+

COMPTE RENDU

Projet – VHDL

ARAUJO AMARAL

Fernando Lucas

AUDEBRAND Mathieu

4ème année – Groupe B

Tables des matières

[I. Introduction 3](#_Toc532590522)

[1. Objectif 3](#_Toc532590523)

[2. Principe d’un réseau de neurone 3](#_Toc532590524)

[3. Méthode de résolution 5](#_Toc532590525)

[II. Première Partie : Création d’un Neurone 5](#_Toc532590526)

[1. Création du Package : PackageSystem 5](#_Toc532590527)

[2. Création de l’entité Neurone 6](#_Toc532590528)

[3. Vérification avec l’entité de Test : Test2Neurones 6](#_Toc532590529)

[III. Deuxième Partie : création d’une couche de neurone 7](#_Toc532590530)

[1. Création de l’entité Couche 7](#_Toc532590531)

[2. Vérification avec l’entité de Test : TestCouche 8](#_Toc532590532)

[IV. Troisième Partie : implantation d’un réseau de neurones 9](#_Toc532590533)

[1. Création de l’entité Réseau de Neurones : ReseauNeurones 9](#_Toc532590534)

[2. Vérification avec l’entité de Test : TestReseauNeuronal 10](#_Toc532590535)

[V. Conclusion 11](#_Toc532590536)

[VI. ANNEXE 12](#_Toc532590537)

# Introduction

## Objectif

Ce projet de VDHL a pour but de mettre en œuvre les connaissances acquises lors des séances de cours et de travaux dirigés concernant ce langage de programmation. Aussi, un autre objectif est l’approfondissement des connaissances du logiciel ModelSim. L’objectif de ce Projet est de réaliser tout d’abord un neurone puis un réseau de neurones. Tout au long du projet, le fonctionnement de ces entités sera vérifié à travers des simulations.

## Principe d’un réseau de neurone

#### Neurone Biologique

L’unité cellulaire de base du système neurophysiologique est le neurone. Le cerveau en contient environ mille milliards. Le neurone est constitué d’un corps cellulaire (ou noyau). Il se ramifie en dendrites par où s’achemine l’information vers le neurone. Une fois l’information traitée par le neurone, elle est envoyée sur l’axone. La jonction entre deux neurones s’appelle synapse. Les informations parvenant au neurone peuvent venir d’autres neurones ou de capteurs sensoriels tels que le touché, la vue, l’odorat, etc., ou d’autres capteurs internes à l’organisme. Les informations sont de type électrique. Un neurone reçoit une information électrique de plusieurs sources, la transforme, puis la transmet vers d’autres neurones.

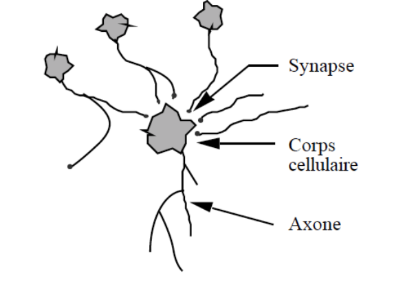


Figure : Neurone biologique

#### Neurone Artificiel

Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d’entrées en provenance de neurones en amont. À chacune de ces entrées est associé un poids représentatif de la force de la connexion. Chaque processeur élémentaire est doté d’une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones en aval. Le neurone additionne dans un premier temps les entrées avec leurs poids respectifs, puis une fonction de transfert calcul la valeur de l’état du neurone. Nous utiliserons ici une fonction de seuillage simple :

Z = autrement, avec, , T des constantes

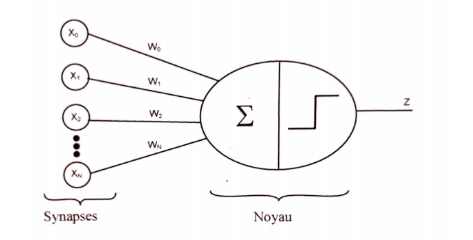


Figure : Neurone artificiel

La phase d’apprentissage du réseau permet de déterminer les poids Wi. Nous ne nous intéressons ici qu’à la phase d’utilisation proprement dite, c’est-à-dire lorsque ces poids sont connus et considérés comme des constantes en tableau.

