REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud Mammeri de Tizi-Ouzou
Faculté de Génie Electrique et de l'informatique
Département d'Electronique

Mémoire

de fin d'études

En vue de l'obtention du diplôme MASTER en
Electronique option TELECOMMUNICATION et RESEAUX

Thème :

Implémentation d'un réseau de neurones d'un micro capteur sur un FPGA

Proposé par :
Mr M. Laghrouche

Présenté par :
Mr Idir MELLAL

RESUME :

Dans ce travail nous avons implémenté un réseau de neurones sur un fpga .

Pour cela nous avons partagé le travail en deux parties : partie théorique et partie pratique.

Dans la première partie nous avons développé les idées suivantes :

- Les réseaux de neurones : principe, utilité et types
- Les FPGA : définition, architecture et programmation
- Le VHDL : définition et utilité
- La carte Virtex II du Xilinx : étude générale
- L'implémentation du programme sur le composant

Dans la deuxième partie, nous avons commencé par l'implémentation de 3 neurones différents sur un FPGA. Le premier neurone est à fonction de seuil. Le deuxième est un neurone à fonction linéaire f(x)=x. Le troisième est un neurone non linéaire à fonction sigmoïde. Une fois les neurones définis, il ne reste que la définition de notre réseau de neurone. Le réseau du neurone implémenté est un réseau qui modélise le comportement d'un micro capteur. Les résultats de la synthèse et de la simulation seront rapportés à chaque étape.

Nous terminons ce travail par une par une conclusion générale et quelque perspective pour l'avenir.

Avant propos :

Ce travail vise à implémenter un réseau de neurones sur un FPGA de type Virtex II en utilisant le langage de description VHDL.

*Il a été précédé d'un stage de deux mois au sein de l'équipe SoC (System On Chip) de la division microélectronique et nanotechnologie dans le centre de développement des technologies avancées "**CDTA** ".*

Pour information, le CDTA est créé par le décret n° 88.61 du 22 Mars 1988, et a pour mission d'élaborer et de mettre en œuvre des programmes de recherche scientifique et de développement technologique en technologie de l'information, architecture des ordinateurs, microélectronique, robotique et intelligence artificielle, technologie des laser, dépôts de couches minces. Il contribue activement au développement socio-économique et industriel de notre société en menant des actions de valorisation et de création de PME innovantes.

Remerciements :

Mes remerciements à mes parents, à toute ma famille et à tous mes amis.

Mes remerciements les plus sincères vont à tout ceux qui ont participé à ce travail et ceux qui m'ont soutenu en particulier mon promoteur Mr Laghrouche et Mr Achour pour leurs conseils et orientations.

Je remercie *l'équipe SoC (System On Chip) de la division microélectronique et nanotechnologie du centre de développement des technologies avancées "**CDTA** " à* leur tête Melle F.ABID et Melle N. IZEBOUDJENE.

Je remercie aussi les membres du jury.

Table des matières :

Chapitre *II*: LE CIRCUIT FPGA

Chapitre III : l'implémentation

Chapitre IV : la partie pratique

Liste des figures :

Introduction générale :

Depuis l'antiquité l'être humain n'a cessé d'améliorer son quotidien en s'inspirant en premier lieu des phénomènes naturels. En étudiant tout ses phénomènes, on cherche à les reproduire pour faciliter notre quotidien. Ses dernières années avec l'apparition des nouvelles technologies, l'étude de ces phénomènes est devenue de plus en plus facile et simple. Alors l'homme a pu résoudre plusieurs questions qui révélaient de l'impossible il ya quelques années.

Réseaux de neurones, algorithme génétique, colonies de fourmis…..Tous inspirés de la nature. Dans ce travail, nous allons faire l'implémentation d'un réseau de neurones sur un composant qui est toujours sujet de recherche qui s'appelle FPGA (Feild Programmable Gate Array). Pour cela nous avons divisé ce mémoire en deux parties, une partie théorique et une partie pratique.

Dans la première partie, nous avons suivi le plan suivant :

Au premier chapitre nous allons faire une étude générale des réseaux de neurones dans laquelle nous donnons des définitions et les applications de ces derniers.

Au deuxième chapitre, nous étudierons en général les composants programmables et les FPGA en particulier. Nous allons aussi aborder le langage VHDL.

Au troisième chapitre, nous étudierons l'implémentation des RNA. Nous parlerons aussi des outils utilisés pour la réalisation de ce travail. Les étapes d'implémentation seront aussi développées dans ce chapitre

Dans la partie pratique, nous développons les titres suivants :

- Architecture matérielle des neurones
- Programmation d'un neurone: -à fonction de seuil
  -à fonction linéaire
  -à fonction sigmoïde
- Programmation des couches du réseau
- Programmation du réseau

Nous terminerons ce travail par une conclusion générale suivie par une annexe qui contient tout les programmes en vhdl.

Chapitre *I : Introduction aux réseaux de neurones*

I.a- Introduction :

Dans ce chapitre nous allons rappeler les définitions fondamentales relatives aux réseaux de neurones ainsi que leurs propriétés mathématiques. Nous décrirons le principe de fonctionnement en faisant une comparaison avec le neurone biologique. Nous nous basons sur la propriété la plus importante en l'occurrence l'apprentissage et nous montrons ses types. Enfin nous donnons les domaines d'application de cette technique tout en donnant quelques exemples.

I.b- le neurone biologique et le neurone artificiel :

Le premier neurone réalisé remonte à 1943 quand MM. Mac Culloch et Pitts , ont fait une tentative de modélisation mathématique du cerveau humain, ils ont réalisé un neurone avec une sortie binaire. Ils ont démontré théoriquement qu'un réseau de ces neurones formels peut reproduire des fonctions logiques et mathématiques.[1]

Un neurone est une cellule du système nerveux, il est composé de :

- Les dendrites : ce sont des réseaux réceptifs arborescents de fibres nerveuses qui conduisent les signaux électriques dans la cellule.
- Le corps cellulaire : il somme efficacement les signaux reçus et en fixe le seuil.
- L'axone : c'est une simple fibre longue qui conduit le signal de la cellule aux autres neurones.

Le point de contact entre un axone d'une cellule et une dendrite d'une autre cellule est appelé synapse. C'est l'arrangement des neurones et des forces des synapses individuelles, déterminées par un processus chimique complexe qui établit la fonction effectuée par le réseau de neurones.

La Figure I.1 illustre un schéma d'un neurone biologique :

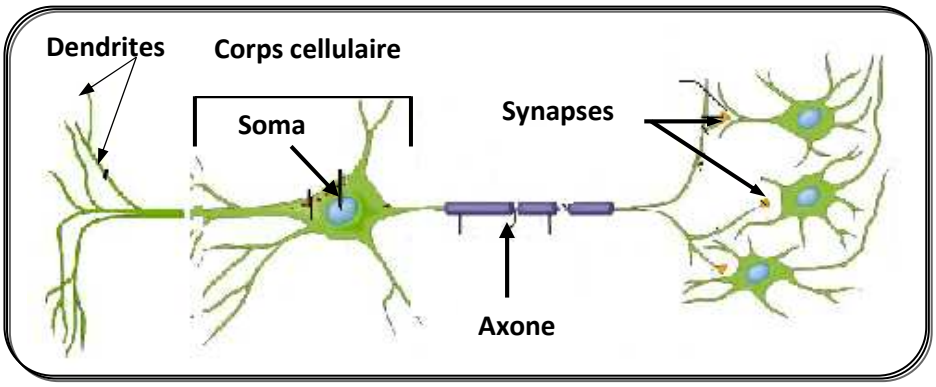


Figure I.1 : Neurone biologique

I.c- Le neurone formel (Artificiel) :

I.c.1- Le modèle mathématique :

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

La figure I.2 décrit le fonctionnement global d'un neurone artificiel :

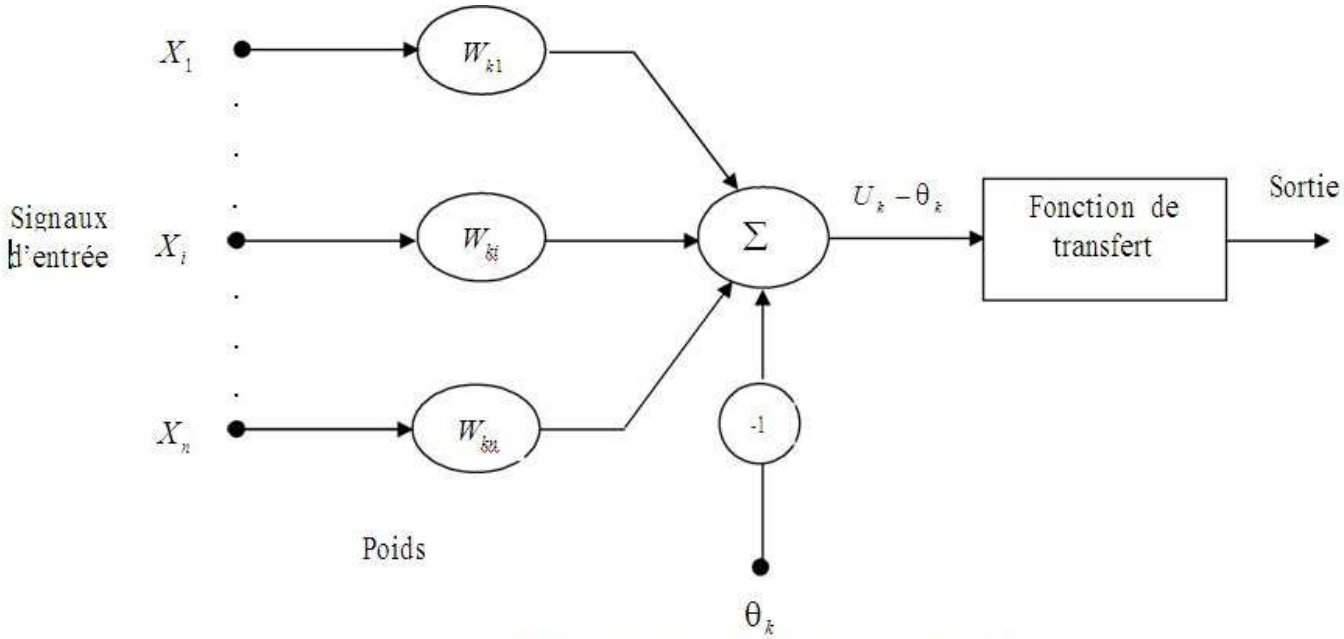


Figure I.2 : Schéma fonctionnel du neurone formel.

On peut donc, d'après la figure I.2, définir un neurone formel par les paramètres suivants :

1. Un ensemble des connexions (ou synapses) dont chacune se caractérise par un poids réel. Le signal X_i se trouvant à l'entrée de la synapse i qui est connectée au neurone k. Ce signal est multiplié par le poids de la synapse w_{ki}, w_{ki} étant le poids de la connexion dirigée du neurone i vers le neurone k. Si le poids est positif on aura un effet excitateur, s'il est négatif l'effet est inhibiteur.
2. La somme pondérée $U_k - \theta_k$ des signaux d'entrée $(X_i)_{1\le i\le n}$ du neurone i qui sont en même temps les signaux de sortie des neurones de la couche amont auquel ce neurone est connecté est donnée par l'expression I.1 :

$$U_k - \theta_k = \sum_{i=1}^{n} w_{ki} X_i + (-1)\theta_k = \sum_{i=0}^{n} w_{ki} X_i \qquad (I.1)$$

Où : $w_{i0} = \theta_k$, $X_0 = 1$ et n le nombre de neurones de la couche en amont.

3. Le seuil θ_k propre au neurone k est un nombre réel représentant la limite à partir de laquelle le neurone s'activera. Ce seuil peut jouer le rôle de poids de la connexion qui existe entre l'entrée fixée à +1 et le neurone k .
4. La fonction de transfert f limite en général l'amplitude de la sortie $Y_k = f\left(U_k - \theta_k\right)$ du neurone entre $[-1,+1]$ ou $[0,+1]$. Elle existe sous différentes formes, telles que : fonction non linéaire, fonction à seuil binaire, fonction linéaire à seuil, …..etc. [3]

On distingue deux phases qui décrivent le comportement d'un neurone (i):

- Le calcul de la somme pondérée des entrées (A_i), selon l'expression suivante :

$$A_i = \sum W_{ji} X_j \qquad \textbf{(I.2)}$$

- A partir de cette valeur, une fonction seuil calcule la valeur de l'état du neurone :

$$Y_i = f(A_i) \qquad \textbf{(I.3)}$$

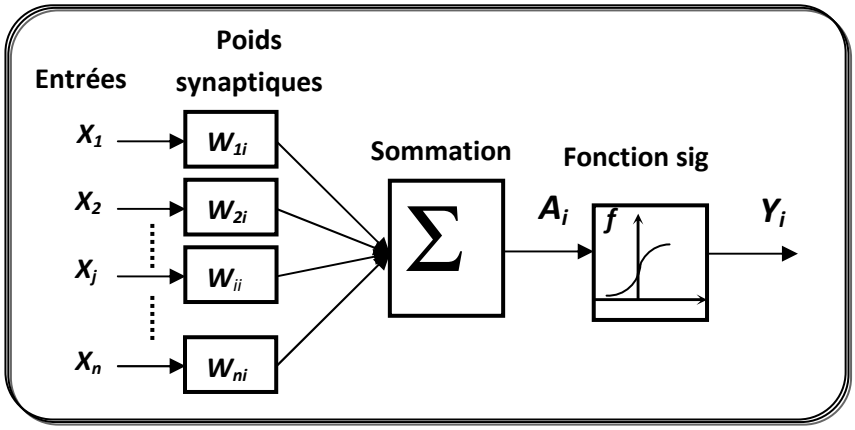


Figure I.3 : Modèle mathématique du neurone

Où :

X_j : les entrées, qui peuvent être les états des neurones en amant.

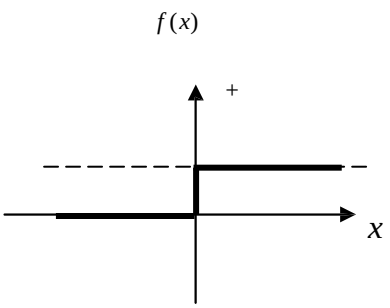
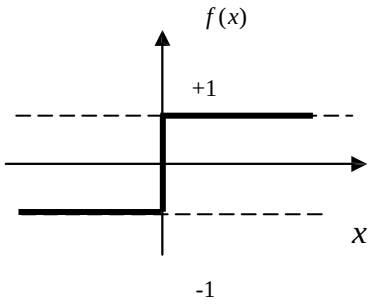
Y_i : l'état d'un neurone i.

A_i : l'activité du neurone i.

W_{ji} : le poids synaptiques de la connexion entre les neurones j et i.

I.c.2- Les fonctions de transfert:

En général, la fonction de transfert f d'un neurone représente l'état d'activation de celui-ci. Les plus courantes sont présentées sur la table suivante, avec leurs équations mathématiques. On remarquera qu'à la différence des neurones biologiques dont l'état est binaire, la plupart des fonctions de transfert offrent une infinité de valeurs comprises dans l'intervalle $[0,+1]$ ou $[-1,+1]$.

Catégorie	Type	Equation	Allure
Seuil	Binaire (fonction Heaviside)	$f(x) = \begin{cases} 0 \ si \ x < 0 \\ 1 \ si \ x \geq 0 \end{cases}$	
	Signe	$f(x) = \begin{cases} 1 \ si \ x > 0 \\ -1 \ si \ x \leq 0 \end{cases}$	

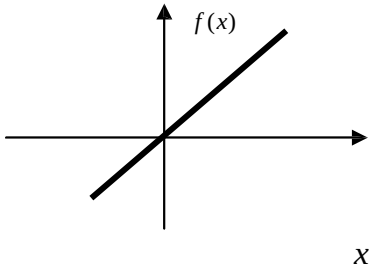
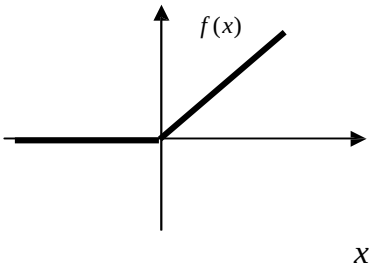
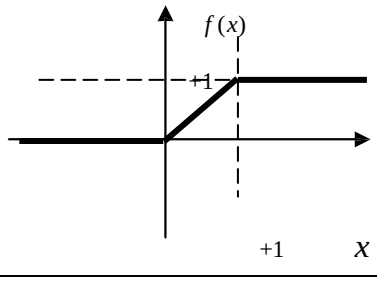
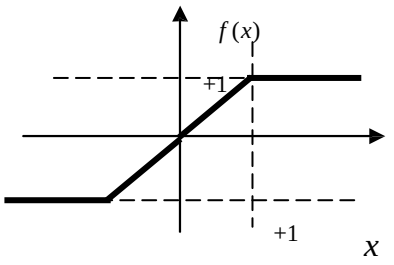
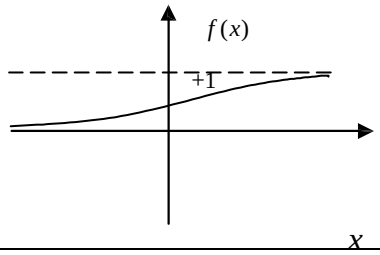
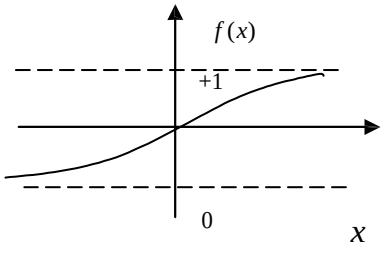
Linéaire	Identité	$f(x) = x$	
	Linéaire positif	$f(x) = \begin{cases} 0 \ si \ x < 0 \\ x \ si \ x > 0 \end{cases}$	
	Saturé positif	$f(x) = \begin{cases} 0 \ si \ x < 0 \\ 1 \ si \ x \geq 1 \\ x \sin on \end{cases}$	
	Saturé symétrique	$f(x) = \begin{cases} -1 \ si \ x \leq -1 \\ 1 \ si \ x \geq 1 \\ x \sin on \end{cases}$	
Non linéaire	Logistique (Sigmoïde)	$f(x) = \frac{1}{1+e^{-x}}$	
	Tan-sigmoïde (Tanh)	$f(x) = \frac{2}{1+e^{-x}} - 1$	

Table I.1 : Les fonctions de transfert.

I. d- Réseaux de neurones artificiels :

Un neurone élémentaire est très limité, en effet, un neurone réalise une simple fonction non linéaire, paramétrée de ses variables d'entrée. L'intérêt des neurones apparait dans la propriété qui réside en leur association dans une structure par une certaine logique d'interconnexion, cette structure est appelée : le réseau de neurones ou bien par abréviation ANN (Artificial 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 simple. Dans un tel réseau, les entrées d'un neurone sont soit les entrées du réseau global, soit les sorties d'autres neurones. Le pouvoir de traitement (La connaissance) du réseau est stocké dans les poids synaptiques dont les valeurs sont en général déterminées par une opération dite l'apprentissage. La structure d'interconnexion entre les différents neurones détermine la topologie du réseau. On peut classer les réseaux de neurones artificiels selon 2 critères différents [3]:

- Selon l'architecture : qui comporte trois grandes catégories :
  1. Réseau neuronal multicouche de type Feed-Forward (réseau non bouclé)
  2. Réseau récurent (réseau bouclé).
  3. Réseau cellulaire.

- Selon le mode d'apprentissage :
  1. Apprentissage supervisé
  2. Apprentissage non supervisé
  3. Apprentissage renforcé

I.d.1-Classification selon l'architecture :

I.d.1.1- Réseaux multicouches de types Feed-Forward*:*

Les «Feed-forward networks» : ce sont des réseaux à circulation de l'information vers l'avant, et dont lesquels l'organisation des neurones est en couches successives. Le calcul se fait en propageant les données de l'entrée vers la sortie. Dans cette catégorie on distingue les réseaux à une seule couche (exemple: le Perceptron) et les réseaux multicouches (possédant une couche d'entrée, une couche de sortie et une ou plusieurs couches cachées). Dans ce type, chaque neurone est connecté à tous les neurones de la couche précédente. La figure suivante nous montre bien son architecture.

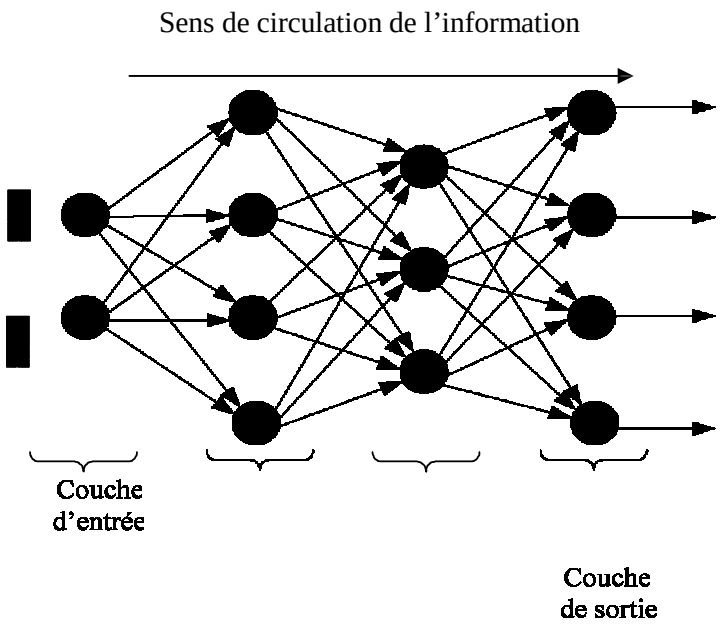


Figure I.4 : structure d'un réseau neuronal Feed-Fo- ward.

I. d.1.2- Réseaux récurrents :

Les réseaux récurrents sont appelés également réseaux bouclés ou dynamiques (Figure I.5), ce sont des réseaux dans lesquels il y a retour en arrière de l'information. Des connexions apparaissent entre les sorties du réseau et les neurones qui se trouvent dans des couches en amont. La sortie d'un neurone du réseau peut donc être fonction d'elle même, alors la notion du temps est explicitement prise en considération . A chaque connexion d'un neurone bouclé est attaché un retard, multiple entier de l'unité de temps choisie. La figure I.4 illustre la structure d'un tel réseau :

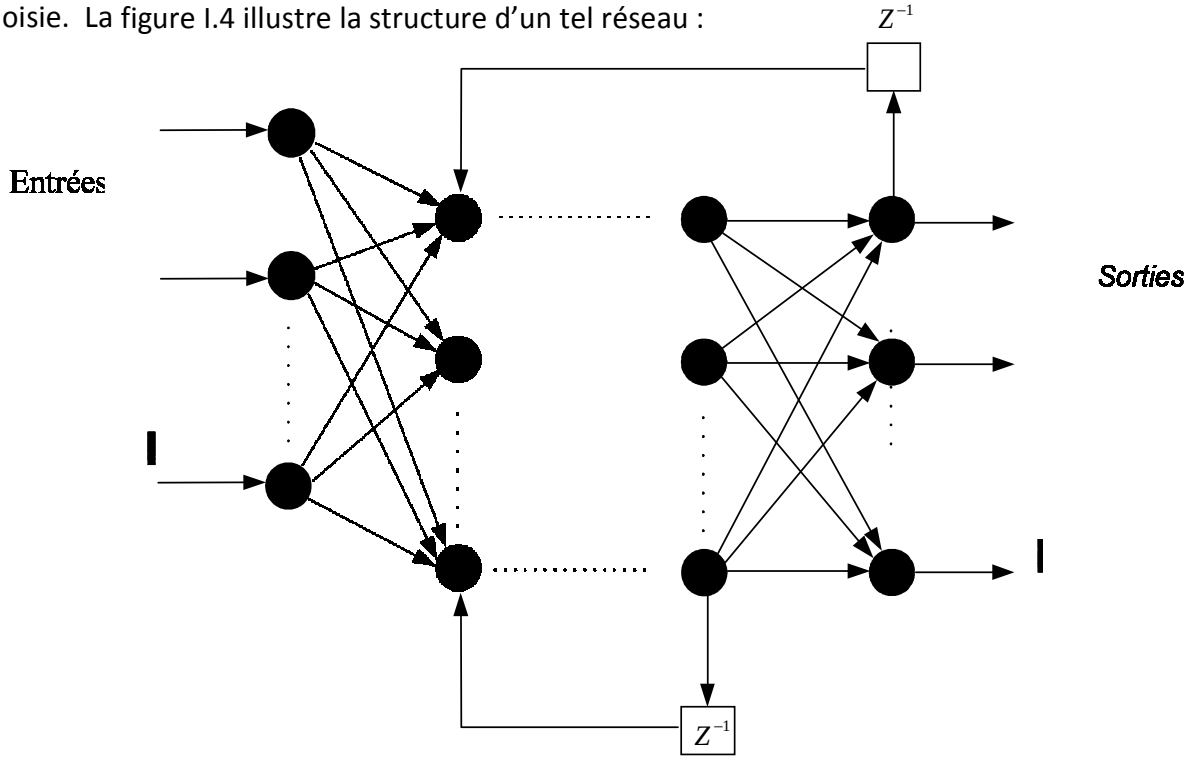


Figure I.5: Structure d'un réseau neuronal récurrent.

Ces réseaux sont souvent utilisés pour identifier des systèmes dynamiques non linéaires.

I.d.1.3- Réseaux cellulaires*:*

Dans un réseau cellulaire (Figure I.6), les neurones sont entièrement connectés dans un plan où chaque neurone est relié à tous les neurones qui l'entourent. Une connexion entre deux neurones peut être bidirectionnelle.

