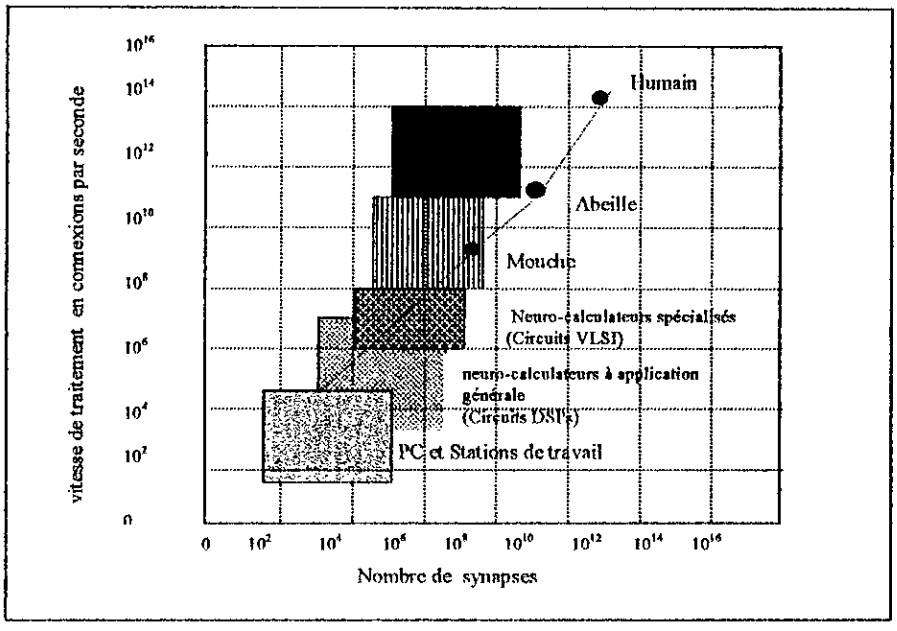


Figure I.6. Comparaison des différents matériels utilisés : vitesse de traitement en fonction du nombre de synapse.

Si l'on considère l'ensemble des implémentations possibles, les paramètres essentiels qui doivent être considérés sont le parallélisme, les performances (nombre de neurones et complexité des connexions), la flexibilité et leur relation à la surface du silicium. La Figure I.7 montre les performances obtenues en fonction de la flexibilité pour les différentes implémentations utilisées. Comme le montre la Figure I.7, ces performances sont souvent obtenues au prix d'une perte dans la flexibilité. Une solution optimale, si elle existe, est de réaliser un compromis entre ces deux paramètres [16] !

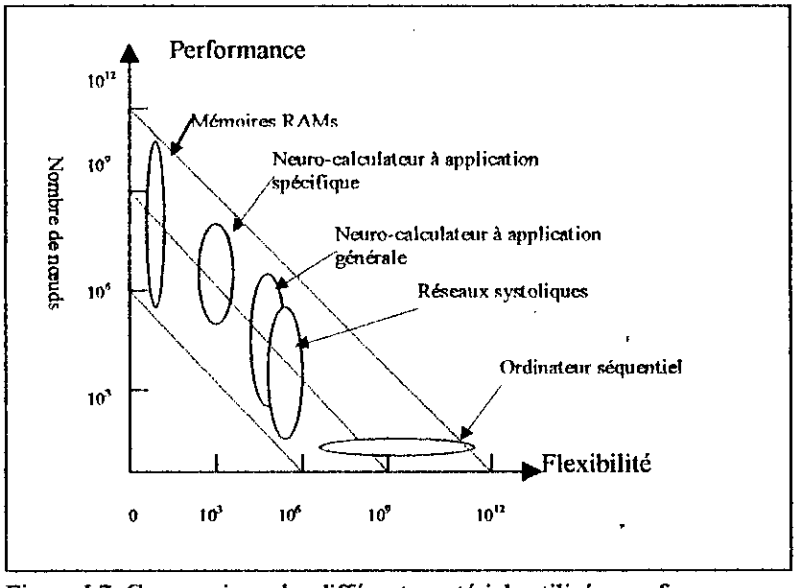


Figure I.7. Comparaison des différents matériels utilisés : performance en fonction de la flexibilité

I.3.1 Implémentation software

L'implémentation software considère les deux phases : apprentissage et généralisation de l'algorithme. Jusqu'à présent le guide principal pour l'élaboration de ces algorithmes a été l'utilisation du modèle simple du neurone et des synapses comme ceux définis par les équations mathématiques (1.15)-(1.28).
Par conséquent, la réalisation de tels algorithmes peut être formulée par une suite consécutive de multiplications matrice-vecteur, une suite consécutive de mise à jour de produits externes et une suite consécutive de multiplication vecteur-matrice. En terme fonctionnel, ces formulations font appel au processeur MAC (multiplieur- accumulateur). Les ordinateurs conventionnels sont basés sur le model de la machine série de Von Newman dans lequel une seule instruction est exécutée à la fois (1 seul MAC). Les simulateurs de réseaux de neurones existants à l'heure actuelle (exp. MATLAB), offrent à l'utilisateur des bibliothèques et un environnement flexible lui permettant de s'adapter à son application. Néanmoins, l'inconvénient majeur de ces derniers est que le parallélisme spatio-temporel caractéristique aux réseaux de neurones n'est pas exploité, sauf à moindre degré dans le cas de processeur RISC [16].
Pour palier à ce problème, des implémentations hardware parallèles sont utilisées.

I.3.2 Implémentation hardware

On distingue deux grandes approches d'implémentations hardware parallèles des réseaux de neurones [17] :

I.3.2.1 Les neuro- calculateurs à application générale

Ce genre d'architecture est basé sur l'utilisation de calculateurs hôtes couplés à des cartes accélératrices agissant en co-processeurs ou processeurs de calcul arithmétique spécialisés. (Exp. , Transputer INMOS, Texas, MS320, Processeur Zisc d'IBM).

La Figure I.8 montre une architecture typique d'un neuro-calculateur à application générale.

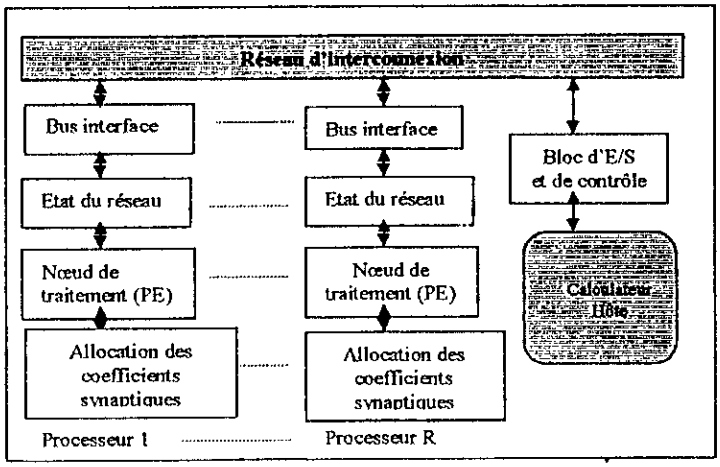


Figure I.8. Structure d'un neuro-calculateur à application générale

La structure comprend R processeurs identiques et programmables, connectés à travers un réseau d'interconnexion. En plus de l'accélération apportée, la nature programmable de ces circuits permet de couvrir un grand nombre de modèles de réseaux de neurones d'où une

flexibilité élevée (voir Figure I.8). Néanmoins ces circuits sont limités dans la taille et le parallélisme : en effet les circuits DSPs sont assez chers et des systèmes comportants quelques dizaines de processeurs sont irréalisables du point de vue coût et taille physique. De plus, plus le nombre de processeurs augmente plus le délai de communication augmente. Pour remédier à ces problèmes on utilise les implémentations VLSI ou neuro-calculateurs à application spécifique

I.3.2.2 Implémentation VLSI ou neuro- calculateurs à application spécifique

L'implémentation VLSI des réseaux de neurones est justifiée en premier lieu par la rapidité de traitement qui peut être atteinte en utilisant des architectures massivement parallèles vient ensuite la possibilité de réaliser des systèmes portables. On distingue deux grandes approches d'implémentation VLSI des réseaux de neurones [15] :

- Les implémentations qui intègrent la phase d'apprentissage et la phase de test/généralisation dans un même circuit d'où le terme anglais "***on chip training circuits***". Ce genre d'implémentation permet une flexibilité et une adaptabilité du circuit à plusieurs applications, néanmoins ces circuits ne sont pas très performants (complexité d'interconnexion et rapidité de traitement).
- Les implémentations qui intègrent seulement la phase de généralisation. Dans ce genre de circuits, l'apprentissage est réalisé en software permettant ainsi de générer les poids synaptiques. L'implémentation hardware du réseau de neurone consiste à charger ces poids synaptiques dans des mémoires d'où le terme anglais « ***off-chip training circuits***», et à implémenter les fonctions de sommation et d'activation. Ce genre d'implémentation est le plus utilisé ; et les circuits réalisés sont très performants néanmoins, ils ne peuvent s'adapter à d'autres applications.

L'implémentation VLSI des réseaux de neurones peut être analogique ou digitale. Les deux styles d'implémentation ont montré un très grand succès relativement à leur propre domaine d'application. Le choix d'un style par rapport à un autre dépend de plusieurs facteurs tel que la vitesse, la précision, la flexibilité, la programabilité, le transfert et la mémorisation de données [18].

L'implémentation analogique des réseaux de neurones est facile à réaliser néanmoins ces circuits sont : sensibles aux bruits, à la température, aux variations de la tension d'alimentation, moins précis et consomment une grande surface.

Comparés aux circuits analogiques, les circuits digitaux sont moins sensibles : au bruit, à la variation de la température et à la tension d'alimentation. De plus ces circuits sont plus précis et permettent d'intégrer des réseaux de grande taille.

Durant ces deux dernières années, les travaux de recherche visent l'implémentation des réseaux de neurones sur les circuits FPGAs (Field Programmable Gate Arrays) [19, 20 ,21]. En effet la nature reprogrammable des FPGAs a permis de réaliser des circuits très flexibles.

I.4 Application des réseaux de neurones

On peut déterminer quelques-unes des caractéristiques des problèmes bien adaptés à une résolution par les réseaux de neurones :

- Les règles qui permettent de résoudre le problème sont inconnues ou très difficiles à expliciter ou formaliser. Cependant, on dispose d'un ensemble d'exemples qui correspondent à des entrées du problème et aux solutions qui leur sont données par des experts.
- Le problème fait intervenir des données contaminées de bruits.
- Le problème peut évoluer, par exemple en faisant varier ses conditions initiales.
- Le problème nécessite une grande rapidité de traitement, il doit par exemple être traité en temps réel.

On peut donc dresser une liste des domaines d'applications privilégiés :

- Robotique
- Classification et reconnaissance de formes
- Approximation de fonctions
- Optimisation
- Traitement du signal.

I.5 Conclusion

Dans ce chapitre, nous avons présenté les réseaux de neurones de manière générale, en particulier le modèle du perceptron multicouche. Ensuite, nous avons présenté l'implémentation des réseaux de neurones. On s'aperçoit que lors de la conception des réseaux de neurones, les paramètres essentiels qu'il faut prendre en considération sont le parallélisme et la flexibilité. Les questions qu'on se pose :

- Le perceptron multicouche est-il approprié à la classification ?
- Comment dimensionner un réseau de neurones ?
- Comment concevoir une architecture FPGA parallèle et flexible ?

Les réponses se trouvent aux chapitres suivants.

Chapitre II

CLASSIFICATION AUTOMATIQUE DES ARYTHMIES CARDIAQUES

II.1 Introduction

Le problème de la classification automatique du signale cardiaque a suscité l'intérêt des chercheurs depuis la fin des années 1950. Le premier programme capable de faire la reconnaissance automatique de l'électrocardiographie (ECG) a été mis en œuvre en 1961. Ce programme et ses successeurs, basé sur les dérivations X-Y-Z de Frank ont été développés dans l'ancien hôpital de Washington par Pipberger et ses collaborateurs. Un programme similaire basé sur le système conventionnel à douze dérivations a été développé en 1962 par Carcers [22]. De nombreux travaux ont été réalisés par la suite. L'analyse et l'interprétation automatisé des tracés électriques du cœur sont devenus aujourd'hui, du moins aux USAs et au Japon, un acte de routine, et constituent une aide au diagnostic dont l'intérêt va croissant. Ces systèmes amènent de grands progrès quant à la fiabilité, l'automaticité ainsi que diverses facilités d'utilisation. Les tâches principales d'un système d'électrocardiographie automatisé sont :

- Acquisition des données, traitement préalable et conditionnement du signal.
- Le stockage des données pour une utilisation ultérieure.
- Le calcul des paramètres électriques du signal cardiaque.
- La recherche de l'interprétation : reconnaissance de la forme, classification et diagnostic en termes médicaux.

Actuellement, le succès commercial apporté par les systèmes d'électrocardiographie automatiques est beaucoup plus lié à leur possibilité d'extraction des paramètres et de stockage des données qu'à leur possibilité de classification. Toutefois, de part le monde des efforts sont consentis afin de mettre en œuvre des classificateurs fiables. Pour notre part, nous nous intéressons à la classification des arythmies cardiaques.
Nous commençons par présenter les principes généraux de la classification, par la suite nous étudions la classification des arythmies cardiaques en premier lieu sous un aspect clinique ensuite nous ferons une synthèse des méthodes de classification automatique de l'ECG.

II.2 Principes généraux de la classification

De manière générale, la classification consiste à attribuer une forme représentée par un vecteur à une classe parmi d'autres.
Le problème de la classification peut être formulé comme suit :

On cherche un classificateur **C**, qui à tout signal d'entrée **X**, associe une décision de classe **C(x)** :

$$\mathbf{X} \rightarrow \mathbf{C(x) = 1,2,..., k}$$ quand ***k classes*** sont possibles.

Plusieurs études ont été faites dans la littérature afin de rapprocher les approches classiques basées sur les méthodes statistiques et les nouvelles approches basées sur les réseaux de neurones et plus particulièrement les réseaux multicouches[23, 24, 25]. Dans ce qui suit nous reportons, quelques résultats essentiels.

La Figure II.1.a montre une représentation schématique d'un classificateur traditionnel et la Figure II.1.b celle d'un classificateur neuronal.

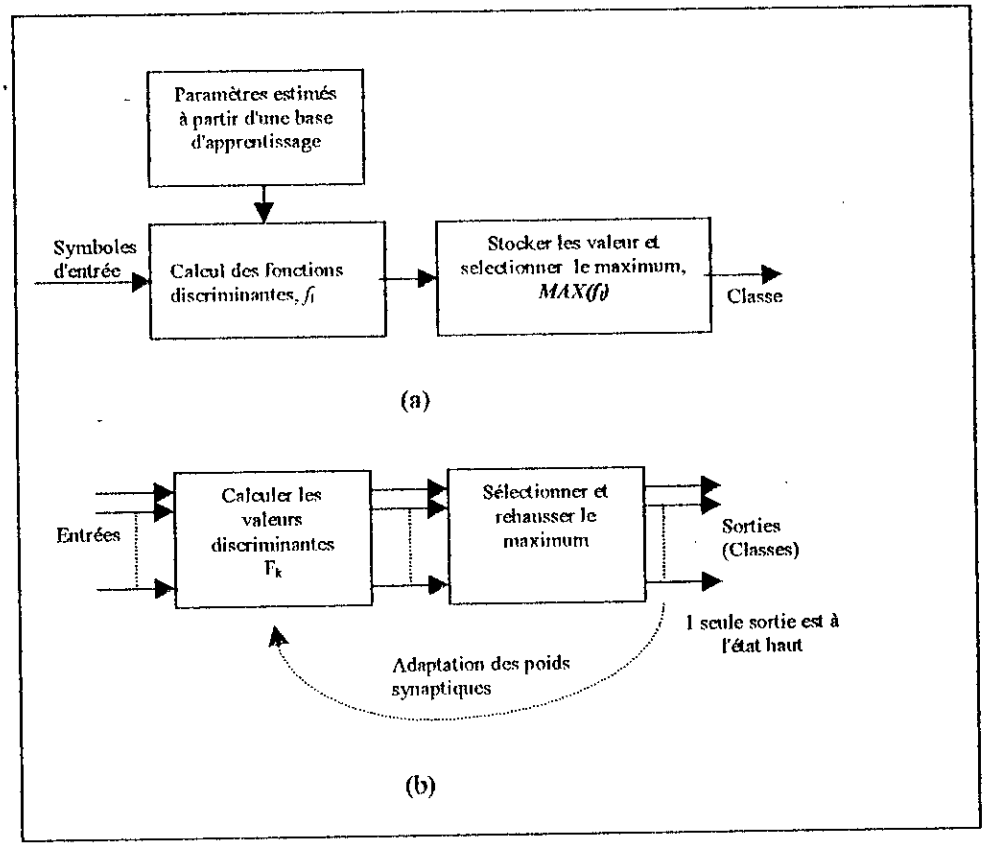


Figure II.1. (a) Classificateur traditionnel. (b) Classificateur à base de réseau de neurones

II.2.1 Classificateur traditionnel

Le classificateur traditionnel (Figure 1a) contient principalement deux blocs :

Le premier calcul les fonctions discriminantes pour chaque classe; celles ci sont données par:

$$fi(X) = Prob(classe\ i/X),$$

où X représente le vecteur d'entrée à classer, et l'indice ***i*** représente la classe correspondante.

Le deuxième bloc sélectionne le maximum de toutes les fonctions discriminantes ***fi(x),*** noté ***Max{fi(X)}.***
Ainsi la classe du vecteur X, *classe(X)*, est définie comme suit :

$$classe(X) = i/\ fi(X) = Max\{fi(X)\}$$

Pour estimer les fonctions discriminantes ou probabilités, ***fi(X),*** on dispose en général d'un ensemble de formes pour lesquelles la classe d'appartenance est connue (cet ensemble est appelé base d'apprentissage) et à partir duquel, on établit la configuration du classificateur (c'est à dire l'ensemble des f_i).
La probabilité de bonne reconnaissance des exemples d'apprentissage est appelée taux d'apprentissage, celle pour des formes quelconques est appelée taux de généralisation.

On distingue d'une façon générale deux grandes techniques de discrimination : les méthodes dites *non paramétriques* pour lesquelles les probabilités pour un vecteur X sont estimées par comptage des éléments de la base d'apprentissage.

Les méthodes dites *paramétriques* ou *bayésiennes*, où l'on se donne un modèle de la distribution de chaque classe (en général Gaussien) dont on estime les paramètres (moyenne, covariance) à partir de la base d'apprentissage et où l'on cherche la classe ayant la plus grande probabilité.

Comme indiqué dans la Figure II.1a, dans le classificateur traditionnel basé sur les modèles statistiques, les paramètres estimés à partir de la base d'apprentissage (moyenne, covariance) sont constants.

II.2.2 Classificateur neuronal

Le classificateur neuronal de la Figure II.1.b est composé de deux blocs : le premier réalise des fonctions de discrimination

$X \rightarrow F_k(X)$, où $F_k(X)$ est un vecteur à k composantes.

Celles ci sont transmises parallèlement au second bloc qui sélectionne et rehausse le maximum. Dans le cas du classificateur neuronal, on réalise un apprentissage qui consiste à modifier les poids synaptiques de telle sorte à approcher le mieux les probabilités $F_k(X)$, à partir de l'échantillonnage que constitue la base d'apprentissage. Pour se faire, on cherche à minimiser une énergie E exprimée par la distance moyenne entre valeur obtenue $F_k(X)$ et une valeur désirée, d(X). Après classification, seulement la sortie correspondant à la classe la plus appropriée est à l'état haut, les autres classes sont à l'état bas.

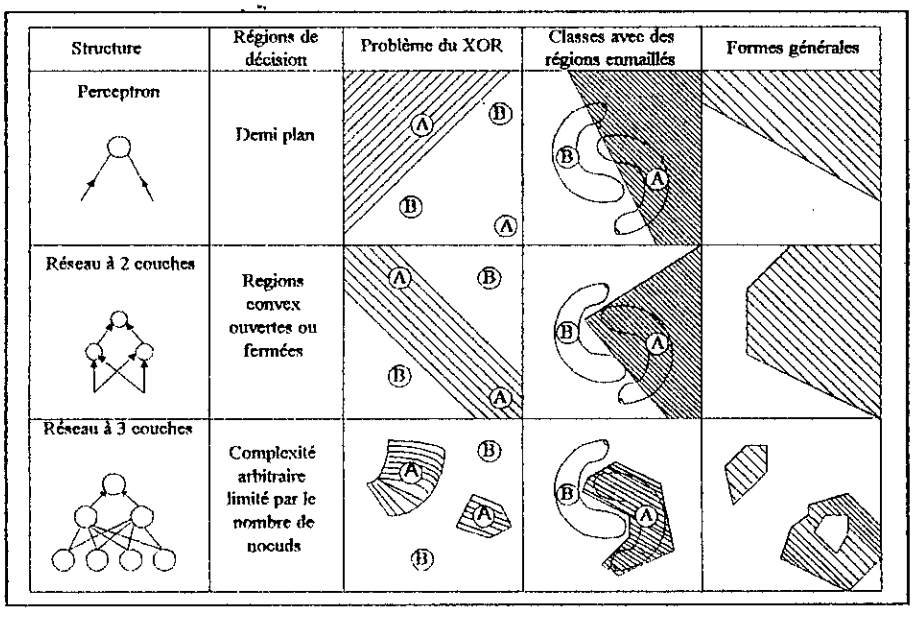


Figure II.2. Interprétation géométrique du rôle des couches caohées d'un réseau de neurones.

Les travaux de recherches montrent des analogies avec différentes méthodes statistiques d'analyse des données, en particulier l'analyse discriminante. Il est montré dans [25] que la rétropropagation du gradient (réseau multicouche) revient à faire une analyse discriminante

d'une population de N individus (N étant le nombre d'exemples pris en compte dans l'apprentissage) décrits par n paramètres (n étant le nombre de neurones d'entrée) et projetés dans un hyperplan de dimension p (p étant le nombre d'unités cachées).

La Figure II.2, donne une interprétation géométrique du rôle des couches cachées dans un réseau de neurones [23].
Dans son article[23], Lippmann montra que les régions de décision formées par un perceptron sont similaires à celles obtenues en utilisant un classificateur Gaussien. Néanmoins, l'algorithme du perceptron est facile à implémenter comparé au classificateur Gaussien et ne nécessite pas plus que de stocker l'information concernant les poids synaptiques.
Cependant, le model simple perceptron, et celui du classificateur Gaussien ne sont appropriés quand les classes ne peuvent être séparées par des hyperplans. Le problème du XOR est souvent reporté dans la littérature pour montrer les limites du perceptron. Par ailleurs, un réseau multicouche permet d'obtenir des régions de complexité arbitraire ; de plus le traitement des données s'effectue de manière massivement parallèle.

Nous utilisons ces résultats pour justifier notre choix de l'approche neuronle en particulier le perceptron multicouche pour la classification des arythmies cardiaques.

II.3 Classification de l'ECG et des arythmies cardiaques

II.3.1 Définition du signal cardiaque

L'activité cardiaque constitue l'un des plus importants paramètres déterminant l'état d'un malade. Elle se traduit par l'apparition de plusieurs ondes sur le tracé de l'électrocardiographe de surface (ECG) : c'est le signal cardiaque. La Figure II.3 montre la courbe caractéristique d'un signal cardiaque issue d'un ECG[26].

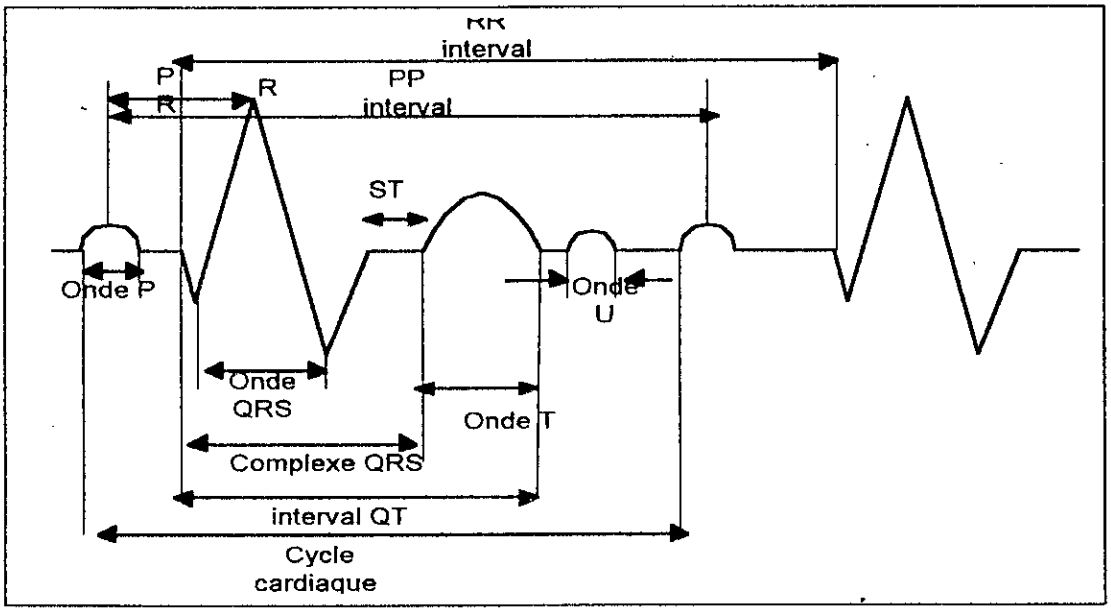


Figure II.3. Courbe caractéristique du signal cardiaque

On peut distinguer dans cet ECG différentes ondes (complexes), différents segments ainsi que différents intervalles. Ces composants réunis forment le cycle cardiaque complet, c'est ainsi que :

- **L'onde P** représente la déflexion produite par la dépolarisation des oreillettes. Dans les conditions normales la durée de l'onde P est inférieure ou égale à 0.11 secondes.
- **Le complexe QRS** représente l'ensemble des déflexions correspondant à la dépolarisation des ventricules. La valeur normale de l'onde QRS est inférieure à 0.1 secondes, habituellement entre 0.06 secondes et 0.08 secondes.
- **L'onde R** représente la première déflexion positive au cours de la dépolarisation ventriculaire.
- **L'onde T** représente la déflexion produite par la repolarisation ventriculaire.
- **L'onde U** est une déflexion habituellement positive que l'on rencontre après l'onde T et qui précède l'onde P suivante. Sa signification exacte est encore discutée.

Entre ces différentes déflexions s'inscrivent sur le tracé des intervalles, qui sont :

- **L'intervalle RR** représente la distance entre deux ondes R successives. L'intervalle RR permet de mesurer la fréquence ventriculaire ou fréquence cardiaque du cœur.
- **L'intervalle PP** représente la distance entre deux ondes P successives. L'intervalle PP permet de mesurer la fréquence auriculaire. Si le rythme cardiaque est régulier l'intervalle PP est identique à l'intervalle RR.
- **L'intervalle PR** correspond au temps de conduction auriculo-ventriculaire. L'intervalle PR se mesure du sommet de l'onde P jusqu'au sommet de l'onde QRS. Chez un sujet normal, la valeur de l'intervalle PR varie entre 0.12 secondes et 0.20 secondes.

