الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université de Saad Dahleb Blida

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Projet de Fin d'Études

présenté par

GALIZRA Abdelwaheb

pour l'obtention du diplôme Master 2 en Électronique

option : système de vision et Robotique

Thème

Implémentation des réseaux de neurones artificiels sur circuit FPGA en utilisant la représentation en virgule fixe

Proposé par : Mme BOUGHERIRA Hamida

Année Universitaire 2013-2014

Remerciements

Mes remerciements s'adressent tout d'abord a dieu, toute la grâce pour m'avoir guidé et honoré par la lumière de la compréhension et de m'avoir accordé la connaissance de la science.

*Je remercie ma promotrice Mme**. BOUGHERIRA Hamida** qui m'a proposé ce sujet et qui a patiemment suivi mon travail et qui a prodigué ses précieux et minutieux conseils.*

Je profite par la même occasion, pour remercier l'ensemble des enseignants qui ont contribué à notre formation.

Mes remerciements s'adressent également aux membres du jury en acceptant d'évaluer ce modeste travail.

Dédicaces

A mes chers parents, mes frères et mes sœurs.

A ma femme et ma fille ***WAFAA***, pour leur patience, leurs sacrifices et leurs encouragements.

A tous mes amis et collègues pour leurs soutiens et encouragement

A toute la promotion de l'année universitaire 2013/2014 de l'institut d'électronique.

A tous mes collègues du ministère de la poste et des TIC.

ملخص:

يقدم هذا العمل طريقة لتنفيذ الكيان المادي الرقمي المبني على مصفوفة البوابات المبرمجة حلقيا (FPGA) للشبكات العصبية الاصطناعية الأمامية Feed Fawrd باستعمال لغة الكيان المادي VHDLفي بيئة ISE Xilinxو بالاعتماد على تمثيل الأعداد بصيغة النقطة الثابتة.

كلمات المفاتيح: الشبكات العصبية الاصطناعية، لغة VHDL، صيغة النقطة الثابتة.

Résumé :

Les travaux présentés dans cette thèse portent, essentiellement, sur l'implémentation des réseaux de neurones de type FFAN sur FPGA par l'utilisation du langage VHDL sous l'environnement ISE de Xilinx, et la représentation des données, codés en virgule fixe

Mots clés : Réseaux de neurones, Implémentation, FPGA, virgule fixe, Langage VHDL.

Abstract :

The work presented in this thesis are essentially about implementing neural networks FFAN guy on FPGA using VHDL under Xilinx ISE environment, and data representation, coded fixed point

Key words: Neural networks, Implementation, FPGA, Control.

Listes des acronymes et abréviations

ASIC: Application Specific Integrated Circuits.

CLB: Configurable Logic Block.

CPLD: Complex Logic Programmable Device.

FPGA: Field Programmable Gate Array.

FFANN: Feed Forward Artificial Neural Network.

IOB: Input Output Bloc.

LUT: Look Up Table.

LCA: Logic Cells Arrays.

LSB: Least Significant bit.

MLP: Multi-Layer-Perceptron.

MSB: Most Significant bit.

PAL: Programmable Array Logic.

RAM: Random Access Memory.

RNA: Réseaux de Neurones Artificiels.

RTL: Register Transfer Level.

SRAM: Static Random Access Memory.

VHDL: Very High Speed Integrated Hardware Description Language.

d_j : sortie désirée.

E : erreur quadratique.

F : fonction de transfert.

m : nombre de neurones à l'entrée.

n : le nombre de neurones à la sortie.

W_i : les poids.

x_i : l'entrée du i du neurone j.

y_j : sortie calculée.

η : coefficient d'apprentissage.

Table des matiéres

Liste des figures

Liste des tableaux

Introduction générale

L'évolution technologique permet de réaliser des fonctions de plus en plus complexes et des traitements de plus en plus rapides tout en consommant de moins en moins d'énergie. Les applications de traitement de l'information dans tous les domaines (médical, télécommunication,…et utilisent des algorithmes de plus en plus sophistiqués dont la complexité ne cesse de croître. L'implantation de ces algorithmes nécessite un traitement de données en temps réel tout en respectant des contraintes liées à la latence et/ou à la cadence.

En plus, la croissance de l'intégration des systèmes embarqués a amplifié le besoin de développement, d'une part, de nouveaux outils et méthodologies et d'autre part de création de nouvelles architectures tout en respectant des contraintes liées à l'application et aux domaines (temps réel, ressources d'implantation, consommation…etc).

Parmi les outils de prototypage les plus utilisés on trouve les FPGA (pour Field Programmable Gate Array) qui sont des circuits re-configurables utilisés dans divers domaines tels que le traitement d'images, le traitement de signal, ou la cryptologie.

Les RNA sont utilisés dans une grande variété d'applications comme l'identification et la commande des systèmes, l'approximation des fonctions, les systèmes de vison et traitement d'images, robotique, la reconnaissance des formes, etc… L'implémentation matérielle (hardware) des réseaux de neurones artificiels (RNA) sur un circuit FPGA permet d'obtenir des circuits de traitement rapides, puissants, et embarqués et répond ainsi aux besoins technologiques modernes.
Toutefois, l'implantation de ces applications se heurte souvent à des contraintes de précision, de surface, de consommation. L'implantation en virgule fixe est un type de représentation le plus utilisé dans les travaux effectués sur l'implémentation de RNA

sur circuits, car il représente le meilleur compromis entre la consommation des ressources, la fréquence et la précision.

L'essentiel du présent projet est d'élaborer une architecture neuronale de type MLP (MultiLayer Perceptron) implantée sur FPGA illustrée par une application permettant la classification des objets détectés par un robot mobile. Le présent mémoire comporte trois chapitres :

- Le premier chapitre sera consacré à une étude générale des réseaux de neurones, dans lequel nous présenterons sommairement, les réseaux de neurones et leurs différents domaines d'applications. Nous abordons le circuit ciblé par l'implémentation, le FPGA, avec son architecture et les différentes méthodes de sa programmation, ainsi que le langages VHDL et environnements utilisés.
- Le chapitre deux traites la technique de représentation des nombres pour les opérations arithmétiques. Les différents détails d'une technique à virgule fixe sont introduits à travers différents exemples.
- Ensuite, dans le chapitre trois, sera consacré aux résultats, simulation et la conception d'un neurone élémentaire et nous avons présenté aussi le modèle du perceptron multicouche à travers un exemple de la classification.

  On terminera par une conclusion générale.

Chapitre 1 Généralités

1.1 INTRODUCTION

Cette partie est consacrée à une présentation générale des RNA et leur utilisation, des circuits FPGA, des langages VHDL, ainsi que des outils de conception ISE de Xilinx. Tout d'abord, nous présenterons le neurone formel qui est l'élément de base des RNA et nous en étudierons quelques caractéristiques. Nous exposerons, par la suite, les principaux types de RNA et nous donnerons un aperçu de quelques domaines d'application des réseaux de neurones. Nous présenterons, ensuite, les circuits FPGA avec leur architecture ainsi que les différentes méthodes de leur programmation, le langage VHDL et l'outil de conception ISE de Xilinx.

1.2 LES NEURONES ARTIFICIELS

1.2.1 Le neurone formel

La Figure 1 montre la structure d'un neurone artificiel. Chaque neurone reçoit un nombre variable d'entrées en provenance des neurones amont. A chacune de ces entrées est associée une abréviation de Weight (poids en Anglais) représentatif de l'entrée X, dont la valeur est poids W .Chaque neurone élémentaire est doté d'une fonction de transfert (fonction d'activation) qui donne une sortie unique Y [4].

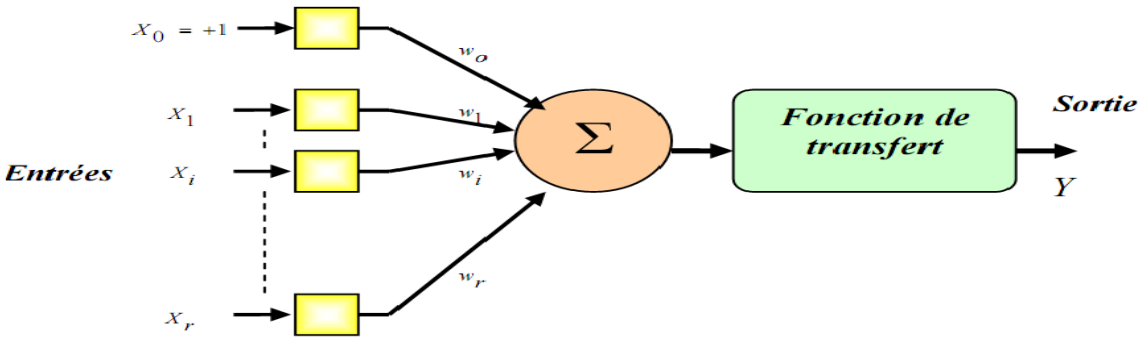


Figure 1.Le neurone artificiel

On distingue deux phases : La première est le calcul de la somme des entrées X i, à partir de cette valeur, une fonction de transfert f calcule la valeur de l'état du neurone selon l'expression suivante :

$$Y = f\left(\sum_{i=0}^{r} W_i . X_i\right)$$

1.2.2 La fonction d'activation

L'état que peut prendre un neurone est en général binaire mais peut aussi être une valeur discrète ou bien continue. Selon le type de cet état, on aura donc différentes formes pour la fonction de transition (d'activation)**.**

Le tableau suivant représente les fonctions d'activation les plus utilisées actuellement

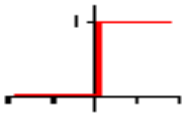
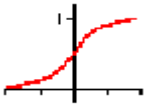
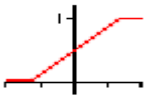
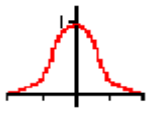
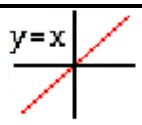
Fonction binaire ou seuil		$f(x) = \begin{cases} 0 & if \quad x < 0 \\ 1 & if \quad x \geq 0 \end{cases}$
Fonction sigmoïde		$f(x) = \frac{1}{1 + e^{-\beta x}}$
Fonction linéaire seuillée		$f(x) = \begin{cases} 0 & if \quad x \leq x_{\min} \\ mx+b & if \quad x_{\max} > x > x_{\min} \\ 1 & if \quad x_{\max} \end{cases}$
Fonction gaussienne		$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$
Fonction identité		$f(x) = x$

Tableau 1.les fonctions d'activations [2].

1.3 PRESENTATION DES RESEAUX DE NEURONES

Un neurone élémentaire est limité en ces applications, en effet, un neurone réalise une simple fonction non linéaire, paramétrée, de ses variables d'entrées. L'intérêt des neurones réside dans la propriété qui résultent de leur association dans une structure, par une certaine logique d'interconnexion, cette structure est appelée : le réseau de neurones ou bien par l'abréviation ANN (Artificiel Neural Network). Le comportement collectif ainsi obtenu permet de réaliser des fonctions d'ordre supérieur par rapport à la fonction élémentaire réalisée par un neurone [5].

Dans un tel réseau, les entrées d'un neurone sont soit les entrées du réseau globale, soit les sorties d'autres neurones. Les valeurs des poids du réseau sont en général, déterminées par une opération dite l'apprentissage. Suivant la logique d'interconnexion choisie, les réseaux de neurones se distinguent en deux grandes familles : les réseaux non bouclés (statique) et les réseaux bouclés (dynamique), les Figures (2 et 3) illustrent le schéma synoptique des deux réseaux respectivement.

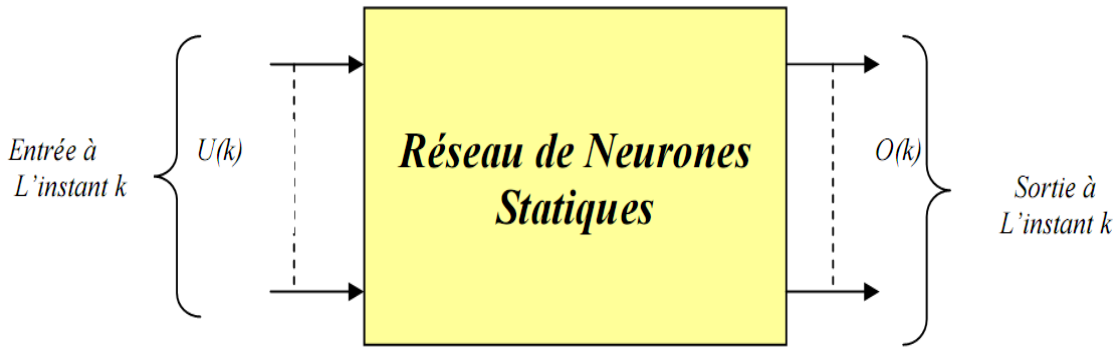


Figure 2.Schéma synoptique d'un réseau de neurones non bouclé (Statique) [3].

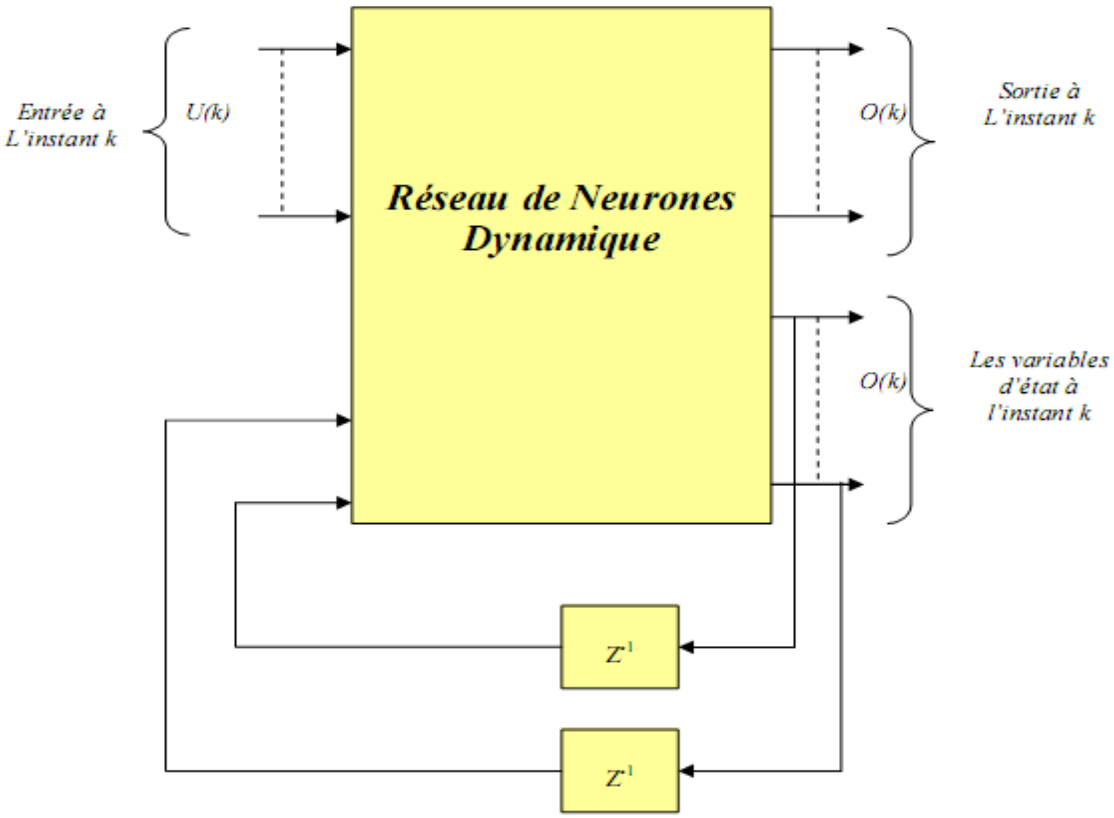


Figure 3. Schéma synoptique d'un réseau de neurones bouclé (dynamique) [3].

1.4 CLASSIFICATION TOPOLOGIQUE DES RESEAUX DE NEURONES

La topologie ou architecture d'un RNA est la manière selon laquelle les neurones sont connectés et organisés. Généralement, les RNA peuvent être d'une connectivité totale ou chaque neurone est relié à tous les autres neurones du réseau ; ou d'une connectivité locale dont les neurones ne sont liés qu'à leurs voisins. Alors, On distingue deux topologies des RNA : les réseaux de neurones non bouclés ou réseaux « feed-forward » et les réseaux de neurones bouclés ou réseaux « feed-back » [5].

1.4.1 Les réseaux de neurones non bouclés

Appelés aussi réseaux « proactifs » ou « de type perceptron » ou « feed-forward ». Un réseau non bouclé se caractérise par la propagation des signaux dans un seul sens, de la couche d'entrée vers la couche de sortie, sans utilisation d'aucune boucle de rétroaction. Il est généralement statique car les entrées et les sorties sont indépendantes du temps [1].

Ce type de RNA est utilisé principalement pour effectuer des tâches d'approximation de fonction non linéaire, de classification ou de modélisation de processus statiques non linéaires(Figure.4).

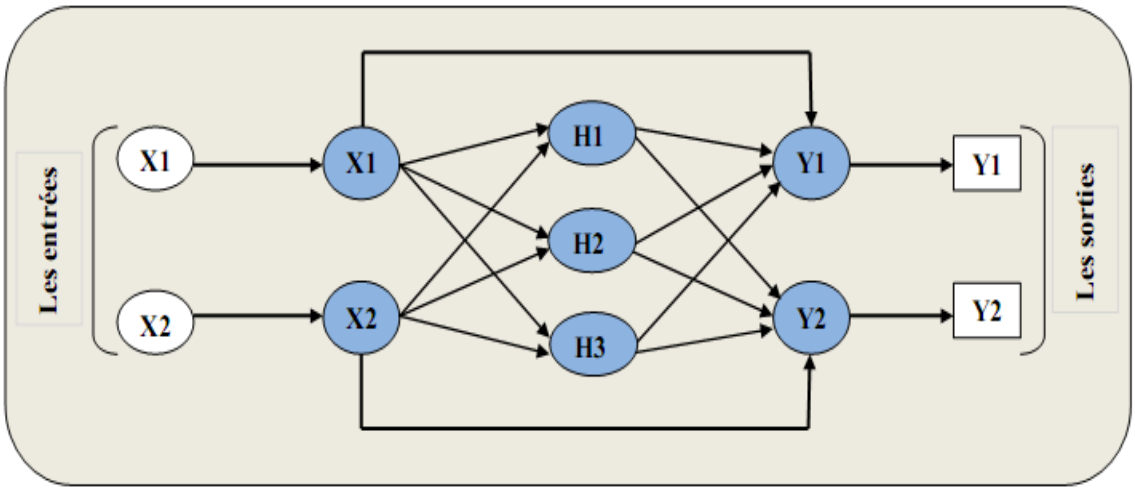


Figure 4.Architecture d'un réseau de neurone non bouclé.

1.4.2 Les réseaux de neurones bouclés

Les réseaux « récurrents » ou bien « feed-back » se caractérisent par la présence, au moins, d'une boucle de rétroaction des neurones de sorties vers les neurones d'entrée. Ces réseaux sont dynamiques ; ils régissent par des équations aux différences non linéaires à cause des retards associés aux connexions (Figure.5). Ils sont utilisés fréquemment à l'affectation des tâches de modélisation de systèmes dynamiques, de commande de processus, ou de filtrage [1].

