

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE



المدرسة الوطنية المتعددة التقنيات  
Ecole Nationale Polytechnique

ECOLE NATIONALE POLYTECHNIQUE  
DEPARTEMENT ELECTRONIQUE  
LABORATOIRE SIGNAL ET COMMUNICATIONS

# Thèse de Doctorat

## En Électronique

Présentée par  
**IZEBOUDJEN NOUMA**  
Magister en électronique à l'ENP

Intitulé

# Plateforme pour l'Implémentation des Réseaux de Neurones sur FPGA : Application à l'Algorithme de la Rétro Propagation du Gradient (RPG)

Soutenue publiquement le 09 Avril 2014 devant le jury composé de :

<b>Président :</b> GUERTI MHANIA	PROFESSEUR	ENP
Directeur de thèse : FARAH AHCENE	PROFESSEUR	ENSTA
Co-directeur de thèse: BESSALAH HAMID	MAITRE DE RECHERCHES/A	CDTA
Examineur: BABA ALI RIAD	MAITRE DE CONFERENCES/A	USTHB
Examineur: LARBES CHERIF	PROFESSEUR	ENP
Examineur: TOUHAMI RACHIDA	PROFESSEUR	USTHB

**ENP 2014**

## ملخص

إن العمل المنجز في إطار هذه الأطروحة يتعلق بتصميم منصة لتنفيذ شبكات الأجهزة العصبية الاصطناعية المبرمجة على FPGA. في صلب موضوعنا خوارزمية الانتشار الرجعي للخطاء RPG. ولقد كرسنا الفصل الأول من هذا العمل في عرض لمحة عامة عن الشبكات العصبية وبالأخص PERCEPTRON المتعدد الطبقات المستند إلى خوارزمية (RPG). في الفصل الثاني أجرينا دراسة عن درجة ما وصلت إليه الأبحاث في ما يخص تنفيذ أجهزة الشبكات العصبية ، وقد أدى ذلك بنا إلى اقتراح مقاربة جديدة لتصنيف الدوائر العصبية تبدأ من الأجهزة القياسية وتنتهي إلى الدوائر والأنظمة المدمجة على الشرائح. في الفصل الثالث، أجرينا دراسة مختلف الجوانب لتنفيذ خوارزمية RPG على FPGA ، أي ودرجة التوازي ، واختيار لغة وصف الأجهزة و تقييم الأداء من البنية المقترحة. وأولينا اهتماما خاصا بتأثير اختيار دائرة المضاعف على أداء الشبكة العصبية؛ من حيث المساحة وسرعة التنفيذ. كما أخذنا بعين الاعتبار الجوانب ذات الصلة باختيار عائلة FPGA و كذا المسألة المتعلقة بكثافة التكامل. في الفصل الرابع، طبقنا البرمجة الديناميكية على خوارزمية. وقد تم إجراء دراسة مقارنة لثلاثة أساليب في البرمجة الديناميكية ل FPGA: برمجة ثابتة، برمجة ديناميكية شاملة وبرمجة محلية. في الفصل الخامس، نقترح منهجية جديدة لتصميم الشبكات العصبية الاصطناعية، مبنية على أساس تطبيق مفهوم إعادة الاستخدام للتصميم design reuse. المنهجية المقترحة تشكل العمود الفقري لبناء منصة تجمع كل التقنيات والقدرات المرتبطة بتنفيذ FPGA على شبكات الأجهزة العصبية.

**الكلمات الرئيسية:** الشبكات العصبية، منصة، خوارزمية الانتشار الرجعي للخطاء، عائلة FPGA ، البرمجة الديناميكية ، إعادة الاستخدام، والتوازي

## Résumé

Le travail effectué dans le cadre de cette thèse se rapporte à la conception d'une plateforme pour l'implémentation hardware des réseaux de neurones sur FPGA, plus particulièrement l'algorithme de la rétro propagation du gradient (RPG). Les réseaux de neurones étant au cœur de notre sujet, nous avons consacré le premier chapitre à une présentation générale des réseaux de neurones, en particulier le perceptron multicouche basé sur l'algorithme RPG. Dans le deuxième chapitre, nous avons effectué une étude sur l'état de l'art concernant l'implémentation hardware des réseaux de neurones. Ce qui nous a conduits à proposer une nouvelle approche de classification du hardware neuronal allant des circuits standards jusqu'aux circuits et systèmes sur puce. Dans le chapitre III, nous avons effectué une étude des différents aspects liés à l'implémentation hardware de l'algorithme RPG sur FPGA, à savoir, l'apprentissage, le degré de parallélisme, le choix du langage de description matérielle ainsi que l'évaluation des performances de l'architecture proposée. Nous avons consacré une attention particulière quand à l'influence du choix du multiplieur sur les performances du réseau de neurones ; en terme de surface et de temps d'exécution. Aussi, dans ce chapitre, nous avons pris en considération les aspects liés au choix de la famille des circuits FPGAs et le problème lié à la densité d'intégration. Dans le chapitre IV, nous avons appliqué la reconfiguration dynamique à l'algorithme RPG. Pour cela, nous avons effectué une étude comparative des trois approches de reconfiguration dynamiques, à savoir la reconfiguration statique, la reconfiguration dynamique globale et la reconfiguration dynamique locale. Dans le chapitre V, nous avons proposé une nouvelle méthodologie pour l'implémentation hardware des réseaux de neurones, basée sur l'application du concept de réutilisation « design reuse ». La méthodologie proposée constitue l'épine dorsale pour la construction d'une plateforme permettant de regrouper l'ensemble des techniques et moyens liés à l'implémentation sur FPGA des réseaux de neurones.

**Mots clés:** Réseaux de neurones, Plateforme, rétropropagation du gradient, FPGA, reconfiguration dynamique, réutilisation, parallélisme

## Abstract

The work presented in the context of this thesis concerns the design of a platform for the hardware implementation of neural networks into FPGAs, in particular the back propagation algorithm (RPG). Neural networks are at the heart of our subject, we have devoted the first chapter to an overview of neural networks, especially the multilayer perceptron based on the RPG algorithm. In the second chapter, we conducted a study on the state of the art on the hardware implementation of neural networks. This has led us to propose a new classification approach for neural hardware ranging from standard chips to embedded circuits and systems on chip. In Chapter III, we conducted a study of various aspects of hardware implementation of the RPG algorithm on FPGA, namely the learning procedure, the degree of parallelism, the choice of the hardware description language and the performance evaluation of the proposed architecture. We devoted a special attention regarding the influence of the multiplier type on the performance of the neural network; in terms of area and execution time. Also, in this chapter we have considered the aspects related to the choice of the FPGA family circuits and the problem related to the integration density. In Chapter IV, we applied the dynamic reconfiguration algorithm to the RPG algorithm. For this, we performed a comparative study of three approaches, namely the static reconfiguration, the global dynamic reconfiguration and the local dynamic reconfiguration. In Chapter V, we have proposed a new methodology for hardware implementation of neural networks, based on the application of the design reuse concept. The proposed methodology is the backbone for the construction of a platform to bring the set of all the techniques and capabilities associated with the implementation of neural networks into FPGAs.

**Keywords:** Neural Networks, Platform, backpropagation, FPGA, dynamic reconfiguration, design reuse, parallelism

## *AVANT PROPOS*

Le travail présenté dans cette thèse de Doctorat, préparée à l'Ecole Nationale Polytechnique d'Alger (ENP), a été effectué au sein de la Division Microélectronique et Nanotechnologies (DMN) du Centre de Développement des Technologies Avancées (CDTA), en collaboration avec le département Computer Engineering de la Queen's University of Belfast (UK), à la faveur d'une bourse de finalisation de thèse de doctorat accordée par le Ministère de l'Enseignement Supérieur et de la Recherche Scientifique (MESRS) et entrant dans le cadre du programme national exceptionnel de formation à l'étranger ( PNE ).

## Dédicaces

*A la mémoire de ma mère, que dieu l'accueille  
dans son vaste paradis*

*A mon père, pour l'éducation qu'il m'a donné  
et ses sacrifices tout au long de la vie*

*A tous ceux qui me sont chers*

*Je dédie le fruit de ce modeste travail*

## ***REMERCIEMENTS***

Tout d'abord, je tiens à remercier Madame MHANIA GHERTI d'avoir accepté la présidence de mon jury de thèse, ainsi que Madame RACHIDA TOUHAMI, Messieurs RIAD BABA ALI et CHERIF LARBES qui m'ont fait l'honneur de faire partie de mon jury de thèse.

Je remercie tout particulièrement Monsieur AHCENE FARAH de m'avoir encadré durant cette thèse de doctorat, ainsi que mes co-encadreurs, en Algérie, Monsieur HAMID BESSALAH et à l'étranger, Monsieur AHMED BOURIDENE. Je les remercie tous, pour leur aide, leurs précieux conseils, leur disponibilité ainsi que les grandes valeurs humaines que j'ai trouvé en eux.

Je tiens à remercier tous mes collègues de travail pour leur soutien moral.

J'adresse aussi mes remerciements à ma famille pour sa patience et ses encouragements à finaliser ce travail.

# SOMMAIRE

## SOMMAIRE

### LISTE DES FIGURES

### LISTE DES TABLEAUX

### ABREVIATIONS

## INTRODUCTION GENERALE.....1

## CHAPITRE I LES RESEAUX DE NEURONES : PRINCIPES ET DEFINITIONS

I.1 Introduction .....	5
I.2 Historique .....	5
I.3 Fondements biologiques des réseaux de neurones artificiels.....	6
I.3.1 Le système nerveux.....	7
I.3.2 Physiologie du neurone.....	8
I.3.2.1 Description du neurone.....	8
I.3.2.2 Caractéristiques des neurones.....	10
I.3.2.3 Fonctionnement du neurone .....	11
I.3.3 Organisation des neurones dans le système nerveux central .....	12
I.4 Réseaux de neurones artificiels.....	14
I.4.1 Le neurone artificiel.....	14
1.4.1.1 Première génération : les modèles du neurone à fonction échelon.....	15
1.4.1.2 Seconde génération : Le neurone à fonction d'activation continue....	16
1.4.1.3 Troisième génération : Le neurone impulsionnel.....	16
I.4.2 Construction des réseaux de neurones.....	17
I.4.2.1 Architecture des réseaux de neurones.....	17
I.4.2.2 Apprentissage des réseaux de neurones .....	19
I.4.2.2.1 Apprentissage supervisé.....	19
I.4.2.2.2 Apprentissage renforcé.....	19
I.4.2.2.3 Apprentissage non supervisé.....	20
I.5 Les règles d'apprentissage.....	21
I.5.1. La règle de Hebb.....	21
I.5.2 La règle d'apprentissage du Perceptron.....	21
I.5.3 La règle de Widrow-Hoff ou règle delta.....	22
I.5.4 La règle delta généralisé.....	22
I.6 Le perceptron multicouches et l'algorithme de la rétropropagation du gradient.....	22
I.7 Propriété des réseaux de neurones.....	26
I.8 Domaines d'applications des réseaux de neurones.....	26
I.9 Conclusion.....	27

## CHAPITRE II APERCU SUR L'IMPLEMENTATION HARDWARE DES RESEAUX DE NEURONES : ETAT DE L'ART ET PERSPECTIVES

II. 1 Introduction.....	28
II.2 Architecture hardware de base des réseaux de neurones.....	29
II.2.1 Architecture hardware de base du neurone.....	29
II.2.2 Les mesures de performances .....	30

II.3	Etat de l'art sur les différents types d'implémentation du hardware neuronal.....	31
II.3.1	Les différentes approches de classification du hardware neuronal .....	31
II.3.1.1	Classification selon NÖRDSTROM .....	32
II.3.1.2	Classification selon AARON FERRUCCI .....	33
II.3.1.3	Classification selon GLESNER .....	34
II.3.1.4	Classification selon PAOLO IENNE .....	34
II.3.1.5	Classification selon HEEMSKERK .....	35
II.3.1.6	Classification selon ISIK AYBAY .....	36
II.3.1.7	Classification selon BEIU .....	37
II.3.1.8	Classification selon T. SHCOENAUER .....	38
II.3.1.9	Classification selon SORIN DRAGHICI.....	39
II.3.1.10	Classification selon Clark Lindsey .....	40
II.3.1.11	Classification selon KRISTIAN NICHOLS .....	41
II.3.1.12	Classification selon VIPAN KAKKAR.....	42
II.3.1.13	Classification selon SAUMIL G. MERCHANT.....	43
II.3.2	Synthèse des différentes approches de classification .....	44
II.3.3	Proposition d'une approche pour la classification du hardware neuronal .....	45
II.3.4	Description des différents types d'implémentation du hardware neuronal .....	48
II.3.4.1	Implémentation du hardware neuronal sur des chips standard .....	48
II.3.4.1.1	les cartes accélératrices .....	48
II.3.4.1.2	Implémentations sur les machines parallèles.....	49
II.3.4.2	Implémentation du hardware neuronal sur les chips VLSI.....	52
3.4.2.1	Implémentation sur circuit ASIC analogique .....	53
II.3.4.2.2	Implémentation sur circuit ASIC digital .....	55
II.3.4.2.3	Implémentation sur circuit ASIC hybride .....	58
II.3.4.3	Implémentation sur circuit FPGA .....	60
II.3.4.3.1	Utilisation de la configuration et la reconfiguration des circuits FPGAs .....	60
II.3.4.3.2	Utilisation du circuit FPGA comme Co-processeur et carte accélératrice.....	63
II.3.4.3.3	Utilisation des descriptions Soft-Core.....	64
II.3.4.4	Utilisation des systèmes embarqué et systèmes sur puce .....	66
II.4	Conclusion .....	69

## **CHAPITRE III IMPLEMENTATION DE L'ALGORITHME RPG SUR FPGA**

III.1	Introduction .....	70
III.2	Rappel sur l'algorithme de rétropropagation du gradient .....	70
III.2.1	Les différentes approches d'implémentation de l'algorithme RPG .....	71
III.3	Etude du parallélisme pour l'implémentation de l'algorithme RPG .....	73
III.3.1	Aperçu des différents types de parallélisme .....	73
III.3.1.1	Parallélisme de session d'apprentissage .....	73
III.3.1.2	Parallélisme d'exemple d'apprentissage .....	73
III.3.1.3	Parallélisme de Couche .....	75
III.3.2	Choix du langage de description matérielle .....	76
III.3.2.1	Le langage VHDL et le RTL .....	76
III.3.2.1.1	Description VHDL du neurone .....	76
III.3.2.1.2	Le bloc des poids synaptiques .....	77
III.3.2.1.3	le bloc MAC .....	77
III.3.2.1.4	Bloc d'activation .....	78
III.3.2.1.5	Unité du contrôle du neurone .....	79

III.3.2.2	Le langage Handel-C .....	81
III.3.2.2.1	Description du neurone en Handel-C .....	82
III.3.2.3	Etude comparative entre le VHDL et le Handel-C .....	83
III.3.3	Le Handel-C et le parallélisme des neurones .....	86
III.3.4	Le Handel-C et le parallélisme des synapses .....	86
III.3.5	Comparaison entre le parallélisme des neurones et le parallélisme des synapses.....	87
III.4	Proposition d'une architecture pour l'implémentation de l'algorithme RPG .....	90
III.4.1	Module de propagation.....	91
III.4.2	Module de calcul d'erreur .....	92
III.4.3	Module de mise à jour des poids synaptiques .....	94
III.4.4	Module de control .....	94
III.5	Evaluation des performances de l'architecture de l'algorithme RPG .....	96
III.5.1	Etude de l'influence du type de multiplieur sur le réseau de neurone.....	96
III.5.2	Influence du choix de la famille des circuits FPGAs sur les performances du réseau de neurones .....	97
III.5.2.1	Implémentation « off chip training » d'un réseau de neurones pour la classification des troubles du rythme cardiaque.....	99
III.5.2.2	Implémentation « on chip training » du XOR logique .....	100
III.5.2.3	Détermination de la dimension maximale du réseau de neurone.....	101
III.6	Conclusion .....	102

## **CHAPITRE IV UTILISATION DE LA RECONFIGURATION DYNAMIQUE POUR L'IMPLEMENTATION DE L'ALGORITHME RPG**

IV.1	Introduction .....	104
IV.2	Concepts de base .....	104
IV.2.1	La configuration statique des FPGA .....	104
IV.2.2	La reconfiguration dynamique des FPGA (RTR) .....	105
IV.2.2.1	La RTR globale .....	105
IV.2.2.2	La RTR locale .....	106
IV.2.3	Mesures de performances .....	107
IV.3	Application de la reconfiguration dynamique à l'algorithme RPG .....	108
IV.3.1	Estimation du nombre de connections du réseau de neurones .....	109
IV.3.2	Estimation du temps d'exécution .....	109
IV.3.3	Estimation de la surface du réseau .....	110
IV.3.4	Configuration statique de l'algorithme RPG .....	112
IV.3.5	Application de la RTR globale à l'algorithme RPG .....	112
IV.3.6	Application de la RTR locale à l'algorithme RPG .....	116
IV.3.6.1	Estimation du temps de reconfiguration et de la taille du fichier ...	118
IV.4	Etude comparative .....	125
IV.5	Conclusion .....	127

## **CHAPITRE V : APPLICATION DES CONCEPTS DE « DESIGN REUSE » POUR L'IMPLEMENTATION DE L'ALGORITHME RPG**

V.I	Introduction .....	129
V.2	Concepts de base .....	132
V.2.1	Règles de conception .....	132
V.2.1.1	Règles de conception pour l'utilisation .....	132



V.2.1.2 Règles de conception pour la réutilisation « Reuse » .....	133
V.2.2 Le modèle en couche .....	133
V.2.3 Le modèle industriel pour le <i>Reuse</i> .....	134
V.2.4 Supports techniques pour le <i>Reuse</i> .....	135
V.2.4.1 Les outils de conception .....	135
V.2.4.2 Le VHDL et le <i>Reuse</i> .....	135
V.2.4.3 Les règles pour le codage .....	136
V.3 Application des règles de conception pour la réutilisation des réseaux de neurones .....	137
V.4 Méthodologie de conception de l’algorithme RPG basée sur les concepts de DFR et DWR .....	137
V.4.1 Stratégie de DFR .....	139
V.4.1.1 Sélection des différents IPs .....	139
V.4.1.1.1 Sélection de l’IP-ANN « off chip training » .....	140
V.4.1.1.2 Sélection de l’IP-ANN « on chip training statique» .....	142
V.4.1.1.3 Sélection de l’IP-ANN « on chip training RTR» .....	142
V.4.1.2 Structure du code VHDL de l’algorithme RPG .....	143
V.4.1.2.1 Description VHDL paramétrée de l’IP-ANN .....	143
V.4.1.2.2 Description VHDL paramétrée de l’IP neurone .....	145
V.4.2 Stratégie de « DWR » .....	145
V.4.2.1 Le multiplieur .....	147
V.4.2.2 L’accumulateur .....	147
V.4.2.3 Le circuit mémoire .....	147
V.4.2.4 La fonction d’activation .....	148
V.5 Estimation du coût de « Design Reuse » .....	148
V.6 Evaluation de la qualité de l’IP-ANN .....	152
V.7 Conclusion .....	158
<b>CONCLUSION GENERALE</b> .....	159
<b>REFERENCES BIBLIOGRAPHIQUES</b> .....	162
<b>ANNEXES</b>	
Annexe-A.....	170
Annexe-B.....	173
Annexe-C.....	185

---

## LISTE DES FIGURES

- Figure I.1 Système nerveux humain
- Figure I.2 Différentes fonctions du système nerveux
- Figure I.3 tissu nerveux : neurones et cellules gliales
- Figure I.4 Anatomie du neurone
- Figure I.5 Différents types de neurones
- Figure I.6 Fonctionnement du neurone et genèse du potentiel d'action
- Figure I.7 Organisation en couche des neurones dans le cortex cérébral
- Figure I.8 Différents types de réseaux
- Figure I.9 Neurone artificiel. (a) représentation schématique (b) modèle mathématique
- Figure I.10 Fonctions linéaire (à gauche) et sigmoïdale (à droite)
- Figure I.11 génération du potentiel d'action dans le modèle du neurone impulsionnel
- Figure I.12 Différentes architectures des réseaux de neurones
- Figure I.13 Principe de l'apprentissage supervisé
- Figure I.14 Principe de l'apprentissage par renforcement
- Figure I.15 Principe de l'apprentissage non supervisé
- Figure I.16 représentation fonctionnelle de l'algorithme de la Rétro-propagation du gradient
- 
- Figure II.1 Architecture de base du neurone artificiel
- Figure II.2 Différents types d'implémentation hardware des réseaux de neurones
- Figure II.3 Différentes approches de classification du hardware neuronal
- Figure II.4. Classification du hardware neuronal selon NÖRDSTROM
- Figure II.5 Classification du hardware neuronal selon Aron Ferruci
- Figure II.6 Classification des neurocalculateurs selon PAOLO IENNE
- Figure II.7 Classification des neurocalculateurs selon HEEMSKERK
- Figure II.8 Classification des neurocalculateurs selon Isik Aybay
- Figure II.9 Classification des réseaux de neurones selon T. Shcoenauer
- Figure II.10 Classification selon SORIN DRGHICI
- Figure II.11 Classification selon *Clark Lindsey*
- Figure II.12 Classification des neurocalculateurs selon KRISTIAN NICHOLS
- Figure II. 13 Classification du hardware neuronal selon VIPAN KAKKAR
- Figure II.14 Classification du hardware neuronal selon SAMUEL G. MERCHANT
- Figure II.15 Proposition d'une nouvelle approche pour la classification du hardware neuronal
- Figure II.16 Implémentation du hardware neuronal à base de standard chips

- 
- Figure II. 17 Architecture de la *Connection Machine*
- Figure II.18 Architecture du système *NETSIM*
- Figure II. 19 Architecture du système IBM-GF11
- Figure II.20 - La machine iPSC/860 et description d'un noeud
- Figure II.21 Architecture d'un neurone analogique
- Figure II.22 Architecture de la carte ETANN-80177NX
- Figure II.23 Principe de la rétine synaptique. (a) Section de la rétine. (b) Model équivalent.
- Figure II.24 (a) Architecture du circuit AN221E04 ANADIGM. (b) Architecture du bloc CAB
- Figure II.25 Architecture du système CNAPS
- Figure II.26 Architecture du circuit neuronal IBM ZISC036
- Figure II.27 Architecture du circuit MANTRA-I (EPFL) et du bloc PE
- Figure II.28 Bloc digramme du circuit MDAC
- Figure II.29 Architecture interne du circuit MDAC
- Figure II.30 (a) Transistor MOS (b) Résistance équivalente (c) L'horloge de contrôle
- Figure II.31 Classification des circuits FPGAs selon leurs configurations
- Figure II.32 Utilisation de la reconfiguration partielle pour l'implémentation de l'algorithme RPG
- Figure II. 33. (a) Reconfiguration partielle dans VIRTEX-II (b) Layout des blocs reconfigurables
- Figure II.34 Architecture de RAPTOR2000
- Figure II. 35 Réseau de neurone impulsionnel sur FPGA. (a) architecture du neurone (b) architecture de la synapse. (c) architecture du Soma
- Figure II.36 Schéma simplifiée du SoftTOTEM NC3003
- Figure II.37 Description de l'approche VNP
- Figure II.38 Architecture du SOC pour le perceptron multicouche
- Figure II.39 Architecture HW/SW pour l'implémentation du MLP
- 
- Figure III.1 Principe de l'approche « off chip training »
- Figure III.2- Pseudo code de l'algorithme RPG
- Figure III.3 Parallélisme de session d'apprentissage
- Figure III.4 Parallélisme d'exemples d'apprentissage
- Figure III.5 Parallélisme de couche
- Figure III.6 Parallélisme de neurones
- Figure III.7 Parallélisme des synapses
- Figure III.8 Architecture du neurone
- Figure III.9 Architecture du bloc de poids synaptique
- Figure III.10 Architecture du bloc MAC
- Figure III.11 Organigramme de contrôle du neurone
- Figure III.12 Pseudo -code VHDL d'un neurone

Figure III.13 Le Handel-C/ANSI-C

Figure III.14 description séquentielle

Figure III.15 Equivalent hardware de la description parallèle: par { ... }

Figure III.16 Pseudo-code d'un neurone en langage Handel-C

Figure III.17 Architecture du circuit Virtex-E

Figure III.18 Layout d'une description VHDL

Figure III.19 Layout d'une description Handel-C

Figure III.20 Pseudo Code Handel-C pour exprimer le parallélisme des neurones

Figure III.21 Pseudo Code Handel-C pour exprimer le parallélisme des synapses.

Figure III.22 Problème de séparation non linéaire du XOR-logique

Figure III. 23 Topologie du réseau de neurone pour résoudre le problème du XOR\_logique

Figure III.24 Architecture générale du circuit VIRTEX-II

Figure III.25 Architecture interne du circuit VIRTEX-II (a) Architecture d'un CLB. (b) Architecture d'un slice

Figure III. 26 Architecture générale de l'algorithme RPG

Figure III. 27 (a) Module de Propagation. (b) Architecture d'une couche. (c) Architecture d'un neurone

Figure III.28 Chronogramme de simulation du module de propagation

Figure III.29. Architecture du module Calcul d'erreur

Figure III.30 Chronogramme de simulation du module de calcul de l'erreur

Figure III.31 Architecture du module de mise à jour des poids synaptiques

Figure III.32 Chronogramme de simulation du module de mise à jour des poids synaptiques

Figure III.33 Organigramme et diagramme d'état du module du control de l'algorithme RPG

Figure III.34 Architecture générale du circuit Virtex-4

Figure III.35 Architecture d'un CLB

Figure III.36 Architecture d'une tuile DSP48 dans le FPGA virtex-4

Figure III. 37 Classificateur des troubles du rythme cardiaque

Figure IV.1 Configuration statique d'un FPGA

Figure IV.2 Reconfiguration dynamique d'un FPGA

Figure IV.3 La RTR globale

Figure IV.4 La RTR locale

Figure IV.5 Différentes étapes de l'algorithme RPG

Figure IV.6 Densité fonctionnelle de la configuration statique en fonction du nombre de neurone

Figure IV.7 Application de la RTR globale à l'algorithme RPG

Figure IV.8 Densité fonctionnelle de la RTR globale en fonction du nombre de neurones

Figure IV.9 Rapport de configuration en fonction du nombre de neurones

Figure IV.10. Rapport de configuration en fonction du nombre d'itérations. (n), le nombre de neurones ou dimension du réseau.

Figure IV.11 Application de la RTR locale à l'algorithme RPG

Figure IV.12 Densité fonctionnelle de la RTR locale en fonction du nombre de neurones

Figure IV.13 Approche de conception modulaire

Figure IV.14 layout du neurone sans application des contraintes physiques

Figure IV.15 Application des contraintes physiques au layout du neurone

Figure IV.16 Layout de l'algorithme RPG Complet : réseau (3,3, 1)

Figure IV.17 Rapport de Configuration en fonction du nombre de neurone

Figure IV.18 Variation du rapport de configuration en fonction du nombre d'itérations

Figure IV.19 Comparaison des densités fonctionnelles : Configuration statique, RTR globale et RTR locale

Figure V.1 Evolution de la loi de Moore et de la productivité

Figure V.2 Modèle en couche de conception

Figure V.3 Modèle Industriel pour le *Reuse* des IP

Figure V.4 Les niveaux de maturité des outils de conception

Figure V.5 Méthodologie de conception de l'algorithme RPG : Application du DFR et DWR

Figure V.6 Structure externe de l'IP\_ANN « Off chip training »

Figure V.7 (a) Structure interne de l'IP ANN. (b) Structure interne de la couche. (c) structure du neurone

Figure V.8 Machine d'état du module de control de l'IP ANN. (a) Type Moore (b) Type Mealy

Figure V.9 Structure Externe de l'IP ANN « On chip training statique»

Figure V.10 Etapes pour l'application de la reconfiguration dynamique. (a) RTR globale (b) RTR locale

Figure V.11 Pseudo code de l'IP ANN

Figure V.12 Pseudo code pour la génération des différentes couches

Figure V.13 Pseudo code de l'IP neurone.

Figure V.14 Structure de la bibliothèque des Cores constituant l'IP neurone.

Figure V.15 Estimation du temps de conception pour l'algorithme RPG « off chip training »

Figure V.16 Estimation du temps de conception pour l'algorithme RPG « On chip training statique »

Figure V.17 Estimation du temps de conception pour l'algorithme RPG « on chip training - RTR»

Figure V.18 Evolution du temps de conception en fonction du nombre de neurones

Figure Annexe B-1 : Organisation VHDL des fichiers pour la description du réseau 2-2-1

Figure Annexe- B-2 Schéma RTL généré : réseau 2-2-1

# LISTE DES TABLEAUX

- Tableau III.1 : Comparaison entre le VHDL et le Handel-C
- Tableau III.2. Table de vérité de la fonction XOR
- Tableau III.3 Comparaison entre les performances du parallélisme des neurones et le parallélisme des synapses
- Tableau III.4 Différents type de multiplieurs
- Tableau III.5 Résultats de synthèse du XOR pour les différents types de multiplieurs
- Tableau III.6 Performances du classificateur des troubles du rythme cardiaque
- Tableau III.7 Performances du réseau XOR logique
- Tableau III.8 Utilisation des ressources/ complexité du réseau
- Tableau IV.1 Les ressources utilisées du circuit FPGA Virtex-II.
- Tableau IV.2. Reconfiguration globale: Nombre d'itérations en fonction de la taille du réseau
- Tableau IV.3 Nombre maximal des « *frames* », taille des « *frames* » et taille du « *Bit-stream* »
- Tableau IV.4 Nombre de « *frames* » des différentes colonnes du circuit Virtex-II
- Tableau IV.5 Estimation du temps de reconfiguration partielle et taille du fichier « *Bit-stream* »
- Tableau IV.6 Reconfiguration partielle : Nombre d'itération en fonction de la taille du réseau
- Tableau IV.7 Comparaison des trois approches proposées pour la configuration et reconfiguration dynamique
- Tableau IV.8 Comparaison de REC-ANN avec RTR-MANN et RRANN-II
- Tableau V.1 Estimation du coût d'une conception avant application du Reuse
- Tableau V.2 Estimation du coût de la conception de l'algorithme RPG en utilisant le concept de Reuse
- Tableau V.3 Estimation du temps DWR
- Tableau V.4 ANN-Open More Assessment program

---

## ABREVIATIONS

ART:	Artificial Resonance Theory
ASIC:	Application Specific Integrated Circuit
CAB:	Configurable Analog Bloc
CC:	Charge Coupled Devices
CLB:	Configurable Logics Blocs
DFR:	Design For Reuse
DSP:	Digital Signal Processor
DWR:	Design With Reuse
EPROM:	Erasable Programmable ROM
FPAA:	Filed Programmable Analog Array
FPGA:	Field Programmable Gate Array
LMS:	Linear Mean Square
LSI:	Large Scale Integration
MCPS :	Millions de Connexions par Seconde
MCUPS:	Millions de Connexion Upadate Par Seconde
MIMD:	Multiple Instruction Multiple Data
MISD :	Multiple Instruction Single Data
MLP :	Multi-Layer Perceptron
NRTR:	Non Run time Reconfiguration
PE:	Processing Element
PU:	Processing Unit
RAM:	Random Access Memory
RBF:	Radial Basis Function
RISC:	Reduced Instruction Set Computer
RPG:	RétroPropagation du Gradient
RTR:	Run Time Reconfiguration
SIMD:	Single Instruction Multiple Data

SISD :	Single instruction Single Data
SNC :	Système Nerveux Central
SNP :	Système Nerveux Périphérique
SNNS:	Stuttgart Neural Network Simulator
SOC:	System on Chip
VHDL:	Very high speed Hardware Description Language
VLSI:	Very Large Scale Integration
WSI:	Wafer scale Integration
ZISC :	Zero Instruction Set Computer



---

# *INTRODUCTION GENERALE*

---

**L'** homme, de part sa curiosité et sa nature à chercher à comprendre et à interpréter les phénomènes pour pouvoir par la suite développer, innover voir même inventer, a toujours été intrigué et fasciné par le fonctionnement de cette complexe « machine intelligente » qu'est le cerveau qu'il soit humain ou animal.

**P**artant de ce principe, la neuroscience qui a pour but d'étudier, d'une part, l'organisation des éléments du système nerveux (en d'autres termes l'architecture du cerveau) et d'autre part, les liens entre ce dernier et le comportement (i.e. la science cognitive), est née. Les retombées épistémologiques et philosophiques de cette dernière sont considérables, particulièrement dans la construction des modèles mathématiques des réseaux de neurones, ce qu'on appelle réseaux de neurones artificiels. Ceux-ci se fondent sur l'utilisation massive d'unités de calculs élémentaires qui acquièrent une connaissance sur le monde extérieur par le biais de profils d'activités distribuées parmi leurs nœuds.

**P**ar leur style de programmation, ces modèles sont entièrement différents des machines dites de « Von Newman », puisque les réseaux de neurones se basent sur des méthodes d'apprentissage. A une entrée donnée, celui-ci apprend à fournir une sortie la mieux adaptée avec l'environnement et l'application traitée. Une fois l'apprentissage réalisé, les méthodes connexionnistes permettent d'effectuer des généralisations qui présentent certaines qualités de résistance au bruit.

**L**es apprentissages qui sont des processus itératifs sont généralement longs à converger et nécessitent beaucoup de puissance de calcul. D'autre part, les résultats de l'apprentissage dépendent d'un grand nombre de paramètres qui peuvent être déterminés seulement expérimentalement, car ils dépendent fortement de la nature de l'application traitée.

**L**e parallélisme est souvent cité dans la recherche en réseau de neurones comme un atout essentiel des modèles connexionnistes. L'architecture de ces derniers autorise un grand nombre de neurones à travailler de façon concurrente. Cet aspect est la condition requise pour l'obtention d'implantations très rapides. Une panoplie d'implantations est proposée dans la littérature allant des cartes accélératrices et circuits DSPs standards jusqu'au neuro-calculateurs à applications spécifiques ou implantation sur des ASICs et des circuits FPGAs.

Chaque type d'implantation possède ses propres avantages et inconvénients par rapport à l'autre. Une connaissance sur l'état de l'art est nécessaire pour pouvoir aborder des travaux de recherche dans le domaine des réseaux de neurones.

**U**n autre paramètre de grande importance est la flexibilité, c'est-à-dire la possibilité d'apporter des modifications sur la dimension du réseau de neurones, ses propriétés architecturales, sa topologie voir même son mode d'apprentissage.

**I**l est connu, que les implantations logicielles sur un ordinateur conventionnel offrent une grande flexibilité à l'utilisateur. Néanmoins, dans ces machines, le parallélisme inhérent aux réseaux de neurones n'est pas exploité car les ordinateurs conventionnels sont basés sur le model de la machine série de Von Newman dans laquelle une seule instruction est exécutée à la fois.

Le problème du choix du support matériel ou logiciel pour l'implémentation des réseaux de neurones et qui a fait l'objet de plusieurs travaux et rapports de recherches, est un problème qui est toujours posé.

Pour soulever ce problème, un consensus dans la communauté scientifique a établi le fait qu'une solution optimale est celle qui prend en considération les performances obtenues en utilisant des implantations hardware spécifiques et la flexibilité obtenue par les outils software et les circuits d'application générale.

Depuis leur introduction dans le commerce au milieu des années 1980s, et grâce aux avancées dans le développement de la technologie microélectronique d'une part et celui des outils de conception (front end et back end), d'autre part, les circuits programmables FPGAs (Field Programmable Gate Arrays) ont progressé de manière évolutionnaire et révolutionnaire.

Le processus d'évolution a permis la réalisation de circuits FPGAs très rapides, présentant une grande densité d'intégration et un meilleur support technique. La révolution concerne l'intégration des multiplieurs de hautes performances, des microprocesseurs et des circuits DSPs. Ceci à eu une incidence directe sur l'implantation sur FPGA des réseaux de neurones et plusieurs travaux ont été entamés dans ce sens.

Contrairement aux circuits ASICS, les FPGAs permettent :

- Un compromis entre la flexibilité des standards chips et les performances des ASICs.
- La reconfiguration qui peut être statique ou dynamique.

Cependant, l'inconvénient majeur des circuits FPGAs est la nécessité d'une programmation au bas niveau pour exploiter entièrement les potentialités des ces derniers.

Les outils de synthèse basés sur les langages de haut niveau (VHDL, VERILOG) sont proposés afin d'établir un lien entre les spécifications de haut niveau et les contraintes matérielles bas niveau du circuit FPGA utilisé. Cependant, le problème qui se pose est que ces outils ne sont pas spécifiques à des algorithmes ou bien des à applications spécifiques aux réseaux de neurones.

Par conséquent, des concepts spéciaux doivent être envisagés pour le développement de plateformes dédiées à la génération automatique des réseaux de neurones.

Vu la diversité des travaux de recherche dans le domaine de l'implémentation hardware des réseaux de neurones, et afin de définir des objectifs qui nous permettent d'insérer notre travail au sein des travaux de recherche à l'échelle internationale, nous avons pris en considération les rapports directifs qui constituent une feuille de route, en anglais-roadmap, pour la communauté scientifique et aussi pour les organismes de financement. Ces rapports décrivent l'état de l'art du domaine et permettent une perception commune par les chercheurs et une vision de la recherche qui sera intéressante, stimulante et bénéfique dans le domaine pour une décennie ou plus.

Les premières traces de ces roadmap, peuvent être trouvés dans le rapport publié en 2000 et enrichi en 2001/2002 sous le nom de *NeuroNetRoadmap* « First Report on the Use of Neural Networks and Other Computational Intelligence Techniques in Intelligent Multimedia Systems », projet inclut dans le programme *ESPRIT-III* de la commission européenne [1].

Le deuxième roadmap a été publié en 2007, sous le nom « *Roadmap for NeuroIT Challenges for the Next Decade* » [2]

À l'issue de cette étude, il en ressort les grandes lignes directives dans le domaine de l'implémentation hardware sur FPGA des réseaux de neurones, à savoir :

- Recherche des performances architecturales sur FPGA
- Prise en considération de la phase d'apprentissage
- Flexibilité : Conception paramétrée et modulaire
- Utilisation des langages de description de haut niveau pour les réseaux de neurones.

À partir de ces recommandations, nous proposons comme solution le développement d'une plateforme pour l'implantation des réseaux de neurones sur FPGA. Nous nous intéressons au perceptron multicouche basé sur l'algorithme de la rétropropagation du gradient (RPG).

La démarche scientifique adoptée dans cette thèse suit les recommandations citées ci-dessus et auxquelles nous apportons une contribution dans la recherche des solutions aux problèmes posés lors de l'implémentation hardware des réseaux de neurones, à savoir :

- Proposition d'une approche pour la classification du hardware neuronal.
- Proposition d'une architecture parallèle pour l'implémentation de l'algorithme de la rétropropagation du gradient et qui prend en charge la phase d'apprentissage « on chip training ». Nous proposons d'introduire le langage Hadel-C pour réaliser le parallélisme. Une étude comparative entre les langages VHDL et Handel-C est effectuée dans ce sens.
- Etude des performances du réseau de neurones sur les dernières familles VIRTEX des circuits FPGAs à savoir les familles Virtex-2 et Virtex-4. Une comparaison des performances entre ces deux familles est établie sur trois exemples d'application : le problème du XOR, un classificateur des troubles du rythme cardiaque et un réseau complexe de grande taille.
- De nouvelles conclusions, directives et orientations sont données sur les dernières plateformes FPGAs.
- Amélioration de la densité d'intégration par l'utilisation de la reconfiguration dynamique « Run time reconfiguration » : nous proposons une nouvelle stratégie pour cela.
- Amélioration de la flexibilité par l'introduction des concepts de « *Design for Reuse* » et de « *Design with Reuse* » pour l'implantation des réseaux de neurones.
- Proposition d'une plateforme interactive qui prend en charge l'implémentation de l'algorithme RPG depuis les spécifications haut niveau jusqu'au circuit FPGA.
- De nouvelles orientations et directives pour les travaux de recherches futures.

Les réseaux de neurones étant au cœur de notre sujet, nous avons consacré le premier chapitre à une présentation générale des fondements théoriques, des modèles mathématiques et règles d'apprentissages des réseaux de neurones en particulier le perceptron multicouche, basé sur l'algorithme de la rétropropagation du gradient.

Le chapitre II, est consacré à un état de l'art sur l'implémentation hardware des réseaux de neurones : principes et perspectives. Une synthèse sur les différents travaux de recherche académiques et industriels dans ce domaine est mise en valeur ainsi qu'une proposition d'une nouvelle approche pour la classification du hardware neuronal.

Le troisième chapitre est consacré à l'implémentation de l'algorithme RPG sur FPGA.

Le chapitre IV est consacré à l'étude et l'application de la reconfiguration dynamique à l'algorithme RPG.

Dans le chapitre V, nous présentons une méthodologie pour l'application des concepts de « *Design Reuse* » à l'algorithme de la rétropropagation du gradient.

Enfin nous terminerons par une conclusion générale.

---

# *CHAPITRE I*

---

## **Les réseaux de neurones: Principes et définitions**

*"Le neurone est une terra incognita dont nous sommes les Magellan et  
Dont nous sommes loin d'avoir fini d'explorer la complexité"  
Pierre-Marie UEDO, Chercheur CNRS*

## I.1 Introduction

Le XX<sup>ème</sup> siècle a été marqué par deux grandes découvertes fondamentales et incontournables : la première, en 1906, par Camillo Golgi et Santiago Ramon Y. Cajal sur la structure et l'organisation du neurone biologique et qui leur a valu le prix Nobel de médecine, et la deuxième, en 1943, par McCulloch et Walter Pitts sur le neurone formel qui symbolise le modèle mathématique simplifié du neurone biologique. Entre ces deux dates et jusqu'à ce jour, une multitude de travaux de recherches multidisciplinaires sont venus compléter ces découvertes et ont donné naissance à de nouvelles disciplines, voir de nouveaux concepts, paradigmes et outils d'analyse; parmi ces derniers les réseaux de neurones artificiels font l'objet de notre étude dans ce premier chapitre.

Inspiré du comportement du cerveau, un réseau de neurones artificiels peut être défini comme un système possédant une capacité d'apprentissage de l'environnement extérieur. Il est capable de capter les dépendances de haut niveau en modélisant un processus physique, à partir des données expérimentales, mises à sa disposition afin de généraliser sur de nouvelles données.

Aujourd'hui, une myriade de modèles, de règles d'apprentissages et d'architectures neuronales sont disponibles. L'objectif principal est de résoudre les problèmes du monde réel, traditionnellement classés difficiles, en proposant des outils robustes et des solutions nouvelles, alternatives et concurrentes comparées à celles obtenues en appliquant les méthodes statistiques ou bien celles de l'intelligence artificielle.

Les réseaux de neurones sont au cœur de notre travail, tout au long de ce chapitre nous allons d'abord présenter l'historique de la modélisation des réseaux de neurones artificiels. Ensuite, nous présenterons les fondements biologiques sur lesquels reposent les concepts de base des réseaux de neurones artificiels. Cette section sera suivie par une présentation des modèles mathématiques, des règles d'apprentissages et des différentes architectures des réseaux de neurones. Par la suite, nous nous attèlerons d'avantage sur les notions théoriques auxquelles nous avons fait appel pour élaborer notre travail, à savoir le perceptron multicouche basé sur l'algorithme de la rétro-propagation du gradient (RPG). Dans les sections I.7 et I.8 nous présenterons les propriétés générales ainsi que les domaines d'application des réseaux de neurones et enfin nous terminerons par une conclusion.

## I.2 Historique [3, 4, 5]

L'histoire des réseaux de neurones artificiels a été tissée à travers des découvertes conceptuelles et des développements technologiques survenus à diverses époques et marqués par des périodes d'activités de recherche intensives et de déclin. Bien que les premiers travaux de recherches remontent à la fin du 19<sup>ème</sup> et au début du 20<sup>ème</sup> siècle, grâce aux travaux multidisciplinaires en physique, en psychologie et en neurophysiologie par des scientifiques tels que Hermann Von Helmholtz, Ernst Mach et Ivan Pavlov, un consensus est établi au niveau de la communauté scientifique pour dire que la naissance du domaine des réseaux de neurones artificiels remonte au début des années 1940s, grâce au travail pionnier de McCulloch et Walter Pitts. Ces deux chercheurs montrèrent que des réseaux de neurones formels simples peuvent théoriquement réaliser des fonctions logiques, arithmétiques et symboliques complexes et donc émuler un ordinateur.

En 1949, D.Hebb, physiologiste américain, publia son livre intitulé « *The organization of behavior* », dans lequel il exposa ses idées sur l'apprentissage pour la première fois. La règle de Hebb qu'il proposa est l'une des règles d'apprentissage sur laquelle repose la plupart des algorithmes connexionnistes d'aujourd'hui.

En 1957, Rosenblatt développa le modèle du perceptron. C'est un réseau mono-couche inspiré du système visuel et appliqué à la reconnaissance des formes. Basé sur la règle de Hebb, le perceptron est considéré comme étant le premier système artificiel capable d'apprendre par expérience. Malheureusement, le perceptron est un classificateur linéaire et donc il ne peut résoudre qu'une classe limitée de problèmes.

En 1960, B. Widrow et T. Hoff développèrent le modèle du réseau Adaline (Adaptive Linear Element). Dans sa structure, le modèle ressemble au perceptron, cependant la loi d'apprentissage est différente. Ils proposent la minimisation des erreurs quadratiques en sortie comme algorithme d'apprentissage du réseau. Le réseau Adaline est considéré comme modèle de base des réseaux multicouches.

En 1969, M. Minsky et S. Papert montrèrent dans leur livre, intitulé PERCEPTRONS, les limitations théoriques du perceptron. Ces limitations concernent l'impossibilité de traiter des problèmes non linéaires en utilisant ce modèle. Les retombées des résultats scientifiques de Minsky et Papert ont eu pour impact de freiner l'enthousiasme de la plupart des chercheurs dans ce domaine, particulièrement les informaticiens. Cette stagnation a duré presque 20 ans. Durant cette période, les chercheurs et investisseurs se sont tournés vers l'approche de l'intelligence artificielle, qui semblait être plus prometteuse.

Le renouveau de cette discipline repris en 1982 grâce à J. J. Hopfield, un physicien émérite, qui, après avoir remarqué la similarité entre les réseaux du type proposé par McCulloch et Pitts, avec un système élémentaire à moment magnétique ou spins, étudia la capacité de stockage et de restauration de l'information « mémoires associatives ». L'un des apports majeurs de Hopfield fut l'idée d'utiliser une fonction d'énergie pour prouver la stabilité des réseaux de neurones. Avec une telle énergie, les états changent jusqu'à atteindre un minimum local. Ce travail intéressa les physiciens en raison de l'isomorphisme du modèle de Hopfield avec le modèle d'Ising, appelé aussi verres de spins. Il est important de noter que ce modèle ne levait pas les limites du perceptron et de ses variantes. Malgré cela, le perceptron et les raisons de son échec s'avéraient être oubliés.

Par la suite Hinton et son équipe proposèrent, en 1983, la machine de Boltzman: le premier modèle levant les limitations du perceptron 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 linéairement séparables. Malheureusement, la convergence de l'algorithme s'avère extrêmement longue, en raison du caractère probabiliste des éléments utilisés.

Ce défaut a été corrigé par l'algorithme de la rétropropagation du gradient proposé en 1986, par trois chercheurs Rumelhart, Hinton et William.

Enfin, en 1989 Moody et Darken exploitent quelques résultats de l'interpolation multi-variables pour proposer le réseau à fonctions de base radiales, connu sous l'appellation anglophone « *radial basis function network* ».

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

### **I.3 Fondements biologiques des réseaux de neurones artificiels [6, 7]**

La genèse des réseaux de neurones artificiels et de leur modélisation est fondée sur la compréhension de la structure et du fonctionnement du cerveau de manière générale; plus particulièrement celle du neurone. De ce fait, et avant de présenter les réseaux de neurones artificiels, nous avons jugé utile d'introduire une section permettant une présentation très succincte du système nerveux humain, qui représente la créature la plus



complexe et la plus fascinante dans notre univers et qui malgré tous les progrès scientifiques et technologiques atteints jusqu'à aujourd'hui, sa compréhension reste un champ de recherche ouvert.

### I.3.1 Le système nerveux

Le système nerveux est composé de deux sous systèmes : le *système nerveux central* (SNC) et le *système nerveux périphérique* (SNP). La figure I.1 illustre la répartition du système nerveux dans le corps humain.

Le SNC se compose de l'encéphale et de la moelle épinière. L'encéphale correspond aux trois organes qui sont situés dans la cavité de la boîte crânienne, à savoir le cerveau, le cervelet et le tronc cérébral. La moelle épinière constitue la prolongation du tronc cérébral ; elle est située dans le canal rachidien qui résulte de la superposition des vertèbres de la colonne vertébrale.

Le SNP comprend toutes les parties du système nerveux situées à l'extérieur du SNC. Ces organes correspondent aux différents nerfs rattachés à l'encéphale et à la moelle épinière, appelés respectivement, nerfs crâniens, les nerfs rachidiens, de même que les ganglions et les récepteurs sensoriels.

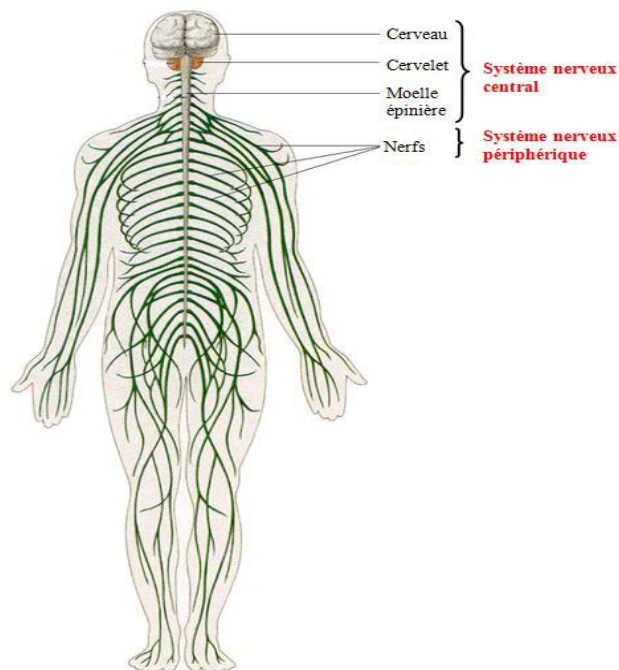


Figure I.1 Système nerveux humain

Le système nerveux possède trois fonctions essentielles :

- Une fonction sensitive de détection grâce à des récepteurs ou neurones sensoriels qui détectent toutes les modifications de l'organisme et de l'environnement extérieur.
- Une fonction d'intégration et d'analyse des informations qu'il reçoit des récepteurs.
- Une fonction motrice permettant la contraction des diverses cellules musculaires de l'organisme.

La figure I.2 explique de manière schématique le fonctionnement du système nerveux. Le SNP organise les informations qui arrivent (inputs) et celles qui sont produites par le cerveau (outputs). Le SNC reçoit l'information, fait un traitement et produit une

réponse, basée sur l'expérience de l'individu, sur les reflexes ainsi que sur les conditions qui prévalent dans l'environnement externe. Il est intéressant de savoir qu'une information ne peut pas aller au cerveau si elle ne passe pas d'abord par le SNP.

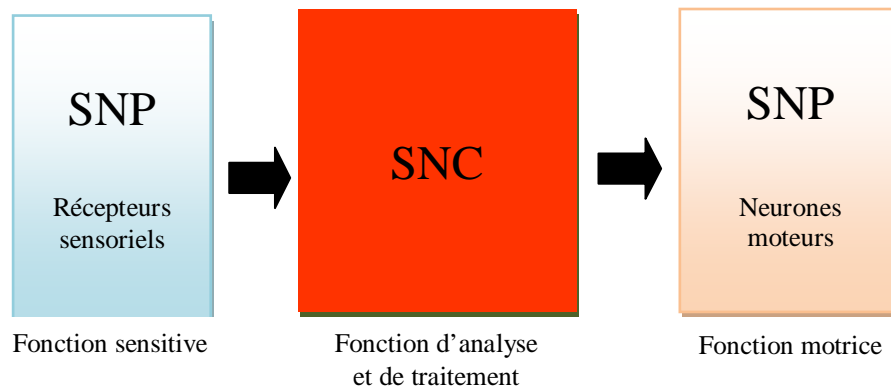


Figure I.2 Différentes fonctions du système nerveux

De point de vue cellulaire, le tissu du système nerveux est constitué principalement de deux grandes catégories de cellules : les *neurones*, qui sont des cellules fonctionnelles et les *cellules gliales* qui sont des cellules de soutien et de protection aux neurones. La figure I.3 montre une vue de ces cellules. Dans ce qui suit nous présentons la physiologie du neurone.

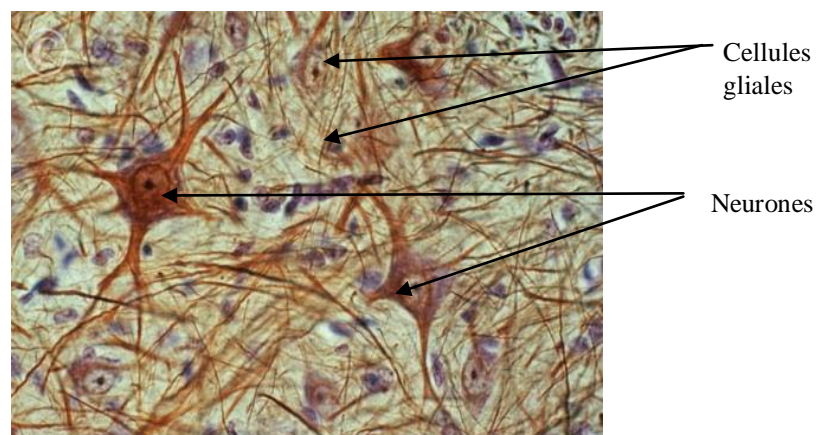


Figure I.3 tissu nerveux : neurones et cellules gliales [8]

## I.3.2 Physiologie du neurone

### I.3.2.1 Description du neurone

Le neurone est une cellule vivante constituant l'unité structurelle et fonctionnelle de base du système nerveux. La figure I.4 montre l'anatomie détaillée d'un neurone appelé aussi cellule nerveuse. On distingue principalement trois régions distinctes : le corps cellulaire, les dendrites et l'axone.

- *Le corps cellulaire* appelé également *soma* ou *péricaryon*, contient le *noyau* et le *cytoplasme*. On y trouve un *réticulum endoplasmique* rugueux, lieu de synthèse

protéique, un appareil de **Golgi**, lieu de stockage et de maturation de protéines. Il constitue le point de passage obligatoire et régulateur au trafic de petites vésicules et plusieurs microtubules qui procurent l'énergie indispensable au métabolisme cellulaire. Le diamètre du péricaryon varie selon le type de neurone, entre 5 et 120  $\mu\text{m}$ .

- **Les dendrites** sont l'extension du péricaryon ; elles servent à augmenter la surface de réception de l'influx nerveux et sont souvent recouvertes de structures en forme de bourgeon, appelées **épines dendritiques**. Un neurone typique contient plusieurs dizaines de milliers d'épines dendritiques ; chacune forme une **synapse**.
- **L'axone** : il est unique. Il conduit l'impulsion produite par le neurone, appelée aussi influx nerveux. Dans sa structure, l'axone émerge du péricaryon, en formant tout d'abord un cône d'émergence, extrêmement riche en microtubules, du fait que c'est le point de départ de l'influx nerveux. L'axone décrit un trait plus ou moins lent avant de se terminer par une arborescence terminale qui donne lieu à plusieurs terminaisons nerveuses. Au bout de chaque terminaison on trouve un renflement, le bouton terminal ou **bouton synaptique (synapse)** qui contient plusieurs résidus synaptiques rempli de neurotransmetteurs. La membrane de l'axone renferme l'**exoplasme** qui est un prolongement du cytoplasme du péricaryon; celui-ci est parcourue dans toute sa longueur par des **micro-filaments** et d'un **microtubule** qui stabilisent la structure de l'axone et assurent le transfert bidirectionnel de substances entre le péricaryon et la terminaison axonale. De l'extérieur, l'axone est entouré par des **cellules de Schwann**, séparées par les **nœuds de Ranvier**, qui confèrent une **couche de myéline** protectrice et qui accélère la conduction électrique.

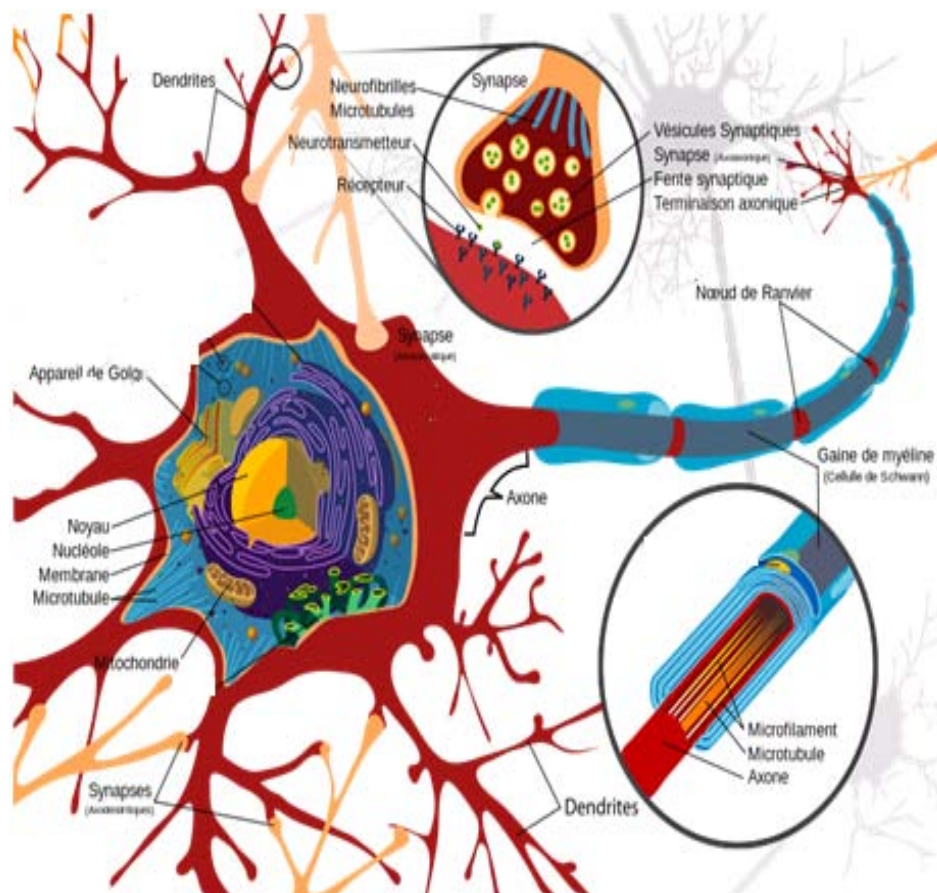


Figure I.4 Anatomie du neurone [9]

### I.3.2 Caractéristiques des neurones

Les neurones présentent plusieurs caractéristiques qui les distinguent des autres types des cellules de l'organisme :

- Tout d'abord, leur nombre, estimé entre 86 à 100 milliards, est quasi complet dès la naissance.
- Une autre particularité du neurone c'est qu'il sert de lieu de transmission et de traitement de l'information. En effet, moyennant un processus électrochimique, le neurone possède deux propriétés physiologiques fondamentales: l'*excitabilité*, c'est-à-dire la capacité de répondre aux stimulations et de convertir celles-ci en impulsions nerveuses, et la *conductivité*, c'est-à-dire la capacité de transmettre des impulsions.
- Le neurone est une cellule sécrétrice qui sécrète des neurotransmetteurs au niveau de sa terminaison axonale.
- C'est une cellule amitotique, exception faite de quelques zones.
- Chaque jour, on perd des dizaines de milliers de neurones. Même s'ils ne se régénèrent pas, cette perte ne se traduit pas par un trouble mentale ; ceci est dû à la souplesse des neurones qui peuvent former de nouvelles connexions, palliant ainsi à leurs pertes.
- La cellule nerveuse est une cellule polarisée. On distingue deux pôles : le pôle somato-dendritique qui reçoit le signal et le pôle axonal qui le propage.
- Le neurone est une cellule à métabolisme très élevé. En effet, le neurone requiert un approvisionnement constant et régulier en oxygène et en glucose.
- Les neurones se caractérisent par un polymorphisme inégale où, on distingue plus de 150 types selon la taille, la polarité, la structure, la fonction, la localisation, etc. Cependant, on peut distinguer, selon leur forme, trois grandes catégories non exclusives (Figure I.5):
  - Les neurones unipolaires ou pseudo-unipolaires, souvent sensoriels (olfactifs, rétine)
  - Les neurones bipolaires, par exemples les inter-neurones,
  - Les neurones multipolaires; c'est les cas des moteur-neurones
- Sur le plan fonctionnel, on distingue les neurones sensibles qui conduisent l'influx nerveux vers le système nerveux centrale et les neurones moteurs qui conduisent l'influx loin du système nerveux central, entre les deux s'interposent d'autres neurones qu'on appel les inter-neurones.
- Une des caractéristiques extraordinaire du neurone est sa *plasticité*, c'est-à-dire sa capacité de se modifier et de s'adapter par expérience, en fonction des besoins. La plasticité neuronale est présente à tous les niveaux : corps cellulaire, axone, dendrites, synapses pour s'étendre jusqu'au cerveau. On parle alors d'un système dynamique en perpétuelle reconfiguration.

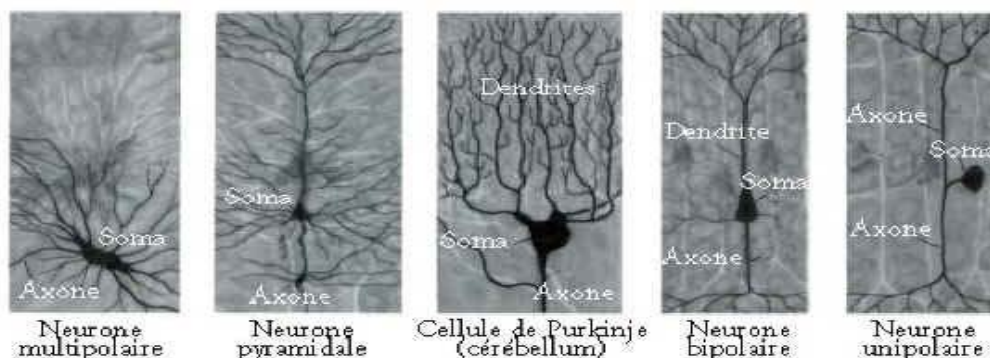


Figure I.5 Différents types de neurones [10]

### I.3.2.3 Fonctionnement du neurone

Le rôle fondamental d'un neurone est de recevoir, propager et transmettre le signal nerveux. Sa membrane plasmique possède des propriétés électrochimiques particulières qui font qu'elle peut réagir à un stimulus et propager son action jusqu'à la terminaison nerveuse. La membrane plasmique des neurones comporte des canaux et des pompes capables de réguler la répartition des ions de part et d'autre de la membrane selon leur charge électrique et leur concentration.

Au repos, la membrane plasmique est polarisée. En effet, les ions sodium ( $\text{Na}^+$ ) et potassium ( $\text{K}^+$ ) sont inégalement répartis de part et d'autre de la membrane. Les ions ( $\text{Na}^+$ ) sont concentrés dans le milieu extracellulaire, alors que les ions ( $\text{K}^+$ ) se trouvent dans le cytoplasme du neurone. Ce déséquilibre est maintenu grâce à la pompe sodium-potassium ( $\text{Na}^+/\text{K}^+$ ). L'intérieur du neurone est donc plus négatif que le milieu extracellulaire: on parle de potentiel de repos. A ce moment, la différence de potentiel entre l'extérieur et l'intérieur est de l'ordre de  $-70 \text{ mV}$ .

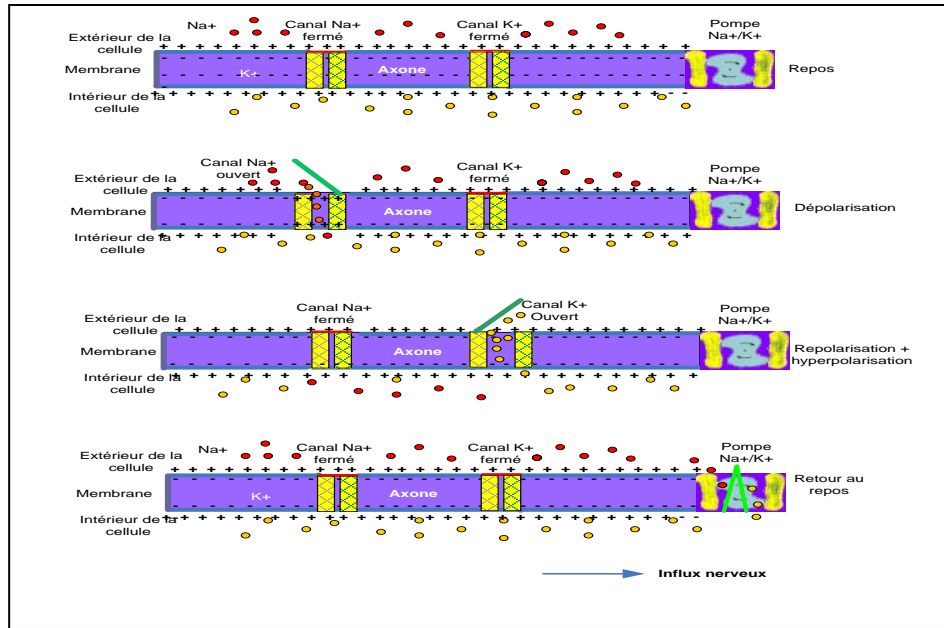
Lorsque la membrane du neurone est déstabilisée, les portes des canaux s'ouvrent. Cependant, les portes des canaux sodium s'ouvrent plus rapidement que celle des canaux à potassium. Les ions  $\text{Na}^+$  entrent donc massivement dans le neurone, rendant celui-ci moins négatif. Si le *seuil* d'excitation est atteint, la réponse de l'axone présente toujours la même amplitude : le neurone obéit à la loi du « tout ou rien » et il se produit alors :

- Une **dépolarisation** : ouverture des canaux à sodium ( $\text{Na}^+$ ), et entrée massive de sodium ; l'intérieur de la fibre devient positif, atteinte du maximum du potentiel de la membrane.
- Une **repolarisation** : ouverture des canaux à potassium ( $\text{K}^+$ ), et fermeture des canaux à sodium; sortie de potassium.
- Une **hyperpolarisation** : les canaux à potassium ( $\text{K}^+$ ) restent ouverts plus longtemps ; ainsi beaucoup d'ions sortiront ce qui diminuera le potentiel intracellulaire plus bas que le potentiel de repos.
- **Retour** au potentiel de repos : La pompe  $\text{Na}^+/\text{K}^+$  se chargera de ramener les ions  $\text{Na}^+$  à l'extérieur de la cellule et les ions  $\text{K}^+$  à l'intérieur. Le potentiel de repos sera ainsi rétabli.

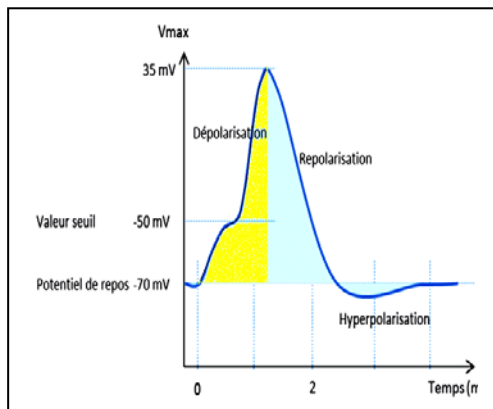
La figure I.6 montre les différentes étapes de création du potentiel d'action pour le fonctionnement du neurone. Le long d'un neurone, l'influx nerveux se propage et déclenche dans les terminaisons neuronales, la libération de *neurotransmetteurs*. Ces derniers se trouvant dans la fente synaptique, vont aller se fixer sur un autre neurone, créant une dépolarisation (ouverture des canaux à sodium) et un nouvel influx nerveux. Il est important de noter que la génération d'un potentiel d'action est le fruit de nombreuses dépolarisations. L'action d'une seule synapse est pratiquement sans effet.

On peut donc assimiler le fonctionnement du neurone comme suit :

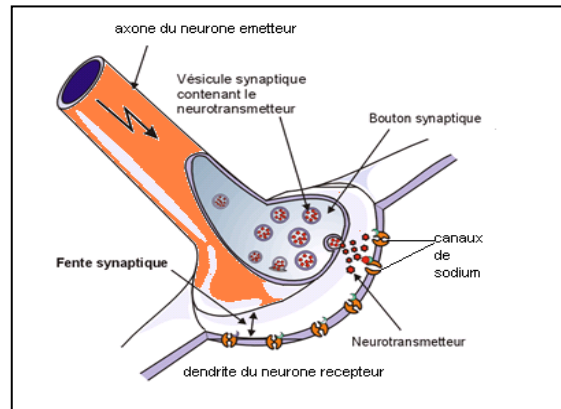
- Les dendrites reçoivent l'influx nerveux d'autres neurones.
- Le neurone évalue alors l'ensemble de la stimulation qu'il reçoit (c'est à dire sa dépolarisation par rapport à l'extérieur).
- En fonction du seuil de cette stimulation, le neurone transmet ou non un signal de type « tout ou rien » le long de son axone. On dira alors que le neurone est « excité » 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.



(a)



(b)



(c)

Figure I.6 Fonctionnement du neurone et genèse du potentiel d'action. (a) Canaux et pompes au niveau de la membrane plasmique (noyau + axone) (b) Représentation graphique du potentiel d'action (c) Fonctionnement au niveau synaptique.

### I.3.3 Organisation des neurones dans le système nerveux central

Les différentes études anatomiques et histologiques de la texture du système nerveux central (SNC) permettent de ressortir certaines caractéristiques liées à l'organisation des neurones dans le système nerveux. Ces caractéristiques peuvent être résumées comme suit:

- D'abord on peut discerner deux substances : la substance grise et la substance blanche. La substance grise correspond aux corps cellulaires des neurones alors que la substance blanche correspond aux axones des neurones (fibres nerveuses). Selon les régions du SNC, la répartition substance blanche / substance grise va changer de même que leurs proportions respectives.
- Les corps cellulaires neuronaux de la substance grise se regroupent pour former trois types de structures : les cortex, les noyaux et les cornes. Les cortex sont localisés à la surface du SNC, dans le cerveau et le cervelet. Les noyaux sont localisés dans la

profondeur de l'encéphale et du tronc cérébral alors que les cornes sont localisées au niveau de l'hippocampe et de la substance grise de la moelle épinière.

- Particulièrement, le cortex cérébral qui est le centre d'intégration et le siège des fonctions cognitives évoluées caractérisant l'Homme, est organisé en six couches cellulaires distinctes numérotées de I à VI, comme le montre la figure I.7. Chaque couche est caractérisée par un type de neurones (pyramidales, granulaires, etc.) et qui vont participer avec une fonction spécifique afin de faire un traitement des signaux qui arrivent au niveau du cortex en provenance d'autres structures du cerveau et vis versa. Chez les autres créatures, le nombre de couches est différent, à titre d'exemple, cinq chez le dauphin et trois chez les reptiles.
- Dans ces couches, les neurones s'organisent en unités fonctionnelles prenant la forme de colonnes perpendiculaires à la surface du cortex, chacune assurant une fonction précise.

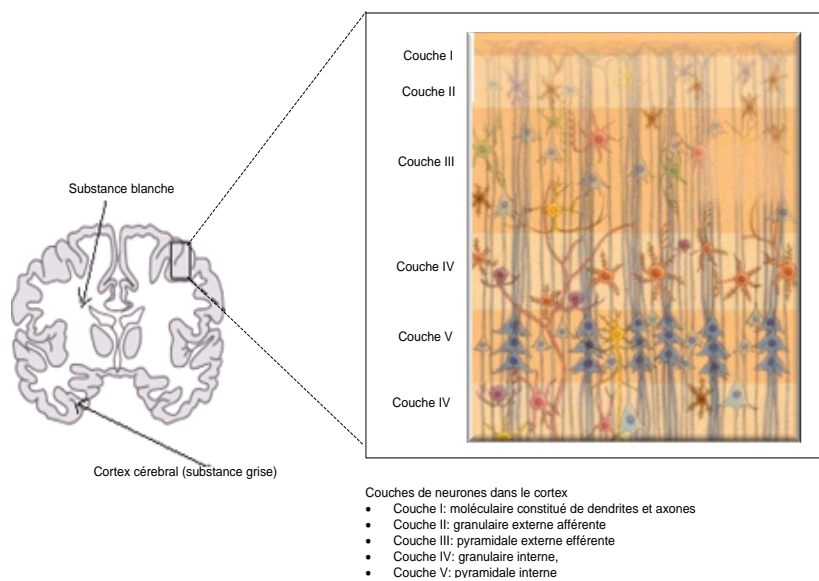


Figure I.7 Organisation en couche des neurones dans le cortex cérébral [11]

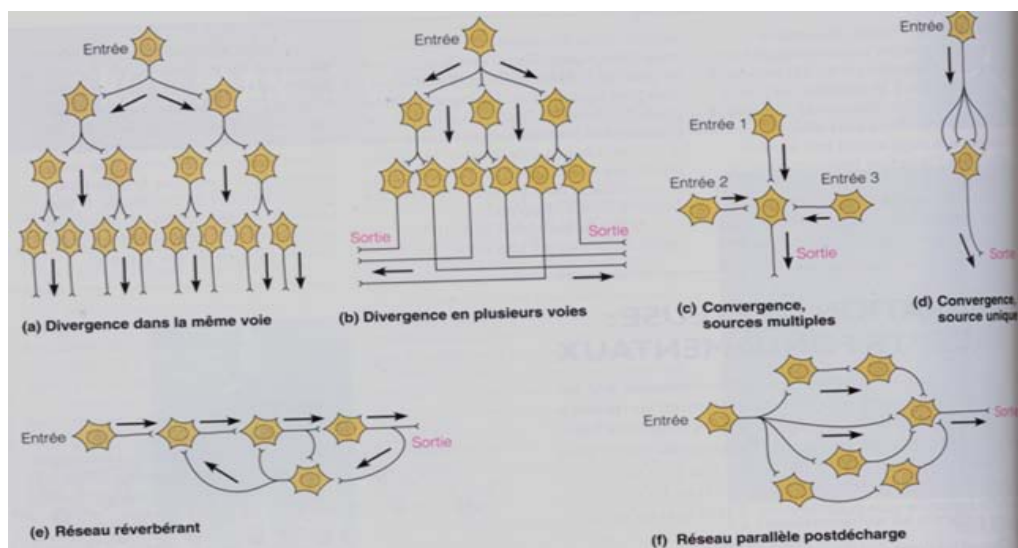


Figure I.8 Différents types de réseaux [12]

- Notons enfin que les opérations, les plus complexes de traitement de l'information au niveau du SNC, se basent sur des assemblages plus ou moins complexes de réseaux de neurones. L'organisation de ces neurones repose sur deux propriétés générales, à savoir les propriétés de *convergence* et de *divergence* des neurones. Dans le cas de la convergence, plusieurs neurones vont converger vers un neurone post synaptique ; ceci va assurer une sommation spatiale des informations et provoquer un seul potentiel d'action. Dans le cas de la divergence, un seul potentiel d'action sera distribué vers plusieurs neurones. La figure I.8 montre les différents types de convergence et divergence des neurones ainsi que les réseaux qui peuvent être formés.

#### **I.4 Réseaux de neurones artificiels [13-21]**

Les réseaux de neurones artificiels sont une tentative de modélisation du cerveau humain. Ils possèdent plusieurs caractéristiques fondamentales parmi lesquelles:

- Ils sont composés d'un grand nombre d'unités simples de traitement (processeurs élémentaires) qui ressemblent aux neurones du cerveau.
- Ils sont composés de deux ou plusieurs couches organisées hiérarchiquement. Typiquement, celles-ci incluent une couche d'entrée, dont les unités de traitement encodent la représentation initiale de la situation, une ou plusieurs couches cachées, dont les unités combinent l'information à partir des unités d'entrée, et une couche de sortie, dont les unités produisent la réponse du système à la situation.
- Les unités simples de traitement sont connectées à d'autres unités de traitement dans différentes couches (et quelquefois à l'intérieur de la même couche). Le poids des connexions varie avec l'expérience du système et est cruciale pour déterminer le traitement effectué.
- Comme dans le cerveau, une unité de traitement donnée s'active lorsque le niveau de stimulation qu'elle reçoit de toutes les autres unités de traitement auxquelles elle est connectée dépasse un certain seuil. Le niveau de stimulation qu'une unité reçoit de chaque unité à laquelle elle est connectée est déterminé, d'une part, par le degré d'activation de l'unité de traitement qui envoie l'activation et, d'autre part, par le poids de la connexion entre les unités.
- Comme dans le cerveau, l'activité de la plupart des unités de traitement se produit en parallèle (simultanément).
- La connaissance est représentée par le poids des connexions au sein de toutes les unités du système.
- L'apprentissage se produit à partir du système qui reçoit les données, élabore une réponse, observe la différence entre la réponse fournie et la réponse correcte et ajuste le poids des connexions entre les unités de traitement afin de produire une meilleure réponse. Les ajustements incluent le renforcement de certaines connexions et l'affaiblissement d'autres.
- La généralisation de la connaissance du système est fondée sur la similitude de nouvelles situations à celles déjà rencontrées par le système.

##### **I.4.1 Le neurone artificiel**

Le neurone artificiel est une unité de calcul qui génère un signal de sortie en fonction des signaux d'entrée. Le tableau I.1 montre la correspondance entre le neurone biologique et le neurone artificiel. Une représentation schématisée simplifiée du fonctionnement de ce dernier est montrée sur la figure I.9 (a) et la figure I.9 (b) représente son modèle mathématique.



Tableau I.1 Analogie entre le neurone biologique et le neurone artificiel

<i>Neurone biologique</i>	<i>Neurone artificiel</i>
Corps cellulaire (Somma)	Fonction d'activation
Axones	Signal de sortie
Synapses	Poids synaptiques (connexions)
Dendrites	Signal d'entrée

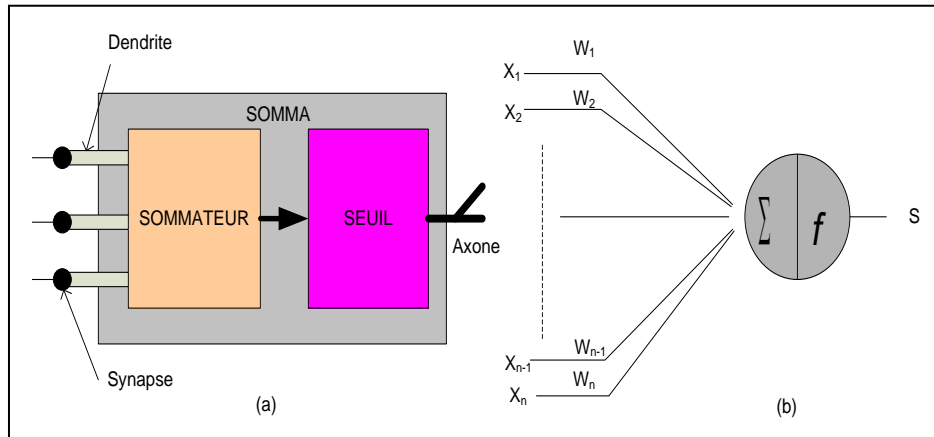


Figure I.9 Neurone artificiel. (a) représentation schématique (b) modèle mathématique

Dans le modèle mathématique, le neurone est modélisé par deux fonctions :

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

$$P = \sum_{i=1}^n W_n X_n \quad (\text{I.1})$$

Où,  $n$ , représente le nombre total des entrées du neurone et les  $W_n$ , représentent les poids synaptiques.

