الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

وزارة التعليم العالي والبحث العلمي
جامعة أكلي محند أولحاج
-البويرة-

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université Akli Mohand Oulhadj - Bouira -
Tasdawit Akli Muḥend Ulḥağ - Tubirett -

جامعة البويرة

كلية العلوم والعلوم التطبيقية
Faculté des Sciences et des Sciences Appliquées

المرجـــــع:...../م م/ 2021
Référence : ...../MM/2021

# Mémoire de Master

**Présenté au :**

**Département:**Génie Électrique.

**Domaine:**Sciences et Technologies.

**Filière :**Electronique.

**Spécialité:** Electronique des systèmes embarqués.

**Réalisé par :**

**BOURAHLA Abla**

Et

**AZI Roumaissa**

# Thème

## Implémentation d'un réseau de neurone artificiel sur FPGA

Soutenu le: **30/10/2021**

Devant la commission composée de :

| | | | | |
|---|---|---|---|---|
| Mr : | Bouhedda Ali | M.A.A | Univ. Bouira | Président |
| | HAROUN Smail | M.C.B | Univ. Bouira | Rapporteur |
| | Mellah Hacene | M.C.B | Univ. Bouira | Examinateur |

# *Dédicace*

*Je dédie ce modeste travail à :*

*Mon père MOUHAMED, qui peut être fière de trouver ici le résultat de longues années de sacrifice pour m'aider a avancé dans la vie.*

*Merci pour les valeurs nobles, pour l'éducation et tous les efforts venus de toi.*

*Ma mère RAI FATIHA qui a ouvrée pour ma réussite, de par son amour, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie. L'expression de mes sentiments et de mon éternelle gratitude.*

*A mes chères sœurs BOUCHRA et NOURHANE et mes chers frères FARES et ABD ELMALEK.*

*A tous mes amis surtout l'étudiante CHERAREK FATMA.*

*J'exprime mes sentiments les plus profonds*

*A Mme IZBOUDJEN NOUMA.*

*A tous mes enseignants de master et licence.*

*A toute personne qui m'a encouragé ou aidé.*

*« ABLA »*

# *Dédicaces*

*Je tiens à dédier ce modeste travail à :*

*A ma très chère Mère et à mon cher Père, en témoignage et en gratitude de leurs dévouements, de leurs soutien permanent durant toutes mes années d'études, leurs sacrifices illimités, leurs réconforts moraux, eux qui ont consenti tant d'effort pour mon éducation, mon instruction et pour me voir atteindre ce but, pour tout cela et pour ce qui ne peut être dit, mes affectations sans limite…*

*A ceux qui sont la source de mon inspiration et mon courage, à qui je dois de l'amour et de reconnaissance :*

*A mes chers Frères.*

*A ma chère Sœur.*

*A toute ma famille.*

*A mes chers Amis pour tous les moments de joie et de peine qu'on a passée ensemble, A leurs Familles aussi.*

*A vous tous un grand merci*

*A. ROUMAISSA*
*« ROUFA »*

# Remerciements

Avant tout, nous louons Allah le tout-puissant pour nous avoir donné la patience et la volonté afin de mener à bien ce modeste travail.

Je tiens à remercier, en premier lieu, **Mme. IZEBOUDJEN Nouma,** de nous avoir proposé ce sujet qui nous a mené vers la découverte du monde de la recherche. Nous la remercions pour ses conseils et leurs précieuses orientations qu'ils n'ont cessé de nous apporter tout au long de ce travail.

De même, nous adressons notre remerciement à notre promoteur **Dr HAROUN Smail** pour sa confiance, son aide et soutien et tous ces conseils au long de notre travail.

Nos remerciements également vont aux membres du jury pour l'intérêt qu'ils ont porté à notre travail.

Enfin, Nous associons à ces remerciements tous ceux qui ont contribué à réaliser ce travail.

# Résumé

Les travaux présentés dans ce thème portent essentiellement sur la conception et l'implémentation d'un réseau de neurone artificiel (RNA) sur une carte FPGA.

Nous avons ciblé comme application dans ce travail la conception et la simulation d'un réseau de neurone par l'apprentissage sous MATLAB, et l'implémentation d'un classificateur neuronal de type Feed-Forward sur une carte FPGA de la famille Artix7 par l'utilisation du langage VHDL sous l'environnement Xilinx VIVADO.

**Mots clés** : Réseau de neurone, Implémentation, Conception, FPGA, Feed-Forward, VHDL.

# Table des Matières

# Liste des Figures

# Liste des Tableaux

# Listes des Acronymes

**ANN :**Artificial Neural Network.

**CAO :** Conception Assisté par Ordinateur.

**CLB** : Blocs Logiques Configurables.

**DOD :** Département de la défense américaine.

**DSP**: Digital Signal Processor.

**FF :** Flip Flop

**FPGA**: Field Programmable Gate Arrays.

**IOB**: Input Output Bloc.

**LCA**: logic cells arrays.

**LUT**: Look Up Table

**LEARNGDM**: Learning gradiant descent with momentum backpropagation.

**LOGSIG :** tangent sigmoïde

**MAC** : multiplieur et accumulateur

**MLP**:Multi-Layer Perceptron.

**MSE:**Mean Squre Error.

**RAM**: Read Accès Memory.

**ROM**: Read Only Memory.

**RNA**: Réseau de Neurone Artificielle.

**RPG** : rétro-propagation du gradient.

**RTL**: Register Transfer Level description.

**SoC:** System on Chip.

**TRAINGDM**: Train gradient decsent with momentum backpropagation.

**VHDL**: VHSIC Hardware Description Langauge.

**VHSIC**: Very High-Speed Integrated Circuit.

**USB** : Universal Serial Bus.

# Introduction Générale

Les réseaux de neurones artificiel (RNA) ensembles d'opérateurs non linéaires interconnectés entre aux, forment une famille de fonction non linéaire permettent de construire une large classe de modèles et de correcteurs par l'apprentissage, ils ont de mémoriser la connaissance à partir d'exemple. Leurs fonctions sont similaires à celle de cerveau humain dans la mesure où les connaissances sont acquises par l'apprentissage et utilisent les poids de connexion entre les neurones pour mémoriser ces connaissances sous une forme réduite. Ces propriétés rendent les réseaux de neurones adaptatifs aux environnements changeants, résistants aux bruits et tolérants aux pannes [1].

Dans ce travail, nous allons faire l'implémentation d'un réseau de neurones sur un composant qui est toujours sujet de recherche qui s'appelle FPGA(Field Programmable GateArray). Pour cela nous avons divisé ce mémoire en deux parties, une partie théorique et une partie pratique et nous avons suivi le plan suivant :

Le premier chapitre sera consacré à une étude générale des réseaux de neurones et leurs différents domaines d'application. Nous abordons le circuit ciblé pour l'implémentation, le FPGA, avec son architecture et les différentes méthodes de sa programmation ainsi que le langage VHDL.

Ensuite, dans le second chapitre nous présentons les étapes principales de la conception d'un réseau de neurone sur MATLAB.

Dans le troisième chapitre, nous étudieront l'implémentation de RNA basé sur le module Feed-Forward proposé par le centre de recherche CDTA. Nous parlerons aussi sur l'outil de conception VIVADO utilisé pour la réalisation de ce travail. Les étapes d'implémentation seront aussi développées dans ce chapitre. Et en fin nous terminons ce travail par une conclusion générale.

## 1. Introduction

Ce chapitre est consacré à une présentation générale des RNA et leur utilisation, des circuits FPGA, des lagunages VHDL. Tout d'abord, nous présenterons un neurone (biologique et formel) qui est l'élément de base des RNA ; par la suite nous exposerons les principaux types de RNA et nous en étudierons un aperçu de quelques leurs domaines d'application. Par la suit nous présenterons les circuits FPGA et leur architecture ainsi que la différente méthode de programmation, le langage de description VHDL.

## 2. Les Réseaux de neurones

### 2.1. Du neurone biologique au neurone artificiel (formel) :

#### 2.1.1. Neurone biologique :

En biologie ; les neurones sont des cellules du tissu nerveux qui constituent l'unité fonctionnelle du système nerveux, généralement les neurones sont constitués de trois parties :

- **Les dendrites :** sont des réseaux de fibres nerveuses qui collectent les informations en provenance des autres neurones.
- **Le corps cellulaire (le soma)** : il somme les signaux reçus et renvoi une impulsion en sortie.
- **L'axone** : est une fibre longue qui conduit le signal sortant de la cellule aux autres neurones.

Le relais entre deux neurones est appelé la synapse, il assure la transmission de l'influx nerveux.

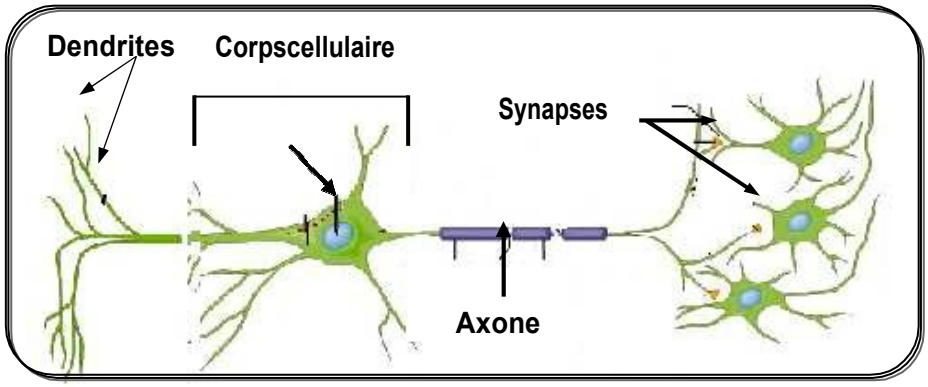



**Figure 1.1 :** Neurone biologique.

### 2.1.2. Neurone artificiel (formel) :

Warren Mc Culloch (neuropsychiatre), walterPitts (logicien) voulaient simuler de façon simplifiée le neurone naturel. Dans leur approche, le neurone formel est un automate non linéaire à seuil comportant plusieurs entrées et une sortie [2].

Le neurone artificiel est un processus de calcul qui fait la somme des signaux d'entrées en provenance de neurones et renvoie une fonction f de cette somme en sa sortie appartenant à un niveau situé en amont ; à chacune de ces entrées est associée une abréviation de weight (poids en Anglait) représentatif de l'entrée X, dont la valeur est poids W .chaque neurone a une fonction de transfert qui donne une sortie unique Y[3].

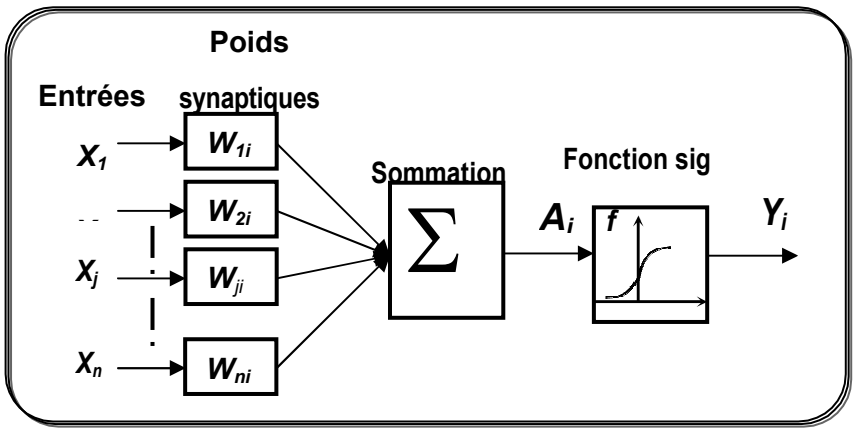



**Figure 1.2** : Model mathématique d'un neurone artificiel.

La somme pondérée des entrées $\mathbf{A_i}$ :

$$\mathbf{A_i} = \sum \mathbf{W_{ji} \cdot X_j} \qquad \textbf{(1.1)}$$

Une fonction seuil calcule la valeur de l'état de neurone à partir de (1.1) :

$$\mathbf{Y_i = f\,(A_i)} \qquad \textbf{(1.2)}$$

Où :

$\mathbf{X_j}$ : Les entrées.

$\mathbf{Y_i}$ : L'état d'un neurone i.

$\mathbf{A_i}$ : L'activité de neurone i.

$\mathbf{W_{ji}}$ : Les poids synaptiques entre les neurones i et j.

La transition entre le neurone biologique et le neurone artificiel est décrire dans le tableau suivant :

| **Neurone biologique** | **Neurone artificiel** |
|---|---|
| Synapse | Poids de connexion |
| Axones | Signal de sortie |
| Dendrite | Signal d'entrée |
| Somma | Fonction d'activation |

**Tab 1.1:**la transition entre le neurone biologique et le neurone artificiel.

### 2.2. La fonction d'activation :

Cette fonction permet de définir l'état interne du neurone en fonction de son entrée totale.

#### 2.2.1. Fonction binaire ou seuil :

Quelques exemples des fonctions binaires à seuil :

**a) Fonction Heaviside :**

$$h(x)=\begin{cases}1 & \text{si } x \geq 0\\ 0 & \text{sinon}\end{cases}$$

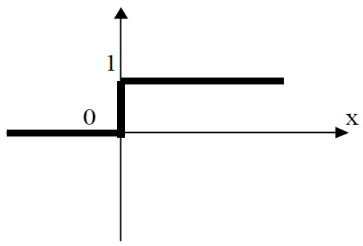



**Figure1.3** : Fonction Heaviside.

**b) Fonction signe :sgr(x)**

$$Sgr(x)=\begin{cases}+1 & \text{si } x \geq 0\\ -1 & \text{sinon}\end{cases}$$

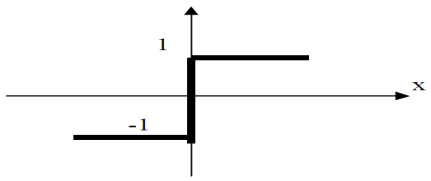



**Figure1.4** : Fonction signe.

#### 2.2.2. Fonction linéaire ou identité :

$$L(x)=x$$

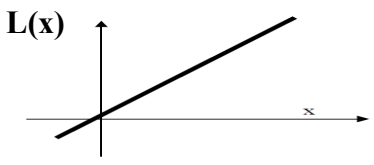



**Figure1.5** : Fonction identité.

### 2.2.3. Fonction linéaire a seuil ou multi-seuils :

$$F(x) = \begin{cases} x & x \in [\,u, v\,] \\ v & \text{si } x \geq v \\ u & \text{si } x \leq u \end{cases}$$

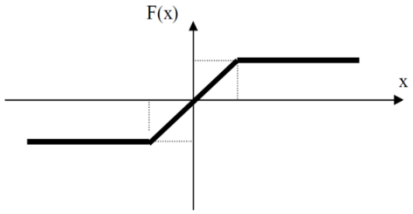



**Figure 1.6** : Fonction linéaire à seuil.

### 2.2.4. La fonction sigmoïde :

$$f(x) = \frac{1}{1 + e^{-x}}$$

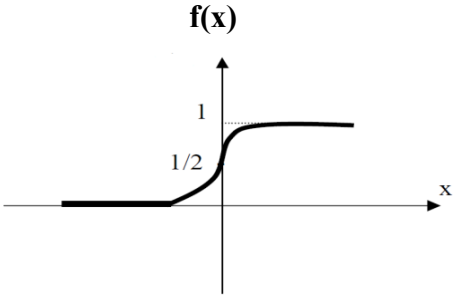



**Figure1.7 :** Fonction sigmoïde.

### 2.3. Réseaux de neurones artificiels :

Un neurone élémentaire est très limité, en effet, un neurone réalise une simple fonction non linière, paramétrée de ses variables d'entrée ; le rôle des neurones apparait dans la propriété qui réside en leur association dans une structure par une certain logique d'interconnexion ; c'est le réseau de neurone artificielle par abréviation ANN (Artificial Neural Network). [4]

Le comportement collectif permet de réaliser des fonctions d'ordre supérieur par rapport à la fonction élémentaire réalisée par un neurone simple. Dans un tel réseau, les entrées d'un neurone sont soit les entrées du réseau global, soit les sorties d'autres neurones. Le pouvoir de traitement du réseau est stocké dans les synaptiques dont les valeurs sont en générale déterminées par une opération dite l'apprentissage.

La structure d'interconnexion entre les différents neurones détermine la topologie du réseau. On peut classer les réseaux de neurones selon deux (2) critères différents [5] :

- Selon l'architecture.
- Selon le mode d'apprentissage.