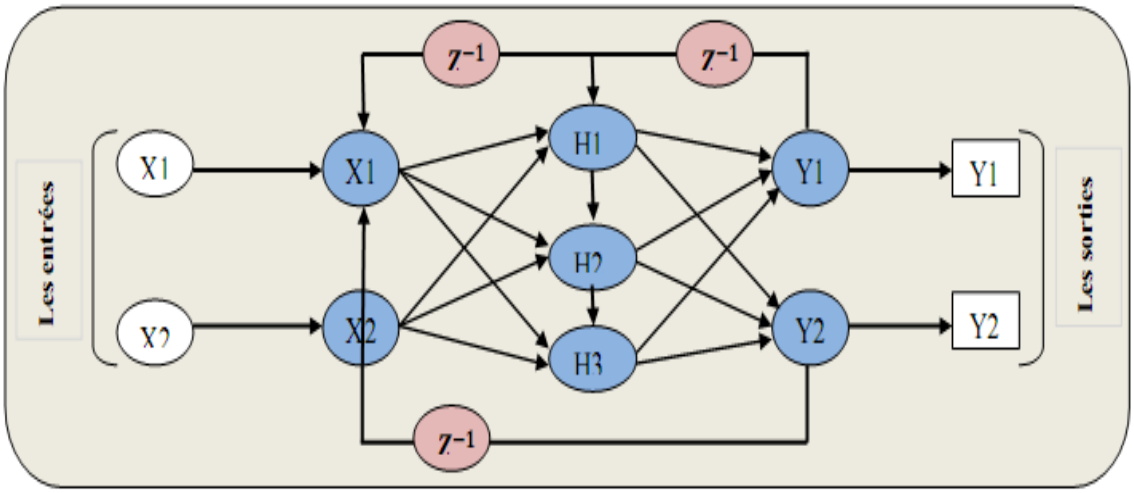


Figure 5. Architecture d'un réseau de neurone bouclé.

1.5 L'APPRENTISSAGE DES RESEAUX DE NEURONES

L'apprentissage est le but principal du développement de modèles à base des réseaux de neurones. Il est réalisé par la modification des poids de connexion du réseau, généralement par des algorithmes spécifiques, afin d'obtenir des valeurs optimales appropriées a ces poids [2].

1.5.1 L'apprentissage non supervisé

Ce type d'apprentissage appelé aussi auto-organisation et compétitif, consiste à présenter au réseau seulement les vecteurs d'entrée qui seront groupés en classes par extraction des propriétés. Ce réseau doit détecter des points communs aux exemples présentés, par la modification des poids, afin de fournir la même sortie pour des entrées aux caractéristiques proches(Figure.6).

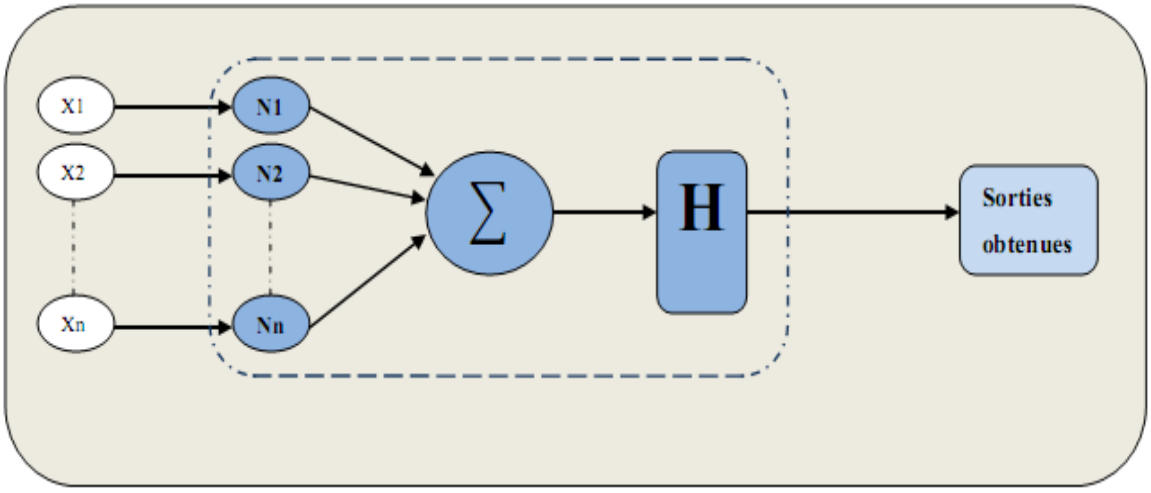


Figure 6.Mode d'apprentissage non supervisé.

1.5.2 L'apprentissage supervisé ou associatif

Ce type d'apprentissage est basé sur la minimisation d'une fonction d'erreur ou fonction de cout à chaque itération de l'algorithme (Figure.7). Il consiste à présenter au réseau l'entrée et la sortie désirée pour qu'il puisse adapter ses poids synaptiques dans le but d'obtenir la sortie souhaitée [1].

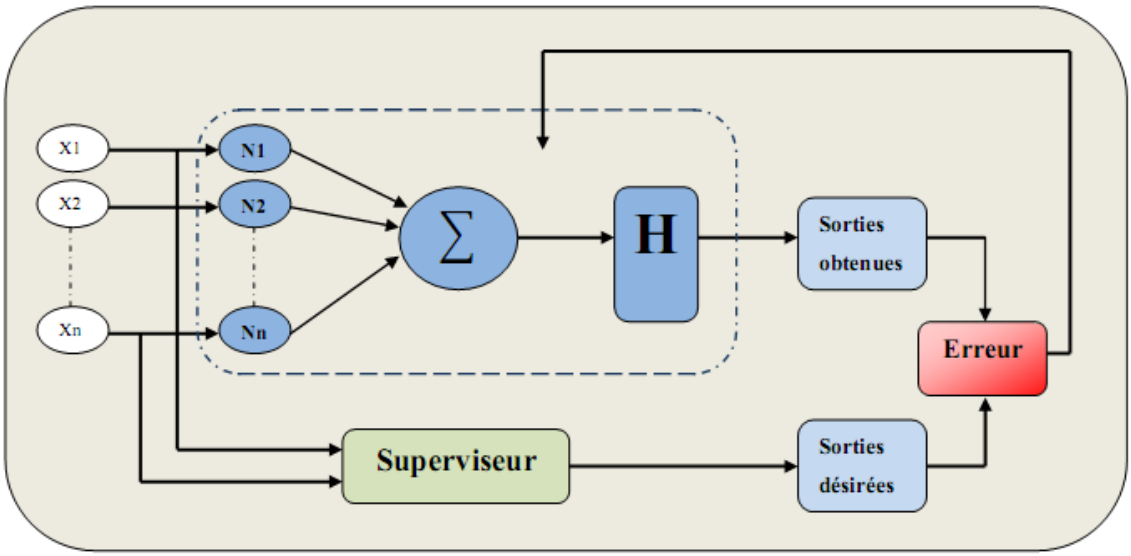


Figure 7.Mode d'apprentissage supervisé.

1.6 LES LIMITATIONS D'UN RESEAU DE NEURONES

1.6.1 Avantages

- Implémentation du parallélisme.
- Apprentissage.
- Robustesse: données bruitées ou incomplètes.
- Généralisation à des Modèles similaires.
- Trouve des solutions aux problèmes non linéaires.
- Trouve des solutions aux problèmes qui n'ont pas une modélisation.

1.6.2 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 extensibles (l'ajout d'un neurone).

1.7 LES CIRCUITS FPGA

Le concept de logique programmable a été proposé par G. Estrin en 1963. L'apparition des FPGA s'est d'abord fait au travers des circuits logiques programmables de type PAL (Programmable Array Logic) et avec les évolutions en électronique, différentes familles

de circuits programmables ont commencé à apparaître : les CPLD (Complex Logic Programmable Device), puis les FPGA (Field Programmable Gate Arrays).Les premiers circuits FPGA ont été commercialisés par la firme XILINX en 1985 et c'est un marché qui n'a pas cessé de croître depuis. Au fur et à mesure que la complexité des FPGA s'est développée, leurs possibilités d'emploi se sont accrues jusqu'à concurrencer sérieusement les circuits ASIC pour des petits volumes de production. Le recours aux ASIC impose en effet des temps de développement et de fabrication de l'ordre de plusieurs mois. Les FPGA, par contre, permettent une reconfiguration à volonté dans un temps très court (de l'ordre de quelques millisecondes) et donc une évolution des circuits au cours de leur période d'exploitation [2].

1.7.1 Caractéristiques des circuits FPGA

Le premier avantage apporté par les circuits FPGA est la souplesse de la programmation, ce qui permet de multiplier les essais, d'optimiser de diverses manières l'architecture développée et de vérifier à divers niveaux de la simulation la fonctionnalité de cette architecture [3].

Le second avantage des FPGA est la possibilité de la reconfiguration dynamique partielle ou totale des circuits, ce qui permet, d'une part, une meilleure exploitation du composant et une réduction de la surface de silicium employé, d'autre part la programmation en temps réel (quelques microsecondes) tout ou une partie du circuit.

Le composant FPGA réalise par conséquence un bon compromis Flexibilité/ Performance (Figure.8).

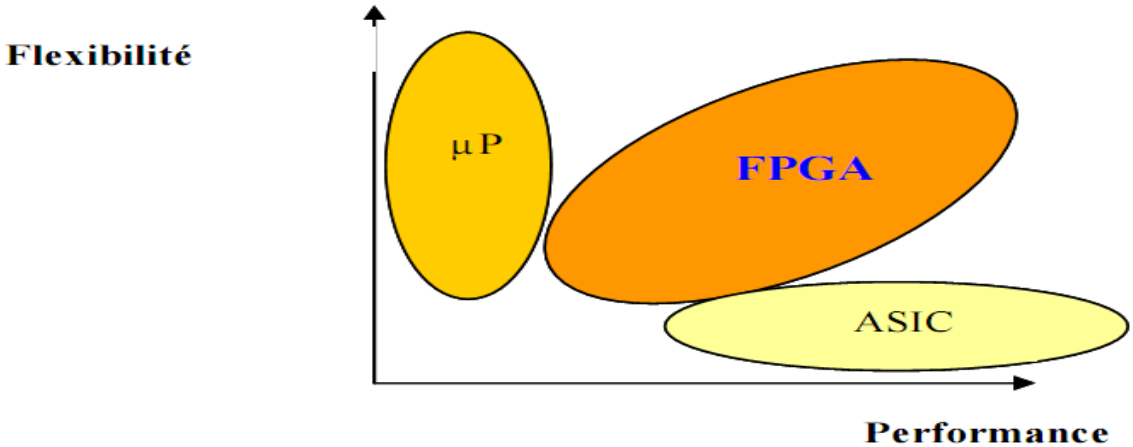


Figure 8.Les circuits FPGA réalisent le compromis Flexibilité/performance [3].

1.7.2 Architecture des FPGA

Les circuits FPGA sont constitués d'une matrice de blocs logiques programmables entourés de blocs d'entrée sortie programmable. L'ensemble est relié par un réseau d'interconnexions programmable [3].

Les FPGA sont bien distincts des autres familles de circuits programmables tout en offrant le plus haut niveau d'intégration logique. Il existe actuellement plusieurs fabricants de circuits FPGA, Xilinx et Altera sont les plus connus, et plusieurs technologies et principes organisationnels (Figure.9).

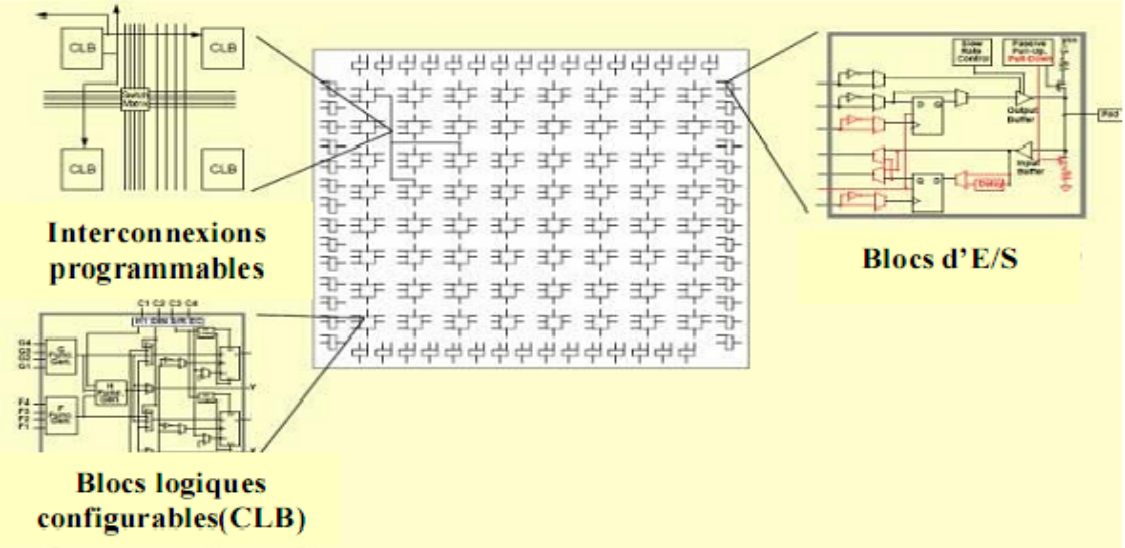
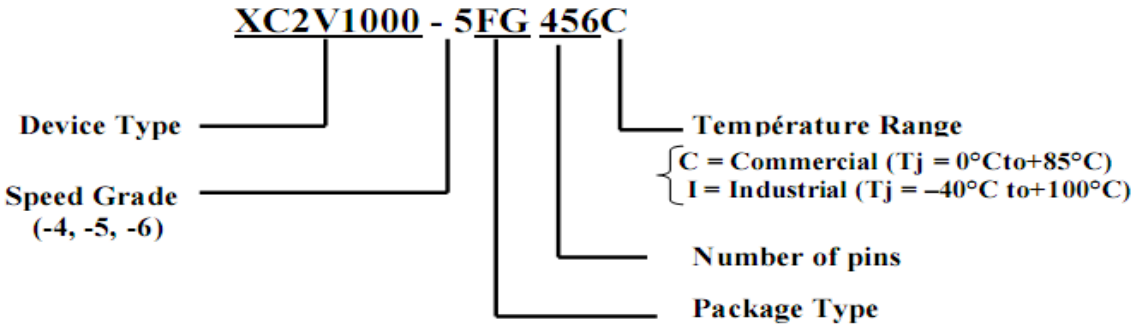


Figure 9.Architecture interne du FPGA [3].

1.7.3 Nomenclature des circuits FPGA

Les circuits FPGA suivent la nomenclature suivante **[2]**:



- ***Device type*** : représente le type de la famille dans notre exemple VIRTEX-II
- ***Speed Grade*** : représente la vitesse du composons selon la technologie utilisée dans la fabrication du circuit.

1.7.4 Application des FPGA

Les FPGA sont utilisés dans de nombreuses applications, on en cite dans ce qui suit quelques unes:

- Prototypage de nouveaux circuits ;
- Fabrication de composants spéciaux en petite série ;
- Adaptation aux besoins rencontrés lors de l'utilisation ;
- Systèmes de commande à temps réel ;
- DSP (Digital Signal Processor) ;
- Domaine de robotique ;
- Imagerie médicale **[20]**.

1.7.5 Programmation des circuits FPGA

Pour programmer un FPGA, il faut commencer par décrire le design en utilisant un langage de description matériel (tel que VHDL, Verilog, ABEL,...). Le synthétiseur va ensuite générer la liste d'interconnexion (netlist) qui permettra de simuler le placement de tous les composants au sein du FPGA, d'effectuer le routage entre les différentes cellules logiques et de calculer le délai des différents signaux. Si le routage a pu être correctement effectué, le bit stream qui sera prêt à être envoyé vers le FPGA est alors généré [3].

1.8 LANGAGE DE DESCRIPTION VHDL

1.8.1 Bref historique

Le VHDL (Very-high-speed-integrated-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 depuis 1987, qui aurait du 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 du langage VHDL s'est faite en 1993 (IEEE 1164) et en 1996, la norme 1076.3 a permis de standardiser la synthèse VHDL [3].

1.8.2 Les avantages du VHDL

Il faut avoir à l'esprit que ce langage permet de matérialiser les structures électroniques d'un circuit. En effet les instructions écrites dans ce langage se traduisent par une configuration logique de portes et de bascules qui est intégrée à l'intérieur des circuits PLDs. C'est pour cela qu'on préfère parler de description VHDL que de langage ou programmation VHDL [3].

Le VHDL est un langage qui permet de faire :

a. ***Spécification***

Le langage VHDL est très bien adapté à la modélisation des systèmes numériques complexes grâce à son niveau élevé d'abstraction. Le partitionnement en plusieurs sous ensembles permet de subdiviser un modèle complexe en plusieurs éléments prêts à être développés séparément.

b. ***Simulation***

Le VHDL est également un langage de simulation. Pour ce faire, la notion de temps, sous différentes formes, y a été introduite. Des modules, destinés uniquement à la simulation, peuvent ainsi être créés et utilisés pour valider un fonctionnement logique ou temporel du code VHDL. La possibilité de simuler avec des programmes VHDL devrait considérablement faciliter l'écriture de tests avant la programmation du circuit et éviter ainsi de nombreux essais sur un prototype qui sont beaucoup plus coûteux et dont les erreurs sont plus difficiles à trouver. Bien que la simulation offre de grandes facilités de test, il est toujours nécessaire de concevoir les circuits en vue des tests de fabrication, c'est-à-dire en permettant l'accès à certains signaux internes [20].

c. ***Conception***

Le VHDL permet la conception de circuits avec une grande quantité de portes. Les circuits actuels comprennent, pour les FPGA par exemple, entre 500 et 1000'000 portes et ce nombre augmente très rapidement. L'avantage d'un langage tel que celui-ci par rapport aux langages précédents de conception matérielle est comparable à l'avantage d'un langage informatique de haut niveau (Pascal, ADA, C) vis-à-vis de l'assembleur.

d. ***La Synthèse***

La synthèse permet de générer automatiquement à partir du code VHDL un schéma de câblage permettant la programmation du circuit cible. La description du code pour

une synthèse est, en théorie, indépendante de l'architecture du circuit. En pratique, le style utilisé aura une influence sur le résultat de la synthèse, influence liée au type de circuit et au synthétiseur utilisé. Le code devra être optimisé pour un type circuit. Si le besoin d'optimisation n'est pas très important, cette partie peut se dérouler très rapidement, alors qu'au contraire, si le besoin d'optimisation est grand, les subtilités pour coder de manière adéquate en VHDL obligeront le concepteur à y consacrer beaucoup de temps. D'ailleurs, ce n'est souvent plus en VHDL que l'on optimisera, car ce langage se révèle, pour l'instant en tout cas, peu adapté à cet usage. Il faudra travailler à un niveau plus proche du circuit ou acheter des macros pré-existantes.

1.8.3 Structure d'une description VHDL simple

La structure typique d'une description VHDL est composée de 2 parties indissociables qui sont : l'entité et l'architecture.

a. ***L'entité***

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

b. ***L'architecture***

L'architecture (ARCHITECTURE) contient les instructions VHDL permettant de décrire et de réaliser le fonctionnement attendu. Elle représente la structure interne de la description. Les entités et architectures sont des unités de conception dites primaire et secondaire. Un couple entité-architecture donne une description complète d'un élément ; ce couple est appelé modèle.

c. ***Déclaration des bibliothèques***

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques. L'IEEE les a normalisées et plus particulièrement la bibliothèque IEEE1164. Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques, etc.

