REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA

Faculté de **T**echnologie

Département du Génie Electrique

Mémoire de fin d'études de Master

Filière : Télécommunication
Spécialité : Systèmes de télécommunication

Etude et implémentation d'un réseau de neurones sur une plateforme de type Pynq.

Présenté par :

FARADJI Mohamed

ZERDALI Abdelhak

Proposé et Dirigé par :

Dr. BENGHARBIA Bilel

Dr. KARA Redha

Promotion 2019

Dédicaces

Je dédie ce modeste travail :

A mes très chers parents pour leur soutient et encouragement durant toutes mes années d'études et sans lesquels je n'aurais jamais réussi,

A toute ma famille, grande et petite sans exception,

A tous les enseignants et enseignantes durant mes années d'études avec lesquels j'ai beaucoup appris,

A mon binôme légendaire Faradji Mohamed et toute sa famille,

A mes chères amies

A Choauib, Mokhtar, Abderrahmane, Khaled, Rabeh et Hamid . . . et à toutes mes camarades de groupe SYSTEME DE TELECOMMUNICATIONS.

Ainsi qu'à toutes les personnes que j'ai connues, qui m'ont aidées, soutenues et encouragées.

Abdelhak

Dédicaces

Je dédie ce modeste travail :

A mes très chers parents pour leur soutient et encouragement durant toutes mes années d'études et sans lesquels je n'aurais jamais réussi,

A toute ma famille, grande et petite sans exception,

A tous les enseignants et enseignantes durant mes années d'études avec lesquels j'ai beaucoup appris,

A ce qui est très chère à mon cœur Aicha Gacem

A mon binôme légendaire Zerdali Abdelhak et toute sa famille,

A mes chères amies

A Abderrahmane, Mohamed, Nouri, Newfel ,Houssem et Mokhtar . . . et à toutes mes camarades de groupe SYSTEME DE TELECOMMUNICATIONS.

Ainsi qu'à toutes les personnes que j'ai connues, qui m'ont aidées, soutenues et encouragées.

Mohamed

الملخص

في هذا العمل، قمنا بتطوير و تنفيذ شبكة عصبية اصطناعية من نوع « ADALINE » شبكة خطية تكيفية على منصة FPGA من فئة « Zynq » من Xilinx مناسبة لنظام يعمل على كشف ترددات و سعات الإشارات التوافقية، هذه الشبكة العصبية تمت محاكاتها كرمز python مع معالج ثم نُفِّذت على شكل IPcore (نواة ذات ملكية فكرية) مخصصة، يديرها معالج zynq-7020.

الكلمات المفتاحية : شبكة عصبية اصطناعية، شبكة خطية توافقية، FPGA، الاشارات التوافقية، نواة ذات ملكية فكرية، تعلُّم.

Résumé

Dans le présent travail, nous avons développé et implémenté un réseau de neurone artificiel de type ADALINE « Réseau d'Adaptation Linéaire » à base d'un circuit FPGA de type « Zynq » de Xilinx. Pour un système qui sert à détecter les fréquences et les amplitudes des signaux harmoniques, dans le but de faire un diagnostic de défauts. Ce réseau de neurones est simulé sous forme d'un code python avec un processeur, puis implémenté sous la forme d'un IPcore personnalisé (propriété intellectuelle), géré par un processeur zynq-7000.

Mots clés : réseau de neurone artificiel, réseau d'adaptation linéaire, FPGA, les signaux harmoniques, IPcore, apprentissage.

ABSTRACT

In the present work, we have developed and implemented an artificial neural network of type ADALINE « Adaptive Linear Network » on Xilinx's « zynq » FPGA circuit. For a system that detects frequencies and amplitudes of harmonic signals, in order to make a fault diagnosis. This neural network is simulated as python code with a processor then implemented as a custom IPcore (intellectual property), managed by a zynq-7020 processor.

Key words: artificial neural network, adaptive linear network, FPGA, harmonic signals, IP-Core, learning.

Table des matières

Liste des figures……..……...……....1
Liste des tableaux……..……...…….3
Abréviations……..……4
Introduction générale…….……..……..6
Chapitre I : Réseaux de neurones……..…….……8
I.1 Introduction…… ..…….……...8
I.2 Le neurone ……. …….……..……..8
I.3 Bases biologiques…… ……...……..9
I.4 Neurone formel (Artificiel) ……10
I.4.1. Définition ……..…...……...10
I.4.2. Les paramètres ……..…..……... 10
I.5. Principe de fonctionnement ……..……..……...11
I.6. Architecture des réseaux de neurones …….……..…… ……...11
I.6.1. Réseaux statiques ……...…..……..……..……...12
I.6.2. Réseaux dynamiques ……...…….…… ……12
I. 7. Apprentissage des neurones artificiels …….. ……..12
I.7.1. Définition …...……...……12
I.7.2. Apprentissage supervisé… ……..…… …….13
I.7.3. Apprentissage non supervisé ……...……...13
I. 8. Perceptron …… …..……13
I.8.1. Perceptron d'une seule couche ……...……..……..13
I.8.2. Perceptron multicouche ……. …….…..……14
I.9. Modèle Adaline ……... ……..……...15
I. 10. Conclusion ……. ……..…….19
Chapitre II : Les circuits programmable FPGA…….…..20
II.1 Introduction……...…20
II.2 Les circuits programmable FPGA…… ……20
II.3 Structure interne ……. ……..…..22
II.3.1 Logique des FPGA…….……..….. 22
II.3.2 Interconnexion……...…..……...24
II.3.3 Entrée et sortie……...…..…….…...25

II.3.4 Horloge……..……26
II.3.5 Configuration…….. ……..27
II.3.6 Consommation d'énergie…… ……. …….....28
II.4 Les Familles des FPGA ……...28
II.6 Conclusion …… ……..…..…..31
Chapitre III : Etude pratique et implémentation ……..32
III.1 Introduction……..……32
III.2 Présentation des outils de développement ……..…32
III.2.1 logiciels utilisés…….32
III.2.2 librairies et modules utilisés……..34
III.3 Les outils de développements : ……. …….…32
III. 3.1 PYNQ-Z2 …… ……....34
III. 3.2 Configuration de la carte ……..… ……..…..…35
III. 3.3 Spécifications de produit …….……...…36
III. 3.4 Allumer le PYNQ-Z2 …….…….....36
III 4 Création de notre projet …….…….37
III. 4.1 Notre architecture ……..…… 37
III. 4.2 Architecture du réseau de neurones Avec system generator ……..…….38
III. 4.3 Création de IP core …….……..……..39
III. 4.4 Conceptions de notre architecture sur vivado suite designe …… ……..40
III. 4.5 Conception avec vivado suite design ……..……42
III. 4.6 Conception de l'algorithme de la détection des harmoniques……..43
III. 4.7 Conception hardware ……..……...48
III. 4.8 Comparaison et interprétation des résultats …….....51
III. 5 Conclusion ……..51
Conclusion générale…….52
Annexes…… ……...53
Références bibliographiques……..……54

Liste des figures

Figure I.1 Représentation d'un neurone réel. ……. 9
Figure I.2 Modèle de base d'un neurone formel ……..……. 11
Figure I.3 Le Perceptron : structure et comportement……... 13
Figure I.4 Modèle du Perceptron Multi–Couches ……..…….. 14
Figure I.5 Schéma de principe de l'adaline .……. 16
Figure I.6 Signal fourni en apprentissage au réseau adaline……..…… 17
Figure I.7 Erreur entre signal initial et le signal prédit par le réseau de neurones…….....17
Figure I.8 Signal fourni en apprentissage au réseau adaline ……..…... ……18
Figure I.9 Erreur entre signal initial et le signal prédit par le réseau de neurones…….....18
Figure I.10. Erreur entre le signal initial et le signal prédit par le réseau de neurones……19
Figure II.1 Structure interne du FPGA ….…… 21
Figure II.2 Structure interne simplifiée du FPGA ……..….. ……..21
Figure II.3 Une cellule logique est la pierre de fondation d'une FPGA……...… …….23
Figure II.4 Bloc d'entrée / sortie générique ……..…….25
Figure II.5 Réseaux de distribution d'horloge. …… ……26
Figure II.6 Configuration de FPGA basés sur la RAM statique ……..…28
Figure II.7 Inverseur CMOS ……28
Figure II.8 Présentation du SoC Zynq-7000 ……..…… 30
Figure III.1 La carte PynqZ2 ……. ……35
Figure III.2 Architecture du notre réseau Avec system generator…….…… 38
Figure III.3 Onglet compilation dans la fenêtre du system generator …. ……..…….. 39
Figure III.4 Onglet clocking dans la fenêtre du system generator ……..40
Figure III.5 Conceptions de notre architecture sur vivado suite designe ……..…… 41
Figure III.6 L'organigramme de l'algorithme software……..44
Figure III.7 le signal cible traitée ……..……...……...45
Figure III.8 deux signaux d'entrée cosinus et sinus avec une fréquence de f0……...…..46
Figure III.9 deux signaux d'entrée cosinus et sinus d'entrée avec 2f0 ……... ……...46
Figure III.10 deux signaux d'entrée cosinus et sinus d'entrée avec 3f0 ……..... 46
Figure III.11 Résultat d'apprentissage ……..……... ……..47
Figure III.12 l'erreur d'apprentissage ……..…….....47
Figure III.13l'harmonique détecté en fonction cosinus ……..……... ……..48

Figure III.14 les harmoniques détectés en fonction sinus ……..……..... .48
Figure III.15 L'organigramme de l'algorithme hardware ……... ……..49
Figure III.16 l'harmonique détecté en fonction cosinus ……..……..... 50
Figure III.17 les harmoniques détectés en fonction sinus ……...……... ……...51

Liste des tableaux

Tableau III.1 les spécifications du carte PynqZ2 36
Tableau III.2 comparaisons entre les exécutions hardware et software 51

Abréviations

3D : three-dimensional.

ADALINE : ADAptive LInear Element.

ALU : arithmetic logic unit .

APU:Accelerated Processing Unit.

ARM : Advanced RISC Machines.

AXI: Advanced Extensible Interface.

CLB : Configurable Logique Bloc.

CMOS : Complementary metal-oxide-semiconductor.

CPU:Central Processing Unit.

DDR3: Double Data Rate Type 3.

DSP:digital signal processing.

DPU : Deep Learning Processing Unit .

E/S: entré/sortie.

FPGA : Field Programmable Gate Array.

HDL: Hardware Description Language.

IC : Integrated Circuit

IOP:input-output processor.

IP: intellectual property.

ISE: Integrated Synthesis Environment.

JTAG :Joint Test Action Group.

LED: light emmiting diode.

LUT : look up table .

MAC : accès aux médias .

Mcode: machine code.

MLP : Multi Layer Perceptron.

MOS: metal-oxide-semiconductor.

MPcore:multicore processor.

PL : processing system .

PS : programmable logic .

PCB : Printed Ciruit Board.

PHY : PHYsic signalization.

PMC : Perceptron Multi-Couche.

RISC:reduced instruction set computer.

RNA : Réseaux de Neurones Artificiels .

SOC:System-On-Chip.

SRAM : Static Random Acces Memory.

TCL: Tool Command Language.

UART:universal asynchronous receiver-transmitter.

USB:Universal Serial Bus.

VHDL:Very High-Speed Integrated Circuit Hardware Description Language.

XSG : xilinx system generator .

Introduction générale

Depuis une dizaine d'années, l'utilisation des réseaux de neurones artificiels s'est développée dans de nombreuses disciplines (sciences économiques, écologie et environnement, biologie et médecine…). Ils sont notamment appliqués pour résoudre des problèmes de classification, de prédiction, de catégorisation, d'optimisation, de reconnaissance des formes et de mémoire associative.

Les réseaux de neurones artificiels sont des implémentations (software ou hardware) de la structure neuronale des cerveaux biologiques. Le cerveau contient des neurones similaires aux commutateurs organiques. Ceux-ci peuvent changer leur état de sortie en fonction de la force de leur entrée. L'apprentissage se produit en activant à plusieurs reprises certaines connexions neuronales par rapport à d'autres, ce qui renforcent ces connexions. Cela les rend plus susceptibles de produire un résultat souhaité à partir d'un apport spécifié. Les réseaux de neurones peuvent apprendre à résoudre de nouvelles tâches. Contrairement aux méthodes traditionnelles de résolution de problèmes, aucun algorithme explicite ne doit être dérivé.

L'utilisation du FPGA pour l'implémentation de réseaux neuronaux offre une flexibilité dans les systèmes programmables. Le réseau de neurone artificiel basé sur les FPGA a été assez bien appliqué avec les applications de la classification, prédiction et optimisation. La programmabilité des FPGA reconfigurables permet la disponibilité de matériel spécifique rapide pour des applications étendues. Sa programmabilité pourrait créer les conditions nécessaires à l'exploration de nouveaux algorithmes de réseaux de neurones. Les FPGA combinent les avantages des systèmes hardware et software en fournissant un matériel reprogrammable et souple.

Ce travail de fin d'étude, a pour objectif l'étude et l'implémentation d'un réseau de neurone de type ADALINE sur une structure embarque de type FPGA. Le réseau de neurone est réalisé d'abord sous forme d'un IP-core et qui sera géré par la suite par un processeur hardcore de type ARM implémenté dans la plateforme pynq-Z2. Pour voir l'efficacité du IP-core réalisé en terme de temps d'exécution, nous avons implémenté un programme software du même architecture ADALINE sur la même plateforme.

Le manuscrit de la thèse s'articule autour de trois chapitres :

Le premier chapitre contient une description générale des réseaux de neurones et de ses différents types ainsi que les déférents types d'apprentissage.

Le deuxième chapitre fournit une introduction à la technologie FPGA. Une connaissance de base des blocs de construction et de l'architecture sous-jacente est importante pour développer des solutions efficaces en termes de ressources. Dans ce chapitre, nous examinons les caractéristiques architecturales de base des FPGA, une connaissance de base de la structure interne et la manière dont une conception correspond à cette architecture peuvent être utilisées pour développer une implémentation plus efficace.

Le troisième chapitre décrit le processus de conception et d'implémentation d 'une application de réseau neuronal sur une carte FPGA. Outre les outils et les logiciels utilisés pour l'implémentation ; nous allons présenter une étude comparative des exécutions software et hardware de notre réseau neuronal.

Chapitre I : Réseaux de neurones

1. Introduction

Il existe plusieurs technologies dans la vie courante qu'ont permis aux scientifiques d'élaborer et de perfectionner des concepts pour différents domaines, dans ces concepts il existe un concept qui est utilisée dans plusieurs domaines de recherches et de différentes manières, ce concept est celle des réseaux de neurones artificiels (RNA).

Les réseaux de neurones artificiels sont présentés comme la vague du futur en informatique. Ce sont en effet des mécanismes d'autoapprentissage qui ne nécessitent pas les compétences traditionnelles d'un programmeur.

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit. Toute structure hiérarchique de réseaux est évidemment un réseau. [1]

Dans ce chapitre on va commencer par une explication du modèle biologique et neurone formel , Celle-ci est suivie de quelques définitions réseaux de neurones artificiels, Ensuite, nous pourrons étudier comment est architecturé un réseau de neurones artificiels, les types d'apprentissage des réseaux de neurones et les réseaux de neurones les plus utilisés (perceptron multicouche (PMC)), nous pourrons enfin voir comment créer et manipuler, avec Matlab, de tels réseaux : comment créer le réseau, comment définir ses propriétés, comment l'entraîner, et enfin, comment l'utiliser. (Avec toolbox, sans toolbox)

2. Le neurone

Le neurone est une cellule composée d'un corps cellulaire et d'un noyau. Le corps cellulaire se ramifie pour former ce que l'on nomme les dendrites. Celles-ci sont parfois si nombreuses que l'on parle alors de chevelure dendritique ou d'arborisation dendritique. C'est par les dendrites que l'information est acheminée de l'extérieur vers le soma, corps du neurone. L'information traitée par le neurone chemine ensuite le long de l'axone (unique) pour être transmise aux autres neurones. La transmission entre deux neurones n'est pas directe. En fait, il existe un espace intercellulaire de quelques dizaines d'Angströms (10^{-9} m) entre l'axone du neurone afférent et les dendrites (on dit une dendrite) du neurone efférent. La jonction entre deux neurones est appelée la synapse. [1]

3. Bases Biologiques

Le neurone est le processeur élémentaire du cerveau. Chaque neurone traite l'information qui lui parvient localement, puis transmet aux autres neurones qui lui sont connectés l'information qu'il a traitée. Ces cellules peuvent apprendre en changeant l'intensité de leurs connexions avec d'autres cellules ou détruire ou même créer de nouvelles connexions. Elles peuvent aussi changer leur(s) règle(s) de traitement de l'information. Ce processus de changement est appelé apprentissage et joue un rôle fondamental dans le comportement du neurone. [2]

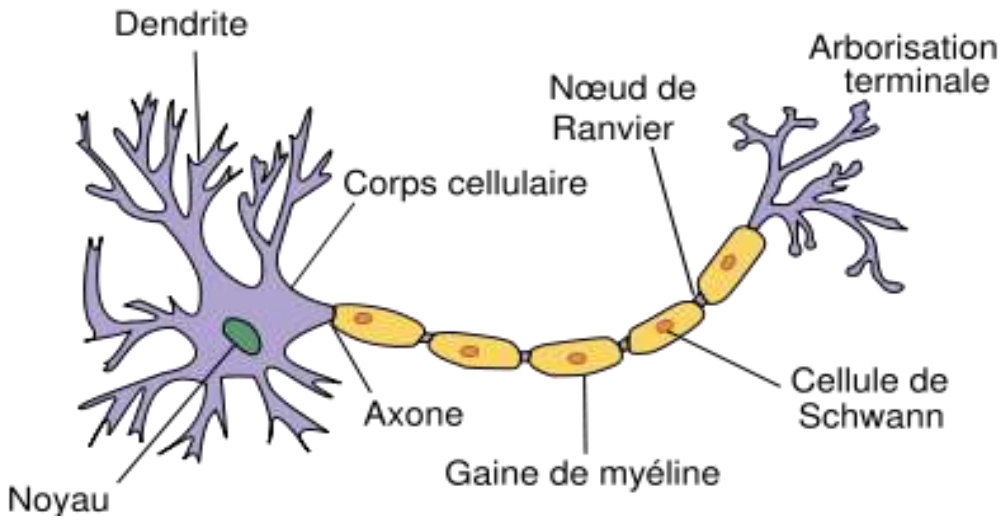


Figure I.1. Représentation d'un neurone réel.