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

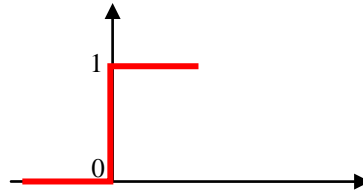
$$S = f(P) = f\left(\sum_{i=1}^n W_n X_n\right) \quad (\text{I.2})$$

La modélisation du neurone est passée par trois générations. La première génération est celle des neurones à fonction d'activation échelon. La seconde génération est celle à fonction d'activation continue et la troisième génération est basée sur le modèle du neurone impulsionnel. Dans ce qui suit nous allons présenter chaque génération.

#### 1.4.1.1 Première génération : les modèles du neurone à fonction échelon

La première génération des neurones était basée sur le modèle de McCulloch-Pitts. Ce modèle utilise la fonction d'activation à seuil tel que :

$$\begin{cases} S = f(P) = 0 & \text{si } p \leq 0 \\ S = f(P) = 1 & \text{si } P > 0 \end{cases} \quad (\text{I.3})$$



Dans le modèle défini par McCulloch-Pitts, seuls les problèmes de classificateurs avec des classes linéairement séparables peuvent être résolus avec ce type de neurones (Cas d'une porte OU par exemple).

#### 1.4.1.2 Seconde génération : Le neurone à fonction d'activation continue

La seconde génération de neurones est basée sur des unités de calcul qui construisent une réponse continue résultant de la somme pondérée des entrées. Les fonctions d'activation sont classiquement les fonctions linéaires saturées ou sigmoïdales comme montré sur la figure I.10.

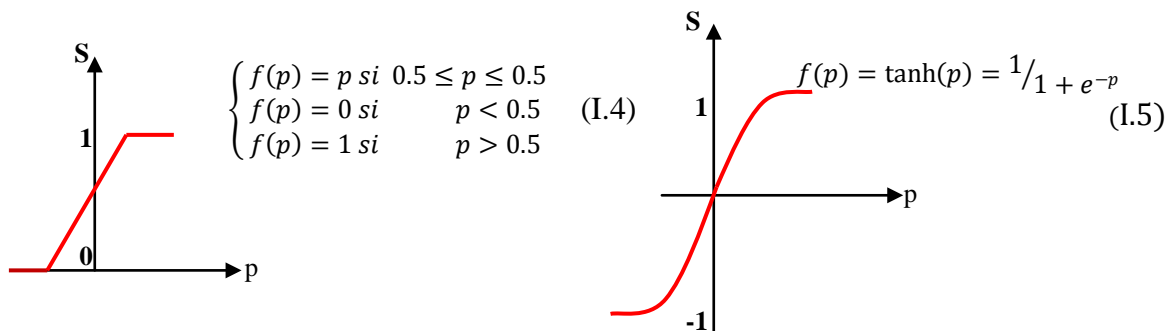


Figure I.10 Fonctions linéaire (à gauche) et sigmoïdale (à droite)

Les neurones issus de la seconde génération sont capables de calculer des fonctions à partir des valeurs non booléennes (possibilités d'avoir des réponses intermédiaires). Ces neurones comportent des fonctions d'activation dérivables ce qui présente un intérêt crucial dans l'apprentissage des réseaux multicouches.

#### 1.4.1.3 Troisième génération : Le neurone impulsionnel

Le modèle du neurone décrit par l'équation (1.1) est un modèle à temps discret: on suppose un signal constant à l'entrée du neurone jusqu'à ce que la sortie soit calculée. Ainsi la caractéristique du neurone est atemporelle. Dans le modèle impulsionnel, on considère que les neurones encodent l'information dans les instants de l'émission d'une impulsion. La figure I.11 (a) montre deux neurones : l'un placé avant la synapse (neurone pré-synaptique) et l'autre placé après la synapse (post-synaptique). Le neurone pré-synaptique émet une impulsion électrique appelée *potentiel d'action*, *spike* ou encore *pulse*. Chaque spike pré-synaptique engendre un potentiel d'action excitateur (figure I.11b). Les différents spikes vont s'ajouter ; et lorsque le seuil est atteint alors le neurone post-synaptique émet à son tour un spike.

Les modèles de neurones impulsionnels proposés dans la littérature peuvent être regroupés en quatre grandes familles : le modèle de *Hodgkin & Huxley*, le modèle *Integrate & Fire* [40], le modèle *Spike Response* [41] et le modèle *Izhikeviche*.

Il a été prouvé que les modèles impulsionnels des neurones sont plus réalistes, précis et peuvent constituer de puissants outils de calcul. Néanmoins, la modélisation, l'apprentissage et l'implémentation des réseaux de neurones impulsionnels est complexe et est considérée comme un sujet de recherche ouvert.

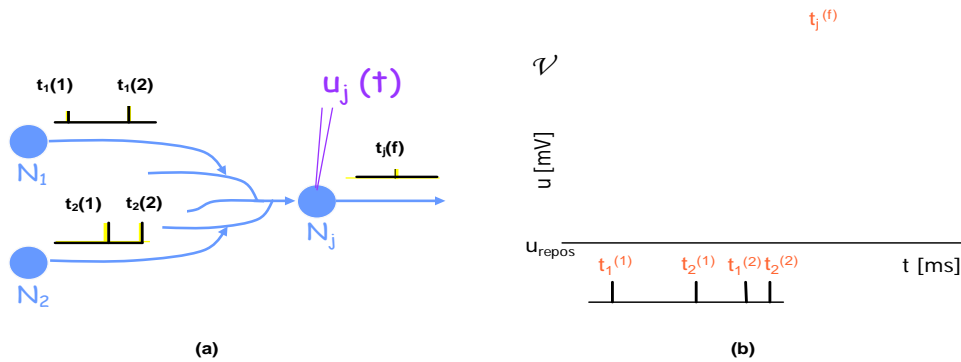


Figure I.11 génération du potentiel d'action dans le modèle du neurone impulsionnel

## I.4.2 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 connexion. Les réseaux de neurones peuvent se distinguer par :

- L'architecture
- Le mode d'apprentissage

### I.4.2.1 Architecture des réseaux de neurone

D'un point de vue architectural, les réseaux de neurones peuvent être regroupés en deux catégories comme montré dans la figure Figure I.12:

- Les réseaux à circulation de l'information vers l'avant «Feed-forward networks» et dans lesquels les neurones sont organisés en couche successives. Le calcul se fait en propageant les données de l'entrée vers la sortie. Dans cette catégorie on distingue les réseaux à une seule couche (exemple : le perceptron) et les réseaux multicouches possédant une couche d'entrée, une couche de sortie et une ou plusieurs couches cachées.
- Les réseaux récurrents bouclés ou «Feed-back network» : ils présentent au moins une boucle de rétroaction. Dans cette catégorie on distingue les réseaux compétitifs, le réseau de Kohonen, les réseaux de Hopfield et les modèles ART « Théorie de la Résonance Artificielle ».

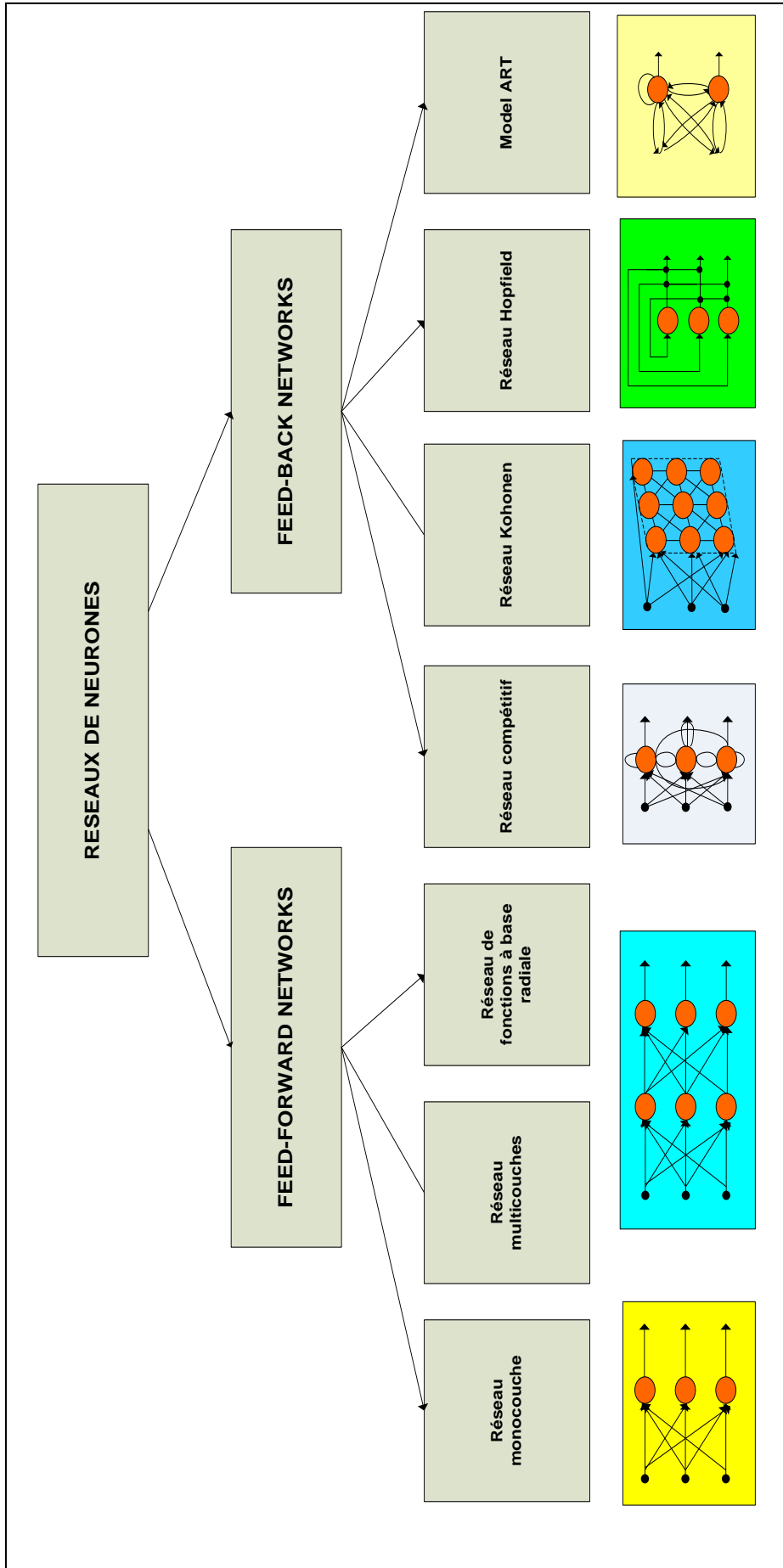


Figure I.12 Différentes architectures des réseaux de neurones

### I.4.2.2 Apprentissage des réseaux de neurones

L'apprentissage est une phase de développement d'un réseau de neurone durant laquelle le réseau de neurones modifie sa structure interne jusqu'à l'obtention du comportement désiré. Pour cela il nécessite :

- Un ensemble d'exemples d'apprentissages
- La définition d'une fonction de coût qui mesure l'écart entre les sorties du réseau de neurone et les sorties désirées.
- Un algorithme de minimisation de la fonction de coût par rapport aux paramètres.

Il existe trois types d'apprentissages :

#### I.4.2.2.1 Apprentissage 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 comme l'indique la figure I.13

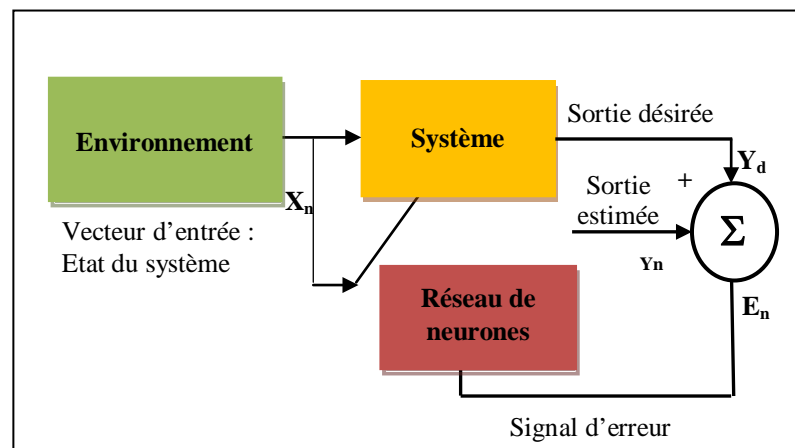


Figure I.13 Principe de l'apprentissage supervisé

Dans la figure I.13, le *Système* à apprendre constitue un professeur pour le réseau, le *réseau de neurones* prend comme stimulus le même vecteur des variables explicatives que le *Système*. Ce dernier, en réponse à l'entrée présentée, fournit une *Sortie désirée*,  $Y_d$ , qui représente pour le réseau le comportement de référence. Parmi les algorithmes à apprentissage supervisé, on reconnaît dans la littérature l'algorithme à Rétro-propagation du gradient.

#### I.4.2.2.2 Apprentissage renforcé

C'est un apprentissage qui élabore et assimile une *critique* à l'issue de chaque exemple traité par la machine d'apprentissage. En effet l'algorithme d'apprentissage renforcé utilise des indications imprécises sur le comportement final «échec ou succès» du réseau. L'ajustement des poids du réseau se fait uniquement à partir d'une simple évaluation de la qualité de sa réponse. LA figure I.14 montre le principe de l'apprentissage par renforcement.

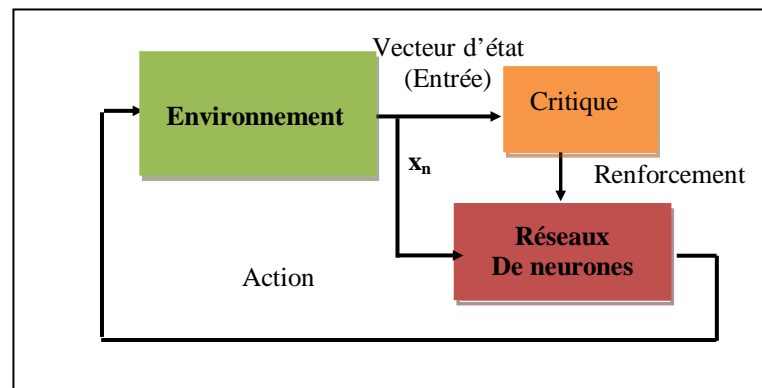


Figure I.14 Principe de l'apprentissage par renforcement

Un apprentissage par renforcement est nécessaire pour la plupart des réseaux en interaction avec l'environnement. Ces interactions fournissent rarement des exemples du comportement désiré mais uniquement une évaluation du comportement (une récompense ou une pénalité). Le réseau doit alors découvrir les réponses qui lui apportent le maximum de récompenses.

#### I.4.2.2.3 Apprentissage non supervisé

L'apprentissage supervisé s'effectue sous le contrôle d'un expert, alors que l'apprentissage non supervisé est autodidacte. Les paramètres internes du réseau ne sont modifiés qu'avec le seul stimulus, aucune réponse désirée n'est prise en considération. La Figure I.15 montre que la sortie du réseau n'est pas utilisée par la procédure d'apprentissage.

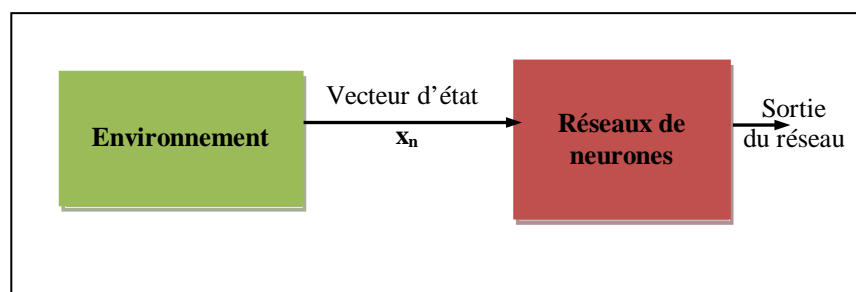


Figure I.15 Principe de l'apprentissage non supervisé

Par nature, ce type d'apprentissage construit une représentation interne de la connaissance, issue de l'environnement. Cet apprentissage est basé sur une mesure de la qualité de la représentation de la connaissance pour ajuster en conséquence les paramètres internes du réseau de neurone, un critère interne souvent utilisé pour modifier les poids des neurones.

Il faut souligner que l'efficacité de l'apprentissage augmente avec le nombre d'exemples d'apprentissage. Ceci est un facteur très important qu'il faut noter : quelque soit l'algorithme choisi, la qualité de l'apprentissage des réseaux de neurone est d'autant meilleure que l'on dispose d'un ensemble d'apprentissage riche en exemples. On sait par ailleurs que la capacité de généralisation des réseaux de neurone nécessite également des exemples nombreux, qui de plus, doivent être bien distribués dans le domaine de validité souhaité par le modèle.

## I.5 Les règle d'apprentissage

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

### I.5.1. La règle de Hebb

L'apprentissage de Hebb est le premier mécanisme d'évaluation des synapses, proposée par D.O Hebb en 1949. Il s'applique aux connexions entre neurones, La règle de Hebb s'exprime de la façon suivante : "*Si deux cellules sont activées en même temps alors la force de la connexion augmente*"

La modification de poids dépend de la co-activation des neurones pré-synaptiques et post- synaptique. La loi de Hebb peut être modélisée par les équations suivantes :

$$W_{ij}(t+1) = W_{ij}(t) + \eta S_i S_j \quad (I-6)$$

Où,

$W_{ij}(t)$  est le poids de la connexion reliant les neurones  $S_i$  et  $S_j$  à l'étape  $t$ .

$W_{ij}(t+1)$  est le nouveau poids à l'étape  $t+1$ .

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

$t$  : représente l'étape d'apprentissage.

L'algorithme d'apprentissage modifie de façon itérative les poids pour adapter la réponse obtenue à la réponse désirée. Il s'agit en fait de modifier les poids lorsqu'il y a erreur seulement.

1. Initialisation des poids et du seuil  $S$  à de faibles valeurs aléatoires.
2. Présentation d'une entrée  $X = (x_1, \dots, x_n)$  de la base d'apprentissage.
3. Calcul de la sortie obtenue  $S$  pour cette entrée.
4. Si la sortie  $S$  est différente de la sortie désirée pour cet exemple d'entrée alors on modifie les poids synaptiques selon l'équation (I-3).
5. Tant que tous les exemples de la base d'apprentissage ne sont pas traités correctement, retour à l'étape 2.

### I.5.2 La règle d'apprentissage 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 :

$$\begin{cases} W_{ij}(t+1) = W_{ij}(t) + \eta x_i & \text{Si la sortie actuelle est égale à 0 et doit être égale à 1.} \\ W_{ij}(t+1) = W_{ij}(t) - \eta x_i & \text{Si la sortie actuelle est égale à 1 et doit être égale à 0.} \\ W_{ij}(t+1) = W_{ij}(t) & \text{Si la sortie est correcte.} \end{cases} \quad (I.7)$$

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'algorithme d'apprentissage du Perceptron est semblable à celui utilisé pour la loi de Hebb. Les différences se situent au niveau de la modification des poids.

L'inconvénient majeur du Perceptron est qu'il ne s'applique que si les classes sont linéairement séparables.

### I.5.3 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. Elle est aussi connue sous le nom de la méthode des moindres carrés ou LMS (**L**east **M**ean **S**quar). Le principe est le suivant :

- Calculer l'erreur quadratique selon la formule :

$$E = \sum_{j=1}^n (d_j - y_j)^2 \quad (\text{I-8})$$

Avec :

$$y_i = \sum_{i=1}^m x_i W_{ji} \quad (\text{I-9})$$

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

$$W_{ji}(t+1) = W_{ji}(t) + \eta x_i (d_j - y_j) \quad (\text{I-10})$$

Où,

- $n$  : nombre de neurones à la sortie
- $d_i$  : 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
- $\eta$  : Coefficient d'apprentissage.

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

### I.5.4 La règle delta généralisé

La règle delta généralisée ou encore appelée règle de Rétro-propagation 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 cette règle dans la prochaine section.

## I.6 Le perceptron multicouches et l'algorithme de la rétropropagation du gradient

On considère que le perceptron multicouche (MLP) est constitué de trois couches de neurones ; la première couche avec  $n_0$  entrées et  $n_1$  unités (neurones), la seconde avec  $n_2$  unités et la sortie avec  $n_3$  unités.

L'algorithme de Rétro propagation du gradient est utilisé dans le Perceptron multicouche.

La Figure I.16 montre une représentation fonctionnelle de l'algorithme.



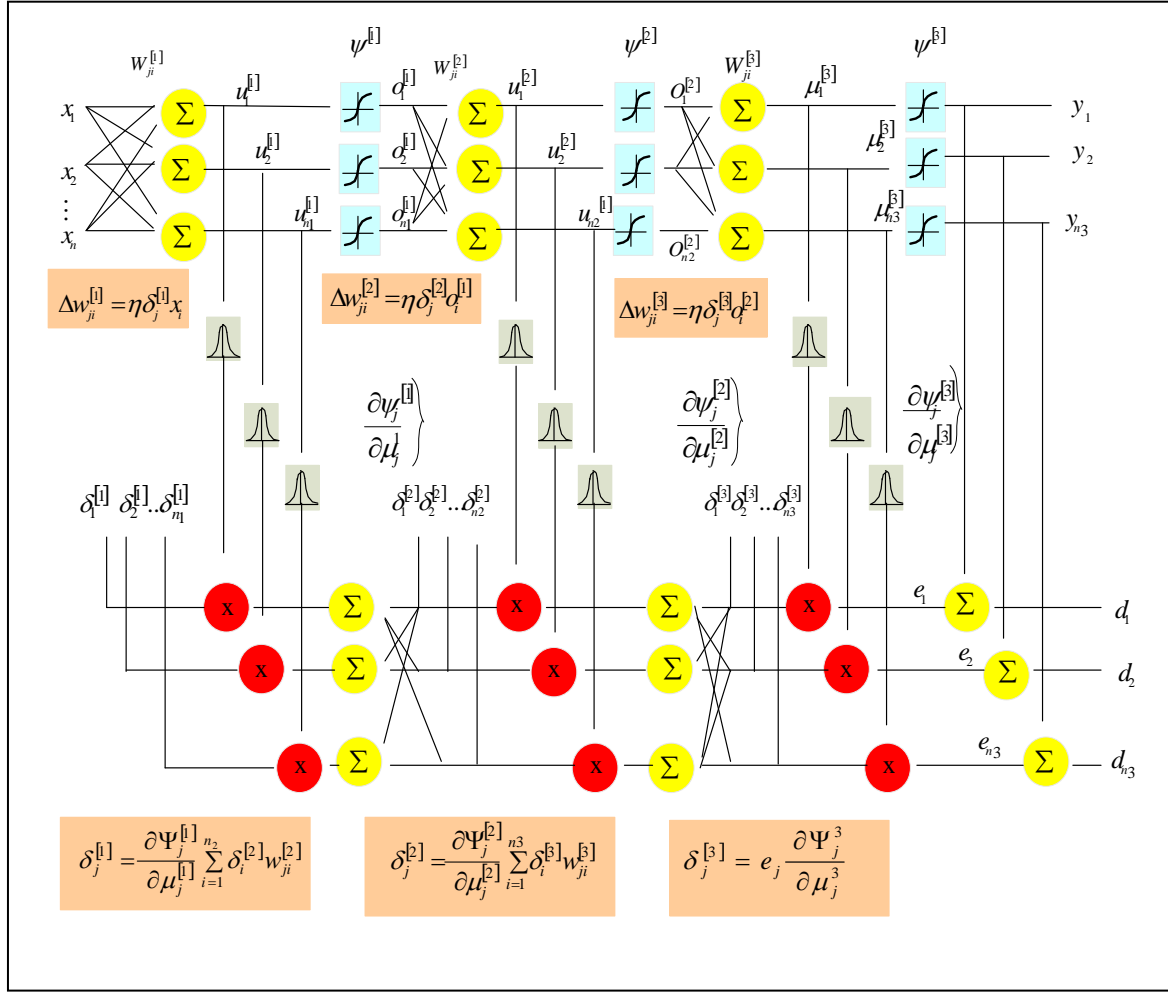


Figure I.16 représentation fonctionnelle de l’algorithme de la Rétro-propagation du gradient

La phase la plus importante où 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 celle désirée soit minimisée. Malgré tout, il ne faut pas oublier que l’on a une couche cachée. On rétro propage 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 ce qu’on obtienne une erreur de sortie considérée comme négligeable.

L’algorithme de la Rétro-propagation standard utilise la méthode de plus forte descente pour la minimisation de la moyenne carrée de l’erreur locale, qui est donnée par :

$$E_p = \frac{1}{2} \sum_{j=1}^{n_3} (d_{jp} - y_{jp})^2 = \frac{1}{2} \sum_{j=1}^{n_3} e_{jp}^2 \tag{I.11}$$

Et de l’erreur globale donnée par :

$$E = \sum_p E_p = \frac{1}{2} \sum_p \sum_j (d_{jp} - y_{jp})^2 \tag{I.12}$$

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 :

$$\Delta W_{ji}^{[s]} = -\eta \frac{\partial E_p}{\partial W_{ji}^{[s]}} \quad \eta > 0 \quad (\text{I.13})$$

$\eta$ , est le coefficient d'apprentissage

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

Pour la couche de sortie ( $s=3$ )

$$\Delta W_{ji}^{[3]} = -\eta \frac{\partial E_p}{\partial W_{ji}^{[3]}} = -\eta \frac{\partial E_p}{\partial \mu_j^{[3]}} \cdot \frac{\partial \mu_j^{[3]}}{\partial W_{ji}^{[3]}} \quad (\text{I.14})$$

$$\text{avec } \mu_j^{[3]} = \sum_{i=1}^{n_2} W_{ji}^{[3]} x_i^{[3]} = \sum_{i=1}^{n_2} W_{ji}^{[3]} O_i^{[2]} \quad (j=1, \dots, n_3) \quad (\text{I.15})$$

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

$$\delta_j^{[3]} = -\frac{\partial E_p}{\partial \mu_j^{[3]}} = -\frac{\partial E_p}{\partial e_j} \cdot \frac{\partial e_j}{\partial \mu_j^{[3]}} = e_{jp} \cdot \frac{\partial \psi_j^{[3]}}{\partial \mu_j^{[3]}} \quad (\text{I.16})$$

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

$$\Delta W_{ji}^{[3]} = \eta \delta_j^{[3]} x_i^{[3]} = \eta \delta_j^{[3]} O_i^{[2]} \quad (\text{I.17})$$

$$\text{Où, } \delta_j^{[3]} = e_{jp} (\psi_j^{[3]})' = (d_{jp} - y_{jp}) \frac{\partial \psi_j^{[3]}}{\partial \mu_j^{[3]}} \quad (\text{I.18})$$

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

$$\begin{aligned} \Delta W_{ji}^{[2]} &= -\eta \frac{\partial E_p}{\partial W_{ji}^{[2]}} = -\eta \frac{\partial E_p}{\partial \mu_j^{[2]}} \cdot \frac{\partial \mu_j^{[2]}}{\partial W_{ji}^{[2]}} \\ &= \eta \delta_j^{[2]} x_i^{[2]} = \eta \delta_j^{[2]} O_i^{[1]} \end{aligned} \quad (\text{I.19})$$

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

$$\begin{aligned} \delta_j^{[2]} &= -\frac{\partial E_p}{\partial \mu_j^{[2]}} \\ &= -\frac{\partial E_p}{\partial O_j^{[2]}} \cdot \frac{\partial O_j^{[2]}}{\partial \mu_j^{[2]}} \end{aligned} \quad (\text{I.20})$$

$$\text{Avec } \mu_j^{[2]} = \sum_{i=1}^{n_1} W_{ji}^{[2]} x_i^{[2]} = \sum_{i=1}^{n_1} W_{ji}^{[2]} O_i^{[1]} \quad (j=1, \dots, n_2) \quad (I.21)$$

Sachant que

$$O_j^{[2]} = \psi_j^{[2]}(\mu_j^{[2]}) \quad (I.22)$$

On obtient

$$\delta_j^{[2]} = -\frac{\partial E_p}{\partial O_j^{[2]}} \cdot \frac{\partial \psi_j^{[2]}}{\partial \mu_j^{[2]}} \quad (I.23)$$

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

$$\begin{aligned} \frac{-\partial E_p}{\partial O_i^{[2]}} &= -\sum_{i=1}^{n_3} \frac{\partial E_p}{\partial \mu_j^{[3]}} \cdot \frac{\partial \mu_j^{[3]}}{\partial O_i^{[2]}} \\ &= \sum_{i=1}^{n_3} \left( -\frac{\partial E_p}{\partial \mu_j^{[3]}} \right) \frac{\partial}{\partial O_j^{[2]}} \left[ \sum_{i=1}^{n_3} W_{ji}^{[3]} x_i^{[3]} \right] \\ &= \sum_{i=1}^{n_3} \delta_i^{[3]} \frac{\partial}{\partial O_i^{[2]}} \left[ \sum_{i=1}^{n_3} W_{ji}^{[3]} O_i^{[2]} \right] \\ &= \sum_{i=1}^{n_3} \delta_i^{[3]} W_{ji}^{[3]} \end{aligned} \quad (I.24)$$

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

$$\delta_j^{[2]} = \frac{\partial \psi_j^{[2]}}{\partial \mu_j^{[2]}} \sum_{i=1}^{n_3} \delta_i^{[3]} W_{ji}^{[3]} \quad (I.25)$$

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

$$\Delta W_{ji}^{[1]} = \eta \delta_j^{[1]} x_i^{[1]} = \eta \delta_j^{[1]} O_i^{[0]} = \eta \delta_j^{[1]} x_i \quad (I.26)$$

Où l'erreur locale est déterminée par :

$$\delta_j^{[1]} = \frac{\partial \psi_j^{[1]}}{\partial \mu_j^{[1]}} \sum_{i=1}^{n_2} \delta_j^{[2]} W_{ij}^{[2]} \quad (I.27)$$

$$\mu_j^{[1]} = \sum_{i=1}^{n_0} W_{ji}^{[1]} x_i \quad (j=1, \dots, n_1) \quad (I.28)$$

Les étapes d'apprentissage de l'algorithme de la Rétro-propagation 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 présentées  $W_{ji}^{[s]}$  dans les équations (I.28), (I.18) et (I.24).
3. Spécifier la sortie désirée et évaluer les erreurs locales  $\delta_j^{[s]}$  pour toutes les couches en utilisant les équations (I.16), (I.25) et (I.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 (I.26), (I.19) et (I.17).
5. Calculer l'erreur  $E_p$  en utilisant l'équation (I.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 que le réseau converge.

Notons aussi, qu'en plus du coefficient d'apprentissage,  $\eta$ , on définit dans la formule de variation des poids synaptiques,  $\Delta W$ , un facteur momentum  $\alpha$ , tel que :

$$\Delta W(t+1) = W(t) + \eta \delta \eta \alpha \Delta \quad (1.29)$$

Le facteur *momentum* permet d'améliorer la vitesse de convergence tout en prévenant l'algorithme des oscillations.

## I.7 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 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.8 Domaines d'applications des réseaux de neurones

Se trouvant à l'intersection de différents domaines (informatique, électronique, science cognitive, neurobiologie et même philosophie), l'étude des réseaux de neurones est une voie prometteuse de l'Intelligence Artificielle, qui a des applications dans de nombreux domaines :

- **Défense** : Direction des armes, poursuite des cibles, radars : traitement, compression, suppression du bruit, identification signal/image, etc.
- **Industrie** : contrôle qualité, control de processus, diagnostic de panne, corrélations entre les données fournies par différents capteurs, analyse de signature ou d'écriture manuscrite, synthèse de la parole, système de guidage automatique des véhicules, etc.
- **Divertissement** : Animation, effets spéciaux.

- **Finance** : prévision et modélisation du marché (cours de monnaies...), prévision des indicateurs économiques, sélection d'investissements, attribution de crédit, prévision des prix, etc.
- **Télécommunications et informatique** : analyse du signal, élimination du bruit, reconnaissance de formes (bruits, images, paroles), compression de données, etc.
- **Médical** : analyse des signaux EEG, ECG, prothèses, analyse du cancer, etc.
- **Environnement** : évaluation des risques, analyse chimique, prévisions et modélisation météorologiques, gestion des ressources, etc.

## I.9 Conclusion

Les réseaux de neurones artificiels ont progressé grâce à l'émergence de nouveaux concepts sur la base de fondements biologiques et accompagnés de modèles mathématiques. Aujourd'hui, ils sont devenus des outils robustes et incontestables dans plusieurs domaines d'applications.

Cependant, et comparé aux autres sciences, le développement des réseaux de neurones artificiels semble connaître des périodes de déclin au lieu d'une évolution constante.

Face à ce constat, la question qu'on peut se poser est la suivante : Que se passe-t-il dans les dix ou vingt prochaines années ?

Pour répondre à cette question, il faut d'abord connaître les limites actuelles des réseaux de neurones artificiels et ensuite chercher les perspectives à venir :

En premier lieu, la modélisation du neurone biologique est un sujet de recherche ouvert ; car à l'heure actuelle et malgré les énormes progrès effectués en biologie, en microscopie optique et en imagerie médicale, notre compréhension du cerveau est encore très loin de la réalité. En effet, beaucoup de questions demeurent posées et de nouvelles découvertes sont venues remettre en question le modèle du neurone formel, qui paraît très simple par rapport à la complexité du neurone biologique.

En second lieu, il faut savoir que les concepts et leurs modélisations mathématiques ne sont pas suffisants ; à moins qu'il y ait une technologie permettant de les implémenter.

Ce dernier point nous amène à poser les deux questions suivantes :

1. Quelles sont les différentes technologies d'implémentation des réseaux de neurones ?
2. Ces technologies sont-elles suffisantes pour implémenter des modèles aussi complexes que le neurone ?
3. Pour répondre à ces questions, nous allons tenter d'aborder, dans le prochain chapitre, les différentes technologies d'implémentation des réseaux de neurones : état de l'art et perspectives.

---

# *CHAPITRE II*

---

**APERÇU SUR L'IMPLEMENTATION HARDWARE DES RESEAUX  
DE NEURONES: ETAT DE L'ART ET PERSPECTIVES**

## II. 1 Introduction

Les systèmes de calcul construits à base de réseaux de neurones forment la sixième génération des calculateurs ou ordinateurs. Les quatre premières générations des ordinateurs se distinguaient principalement par la technologie d'implémentation utilisée. Ils étaient conçus respectivement de tubes cathodiques, de transistors, de circuits intégrés LSI et enfin de circuits VLSI. Les troisième et quatrième générations représentent les machines parallèles. La cinquième génération représente les systèmes basés sur les connaissances, ils forment une combinaison de l'intelligence artificielle et du hardware parallèle. La sixième génération peut être vue comme l'intégration de la perception dans la conception du calculateur et la programmation à partir de la science cognitive et la neuroscience. On parle alors de *neurocalculateurs* ou « *neurocomputers* » et de « *neurochips* ».

La machine future doit réaliser un système de calcul coopératif qui intègre différents sous systèmes, chacun spécialisé dans sa structure et sa fonctionnalité. Un certain nombre de ces systèmes doit impérativement implémenter un grand nombre de réseaux de neurones qui interagissent en temps réel avec le monde extérieur.

Il apparaît clair, que les réseaux de neurones continueront à détenir une place importante dans les applications futures. Ceci étant, le concepteur de ces algorithmes se trouve souvent confronté au problème du choix du style d'implémentation software ou hardware des réseaux de neurones, qui joue un rôle déterminant sur les performances de l'application ciblée.

L'implémentation software considère les deux phases : apprentissage et généralisation des algorithmes neuronaux. De manière générale, 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 Von Newman dans lequel une seule instruction est exécutée à la fois. Les simulateurs de réseaux de neurones existants à l'heure actuelle, tel que Matlab Neural tool box [22], l'outil Sankom [23] et l'outil SNNS [24], 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 caractéristique aux réseaux de neurones n'est pas exploité. Pour palier à ces problèmes, dès le début des années 1980s, les chercheurs du domaine ont soutenu l'idée que l'implémentation dans un hardware spécial et dans lequel plusieurs éléments de traitement sont connectés en parallèles, représente un futur prometteur pour le développement et l'exploitation des capacités des réseaux de neurones.

Néanmoins, durant cette période, les travaux de recherche dans l'implémentation hardware des réseaux de neurones ont coïncidés avec la croissance fabuleuse de la puissance de calcul des processeurs conventionnels de la machine de Von Newman, tel le processeur Intel Pentium, qui a permis aux simulateurs software neuronaux d'atteindre un grand succès dans un large domaine d'applications. Les performances du processeur Intel continuaient à augmenter de façon extraordinaire. Entre temps, le développement et la commercialisation du hardware neuronal a été lent et a connu un succès très modeste à cette époque. Ceci est dû au fait qu'il n'existait pas un consensus clair sur la façon d'exploiter les capacités des technologies VLSI disponibles pour l'implémentation hardware massivement parallèle des réseaux de neurones. Une autre raison qui a freiné la construction du hardware neuronal par rapport au software durant cette période, trouve son explication sur le fait que le nombre et la variété des modèles neuronaux a connu une croissance rapide. Pour plusieurs modèles, les

performances sont difficilement connues et établis. Paradoxalement, ces performances ne peuvent être testées complètement, seulement lorsqu'un hardware dédié sera disponible.

Cette situation a amené certains chercheurs à poser la question suivante : Pourquoi l'implémentation hardware des réseaux de neurones ? Les réponses à cette question résident essentiellement sur la vitesse dans les réseaux de grande taille. En effet, même le processeur le plus rapide ne peut pas donner des réponses en temps réel et un apprentissage des réseaux de neurones de grande taille. D'autres facteurs tel que: la miniaturisation, la portabilité, le poids, la confidentialité et la dissipation de puissance appellent à une implémentation hardware dédiée. Partant de là, un nouveau regain d'intérêt a été donné quand à l'implémentation hardware des réseaux de neurones ; ainsi à partir du début des années 1990s, le nombre de réalisations qui ont eu un grand succès a augmenté de façon considérable.

Ce chapitre, présente un aperçu sur les différents travaux d'implémentation hardware des réseaux de neurones. D'abord les composantes essentielles constituant l'architecture du neurone, élément de traitement de base de tous les *neurocalculateurs* et *neurochips*, seront présentées. Cette section sera suivie par la présentation des différents types d'implémentation du hardware neuronal. Une attention particulière est donnée aux différentes approches de classification du hardware neuronale. Par la suite nous proposerons une nouvelle approche de classification du hardware neuronal qui sera suivie par des exemples de réalisations concrètes.

Certaines de ces réalisations sont déjà disponibles, d'autres présentent des études qui ont été menées par des groupes de recherches. Un grand nombre de ces études conceptuelles sont menées principalement par les Etat Unis, le Japon et l'Europe. Une synthèse permettant une vision sur l'état de l'art ainsi que les perspectives des travaux de recherches futures sur l'implémentation hardware des réseaux de neurones seront présentées..

## II.2 Architecture hardware de base des réseaux de neurones

De point de vue architectural, un réseau de neurone peut être vu comme un ensemble d'éléments processeurs interconnectés entre eux selon une topologie particulière et travaillant en concurrence. Chaque élément processeur implémente un neurone artificiel. Ce dernier représente l'élément de base de n'importe quelle implémentation hardware neuronale.

### II.2.1 Architecture hardware de base du neurone

A partir du modèle mathématique du neurone présenté dans le chapitre I, un modèle électronique a été développé. Une investigation dans la littérature a révélée que le bloc diagramme de la figure II.1 est adapté comme architecture de base ou modèle électronique équivalent au modèle mathématique du neurone artificiel [25], [26].

Le bloc de la fonction d'activation réalise la somme des produits de l'équation:  $P = \sum W_n X_n$

Ce Bloc est toujours implémenté à l'intérieur du circuit.

Les autres blocs, tel que l'état du neurone, le bloc des poids synaptiques et le bloc de la fonction de transfert peuvent être implémentés à l'intérieur ou bien à l'extérieur du circuit. Le calcul de ces derniers peut être réalisé par un ordinateur. On parle alors de « *on chip* » et « *off chip* » implémentation.



Le transfert des données entre les différents blocs est réalisé par le bloc de control qui est toujours implémenté « *on chip* ». Les paramètres de control sont utilisés pour le circuit par un ordinateur.

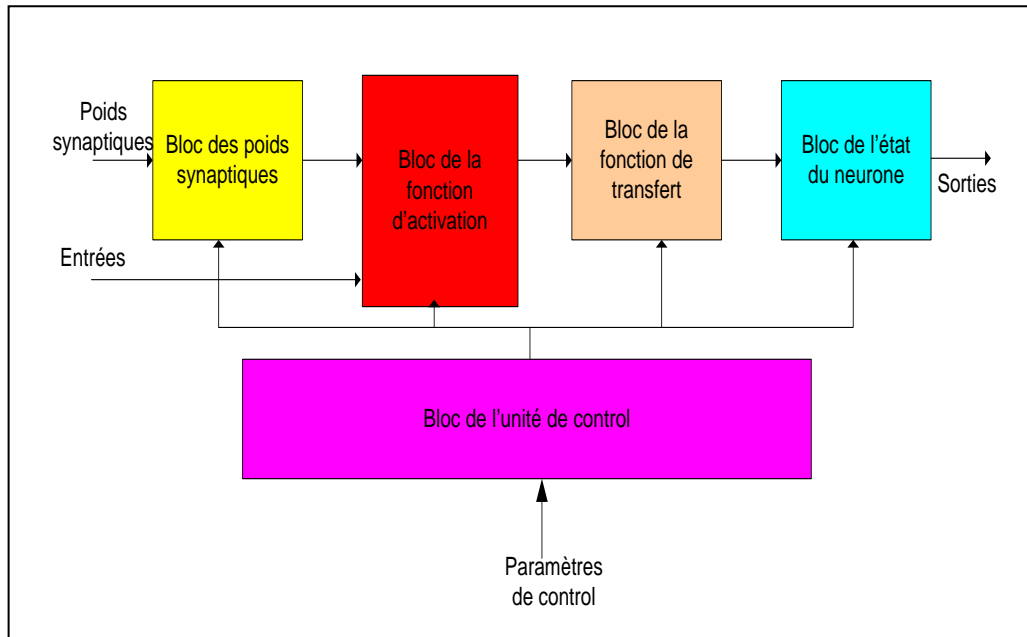


Figure II.1 Architecture de base du neurone artificiel

Pour le perceptron multicouche et le réseau de Hopfield], la fonction de transfert peut être représentée par une fonction seuil, une fonction en rampe ou bien la fonction sigmoïde. Pour la machine de Boltzmann, la fonction de transfert est représentée par une fonction seuil à laquelle on ajout un bruit. Pour le réseau de Kohonen], ce qui est calculé par le bloc d'activation correspond à la distance euclidienne entre les entrées et le vecteur des poids synaptiques. Le bloc de la fonction de transfert implémente l'opération de la recherche du minimum de toutes les distances euclidiennes calculées et la détermination des index qui dénotent les neurones dans les cartes auto-organisatrices.

Les états de neurones ainsi que les poids peuvent être stockés en digital ou bien en analogique. Les poids peuvent être chargés de manière statique ou bien dynamique.

## II.2.2 Mesures de performances

Afin de pouvoir comparer entre les différentes réalisations, les mesures de performances suivantes sont utilisées dans la littérature [27] :

- Le **MCPS** (Millions de Connexions par Seconde, pour le module propagation seulement).
- Le **MCUPS** (Millions de Connexion Update Par Seconde, pour les modules de rétropropagation et de mise à jour des poids synaptiques de l'algorithme RPG). Cette mesure de performance dépend de la taille du réseau.
- L'**efficacité** qui est calculée en divisant le nombre MCUPS par le nombre de poids synaptiques du réseau de neurones en question.

Dans ce qui suit un état de l'art des différentes implémentations du hardware neuronal est donné.

## II.3 Etat de l'art sur les différents types d'implémentation du hardware neuronal

Une multitude d'implémentation hardware des réseaux de neurones est citée dans la littérature en commençant par les PC et stations de travail jusqu'aux implémentations basées sur la technologie VLSI ou bien celles basées sur le calcul optique. Néanmoins et comme le montre la figure II.2, on est loin de l'objectif principal de la neuroscience. 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.

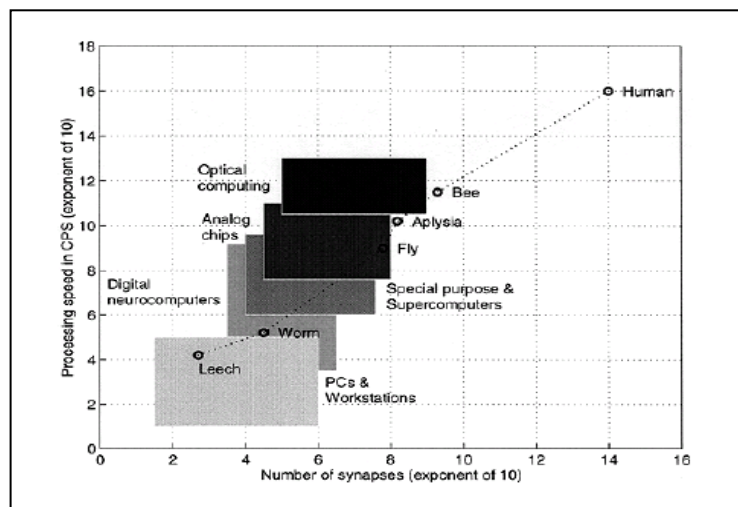


Figure II.2 Différents types d'implémentation hardware des réseaux de neurones. D'après la référence [28]

### II.3.1 Les différentes approches de classification du hardware neuronal

Afin de comprendre les caractéristiques de base relatives aux différentes implémentations hardware, il est nécessaire de faire une classification des différentes architectures et de définir des critères pour la classification de ces dernières.

Nonobstant, la classification du hardware neuronal est un problème qui est toujours posé. En effet, et vu la diversité des réalisations, plusieurs travaux de recherches se sont intéressés à proposer des approches pour la classification des différents neurocalculateurs et à l'heure actuelle, il n'existe toujours pas un consensus concernant les critères de classification.

A côté de la classification et des critères de classification, un autre problème des neurocalculateurs est celui de l'évaluation de leurs performances et les Benchmarks qui leurs sont associés.

Pour notre part, nous avons identifiés treize (13) approches de classification pluri-temporelle, de 1992 jusqu'à 2010 et que nous avons classé selon l'année de publication ainsi que l'auteur principal de l'article, comme suit (Figure II.3) :

- **1992** : Classification selon **NÖRDSTROM**
- **1994** : Classification selon **AARON FERRUCI**
- : Classification selon **GLESNER**
- **1995** : Classification selon **PAOLO IENNNE**
- : Classification selon **HEEMSKERK**
- **1996** : Classification selon **ISIK AYBAY**

- **1997** : Classification selon **BEIU**
- **1998** : Classification selon **T. SCHOENAUER**
- **2000** : Classification selon **SORIN DRAGHICI**
- **2002** : Classification selon **LINDSEY**
- **2004** : Classification selon **NICHOLS**
- **2009** : Classification selon **KRISTIAN NICHOLS**
- **2010** : Classification selon **SAUMIL G. MERCHANT**

Les sections suivantes sont consacrées à la présentation des différentes approches de classification des neurocalculateurs.



Figure II.3 Différentes approches de classification du hardware neuronal

### II.3.1.1 Classification selon NÖRDSTROM

**Présentation:** En 1992, Nördstrom & Svensson [29] ont effectué une étude sur l'utilisation des machines parallèles dans l'implémentation des réseaux de neurones. En partant des quatre classes architecturales définies par Flynn [30], à savoir SISD- SIMD- MISD et MIMD, Nördstrom & Svensson ont montré qu'une architecture de type SIMD est plus appropriée dans l'implémentation des réseaux de neurones dans les machines parallèles.

**Critères de classification:** la classification des neurocalculateurs est basée sur le nombre et la complexité des processeurs utilisés. Le nombre de processeurs,  $N$ , est lié au degré de parallélisme et est défini comme suit :

- **Massivement parallèle**,  $N \geq 2^{12}$
- **Hautement parallèle**,  $2^8 < N \leq 2^{12}$
- **Modérément parallèle**,  $2^4 \leq N \leq 2^8$
- **Légèrement parallèle**,  $2 < N \leq 2^4$

La complexité des processeurs est liée à la communication entre les processeurs dans une architecture SIMD à savoir : réseau systolique, réseau en anneau, réseau en maille et aussi au type du processeur utilisé : DSP, implémentation VLSI ou autre. La figure II.4 montre les différentes classes utilisées.

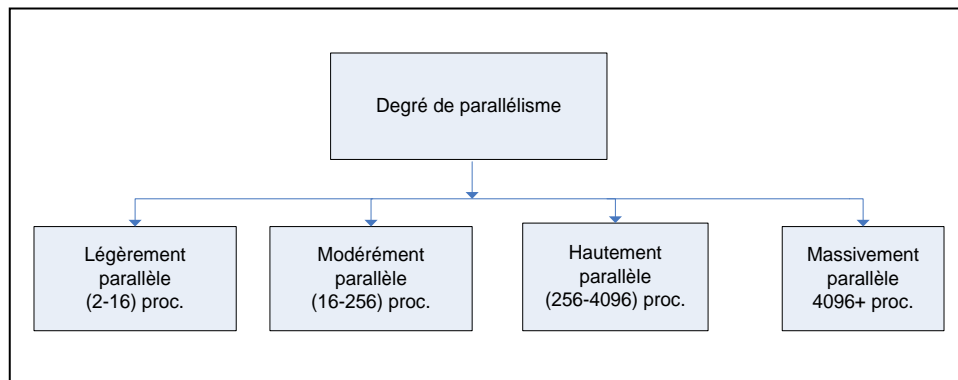


Figure II.4. Classification du hardware neuronal selon NÖRDSTROM

**Exemples :**

- Dans la classe des implémentations *massivement parallèles*, les implémentations suivantes sont citées : AAP-2 [31], CM-2 [32], Mas-Par (MP-1) [33] et DAP [34]
- Dans la classe des implémentations *hautement parallèles*, les implémentations suivantes sont citées : REMAP [35], L-Neuro [36], CNAPS [37], GF11 [38], et UCL-Neurocomputer [39]
- Dans la classe des implémentations *modérément parallèles*, les implémentations suivantes sont citées : Sandy/8 [40], RAP [41] et Warp [42].
- Dans la classe des implémentations *légèrement parallèles*, les implémentations suivantes sont citées: Hitachi-WSI [43] et Siemens [44].

**Avantages et Inconvénients :** L'avantage de la classification proposée par Nördstrom & Svensson est qu'ils ont défini et quantifié le degré de parallélisme en fonction du nombre de processeurs. Cependant, les autres propriétés architecturales, telle que la conception analogique, digitale, hybride ou DSP n'apparaissent pas.

**II.3.1.2 Classification selon AARON FERRUCI**

**Présentation :** Cette approche a été proposée par AARON FERRUCI au Début de l'année 1994 [45]. Ce dernier a défini les critères de classification suivants (Figure II.5):

**Critères de classification**

- **Représentation des données :** Les valeurs des poids synaptiques ainsi que les entrées/sorties peuvent être de type analogique, digital ou stochastique. Dans le cas des données digitales les représentations en virgule flottante ou en point fixe sont utilisées.
- **Stratégie d'interconnexion :** qui peut être de type SIMD ou MIMD
- **Précision :** qui varie d'une implémentation à une autre selon l'application ciblée.
- **Technologie :** qui peut être de type VLSI, DSP, FPGA ou bien WSI (Wafer Scale Integration)
- **Mapping de l'algorithme :** représentant le degré de parallélisme utilisé. Trois types de parallélismes sont utilisés : le parallélisme des neurones, le parallélisme des synapses et le parallélisme des couches.

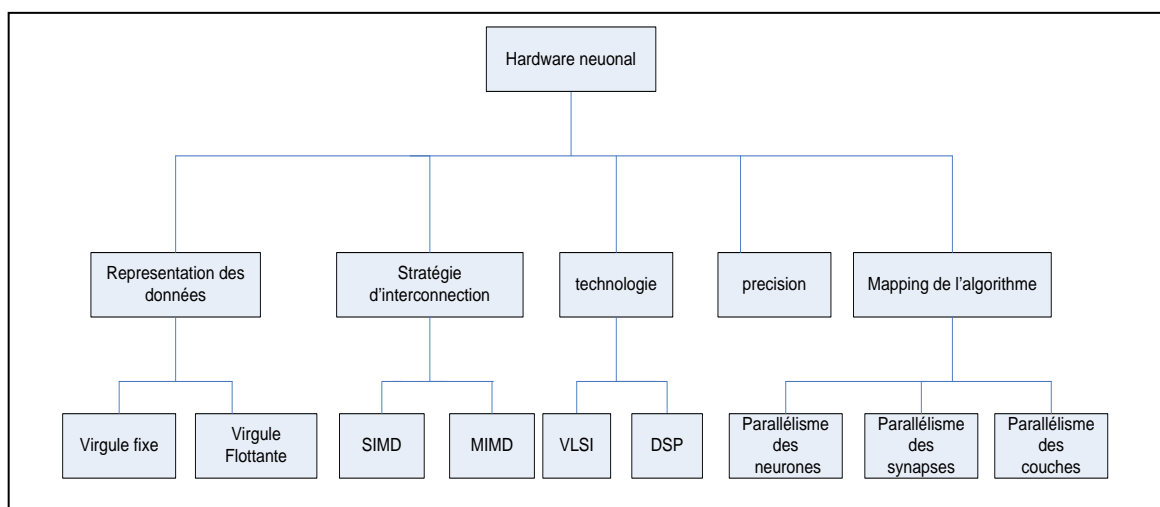


Figure II.5 Classification du hardware neuronal selon Aron Ferruci

**Exemples :**

- Dans sa thèse, Aron Ferruci s'est intéressé à la classification des neurocalculateurs suivants : CM-2 [32], CNAPS [37], GF11 [38], Hitachi-WSI [43], IPSC/80 [46], RICHO-LSI [47], SPERT [48] et GANGLION [49].

**Avantages et Inconvénients:**

L'inconvénient majeur de cette étude est que l'auteur s'intéresse uniquement à la classification des implémentations hardware spécifiques à de l'algorithme de la rétropropagation du gradient.

**II.3.1.3 Classification selon GLESNER**

**Présentation:** Une classification complètement différente a été proposée par Glesner et Pöchmüller [50], [51], en 1994, basée sur les critères suivants :

**Critères de classification**

- **L'évidence biologique** qui représente le degré d'imitation des systèmes biologiques
- **Le mapping du hardware :** qui considère le parallélisme des neurones, le parallélisme des synapses ou bien le parallélisme des couches
- **La technologie d'implémentation** qui peut être digitale, analogique ou hybride

**Exemples:** Aucun exemple n'a été cité dans l'article.

**Avantages et inconvénients :**

Les auteurs ont présenté un nouveau critère de classification. Cependant, nous n'avons pas pu recueillir plus d'informations sur le sujet.

**II.3.1.4 Classification selon PAOLO IENNE**

**Présentation:** En 1995, Paolo Ienne [52] a proposé une classification du hardware neuronal selon les deux critères suivants à savoir : la flexibilité et les performances.

**Critères de classification:**

- Les implémentations sur les **systèmes séries** très flexibles.
- Les implémentations **parallèles sur les circuits standards**
- Les implémentations sur les **systèmes parallèles VLSI spécifiques** à un seul type d'algorithme neuronal
- Les implémentations sur les **systèmes parallèles VLSI programmables** dédiés à plusieurs types d'algorithmes neuronaux.

La figure II.6 montre les différentes classes des neuro-calculateurs proposés par Paolo Ienne.

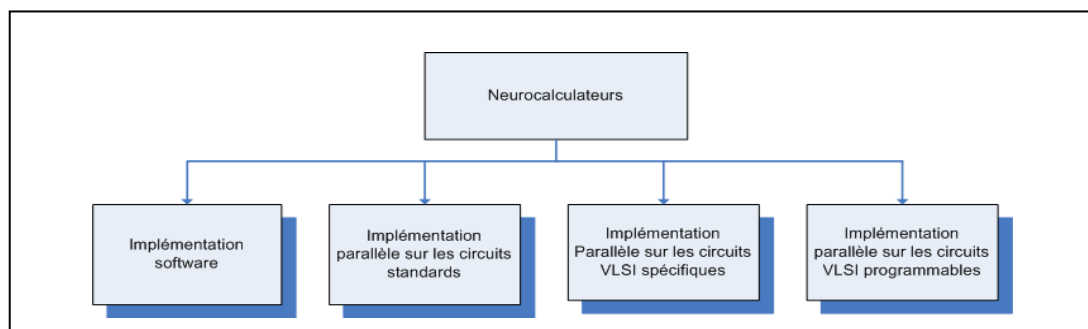


Figure II.6 Classification des neurocalculateurs selon PAOLO IENNE

**Exemples :**

- Les *systèmes séries* sont utilisés pour désigner les implémentations logiciels des réseaux de neurones sur un PC ordinaire, une station SUN ou bien celles utilisant une carte accélératrice DSP de type TMS320. Un exemple de simulateur logiciel a été utilisé par la compagnie générale des Eaux (Paris) depuis Septembre 1992. L'outil en question a permis une réduction de 30% de l'erreur de prédiction de la demande en eau [53].
- Les systèmes *parallèles sur les circuits standards* sont utilisés pour désigner les implémentations utilisant plusieurs processeurs d'application générale « general pupose » montés en parallèles. Des exemples d'implémentations cités dans cette catégorie sont : le simulateur SPRINT [54] qui est un processeur d'application générale conçu vers la fin des années 1980s et qui a été adapté aux réseaux de neurones. Le problème de ce genre d'implémentations est que plus le nombre de processeurs augmente plus la communication entre ces derniers devient lente.
- Un deuxième exemple cité par l'auteur est le simulateur MUSIC « MULTI-processor System with Intelligent Communication » [55], conçu spécialement pour simuler l'algorithme RPG. Ce système supporte 60 processeurs connectés entre eux et atteint jusqu'à 330 MCUPS. Ce qui le rend très performant néanmoins son prix n'est pas accessible.
- Les systèmes *parallèles VLSI* sont utilisés pour désigner les implémentations qui exploitent le parallélisme large des réseaux de neurones. Dans cette catégorie l'auteur distingue entre :
  - Les *systèmes VLSI spécifiques à un seul type d'algorithmes* tel que le circuit Ni1000 commercialisé par Intel [56], le circuit Richo-LSI [47] et le circuit Hitachi-WSI « Wafer cale Integration » développé par Hitachi [43];
  - Les *systèmes VLSI programmables* peuvent prendre en charge l'implémentation de plusieurs types d'algorithmes neuronaux tel que le circuit CNAPS [41], SYNAPSE [57] et Mantra-I [58].

**Avantages et inconvénients:**

En résumé, la classification selon PAOLO IENNE avait pour but de distinguer les implémentations flexibles de celles performantes. L'auteur recommande une orientation vers un compromis entre la flexibilité et les performances. Cependant, la classification concerne le domaine digital uniquement.

**II.3.1.5 Classification selon HEEMSKERK**

**Présentation:** Durant la même année, 1995, Jan N. H. HEEMSKERK [59] a proposé une nouvelle classification comme le montre la figure (Figure II.7):

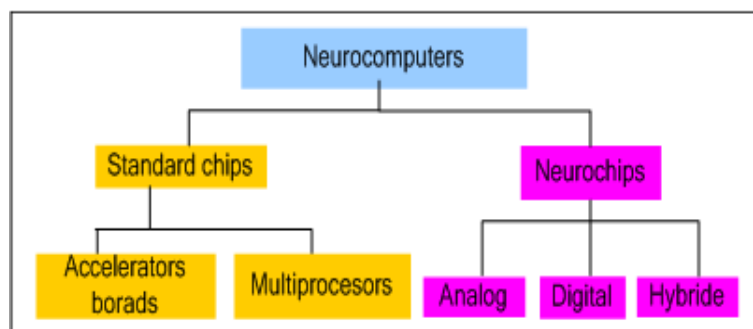


Figure II.7 Classification des neurocalculateurs selon HEEMSKERK

**Critères de classification :**

- Implémentation sur les circuits « **Standards chips** » dans laquelle on distingue les *cartes accélératrices* et les *circuits multiprocesseurs*
- Implémentation sur les « **Neurochips** » dans laquelle on distingue les implémentations de type *analogiques*, *digitales* et *hybrides*.
- HEEMSEK s'est basé sur les mesures de performances **MCPS** et **MCUPS** pour comparer entre les différentes implémentations disponibles à cette époque.

**Exemples :**

- Dans la classe des *cartes accélératrices* les réalisations ci-dessus sont citées : Le co processeur ANZA - PLUS développé par « Heicht Neilson Cooperation » (USA) [60]. Le NT600 [61] développé par la compagnie Neural Technologies limited et California Scientific Software), le circuit Intel Ni1000 [53] développé par Intel (USA), la carte IBM NEP développée par la compagnie IBM [62], et les circuit Neuro turbo-I et Neuro turbo-II développés par l'institut Nagoya de technologie (Japan) [63].
- Dans la classe des *neurocalculateurs* construits à partir des processeurs et *multiprocesseurs standards*, les réalisations suivantes : le système WISARD développé par le collège Impérial de Londres (UK) [64], **Sandy/8** développé par le laboratoire Fujitsu (Japan) [40], le co-processeur COKOS (university of Tübingen, Allemagne) [65], le système TI-NETSIM développé par Texas Instrument et l'université de Cambridge (UK) [66]
- Dans la classe des *Neurochips* les réalisations suivantes sont citées : Le circuit SYNAPSE réalisé par Siemens (Allemagne) [57], [67], CNAPS (Adaptative solutions, USA) [37], Hitachi-WSI (Japan) [43], L-Neuro 1.0 (Neuromimetic chip, France) [36], UTAK1 réalisé par l'université de Catalogne (Espagne) [68], le circuit Mantra-I réalisé par l'institut Fédéral Suisse de technologie [58] et le processeur IBM ZISC036 réalisé par « IBM Micro Electronic » en France [69].

**Avantages et inconvénients :**

L'auteur a proposé de nouveaux critères de classification qui permettent de couvrir un large domaine d'implémentations du hardware neuronale (analogique, digital et hybride). Cependant l'approche de classification proposée est très générale et ne montre pas les techniques de conception de chaque classe. Enfin, l'auteur propose une orientation vers la conception de systèmes hybrides, rapides et à grande densité d'intégration.

**II.3.1.6 Classification selon ISIK AYBAY**

**Présentation:** En 1996, Isik Aybay [70], a développé une analyse dans laquelle il remarque que les approches de classifications existantes ne font pas apparaître toutes les caractéristiques d'une implémentation hardware. Certaines caractéristiques peuvent être perdues et le lecteur passe beaucoup de temps pour comparer entre deux implémentations hardwares. Ce qui l'a conduit à proposer la classification selon les critères suivants (Figure II.8):

**Critères de classification :**

- **Type du hardware neuronal** qui peut être un « *Neurochip* » (*NC*) ou bien un « *Neurocomputer* » (*NS*)

- **Type d'implémentation** qui désigne si le circuit implémente plusieurs algorithmes neuronaux « **General Purpose** » ou bien un algorithme spécifique « **Special Purpose** »

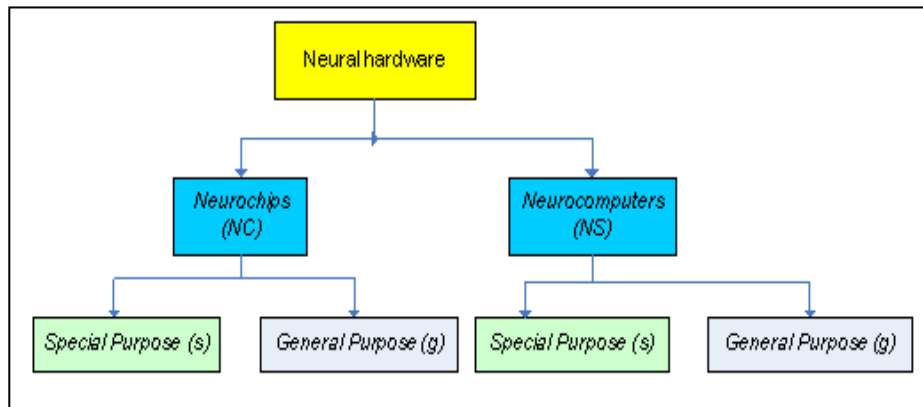


Figure II.8 Classification des neurocalculateurs selon Isik Aybay

En plus de ces critères l'auteur définit plusieurs autres attributs qu'il utilise pour la classification, à savoir :

- Le **signal d'entrée (I)** : digital (**d**), analogique (**a**), hybride (**h**)
- Le **bloc d'activation (A)** : digital (**d**), analogique (**a**)
- Le **stockage des poids synaptiques (W)** : digital (**d**), analogique (**a**)
- Le **stockage des neurones (S)** : digital (**d**), analogique (**a**)
- La **fonction de transfert (T)** : digital (**d**), analogique (**a**)
- Le **type d'apprentissage (L)** : « on- chip » (**o**), ou bien « off-chip » (**f**)

#### **Exemples**

- A titre d'exemple, **CNAPS** [37] est un « **neurocomputer general purpose** (NSgd) possédant les caractéristiques suivantes : Id/Ad/Wdo/Sdo/Tdo/Lo.
- Le circuit **ETANN** [71], est un Neuro-Chip hybride possédant (NCgh) possédant les caractéristiques suivantes: Ia/Aa/Wdo/Sao/Tao/Lf.

#### **Avantages et inconvénients**

Comparée aux approches de HEEMSEK et de PAOLO IENNE, l'approche d' ISIK AYBAY pour la classification du hardware est complémentaire car elle introduit de nouveaux attributs qui mettent en évidence les caractéristiques architecturales du hardware en question. Néanmoins elle ne donne pas importance aux mesures de performances.

#### **II.3.1.7 Classification selon BEIU**

**Présentation:** Vu la diversité des réalisations, BEIU a proposé une classification historique des neurocalculateurs, basée sur les critères suivants [72] :

##### **Critères de classification:**

- **Date de publication**
- **Evolution technologique**

##### **Exemples :**

BEUI a commencé par le premier neurocalculateur historique, **MARK-III** paru en 198 et en passant par les architectures à base de cartes accélératrices, de circuits DSP, de



transputers, de réseaux systoliques et du processeur RISC. A cet effet, une cinquantaine de réalisations des neurocalculateurs ont été classé par l'auteur.

### *Avantages et inconvénients*

Un grand effort de recherche a été fourni pour la classification du hardware neuronal. Cependant, l'étude concerne seulement le domaine digital. Aucun exemple d'implémentation neuronale analogique n'a été fourni par l'auteur.

### II.3.1.8 Classification selon SHCOENAUER

**Présentation :** En 1998, T. Schoenauer & al. [72], ont proposé une classification basée sur les critères suivants (figure II.9) :

#### *Critères de classification :*

- **Type du hardware neuronal « neurohardware »** qui peut être : un « *Neurochip* », une « *Carte accélératrice* » ou bien un « *Neurocomputer* »
- **Strategie d'interconnexion des processeurs** qui peut être : « *Bidimensional Mesh* », « *Systolic Ring* », « *Systolic Ring with GlobalBbus* », « *Broadcast Bus* », « *Linear Array* » ou « *Crossbar* »
- **Degré de parallélisme** qui peut être à grain large « *Coarse- Grain* », à grain moyen « *Medium Grained* », à grain fin « *Fined Grain* » et « *Massive* »
- **Représentation des données** qui peut être fixe, en virgule flottante ou autre
- **Partition des processeurs :** par synapse, neurone, sous réseau ou réseau

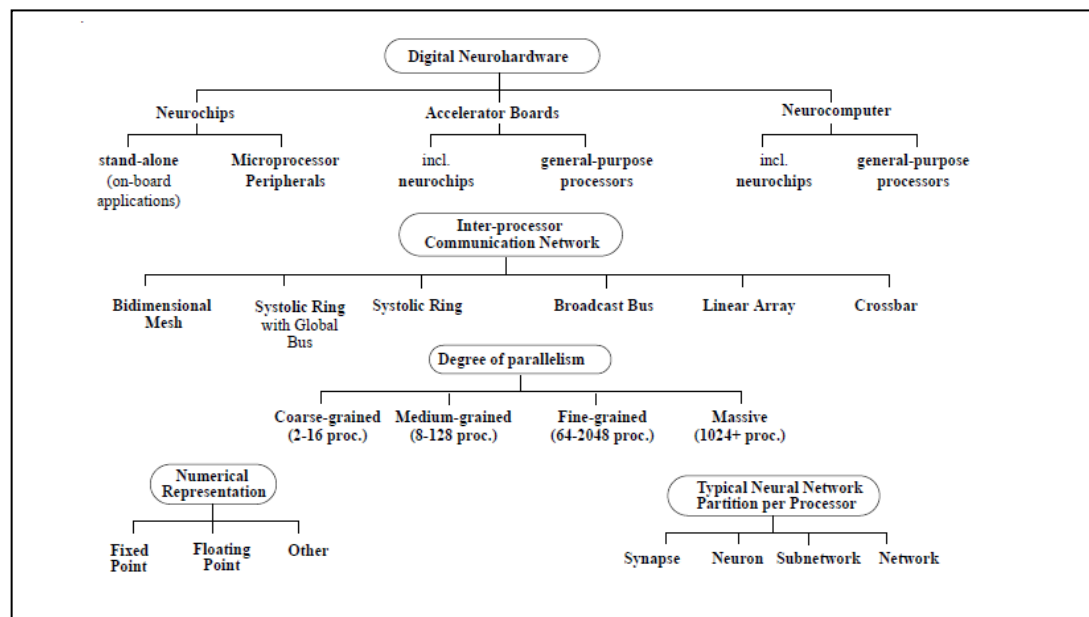


Figure II.9 Classification des réseaux de neurones selon T. Shcoenauer [69]

#### *Exemples :*

Trois exemples d'implémentation hardware ont été cités par l'auteur, à savoir : CNAPS [37], SYNAPSE-I [57], [67] et NESPINN [74].

#### *Avantages et inconvénients*

D'après les critères cités ci-dessus, la classification proposée par T. Shcoenauer met l'accent sur les implémentations hardware faisant appel à plusieurs processeurs

communiquant entre eux. Les aspects de conception analogique et hybride n'apparaissent pas.

### II.3.1.9 Classification selon SORIN DRAGHICI

**Présentation:** En 2000, Draghici [75] proposa une classification comme suite (Figure II.10) :

#### **Critères de classification :**

- **Type de l'implémentation** qui regroupe les classes suivantes : *Analogique, Digitale Mixed, Optical, Pulse Stream, RAM*
- **Paradigme d'interconnexion des neurones** qui regroupe les classes suivantes : « *layered-architectures* », « *fully-interconnected-networks* », « *locally-connected-networks* », « *reconfigurables-architectures* »,
- **Type de l'architecture** : qui regroupe les classes suivantes : « *On-chip-controller* », « *Stand-alone-computer* » et « *Accelerator boards* »
- **Degré de parallélisme** : qui regroupe les classes suivantes : « *Low-level-parallelism* », « *Medium-parallelism* », « *High-parallelism* », « *Massive-parallelism* »
- **Objectif de l'implémentation** : qui regroupe les classes suivantes : « *Model free learning* », « *Specific algorithm implementation* », « *Emulation of special biological functions* »
- **Partition des processeurs** : par *synapse, réseau* ou *sous réseau*
- **Type de l'apprentissage** : « *on-chip* », « *off-chip* » et « *chip-in-the-loop* »

#### **Exemples :**

Plusieurs références ont été cités pour chaque critère de classification parmi lesquels : le circuit ETANN [71], [76], ART-1 [77], le circuit PLN « Probabilistic Logic Node » [78], le circuit GSN « Goal Seeking Neuron » [79], le circuit PRAM « Probabilistic RAM » [80], [81], [82], [83], [84], [85]

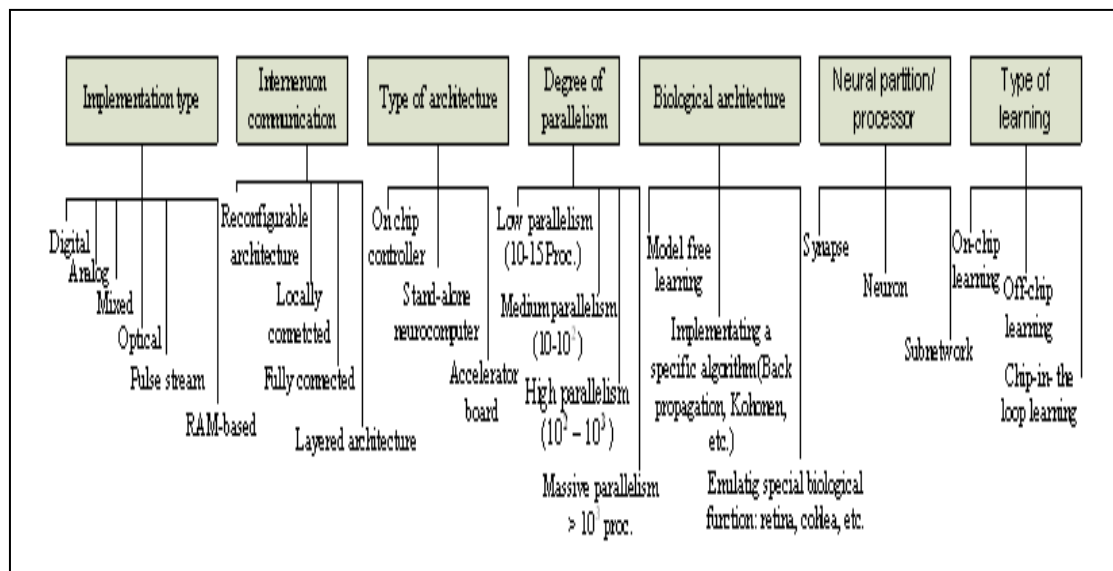


Figure II.10 Classification selon SORIN DRAGHICI

### **Avantages et Inconvénients**

L'auteur a présenté plusieurs critères de classification du hardware neuronal, cependant et contrairement aux approches citées précédemment, il ne s'est intéressé qu'aux implémentations de type analogique. De plus, il n'existe pas une barrière claire entre les différentes classes qu'il a cité ; à titre d'exemple le circuit dans la référence [81] a été considéré aussi bien pour la classe « Paradigme d'interconnexion des neurones » que pour la classe « objectif d'implémentation »

### II.3.1.10 Classification selon Clark Lindsey

**Présentation :** En 2002, Lindsey et Thomas Lindbald [86] ont proposé une classification du hardware neuronal selon les critères suivants (Figure II.11) :

**Critères d'implémentation :**

- **VLSI Implementation** dans laquelle sont classées les implémentations de type analogique, digital et hybride. Dans le cas des circuits digitaux, les architectures suivantes sont citées : Architecture Slice, Architecture à base de multiprocesseurs et les fonctions à base radial.
- **Accelerator boards and Neurocomputers** »

**Exemples :**

- Dans la classe **VLSI Implementation** les circuits suivants sont cités : CNAPS [37], IBMZISC036 [69], Intel ETANN [71], Nestor Ni1000 [56], Philips L-Neuro [36], Richo RN-200 [87] et autres. Tous ces circuits sont classés selon le type d'implémentation analogique, digitale ou hybride
- Dans la classe **Accelerator boards and Neurocomputers**, les réalisations suivantes sont citées : Accurate Automation and ISA Cards [88], Adaptative solutions CNAPS-ISA [89], NESTOR PCI and VME Cards [90], BrainMaker Accelerators [91] , Synapse [57], [67] et Siemens MA-16 chip [92].

**Avantages et Inconvénients**

- Clark Lindsey utilise le type d'apprentissage, le type d'architecture, la précision, le nombre de neurones, le nombre de synapses ainsi que les mesures de performances : (MCPS) et (MCUPS) pour comparer entre les différentes implémentations recensées.
- Il propose une classification similaire à celle de Heemskerk, néanmoins il utilise la terminologie « VLSI Implémentation » au lieu de « NeuroChip » définie par Heemskerk.

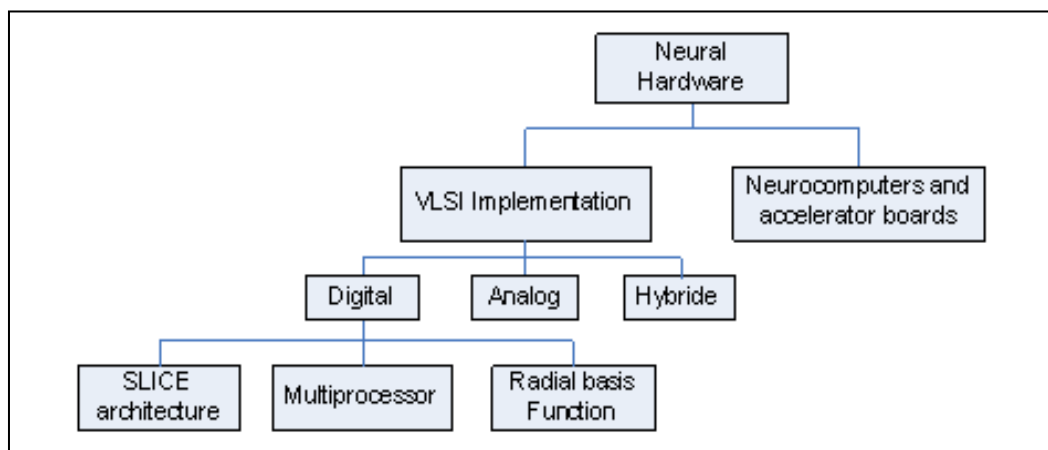


Figure II.11 Classification selon *Clark Lindsey*

### II.3.1.11 Classification selon KRISTIAN NICHOLS

*Présentation :* En 2004, KRISTIAN NICHOLS [93] proposa une classification du hardware neuronal selon les critères ci-dessous (Figure II.12):

*Critères de classification :*

- **Algorithme d'apprentissage** qui désigne d'une part, si pour un algorithme donné, l'apprentissage est de type « *on chip* » ou « *off chip* » et d'autre part si le circuit peut implémenter un seul type d'algorithme ou bien plusieurs types d'algorithmes. Dans cette classe les travaux suivants sont cités :
- **Représentation du signal** qui désigne les implémentations utilisant une représentation en virgule fixe ou bien celles utilisant une représentation en train d'impulsion « Spike Train ».
- **Type du multiplieur :** qui désigne les différentes architectures des multiplieurs utilisés pour l'implémentation hardware des réseaux de neurones. Pour cela l'auteur cite cinq types d'architectures qui sont utilisées pour l'implémentation des multiplieurs dans les réseaux de neurones, à savoir : le multiplieur bit-série « *Bit-serial multiplier* », le multiplieur *Pipeline*, *élimination du multiplieur par un circuit équivalent*, utilisation du *multiplexage temporel* et utilisation des *neurones virtuels*.
  - ✓ *Le multiplieur bit-série* utilise un seul bit à la fois. Dans ce genre d'implémentation, le temps de multiplication croît quadratiquement,  $O(n^2)$ , avec la taille du signal donné.
  - ✓ L'utilisation de multiplieur *Pipeline* permet de lever ce problème. *L'élimination du multiplieur* est obtenue en utilisant une certaine représentation du signal qui permet de remplacer le multiplier par une simple fonction logique.
  - ✓ *Le multiplexage temporel* est utilisé pour réduire le nombre de multiplieurs requis pour l'implémentation d'un algorithme neuronal.
  - ✓ Le concept des *neurones virtuels* est identique au concept de mémoire virtuel dans le cas des ordinateurs. Pour cela, une carte de prototypage FPGA est d'abord choisie comme plateforme de base, ensuite tous les paramètres du réseau de neurone (poids synaptiques, entrées/sorties, fonction d'activation, etc.) sont stockés dans des mémoires externes et on implémente le reste de l'architecture (principalement le multiplieur) dans le circuit FPGA. L'avantage de cette approche est que le nombre maximal de neurones qui peuvent être supportés dépend seulement de la taille des mémoires externes disponibles et non des neurones résidant dans le circuit FPGA. Néanmoins, le temps perdu pour accéder aux mémoires constitue une contrainte sérieuse qu'il faut prendre en considération.