Exemple :

```
 9  Library IEEE;
10  Use IEEE.STD_LOGIC_1164.ALL;
11  Use IEEE.NUMERIC_STD.ALL;
12  Use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

d. ***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 défini en entrée **(in)**, en sortie **(out)**, en entrée/sortie **(inout)** ou en buffer, c'est-à-dire qu'il est en sortie mais il peut être utilisé comme entrée à l'intérieur de la description.

Exemple:

```
15  entity ana is
16  port (DEC :in std_logic_vector(3 downto 0);
17  SEG:out std_logic_vector(6 downto 0)  );
18  end ana ;
```

1.8.4 Etapes nécessaires au développement d'un projet sur FPGA

Le développement en VHDL nécessite l'utilisation de deux outils : le simulateur et le synthétiseur. Le premier va nous permettre de simuler notre description VHDL avec un fichier de simulation appelé « test-bench » ; cet outil interprète directement le langage VHDL et il comprend l'ensemble du langage. L'intégration finale dans le circuit cible est réalisée par l'outil de placement et routage. La figure 10 donne les différentes étapes nécessaires au développement d'un projet sur circuit FPGA **[4]**.

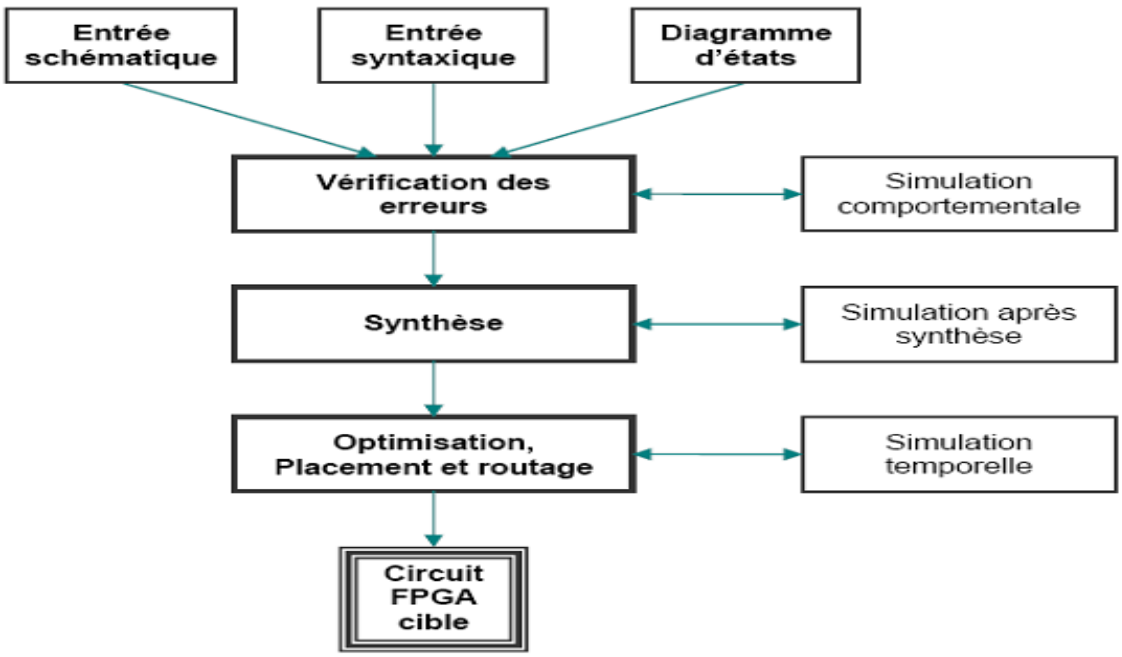


Figure 10.Organisation fonctionnelle de développement d'un projet sur circuit FPGA.

a. ***Saisie du texte VHDL***

La saisie du texte VHDL se fait sur le logiciel « ISE Xilinx Project Navigator ». Ce logiciel propose une palette d'outils permettant d'effectuer toutes les étapes nécessaires au développement d'un projet sur circuit FPGA. Il possède également des outils permettant de mettre au point une entrée schématique ou de créer des diagrammes d'état, qui peuvent être utilisés comme entrée au lieu du texte VHDL. La figure 11 montre comment se présente le logiciel «ISE Xilinx Project Navigator» [11].

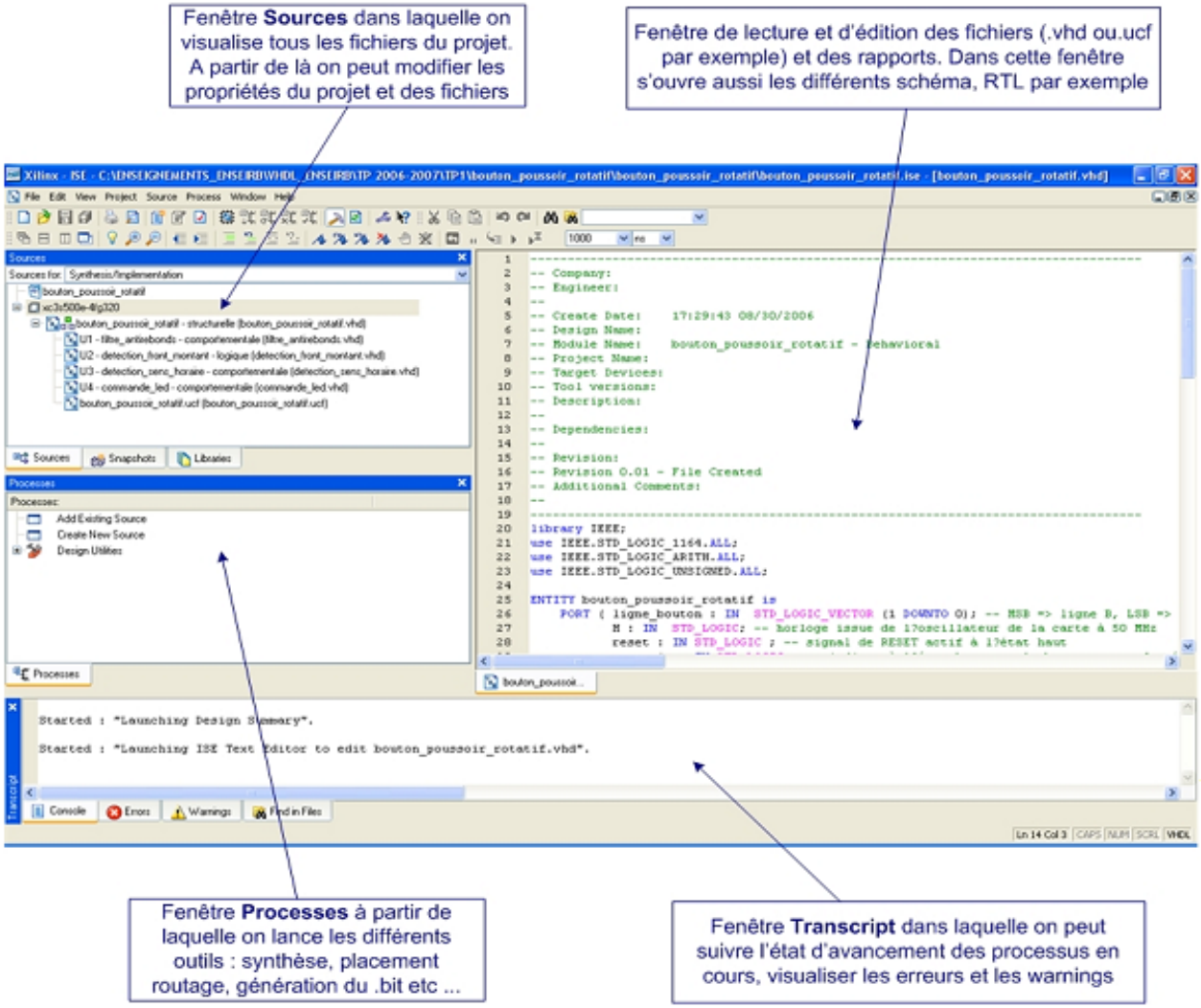


Figure 11.Écran principal du navigateur de projet de Xilinx ISE.

b. ***Vérification des erreurs***

Cette étape est effectuée en appuyant sur le bouton (check syntax) ; elle permet de vérifier les erreurs de syntaxe du texte VHDL et d'afficher les différentes alarmes (warnings) liées au programme [11].

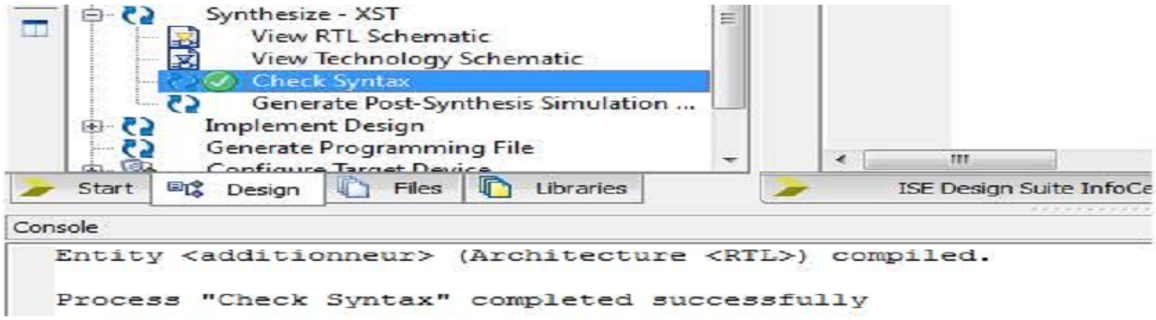


Figure 12.Fenêtre de la vérification des erreurs.

c. ***La simulation***

La simulation permet de vérifier le comportement d'un design avant ou après l'implémentation dans le composant cible.

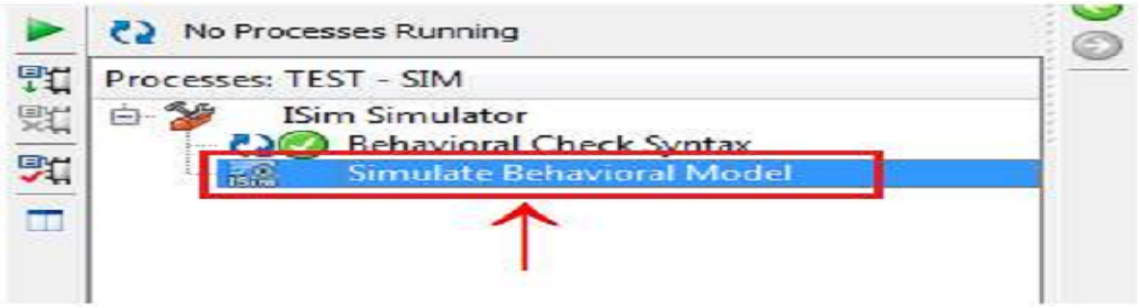


Figure 13.Fenêtre de la simulation.

Chapitre 2 La réalisation des opérations arithmétique en virgule fixe

2.1 INTRODUCTION

Afin de réaliser l'opération complète modélisant la sortie d'un neurone, nous commençons par la réalisation des opérateurs arithmétiques nécessaires pour le fonctionnement.

La première étape est la description VHDL des opérateurs d'addition et de multiplication. Notons que dans la description des opérateurs arithmétiques nécessaires pour la modélisation des neurones artificiels, il est important de choisir la technique de représentation des nombres. Les représentations des nombres en format simple, en virgule fixe ou en virgule flottante est toujours un choix qui dépend de l'application à implanter.

Dans notre projet nous s'intéressons seulement à la représentation en virgule fixe où les données sont codées sur 19 bits, 5 bits avant la virgule et 14 bits après la virgule.

2.2 REPRESENTATION EN VIRGULE FIXE

Les données en virgule fixe sont composées d'une partie fractionnaire et d'une partie entière pour lesquelles le nombre de bits alloués reste figé au cours du traitement. La figure 12 représente une donnée en virgule fixe composée d'un bit de signe et « b-1 » bits repartis entre la partie entière et la partie fractionnaire. Soit « m » le paramètre représentant la position de la virgule par rapport au bit le plus significatif (MSB) et « n » la position de la virgule par rapport au bit le moins significatif (LSB).

Dans le cadre d'une donne signée, les différents paramètres respectent la relation :

$$b = m + n + 1$$

Les paramètres « m » et « n » sont des entiers et représentent la distance, en nombre de bits, entre la virgule et les bits MSB et LSB.

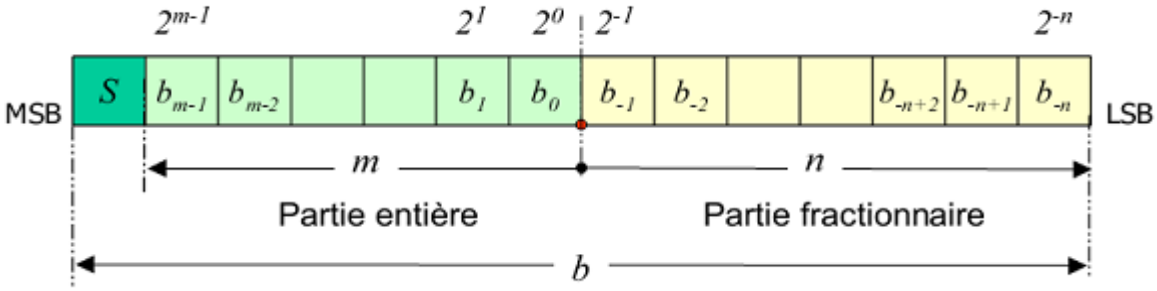


Figure 14.Représentation des données en virgule fixe***.***

2.3 LES OPERATIONS ARITHMETIQUES

Les opérateurs arithmétiques ou logiques de base : +, - , NOT, * , AND , >, > , = , « shift left » et « shift right » sont des fonctions courantes disponibles dans des paquetages standards tels que IEEE.NUMERIC_STD. Le synthétiseur les accepte tels qu'ils sont avec des types « integer », « signed » ou « unsigned ».

2.3.1 L'opération d'addition

L'addition de deux opérandes a et b nécessite qu'ils possèdent un format commun. Le type de représentation, la longueur de la partie entière et la longueur de la partie fractionnaire doivent être identiques pour les deux opérandes. Si cette condition n'est pas respectée il est nécessaire de modifier le format des opérandes afin d'obtenir un format identique (bc, mc, nc).Le format commun garantissant l'absence de perte d'information est présenté par la figure 14.

Pour les données ayant un format différent du format commun, il est nécessaire d'étendre le nombre de bits des parties entières et fractionnaires en suivant certaine règles, Pour la partie fractionnaire, les (nc-na) bits supplémentaires sont mis à 0.Pour la partie entière, le bit de signe est étendu. Dans le cas du complément à 2, les (mc-ma) nouveaux bits prennent la valeur du bit de signe.

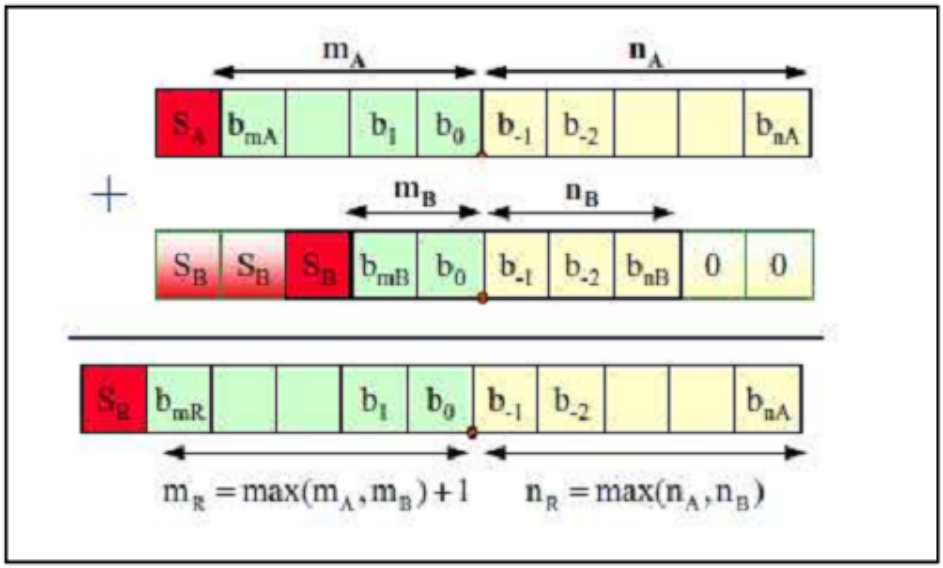


Figure15.Addition en virgule fixe.

a. ***Additionneur 1-bit***

Lorsque on additionne deux bits x et y, le résultat est la somme S des chiffres et la retenue R. La table de vérité est la suivante :

X	Y	**S**	**R**
0	0	**0**	**0**
0	1	**1**	**0**
1	0	**1**	**0**
1	1	**0**	**1**

Tableau 2.Table de vérité d'additionneur 1-bit.

On en déduit que S = x'.y + x.y' = x⊕y , et que R = x.y.

b. ***Additionneur n-bits***

Pour effectuer l'addition de deux nombres de « n » bits, il suffit de chaîner entre eux « n » additionneurs 1bit complets.

La retenue est ainsi propagée d'un additionneur à l'autre. Un tel additionneur est appelé un additionneur série. Bien que tous les chiffres des deux nombres de n-bits X et Y soient disponibles simultanément au début du calcul, à t =0, le temps de calcul est déterminé par la propagation de la retenue à travers les « n » additionneurs 1-bit.

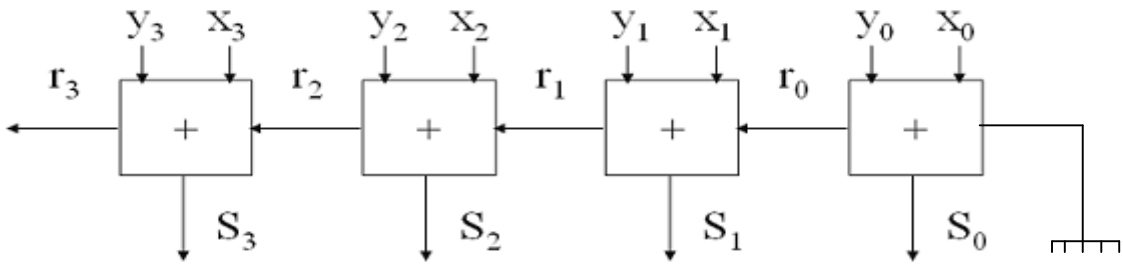


Figure 16.Additionneur 4 bits.

c. ***Réalisation d'un additionneur***

Un aperçu sur la description VHDL d'un additionneur, en utilisant la représentation en virgule fixe est donne par la figure 17.