### 2.3.1. Classification selon l'architecture :

L'architecture ou la topologie d'un RNA est la manière selon laquelle les neurones sont organisés ; on distingue trois architectures :

#### 2.3.1.1 Les réseaux de neurones non bouclés :

Les réseaux de neurones non bouclés sont des réseaux de type « perceptron » ou «Feed-Forward », ces réseaux sont caractérisés par la propagation des signaux dans un seul sens de la couche d'entrée vers la couche de sortie ; leurs entrées et sorties sont indépendantes du temps alors, le réseau est statique. [6]

Ce type de réseau de neurone artificiel est utilisé généralement pour la classification ou de modélisation de processus statique (Figure 1.8)

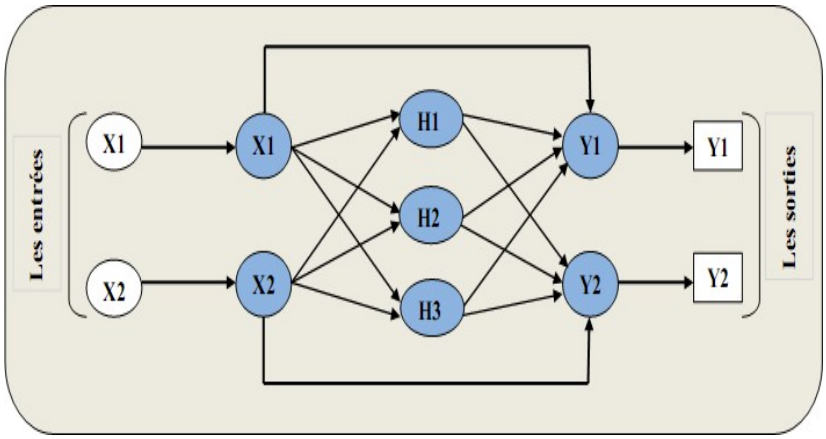



**Figure 1.8 :** Architecture d'un réseau de neurone non bouclé.

#### 2.3.1.2 Réseau de neurones bouclés :

Appelées aussi réseaux « récurrente » ou « feed-back », ces réseaux sont dynamiques ; ils se caractérisent par la présence, au moins, d'une boucle de rétroaction des neurones de sorties vers les neurones d'entrée ; ce type de réseau de neurone artificiel est utilisé principalement pour effectuer des taches de modélisation de système dynamique de commande de processus, ou de filtrage (Figure1.9). [6]

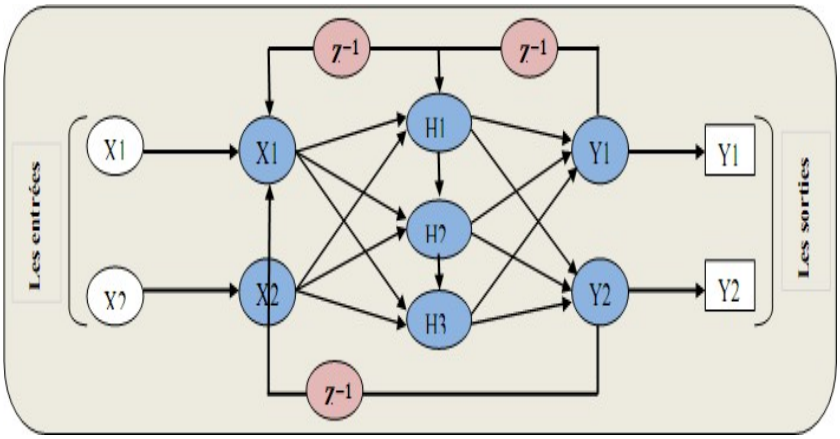



**Figure 1.9** : Architecture d'un réseau de neurone bouclé.

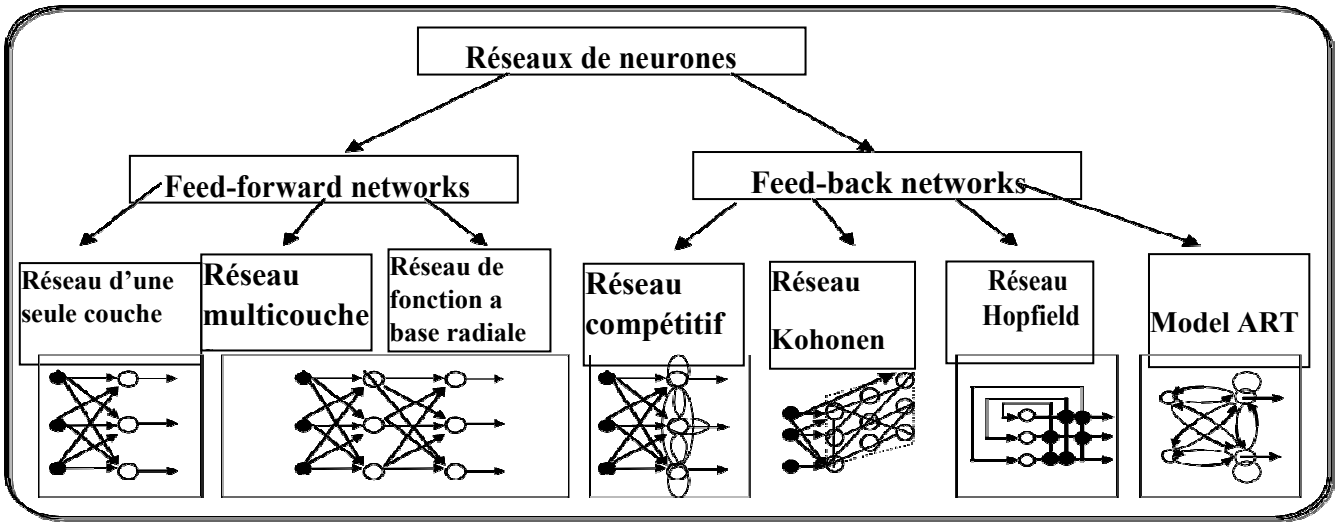



**Figure 1.10** : Différentes architectures des réseaux de neurones.

### 2.3.2. Classification selon l'apprentissage :

#### 2.3.2.1 Définition :

L'apprentissage est une phase de développement de modèles à base des RNA. Il est réalisé par la modification des poids de connexion du réseau par des algorithmes spécifiques, afin d'obtenir des valeurs optimales appropriées à ces poids [7].

#### 2.3.2.2. Principe

Les caractéristiques du réseau peuvent être modifiées en réaction aux stimuli extérieure que nous lui soumettons, de manière à ce qu'il réagisse si un même stimulus lui est appliqué ultérieurement. Le réseau subit une correction qui le fait réagir différemment s'il confronté à la même situation, donc il va s'améliorer pour chaque erreur qu'il fait. Si l'apprentissage est terminé, le réseau va fournir pour chaque stimulus d'entrée, la sortie désirée par l'opérateur [8].

### 2.3.2.3. Apprentissage supervisé

Dans l'apprentissage supervisé, la fonction d'erreur est minimisée à chaque itération de l'algorithme ; il consiste à comparer le résultat obtenu avec le résultat désiré pour qu'il puisse adapter les poids synaptiques de ce réseau dans le but d'obtenir la sortie souhaitée.[6]

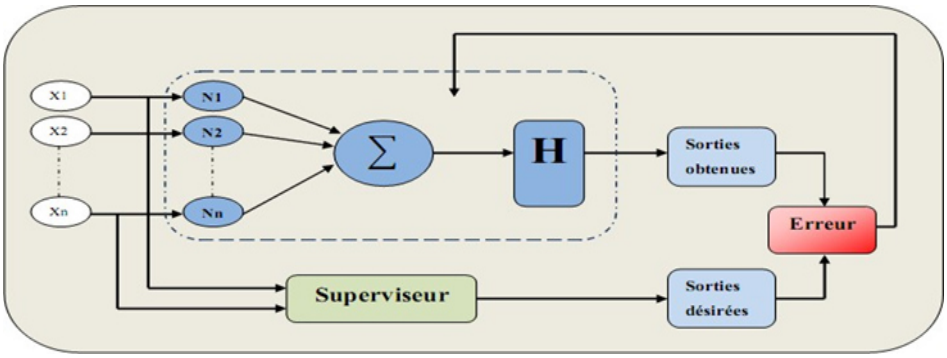



**Figure 1.11 :** Mode d'apprentissage supervisé.

### 2.3.2.4. Apprentissage non supervisé

Ce type d'apprentissage consiste à présenter au réseau seulement les vecteurs d'entrée qui seront groupés en classes par extraction des propriétés, aucune réponse désirée n'est prise en considération la Figure 1.12 montre que la sortie n'est pas utilisée par la procédure de l'apprentissage [6].

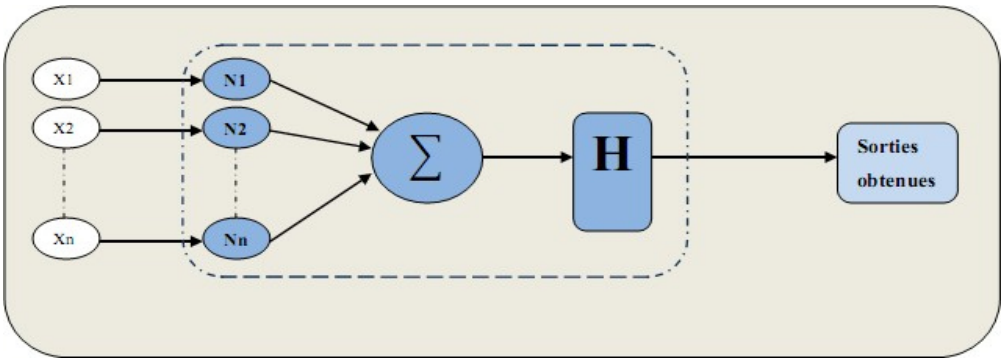



**Figure 1.12 :** Mode d'apprentissage non supervisé.

### 2.3.2.5. Apprentissage renforcé

Ce type d'apprentissage élabore une critique au résultat de chaque exemple traité par la machine d'apprentissage, le vecteur des variables présentés dans la Figure I.13 exprime les indications imprécises sur le comportement final utilisé par l'algorithme de critique [9].

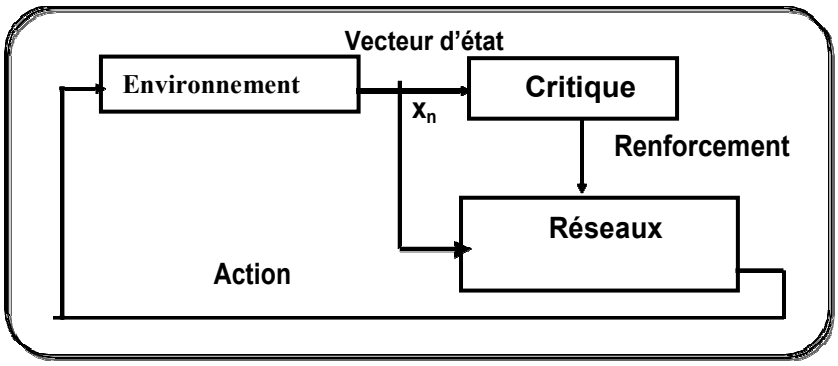



**Figure 1.13 :** Mode d'apprentissage renforcé.

## 2.4. Les avantages et les inconvénients de réseau de neurone

### 2.4.1. Les avantage

- Apprentissage.
- Implémentation du parallélisme.
- Robustesse : données bruitées ou incomplètes.
- Généralisation a des modèles similaires.
- Trouve des solutions aux problèmes.
- Trouvent des solutions aux problèmes qui n'ont pas une modélisation.

### 2.4.2. Les inconvénients

- N'ont pas encore expliqué le fonctionnement du cerveau.
- Les poids ne sont pas interprétables.
- L'apprentissage n'est pas toujours évident.
- Ne sont pas extensible (l'ajout d'un neurone).

## 2.5. Domain d'application et mise en œuvre

- Robotique et capteurs
- La modélisation de processus dynamiques
- La commande de processus
- Traitement de signal
- La classification et reconnaissance des formes

## 3. LES CIRCUITS FPGA

### 3.1. Définition

FPGA (Field Programmable Gate Arrays) est un circuit intégré composé d'un réseau de cellules programmables reliées entre eux grâce à une matrice de routage. Chaque cellule est capable de réaliser une fonction désirée.

### 3.2. Architecture de FPGA

Un circuit FPGA appelé aussi LCA (logic Cells Arrays) signifiant réseau de cellule logique, il contient des blocs logiques programmables CLB permettant de réaliser des fonctions combinatoires et des fonctions séquentielles ; et blocs d'entrées/sorties IOB positionnées sur la périphérie du composant dont le rôle est de gérer les entrée-sorties réalisant l'interface avec les modèles extérieurs.

Dans les FPGA le temps de propagation dans les couches logiques de circuit dépend de l'organisation et de la distance entre les macro-cellules interconnectées.

La structure d'un FPGA diffère d'un constructeur à un autre mais elle garde la même architecture globale illustrée par la figure suivante :

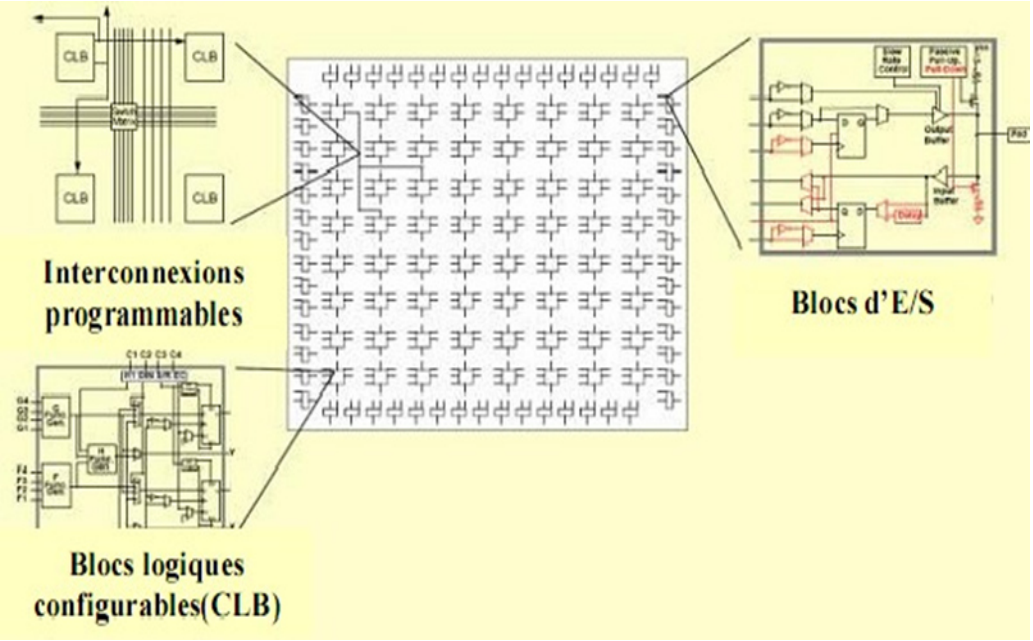



**Figure 1.14 :** Architecture interne du FPGA.

### 3.3. Programmation du circuit FPGA

Une première étape dans la programmation d'un circuit FPGA consiste à connecter les entrées/sorties (routage du programme) et les blocs logiques entre eux.

Les circuits FPGA disposent de plusieurs ressources de routage. En générale, les niveaux

Hiérarchiques disponibles sont :

- Connexions directes vers les voisins proches.
- Connexions générales à travers des matrices de routages et des canaux disposés suivant une topologie simple
- Connexions à longue distance
- Distribution d'horloge spécifique

La seconde étape de programmation des circuits FPGA est la définition des connexions entre les switches et les blocs logiques. Cet outil de design traduit le langage introduit par l'utilisateur (VHDL, Verilog).