La figure 1 montre la représentation graphique d'un neurone. D'après la figure 1, vous pouvez constater que le neurone est composé des parties suivantes :

- Corps cellulaire : Noyau du neurone.
- Dendrites : Récepteurs principaux du neurone.
- Axone : Fibre nerveuse de transport pour les signaux émis par le neurone.
- Synapse : Connexion entre la dendrite et l'axone de deux neurones [2]

4. Neurone formel (Artificiel)

4.1. Définition

Neurone formel est un processeur élémentaire qui réalise une somme pondérée des signaux qui lui parviennent. La valeur de cette sommation est comparée à un seuil et la sortie du neurone est une fonction non linéaire. [3]

Un neurone est une fonction paramétrée non linéaire de ses variables d'entrée. Naturellement, un réseau de neurones est la composition des fonctions non linéaires de deux neurones ou plus. », [4]

Les variables du neurone sont souvent appelées entrées du neurone et sa valeur est sa sortie.

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit [1]

Les variables du neurone sont souvent appelées entrées du neurone et sa valeur est sa sortie.

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit [1]

4.2. Les paramètres

Les paramètres sont affectés aux entrées des neurones ; la sortie du neurone est une combinaison non linéaire des entrées {xi}, pondérées par les paramètres {Wi}, souvent appelées poids.

Le potentiel v le plus fréquemment utilisé est la somme pondérée des entrées, avec un terme constant supplémentaire appelé « biais », [4]

Soient :

- d = » la dimension de l'espace des entrées.
- Pi = » un vecteur d'entrée.
- w = » un vecteur de poids.
- b = » le biais.
- v = » un coefficient (< 0.05).
- e = » l'erreur.

On peut définir la sortie du réseau ainsi :

$$Y(pi) = \sum_{j=1}^{d}(Wj \cdot Pi,j) + b = W \cdot Pi + b \quad (1.1)$$

L'erreur=la cible(n)-la sortie(n)

Les formules à appliquer pour mettre à jour les poids et biais lorsque l'on entre un individu dans le réseau sont donc :

$$W_{new} = W_{old} + 2 \cdot \nu \cdot e(Pi) \cdot Pi \quad (1.2)$$

$$b_{new} = b_{old} + 2 \cdot \nu \cdot e(Pi). \quad (1.3)$$

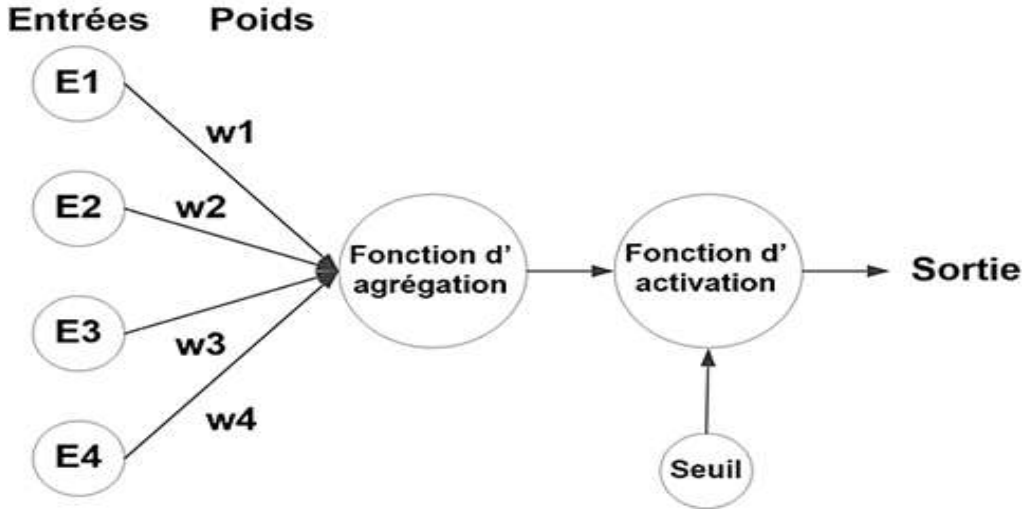


Figure I.2. Modèle de base d'un neurone formel.

Nous pouvons définir un neurone par les trois éléments suivants [2]:

- La fonction d'entrée totale qui définit le prétraitement effectué sur les entrées.
- La fonction d'activation (ou d'état) du neurone qui définit son état interne en fonction de son entrée totale.

* La fonction de sortie qui calcule la sortie du neurone en fonction de son état d'activation.

5. Principe de fonctionnement

Un neurone reçoit des entrées et fournit une sortie, grâce à différentes caractéristiques:

- Des poids accordés à chacune des entrées, permettant de modifier l'importance de certaines par rapport aux autres.
- Une fonction d'agrégation, qui permet de calculer une unique valeur à partir des entrées et des poids correspondants.
- Un seuil (ou biais), permettant d'indiquer quand le neurone doit agir.
- Une fonction d'activation, qui associe à chaque valeur agrégée une unique valeur de sortie dépendant du seuil. [5]

6. Architectures des réseaux de neurones

L'architecture d'un réseau de neurones définit son fonctionnement et joue un rôle important dans son comportement. Elle est en fonction de ses couches et de la structure

des connexions de ses neurones ; ces paramètres permettent de distinguer les différentes classes et/ou types d'architecture neuronales. [6]

Donc Il existe trois grands types d'architectures de réseaux de neurones :

6.1. Réseaux statiques

Ce type de réseaux est organisé généralement en couches de neurones. Chaque neurone d'une couche reçoit ses entrées à partir des neurones de la couche précédente ou tout simplement de l'entrée du réseau. Dans tels réseaux il n'existe pas de feed-back (boucles de retour d'informations). Le traitement des données se fait en sens unique et le flux d'informations circule directement de la couche d'entrée à la couche de sortie ; le traitement est donc réalisé en boucle ouverte. Ces réseaux peuvent être utilisés pour les problèmes de classification ou d'approximation des fonctions.

6.2. Réseaux dynamiques

Les réseaux dynamiques ou bien récurrents, sont les réseaux pouvant comporter des boucles (feed-back) entre les neurones. En général, la sortie de chaque neurone peut être envoyée vers l'entrée de tous les autres neurones du réseau. Ainsi, ces boucles ramènent l'information en provenance de la couche de sortie sur la couche d'entrée simultanément avec le signal d'entrée présent au même instant.

Un réseau dynamique peut donner une sortie différente en lui présentant la même entrée à des instants différents, contrairement aux réseaux statiques qui donnent la même sortie pour une même entrée. Au début, ces réseaux étaient souvent utilisés pour les problèmes de classification et de mémorisation. Actuellement, dans plusieurs travaux la structure interne dynamique de ces réseaux est exploitée pour l'identification ou la commande des systèmes dynamiques.

7. Apprentissage des neurones artificiels

L'apprentissage est vraisemblablement la propriété la plus intéressante des réseaux neuronaux. Elle ne concerne cependant pas tous les modèles, mais les plus utilisés.
Une fois qu'un réseau a été structuré pour une application particulière, ce réseau est prêt à être formé. Pour démarrer ce processus, les poids initiaux sont choisis de manière aléatoire. Ensuite, l'apprentissage commence.

7.1. Définition

L'apprentissage est la modification des poids du réseau dans l'optique d'accorder la réponse du réseau aux exemples et à l'expérience. Il est souvent impossible de décider à

priori des valeurs des poids des connexions d'un réseau pour une application donnée. A l'issu de l'apprentissage, les poids sont fixés : c'est alors la phase d'utilisation.

Au niveau des algorithmes d'apprentissage, Il existe deux approches d'apprentissage :

7.2. Apprentissage supervisé

Apprentissage supervisé implique un mécanisme fournissant au réseau le résultat souhaité, soit en "évaluant" manuellement les performances du réseau, soit en fournissant les résultats souhaités avec les entrées. La grande majorité des réseaux utilisent l'apprentissage supervisée.

Les coefficients synaptiques sont évalués en minimisant l'erreur (entre sortie souhaitée et sortie obtenue) sur une base d'apprentissage. [7]

7.3. Apprentissage non supervisé

Apprentissage non supervisé est l'endroit où le réseau doit donner un sens aux entrées sans aide extérieure, il est utilisé pour effectuer une caractérisation initiale des entrées. On ne dispose pas de base d'apprentissage. Les coefficients synaptiques sont déterminés par rapport à des critères de conformité : spécifications générales. [7]

8. Perceptron

8.1. Perceptron d'une seule couche

Avant d'aborder le comportement collectif d'un ensemble de neurones, nous allons présenter le Perceptron (un seul neurone) en phase d'utilisation. L'apprentissage ayant été réalisé, les poids sont fixes. Le neurone de la figure 3 réalise une simple somme pondérée de ses entrées, compare une valeur de seuil, et fourni une réponse binaire en sortie.

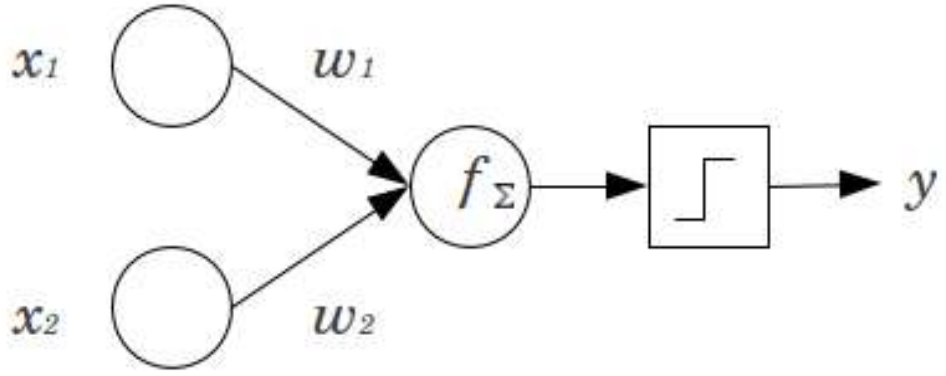


Figure I.3. Le Perceptron : structure et comportement.

8.2. Perceptron multicouche

D'une façon générale, un perceptron multicouche (Multi Layer Perceptron) contient une couche d'entrée, une ou plusieurs couches cachées, chargées des calculs possédant une fonction d'activation et une couche de sortie. La couche d'entrée représente toujours

une couche virtuelle associée aux entrées du système, elle ne contient aucun neurone tandis que les couches suivantes sont des couches de neurones. La couche de sortie correspond aux sorties du système. [8]

Dans le modèle du Perceptron Multi–Couches, les perceptrons sont organisés en couches. Les perceptrons multi–couches sont capables de traiter des données qui ne sont pas linéairement séparables Avec l'arrivée des algorithmes de rétro propagation, ils deviennent le type de réseaux de neurones le plus utilisé.

L'utilité de plusieurs couches cachées n'a pas été démontrée. La Figure 4 illustre la structure d'un PMC présentant quatre neurones en entrée, trois neurones sur la couche cachée et deux en sortie. [9]

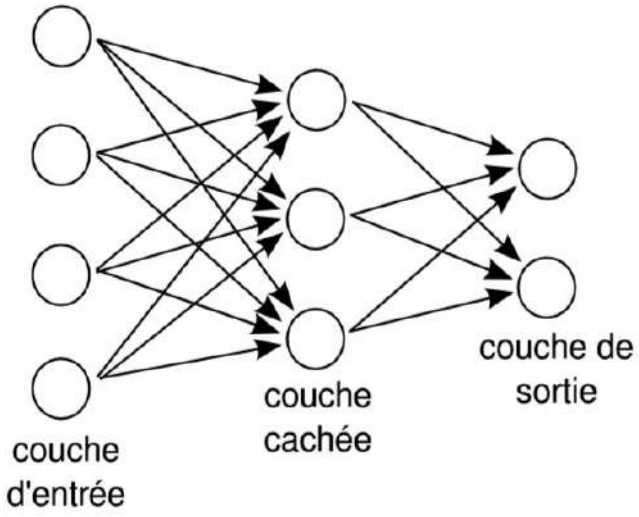


Figure I.4. Modèle du Perceptron Multi–Couches.

Lorsque tous les neurones d'une couche sont connectés aux neurones de la couche suivante, on parle alors de couches complètement connectées. Les équations de propagation décrites plus haut s'appliquent à tous les neurones.

Cependant, le passage d'une couche à l'autre peut être formalisé sous forme matricielle. Soit un PMC dont le nombre de neurones sur la couche d'entrée est n0, n1 sur la couche cachée et n2 sur la couche de sortie. Les poids de la couche cachée peuvent s'écrire sous la forme d'une matrice $W \in Rn1 \times n0$.

Pour un vecteur $X \in Rn0$ présenté en entrée, le vecteur potentiel V et le vecteur de sortie Y pour la couche cachée s'écrivent donc :

$$V = WX \tag{1.4}$$

$$Y = \Phi(V) \tag{1.5}$$

Avec Φ une fonction d'activation. La sortie de la couche cachée devient ensuite l'entrée de la dernière couche [9]

8.2.1. Rétro propagation du gradient pour un Perceptron Multi–Couches

La rétro propagation du gradient s'applique de manière récursive de la couche de sortie à la couche d'entrée. L'initialisation se fait donc par le calcul de l'erreur à la sortie de la dernière couche. Soit l'échantillon d'apprentissage X ∈ Rn0, et le vecteur de sortie désiré D ∈ Rn2. L'erreur sur la couche de sortie peut être définie comme précédemment :

$$L = \frac{1}{2} \| Y2 - D \|^2 \tag{1.6}$$

Où le vecteur Y2 représente la sortie du PMC (i.e. la sortie de la couche 2). De manière immédiate, le gradient de l'erreur en sortie peut être calculé :

$$\frac{\partial L}{\partial Y2} = Y2 - D \tag{1.7}$$

La rétro propagation du gradient s'applique alors récursivement de la couche de sortie à la couche d'entrée. Pour une couche dont X est le vecteur d'entrée, V le potentiel, W la matrice de poids et Y la sortie, la rétro propagation s'écrit donc :

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial V} \tag{1.8}$$

$$= \phi'(V) \frac{\partial L}{\partial Y} \tag{1.9}$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial V} \frac{\partial V}{\partial X} \tag{1.10}$$

$$= W^T \frac{\partial L}{\partial V} \tag{1.11}$$

8.2.2. Mise à jour des poids

Une fois la rétro propagation du gradient effectuée pour toutes les couches, les matrices de poids sont mises à jour :

$$W \leftarrow W - \lambda dW \tag{1.12}$$

Où λ est le taux d'apprentissage. [9]

9. Modèle Adaline

Au début des années 60 B. Widrow et M.E. Hoff ont proposé un système adaptatif, qu'ils ont appelé adaline (de l'anglais adaptive linear élément).

La structure de l adaline diffère du perceptron par l'utilisation d'une seule cellule d'association et utilisation d'une fonction de seuil diffèrent de celle de Heaviside (-1 et t+1).

De plus, il utilise un algorithme adaptif pour mesurer l'écarte entre la sortie réelle et la sortie du processeur élémentaire. Le schéma de l adaline est représenté en figure 5. Le plus souvent, les entrées sont binaires et la réponse souhaitée est également binaire.

L'adaline est le seul réseau de neurones artificiels utilisé massivement dans l'industrie, ce circuit est en effet utilisé dans la télécommunication pour améliorer le signal sur bruit en prenant en compte la variation de l'impédance des différentes lignes téléphoniques. [11]

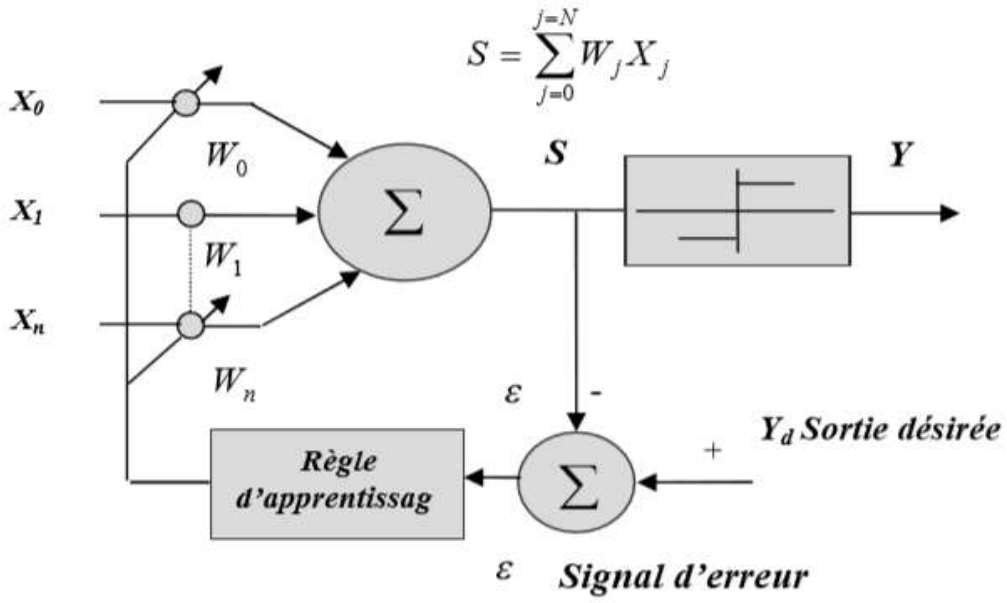


Figure I.5. Schéma de principe De l'adaline

- **Prédiction du signal avec Adaline**

Nous allons ici illustrer comment un réseau de type adaline peut être utilisé. Son point fort est qu'il possède la capacité de s'adapter à tout signal d'entrée. [10]

a/- Création d'un réseau Adaline sous Matlab avec toolbox :

Nous allons définir un signal que nous échantillons à une fréquence de 40Hz. Le but de ce réseau est connaissant 5 valeurs successives de notre signal, de prédire la valeurSuivante en utilisant le toolbox du Matlab2017b. [10]

Le script Matlab utilisé est le suivant :

```
clc;
clear all;
close all;
time = 0:0.025:5;
```

```
T = sin (time*4*pi) + sin (time *3* pi);
plot (time, T)
P = zeros (5, Q);
P (1 ,2 : Q) = T (1 ,1:( Q -1)) ;
P (2 ,3 : Q) = T (1 ,1:( Q -2)) ;
P (3 ,4 : Q) = T (1 ,1:( Q -3)) ;
P (4 ,5 : Q) = T (1 ,1:( Q -4)) ;
P (5 ,6 : Q) = T (1 ,1:( Q -5)) ;
Net = enclin (P, T) ;
a = sim (net, P);
plot (time, a, time, T)
```