#### Réseau neuronal

L’agencement des neurones en réseau peut prendre différentes formes. L’une des plus classiques est "l’organisation en couches". Chaque neurone voit alors toutes les sorties de la couche précédente. Le nombre d’entrées N fixe donc également le nombre de neurones sur la première couche. Chaque couche contient 2 fois moins de neurones que la couche précédente. Pour les neurones des couches inférieures, le nombre de données réellement produites par la couche supérieure est ainsi inférieur à N. Les entrées manquantes doivent alors être considérées à 0.

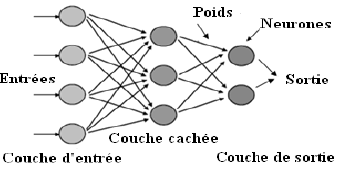


Figure : Réseau neuronal en couches

## Méthode de résolution

Pour réaliser ce projet nous avons décidé de diviser le problème en sous-étapes pour le simplifier. Ainsi, nous avons tout d’abord modélisé le fonctionnement d’un neurone puis d’une couche de neurone et enfin nous avons modélisé le réseau de neurones. Nous avons vérifié le bon fonctionnement de chaque entité modélisée à l’aide d’entités de test avant de passer à l’étape suivante. Nous avons aussi créé un package contenant toutes les constantes et tableaux nécessaires à la réalisation du réseau neuronal.

# Première Partie : Création d’un Neurone

Dans cette première partie nous avons compris le fonctionnement d’un neurone seul puis nous l’avons modélisé. Dans un premier temps, nous avons déclaré les différentes constantes essentielles au fonctionnement du neurone dans un package. Dans un second temps, nous avons créé une entité Neurone que nous avons par la suite testé à l’aide d’une entité de test.

## Création du Package : PackageSystem

Les paramètres T, Vmin, Vmax et N sont considérés comme des constantes globales pour le système et leurs valeurs sont imposées. Nous les avons donc définies dans un package *PackageSystem :*

* = 2, et représente la valeur minimale de l’état d’un neurone ;
* = 5, et représente la valeur maximale de l’état d’un neurone ;
* T = 10, et représente la valeur de seuil déterminant si la valeur de sortie du neurone vaut Vmin ou Vmax ;
* N = 4, et représente le nombre d’entrées du réseau (identique au nombre de neurones de la première couche).

La Constante N est de type « positive », car il faut obligatoirement un nombre positif de neurone pour réaliser le réseau. Cependant, nous avons choisi de choisir le type « integer » pour Vmin, Vmax et T car rien n’empêche de travailler avec des valeurs négatives pour le seuil et pour la valeur des états

Comme demandé, nous avons aussi déclaré un type tableau d’entier non contraint, dont les indexes sont de type « positive » nommés TAB et un sous-type correspondant pour des indices allant de 1 à N que nous avons nommés TAB\_N.

## 

## Création de l’entité Neurone

Une entité Neurone possède N entrées, une sortie. Les poids de ces synapses sont considérés comme des constantes génériques.

Pour réaliser cette entité nous avons mis en port d’entrée un tableau (*Tab\_IN\_Synapses)* de type TAB\_N qui représente les valeurs Xi des N neurones d’entrée. La sortie est unique et représente l’état du neurone, elle peut être modélisé par un port de sortie (*Zout) d*e type integer. Enfin la variable générique *Tab\_PoidsSynapses* de type TAB\_N représente les poids des synapses du neurone. Ces poids sont initialisés à 1 par défauts.

Le corps d’architecture *behavior* de l’entité neurone reprend le comportement d’un neurone artificiel décrit dans l’introduction. A travers cette architecture, le neurone réalise donc deux opérations :

* La somme des produits Wi \* Xi, calculée au moyen d’une variable intermédiaire Y et d’une boucle for allant de 1 à N.
* L’utilisation de la fonction de seuillage qui donne la valeur de sortie de l’entité (.

Cette description est séquentielle et de type comportemental, nous l’avons donc réalisé à l’aide d’un process qui contient le tableau des valeurs d’entrées Xi dans sa liste sensible. Ainsi, à chaque changement des valeurs Xi, le process sera exécuté et la valeur de sortie recalculé.

