REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Ecole Nationale Supérieure Polytechnique
Département d'électronique
Laboratoire **D**ispositifs de **C**ommunication et de **C**onversion **P**hotovoltaïque

Mémoire de Magister en Electronique
Ecole Doctorale : Energies Renouvelables
Option : Photovoltaïque

Présenté par :
GUELLAL Amar
Ingénieur d'Etat en Electronique, ENP

Thème

Implémentation sur FPGA d'une commande MLI on-line basée sur le principe des réseaux de neurones

Membres du Jury

M HADDADI Mourad, Pr	Président, ENSP
M LARBES Chérif, MC	Rapporteur, ENSP
M AIT CHEIKH Med Salah, MC	Examinateur, ENSP
M MALEK Ali, MR	Examinateur, CDER
M TAGHI Mohand Oussaid, CC	Examinateur, ENSP

Année universitaire 2008/2009

Dédicace

Je dédie ce travail à :

Mes parents,

Mes frères et mes soeurs,

Toute ma famille,

Ainsi qu'à tout mes amis.

Ahmed Wid Aissa

Remerciement

Je remercie le bon Dieu de m'avoir donné la volonté et la patience qui m'ont permis de mener à bien ce travail.

Je remercie vivement mon Promoteur, Monsieur LARBES, pour son suivi, son aide documentaire et son soutien matériel précieux.

Je tiens à adresser mes sincères remerciements à tous les membres du Jury chargé d'examiner la soutenance de mon projet de fin d'étude..

Je voudrai exprimer mon profond respect à tous les Enseignants qui m'ont encadré durant mon étude.

Je ne saurai oublier de remercier toute personne qui, d'une manière ou d'une autre, m'a aidé dans l'élaboration de ce travail.

بسم الله الرحمن الرحيم

ملخص:

إن تقنية تغير عرض الدفع (MLI) "المبرمجة" التي وضعها كل من باتل و هوفت تمثل حلا ممتـــازا للتحكم في سرعة المحرك اللامتزامن.لكنها لم تلق انتشارا واسعا و ذلك بسبب استحالة حساب زوايا التبديل في نفس الوقت الذي يشتغل فيه المحرك.الهدف من هذا المشروع هو برمجة تقنية جديدة على بطاقة "FPGA ",هذه التقنية تعتمد على طريقة الشبكات العصبية , و تحسب مباشرة زوايا التبديل مع اشتغال المحرك و بدقة متساوية مع دقة تقنية باتل و هوفت,هذه الدقة تسمح بحذف المركبات التوافقية الأولى المختارة من أجل الحذف, و ضبط المركبة الأساسية.

كلمات مفاتيح: بطاقة "FPGA " , "VHDL ", "MLI ", المحرك اللامتزامن, الشـــبكات العصـــبية ,

Résumé :

La technique de modulation de largeur d'impulsion (MLI) calculée avec élimination d'harmonique et asservissement du fondamentale (HEVC) de Patel et Hoft, est une alternative très attirante pour la commande de vitesse du moteur asynchrone triphasé. Mais son utilisation est *limitée* par le fait que les angles de commutation ne peuvent être calculés en fonctionnement dans des applications temps réel. Le but de ce mémoire est d'implémenter sur un circuit FPGA un *nouvel algorithme MLI temps réel* basé sur le principe de MLI (HEVC) et les réseaux de neurones. Cet algorithme présente une précision de calcul pratiquement égale à celle de la technique de Patel et Hoft permettant ainsi une *élimination* des premiers harmoniques sélectionnés et un asservissement du fondamental.

Mots clés : FPGA, VHDL, MLI, moteur asynchrone, réseaux de neurones, temps réel, harmonique, Patel, Hoft.

Abstract :

The calculated pulse width modulation (PWM) technique of Harmonic Elimination and Voltage Control (HEVC) is an attractive alternative for speed control of an induction motor. However, its application is limited by the fact that the switching angles cannot be calculated online in real-time applications. The aim of this project is the implementation of new online PWM algorithm based on the HEVC PWM and the neural network system. This latter gives the same accuracy as Patel and Hoft's algorithm leading to the elimination of the low order harmonics and to the control of the fundamental voltage.

Keywords: FPGA, VHDL, PWM, simulation, on-line, neural network, induction motor, Patel, Hoft.

Sommaire

Liste des figures

Liste des tableaux

Introduction

Introduction

Dans le domaine industriel, la vitesse variable est assurée généralement par le moteur à courant continu. Mais celui-ci présente plusieurs inconvénients : coût élevé, présence d'un collecteur et entretien fréquent. Pour éviter ces inconvénients, on utilise de plus en plus, dans la gamme des faibles et moyennes puissances, le moteur asynchrone triphasé d'induction à cage qui présente des avantages intéressants : coût réduit, robustesse, simplicité, entretien réduit et encombrement faible.

Pour obtenir une vitesse variable, ce moteur est alimenté à partir d'un onduleur MLI (Modulation en Largeur d'Impulsions). La sortie de ce dernier est variable en tension et en fréquence avec le rapport V/f constant pour maintenir le flux constant. Vu la présence d'*harmoniques* dans le signal de sortie dans ce genre d'onduleur, le fonctionnement du moteur sera accompagné de *pulsations de couple* et d'*échauffement* de celui-ci à cause des pertes. Les pulsations de couple sont très gênantes, surtout aux faibles vitesses. Pour pallier à ces inconvénients, on doit donc éliminer correctement les harmoniques. Autrement dit, on doit calculer les angles de commutation du signal MLI avec une plus grande précision.

Le but de notre mémoire est d'implémenter sur un circuit FPGA une commande MLI *'on-line'* avec asservissement du fondamental et élimination sélective d'harmonique. Le principe de cette commande est inspiré de la technique de Patel et Hoft [1] mais les angles de commutation sont calculés en se basant sur le principe des réseaux de neurone.
Ainsi notre mémoire est organisé en quatre chapitres :

Le premier chapitre introduit des généralités, il présente une vue d'ensemble sur les machines asynchrones, les onduleurs de tension et les techniques de commande MLI.

Au deuxième chapitre, nous nous intéressons à la technique MLI '*programmée'*, avec *élimination sélective d'harmonique* et *asservissement du fondamental* de Patel et Hoft. Nous présentons une méthode numérique (Newton-Raphson) qui calcule les angles exacts de commutation. Enfin nous proposons un programme décrit en MATLAB pour construire une base de données des angles exacts. Cette base sera utilisée par la suite dans l'étape d'apprentissage de notre système MLI neuronal.