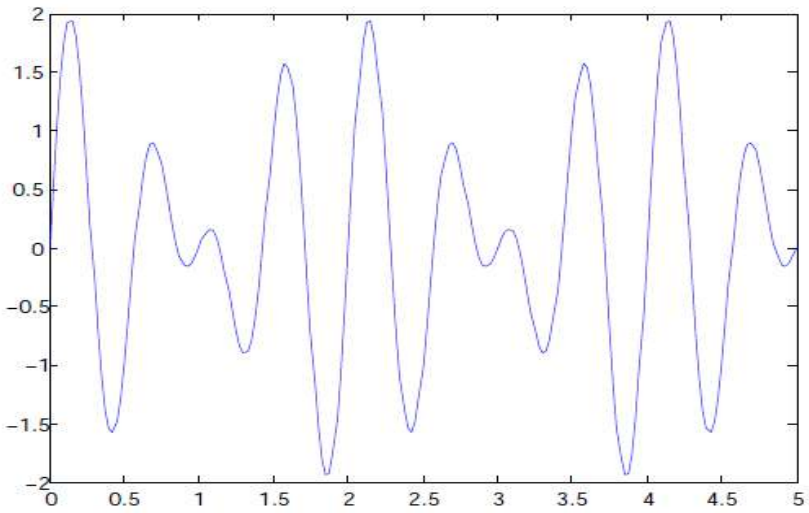


Figure I.6. Signal fourni en apprentissage au réseau adaline

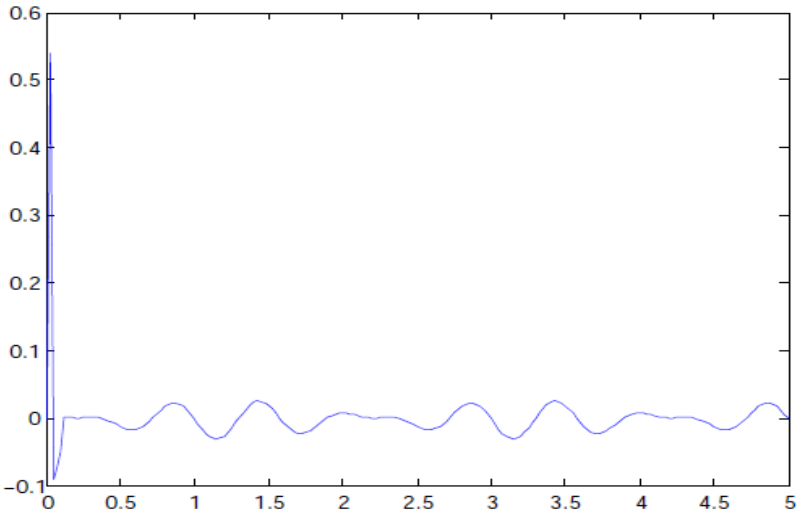


Figure I.7. Erreur entre le signal initial et le signal prédit par le réseau de neurones

b/- Création d'un réseau Adaline sous Matlab sans toolbox :

L'entrainement d'un réseau de neurone Adaline peut être effectué aussi par un programme Matlab sans utiliser le toolbox comme montré par l'exemple suivant :

```
clc; clear all; close all;
T = [0 :0.001 :5] ; y = zeros (size(t));
T = sin (t*4*pi) + sin (t*3* pi) ;
N = size(t,2) ; P = 4 ; x = convmtx (t, P) ; x = x(:,1: N);
eps = ones(size(t));
v = 0.004;w = rand (1, P)/8;
for n=1: N
y(n) = w*x(:,n);
eps(n) = T(n)-y(n);w = w + 2*V*eps(n)* x(:,n)';
end
figure (1)
plot (t,T,'b',t,y,'-r' ),grid,title('target and predicted
signal'); xlabel('time[sec]')
figure(2);plot(t,eps),grid,title('prediction')
```

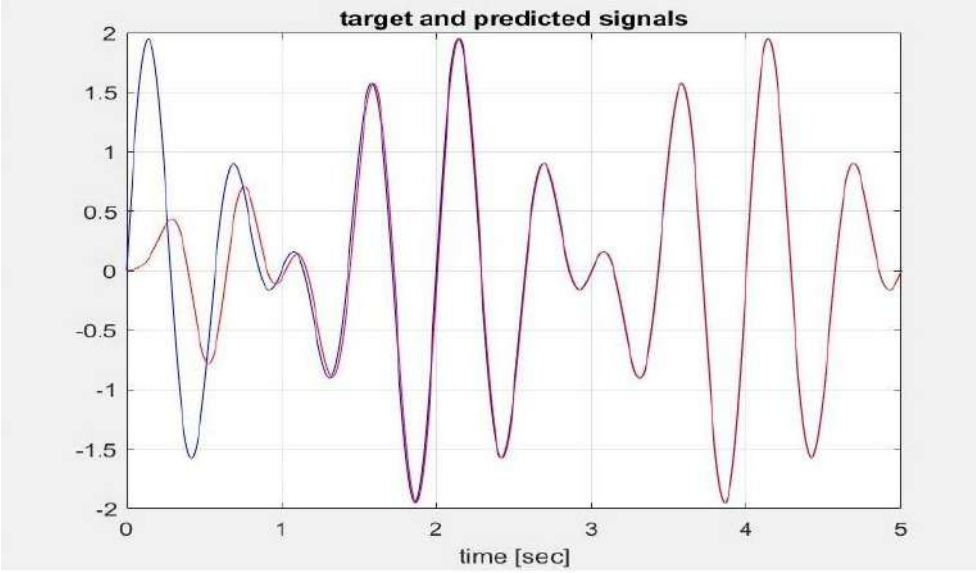


Figure I.9. Signal fourni en apprentissage au réseau adaline

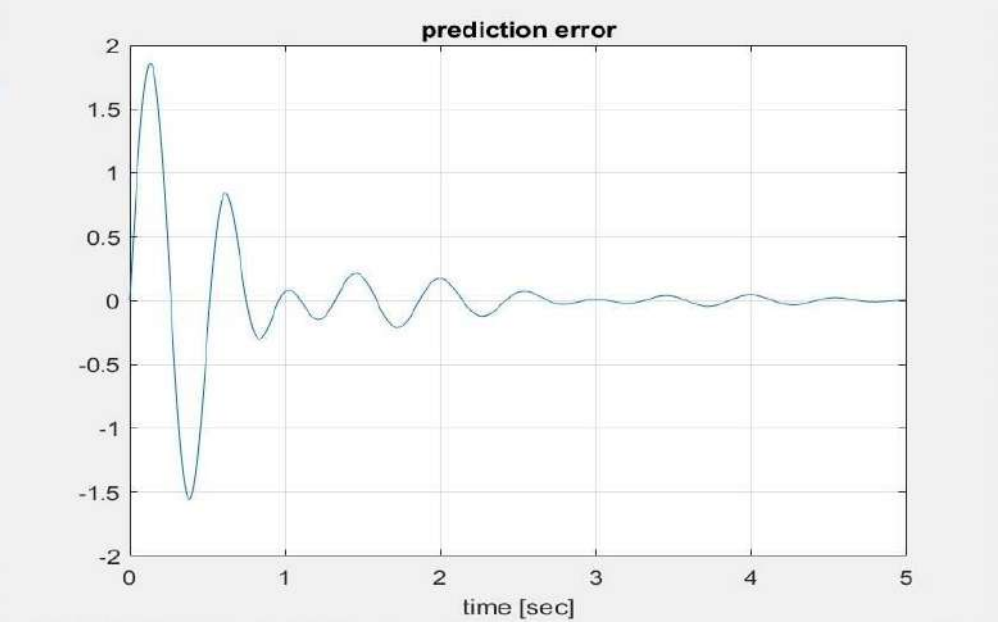


Figure I.10. Erreur entre le signal initial et le signal prédit par le réseau de neurones

10. conclusion :

A partir du fonctionnement du cerveau humain et du modèle neuronal biologique simple, les chercheurs ont arrivé à construire des modèles neuronaux artificiels plus complexes.

Les réseaux de neurones artificiels, sont le cœur de plusieurs applications réelles, un outil indispensable qui a fait ses preuves dans la pratique par leurs caractéristiques de généralisation et de robustesses face au bruit qui font rêver les chercher qui veulent avoir le parfait.

Dans ce chapitre nous avons donné une description sur les réseaux de neurones artificiels, les bases biologiques et les différentes architectures des réseaux de neurones. Nous avons aussi présenté une définition de l'apprentissage des réseaux de neurones, puis nous avons parlé sur le perceptron multicouche, Ensuite, on a fait un exemple sur Matlab d'un réseau Adaline une fois avec toolbox et la deuxième fois sans l'utiliser.

Chapitre II : Circuits programmable FPGA

1.Introduction

L'idée de base du matériel programmable est d'avoir un circuit générique où la fonctionnalité peut être programmé pour une application particulière. Il représente la fonctionnalité en tant que circuit, où le celui-ci peut être programmé pour répondre aux exigences d'une application. La distinction clé est que la fonctionnalité est implémentée comme un système parallèle plutôt que séquentiel. Toute fonction logique peut être implémenter à l'aide d'une représentation à deux niveaux (OR ou AND). Par conséquent, il était logique que les anciens dispositifs logiques programmables consistaient en une architecture matricielle programmable à deux niveaux, Field Programmable Gate Array ' FPGA ' est une percée technologique actuelle dans le domaine de la conception de système embarqué. Grâce à ses caractéristiques remarquables, telles que vitesse, parallélisme, coût, efficacité et capacités reconfigurables dans notre projet, nous préférions l'utiliser car l'utilisation du FPGA pour l'implémentation de réseaux neuronaux offre une flexibilité dans les systèmes programmables.

2. Les circuits programmable FPGA

Définition :

Un FPGA est un appareil contenant une matrice de circuits logiques reconfigurables en réseau de portes. Lorsqu'un FPGA est configuré, le circuit interne est connecté de manière à créer une implémentation matérielle de l'application software. Contrairement aux processeurs, les FPGA utilisent un matériel dédié pour la logique de traitement et ne disposent pas de système d'exploitation. Les FPGA sont véritablement parallèles par nature, de sorte que différents traitements ne doivent pas concurrencer pour les mêmes ressources. En conséquence, les performances d'une partie de l'application ne sont pas affectées par l'ajout de traitements. En outre, plusieurs boucles de contrôle peuvent être exécutées sur un même périphérique FPGA à des vitesses différentes. Les systèmes de contrôle basés sur FPGA peuvent appliquer une logique de verrouillage critique et peuvent être conçus pour empêcher un opérateur de forcer les E / S. Cependant, contrairement aux conceptions de carte de circuit imprimé (PCB) câblées qui ont des ressources matérielles fixes, les systèmes basés sur FPGA peuvent littéralement recâbler leurs circuits internes pour permettre la reconfiguration après le déploiement du système de contrôle sur le terrain. Les dispositifs FPGA offrent les performances et la fiabilité d'un circuit hardware dédié.

Un seul FPGA peut remplacer des milliers de composants discrets en incorporant des millions de portes logiques dans une seule puce de circuit intégré (IC). Les ressources internes d'une puce FPGA sont constituées d'une matrice de blocs logiques configurables (CLB)

entourés d'une périphérie de blocs d'E / S illustrée à la Figure II.1 Les signaux sont acheminés au sein de la matrice FPGA par des commutateurs d'interconnexion programmables et des routes filaires. [12]

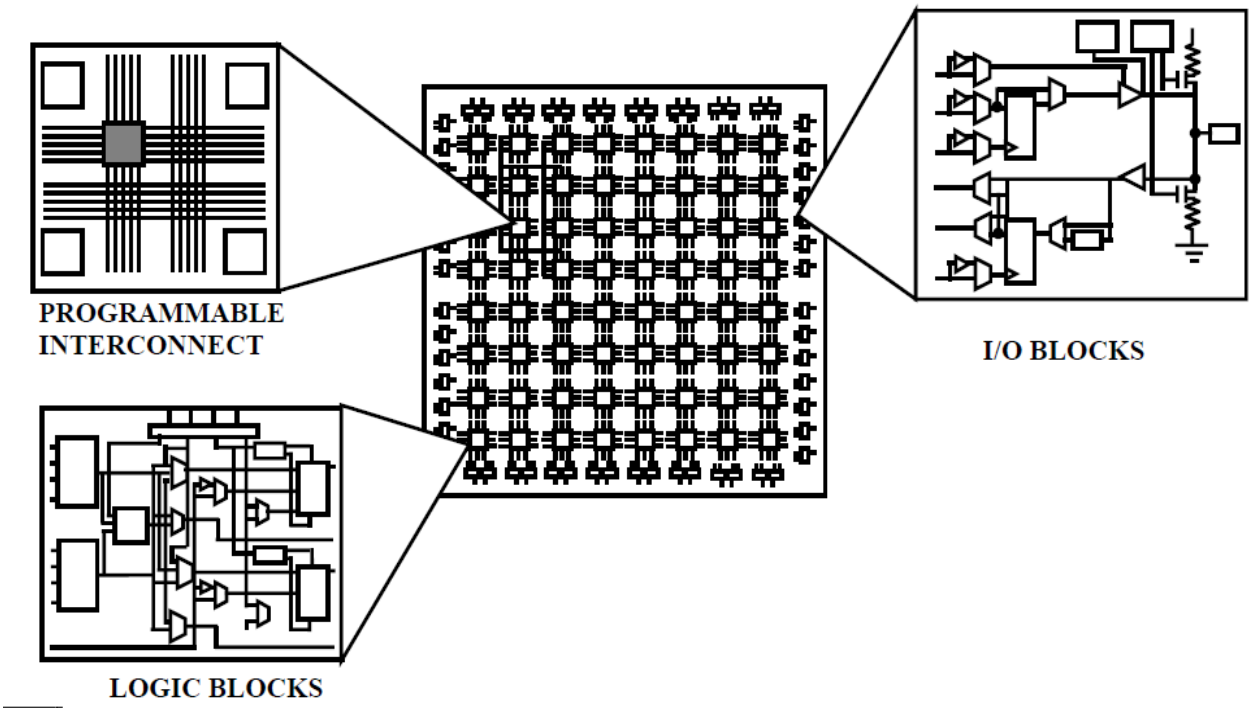


Figure II.1 Structure interne du FPGA [12]

Dans un FPGA, les blocs logiques sont implémentés à l'aide des portes d'entrée multiples à faible niveau de fan-in, ce qui lui confère une conception plus compacte par rapport à une

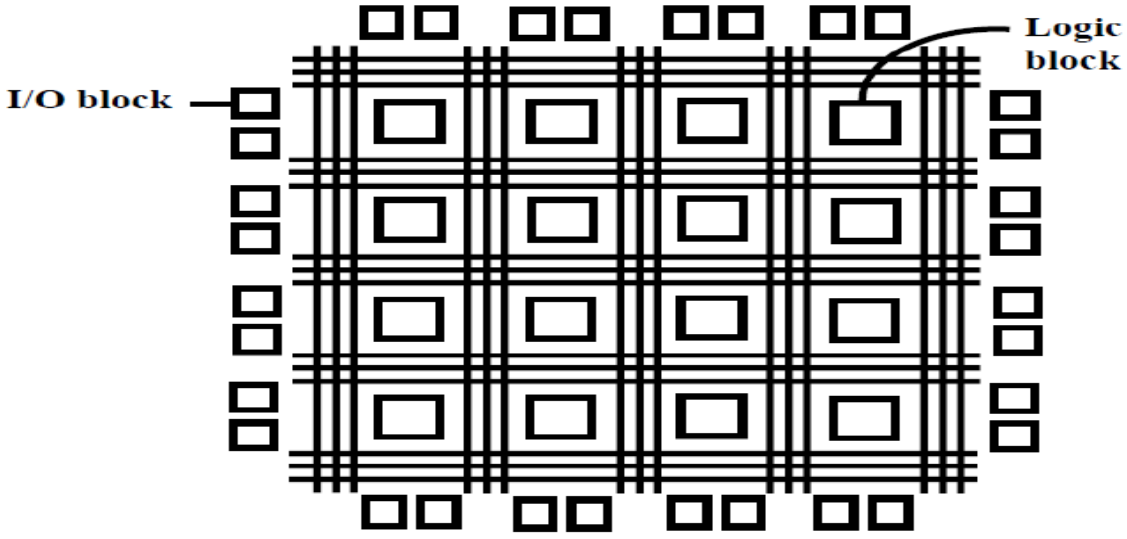


Figure II.2 Structure interne simplifiée du FPGA [12]

implémentation avec une logique AND-OR à deux niveaux. FPGA fournit à son utilisateur un moyen de configurer : L'intersection entre les blocs logiques AND et La fonction de chaque bloc logique. [1]

Le bloc logique d'un FPGA peut être configuré de manière à fournir une fonctionnalité aussi simple que celle d'un transistor ou aussi complexe que celle d'un microprocesseur. Il peut être utilisé pour mettre en œuvre différentes combinaisons de fonctions logiques combinatoire et séquentielle. [13]

3. Structure interne

La figure II.2 illustre la structure de base et les composants essentiels d'un FPGA générique. La logique programmable consiste en un ensemble de blocs à grain fin utilisés pour implémenter la logique de l'application. Ceci est parfois appelé la structure du FPGA. Les blocs logiques sont généralement basés sur une architecture de (LUT), leur permettant de mettre en œuvre toute fonction arbitraire des entrées. Les blocs logiques sont généralement disposés en mosaïque dans une structure en grille et interconnectés via une matrice de routage programmable qui permet aux blocs d'être connectés dans des configurations arbitraires. Les blocs d'entrée et de sortie (E / S) interface entre les composants internes ou le cœur du FPGA et les périphériques externes. Le routage signifie que pratiquement n'importe quel signal peut être acheminé vers n'importe quelle broche d'E / S du périphérique. [12]

Outre ces fonctionnalités de base, la plupart des FPGA fournissent une forme de synchronisation d'horloge permettant de contrôler la synchronisation d'un signal d'horloge par rapport à une source externe.

Un réseau de distribution d'horloge fournit des signaux d'horloge à toutes les parties du FPGA tout en limitant le décalage d'horloge entre les différentes sections d'une conception. Il existe également une logique dédiée pour charger la configuration dans le FPGA. Cette logique ne forme pas directement une partie dans la conception de l'utilisateur, mais c'est la charge supplémentaire nécessaire pour que les FPGA soient programmables et compatibles fournit les mécanismes permettant de charger la conception de l'utilisateur sur un FPGA.Chacune de ces composantes est examinée un peu plus en détail dans les sections suivantes.