## Vérification avec l’entité de Test : Test2Neurones

Une fois l’entité Neurone créée, nous avons testé son comportement à travers une entité de Test nommée *Test2Neurones*. Ce test, vise à créer deux neurones à poids différents, et à les alimenter par une série de valeurs.

Pour réaliser cette entité, nous avons seulement besoin d’un seul composant Neurone. Nous avons choisis de créer un neurone (*Neurone1*) avec des valeurs de poids Wi initialisés par le paramètre générique (2, 3, 2, 2), et un autre neurone (*Neurone2)* sans poids initialisés qui prend donc les valeurs des poids données par défaut (1, 1, 1, 1).

On ajoute deux signaux Out1 et Out2 de type integer que nous connectons a la sortie des neurones pour pouvoir les observer. Enfin, on utilise un signal *Tab\_In* de type TAB\_N qui permet de donner et faire variés les valeurs Xi en entrée des deux neurones. A travers un process, on donne d’abord en entrée les valeurs (2, 2, 2, 2) puis (5, 2, 2, 2).



Figure : Simulation de l'entité Test2Neurones

A travers la simulation, on observe que notre sortie Out1 est égale à 5 alors que notre sortie Out2 est égale à 2. En effet pour le neurone1, nous avonsd’où la valeur (= 5) en sortie. Pour le neurone2, nous avons d’où la valeur (= 2) en sortie. De plus nous observons que quand les valeurs d’entrées varient les sorties sont recalculés et cohérentes avec le comportement du neurone.

# Deuxième Partie : création d’une couche de neurone

En s’inspirant de la figure 3, nous avons remarqué que le réseau de neurone se représente par un agencement de couche. Nous avons ainsi créé une entité représentant une couche de neurones. Nous avons par la suite testé l’entité Couche à l’aide d’une entité de test.

## Création de l’entité Couche

Une entité Couche possède N entrées nous avons mis en port d’entrée un tableau (*Entree)* de type TAB\_N qui représente les sorties des neurones de la couche supérieur. Elle possède également N sorties qui représentent l’état des neurones de la couche. Elle peut être représentée par un port de sortie (*Sortie)*  de type TAB\_N. Etant donné la flexibilité qui nous étais permise concernant la gestion des poids Wi des synapses, nous avons décidé de les déclarer en variable générique (*Tab\_PoidsSynapsesCouche)* avec comme type une matrice 2D (déclarer dans le package) de taille N\*N initialisé à 1 par défaut. Cette méthode permet de mettre un poids différents sur chaque synapse de chaque neurone. Enfin, nous avons aussi mis en variable générique le nombre de neurones (*Nb\_neurones)* qui entrait dans la couche qui est de type integer et que l’on initialise par défaut à 0. Cette variable nous permet de gérer le fait que dans un réseau neuronal, le nombre de données réellement produites par la couche supérieure est inférieur à N (à partir de la deuxième couche).

Le corps d’architecture *behavior* de l’entité Couche comprend plusieurs données :

* le composant Neurone qui est nécessaire à la création d’une couche ;
* un signal (*tempSortie*) de type TAB\_N qui permettant d’initialisé les valeurs de sortie à 0 et qui fait l’intermédiaire entre la sortie des différents neurones et la sortie de la couche ;
* une description du comportement de la couche ;

Le comportement de la couche est assez simple. Nous avons généré, un nombre de neurones égaux à la variable générique *Nb\_neurones* à l’aide des instructions Generate et For.

Pour chacun de ces neurones créés, nous avons connecté l’entrée de la couche à l’entrée du neurone car tous les neurones d’une ligne ont la même entrée (*Entree*) comme on le remarque sur la figure 3. De plus, nous avons connecté la sortie de chaque neurone à la case du tableau temporaire *tempSortie.* Enfin, nous avons passé en paramètre générique de chaque neurone la du tableau 2D : *Tab\_PoidsSynapsesCouche correspond aux poids de ces synapses*.