*Exemples :*

- Dans la classe « **Algorithme d'apprentissage** » les circuits suivants sont cités : **RRANN** [94], **RENCO** [95], **ACME** [45], **FAST** [96] et **REMAP** [67]
- Dans la classe **Représentation du signal**, les travaux suivants sont cités : **CAM-Brain machine** [97], **ECX card** [98], **RRANN** [94] et les travaux de **Holt & Baker** [99].
- Dans la classe **Type du multiplieur**, les références suivantes sont citées : [100], [101], [35], [96], [102], [94], [95], [97], [98] et [103].

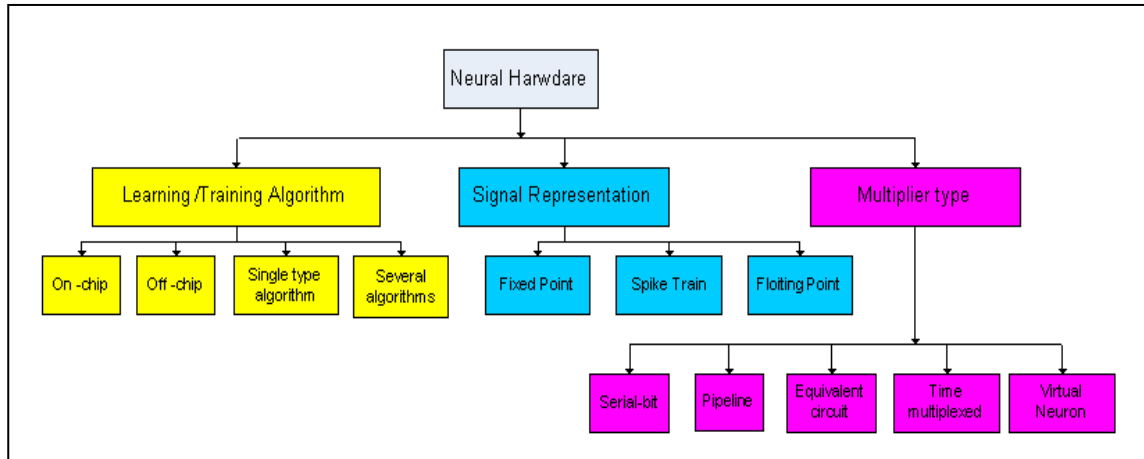


Figure II.12 Classification des neurocalculateurs selon KRISTIAN NICHOLS

**Avantages et Inconvénients :**

Dans son approche, NICHOLS a introduit un nouveau critère de classification correspondant au type du multiplieur utilisé. Néanmoins, il s'intéresse à la classification des différents types d'implémentations hardware des réseaux de neurones, sur un support FPGA uniquement.

**II.3.1.12 Classification selon VIPAN KAKKAR**

**Présentation :** En 2009, Vipin Kakkar proposa une étude comparative entre les implémentations analogiques, digitales et mixtes des réseaux de neurones [104]. Ces dernières furent classées comme suit :

**Critères de classification**

- Implémentation biologique
- Implémentation optique
- Implémentation électrique

Dans la classe de l'implémentation biologique, on retrouve les implémentations de type analogique et digital. La comparaison entre ces dernières était basée sur les critères suivants : consommation de puissance, surface et stockage des poids synaptiques. La figure II.13 montre l'approche proposée.

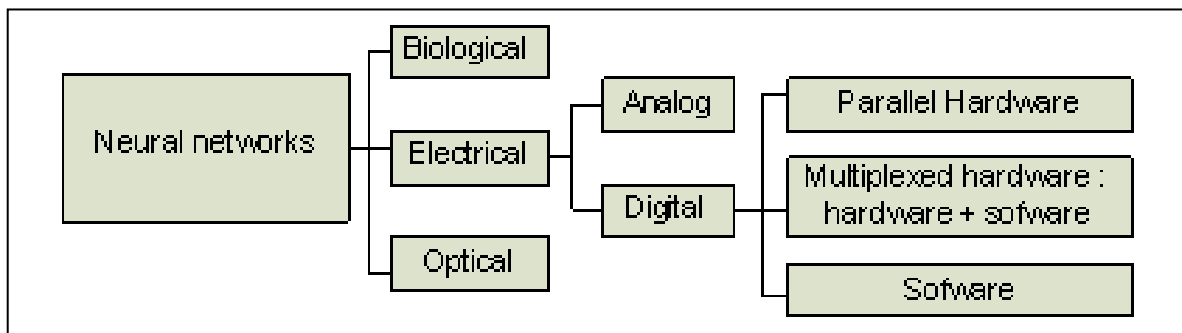


Figure II. 13 Classification du hardware neuronal selon VIPAN KAKKAR

**Exemples**

Les exemples d'implémentation n'ont pas été clairement présentés par l'auteur.

**Avantages et Inconvénients**

Les techniques de conception analogique, digitale et mixte ont été considérées. Néanmoins, les récents développements de conception, tel que les systèmes sur puce et les systèmes embarqués ne sont pas pris en considération par l'auteur.

**II.3.1.13 Classification selon SAUMIL G. MERCHANT**

**Présentation :** En 2010, SAUMIL G. MERCHANT et Peterson [105] proposèrent une approche de classification comme suit (figure II.14) :

**Critères de classification**

- Implémentation digitale
- Implémentation analogique
- Implémentation hybride ou mixte

Les implémentations de type digital ont été divisées en classe FPGA et ASIC. Aussi dans cette catégorie, de sous classes ont été définies par rapport au paramètres de conception tel que : représentation des données, flexibilité de conception, apprentissage on-chip/off-chip et implémentation de la fonction de transfert.

**Exemples**

- Dans la classe implémentation digitale, les références aux circuits suivants sont citées : [106], SYNAPSE [34], [107], CNAPS [37], MY-NEUROPOWER [108].
- Dans la classe implémentation analogique, les références suivantes sont citées : [109], [110], ETANN [71], [111].
- Dans la classe implémentation hybride, les références suivantes sont citées : [112], [113]

**Avantages et Inconvénients**

Les trois types d'implémentation analogique, digitale et mixte sont considérés par l'auteur. Dans la classe implémentations, digitale, les paramètres de conception liés l'implémentation des ASICs et FPGA, tel que la flexibilité, représentation des données, etc. sont pris en considération par l'auteur. Cependant aucune indication n'est donnée à la conception analogique ou mixte. Aussi, les nouvelles techniques et approches de conception telle que les sur puce et systèmes embarqués ne sont pas pris en considération.

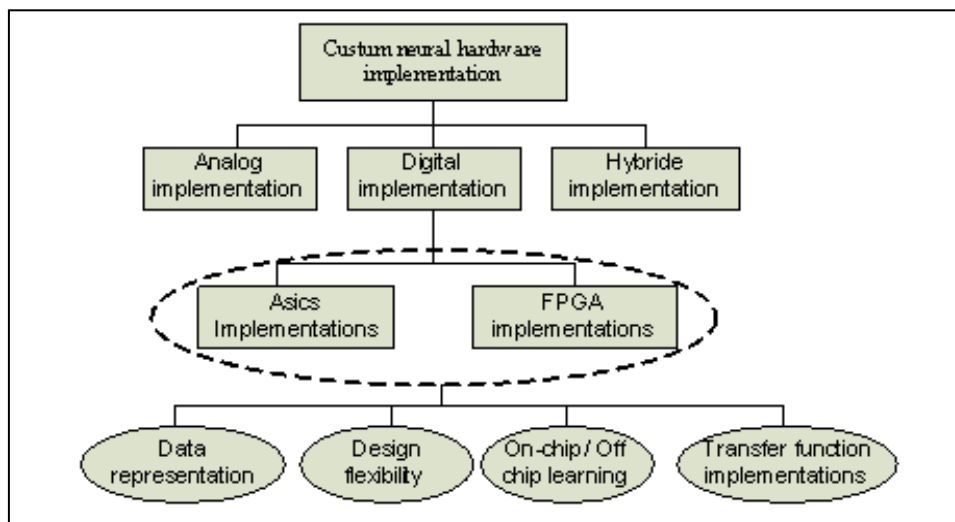


Figure II.14 Classification du hardware neuronal selon SAMUEL G. MERCHANT

### II.3.2 Synthèse des différentes approches de classification

Sur la base des différentes approches de classification déjà citées, il en ressort les points suivants :

- Dans la première approche de classification, les travaux de Nordstrom, ont permis de quantifier quatre niveaux de granularité qui peuvent être utilisés pour une implémentation parallèle efficace d'un réseau de neurone sur des machines massivement parallèles. Sa classification a permis de comparer entre les différentes réalisations disponibles à cette époque. Néanmoins, l'auteur ne s'est intéressé qu'à la classification des implémentations digitales des réseaux de neurones sur les machines parallèles.
- Aron Ferruci a introduit de nouveaux critères pour la classification telle que la représentation des données, la précision, la technologie utilisée et le mapping de l'algorithme. L'inconvénient majeur de cette étude est que l'auteur s'intéresse uniquement à la classification des implémentations hardware spécifiques à l'algorithme de la rétropropagation du gradient.
- Glesener a introduit un nouveau critère qui est l'évidence biologique. Malheureusement nous n'avons pas réussi à recueillir assez d'information sur son travail.
- Paolo Ienne a défini la classification selon les critères de flexibilité et de performances du hardware neuronal. Sa classification a permis de distinguer entre les systèmes spécifiques dédiés à un seul type d'algorithme et les systèmes programmables dédiés à plusieurs types d'algorithmes. Néanmoins, l'approche de classification proposée, concernait l'implémentation digitale uniquement.
- Hemskerk a introduit les notions de « standard chip » et de « Neurochip ». Sa classification était plus générale, car contrairement aux approches précédentes, les implémentations analogiques et hybrides sont prises en considération dans la classification.
- Isik Aybay a distingué entre les « Neurochip » et les « Neurocomputers ». Aussi, il a introduit de nouveaux attributs qui mettent en évidence les caractéristiques architecturales du hardware en question. Néanmoins son approche ne semble pas donner une importance aux mesures de performances.
- Beiu a proposé une classification selon la date d'apparition du hardware neuronal et son évolution technologique. Néanmoins, l'auteur ne s'est intéressé qu'aux implémentations digitales. Les réalisations de type analogiques et hybrides (analogiques-digitales) n'apparaissent pas dans la classification de Beiu.
- Schonauer a proposé une classification dans laquelle il a rassemblé l'ensemble des critères proposés par Isik Aybay, Aron Ferruci et Nordstrom. Néanmoins, sa classification ne met l'accent que sur les implémentations hardware faisant appel à plusieurs processeurs communiquant entre eux. Les aspects de conception analogique et hybride n'apparaissent.
- Sorin Draghici, a présenté plusieurs critères pour la classification du hardware neuronal ; Cependant et contrairement aux approches précédentes, il ne s'est intéressé qu'aux implémentations de type analogique. De plus, il n'existe pas une barrière claire entre les différentes classes proposées.
- Clark Lindsey a donné une classification semblable à celle de Hemskerk ; cependant il a utilisé la terminologie « *Implémentation VLSI* » au lieu du terme « *Neurochip* » utilisé par Hemskerk dans sa classification. Aussi, sa classification ne fait pas ressortir les nouvelles approches et techniques de conception VLSI.

- Nichols a introduit de nouveaux critères liés à l'algorithme d'apprentissage tel que le « *on chip learning* » et le « *off chip learning* ». Il a aussi utilisé le « *type de multiplieur* » comme critère de classification. Néanmoins, son étude porte sur les implémentations sur circuit FPGA uniquement.
- Vipin Kakkar a considéré les techniques de conception analogique, digitale et mixte. Néanmoins, les récents développements de conception, tel que les systèmes sur puce et les systèmes embarqués ne sont pas cités par l'auteur.
- Saumil G. Merchant Les trois types d'implémentation analogique, digitale et mixte. Néanmoins, les récents développements de conception, tel que les systèmes sur puce et les systèmes embarqués ne sont pas cités par l'auteur.

En résumé, si on fait une synthèse des différentes approches, on s'aperçoit qu'il n'existe pas un consensus clair sur les critères de classification ainsi que sur les définitions mêmes de ces critères. A titre d'exemple, Aron Ferruci utilise le terme « *stratégie d'interconnexion* » pour désigner les architectures de types SIMD, MIMD, Ring, Systolic, etc. Alors que Nordstrom utilise le terme « *complexité des processeurs* » et T. Shchoenauer utilise la terminologie « *Inter-processor communication Network* ». Il en est de même pour le terme « *neurocomputer* » qui désigne soit une implémentation « *standard chip* » ou un « *neurochip* » chez Heemskerk ; alors que chez T. Schoenauer le même terme désigne une sous classe du « *neurohardware* ». Le terme neurocomputer désigne dans ce cas un système construit d'un software et d'un hardware.

Ceci étant, chaque approche a apporté un nouveau critère de classification par rapport à sa précédente en fonction du domaine d'intérêt de l'auteur.

L'examen de l'ensemble des approches permet de conclure qu'elles sont complémentaires les une par rapport aux autres. Cependant, la technologie VLSI est en évolution continue et de nouvelles approches et techniques de conception sont appliquées à l'implémentation hardware des réseaux de neurones. Ceci nous conduit à proposer une nouvelle approche de classification basée sur de nouveaux critères dans la prochaine section.

### II.3.3 Proposition d'une approche pour la classification du hardware neuronal [114]

#### *Présentation*

A la suite de l'examen des différentes approches de classification du hardware neuronal, nous proposons une nouvelle approche qui prend en charge d'une part les éléments consensuels et d'autre part elle prend en considération l'évolution de la technologie VLSI, et des approches et techniques de conception. La figure II.15 montre notre approche pour la classification du hardware neuronale. Cette classification est faite selon l'évolution de la technologie de conception des circuits intégrés.

**Critères de classification :** D'abord les circuits sont classés se les critères

- Standard Chip
- VLSI Chips

Dans la catégorie Standards Chips, sont classées les réalisations basées sur les

- Cartes accélératrices
- Machines parallèles.

Dans la classe VLSI Chips, sont classées les réalisations basées sur les

- ASIC,
- FPGA
- Systèmes on Chip « SOC ».



---

Dans la classe des ASICs sont classées les réalisations basées sur

- Implémentation digitale,
- Implémentation analogique
- Implémentation hybride.

Dans la classe Implémentation digitale, les architectures suivantes sont classées :

- Slice architecture
- Multiprocessor architecture
- Systolic architecture
- SIMD architecture
- MIMD architecture
- RBF architecture

Dans la classe Implémentation analogique, nous avons identifiés les classes suivantes :

- Circuits dédiés pour implémentation une fonction biologique « Biological function » (rétine synaptic cochlea)
- Circuits dédiés à résoudre le problème liés l'implémentation des poids synaptiques (Floating Gate CMOS : « FGC »)
- Circuits dédiés à résoudre le problème de l'implémentation de la fonction d'activation (Charge Coupled devices : « CCD »)

Dans la classe Implémentation hybride, nous avons identifiés les techniques d'implémentation suivantes :

- Multiplying Analog to Digital Converters « MDAC »
- Pulse Stream
- Switched Capacitors «SC»
- Switched resistors «SR»

Dans la classe Implémentation FPGA, nous avons identifiés :

- Implémentations dans lesquelles le circuit FPGA est utilisé comme Coprocesseur.
- Implémentation exploitant la reconfiguration dynamique du circuit FPGA.
- Implémentation de SOFTCORE dans lequel le réseau de neurone est décrit en utilisant le langage VHDL ou VERILOG.

Dans la classe System on chip « SOC », nous avons identifiés des implémentations dans lesquelles :

- Le processeur est embarqué dans le circuit FPGA « embedded system on chip »
- Le processeur réside à l'extérieur du circuit FPGA « Hardware software Co-design »

Dans les prochaines sections, nous présenterons des exemples d'implémentation hardware des réseaux de neurones conformément à la classification que nous avons proposé.

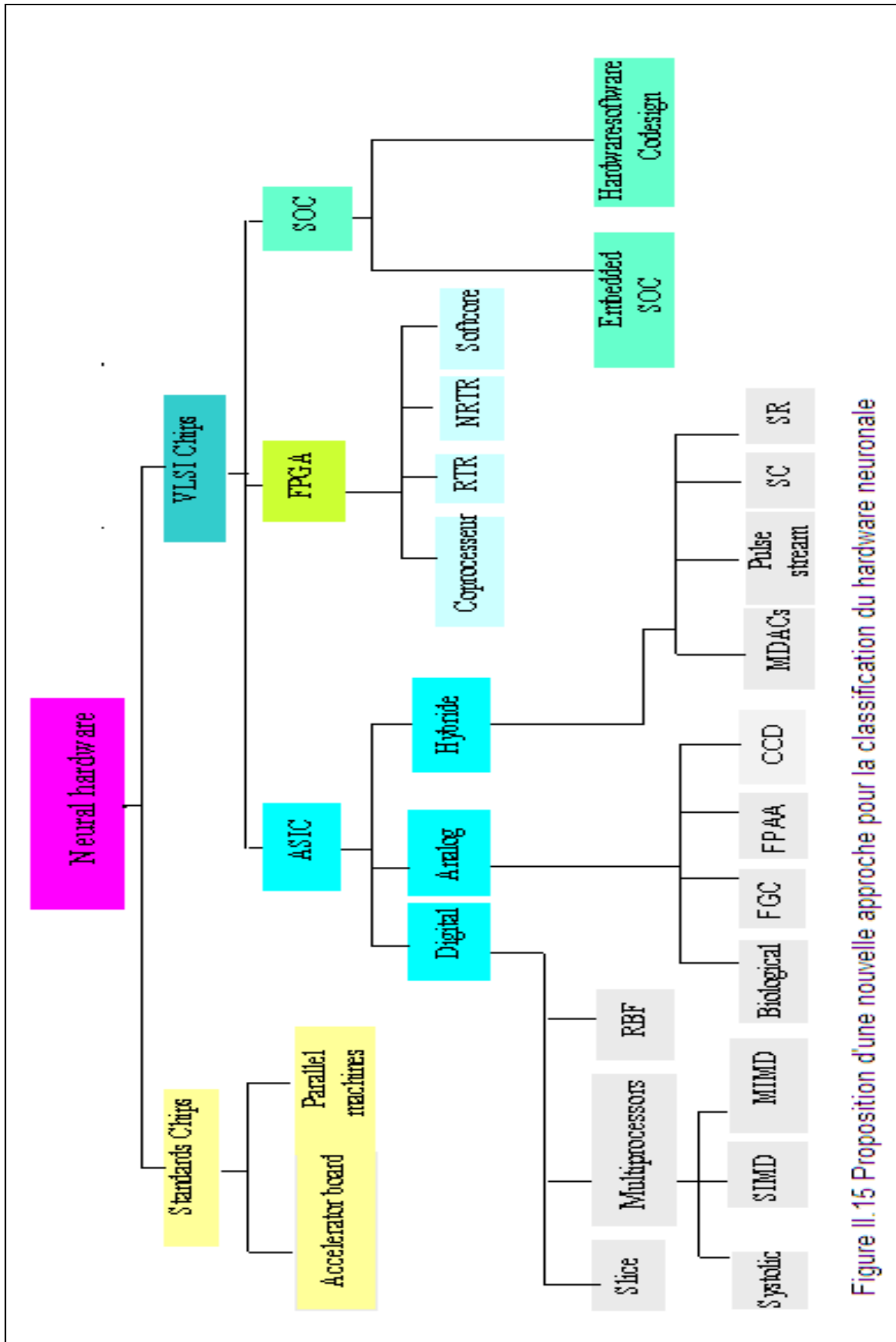


Figure II. 15 Proposition d'une nouvelle approche pour la classification du hardware neuronale

### II.3.4 Description des différents types d'implémentation du hardware neuronal

Dans cette section, une description des différents types d'implémentation est présentée, suivie d'exemples de circuits réalisés.

#### II.3.4.1 Implémentation du hardware neuronal sur des standard chips

Les implémentations à base de standard chip peuvent supporter un large spectre de modèles de réseaux de neurones. La figure II.16 montre l'architecture générale. La structure est constituée d'une matrice de processeurs parallèles identiques et connectés entre eux à travers un bus de transmission parallèle. Chaque unité physique (PU) exécute une section du réseau virtuel. Pour programmer le neurocalculateur, les PEs sont partitionnés à travers les mémoires locales des processeurs physiques. La mise à jour des PEs signifie la diffusion de la mise à jour à travers le bus. Les unités qui nécessitent un accès à cette information stockent la mise à jour dans les mémoires locales « local system state copy ». On peut diviser les neurocalculateurs à application générale en deux catégories : ceux à base de cartes accélératrices et les implémentations parallèles basées sur les multiprocesseurs.

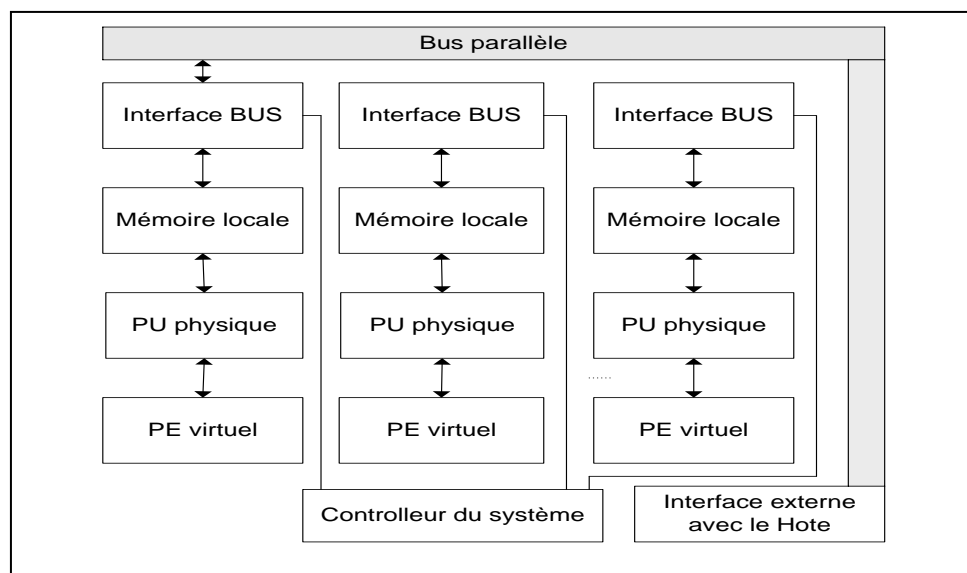


Figure II.16 Implémentation du hardware neuronal à base de standard chips

##### II.3.4.1.1 les cartes accélératrices

Les cartes accélératrices sont les plus utilisées dans la commercialisation des neurocalculateurs car elles ne sont pas coûteuses, largement disponibles et faciles à connecter à un PC ou une station de travail. Souvent, un outil software convivial accompagne ces dernières. La rapidité de traitement qui peut être obtenue par l'utilisation des cartes accélératrices est de l'ordre de deux, comparée aux implémentations séquentielles. Un inconvénient de ce genre de neurocalculateurs est leur inflexibilité car elles n'offrent pas beaucoup de possibilités pour le rajout de nouveaux paradigmes ou algorithmes. Des exemples de neurocalculateurs commercialisés sont présentés ci-dessous :

##### *La carte ANZA Plus (HNC [Hecht-Neilson Cooperation] CA, USA) :*

Ce genre de cartes contient le processeur NC68020 et le coprocesseur MC68881. Les performances sont 1.5 CUPS durant l'apprentissage et 6 MCPS durant la généralisation. Un

outil software muni d'une bibliothèque dédiée aux réseaux de neurones accompagne la carte **ANZA plus** [61].

***La carte NT6000 (Neural Technologies Limited, HNC):***

La carte **NT6000** [62] est équipée d'un circuit DSP TMS320 et d'un processeur neuronal NISP (Neural Instruction Set Processeur) qui atteint 2 MCPS. Ce genre de cartes est équipé d'un package software permettant l'interfaçage avec d'autres simulateurs neuronaux tels que le simulateur « BrainMaker » et « NeuralWorks ». Ce qui rendait un PC de type 386 ou 486 de cette époque, un excellent outil de simulation des réseaux de neurones. Néanmoins les paradigmes d'implémentation sont limités à l'algorithme de la rétropropagation du gradient et aux cartes de Kohonen, et une faible flexibilité est laissée pour le développement de nouveaux paradigmes.

***La carte Ni1000 (Intel Corp., CA, USA):***

C'est une carte accélératrice spécialement développée pour les applications de la reconnaissance optique des caractères « *Optical Character Recognition* », (OCR). Le software qui accompagne la carte est appelé NestorACCESS et utilise principalement les fonctions à base radiale (RBF) et l'algorithme de la rétropropagation du gradient. Les performances citées sont une rapidité entre 100-1000 fois comparées à une implémentation sur un PC seul ou bien un circuit DSP [56].

***La carte Neuro Turbo-I et Neuro Turbo-II (Nagoya Institute of Technology, Mitec Corp., Japan) :***

La première version du *Neuro Turbo-I* consistait en une carte intégrant 4 circuits DSPs permettant ainsi d'améliorer la vitesse de traitement par un facteur de 2. Par la suite cette architecture a été remplacée par 16 circuits DSP de Motorola (MB68220) Fujitsu, arrangés en hyper cubes. Le successeur de *Neuro Turbo-I* et le circuit commercial *Neuro turbo-II* qui contient le circuit DSP96002 de Motorola. Les performances obtenues sont 16.5 MCUPS [63].

### **II.3.4.1.2 Implémentations sur les machines parallèles**

Les machines parallèles sont des implémentations basées sur l'utilisation de plusieurs matrices de processeurs.. Les implémentations peuvent varier des architectures simples à faible prix, aux architectures avec des processeurs sophistiqués tel les transputers ou bien les circuits DSPs. Comparés aux cartes accélératrices, les implémentations sur les machines parallèles à base de multiprocesseurs offrent une meilleure flexibilité pour la programmation des fonctions neuronales Cependant, la programmation de telles machines et l'obtention de hautes performances nécessite la connaissance approfondie de l'architecture de la machine et le réglage détaillé du logiciel. Les réalisations les plus connues sont les suivantes :

***La Connection Machine CM-1/2/5 (Thinking Machine Corporation TMC Cambridge Massachusetts) :***

La *Connection -Machine* [28] est l'une des machines parallèles les plus populaires pour le calcul neuronal. La première version, CM-1 fût développée durant la période 1981-1986. La seconde version est apparue en 1987. La CM-1 et CM-2, prennent forme d'un cube de 1,5 m de hauteur divisé également en 8 sous cubes. Chaque sous cube est composé de 16 cartes et un processeur principal, appelé séquenceur. Chaque carte est composée de 32 chips. Chip est composé d'un cana de communication appelé routeur, de 16 processeurs et 16 RAMs. Les machines CM-1 et CM-2 sont basées sur architectures de type SIMD. Comparée à la CM-1, la CM-2 dispose du coprocesseur WEITEK à virgule flottante et plus de mémoires

RAMs. La programmation software pour les deux machines est basée sur le langage de programmation LISP. En 1992, la *Thinking Machine* est passé de la CM-2 à la CM-5 avec une architecture de type MIMD basée sur processeur SPARC- RISC. La figure II.17 montre l'architecture de la machine.

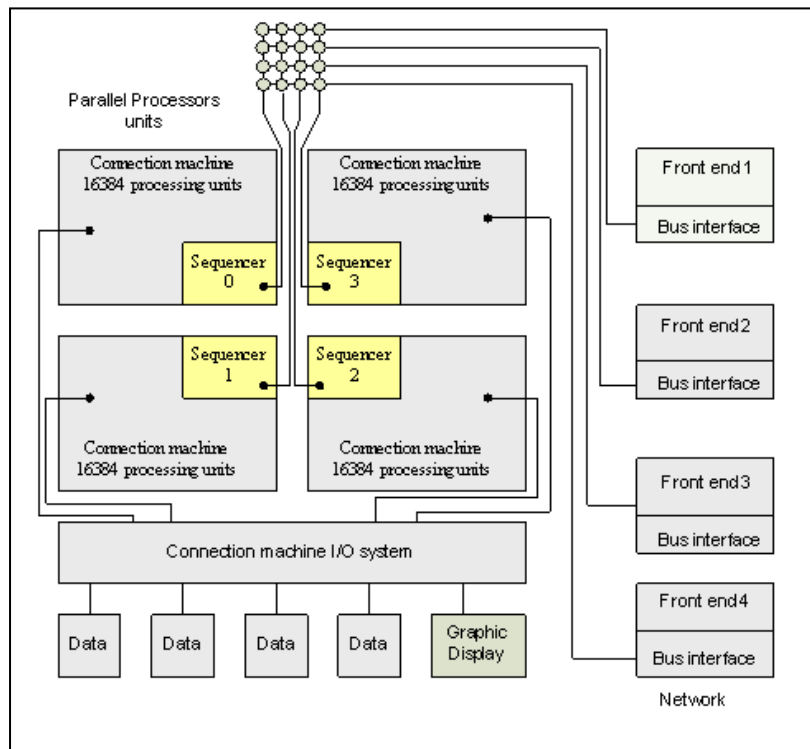


Figure II. 17 Architecture de la

### **Le système NETSIM :**

Le système NETSIM [115] consiste en un ensemble de cartes accélératrices arrangées en une structure 3D et contrôlées par un PC hôte comme le montre la figure II.18. Chaque carte accélératrice est formée :

- D'un microprocesseur standard de type 80188 et son programme mémoire associé
- Deux circuits dédiés « le circuit de simulation » et le circuit de communication
- Une mémoire pour stocker les poids synaptiques, le nombre d neurones à l'entrée ainsi que le nombre de réseaux de neurones par carte *NETSIM*.
- Un système BIOS dans une mémoire EPROM.

Le circuit de simulation est utilisé comme coprocesseur pour réaliser les différents calculs relatifs aux modèles neuronaux. Une extension au microprocesseur a été faite pour réaliser les fonctions suivantes :

- Dummy cycle qui réalise le management des fonctions
- Repeat-Multiply and Sum qui resolve la multiplication du vecteur d'entrée avec les poids synaptiques
- READ-WRITE qui déplace les données de la mémoire et permet l'accès aux synapses à partir du microprocesseur
- Repeat-Multiply-Sum an Write qui permet la mise à jour des synapses durant la phase d'apprentissage.

Le circuit de communication permet de connecter les différentes cartes entre eux. Ce circuit est formé d'un Bus de 64 bits dont les premiers 16 bits représentent l'adresse de

destination. Et 48 bits pour la donnée. Le schéma d'adressage permet +-15 cartes *NETSIM* de se connecter dans les trois dimensions.

Le PC Hôte agit comme le contrôleur du neurocalculateur et permet de programmer les conditions initiales et les caractéristiques du réseau de neurones.

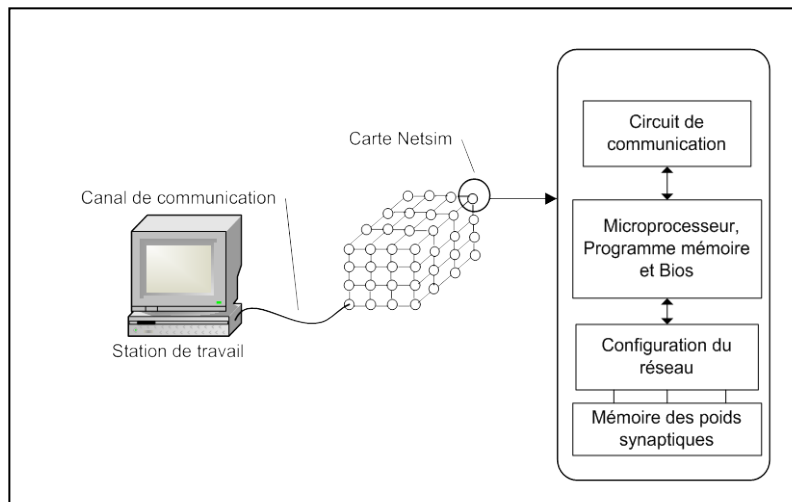


Figure II.18 Architecture du système *NETSIM*

### Le système *IBM GF11* :

*IBM GF11* [38] est un ordinateur expérimental SIMD avec 566 processeurs réalisant une exécution maximale de 11 Gigaflops. Chaque processeur est capable d'exécuter 20 millions d'action de flottant ou des opérations de point fixées par seconde. Les processeurs contiennent les mots de 16 Koctets de RAM statique et les mots de 512 Koctets de mémoire dynamique. Les processeurs sont connectés via un réseau reconfigurable de type BENES. La Figure II.19 montre l'architecture du système *IBM GF11*.

Le *GF11* a été évalué sur l'outil *NETtalk* [116], jeu de formation, en utilisant le réseau 203-60-26. Avec 356 processeurs opérationnels, le *GF11* réalise 901 MCUPS. Les auteurs estiment qu'avec 566 processeurs opérationnels, 1231 MCUPS seraient réalisés.

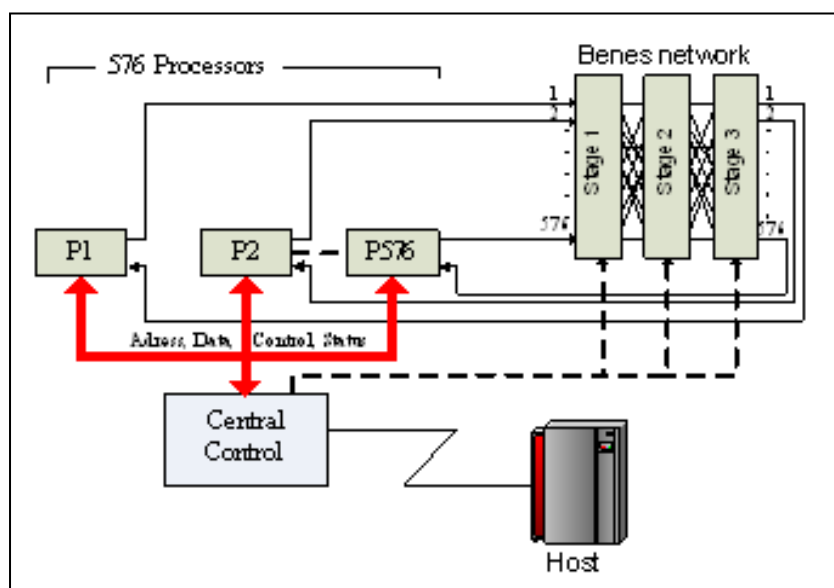


Figure II.19 Architecture du système *IBM-GF11*

### La machine iPSC/860

L'iPSC/860 [46] est une machine parallèle de type MIMD à mémoire distribuée. Elle est constituée de nœuds qui sont interconnectés suivant une topologie en hyper cube. L'ensemble des nœuds est relié au SRM « Service Resource Manager » qui s'occupe des liaisons de l'iPSC avec le réseau externe et gère également l'accès au disque dur (Figure II.20).

Chaque nœud est constitué d'un processeur i860 d'Intel de 16 Méga octet de mémoire et d'un module de routage « Direct Connect Module : DCM » à 8 voies permettant d'envoyer et de recevoir des messages : 7 vers les autres nœuds et un réservé au système pour les entrées sorties (I/O). Les sept voies permettent d'avoir un degré de connectivité maximale de 7 (au total 128 nœuds ou processeurs). Les programmes peuvent être écrits dans un sous ensemble du langage C. Ce dernier supporte la communication inter-processeur et d'autres fonctions nécessaires.

Il est à noter que l'iPSC/860 est une machine parallèle très puissante en calcul. Cependant, elle présente une disproportion entre sa vitesse de calcul et son temps de communication. A titre d'exemple et pour des buts d'évaluation de performances, les données employées dans le projet de *NETtalk* ont été implantées sur l'iPSC/860, employant des différents nombres de processeurs. L'iPSC/860 à 32 processeurs a réalisé 12 MCUPS, alors que l'iPSC/860 à 1 processeur a réalisé 1 MCUPS !

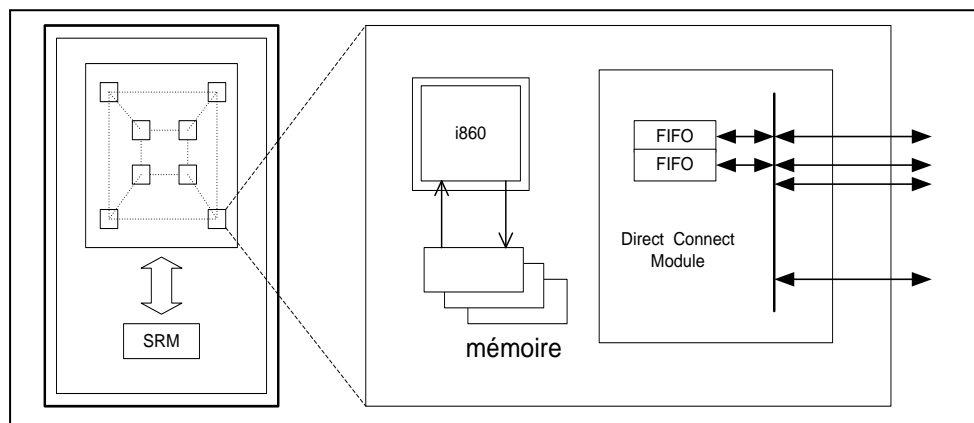


Figure II.20 - La machine iPSC/860 et description d'un nœud

#### II.3.4.2 Implémentation du hardware neuronal sur les chips VLSI

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 [117]:

- 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.

Trois grandes approches de conception peuvent être considérées lors d'une implémentation VLSI. Il s'agit de l'approche ASIC qui désigne une implémentation d'un circuit pour une application spécifique qu'il soit de type analogique, digital ou mixte; l'approche FPGA qui désigne la réalisation dans un circuit programmable et l'approche SOC qui consiste à intégrer tout un système dans une seule puce.

Le choix d'un style par rapport à un autre dépend de plusieurs facteurs tels que la vitesse, la précision, la flexibilité, la programmation, le degré de parallélisme, le transfert et la mémorisation des données, la consommation de puissance, la miniaturisation, la sécurité, etc.

### II.3.4.2.1 Implémentation sur circuit ASIC analogique

Les circuits analogiques sont simples et très rapides (10 à 100 fois plus rapides que les circuits digitaux), cependant leur implémentation sur du silicium est fastidieuse et nécessite une bonne connaissance des lois physiques et modèles mathématiques des transistors, des circuits de base, ainsi qu'une bonne expérience du dessin de masque « layout » du circuit. Les circuits analogiques possèdent des caractéristiques intéressantes qui peuvent être directement utilisées pour implémenter des réseaux de neurones. Ainsi, l'amplificateur opérationnel de la figure II.21, peut être utilisé pour implémenter la fonction de transfert d'un neurone et les poids synaptiques peuvent être représentés par des résistances ou leurs circuits équivalents [118].

Du point de vue caractéristique, l'inconvénient des circuits analogiques est leur sensibilité au bruit, à la température, à la tension d'alimentation ; ils sont moins précis et consomment une grande surface.

Du point de vue techniques de conception, les résistances qui sont utilisées pour implémenter les poids synaptiques ne sont pas adaptables (programmables), ce qui limite l'implémentation du réseau de neurone à seulement l'approche « off chip training ». D'autres techniques sont utilisées pour surmonter ce problème, tel que la technique CCD (Charge Coupled Devices) [119], néanmoins cette dernière est sensible à la variation des paramètres du procédé technologique. Des exemples de neurochips analogiques sont décrits ci-dessous :

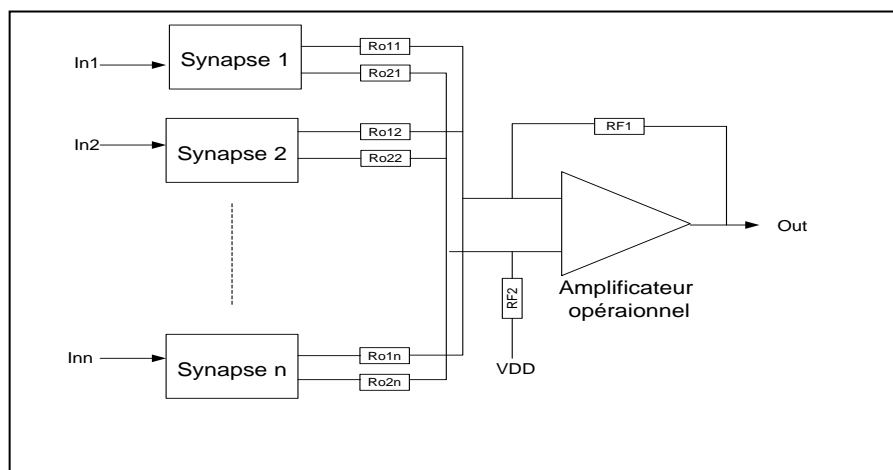


Figure II.21 Architecture d'un neurone analogique

#### *Le circuit ETANN (Intel Inc, CA, USA)*

L'ETANN, « Electrically Trainable Analog Neural Network » (ETANN-801770NX) est le premier neurochip analogique développé en 1989 par Holler et al. [71]. Le circuit est basé sur l'implémentation « off chip training » d'un réseau de neurone à une ou deux couches.



Le circuit accepte jusqu'à 64 entrées (neurones) et 1024 poids synaptiques. Il peut être configuré pour 64 neurones dans la couche cachée et 64 neurones à la sortie. Chaque neurone est connecté à 16 tensions de polarisation internes fixes.

Le circuit représentant les poids synaptiques est basé sur le multiplieur de Gilbert [120]. Les sorties issues de chaque synapse sont sommées et présentés à un amplificateur à seuil qui représente le neurone. Le circuit performe 2GCPs. La figure II.22, montre la carte ETANN- 80177NX.

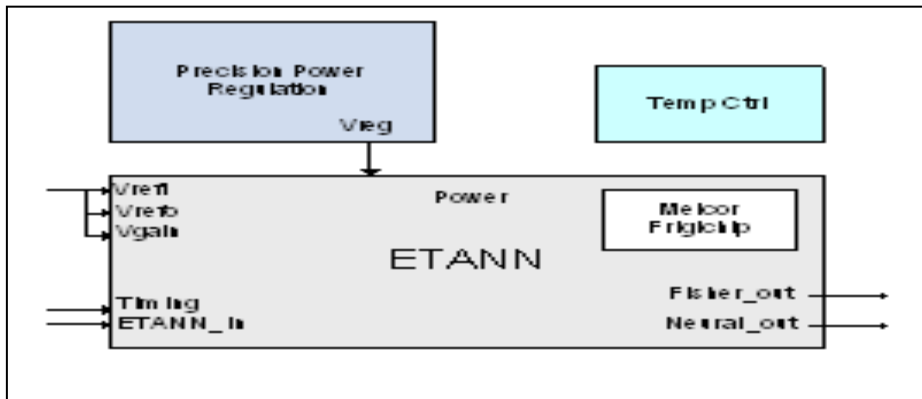


Figure II.22 Architecture de la carte ETANN-80177NX

### La rétine synaptique

Maeda et al. (1989) [121] ont implémenté un circuit neuromorphique analogique basé sur le comportement des neurones liés à la rétine propre au système visuel. La Figure II.23 (a) montre une section de la rétine visuelle. Cette dernière est composée d'un ensemble de neurones récepteurs. La lumière à l'entrée de chaque nœud est adaptée en accord avec la valeur moyenne de la lumière de l'espace extérieur. La Figure II.23 (b) montre le modèle analogique équivalent où chaque noeud (neurone) est connecté seulement à ses voisins avec des connexions fixes. Dans ce genre d'implémentation il n'y a pas d'apprentissage mais en général, les neurones adaptent leur comportement en accord avec les stimuli qu'ils reçoivent en entrée.

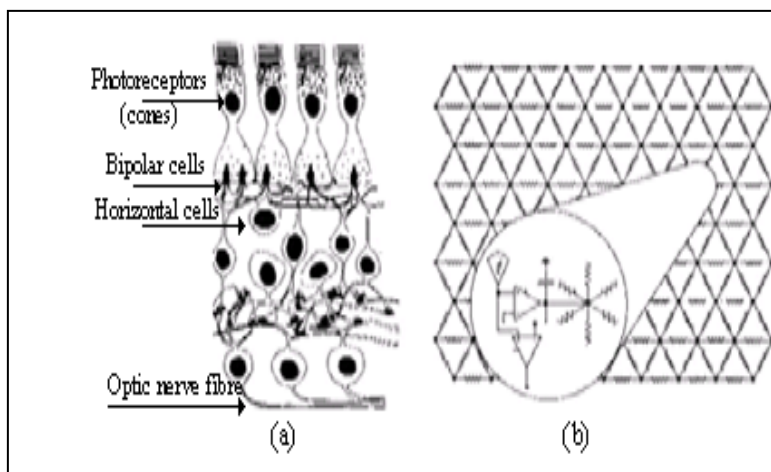


Figure II.23 Principe de la rétine synaptique. (a) Section de la rétine. (b) Model équivalent.

Des versions plus améliorées du modèle de la rétine synaptique sont proposées dans [122], [123], [124] et [125].

**La plateforme analogique FPAA : AN221E04(Anadigm Inc.)**

Dans la référence [126], P. Rock et ses co-auteurs ont utilisé une plateforme hardware analogique pour implémenter un réseau de neurone impulsif évolué, autrement dit, « Spiking Neural Network : SNN »

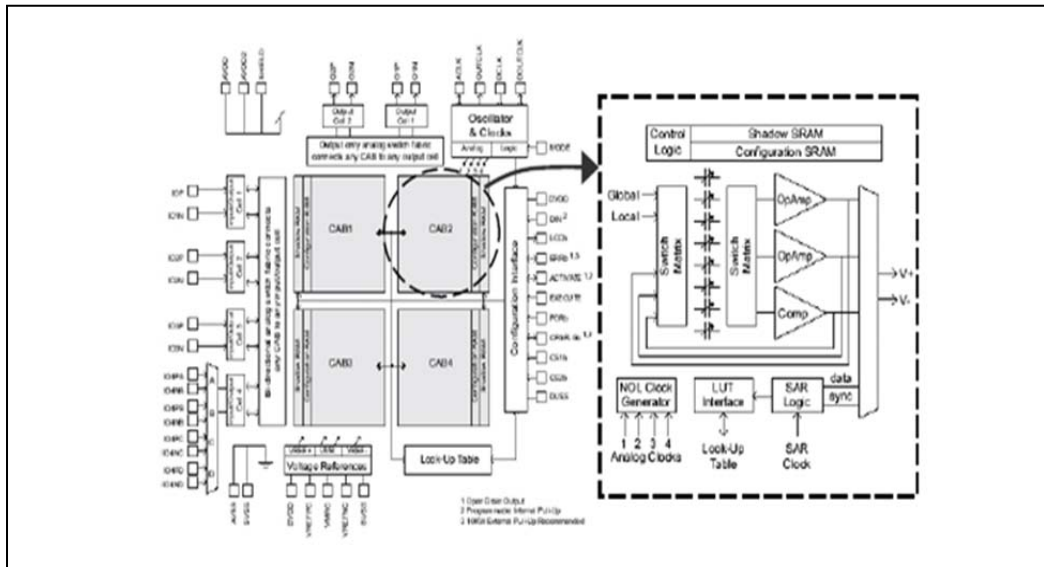


Figure II.24 (a) Architecture du circuit AN221E04 ANADIGM. (b) Architecture du bloc CAB

La plateforme en question est un réseau de matrices programmables «Field Programmable Analog Array : FPAA », basé sur le circuit AN221E04 de la compagnie ANDIGM [127]. La figure II.24 (a) montre l'architecture du circuit AN221E04. Ce dernier est basé sur le modèle des capacités commutées (switched capacitor model). Ainsi plusieurs configurations être réalisées grâce à la manipulation des switches à l'intérieur des blocs configurables analogiques (CAB). Chaque CAB contient deux amplificateurs opérationnels (op-amp), un comparateur et 8 capacités variables (Figure II.24 (b)). En comparant les performances du FPAA à une implémentation software du réseau SNN, les auteurs montrent que le FPAA dispose de performances meilleures quant aux changements de l'environnement.

**II.3.4.2.2 Implémentation sur circuit ASIC digital**

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. Des exemples sont cités ci-dessous :

**Le système CNAPS (Adaptive Solutions, Inc., USA)**

Le système CNAPS [37] (Connected Network of Adaptive Processors) de la figure II.25, présente une architecture de type SIMD ayant 64 unités de traitement (PN0, ..., PN63). Ces derniers sont implémentés dans le circuit intégré N64000 qui est comparable à un circuit DSP mais dont les multiplieurs sont spécialement conçus avec une précision réduite. Le système standard est constitué de quatre circuits N64000, intégrés sur une carte et contrôlés par un circuit séquenceur qui émet des commandes à tous les chips via un bus d'instruction de 31 bits. Chaque processeur PN possède une mémoire SRAM de 4 Kbytes pouvant stocker des poids synaptiques dont la taille de la donnée peut aller jusqu'à 16 bits. Tous les processeurs reçoivent la même instruction qu'ils exécutent en parallèle. Les circuits N64000 sont connectés en cascade et communiquent via un bus. CNAPS supporte l'implémentation « on chip

training » seulement. Pour le fonctionnement du système, chaque neurone est assigné à un élément de traitement PN. Pour des réseaux de neurones de grande taille, un élément PN peut implémenter plusieurs neurones de façon séquentielle. La figure II.25 montre deux caractéristiques importantes du système *CNAPS* à savoir :

- La régularité de la structure ce qui permet d'étendre le nombre de N64000 jusqu'à 8 circuits
- La connectivité réduite rendant l'accès aux mémoires simple, néanmoins si la taille du réseau à implémenter dépasse les capacités des mémoires, le système devient très lent.

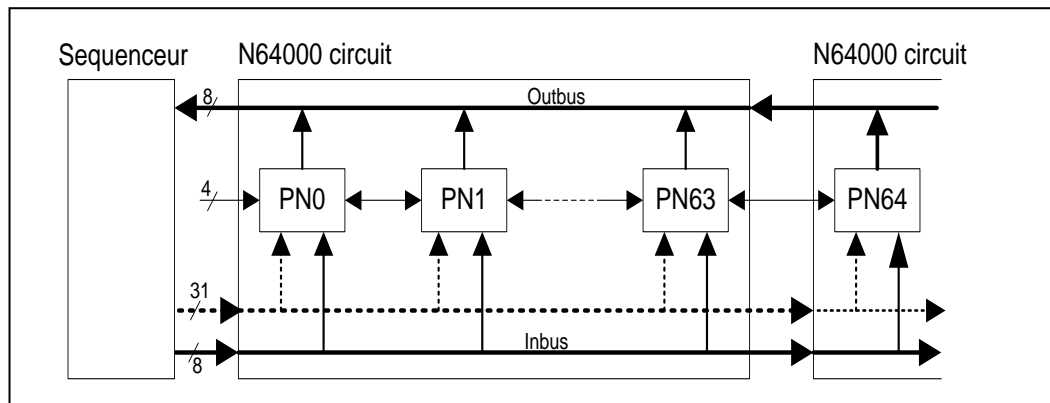


Figure II.25 Architecture du système CNAPS

Un seul circuit performe 1.6 GCPS et 256 MCPS pour des poids synaptiques de 8 à 16 bits et une performance de 12.8 GCPS pour des poids synaptiques de 1 bit.

Le système *CNAPS* complet est constitué d'un serveur qui permet de se connecter à la machine hôte, un Codenet et un ensemble d'outils software de développement. Il supporte l'algorithme de Kohonen, l'algorithme de la rétropropagation du gradient, et des fonctions de traitement du signal tel que la convolution et la quantification vectorielle. En 1994, le système *CNAPS* fut considéré comme étant le neurocalculateur le plus rapide du monde.

### ***Le circuit IBM ZISC036 (IBM Micro Electronics, France)***

Le circuit Zero Instruction Set Computer (ZISC036) [69] est le premier chip développé par la compagnie IBM pour l'implémentation des réseaux de neurones de type RBF (Radial Basis Function), fonctions à base radiales et utilisé dans le traitement d'image pour la classification en temps réel et la reconnaissance des formes. Pour cela le processeur ZISC utilise les fonctions RBF et la méthode du plus proche voisin « *K-nearest neighbour* ». La figure II.26 présente l'architecture générale du processeur ZISC. La première version du circuit possède 64 entrées de 8 bits chacune et 36 neurones qui implémentent 36 RBF fonctions. Plusieurs circuits peuvent être montés en cascades permettant ainsi l'implémentation des réseaux de grande taille. Pour une fréquence de 33 Mhz, ZISC peut réaliser une classification d'un vecteur de 64 éléments en moins de 3  $\mu$ sec.

Une version plus évoluée du processeur ZISC est le ZISC078 implémentant 78 neurones. Pour une fréquence de 50 Mhz, le circuit peut réaliser une classification de 1 millions de formes en 1 sec. Le circuit a été fabriqué en utilisant l'approche « standard Cell » de la technologie CMOS 0.25  $\mu$ m d'IBM. Un exemple d'utilisation du processeur IBM ZISC est le projet CogniSight/CogniMen : C'est une implémentation hardware de réseaux de neurones utilisé dans l'inspection des poissons : « fish inspection » [128].

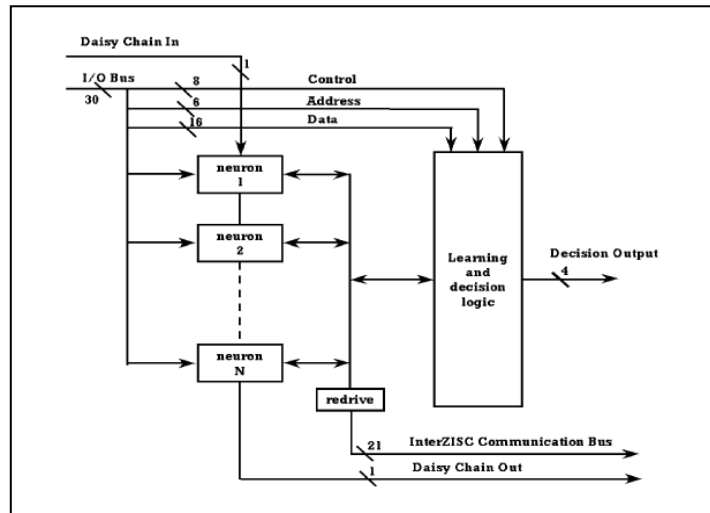


Figure II.26 Architecture du circuit neuronal IBM ZISC036

**Le circuit MANTRA-I (EPFL, Suisse)**

Les implémentations décrites précédemment traitent le parallélisme des neurones qui consiste à assigner un élément processeur par neurone. Un degré de parallélisme plus fin peut être obtenu en utilisant le parallélisme des synapses.

Un des rares neurocalculateurs qui traite le parallélisme des synapse, est le circuit MANTRA [54] développé par à l'Ecole polytechnique Fédérale de Lausanne (EPFL). La figure II.27 montre l'architecture de MANTRA. C'est une matrice à 2-D de 40x40 éléments processeurs systoliques appelés GENES-IV (Generic Element for Neuro- Emulator Systolic Array) [129]. Ces processeurs sont connectés en ligne série avec leurs quatre voisins, comme montré dans la figure II.27. Les opérations sont réalisées par les PEs situés sur Nord-Ouest jusqu'au Sud -Est diagonale.

L'architecture de MANTRA est scalaire et peut implémenter plusieurs modèles de réseaux de neurones tel que le réseau de Hopfield, le perceptron multicouche et le model de Kohonen. Les performances citées sont : une fréquence maximale 10 Mhz, 400 MCPS et 133 MCUPS (pour l'algorithme de la rétropropagation du gradient).

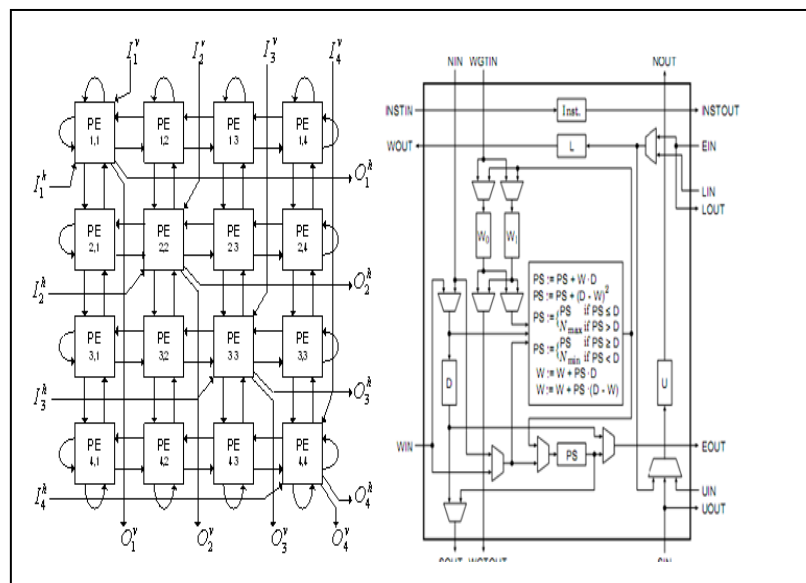


Figure II.27 Architecture du circuit MANTRA-I (EPFL) et du bloc PE [125]

### II.3.4.2.3 Implémentation sur circuit ASIC hybride

Les techniques de conception hybride ont pour but de combiner les avantages des circuits digitaux et analogiques. Généralement, les entrées/sorties du réseau sont digitaux alors qu'à l'intérieur du réseau, les circuits sont analogiques. Néanmoins la condition initiale pour marier ces deux techniques est que le procédé technologique utilisé pour le circuit analogique doit être compatible avec le procédé technologique digital en question. Il existe plusieurs techniques de conception hybrides comme suit:

#### *Multiplying Digital-to-Analog Converters « MDACs »*

Dans cette technique les poids synaptiques sont codés en digital et stockés dans des cellules RAM. Par la suite, les poids sont transférés vers le circuit MDAC qui effectue la multiplication des poids synaptiques codés en digital avec le signal d'entrée qui est analogique. Cette approche, permet de soulever le problème des poids synaptiques, néanmoins elle est gourmande en surface, et son prix de revient est élevé. A titre d'exemple, dans [130], les auteurs ont proposé le bloc digramme de la figure II.28 pour implémenter le neurone synaptique. Les principaux composants du circuit sont le circuit « Sign-Bit » le convertisseur tension-courant « V-I », et le miroir de courant « Binary Weighted Current Mirrors ». Ainsi, le circuit MDAC reçoit une entrée de 5-bit de la mémoire (D0-D4). Les entrées D0-D3 représentent l'entrée du circuit miroir de courant et D4, l'entrée du circuit « Sign-Bit ». L'entrée analogique  $V_{in}$  est assignée au convertisseur « V-I » pour obtenir un courant de sortie  $I_{out}$  proportionnel au produit poids synaptique avec le signal d'entrée.

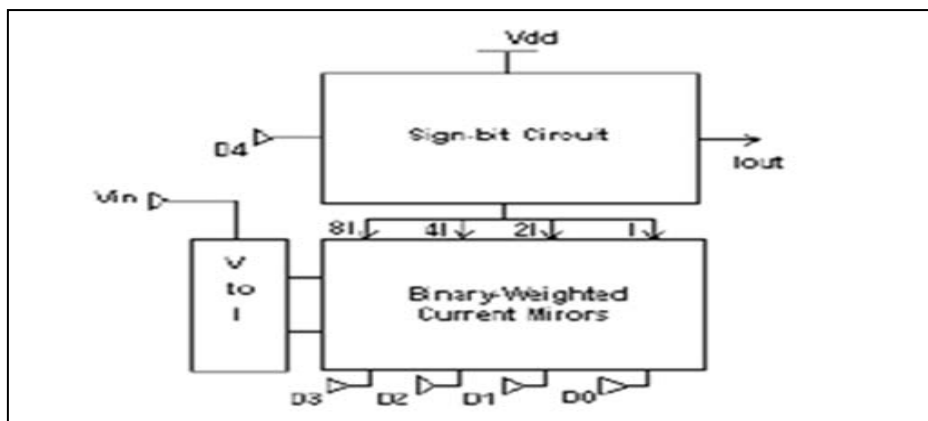


Figure II.28 Bloc digramme du circuit MDAC [130]

L'architecture interne du circuit MDAC est montrée dans la figure II.29. Dans cette figure, les transistors M1, M2 et M3 sont utilisés par le convertisseur « V-I » ; les transistors M4-M15 par le circuit miroir de courant « Binary weighted Current Mirrors » et les transistors M16-M21 sont utilisés par le circuit « Sign-Bit ». Le circuit MDAC a été implémenté selon le procédé de la technologie 0.1  $\mu\text{m}$ .

#### *La technique « Pulse stream encoding »*

Dans cette technique, un train d'impulsion est utilisé par le circuit neuronal. Cette technique utilise le principe du neurone biologique. Quand un neurone est excité : état « ON », alors un train d'impulsion est acheminé vers la sortie ; et quand il est inhibé, aucune impulsion n'est transmise : état « OFF ». L'approche d'implémentation hybride est utilisée car il s'agit de faire la multiplication analogique sous le control d'un circuit digital. La première implémentation neuronale utilisant cette technique fût proposée dans [131].

L'avantage de cette technique est l'immunité au bruit, l'insensibilité à l'atténuation du signal faible consommation et puissance. La multiplication analogique est attractive dans les implémentations VLSI pour des raisons de vitesse et de surface et de puissance comparée à l'implémentation digitale. L'inconvénient est l'interférence électromagnétique, et l'interférence du signal ainsi que d'autres problèmes liées au procédé technologique mixte.

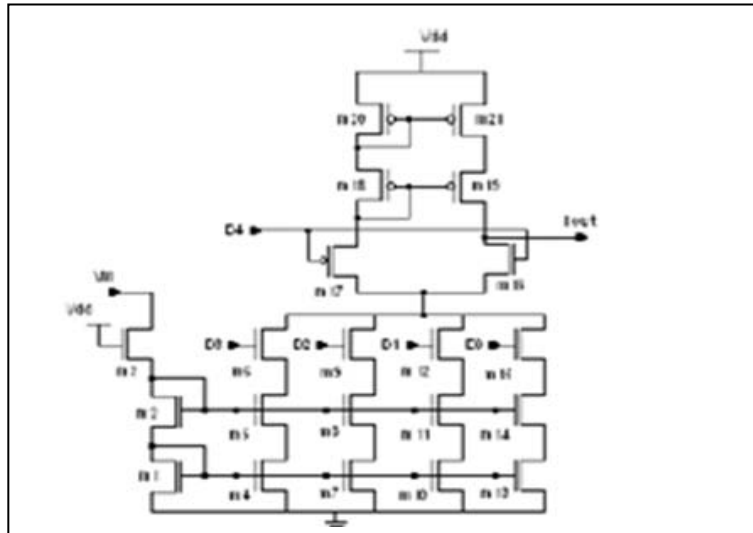


Figure II.29 Architecture interne du circuit MDAC [130]

#### **La technique « Switched Capacitors : SC »**

Les Implémentations ANNs basées sur cette technique sont adéquates pour résoudre des problèmes d'optimisation linéaire ainsi que pour la reconnaissance de la parole. Dans [131], un circuit neuronal de type « SC » a été proposé pour résoudre un problème d'optimisation. Dans la référence [132], un circuit neuronal pour le problème de Cochlea a été proposé.

#### **La technique « Switched Resistor : SR »**

Les techniques « SC » et « MDAC » étant gourmandes en surface, certains auteurs suggèrent l'utilisation d'un transistor MOS pour la réalisation du switch. Ce dernier peut être réalisé en appliquant un signal d'horloge  $\phi$  à la grille du transistor MOS. La figure II.30 (a et b), montre le transistor MOS et son modèle de résistance équivalent, où

$$R_{on} = \frac{V_s - V_d}{I_{on}}$$

Où  $V_s$  et  $V_d$ , représentent les tensions source et drain du transistor MOS respectivement.  $I_{on}$ , représente le courant du transistor.

La Figure II.30 (c) montre le signal d'horloge  $\phi$ .

Dans [133], l'auteur montre que la résistance du transistor contrôlée par l'horloge s'écrit sous la forme :

$$R = R_{on} \frac{T}{\tau}$$

Où  $T$ , représente la période du signal d'horloge et  $\tau$  la durée de l'impulsion.

Ainsi, grâce à cette méthode on peut réaliser une résistance programmable représentant les poids synaptiques, ce qui soulève le problème de programmation ou modification des poids qui est posé dans la conception analogique.

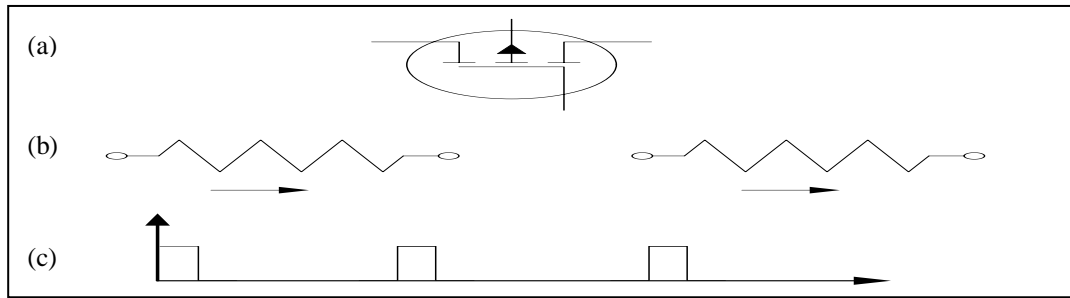


Figure II.30 (a) Transistor MOS (b) Résistance équivalente (c) L'horloge de contrôle

L'horloge peut être réalisée par les circuits « Programmable Logic Arrays » (PLA) [134].

L'inconvénient de l'approche est le bruit dû aux capacités parasites entre le substrat et la source d'une part, et le substrat et le drain d'autre part.

### II.3.4.3 Implémentation sur circuit FPGA

L'utilisation des circuits FPGAs (Field Programmable Gate Arrays) dans l'implémentation des réseaux de neurones remonte au début des années 1990s. Souvent, le thème « *Reconfigurable Computing for ANNs* » est cité dans la littérature spécialisée [135]. Ce thème de recherche regroupe toutes les techniques qui permettent d'accélérer les performances d'une application ou d'un algorithme neuronal donné, au-dessus de ce qui est obtenu par les standards chip ou bien les circuits d'application générale.

De manière générale, les circuits FPGAs permettent de réaliser un meilleur compromis entre la flexibilité des standards chips et les performances des circuits ASICs.

Les premiers circuits FPGAs, tel que les circuits de la famille XC3000 et XC4000 étaient très limités en terme de performances et de densité d'intégration, ce qui empêchait l'implémentation hardware des réseaux de neurones de grande taille.

Grâce aux avancées dans le développement de la technologie microélectronique d'une part, et celui des outils de conception (*front end* et *back end*), d'autre part, les circuits programmables FPGAs ont progressé de manière évolutionnaire et révolutionnaire.

Le processus d'évolution a permis la réalisation de circuits FPGAs très rapides, présentant une grande densité d'intégration et un meilleur support technique. La révolution concerne l'intégration des multiplieurs de haute performance, des microprocesseurs, des circuits DSPs et des circuits de communication. Cette progression a une incidence directe sur l'implantation des réseaux de neurones sur FPGA. De ce fait, plusieurs travaux ont été entamés dans ce sens [136].

Les principaux thèmes de recherches concernent l'exploitation de la configuration et de la reconfiguration des circuits FPGAs, l'utilisation des circuits FPGA en tant qu'accélérateurs du hardware (Coprocesseurs) et le développement de composants *IPs*, « Intellectual Property » en tant que « *SoftCore* ».

Dans ce qui suit nous présentons un état de l'art sur chaque thème.

#### II.3.4.3.1 Utilisation de la configuration et la reconfiguration des circuits FPGAs

Toutes les implémentations des réseaux de neurones ont pour but d'exploiter soit la configuration soit la reconfiguration des circuits FPGA. L'identification du but de la configuration permet de comprendre la motivation derrière chaque type d'implémentation.

Dans la littérature on peut distinguer quatre modes de configuration et programmation des circuits FPGAs comme le montre la Figure II.31 [137] :

- La configuration est le processus de charger des données spécifiques à un design dans un ou plusieurs FPGAs pour définir l'opération fonctionnelle des blocs internes ainsi que leur interconnexions.
- La reconfiguration est le processus de configurer le circuit une deuxième fois; soit pour une deuxième utilisation de la même fonction soit pour l'implémentation d'une autre fonction.
- Un dispositif ou un circuit est défini comme étant reconfigurable globalement, s'il est possible de le reconfigurer sélectivement, tandis que l'état de repos du reste du dispositif est inactif, mais il conserve son information configurée. Dans la littérature, la terminologie de « Global run time reconfiguration » ou « RTR globale » est utilisée
- Un circuit FPGA est reconfigurable dynamiquement, s'il peut être partiellement reconfiguré durant son fonctionnement, c.-à-d. une partie du circuit correspondant à certaines fonctions logiques et leur interconnexions peut être changée sans affecter le fonctionnement de la logique restante (dans la littérature les terminologies : « *Partial run time reconfiguration* » ou bien « *local Run Time Reconfiguration* », *local RTR*, sont utilisées. Aussi, on peut parler de reconfiguration dynamique dans le cas où plusieurs circuits FPGAs sont connectés entre eux et il s'agit de reconfigurer un seul composant FPGA tout en maintenant les autres circuits en fonctionnement.

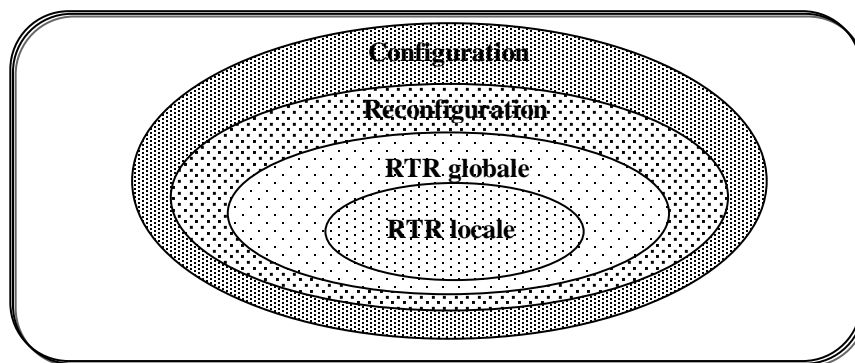


Figure II.31 Classification des circuits FPGAs selon leurs configurations

Les travaux qui ont exploités la configuration et la reconfiguration des circuits FPGAs pour l'implémentation des réseaux de neurones peuvent être classés comme suit :

#### ***Prototypage et simulation des réseaux de neurones : Approche Non RTR (NRTR)***

Dans cette classe, la configuration et reconfiguration du circuit FPGA sont exploitées de manière illimitée pour différentes stratégies d'implémentation des réseaux de neurones et de façon à réaliser des simulations et des preuves de concept en un temps très court. Le projet GANGLION est un bon exemple de prototypage rapide qui a connu un grand succès [49].

GANGLION implémente un réseau multi-couches, constitué de (12-14-4) neurones, et est utilisé en traitement d'images. Le réseau en question est implémenté sur deux circuits FPGAs de la famille XC3090.

#### ***Amélioration de la densité d'intégration (Approche RTR)***

Dans cette classe, sont présentées les méthodes qui augmentent la fonctionnalité par unité de surface à travers la reconfiguration partielle du circuit FPGA. Ainsi, il est possible de



réaliser un multiplexage temporel d'un circuit FPGA pour l'implémentation séquentielle de chaque phase d'un algorithme neuronal.

Dans [94], Eldredge et ses co-auteurs, ont proposé une approche nommée *RRANN* « *Run Time reconfiguration Artificial Neural Network* » qui consiste à diviser l'algorithme de la rétropropagation du gradient en trois modules : le module propagation, le module de rétropropagation et le module de mise à jour des poids synaptiques, comme le montre la Figure II.32. D'abord, le module de propagation est implémenté sur un circuit FPGA de la famille XC3090. Ensuite, le circuit est reconfiguré pour implémenter le deuxième module et ainsi de suite. De ce fait, l'implémentation de chaque module est exécutée dans les mêmes ressources du circuit FPGA en question.

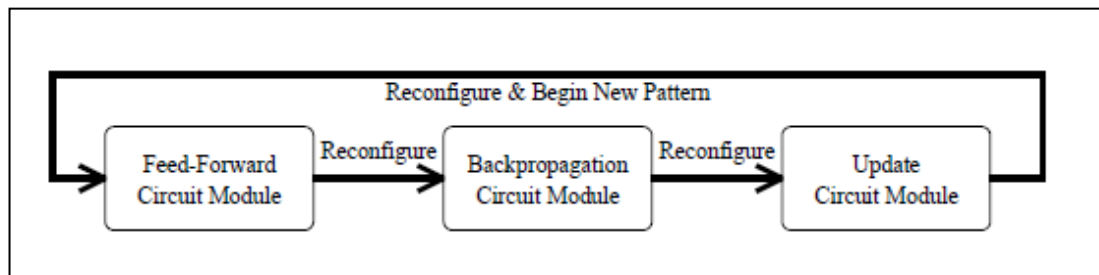


Figure II.32 Utilisation de la reconfiguration partielle pour l'implémentation de l'algorithme RPG

Les auteurs rapportent que l'utilisation de la RTR permet un gain de 500% de la densité d'intégration. (Algorithme RPG complet : 1 neurone/FPGA, si on utilise la RTR : 6 neurones/FPGA).

Aussi, Il est possible de réaliser un multiplexage temporel du circuit FPGA pendant l'exécution. Cette technique est connue sous le nom de « *Dynamic Constant Folding* ».

James Roxby [138] a implémenté un réseau de neurones de dimension 4-8-8-4 sur un circuit FPGA de la famille Virtex-XCV600BG560, en utilisant des multiplieurs à coefficient constant représentant les poids synaptiques. James Roxby montre que les poids synaptiques peuvent être modifiés (reprogrammés) toutes les 69  $\mu$  Sec. Ce type d'implémentation peut être utilisé lorsque le parallélisme des exemples d'apprentissage « *training level parallelism* » est considéré avec la mise à jour en « batch mode » des poids synaptiques à chaque itération. La même idée a été adoptée par Zhu et al. [139].

Upegui et ses co-auteurs [140] ont présenté une méthodologie permettant de définir la topologie d'un réseau ANN et qui est basée sur un algorithme génétique. Le circuit final est implémenté en exploitant les propriétés de la reconfiguration partielle dans les circuits FPGA de type Virtex-II. La figure II. 33 (a) montre le principe de la reconfiguration partielle dans la Virtex-II. Et la figure II.33 (b) montre la topologie des blocs reconfigurables.

Dans la figure II.33, chaque module reconfigurable représente une couche du réseau ANN en question. Chaque module communique avec ses voisins à travers un bus macro implémenté spécialement dans les circuits FPGA. Pour chaque module, il existe une panoplie de configurations. Un algorithme génétique détermine la configuration (topologie) qui doit être choisie.

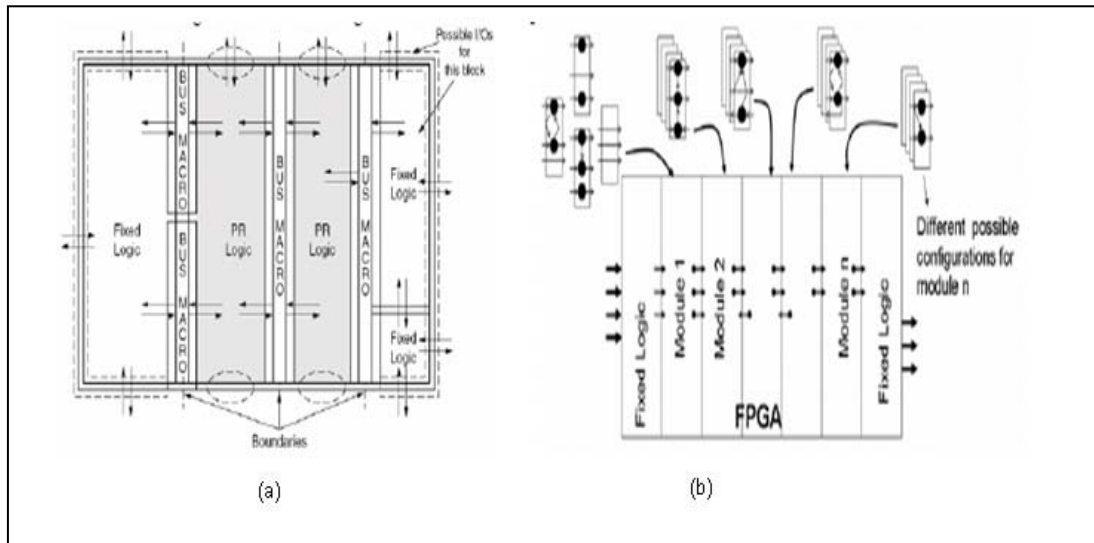


Figure II. 33 (a) Reconfiguration partielle dans VIRTEX-II (b) Layout des blocs reconfigurables [168]

Les méthodes d'amélioration de la densité font appel à la reconfiguration de l'FPGA. Les bonnes performances sont obtenues si le temps de reconfiguration est faible comparé au temps d'exécution total. Ce point sera traité en détail dans le chapitre IV.

#### II.3.4.3.2 Utilisation du circuit FPGA en Co-processeur et carte accélératrice *Le circuit RAPTOR2000*

Dans [141] Mario Pormann et ses co-auteurs, ont présenté un système de prototypage universel appelé RAPTOR2000. Ce dernier est dédié à l'implémentation des réseaux de neurones de manière générale et plus particulièrement les cartes auto-organisatrices de Kohonen, connues sous le nom de « Self Organizing Map ».

RAPTOR2000 est une carte accélératrice constituée principalement de trois à six modules pour applications spécifiques (*ASM*) comme le montre la figure II.34. Un *bus Local* permet la communication interne entre les différents modules *ASM* d'une part, et entre le PC hôte et l'*ASM*, d'autre part. Un autre bus « Broadcast Bus » peut être utilisé pour la communication simultanée entre les *ASMs*. De plus, une mémoire SRAM est accessible aux différents modules *ASM* via le « *Broadcast bus* ». D'autres fonctions, tel que la gestion des mémoires, l'arbitrage des bus, la détection d'erreurs et la configuration JTAG sont intégrés dans les deux circuits programmables de type CPLDs. Le bus PCI permet de connecter RAPTOR2000 à l'ordinateur hôte ou bien la station de travail. RAPTOR2000 est aussi équipé d'une carte fille utilisée pour la simulation des réseaux de neurones.