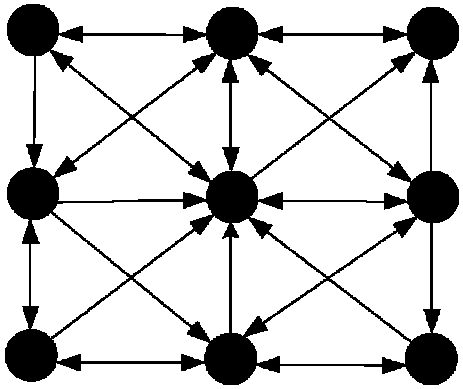


Figure I.6 : Réseau cellulaire.

I.d.2-Classification selon l'apprentissage :

Définition

L'apprentissage est une phase de développement d'un réseau de neurone, elle consiste à estimer les paramètres des neurones du réseau, à fin que celui-ci remplisse au mieux sa tache qui lui est affectée. Pour cela il nécessite :

- Un ensemble d'exemples d'apprentissages : en effet les réseaux de neurones sont des fonctions paramétrées, utilisées pour réaliser des modèles statistiques à partir d'exemples (dans le cas de la classification) ou de mesures (dans le cas de la modélisation) ; leurs paramètres sont calculés à partir de ces exemples ou couples {entrée, sortie}.

- La définition d'une fonction de coût qui mesure l'écart entre les sorties du réseau de neurones et les sorties désirées.

- Un algorithme de minimisation de la fonction de coût par rapport aux paramètres.

I.d.2.1- Apprentissage supervisé :

Dans l'apprentissage supervisé, un superviseur qui connaît parfaitement la sortie désirée ou correcte, guide le réseau en lui apprenant à chaque étape le bon résultat. Donc l'apprentissage ici, consiste à comparer le résultat obtenu avec le résultat désiré, puis à ajuster les poids des connexions pour minimiser la différence entre les deux comme l'indique la **Figure I.7**

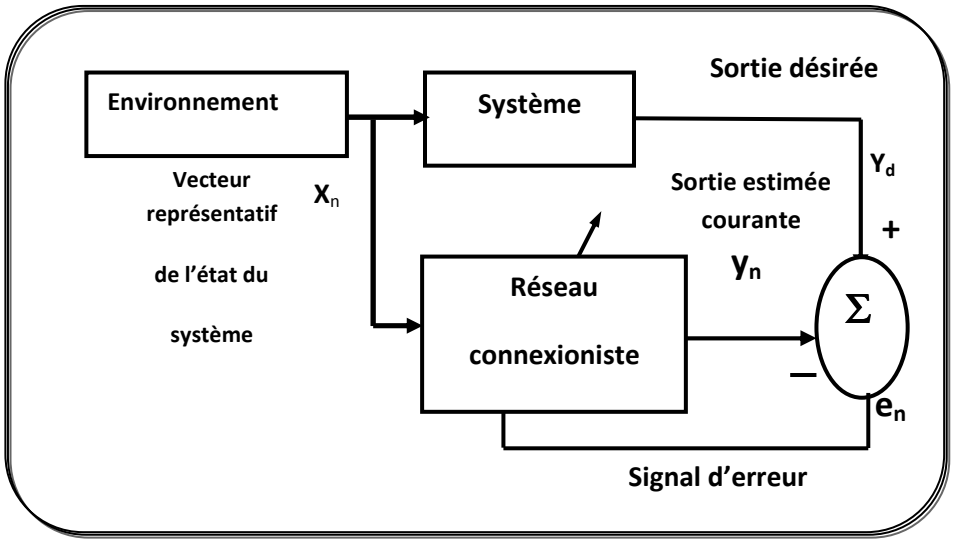


Figure I.7 : Principe de l'apprentissage supervisé

Dans cette figure, le système à apprendre constitue un superviseur pour le réseau. Le Réseau connexionniste prend comme stimulus le même vecteur des variables explicatives que le système. Ce dernier, en réponse à l'entrée présentée, fournit une sortie désirée qui représente pour le réseau le comportement de référence. Parmi les algorithmes à apprentissage supervisé, on reconnaît dans la littérature l'algorithme à Rétro-propagation du gradient.

I.d.2.2- Apprentissage non supervisé :

L'apprentissage supervisé s'effectue sous le contrôle d'un expert, alors que l'apprentissage non supervisé est autodidacte. Les paramètres internes du réseau ne sont modifiés qu'avec les seuls stimuli, aucune réponse désirée n'est prise en considération. La **Figure I.8** montre que la sortie du réseau n'est pas utilisée par la procédure d'apprentissage.

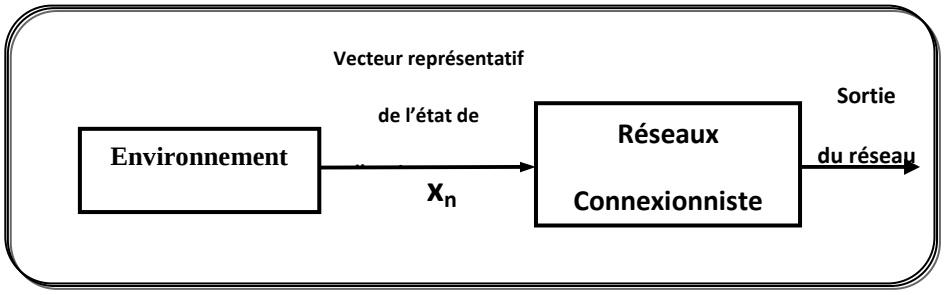


Figure I.8 : Principe de l'apprentissage non supervisé

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

I.d.2.3- Apprentissage renforcé :

C'est un apprentissage qui élabore et assimile une critique à l'issue de chaque exemple traité par la machine d'apprentissage. En effet l'algorithme de critique utilise des indications imprécises sur le comportement final «échec ou succès» du réseau à travers le vecteur des variables explicatives de la **Figure I.9**. L'ajustement des poids du réseau se fait uniquement à partir d'une simple évaluation de la qualité de sa réponse.

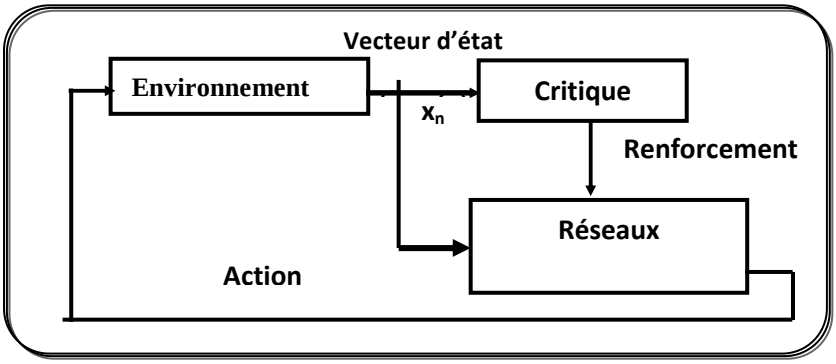


Figure I.9 : Principe de l'apprentissage par renforcement

I.e- Perceptron multicouches (MLP) :

Le perceptron multicouche MLP (Figure I.9) est un réseau neuronal de type Feed-forward, avec une ou plusieurs couches cachées entre l'entrée et la sortie. Chaque neurone dans une couche est connecté à touts les neurones de la couche précédente et de la couche suivante (excepté les couches d'entrée et de sortie) et il n'y a ni connexions entre les neurones d'une même couche ni entre deux couches non successives. Les fonctions d'activation utilisées dans ce type de réseaux sont principalement les fonctions sigmoïdes.

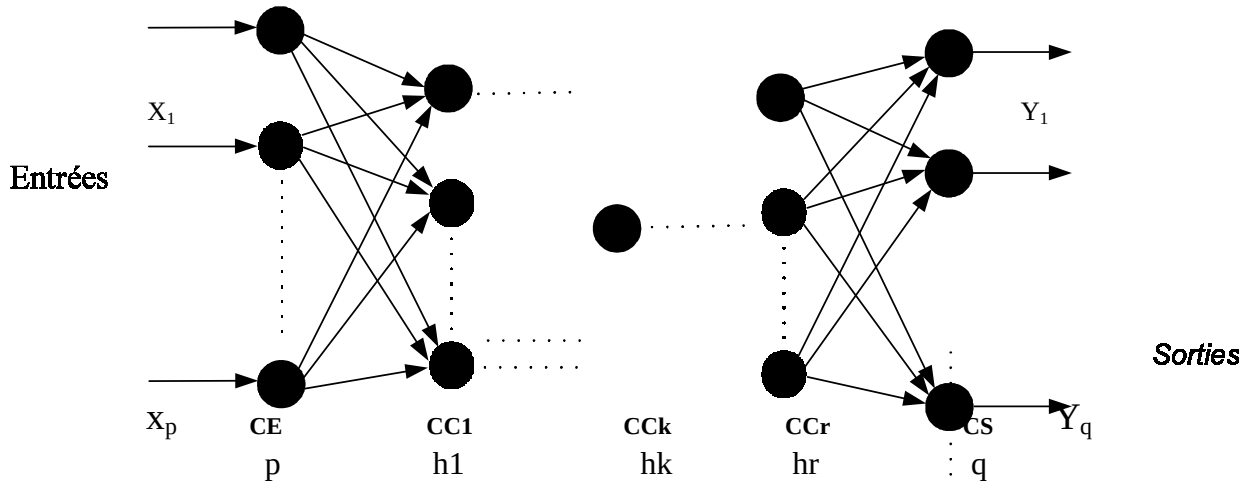


Figure I.10 : Perceptron multicouches.

CE : couche d'entrée. CC : couche cachée. CS : couche de sortie.
p : nombre de neurones d'entrée. q : nombre de neurones de sortie.
r : nombre de couches cachées.
h_k : nombre de neurones dans la couche cachée numéro k.

I.f- Choix de l'architecture d'un réseau de neurones :

Le choix de l'architecture d'un réseau de neurones joue un rôle crucial dans le contrôle de sa capacité, il consiste en un problème d'optimisation. Le processus d'optimisation a pour objectif de réduire dans la mesure du possible le nombre d'unités cachées ainsi que la complexité du réseau et d'améliorer les temps de calcul tout en gardant les capacités de généralisation qui consistent à obtenir des approximations satisfaisantes. En effet si le nombre de neurones et le nombre de couches cachées est trop petit, le réseau possède trop peu de paramètres et ne peut capter toutes les dépendances qui servent à modéliser la fonction de régression, on obtiendra de ce fait de mauvaises performances en apprentissage et en validation. À l'inverse, si le nombre de neurones ou le nombre de couches cachées est trop grand, le nombre de paramètres du modèle augmente, ce qui entraînera un surapprentissage, donc une limite au niveau de la généralisation. En général le nombre de couches cachées d'unités est déterminé empiriquement. Cela consiste à tester plusieurs architectures puis retenir la meilleure.

I.g- Avantages et inconvénients des réseaux de neurones :

Dans ce qui suit nous allons présenter les avantages et les inconvénients des réseaux de neurones.

I.g.1. Avantages :

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

I.g.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).

I.h- Domaines d'application et mise en œuvre

- Robotique et Capteurs
- La modélisation de processus dynamiques non linéaires
- La commande de processus
- La classification et Reconnaissance des formes
- Traitement de signal
- Approximation et prédiction des fonctions et autres….

Pour utiliser les réseaux de neurones dans l'une des ces applications il faut suivre certaines procédures :

- Il faut tout d'abord choisir l'architecture du réseau, c'est-à-dire les entrées externes, le nombre de neurones cachés, et l'agencement des neurones entre eux, de telle manière que le réseau soit en mesure de reproduire ce qui est déterminé dans les données, le nombre de poids ajustables est un des facteurs fondamentaux de la réussite d'une application : si le réseau possède un trop grand nombre de poids, c'est-à-dire si le réseau est trop "souple", il risque de s'ajuster au bruit qui est présent dans les données de l'ensemble d'apprentissage, et, même en l'absence de bruit, il risque de présenter des oscillations non significatives entre les points d'apprentissage, donc de posséder de mauvaises propriétés d'interpolation ou "généralisation", si ce nombre est trop petit, le réseau est trop "rigide" et ne peut reproduire la partie déterministe de la fonction. Le problème de la détermination de l'architecture optimale est resté pendant longtemps un problème ouvert, mais il existe actuellement diverses méthodes, mettant notamment en jeu des tests statistiques, qui permettent de déterminer cette architecture pour une vaste classe de réseaux.

- Il faut calculer les poids du réseau ou, en d'autres termes, estimer les paramètres du réseau à partir des exemples, en minimisant l'erreur d'approximation sur les points de l'ensemble d'apprentissage, de telle manière que le réseau réalise la tâche désirée. Ce calcul des coefficients synaptiques constitue l'apprentissage supervisé pour le réseau de neurones.

- Il faut enfin estimer la qualité du réseau obtenu en lui présentant des exemples qui ne font pas partie de l'ensemble d'apprentissage.

Jusqu'à ce moment nous n'avons parlé que des ANNs en software sans aborder la matérialisation de cet outil important. L'implémentation des ANNs peut être faite de plusieurs façons selon les performances que l'on cherche de l'application. En fait, il est possible d'implémenter un ANN en utilisant des Microcontrôleurs.[4]

Selon les performances requises de l'application, les réseaux de neurones peuvent être aussi implémentés en software sur un ordinateur conventionnel, à l'aide de processeurs DSPs (**D**igital **S**ignal **P**rocessing) programmables (circuits à application générale), en VLSI (**V**ery **L**arge **S**cale **I**ntegration) sur du silicium (circuits dédiés en full custom, cellules standards, FPGA), en optique ou en combinant ces techniques.

I.i- Conclusion :

A travers ce chapitre, nous avons étudié les ANNs d'une manière générale et les différents modes et architectures dont ils disposent. Nous avons vu leur classification selon l'architecture ou le mode d'apprentissage. Nous avons abordé quelques limites et avantages.

Dans le chapitre suivant, nous allons étudier les FPGA et le langage de programmation VHDL que nous allons utiliser dans la partie pratique.

CHAPITRE II : LE CIRCUIT FPGA

II.a- Introduction :

Dans ce chapitre, nous allons étudier les FPGA, en commençant par une introduction aux circuits programmables qui ont précédé les FPGA. On montre l'architecture, les types et les avantages de ce composant. En dernière partie, nous faisons un rappel du VHDL et de son utilité.

II.b- Les circuits programmables :

Il y a quelques années la réalisation d'un montage en électronique numérique impliquait l'utilisation d'un nombre important de circuits intégrés logiques. Ceci avait pour conséquences un prix de revient élevé, une mise en œuvre complexe et un circuit imprimé de taille importante.

En 1985, la société Xilinx introduisait un nouveau type de circuit électronique appelé *FPGA* (*Field-Programmable Gate Array*) destiné à remplacer de petites quantités de logique discrète. Le principe de ces méta-circuits, dérivés des *PAL* (*Programmable Array Logic*), est de fournir une certaine quantité de logique, entièrement programmable : après une phase de configuration, le FPGA se comporte comme le circuit décrit par sa programmation. L'originalité des FPGA Xilinx est de stocker la configuration dans une mémoire vive statique (*SRAM*), permettant ainsi de reprogrammer ces circuits à volonté, y compris dans un système en train de fonctionner. Cette reprogrammabilité ne faisait pas partie des objectifs de Xilinx, mais elle a permis d'ouvrir les FPGA à un nouveau champ d'applications.

En regroupant des FPGA sur une carte, on peut fabriquer un coprocesseur physiquement figé, mais capable de se comporter comme n'importe quel circuit d'une taille maximale donnée.

Ce coprocesseur matériel reconfigurable, nommé Mémoire Active Programmable (ou *PAM* pour *Programmable Active Memory*) développé en 1987 par Jean Vuillemin, peut être utilisé dans de nombreuses applications. Il ne nécessite pas de développement matériel, et la conception de configurations est nettement plus simple que la véritable conception de circuits intégrés. De plus, la possibilité de compiler une configuration, et de l'essayer immédiatement permet d'obtenir des cycles de mise au point très courts. La facilité de développement sur PAM s'apparente donc largement à celle du logiciel. D'autre part, malgré le coût en densité et en vitesse dû à la configurabilité, les applications sont bien implantées sous forme de circuits, et l'on peut donc espérer obtenir des performances du même ordre que celles des solutions sur mesure.

Les Mémoires Actives Programmables représentent donc une solution intermédiaire entre les implantations logicielles et matérielles. Elles essaient de combiner les avantages de ces deux approches, sans en avoir les inconvénients.

Xilix a effectué une étude de touts ses composants, avant de développer les FPGA, alors elle a constaté que les dérives de PLD conduisent relativement à un gaspillage de ressource vu le nombre important des ressources inutilisés et perdus en même temps.[5]
Le schéma suivant illustre les différents circuits programmables disponibles sur la marche classée selon leurs architectures et leur reprogrammabilité. On distingue :

- PLD : Programmable Logic Device
- PAL : Programmable Array Logic
- PLA : Programmable Logic Array
- CPLD : Complex Programmable Logic Array
- ASIC : Applicated Specific integrated circuit
- FPGA : Field Programmable Gate Array
- Pic / Microcontroller
- Mémoires (RAM,ROM,PROM …) …

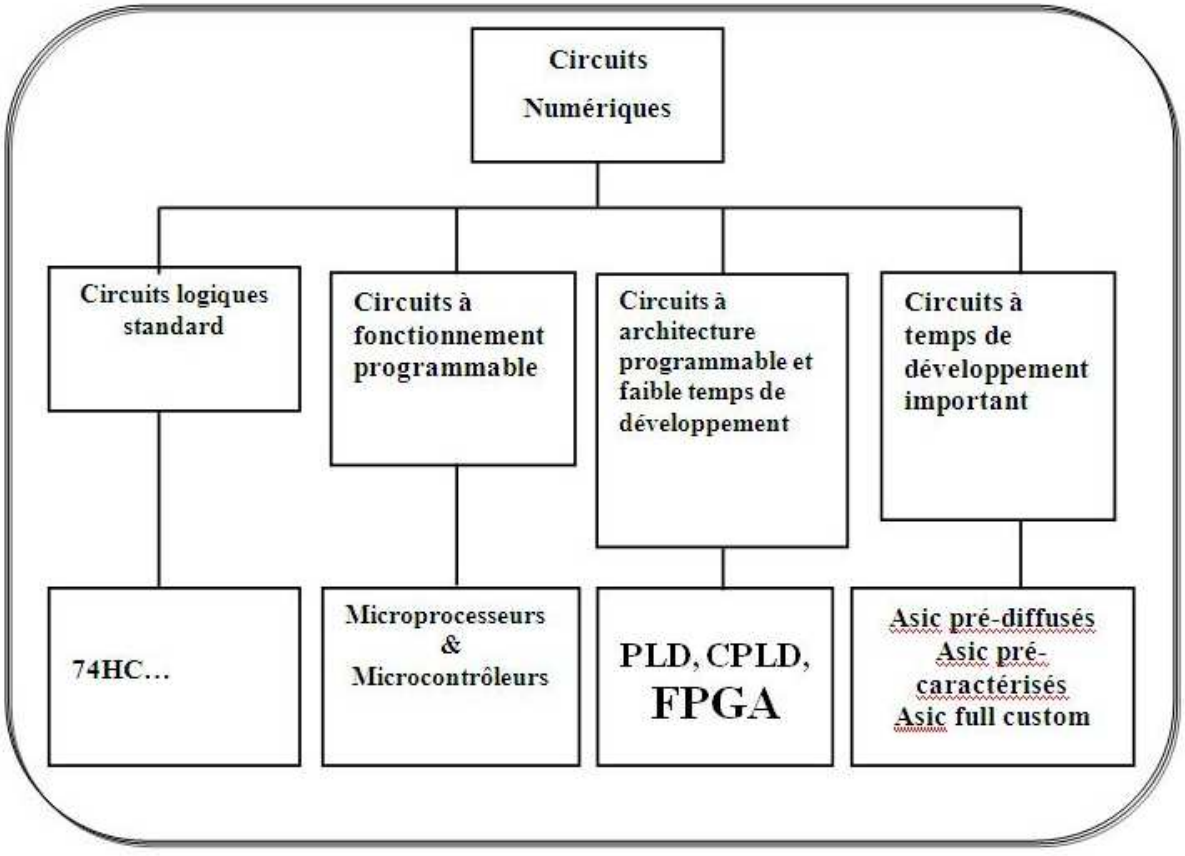


Fig :II-1 : circuits programmables

II.c- Les FPGA :

Les FPGAs (**F**ield **P**rogrammable **G**ate **A**rrays) ou "réseaux logiques programmables" sont des composants VLSI entièrement reconfigurables ce qui permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs. Les circuits FPGAs 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.

Les FPGA, sont aujourd'hui utilisés dans des systèmes électroniques complexes de plus en plus variés. Pour répondre aux exigences du marché les fabricants proposent des gammes de choix très étendues, dans lesquelles on trouve des architectures, des tailles, des fonctionnalités bien différentes. Le développeur se trouve alors face à un large choix dans l'espace de conception d'où la disposition des différents outils capables de comparer entre elles les différentes architectures de composants reconfigurables.

Grâce aux évolutions de la technologie microélectronique, les FPGA deviennent de plus en plus performants avec des capacités sans cesse d'augmenter (jusqu'à 10 millions de portes pour quelques circuits Virtex de Xilinx). Longtemps réalisés autour de blocs de logique configurable à base de LUT (Look Up Table), les FPGA peuvent aujourd'hui comporter de larges mémoires RAM configurables, des opérateurs arithmétiques complexes (comme les Block Select RAM et les blocs multiplieur du Virtex II de Xilinx) et des cœurs de microprocesseurs (tel que le cœur ARM 9 intégré sur la puce des composants Excalibur d'Altera). Leur complexité et leurs possibilités en terme de réalisation de systèmes électroniques sur puce (SOC, System On Chip), ont permis aux FPGA de ne plus être seulement utilisés pour des applications de types prototypages rapides ou Glue Logic, prenant ainsi de la place dans l'espace de conception proposé aux développeurs, longtemps contraints d'utiliser des circuits du type ASIC (Application Specific Integrated Circuit), chers et de fabrication délicate.

II.c.1- Structure et architecture :

Un circuit FPGA contient un très grand nombre de macro cellules avec une très grande souplesse d'interconnexion entre eux. Dans le FPGA, le temps de propagation dans les couches logiques du circuit dépend de l'organisation et de la distance entre les macro-cellules interconnectées. L'architecture, retenue par Xilinx, se présente sous forme de deux blocs :

- Un bloc appelé circuit configurable.
- Un bloc appelé réseau mémoire SRAM.

La couche dite "circuit configurable" est constituée d'une matrice de blocs logiques configurables CLB permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs entrés/sorties (IOB) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs. La programmation du circuit FPGA appelé aussi LCA (logic cells arrays) consistera par le biais de l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ à interconnecter les éléments des CLB et des IOB afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM.