Pour finir, il suffit de reliés la sortie temporaire *tempSortie* à la sortie de la couche une fois la génération des neurones terminés. Nous aurons une sortie avec des valeurs d’états ( ou) pour les neurones de la couche et les sorties de la couche qui ne représentent pas de neurones seront considérés à 0.

## Vérification avec l’entité de Test : TestCouche

Une fois l’entité Couche crée, nous avons testé son comportement à travers une entité de Test nommée *TestCouche*. Cette entité vise à créer un test qui se rapproche le plus possible d’un réseau de neurones.

Pour réaliser cette entité, nous avons seulement besoin d’un seul composant Couche. Nous avons choisis de créer trois couches. *Couche\_with4N* possède N (= 4) neurones, *Couche\_with2N* possède deux fois moins de neurones (N/2 = 2) que la 1ère couche et *Couche\_with1N* possède deux fois moins de neurones (N/4 = 1) que la deuxième couche. Nous avons aussi connecté les entrés et le sorties à l’aide de signaux (*test\_sortie4N*, test\_sortie2N, test\_sortie1N) pour simulé un réseau de neurone en couche. Ainsi la sortie de *Couche\_with4N* est reliée à l’entrée de *Couche\_with2N* et la sortie de cette deuxième couche est reliée à la sortie *Couche\_with1N.*

Nous avons décidé de créés une matrice de dimensions 3 de tailles N\*N\*N dans le package que nous avons ensuite utilisé ici pour données les valeurs des poids des différents synapses. Ainsi chaque ligne représente les poids d’une couche et dans cette même ligne, il y a plusieurs tableaux représentant les poids des synapses de chaque neurone présent dans la couche.

Enfin, l’entrée de la première couche est relié aux valeurs d’entrées, ici (1,1, 1, 1) puis (1, 2, 2, 1)

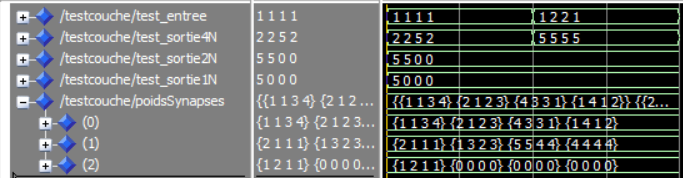


Figure : Simulation de l'entité TestCouche

A travers cette simulation, on peut déjà observer un comportement ressemblant à un réseau de neurone. Les états des neurones sont à suivant les valeurs des entrées et des poids des synapses. De plus, les entrées manquantes sont considérées à 0.

# Troisième Partie : implantation d’un réseau de neurones

Dans cette dernière partie nous avons réalisé une entité réseau de neurone en nous aidant des entités créées précédemment. Nous avons ensuite réalisé plusieurs tests de ce réseau neuronal en modifiant le nombre de couches de celui-ci.

## Création de l’entité Réseau de Neurones : ReseauNeurones

L’entité représentant un réseau de neurones possède une entrée *nombre\_neurones* de type positive qui représente le nombre de neurone de la première couche. Pour cela, il suffit d’un type positive. Elle est aussi constitué d’une entrée *Tab\_entree* de type TAB\_N qui prends les valeurs d’entrées de la première couche et d’une sortie *sortie\_reseau* qui est un tableau dont la taille est égale au nombre de neurone dans la couche de sortie et qui contient l’état des neurone de la dernière couche. Le nombre de neurone dans cette couche de sortie est égale à. C’est pourquoi le tableau de sortie est de type TAB(1 to à ). Enfin, pour compléter cette entité, nous avons ajouté une variable générique de type positive représentant le nombre de couche et un tableau de dimension 3 de tailles N\*N\*N représentant les valeurs des poids des différents synapses du réseau de neurones comme vu dans l’entité testCouche.

Pour réaliser ce réseau de neurones nous avons besoin d’un seul composant Couche. Nous avons aussi ajouté un signal *Tab\_sorties* de type MAT\_DIM\_2N (matrice de dimension 2) de taille N\* que l’on initialise à zéro et qui va nous permettre de conserver toutes les sorties des couches.