II.3.2 Les arythmies cardiaques

Les troubles du rythme cardiaque ou arythmies sont dus essentiellement à des défaillances de certaines zones du cœur. Ces défaillances se manifestent par une déformation de l'électrocardiogramme (ECG) qui est un moyen classique de dépistage des différentes arythmies.

Traditionnellement, la méthodologie suivie par les médecins et cardiologues pour l'interprétation de l'ECG est basée sur l'observation, l'expérience et la mesure des données. La méthodologie en question consiste à :

- Localiser les ondes **P**, les complexes **QRS**, les complexes **T** et les ondes **U**
- Interpréter la morphologie (forme) des différentes ondes et complexes localisés
- Calculer les amplitudes et caractéristiques temporelles entre les différentes ondes et complexes tel que l'intervalle RR, l'intervalle PP, l'intervalle PR, le segment QT le segment ST.

Dans la terminologie médicale, un signal est dit normal si tous les paramètres cités ci-dessus sont normaux.
Dans le cas des troubles du rythme cardiaque et selon l'étage de leur origine : sinusal, auriculaire ou ventriculaire, on définit les classes suivantes [26, 27, 28] :
- Les troubles du rythme sinusal.
- Les troubles du rythme auriculaire.
- Les troubles du rythme ventriculaire.

- **Les troubles du rythme sinusal** se défissent par une onde P et un complexe QRS tous deux de morphologie normale, mais les intervalles PP et/ou RR présentent des valeurs différentes des valeurs du signal normal. Les troubles du rythme sinusal sont la tachycardie sinusale et la bradycardie sinusale. La Figure II.4 montre les tracés ECGs relativement à chacun des troubles du rythme sinusal citées ci dessus.

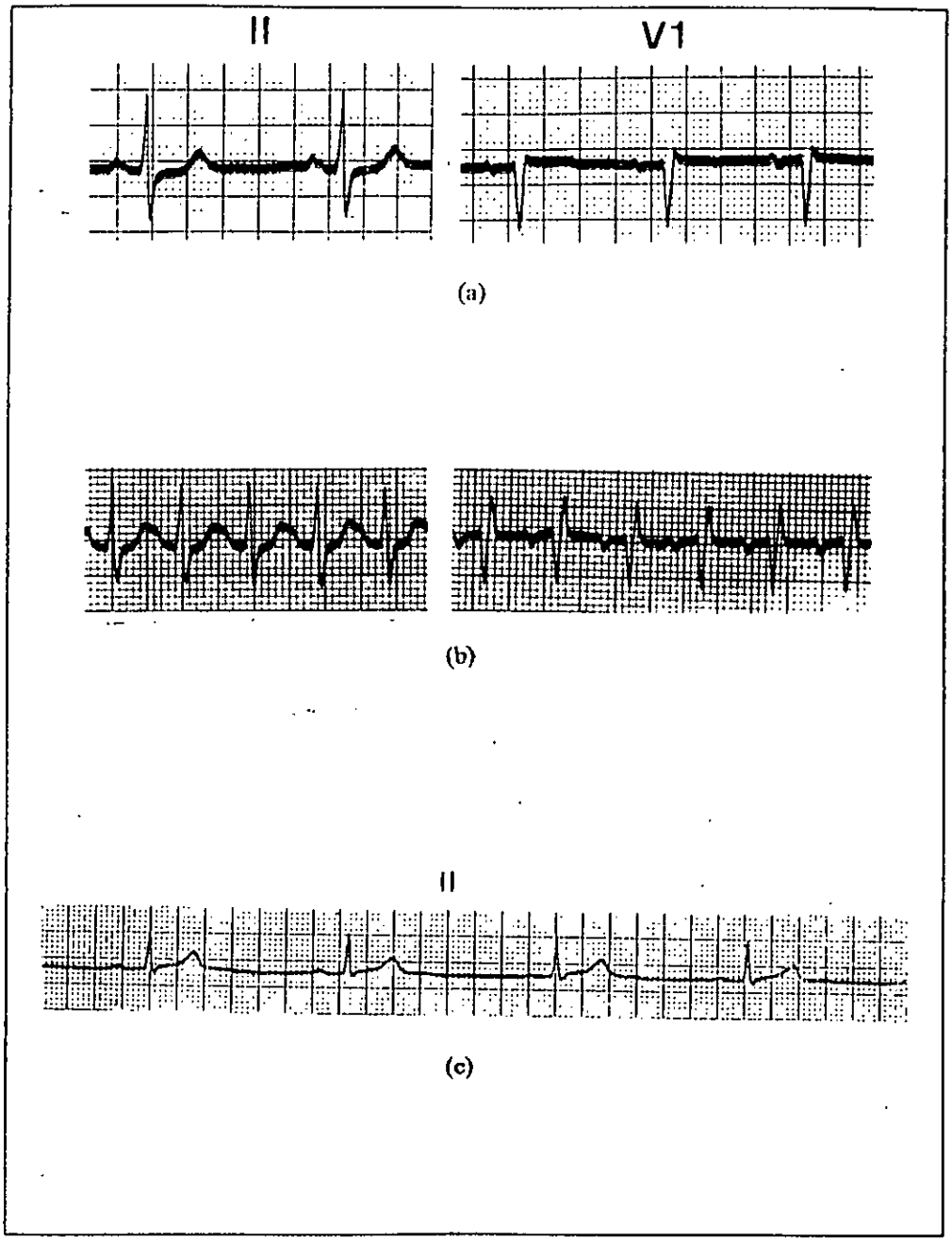


Figure II.4. Troubles du rythme sinusale. (a): signal normal, dérivation DIIet VI. (b): Tachycardie sinusale . (c): Bradycardie sinudale, dérivation DII.

- **Les troubles du rythme auriculaire** se définissent par une onde P de morphologie anormale et un complexe QRS de morphologie normale. En plus de la morphologie anormale de l'onde P, les caractéristiques temporelles des différents intervalles : PP, PR et RR permettent de distingue entre les extrasystoles auriculaires, la tachycardie auriculaire le flutter auriculaire et la fibrillation auriculaire. La Figure II.5 montre les tracés ECGs relativement à chacun des troubles du rythme auriculaire citées ci dessus.

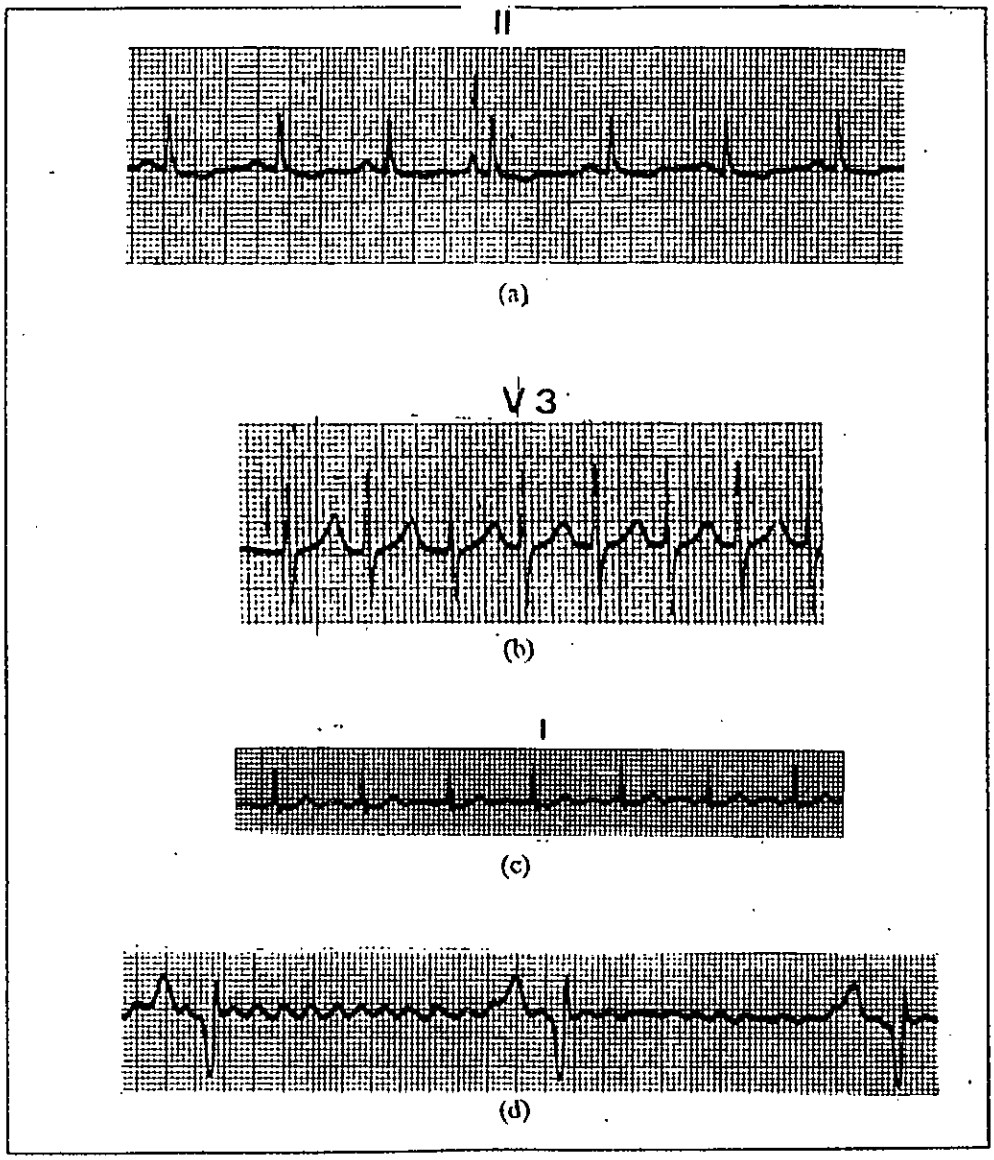


Figure II.5. Troubles du rythme auriculaire. (a): Extrasystoles auriculaires, dérivation DII. (b) : Tachycardie auriculaire, dérivation V3. (c) Flutter Auriculaire, dérivation V1. (d) Fibrillation auriculaire, dérivation V1.

- **Les troubles du rythme ventriculaire** sont définies par un complexe QRS de morphologie anormale et une onde P de morphologie normale. De plus, la connaissance des caractéristiques temporelles des différents intervalles PP, PR et RR permet de différencier entre les différentes anomalies affectant le rythme ventriculaire à savoir : l'extrasystole ventriculaire, la tachycardie ventriculaire, le flutter ventriculaire et la fibrillation ventriculaire. La Figure II.6 montre les tracés ECGs relativement à chacun des troubles du rythme ventriculaire cités ci dessus.
- Certaines anomalies affectent l'onde P et le complexe QRS en même temps, telle que la tachycardie supraventriculaire.

En résumé, la morphologie de l'onde P, du complexe QRS et les paramètres temporelles RR, PR et PP sont des paramètres important pour la classification.

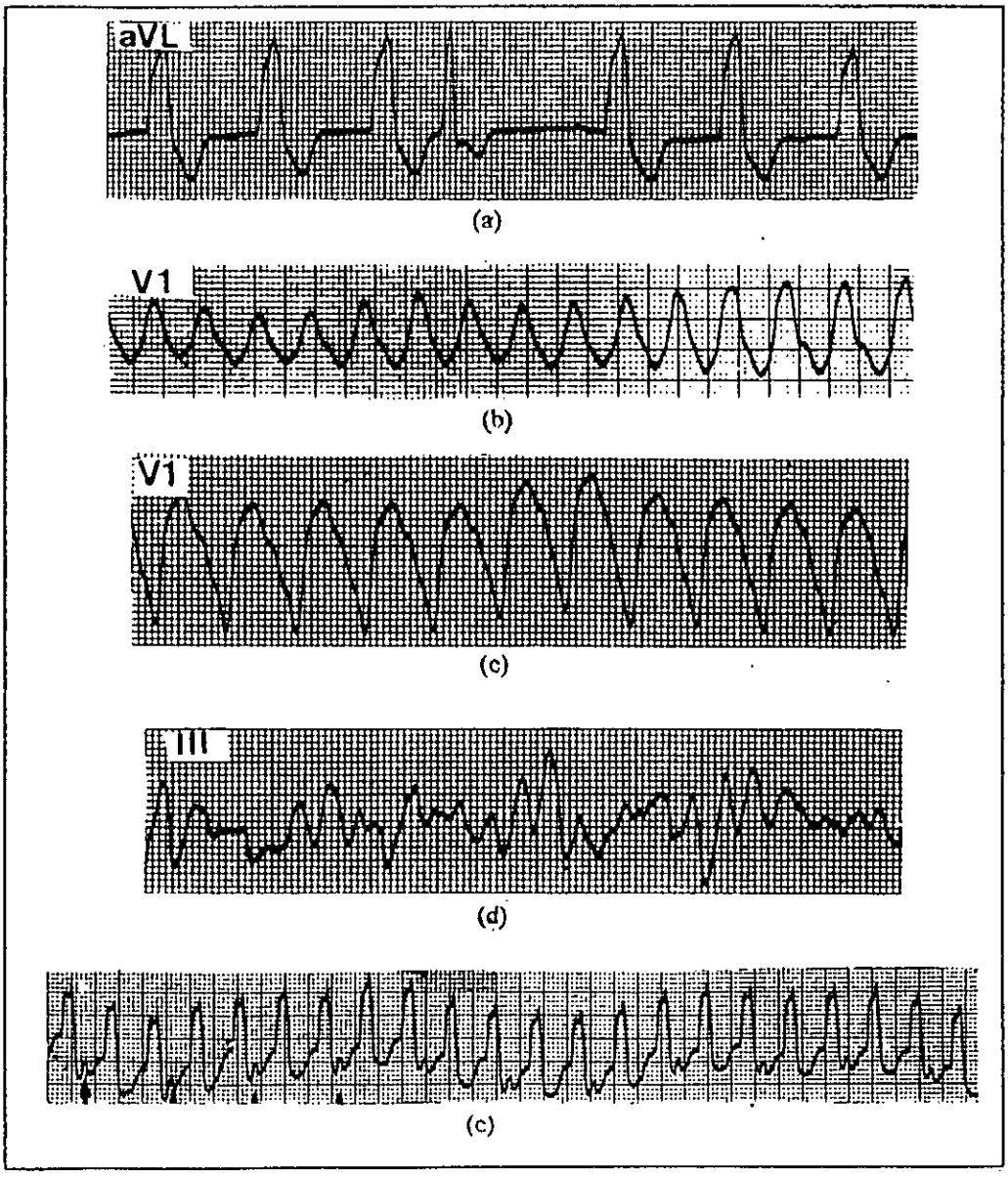


Figure II.6. troubles du rythme ventriculaire. (a); Extrasystoles ventriculaires , dérivation aVL. (b): Tachycardie ventriculaire, V1. (c) Flutter ventriculaire, dérivation DII. (d): Fibrillation ventriculaire, dérivation V1. (e): tachycardie supraventriculaire, dérivation V1.

II.3.3 Synthèse des méthodes de classification automatique de l'ECG

Comme nous l'avons souligné dans la section II.1, les travaux de recherches pour la classification du signal cardiaque remontent à 1957 à l'instigation de Piberger [22]. Les premières approches étaient principalement basées sur l'utilisation de la méthode des arbres et la méthode statistique. Pipberger classifia ces approches comme étant la première et seconde génération de programmes de reconnaissance automatique de l'ECG. Par la suite de nouvelles approches ont été développées telle que l'approche syntaxique, les systèmes experts, la logique floue et enfin les réseaux de neurones.

Nous avons résumé les principales approches dans la Figure II.7.

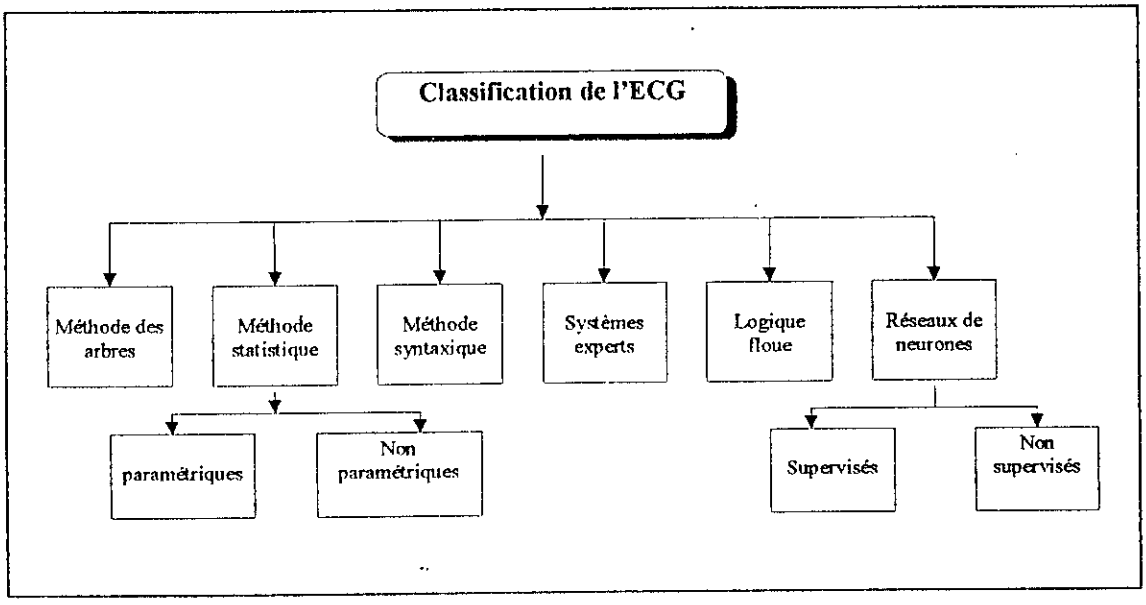


Figure II.7. Différentes approches utilisées pour la classification de l'ECG

II.3.3.1 Méthode des arbres [22]

La méthode des arbres est basée sur l'utilisation d'un arbre de décision comme la montre Figure II.8. Les entrées de ces programmes sont les caractéristiques temporelles et les amplitudes des différentes ondes et complexes permettant de distinguer un signal normal d'un signal pathologique. L'avantage de ces programmes est que les critères de diagnostic utilisés sont familiers au cardiologue et la logique est facile à suivre et à comprendre. Les critères de décision sont guidés par un raisonnement déductif basé sur des connaissances électrophysiologiques préalables. De plus ces programmes sont flexibles à toute modification. Cependant, l'un des inconvénients majeurs de l'imitation humaine est que les critères de diagnostic différent d'un médecin à un autre. Cette différence dans la préférence d'un critère par rapport à un autre a donné naissance à des systèmes commerciaux où l'utilisateur pouvait faire entrer ses propres préférences.
Dans une étude d'évaluation et de synthèse, Jos L.Willems [22] reporta que la précision de ces programmes ne dépassait pas 60%, vu que les caractéristiques temporelles ne suffisent pas à elles-mêmes pour classer les signaux cardiaques.

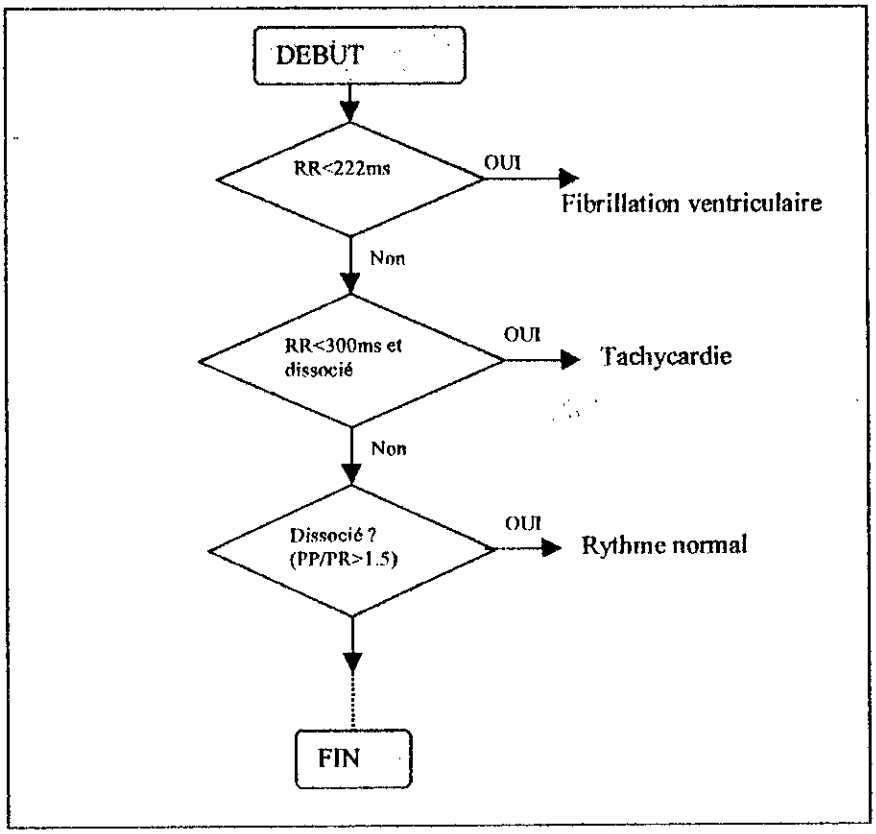


Figure II.8. Méthode des arbres

II.3.3.2 Approche statistique [22, 29, 30]

Dans l'approche statistique, le signal cardiaque subit un traitement mathématique préalable pour extraire les paramètres les plus représentatifs de ce dernier. La règle de Bayes est utilisée par la suite dans le processus de classification.
La décision bayésienne permet de reconnaître si un patient ayant un symptôme S est atteint d'une maladie, *Di* parmi *k* maladies D_k. La règle de Bayes s'exprime comme suit :

$$\Pr ob(D_i|S) = \frac{\Pr ob(D_i|S)\Pr ob(S)}{\sum_k \Pr ob(S|D_k)\Pr ob(D_k)}$$

Pour appliquer cette formule les maladies D_k, sont supposées indépendantes les unes des autres (classes séparables).

Les premiers travaux dans ce domaine ont été proposés par Pipberger et largement appliquées au signal vectocardiographique (VCG). Initialement, Pipberger réalisa un programme qui fait la classification de l'infarctus du myocarde et de l'hypertrophie ventriculaire par rapport au signal normal. En utilisant l'approche statistique, Pipberger montra que les performances de son système ont été améliorées de 20% jusqu'à 40% par rapport à la méthode des arbres.

Cependant, l'un des inconvénients majeurs de l'approche statistique est la nécessité d'une très large base de données pour tester les procédures de classifications. La collection d'une base de donnée nécessite un grand effort ; de plus les paramètres statistiques qui sont préalablement estimés lors de l'apprentissage, diffèrent considérablement d'une population à une autre et sont fonction de la race, du sexe, de la taille et de l'âge des patients.

La présence de plusieurs anomalies complique davantage les méthodes de classification statistiques : problème de séparation des classes.

II.3.3.3 Approche syntaxique[31, 32]

Si l'approche statistique offre un formalisme mathématique rigoureux, elle oublie la nature des formes des signaux. L'approche syntaxique, utilise des paramètres descriptifs liés à la nature même des formes du signal ECG. Elle suppose que l'ECG est composé par l'agencement structuré de formes élémentaires ou primitives qui sont les pics et les segments. C'est ainsi que, les pics sont combinés pour former des complexes et les pics et les segments sont combinés pour former le cycle cardiaque. Par la suite à chacune de ces primitives est attribué un symbole de l'alphabet $\sum = \{ K^+, K^-, E, \prod \}$ constituant le vocabulaire de l'ECG. Où K^+ représente le pic positif, K^- le pic négative, E une ligne droite et $\prod$ un contour.
L'approche syntaxique a été appliquée pour la reconnaissance de la forme de l'ECG, l'extraction et la mesure des paramètres et enfin pour la classification.

Dans une étude d'évaluation, Y. Suzuki [33] reporta que la sélection des primitives est un problème dépendant de la forme du signal et n'ayant pas une solution générale. Les primitives constituant l'ECG varient légèrement pour un même patient et d'un patient à un autre et dépendent des conditions du patient. Dans de tels cas la méthode syntaxique ne donne pas de bons résultats.

II.3.3.4 Les systèmes experts et la logique floue [34, 35]

Pour leur part, les systèmes experts et la logique floue ont été référencés dans la littérature. Cependant, d'après Jos L.Willems [22] l'utilisation clinique de ces approches a été très limitée.

II.3.3.5 Réseaux de neurones

Aucune des méthodes citées précédemment, n'a pu résoudre le problème de la classification du signal cardiaque. Nous pouvons résumer ces problèmes comme suit :

- La morphologie et les caractéristiques temporelles du signal cardiaque varient au cours du temps pour le même patient et entre différents patients.
- L'efficacité de ces techniques dépend du choix des paramètres sélectionnés pour la classification et des seuils utilisés.
- La nécessité de la connaissance des règles (dans le cas des systèmes experts, logique floue).
- Les modèles utilisés pour le traitement du signal ECG étaient des modèles linéaires.

Durant ces dernières années, les réseaux de neurones ont été intensivement utilisés pour la classification du signal cardiaque, vu les avantages qu'ils présentent à savoir :

- Habilité d'apprentissage et d'auto-organisation à partir d'exemples.
- Possibilité de lire des formes en réponse à de nouvelles formes (généralisation).
- Non nécessité de la connaissance de règles.
- Possibilité d'utiliser des modèles non linéaires.

Parmi les classificateurs d'arythmies cardiaques utilisant l'approche neuronale qui ont été développés, nous étudions deux classificateurs le premier est un classificateur neuronal hybride utilisant un apprentissage supervisé et le second est un classificateur neuronal à apprentissage non supervisé

II.3.3.5.1 Classificateur neuronal supervisé

Un système ICD "implantable cardioverter defibrillator" est proposé par P. Leong et al. dans [36] pour la classification trois types d'arythmies : la fibrillation ventriculaire, la tachycardie ventriculaire, la tachycardie supraventriculaire et du rythme sinusal normal. Un ICD délivre une décharge électrique au cours du traitement d'une arythmie. La plupart des ICDs actuels font la classification par la méthode des arbres et en se basant essentiellement sur la durée des différents intervalles RR, PR, PP...etc. Cependant, comme nous l'avons souligné précédemment, cette méthode n'est pas toujours fiable dans la classification des arythmies et par conséquent elle peut conduire dans certaines situations à une thérapie non appropriée.

Le classificateur proposé dans [36] est un classificateur hybride composé de deux sous classificateurs : un classificateur temporel basé sur la méthode des arbres et un classificateur neuronal de morphologie basé sur l'algorithme de la rétropropagation du gradient. La Figure II.9 montre le système de classification proposé. Le classificateur de morphologie a été conçu pour détecter la variation de morphologie dans la tachycardie ventriculaire alors que le classificateur temporel détecte les autres arythmies.

Le classificateur temporel a été implémenté en software alors que le classificateur neuronal de morphologie a été intégré sur un chip en utilisant une implémentation analogique.