Chaque module *ASM* peut être équipé de plusieurs types de circuits FPGAs de la famille Virtex, émulant des circuits complexes de 400.000 à 2.5 millions de portes logiques. La configuration d'un module *ASM* peut être réalisée soit par l'ordinateur hôte, soit à travers un autre bus PCI ou bien un autre module *ASM*. Ainsi, il est possible qu'un FPGA se reconfigure par les données localisées quelque part dans le système. Le temps de reconfiguration est inférieur à 20 ms. En utilisant cinq circuits FPGA de la famille Virtex-E, RAPTOR2000 est capable de réaliser des réseaux de neurones 190 fois plus rapides que ceux implémentés en software

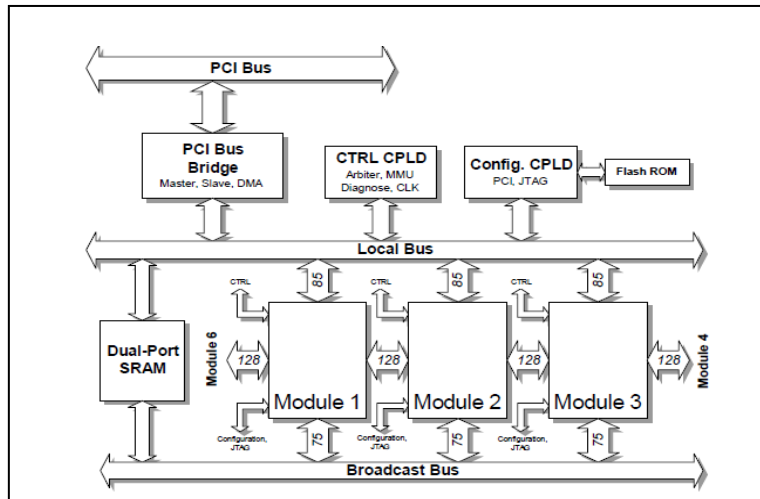


Figure II.34 Architecture de RAPTOR2000 [169]

**Le circuit Heterogenous spiking neural network**

Dans [142] Shayani et ses co-auteurs ont proposé une implémentation sur FPGA du model évolutif du neurone impulsionnel en introduisant une architecture flexible pour le model du dendrite et du soma d'un neurone. La figure II.35 (a) montre l'architecture générale du neurone qui consiste en une chaine d'unités synaptiques. La figure II.35 (b) montre l'architecture d'une synapse et la figure II.35 (c) celle d'un Soma. Dans l'article [138], les auteurs montrent qu'un réseau de 161 neurones et 1610 synapses, peut être implémenté sur un circuit FPGA-VIRTEX5. Il occupe 85% des ressources de ce dernier à une fréquence de 160Mhz. Les auteurs montrent que ce circuit est 43/ plus rapide que celui proposé dans [143]

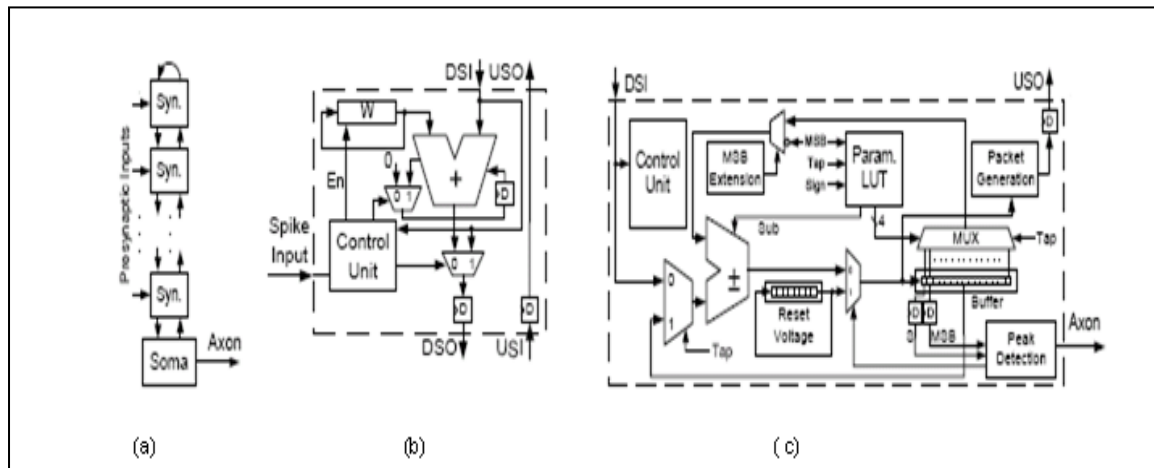


Figure II. 35 Réseau de neurone impulsionnel sur FPGA. (a) architecture du neurone (b) archietcure de la synapse. (c) architecture du Soma [170]

**II.3.4.3.3 Utilisation des descriptions Soft-Core**

Le terme « Soft-Core » est utilisé pour désigner une description d'un composant « core » en langage VHDL, Verilog ou autre. Ce genre de descriptions permet une flexibilité et un grand degré d'indépendance vis-à-vis de la technologie cible. Néanmoins le circuit obtenue est souvent non optimisé de point de vue surface et performances.

Parmi les travaux qui ont réalisés des descriptions « Soft-Core » des réseaux de neurones, nous citons le circuit **SoftTOTEM NC3003** [140] et le travail présenté dans [143], par M. Diepenhorst et ses co-auteurs.

### Le circuit SoftTOTEM

**SoftTOTEM NC3003** est un processeur Soft-Core, développé et commercialisé par la société italienne *NeuriCam*, en collaboration avec l'université italienne *Tronto* et l'université anglaise *Kent* [144]. Le premier circuit, sous le nom de TOTEM NC3000, a été conçu en 1999 en utilisant l'approche de conception « Full Custom » ; et est considéré comme un « Hard Core ». Par la suite, et afin de pouvoir suivre l'évolution technologique et les contraintes « time to market », une version « Soft-Core » a été proposée en 2002, d'où le nom de *SoftTOTEM*.

Le NC3003 a été conçu pour implémenter le perceptron multicouche, mais il peut aussi être utilisé pour implémenter certaines fonctions de traitement du signal tel que le filtrage, la convolution, les opérations de multiplication et d'accumulation. La figure II. 36, montre l'architecture simplifiée du circuit. Les caractéristiques principales de cette dernière sont résumées comme suit :

- L'architecture est de type SIMD, optimisée pour l'exécution des opérations de multiplication et d'accumulation (MAC) en 1 cycle d'horloge sur 32 unités de traitement MAC montées en parallèle.
- Les données sont représentées en virgule fixe et en complément à 2.
- Les poids synaptiques sont stockés dans des mémoires de 256x 8bits
- Les données ont un format limité : Entrées : 16 bits, poids synaptiques : 8 bits et la sortie sur 16 bits.
- Le registre à décalage Barrelet « Barrel shift register » [145] est utilisé pour la troncature du résultat.
- Le NC3003 permet d'implémenter un réseau multicouche pouvant contenir jusqu'à 32 neurones avec des performances de 750 Millions MAC opérations @ 25 MHz. D'après le Datasheet [146], le NC3003 est portable en ASIC ou bien en circuit FPGA de Xilinx.

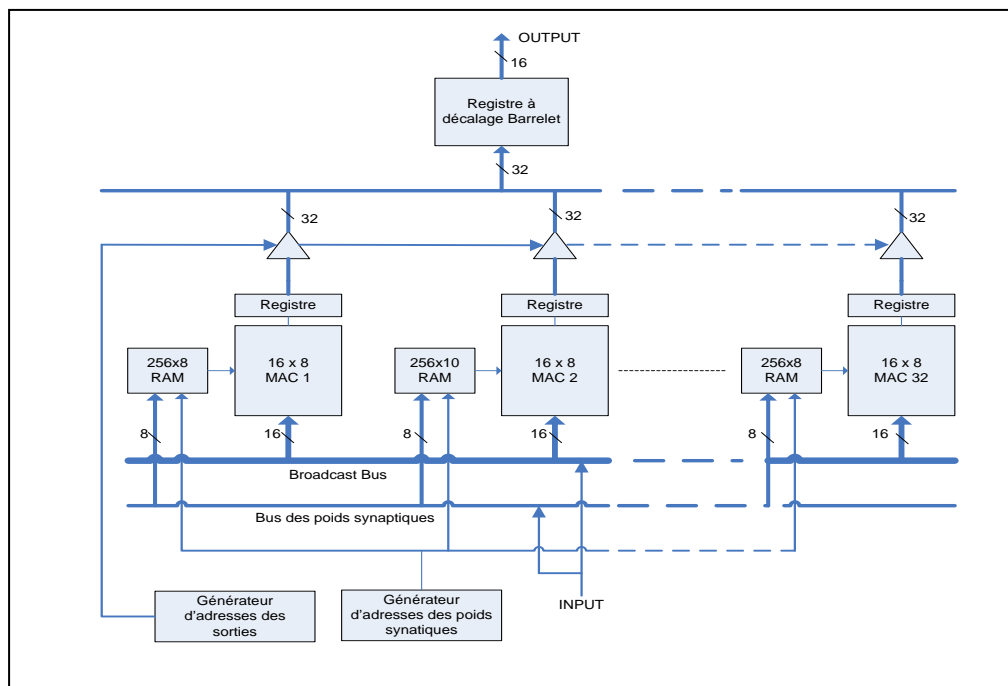


Figure II.36 Schéma simplifiée du SoftTOTEM NC3003

***Le circuit Virtual Neural Processor (VNP)***

Le deuxième exemple d'utilisation des descriptions « Soft-Core » pour l'implémentation des réseaux de neurones est le concept du processeur neuronal virtuel « Virtual Neural Processor », VNP, développé par M. Diepenhorst dans [147]. La figure II.37 montre les étapes principales de cette approche.

- La première étape consiste à établir un modèle du neurone et du réseau de neurone en question. A partir de ce modèle, une description VHDL au niveau comportementale du réseau est générée moyennant les outils de génération du code VHDL. Par la suite la description comportementale est convertie en une description structurelle du réseau de neurones moyennant une bibliothèque de portes logiques élémentaires. La dernière étape consiste à introduire la description structurelle obtenue dans un outil de synthèse pour cibler une implémentation d'un circuit ASIC ou bien d'un FPGA.

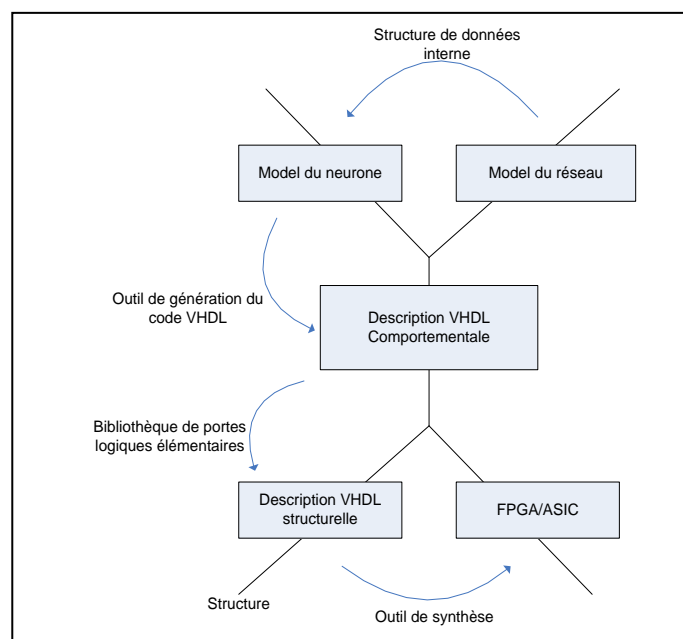


Figure II.37 Description de l'approche VNP

- Deux autres exemples de Soft-Core pour les réseaux de neurone se trouvent dans les références [148] et [1495].

**II.3.4.4 Utilisation des systèmes embarqués et systèmes sur puce « SOC »**

Avec l'évolution de la technologie, il est devenu possible d'intégrer sur une seule puce des systèmes complexes qui intègrent des processeurs, des circuits de control, de mémoires, des circuits d'interface ainsi qu'une partie du système d'exploitation. On parle alors de systèmes intégrés sur une puce « System on Chip », SOC, ou systèmes embarqués.

L'approche de conception de ces systèmes est très complexe et diffère de l'approche classique de conception des circuits ASIC ou FPGA ; car d'une part le problème de la conception ne se situe pas seulement au niveau de vérification de la fonctionnalité du circuit, mais aussi et avant tout, il faut garantir une bonne communication entre les différents IPs ou composants du système. En d'autres termes, il faut définir à chaque instant, le circuit qui commande et joue le rôle de maître tel le processeur, et celui qui est commandé et joue le rôle d'esclave tel que les mémoires par exemple. Certains circuits peuvent jouer le rôle de maître

et d'esclave en même temps, ceci en fonction de leur communication avec les différents IPs du système en question.

D'autre part, l'approche de conception des SOC nécessite une expertise tant en software, qu'en algorithmique et aussi en hardware. La création d'équipes multi disciplinaires n'est pas chose facile et même lorsqu'elle existe, la réalisation d'un projet de ce genre nécessite beaucoup de temps à cause du problème de la communication entre les différentes équipes (chacune sa spécialité).

Afin de résoudre ce problème, actuellement plusieurs approches sont entraînées d'émerger pour proposer des plateformes de conception basées sur la conception conjointe « Codesign » et le partitionnement hardware-software. Néanmoins, les solutions ne sont toujours pas complètes et les travaux de recherche sont à leur âge de pierre comparés aux approches de conception classiques.

Malgré cette difficulté et malgré la complexité des réseaux de neurones, certains travaux sont entrain d'émerger afin d'utiliser l'approche de conception des SOC pour l'implémentation des réseaux de neurones.

- Dans [150], Denis F. Wolf et ses co-auteurs, ont proposé un SOC pour implémenter l'algorithme du perceptron multicouche (MLP). La figure II.38 montre l'architecture du système. Cette dernière est composée d'un processeur *Nios* connecté à un réseau de neurones.

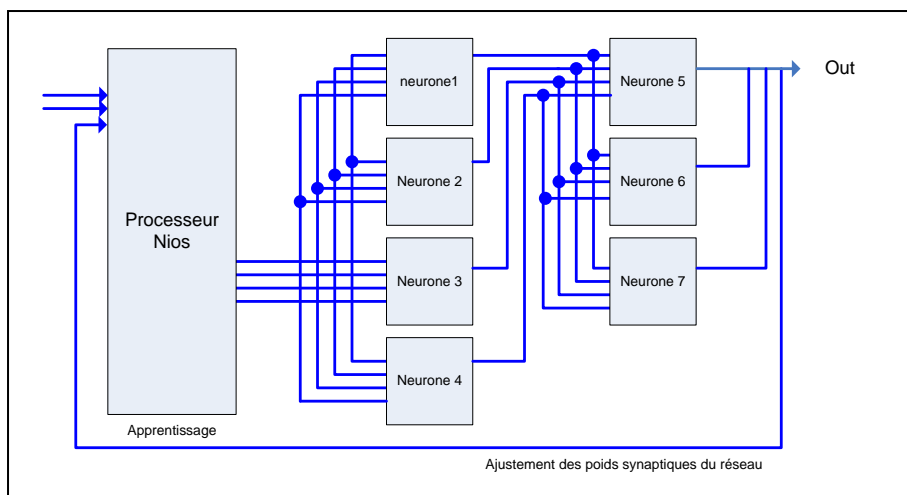


Figure II.38 Architecture du SOC pour le perceptron multicouche

Le processeur *Nios* est développé par la compagnie Altera [151]. Il réalise l'apprentissage software du MLP. Le réseau de neurone se charge de la partie lecture et propagation des données du MLP. Ainsi la sortie du réseau de neurones est reliée à l'entrée du processeur qui effectue la mise à jour des poids synaptiques. Les nouvelles valeurs sont communiquées au réseau de neurones et la même opération est répétée jusqu'à ce que la convergence soit obtenue.

Le model hardware du réseau a été conçu en utilisant l'outil *Quartus* d'Altera [152]. Chaque neurone est composé de quatre multiplieurs permettant la multiplication de chaque entrée avec le poids synaptiques correspondants. Les résultats ainsi obtenus sont additionnés aux valeurs des biais et enfin la fonction de transfert délivre la sortie du neurone. La partie software de l'algorithme (apprentissage) a été implémentée en utilisant le langage C. Elle est exécutée par le processeur *Nios*.

Afin de pouvoir comparer les performances de l'approche SOC proposée, avec celles existantes, quatre types de tests ont été effectués : le premier concerne l'implémentation de toutes les phases de l'algorithme du MLP dans le circuit FPGA, sans le processeur *Nios*. Le deuxième test concerne l'exécution du MLP entièrement sur le processeur *Nios*. Le troisième test concerne l'exécution du MLP sur le SOC et le quatrième concerne l'exécution du MLP sur un ordinateur de type Intel Pentium-III.

Les résultats obtenus ont montré que le Pentium-III permet un temps d'exécution de  $23 \mu\text{s}@550\text{Mhz}$ , le processeur *Nios* donne un résultat de  $144\mu\text{s}@40\text{Mhz}$ , le SOC permet un temps d'exécution de  $600\text{ns}@40\text{Mhz}$  et le MLP sans le processeur permet un temps d'exécution de  $32\text{ns}@40\text{Mhz}$ . Il est évident que l'implémentation de tout le réseau de neurones dans un hardware dédié tel que le FPGA est la plus performante comparée aux autres approches. Cependant, cette approche est difficile à implémenter et manque de flexibilité. L'utilisation de l'approche SOC permet de réaliser un compromis entre la flexibilité des implémentations software et les performances obtenues en utilisant un hardware dédié. Ainsi il est possible de réaliser une partie de l'algorithme en software et laisser la partie hardware s'occuper du traitement des parties critiques de l'algorithme.

L'ensemble du SOC a été implémenté sur le circuit FPGA APEX20k200E d'Altera. Le processeur a occupé 25% de la surface du circuit et le réseau de neurone 50% de cette dernière.

- Un autre exemple de l'utilisation des systèmes embarqués pour l'implémentation des réseaux de neurone, concerne le travail proposé dans [153]. La figure II.39 montre l'architecture du système proposé.

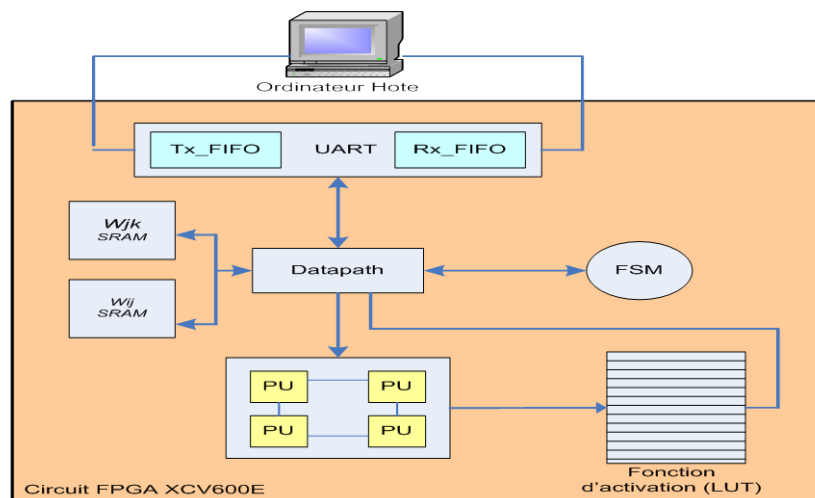


Figure II.39 Architecture HW/SW pour l'implémentation du MLP

Le système embarqué est formé en appliquant les quatre transformations principales

- partitionnement de l'algorithme
- Allocation des partitions au microprocesseur et unités de traitement hardware
- Planification du temps d'exécution des différentes fonctions de l'algorithme
- Mapping de l'algorithme en software et en hardware
- 

Partant de ce principe, le système a été partitionné en deux parties : software et hardware. Un gestionnaire des tâches, installé dans l'ordinateur hôte, agit comme un contrôleur maître. L'apprentissage est exécuté en software dans l'ordinateur hôte afin de déterminer les poids synaptiques qui satisfassent les critères de convergence. Pour une

architecture spécifique du MLP, l'interface RS232 assure le transfert des poids synaptiques vers le circuit FPGA (XCV600E) qui agit en co-processeur. La SRAM est utilisée pour stocker les poids synaptiques. Une fois l'unité software détermine les poids synaptiques appropriés, la partie de propagation est exécutée dans les unités de traitement (PU) pour déterminer la sortie désirée du réseau. Une machine d'état finie (FSM) assure la synchronisation du module de propagation. Les unités de traitement (PU) sont arrangées en architecture de grille « *grid architecture* » afin d'assurer la multiplication vecteur- matrice. Chaque (PU) est constitué d'un circuit multiplieur accumulateur (MAC). La fonction d'activation est implémentée par une table mémoire (LUT).

L'approche proposée a été comparée avec plusieurs travaux de la littérature dont le RRANN [94], et le circuit Hitachi-WSI [43]. Les résultats obtenus ont montré que le circuit proposé permet d'atteindre des performances de 327 MCPS. Ces résultats sont meilleurs comparé au circuit RRANN (722 KCUPS) et au circuit de Hitachi-WSI (138 MCPS).

## II.4 Conclusion

Dans ce chapitre, nous avons présenté un aperçu général sur les différentes possibilités d'implémentation hardware des réseaux de neurones. A travers, notre nouvelle approche de classification du hardware neuronal, nous avons montré, que durant ces trois dernières décennies, il ya eu une nette évolution dans la conception des réseaux de neurones. Le choix d'une technologie d'implémentation par rapport à une autre, dépend du domaine d'application et des contraintes de conception. Dans chaque type d'implémentation un certains nombre de problèmes doit être pris en charge par le concepteur. Ces problèmes peuvent être énumérés comme suit :

- Le parallélisme qui permet le traitement en temps réel
- La flexibilité et l'adaptabilité du circuit
- Les performances relativement par rapport à la complexité des connexions synaptiques
- La représentation des données
- La précision
- La conception du circuit réalisant la fonction de transfert
- La conception du bloc d'activation
- L'apprentissage « on chip » ou bien « off chip »

Parmi les différents types implémentations citées, celles utilisant les circuits FPGA semblent être très attractifs car elles permettent un compromis entre les performances des circuits ASIC et ceux des circuits standard chip.

Dans le domaine de la conception, les axes porteurs et pour lesquels les travaux de recherche sont à leur âge de pierre concernent :

- L'implémentation « *on chip* » des réseaux de neurones
- L'exploitation de la reprogrammabilité dynamique des circuits FPGAs
- L'utilisation des langages de description de haut niveau tel que le langage VHDL, le langage Verilog, le langage System-C et le Handel-C pour la réalisation de Soft-Cores et des systèmes sur puce (SoC)
- L'application du concept de « reuse » et de « design for reuse »
- L'utilisation de la virgule flottante pour la représentation des données
- L'exploitation des dernières familles des circuits FPGAs de type VIRTEX.

Ces points seront discutés lors des prochains chapitres.



---

# *CHAPITRE III*

---

**IMPLEMENTATION DE L'ALGORITHME RPG SUR FPGA**

### III.1 Introduction

Ce chapitre est consacré à l'implémentation de l'algorithme de la rétropropagation du gradient, RPG, sur les familles des circuits FPGA de Xilinx.

Nous avons montré au chapitre II, que parmi les paramètres qu'il faut prendre en considération lors de l'implémentation des réseaux de neurones :

- Le parallélisme qui permet le traitement en temps réel
- Les performances relativement par rapport à la complexité des connexions synaptiques
- La précision des données
- La surface de silicium
- L'apprentissage « on chip » ou bien « off chip »

Pour atteindre ces objectifs, une bonne implémentation hardware doit avoir les propriétés architecturales suivantes :

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

Les points cités ci-dessus sont pris en considération pour l'implémentation de l'algorithme RPG.

D'abord nous commençons par un rappel de l'algorithme RPG, et les différents types d'implémentation hardware : « *Off chip training* » et « *On chip training* »

La section III.3 est consacrée à l'étude des différents types de parallélisme et le choix du langage Handel-C pour la description et l'implémentation parallèle de l'algorithme RPG.

Les résultats de cette étude sont utilisés pour proposer une architecture pour l'implémentation de l'algorithme RPG qui sera présentée dans la section III.4.

La section III.5 est consacrée à l'évaluation des performances de l'architecture proposée. Une attention particulière est donnée quand aux choix du type du multiplieur, de la précision des données, de l'évolution de la technologie des circuits FPGAs et l'influence de chacun de ces paramètres sur les performances du réseau de neurone.

Enfin nous terminerons par une conclusion.

### III.2 Rappel sur l'algorithme de rétropropagation du gradient

L'algorithme RPG présenté dans le chapitre I (Figure I.16) s'exécute en trois phases consécutives:

- Phase de propagation.
- Phase de rétropropagation ou calcul d'erreur
- Phase de mise à jour des poids synaptiques.

Dans la *phase de propagation* un échantillon  $x_i$  est appliqué à la couche d'entrée et le signal résultant est propagé dans le réseau. Ainsi, pour chaque index  $i$  (index du neurone) et index  $j$  (index de la couche)

$$\mu_j^{[i]} = \sum_{i=1}^{n_0} W_{ji}^{[i]} x_i \quad (\text{III.1})$$

$$O_j^i = f(x_j) = \frac{1}{1 + \exp(-\mu_j^i)} \quad (\text{III.2})$$

Où,  $\mu_j^i$ , représente la somme pondérée des poids synaptiques et  $o_j^i$  la sortie de la fonction d'activation sigmoïde,  $\Psi$ .

La **phase de rétropropagation** calcule les erreurs locales,  $\delta$ , pour chaque couche en commençant par la couche de sortie jusqu'à la couche d'entrée :

$$\delta_i^L = f'(u_i^L)(d_i - y_i) \tag{III.3}$$

$$\delta_j^{l-1} = f'(u_j^{l-1}) \sum_{i=1}^{N_l} w_{ij} \delta_i^l \quad 1 \leq i \leq N_l \quad , \quad 1 \leq l \leq L \tag{III.4}$$

Dans la **phase de mise à jour des poids synaptiques**, on calcule les nouvelles valeurs des poids synaptiques selon les formules :

$$w_{ij}^l(t+1) = w_{ij}^l(t) + \Delta w_{ij}^l(t) \tag{III.5}$$

$$\Delta w_{ij}^l(t) = \eta \delta_i^l y_j^{l-1} \tag{III.6}$$

Il existe deux modes d'apprentissage : le mode « online » et le mode « batch ». Dans le mode « online », on met à jour les poids synaptique pour chaque échantillon d'apprentissage présenté dans le réseau de neurones. Dans le mode « batch », on calcule d'abord les erreurs pour tous les échantillons sans mettre à jour les poids (on additionne les erreurs) et lorsque l'ensemble des données est passé dans le réseau, on applique la rétropropagation en utilisant l'erreur totale.

### III.2.1 Les différentes approches d'implémentation de l'algorithme RPG

L'algorithme de la rétropropagation du gradient a été implémenté en utilisant les deux approches « *off-chip training* » et « *on-chip training* ».

Dans l'approche « *off chip training* » l'apprentissage (phases de **propagation** et **rétropropagation**) est réalisé en software permettant ainsi de générer les poids synaptiques. L'implémentation hardware du réseau de neurone consiste à implémenter la phase de **généralisation** de l'algorithme RPG et dont le principe est similaire à l'implémentation de la phase de propagation. La figure III.1 montre le principe de base de l'approche « *off chip training* ».

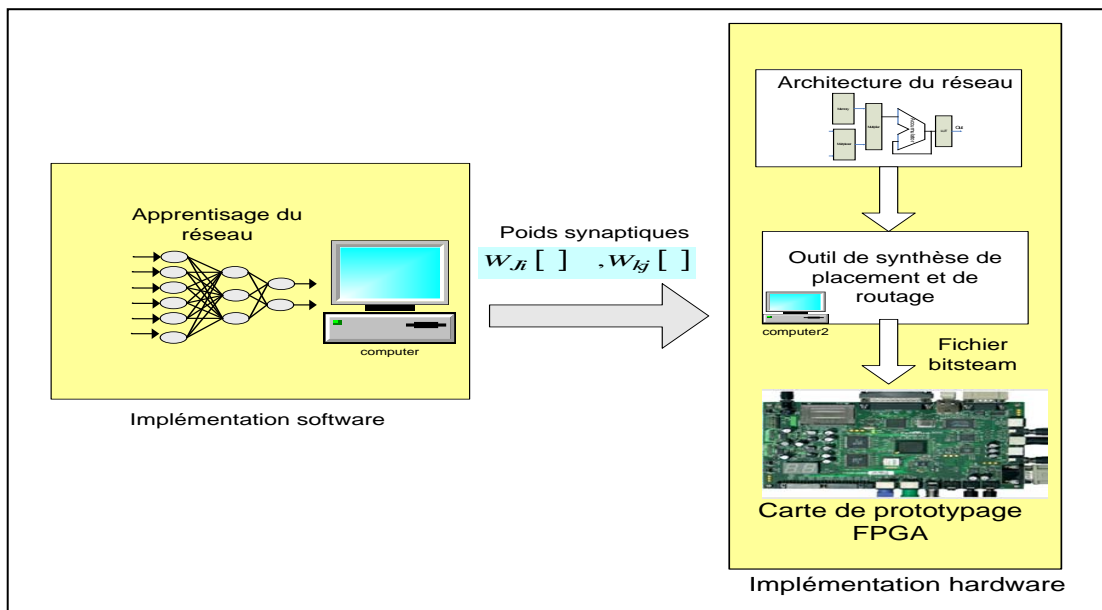


Figure III.1 Principe de l'approche « off chip training »

La programmation de l'apprentissage de l'algorithme RPG se fait moyennant le langage de description « C », ou bien l'utilisation du *Neural tool box* de Matlab [22]. La figure III.2, montre le pseudo code de l'algorithme RPG. Notons que dans ce cas, le code s'exécute séquentiellement dans la machine.

Une fois les poids synaptiques déterminés, et l'architecture spécifiée, on utilise un outil de synthèse qui fait le mapping de l'architecture en portes logiques. Par la suite un outil de placement et de routage de ces blocs logiques est utilisé et enfin le fichier bitstream obtenu est programmé dans la carte de prototype FPGA.

```

Define MAXI, MAXJ, MAXK      */Nombre de neurones de la couche d'entrée, la
                             */couche cachée et la couche de sortie
Define  $\eta$ ,  $\alpha$           */Taux d'apprentissage et momentum
Define  $\varepsilon$            */ 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 calculée
Float Wji[MAXJ][MAXI]        */ Matrice des poids synaptiques entre couche cachée
                             */ et couche d'entrée
Float Wkj[MAXK][MAXJ]        */ Matrice des poids synaptiques entre couche de
                             */ sortie et couche cachée
:
:
:
Faire
Pour chaque exemple de la base d'apprentissage Faire
- Lecture des échantillons du signal d'entrée
- Lecture de la sortie correspondante
- Initialisation des poids synaptiques
- Calcul de la phase de propagation
- Calcul de l'erreur
- Calcul de la phase de rétropropagation
- Ajustement des poids synaptiques
Fin Pour
Tant que l'erreur >  $\varepsilon$  et le nombre d'itération < Max_n

```

Figure III.2- Pseudo code de l'algorithme RPG

Contrairement à l'approche « *off-chip training* », l'approche « *on chip training* », désigne l'implémentation des trois phases de l'algorithme à savoir la phase de *propagation*, la phase de *rétropropagation* et la phase de *calcul d'erreur* dans un seul circuit ou plusieurs circuits FPGAs. Dans ce dernier cas, des problèmes de communication entre les FPGA peuvent surgir.

Aussi, et vu les contraintes de surface et de vitesse des premières familles des circuits FPGA, plus particulièrement les familles XC3000, XC4000, XC6000 ainsi que les premières familles de la famille Virtex des circuits FPGA de Xilinx, la plupart des travaux de la littérature se sont intéressés à l'implémentation « *off chip training* » de l'algorithme RPG.

Actuellement, le défi à relever est de pouvoir implémenter tout le réseau de neurone sur un seul circuit FPGA.

### III.3 Etude du parallélisme pour l'implémentation de l'algorithme RPG

Avant de discuter le parallélisme de l'algorithme RPG, il est nécessaire de définir la terminologie des différents types de parallélisme des réseaux de neurones.

#### III.3.1 Aperçu des différents types de parallélismes

Une excellente introduction aux différentes implantations parallèles du calcul neuronal a été proposée par Nordstrom et Svensson dans [29]. Un réseau de neurones peut être programmé de la façon suivante :

*Pour chaque session d'apprentissage*  
 || *Pour chaque exemple d'apprentissage*  
 || *Pour chaque couche du réseau*  
 || *Pour tous les neurones de la couche*  
 || *Pour tous les synapses du neurone*  
 || *Pour tous les bits de la valeur des synapses*

Ceci montre qu'il existe six degrés de parallélisme comme suit :

Parallélisme de session d'apprentissage  
 || Parallélisme des exemples d'apprentissage  
 || Parallélisme des couches  
 || Parallélisme des neurones  
 || Parallélisme des synapses  
 || Parallélisme des bits

Dans ce qui suit chaque type de parallélisme est présenté

#### III.3.1.1 Parallélisme de session d'apprentissage

Dans ce cas, plusieurs types ou tailles d'architectures neuronales, mais également plusieurs paramètres expérimentaux (liés à l'algorithme d'apprentissages, par exemple) peuvent être traités de façon concurrente. Le parallélisme de session, consiste à allouer à chaque processeur de la machine parallèle, une architecture ou une taille de réseau de neurones comme le montre la figure III.3 ci-dessus.

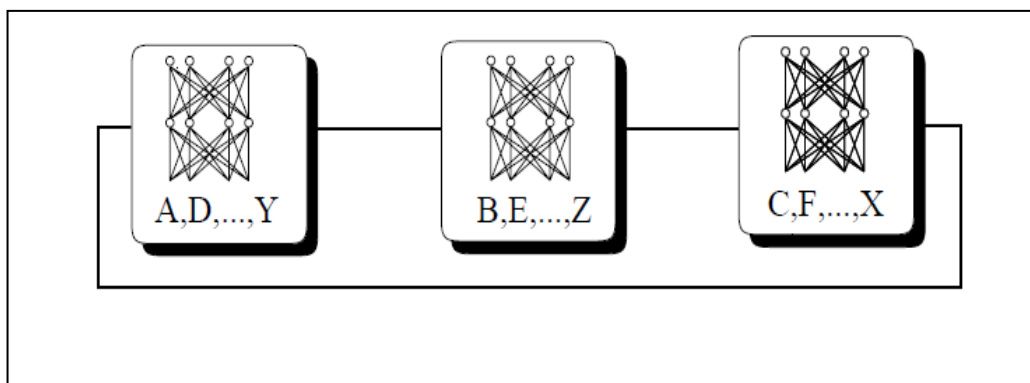


Figure III.3 Parallélisme de session d'apprentissage

#### III.3.1.2 Parallélisme d'exemple d'apprentissage

Lorsqu'un algorithme d'apprentissage gère simultanément plusieurs exemples, les calculs relatifs à chacun d'eux peuvent être traités de façon concurrente. Dans ce cas, chaque processeur ou neurone traite un exemple d'apprentissage comme présenté dans la figure III.4

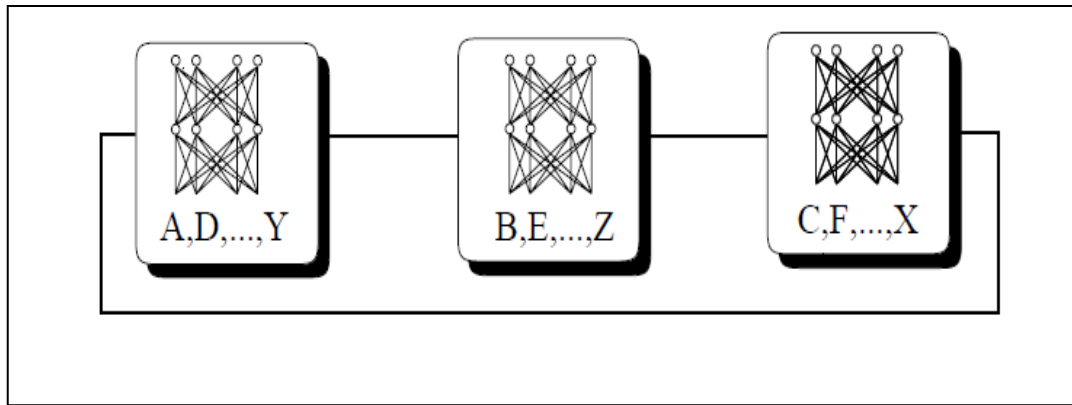


Figure III.4 Parallélisme d'exemples d'apprentissage

**III.3.1.3 Parallélisme de Couche**

Dans ce cas, les couches d'un même réseau de neurones travaillent de façon concurrente. Ce cas est considéré lorsqu'on veut traiter simultanément plusieurs exemples d'apprentissages : cas de l'apprentissage « batch mode » ou bien lorsqu'on veut traiter simultanément les phases de *propagation*, de *rétropropagation* et de *calcul d'erreur* de l'algorithme RPG.

Le pipeline est souvent utilisé dans le parallélisme de couche. La Figure III.5 montre un cas où deux exemples d'apprentissages sont traités en parallèle.

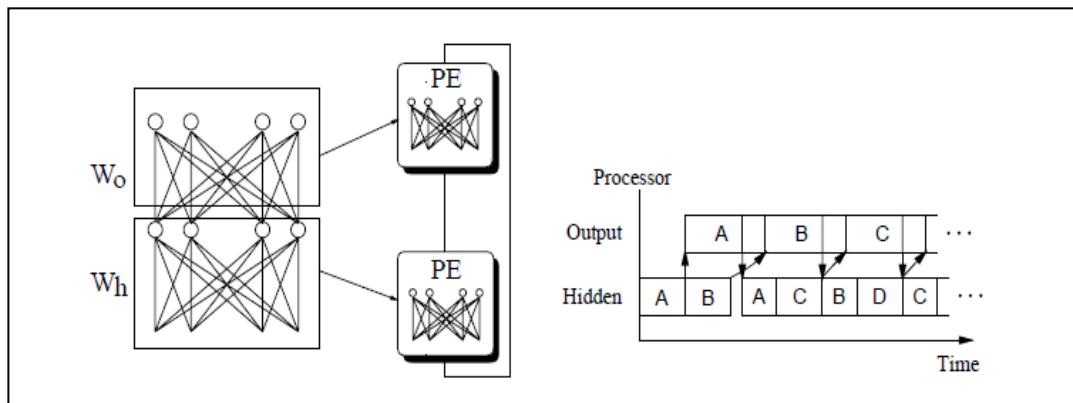


Figure III.5 Parallélisme de couche

**III.3.1.4 Parallélisme de neurone**

Le parallélisme des neurones est utilisé lorsque les différents neurones d'une même couche peuvent travailler de façon concurrente. C'est cette forme de parallélisme qui est en général désignée lorsqu'on emploie l'expression *parallélisme neuronal*. La figure III.6 montre un exemple d'un réseau multicouche, où les neurones d'une même couche travaillent de façon concurrente et les trois couches utilisent les mêmes neurones.

**III.3.1.5 Parallélisme des synapses**

Lorsque plusieurs synapses d'un même neurone travaillent de façon concurrente, on parle alors de parallélisme des synapses ou des connexions (Figure III.7). Cette forme de parallélisme constitue le plus haut degré de parallélisme qui peut être atteint.

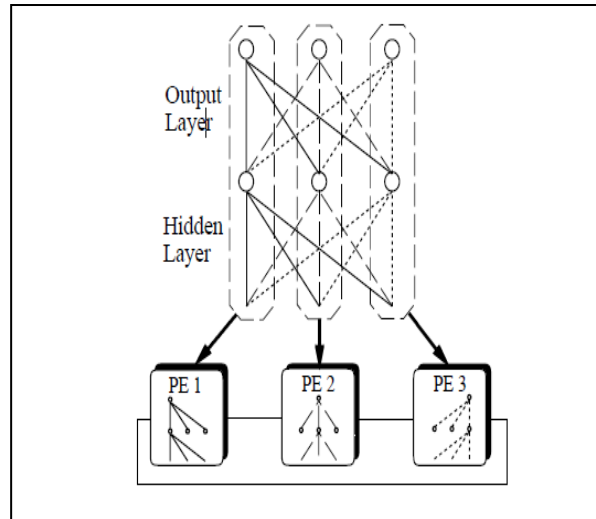


Figure III.6 Parallélisme de neurones

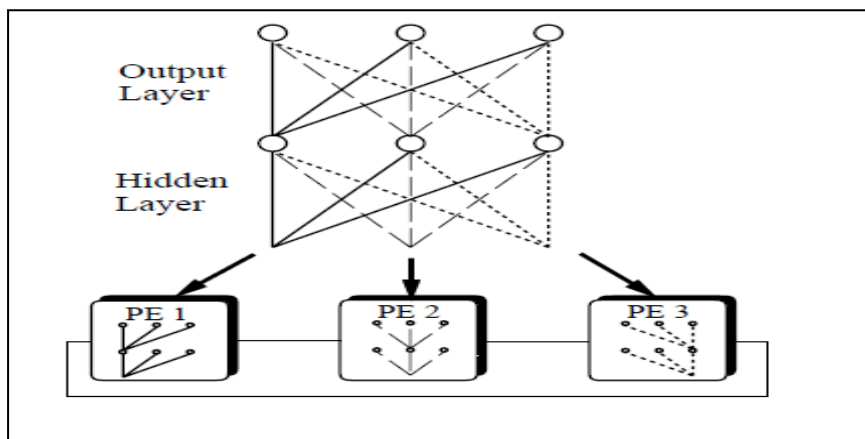


Figure III.7 Parallélisme des synapses

### III.3.1.6 Parallélisme des bits

Ce type est utilisé lorsque tous les bits d'une donnée travaillent simultanément. Cette forme de parallélisme n'est pas propre aux réseaux de neurones et demande beaucoup de ressources hardware.

### III.3.1.7 Synthèse des différents types de parallélisme

Les deux premiers types de parallélisme sont classés dans la catégorie *parallélisme à gros grain*. Les autres types sont classés dans la catégorie de *parallélisme à grain fin*.

Notons que dans le cas de l'algorithme RPG, les exemples d'apprentissages sont considérés séquentiellement l'un après l'autre, et l'erreur est calculée localement. Par conséquent, le parallélisme de session d'apprentissage, le parallélisme des exemples d'apprentissage et le parallélisme des couches cités précédemment ne peuvent être pris en considération.

De même le parallélisme des bits nécessite beaucoup de ressources et une arithmétique particulière. Par conséquent, nous nous intéressons uniquement à l'implémentation du parallélisme des neurones et des synapses.

### III.3.2 Choix du langage de description matérielle

Lors de l'implémentation d'un algorithme, le concepteur se trouve souvent confronté au choix du langage de description et du niveau d'abstraction utilisé. En effet, dans la littérature une panoplie de langages est offerte. Certains d'entre eux, comme le *VHDL* [154] et le *VERILOG* [155] représentent des langages de description au niveau RTL (*Register Transfer level*); d'autres comme le *SystemC* [156], le *StreamC* [157], Le *Handel-C* [158] et le *SpecC* [159] représentent des langages de description au niveau algorithmique.

Pour notre part, nous avons choisi d'utiliser les deux langages VHDL et Handel-C pour la description parallèle de l'algorithme RPG et de comparer entre les deux styles de description. Notre choix est justifié par leur disponibilité à notre niveau.

#### III.3.2.1 Le langage VHDL et le RTL

Le langage VHDL est un standard industriel de la norme IEEE 1987 (VHDL'87), puis révisé en 1992 (VHDL'92) et en 1997 (VHDL'97).

Initialement, le VHDL a été conçu avec une sémantique de simulation. Par la suite, il a été utilisé pour la synthèse et la preuve formelle

En ce qui concerne le domaine de la synthèse, son utilisation est réduite à un sous ensemble qui correspond à des représentations digitales identifiables. La synthèse au niveau RTL [160] permet de générer une description plus élevée que la description au niveau des portes logiques et qui est formée d'additionneurs, de multiplieurs, de mémoires et de registres.

Généralement, une description d'un circuit en VHDL est constituée :

- D'une ou plusieurs bibliothèques dans lesquelles sont stockés des unités de conception.
- Les unités de conception peuvent être de cinq catégories différentes qui sont les vues externes des modèles (entité et architecture), les vues externes et internes des paquetages et les configurations.
- Les unités de conception contiennent des descriptions d'actions séquentielles ou concurrentes faisant intervenir des objets appartenant à trois classes : constantes, variables et signaux.
- Chaque objet est typé et son type appartient à une des quatre familles de type scalaire, composite et accès aux fichiers.

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, à son tour ce dernier peut être modélisé par 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 exploitées dans la description du réseau de neurone.

##### III.3.2.1.1 Description VHDL du neurone

Pour des buts de comparaison, nous considérons la description VHDL d'un neurone car il représente l'élément de base de n'importe quel réseau de neurones. La figure III.8 représente l'architecture modélisant le neurone. Celle-ci est constituée des éléments suivants:

- Le neurone effectue la somme pondérée des poids synaptiques modélisée par l'équation (III.1). Cette architecture est constituée d'un multiplieur, d'un additionneur et d'un accumulateur. L'ensemble constitue un multiplieur accumulateur (MAC)
- Le signal d'entrée et les poids synaptiques sont stockés dans des mémoires.
- Une table de transfert (LUT) implémente la fonction d'activation de l'équation (III.2)
- Le neurone est contrôlé par une unité de control



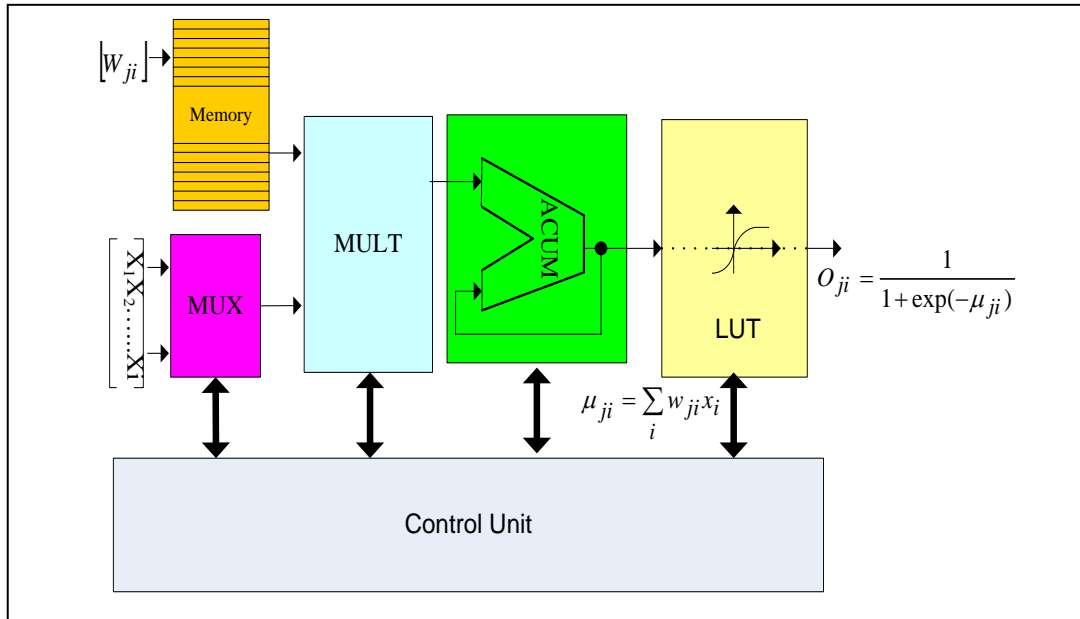


Figure III.8 Architecture du neurone

**III.3.2.1.2 Le bloc des poids synaptiques:**

Pour les poids synaptiques, il suffit d'utiliser une mémoire de type RAM pour le stockage de ces valeurs. Cependant et comme il faut initialiser d'abord ces poids à des valeurs aléatoires et ensuite faire leur mise à jour et stockage, nous proposons d'ajouter un multiplexeur à l'entrée de la RAM pour le mixage entre les valeurs d'initialisation et ceux de la mise à jour. Ainsi notre bloc des poids synaptiques est constitué d'un multiplexeur et d'une RAM (Figure III.9)

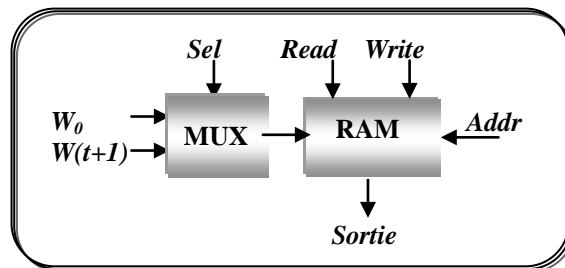


Figure III.9 Architecture du bloc de poids synaptique

On utilise une RAM avec les caractéristiques suivantes :

- Une *Entrée* d'initialisation dotée d'un signal d'écriture, un signal de lecture et un signal d'adressage; cette entrée est précédée par un multiplexeur à deux entrées, la première pour l'initialisation des poids synaptique (  $W_0$  valeur initiale aléatoire ), la deuxième pour la valeur de la mise à jour des poids synaptiques.
- Une *Sortie* pour la lecture des valeurs écrites.

**III.3.2.1.3 le bloc MAC**

Il est composé d'un multiplieur est d'un accumulateur comme le montre la Figure III.10

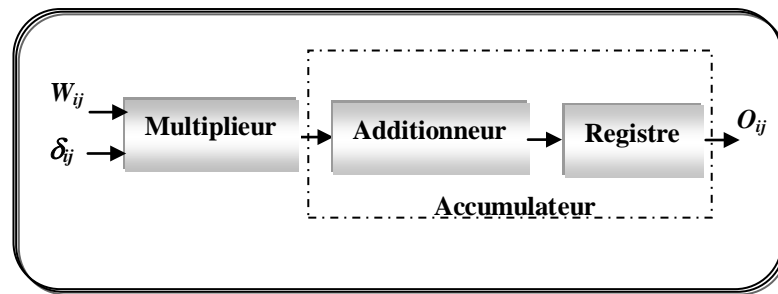


Figure III.10 Architecture du bloc MAC

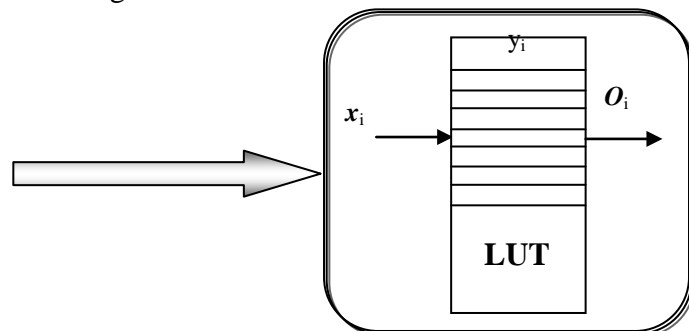
- Le multiplieur est utilisé pour la multiplication synaptique, qui est nécessaire pour le calcul de la somme pondérée. Dans cette multiplication on prend la valeur de l'activation de la couche précédente et on la multiplie par la valeur du poids synaptique du neurone. Cette valeur est enregistrée et accumulée avec les produits des autres neurones de la couche.
- L'accumulateur est composé d'un additionneur et d'un registre. Il commence par le chargement de la première valeur reçue du multiplieur et il la charge dans le registre puis il additionne les données reçues du multiplieur avec celles qui se trouvent dans le registre. Le registre de données est un simple registre de  $N$  bits de type parallèle/parallèle. Ce registre sauvegarde la valeur de la multiplication synaptique des différents exemples pour qu'elle soit accumulée avec celle qui la suit, et garde la dernière valeur de l'accumulation pour la transmettre à la prochaine couche.

Ainsi, on utilise le multiplieur et l'accumulateur pour calculer la valeur du neurone et le registre pour la stocker. Une fois cette valeur est calculée, on utilise le bloc de la fonction d'activation pour le calcul de la valeur d'activation du neurone.

#### III.3.2.1.4 Bloc d'activation :

Le rôle du bloc de la fonction d'activation est de prendre la valeur de la somme pondérée calculée par le neurone et de lui appliquer la fonction sigmoïde, pour générer la valeur d'activation du neurone. la fonction sigmoïde est définie comme suit :

$$y_j = f(x_j) = \frac{1}{1 + \exp(-\alpha x_j)}$$



La modélisation de cette fonction nécessite l'implémentation des opérations de division et d'exponentiel. Chacun de ces opérateurs nécessite un nombre important des ressources de l'FPGA. Pour remédier à ce problème on utilise les Look-Up-Tables (LUTs) du circuit FPGA pour la modélisation de cette fonction.

Les LUTs sont des mémoires de type ROM adressées par la valeur de la somme pondérée. Les résultats de la fonction d'activation seront chargés dans cette ROM, donc on aura la valeur de la somme pondérée en entrée et celle de l'activation en sortie.

### III.3.2.1.5 Unité du contrôle du neurone

C'est une unité sous forme de diagramme d'état (de type Moor), qui est utilisée pour contrôler l'exécution des différentes étapes du calcul du neurone. On rappelle : Dans une machine de Moore, les sorties ne varient que lorsque le système change d'état (Les sorties ne sont fonctions que des états et non pas des états et des données d'entrée). Par contre la machine de Mealy produit des sorties après la réception de données d'entrée. La figure III.12 montre l'organigramme du fonctionnement du neurone.

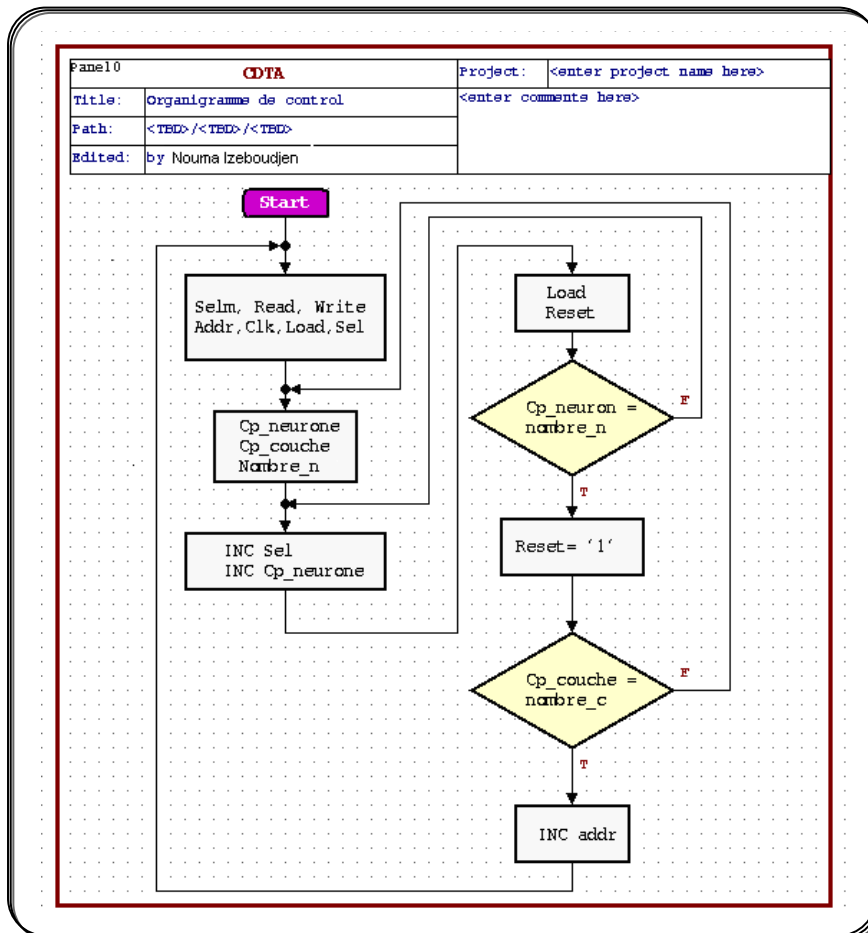


Figure III.11 Organigramme de contrôle du neurone

Le diagramme de la figure III.11 est divisé en quatre phases de contrôle : phase du départ, phase d'initialisation, phase de multiplication et d'accumulation et phase de stockage de la somme pondérée.

#### Phase de départ :

Comme tous les diagrammes d'état dans l'implémentation des réseaux de neurones, ce diagramme commence par un état de *RESET* : un état durant lequel le système à contrôler reste en attente tant qu'il n'est pas en phase d'exécution, il est maintenu dans cet état par le contrôle global de ce module. Il sert aussi à la remise à zéro des différentes variables pour entamer l'exécution d'une autre itération de ce module. Si cet état est activé par le contrôle global, alors le contrôle passe à la phase d'initialisation.

*Phase d'initialisation:*

Cette phase a pour rôle d'initialiser les valeurs des adresses des RAMs et des multiplexeurs de poids synaptique et de neurones, l'accumulateur et initialisé en mettant le *RESET* à la valeur "0" et le *LOAD* à "1". Une fois cette phase est terminée, on passe à la multiplication synaptique et l'accumulation.

*Phase de la multiplication synaptique et de l'accumulation:*

Cette phase est responsable du contrôle de la partie du neurone qui calcule la somme pondérée. On commence d'abord par le chargement de la valeur des poids synaptiques dans le multiplieur. Après la multiplication l'accumulateur charge cette valeur en mettant le *LOAD* à la valeur "1". Une fois cette opération terminée le sélecteur du neurone est incrémenté et la deuxième valeur est chargée dans le multiplieur, mais cette fois le *LOAD* est mise à zéro pour que cette valeur soit accumulée avec celle qui la précède. Ce processus est répété *N* fois (*N* : nombre du neurone dans la couche).

*Stockage de la somme pondérée:*

Après avoir calculé la somme pondérée, on met la valeur du *RESET* à la valeur "1" pour qu'elle soit stockée et transmise à la LUT pour générer la fonction d'activation.

La Figure III.12 montre le pseudo-code VHDL du neurone. Notons, qu'il faut d'abord décrire en VHDL les circuits mémoire, multiplieur, additionneur et accumulateur ; ensuite ces circuits sont instanciés comme le montre la figure III.12.

```

entity Neurone is
  generic(nb_bit:integer:= ;addr:integer:=);-- taille du mot, nombre d'adresse
  port(in1: 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 (deep: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 (deep: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_n1P);
end;

```

Figure III.12 Pseudo-code VHDL d'un neurone

### III.3.2.2 Le langage Handel-C

Le langage Handel-C [154] a été développé par la compagnie CELOXICA dans le souci d'implémenter les algorithmes en un temps très court permettant ainsi un prototypage rapide des circuits intégrés. Il est fortement basé sur la syntaxe du langage C-ANSI [161] à laquelle on a rajouté un ensemble d'instructions nécessaires pour l'implémentation hardware (Figure III.13). Les principales caractéristiques sont :

- Il permet de réaliser une implémentation hardware parallèle par le biais de l'instruction *par {...}*
- L'implantation séquentielle est réalisée par le biais de l'instruction *seq {...}*
- L'affectation de variable prend exactement un cycle d'horloge
- Les tailles des variables peuvent être définies au bit près, ceci afin d'occuper le moins de ressources possible (exemple : `unsigned int 10 x;` instancie une variable de 10 bits)
- L'implémentation hardware d'une mémoire est réalisée par les primitives *ram[ ..]* et *rom[...]*. Ces instructions permettent de générer directement les blocs de mémoire enfouis dans les FPGAs ou bien externes à celui-ci.
- Les types non entiers ne sont pas implémentés directement dans le langage, mais des bibliothèques de fonctions existent, tant en virgule flottante qu'en virgule fixe, ce qui permet d'adapter facilement certains algorithmes; néanmoins l'utilisation de nombres flottants est évidemment très pénalisante en termes de ressources matérielles et de temps de traitement.

La compilation d'un code Handel-C peut générer au choix :

- Un exécutable natif Windows pour effectuer la simulation du code (grâce à un des compilateurs natifs susnommés ou au GCC)
- Une netlist EDIF pour cibler directement un back-end FPGA Altera, Xilinx ou Actel
- Un code VHDL (et maintenant Verilog) destiné à un outil de synthèse, afin de cibler un ASIC et/ou s'interfacer avec du code VHDL ou bien Verilog développé.
- Lors de l'implémentation Hardware (génération de netlist EDIF), un programme écrit en Handel-C est transformé en une partie opérative et une partie de contrôle. La partie opérative dérive de l'assignement des variables et des expressions et la partie control dérive des expressions de contrôle tel que les boucles « Loops » ainsi que de la structure du programme (séquentiel ou parallèle).
- L'utilisation des machine d'état n'est pas nécessaire car elle est prise en charge indirectement par les instructions : *par {...}* et *seq{...}*

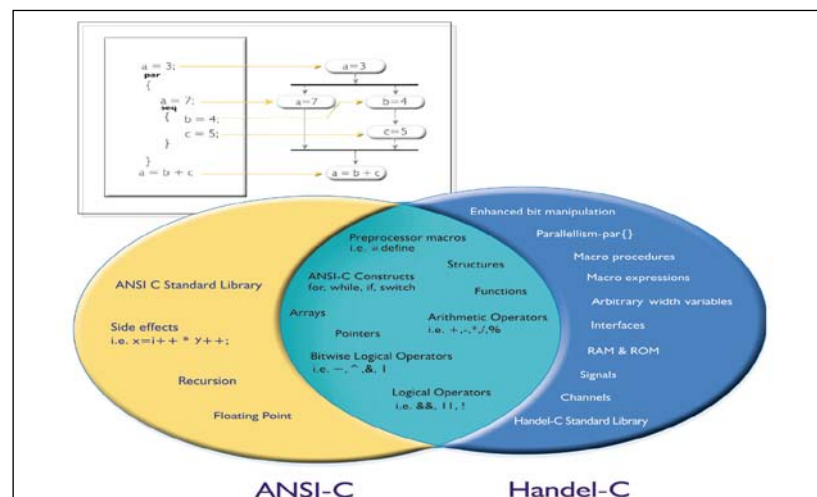


Figure III.13 Le Handel-C/ANSI-C

En terme de hardware, une description séquentielle en Handel-C est représentée comme suit (figure III.14) :

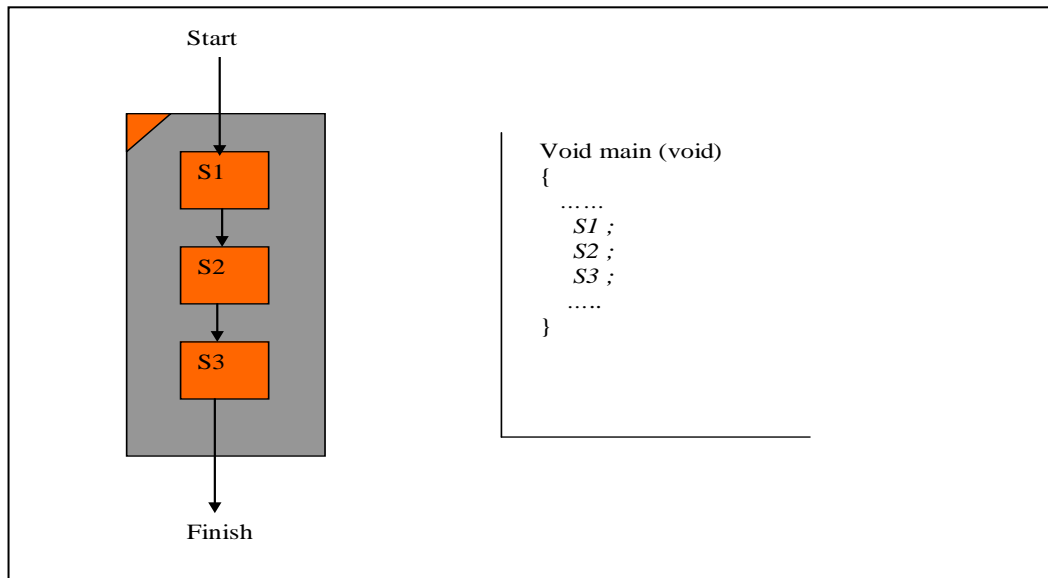


Figure III.14 description séquentielle

De même, l'équivalent hardware d'une instruction parallèle *par{... }* est représenté comme suit (Figure III.15) :

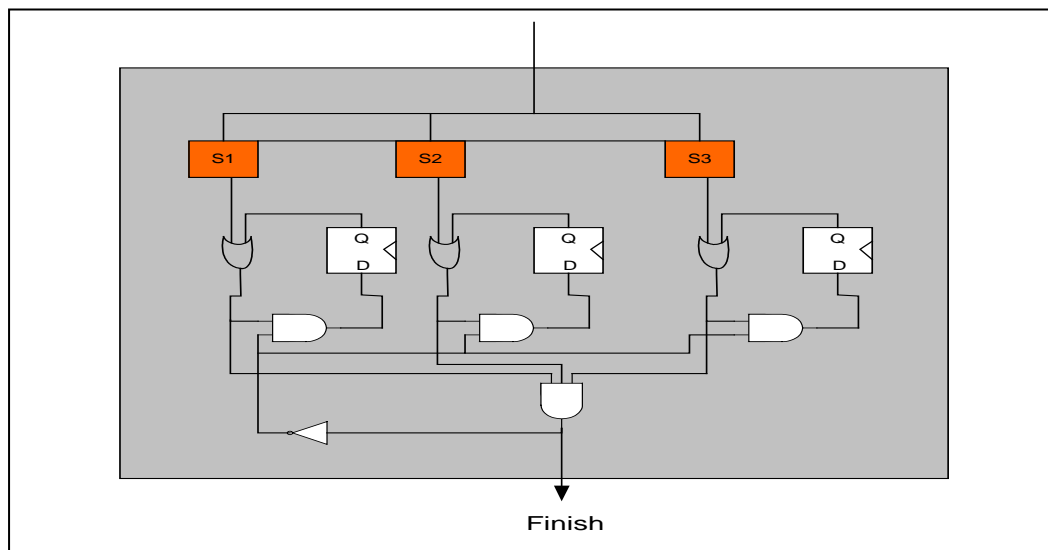


Figure III.15 Equivalent hardware de la description parallèle: *par { ... }*

### III.3.2.2.1 Description du neurone en Handel-C

Comme nous l'avons signalé, le Handel-C s'inspire du langage C auquel on rajoute des instructions pour l'implémentation sur un hardware spécifique.

La description des équations (III.1) et (III.2) modélisant le neurone, est donnée par le pseudo code ci-dessous (Figure III.16):

```

Void main (void)
s = 0;
for (i=0;i<height1;i=i+1)
{
    // Produit des poids synaptiques avec le vecteur d'entrée
     $\mu [i]=(0@W[i]) * (0@ x[i]);$ 
    // Calcul de la somme pondérée
    Out[i]= s + Y[i];
    s= out[i];
    // Calcul de la fonction d'activation
    z[i]= LUT[i][out[i];
}
End;
    
```

Figure III.16 Pseudo-code d'un neurone en langage Handel-C

### III.3.2.3 Etude comparative entre le VHDL et le Handel-C

Afin d'étudier les solutions que peut apporter le langage Handel-C aux problèmes d'implémentations des réseaux de neurones, nous avons effectué une comparaison entre le VHDL et le Handel-C. Les performances étudiées pour la comparaison sont : la surface exprimée en nombre de bloc logique configurables (CLB), la fréquence (MHZ), et la complexité du programme en nombre de lignes.

Le premier circuit FPGA ciblé est celui de la famille Virtex -E XCV2000E [162] spécifique à la carte de prototypage RC1000 de Celoxica [163] disponible à notre niveau. Le circuit XCV2000E est constitué des éléments suivants (Figure III.17):

- Blocs logiques configurables (**CLB**) nécessaires pour implémenter les fonctions logiques
- Les blocs d'entrée sortie **IOB** réalisant l'interface avec l'extérieur
- Des blocs mémoires dédiés (**BRAM**)
- Des circuits **DLL** pour la distribution de l'horloge
- Une matrice de routage **VersaRing**

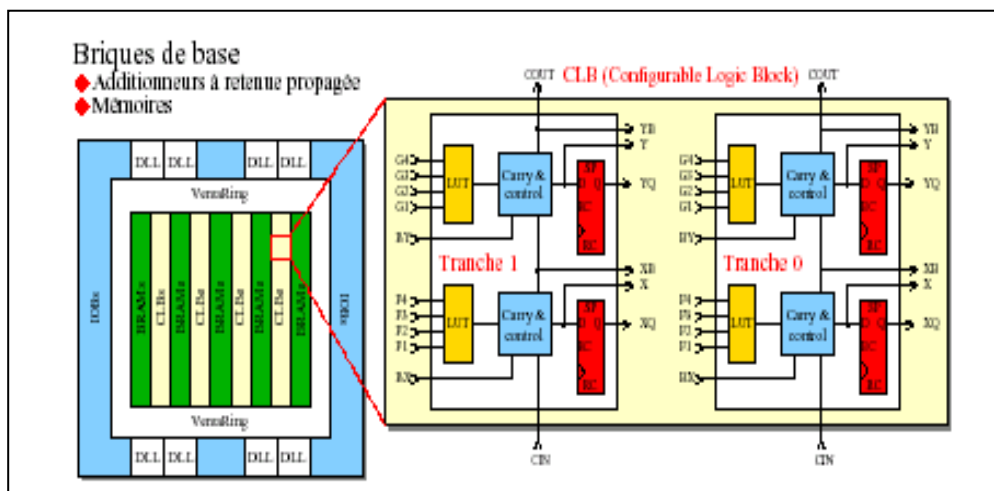


Figure III.17 Architecture du circuit Virtex-E

Le bloc de base du CLB est une cellule logique (LC) contenant un générateur de fonction à 4 entrées, une logique de retenue « Carry & Control » et un circuit de stockage. Chaque CLB contient 4 cellules logiques (LC). Le générateur de fonction est implémenté comme une mémoire LUT à 4 entrées (Figure III.18). En plus des blocs mémoires dédiés (BRAM), d'autres mémoires peuvent être implémentées à partir des LUT. On parle alors de mémoires distribuées. La famille Virtex-E peut implémenter jusqu'à 518400 de portes logiques et offre une fréquence allant jusqu'à 133 MHz.

Le tableau III.1 représente les résultats de comparaison obtenus.

Tableau III.1 : Comparaison entre le VHDL et le Handel-C

Performance Code	Nombre de CLB	Fréquence maximale (MHZ)	Temps de réponse (délai en ns)	Nombre de lignes
VHDL	32	194.59	23.711	1440
Handel-C	97	195	23	102
Rapport	~3	~ 1	~ 1	14

A travers le tableau III.1, nous pouvons conclure que le mapping des langages VHDL et Handel-C en hardware permet d'obtenir des implémentations ayant le même ordre de fréquence et de délai d'exécution. Néanmoins, en termes de surface, une description Handel-C résulte en une surface trois fois plus grande que celle d'une description en VHDL. Les figures III.18 et III.19 montre les résultats de l'implémentation (après placement et routage) des deux descriptions en utilisant l'outil ISE Fondation [164].

Notons que :

*La fréquence maximale,  $F_{max}$* , est calculée après synthèse du circuit par

$$F_{max} = 1/\text{Période minimale} \quad (\text{III.7})$$

*Le temps de réponse,  $Tr$* , est donné par l'équation (III.8) :

$$Tr = (\text{Minimum period} + \text{Minimum input arrival time before clock} + \text{Maximum output required time after clock} + \text{Maximum combinational path delay}) \quad (\text{III.8})$$

En termes de description du circuit, le tableau III.1 montre que le Handel-C est nettement plus simple et plus rapide à décrire que le langage VHDL (le rapport du nombre de ligne du code entre le VHDL et le Handel-C est de l'ordre 14). De plus, le Handel-C offre la possibilité de décrire le parallélisme par une simple instruction : *par{ ... }*.



Par conséquent, nous proposons d'utiliser le langage Handel-C pour étudier les différents types de parallélisme de l'algorithme RPG. Ceci nous permettra de choisir l'architecture parallèle la plus appropriée pour une implémentation sur FPGA. Une fois l'architecture choisie, nous utiliserons le langage VHDL pour la description et l'implémentation de cette dernière et ceci afin d'optimiser la surface du réseau de neurone.

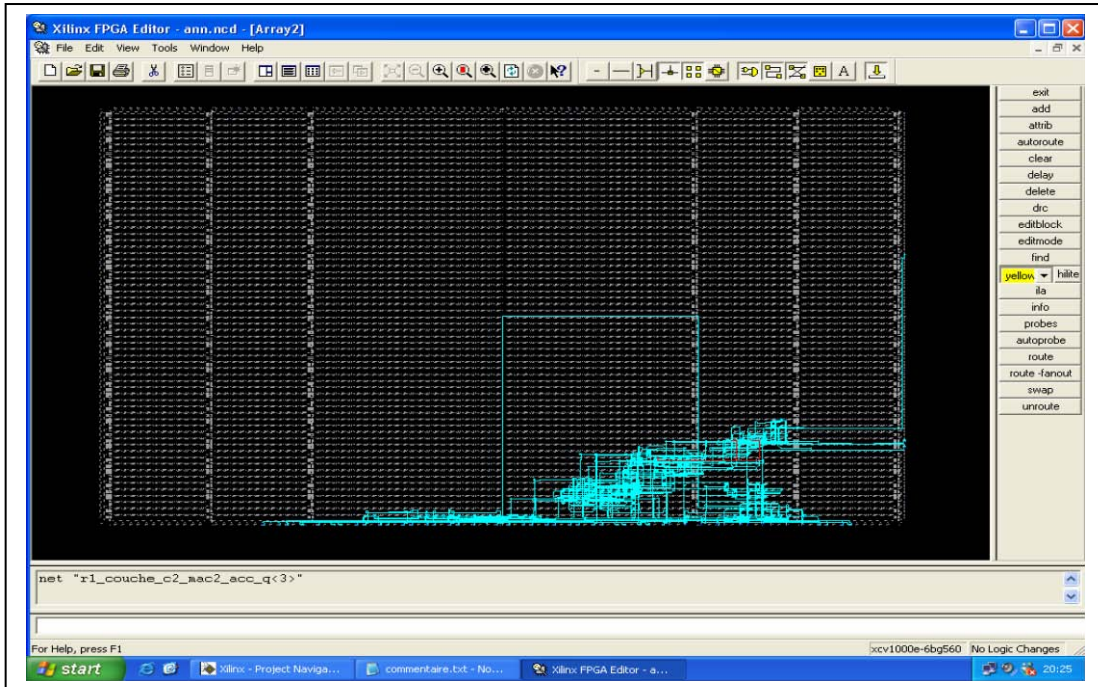


Figure III.18 Layout d'une description VHDL

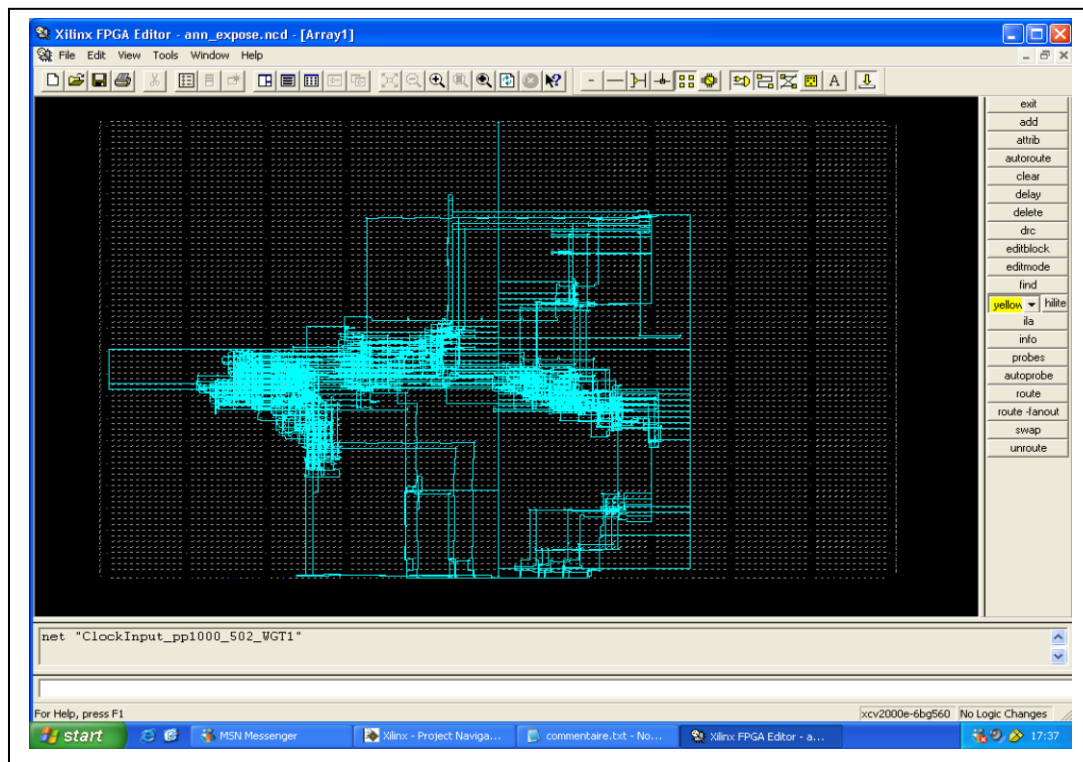


Figure III.19 Layout d'une description Handel-C

### III.3.3 Le Handel-C et le parallélisme des neurones

Soit un réseau de neurones de trois couches constitués de  $N$  neurones à la couche cachée et  $M$  neurones dans la couche de sortie. L'extrapolation de l'équation III.1 à la première couche s'écrit comme suit :

$$\left\{ \begin{array}{l} \mu_1^1 = W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + \dots + W_{1n}x_n \\ \mu_2^1 = W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + \dots + W_{2n}x_n \\ \vdots \\ \mu_n^1 = W_{n1}x_1 + W_{n2}x_2 + W_{n3}x_3 + \dots + W_{nn}x_n \end{array} \right.$$

```
#define Neuron_in height 1 // Nombre de neurones de la couche d'entrée
#define Neuron_hid height2 // Nombre des neurones de la couche cachée

Void main (void)
for (i=0;i<height1;i=i+1)
{
    par (j=0;j<height2;j=j+1) // Parallélisme des neurones
    {
        μ [j][i] = (0@W[j][i]) * (0@ x[i]);
    }
}
for (j=0, j< height2, j++)
{ s=0;
    for(i=0;i<height1;i++)
    {
        out[j]= s + μ [j][i];
        s= out[j];
    }
}
par (j=0; j<height2;j++)
{ z[j]= LUT[j][out[j]]; } // Calcul des fonctions d'activation en parallèle
```

Figure III.20 Pseudo Code Handel-C pour exprimer le parallélisme des neurones

Le parallélisme des neurones est un parallélisme vertical car toutes les sommes pondérées,  $\mu_j^i$ , se calculent en même temps. Dans ce cas, Il faut un multiplieur par neurone. Le nombre de multiplieurs dans une couche est égal au nombre de neurones de la même couche. Pour un réseau de neurones de deux couches le nombre de multiplieurs est de  $(N+M)$ .

Le parallélisme des neurones est réalisé grâce à l'instruction `par{ }` qui sera appliquée a tous les neurones de la couche cachée et de la couche de sortie du réseau de neurone en question. Le pseudo code de ce parallélisme est donné dans la Figure III.20.

