

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE  
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA  
RECHERCHE SCIENTIFIQUE



المدرسة الوطنية المتعددة التقنيات  
Ecole Nationale Polytechnique

ÉCOLE NATIONALE POLYTECHNIQUE  
DÉPARTEMENT D'ÉLECTRONIQUE  
LABORATOIRE DES DISPOSITIFS DE  
COMMUNICATION  
ET DE CONVERSION PHOTOVOLTAÏQUE



MÉMOIRE DE PROJET DE FIN D'ÉTUDES  
POUR L'OBTENTION DU DIPLOME D'INGÉNIEUR D'ÉTAT ÉLECTRONIQUE

---

## Générateur intelligent de multi-réseaux neuronaux artificiels

Application : La commande SHE PWM pour le contrôle de vitesse des  
moteurs Asynchrones

---

*Auteur :*  
BOURENANE Aomar

*Superviseurs :*  
Pr. LARBES Cherif  
Dr. GUELLAL Amar

Présenté et soutenu publiquement le 21 juin 2018

### Composition du Jury :

HADDADI Mourad	Professeur	ENP	Président
AIT- CHEIKH Mohamed Salah	Professeur	ENP	Examineur
LARBES Cherif	Professeur	ENP	Rapporteur
GUELLAL Amar	MCB	ESSA	Rapporteur

ENP 2018

Ecole Nationale Polytechnique (ENP)

10, Avenue des Frères Oudek, Hassen Badi, BP 182 16200 El Harrach, Alger, ALGERIE

[www.enp.edu.dz](http://www.enp.edu.dz)



RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE  
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA  
RECHERCHE SCIENTIFIQUE



المدرسة الوطنية المتعددة التقنيات  
Ecole Nationale Polytechnique

ÉCOLE NATIONALE POLYTECHNIQUE  
DÉPARTEMENT D'ÉLECTRONIQUE  
LABORATOIRE DES DISPOSITIFS DE  
COMMUNICATION  
ET DE CONVERSION PHOTOVOLTAÏQUE



MÉMOIRE DE PROJET DE FIN D'ÉTUDES  
POUR L'OBTENTION DU DIPLOME D'INGÉNIEUR D'ÉTAT ÉLECTRONIQUE

---

## Générateur intelligent de multi-réseaux neuronaux artificiels

Application : La commande SHE PWM pour le contrôle de vitesse des  
moteurs Asynchrones

---

*Auteur :*  
BOURENANE Aomar

*Superviseurs :*  
Pr. LARBES Cherif  
Dr. GUELLAL Amar

Présenté et soutenu publiquement le 21 juin 2018

**Composition du Jury :**

HADDADI Mourad	Professeur	ENP	Président
AIT- CHEIKH Mohamed Salah	Professeur	ENP	Examineur
LARBES Cherif	Professeur	ENP	Rapporteur
GUELLAL Amar	MCB	ESSA	Rapporteur

ENP 2018

Ecole Nationale Polytechnique (ENP)

10, Avenue des Frères Oudek, Hassen Badi, BP 182 16200 El Harrach, Alger, ALGERIE

[www.enp.edu.dz](http://www.enp.edu.dz)

Au nom de Dieu, le Tout Miséricordieux, le Très Miséricordieux

« *Ô mon Seigneur, accroît mes connaissances* »

Sourate Tâ-Hâ, Verset 114

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

« *وَقُلْ رَبِّ زِدْنِي عِلْمًا* »

سورة طه ، الآية 114

In the name of God, Most Gracious, Most Merciful.

« *O my Lord, advance me in knowledge* »

Sourate Tâ-Hâ, Verse 114

**ملخص:** يتم وضع هذه الأطروحة في مجال التطبيقات الموجهة للشبكات العصبية الاصطناعية التي تتطلب التشغيل في الوقت الحقيقي والتحكم الدقيق واستهلاك موارد المعدات. ومع ذلك ، ثبت أن تنفيذ هذه الشبكات صعب للغاية خاصة للمجالات العلمية التي ليس لها خلفية في الإلكترونيات. ينعكس تعقيد انجاز الشبكات العصبية في مرحلتين أساسيتين ، مرحلة التنفيذ ، التي تتطلب فيها RNAs التوازي والمرونة ، ومرحلة تطوير طوبولوجية الشبكة العصبية ، والتي تعد واحدة من أهم مشاكل الذكاء الاصطناعي وبالأخص RNAs ، في بعض المشكلات العلمية المعقدة ، تحقيق هذه المرحلة يتطلب سنوات من الخبرة والحس. في هذا السياق ، مهمتنا هي الإجابة على هذه المشاكل من خلال اقتراح تطبيق ++C ذكي ينتج شبكات RNA متعددة في نوعين "التسلسلية والمتوازية" مع طوبولوجيا مرنة. لقد بذلنا كل ما في وسعنا لجعل التطبيق سهل الاستخدام ، بحيث يمكن الاستفادة منه في جميع مجالات الأبحاث. في الفصل الأول ، عرضنا تقنية SHE PWM التي استخدمنا كتطبيق للتحقق من عمل التطبيق. وقد خصص الفصل الثاني لتقديم بنية RNA وانشائها كاملة على MATLAB. في الفصل الأخير ، كشفنا عن البنية الداخلية لتطبيق ++C وجميع أوصاف VHDL. كما تم عرض المحاكاة والنتائج التجريبية.

**كلمات مفاتيح:** الذكاء الاصطناعي ، الشبكات العصبية الاصطناعية (ANN) ، FPGA ، تطبيق ++C ، المحرك اللامتزامن

**Abstract:** This thesis is based on the general theme of artificial neural networks oriented to the applications requiring real-time operation and control of precision and consumption of hardware resources. However, the implementation of these networks proved very difficult. Especially for scientific fields that do not have a background in electronics. The complexity of the realization of neural networks is reflected in two essential phases, the implementation phase, in which the RNAs require parallelism and flexibility, and the development phase of the neural network topology, which is one of the major mysteries of artificial intelligence and especially RNAs, in some complex scientific problems the realization of this phase, may require years of experience and intuition. In this context, our work is to resolve these problems by proposing a smart C ++ based application for the implementation of a muliti-RNA network in two types of operation "serial and parallel" and with a flexible topology. We have done everything to make the application very easy to use, so that all areas of research can benefit. In the first chapter, we exposed the SHE PWM technique which will serve us as a validation application. The second chapter has been devoted to present the RNA architecture and a complete implementation on MATLAB. In the last chapter, we exposed the internal architecture of our C ++ application and all VHDL descriptions. Simulations and experimental results were also presented.

**Key words:** Artificial Intelligence, Artificial Neural Networks (ANN), FPGA, C ++ application, induction motor.

**Résumé :** Ce mémoire se positionne dans la thématique des réseaux de neurones artificiels orientés vers les applications nécessitant le fonctionnement en temps réel et un contrôle de précision et de la consommation en ressources hardwares. Cependant, l'implémentation de ces réseaux s'est avérée très difficile. Notamment pour les domaines scientifiques qui n'ont pas un background en électronique. La complexité de la réalisation des réseaux de neurones se traduit en deux phases essentielles, la phase d'implémentation dans laquelle les RNAs nécessitent un parallélisme et une flexibilité et la phase d'élaboration de la topologie du réseau neuronal qui constitue l'un des grands mystères de l'intelligence artificielle et particulièrement des RNAs, dans certains problèmes scientifiques complexes la réalisation de cette phase, pourrait nécessiter des années d'expérience et d'intuition. Dans ce contexte, notre travail consiste à répondre à ces problèmes en proposant une application basée sur C ++ intelligente pour l'implémentation d'un réseau muliti-RNA en deux types de fonctionnement "série et parallèle" et avec une topologie flexible. Nous avons tout fait pour rendre l'application très simple à l'utilisation, pour que tous les domaines de la recherche puissent en profiter. Dans le premier chapitre, on a exposé la technique SHE PWM qui nous servira comme une application de validation. On a consacré le deuxième chapitre pour présenter l'architecture de RNA et une implémentation complète sur MATLAB. Dans le dernier chapitre, on a exposé l'architecture interne de notre application C++ et toutes les descriptions VHDL. Des simulations et des résultats expérimentaux ont également été présentés.

**Mots clés :** Intelligence artificielle, Réseaux de neurones artificiels (RNA), FPGA, application C++, moteur asynchrone.

# *Dédicaces*

**Je dédie ce travail**

*A mon père ...*

*A ma mère ...*

*A ma sœur et mes frères ...*

*A toute la famille BOURENANE ...*

*A tous mes amis ...*

*A tous les membres de l'ENP.*

# *Remerciements*

La dernière phase académique de cet Ingéniaurat est l'aboutissement d'un processus de construction auquel plusieurs personnes ont apporté, chacun, sa pierre à l'édification.

...

Tout d'abord je voudrais remercier Dieu, le tout-puissant, pour m'avoir donné la force, la patience, la volonté et toutes ces bénédictions.

...

Je tiens à remercier de tout cœur mes parents qui m'ont toujours soutenus dans cette aventure et n'ont reculé devant aucune épreuve. Je leur serai toujours reconnaissant pour tout ce qu'ils ont fait pour moi. Je salue ici toutes leurs déterminations, leurs efforts et leurs sens du sacrifice.

...

J'adresse mes respectueux remerciements à mon encadreur, Pr. LARBES Chérif, Professeur à l'ENP, pour toute son assistance, sa rigueur, ses interpellations, son entière disponibilité tout au long de cette période. Outre ses grandes qualités scientifiques, j'ai également beaucoup apprécié ses qualités humaines qui ont fait que ce travail s'est déroulé dans une ambiance agréable.

...

J'exprime toute ma reconnaissance à Dr. Amar Guellal, de m'avoir guidé et soutenu. Il a donné l'exemple d'excellence en tant que chercheur, conseiller et instructeur. Il a fait preuve de disponibilité et de patience, et m'a fait partager sa grande culture, pas uniquement scientifique. Pour tout cela, je vous dis un grand merci

...

Je voudrai exprimer mon profond respect aux membres du jury, et de les remercier d'avoir accepté de m'accompagner lors de la dernière phase de ce travail.

...

Je remercie SAADI Khalid et OUADRIA Anes Abderrahim pour leur aide, le temps passé ensemble et le partage de leur expertise qui ont contribué au succès de mon PFE. Un grand merci pour tout le corps enseignant du département d'Électronique pour tous leurs enseignements et leurs volontés réelles de nous transmettre le savoir.

...

Pour finir, je tiens à remercier toute personne qui, d'une manière ou d'une autre, m'a aidé dans l'élaboration de ce travail.

...

# Table des matières

Liste des tableaux

Table des figures

Liste des Abréviations

<b>Introduction générale</b>	<b>11</b>
<b>1 La commande SHE PWM</b>	<b>14</b>
1.1 Introduction	14
1.2 Présentation du banc d'essai utilisé	15
1.2.1 Description du matériel	15
1.2.2 Caractéristiques du moteur asynchrone triphasé étudié	17
1.3 Commande des moteurs asynchrones	20
1.3.1 Commande scalaire	20
1.3.2 Le découplage entre la fréquence et l'amplitude du fondamental	22
1.4 La technique SHE PWM	26
1.4.1 Introduction	26
1.4.2 Fondement théorique	26
1.5 Génération de la base de données	30
1.5.1 Méthode théorique de calculs des angles exacts de commutation	30
1.5.2 Algorithme de calcul des angles initiaux	31
1.5.3 Algorithme de calcul des angles exacts de commutation	31
1.5.4 Structure de code MATLAB	33
1.5.5 Résultats et interprétations	35
1.6 Conclusion	39
<b>2 Les réseaux de neurones artificiels</b>	<b>41</b>
2.1 Introduction	41
2.2 Historique	42
2.3 Du neurone biologique au neurone artificiel	43
2.3.1 Réseau de neurones biologiques	43
2.3.2 Neurone biologique	44
2.3.3 Fonctionnement du neurone biologique	45
2.3.4 Le réseau de neurones artificiels	46
2.3.5 Le neurone artificiel et son modèle mathématique	46
2.3.6 L'architecture d'un RNA	47
2.3.7 Les réseaux feedforward multicouches	48
2.3.8 Fonction d'activation	49
2.3.9 Analyse comparative.	50

2.4	<b>Processus d'apprentissage</b>	54
2.4.1	Type d'apprentissage	54
2.4.2	Apprentissage off-line	55
2.4.3	Apprentissage supervisé	55
2.4.4	Algorithme d'apprentissage "rétropropagation du gradient" du Réseau multicouches MLP	56
2.5	<b>Choix de topologie et l'implémentation du processus d'apprentissage sur MATLAB</b>	56
2.5.1	Solution originale	57
2.5.2	Topologie	60
2.5.3	Apprentissage des six RNAs	61
2.6	<b>Implémentation complète de la solution sur MATLAB</b>	62
2.6.1	Bloc de normalisation	62
2.6.2	Implémentation de la couche cachée et la couche de sortie	62
2.6.3	Bloc de dé-normalisation	63
2.7	<b>Génération des sorties des 6 RNAs</b>	63
2.7.1	Résultats	64
2.7.2	interprétations	64
2.8	<b>Conclusion</b>	66
3	<b>Générateur intelligent de multi-IP cores</b>	67
3.1	<b>Introduction</b>	67
3.2	<b>Le circuit FPGA utilisé et méthodes de développement</b>	68
3.3	<b>Prise en main de l'application C++</b>	68
3.3.1	Interface de l'application C++	69
3.3.2	Étapes de configuration	69
3.4	<b>L'application vue de l'intérieur</b>	72
3.4.1	Structure de l'application C++	72
3.4.2	Développement des fonctions de base de l'application C++	73
3.5	<b>Description VHDL</b>	74
3.5.1	Structure de la description VHDL	74
3.5.2	Blocs fonctionnels spécifiques à l'application SHE PWM	74
3.5.3	Blocs fonctionnels spécifiques à la génération des RNAs.	80
3.6	<b>Simulation et validation expérimentale</b>	100
3.6.1	Simulation	100
3.6.2	Le banc d'essai	100
3.6.3	Résultats expérimentaux	101
3.7	<b>Conclusion</b>	107
	<b>Conclusion générale</b>	<b>108</b>
	<b>Bibliographie</b>	<b>110</b>

# Liste des tableaux

1.1	Caractéristiques principales du moteur . . . . .	20
1.2	Tableau récapitulatif des deux commandes Scalaires & Vectorielle	20
1.3	Les angles de commutation pour M=23 . . . . .	36
1.4	Nombre d'harmoniques à éliminer en fonction des intervalles de variation de $i_m$ . . . . .	37
2.1	L'erreur moyenne et maximale entre les angles de commutation PWM exacts et ceux calculés par l'algorithme MATLAB . . . . .	64
3.1	Intervalle de variation de $i_m$ pour chaque réseau RNA-i . . . . .	75
3.2	La LUT de la fonction tangente-hyperbolique . . . . .	84
3.3	La LUT de la fonction $f(x) = \text{sig}2(x) - \frac{1}{2}$ . . . . .	88

# Table des figures

1.1	Image réelle de la machine asynchrone . . . . .	15
1.2	Schéma de connexions de l'ensemble des éléments . . . . .	16
1.3	Vue éclatée d'un moteur asynchrone. . . . .	17
1.4	Représentation du stator. . . . .	17
1.5	Représentation du rotor . . . . .	18
1.6	Courbe du couple utile $\Gamma_u$ en fonction de la fréquence de rotation $n$ . . . . .	18
1.7	Schéma de câblage . . . . .	19
1.8	Déplacement de la caractéristique <i>Couple-vitesse</i> en fonction de la fréquence . . . . .	21
1.9	Schéma synoptique du système de commande de moteur asynchrone . . . . .	23
1.10	Schéma synoptique du système de commande de moteur asynchrone à boucle de retour . . . . .	25
1.11	Tension normalisée de l'une des sorties SHE PWM . . . . .	27
1.12	Tension de sortie normalisée d'un SHE PWM à cinq angles. . . . .	27
1.13	Organigramme illustrant l'algorithme de Taufik, Mellitt et Goodmanla (TMG). . . . .	31
1.14	Structure de code MATLAB. . . . .	33
1.15	Trajectoires des angles de commutation dans les six intervalles selon im. . . . .	35
1.16	Résultats : Signaux PWMs & Analyse spectrale. . . . .	38
2.1	Prévisions de chiffre d'affaires du marché des applications d'IA en Europe . . . . .	41
2.2	Le réseau de neurones biologiques . . . . .	44
2.3	Le neurone biologique . . . . .	44
2.4	Le modèle d'un neurone artificiel . . . . .	46
2.5	Exemple d'un réseau feedforward multicouches . . . . .	48
2.6	Liste de fonctions d'activation usuelles . . . . .	49
2.7	Les types d'apprentissages . . . . .	54
2.8	Étapes d'analyse de la base de données et d'implémentation . . . . .	59
2.9	Topologies des 6 réseaux de neurones artificiels . . . . .	60
2.10	Apprentissage des 6 réseaux de neurones artificiels . . . . .	61
2.11	Processus d'implémentation et de génération des sorties d'un RNA . . . . .	63
2.12	Résultats : Signaux PWMs & Analyse spectrale à la sortie du RNA-4 . . . . .	65
3.1	Interface de l'application C <sup>++</sup> . . . . .	69
3.2	Exemple du dossier et fichiers sources pour 6 RNAs . . . . .	70

3.3 Exemples de messages pour 2 RNAs . . . . .	71
3.4 Exemple de dossier final pour 6 RNAs dans les deux modes de fonctionnement série et parallèle. . . . .	71
3.5 Exemple de code VHDL généré pour 6 RNAs dans les deux modes de fonctionnement série et parallèle. . . . .	72
3.6 Exemples de messages pour 2 RNAs . . . . .	72
3.7 Schéma synoptique du module "générateur PWMs" . . . . .	76
3.8 Schéma de branchement des trois modules . . . . .	78
3.9 Algorithme de la génération des signaux PWMs . . . . .	79
3.10 La fonction Tangente-hyperbolique . . . . .	81
3.11 La fonction Tangente-hyperbolique . . . . .	82
3.12 Implémentation de la fonction tanh . . . . .	85
3.13 Simulation de décodeur de plages d'adressages . . . . .	86
3.14 Simulation de décodeur de plages d'adressages . . . . .	87
3.15 La fonction sigmoïde2(x) . . . . .	87
3.16 La simulation de la fonction sigmoïde2(x) . . . . .	88
3.17 Implémentation de la fonction tanh . . . . .	89
3.18 Somme des produits . . . . .	91
3.19 Somme des produits séquentiels . . . . .	91
3.20 Schéma synoptique d'un seul neurone artificiel . . . . .	93
3.21 Concept de multiplexage des couches . . . . .	94
3.22 État de flux de deux machines d'état . . . . .	95
3.23 Schéma synoptique d'un RNA en fonctionnement parallèle . . . . .	97
3.24 Schéma synoptique d'un RNA en fonctionnement série . . . . .	99
3.25 Angles de commutation sur Vivado pour $i_m=50\%$ . . . . .	100
3.26 Le banc d'essais . . . . .	100
3.27 Visualisation des tensions de phases de l'onduleur pour (a) : $i_m=64\%$ ( $T=31.25ms$ ) . . . . .	101
3.28 Visualisation des tensions de phases de l'onduleur pour (b) : $i_m=32\%$ ( $T=62.5ms$ ) . . . . .	101
3.29 Visualisation de la tension entre deux phases pour $i_m=64\%$ ( $T=31.25 ms$ ), (a) : La tension d'entrée =50VDC . . . . .	102
3.30 Visualisation de la tension entre deux phases pour $i_m=64\%$ ( $T=31.25 ms$ ), (b) : La tension d'entrée =70 VDC . . . . .	102
3.31 Les tensions de phases, la tension entre phases correspondante et le spectre fréquentiel de cette tension entre phases pour une tension d'entrée de 50 VDC, (a) : $i_m=64\%$ ( $T=31.25ms$ ). . . . .	103
3.32 Les tensions de phases, la tension entre phases correspondante et le spectre fréquentiel de cette tension entre phases pour une tension d'entrée de 50 VDC, (b) : $i_m=32\%$ ( $T=62.5ms$ ). . . . .	104
3.33 Les tensions de phases, la tension entre phases correspondante et le spectre fréquentiel de cette tension entre phases pour une tension d'entrée de 50 VDC, (c) : $i_m=16\%$ ( $T=125ms$ ). . . . .	105

# Liste des Abréviations

<b>RNA</b>	<b>R</b> réseau de <b>N</b> eurones <b>A</b> rtificiel
<b>EV</b>	<b>E</b> lectrical <b>V</b> ehicle
<b>SHE</b>	<b>S</b> elective <b>H</b> armonic <b>E</b> lemination
<b>PWM</b>	<b>P</b> ulse <b>W</b> idth <b>M</b> odulation
<b>VHDL</b>	<b>V</b> L <sub>V</sub> <b>H</b> SIC <b>H</b> ardware <b>D</b> escription <b>L</b> anguage
<b>IP</b>	<b>I</b> ntellectual <b>P</b> roperty
<b>IM</b>	<b>I</b> nduction <b>M</b> otor
<b>ENP</b>	<b>E</b> cole <b>N</b> ationale <b>P</b> olytechnique
<b>FPGA</b>	<b>F</b> ield <b>P</b> rogrammable <b>G</b> ate <b>A</b> rray
<b>TMG</b>	<b>T</b> aufik <b>M</b> ellitt <b>G</b> oodmanla
<b>IA</b>	<b>I</b> ntelligence <b>A</b> rtificielle
<b>RNB</b>	<b>R</b> réseau de <b>N</b> eurones <b>B</b> iologique
<b>MLP</b>	<b>M</b> ulti <b>L</b> ayer <b>P</b> erceptron
<b>ASIC</b>	<b>A</b> pplication <b>S</b> pecific <b>I</b> ntegrated <b>C</b> ircuit
<b>DSP</b>	<b>D</b> igital <b>S</b> ignal <b>P</b> rocessor
<b>RBF</b>	<b>R</b> adial <b>B</b> asis <b>F</b> unction

# Introduction générale

De nos jours, on ne peut pas parler de l'intelligence artificielle ou de machine learning sans parler de réseau de neurones artificiels, un ensemble d'algorithmes dont le fonctionnement est inspiré des neurones biologiques et qui s'appuie également sur les méthodes statistiques.

Ces dernières années, l'intelligence artificielle a vécu une accélération, mais pour autant les réseaux de neurones qui sont derrière demeurent un grand mystère. Il faut ouvrir cette boîte noire pour expliquer les résultats, c'est ce que nous allons aborder dans ce travail mais pas seulement ça. On sait tous que l'un des plus puissants atouts des réseaux de neurones est la généralisation à partir d'une base de données d'apprentissage. C'est de ce même point fort que notre application finale sera dotée, c'est le pouvoir de généraliser des algorithmes pouvant créer n'importe quels réseaux de neurones de type "*Feed-Forward neural networks*" ou encore n'importe quel ensemble de réseaux de neurones de ce type fonctionnant dans un même environnement avec ou sans interaction ou dépendance.

La majorité des problèmes de la recherche scientifique se traduisent en un modèle mathématique qui sera représenté sous forme d'un ensemble d'équations algébriques ou différentielles généralement non-linéaires. Pour résoudre le système on fait appel à la machine avec trois méthodes qui ont des avantages tout à fait différents, deux méthodes classiques qui sont la méthode purement analytique et la méthode numérique. La première nécessite impérativement l'existence d'une solution que l'on peut décrire à l'aide d'une fonction mathématique bien connue, son point fort est sa précision qui se limite seulement à la quantification, mais malheureusement pour trouver cette dernière on fait recours à un développement mathématique fastidieux, et ce qui est encore plus désavantageux pour nous les électroniciens est sa description hardware qui est généralement très complexe voir impossible.

La deuxième méthode est l'étude des algorithmes permettant de résoudre numériquement par discrétisation les problèmes de mathématiques continues. Cette méthode peut être facilement décrite en hardware et facilement implémentée. cependant dans beaucoup d'applications le fonctionnement en temps réel est indispensable, et comme le système est itératif ceci va engendrer un compromis entre la précision et le temps de calcul. Donc on n'utilise cette méthode que dans les opérations off-Line ou de faible précision.

La dernière méthode qui constitue d'ailleurs le fruit de notre travail est la résolution par l'utilisation des réseaux de neurones artificiels et des algorithmes génétiques qui ne relèvent, en tout cas pas à première vue, d'une logique cartésienne classique, mais qui conduit à de meilleurs résultats qui combinent les deux avantages, calcul on-line et précision, cependant, cette

méthode relève beaucoup de défis pour un concepteur de RNA par rapport aux méthodes conventionnelles, comme la nécessité d'équilibrer les ressources hardwares, la précision numérique requise et la vitesse d'exécution.

Comme j'ai déjà expliqué dans le troisième paragraphe, il suffit juste d'une thématique traduite en équations mathématiques pour faire appel aux réseaux de neurones artificiels. Toutefois les différents domaines de la recherche ont des orientations et des perspectives très différentes, de ce fait ils n'ont pas forcément une formation en électronique et plus précisément un background solide en descriptions matérielles. Ce travail donne enfin une chance à tous ces domaines non seulement d'avoir la facilité d'implémentation des réseaux de neurones avec des simples clics, mais aussi la possibilité d'exiger plein de paramètres en utilisant une large gamme de configurations possibles avec l'architecture la plus optimisée possible.

La thématique que nous proposons dans ce travail et qui nous permettra de valider le bon fonctionnement de notre générateur de RNA est la commande de vitesse des véhicules électriques (EV) entraînés par un moteur asynchrone. La modulation de largeur d'impulsion avec élimination sélective des harmoniques (SHE PWM) est une alternative intéressante pour la commande de vitesse d'un moteur asynchrone. Cependant, son utilisation est impossible dans les applications temps réel, comme celle des véhicules électriques, vu que les angles de commutation ne peuvent pas être calculés et ensuite générés en temps réel. Pour répondre à ce problème, nous allons utiliser un nouvel algorithme PWM on-line basé sur la théorie des réseaux de neurones artificiels (RNA) en combinaison avec la technique SHE PWM proposée par Dr Amar Guellal. Le principe qui sera utilisé est la commande scalaire  $V/f$  constant, mais on proposera ensuite des blocs permettant le découplage définitif entre l'amplitude et la fréquence du fondamentale.

La finalité de notre travail se résume en une application intelligente basée en  $C^{++}$  qui permet de générer les descriptions VHDL sous forme d'IP cores<sup>1</sup> de non seulement un seul RNA, mais tout un ensemble de RNA largement configurables et en deux modes de fonctionnement que je nome comme suit, mode fonctionnement série et mode fonctionnement parallèle<sup>2</sup>, l'intelligence de ce générateur de RNA réside dans l'étude des différentes caractéristiques et dans la détection des différents points communs dans l'ensemble des réseaux afin de proposer des optimisations hardwares comme c'est le cas dans le fonctionnement série où il est doté d'un pouvoir de décision, en effet s'il trouve que deux ou plusieurs réseaux de neurones ont des caractéristiques qui permettent de les concaténer dans un seul RNA, il les rassemble dans un seul RNA doté des mêmes performances avec une consommation en ressources très proches de la consommation d'un seul et unique RNA, en

---

1. Un IP (intellectual property) core est un bloc de logique ou de données utilisé dans la fabrication d'un FPGA (Field Programmable Gate Array) ou d'un circuit intégré spécifique à l'application (ASIC) pour un produit. Idéalement, il devrait être entièrement portable - c'est-à-dire, capable d'être facilement inséré dans n'importe quelle technologie de fournisseur ou méthodologie de conception. Les récepteurs / émetteurs asynchrones universels (UART), les unités centrales (CPU), les contrôleurs Ethernet et les interfaces PCI sont tous des exemples de cœurs IP [34].

2. L'explication des deux fonctionnements série et parallèle est bien détaillée dans le chapitre 2

plus de ce qu'on a cité l'application propose en option des blocs additionnels permettant une génération de différentes fonctions liées à l'environnement externe de RNA comme les blocs de *normalisation* et *reverse normalisation* qui sont dotés d'une représentation dynamique de données dont on parlerons dans la suite de ce document, et d'autres fonctions liées à l'application SHE PWM comme le *générateur PWM*, le *variateur de plage de fréquence* et le *régulateur interne de fréquence*. Tous ces blocs sont aussi configurables et généralisés pour n'importe quels réseaux de neurones. La réalisation de tous ces blocs était extrêmement difficile, mais les généralisés était un défi encore plus contraignant.

Dans le chapitre 1, nous commencerons par une introduction qui révélera les différents objectifs de notre partie applicative (la commande SHE PWM) et qui mettra l'accent sur sa nécessité dans le secteur des EVs<sup>3</sup>, on présentera ensuite notre banc d'essai en donnant toutes ses caractéristiques et toutes les notions que nous jugeons indispensables pour la compréhension de cette section, nous allons introduire aussi la commande que nous allons exploiter dans ce projet et expliquer comment nous avons pu réaliser des blocs permettant le découplage définitif entre l'indice de modulation et la fréquence du fondamental, et de mettre en évidence leur utilité dans les applications d'asservissement en donnant plusieurs exemples concrets. ensuite, nous exposerons analytiquement le principe de la commande SHE PWM et nous expliquerons tout le processus de génération de la base de données. On terminera enfin par des exemples d'illustrations, des interprétation et une conclusion de ce chapitre

Dans le chapitre 2, nous allons essayer de répondre à plusieurs questions concernant les réseaux de neurones et le mystère qu'il y a derrière , commençant par un bref historique et en développant les différents concepts du neurone biologique qui représente la source d'inspiration pour les neurones artificiels tels que sa compositions, son fonctionnement et son architecture, nous exposerons ensuite une analyse comparative entre ces deux types de neurones : artificiels et biologiques, de plus, nous expliquerons le processus d'apprentissage utilisé, le choix de la topologie où nous exposerons une solution originale pour l'opérationnel et un justificatif concret pour nos choix. Enfin, nous terminerons avec une implémentation complète des différents blocs sur MATLAB, des interprétations et une conclusion.

Dans le dernier chapitre, nous allons pénétrer profondément dans la boîte noire RNA, nous allons expliquer toutes les étapes nécessaires pour la mise en œuvre de l'application  $C^{++}$ , en plus du principe de sa réalisation, nous allons prouver l'efficacité et l'intelligence de cette dernière. Nous présenterons aussi les étapes d'implémentation des différents blocs sur le circuit FPGA. En insistant sur l'optimisation de cette implémentation vu l'importance de cette dernière et son impact sur les performances de notre système.

Enfin, on terminera par une conclusion générale.

---

3. Un véhicule électrique , également appelé EV , utilise un ou plusieurs moteurs électriques ou moteurs de traction pour la propulsion.

## Chapitre 1

# La commande SHE PWM pour le contrôle de vitesse des moteurs Asynchrones

### 1.1 Introduction

Le véhicule électrique n'est pas un prototype de plus que nous présentent les constructeurs lors des salons automobiles ni une figure de style dévoilant un futur hypothétique des transports. S'il préfigure l'une des meilleures alternatives pour nos déplacements de demain, c'est avant tout un véhicule qui est utilisé depuis plus d'un siècle et dont des millions d'unités (tous véhicules électriques confondus) circulent tous les jours [6].

À notre époque, la voiture électrique fait l'objet de toutes les attentions. Elle apporte une réponse à des problèmes sociétaux d'actualité : qualité de l'air, changement climatique, épuisement des réserves pétrolières... Cela dit, il faut éviter de céder à la tentation de voir dans le véhicule électrique la seule issue technologique pour l'avenir et tenir compte de quelques limitations importantes qui conditionneront le succès de ce mode de propulsion [9].

En fait, la technologie la plus importante et qui représente le plus de challenge est la batterie ou le stockage d'énergie en général, qui nécessite plus d'attention en vue d'augmenter l'autonomie des EVs [8]. D'autre part, l'électronique de puissance et l'entraînement des moteurs en particulier, faisant l'objet de la partie applicative de ce projet de fin d'études, sont également une technologie fondamentale, qui devrait être prise en compte afin d'améliorer les performances entières des EVs y compris l'autonomie [8, 14, 11].

Nous avons préféré utilisé dans ce travail le moteur asynchrone (IM) par rapport à son homologue le moteur à courant continu (DCM) en raison de sa robustesse, entretien moindre, une puissance plus élevée par rapport au poids et un coût moindre par rapport à la puissance [22].

Pour commander ce moteur nous avons utilisé un onduleur triphasé réalisé par des étudiants de l'ENP. Dans la pratique, les stratégies de commande produisent des harmoniques indésirables qui ont de nombreux effets non-acceptables sur le fonctionnement du moteur, tel que l'échauffement qui est dû à l'augmentation des pertes et les pulsations de couple qui deviennent

gênantes surtout aux faibles vitesses [1, 41].

Une solution bien connue pour résoudre le problème des harmoniques indésirables est l'utilisation de la commande en modulation de largeur d'impulsion avec élimination sélective des harmoniques (SHE PWM) [41, 23].

Dans ce chapitre, nous allons commencer par une présentation générale du banc d'essai et de la machine asynchrone utilisés. nous parlerons des différentes caractéristiques de cette dernière comme sa constitution et son principe de fonctionnement. ensuite, nous expliquerons le type de commande que nous allons exploiter dans ce projet sans oublier de parler des blocs additionnels permettant le découplage final entre la fréquence et l'indice de modulation et de leur utilité dans les applications d'asservissement. Par la suite, nous introduiront le principe de la commande SHE PWM et la méthode utilisée pour la génération de la base de données, nous terminerons enfin par quelques exemples d'illustration, des interprétations et une conclusion.

## 1.2 Présentation du banc d'essai utilisé

### 1.2.1 Description du banc d'essai et des différents composants principaux de la machine asynchrone

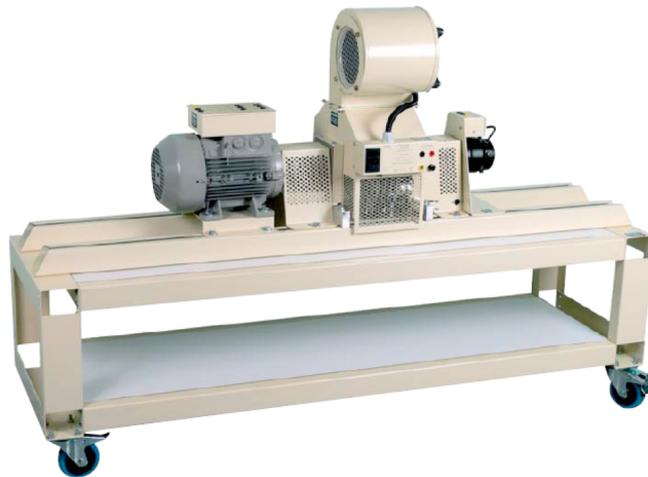


FIGURE 1.1 – Image réelle de la machine asynchrone

Le système global (commande+machine) est composé des éléments principaux suivant :

- Deux alimentations continues, une pour alimenter le bloc de commande de 12 V de l'onduleur triphasé, l'autre de puissance utilisée pour amplifier les trois signaux de commande PWM.
- Une carte FPGA Rev D ZedBoard dans laquelle nous avons implémenté un régulateur de fréquence, un variateur de vitesse et convertisseur de

- fréquence à  $V/f$  constant qui commandent la vitesse de rotation du moteur suivant différentes rampes d'accélération ou de décélération.
- Le groupe moteur utilisé comprend un moteur asynchrone triphasé 1500 W à rotor à cage couplé à une charge, le frein à poudre. Il est composé de différents éléments (moteur et accessoires de mesures).
  - L'élément frein à poudre permet de freiner le moteur en lui appliquant un couple résistant réglable manuellement.
  - Le capteur de couple relié au module GRANMECA-V2 permet l'affichage du moment du couple utile du moteur, en relevant et affichant la vitesse et le couple.
  - Une dynamo tachymétrique au bout de l'arbre reliée au même module permet l'affichage de la fréquence de rotation en tr/min du moteur.
  - Une interface de supervision pour PC permet d'afficher en temps réel sur un PC les différentes valeurs (couple, vitesse de rotation, puissance utile).

Ci-dessous une figure montrant un schéma de connexions de l'ensemble des éléments :

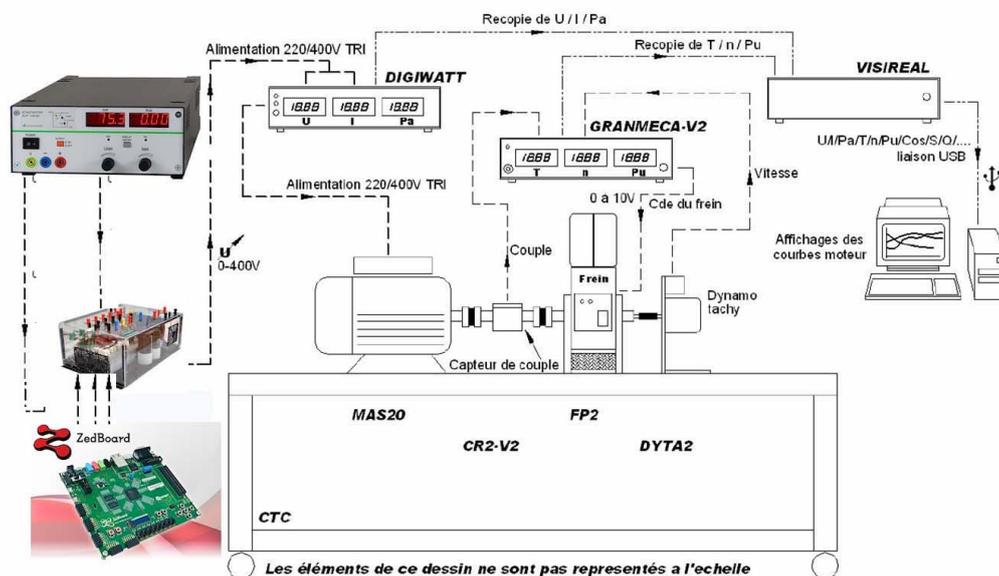


FIGURE 1.2 – Schéma de connexions de l'ensemble des éléments

## 1.2.2 Caractéristiques du moteur asynchrone triphasé étudié

### Constitution

Les machines asynchrones se composent de plusieurs éléments comme le montre la figure 1.3 présentée ci-dessous :

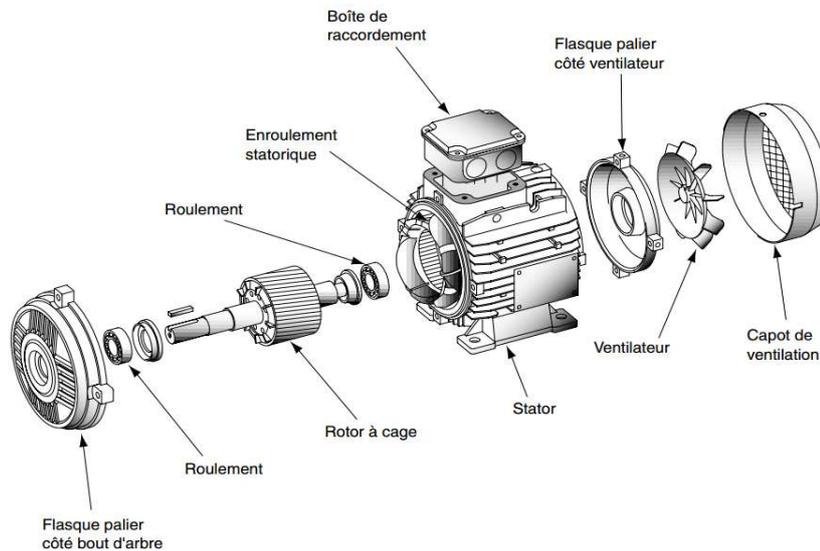


FIGURE 1.3 – Vue éclatée d'un moteur asynchrone.