Dans le troisième chapitre, après avoir introduit la théorie des réseaux de neurones, nous présenterons notre système neuronal comme une solution pour notre problème. Nous utiliserons la base de données trouvée précédemment pour sortir les caractéristiques de ce

système. Enfin, nous terminerons ce chapitre par une comparaison entre les angles exacts et les angles calculés par notre système MLI neuronal.

Dans le dernier chapitre, nous commençons par donner un aperçu général sur les circuits FPGA. Une description de la carte de développement FPGA Memec Design V2MB1000, que nous avons utilisé pour implémenter et tester notre solution, ainsi qu'une explication du langage de description VHDL sont ensuite proposées. A la fin de ce chapitre nous proposons un programme écrit en VHDL implémentant notre commande MLI neuronale sur la carte FPGA. La simulation avec le logiciel ModelSim ainsi que l'implémentation sur la carte de développement Memec Design V2MB1000 sont présentées ensuite. Nous terminons enfin ce mémoire par une conclusion.

Chapitre 1

MOTEURS ASYNCHRONES ET ONDULEURS DE TENSION

I.1. LE MOTEUR ASYNCHRONE

I.1.1. Introduction

Le moteur asynchrone présente de nombreux avantages par rapport au moteur à courant continu. L'absence du système balais collecteur permet d'accroître sa fiabilité, et de limiter son coût de fabrication et d'entretien. Par ailleurs, les progrès réalisés en matière de commande et les développements technologiques, tant dans le domaine de l'électronique de puissance que celui de la micro électronique, ont rendu possible l'usage de commandes performantes faisant du moteur asynchrone un concurrent potentiel dans les domaines de la vitesse variable [2].

I.1.2. Définition

Un moteur asynchrone est une machine à 2p pôles, alimentée à partir du réseau alternatif de fréquence f et qui ne tourne pas exactement à la vitesse synchrone N définie par:

$$N = f / p = \omega / 2\pi p \quad [\text{t/s}] \qquad (1.1)$$

Le moteur asynchrone est formé d'un stator, relié à la source triphasée, et d'un rotor constitué d'un enroulement polyphasé en court-circuit.

Le stator crée un flux tournant à la vitesse angulaire synchrone:

$$\Omega = \omega / p. \qquad (1.2)$$

Ce flux tournant balaye les enroulements du rotor et y induit des courants. L'interaction entre le flux statorique et les courants induits rotoriques crée le couple de rotation du rotor.

Si le rotor tournait à la même vitesse que le flux tournant, le flux à travers les enroulements du rotor ne variant plus, il n'y aurait plus de courants induits dans le rotor, donc il n'y aurait pas de couple.

Le rotor tourne à une vitesse Ω' plus petite que Ω. L'écart entre Ω' et Ω augmente lorsque le couple résistant sur l'arbre du rotor augmente.

On appelle *glissement* l'écart des vitesses angulaires synchrone Ω et réelle Ω' rapporté à la vitesse synchrone Ω :

$$g= (\Omega-\Omega')/\Omega = (\omega-\omega')/\omega=(N-N')/N \qquad (1.3)$$

Avec: $N=\Omega/2\pi$ et $N'=\Omega'/2\pi$ (t/s) (1.4)

I.2 Commande des moteurs asynchrones

I.2.1 Introduction

On distingue deux types de commandes ; les commandes scalaires et les commandes vectorielles.

La commande scalaire : est basée sur le modèle en régime permanent, elle est simple à implanter avec une dynamique lente. Elle contrôle les grandeurs en amplitude.

La commande vectorielle : basée sur le modèle transitoire, précise et rapide elle permet le contrôle du couple à l'arrêt ; chère (encodeur incrémental ou estimateur de vitesse, DSP…). Elle contrôle les grandeurs en amplitude et en phase [3].

I.2.2 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'actionneur utilisé (onduleur de tension ou de courant). L'onduleur de tension étant maintenant le plus utilisé en petite et moyenne puissance, c'est la commande en *V/f* qui est la plus utilisée.

1-Contrôle en *V/f* de la machine asynchrone

Son principe est de maintenir *V/f*=Constant ce qui signifie garder le flux maximal 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 :

$$T\max = \frac{3pV^2}{4\pi f\,(Rs + \sqrt{Rs^2 + [2\pi f\,(Ls + Lre)]^2})} \qquad (1\text{-}5)$$

Avec Rs et Ls la résistance et l'inductance du stator, Lre l'inductance du rotor ramenée au stator, p le nombre de paires de pôles, V: la tension efficace d'entrée du moteur (d'une phase), f est la fréquence de la tension d'alimentation.

2-Contrôle scalaire du courant

La différence avec la commande précédente, c'est qu'un onduleur (commutateur) de courant qui est utilisé. On impose directement des courants dans les phases de la machine. La valeur du courant *Id* (courant continu) est égale à une constante près à la valeur efficace du courant imposé *Is*. Elle est imposée par régulation à l'aide d'un pont redresseur contrôlé.

I.2.3 Commande vectorielle

La commande vectorielle a été introduite il y a longtemps. Cependant, elle n'a pu être implantée et utilisée réellement qu'avec les avancées en micro-électronique. En effet, elle nécessite des calculs de transformé de Park, une évaluation de fonctions trigonométriques, des intégrations, des régulations… ce qui ne pouvait pas se faire en pure analogique.

Le contrôle de la machine asynchrone requiert le contrôle du couple, de la vitesse ou même de la position.

I.3 Onduleur de tension

I.3.1 Définition et principe

Les onduleurs de tension sont des convertisseurs statiques qui servent à alimenter, à fréquence fixe ou variable, des charges alternatives. Le but recherché est l'obtention pour chaque tension de sortie d'une forme d'onde approximant au mieux la sinusoïde.

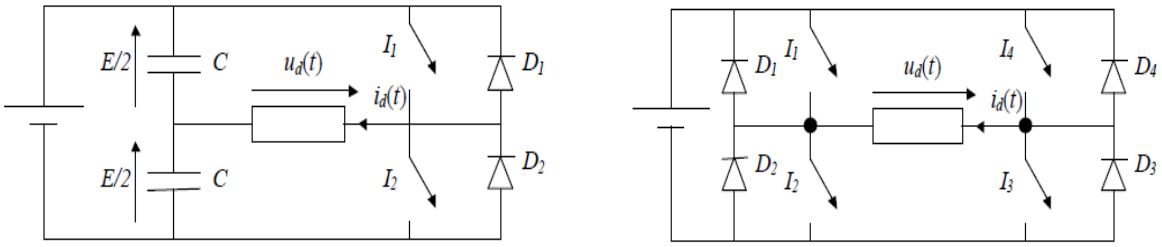
L'onduleur est dit autonome si l'établissement et la connexion entre l'entrée et la sortie ne dépendent que de la commande des semi-conducteurs. On distingue deux types d'onduleurs :

Les onduleurs autonomes de tension, qui sont alimentés par une source de tension continue, d'une impédance interne négligeable et de tension constante, peu affectée par les variations du courant qui la traverse.