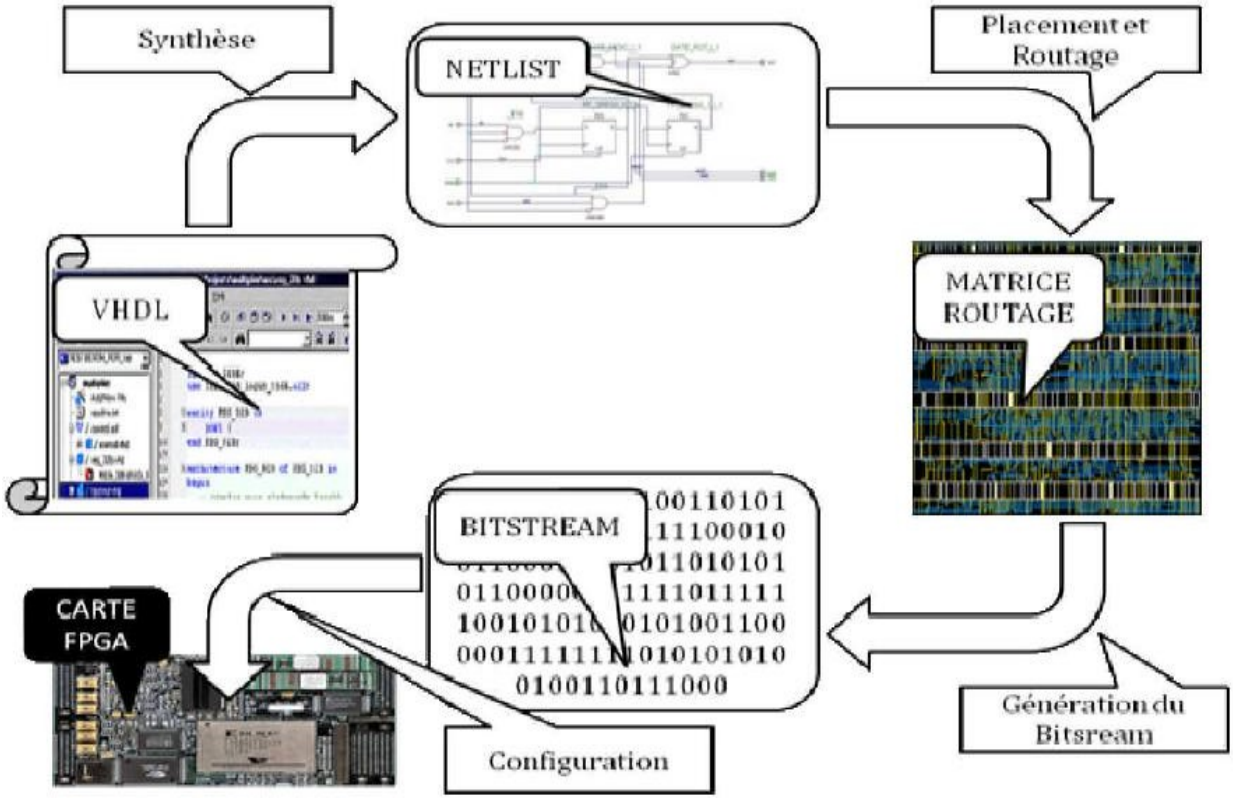



**Figure 1.15 :** Cycle de programmation de FPGA.

### 3.4. Application des FPGA

Les circuits FPGA sont utilisées généralement dans :

- Prototypage de nouveau circuit.
- Fabrication de composants spéciaux.
- Adaptation aux besoins rencontrés lors de l'utilisation.
- DSP(Digital Signal Processor)
- Imagerie médicale …

### 3.5. Description de FPGA famille XilinxZynq 7000

La famille Zynq 7000 est un circuit basé sur l'architecture SoCXilinx All Programmable. Cette famille fait partie de la famille série 7 des FPGA dont les performances basculent entre les deux familles Artix (7010, 7015,7020) et Kintex (7030,7045 et 7100).

La famille Zynq 7000 caractérisé par :

- 85000 slices qui comprennent de la logique combinatoire et ressources de registre.
- 560 KB bloc de RAM.
- 220 DSP slices.

Cependant, cette famille intègre un système de traitement basé sur ARM Cortex A9 qui comporte soit deux cœurs ou un seul cœur riche en fonctionnalités avec une logique programmable d'un FPGA Xilinx de 28 nm dans un seul puce.

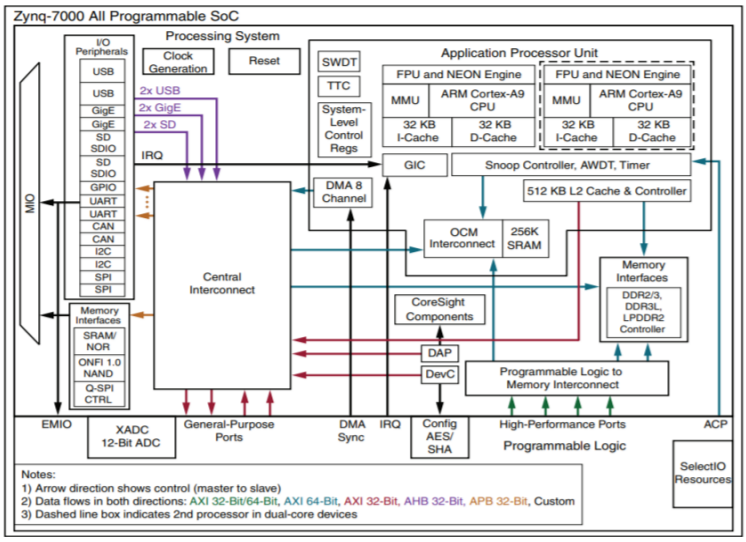



**Figure 1.16 :** Architecture de l'FPGA Zynq 7000.

### 3.6. Description de la carte Nexys4 DDR de FPGA Artix-7de Xilinx

La carte Nexys4 DDR peut être alimentée par une prise murale 5V ou par micro USB connecteur. Elle est caractérisée par :

- 15850 slices et 8 bascules (flip-flops).
- 4860 KB bloc de RAM.
- 170 blocs d'I/O.
- Fréquence d'horloge interne de 450 MHz.

La carte s'intègre à Xilinx Vivado Design Suite, fournissant des fichiers de contraintes pour tous les connexions matérielles [10].

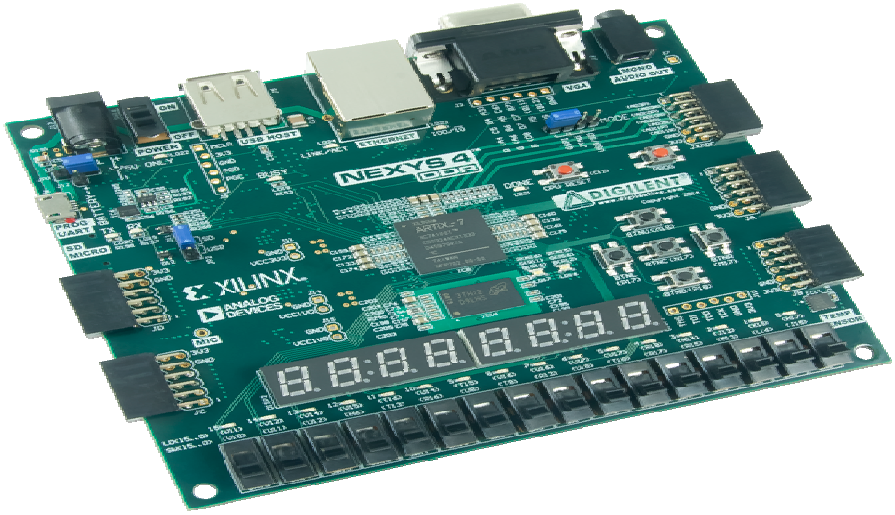



**Figure 1.17 :** La carte Nexys4 DDR.

## 4. LANGUAGE DE DESCRIPTION VHDL

### 4.1. Historique

VHDL (Verry High Speed circuit Hardware Description Langage) a été commandé par le DOD (Département de la défense américaine) pour décrire les circuits complexes, de manière à établir un langage commun avec ses fournisseurs. C'est un langage standard IEEE1076 depuisb1987, qui aurait dû assurer la portabilité du code pour les différents outils de travail (simulation, synthèse, pour tous les circuits et tous les fabricants). La mise à jour de langage VHDL s'est faite en 1993 (IEEE 1164) et en 1996, la norme 1076.3 a permis de standardiser la synthèse VHDL [5].

### 4.2. Les avantages de VHDL

L'ambition des concepteurs du langage est de fournir un outil de description qui permet de créer des modèles de simulation. Initialement réservé au monde des circuits numériques, VHDL est en passe d'être étendu aux circuits analogiques.

Les intérêts majeurs du langage sont :

- Des niveaux de description très divers : VHDL permet de représenter le fonctionnement d'une application tant du point de vue système que du point de vue circuit, en descendant jusqu'aux opérateurs les plus élémentaires. A chaque niveau, la description peut être structurelle (portrait des interconnexions entre des sous-fonctions) ou comportementale (langage évolué).
- Son aspect : le développement des circuits logique a conduit chaque fabriquant à développer son propre langage de description. VHDL est en passe de devenir un langage commun aux nombreux outils de CAO, indépendants ou liés à des producteurs de circuits, des outils d'aide à la programmation.

### 4.3. Structure d'une description VHDL simple

La structure typique d'une description on VHDL es composée de **:**

#### 4.3.1. L'entité

L'entité (ENTITY) est vue comme une boite noire avec des entrées et des sorties caractérisées par des paramètres. Elle représente une vue externe de la description.

#### 4.3.2. L'architecture

L'architecture (ARCHITECTURE) contient les instructions VHDL permettant d'écrire et de réaliser le fonctionnement attendu. Elle représente la structure interne de la description.

#### 4.3.3. Déclaration des bibliothèques

Tous la description VHDL sont utilisés pour la synthèse à besoin de la bibliothèque. L'IEEE les a normalisées et plus particulièrement la bibliothèque IEEE1164. Elles contiennent les définitions des types de signaux électronique**,** des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques, etc

### 4.3.4. Déclaration de l'entité et des entrées/sorties

Cette opération permet de définir le nom de la description VHDL ainsi que les entrées et sorties utilisées, l'instruction qui les définit est **port :**

Le signal peut être définit en entrée : (**in**), en sortie (**out**) ; en entrée/sortie (**inout**) ou en bouffer.

### 4.4. Les déférents styles de description d'une architecture

Les trois descriptions principales sont :

- Description par flot de données.
- Description comportementale.
- Description structurelle

Il existe aussi des styles de descriptions mixtes, ces derniers combinent les trois styles de descriptions citées auparavant, et enfin une description très importante pour la simulation qui est l'architecture de test.

## 5. Conclusion

A la fin de ce chapitre, nous avons étudié les réseaux de neurones artificiels d'une manière générale, nous avons vu leur classification selon l'architecture ou le mode d'adressage ; nous avons mentionné quelques limites et avantages, par la suite nous allons étudier les FPGAs et le langage de programmation VHDL que nous allons utiliser dans la partie pratique.

## 1. Introduction

La conception software d'un réseau de neurone est utilisée généralement pour déterminer la taille de ce réseau, le coefficient d'apprentissage, la matrice de poids synaptique.

Nous présentons dans ce chapitre les définitions de l'implémentation et l'implémentation software ainsi que l'algorithme et les règles de calcul utilisés pendant la conception d'un réseau de neurone sous Matlab ; par la suite nous avons choisir quelques exemples pour montrer les procédures systématiques pour la conception d'un modèle ANN.

## 2. Définitions

### 2.1 L'implémentation

Le terme d'implémentation peut présenter sous plusieurs synonymes, les plus proches et les plus justes sont : la construction, l'adaptation, l'exécution, la fabrication, l'intégration, la réalisation ; donc on peut définir l'implémentation par la mise en œuvre d'un outil informatique à partir de document.

L'implémentation d'un ANN est le développement et l'intégration d'un algorithme de réseau de neurone dans un outil informatique.

Le schéma ci-dessous montre un exemple d'implémentation d'un réseau de neurones :

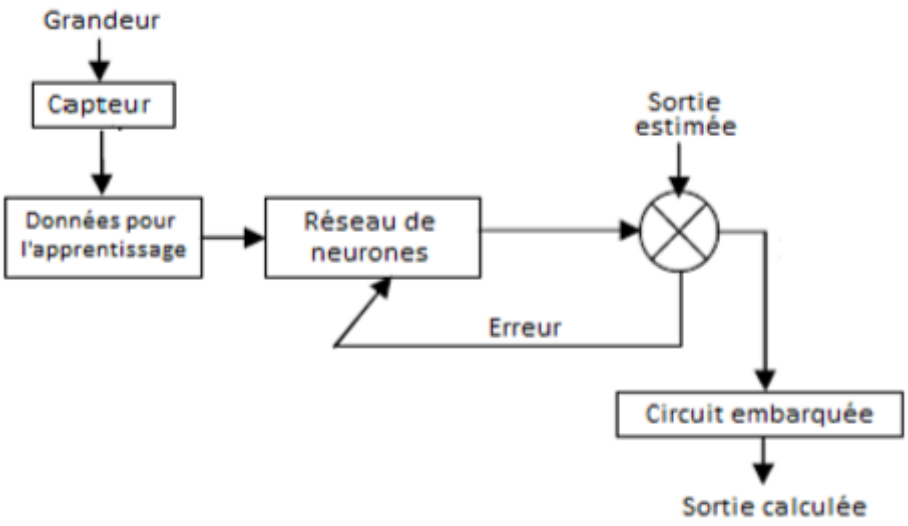



**Figure 2.1 :** schéma générale d'implémentation d'un ANN.

## 2.2 Implémentation software

L'implémentation software considère les deux phases : apprentissage et généralisation de l'algorithme ;pour élaborer ces algorithmes, on doit utiliser de modèle simple du neurone et de synapses.

Par conséquent, la réalisation de tel algorithme peut être formulée par 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 modèle de la machine série de Von Newman dans lequel une seule instruction est exécuté à la fois (1 seul MAC). Les simulateurs de réseau de neurones existants a l'heure actuelle (exp. MATLAB), offre à l'utilisateur des bibliothèques et un environnement flexible lui permettant de s'adapter à son application.

L'implémentation hardware parallèle est utilisée puisque le parallélisme spatio-temporel de réseau de neurone n'est pas exploité, sauf à moindre degré dans le cas de processeur RISC [2].

## 2.3 L'algorithme de rétro-propagation du gradient d'erreur [11]

L'apprentissage supervisé consiste à résoudre progressivement la différence entre les valeurs de sortie désirée et les valeurs de réseau. Il se fait par la modification progressive des poids synaptiques. Pour cela, on propage à partir de l'erreur constatée sur la couche de sortie, la correction apportée à tous ; cette technique s'appelle ***la rétro-propagation du gradient de l'erreur RGP.***

La règle du gradient de l'erreur est développée pour résoudre les problèmes de traitement de signal a ensuite été exploitée pour obtenir l'algorithme de rétro propagation du gradient de l'erreur (back propagation) pour les réseaux de neurone multicouche.

L'algorithme de rétro propagation est pour minimiser une fonction de cout 'E' exprimé par :

$$\mathbf{E=}\sum_{k}(\boldsymbol{d_k}-\boldsymbol{S_k})^2 \quad \mathbf{(2.1)}$$

Avec : $\boldsymbol{d_k}$la sortie désirée pour le neurone d'indice k et $S_k$ la sortie obtenue par le réseau.

L'apprentissage comporte la phase de propagation qui permet de calculer la sortie de réseau en fonction de l'entrée ; où chaque neurone effectue la somme pondérée des entrées et applique ensuite la fonction d'activation f pour obtenir la mise à jour du neurone, tel que :

$$S_i = f(pi) \qquad \text{Oùpi} = \sum_{j=0}^{n} (w_{ij}.x_j) \tag{2.2}$$

Avec :**pi**le potentiel post-synaptique du neurone i.

$x_j$L'état de neurone de la couche précédente.

$w_{ij}$ Le poids de la connexion entre les deux neurones.

L'algorithme de rétropropagation consiste à calculer une descente de gradient sur le critère 'E' ; le gradient de E est calculé pour tous les poids par :

$$\frac{\partial E}{\partial Wi_j} = \frac{\partial E}{\partial pi} \frac{\partial pi}{\partial Wi_j} = \frac{\partial E}{\partial pi} x_j \tag{2.3}$$

Le gradient sera ensuite noté par $C_i$ avec :

$$\mathbf{C1} = -\frac{\partial \mathbf{E}}{\partial \mathbf{p1}}$$

On distingue deux cas, suivant que le neurone d'indice i est neurone de sortie ou non. Pour la couche de sortie le gradient lié aux cellules de sortie est obtenu par l'équation suivante :

$$\mathbf{C_1} = -\frac{\partial E}{\partial p1} = -\frac{\partial}{\partial p1} (\sum_{K} (d_K - S_K)^2) = 2(d_i - S_i).\, f'(p1) \tag{2.4}$$

Puisque$S_i$dépend de **pi**avec $S_i = f(pi)$

Pour les neurones de couches cachées, l'ordre de calcul du gradient est l'inverse de celui utilisé pour la mise à jour des états dans le réseau ; il s'effectue de la couche de sortie vers l'entrée c'est la rétro-propagation ; l'expression de gradient est alors :

$$\mathbf{C_1} = -\frac{\partial \mathbf{E}}{\partial \mathbf{p_1}} = -\sum_{\mathbf{k=0}}^{\mathbf{n}} \frac{\partial \mathbf{E}}{\partial \mathbf{Pk}} \frac{\partial \mathbf{Pk}}{\partial \mathbf{p_1}} = \sum_{\mathbf{k=0}}^{\mathbf{n}} \mathbf{C_K} \frac{\partial \mathbf{pk}}{\partial \mathbf{p_1}} = \sum_{\mathbf{k=0}}^{\mathbf{n}} \mathbf{C_K} \frac{\partial \mathbf{pk}}{\partial \mathbf{Si}} \frac{\partial \mathbf{Si}}{\partial \mathbf{p_1}} \tag{2.5}$$

Soit encore :$C_1 = f'(p1) \sum_{k=0}^{n} (w_{K1}.C_K)$

Avec$C_K$le gradient du neurone K de la couche suivante.