Les deux principaux éléments sont appelés le stator et le rotor :

1. Le stator = inducteur

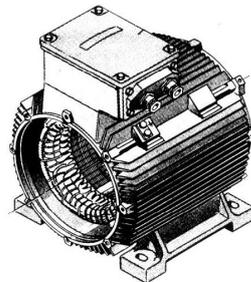


FIGURE 1.4 – Représentation du stator.

Il est constitué de trois enroulements (bobines) parcourus par des courants alternatifs triphasés et possède  $p$  pair de pôles. Les courants alternatifs dans le stator créent un champ magnétique tournant de pulsation de synchronisme :

$$\Omega_s = \frac{\omega}{p} \quad (1.1)$$

$\Omega_s$  : vitesse synchrone de rotation du champ tournant en  $rad.s^{-1}$  avec  $\Omega_s = 2.\pi.n_s$  ( $n_s$  vitesse de synchronisme en  $tr/min$ ).

$\omega$  : pulsation des courants alternatif en  $rad.s^{-1}$  avec  $\omega = 2.\pi.f$

## 2. Rotor = induit

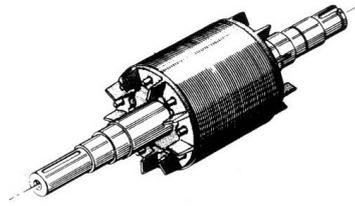


FIGURE 1.5 – Représentation du rotor

Le rotor n'est relié à aucune alimentation. Il tourne à la vitesse  $n$ .

C'est un rotor à cage d'écureuil, constitué de barres conductrices. Les extrémités de ces barres sont réunies par deux couronnes conductrices, on dit que le rotor est en court-circuit.

Le rotor tourne à la vitesse  $n$  plus petite que la vitesse de synchronisme  $n_s$ . Il y a l'apparition d'un glissement noté  $g$  :

$$g = \frac{(n_s - n)}{n_s} \quad (1.2)$$

### Caractéristiques et principe de fonctionnement du moteur asynchrone triphasé

#### 1. Fonctionnement à vide

A vide le moteur n'entraîne pas de charge. Le glissement est nul. Soit  $g = 0$  et donc  $n_0 = n_s$

#### 2. Fonctionnement en charge

Le moteur fournit maintenant de la puissance active, le stator appelle un courant actif. La courbe ci-dessous donne la valeur du couple utile  $\Gamma_u$  en fonction de la vitesse de rotation  $n$  :  $\Gamma_u = f(n)$ .

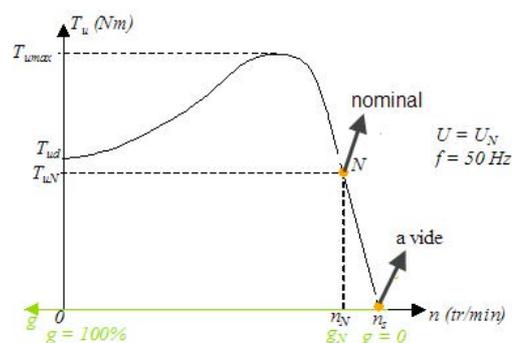


FIGURE 1.6 – Courbe du couple utile  $\Gamma_u$  en fonction de la fréquence de rotation  $n$

Caractéristique obtenue pour une tension de phase  $U$  constant et une fréquence constante.

On remarque que le couple est important au démarrage et il varie de façon presque linéaire au voisinage de la fréquence de rotation nominale. Cette zone correspond au fonctionnement normal du moteur.

### 3. Couplage étoile/triangle

Le branchement des bobines sur le réseau s'effectue au niveau de la plaque à borne située sur le dessus du moteur. On dispose ainsi de 6 connexions, une pour chacune des extrémités trois bobines. Les bornes sont reliées aux bobines suivant le schéma 1.7

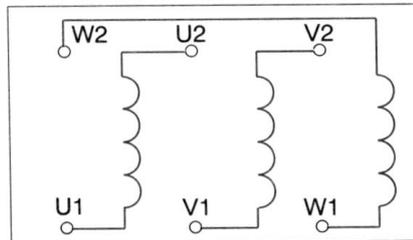


FIGURE 1.7 – Schéma de câblage

L'arrivée de l'énergie se fait en  $U1, V1, W1$ .

Deux types de branchement possibles étoile ou triangle. Dans notre expérience, nous avons utilisé le branchement étoile :

Branchement étoile	Branchement triangle
<p>La tension aux bornes de chaque bobinage est <math>V = U / \sqrt{3}</math> Le courant de chaque bobinage est <math>I</math>.</p>	<p>La tension entre chaque bobinage est <math>U</math>. Le courant de chaque bobinage est <math>J = I / \sqrt{3}</math></p>

#### 4. Caractéristiques principales du moteur

Le tableau ci-dessous résume les caractéristiques principales du moteur utilisé :

TABLE 1.1 – Caractéristiques principales du moteur

Puissance Utile	1500W
Vitesse de rotation	1423tr/min
Nombre de pôles	4
Fréquence réseau	50Hz
Tension d'alimentation en couplage Triangle	230V
Tension d'alimentation en couplage Etoile	400V
Intensité absorbée en couplage Triangle	5.9A
Intensité absorbée en couplage Etoile	3.4A
Facteur de puissance	0.85

### 1.3 Commande des moteurs asynchrones

On distingue deux types de commandes : les commandes scalaires et les commandes vectorielles, le tableau 1.2 ci-dessous est un récapitulatif des deux commandes permettant de résumer les grands axes de ressemblances et de différences entre les deux méthodes :

TABLE 1.2 – Tableau récapitulatif des deux commandes Scalaire & Vectorielle [7]

Commande scalaire	Commande vectorielle
<ul style="list-style-type: none"> <li>• Basée sur le modèle régime permanent</li> <li>+ Simple à implanter</li> <li>- Dynamique lente</li> </ul>	<ul style="list-style-type: none"> <li>• basée sur le modèle transitoire</li> <li>+ Précise et rapide</li> <li>+ Contrôle du couple à l'arrêt</li> <li>- Chère (encodeur incrémental ou estimateur de vitesse, DSP...)</li> </ul>
Contrôle des grandeurs en amplitude	Contrôle des grandeurs en amplitude et en phase

Dans notre cas, nous avons opté pour la commande scalaire que nous allons développer dans la suite de ce chapitre.

#### 1.3.1 Commande scalaire

Plusieurs commandes scalaires existent selon que l'on agit sur le courant ou sur la tension. Elles dépendent surtout de la topologie de l'onduleur utilisé (onduleur de tension ou de courant). Actuellement, l'onduleur de tension est le plus utilisé en petite et moyenne puissance avec une commande en  $V/f$  constant [26] expliquée ci-dessous :

### contrôle en V/f constant

Son principe est de maintenir  $V/f = \text{Constant}$  ce qui signifie garder le flux constant. Le contrôle du couple se fait par l'action sur le glissement. En effet, d'après le modèle établi en régime permanent, le couple maximum s'écrit :

$$C_{max} = \frac{3p}{2Nr} \left( \frac{V}{\Omega_s} \right)^2$$

$V$  est la tension efficace d'entrée du moteur (d'une phase),  $\Omega_s$  est la pulsation statorique,  $Nr$  est l'inductance de fuite rotorique et est  $p$  le nombre de paires de pôles.

On voit bien que le couple est directement proportionnel au carré du rapport de la tension sur la fréquence statorique, par conséquent pour faire varier la vitesse du moteur, il faut faire varier la fréquence et la valeur efficace des tensions d'alimentation conjointement.

On peut caractériser deux régimes de fonctionnement :

- ( $\Omega < \Omega_s$ ) : Dans ce cas la fréquence est en dessous de la fréquence nominale, le rapport  $V/f$  doit être maintenu constant pour que le flux totalisé reste approximativement constant, on obtient un couple de sortie maximum  $C_{max}$  constant. Voir la figure 1.8.
- ( $\Omega > \Omega_s$ ) : On est en régime de "défluxage", ce régime permet de dépasser la vitesse nominale de la machine, on l'appelle donc aussi régime de survitesse. Dans ce cas, le couple est proportionnel à  $\frac{1}{f^2}$ , voir la figure 1.8.

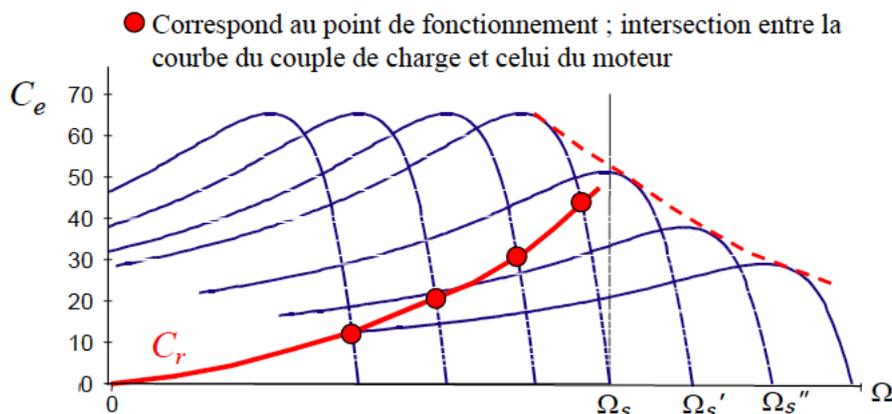


FIGURE 1.8 – Déplacement de la caractéristique Couple-vitesse en fonction de la fréquence

Dans notre étude, on s'intéresse qu'à la région située en dessous de la fréquence nominale, c'est au régime ( $\Omega < \Omega_s$ ), voir 1.3.1

### 1.3.2 Le découplage entre la fréquence et l'amplitude du fondamental

Dans cette partie, nous allons essayer d'expliquer la relation de dépendance entre l'indice de modulation<sup>1</sup> et la fréquence du fondamental<sup>2</sup> mais aussi d'expliquer la problématique qui nous a poussé à découpler la dépendance entre la fréquence et l'amplitude du fondamental créée par le biais de la commande  $v/f$  constant en donnant quelques exemples.

Expliquant tout d'abord la relation existante entre la fréquence et l'amplitude (l'indice de modulation  $im$ ) du fondamental.

#### Relation de dépendance "fréquence & Amplitude"

La commande utilisée est  $v/f$  constant ce qui implique :

$$\frac{V}{f} = C^{te} \quad (1.3)$$

Si on prend un cas particulier dans lequel l'indice de modulation  $im = 1$ , dans ce cas précis, la fréquence est égale à la fréquence nominale  $f_0$  et l'amplitude est maximale égale à l'amplitude de la tensions d'aimantation  $V = \frac{E}{2}$ , la constante dans la relation (1.3) devient alors :

$$\frac{V}{f} = \frac{E}{f_0}$$

Ce qui implique :

$$\frac{V}{E} = \frac{f}{f_0} = im \quad \text{voir l'expression de } im \text{ en } (1.8)$$

D'où

$$f = im f_0 \quad (1.4)$$

Dans notre cas, la fréquence nominale par défaut<sup>3</sup> est  $f_0 = 50\text{Hz}$

L'expression(1.4) montre bien qu'il existe une dépendance mutuelle<sup>4</sup> entre la fréquence et l'indice de modulation, dans notre travail nous avons ajouté deux blocs pour pouvoir éliminer d'une façon définitive la relation mutuelle existante.

Le schéma synoptique suivant décrira l'emplacement et dans quel niveau interviendra dans chaque bloc :

- 
1. Parler de l'indice de modulation équivaut à parler de l'amplitude du fondamental.
  2. Pour comprendre pourquoi on parle du fondamental référez vous à la section 1.4 dans laquelle la technique SHE PWM est expliquée.
  3. Le mot par défaut est utilisé car la fréquence  $f_0$  est paramétrable et la commande qui va être expliquée par la suite utilise la fréquence 50 Hz par défaut.
  4. Le mot mutuelle est important, changer la fréquence revient à changer  $im$  et vice-versa ceci impliquera des changements dans la partie implémentation qui sera décrite par la suite.

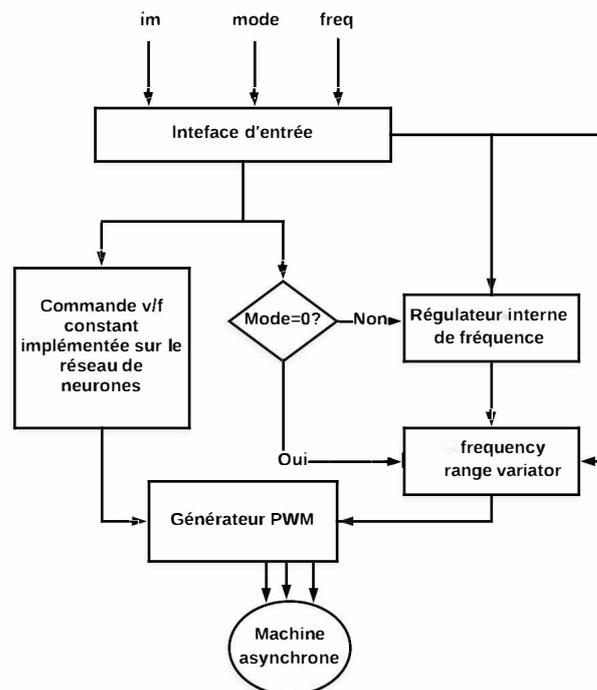


FIGURE 1.9 – Schéma synoptique du système de commande de moteur asynchrone

### Le bloc variateur de plage de fréquence "*frequency range variator*"

La fonctionnalité principale de ce bloc est de modifier toute la plage de fréquence de fonctionnement en se basant sur la fréquence nominale  $f_0$  du bloc générateur de signaux PWM qui commandent l'onduleur, ce changement est géré par les deux entrée externes  $freq$  et  $mode$ .

L'entrée  $mode$  permet de sélectionner le mode variateur de fréquence ( $mode=0$ ) ou d'activer le mode régulateur de fréquence ( $mode=1$ ) que nous allons expliquer par la suite, tandis que l'entrée  $freq$  permet dans ce mode ( $mode=0$ ) de choisir la fréquence nominale, par défaut  $freq = 50$  qui correspond à la fréquence nominale  $f_0 = 50\text{Hz}$ .

Résumons, pour activer le mode variateur de fréquence seulement :  $mode=0$  et  $freq = \text{"valeur de } f_0 \text{ désirée"}$

L'équation qui permet de contrôler toute la plage de fréquence selon l'entrée  $freq$  est :

$$f = im f_0 \frac{50}{freq} \quad (1.5)$$

On voit bien que dans le cas où  $freq=50$  on obtient l'équation classique (1.4).

si on pose :

$$f'_0 = f_0 \frac{50}{freq}$$

On obtient :

$$f = im f'_0 \quad (1.6)$$

Cette équation est la même que celle trouvée en (1.8) mais à fréquence  $f_0$  variable.

Dans la partie implémentation, nous avons utilisé une entrée  $freq$  de 7 bits<sup>5</sup> donc :

La plage de variation de  $freq$  est : [1 ; 127]  $\longleftrightarrow$  ce qui équivaut à une plage de variation de  $f_0$  : [19.68 ; 2500] Hz

Ce bloc nous a permis de réaliser une demi-dépendance entre la fréquence et l'amplitude du fondamentale car on peut toujours faire varier la plage de fréquence comme on veut mais en changeant  $im$  on a toujours une variation de fréquence instantanée due à la loi  $v/f$  constant comme le montre l'équation (1.6). L'autre moitié de dépendance est réalisée à l'aide du bloc présenté ci-après.

### Le bloc régulateur interne<sup>6</sup> de fréquence

Ce circuit permet le découplage définitif de la fréquence et l'indice de modulation, comme nous avons vu précédemment à l'aide de l'équation (1.6) si on change  $im$  le bloc de commande  $v/f$  constant va imposer une fréquence de travail  $f$ , cependant si on veut garder une fréquence fixe quelque soit  $im$  on active ce bloc en mettant l'entrée externe  $mode = 1$ , dans ce cas l'entrée  $freq$  nous permettra de fixer une fréquence de travail  $f$  de notre choix selon l'équation suivante :

$$f = \frac{50}{freq} \text{ Hz} \quad (1.7)$$

Dans la partie implémentation, nous allons voir que ce bloc est plus difficile à réaliser par rapport au variateur de fréquence, car il dépend de la sortie de réseau de neurones contrairement au variateur de fréquence.

### Exemples montrant l'utilité de découplage

L'utilité de ce découplage s'élargit à plusieurs thématiques qui cherchent à avoir une dépendance entre  $im$  et  $f$ , notamment si on cherche à faire un asservissement de fréquence ou de l'indice de modulation, dans ce cas on aura

5. C'est un choix arbitraire, l'utilisateur pourra ensuite changer le nombre de bits pour atteindre la plage de fréquence désirée.

6. Le mot interne est utilisé ici pour souligner le fait que la régulation de fréquence est interne, la modification de l'indice de modulation crée tout un changement de paramètres dans le bloc réseau de neurones et donc un changement de fréquence ce qui engendra une correction à l'intérieur du circuit de commande, la consigne dans ce cas est l'entrée  $freq$ .

qu'à attaquer l'entrée *freq* définie précédemment.

L'un des problèmes que nous pourrions discuter est en maintenant dans les faibles fréquences le rapport  $v/f$  constant, la valeur de la tension sera très faible, ce qui crée des problèmes de démarrage du moteur asynchrone. L'une solution est d'asservir le système en maintenant l'entrée *freq* constante et en augmentant la valeur de l'indice de modulation *im* présentées précédemment, ce qui nous permettra d'atteindre facilement les faibles valeurs de vitesses en augmentant l'indice de modulation sans toucher à la fréquence (et par conséquent à la vitesse)

Nous pouvons citer aussi l'une des fonctionnalités indispensable de bloc régulateur interne de fréquence que nous avons testé lors de nos expériences, nous pouvons augmenter le couple moteur en fixant une fréquence donnée et en augmentant l'indice de modulation, par exemple : si on fixe la fréquence à  $f = 25\text{Hz}$  on devrait avoir selon l'équation (1.4) un indice de modulation  $im = 0.5$ . Or en utilisant ce régulateur, nous pouvons fixer la fréquence  $f$  à 25 Hz et doubler l'indice de modulation à  $im = 1$  ce qui donnera une vitesse plus faible, mais un couple moteur plus important. Cette fonctionnalité peut être employée afin d'asservir d'une part le fonctionnement à basse vitesse où la chute de tension ne peut pas être négligée [7], on compense alors en augmentant la tension à travers *im* sans toucher à la fréquence et donc à la vitesse (voir le bloc Compensateur/Correcteur figure 1.10). D'autre part, si la machine est chargée, la vitesse a tendance à baisser, le régulateur va fournir plus de couples (donc plus de glissement) afin d'assurer cet équilibre. La pulsation statorique est donc modifiée pour garder cet équilibre (voir le bloc Compensateur/Correcteur figure 1.10). La tension est calculée de manière à garantir le mode de contrôle en  $V/f$  de la machine ce qui est facilement réalisable grâce au découplage comme le montre la figure 1.10 suivante :

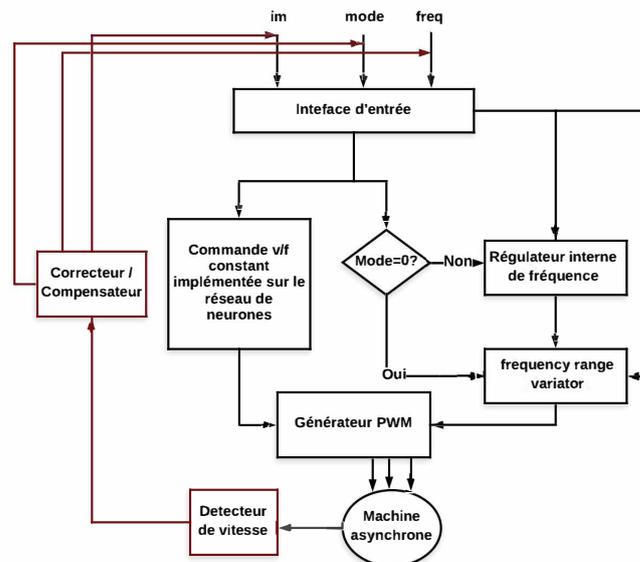


FIGURE 1.10 – Schéma synoptique du système de commande de moteur asynchrone à boucle de retour

## 1.4 La technique SHE PWM

### 1.4.1 Introduction

La technique SHE PWM, littéralement "Selective Harmonic Eliminated Pulse-Width Modulation" a été fondée la première fois par Turnbull en 1964 puis développée par Patel et Hoft [31, 30].

Cette technique consiste à former l'onde de sortie d'une succession de créneaux de largeurs variables et contrôlables. Les angles de commutation sont déterminés de façon à éliminer certains harmoniques gênants dans l'onde de sortie qui se trouve dans la gamme de fréquence dont le moteur asynchrone est très sensible, ceci améliore ainsi le rendement du système onduleur-machine par la réduction des ondulations du couple, en plus des pointes de courant et des pertes dans la machines[5, 35].

Malgré la difficulté de calcul des angles de commutation, la technique SHE PWM présente plusieurs avantages par rapport à la PWM engendrée à modulation sinusoïdale (SPWM)<sup>7</sup> [36, 15, 3], meilleure élimination des harmoniques indésirables, pertes de commutation moindres, la limite minimale de largeur d'impulsion peut être atteinte facilement et la sur-modulation est possible.

### 1.4.2 Fondement théorique de la technique SHE PWM

Le but de cette technique est d'asservir l'amplitude du fondamental et d'annuler les amplitudes des  $(m - 1)$  premiers harmoniques, où  $m$  représente le nombre d'angles de commutation de la tension de sortie de l'onduleur par quart-d'onde.

La tension à la sortie de l'onduleur est définie en fonction des angles exacts de commutation  $\alpha_1, \alpha_2 \dots \alpha_m$  qui correspondent aux instants de commutation de la tension d'une valeur positive  $+\frac{E}{2}$  à une valeur négative  $-\frac{E}{2}$  ou inversement, voir la figure 1.11.

Les signaux MLI décrivant les trois tensions de sortie du convertisseur doivent posséder des propriétés qui contribuent à orienter leurs caractéristiques vers celles d'une onde sinusoïdale. Afin de s'en approcher le plus possible, on pourra, dans certains cas leur attribuer les mêmes propriétés de symétrie qu'une onde sinusoïdale [13]. Pour atteindre ces attentes, les signaux de sortie doivent être construits suivant ces quartes règles :

1. La tension de sortie de l'onduleur est construite de façon à présenter une symétrie demi-onde (fonction impaire par rapport à l'angle  $\pi$ ).

7. Parmi les variantes de la modulation PWM engendrée, elle consiste à comparer une tension de référence de fréquence  $F_r$ , image du signal souhaité à la sortie, appelée modulante, avec une porteuse triangulaire ou en dents de scie de fréquence  $F_p$ . Les points d'intersection entre la modulante et la porteuse correspondent aux instants de commutation au moment desquels l'onduleur change d'état.

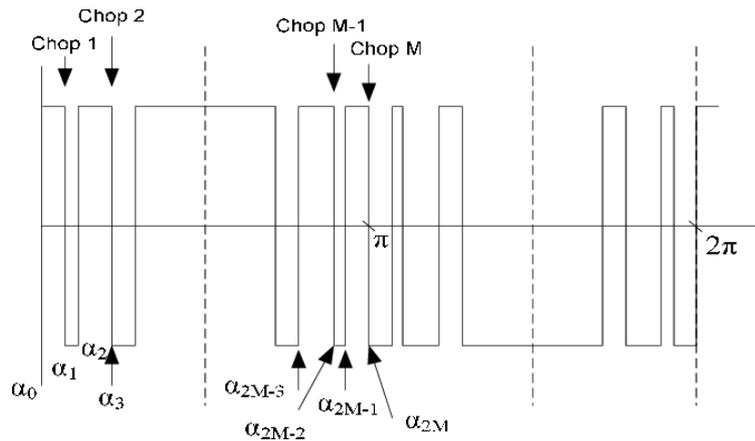


FIGURE 1.11 – Tension normalisée de l’une des sorties SHE PWM [37]

2. La tension de sortie de l’onduleur est construite de façon à présenter une symétrie quart-onde (fonction paire par rapport à l’angle  $\frac{\pi}{4}$ )<sup>8</sup>.
3. L’amplitude du fondamental est fixée à la valeur  $im$  et les amplitudes des  $(m-1)$  premiers harmoniques sont annulés.
4. Dans notre étude on a utilisé un onduleur triphasé, donc les harmoniques trois et multiples de trois sont éliminés automatiquement [36, 23, 18].

Une forme d’onde SHE PWM à cinq angles de commutation est illustrée par la figure 1.12, elle servira d’exemple pour expliquer comment appliquer les règles que nous avons citées ci-dessus.

Dans cet exemple, un système monophasé est supposé et l’indice de modulation de la tension de sortie  $im=85\%$ .

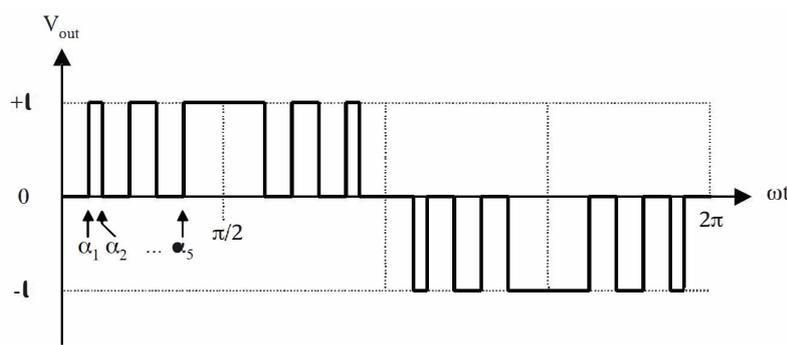


FIGURE 1.12 – Tension de sortie normalisée d’un SHE PWM à cinq angles.

8. Ces deux symétries permettent de supprimer certains types d’harmoniques, ce qui simplifie le développement en série de Fourier de cette tension et réduit le taux d’harmoniques pour la suite voir 1.4.2.

Le taux  $im$  est le taux de modulation défini par :

$$im = \frac{V}{\frac{E}{2}} \quad (1.8)$$

avec  $V$  la valeur de tension du fondamental.

soit  $f(\alpha)$  la tension de sortie où  $\alpha = \omega t$ . Comme tous les signaux peuvent être développés en série de Fourier, on peut écrire  $f(\alpha)$  sous la forme :

$$f(\alpha) = a_0 + \sum_{n=1}^{\infty} (a_n \sin(n\alpha) + b_n \cos(n\alpha)) \quad (1.9)$$

Les coefficients  $a_n$  et  $b_n$  sont donnés par :

$$\begin{aligned} a_0 &= \frac{1}{2\pi} \int_0^{2\pi} f(\alpha) d\alpha \\ a_n &= \frac{1}{\pi} \int_0^{2\pi} f(\alpha) \cos(n\alpha) d\alpha \\ b_n &= \frac{1}{\pi} \int_0^{2\pi} f(\alpha) \sin(n\alpha) d\alpha \end{aligned} \quad n = 1, 2, 3, 4... \quad (1.10)$$

Comme  $f(\alpha)$  présente une symétrie demi-onde voir la règle 1 dans 1.4.2 :

$$f(\alpha + \pi) = -f(\alpha)$$

La valeur moyenne  $a_0$  est nulle et seulement les harmoniques impairs existent. Par conséquent, l'indice  $n$  prend les valeurs impaires 1, 3, 5, 7, 9, ...

Les coefficients  $a_n$  et  $b_n$  sont alors donnés par :

$$\begin{aligned} a_0 &= 0 \\ a_n &= \frac{2}{\pi} \int_0^{2\pi} f(\alpha) \cos(n\alpha) d\alpha \\ b_n &= \frac{2}{\pi} \int_0^{2\pi} f(\alpha) \sin(n\alpha) d\alpha \end{aligned} \quad n = 1, 3, 5, 7... \quad (1.11)$$

En remplaçant  $f(\alpha)$  par sa valeur normalisée voir la figure 1.11, on obtient :

$$a_n = \frac{2}{\pi} \left[ \int_{\alpha_0}^{\alpha_1} (-1)^0 \sin(n\alpha) d\alpha \right] + \dots + \frac{2}{\pi} \left[ \int_{\alpha_{2M}}^{\alpha_{2M+1}} (-1)^{2M} \sin(n\alpha) d\alpha \right] \quad (1.12)$$

Avec  $\alpha_{2M+1} = \pi$ , voir la figure 1.11

En calculant la somme on obtient :

$$a_n = \frac{2}{\pi} \left[ \sum_{i=0}^{2M} \int_{\alpha_{\alpha_i}}^{\alpha_{\alpha_{i+1}}} (-1)^i \sin(n\alpha) d\alpha \right] = \frac{2}{n\pi} \left[ \sum_{i=0}^{2M} (-1)^i (\cos(n\alpha_i) - \cos(n\alpha_{i+1})) \right] \quad (1.13)$$

Avec  $\alpha_0 < \alpha_1 < \alpha_3 \dots < \alpha_{2M+1}$

En développant la somme de l'expression (1.13) :

$$a_n = \frac{2}{n\pi} \left[ \cos(n\alpha_0) - \cos(n\alpha_{2M+1}) + 2 \sum_{i=1}^{2M} (-1)^i \cos(n\alpha_i) \right] \quad (1.14)$$

En remplaçant les deux premiers termes par leurs valeurs :

$$\begin{aligned} \alpha_0 = 0 &\quad \longrightarrow \quad \cos(n\alpha_0) = 1 \\ \alpha_{2M+1} = \pi &\quad \longrightarrow \quad \cos(n\alpha_{2M+1}) = (-1)^n \end{aligned}$$

D'où

$$a_n = \frac{2}{n\pi} \left[ -1 - (-1)^n + 2 \sum_{i=1}^{2M} (-1)^i \cos(n\alpha_i) \right] \quad (1.15)$$

tel que  $n$  ne prend que des valeurs impaires comme on l'a expliqué grâce à la règle 1 dans 1.4.2 ce qui donne :

$$a_n = \frac{4}{n\pi} \left[ 1 + \sum_{i=1}^{2M} (-1)^i \cos(n\alpha_i) \right] \quad (1.16)$$

En utilisant la même approche pour le coefficient  $b_n$  on trouve :

$$b_n = \frac{-4}{n\pi} \sum_{i=1}^{2M} (-1)^i \sin(n\alpha_i) \quad (1.17)$$

D'autre part grâce à la règle 2 la forme d'onde  $f(\alpha)$  peut s'écrire sous forme :

$$f(\alpha) = f(\pi - \alpha)$$

La figure 1.11, permet de déduire que :  $\alpha_i = \pi - \alpha_{2M-i+1}$  avec  $i = 1, 2, \dots, M$

On trouve :

$$\begin{aligned} \sin(n\alpha_i) &= \sin(n(\pi - \alpha_{2M-i+1})) = \sin(n\pi) \cos(n\alpha_{2M-i+1}) - \cos(n\pi) \sin(n\alpha_{2M-i+1}) \\ \cos(n\alpha_i) &= \cos(n(\pi - \alpha_{2M-i+1})) = \cos(n\pi) \cos(n\alpha_{2M-i+1}) - \sin(n\pi) \sin(n\alpha_{2M-i+1}) \end{aligned} \quad (1.18)$$

De plus  $n$  est impair :

$$\begin{aligned} \sin(n\pi) = 0 \ \& \ \cos(n\pi) = -1 &\quad \longrightarrow \quad \sin(n\alpha_i) = \sin(n\alpha_{2M-i+1}) \\ &\quad \longrightarrow \quad \cos(n\alpha_i) = -\cos(n\alpha_{2M-i+1}) \end{aligned}$$

Ainsi selon la règle finale 4 dans 1.4.2 l'indice  $n$  prend des valeurs impaires différentes d'un multiple de 3 :

$$n = 1, 5, 7, 11, \dots$$

En remplaçant ces deux dernière expression dans (1.19), on obtient enfin les



Ce système peut être résolu par la méthode itérative de Newton-Raphson.<sup>10</sup>

### 1.5.2 Algorithme de calcul des angles initiaux

Pour assurer la convergence de cette méthode, on doit obtenir un bon estimé initial de la solution exacte recherchée. Un algorithme a été mis au point pour estimer les valeurs initiales, c'est l'algorithme de Taufik, Mellitt et Goodmanla (TMG) [18] que nous avons implémenté sur MATLAB selon l'organigramme suivant :

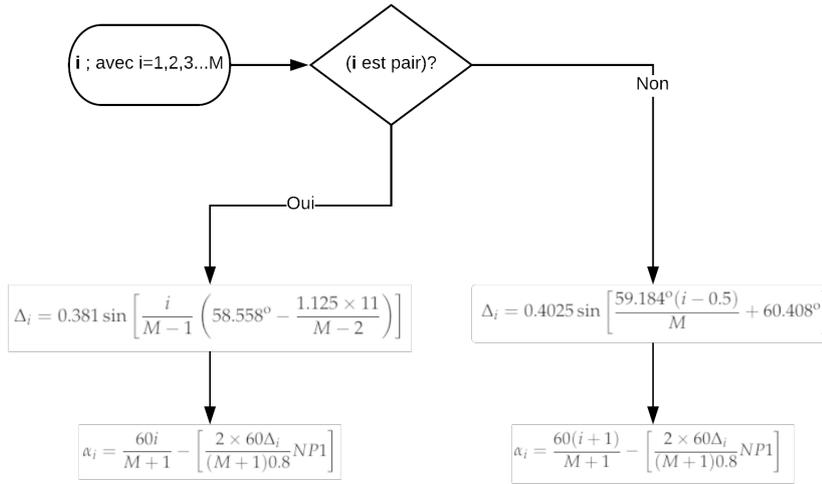


FIGURE 1.13 – Organigramme illustrant l'algorithme de Taufik, Mellitt et Goodmanla (TMG).

NP1 est le taux de modulation identique à  $im$ , il prend des valeurs entre 0 et 0.8. Une fois, ces valeurs calculées, on procède aux étapes itératives de Newton-Raphson expliquées ci-dessus.

### 1.5.3 Algorithme de calcul des angles exacts de commutation

Soit  $\alpha$  le vecteur contenant les angles de commutation définis comme suit :

$$\alpha = (\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_M)$$

Soit  $\alpha^*$  le vecteur solution de système d'équation (1.20)

$$\alpha^* = (\alpha_1^*, \alpha_2^*, \alpha_3^*, \dots, \alpha_M^*) \quad (1.21)$$

Le développement en série de Taylor au voisinage d'un estimé  $\alpha^{(k)}$  proche de  $\alpha^*$ , tel que  $k$  représente la  $k^{me}$  itération, donne

$$a_i(\alpha) = a_i(\alpha^{(k)}) + \alpha^* - \alpha^{(k)} \quad i = 1, 2, 3, \dots, M$$

10. Le système des équations (1.20) satisfait les conditions de la méthode de Newton-Raphson, i.e. tous les équations  $a_i$  sont continues et continument différentiables.

$$\begin{aligned}
a_i(\alpha) &= a_i(\alpha^{(k)}) + \sum_{j=1}^M \left[ \frac{\partial a_i(\alpha)}{\partial \alpha_j} \right]_{\alpha=\alpha^{(k)}} (\alpha^* - \alpha^{(k)}) + \dots + \frac{1}{2} \sum_{j=1}^M \sum_{r=1}^M (\alpha_j^* - \alpha_j^{(k)}) \\
&\quad (\alpha_r^* - \alpha_r^{(k)}) \left[ \frac{\partial^2 a_i(\alpha)}{\partial \alpha_j \partial \alpha_r} \right]_{\alpha=\alpha^{(k)}} + \dots + \dots = 0
\end{aligned} \tag{1.22}$$

En négligeant les termes supérieurs ou égaux au deuxième ordre, l'équation (1.22) devient :

$$a_i(\alpha^{(k)}) = - \sum_{j=1}^M \left[ \frac{\partial a_i(\alpha)}{\partial \alpha_j} \right]_{\alpha=\alpha^{(k)}} (\alpha^* - \alpha^{(k)}) \tag{1.23}$$

La jacobienne de ce système peut s'écrire :

$$Jacob^{(k)} = \left( Jacob_{i,j}^{(k)} \right)$$

avec

$$Jacob_{i,j}^{(k)} = \left[ \frac{\partial a_i(\alpha)}{\partial \alpha_j} \right]_{\alpha=\alpha^{(k)}} \quad i, j = \{1, 2, 3, \dots, M\}^2$$

Explicitement

$$Jacob^{(k)} = \frac{8}{\pi} \begin{pmatrix} \sin(\alpha_1) & -\sin(\alpha_2) & \dots & \sin(\alpha_M) \\ \sin(5\alpha_1) & -\sin(5\alpha_2) & \dots & \sin(5\alpha_M) \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \sin(n\alpha_1) & \sin(n\alpha_2) & \sin(n\alpha_1) & \sin(n\alpha_M) \end{pmatrix} \quad n = 1, 5, 7, 11, \dots \tag{1.24}$$

Le vecteur erreur est défini comme suit

$$\Delta_{\alpha}^{(k)} = \left( \Delta_{\alpha_1}^{(k)}, \Delta_{\alpha_2}^{(k)}, \Delta_{\alpha_3}^{(k)}, \dots, \Delta_{\alpha_M}^{(k)} \right)$$

Où

$$\Delta_{\alpha_i}^{(k)} = \alpha_i^* - \alpha_i^{(k)}$$

Si on pose

$$\gamma^{(k)} = \left( \gamma_1^{(k)}, \gamma_2^{(k)}, \gamma_1^{(k)}, \dots, \gamma_M^{(k)} \right)$$

Avec

$$\gamma^{(k)} = a_i(\alpha^{(k)})$$

Le système d'équations (1.23) devient

$$- Jacob^{(k)} \Delta_{\alpha}^{(k)} = \gamma^{(k)} \tag{1.25}$$

On constitue le système d'équations linéaires (1.25) sur MATLAB, le calcul se fait d'une façon itérative tel que

$$\alpha^{(k+1)} = \alpha^{(k)} + \Delta_{\alpha}^{(k)}$$

Le calcul itératif s'arrête lorsque l'erreur maximale tolérée  $\epsilon$  est atteinte

$$erreur = |\alpha^* - \alpha^{(k)}| \leq \epsilon$$

Pour nos calculs, nous avons utilisé une valeur  $\epsilon = 1.0e^{-15}$ , ce qui donne des résultats très satisfaisants.