Les onduleurs autonomes de courant qui sont alimentés par une source de courant [5].

I.3.2 Les Onduleurs monophasés de tension

Ce type d'onduleurs est destiné à alimenter des charges alternatives monophasées, on distingue deux configurations de base [4] [5]: en demi-pont ou en pont complet.



a- En demi-pont b- En pont

Figure 1.1 Schéma de l'onduleur

I.3.2 Les onduleurs triphasés

Les onduleurs monophasés sont utilisés pour des applications de faible puissance, alors que les onduleurs triphasés couvrent la gamme des moyennes et des fortes puissances.

L'objectif de cette topologie est de fournir une source de tension triphasée, dont l'amplitude, la phase et la fréquence sont contrôlables [6].

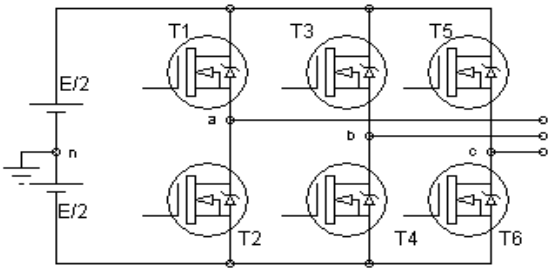


Figure 1.2 Schéma de principe d'un onduleur triphasé de tension

I.3. 3 La commande des onduleurs

Plusieurs stratégies de commande des onduleurs ont été développées dans la littérature dont les principes consistent soit à:

- La génération des signaux de commande des interrupteurs de puissance par l'asservissement de la tension de sortie de l'onduleur à une référence de tension sinusoïdale:

c'est la commande dite implicite, technique analogique telle que principalement la MLI engendrée, la modulation Delta.

- La détermination des instants de commutation des composants de puissance formant l'onduleur par le biais du développement en série de Fourier des formes d'onde souhaitées en sortie répondant à des critères bien définies (taux d'harmoniques, valeur du terme fondamental,..). C'est la commande dite explicite où la commande des interrupteurs peut être analogique ou numérique telle que la technique de la Sortie Sinusoïdale Synthétisée ou la modulation programmée.

I.3. 3.1 Technique MLI engendrée

La commande MLI Triangulo-sinusoïdale consiste à comparer une valeur de tension de référence de fréquence Fr, image du signal souhaité à la sortie appelée modulante, à une porteuse triangulaire ou en dent de scie de fréquence Fp. Les points d'intersection entre la modulante et la porteuse engendre l'enclenchement/déclenchement constituant ainsi une impulsion de durée variable et l'ensemble de ces impulsions reconstitue, de ce fait, le fondamental de la sinusoïde de référence [8].

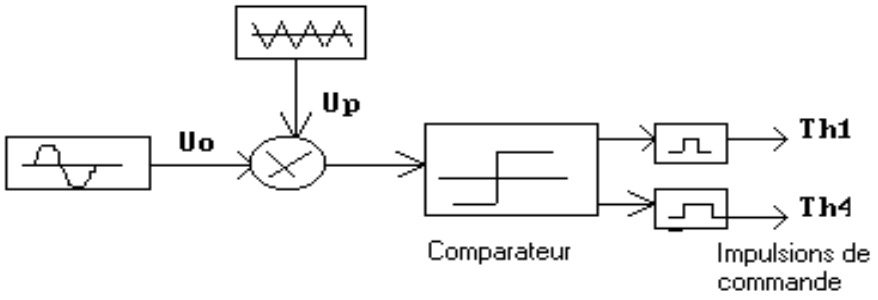


Figure 1.3 Schéma synoptique de la loi de modulation triangulo-sinusoidale

I.3.3.2 Technique MLI calculée (programmée)

a- Introduction :

Cette technique consiste à calculer les instants de commutation des interrupteurs (séquences de fonctionnement) de manière à répondre à certains critères portant sur le spectre fréquentiel de l'onde délivrée par l'onduleur. Ces séquences de fonctionnement sont alors mémorisées et restituées cycliquement pour assurer la commande des interrupteurs. Les critères usuellement retenus sont: l'élimination d'harmoniques de rangs spécifiés ou l'élimination d'harmoniques dans une bande de fréquences spécifiée [5] [9].

b- Principe :

La technique MLI « programmée »est basée sur l'algorithme de Patel et Hoft. Dans cette technique, il est possible d'asservir le fondamental de la tension MLI et d'annuler les amplitudes des (m-1) premiers harmoniques.

La tension MLI « programmée » est définie en fonction des angles exacts de commutation $\alpha_1,...,\alpha_m$ qui correspondent aux instants de commutation de la tenion MLI « programmée » d'une valeur positive +E/2 à une valeur négative –E/2 ou inversement. Ces angles sont dits 'exacts' car ils sont calculés avec une très grande précision. L'indice m est le nombre d'angles de commutation de la tension MLI 'programmée' ou nombre de commutations par quart-d'onde.

La tension MLI est construite de façon à présenter une symétrie demi-onde (fonction impaire par rapport à l'angle π). Cette symétrie permet de supprimer certains types d'harmoniques, ce qui simplifie le développement en série de Fourier de cette tension MLI et réduit le taux d'harmoniques. Ensuite, on *fixe l'amplitude du fondamental* à la valeur im et on *annule les amplitudes des (m-1) premiers harmoniques*.

Le taux im est le taux de modulation défini par :

$$im = \frac{V}{(E/2)} \tag{1.6}$$

Avec V la tension du fondamental, E la tension continue alimentant l'onduleur (figure 1.1) et E/2 la tension nominale Vn.

Pour calculer les angles exacts de commutation $\alpha_1,...,\alpha_m$, on doit résoudre un système de m équations non linéaires à m inconnues $\alpha_1,...,\alpha_m$. Ces angles exacts de commutation sont calculés par ordinateur puis convertis en valeurs temporelles.

Un circuit *numérique* génere alors la tension MLI « programmée » en fonction du temps.

En conclusion, on peut dire que la technique MLI « programmée » présente de nombreux avantages :

- Asservissement de la tension V du fondamental
- Variation de la fréquence f du fondamental en utilisant la relation de conversion d'une valeur angulaire en valeur temporelle : $\alpha = 2\pi ft$
- Elimination des (m-1) premiers harmoniques

Ces avantages permettent de remplacer l'alimentation sinusoïdale *idéale* avec une alimentation pratique ayant un taux d'harmoniques que l'on peut réduire à volonté.