```
1  ------------------------additionneur-----------------------
2  ------------------------------------------------------------
3  library IEEE;
4  library IEEE_proposed;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use IEEE_proposed.fixed_pkg.all;
7  use IEEE_proposed.fixed_float_types.all;
8
9  entity add is
10   port (
11     X : in  sfixed (4 downto -14);
12     y : in  sfixed (4 downto -14);
13     P : out  sfixed (4 downto -14));
14 end add;
15
16 architecture arch_add of add is
17   signal s:sfixed(4 downto -14);
18 begin
19
20         s<=resize(x+y,s);
21         p<=s;
22
23 end arch_add;
```

Figure 17.la description VHDL d'un additionneur en virgule fixe.

Dans notre bibliothèque l'opérateur d'addition « + » gère les retenues entrantes et sortantes.

En clair, les deux entrées et la sortie doivent être de même taille :

N bits + N bits = (N+1) bits, 1 bit pour les calculs de débordement.

Le résultat de la simulation est le suivant :

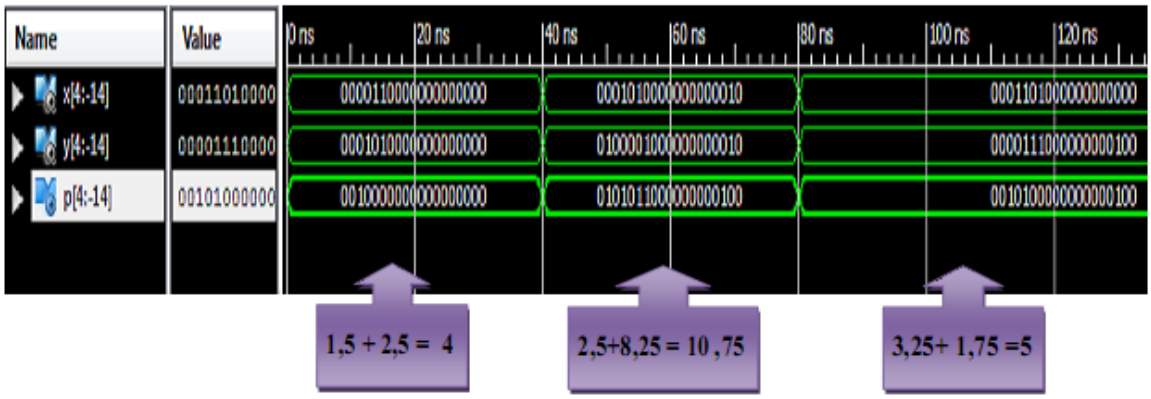


Figure 18.Simulation d'un additionneur.

2.3.2 L'opération de multiplication

Pour une multiplication, les deux opérandes doivent posséder la même représentation mais le nombre de bits réservés pour chaque partie peut être différent. Néanmoins, il est nécessaire avant d'effectuer l'opération d'étendre le bit de signe [4]. La multiplication de deux nombres en virgule fixe entraîne le doublement du bit de signe, celui-ci peut être éliminé automatiquement à l'aide d'un décalage à gauche. Pour un codage en complément à 2 on peut considérer que le bit de signe redondant appartient à la partie entière. Le format du résultat de la multiplication de deux opérandes a et b est représenté par la figure 19 :

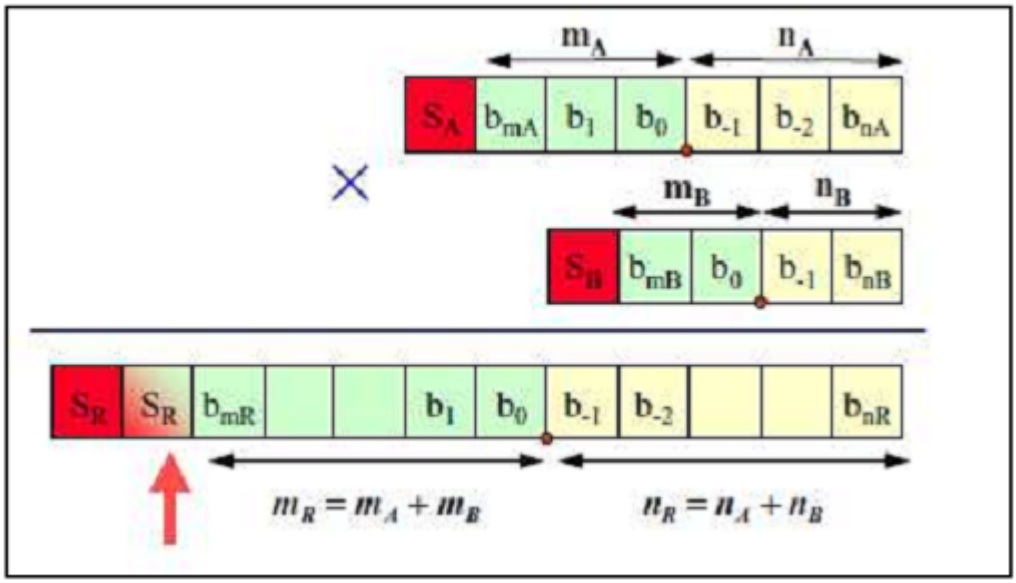


Figure19.Multiplication en virgule fixe.

La multiplication est une opération compliquée à réaliser et consommatrice de ressources en portes logique Il faut remarquer tout d'abord qu'un décalage à gauche

d'un mot binaire correspond à une multiplication par 2 et qu'un décalage à droite correspond à une division par 2.

La multiplication consiste en une série d'addition de la version décalée du multiplicande là où le bit correspondant du multiplieur vaut 1 (algorithme « Shift and Add»). Il suffit pour s'en persuader de regarder l'exemple suivant :

(14) multiplicande		1110
× (11) multiplieur	× 1011	
		1110
		1110
		0000
		1110
(154) produit		10011010

Il faut tout de suite remarquer que dans notre exemple, 4 bits x 4 bits = 8 bits. D'une manière générale : N1 bits x N2 bits = (N1+ N2) bits.

a. ***Réalisation un multiplieur en virgule fixe***

```
1  ------------------------- multiplieur ---------------------
2  -----------------------------------------------------------
3  library IEEE;
4  library IEEE_proposed;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use IEEE_proposed.fixed_pkg.all;
7  use IEEE_proposed.fixed_float_types.all;
8
9  entity Multiplier is
10   port (
11     X, Y : in  sfixed (4 downto -14);
12     P    : out  sfixed (4 downto -14));
13 end Multiplier;
14
15 architecture arch_mult of Multiplier is
16     signal z:sfixed(4 downto -14);
17 begin
18      z<=resize(x*y,z);
19      p<=z;
20 end arch_mult;
```

Figure 20.La description VHDL d'un multiplieur en virgule fixe***.***

La simulation de ce multiplieur est décrite par la figure 21

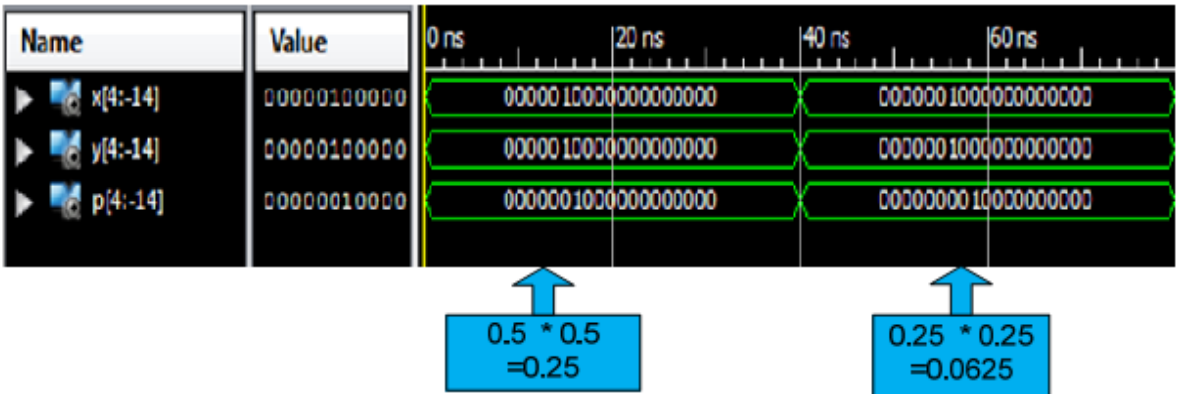


Figure 21.Simulation d'un multiplieur.

2.3.3 L'opération de soustraction

a. ***Demi-soustracteur***

La table de vérité pour un demi-soustracteur est la suivante :

A	B	**D**	**C**
0	0	**0**	**0**
0	1	**1**	**1**
1	0	**1**	**0**
1	1	**0**	**0**

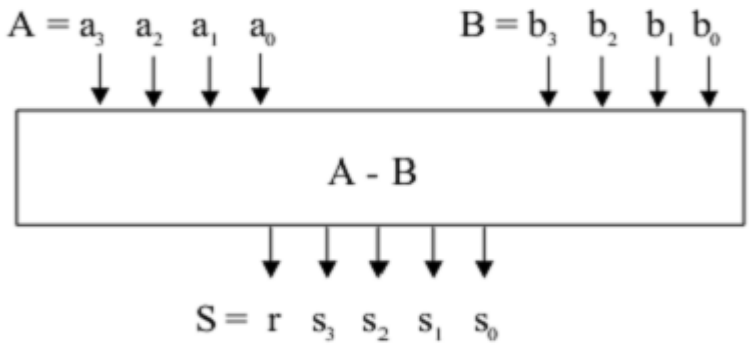
Tableau 3. Table de vérité du demi-soustracteur.

Où D représente le résultat de la soustraction A – B et C la retenue. Nous en déduisons les expressions logiques définissant D et C :

$$\begin{cases} D = \overline{A}\,B + A\,\overline{B} = A \oplus B \\ C = \overline{A}\,B \end{cases}$$

b. ***Soustracteur n-bits***

Ce paragraphe présente la structure de base permettant de réaliser la soustraction entre deux nombre binaires de n bits.



La procédure de soustraction de nombre binaire est semblable à celle utilisée en décimal c'est à dire :

- le signe du résultat est le signe du nombre le plus grand en valeur absolue.
- le résultat est obtenu en soustrayant le nombre le plus petit en valeur absolue du nombre le plus grand.

111	011
101,0 (5)	1100 (12)
−011,1 (-3.5)	−0011 (-3)
001,1 (1.5)	1001 (9)