#### 1.5.4 Structure de code MATLAB

En réalité, le code MATLAB que nous avons implémenté permet de réaliser beaucoup de fonctions telles que le calcul d'angles, l'apprentissage automatique des réseaux de neurones, la simulation des réseaux de neurones, l'analyse spectrale... etc Le plus intéressant est que toutes ces fonctionnalités sont généralisées et fonctionnent en appelant une seule et unique fonction que nous avons nommée "generate\_all\_parameters\_and\_test".

En effet, ce que nous avons réalisé dans ce projet a été dupliqué, de sorte que toute la partie qui sera par la suite implémentée sur le hardware est étudiée, implémentée et testée sur MATLAB.

Nous allons expliquer les différentes fonctionnalités au fur a mesure que nous avançons dans ce document.

Pour cette partie, nous avons utilisé la structure illustrée par la figure 1.14.

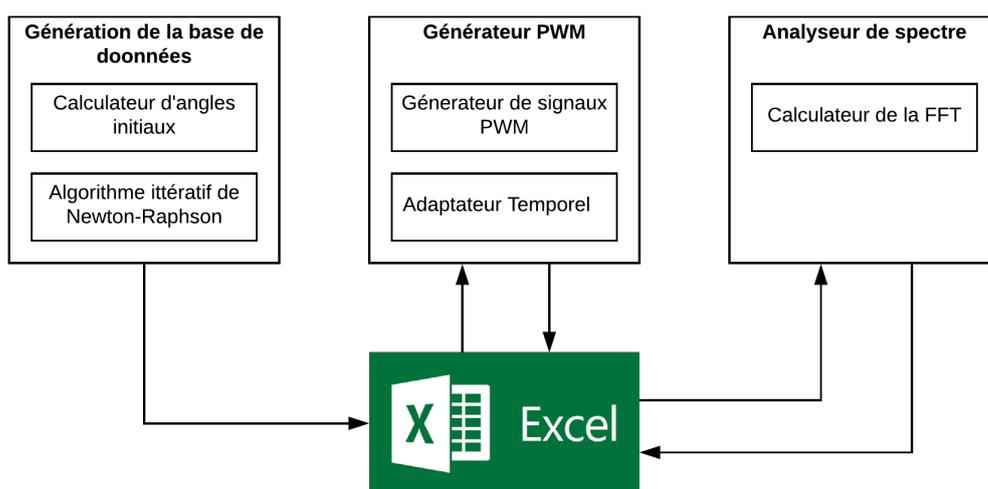


FIGURE 1.14 – Structure de code MATLAB.

1. Le bloc générateur de base de données, génère comme son nom l'indique une base de données contenant les valeurs exactes<sup>11</sup> des angles de commutation, ensuite elle est enregistrée dans un fichier Excel "*data-base.xls*", ce bloc est composé d'un :
  - Calculateur d'angles initiaux qui calcule les angles initiaux selon l'algorithme expliqué en 1.5.2
  - Algorithme itératif de Newton-Raphson qui calcule les valeurs d'angles exacts selon l'algorithme expliqué en 1.5.3
2. Le bloc générateur PWM, génère deux signaux PWM (un signal d'une phase PWM et sa version déphasé par 120°) et le signal entre 2 phases à partir des angles de commutation déjà enregistrés dans le fichier Excel, ces signaux sont enregistrés dans le fichier excel, ce bloc est composé d'un :
  - Générateur PWM : génère les signaux PWM sans unité temporelle.
  - Adaptateur temporel : ce bloc permet de définir la base du temps du signal, ce qui donne à la sortie un signal avec une représentation réelle, on peut directement extraire de ce signal la période et les différents états à chaque instant.  
Pour générer les signaux de commande PWM on a besoin de transformer les angles de commutation en instants de commutation. Dans notre application on a opté pour la commande  $v/f = C^{te}$ .

La relation entre l'angle de commutation et l'instant de commutation est donnée par la relation suivante.

$$t_i = \frac{\alpha_i}{360} \times \frac{1}{f} \quad (1.26)$$

En utilisant les équations (1.4) et (1.26) on trouve :

$$t_i = \frac{\alpha_i}{18000im} \quad (1.27)$$

Et comme nous avons exprimé im en pourcentage, on trouve :

$$t_i = \frac{\alpha_i}{180im} \quad (1.28)$$

De plus, pour générer les signaux de commande à partir des instants de commutation on a utilisé une horloge de 1MHz, donc les instants de commutation doivent être exprimés en  $\mu s$ , équation (1.29).

$$t_i(\mu s) = \frac{10^5 \alpha_i}{18im} \quad (1.29)$$

3. Le bloc Analyseur de spectre, génère les spectres des différents signaux issus du bloc générateur PWM qui sont enregistrés par la suite dans le fichier Excel, ce bloc est composé d'un calculateur de la FFT qui calcule la transformée de Fourier et génère un spectre dans lequel la fréquence

11. voir les détails sur la précision dans la section 1.5.3.

du fondamental correspond exactement à la période du signal PWM et son amplitude à l'indice de modulation  $im$ .

### 1.5.5 Résultats et interprétations

Dans cette partie, nous allons exposer quelques résultats illustrant les différents blocs cités dans la section 1.5.4.

#### Exemple base de données contenant les angles de commutations pour $M=23$ :

Le pas que nous avons utilisé est de  $\Delta_{im} = 0.1\%$  qui correspond à  $\Delta_{im} = 0.001$ .

On donne dans le tableau 1.3 dans la page suivante des angles de commutation pour quelques valeurs de  $im$

#### Trajectoires des angles de commutation

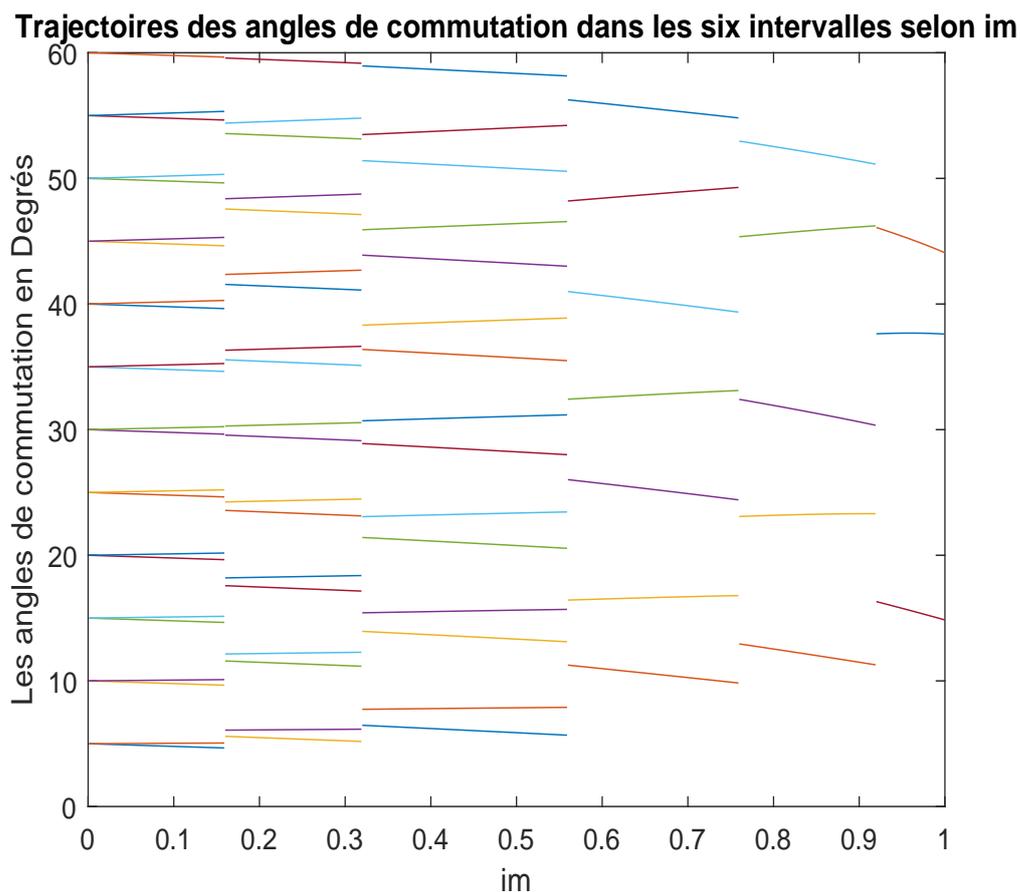


FIGURE 1.15 – Trajectoires des angles de commutation dans les six intervalles selon  $im$ .

Nous avons utilisé un pas 0.001 ce nous a permis de construire une base de données de 1000 valeurs, pour chaque valeur de  $im$  on résout le système

L'angle	La valeur de im en %				
	0,1	0,2	2	2,1	2,2
$\alpha_1$	4,997865555	4,995731065	4,957302193	4,955166797	4,953031352
$\alpha_2$	5,000349241	5,00069849	5,006986044	5,007335392	5,007684743
$\alpha_3$	9,997863412	9,995726692	9,9572428	9,95510351	9,952964081
$\alpha_4$	10,00064679	10,00129352	10,01292173	10,013567	10,01421218
$\alpha_5$	14,99784396	14,99568769	14,956836	14,95467538	14,95251453
$\alpha_6$	15,00090585	15,00181155	15,01808634	15,01898891	15,01989132
$\alpha_7$	19,99781688	19,99563346	19,95628022	19,95409102	19,95190152
$\alpha_8$	20,00113398	20,00226774	20,02263587	20,02376519	20,02489428
$\alpha_9$	24,99778868	24,99557701	24,95570707	24,95348872	24,95127
$\alpha_{10}$	25,00133575	25,00267123	25,02666305	25,02799329	25,02932325
$\alpha_{11}$	29,99776397	29,99552758	29,9552091	29,95296563	29,95072179
$\alpha_{12}$	30,00151401	30,00302774	30,0302254	30,0317336	30,03324151
$\alpha_{13}$	34,99774616	34,99549197	34,95485425	34,95259312	34,95033162
$\alpha_{14}$	35,00167063	35,00334099	35,03335997	35,03502503	35,0366898
$\alpha_{15}$	39,99773786	39,99547539	39,95469413	39,95242533	39,95015618
$\alpha_{16}$	40,00180689	40,00361354	40,03609178	40,03789381	40,03969558
$\alpha_{17}$	44,9977411	44,99548193	44,95476884	44,95250432	44,95023951
$\alpha_{18}$	45,00192373	45,00384727	45,03843893	45,04035889	45,04227866
$\alpha_{19}$	49,99775752	49,99551483	49,95511009	49,95286335	49,95061639
$\alpha_{20}$	50,00202191	50,0040437	50,04041582	50,04243537	50,04445481
$\alpha_{21}$	54,99778843	54,99557673	54,95574331	54,95352907	54,95131468
$\alpha_{22}$	55,00210213	55,00420421	55,04203543	55,04413682	55,04623816
$\alpha_{23}$	59,99783491	59,99566978	59,95668912	59,95452306	59,95235696
	3,5	3,6	15,7	15,8	5,9
$\alpha_1$	4,925265994	4,923129839	4,664215852	4,662071959	4,65992799
$\alpha_2$	5,012226357	5,012575706	5,054727043	5,055073493	5,055419895
$\alpha_3$	9,925138734	9,922997328	9,66271631	9,660554578	9,658392654
$\alpha_4$	10,02259187	10,02323585	10,10031776	10,1009463	10,10157468
$\alpha_5$	14,92440209	14,92223793	14,65846114	14,65626406	14,65406667
$\alpha_6$	15,03160677	15,03250672	15,13986415	15,14073685	15,14160927
$\alpha_7$	19,92340971	19,92121584	19,6532634	19,65102683	19,64878987
$\alpha_8$	20,0395503	20,04067597	20,17482615	20,17591592	20,17700534
$\alpha_9$	24,92239399	24,92017023	24,6482336	24,64596098	24,64368791
$\alpha_{10}$	25,04658674	25,04791268	25,2059997	25,20728491	25,20856973
$\alpha_{11}$	29,9215173	29,91926813	29,6441117	29,64181125	29,63951034
$\alpha_{12}$	30,0528171	30,0543208	30,23384335	30,23530528	30,23676683
$\alpha_{13}$	34,92089825	34,91863152	34,64141607	34,63909919	34,63678187
$\alpha_{14}$	35,05830586	35,05996662	35,25862748	35,26024895	35,26187006
$\alpha_{15}$	39,92062639	39,91835247	39,64052067	39,63820092	39,63588076
$\alpha_{16}$	40,06309605	40,06489433	40,28051282	40,28227737	40,28404161
$\alpha_{17}$	44,92077091	44,91850207	44,64169907	44,63939142	44,63708343
$\alpha_{18}$	45,06721827	45,06913536	45,299596	45,30148742	45,30337862
$\alpha_{19}$	49,92138614	49,91913612	49,64515166	49,64287203	49,64059213
$\alpha_{20}$	50,07069657	50,07271433	50,31593822	50,3179403	50,31994225
$\alpha_{21}$	54,92251525	54,92029895	54,65102407	54,64878887	54,64655349
$\alpha_{22}$	55,0735523	55,07565313	55,32958477	55,33168115	55,33377748
$\alpha_{23}$	59,92419298	59,92202616	59,65942058	59,65724646	59,65507227

TABLE 1.3 – Les angles de commutation pour M=23

d'équations de Patel et Hoft (1.20) afin de trouver la matrice  $\alpha$  des angles de commutation correspondante qui est enregistrée par la suite dans le fichier excel "database.xls".

De plus pour que l'implémentation hardware que nous expliquerons dans un prochain chapitre soit efficace<sup>12</sup>, et pour permettre la convergence de l'étape d'apprentissage<sup>13</sup> qui sera présentée dans le chapitre 2, on divise l'intervalle de variation de  $im$  en six voir la figure 1.15.

Le choix du nombre d'angles de commutation (c'est-à-dire le nombre d'harmoniques à éliminer) dans chaque intervalle dépend de la valeur de l'indice  $im$ , puisque l'effet des harmoniques augmente lorsque  $im$  diminue nous avons pris le choix approprié qui est illustré dans le tableau 1.4 page 27, ce choix permet aussi d'optimiser notre algorithme en minimisant la surface des ressources lors de l'implémentation [15, 3].

TABLE 1.4 – Nombre d'harmoniques à éliminer en fonction des intervalles de variation de  $im$

L'intervalle de $im$	Nombre d'angles de commutation	Nombre d'harmoniques à éliminés	Première fréquence minimale non éliminée
$0.001 < im \leq 0.159$	23	22	$f_{min} = 355\text{Hz}$
$0.16 < im \leq 0.319$	19	18	$f_{min} = 472\text{Hz}$
$0.32 < im \leq 0.559$	15	14	$f_{min} = 752\text{Hz}$
$0.56 < im \leq 0.759$	07	06	$f_{min} = 644\text{Hz}$
$0.76 < im \leq 0.919$	05	04	$f_{min} = 646\text{Hz}$
$0.92 < im \leq 1$	03	02	$f_{min} = 506\text{Hz}$

### interprétation

- Le tableau 1.4 montre bien que toutes les fréquences non éliminées sont très lointaines du spectre fréquentiel dans lequel le moteur est sensible. De ce fait, de nombreux effets indésirables sur le fonctionnement du moteur sont éliminés, tels que son échauffement qui est dû à l'augmentation des pertes et les pulsations de couple qui deviennent gênantes surtout aux faibles vitesses.
- On remarque aussi d'après la figure 1.15 une augmentation de la non-linéarité de la variation des angles de commutation, ce qui posera un problème critique si nous ne faisons pas un bon choix de sélection de base de données c'est ce que nous allons voir dans le chapitre 2 dans la partie apprentissage.

### Les signaux PWMs et l'analyse spectrale

La figure 1.16 illustre d'une part le bon fonctionnement du principe d'élimination d'harmoniques. D'autre part, il permet d'éclaircir les différentes

12. Efficace ie. les angles calculés par l'implémentation hardware soient très proche des valeurs exactes calculées par la méthode itérative de Newton-Raphson.

13. Le choix de la base de données d'apprentissage est une étape primordiale pour la convergence des réseaux de neurones.

fonctions des blocs exposés dans la partie structure MATLAB 1.5.4.

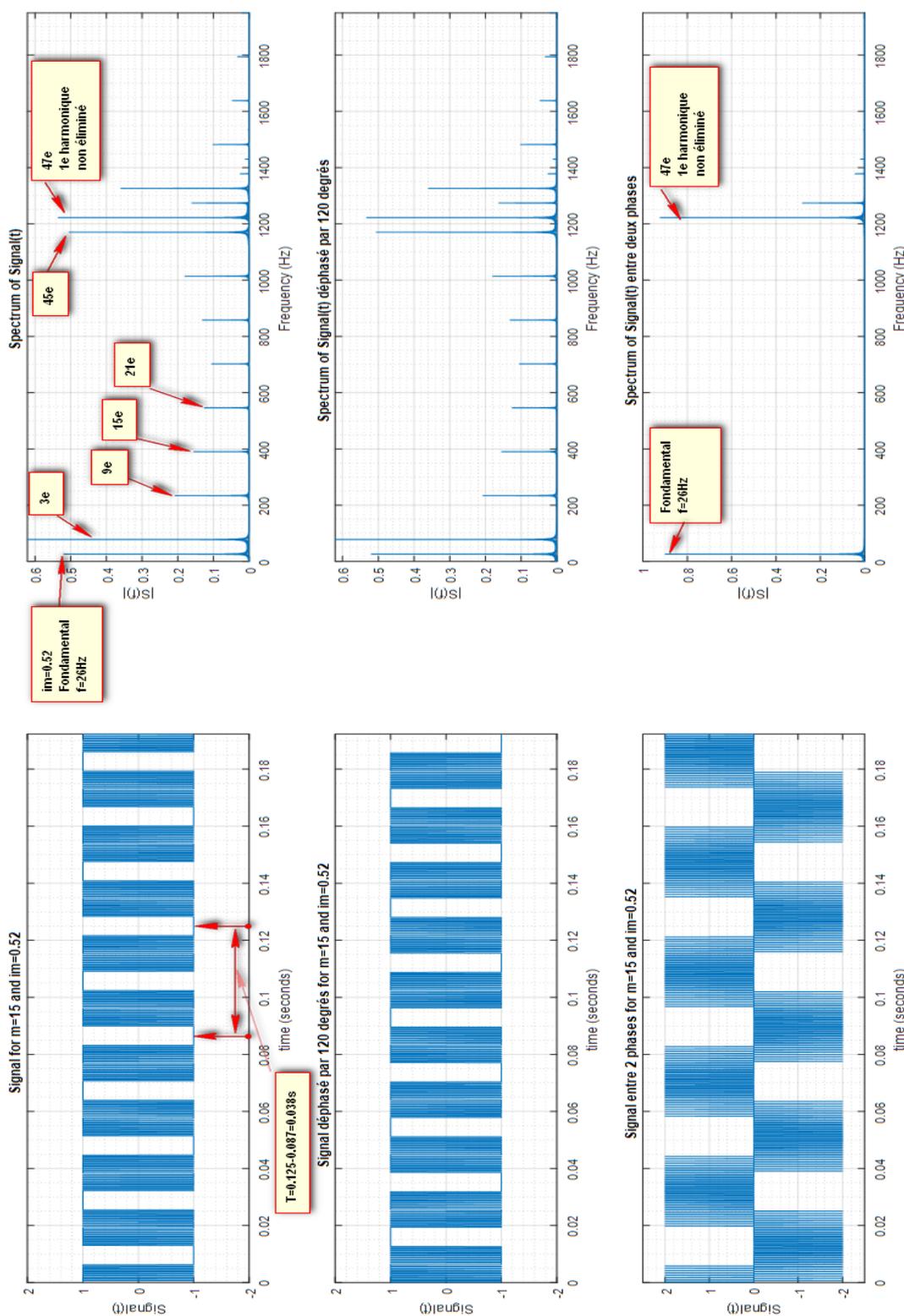


FIGURE 1.16 – Résultats : Signaux PWMs & Analyse spectrale.

### interprétation

- La figure 1.16 montre que les tensions de phase sont périodiques de fréquence de  $f = 26,32\text{Hz} \approx 26\text{Hz}$ <sup>14</sup> ( $T = 0,038\text{s} \approx 26^{-1}\text{s}$ ) pour  $im = 0,52$ .
- On constate aussi l'existence de 15 angles de commutation par quart de période pour  $im=0.52$ .
- L'analyse spectrale de la figure 1.16 montre, comme prévu, l'élimination des harmoniques multiples du fondamental dans le cas des signaux de phases : tous les multiples pairs du fondamentales par le biais de la règle 1 dans 1.4.2 et l'élimination des multiples 5e, 7e, 11e, 13e, 17e, 19e, 23e, 25e, 29e ... 47e grâce à la technique SHE PWM et dans le cas des signaux entre deux phases, en plus des multiples éliminés dans le cas de signaux de phase les harmoniques de 3 et de multiples de 3 sont aussi éliminés automatiquement grâce à l'utilisation d'un onduleur triphasé [36, 24, 18].
- De plus, la figure 1.16 montre que l'amplitude du fondamental est presque égale à  $im$ .

## 1.6 Conclusion

On dit souvent que "*La nécessité est la mère de l'invention.*", dans ce chapitre, nous avons introduit l'un des fameux problèmes qui tracasse la commande de vitesse d'une machine Asynchrone dans un EV, et nous avons proposé d'utiliser la commande SHE PWM qui semble être le candidat adéquat compte tenu de ses avantages : meilleure élimination des harmoniques indésirables, pertes de commutation moindres, la limite minimale de largeur d'impulsion peut être atteinte facilement et possibilité de la sur-modulation.

Cependant, le calcul des valeurs exactes des angles de commutation, lors de l'utilisation de cette technique exige la résolution d'un système de  $m$  équations non-linéaires à  $m$  inconnus, pour ceci nous avons proposé et démontré au cours de ce chapitre la méthode itérative de Newton-Raphson qui exige non seulement un temps de calcul considérable mais aussi une nécessité d'estimer les valeurs initiales de la solution pour assurer la convergence de cette dernière en utilisant l'algorithme de Taufik, Mellitt et Goodman.

Nous ne pouvons pas nier que le calcul des angles de commutation par cette méthode soit précis ( $erreur < 1.0e^{-15}$ ) mais ce nécessite un temps de calcul très élevé qui empêche une commande de vitesse en temps réel "*on-line*". En conséquence, la technique SHE PWM est un traitement "*off-line*". Autrement dit, l'ordinateur principal effectuant le traitement ne contrôle pas et ne lit pas immédiatement les données de ses périphériques d'entrée. Les données sont préparées et stockées sur un périphérique de stockage et sont ensuite mises à disposition si nécessaire, un tel procédé requiert une large

14. Le calcul théorique donne  $f = 26\text{Hz}$  selon l'équation (1.4) avec  $im = 0.52$  et  $f_0 = 50\text{Hz}$ .

mémoire pour stocker tous les angles calculés et atteindre une bonne précision.

Cet inconvénient a poussé les chercheurs à concevoir d'autres techniques permettant le calcul des angles de commutation de la SHE PWM en temps réel "*on-line*". Selon la même approche, dans notre prochain chapitre nous allons proposer un nouvel algorithme basé sur la théorie des Réseaux de Neurons Artificiels (Artificial Neural Network (RNA)).

Cependant il est utile de mentionner que le défi n'est pas seulement de donner une solution à la problématique citée ci-dessus, mais de donner une solution pour tout problème analogue nécessitant une utilisation hardware des réseaux de neurones pour un calcul en temps réel mais aussi de répondre à toutes les exigences possibles pouvant être relevées par les chercheurs de différents domaines afin d'obtenir l'architecture RNA la plus adéquate et la plus optimisée possible.

## Chapitre 2

# Les réseaux de neurones artificiels

### 2.1 Introduction

Aujourd'hui, l'Intelligence Artificielle, branche de l'Informatique fondamentale bouleverse les sciences, l'essor des objets connectés et l'augmentation des capacités technologiques, nombreuses sont les recherches tentant de simuler le comportement humain et également de reproduire ses capacités cognitives. Dans ce contexte le Deep Learning, littéralement "*l'apprentissage profond*", est actuellement un des axes de recherche les plus explorés. «*Je n'ai jamais vu une révolution aussi rapide. On est passé d'un système un peu obscur à un système utilisé par des millions de personnes en seulement deux ans.*» précise Yann LeCun, un des pionniers du deep learning, tout ça grâce l'implication des réseaux de neurones dans tous les domaines scientifiques qui sont désormais l'outil phare de l'intelligence artificielle (IA).

Toutes les grandes entreprises tech s'y mettent : Google, IBM, Facebook, Microsoft, Amazon, Adobe, Yandex ou encore Baidu investissent des fortunes. Comme le montre notre graphique Statista 2.1, le secteur de l'intelligence artificielle est en plein essor en Europe : la valeur de ce marché devrait presque atteindre 8 milliards de dollars à l'horizon 2025.

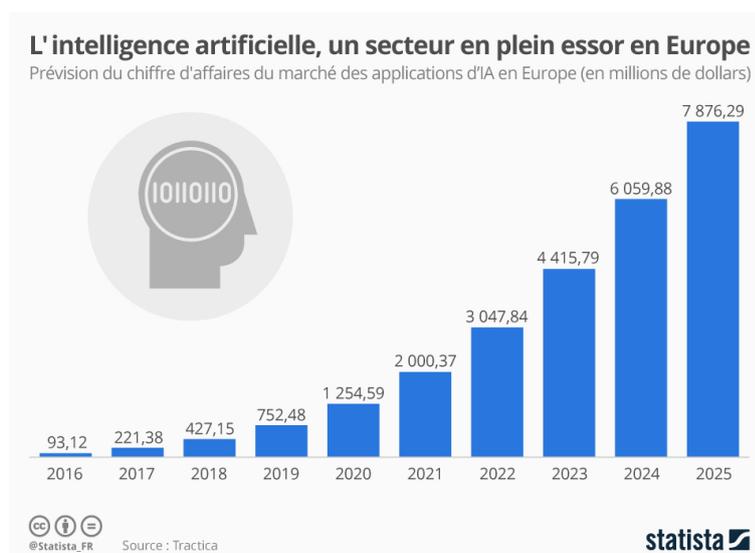


FIGURE 2.1 – Prévisions de chiffre d'affaires du marché des applications d'IA en Europe

Ce système d'apprentissage et de classification, basé sur des « réseaux de neurones artificiels » numériques, est, pêle-mêle, utilisé par Siri, Cortana et Google Now pour comprendre la voix, être capable d'apprendre à reconnaître des visages.

Dans ce chapitre, nous allons essayer comme nous en avons déjà parlé d'ouvrir cette boîte noire qui enveloppe le mystère du réseau de neurones. Commenant par un bref historique et en développant les différents concepts du neurone biologique qui représente la source d'inspiration pour les neurones artificiels, nous exposerons par la suite une analyse comparative entre ces deux types de neurones, de plus, on explique le processus d'apprentissage utilisé, le choix de la topologie où nous exposerons une solution originale pour l'opérationnel et nous terminerons avec une implémentation complète des différents blocs sur MATLAB, des interprétations et une conclusion.

## 2.2 Historique [21]

Les simulations de réseaux neuronaux semblent être un développement récent. Cependant, ce domaine a été créé avant l'avènement des ordinateurs, et a survécu à au moins un revers majeur et plusieurs époques.

Les réseaux de neurones ont été utilisés avec des ordinateurs depuis 1949 quand D.Hebb, un physiologiste américain, a publié son livre intitulé "*l'organisation du comportement*", dans lequel il a exposé certaines de ses idées sur l'apprentissage pour la première fois, la règle dite Hebb qu'il a proposé était l'une des règles d'apprentissage sur lesquelles repose la plupart des algorithmes connexionnistes d'aujourd'hui [28]. Au fil des années, de nombreuses architectures de réseaux neuronaux ont été présentées.

En 1957, F. Rosenblatt a développé le modèle du perceptron, l'un des premiers réseaux de neurones, qui tentait de comprendre la mémoire humaine, l'apprentissage et les processus cognitifs. En 1960, Rosenblatt a démontré le Mark 1 perceptron. Le Mark 1 était la première machine à pouvoir apprendre à identifier des motifs optiques. Basé sur la règle Hebb, le perceptron était alors considéré comme la première machine à pouvoir apprendre de l'expérience. Malheureusement, il était incapable d'apprendre à reconnaître les entrées qui n'étaient pas «linéairement séparables» [17]. Cela se révélerait être un énorme obstacle qui prendrait du temps à surmonter.

Un nouveau modèle neuronal a été développé par B. Widrow et T. Hoff en 1960, appelé le réseau Adaline (Adaptative Linear Element). Dans sa structure, il ressemble au perceptron, mais la règle d'apprentissage était différente. Ils ont proposé la minimisation des erreurs de sortie quadratiques en tant qu'algorithme d'apprentissage. Le réseau Adaline est considéré comme le modèle de base des réseaux multicouches [28].

En 1969, les limites théoriques du perceptron ont été démontrées par M. Minsky et S. Papert. Ces limitations concernaient l'impossibilité de traiter

des problèmes non-linéaires à l'aide de ce modèle. L'impact de leurs résultats a frustré la plupart des chercheurs dans ce domaine, en particulier les scientifiques de calcul. Cette stagnation a duré près de 20 ans. Pendant cette période, les chercheurs et les investisseurs se sont tournés vers l'approche de l'intelligence artificielle, qui semblait plus prometteuse.[16]

Cette discipline a été ramenée à la vie, en 1982, grâce à J. J. Hopfield, éminent physicien, qui a pu détecter la similitude entre les réseaux proposés par McCulloch et Pitts, avec un système élémentaire à moment magnétique ou spin, puis il a étudié le stockage et la restauration de l'information "mémoires associatives". quand il eut l'idée d'utiliser une fonction d'énergie pour maintenir la stabilité des réseaux neuronaux, avec une telle fonction, les états tendent à changer à un minimum local. Ce travail intéressait les physiciens à cause de l'isomorphisme du modèle de Hopfield avec le modèle d'Ising, il est important de noter que ce modèle n'a pas éliminé les limites du perceptron et de ses variantes, malgré cela, le perceptron et les raisons de son échec se sont vite fait oublier [16, 28].

Ensuite, la machine de Boltzmann a été proposée en 1983 par Hinton et son équipe, c'était le premier modèle qui a supprimé les limitations de perceptron de façon satisfaisante. Ce modèle utilise ce que l'on appelle des cellules cachées dont le rôle est de calculer des variables intermédiaires pour effectuer des fonctions séparables non-linéaires. Malheureusement, la convergence de l'algorithme était extrêmement longue, elle présentait un défaut dû à sa nature probabiliste, et elle a été corrigée par l'algorithme de rétropropagation de gradient proposé en 1986 par trois chercheurs, Rumelhart, Hinton et William.

Enfin, en 1989, Moody et Darken ont exploité certains résultats de l'interpolation multi-variable pour proposer ce qu'on appelle le réseau de fonctions de base radiale [28].

Récemment, les nouvelles découvertes en neurobiologie et l'intérêt explosif du traitement parallèle, en plus du développement de la technologie des semi-conducteurs, ont donné une grande impulsion au domaine des réseaux de neurones.

## 2.3 Du neurone biologique au neurone artificiel

### 2.3.1 Réseau de neurones biologiques

Les réseaux de neurones artificiels tirent une grande partie de leur inspiration du système nerveux biologique. Il est donc très utile d'avoir une certaine connaissance de la façon dont ce système est organisé.

Le cerveau est principalement composé d'environ 10 milliards de neurones, chacun relié à environ 10 000 autres neurones. Chacun des blobs<sup>1</sup> jaunes dans l'image 2.2 ci-dessous sont des corps de cellules neuronales (soma), et les lignes sont les canaux d'entrée et de sortie (dendrites et axones) qui les relient.

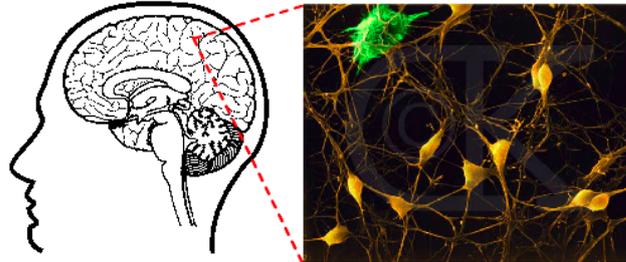


FIGURE 2.2 – Le réseau de neurones biologiques [38]

### 2.3.2 Neurone biologique

L'élément fondamental du réseau neuronal est appelé un neurone. Comme le montre la figure 2.3, un neurone se compose principalement de quatre parties : les dendrites (1), le soma (2), l'axone (3) et les synapses (4).

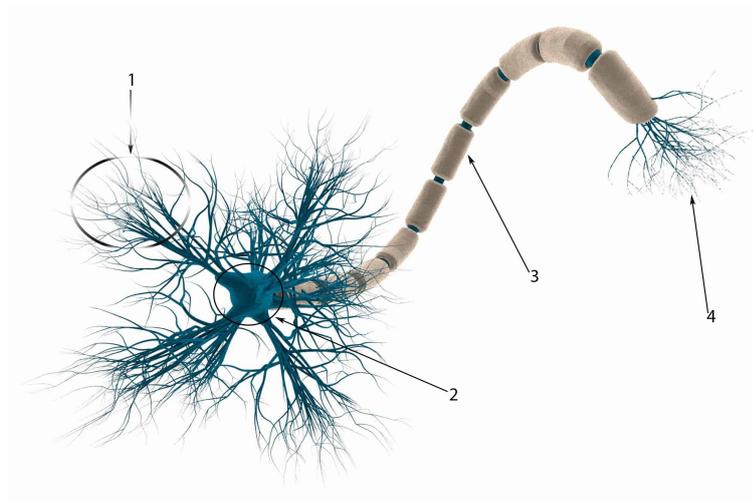


FIGURE 2.3 – Le neurone biologique

---

1. Le mot anglais blob peut désigner une goutte (d'un liquide épais), une tache ou une zone informe (ou floue).

**Définition 2.3.2.1** *Les dendrites : sont la structure arborescente qui reçoit le signal des neurones environnants, où chaque ligne est connectée à un neurone [20].*

**Définition 2.3.2.2** *Le corps cellulaire ou soma : contient le noyau cellulaire et le cytoplasme qui l'entoure et Il est de forme très variable. Il assure la synthèse des constituants nécessaires à la structure et aux fonctions du neurone et ce, pendant toute la vie de l'individu [10].*

**Définition 2.3.2.3** *L'axone : est un cylindre mince qui transmet le signal d'un neurone à d'autres. A la fin de l'axone, le contact avec les dendrites se fait par une synapse [20].*

**Définition 2.3.2.4** *La synapse : désigne une zone de contact fonctionnelle qui s'établit entre deux neurones, ou entre un neurone et une autre cellule. Elle assure la conversion d'un potentiel d'action déclenché dans le neurone présynaptique en un signal dans la cellule postsynaptique [40].*

### 2.3.3 Fonctionnement du neurone biologique

Les neurones remplissent essentiellement la fonction suivante : toutes les entrées dans la cellule, qui peuvent varier en fonction de la puissance de la connexion ou de la fréquence du signal entrant, sont additionnées. La somme d'entrée est traitée par une fonction de seuil et produit un signal de sortie. Le temps de traitement d'environ 1 ms par cycle et la vitesse de transmission des neurones d'environ 0,6 à 120ms sont comparés lentement à un ordinateur moderne [42, 19].

Le cerveau fonctionne à la fois en parallèle et en série. Cette nature du cerveau est facilement visible à partir de l'anatomie physique du système nerveux et du temps nécessaire pour effectuer des tâches. Par exemple, un humain peut reconnaître l'image d'une autre personne en environ 100 ms. Compte tenu du temps de traitement de 1 ms pour un neurone individuel, cela implique qu'un certain nombre de neurones, mais moins de 100, sont impliqués en série, tandis que la complexité de la tâche est la preuve d'un traitement parallèle, car une tâche de reconnaissance difficile ne peut pas être effectuée par un si petit nombre de neurones, exemple tiré de [42]. Ce phénomène est connu sous le nom de règle des 100 étapes.

Les systèmes neuronaux biologiques ont généralement une tolérance aux fautes très élevée. Des expériences avec des personnes ayant subi des lésions cérébrales ont montré que les dommages des neurones jusqu'à un certain niveau n'influencent pas nécessairement les performances du système, même si des tâches telles que l'écriture ou la parole peuvent devoir être réappries. Cela peut être considéré comme un recyclage du réseau.

### 2.3.4 Le réseau de neurones artificiels

Comme le réseau neuronale biologique, Le RNA est une interconnexion de nœuds, analogue aux neurones. Chaque réseau neuronal possède trois composantes essentielles : le caractère de nœud, la topologie de réseau, et l'apprentissage des règles.

- Le caractère nœud détermine la manière dont les signaux sont traités par le nœud, tel que le nombre d'entrées et de sorties associées au nœud, le poids associé à chaque entrée et sortie, et la fonction d'activation.
- La topologie de réseau détermine les moyens parmi lesquels les nœuds sont organisés et connectés.
- Les règles d'apprentissage déterminent la façon dont les poids sont initialisés et ajustés.

### 2.3.5 Le neurone artificiel et son modèle mathématique

Un neurone est une unité de traitement de l'information qui est fondamentale pour le fonctionnement d'un réseau de neurones, il contient toutes les structures d'une cellule animale. La complexité de la structure et des processus dans une cellule simple est énorme. Même les modèles de neurones les plus sophistiqués des réseaux de neurones artificiels semblent relativement semblables à des jouets. Le diagramme de la figure 2.4 montre le modèle d'un neurone, qui constitue la base de la conception d'un réseau neuronal artificiel.