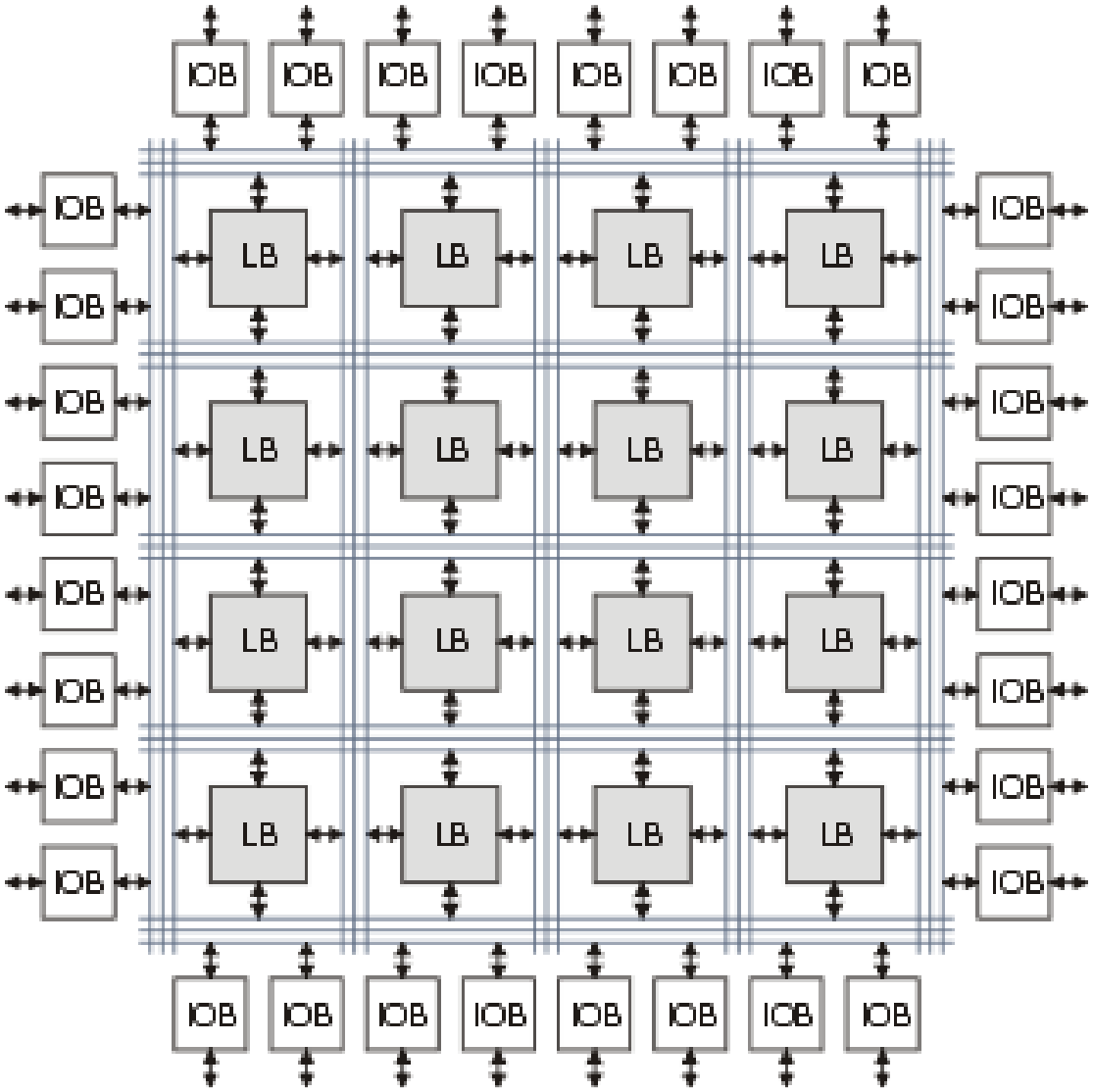


Fig II-2 : architecture des fpga

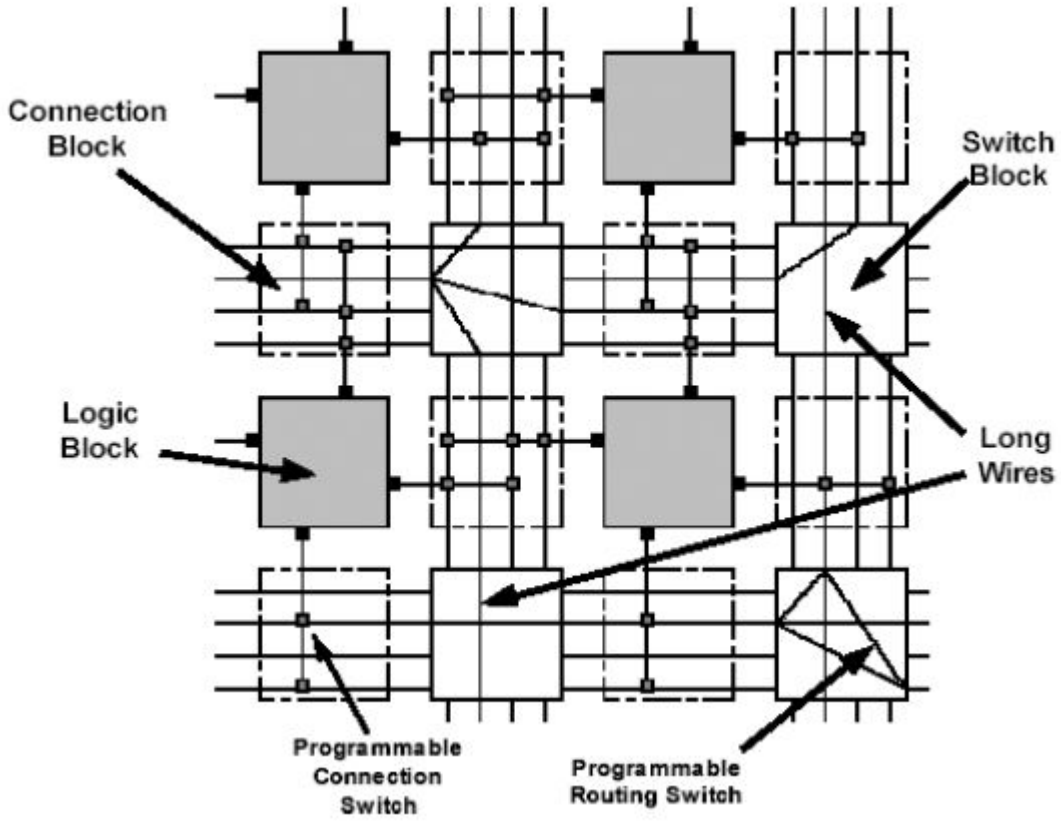


Fig II-3 : vue interne d'un fpga

II.c.3- Programmation

Il existe plusieurs technologies de programmation des FPGA. Dans le but de réduire l'encombrement de la logique destinée à la programmation, et d'accélérer le fonctionnement des circuits, on peut arriver à des circuits programmables une seule fois (*One-Time Programmable* ou *OTP*). Par contre, si la reprogrammabilité est importante, on peut opter pour des solutions plus coûteuses, mais reprogrammables à volonté. Il existe deux types majeurs de systèmes de programmation :

-Les FPGA anti-fusible sont extrêmement petits, et leur résistance est très faible. Ils permettent donc d'obtenir les circuits les plus compacts et les plus rapides. Ils sont bien entendu programmables une seule fois, ce qui les limite à une utilisation comme remplacement de pré diffusés standards.

Le point de connexion est de type ROM, c'est-à-dire que la modification des points est irréversible. Dans l'état initial, le fusible est présent et il n y a pas de contacte. Pour établir le contact, il faut détruire le fusible. [6]

Leurs avantage est le nombre important des points de connexion a cause de la surface réduite des fusibles.

Leurs inconvénient est la non réversibilité de la liaison. [6]

-Les FPGA à SRAM (*Static Random Access Memory*) permettent de reconfigurer les circuits à volonté, dans des temps relativement courts. La logique de configuration peut prendre jusqu'à 20 % de la surface totale, et la résistance des transistors est telle que ces circuits sont nettement ralentis par l'utilisation de cette technologie.

Seuls les circuits à base de SRAM nous intéressent, puisque le modèle des Mémoires Actives Programmables suppose l'utilisation de ces circuits comme coprocesseurs versatiles.

La reprogrammation en temps réel des circuits, provoquant la reconfiguration du coprocesseur est un domaine de recherche de plus en plus important. Les FPGA standards à base de SRAM comme les Virtex et les Spartan de Xilinx se reconfigurent en quelques dizaines de millisecondes. On peut envisager d'utiliser une PAM (Programmable Active Memory) en temps partagé entre différents processus, ou pour obtenir du silicium virtuel en appliquant plusieurs configurations au même circuit.

Ce type d'applications demande des temps de commutation extrêmement courts, et il est nécessaire de procéder à des adaptations, comme l'adjonction de plusieurs couches de mémoire, servant de mémoire tampon de configuration. De même, on peut tirer profit de la reconfiguration partielle d'un FPGA, en changeant des morceaux de configuration en fonction du besoin. On gagne ainsi en flexibilité, et en temps d'accès. Nous fournirons les étapes de développement des FPGA :

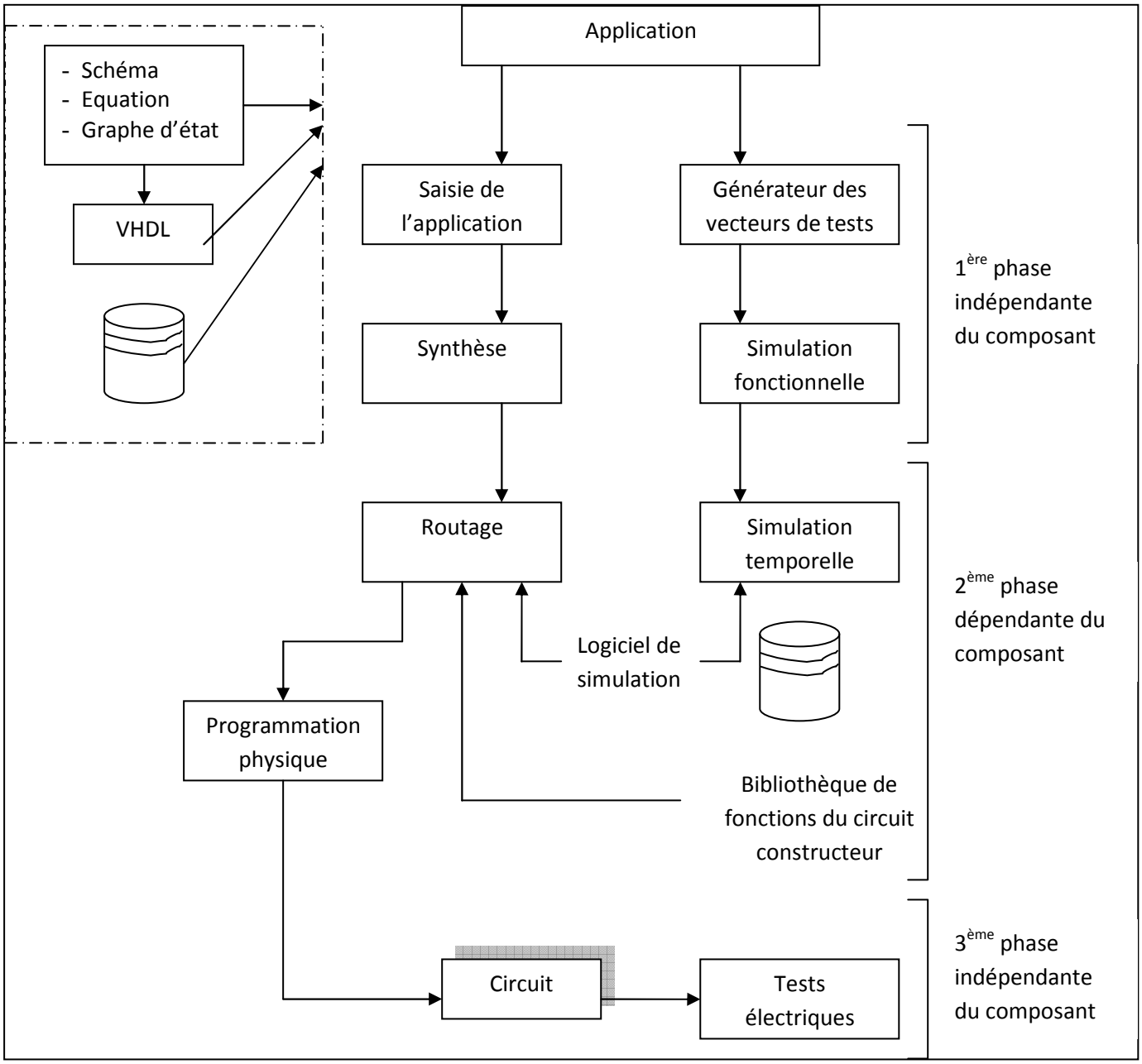
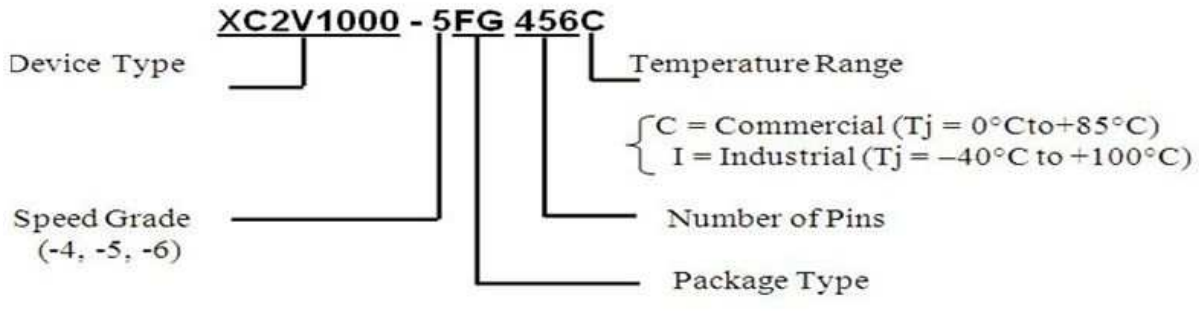


Fig : II-4 étapes de développement des FPGA

II.c.3- Nomenclature des circuits FPGA :

La nomenclature des FPGA correspond à ce qui suit :



Cet exemple est la nomenclature de Vrtex 2 de la companie Xilinx qui possède 1000 * 1000 portes (1 Million) et 456 pins, type de package FG et destiné aux applications commerciales**.**

II.d- Les éléments de différentes architectures :

Il existe plusieurs constructeurs des FPGA : Xilinx, Altera, Cyprex… Mais la firme qui domine le domaine c'est bien Xilinx qui produit actuellement les FPGA de version Virtex et Spartan. La société Xilinx a mis sur le marché des FPGAs Virtex et Xilinx de version 6. Quelle que soit l'architecture choisie, les éléments constitutifs d'un FPGA sont toujours à peu près les mêmes. Chaque fabricant ayant ses variantes par rapport à un autre. Nous pouvons citer un certain nombre de ces éléments :

II.d.1- Les éléments logiques :

Ce sont les briques de bases de tout FPGA. Grâce à leur configuration on peut réaliser dans ces blocs toutes les opérations de logique combinatoire (dans la limite d'un nombre d'entrées). Ces blocs ont souvent la même constitution et cela malgré la différence de fabricants et d'architectures. Ils sont généralement constitués d'une ou plusieurs LUTs (Look Up Table) qui contiennent, après configuration, la table de vérité de la fonction logique qu'elles doivent réaliser ou alors un ensemble de valeurs qui sont mémorisées comme dans une ROM. La taille des LUTs est généralement de 4 entrées, et de nombreuses études sur la taille des LUTs ont donné des résultats dans ce sens. Les LUTs sont généralement suivies d'un registre de sortie, ce qui permet de synchroniser, si nécessaire, la sortie sur une horloge. La plupart des blocs logiques de bases sont munis d'une chaîne de propagation rapide de retenue, afin de former de petits additionneurs rapides.

II.d.2- Les éléments de mémorisation :

Pour des applications plus importantes, les FPGAS demandent souvent des capacités de stockage (citons en exemple les applications du traitement d'images). La nécessité d'intégrer des blocs de mémoires directement dans l'architecture des FPGAs est vite devenue cruciale. De cette façon les temps d'accès à la mémoire sont diminués puisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit.

II.d.3- Les éléments de routages :

Les éléments de routages sont les plus importants dans un FPGA. En effet, les ressources de routages représentent la plus grosse partie de silicium consommée sur la puce réalisant le circuit. Ces ressources sont composées de segments (de longueurs différentes) qui permettent de relier entre eux les autres éléments via des matrices de connexions. Le routage de ces ressources est un point critique du développement d'une application sur un FPGA, et les méthodes utilisées pour les ASICs ne sont plus tout à fait valables (du fait de la segmentation des ressources). Si ces éléments sont importants c'est parce qu'ils vont déterminer la vitesse et la densité logique du système. Il existe plusieurs possibilités pour interconnecter deux blocs.

II.d.4- Les éléments d'entrées/sorties :

Sur une carte, le circuit FPGA appartient à un système d'ensemble pouvant contenir des parties micro programmées comme dans le cas du "Co-Design". Le circuit doit donc avoir un lien avec son environnement, c'est le but des éléments d'entrées/sorties. Ceux-ci peuvent bénéficier de protections, de buffer ou d'autres éléments permettant la gestion des entrées et des sorties. En particulier, il est à noter que les circuits actuels proposent différentes normes pour les niveaux d'entrées et de sorties (par exemple : LVTTL 2 - 54mA, PCI 5V, PCI 3.3V, HSTL…) qui par configuration peuvent êtres choisis afin de s'adapter à l'environnement.

II.d.5- Les éléments de contrôle et d'acheminement des horloges :

Il paraît évident que dans tout système électronique relativement important, il faut disposer d'horloges et qu'elles sont souvent d'une importance capitale pour le bon fonctionnement du système. De ce fait, les FPGAs sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routages spécialement adaptées au transport d'horloges sur de longues distances.

II.e- Les familles des FPGAs de Xilinx :

La tendance des dernières générations est de cibler certains créneaux porteurs du marché, comme la solution "bas coût" ou, à l'opposé la solution "haute performance" : la taille, le type et le nombre de cellules qui varient suivant les familles de composants. Ainsi, la panoplie présentée par le fabricant montre sa volonté de couvrir tous les segments de marché. Les nouveaux produits de Xilinx sont les VIRTEX 6 ET les SPARTAN 6. Ces deux composants offrent une très grande performance avec un nombre important de ressource disponible et un grand nombre de bus de connexion avec le monde externe.

II.e.1- Etude de la carte FPGA VIRTEX II du Xilinx :

Dans ce travail, nous allons utiliser la carte FPGA de Xilinx Virtex-II **XC2V1000-4FG456C** qui est représentée dans les figures II-5 et II-6. Cette carte servira à implémenter un réseau de neurone. Nous commençons son étude par une présentation des parties constituant la carte Virtex-II.

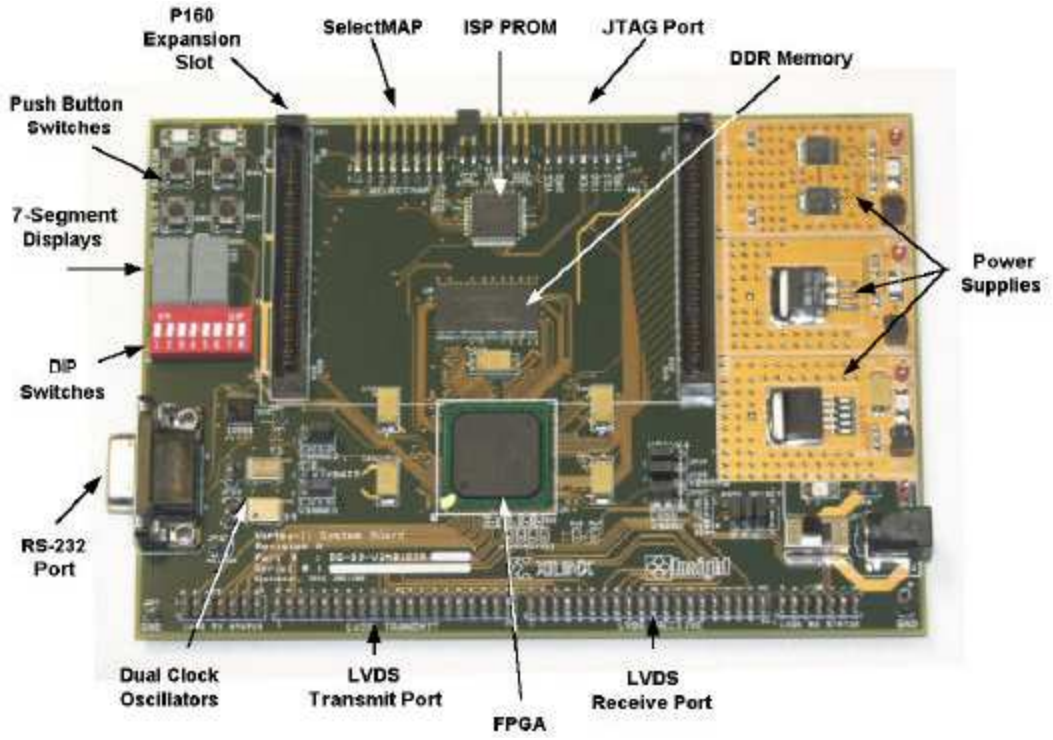


Figure II-5 : vue externe de la carte

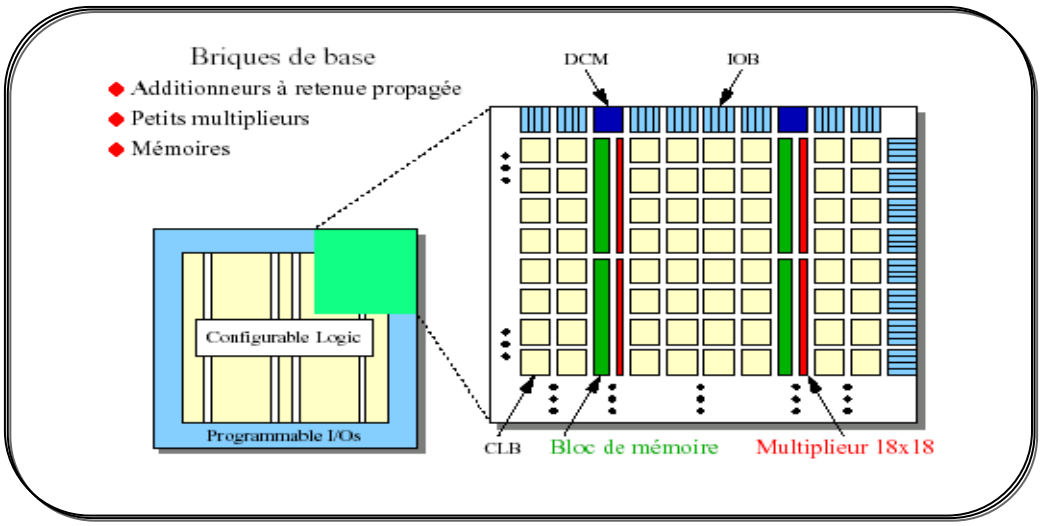


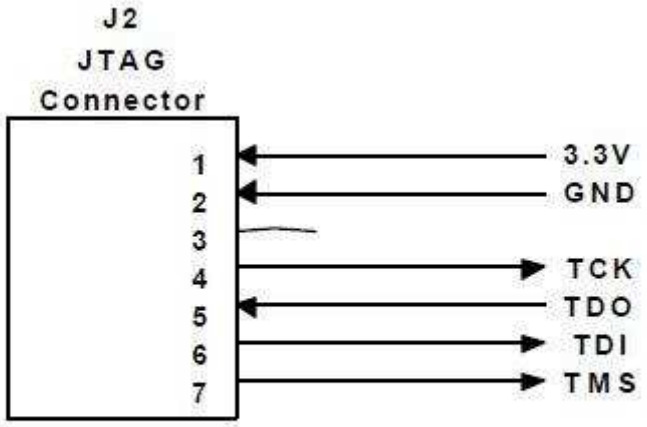
Figure II-6 : Architecture interne de la famille VIRTEX-II

II.e.1.1- Les différents composants de la carte :

On peut résumer les principaux composants de la carte comme suit :

- Elle contient 1 million de portes dans un package de type FG avec 456 pins.
- Elle intègre une mémoire DDR de 16 M *16 et une ISP PROM.
- Elle possède deux clock de 100 M et 24 M contrôlés par des jumpers (respectivement J24 et J23) de plus elle contient un soket pour ajouter d'éventuel oscillateur.
- 2 afficheurs 7 segments pour les données.
- Une LED pour le test qui correspond à la pin A9.

- Un Switch et des boutons poussoirs pour le contrôle.
- De port de connexion RS232.
- Le port JTAG qui sert à programmer In Situ Programming (ISP) la PROM et le FPGA. Le J2 JTAG connector est représenté dans la figure suivante :



figurell.7 – J2 JTAG Connector

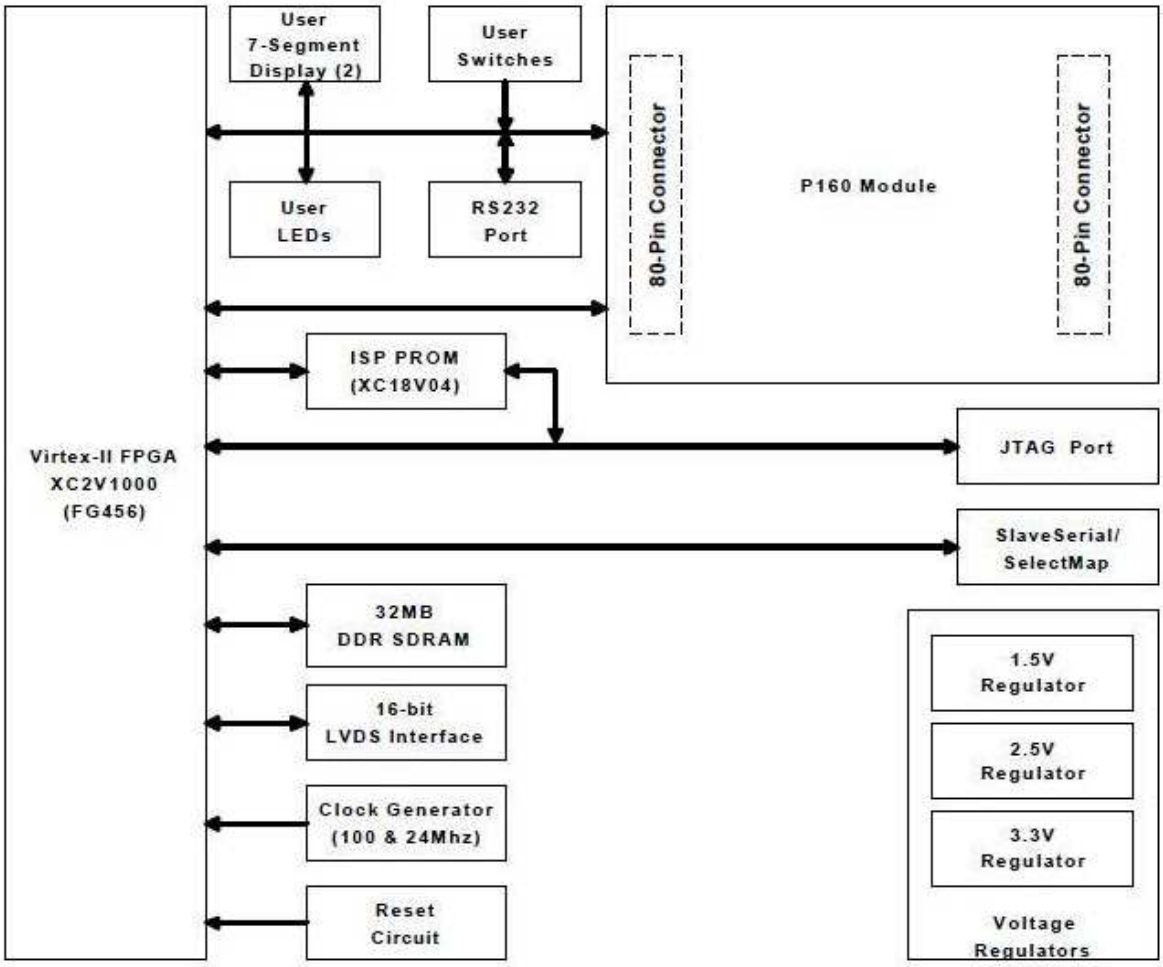
- Port d'expansion pour connecter une carte esclave si on veut augmenter les ressources.
- Un bloc d'alimentation pour les différents composants.
- Des dizaines de jumpers pour commander les différents composants. En suivant la documentation fournie par le constructeur on modifie les jumpers selon nos besoins.
- Des outils arithmétiques intégrés comme les multiplieurs, les multiplexeurs et les registres…

La carte Virtex II offre alors une très grande gamme de solution pour les applications de télécommunication, réseau, vidéo et les DSP.