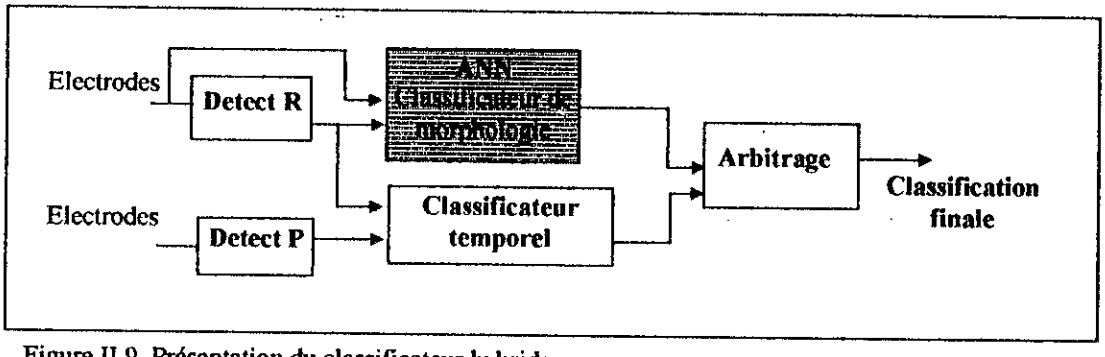


Figure II.9. Présentation du classificateur hybride

En utilisant cette approche, la moyenne de classification rapportée est de 90%.

Discussion: Dans ce système, la variation de la morphologie est prise en considération uniquement lorsqu'il s'agit de la tachycardie ventriculaire. En réalité, le problème ne se pose pas seulement pour cette arythmie mais aussi pour n'importe quel signal ECG.

II.3.3.5.2 Classificateur neuronal non supervisé

Un autre système ICD a été proposé par P. Voukydis dans [37] pour la classification des arythmies cardiaques. La Figure II.10 montre le système proposé. Il est composé de deux sous classificateurs neuronales montés en cascade. Le premier classificateur est un réseau de neurones à treize entrées et trois sorties. Les entrées du réseau décrivent l'amplitude et la durée du complexe QRS. La sortie du réseau permet de distinguer entre un signal de morphologie normal parmi quatre types de signaux anormaux (aberant#1, aberant#2, aberant#3). La sortie du premier classificateur en combinaison avec, la durée et la variance de l'intervalle RR sont appliquées à l'entrée du second classificateur pour identifier les arythmies suivantes : rythme sinusal normal, tachycardie sinusale, fibrillation auriculaire lente, fibrillation auriculaire rapide, complexes tachycardiques étroits, complexes tachycardiques larges, rythme rapide-étroit, et la fibrillation ventriculaire.
Les deux classificateurs sont basés sur l'apprentissage non supervisé en utilisant l'algorithme de Kohonen.

Discussion: Dans cette approche, l'auteur propose d'utiliser le réseau Kohenen car le temps d'apprentissage est inférieur au temps d'apprentissage de l'algorithme de la rétropropagation du gradient. La moyenne de classification est de 90%. D'après les résultats rapportés dans l'article, l'auteur a réussi à différencier entre les complexes QRS larges et les complexes QRS étroits. Cependant, et dans la plupart des cas, il n'a pas réussi à classer la fibrillation auriculaire. Nous constatons, que les caractéristiques utilisées par l'auteur décrivent seulement la morphologie du complexe QRS. Raison pour laquelle, la fibrillation auriculaire qui est cliniquement décrite par une anomalie de l'onde P et de la fréquence auriculaire (intervalle PP), n'a pas pu être détectée.

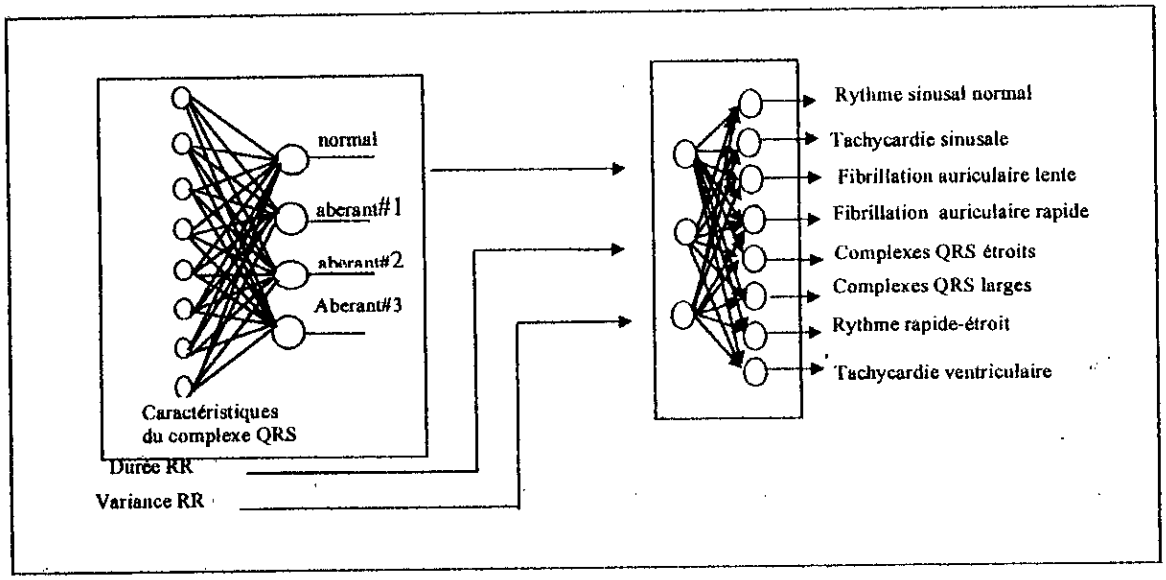


Figure II.10. Classificateur neuronal non-supervisé

II.4 Conclusion

Si nous faisons la synthèse des travaux énoncés dans ce chapitre, nous retenons les points essentiels suivants :

1. La classification des arythmies cardiaques doit prendre en charge la morphologie de l'onde P, la morphologie du complexe QRS et les caractéristiques temporelles du signal cardiaque à savoir l'intervalle PP, l'intervalle RR et l'intervalle PR.
2. Les réseaux de neurones sont les plus appropries pour la classification.
3. Un réseau multicouche et plus particulièrement le perceptron multicouche basé sur l'algorithme de la rétropropagation du gradient de l'erreur, permet d'obtenir des plans de séparation de complexité arbitraire.

Sur la base de ces constatations, nous proposons dans le prochain chapitre, un classificateur neuronal des arythmies cardiaques que nous appelons CNAC.

Chapitre III

CONCEPTION D'UN CLASSIFICATEUR NEURONAL DES ARYTHMIES CARDIAQUES(CNAC)

III.1 Proposition d'un classificateur neuronal

Nous avons montré au chapitre précédant que :

1. La classification des arythmies cardiaques doit prendre en charge la morphologie de l'onde P, la morphologie du complexe QRS et les caractéristiques temporelles du signal cardiaque à savoir l'intervalle PP, l'intervalle RR et l'intervalle PR.
2. Les réseaux de neurones sont les plus appropries pour la classification.
3. Un réseau multicouche et plus particulièrement le perceptron multicouche basé sur l'algorithme de la rétropropagation du gradient de l'erreur, permet d'obtenir des plans de séparation de complexité arbitraire.

Dans cette optique, nous proposons de réaliser un classificateur des arythmies suivantes [38],[39]: rythme sinusal normal (**SN**), tachycardie sinusale (**TS**), bradycardie sinusale (**BS**), extrasystoles auriculaires (**ESE**), tachycardie auriculaire (**TA**), flutter auriculaire (**FA**), fibrillation auriculaire (**FibA**), extrasystoles ventriculaire (**ESV**), tachycardie ventriculaire (**TV**), fibrillation ventriculaire (**FibV**), flutter ventriculaire (**FV**) et la tachycardie supraventriculaire (**TSV**).

Nous avons appelé le classificateur CNAC.

III.2 Structure du CNAC

La Figure III.1 montre la structure générale du **CNAC**.
Il est composé de deux classificateurs RN1 et RN2 montés en cascades. RN1 est un classificateur neuronal qui permet de classer les ondes P et QRS en morphologie normale et anormale. Les sorties de RN1 en combinaison avec les valeurs des caractéristiques temporelles RR, PR et PP sont appliquées à l'entrée du deuxième classificateur RN2 dont les sorties sont les différentes arythmies proposées précédemment.

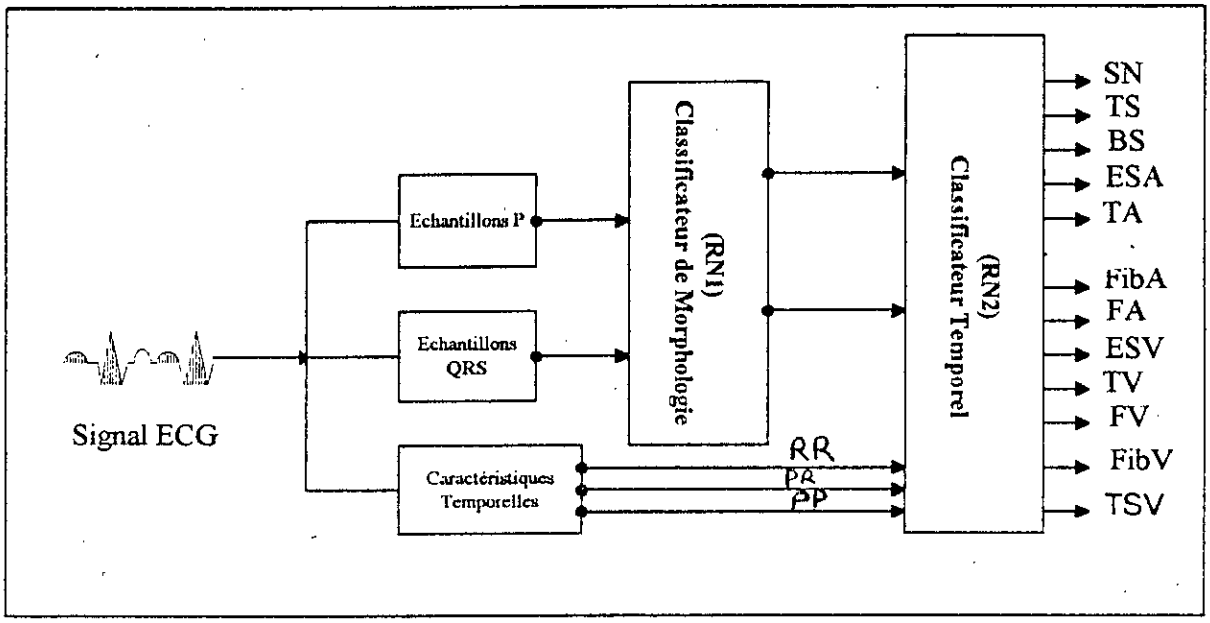


Figure III.1. Architecture du classificateur des arythmies cardiaques, CNAC.

III.2.1 Classificateur de Morphologie

Deux sous classificateurs ***RN1_P*** et ***RN1_QRS*** sont utilisés pour classer les ondes P et les complexes QRS séparément comme montré dans la Figure III.2. Chaque réseau est composé de Trois couches : une couche d'entrée reliée aux échantillons P ou QRS , une couche de sortie dont le nombre de neurones est déterminé par le nombre de classes désirées (dans notre cas deux classes pour chaque onde : normal/anormal), et une couche cachée dont le nombre de neurone est déterminée lors de l'apprentissage du réseau.

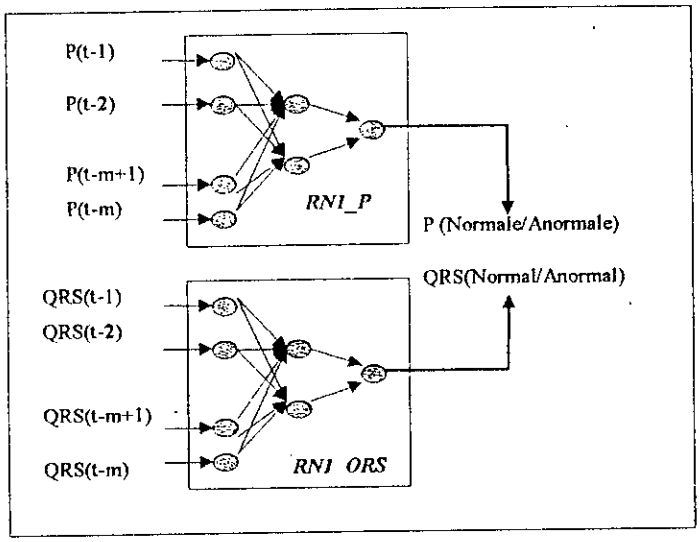


Figure III.2. Classificateur de morphologie

P(Normale) = 1 ; P(Anormale)=0 ; QRS(Normal)=1 ; QRS(Anormal)=0.

III.2.2 Classificateur temporel

L'entrée du classificateur temporel, ***RN2***, est déterminée par les valeurs de l'onde P et QRS issues du classificateur de morphologie et les valeurs des intervalles PP, PR et RR (Voir Figure III.3). La sortie de celui ci, correspond aux différentes arythmies à classer et qui sont au nombre de douze. La couche intermédiaire est déterminée lors de l'apprentissage.

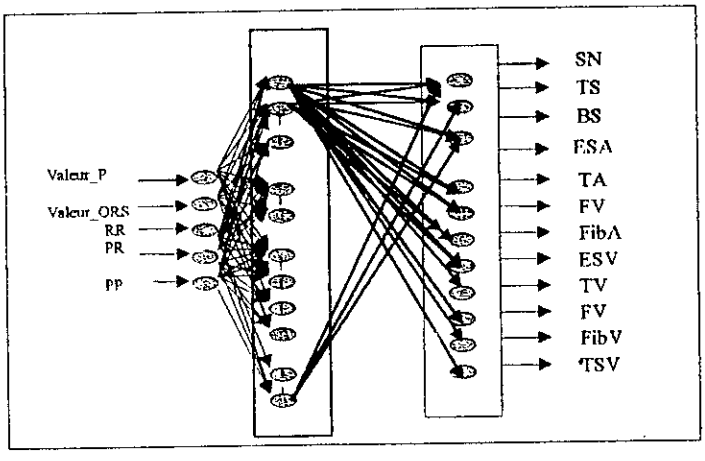


Figure III.3. Classificateur temporel

III.3 Implémentation software

Le but de l'implémentation software est de déterminer, pour chacun des trois classificateurs ***RN1_P***, ***RN1_QRS*** et ***RN2*** cités précédemment :

1. La taille du réseau de neurones,
2. Les paramètres du réseau à savoir, le coefficient d'apprentissage,η ainsi que le facteur momentum, α
3. La matrice des poids synaptiques,

Qui permettent d'atteindre les meilleurs performances pendant l'apprentissage et après le test du CNAC.

III.3.1 Définitions des performances d'un classificateur

Afin d'évaluer les performances de notre classificateur CNAC, nous avons utilisé des mesures de performances communes aux systèmes de diagnostic. En général, on définit les paramètres suivants [40]:

Valeur positive vraie (TP')

La valeur positive vraie ou "true positive" TP', est définie par le nombre des patients testés, et pour lesquels l'anomalie identifiée par le système de décision coïncide avec celle identifie par le médecin.

Valeur négative vraie (TN')

La valeur négative vraie ou "true negative" TN', est définie par le nombre de patients ne présentant pas une anomalie, et pour lesquels le système de décision affirme le même résultat.

Valeur positive fausse (FP')

La valeur positive fausse ou "false positive" FP', est définie par le nombre de patients présentant une anomalie (identifiée par le médecin) et pour lesquels les système, par erreur de classification n'identifie pas l'anomalie.

Valeur négative fausse (FN')

La valeur négative fausse ou "false negative" FN' est définie par le nombre de patients ne présentant pas une anomalie (affirmée par le médecin), et pour lesquels le système, par erreur de classification identifie une anomalie.

A partir de ces mesures on définit les pourcentages suivants :

$$\%CC = 100 \times (TP' + TN')/N$$
$$\%FP = 100 \times FP'/(TN' + FP')$$
$$\%FN = 100 \times FN'/(TP' + FN')$$

Où ***%CC***, ***%FP***, ***%FN*** représentent le pourcentage de classification correcte, le pourcentage de classification d'une anomalie et le pourcentage de classification des signaux normaux, respectivement et ***N*** est un nombre représentant l'ensemble des vecteurs de test.

De même, on définit la sensibilité ***SE***, la spécifité "specifity", ***SP***, et la précision du système ***PR***, comme suit :

$$SE = 100 \times TP'/(TP' + FN')$$
$$SP = 100 \times TN'/(TN' + FP')$$
$$PR = 100 \times TP'/(TP' + FP')$$

III.3.2 Construction de la base de données ECG

Vu la non-disponibilité d'une base de donnée ECG à notre niveau, nous avons scanné des images représentant les signaux ECGs en question. Par la suite, nous avons prélevé manuellement et à des intervalles réguliers des échantillons pour chaque signal représentant l'onde P et le complexe QRS ainsi que les différentes valeurs des caractéristiques temporelles PP, PR et RR relativement à chaque type d'arythmie.

Pour l'apprentissage et le test du classificateur de morphologie, la base de données et constituée de quinze (15) vecteurs représentant les ondes P de morphologie normale/anormale et de quinze (15) vecteurs représentant les complexes QRS de morphologie normale/anormale. Chaque vecteur est constitué de 10 échantillons.

Pour l'apprentissage et le test du classificateur temporel ***RN2***, nous avons construit une base de données de 48 vecteurs représentant les différentes arythmies et réparties de façon équitable, soit 4 vecteurs pour chaque arythmie.

III.3.3 Programmation

Notre choix étant porté sur l'algorithme de la rétropropagation du gradient, un programme est écrit en langage C pour l'implémentation software de cet algorithme. Nous utilisons les équations de la section I.2.3 du chapitre 1. La fonction d'activation est la fonction sigmoïde définie par :

$$\psi(x) = \frac{1}{1 + \exp(-x)}$$

Le principe de l'algorithme est le suivant :

- **Algorithme**

```
Define MAXI, MAXJ, MAXK   */ nombre des neurones de la couche d'entrée, la couche
                             cachée et la couche de sortie.
Define η, α,              */ Taux d'apprentissage et momentum
Define ε                  */ erreur
Define Max_n              */ nombre maximum d'itérations
Float X[MAXI]             */ Vecteur d'entrée
Float Yd[MAXK]            */ Vecteur de sortie désirée
Float Y_out[MAXK]         */Vecteur de sortie calculé
Float Wji[MAXJ][MAXI]     */ Matrice des poids synaptiques entre couche
                             cachée et l'entrée
Float Wkj[MAXK][MAXJ]     */Matrice des poids synaptiques entre couche
                             de sortie et couche cachée

Faire

  Pour chaque exemple d'apprentissage Faire
      Lecture des échantillons du signal d'entrée (P/QRS)
      Lecture de la sortie correspondante
      Initialisation des poids synaptiques
      Propagation de l'entrée vers l'avant
      Calcul de l'erreur
      Propagation de l'erreur vers l'arrière
      Ajustement des poids synaptiques
      Mise à jour de l'erreur
  Fin pour
Tant que l'erreur >ε et le nombre d'itération <Max_n
```

Dans une première étape, un programme est écrit pour l'apprentissage des réseaux ***RN1_P*** et ***RN1_QRS*** (ANNEXE 1).

L'entrée du programme est un fichier contenant :

1. Les différentes valeurs des échantillons représentant les signaux P et QRS (exemples d'entrée)
2. La sortie désirée.
3. Les valeurs initiales des poids synaptiques.

La sortie de chaque programme est un fichier contenant

1. La valeur de l'erreur calculée à chaque itération
2. L'information sur la convergence de l'algorithme :
   - Dans le cas où l'algorithme convergerait, le programme affiche l'erreur finale à la convergence et les valeurs finales des poids synaptiques du réseau en question.
   - Dans le cas contraire, le programme afficherait : "*non-convergence à l'itération Max_n*".

Un programme similaire est appliqué pour l'apprentissage du classificateur temporel, RN2.

Un deuxième programme est développé en vue d'effectuer le test/généralisation sur de nouveau exemples (ANNEXE 2).

Le pseudo-code du programme de test est :

- **. Algorithme**

```
Define MAXI, MAXJ, MAXK   */ nombre des neurones de la couche d'entrée, la couche
                             cachée et la couche de sortie.
Define η, α,              */ Taux d'apprentissage et momentum
Define  ε                 */ erreur
Define Max_n              */ nombre maximum d'itérations
Define seuil P/QRS        */ Seuil pour la classification
Float X[MAXI]             */ Vecteur d'entrée
        ⋮
Lecture d'un nouvel exemple
Lecture des poids synaptiques obtenus après convergence
Propagation de l'entrée vers l'avant
Calcul d'erreur
/******* Pour le classificateur RN1 ******/
  Si  Y_out> seuil P/QRS
   alors P/QRS morphologie normale
 Sinon
   P/QRS  morphologie anormale
/************************************/
/******Pour le classificateur RN2*******/
 Si  0<Y_out<seuil1 alors SN
    Seuil1<Y_out<seul2 alors TS
    Seuil2<Y_out<seuil3 alors BS
   Seuil3<Y_out<seuil4 alors ESA
              ⋮
  Seuil11<Y_out<seuil12  alors TSV
```

L'entrée du programme de test est un fichier contenant :
1. Les différentes valeurs des poids synaptiques obtenues après convergence.
2. Le signal à tester.

La sortie du programme de test est un fichier donnant l'information sur :
- La morphologie Normale/Anormale du signal représentant soit l'onde P soit l'onde QRS, dans le cas du classificateur de morphologie.
- Le type de l'arythmie, dans le cas du classificateur temporel.

III.3.4 Apprentissage, simulation et test

Nous discutons dans cette section, les conditions d'apprentissage, les résultats de simulations et de test des trois classificateurs *RN1_P*, *RN1_QRS* et *RN2*.

III.3.4.1 Classificateur de morphologie *RN1_P*

III.3.4.1.1 Apprentissage

Nous entraînons le réseau par les échantillons représentant les signaux des ondes P relativement à chaque type de dérivation standard. Pour une dérivation donnée, nous

sélectionnons une base d'apprentissage constituée de quatre ondes P de morphologie normale et quatre ondes P de morphologie anormale.
Les signaux représentant les ondes P de morphologie normale sont entraînés pour avoir une sortie de valeur 1, alors que ceux représentant les ondes P de morphologie anormale sont entraînées pour avoir une sortie de valeur nulle.

III.3.4.1.2 Dimensionnement du réseau

Il n'existe pas de résultat théorique, ni même de règle empirique satisfaisante, qui permette de dimensionner correctement un réseau de neurones en fonction du problème à résoudre.
La conception d'un réseau multicouche se fait de manière expérimentale, la difficulté se pose généralement au moment du choix du nombre de couches intermédiaires et du nombre de neurones dans chacune de celles-ci. Pour la couche d'entrée (respectivement de sortie), elle contient autant de neurones que d'entrées (respectivement de classes à discriminer).
De ce fait, la couche d'entrée du classificateur ***RN1_P*** est constituée de 10 neurones et la couche de sortie est constituée d'un seul neurone.
Pour déterminer le nombre de neurones de la couche cachée, nous avons procédé de la manière suivante :

1. Préparer une base d'échantillons normalisés pour l'apprentissage.
2. Fixer le nombre de neurones en entrée et en sortie.
3. Désigner un nombre de neurones arbitraire dans la couche cachée.
4. Fixer une erreur de très faible valeur.
5. Fixer le coefficient d'apprentissage et le facteur momentum α à des valeurs aléatoires situées dans l'intervalle [0, 1], ($\eta = 0.5$, $\alpha = 0.9$).
6. Lancer le programme d'apprentissage (ANNEXE 1).

Tant que le processus diverge, augmenter le nombre de neurones. Si celui ci est trop élevé, augmenter le nombre de couches intermédiaires. Fixer alors le nombre de couches et le nombre de neurones.

Après plusieurs simulations, nous avons obtenu la convergence pour paramètres suivants :

Couche d'entrée : 10 neurones
Couche cachée : 4 neurones
Couche de sortie : 1 neurone
Coefficient d'apprentissage, η : 0.5
Momentum, α : 0.9
Erreur atteinte, $\varepsilon = 10^{-2}$
Nombre d'itérations : 100

Après apprentissage, on détermine grâce au programme de test (ANNEXE 2) les valeurs réelles des seuils normal/anormal atteints par le classificateur. A partir de ces seuils on détermine les performances du classificateur.

La Figure III.4 montre les résultats de test du classificateur ***RN1_P.***

Onde P	Sortie calculée par *RN1_P*	Résultat du programme Test	Onde P	Sortie calculée par *RN1_P*	Résultat du programme Test
 P1 (N)	0.512	N	 P8(A)	0.134	A
 P2 (A)	0.462	A	 P9(N)	0.598	N
 P3 (N)	0.509	N	 P10(N)	0.521	N
 P4 (N)	0.54	N	 P11(A)	0.494	A
 P5(A)	0.723	N	 P12(N)	0.504	N
 P6(A)	0.498	A	 P13(N)	0.529	N
 P7(N)	0.620	N	 P14(N)	0.541	N
 P15(N	0.594	N	——	——	——

Figure III.4. Résultat de test du Classificateur ***RN1_P***

Où
N : signal normal
A : Signal anormal
Pour le test et après plusieurs essaies, nous avons fixé les seuils comme suit :
Seuil_P < 0.50 => signal (**A**)
Seuil_P ≥ 0.50 => signal (**N**)

A partir des résultats de classification, nous déterminons les valeurs TP', FP', TN' et FN' définis dans la section III.3.1 :

TP' = 4,
FP' = 1,
TN' = 10,
FN' = 0

A partir de ces valeurs nous déterminons les performances du classificateur

%CC = 93,
%FP = 9,
%FN =0,
%SE =100,
%SP = 90 et
%PR =80.

Où , **%CC** : pourcentage de classification correcte
%FP : pourcentage d'erreur dans le diagnostic d'une anomalie "false positif"
%FN : pourcentage d'erreur de classification de signaux normaux "false negatif"
%SE : sensibilité du classificateur
%SP : spécifité
%PR: précision du classificateur.

III.3.4.1.3 Etude de l'influence du coefficient d'apprentissage et du momentum sur la convergence du réseau *RN1_P*

Dans cette section nous étudions l'influence du coefficient d'apprentissage η et du momentum α sur la convergence du réseau ***RN1_P***. Pour cela, nous avons effectués les simulations suivantes :

Simulation 1 : α=0.9, η=0.1
Simulation 2 : α=0.9, η=0.5
Simulation 3 : α=0.9, η=0.9
Simulation 4 : α=0.5, η=0.5
Simulation 5 : α=0.1, η=0.5

Pour les trois premières simulations, le momentum α est maintenu constant à une valeur α=0.9 alors que le coefficient d'apprentissage est variable (η=0.1, η=0.5 et η=0.9).
La Figure III.5 montre l'influence du coefficient d'apprentissage sur la convergence de ***RN1_P***, pour les trois premières conditions de simulation citées ci dessus.