La modification des poids est présentée par :

$$W_{ij}^{t+1} = W_{ij}^{t} + \alpha\,.\,C_1.S_j \tag{2.6}$$

Avec **α :** est un nombre petit, positif qui représente le pas de déplacement.

L'apprentissage s'arrête lorsque l'erreur calculée sur l'ensemble de la base d'apprentissage soit inférieure à un seuil déterminé par l'utilisateur.

### 2.4 La règle de correction d'erreur

La règle de correction d'erreur s'inscrit dans le modèle d'apprentissage supervisé ; autrement dit elle s'inscrit où l'on fournit au réseau une entrée et la sortie souhaité.

Si en prend Y comme étant la sortie calculée par le réseau et D la sortie désirée ; cette règle va utiliser l'erreur (D-Y) pour modifier les poids des connexions et pour diminuer ainsi l'erreur globale du système ; le réseau va s'adapter jusqu'à que Y soit égal à D.

## 3. Description de l'outil MATLAB

MATLAB est un langage de programmation, développé par la société The MathWorks.

L'outil MATLAB permet de faire des calculs numériques tels que les opérations sur des matrices, d'afficher des courbes, de mettre en œuvre des algorithmes ; le logiciel est le plus utilisé dans le monde d'ingénierie, il touche plusieurs domaines tels que les systèmes automobiles, les dispositifs de surveillance de la santé, les réseaux électriques intelligents, l'apprentissage automatique, la robotique, le traitement du signal, le traitement d'images, les systèmes de communications, les finances, et bien plus encore ;

Il peut s'utiliser avec des boites à outils « toolbox »à chaque domaine est associé une ou plusieurs boites à outils spécifiques, par exemple dans le domaine des Mathématiques, Statistiques,et d'Optimisation on trouve la : Partial Différentiel Equation Toolbox (pour les équations différentiels) ou là : CurveFitting Toolbox (pour l'ajustement des courbe) ou encore la : Neural Network Toolbox (pour les réseaux de neurones) qu'on a l'opportunité d'expliquer ultérieurement.

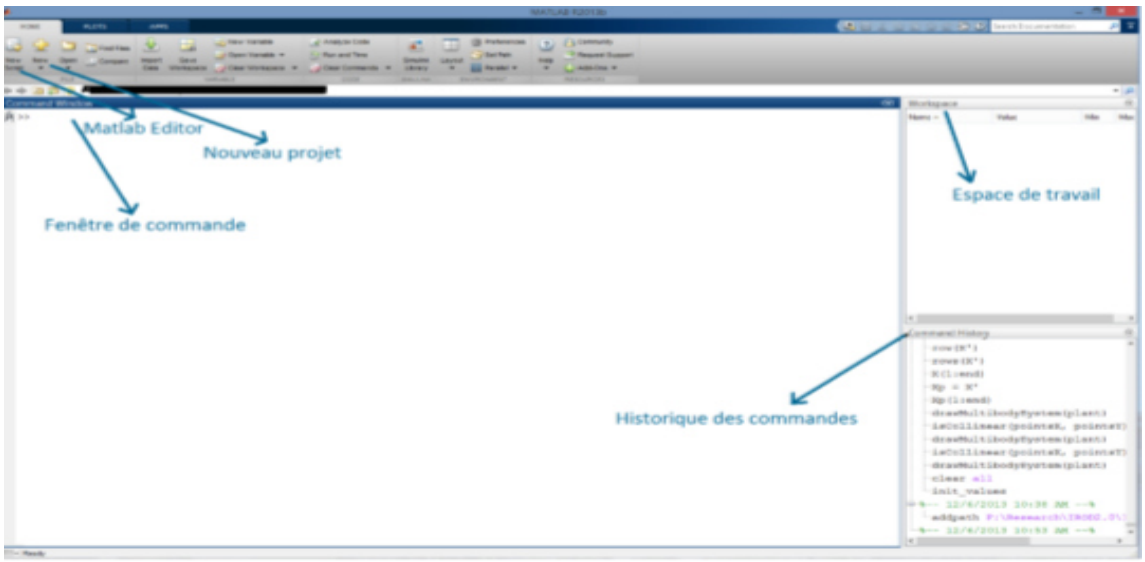



**Figure 2.2 :** Interface MATLAB principale.

- **Nouveau projet (New)** : pour ouvrir une nouvelle fenêtre de commande Window.
- **Fenêtre de commande (command Window)** : est une zone pour saisir les instructions de programme une après l'autre.
- **Editeur (MATLAB Editor) :**est une zone pour écrire les instructions à la fois.
- **Espace de travail (Workspace)** : Dans cette zone, on verra toute les variables et les constantes qu'on utilise actuellement dans le Command Window, leurs noms, leurs formats, leurs tailles, et leurs types y seront aussi affichées.
- **Historique des commandes (Command History) :** cette zone pour afficher un journal d'état qu'on a exécuté dans les sessions de MATLAB.

## 4. Construction d'un réseau de neurones sur MATLAB

La conception d'un modèle ANN suit la procédure systématique suivante :

### 4.1. Collecte de données

La première étape de conception des modèles ANN est la collection et la préparation des échantillons de données.

### 4.2 Construire le réseau

#### 4.2.1. L'apprentissage

MATLAB nous aide a créé un réseau de neurone de propagation rétroactif basé sur le modèle MLP en utilisant la fonction intégrée **'newff'** qui initialise automatiquement les poids et les biais.

Cette commande est appelée comme suit : [12]

**newff :** créé un réseau de Feed-Forward-backpropagation et initialise automatiquement le réseau.

> **net = newff (P, T, S) ;**
> **net = newff (PR, [S1 S2 …SNI], {TF1, TF2, …, TFNI}, BTF, BLF, PF) ;**
> **net = newff (P, T, S, TF, BTF, BLF, PF, IPF, OPF, DDF) ;**

Tel que :

**P** : la matrice R*Q1 de Q1 représentant les vecteurs d'entrée d'élément R.

**T** : la matrice SN*Q2 de vecteurs cibles d'élément SN représentatifs Q2.

**S** : les tailles de N-1 couches cachées, S1 a S (N-1), valeur par défaut = [].

**PR** : lamatrice R*2 des valeurs min et max pour les éléments d'entrée R.

**Si** : le nombre de neurones dans la iéme couche, i=1, …, NI.

**NI** :le nombre de couches.

**TFI :** fonction de transfert de la couche i ; la valeur par défaut est 'transig' pour les couches cachés et ' purelin' pour la couche de sortie.

**BTF** : fonction d'apprentissage de la rétro-propagation, défaut = 'traingdx'.

**BLF** : fonction d'apprentissage de la rétro-propagation, défaut = 'learngdm'.

**PF** : fonction de performance, défaut ='mse'.

La fonction **'newff'**utilise un générateur de nombres aléatoires pour créer les valeurs initiales des pondérations du réseau.

Si les neurones doivent avoir différentes fonctions de transfert, ils doivent être disposés en différentes couches.

### 4.2.2. L'entrainement

La phase d'entrainement nécessite de déclarer les paramètres suivants :

- Le nombre de cycle pendant l'apprentissage (itération, epochs).

- La fréquence de vérification de l'erreur d'apprentissage.
- Le taux d'apprentissage (η : mu).

Il faut aussi déclarer la division des données en trois parties selon les phases : d'apprentissage, de test et validation.

La fonction '**train**' est utilisé pour former le réseau chaque fois qu'un train est appelé.

> **net1= train (net, P, T) ;**

Tel que :

**net1 :** nouvel objet réseau.

**net :** le réseau MLP initial généré par newff.

**P** : vecteur d'entrée mesuré du réseau.

**T** : vecteur de sortie mesuré.

Les paramètres d'entrainements du réseau sont définis comme suit :

> **trainParam :** propriété définit les paramètres et les valeurs de la fonction de formation actuelle.
> **net.trainParam :** les champs de cette propriété dépendant de la propriété fonction de formation.

Les composants de '**trainParam' :**

> **net.trainParam.epochs :** pour indiquer le nombre maximum d'époques à former pour l'algorithme.
> **net.trainParam.show** : pour indiquer à l'algorithme le nombre d'époques doit y avoir entre chaque présentation de la performance (les époques entre les progrès).
> **net.trainParam.goal** : objectif de performance.
> **net.trainParam.timeinf :** durée maximal d'entrainement en secondes.
> **net.trainParam.min_grad :** gradient de performance minimal.
> **net.trainParam.max_fail :** échecs de validation maximum.

La fonction de performance 'performFcn' est utilisée pour mesurer les performances de réseau ; elle détermine à quel point l'ANN accomplit sa tâche.

Les fonctions de performance :

**Mse** : fonction d'erreur quadratique moyenne ; cette fonction est utilisée généralement pour la régression linéaire.

La formation de réseau s'arrête si :

- Le nombre maximum d'époques est atteint.
- La performance a été au minimum.
- Le temps maximum a été dépassé.
- Les performances de validation ont augmenté de plus de max_fail fois depuis sa dernière diminution.

**Train** appelle la fonction indiquée par '**net.trainFcn'**, en utilisant les valeurs du paramètre d'apprentissage indiquées par '**net.trainParam'**. [12]

### 4.2.3. La simulation

La phase de simulation est faite par l'utilisation de la décomposition des données entre teste et validation ; tel que les performances du réseau doivent être vérifiées lorsque la formation est terminée.

Pour appeler la simulation de tests qui seront exposées au réseau en utilise la fonction 'sim' comme suit :

> **Y = sim (net1, P) ;**

Tel que :

**Y** : la sortie mesurée.
**P** : vecteur d'entrée.
**net1** : objet MLP final.

La commande sim est appliquée pour tester dans quelle mesure le net résultant MLP est proche des données [12]

### 4.3 Exécution

Pour illustrer la conception et la simulation des réseaux de neurones, nous avons proposé deux exemples simples utilisant le programme MATLAB, ces exemples montrent l'approximation des fonctions logique « OU » et « OU exclusif » par un réseau de neurones.

#### 4.3.1 La porte « OU »

La table de vérité de la fonction « OU » :

| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Tab2.1 :** table de vérité « OU ».

La porte « OU » peut être modélisé par un réseau de neurones simple : un perceptron monocouche.

Les couches de ce réseau sont : une couche d'entrée avec deux(2) neurones **x1**et **x2** et une couche de sortie avec un seul neurone **y**. La fonction de transfert utilisée est une fonction linéaire.

#### 4.3.2 La description de programme sous MATLAB pour la porte « OU »

>>**net= newp ([0 1 ;-2 2] ,1)** ; % création un perceptron d'un vecteur avec deux éléments dans un intervalle choisie entre [0 1] et [-2 2] et un neurone de sortie ; l'apprentissage utilise une fonction LEARNP (la règle de correction d'erreur).

>>**p= [0 0 1 1 ;0 1 0 1] ;**

% création d'un vecteur avec deux éléments de (4) quatre valeurs qui illustre la fonction logique « OU ».

**>> t= [0 1 1 1] ;**%la sortie correspondante d'un vecteur avec élément de (4) quatre valeur.

**>>y= sim (net, p)**%simulation la sortie de réseau en faisant une initialisation du réseau de neurone.

**y=  1  1  1  1**% la sortie de réseau.

Le programme n'est pas entrainé donc en constate que la sortie est différée de celle désirée ; pour cela en continue le programme avec la mise à jour des poids.

**>>net.trainparam.epoch= 20 ;**%nombre d'itération maximal.

**>>net=train (net,p, t) ;**%l'entrainement de réseau.

Le réseau est bien entrainé, la figure suivante visualise le réseau utilisé :

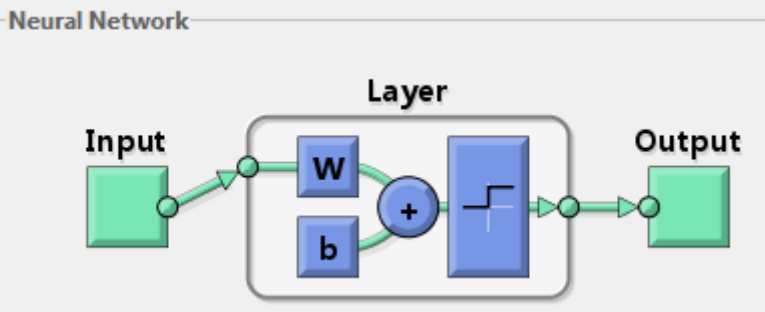



**Figure 2.3 :** L'architecture de réseau de neurone pour la fonction « OU ».

La figure suivante décrit l'entrainement du réseau, la ligne bleu est le but à atteindre. Au bout de deuxième itération, le résultat désiré est atteint.

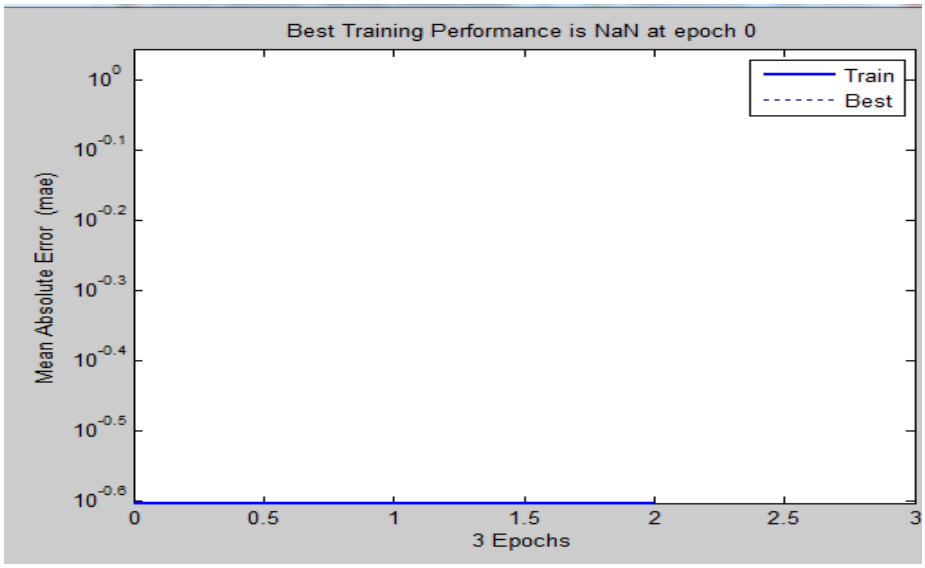



**Figure 2.4 :** l'entrainement de réseau de neurone.

**>>y= sim (net, p)**%on simule de nouveau la sortie du réseau.

**y =0 1 1 1**

### 4.3.3 La porte « OU Exclusif »

La table de vérité de fonction « OU Exclusif », que l'on doit modéliser par perceptron multicouche :

| x1 | x2 | y |
|---|---|---|
| **0** | **0** | **0** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

**Tab2.2 :** table de vérité « OU Exclusif ».

La porte « OU Exclusif » peut être modélisé par un perceptron multicouche.

Les couches de ce réseau sont : une couche d'entré avec deux (2) neurones **x1**et **x2** et une couche cachée avec (3) trois neurones cachés et une couche de sortie avec un seul neurone **y**.La fonction de transfert utilisée est une fonction sigmoïde dans la couche cachée et une fonction linière dans la couche de sortie.

### 4.3.4 La description de programme sous MATLAB pour « OU Exclusif »

**>>P= [-1 -1 1 1 ;-1 1 -1 1]** ; %initialisation de la fonction logique « OU Exclusif », on utilise des valeurs -1 et 1, car la fonction tansig exige de telles valeurs**.**

**>> t= [-1 1 1 -1] ;%** la cible de sortie.

**>>net = newff (minmax(p), [3 1], {'tansig' 'purelin'}) ;**%création le réseau de neurone. L'intervalle des données utilisées est indiqué par la fonction minmax(p). Ce réseau utilise la règle d'apprentissage par rétro- propagation.