```
1  ------------------------ soustracteur-----------------------
2  ------------------------------------------------------------
3  library IEEE;
4  library IEEE_proposed;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use IEEE_proposed.fixed_pkg.all;
7  use IEEE_proposed.fixed_float_types.all;
8
9  entity soust is
10   port (
11     X : in  sfixed (4 downto -14);
12     y : in  sfixed (4 downto -14);
13     P : out  sfixed (4 downto -14));
14 end soust;
15
16 architecture arch_sous of soust is
17   signal s : sfixed(4 downto -14);
18 begin
19
20         s <= resize(x-y,s);
21         p<=s;
22
23 end arch_sous;
```

Figure 22.La description VHDL d'un soustracteur en virgule fixe.

La simulation de ce soustracteur est décrite par la figure 23.

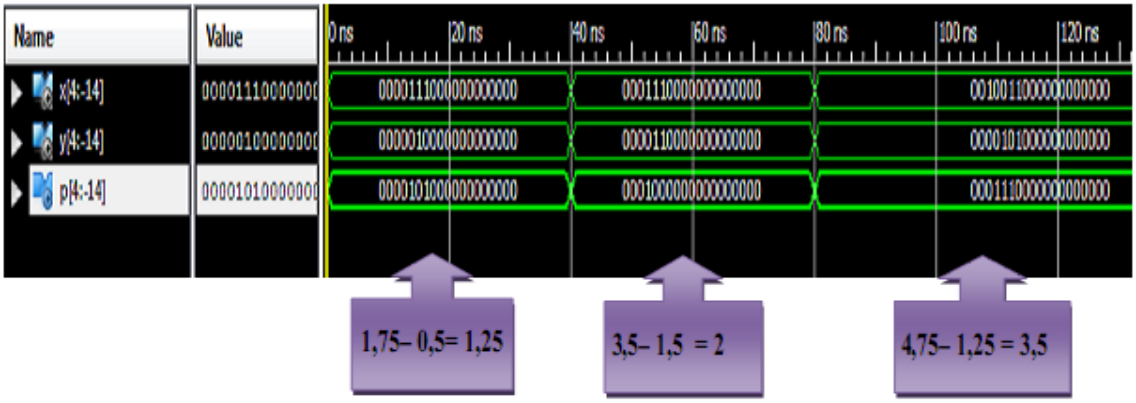


Figure 23.Simulation d'un soustracteur.

2.4 CONCLUSION

Dans ce chapitre, nous avons détaillé la manière adoptée pour la représentation des nombres pour les différentes opérations arithmétiques. Nous avons choisi de travailler avec la technique de la virgule fixe pour représenter la dynamique des nombres traités par le réseau de neurones qu'on désire concevoir.

La conception du circuit électronique numérique simulant l'architecture neuronale étant notre objectif. Dans le chapitre suivant, nous présenterons notre modèle de neurone en architecture numérique et nous le validerons par une application.

Chapitre 3 Implémentation et Résultats

3.1 INTRODUCTION

Depuis longtemps des travaux de recherche visent à proposer un modèle du cerveau humain. Les progrès théorique et technologique réalisés dans des domaines tels que les sciences cognitives ou les neurosciences mais aussi de l'informatique et de l'électronique numérique permettent de nos jours d'envisager des premières réalisations de composants intégrant des centaines de milliers de neurones. A terme, les circuits numériques pourraient être dotés de capacités intellectuelles « comparables » à celles des êtres humains.

Notre objectif est de suivre relativement le modèle du neurone biologique en utilisant une électronique purement numérique. La conception en numérique permet de réaliser un modèle suffisamment flexible. Ce modèle nous permettra de construire un réseau extensible et évolutif qui servira pour la résolution des problèmes complexe. L'électronique numérique offre aussi une méthodologie de conception structurée par le flot de conception et une description haut niveau (VHDL ou Verilog).

Le but de ce chapitre est de présenter la conception et le fonctionnement d'un neurone artificiel standard d'une manière purement numérique. Ce modèle se présentera comme étant une cellule élémentaire qui servira pour la construction, par la suite, d'un réseau de neurones multicouches plus complexe. Nous décrivons l'unité de contrôle qui gère le fonctionnement du neurone. Ce contrôleur se présente sous la forme d'une machine d'états finis. Nous validerons la construction générale d'un réseau de neurones multicouches par un exemple relativement simple **[4]**.

3.2 CONCEPTION D'UN NEURONE ELEMENTAIRE

3.2.1 Introduction

La fonction réalisée par un neurone artificiel unitaire dans un perceptron multicouche étant une entrée du neurone, est le poids synaptique est exprime par l'équation 2.1 ou $\mathbf{X_i}$ étant une entrée, $\mathbf{W_i}$ est le poids synaptique du neurone relatif à l'entre **i** considérée, ***f*** étant la fonction d'activation du neurone, et **n** est le nombre total d'entrées pour le neurone **[4]**.

$$\boldsymbol{sortie} = \boldsymbol{f}\left(\sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} \boldsymbol{W_i X_i}\right) (2.1)$$

Il existe plusieurs modèles pour la fonction d'activation, mais la plus utilisée est la fonction sigmoïde binaire définie par :

$$\boldsymbol{f(x)} = \frac{\mathbf{1}}{\mathbf{1+\exp(-x)}} (2.2)$$

Ce paragraphe a pour objectif de décrire le fonctionnement d'un neurone artificiel.

3.2.2 Le contrôle de fonctionnement

Le fonctionnement d'un neurone artificiel peut être modélisé à l'aide d'une machine d'états finis qui assure la coordination entre les différentes opérations réalisées. La synchronisation entre les différentes tâches est répartie sur cinq états. Au début, le neurone est au repos (état S0), un signal « start » permet le passage vers l'état suivant(S1), au cours duquel une des entrées du neurone sera multipliée par son poids synaptique approprié. Le troisième état (S2) nous permettra d'acheminer le résultat partiel obtenu vers un accumulateur. Ces deux derniers états se répèteront jusqu'à ce que le nombre des entrées du neurone « nb » soit atteint (état S3).le dernier état (S4) nous permettra la génération d'un signal «fin_calcul » indiquant que le neurone avait terminé sa tâche.

Ce signal sera d'une importance majeure pour les neurones de la couche suivante dans le cas d'un réseau de neurones multicouches. Cette machine d'états finis est décrite par la figure 24 ci dessous.

L'utilité des signaux « reset_sync », « ena_cpt », « ena1 », « ena2 » et « Fin » sera mis en évidence avec le fonctionnement global du neurone artificiel.

Les entrées/sorties de la machine d'états finis seront des entrées/sorties du neurone et/ou des différents composants élémentaires à l'intérieur du même neurone. Cette machine d'états finis jouera le rôle d'un contrôleur à l'intérieur de chaque neurone artificiel. D'où la nécessité d'introduire dés maintenant un signal de reset général noté «reset_gen» caractérisant tout système synchrone. La structure externe de la machine d'états finis est représentée par la figure 24.

Figure 24.Les entrées/sorties de la machine d'états finis.

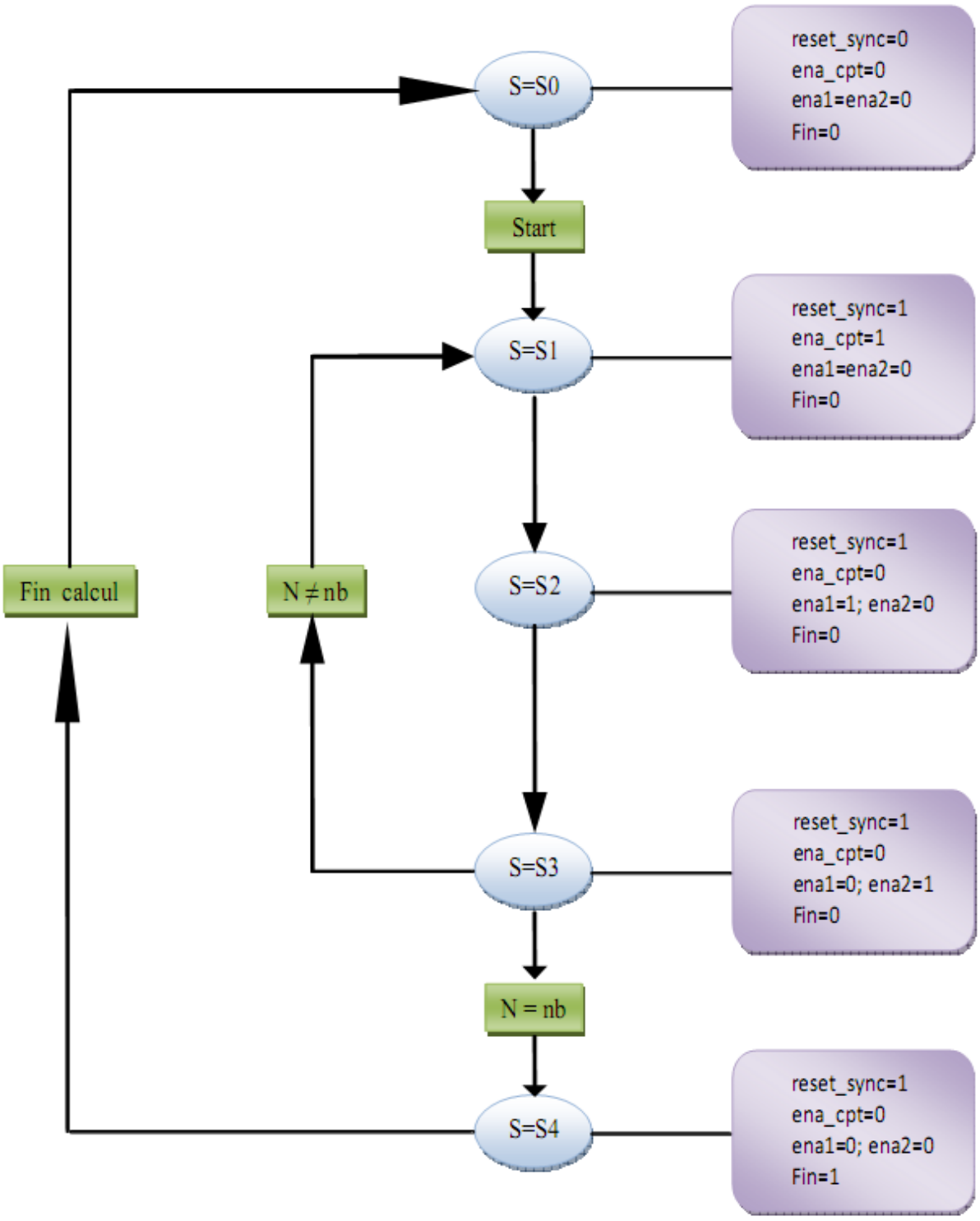


Figure 25.Machine d'états finis modélisant les différentes opérations réalisées par un Neurone.

3.2.3 Architecture d'un neurone élémentaire

Les signaux «A_0» jusqu'à «A_n» représentent les entrées du neurone, «B_0» jusqu'à «B_n» représentent les poids synaptiques, «CLK»étant l'horloge du système et «Start» «B_n» n'est autre que le signal suivant lequel le neurone commence son fonctionnement. Vu de l'extérieur le neurone est schématisé par la figure 26.

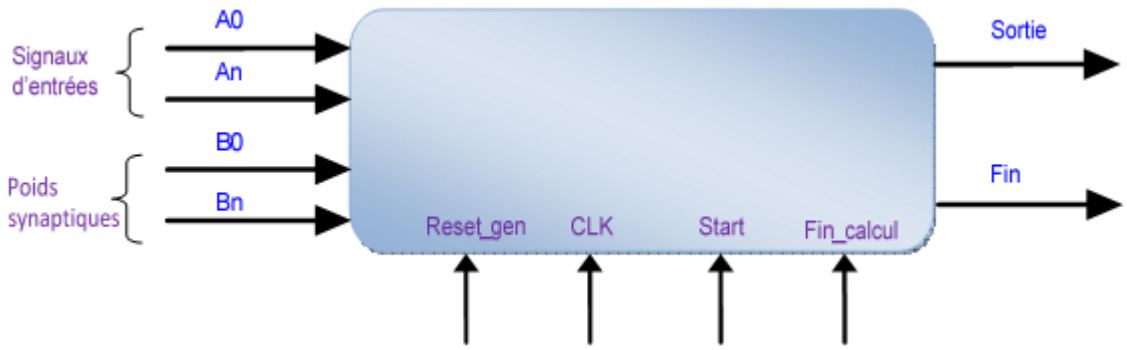


Figure 26.Les entrées/sorties du neurone artificiel numérique proposé.

La multiplication des signaux d'entrées du neurone avec les différents poids synaptiques est assurée par deux multiplexeurs commandés par un compteur qui permet la sélection des entrées appropriées et leurs coefficients. Le résultat de chacune des multiplications sera cumulé avec celle qui a précédé jusqu'à la dernière opération (nb=N). Une fois la dernière multiplication achevée, la machine d'états permet le passage du résultat de l'accumulateur vers le bloc qui gère la fonction d'activation.

En résumé, la conception d'un neurone artificiel numérique nécessite un compteur, deux multiplexeurs, un multiplieur, un additionneur accumulateur, une machine d'états finis et un circuit modélisant la fonction d'activation choisie.

Nous proposons l'architecture générale d'un neurone artificiel dont le schéma blocs et les interconnexions sont données par a figure 27.

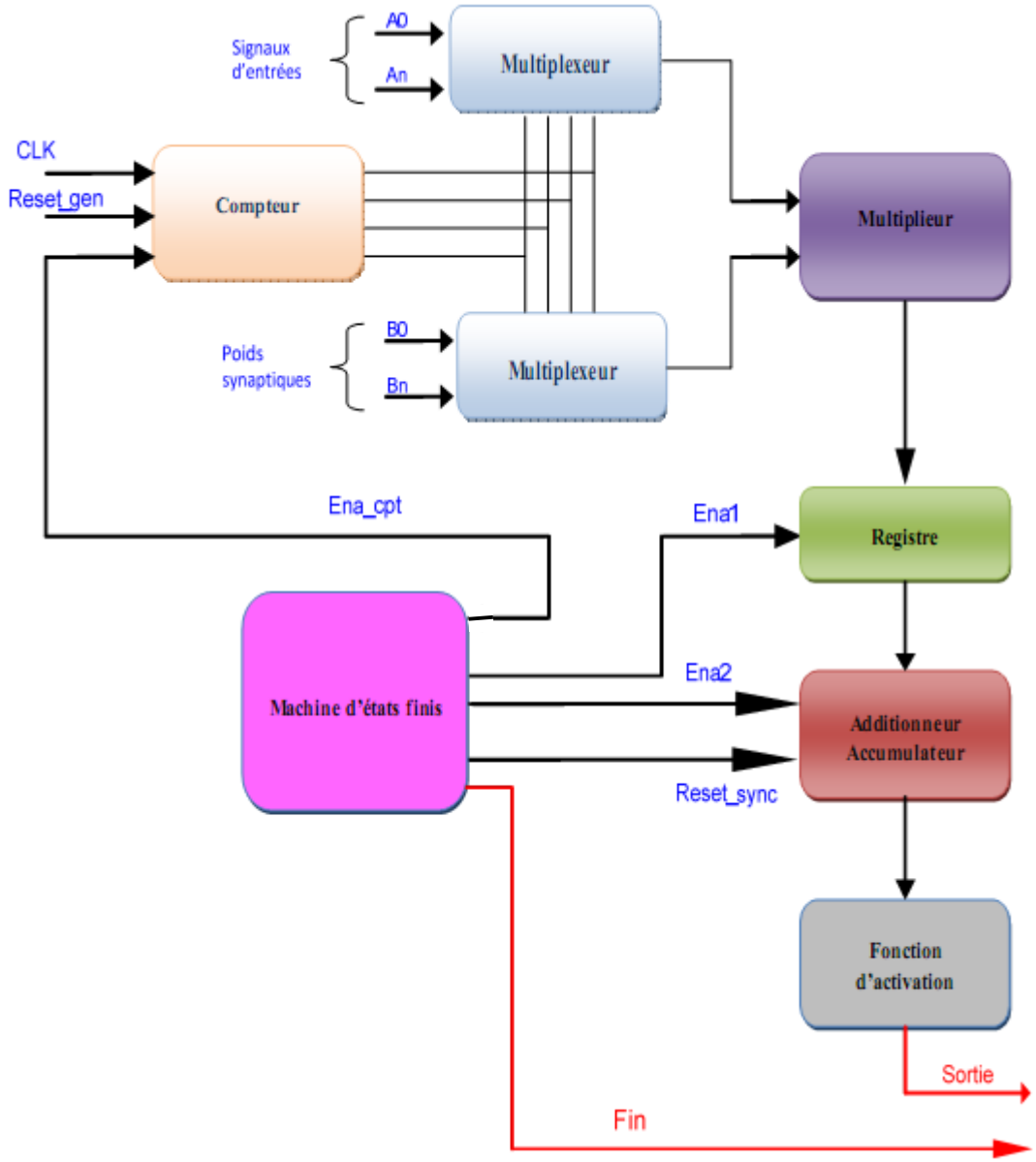


Figure 27.Architecture générale d'un neurone artificiel à N entrées.

a. ***Le compteur***

son role est de compter le nombre d'opération realiser par le neurone.

b. ***Le multiplexeur***

Pour sélectionner chaque entrée avec son propre poids.

c. ***Le multiplieur***

Il sert à faire la multiplication des poids synaptiques avec les valeurs des exemples correspondants qui sont présentés à son entrée. Cette opération est nécessaire pour le calcul de la somme pondérée. Les valeurs calculées dans ce multiplieur sont transmises de sa sortie vers l'entrée de l'accumulateur.

d. ***L'accumulateur***

C'est l'élément qui effectue la somme pondérée des valeurs venant, après le calcul, du multiplieur. L'accumulateur se compose d'un additionneur et d'un registre. L'opération de l'accumulation se fait par le chargement de la première valeur venant du multiplieur dans le registre puis l'additionner avec les données reçues du multiplieur.

e. ***La fonction d'activation***

Ce bloc représente la fonction d'activation sigmoïde, son rôle est de prendre la valeur de la somme pondérée calculée par le neurone et lui appliquer la fonction sigmoïde pour transmettre une valeur d'activation à la sortie du neurone **[9]**.

Le problème de la fonction sigmoïde est qu'elle est difficile à implémenter. Pour remédier à ce problème, trois méthodes sont utilisées (figure 28).

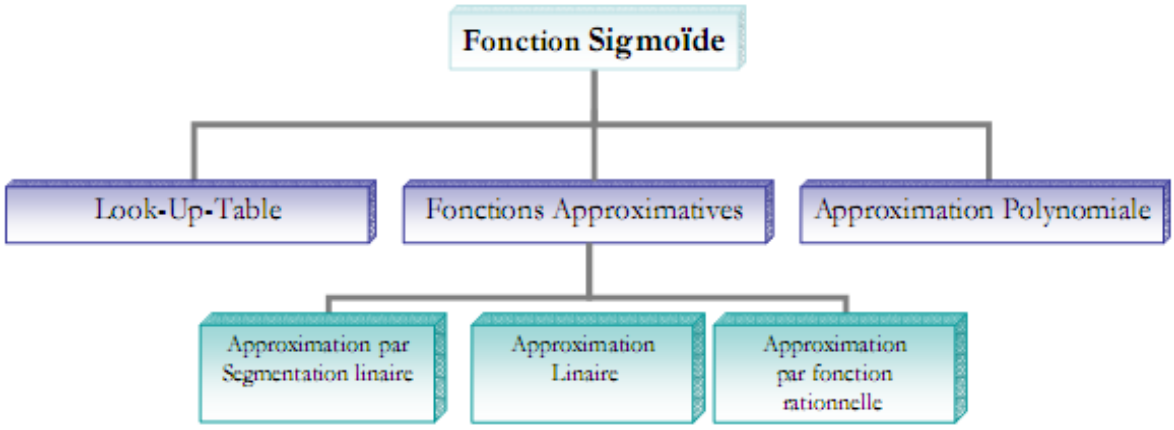


Figure28 .différentes représentations de la fonction d'activation.

Pour notre cas on a utilisé l'approximation par segmentation linéaire, une autre alternative consiste à utiliser une approximation linéaire par morceaux de la fonction sigmoïde, dans ce cas il s'agit de diviser la fonction sigmoïde en plusieurs fonction sigmoïde segments, avec X compris dans l'intervalle [-5, 5] et Y dans l'intervalle [0, 1].

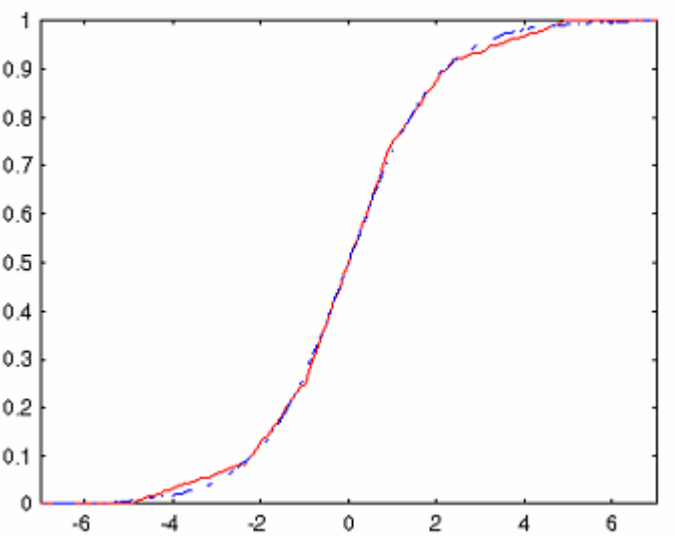


Figure 29.L'approximation par segmentation linéaire de la fonction sigmoïde.

Le tableau suivant va expliquer mieux cette approximation :

X appartient a :	**La fonction de sortie Y**
[0, 0.99]	Y=0 .25X+0.5
[1, 2.3749]	Y=0.125X+0.625
[2.375, 4.9]	Y=0.03125X+0.84375
5	Y=1
[-1, 00]	Y=0.23105X+0.49653