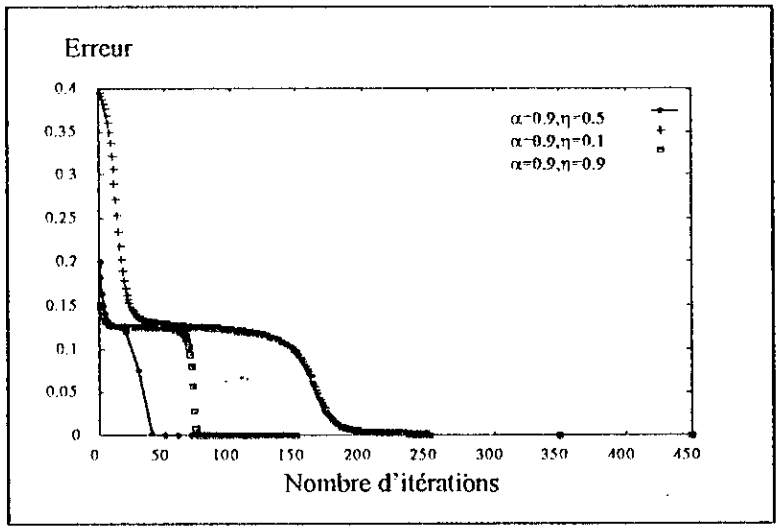


Figure III.5 Evolution de l'erreur en fonction du coefficient d'apprentissage, η. Le momentum α, fixé.

D'après la Figure III.5, la convergence est rapide pour η=0.5 et α=0.9.

Dans les deux dernières simulations, nous fixons le coefficient d'apprentissage η=0.5 puis nous faisons varier le momentum (α=0.9, α=0.5, α=0.1).
La Figure III.6, montre l'influence du momentum α sur la convergence du ***RN1_P***.

Pour les différentes simulations citées ci-dessus, nous remarquons que la convergence est meilleure pour α=0.5 et η=0.5.

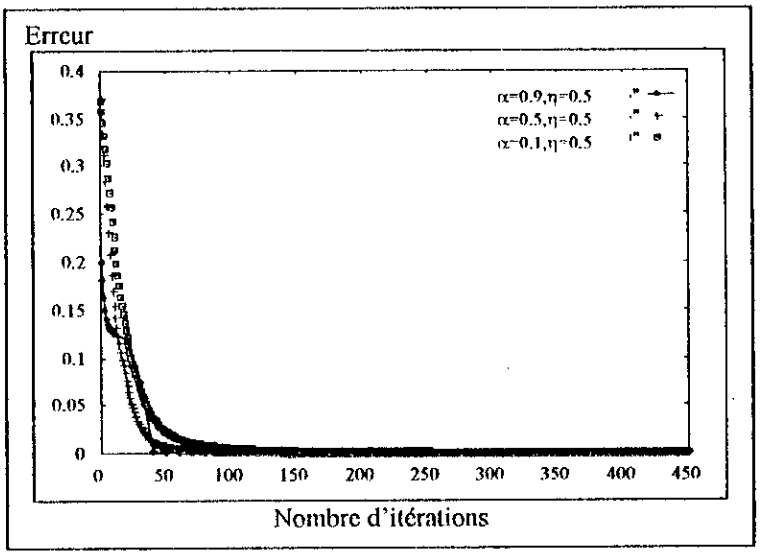


Figure III.6 Evolution de l'erreur en fonction du momentum α.
Le coefficient η étant fixe.

Néanmoins, le critère de convergence ne suffit pas à lui seul pour déterminer les meilleures valeurs paramètres α, η.
Afin de déterminer les meilleures valeurs des paramètres α et η, nous devons étudier l'influence de ces paramètres sur les performances du classificateur.

III.3.4.1.4 Etude de l'influence du coefficient d'apprentissage, et du momentum sur les performances du classificateur *RN1_P*

Dans cette section nous étudions l'influence du coefficient d'apprentissage, η, et du facteur momentum, α, sur les performances du classificateur ***RN1_P***.
Afin d'évaluer les performances du classificateur nous avons calculé les paramètres %CC, %FP, %FN, SE, SP et PR pour les mêmes simulations cités ci dessus, à savoir :

Simulation 1 : α=0.9, η=0.1
Simulation 2 : α=0.9, η=0.5
Simulation 3 : α=0.9, η=0.9
Simulation 4 : α=0.5, η=0.5
Simulation 5 : α=0.1, η=0.5

Le tableau III.1 présente les différentes mesures des performances %CC, %FP, %FN, %SE, %SP, et %PR.

Tableau III.1. Evaluation des performances de RN1_P en fonction de α , η

SIMULATION	α	η	%CC	%FP	%FN	%SE	%SP	%PR
1	0.9	0.1	73	11	25	50	88	75
2	0.9	0.5	93	9	0	100	90	80
3	0.9	0.9	86	16	25	100	83	60
4	0.5	0.5	86	9	25	75	90	75
5	0.1	0.5	86	9	25	75	90	75

D'après le tableau III.1 nous remarquons que les meilleures performances du classificateur ***RN1_P*** sont obtenues pour les valeurs α=0.9 et η=0.5 et pour lesquelles le classificateur possède les performances suivantes :

%CC=93
%FP= 9
%FN=0
%SE=100
%SP=90
%PR=80

Où , **%CC :** pourcentage de classification correcte
%FP : pourcentage d'erreur dans le diagnostic d'une anomalie "false positif"
%FN : pourcentage d'erreur de classification de signaux normaux "false negatif"
%SE : sensibilité du classificateur
%SP : spécifité
%PR: précision du classificateur

III.3.4.2 Classificateur de morphologie *RN1_QRS*

III.3.4.2.1 Apprentissage

Nous entraînons le réseau ***RN1_QRS*** par les échantillons représentant les signaux des complexes QRS relativement à chaque type de dérivation standard. Pour une dérivation donnée, nous sélectionnons une base d'apprentissage constituée de quatre complexes QRS de morphologie normale et quatre complexes QRS de morphologie anormale.

Les signaux représentant les complexes QRS de morphologie normale sont entraînés pour avoir une sortie de valeur 1, alors que ceux représentant les complexes QRS de morphologie anormale sont entraînés pour avoir une sortie de valeur nulle.

III.3.4.2.2 Dimensionnement du réseau

Nous appliquons les mêmes démarches présentées dans la section III.3.4.1.2, pour dimensionner ***RN1_QRS***.
La couche d'entrée du classificateur ***RN1_P*** est constituée de 10 neurones et la couche de sortie est constituée d'un seul neurone.
Après plusieurs simulations, nous avons obtenus la convergence pour les paramètres suivants :

Couche d'entrée : 10 neurones
Couche cachée : 4 neurones
Couche de sortie : 1 neurone
Coefficient d'apprentissage, η : 0.5
Momentum, α : 0.9
Erreur atteinte, $\varepsilon = 10^{-2}$
Nombre d'itérations : 114

Après apprentissage, on détermine grâce au programme de test (Annexe2) les valeurs réelles des seuils normal/anormal atteints par le classificateur. A partir de ces seuils on détermine les performances du classificateur.

La Figure III.7 montre les résultats de test du classificateur ***RN1_QRS, où***

N : signal normal
A : Signal anormal

Pour le test et après plusieurs essaies, nous avons fixé les seuils comme suit :
Seuil_QRS < 0.15 => signal (A)
Seuil_QRS≥0.15 =>signal (N):

A partir des résultats de test, nous déterminons les valeurs TP', FP', TN' et FN' définis dans la section III.3.1

$$TP' = 5,\ FP' = 0,\ TN' = 8,\ FN' = 2$$

A partir de ces valeurs nous déterminons les performances du classificateur

$$\%CC = 86,\ \%FP = 0,\ \%FN = 28,\ \%SE = 71,\ \%SP = 100 \quad \text{et } \%PR = 100.$$

Où , **%CC :** pourcentage de classification correcte
%FP : pourcentage d'erreur dans le diagnostic d'une anomalie "false positive"
%FN : pourcentage d'erreur de classification de signaux normaux "false negative"
%SE : sensibilité du classificateur
%SP : spécifité
%PR: précision du classificateur

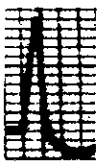
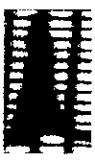
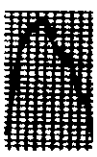
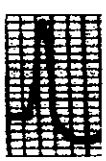
Complexe QRS	Sortie du RN1_QRS	Résultat du programme Test	Complexe QRS	Sortie du RN1_QRS	Résultat du programme Test
 QRS1(N)	0.323	N	 QRS9(N)	0.036	A
 QRS2(A)	0.148	A	 QRS10(N)	0.231	N
 QRS3(N)	0.322	N	 QRS11(A)	0.220	N
 QRS4(N)	0.160	A	 QRS12(N)	0.138	A
 QRS5(A)	0.136	A	 QRS13(N)	0.261	N
 QRS6(A)	0.109	A	 QRS14(N)	0.170	N
 QRS7(N)	0.278	N	 QRS15(A)	0.09	A
 QRS8(A)	0.102	A	——	——	——

Figure III.7 Résultat de test du classificateur RN1_QRS.

III.3.4.2.3 Etude de l'influence du coefficient d'apprentissage et du momentum sur la convergence du réseau *RN1_QRS*

Dans cette section nous étudions l'influence du coefficient d'apprentissage η et du momentum α sur la convergence du réseau ***RN1_QRS***. Pour cela, nous avons effectué les simulations suivantes :

Simulation 1 : α=0.9, η=0.1
Simulation 2 : α=0.9, η=0.5
Simulation 3 : α=0.9, η=0.9
Simulation 4 : α=0.5, η=0.5
Simulation 5 : α=0.1, η=0.5

Pour les trois premières simulations, le momentum α est maintenu constant à une valeur α=0.9 alors que le coefficient d'apprentissage est variable (η=0.1, η=0.5 et η=0.9).
La Figure III.8 montre l'influence du coefficient d'apprentissage sur la convergence de ***RN1_QRS***, pour les trois premières conditions de simulation citées ci dessus.

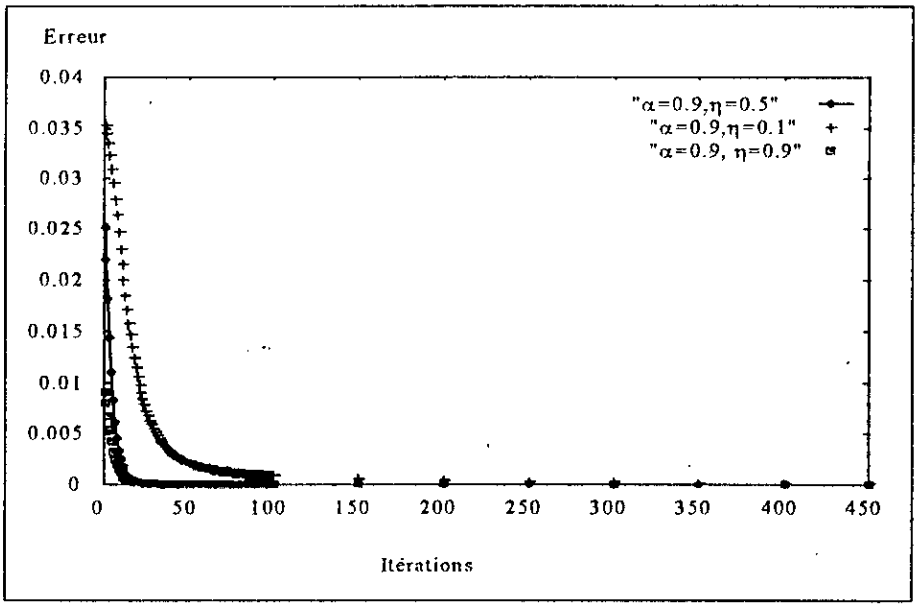


Figure III.8 Evolution de l'erreur en fonction du coefficient η, le momentum α étant fixe.

D'après la Figure III.8, la convergence est rapide pour η=0.9 et α=0.9.

Dans les deux dernières simulations, nous fixons le coefficient d'apprentissage η=0.5 puis nous faisons varier le momentum (α=0.9, α=0.5, α=0.1).
La Figure III.9, montre l'influence du momentum α sur la convergence du ***RN1_QRS***.

Pour les différentes simulations citées ci-dessus, nous remarquons que la convergence est meilleure pour α=0.9 et η=0.5.

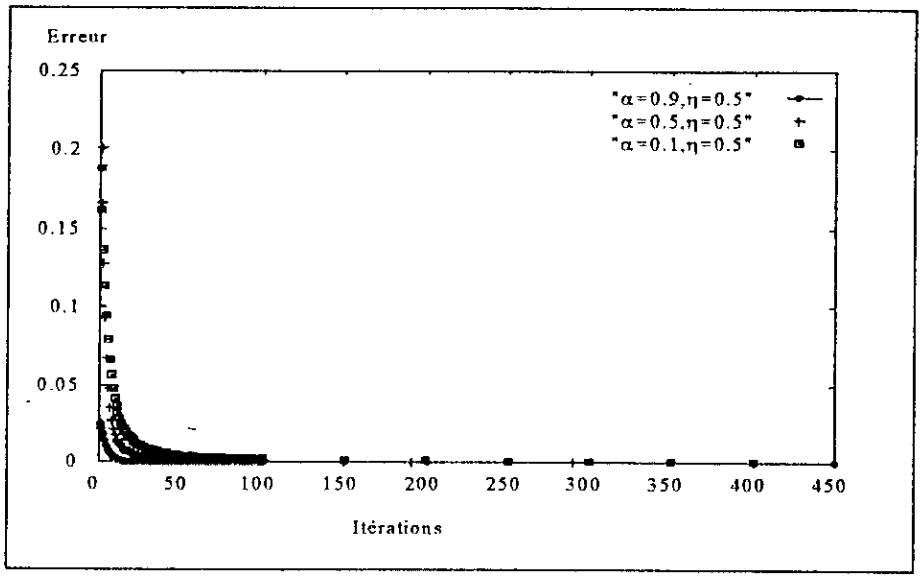


Figure III.9 Evolution de l'erreur en fonction du momentum α, le coefficient η étant fixe.

Néanmoins, le critère de convergence ne suffit pas à lui seul pour déterminer les meilleurs valeurs paramètres α, η.

Afin de déterminer les meilleures valeurs des paramètres α et η, nous devons étudier l'influence de ces paramètres sur les performances du classificateur.

III.3.4.2.4 Etude de l'influence du coefficient d'apprentissage, et du momentum sur les performances du classificateur *RN1_QRS*

Dans cette section nous étudions l'influence du coefficient d'apprentissage, η, et du facteur momentum, α, sur les performances du classificateur ***RN1_QRS***.
Afin d'évaluer les performances du classificateur nous avons calculé les paramètres %CC, %FP, %FN, %SE, %SP et %PR pour les mêmes simulations cités ci dessus, à savoir :

Simulation 1 : α=0.9, η=0.1
Simulation 2 : α=0.9, η=0.5
Simulation 3 : α=0.9, η=0.9
Simulation 4 : α=0.5, η=0.5
Simulation 5 : α=0.1, η=0.5

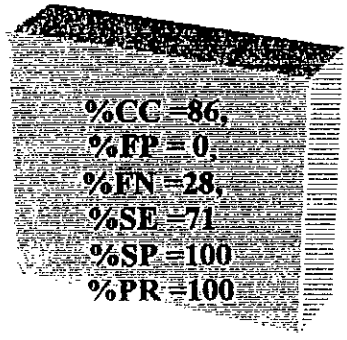
Le tableau III.2 présente les différentes mesures des performances %CC, %FP, %FN, %SE, %SP, et %PR.

Tableau III.2. Evaluation des performances de RN1_QRS en fonction de η, α

SIMULATION	α	η	%CC	%FP	%FN	%SE	%SP	%PR
1	0.9	0.1	86	10	20	80	90	80
2	0.9	0.5	86	0	28	71	100	100
3	0.9	0.9	80	11	33	66	88	80
4	0.5	0.5	80	11	33	66	88	80
5	0.1	0.5	80	11	33	66	88	80

Le tableau III.2 montre que pour les simulations 3,4 et 5 les performances du classificateur RN1_QRS sont identiques.
De même, le pourcentage de classification correcte %CC, est le même pour les simulations 1 et 2, soit %CC=86 .
La simulation 2, a permis de réduire le pourcentage d'erreur dans le diagnostic , %FP, à zéro. Cette réduction a permis d'atteindre une précision , %PR et une spécifité ,%SP de 100. Néanmoins, ceci est réalisé au prix d'une réduction de la sensibilté, %SE, et une augmentation d'un diagnostic "false negatif", %FN,en comparaison avec la simulation 1.

Enfin, nous choisissons de prendre les paramètres α, β de la simulation 2 vu l'importance du paramètre FP.
De ce fait, les performances du classificateur RN1_QRS sont:



III.3.4.3 Classificateur temporel *RN2*

III.3.4.3.1 Apprentissage

Les caractéristiques temporelles PP, PR et RR en combinaison avec les sorties de RN1_P et RN1_QRS forment le vecteur d'entrée de RN2. Pour effectuer l'apprentissage, nous avons sélectionné 24 vecteurs. Nous avons fixé une sortie désirée pour chaque arythmie, comme indiqué ci dessous :

Arythmie	Sortie désirée
SN	0
TS	0.08
BS	0.16
ESE	0.24
TA	0.32
FibA	0.40
FA	0.48
ESV	0.56
FibV	0.64
FV	0.72
TV	0.80
TSV	1

III.3.4.3.2 Dimensionnement du réseau

Nous appliquons la même stratégie présentée dans III.3.4.1 , pour le dimensionnement de RN2

La couche d'entrée du classificateur ***RN2*** est constituée de 5 neurones et la couche de sortie est constituée d'un seul neurone.

Après plusieurs simulations, nous avons obtenu la convergence pour les paramètres suivants :

Couche d'entrée : 5 neurones
Couche cachée : 10 neurones
Couche de sortie : 1 neurone
Coefficient d'apprentissage, η : 0.5
Momentum, α : 0.9
Erreur atteinte, $\varepsilon = 10^{-2}$
Nombre d'itérations : 47

III.3.4.3.3 Etude de l'influence du coefficient d'apprentissage et du momentum sur la convergence du réseau *RN2*

Dans cette section nous étudions l'influence du coefficient d'apprentissage η et du momentum α sur la convergence du réseau ***RN2***. Pour cela, nous avons effectué les simulations suivantes :

Simulation 1 : α=0.9, η=0.1
Simulation 2 : α=0.9, η=0.5
Simulation 3 : α=0.9, η=0.9
Simulation 4 : α=0.5, η=0.5
Simulation 5 : α=0.1, η=0.5

Pour les trois premières simulations, le momentum α est maintenu constant à une valeur α=0.9 alors que le coefficient d'apprentissage est variable (η=0.1, η=0.5 et η=0.9).

La Figure III.10 montre l'influence du coefficient d'apprentissage sur la convergence de ***RN2***, pour les trois premières conditions de simulation citées ci dessus.

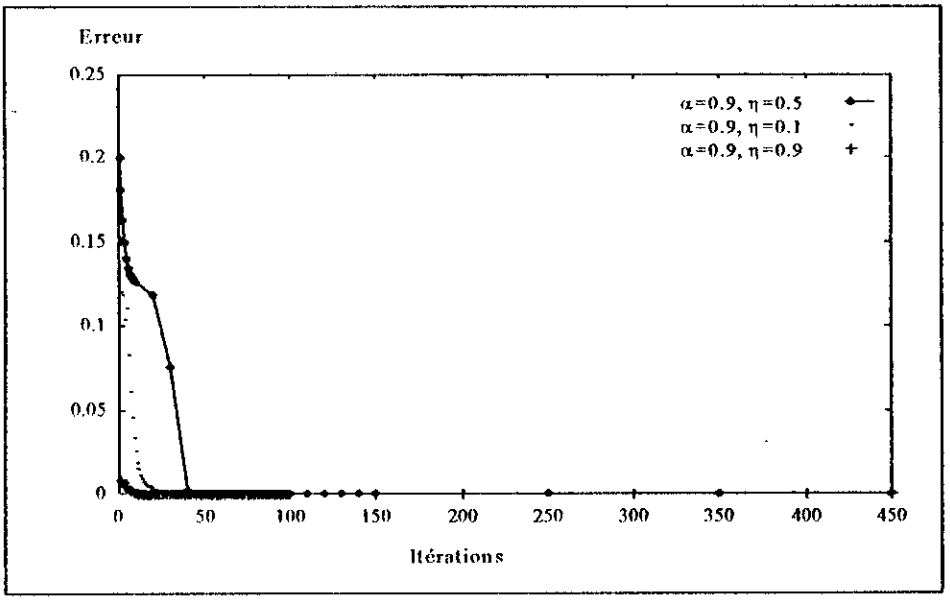


Figure III.10 Evolution de l'erreur en fonction du coefficient η, le momentum α étant fixe.

D'après la Figure III.10, la meilleur convergence est obtenue pour η=0.9 et α=0.9.

Dans les deux dernières simulations, nous fixons le coefficient d'apprentissage η=0.9 puis nous faisons varier le momentum (α=0.9, α=0.5, α=0.1).
La Figure III.11, montre l'influence du momentum α sur la convergence du ***RN2***.

Pour les différentes simulations citées ci-dessus, nous remarquons que la meilleur convergence est obtenue pour α=0.9 et η=0.9.

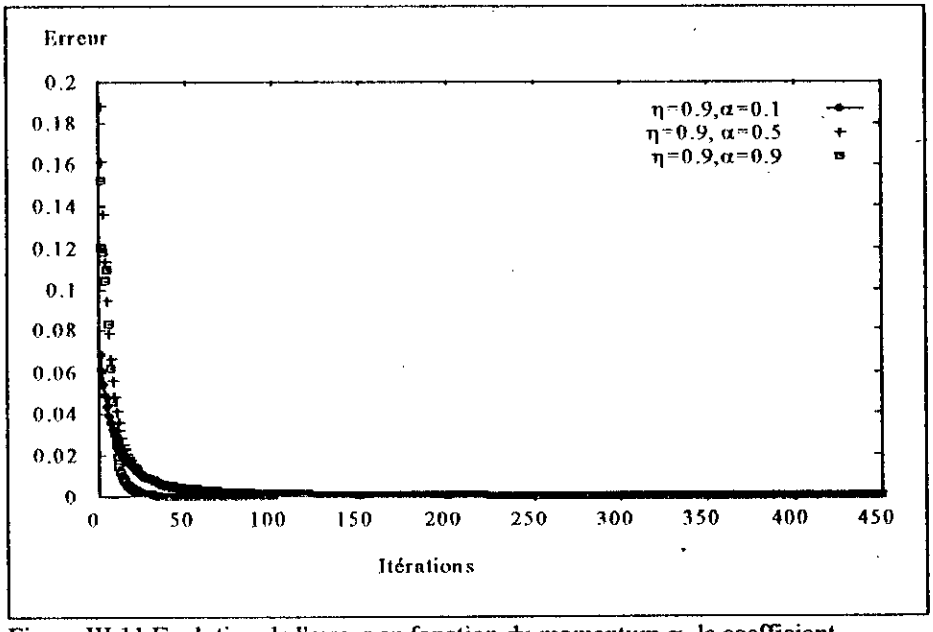


Figure III.11 Evolution de l'erreur en fonction du momentum α, le coefficient η étant fixe.

III.3.4.3.4 Etude de l'influence du coefficient d'apprentissage, et du momentum sur les performances du classificateur *RN2*

Dans cette section nous étudions l'influence du coefficient d'apprentissage, η, et du facteur momentum, α, sur les performances du classificateur ***RN2***.

Afin d'évaluer les performances du classificateur nous avons calculé les paramètres %CC, %FP, %FN, %SE, %SP et %PR pour les mêmes simulations cités ci dessus, à savoir :

Simulation 1 : α=0.9, η=0.1
Simulation 2 : α=0.9, η=0.5
Simulation 3 : α=0.9, η=0.9
Simulation 4 : α=0.5, η=0.5
Simulation 5 : α=0.1, η=0.5

Le tableau III.3 présente les différentes mesures des performances %CC, %FP, %FN, %SE, %SP, et %PR.

Tableau III.3. Evaluation des performances de RN2 en fonction de η, α

SIMULATION	α	η	%CC	%FP	%FN	%SE	%SP	%PR
1	0.9	0.1	76	42	3	98	57	85
2	0.9	0.5	76	42	3	98	57	85
3	0.9	0.9	76	42	3	98	57	85
4	0.5	0.5	76	42	3	98	57	85
5	0.1	0.5	76	42	3	98	57	85

Le tableau III.3 montre que pour les performances du classificateur RN2 sont les mêmes pour toutes les simulations 1,2,3,4 et 5.
De ce fait, les performances du classificateur RN2 sont :

%CC =76,
%FP = 42,
%FN =3,
%SE =98,
%SP =57,
%PR =85

III.4 Discussion et Conclusion

Dans ce chapitre nous avons proposé une nouvelle approche pour la classification des arythmies cardiaques basée sur l'utilisation d'un classificateur de la morphologie de l'onde P, ***RN1_P***, un classificateur pour la morphologie du complexe QRS, ***RN1_QRS*** et un classificateur temporel, ***RN2***.
Par la suite nous avons présenté la conception software de chaque classificateur.
Nous avons montré que les performances d'un classificateur dépendent étroitement de la taille du réseau de neurones, du coefficient d'apprentissage η et du facteur mometum,α.

Si nous faisons une évaluation globale des trois classificateurs, nous retenons les résultats suivants qui correspondent au taux de classification correcte pour chaque classificateur :

Classificateur	%CC
RN1_P	93
RN1_QRS	86
RN2	76

Ces résultats montrent que :

1. Le classificateur de la morphologie de l'onde P, RN1_P, présente des performances supérieures à ceux du classificateur de la morphologie du complexe QRS, RN1_QRS.
2. Les classificateurs RN1_P et RN1_QRS présentent des performances supérieures à ceux classificateur temporel, RN2.

Les résultats énoncés précédemment, trouvent leur justification, dans les points ci dessous, respectivement, :

1. L'onde P présente une forme simple et régulière, en comparaison avec le complexe QRS.
2. ***RN1_P*** et ***RN1_QRS*** sont conçus pour séparer entre deux classes (Normal ou Anormal) alors que ***RN2*** à pour tâche la séparation de 12 classes (12 anomalies).