>>**y= sim (net, p**) % la simulation de sortie de réseau en faisant une initialisation.

**y= 1.9770 0.3659 0.5591 -0.2242** % les résultats sans entrainement

**>>net.trainparam.epochs = 50 ;%** le nombre d'itérations maximale (50).

>>**net= train (net, p, t) ;**%l'entrainement de réseau.

- **L'architecture**

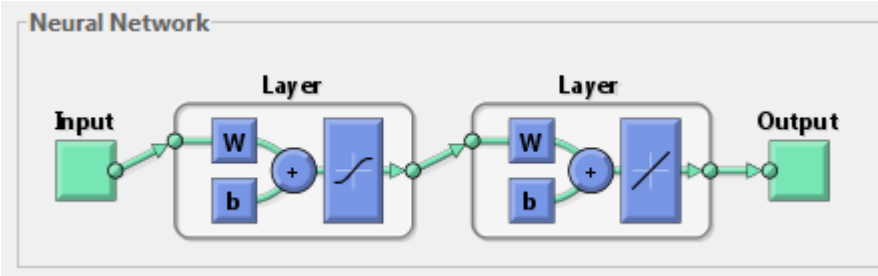



**Figure 2.5 :** L'architecture de réseau de neurone pour la fonction « OU Exclusif ».

- **Performance d'apprentissage**

La figure 2.6 montre l'entrainement du réseau .la ligne bleue montre l'entrainement du réseau. On atteint le résultat désiré au bout de six itérations.

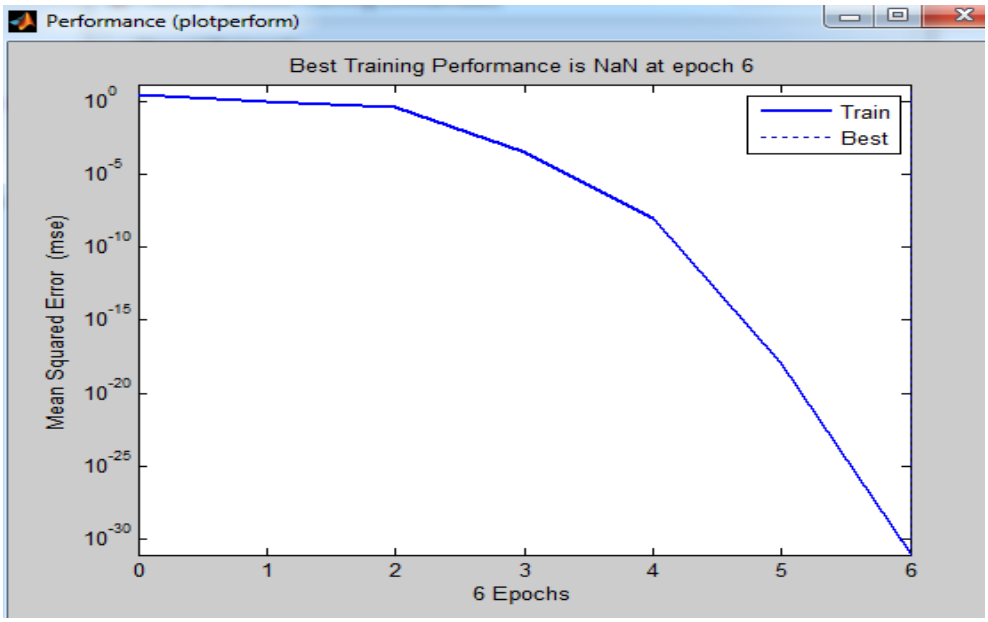



**Figure 2.6 :** l'entrainement du réseau.

- **Training**

Le pas du gradient est un facteur déterminant dans la vitesse de convergence du réseau neurone. Ou grad=7.7803$^{e}$-016 a 6 itérations.

Dans le cadre de notre exemple le taux d'apprentissage est fixé à 1$^{e}$-009 et un nombre maximum d'itération à 50 itérations. Durant l'apprentissage, le réseau est validé toutes les 6 itérations.

Val-Fail : échec de validation maximum=0.

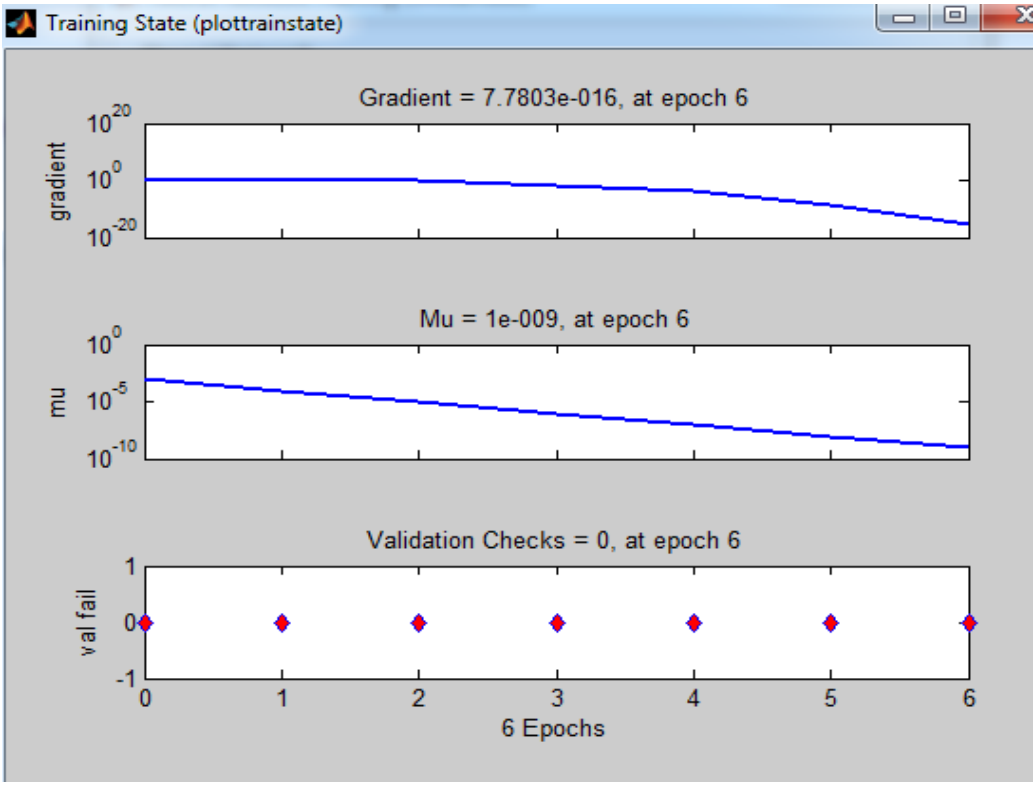



**Figure 2.7 :** Training modèle réseaux de neurone.

- **Régression**

D'après les résultats numériques, nous pouvons conclure que le modèle neuronal a un coefficient de corrélation égal à l'unité (R=1) ; ceci montre que les valeurs mesurées et celles estimées par le modèle neuronal développé dans cet exemple sont égaux.

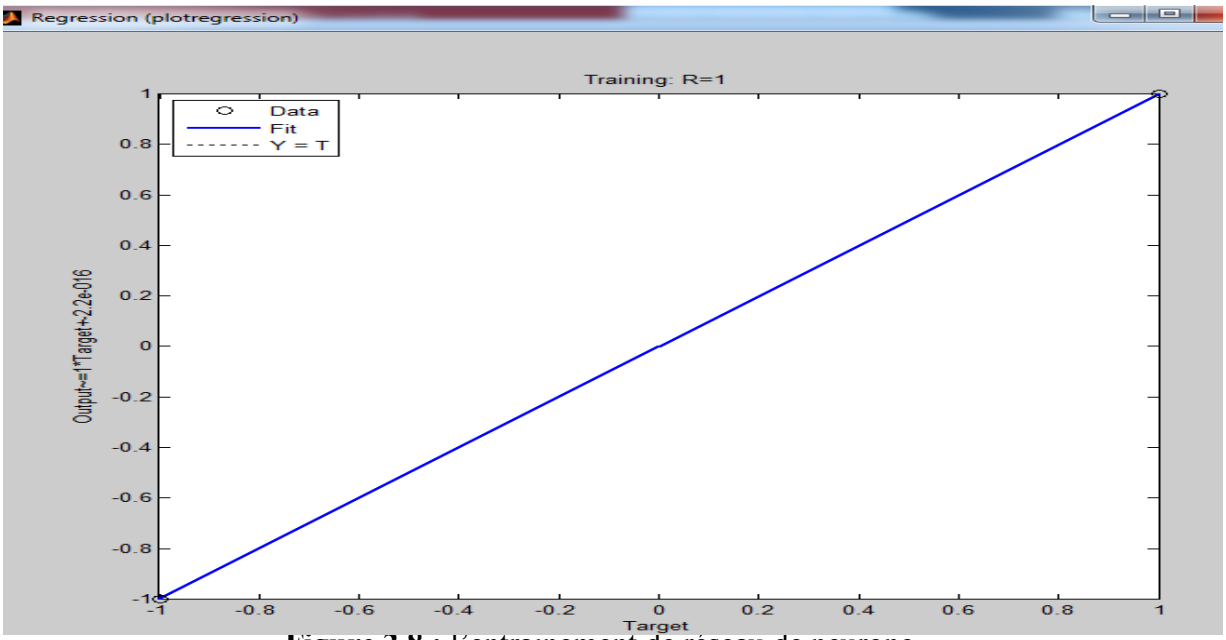



**Figure 2.8 :** l´entrainement de réseau de neurone.

**>>y= sim (net, p)** % on simule de nouveau la sortie de réseau

**y =**

**-1.0000 1.0000 1.0000 -1.0000**

On obtient alors la sortie désirée.

## 5. Conclusion

Dans ce chapitre nous avons défini l'implémentation et l'implémentation software ainsi que l'algorithme de rétro propagation utilisée pour la conception d'un réseau de neurones sous MATLAB.

Par la suite nous avons présenté une étude sur les étapes les plus importantes dans la création d'un réseau de neurones ; pour cela nous avons proposé deux exemples qui nous permettent de mettre en application les grandes lignes que nous avons détaillées dans notre étude ; nous avons pu aussi voir leur utilité au niveau de l'entrainement des réseaux et de différencier entre les réseaux monocouche et multicouche.

## 1. Introduction

L'objectif final de ce mémoire est de contribuer à l'implémentation d'un réseau de neurone artificiel sur un circuit FPGA.

Ce chapitre concerne a l'implémentation hardware d'un réseau de neurone, nous nous intéressons par principe à implémenter un réseau de neurone de type le module Feed-Forward.

Dans ce sens nous présentons le neurone constituant ce dernier, son architecture interne, par la suit nous validerons par une simulation et synthèse les résultats obtenus pour un neurone sur un circuit FPGA (**xc7z020clg484-1**), et pour un réseau de neurone sur les déférentes circuits FPGA (***zynq7000 xa7z010clg255-1I ;*****Arix7 *:xc7a1001csg324-21 ;*** **Arix7 :xca200tfbg-484-3**) et nous conclurons les résultats d'implémentation pour chaque famille.

## 2. l'outil de conception Xilinx VIVADO

XilinxVivado est l'environnement de développement logiciel de Xilinx. Il offre un ensemble complet d'outils familiers et puissants, de bibliothèque et de méthodologies.

Ce logiciel permet essentiellement d'effectuer les différentes étapes de la conception et e la réalisation de circuit numérique sur FPGA. Il permet entre autres de faire la description, la simulation, la synthèse et l'implémentation d'un circuit, puis la programmation d'un circuit FPGA Vivado doit être installé sur le PC d'utilisateur [13].


**Figure 3.1** : XILINX VIVADO.

### 3. L'architecture de module Feed-Forward

On s'intéresse à l'implémentation du module Feed-Forward d'un réseau de neurones. L'architecture interne de ce dernier est représentée dans la figure 3.2 ci-dessous.

Le Feed-forward (propagation) est composé de 3 couches d'architecture similaire comme le montre la figure 3.2.

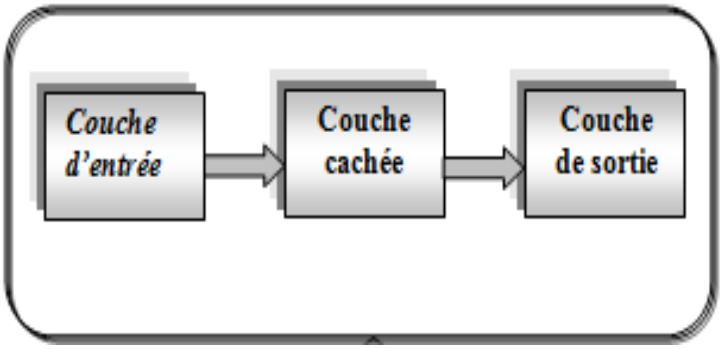



**Figure 3.2 :** Architecture Feed-Forwards.

Les équations qui gèrent la partie Feed-Forward sont :

- $\mu_j^{[1]} = \sum_{i=1}^{n_0} W_{ji}^{[1]} x_i$ **(3.1) :** Cette équation représente la somme pondérée pour chaque neurone d'entrées.
- $y_j = f(x_j)$ **(3.2) :** représente la fonction d'activation du neurone, qui sont dans notre cas une fonction sigmoïde et rampe à saturation.

On s'intéresse à l'implémentation d'un réseau de neurones constitué de cinq (5) neurones : deux neurones dans la couche d'entrée, deux neurones dans la couche cachée et un neurone dans la couche de sortie, comme le montre la figure 3.3.

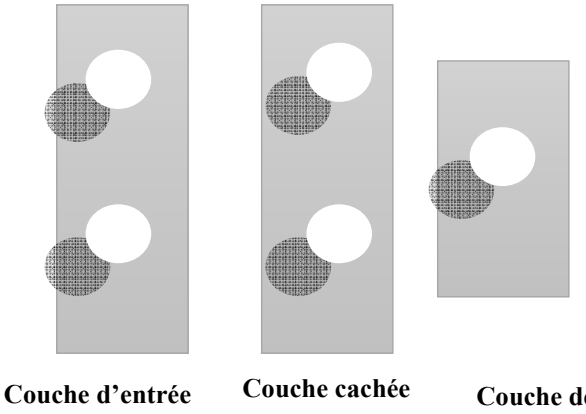



**Figure 3.3 :** Représentation d'un réseau de neurone (2 2 1).

### 3.1. Présentation de l'architecture d'une couche

Chacune de ces couches (couche d'entré et couche cachée) est constituée de deux neurones précédé par un bloc de multiplexage. La couche de sortie quant à elle est constituée d'un seul neurone. La figure ci-dessous montre l'architecture interne d'une couche.

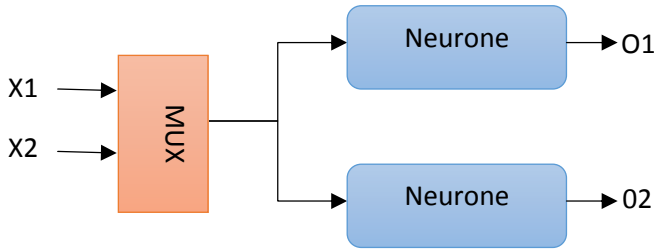



**Figure 3.4 :** Architecture de la couche.

### 3.2 L'architecture du neurone

Le neurone est l'élément de base de module Feed-Forward, il constitué d'un bloc MAC, un bloc des poids synaptique et un bloc d'activation.

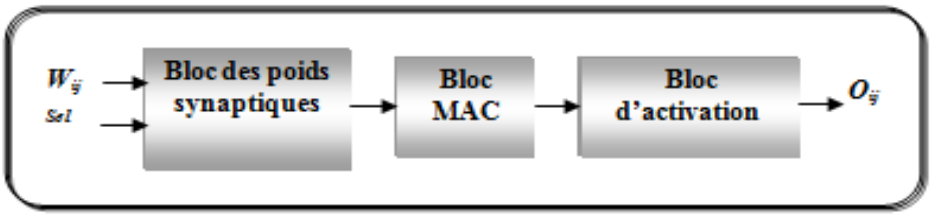



**Figure 3.5 :** Neurone Feed-Forward.

### 3.3.Bloc des poids synaptiques

Pour stocker les poids synaptiques il suffit d'utiliser une RAM, donc il faut initialiser ces poids à des valeurs aléatoires et les mettre dans la RAM puis faire la mise à jour de ces poids et les stocker ; le multiplexeur est nécessaire à l'entrée de la RAM pour le mixage entre les valeurs d'initialisation et celle de la mise à jour.