La technique MLI programmée avec *asservissement du fondamental* et *élimination harmonique* sera décrite en détail dans le chapitre suivant.

Chapitre 2

TECHNIQUE MLI CALCULEE

II. 1-DESCRIPTION de la TECHNIQUE de PATEL et HOFT [1]

Soit la tension de sortie à deux états de l'onduleur demi-pont de la Figure 1.1. Les angles de commutation impairs $\alpha 1, \alpha 3, \ldots$ définissent des transitions négatives, tandis que les angles de commutation pairs $\alpha 2, \alpha 4, \ldots$ définissent des transitions positives. On suppose la tension de sortie périodique d'amplitude unité. Dans ce cas, la tension de sortie f(α) ou Vao peut s'écrire en série de Fourier:

$$f(\alpha) = a_0 + \sum_{n=1}^{\infty} (a_n \sin(n\alpha) + b_n \cos(n\alpha)) \qquad (2.1)$$

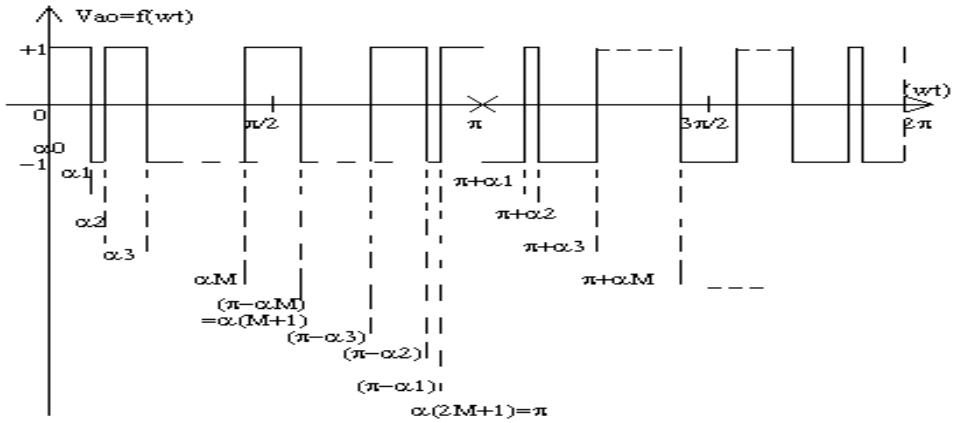


Figure 2.1 graphe d'une tension MLI calculée

Les coefficients a_n et b_n sont donnés par:

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(\alpha)d\alpha$$
$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(\alpha)\sin(n\alpha)d\alpha \qquad \text{n=1,2,3,4,…} \qquad (2.2)$$
$$b_n = \frac{1}{\pi}\int_0^{2\pi} f(\alpha)\cos(n\alpha)d\alpha$$

D'autre part comme f(α) présente une symétrie demi-onde : f(α+ π)= - f(α)
La *valeur moyenne a0 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 :

$$a_0 = 0$$
$$a_n = \frac{2}{\pi}\int_0^{\pi} f(\alpha)\sin(n\alpha)d\alpha \qquad (2.3)$$
$$b_n = \frac{2}{\pi}\int_0^{\pi} f(\alpha)\cos(n\alpha)d\alpha$$

Remplaçons f(α) par sa valeur dans l'équation (2.3):

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

$$a_n = \frac{2}{\pi}\left[\sum_{k=0}^{2M}\int_{\alpha_k}^{\alpha_{(k+1)}} (-1)^k \sin(n\alpha)d\alpha\right] \qquad \text{n=1,3,5,...}$$

$$a_n = \frac{2}{n\pi}\left[\sum_{k=0}^{2M}(-1)^k\left(\cos(n\alpha_k) - \cos(n\alpha_{k+1})\right)\right]$$

Avec $\alpha_{2M+1}=\pi$ et $\alpha 0 < \alpha 1 < \alpha 2 < \ldots < \alpha(2M+1)$.

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

Comme :

$$\alpha_0 = 0$$

$$\alpha_{2M+1} = \pi$$

On déduit :

$$\cos(n\alpha_0) = 1$$

$$\cos(n\alpha_{2M+1}) = (-1)^n$$

D'où :

$$a_n = \frac{2}{n\pi}\left[1 - (-1)^n + 2\sum_{K=1}^{2M}(-1)^K\cos(n\alpha_K)\right]$$

De même pour le coefficient b_n, on trouve, après simplifications, le résultat suivant :

$$b_n = \frac{-4}{n\pi}\sum_{K=1}^{2M}(-1)^K\sin(n\alpha_K)$$

Comme n doit être impair on peut écrire :

$$a_n = \frac{4}{n\pi}\left[1 + \sum_{K=1}^{2M}(-1)^K\cos(n\alpha_K)\right] \quad \text{n impair} \qquad (2.4a)$$

$$b_n = \frac{4}{n\pi}\left[-\sum_{K=1}^{2M}(-1)^K\sin(n\alpha_K)\right] \quad \text{n impair} \qquad (2.4b)$$

D'autre part la forme d'onde f(wt)=Vao(t) présente une symétrie quart-d'onde ie :

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

Et d'après la figure 2.1 on a :

$$\alpha_K = \pi - \alpha_{2M-K+1} \qquad K=1,2,\ldots,M$$

d'où :

$$\sin(n\alpha_K) = \sin(n(\pi - \alpha_{2M-K+1}))$$

$$\sin(n\alpha_K) = \sin(n\pi)\cos(n\alpha_{2M-K+1}) - \cos(n\pi)\sin(n\alpha_{2M-K+1})$$

Pour n impair on a :

$$\sin(n\pi) = 0 \quad \cos(n\pi) = -1$$

D'où :

$$\sin(n\alpha_K) = \sin(n\,\alpha_{2M-K+1}) \qquad K=1,2,\ldots,M \qquad (2.5)$$

Remplaçons (2.5) dans (2.4b) :

$$b_n = \frac{4}{n\pi}\sum_{k=1}^{M}\left(\sin(n\alpha_k) - \sin(n\alpha_{2M-k+1})\right) = 0$$

D'autre part :

$$\cos(n\alpha_k) = \cos(n(\pi - \alpha_{2M-k+1}))$$
$$\cos(n\alpha_k) = \cos(n\pi)\cos(n\alpha_{2M-k+1}) + \sin(n\pi)\sin(n\alpha_{2M-k+1})$$

D'où :

$$\cos(n\alpha_k) = -\cos(n\alpha_{2M-k+1}) \qquad (2.6)$$

Remplaçons (2.6) dans (2.4a), on obtient :

$$a_n = \frac{4}{n\pi}\left[1 + 2\sum_{k=1}^{M}(-1)^k \cos(n\alpha_k)\right] \qquad (2.7)$$

Avec *n impair* et *différent d'un multiple de 3*.