Ce tableau donne quelques nombre sur la structure interne de la Virtex II:

		CLB (1 CLB = 4 slices = Max 128 bits)				SelectRAM Blocks			
Device	System Gates	Array Row x Col.	Slices	Maximum Distributed RAM Kbits	Multiplier Blocks	18-Kbit Blocks	Max RAM (Kbits)	DCMs	Max I/O Pads(1)
XC2V1000	1M	40 x 32	5,120	160	40	40	720	8	432

La figure suivante résume en général toutes les parties de la carte :



figurell.8 – Virtex-II System Board Block Diagram

D'après la figure II-6, on définit les parties suivantes de la carte :

II.e.1.2- Les différents blocs de la carte :

II.e.1.2.1- les éléments d'entrée/sortie programmable (IOB ou Input/Output Block) :

Ils constituent l'interface entre les bornes du circuit et les CLB. Le dispositif de routage VersaRing offre les ressources nécessaires à l'interconnexion des CLB aux IOB. Nous pouvons ainsi modifier le système implanté sur le FPGA sans interférer avec l'attribution des bornes. Cette caractéristique s'avère importante si nous souhaitons développer des nouvelles versions d'un produit tout en conservant la compatibilité au niveau du boîtier.

II.e.1.2.2. Les éléments logiques (CLB ou Configurable Logic Block) :

Composés de quatre cellules logiques (LC ou Logic Cell) réparties en deux tranches identiques **Figure II.6**, ils servent à construire les circuits numériques implantés sur le FPGA. Chaque LC contient essentiellement :

1. Les éléments logiques configurables (CLBs) :

C'est l'unité fondamentale du bloc logique qui fournit des éléments utilitaires pour la logique combinatoire et la logique synchrone, y compris les éléments du stockage de base : buffers à 3 états à associer avec chaque CLB. Les CLBs incluent quatre parties identiques appelées SLICE et deux buffers à 3 états.

Chaque SILICE est équivalente et contient :

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

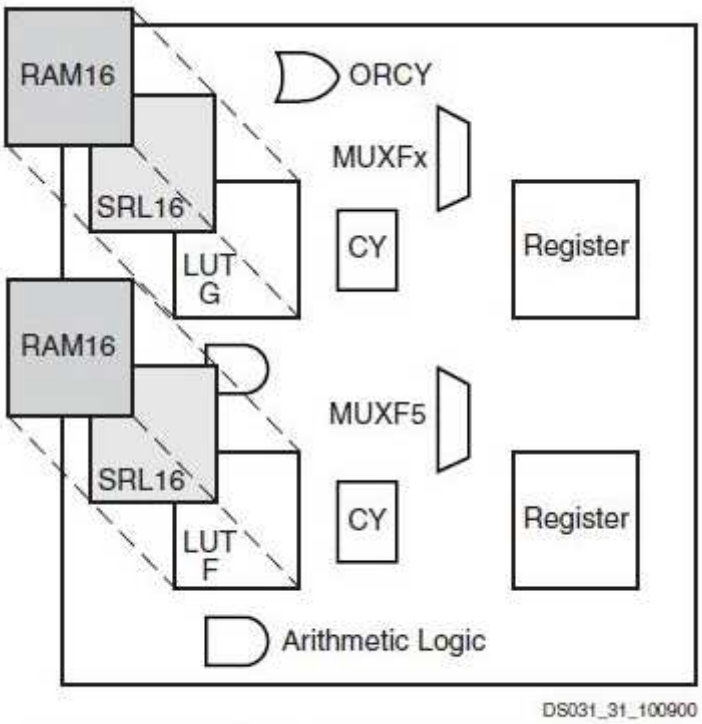


figure II.9: Virtex II Slice Configuration

2. Les elements mémoires (Select RAM) :

Ils possèdent des signaux d'initialisation (set et reset) synchrones ou asynchrones. Le bloc Select RAM produit de large élément de stockage (18 Kbit), programmable de 16K x 1 bit à 512 x 36 bits et peut être en deux blocs RAM (dual-port RAM).

3. Les blocs Multiplieurs :

Les blocks multiplieurs sont des multiplieurs de18x18 bits. Le circuit VIRTEX-II incorpore une grande densité de blocs multiplieurs. Chaque multiplieur peut être associé au bloc Select RAM ou peut être utilisé indépendamment, voir **Figure II.6**.

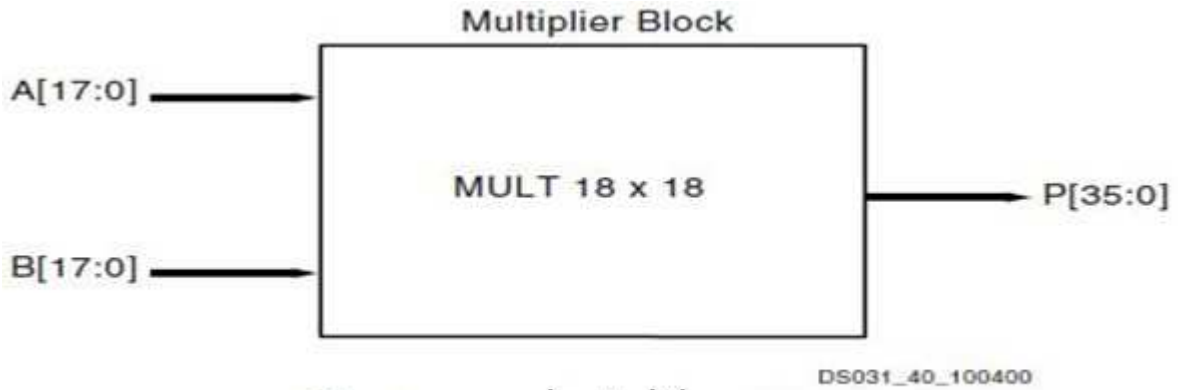


Figure II.10 un element multiplieur

4. Les blocs DCM (Digital Clock Manager) :

Le DCM produit le nouveau système d'horloges (soit intérieurement ou extérieurement au FPGA), il produit une large gamme de fréquences de l'horloge de multiplication et même la division, le bloc logique programmable contient plus de 12 DCM.

Par exemple, CLK2X et CLK2X180 doublent la fréquence de l'horloge tandis que CLKDV devise la fréquence sur des nombre :1.5, 2, 2.5, 3……..,7.5, 8, 9……16.

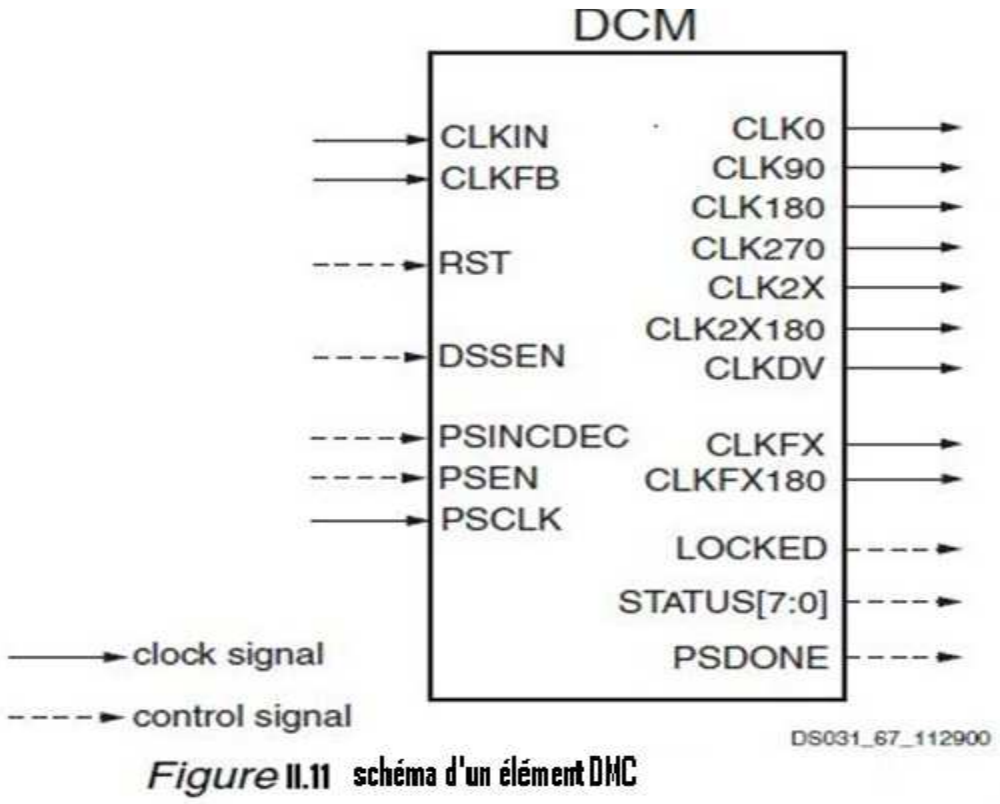


Figure II.11 schéma d'un élément DMC

Pour la construction de la Virtex II, Xilinx a utilisé les technologies de pointe 0.15 µm / 0.12 µm CMOS à 8 couches de métallisation, le processus et l'architecture Virtex-II sont optimisés pour une grande vitesse d'horloge avec une faible consommation d'énergie. Elle combine entre la flexibilité dans l'intégration et la densité d'intégration plus de 10 millions de portes logiques pour les dernières versions.

II.f- Le langage VHDL :

Développé dans les années 80 aux États-Unis, le langage de description VHDL est devenu une norme IEEE numéro 1076 en 1987. Révisée en 1993 pour supprimer quelques ambiguïtés et améliorer la portabilité du langage, cette norme est vite devenue un standard en matière d'outils de description de fonctions logiques.[8][9]

II.f.1- Pourquoi un langage de description ?

On utilise le langage VHDL pour :

- concevoir des ASIC,
- programmer des composants programmables du type PLD, CPLD et FPGA,
- concevoir des modèles de simulations numériques ou des bancs de tests.

L'électronicien a toujours utilisé des outils de description pour représenter des structures logiques ou analogiques. Le schéma structurel que l'on utilise depuis si longtemps et si souvent n'est en fait qu'un outil de description graphique. Aujourd'hui, l'électronique numérique est de plus en plus présente et tend bien souvent à remplacer les structures analogiques utilisées jusqu'à présent. Ainsi, l'ampleur des fonctions numériques à réaliser nous impose l'utilisation d'un autre outil de description. Il est en effet plus aisé de décrire un compteur ou un additionneur 64 bits en utilisant l'outil de description VHDL plutôt qu'un schéma. Le deuxième point fort du VHDL est d'être "un langage de description de haut niveau". En fait, un langage est dit de haut niveau lorsqu'il fait le plus possible abstraction de l'objet auquel ou pour lequel il est écrit. Dans le cas du langage VHDL, il n'est jamais fait référence au composant ou à la structure pour lesquels on l'utilise. Ainsi, il apparaît deux notions très importantes :

- **portabilité** des descriptions VHDL, c'est-à-dire, possibilité cibler une description VHDL dans le composant ou la structure que l'on souhaite en utilisant l'outil que l'on veut (en supposant, bien sûr, que la description en question puisse s'intégrer dans le composant choisi et que l'outil utilisé possède une entrée VHDL). On parle alors de portabilité vis-à-vis des circuits et vis-à-vis des logiciels de conception. Le premier aspect se traduit par une indépendance par rapport au circuit utilisé, on peut changer de composant au milieu de développement. Le deuxième aspect de la portabilité concerne les outils de développement. Le même programme est accepté par des logiciels de la CAO (Conception Assistée par Ordinateur) très divers. [10][11]
- **conception de haut niveau**, c'est-à-dire qui ne suit plus la démarche descendante habituelle (du cahier des charges jusqu'à la réalisation et le calcul des structures finales)

mais qui se “limite” à une description comportementale directement issue des spécifications techniques du produit que l'on souhaite obtenir.[3]

Les différentes étapes de développement d’un circuit sont montrées dans la figure suivante :

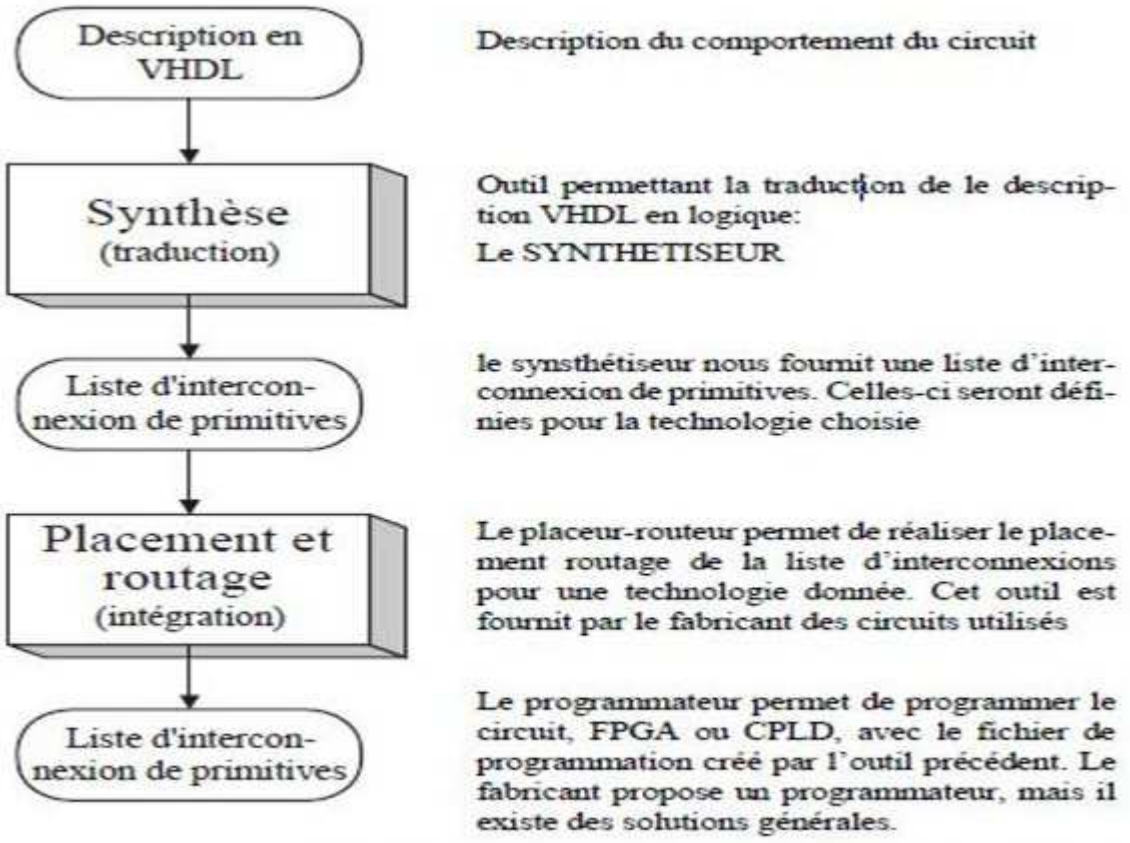


figure II.12: différentes étapes pour la réalisation d'un circuit

II.f.2- Les descriptions VHDL :

Il existe plusieurs descriptions possibles pour un seul composant :

- Description structurelle : elle consiste à décrire la mise en œuvre et l’interconnexion des composants du circuit en utilisant des signaux internes pour connecter les pins l’une a l’autre.
- Description schématique : elle consiste à donner le schéma du circuit en porte logique el les connectant pour former le circuit.
- Description comportementale : elle consiste à définir les équations booléennes qui définissent les différents signaux.
- Description par flot de données : elle consiste à décrire le circuit par des équations en utilisant les opérateurs logiques.

g- Conclusion :

Dans ce chapitre nous avons vu les FPGAs et la carte Virtex II en particulier. Ensuite on a fait un aperçu surfacique sur le langage de description VHDL. Dans le chapitre suivant nous nous intéressons à l'implémentation des réseaux de neurone sur FPGA. Nous posons les problèmes et nous exposons les solutions apportées.

Chapitre III : l'implémentation

III.a- Introduction :

L'implémentation d'un réseau de neurones est aujourd'hui plus simple qu'elle était il y a quelques années. L'électronique numérique qui offre des composants simple, robuste et non onéreux a facilité la tâche. Dans ce chapitre nous allons faire une étude générale sur les différentes méthodes d'implémentation des réseaux de neurones. Nous abordons quelques principes relatifs à cette implémentation comme le parallélisme.

III.b- Conception des ANNs :

Lors de la conception d'un réseau de neurones, il faut prendre en considération les paramètres suivants :

- Le parallélisme qui permet un traitement performant.
- Les performances relatives par rapport à la complexité des connections synaptiques.
- La flexibilité ou l'adaptabilité du circuit
- La surface du silicium

La notion de parallélisme dans les ANNs :

Le parallélisme peut se manifester sur 3 niveaux :

- Entre les couches du réseau de neurones : cela signifie qu'au moins un multiplieur doit être utilisé pour chaque couche.
- Au niveau des nœuds ou neurones : dans ce cas, un seul neurone suffit.
- Au niveau des connexions synaptiques : c'est le plus haut degré de parallélisme souhaité dans un réseau de neurones. Pour atteindre ce parallélisme, il faut prévoir un multiplieur pour chaque connexion. Cependant, l'utilisation d'un grand nombre de multiplieurs pour un grand nombre de connexions synaptiques est une pénalité sévère pour les circuits digitaux en particulier les circuits FPGAs à cause de leurs ressources limitées.

III.c- Architectures du neurone :

Suivant la logique précédente et selon le concept du parallélisme, on peut distinguer 3 architectures:

III.c.1- Serial Processing (SP) : cette architecture est la plus simple à réaliser. Elle ne nécessite pas beaucoup de ressources. L'utilisation d'un accumulateur après la multiplication des entrées, nous évite l'utilisation d'autre composant multiplieur. Le principe consiste à multiplier deux entrées (Wi et Xi) ensuite additionner le résultat obtenu pour être accumuler avec l'ancienne valeur déjà accumulée. Une fois toutes les valeurs calculées, le résultat sera envoyé à la fonction de transfert. La figure suivante illustre les différents blocs.

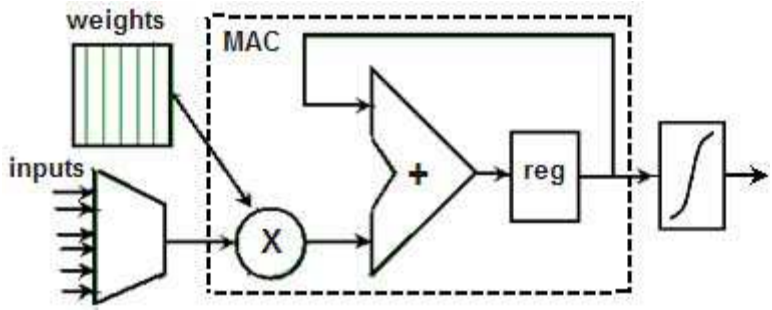


Figure III.1 : modèle d'un neurone avec SP

III.C.2- Partial Parallel Processing (PPP)*:* cette technique utilise des multiplieurs suivis par des additionneurs comme le montre la figure suivante. On utilise autant de multiplieurs que de paires (Wi et Xi). Le résultat final sera envoyé au bloc fonction de transfert.

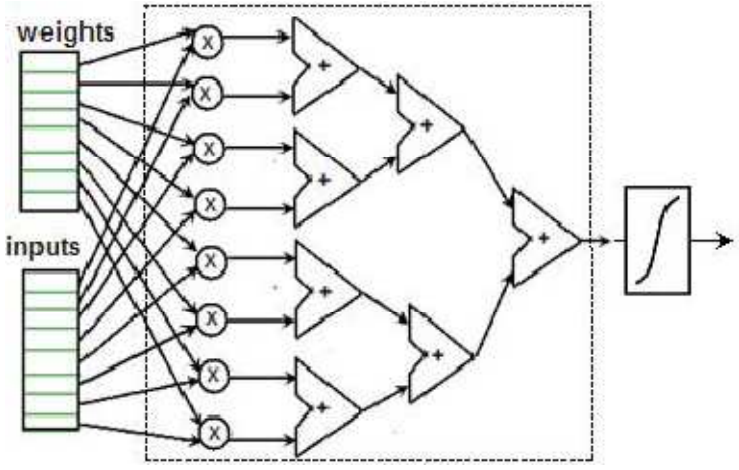


Figure III. : Modèle PPP

III.c.3- Full Parallel Processing (FPP): Cette technique utilise pour chaque paire (Wi et Xi) un multiplieur (MUL), comme dans le model PPP, mais on remplace touts les additionneurs par un seul de nombre d'entrée convenable. La sortie de cet additionneur sera connectée à l'entée de la fonction de transfert comme le montre la figure suivante :

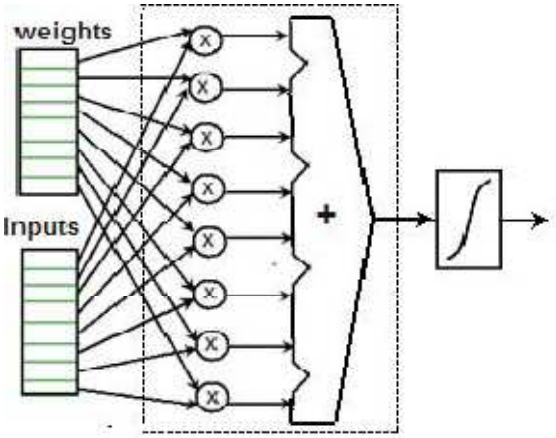


Figure III. :Modèle FPP

Chaque technique présente des avantages et des inconvénients. Ce qui fait que le choix de la technique à utiliser dépend des ressources disponibles, de l'utilisation de notre réseau et le besoin de parallélisme.

Dans notre cas nous optons pour une architecture très simple en l'occurrence la SP qui sera développée dans la partie pratique sachant que :

- Pour un même neurone, un seul circuit MAC est utilisé pour calculer la somme des produits.
- Chaque circuit MAC possède sa propre ROM où sont mémorisés les poids synaptiques.
- Pour la même couche un calcul parallèle des neurones est réalisé.
- Le calcul entre couches se fait en série

Comme nous pouvons le constater, l'architecture résultante :

- Intègre un haut degré de parallélisme
- Possède une simplicité de conception caractérisée par une répétition de cellules simples

III.d- Les logiciels utilisés :

III.d.1- ISE 9.2 : Cet logiciel développé par la société Xilinx sert pour la programmation des composants de Xilinx : les FPGAs et les CPLDs. A l'heure actuelle la firme propose sur le marché la nouvelle version 12. Xilinx produit 2 gammes de composants : les Virtex qui sont dans leur version 6 et les Spartan eux aussi en version 6. Cet outil nous permet de faire plusieurs applications en commençant par la saisie du code vhdl (nom.vhdl) pour faire ensuite la vérification de la syntaxe, le fichier des contraintes et le fichier testbench. Il possède aussi un simulateur intégré ISE Simulator qui peut être utilisé pour simuler le circuit mais il est très limité. Pour cela nous allons utiliser un autre simulateur de MentorGraphics appelé ModelSim. Nous allons utiliser l'outil ISE 9.2 pour faire l'implémentation qui a 3

étapes : Translate, Mape et enfin Place and Rote. Durant chacune des étapes précédentes, l'outil produit un fichier spécifique. Le dernier fichier est d'extension « .bit » (nom.bit) qui sera implémenté sur le FPGA sans intervention d'un autre matériel sauf un câble de connexion qui peut être dans notre cas soit un câble USB soit un câble JTAG.

III.d.2- ModelSim PE Student Edition 6.6b : c'est l'outil de simulation développé par la société Mentor Graphics pour simuler les circuits numériques. Il est gratuitement téléchargeable sur le net pour l'utilisation éducative, sur le site de Mentor Graphics. Il nous permet de valider le circuit avant de passer à l'implémentation sur circuit. Son utilisation est très simple, il suffit juste de le configurer sur l'outil ISE dans les propriétés du projet.

III.e- Programmation et implémentation sur FPGA Virtex II :

Les étapes de l'implémentation sur un FPGA sous ISE 9.2 sont :

- création d'un projet
- Ecriture du programme ensuite enregistrement
- Vérification de la syntaxe et correction d'éventuelles erreurs
- Création des contraintes temporelles du composant à régler selon le composant utilisé et la fréquence
- Assignement des pins
- Simulation du module après création d'un module testbench
- Vérification par ISE des contraintes temporelles si ok
- Etape implement and design : 3 étapes
  - Translate : c'est la traduction du fichier d'extension nom .edif représentants la netlist des I \0 en un fichier nom. ngd compréhensible par l'étape suivante
  - Map : c'est l'étape où l'outil ISE répartie d'une manière non finie et désigne les blocs utilisé indépendamment du type de FPGA. Il donne les rapports et les pourcentages qui peuvent être > à 100%.
  - Place and route : dans cette étape, ISE applique sur la structure de FPGA utilisé le placement et le routage. Si une insuffisance des ressources est constatée l'étape ne sera pas faite. Si l'étape est réussie, on passe au
- Chargement de fichier (non.bit) à l'aide d'un câble parallèle JTAG. On connecte la carte FPGA au micro. Ou commence l'implémentation.