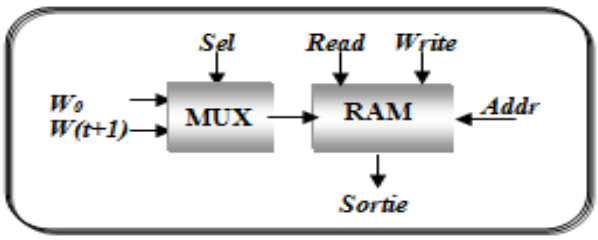



**Figure 3.6 :** Architecture du bloc des poids synaptiques.

### 3.4Bloc d'activation

Le bloc de la fonction d'activation est pour prendre la valeur de la somme pondérée calculé par le neurone pour générer la valeur d'activation du neurone et de lui appliquer la fonction désignée. Pour notre réseau nous avons utilisé la fonction sigmoïde définie comme suit :

$$y_j = f(x_j) = \frac{1}{1+\exp(-\alpha x_j)} \quad \textbf{(3.3)}$$

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 nous utilisons les Look-Up-Tables (LUTs) de l'FPGA pour la modélisation de cette fonction.

Dans notre cas nous utilisons ces LUTs en ROM adressée par la valeur de la somme pondérée et les valeurs d'activation seront chargées dans cette ROM, donc nous aurons la valeur de la somme pondérée en entrée, et celle de l'activation en sortie. La figure 3.7 montre le tracé de la fonction sigmoïde.

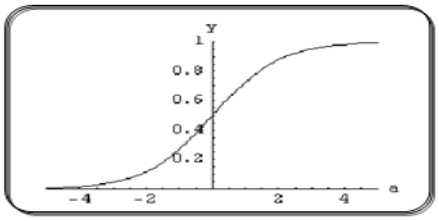



**Figure 3.7 :** la fonction sigmoïde.

### 4. simulation et synthèse d'un neurone

L'architecture de neurone présenté dans la Figure 3.9 peut-être décrire en VHDL comme suit :

C:/Users/FARES/Desktop/4/project_9.srcs/sources_1/new/NEURONE.vhd

```
12  LIBRARY ieee;
13  USE ieee.std_logic_1164.all;
14  USE ieee.std_logic_arith.all;
15
16
17  ENTITY NEURONE IS
18      GENERIC(
19          m : integer := 16;
20          n : integer := 16
21      );
22      PORT(
23          X      : IN      std_logic_vector (m-1 DOWNTO 0);
24          addr   : IN      std_logic_vector (7 DOWNTO 0);
25          clk    : IN      std_logic;
26          load   : IN      std_logic;
27          read   : IN      std_logic;
28          reset  : IN      std_logic;
29          sel    : IN      std_logic;
30          w      : IN      std_logic_vector (n-1 DOWNTO 0);
31          write  : IN      std_logic;
32          wt1    : IN      std_logic_vector (n-1 DOWNTO 0);
33          -- O11    : OUT     std_logic_vector (15 DOWNTO 0);
34          X11    : OUT     std_logic_vector (m-1 DOWNTO 0);
35          wc     : OUT     std_logic_vector (15 DOWNTO 0)
36      );
37
38  -- Declarations
39
```

```
40  END NEURONE ;
41
42
43
44  ARCHITECTURE struct OF NEURONE IS
45
46      -- Architecture declarations
47
48      -- Internal signal declarations
49      SIGNAL wn : std_logic_vector(n-1 DOWNTO 0);
50   -- SIGNAL wc : std_logic_vector(n-1 DOWNTO 0);
51      -- Implicit buffer signal declarations
52      SIGNAL X11_internal : std_logic_vector (m-1 DOWNTO 0);
53      SIGNAL wc_internal  : std_logic_vector (15 DOWNTO 0);
54
55
56      -- Component Declarations
57      COMPONENT MAC
58      GENERIC (
59          n : integer := 16;
60          m : integer := 16
61      );
62      PORT (
63          WC     : IN      std_logic_vector (m-1 DOWNTO 0);
64          X      : IN      std_logic_vector (m-1 DOWNTO 0);
65          clk    : IN      std_logic ;
66          load   : IN      std_logic ;
67          reset  : IN      std_logic ;
```

```
68          X11    : OUT     std_logic_vector (m-1 DOWNTO 0)
69      );
70      END COMPONENT;
71      COMPONENT MUX
72      GENERIC (
73          n : integer := 16
74      );
75      PORT (
76          w    : IN      std_logic_vector (n-1 DOWNTO 0);
77          wt1 : IN      std_logic_vector (n-1 DOWNTO 0);
78          sel : IN      std_logic ;
79          wn   : OUT     std_logic_vector (n-1 DOWNTO 0)
80      );
81      END COMPONENT;
82      COMPONENT RAM
83      PORT (
84          write : IN      std_logic ;
85          addr   : IN      std_logic_vector (7 DOWNTO 0);
86          wn     : IN      std_logic_vector (15 DOWNTO 0);
87          read   : IN      std_logic ;
88          wc     : OUT     std_logic_vector (15 DOWNTO 0)
89      );
90      END COMPONENT;
91
92      -- Optional embedded configurations
93      -- pragma synthesis_off
94     -- FOR ALL : LUT_SIG USE ENTITY PROJET_2004.LUT_SIG;
95    --   FOR ALL : MAC USE ENTITY PROJET_2004.MAC;
```

```
96     -- FOR ALL : MUX USE ENTITY PROJET_2004.MUX;
97      -- FOR ALL : RAM USE ENTITY PROJET_2004.RAM;
98      -- pragma synthesis_on
99
100
101 BEGIN
102     -- Instance port mappings.
103     I0 : MAC
104         GENERIC MAP (
105             n => 16,
106             m => 16
107         )
108         PORT MAP (
109             -- WC     => WC,
110             WC    => wc_internal,
111             X      => X,
112             clk    => clk,
113             load   => load,
114             reset  => reset,
115             X11    => X11_internal
116         );
117     I2 : MUX
118         GENERIC MAP (
119             n => 16
120         )
121         PORT MAP (
122             w    => w,
123             wt1 => wt1,
```

```
124                 sel => sel,
125                 wn  => wn
126             );
127         I1 : RAM
128             PORT MAP (
129                 write => write,
130                 addr  => addr,
131                 wn    => wn,
132                 read  => read,
133                wc     => wc_internal
134                 -- wc     => wc
135             );
136
137         -- Implicit buffered output assignments
138         X11 <= X11_internal;
139         wc <= wc_internal;
140
141     END struct;
142
```

**Figure 3.8 :** le programme VHDL d'un neurone.

### 4.1. Simulation d'un neurone

La simulation est faite sur l'outil de conception XilinxVivado comme suit :

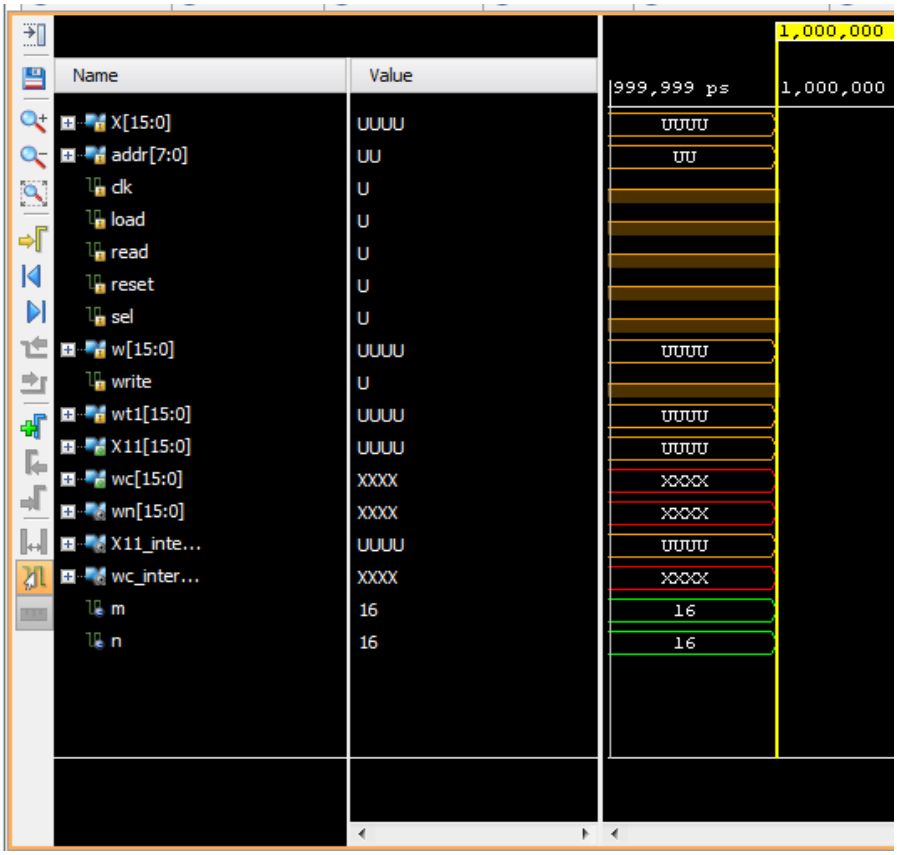
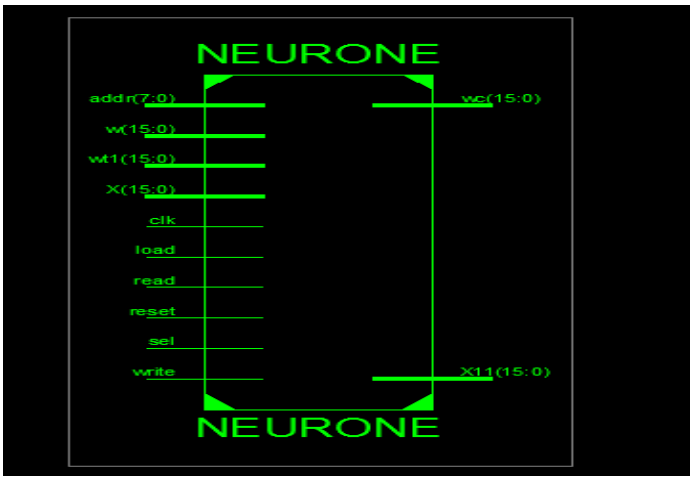



**Figure3.9** : Simulation d'un neurone.

## 4.2 Génération du schéma RTL

### 4.2.1 La synthèse d'un neurone

La Figure3.10 montrée schémas génèrent par l'outil de synthèse qui a transformé la description VHDL du neurone en schéma au niveau RTL (Register Transfer Leveldescrption). Ces résultats montrent bien la conformité de notre description VHDL avec le modèle du neurone proposée.



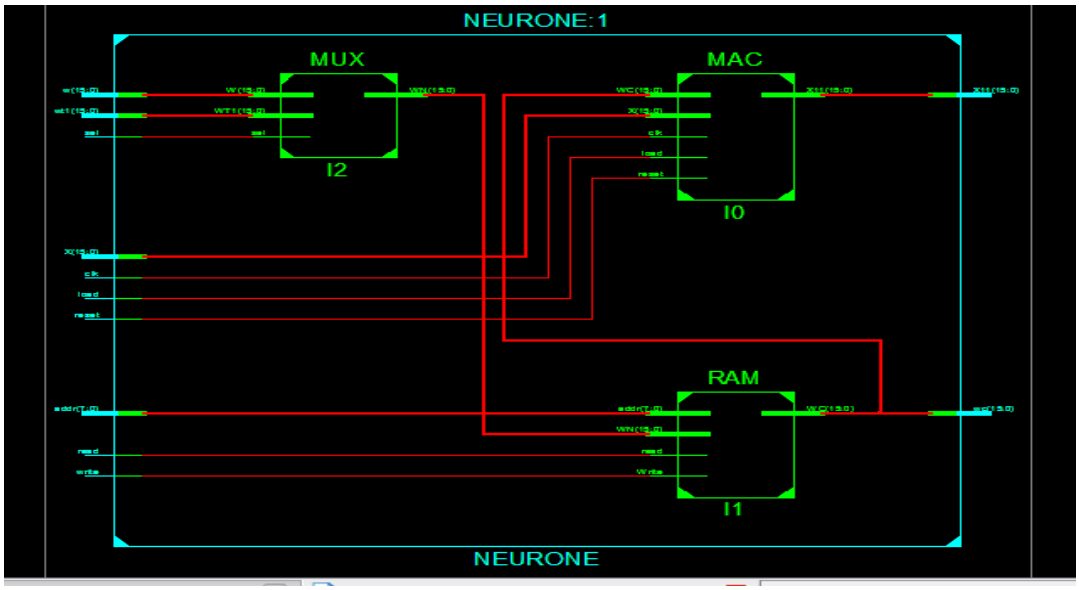




**Figure3.10 :** Les schémas de neurone au niveau RTL.

### 4.2.2 Les résultats de synthèse

Le rapport de la consommation des ressources disponibles sur le FPGA (xc7z020clg484-1) est donné dans ce tableau :

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 4128 | 106400 | 3% |
| Number of Slice LUTs | 1387 | 53200 | 2% |
| Number of fully used LUT-FF pairs | 1040 | 4475 | 23% |
| Number of bonded IOBs | 94 | 200 | 47% |
| Number of BUFG/BUFGCTRLs | 3 | 32 | 9% |
| Number of DSP48E1s | 1 | 220 | 0% |

Tab3.1 : Table de résultats de l'analyse RTL.

D'après le tableau ci-dessus, le circuit neurone occupe 3% de registres, 2% de bloc LUT, 23% de paires LUT-Flip FLOP (FF), 47 % d'entrées sorties 5IOBs) et 9% de Buffers. On peut dire que les ressources relatives aux IOB sont très importantes (47%).

### 4.2.3 Implémentation de neurone

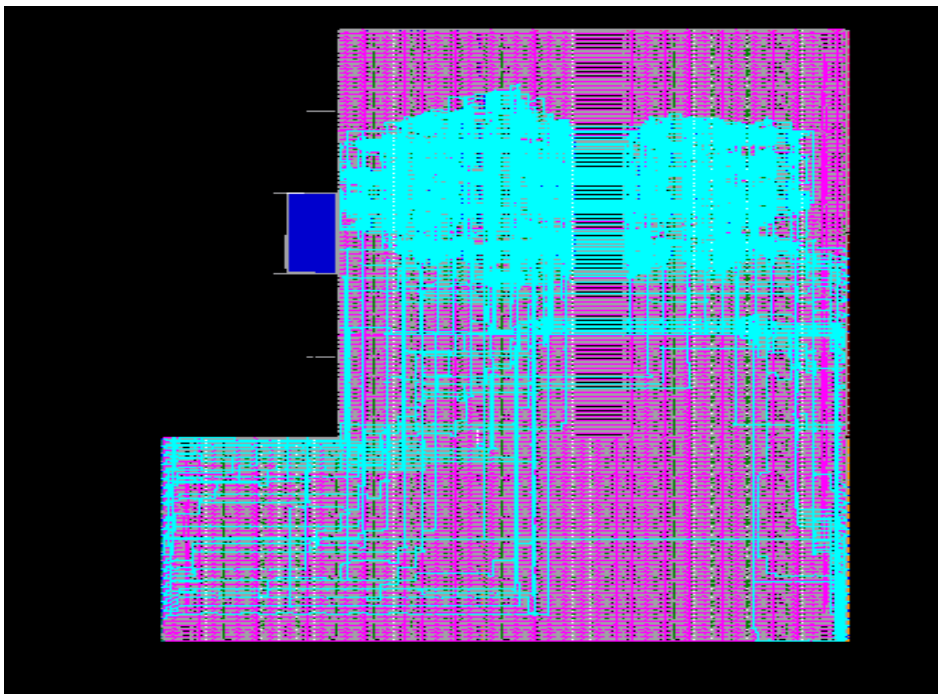


Figure3.11 : Implémentation d'un neurone.

D'après laFigure3.11 et pour le circuit FPGA (xc7z020clg484-1) sélectionné, le neurone occupe plus de 50% de la surface FPGA. Donc si on veut implémenter un réseau de plusieurs neurones il faut cibler un autre circuit FPGA et une autre carte beaucoup plus performante en terme de surface (nombre de portes, CLB, I/O, mémoires, etc.