3.1 Logique des FPGA

La plus petite unité logique du FPGA est la cellule logique. C'est l'unité logique à grains la plus fine produisant un seul bit en sortie. Il se compose généralement d'une petite LUT (lookup table), table de conversion et d'un verrou sur la sortie, comme indiqué dans Figure II.3. Le loquet est utilisé pour enregistrer la sortie. Sur certains FPGA, les deux sorties peuvent être

fournies, comme dans Figure II.3; sur d'autres, un multiplexeur peut être utilisé pour choisir si le signal enregistré ou non enregistré est sorti. Les registres peuvent être utilisés pour construire des machines à états finis, des compteurs et pour enregistrer des données dans les systèmes en pipeline.

La LUT peut implémenter toute fonction arbitraire de ses entrées. Des fonctions plus complexes peuvent être mis en œuvre en cascadant plusieurs niveaux de LUT en connectant la sortie, X, à l'entrée de la couche suivante. Les fabricants de FPGA doivent faire un compromis en choisissant la taille des LUT. Envisager de mettre en œuvre une simple porte AND à deux entrées. Avec une grande LUT, une fraction importante de la logique serait inutilisée de plus petites TAL permettraient une utilisation plus importante des ressources. Cependant, lors de l'implémentation complexe fonctions avec de petites LUT, plusieurs cellules logiques devraient être mises en cascade, ce qui peut considérablement augmenter le délai de propagation. Pour des fonctions complexes, il serait préférable d'utiliser une table de correspondance plus grande. Pour équilibrer cela, la surface de silicium de la table croît de manière exponentielle avec le nombre d'entrées est plus efficace pour scinder une fonction complexe en plusieurs LUT. [12]

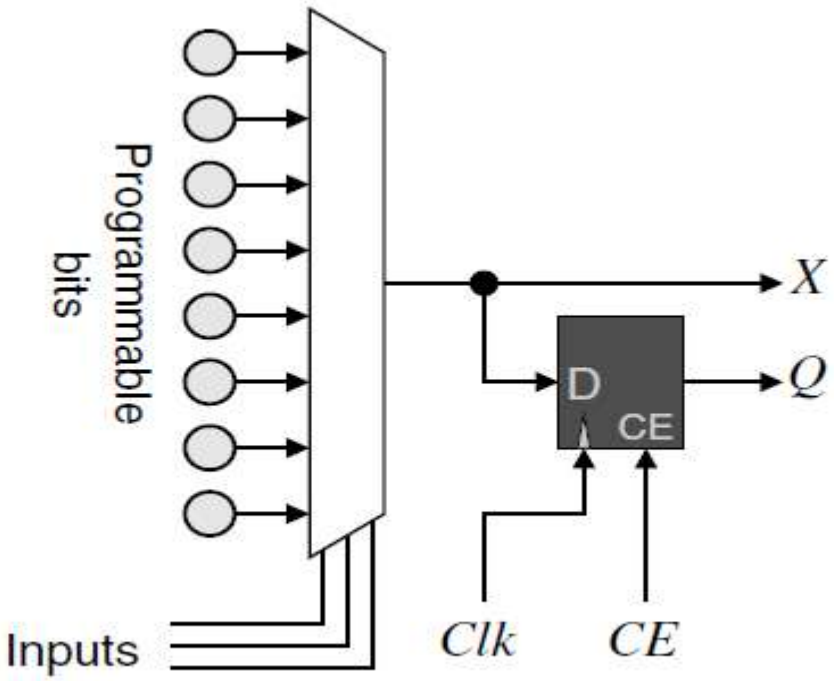


Figure II.3 Une cellule logique est la pierre de fondation d'une FPGA[12]

Les FPGA homogènes à grain fin consistent en une mer de blocs logiques identiques. Cependant, la plupart des FPGA actuels se sont éloignés de cette structure homogène, avec un mélange hétérogène de à la fois des blocs logiques à grain fin et des blocs plus gros à grain grossier destinés à l'accélération couramment utilisés opérations. Cela présente à la fois des

avantages et des inconvénients en termes de conception. Les systèmes hétérogènes seront généralement plus rapides, car les blocs à grains grossiers n'auront pas le même niveau de frais généraux que ceux Cependant, les dispositifs hétérogènes sont également plus difficiles à programmer, et si le mélange des blocs spécialisés n'est pas correct, une grande partie des fonctionnalités peut rester sous-utilisée. [12]

3.2 Interconnexion

L'interconnexion programmable a pour but de permettre aux différents blocs logiques d'être utilisés de manière flexible connecté pour mettre en œuvre la fonctionnalité souhaitée. Il est impossible de fournir un dédié connexion directe de toutes les sorties possibles à toutes les entrées possibles. Dans une application donnée, seul une très petite fraction de ces connexions serait nécessaire ; cependant, aucune des connexions potentiellement peut être requis. La solution consiste à disposer d'un plus petit ensemble de ressources de connexion (lignes de routage) pouvant être utilisées. Partagés de manière flexible pour créer les connexions requises pour une application donnée.

Puisque l'interconnexion prend place sur le silicium, il y a un compromis. Si seulement quelques lignes de routage sont fournies, plus d'espace est disponible pour la logique, mais des circuits complexes ne peuvent pas être construits en raison des ressources d'interconnexion insuffisantes. Alternativement, si trop de lignes de routage sont fournies, beaucoup ne seront pas utilisés, ce qui réduit la zone disponible pour la logique. Il y a un optimum quelque part entre ces extrêmes, où des lignes de routage suffisantes sont disponibles pour la plupart des applications, sans utiliser trop d'espace.

Le réseau de connexions pour la plupart des FPGAs sont basés sur une structure de grille, semblable à celle montré par la Figure II.2. À chaque intersection de grille, une forme de commutateur de traverse permet une connexion programmable entre les lignes de routage horizontale et verticale.

Un autre facteur qui doit être considéré est le retard de propagation par le réseau d'interconnexion. Chaque commutateur que le signal passe à travers ajoutera au retard. Cela a conduit à une structure segmentée, où pas toutes les lignes de routage est commuté à chaque jonction. En outre, des connexions directes spécifiques peuvent être fournies entre les blocs logiques adjacents pour réduire le nombre de signaux qui doivent passer par la matrice interconnecte. Tout en ayant une gamme de différentes ressources d'interconnexion améliore la flexibilité et la vitesse, il peut rendre le processus de trouver de bonnes routes nettement plus complexes, en particulier pour les dessins qui utilisent la plupart de la logique disponible sur la FPGA.

Il est important qu'une seule sortie pilote une ligne de routage. Si deux sorties sont connectées ensemble avec un haut et l'autre bas, l'alimentation sera effectivement court-circuitée, endommageant la puce. L'un des rôles des outils de programmation FPGA est de veiller à ce que cela ne se produise jamais. Périphériques utilisant le bus les structures de routage doivent limiter le courant des transistors de commande où utiliser un drain ouvert et un pull-up passif. Les deux approches affectent la vitesse et la dissipation de puissance. [12]

3.3 Entrée et sortie

Les blocs d'entrée et de sortie permettent la connexion de signaux entre le FPGA et des périphériques externes. La flexibilité des FPGA exige qu'ils soient capables de se connecter à un large éventail d'autres composants. La plupart des broches d'entrée / sortie peuvent être programmées comme entrées, sorties ou les deux et être connectées directement à la mémoire interne logique du FPGA, ou passer par les registres. Cette gamme de fonctionnalités peut être fournie par un périphérique d'E / S générique. Tel que celui illustré à la figure II.4, chaque multiplexeur étant contrôlé par un programmable bit de configuration. La principale complexité des E / S provient du grand nombre de signaux standards différents utilisées par différents appareils et en assurant une synchronisation appropriée entre le FPGA et les composants hors puce.

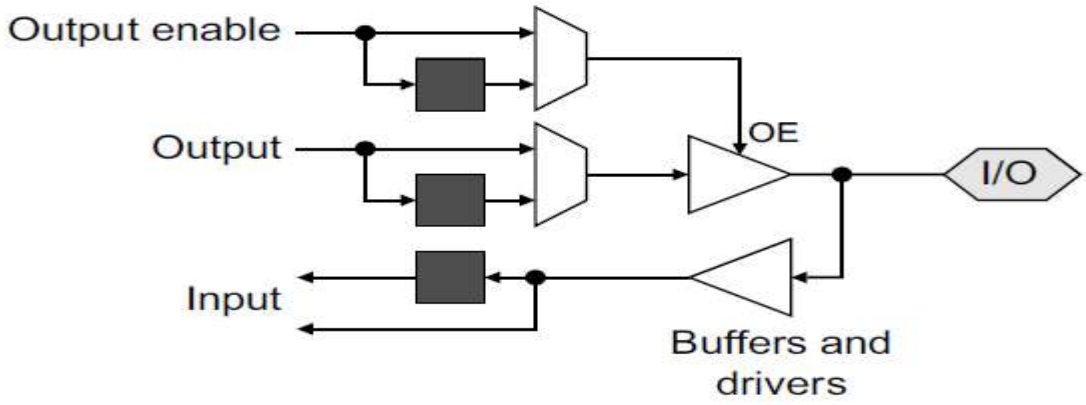


Figure II.4 Bloc d'entrée / sortie générique [12*]*

Lors de l'interfaçage de signaux haute vitesse, il est important de prendre en compte les effets sur la ligne de transmission. En règle générale, toute connexion plus longue qu'un huitième environ du temps de montée doit avoir la terminaison appropriée pour minimiser les réflexions du signal. Les FPGA étant devenus plus grands et plus rapides, il a devenu pratique courante de fournir une terminaison programmable sur puce. Si non fourni, il est nécessaire d'assurez-vous que la terminaison appropriée est construite hors puce pour des signaux de commutation rapides. De nombreux protocoles de communication de haut niveau nécessiteraient d'importantes ressources FPGA simplement pour gérer le protocole. Bien que

cela puisse être mis en œuvre dans la structure du FPGA, la conception est souvent assez complexe et est mieux implémenté en utilisant une puce d'interface dédiée. Ces puces gèrent généralement l'accès aux médias (MAC) et la signalisation physique (PHY) ne nécessitant que des signaux logiques standard dans leur interface avec le FPGA. Certains FPGA intègrent des circuits MAC et PHY dédiés aux communications couramment utilisées protocoles, tels que PCI Express et Gigabit Ethernet. La plupart des FPGA offrent une gamme de connexions d'E / S générales et dédiées. [12]

3.4 Horloge

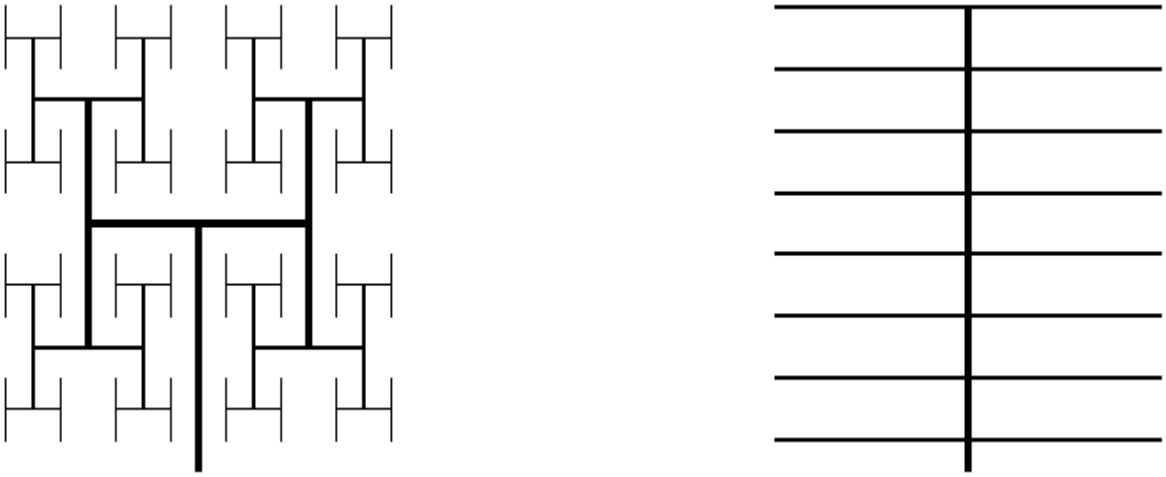


Figure II.5 Réseaux de distribution d'horloge. À gauche : structure en H ; à droite : colonne vertébrale et structure des côtes. [12]

Les FPGA sont conçus principalement comme des dispositifs synchrones, ils nécessitent donc un signal d'horloge. Bien que multiples horloges puissent être utilisées dans une conception, les registres, les mémoires synchrones et les blocs d'E / S ne peuvent être contrôlés que par une horloge. Un domaine d'horloge comprend tous les circuits contrôlés par une horloge particulière. Étant donné qu'une horloge pilote potentiellement de nombreuses entrées sur l'ensemble du FPGA, il existe deux options spéciales. Caractéristiques des lignes d'horloge qui les distinguent des autres interconnexions d'un FPGA. Tout d'abord, c'est important que tous les registres d'un domaine d'horloge soient synchronisés simultanément. Le décalage d'horloge est la différence heure d'arrivée du signal d'horloge aux registres successifs d'un dessin (Friedman, 2001). Tout biais réduira la marge de synchronisation et impact sur la vitesse d'horloge maximale d'une conception. Bien que l'asymétrie puisse être minimisée en utilisant des réseaux de distribution d'horloge H-tree ou X-tree, ceux-ci ne correspondent pas bien à la structure en grille de FPGA (Lamoureux etWilton, 2006). Au lieu de cela, une structure de colonne vertébrale et de côtes est généralement utilisée (Figure II.5), où l'horloge est distribuée aux rangées à l'aide d'une épine puis aux éléments de la rangée à l'aide des nervures (Lamoureux

et Wilton, 2006). Une série de mémoires tampons est utilisée dans le réseau de distribution pour gérer les grande fan-out. En pratique, il est un peu plus complexe que cela, car le réseau doit prendre en charge plusieurs domaines d'horloges, et de nombreux FPGA ont également un mélange d'horloges locales et globales. [12]

3.5 Configuration

Tous les commutateurs, structures et options du FPGA sont contrôlés par des bits de. Plus généralement, les données de configuration sont contenues dans le FPGA dans des cellules de mémoire statique (SRAM), bien que les appareils à programmation flash sont disponibles chez certains fabricants. Les FPGA programmés en flash sont non volatiles, conserver leur configuration après la mise hors tension, afin qu'ils puissent immédiatement commencer à travailler après mise sous tension. Les FPGA à mémoire statique perdent leur configuration à la mise hors tension, ce qui nécessite la configuration fichier à recharger chaque fois que le système est allumé. La mémoire statique avait l'avantage d'être infiniment reprogrammable ; La mémoire flash a généralement un nombre limité de cycles d'écriture.

Un FPGA est configuré en chargeant le fichier de configuration dans la mémoire interne. Ceci est généralement accompli en transmettant le fichier de configuration sur le FPGA soit en série, soit en parallèle 8, 16 ou en certains cas 32 bits à la fois. Dans un système autonome, le fichier de configuration est généralement contenu dans un fichier. Mémoire flash externe, telle que celle illustrée à la figure II.6, avec la carte configurée pour configurer automatiquement le FPGA au démarrage. Les FPGA peuvent également être programmés en série via un JTAG (Joint Test Action Group) interface, et cette interface est couramment utilisée lors de la programmation de FPGA à partir d'un système informatique hôte. La nécessité de transférer la configuration dans le FPGA chaque fois que le système est alimenté expose le fichier de configuration rendant la propriété intellectuelle contenue dans vulnérable au piratage ou au retournement ingénierie. Les FPGA haut de gamme surmontent cette limitation en chiffrant le fichier de configuration et déchiffrer la configuration une fois qu'elle est dans le FPGA. Certains FPGA prennent également en charge la relecture de la configuration à partir du FPGA. Cette fonctionnalité peut être utile pour déboguer une conception avec les outils appropriés, mais n'est généralement pas utilisée en fonctionnement normal. [12]

3.6 Consommation D'énergie

Dans les applications intégrées, la consommation électrique du système, et en particulier du FPGA, est essentielle. La puissance dissipée par un FPGA peut être divisée en deux composantes : la puissance statique et la puissance dynamique. La puissance dynamique est la puissance nécessaire pour commuter les signaux de 0 à 1 et vice versa. Inversement, l'énergie statique est celle qui est consommée sans commutation, simplement en raison du périphérique allumé. La plupart des FPGA sont basés sur la technologie CMOS. Un inverseur CMOS, le circuit CMOS le plus élémentaire, est présenté à la figure II.7. Il se compose d'un transistor de

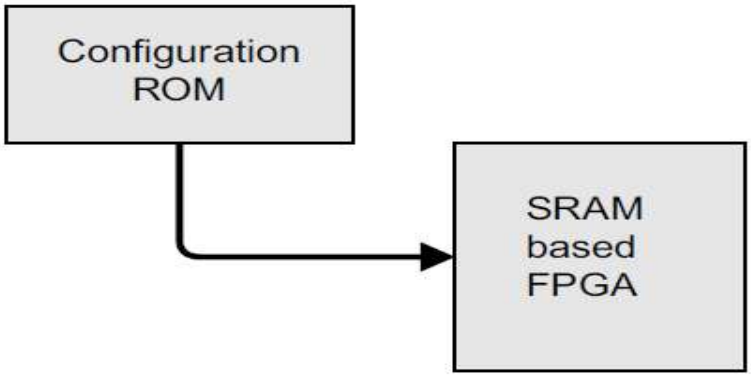


Figure II.6 Configuration de FPGA basés sur la RAM statique [12]

tirage à canal P et d'un transistor à tirage bas à canal N transistor. Un seul des transistors est allumé à la fois et, comme les transistors MOS ont une très haute résistante aux chocs, les circuits CMOS ont généralement une très faible dissipation de puissance statique. [1]

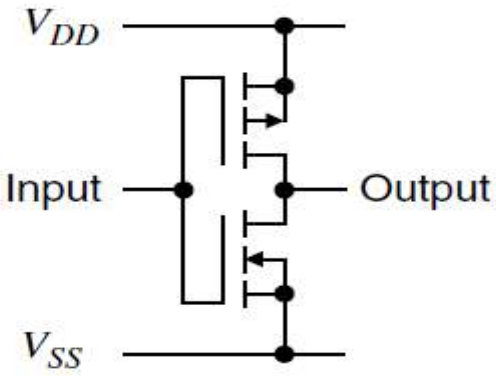


Figure II.7 Inverseur CMOS.