Si nous voulons, par exemple, ajouter d'autre contraintes spatiales on doit agir à l'étape d'assignement des contraintes par exemple spécifier d'autres entrées\sorties (I\0) comme : switchs, leds….

Nous allons agir de cette façon pour affecter des pins à des I\0 ciblées. Dans le volet contraintes (constraints) nous allons choisir « Assigne package pins » dont on procède à l'affectation des pins selon nos besoins.

Lorsque nous chargeons le fichier (nom. Bit) pour être implémenté physiquement sur le composant FPGA, l'outil ISE nous offre la possibilité d'utiliser d'autres composants comme des PROM ou des CPLD pour stoker et sauvegarder le programme général si on ne veut pas le perdre après coupure de l'alimentation. Pour les Virtex II, il n'y a pas de CPLD intégré, il y a juste une EPROM qu'on peut charger par le même câble en choisissant convenablement la position des jumpers.

III.f- Etude évolutive de fichier nom.vhdl durant l'implémentation :

Les différentes extensions de chaque fichier engendré à chaque étape sont données dans ce schéma :

ETAPE	NOM du FICHIER et son EXTENSiON
Saisie du code	Nom.vhdl
Synthèse	Nom.edif (Electric Data Interchange Format)
Implementation : - translate -Mapping -Place and Route	Nom.ngd (Native Generic Database) Nom.ncd (Native Circuit Description) Nom.bit
Configuration	Nom.bit
Fichier des contraintes	Nom.ucf (User Constraints File)
Fichier testbench	Nom.tbw (TestBench Wave)

III.g- Le chargement du fichier sur le composant FPGA :

Le chargement du fichier (nom.bit) sur le composant programmable se fait à travers un câble JTAG de norme J2 JTAG. Le chargement du programme se fait selon le schéma suivant :

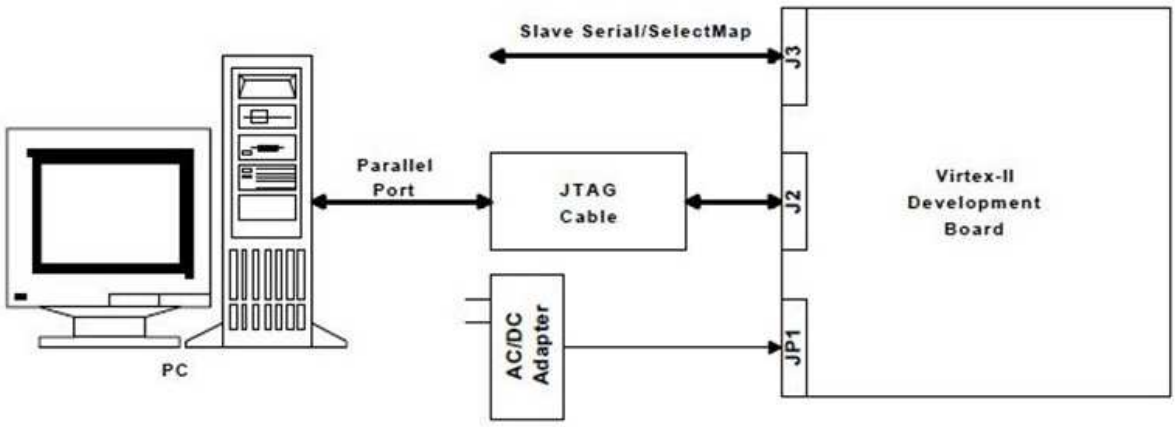


Figure III.4 : Chargement du programme

L'alimentation de la carte doit être faite par un adaptateur 5V avant de charger le fichier de bitstream (nom.bit).

Le câble JTAG comme le montre la figure III.4 configure le FPGA mais il peut aussi être utilisé pour configurer la mémoire ISP EPROM. Il est connecté au PC par un port parallèle. Lors de chargement de ce FPGA, nous devons respecter 2 conditions :

1- Le branchement des fiches du câble JTAG doit être respecté selon la documentation.
2- On utilise le processus iMPACT process pour charger le bit stream sur le FPGA.

Dans notre cas avec la carte FPGA Virtex II, l'outil ISE nous demandera pendant le chargement si nous voulons associer la mémoire ISP PROM pour la configurer, de même s'il y avait un CPLD intégré sur la carte, alors on choisit non en on clique sur BYPASS pour sauter cette étape.

III.h- Programmation des neurones, couches et réseau :

Nous allons utiliser une description structurelle de ces 3 composants car elle offre plus de simplicité au code et à l'utilisateur. Dans cette partie nous exposons des prototypes de codes pour chaqu'un.

III.h.1- Le neurone : Le programme suivant nous donne un exemple d'un neurone à fonction de seuil.

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;  use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;   use IEEE.STD_LOGIC_SIGNED.ALL;

entity neurone is

en : in  STD_LOGIC;  rst : in  STD_LOGIC;  x : in  STD_LOGIC_VECTOR (7 downto 0);

w : in  STD_LOGIC_VECTOR (7 downto 0); resultat : out  STD_LOGIC);  end neurone;

architecture structural of neurone is

signal s3   :  std_logic_vector (15 downto 0);   signal s4   :  std_logic_vector (15 downto 0);

component ACCU port (d : in     std_logic_vector (15 downto 0);  CLK : in std_logic ;   RST : in std_logic ;  EN  : in std_logic ;

Sortie     : out    std_logic_vector (15 downto 0)  ); end component;

component multi port ( a1 :in std_logic_vector(7 downto 0);    b1 :in std_logic_vector(7 downto 0);

p : out std_logic_vector(15 downto 0) );  end component;

component comparateur port (AC : in std_logic_vector(n-1 downto 0);     sortie_c :out std_logic ); end component;

begin

inst_Multi :                multi port map( a1 => x , b1 => w, p => s3);

inst_ACCU :  ACCU port map (d => s3, rst=>rst, clk => clk, en=> en, sortie => s4 );

inst_comparateur: comparateur port map ( AC=> s4,  sortie_c => resultat);     end structural;
```

III.h.2- La couche : Dans ce programme on procède à l'interconnexion des neurones entre eux pour réaliser la couche du réseau.

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;  use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;   use IEEE.STD_LOGIC_SIGNED.ALL;
entity couche is
generic(n: couche :=16 );    Port ( clk : in  STD_LOGIC;
en : in  STD_LOGIC;  rst : in  STD_LOGIC;  in1 : in  STD_LOGIC_VECTOR (7 downto 0);……
ini : in  STD_LOGIC_VECTOR (7 downto 0);
resultat1 : out  STD_LOGIC_vector(15 downto 0) );…. resultatl : out  STD_LOGIC_vector(15 downto0));  end neurone;
architecture structural of neurone is
component Neurone1  port ( x1 :in std_logic_vector(7 downto 0);    w1 :in std_logic_vector(7 downto 0);
SORTIE1 : out std_logic_vector(15 downto 0) );  end component;……
component Neuronel  port ( xl :in std_logic_vector(7 downto 0);    wl :in std_logic_vector(7 downto 0);
SORTIE1 : out std_logic_vector(15 downto 0) );  end component;
Begin
Neurone1:Neurone1 map( x1=>in1,en=>en , ….) ;……
Neuronel:Neuronel map( xl=>inl,en=>en , ….) ;  end ;
```

III.h.3- Le réseau : De même, le programme de notre réseau regroupe les différentes couches développées précédemment.

```
entity reseau  is
   Port ( clk : in  STD_LOGIC;
en : in  STD_LOGIC;  rst : in  STD_LOGIC;  entrée1 : in  STD_LOGIC_VECTOR (7 downto 0);……entréei : in  STD_LOGIC_VECTOR (7 downto 0);
sortie1 : out  STD_LOGIC_vector(15 downto 0) );…. sortiel : out  STD_LOGIC_vector(15 downto0));  end neurone;
architecture structural of reseau is
component couche1  port ( in1 :in std_logic_vector(7 downto 0);    ini :in std_logic_vector(7 downto 0);
resultat1 : out std_logic_vector(15 downto 0);…… );  end component;……
component Neuronel  port ( inl :in std_logic_vector(7 downto 0);    inl :in std_logic_vector(7 downto 0);
resultat1 : out std_logic_vector(15 downto 0) ;……);  end component;
Begin     couche1:couche1 map( in1=>entree1,en=>en , ….) ;……..
            couchel:couchel map( inl=>entreel,en=>en , ….) ;     end
```

III.i- Conclusion :

Dans ce chapitre, nous avons abordé la problématique de l'implémentation en commençant par les différentes architectures et en passant par la notion de parallélisme pour finir avec les outils utilisés et l'étape de chargement sur le FPGA du code.

Dans ce qui suit nous allons entamer la partie pratique qui consiste en l'implémentation sur FPGA de type Virtex II de 3 neurones avec 3 fonctions de transferts différentes : à seuil, linéaire et sigmoïde.

A la fin de cette partie pratique, nous allons implémenter un réseau de neurones d'un capteur réalisé par [3].

Chapitre IV : la partie pratique

IV.a- Introduction :

Dans cette dernière partie, nous allons implémenter 3 neurones différents sur un FPGA. Le premier neurone est un neurone à fonction de seuil, il est le plus simple car il ne nécessite pas une fonction de transfert complexe. Sa sortie possède 2 états (1 ou 0).

Le deuxième neurone est un neurone avec une fonction de transfert linéaire, c'est-à-dire f(x)=x. celui-ci est aussi simple à réaliser.

Le dernier cas est le neurone avec une fonction de transfert non linéaire sigmoïde qui est la plus difficile à implémenter. Sa fonction mathématique est **: f(x)=1 / (1+e^{ax})**.Pour cela on utilise une approximation de cette fonction pour la numériser et l'implémenter ensuite.

La programmation et l'implémentation du réseau de neurones sur le FPGA Virtex2, seront faites en dernier pour utiliser les programmes de ces neurones dans la formation des couches du réseau.

IV.b- Le neurone à seuil :

Cet exemple a été réalisé durant le stage effectué au CDTA. Il a été implémenté sur un FPGA de la famille Spartan 3E. Nous allons montrer les schémas et les résultats de cette implémentation.

IV.b.1- Schéma du neurone : Pour réaliser ce circuit on procède à la programmation structurelle en définissant circuit par circuit, enfin on les relie par un seul programme structurel qui contient :

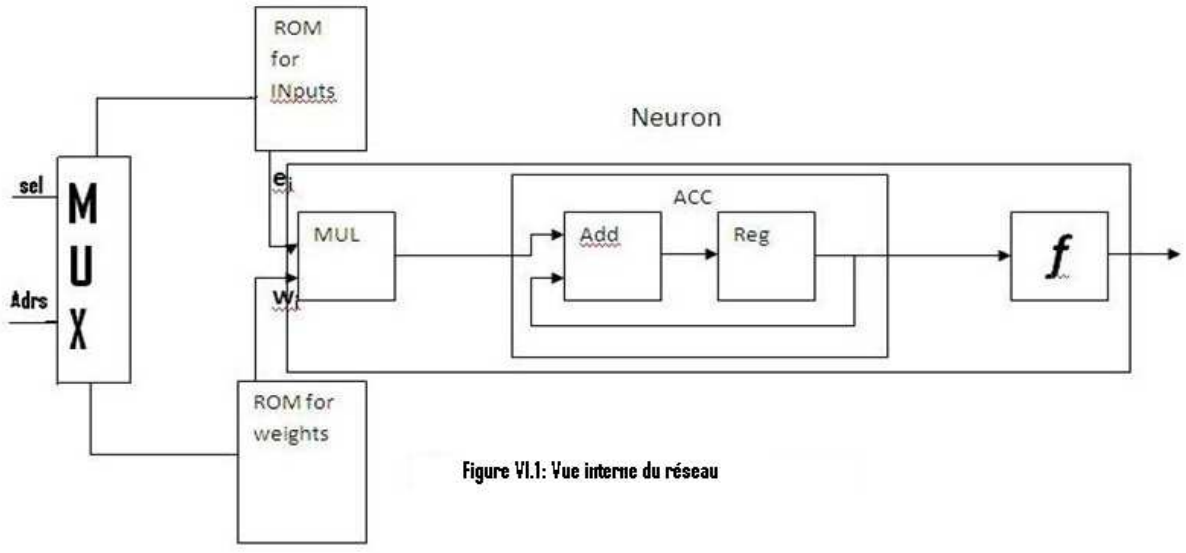


Figure VI.1: Vue interne du réseau

1-L'Accumulateur : il est formé de 2 composants, un additionneur et un registre. Leurs programmes sont donnés en annexe.

2-Le Multiplieur : il fait le produit entre les valeurs des poids (Wi) et les valeurs des entrées (Xi) pour envoyer le résultat a l'entrée de l'accumulateur comme mentionné dans la figure IV-1

3-La fonction de transfert : dans notre cas c'est une fonction de seuil qui nous rend 1 ou 0 en sortie.

L'ensemble de ses 3 composants constitue le neurone, sans les valeurs des poids et des entrées.

4-Les ROM des poids et des entrées : elles contiennent les valeurs Wi et Xi codées sur 8 bits.

5-Le Multiplexeur : il sert pour sélectionner les Wi et les Xi correspondants.

Pour bien comprendre le schéma du circuit nous l'avons divisé en deux blocs :le neurone et le bloc MUX-ROM

Nous présentons dans ce qui suit les différents composants du circuit avec leurs entrées, leurs sorties et la taille de chaque vecteur.

IV.b.2- La vue générale du circuit :

Les composants de circuit sont montrés dans la figure suivante :

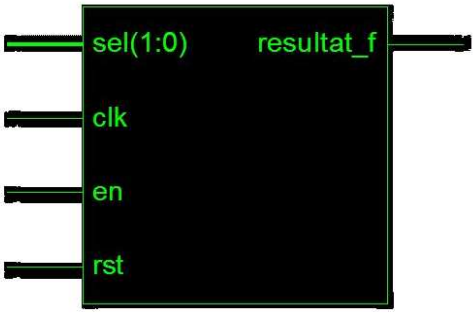


Figure VI.2 : Vue du circuit

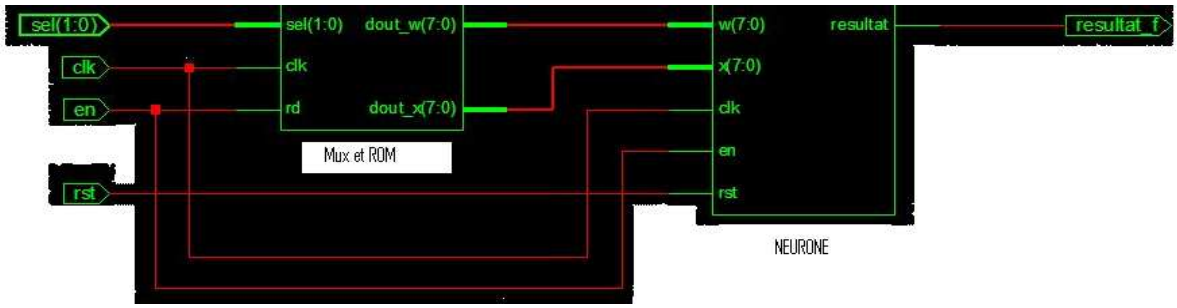


Figure VI.3 : Vue interne du circuit

Chaqu'un des 2 composants est constitué de plusieurs autres composants :

- **Les ROM et le MUX** : il est le bloc de sélection et de contrôle car à partir duquel on choisit les valeurs et on active le composant.

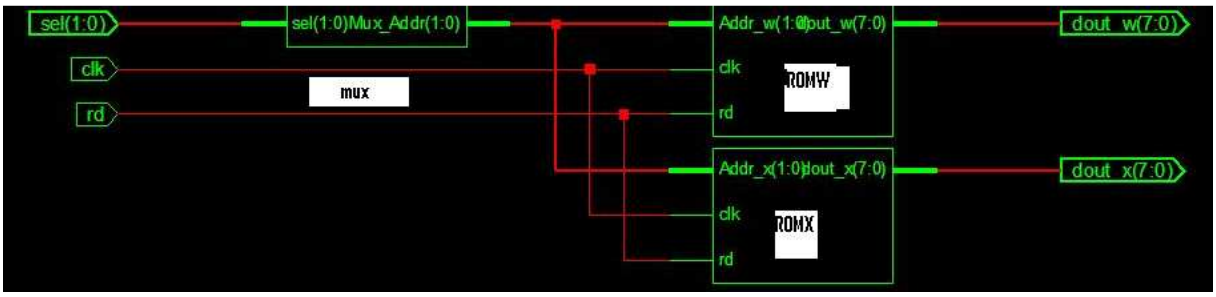


Figure VI.4 : Vue interne du bloc ROM-MUX

- **Le neurone :**

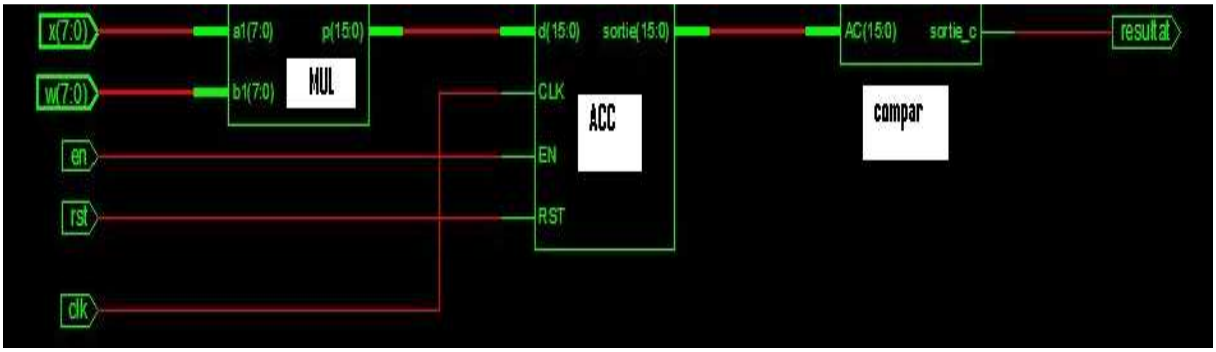


Figure VI.5 : schéma du neurone

L'accumulateur est de sa part constitué de 2 composants comme mentionné auparavant :

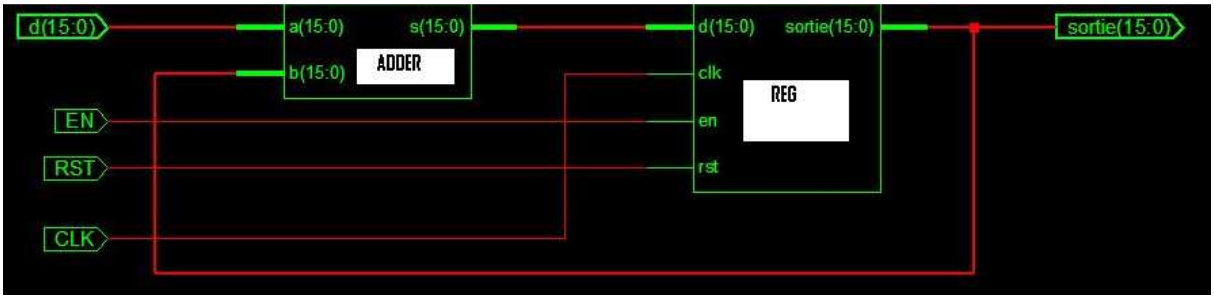


Figure VI.6 : schéma de l'accumulateur

L'étape importante pour l'implémentation est la création de fichiers de contraintes temporelles et spatiales. Pour cela on utilise dans la fenêtre du ISE l'icône User constraints et on assigne aux différents signaux la pin convenable en se basant sur le datasheet du FPGA. Le tableau suivant résume le fichier de contrainte (nom.ucf) utilisé lors de l'implémentation sur la SPARTAN 3E. On a utilisé 2 switchs pour la sélection, 1 switch pour le

reset et un autre pour le signal enable. Le résultat final est affiché par la led 0. Pour l'horloge clk on a utilisé celle de 50MHz.

Port Name	Port Direction	Location	Pad to Setup	Clock to Pad	
clk	INPUT	C9	N/A	N/A	50Mhz
en	INPUT	H18		N/A	SW2
resultat_f	OUTPUT	F12	N/A		led0
rst	INPUT	N17		N/A	SW3
sel<0>	INPUT	L13		N/A	SW0
sel<1>	INPUT	L14		N/A	SW1

Avant de passer à l'implémentation, on doit simuler le fonctionnement du circuit à l'aide de ModelSim. Pour cela on crée un fichier testbench pour donner des valeurs fixes aux différents vecteurs, ensuite on le simule.

Une fois le fichier testbench créé, il ne reste que la simulation. On choisit le simulateur souhaité dans notre cas nous avons utilisé le ModelSim .

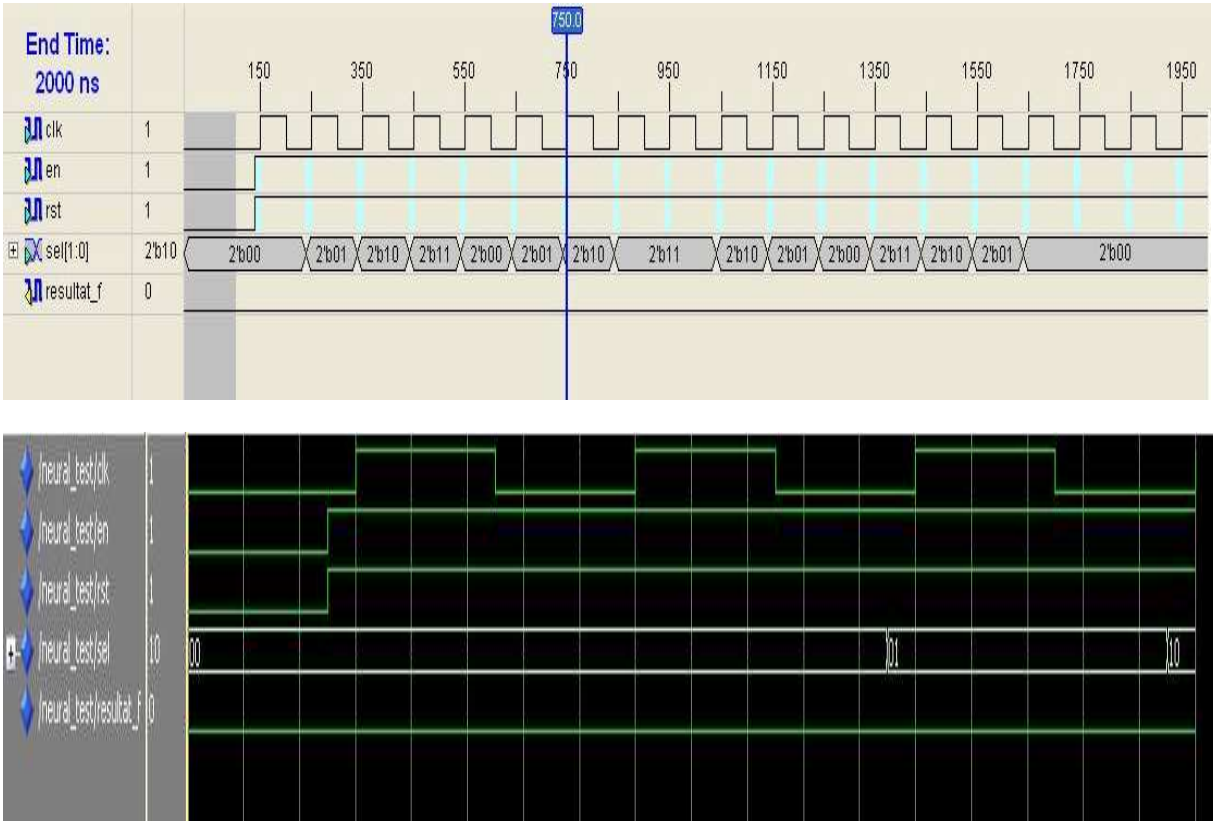


Figure VI.7 : Simulation du neurone

- **Résultats de synthèse avec l'outil XST sous ISE :**

Le tableau suivant résume la consommation en ressource disponible sur le composant :

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	17	9,312	1%
Number of 4 input LUTs	11	9,312	1%
Logic Distribution			
Number of occupied Slices	14	4,656	1%
Number of Slices containing only related logic	14	14	100%
Number of Slices containing unrelated logic	0	14	0%
Total Number of 4 input LUTs	24	9,312	1%
Number used as logic	11		
Number used as a route-thru	13		
Number of bonded IOBs	6	232	2%
IOB Flip Flops	1		
Number of GCLKs	1	24	4%
Total equivalent gate count for design	333		
Additional JTAG gate count for IOBs	288		

On remarque bien que le projet n'a consommé que 1% des ressources disponibles.