## 5. Synthèse et implémentation d'un réseau de neurones (2 2 1)

Afin de valider le fonctionnement de l'architecture proposée, nous avons considéré un réseau de taille (2, 2, 1), donc nous avons au total 3 nœuds (neurones), une couche d'entrée sert au transfert des entrées, une couche cachée comportant 2 nœuds et une couche de sortie d'un nœud. Le réseau ainsi formé est un classificateur neuronal réalisant la fonction d'un XOR.
La synthèse et l'implémentation du réseau de neurones ont été effectuées en ciblant sur les circuits FPGAs suivant :

1. Carte ZYBO intégrant le circuit **FPGA Zynq 7000 : xa7z010clg225-1I**,
2. Carte Nexys4DDR intégrant le circuit FPGA : **Artix 7 : xc7a100lcsg324-2l**
3. **Artix-7 : XCa200tfbg-484-3**

### 5.1 Les résultat de synthèse

Les figures ci-dessous montrent les résultats de synthèse pour chaque famille de circuit FPGA :

#### 5.1.1 Résultat de synthèse sur le circuit Zynq 7000 :xa7z010clg225-1I

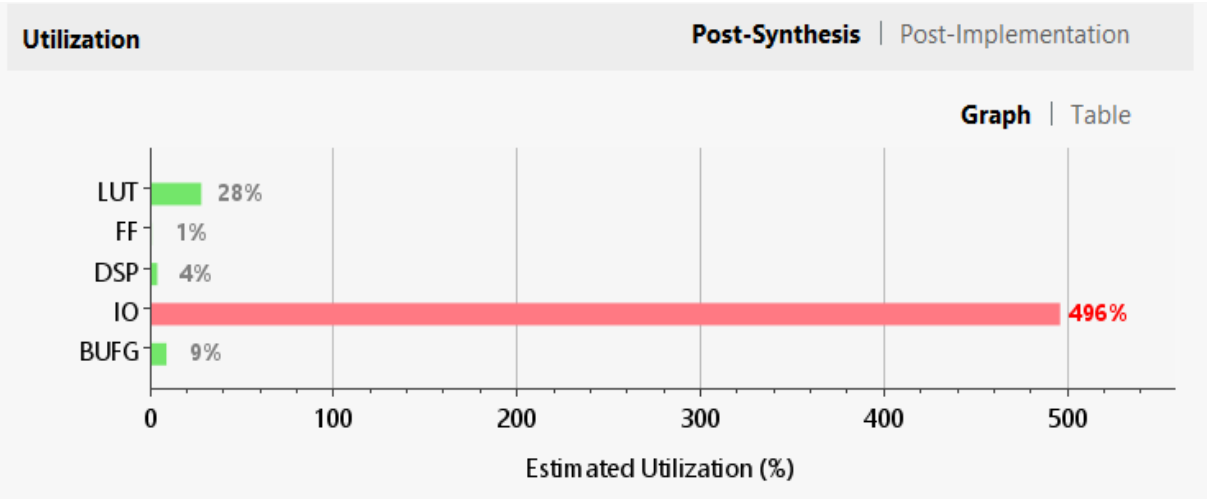



**Figure 3.12** : Résultat de synthèse sur le circuit Zynq7000 : xa7z010clg225-1I.

D'après la figure3.12on voit bien qu'il est impossible d'implémenter un réseau de 2 2 1 (5 neurones) sur la carte Zybo car il ya un dépassement des ressources entrées/sorties (IO) : 496% qui est 4 fois plus que le nombre IO du circuit FPGA en question.

### 5.1.2 Résultats de synthèse sur le circuit Artix 7 : xc7a100lcsg324-2l

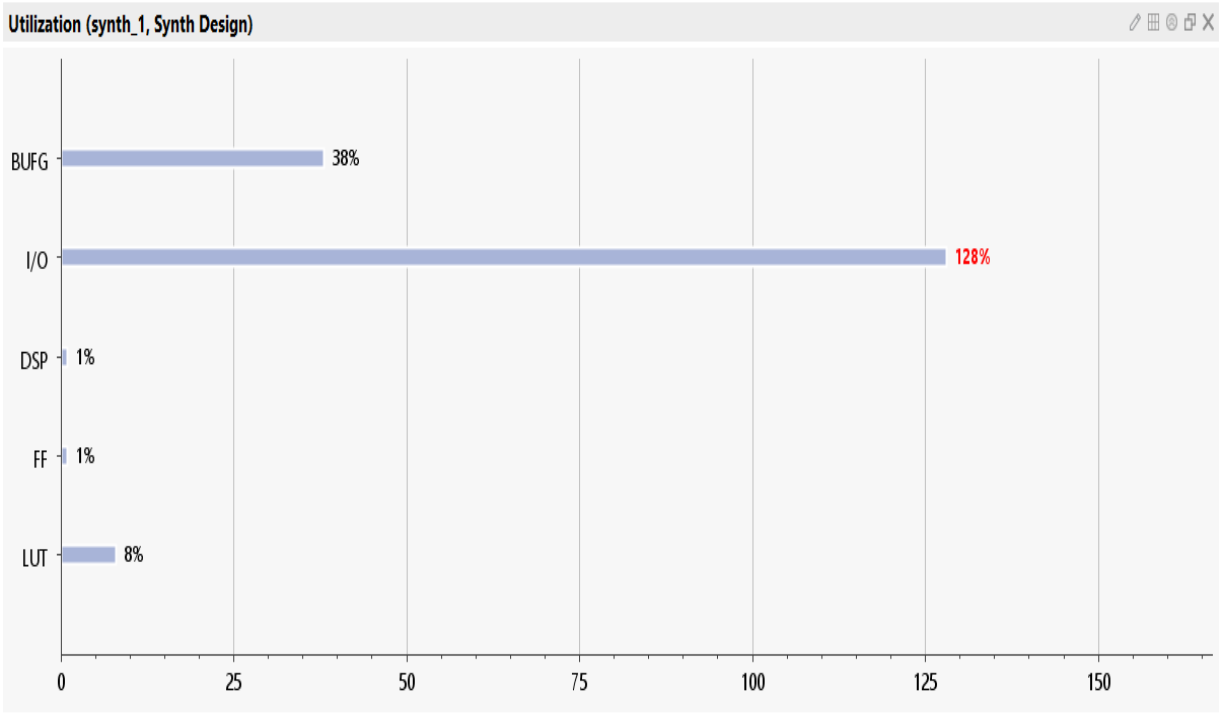



**Figure 3.13 :** Résultats de synthèse sur le circuit Artix 7 : xc7a100lcsg324- 2l.

Il en est de même pour la figure 3.13 ci-dessus qui montre l'impossibilité d'implémenter le réseau (2 2 1) sur la carte Nexys4DDR (Consommation de 128 des IO du circuit Artix 7 XCa100t324).

### 5.1.3 Résultats de synthèse sur le circuit Artix 7 : XCa200tfbg-484-3

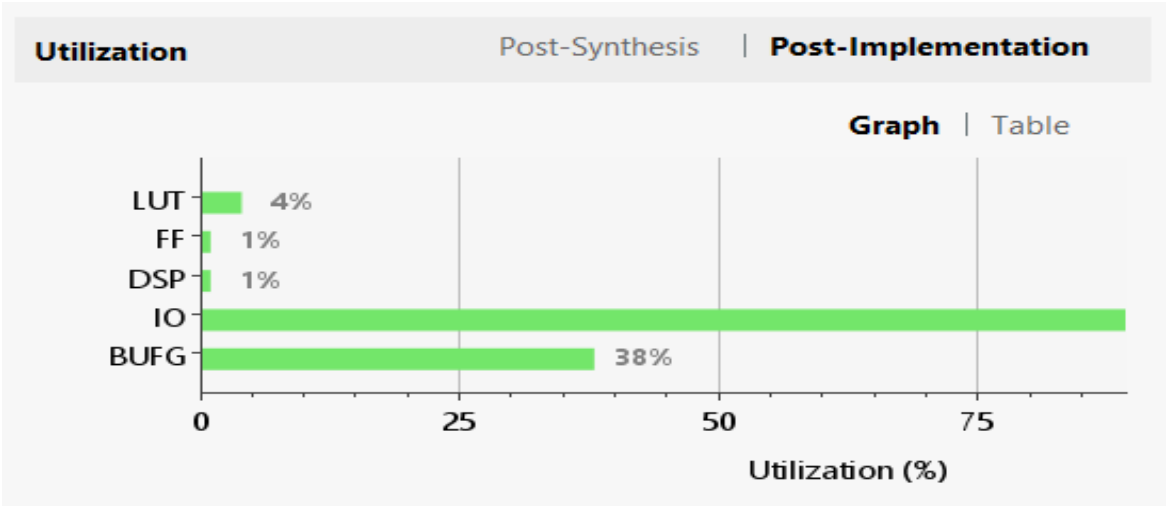



**Figure 3.14 :** Résultats de synthèse sur le circuit Artix 7 : XCa200tfbg-484-3.

Par contre en ciblant le circuit FPGA : XCa200tfbg-484-3,on voit que la synthèse se fait correctement avec un pourcentage de presque 80 d'IO ; 1% de DSP, 4% de mémoire LUT, 38% de buffer (BUFG) et 1% de Flip Flop (FF). Donc c'est ce circuit qui conviendra pour l'implémentation du réseau de neurone 2 2 1 sur FPGA.

### 5.2. Génération du schéma RTL

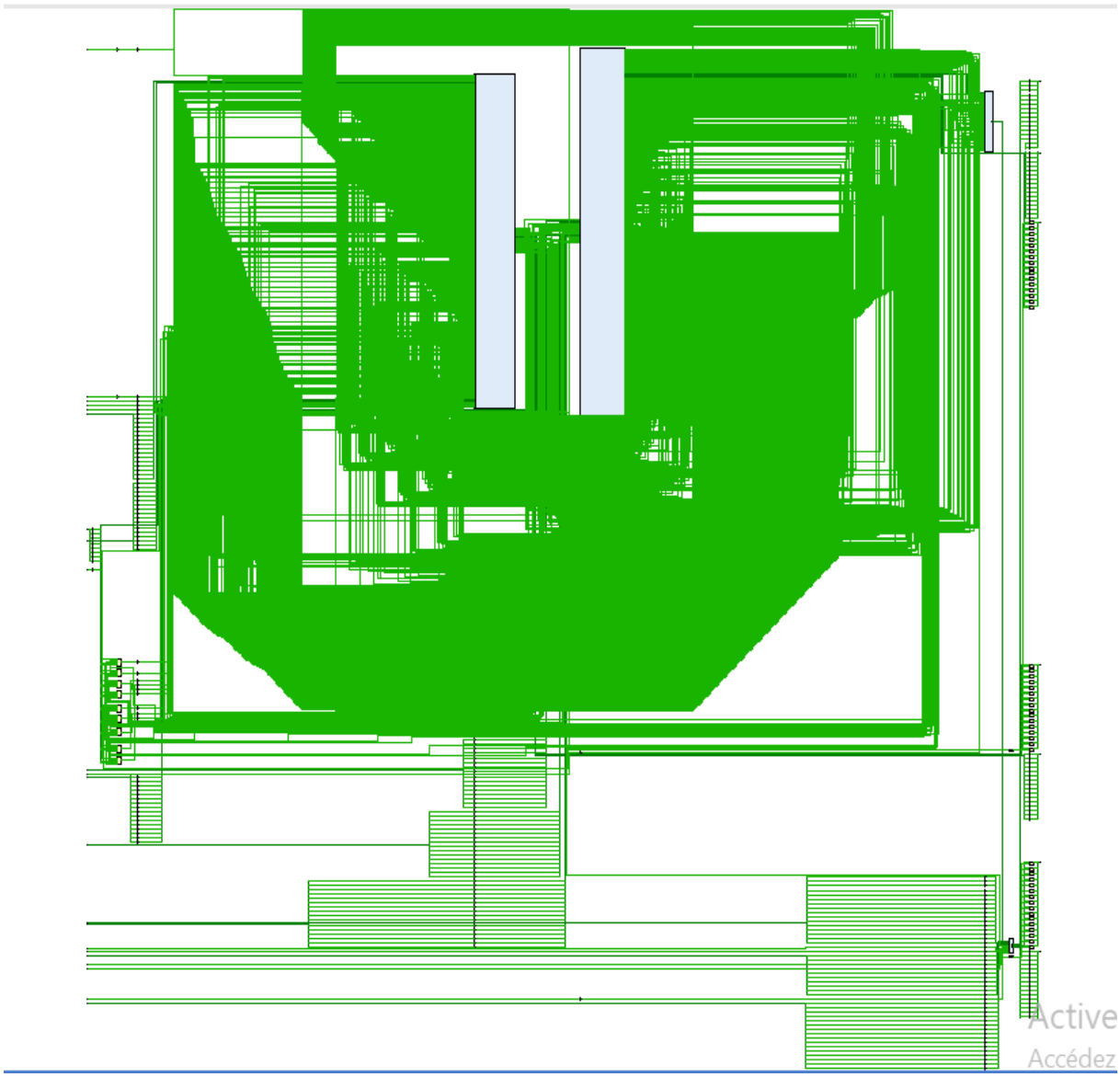



**Figure 3.15 :** Génération du schéma au niveau RTL (Register Transfer Level).

### 5.3 Résultats d'implémentation (Layout sur FPGA)

Le réseau de neurone qui réalise la fonction XOR est bien implémenté comme la figure3.16 montre :

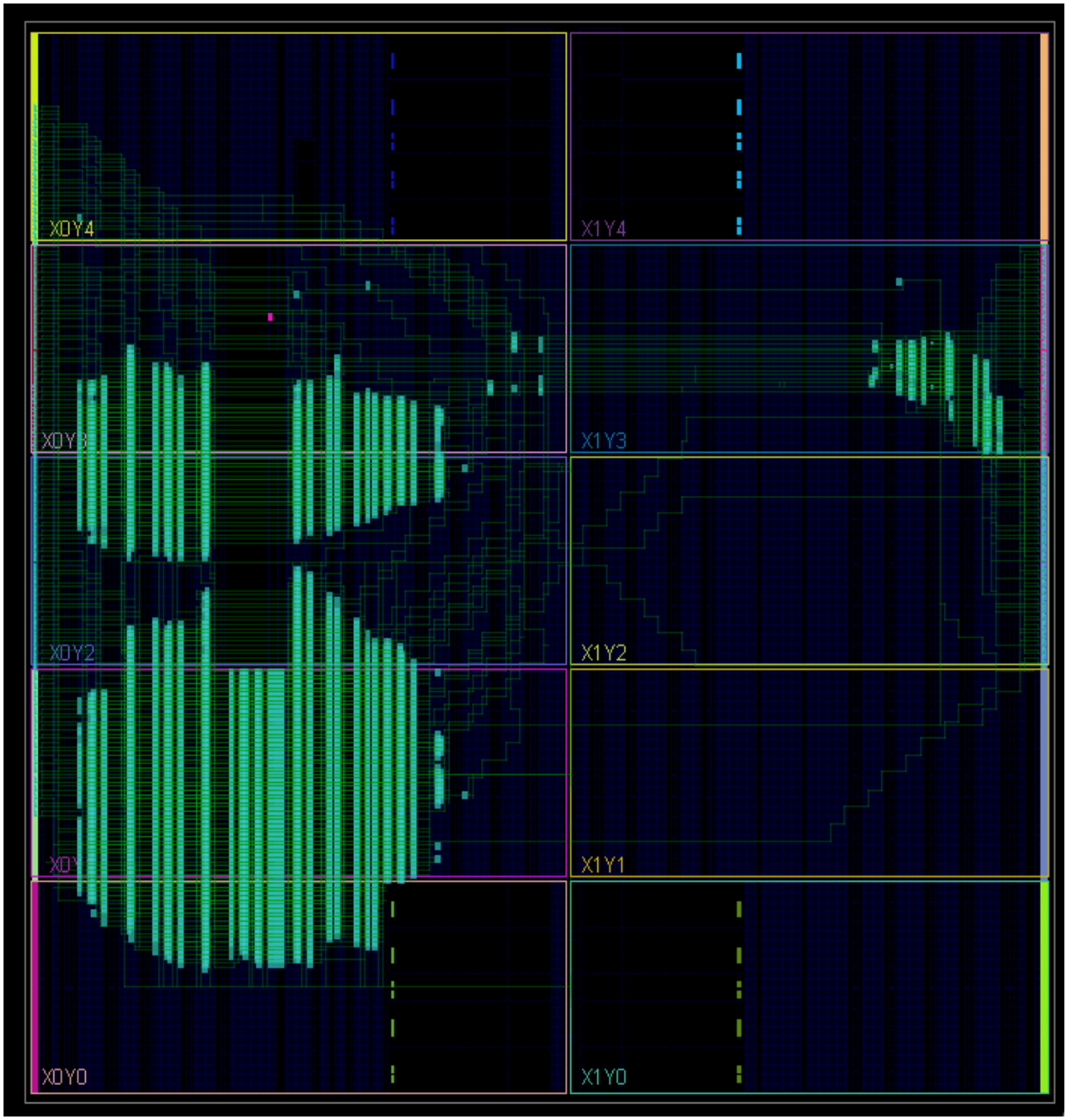



**Figure3.16 :** Résultats d'implémentation sur le circuit Artix 7 : **XCa200tfbg-484-3.**

### 5.4 Analyse de la puissance dissipée dans le circuit FPGA

En plus des résultats de synthèse l'outil VIVADO permet de calculer la puissance dissipé ou consommée dans le FPGA (voir figure 3.17 ci-dessous). Cette figure montre que la puissance statique du circuit FPGA est de 1,674 W et la puissance dynamique est de 140 W ce qui correspond à 99% de la puissance dissipée. 47% sont consommés par les signaux internes, 32% par les signaux d'entre sortie (IO), 19 % par la circuiterie logique interne de l'FPGA et une faible consommation par les DSP (correspondant à 1 %). Donc si on veut diminuer la puissance il faut optimiser les signaux internes (routage interne) ou bien il faut cibler une autre famille de circuit FPGA.