4.Les Familles des FPGA

Dans la section précédente, les caractéristiques architecturales des FPGA ont été examinés d'une manière générale. Dans cette section, les caractéristiques particulières d'appareils sont examinées plus en détail. Les deux principaux fabricants de FPGA en termes de part de marché

sont Xilinx et Altera, bien que plusieurs d'autres qui fournissent des FPGA. Les offres actuelles de chacun de ces éléments sont décrites à leur tour. [12]

- **Xilinx**

Xilinx est le premier fournisseur mondial de technologies et d'appareils programmables. La société va au-delà de la logique programmable traditionnelle pour permettre la programmabilité hardware et software, intégrer des fonctions de signaux mixtes analogiques et numériques et permettre de nouveaux niveaux d'interconnexion programmable dans les circuits intégrés 3D monolithiques et multiples. Les produits Xilinx sont associés à un environnement de conception de nouvelle génération et à l'IP pour répondre à un large éventail de besoins des clients, de la logique programmable à l'intégration de systèmes programmables. [12]

- **zynq**

La famille Zynq-7000 est basée sur l'architecture Xilinx SoC. Ces produits intègrent un système de traitement (PS) ARM Cortex -A9 MPCore riche en fonctionnalités et une logique programmable (PL) Xilinx dans un seul appareil, construit sur une technologie de pointe technologie de processus haute performance, basse consommation (HPL), 28 nm et High-k Gate (HKMG). Les processeurs ARM Cortex-A9 MPCore constituent le cœur du système PS, qui comprend également une mémoire intégrée, des interfaces de mémoire externes et un riche ensemble de périphériques d'E/S. [15]

La famille Zynq-7000 offre la flexibilité et l'évolutivité d'un FPGA, tout en offrant performances, puissance et facilité d'utilisation généralement associées aux ASIC et ASSP. La gamme de périphériques de la gamme de SoC Zynq-7000 permet aux concepteurs de cibler des applications sensibles aux coûts ainsi que des performances élevées à partir d'une plate-forme unique en utilisant des outils standard du secteur. Alors que chaque périphérique de la famille Zynq-7000 contient le même PS, les ressources PL et I / O varient d'un périphérique à l'autre. [15]

L'architecture Zynq-7000 mappe de manière pratique la logique et le logiciel personnalisés dans le PL et le PS respectivement. Il permet la réalisation de fonctions système uniques et différenciées. L'intégration du PS avec le PL offre des niveaux de performance inégalés par les solutions à deux puces (par exemple, un ASSP avec FPGA) en raison de la bande passante d'E / S, du couplage libre et des budgets de puissance limités. [15]

Diagramme

La figure II.10 illustre les blocs fonctionnels du SoC Zynq-7000. Le PS et le PL se trouvent dans des domaines d'alimentation distincts, ce qui permet à l'utilisateur de ces dispositifs de mettre le PL hors tension pour la gestion de l'alimentation, le cas échéant.[15]

Le SoC Zynq-7000 est composé des principaux blocs fonctionnels suivants :

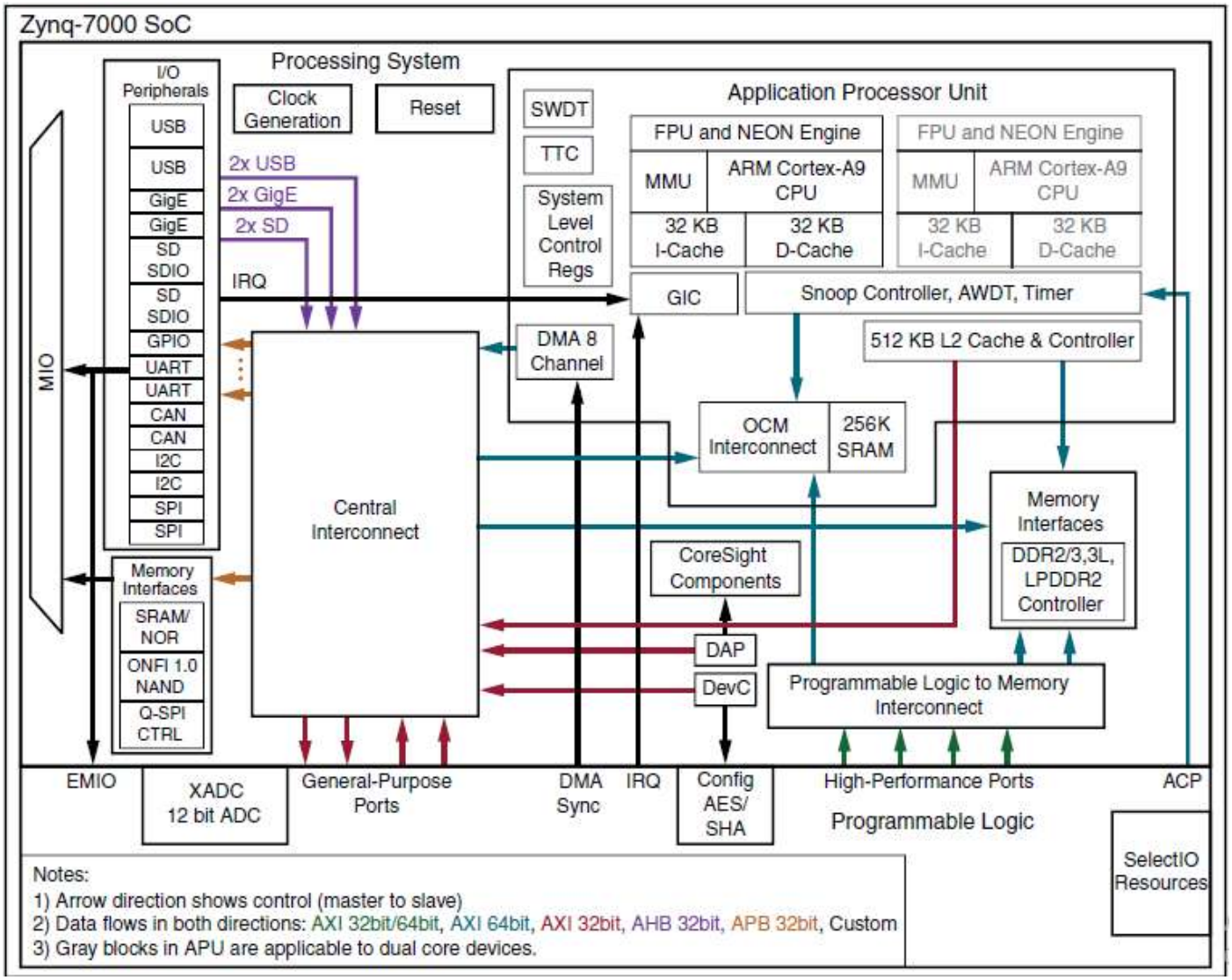


Figure II.8 Présentation du SoC Zynq-7000[15]

- Système de traitement (PS)
- Processeur d'application (APU)
- Interfaces mémoire
- Périphériques d'E / S (IOP)
- Interconnexion
- Logique programmable (PL)

5. Conclusion

L'implémentation d'un réseau de neurone sur FPGA est possible mais des exigences doivent être réfléchies en avance, telle que Précision à utiliser (virgule flottante / virgule fixe / entier) et nombre maximum de neurones dans chaque couche. Si ce nombre n'est pas choisi avec soin, un débordement de mémoire tampon peut survenir et le réseau tout entier va se perdre. Il est un peu difficile d'implémenter des fonctions non linéaires dans le matériel sans un support de librairie approprié. Des options plus simples consisteraient à utiliser des tables de correspondance ou à implémenter une fonction linéaire par morceaux, une fois cette conception de base terminée, l'implémentation devient beaucoup plus facile.

Chapitre III : Etude pratique et implémentation

1. Introduction

Dans ce chapitre, nous présenterons notre travail qui permet la détection des harmoniques d'une fréquence f0 de n'importe quel signal sinusoïdal donné, en utilisant notre architecture du réseau de neurones.

Nous présenterons une comparaison entre l'implémentation de notre algorithme sur une carte FPGA " PINQ Z2 " et son simulation software avec Python.

2. Présentation des outils de développement

2.1. Logiciels utilisés

2.1.1. Python

Python est un langage de programmation de haut niveau interprété (il n'y a pas d'étape de compilation) et orienté objet avec une sémantique dynamique. Il est très sollicité par une large communauté de développeurs et de programmeurs. Python est un langage simple, facile à apprendre et permet une bonne réduction du cout de la maintenance des codes. Les bibliothèques (packages) python encouragent la modularité et la réutilisabilité des codes. Python et ses bibliothèques sont disponibles (en source ou en binaires) sans charges pour la majorité des plateformes et peuvent être redistribués gratuitement. [15]

2.1.2 Vivado Design Suite

Vivado Design Suite est une suite logicielle produite par Xilinx pour la synthèse et l'analyse de conceptions HDL, remplaçant Xilinx ISE avec des fonctionnalités supplémentaires pour le développement de système sur puce et la synthèse de haut niveau. Vivado représente une réécriture et une refonte complète du flux de conception (comparé à ISE) et a été décrit par les critiques comme "bien conçu, étroitement intégré, rapide, évolutif, facile à gérer et intuitif" [16][17][18]

2.1.3 System generator

Xilinx System Generator est un outil de programmation FPGA fourni par Xilinx. Il est spécifiquement axé sur les FPGA Xilinx, permettant aux développeurs de travailler dans l'environnement Simulink et de générer des cœurs paramétrés particulièrement optimisés pour les FPGA Xilinx. L'outil est intégré à Xilinx ISE Design Suite (System Edition) et à Xilinx Vivado HL (System Edition). Par défaut, Xilinx Blockset contient plus de 90 blocs DSP, allant de simples additionneurs, multiplicateurs, etc. à des blocs complexes tels que des blocs de correction d'erreur directe, des FFT, des filtres et des mémoires, etc. De plus, System Generator

inclut également les blocs mcode et Black Box, qui peuvent être utilisés pour intégrer les codes mcode et HDL, respectivement, directement dans l'environnement de conception Simulink.[19]

2.1.4 PYNQ

PYNQ est un projet open source de Xilinx® qui facilite la conception de systèmes intégrés avec les systèmes Xilinx Zynq® sur des puces (SoC).

En utilisant le langage et les bibliothèques Python, les concepteurs peuvent exploiter les avantages de la logique programmable et des microprocesseurs de Zynq pour créer des systèmes embarqués plus performants et plus intéressants.

Les utilisateurs PYNQ peuvent désormais créer des applications intégrées hautes performances avec : exécution matérielle parallèle, traitement vidéo à haute cadence, algorithmes accélérés par le matériel, traitement du signal en temps réel, IO large bande passante et Le contrôle de faible latence.[21]

2.1.5 Jupyter Notebook

Jupyter Notebook est un environnement informatique interactif basé sur un navigateur. Vous pouvez créer des cahiers Jupyter comprenant du code en direct, des widgets interactifs, des graphiques, des textes explicatifs, des équations, des images et des vidéos. Une carte Zynq compatible PYNQ peut être facilement programmée dans Jupyter Notebook à l'aide de Python. En utilisant Python, les développeurs peuvent utiliser des bibliothèques de matériel et des superpositions sur la logique programmable. Les bibliothèques de matériel, ou les superpositions, peuvent accélérer les logiciels s'exécutant sur une carte Zynq et personnaliser la plate-forme matérielle et les interfaces. L'image PYNQ est une image Linux amorçable et inclut le package pynq python et d'autres packages open source. [21]

2.1.6 Overlay

Les overlays sont analogues aux bibliothèques de logiciels. Un programmeur peut télécharger des superpositions dans le Zynq® PL lors de l'exécution pour fournir les fonctionnalités requises par l'application logicielle.

Un overlay est une classe de conception de logique programmable. Les conceptions logiques programmables sont généralement hautement optimisées pour une tâche spécifique. Les overlays sont toutefois conçus pour être configurables et réutilisables pour un large éventail d’applications. Un PYNQ overlay aura une interface Python, permettant à un programmeur de l’utiliser comme tout autre paquet Python. [31]

2.2. Librairies et modules utilisés

2.2.1. Librairie Numpy

NumPy est le paquet fondamental du calcul scientifique avec Python. Il contient entre autres:

* Un puissant objet tableau N-dimensionnel.
* Fonctions sophistiquées (diffusion).
* Algèbre linéaire utile, transformée de Fourier et capacités de nombres aléatoires.

Outre ses utilisations scientifiques évidentes, NumPy peut également être utilisé comme un conteneur multidimensionnel efficace de données génériques. Des types de données arbitraires peuvent être définis. Cela permet à NumPy de s'intégrer de manière transparente et rapide à une grande variété de bases de données. [23]

2.2.2. Librairie Matplotlib

Matplotlib est une libraire Python capable de produire des graphes de qualité. matplotlib peut être utilisé dans des scripts Python, le shell Python et IPython, le notebook Jupyter, des serveurs d'application web et dans quatre outils d'interface graphique. Matplotlib essai de rendre les tâches simples "simples" et de rendre possible les choses compliqués. Vous pouvez générer des graphes, histogrammes, des spectres de puissance (lié à la transformée de Fourier), des graphiques à bares, des graphiques d'erreur, des nuages de dispersion, etc… en quelques lignes de code.

Pour des graphiques simples, le module matplotlib.pyplot fournit une interface comme MATLAB spécialement adaptée avec IPython.[24]

2.2.3. Le Module time

Le module time en python est une des façons les plus simple de manipuler le temps dans un programme. Un temps en python est, par défaut, un nombre représentant des secondes. Ça permet par exemple d'attendre un certain nombre de secondes, d'afficher une date avec un format spécifique. [25]

2.2.4. Le Module Struct

Ce module effectue des conversions entre les valeurs Python et les structures C représentées sous forme d'objets octets Python. Les chaînes de format sont le mécanisme utilisé pour spécifier la disposition attendue lors de la compression et de la décompression des données. [26].

3. Les outils de développements :

3.1. PYNQ-Z2

PYNQ-Z2 est une carte de développement FPGA basée sur le FPGA ZYNQ XC7Z020, spécialement conçue pour prendre en charge PYNQ, un nouveau cadre open source permettant aux programmeurs intégrés d'explorer les possibilités des SoC xilinx ZYNQ sans avoir à concevoir de circuits logiques de programmation. Bénéficiant de la logique programmable et du processeur ARM avancé de ZYNQ, les concepteurs peuvent créer des systèmes embarqués plus puissants. De plus, les SoC peuvent être programmés en Python et le code peut être développé et testé directement sur le PYNQ-Z2. [27]

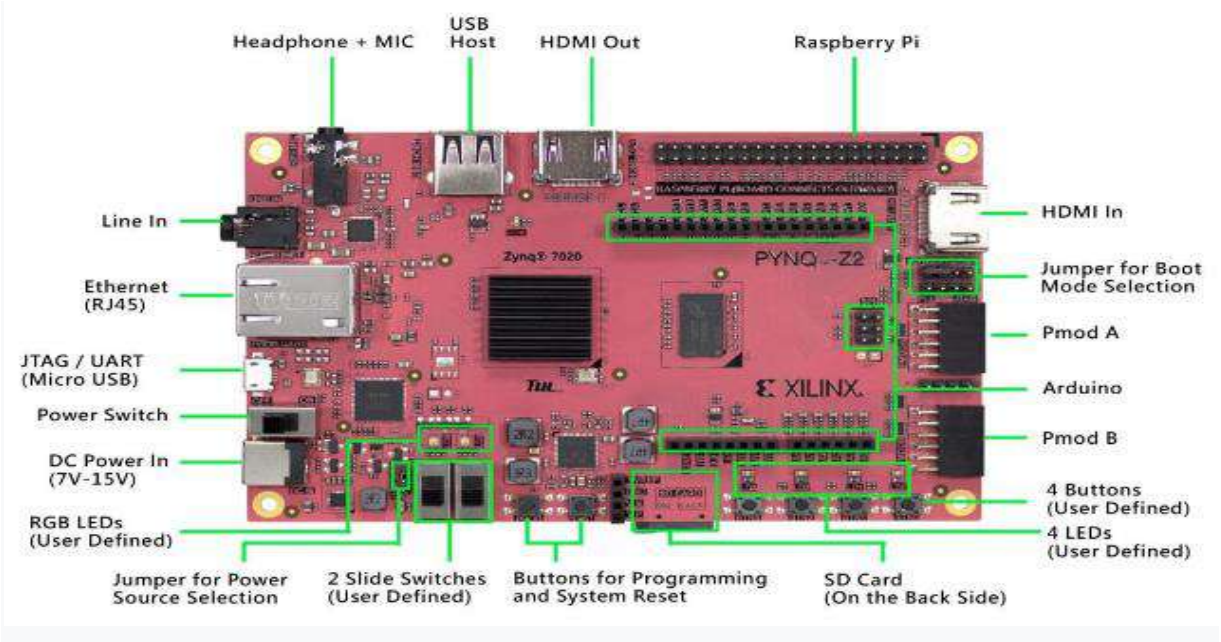


Figure III.1 La carte PynqZ2[27]

3.2. Configuration de la carte

1- Plaçons le cavalier ** Boot ** sur la position SD. (Cela permet à la carte de démarrer à partir de la carte Micro-SD)

2- Pour alimenter la carte à l'aide du câble micro USB, réglons le cavalier d'alimentation sur la position USB.

3- Insérons la carte Micro SD contenant l'image PYNQ-Z2 dans la fente pour carte Micro SD située sous la carte.

4- Connectons le câble USB à Notre PC et au port UART MicroUSB de la carte.

5- Connectons le port Ethernet en suivant les instructions ci-dessous.

6- Allumons le PYNQ-Z2 et vérifions la séquence de démarrage en suivant les instructions ci-dessous. [28]

3.3. Spécifications de produit

Tableau III.1 les spécifications du carte PynqZ2[27]

ZYNQ XC7Z020-1CLG400C • Processeur double cœur Cortex-A9 à 650MHz • Contrôleurs de périphériques à large bande passante : Ethernet 1G, USB 2.0, SDIO • Contrôleur de périphérique à faible bande passante : UART. • Programmable à partir de JTAG, flash Quad-SPI et carte microSD • Logique programmable équivalente à FPGA Artix-7 • 13 300 tranches logiques, chacune avec quatre LUT à 6 entrées et 8 bascules • 630 Ko de blockRAM rapide • 220 tranches DSP • Convertisseur analogique-numérique sur puce (XADC	**USB et Ethernet** • PHY Gigabit Ethernet • Circuits de programmation Micro USB-JTAG • Pont Micro USB-UART • USB 2.0 OTG PHY (supporte uniquement l'hôte) **Mémoire** • DDR3 de 512 Mo avec bus 16 bits à 1050 Mbits / s • fente microSD **Alimentation** • Alimenté par une source d'alimentation externe USB ou 7V-15V

3.4 Allumer le PYNQ-Z2

Comme indiqué à l'étape 6 de l'installation de la carte, plaçons l'interrupteur en position ON pour allumer la carte. La LED rouge s'allumera immédiatement pour confirmer que la carte est sous tension. Après quelques secondes, le voyant jaune / vert / terminé s'allume pour indiquer que le périphérique Zynq® est opérationnel.