Dans l’architecture *behavior* de RéseauNeurones, nous créons la première couche à l’aide d’un générate qui prends en entrée, l’entrée du réseau (*Tab\_entree*) et qui sort la première ligne du tableau *Tab\_sortie.* A l’aide d’un deuxième générate et d’un for, nous créons les autres couches du réseau. Ces couches prennent en entrée, la sortie de la couche précédente. Pour calculer le nombre de neurone dans chaque couche, nous avons utilisé la formule suivante : *Nb\_neurones* =. Comme vu dans l’entité testCouche, nous associons à chaque couche une ligne de la matrice 3D contenant les poids des synapses des neurones. Enfin, nous connectons à la sortie les neurones de la dernière couche en enlevant les couches considérées à 0 à l’aide de la formule pour calculer le nombre de neurone en sortie comme vu précédemment.

## Vérification avec l’entité de Test : TestReseauNeuronal

Une fois l’entité ReseauNeurones créée, nous avons testé son comportement à travers une entité de Test nommée *TestReseauNeuronal*. Cette entité vise à créer un test qui permet de vérifier le réseau neuronal.

Pour réaliser cette entité de test, nous avons seulement besoin d’un seul composant ReseauNeurones. Nous avons déclaré une constante *nb\_couches* qui peut être changé à tout moment et qui est connecté à la variable générique *nombre\_couches* du réseau de neurone. Une autre constante *poids4N\_1* est une matrice de dimension 3 de taille N\*N\*N qui contient les différents poids du réseau. Enfin, nous avons ajouté 2 signaux. *test\_sortie* est un tableau qui contient les états des neurones de la dernière couche et test\_entree, un tableau de taille N qui contient les entrées de la première couche, ici (1, 2, 1, 2) puis (5, 2, 1, 2).

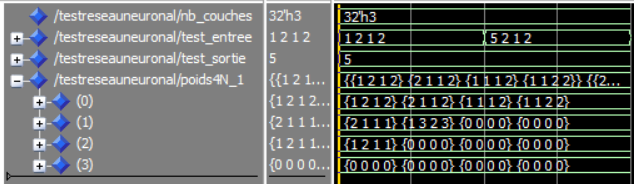


Figure : Simulation de l'entité TestReseauNeurones avec N = 4 et 3 couches

Avec cette première simulation, nous avons un nombre de couche égale à 3 et un nombre de neurones sur la première couche égale à 4. On observe que l’on à bien qu’une seul sortie comme souhaité et qu’elle est à l’état ce qui est correct par rapport à l’entrée et aux poids des synapses.

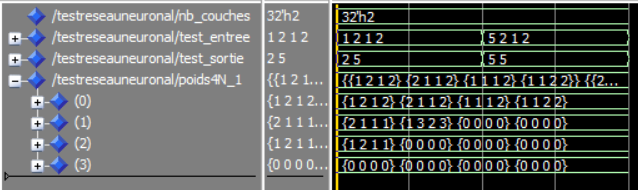


Figure : Simulation de l'entité TestReseauNeurones avec N = 4 et 2 couches

Avec cette deuxième simulation, nous avons un nombre de couche égale à 2 et un nombre de neurones toujours égale à 4. On observe que l’on a bien un tableau de deux sorties comme souhaiter et qu’elles sont respectivement à l’état puis ce qui est correct par rapport à l’entrée et aux poids des synapses.

# Conclusion

Pour conclure, à travers ce projet VHDL. Nous avons mis en œuvre nos connaissances acquises lors des séances de cours et de TP concernant ce nouveau langage. Nous avons approfondi nos connaissances du logiciel ModelSim, à travers les différentes simulations nécessitant ce logiciel. Nous avons compris les bases d’un réseau de neurones et la complexité qui lié à la création d’un neurone artificiel.

Concernant le poids des synapses nous avons choisi cette option, car nous avons pensé que cela pouvait être utile de nous-même rentrer les poids que l’on veut pour chaque synapse. Cela peut être fastidieux de le faire à la main mais nous pouvons imaginer par la suite un programme complémentaire permettant de rentrée ces valeurs directement dans le réseau de neurones à partir d’un fichier par exemple.

METTRE CODE EN ANNEXE !!

# ANNEXE