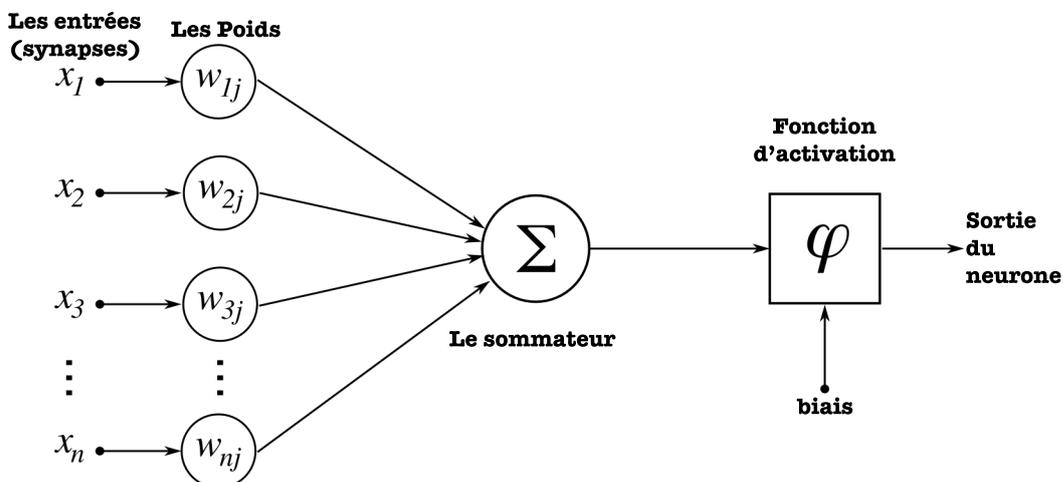


FIGURE 2.4 – Le modèle d'un neurone artificiel

Quatre éléments de base du modèle peuvent être identifiés :

1. **Un ensemble de synapses ou d'entrées**, dont chacun est caractérisé par le poids ou la force qui lui est propre. Spécifiquement, un signal  $x_i$  à l'entrée de la synapse  $i$  connectée au neurone  $j$  est multiplié par le poids synaptique  $w_{i,j}$ . Le premier indice fait référence au neurone en question et le second indice à la synapse d'entrée à laquelle le poids fait référence.

Contrairement aux vrais neurones, les poids synaptiques peuvent avoir des valeurs négatives

2. **Un sommateur** pour sommer les signaux d'entrée, pondérés par les synapses respectives du neurone, jusqu'à ce que les opérations décrites constituent un combineur linéaire.
3. **Une fonction d'activation** pour limiter l'amplitude de la sortie d'un neurone. La fonction d'activation est également appelée fonction d'écrasement, car elle écrase (limite) la plage d'amplitude admissible du signal de sortie à une certaine valeur finie. Typiquement, la plage d'amplitude normalisée de la sortie d'un neurone s'écrit comme l'intervalle fermé  $[0; 1]$  ou alternativement  $[-1; 1]$ .
4. **Un biais** est appliqué de l'extérieur et qui a pour effet d'augmenter ou de diminuer l'entrée nette de la fonction d'activation, selon qu'elle soit positive ou négative, respectivement.

Mathématiquement parlant, on peut décrire un neurone  $j$  donné par l'équation suivante :

$$Sortie_j = f_j \left( \sum_{i=1}^n (W_{i,j} \times x_i) + biais_j \right) \quad (2.1)$$

avec :

$x_i$  : l'entrée de la  $i^{eme}$  synapse .

$W_{i,j}$  : le poids synaptique correspondant au  $j^{eme}$  neurone et à la  $i^{eme}$  entrée.

$f_j$  : la fonction d'activation de  $j^{eme}$  neurone.

### 2.3.6 L'architecture d'un RNA

L'architecture d'un réseau neuronal artificiel est défini par la manière dont les différents neurones sont disposés ou placés les uns par rapport aux autres. Ces arrangements sont structurés essentiellement en dirigeant les connexions synaptiques des neurones.

En général, un réseau neuronal artificiel peut être divisé en trois parties, nommées couches, qui sont connues comme :

- (a) **Couche d'entrée** : cette couche est chargée de recevoir des informations (données), des signaux, des caractéristiques ou des mesures provenant de l'environnement externe. Ces entrées (échantillons ou modèles) sont généralement normalisées à l'intérieur des valeurs-limites produites par les fonctions d'activation. Cette normalisation entraîne une meilleure précision numérique pour les opérations mathématiques effectuées par le réseau.
- (b) **Couche cachée** (intermédiaire ou invisibles) : ces couches sont composées de neurones qui sont responsables de l'extraction de motifs associés

au processus ou au système analysé. Ces couches exécutent la majeure partie du traitement interne à partir d'un réseau.

- (c) **Couche de sortie** : cette couche est également composée de neurones, et est donc responsable de la production et de la présentation des sorties du réseau final. Les architectures principales des réseaux de neurones artificiels, en considérant la disposition des neurones, ainsi que leur interconnexion et la composition de leurs couches, peuvent être divisées comme suit : **(i)** réseau de feedforward monocouche, **(ii)** réseaux feedforward multicouches, **(iii)** les réseaux récurrents et **(iv)** les réseaux maillés.

Dans notre projet, nous avons utilisé les réseaux feedforward multicouches, nous allons alors nous limiter à présenter que cette architecture.

### 2.3.7 Les réseaux feedforward multicouches

Les réseaux de feedforward multicouches sont composés d'une ou de plusieurs couches neurales cachées figure 2.5. dans lesquels les flux de données des unités d'entrée vers les unités de sortie sont dirigés strictement en aval. Le traitement des données peut s'étendre sur plusieurs couches d'unités, mais aucune connexion de rétroaction n'est présente.

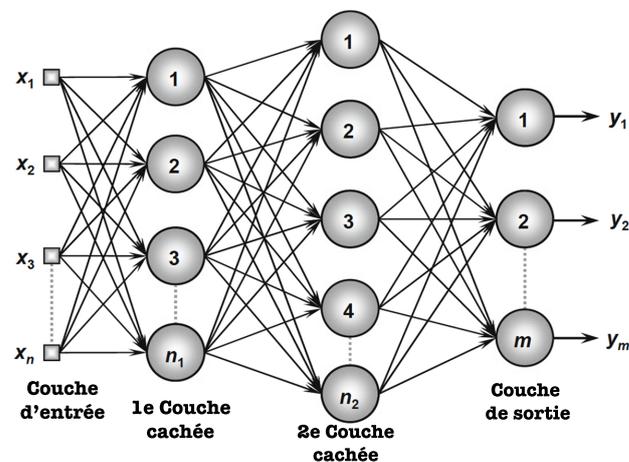


FIGURE 2.5 – Exemple d'un réseau feedforward multicouches

Ils sont employés dans la solution de divers problèmes, tels que ceux liés à l'approximation de fonctions, la classification de modèles, l'identification de systèmes, le contrôle de processus, l'optimisation, la robotique, et ainsi de suite.

La figure 2.5 montre un réseau "feedforward" multicouches, composées d'une couche d'entrée avec n signaux d'échantillons, deux couches neurales

cachées consistant en neurones  $n_1$  et  $n_2$  respectivement, et, enfin, une couche neurale de sortie composée de  $m$  neurones représentant les valeurs de sortie respectives du problème en cours d'analyse. Parmi les principaux réseaux utilisant des architectures feedforward multicouches, on trouve le Perceptron Multilayer (MLP) et la Radial Basis Function (RBF)

### 2.3.8 Fonction d'activation

Dans les réseaux de neurones artificiels et biologiques, un neurone ne sort pas seulement son entrée. Au lieu de cela, il y a une autre étape, appelée fonction d'activation, analogue au taux de potentiel d'action<sup>2</sup> tirant dans le cerveau. La fonction d'activation prend la même somme pondérée que précédemment 2.1, puis la transforme une fois de plus avant de la sortir finalement. De nombreuses fonctions d'activation ont été proposées, mais pour l'instant nous allons donner un tableau qui représente quelques fonctions d'activation telles que la fonction sigmoïde qui est historiquement, la fonction d'activation la plus ancienne et la plus populaire.

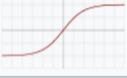
Nom	Graphe	Équation	Dérivée	Étendue	Ordre de continuité
Identité/Rampe		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	$C^\infty$
Marche/Heaviside		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	$C^{-1}$
Logistique (ou marche douce, ou sigmoïde)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	$C^\infty$
Tangente Hyperbolique (TanH)		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	$C^\infty$
Arc Tangente (ArcTan ou Tan <sup>-1</sup> )		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$(-\frac{\pi}{2}, \frac{\pi}{2})$	$C^\infty$
Signe doux <sup>6</sup>		$f(x) = \frac{x}{1 +  x }$	$f'(x) = \frac{1}{(1 +  x )^2}$	$(-1, 1)$	$C^1$
Unité de Rectification Linéaire (ReLU) <sup>7</sup>		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty[$	$C^0$

FIGURE 2.6 – Liste de fonctions d'activation usuelles

Il faut noter que dans notre implémentation hardware, nous avons implémenté trois fonctions d'activation qui sont : la fonction linéaire, Tangente-sigmoïde et Log-sigmoïde.

2. En physiologie, un potentiel d'action se produit lorsque le potentiel membranaire d'un site axonal spécifique augmente et diminue rapidement [4].

### 2.3.9 Analyse comparative.

#### Réseau de neurones biologiques RNB



Les réseaux de neurones biologiques ont les parties suivantes :

#### 1. Structure

A.Soma [2.3.2.2](#)

B.Dendrites [2.3.2.1](#)

C.Synapse [2.3.2.4](#)

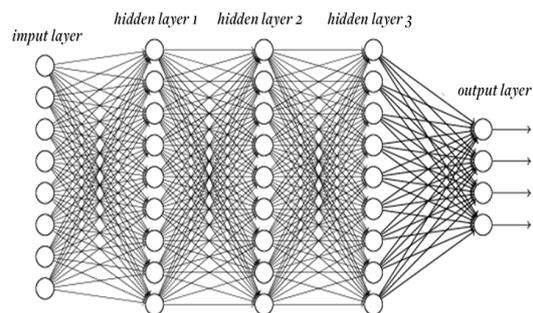
D.Axone [2.3.2.3](#)

E.Les informations provenant d'autres neurones, sous la forme d'impulsions électriques, pénètrent dans les dendrites aux points de connexion appelés synapses. L'information circule des dendrites vers le soma où elle est traitée. Le signal de sortie, un train d'impulsions, est ensuite envoyé vers le bas de l'axone à la synapse des autres neurones.

#### 2. Couches

A.Les réseaux de neurones biologiques sont construits de manière tridimensionnelle à partir de composants microscopiques. Ces neurones semblent capables d'interconnexions presque illimitées.

#### Réseau de neurones artificiel RNA



Les réseaux de neurones artificiels ont également les composants suivants qui sont équivalents aux réseaux de neurones biologiques :

#### 1. Structure

A.Nœud

B.Entrées

C.Poids

D.Sortie

E.L'arrangement et les connexions des neurones forment le réseau qui a trois types de couches. La première couche est appelée couche d'entrée, elle est la seule couche exposée aux signaux externes et qui transmet des signaux aux neurones de la couche suivante de deuxième type, appelée couche cachée. Cette dernière extrait les caractéristiques pertinentes des signaux reçus. Ces caractéristiques sont ensuite dirigées vers la couche de sortie, qui est la dernière couche du réseau [29].

#### 2. Couches

A.Dans les RNAs, c'est des simples regroupements des neurones artificiels en couches, qui sont ensuite connectées les unes aux autres. La connexion des couches peut également varier. Fondamentalement, tous les réseaux de neurones artificiels ont une structure de topologie similaire.

**3. Taille**

A.  $10^{11}$  neurones

**4. Fonctions**

A. Les dendrites fournissent des signaux d'entrée aux cellules.

B. Une synapse est capable d'augmenter ou de diminuer la force de la connexion. C'est ici que les informations sont stockées.

C. L'axone envoie un signal de sortie à une autre cellule. Les terminaux axonaux fusionnent avec les dendrites des autres cellules.

D. Le cerveau est constitué d'un grand nombre de composants (environ 10<sup>11</sup>), dont chacun est relié à de nombreux autres composants (environ 10<sup>4</sup>), dont chacun effectue un calcul relativement simple, dont la nature n'est pas claire, par des connexions lentes [29].

**5. Apprentissage**

A. Ils sont capables de tolérer l'ambiguïté, d'apprendre sur une base de données très appauvries et désorganisées (par exemple, apprendre une grammaire à partir d'échantillons aléatoires de langage naturel mal structuré et non-systématique).

B. Ils apprennent de l'expérience passée pour améliorer leurs propres niveaux de performance [29].

C. L'apprentissage dans les systèmes biologiques implique des ajustements aux connexions synaptiques qui existent entre les neurones.

**6. Méthode de calcul**

A. Les réseaux de neurones biologiques communiquent par des impulsions, la synchronisation des impulsions pour transmettre des informations et effectuer des calculs est bien gérée. (Parallèle et distribué) [29].

**3. Taille**

A.  $10^2 - 10^4$  neurones

**4. Fonctions**

A. Le neurone artificiel reçoit des intrants analogues aux impulsions électrochimiques que les dendrites reçoivent d'un autre neurone.

B. Les signaux artificiels peuvent être modifiés par des poids d'une manière similaire aux changements physiques qui se produisent dans les synapses.

C. La sortie du neurone artificiel correspond à des signaux émis par le neurone biologique sur son axone.

D. Les neurones sont connectés de manière aléatoire ou uniforme, et tous les neurones effectuent le même calcul. Chaque connexion est associée à un poids numérique. La sortie de chaque neurone est calculée comme une fonction monotone de la somme des produits des neurones d'entrée avec leur correspondant poids de connexion [29].

**5. Apprentissage**

A. Plus précisément des données et des règles formatées et structurées sont requises.

B. Ils apprennent également de l'expérience passée pour améliorer leurs propres niveaux de performance.

C. Cela est également vrai pour les réseaux de neurones artificiels.

**6. Méthode de calcul**

A. Les réseaux de neurones artificiels sont basés sur un modèle de calcul impliquant la propagation d'une variable continue d'une unité de traitement à une autre. (Série et centralisé).

**6. Éléments de traitement**

**A.** Les capacités de traitement suivent des processus hautement parallèles fonctionnant sur des représentations réparties sur de nombreux neurones.

**B.** Le réseau de neurones dans le cerveau forme un système de traitement massivement parallèle.

**C.** 10<sup>14</sup> synapses.

**7. Vitesse de traitement**

**A.** lents – Les neurones ont besoin de plusieurs millisecondes pour réagir au stimulus.

**B.** Le «temps de cycle» élémentaire des neurones (c'est-à-dire leur constante de temps) est de l'ordre d'une milliseconde. (100 Hz).

**8. Tolérance aux erreurs**

**A.** Il a le potentiel d'être tolérants aux dommages partiels. Cela signifie qu'une récupération partielle des dommages est possible si les unités de santé prennent en charge les fonctions précédemment exercées par les zones endommagées.

**8. Connexion**

**A.** Un nombre très élevé de connexions entre les neurones.

**B.** Le tissu neural implique une connectivité aléatoire sans direction privilégiée.

**9. Force**

**A.** Plusieurs facteurs tel que les neurotransmetteurs qui influent la force des synapses.

**10. Stockage d'informations**

**A.** Les informations sont stockées dans les synapses.

**6. Éléments de traitement**

**A.** Les capacités de traitement capturent un calcul hautement parallèle basé sur des représentations distribuées.

**B.** Le modèle émule un réseau neuronale biologique.

**C.** 10<sup>8</sup> transistors.

**7. Vitesse de traitement**

**A.** Rapide – cela peut atteindre des temps de commutation de quelques nanosecondes.

**B.** Les temps de propagation des portes en silicium sont de l'ordre de nanoseconde, c'est-à-dire un million de fois plus rapide.

**8. Tolérance aux erreurs**

**A.** Il a le potentiel d'être tolérant aux erreurs, ou des performances robustes, toutefois ses performances se dégradent progressivement dans des conditions de fonctionnement défavorables.

**8. Connexion**

**A.** Moins de connexions mais souvent une topologie entièrement connectée.

**B.** La connectivité est spécifiée avec précision.

**9. Force**

**A.** La force des synapses déterminée uniquement par les poids.

**10. Stockage d'informations**

**A.** Les informations sont stockées dans la matrice des poids.

**11. Transmission d'informations**

A. Les neurones transmettent des informations à l'aide de signaux électriques.

**7. Support de communication**

A. Le support de communication est le tissu neural ce qui implique des transmissions stochastiques.

**8. Transduction du signal**

A. Transduction de signal codée par impulsions (modulation de fréquence).

B. Transduction retardée.

C. Le tissu neural est asynchrone et il n'y a pas d'horloge de synchronisation

**11. Transmission d'informations**

A. Transmet également des informations en utilisant des signaux électriques.

**7. Support de communication**

A. Il est structuré formellement ce qui le rend déterministe.

**8. Transduction du signal**

A. Signal à valeurs numériques (modulation d'amplitude).

B. Les facteurs temporels de la transduction du signal sont négligeables..

C. Il peut avoir une horloge de synchronisation et un comportement asynchrone.

**Similitudes**

- Les RNBs traitent l'information en parallèle, ceci est également vrai pour les RNAs.
- L'apprentissage dans RNBs passe par des expériences antérieures qui améliorent leur niveau de performance, ce qui est également vrai pour les RNAs.
- L'apprentissage dans les RNBs implique un ajustement des connexions synaptiques, en RNAs également par l'ajustement des poids.
- La transmission de l'information dans les RNBs et les RNAs utilise des signaux électriques.
- Le stockage de l'information dans les RNBs se fait aux synapses, RNAs il se fait dans la matrice des poids.

**Différences**

- Le traitement de l'information dans les RNBs est généralement lent. Dans les RNAs, le traitement de l'information est très rapide.
- Le cerveau est capable de résoudre des problèmes qu'aucun ordinateur numérique ne peut gérer efficacement.
- Les RNBs sont construits de manière tridimensionnelle à partir de composants microscopiques capable d'une interconnexion presque illimitée. Les RNAs sont un simple regroupement de la primitive neuronale avec une interconnexion limitée.

- La connectivité dans les RNBs est aléatoire alors que dans les RNAs, la connectivité est précisément spécifiée.
- Les performances ont tendance à se dégrader fortement sous les dommages partiels dans les RNAs alors que les RNBs ont le potentiel d'être tolérants aux dommages [29].

## 2.4 Processus d'apprentissage

### 2.4.1 Type d'apprentissage

Le processus le plus important dans la réalisation d'un réseau de neurones est l'étape d'apprentissage, le terme d'apprentissage fait référence à de nombreux concepts selon différents points de vue, et il est difficile de s'entendre sur une définition précise du terme. Dans les réseaux de neurones, l'apprentissage signifie un ajustement des biais et des poids de connexion.

nous allons donner dans le diagramme 2.7 suivant une vue globale des différents types d'apprentissages qui existent et nous allons situer sur le diagramme le type que nous avons choisis pour notre étude, voir (les flèches en rouge).

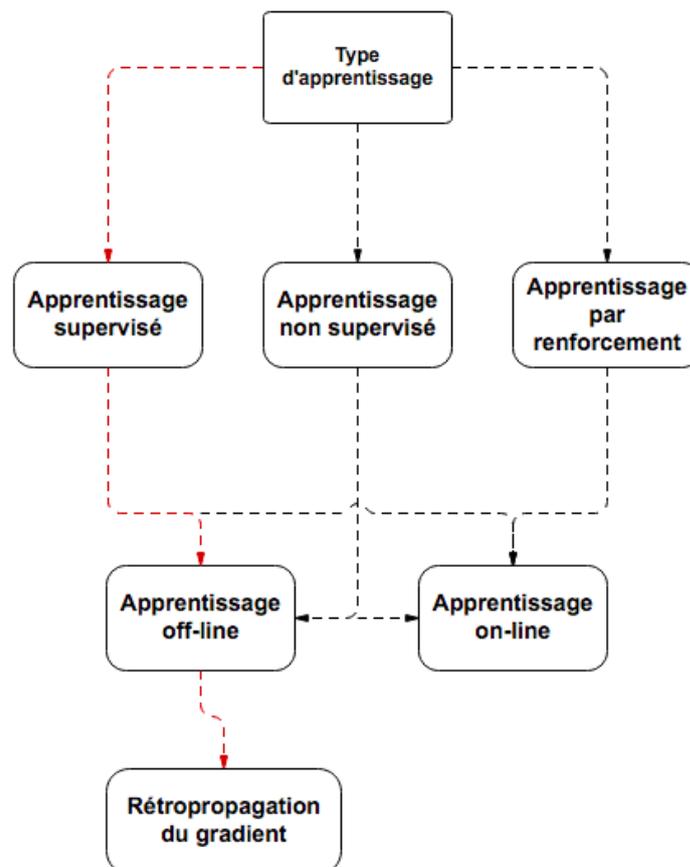


FIGURE 2.7 – Les types d'apprentissages

Dans cette section, nous allons nous limiter à exposer uniquement ce que nous avons utilisé comme méthodes.

### 2.4.2 Apprentissage off-line

Dans l'apprentissage hors ligne, également appelé apprentissage par lots, les ajustements sur les vecteurs de poids et les seuils du réseau sont effectués après que tout l'ensemble d'apprentissage soit présent. Par conséquent, tous les échantillons d'apprentissage doivent être disponibles pendant tout le processus d'apprentissage ce qui est en notre faveur car notre database est présente.

La phase d'apprentissage se fait une seule fois off-line avant l'implémentation, dans notre cas nous avons utilisé en *commande line*<sup>3</sup> l'outil *nstart tool* de MATLAB afin de réaliser l'apprentissage, l'organigramme 2.8 qui va être présenté par la suite résumera les différentes étapes d'apprentissage.

### 2.4.3 Apprentissage supervisé

Expliquons tout d'abord le choix de ce type d'apprentissage, comme on a vu dans la section ( Génération de la base de donnée 1.5, chapitre 1) toute la base de données nécessaire a été générée et avec des résultats impeccables<sup>4</sup>, donc notre choix pour ce type d'apprentissage est évident ne serai-ce que puisqu'il est le plus utilisé et le plus approprié à notre application.

La stratégie d'apprentissage supervisé consiste à disposer des sorties désirées pour un ensemble donné de signaux d'entrée, en d'autres termes, chaque échantillon d'apprentissage est composé des signaux d'entrée et de leurs sorties correspondantes. Désormais, il nécessite une table avec des données d'entrée/sortie, également appelée table attribut/valeur, qui représente le processus et son comportement (voir un exemple de table présentée dans la section "Résultats et interprétations" 1.5.5, chapitre 1). C'est à partir de cette information que les structures neurales (poids et biais) vont être construites.

Dans ce cas, l'application de l'apprentissage supervisé dépend uniquement de la disponibilité de ce tableau attribut/valeur, et il se comporte comme si un «coach» enseignait au réseau quelle est la réponse correcte pour chaque échantillon présenté pour son entrée.

Les poids et seuils synaptiques du réseau sont continuellement ajustés par l'application d'actions comparatives, exécutées par l'algorithme d'apprentissage lui-même, qui supervise l'écart entre les sorties produites par rapport aux sorties désirées, en utilisant cette différence sur la procédure d'ajustement. Le réseau est considéré comme "formé" lorsque cette divergence se situe dans une plage de valeurs acceptables, en tenant compte des objectifs

3. Deux méthodes peuvent être utilisées : méthode graphique qui n'est pas optimisé car on doit la reconfigurer manuellement pour chaque apprentissage, méthode commande line en utilisant des lignes de codes avec une génération automatique.

4. Impeccable dans ce contexte veut dire que les données sont de grande précision (erreur  $\leq 1.0^{-15}$ ) et le résultat spectrale est celui qui a été souhaité (élimination de toutes les harmoniques désirées).

de généralisation des solutions.

#### 2.4.4 Algorithme d'apprentissage "rétropropagation du gradient" du Réseau multicouches MLP [2]

Dans ce mémoire nous avons conçu un réseau MLP à partir de la base de données d'apprentissage *Database*. Chaque élément de cette base de données est appelé exemple d'apprentissage et se présente sous la forme d'un couple  $(x, y^*)$  où  $x$  est une valeur d'entrée du réseau et  $y^*$  la valeur désirée correspondante pour les sorties. L'architecture du réseau, la structure de ses connexions, ainsi que les fonctions d'activation, peuvent être fixées en fonction de la tâche que doit remplir le réseau et sous certaines conditions.

Le but de l'apprentissage est donc de déterminer les valeurs  $w^*$  de la matrice  $W$  des poids des connexions du réseau de telle sorte que les sorties ( $y$ ) soient proches des valeurs désirées  $y^*$ . Il existe plusieurs méthodes d'apprentissage, l'algorithme de la rétropropagation du gradient est très connu et le plus utilisé dans les applications des réseaux de neurones de type MLP. Cette technique d'apprentissage supervisé utilise une procédure de descente du gradient, travaillant sur l'erreur quadratique entre la sortie du réseau et la sortie désirée. Elle calcule les dérivées partielles de l'erreur en sortie par rapport à tous les poids du réseau puis applique une procédure de gradient pour minimiser l'erreur. À chaque itération, on retire un exemple d'apprentissage  $(x, y^*)$  et on calcule une nouvelle estimation des poids  $W(t)$ . Cette itération est réalisée en deux phases :

##### (a) Propagation

À chaque itération, un élément de l'ensemble d'apprentissage est introduit à travers la couche d'entrée. L'évaluation des sorties du réseau se fait couche par couche, de l'entrée du réseau vers sa sortie.

##### (b) Rétro propagation

Cette étape est similaire à la précédente. Cependant, le calcul s'effectue dans le sens inverse. À la sortie du réseau, on forme le critère de performance en fonction de la sortie réelle du système et sa valeur désirée. Puis, on évalue le gradient de cette performance par rapport aux différents poids en commençant par la couche de sortie et en remontant vers la couche d'entrée.

## 2.5 Choix de topologie et l'implémentation du processus d'apprentissage sur MATLAB

L'élaboration de la topologie du réseau neuronal pour un problème est un «art noir», dans certains problèmes scientifiques complexes, il pourrait y avoir des années d'expérience et d'intuition afin de réaliser une topologie fonctionnelle. Personnellement, je trouve que ceci représente le plus grand point faible des réseaux de neurones, cependant nous allons donner une solution qui ne sera pas définitive, mais qui va résoudre pleins de problèmes et

qui donnera naissance à tout un axe de recherche et de développement.

Pour définir l'architecture du réseau de neurones, il faut déterminer tout d'abord le nombre de couches cachées du réseau ainsi que le nombre de neurones sur chacune d'entre elles. Tout cela relève de l'intuition de l'opérationnel et de sa capacité à expérimenter plusieurs structures dans le but de retenir celle qui fournira les meilleurs résultats. Plus le réseau est complexe, plus il possède de couches cachées et de connexions, plus il est capable de reconnaître ce qui lui est présenté à travers l'échantillon d'apprentissage.

Il faut tout de même noter que la complexité croissante du réseau n'améliore pas forcément la reconnaissance sur l'échantillon test, et que le choix le plus optimal de point de vue temps d'exécution et ressource d'implémentation est d'avoir le moins de couches possibles, le moins de neurones possibles et le moins de non-linéarités<sup>5</sup> possibles.

### 2.5.1 Solution originale

La solution qu'on va proposer est une méthode indispensable pour l'opérationnel, les réseaux de neurones ne fournissent pas les explications concernant leurs résultats, ce qui limite donc l'analyse des phénomènes existants. Cette limite est due à l'opacité des réseaux de neurones qui empêche une analyse pertinente des solutions obtenues, ce que nous pouvons faire c'est plutôt analyser la source qui est l'origine de tous les problèmes notamment la divergence de la phase d'apprentissage, c'est la base de donnée qui doit être analysé et bien choisie comme on l'a mentionner dans la section (Trajectoires des angles de commutation 1.5.5, chapitre 1).

Cette analyse se fait en plusieurs étapes :

- (a) Le bon choix de la base de données de départ est primordiale, c'est lui qui va construire les différents paramètres du réseau et qui va engendrer la toute première source d'erreur (si la base n'est pas correcte) qui va être accumulée durant toute la suite de processus (Limite de nombre de bits, erreurs de quantifications, approximations ...Etc)
- (b) Diviser la base de données en plusieurs ensembles d'échantillons pour assurer la convergence dans le calcul des paramètres des réseaux de neurones artificiels, et pour augmenter aussi la précision et l'efficacité de l'algorithme, ces derniers doivent respecter un critère clé qui est la linéarité de chaque exemple d'apprentissage  $(x, y^*)$  situé dans le même ensemble d'échantillons, cela veut dire que les sorties  $y^*$  de même ensemble d'échantillons ne doivent pas avoir des changements brusques. Un exemple très descriptif est donné dans le chapitre 1 figure 1.15 où toutes les sorties de même ensemble d'échantillons représentent une linéarité sauf le dernier intervalle de  $im$  qui ne représente pas une très

---

5. Le choix de la non-linéarité est dû au choix de la fonction d'activation de chaque neurone à part, on trouve des fonctions linéaires et des fonction d'activation non-linéaires. Voir la section ??

bonne linéarité ceci va engendrer une augmentation de l'erreurs maximale des angles de sortie du réseau de neurone qu'on expliquera par la suite.

- (c) Pour chaque ensemble d'échantillons on crée un réseau de neurones associé, qui contiendra bien sûr des biais, des poids et une topologie propre à cette base de données. Notre application finale que nous allons exposer dans le chapitre 3 nous donnera le choix de créer l'ensemble de ces réseaux de neurones simultanément avec deux types de fonctionnement série et parallèle<sup>6</sup> d'une façon intelligente.
- (d) Dans la majorité des problèmes, notamment dans ce cas où on a un seul problème divisé en plusieurs sous-problèmes, le fonctionnement qui sera choisi est le fonctionnement série dans lequel l'ensemble des réseaux de neurones fonctionneront dans un même environnement avec interaction et dépendance, dans un tel environnement les RNAs ne fonctionnent pas tous simultanément, on ne trouve que quelques-uns qui fonctionnent parallèlement (en même temps) les autres vont être gérés dans un fonctionnement séquentiel synchronisé. Dans notre partie applicative, on a créé 6 réseaux de neurones fonctionnant tous en série encapsuler dans un seul RNA, car pour une entrée (*im*) donnée on a besoin d'activer qu'un seul RNA pour calculer les angles de sortie. Il faut noter que les performances de cet unique RNA équivalent les 6 réseaux de neurones implémentés tous en même temps et avec quasiment les mêmes ressources hardware que demande un seul d'entre eux. Ceci paraît magique, mais on va l'expliquer en détail dans le chapitre 3

Toutes ces étapes sont bien décrites dans l'organigramme 2.8 présenté dans la page suivante.

---

6. Ces deux types de fonctionnement sont expliqués dans le chapitre 3

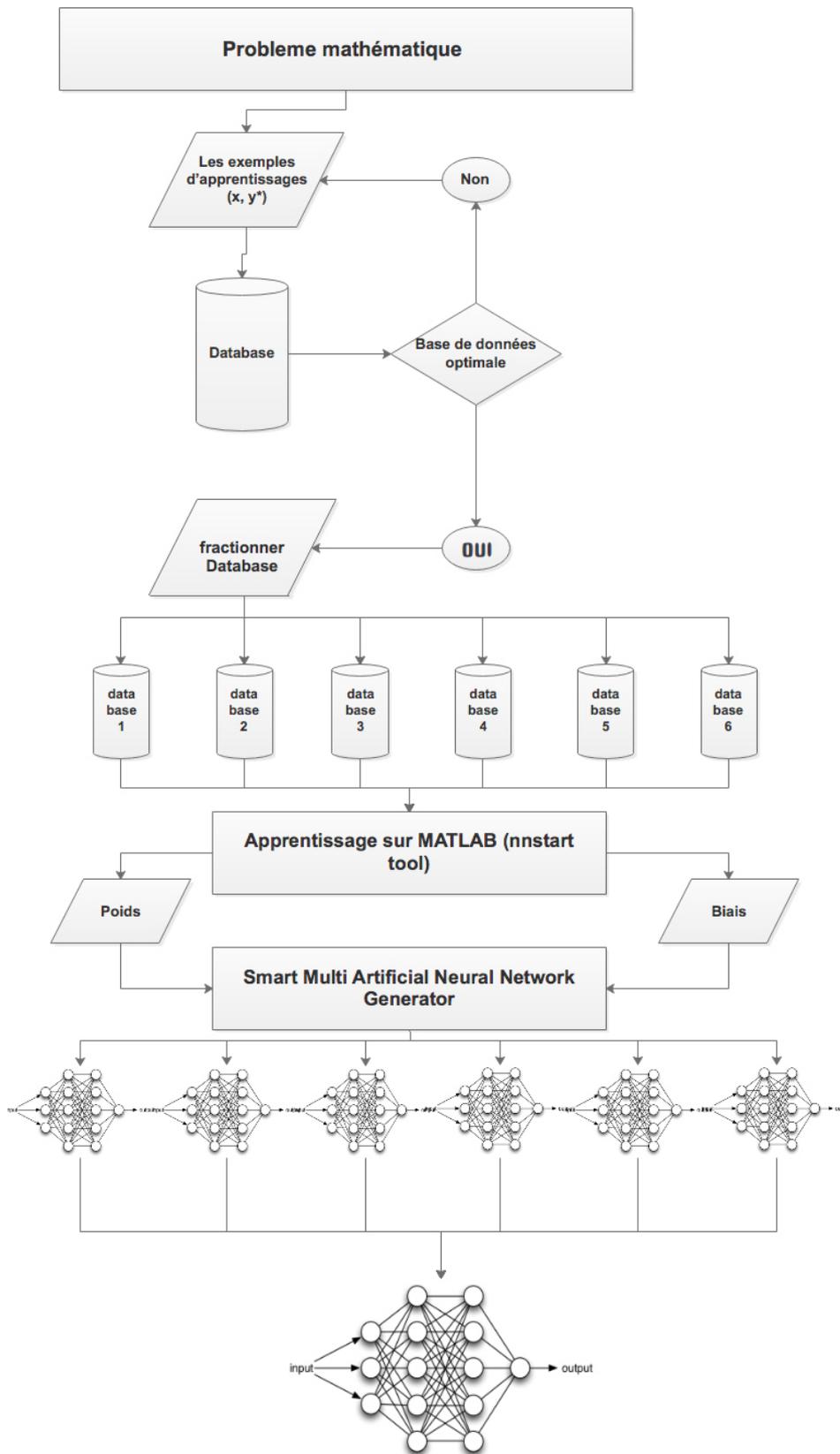


FIGURE 2.8 – Étapes d’analyse de la base de données et d’implémentation

### 2.5.2 Topologie

Nous avons appliqué la solution que nous avons proposée ci-dessus pour des raisons qui ont été expliquées, la base de données a été divisée en 6 intervalles selon la valeur de  $im$ , pour comprendre l'intérêt de point de vue applicatif voir le chapitre 1 de l'application SHE PWM section 1.5.5.

1. Pour les six RNAs on a choisi une seule couche cachée avec un seul neurone contenant une fonction d'activation non-linéaire qui est la tangente-hyperbolique.
2. La couche d'entrée possède une seule entrée qui est l'indice de modulation  $im$  pour chaque réseau de neurones.
3. La couche de sortie de chacun des 6 RNAs possède un nombre de neurones différents (3,5,7,15,19 et 23) avec une fonction d'activation linéaire, voir la figure 2.9.

Dans la littérature, il a été rapporté que le réseau à trois couches avec fonction d'activation sigmoïde dans la couche cachée et la fonction d'activation linéaire dans la couche de sortie peut approximer toute fonction non-linéaire à un degré de précision pourvu qu'un nombre suffisant de neurones soit disponible [21]. Cependant, pour réaliser tous les types de non-linéarité en utilisant trois couches, un grand nombre de neurones est nécessaire et il peut en résulter un RNA énorme.

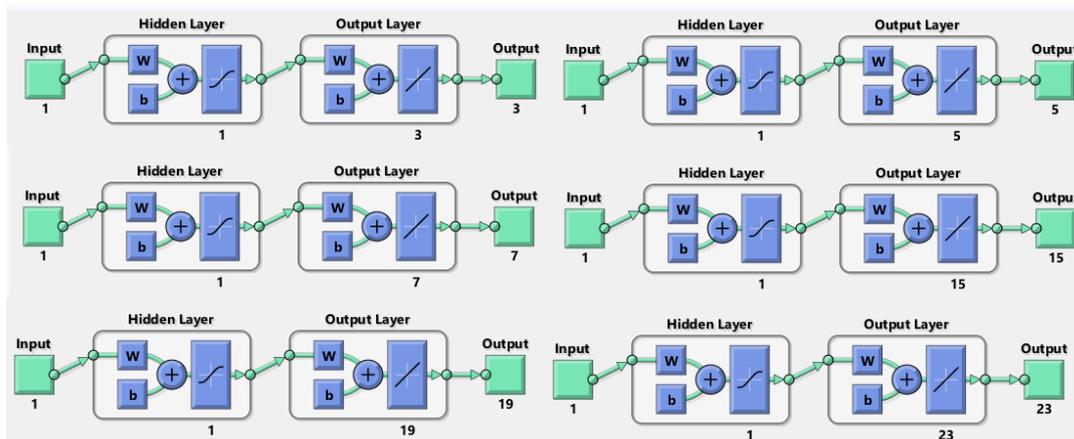


FIGURE 2.9 – Topologies des 6 réseaux de neurones artificiels

La justification des choix présentés ci-dessus est simple, le but est d'avoir l'architecture la plus optimale de point de vue consommation en ressources hardware, temps d'exécution et précision, ceci dit qu'il faut choisir le moins de couches possibles, le moins de neurones possibles et le moins de linéarités possibles afin d'assurer la précision désirée, pour notre choix de précision, on a opté pour des erreurs de l'ordre de centième de degrés. Voir le tableau des erreurs à la sortie des RNAs 2.1.