### III.3.4 Le Handel-C et le parallélisme des synapses

Considérons, de nouveau, le système défini par l'équation III.1 comme suit :

$$\begin{cases}
 \mu_1^1 = W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + \dots + W_{1n}x_n \\
 \mu_2^1 = W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + \dots + W_{2n}x_n \\
 \vdots \\
 \mu_n^1 = W_{n1}x_1 + W_{n2}x_2 + W_{n3}x_3 + \dots + W_{nn}x_n
 \end{cases}$$

Le parallélisme des synapses est un *parallélisme horizontal* car tous les produits internes des poids synaptiques avec le vecteur d'entrée,  $W_{ji}x_i$ , se calculent en parallèle pour chaque neurone  $\mu_j^i$ . Dans ce cas, le nombre de multiplieurs dans un neurone est égal au nombre des synapses. Dans une couche donnée, le nombre de synapses dans un neurone est égal au nombre de sorties de la couche précédente. Pour la couche cachée, chaque neurone reçoit un nombre de synapses proportionnel aux entrées du réseau de neurones. Si le réseau de neurones est de grande taille, alors le nombre de multiplieurs, i.e. nombre de ressources hardware, est donc très grand. Si  $N$  est le nombre de neurones de la couche cachée et  $S$  le nombre des synapses. Alors il faudrait  $(N \times S)$  multiplieurs pour effectuer le parallélisme des synapses et des neurones dans une couche. Pour un réseau de trois couches, il faudrait  $(N \times S) + (N \times M)$  multiplieurs. La figure III.21 Montre le pseudo code Handel-C pour effectuer le parallélisme des synapses et des neurones en même temps.

```

#define Neuron_in height1 // Nombre de neurones de la couche d'entrée
#define Neuron_hid height2 // Nombre des neurones de la couche cachée
Void main (void)
par (i=0;i<height1;i=i+1) // Parallélisme des synapses
{
    par (j=0;j<height2;j=j+1) // Parallélisme des neurones
    {
         $\mu [j][i] = (0@W[j][i]) * (0@ x[i]);$ 
        out[j]= s +  $\mu [j][i]$ ;
        s= out[j];
    }
}
par (j=0; j<height2;j++)
{ z[j]= LUT[j][out[j]]; } // Calcul des fonctions d'activation en parallèle

```

Figure III.21 Pseudo Code Handel-C pour exprimer le parallélisme des synapses.

### III.3.5 Comparaison entre le parallélisme des neurones et le parallélisme des synapses

Afin de comparer entre les performances obtenues des deux types de parallélismes, nous considérons l'implémentation d'un réseau de neurone utilisé pour résoudre le problème du « OU exclusif » ou XOR-logique.

Rappelons, que le problème du XOR-logique est un benchmark classique de séparation non linéaire [165]. Il est utilisé dans plusieurs applications plus particulièrement les jeux. Soit  $Z$ , la sortie de la porte XOR et  $X1, X2$  ses entrées. La fonction  $Z = f(x1, x2)$  est donnée par le tableau III.2. Dans la figure III.22, les cercles vides représentent la sortie  $Z = 0$  et les cercles remplis représentent la sortie  $Z = 1$ . La figure III.23 montre que dans le plan  $XY$ , il n'existe pas de ligne permettant de séparer entre les deux états « 0 » et « 1 »

Tableau III.2. Table de vérité de la fonction XOR

X1	X2	Z
0	0	0
0	1	1
1	0	1
1	1	0

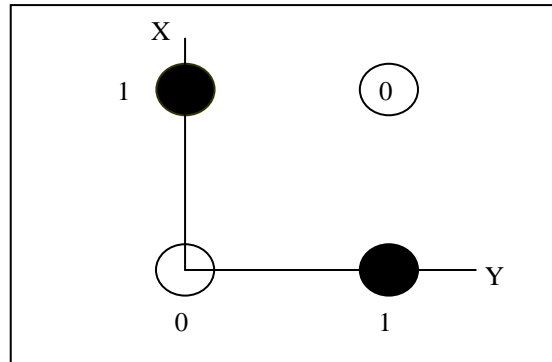


Figure III.22 Problème de séparation non linéaire du XOR-logique

L'algorithme de la rétropropagation du gradient est utilisé pour résoudre le problème de séparation non linéaire du problème du XOR-logique. La topologie minimum d'un réseau de neurone pour résoudre ce problème est constituée de trois couches, dont une couche cachée. La couche d'entrée est constituée de deux neurones, la couche cachée de deux neurones et la couche de sortie d'un seul neurone. Nous avons donc un réseau de neurones de dimension (2,2, 1) comme le montre la figure III.23

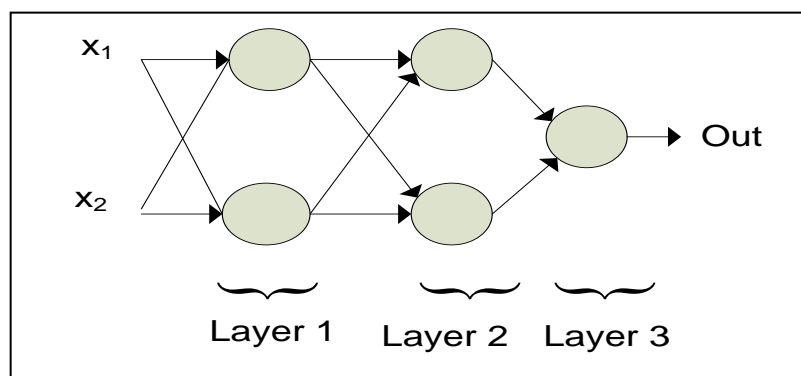


Figure III. 23 Topologie du réseau de neurone pour résoudre le problème du XOR\_logique

Pour l'implémentation du réseau (2, 2, 1), nous avons d'abord ciblé le circuit FPGA de la famille VIRETX-E (XCV2000E) spécifique à la carte RC1000 de Celoxica. Néanmoins, les résultats de synthèse ont montré une sur utilisation des ressources du circuit FPGA.

Par la suite, nous avons ciblé des circuits FPGAs de famille Virtex-II [166] qui est supérieure au circuit XCV2000E et qui, en plus des blocs logiques configurables et des mémoires embarqués, intègre des multiplieurs. La figure III.24 montre l'architecture générale des circuits FPGA de la famille Virtex-II. Celle-ci est constituée des éléments suivants :

- Blocs logiques configurables (**CLB**) nécessaires pour implémenter les fonctions logiques
- Les blocs d'entrée sortie **IOB** réalisant l'interface avec l'extérieure
- Des blocs mémoires dédiés
- Des blocs de multiplieurs dédiés très performants
- Des circuits **DCM** (Digital Clock Manager) permettant de réaliser une large gamme de fréquences de l'horloge (multiplication et division de l'horloge).

Les CLBs incluent quatre parties identiques appelées slice. Les quatre slices son organisés en deux colonnes de deux slices chacune avec deux chaînes de retenue logique et une chaîne pour le décalage (Figure III.25 (a)). Chaque slice contient :

- Deux générateurs de la fonction (F & G) à quatre entrées chacune.
- Deux éléments du stockage.
- Des portes de la logique arithmétiques.
- Grands multiplexeurs.
- Une large fonctionnalité.
- Une logique de retenue rapide.
- Une chaîne de cascade Horizontale (porte OU).

Chaque générateur de fonction peut être programmé comme une LUT à 4 entrées, une mémoire distribuée de 16 bits ou bien un registre à décalage variable (Figure III.25 (b)).

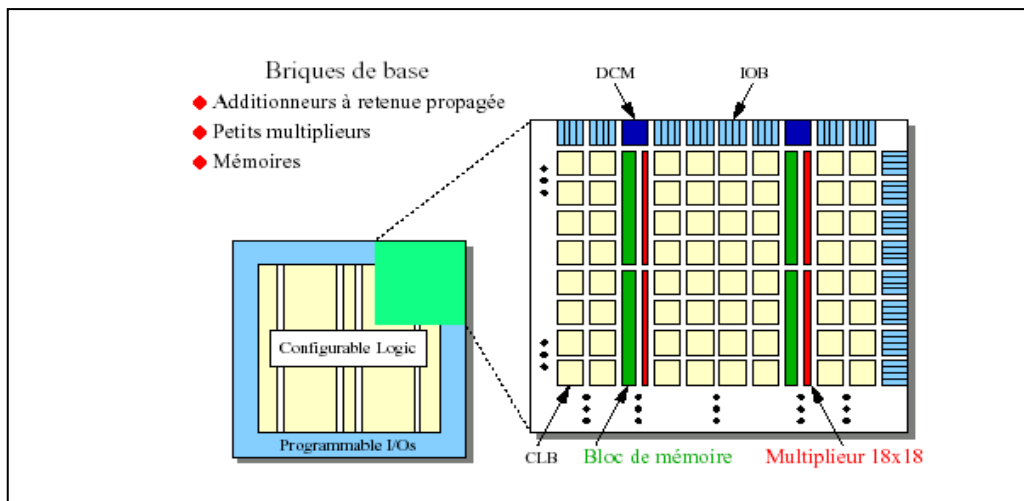


Figure III.24 Architecture générale du circuit VIRTEX-II

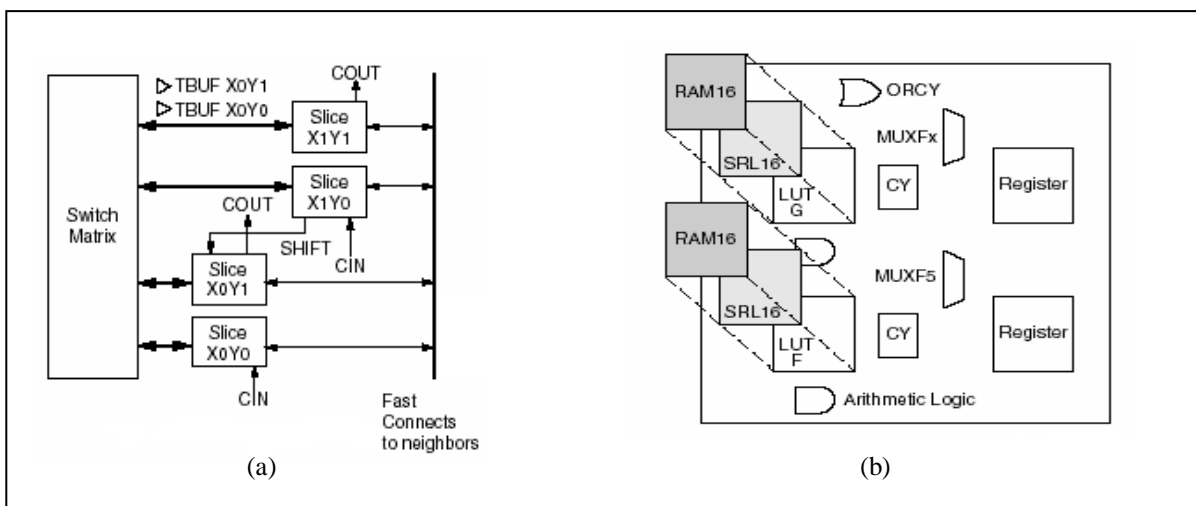


Figure III.25 Architecture interne du circuit VIRTEX-II (a) Architecture d'un CLB. (b) Architecture d'un slice

Pour l'implémentation du XOR sur la famille VIRTEX-II, plusieurs essais de synthèse ont été effectués en commençant par la famille XC2V1000, jusqu'à ce que nous atteignons le circuit XC2V8000. Toutes les familles inférieures au circuit XC2V8000 ont donné une sur-utilisation des ressources du circuit FPGA lors de la synthèse.

Le tableau III.3 présente les résultats d'implémentation du réseau (2, 2, 1) pour la famille VIRETX -XC2V8000. Les performances étudiées sont le temps d'exécution, la fréquence maximale, le nombre de multiplieurs et le pourcentage d'utilisation des CLB.

Le tableau III.3 montre que l'utilisation du parallélisme des synapses permet un temps de réponse plus rapide que celui du parallélisme des neurones (rapport de 1.6). Il en est de même pour la fréquence (rapport de 1.25). Par contre, le parallélisme des synapses occupe plus de ressources en terme d'utilisation des multiplieurs (rapport de 1.58) et de CLB (rapport de 1.3).

Ces résultats montrent que le parallélisme des synapses est très gourmand en surface, ce qui est contraignant pour des réseaux de neurones de grande taille.

Par conséquent, nous adoptons le parallélisme des neurones pour l'implémentation de l'algorithme RPG.

Tableau III.3 Comparaison entre les performances du parallélisme des neurones et le parallélisme des synapses

Performance Parallélisme	Temps (ns)	Fréquence maximale (Mhz)	Nombre de multiplieurs	% CLB
Parallélisme des neurones	60	48	12	69
Parallélisme des synapses	37.5	60	19	90
Rapport	1.6	1.25	1.58	1.3

### III.4 Proposition d'une architecture pour l'implémentation de l'algorithme RPG [167]

Les résultats obtenus dans les sections précédentes sont utilisés pour proposer une architecture permettant d'implémenter les trois phases de l'algorithme RPG. La Figure III.26, montre l'architecture générale proposée.

L'architecture de la figure III.26 est composée d'un module de « *Propagation* » qui permet d'implémenter les équations (III.1, III.2) liées à la phase de propagation, un module de « *Calcul d'erreur* » ou rétro propagation, permettant l'implémentation des équations (III.3 et III.4) liées à la phase de propagation et un module de « *Mise à jour des poids synaptiques* » permettant d'implémenter les équations (III.5 et III.6).

L'ensemble des modules « *propagation* », « *Calcul d'erreur* » et « *mise à jour des poids synaptiques* » est contrôlé par une unité de contrôle. L'ANNEXE-B présente la description VHDL et RTL de l'architecture proposée pour l'algorithme RPG et appliqué à un réseau de taille 2-2-1.

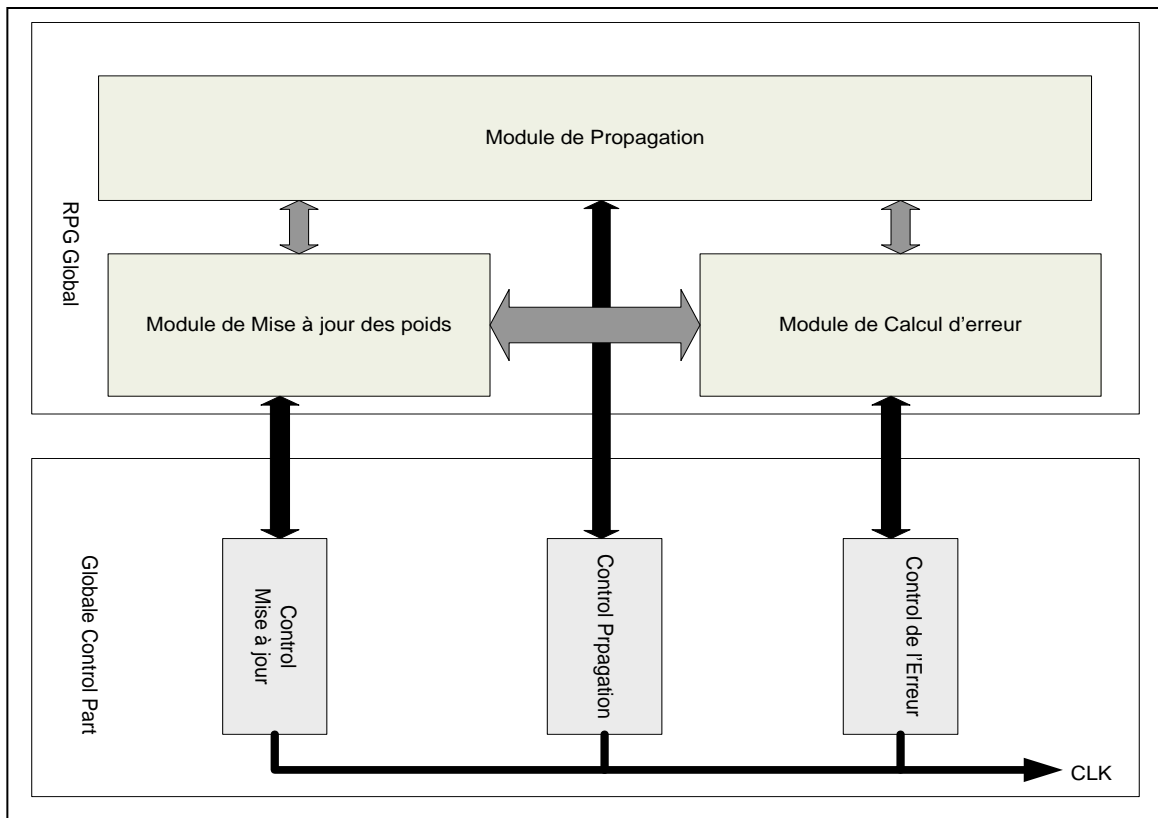


Figure III. 26 Architecture générale de l'algorithme RPG

Dans ce qui suit, l'architecture interne de chaque module sera présentée.

#### III.4.1. Module de propagation

L'architecture du module de propagation est présentée dans la figure III.28. Elle présente les caractéristiques suivantes :

- Le module de propagation est constitué de trois couches : une couche d'entrée, une couche cachée et une couche de sortie (Figure III.27 (a))
- Chaque couche est constituée par un nombre « N » de neurones (Figure III.27 (b))
- Le calcul entre les couches se fait en série
- Pour la même couche, un calcul parallèle des neurones est réalisé
- Chaque neurone implémente les équations (III.1, III.2). L'architecture de celui-ci est celle présentée dans la section III.3.2.1.1 (Figure III.27 (c))
- Le module de propagation est contrôlé par une unité de contrôle qui est similaire au contrôle du neurone, seulement on ajoute la synchronisation entre les trois couches. .

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

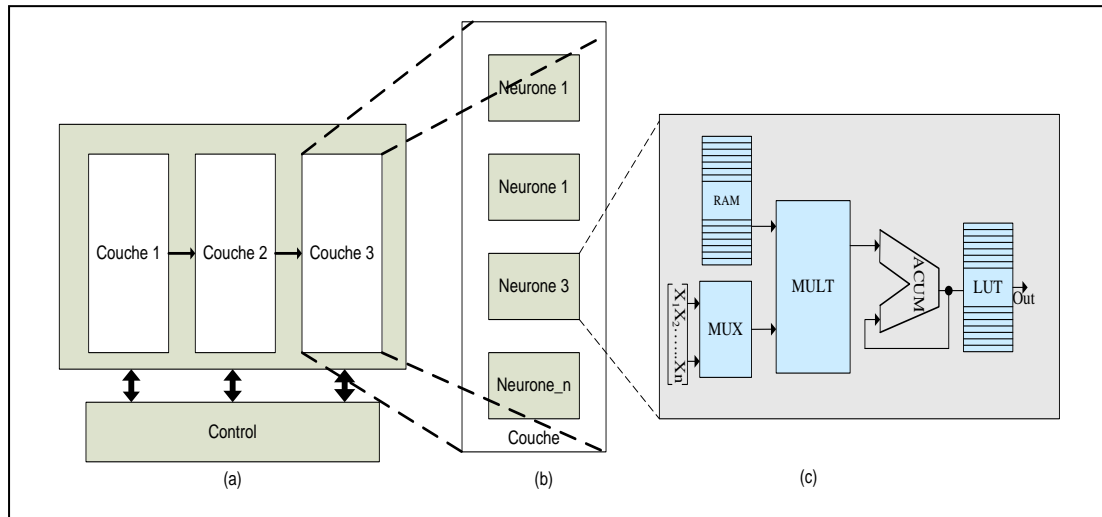


Figure III.27 (a) Module de Propagation. (b) Architecture d'une couche. (c) Architecture d'un neurone

Afin de valider le fonctionnement de l'architecture du module de propagation, nous considérons un réseau de neurones de dimension (4, 4, 4). La figure III.28, montre les résultats de la simulation fonctionnelle obtenue en utilisant l'outil de simulation ModelSIM [168]. D'après cette figure, le résultat de calcul de la sortie du réseau n'est obtenu qu'après 9 cycles d'horloges.

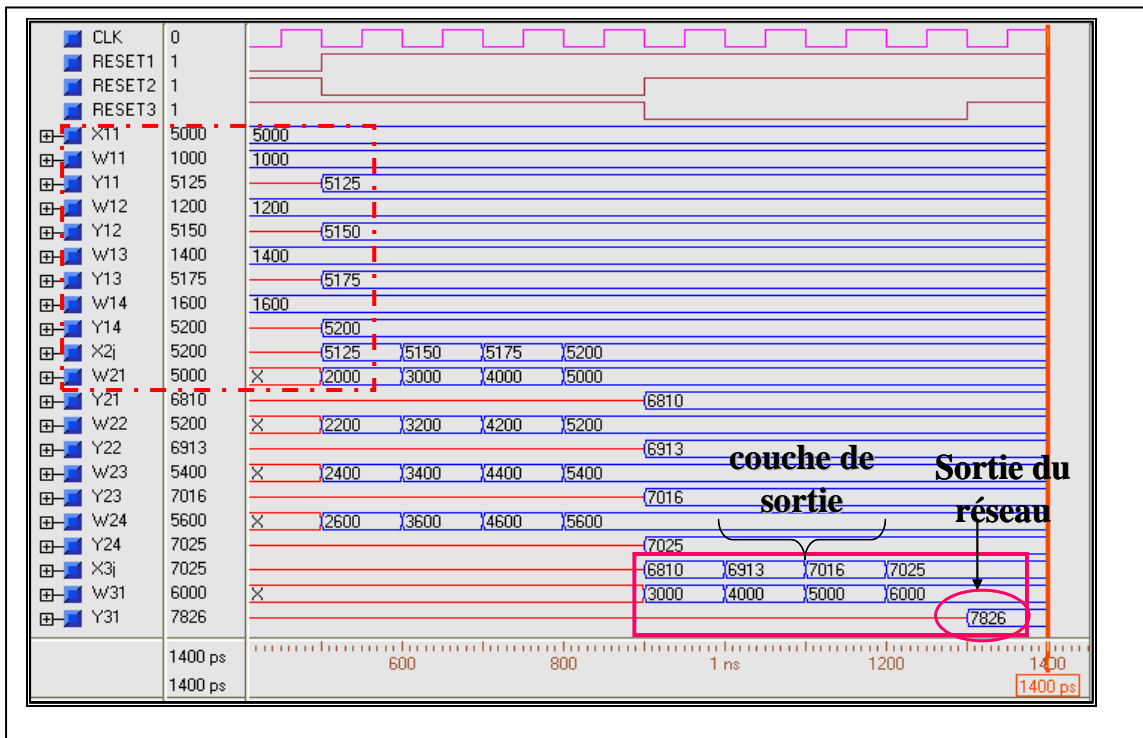


Figure III.28 Chronogramme de simulation du module de propagation

### III.4.2 Module de calcul d'erreur (rétropropagation)

Le module de rétropropagation est le second des trois modules qui sont responsables d'assigner des erreurs à chaque neurone d'entrée dans le réseau de neurone. Il est constitué



d'une unité opératrice et d'une unité de contrôle. L'unité opératrice est constituée de trois couches comme le montre la figure III.29. Ce module possède les caractéristiques architecturales suivantes :

- Le calcul de l'erreur se fait en sens inverse du module de propagation : de la couche de sortie du réseau vers la couche d'entrée
- La couche d'entrée permet le calcul de l'équation (III.3 et III.4). Elle est constituée d'un bloc permettant le calcul de la dérivée de la fonction d'activation, un bloc permettant le calcul de la différence entre la sortie désirée et la sortie calculée et un bloc permettant la multiplication de ces deux fonctions.
- La couche cachée et la couche de sortie possèdent une architecture identique permettant de calculer l'équation III.5. L'architecture de la couche cachée est constituée d'un bloc MAC permettant le calcul de la somme des produits de l'erreur locale, avec les poids synaptiques ; et d'un deuxième multiplieur, MULT2, permettant le calcul du produit de la dérivée de la fonction sigmoïde avec les sorties du MAC.

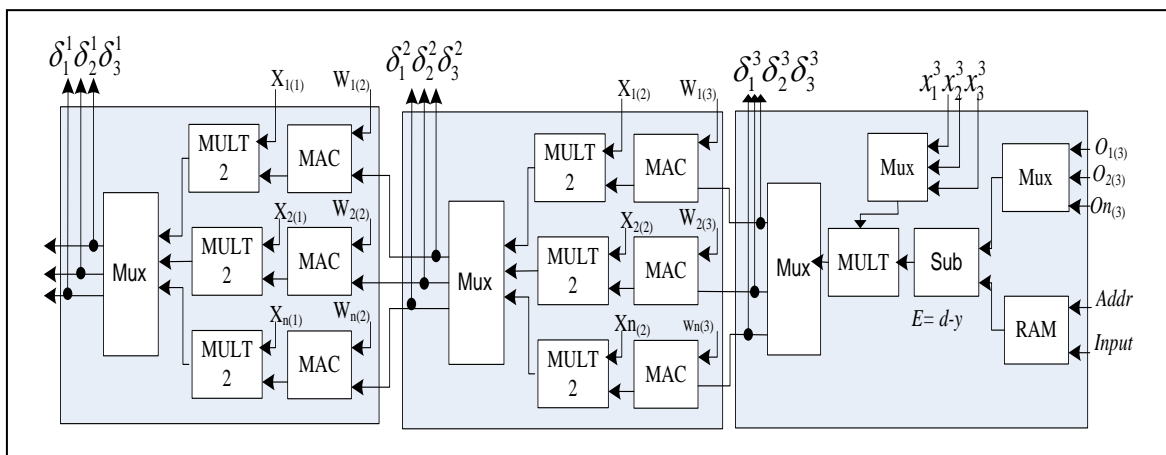


Figure III.29. Architecture du module Calcul d'erreur

La figure III.30 montre la validation fonctionnelle du module de calcul d'erreur pour le même réseau de taille (4, 4, 4).

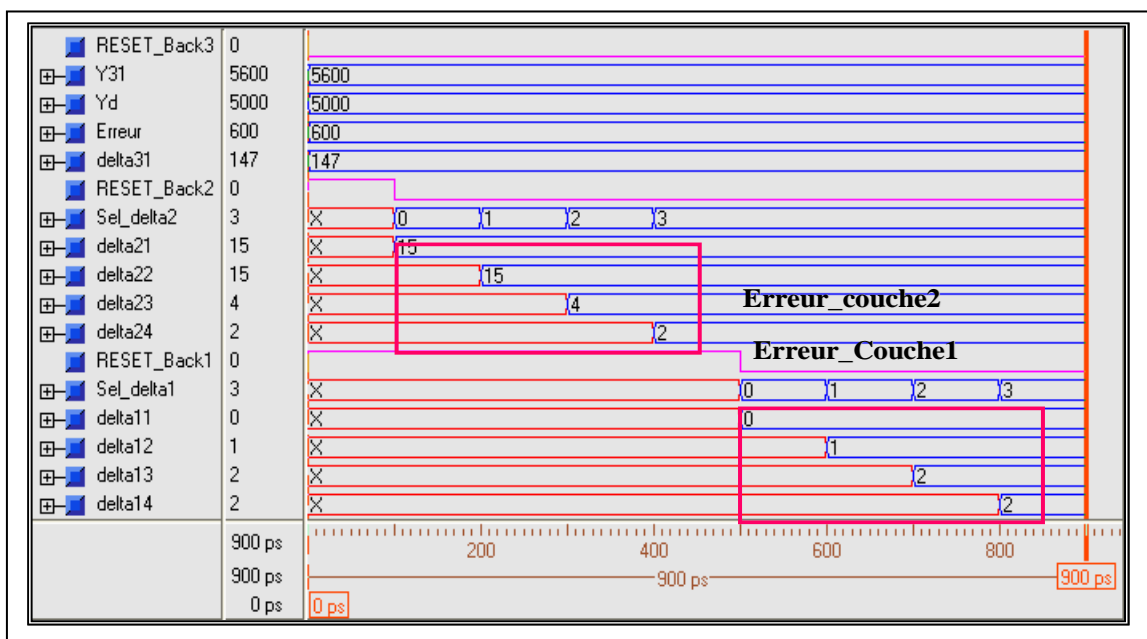


Figure III.30 Chronogramme de simulation du module de calcul de l'erreur

### III.4.3. Module de mise à jour des poids synaptiques

C'est le troisième module de l'algorithme RPG. Il est composé de trois couches identiques (Figure III.31). Chaque couche possède une architecture qui permet d'implémenter les équations III.5 et III.6. L'architecture résultante est constituée d'un :

- Bloc RAM pour le stockage du coefficient d'apprentissage ( $\eta$ ), qui va être multiplié avec les valeurs des blocs d'activation du module propagation et les valeurs des erreurs locales de la première couche du module calcul d'erreur.
- Deux blocs Multiplexeurs. Le premier permet le multiplexage des valeurs des blocs d'activation et le deuxième est chargé du multiplexage des valeurs des poids synaptiques du module propagation.
- D'un bloc « Multiplier » à trois entrées. Il est responsable de la multiplication des valeurs des sorties du bloc d'activation du module *propagation*, avec la valeur stockée dans la RAM, et les valeurs de mise à jour de la première couche du module calcul d'erreur.
- D'un bloc « Additionneur » permettant d'ajouter les sorties du multiplieur avec les valeurs qui se présentent à la sortie de la première couche de calcul d'erreur afin d'obtenir la première valeur de mise à jour d'après l'équation III.6.
- D'un bloc « Démultiplexeur » permettant le démultiplexage des poids synaptiques afin de les introduire un par un dans les différentes couches de ce module pour les itérations qui suivent.

La figure III.32 montre les résultats de simulations du module de mise à jour des poids synaptiques pour le même réseau de taille (4, 4, 4).

Après validation des trois modules de *propagation*, *calcul d'erreur* et *mise à jour des poids synaptiques*, nous étudions la partie de control de l'algorithme RPG.

### III.4.4 Module de control

Le module de control est responsable du control des trois phases de l'algorithme RPG pour assurer un bon fonctionnement de l'architecture globale de ce dernier. La figure III.33 montre l'organigramme du module de control ainsi que le diagramme d'état qui est basée sur la machine de Mealy.

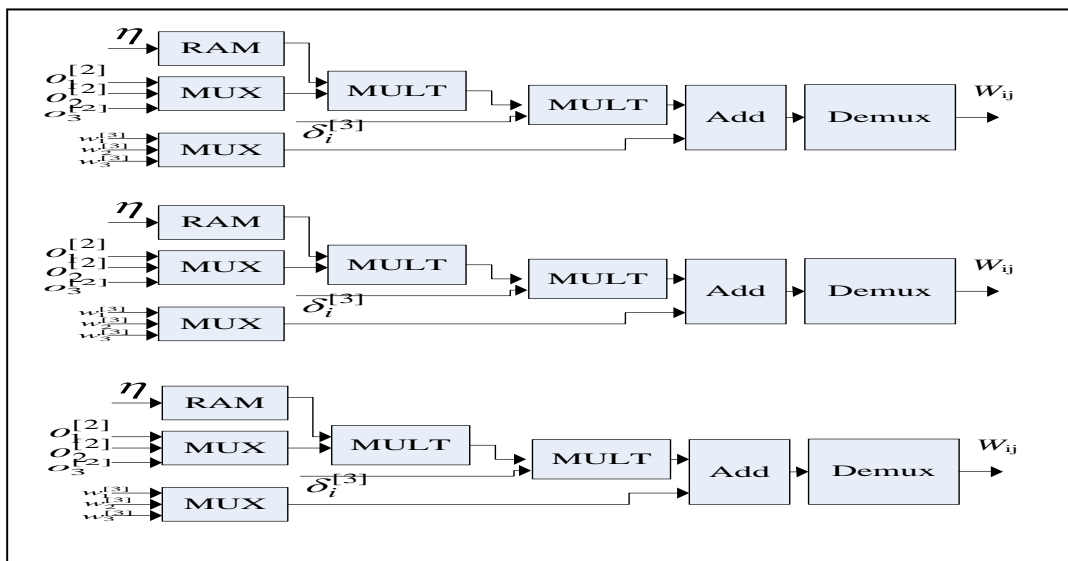


Figure III.31 Architecture du module de mise à jour des poids synaptiques

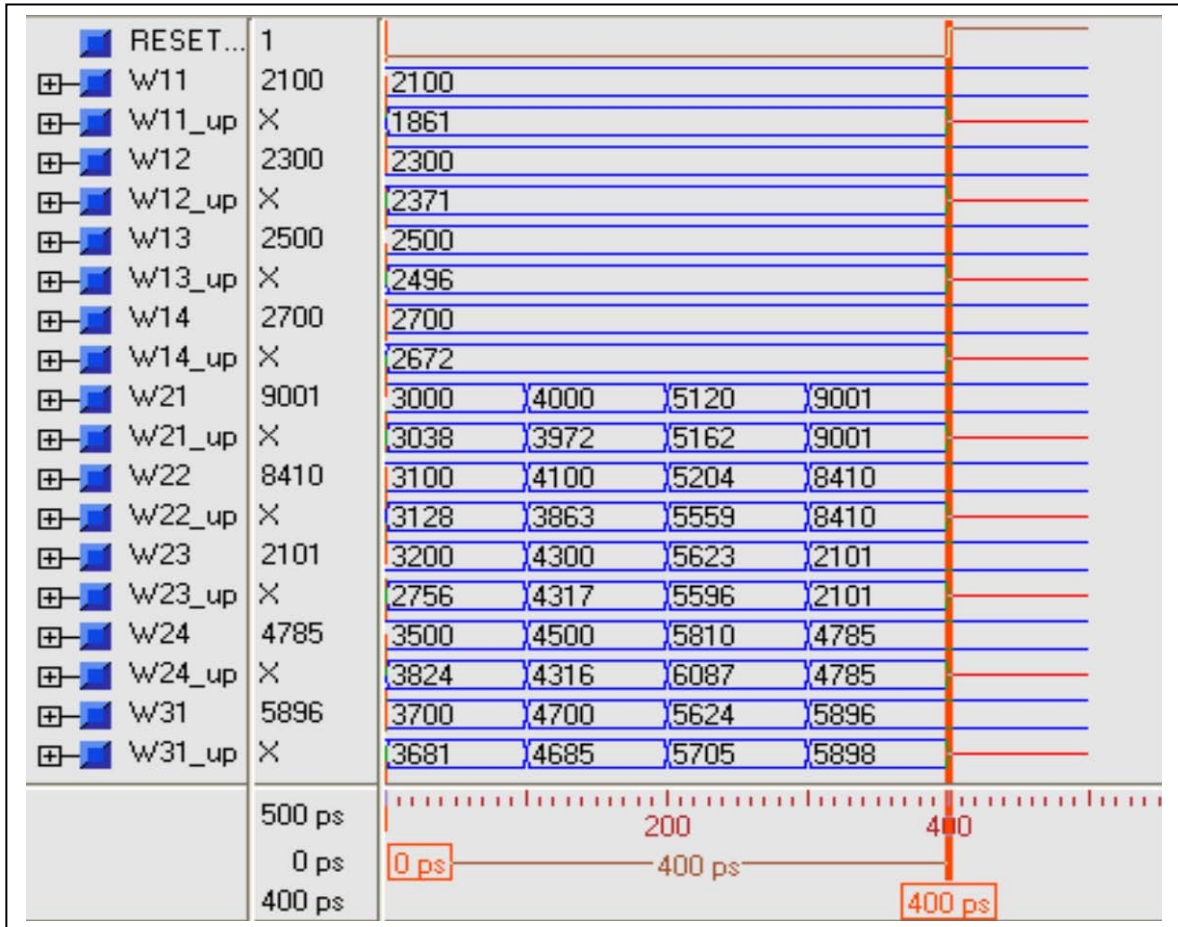


Figure III.32 Chronogramme de simulation du module de mise à jour des poids synaptiques

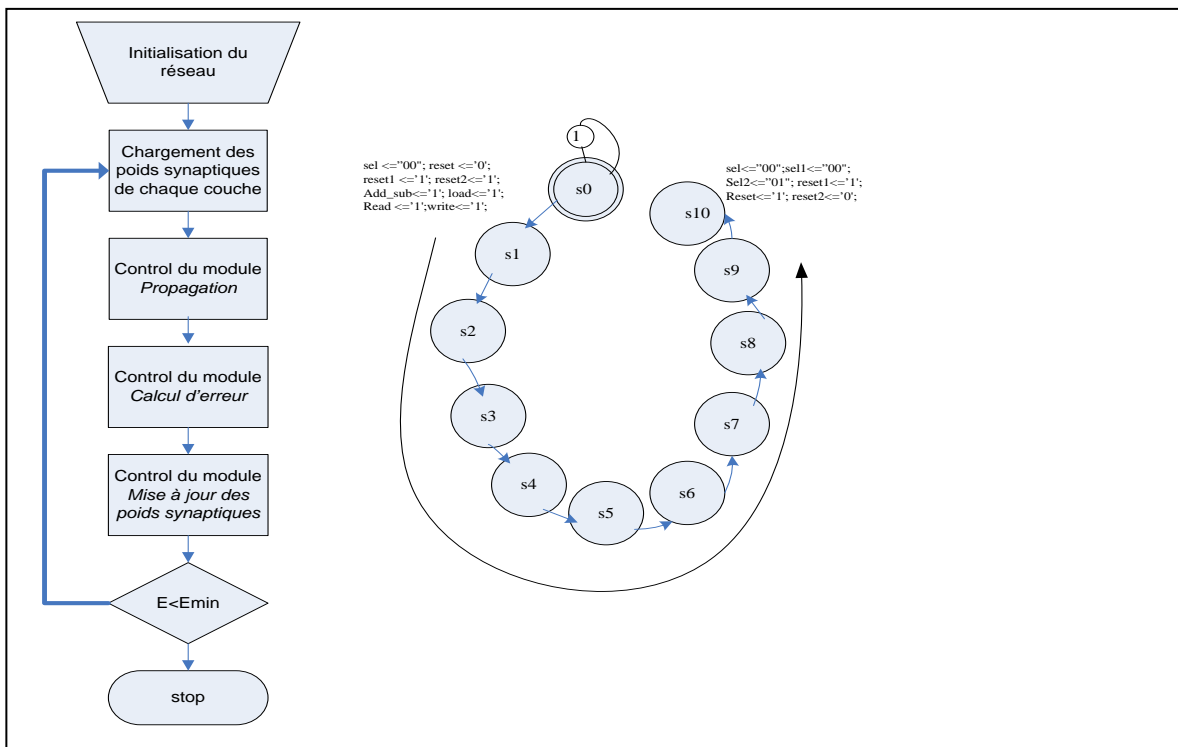


Figure III.33 Organigramme et diagramme d'état du module du control de l'algorithme RPG

### III.5 Evaluation des performances de l'architecture de l'algorithme RPG

Cette section est consacrée à l'évaluation des performances de l'architecture de l'algorithme RPG proposée. La première partie est consacrée à l'étude de l'influence du type du multiplieur sur les performances du réseau de neurone. La seconde partie est consacrée à l'influence de la taille des données sur les performances du réseau de neurone et la troisième section est consacré à l'étude des performances du réseau de neurones en utilisant les dernières familles VIRTEX des circuits FPGAs à savoir les familles Virtex-II et Virtex-4 [169]. Une comparaison des performances entre ces deux familles est établie sur trois exemples d'application : le problème du XOR, un classificateur des troubles du rythme cardiaque et un réseau complexe de grande taille.

#### III.5.1 Etude de l'influence du type de multiplieur sur le réseau de neurone

Afin de réaliser cette étude, nous avons choisi d'utiliser les multiplieurs décrits dans le tableau III.4. Chaque multiplieur est un IP-softcore. Les quatre premiers multiplieurs sont construits à partir de la bibliothèque des multiplieurs qui est donnée par le module *Coregenerator* de l'outil ISE Fondation. Les deux derniers sont obtenus du domaine public [170].

Tableau III.4 Différents type de multiplieurs

<b>Multiplieur bit- Série</b>	<b>(MBS)</b>
<b>Multiplieur Série- pipeline</b>	<b>MSP</b>
<b>Multiplieur Parallèle</b>	<b>MP</b>
<b>Multiplieur Parallèle – Pipeline</b>	<b>MPP</b>
<b>Multiplieur de Booth</b>	<b>MB</b>
<b>Multiplieur en Virgule Flottante</b>	<b>MVF</b>

Afin d'évaluer les performances du réseau de neurones, nous avons choisi d'appliquer les différents multiplieurs cités ci-dessus sur l'exemple du XOR décrit précédemment. Les performances étudiées sont la surface en %CLB, le temps de réponse ou délai (ns) et la fréquence (Mhz). L'étude a été faite sur le circuit FPGA de la famille Virtex-II. Le tableau III.5 résume les résultats de synthèse obtenus :

D'après ce tableau on constate que le *multiplieur bit-Série* offre la plus petite surface, comparé aux autres multiplieurs (surface x 10.8 comparé au multiplieur en virgule flottante).

Le multiplieur *Série- Pipeline* consomme plus de surface et met plus de temps pour répondre. Le multiplieur *Parallèle- Pipeline* offre le plus petit temps de réponse et la meilleure fréquence par rapport aux autres multiplieurs, c'est donc le composant le plus rapide.

Le multiplieur de *Booth* offre une surface inférieure aux *multiplieurs parallèles* et *Parallèles -Pipeline*, néanmoins son temps de réponse est supérieur à ces deux derniers.

Le *multiplieur en Virgule flottante* consomme la plus grande surface et le plus grand temps d'exécution. Néanmoins son utilisation peut être justifiée par le fait qu'il offre une

meilleure précision par rapport aux autres multiplieurs qui utilisent la virgule fixe au lieu de la virgule flottante.

Comparé aux différents multiplieurs, le *multiplieur parallèle* semble apporter le meilleur compromis entre la surface et le temps d'exécution.

Notons que la fréquence n'est pas directement proportionnelle au temps d'exécution.

Tableau III.5 Résultats de synthèse du XOR pour les différents types de multiplieurs

<b>Performance</b> <b>Multiplieur type</b>	<b>%CLB</b>	<b>Temps de réponse (ns)</b> <b>(1ère itération)</b>	<b>Fréquence Max (Mhz)</b>
<b>MBS</b>	<b>9</b>	<b>476.83</b>	<b>97.088</b>
<b>MSP</b>	<b>11</b>	<b>636.57</b>	<b>89.328</b>
<b>MP</b>	<b>22</b>	<b>59.51</b>	<b>48.098</b>
<b>MPP</b>	<b>24</b>	<b>53.55</b>	<b>100.017</b>
<b>MB</b>	<b>17</b>	<b>87.51</b>	<b>81.055</b>
<b>MVF</b>	<b>98</b>	<b>205.45</b>	<b>30.405</b>

### III.5.2 Influence du choix de la famille des circuits FPGAs sur les performances du réseau de neurones

Afin d'étudier l'influence de la technologie des circuits FPGA sur les performances du réseau de neurone, nous considérons trois exemples d'applications qui diffèrent dans la complexité, à savoir :

- Implémentation d'un réseau de neurones pour la classification des troubles du rythme cardiaques. Nous considérons l'implémentation « *off chip training* » de l'algorithme RPG
- Implémentation du réseau XOR Logique
- Implémentation d'un réseau de grande taille L'étude de ces trois exemples a été faite sur les circuits FPGA de la famille Virtex-II et Virtex4.

Le circuit Virtex-II étant présenté dans la section III.3.4, nous commençons d'abord par présenter la famille Virtex-4 avant de présenter les différents exemples d'applications.

La figure III.34 montre une vue générale du circuit Virtex-4. Ce dernier se présente sous forme d'une matrice qui intègre des blocs CLBs, différents types de bloc mémoire des circuits DSPs ; des entrées sorties très rapides, des circuits de control de l'horloge et jusqu'à 2 processeurs Power-PC. Les différents composants sont disposés en colonne. Les circuits Virtex-4 se présentent sous forme de plateforme. La plateforme virtex-4 LX est dédiée aux applications qui intègrent une grande logique. La plateforme SX est dédiée vers les applications de traitement du signal. La plateforme FX est orientée pour les implémentations des systèmes sur puce car elle contient jusqu'à deux processeurs Power-PC.

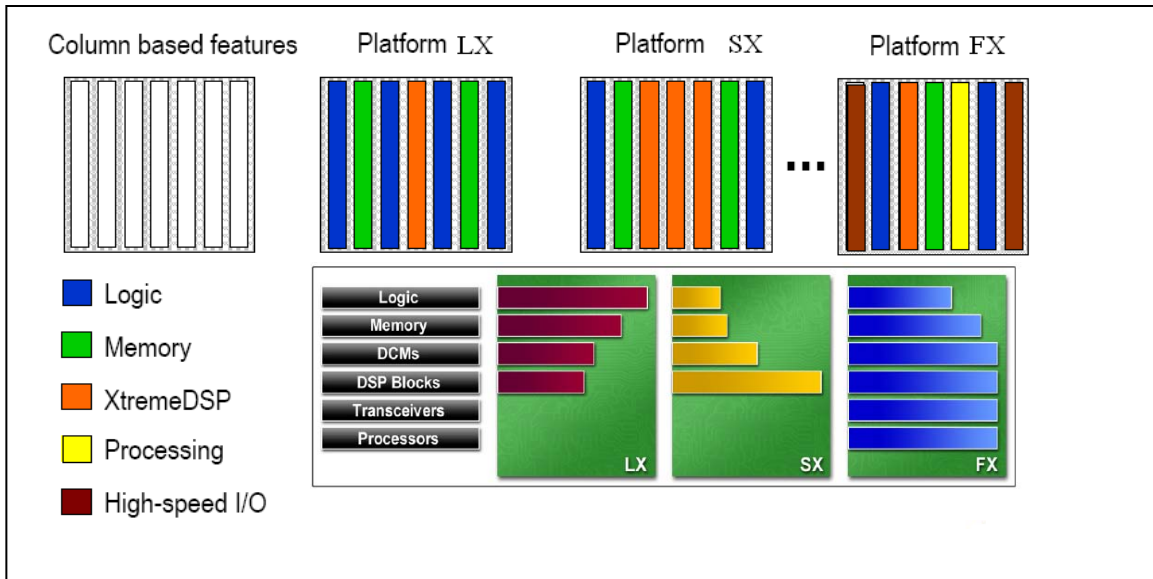


Figure III.34 Architecture générale du circuit Virtex-4

Les blocs logiques configurables (CLB) sont la source principale pour l'implémentation des circuits séquentiels et combinatoires. Chaque CLB est connecté à la matrice d'interconnexion « switch matrix ». Un élément CLB contient quatre Slices comme montré dans la figure III.35. Les slices sont regroupés en paire de deux colonnes. La colonne gauche contient les SLICEM et la colonne droite contient les SLICEL. Les éléments communs aux deux paires de slice sont deux générateurs de fonctions logiques (ou LUT), deux éléments de stockage, des multiplexeurs, une logique de retenue et des portes arithmétiques. Ces éléments sont utilisés par les deux types de slice pour réaliser des fonctions logiques, arithmétiques et des mémoires. SLICEM supporte deux autres circuits : Mémoire distribuée pour le stockage des données et un registre à décalage de 16 bits.

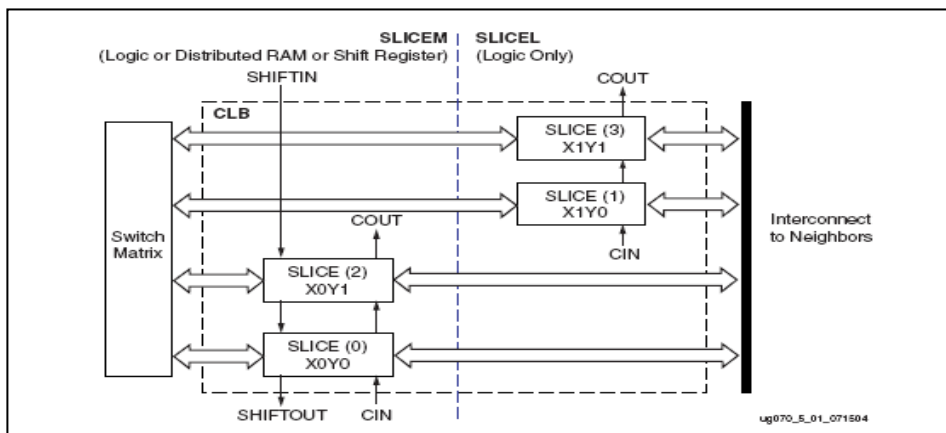


Figure III.35 Architecture d'un CLB

Une autre particularité des circuits Virtex-4 est les DSP48 slices. Les DSP48 sont disposés sur des colonnes verticales et organisés en paire de deux, formant ainsi une tuile. Un bloc DSP48 est composé d'un multiplieur à deux opérandes, complémentés à deux, sur 18 bits chacun, suivi de multiplexeurs et d'un additionneur/soustracteur à trois entrées sur 48 bits (figure III.36). L'ensemble implémente la fonction d'un multiplieur accumulateur (MAC). Il est à noter que le DSP48 n'est disponible que sur les plateformes Virtex-4SX et Virtex-4FX.

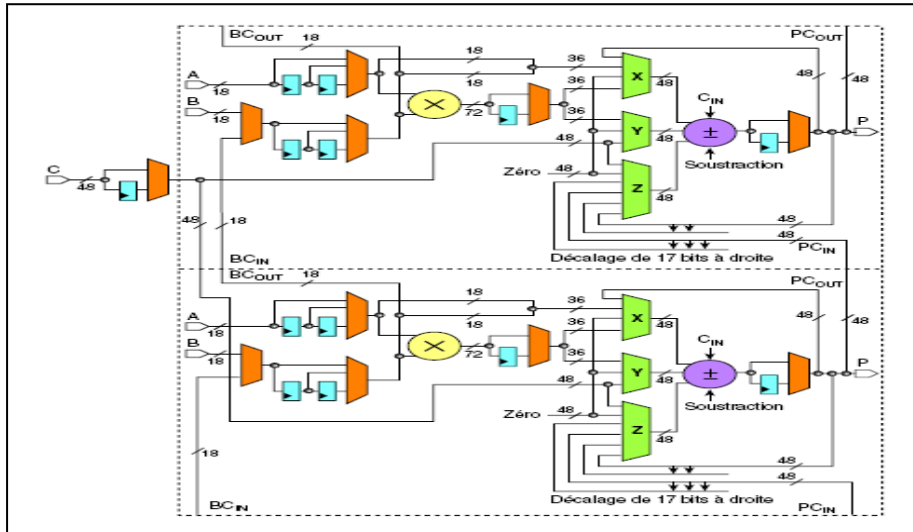


Figure III.36 Architecture d'une tuile DSP48 dans le FPGA virtex-4

### III.5.2.1 Implémentation « off chip training » d'un réseau de neurones pour la classification des troubles du rythme cardiaque

Le premier exemple étudié concerne la réalisation d'un classificateur des troubles du rythme cardiaque suivants [163], [171]: Tachycardie auriculaire (TA), Tachycardie ventriculaire (TV), la tachycardie supra-ventriculaire (TSV) et fibrillation auriculaire (FA). La Figure III.37 montre la structure générale du système proposé. Ce dernier est composé de deux classificateurs RN1 et RN2 montés en cascade. RN1 est un classificateur neuronal qui permet de classer les ondes (P) et (QRS) d'un signal ECG, en morphologie normale et anormale. Les sorties de RN1 en combinaison avec les valeurs des caractéristiques temporelles du signal ECG sont appliquées à l'entrée du deuxième classificateur RN2 dont les sorties sont les différentes arythmies proposées précédemment.

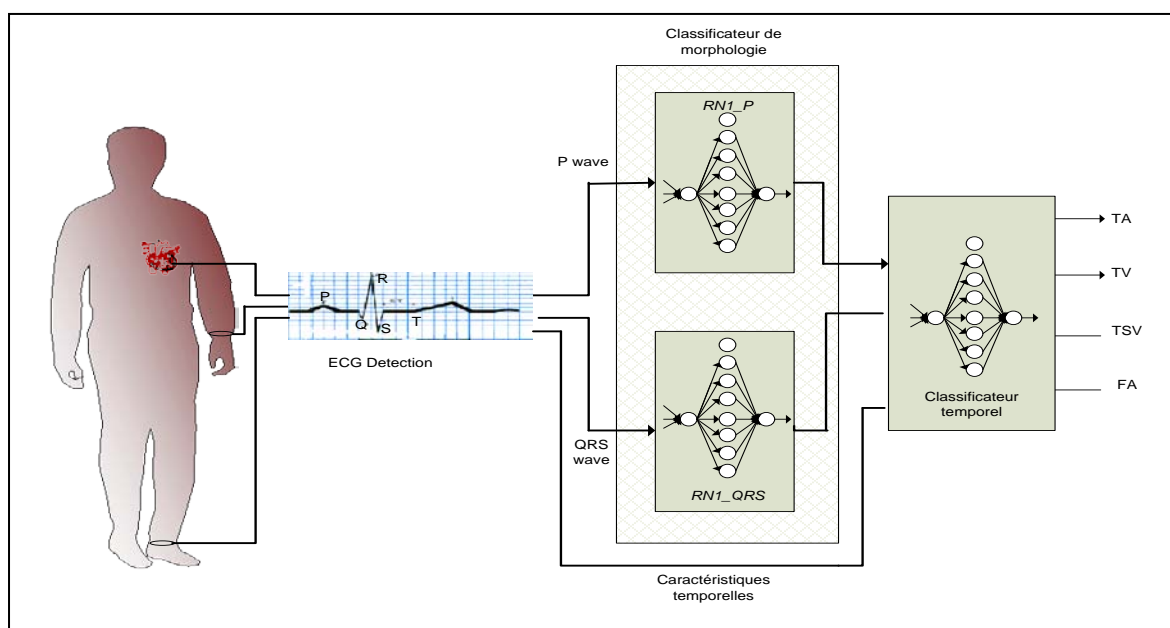


Figure III.37 Classificateur des troubles du rythme cardiaque

L'apprentissage des trois réseaux RN1\_P, RN1\_Q et RN2 a été effectué « off chip » moyennant l'outil MATLAB 6.5.

Après apprentissage la dimension du réseau ainsi que les poids synaptiques ont été fixés. Chaque sous-réseau de neurones possède une dimension de (1, 8, 1). La taille de données des poids synaptiques est de 24 bits.

Les performances étudiées sont : le pourcentage de CLBs, le temps de réponse,  $T_r$ , et le Nombre de Million de connexion par seconde (MCPS)

Le tableau III.6 montre les résultats d'implémentation du réseau (1, 8, 1) sur les familles Virtex-II et Virtex-4. Pour cette application nous avons sélectionnés les circuits XC2V1000 et XC4VLX15 pour les familles Virtex-II et Virtex-4 respectivement.

Comme nous pouvons le constater, la famille XC4VLX15 permet de meilleures performances en termes de surface (gain de 19%), de vitesse (gain de 1.6) et de MCPS (gain de 1.6)

Tableau III.6 Performances du classificateur des troubles du rythme cardiaque

FPGA	CLB %	Temps de réponse (ns)	MCPS
XC2V1000	99	44.46	360
XC4VLX15	82	26.76	597
Gain	19	1.6	1.6

### III.5.2.2 Implémentation « on chip training » du XOR logique [167], [172]

Le deuxième exemple étudié consiste à étudier les performances de l'implémentation « on chip » du réseau (2, 2, 1) présenté dans la section III.3.5, sur les familles Virtex- II et Virtex-4. La taille des données à l'entrée du réseau est de 16 bits. Les sorties sont sur 32 bits. Le tableau III.7 montre les résultats de synthèse qui sont obtenues sur les familles XC2V1000, XC2V8000, XC4VLX15 et XC4VLX80.

Tableau III.7 Performances du réseau XOR logique

FPGA	% CLB	% I/O	% MULT	% LUT	Temps de réponse (ns)	MCUPS
XC2V1000	280	92	19	78	-	-
XC2V8000	23	23	7	8	59.5	202
XC4VLX15	173	83	15	65	-	-
XC4VLX80	30	25	15	11	47.93	250
Gain	7	2	8	3	1.24	1.24



Le tableau III.7 montre que les contraintes de surface et vitesse ne pouvaient être atteintes sur les familles XC2V1000 et XC4VLX15 des circuits Virtex-II et Virtex-4 respectivement. Plusieurs essais ont été effectués jusqu'à ce qu'on atteigne les familles XC2V8000 et XC4VLX80.

Les résultats de synthèse montrent que la famille Virtex-II offre de meilleures performances en terme d'utilisation des ressources CLBs (gain de 7% en surface), d'entrées - sorties (gain de 2% en I/O), de % LUT (gai de 3%) et d'utilisation des multiplieurs (gain de 8% de MULT). Ceci est dû au fait que la Virtex-II intègre plus de multiplieurs que le circuits Virtex-4 dans lequel le composant MAC est intégré dans le DSP48 (Le circuit XC4VIX80 possède 80 MAC DSP alors que le circuit XC2V8000 possède 168 bloc multiplieurs). En termes de temps de réponse, le circuit Virtex-4 est plus rapide que le circuit Virtex-II et permet d'obtenir un meilleur MCUPS (gain de 1.24).

L'implémentation On chip requière beaucoup de multiplieurs (module de propagation, de rétropropagation et de mise à jour de poids synaptiques), c'est pour cette raison qu'il est préférable d'utiliser la Virtex-II, dans le cas où les contraintes temporelles ne sont pas critiques.

### III.5.2.3 Détermination de la dimension maximale du réseau de neurone

Dans le troisième exemple, on fixe le circuit FPGA Virtex-II et on considère trois réseaux de complexité croissante. Le premier est un réseau de neurone de dimension (2, 2, 1). Le deuxième est un réseau de dimension (4, 4, 2) et le troisième de dimension (8, 8, 4). Les entrées sorties sont codées sur 16 bits. Le tableau III.8 montre les résultats de la synthèse sur la famille XC2V8000.

Tableau III.8 Utilisation des ressources/ complexité du réseau

Taille du réseau	CLB%	%I/O	%MULT	% LUT
(2, 2, 1)	23	23	7	8
(4, 4, 2)	46	46	14	16
(8, 8, 4)	92	92	28	32
(16,16, 8)	-	-	-	-

Le tableau III.8 montre que, plus la taille du réseau de neurones augmente plus il devient difficile de l'intégrer sur un seul circuit FPGA.

Les résultats indiquent que la dimension maximale qu'on peut implémenter sur un circuit XC2V8000 est un réseau de taille (8, 8, 4) correspondant à 12 neurones.

### III.6 Conclusion

Dans ce chapitre nous avons étudiés plusieurs aspects liés à l'implémentation de l'algorithme RPG sur FPGA.

Le premier point concerne les différents aspects du parallélisme. Nous avons montré que parmi les six degrés de parallélismes cités, seul le parallélisme des neurones et le parallélisme des synapses, peuvent être pris en considération dans le cas de l'algorithme RPG qui nous intéresse.

Le deuxième point concerne le choix du langage de description matériel pour l'implémentation du parallélisme. A travers une étude comparative entre le langage VHDL et le langage Handel-C, nous avons montré qu'en terme de description du circuit, le Handel-C est nettement plus simple et plus rapide à décrire que le langage VHDL. Ceci vient du fait que le Handel-C est basé sur le langage ANSI-C auquel on a ajouté des instructions pour l'implémentation hardware, plus particulièrement l'instruction *par{ }* qui permet de réaliser une implémentation hardware parallèle.

Aussi et à travers cette étude, nous avons montré que le mapping des langages VHDL et Handel-C en hardware permet d'obtenir des implémentations ayant le même ordre de fréquence et de temps de réponse. Néanmoins, en termes de surface, une description Handel-C résulte en une surface trois fois plus grande que celle d'une description en VHDL. Ce ci nous a conduit à proposer d'adopter le langage Handel-C pour étudier les différents types de parallélisme de l'algorithme RPG et de choisir l'architecture parallèle la plus appropriée pour une implémentation sur FPGA.

L'étude comparative entre le parallélisme des neurones et celui des synapses montre qu'il est plus judicieux d'utiliser le parallélisme des neurones, car il occupe moins de ressources du circuit FPGA. Cette étude nous a permis de se fixer sur l'architecture générale de l'algorithme RPG qui est basée sur le parallélisme des neurones.

Une fois l'architecture choisie, nous avons utilisé le langage VHDL pour sa description et son implémentation hardware. C'est l'objet du troisième point étudié et dans lequel nous avons proposé une architecture permettant d'implémenter les trois phases de l'algorithme RPG à savoir la phase de propagation, la phase de rétropropagation ou calcul d'erreur et la phase de mise à jour des poids synaptiques. Nous avons proposé une architecture simple, régulière, parallèle et pouvant prendre en considération les deux type d'implémentation « on chip training » et « off chip training ».

Le quatrième point consiste à évaluer les performances de l'architecture proposée en fonction du type du multiplieur. Nous avons montré que le multiplieur parallèle de *Coregen* offre le meilleur compromis entre la surface et le temps d'exécution comparé aux autres multiplieurs (multiplieur série, multiplieur série pipeline, multiplieur de Booth, multiplieur parallèle pipeline et le multiplieur en virgule flottante).

Le cinquième point consiste à étudier l'influence du choix de la famille des circuits FPGAs sur les performances du réseau de neurones. Sur ce sujet, nous avons aboutit aux conclusions suivantes :

Lors de l'implémentation off chip training, le circuit virtex-4 offre de meilleures performances en termes de surface et temps de réponse.

Lors de l'implémentation on chip, le circuit Virtex-II offre les meilleures performances en terme de surface alors que le circuit virtex-4 permet un temps de réponse et des performances de connexion du réseau, MCUPS, meilleures. Cette différence entre l'implémentation on chip et l'implémentation off chip trouve son explication dans le fait que les modules de rétropropagation et de mise à jour des poids synaptiques effectuent plus de calculs ; par conséquent ils requièrent plus d'opérations arithmétiques et logiques, plus particulièrement les multiplieurs. Le circuit Virtex-II contient un nombre de multiplieurs supérieur à celui du circuit VIRTEX-4. C'est pour cette raison que nous recommandons d'utiliser la Virtex-II, dans le cas où les contraintes temporelles ne sont pas critiques.

Dans le dernier point nous avons montré que plus la dimension du réseau augmente, plus il devient difficile d'implémenter tout l'algorithme RPG. La dimension maximale pour la famille XC2V800 est un réseau de (8, 8, 4) correspondant à 12 neurones.

Afin d'augmenter la dimension du réseau de neurones à implémenter, nous proposons d'utiliser la reprogrammabilité dynamique des circuits FPGA ou « *Run-Time Reconfiguration* (RTR) ». Cette dernière fera l'objet du prochain chapitre.

---

# *CHAPITRE IV*

---

**UTILISATION DE LA RECONFIGURATION DYNAMIQUE POUR  
L'IMPLEMENTATION DE L'ALGORITHME RPG**

## IV.1 Introduction

La reconfiguration dynamique est un concept apparu avec l'avènement des circuits FPGAs. Aujourd'hui, presque trente ans après, des évolutions importantes ont eu lieu en ce qui concerne l'idée même de reconfigurer dynamiquement un composant.

Afin de tirer profit des dernières améliorations technologiques des circuits FPGAs, nous proposons, dans ce chapitre, d'exploiter la reconfiguration dynamique des circuits FPGAs pour améliorer la densité d'intégration des réseaux de neurones. Nous nous intéressons à l'exploitation de la reconfiguration dynamique des circuits FPGA de la famille Virtex-II.

La section IV.2 est consacrée à la définition des concepts de base liés à la configuration et reconfiguration des circuits FPGAs, ainsi que les mesures de performances de cette dernière. Dans la section IV.3, nous présentons trois différentes approches de la reprogrammabilité dynamique appliquées à l'algorithme RPG. Les différents résultats obtenus seront discutés dans la section IV.4 et comparés à ceux de la littérature. Nous terminons par une conclusion.

## IV.2 Concepts de base [173]

### IV.2.1 La configuration statique des FPGA

La reconfiguration statique des FPGAs est utilisée lorsque le composant ne réalise qu'une fonction entre sa mise sous tension et son arrêt. Dans ce cas, du fait que la fonction du composant est déterminée par le contenu d'une SRAM, qui est une mémoire volatile, il est nécessaire de mettre à jour le contenu de la mémoire de configuration à chaque mise sous tension. Cette phase est la reconfiguration du FPGA. Elle consiste à transférer le contenu d'une mémoire non volatile de type FLASH ou EPROM, dans la mémoire de configuration du circuit FPGA.

Une autre alternative, est de reconfigurer le composant par l'intermédiaire d'un processeur. Dans ce cas, le contenu de la mémoire de configuration est stocké soit dans une mémoire non volatile du type de celle déjà évoquée, soit dans un disque dur. C'est le processeur qui écrit les données dans la mémoire de configuration du FPGA. Une fois la reconfiguration terminée, le FPGA réalise les calculs jusqu'à l'arrêt du composant comme le montre la figure IV.1

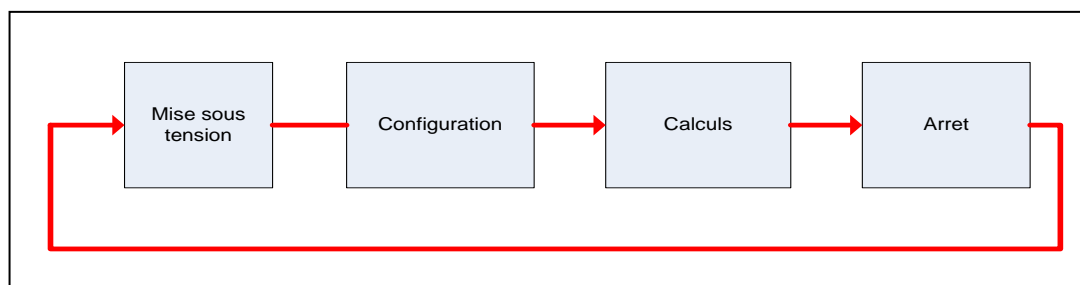


Figure IV.1 Configuration statique d'un FPGA

Dans ce cas de figure, on ne tire pas vraiment profit de la SRAM de configuration du moment que le hardware reste figé. L'avantage d'une configuration statique est sa simplicité d'utilisation et elle est supportée par tous les outils de conception FPGAs.

### IV.2.2 La reconfiguration dynamique des FPGA (RTR)

Pour tirer pleinement profit de cette souplesse, on peut exploiter tous les avantages liés à l'utilisation d'une SRAM pour stocker les configurations. En effet, rien n'empêche pratiquement de modifier le contenu de la SRAM entre la mise sous tension du composant et son arrêt. La fonction réalisée par le composant peut donc évoluer en cours de fonctionnement. On parle alors de reconfiguration dynamique. Souvent le terme « Run time reconfiguration » (RTR) est utilisé dans la littérature pour désigner la reconfiguration dynamique. Dans ce cas de figure, la configuration principale est divisée en des sous configurations, chacune implémentant une fraction de l'application.

Alors que dans la configuration statique le FPGA est configuré une seule fois avant l'exécution, la RTR reconfigure le FPGA plusieurs fois durant le fonctionnement normal de l'application comme le montre la figure IV.2.

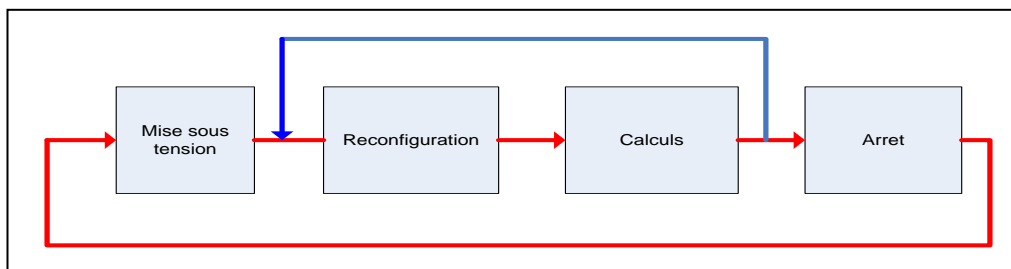


Figure IV.2 Reconfiguration dynamique d'un FPGA

La nature dynamique du hardware introduit deux nouveaux concepts. Le premier consiste à réaliser un partitionnement temporel de l'algorithme utilisé, et dans lequel plusieurs parties ou modules du circuit travaillent de façon concurrente. Ce type de partitionnement est très difficile à réaliser et à notre connaissance, à l'heure actuelle il n'existe pas d'outils qui supportent ce genre de partitionnement.

Le deuxième type consiste à réaliser un partitionnement structurel de l'algorithme. Il s'agit de diviser l'application en des modules qui travaillent séquentiellement dans le temps. Quoique des outils commencent à émerger pour réaliser ce genre de partitionnement, la communication entre les différents modules doit être réalisée minutieusement pour garantir le bon fonctionnement de l'application.

Dans notre cas, nous considérons le partitionnement structurel de la RTR.

Deux approches peuvent être utilisées pour implémenter la RTR : la RTR globale et la RTR locale.

Les deux techniques utilisent plusieurs configurations pour une seule application et les deux reconfigurent le circuit FPGAs durant l'exécution de l'application. La différence principale entre ces deux techniques réside dans la manière avec laquelle le hardware dynamique est alloué. Ces deux techniques ainsi que leur impact seront discutés dans les sections suivantes.

#### IV.2.2.1 La RTR globale

La RTR globale consiste à modifier toute la mémoire de configuration du FPGA durant chaque étape de configuration. En d'autres termes, l'application est divisée en plusieurs phases et chaque phase est implémentée dans tout le FPGA. La figure IV.3 montre un exemple d'une application constituée de trois phases : A, B et C. L'application s'exécute en reconfigurant séquentiellement les trois configurations A, B et C.

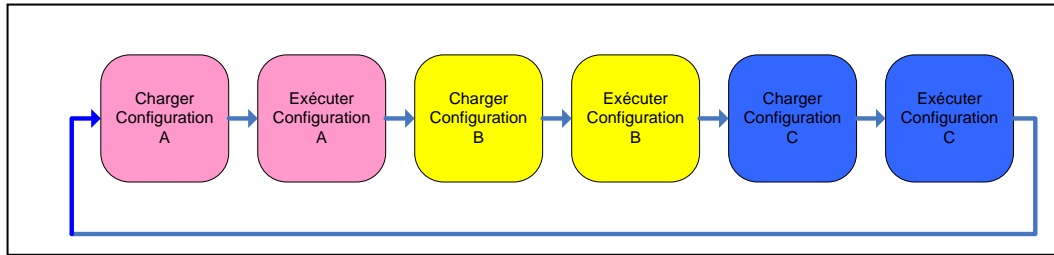


Figure IV.3 La RTR globale

Le concepteur ayant à réaliser la RTR globale doit d'abord réaliser un partitionnement initial et par la suite renouveler la procédure pour un deuxième partitionnement du design jusqu'à exécution de toutes les partitions. Durant le fonctionnement, la configuration courante doit terminer ses calculs en plaçant ses résultats dans des mémoires ou bien des registres. La configuration qui suit, commence par lire ces résultats intermédiaires et calcule le résultat qui suit. Ce processus « configurer et exécuter » se répète jusqu'à la fin de l'application.

L'un des principaux avantages de la RTR globale est sa simplicité car les partitions utilisées sont des partitions à gros grains. Les outils conventionnels de conception peuvent être utilisés une fois que le partitionnement manuel a été réalisé. Chaque partition peut être simulée, synthétisée et implémentée comme un module indépendant des autres.

#### IV.2.2.2 La RTR locale

Comme son nom l'indique, la RTR locale considère la reconfiguration d'un sous ensemble de la logique pendant l'exécution de l'application, c.-à-d. une partie du circuit correspondant à certaines fonctions logiques et leur interconnexions peut être changée sans affecter le fonctionnement de la logique restante. La RTR locale offre une meilleure flexibilité et une granularité plus fine que la RTR globale.

Cette approche permet non seulement de diminuer le temps de reconfiguration, du fait que moins de données sont mises à jour, mais aussi d'envisager que le traitement réalisé par les ressources qui ne sont pas reconfigurées ne soit pas interrompu. On parle dans ce cas de reconfiguration partielle à chaud. La figure IV.4 montre un exemple d'application de la RTR locale. Dans la première étape le bloc **A** et le bloc **B** sont chargés. Dans la deuxième étape le bloc **A** continue son fonctionnement alors que le bloc **B** qui est inactive, est remplacé par les blocs **C** et **D**. Entre chaque étape d'exécution, seulement un sous ensemble du hardware est reconfiguré.

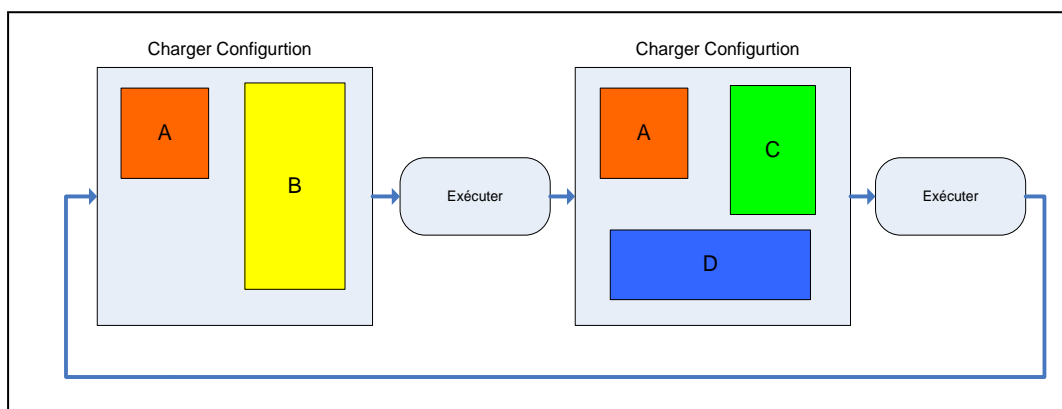


Figure IV.4 La RTR locale

L'avantage de la RTR locale est qu'elle offre une grande flexibilité par rapport à la RTR globale néanmoins le concepteur de ce genre de reconfiguration se trouve confronté à la complexité dans la gestion de la communication entre les différents blocs à reconfigurer. Les outils de conception traditionnels ne sont pas suffisants pour réaliser la RTR locale.

### IV.2.3 Mesures de performances

Afin de pouvoir comparer entre les performances d'une architecture utilisant la configuration statique et celle utilisant la configuration dynamique, Wirthlin's [174] a introduit et définit une nouvelle mesure appelée densité fonctionnelle, ( $D$ ), comme suit :

$$D = \frac{\text{Performance}}{\text{Coût}} = \frac{1/(\text{Temps d'exécution})}{(\text{Surface du circuit})} \quad (\text{IV.1})$$

La performance est un Benchmark qui se mesure par l'inverse du temps d'exécution d'une application et reflète la vitesse de traitement ou latence d'un système [175]. Plus le temps d'exécution est petit, plus le système est performant. La plupart du temps, la performance est obtenue à un prix donné, exprimé par la surface du circuit.

Dans le cas de la configuration statique, la densité fonctionnelle s'exprime comme suit :

$$D_s = \frac{1}{A_s \times T_e} \quad (\text{IV.2})$$

Où

$A_s$ , représente la surface totale du circuit

$T_e$ , représente le temps d'exécution

Dans le cas de la configuration dynamique la densité fonctionnelle s'exprime par :

$$D_{RTR} = \frac{1}{A_{RTR} \times (T_e + T_c)} \quad (\text{IV.3})$$

, Où

$A_{RTR}$ , représente la surface de la partie reconfigurée

$T_e$ , représente le temps d'exécution de la partie reconfigurée

$T_c$ , représente le temps de reconfiguration

Le temps d'exécution dépend de l'application. Par contre le temps de reconfiguration dépend du composant FPGA utilisé.

Si on définit le rapport de configuration,  $f$ , par :

$$f = \frac{T_c}{T_e} \quad (\text{IV.4})$$

Alors le temps total d'opération peut s'exprimer par :

$$T = T_e + T_c = T_e(1 + f) \quad (\text{IV.5})$$

En substituant l'équation (IV.5) dans l'équation (IV.3), la densité fonctionnelle devient :

$$D_{RTR} = \frac{1}{A_{RTR} T_e (1 + f)} \quad (\text{IV.6})$$



Comme le montre l'équation (IV.6), un temps de reconfiguration élevé peut être toléré si le temps d'exécution est lui aussi très élevé. L'équation (IV.6) montre que la RTR est bénéfique pour des systèmes ayant une configuration à gros grain et dont le traitement des données est très grand.

Si  $T_e \gg T_c$  (i.e.  $f \rightarrow 0$ ), la densité fonctionnelle atteint une valeur maximale définie par :

$$D_{\max} = \lim_{f \rightarrow 0} D_r = \frac{1}{A_{RTR} T_e} \quad (\text{IV.7})$$

En substituant l'équation (IV.7) dans l'équation (IV.6), la densité fonctionnelle s'écrit

$$D_{RTR} = \frac{D_{\max}}{(1+f)} \quad (\text{IV.8})$$

L'équation (IV.8) ci-dessus, montre que la densité fonctionnelle croît à une valeur maximale  $D_{\max}$  et décroît lorsque le rapport de configuration,  $f$ , croît.

Afin de pouvoir comparer entre la reconfiguration dynamique et la reconfiguration statique, on définit le taux d'amélioration maximum:

$$I_{\max} = \frac{D_{\max}}{D_s} - 1 \quad (\text{IV.9})$$

Les conditions pour lesquels la RTR améliore la densité fonctionnelle sont données par :

$$\begin{aligned} D_{RTR} &\geq D_s \\ \frac{1}{A_{RTR}(T_e + T_c)} &\geq D_s \\ \frac{1}{D_s} \left( \frac{1}{A_{RTR} \times T_e} \right) &\geq 1 + \frac{T_c}{T_e} \end{aligned} \quad (\text{IV.10})$$

Ce qui revient à écrire :

$$\begin{aligned} \frac{D_{\max}}{D_s} - 1 &\geq f \\ I_{\max} &\geq f \end{aligned} \quad (\text{IV.11})$$

L'équation (IV.11) donne le rapport maximum de reconfiguration au dessus duquel la reconfiguration dynamique n'est pas justifiée.

### IV.3 Application de la reconfiguration dynamique à l'algorithme RPG

Dans cette section les différentes approches de la reconfiguration dynamique (RTR) seront appliquées à l'algorithme RPG et comparées avec la configuration statique.

Rappelons que l'algorithme RPG est composé de trois phases consécutives à savoir : la phase de *propagation*, la phase de *retropropagation* ou calcul d'erreur et la *phase de mise à jour des poids synaptiques* comme le montre la figure IV.5

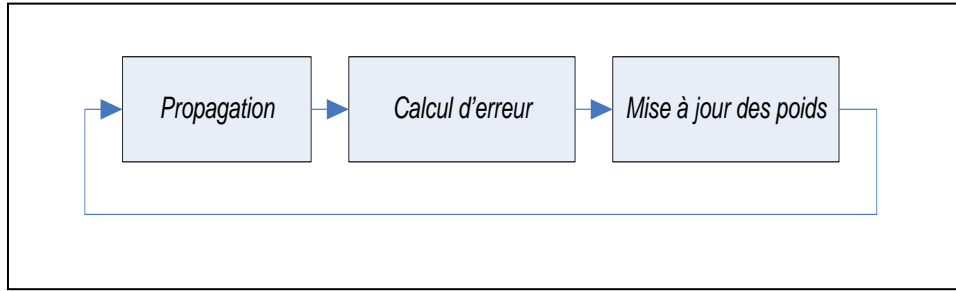


Figure IV.5 Différentes étapes de l'algorithme RPG

Les avantages et les limites de l'application de la RTR peuvent être analysés en mesurant la densité fonctionnelle de la configuration statique et de la configuration dynamique.