[-2, -1]	Y=0.14973X+0.41285
[-3, -2]	Y=0.07177X+0.25902
[-4, -3]	Y=0.02943X+0.13404
[-5, -4]	Y=0.01129X+0.06248
-5	Y=0

Tableau 4.Table d'approximation de la fonction sigmoïde [3].

3.3 ARCHITECTURE D'UN RESEAU MULTICOUCHE

Ils existent plusieurs architectures des réseaux de neurones, mais nous s'intéressons à l'architecture du perceptron multicouche **[9]**.

3.3.1 Introduction

Le mécanisme perceptron fut inventé par le psychologue Frank ROSENBLAT à la findes années 50. Il représentait sa tentative d'illustrer certaines propriétés fondamentales des systèmes intelligents en général.

3.3.2 Principe du perceptron multicouche

Le perceptron multicouche est un réseau comportant L couches, chaque neurone d'une couche étant totalement connecté aux neurones de la couche suivante.

Un réseau MLP comporte une ou plusieurs couches cachées à fonction d'activation de type sigmoïde ainsi qu'une couche de sortie. Les neurones de la couche cachée « C » reçoivent des informations des neurones de la couche « C−1 » et sont connectées aux neurones de la couche « C+ 1 ». Il n'existe pas de connexions entre les neurones d'une même couche.Chaque neurone de la couche de sortie réalise une fonction non linéaire des entrées du réseau.

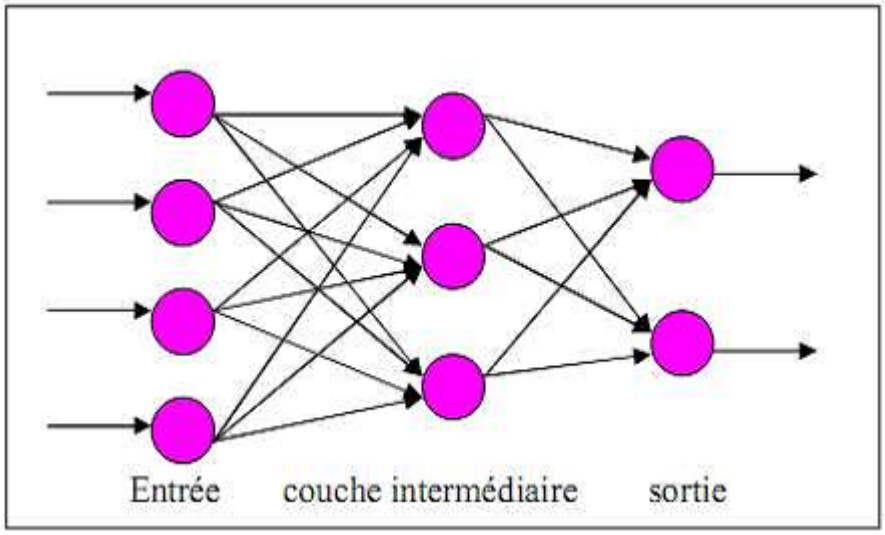


Figure30.Exemple d'un perceptron multicouche.

3.3.3 Conception de Perceptron multicouche

Après avoir présenté le fonctionnement d'un neurone unitaire, il est maintenant possible de voir l'emplacement et la coordination de ce neurone avec d'autres qui sont similaires [16].

L'ensemble forme une architecture de perceptron multicouche. Vu que la transmission des données se fait d'une couche à une autre qui la suit, le signal « Start » pour les neurones d'une couche, autre que la première, sera reliée au signal « Fin» des neurones de la couche précédente.

Cela permet le déclenchement du fonctionnement de ceux ci juste après que la précédente ait fourni son résultat. Une porte « Et » permet d'assurer le déclenchement simultané de tous les neurones de la couche qui suit.

Une fois les neurones de la dernière couche ont fournis leurs résultats, le signal « Fin » de chacun des neurones de la dernière couche sera relié aux entrées « Fin_calcul » de tous lesneurones des couches précédentes permettant le retour de tous les neurones du réseau vers un état initial S0 de la machine d'états finis.

Le principe d'une architecture d'un perceptron multicouche est décrit par la figure 31 Il est important de signaler que cette architecture est très générale. Bien que le principe reste lemême, théoriquement, la manipulation d'un seul neurone et quelques multiplexeurs permet la réalisation électronique de tout réseau au dépend de la fréquence souhaitée et de l'application traitée.

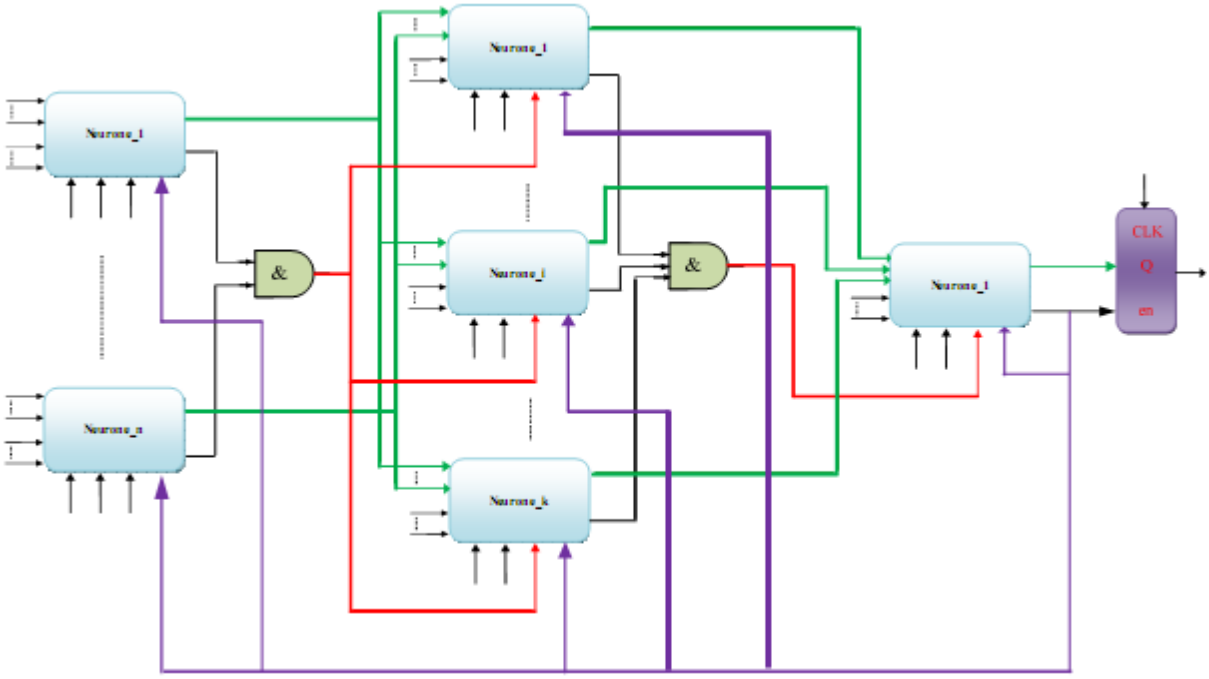


Figure 31.Principe d'architecture générale d'un perceptron multicouche.

3.4 EXEMPLE DE RESEAU STATIQUE

L'implantation électronique des réseaux de neurones statiques est beaucoup plus facile que celle des réseaux dynamiques. En effet, l'apprentissage pour ces réseaux, se fait par logiciel pendant les simulations théoriques. La matrice des coefficients fixes et adéquats à la réalisation de l'application sera directement implantée d'une manière câblée et définitive évitant ainsi l'utilisation d'une mémoire. L'implantation de l'algorithme d'apprentissage au sein du circuit, dans ce cas, n'a pas de sens.

Dans ce qui suit, nous proposons de réaliser un exemple de circuit statique qui réalise la classification de plusieurs types de d'objet en deux catégories.

On peut implémenter cet application de clasification de réseaux de neurones proposés, sur un robot mobile équipé d'une caméra embarquée et capteursce système acquiert les données et les caractéristiques des images fournies par la caméra et les capteurs.

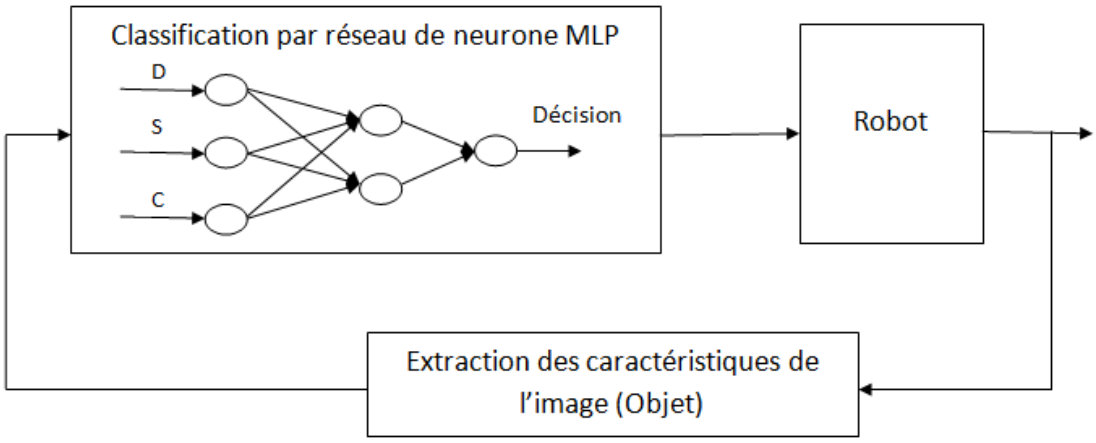
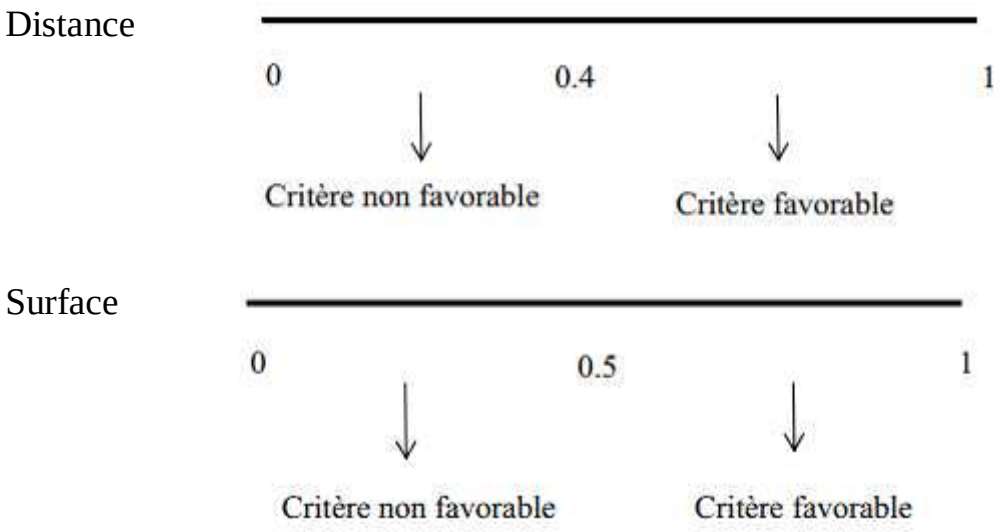


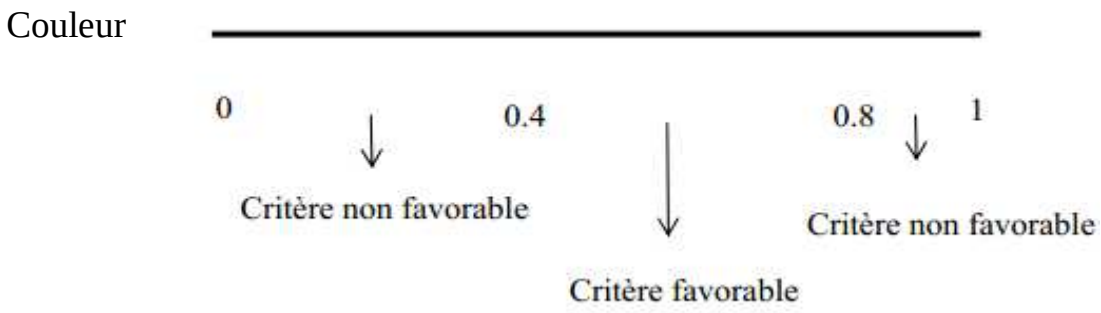
Figure 32.Shéma synoptique de visuel neuronal.

3.4.1 Présentation de l'exemple

Le problème réside dans la séparation d'une quantité d'un objet en deux familles suivant la qualité. On notera que «X» une unité de ce produit. La séparation s'effectue selon 3 critères: ladistance, la sourface et la couleur. Deux parmi ces trois critères sont des facteurs déterminants dans la sélection : la surface et la distance

Imaginons qu'on fait une normalisation de ces trois critères. C'est à dire que chaque critère peut être modélisé par un nombre continu qui appartient à [0,1]. Dans ce cadre, chaque «X» est symbolisée par trois nombres continus appartenant chacun à [0,1].





La séparation se fait en deux classes, la 1ère englobe toute «X» ayant un critère favorable pour la distance et la sourface, et non nécessairement un critère favorable pour la couleur. La 2ème classe contient toute «X» qui ne satisfait pas aux suggestions de la 1ère classe.

La classification se fait suivant trois critères, notre réseau possède donc trois entrées. Pour distinguer deux classes, il suffit de prendre un seul neurone de sortie. On a choisi de prendre une seule couche cachée qui contient deux neurones, une pour tenir compte du facteur déterminant, l'autre tient compte d'une idée de prétraitement qu'on appelle le taux de qualité.

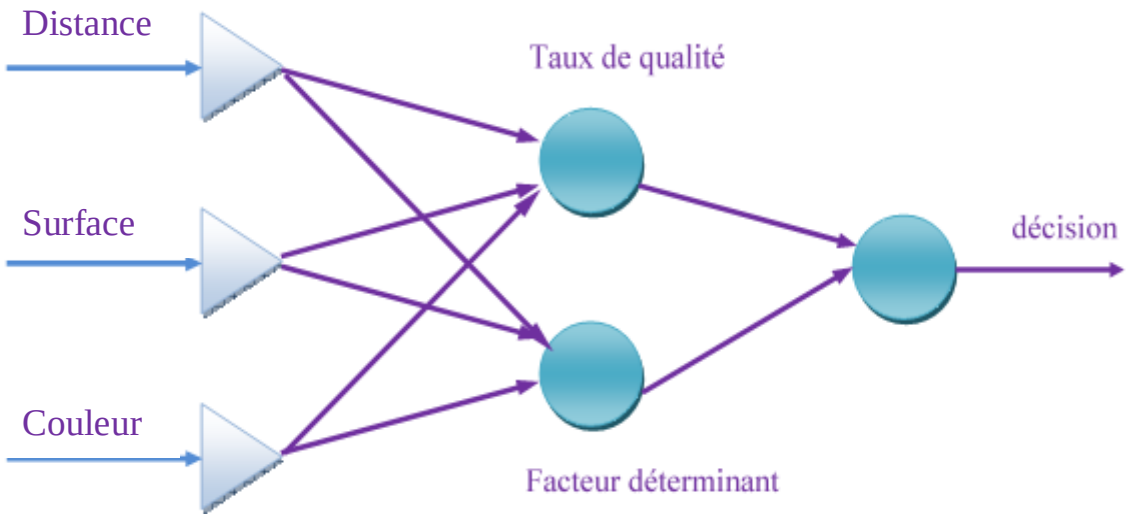


Figure 33.Architecture du réseau de neurones pour l'application de classification.

- **Taux de qualité:** on définit le taux de qualité comme étant le nombre de critères favorables divisés par 3 (on a 3 critères).
- **Facteur déterminant :** on veut que cette cellule soit à 1 si la distance et la surface sont conformes aux normes, si non elle est à 0.

- **Cellule de décision :** cette cellule s'active uniquement pour une «X» de 1ère classe.

Facteur déterminant	Taux de qualité		
	0.33	**0.66**	**1**
0	**2éme classe**	**2éme classe**	**-**
1	**-**	**1ére classe**	**1ére classe**

Tableau 4 .les paramètres de classification.

3.4.2 Apprentissage sous « MATLAB »

Notre idée de conception repose sur le fait qu'on impose des valeurs de sortie pour chaque prototype donné. L'apprentissage est donc supervisé. Le but de l'apprentissage est de trouver deux matrices W1 et W2, relatives à la couche 1 et à la couche 2, des connexions qui seraient valables pour classer n'importe quel produit « X » présenté à l'entrée du réseau [9].

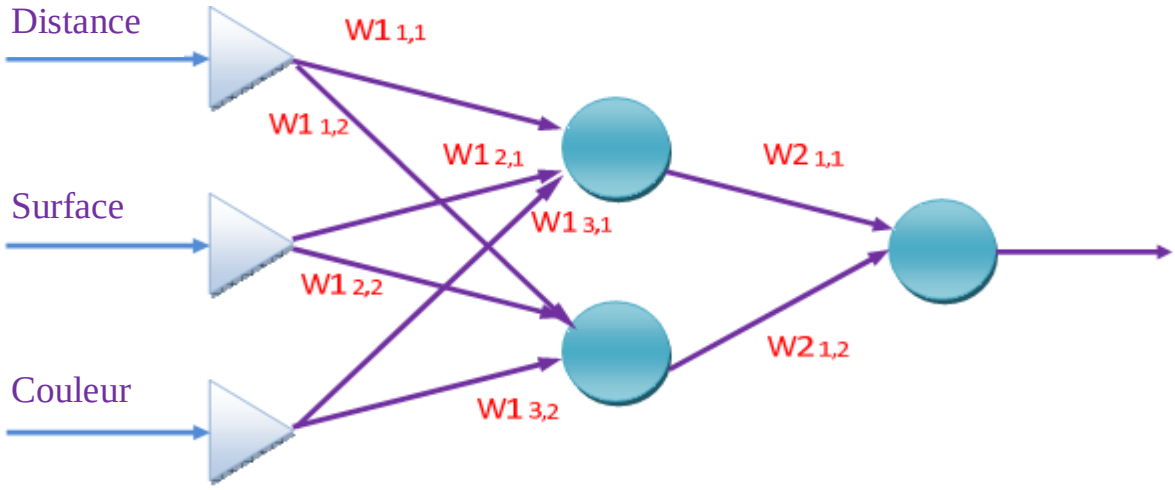


Figure 34.Les matrices d'interconnexions entre les neurones du réseau.

Les sorties désirées de la couche cachée sont reparties dans l'intervalle [0,1]. Dans ce cas une fonction d'activation discrète ne serait pas valable. On choisit alors la fonction sigmoïde binaire définie par l'équation :

$$F(p) = 1/(1+\exp(-p))$$

Le neurone de sortie est une cellule de décision, l'idée qui convient est de lui associer une fonction discrète. On adopte la fonction suivante :

$$F(p) = \begin{cases} 1 \text{ si } p \geq 1 \\ 0 \text{ si } p < 0 \end{cases}$$

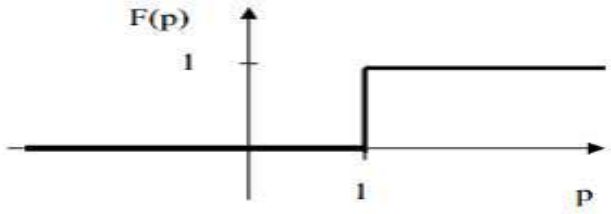


Figure 35 .Fonction d'activation du neurone de sortie.

Le but de la simulation est de trouver deux matrices W1 et W2 respectivement pour les couches 2 et 3 qui seront utiles pour classifier n'importe quel produit «X». Après apprentissage les deux matrices trouvées sont :

$$W1=\begin{pmatrix} 0.3204 & 0.6139 \\ 1.5239 & 2.7438 \\ -0.4720 & -3.3546 \end{pmatrix}, W2 = \begin{pmatrix} 1.6835 \\ -0.3166 \end{pmatrix}$$

3.5 IMPLANTATION DE L'APPLICATION SUR UN CIRCUIT

3.5.1 Architecture d'un neurone à 2 entrées

Nous avons déjà présenté la structure générale d'un neurone à « nb » entrées. Pour le cas particulier où le neurone a trois entrées, le comptage sera assuré par un compteur modulo 2. Les opérandes de la multiplication seront issus des multiplexeurs 2 vers 1.L'ensemble est contrôlé par une machine d'états finis où le paramètre N est égal à 2.

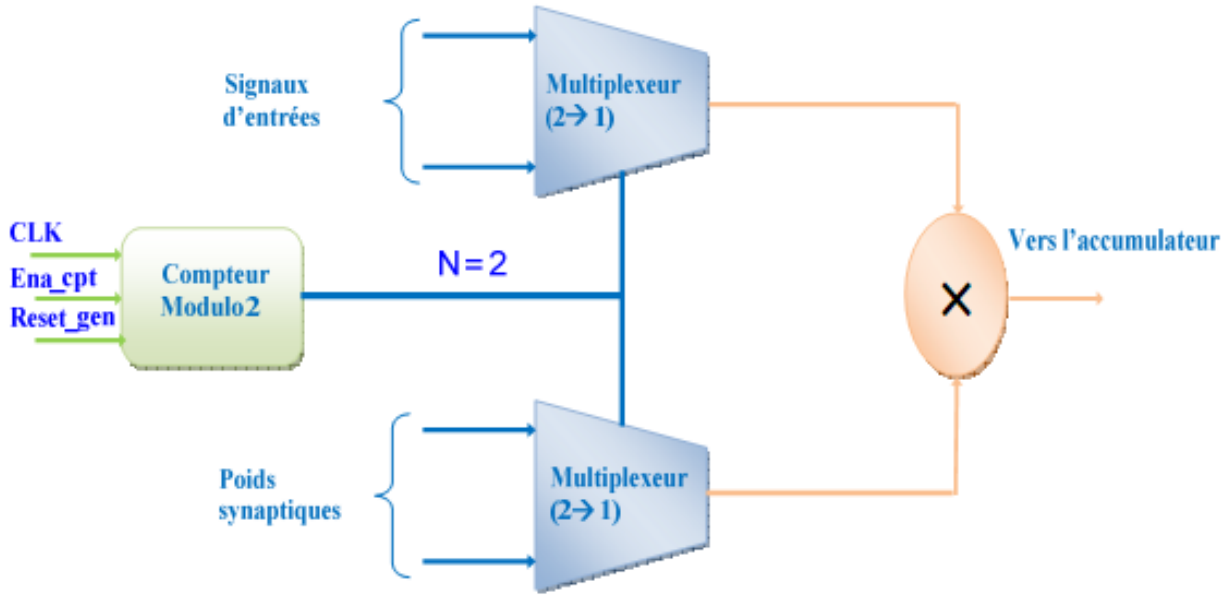


Figure 36.Modules de multiplication d'un neurone à deux entrées.

3.5.2 Architecture d'un neurone à 3 entrées

Pour le neurone à trois entrées, on a besoin un compteur modulo 3. Les multiplexeurs garantissent le passage des données en alternance jusqu'au multiplieur. Comme pour le cas précèdent, la machine d'états finis contrôle le bon déroulement des différents événements.

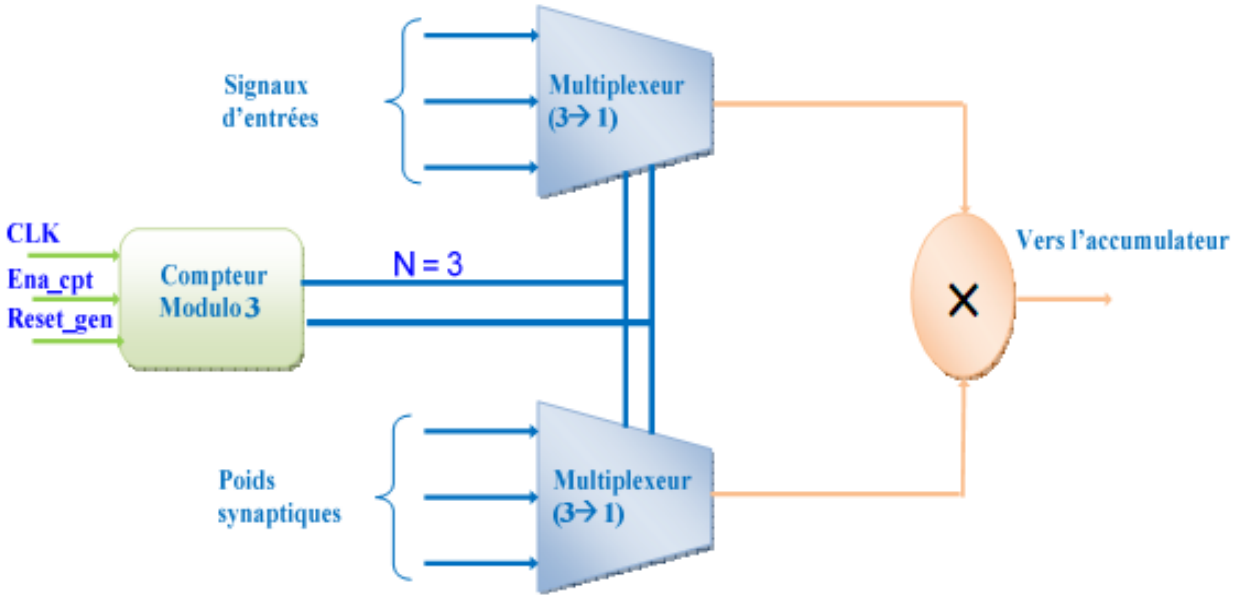


Figure 37.Modules de multiplication d'un neurone à trois entrées.

3.5.3 Architecture du réseau complet

À la fin de la simulation mathématique de l'étape d'apprentissage, les coefficients synaptiques sont alors définitifs, c'est-à-dire qu'on peut les considérer comme des constantes.

Dans la structure qu'on propose, les entrées « coefficients » sont électriquement câblées à l'intérieur. Cela nous permet de gagner la surface nécessaire à l'implantation d'une mémoire dans laquelle les coefficients peuvent être stockés.

Entre les deux couches du réseau, une porte «Et » à deux entrées guide le déroulement des calculs couche par couche en contrôlant le signal « Start » du neurone de la couche de sortie.L'implantation d'un registre à la sortie a été imposée par le fait qu'on peut perdre l'information à cause du signal « Fin_calcul ».

Enfin, pour avoir le résultat final sur un seul bit, un circuit de décision est placé à la sortie, il réalise une comparaison entre l'activation finale et un seuil bien déterminé.

3.6 LES PROGRAMMES VHDL

3.6.1 Compteur