Après une minute, Nous devriez voir deux LED bleues ** et quatre ** LED jaunes / vertes clignoter simultanément. Les voyants bleus s'allument et s'éteignent tant que les voyants jaunes / verts restent allumés. Le système est maintenant démarré et prêt à être utilisé.

Une fois que votre tableau est configuré, vous devez vous y connecter pour pouvoir utiliser le Jupyter Notebook.

Nous aurons besoin d'un port Ethernet disponible sur notre ordinateur et des autorisations pour configurer notre interface réseau. Avec une connexion directe, nous pourrons utiliser PYNQ, mais à moins que nous ne puissiez relier la connexion Ethernet de la carte à une connexion Internet de notre ordinateur, notre carte n'a pas accès à Internet. Nous ne pourrons pas mettre à jour ou charger de nouveaux packages sans accès à Internet. [28]

Connectez-vous directement à un ordinateur (IP statique) :

1. Attribuons à votre ordinateur une adresse IP statique.
2. Connectons la carte au port Ethernet de votre ordinateur.
3. Naviguons vers http://192.168.2.99:9090 [28]

3.5 Connexions à Jupyter Notebook

Une fois notre carte est configurée, pour nous connecter à Jupyter Notebooks, ouvrons un navigateur Web et accédez à http://192.168.2.99:9090 car notre carte est connectée à l'ordinateur via une adresse IP statique.

Un écran de connexion nous sera présenté. Le nom d'utilisateur est xilinx et le le mot de passe est également xilinx [28]

4. Création de notre projet

4.1 Notre architecture

Notre programme a l'objectif de détecter les fréquences d'harmonique a une fréquence f0 déterminer à travers un réseau de neurone et les enregistrer comme des poids « w » les entrées de ce programme se compose des 3 signaux sinus et 3 signaux cosinus des fréquences de 1f0 ,2f0 et 3f0, le type de notre réseau de neurone est ADALINE avec un apprentissage supervisé.

Le Système Générateur est un outil puissant qui intègre le processus de conception FPGA de Xilinx au Simulink de MATLAB, qui utilise une description de haut niveau pour réaliser facilement un système complexe. Nous concevons d'abord le système et vérifions sa fonctionnalité dans l'environnement Simulink. La description graphique de haut niveau de Simulink facilite considérablement la modélisation, la simulation et l'analyse de la conception. Ensuite, nous pouvons générer la description VHDL de la conception et l'ajouter à notre projet dans le logiciel Vivado suite design. [20]

4.2. Architecture du réseau de neurones Avec system generator

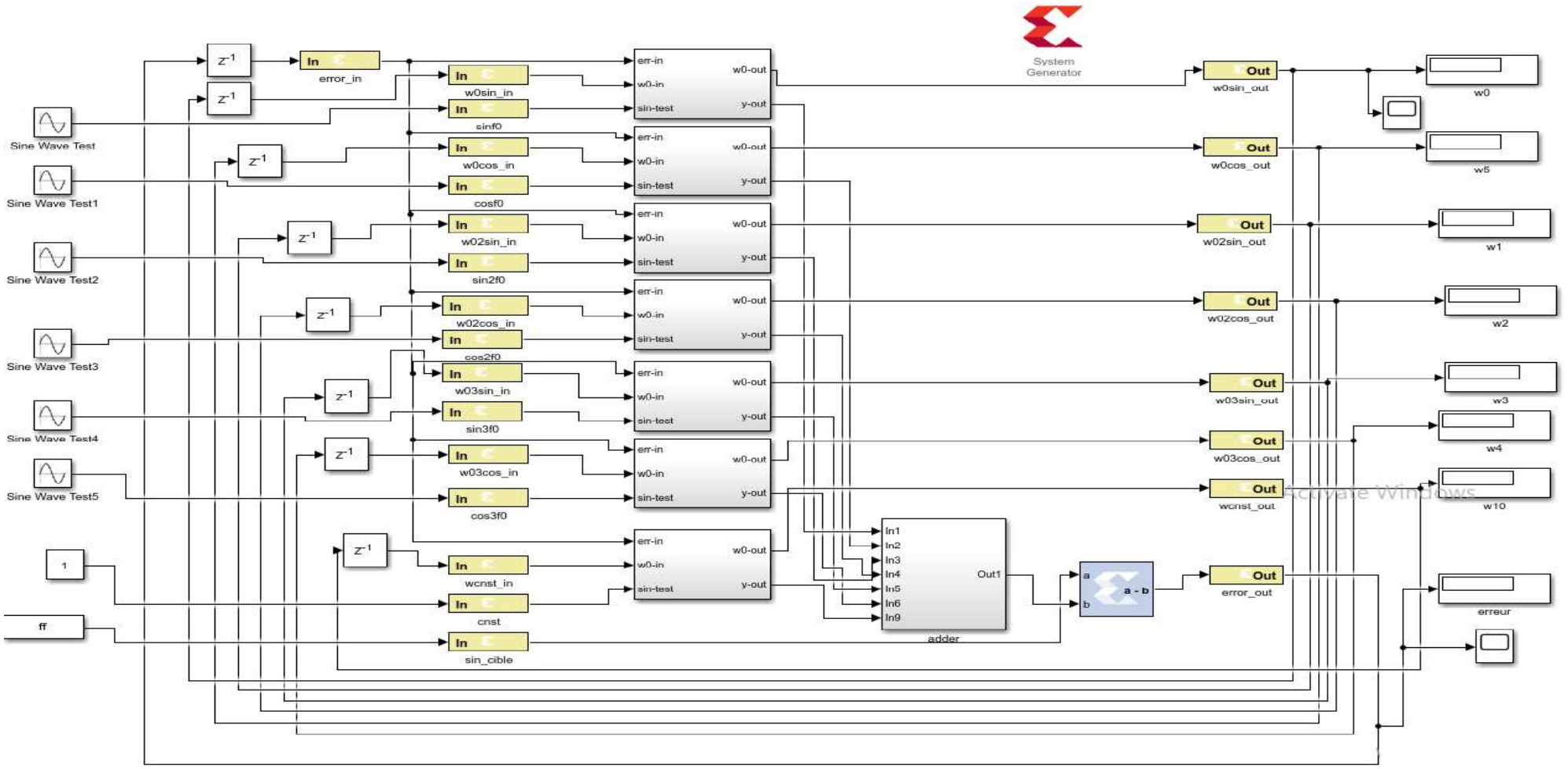


Figure III.2 Architecture du notre réseau Avec system generator

4.3. Création de IP core

Tout modèle Simulink utilisant des blocs Xilinx doit inclure un bloc « system generator». Ce bloc nous permet de contrôler les paramètres du système et de la simulation. Il gère également la génération de code HDL. Le symbole et la boîte de dialogue du bloc «System Generator» sont illustrés à la figure III.3

System Generator compile la conception en représentations de bas niveau. Nous pouvons choisir le type de représentation de bas niveau à partir du paramètre « Compilation » de la boîte de dialogue. Nous choisirons « IP catalog» comme indiqué dans la figure3.Comme vous pouvez le voir sur la figure4, nous avons choisi « VHDL » comme outil de synthèse.

Nous avons également choisi le périphérique cible « PynqZ2 » dans le paramètre « Part » de la boîte de dialogue, et donner au logiciel le dossier de destination pour stocker les fichiers générés.

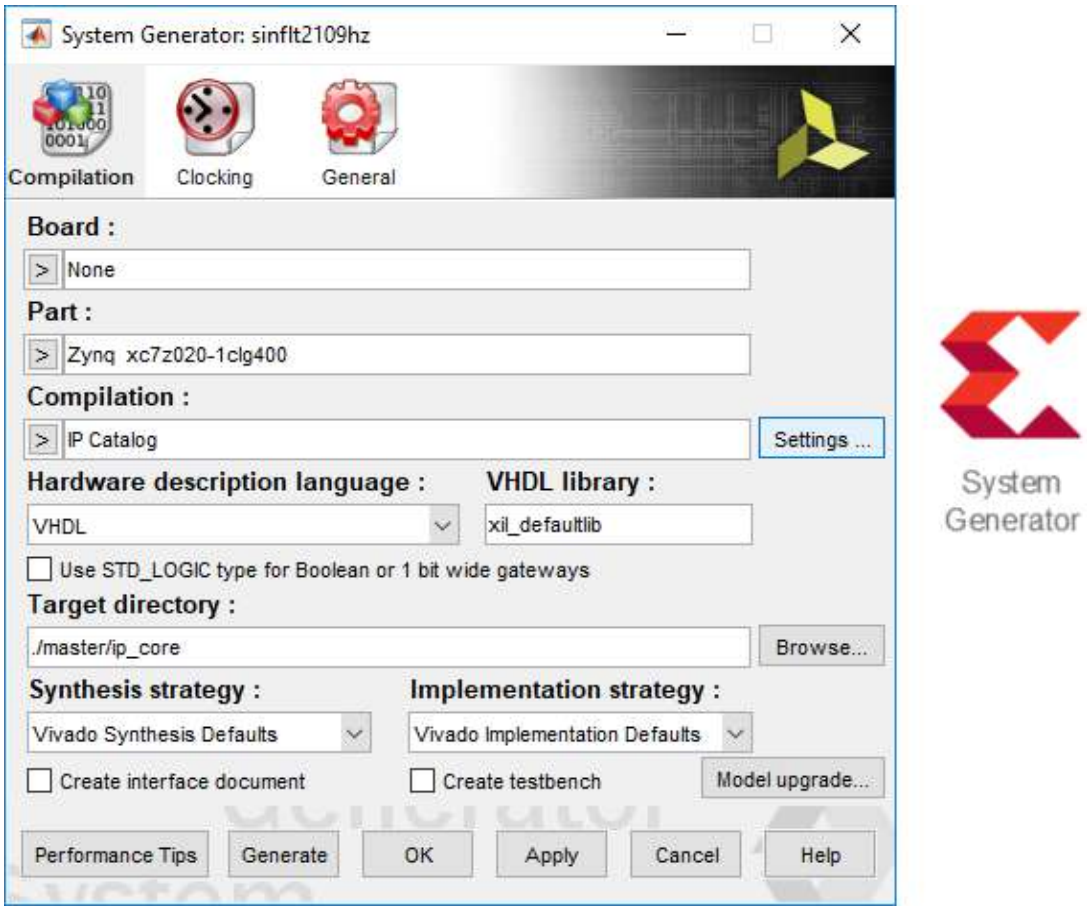


Figure III.3 onglet compilation dans la fenêtre du system generator

La figure III.4 ci-dessous montre l'onglet «Clocking» de la boîte de dialogue. Le premier paramètre «FPGA clock period (ns))» définit la période de l'horloge souhaitée pour la conception. Ce paramètre peut être transmis à l'outil de synthèse lors des prochaines étapes de la conception. Il peut guider le logiciel de synthèse dans le choix d'une implémentation appropriée en fonction des

exigences d'horloge de la conception. Dans la figure ci-dessus, la «FPGAclock period (ns)» est fixée à 10 nanosecondes. Cela signifie que nous nous attendons à ce que la conception soit exécutée avec une période d'horloge de 10ns sur le tableau. Simulink peut utiliser une forme normalisée de cette période d'horloge dans ces simulations. Cette normalisation est spécifiée par le paramètre « Simulink system period (s) » de la figure III.4. En définissant ce paramètre sur « 1/400 », toutes les 10 ns de l'implémentation e matérielle sont représentées par « 1/400 » seconde dans l'environnement Simulink.

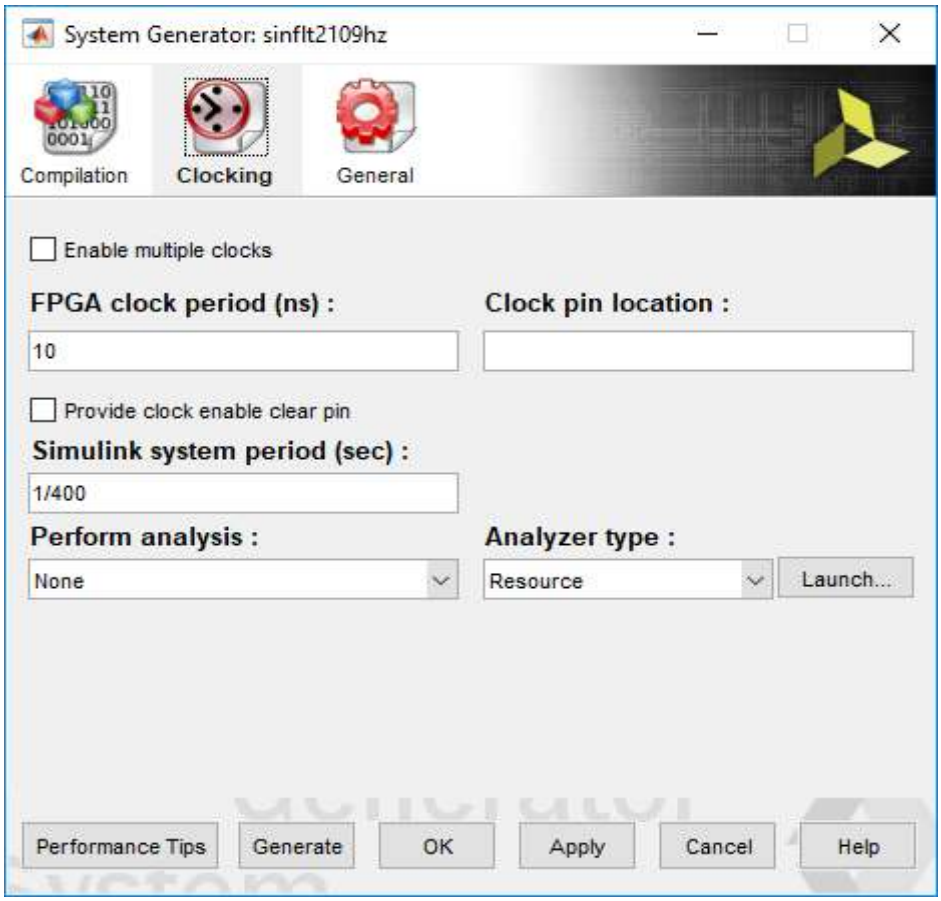


Figure III.4 Onglet clocking dans la fenêtre du system generator

Après avoir défini tous les paramètres de la conception, nous pouvons générer la description VHDL du modèle en appuyant sur le bouton "Generate" de la figureIII.4, Ceci produira un Ip core qui sera ajouter par la suite dans la librairie des ipcore personnalisé du Vivado qui peut être ajouté à notre projet dans Vivado.

4.4 Conceptions de notre architecture sur vivado suite designe

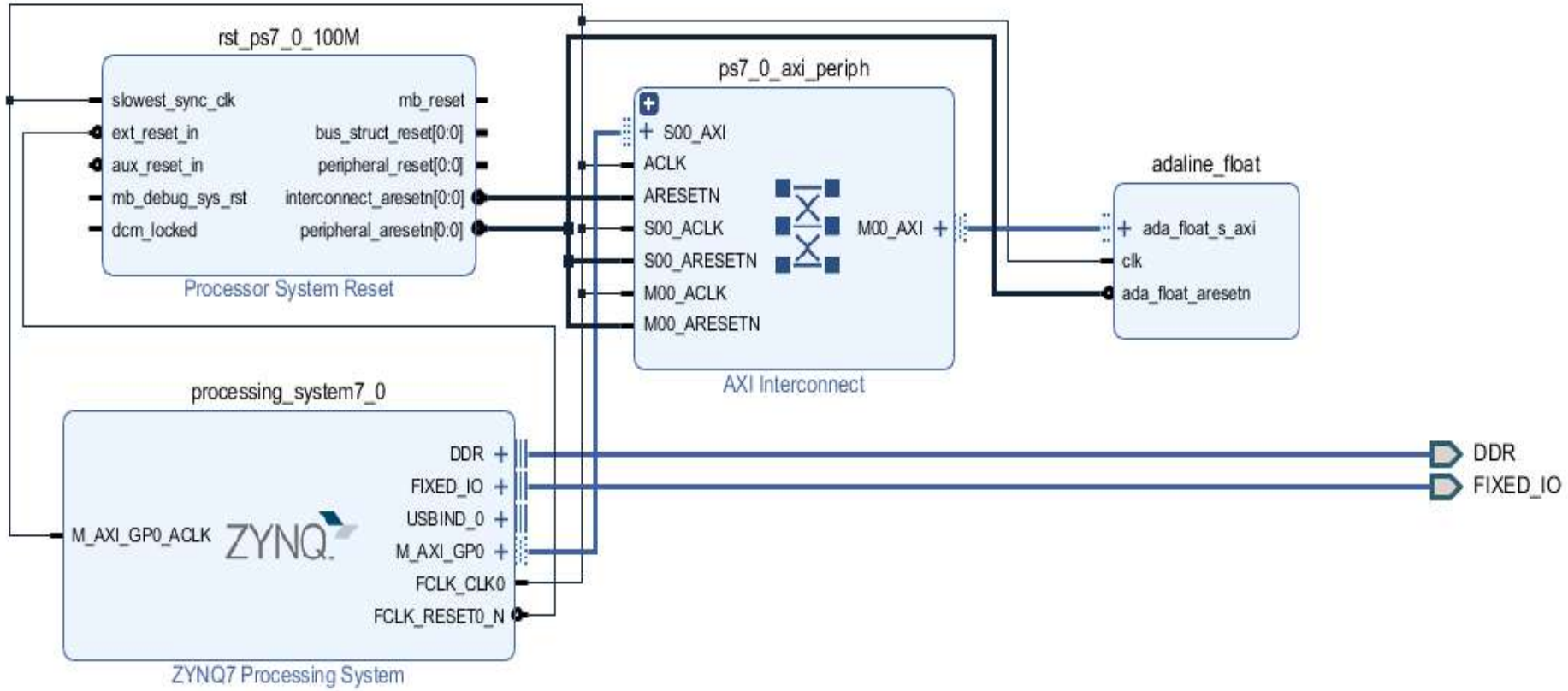


Figure III.5 conceptions de notre architecture sur vivado suite designe

4.4.1 Processor System Reset

Le Processor System Reset Xilinx nous permet d'adapter la conception à nos applications en définissant certains paramètres pour activer / désactiver les fonctions. Les caractéristiques paramétrables de la conception sont décrites dans Paramètres de conception du module de réinitialisation du système de processeur. [29]

4.4.2 The AXI Interconnect IP

Le AXI Interconnect IP connecte un ou plusieurs périphériques maîtres AXI mappés en mémoire à un ou plusieurs périphériques esclaves mappés en mémoire. Les interfaces AXI sont conformes aux spécifications AMBA® AXI version 4 d'ARM, y compris le sous-ensemble d'interfaces de registre de contrôle AXI4-Lite. L'interconnexion IP est uniquement destinée aux transferts mappés en mémoire ; Les transferts AXI4-Stream ne sont pas applicables. AXI Interconnect IP peut être utilisé à partir du catalogue IP de Vivado® sous la forme d'un noyau de EDK (Embedded Development Toolkit) ou d'un noyau autonome à partir du catalogue IP de CORE Generator ™ [30]

4.4.3 Zynq-7000 Processing System

Xilinx fournit le wrapper IP du système de traitement du Zynq®-7000 pour accélérer votre conception et sa configuration pour vos produits intégrés.