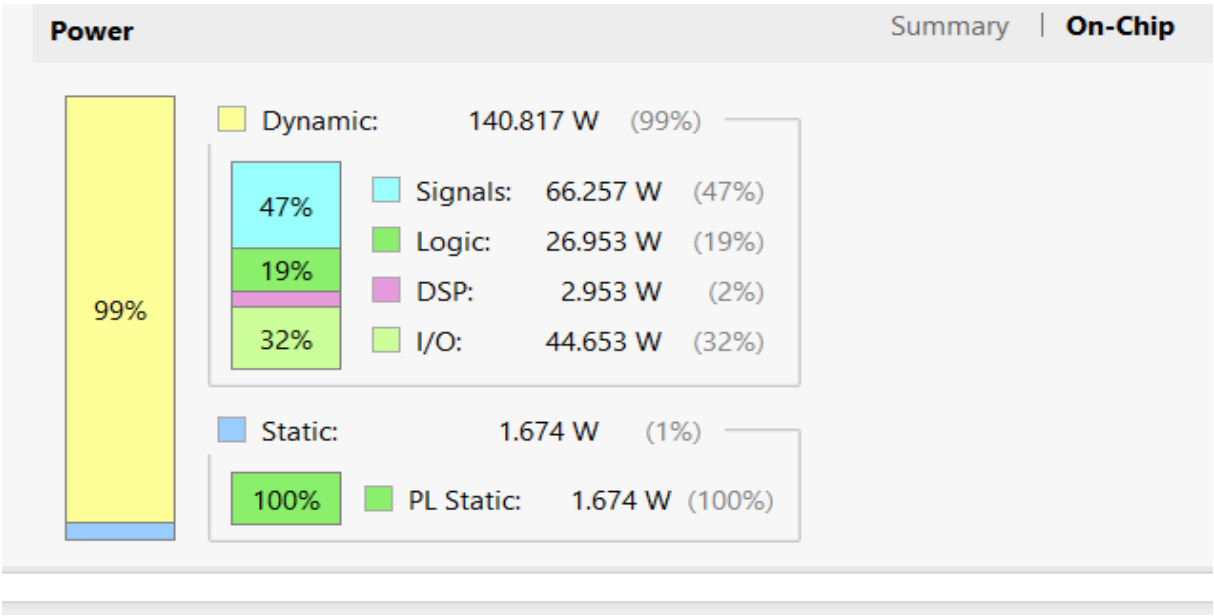



**Figure3.17 :** Consommation de puissance dans le FPGA.

### 6. Conclusion

La conception d'un réseau de neurone est très complexe et nécessite beaucoup de temps pour la mise en œuvre ; dans ce chapitre nous avons présenté les résultats de synthèse et implémentation de module Feed-Forward.

Nous avons commencé par une description générale de l'outil de conception VIVADO et les composants de module Feed-Forward ; par la suit nous avons implémenté un neurone sur la carte Zybo intégrant le circuit FPGA (xc7z020clg484-1), cette partie permet de conclure que cette série n'est pas appropriée pour l'implémentation d'un module Feed-Forwrad. Ensuite nous avons implémenté un classificateur neuronal réalisant la fonction(XOR) sur les déférentes cartes FPGA :Carte ZYBO intégrant le circuit FPGA Zynq 7000 : xa7z010clg225-1I, Carte Nexys4DDR intégrant l'Artix 7 : xc7a100lcsg324-2l puis sur Artix-7 : XCa200tfbg-484-3 et on conclut les

résultats pour chaque famille, nous conclurons que la carte Artix-7 : XCa200tfbg-484-3 est celle qui convient pour l'implémentation de réseau étudié.

# Conclusion Générale

Le travail effectué dans ce mémoire se rapporte à la conception et l'implémentation d'un réseau de neurone artificiel de type Feed-Forward sur une carte FPGA.

Les réseaux de neurone sont une nouvelle technique de traitement de l'information ; ils se traduisent par des algorithmes mettant en jeu des concepts associés à la nature de cerveau par la notion d'apprentissage.

L'implémentation de réseau de neurone sur les composants reconfigurable FPGA ne pose plus un problème avec ce type de composants. En fait, de nombreuses et récentes études sont faites sur ce domaine notamment au CDTA où une équipe de recherche travail sur ça.

Dans la première partie de ce projet, nous avons fait une étude théorique et technique sur les réseaux de neurone et les FPGA ainsi que le langage de description VHDL pour comprendre le travail effectué.

Dans la seconde partie nous sommes intéressés sur la conception et l'implémentation de réseau de neurone sous MATLAB. Nous avons présenté les étapes nécessaires pour la création de ce dernier, nous avons trouvé certaines difficultés concernant cette étude à cause de manque la formation et les formateurs dans le domaine de l'intelligence artificielle pour cela on a prend des simples exemples pour faire différencier entre deux types de réseau de neurone monocouche et multicouche basé sur l'algorithme de rétropropagation du gradient et concluons le résultat d'implémentation simulés sous l'outil MATLAB.

Le troisième chapitre est consacré pour l'implémentation hardware sous l'environnement Xilinx VIVADO à partir d'un code VHDL d'un classificateur neuronal réalisant la fonction XOR sur déférente carte FPGA, on a conclu que la carte Artix7 est celle qui convient à l'implémentation de module Feed-Forward. Les résultats de synthèse et l'implémentation de ce module nous permet aussi de conclure la puissance dissipée par cette carte.

Le travail effectué nous a permis de découvrir le domaine de la recherche et plus particulièrement le domaine de l'intelligence artificielle et familiariser avec logiciel XilinxVivado qui permet la mise en œuvre des systèmes embarqués.

# Références bibliographiques

[1]Rachid ELFARISSI, Youssef GARGAR, « Introduction aux réseaux de neurones », Mémoire de PFE en Science de la Matière Physique, juin 2009.

[2] N. IZEBOUDJENE, « Conception et implémentation en FPGA d'un classificateur neuronal des arythmies cardiaques », Mémoire Magister en électronique option Télécommunication Polytechnique Alger ,1999.

[3] Redouan ZAHRA, Aboubakre BELARBI « Implémentation des réseaux de neurones de type FFANN avec fonction d'activation seuil sur circuit FPGA », mémoire de PFE d'ingénieur, Septembre 2007.

[4] G.dreyfus, J.M.martinez, M.Samuelides, M.B.Gordon, S.Thiria, L.Hérault, « réseaux de neurones, méthodologie et application », 2002.

[5] B.IDJERI, « Modélisation d'un micro-capteur anémométrique par les réseaux de neurones », Mémoire Magistère en électronique option Télédétection, Tizi Ouzou ,2006.

[6] Mourad ABIDI, « Réalisation et implantation d'un réseau de neurones sur une architecture matérielle distribuée à base de réseau sur puce », Mémoire de PFE d'ingénieur, Ecole National d'ingénieurs de Sousse, Juin 2014.

[7] Cherrad BENBOUCHAMA, « Contribution à l'implémentation des réseaux de neurones artificiels sur hardware reconfigurable FPGA », Mémoire de magister, Ecole National Polytechnique, El Harrach, Juin 2008.

[8] A.Bensmail, «Implémentation des réseaux de neurone sur FPGA par l'utilisation de la reconfiguration dynamique », Mémoire de master de systèmes embarqués, ENST, 2016.

[9] Idir MELLAL, « Implémentation d'un réseau de neurone d'un micro capteur sur un FPGA », Mémoire de PFE Master en électronique option Télécommunication et Réseau, Tizi Ouzou.

[10] Daniel LEÒN GONZÀLZ, «FPGA implementation of an AD-HOC RISC-V SoC for industrial IoT », Mémoire de master IoT en informatique, université Complutense de Madride, juillet 2020.

[11] Reconnaissance de caractères manuscrits par réseau de neurones, thèse d'Al Falou Wassim, Décembre 1998, université saint joseph.

[12]http://ch.mathworks.com/help/nnet/getting-started-with-neural-network-toolbox.html, Consulté le : 22/09/2021.

[13] « Compréhension – interpréter l'information », Ian's Web Page ; http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom.

# Annexe

C:/Users/FARES/Desktop/4/project_9.srcs/sources_1/new/ACC.vhd

```
12  LIBRARY ieee;
13  USE ieee.std_logic_1164.all;
14  USE ieee.std_logic_arith.all;
15
16
17  ENTITY ACC IS
18  -- Declarations
19      GENERIC(
20       n : integer := 16
21
22     );
23  PORT(
24       clk     : IN     std_logic;
25       din     : IN     std_logic_vector (n-1 DOWNTO 0);
26       load    : IN     std_logic;
27       reset   : IN     std_logic;
28       dout    : OUT    std_logic_vector (n-1 DOWNTO 0)
29     );
30  END ACC ;
31
32  -- hds interface_end
33  ARCHITECTURE Accumulation OF ACC IS
34
35  SIGNAL MW_I0cregl, MW_I0nregl : std_logic_vector(n-1 DOWNTO 0);
36  BEGIN
37     dout <= MW_I0cregl;
38     I0seq: PROCESS (clk, reset, MW_I0nregl,din, MW_I0cregl, load)
39      VARIABLE dtemp, ntemp, ctemp : std_logic_vector(n DOWNTO 0);
```

Programme VHDL d'un accumulateur.

C:/Users/FARES/Desktop/4/project_9.srcs/sources_1/new/MAC.vhd

```
12  LIBRARY ieee;
13  USE ieee.std_logic_1164.all;
14  USE ieee.std_logic_arith.all;
15
16
17  ENTITY MAC IS
18     GENERIC(
19        n : integer := 16;
20        m : integer := 16
21     );
22     PORT(
23        WC     : IN     std_logic_vector (m-1 DOWNTO 0);
24        X      : IN     std_logic_vector (m-1 DOWNTO 0);
25        clk    : IN     std_logic;
26        load   : IN     std_logic;
27        reset  : IN     std_logic;
28        X11    : OUT    std_logic_vector (m-1 DOWNTO 0)
29     );
30
31  -- Declarations
32
33  END MAC ;
34
35
36
37  ARCHITECTURE MAC OF MAC IS
38
39     -- Architecture declarations
```

Programme VHDL d'un MAC.

C:/Users/FARES/Desktop/4/project_9.srcs/sources_1/new/MULT.vhd

```
12  LIBRARY ieee;
13  USE ieee.std_logic_1164.all;
14  USE ieee.std_logic_arith.all;
15
16
17  ENTITY MULT IS
18     GENERIC(
19        m : integer := 16
20     );
21     PORT(
22        din0 : IN     std_logic_vector (m-1 DOWNTO 0);
23        din1 : IN     std_logic_vector (m-1 DOWNTO 0);
24        prod : OUT    std_logic_vector (m-1 DOWNTO 0)
25     );
26
27  -- Declarations
28
29  END MULT ;
30
31  -- hds interface_end
32  ARCHITECTURE MULT OF MULT IS
33  BEGIN
34
35  I0combo: PROCESS (din0, din1)
36     VARIABLE dtemp : std_logic_vector(2*m-1 DOWNTO 0);
37     BEGIN
38        dtemp := (unsigned(din0) * unsigned(din1));
39        prod <= dtemp(m-1 DOWNTO 0);
```

Programme VHDL d'un multiplieur.

C:/Users/FARES/Desktop/4/project_9.srcs/sources_1/new/MUX.vhd

```
12  LIBRARY ieee;
13  USE ieee.std_logic_1164.all;
14  USE ieee.std_logic_arith.all;
15
16
17  ENTITY MUX IS
18  -- Declarations
19  generic(n:integer:=16);
20  PORT(
21        W : IN     std_logic_vector (n-1 DOWNTO 0);
22        W1 : IN     std_logic_vector (n-1 DOWNTO 0);
23        sel  : IN     std_logic;
24        WN : OUT    std_logic_vector (n-1 DOWNTO 0)
25     );
26  END MUX ;
27
28  -- hds interface_end
29  ARCHITECTURE MUX OF MUX IS
30
31     SIGNAL MW_I0din01 : std_logic_vector(n-1 DOWNTO 0);
32     SIGNAL MW_I0din11 : std_logic_vector(n-1 DOWNTO 0);
33
34  BEGIN
35
36  PROCESS (MW_I0din01, MW_I0din11,  sel)
37     VARIABLE dtemp : std_logic_vector(n-1 DOWNTO 0);
38     BEGIN
39
```

Programme VHDL d'un multiplexeur.

C:/Users/FARES/Desktop/4/project_9.srcs/sources_1/new/RAM.vhd

```
12  LIBRARY ieee;
13  USE ieee.std_logic_1164.all;
14  USE ieee.std_logic_arith.all;
15
16
17  ENTITY RAM IS
18  -- Declarations
19   PORT(
20        Write : IN     std_logic;
21
22        addr  : IN     std_logic_vector (7 DOWNTO 0);
23        WN    : IN     std_logic_vector (15 DOWNTO 0);
24        read  : IN     std_logic;
25        WC    : OUT    std_logic_vector (15 DOWNTO 0)
26      );
27  END RAM ;
28
29  -- hds interface_end
30  ARCHITECTURE RAM OF RAM IS
31  TYPE MW_I0ram_typel IS ARRAY (255 DOWNTO 0) OF std_logic_vector(15 DOWNTO 0);
32     SIGNAL MW_I0ram_tablel : MW_I0ram_typel;
33
34
35  begin
36    I0write: PROCESS (addr, Write, WN)
37     VARIABLE itemp : INTEGER RANGE 255 DOWNTO 0;
38     VARIABLE iaddr : INTEGER RANGE 255 DOWNTO 0;
39     VARIABLE atemp : std_logic_vector(7 DOWNTO 0);
```

Programme VHDL de la RAM.

## ملخص

يركز العمل المقدم في هذا الموضوع على تصميم وتنفيذ شبكة عصبية اصطناعية على لوحة FPGA.

لقد استهدفنا كتطبيق في هذا العمل تصميم ومحاكاة شبكة عصبية من خلال التعلم تحت MATLAB، وتنفيذ مصنف عصبي من النوع Feed-Forward على بطاقة FPGA من عائلة Artix7 باستخدام لغة VHDL في بيئة Xilinx VIVADO.



**الكلمات المفتاحية**: الشبكة العصبية،التنفيذ،التصميم،FPGA،Feed-Forward،VHDL.



## Résumé

Les travaux présentés dans ce thème portent essentiellement sur la conception et l'implémentation d'un réseau de neurone artificiel sue une carte FPGA.

Nous avons ciblé comme application dans ce travail la conception et la simulation d'un réseau de neurone par l'apprentissage sous MATLAB, et l'implémentation d'un classificateur neuronal de type Feed-Forward sur une carte FPGA de la famille Artix7 par l'utilisation du langage VHDL sous l'environnement Xilinx VIVADO.



**Mots clés** : Réseau de neurone, Implémentation, Conception, FPGA, Feed-Forward, VHDL.



## Abstract

The work presented in this topic mainly concerns the design and the implementation of an atificial neuron network on the FPGA board.

We have targeted as an application in this work the design and simulation of a neural network by learning under MATLAB, and the implementation of a Feed -Forward type neural classifier on an FPGA card of the Artix7 family by using the VHDL language under the Xilinx VIVADO environment.



**Keywords:** Neural network, Implementation, Design, FPGA, Feed-Forward, VHDL.