Une fois les performances évaluées et jugées satisfaisantes, nous retenons les matrices des poids synaptiques suivants :

Pour le réseau ***RN1_P*** (10-4-1), les poids synaptiques *Wji* et *Wkj* obtenus après apprentissage sont :

$$Wji = \begin{bmatrix} -0.050 & -0.05 & -0.050 & -0.050 \\ -0.44 & -0.37 & -0.115 & -0.354 \\ -0.80 & -0.67 & -0.202 & -0.462 \\ -1.38 & -1.129 & -0.149 & -1.652 \\ -1.47 & -1.246 & -0.764 & -2.089 \\ -1.52 & -1.337 & -0.759 & -2.517 \\ -1.29 & -1.268 & -1.582 & -0.169 \\ -0.76 & -0.858 & -1.807 & 1.102 \\ -0.19 & -0.387 & -1.813 & 1.843 \\ -0.10 & -0.301 & -1.826 & 2.636 \end{bmatrix}, \quad Wkj = \begin{bmatrix} 0.657 \\ 0.946 \\ 1.763 \\ -2.591 \end{bmatrix}$$

Pour le réseau ***RN1_QRS*** **(10-4-1)**, les poids synaptiques *Wji* et *Wkj* obtenu après apprentissage sont :

$$Wji = \begin{bmatrix} -0.050 & -0.050 & -0.050 & -0.050 \\ 0.308 & -0.342 & -1.069 & -0.657 \\ 0.576 & -0.401 & -1.455 & -1.081 \\ -1.083 & -1.227 & -2.705 & -1.650 \\ -0.383 & -0.058 & -1.000 & -0.903 \\ -0.781 & -0.075 & -1.000 & -1.053 \\ -1.410 & 0.027 & -1.293 & -1.350 \\ 2.980 & 0.625 & -2.362 & -0.898 \\ 1.309 & 0.529 & -0.565 & -0.068 \\ 0.639 & 0.169 & -0.255 & -2.253 \end{bmatrix}, \quad Wkj = \begin{bmatrix} -2.253 \\ -1.710 \\ -0.268 \\ -0.841 \end{bmatrix}$$

Les poids synaptiques obtenus pour le réseau RN2 (5-10-1) sont :

$$Wji = \begin{bmatrix} -1.004 & -1.219 & -1.080 & -0.698 & -1.462 & -1.209 & -0.609 & -1.253 & -0.489 & -1.147 \\ -0.710 & -0.764 & 0.129 & -0.205 & -0.690 & -0.759 & -0.521 & -0.511 & -0.286 & -0.294 \\ -0.666 & -1.034 & -1.367 & -1.558 & 0.504 & -1.124 & -1.555 & -1.00 & -1.411 & -0.302 \\ -0.093 & 0.372 & -0.024 & -0.362 & -0.341 & 0.328 & -0.446 & -0.078 & -0.335 & -3.403 \\ -0.719 & -0.307 & -1.569 & -1.395 & 0.187 & -0.994 & -1.367 & -1.228 & -1.731 & -0.234 \end{bmatrix}$$

$$Wkj = \begin{bmatrix} -0.671 & -0.504 & 0.900 & 0.765 & -2.019 & 0.215 & 1.042 & 0.329 & 0.606 & -0.821 \end{bmatrix}^{-1}$$

Les poids synaptiques ainsi déterminés seront utilisés dans le prochain chapitre, pour l'implémentation digitale du CNAC.

Chapitre IV

IMPLEMENTATION DIGITALE DU CNAC

IV.1 Introduction

Cette partie concerne l'implémentation digitale du classificateur neuronal des arythmies cardiaques, CNAC, sur FPGA. Les poids synaptiques déterminés dans la phase d'apprentissage du classificateur seront utilisés pour l'implémentation digitale du CNAC. Comme nous l'avons mentionné au chapitre précédent, CNAC est composé de deux sous classificateurs : un classificateur de morphologie RN1 en cascade avec un classificateur temporel ***RN2****. A son tour, le classificateur de morphologie est composé de deux sous classificateurs : un sous classificateur neuronal pour l'onde P,* ***RN1_P****, et un autre sous classificateur neuronal pour le complexe QRS,* **RN2_QRS**. *Pour l'implémentation hardware, nous devons considérer chaque réseau de neurones indépendamment des autres. De ce fait, Vu la complexité des réseaux de neurones d'une part et le nombre de circuits requis d'autre part, nous avons opté pour une méthodologie de conception descendante basée sur l'utilisation de la synthèse logique et du langage VHDL. Avant de passer à l'implémentation digitale du CNAC sur FPGA, nous présentons les éléments nécessaires pour la conception du circuit.*

IV. 2 La technologie FPGA [41, 42 , 43]

Les FPGAs "Field Programmable Gate Arrays" sont des circuits logiques programmables.
A l'origine des composants programmables, on retrouve la technologie des mémoires PROM. Une PROM est un circuit programmable une seule fois, constituée d'une matrice de cellules configurées pour la lecture uniquement. En effet, une mémoire peut servir à codifier un ensemble de vecteurs logiques avec les entrées sur l'ensemble des bits d'adresse et en sortie les bits de données. Avec cette stratégie, n'importe quelle table de vérité d'une fonction logique peut être implémentée. Mais cette technique, pour puissante qu'elle soit est un gâchis du silicium avec toutes les conséquences qui en découlent (rapidité d'action, coût de fabrication...). Pour cela, une structure faite de matrices de connexions associées à des portes logiques a été développée et implémentée dans les composants que l'on a appelé PLD "Programmable Logic Device".
Un PLD comprend une matrice de portes AND connectée à une matrice de portes OR. Un circuit logique destiné à être implémenté en PLD est alors représenté sous forme de sommes de produits.
La version essentielle des PLD est la matrice programmable logique PAL (Programmable Array Logic). Une PAL est constituée d'un plan de portes AND programmables suivi d'un plan de portes OR fixes. Une version plus flexible du PAL est le PLA "programmable logic array".
Les PLAs sont constitués par un plan de portes AND suivi d'un plan de portes OR. Mais dans ce cas les connexions dans les deux plans sont programmables.
Avec leurs simples structures, les deux types de PLD décrits précédemment permettent d'atteindre des performances en vitesse élevées. Cependant, leur principal défaut c'est qu'ils ne peuvent implémenter que de petits circuits logiques représentant un nombre modéré de sommes de produits, car la structure des interconnexions dans ces circuits croit avec le nombre des termes de produits.
A l'autre bout de la chaîne, on retrouve les circuits MPGAs " Mask programmable Gate Arrays". Un MPGA est un circuit VLSI "Very large Scale Integration" dans lequel toutes les couches des masques qui définissent la circuiterie de la puce sont prédéfinies par le manufacturier à l'exception des couches métalliques qui permettent de réaliser les connexions entre les transistors de la matrice MPGA. L'avantage principal des MPGAs est qu'ils permettent l'implémentation de circuits plus larges. Ceci est dû principalement à la structure de leur interconnexion qui est proportionnelle à la logique utilisée. Néanmoins, ces circuits nécessitent un retour chez le manufacturier ; par conséquent le temps de fabrication est assez élevé.
Dans l'industrie électronique, il est vital d'atteindre le marché avec de nouveaux produits en un temps le plus court que possible. Une telle réduction du temps de développement (conception) et de production est devenue essentielle. De plus, il est important que les risques financiers entraînés dans le développement d'un nouveau produit soient limités de telle sorte que plus d'idées nouvelles peuvent être prototypées.
D'un point de vu économique, les circuits FPGAs ont émergés comme une solution ultime aux problèmes de risques car ils permettent une production instantanée (in situ) et des prototypes à faible prix comparé aux MPGAs.
D'un point de vue technologique, un FPGA combine la programmabilité d'un PLD et la structure des interconnexions d'un MPGA. Ce qui résulte en des composants logiques programmables et à très grande densité d'intégration.

IV.2.1 Qu'est qu'un FPGA ?

Comme un MPGA, un FPGA consiste en une matrice d'éléments singuliers (blocs logiques) qui peuvent être interconnectés de façon générale. Comme un PLD, les interconnexions entre les éléments sont programmées par l'utilisateur. Les premiers FPGAs ont été introduits en 1985 par la compagnie Xilinx. Par la suite plusieurs autres compagnies telles que Actel, Altera, Plus, Advanced Microdevices (AMD), Quick-logic et autres, ont introduit de nouvelles architectures FPGAs.
La figure IV.1 montre les quatre classes principales et commercialement disponibles des FPGAs : "Symmetrical array", "Row-based array", "Sea-of-Gates" et les "Hierarchical PLD".
Deux paramètres caractérisent un FPGA : les blocs logiques et les ressources d'interconnexions. Un circuit logique peut être implémenté dans un FPGA en partionnant la logique dans les blocs logiques individuels et en interconnectant ces blocs à travers les switches nécessaires.

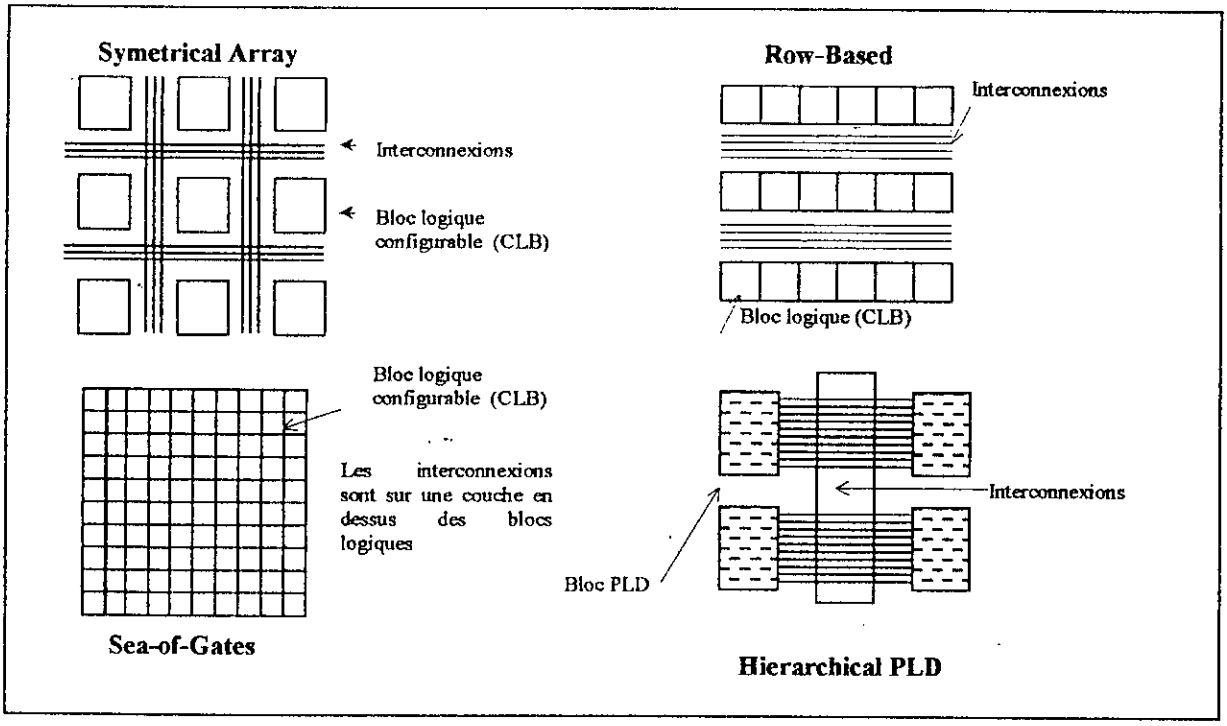


Figure IV.1. Les quatre classes des FPGA commercialement disponibles

IV.2.1.1 Structures d'un Bloc logique

Il existe deux catégories de blocs logiques caractérisant un circuit FPGA : les structures basées sur les tables de transfert "look-up table based (LUT-based)" et celles basées sur les multiplexeurs "multiplexor-based (MUX-based)". Une LUT peut implémenter n'importe quelle fonction booléenne de m entrées, $m \geq 2$, d'où le nom de générateur de fonction. Pour une architecture donnée m est fixé. Dans les architectures commerciales m est typiquement entre 3 et 6. De plus, chaque bloc logique (CLB) peut avoir un ou plusieurs générateurs de fonctions (LUT) qui sont connectés à d'autres éléments logiques telles que les bascules, les circuits à retenue logique, etc. Les Figures IV.2.a et IV.2.b montrent les blocs de base des circuits FPGA de la compagnie Xilinx pour la famille XC3000 et XC4000 respectivement.

Un FPGA de la famille XC3000 possède deux générateurs de fonctions, cinq entrés et deux sorties logiques par CLB, ce qui minimise ses options et restreint ses modes d'application, mais il contient par ailleurs un oscillateur cristal qui le rend rapide pour des opérations simples.

Un FPGA de la famille XC4000 possède trois générateurs de fonctions (LUT F, LUT G et LUT H), des bascules FF et quatre sorties. Cette famille intègre un très grand nombre de portes (plus de 125000 portes actuellement) et possède une retenue logique rapide.

Dans les architectures Mux-based, le bloc de base est une configuration de multiplexeurs avec quelques portes logiques AND et OR. La Figure IV.2.c montre l'architecture de base d'un bloc logique du circuit FPGA de la compagnie Actel.

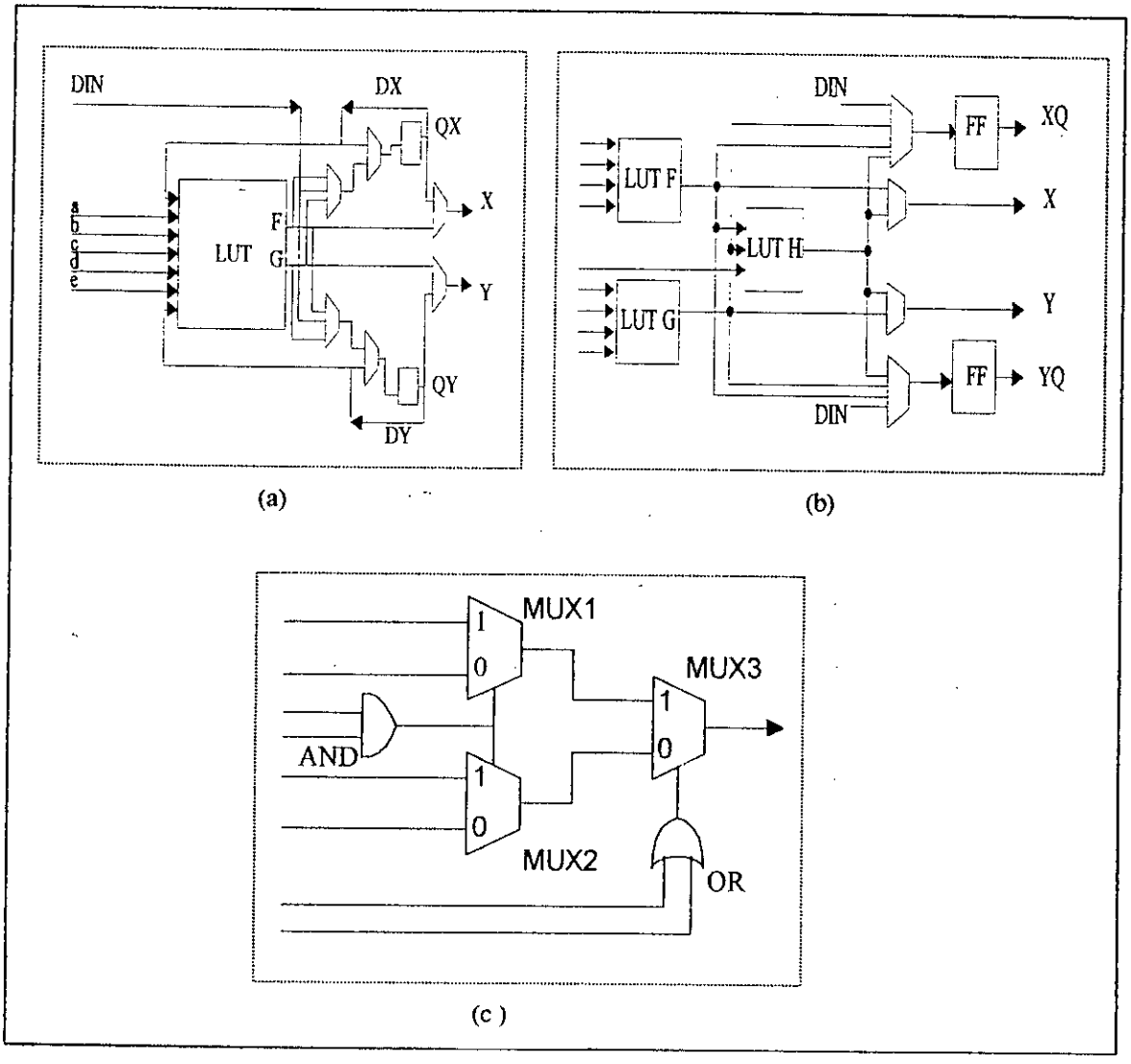


Figure IV.2. Différentes architectures de bloc logique configurables : (a) Compagnie Xilinx-famille XC3000. (b) Compagnie Xilinx Famille XC4000. (c) Compagnie Actel: act2.

IV.2.1.2 Ressources des interconnexions

Les interconnexions entre les blocs logiques doivent être programmées pour réaliser la connexion désirée. Il existe deux types d'interconnexions : les interconnexions ***Reprogrammables*** et les interconnexions programmables une seule fois ***"one time programmable*** (OTP)".
Puisqu'il existe plusieurs façons pour implémenter un élément programmant, il est devenu d'usage de parler de la technologie de programmation.
Les technologies de programmation qui sont actuellement utilisées dans les produits commerciaux sont : les cellules statiques RAM (SRAM), les "antifuses", les transistors EPROM et EEPROM. Les références [41], [42], contiennent plus détails concernant ce sujet.

Particulièrement, la compagnie Xilinx offre des circuits FPGAs reprogrammables, utilisant la technologie de programmation SRAM. Dans ce type de circuits, la connexion entre les différents blocs logiques est assurée par un programme enregistré dans une mémoire. Ce programme est chargé automatiquement lors de chaque mise en route dans une mémoire interne du FPGA. La modification du programme peut changer le fonctionnement du circuit même en cours de marche du système, cette flexibilité peut être assez intéressante pour certaines applications , nous citons à titre d'exemple le circuit FPGA proposé dans [44] qui réalise la fonction de compression lors de l'émission d'un signal et la fonction de décompression lors de la réception , et le circuit FPGA pour les réseaux de neurones reprogrammables [45].
Par contre à la différence de la compagnie Xilinx, la compagnie Actel offre des circuits FPGAs OTP basés sur la technologie antifuse.

Dans notre étude nous nous intéressons à l'implémentation du CNAC en utilisant les circuits FPGAs de la compagnie Xilinx.

IV.3 La synthèse

IV.3.1 Définition de la synthèse

De manière générale on définit la synthèse comme un processus de translation d'une description comportementale en une description structurelle ou physique.

IV.3.2 Les outils de Synthèse

Actuellement les différentes étapes du processus de conception d'un circuit intégré, de la spécification comportementale au dessin de masques ou "layout", s'emboîtent correctement.
Pour différencier les différents types de synthèse, et par conséquent entre les différents outils de synthèse qui en découlent, on utilise l'arbre Y [46] comme présenté dans la Figure IV.3. Les axes dans le graphe Y représentent trois différents domaines de description : le domaine comportemental, le domaine structurel et le domaine physique.

La représentation structurelle réalise un mapping d'une représentation comportementale sur un ensemble de composants et de connexions sous des contraintes tel que le coût, la surface et le délai.

La représentation physique réalise le mapping d'une structure sur du silicium.

En parcourant les arcs dans le graphe **Y**, du plus haut niveau d'abstraction au plus bas niveau, on distingue :

- La synthèse architecturale.
- La synthèse microarchitecturale encore appelé synthèse de transfert entre registres "register transfer synthesis".
- La synthèse de circuits.
- La synthèse de masques.

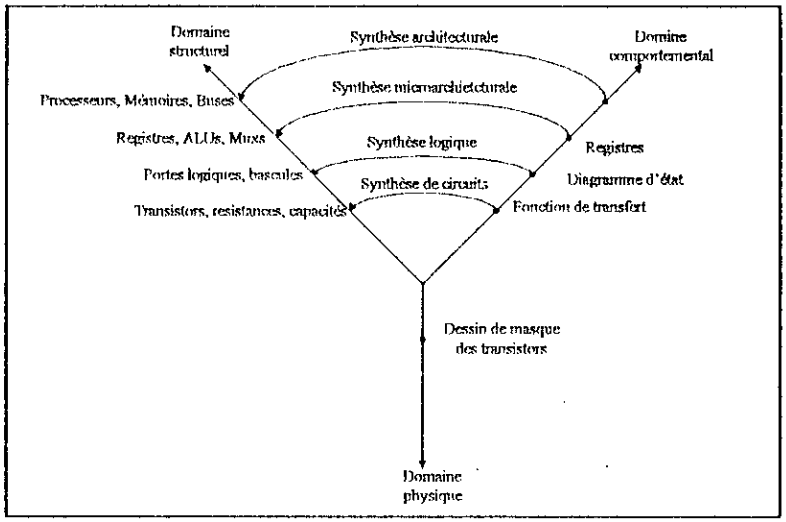


Figure IV.3. Présentation du Graphe **Y**

Pour décrire un comportement; la synthèse de circuits utilise les fonctions de transfert et les diagrammes temporels. La structure résultante est un circuit décrit par des transistors, des résistances et des capacités (Figure IV.3).

La synthèse logique, utilise des expressions booléennes et des diagrammes d'état ; la structure résultante est un circuit décrit, à un niveau plus supérieur que la synthèse de circuits, par des portes logiques et des bascules (Figure IV.3).

Dans le domaine microarchitecturale ou "register-transfer", le temps est divisé en des intervalles appelés états. On utilise une description à base de registres qui spécifie pour chaque état, la condition à tester. La structure résultante est un circuit décrit, à un niveau plus supérieur que la synthèse logique, par des registres, des unités arithmétiques et logiques (ALUs) et des multiplexeurs (Figure IV.3).

Le niveau système utilise des variables et des opérateurs d'un langage pour exprimer la fonctionnalité des composants d'un système. La structure résultante du circuit est représentée par des processeurs des mémoires et des buses (Figure IV.3).

IV.4 Le langage VHDL [47, 48, 49]

IV.4.1 Présentation du VHDL

Le VHDL est l'acronyme du mot anglais (VHSIC HDL : Very high Speed Integrated Circuits Hardware description langage), langage de description matérielle du projet VHSIC traitant les circuits intégrés à très grande vitesse.
Historiquement le VHDL a été initié par le DOD de département de la défense américaine, il est devenu standard de l'IEEE en 1987 (VHDL'87) puis révisé en 1992 (VHDL'92).
Le VHDL peut être utilisé pour plusieurs objectifs à savoir :

- Documentation.
- Simulation.
- Synthèse.
- Preuve formelle.

Généralement, une description d'un système en VHDL est constituée :
1. D'une ou plusieurs bibliothèques dans les quelles sont stockes des unités de conception.
2. Les unités de conception peuvent être de cinq catégories différentes qui sont les vues externes des modèles (entity et architecture), les vues externes et internes de paquetage et les configurations.
3. Les unités de conception contiennent des descriptions d'action séquentielles ou concurrentes faisant intervenir des objets appartenants à 3 classes : constantes, variables et signaux.
4. Chaque objet est typé et son type appartient à une des 4 familles de type scalaire, type composite, type accès ou fichier.

Parmi les avantages du VHDL nous insistons sur les aspects suivants :
- Le langage permet une description hiérarchique : un système peut être modélisé par un ensemble de composants, a son tour, peut être modélisé par une intermédiaire d'un ensemble de sous composants.
- Le langage n'est pas spécifique à une technologie particulière mais peut quand même supporter des caractéristiques spécifiques à une technologie cible.
- Le VHDL permet des descriptions génériques ou paramétrées. Ces caractéristiques seront largement exploités dans la description du réseau de neurone.

IV.4.2 Synthèse avec le VHDL

VHDL est un langage conçu avec une sémantique de simulation.
Pour le domaine de la synthèse, son utilisation est réduite à un sous-ensemble qui corresponde à des représentations digitales identifiables. Les résultats d'une synthèse dépendent du style VHDL utilisé. A l'heure actuelle, les algorithmes de synthèse avec le VHDL les plus utilisés permettent d'optimiser les circuits décrits au niveau comportemental et de cibler la technologie correspondante.

IV.4.3 La simulation en VHDL

La simulation en VHDL constitue l'un des aspects les plus importants du langage VHDL.
Les modèles et composants doivent être vérifiés dans toutes les étapes de développement.

Un simulateur doit vérifier si le modèle est syntaxiquement correct. Une fois la syntaxe vérifiée, le simulateur procède à la vérification fonctionnelle du modèle. Pour le simulateur les objets manipulés sont au minimum de trois types : les processus, les signaux et les pilotes. C'est à partir des stimulus fournis par les entrées que le simulateur évalue les expressions d'un processus via les pilotes, puis passant an processus suivant jusqu'à attendre le processus auquel est relié le signal de sortie.

IV.5. Implémentation digitale du CNAC [50, 51]

Les trois classificateurs ***RN1_P***, ***RN1_QRS***, et ***RN2*** qui constituent CNAC possèdent la même structure. La différence réside dans la taille du réseau à implémenter. Dans ce qui suit nous présenterons la conception relativement à un seul réseau de neurones, par la suite nous présenterons les résultats pour chacun des trois réseaux.

IV.5.1 Approche de conception

L'approche de conception est l'approche descendante basée principalement sur la synthèse avec le VHDL, que nous avons présenté dans la section IV.3.1
Comme l'illustre la Figure IV.4, une architecture est fixée pour le réseau de neurones, cette phase est suivie par le mapping de cette architecture en une description VHDL. Par la suite, nous utilisons l'outil de synthèse GALILEO [52], qui réalise automatiquement le mapping du VHDL en technologie FPGA, selon les contraintes de vitesse ou surface. Le résultat de la synthèse est une netlist en format XNF. Celle ci est utilisée à l'entrée de l'outil XACT [53] de la compagnie Xilinx qui réalise le placement et routage du réseau de neurones. Une vérification de la fonctionnalité du circuit est nécessaire à chaque phase.
Dans ce qui suit, chaque phase sera décrite en détail.

IV.5.2. Architecture du réseau de neurones

Nous avons montré au chapitre I que lors de la conception d'un réseau de neurone, les paramètres qu'il faut prendre en considération lors de la conception d'un réseau de neurones sont :

- Le parallélisme qui permet le traitement en temps réel.
- Les performances relativement par rapport à la complexité de connexions synaptiques.
- La flexibilité ou adaptabilité du circuit.
- La surface du silicium.

Pour atteindre ces objectifs, une bonne implémentation VLSI doit avoir les propriétés architecturales suivantes [54] :

- Simplicité de conception caractérisée par une architecture basée sur des copies de quelques cellules simples.
- Régularité de la structure permettant de réduire les interconnexions
- Possibilité d'extension et de réduction de l'architecture.