Le système de traitement IP est l'interface logicielle autour du système de traitement Zynq-7000. La famille Zynq-7000 comprend un système de traitement intégré (PS) de type système sur puce (SoC) et une unité de logique programmable (PL), fournissant une solution de SoC extensible et flexible sur une seule puce. Le wrapper IP du système de traitement agit en tant que connexion logique entre le PS et le PL tout en vous aidant à intégrer des IP personnalisées et intégrées au système de traitement à l'aide de l'intégrateur IP Vivado²®.[31]

4.4.4 Adaline_flot

Le bloque Adaline_float représente notre IPcore créer par le system generator que nous avons démontré la procédure de sa création dans la section précédente.

4.5 Conception avec vivado suite design

4.5.1 Création de bitstream et le fichier tcl

Le fichier Tcl doit être généré et fourni avec le fichier bitstream dans le cadre d'une superposition. Le fichier Tcl peut être généré dans Vivado en exportant le diagramme IP Integrator

à la fin du processus de conception de superposition. Le fichier Tcl doit être fourni avec un flux binaire lors du téléchargement d'une superposition. La classe PYNQ PL analysera automatiquement le Tcl.

4.5.1.a Exportation d'un fichier tcl

Le Tcl de Vivado IP Integrator pour la conception PL est utilisé par PYNQ pour identifier automatiquement la configuration du système Zynq, ainsi que les versions IP, les interruptions, les réinitialisations et autres signaux de contrôle. Sur la base de ces informations, certaines parties de la configuration système peuvent être automatiquement modifiées à partir de PYNQ, des pilotes peuvent être attribués automatiquement, des fonctionnalités activées ou désactivées et des signaux peuvent être connectés aux méthodes Python correspondantes.

Un fichier Tcl personnalisé ou créé manuellement peut être utilisé pour créer un projet Vivado, mais Vivado doit être utilisé pour générer et exporter le fichier Tcl du diagramme. Ce Tcl généré automatiquement doit permettre de l'analyser correctement par le PYNQ.

Pour générer notre fichier Tcl pour la conception de notre diagramme, nous devons cliquer sur File > Export > Block Design.

4.5.1.b Bitstream " flux binaire "

Un bitstream d'FPGA est un fichier contenant les informations de programmation d'un FPGA. Un périphérique FPGA Xilinx doit être programmé à l'aide d'un train de bits spécifique pour qu'il se comporte comme une plate-forme matérielle intégrée. Ce flux binaire est généralement fourni par le concepteur de matériel qui crée la plate-forme intégrée.[32]

4.5.1.c Copier les fichier bitstream et tcl dans la carte mémoire SD

Les fichiers tcl et bitstream doivent avoir le même nom, nous avons les nommé adalineIp dans notre projet et les copier dans la carte SD de la carte Pynqz2 à la distination pynq > xilinx > pynq > overlays > adalineIp.

4.6 Conception de l'algorithme de la détection des harmoniques

L'organigramme fonctionnel de la détection des harmoniques est illustré par la figure III.16. L'algorithme prend comme entrée 3 signaux sinus et 3 signaux cosinus avec une cible. Les signaux d'entrée sont générés et placées dans une matrice de 6*1024 éléments, ces signaux prennent une fréquence f0 déterminée. Les poids sont aléatoirement initialisés dans un intervalle de [0,1] et ils

sont placées dans un vecteur de 6 éléments. Le coefficient d'apprentissage v est définit par 0,02. A chaque itération, trois opérations sont appliquées dans la boucle d'apprentissage :

1) Calculer le signal de sortie : le produit entre les signaux d'entrée et les poids.
2) Calculer l'erreur : la différence entre la cible et la sortie.
3) Corriger les poids : Wnew = Wold + 2 * v * err * E.

Wnew : la nouvelle valeur de poids.

Wold : l'ancien valeur de poids.

Err : L'erreur.

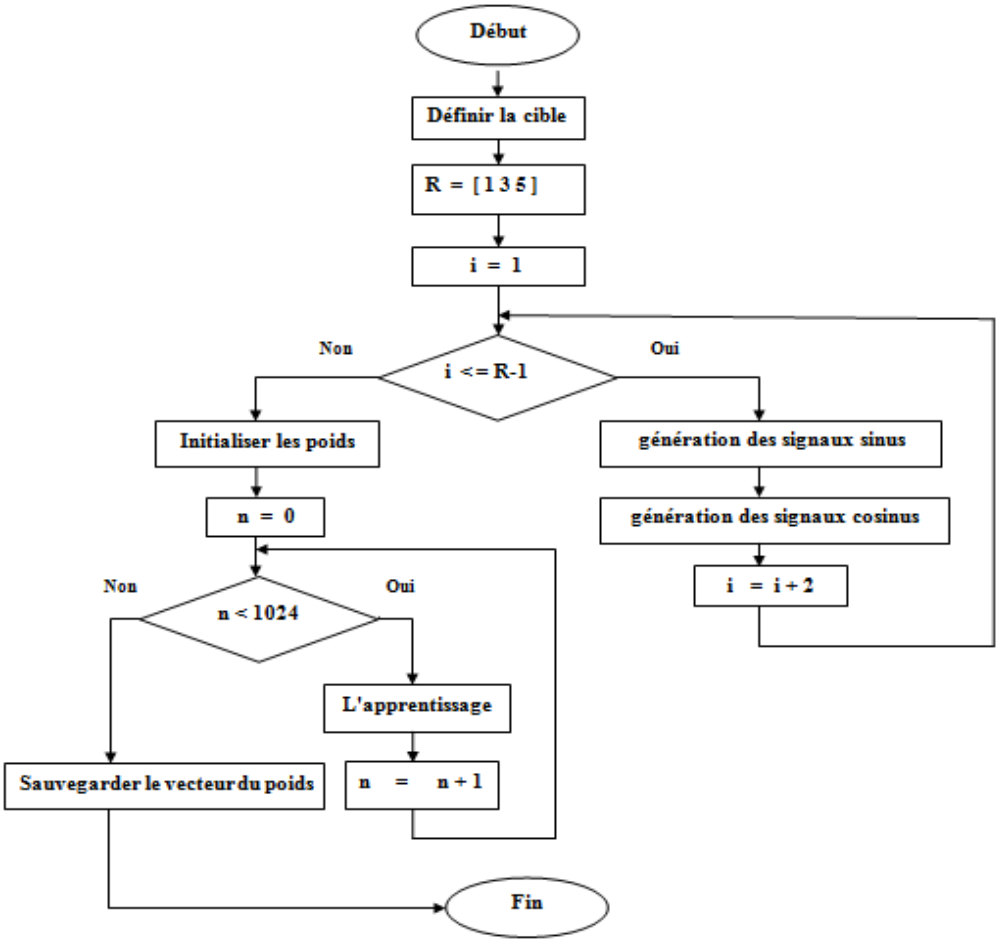


Figure III.6 L'organigramme de l'algorithme software

V : Coefficient d'apprentissage.

E : Signal d'entrée.

Finalement on sauvegarde le vecteur du poids dans un fichier texte.

4.6.1 La simulation de l'algorithme

L'algorithme proposée pour la détection des harmoniques est décrit en langage python. Pour l'algorithme un test a été effectué pour vérifier son fonctionnement, calculer le temps d'exécution de l'algorithme et de la boucle d'apprentissage.

L'algorithme de la détection des harmoniques est exécuté à l'aide de Jupiter notebook.

La configuration du matériel utilisé dans notre simulation est :

- Un PC de CPU i5-2450M 2.50 GHZ
- RAM de taille 4 Gb

Résultats obtenus

Les résultats pour une fréquence f0 égale à 50 Hz sont les suivants :

4.6.1.*a La cible*

La cible traitée dans l'algorithme de la détection des harmoniques est définie par l'équation suivante :

$$Cible = 5{,}25.\sin(2.\pi.f0.t) + 0{,}25.\cos(6.\pi.f0.t) + 0{,}75.\sin(6.\pi.f0.t) \quad (3.1)$$

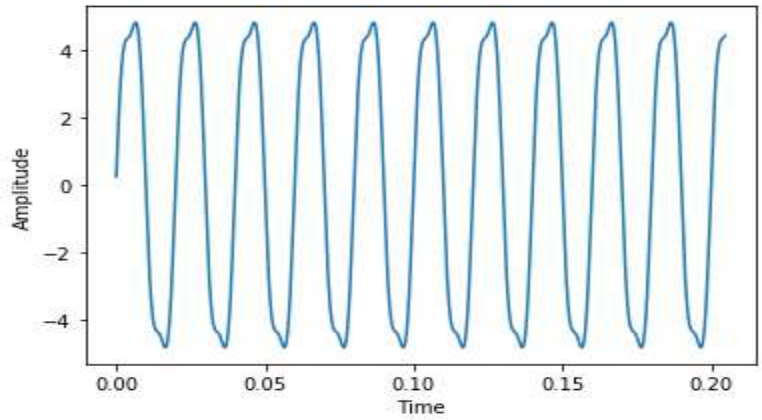


Figure III.7 Le signal cible à traité

4.6.1.b Signaux d'entrée

Les signaux d'entrée aide à détecter les harmoniques de la fréquence f0 pour notre exemple nous souhaitons détecter les trois premiers harmoniques sinus et cosinus alors nous avons utilisé des signaux de fréquences f0,2f0 et 3f0 comme des signaux d'entrées dans notre architecture.

Les signaux d'entrée à fréquences f0,2f0 et 3f0 sont illustrés par les figures suivantes :

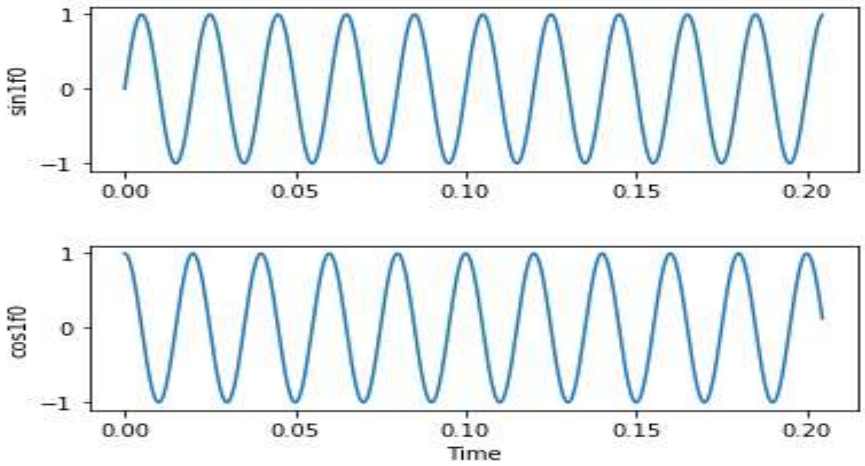


Figure III.8 un signal d'entrée avec une fréquence de f0

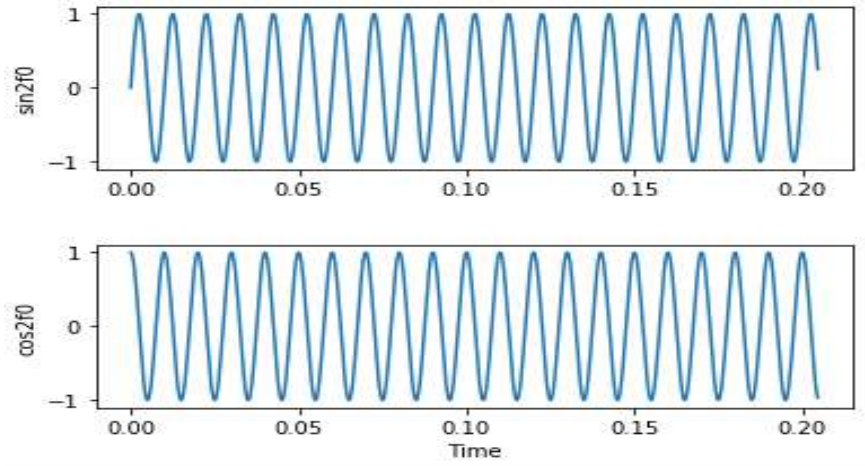


Figure III.9 un signal d'entrée avec 2f0

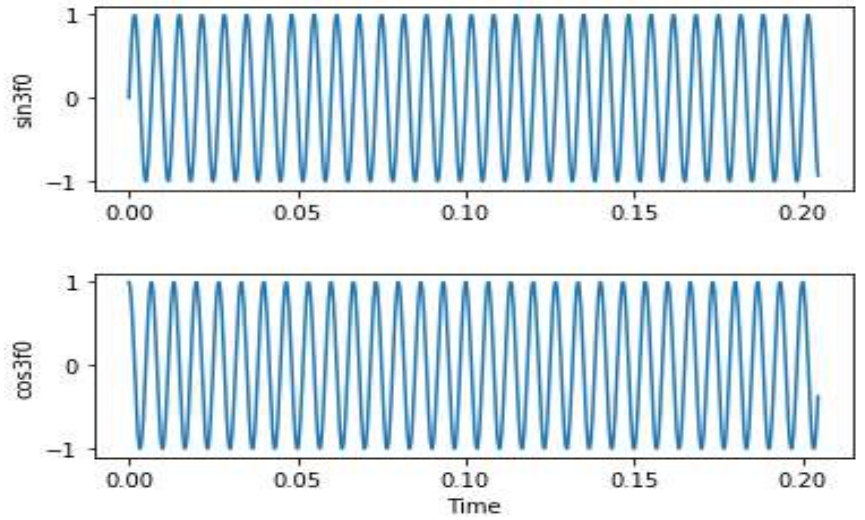


Figure III.10 un signal d'entrée avec 3f0

4.6.1.c La sortie

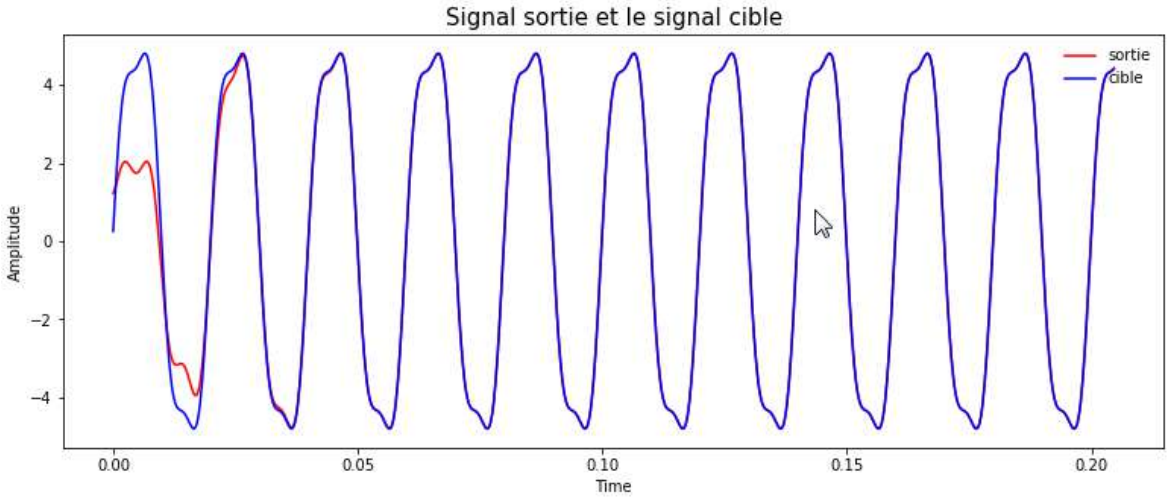


Figure III.11 Résultat d'apprentissage

La figure III.11 illustre que le taux de l'apprentissage augmente avec le temps « time » jusqu'à atteindre une adaptation complète entre la cible et la sortie, ceci reflète qu'à chaque milliseconde le réseau de neurone Adaline apprend plus d'informations.

4.6.1.d L'erreur

La figure III.12 illustre que l'erreur atteint le maximum au début de l'apprentissage et tend vers le zéro avec le temps « time ».

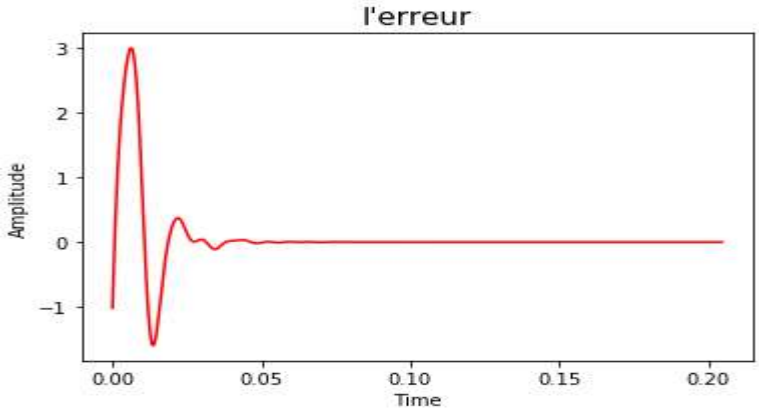


Figure III.12 l'erreur d'apprentissage

4.6.1.e Les poids

La figure III.13 et la figure III.14 illustrent les poids après l'apprentissage, ces poids représentent les harmoniques détectés en fonction sinus et cosinus.