On considère une alimentation unité,ie Ed/2=1.

Le coefficient a_n est *l'amplitude de l'harmonique de rang n* du signal suivant:

$$V_{a0}(t) = f(\omega t) = \sum_{n=1}^{\infty} a_n \sin(n\omega t) \qquad (2.8)$$

Le système d'équations (2.7) possède m variables inconnues $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_m$ appelées *angles de commutation exactes*. Le problème est de calculer les valeurs de celles-ci qui permettent:

D'annuler les amplitudes a_n *des (m-1) premiers harmoniques* f_n :

$$f_n(\omega t) = a_n . \sin(n\omega t) \qquad n \neq 1$$

D'assigner une valeur déterminée au fondamental f_1 :

$$f_1(\omega t) = a_1 . \sin(\omega t)$$

Ces équations sont *non linéaires*. On utilisera la *méthode de Newton-Raphson* pour résoudre ce système de m équations non linéaires à m inconnues, méthode que l'on décrira en détail le paragraphe suivant.

II.2 calcul des valeurs exactes des angles de commutation par la méthode de NEWTON-RAPHSON

II.2.1 DESCRIPTION

La relation (2.7) est un système de m équations non linéaires à m inconnues $\alpha_1,\ldots,\alpha_m$. On assigne une valeur déterminée im, appelée indice de modulation, à l'amplitude a_1 du fondamental et on annule les amplitudes a_n des (m-1) premiers harmoniques.

On résout ce système par la méthode itérative de Newton-Raphson. Celle-ci converge bien (quadratiquement) si l'on possède un bon estimé initial de la solution. Cet estimé peut être obtenu par la méthode du gradient. Mais, celle-ci étant lente (convergence linéaire) [18], on utilisera plutôt l'algorithme 'on-line' de Taufik, Mellitt et Goodman [12] pour estimer rapidement les valeurs initiales de la solution du système non linéaire.

Pour les montages triphasés, les harmoniques de rang 3 et multiple de 3 sont inopérants. Pour cette raison les triplets ne sont pas éliminés dans cette étude.

D'autre part, il faut éliminer deux harmoniques de tension pour éliminer un harmonique de courant.

Comme l'amplitude du fondamental doit être fixée à une valeur déterminée, ceci fixe la première valeur de m à 3 (m étant le nombre de commutations par quart d'onde ou nombre de découpages par demi-onde). Par conséquent lorsque m augmente successivement par pas égal à 2, le nombre d'harmoniques de courant qui seront éliminés augmente par pas égal à 1[12].

Finalement on obtient un système de m équations non linéaires de la forme:

$$a_n = \frac{4}{n\pi}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(n\alpha_k)\right]$$

avec n=1,5,7,11,13,… et m=1,3,5,7,9,11,… (m impair).

Par exemple pour m égal à 3, n prend les valeurs 1,5,7; pour m égal à 5, n prend les valeurs 1,5,7,11,13; pour m égal à 7, n prend les valeurs 1,5,7,11,13,17,19 etc...

Le système (2.13) s'écrit encore:

$$a_1 = \frac{4}{n}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(\alpha_k)\right] = -im$$

$$a_5 = \frac{4}{5\pi}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(5\alpha_k)\right] = 0$$

$$a_7=\frac{4}{7\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(7\alpha_k)\right]=0$$

$$a_{11}=\frac{4}{11\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(11\alpha_k)\right]=0 \qquad (2.9)$$

$$a_{13}=\frac{4}{13\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(13\alpha_k)\right]=0$$

……

$$a_n=\frac{4}{n\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(n\alpha_k)\right]=0$$

Ces amplitudes sont normalisées i.e la tension d'alimentation continue est supposée égale à l'unité.

On doit signaler que la valeur de l'indice de modulation im assignée au fondamental est un indice sans dimension variant de 0 à 1. Pour obtenir la valeur correspondante en volt, il faut multiplier im par Ed/2, la tension d'alimentation continue de l'onduleur demi-pont.

D'autre part la méthode itérative de Newton-Raphson ne converge pas pour une valeur positive de im. C'est pourquoi on assigne une valeur négative (-im) au fondamental. Ce qui correspond à un déphasage de π du fondamental. Ce déphasage est sans effet sur le moteur.

En résumé, on a un système de forme générale :

$$f_1(\alpha_1,\alpha_2,\alpha_3,..,\alpha_m)=\frac{4}{\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(\alpha_k)\right]+im=0$$

$$f_2(\alpha_1,\alpha_2,\alpha_3,..,\alpha_m)=\frac{4}{5\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(5\alpha_k)\right]=0$$

$$f_3(\alpha_1,\alpha_2,\alpha_3,..,\alpha_m)=\frac{4}{7\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(7\alpha_k)\right]=0 \qquad (2.10)$$

……

$$f_m(\alpha_1,\alpha_2,\alpha_3,..,\alpha_m)=\frac{4}{n\pi}\left[1+2\sum_{k=1}^{m}(-1)^k\cos(n\alpha_k)\right]=0$$

Pour résoudre ce système, on va utiliser la méthode itérative de Newton-Raphson. Mais on doit localiser préalablement la solution cherchée.

II.2.2 Estimation initiale de la solution: [12]

Pour assurer la convergence de la méthode de Newton-Raphson, on doit obtenir un bon estimé initial de la solution 'exacte' recherchée. On peut utiliser la méthode du gradient mais on utilisera l'algorithme de Taufik, Mellitt et Goodman[12].

II.2.3 Résolution du système non linéaire par la Méthode de Newton-Raphson. [18]

Notons :

$$\alpha^* = (\alpha_1^*, \alpha_2^*, ..., \alpha_m^*)$$

Le *vecteur solution* du système non linéaire (2.10):

$$f_i(\alpha) = 0 \qquad i = 1, m$$

Avec $\alpha = (\alpha_1, ..., \alpha_m)$.

Si chaque fonction fi est continue et continûment différentiable, alors on peut la développer en série de Taylor dans le voisinage d'un estimé $\alpha^{(k)}$ (obtenu à la kième itération) proche de α^*.