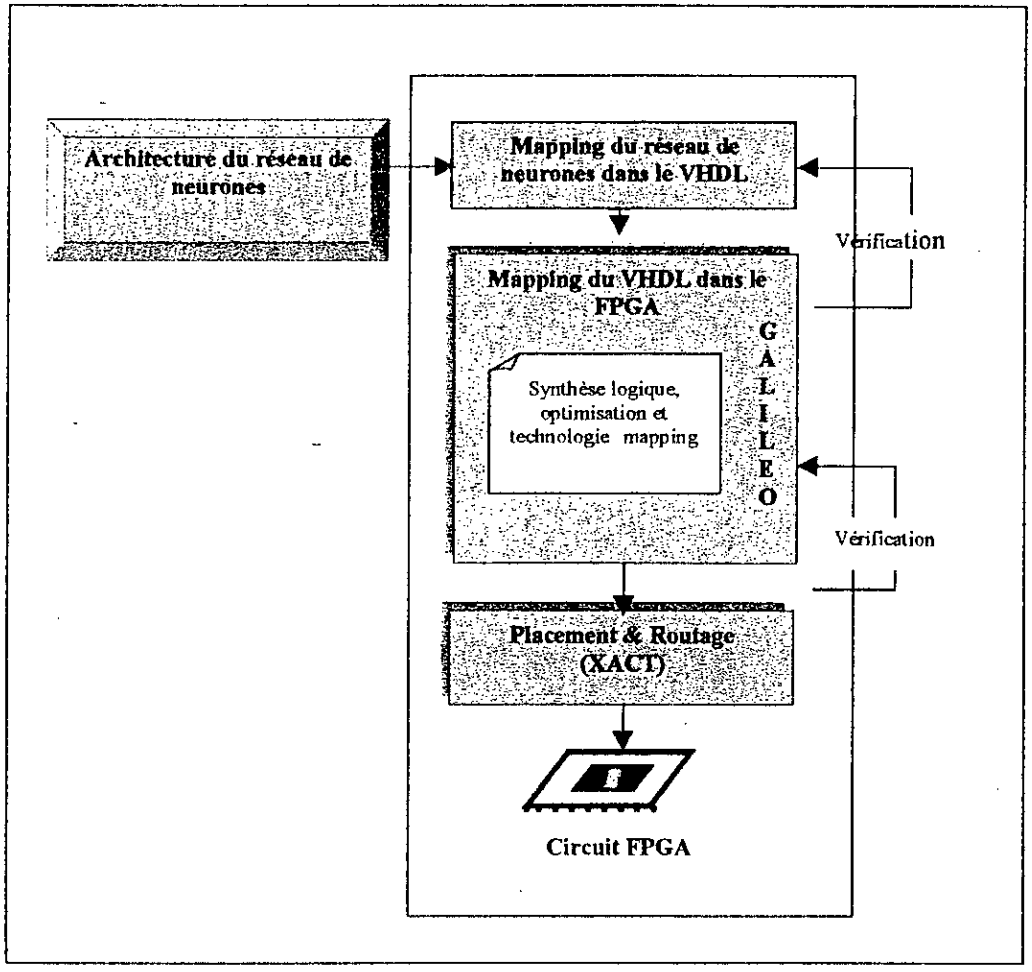


Figure IV.4. Méthodologie de conception du réseau de neurones

La conception d'une architecture complètement parallèle nécessite [51] :

- Le parallélisme entre toutes les couches du réseau de neurones : cela signifie qu'au moins un multiplieur doit être utilisé pour chaque couche.
- Le parallélisme des nœuds ou neurones : Pour réaliser cette condition, un seul neurone suffit.
- Le parallélisme des connexions synaptiques : c'est le plus haut degré de parallélisme, souhaité dans un réseau de neurones. Pour atteindre ce parallélisme, il faudrait prévoir un multiplieur pour chaque connexion. Cependant, l'utilisation d'un grand nombre de multiplieurs pour un grand nombre de connexions synaptiques est une pénalité sévère pour les circuits digitaux et plus particulièrement les circuits FPGAs à cause de leur ressources limitées (complexité d'intégration et problèmes de délai).

Par conséquent, nous considérons uniquement le parallélisme des nœuds. Par ailleurs, le transfert de données entre les couches se fait en série car un neurone est conçu pour calculer une seule connexion synaptique à la fois.

IV.5.2.1 Architecture du neurone

Sur la base des constatations ci dessus, nous proposons une architecture FPGA pour le modèle du neurone artificielle (Voir Figure IV.5.a et Figure IV.5.b). Comme l'illustre la Figure IV.5.b, ce modèle est principalement basé sur :

- Un circuit mémoire (ROM) permettant de stocker les poids synaptiques.
- un circuit multiplieur accumulateur (MAC) qui calcul la somme pondérée et
- Une table de transfert (LUT) qui implémente la fonction d'activation.

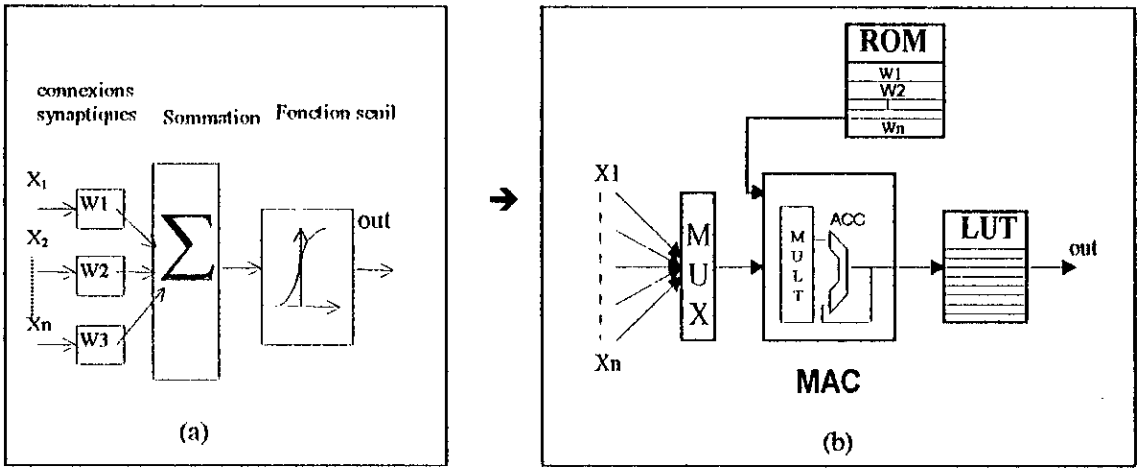


Figure IV.5. (a) : Neurone artificiel. (b): Model FPGA équivalent

IV.5.2.2 Architecture du réseau de neurones

L'architecture FPGA résultante du réseau de neurones est présentée dans la Figure IV.6. Elle présente les caractéristiques suivantes :

- Pour un même neurone, seulement un seul circuit MAC est utilisé pour calculer la somme des produits
- Chaque circuit MAC possède sa propre ROM où sont mémorisés les poids synaptiques. La profondeur de chaque ROM est égale au nombre de nœuds relativement à sa couche d'entrée.
- Pour la même couche, un calcul parallèle des neurones est réalisé.
- Le calcul entre les couches se fait en série.
- Le réseau de neurones est contrôlé par une unité de contrôle.

Comme nous pouvons le constater l'architecture résultante :

- Intègre un haut degré de parallélisme,
- Une simplicité de conception caractérisée par une répétition (ou copie) de cellules simples.
- Une régularité de la structure.

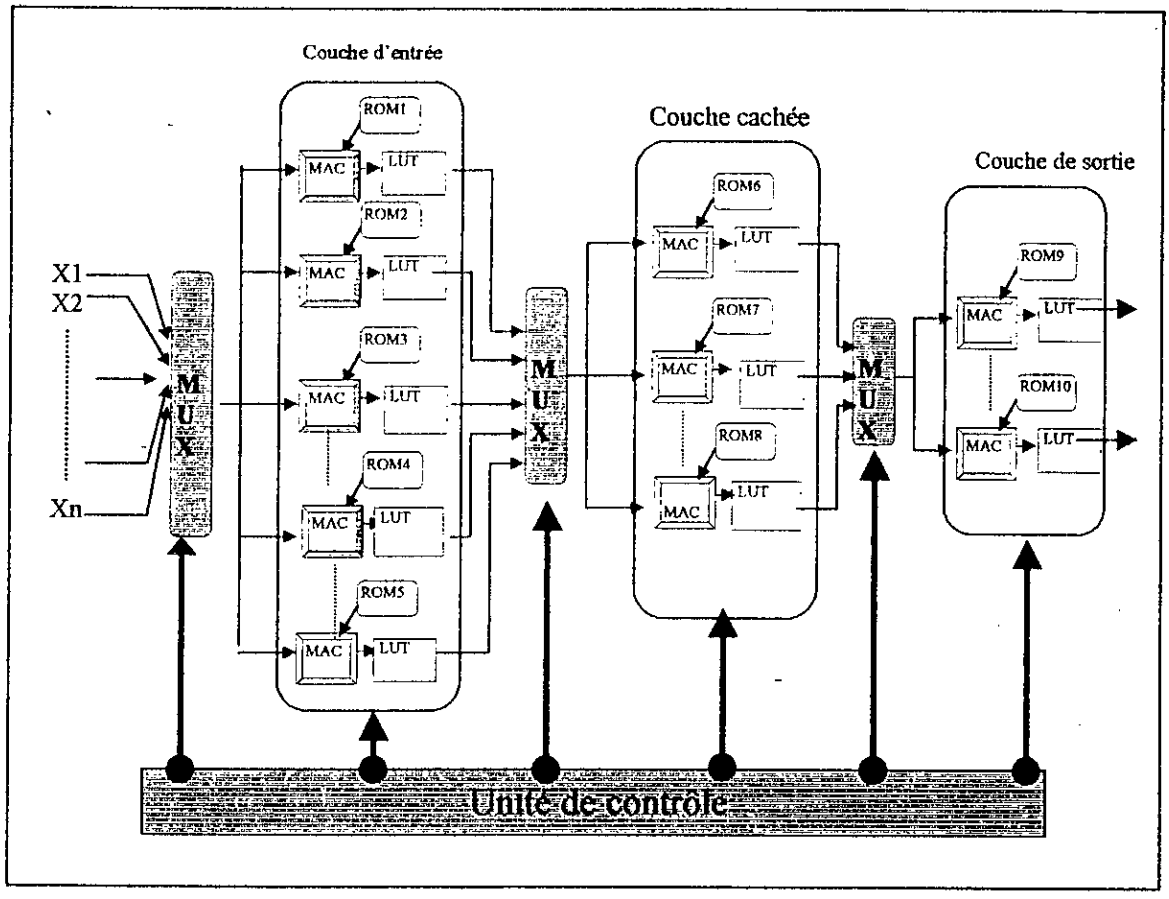


Figure IV.6. Architecture du réseau de neurone.

Maintenant qu'on est-il de la flexibilité du réseau de neurones ?
La réponse à cette question, nous la trouvons au paragraphe qui suit.

IV.5.3 Mapping de l'architecture en VHDL

Ayant fixé une architecture, cette partie concerne la description VHDL du réseau de neurones. Nous, nous sommes fixés comme objectif une description flexible du réseau de neurone.
La capacité du langage VHDL à supporter les descriptions paramètrées est la solution clé pour obtenir un réseau de neurones flexible qui peut s'adapter à plusieurs spécifications [55].
En plus de son émergence comme un standard industriel pour la conception hardware, le VHDL possède des caractéristiques supplémentaires tel que l'encapsulation, l'héritage, la généricité, et la réutilisation à l'intérieur d'une description.

- L'encapsulation réduit le nombre de détails que le concepteur doit confronter, et ceci à travers la représentation d'une conception comme un ensemble d'unités de conception interactives. Ainsi, le concepteur n'a pas à savoir comment ces unités fonctionnent à l'intérieur mais, il doit concentrer ces efforts sur la définition d'une interface appropriée entre ces unités. Dans le langage VHDL, l'encapsulation est renforcée à travers l'utilisation des paquetages, les appels de fonctions, les procédures et la déclaration des entités.
- L'héritage, enVHDL, est réalisé à travers l'instanciation des composants. L'instanciation d'un composant est le fait de prendre une copie d'un composant déclaré et de le personnaliser afin de répondre aux spécifications de la conception.
- La généricité est le moyen de transmettre une information à un bloc. Cette information est statique pour le bloc. Un bloc générique est vu de l'extérieur comme un bloc paramétré.

De l'intérieur du bloc, ses paramètres sont vus comme des constantes et l'on peut les manipuler comme tels.

- La réutilisation peut être réalisée en construisant des bibliothèques paramétrées (génériquës) de composants, de cœurs "cores", de macro-cellules et de paquetages.

Notre approche pour la description VHDL du réseau est illustrée dans la Figure IV.7.
La description VHDL du réseau commence par la création d'un composant *neurone*, ensuite un composant *couche* est crée et enfin un *réseau* est décrit.

- Le composant ***neurone*** est composé d'un composant MAC, d'un composant ROM et d'un composant LUT.
- Le composant ***couche*** composé d'un ensemble de *neurones*.
- Le ***réseau*** est composé d'un ensemble de *couches* (couche_1, couche_2, couche_3) et de multiplexeurs.

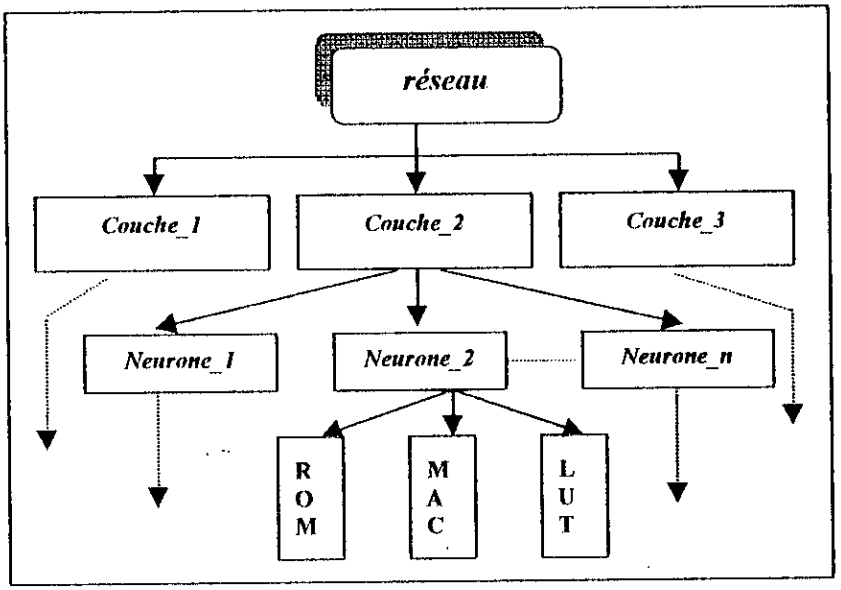


Figure IV.7. Approche pour la description d'un réseau de neurones

IV.5.3.1Flexibilité du neurone

La description VHDL d'un neurone nécessité la description préalable du circuit MAC et des mémoires ROM et LUT (ANNEXE 3). Pour réaliser la flexibilité de ces composants, la taille des mots (nb_bit) du MAC et la taille des mémoires (addr et addr1) sont choisies comme des paramètres génériques (Voir Figure IV.8 et ANNEXE 3). Les paramètres qui réalisent la flexibilité du neurone, sont : la taille du vecteur d'entrée (nb_bit) du neurone, l'adresse (addr) et l'instantiation des composants (component MAC, component ROM et component LUT de la Figure IV.8).
Nous pouvons appliquer cette description à un autre neurone de taille différente.
Nous pouvons aussi , grâce à cette description, changer les performances du neurone en choisissant d'autres descriptions prédéfinis des composants MAC, ROM et LUT stockes préalablement dans une bibliothèque sans changer le code de description VHDL du neurone.

IV.5.3.3 Flexibilité de la couche

Comme l'illustre la Figure IV.9, les paramètres qui réalisent la flexibilité de la couche sont : la taille des données (nb_bit) à l'entrée de la couche et le nombre d'adresse dans cette couche (addr). Avec cette description, nous pouvons facilement modifier le nombre de neurones d'une couche par une simple modification du code VHDL relativement à cette couche. De plus, nous pouvons facilement changer les performances de la couche en choisissant une autre description architecturale du composant neurone.

IV.5.3.3 Flexibilité du réseau de neurones

Les paramètres qui réalisent la flexibilité du réseau de neurones sont : la taille du mot (nb_bit), la taille des mémoires (addr1 et addr2) relativement à chaque couche, la taille des des deux multiplexeurs (ad_sel et ad_sel2) constituant le réseau et l'instanciation des couches (*component couche_mux1, component couche_mux2*). Avec cette description, nous pouvons facilement modifier la taille du réseau. Nous pouvons aussi changer les performances du réseau en utilisant d'autres descriptions architecturales des différentes couches, préalablement stockées dans une bibliothèque.

```
entity Neurone is
 generic(nb_bit:integer:=;addr:integer:=);-- taille du mot, nombre d'adresse
 port(in1 : in unsigned(addr-1 downto 0);
      in2:in std_logic_vector((nb_bit -1) downto 0);
      clk,rst,ready,read_en1:in std_logic;
      out: out std_logic_vector((nb_bit-1) downto 0));
end Neurone;

architecture structurelle of Neurone is
 component ROM1 is                    -- instantiation du composant ROM
  generic(dec:integer:= ;width:integer:= ;addr:integer:= );
  port(data_in:in unsigned((addr-1) downto 0);
       read_en:std_logic;
       data_out:out std_logic_vector((width-1) downto 0));
 end component;

 component MAC1 is                    -- instantiation du MAC
  generic(nb_bit:integer:= );
  port(x,y: in std_logic_vector(nb_bit-1 downto 0);
       clk,rst:in std_logic;
       z:out std_logic_vector(nb_bit-1 downto 0));
 end component;

 component LUT1 is                    -- instantiation de la LUT
  generic(dec:integer:= ;width:integer:= ;addr1:integer:= );
  port(lut1_in:in std_logic_vector(addr1-1) downto 0);
       read_en:std_logic;
       lut1_out:out std_logic_vector((width-1) downto 0));
 end component;

begin
ROM1 : ROM1 port map (data_in=>in1, read_en=>ready, data_out=>w);
MAC1 :MAC1 port map(x=>w, y=>in2, clk=>clk, rst=>rst, z=>v);
RESULT: LUT1 port map(lut1_in=>v, read_en=>read_en1,lut1_out=>out_n1);
end;
```

Figure IV.8. Pseudo-code VHDL du circuit neurone.

```
entity couche1 is
generic(nb_bit:integer := ; addr:integer := ),  -- taille du mot;
                                                -- nombre d'addresse
port(in1:unsigned(addr-1 downto 0);
     in2:in std_logic_vector(nb_bit-1 downto 0);
     clk,rst,ready,read_cn1 : in std_logic;
     out1,out2,out3,.....outn: out std_logic_vector(nb_bit-1 downto 0));
end couche1_P;

architecture couche1_P of couche1_P is
component N1                          -- instanliation du composant neurone N1
generic(nb_bit:integer := ;addr:integer := );
port(in1 : unsigned(addr-1 downto 0);
     in2 : in std_logic_vector((nb_bit -1) downto 0);
     clk,rst,ready,read_cn1 :in std_logic;
     out : out std_logic_vector((nb_bit -1) downto 0));
end component;

component N2
generic(nb_bit:integer := ;addr:integer := );
port(in1 : unsigned(addr-1 downto 0);
     in2 : in std_logic_vector(nb_bit -1) downto 0);
     clk,rst,ready,read_cn1 :in std_logic;
     out : out std_logic_vector((nb_bit -1) downto 0));
end component;


component Nn                          -- instanliation du n-ième neurone Nn
generic(nb_bit:integer := ;addr:integer := );
port(in1 : unsigned(addr-1 downto 0);
     in2 : in std_logic_vector(nb_bit -1) downto 0);
     clk,rst,ready,read_cn1 :in std_logic;
     out : out std_logic_vector((nb_bit -1) downto 0));
end component;

begin
N1 : N1 port map(in1 =>in1,in2=>in2, clk=>clk,rst=>rst,
                 ready=>ready,read_cn1=>read_cn1,out=>out1);

N2 : N2 port map(in1=>in1,in2=>in2, clk=>clk,rst=>rst,
                 ready=>ready,read_cn1=>read_cn1,out_n=>out2);


Nn : Nn port map(in1=>in1,in2=>in2, clk=>clk,rst=>rst,
                 ready=>ready,read_cn1=>read_cn1,out=>outn);
end;
```

Figure IV.9 Pseudo-code VHDL de la couche d'un réseau de neurone.

```
entity Reseau is                          -- Description d'un réseau à deux couches
 generic (nb_bit:integer:=..;addr1:integer:=..;
        addr2:integer:=..;               -- taille du mot, taille des mémoires
                                          -- 1re et 2ème couche
        nd_sel1:integer:=..;nd_sel2:integer:=..;..);  -- taille des multiplexeurs
                                                 1re et 2ème couche
 port(X1,X2,X3,X4,X5,X6,........: in std_logic_vector(nb_bit-1 downto 0);
        ad1: unsigned(addr1-1 downto 0);
        ad2: unsigned(addr2-1 downto 0);
        s1: in std_logic_vector(nd_sel1-1 downto 0);
        s2: in std_logic_vector(nd_sel2-1 downto 0);
      clk1,rst,ready,read_en: in std_logic;
        out1,out2,....outn: out std_logic_vector(nb_bit-1 downto 0));
end Reseau;

Architecture structurelle of Reseau is

 component couche_mux1 is              -- instantiation du composant
                                          couche1+multiplexeur1
   generic(nb_bit:integer:=..;addr:integer:=..;nd_sel:integer:=..);
   port(P1,P2,P3,P4,P5,P6,.....: in std_logic_vector(nb_bit-1 downto 0);
        in1_C: unsigned(addr-1 downto 0);
        in2_C: in std_logic_vector(nd_sel-1 downto 0);
        clk1,rst,ready,read_en: in std_logic;
        out_lay1,out_lay2,......: out std_logic_vector(nb_bit-1 downto 0));
end component;

component couche_mux2 is               -- instantiation du composant
                                          couche2+multiplexeur2
generic(nb_bit:integer:=..;addr:integer:=..;nd_sel:integer:=..);
port(P1,P2,P3,P4,.......: in std_logic_vector(nb_bit-1 downto 0);
     in1_C: unsigned(addr-1 downto 0);
     in2_C: in std_logic_vector(nd_sel-1 downto 0);
     clk,rst,ready,read_en: in std_logic;
     out_lay1,out_lay2,...,out_layn: out std_logic_vector(nb_bit-1 downto 0));
end component;

begin
R1: couche_mux1 port map(P1=>X1,P2=>X2,P3=>X3,P4=>X4,P5=>X5,
                  P6=>X6,P[illegible]......,
                  in2_C=>s1,clk1=>clk,rst=>rst,ready=>ready,
                  in1_C=>ad1,read_en=>read_en);.......
R2: couche_mux2 port map(P1=>W1,P2=>W2,P3=>W3,
                          P4=>W4,......,in2_C=>s2,
                          clk=>clk,rst=>rst,ready=>ready,
                          read_en=>read_en,in1_C=>ad2,out_lay1=>out1,
                          ......,out_layn=>[illegible]);
end;
```

Figure IV.10. Pseudo-code VHDL d'un réseau de neurones à trois couches

IV.5.4 Mapping du VHDL dans le FPGA

Durant cette phase, les différentes descriptions VHDL sont synthétisées par GALILEO [52] qui est un outil de synthèse logique dédié aux circuits FPGAs.
GALILEO estime la surface des circuits en termes de bloc logiques configurables (CLB).
Pour notre part, nous avons choisi de cibler la famille XC4000 car elle permet une haute densité d'intégration et possède de meilleures ressources logiques. Le résultat de la synthèse est un fichier « netlist » qui sera utilisé par l'outil de placement et routage XACT de XILINX [53] afin de réaliser les connexions nécessaires dans la matrice FPGA.

IV.5.5 Résultats

Dans cette section, nous présentons les résultats de la synthèse pour les trois classificateurs ***RN1_P***, ***RN1_QRS*** et ***RN2***.

Pour chaque classificateur, la couche d'entrée n'effectue pas les calculs, mais sert uniquement au transfert de données.

Dans tout ce qui suit, nous avons fixé un codage des données sur 10 bits correspondant à une précision de deux chiffres après la virgule, ce qui est largement suffisant pour notre application.

Chaque composant MAC réalise la fonction

$$Y_j = \sum_i W_{ji} X_i$$

pour réaliser cette opération, nous avons utilisé le multiplieur à coefficient constant qui a développé par A. Oudjida [56] et auquel nous avons ajouté le composant accumulateur (voir description du MAC, ANNEXE 3).

Puisque les données sont codées sur 10 bits, les sorties de la couche intermédiaire doivent être représentées sur 20 bits et ceux de la couche de sorties sur 30 bits.
Afin de réduire l'espace mémoire généré par les composant MAC et LUT, nous avons procédé à un scalling « troncature » de 10 bits à la sortie de chaque MAC (voir description MAC, ANNEXE 3).

Notons enfin que cette façon de procéder permet de minimiser la surface globale du réseau de neurones.

La synthèse de chaque réseau de neurones se fait de façon hiérarchique comme signalé dans la section IV.5.3.

Le tableau IV.1 présente les principaux résultats de synthèse en terme de CLBs, obtenus pour les trois classificateurs ***RN1_P***, ***RN1_QRS*** et ***RN2***.

Tableau IV.1. Résultat de la synthèse

CLASSIFICATEUR	*RN1_P*	*RN1_QRS*	*RN2*
Nombre de CLBs	523	550	800

La Figures IV.11 est une représentation structurelle du réseau RN1_P et la Figure IV.12 montre le résultat du mapping de ce dernier dans le circuit la famille FPGA XC4000. Pour la clarté de l'image, nous avons omis le routage.

Des résultats semblables sont obtenus pour le classificateur RN1_QRS et RN2. La différence réside dans la taille du circuit FPGA.