IV.c- Le neurone linéaire :

De même que le premier neurone, le neurone linéaire est construit à partir des composants simples interconnectés entre eux et décrit dans un programme structurel. Le neurone choisi possède 6 entrées de 18 bits chaqu'une, un signal de sélection « sel » sur 3 bits, un signal pour le reset « rst » et un autre pour l'horloge « clk ». Il a une sortie de 18 bits. Son schéma général est donné dans cette figure:

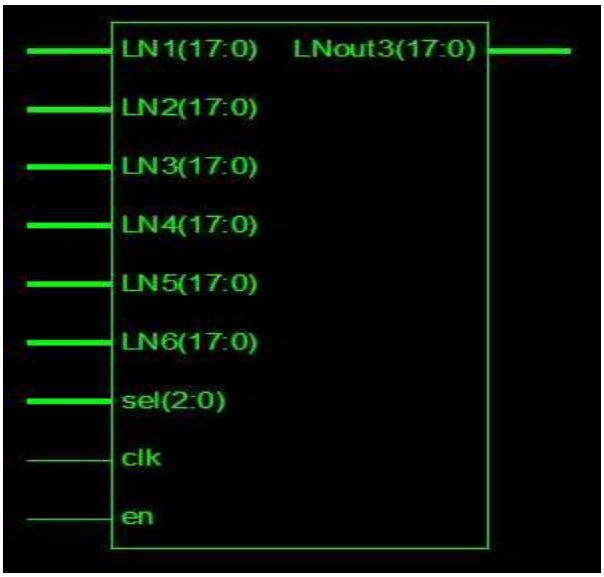


figure VI.8: Figure du neurone linéaire

Le choix des paramètres de neurone est fait de telle sorte qu'il sera utilisé pour la réalisation du réseau final.

Les composants qui constituent le neurone sont montrés dans la figure suivante :

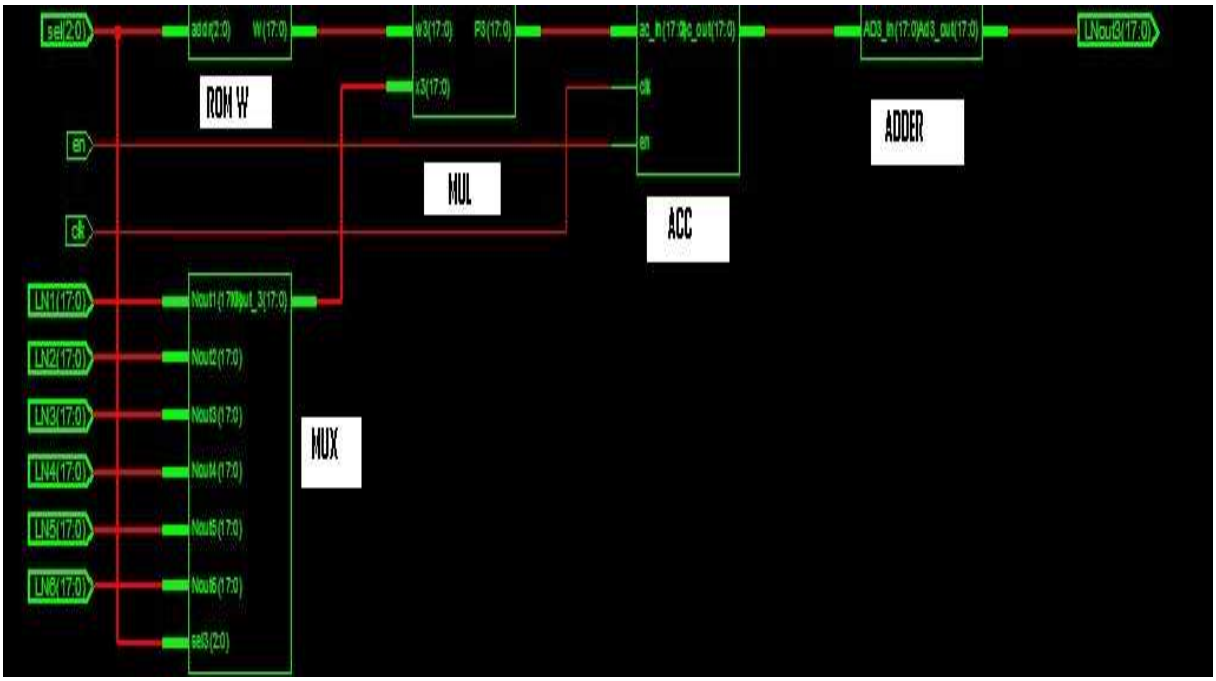


Figure VI.9 : Vue interne du neurone

Le rapport de la consommation des ressources disponibles sur le FPGA Virtex II est donné dans ce tableau :

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	18	10,240	1%
Number of 4 input LUTs	114	10,240	1%
Logic Distribution			
Number of occupied Slices	63	5,120	1%
Number of Slices containing only related logic	63	63	100%
Number of Slices containing unrelated logic	0	63	0%
Total Number of 4 input LUTs	124	10,240	1%
Number used as logic	114		
Number used as a route-thru	10		
Number of bonded IOBs	131	324	40%
Number of MULT18X18s	1	40	2%
Number of GCLKs	1	16	6%
Total equivalent gate count for design	5,083		
Additional JTAG gate count for IOBs	6,288		

Les résultats de la simulation et le fichier testbench sont donnés ci-dessous :

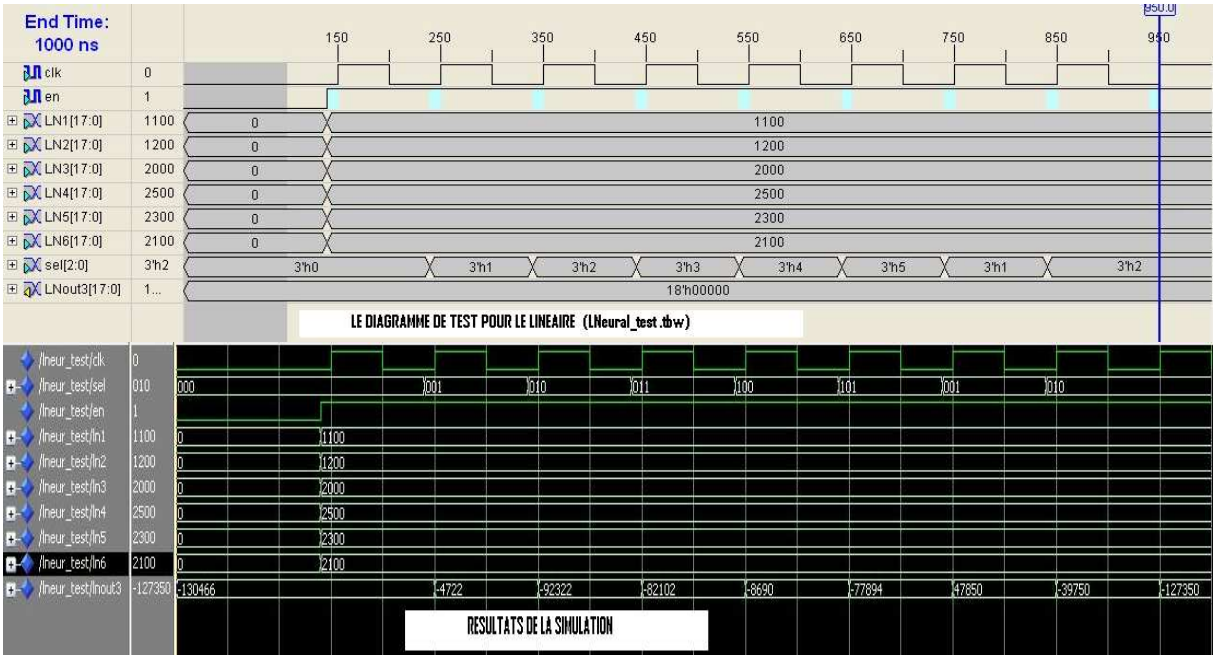


Figure VI.10 : Simulation du neurone linéaire

La figure suivante nous montre comment sont réparties les ressources du composant FPGA et le placement de design:

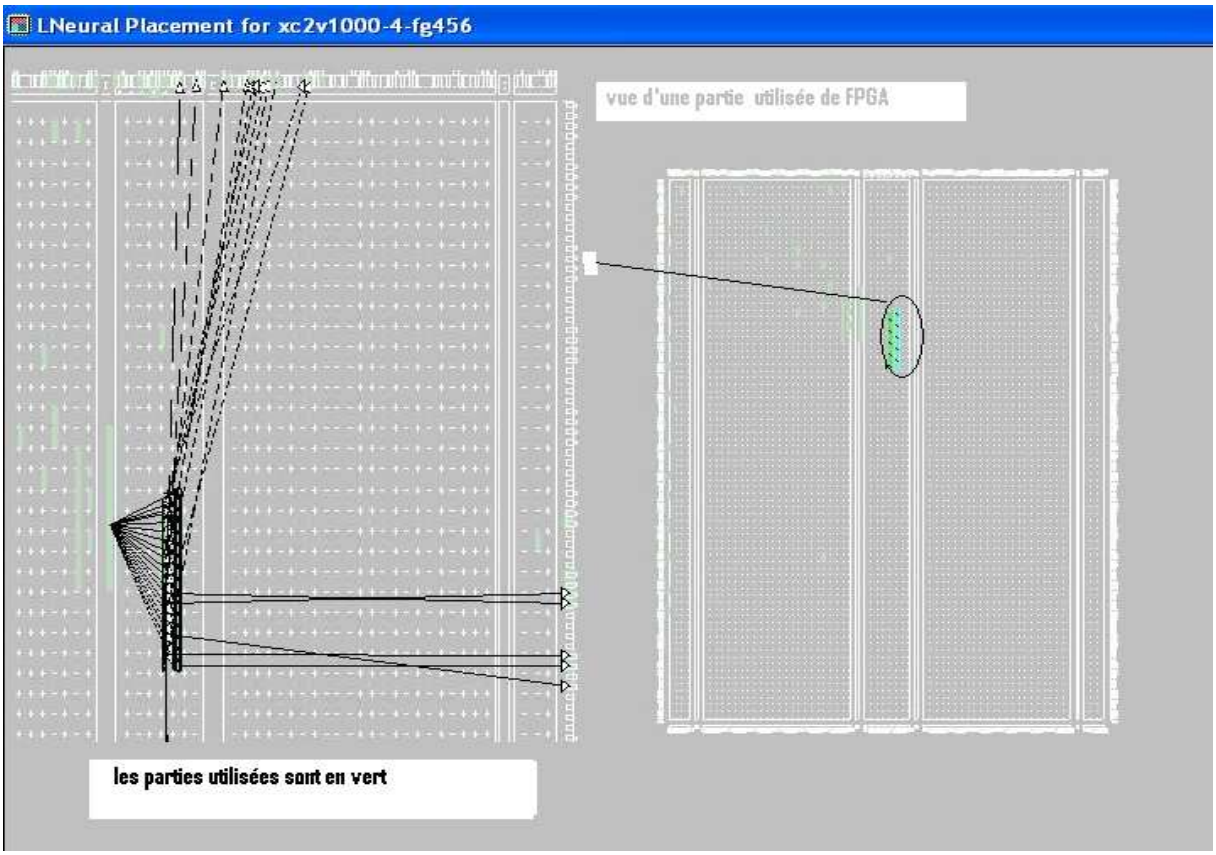


Figure VI.11 : Répartition des ressources

Pour bien visualiser le routage, nous avons inséré les figures suivantes sur le routage du design :

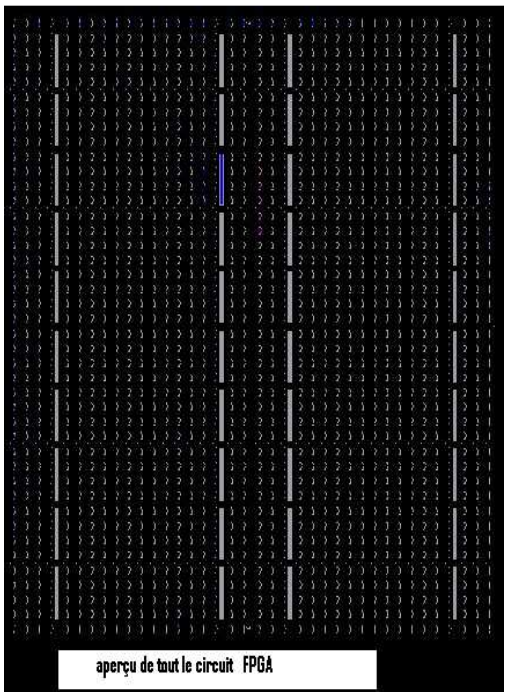


Figure VI.12 :routage du composant

Nous faisons l'agrandissement d'une partie pour voir plus de détail :

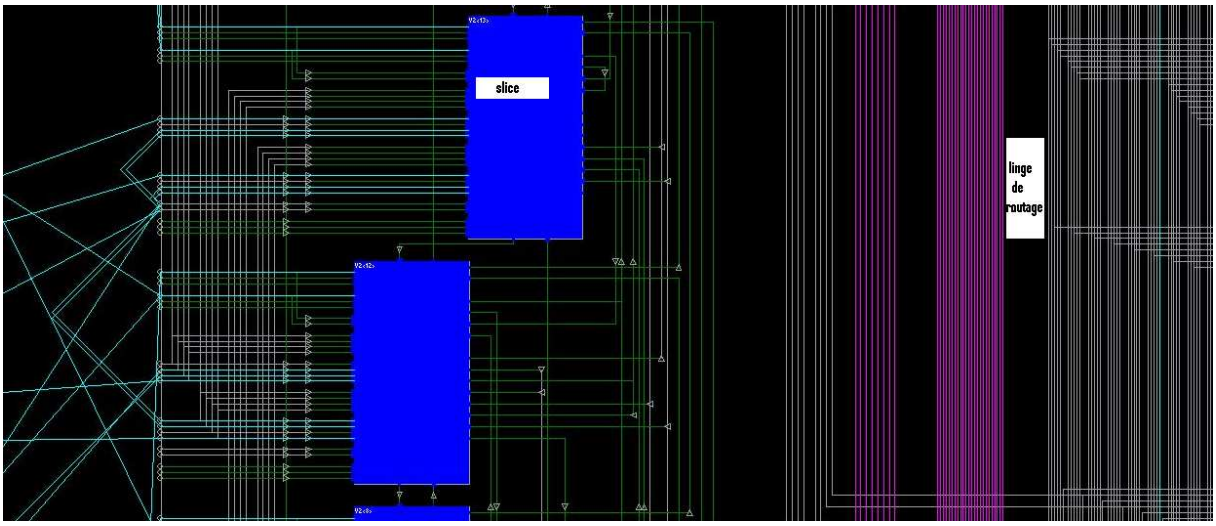


Figure VI.13 : agrandissement d'une partie du composant

Le schéma de la slice est donné ci-dessous :

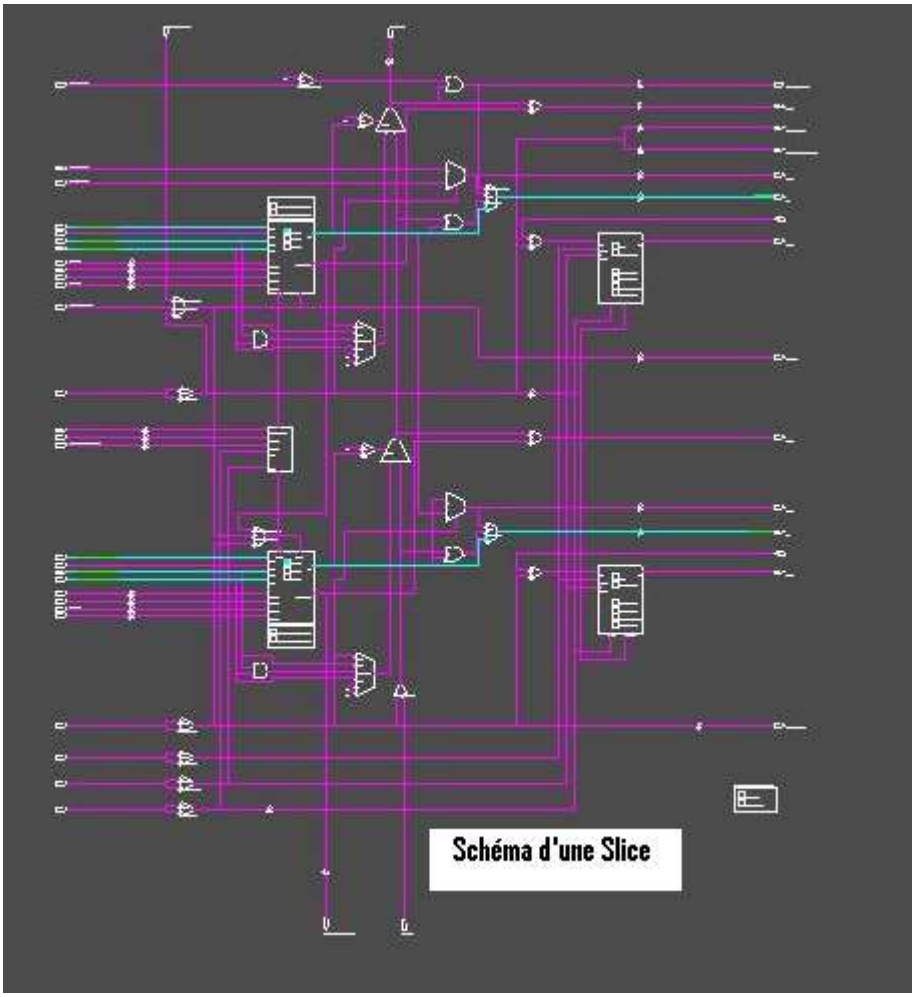


Figure IV.14 : schéma interne d'une slice

IV.d- Le neurone à sigmoïde :

Dans cette partie, nous allons étudier d'abord cette fonction de transfert, en étudiant son équation mathématique. Nous donnerons ensuite une approximation linéaire pour qu'il ait possibilité de l'implémenter.

IV.d.1- Etude générale :

La Sigmoïde est une fonction non linéaire qui se base sur la fonction exponentielle. Son équation mathématique est : $logsig(n) = 1 / (1 + \exp(-\lambda n))$

Son graphe est le suivant :

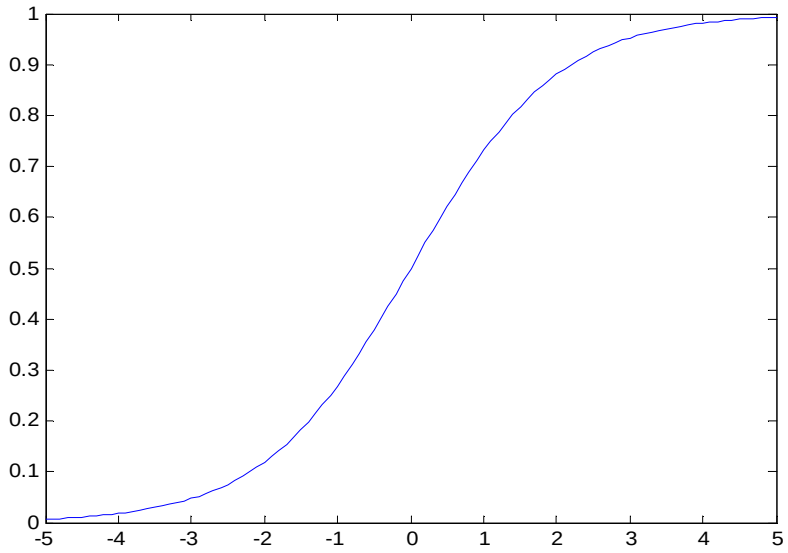


Figure IV.15 : graphe de la sigmoïde

La fonction sigmoïde est définie sur tout l'ensemble des réels :]- ∞ , +∞[

On remarque que la fonction a un point de symétrie qui est (0, 0.5). Elle est aussi limitée par deux asymptotes y=1 et y=0.

IV.d.2- Approximation de la fonction :

Il existe plusieurs méthodes, pour faire l'approximation de cette fonction. La méthode choisie dans notre cas s'appelle :Piece Wise Linear Approximation (PWL). [9]

Cette approche consiste à trouver une approximation linéaire de l'équation mathématique. La relation mathématique choisie pour l'approximation est la suivante :

$$\Theta(v) = \begin{cases} 1 & \text{Si } v > 4 \\ 0.0625\,v + 0.75 & \text{Si } 0 < v \leq 4 \\ 0.5 & \text{Si } v = 0 \\ 0.0625\,v + 0.25 & \text{Si } -4 \leq v < 0 \\ 0 & \text{Si } v < -4 \end{cases}$$

La figure suivante compare graphiquement les 2 relations :

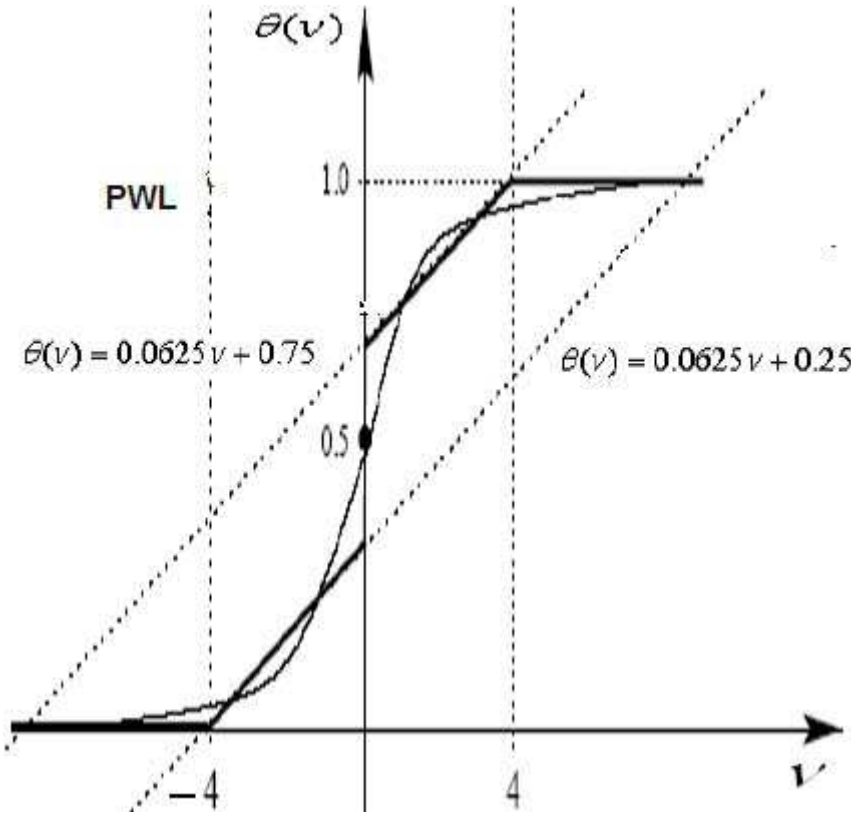


Figure IV.16 : Comparaison entre l'approximation et la fonction réelle.

On remarque que cette approximation est proche de la courbe réelle, mais il faut mentionner l'existence d'autres approximations plus complexes et plus précises qui divisent la courbe en plusieurs segments et prennent un intervalle plus large que]-4 , 4[.

IV.d.3- Interpolation de la relation :

Le principe de cette approximation est de diviser la partie linéaire en petits segments. Chaque segment sera représenté par une valeur unique. Cette valeur sera utilisée dans le code de la fonction sigmoïde pour l'implémenter sur le neurone.

Pour cela, nous avons choisi un pas de 0.5, ce qui veut dire 8 segments pour la partie positive et 8 autres pour la partie négative. Comme la fonction est symétrique au point (0 , 0.5), les valeurs négatives peuvent être déduites directement en faisant : $y_- = 1 - y_+$

Pour calculer les valeurs de la sigmoïde, on suit la méthode appliquée sur ces 2 exemples, l'un positif et l'autre négatif :

- Intervalle] 0 , 0.5] : Θ (v)=0.0625 v + 0.75

Θ (v)=0.0625 * 0.5 + 0.75 = 0 .78125

- Intervalle [-0.5 , 0[: Θ (v)= 0.0625 v + 0.25

Θ (v)= 0.0625*(-0.5) + 0.25 = 1 - Θ (0.5) = 0.21875

Le tableau suivant donne les résultats sur les différents segments :

Valeurs positives		Valeurs négatives	
Segment	Valeurs	segment	valeurs
] 0 , 0.5]	0.78125	[-0.5 , 0[	0.21875
] 0.5 , 1]	0.81250	[-1 , -0.5[	0.18750
] 1 , 1.5]	0.84375	[-1.5 ,1[	0.15625
] 1.5 , 2]	0.87500	[-2 , -1.5 [	0.12500
] 2 , 2.5]	0.90625	[- 2.5 , -2[	0.09235
] 2.5 , 3]	0.93750	[-3 , -2.5 [	0.06250
] 3 , 3.5]	0.96875	[-3.5 , -3 [	0.02125
] 3.5 , 4]	1	[-4 , -3.5 [	0

Dans l'implémentation, on se contente d'utiliser 4 chiffres après la virgule et ça pour suivre le modèle du réseau neuronal à implémenter.

Avant d'arriver au programme du neurone, on doit d'abord définir les différents blocs qui le constituent comme: la fonction de transfert sigmoïde, les ROM, le MUX... Quelques codes de ces composants seront donnés dans les annexes à la fin du document.

IV.d.4- Les résultats de la synthèse du neurone :

Une fois le code structurel du neurone est vérifié par l'outil ISE pour détecter les erreurs de la syntaxe, on passe à la simulation pour valider son model. Ensuite, ça sera l'étape de Implement Design avec ses 3 étapes : Translate, Mape et Place and Rote. Nous relevons les différents schémas et le tableau de consommation des ressources de ce neurone.

Les caractéristiques du neurone réalisé sont :

-2 entrées de 18 bits, « Ta » et « V ».

-un signal de sélection « sel » de 2 bits pour sélectionner les adresses de la ROM et l'entrée convenable.

-un signal d'activation « en » et un signal d'horloge « clk ».

-en sortie, le neurone nous donne un signal, Nout1, de 18 bit.