On obtient :

$$f_i(\alpha^*) = f_i(\alpha^{(k)} + (\alpha^* - \alpha^{(k)}))$$

$$= f_i\left(\alpha^{(k)}\right) + \sum_{j=1}^{m}\left[\frac{\partial f_i(\alpha)}{\partial \alpha_j}\right]_{\alpha=\alpha^{(k)}} \cdot\left(\alpha_j^* - \alpha_j^{(k)}\right) + \ldots$$

$$\ldots + \frac{1}{2!}\sum_{j=1}^{m}\sum_{r=1}^{m}\left(\alpha_j^* - \alpha_j^k\right)\left(\alpha_r^* - \alpha_r^{(k)}\right)\left[\frac{\partial^2 f_i(\alpha)}{\partial \alpha_j \partial \alpha_r}\right]_{\alpha=\alpha(k)} + \ldots\ldots\ldots + \ldots\ldots = 0 \qquad (2.11)$$

Pour $i = 1, \ldots, m$

Si $\alpha^{(k)}$ est un estimé proche de α^*, les éléments $\left(\alpha_i^* - \alpha_i^{(k)}\right)^2$ sont négligeables ainsi que les termes de degré supérieur.

Le système (2.11) s'écrit donc:

$$\sum_{j=1}^{m}\frac{\partial f_i(\alpha)}{\partial \alpha_j}_{\alpha=\alpha^{(k)}} \cdot\left(\alpha_j^* - \alpha_j^{(k)}\right) = -f_i(\alpha^{(k)}) \qquad (2.12)$$

Avec $i = 1, \ldots, m.$

Définissons la matrice des dérivées premières :

$$E^{(k)} = \left(E_{ij}^{(k)}\right)$$

Avec $E_{ij}^{(k)} = \left[\frac{\partial f_i(\alpha)}{\partial \alpha_j}\right]_{\alpha=\alpha^{(k)}}$ $i = 1,....,m$ $j = 1,...,m$

D'où :

$$E^{(k)} = \frac{8}{\pi}\begin{bmatrix} \sin(\alpha_1) & -\sin(\alpha_2) & ... & \sin(\alpha_m) \\ \sin(5\alpha_1) & -\sin(5\alpha_2) & ... & \sin(5\alpha_m) \\ ... & ... & ... & ... \\ \sin(n\alpha_1) & -\sin(n\alpha_2) & ... & \sin(n\alpha_m) \end{bmatrix}$$

Définissons le *vecteur erreur* :

$$\Delta\alpha^{(k)} = \left[\Delta\alpha_1^{(k)}, \Delta\alpha_2^{(k)}, ..., \Delta\alpha_m^{(k)}\right]^t \qquad (2.13)$$

Avec $\Delta\alpha_j^{(k)} = \alpha_j^* - \alpha_j^{(k)}$

Soit le vecteur :

$$F^{(k)} = \left[F_1^{(k)}, F_2^{(k)}, \quad ... \quad , F_m^{(k)}\right]$$

Avec: $F_i^{(k)} = -f_i\left(\alpha^{(k)}\right).$

Alors le système (2.12) s'écrit sous la forme matricielle suivante:

$$E^{(k)}.\Delta\alpha^{(k)} = F^{(k)} \qquad (2.14)$$

Où $\Delta\alpha^{(k)}$ est le *vecteur inconnu*.

Le système (2.14) est un système linéaire que l'on peut résoudre par l'algorithme de Gauss [18].

Une fois le vecteur $\Delta\alpha^{(k)}$ déterminé, on obtient un meilleur estimé $\alpha^{(k+1)}$ de α^* par la relation :

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

On continue jusqu'à ce que :

$$\left|\alpha^* - \alpha^{(k)}\right| \to 0$$

En pratique, α^* étant l'inconnue, on arrête les opérations par l'un des tests suivants :

1) $k \geq KMAX$

2) $\left|f_i\left(\alpha^{(k+1)}\right)\right| \leq E0$

Où E0 est une borne supérieure de l'erreur fixée à priori et KMAX le nombre maximum d'itérations admissible.

La figure 2.2 donne, à titre d'exemple, le graphe des angles de commutation exacts calculés pour m égal à 3 et 5

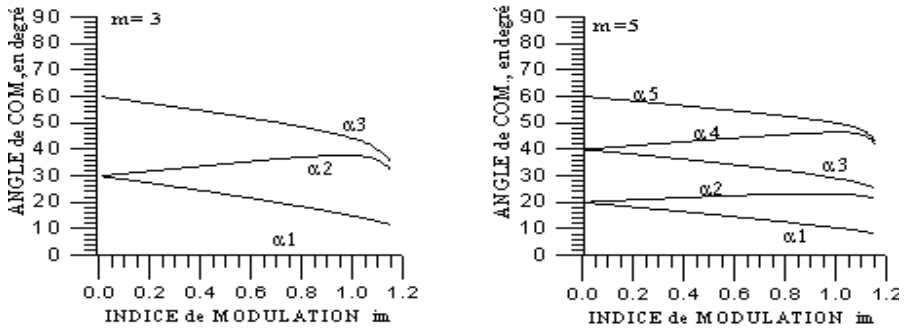


Figure 2.2 Courbes des angles de commutation exacts pour m égal à 3 et 5.

II.3 La base de données des angles exacts

Pour construire notre base de données des angles exacts qui sera utilisées dans l'étape d'apprentissage de notre système neuronal on fait le choix présenté dans le tableau 2.1 où on augmente le nombre d'angles (nombre d'harmonique éliminés) quand la fréquence diminue

Indice im	Nombre d'angles m
0< im <0.1	23
0.1 < im <0.2	19
0.2 < im <0. 4	15
0. 4< im <0.6	7
0. 6< im <0.8	5
0. 8< im <1	3

.

Tableau 2.1 Nombre d'angle en fonction de im

Le tableau suivant donne les angles de commutation exactes calculées par un programme écrit sur MATLAB pour « 0.1 < im <0.2 » et « m=19 »