```
1  -------------compteur modulo3 ------------------------------------------
2  ------------------------------------------------------------------------
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.std_logic_arith.all;
6  use IEEE.STD_LOGIC_UNSIGNED.ALL;
7  entity compt_3 is
8     port(
9        clk,ena,preset :in std_logic;
10       q              :out std_logic_vector(1 downto 0));
11 end compt_3;
12
13
14 architecture arch_compt_3 of compt_3 is
15
16    signal count : std_logic_vector(1 downto 0):="00";
17    constant un  : std_logic_vector(1 downto 0):="01";
18
19
20 begin -- arch_compt_3
21
22   comptage:process(clk,preset,ena)
23    begin --process
24    if clk'event and clk='1' then
25      if preset ='1' then
26         if ena='1' then
27             if count="10" then
28                 count<="00";
29             else
30                count<=count + un;
31             end if;
32         end if;
33     else
34         count<="10" ;
35      end if;
36    end if;
37   end process;
38   q<=count;
39
40 end arch_compt_3;
```

3.6.2 Multiplieur

```
-- multiplieur ------------------------------------------
----------------------------------------------------------
library IEEE;
library IEEE_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE_proposed.fixed_pkg.all;
use IEEE_proposed.fixed_float_types.all;
----------------------------------------------------------
--  entity
----------------------------------------------------------
entity Multiplier is

  port (
    X, Y : in  sfixed (4 downto -14);
    P    : out  sfixed (4 downto -14));

end Multiplier;

architecture arch_mult of Multiplier is
    signal z:sfixed(4 downto -14);
begin
     z<=resize(x*y,z);
     p<=z;
end arch_mult;
```

3.6.3 Additionneur

```
----------------------additionneur-----------------------------------------
-----------------------------------------------------------------------------
library IEEE;
library IEEE_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE_proposed.fixed_pkg.all;
use IEEE_proposed.fixed_float_types.all;
-----------------------------------------------------------------------------
--  entity
-----------------------------------------------------------------------------
entity add is

  port (
    X : in  sfixed (4 downto -14);
    y : in  sfixed (4 downto -14);
    P : out  sfixed (4 downto -14));

end add;

architecture arch_add of add is
  signal s:sfixed(4 downto -14);
begin

        s<=resize(x+y,s);
        p<=s;

end arch_add;

```

3.6.4 Multiplexeur des poids

```
-- multiplexeur  3 vers 1    ***W_1  --------- les entrees sont a 19 bits----------
--------------------------------------------------------------------------------
library IEEE;
library IEEE_proposed;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE_proposed.fixed_pkg.all;
use IEEE_proposed.fixed_float_types.all;

--------------------------------------------------------------------------------
--  entity
--------------------------------------------------------------------------------
entity mux_3_w1_2 is

  port (

    sel3       : in  std_logic_vector(1 downto 0);
    sortie     : out sfixed (4 downto -14));

end mux_3_w1_2;

--------------------------------------------------------------------------------
--  architecture
--------------------------------------------------------------------------------
architecture arch_mux_3_w1_2 of mux_3_w1_2 is

constant autres : sfixed (4 downto -14):= "UUUUUUUUUUUUUUUUUUU";
 signal a0, a1 ,a2    :  sfixed (4 downto -14);
begin  -- arch_mux8b3_1

  a0<=to_sfixed(0.6139,a0);
  a1<=to_sfixed(2.7438,a1);
  a2<=to_sfixed(-3.3546,a2);
  process (sel3)
  begin  -- process
    case sel3 is
      when     "00"   =>     sortie<=a0;
      when     "01"   =>     sortie<=a1;
      when     "10"   =>     sortie<=a2;
      when     others =>     sortie<= autres;

    end case;
  end process;

end arch_mux_3_w1_2;
```

3.6.5 Machine d'état

```
-- machine d'etats finis pour un neurone 3 entrees----------------------------
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

--------------------------------------------------------------------------------
--  entity
--------------------------------------------------------------------------------
entity machine_3 is
 port (
   clk,start,fin_calcul,reset_gen              :in std_logic;
   N                                           :in std_logic_vector(1 downto 0);
   ena_cpt, ena1,ena2,reset_sync               :out std_logic;
   fin                                         :buffer std_logic);
end machine_3;


--------------------------------------------------------------------------------
--  architecture
--------------------------------------------------------------------------------
architecture arch_mach3 of machine_3 is
  type state_machine is (s0,s1,s2,s3,s4);
  signal state :state_machine;
begin  -- arch_fsm3:
  process (clk,start,N,fin_calcul)
  begin
   if reset_gen='0' then state<=s0;
   elsif clk'event and clk='1' then
     case state is
       when  s0=>
             if start= '1' then
               state<= s1;
             else
               state<= s0;
             end if;

        when s1=>
               state<= s2;

       when s2=>
               state<= s3;

       when s3=>
               if N="10"then
               state<=s4;
             else
               state<=s1;
             end if;
       when s4=>
               if fin_calcul='1' then
                 state<= s0;
               else
                 state<=s4;
               end if;
    when others =>
       state<=s0;
     end case;

   end if;
  end process;

  reset_sync <= '0' when state = s0 else  '1';
  ena_cpt <= '1'    when state = s1 else  '0';
  ena1 <= '1'       when state = s2 else  '0';
  ena2 <= '1'       when state = s3 else  '0';
  fin <= '1'        when state = s4 else  '0';

end arch_mach3;
```

3.6.6 Registre

```
library IEEE;
library IEEE_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE_proposed.fixed_pkg.all;
use IEEE_proposed.fixed_float_types.all;

--------------------------------------------------------------------------------
--  entity
--------------------------------------------------------------------------------
entity registre_1 is

  port (
    clk,ena : in std_logic;
     D: in sfixed (4 downto -14);
     Q: out sfixed (4 downto -14));
end registre_1;

--------------------------------------------------------------------------------
--  architecture
--------------------------------------------------------------------------------
architecture arch_registre_1 of registre_1 is

signal Q_int :sfixed (4 downto -14):="00000000000000000000";

begin  -- arch_registre

process (clk)
begin  -- process
   if clk'event and clk='1' then
   if ena='1' then
     Q_int<=D;
   else
     Q_int<=Q_int;
   end if;

  end if;

end process;

   Q<= Q_int;

end arch_registre_1;
```

3.6.7 Porte AND

```
---------------------- porte AND2 ----------------------------------------------
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity porteand is

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

end porteand;

architecture arch_porteand of porteand is

begin  -- arch_porteand

  c<= a and b;

end arch_porteand;
```

3.6.8 Accumulateur

```
-- additionneur accumulateur 19 bits
--------------------------------------------------------------------------------
library IEEE;
library IEEE_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE_proposed.fixed_pkg.all;
use IEEE_proposed.fixed_float_types.all;
--------------------------------------------------------------------------------
--  entity
--------------------------------------------------------------------------------
entity addacc is

  port (
    clk, ena , reset  : in  std_logic;
    P                 : in  sfixed (4 downto -14);
    R               : buffer sfixed(4 downto -14));

end addacc;

--------------------------------------------------------------------------------
--  architecture
--------------------------------------------------------------------------------
architecture arch_addacc of addacc is

  component registre
  port (
    clk,ena,reset : in std_logic;
     D: in sfixed (4 downto -14);
     Q: out sfixed (4 downto -14));
  end component;

  component add
  port (
    X : in  sfixed (4 downto -14);
    Y : in  sfixed (4 downto -14);
    P : out  sfixed (4 downto -14));
  end component;

   signal Qint : sfixed(4 downto -14);
   signal Rint : sfixed(4 downto -14):="00000000000000000000";

begin  -- arch_addacc

    c0: add port map (P,Rint,Qint);
    c1: registre port map (clk,ena,reset,Qint,rint);
    R <= rint;

end arch_addacc;
```

3.6.9 Fonction sigmoïde

```
1
2   ---------------Fonction sigmoide------------
3   ----------------------------------------------
4   library IEEE;
5   library IEEE_proposed;
6   use IEEE.STD_LOGIC_1164.ALL;
7   use IEEE_proposed.fixed_pkg.all;
8   use IEEE_proposed.fixed_float_types.all;
9
10
11  entity sigmoide is
12      Port ( x : in  sfixed (4 downto -14);
13             Z : out  sfixed (4 downto -14));
14  end sigmoide;
15
16  architecture Behavioral of sigmoide is
17
18
19     ---LES INTERVALLES positive
20    signal int_1 : sfixed (4 downto -6);
21    signal int_11 : sfixed (4 downto -6);
22    signal int_2 : sfixed (4 downto -6);
23    signal int_21 : sfixed (4 downto -6);
24    signal int_3 : sfixed (4 downto -6);
25    signal int_31 : sfixed (4 downto -6);
26    signal int_4 : sfixed (4 downto -6);
27    signal int_5 : sfixed (4 downto -6);
28
29    ---les poly
30    signal coef_a1 : sfixed (4 downto -14);
31    signal coef_b1 : sfixed (4 downto -14);
32    signal coef_a2 : sfixed (4 downto -14);
33    signal coef_b2 : sfixed (4 downto -14);
34    signal coef_a3 : sfixed (4 downto -14);
35    signal coef_b3 : sfixed (4 downto -14);
36    ---les signal
37
38     signal y         :   sfixed (4 downto -14);
39
40    ---LES INTERVALLES negative
41    signal int_n_1 : sfixed (4 downto -6);
42    signal int_n_2 : sfixed (4 downto -6);
43    signal int_n_3 : sfixed (4 downto -6);
44    signal int_n_4 : sfixed (4 downto -6);
45    signal int_n_5 : sfixed (4 downto -6);
46     ---les coef_negative
47    signal coef_n_a1 : sfixed (4 downto -14);
48    signal coef_n_b1 : sfixed (4 downto -14);
49    signal coef_n_a2 : sfixed (4 downto -14);
50    signal coef_n_b2 : sfixed (4  downto -14);
51    signal coef_n_a3 : sfixed (4 downto -14);
52    signal coef_n_b3 : sfixed (4 downto -14);
53    signal coef_n_a4 : sfixed (4 downto -14);
54    signal coef_n_b4 : sfixed (4 downto -14);
55    signal coef_n_a5 : sfixed (4 downto -14);
56    signal coef_n_b5 : sfixed (4 downto -14);
57
58    begin
59      ---LES INTERVALLES pos
60          int_1 <= to_sfixed (0.00, 4, -6);
61          int_11 <= to_sfixed (0.99, 4, -6);
62          int_2 <= to_sfixed (1.00, 4, -6);
63          int_21 <= to_sfixed (2.3749,4, -6);
64          int_3 <= to_sfixed (2.375, 4, -6);
65          int_31 <= to_sfixed (4.90, 4, -6);
66          int_4 <= to_sfixed (5.00, 4, -6);
67          int_5 <= to_sfixed (-5.00, 4, -6);
```

```
68    --- Les intervalles neg
69         int_n_1 <= to_sfixed (-1.0, 4, -6);
70         int_n_2 <= to_sfixed (-2.0, 4, -6);
71         int_n_3 <= to_sfixed (-3.0, 4, -6);
72         int_n_4 <= to_sfixed (-4.0, 4, -6);
73         int_n_5 <= to_sfixed (-5.0, 4, -6);
74    ---les poly POS
75         --INT_1
76         coef_a1 <= to_sfixed (0.25,4,-14);
77         coef_b1 <= to_sfixed (0.50,4,-14);
78         --INT_2
79         coef_a2 <= to_sfixed (0.125,4,-14);
80         coef_b2 <= to_sfixed (0.625,4,-14);
81         --INT_3
82         coef_a3 <= to_sfixed (0.03125,4,-14);
83         coef_b3<= to_sfixed (0.84375,4,-14);
84
85         ---les poly_neg
86         --INT_1
87         coef_n_a1 <= to_sfixed (0.23105,4,-14);
88         coef_n_b1 <= to_sfixed (0.49653,4,-14);
89         --INT_2
90         coef_n_a2 <= to_sfixed (0.14973,4,-14);
91         coef_n_b2 <= to_sfixed (0.41285,4,-14);
92         --INT_3
93         coef_n_a3 <= to_sfixed (0.07177,4,-14);
94         coef_n_b3 <= to_sfixed (0.25902,4,-14);
95         --INT 4
96         coef_n_a4 <= to_sfixed (0.02943,4,-14);
97         coef_n_b4 <= to_sfixed (0.13404,4,-14);
98         --INT_5
99         coef_n_a5 <= to_sfixed (0.01129,4,-14);
100        coef_n_b5 <= to_sfixed (0.06248,4,-14);
101
102   process(x)
103    begin
104      ---intervalle pos
105      --intervalle_1 :[0,0.99 ]
106      if ((x >= int_1) and (x <= int_11)) then
107            y<= resize(( x * coef_a1) + coef_b1,y);
108
109      --intervalle_2:[1,2.3749 ]
110      elsif((x >= int_2) and (x <= int_21)) then
111            y<= resize(( x * coef_a2) + coef_b2,y);
112
113       --intervalle_3:[2.375,4.9 ]
114      elsif((x >= int_3) and (x <= int_31)) then
115            y<= resize(( x * coef_a3) + coef_b3,y);
116
117       --intervalle_4:[5,+00 ]
118      elsif(x >= int_4)  then
119           y<= to_sfixed(1.0,4,-14);
120
121       ---intervalle neg
122
123      --intervalle 1:
123      --intervalle_1:
124      elsif ((x >= int_n_1) and (x < int_1)) then
125            y<= resize(( x * coef_n_a1) + coef_n_b1,y);
126
127      --intervalle_2:
128      elsif((x >= int_n_2) and (x < int_n_1)) then
129            y<= resize(( x * coef_n_a2) + coef_n_b2,y);
130
131      --intervalle_3:
132      elsif((x >= int_n_3) and (x < int_n_2)) then
133            y<= resize(( x * coef_n_a3) + coef_n_b3,y);
134      --intervalle_4:
135      elsif((x >= int_n_4) and (x < int_n_3)) then
136            y<= resize(( x * coef_n_a4) + coef_n_b5,y);
```

```
137       --intervalle_5:
138       elsif((x >= int_n_5) and (x < int_n_4)) then
139             y<= resize(( x * coef_n_a5) + coef_n_b5,y);
140
141       --intervalle_6:[-00,-5]
142       elsif(x < int_n_5)  then
143            y<= to_sfixed(0.0,4,-14);
144
145       else y<="UUUUUUUUUUUUUUUUUUU";
146
147       end if;
148
149     end process;
150
151        z<=Y;
152
153  end Behavioral;
```

3.6.10 Neurone 3_1

```
1   -- neurone 3 entrees ------------- chaque entree est codee sur 19 bits---------
2   ----------------------------------------------------------------------------------
3
4   library IEEE;
5   library IEEE_proposed;
6   use IEEE.STD_LOGIC_1164.ALL;
7   use ieee.std_logic_arith.all;
8   use IEEE_proposed.fixed_pkg.all;
9   use IEEE_proposed.fixed_float_types.all;
10
11  ----------------------------------------------------------------------------------
12  --  entity
13  ----------------------------------------------------------------------------------
14  entity neurone3 is
15
16    port (
17      a0, a1,a2                              : in  sfixed(4 downto -14);
18      clk,start, fin_calcul,reset_gen        : in  std_logic;
19      result                                 : out sfixed(4 downto -14);
20      fin                                    : buffer std_logic);
21
22  end neurone3;
23
24  ----------------------------------------------------------------------------------
25  --  architecture
26  ----------------------------------------------------------------------------------
27  architecture arch_neurone3 of neurone3 is
28
29       clk,ena,preset : in std_logic;
30       q      : out std_logic_vector(1 downto 0));
31    end component;
32
33    component mux_3_1
34
35    port (
36      a0, a1 ,a2    : in  sfixed (4 downto -14);
37      sel3       : in  std_logic_vector(1 downto 0);
38      sortie     : out sfixed (4 downto -14));
39
40    end component;
41    component mux_3_w1_2
42
43    port (
44
45      sel3       : in  std_logic_vector(1 downto 0);
46      sortie     : out sfixed (4 downto -14));
47
```

```
48  end component;
49
50    component Multiplier
51
52    port (
53      X, Y : in  sfixed (4 downto -14);
54      P    : out  sfixed (4 downto -14));
55
56    end component;
57
58    component registre_1
59
60    port (
61      clk,ena : in std_logic;
62       D: in sfixed (4 downto -14);
63       Q: out sfixed (4 downto -14));
64    end component;
65
66    component addacc
67
68    port (
69      clk, ena , reset  : in  std_logic;
70      P                 : in  sfixed (4 downto -14);
71      R               : buffer sfixed(4 downto -14));
72
73    end component;
74
75    component machine_3
76     port (
77     clk,start,fin_calcul,reset_gen              :in std_logic;
78     N                                           :in std_logic_vector(1 downto 0);
79     ena_cpt, ena1,ena2,reset_sync               :out std_logic;
80     fin                                         :BUFFER std_logic);
81     end component ;
82
83     component sigmoide
84      Port ( x : in  sfixed (4 downto -14);
85             Z : out  sfixed (4 downto -14));
86     end component;
87     component decision
88
89       port (
90         D  : in  sfixed(4 downto -14);
91         Q : out std_logic);
92
93     end component ;
94
95     signal N                            : std_logic_vector(1 downto 0);
96     signal int1,int2,int3,int4,int5,int6: sfixed (4 downto -14);
97     signal ena_cpt,ena1,ena2,reset_sync : std_logic;
98
99  begin  -- arch_neurone3
100
101    c0: machine_3 port map(clk,start,fin_calcul,reset_gen,N,ena_cpt,ena1,ena2,reset_sync,fin);
102    c1: compt_3  port map (clk,ena_cpt,reset_gen,N);
103    c2: mux_3_1 port map (a0,a1,a2,N,int1);
104    c3: mux_3_w1_2  port map(N,int2);
105    c4: Multiplier port map (int1,int2,int3);
106    c5: registre_1 port map (clk,ena1,int3,int4);
107    c6: addacc port map (clk,ena2,reset_sync,int4,int5);
108    c8: sigmoide port map (int5,result);
109
110 end arch_neurone3;
```

3.6.11 Réseau 3_2_1

```
-- reseau a 2 couches avec 2 neurones a la premiere et une neurone a la seconde
-- les neurones de la premiere couche sont a trois entrees----------------------
--------------------------------------------------------------------------------
library IEEE;
library IEEE_proposed;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use IEEE_proposed.fixed_pkg.all;
use IEEE_proposed.fixed_float_types.all;

--------------------------------------------------------------------------------
--  entity
--------------------------------------------------------------------------------
entity reseau3_2_1 is

  port (
    a0, a1, a2                  : in  sfixed(4 downto -14);

    clk, start, reset_gen       : in  std_logic;
    res                         : out std_logic;
    sortie                      : out sfixed(4 downto -14));

end reseau3_2_1;