- Schéma général du neurone :

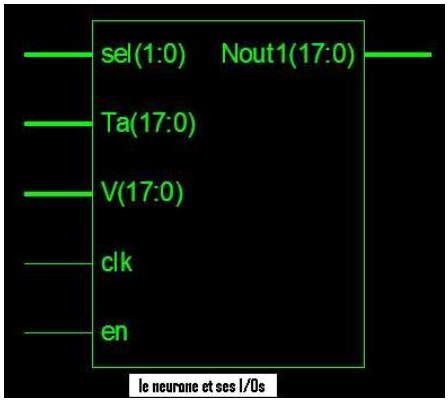


Figure VI.17 : Vue générale du neurone à sigmoïde

- Schéma interne du neurone : on voit bien les différents blocs du neurone

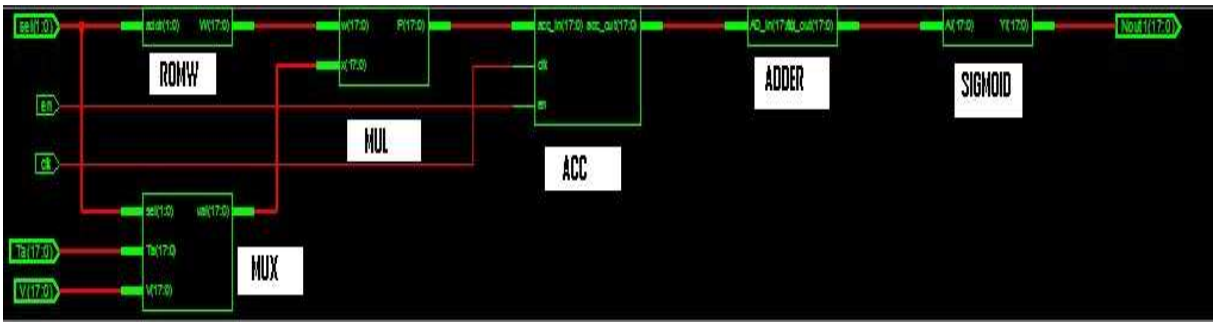


Figure VI.18 : Vue interne du neurone à sigmoïde

- Résultat de la simulation :

Nous donnons ci-dessous le résultat de la simulation :

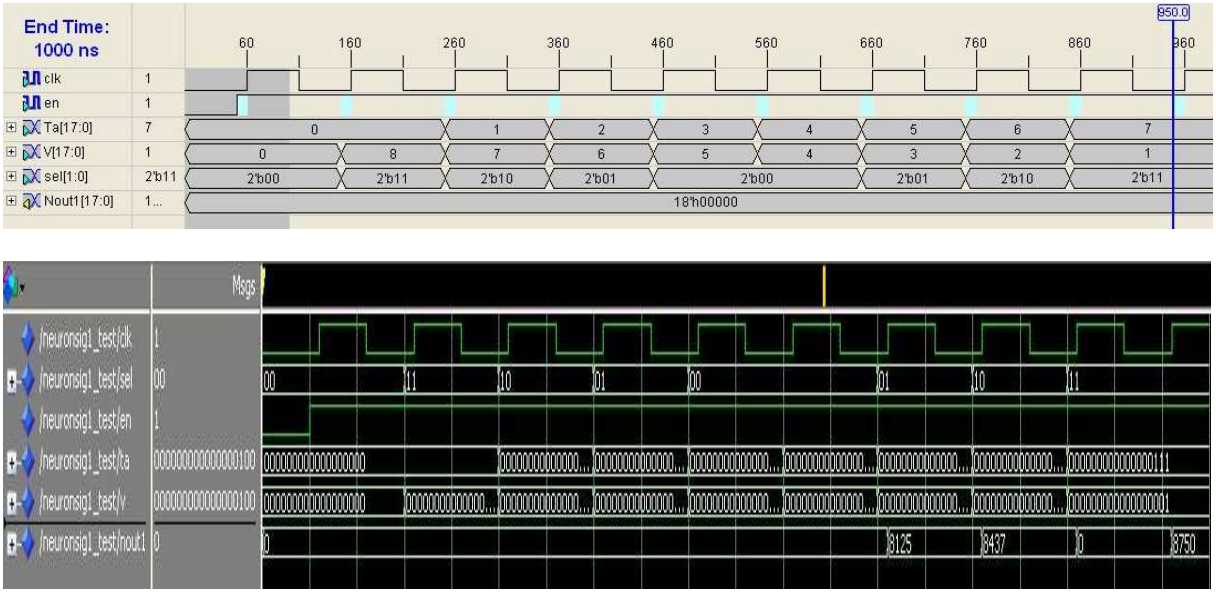


Figure VI.19 : Simulation du neurone

Nous allons donner maintenant les rapports et pourcentage de consommation des ressources de ce FPGA Virtex II :

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	50	10,240	1%
Number used as Flip Flops	18		
Number used as Latches	32		
Number of 4 input LUTs	346	10,240	3%
Logic Distribution			
Number of occupied Slices	205	5,120	4%
Number of Slices containing only related logic	205	205	100%
Number of Slices containing unrelated logic	0	205	0%
Total Number of 4 input LUTs	399	10,240	3%
Number used as logic	346		
Number used as a route-thru	53		
Number of bonded IOBs	58	328	17%
IOB Latches	18		
Number of MULT18X18s	1	40	2%
Number of GCLKs	2	16	12%
Total equivalent gate count for design	7,388		
Additional JTAG gate count for IOBs	2,784		

IV.e- La programmation de la couche du réseau :

La première couche de notre réseau est formée de 6 neurones à fonction de transfert sigmoïde. A l'entrée de la couche, nous possédons 2 entrées de données de 18 bits chaqu'une « Ta » et « V », et des signaux de contrôle « en » et « clk » qui sont des std_logic (1bit), et un signal de sélection couche « selc » de 2 bits. En sortie la couche a 6 sorties de 18 bits issues de chaque neurone de la couche. Elles seront connectées au neurone linéaire de qui constitue la couche de sortie.

La figure suivante nous donne une vue générale de la couche :

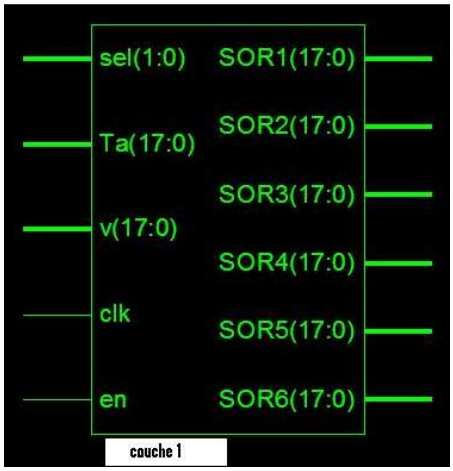


figure VI.20 : Vue externe de la couche

Le schéma interne de la couche est le suivant :

Elle contient les 6 neurones à fonction de transfert sigmoïde et explique comment ils sont interconnectés.

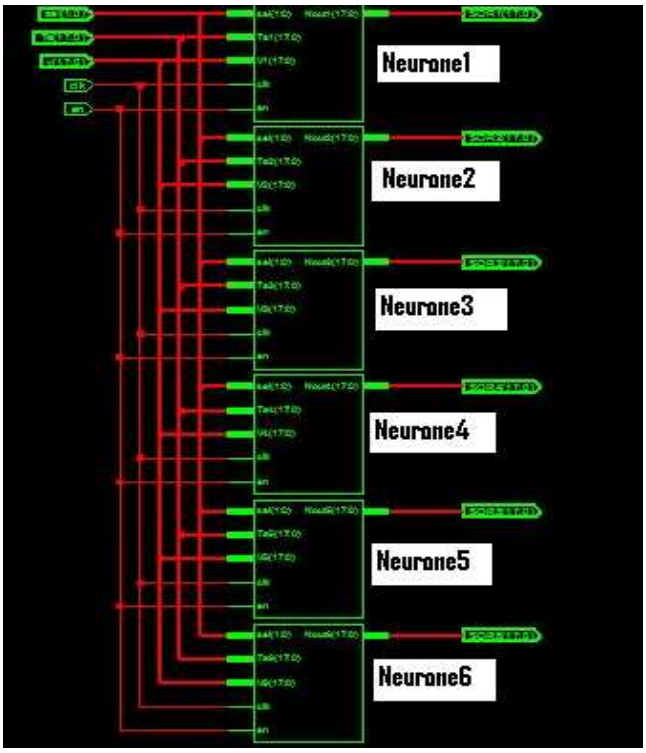


Figure VI.21 : Vue interne de la couche

Nous donnons dans la figure suivante le résultat de la simulation et le fichier test :

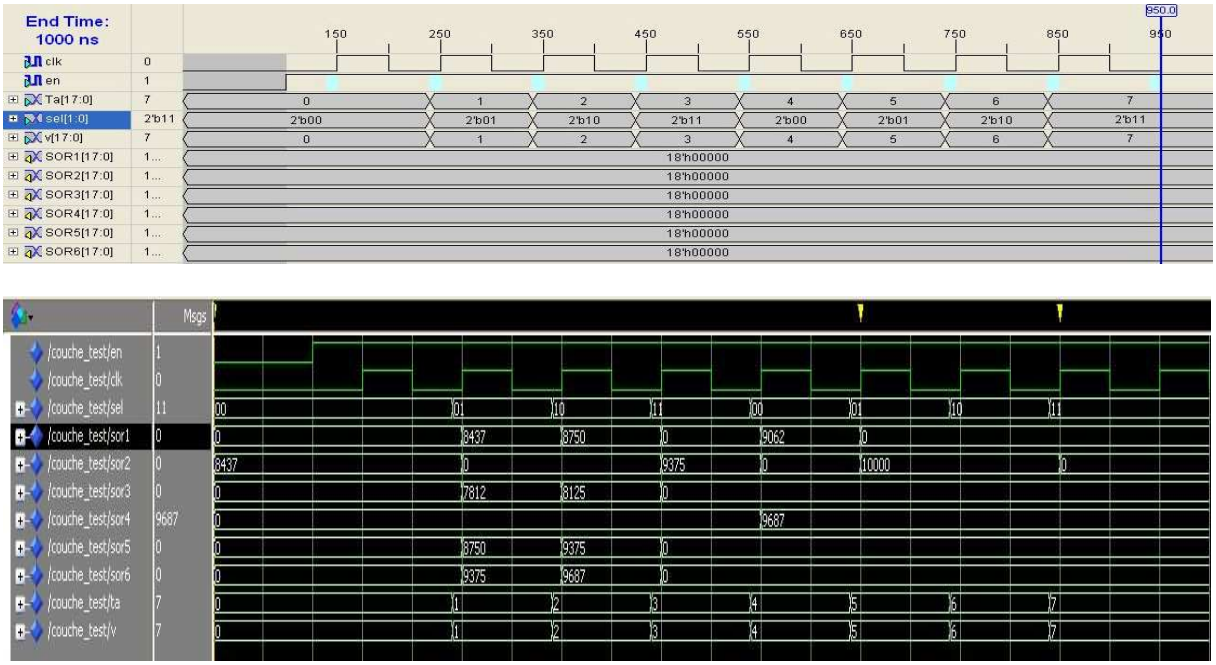


Figure VI.22 : Simulation de la couche

Le tableau suivant nous donne les taux de consommation des ressources :

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	300	10,240	2%
Number used as Flip Flops	108		
Number used as Latches	192		
Number of 4 input LUTs	1,998	10,240	19%
Logic Distribution			
Number of occupied Slices	1,201	5,120	23%
Number of Slices containing only related logic	1,201	1,201	100%
Number of Slices containing unrelated logic	0	1,201	0%
Total Number of 4 input LUTs	2,320	10,240	22%
Number used as logic	1,998		
Number used as a route-thru	322		
Number of bonded IOBs	148	328	45%
IOB Latches	108		
Number of MULT18X18s	6	40	15%
Number of GCLKs	7	16	43%
Total equivalent gate count for design	43,947		
Additional JTAG gate count for IOBs	7,104		

On remarque que les pourcentages commencent à augmenter à cause du nombre important de cellules utilisées. La surface de silicium utilisée a augmenté.

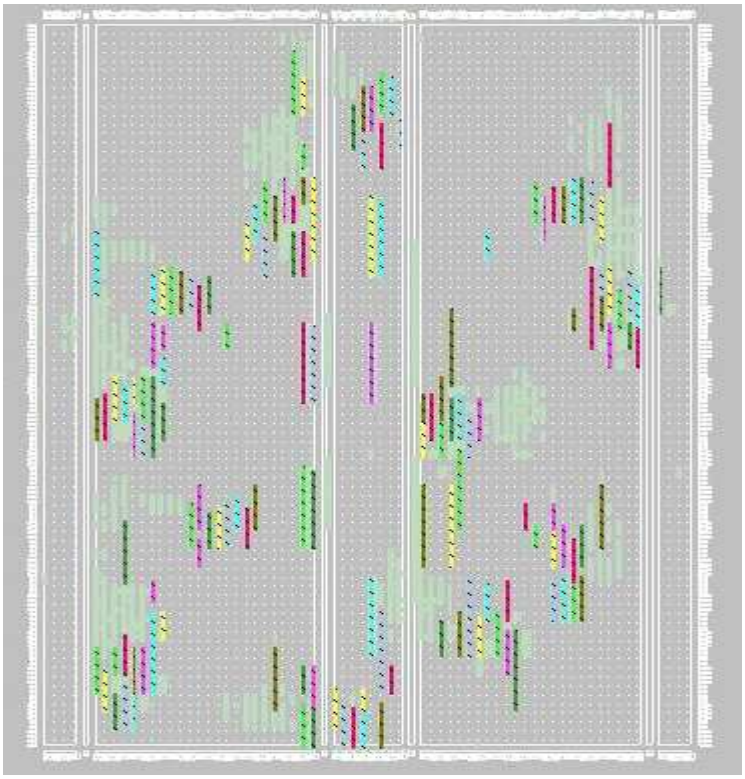


Figure VI.22 : les ressources consommées par la couche

On constate que la surface utilisée sur le FPGA pour implémenter la couche est plus importante par rapport à un seul neurone.

IV.f- La programmation de réseau :

Notre réseau est représenté dans la figure suivante, il modélise le comportement d'un capteur réalisé par [3].

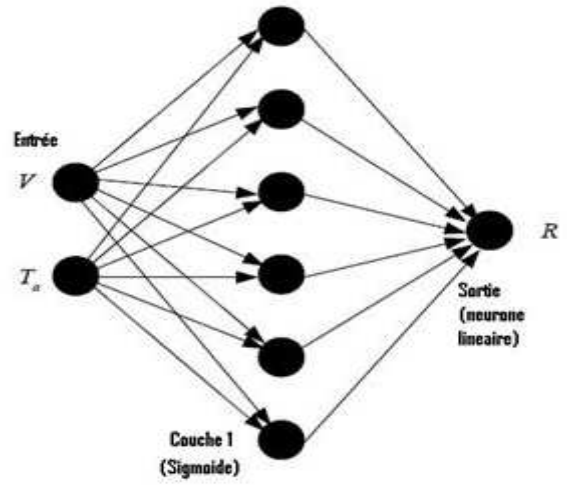


Figure IV.23 : Architecture du réseau

La sortie du neurone est: $R = Purelin(W_1 \log sig\ W_0 U + b_0) + b_1)$

Où :
U : Vecteur colonne des variables d'entrées du modèle ANN
W_0 : La matrice des poids de la couche d'entrée vers la couche cachée.
W_1 : La matrice des poids de la couche cachée vers la couche de sortie.
b_0 : Vecteur colonne des seuils de la couche cachée.
b_1 : Vecteur colonne des seuils de la couche d'entrée.

Les poids et les seuils finaux retenus du modèle ANN sont donnés ci-dessous :

$$W_0 = \begin{pmatrix} -3.2955 & -0.2270 \\ -0.6754 & -0.3118 \\ -1.2807 & -0.3101 \\ 3.8370 & 0.1412 \\ 0.0952 & -0.3733 \\ -11.3493 & -0.1035 \end{pmatrix} \qquad W_1 = [-1.9792 \;\; 2.7194 \;\; -0.3233 \;\; -1.2785 \;\; 2.3055 \;\; 4.5655]$$

$$b_0 = \begin{pmatrix} -0.8830 \\ 1.3403 \\ -0.0274 \\ -0.3534 \\ -1.5006 \\ -2.7957 \end{pmatrix} \qquad W_0 = -0.606$$

Nous allons donner maintenant le schéma général du réseau suivi du schéma interne :

Figure IV.24 : Schéma du réseau

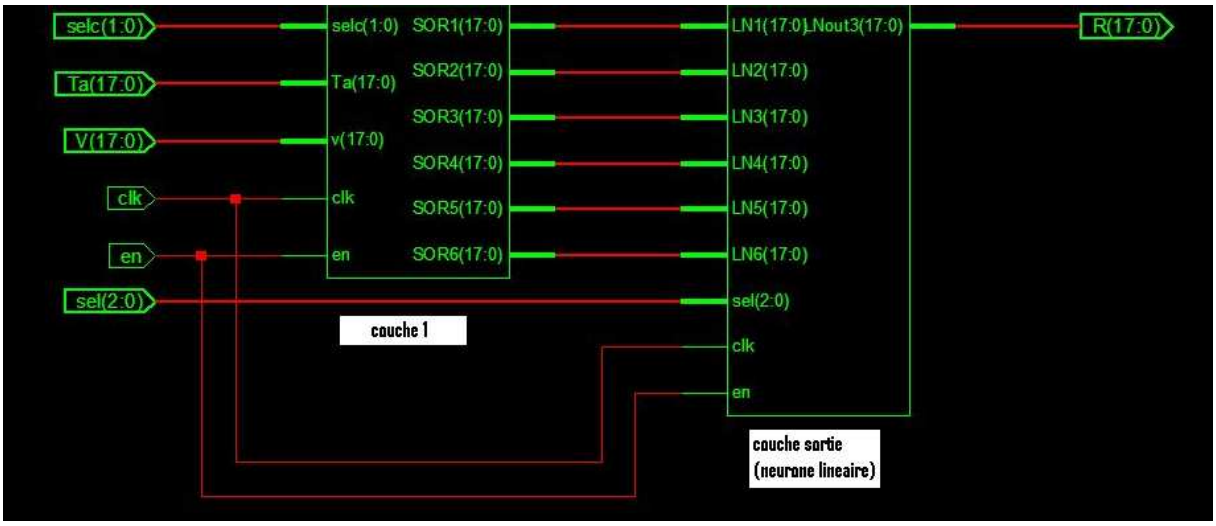


Figure IV.25 : Schéma interne du réseau

Nous remarquons bien comment sont reliées les sorties de la couche1 et les entrées du neurone de sortie. La simulation du réseau est donnée dans cette figure :

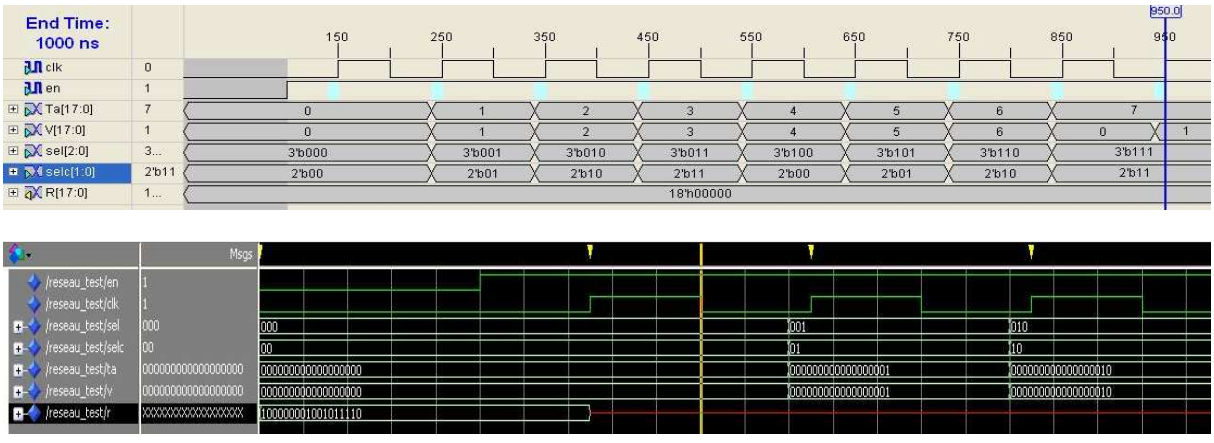


Figure IV.26 : Simulation du réseau

Les rapports de consommation sont donnés dans le tableau suivant :

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	426	10,240	4%
Number used as Flip Flops	126		
Number used as Latches	300		
Number of 4 input LUTs	2,111	10,240	20%
Logic Distribution			
Number of occupied Slices	1,256	5,120	24%
Number of Slices containing only related logic	1,256	1,256	100%
Number of Slices containing unrelated logic	0	1,256	0%
Total Number of 4 input LUTs	2,443	10,240	23%
Number used as logic	2,111		
Number used as a route-thru	332		
Number of bonded IOBs	61	328	18%
Number of MULT18X18s	7	40	17%
Number of GCLKs	7	16	43%
Total equivalent gate count for design	49,075		
Additional JTAG gate count for IOBs	2,928		

Dans cette dernière étape, on constate bien que le nombre de ressources consommées est plus important : sur les 10240 slices registres on a utilisé 426 qui représentent 4% , 24 % des slices, 23 % des LUTs et 18 % des IOs.

Nous allons montrer dans cette figure le résultat de placement des composants sur le circuit.

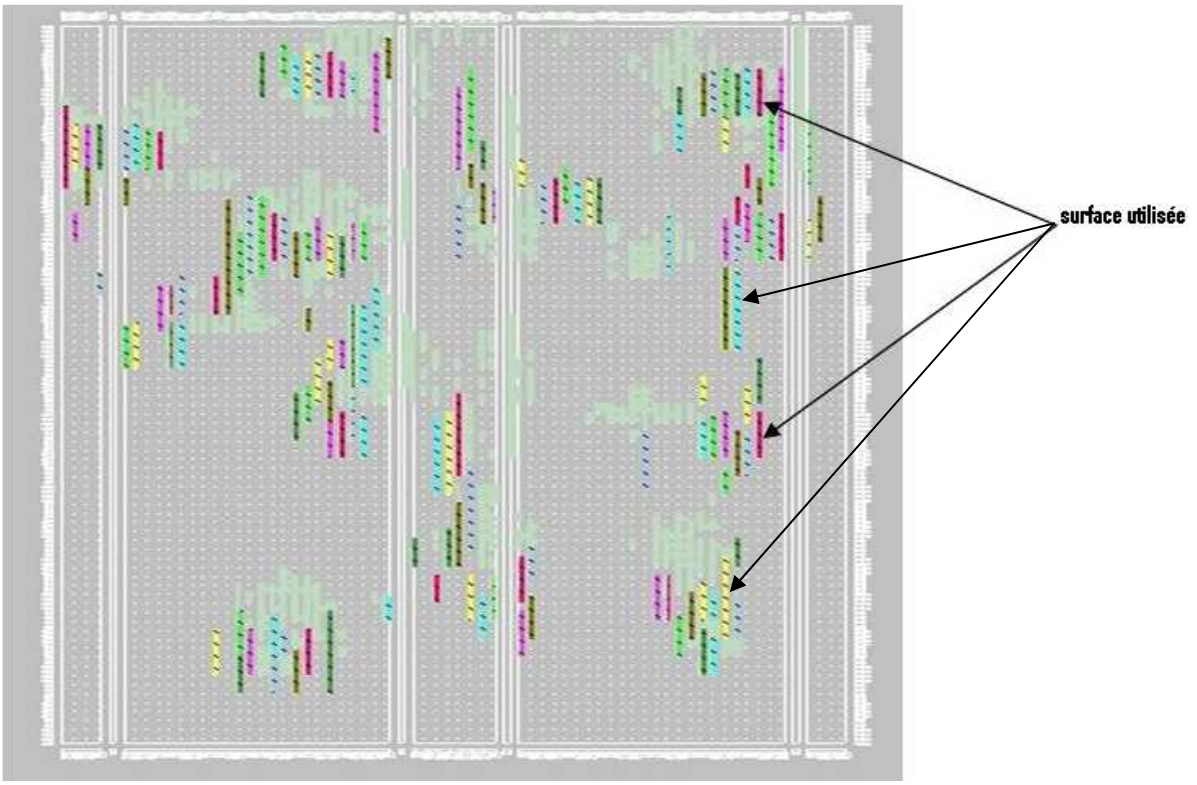


Figure IV.25 : Résultats du placement

IV.g- Conclusion :

Dans ce chapitre, nous avons proposé une méthode pour implémenter un réseau de neurone sur un FPGA. Le FPGA utilisé appartient à la firme Xilinx série Virtex II.

Nous avons commencé le chapitre par la description de 3 neurones. En suite, on a décrit la couche. Le réseau de neurone implémenté est un réseau qui modélise un micro capteur. La description des composants est faite en utilisant le langage VHDL. Nous avons proposé une architecture FPGA pour cette implémentation de type série.

Les rapports de l'opération sont pris à chaque fois dans un tableau pour nous permettre de comparer le tau de ressources consommées. Les résultats de la simulation sont à chaque fois reportés.

Conclusion générale :

L'implémentation des réseaux de neurones ne pose plus de gros problème avec l'avènement des FPGAs. En fait, la nano technologie et la micro électronique ont mis au point de nouveaux composants pour la programmation. De nombreuses et récentes études sont faites sur le domaine, notamment au CDTA où une équipe de recherche travaille sur cette thématique.

Dans ce travail, nous avons implémenté un réseau de neurones sur un FPGA. Nous avons exposé 3 architectures possibles pour faire cette implémentation. Nous avons choisi une architecture simple à réaliser et ne consomme pas beaucoup de ressources. C'est l'architecture « SP » (Serial Processing).