L'angle	la valeur de Im									
	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.20
α1	5,71795	5,69226	5,66655	5,64083	5,61511	5,58937	5,56362	5,53785	5,51208	5,48629
α2	6,05435	6,05928	6,06420	6,06911	6,07401	6,07891	6,08380	6,08868	6,09356	6,09842
α3	11,7158	11,6898	11,6638	11,6378	11,6117	11,5856	11,5595	11,5334	11,5065	11,4814
α4	12,0987	12,1076	12,1165	12,1253	12,1341	12,1429	12,1517	12,1604	12,1691	12,1778
α5	17,7112	17,6847	17,6589	17,6317	17,6051	17,5785	17,5518	17,5251	17,4983	17,4715
α6	18,1365	18,1488	18,1610	18,1732	18,1853	18,1974	18,2095	18,2215	18,2335	18,2455
α7	23,7062	23,6793	23,6522	23,6252	23,5981	23,5709	23,5437	23,5164	23,4891	23,4617
α8	24,1693	24,1845	24,1996	24,2147	24,2298	24,2448	24,2598	24,2747	24,2896	24,3044
α9	29,7020	29,6747	29,6472	29,6198	29,5922	29,5647	29,5370	29,5093	29,4816	29,4537
α10	30,1977	30,2155	30,2332	30,2509	30,2685	30,2861	30,3037	30,3212	30,3386	30,3560
α11	35,6994	35,6718	35,6442	35,6165	35,5887	35,5609	35,5331	35,5051	35,4772	35,4491
α12	36,2224	36,2424	36,2624	36,2823	36,3022	36,3221	36,3419	36,3617	36,3814	36,4011
α13	41,6990	41,6714	41,6437	41,6160	41,5883	41,5605	41,5326	41,5047	41,4768	41,4488
α14	42,2434	42,2654	42,2873	42,3092	42,3311	42,3529	42,3748	42,3965	42,4183	42,4400
α15	47,7010	47,6736	47,6462	47,6188	47,5913	47,5638	47,5363	47,5087	47,4810	47,4534
α16	48,2610	48,2846	48,3082	48,3318	48,3554	48,3789	48,4025	48,4260	48,4495	48,4729
α17	53,7059	53,6790	53,6521	53,6252	53,5983	53,5713	53,5443	53,5173	53,4903	53,4633
α18	54,2752	54,3002	54,3252	54,3502	54,3752	54,4001	54,4251	54,4501	54,475	54,5000
α19	59,7137	59,6876	59,6616	59,6355	59,6094	59,5833	59,5571	59,5310	59,5049	59,4787

Tableau 2.2 : Les angles de commutation exacts pour m=19

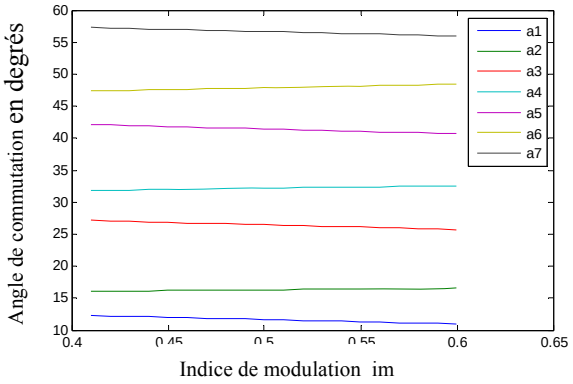


Figure 2. 3 Angles de commutation pour m=7

II.4 Conclusion

Dans ce chapitre, nous avons montré que le calcul des valeurs *exactes* des angles de commutation d'une tension MLI, avec asservissement du fondamental et élimination harmonique, exige la résolution d'un système de m équations non linéaires à m inconnues, m étant le nombre de commutations par quart-d'onde.

Nous avons mis au point un programme sur MATLAB pour résoudre ce système non linéaire et calculer les angles exacts en fonction de l'indice m [sélection des (m-1) premiers harmoniques à éliminer] et du taux de modulation im (asservissement de la valeur du fondamental de la tension MLI).

Le calcul des angles exacts, à l'aide de ce programme, est précis mais requiert un *temps de calcul très élevé* qui empêche une commande de vitesse temps réel ('on-line'). La technique de Patel et Hoft est donc une technique 'off-line'. Autrement dit, les valeurs exactes des angles de commutation sont d'abord calculées sur ordinateur ensuite ces valeurs sont stockées en mémoire pour pouvoir être utilisées dans une commande de vitesse d'un moteur. Un tel procédé requiert une capacité *mémoire élevée* qui augmente le coût de l'application.

Des solutions 'on-line' ont été proposées mais leur application reste limitée à cause de la *faible précision de calcul* des angles de commutation et d'un *temps de calcul relativement élevé* pour permettre une application temps réel.

Dans le chapitre suivant, on se propose de décrire un nouvel algorithme 'on-line', basé sur le principe des réseaux de neurone, qui calcule les angles de commutation avec une meilleure précision et un temps de calcul plus faible.

Chapitre 3

Réseaux de neurone artificiels et MLI neuronale

III.1 Réseaux de neurone artificiels

III.1.1 Introduction

Avec l'apparition des ordinateurs l'homme a découvert un moyen d'effectuer diverses tâches avec deux capacités non négligeables que lui ne possède pas : la rapidité et la précision.

Cependant l'exécution d'une tâche pour l'ordinateur nécessite sa programmation préalable par l'homme. Cette caractéristique fait apparaître les ordinateurs comme des machines exécutant des ordres « aveuglement » et l'homme n'a pas désespéré de voir un jour construire une machine à son image, c'est à dire intelligente, capable d'apprendre, de raisonner, de réfléchir sans son intervention.

Ce sont des recherches basées sur le fonctionnement du cerveau qui ont constituées le point de départ de cette gigantesque recherche.

Des travaux de neurobiologistes ont, en effet, révélé que le cerveau est constitué d'un nombre extrêmement élevé d'unités de traitement élémentaire de l'information **(les neurones biologiques)** fortement interconnectées.

L'information contenue dans le cerveau est stockée dans les connexions entre les neurones et c'est la coopération entre les neurones, qui effectuent un traitement fortement parallèle et distribué, qui donne sa puissance au cerveau.

III.1.2 Définition

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

III.1. 3 Applications

Les réseaux de neurones servent aujourd'hui `a toutes sortes d'applications dans divers domaines. Par exemple, pour développé un auto-pilote pour avion, ou encore un système de guidage pour automobile, on a conçu des systèmes de lecture automatique de chèques bancaires et d'adresses postales, on produit des systèmes de traitement du signal pour différentes applications militaires, un système pour la synthèse de la parole, des réseaux sont aussi utilisés pour bâtir des systèmes de vision par ordinateur, pour faire des prévisions sur les marchés monétaires, pour évaluer le risque financier ou en assurance, pour différents processus manufacturiers, pour le diagnostic médical, pour l'exploration pétrolière ou gazière, en robotique, et en télécommunication, enfin les réseaux de neurones ont aujourd'hui un impact considérable et, il y a fort à parier, que leur importance ira grandissant dans le futur.

III.1. 4 Le modèle neurophysiologique

Le neurone est une cellule composée d'un corps cellulaire et d'un noyau. Le corps cellulaire se ramifie pour former ce que l'on nomme les dendrites. Celles-ci sont parfois si nombreuses que l'on parle alors de chevelure dendritique ou d'arborisation dendritique. C'est par les dendrites que l'information est acheminée de l'extérieur vers le soma, corps du neurone **[17]**.

L'information traitée par le neurone chemine ensuite le long de l'axone pour être transmise aux autres neurones. La transmission entre deux neurones n'est pas directe. En fait, il existe un espace intercellulaire de quelques dizaines d'Angströms **(10^{-9} m)** entre l'axone du neurone afférent et les dendrites du neurone efférent. La jonction entre deux neurones est appelée la synapse figure 3.1.