### 2.5.3 Apprentissage des six RNAs

Pour l'apprentissage des six réseaux de neurones, on a créé un programme MATLAB qui fait le *learnig* d'une façon automatique des 6 RNAs selon le diagramme suivant :

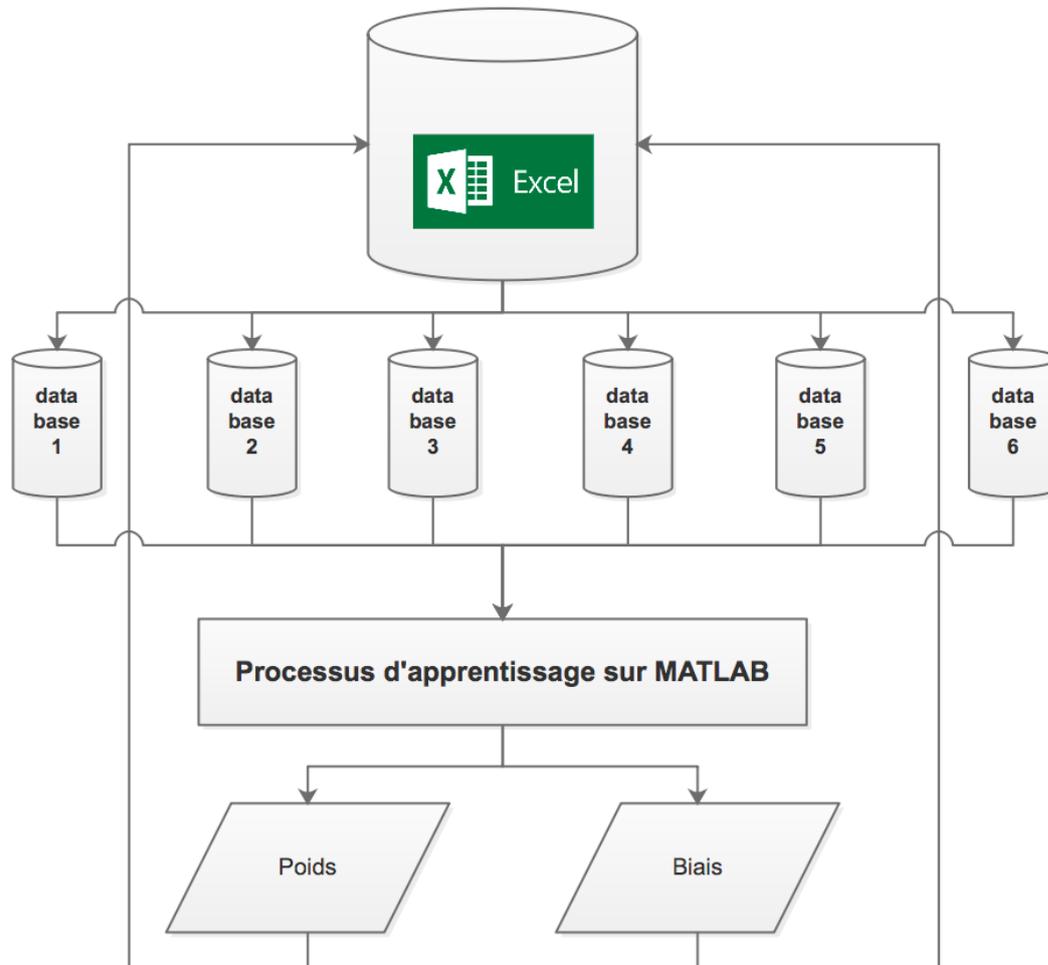


FIGURE 2.10 – Apprentissage des 6 réseaux de neurones artificiels

Pour chaque RNA, on a utilisé en moyenne 100 couple d'entrée( $1 \times 1$ ) et sortie( $1 \times m$  avec  $m=3,5,7,15,19$  et  $23$ ) avec les caractéristiques suivantes :

(a) **Pour l'entraînement 95%**

Ces éléments sont présentés au réseau pendant la formation, et le réseau est ajusté en fonction de son erreur.

(b) **Pour la validation 05%**

Ces éléments sont utilisés pour mesurer la généralisation du réseau, et pour arrêter la formation lorsque la généralisation cesse d'améliorer.

(c) **Pour le teste 05%**

Ces éléments n'ont aucun effet sur la formation et fournissent ainsi une mesure indépendante de la performance du réseau pendant et après la formation.

- (d) Nombre d'itération=100000.
- (e) Erreur d'entraînement ciblée =  $1e^{-5}$ .
- (f) L'entrée est normalisée entre [0; 1].
- (g) La sortie est normalisée entre [0; 1].

A la fin de cet apprentissage on obtient pour chaque RNA deux tables de données une pour les poids et une pour les biais, voir le diagramme 2.10.

## 2.6 Implémentation complète de la solution sur MATLAB

Il est important de noter qu'en créant une classe net<sup>7</sup> toute l'implémentation qu'on va réaliser sera déjà implémentée sur MATLAB, le but d'implémenter tout les bloc de l'architecture RNA est d'avoir déjà une idée préalable sur toutes les étapes d'implémentation hardware, avoir une idée globale sur l'architecture RNA et de pouvoir simuler les sorties des différents blocs afin d'avoir une base de données de référence qui permettra de comparer ces sorties avec les sorties des blocs hardware qui seront implémentés.

### 2.6.1 Bloc de normalisation

Durant la phase d'apprentissage, on a normalisé toutes les données de la base d'apprentissage, afin qu'elles soient actives, en moyenne sur la partie linéaire de la fonction d'activation, voir la caractéristique (f). Par conséquent le but de ce bloc est de calculer la valeur normalisée de im qui sera l'entrée du réseau RNA-i. La fonction de normalisation est donnée par l'équation (2.3).

$$output = \frac{OUTPUT_{max} - OUTPUT_{min}}{INPUT_{max} - INPUT_{min}}(input - INPUT_{max}) + OUTPUT_{min} \quad (2.2)$$

Dans l'étape d'apprentissage, on a utilisé  $OUTPUT_{max} = 1$  et  $OUTPUT_{min} = 0$ , en remplaçant dans l'équation 2.3 on trouve

$$output = \frac{1}{INPUT_{max} - INPUT_{min}}(input - INPUT_{max}) \quad (2.3)$$

### 2.6.2 Implémentation de la couche cachée et la couche de sortie

On utilise l'algorithme suivant :

1. Après l'étape de normalisation, chaque entrée est multipliée par son poids et additionnée à son biais correspondant.
2. La couche cachée est représentée par une fonction tangente-hyperbolique déjà implémentée sur MATLAB.
3. La couche de sortie est constituée de plusieurs neurones linéaires, l'entrée de chaque neurone est la sortie de neurone de la couche cachée multipliée par son poids et additionnée à son biais correspondant.

Ce processus est répété 6 fois automatiquement.

<sup>7</sup>. C'est une commande MATLAB qui permet de créer une classe d'un réseau de neurones contenant toutes ses caractéristiques et toutes ses fonctions.

### 2.6.3 Bloc de dé-normalisation

Durant la phase d'apprentissage, on a normalisé toutes les sortie du RNA, voir la caractéristique (g). Par conséquent le but de ce bloc est de faire la dé-normalisation de ces sorties afin de les obtenir sous forme d'angles. La fonction de dé-normalisation est donnée par l'équation (2.4).

$$angle = \frac{ANGLE_{max} - ANGLE_{min}}{SORTIE_{max} - SORTIE_{min}}(sortie - SORTIE_{min}) + ANGLE_{min} \quad (2.4)$$

$ANGLE_{max}$  et  $ANGLE_{min}$  sont les valeurs max et min respectivement de chaque angle de la base de données.

On a ajouté à la sortie finale un générateur du bruit afin de donner un aspect réel aux simulations.

## 2.7 Génération des sorties des 6 RNAs

En utilisant l'implémentation expliquée dans la section précédente, le bloc générateur PWM et le bloc Analyseur de spectre vus dans la section 1.5.4 chapitre 1 on génère une nouvelle base de données qui sera comparée à notre database de départ, les signaux PWMs et l'analyse spectrale.

L'organigramme 2.11 suivant résumera tout le processus.

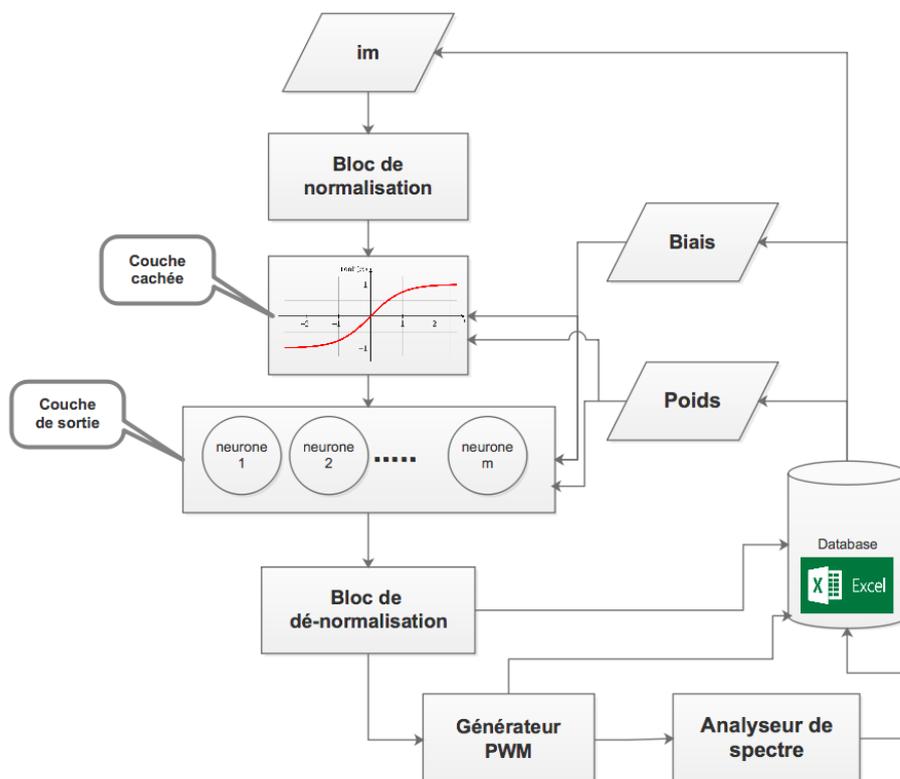


FIGURE 2.11 – Processus d'implémentation et de génération des sorties d'un RNA

### 2.7.1 Résultats

Le tableau 2.1 ci-dessous donnera une idée globale sur les performances de notre implémentation.

TABLE 2.1 – L’erreur moyenne et maximale entre les angles de commutation PWM exacts et ceux calculés par l’algorithme MATLAB

RNA-i	Nbre de neurones couche cachée	Nbre de neurones couche de sortie	L’algorithme d’apprentissage	Les fonctions d’activation	Nbre d’éléments dans database	Erreur maximale	Erreur moyenne
RNA-1	1	3	Back-propagation	Hyperbolique	03 × 081	0,0792	0,0365
RNA-2	1	5	Algorithme	Tangent-Sigm- oïde+	05 × 160	0,0649	0,0288
RNA-3	1	7			07 × 200	0,0366	0,0142
RNA-4	1	15			15 × 240	0,0168	0,0065
RNA-5	1	19			19 × 160	0,0122	0,0052
RNA-6	1	23		linéaire	23 × 159	0,0128	0,0051

La figure 2.12 présentée dans la page suivante illustre d’une part le bon fonctionnement de notre implémentation, d’autre part, il permet de prouver que le principe d’élimination d’harmoniques à la sortie des RNAs est fonctionnel malgré les erreurs que nous avons tolérées.

### 2.7.2 interprétations

- Comme prévu l’erreur maximale est de l’ordre de centième de degrés voir le tableau d’erreur 2.1
- Comme nous avons prédit dans les étapes d’analyse (b) l’erreur dans le dernier intervalle de  $im$  qui correspond au RNA-1 est la plus grande des erreurs à cause de la non-linéarité, cependant si on n’a pas suivi la solution que nous avons proposée, on n’aura jamais de convergence en utilisant une topologie aussi simple (sans diviser la base de données voir 2.5.1). Pour avoir une meilleure précision dans cet intervalle, il suffit de rendre la topologie plus complexe ou de diviser cet intervalle en d’autres sous intervalles (et donc d’autres sous RNAs). Dans notre application, cet ordre d’erreur est très acceptable.
- La figure 2.12 montre que les tensions de phase sont périodique de fréquence de  $f = 26,32\text{Hz} \approx 26\text{Hz}$ <sup>8</sup> ( $T = 0,038\text{s} \approx 26^{-1}\text{s}$ ) pour  $im = 0,52$ .
- On constate aussi l’existence des 15 angles de commutation par quart de période pour  $im=0.52$ .
- L’analyse spectrale de la figure 2.12 montre, comme prévu, l’élimination des harmoniques multiples de fondamentales dans le cas des signaux de phases : tous les multiples pairs de fondamentales par le biais de la règle 1 dans 1.4.2 et l’élimination des multiples 5e, 7e, 11e, 13e, 17e, 19e, 23e, 25e, 29e ... 47e grâce à la technique SHE PWM et dans le cas des signaux entre deux phases, en plus des multiples éliminés dans le cas

8. Le calcul théorique donne  $f = 26\text{Hz}$  en selon de l’équation (1.4) avec  $im = 0.52$  et  $f_0 = 50\text{Hz}$ .

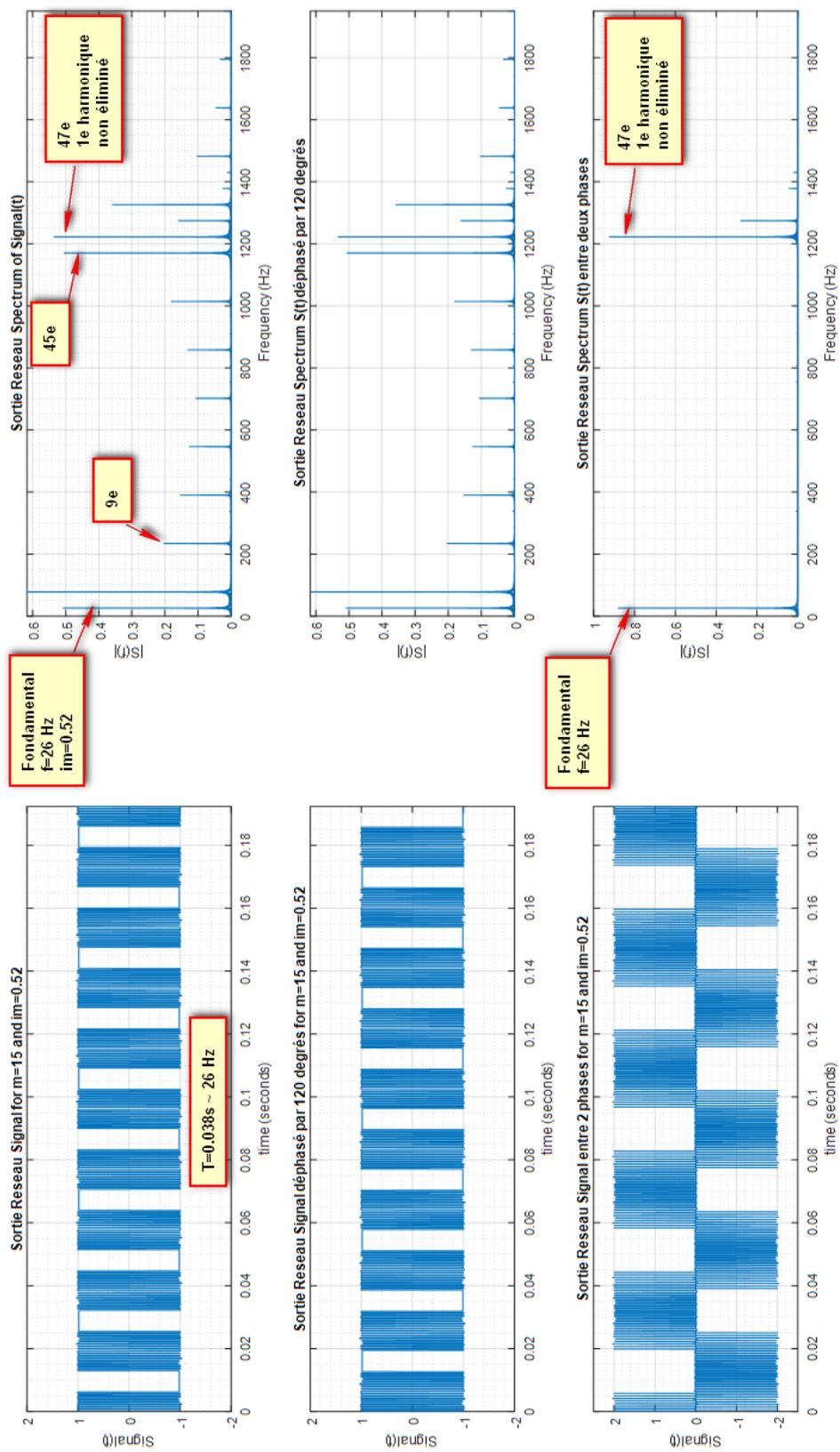


FIGURE 2.12 – Résultats : Signaux PWMs & Analyse spectrale à la sortie du RNA-4

de signaux de phase les harmoniques de 3 et de multiples de 3 sont aussi éliminés automatiquement grâce à l'utilisation d'un onduleur triphasé [36, 24, 18].

- De plus, la figure 2.12 montre que l'amplitude du fondamental est presque égale à  $im$ .

## 2.8 Conclusion

En conclusion, dans ce chapitre on a introduit tous les principes de bases qui vont nous permettre de faire une implémentation hardware des RNAs, on a exposé toutes les caractéristiques des réseaux de neurones artificiels inspirés du neurone biologique (parallélisme, non-linéarité et apprentissage) qui leur ont permis d'effectuer efficacement de nombreuses tâches qu'un ordinateur numérique conventionnel ne pourra jamais faire. On a proposé une solution originale qui ne sera pas définitive mais qui va résoudre pleins de problèmes scientifiques et qui donnera naissance à tout un axe de recherche et de développement et qui permettra enfin de maîtriser le mystère des RNAs.

À présent, on a bien ouvert la boîte noire RNA d'une façon théorique, et nous avons maîtrisé son implémentation sur MATLAB ce qu'il nous permet de faire une simulation réelle, cependant la question qui se pose est comment réaliser tout ce que nous avons présenté précédemment sur le hardware et encore plus, comment généraliser tous les blocs sous forme d'IP cores pour n'importe quelle configuration choisie par l'utilisateur, la réponse à ces questions sera présentée dans notre prochain chapitre.

## Chapitre 3

# Générateur intelligent de multi-IP cores

### 3.1 Introduction

Avant de commencer ce chapitre, nous jugeons qu'il est indispensable de clarifier l'utilisation du titre "Générateur intelligent de multi-IP cores" ou encore "Smart Multi Artificial Neural Network Generator", en ce qui concerne le terme "IP cores" nous l'avons déjà défini dans l'introduction générale, les modules que nous allons implémenter sont des IP cores dans tous les sens du mot, car ils sont dotés de toutes les caractéristiques de ce dernier (bloc de logique ou de données utilisé dans la fabrication d'un FPGA<sup>1</sup> ou d'un circuit intégré spécifique à l'application (ASIC)<sup>2</sup>, entièrement portable - c'est-à-dire, capable d'être facilement inséré dans n'importe quelle technologie de fournisseur ou méthodologie de conception sans oublier le plus important, ils sont configurables), le mot intelligent est utilisé, car l'application génératrice est capable de prendre des décisions selon la configuration et le mode de fonctionnement choisis par l'utilisateur, c'est ce que nous allons expliquer dans la suite de ce chapitre.

En expliquant le terme IP cores vous avez sans doute compris que nous allons utiliser une implémentation matérielle, car la mise en œuvre traditionnelle par un logiciel fonctionnant dans un processeur généraliste ne répondra pas vraiment aux exigences en temps réel dans de nombreux cas, en particulier dans l'intelligence artificielle, contrairement à cela, l'implémentation matérielle permet aux réseaux neuronaux de tirer pleinement parti de leur parallélisme inhérent et d'exécuter les instructions plus rapidement.

En parlant des FPGA et des ASIC vous aurez sans doute aussi deviné que nous allons utiliser des circuits numériques, ceci se justifie facilement, car l'implémentation numérique est plus populaire et de multiples avantages : une plus grande précision, une faible sensibilité aux bruits et une plus grande flexibilité. D'autre part, les systèmes analogiques sont plus difficiles à concevoir et ne peuvent être réalisables pour les applications à grande échelle [33].

---

1. FIELD PROGRAMMABLE GATE ARRAY

2. APPLICATION SPECIFIC INTEGRATED CHIP

L'implémentation numérique des RNAs peut être réalisée sur plusieurs types de circuits numériques tels que l'implémentation sur (FPGA), l'implémentation sur (DSP)<sup>3</sup> ou encore l'implémentation sur (ASIC) [33, 12].

L'implémentation sur des DSP n'est pas matérielle mais séquentielle, ce qui la rend une architecture non-parallèle. D'autre part, l'implémentation ASIC n'offre ni la possibilité de reconfiguration, ni de répétabilité ou de testabilité. En conséquence, l'implémentation FPGA est la plus appropriée pour les RNAs, car elle préserve l'architecture parallèle des neurones et peut être reconfigurée par l'utilisateur.

Cependant, il existe certains défis et compromis pour la mise en œuvre des réseaux de neurones sur les FPGA. Généralement, il nécessite de grandes ressources en raison des fonctions d'activation non-linéaires et de plusieurs poids synaptiques (multiplicateurs) présents dans le réseau. C'est un problème majeur, car il devrait y avoir un équilibre entre précision, rapidité et le coût (consommation en ressources logiques).

En effet, malgré que la réalisation d'un RNA fonctionnel sur FPGA est difficile, mais ceci reste incomparable devant le défi de lui attribuer les fonctionnalités d'un IP core configurable qui demande un temps de développement important. Dans ce chapitre, nous allons essayer d'expliquer ceci.

## 3.2 Le circuit FPGA utilisé et méthodes de développement

Nous jugeons qu'il n'est pas essentiel de faire tout un chapitre pour introduire les circuits FPGAs, le langage descriptions matériel VHDL ou encore le langage C++ utilisé afin de créer notre application, car tous ces derniers ne sont pas l'objet de ce travail et peuvent être facilement documentés sur le net.

Nous avons utilisé deux langages qui n'ont aucune relation entre eux, le VHDL est un langage de description matériel alors que le C++ est un langage machine, le C++ n'est qu'un outil de développement, on ne l'utilise pas comme une couche d'abstraction tel que le fait l'application HLS de Xilinx par exemple. D'une autre manière on utilise le C++ afin de créer une base d'interaction avec l'utilisateur général pour qu'il puisse introduire les différents paramètres de configurations qu'il désire (nombre de RNA à créer, Mode de fonctionnement, nombre d'entrées, sorties, couches cachées, neurones dans chaque couche...), ensuite l'application C++ aura le rôle de construire les fichiers sources VHDL des différents IP cores d'une façon automatique et intelligente.

## 3.3 Prise en main de l'application C++

Nous avons tout fait pour rendre l'application très simple à l'utilisation, pour que tous les domaines de la recherche puissent en profiter.

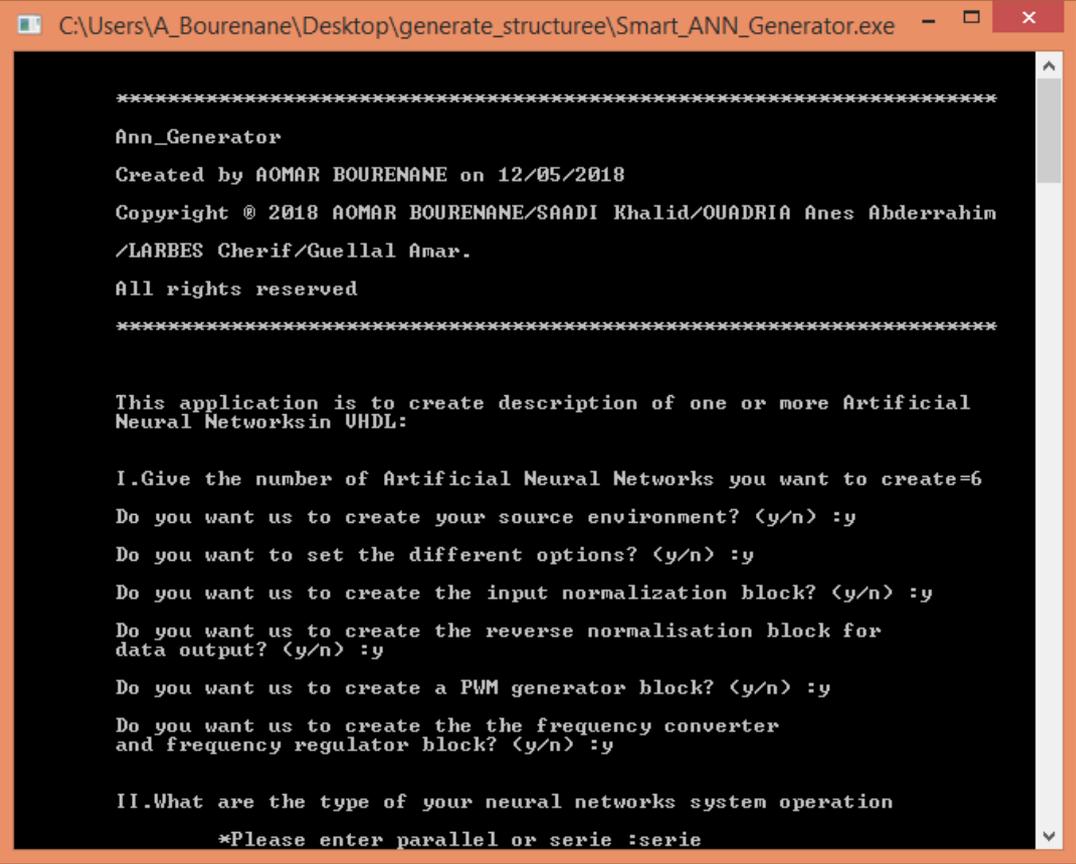
---

3. DIGITAL SIGNAL PROCESSOR

Nous avons préféré expliquer au début la méthode de mise en marche de l'application ensuite expliquer son mécanisme de fonctionnement.

### 3.3.1 Interface de l'application C++

L'interface de l'application est sous forme d'un terminal qui permettra à l'utilisateur d'introduire les différents paramètres comme le montre la figure 3.1 ci-dessous.



```
*****
Ann_Generator
Created by AOMAR BOUREMANE on 12/05/2018
Copyright © 2018 AOMAR BOUREMANE/SAADI Khalid/OUADRIA Anes Abderrahim
/LARBES Cherif/Guellal Amar.
All rights reserved
*****

This application is to create description of one or more Artificial
Neural Networks in VHDL:

I.Give the number of Artificial Neural Networks you want to create=6
Do you want us to create your source environment? (y/n) :y
Do you want to set the different options? (y/n) :y
Do you want us to create the input normalization block? (y/n) :y
Do you want us to create the reverse normalisation block for
data output? (y/n) :y
Do you want us to create a PWM generator block? (y/n) :y
Do you want us to create the the frequency converter
and frequency regulator block? (y/n) :y

II.What are the type of your neural networks system operation
*Please enter parallel or serie :serie
```

FIGURE 3.1 – Interface de l'application C++

### 3.3.2 Étapes de configuration

Les étapes de configuration sont simples, on les présente dans l'ordre comme suit :

1. Dans la première étape le programme nous demandera le nombre des RNAs à créer.
2. Le programme nous demande ensuite si on veut créer l'environnement source dans lequel on trouve tous les fichiers sources que l'application

aura besoin pour générer les RNAs tels que les poids, les biais, les valeurs max et min des entrées et sorties, ces derniers doivent être alimentés par l'utilisateur lui même<sup>4</sup> après avoir fait l'étape d'apprentissage expliquée en chapitre 2.

Exemple pour nombre de RNA=6.

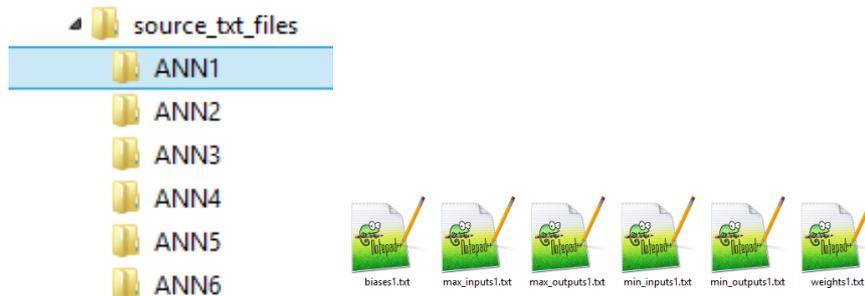


FIGURE 3.2 – Exemple du dossier et fichiers sources pour 6 RNAs

3. Le programme nous demandera ensuite si nous voulons défiler les différentes options, dans cette partie on trouve des blocs optionnels que l'utilisateur choisira ou pas d'implémenter. Cette étape est expliquée en détail dans la section 3.5.2.
4. Le programme nous demandera dans cette étape de choisir un mode de fonctionnement, ceci dépend particulièrement de l'application à réaliser en utilisant les réseaux de neurones, les modes de fonctionnement sont expliqué dans la section II.
5. Dans cette dernière étape, le programme nous demandera d'introduire les différents paramètres des différent RNAs l'un après l'autre. Ces paramètres sont : le nombre de couches cachées, le nombre d'entrées dans la couche d'entrée, le nombre de neurones dans la couche cachée et dans la couche de sortie, le type de fonction d'activation des neurones de : la couche cachée et de la couche de sortie.

Durant l'exécution des différentes étapes, l'application nous affichera s'il y a eu des erreurs, si les différents paramètres ont bien été chargés, dans le cas où nous avons choisi le fonctionnement série, l'application nous indiquera si elle a décidé de mettre en œuvre ce fonctionnement en cas de possibilité de concaténation des RNAs comme le montre la figure 3.3 présentée dans la page suivante.

La finalité de ce processus est la génération d'un dossier qui contient les codes sources VHDL des différents IP cores. La figure 3.4 présentée dans la page suivante montre un exemple de dossier final pour 6 RNAs dans les deux modes de fonctionnement série et parallèle.

4. Remarque importante, si le dossier source est déjà existant il faut choisir non sinon ce dernier sera écrasé

```

*****
Your system is conceivable:the 2 neural network are concatenated
into a single neural network of different number of outputs neurons
*****

./source_txt_files/ANN1/min_inputs1.txt is charged
./source_txt_files/ANN1/max_inputs1.txt is charged
./source_txt_files/ANN1/max_outputs1.txt is charged
./source_txt_files/ANN1/min_outputs1.txt is charged

Weights between the Input layer and first Hidden layer are
charged from ./source_txt_files/ANN1/weights1.txt

Biases of the first Hidden layer is charged from
./source_txt_files/ANN1/biases1.txt

Weights between the last Hidden layer and Output layer are
charged from ./source_txt_files/ANN1/weights1.txt

Biases of the Output layer is charged from
./source_txt_files/ANN1/biases1.txt

./source_txt_files/ANN2/min_inputs2.txt is charged
./source_txt_files/ANN2/max_inputs2.txt is charged
./source_txt_files/ANN2/max_outputs2.txt is charged
./source_txt_files/ANN2/min_outputs2.txt is charged

Weights between the Input layer and first Hidden layer are
charged from ./source_txt_files/ANN2/weights2.txt

Biases of the first Hidden layer is charged from
./source_txt_files/ANN2/biases2.txt

Weights between the last Hidden layer and Output layer are
charged from ./source_txt_files/ANN2/weights2.txt

Biases of the Output layer is charged from
./source_txt_files/ANN2/biases2.txt

```

FIGURE 3.3 – Exemples de messages pour 2 RNAs

On remarque que dans le cas où le fonctionnement est parallèle on trouve 6 sous-dossiers et dans le cas série on ne trouve qu'un seul voir la figure 3.4, cela est évident dans le mode série le résultat n'est qu'un seul RNA contenant les 6 RNAs créés en parallèles.

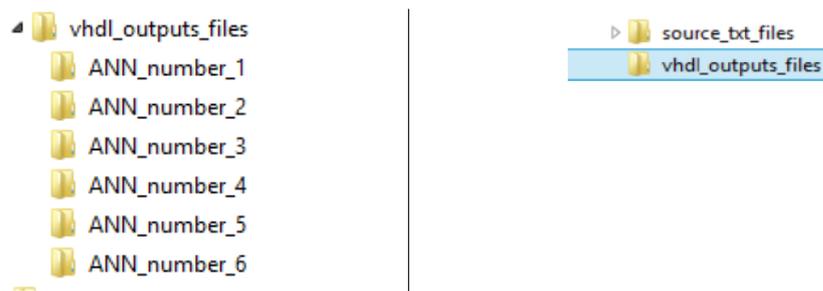


FIGURE 3.4 – Exemple de dossier final pour 6 RNAs dans les deux modes de fonctionnement série et parallèle.

Dans chacun des dossiers ci-dessus, on trouve les fichiers présentés dans la figure suivante.(à gauche pour un des dossiers en fonctionnement parallèle et à droite fichiers sources pour le fonctionnement série)

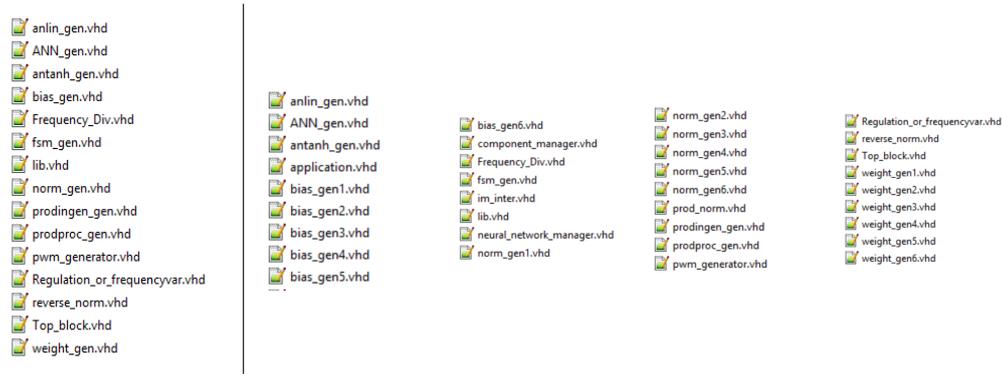


FIGURE 3.5 – Exemple de code VHDL généré pour 6 RNAs dans les deux modes de fonctionnement série et parallèle.

### 3.4 L’application vue de l’intérieur

#### 3.4.1 Structure de l’application C++

L’application que nous avons réalisé contient un très grand nombre de lignes de code (environ 6382 lignes!) ce qui nous a obligé de travailler dans un environnement modulaire.

On peut diviser les fichiers sources C++ de l’application en deux grandes catégories, dans la première, on ne trouve que les fonctions de généralisation qui n’ont aucune relation avec le VHDL, dans la deuxième catégorie, on trouve les fonctions de construction des fichier VHDL, cette dernière est composée de deux sous-catégories de code, l’une permet la construction des codes sources VHDL des IP cores du mode parallèle et l’autre permet la construction des codes sources VHDL des IP cores du mode série. Le diagramme suivant résume la structure de l’application.

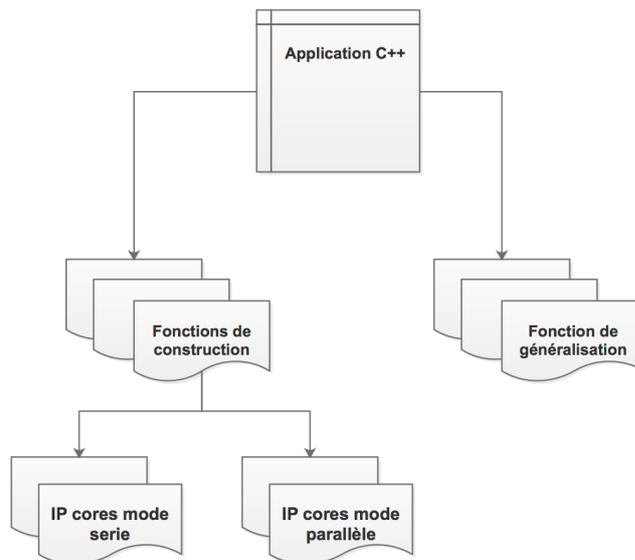


FIGURE 3.6 – Exemples de messages pour 2 RNAs

### 3.4.2 Développement des fonctions de base de l'application C++

Dans cette partie, nous présenterons les méthodes de bases que nous avons utilisées afin de réaliser les fichiers sources C++.

#### i Fonctions de construction des fichiers VHDL série/parallèle

L'idée de codage utilisée pour générer les fichiers VHDL des deux modes est très simple. Nous avons utilisé les commandes de lecture et d'écriture à partir du fichiers [21], puis les descriptions VHDL sont écrites en fonction des spécifications choisie par l'utilisateur.

Exemple :

```
fichier<<endl<<"use IEEE.STD_LOGIC_1164.ALL;";
```

Cette instruction du code C++ précédente permet d'ajouter par exemple la bibliothèque "std\_logic\_1164.all" à un fichier de description VHDL.

#### ii Fonctions de Généralisation

L'instruction du code C++ précédente permet d'ajouter la bibliothèque "std\_logic\_1164.all" à un fichier de description VHDL. Cependant, la ligne montrée est constante, elle n'a pas besoin d'être modifiée avec différentes anatomies RNAs qui sont spécifiées par l'utilisateur, le défi dans cette application était la manière de générer des codes VDHL qui seront adaptés aux différents paramètres des RNAs introduits par l'utilisateur.

Exemple :

```
fichier<<endl<<"    neuron: in std_logic_vector("<<countxdownto0(nann)<<" downto 0);";
```

Dans cet exemple, nous pouvons voir l'instruction C++ qui génère une entrée "neuron", mais cette fois ci le signal est utilisé pour choisir un des RNAs créés, ce qui fait que la taille de ce signal dépend du nombre de RNAs. Ainsi, si un utilisateur veut 5 RNAs, le nombre 5 ne pourra être représenté qu'avec 3 bits au minimum, le résultat donne alors "neuron : in\_std\_logic\_vector(2 downto 0);".

Un autre défi est l'initialisation ou assignation des signaux, dynamiser la virgule<sup>5</sup>...etc.

Cela a été fait par un ensemble de fonctions simples, ci-dessous une liste de quelques fonctions utilisées.

---

5. Le nombre de bits de la partie entière et la partie fractionnaire d'un signal devient variable.

```

8 void charge_file(int nhl[],float* x,string input_file,int input,int nhl);
9 void charge_parameters(int nhl[],float* w,float* b,string weights_file,string biases_file,int nhl,int h);
10 void max_of_sizes(int* maxhlayer,int* maxlayer,int* maxihlayer,int* maxholayer,int nhl,int nhl[]);
11 std::string functionactivation(string DIRECTORY_name);
12 std::string toBinary( int n, int nmax);
13 std::string generatornchar(char c, int nbit);
14 std::string vectaffect(int c, int up,int down,int nbit);
15 std::string weightinitiate(float n);
16 std::string biasinitiate(float a);
17 std::string output_initiate(float a);
18 std::string xtinitiate(float a);
19 std::string emininitiate(float a,float b);
20 std::string diff_initiate(float a);
21 int* dynamic_width( float input,int size);

```

La fonction de la ligne 12 par exemple appelée "toBinary" prend un nombre  $n$ , et sa valeur maximale et donne comme sortie dans une chaîne la représentation de ce nombre dans la représentation complément à 2 avec une largeur appropriée.