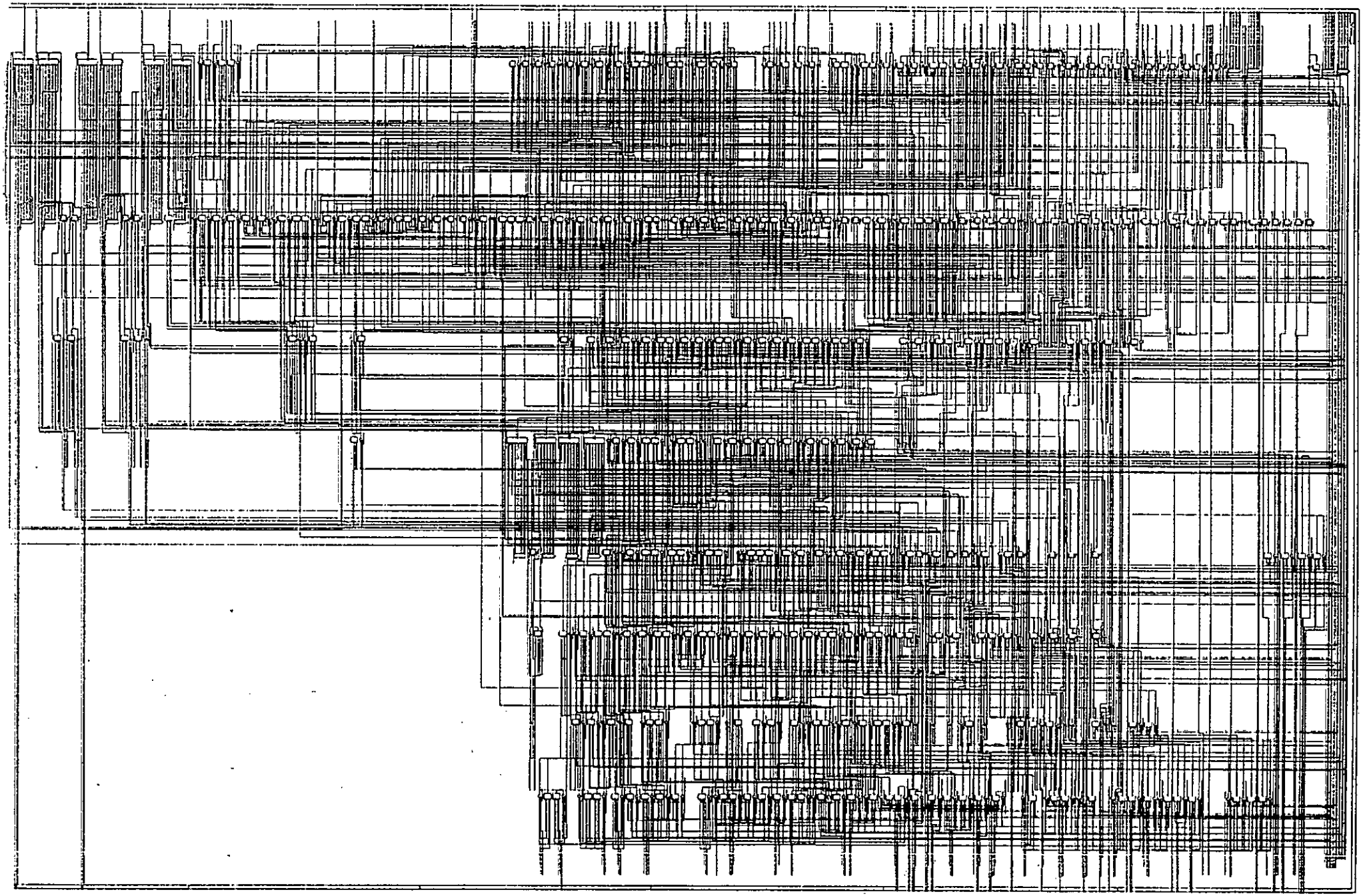


Figure IV.12

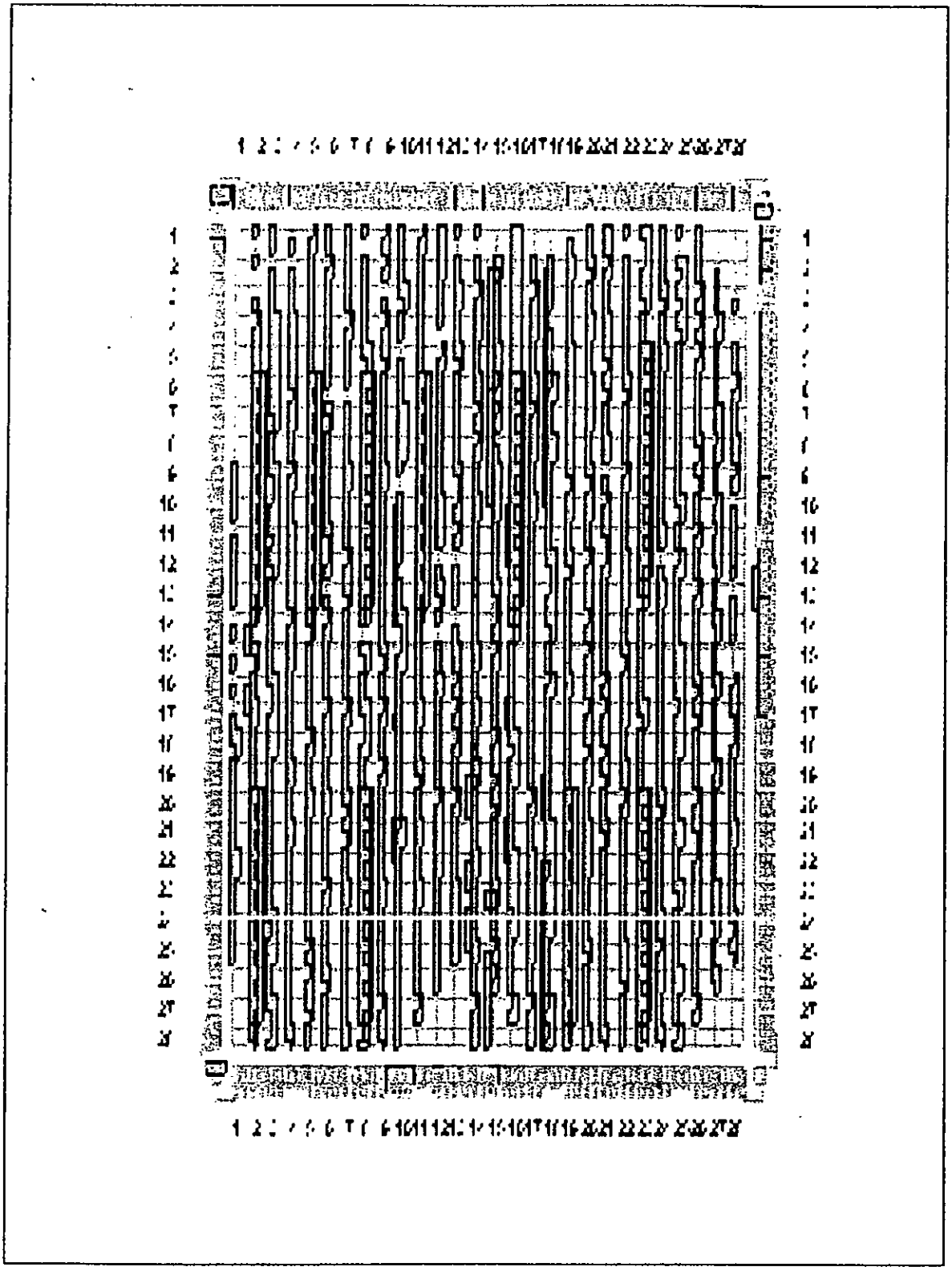


Figure IV.12. Résultat du mapping du VHDL dans le circuit FPGA-XC4000 : classificateur de morphologie *RNI_P*.

IV.6 Conclusion

Dans ce chapitre, nous avons proposé une méthodologie pour l'implémentation digitale des réseaux de neurones, basée sur la synthèse et en utilisant le langage de description VHDL.
Les réseaux de neurones sont des circuits complexes et un temps énorme est perdu dans la conception de ces derniers. L'utilisation d'un outil de synthèse permet de réduire considérablement ce dernier.
Nous avons proposé une architecture FPGA régulière et intégrant un haut degré de parallélisme.

Nous avons proposé une approche de description VHDL flexible et paramétrique des réseaux de neurones permettant, par de simples modifications, la génération de réseaux de neurones de taille quelconque.

Par la suite, nous avons appliqué cette approche pour l'implémentation des trois classificateurs constituant CNAC, à savoir ***RN1_P***, ***RN1_QRS*** et ***RN2***.

Ces circuits ont été synthétisés, en ciblant la famille FPGA-XC4000.

Nous avons effectué pour chaque réseau, une synthèse permettant d'optimiser la surface du circuit. Ce choix est bien justifié du fait que le cycle du signal cardiaque est assez lent (0.8sec/ cycle) et par conséquent ne nécessitant pas des performances en vitesse.

Les réseaux RN1_P, RN1_QRS et RN2 occupent chacun un seul FPGA.

Ces trois circuits constituent le classificateur des arythmies cardiaques, CNAC.

CONCLUSION GENERALE

Le travail effectué dans le cadre de cette thèse se rapporte à la conception et l'implémentation en technologie FPGA (Field Programmable Gate Array) d'un classificateur neuronal des arythmies cardiaques.

Dans un premier temps, une étude de synthèse sur l'aspect clinique des arythmies cardiaques d'une part et d'autre part sur les approches développées dans la littérature pour la classification des arythmies en question, nous ont permis de ressortir les points suivants :

1. La classification des arythmies cardiaques doit prendre en charge la morphologie du complexe QRS, la morphologie de l'onde P et les caractéristiques temporelles du signal cardiaque à savoir l'intervalle PP, l'intervalle PR et l'intervalle RR.
2. Les réseaux de neurones sont très appropriés pour la classification du signal cardiaque, vu les principales propriétés qu'ils présentent à savoir :
   - Habilité d'apprentissage et d'auto organisation à partir des exemples
   - Possibilité de lire des formes en réponse à de nouvelles formes non apprises.

Ces propriétés semblent être très attrayantes pour l'étude du signal cardiaque. En effet, le problème majeur de ce dernier réside dans sa variation au cours du temps pour un même individu et entre plusieurs individus (un signal qui est normal pour une personne pourrait ne pas l'être pour une autre personne).

Cette étude, nous a amené à proposer une nouvelle approche pour la classification des arythmies cardiaques suivantes :

Rythme sinusal normal (**SN**), tachycardie sinusale (**TS**), bradycardie sinusale (**BS**), Extrasytoles auriculaires (**ESE**) tachycardie auriculaire (**TA**), flutter auriculaire (**FA**), fibrillation auriculaire (**FibA**), Extrasystoles ventriculaire (**ESV**), tachycardie ventriculaire (**TV**), fibrillation ventriculaire (**FibV**), flutter ventriculaire (**FV**) et la tachycardie supraventriculaire (**TSV**).

A la différence des autres approches que nous avons étudié, notre approche prend en considération la forme du signal représentant l'onde P, la forme du signal représentant le complexe QRS ainsi que les caractéristiques temporelles PP, PR et RR du signal cardiaque.

Une évaluation globale du système a donné un taux de classification de 93% pour le classificateur de morphologie de l'onde P, ***RN1_P***, un taux de 86% pour le classificateur de morphologie du complexe QRS, ***RN1_QRS***, et un taux de 76% pour le classificateur temporel, ***RN2***.

Ces résultats se justifient par les faits suivants :

D'une part l'onde P présente une forme simple et régulière en comparaison avec le complexe QRS et d'autre part, les classificateurs RN1_P et RN1_QRS sont conçus pour séparer entre deux classes (anormal/normal) ; alors que RN2 à pour tâche la séparation de 12 classes (12 arythmies).

En réalité, le problème de la séparation des classes constitue l'un des sujets de recherche qui n'a pas encore trouvé une solution définitive.

En ce qui concerne l'implémentation FPGA du CNAC, nous pouvons dire que le choix d'une architecture modulaire, parallèle et régulière ; l'utilisation d'une approche descendante de conception basée sur la synthèse de haut niveau et l'application du langage de description VHDL ont permis l'implémentation sur FPGA des trois classificateurs ***RN1_P*** , ***RN1_QRS*** et ***RN2***.
De plus en fixant comme objectif l'optimisation de la surface, nous avons réussi, grâce à un choix judicieux des composants constituant le réseau de neurones, et à l'utilisation de l'outil de synthèse, à implémentér chaque classificateur sur uniquement 1seul FPGA.

L'originalité de notre travail est la proposition d'une description VHDL paramétrée des réseaux de neurones, permettant par de simples modifications, la génération de réseaux de neurones de taille quelconque. Les réseaux ainsi décrits, peuvent être réutilisé pour couvrir un large domaine d'applications et avec d'autres performances.

Enfin, nous pouvons dire que les premiers résultats obtenus tant au niveau software que hardware, montrent l'efficacité des approches proposées, toutefois il est nécessaire d'utiliser par la suite une base de donné standard et de tester tout le système intégré, afin de comparer les résultats obtenus avec d'autres travaux de la littérature.

BIBLIOGRAPHIES

[1] W.S. McCulloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bull. Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[2] D. O. Hebb, "The Organization of Behavior" *John Wiley & Sons*, New York, 1949.

[3] R. Rosenblatt, "Principles of Neurodynamics", *Spartan books*, New York, 1962.

[4] B. Widrow, R. Winter," Neural Nets for Adaptive Filtering and Adaptive pattern Recognition", *IEEE Computer*, vol. 21, no. 3, pp. 25-39, 1988.

[5] M. Minsky and S. Papert," Perceptrons: An Introduction to computational geometry ", *MIT Press*, Cambridge, Mass., 1969.

[6] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Science*, USA 79, vol.79, pp. 2554-2558, 1982.

[7] G.E Hinton, T.G. Senjnowski, D.H. Ackley, "Botzman Machines: Constraint Satisfaction Networks that Learn," *Technical report* no. CMU-CS-84-119, Carnegie-Mellon Univ., Pittsburgh, 1984.

[8] P. Werbos, "Beyond Regression: New tools for Prediction and Analysis in the Behavioral Sciences," *Ph.D. thesis*, Dept. of Applied Mathematics, Harvard University, Cambridge, Mass.1974

[9] D. Rumelhart, G. Hinton, R. Williams, "Learning Internal Representation by Error Propagation," *Parallel distributed processing, MIT Press*, pp.318-362, 1986

[10] D. W. Patterson, "Artificial Neural Networks," Theory and Applications," *Prentice Hall*, 1996.

[11] J. A. Freeman, D. M. Skapura, "Neural Networks Algorithms, Applications, and Programming Techniques," *CNS, Addison-Wesley Publishing Company*, July 1992.

[12] W. Schiffman, M. Jost, R. Werner, "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons", --- September, 1994.

[13] A. K. Jain, Mao, K.M. Mohiuddin, "Artificial Neural Networks: A tutorial", *IEEE Computer*, pp.31-44, March 1996.

[14] A. Cichocki, R. Unbehauen, "Neural Networks for Optimization and Signal Processing "*Jhon Willy & Sons*, 1994.

[15] U. Ramacher, U.Ruckert, " VLSI Design of Neural Networks," *Kluwer Academic Publishers*, 1991.

[16] P. Ienne, G. Kuhn, «Digital Systems for Neural Networks, » *SPIE*, Vol CR57, pp. 314-345, 1995.

[17] C. Jutten, A. Guerin, "Emulateurs de Réseaux Neuronaux: Vers un système de C.A.O d'architectures neuronales," *Journées d'électronique 1989. Réseaux de neurones artificiels*, pp. 207-221, octobre 1989.

[18] M. I. Elmasry, " VLSI Artificial Neural Networks Engineering," *Kluwer Academic Publishers*, 1994.

[19] C E. Cox and W. E Blanz, "GANGLION- A Fast Field Programmable Gate Array Implementation of a Connectionist Classifier ", *IEEE JSSC*, Vol. 27, No. 3, pp. 288-299, March 1992.

[20] H. Ossoing, «Design and FPGA- Implementation of a Neural Network», *ICSPAT'96*, pp. 939-943.

[21] N. M. Botros, M. Abdul-Aziz, "Hardware Implementation of an Artificial Neural Network", *ICSPAT'96*, pp. 1252-1256 .

[22] J. L. Willems, "A Review of ECG Analysis: Time to Evaluate and Standardise" *CRC critical reviews in Medical Informatics*. Vol. 1, No. 2, pp. 165-207, 1987.

[23] R. P. Lippman, « An Introduction to Computing with Neural Nets », *IEEE ASSP Magazine*, pp. 4 -22. April 1987.

[24] F. Vallet, "Reconnaissance de Signaux par Réseaux de Neurones: Aspect Algorithmiques," *Neurosciences et Sciences de l'Ingénieur (NSI 91)* , Mai 1991.

[25] P. Gallinari, S. Fogelman, F. Oulie, "Progressive Design of M.L.P Architecture", *Neuro-Nîmes*, pp. 171-182, 1988.

[26] J.Gay, M. Desnos, P. Benoit, " L'Electrocardiogramme Savoir l'interpreter, " *Editions Frison Roche*, Paris 1990.

[27] M. J. Goldman, "Electrocardiographie clinique," *Editions Vigot*, 1977.

[28] N. Izeboudjen, " Conception et Réalisation d'un Système dédié à la classification des Anomalies Cardiaques par l'approche des Réseaux de neurones", *Rapport de Recherche*, CDTA, Alger, Mars 1997.

[29] H. V. Pipberger, D. McCaughan, D. Littman, "Clinical Application of a Second Generation Electrocardiographic Computer Program," *Am. J. Cardiol.* pp. 535-597, 1975.

[30] A. Bellil, "Etude et Réalisation d'un Classificateur basé sur la transformé de Walsh pour la Détection en Temps Réel d'Anomalies Cardiaques", *Thèse de Magister*, USTHB, 1989.

[31] P. Trahanias, E. Skordalakis,"Syntactic Pattern Recognition of the ECG", *IEEE Trans. On Pattern Analysis and Machine Intelligence*, Vol.12, no. 7, pp. 648-657, July. 1990.

[32] A. Nasri, "Approche Ascendante pour la Reconnaissance Syntaxique du Complexe QRS: Mise en œuvre et 2valuation", *Thèse de Magister*, ENP 1994.

[33] Y. Suzuki, "Self-Organization QRS-Wave recognition in ECG Using Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 6, No. 6, pp. 1469-1477, November 1995.

[34] J. C. Doue, "The Role of Artificial Intelligence in standardizing ECG Criteria," *Computer ECG Analysis: towards standardization, Eds. North Holland*, Amsterdam, 1986.

[35] R. Degani, G. Pacini, "Fuzzy Classification of electrocardiograms," *in Optimization of Computer ECG Processing, Eds. North- Holland*, Amsterdam, 1980.

[36] P. H. W. Leong, M. A. Jabri, "A Low Power VLSI Arrhythmia Classifier," *IEEE Trans. On Neural Networks*, Vol. 6, No. 6, pp. 1435- 1445, November 1996.

[37] P C. Voukydis, "An Artificial Neural Network System for Classification of the Cardiac Rhythm," *ICSPAT*, pp. 248-252, October. 1995.

[38] **N. Izeboudjen, A. Farah,** " A New Neural Network System for arrhythmia's Classification," *NC'98, International ICSC/IFAC Symposium on neural Computation.* Vienna, September 23-25, pp. 208-212.

[39] **N. Izeboudjen, A. Farah, H. Boumeridja, S. Titri,** " Software and Hardware Implementation of a Neural Network for Arrhythmia's Classification", *Conference on Soft Computing and Applications, CSCA '99* , Alger (à paraître).

[40] J. H. Van Bemmel, M. A. Musen ,"Medical Handbook", http://*www.mieur.nl/handbook*

[41] M. Trimberger, "Field Programmable Gate Array technology," *Kluwer Academic Publishers*, 1995.

[42] S. D. Brown, R. J. Francis, Z. G. Vranesic, " Field -Programmable Gate Arrays", *Kluwer Academic Publishers*, 1997.

[43] R. Murgai, R. K. Brayton, A Sangiovanni-Vincentelli, " Logic Synthesis for Field-Programmable Gate Arrays," *Kluwer Academic Publishers*, 1995.

[44] G. R. Goslin, "Using FPGAs in Digital Signal Processing Applications", *ICSPAT'95*, pp. 145 -149.

[45] J. M. Moreno Arostegui, J. Cabestany, E. Canto, J. Faura, J.M Insenser, " The Role of Dynamic Reconfiguration for Implementing Artificial Neural Networks Models in Programmable Hardware", *IWANN'99*, pp. 85-94.

[46] D. Gajski, N. Dutt, A. Wu, S. Lin' "High-Level Synthesis" *kluwer Academic Publishers*, 1997.

[47] R. Airiau, J. M. Berge, V. Olive, J. Rouillard, "VHDL du Language a la Modélisation," *Presses Polytechniques et Universitaires Romandes et CNEST- ENST*, 1990.

[48] S. Mazo, P. Langstrat, " A Guide To VHDL" *Kluwer Academic Publisher*, 1997.

[49] R. Airiau, J. M. Berge, V. Olive, "Circuit Synthesis with VHDL," *Kluwer Academic Publishers*, 1994.

[50] **N. Izeboudjen, A. Farah, S. Titri, H. Boumeridja**, "Artificial Neural Networks Synthesis: A VHDL Methodology for FPGA Implementation", *Actes de la deuxième Conférence Internationale d'Electronique sur les Signaux, Systèmes et Automatique, Blida, Alger, SSA2'99*. Vol. 2, pp. 455-460, Mai 1999.

[51] **N. Izeboudjen , A. Farah, S. Titri, H. Boumeridja,** " Digital Implementation of Artificial Neural Networks: From VHDL Description to FPGA Implementation", *Lecture Notes in Computer Science, Inernational Work on Artificial Neural Networks (IWANN'99)-* , *Springer Verlag Editor*, Vol. 2, pp. 139-148, Juin 1999.

[52] GALILEO HDL, *Synthesis Manual, Exemplar Logic.*

[53] XACT, *User manual.*

[54] P. Trealeven, M. Pacheco and M. Vellasco, " VLSI Architectures for Neural Networks", *IEEE MICRO*, pp. 8-27, December 1989.

[55] M. S. ben Romdhane, V. K. Madisetti, Jhon W. Hines, "Quick turnaround ASIC Design in VHDL, Core-based behavioral Synthesis," *Kluwer Academic Publishers*, 1996.

[56] A. K. Oudjida, "High Speed and Very Compact Two's Complement Serial/Parlle Multipliers Using FPGA, " *ICSPAT'95*, Boston, pp. 145-149, October 1995.