Dans le cas d'un réseau de neurones, la densité fonctionnelle peut être exprimée en fonction du nombre de mise à jour par seconde des poids synaptiques (WUPS), défini par :

$$P(WUPS) = \frac{C(n)}{T(n)} \quad (IV.12)$$

La densité fonctionnelle s'écrit :

$$D = \frac{C(n)}{A(n) \times (T(n) + T_c)} \quad (IV.13)$$

Où

$C(n)$  représente le nombre de connections

$A(n)$  la surface du réseau de neurones en question

$T(n)$  le temps d'exécution

$T_c$  le temps de configuration

$n$ , le nombre de neurones

### IV.3.1 Estimation du nombre de connections du réseau de neurones

Le nombre de connections  $C(n)$  d'un réseau de neurones est déterminé par le nombre de couches ( $l$ ) et le nombre de nœuds de chaque couche comme suit :

$$C(n) = \sum_{x=1}^{l-1} \text{noeud}(x) \times \text{noeud}(x+1) \quad (IV.14)$$

Nous considérons un réseau de neurones de trois couches ( $l=3$ ), la première couche contient  $n_1$  neurones, la deuxième couche contient  $n_2$  neurones et la troisième couche  $n_3$  neurones.

Sachant que le nœud représente l'entrée d'un neurone, et en utilisant l'équation (IV.14), le nombre de connections disponibles dans un réseau de trois couche s'écrit :

$$C(n) = n_1 \times n_2 + n_2 \times n_3 \quad (IV.15)$$

### IV.3.2 Estimation du temps d'exécution

Le temps d'exécution  $T(n)$ , représente le temps requis pour compléter une itération de l'algorithme RPG. Il est obtenu en multipliant le nombre de cycles d'horloge par la fréquence de l'horloge. Pour l'algorithme RPG, il est défini comme suit:

$$T(n) = T(\text{Pr opagation}) + T(\text{Calcul d' erreur}) + T(\text{Mise à jour des poids}) \quad (IV.16)$$

Où

$T$  (*Propagation*) est le temps d'exécution du module de propagation

$T$  (*Calcul d'erreur*) est le temps d'exécution du module de calcul d'erreur

$T$  (*Mise à jour des poids*) est le temps d'exécution du module de mise à jour des poids synaptiques

Le nombre de cycles d'horloge est déterminé en fonction du nombre de neurones et du nombre de couches de l'architecture utilisée.

Pour notre cas, nous avons repris l'architecture de l'algorithme RPG proposée dans la section III.4 du chapitre 3 et les résultats de simulation de chaque module.

Pour le module de *Propagation*, le temps d'exécution est estimé comme suit :

$$T(\text{Pr opagation}) = T_{clk} \left( \frac{3}{2} + n_1 + n_2 + n_3 \right) \quad (\text{IV.17})$$

Pour le module de *Calcul d'erreur*, le temps d'exécution est estimé comme suit :

$$T(\text{Calcul d'erreur}) = T_{clk} \left( \frac{1}{2} + n_3 + n_2 + n_1 \right) \quad (\text{IV.18})$$

Pour le module de *Mise à jour des poids synaptiques*, le temps d'exécution est estimé comme suit :

$$T(\text{Mise à jour des poids}) = T_{clk} (1 + n_3 + n_2 + n_1) \quad (\text{IV.19})$$

Où  $T_{clk}$  représente le temps d'un cycle d'horloge (l'inverse de la fréquence d'horloge). Il est estimé par l'outil de synthèse.

### IV.3.3 Estimation de la surface du réseau : état des lieux et proposition

Un point important qu'il faut prendre en considération est l'estimation de la surface du réseau de neurones. Pour déterminer la surface du réseau  $A(n)$  en fonction du nombre de neurones, nous considérons que le plus petit élément qu'on peut implémenter sur un circuit FPGA est le neurone. Par conséquent, la surface  $A(n)$  de n'importe quel réseau de neurones peut s'exprimer comme un multiple de la surface du neurone, comme suit :

$$A(n) = n \times a_{neurone} \quad (\text{IV.20})$$

Où  $a_{neurone}$  représente la surface du neurone ;  $n$  étant le nombre de neurones.

En remplaçant les expressions de  $C(n)$ ,  $T(n)$  et  $A(n)$  dans l'équation (IV.13), la densité fonctionnelle s'écrit :

$$D = \frac{n_1 \times n_2 + n_2 \times n_3}{(3T_{clk} + 3n_1T_{clk} + 3n_2T_{clk} + 3n_3T_{clk} + T_c)(n \times a_{neurone})}$$

Si on prend  $n_1=n_2=n_3=n$ , la densité s'écrit :

$$D = \frac{2n}{a_{neurone} \times (3T_{clk} (3n + 1) + T_c)} \quad (\text{IV.21})$$

Comment estimer la surface du neurone ?

D'abord, il faut savoir que l'outil de synthèse fournit les résultats du mapping de l'algorithme en termes de bloc logiques configurables (CLB), de mémoires LUT, de multiplieurs, d'entrée sortie, etc.

Pour estimer la surface, dans [94] l'auteur utilise le nombre de CLB comme métrique commun aux circuits FPGAs. Cependant le nombre de CLBs ne peut pas donner à lui seul une estimation proche de la réalité surtout dans les dernières familles de circuits FPGAs qui, en plus des CLBs, intègrent des mémoires, des multiplieurs, des circuits DSPs et des processeurs.

Dans [93], l'auteur utilise une autre métrique représentée par la densité des neurones (nombre de neurones/FPGA). Cependant, cette métrique engendre une erreur de quantification car ce ne sont pas toutes les ressources du circuit FPGA qui sont utilisées dans une implémentation.

Pour remédier à ce problème nous proposons d'utiliser le nombre équivalent de portes logiques au lieu du nombre de CLBs car il exprime l'équivalent des CLBs, des LUT, des multiplieurs, etc. en terme de portes logique.

Cependant, l'estimation du nombre équivalent de portes logiques n'est pas facile à faire; car elle nécessite la connaissance approfondie de l'équivalence logique de chaque ressource du circuit FPGA.

Dans la référence [176] de Xilinx, une approche est proposée pour l'estimation du nombre équivalent de portes pour les CLBs, et les LUT dans les circuits FPGA de la famille XC4000. Ce qui est insuffisant pour les circuits de la famille Virtex car ils intègrent des multiplieurs, des mémoires en plus des CLB et des LUTs.

Afin de remédier à ce problème, nous avons utilisé les résultats du mapping de l'algorithme obtenus dans les rapports fournis par l'outil ISE Fondation, après implémentation physique (placement et routage) de l'architecture.

Partant de ce principe, nous définissons une nouvelle métrique exprimée par le nombre équivalent de portes logiques/neurone comme référence pour estimer la surface minimale  $a_{neurone}$ . Par conséquent, la densité fonctionnelle sera exprimée en WUPS/Gate « Weight Update Par Seconde par Gate ».

Le tableau IV.1 Donne les ressources utilisés durant la synthèse et ceux obtenus après implémentation du design d'un réseau de neurones (3, 3, 3) correspondant à 9 neurones.

Tableau IV.1 Les ressources utilisées du circuit FPGA Virtex-II.

Module	Nombre de CLB	Nombre de LUT	Nombre de MULT	Nombre équivalent de portes logiques	Nombre de portes/neurone	T <sub>clk</sub> (ns)
<b>RPG complet</b>	31426	21718	28	485942	53994	75.92
<b>Propagation</b>	15200	10800	9	252127	28014	75.92
<b>Calcul d'erreur</b>	7800	6700	13	132548	14727	75.92
<b>Mise à jour des poids</b>	8326	4013	6	99657	11073	75.92
<b>Control global</b>	100	205	0	1610	-	75.92

#### IV.3.4 Configuration statique de l'algorithme RPG

La configuration statique consiste à charger les trois phases de l'algorithme RPG simultanément sur le circuit FPGA ciblé et de le configurer pour chaque exemple d'apprentissage.

Dans ce cas,  $T_c=0$  ;

L'équation IV.20 devient :

$$D_s = \frac{2n}{3a_{neurone\_s} \times T_{clk} (3n + 1)} \quad (IV.22)$$

D'après le tableau IV.1

$$a_{neurone\_s} = 53994$$

$$T_{clk} = 75.92 \text{ (ns)}$$

La figure IV.6 montre la variation de la densité fonctionnelle statique en fonction du nombre de neurones.

D'après cette figure la densité fonctionnelle de la configuration statique croît lorsque le nombre de neurones augmente. A partir de 18 neurones, cette dernière se stabilise dans l'intervalle [53–54] WUPS/gate.

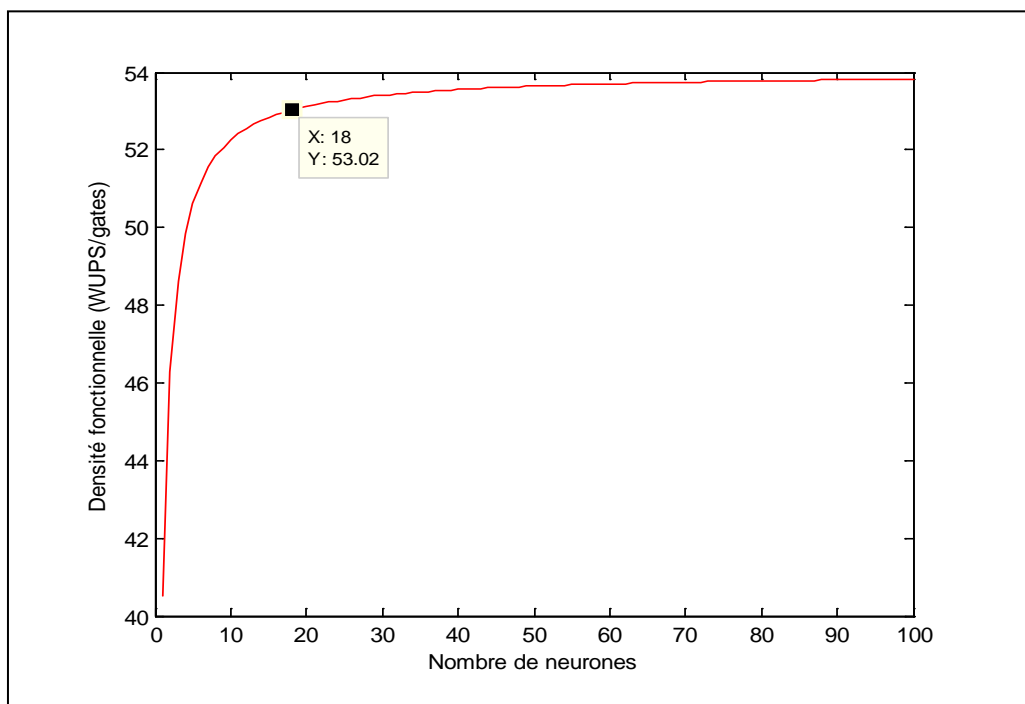


Figure IV.6 Densité fonctionnelle de la configuration statique en fonction du nombre de neurone

#### IV.3.5 Application de la RTR globale à l'algorithme RPG

L'approche adoptée pour appliquer la RTR globale consiste à diviser l'algorithme RPG en trois modules et configurer séquentiellement chaque module séparément des autres comme le montre la figure IV.6. D'abord on charge le module qui implémente la phase de propagation dans le circuit FPGA. Une fois le module chargé, on attend que l'application liée à cette phase soit exécutée. Ensuite on charge le module de calcul d'erreur et on exécute son application. Le même processus est effectué pour la phase de mise à jour des poids

synaptiques. Il est à noter que dans cette approche chaque module est implémenté avec son circuit de control.

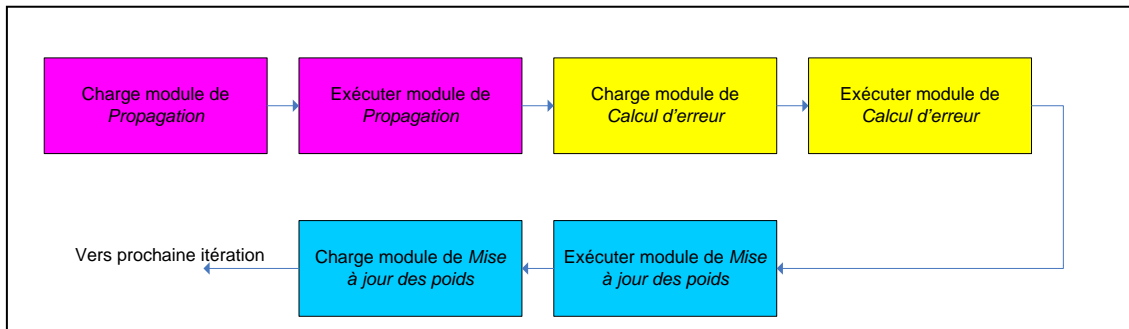


Figure IV.7 Application de la RTR globale à l’algorithme RPG

Pour le calcul de la densité fonctionnelle globale, on considère le temps d’exécution du module le plus long et la surface du module le plus grand. Le module de propagation répond à ce critère. La densité fonctionnelle de la RTR globale est :

$$D_{RTRg} = \frac{2n}{3 \times a_{neurone\_RTRg} \times T_{clk} \left(n + \frac{1}{2}\right)} \tag{IV.23}$$

D’après le tableau (IV.1)

$$T_{clk} = 75.92 \text{ (ns)}$$

$$a_{neurone\_RTRg} = 28014$$

La figure IV.8 montre la densité fonctionnelle de la configuration globale en fonction du nombre de neurones. Cette dernière croit lorsque le nombre de neurones croit. La valeur maximale atteinte se situe dans l’intervalle [300-320].

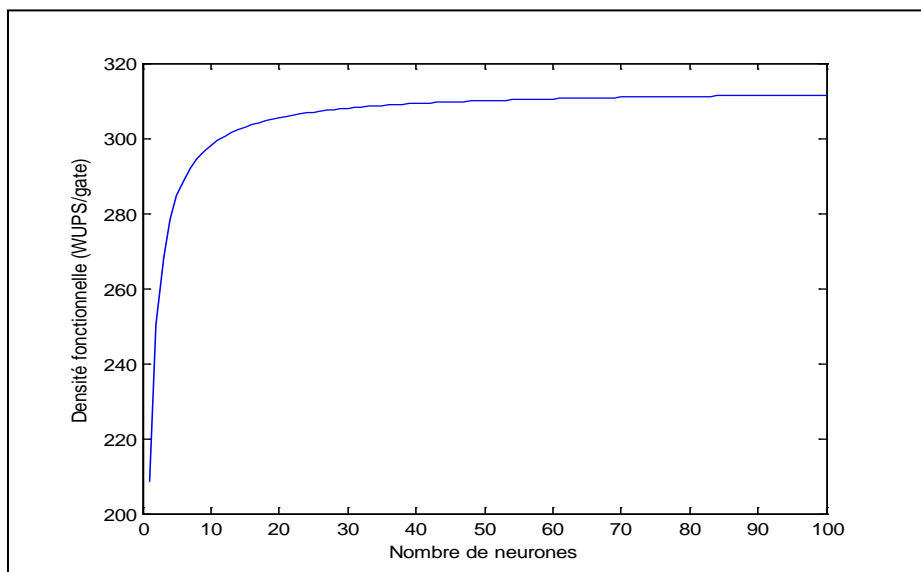


Figure IV.8 Densité fonctionnelle de la RTR globale en fonction du nombre de neurones

Afin d'évaluer l'apport de la RTR globale, il est nécessaire de considérer le rapport de configuration  $f$ , et le taux d'amélioration,  $I_{max}$ , comme présenté dans la section IV.2.3.

L'application des équations (IV.4) et (IV.9) à la configuration globale donne :

$$f = \frac{T_c}{T_e} = \frac{T_c}{3 \times T_{clk} \left(n + \frac{1}{2}\right)} \quad (IV.24)$$

Le temps de reconfiguration est donné par le datasheet de Xilinx

$$T'_c = 7.73ms$$

Lorsqu'on applique la RTR globale à l'algorithme RPG, on reconfigure trois fois le circuit. Par conséquent, le temps de reconfiguration devient :

$$T_c = 3 \times T'_c = 23.19ms$$

$$I_{max} = \frac{D_{RTR-g}(\max)}{D_S} - 1 \quad (IV.25)$$

La densité fonctionnelle de la RTR globale,  $D_{RTR-g}(\max)$ , est obtenue pour un temps de configuration,  $T_c$ , nul.

$$\begin{aligned} I_{max} &= 3 \times \frac{a_{neurone\_s}}{a_{neurone\_RTRg}} - 1 \\ &= 478\% \end{aligned} \quad (IV.26)$$

Ce résultat montre que la reconfiguration globale permet un gain de 478% dans la surface.

Pour pouvoir justifier l'utilisation de la RTR globale, il faudrait prendre en considération le rapport de configuration,  $f$ , ainsi que le gain  $I_{max}$ .

La RTR est justifiée quand le rapport de configuration,  $f$ , est inférieur au taux,  $I_{max}$ .

La figure IV.9 montre la variation du rapport de configuration en fonction du nombre de neurones ainsi que le gain,  $I_{max}$ .

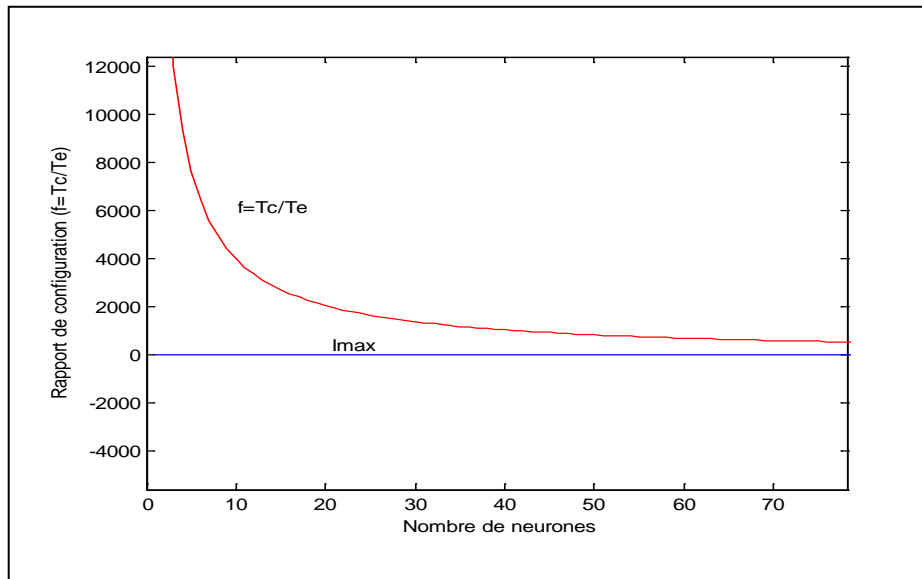


Figure IV.9 Rapport de configuration en fonction du nombre de neurones

Comme montré dans la figure IV.9, le rapport de configuration est supérieur au taux  $I_{\max}$ . Ceci s'explique par le fait que le temps de configuration est nettement supérieur au temps d'exécution.

Quoique la densité fonctionnelle de la configuration globale soit supérieure à celle de la configuration statique, et le gain en surface est de 478%, la reconfiguration globale ne peut être intéressante dans ce cas de figure, sauf si on prend en considération le nombre d'itérations pour la convergence de l'algorithme RPG à une erreur minimale.

Soit  $i$ , le nombre d'itérations ; le temps d'exécution devient :

$$T_e = 3 \times T_{clk} \left( n + \frac{1}{2} \right) \times i \quad (\text{IV.27})$$

Et le rapport de configuration  $f$  :

$$f = \frac{127 \times 10^3}{\left( n + \frac{1}{2} \right) \times i} \quad (\text{IV.28})$$

La figure IV.10 montre la variation du rapport de configuration en fonction du nombre d'itérations, pour plusieurs dimensions du réseau de neurones. Le tableau IV.2 donne le nombre d'itérations à partir duquel la configuration globale de l'algorithme RPG est justifiée pour plusieurs tailles de réseaux de neurones.

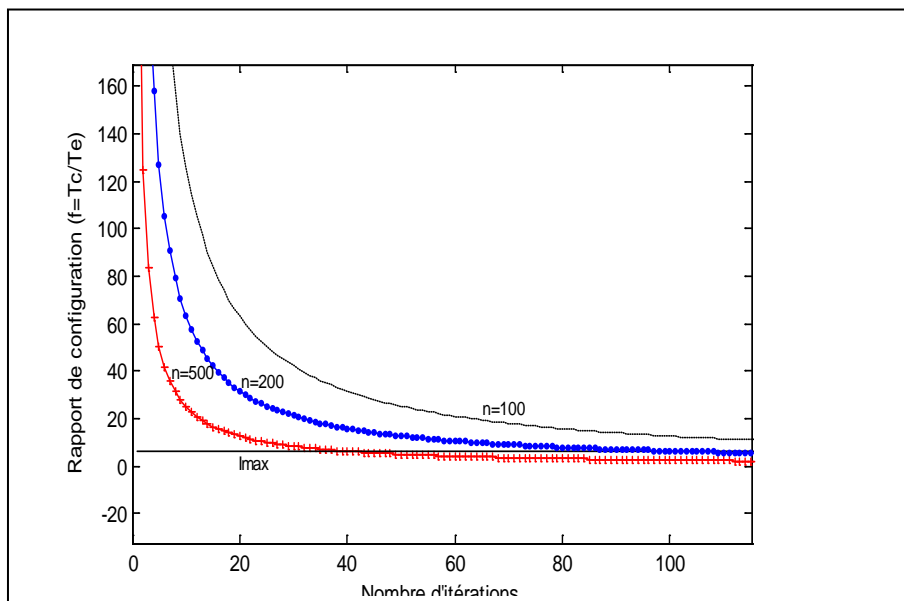


Figure IV.10. Rapport de configuration en fonction du nombre d'itérations. ( $n$ ), le nombre de neurones ou dimension du réseau.

### Discussion

En observant la figure IV.10 et le tableau IV.2, nous pouvons conclure les points suivants :

- La reconfiguration globale est intéressante pour des réseaux de grande taille ( $n \geq 9$  neurones) et après un certain nombre d'itérations.
- Pour un réseau de 9 neurones, on doit appliquer la RTR toutes les 2226 itérations. Ce nombre décroît lorsque le nombre de neurones croît. A titre d'exemple, pour un réseau de dimension : 500 neurones, on reconfigure le circuit après les 41 itérations (Voir



tableau IV.2). En d'autres termes, on charge le module qui implémente la phase de propagation dans le circuit FPGA. Une fois le module chargé, on attend que l'application liée à cette phase soit exécutée pendant 41 itérations. Ensuite on charge le module de calcul d'erreur et on exécute son application pendant 41 itérations. Le même processus est effectué pour la phase de mise à jour des poids synaptiques.

**Tableau IV.2. Reconfiguration globale :**

Nombre d'itérations en fonction de la taille du réseau

Dimension du réseau (n)	Nombre d'itérations (i)
3	217714
9	2226
50	419
100	210
200	105
500	41

#### IV.3.6 Application de la RTR locale à l'algorithme RPG [163], [168]

Dans cette approche, l'architecture considérée est constituée des trois modules : *Propagation*, *Calcul d'erreur* et *Mise à jour des poids synaptiques*. Contrairement à l'architecture de la RTR globale qui considère chaque module avec sa partie de contrôle partiel, dans le RTR locale l'ensemble des trois modules est contrôlé par un module de control global de l'algorithme RPG.

La figure IV.11 montre l'approche proposée pour l'implémentation de la RTR locale. Dans la première phase, on implémente le module de propagation ainsi que le module de contrôle global de l'algorithme RPG. Une fois ces deux modules sont exécutés, dans la deuxième phase, on reconfigure la partie de propagation tout en laissant le control global en fonctionnement normal. Ainsi, la surface du FPGA qui était occupée par la partie du module de *propagation* sera remplacée par le module de *calcul d'erreur* et de *mise à jour des poids synaptiques*.

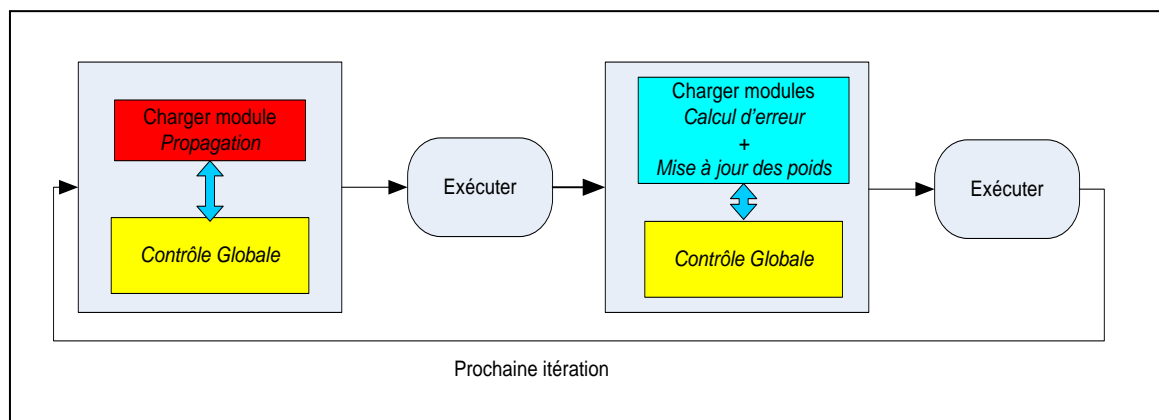


Figure IV.11 Application de la RTR locale à l'algorithme RPG

**Première phase**

La surface du circuit est :

$$A_1 = a_{control} + nxa_{propagation} \quad (IV.29)$$

Le temps d'exécution est :

$$T_1 = T(Propagation)$$

$$T_1 = 3 \times T_{clk} \left( n + \frac{1}{2} \right) \quad (IV.30)$$

D'après le tableau IV.1

$$A_1 = 1610 + n \times 28014$$

**Deuxième phase**

La surface du circuit est :

$$A_2 = 1610 + n \times 25800 \quad (IV.31)$$

Le temps d'exécution est :

$$T_2 = T(calcul\ d'\ erreur) + T(mise\ à\ jour)$$

$$T_2 = 3T_{clk} \times \left( 2n + \frac{1}{2} \right) \quad (IV.32)$$

D'après le tableau IV.1, le module de propagation possède la plus grande surface.

Par contre le temps d'exécution sera celui du temps des blocs de calcul d'erreur et mise à jour des poids.

La densité fonctionnelle dans la RTR locale s'écrit comme suite

$$D_{RTR\_l} = \frac{2n^2}{3T_{clk} (1610 + 28014 \times n) \left( 2n + \frac{1}{2} \right)} \quad (IV.33)$$

La figure IV.12 montre la variation de la densité fonctionnelle de la RTR locale, en fonction du nombre de neurones. Cette dernière croît avec le nombre de neurones, le maximum atteint est dans l'intervalle [14 et 14.5] à partir de n=16 neurones.

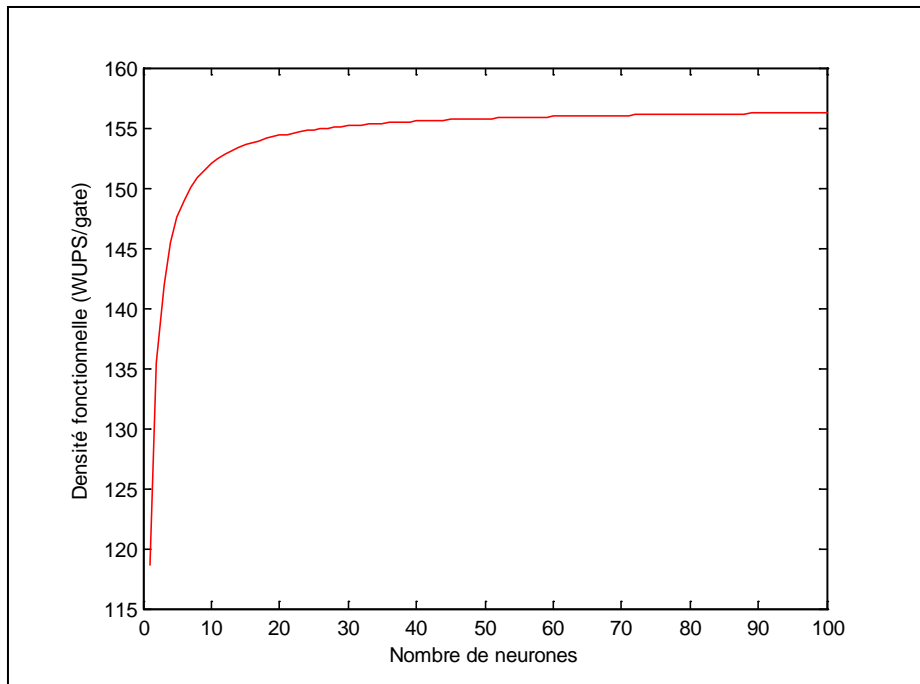


Figure IV.12 Densité fonctionnelle de la RTR locale en fonction du nombre de neurones

Le rapport de configuration est :

$$f = \frac{T_c}{3T_{clk} \left(2n + \frac{1}{2}\right)} \quad (\text{IV.34})$$

Dans ce cas le temps  $T_c$ , représente le temps de reconfiguration partielle appliquée à l'algorithme RPG.

#### IV.3.6.1 Estimation du temps de reconfiguration et de la taille du fichier « *Bit-stream* »

Comme signalé dans la section IV.2, le temps de reconfiguration partielle ou locale, ( $T_c$ ), dépend de la taille du fichier de configuration « *Bit-stream* », de la fréquence de reconfiguration du circuit, de la surface du circuit et de la famille du circuit FPGA utilisée.

Si on considère que la fréquence de reconfiguration est de 50 Mhz, cela signifie que nous pouvons reconfigurer un fichier de 50 Megabyte/sec.

Comment calculer la taille du fichier « *Bitstream* » ?

La taille du nouveau fichier « *Bit-stream* » est obtenue en multipliant le nombre de « *frames* » du design avec la taille des « *frames* ».

Soit  $B$ , la taille du fichier « *Bit-stream* »

$$B = N_{fr} \times Fr \quad (\text{IV.35})$$

Avec

$N_{fr}$ , le nombre de « *frames* » et  $Fr$  la taille des « *frames* »

Le tableau IV.3 donne le nombre maximal des « *frames* », la taille des « *frames* », et la taille du fichier « *Bit-stream* », pour les différentes familles Virtex-II [166].

Le tableau IV.4 donne, en détail, le nombre de « *frames* » des différents éléments du circuit Virtex-II.

A partir de ce tableau le nombre de « *frames* » maximale peut être déduit.

Ainsi d'après le tableau IV.4, le nombre de « *frames* » d'un circuit conçu, est donné par :

$$N_{fr} = N_{col}^{CLB} \times N_{fr/col}^{CLB} + N_{Col}^{IOB} \times N_{fr/col}^{IOB} + N_{col}^{IOI} \times N_{fr/col}^{IOI} + N_{col}^{BRAM} \times N_{col/fr}^{BRAM} \quad (\text{IV.36})$$

Avec

$N_{col}^i$ , le nombre de colonnes de l'élément  $i = CLB, IOB, IOI$  ou  $BRAM$

$N_{fr/col}^i$ , le nombre de « *frames* » par colonnes de l'élément  $i = CLB$  ou  $IOB$  ou  $IOI$  ou  $BRAM$

Pratiquement il existe des outils tel que JBITS [177], Parbits [178] ou bien PlanAhead [179], permettant de gérer et de générer automatiquement le fichier « *Bit-stream* » pour la reconfiguration partielle.

Vu la non disponibilité de ces outils à notre niveau, nous avons procédé à une méthode manuelle, qui est basée sur la conception modulaire pour la génération du fichier « *Bit-stream* » [180]. La figure IV.13 montre l'organigramme de l'approche de conception modulaire.

Tableau IV.3 Nombre maximal des « frames », taille des « frames » et taille du « Bit-stream »

Device	Frames	Frame Length (32-bit Words)	Configuration Bits <sup>(1)</sup>	Default Bitstream Size <sup>(2)</sup>
XC2V40	404	26	336,128	338,976
XC2V80	404	46	594,688	598,816
XC2V250	752	66	1,588,224	1,593,632
XC2V500	928	86	2,553,856	2,560,544
XC2V1000	1104	106	3,744,768	4,082,592
XC2V1500	1280	126	5,160,960	5,170,208
XC2V2000	1456	146	6,802,432	6,812,960
XC2V3000	1804	166	9,582,848	10,494,368
XC2V4000	2156	206	14,212,352	15,659,936
XC2V6000	2508	246	19,742,976	21,849,504
XC2V8000	2860	286	26,174,720	26,194,208

Tableau IV.4 Nombre de « frames » des différentes colonnes du circuit Virtex-II

Column Type: →	IOB		IOI		CLB		BRAM		BRAM Interconnect		GCLK	
Device: ↓	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:	Columns per Device:	Frames per Column:
XC2V40	2	4	2	22	8	22	2	64	2	22	1	4
XC2V80	2	4	2	22	8	22	2	64	2	22	1	4
XC2V250	2	4	2	22	16	22	4	64	4	22	1	4
XC2V500	2	4	2	22	24	22	4	64	4	22	1	4
XC2V1000	2	4	2	22	32	22	4	64	4	22	1	4
XC2V1500	2	4	2	22	40	22	4	64	4	22	1	4
XC2V2000	2	4	2	22	48	22	4	64	4	22	1	4
XC2V3000	2	4	2	22	56	22	6	64	6	22	1	4
XC2V4000	2	4	2	22	72	22	6	64	6	22	1	4
XC2V6000	2	4	2	22	88	22	6	64	6	22	1	4
XC2V8000	2	4	2	22	104	22	6	64	6	22	1	4

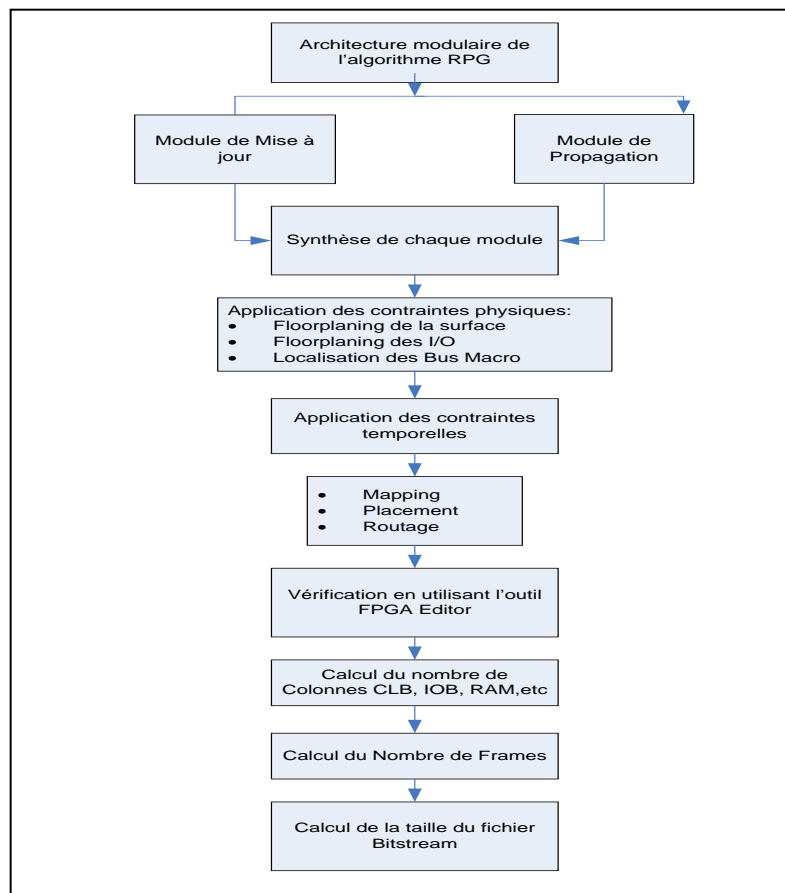


Figure IV.13 Approche de conception modulaire

L'approche suit les étapes suivantes :

### ***Architecture modulaire***

L'architecture globale est divisée en plusieurs modules. Pour notre cas, l'algorithme RPG est divisé en deux modules : le module de propagation et le module de Calcul d'erreur avec celui du module de mise à jour des poids synaptiques.

### ***Synthèse modulaire***

- Une synthèse de chaque module est effectuée indépendamment de l'autre.
- Il faut s'assurer qu'il n'existe pas des signaux communs entre les deux modules sauf le signal d'horloge.
- Tous les signaux intermédiaires doivent être des bus Macro

### ***Application des contraintes physiques :***

Le but de cette étape est l'assignement des contraintes de surface et de pins I/Os pour chaque module ainsi que le Bus Macro. Pour cela il faut respecter les règles suivantes :

- Il faut un minimum de 4 slices pour un module
- La largeur du module doit être un multiple de 4 slices
- Les frontières entre deux modules doivent être de part et d'autre d'une colonne.
- Les modules configurables doivent être dans des colonnes différentes.
- Il faut spécifier les pins I/O pour chaque module. Ces pins doivent être dans la région proche de la surface du module et dans sa plage de colonnes.
- Les modules communiquent entre eux grâce au bus macro, par conséquent il faut poser des contraintes sur ce dernier

**Implémentation active**

Dans la phase de l'implémentation active, on réalise les phases de « translation », de « mapping » et de « placement et routage » de chaque module à reconfigurer indépendamment des autres.

**Application des contraintes temporelles**

Dans cette phase les contraintes temporelles statiques sont appliquées et la simulation fonctionnelle doit être effectuée pour vérifier la fonctionnalité des différents modules.

**Vérification en utilisant l'outil FPGA Editor**

L'outil FPGA editor permet de vérifier manuellement si les modules à reconfigurer respectent les contraintes citées ci-dessus. Une fois cette étape franchie, on génère le nombre de colonnes des différents modules, le nombre de « frames » et la taille des fichiers « Bit-streams » partiels

La figure IV.14 montre le « layout » d'un neurone implémenté sans la conception modulaire. La Figure IV .15 représente le même circuit mais cette fois ci, en utilisant la conception modulaire.

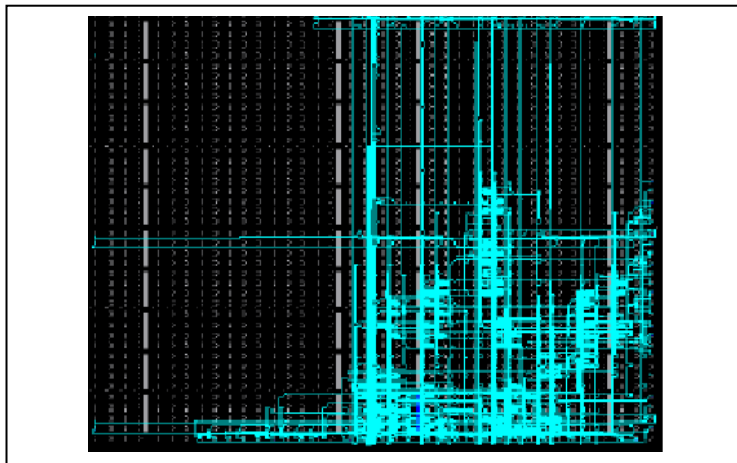


Figure IV.14 layout du neurone sans application des contraintes physiques

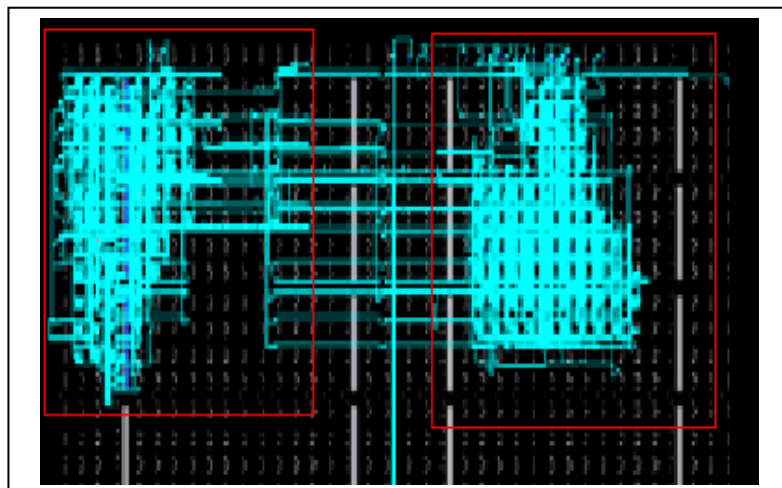


Figure IV.15 Application des contraintes physiques au layout du neurone

La figure IV.16 montre le layout de l'algorithme RPG complet de l'exemple (3, 3, 3). Le bloc 1 montre le module de propagation, le bloc 2 le module de mise à jour et calcul d'erreur et le bloc 3 le module de control global.

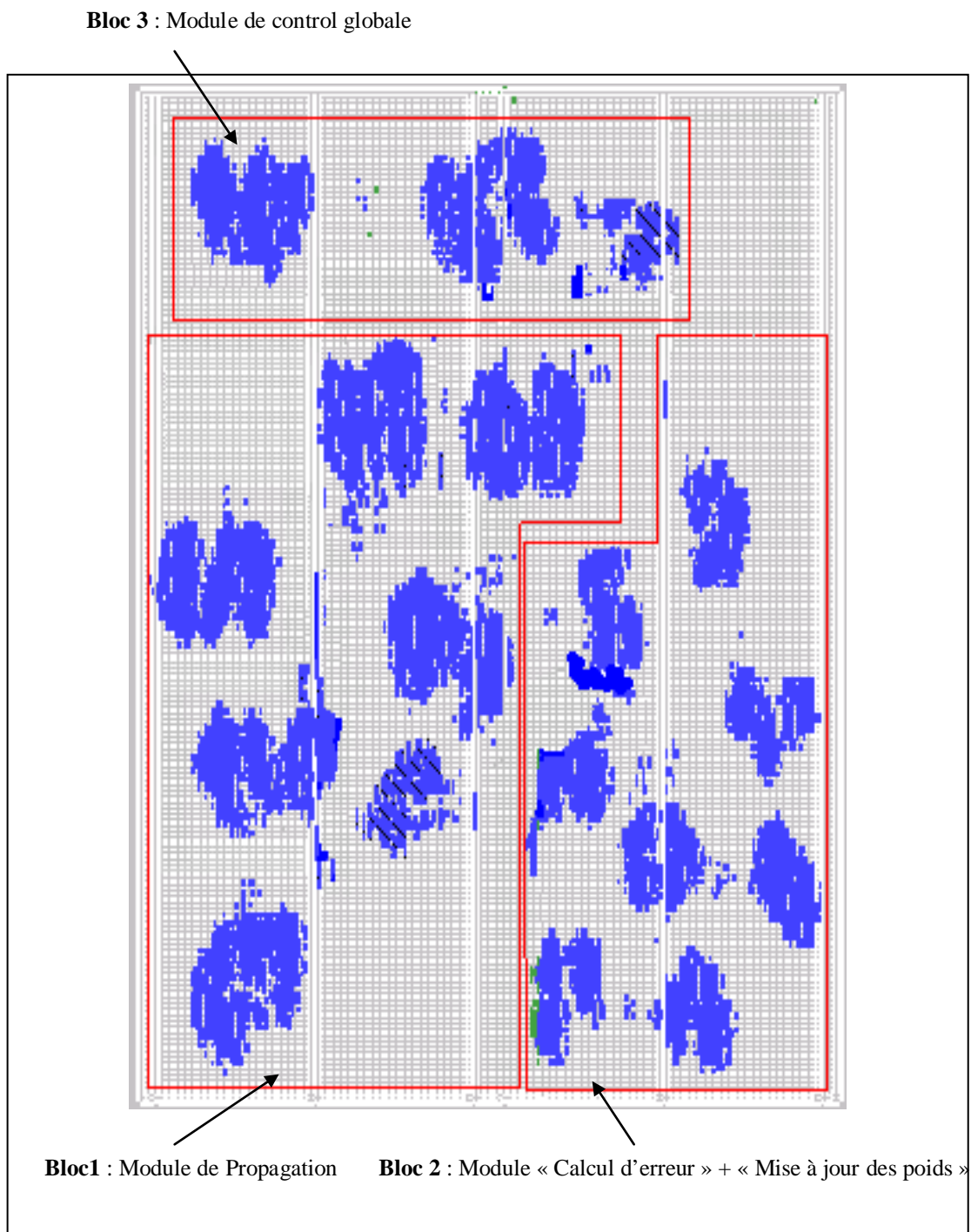


Figure IV.16 Layout de l'algorithme RPG Complet : réseau (3,3, 1)

Une fois les modules générés, le nombre de frame utilisé dans chaque module est calculé.

Le tableau IV.5 donne les temps de reconfiguration en fonction de la taille du fichier « *Bit-stream* » obtenus pour les différents modules de l'algorithme RPG. Ainsi le temps de

reconfiguration partielle est estimé à 2 ms au lieu de 7ms dans le cas d'une reconfiguration globale d'un module de l'algorithme RPG.

Tableau IV.5 Estimation du temps de reconfiguration partielle et taille du fichier « Bit-stream »

Module	Taille du Bitsream (Megabytes)	Temps de configuration, T <sub>c</sub> , (ms)
Propagation	0.1	2
« Calcul d'erreur » + « Mise à jour »	0.1	2

La figure IV.17 montre la variation du rapport de configuration en fonction du nombre de neurones ainsi que le gain en surface, I<sub>max</sub>. Ce dernier est donné par :

$$I_{\max} = \frac{53994n}{1610 + 28014n} \times \frac{3n + 1}{2n + \frac{1}{2}} - 1 \tag{IV.37}$$

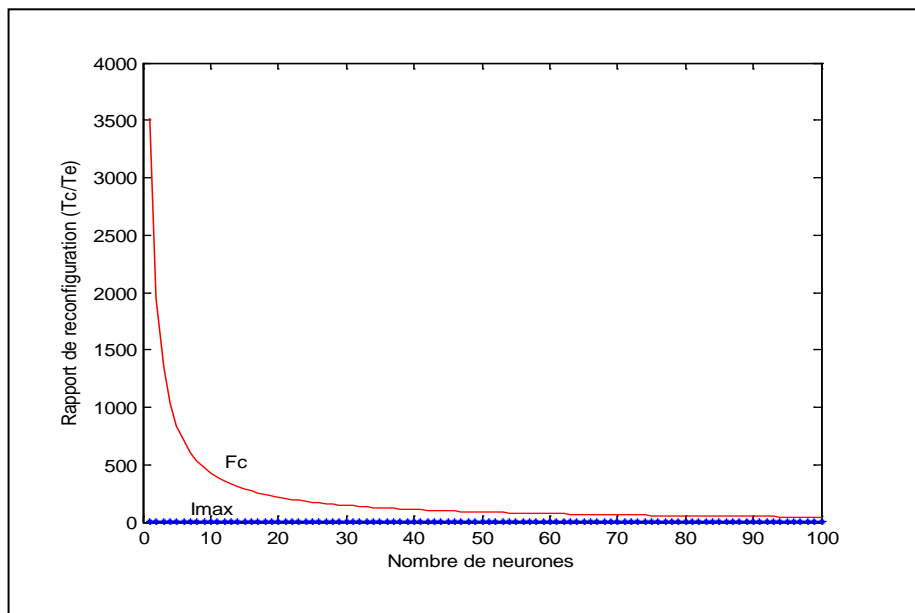


Figure IV.17 Rapport de Configuration en fonction du nombre de neurone

Comme montré dans la figure IV.17, le rapport de configuration est toujours supérieure au taux I<sub>max</sub>. Ceci s'explique par le fait que le temps de configuration est supérieur au temps d'exécution.

Quoique la densité fonctionnelle de la configuration globale soit supérieure à celle de la configuration statique, et le gain en surface, I<sub>max</sub>, est de 188%.

Comme dans le cas de la RTR globale, la RTR locale ne peut être intéressante, sauf si on prend en considération le nombre d'itérations pour la convergence de l'algorithme RPG à une erreur minimale.



Soit  $i$ , le nombre d'itérations ; le temps d'exécution devient :

$$T_e = 3 \times T_{clk} \left(2n + \frac{1}{2}\right) \times i \quad (\text{IV.38})$$

Et le rapport de configuration  $f$  :

$$f = \frac{8.78 \times 10^3}{\left(2n + \frac{1}{2}\right) \times i} \quad (\text{IV.39})$$

La figure IV.18 montre la variation du rapport de configuration en fonction du nombre d'itérations, pour plusieurs dimensions du réseau de neurones. Le tableau IV.6 donne le nombre d'itérations à partir duquel la reconfiguration partielle de l'algorithme RPG est justifiée pour plusieurs tailles de réseaux de neurones.

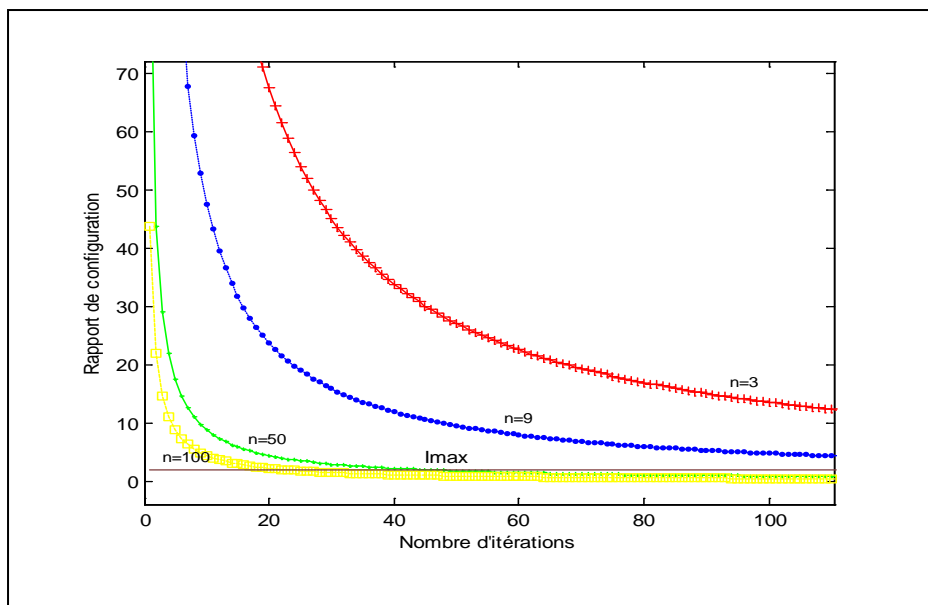


Figure IV.18 Variation du rapport de configuration en fonction du nombre d'itérations

**Tableau IV.6 Reconfiguration partielle :** Nombre d'itération en fonction de la taille du réseau

Dimension du réseau (n)	Nombre d'itérations (i)
3	623
9	253
50	47
100	23
500	5

**Discussion :**

L'application de la RTR locale à l'algorithme RPG fait ressortir les points suivants :

- Le temps de reconfiguration de la RTR locale est nettement inférieur à celui de la RTR globale.
- Pour un réseau de 3 neurones (taille minimale d'un réseau de neurones), la RTR locale n'est justifiée que si le nombre d'itérations,  $i$ , est supérieur ou égale à 623 (voir Tableau IV.6 :  $i \geq 623$ ).
- Le nombre d'itérations,  $i$ , décroît lorsque le nombre de neurones croît. Ainsi pour un réseau de 50 neurones, on doit appliquer la RTR toutes les 47 itérations. Et pour un réseau de 500 neurones, la RTR locale peut être appliquée toutes les 5 itérations. En d'autres termes, on charge le module qui implémente la phase de propagation dans le circuit FPGA. Une fois le module chargé, on attend que l'application liée à cette phase soit exécutée pendant 5 itérations. Ensuite on charge le module de calcul d'erreur et de mise à jour des poids et on exécute son application pendant 5 itérations.

**IV.4 Etude comparative**

Dans cette section, une étude comparative entre les trois approches proposées est effectuée en premier lieu. Par la suite, les résultats obtenus seront comparés à ceux de la littérature : RTR-MANN [93] et RRANN-II [94].

Pour la première comparaison, les densités fonctionnelles de la configuration statique, de la reconfiguration dynamique globale et celle de la reconfiguration partielle sont tracées dans la figure IV.19.

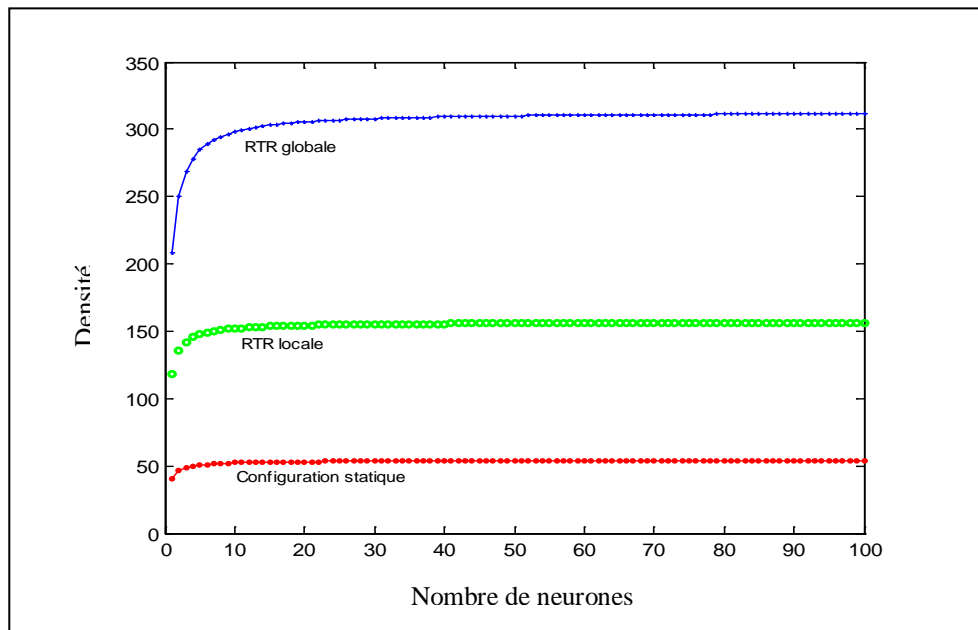


Figure IV.19 Comparaison des densités fonctionnelles : Configuration statique, RTR globale et RTR locale

Le tableau IV.7 regroupe les résultats de mesures de performances utilisées pour la comparaison entre la configuration statique, la RTR globale et la RTR partielle, en terme du temps de reconfiguration ( $T_c$ ), du gain en surface (%  $I_{max}$ ), et du gain en temps de reconfiguration (% $T$ ).

Tableau IV.7 Comparaison des trois approches proposées pour la configuration et reconfiguration dynamique

Configuration	Temps de configuration (ms)	Gain en surface %Imax	Gain en temps de reconfiguration %T
Statique	7	0	66
RTR globale	21	478	0
RTR locale	2	188	90

Afin de comparer les performances de notre approche (REC-ANN) avec ceux de la littérature, nous considérons les résultats des travaux cités dans les références [93], [94].

Les mesures de performances utilisées sont la densité fonctionnelle (D), le gain en densité fonctionnelle (%GD), le temps de reconfiguration ( $T_c$ ) et le gain en temps de reconfiguration (%T). La comparaison est faite en considérant un réseau de 60 neurones.

Le tableau IV.8 résume les performances obtenues dans les trois approches REC-ANN, RTR-MANN et RRANN-II.

Tableau IV.8 Comparaison de REC-ANN avec RTR-MANN et RRANN-II

Configuration	RTR-MANN (XCV2000-E) [93]			RRANN-II (XC3000) [94]			REC-ANN (Notre approche) (XC2V100)		
	statique	globale	partielle	statique	Globale	Partielle	statique	globale	partielle
Densité fonctionnelle (D)	-	-	78.05*	146**	148**	219**	53.7	310	155.9
Gain en densité, (% GD)**	-	-	-	0	1.35	33.33	0	82.6	65.5
Temps de reconfiguration ( $T_c$ )	-	-	-	6.6	19.8	1.16	7	21	2
Gain en temp (% $T_c$ )	-	-	-	66.6	0	94	66	0	90

(\*) La densité fonctionnelle est exprimée en WUPS/CLB/FPGA. Le circuit utilisé étant le VIRTEX-E

(\*\*) La densité fonctionnelle est exprimée en WUPS/CLB. Le circuit utilisé étant le XC3000

(\*\*\*) Vu que les métriques pour le calcul de la densité fonctionnelle sont différentes, nous avons utilisé le gain

en densité fonctionnelle  $\% D = \frac{D_{RTR} - D_s}{D_{RTR}}$  pour comparer entre RRANN-II, RTR-MANN et REC-ANN.

**Discussion :**

L'étude comparative des trois approches proposées fait ressortir les points suivants :

- La RTR globale permet le meilleur gain en terme de surface avec un taux de 478%, comparé au gain de la RTR locale (188%) et à celui de la configuration statique (%Imax=0) (tableau IV.7).
- La RTR locale permet le meilleur gain en temps de reconfiguration. Ce dernier atteint un taux de 90% comparé au gain de la configuration statique (66%) et à celui de la reconfiguration globale (0%) (Voir tableau IV.7).
- En comparant notre approche (REC-ANN) avec RTR-MANN et RRANN-II, nous pouvons conclure que REC-ANN offre des meilleures performances en termes de gain en densité fonctionnelle (%GD), par contre l'approche RRANN-II offre le meilleur gain en temps de reconfiguration. Ce résultat est logique du fait d'une part, le circuit XC3000 utilisé dans RRANN-II possède des temps de configuration et reconfiguration inférieurs à ceux du circuit XC2V1000 qui utilisé par notre approche ; d'autre part, les ressources logiques du circuit XC2V1000 sont beaucoup plus importantes que celles du circuit XC3000 et XCV2000-E.

**IV.5 Conclusion**

Nous avons mentionné au début de ce chapitre que l'objectif de la reconfiguration dynamique est d'augmenter la densité d'intégration des réseaux de neurones. Nous pouvons donc conclure qu'il est possible d'atteindre cet objectif à condition de prendre en considération le temps de reconfiguration dynamique.

Pour des applications qui nécessitent un traitement en temps réel, la reconfiguration dynamique ne peut être justifiée que si le temps de reconfiguration est inférieur au temps d'exécution. Cette condition ne peut être atteinte que si on prend en considération le nombre d'itérations dans le calcul de l'erreur minimale de l'algorithme RPG.

Une autre opportunité d'exploitation de la reconfiguration dynamique est lors de l'implémentation de l'apprentissage « batch mode » de l'algorithme RPG ?

Comparé aux travaux de la littérature, nous avons montré que notre approche, REC-ANN, offre de meilleures performances en termes de densité fonctionnelle. En ce qui concerne le gain en temps de reconfiguration, nous pouvons conclure que les résultats sont similaires puisque la différence est de l'ordre de 4% uniquement (voir tableau IV.8).

En ce qui concerne la mise en oeuvre de la reconfiguration dynamique, nous avons appliqué l'approche de conception modulaire. Les différentes étapes de cette dernière sont réalisées manuellement moyennant les différents modules de l'outil ISE foundation. Néanmoins, cette approche est longue et nécessite une expertise dans la conception pour sa mise en oeuvre.

Afin de soulever ce problème, des outils sont entrain d'émerger pour la mise en oeuvre et l'exploitation de la reconfiguration dynamique. Néanmoins, ces outils sont utilisés pour des travaux de recherche et à notre connaissance, à l'heure actuelle aucun outil industriel n'est encore développé.

Le développement d'un outil automatique qui prend en charge ce problème constitue donc un axe de recherche porteur.

Le choix de la famille du circuit FPGA et de la carte de prototypage pour valider les performances obtenues et comparer entre la théorie et la pratique, est un point crucial sans lequel la reconfiguration dynamique reste seulement une possibilité théorique des circuits FPGAs, tel le cas aujourd'hui.

Notons enfin qu'en plus de l'amélioration de la densité d'intégration, l'un des avantages directs, c'est la flexibilité obtenue à travers l'utilisation de la configuration et reconfiguration des circuits FPGAs. Ainsi, au lieu de configurer le même algorithme, il est possible d'appliquer la RTR pour configurer différents algorithmes dans le même circuit.

Une autre façon d'augmenter la flexibilité de l'algorithme RPG, est de lui appliquer le concept de « Design reuse ». C'est l'objet du prochain chapitre.

---

# *CHAPITRE V*

---

**APPLICATION DES CONCEPTS DE DESIGN REUSE POUR  
L'IMPLEMENTATION DE L'ALGORITHME RPG**

## V.1 Introduction

Dans le domaine de la conception, le terme anglais « *Reuse* » est adopté pour désigner la réutilisation de blocs ou de données déjà conçus dans une nouvelle conception hardware ou software. Néanmoins les concepts de « *Design ForReuse* », et de « *Design With Reuse* », sont beaucoup plus utilisés dans la conception et le développement hardware.

La première question qui vient à l'esprit est: pourquoi le *Reuse* dans la conception hardware?

Une bonne réponse à cette question trouve son explication dans la figure V.1.

En effet, depuis la naissance du premier circuit intégré dans les années soixante, le domaine de la microélectronique n'a cessé de se développer. Ce développement a déjà été exprimé depuis 1965 par Gordon Moore [181], ingénieur de Fairchild Semi-conducteur, un des deux fondateurs d'Intel. Il expliquait que la complexité des semi-conducteurs doublait tous les dix-huit mois à coût constant depuis 1959, date de leur invention. Cette augmentation exponentielle fut rapidement nommée Loi de Moore ou, compte tenu de l'ajustement ultérieur, première loi de Moore.

En 1975, Moore [182] réévalua sa prédiction en posant que le nombre de transistors des microprocesseurs sur une puce de silicium double tous les deux ans. Bien qu'il ne s'agisse pas d'une vraie loi physique, cette prédiction s'est révélée étonnamment exacte.

Entre 1971 et 2001, la densité des transistors a doublé chaque 1,96 année. En conséquence, les machines électroniques sont devenues de moins en moins coûteuses et de plus en plus puissantes.

La deuxième loi (figure V.1) est à peu près vérifiée depuis 1973, mais depuis 2004, elle souffre d'un petit ralentissement dû notamment à des difficultés de dissipation thermique, qui empêchent une montée en fréquence en dépit de la taille plus faible des composants.

Cette évolution rapide de la densité d'intégration va de pair avec une augmentation de la complexité de conception des circuits intégrés.

Le terme de complexité de conception peut signifier plusieurs choses.

Suivant les références [183] et [184], les principaux facteurs impliquant une complexité croissante sont :

- L'augmentation du nombre de transistors par circuit
- L'usage de nouvelles architectures de circuits et de nouveaux algorithmes
- L'agrandissement de l'équipe de conception
- L'utilisation de technologies récentes

Bien sur, le temps de conception augmentera si la complexité du circuit augmente, ce qui pénalise le temps de mise sur le marché du produit final « *time to market* ».

En gardant un processus de conception identique à ce qui se fait à l'heure actuelle, il faudrait augmenter la taille des équipes de développement. Toutefois il a été démontré qu'en augmentant la taille des équipes, l'efficacité n'évolue pas de manière proportionnelle. Ainsi, comme montré dans la figure V.1 il apparaît une faille dans la productivité connue sous le nom de « design gap ».

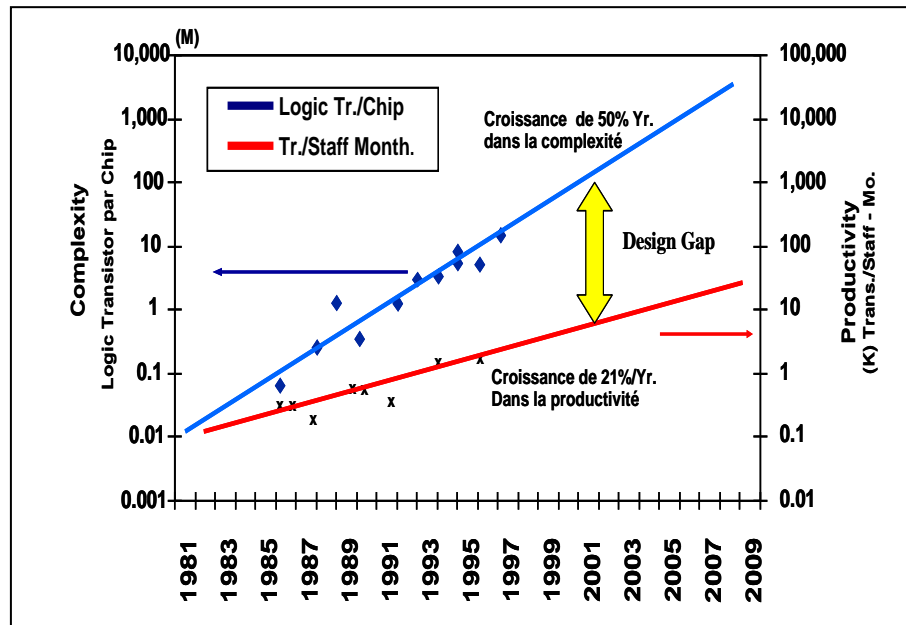


Figure V.1 Evolution de la loi de Moore et de la productivité

Afin de résoudre le problème du « design gap », l'International Technology Road Map for Semiconductors (ITRS) propose les trois solutions suivantes [184] :

- L'exploitation du « Reuse »
- Le développement de méthodologies avancées et d'outils pour la conception et le test ; en particulier par l'utilisation de niveaux d'abstraction élevés.
- Le développement de plateformes de conception dédiées à des applications spécifiques

Ces solutions constituent des challenges qui peuvent être considérées comme directives de recherches actuelles dans le domaine. Il est à noter que l'ITRS est représenté par des experts et sponsorisé par les cinq associations suivantes : la SIA (The US Semiconductor International Association), la EECA (European Electronic Component Manufacturer's Association), la EIAJ (Electronic Industries Association of Japan), la KSIA (Korea Semiconductor Industry Association) et la TSIA (Taiwan Semiconductor Industry Association).

Quoique les trois solutions proposées pour résoudre le problème du design gap sont interdépendantes et complémentaires les unes des autres, nous nous intéressons plus particulièrement à l'exploitation du « Reuse ».

Dans la littérature, le « Reuse » désigne principalement deux choses : le « Design For Reuse » et le « Design With Reuse ».

Dans ce chapitre nous proposons l'application de ces concepts à l'algorithme de la rétropropagation du gradient (RPG).



Pourquoi le « *Design For Reuse* » et le « *Design With Reuse* » dans la conception des réseaux de neurones?

L'idée est justifiée principalement par les trois points suivants et qui sont inspirés principalement par les orientations de l'*ITRS* dans le domaine de la conception des circuits intégrés de manière générale et aussi par le *Neural Network roadmap* [1,2] dans la conception de hardware neuronal:

- D'un point de vue architectural, les réseaux de neurones et plus particulièrement l'algorithme RPG, peuvent être vus soit comme des systèmes sur puce à part entière (voir chapitre II, section II.3.4.4) ; soit comme un sous bloc d'un système complexe qui contribue à la solution complète du système en question (chapitre II, section II.3.4.3.6). Par voie de conséquence, l'application du concept de *Reuse* contribue à diminuer la complexité de conception de ces systèmes.
- L'application du *Reuse* à un haut niveau d'abstraction permet d'obtenir des descriptions des réseaux de neurones génériques, paramétriques, configurables et réutilisables, permettant ainsi d'augmenter la flexibilité tant recherchée dans les implémentations hardware des réseaux de neurones.
- Actuellement, et dans le but de diminuer le facteur « *time to market* », les concepteurs de circuits ASICs et FPGAs, plus particulièrement ceux des circuits DSPs, sont entrain de proposer des plateformes pour la conception hardware des fonction DSPs élémentaires tel que les circuits de filtrage, de corrélation, de convolution, etc. [185]. Ces plateformes utilisent une méthodologie de conception basée principalement sur le concept du design reuse. L'objectif final est d'offrir au concepteur hardware, des outils simples et similaires à l'outil MATLAB en software. Or, on sait que du point de vue software, l'outil *neural network tool box* est intégré dans l'outil MATLAB, alors pourquoi ne pas préparer le terrain pour le développement d'une plateforme dédiée à la conception et l'implémentation hardware des réseaux de neurones et qui sera intégré dans les outils hardware?

Dans la littérature, les travaux qui ont proposé, de manière implicite, des implémentations basées sur le concept de *Reuse*, sont les processeurs **Soft Totem NC3003** et **VNP**. Ces travaux proposent des descriptions SoftCores des réseaux de neurones (Voir section II.3.4.3.6 du chapitre II).

**Soft Totem NC3003** [144] a été d'abord conçu et implémenté en utilisant l'approche de conception « full custom » selon une technologie particulière, par la suite une description software a été développée dans le but de le rendre indépendant de la technologie et par conséquent pouvoir le réutiliser. Néanmoins l'inconvénient de cette description est que les données ont un format limité, le réseau est limité à 32 neurones et aucune information n'est donnée quand à l'implémentation « *on chip training* » ou bien « *off chip training* » de l'algorithme RPG.

Le processeur neuronal virtuel **VNP** [147], propose une approche basée sur l'utilisation d'une bibliothèque de portes logiques élémentaires. Néanmoins, l'utilisation d'une description au niveau de portes logiques est une tâche fastidieuse, lente, encombrante et pourrait induire à des erreurs dans la conception finale du réseau de neurone en question.

A la différence des approches précédentes, nous proposons une approche de conception des réseaux de neurones basée sur la description VHDL d'une bibliothèque hiérarchique et synthétisable de circuits modélisant les différentes implémentations de

l'algorithme RPG : « *on chip training* », « *off chip training* » et utilisation de la reconfiguration dynamique, « RTR ».

Les circuits à l'intérieur de cette librairie sont paramétrés et génériques. Ainsi, à chacun des modèles de réseau correspond une bibliothèque paramétrée de cellules (multiplieurs série/ parallèle, additionneurs, multiplexeurs, ROMs, RAMs, etc).

Les paramètres en question concernent : le nombre de couches du réseau de neurones, le nombre de neurones de chaque couche du réseau, la taille du mot, le degré de multiplexage, la précision etc. De plus chaque circuit du réseau est décrit selon plusieurs variétés architecturales possibles, permettant ainsi de: couvrir un large domaine d'applications, d'atteindre des performances élevées, une grande flexibilité, une indépendance par rapport à la technologie et un prototypage rapide.

Pour atteindre cet objectif, les concepts de « *design with reuse* » et de « *design for reuse* » ont été appliqués tout au long du projet. La section V.2 est consacrée à la présentation des concepts de base liés au **Reuse**. La section V.3 est consacrée à l'application de ces concepts à l'algorithme RPG. Dans la section V.4, une estimation du coût du « Design Reuse » est présentée. Dans la section V.5, une évaluation du code VHDL de l'IP est présentée moyennant l'outil OpenMore. Et enfin une conclusion.

## V.2 Concepts de base

Le « **Reuse** » présente des challenges spécifiques aux concepteurs. Néanmoins, pour être réutilisable, un design doit d'abord être utilisable. En d'autres termes le design doit être correct et robuste. Pour cela il faut appliquer certaines règles :

### V.2.1 Règles de conception

#### V.2.1.1 Règles de conception pour l'utilisation

Conformément au manuel de méthodologie de Reuse : RMM [186], pour avoir un bon design, les règles de base suivantes doivent être appliquées :

- Bien coder
- Bien commenter
- Bien vérifier
- Bien documenter

#### V.2.1.2 Règles de conception pour la réutilisation « Reuse »

En plus des règles citées ci-dessus, pour être réutilisable, un design doit vérifier les règles suivantes :

- **Conçu pour résoudre un problème général** : Ce qui signifie que le circuit conçu doit être **configurable** pour plusieurs applications.
- **Conçu indépendamment de la technologie** : Pour les Soft macro, cela signifie que l'outil de synthèse doit produire des résultats ayant une qualité suffisante pour une variété de bibliothèques. Pour les hard macro, cela signifie qu'il faut prévoir une stratégie efficace pour mapper le macro dans de nouvelles technologies.
- **Portable** : **Conçu pour être simulé avec une variété de simulateurs**. Un macro ayant un fichier de vérification « testbench » qui marche pour un seul simulateur n'est pas portable. Une bonne pratique du « Reuse » doit produire des descriptions de

fichiers « testbench » en VHDL et Verilog et doit être portable sur la plupart des simulateurs commerciaux.

- **Conçu avec des interfaces standards**
- **Vérifié avec un haut degré de confiance** : En plus de la vérification fonctionnelle, un prototypage de l'IP conçu doit être réalisé et testé dans un système réel.
- **Entièrement documenté en terme d'applications appropriées et de restrictions** : En particulier, les configurations valides et les valeurs des paramètres doivent être clairement mentionnées. Les interfaces et les restrictions sur l'utilisation du macro doivent être documentés.

Ainsi on voit qu'un effort supplémentaire est nécessaire pour concevoir un module réutilisable.

### V.2.2 Le modèle en couches

A partir des règles établies précédemment, il apparaît clair que le passage d'une simple conception à une conception réutilisable fournit un module qui a un certain niveau de qualité. Le modèle en couche (Figure V.2) permet de classer un bloc par son niveau de qualité [187] :

- **Fonctionnel** : C'est le niveau minimum qu'un bloc doit atteindre pour être utilisé. Le module doit être vérifié par rapport à l'application visée et les timings respectés. Toutefois, le code n'est pas optimisé, rien n'est documenté et la moindre modification sera très coûteuse.
- **Maintenable** : Par rapport au niveau fonctionnel, un bloc maintenable serait tout aussi fonctionnel mais également bien documenté (schéma, codage propre et clair, commentaire de code, spécifications sommaires, etc.). Par contre il n'est pas nécessairement optimisé. L'intérêt d'un tel module est qu'il pourra être modifié mais il n'est pas entièrement configurable.
- **Réutilisable** : Un module réutilisable est un enrichissement d'un module maintenable. Dans ce cas le module est beaucoup plus configurable, adapté à un usage général, et les règles de réutilisation sont complètement respectées.

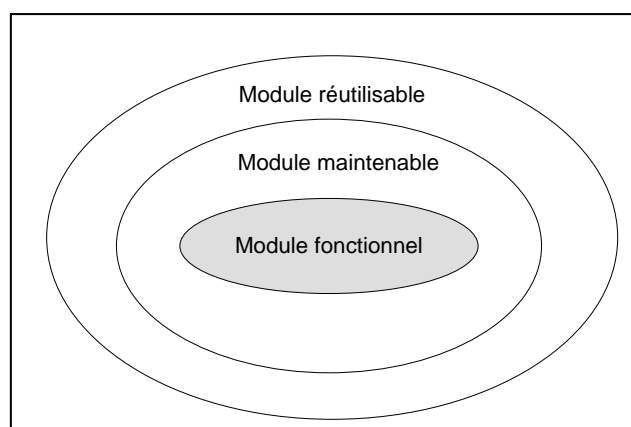


Figure V.2 Le modèle en couche de conception

Le modèle en couche obtenue permet de voir que chaque niveau de qualité est un enrichissement par rapport à un niveau inférieur. Ainsi le passage d'un niveau à un autre supérieur nécessite un effort supplémentaire de conception.

### V.2.3 Le Modèle Industriel pour le Reuse

Le *Reuse*, comme solution au design gap, a créé une nouvelle branche dans l'industrie des semi-conducteurs, appelée **IP- Business** [188], [189] comme le montre la figure V.3 :

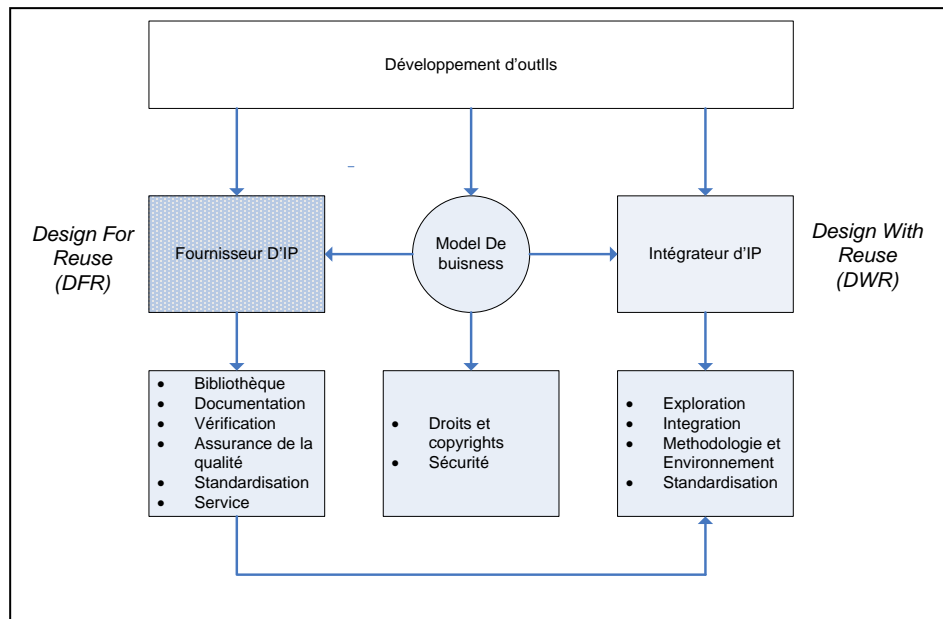


Figure V.3 Modèle Industriel pour le Reuse des IP

Le modèle met en valeur trois acteurs principaux :

- **Les fournisseurs/développeurs d'IPs** qui délivrent des bibliothèques de composants IPs conçus sous forme de « hardcores », de « softcores » ou bien de « firmcores ». Dans cette situation, il s'agit de faire une conception planifiée des IPs dans le but de les réutiliser. On parle alors de conception en vue de la réutilisation, en anglais « *Design For Reuse* » (**DFR**). Les aspects de documentation, de vérification, d'assurance de qualité, de standardisation et de service après vente (maintenance, mis à jour, garantie, etc.) constituent eux aussi les problèmes qui doivent être pris en charge et résolus par les fournisseurs/développeurs d'IPs.
- **Les intégrateurs d'IPs** sont typiquement les concepteurs/utilisateurs des circuits fournis par les fournisseurs/développeurs d'IPs. Dans cette situation, il s'agit de faire des conceptions en réutilisant les IPs délivrés par les fournisseurs d'IPs. On parle alors de conception avec la réutilisation ou en anglais « *Design With Reuse* », (**DWR**). Les principaux problèmes dont doivent faire face les intégrateurs d'IPs sont : l'exploration et l'exploitation des différents IPs, l'intégration de ces derniers dans un système, la méthodologie de réutilisation et la standardisation du processus d'acquisition des IPs.
- **Les développeurs d'outils** qui fournissent aux développeurs et aux intégrateurs d'IPs des outils et des méthodologies permettant de supporter le développement des IPs et leur intégration dans des systèmes.
- Un autre élément clé est **le modèle de business** pour la distribution, le service et le commerce des IPs.

## V.2.4 Supports techniques pour le *Reuse*

Afin de pouvoir appliquer le concept de « *Reuse* » il faut d'abord spécifier l'outil de conception, ensuite définir un langage de description et lui appliquer des règles de codage.

### V.2.4.1 Les outils de conception

La figure V.4 montre les différents niveaux de maturité de outils de conception disponibles.

La plupart des développeurs d'outils proposent aux développeurs et intégrateurs d'IPs, un certain nombre d'outils : éditeurs de schémas, simulateurs HDL, synthèse logique, synthèse FPGA. Ces outils constituent le support de base pour n'importe quel projet de conception. Le *Reuse* du hardware est au bas niveau est appel l'utilisation au niveau portes logiques.

Dans le deuxième niveau les outils de synthèse au niveau RTL « *Register Transfer Level* » sont proposés. Le *Reuse* du hardware appel l'utilisation de composants tel que les additionneurs, multiplieurs, circuits de mémorisations, etc. ces derniers sont stockés dans des bibliothèques. Un exemple de bibliothèque est le module *CoreGenerator* de l'outil ISE Fondation.