Les fonctions des lignes 15 et 16 par exemple transforment les valeurs de biais et de poids fournies dans les fichiers par les utilisateurs en leur représentation binaire appropriée.

## 3.5 Description VHDL

### 3.5.1 Structure de la description VHDL

Cette partie sera consacrée à la description matérielle des différents IP cores qui seront générées par l'application C++ présentée ci-dessus, on peut les classer en deux grandes catégories : les blocs fonctionnels spécifiques à l'application SHE PWM et blocs fonctionnels spécifiques à la génération des RNAs.

### 3.5.2 Blocs fonctionnels spécifiques à l'application SHE PWM

#### i Sélectionneur de RNA

D'après l'étude qui a été faite dans le chapitre 2 section "solution originale 2.8" et le tableau 2.1 nous avons divisé l'intervalle de variation de  $im$  en six intervalles. Pour chaque intervalle on a construit un réseau RNA-i.

Par conséquent, le but du module "Sélection du Réseau" est la sélection du réseau RNA-i qui convient à l'entrée  $im$ .

Dans l'étape d'apprentissage nous avons exprimé  $im$  en pourcentage c'est-à-dire  $im$  est compris entre 0 et 100 %. Dans la conception nous avons dimensionné l'entrée  $im$  sur dix bits, sept bits pour la partie entière et trois bits pour la partie fractionnaire, donc le pas de variation de  $im$  est  $2^{-4} = 0.0625\%$ . Ce

choix peut être changé selon le choix du pas de variation de  $im$ . De plus, pour simplifier le choix de chaque intervalle en fonction de  $im$  on a défini les limites de chaque intervalle comme indiqué sur le tableau 3.1. Ce choix permet donc de minimiser l'espace consommé sur le circuit FPGA. Cette entrée doit être adaptée à l'entrée de nos différents RNAs par un autre module que nous allons introduire dans une autre partie.

TABLE 3.1 – Intervalle de variation de  $im$  pour chaque réseau RNA-i

intervalle de variation RNA-i	La limite inférieure de $im$ en pourcentage $im_{min} \leq im < im_{max}$	Intervalle en binaire	(inter)
RNA-6	$01\% \leq im < 16\%$	000000000000	110
RNA-5	$16\% \leq im < 32\%$	001000000000	101
RNA-4	$32\% \leq im < 56\%$	010000000000	100
RNA-3	$56\% \leq im < 76\%$	011100000000	011
RNA-2	$76\% \leq im < 92\%$	100110000000	010
RNA-1	$92\% \leq im < 100\%$	101110000000	001

D'après le Tableau 3.1 on trouve

RNA-1 =  $im(10)$  and  $((\text{not } im(9)$  and  $im(8)$  and  $im(7)$  and  $im(6)))$  or  $((im(9)$  and  $\text{not } im(8)$  and  $\text{not } im(7))$  and  $(\text{not } im(6)$  or  $(im(6)$  and  $\text{not } im(5)$  and  $\text{not } im(4))))$ ;

RNA-2 =  $im(10)$  and  $\text{not } im(9)$  and  $((\text{not } im(8)$  and  $im(7)$  and  $im(6))$  or  $(im(8)$  and  $(\text{not } im(7)$  or  $\text{not } im(6))))$ ;

RNA-3 =  $(\text{not } im(10)$  and  $im(9)$  and  $im(8)$  and  $im(7))$  or  $(im(10)$  and  $\text{not } im(9)$  and  $\text{not } im(8)$  and  $(\text{not } im(7)$  or  $\text{not } im(6)))$ ;

RNA-4 =  $\text{not } im(10)$  and  $im(9)$  and  $(\text{not } im(8)$  or  $\text{not } im(7))$ ;

RNA-5 =  $\text{not } im(10)$  and  $\text{not } im(9)$  and  $im(8)$ ;

RNA-6 =  $\text{not } im(10)$  and  $\text{not } im(9)$  and  $\text{not } im(8)$ ;

Inter (3) = RNA-6 or RNA-4 or RNA-2

Inter (2) = RNA-5 or RNA-4 or RNA-1

Inter (1) = RNA-3 or RNA-2 or RNA-1

Où "Inter" représente l'indice du réseau en binaire, voir Tableau 3.1.

En se basant sur les équations écrite ci-dessus une conception basée sur la logique combinatoire du module "Sélectionneur de RNA" est créée sous VHDL afin de sélectionner le réseau adéquat à l'entrée  $im$ .

## ii Variateur de plage de fréquence "frequency range variator"

Ce bloc est décrit en détail dans le chapitre 1 section 1.3.2, nous ne allons pas encore l'expliquer dans cette partie.

L'horloge utilisée pour les RNAs est de 10ns (fixée par la carte de développement), pour pouvoir contrôler la plage de variation de la fréquence on a implémenter un diviseur de fréquence à fréquence de sortie variable contrôlée par l'entrée externe "freq" selon l'équation 1.6 vue dans le chapitre 1, cette dernière est utilisée comme une horloge dans le générateur PWM afin de contrôler la fréquence de fondamental des signaux PWMs comme le montre la figure 3.8.

## iii Le bloc régulateur interne de fréquence

Ce bloc est décrit en détail dans le chapitre 1 section 1.3.2, nous ne allons pas encore l'expliquer dans cette partie.

Pour réaliser ce bloc, on doit garder la fréquence fixe quel que soit  $im$ . Cependant, la fréquence et l'indice de modulation sont fortement couplés selon l'équation (1.4) vue dans le chapitre 1, pour les découpler, on doit diviser cette équation par  $im$  ce qui posera un problème d'implémentation sur un circuit FPGA. L'astuce que nous avons utilisé pour éviter cette division est de se servir du diviseur de fréquence qui a été déjà implémenté dans le bloc "frequency range variator" en injectant  $x = im \times freq$  dans ce dernier à la place de "freq" ceci va engendrer une division indirecte produisant un système contrôlable par l'équation (1.7) vue dans le chapitre 1.

## iv Génération des signaux PWM

Le but de ce module est de générer les trois signaux PWMs à partir des angles de commutation  $\alpha_i$  générés par le bloc de dé-normalisation qu'on va exposer dans la suite de ce document.

La génération se fait en deux étapes comme le montre la figure 3.7 suivante .

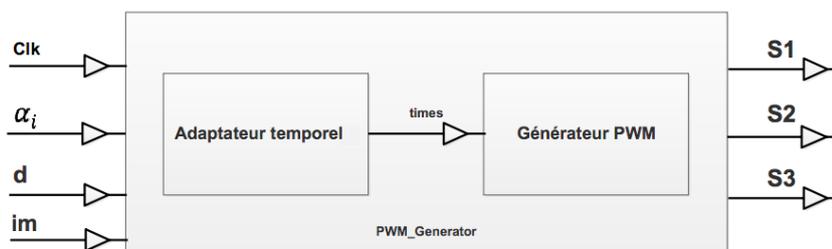


FIGURE 3.7 – Schéma synoptique du module "générateur PWMs"

Avec  $\alpha_i$  sont les angles de commutation en degrés, (S1, S2 et S3) signaux PWMs et  $d$  le nombre d'angles de commutation (voir le tableau 1.4)

L'horloge Clk utilisée dans ce bloc correspond à la sortie du variateur de plage de fréquence.

### a Étape 1

Les angles de commutation doivent passer par un adaptateur temporel que nous avons expliqué en détail dans le chapitre 1 section 2 : ce bloc permet de définir la base du temps du signal en convertissant les angles de commutations  $\alpha_i$  en instants de commutation  $ti$  en utilisant l'équation (1.29) .

Faisons dans une première phase des multiplications par le coefficient suivant

$$coef = \frac{10^5 \alpha_i}{18}$$

ce qui nous donne des instants de commutation non divisés par  $im$  qu'on notera  $time_i$  tel que

$$time_i = \alpha_i \times coef$$

Dans une deuxième phase, nous devons faire une a division sur  $im$  ce qui pose un problème d'implémentation sur un circuit FPGA. Pour éviter cette division, dans l'étape de génération de signaux de commande, à chaque front montant de l'horloge ( $1\mu s$ ), au lieu d'ajouter 1 à l'accumulateur de temps "cm", on ajoute  $im$  et on compare "cm" avec  $times_i$ . sans oublier que nous avons dimensionné  $im$  sur 11 bits dont 4 bits représente la partie fractionnaire En conséquence, les valeurs de  $times_i$  doivent être multipliées par  $2^4 = 16$ .

### b Étape 2

Pour générer les signaux PWMs  $s1$ ,  $s2$  et  $s3$  à partir des instants de commutation  $times_i$  calculés dans l'étape précédente, on doit utiliser un générateur PWM qui exécute plusieurs tâches.

Au début, le signal  $s1$  est à 1 et son accumulateur de temps "cm1" commence à 0, puis à chaque front, on l'incrémente par  $im$  et on le compare avec  $times_1$ . Lorsque "cm1" devient supérieur à  $times_1$ , on inverse  $s1$  et on commence à comparer "cm1" avec  $times_2$  et ainsi de suite jusqu'à ce que "cm1" devient supérieur à  $times_m$ . Après on commence à comparer "cm1" avec  $times_{T/2} - times_m$ , où  $times_{T/2} = 111101000010010000000000$ , la valeur de  $times$  correspondant à la demi période ( $\alpha = 180$ ). Lorsque "cm1" devient supérieur à  $times_{T/2} - times_m$ , on inverse  $s1$  et on commence à comparer "cm1" avec  $times_{T/2} - times_{m-1}$  et ainsi de suite jusqu'à ce que "cm1" devient supérieur à  $times_{T/2} - times_1$ . Après, on compare "cm1" avec  $times_{T/2}$ . Lorsque "cm1" devient supérieur à  $times_{T/2}$ ,

on inverse s1 et on met "cm1" à 0 et on répète le processus, Figure 3.9.

D'après la base de données utilisée durant l'apprentissage pour tous les réseaux RNA-i, on a remarqué que  $times_m < 60^\circ$ . Aussi, on sait que s2 est déphasé de  $120^\circ$  par rapport à s1, et d'après la figure 1.16, on voit que après  $120^\circ$  le signal s1=0 et l'instant de commutation suivant se trouve à  $times_{T/2} - times_m$ . En conséquence, au début le signal s2 démarre à 0, son accumulateur de temps "cm2" commence à  $times_{T/3} = 101000101100001010101010$  et "cm2" sera comparé avec l'instant de commutation  $times_{T/2} - times_m$  et on répète le même processus de génération de s1, Figure 3.9. Pour le signal s3, on sait qu'il est diphasé de  $240^\circ$  par rapport à s1, et d'après la figure 1.16 on voit que après  $240^\circ$  le signal s1=1 et l'instant de commutation suivant se trouve à  $times_{T/2} - times_m$ . En conséquence, au début le signal s3 démarre à 1, son accumulateur de temps "cm3" commence à  $times_{T/6} = 010100010110000101010101$  et "cm3" sera comparé avec l'instant de commutation  $times_{T/2} - times_m$  et on répète le même processus de génération de s1, Figure 3.9.

D'après la Figure 3.9, les trois signaux sont générés parallèlement et sont indépendants l'un par rapport à l'autre ce qui confirme l'utilité du circuit FPGA pour générer un signal PWM triphasé.

Le schéma de branchement des trois modules, frequency range variator, régulateur interne de fréquence et Génération des signaux PWM <sup>6</sup> est indiqué ci-dessous.

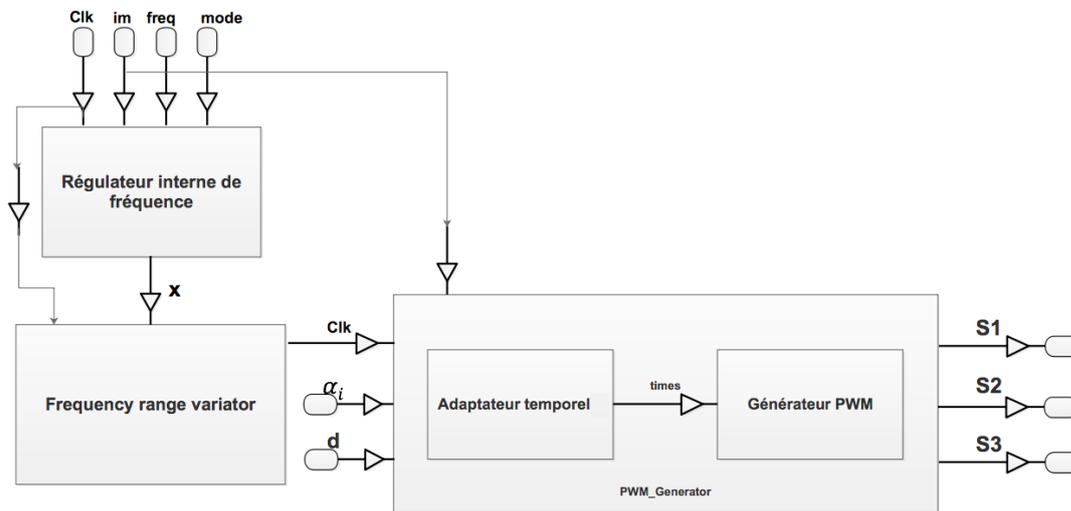


FIGURE 3.8 – Schéma de branchement des trois modules

6. Le rôle de l'entrée mode est bien expliqué dans le chapitre 1 section 1.3.2.

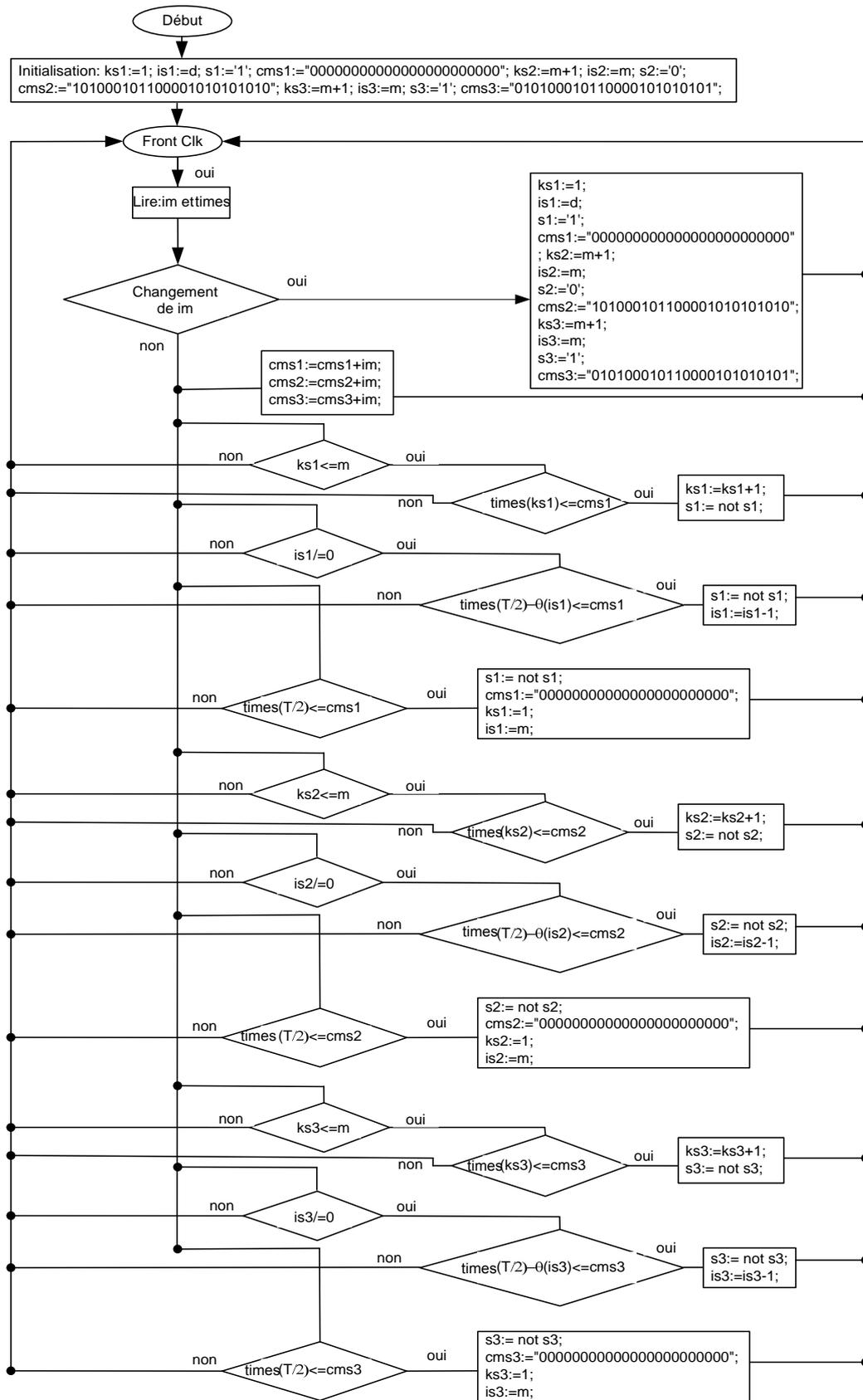


FIGURE 3.9 – Algorithme de la génération des signaux PWMs

### 3.5.3 Blocs fonctionnels spécifiques à la génération des RNAs.

On peut classer ces blocs en trois catégories : les blocs du fonctionnement série, blocs du fonctionnement parallèle, blocs invariants.

#### I Blocs fonctionnels invariants

Ceux sont les IP cores qui ne subissent aucun changement de configuration quelle que soit l'anatomie du réseau de neurones choisie, en plus du régulateur interne de fréquence, le variateur de plage de fréquence et le générateur PWM que nous avons expliqués ci-dessus, on trouve les fonctions d'activations.

##### i Les fonctions d'activations.

Pour avoir plus d'information sur leur utilité et d'autres détails voir le chapitre 1.

Les fonctions d'activations sont gourmandes en surface, en effet, ce dernier est le plus complexe, et naturellement celui qui consomme le plus de ressources FPGA. Compte tenu de la facilité avec laquelle les opérations arithmétiques peuvent être mises en œuvre, la fonction d'activation reste le principal facteur limitant de la performance.

La mise en œuvre de la fonction d'activation est l'un des plus importants problèmes de conception arithmétique lors de la mise en œuvre des RNA sur des cartes FPGA. Il existe de nombreux types de fonctions d'activation, voir le chapitre 2. De notre côté on implémente trois fonctions d'activation qui sont le sigmoïde tangent, le log sigmoïde et la fonction d'activation linéaire. Pour la mise en œuvre des deux premières fonctions non-linéaires, de nombreux travaux ont été proposés [32, 27, 25, 39].

Notre implémentation des trois fonctions est principalement basée sur le travail présenté par Promod Kumar Meher dans son article intitulé "Une table de recherche optimisée pour l'évaluation de la fonction sigmoïde pour les réseaux de neurones artificiels" [25].

L'approche adoptée pour implémenter les fonctions non-linéaires sont la méthode hybride qui utilise une table de correspondance<sup>7</sup> couplée à une approximation linéaire par morceaux<sup>8</sup>.

---

7. Dans les méthodes à base de LUT, la plage d'entrée est divisée en sous-plages égales et chaque sous-plage est approximée par une valeur stockée dans LUT. Cette méthode est utilisée pour implémenter la tangente hyperbolique.

8. L'approximation linéaire généralement par morceaux utilise une série de segments linéaires pour approximer la fonction d'activation. Le nombre et les emplacements de ces segments sont choisis de telle sorte que l'erreur, le temps de traitement et l'utilisation de la zone soient minimisés. L'utilisation de multiplicateurs devrait être évitée pour des implémentations matérielles efficaces utilisant cette méthode d'approximation, car les multiplicateurs sont des composants matériels coûteux en termes de surface et de délai. Cette méthode est utilisée dans [39] pour la mise en œuvre de la fonction tangente hyperbolique et sigmoïde.

### i.a Implémentation de la fonction Tangente-hyperbolique

Cette fonction est définie par l'équation suivante.

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.1)$$

Cette équation produit une courbe en forme de S tel que nous pouvons le voir sur la figure suivante.

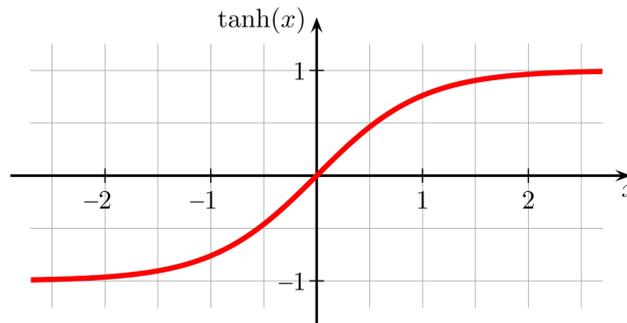


FIGURE 3.10 – La fonction Tangente-hyperbolique

Comme indiqué précédemment pour implémenter cette fonction, il faut diviser les intervalles pour s'assurer que les tables de recherche restent petites et produisent un minimum d'erreur.

#### i.a.1 Caractéristiques du Tanh et stratégie de mise en œuvre

##### ♣ Propriété 1

$$\tanh(x) = -\tanh(-x) \quad (3.2)$$

Compte tenu de cette propriété (symétrie miroir sur l'axe Y), seule la moitié droite de la courbe de la figure 3.10 devrait être stockée. Ceci est fait en effectuant l'opération de complément à 2 sur la sortie LUT si l'entrée est négative.

##### ♣ Propriété 2

$$\lim_{x \rightarrow 0} \tanh(x) = x \quad (3.3)$$

Cela signifie que pour les petites valeurs de l'entrée,  $\tanh(x)$  est linéaire, alors stocker des valeurs dans la LUT pour cette région pourrait être évité, puisque les valeurs  $\tanh(x)$  correspondantes pourraient être obtenues directement à partir des valeurs d'entrée. Cela pourrait être mis en œuvre par un simple multiplexeur.

##### ♣ Propriété 3

$$\lim_{x \rightarrow \infty} \frac{d \tanh(x)}{dx} = 0 \quad (3.4)$$

$$\lim_{x \rightarrow \infty} \tanh(x) = 1 \quad (3.5)$$

D'après ces deux équations, la variation de  $\tanh(x)$  est insignifiante pour les grandes valeurs d'entrée, c'est-à-dire que pour  $|x| \geq 3$  la variation est inférieure à 0,00015 et une valeur de +1 peut être stockée dans le LUT pour toutes les valeurs supérieures à 3.

En utilisant toutes ces propriétés ensemble, les valeurs de  $\tanh(x)$  qui doivent être stockées dans le LUT sont celle qui correspondent à :

$$\delta_{min} < x < \delta_{max}$$

Où  $\delta_{min}$  et  $\delta_{max}$  représentent les valeurs limites qui pourraient être déduites des exigences de précision tel que

$$|\tanh(\delta_{min}) - (\delta_{min})| \leq \epsilon \quad \& \quad |\tanh(\delta_{max}) - (1)| \leq \epsilon$$

où  $\epsilon$  est l'erreur maximale autorisée.

$\delta_{max}$  est la limite des valeurs d'entrée au-dessus desquelles le tanh stocké dans la table LUT est +1. Il faut généralement des valeurs plus petites que 3 pour la raison que  $|\tanh(x) - \tanh(3)| < 0.00015$ , ce qui est une précision très supérieure à ce qui est requis pour de nombreuses applications, y compris les réseaux de neurones. La figure 3.11 montre clairement que pour  $x = 2$ ,  $\tanh(x) > 0,96$ , pour  $x = 2,4$ ,  $\tanh(x) > 0,98$ , et pour  $x = 2,7$ ,  $\tanh(x) > 0,99$ . Ainsi,  $\tanh(x)$  peut être approché par +1 pour  $x > \delta_{max}$  où  $\delta_{max}$  est 2,2, 2.4 ou 2.7 si les erreurs maximales autorisées sont respectivement de 0.04, 0.02 ou 0.01.

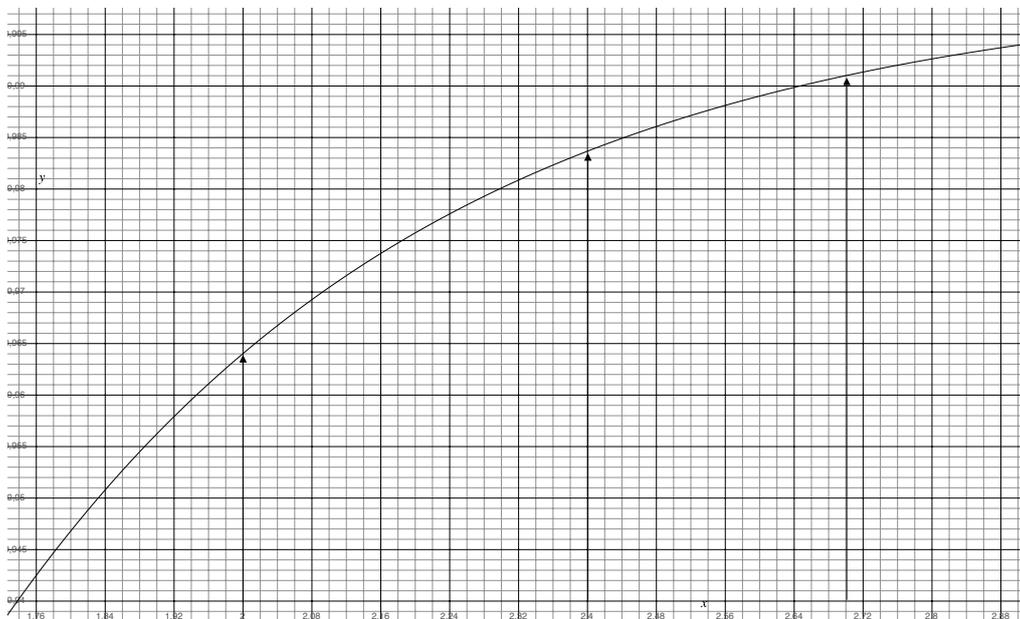


FIGURE 3.11 – La fonction Tangente-hyperbolique

De même, pour choisir  $\delta_{min}$ , nous analysons le comportement du  $\tanh(x)$ . Le développement de Taylor du  $\tanh(x)$  près de zéro donne

$$\tanh(x) = x - \frac{x^3}{3} + \frac{2x^5}{15} - \frac{17x^7}{315} + O(x^9) \quad (3.6)$$

Lorsque  $x$  tend vers zéro, les termes d'ordre élevé peuvent être ignorés.

Nous pouvons trouver une valeur de  $\delta_{min}$  au-dessus duquel nous pouvons supposer que  $\tanh(x) = x$  en utilisant l'équation suivante.

$$\left| \delta_{min} - \left( \delta_{min} - \frac{\delta_{min}^3}{3} + \frac{2\delta_{min}^5}{15} \right) \right| < \epsilon \quad (3.7)$$

En simplifiant nous trouvons.

$$\frac{\delta_{min}^3}{3} + \frac{2\delta_{min}^5}{15} < \epsilon \quad (3.8)$$

En simplifiant, nous trouvons que pour une erreur  $\epsilon = 0.02 \Rightarrow \delta_{min} = 0.390625$

### i.a.2 Conception d'une LUT optimisée par Maher pour la tangente hyperbolique

Contrairement aux lookup-tables conventionnelles où chaque mot de valeur d'entrée correspond à un emplacement dans une LUT, Maher a utilisé dans son travail [45] ce que l'on appelle l'adressage par plage, où une adresse correspond à une plage de valeurs d'entrée, réduisant de ce fait le nombre de mots stockés.

Pour un sous-domaine donné, la valeur stockée est la moyenne des valeurs limites de la fonction dans ce sous-domaine. Ceci est différent des autres travaux où ils stockaient la valeur de la fonction correspondant à l'adresse de la limite inférieure, dans ce cas la différence entre les valeurs maximum et minimum de la fonction pourrait être le double de l'erreur admissible.

La conception du LUT se fait en suivant les étapes proposées par Maher qui sont.

#### ♣ Détermination des limites supérieure et inférieure de l'entrée LUT ( $\delta_{min}$ et $\delta_{max}$ )

pour  $\epsilon = 0.02 \Rightarrow \delta_{min} = 0.390625$  &  $\delta_{max} = 2.4$ .

#### ♣ Sélection du nombre de bits de l'entrée (largeur de l'adresse d'entrée "précision")

Par des simulations, il a été trouvé qu'une largeur de 9 bits de valeurs d'entrée représentées dans le complément à 2 permet d'avoir  $|\tanh(x1) -$

$|\tanh(x_2) - \tanh(x_1)| \leq \epsilon = 0,02$  où  $x_1$  et  $x_2$  sont deux entrées consécutives.

♣ **Détermination des limites de chaque bande d’adresses**

La plage de  $\tanh(x)$  pour  $0.3906250 < x < 2.4$  est divisée en  $n$  sous-domaines  $R_i(x_{i1}, x_{i2})$  tels que  $|\tanh(x_{i1}) - \tanh(x_{i2})| \leq 2\epsilon$ . alors tout  $(x_{i1}, x_{i2})$  doit être déterminé tel que  $i = 1, 2, \dots, n$ . Où  $n$  est déterminé comme la limite supérieure du dernier domaine pour lequel  $\tanh(x_{n2}) \geq 2.4$ .

♣ **Affectation de de la LUT**

On stocke la moyenne des valeurs limites de la fonction, ce qui signifie.

$$\tanh(x_i) = \frac{\tanh(x_{i1}) + \tanh(x_{i2})}{2}$$

En se basant sur ces étapes, nous avons développé un simple programme C++ pour générer les mots de la LUT, les résultats sont donnés dans le tableau 3.2, contrairement au travail de Maher, la valeur stockée est la plus proche de la moyenne des limites possibles imposé par la représentation binaire, ce qui signifie que l’erreur dans un sous-domaine peut dépasser l’erreur imposée par le critère  $|\tanh(x_{i1}) - \tanh(x_{i2})| \leq 2\epsilon$ . Comme nous pouvons le voir dans le tableau 3.2, l’erreur maximale est devenue 0.0338394.

Remarque : N/A veut dire Non Applicable.

TABLE 3.2 – La LUT de la fonction tangente-hyperbolique

N° du mot :LUT	$x_{i1}$	$x_{i2}$	Moyenne $\tanh(x)$	Valeur stockées	Valeur stockées en binaire	Erreur maximale
	...	0.390625	x	N/A	N/A	N/A
1	0.390625	0.453125	0.398182	0.390625	00011001	0.0338394
2	0.453125	0.515625	0.44939	0.453125	00011101	0.0286606
3	0.515625	0.578125	0.497809	0.5	00100000	0.0256837
4	0.578125	0.640625	0.543313	0.546875	00100011	0.0255737
5	0.640625	0.703125	0.585836	0.578125	00100101	0.0282226
6	0.703125	0.78125	0.629886	0.625	00101000	0.0284236
7	0.78125	0.859375	0.67468	0.671875	00101011	0.0240605
8	0.859375	0.9375	0.715003	0.71875	00101110	0.0228145
9	0.9375	1.048675	0.757681	0.75	00110000	0.0312907
10	1.048675	1.171875	0.803082	0.796875	00110011	0.0279973
11	1.171875	1.328125	0.846831	0.84375	00110110	0.0250403
12	1.328125	1.53125	0.889714	0.890625	00111001	0.0218347
13	1.53125	1.859375	0.93163	0.9375	00111100	0.0268617
14	1.859375	2.90625	0.973329	0.96875	00111110	0.0252879
15	2.90625	...	NA	1	01000000	NA

La Figure 3.12 montre que la mise en œuvre de la tangente hyperbolique est constituée de plusieurs blocs, assurant le fonctionnement décrit précédemment.

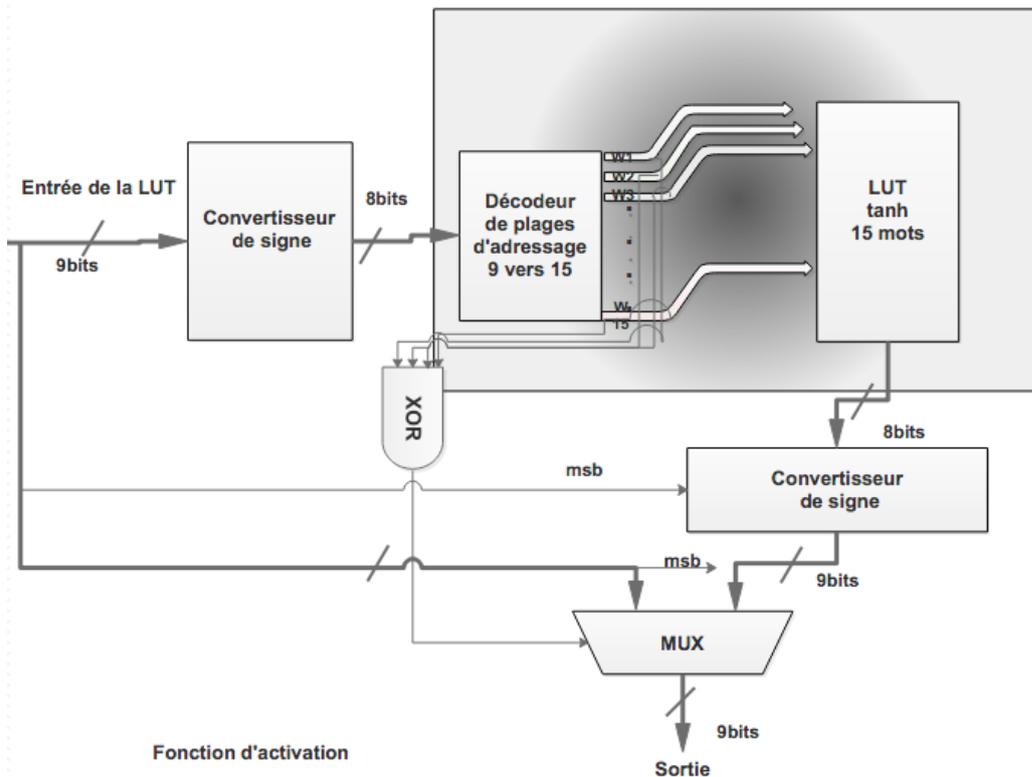


FIGURE 3.12 – Implémentation de la fonction tanh

### ♣ Un convertisseur de signe

Il a été implémenté avant et après le bloc LUT. Le rôle du premier est de calculer la grandeur de l'entrée pour l'affecter à la LUT, puisque celui-ci ne traite que des nombres positifs. Ainsi, lorsque l'entrée est négative, le mot LUT correspondant à l'amplitude d'entrée devient négatif pour avoir la valeur  $\tanh(x)$  correcte de cette entrée négative  $x$  grâce au second convertisseur de signe.

### ♣ Un décodeur de plage d'adressage

Il a été implémenté pour effectuer l'adressage des plages de la LUT. C'est-à-dire que lorsqu'il est alimenté avec la grandeur d'entrée, il détermine la plage correspondante, à travers ses sorties de sélection de mots ( $w_1, w_2, \dots, w_{15}$ ).

### ♣ Un multiplexeur

Son rôle est de laisser passer les mots LUT ou l'entrée directement. La décision de laisser passer l'entrée directement ne se fait que si  $|x| < 0,390625$ . Cela pourrait être réalisé en utilisant une porte logique NOR pour tous les signaux ( $w_1, w_2, \dots, w_{15}$ )

La Figure 3.13 montre les valeurs des signaux de sélection de mots ( $w_1, w_2, \dots, w_{15}$ ) par rapport à l'ensemble de ses entrées. Comme il est clair sur la figure, les entrées sont divisées en trois régions.

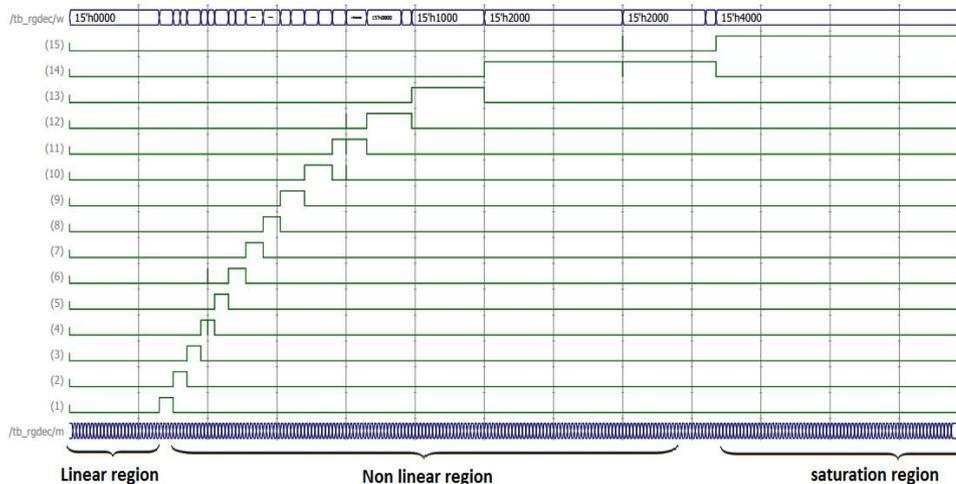


FIGURE 3.13 – Simulation de décodeur de plages d'adressages

### ♣ Une région linéaire

L'entrée est passée directement à la sortie sans passer par la LUT, cette région est sélectionnée lorsque tous les signaux ( $w_1, w_2, \dots, w_{15}$ ) sont au niveau bas (la valeur 0).

### ♣ Une région non-linéaire

nous pouvons voir clairement la forme de tangente hyperbolique formée lorsque chaque signal des signaux ( $w_1, w_2, \dots, et w_{15}$ ) obtient la valeur 1 au niveau de la sous-plage qui lui appropriée et 0 dans le autres sous-plages.

### ♣ Une région de saturation

Dans cette région, la sortie correspondra à la valeur fixe 1 quel que soit l'entrée  $x$ , car dans ce cas  $|x| \geq 2.4$  ceci est réalisé à l'aide du bloc bit-reducer voir [i.a.4](#).

Ceci conclut que le décodeur de gamme fonctionne correctement et décode les valeurs d'entrée en sous-plages qui peuvent être sélectionnée lorsque leur signal de sélection  $w_i$  est mise à 1.

Cette figure 3.14 montre que le bloc fonctionnel Tangent fonctionne parfaitement, la première valeur d'entrée est dans la plage linéaire, donc la sortie est égale à l'entrée et la troisième valeur de l'entrée est dans la région non linéaire.