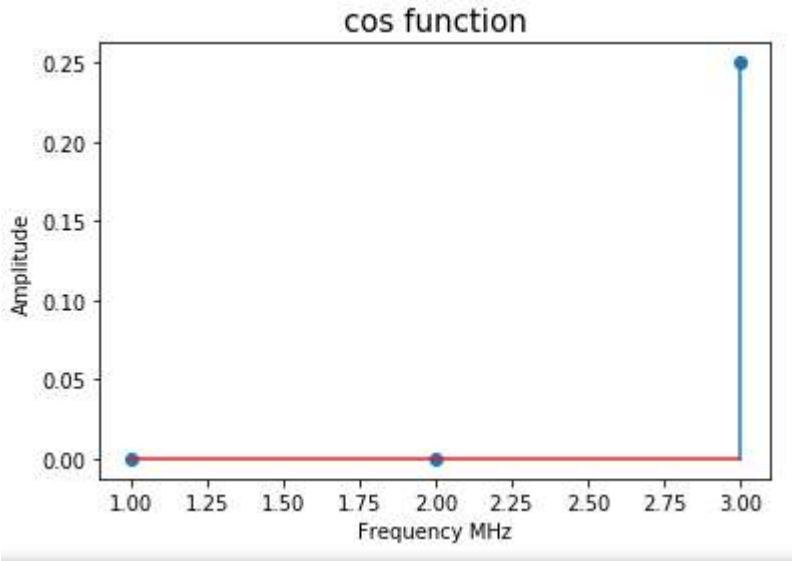


Figure III.13 l'harmonique détecté en fonction cosinus

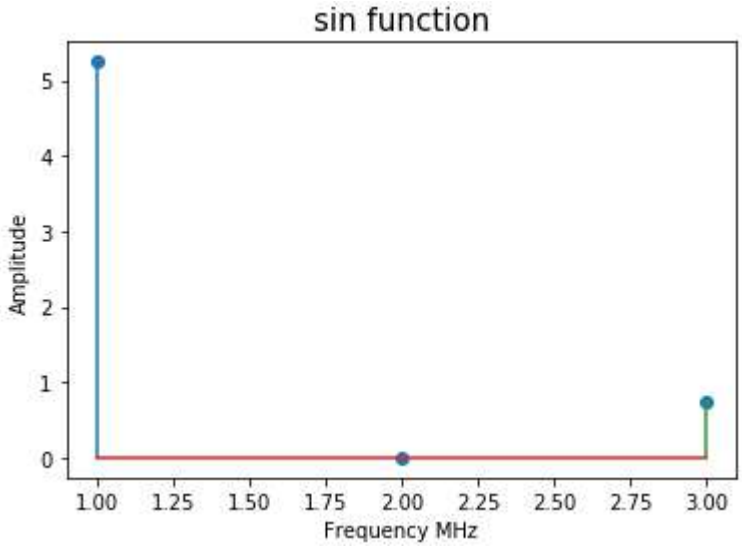


Figure III.14 les harmoniques détectés en fonction sinus

4.7 Conception hardware

Les étapes pour implémenter l'algorithme de la détection des harmoniques avec la carte Pynq Z2 sont illustrés sur la figure III.15. L'algorithme hardware prend les mêmes signaux d'entrée que l'algorithme software. Les signaux d'entrée sont déclarés et placés dans un fichier python (data_in.py) à l'extérieur, et ils sont exécutés à l'intérieur de l'algorithme.

Les poids sont initialisés par zéros. A chaque itération, quatre opérations sont appliquées dans la boucle d'apprentissage :

1 - Lire et affecter les adresses des poids d'entrée aux poids de sortie du point précédent.

2 - Lire et affecter l'adresse d'erreur d'entrée à l'erreur de sortie du point précédent.

3 - Lire et affecter les adresses des signaux d'entrées et de signal cible à leur valeur actuelle.

4 - L'écriture des poids et l'erreur de sortie.

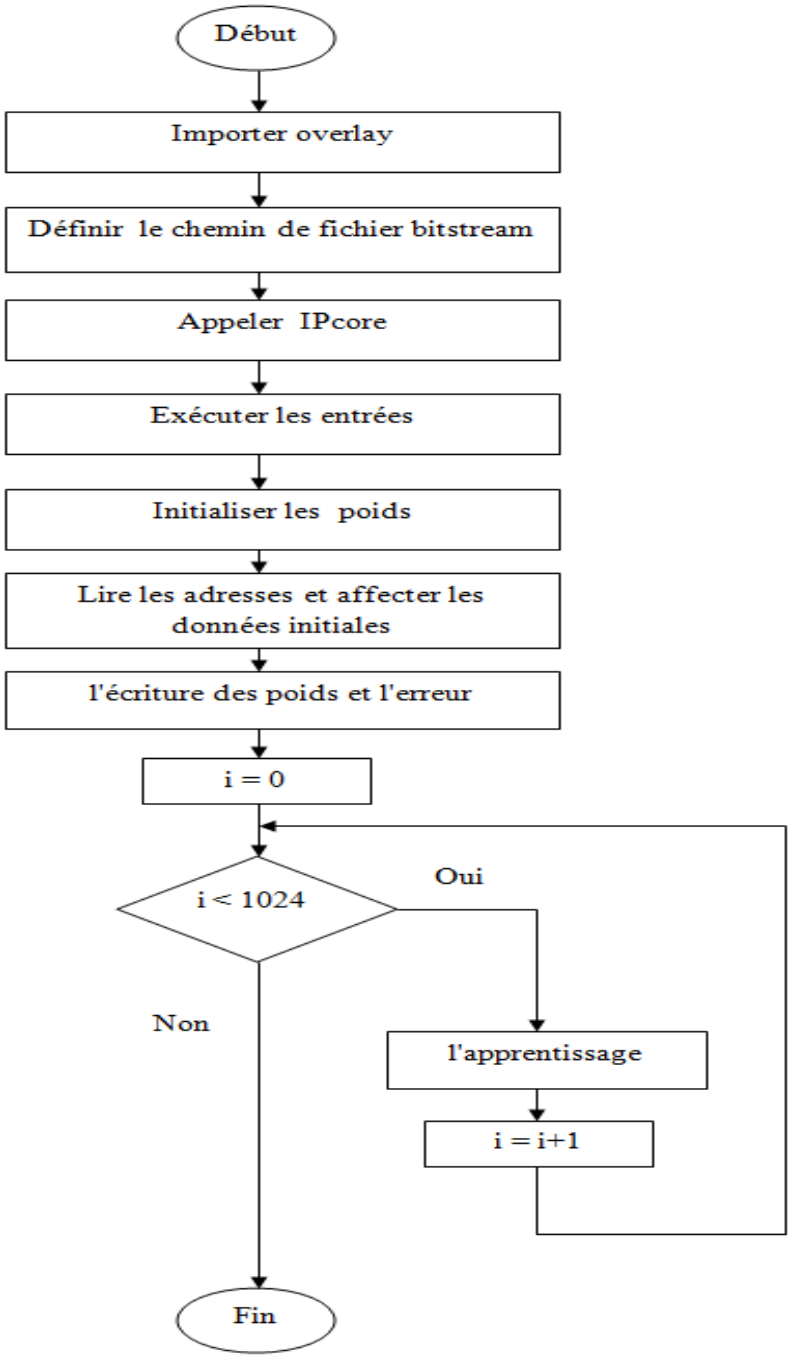


Figure III.15 L'organigramme de l'algorithme hardware

4.7.1 L'implémentation de l'algorithme

L'algorithme hardware proposée pour la détection des harmoniques, est décrite en langage python. Pour l'algorithme un test a été effectué pour vérifier son fonctionnement et calculer son temps d'exécution.

L'algorithme hardware de la détection des harmoniques est exécuté dans une carte Pynq Z2 à l'aide de Jupyter notebook.

La configuration du matériel utilisé dans notre implémentation est :

- Un processeur 650MHz dual-core Cortex-A9
- RAM de taille 512MB.

4.7.2 Résultats obtenus

Les résultats pour une cible et signaux d'entrée similaires aux algorithme software sont les suivants:

4.7.2.a Les poids

Les résultats des poids après l'implémentation du l'algorithme hardware sont les suivants :

w0sin_out = 5.249999523162842

w0cos_out = $6.678574777652102.10^{-07}$

w02sin_out = $4.3114027903357055.10^{-06}$ } tend vers 0.

w02cos_out = $-9.030279102262284.10^{-08}$

w03sin_out = 0.7499939799308777

w03cos_out = 0.2500055134296417

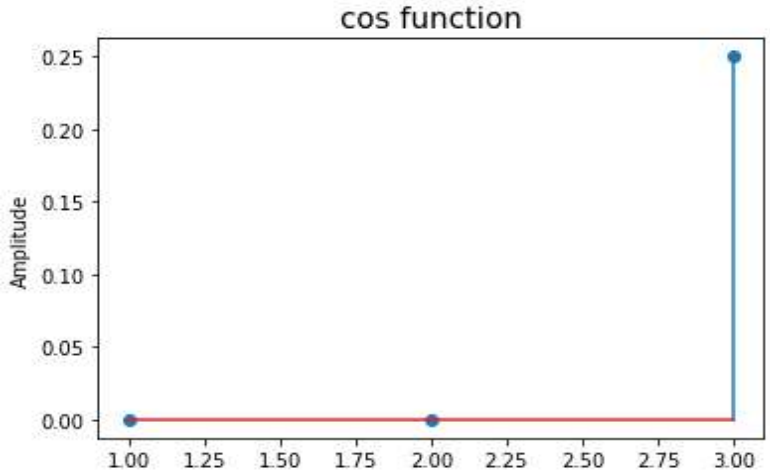


Figure III.16 l'harmonique détecté en fonction cosinus

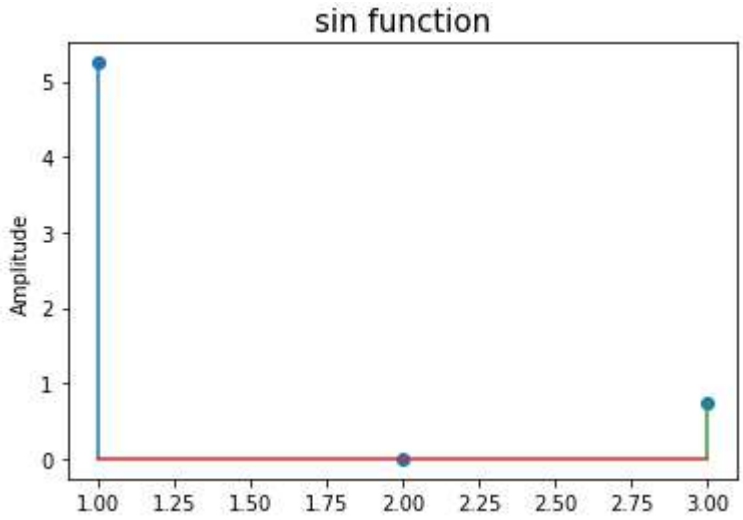


Figure III.17 les harmoniques détectés en fonction sinus

4.8 Comparaison et interprétation des résultats

Tableau III.2 comparaisons entre les exécutions hardware et software

	Temps d'exécution (s)	L'erreur
Algorithme software	0,009	$-6{,}29.10^{-07}$
Algorithme hardware	1.087	$-5{,}72.10^{-06}$

Constat ; la différence entre le temps d'apprentissage d'exécution hardware et software « presque 120 fois » est à cause du paquet « struct » qui prend un temp considérable pour se compresser et se décompresser de ces données.

Nous n'avons pas noté une grande déférence entre les deux exécutions concernant l'erreur d'apprentissage.

5. Conclusion

Dans ce chapitre nous avons présenté une conception matérielle d'un réseau de neurones qui nous a permis de détecter les harmoniques. Au cours de ce chapitre, nous avons développé deux algorithmes différents dont la première présente un code software et la deuxième un code hardware. Finalement une exécution a été effectué à ces algorithmes à l'aide de Jupyter notebook et la carte pynq-z2.

Conclusion générale

Les réseaux de neurones ont prouvé leur efficacité grâce à leur puissance d'apprentissage et ceci en donnant les meilleurs résultats d'évaluation.

L'implémentation des circuits sur les composants reconfigurables FPGA est un sujet d'actualité, mais il faut prend en considération les ressources à utilisées.

Dans ce travail, nous avons réussi à créer un algorithme d'une neurone linéaire adaptatif (Adaline) pour détecter les harmoniques d'une fréquence fondamentale et les enregistrer sous forme de poids. Cet algorithme est après testé avec le langage de programmation python.

L'architecture parallèle d'Adaline convient à l'implémentation sur des circuits FPGA. Nous avons montré comment procéder cet implémentation étape par étape à l'aide de la carte PynqZ2 de Xilinx.

Les adresses E/S d'IPcore :

```
#define ADA_FLOAT_W0SIN_IN 0x0/**< w0sin_in */
#define ADA_FLOAT_W0COS_IN 0x4/**< w0cos_in */
#define ADA_FLOAT_W03SIN_IN 0x8/**< w03sin_in */
#define ADA_FLOAT_W03COS_IN 0xc/**< w03cos_in */
#define ADA_FLOAT_W02SIN_IN 0x10/**< w02sin_in */
#define ADA_FLOAT_W02COS_IN 0x14/**< w02cos_in */
#define ADA_FLOAT_SINF0 0x18/**< sinf0 */
#define ADA_FLOAT_SIN_CIBLE 0x1c/**< sin_cible */
#define ADA_FLOAT_SIN3F0 0x20/**< sin3f0 */
#define ADA_FLOAT_SIN2F0 0x24/**< sin2f0 */
#define ADA_FLOAT_ERROR_IN 0x28/**< error_in */
#define ADA_FLOAT_COSF0 0x2c/**< cosf0 */
#define ADA_FLOAT_COS3F0 0x30/**< cos3f0 */
#define ADA_FLOAT_COS2F0 0x34/**< cos2f0 */
#define ADA_FLOAT_ERROR_OUT 0x38/**< error_out */
#define ADA_FLOAT_W02COS_OUT 0x3c/**< w02cos_out */
#define ADA_FLOAT_W02SIN_OUT 0x40/**< w02sin_out */
#define ADA_FLOAT_W03COS_OUT 0x44/**< w03cos_out */
#define ADA_FLOAT_W03SIN_OUT 0x48/**< w03sin_out */
#define ADA_FLOAT_W0COS_OUT 0x4c/**< w0cos_out */
#define ADA_FLOAT_W0SIN_OUT 0x50/**< w0sin_out */
```

Bibliographie

[1] TOUZET**,** Claude. Les réseaux de neurones artificiels, Cours, exercices et travaux pratiques :" Introduction au connexionnisme ",129p :1992

[2] MUXIKA OLASAGASTI, Efiaut." Application des Réseaux de Neurones a l'Identification d'un Axe de Machine-Outil ", Thèse de doctorat, INPG : 2002

[3] Bernard GOSSELIN," Application de réseaux de neurones artificiels a la reconnaissance au de caractères manuscrits " 231p. Thèse de doctorat. Faculté Polytechnique de Mons : 95-96

[4] DREYFUS, Gérard." Neural networks, méthodology and applications " .509p.paris :2002

[5]Principe de fonctionnement d'un neurone formel.https://www.editionseni.fr/open/mediabook.aspx?idR=2dac0ec804e7e66899c47f486d228ff0

[6] S.ZEGHLACHE, " Commande Intelligente ", cours, université Mohamed boudiaf-m'sila.2008

[7] PETITJEAN**,** Gérald. Cours, réseaux de neurones :" Introduction aux réseaux de neurones " 102p

[8] Mme BABA HAMED Amel née BENSEMAIN : "Vers Un Modèle De Classification Neuronale Des Données Médicales A Base De La Technologie FPGA".173p. thèse de doctorat. Université Aboubakr Belkaïd Tlemcen : 2017

[9] BUYSSENS, Pierre. Traitement des images : " Fusion de différents modes de capture pour la reconnaissance du visage appliquée aux e_transactions " 242p. Thèse de doctorat. Université de Caen Basse-Normandie : 2011

[10] Cyrille FAUCHEUX, Alexandre LISSY. Rapport de projet " Réseaux de neurones sous Matlab ", École Polytechnique de l'Université de Tours,

[11] Marc PARIZEAU " Réseaux de neurones ".université Laval 124p :2004

[12]:DESIGN FOR EMBEDDED IMAGE PROCESSING ON FPGAS Donald G. Bailey Massey University, New Zealand

[13]: design of embedded processors,version2,module 4 EE IIT,Kharagpur.

[14]: Xilinx7 Series FPGAs Data Sheet: Overview

[15]: Zynq-7000 SoC Technical Reference Manual

[16]:https://jeanfistonmikwa.skyrock.com/3030412618-Python-est-un-langage-de-programmation.html

[17]:Morris, Kevin (2014-02-25). "Xilinx vs. Altera, Calling the Action in the Greatest Semiconductor Rivalry". Electronic Engineering Journal.

[18] :Vivado Design Suite, Xilinx Website

[19] :Vivado Design Suite, First version released in 2012, Xilinx Downloads

[20]:https://www.allaboutcircuits.com/technical-articles/use-the-xilinx-system-generator-to-implement-a-simple-dds/

[21] :http://www.pynq.io/

[22]:https://pynq.readthedocs.io/en/latest/pynq_overlays/pynqz2/pynqz2_base_overlay.html#pynq-z2-block-diagram

[23] :https://www.numpy.org/

[24] :https://matplotlib.org/

[25]: https://docs.python.org/2/library/struct.html

[26]: https://docs.python.org/2/library/struct.html

[27]:https://www.dfrobot.com/product1838.html?fbclid=IwAR2D28qsx2K3DxU_u8CKI1imKhOSQcgHHqi-HyYXtImcCN4muOgrNy_c0h8

[28] : Python productivity for Zynq Pynq Documentation Release 2.2, May 29, 2018

[29] :https://www.xilinx.com/products/intellectual-property/proc_sys_reset.html

[30] :https://www.xilinx.com/products/intellectual-property/axi_interconnect.html

[31] :https://www.xilinx.com/products/intellectual-property/processing_system7.html

[32]:https://pynq.readthedocs.io/en/latest/pynq_overlays/pynqz2/pynqz2_base_overlay.html#pynq-z2-block-diagram.