ANNEXES

ANNEXE 1

/********** Programme d'apprentissage **********/
/********** Rétropropagation du gradient - 3 couches *******/
/********** ON-LINE LEARNING **********/

```
#include <math.h>;
#include <stdio.h>;
#include <windowsx.h>
#define MAXI 10
#define MAXJ 4
#define MAXK 1
#define MAXN 1000
#define eta 0.5
#define alpha 0.1
#define epsi 0.01

int i,j,k;
float xQ[MAXI];//vecteur d'entree
float xQV[MAXI];
float yQ[MAXK];//vecteur de sortie
float xQ1[MAXI];//vecteur d'entrée dupliqué,
float xQV1[MAXI];
float yQ1[MAXK];//vecteur de sortie dupliqué,
float wjiQ1[MAXJ][MAXI];//poids synaptiques couche ji dupliquée
float wkjQ1[MAXK][MAXJ];//poids synaptiques couche kj dupliquée
float out_pjQ[MAXJ]; // sorties de la première couche (exemple Q,couche j)
float net_pjQV[MAXJ];
float out_pjQV[MAXJ];
float net_pjQ[MAXJ]; // valeurs des nets de la couche j, exemple Q
float net_pkQ[MAXK]; // valeurs des nets de la couche k, exemple Q
float out_pkQ[MAXK]; // sorties de la deuxième couche k, exemple Q
float net_pkQV[MAXK];
float out_pkQV[MAXK];
float sortie_Q;
FILE *vectinQ;
FILE *vectoutQ;
FILE *resultat;
FILE *wjinQ;
FILE *wkjinQ;

/********** programmes principal ********/
main()
{
resultat=fopen("resultat","w");
 for(j=0;j<MAXJ;j++)
  {
   net_pjQ[j]=0;
  }
 for(j=0;j<MAXJ;j++)
  {
   net_pjQV[j]=0;
  }
 for(k=0;k<MAXK;k++)
  {
   net_pkQV[k]=0;
  }
 for(k=0;k<MAXK;k++)
   {
   net_pkQ[k]=0;
   }
 propagation_QRS();
 fclose(resultat);
}
propagation_QRS()
{
saisie_xQ();          //appel de fonction saisie du vecteur d'entrée xq -exemple1
saisie_yQ();          //appel de fonction saisie du vecteur de sortie yq --exemple1
saisie_wjiQ();        //appel de fonction initialisation des poids synaptiques couchej→couche i
saisie_wkjQ();        //appel de fonction initialisation des poids synaptiques couche k→couche i
forward_propQ();  // appel de fonction forward propagation -exemple1
```

```
erreur_propQ(); // appel de fonction clacul d'erreur –exemple1
impression_WQ(); // appel de fonction impression des poids synaptiques après apprentissage-exemple1
entree_2(); //appel de fonction saisie (lecture) du vecteur d'entrée exemple2
sortie_2(); // appel de fonction saisie du vecteur de sortie correspondante–exemple 2
forward_2(); // appel de fonction forward propagation –exemple2
erreur_2(); //appel de fonction clacul d'erreur –exemple2
impression_2(); // appel de fonction impression des poids synaptiques après apprentissage-exemple2
entree_3(); //appel de fonction saisie (lecture) du vecteur d'entrée -exemple 3
sortie_3(); // appel de fonction saisie du vecteur de sortie correspondante–exemple 3
forward_3(); //appel de fonction clacul d'erreur –exemple 3
erreur_3();//appel de fonction clacul d'erreur –exemple 3
impression_3(); // appel de fonction impression des poids synaptiques après apprentissage- exemple 3
entree_4(); //appel de fonction .....exemple 4
sortie_4(); //
 forward_4();
erreur_4();
impression_4();
entree_5(); //appel de fonction.....exemple5
sortie_5();
forward_5();
erreur_5();
impression_5();
entree_6(); //.appel de fonction.....exemple 6
sortie_6();
forward_6();
erreur_6();
impression_6();
entree_7(); //appel de fonction.....exemple 7
sortie_7();
forward_7();
erreur_7();
impression_7();
entree_8(); //appel de fonction.....exemple 8
sortie_8();
forward_8();
erreur_8();
impression_8();
}
/************lecture des exemples d'apprentissage *********/
/*************************************************************/
saisie_xQ()
{
//char fichier_exempleQ[20];
printf("\n");
printf("\n APPRENTISSAGE DU RESEAU: ALGORITHME BACK-PROPAGATION ONDE QRS");
printf("\n");
  if((vectinQ=fopen("onde_Q.txt","r"))==NULL)
   {
    printf("\n lecture de l'exemple Q %d:");
            for(i=0;i<MAXI;i++)
              {
printf("\ndonnez le vecteur d'entrée xQ[%d]:",i);
        scanf("%f",&xQ[i]);
              }
    vectinQ=fopen("onde_Q.txt","w");
    fwrite(&xQ,sizeof(xQ),1,vectinQ);
    fclose(vectinQ);
   }
    for(i=0;i<MAXI;i++)
     {
      printf("\nxQ[%d]: %f",i,xQ[i]);
     }
    vectinQ=fopen("onde_Q.txt","r");
    fread(xQ1,sizeof(xQ1),1,vectinQ);
    fclose(vectinQ);
//getch();
   for(i=0;i<MAXI;i++)
    {
     printf("\nxQ1[%d]: %f",i,xQ1[i]);
    }
}
/*************Saisie du vecteur sortie desirée *********/
saisie_yQ()
{
char fichier_sortieQ[21];
```

```
  if((vectoutQ=fopen("sortie_Q.txt","r"))==NULL)
   {
    printf("\n lecture de la sortie Q %d:");
     for(k=0;k<MAXK;k++)
       {
       printf("\ndonnez la sortie correspondante yQ[%d]:",k);
       scanf("%f",&yQ[k]);
            }
     vectoutQ=fopen("sortie_Q.txt","w");
     fwrite(&yQ,sizeof(yQ),1,vectoutQ);
     fclose(vectoutQ);
   }
     for(k=0;k<MAXK;k++)
      {
      printf("\nyQ[%d]: %f",k,yQ[k]);
      }
     vectoutQ=fopen("sortie_Q.txt","r");
     fread(yQ1,sizeof(yQ1),1,vectoutQ);
     fclose(vectoutQ);
 for(k=0;k<MAXK;k++)
    {
    printf("\nyQ1[%d]: %f",k,yQ1[k]);
    }
}
/***************Initialisation des Poids Synaptiques Couche ji-************/
/*****************************************************************************
saisie_wjiQ()
{
 char s[20];
printf("\n INITIALISATION DES POIDS SYNAPTIQUES ONDE QRS");
printf("\n");
    wjinQ=fopen("WJI_P.txt","r");
    for(j=0;j<MAXJ;j++)
    {
              for(i=0;i<MAXI;i++)
              {
        //printf("\n I don't worry");
                 fscanf(wjinQ, "%s",&s);
        fscanf(wjinQ, "%f",&wjiQ1[j][i]);
        printf("\n wjiQ[%d][%d] = %f ",j,i, wjiQ1[j][i]);
              }
    }
fclose(wjinQ);
 for(j=0;j<MAXJ;j++)
    {
     for(i=0;i<MAXI;i++)
     {
      fprintf(resultat,"\n wjiQ[%d][%d]= %f",j,i,wjiQ1[j][i]);
     }
    }
getch();
}
/***************Initialisation des poids synaptiques Couche k-j*****/
saisie_wkjQ()
{
 char s[20];
   wkjinQ=fopen("WKJ_P.txt","r");
    for(k=0;k<MAXK;k++)
    {
              for(j=0;j<MAXJ;j++)
              {
      fscanf(wkjinQ, "%s",&s);
       fscanf(wkjinQ, "%f",&wkjQ1[k][j]);
       printf("\n wkjQ[%d][%d] = %f ",k,j, wkjQ1[k][j]);
      }
    }
 fclose(wkjinQ);
}
   for(k=0;k<MAXK;k++)
    {
    for(j=0;j<MAXJ;j++)
              {
       fprintf(resultat,"\n wkjQ[%d][%d]= %f",k,j,wkjQ1[k][j]);
              }
    }
}
```

```
}
 ************************Forward Propagation************/
 ********************************************************/
forward_propQ()
{
printf("\n APPRENTISSAGE DU RESEAU: ALGORITME BACK-PROPAGATION ONDE QRS") ;
printf("\n") ;
 **********************couche 1********/
  for(j=0;j<MAXJ;j++)
   {
    for(i=0;i<MAXI;i++)
            {
       net_pjQ[j]=net_pjQ[j]+(wjiQ1[j][i] * xQ1[i]);
            }
       printf("\nvaleur du netpjQ[%d]:%f",j,net_pjQ[j]);
       out_pjQ[j]=1/(1+exp(-net_pjQ[j]));
       printf("\nla sortie out_pjQ[%d]:%f",j,out_pjQ[j]);
}
 **********************couche 2**********/
  for(k=0;k<MAXK;k++)
   {
    for(j=0;j<MAXJ;j++)
          {
      net_pkQ[k]=net_pkQ[k]+(wkjQ1[k][j] * out_pjQ[j]);
     }

    printf("\n la valeur du netpkQ[%d]:%f",k,net_pkQ[k]);
    out_pkQ[k]=1/(1+exp(- net_pkQ[k]));
    printf("\n la sortie out_pkQ[%d]:%f",k,out_pkQ[k]);
    return(out_pkQ[k]);
          //getch();
   }
}
 *****************Calcul d'erreur***************/
erreur_propQ()
{
int n;
float ETQ1;
float ETQ;
float delta_kQ[MAXK];
float delta1_jQ[MAXJ];
float delta_jQ[MAXJ];
float delta_wkjQ[MAXK][MAXJ]; //erreur globale sur.....
float delta_wjpQ1[MAXP]; //erreur sur les poids de la couche ji.exple
float delta_wjiQ[MAXJ][MAXI]; //erreur sur les poids de la couche ji
ETQ1=0;
for(k=0;k<MAXK;k++)
{
  for(j=0;j<MAXJ;j++)
  {
   delta_wkjQ[k][j]=0;
  }
}
for(j=0;j<MAXJ;j++)
{
 for(i=0;i<MAXI;i++)
  {
   delta_wjiQ[j][i]=0;
  }
}
  for (k=0;k<MAXK;k++)
   {
    ETQ1=ETQ1+((yQ1[k]-out_pkQ[k])*(yQ1[k]-out_pkQ[k]));
    printf("\n Erreur ETQ1:%f",ETQ1);
   }
   ETQ=0.5*ETQ1;
printf(resultat."\n  Erreur totale ETQ: %f",ETQ);

 ********************************************************
  for(n=0;n<MAXN && ETQ>epsi;n++)
   {
    for(j=0;j<MAXJ;j++)
     {
      delta1_jQ[j]=0;
     }
```

```
   /************* erreur à la sortie du reseau  *********/
    for(k=0;k<MAXK;k++)
     {
     delta_kQ[k]=out_pkQ[k]*(1-out_pkQ[k])*(yQ1[k]-out_pkQ[k]);
     printf("\n l'erreur ... la sortie delta_kQ[%d]:%f",k,delta_kQ[k]);
//getch();
     }
   /************** erreur ...à la 2ème couche *************/
    for(j=0;j<MAXJ;j--)
     {
      for(k=0;k<MAXK;k++)
       {
        delta1_jQ[j]= delta1_jQ[j]+(delta_kQ[k]*wkjQ1[k][j]);
       }
     delta_jQ[j]=delta1_jQ[j]*out_pjQ[j]*(1-out_pjQ[j]);
     printf("\n l'erreur ... la 2Śme couche delta_jQ[%d]:%f",j,delta_jQ[j]);
     }

   /********* calcul d'erreur sur les poids synaptiques *********/

   /******** erreur des poids de la couche_kj ***********
   for(k=0;k<MAXK;k--)
   {
    for(j=0;j<MAXJ;j--)
            {
       delta_wkjQ[k][j]=eta*delta_kQ[k]*out_pjQ[j]+alpha*delta_wkjQ[k][j];
            printf("\n erreur globale sur les poids delta_wkjQ[%d][%d]:%f",k,j,delta_wkjQ[k][j]);
            wkjQ1[k][j]=wkjQ1[k][j]+ delta_wkjQ[k][j];
            printf("\n nouvelle valeur des poids wkjQ1[%d][%d]:%f",k,j,wkjQ1[k][j]);
            }
   }

    /****** erreur des poids de la couche_ji *******/
    for(j=0;j<MAXJ;j--)
    {
            for(i=0;i<MAXI;i++)
             {
       delta_wjiQ[j][i]=eta*delta_jQ[j]*xQ1[i]+alpha*delta_wjiQ[j][i];
             printf("\n erreur globale sur les poids delta_wjiQ[%d][%d]:%f",j,i,delta_wjiQ[j][i]);
             wjiQ1[j][i]=wjiQ1[j][i]+delta_wjiQ[j][i];
             printf("\n nouvelle valeur des poids wjiQ1[%d][%d]:%f",j,i,wjiQ1[j][i]);
             }
    }

//***************** calcul de la nouvelle sortie **********/
           for(k=0;k<MAXK;k++)
            {
      net_pkQ[k]=0;
            }
  for(j=0; j<MAXJ;j++)
   {
   net_pjQ[j]=0;
   }
   for(j=0;j<MAXJ;j++)
    {
     for(i=0;i<MAXI;i++)
      {
      net_pjQ[j]=net_pjQ[j]+(wjiQ1[j][i]*xQ1[i]);
      }
     out_pjQ[j]=1/(1+exp(-net_pjQ[j]));
     printf("\n la nouvelle sortie out_pjQ[%d]:%f",j,out_pjQ[j]);
     };
            for(k=0;k<MAXK;k++)
             {
              for(j=0;j<MAXJ;j++)
               {
        net_pkQ[k]=net_pkQ[k]+(wkjQ1[k][j]*out_pjQ[j]);
               }
        out_pkQ[k]=1.(1+exp(-net_pkQ[k]));
        printf("\n la nouvelle sortie out_pkQ[%d]:%f",k,out_pkQ[k]);
               }
/****************** calcul de la nouvelle erreur **********/
ETQ1=0;
    for (k=0;k<MAXK;k++)
     {
     ETQ1=ETQ1+((yQ1[k]-out_pkQ[k])*(yQ1[k]-out_pkQ[k]));
```

```
        printf("\n nouvelle Erreur ETQ1:%f",ETQ1);
        }
//getch();
ETQ=0.5*ETQ1;
fprintf(resultat,"\n %d",n+1);
fprintf(resultat," %f",ETQ);
}
 if(ETQ<epsi )
  {
   fprintf(resultat,"\n convergence à l'iteration:%d", n);
  }
 else
 {
 fprintf(resultat,"\n pas de convergence à l'itération:%d",n);
 }
}
//*******************************************************
impression_WQ()
{
    for(j=0;j<MAXJ;j++)
     {
      for(i=0;i<MAXI;i++)
       {
        fprintf(resultat,"\n wjiQ[%d][%d]:%f",j,i,wjiQ1[j][i]);
       }
     }
   for(k=0;k<MAXK;k++)
    {
     for(j=0;j<MAXJ;j++)
            {
       fprintf(resultat,"\n wkjQ[%d][%d]:%f",k,j,wkjQ1[k][j]);
            }
    }
}
getch();
```

ANNEXE 2

/********** Programme de Test********** /
/********** Algorithme de la rétropropagation du gradient **********/

```
#include <math.h>;
#include <stdio.h>;
#define MAXI 5
#define MAXJ 10
#define MAXK 1
int i,j,k;

float xQV[MAXI];
float xQV1[MAXI];
float wjiQ[MAXJ][MAXI]; //poids synaptiques couche ij,exemple P
//float wjiQ1[MAXJ][MAXI];//poids synaptiques couche ij dupliques
float wkjQ[MAXK][MAXJ]; //poids synaptiques couche jk,exemple P
//float wkjQ1[MAXK][MAXJ];//poids synaptiques couche jk dupliques
float net_pjQV[MAXJ];
float out_pjQV[MAXJ];
float net_pkQV[MAXK];
float out_pkQV[MAXK];
float sortie_Q;
FILE *vectinQV;
FILE *wjinQ;
FILE *wkjinQ;
FILE *test;
/********** programmes principal ********/
main()
{
test=fopen("test.txt", "w");
 for(j=0;j<MAXJ;j++)
  {
   net_pjQV[j]=0;
  }
 for(k=0;k<MAXK;k++)
  {
   net_pkQV[k]=0;
  }
propagation( );
fclose(test);
}
propagation( )
{
lecture_Q( );
saisie_wjiQ( );
saisie_wkjQ( );
generalisation_Q( );
}
/********** TEST/Généralisation sur apprentissage *******/
/**********lecture de nouveaux exemples *********/
lecture_Q( )
{
char fichier_exemple[20];
printf("\n ");
printf("\n ****** GENERALISATION SUR APPRENTISSAGE ONDE QRS ******");
//printf("\n entrer le nom du fichier ... vérifier %d:");
//scanf("%s",fichier_exemple);
  if((vectinQV=fopen("fichier_exemple.dat","r"))==NULL)
   {
            for(i=0;i<MAXI;i++)
             {
printf("\ndonnez le vecteur ... vérifier xQV[%d]:",i);
              scanf("%f",&xQV[i]);
             }
  vectinQV=fopen("fichier_exemple.dat","w");
  fwrite(&xQV,sizeof(xQV),1,vectinQV);
   }
  fclose(vectinQV);
   for(i=0;i<MAXI;i++)
    {
     printf("\nxQV[%d]: %f",i,xQV[i]);
    }
```

```
    vectinQV=fopen("fichier_exemple.dat","r");
    fread(xQV1,sizeof(xQV1),1,vectinQV);
    fclose(vectinQV);
   for(i=0;i<MAXI;i++)
     {
      printf("\nxQV1[%d]: %f",i,xQV1[i]);
     }
}
/*********************************************/
saisie_wjiQ( )
{
 char s[20];
printf("\n lecture des poids synaptiques onde QRS");
printf("\n");
    wjinQ=fopen("WJI_P.txt","r");
    for(j=0;j<MAXJ;j++)
     {
               for(i=0;i<MAXI;i++)
                {
                  fscanf(wjinQ, "%s",&s);
         fscanf(wjinQ, "%f",&wjiQ[j][i]);
         printf("\n wjiQ[%d][%d] = %f ",j,i, wjiQ[j][i]);
                }
     }
 fclose(wjinQ);
  for(j=0; j<MAXJ;j++)
     {
      for(i=0;i<MAXI;i++)
       {
       fprintf(test,"\n wjiQ[%d][%d]= %f",j,i,wjiQ[j][i]);
       }
     }
getch();
}
/*************** lecture des poids synaptiques Couche k-j*****/
saisie_wkjQ( )
{
 char s[20];
   wkjinQ=fopen("WKJ_P.txt","r");
    for(k=0;k<MAXK;k++)
     {
               for(j=0;j<MAXJ;j++)
                {
         fscanf(wkjinQ, "%s",&s);
         fscanf(wkjinQ, "%f",&wkjQ[k][j]);
         printf("\n wkjQ[%d][%d] = %f ",k,j, wkjQ[k][j]);
       }
     }
  fclose(wkjinQ);{
   for(k=0;k<MAXK;k++)
    {
     for(j=0;j<MAXJ;j++)
               {
        fprintf(test,"\n wkjQ[%d][%d]= %f",k,j,wkjQ[k][j]);
               }
    }
}
}
/*************************************************************/
generalisation_Q( )
{
/********************** couche 1*******/
  for(j=0;j<MAXJ;j++)
   {
    for(i=0;i<MAXI;i++)
            {
             net_pjQV[j]=net_pjQV[j]+(wjiQ[j][i] * xQV1[i]);
            }
printf("\nvaleur du netp_jQV[%d] après généralisation:%f\n",j,net_pjQV[j]);
   getch();
   out_pjQV[j]= 1/(1+exp(-net_pjQV[j]));
printf("\nla sortie out_pjQV[%d] après généralisation:%f\n",j,out_pjQV[j]);
   getch();
   }
  getch();
```

```
/**********************couche 2**********/
  for(k=0;k<MAXK;k++)
   {
    for(j=0;j<MAXJ;j++)
            {
             net_pkQV[k]=net_pkQV[k]+(wkjQ[k][j] * out_pjQV[j]);
            }
            printf("\n la valeur du netpkQV[%d] après
g.n.réalisation:%f\n",k,net_pkQV[k]);
            out_pkQV[k]=1/(1+exp(-net_pkQV[k]));
fprintf(test,"\n la sortie out_pQkV[%d] après g.n.réalisation:%f",k,out_pkQV[k]);
            getch();
   }
if(sortie_QRS<0.15)
 {
   fprintf(test,"\n complexe QRS de morphologie anormale (N)");
 }
else
 {
   fprintf(test,"\n complexe QRS normale");
 }
}
getch();
```

ANNEXE B

1. Synthese VHDL d'un MAC

-- Description VHDL du MAC

```
library ieee;
use ieee.std_logic_1164.all;
library exemplar;
use exemplar.exemplar.all;

entity MAC1_P is
 generic(nb_bit:integer:=10);
 port(x,y: in std_logic_vector(nb_bit-1 downto 0);
     clk,rst:in std_logic;
     z:out std_logic_vector(nb_bit-1 downto 0));
end MAC1_P;

architecture structural of MAC1_P is

component multiplier
 generic(N:integer:=10);
   port( A,B :in std_logic_vector(N downto 1);
       P : out std_logic_vector(2*N downto 1));
end component;

component acum1_P
 generic (nb_bit:integer:=10);
  port(d:in std_logic_vector((2*nb_bit-1) downto 0);
    clk:in std_logic;
    rst:in std_logic;
    q:out std_logic_vector( (nb_bit-1) downto 0));
 end component;
signal d1: std_logic_vector((2*nb_bit-1) downto 0);

begin
  mult: multiplier generic map(N=>10)
            port map(A=>x, B=>y, P=>d1);
  acc:acum1_P generic map(nb_bit=>10)
    port map(d=>d1, clk=>clk, rst=>rst, q=>z);
  end;
```

-- Description de l'accumulateur

```
library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_signed.all;
library exemplar;
use exemplar.exemplar_1164.all;
entity acum1_P is
 generic (nb_bit:integer:=10);
 port(d:in std_logic_vector((2*nb_bit-1) downto 0);
    clk:in std_logic;
    rst:in std_logic;
    q:out std_logic_vector( (nb_bit-1) downto 0));
 end acum1_P;

architecture behav of acum1_P is
 signal qi,q1: std_logic_vector((2*(nb_bit)-1) downto 0);
 begin
    acum:process(clk,rst,d)
     begin
      if (rst='1') then
       qi <= "00000000000000000000";
       elsif (clk='1' and clk'event) then
         qi <= (qi) + (d);
       end if;
    end process acum;
 q1<=sr(qi,10);
 q <=extend(q1,10);
end;
```

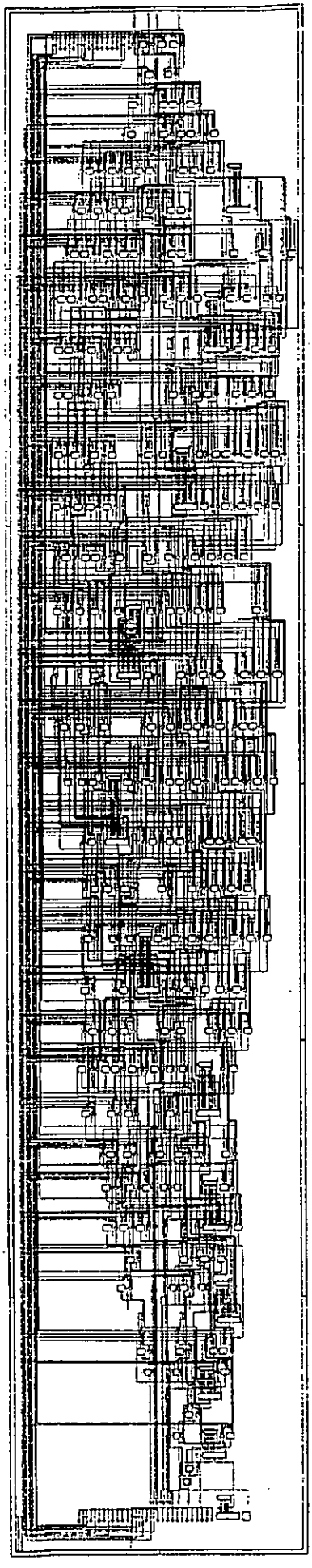
-- **Résultat de la synthèse du MAC**

```
*****************************************
**************

Cell: mac1_p_10   View: structural   Library:
work

*****************************************
**************

Total accumulated area :
 Number of FG Function Generators :   200
 Number of H Function Generators :     9
 Number of Packed CLBs :             104
 Number of CLB Flip Flops :           20
```



ANNEXE 3

2. Synthèse VHDL d'une ROM

-- **Description VHDL de la ROM**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;

entity ROM1_P is
generic (deep:integer:=16;width :integer:=10;addr:integer:=4);
port(data_in:in unsigned((addr-1) downto 0);
    read_en:std_logic;
    data_out:out std_logic_vector((width-1) downto 0));
end ROM1_P;

architecture ROM1_P of ROM1_P is

type ROM is array(deep-1 downto 0) of std_logic_vector((width-1) downto 0);
constant ROM_data:ROM:=("1111111011",-- -0.05
            "1111010100",-- -0.44
            "1110110000",-- -0.80
            "1101110110",-- -1.38
            "1101101101",-- -1.47
            "1101101000",-- -1.52
            "1101111111",-- -1.29
            "1110110100",-- -0.76
            "1111101101",-- -0.19
            "1111110110",-- -0.10
    others=>("----------"));
begin
  process(data_in,read_en)
    begin
     if read_en= '1' then
          data_out<= ROM_data(conv_integer(data_in));
          else
        data_out<= (others =>'Z');
     end if;
   end process;
end;
```

-- Résultats de synthèse de la ROM

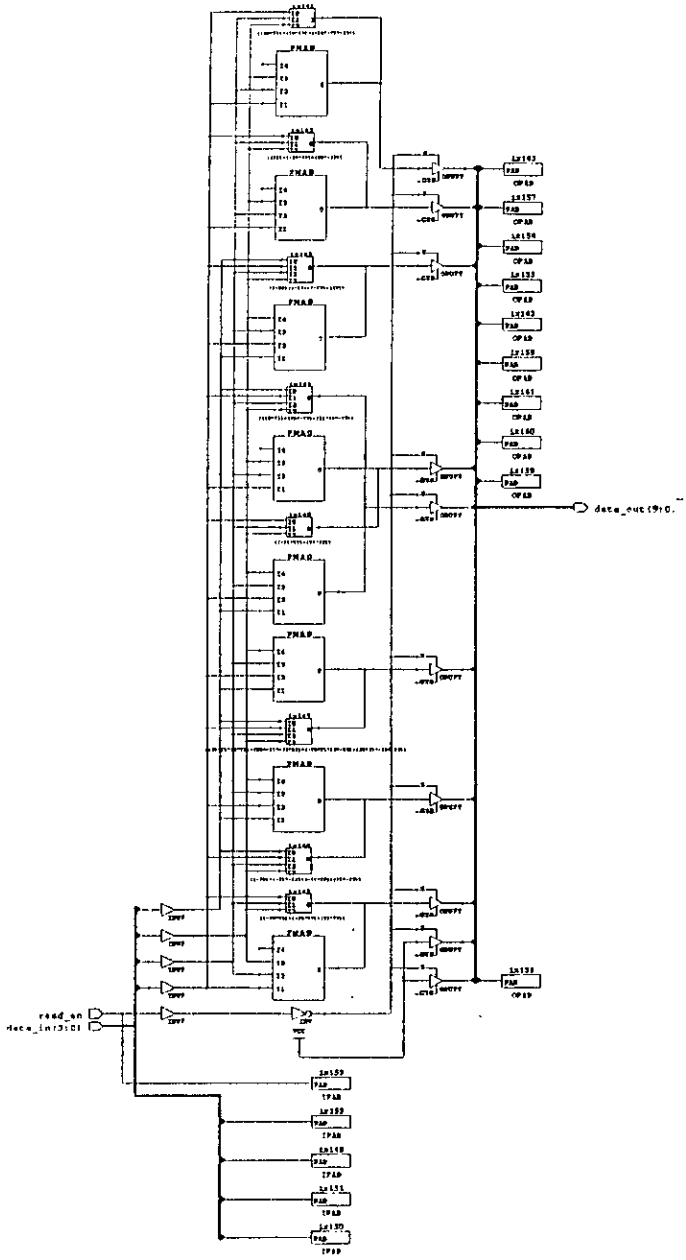
```
**************************************
*******************

Cell: rom1_p_16_10_4   View: rom1_p
Library: work

**************************************
*******************

Total accumulated area :
 Number of FG Function Generators :    8
 Number of Packed CLBs :               4
 Number of IBUF :                      5
 Number of OBUFT :                     10

 Number of ports :                     15
 Number of nets :                      30
```



ANNEXE B

3. Synthèse VHDL d'une LUT

-- **Description VHDL de la LUT**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
--use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
--library exemplar;
--use exemplar.exemplar.all;


entity lut1_P is
generic (deep:integer:=1024;width:integer:=10;addr1:integer:=10);
port(lut1_in:in std_logic_vector((addr1-1) downto 0);
     read_en:std_logic;
     lut1_out:out std_logic_vector((width-1) downto 0));
end lut1_P;

architecture LUT1_P of LUT1_P is
subtype word is std_logic_vector((width-1) downto 0);
type LUT is array(deep-1 downto 0) of word;
constant LUT_data:LUT:=("0000000000","0000000000","0000000000",
                        "0000000000","0000000000",
                        "0000000001", -- 0.0
                        "0000000100", --0.04
                        "0000001011", --0.11
                        "0000011010", --0.26
                        "0000110010", --0.5
                        "0001001001", --0.73
                        "0001011000", --0.88
                        "0001011111", --0.95
                        "0001100010", --0.98
                        "0001100011", --0.99
                        "0001100011", --0.99
                        "0001100011","0001100011","0001100011",
                        "0001100011","0001100011",
                        "0001100011","0001100011","0001100011","0001100011","0001100011",
                        "0001100011","0001100011","0001100011","0001100011",
                        others=>("0001100100")
                       );

begin
  process(lut1_in,read_en)
    begin
      if read_en= '1' then

            lut1_out<= LUT_data(conv_integer(lut1_in));

       else
          lut1_out<= (others =>'Z');
      end if;
   end process;
end;
```

Résultat de synthèse de la LUT

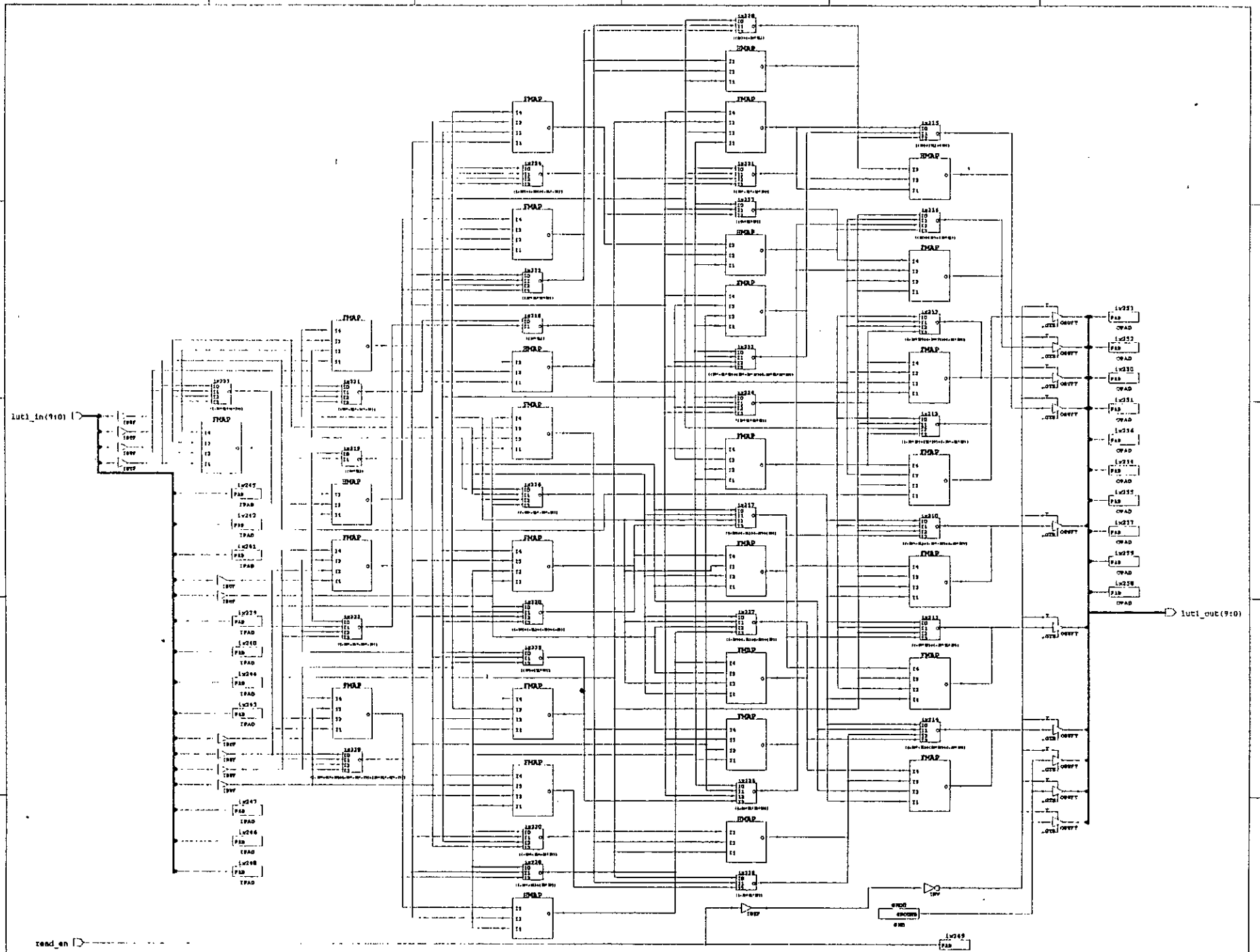
```
*********************************************************

Cell: lut1_p_1024_10_10   View: lut1_p   Library: work

*********************************************************

Total accumulated area :
 Number of FG Function Generators :   22
 Number of H Function Generators :    7
 Number of Packed CLBs :             11
 Number of IBUF :                    11
 Number of OBUFT :                   10

 Number of ports :                   21
 Number of nets :                    63
```


lut1_in(9:0)
read_en
lut1_out(9:0)
lut1_p_1024_10_10