Nous avons essayé de faire une étude sur les réseaux de neurones et les FPGA pour bien comprendre le travail. A la fin du document, nous avons implémenté un réseau de neurones qui modélise le comportement d'un micro-capteur sur un FPGA Virtex II de Xilinx. Nous avons également utilisé le langage VHDL pour faire la description matérielle des circuits.

Les perspectives de recherche dans le domaine sont très vastes car la matérialisation et le passage du Software au Hardware reste toujours problématique. Une modélisation d'un système qui contient un micro capteur et qui communique ses données par un bus de communication sera un très bon thème pour parfaire notre travail.

ANNEXE1 : Codes de neurone à seuil

1-ADDER :

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ad is

   Port ( a : in  STD_LOGIC_VECTOR (15 downto 0);

          b : in  STD_LOGIC_VECTOR (15 downto 0);

          s : out  STD_LOGIC_VECTOR (15 downto 0));
end ad;

architecture Behavioral of ad is

begin

s <= a+b;   end Behavioral;
```

2-registre

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;            use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity reg is

   Port ( d : in  STD_LOGIC_VECTOR (15 downto 0);  clk : in  STD_LOGIC;  en : in
STD_LOGIC;

 rst : in  STD_LOGIC;  sortie : out  STD_LOGIC_VECTOR (15 downto 0));

  end reg;

architecture Behavioral of reg is

begin

process(clk , rst)

begin    if rst='0' then

       sortie <= (others => '0');     elsif (CLK'event and CLK='1')then

       if en='1' then      sortie <= D;

       end if; end if;

 end process;

end Behavioral;
```

3-accumulateur:

```
library IEEE;  use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;   use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.STD_LOGIC_SIGNED.ALL;    library ieee;

use ieee.std_logic_1164.all;   use ieee.numeric_std.all;

entity ACCU is

   port  (CLK    : in    std_logic;   RST    : in    std_logic;    EN     : in    std_logic;

      d      : in     std_logic_vector (15 downto 0);  sortie  : out    std_logic_vector (15 downto 0) ); end ACCU;

architecture structural of ACCU is

signal s1   :  std_logic_vector (15 downto 0); signal s2   :  std_logic_vector (15 downto 0);

component ad port (A : in std_logic_vector (15 downto 0);

   B      : in     std_logic_vector (15 downto 0);   S      : out    std_logic_vector (15 downto 0));  end component;

component reg port (     CLK    : in    std_logic;    RST    : in    std_logic;    EN     : in    std_logic;

 D: in  std_logic_vector (15 downto 0);  sortie      : out    std_logic_vector (15 downto 0) );  end component;

 begin

INST_REG :  reg port map(CLK=>CLK,RST=>RST,EN=>EN,D=>s2, sortie=>s1);

INST_ADD : ad port map(A=>D,B=>s1,S=>s2); sortie <= s1;      end structural;
```

4-Multiplieur:

```
ibrary IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;use
IEEE.STD_LOGIC_SIGNED.ALL;    entity multi is

Port ( a1 : in  STD_LOGIC_VECTOR (7 downto 0); b1 : in  STD_LOGIC_VECTOR (7 downto 0);

p : out  STD_LOGIC_VECTOR (15 downto 0)); end multi;

architecture Behavioral of multi is

begin     ProcessMultiplieur : PROCESS (a1, b1)

VARIABLE in1, in2, sor : INTEGER ;

BEGIN

in1 := CONV_INTEGER (a1);  ---(7 downto 0)) ;in2 := CONV_INTEGER (b1);  sor := in1 * in2 ;

p <= CONV_STD_LOGIC_VECTOR(sor, 16);

END PROCESS ProcessMultiplieur ;

end Behavioral;
```

5-Comparateur:

```
library IEEE;  use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_SIGNED.ALL;

entity comparateur is

generic(n: natural :=16);

port( AC: in std_logic_vector(n-1 downto 0);  sortie_c: out std_logic); end comparateur;

architecture Behavioral of comparateur is

constant seuil : std_logic_vector(n-1 downto 0) := "0000000000001000";   begin

 process(AC)        begin

      if (AC >= seuil) then     sortie_c <= '1';  elsif (AC < seuil ) then      sortie_c <= '0';   end if;  end process; end Behavioral;
```

6-Le Neurone:

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.STD_LOGIC_SIGNED.ALL;  entity neurone is

generic(n: natural :=16 ); Port ( clk : in  STD_LOGIC;en : in  STD_LOGIC; rst : in  STD_LOGIC; x : in  STD_LOGIC_VECTOR (7 downto 0);

w : in  STD_LOGIC_VECTOR (7 downto 0); resultat : out  STD_LOGIC); end neurone;

architecture structural of neurone is

signal s3 : std_logic_vector (15 downto 0); signal s4 : std_logic_vector (15 downto 0);

component ACCU port (d : in std_logic_vector (15 downto 0);CLK : in std_logic ; RST : in std_logic ; EN  : in std_logic ;

Sortie : out    std_logic_vector (15 downto 0)) ; end component;

component multi port ( a1 :in std_logic_vector(7 downto 0); b1 :in std_logic_vector(7 downto 0);p : out std_logic_vector(15 downto 0)   );

end component;

component comparateur port (AC : in std_logic_vector(n-1 downto 0);

sortie_c :out std_logic ); end component;

begin

inst_Multi :                multi port map( a1 => x , b1 => w, p => s3);

inst_ACCU :  ACCU port map (d => s3, rst=>rst, clk => clk, en=> en, sortie => s4 );

inst_comparateur: comparateur port map

(AC=> s4,  sortie_c => resultat);

end structural;
```

7-le Multiplixeur:

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux_rom is       Port ( sel : in  STD_LOGIC_VECTOR (1 downto 0);  Mux_Addr : out  STD_LOGIC_VECTOR (1 downto 0)  );

end mux_rom;

architecture Behavioral of mux_rom is

constant  In0 :   STD_LOGIC_VECTOR (1 downto 0) :="00" ;  constant  In1 :   STD_LOGIC_VECTOR (1 downto 0) :="01" ;

constant  In2 :   STD_LOGIC_VECTOR (1 downto 0) :="10" ;  constant  In3 :   STD_LOGIC_VECTOR (1 downto 0) :="11" ;

begin           process (sel)

begin      case sel is

     when "00" => Mux_Addr  <= In0;      when "01" => Mux_Addr <= In1;

     when "10" => Mux_Addr <= In2;  when "11" => Mux_Addr <= In3;  when others => Mux_Addr <= In0;   end case;end process;

end Behavioral;
```

8-ROM des X:

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_SIGNED.ALL;entity romx is

 Port ( Addr_x : in  STD_LOGIC_VECTOR (1 downto 0);  clk : in  STD_LOGIC;  rd : in  STD_LOGIC;

   dout_x : out  STD_LOGIC_VECTOR (7 downto 0)); end romx;

architecture Behavioral of romx is

type rom is array(0 to 3) of std_logic_vector(7 downto 0);   constant  mem: rom :=( "00000001",   "00001010",   "00001010",
"00001000",  others =>("--------" ) );

begin

process(clk, rd)   begin      if (clk'event and clk='1') then      if rd = '1' then      dout_x <= mem(conv_integer(addr_x));

else  dout_x <=  "00000000";  end if;   end if;

 end process;              end Behavioral;
```

9-ROM des W:

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_SIGNED.ALL;

entity romw is     Port ( Addr_w : in  STD_LOGIC_VECTOR (1 downto 0);   clk : in  STD_LOGIC;  rd : in  STD_LOGIC;

dout_w : out  STD_LOGIC_VECTOR (7 downto 0));end romw;

architecture Behavioral of romw is

type rom1 is array(0 to 3) of std_logic_vector(7 downto 0);

constant  mem1: rom1 :=( "00000000",  "00000011",   "00000000",  "00000100", others =>("--------" )  );

begin    process(clk, rd)       begin

   if (clk'event and clk='1') then   if rd = '1' then dout_w <= mem1(conv_integer(addr_w)); else   dout_w <=  "00000000";   end if;   end if;
```

10-bloc ROMs et MUX:

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux_test1 is

   Port ( sel : in  STD_LOGIC_VECTOR (1 downto 0);

    clk : in  STD_LOGIC;     rd : in  STD_LOGIC; dout_w : out  STD_LOGIC_VECTOR (7 downto 0);

 dout_x : out  STD_LOGIC_VECTOR (7 downto 0 )  );   end mux_test1;

architecture structural of mux_test1 is

component ROMW Port ( Addr_w : in  STD_LOGIC_VECTOR (1 downto 0);

         Dout_w : out  STD_LOGIC_VECTOR (7 downto 0);   clk : in  STD_LOGIC;

         rd : in  STD_LOGIC );   end component;

component ROMX   Port (  Addr_x : in  STD_LOGIC_VECTOR (1 downto 0);

  clk : in  STD_LOGIC;  rd : in  STD_LOGIC;

  dout_x : out  STD_LOGIC_VECTOR (7 downto 0));   end component;

component mux_rom  Port ( sel : in  STD_LOGIC_VECTOR (1 downto 0);

 Mux_Addr : out  STD_LOGIC_VECTOR (1 downto 0)   );  end component;

signal S7 : std_logic_vector(1 downto 0);

begin

inst_ROMW : ROMW port map(dout_w => dout_w , Addr_W => S7, clk =>clk, rd => rd );

inst_ROMX : ROMX port map(dout_x => dout_x , Addr_X => S7, clk =>clk, rd => rd );

inst_MUX_ROM : MUX_ROM port map( mux_addr => S7, sel => sel ); -----,in0=>in0, in1=>in1,in2=>in2,in3=>in3);

end structural;
```

11-code complet du composant:

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;  use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity neural is

Port ( -------rd : in  STD_LOGIC; clk : in  STD_LOGIC;  en : in  STD_LOGIC;   rst : in  STD_LOGIC;

 sel : in  STD_LOGIC_VECTOR (1 downto 0);  resultat_f : out  STD_LOGIC);   end neural;

architecture structural of neural is

signal  sx :std_logic_vector (7 downto 0);signal  sw :std_logic_vector (7 downto 0);

component mux_test1  port (sel : in  STD_LOGIC_VECTOR (1 downto 0); clk : in  STD_LOGIC;        rd : in  STD_LOGIC;

dout_w : out  STD_LOGIC_VECTOR (7 downto 0);  dout_x : out  STD_LOGIC_VECTOR (7 downto 0) );   end component ;

component neurone port  ( clk : in  STD_LOGIC;    en : in  STD_LOGIC;  rst : in  STD_LOGIC;

        x : in  STD_LOGIC_VECTOR (7 downto 0);    w : in  STD_LOGIC_VECTOR (7 downto 0);

  resultat : out  STD_LOGIC);    end component ;

begin

inst_mux_test1 : mux_test1 port map ( clk =>clk, rd=> en, sel => sel, dout_x => sx, dout_w=>sw);

inst_neurone : neurone port map  (clk =>clk, en => en, rst =>rst, x=> sx, w => sw, resultat=> resultat_f);

end structural;
```

ANNEXE 2 : Programme de neurone à fonction linéaire

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LNeural is

    Port ( clk : in  STD_LOGIC;  sel : in  STD_LOGIC_vector (2 downto 0); en : in  STD_LOGIC;

 LN1 : in  STD_LOGIC_VECTOR (17 downto 0);  LN2 : in  STD_LOGIC_VECTOR (17 downto 0);

 LN3 : in  STD_LOGIC_VECTOR (17 downto 0); LN4 : in  STD_LOGIC_VECTOR (17 downto 0);

LN5 : in  STD_LOGIC_VECTOR (17 downto 0);  LN6 : in  STD_LOGIC_VECTOR (17 downto 0);

LNout3 : out  STD_LOGIC_VECTOR (17 downto 0));       end LNeural;

architecture Structural of LNeural is

 signal V1  :  std_logic_vector (17 downto 0);  signal V2  :  std_logic_vector (17 downto 0);

signal V3  :  std_logic_vector (17 downto 0);   signal V4  :  std_logic_vector (17 downto 0);

----------la ROM des poids Wi----------

component ROM3_1  Port ( addr : in  STD_LOGIC_VECTOR (2 downto 0);

                  W : out  STD_LOGIC_VECTOR (17 downto 0));  end component;

 ----------le Multiplieur--------

component MUL3   Port ( x3 : in  STD_LOGIC_VECTOR (17 downto 0);

  w3 : in  STD_LOGIC_VECTOR (17 downto 0);  P3 : out  STD_LOGIC_VECTOR (17 downto 0)); end component;

----------l'Additionneur ----------

component add3    Port ( AD3_in : in  STD_LOGIC_VECTOR (17 downto 0);

                  Ad3_out : out  STD_LOGIC_VECTOR (17 downto 0)); end component;
----------le Multiplixeur----------

component mux3    Port ( Nout1 : in  STD_LOGIC_VECTOR (17 downto 0);

 Nout2 : in  STD_LOGIC_VECTOR (17 downto 0);  Nout3 : in  STD_LOGIC_VECTOR (17 downto 0);

 Nout4 : in  STD_LOGIC_VECTOR (17 downto 0);Nout5 : in  STD_LOGIC_VECTOR (17 downto 0);

 Nout6 : in  STD_LOGIC_VECTOR (17 downto 0);  Mout_3 : out  STD_LOGIC_VECTOR (17 downto 0);

  sel3 : in  STD_LOGIC_VECTOR (2 downto 0));              end component;

----------l'ACCUMULATEUR----------

component acc3     Port ( clk : in  STD_LOGIC;    en  : in  std_logic; ac_in  : in   STD_LOGIC_VECTOR (17 downto 0);

   ac_out : inout  STD_LOGIC_VECTOR (17 downto 0)   ); end component;

  begin

INST_MUX3 : mux3 port map ( sel3 =>sel, Nout1 =>LN1, Nout2 =>LN2, Nout3 =>LN3, Nout4 =>LN4, Nout5 =>LN5,  Nout6
=>LN6, Mout_3=>V1  );

INST_ROM3_1 : ROM3_1  port map (addr=>sel, w => V2   );

INST_MUL3  : mul3 port map (x3 =>V1,w3=>V2, p3=> V3   );

INST_acc3 : acc3  port map ( clk => clk, en=>en , ac_in => V3, ac_out => V4 );

INST_ADD3 : add3 port map ( ad3_in => v4, ad3_out =>  LNout3  );

end Structural;
```

ANNEXE3 : Programme de neurone à fonction sigmoide

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity neural2_1 is

 Port ( clk : in  STD_LOGIC;  selc : in  STD_LOGIC_vector (1 downto 0); en : in  STD_LOGIC;  Ta1 : in  STD_LOGIC_VECTOR (17 downto
0);

        V1 : in  STD_LOGIC_VECTOR (17 downto 0);  Nout1 : out  STD_LOGIC_VECTOR (17 downto 0));end neural2_1;



architecture Structural of neural2_1 is

signal T1  :  std_logic_vector (1  downto 0);signal T2  :  std_logic_vector (17 downto 0);signal T3  :  std_logic_vector (17 downto 0);

signal T4  :  std_logic_vector (17 downto 0);signal T5  :  std_logic_vector (17 downto 0);signal T6  :  std_logic_vector (17 downto 0);

----------la ROM des poids Wi----------

component ROM111  Port ( addr : in  STD_LOGIC_VECTOR (1 downto 0);W : out  STD_LOGIC_VECTOR (17 downto 0));end
component;

----------le Multiplieur--------

component MUL21   Port ( x : in  STD_LOGIC_VECTOR (17 downto 0);

  w : in  STD_LOGIC_VECTOR (17 downto 0); P : out  STD_LOGIC_VECTOR (17 downto  0)  );end component;

----------l'Additionneur ----------

component add21

Port ( AD_in : in  STD_LOGIC_VECTOR (17 downto 0); Ad_out : out  STD_LOGIC_VECTOR (17 downto 0));end component;

----------la fonction d'activation ----------

component LUT_SIG1 IS

port ( Ai     : in std_logic_vector (17 downto 0);    Yi     : out std_logic_vector (17 downto 0)    );end component;

----------le Multiplixeur----------

component mux21    Port ( selc : in  STD_LOGIC_VECTOR (1 downto 0);   V, Ta : in std_logic_vector (17 downto 0);

    val : out  STD_LOGIC_VECTOR (17 downto 0)   );       end component;

----------l'ACCUMULATEUR----------

component acc2_1    Port ( clk : in  STD_LOGIC; en  : in  std_logic; acc_in  : in   STD_LOGIC_VECTOR (17 downto 0);

acc_out : inout  STD_LOGIC_VECTOR (17 downto 0)   );end component;

 begin

INST_MUX21 : mux21 port map ( selc =>selc, val =>T3, Ta =>Ta1, v=>v1   );

INST_ROM111 : ROM111  port map (addr=>selc, w => T2   );

INST_MUL21  : mul21 port map (x =>T3,w=>T2, p=> T4   );

INST_acc2_1 : acc2_1  port map ( clk => clk, en=>en , acc_in => T4, acc_out => T5 );

INST_LUT_SIG1 : LUT_SIG1  port map (  Ai =>T6,Yi => Nout1  );

INST_ADD21 : add21 port map ( ad_in => T5, ad_out => T6  );
```

ANNEXE 3 : Le programme de la couche :

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity couche1 is

    Port ( en  : in  STD_LOGIC; clk : in  STD_LOGIC;  selc : in  STD_LOGIC_VECTOR (1 downto 0);

SOR1 : out  STD_LOGIC_VECTOR (17 downto 0); SOR2 : out  STD_LOGIC_VECTOR (17 downto 0);SOR3 : out  STD_LOGIC_VECTOR (17
downto 0);

 SOR4 : out  STD_LOGIC_VECTOR (17 downto 0);SOR5 : out  STD_LOGIC_VECTOR (17 downto 0);SOR6 : out  STD_LOGIC_VECTOR (17
downto 0);

 Ta : in  STD_LOGIC_VECTOR (17 downto 0);  v  : in  STD_LOGIC_VECTOR (17 downto 0) );end couche1;

architecture Structural of couche1 is          ----------Neurone1

component neural2_1   Port ( clk : in  STD_LOGIC; selc : in  STD_LOGIC_vector (1 downto 0); en : in  STD_LOGIC;

Ta1 : in  STD_LOGIC_VECTOR (17 downto 0); V1 : in  STD_LOGIC_VECTOR (17 downto 0); Nout1 : out  STD_LOGIC_VECTOR (17 downto 0));
end component;

component neural2_2   Port ( clk : in  STD_LOGIC; selc : in  STD_LOGIC_vector (1 downto 0); en : in  STD_LOGIC;

Ta2 : in  STD_LOGIC_VECTOR (17 downto 0); V2 : in  STD_LOGIC_VECTOR (17 downto 0);
Nout2 : out  STD_LOGIC_VECTOR (17 downto 0)); end component;

component neural2_3   Port ( clk : in  STD_LOGIC;       selc : in  STD_LOGIC_vector (1 downto 0); en : in  STD_LOGIC;

Ta3 : in  STD_LOGIC_VECTOR (17 downto 0); V3 : in  STD_LOGIC_VECTOR (17 downto 0);

Nout3 : out  STD_LOGIC_VECTOR (17 downto 0));          end component;

component neural2_4   Port ( clk : in  STD_LOGIC; selc : in  STD_LOGIC_vector (1 downto 0); en : in  STD_LOGIC;

Ta4 : in  STD_LOGIC_VECTOR (17 downto 0); V4 : in  STD_LOGIC_VECTOR (17 downto 0); Nout4 : out  STD_LOGIC_VECTOR (17 downto 0));
end component;

component neural2_5   Port ( clk : in  STD_LOGIC; selc : in  STD_LOGIC_vector (1 downto 0); en : in  STD_LOGIC;

Ta5 : in  STD_LOGIC_VECTOR (17 downto 0); V5 : in  STD_LOGIC_VECTOR (17 downto 0);

Nout5 : out  STD_LOGIC_VECTOR (17 downto 0)); end component;

component neural2_6   Port ( clk : in  STD_LOGIC;  selc : in  STD_LOGIC_vector (1 downto 0); en : in  STD_LOGIC;

Ta6 : in  STD_LOGIC_VECTOR (17 downto 0);  V6 : in  STD_LOGIC_VECTOR (17 downto 0);

Nout6 : out  STD_LOGIC_VECTOR (17 downto 0));  end component;

begin

INST_neural2_1 : neural2_1 port map ( selc =>selc,  Ta1 =>Ta, v1=>v , en=> en , clk => clk  , Nout1 =>SOR1   );

INST_neural2_2 : neural2_2 port map ( selc =>selc,  Ta2 =>Ta, v2=>v , en=> en , clk => clk  , Nout2 =>SOR2   );

INST_neural2_3  : neural2_3 port map ( selc =>selc,  Ta3 =>Ta, v3=>v, en=> en , clk => clk  , Nout3 =>SOR3   );

INST_neural2_4 : neural2_4  port map ( selc =>selc,  Ta4 =>Ta, v4=>v, en=> en , clk => clk  , Nout4 =>SOR4   );

INST_neural2_5 : neural2_5  port map ( selc =>selc,  Ta5 =>Ta, v5=>v, en=> en , clk => clk  , Nout5 =>SOR5   );

INST_neural2_6 : neural2_6 port map  ( selc =>selc,  Ta6 =>Ta, v6=>v, en=> en , clk => clk  , Nout6 =>SOR6   );

end Structural;
```

ANNEXE 4 : le programme complet de réseau

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity reseau is  Port ( en : in  STD_LOGIC; clk : in  STD_LOGIC;

sel : in  STD_LOGIC_vector (2 downto 0);

 selc : in  STD_LOGIC_VECTOR (1 downto 0);Ta : in  STD_LOGIC_VECTOR (17 downto 0);

        V : in  STD_LOGIC_VECTOR (17 downto 0);  R : out  STD_LOGIC_VECTOR (17 downto 0));  end reseau;

architecture Structural of reseau is

 component couche1 Port ( en  : in  STD_LOGIC; clk : in  STD_LOGIC; selc : in  STD_LOGIC_VECTOR (1 downto 0);

 SOR1 : out  STD_LOGIC_VECTOR (17 downto 0);  SOR2 : out  STD_LOGIC_VECTOR (17 downto 0);

SOR3 : out  STD_LOGIC_VECTOR (17 downto 0);  SOR4 : out  STD_LOGIC_VECTOR (17 downto 0);

  SOR5 : out  STD_LOGIC_VECTOR (17 downto 0);   SOR6 : out  STD_LOGIC_VECTOR (17 downto 0);

 Ta : in  STD_LOGIC_VECTOR (17 downto 0); v  : in  STD_LOGIC_VECTOR (17 downto 0) );end component;

component  LNeural    Port ( clk : in  STD_LOGIC;  sel : in  STD_LOGIC_vector (2 downto 0); en : in  STD_LOGIC;

  LN1 : in  STD_LOGIC_VECTOR (17 downto 0); LN2 : in  STD_LOGIC_VECTOR (17 downto 0); LN3 : in  STD_LOGIC_VECTOR (17 downto 0);

LN4 : in  STD_LOGIC_VECTOR (17 downto 0);  LN5 : in  STD_LOGIC_VECTOR (17 downto 0);  LN6 : in  STD_LOGIC_VECTOR (17 downto 0);
LNout3 : out  STD_LOGIC_VECTOR (17 downto 0));       end component;

signal F1  :  std_logic_vector (17 downto 0);signal F2  :  std_logic_vector (17 downto 0);signal F3  :  std_logic_vector (17 downto 0);

signal F4  :  std_logic_vector (17 downto 0);signal F5  :  std_logic_vector (17 downto 0);signal F6  :  std_logic_vector (17 downto 0);

begin

inst_couche1 :  couche1 port map(en =>en,clk=> clk,
selc=>selc,SOR1=>F1,SOR2=>F2,SOR3=>F3,SOR4=>F4,SOR5=>F5,SOR6=>F6,Ta=>Ta,V=>V );

inst_LNeural :  LNeural port map(en=>en,sel=>sel,clk=>clk, LNout3=>R ,LN1=>F1,LN2=>F2,LN3=>F3,LN4=>F4,LN5=>F5,LN6=>F6 );

end Structural;
```

Bibliographie:

[1] G.Dreyfus,J-M. Martinez, M.Samuelides, M.B. Cordon, F. Badran, S. theria, L. herault, "Reseaux de neurons" , Eyroles edit

[2]N. IZEBOUDJENE. Mémoire Magistère en électronique option Télécommunication Polytechnique Alger (1999)

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

[4] T.islam,C.Paramanik,H. Saha, "Modeling, simulation and temperature compensation of porous polysilicon capacitive humidity sensor using ANN technique", measurement review ,2004

[5] Christian Tavernier , " Circuits logiques programmables ''Dunod .P[200-262]

[6]Alexandre NKETSA, ''Informatique Industrielle et Circuits logiques programmables Mémoires, PLD, CPLD et FPGA'' , ellipse P[205-215]

[7] Laurent DAUTRIEU and Didier DEMIGNY, "Logique programmable: Architecture des FPGA et CPLD Méthodes de Conception , Le language VHDL, Eyrolles edit, p 137-156

[8] Kevin SKAHILL , "VHDL for Programmable Logic", ADISSON-WESLEY edit , p 74-94

[9] Mark ZWOLINSKI "Digital System Design with VHDL" ,Pearson,2nd édition 2004 p 206-209

[10]Jaques Weber, Maurice Meandere, ''Le langage VHDL cours et exercices'', 2nd édition Dunod

[11] Sa'ad Ahmed Al-Kazzaz, Rafid Ahmed Khalil,''FPGA implementation of artificial neurons: comparaison study'' ieee xplore

Web :

Xilinx Company web: **www.xilinx.com**

Altera company web**:** **www.altera.com**