--------------------------------------------------------------------------------
--  architecture
--------------------------------------------------------------------------------
architecture arch_reseau3_2_1 of reseau3_2_1 is

  component neurone2
   port (
    a0, a1                              : in  sfixed(4 downto -14);

    clk,start, fin_calcul,reset_gen     : in  std_logic;
    result                              : out std_logic;
    fin                                 : buffer std_logic);
  end component;
  component neurone3
   port (
    a0, a1,a2                             : in  sfixed(4 downto -14);

    clk,start, fin_calcul,reset_gen       : in  std_logic;
    result                                : out sfixed(4 downto -14);
    fin                                   : buffer std_logic);
  end component;
COMPONENT neurone3_1

  port (
    a0, a1,a2                             : in  sfixed(4 downto -14);
    clk,start, fin_calcul,reset_gen       : in  std_logic;
    result                                : out sfixed(4 downto -14);
    fin                                   : buffer std_logic);

end COMPONENT;
  component porteand
```

```
56      port (
57        a, b : in  std_logic;
58        c    : out std_logic);
59    end component;
60
61    component registre_1
62
63    port (
64      clk,ena : in std_logic;
65       D: in sfixed (4 downto -14);
66       Q: out sfixed (4 downto -14));
67    end component;
68
69    signal fin_int : std_logic;
70    signal fin1,fin2,start2,fin,sortie_int : std_logic;
71    signal sortie1,sortie2,s :sfixed(4 downto -14);
72
73  begin  -- arch_reseau3_2_1
74
75    c0:neurone3 port map (a0,a1,a2,clk,start,fin_int,reset_gen,sortie1,fin1);
76    c1:neurone3_1 port map (a0,a1,a2,clk,start,fin_int,reset_gen,sortie2,fin2);
77    c2:porteand port map (fin1,fin2,start2);
78    c3:neurone2 port map (sortie1,sortie2,clk,start2,fin_int,reset_gen,sortie_int,fin);
79    fin_int<=fin;
80    res<=sortie_int;
81
82    end arch_reseau3_2_1;
```

3.7 SIMULATION

3.7.1 Simulation D'un neurone à trois entrés

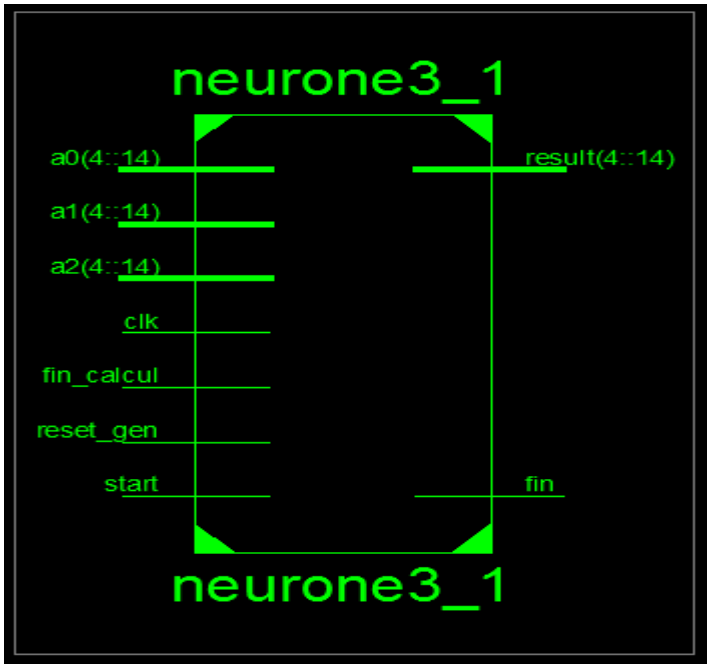


Figure 29.Vue d'ensmeble d'un neurone.

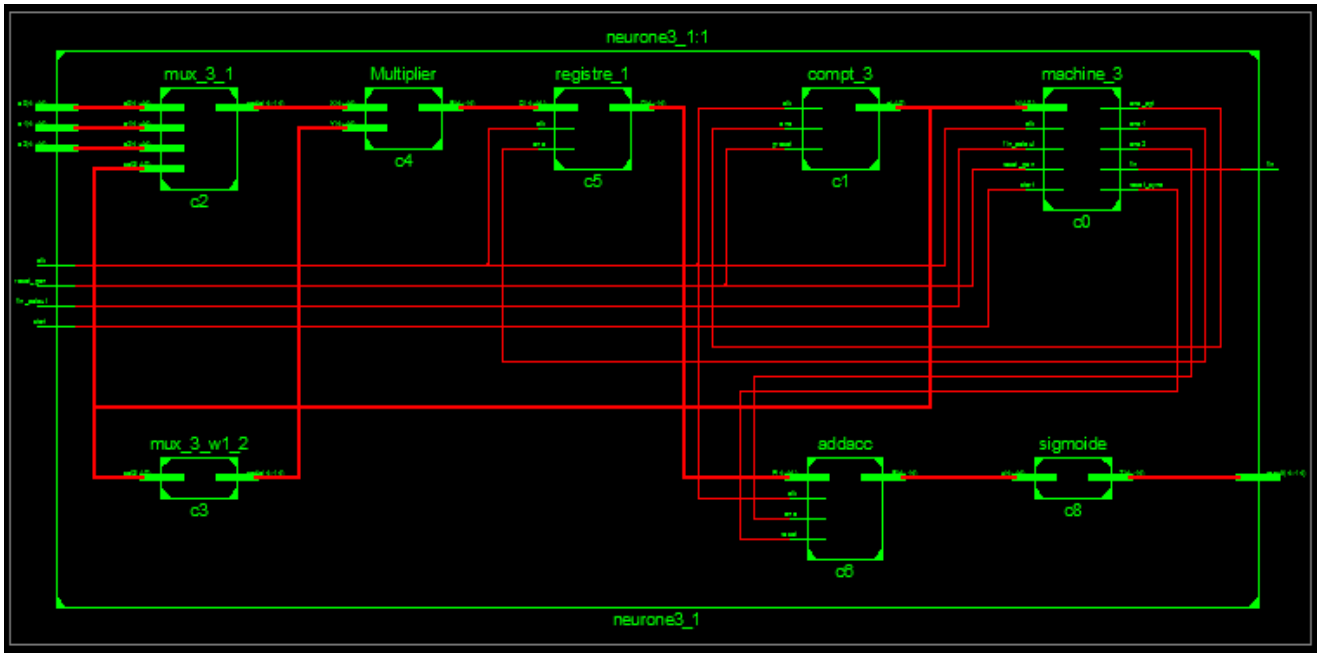


Figure39.Shéma RTL d'un neurone.

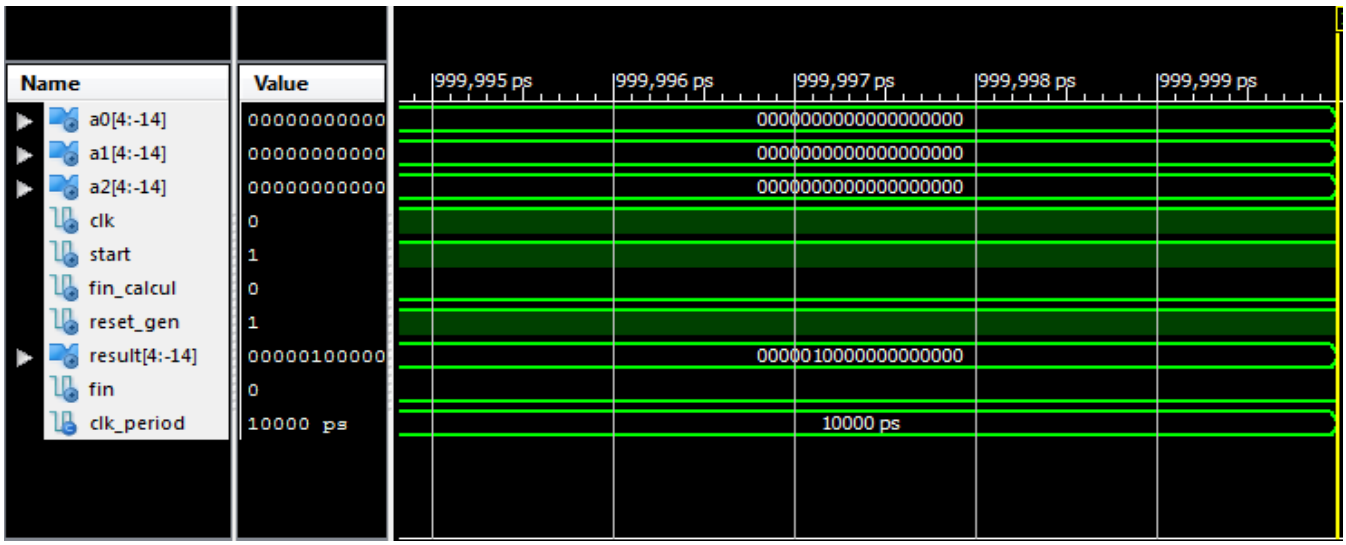


Figure 40.Simulation d'un neurone.

La figure 32 montre la simulation d'un neurone avec les valeur suivantes :

Les entrés :a0=a1=a2=0 ;

Les poids :b0=b1=b2=0 ;

Resultat=0.5 ;

3.7.2 Test de l'application

Nous avons testé notre exemple sous ISE pour plusieurs vecteurs. Notons que chaque vecteur est décrit comme suit : [Distance, Surface, Couleur].les résultats obtenus sont présenter par les simulations suivantes :

a. ***1ere exemple***

On prend le vecteur V1 suivant : [0.5, 0.75, 0.5].La simulation est décrit par la figure 41 :

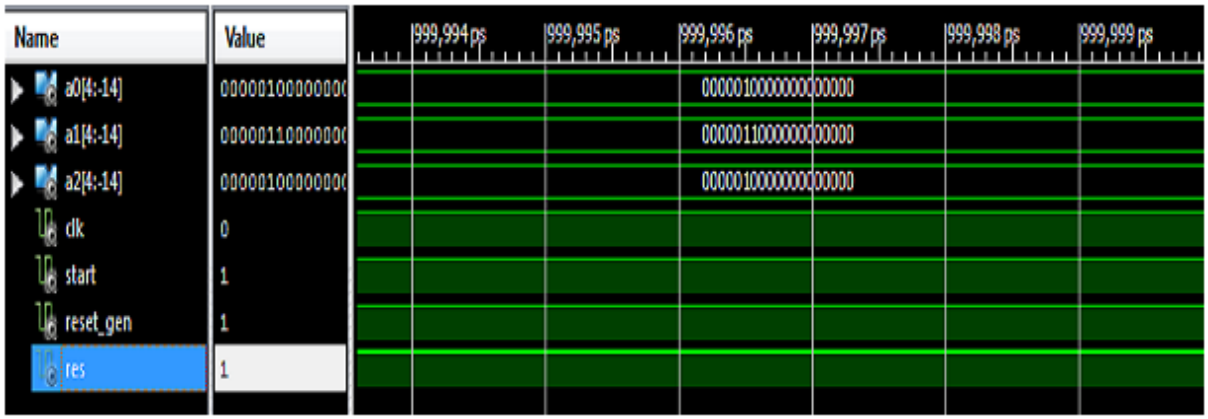


Figure 41.Résultat de simulation de V1.

Pour ce cas on a pris des valeurs de distance et surface appartenant à l'intervalle des critères favorables le variable « res » qui présente la sortie de cellule de décision prend la valeur 1 signifient que le produit appartient à la classe 1.

b. ***2eme exemple***

Le deuxième vecteur V2 est le suivant : [0.25, 0.25, 0.5]. La simulation de ce vecteur est décrite par la figure 42 :

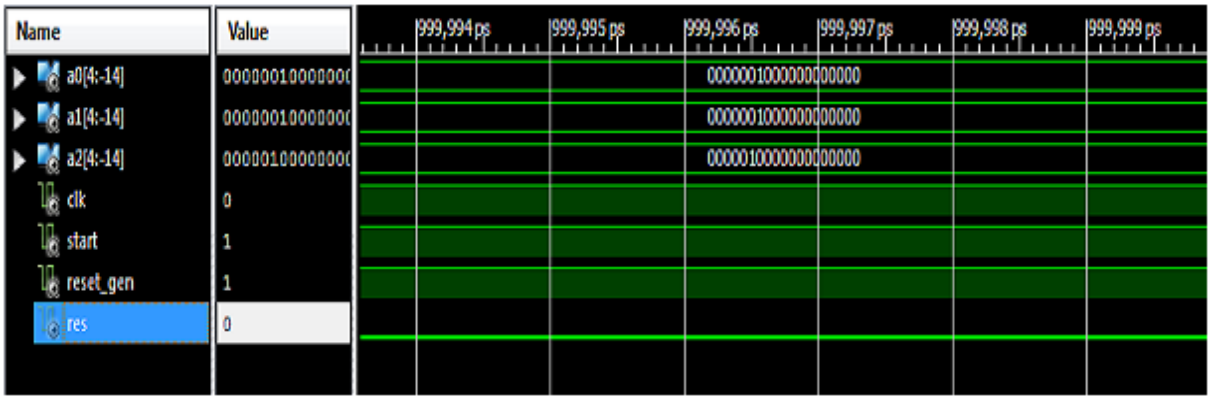


Figure 42.Résultat de simulation de V2.

Pour V2 les critères de distance et de surface ne sont pas favorables par suite la sortie de cellule de décision vaut 0.

c. ***3eme exemple***

Le troisième vecteur de test V3 : [0.75, 0.75, 0.25]. La simulation est décrite par la figure 43:

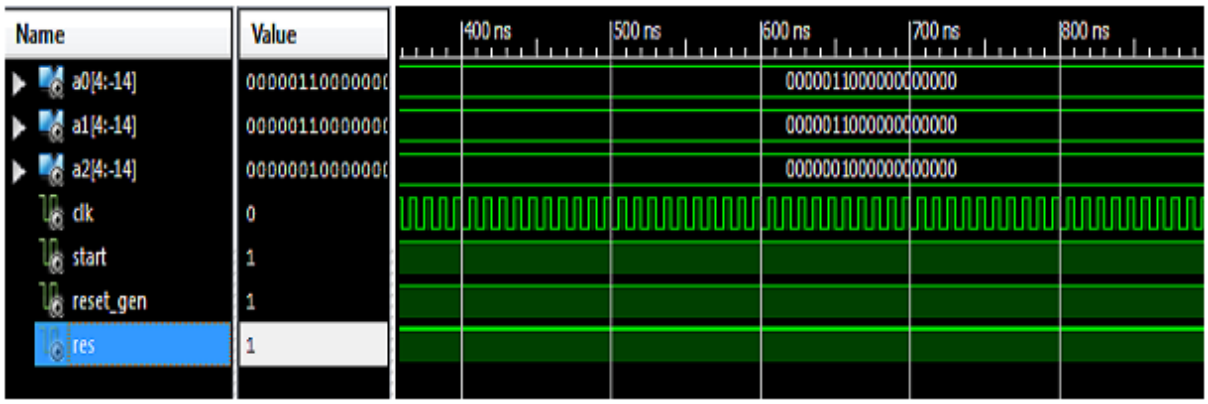


Figure 43.Résultat de simulation de V3.

La simulation de V3 montre que la variable « res » prend la valeur 1 donc le produit appartient à la premiére classe. D'après les simulations qu'on a fait on peut conclure que notre réseau fonctionne convenablement.

3.8 CONCLUSION

Au cours de ce chapitre, nous avons analysé une architecture générale d'un neurone artificiel.Aussi, l'extension d'un neurone élémentaire vers un réseau de neurones multicouches a été étudiée. Un exemple illustrant un réseau pour la classification a été implanté, simulé et testé.

Conclusion générale

Le travail effectué dans le cadre de ce mémoire se rapporte à la conception et l'implémentation d'un réseau de neurones de type FFANN sur circuit FPGA.

Dans un premier lieu, une étude de synthèse sur les réseaux de neurones nous a permis de parcourir les différents types et algorithmes des réseaux de neurones existants.

En deuxième lieu nous nous sommes intéressés à l'étude des circuits FPGA, leurs structures internes et les différents circuits FPGAs le langage VHDL et l'outil ISE Xilinx.

Par la suite nous avons présentés la représentation des donné en virgule fixe.

Avec le langage VHDL, les résultats de synthèses ont montré une grande précision atteinte par la représentation en virgule fixe peut réaliser un bon compromis précision/ressources. Concernant l'implémentation de la fonction d'activation sigmoïde, nous avons constaté que l'approximation linéaire est, à la fois, la moins onéreuse en ressources et la plus rapide. Alors que celle basée sur le polynôme approximatif, peut garantir une meilleure précision [3].

On peut dire que le but de notre projet est atteint, avec acceptante, en effet les résultats de simulation sur l'environnement ISE Xilinx ont étés obtenus comme prévu.

Il est a noté dans ce travail que la maitrise des outils de Xilinx facilite la conception et la mise en œuvre des systèmes électroniques embarqués.

Le projet que nous avons envisagé était très bénéfique pour moi. Il mon permis de concrétiser et de consolider d'une manière pratique les différentes notions que nous avons acquis durant notre formation. En particulier, ce projet nous a permis de découvrir un domaine de recherche très prometteur et d'actualité. Ceci était très utile et pourrait guider et orienter nos futurs travaux. En bref, nous pouvons dire que ce projet était un bon complément pour notre formation.

Comme perspectives des travaux réalisés, il serait intéressant d'utiliser la représentation en virgule flottante et faire une étude comparative avec l'implémentation en virgule fixe, d'étudier l'implémentation d'autres types de RNA comme les réseaux hopfild ou LVQ et faire une implémentation pratique sur circuit FPGA.

Les Règles d'apprentissage des réseaux de neurones

1 La règle de Hebb (ou règle de corrélation)

La première règle d'apprentissage a été formulée de façon qualitative, par Hebb en1949 grossièrement. Il s'agissait de renforcer la connexion reliant 2 neurones, à chaque fois qu'ils étaient actifs simultanément, dans le cas contraire elle n'est pas modifié **[2]**. Sa formalisation est la suivante : $w_{ij}(t+1) = w_{ij}(t) + \eta o_i o_j$;

Oi,oj: les sorties des deux neurones.

wij : est le poids de la connexion reliant les neurones o_i et o_j

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

t : représente l'étape d'apprentissage.

2 La règle du perceptron

Le Perceptron qui fut le premier modèle solide, sa fonction d'activation est une fonction discrète, les sorties prennent des valeurs binaires (0 ou 1) la règle d'apprentissage est la suivante :

$w_{ij}(t+1) = w_{ij}(t) + \eta x_i$ si la sortie actuelle est 0 et doit être égale à 1.

$w_{ij}(t+1) = w_{ij}(t) - \eta x_i$ si la sortie actuelle est égale à 1 et doit être égale à 0.

$w_{ij}(t+1) = w_{ij}(t)$ si la sortie est correcte.

Avec x_i : les entrées du neurone et η : Le taux d'apprentissage.

3 La règle de Windrow-Hoff ou règle delta

Après avoir étudié la règle du perceptron, Windrow et Hoff constatèrent que le Perceptron se limitait à des sorties binaires, ce qui les amena en 1960 à proposer une

règle intéressante qui consistait à utiliser l'algorithme d'apprentissage du perceptron, en considérant une fonction d'activation continue et différentielle. Dans le perceptron, le signal d'erreur utilisé pour calculer la modification des poids est égal à la différence entre la somme pondérée des entrées après le seuillage et le résultat attendu. Dans la règle de Windrow et Hoff le signal d'erreur est égal à la différence entre la somme pondérée des entrées non seuillées et le résultat attendu **[2].** Cette règle est aussi connue sous le nom de la méthode des moindres carrés dont le principe est :

- Calcul de l'erreur quadratique selon la formule :

Avec :

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

$$y_j = \sum_{i=1}^{m} X_i W_{ij}$$

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

$w_{ij}(t+1) = w_{ij}(t) + \eta x_i (d_i - y_j)$

Où :

n : le nombre de neurones à la sortie.

d_j : est la sortie désirée.

y_j : est la sortie calculée.

x_i : l'entrée du i du neurone j.

m : nombre de neurones à l'entrée.

η : coefficient d'apprentissage.

La règle de Windrow et Hoff pose le problème de l'apprentissage comme un problème de minimisation de l'erreur globale.

Bibliographie

[1] 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 Nationale d'Ingénieurs de Sousse, Juin 2014.

[2] Redouane 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.

[3] Cherrad BENBOUCHAMA,'Contribution à l'implémentation des réseaux de neurones artificiels sur hardware reconfigurable FPGA', mémoire de magister, Ecole Nationale Polytechnique, El-Harrach, Juin 2008.

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

[5] G.dreyfus, J.-M.martinez, M.Samuelides, M.B.Gordon, S.Thiria, L.Hérault, *"réseaux de neurones, méthodologie et applications"*,2002.

[6] JNH Heemskerk, *" overview of neural hardware "*, thèse de pnd, Netherlands 1995 (htp.nrc-apu.com.ac.uk/pub/nn).

[7] M.N. Cirstea, A. Dinu, J.G. Khor, M. McCormick *"Neural and Fuzzy Logic Control of Drives and Power System"*,2002.

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

[9] M. MAHIOU Tarik *"Conception et Réalisation Orientée Objet d'un Logiciel de Data Mining basé sur une Approche Hybride"* thèse pfe d'ingénieur ,U.S.T.H.B ,2005.

[10] Marc Parizeau,*" Réseaux de neurones"* université LAVAL, 2004.

[11] ISE Foundation user Manuel.