/tanh/e	9'b000001100	9'b000011001	9'b000111100
/tanh/s	9'b000001100	9'b000011001	9'b000101110

FIGURE 3.14 – Simulation de décodeur de plages d'adressages

### i.b Simulation de la fonction tanh

Nous avons utilisé le même processus d'implémentation précédent pour générer une LUT optimisée pour une fonction sigmoïde.

$$\text{sig}_\alpha(x) = \frac{1}{1 + e^{-\alpha x}} \quad (3.9)$$

Dans notre implémentation nous avons choisi  $\alpha = 2$  pour faciliter l'approche d'approximation ce qui donne

$$\text{sig}_2(x) = \frac{1}{1 + e^{-2x}} \quad (3.10)$$

Cette fonction a également une courbe en forme de S, cependant, contrairement à la fonction tangente-hyperbolique, celle-ci varie de 0 à 1 et non de -1 à 1 (voir Figure 3.15).

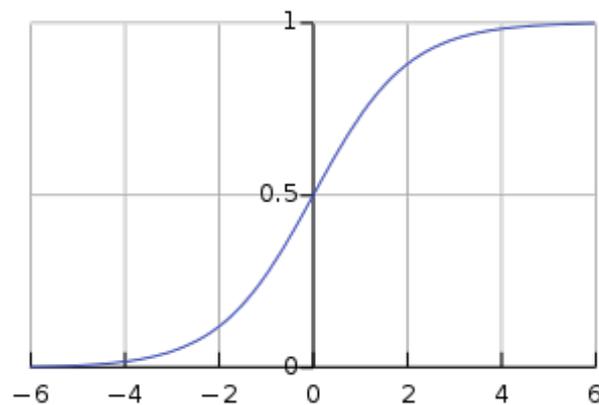


FIGURE 3.15 – La fonction sigmoïde2(x)

En faisant le développement de Taylor au premier ordre, nous obtenons.

$$\text{sig}_2(x) = \frac{1}{2} + x + O(x^2) \quad (3.11)$$

Pour faciliter la mise en œuvre, nous avons envisagé de mettre en place la fonction.

$$f(x) = \frac{1}{1 + e^{-2x}} - \frac{1}{2} \quad (3.12)$$

La nouvelle fonction a clairement des propriétés similaires à  $\tanh(x)$ .

- ♣  $f(x) = -f(x)$
- ♣  $\lim_{x \rightarrow 0} f(x) = x$
- ♣  $\lim_{x \rightarrow \infty} \frac{df(x)}{dx} = 0$
- ♣  $\lim_{x \rightarrow \infty} f(x) = 0.5$

Après avoir implémenté la LUT de  $f(x)$  on peut obtenir la sigmoïde en ajoutant un  $\frac{1}{2}$  à sa sortie.

Nous avons fait un code C++ pour générer la LUT pour la fonction  $f(x)$ , les résultats sont dans le tableau 3.3.

TABLE 3.3 – La LUT de la fonction  $f(x) = sig2(x) - \frac{1}{2}$

N° du mot :LUT	$x_{i1}$	$x_{i2}$	Moyenne $f(x)$	Valeur stockées	Valeur stockées en binaire	Erreur maximale
	...	0.03125	x	N/A	N/A	N/A
1	0.03125	0.09375	0.031179	0.03125	00000010	0.0156301
2	0.09375	0.15625	0.0621168	0.0625	00000100	0.0157618
3	0.15625	0.21875	0.0925793	0.09375	00000110	0.0162546
4	0.21875	0.28125	0.122347	0.125	00001000	0.0173368
5	0.28125	0.34375	0.151221	0.15625	00001010	0.0192192
6	0.34375	0.40625	0.179026	0.171875	00001011	0.020767
7	0.40625	0.46875	0.205618	0.203125	00001101	0.0154694
8	0.46875	0.546875	0.233841	0.234375	00001111	0.0157806
9	0.546875	0.625	0.263194	0.265625	00010001	0.0165378
10	0.625	0.71875	0.292684	0.296875	00010011	0.0195751
11	0.71875	0.8125	0.321775	0.328125	00010101	0.0200578
12	0.8125	0.921875	0.349438	0.34375	00010110	0.0196416
13	0.921875	1.0625	0.37835	0.375	00011000	0.0183094
14	1.0625	1.25	0.408726	0.40625	00011010	0.0178918
15	1.25	1.53125	0.43973	0.4375	00011100	0.0178191
16	1.53125	....	0.467493	0.46875	00011110	0.0134309

### Simulation

La simulation montre le bon fonctionnement du bloc fonctionnel sigmoïd2.

/sigmoid/e	9'b000000000	9'b000010010		9'b011111111
/sigmoid/s	9'b000100000	9'b000101010		9'b001000000

FIGURE 3.16 – La simulation de la fonction sigmoïde2(x)

Il a donné la valeur  $\frac{1}{2}$  quand l'entrée était nulle et 1 quand l'entrée était dans la région de saturation.

La fonction sigmoïd2  $sig2(x) = f(x) + \frac{1}{2}$  est implémentée comme indiqué dans le diagramme suivant.

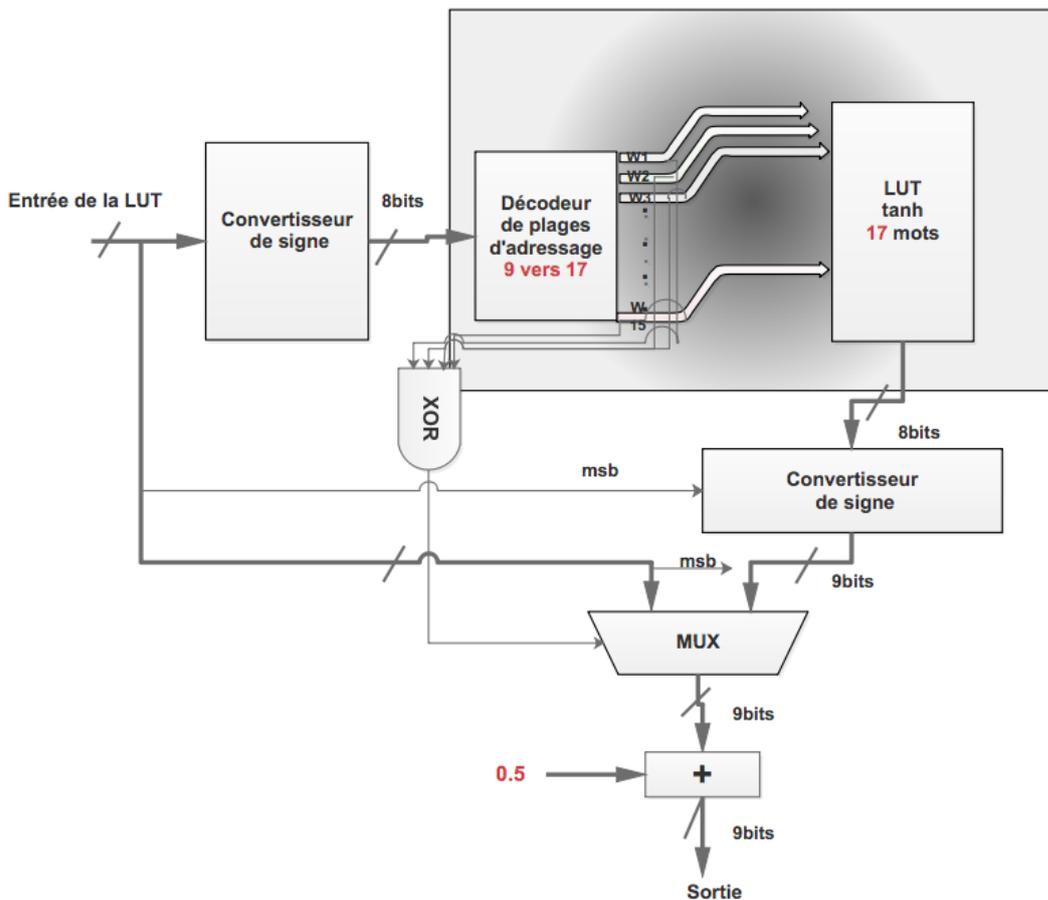


FIGURE 3.17 – Implémentation de la fonction tanh

## II Blocs fonctionnels du fonctionnement parallèle

Dans le fonctionnement parallèle, tous les RNAs créés fonctionnent simultanément, cela veut dire que si le nombre de RNA est égal à  $n$ , cette configuration sera générée  $n$  fois. Tous les éléments essentiels assurant le bon fonctionnement d'un seul RNA sont décrits dans cette partie.

### i.a RNA

Il y a plusieurs aspects à prendre en compte lors de la conception de circuits RNAs, ceux-ci comprennent la représentation des données, les produits, la somme des produits et les mises en œuvre des fonctions d'activation. Les plus importants sont les produits internes et la fonction d'activation non-linéaire déjà présentée ci-dessus.

#### i.a.1 Représentation des données

Un réseau de neurones fonctionne avec des nombres réels, il peut être représenté de plusieurs façons. Cependant, le problème est de savoir équilibrer

entre la précision numérique, la convergence et les ressources FPGA.

La représentation en virgule flottante de précision simple est idéale, car elle offre la plus grande précision (c'est-à-dire une erreur de quantification minimale). Cependant, l'utiliser dans l'implémentation RNA sur les FPGA n'est pas faisable, car il consomme d'énormes quantités de ressources qui sont très limitées dans ce cas, on vira comment on a pu le réaliser indirectement et d'une façon intelligente quand on présentera le bloc de normalisation.

Au lieu de la virgule flottante, la représentation du point fixe peut être utilisée à l'intérieur de RNA. Même si cela signifie moins de précision, ses avantages compromettent ses inconvénients. Il est plus efficace que la virgule flottante et beaucoup plus simple dans les opérations arithmétiques. Il est naturel d'utiliser la représentation du complément à deux pour les nombres négatifs. Il a été mentionné dans [10] que de nombreuses études ont prouvé que 16 bits pour le poids et 8 bits pour la fonction d'activation étaient suffisants. Dans l'architecture proposée à venir, 15 bits sont utilisés pour les poids et 9 bits pour l'entrée de la fonction d'activation.

### i.a.2 Produits internes

Les multiplicateurs ont été identifiés comme l'opérateur arithmétique le plus intensif en surface d'utilisation dans les RNA basés sur FPGA, il existe plusieurs formes de multiplicateurs.

On cite Multiplicateurs bit-série, dans celui-ci le calcul est fait par un bit à la fois, alors que les multiplicateurs entièrement parallèles calculent tous les bits simultanément. Ainsi, le premier peut être mis à l'échelle avec une représentation de signal de toute précision de gamme, cependant, cela signifie que le temps de multiplication en série de bits croît de manière quadratique, avec la longueur de la représentation du signal. Cela signifie qu'il n'est pas efficace dans les applications en temps réel.

D'autres manières ont été essayées, comme imposer des valeurs de puissances de deux des poids synaptiques, de sorte que les produits soient simplifiés à une instruction de décalage. Malheureusement, ce type de conception réduit dramatiquement la performance du RNA.

Une dernière méthode à utiliser est le multiplicateur direct à bits parallèles, Les FPGA ont un nombre limité de ces multiplicateurs dans différentes dimensions. Dans le Zynq-7000 de Zedboard, il ya que 220 multiplicateurs de dimension (18 × 25). Dans l'architecture proposée dans les sections suivantes, seuls 6 multiplicateurs ont été utilisés en parallèle, ce qui permet à la description VHDL générée par l'application C ++ proposée d'avoir à la fois un bon parallélisme et d'être implémentée dans les FPGA à ressources réduites.

### i.a.3 Somme des produits

Le nombre d'additionneurs n'est pas ce qui manque dans les FPGA d'aujourd'hui, et la somme des produits peut être transportée de plusieurs façons. En RNA la somme des produits est utilisée pour accumuler les produits élémentaires des entrées de neurones avec leurs poids synaptiques associés ( $\sum_{i=1}^n (W_{i,j} \times x_i)$ ).

La mise en œuvre de ces derniers dépend du nombre de produits élémentaires à accumuler. Si  $n$  est suffisamment petit, une implémentation directe comme celle de la figure 3.18 suivante peut être utilisée.

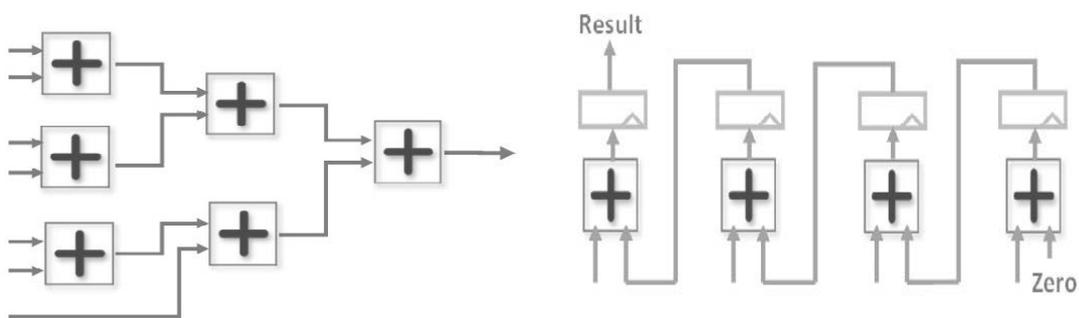


FIGURE 3.18 – Somme des produits

Comme il a été indiqué précédemment dans la mise en œuvre de RNA proposée seulement 6 produits peuvent être calculés en même temps, le multiplicateur reçoit les entrées associées du neurone et les multiplie par leur poids approprié d'une manière séquentielle, cette méthode provoque plus de latence dans le calcul, mais d'un autre côté, il est plus efficace en surface de consommation et préserve la caractéristique de parallélisme entre les neurones. Ainsi, les produits élémentaires pour chaque neurone peuvent être sommés en utilisant un seul additionneur avec une boucle de rétroaction puisqu'ils sortent séquentiellement des multiplicateurs, voir la Figure 3.19.

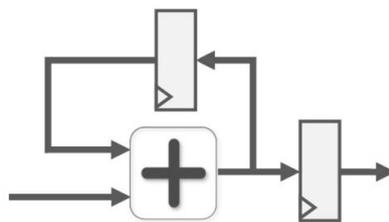


FIGURE 3.19 – Somme des produits séquentiels

### i.a.4 Bit-reducer

La largeur utilisée pour les poids dans ce travail est de 15 bits, 7 bits pour la partie entière et 8 bits pour la partie fractionnaire. Ainsi le produit du

poids (15 bits) avec l'entrée (9 bits) donne un résultat en 24 bits; ce résultat est ensuite transmis à l'additionneur qui donne aussi un résultat en 24 bits. Le problème est de comment adapter la largeur des données provenant de l'additionneur, à la taille de l'entrée du bloc de la fonction d'activation sans affecter les résultats. Nous avons donc introduit ce bloc pour faire cette adaptation.

A cause de la région de saturation dans la fonction tangente hyperbolique, la sortie du LUT pour toute valeur de ses entrées  $x$  qui vérifie  $|x| > 2.4$  est toujours 1. Ainsi, si la sortie de l'additionneur dépasse 2,4 le bit-reducer la change en une valeur qui peut être représentée en 9bits, et qui donne un résultat de 1 à la sortie du LUT.

Puisque l'entrée de la LUT a 3 bits pour la partie entière, alors si la *somme*  $> 2,4$  l'entrée de la LUT est mise à 3, et si la *somme*  $< -2,4$  alors l'entrée de la LUT sera mise à -4. C'est parce que 3 et -4 donnent une valeur de sortie égale à 1 et ils peuvent être représentés en 9 bits (3 bits pour la partie entière).

Dans la page suivante nous représenterons un schéma synoptique d'un seul neurone artificiel.

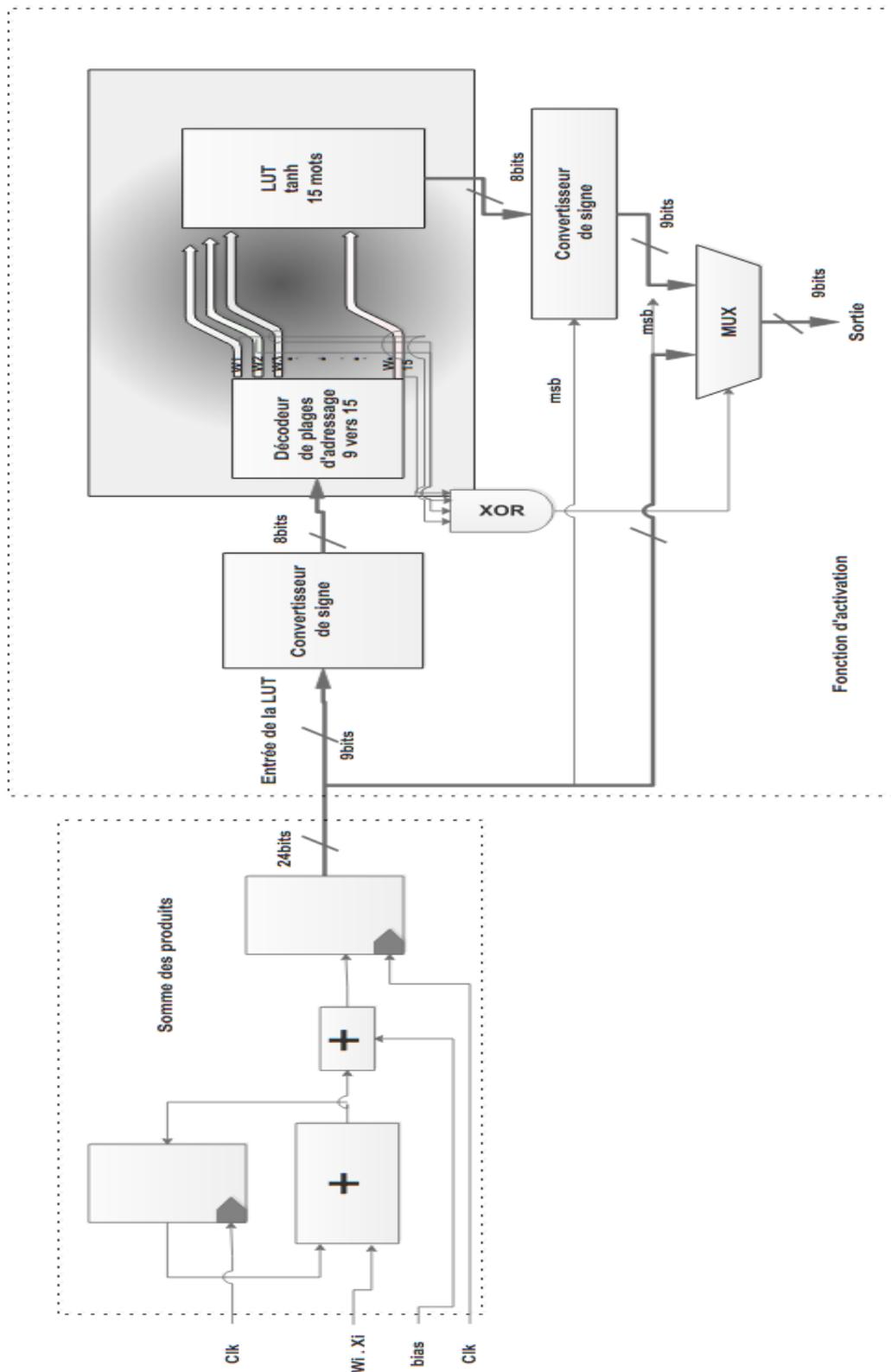


FIGURE 3.20 – Schéma synoptique d'un seul neurone artificiel

### i.a.5 Concept de multiplexage des couches

Pour l'approximation des sorties désirée, les réseaux multicouches se sont avérés très utiles, car ils sont similaires à un RN biologique. Cependant, les implémentations de réseaux multicouches exigeront d'énormes ressources et ne constitueront pas une solution réalisable pour des applications en temps réel telles que les estimateurs pour le contrôle moteur. Nous avons donc proposé une architecture basée sur le concept de multiplexage par couches présenté dans [24], où de grandes RNAs pourraient être implémentées avec un minimum de ressources.

Les données traitées dans un RNA feed-forward multicouche se propagent d'une couche à l'autre, et le calcul se passe dans une couche à la fois, donc nous n'avons pas besoin d'avoir toutes les couches implémentées en même temps, seulement la plus grande couche (avec le nombre maximum de neurones) devrait être implémenté, à laquelle on fait appelle de façon répétée, et donc elle se comporte comme des couches différentes à l'aide d'une unité de contrôle. Le bloc de contrôle assure le calcul complet de RNA en utilisant le multiplexage de couche par la gestion séquentielle des entrées, des poids, des biais et la valeur de sortie de la fonction d'activation (des LUT) appropriée de chaque couche.

Contrairement à l'architecture présentée dans [24], dans notre architecture, nous avons implémenté la couche la plus grande et la couche de sortie, ce qui signifie que la couche la plus grande se comporte comme les couches cachées et non la couche de sortie. Cela signifie que l'ont peut avoir des fonctions d'activation différentes dans la couche de sortie que celles qui se trouvent dans la couche cachée. Donc, la mise en œuvre d'un RNA comme celui de la figure 3.21.a est réduite à la mise en œuvre de l'RNA multiplexé en couches dans la figure 3.21.b.

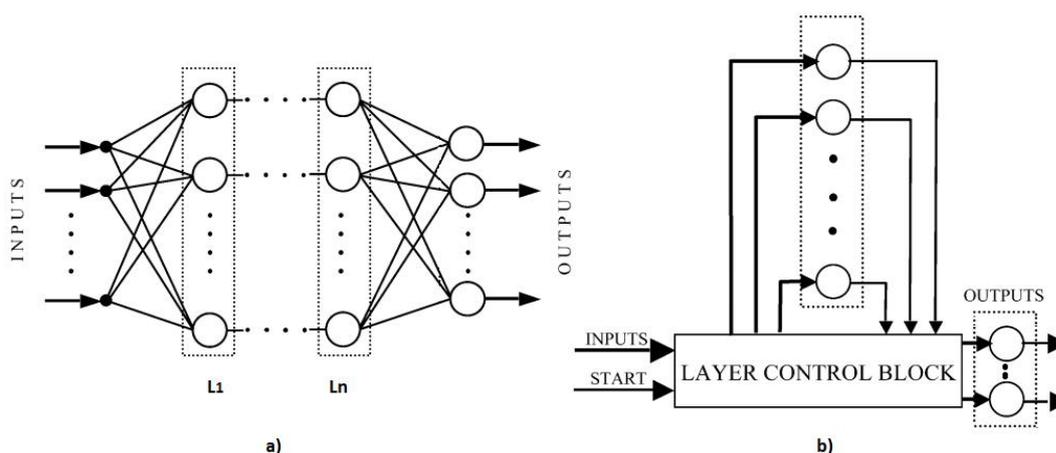


FIGURE 3.21 – Concept de multiplexage des couches[25]

La figure 3.20 représentée dans la page suivante met en évidence un RNA en fonctionnement parallèle.

## ii L'unité de contrôle

L'unité de contrôle synchronise tous les blocs et les données et assure que la couche implémentée se comporte comme toutes les couches cachées. Cependant, sa réalisation est compliquée car son fonctionnement dépendra entièrement de l'anatomie.

Ce bloc contient une machine d'états finis qui produit un signal pour synchroniser tout le système (RNA), et l'application qui le génère a besoin de fonctions C++ supplémentaires et de nombreuses conditions.

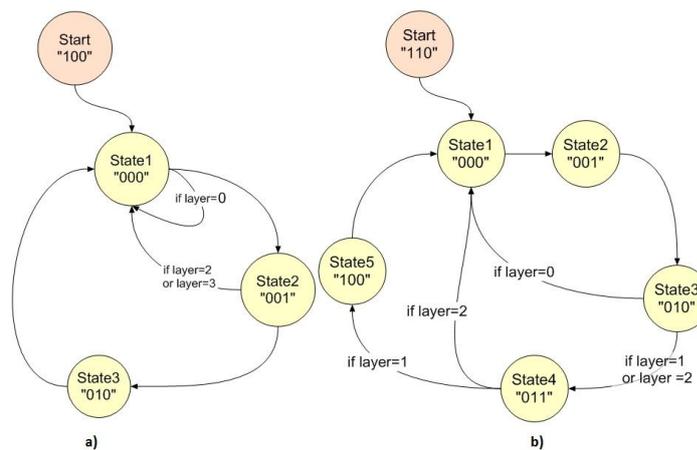


FIGURE 3.22 – État de flux de deux machines d'état

La figure ci-dessus montre le diagramme d'état (un graphe direct) de machines d'état pour deux réseaux neuronaux différents générés par l'application.

La première machine d'état (a) dans la figure 3.22 est pour un RNA qui a 3 couches plus une couche d'entrée contenant 3 entrées, la couche 1 a 1 neurone et les deux couches 2 et 3 ont 2 neurones.

La deuxième machine d'état (b) dans la figure 3.22 est pour un RNA qui a 2 couches plus une couche d'entrée qui contient aussi 3 entrées, la couche 1 a 5 neurones et la couche 2 a 4 neurones.

On peut voir clairement que les machines d'état des deux neurones sont totalement différentes.

## iii Les générateurs des biais et des poids

Ces blocs contiennent les biais et les poids des neurones de l'ensemble RNA. Les unités de contrôle synchronisent leurs sorties afin qu'elles donnent les bonnes valeurs des biais et des poids respectivement au moment approprié.

#### iv Le bloc de normalisation

Le bloc de normalisation permet de centraliser et de normaliser les entrées de RNA selon le principe expliqué dans le chapitre 2 section 2.6.1.

On a réalisé le bloc de normalisation de données en utilisant une stratégie qui rend les données dynamiques, une sorte de virgule flottante sans consommation en surface.

L'entrée de notre RNA est de 15 bits en complément à 2, si on fixe la taille de la partie entière et celle de la partie fractionnaire, on perdra à la fois en précision et en largeur de la plage des valeurs d'entrées possible.

L'astuce est d'utiliser que des fonctions  $C^{++}$  afin de pouvoir assigner pour chaque entrée d'un RNA la taille de la partie entière et celle de la partie fractionnaire la plus appropriée, ce qui nous donnera une précision de  $2e^{-14}$  et une dynamique de  $[-2e^{14}; 2e^{+14}]$  pour les entrées.

L'un des grands problèmes qu'on a rencontré est comment pouvoir extraire de l'information du VHDL dans la partie  $C^{++}$ , l'inverse est possible car c'est le VHDL qui est encapsulé dans le  $C^{++}$  et pas l'inverse, pour ceci on a eu l'idée d'ajouter des éléments non-gourmands en ressources tels que des simple mux pour pouvoir extraire l'information.

Un autre grand problème est la gestion de ce changement de taille pour chaque bloc où l'entrée est injectée car son changement va enchaîner pleins de désaccords dans le branchement de ces derniers.

#### v Le bloc de dé-normalisation

Le bloc de dé-normalisation permet de dé-normaliser les sorties de RNA selon le principe expliqué dans le chapitre 2 section 2.6.3. Son implémentation utilise le même principe que celui de normalisation

#### vi La bibliothèque "lib"

Elle contient la déclaration des différents types de signaux et des initialisations indispensables pour le fonctionnement du RNA

#### vii Le Top bloc

Chaque RNA fonctionnant en parallèle dispose de ce bloc qui permet de raccorder tous les blocs internes de ce dernier (bloc de normalisation, RNA, variateur de plage de fréquence, régulateur interne de fréquence et le générateur PWM).

Ce bloc est fortement lié au choix des blocs optionnels vus en section 3.5.2 que l'utilisateur voudra générer.

La figure 3.23 représente un RNA en fonctionnement parallèle.

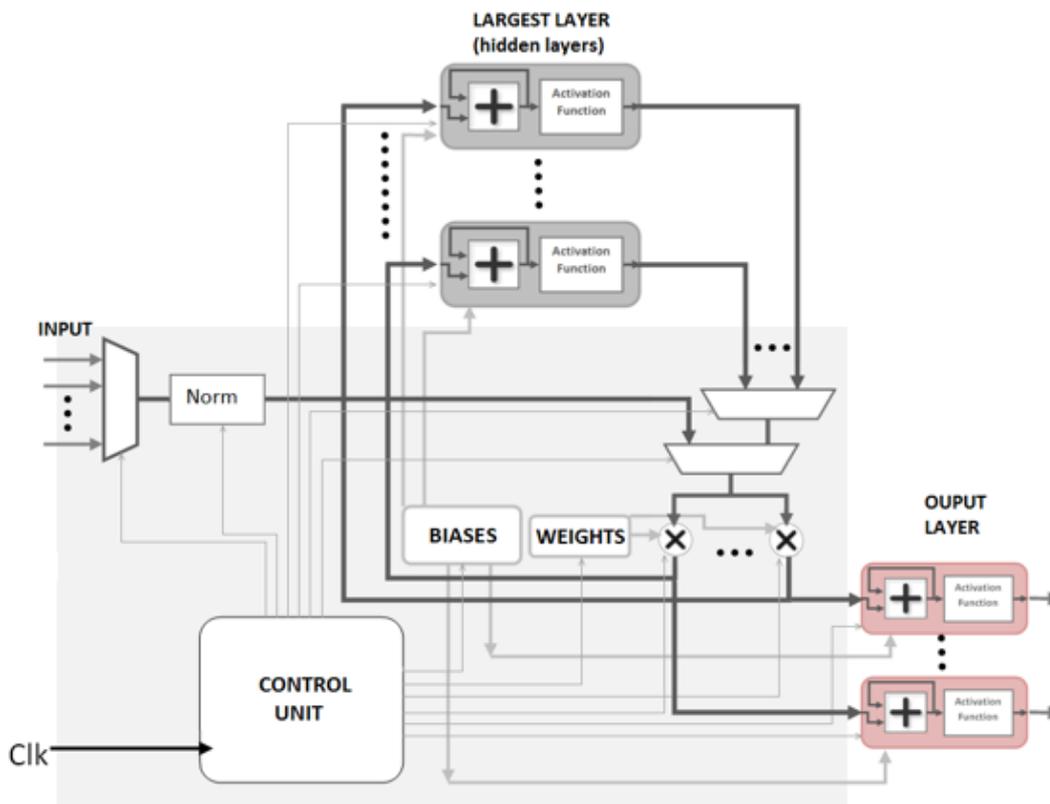


FIGURE 3.23 – Schéma synoptique d'un RNA en fonctionnement parallèle

### III Blocs fonctionnels du fonctionnement sérié

Ce fonctionnement est d'une utilité extrême dans pleines d'application et on en a bien expliqué ceci dans la section "solution originale" section 2.5.1.

Le principe de base est le suivant, si on a une architecture globale contenant une multitude de RNA qui fonctionnent dans le même environnement, mais pas forcément tous simultanément (c'est d'ailleurs le cas dans notre partie applicative.), alors la question qui se posera est comment créer un seul RNA d'une façon automatique qui pourra faire le travail de tous les RNAs, mais qui consommera en surface FPGA ce que consomme un seul ?.

L'idée ressemble à celle de multiplexage de couche, l'application C++ choisi le plus grand RNA (plus de couches et plus de neurones) comme neurone de base a implémenté et à gérer séquentiellement, ceci n'est pas toujours évidant, les différents RNAs doivent contenir certains critères de ressemblance

entre eux, nous avons traité quelques cas comme celui des RNA identiques mais à nombre de neurones de sortie différent, c'est de ces critères que l'application décide de faire l'implémentation ou pas.

Ce RNA disposera de toute la configuration exposée dans la section "Blocs Fonctionnels du fonctionnement parallèle" (le Top bloc, la bibliothèque "lib", le bloc de dé-normalisation, le bloc de normalisation, les générateurs des biais et des poids et le RNA parallèle) et aussi de deux autres blocs essentiels qui vont jouer le rôle de l'unité de contrôle de l'ensemble des RNAs, ces deux blocs sont.

### i.a Component Manager

On notera le RNA choisi par  $RNA_{choisi}$  dans la suite de ce document.

Ce bloc permet de gérer l'environnement externe de  $RNA_{choisi}$ , c'est le premier élément auquel on fait appel dans le Top bloc, chaque RNA interne<sup>9</sup> possède sa propre anatomie et sa représentation des données (la taille de la partie entière et fractionnaire), ce bloc va associer le bloc de normalisation approprié pour chaque RNA interne, un autre rôle important est la résolution du problème de la gestion des changements de taille pour chaque bloc où l'entrée est injectée car son changement va enchaîner pleins de désaccord dans le branchement de ces derniers, ceci est fait en adaptant chaque entrée à son format convenable, sachant qu'il y a deux types d'entrée, une sur 15 bits avec une représentation dynamique qui est l'entrée de RNA et l'autre type d'entrée représente les entrées qui sont injectées dans les blocs spécifique à l'application SHE PWM et qui ont des tailles variables. Un dernier rôle est le choix de nombre de sorties, car celui-ci varie pour chaque RNA.

### ii.b Neural Network Manager

Ce bloc permet de gérer l'environnement interne de  $RNA_{choisi}$ , c'est le premier élément auquel on fait appel dans l'architecture interne du RNA, chaque RNA interne<sup>10</sup> possède sa propre anatomie et ses propres poids et biais, le rôle de ce bloc et de gérer tous ces paramètres internes pour chaque RNA, il est important de noter que ce bloc est très sensible à la synchronisation des données et le chargement des données doit être impérativement fait au moment opportun sans avoir des latences.

---

9. L'un des RNAs concaténé à l'intérieur de  $RNA_{choisi}$ .

10. L'un des RNAs concaténé à l'intérieur de  $RNA_{choisi}$ .

La figure 3.24 représente un RNA en fonctionnement sérié.

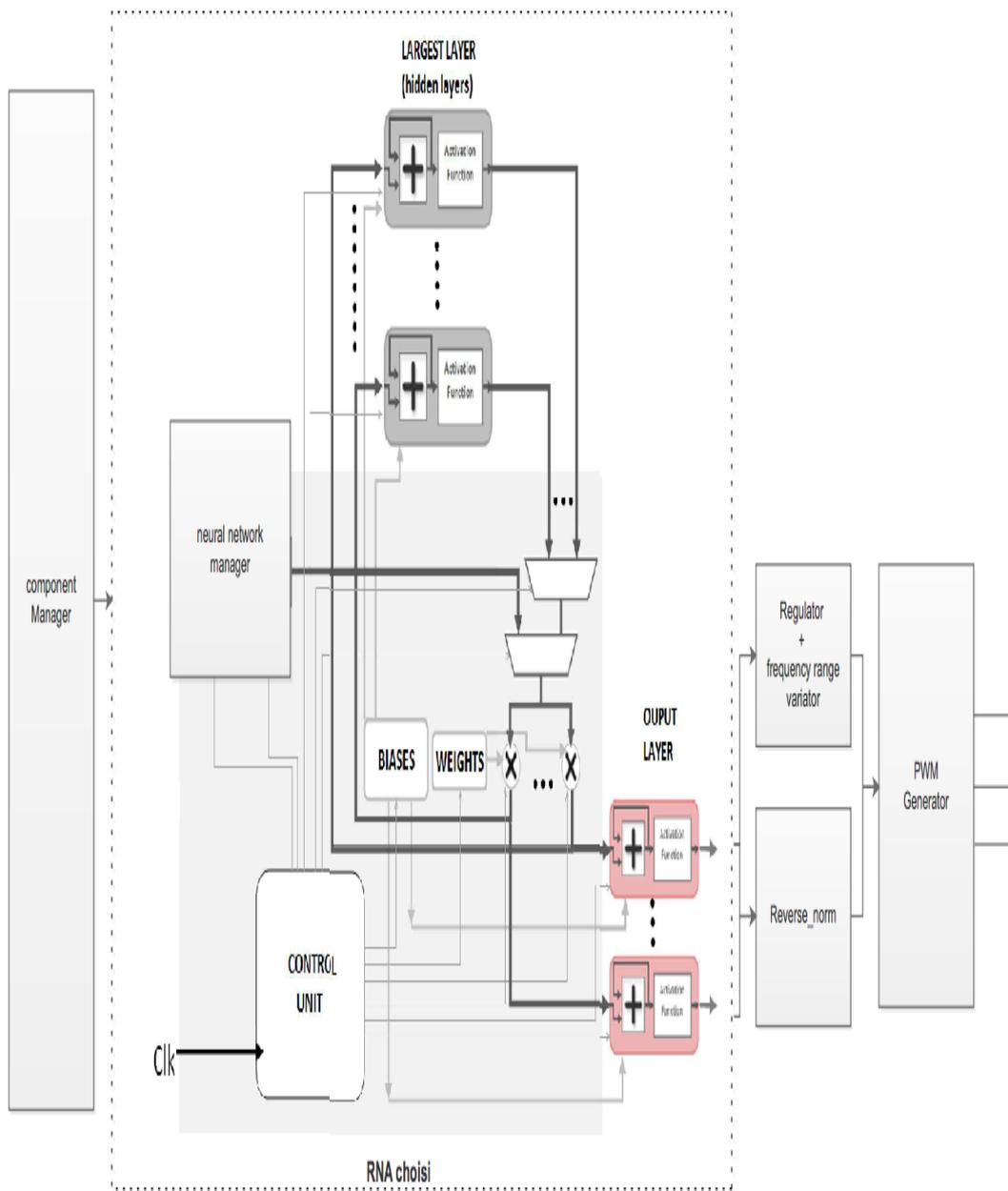


FIGURE 3.24 – Schéma synoptique d'un RNA en fonctionnement sérié

## 3.6 Simulation et validation expérimentale

### 3.6.1 Simulation

La figure suivante montrent les valeurs d'angles de commutation pour  $im=50\%$

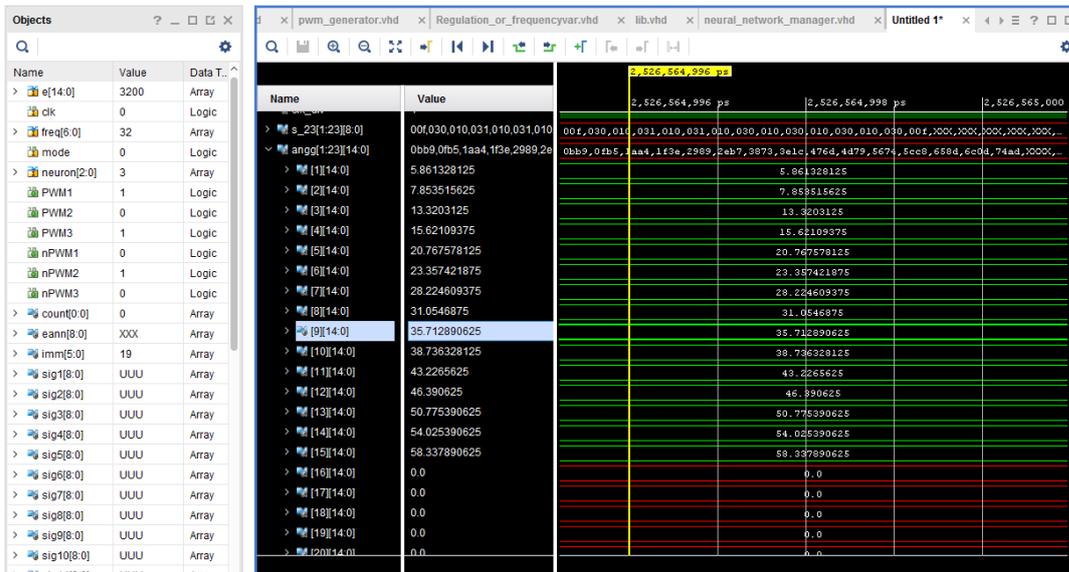


FIGURE 3.25 – Angles de commutation sur Vivado pour  $im=50\%$

### 3.6.2 Le banc d'essai

Dans le but de valider les résultats de simulation et d'implémentation FPGA et de tester l'efficacité de l'algorithme proposé, on l'a appliqué à la commande de la vitesse d'une machine asynchrone triphasée pilotée par un onduleur de tension triphasé comme expliqué dans le chapitre 1 section 1.2.1.