Le troisième niveau de maturité est obtenu lorsque l'outil est dédié à des applications spécifiques tel que les fonctions DSP élémentaires (FFT, DCT, FIR, etc.). Le *Reuse* appel l'utilisation des fonctions au lieu de circuits élémentaires. *System Generator for DSP* [190], *FIRGEN* [191], et *LMSGEN* [192] sont des exemples d'outils dédiés aux applications du traitement du signal.

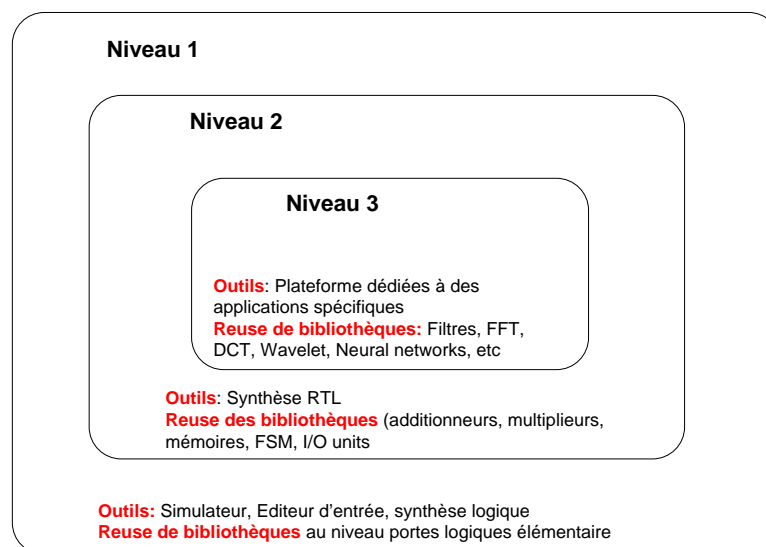


Figure V.4 Les niveaux de maturité des outils de conception

### V.2.4.2 Le VHDL et le *Reuse*

En plus de son émergence comme standard industriel pour la conception hardware, le VHDL possède des caractéristiques supplémentaires qui supportent le *Reuse*.

- **Indépendance vis-à-vis de la technologie :** Un circuit est décrit en VHDL indépendamment de la technologie. Par la suite l'outil de synthèse permet de choisir parmi les technologies disponibles.
- **Portabilité :** Etant un standard IEEE, le VHDL permet la portabilité la réutilisation des modèles VHDL sur une grande gamme d'outils de conception et simulation.

- **L'encapsulation** : L'encapsulation réduit le nombre de détails que le concepteur doit confronter et ceci à travers la représentation d'une conception comme un ensemble de modules interactifs. Ainsi, le concepteur n'a pas à savoir comment ces modules 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 « packages », les appels de fonctions, les procédures et la déclaration des entités.
- **Les packages** sont des fichiers contenant des données, des génériques, des fonctions et des procédures qui peuvent être partagés et réutilisés dans plusieurs projets ou sous projets. Par l'utilisation d'un package, l'information stockée dans ce dernier est utilisée sans être dupliquée.
- **Les entités** : La déclaration d'une entité représente une interface à un module VHDL. Elle est similaire à un icône dans un environnement graphique.
- **L'héritage** en VHDL 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 une entité ou un bloc. Cette information est statique pour le bloc. Un bloc générique est vu de l'extérieur comme un bloc paramétré. Il existe trois sortes de paramètres génériques en VHDL :
  - *Les génériques reliés aux délais*: Ils permettent de modifier le contenu temporel du module.
  - *Les génériques reliés à la structure* : Ils permettent de modifier la taille des signaux et des variables dans le but d'avoir des structures de plus grande ou plus petite taille.
  - *Les génériques reliés au contenu* : Ils permettent de décrire des boucles de longueur paramétrée.
- **La configuration**: Certaines instructions du code VHDL permettent de construire un module complexe à partir de simples modules et de le configurer selon le besoin.
- **Une excellente documentation** : Le VHDL permet la réalisation de la documentation durant la conception ainsi que les mises à jour.

#### V.2.4.3 Règles pour le codage

Afin de promouvoir le Reuse, un effort collaboratif entre chercheurs et industriels a été fourni. Le résultat est un ensemble de règles, recommandations et guidelines pour les concepteurs. Ces règles sont publiées dans le manuel **RMM**. Elles concernent d'abord les règles et les pratiques élémentaires pour une bonne conception. Pour concevoir un module réutilisable il faut considérer, en plus:

- **Des règles au niveau système :**
- **Des règles pour la portabilité**
- **Des règles pour l'horloge CLK et le RESET**
- **Des règles pour la synthèse**
- **Des règles pour la configuration**
- **Des règles pour la vérification**
- **Des règles pour la documentation**

L'*Annexe B* résume en détails les règles et recommandations tel que données dans le livre **RMM**.

### V.3 Application des règles de conception pour la réutilisation des réseaux de neurones

Pour être réutilisable, un réseau de neurone doit satisfaire les règles de conception citées dans la section V.2.1.2 :

- **Configuration** : le réseau de neurone doit être conçu pour résoudre un problème général. Cela signifie, que le circuit doit :
  - **Prendre en charge plusieurs variétés d'algorithmes neuronaux** : algorithme de la rétropropagation du gradient, algorithme de Kohonen, algorithme de Hopfield, etc.
  - **Etre configuré pour plusieurs domaines d'applications** : traitement du signal, traitement de la parole, traitement de l'image, applications industriels, optimisation, etc.

La configuration et la reconfiguration dynamique des circuits FPGAs sont des solutions permettant de programmer plusieurs algorithmes dans le même circuit.

Pour couvrir un large domaine d'applications, le réseau de neurones choisie, doit être de dimension quelconque et possède plusieurs variétés architecturales possibles.

Cependant, vu la complexité des différents algorithmes, nous avons choisi l'algorithme de la rétropropagation du gradient comme premier exemple d'application. Le choix de l'algorithme est justifié par sa large utilisation par la communauté scientifique et son succès à résoudre des problèmes de classification, d'approximation de fonction et autres, dans un large domaine d'applications.

- **Indépendance vis-à-vis de la technologie** : Cette règle peut être satisfaite en utilisant le langage VHDL pour la description du réseau de neurone.
- **Portabilité** : Pour satisfaire cette règle, les outils ISE fondatio et Leonardo Spectrum de Mentor Graphics [193] sont utilisés pour la synthèse et les outils ModelSim de ISE Fondation et Modeltech de Mentor Graphics [194] pour la simulation. Le choix de ces outils est justifié par leur disponibilité à notre niveau.
- **Conçu avec des interfaces standard** : Le VHDL étant un standard IEEE, par conséquent la déclaration des signaux d'interface selon le format IEEE permet de respecter cette règle.
- **Vérifié avec un haut degré de confiance** : Pour cela, un prototype doit être réalisé sur une carte FPGA
- **Entièrement documenté en terme d'applications appropriées et de restrictions** : Une documentation doit être réalisée.

### V.4 Méthodologie de conception de l'algorithme RPG basée sur les concepts de (DFR) et (DWR) [195]

Sur la base des règles de conception établies ci dessus, dans ce qui suit nous proposons une méthodologie de conception de l'algorithme RPG basée sur les concepts de (DFR) et de (DWR)

La figure V.5 montre la méthodologie utilisée. Celle-ci est basée sur une approche de conception descendante « top down », et dans laquelle le concepteur/utilisateur est guidé pas à pas dans le processus de conception de son algorithme.

D'abord, l'utilisateur choisit les différents types d'implémentations de l'algorithme RPG : l'implémentation « *off chip training* », l'implémentation « *on chip training statique* », et l'implémentation « *on chip training (RTR)* », basée sur la reconfiguration dynamique.

Par la suite et selon le type d'implémentation choisi, un IP- ANN est généré.

A ce niveau, le concepteur/ utilisateur peut fixer les dimensions du réseau de neurones en question : nombre de couches, nombre de neurones dans chaque couche, le nombre d'entrées/ sorties du réseau, etc.

L'étape suivante concerne la conception du neurone qui se fait moyennant une bibliothèque de « cores » ou noyaux réutilisables et paramétrés (multiplieurs, accumulateurs, additionneurs, RAM, ROMs, etc.) ; de plus chaque noyau est décrit selon plusieurs variétés architecturales possibles, permettant ainsi de couvrir un large domaine d'applications, d'atteindre des performances élevées et une grande flexibilité.

Une fois le neurone conçu, les différentes couches peuvent être conçues par une simple duplication du neurone et ainsi de suite jusqu'à génération du réseau de neurone en question. Par la suite, un code VHDL au niveau RTL est généré.

Avant la synthèse, la simulation fonctionnelle est requise. Ensuite, le code est passé à travers un outil qui réalise la synthèse au niveau RTL et l'optimisation, selon la technologie FPGA ciblée et selon les contraintes de surface/vitesse.

Le résultat est un fichier « netlist » qui sera utilisé par les outils de placement et routage. A ce niveau la vérification est requise avant le prototypage final du circuit sur FPGA. Afin de guider le concepteur/utilisateur, une documentation est nécessaire à chaque étape du processus de conception.

Ainsi, la méthodologie proposée est basée sur une conception planifiée des réseaux de neurones, dans laquelle les règles de conception pour la réutilisation sont appliquées.

Afin de mieux exploiter le concept de réutilisation, l'approche proposée repose sur l'application du « **DFR** » pour la génération de l'architecture du réseau de neurones et sur le « **DWR** » pour l'exploitation de la bibliothèque des noyaux prédéfinis.



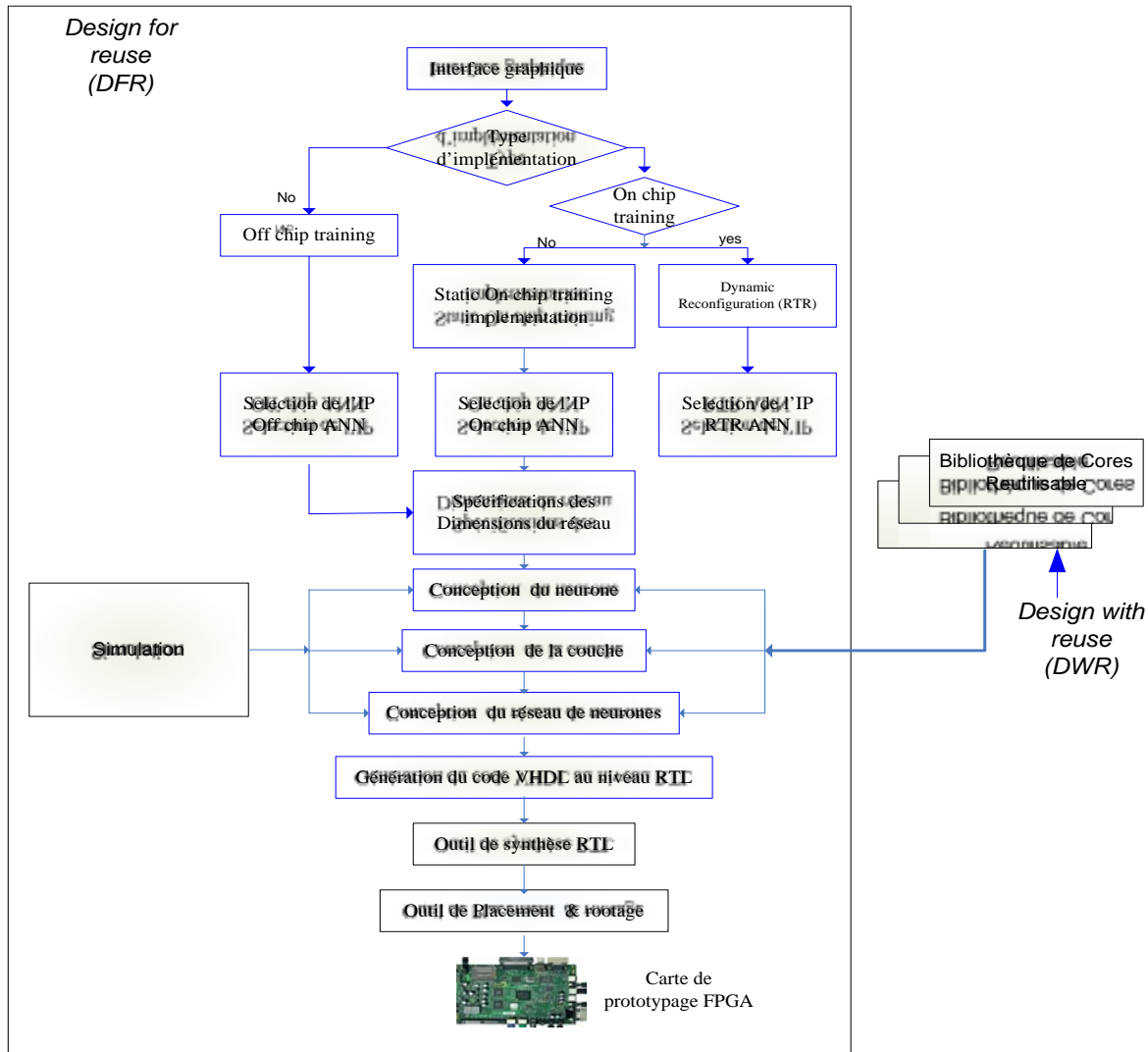


Figure V.5 Méthodologie de conception de l’algorithme RPG : Application du DFR et DWR

Dans ce qui suit l’application de ces deux concepts à l’algorithme RPG est présentée.

#### V.4.1 Stratégie de « DFR »

Afin de faciliter l’application du « **DFR** », il faut d’abord examiner les niveaux de complexité liés à l’algorithme RPG. Ce qui amène à considérer séparément les trois types d’implémentations qui ont été développées dans les chapitres III et IV respectivement: l’implémentation « off chip training », l’implémentation « on chip training statique » et l’implémentation « on chip training » en utilisant la reconfiguration dynamique (RTR).

Chaque type d’implémentation est représenté par un IP-ANN. Le deuxième point à prendre en considération, est la modularité de l’architecture et dans laquelle un effort maximum doit être fourni pour faire ressortir les blocs (modules) similaires.

##### V.4.1.1 Sélection des différents IPs

Afin de faciliter la sélection, nous avons opté pour la représentation structurelle des différents IPs en utilisant un éditeur de schéma comme outil d’entrée pour la conception. Une étude judicieuse nous a amené à choisir l’éditeur de schéma de Mentor Graphics, vue son interface conviviale et sa disposition à faciliter le « DFR ».

#### V.4.1.1.1 Sélection de l'IP –ANN « Off chip training »

La figure V.6 montre l'IP-ANN « off chip training » qui est constitué de deux modules : le module de donnée « data path » et le module de control.

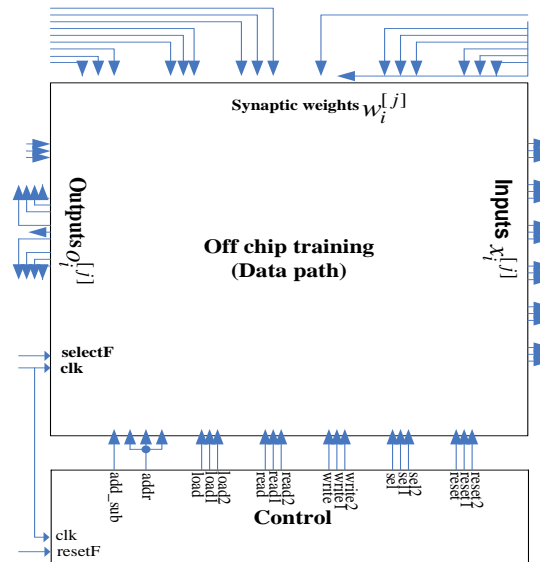


Figure V.6 Structure externe de l'IP-ANN « Off chip training »

A un haut niveau ces deux modules sont représentés par des boîtes noires, et seulement les entrées/sorties du réseau de neurone sont montrées au concepteur/utilisateur.

En cliquant à l'intérieur du module « **Data path** », on peut accéder à l'architecture du réseau qui est constituée de trois boîtes noires représentant les différentes couches possibles qui peuvent être implémentées (Figure V. 7 (a)). En cliquant sur chaque boîte noire, on peut accéder à l'architecture de la couche, formée de plusieurs neurones (Figure V.7. (b)). Enfin, en cliquant sur le module neurone, on peut accéder à l'architecture du neurone représentée par la Figure V.7 (c). Ce dernier est lui aussi constitué par cinq boîtes noires représentant les différents IPs à savoir :

- Le bloc Mémoire
- Le bloc Multiplexeur (Mux)
- Le bloc Multiplieur (Mult)
- Le bloc Accumulateur (Accum)
- Le bloc de la fonction d'activation

Ainsi, en adoptant cette représentation, le concepteur/utilisateur peut modifier le nombre de couche de son réseau de neurones, par une simple copie/coller ou bien une suppression du module couche. Il peut aussi modifier le nombre de neurones dans une couche, par une simple copie/coller ou bien suppression du module neurone, permettant ainsi de concevoir des réseaux de dimension quelconque.

Le module de contrôle est une machine d'état composée d'une de trois phases : control du neurone, control de la couche et control du réseau. En considérant le fait, que les neurones d'une même couche fonctionnent en parallèle, le control de la couche devient similaire au control du neurone. Dans le chapitre III, nous avons proposé une machine d'état de type Moore et dans laquelle le système varie seulement quand son état change (Figure V. 8 (a)). L'inconvénient de cette machine est qu'elle n'est pas réutilisable car si on change le nombre

de neurones d'une couche à une autre, il faudrait changer toute la machine d'état. Pour soulever ce problème, nous avons refait la conception du module de control. Ainsi, la machine de Moore est remplacée par la machine de Mealy dans laquelle on rajoute un compteur Ayant une valeur générique,  $M$ , et une variable de transition  $Max$ . tel que :

$$\begin{cases} \text{if output\_counter} = M \rightarrow Max = 1 \\ \text{else} \rightarrow Max = 0 \end{cases}$$

La figure V. 8 (b) montre la machine d'état utilisé pour appliquer le **DFR**.

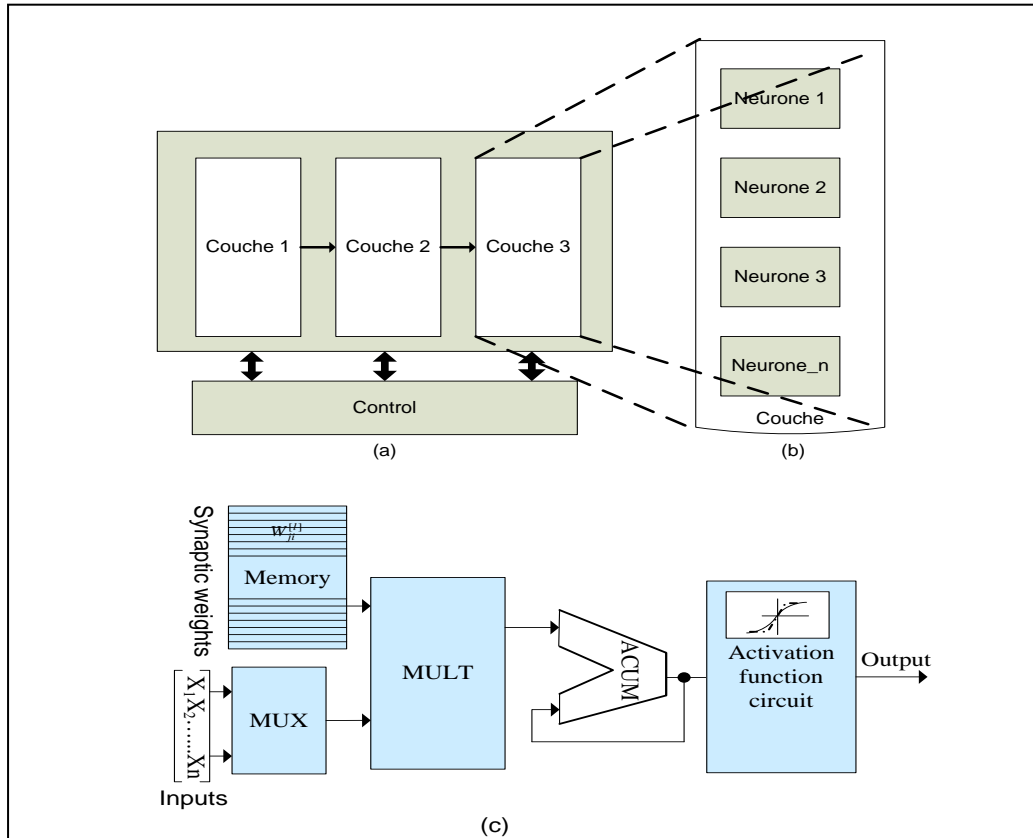


Figure V.7 (a) Structure interne de l'IP ANN. (b) Structure interne de la couche. (c) structure du neurone

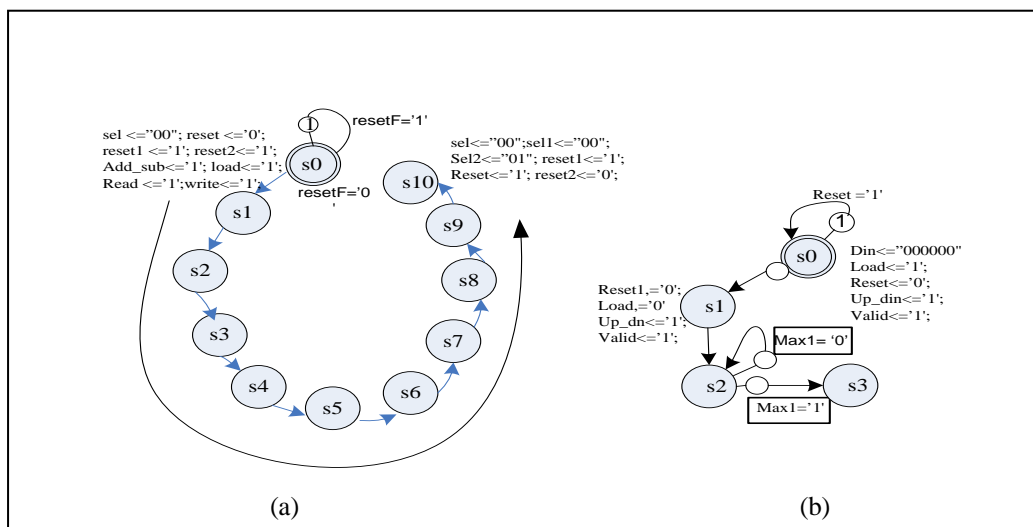


Figure V.8 Machine d'état du module de control de l'IP ANN. (a) Type Moore (b) Type Mealy

**V.4.1.1.2 Sélection de l'IP « On chip training statique »**

L'approche de sélection de l'IP « On chip training statique » est identique à celle de l'IP « off chip ». Néanmoins, la complexité est beaucoup plus élevée dans ce cas. La figure V.9 montre l'IP « on chip training » qui est constitué de deux modules : le module de donnée « data path » et le module de control. Le module « data path » est à son tour constitué de trois sous modules « propagation », « calcul d'erreur » et « mise à jour des poids ». La partie control est constituée du control de chaque sous modules.

Le concepteur/utilisateur peut modifier le nombre de couche de son réseau de neurones, par une simple copie/coller ou bien une suppression du module couche. Il peut aussi modifier le nombre de neurones dans une couche, par une simple copie/coller ou bien suppression du module neurone dans chaque module et chaque sous module.

De même, pour appliquer le DFR, la machine de Moore, présentée dans le chapitre III, a été remplacée par la machine de Mealy.

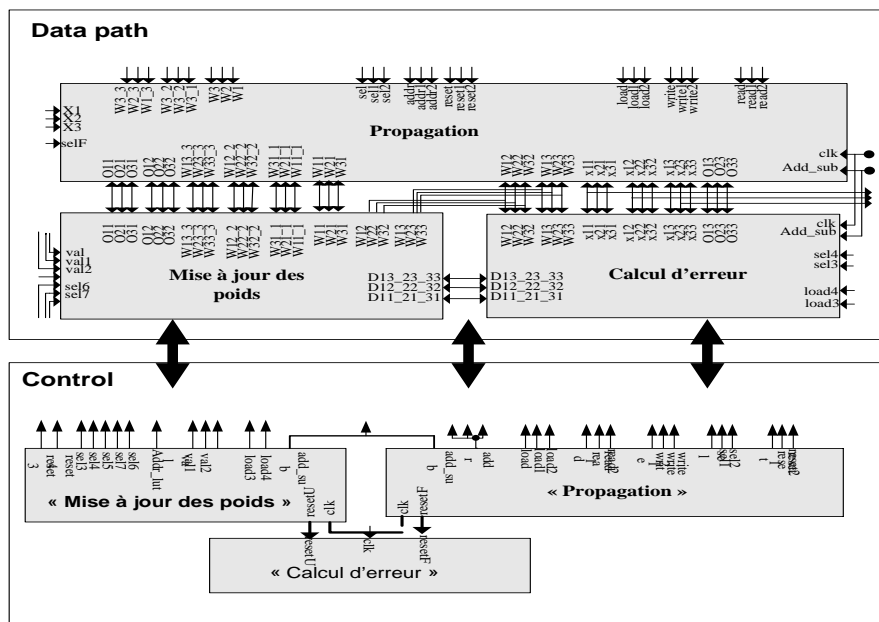


Figure V.9 Structure Externe de l'IP ANN « On chip training statique »

**V.4.1.1.3 Sélection de l'IP « On chip training RTR »**

La reconfiguration dynamique, RTR, qu'elle soit globale ou partielle appelle à considérer quatre niveaux de complexité

1. Conception des différents modules constituant l'algorithme RPG.
2. génération des dessins de masque ou layout des différents modules (placement et routage).
3. Génération des différents fichiers bitstream.
4. Programmation sur FPGA des différents modules.

Les figures V.10 (a) et (b) montrent les étapes suivies pour l'application de la RTR globale et la RTR locale à l'algorithme RPG, respectivement.

La sélection des différents IPs suit une démarche similaire à celle proposée dans les sections précédentes.

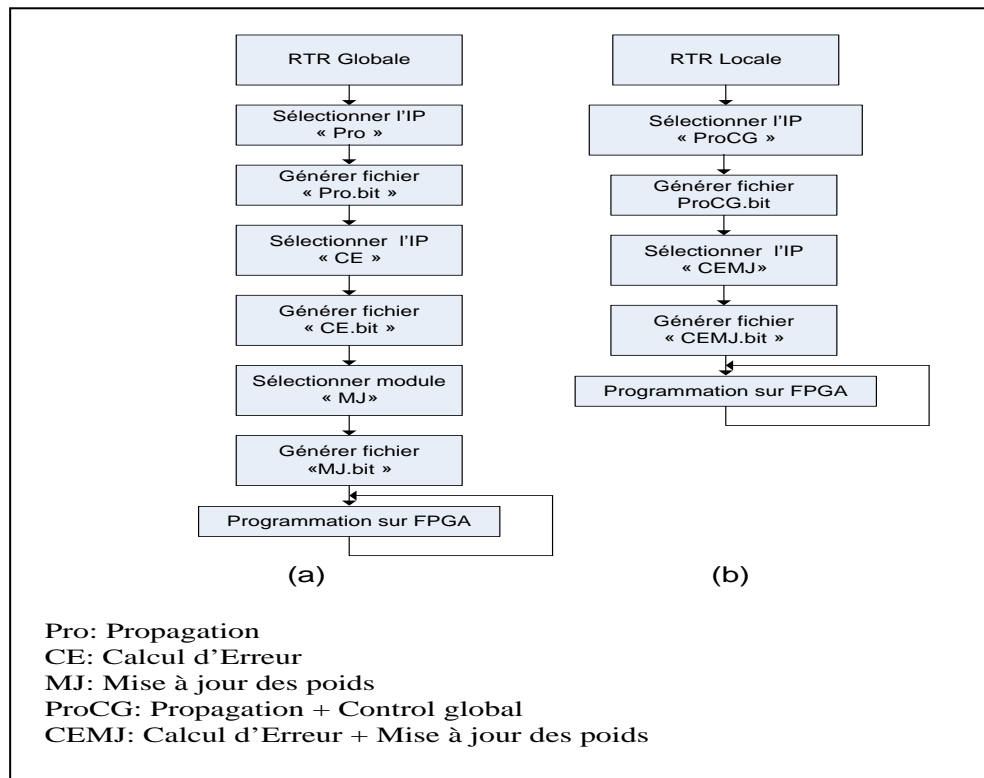


Figure V.10 Etapes pour l'application de la reconfiguration dynamique. (a) RTR globale (b) RTR locale

#### V.4.1.2 Structure du code VHDL de l'algorithme RPG

IL s'agit d'écrire un code VHDL paramétré, accessible au concepteur/utilisateur et dans lequel il peut ajuster les paramètres des différents IPs, afin d'atteindre les spécifications de l'application cible.

De manière générale, les paramètres qui peuvent être instancier sont:

- La précision ou taille du mot,
- La représentation des données
- La taille des mémoires
- La fonctionnalité du core
- La structure de l'architecture

Pour cela, il faut exploiter les caractéristiques de Reuse du langage VHDL à savoir : la genericité, l'héritage, l'encapsulation, les packages, la portabilité, etc.

Par ailleurs, le code VHDL décrit doit tenir compte des règles de codage pour le « reuse ».

##### V.4.1.2.1 Description VHDL paramétrée de l'IP ANN

L'IP réseau de neurone (ANN) est décrit de façon hiérarchique et dans lequel les unités fonctionnelles sont instanciers itérativement selon le paramètre « **GENERIC** » du code VHDL. La figure V.11 montre le pseudo code montrant les paramètres génériques de l'IP ANN.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY ANN IS
  GENERIC (width_inputs: integer := ; -- taille de la donnée entrée
          width_outputs: integer := ; -- taille de la donnée sortie
          num_layers: integer := ; -- nombre des couches
          num_neurons_layer1 := ; -- nombre de neurones dans la 1er couche
          num_neurons_layer2 := ; -- nombre de neurones dans la 2nd couche
          num_neurons_output := ; -- nombre de neurones dans la couche de sortie
          width_weight_layer1 : integer := ; -- taille des poids synaptiques 1ere couche
          width_weight_layer2 : integer := ; -- taille des poids synaptiques 2ème couche);

  PORT ( X: IN std_logic_vector (num_inputs-1 DOWNTO 0);
        WEIGHT_layer1: IN std_logic_vector(width_weight_layer1 -1 down to 0);
        .
        .
        .
        CLK: IN std_logic;
        SEL: IN std_logic;
        RESET: IN std_logic;
        READ: IN std_logic;
        WRITE: IN std_logic
        Outvector: OUT std_logic_vector(num_output -1 down to 0)
        );
End ANN;

```

Figure V.11 Pseudo code de l'IP ANN

Les paramètres de configuration du réseau de neurones sont :

1. Taille des données à l'entrée du réseau
2. Taille des données à la sortie du réseau
3. Nombre de couches
4. Nombre de neurones de la couche d'entrée
5. Nombre de neurones dans la couche cachée
6. Nombre de neurones dans la couche de sortie
7. Taille des poids synaptiques entre la couche cachée et la couche d'entrée
8. Taille des poids synaptiques entre la couche de sortie et la couche cachée

La génération des différentes couches constituant le réseau ANN se fait grâce aux instructions « **GENERATE** » et « **GENERIC MAP** » comme suit (Figure V.12):

```

-- Génération de la couche d'entrée
--
L1: if (i<num_neurons_layer1) generate

  Begin
  Neuron_in: neuron generic map (num_inputs ->num_input_nets,
                                width_weight->width_weights_layer1
                                port map ( ..... ));
  END generate L1;

```

Figure V.12 Pseudo code pour la génération des différentes couches

L'instruction « **GENERATE** » permet la génération automatique des couches et des neurones constituant chaque couche.

Pour une meilleure portabilité du code VHDL, les packages utilisés sont :  
« **ieee\_std\_logic\_1164.all** » pour les signaux logiques  
« **ieee\_std\_logic\_arith.all** » pour les opérations arithmétiques.

Aussi, les règles de codage pour la conception et la synthèse sont pris en considération lors de la description de l'IP ANN.

Ainsi, avec cette description, le concepteur/utilisateur peut facilement modifier la dimension et la configuration du réseau de neurone. La description peut être portée par n'importe quel outil de synthèse vu que le code VHDL utilisé est adapté pour la synthèse de manière générale et non spécifique à un outil particulier.

#### V.4.1.2.2 Description VHDL paramétrée de l'IP neurone

L'IP neurone est décrit de telle sorte à faire ressortir les paramètres génériques. Ceci se fait grâce à l'instruction « **GENERIC** »

La figure V.13 montre le pseudo code décrivant le neurone.

Les paramètres génériques sont :

1. La taille des données du vecteur d'entrée du neurone
2. La taille des poids synaptiques
3. La taille des données du vecteur de sortie du neurone

Dans cette description, les différents composants ou « core » constituant l'architecture du neurone sont instanciers. Chaque « core » possède, lui aussi, des paramètres génériques lui permettant de s'adapter à ses propres besoins. Aussi, il peut être décrit selon plusieurs variétés architecturales. Ainsi en adoptant cette description, le concepteur/utilisateur peut modifier les paramètres du neurone.

Il peut aussi modifier l'architecture du neurone par l'instanciation d'autres composants stockés dans bibliothèque. A titre d'exemple, l'instruction « **For all : LUT\_256 use ENTITY *algorithme.LUT\_256*** », permet d'utiliser l'architecture LUT\_256 qui se trouve dans la bibliothèque *algorithme*.

Notons que la description du neurone a été faite en tenant compte des règles de codage du VHDL.

#### V.4.2 Stratégie de « DWR »

Le « **DWR** » repose sur l'utilisation d'une bibliothèque de composants préalablement conçus, vérifiés et stockés dans une bibliothèque. La figure V.14 montre la structure de la bibliothèque. Cette dernière est composée des différents IP constituant le neurone.

Pour la construction de la bibliothèque nous avons opté pour l'utilisation des IPs de Xilinx « *CoreGenerator* ».

« *CoreGenerator* » est une bibliothèque de composants paramétrés et optimisés pour être intégré dans les circuits FPGA de Xilinx. La bibliothèque couvre une large gamme d'IPs incluant les fonctions logiques élémentaires, les circuits DSP, les mémoires, les fonctions mathématiques et d'autres IP dédiés à des applications industriels, etc. Une documentation détaillée (Datasheet, guide d'utilisateur, etc.) est fournie pour chaque IP. La bibliothèque *CoreGenerator* est intégrée avec l'outil ISE Fondation ou bien l'outil Mentor Graphics.

Dans l'outil ISE fondation, une interface graphique est offerte à l'utilisateur afin de régler les paramètres de chaque IP. Dans ce qui suit les paramètres de chaque IP constituant le neurone seront décrits :

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY Neurone IS
  GENERIC( WIDTH_INPUTS : integer :=
           WIDTH_OUTPUTS: integer :=
           WIDTH_WEIGHTS: integer := );
  PORT(
    entrée      : IN      std_logic_vector (WIDTH_INPUTS -1 DOWNT0 0);
    poids       : IN      std_logic_vector (WIDTH_WEIGHTS -1 DOWNT0 0);
    add_sub     : IN      std_logic;
    clk         : IN      std_logic;
    load        : IN      std_logic;
    read        : IN      std_logic;
    rst         : IN      std_logic;
    write       : IN      std_logic;
    sortie      : OUT     std_logic_vector (WIDTH_OUTPUTS -1 DOWNT0 0);
    w11         : OUT     std_logic_vector (WIDTH_WEIGHTS -1 DOWNT0 0);
    x11         : OUT     std_logic_vector (WIDTH_OUTPUTS -1 DOWNT0 0)
  );
ARCHITECTURE STRUCTURELLE OF Neurone IS

  -- Component Declarations

  COMPONENT LUT_256 -- instantiation du Core LUT
  GENERIC (WIDTH_ADDR : integer :=
          WIDTH_LUT   : integer := );
  PORT (
    addr_lut : IN      std_logic_vector (WIDTH_ADDR -1 DOWNT0 0);
    dout     : OUT     std_logic_vector (WIDTH_LUT -1 DOWNT0 0)
  );
  END COMPONENT;

  COMPONENT RAM -- Instantiation du Core RAM
  GENERIC (WIDTH_DATA_RAM : integer := );
  PORT (
    read : IN      std_logic ;
    write : IN     std_logic ;
    din  : IN     std_logic_vector (WIDTH_DATA_RAM -1 DOWNT0 0);
    dout : OUT     std_logic_vector (WIDTH_DATA_RAM -1 DOWNT0 0)
  );
  END COMPONENT;

  COMPONENT accumulateur - Instantiation du Core Accumulateur
  GENERIC ( WIDTH_INPUT : integer :=
           WIDTH_OUTPUT: integer := );
  PORT (
    add_sub : IN      std_logic ;
    clk     : IN      std_logic ;
    din     : IN     std_logic_vector (WIDTH_INPUT -1 DOWNT0 0);
    load    : IN     std_logic ;
    rst     : IN     std_logic ;
    dout    : OUT     std_logic_vector (WIDTH_OUTPUT -1 DOWNT0 0)
  );
  END COMPONENT;

  COMPONENT multiplieur --- Instantiation du core Multiplieure
  GENERIC (WIDTH_INPUT0 : integer :=
          WIDTH_INPUT1 : integer :=
          WIDTH_OUTPUT  : integer := );
  PORT (
    din0 : IN      std_logic_vector (WIDTH_INPUT0 -1 DOWNT0 0);
    din1 : IN     std_logic_vector (WIDTH_INPUT1 -1 DOWNT0 0);
    prod : OUT     std_logic_vector (WIDTH_OUTPUT -1 DOWNT0 0)
  );
  END COMPONENT;

  -- Optional embedded configurations
  -- pragma synthesis_off
  FOR ALL : LUT_256 USE ENTITY Algorithmme.LUT_256;
  FOR ALL : RAM USE ENTITY Algorithmme.RAM;
  FOR ALL : accumulateur USE ENTITY Algorithmme.accumulateur;
  FOR ALL : multiplieure USE ENTITY Algorithmme.multiplieure;
  Pragma synthesis_on
Begin
  -- Port mapping des différents Cores
  |
END;

```

Figure V.13 Pseudo code de l'IP neurone.



### V.4.2.1 Le multiplieur

Plusieurs interfaces sont utilisées pour régler les paramètres du multiplieur, à savoir :

1. Le *type du multiplieur*
2. La *construction du multiplieur*
3. Les *paramètres d'entrée*
4. Les *paramètres de sortie*
5. les *paramètres du pipeline*

- Le *type du multiplieur* est utilisé pour désigner les différentes stratégies de multiplication à savoir : la multiplication parallèle, la multiplication série et la multiplication avec un coefficient constant.

- Le *multiplieur parallèle* prend deux entrées A et B et génère la multiplication de ces deux valeurs en parallèle.
- Dans le *multiplieur séquentiel*, la multiplication est réalisée en effectuant la somme des produits partiels.
- Le *multiplieur à coefficients constants* prend une valeur A quelconque et la multiplie avec une constante définie par l'utilisateur.

- La construction du multiplieur désigne comment le multiplieur sera mappé dans le circuit FPGA. Pour cela trois possibilités sont offertes :

- Construction du multiplieur à base de *look up tables* « LUT »
- Construction à partir des *blocs de multiplieurs (18x18)* dans le cas des circuits FPGA VIRTEX-II et VIRTEX-II PRO.
- Construction à partir des *blocs Xstrem-DSP* dans le cas des circuits Virtex-4

- Les paramètres d'entrées et de sorties concernent :

- La *taille de la donnée* qui varie entre 2bits et 64 bits,
- Le *type de la donnée* qui peut être signé ou non signé.

- Le pipeline peut être sélectionné au *maximum* ou bien au *minimum* afin d'ajuster la latence du circuit.

### V.4.2.2 L'accumulateur

Les paramètres de configuration de l'accumulateur sont

- Le mode d'opération : Addition, Soustraction ou bien Add/Sous
- Paramètres d'entrée : taille des données et type du signal
- Paramètres de sortie : taille de la donnée et type du signal
- Option de synthèse et placement – routage

### V.4.2.4 Le circuit mémoire

Le circuit mémoire peut être représenté par des mémoire RAM ou bien ROM en fonction de l'implémentation « on-chip training » ou bien « off chip training ». Les paramètres de la mémoire sont :

- Mode de configuration : ECRITURE ou bien LECTURE
- Taille de la mémoire : Taille des données et profondeur de la mémoire
- Mode d'écriture : LECTURE APRES ECRITURE, LECTURE AVANT ECRITURE, PAS de LECTURE
- Options de synthèse et d'implémentation : optimisation de la surface, etc.

#### V.4.2.5 La fonction d'activation

Elle peut être implémentée en utilisant une LUT ou bien une approximation de fonction linéaire. Si la fonction d'activation est implémentée à base d'une LUT, les paramètres sont les mêmes que ceux du circuit mémoire.

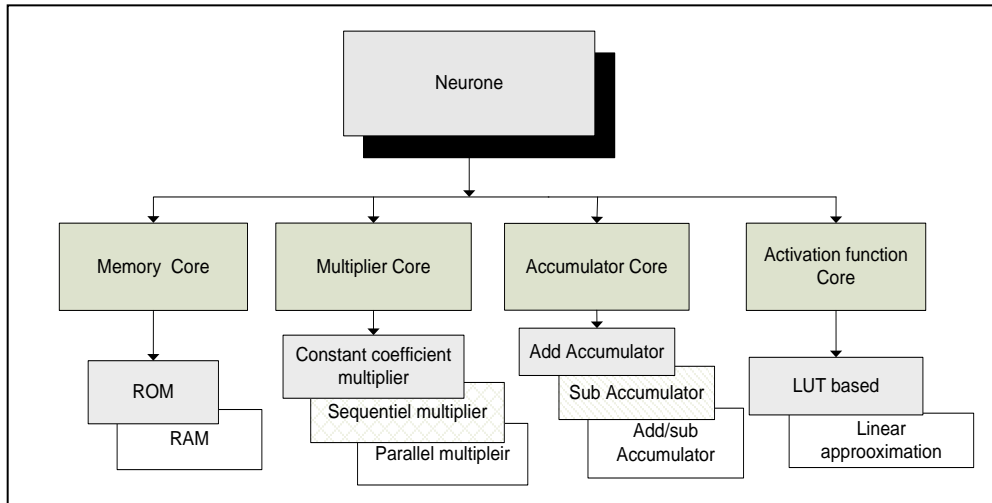


Figure V.14 Structure de la bibliothèque des Cores constituant l'IP neurone.

#### V.5 Estimation du coût du « Design Reuse »

Dans cette section une évaluation du coût du « design Reuse » appliqué à l'algorithme RPG est présentée. Par la suite, une étude comparative est établie entre une conception initiale utilisant un **modèle fonctionnel** (voir modèle en couche, section V.2.2) du réseau de neurone et celle en appliquant le concept de « design Reuse ».

Selon les travaux de recherches publiés dans [196], un modèle mathématique a été établi pour estimer le temps de conception d'un circuit ASIC ou bien FPGA.

Le temps de conception,  $Temps\_concep$ , est défini par

$$Temps\_concep = \sum_{i=1}^6 FPGA\_param(i) \quad (V.1)$$

Où,  $FPGA\_param(i)$  désigne les différentes étapes qui rentrent en considération lors de la conception sur circuit FPGA, à savoir :

1. L'apprentissage
2. La modélisation VHDL
3. La simulation et la génération de vecteurs de tests
4. La synthèse
5. Le placement et le routage
6. La documentation

Afin d'évaluer le  $Temps\_concep$ , nous avons d'abord pris comme exemple, le circuit XOR (2,2,1), et nous avons évalué les temps des différentes étapes de conception du réseau XOR.

Afin de rendre la comparaison simple, une normalisation a été faite. Ainsi les temps sont exprimés en % et l'implémentation « off chip training » est prise comme référence

(100% du temps de conception), les autres types d'implémentations sont exprimées comme multiple de celle-ci.

Le tableau V.1 donne le coût du design **fonctionnel** dans les approches « off chip training », « on chip training statique » et « on chip training - RTR ». L'estimation du temps d'une conception « on chip training statique » est de 176% alors que celui d'une conception « on chip RTR » est de 180%.

Tableau V.1 Estimation du coût d'une conception avant application du Reuse

FPGA_param	Off chip training	On chip training	RTR
Modélisation VHDL	42	80	80
Simulation	34	50	50
Synthèse	4	8	8
Placement & Routage	4	8	12
Documentation	16	30	30
<i>Temps_concep</i>	100%	176%	180%

Le tableau V.2 donne une estimation du coût de la conception de l'algorithme RPG en appliquant le concept de Reuse. Le temps de conception total est estimé à 168% pour une conception « off chip training », 256% pour une conception « on chip statique » et 260% pour une conception « on chip- RTR ».

Tableau V.2 Estimation du coût de la conception de l'algorithme RPG en utilisant le concept de Reuse

FPGA_param	Off chip training	On chip training	RTR
Modélisation VHDL	80	120	120
Simulation	50	70	70
Synthèse	4	8	8
Placement & Routage	4	8	12
Documentation	30	50	50
<i>Temps_concep</i>	168 %	256 %	260 %

Le tableau V.3 donne le temps de conception après application du design Reuse : design with reuse « DWR »

Tableau V.3 Estimation du temps DWR

FPGA_param	Off chip training	On chip training	RTR
<i>Temps_concep</i>	0.5 %	2.08 %	2.5 %

Afin de montrer l'intérêt du « *desig reuse* », nous avons calculé le temps après « *N designs* » fonctionnel et celui obtenu après « *N reuse* ». La figure V.15 montre les résultats obtenus pour l'algorithme RPG « off chip training ».

La figure V.15 montre que le temps d'une conception selon le modèle fonctionnel est inférieur à celui en appliquant le « design reuse ». Néanmoins, la réutilisation de celui ci, nécessite une réécriture du code.

Lorsqu'on applique le « design reuse », on perd du temps dans la conception initiale, cependant la réutilisation du code permet de gagner du temps de façon dramatique.

Les mêmes résultats sont obtenus pour les implémentations « On chip training statique » et « On chip training RTR », comme montré dans les figures V1.16 et V.17 respectivement.

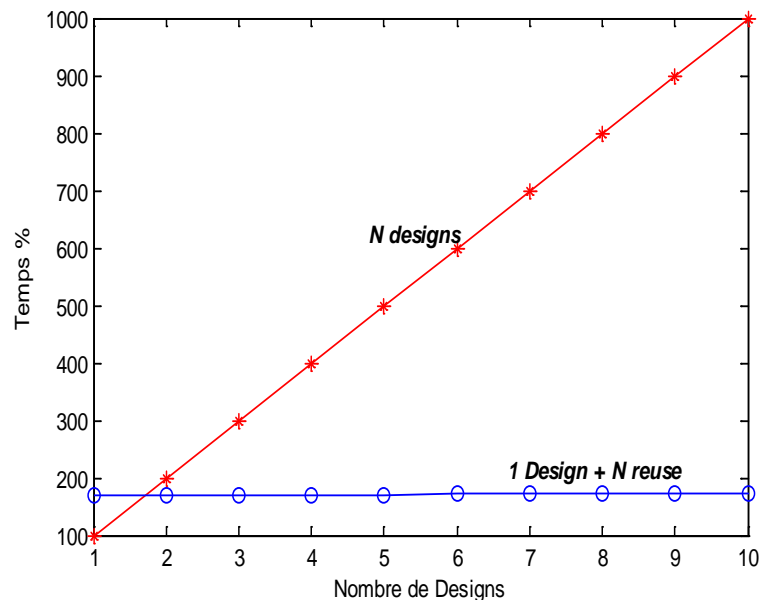


Figure V.15 Estimation du temps de conception pour l'algorithme RPG « off chip training »

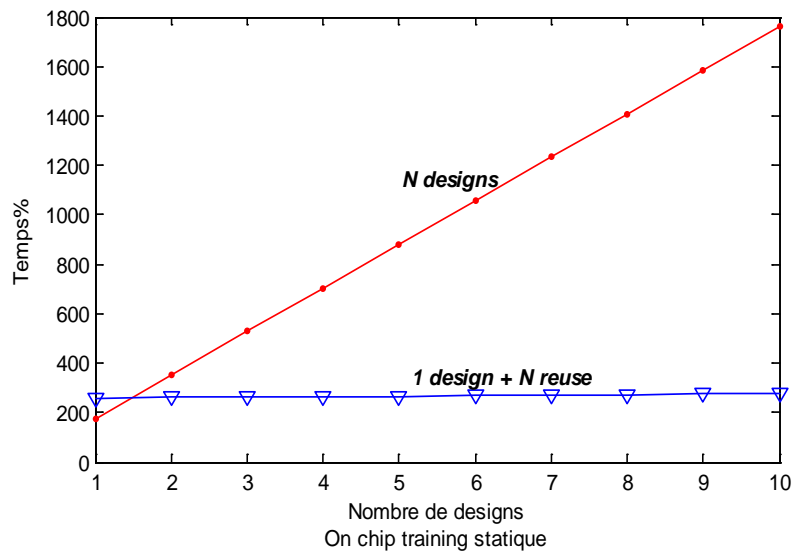


Figure V.16 Estimation du temps de conception pour l'algorithme RPG « On chip training statique »

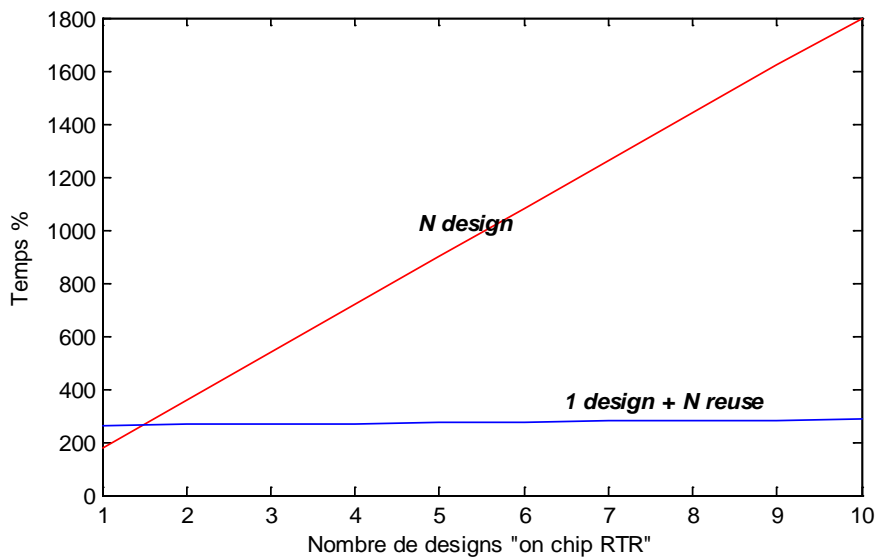


Figure V.17 Estimation du temps de conception pour l'algorithme RPG « on chip training - RTR »

La dernière étude consiste à évaluer le Reuse en fonction de la complexité du réseau de neurones. Pour cela, nous avons considérés trois autres réseaux de neurones : un réseau (3,3,3) un réseau (16,16,16) et un réseau (16,48,64). Le tableau V.4 donne les pourcentages du *Temps-concep*, pour chaque type de réseau.

La figure V.18 montre l'évolution du Temps de conception en fonction du nombre de neurones. Même en appliquant le concept de Design reuse , le temps de conception augmente avec la complexité du réseau. Néanmoins cette augmentation reste toujours inférieure à celle obtenue en utilisant un modèle de conception fonctionnel ou maintenable d'où l'intérêt de l'application du « Design reuse ».

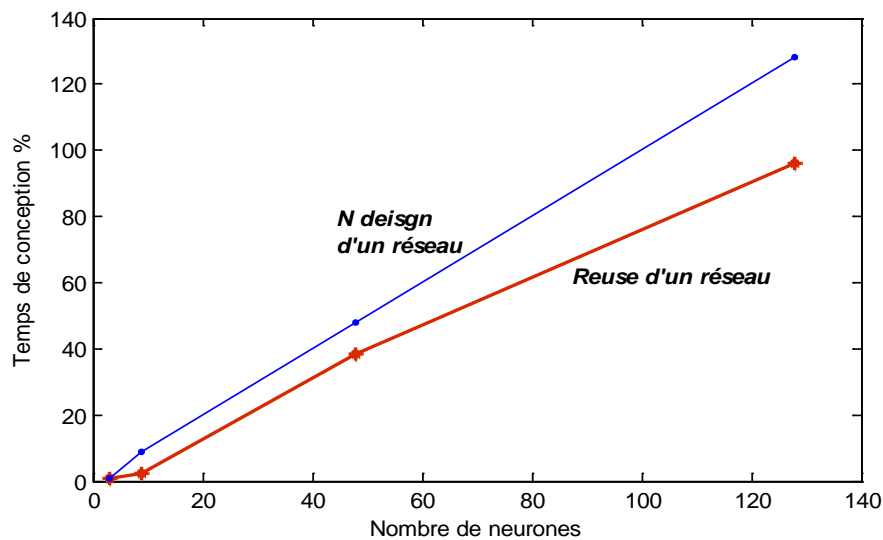


Figure V.18 Evolution du temps de conception en fonction du nombre de neurones

## V.6 Evaluation de la qualité de l'IP\_ANN

Dans cette section, on s'intéresse à l'évaluation de l'IP\_ANN. Pour cela, nous utilisons l'outil connu sous le nom OpenMore et développé par la compagnie « Mentor Graphics » en collaboration avec la compagnie « Synopsys » [197].

OpenMore est un programme conçu pour permettre aux concepteurs d'auto évaluer la qualité du Reuse dans les IPs afin de pouvoir les commercialiser.

Le programme est un fichier Excel contenant les règles pour le codage des IPs et qui sont publiées dans le livre RMM deuxième édition. Les règles sont groupées en trois catégories :

1. La conception « Macro Design Guidelines »
2. La vérification « Macro Verification Guidelines »
3. La livraison « Deliverable Guidelines »

Chaque catégorie est pondérée pour donner un score total de 100%. Dans la catégorie conception, pondérée à 50%, sont présentées les règles qui permettent de concevoir un IP en le rendant réutilisable. Dans la catégorie vérification, sont présentées les règles et guidelines, permettant de simuler l'IP et de générer des fichiers « test-bench » réutilisables. Cette catégorie est pondérée à 35%.

La dernière catégorie contient des guideline pour la documentation et l'emballage de l'IP. Cette catégorie est pondérée à 15%.

Quatre critères sont utilisés pour évaluer une règle ou bien un guideline :

1. Le type
2. L'évaluation « Assessment »
3. le Score « Unweighted score »
4. le Maximum Score « Max Score »

Le type désigne une règle ou bien un guideline. Un score de 5 points est attribué à une règle et un score de 1 point est attribué à un guideline.

Dans la colonne « Assessment », et selon l'utilisation les points suivant sont obtenus

A	= Always	= 2 points
S	= Sometimes	= 1 point
N	= Never	= 0 points
NA	= Not Applicable	= 0 points

Le Score « Unweighted Score » est obtenu en multipliant le score attribué à la règle par le score attribué à l'évaluation.

Le score "Max Score" d'une règle est de 10 points, et celui d'un Guideline est de 2 points.

Les tableaux V.4 (a), (b), (c) et (d) montrent les résultats obtenus lors de l'évaluation de la conception, de la vérification et de la documentation respectivement.

Les résultats obtenus sont les suivants :

# Un score de 329/450 pour les « *Macro design Guidelines* » correspondant à 73%

# Un score de 23/44 pour les « *Verification Guidelines* » correspondant à 52%

# Un score de 40/70 pour les « *Delivrables Guidelines* » correspondant à 57%

#Un score de 33/70 pour « *FPGA Addendum* » correspondant à 47%

Enfin, un score moyen pondéré « Total Weighted Assessment » de 62% est obtenu pour l'évaluation de l'IP ANN (voir tableau V.4.d).

FPGA + OpenMore Assessment Program					
	Portions Copyright 1999-2000 by Synopsys Inc., Mentor Graphics Corp., and Xilinx Inc.				
	Assessment" and "FPGA Soft IP Assessment"				
IP Name:	IP_ANN				
Company:					
Contact:	nizeboudjen@cdta.dz				
RMM2 Section		Assessment	Unweighted Score	Max Score	Comment
I	Macro Design Guidelines		329	450	<i>weighted 50% of total</i>
3	<i>System-Level Design Issues: Rules and Tools</i>		29	46	
3.2	Design for Timing Closure: Logic Design Issues	A	22	22	
3.2.3	Clocking	A	12	12	
3.2.4	Reset	A	10	10	
3.4	Design for Verification: Verification Strategy		7	22	
3.5	System Interconnect and On-Chip Buses	NA	0	2	
5	<i>RTL Coding Guidelines</i>		221	300	
5.2	Basic Coding Practices	A	140	170	
5.2.1	General Naming Conventions	A	42	42	
5.2.3	Architecture Naming Conventions		2	2	
5.2.4	Headers in Source Files	A	10	10	
5.2.5	Use Comments	A	11	12	
5.2.6	Keep Commands on Separate Lines	A	10	10	

Tableau V.(a) ANN-Open More Assessment program



Tableau V.4 (b) ANN-Open More Assessment program (suite)

RMM2 Section	Type	Assessment	Unweighted Score	Max Score	Comment
5.2.7	Line length		1	2	
5.2.8	Indentation		6	12	
5.2.9	HDL Reserved Words not used in HDL description	A	10	10	
5.2.10	Port Ordering		14	16	
5.2.11	Port Maps and Generic Maps		11	12	
5.2.12	VHDL Entity, Architecture, and Configuration Sections		2	2	
5.2.13	Use Functions		0	2	
5.2.14	Use Loops and Arrays	S	3	4	
5.2.15	Use Meaningful Labels		18	34	
5.3	Coding for Portability		36	40	
5.4	Guidelines for Clocks and Resets		9	34	
5.4.1	Avoid Mixed Clock Edges	A	2	24	
5.4.3	Avoid Gated Clocks		2	2	
5.4.4	No Internally Generated Clocks	S	1	2	
5.4.5	Gated Clocks and Low Power Designs		0	2	
5.4.6	Avoid Internally Generated Resets	A	4	4	
5.5	Coding for Synthesis		27	38	
5.5.1	Infer Registers	A	1	2	
5.5.2	Avoid Latches	A	12	12	

5.5.4	Avoid Combinational Feedback	S	1	2
5.5.5	Specify Complete Sensitivity Lists	S	7	12

RMM2 Section	Type	Assessment	Unweighted Score	Max Score	Comment
5.5.6	Blocking and Nonblocking Assignments (Verilog)	NA	0	0	
5.5.8	Case Statements versus if-then-else Statements	A	2	2	
5.5.9	Coding State Machines	A	2	6	
5.6	Partitioning for Synthesis		8	16	
5.6.1	Register All Outputs	A	2	2	
5.6.2	Related Combinational Logic in a Single Module		1	2	
5.6.3	Separate Modules That Have Different Design Goals	A	1	2	
5.6.4	Asynchronous Logic		2	4	
5.6.5	Arithmetic Operators: Merging resources		0	2	
5.6.7	Avoid Point-to-Point Exceptions and False Paths	N	2	4	
5.8	Code Profiling		1	2	
6	<i>Macro Synthesis Guidelines</i>		79	104	
6.2	Macro Synthesis Strategy		12	22	
6.2.1	Macro Timing Budget		5	10	
6.2.2	Subblock Timing Budget	A	5	10	
6.2.4	Subblock Synthesis Process - Three Phases		2	2	
6.2.7	Preserve Clock and Reset Networks		0	0	
6.5	Coding Guidelines for Synthesis Scripts		67	82	

Tableau V.4 © ANN-Open More Assessment program (suite)

Tableau V.4 (d) ANN-Open More Assessment program (suite)

II	Verification Guidelines			23	44	<i>weighted 35% of total</i>
III	Deliverable Guidelines			40	70	<i>weighted 15% of total</i>
RMM2 Section		Type	Assessment	Unweighted Score	Max Score	Comment
9	<i>RMM Deliverables</i>			40	70	
9.1.1	Soft Macro Deliverables			40	70	
9.1.1.2	Verification Files			5	10	
9.1.1.3	Documentation			10	20	
9.1.1.4	System Integration Files			0	0	
IV	FPGA Addendum			37	66	<i>weighted as part of Section I</i>
15.2.1.2	Synchronous vs. Asynchronous Design Style			10	10	
15.2.1.3	System Clocking and Clock Distribution			10	20	
15.2.2.1.2	On-Chip Memory			1	4	
15.2.3	External Operability (I/O Standards)			11	12	
15.2.4	Module (Macro) Implementation			2	6	
15.3.1.3	Finite State Machines			10	10	
15.3.1.4	Pipelining			1	2	
15.3.2	Case and IF-Then-Else			2	2	
	Total Unweighted Assessment			429	630	
	Total Weighted Assessment			62%		

## V.7 Conclusion

Dans ce chapitre, nous avons montré que le concept de « *Design Reuse* » permet de réduire le temps de conception, facteur essentiel pour la réduction du « time to market » et l'amélioration de la productivité.

Nous avons proposé une méthodologie de conception descendante et planifiée des réseaux de neurones et dans laquelle les règles de conception pour la réutilisation sont appliquées.

L'approche a été appliquée à l'algorithme (RPG), avec ces trois variantes : « Off chip training », « On chip training statique » et « On chip training RTR ».

Les architectures proposées sont basées sur une conception modulaire et hiérarchique des différents blocs.

Afin de mieux exploiter le concept de réutilisation, l'approche proposée repose sur l'application du « **DFR** » pour la génération de l'architecture du réseau de neurones et sur le « **DWR** » pour l'exploitation de la bibliothèque des noyaux prédéfinis. Pour cela, un choix judicieux des outils de conception et de langage a été fait.

L'application de l'approche « **DFR** » est très coûteuse en temps : 1.68 fois pour l'implémentation « *Off chip training* », 2.56 fois pour l'implémentation « On chip training statique » et 2.6 fois pour l'implémentation « *On chip training -RTR* ».

Néanmoins après réutilisation (**DWR**), le temps est réduit de manière extraordinaire : 0.5% pour l'implémentation « *Off chip training* », 2.08% pour l'implémentation « On *chip training statique* » et 2.8% pour l'implémentation « *On chip training-RTR* ».

Aussi, nous avons montré l'importance de l'application du Reuse lorsque le nombre de neurone (i.e, la dimension du réseau) augmente.

Afin de pouvoir vérifier la qualité de l'IP\_ANN, nous avons utilisé l'outil **OpenMore**. Ce choix est justifié par le fait que c'est une référence pour les concepteurs et industriels des IPs et aussi il est téléchargeable gratuitement.

L'application de ces règles à l'algorithme RPG, a donné un score moyen de 62%. D'après l'outil OpenMore, un score supérieur à 60% signifie que l'IP est bon pour être réutilisable. Donc, nous pouvons conclure que le code de l'IP\_ANN est réutilisable.

Notons enfin que ce score à été majoré par la section « *Macro Design Guidelines* » et dans laquelle un score de 73% a été obtenu. Par conséquent, des efforts supplémentaires de conception doivent être fournis pour satisfaire les règles de codage VHDL des sections « *Verification Guidelines* » et « *Delivrables Guidelines* ».

---

# *CONCLUSION GENERALE*

---

Le travail effectué dans le cadre de cette thèse se rapporte à l'implémentation sur FPGA des réseaux de neurones, plus particulièrement l'algorithme de la rétropropagation du gradient (RPG). Dans un premier lieu une étude sur l'état de l'art de l'implémentation hardware des réseaux de neurones nous a permis de ressortir les points critiques suivants :

1. L'implémentation hardware des réseaux de neurones est justifiée par la rapidité de traitement qui peut être atteinte en utilisant des architectures massivement parallèles.
2. Le parallélisme et la flexibilité sont des paramètres antagonistes, une solution idéale serait de réaliser un compromis entre ces deux paramètres
3. La classification du hardware neuronal est un problème qui est toujours posé. Pour notre part, nous avons recensé treize approches de classification pluri temporelles. A la suite de l'examen de ces dernières, nous avons proposé une approche qui prend en charge d'une part les éléments consensuels des autres approches et d'autre part, l'évolution de la technologie VLSI et des approches et techniques de conception.
4. Lors de la conception du hardware neuronal, un certain nombre de problèmes doivent être pris en charge par le concepteur. Parmi ces problèmes : Le parallélisme, La flexibilité, Les performances, la précision, l'apprentissage, la technologie FPGA, etc.

A travers une étude comparative entre le langage VHDL et le langage Handel-C, nous avons montré qu'en terme de description du circuit, le Handel-C est nettement plus simple et plus rapide à décrire que le langage VHDL. Nous avons montré aussi que le mapping des langages VHDL et Handel-C en hardware permet d'obtenir des implémentations ayant le même ordre de grandeur des fréquences et du temps de réponse. Néanmoins, en termes de surface, une description Handel-C résulte en une surface trois fois plus grande que celle d'une description en VHDL. Ceci nous a conduit à proposer d'adopter le langage Handel-C pour étudier les différents types de parallélisme de l'algorithme RPG afin de choisir l'architecture parallèle la plus appropriée pour une implémentation sur FPGA.

L'étude comparative entre le parallélisme des neurones et celui des synapses montre qu'il est plus judicieux d'utiliser le parallélisme des neurones, car il occupe moins de ressources du circuit FPGA. Cette étude nous a permis de se fixer sur l'architecture générale de l'algorithme RPG qui est basée sur le parallélisme des neurones. Nous avons proposé une architecture simple, régulière, parallèle et pouvant prendre les deux type d'implémentation « *on chip training* » et « *off chip training* ».

Les performances de l'architecture proposée dépendent du type du multiplieur. Nous avons montré que le multiplieur parallèle de CoreGenerator offre le meilleur compromis entre la surface et le temps d'exécution, comparé aux autres multiplieurs (multiplieur série, multiplieur série pipeline, multiplieur de Booth, multiplieur parallèle pipeline et le multiplieur en virgule flottante).

Plus la dimension du réseau augmente, plus il devient difficile d'implémenter tout l'algorithme RPG. Afin d'augmenter la dimension du réseau à implémenter, nous avons proposé d'utiliser la reprogrammabilité dynamique des circuits FPGA.

Nous avons montré qu'il est possible d'augmenter la densité d'intégration des réseaux de neurones à condition de prendre en considération le temps de reconfiguration dynamique.

L'étude comparative des trois approches proposées a fait ressortir les points suivants :

- La RTR globale permet le meilleur gain en terme de surface avec un taux de 478%, comparé au gain de la RTR locale (188%) et à celui de la configuration statique (%Imax=0) (tableau IV.7).
- La RTR locale permet le meilleur gain en temps de reconfiguration. Ce dernier atteint un taux de 90% comparé au gain de la configuration statique (66%) et à celui de la reconfiguration globale (0%) (Voir tableau IV.7).
- En comparant notre approche (REC-ANN) avec RTR-MANN et RRANN-II, nous pouvons conclure les points suivants :
- REC-ANN offre des meilleures performances en termes de gain en densité fonctionnelle (%GD), par contre l'approche RRANN-II offre le meilleur gain en temps de reconfiguration. Ce résultat est logique du fait d'une part, le circuit XC3000 utilisé dans RRANN-II possède des temps de configuration et reconfiguration inférieurs à ceux du circuits XC2V1000 utilisé par notre approche, et d'autre part les ressources logiques du circuit XC2V100 sont beaucoup plus importantes que celles du circuit XC3000 et XCV2000-E.
- Comparé aux travaux de la littérature, nous avons montré que notre approche, REC-ANN, offre de meilleures performances en terme de densité fonctionnelle. En ce qui concerne le gain du temps de reconfiguration, nous pouvons conclure que les résultats sont similaires puisque la différence est de l'ordre de 4% uniquement.

En ce qui concerne la mise en oeuvre de la reconfiguration dynamique, nous avons appliqué l'approche de conception modulaire. Les différentes étapes de cette dernière sont réalisées manuellement moyennant les différents modules de l'outil ISE foundation. Néanmoins, cette approche est longue et nécessite une expertise dans la conception pour sa mise en oeuvre.

Un autre moyen d'augmenter la flexibilité est de concevoir des IPs réutilisables. Avec l'évolution de la technologie, les réseaux de neurones peuvent être intégrés dans des systèmes sur puce et par voie de conséquence, l'application du « *Design Reuse* » est une des directives essentielles pour diminuer le problème du « design gap ».

Afin de mieux exploiter le concept de réutilisation, l'approche proposée repose sur l'application du « **DFR** » pour la génération de l'architecture du réseau de neurones et sur le « **DWR** » pour l'exploitation de la bibliothèque des Cores prédéfinis. Pour cela, un choix judicieux des outils de conception et de langage a été fait. L'application de l'approche « **DFR** » est très coûteuse en temps : 1.68 fois pour l'implémentation « *Off chip training* », 2.56 fois pour l'implémentation « *On chip training statique* » et 2.6 fois pour l'implémentation « *On chip training -RTR* ». Néanmoins après réutilisation (**DWR**), le temps est réduit de manière extraordinaire : 0.5% pour l'implémentation « *Off chip training* », 2.08% pour l'implémentation « *On chip training statique* » et 2.8% pour l'implémentation

« *On chip training-RTR* ». Aussi, nous avons montré l'importance de l'application du « *Design Reuse* » lorsque le nombre de neurone (i.e. la dimension du réseau) augmente.

Afin de pouvoir vérifier la qualité de l'IP- ANN, nous avons utilisé l'outil **OpenMore**. L'application de ses règles à l'algorithme RPG, a donné un score moyen de 62%. D'après l'outil OpenMore, un score supérieur à 60% signifie que l'IP est bon pour être réutilisable. Donc, nous pouvons conclure que le code de l'IP ANN est réutilisable. Notons que ce score a été majoré par la section « *Macro Design Guidelines* » et dans laquelle un score de 73% a été obtenu. Par conséquent, des efforts supplémentaires de conception doivent être fournis pour satisfaire les règles de codage VHDL des sections « *Verification Guidelines* » et « *Delivrables Guidelines* ».

Notons enfin que la méthodologie proposée constitue l'épine dorsale pour la construction d'une plateforme permettant de regrouper l'ensemble des techniques et moyens liés à l'implémentation sur FPGA de l'algorithme RPG.

En perspective les améliorations suivantes doivent être effectuées afin de répondre pleinement à l'objectif ciblé :

1. En ce qui concerne l'implémentation sur FPGA de l'algorithme RPG, il serait intéressant d'étudier les différentes possibilités d'implémentation de la fonction d'activation et son influence sur les performances du circuit. Ce travail sera exploité dans l'enrichissement de la bibliothèque des IPs pour la construction du réseau de neurones.
2. Une deuxième perspective concerne la mise en œuvre et l'exploitation de la reconfiguration dynamique. En effet, le développement d'un outil permettant d'automatiser les différentes étapes de conception modulaire, constitue un axe de recherche porteur. Aussi, le choix de la famille du circuit FPGA et de la carte de prototypage pour valider les performances obtenues et comparer entre la théorie et la pratique, est un point crucial sans lequel la reconfiguration dynamique reste seulement une possibilité théorique des circuits FPGAs, tel le cas aujourd'hui.
3. Une troisième perspective concerne le concept du « Design Reuse ». Nous avons vu que la validation des IPs nécessite l'application d'un certain nombre de règles qui concernent principalement la conception, la vérification et la documentation. Actuellement l'outil **OpenMore** permet cette validation de façon manuelle et beaucoup de temps est perdu dans ce sens. Il serait donc intéressant d'automatiser ces étapes.
4. Une quatrième perspective concerne l'enrichissement de la bibliothèque des IPs élémentaires, par d'autres types de multiplieurs, de mémoires, accumulateurs, et fonctions d'approximation ; ainsi que l'extension de l'approche proposée à d'autres algorithmes ; à titre d'exemple les algorithmes de KOHONEN et de HOPFIELD.
5. Une cinquième perspective serait la prise en charge du problème de la consommation des différents IPs.

Notons enfin que la mise en œuvre de la plateforme, nécessite l'exécution automatique des outils déployés à cet effet, et le développement d'une interface graphique conviviale.



---

# *REFERENCES BIBLIOGRAPHIQUES*

---

## REFERENCES BIBLIOGRAPHIQUES

- [1] D. Caviglia, "NeuroNetRoadmap", <http://www.image.ece.ntua.gr/old/neuronet/> (last accessed 9/10/2013)
- [2] M. De Camps, A.C. Knoll, M. Kamps "A Roadmap for NeuroIT Challenges for the Next Decade", IEEE Engineering in Medicine and Biology Society, pp. 42-46, Vol. 26, issue 3, 2007
- [3] S.Haykin, "Neural Networks: A Comprehensive Foundation (2nd Edition)", Prentice Hall, 2005.
- [4] R.P. Lippman, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, pp. 4-22 Avril, 1987.
- [5] A. Tixier- Vidal, "De la théorie cellulaire à la théorie neuronale", Biologie Aujourd'hui, 204(4), pp. 253-266 (2010), DOI: 10.1051/jbio/2010015
- [6] S. Guillard, "Histoire du Neurone : Les techniques de l'histoire", LA LETTRE des neurosciences bulletin de la société des neurosciences, Automne-Hiver 2008, N°35, pp 3-7.
- [7] F. Clarac, J-P Ternaux, "Encyclopédie historique des neurosciences- du neurone à l'émergence de la pensée", Edition de Boeck Université, 2008, ISBN 978-2-8041-5898-9
- [8] <http://organisationtissulaire.blogspot.com/p/iv-le-tissu-nerveux.html>
- [9] <http://fr.wikipedia.org/wiki/neurone>
- [10] <http://www.corpshumain.ca/images/>
- [11] [www.maxicours.com](http://www.maxicours.com)
- [12] [http://cours.etsmtl.ca/gts812/Cours/GTS812\\_Cours9\\_images.ppt](http://cours.etsmtl.ca/gts812/Cours/GTS812_Cours9_images.ppt)
- [13] B. Muller, J. Reinhardt, "Neural Networks: Models, Statistical Physics and Codes", Durham and Frankfurt, 1990.
- [14] S. Y. Kung, J. N. Hwang, " Neural Network Architectures for Robotic Applications", IEEE Transactions on Robotics and Automation, Vol. 5, No. 5, pp. 641-657, 1989.
- [15] D. W. Patterson, "Artificial Neural Networks, Theory and Applications", Prentice Hall, 1996.
- [16] J. A. Freeman, D. M. Skapura, "Neural Networks Algorithms, Applications and Programming Techniques", CNS, Addison-Wesley Publishing Company, July 1992.
- [17] W. Schiffman, M. Jost, R. Werner, "Optimisation of the Back Propagation Algorithm for Training Multilayer Perceptrons", Technical report, Sep. 1994 [www.cs.bham.ac.uk/~pxt/NC/schiffmann.bp.pdf](http://www.cs.bham.ac.uk/~pxt/NC/schiffmann.bp.pdf)
- [18] A. K. Jain, K. M. Mahiuddin, "Artificial Neural Networks: A tutorial", IEEE Computer, pp. 31-44, March 1996.
- [19] J .Vreeken, "Spiking neural networks, an introduction", Technical Report UU-CS, Issue: 2003-008 (2003), pp. 1-5
- [20] F. Schwartz, "Méthodologie de conception de systèmes analogiques. Utilisation de l'inversion ensembliste", thèse de doctorat, université de Strasbourg, 2010
- [21] W. MAASS, "Networks of Spiking Neurons: The Third Generation of Neural Network Models", *Neural Networks journal*, Vol. 10, No. 9, pp. 1659-1671, 1997
- [22] H. Demuth, M. Beale, "Neural Network Toolbox for Use with MATLAB", *User Guide Version 4*, 1992.
- [23] A. Zell, T. Korb, T. Sommer and R. Bayer " Recent Developments of the SNNs Neural Network Simulator", In Proceedings of the Applications of Neural Networks Conference, SPIE, volume 1294, pp. 534-544, 1990.
- [24] S. C. Lindsey, T. Lindblad, "Survey of Neural Network Hardware", Invited Paper, SPIE 1995, Symposium on Aerospace/Defense Sensing and Control and Dual Use Photonics, Proc. of Applications and Science of Artificial Neural Networks Conference, SPIE VOL. 2492, part Two, pp.1194-1205, 1995.
- [25] Y. Liao, "Neural networks in hardware: a survey", Department of Computer Science, University of California, 2001.
- [26] D. Anderson, G. McNeill, "Artificial Neural Networks Technology", a DACS State-of-the-Art Report, Contract Number F30602-89-C-0082, 1992.
- [27] K. Mathia, "Benchmarking an MIMD Neural Network Processor", World Congress on Neural Networks 1996 (WCNN'96), San Diego, California, pp.1321-1326, Sep. 1996

- [28] J. Pearson, R. Lippmann, "DARPA Neural Network study", Lexington, MA: MIT Lincoln Laboratory, 1988, International Press, ISBN O-916159-17
- [29] T. Nordstrom, B. Svensson, "Using and Designing Massively Parallel Computers for Artificial Neural Networks", Technical Report TULEA 91:1, Division of Computer Engineering, Lulea University of Technology, Sweden, S-95187, 1995.
- [30] M. J. Flynn, "Some Computer Organization and their Effectiveness", IEEE Transactions on Computers, Vol. 21, pp. 948-60, 1972.
- [31] T.Watanabe, "Neural Network Simulation on A Massively Parallel Cellular Array Processor: AAP-2", International Joint Conference on Neural Networks, WashingtonDC.Vol. 2, pp.155-161, 1989.
- [32] W.D. Hillis, G. L. J. Steel, "Data parallel algorithms", Communications of the ACM. Vol. 29(12): pp. 1170-1183, 1986.
- [33] P. Christy, "Software to support massively parallel computing on the MasPar MP-1", Proceedings of COMPCON, SANFRANSISCO, CA, pp. 29-33, 1990.
- [34] D. J. Hunt, "AMT DAP: A Processor Array in a Workstation Environment", Computer Systems Science and Engineering. Vol. 4(2): pp. 107-114, 1989.
- [35] T. Nordström, "Sparse distributed memory simulation on REMAP3", Research Report No. TULEA 1991:16, Luleå University of Technology, Sweden, 1991.
- [36] M. Duranton, J. A. Sirat, "Learning on VLSI: A General-Purpose Digital Neurochip", Philips Journal of Research. Vol. 45(1): pp. 1-17, 1990.
- [37] D. Hammerstrom, "A VLSI Architecture for High-Performance, Low Cost, On-Chip Learning" International joint conference on neural networks, San Diego .Vol. 2, pp. 537-543, 1990.
- [38] J. Beetem, M. Denneau, D. Weingarten, "The GF11 Parallel Computer", Experimental Parallel Computing Architectures. Dongarra edition. North-Holland, 1987.
- [39] P. M. Bavan, S. Lee and P. Trealeven, "A simple VLSI architecture for Neurocomputing", Proceedings of the international Neural Network Society", First annual Meeting, Boston, Massachusetts, pp 398, September 1988.
- [40] H. Kato, "A parallel Neurocomputer Architecture Towards Billion Connection Updates Per second", In International Joint Conference on Neural Networks, Vol. 2, pp. 47-50, Washington D.C, 1990.
- [41] N. Morgan, "The RAP: A Ring Array Processor for Layered Network Calculations", Proceedings of the Conference on Application Specific Array Processors, Princeton, NJ, pp. 296-308, 1990.
- [42] H. T. Kung, "The Warp Computer: Architecture, Implementation and Performance", *IEEE Transaction on Computers*. Vol. 36, Issue 12, pp. 1523-1528, Dec. 1987.
- [43] M. Yasunaga, "A wafer scale integration neural network utilizing completely digital circuits", In International joint conference on neural networks", Vol. 2, pp. 213-217, Washington DC, 1989.
- [44] U. Ramacher, M. Wesseling. "Systolic synthesis of neural networks." In *International Neural Network Conference*, Vol. 2, pp. 572-576, Paris, 1990.
- [45] A. Ferrucci, "ACME: A Field Programmable Gate Array Implementation of a Self Adapting and Scalable Connectionist Network", Master Thesis, 1994.
- [46] D. Jackson, D. Hammerstrom, " Distributing Back Propagation Networks Over the Intel iPSC Hypercube", IEEE International Joint Conference on Neural Networks, Seattle WA, Vol. 1, pp. 569-574, July 1991.
- [47] H. Eguchi, "Neural Network LSI Chip with On chip Learning", IEEE International Joint Conference on Neural Networks, Seattle, WA, Vol., pp. 453-456, July 1991.
- [48] J. Wawrzynek, K. Asanovi, N. Morgan, "The Design of a Neuro Microprocessor", IEEE Transactions on Neural Networks, Vol. 4, pp. 394-399, May 1993.
- [49] C. E. Cox, W. E. Blanz, "GANGLION: A Fast Field Programmable Gate Array Implementation of a Connectionist Classifier", IEEE Journal of Solid State Circuits, Vol. 3, pp. 288-299, March, 1992.
- [50] M. Yasunaga, N. Masuda, "A Self-Learning Neural Network composed of 1152 Digital Neurons in Wafer-Scale LSIs", Proceedings of the IJCNN-91-Singapore, pp. 1844 -1849, 1991.