FIGURE 3.26 – Le banc d'essais

### 3.6.3 Résultats expérimentaux

Les figures 3.27 et 3.28 montrent la visualisation sur l'oscilloscope des tensions de phases de l'onduleur pour différentes valeurs de  $i_m$ . Les figures 3.29 et 3.30 montrent la visualisation des tensions entre phases de l'onduleur pour différentes valeurs de la tension d'alimentation de l'onduleur. Ainsi, pour valider l'efficacité de l'algorithme proposé par rapport à l'élimination des harmoniques sélectionnés, on a enregistré les tensions de phases de l'onduleur pour différentes valeurs de  $i_m$ , ensuite on a fait une analyse spectrale de ces tensions (figures 3.31, 3.32 et 3.33).

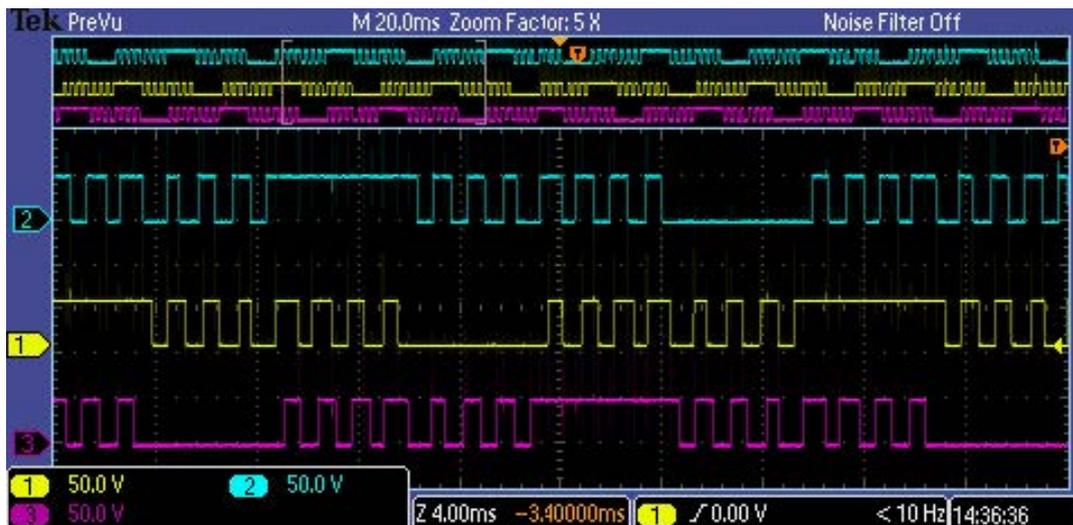


FIGURE 3.27 – Visualisation des tensions de phases de l'onduleur pour (a) :  $i_m=64\%$  ( $T=31.25\text{ms}$ )

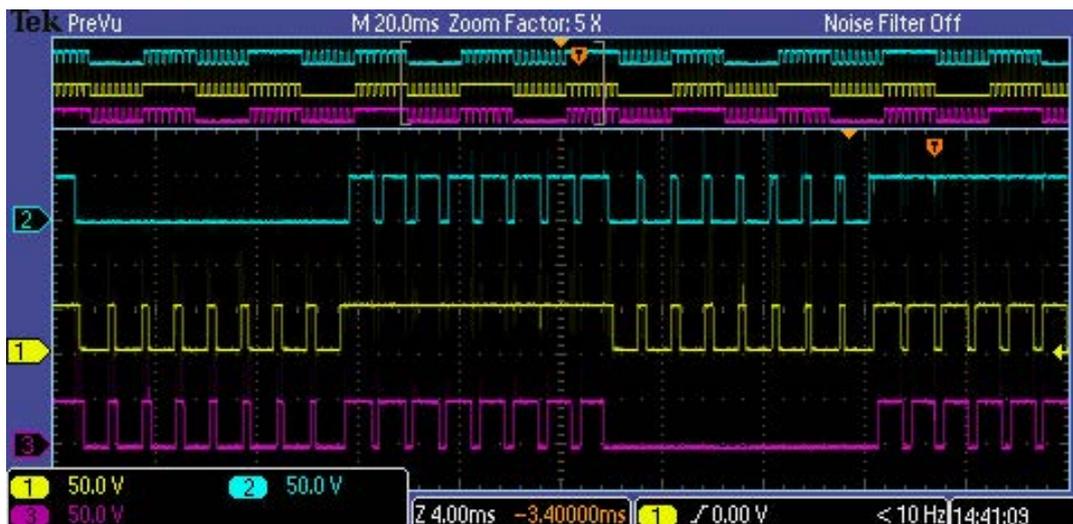


FIGURE 3.28 – Visualisation des tensions de phases de l'onduleur pour (b) :  $i_m=32\%$  ( $T=62.5\text{ms}$ )

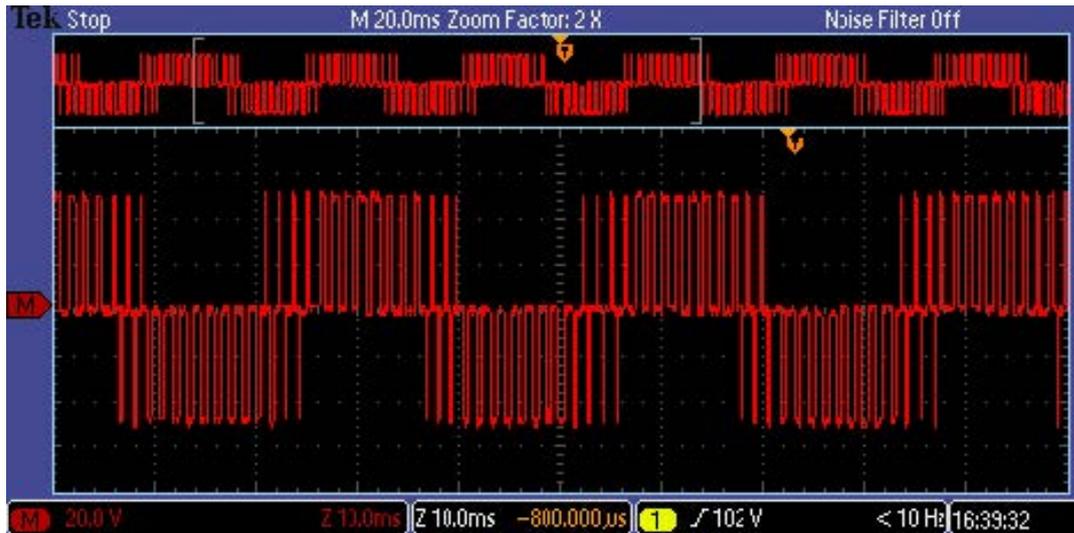


FIGURE 3.29 – Visualisation de la tension entre deux phases pour  $i_m=64\%$  ( $T=31.25$  ms), (a) : La tension d'entrée =50VDC

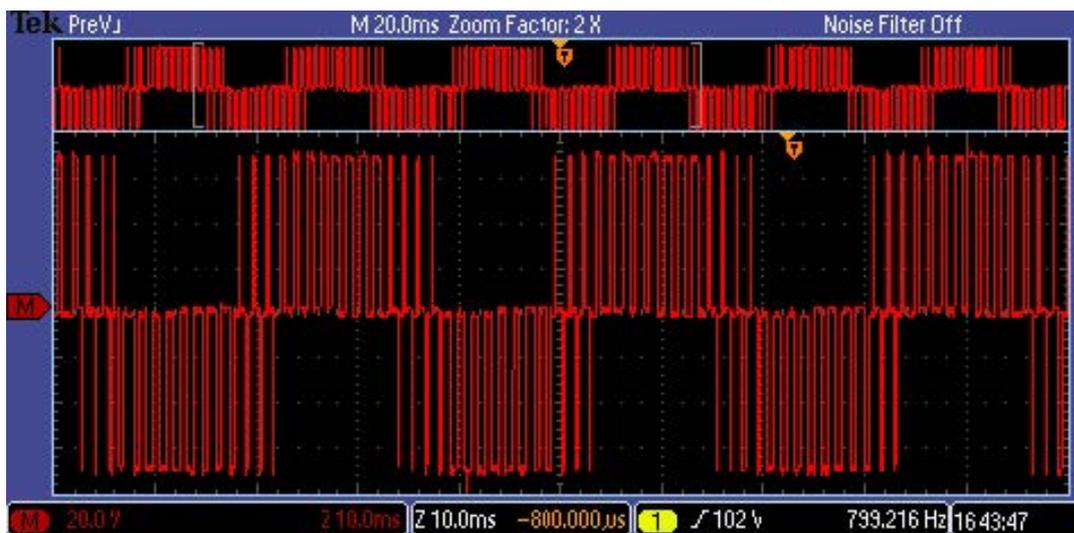


FIGURE 3.30 – Visualisation de la tension entre deux phases pour  $i_m=64\%$  ( $T=31.25$  ms), (b) : La tension d'entrée =70 VDC

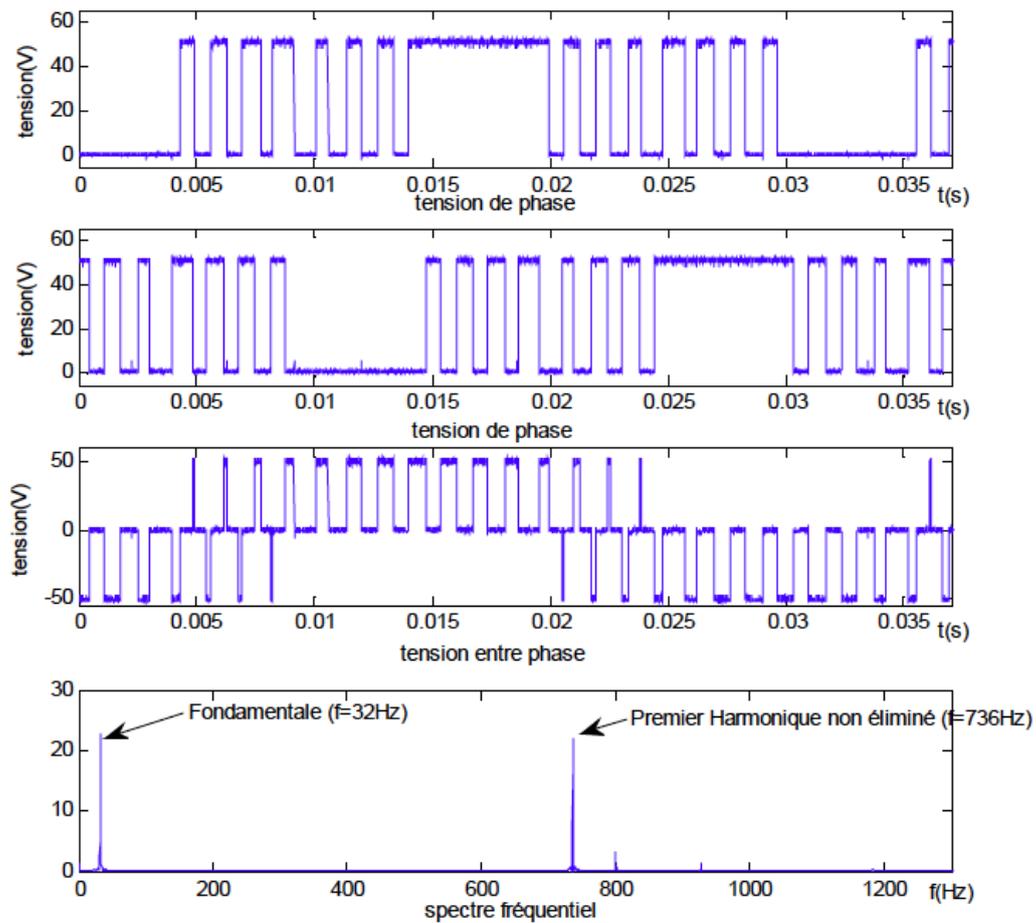


FIGURE 3.31 – Les tensions de phases, la tension entre phases correspondante et le spectre fréquentiel de cette tension entre phases pour une tension d'entrée de 50 VDC, (a) :  $\text{im}=64\%$  ( $T=31.25\text{ms}$ ).

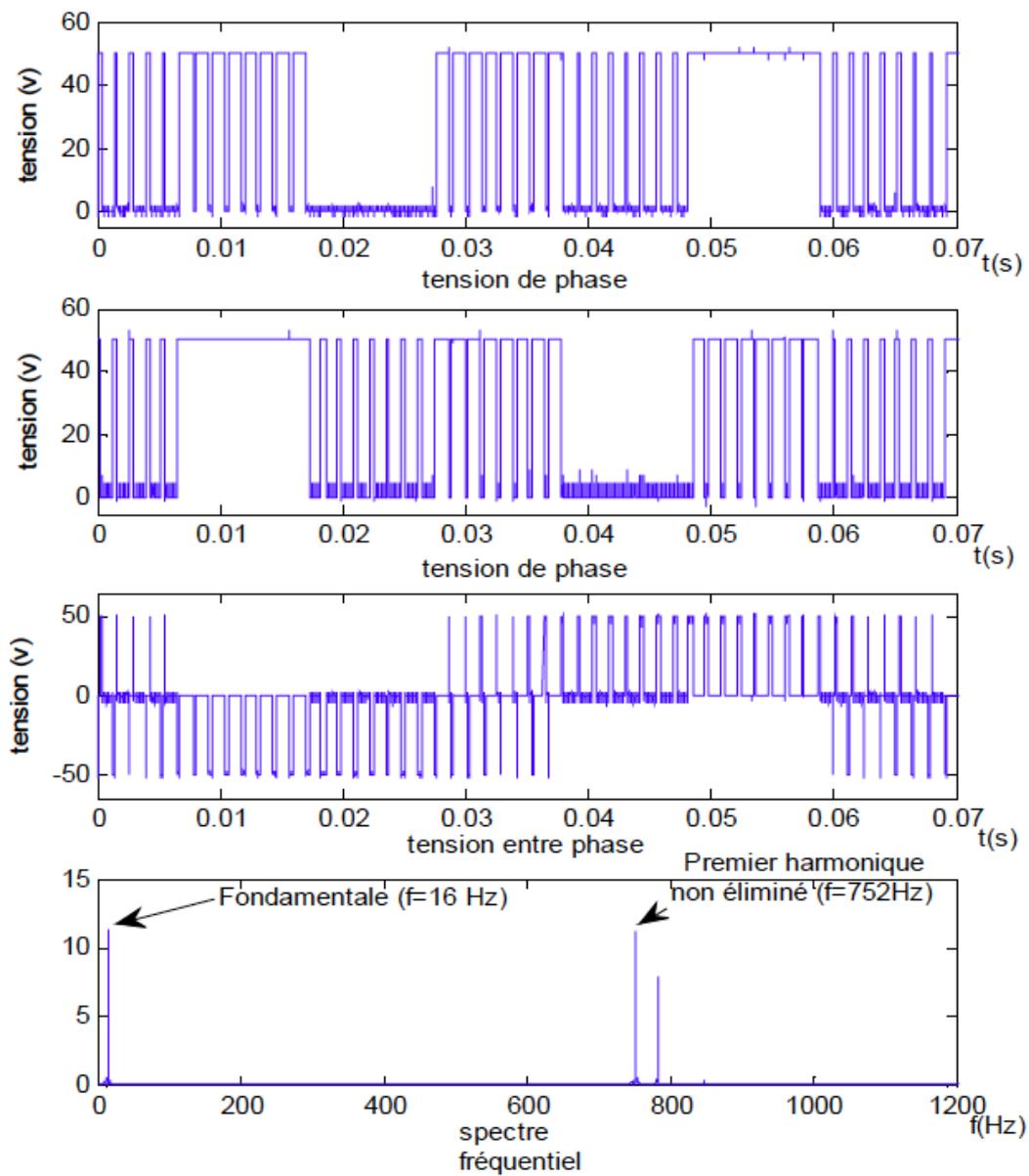


FIGURE 3.32 – Les tensions de phases, la tension entre phases correspondante et le spectre fréquentiel de cette tension entre phases pour une tension d’entrée de 50 VDC, (b) : $i_m=32\%$  ( $T=62.5\text{ms}$ ).

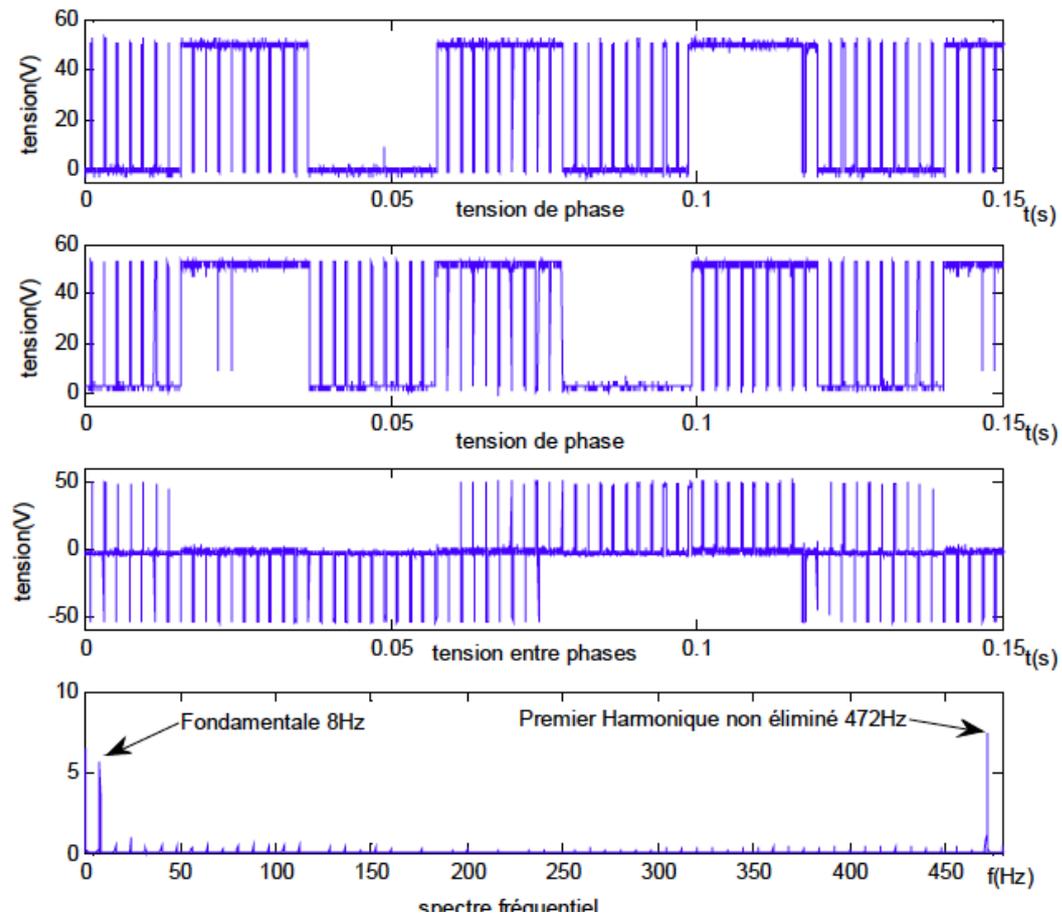


FIGURE 3.33 – Les tensions de phases, la tension entre phases correspondante et le spectre fréquentiel de cette tension entre phases pour une tension d'entrée de 50 VDC, (c) :  $i_m=16\%$  ( $T=125\text{ms}$ ).

## Interprétation

Les figures 3.27, 3.28, 3.29 et 3.30 montrent que les tensions de phases obtenues sont périodiques et de période 31.25 ms pour  $i_m = 64\%$ , 62.5ms pour  $i_m = 32\%$ , 125ms pour  $i_m=16\%$  et 250ms pour  $i_m=8\%$ . Ainsi, on voit clairement l'existence de sept, quinze, dix-neuf et vingt-trois angles de commutation par quart de période pour  $i_m=64\%$ ,  $i_m = 16\%$ ,  $i_m = 32\%$  et  $i_m = 8\%$  respectivement. De plus, de ces figures, on peut vérifier que les trois tensions de phases de l'onduleur sont déphasées de  $120^\circ$ .

De la figure 3.29 et 3.30 on peut vérifier que la tension entre phases varie entre "-E" et "+E", où "E" est l'alimentation continue de l'onduleur. Aussi, on constate que la période est égale à 31.25ms pour  $i_m=64\%$ .

L'analyse spectrale de la tension entre phases dans la figure 3.31 montre que le premier harmonique non éliminé pour  $i_m=64\%$  ( $f=32$  Hz et  $m=7$ ) est le 23ème ( $23*32=736$  Hz), c'est-à-dire les six premiers harmoniques (5, 7, 11, 13, 17 et 19) sont éliminés.

Aussi, l'analyse spectrale de la tension entre phases dans la Figure 3.32 montre que le premier harmonique non éliminé pour  $i_m=32\%$  ( $f=16$  Hz et  $m=15$ ) est le 47e ( $47*16=752$  Hz), c'est-à-dire les quatorze premiers harmoniques (5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37, 41 et 43) sont éliminés.

Ainsi, l'analyse spectrale de la tension entre phases dans la figure 3.33 montre que le premier harmonique non éliminé pour  $i_m=16\%$  ( $f=8$  Hz et  $m=19$ ) est le 59e ( $59*8=472$  Hz), c'est-à-dire les dix-huit premiers harmoniques (5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37, 41, 43, 47, 49, 53 et 55) sont éliminés.

L'analyse spectrale de la tension entre phases dans la Figure 5.23 (d) montre aussi que le premier harmonique non éliminé pour  $i_m=8\%$  ( $f=4$  Hz et  $m=23$ ) est le 71e ( $71*4=284$  Hz), c'est-à-dire les vingt-deux premiers harmoniques (5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37, 41, 43, 47, 49, 53, 55, 59, 61, 65 et 67) sont éliminés. De plus, les figures 3.27, 3.28, 3.29 et 3.30, montrent que les harmoniques multiples de trois sont éliminés automatiquement. De l'analyse spectrale, on voit bien que l'amplitude de la tension fondamentale augmente avec  $i_m$ .

En conséquence, les résultats expérimentaux confirment l'efficacité et la précision de notre implémentation SHE PWM basée sur les RNAs en fonctionnement série pour l'élimination des harmoniques sélectionnés et dans le contrôle de la tension fondamentale.

### 3.7 Conclusion

Dans ce chapitre, nous avons vu en détails ce qu'il y a dans la boîte noire RNA, nous avons expliqué toutes les étapes nécessaires pour la mise en œuvre de l'application C<sup>++</sup>, mais aussi le principe sa réalisation, de plus nous avons prouvé l'efficacité et l'intelligence de cette dernière. Nous avons présenté les étapes d'implémentation des différents blocs sur le circuit FPGA. Nous avons insisté sur l'optimisation de cette implémentation vu l'importance de cette dernière et son impact sur les performances de notre système.

Enfin, dans ce chapitre, on a validé l'efficacité et la précision de l'algorithme proposé par une application industrielle (SHE PWM) et par des essais expérimentaux qui confirment son efficacité et sa précision dans l'élimination des harmoniques sélectionnés et dans le contrôle de la tension fondamentale.

## Conclusion générale

La majorité des problèmes de la recherche scientifique, pour ne pas dire tous se traduisent toujours par un modèle mathématique, sous forme d'un ensemble d'équations algébriques ou différentielles généralement non-linéaires. Pour résoudre le système les chercheurs font appel à la machine en général par deux méthodes conventionnelles qui sont la méthode purement analytique et la méthode numérique. Ces deux choix peuvent être excellents dans les études théoriques, mais quand on passe à la pratique, c'est toute une autre histoire. Dans la pratique, il y a beaucoup d'autres contraintes qu'il faut prendre en considération notamment le temps de calcul, la précision et le coût. Une méthode moderne pouvant résoudre ces problèmes mathématiques tout en contrôlant ces contraintes est les réseaux de neurones artificiels. Cependant, les chercheurs fuissent cette méthode pour deux raisons essentielles, la première est sa nécessité à une implémentation hardware afin qu'elle profite de son point fort qui est le parallélisme, toutefois les domaines de la recherche n'ont pas tous une base en électronique et plus précisément un background en description matérielle. La deuxième raison est le mystère des réseaux de neurones, l'élaboration de la topologie du réseau neuronale pour un problème est un "art noir", dans certains problèmes scientifiques complexes, il pourrait y avoir des années d'expérience et d'intuition afin de réaliser une topologie fonctionnelle.

Dans ce travail, nous avons donné des solutions originales pour ces deux problèmes, donnant enfin une chance à tous ces domaines de la recherches qui n'ont pas forcément le profil d'un électronicien leur permettant de réaliser eux même leur solution basée sur les RNAs et de profiter de la puissance de ces derniers.

La première solution est une application C++ intelligente pour l'implémentation d'un réseau multi-RNA en deux types de fonctionnement "série et parallèle" et avec une topologie flexible. Nous avons tout fait pour rendre l'application très simple à l'utilisation, pour que tous les domaines de la recherche puissent en profiter. La sortie de cette application est un ensemble de fichiers source des descriptions VHDL des différents blocs prêts à l'emploi, il suffit de les charger dans l'environnement source d'un outil de développement tel que Vivado de Xilinx et d'ajouter le fichier de contrainte spécifique à la carte cible utilisée et avec un simple clic le circuit final est implémenté. Il est important de noter que notre but n'était pas seulement réaliser un RNA fonctionnel, on a donné beaucoup du temps pour le rendre configurable pour n'importe quelle topologie choisie par l'utilisateur et avec une architecture aussi optimisée au sein d'un seul RNA, en utilisant le multiplexage de couches et dans l'ensemble des RNAs en utilisant le mode de

fonctionnement série.

La deuxième solution consiste à donner une nouvelle méthode pour l'opérationnel afin qu'il puisse élaborer une topologie du réseau neuronale. Les réseaux de neurones ne fournissent pas les explications concernant leurs résultats, ce qui limite donc l'analyse des phénomènes existants. Cette limite est due à l'opacité des réseaux de neurones qui empêche une analyse pertinente des solutions obtenues. Ce que nous avons fait, c'est plutôt analyser la source qui est l'origine de tous les problèmes notamment la divergence de la phase d'apprentissage. Nous avons donné les étapes et les procédures à faire afin d'élaborer une topologie fonctionnelle avec l'architecture neuronale la plus simple possible ce qui en résulte une consommation plus faible en ressources FPGA .

Nous avons pris le temps de réaliser dans ce travail une partie applicative qui consiste d'une application purement industrielle (SHE PWM) pour valider notre application C<sup>++</sup>. La commande en tension et en fréquences variables d'un onduleur triphasé destiné pour commander une machine asynchrone dans un véhicule électrique a été réalisée par la proposition d'un nouvel algorithme basé sur la théorie des réseaux de neurones artificiels et la commande SHE PWM en v/f constant. On a même développé des blocs généralisés (configurables) qui permettent de découpler la fréquence et l'amplitude du fondamental. Cet algorithme a été validé par simulation, puis par une implémentation sur un circuit FPGA et enfin par des essais pratiques pour la commande de vitesse d'un moteur asynchrone.

Enfin, les résultats d'implémentation finale sur un circuit FPGA montrent aussi que le calcul des instants de commutation et la génération des signaux PWM sont en temps réel (les temps de commutation sont en 62.5 ns) et avec une très bonne précision. Les trois signaux de commande sont générés parallèlement et sont indépendants l'un par rapport à l'autre. Les résultats des essais pratiques, montrent aussi l'élimination effective des harmoniques sélectionnés et la précision du contrôle de la tension fondamentale à la sortie de l'onduleur.

En perspective à ce travail, nous prévoyons de continuer à travailler sur notre application basée sur C ++, pour ajouter des options et des fonctionnalités supplémentaires telles que rendre l'application capable de prendre encore plus de décision, car il y a une infinité de choix de topologies et par conséquent une infinité de cas et de critère à prendre en considération afin de pouvoir concaténer un ensemble de RNAs dans un seul. Nous envisageons aussi de rendre la performance paramétrable en donnant la possibilité à l'utilisateur de choisir le nombre de produits à utilisé, ce qui permet de maîtriser la précision et les ressources. Enfin, mettre en œuvre d'autres types de réseaux de neurones en couches pour offrir plus de flexibilité et de diversité de la topologie des RNAs.

# Bibliographie

- [1] A.DAS, Emadi K.Sivakumar K.GOPAKUMAR et C.Patel R.RAMCHAND. « A Pulse width Modulated Control of Induction Motor Drive Using Multilevel 12-Sided Polygonal Voltage Space Vectors ». In : *IEEE Transactions On Industrial Electronics* (2009), 56 (7) : pp. 2441–2449.
- [2] A.GUELLAL. « Contribution à l'étude et à l'implémentation des commandes en temps réel pour MAS ». Thèse de doct. Electronique, Ecole Nationale Polytechnique, Alger, 2015.
- [3] A.GUELLAL, C.LARBES et L.HASSAINE. « A new on-line Pulse Width Modulation control based on the principle of neural networks ». In : *The First International Conference on Electrical Energy and Systems ICEES* (2013), 133 (2) : pp. 85–94.
- [4] Hodgkin AL et Huxley AF. « Une description quantitative du courant membranaire et de son application à la conduction et à l'excitation dans le nerf ». In : (1952).
- [5] Abdul Moeed AMJAD, Zainal Salam Ahmed SAIF et Ahmed MAJED. « Application of differential evolution for cascaded multilevel VSI with harmonics elimination PWM switching ». In : *Electrical Power and Energy Systems* (2015), 64 (2015) 447–456.
- [6] Association AVEM. *avem*. <http://www.avem.fr>. 2018.
- [7] L. BAGHLI. *Modélisation et commande de la machine asynchrone*. IUFM de Lorraine, UHP. 2005.
- [8] B.OZPINECI, Z.Du J.N.CHIASSON et L.M.TOLBERT. « DC–AC Cascaded H-Bridge Multilevel Boost Inverter with no Inductors for Electric/Hybrid Electric Vehicle Applications ». In : *IEEE Transactions on Industry Applications* (2009), 45 (3) : pp. 936–970.
- [9] Bureau de dépôt BRUXELLES X. *Véhicules électriques*. <http://www.febiac.be>. 2011.
- [10] Neurobranches. CHEZ-ALICE. *LE NEURONE - UNITÉ FONCTIONNELLE DU SYSTÈME NERVEUX*. <http://neurobranches.chez-alice.fr>. 2000-06-17.
- [11] C.H.RIVETTA, A.Emadi G.A.Williamson R.JAYABALAN et B.FAHIMI. « Analysis and Control of a Buck DC–DC Converter Operating With Constant Power Load in Sea and Undersea Vehicles ». In : *IEEE Transactions On Industry Applications* (2006), 42 (2) : pp. 559–572.
- [12] M. CRISTEA et A. DINU. « A new neural network approach to induction motor speed control ». In : (2001), vol. 2, pp. 784–788.
- [13] M.S.A. DAHIDAH et V.G. AGELIDIS. « Selective Harmonic Elimination PWM Control for Cascaded Multilevel Voltage Source Converters : A Generalized Formula ». In : (2008), 23 (4) : pp.1620–1630.

- [14] Ali EMADI, Alireza Khaligh SHELDON et S.WILLIAMSON. « Power Electronics Intensive Solutions for Advanced Electric, Hybrid Electric, and Fuel Cell Vehicular Power Systems ». In : *IEEE Transactions on Power Electronics* (2006), 21(3) : pp. 567–577.
- [15] Amar GUELLAL, Cherif Larbes Linda Hassaine Douadi BENDIB et Ali MALEK. « FPGA based online Artificial Neural Network Selective Harmonic Elimination PWM technique ». In : *Electrical Power and Energy Systems* (2015), pp. 33–43.
- [16] Simon HAYKIN. *Neural Networks - A Comprehensive Foundation, Second Edition*. Prentice Hall, 1998.
- [17] Jeff HEATON. *Introduction to Neural Networks with C*. Ed :Heaton Research Inc WordsRU.com, 2008.
- [18] J.A.TAUFIQ, B.MELLITT et C.J.GOODMAN. « Novel Algorithm for Generating Near Optimal PWM Waveforms for AC Traction Drives ». In : *Electric Power Applications, IEE Proceedings* (1986), 133 (2) : pp. 85–94.
- [19] MURRAY L. BARR et JOHN A. KIERNAN. *The Human Nervous System P35. Anatomical Viewpoint*. Fifth Edition. Harper International, 1988.
- [20] Kiyoshi KAWAGUCHI. *Artificial Neural Networks*. <http://www.ece.utep.edu>. 2000-06-17.
- [21] SAADI KHALID et OUADRIA Anes ABDERRAHIM. *Implementation of Artificial Neural on an FPGA board Application on Induction Motor speed control*. Ecole nationale Polytechnique Algiers Algeria, 2017.
- [22] Adel KHEDHER et Mohamed Faouzi MIMOUNI. « Sensorless-adaptive DTC of double star induction motor ». In : *Energy Conversion and Management* (2010), 51 : pp. 2878–2892.
- [23] Alinaghi MARZOUGH, Hossein IMANEINI et Amirhossein MOEINI. « An optimal selective harmonic mitigation technique for high power converters ». In : *International Journal of Electrical Power and Energy Systems* (2013), 49 : pp. 34–39.
- [24] Alinaghi MARZOUGH, Hossein IMANEINI et Amirhossein MOEINI. « An optimal selective harmonic mitigation technique for high power converters ». In : *International Journal of Electrical Power and Energy Systems* (2013), 49 : pp. 34–39.
- [25] Pramod Kumar MEHER. « An Optimized Lookup-Table for the Evaluation of Sigmoid Function for Artificial Neural Networks ». In : (2010), IConf, pp. 91–95,
- [26] WEI MI. « Extraction de paramètre et domaine de validité du modèle d'un composant de puissance ». Thèse de doct. l'institut National des sciences Appliquées de Lyon, 2002.
- [27] Karl Leboeuf Roberto MUSCEDERE, Huapeng Wu Majid AHMADI et Ashkan Hosseinzadeh NAMIN. « Efficient hardware implementation of the hyperbolic tangent sigmoid function ». In : (2009), pp. 2117–2120.
- [28] IZBOUDJEN NOUMA. « Plateforme pour l'Implémentation des Réseaux de Neurones sur FPGA : Application à l' Algorithme de la Rétro Propagation du Gradient ( RPG ) ». Thèse de doct. Ecole nationale Polytechnique Algiers Algeria Thèse de Doctorat, 2014.
- [29] Akure ONDO et Okitipupa ONDO. « Comparative study of biological and artificial neural networks O.S. Eluyode and Dipo Theophilus Akomolafe ». In : (2013), 23 (4) : pp.1620–1630.

- [30] Hasmukh S. PATEL et Richard G. « Generalized Techniques of Harmonic Elimination and Voltage Control in Thyristor Inverters : Part II-Voltage Control Techniques ». In : *Transactions on Industry Applications* (1974), IA-10 (5) : pp. 666–673.
- [31] Hasmukh S. PATEL et Richard G. HOFTI. « Generalized Techniques of Harmonic Elimination and Voltage Control in Thyristor Inverters : Part I-Harmonic Elimination. » In : *IEEE Transactions on Industry Applications* (1973), IA-9 (3) : pp. 310–317.
- [32] Jagath C. RAJAPAKSE et Amos R. OMONDI. *FPGA Implementations of Neural Networks*. Springer, Ed.
- [33] L. M. REYNERI. « Implementation issues of neuro-fuzzy hardware : Going towards HW/SW codesign ». In : (2003), vol.14, no. 1, pp. 176–194.
- [34] Margaret ROUSE. *IP core (intellectual property core)*. whatis.techtarget.com. 2011.
- [35] Mohamed S.A.DAHIDAH, Georgios KONSTANTINOU et Vassilios G.AGELIDIS. « A Review of Multilevel Selective Harmonic Elimination PWM : Formulations, Solving Algorithms, Implementation and Applications ». In : *IEEE Transactions On Power Electronics* (2015), Vol. 30, NO. 8 pp. 4091–4105.
- [36] Zainal SALAM. « An On-Line Harmonic Elimination Pulse Width Modulation Scheme for Voltage Source Inverter ». In : *Journal of Power Electronics* (2010), 10(1) : pp. 43–50.
- [37] Zainal SALAM et N. BAHARI. « Joint International Conference on Power Electronics ». In : (2010).
- [38] STANFORD. *Neural Networks*. <https://cs.stanford.edu/people/eroberts>.
- [39] Een Shing WANG et Che Wei LIN. « A digital circuit design of hyperbolic tangent sigmoid function for neural networks ». In : (2008), pp.856–859.
- [40] WIKIPEDIA. *Synapse*. <https://fr.wikipedia.org/wiki/Synapse>. 2018.
- [41] Z.DU et L.M.TOLBERT. « Active Harmonic Elimination for Multilevel Converters ». In : *Transactions on Power Electronics* (2006), 21 (2) : pp. 459–496.
- [42] ANDREAS ZELL. *Simulation Neuronaler Netze P24*. Addison-Wesley, 1994.

## Déclaration d'Authenticité

Je, soussigné , certifie que le projet fin d'étude intitulé « Générateur intelligent de multi-réseaux neuronaux artificiels » et le travail qui y est présenté est le mien. Je confirme que :

- Ce travail a été élaboré principalement lors de la candidature pour un diplôme d'ingénieur d'état à cette école.
- Quand une partie de ce PFE a déjà été soumise pour un diplôme ou toute autre qualification à cette école ou dans une autre institution, cela a été clairement indiqué.
- Lorsque j'ai consulté le travail publié par d'autres auteurs, cela est toujours clairement indiqué.
- À l'exception de ces citations, ce PFE est un travail exclusivement personnel.
- J'ai reconnu toutes les principales sources d'aide.
- Lorsque ce PFE est basée sur un travail effectué par moi-même conjointement avec d'autres, j'ai clairement précisé ce qui a été fait par les autres et ma contribution personnelle.

Fait le : 24 juin 2018

---