- [51] M. Glesner, W. Pochmuller, "Neurocomputers: An Overview of Neural Networks in VLSI", Chapman & Hall Pub., ISBN 0-412-56390-8, 1994.
- [52] P. Ienne, "Digital systems for neural networks", Digital Signal Processing Technology, Vol. CR57 of Critical Reviews Series, SPIE Optical Engineering, Orlando, pp. 314-345, 1995
- [53] X. Driancourt, Personal Communication. Neuristique, Inc. Dec. 1994
- [54] A. J. De Groot, S. R. Parker, "Systolic Implementation of Neural Networks", SPIE, The International Society for Optical Engineering. K. Bromley editor, High Speed Computing II, Los Angeles, Vol. 1058, pp. 182-190, f. Jan 1989.
- [55] U. A. Muller, A. Gunzinger, W. Guggenbuhl, "Fast Neural Net Simulation with a DSP Processor Array", IEEE Transactions on Neural Networks, pp. 203-213, Jan. 1995
- [56] Nestor Inc. Providence, R.I Ni1000, Recognition Accelerator Datasheet, 1994.
- [57] U. Ramacher, "Design of a 1st Generation Neurocomputer", VLSI Design of Neural Networks, Kluwer Academic Publishers, Norwell, Mass, pp. 271-310, 1991.
- [58] M. A. Viredaz, P. Ienne, "MANTRA I : A Systolic Neurocomputer", *Proceedings of the International Joint Conference on Neural Networks*, Japan, Vol. III, pp. 3054-3057, Oct. 1993.
- [59] J. N. H. Heemskerk, "Neurocomputers for Brain-Style Processing: Design, Implementation and Application", PhD Thesis, Leiden University, The Netherlands, 1995.
- [60] P. C. Trealeven, "Neurocomputers", International Journal of Neural Computing, Vol. 1, pp. 4-31, 1989.
- [61] <http://neuralnets.web.cern.ch/NeuralNets/nnwInHepHard.html>
- [62] W.A. Hanson, C.A. Cruz, J.Y. Tam, "CONE -Computational Network Environment", Proc. IEEE First Int. Conf. on Neural Networks, pp. III-531-538, 1987.
- [63] J. Onuki, T. Maenosono, "ANN Accelerator by Parallel Processor Based on DSP", Proceedings of the IJCNN-93-Nagoya, pp. 1913-1916, 1993.
- [64] I. Aleksander, W. Thomas, P. Bowden, "WISARD, A Radical New Step Forward in Image Recognition", Sensor Review, pp. 120-124, 1984.
- [65] H. P. Speckman, Thole, W. Rosenstiel "COKOS: A COprocessor for Kohonen's Self Organizing Map", Proceedings of the ICANN-93-Amsterdam, London: Springer-Verlag, pp. 1040-1045, 1993.
- [66] S. C. J Garth, "A Chipset for High Speed Simulation of Neural Network Systems" Proceedings of the IEEE first Int. Conf. on Neural Networks, III-443-452, 1987.
- [67] U. Ramacher, W. Raab, "Multiprocessor and Memory Architecture of the Neurocomputer SYNAPSE-1", Proceedings of the 3rd International Conference on Microelectronics for Neural Networks (Micro Neuro), Edinburgh, pp. 227-231, April 1993.
- [68] F. J. Castillo, J. Cabestany, J.M. Moreno, "The Dynamic Ring Architecture", Proceedings of the ICANN-92-Brighton UK, Amsterdam:Elsevier, pp. 1439-1442, 1992.
- [69] R. David, E. Williams, G. De Tremiolles, P. Tannhof, "Description and practical uses of IBM ZISC036", Proc. SPIE, pp. 3728:198-211, 1999
- [70] I. Aybay, Cetinkaya, Halici, "Classification of Neural Network Hardware", Neural Network World, IDG Co., Vol. 6 No 1, pp 11-29, 1996,
- [71] M. Holler, "An Electrically Trainable Artificial Neural Network (ETANN) with 1024 "Floating Gate Synapse", Proceedings of IACNN 1989, pp. 191-196, 1989.
- [72] V. Beiu, "Digital integrated circuit implementations", IOP Publishing Ltd and Oxford University Press Handbook of Neural Computation release 97/1, pp. 1-34, 1997.
- [73] T. Schoenauer, "Digital Neurohardware: Principles and Perspectives", Neural Networks in Applications - NN'98 -, pp.101-106 Magdeburg 1998.
- [74] A. Jahnke, U. Roth, H. Klar, "A SIMD/dataflow architecture for a neurocomputer for spike-processing neural networks (NESPINN)", MicroNeuro'96, pp 232-237, 1996
- [75] S. Draghici, "Neural NETWORKS IN analog hardware-design and implementation issues", Int. J. Neural Syst. 1:19-42, 2000
- [76] P. Hasler, C. Diorio, B.A. Minch, C. Mead, "Single transistor learning synapses", Advances in neural information processing systems 2. MIT Press, Cambridge 817-824, 1995
- [77] Y. Xie, M. Jabri, "Training algorithms for limited precision feed forward neural networks", technical report, SEPAL, 1991.

- [78] J. Kane, M. Paquin, "POPART: practical optical implementation of adaptative resonance theory 2", IEEE Trans Neural Netw 4:695–702, 1993
- [79] W. Fisher, "A programmable analog neural network processor", IEEE Trans Neural Netw 2:222–229, 1991.
- [80] D. Gorse, G. Taylor, T. Klarkson, "Extended functionality for pRAMs. In: International Conference on Artificial Neural Networks, ICANN'94, pp 705–708, 1994
- [81] C. Schneider, H. Card, "CMOS mean field learning", Electron Lett 27(19):1702–1704, 1991
- [82] C. Mead, M. Ismail, "Analog VLSI implementation of neural systems", Kluwer academic publisher, Boston, 239–246, 1989
- [83] T. Shima, "Neuro chips with on-chip back-propagation and/or hebbian learning", IEEE J Solid State Circuits 27:1868–1876, 1992
- [84] M. Silvott, M. Mahowald, C. Mead, "Real-time visual computations using analog CMOS processing arrays", In: Proceedings Of the stanford advanced research in VLSI conference. MIT Press, Cambridge, 1987
- [85] M. Jabri, B. Flower, "Weight perturbation: an optimal architecture learning technique for Analog VLSI feed forward and recurrent multilayer networks", IEEE Trans Neural Netw. 3(1), pp: 154–157, 1992
- [86] C. S. Lindsey, T. Lindbald, "Survey of neural network hardware", HardwareNNWCourse, <http://www1.cern.ch/NeuralNets/nnwInHepHard.html>
- [87] H. Eguci, "Neural Network LSI chip with on-chip learning", IEEE, Richo Co, Ltd, Yokohama, Seattle, , pp.453-456, 1991
- [88] <http://www.accurate-automation.com/Products/NNP/nnp.html>
- [89] H. McCartor, "Back Propagation Implementation on the Adaptive Solutions CNAPS Neurocomputer Chip", Proceedings of NIPS-3, "Advances in Neural Information Processing Systems 3<sup>rd</sup>ed. R. Lippmann et al. pp. 1028-1031, 1991.
- [90] Nestor Inc. Providence R.I Ni1000 Recognition Accelerator Datasheet, 1994
- [91] <http://www.calsci.com/Applications.html>
- [92] U. Ramacher U, M. Wesseling, "Systolic synthesis of neural networks", IntNeuraNetwConf 2:572–576, 1990
- [93] K. R. Nichols, "A reconfigurable Computing Architecture for Implementing Artificial Neural Networks on FPGA", Master Thesis, University of Guelph, UK, 2004.
- [94] J. G. Eldredge and B. L. Hutchings, "RRANN: The Run Time Reconfiguration Artificial Neural Network", IEEE Custom Integrated Circuits Conference, San Diego, CA, May 1-4, pp 77-80, 1994.
- [95] J. L. Beuchat, J.O haeni and E. Sanchez, "Hardware reconfigurable neural networks", IPPS/SPDP'98 Workshop n° 10, Orlando, Florida, USA, MARS , vol. 1388, pp. 91-98 1998.
- [96] A. Peres-Urbe, "Structure Adaptable Digital Neural Networks", PhD thesis 2052, Logic System laboratory, Computer Science department, Swiss federal Institute of Technology- Lausanne, October 1999.
- [97] H. de Garis, M. Korkin, "The Cam-Brain Machine: an FPGA Based Hardware Tool Which Evolves a 1000 Neuron Net Module in Seconds and Updates a 75 million Neuron Artificial Brain for Real Time Robot", Neurocomputing journal, Vol. 42, Issue 1-4, February 2002.
- [98] M. Skrbek, "Fast neural network implementation", Neural Network World. Vol. 9 (N° 5), pp. 375-391, 1999.
- [99] J. L. Holt, T. E. Baker, "Backpropagation Simulations Using Limited Precision Calculations".International Joint Conference on Neural Networks (IJCNN-91), Vol. 2, pp. 121-126, Seattle, WA, USA, July 8-14 1991.
- [100] J. G. Eldredge, "FPGA Density Enhancement of a Neural Network through Run-Time Reconfiguration", Master thesis, Department of Electrical and Computer engineering, Brigham Young University, 1994
- [101] M. Tavenik, A. Linde, "A Reconfigurable SIMD Computer for Artificial Neural Networks" Licentiate thesis No 189L, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 1995

- [102] T. Nordstrom, "On Line Localized Learning Systems Part II: Parallel Computer Implementation" Research report TULEA 1995:02, Division of Computer Science and Engineering, Lulea University of Technology, Sweden, 1995.
- [103] M. I. Elmasry, "VLSI Artificial neural networks engineering", Kluwer Academic Publishers, 1994.
- [104] V. Kakkar, "Comparative study on analog and gital neural networks", *Int. J.Comput of SciNetwSecur (IJCSNS)* 9(7), pp 14–19, 2009
- [105] S. G. Merchant, G. D. Peterson, "Evolvable block-based neural network design for applications in dynamic environments", *VLSI Design Hindawi Publishing Corporation*. doi:10.1155/2010/251210, 2010
- [106] J. L. Ayala, "Design of a pipelined hardware architecture for real-time neural network computations", *Proceedings of the 45<sup>th</sup>midwest symposium on circuits and systems, MWSCAS'02 Tulsa, Okla, USA, vol 1*, pp 419–422, 2002
- [107] M. Moussa, S. Areibi, K. Nichols, "On the arithmetic precision for implementing back-propagation networks on FPGA: a case study", *Livre: FPGA implementations of neural networks*. Springer, Berlin pp 37–61, 2006, editions: Omondi AR, Rajapakse JC.
- [108] Y. Sato, "Development of a highperformance, general purpose neuro-computer composed of 512 digital neurons", *Proceedings of the International Joint Conference on Neural Networks (IJCNN '93)*, Nagoya, Japan, vol. 2, pp 1967–1970, 1993.
- [109] T. Tang, O. Ishizuka, H. Matsumoto, "Backpropagation learning in analog T-model neural network hardware", *Proceedings of the International Joint Conference on Neural Networks (IJCNN '93)*, Nagoya, Japan, vol. 1, pp 899–902, 1993
- [110] B. Linares-Barranco, E. Sanchez-Sinencio, A. Rodriguez-Vazquez, J. L. Huertas, "A MOS analog adaptive BAM with on-chip learning and weight refreshing. *IEEE Trans. Neural. Net.* 4(3):445–455, 1993
- [111] T. Schmitz, S. Hohmann, K. Meier, J. Schemmel, F. Schurmann, "Speeding up hardware evolution: a coprocessor for evolutionary algorithms", *Evolv. Syst. Biol.Hardw. Lect. Notes Comput. Sci.* 2606:274–285, 2003
- [112] M. R. DeYong, R. L. Findley, C. Fields, "The design, fabrication, and test of a new VLSI hybrid analog-digital neural processing element", *IEEE Trans Neural Netw* 3(3):363–374, 1992.
- [113] A. PassosAlmeida, J. E. Franca, "A mixed mode architecture for implementation of analog neural networks with digital programmability", *Proceedings of the International Joint Conference on Neural Networks (IJCNN'93)*, Nagoya, Japan, vol. 1, pp 887–890, 1993.
- [114] N. Izeboudjen, C. Larbes, A. Farah, "A new classification approach for neural networks hardware: From strandards chips to embedded systems on chip", *Artificial Intelligence Review Journal*, Vol: 4, pp 491-534, 2014, DOI: 10.1007/s10462-012-9321-7
- [115] S. C.J Garth, "A Chipset for High Speed Simulation of Neural Networks Systems", *IEEE First International Conference of Neural Networks*, IEEE Press, pp. III-443-452, June 1987.
- [116] T. J. Sejnowski, C. R. Rosenberg, "Parallel networks that learn to pronounce English text", *Journal of Complex Systems*, Vol 1:145–168, 1987
- [117] U.Ramacher, U. Ruckert, "VLSI design of Neural Networks", Kluwer Academic Publisher, 1991.
- [118] S. J. Prange, "Architectures for a Biology-oriented Neuroemulator", In *VLSI Design of Neural networks book*, Kluwer Academic Publishers, pp 83-102, 1991.
- [119] J. Hoekstra, "Junction Charge Coupled Device Technology for Artificial Neural Networks", In *VLSI Design of Neural networks book*, pp 19-46, Kluwer Academic Publishers, 1991.
- [120] H. A. Castro, S. M. Tam, M. A. Holler, "Implementation and performance of an analognon volatile neural network", *AnalogIntegr. Circuits Signal Process* 4(2):97–113, 1993.
- [121] C. Mead, "Analog VLSI and Neural Networks", Addison-Wesley, Reading, Mass., 1988
- [122] M. Mahowald, "Analog VLSI chip for stereocorrespondence", *Proceedings of IEEE International Symposium on Circuits and Systems*", vol. 6, pp 347–350, 1994.
- [123] E. Culurciello, R. E. Cummings, K.A. Boahen, "A biomorphic digital image sensor", *IEEE J. Solid-State Circuits* 38(2):281–294, 2003.
- [124] S. Kameda, T. Yagi, "An analog VLSI chip emulating sustained and transient response channels of the vertebrate retina", *IEEE Trans Neural Netw* 14(5):1405–1412, 2003

- [125] A. Sanada, K. Ishii, T. Yagi T, "A robot vision system using a silicon retina", *Brain Inspir. Inf. Technol.* 266:135–139, 2010 doi:[10.1007/978-3-642-04025-2\\_23](https://doi.org/10.1007/978-3-642-04025-2_23)
- [126] P. Rocke, B. McGinley, J. Maher, F. Morgan, J. Harkin, "Investigating the suitability of FPAAs for evolved hardware spiking neural networks", In: ICES 2008, LNCS 5216, pp.118–129, 2008
- [127] AN221E04 Datasheet, "Dynamically reconfigurable FPAAwith Enhanced I/O, DS030100-U006b", 2010. Available in the internet at: <http://www.anadigm.com>
- [128] A. Menendez, G. Paillet, "Fish inspection system using a parallel neural network chip and the image knowledge builder application", *Artif.Intell. Mag.* 29(1), pp:21–28, 2008
- [129] P. Ienne, "GENES IV: A Bit Serial Processing Element for a Multi Model Neural Network Accelerator", *Proceedings of the International Conference on Application Specific Array Processors, IEEE Computer Society, Venice, Italy*, pp 345-356, October 1993,
- [130] S. Sudarshan, K. Santosh, S. L. Pinjare, "MDAC Synapse-Neuron for Analog Neural Networks", *IJANP*, 2, 523–527, 2010
- [131] A. Rodriguez, CR Dominguez, A. Rueda, R. L. Huertas, E. Sanchez-sinenco, "Non linear switched capacitor neural networks for optimization problems", *IEEE Trans. Circuits. Syst.* 37(3), pp:384–398, 1990
- [132] J. C Bor, C. Y. Wu, "Analog electronic cochlea design using multiplexing switched capacitor circuits", *IEEE Trans Neural Netw* 7(1):155–166, 1996
- [133] Y. Tsvividis, D. Anastassiou, "Switched- Capacitor Neural Networks," *Electronic letters*, Vol. 23, Number 18, pp. 958-959, 1987
- [134] G. M. Blair, "PLA Design for Single-Clock CMOS", *IEEE Journal of Solid State Circuits*, Vol.27, Issue: 8, pp. 1211-1213, Aug. 1992.
- [135] A. Dehon, "The density advantage of configurable computing", *IEEE Computer*, Vol. 33, N° 5, pp. 41-49, April 2000.
- [136] A. R. OMONDI, J. RAJAPAKSE, "FPGA Implementations of Neural Networks", *Kluwer Academic Publishers*, 2006
- [137] P. Lysaght, "Dynamically Reconfigurable Logic in Undergraduate Projects", *FPGAs*, W. Moore and W. Luk, Eds., Abingdon EE&CS Books, England, 1991.
- [138] P. James-Roxby and, B.A. Blodget, "Adapting Constant Multipliers in a Neural Network Implementation", *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, pp 335-336, 2000.
- [139] J. Zhu, G.J. Milne, B. K. Gunther, "Towards an FPGA Based Reconfigurable Computing Environment for Neural Network Implementations", *Artificial Neural Networks*, 7 - 10 September 1999, Conference Publication No. 470 0 IEE 1999.
- [140] A. Upegui, C. A. Pena-Reyes, E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks", *Microprocess Microsyst Elsevier* 29:211–223, 2005
- [141] M. Porrmann, "Implementation of Artificial Neural Networks on a Reconfigurable Hardware Accelerator", *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP.02)*, IEEE Computer Society, 2002
- [142] H. Shayani, P. J. Bentley, A. M. Tyrrell, " An FPGA-based model suitable for evolution and development of spiking neural networks", *European symposium on artificial neural networks advances in computational intelligence and Learning*, pp 197–202, 2008, ISBN 2-930307-08-0
- [143] B. Schrauwen, J. Van Campenhout, "Parallel hardware implementation of a broad class of spiking neurons using serial arithmetic", *Proceedings of ESANN*, pp 623–628, 2006
- [144] P. LEE, "Advances in the Design of the TOTEM Neurochip", *World Scientific Publishing Compagny*, pp. 1-6. [http://www.lapp.in2p3.fr/aihep/aihep96/abstracts/ai/AI\\_074.ps](http://www.lapp.in2p3.fr/aihep/aihep96/abstracts/ai/AI_074.ps)
- [145] P. Gigliotti, "Implementing Barrel Shifters Using Multipliers", *Application Note: XAPP195 (v1.1)* August, 2004. [www.xilinx.com](http://www.xilinx.com)
- [146] NC3003 - Digital Processor for Neural Networks", *Data sheet, Rel. 12/99*, <http://www.neuricam.com>
- [147] M. Diepenhorst, "Automatic Generation of VHDL Code for Neural Applications", *International Joint Conference on Neural Networks (IJCNN)*, Vol. 4, pp. 2302-2305, 1999.
- [148] A. R. Muñoz, "An IP Core and GUI for Implementing Multilayer Perceptron with a Fuzzy Activation Function on Configurable Logic Devices", *Journal of Universal Computer Science*, vol. 14, no. 10 , pp. 1678-1694, 2008.

- [149] M. J. Pearson, "Design and FPGA Implementation of an Embedded Real-Time Biologically Plausible Spiking Neural Network Processor", International Conference on Field Programmable Logic and Applications, Issue:24-26., pp. 582 – 585, 2005.
- [150] D. F. Wolf, "Using Embedded Processors in Hardware Models of Artificial neural networks", Proc. of the Brazilian Symposium of Intelligent Automation, Canela -Brazil, November, 2001.
- [151] Altera, Co., "Nios Embedded Processor", <http://www.altera.com>
- [152] Altera, Co. "Quartus Altera" [www.altera.com/support/software/sof-quartus.html](http://www.altera.com/support/software/sof-quartus.html)
- [153] A. Uker, A.Z. Alkar, "HW/SW Codesign of FPGA-based Neural Networks", Fifteenth Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN 2006)
- [154] P. J. Ashenden, « The VHDL Cookbook, <http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf>
- [155] Sagdeo, "The Complet Verilog book", Edition Lavoisier, 2000-2009.
- [156] GrotkerThorston and al. "System Design with SystemC", © Lavoisier , 2002.
- [157] StreamC Language Specification <http://www.graphics.stanford.edu/streamlang/streamc3-6-00.pdf>
- [158] Matthew Bowen, "Handel-C Language Reference Manual" Embedded Solutions Limited. Version 2.1
- [159] R. D'omer, A. Gerstlauer, D. Gajski, "SpecC Language Reference Manual", SpecC Technology Open Consortium, [www.specc.org](http://www.specc.org), December 2002.
- [160] R. Airiau, J. M. Berge, V. Olive, "Circuit Synthesis with VHDL", Kluwer Academic Publishers, 1994.
- [161] F. Faber, "Introduction à la Programmation en AINSI-C", 1997. [www.ltam.lu/cours-c](http://www.ltam.lu/cours-c)
- [162] Xilinx, "Vtix-E 1.8 V Field Programmable Gate Arrays", *Perliminary Product Specification DS022-2 (v2.3)*, Xilinx, Inc., 2001.
- [163] Celoxica Ltd. RC1000 Hardware Reference Manual. Celoxica Ltd, Oxfordshire, United Kingdom, v2.3 edition, 2001. [www.celoxica.com](http://www.celoxica.com)
- [164] ISE fondation user guide, [www.xilinx.com](http://www.xilinx.com)
- [165] D. W. Patterson, "Artificial Neural Networks, Theory and Applications", Prentice Hall, 1996.
- [166] Virtex-II user guide, <http://direct.xilinx.com/bvdoc/userguides/ug002.pdf>
- [167] N. Izeboudjen, A. Farah, H. Bessalah, A. Bouridene, N. Chikhi, "Towards a Platform for FPGA Implementation of the MLP Based back Propagation Algorithm" IWANN, Lecturer Notes on Computer Science (LNCS), pp. 497-505, 2007.
- [168] ModelSIM User Guide, [www.model.com](http://www.model.com)
- [169] Virtex-4 Datasheet, [www.xilinx.com](http://www.xilinx.com)
- [170] [www.opencores.org](http://www.opencores.org)
- [171] 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 , Hotel Sheraton 8-9, pp.210-215, Nov. 1999.
- [172] N. Izeboudjen, A. Farah, H. Bessalah, A. Bouridene, N. Chikhi, "High Level Design Approach for FPGA Implementation of ANNs", Encyclopedia of Artificial Intelligence, ISBN 978-1-59904-849-9, publiée par Information Science reference in USA and UK, pp. 831-839, July 2008.
- [173] N. Abel, "Outils et Méthodes pour les Architectures Reconfigurables Dynamiquement à Grain Fin. Synthèse et gestion Automatique des Flux de Données", *Thèse Présentée pour obtenir le grade de docteur de l'université de Cergy-Pontoise, Spécialité : Traitement des Images et du Signal*, 2006.
- [174] M. J. Whirthlin, "Improving Functional Density through Run-Time Circuit Reconfiguration", PhD Thesis, Department of Electrical and Computer Engineering, Brigham Young University, 1997.
- [175] J. L. Hennessy, D. A. Patterson "Computer Architecture: A Quantitative Approach", Morgan Kaufmann 2nd edition, pp 21-28, 1996.
- [176] Gate Count Capacity Metrics for FPGAs, XAPP 059 (V1.1), 1997, [www.xilinx.com](http://www.xilinx.com)
- [177] S. Guccione, D. Levi, P. Sundararajan. "JBits: Java Based Interface for Reconfigurable Computing", Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference. The John Hopkins University, 1999.



- [178] E.L. Horta, J. W. Lockwood, "PARBIT: A Tool to Transform Bit files to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs)", WUCS-01-13, Department of Computer Science, Applied Research Lab, Washington University, pp. 1-37, July 2001.
- [179] N. Dorairaj, E. Shiflet, M. Goosman, "PlanAhead Software as a Platform for Partial Reconfiguration", Xcell Journal, pp. 69-71, 2005.
- [180] Two Flows for Partial Reconfiguration: Module Based or Difference Based, XAPP290 (v1.2), 2004
- [181] G. E. Moore, "Cramming More Components onto Integrated Circuits", Electronics, Vol. 38, Number 8, 1965.
- [182] G. E. Moore, "Progress in digital integrated circuit", IEEE Int. Electron Devices Meeting Technology Digest, pp. 11, Dec. 1975
- [183] G. Gielen, R.A. Rutenbar, "Computer-Aided Design of Analog and Mixed Signal Integrated Circuits", *Proceedings IEEE*, Vol. 88, pp. 1825-1854, Dec. 2000
- [184] International Technology Roadmap for Semiconductors, Edition 2001: <http://www.itrs.net/Links/2001ITRS/Home.htm>
- [185] M. S. Benromdhane, V. K. Madiseti, J. W. Hines, "Quick turnaround ASIC Design, Core Behavioral Synthesis in VHDL", *Kluwer Academic Publishers*, 1996.
- [186] M. Keating, P. Bricaud, "Reuse Methodology Manual for System-On-A-Chip Designs", *Kluwer Academic Publishers*, 2002.
- [187] L. Loiseau, "Methodology Design reuse", Ecole Polytechnique de Montreal, Département Génie Electrique, 2001.
- [188] Daniel. D. Gajski, Allen C.-H. Wu and al. "Essential Issues for IP Reuse", Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific, pp. 37 – 42, 2000
- [189] W. Savage, J. Chilton, R. Camposano, "IP Reuse in the System on a Chip Era", Proceedings of the 13th international symposium on System synthesis, IEEE Computer Society, pp. 2-7, Spain 2000.
- [190] System Generator User Guide, <http://www.xilinx.com/tools/sysgen.htm>
- [191] R. Jain, P. yang, T. Yoshino, « FIRGEN: A Computer Aided Design System for High Performance FIR Filter Integrated Circuits", *IEEE Transaction on Signal Processing*, Vol. 39, pp. 1655-1668, July 1991.
- [192] M. S. benromdhane, V. K. Madesetti, "LMSGEN: A Prototyping Environment for Programmable and Adaptive Digital Filters in VLS", Chapter in *VLSI Signal Processing-VIII* (Editor: Jan Rabaey), Nov. 1994.
- [193] Leonardo Spectrum User Guide, [www.mentorgraphics.com](http://www.mentorgraphics.com)
- [194] ModelSim Mentor Graphics, [www.mentorgraphics.com](http://www.mentorgraphics.com)
- [195] N. Izeboudjen, A. Bouridane, A. Farah, H. Bessalah, "Application of design reuse to artificial neural networks: case study of the back propagation algorithm", *Neural computing and applications journal*, 2012, volume 21, pp: 1531-1544
- [196] J. Liu, "Detailed model shows FPGA's True Costs", EDN, pp 153-158, May 1995.
- [197] [www.openMore.com](http://www.openMore.com)

---

# *ANNEXES*


---



## QUESTIONNAIRE

We are interested in sketching the future of Artificial Neural Networks (ANNs), focusing on two topics:

**ANNs for large-scale applications**  
**ANN hardware**

All the answers will be collected in a forthcoming report: don't forget to add your contact address if you are interested in a copy of the report. For more information on , point your browser to <http://physig.ph.kol.ac.uk/neuronet/index.html>

**YOUR CONTACT ADDRESS:** \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

## 1. GENERAL INTRODUCTORY QUESTIONS

1.1 Which area does your organisation belong to ?

- University       Industry       Research institute       OTHER \_\_\_\_\_

1.2 What is the size of your organisation ?

- < 20       21-50       51-100       101-500       501-1000       > 1000

1.3 How do you rate your knowledge about ANNs ?

- Expert / researcher       Advanced user       Casual user

1.4 What is your level of involvement in ANN technology ?

- Theoretical research       Applied research       Final user

1.5 What are your areas of interest (please check all that apply) ?

- |   |  |   |
|---|--|---|
| <input type="checkbox"/> Pattern recognition    | <input type="checkbox"/> Neurobiology      | <input type="checkbox"/> Speech processing    |
| <input type="checkbox"/> Signal Processing      | <input type="checkbox"/> Simulators        | <input type="checkbox"/> Medical applications |
| <input type="checkbox"/> Image Processing       | <input type="checkbox"/> Cognitive Science | <input type="checkbox"/> Data mining          |
| <input type="checkbox"/> Financial applications | <input type="checkbox"/> Neural hardware   | <input type="checkbox"/> Theory of learning   |
| <input type="checkbox"/> Robotics and Control   | <input type="checkbox"/> OTHER _____       |   |

1.6 Could you please give us some information on your recent work on ANNs ?

.....  
 .....  
 .....  
 .....

1.7 What kind of computers do you use for your ANN related work ?

- Personal computer (Dos / Windows)       Workstation (UNIX, Linux, Solaris, etc.)       Supercomputer / Mainframe
- Accelerator boards (please specify) \_\_\_\_\_
- OTHER \_\_\_\_\_

1.8 What kind of software tools do you use for your ANN related work ?

- I write my own code       Freeware / Shareware       Commercial software
- OTHER \_\_\_\_\_

PLEASE CONTINUE ON THE BACK →



## 2. THE FUTURE OF ANNs FOR LARGE-SCALE APPLICATIONS\*

(\* we indicate with this term all the applications involving the processing of very large amount of data, or with a large number of processing elements or with large needs in terms of processing power.

2.1 Do you have any experience with large-scale applications ?

- no  
 yes (please detail) \_\_\_\_\_

2.2 Do you think that current ANNs can approach large-scale applications in a satisfactory way ?

- yes  
 no, because \_\_\_\_\_

2.3 If you answered "no" to question 2.2 : what are the weak points of the current approach ?

\_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

2.4 If you answered "no" to question 2.2 : what must be improved of ANN technology to better approach large scale problems (e.g. new architectures or algorithms, hardware implementations, software tools, etc.) ?

\_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

## 3. THE FUTURE OF ANN HARDWARE

3.1 Do you have any experience with ANN hardware ?

- no  
 yes (please detail) \_\_\_\_\_

3.2 Do you think that current ANN hardware is satisfactory ?

- yes  
 no, because \_\_\_\_\_

3.3 If you answered "no" to question 3.2 : what are the weak points of current ANN hardware ?

\_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

3.4 If you answered "no" to question 3.2 : what must be improved of ANN hardware technology ?

\_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

Please return the completed questionnaire to the main desk or send it to the following address:

DIBE – University of Genova, Via Opera Pia 11a, 16145 Genova, ITALY  
<http://www.dibe.unige.it/~neuronet> FAX: +39-010-3532777

(May – June 1999)

## ANNEXE B

### DESCRIPTION VHDL DE L'ALGORITHME RPG RESEAU 2-2-1 (XOR)

Dans cette section nous présentons la description VHDL et RTL des trois phases de l'algorithme RPG, à savoir la phase d'apprentissage, la phase de calcul d'erreur et la phase de mise à jour des poids synaptiques. L'organisation des fichiers de description VHDL est définie comme suit (Figure Annexe A-1) :

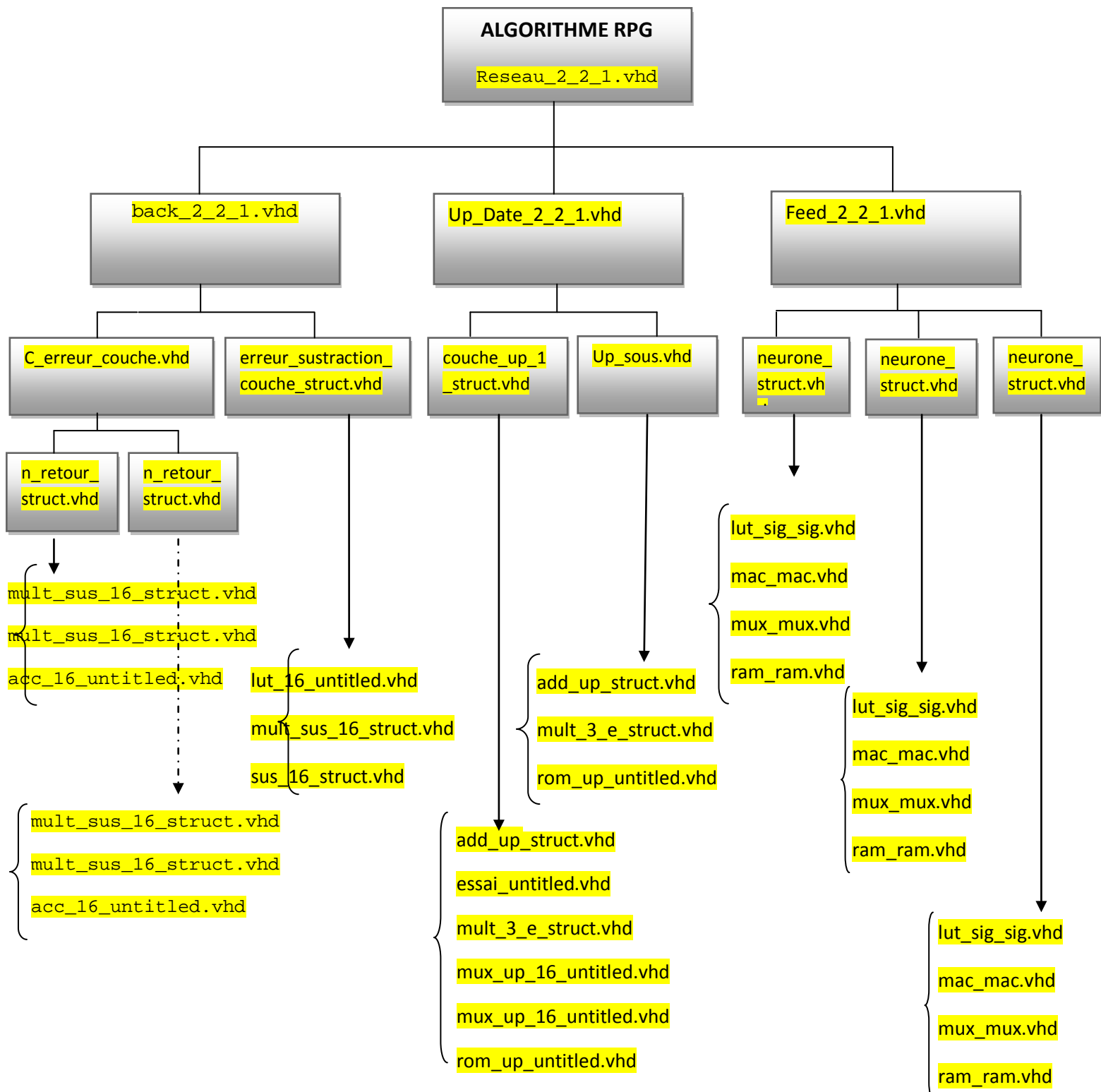


Figure Annexe B-1 : Organisation VHDL des fichiers pour la description du réseau 2-2-1

La description VHDL du réseau (2-2-1) est donnée comme suit:

```

-----
-- Copyright (c) 1995-2003 Xilinx, Inc.
-- All Right Reserved.
-----
--
--
-- Vendor: Xilinx
-- Version : 7.1i
-- Application : sch2vhdl
-- Filename : Reseau_2_2_1.vhf
-- Timestamp :
--
-- Command: C:/Xilinx/bin/nt/sch2vhdl.exe -intstyle ise -family virtex4
--flat -suppress -w Reseau_2_2_1.sch Reseau_2_2_1.vhf
--Design Name: Reseau_2_2_1
--Device: virtex4
--Purpose:
-- This vhdl netlist is translated from an ECS schematic. It can be
-- synthesis and simulted, but it should not be modified.
--
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
-- synopsys translate_off
library UNISIM;
use UNISIM.Vcomponents.ALL;
-- synopsys translate_on

entity Reseau_2_2_1 is
  port ( A          : in      std_logic_vector (15 downto 0);
        addr       : in      std_logic_vector (7  downto 0);
        addr_back  : in      std_logic_vector (15 downto 0);
        addr1      : in      std_logic_vector (7  downto 0);
        A1         : in      std_logic_vector (15 downto 0);
        clk        : in      std_logic;
        load       : in      std_logic;
        load_back  : in      std_logic;
        load1      : in      std_logic;
        read       : in      std_logic;
        read1      : in      std_logic;
        reset      : in      std_logic;
        reset_back : in      std_logic;
        reset1     : in      std_logic;
        sel        : in      std_logic;
        selm       : in      std_logic;
        sel_up     : in      std_logic;
        sell       : in      std_logic;
        val_up     : in      std_logic;
        write      : in      std_logic;
        write1     : in      std_logic;
        W1         : in      std_logic_vector (15 downto 0);
        W2         : in      std_logic_vector (15 downto 0);

```

```

        W3          : in    std_logic_vector (15 downto 0);
        XLXN_8      : in    std_logic;
        X1          : in    std_logic_vector (15 downto 0);
        X2          : in    std_logic_vector (15 downto 0);
        EM1         : out   std_logic_vector (15 downto 0);
        s_lut       : out   std_logic_vector (15 downto 0));
end Reseau_2_2_1;

architecture BEHAVIORAL of Reseau_2_2_1 is
    signal Wt1      : std_logic_vector (15 downto 0);
    signal Wt2      : std_logic_vector (15 downto 0);
    signal Wt3      : std_logic_vector (15 downto 0);
    signal XLXN_3   : std_logic_vector (15 downto 0);
    signal XLXN_62  : std_logic_vector (15 downto 0);
    signal XLXN_63  : std_logic_vector (15 downto 0);
    signal XLXN_65  : std_logic_vector (15 downto 0);
    signal XLXN_66  : std_logic_vector (15 downto 0);
    signal XLXN_67  : std_logic_vector (15 downto 0);
    signal XLXN_68  : std_logic_vector (15 downto 0);
    signal XLXN_69  : std_logic_vector (15 downto 0);
    signal XLXN_70  : std_logic_vector (15 downto 0);
    signal XLXN_71  : std_logic_vector (15 downto 0);
    signal XLXN_74  : std_logic_vector (15 downto 0);
    component back_2_2_1
        port ( add_sub : in    std_logic;
              clk      : in    std_logic;
              load     : in    std_logic;
              reset    : in    std_logic;
              addr_lut : in    std_logic_vector (15 downto 0);
              W12      : in    std_logic_vector (15 downto 0);
              W22      : in    std_logic_vector (15 downto 0);
              X11      : in    std_logic_vector (15 downto 0);
              X13      : in    std_logic_vector (15 downto 0);
              X21      : in    std_logic_vector (15 downto 0);
              Y1       : in    std_logic_vector (15 downto 0);
              EM       : out   std_logic_vector (15 downto 0);
              EM1      : out   std_logic_vector (15 downto 0);
              S_Lut    : out   std_logic_vector (15 downto 0);
              D13      : out   std_logic_vector (15 downto 0));
    end component;

    component up_date_2_2_1
        port ( sel      : in    std_logic;
              val       : in    std_logic;
              A         : in    std_logic_vector (15 downto 0);
              A1        : in    std_logic_vector (15 downto 0);
              D13       : in    std_logic_vector (15 downto 0);
              D13_23    : in    std_logic_vector (15 downto 0);
              O12       : in    std_logic_vector (15 downto 0);
              O13       : in    std_logic_vector (15 downto 0);
              O22       : in    std_logic_vector (15 downto 0);
              W11       : in    std_logic_vector (15 downto 0);
              W13       : in    std_logic_vector (15 downto 0);
              W23       : in    std_logic_vector (15 downto 0);
              Wt12      : out   std_logic_vector (15 downto 0);
              Wt13      : out   std_logic_vector (15 downto 0);
              Wt22      : out   std_logic_vector (15 downto 0));
    end component;

```

```

component feed
  port ( clk      : in      std_logic;
        load     : in      std_logic;
        load1    : in      std_logic;
        read     : in      std_logic;
        read1    : in      std_logic;
        reset    : in      std_logic;
        reset1   : in      std_logic;
        sel      : in      std_logic;
        selm     : in      std_logic;
        sell     : in      std_logic;
        write    : in      std_logic;
        writel   : in      std_logic;
        addr     : in      std_logic_vector (7 downto 0);
        addr1    : in      std_logic_vector (7 downto 0);
        Wt1     : in      std_logic_vector (15 downto 0);
        Wt3     : in      std_logic_vector (15 downto 0);
        W1      : in      std_logic_vector (15 downto 0);
        W23     : in      std_logic_vector (15 downto 0);
        XLXN_20 : in      std_logic_vector (15 downto 0);
        XLXN_22 : in      std_logic_vector (15 downto 0);
        XLXN_23 : in      std_logic_vector (15 downto 0);
        X1      : in      std_logic_vector (15 downto 0);
        O11     : out     std_logic_vector (15 downto 0);
        O13     : out     std_logic_vector (15 downto 0);
        O22     : out     std_logic_vector (15 downto 0);
        W11     : out     std_logic_vector (15 downto 0);
        W13     : out     std_logic_vector (15 downto 0);
        W22     : out     std_logic_vector (15 downto 0);
        X11     : out     std_logic_vector (15 downto 0);
        X12     : out     std_logic_vector (15 downto 0);
        X13     : out     std_logic_vector (15 downto 0));
end component;

```

```

begin
  XLXI_2 : back_2_2_1
    port map (addr_lut(15 downto 0)=>addr_back(15 downto 0),
             add_sub=>XLXN_8,
             clk=>clk,
             load=>load_back,
             reset=>reset_back,
             W12(15 downto 0)=>XLXN_66(15 downto 0),
             W22(15 downto 0)=>XLXN_65(15 downto 0),
             X11(15 downto 0)=>XLXN_67(15 downto 0),
             X13(15 downto 0)=>XLXN_63(15 downto 0),
             X21(15 downto 0)=>XLXN_74(15 downto 0),
             Y1(15 downto 0)=>XLXN_62(15 downto 0),
             D13(15 downto 0)=>XLXN_3(15 downto 0),
             EM(15 downto 0)=>XLXN_71(15 downto 0),
             EM1(15 downto 0)=>EM1(15 downto 0),
             S_Lut(15 downto 0)=>s_lut(15 downto 0));

  XLXI_3 : up_date_2_2_1
    port map (A(15 downto 0)=>A(15 downto 0),
             A1(15 downto 0)=>A1(15 downto 0),
             D13(15 downto 0)=>XLXN_3(15 downto 0),
             D13_23(15 downto 0)=>XLXN_71(15 downto 0),

```



```

O12(15 downto 0)=>XLXN_68(15 downto 0),
O13(15 downto 0)=>XLXN_62(15 downto 0),
O22(15 downto 0)=>XLXN_69(15 downto 0),
sel=>sel_up,
val=>val_up,
W11(15 downto 0)=>XLXN_66(15 downto 0),
W13(15 downto 0)=>XLXN_65(15 downto 0),
W23(15 downto 0)=>XLXN_70(15 downto 0),
Wt12(15 downto 0)=>Wt1(15 downto 0),
Wt13(15 downto 0)=>Wt3(15 downto 0),
Wt22(15 downto 0)=>Wt2(15 downto 0));

XLXI_4 : feed
  port map (addr(7 downto 0)=>addr(7 downto 0),
            addr1(7 downto 0)=>addr1(7 downto 0),
            clk=>clk,
            load=>load,
            load1=>load1,
            read=>read,
            read1=>read1,
            reset=>reset,
            reset1=>reset1,
            sel=>sel,
            selm=>selm,
            sell=>sell,
            write=>write,
            writel=>writel,
            Wt1(15 downto 0)=>Wt1(15 downto 0),
            Wt3(15 downto 0)=>Wt3(15 downto 0),
            W1(15 downto 0)=>W1(15 downto 0),
            W23(15 downto 0)=>W3(15 downto 0),
            XLXN_20(15 downto 0)=>X2(15 downto 0),
            XLXN_22(15 downto 0)=>W2(15 downto 0),
            XLXN_23(15 downto 0)=>Wt2(15 downto 0),
            X1(15 downto 0)=>X1(15 downto 0),
            O11(15 downto 0)=>XLXN_68(15 downto 0),
            O13(15 downto 0)=>XLXN_62(15 downto 0),
            O22(15 downto 0)=>XLXN_69(15 downto 0),
            W11(15 downto 0)=>XLXN_66(15 downto 0),
            W13(15 downto 0)=>XLXN_70(15 downto 0),
            W22(15 downto 0)=>XLXN_65(15 downto 0),
            X11(15 downto 0)=>XLXN_67(15 downto 0),
            X12(15 downto 0)=>XLXN_74(15 downto 0),
            X13(15 downto 0)=>XLXN_63(15 downto 0));

end BEHAVIORAL;
```

```

-----
-----
-- Copyright (c) 1995-2003 Xilinx, Inc.
-- All Right Reserved.
-----
-----
--
--
-- /-----\ /-----\
-- /-----\ \-----\
-- \-----\ \-----\
-- \-----\ /-----\
-- /-----\ /-----\
-- /-----\ \-----\
-- \-----\ \-----\
--
-- Vendor: Xilinx
-- Version : 7.1i
-- Application : sch2vhdl
-- Filename : back_2_2_1.vhf
-- Timestamp :
--
--Command: C:/Xilinx/bin/nt/sch2vhdl.exe -intstyle ise -family virtex4
-flat -suppress -w back_2_2_1.sch back_2_2_1.vhf
--Design Name: back_2_2_1
--Device: virtex4
--Purpose:
-- This vhdl netlist is translated from an ECS schematic. It can be
-- synthesis and simulted, but it should not be modified.
--

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
-- synopsys translate_off
library UNISIM;
use UNISIM.Vcomponents.ALL;
-- synopsys translate_on

entity back_2_2_1 is
  port ( addr_lut : in      std_logic_vector (15 downto 0);
        add_sub  : in      std_logic;
        clk      : in      std_logic;
        load     : in      std_logic;
        reset    : in      std_logic;
        W12      : in      std_logic_vector (15 downto 0);
        W22      : in      std_logic_vector (15 downto 0);
        X11      : in      std_logic_vector (15 downto 0);
        X13      : in      std_logic_vector (15 downto 0);
        X21      : in      std_logic_vector (15 downto 0);
        Y1       : in      std_logic_vector (15 downto 0);
        D13      : out     std_logic_vector (15 downto 0);
        EM       : out     std_logic_vector (15 downto 0);
        EM1      : out     std_logic_vector (15 downto 0);
        S_Lut    : out     std_logic_vector (15 downto 0));
end back_2_2_1;

architecture BEHAVIORAL of back_2_2_1 is
  signal D13_DUMMY : std_logic_vector (15 downto 0);
  component c_erreur_couche
    port ( add_sub : in      std_logic;
          clk      : in      std_logic;
          load     : in      std_logic;
          reset    : in      std_logic;

```

```

        SM      : in      std_logic_vector (15 downto 0);
        W12     : in      std_logic_vector (15 downto 0);
        W22     : in      std_logic_vector (15 downto 0);
        X11     : in      std_logic_vector (15 downto 0);
        X21     : in      std_logic_vector (15 downto 0);
        EM      : out     std_logic_vector (15 downto 0);
        EM1     : out     std_logic_vector (15 downto 0));
end component;

    component erreur_sustraction_couche
port ( X13      : in      std_logic_vector (15 downto 0);
        Y1      : in      std_logic_vector (15 downto 0);
        addr_lut : in      std_logic_vector (15 downto 0);
        SM      : out     std_logic_vector (15 downto 0);
        S_lut   : out     std_logic_vector (15 downto 0));
    end component;

begin
    D13(15 downto 0) <= D13_DUMMY(15 downto 0);
    XLXI_1 : c_erreur_couche
port map (add_sub=>add_sub,
        clk=>clk,
            load=>load,
            reset=>reset,
            SM(15 downto 0)=>D13_DUMMY(15 downto 0),
            W12(15 downto 0)=>W12(15 downto 0),
            W22(15 downto 0)=>W22(15 downto 0),
            X11(15 downto 0)=>X11(15 downto 0),
            X21(15 downto 0)=>X21(15 downto 0),
            EM(15 downto 0)=>EM(15 downto 0),
            EM1(15 downto 0)=>EM1(15 downto 0));

    XLXI_2 : erreur_sustraction_couche
port map (addr_lut(15 downto 0)=>addr_lut(15 downto 0),
        X13(15 downto 0)=>X13(15 downto 0),
            Y1(15 downto 0)=>Y1(15 downto 0),
            SM(15 downto 0)=>D13_DUMMY(15 downto 0),
            S_lut(15 downto 0)=>S_Lut(15 downto 0));

end BEHAVIORAL;

```

```

-----
-----
-- Copyright (c) 1995-2003 Xilinx, Inc.
-- All Right Reserved.
-----
-----
--
--
-- /-----/ \-----/
-- /-----/ \-----/      Vendor: Xilinx
-- \-----/ \-----/      Version : 7.1i
-- \-----/ \-----/      Application : sch2vhdl
-- /-----/ \-----/      Filename : Feed.vhf
-- /-----/ \-----/      Timestamp :
-- \-----/ \-----/
--
--Command: C:/Xilinx/bin/nt/sch2vhdl.exe -intstyle ise -family virtex4
-flat -suppress -w Feed.sch Feed.vhf
--Design Name: Feed
--Device: virtex4
--Purpose:
-- This vhdl netlist is translated from an ECS schematic. It can be
-- synthesis and simulted, but it should not be modified.
--

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
-- synopsys translate_off
library UNISIM;
use UNISIM.Vcomponents.ALL;
-- synopsys translate_on

entity Feed is
  port ( addr      : in      std_logic_vector (7 downto 0);
        addr1     : in      std_logic_vector (7 downto 0);
        clk       : in      std_logic;
        load      : in      std_logic;
        load1     : in      std_logic;
        read      : in      std_logic;
        read1     : in      std_logic;
        reset     : in      std_logic;
        reset1    : in      std_logic;
        sel       : in      std_logic;
        selm      : in      std_logic;
        sell      : in      std_logic;
        write     : in      std_logic;
        writel    : in      std_logic;
        Wt1      : in      std_logic_vector (15 downto 0);
        Wt3      : in      std_logic_vector (15 downto 0);
        Wl       : in      std_logic_vector (15 downto 0);
        W23     : in      std_logic_vector (15 downto 0);
        XLXN_20 : in      std_logic_vector (15 downto 0);
        XLXN_22 : in      std_logic_vector (15 downto 0);
        XLXN_23 : in      std_logic_vector (15 downto 0);
        Xl       : in      std_logic_vector (15 downto 0);
        O11     : out     std_logic_vector (15 downto 0);
        O13     : out     std_logic_vector (15 downto 0);

```

```

        O22      : out   std_logic_vector (15 downto 0);
        W11      : out   std_logic_vector (15 downto 0);
        W13      : out   std_logic_vector (15 downto 0);
        W22      : out   std_logic_vector (15 downto 0);
        X11      : out   std_logic_vector (15 downto 0);
        X12      : out   std_logic_vector (15 downto 0);
        X13      : out   std_logic_vector (15 downto 0));
end Feed;

architecture BEHAVIORAL of Feed is
    signal XLXN_3      : std_logic_vector (15 downto 0);
    signal O11_DUMMY  : std_logic_vector (15 downto 0);
    signal O22_DUMMY  : std_logic_vector (15 downto 0);
    component neurone
        port ( clk      : in     std_logic;
              load     : in     std_logic;
              read     : in     std_logic;
              reset    : in     std_logic;
              sel      : in     std_logic;
              write    : in     std_logic;
              X        : in     std_logic_vector (15 downto 0);
              addr     : in     std_logic_vector (7 downto 0);
              w        : in     std_logic_vector (15 downto 0);
              wt1     : in     std_logic_vector (15 downto 0);
              O11     : out    std_logic_vector (15 downto 0);
              X11     : out    std_logic_vector (15 downto 0);
              wc      : out    std_logic_vector (15 downto 0));
    end component;

    component mux
        port ( sel      : in     std_logic;
              W        : in     std_logic_vector (15 downto 0);
              WT1     : in     std_logic_vector (15 downto 0);
              WN      : out    std_logic_vector (15 downto 0));
    end component;

begin
    O11(15 downto 0) <= O11_DUMMY(15 downto 0);
    O22(15 downto 0) <= O22_DUMMY(15 downto 0);
    XLXI_1 : neurone
        port map (addr(7 downto 0)=>addr(7 downto 0),
                clk=>clk,
                load=>load,
                read=>read,
                reset=>reset,
                sel=>sel,
                w(15 downto 0)=>W1(15 downto 0),
                write=>write,
                wt1(15 downto 0)=>Wt1(15 downto 0),
                X(15 downto 0)=>X1(15 downto 0),
                O11(15 downto 0)=>O11_DUMMY(15 downto 0),
                wc(15 downto 0)=>W11(15 downto 0),
                X11(15 downto 0)=>X11(15 downto 0));

    XLXI_2 : neurone
        port map (addr(7 downto 0)=>addr(7 downto 0),
                clk=>clk,
                load=>load,

```

```

        read=>read,
        reset=>reset,
        sel=>sel,
        w(15 downto 0)=>XLXN_22(15 downto 0),
        write=>write,
        wt1(15 downto 0)=>XLXN_23(15 downto 0),
        X(15 downto 0)=>XLXN_20(15 downto 0),
        O11(15 downto 0)=>O22_DUMMY(15 downto 0),
        wc(15 downto 0)=>W22(15 downto 0),
        X11(15 downto 0)=>X12(15 downto 0));

XLXI_3 : mux
port map (sel=>selm,
W(15 downto 0)=>O11_DUMMY(15 downto 0),
        WT1(15 downto 0)=>O22_DUMMY(15 downto 0),
        WN(15 downto 0)=>XLXN_3(15 downto 0));

XLXI_4 : neurone
    port map (addr(7 downto 0)=>addr1(7 downto 0),
        clk=>clk,
        load=>load1,
        read=>read1,
        reset=>reset1,
        sel=>sel1,
        w(15 downto 0)=>W23(15 downto 0),
        write=>writel,
        wt1(15 downto 0)=>Wt3(15 downto 0),
        X(15 downto 0)=>XLXN_3(15 downto 0),
        O11(15 downto 0)=>O13(15 downto 0),
        wc(15 downto 0)=>W13(15 downto 0),
        X11(15 downto 0)=>X13(15 downto 0));

end BEHAVIORAL;
```

```
-----  
-----  
-- Copyright (c) 1995-2003 Xilinx, Inc.  
-- All Right Reserved.  
-----  
-----  
--
```

```
-----  
--  
-- /-----\ /-----\  
-- /-----\ /-----\  
-- \-----/ \-----/  
-- \-----/ \-----/  
-- /-----\ /-----\  
-- /-----\ /-----\  
-- \-----/ \-----/  
-- \-----/ \-----/  
--  
-- Vendor: Xilinx  
-- Version : 7.1i  
-- Application : sch2vhdl  
-- Filename : Up_Date_2_2_1.vhf  
-- Timestamp :  
--  
-----
```

```
library ieee;  
use ieee.std_logic_1164.ALL;  
use ieee.numeric_std.ALL;  
-- synopsys translate_off  
library UNISIM;  
use UNISIM.Vcomponents.ALL;  
-- synopsys translate_on  
  
entity Up_Date_2_2_1 is  
    port ( A          : in    std_logic_vector (15 downto 0);  
          A1         : in    std_logic_vector (15 downto 0);  
          D13        : in    std_logic_vector (15 downto 0);  
          D13_23     : in    std_logic_vector (15 downto 0);  
          O12        : in    std_logic_vector (15 downto 0);  
          O13        : in    std_logic_vector (15 downto 0);  
          O22        : in    std_logic_vector (15 downto 0);  
          sel        : in    std_logic;  
          val        : in    std_logic;  
          W11        : in    std_logic_vector (15 downto 0);  
          W13        : in    std_logic_vector (15 downto 0);  
          W23        : in    std_logic_vector (15 downto 0);  
          Wt12       : out   std_logic_vector (15 downto 0);  
          Wt13       : out   std_logic_vector (15 downto 0);  
          Wt22       : out   std_logic_vector (15 downto 0));  
end Up_Date_2_2_1;
```

```
architecture BEHAVIORAL of Up_Date_2_2_1 is  
    component couche_up_1  
        port ( sel          : in    std_logic;  
              val          : in    std_logic;  
              A            : in    std_logic_vector (15 downto 0);  
              D13_23_33   : in    std_logic_vector (15 downto 0);  
              O12         : in    std_logic_vector (15 downto 0);  
              O22         : in    std_logic_vector (15 downto 0);  
              W13         : in    std_logic_vector (15 downto 0);  
              W23         : in    std_logic_vector (15 downto 0);  
              W13_3       : out   std_logic_vector (15 downto 0));  
    end component;
```

```

        W23_3      : out   std_logic_vector (15 downto 0));
end component;

component up_sous
  port ( A        : in    std_logic_vector (15 downto 0);
        D13_23_33 : in    std_logic_vector (15 downto 0);
        O13       : in    std_logic_vector (15 downto 0);
        W13       : in    std_logic_vector (15 downto 0);
        Wt3       : out   std_logic_vector (15 downto 0));
end component;

begin
  XLXI_1 : couche_up_1
    port map (A(15 downto 0)=>A(15 downto 0),
              D13_23_33(15 downto 0)=>D13(15 downto 0),
              O12(15 downto 0)=>O12(15 downto 0),
              O22(15 downto 0)=>O22(15 downto 0),
              sel=>sel,
              val=>val,
              W13(15 downto 0)=>W13(15 downto 0),
              W23(15 downto 0)=>W23(15 downto 0),
              W13_3(15 downto 0)=>Wt12(15 downto 0),
              W23_3(15 downto 0)=>Wt22(15 downto 0));

  XLXI_2 : up_sous
    port map (A(15 downto 0)=>A1(15 downto 0),
              D13_23_33(15 downto 0)=>D13_23(15 downto 0),
              O13(15 downto 0)=>O13(15 downto 0),
              W13(15 downto 0)=>W11(15 downto 0),
              Wt3(15 downto 0)=>Wt13(15 downto 0));

end BEHAVIORAL;
```

La figure Annexe A-2 donne la description de l'architecture au niveau RTL obtenue.



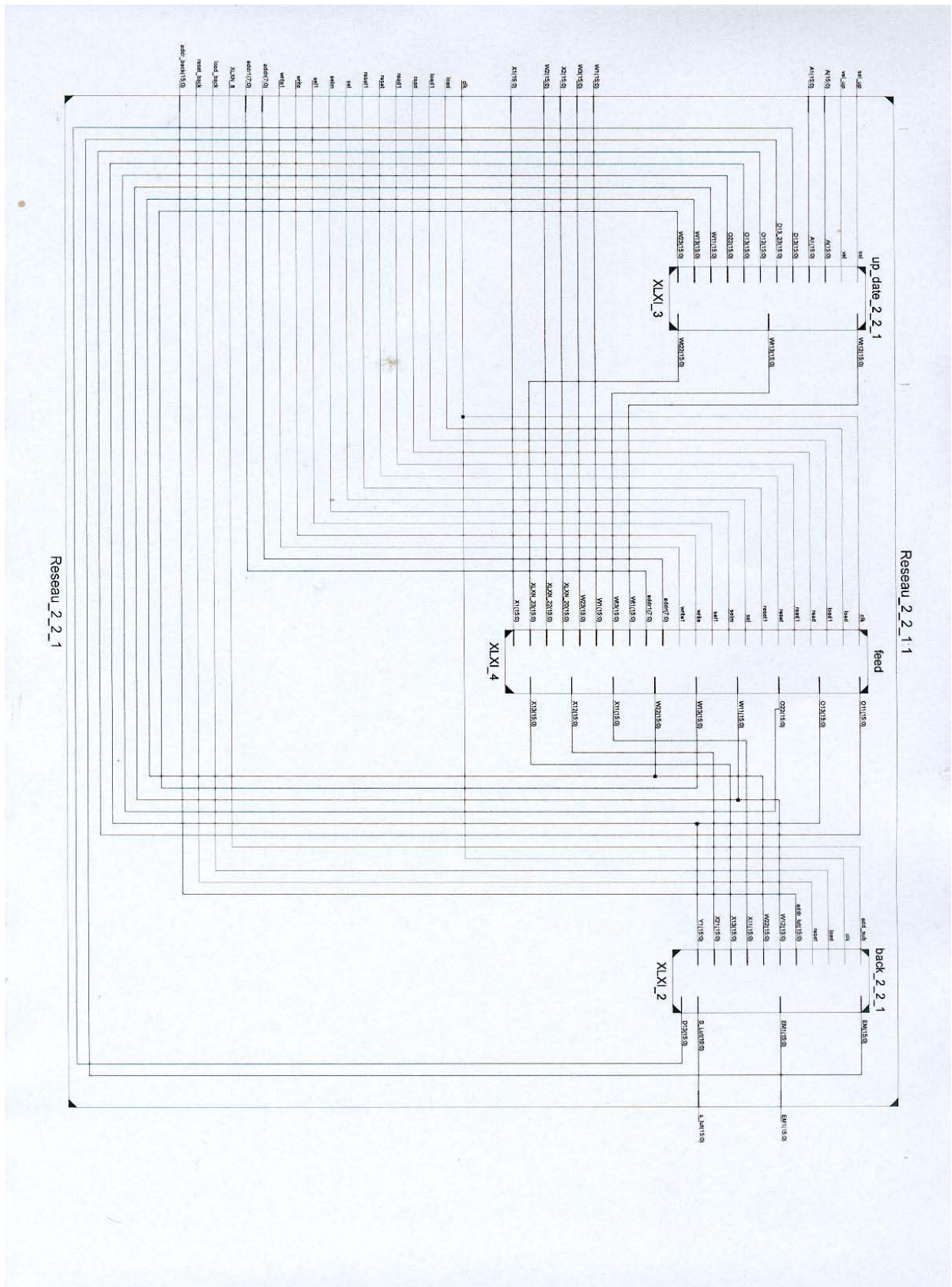


Figure Annexe- B-2 Schéma RTL généré : réseau 2-2-1

## *ANNEXE C*

### **REGLES DE DESIGN**

Les numéros des règles reprennent la notation du livre "Reuse Methodology Manual", pour pouvoir aisément s'y reporter.

#### **REGLES DE BASE**

Les règles de base ont pour but d'une part de permettre un codage clair et compréhensible et d'autre part indiquent également les principes pour faire un bon design (sans nécessairement parler de réutilisation). Ces règles devraient être appliquées quel que soit l'objectif du module.

- 5.2.1.1 – Définir une convention pour nommer les signaux, variable, boucles, process..., et la suivre tout au long du design.
- 5.2.1.2 – Utiliser des minuscules pour nommer les signaux, les variables et les ports.
- 5.2.1.3 – Utiliser des majuscules pour nommer les constantes et les sous-types dérivés de types standards
- 5.2.1.4 – Utiliser des noms descriptifs clairs pour les signaux, ports, fonctions et paramètres
- 5.2.1.6 – Utiliser le nom `clk` pour l'horloge, ou le préfixe `clk_` suivi d'un court descriptif (pour donner sa fonction ou sa fréquence par exemple).
- 5.2.1.8 – Les signaux actifs bas doivent être indiqués clairement par le suffixe `_n`
- 5.2.1.9 – Utiliser le nom `rst` pour un signal de reset, ou le préfixe `rst_` suivi d'un court descriptif (pour donner sa fonction). Si il est actif bas, le préciser par `rst_n`.
- 5.2.1.10 – Déclarer les ports avec MSB à gauche et LSB à droite (7 downto 0 par exemple).
- 5.2.3.1 – Utiliser la notation suivant pour nommer les architectures :
  - *rtl* pour l'architecture synthétisable d'un module, en description RTL ou mixte (RTL + structurelle).
  - *struct* pour une description complètement structurelle (instanciation de sous-blocs) d'un module
  - *behav* pour une architecture comportementale non-synthétisable (utilisée pour la simulation)
  - *tb* pour le test-bench
- 5.2.4.1 – Inclure une entête standard dans toutes les fichiers sources. Inclure les informations essentielles, dont l'utilisateur pourrait avoir besoin.

```
=====
=====
=====
```

```
-- TITLE : Consolidation (basic, not by block, not conditional)
-- DESCRIPTION : Configurable consolidation module
-- Size of the window, threshold and type of consolidation are generic
-- Only odd values for window size are valid.
-- The processing delay (due to the structure) is 3 clk cycles, and you
-- must add the time to reach central pixel.
-- Ex : for 3*3 : 1 line delay + 4 clk cycles.
-- Processes only demultiplexed data.
-- FILE : consolidation.vhd
```

```
--
```

```
=====
```

```
-- CREATION
```

```
-- DATE AUTHOR PROJECT REVISION
```

```
-- 2001/02/13 Ludovic Loiseau DT4101 V1.0
```

```
--
```

```
=====
```

```
-- MODIFICATION HISTORY
```

```
-- DATE AUTHOR PROJECT REVISION COMMENTS
```

```
--
```

- ```
=====
```
- 5.2.5.1 – Commenter généreusement le code pour expliquer tous les process, fonctions et déclarations de types et sous-types. Utiliser des commentaires intelligents (expliquer pourquoi, plutôt que de retraduire littéralement le code)
  - 5.2.5.2 – Tous les ports, signaux et variables, ainsi que les groupes de signaux et de variables doivent être commentés.
  - 5.2.6.1 – Une seule instruction par ligne de code.
  - 5.2.7.1 – Limiter la longueur des lignes de codes à 100 caractères pour améliorer la portabilité entre éditeurs de texte.
  - 5.2.8.1 – Indenter le code pour en améliorer la lisibilité et repérer rapidement la structure des boucles ou des actions conditionnelles.
  - 5.2.8.2 – Indenter par des espaces (et non des tabulations), de largeur 1 à 4. Rester consistant pour tout le codage. 3 semble être la meilleure valeur.
  - 5.2.9.1 – Ne pas utiliser de mots réservés au VHDL ou au Verilog pour nommer les éléments du code (voir la liste des mots réservés en annexe).
  - 5.2.10.1 – Déclarer les ports dans un ordre logique, par fonctionnalité plutôt que par type.
  - 5.2.10.2 – Ne déclarer qu'un seul port par ligne, si possible avec le commentaire associé sur la même ligne.
  - 5.2.10.3 – Pour chaque groupe logique de ports, respecter l'ordre suivant :
    - clocks,
    - resets,
    - enables,
    - autre signaux de contrôle,
    - données.
  - 5.2.10.4 – Commenter chaque groupe de ports.
  - 5.2.11.1 – Utiliser toujours un mapping explicite pour les ports et les paramètres génériques. Préciser toujours la valeur d'un générique au generic map (ne pas garder la valeur par défaut de l'entité).
  - 5.2.12.1 – Un seul fichier VHDL doit contenir l'entité, l'architecture et les configurations (si applicable).
  - 5.2.13.1 – Utiliser des fonctions plutôt que répéter une même section de code. Ne pas oublier de commenter la fonction.
  - 5.2.14.1 – Utiliser des boucles et des tableaux pour améliorer la clarté du code.
  - 5.2.14.2 – Utiliser des opérations vectorielles plutôt que des boucles lorsque c'est possible (ex : mapper directement 2 colonnes d'un tableau plutôt qu'élément par élément).

- 5.2.15.2 – Nommer un process en indiquant son but (ou le signal qu'il génère), précédé du préfixe `do_`.
- 5.2.15.3 – Le nom d'une instance d'un composant doit être `U_component_name`, ou `Ui_component_name` pour plusieurs instances d'un même composant dans une même entité.
- 5.2.15.5 – Ne pas utiliser de noms des signaux, variables ou entités comme étiquette de process.

#### Autres règles importantes :

- Ne pas mélanger majuscules et minuscules dans les noms des signaux, sauf exception précises (par exemple LD pour Line Delay). Utiliser le caractère `_` pour séparer des notions dans un même nom
  - L'entité devrait porter le même nom que le fichier dans lequel elle est décrite.
  - Ne générer qu'un seul signal par process séquentiel, ou alors un groupe de signaux de même niveau logique (par exemple, dans le cas d'une structure pipelinée, on peut assigner dans un même process tous les signaux du même niveau de pipeline).
  - Effacer toute ligne de code mise en commentaire.
  - N'utiliser que le type `std_logic` ou `std_logic_vector` aux port des blocs et sous-blocs. Pour des opérations arithmétiques, convertir les signaux au besoin en interne en type `signed` ou `unsigned`.
  - Ne pas donner de valeur par défaut à un signal. La plupart du temps, cette valeur est ignorée à la synthèse, donc peut amener des mismatches par rapport à la simulation.
  - Tous les noms de signaux, et tous les commentaires doivent être en anglais, afin d'avoir une certaine homogénéité par rapport à la syntaxe du langage.
- Le document "VHDL Work Method", disponible en annexe, présente une proposition de convention pour nommer les signaux. Ce document a été fortement inspiré par les règles données dans l'ouvrage "Reuse Methodology Manual", qui ont ensuite été développées pour en faire une convention.

### **REGLES AU NIVEAU SYSTEME**

Ces règles s'appliquent lorsque le module réutilisable peut être considéré comme un système complet, qui pourrait se suffire à lui-même.

- 3.2.3.1 – Le nombre de domaines d'horloge ainsi que leurs fréquences sont bien documentés. Les timings externes à respecter (setup time, hold time...) ainsi que les PLL/DLL utilisées (ou à utiliser) sont bien définis. Les horloges au niveau système ne doivent utiliser que des arbres de distribution dédiés.
- 3.2.3.2 – Si 2 domaines d'horloge asynchrones doivent être utilisés, s'assurer qu'ils soient synchronisés dans un module simple qui devrait juste être composé de registres permettant de rendre les 2 domaines d'horloge synchrones.
- 3.2.4.1 – La stratégie de reset doit être documentée au niveau système, en particulier pour les points suivants :
- synchrone ou asynchrone ?
  - reset à la mise sous-tension interne ou externe ?
  - chaque bloc doit-il être pouvoir être remis à zéro individuellement (pour débogage) ?
- 3.4.1 – La stratégie de vérification au niveau système doit être développée et documentée avant de commencer la sélection (si le système est un assemblage de modules réutilisables) ou le codage des sous-blocs

- 3.4.2 – La stratégie de vérification au niveau des sous-blocs doit être développée et documentée avant d’en commencer la conception.
- 3.5.5.2 – Soigner la contrôlabilité et l’observabilité pour faciliter le débogage au niveau système. La contrôlabilité est facilitée en donnant la possibilité d’éteindre individuellement chaque sous-module, ou de le mettre en mode débogage (fonctionnement simplifié). L’observabilité est améliorée en ajoutant des bus permettant de contrôler l’état du système.

### **CODAGE POUR PORTABILITE**

- 5.3.1.1 – N’utiliser que des types définis dans les bibliothèques IEEE (ou des sous-types dérivés de bibliothèques IEEE).
- 5.3.1.2 – Utiliser les types `std_logic` et `std_logic_vector` plutôt que `std_ulogic` et `std_ulogic_vector`, moins bien supporté par les outils.
- 5.3.1.3 – Limiter le nombre de sous-types dérivés.
- 5.3.1.4 – Ne pas utiliser les types `bit` et `bit_vector`.
- 5.3.2.1 – Ne pas utiliser de valeurs numériques hard-codées autres que 0, 1 ou 2.
- 5.3.3.1 – Tous les paramètres ou fonctions communes à un design doivent être regroupées dans un package séparé nommé `design_name_pack.vhd`.
- 5.3.6.1 – Utiliser autant que possible des bibliothèques indépendantes de la technologie utilisée.
- 5.3.6.2 – Ne pas instancier de portes ou de composants spécifiques directement dans un design. Les encapsuler dans des sous-modules pour rendre le top-level indépendant de la technologie.

### **REGLES POUR CLOCK ET RESET**

- 5.4.4.1 – Les registres ne doivent être sensibles qu’à un seul front d’horloge (généralement montant). Exception possible pour les mémoires DDR.
- 5.4.3.1 – Pas de logique sur l’horloge (gated clock).
- 5.4.4.1 – Pas d’horloge générées en interne (ou donner un moyen d’utiliser une horloge externe en phase de débogage).
- 5.4.5.1 – Pas de reset générés en interne ou de reset conditionnels (ou donner un moyen d’utiliser un reset externe en phase de débogage).

#### Autres règles importantes :

- Un reset asynchrone est conseillé (plus facile à tester, et également moins de risques de problèmes à la mise sous tension).

### **CODAGE POUR LA SYNTHÈSE**

- 5.5.5.1 – Utiliser des registres indépendants de la technologie lors du codage style RTL pour la logique séquentielle.
- 5.5.2.1 – Le module doit être synchrone et basé sur des bascules D. Ne pas instancier de latches lors du codage RTL, particulièrement des bascules S-R.
- 5.5.2.2 – Lors d’une assignation conditionnelle, utiliser l’une des règles suivantes pour ne pas inférer de latches :
- Assigner une valeur par défaut en début d’un process,
  - Assigner une valeur de sortie pour chaque combinaison des entrées,
  - Terminer l’assignation conditionnelle par la condition `else` ou `when others`.

- 5.5.4.1 – Pas de retour combinatoire, i.e. de boucle sur un process combinatoire.
- 5.5.5.1 – Compléter toutes les listes de sensibilité des process :
  - L'horloge et les signaux asynchrones pour un process séquentiel,
  - Tous les signaux d'entrée d'un process combinatoire.
- 5.5.5.2 – N'indiquer que les signaux nécessaires la liste de sensibilité.
- 5.5.7.1 – Utiliser des signaux plutôt que des variables. Exception possible pour les indices de tableau ainsi que les valeurs intermédiaires accumulées lors de boucles.
- 5.5.8.1 – Utiliser des conditions case plutôt que des conditions if-then-else, quand approprié.
- 5.5.9.1 – La description d'une machine à états doit se faire en utilisant 2 process, un pour la logique séquentielle, un pour la logique combinatoire.
- 5.5.9.2 – Créer un type énuméré pour décrire le vecteur d'état, permettant à l'outil de synthèse d'en optimiser le codage.
- 5.5.9.4 – Pour une machine à état, assigner un état par défaut.

#### Autres règles importantes :

- Utiliser un pipeline massif, permettant d'atteindre des bonnes performances.
- Mettre un registre en sortie de chaque sous-bloc (si le design est complètement synchrone, ce registre existe déjà)
- Mettre un registre en entrée de chaque bloc (à partir d'une certaine complexité seulement, pas nécessaire pour les sous-blocs). Ceci permet de mieux prévoir les timings quelle que soit la complexité et la taille du système dans lequel est intégré le bloc.

### **PARTITIONNEMENT POUR FACILITER LA SYNTHÈSE**

- 5.6.1.1 – Pour chaque sous-bloc d'un design hiérarchique, chaque signal de sortie doit être enregistré en sortie.
- 5.6.3.1 – Isoler le chemin critique des chemins non-critiques, pour permettre aux outils d'optimiser différemment (vitesse pour le chemin critique, surface pour le reste).
- 5.6.4.1 – Éviter d'utiliser de la logique asynchrone, ou si absolument nécessaire, la séparer dans un module spécifique.
- 5.6.5.1 – Partitionner les calculs arithmétiques dans le même module, pour permettre de profiter du partage des ressources (resource-sharing).
- 5.6.8.1 – Pas de glue-logic au top-level.

### **CONFIGURABILITE**

La configurabilité est un des éléments-clés de la réutilisation. Plus un bloc sera configurable, plus il sera adapté à un usage général, plus il sera compatible avec des applications différentes et ainsi, plus il sera réutilisé.

Le but de cette étude n'est pas d'être exhaustif, mais plutôt de montrer les 2 principales manières de concevoir un bloc réutilisable.

1. Ajouter des caractéristiques supplémentaires au module.

Exemple : Caractéristiques possibles pour un filtre LP :

- coefficients génériques,
- taille générique,
- reset asynchrone, facultatif (en mettant une valeur par défaut pour les cas où l'on n'a pas besoin de remise à zéro),

- traitement possible de luminance seule, de chrominance seule, ou des 2 composantes multiplexées (au format 4:2:2).

2. Faire un design modulaire, en facilitant l'intégration de modules aux caractéristiques différentes. Il peut parfois être complexe d'intégrer un grand nombre de caractéristiques dans un même bloc. Créer plusieurs blocs aux caractéristiques différentes peut être plus facile. Dans ce cas, la configurabilité est obtenue en facilitant l'intégration des blocs et en permettant de remplacer un bloc par un autre qui aurait des caractéristiques différentes. Le système ainsi généré est configurable (à condition d'avoir à disposition les sous-blocs adéquats).

Exemple : Pour un filtre LP, il peut être difficile de concevoir un bloc aux coefficients et à la taille génériques qui soit optimal au niveau performances et complexité. Ainsi, on peut être amené à concevoir un certain nombre de filtres de aux coefficients et tailles différentes. Lorsque ces blocs sont intégrés dans un système, il est intéressant de faire une intégration générique pour faciliter le remplacement d'un filtre par un autre. Dans cet exemple, la généricité se ferait au niveau des lignes à délai, en permettant très facilement l'instanciation d'un nombre variable de lignes.

## **VERIFICATION**

La vérification est un des points-clés de la réutilisabilité, puisque seule une vérification rigoureuse permettra de donner à l'utilisateur la confiance nécessaire dans le bloc pour accepter d'intégrer le bloc dans son design. On estime en moyenne que le temps à consacrer à la validation d'un module ou d'un système devrait se situer autour de 70% du temps total de conception du bloc. Ceci est d'autant plus vrai lorsque le module doit être réutilisable.

Un module réutilisable devrait avoir un fonctionnement garanti à un niveau très proche de 100% pour toutes les configurations valides de celui-ci. Ainsi, la validation peut s'avérer extrêmement laborieuse si le module est fortement configurable.

Si on se ramène à notre stratégie de conception de modules réutilisable, en 2 étapes, on suggérera pour chacune des étapes :

- Conception initiale : On doit tester le module de manière rigoureuse, mais seulement pour l'application envisagée pour cette première utilisation. On ne testera donc pas toutes les caractéristiques possibles du bloc (dans le cas où elles existent déjà mais ne sont pas utilisées pour cette première utilisation). Le temps de validation n'est ainsi pas plus long que tester un module conçu seulement spécifiquement pour l'application.

- Re-conception : Il faut refaire un effort de validation pour valider vraiment toutes les caractéristiques que l'on veut garantir pour le module (ces caractéristiques peuvent être celles intégrées lors de la conception initiale mais pas utilisées la première fois ;