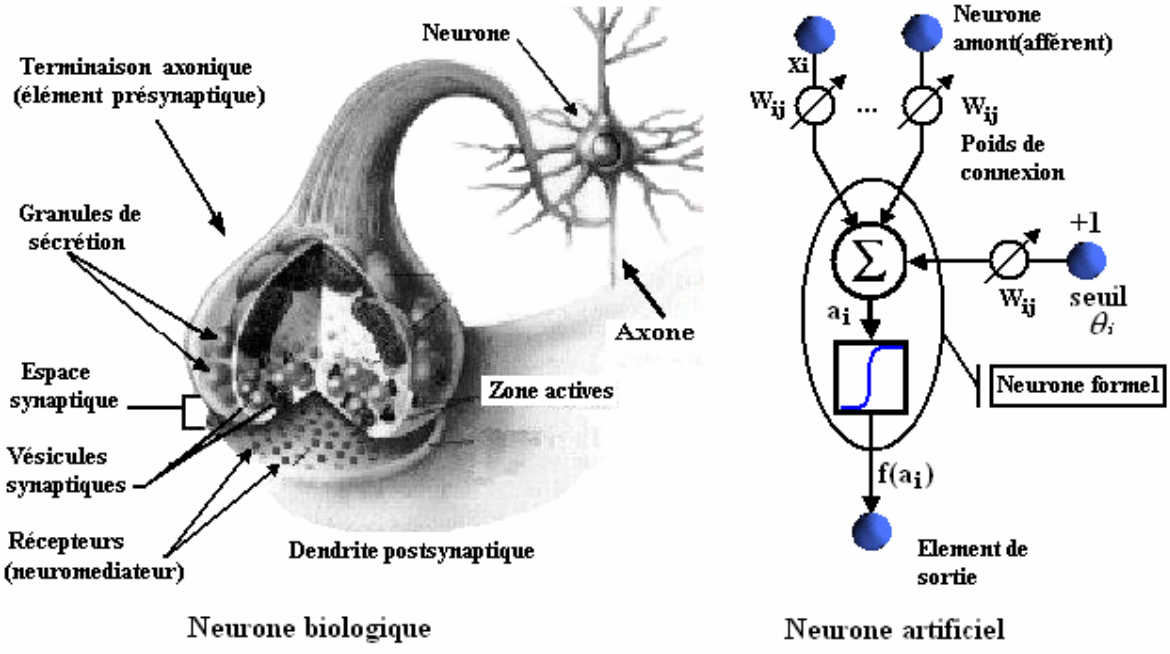


Figure 3.1. Mise en correspondance neurone biologique/neurone artificiel.

III.1.5 Le modèle mathématique

III. 1.5.1 Structure

La figure 3.1 montre la similitude entre un neurone artificiel et un neurone biologique qui est l'objet d'inspiration de la structure artificielle. Comme il est illustré sur la structure artificielle, chaque neurone est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance des neurones amont (afférents). A chacune de ces entrées est associée un poids **W** abréviation de ***weight*** (poids en anglais) représentatif de la force (ou bien de la pondération) de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals (efférents) **[19]**.

III. 1.5.2 Comportement

On distingue deux phases, la première est habituellement le calcul de la somme pondérée des entrées, où le neurone ***i*** reçoit des signaux de ***N*** neurones selon l'expression suivante :

$$a_i = \sum_{j=1}^{N} W_{ij} \cdot X_j$$

A partir de cette valeur, la sortie du neurone sera évaluée par une fonction de transfert.

Lorsque les neurones possèdent une fonction de transition qui est la fonction identité l'équation est celle de la bissectrice, l'état de sortie des neurones est évalué par l'équation suivante :

$$X_i = f^I(a_i) = a_i$$

Ces derniers sont appelés ***automates linéaires***.

Les automates linéaires sont employés, entre autres, par T. Kohonen pour construire son modèle de mémoires associatives **[20]**.

Cependant, si la fonction de transition est égale à une fonction à seuil on parle d'***automate à seuil***. Ces automates ont été utilisés par McCulloch et Pitts dans leur modèle d'automate formel **[21]**.

Pour ces automates les états sont évalués par des fonctions d'activations qui limitent généralement la sortie des neurones. Dans un automate à seuil l'équation qui définit l'état de sortie du neurone (figure 3.1) est :

$$x_i = f^S(a_i - \theta_i)$$

Où θ*i* est ***le seuil***.

Dans la littérature on dénombre un ensemble de fonctions d'activations. Les plus courantes sont présentées à la figure 3.2. La plupart des fonctions de transfert sont continues, offrant une infinité de valeurs possibles appartenant à l'intervalle [0, +1] ou [-1, +1]).

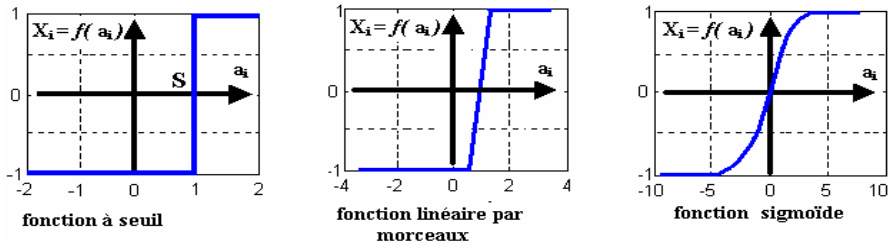


Figure 3.2. Différents types de fonctions de transfert pour le neurone artificiel.

Il se trouve que pour résoudre des problèmes complexes, en utilisant les réseaux de neurones, il est très important d'introduire des non-linéarités au niveau du fonctionnement du réseau. Cette caractéristique du réseau ne peut être obtenue que par l'utilisation d'une fonction d'activation qui est à la fois continue, différentiable par rapport aux paramètres du réseau et bornée. On trouve la fonction sigmoïde, au premier rang des fonctions les plus utilisées dans la phase d'apprentissage. On pourra alors utiliser une technique d'optimisation pour la minimisation d'une certaine fonction de coût, par exemple une descente du gradient.

III.1.6 Différentes topologies neuronales

III.1.6.1 Réseau multicouche (MLP)

Dans les réseaux multicouches MLP « Multi Layer Perceptron », les neurones sont arrangés par couche. Il n'y a pas de connexion entre neurones d'une même couche et les

connexions ne se font qu'avec les neurones des couches avales figure 3.3 Habituellement, chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et à celle-ci seulement.

Ceci nous permet d'introduire la notion de sens de parcours de l'information (de l'activation) au sein d'un réseau.

On appelle couche d'entrée l'ensemble des neurones d'entrée, couche de sortie l'ensemble des neurones de sortie. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelées couches cachées.

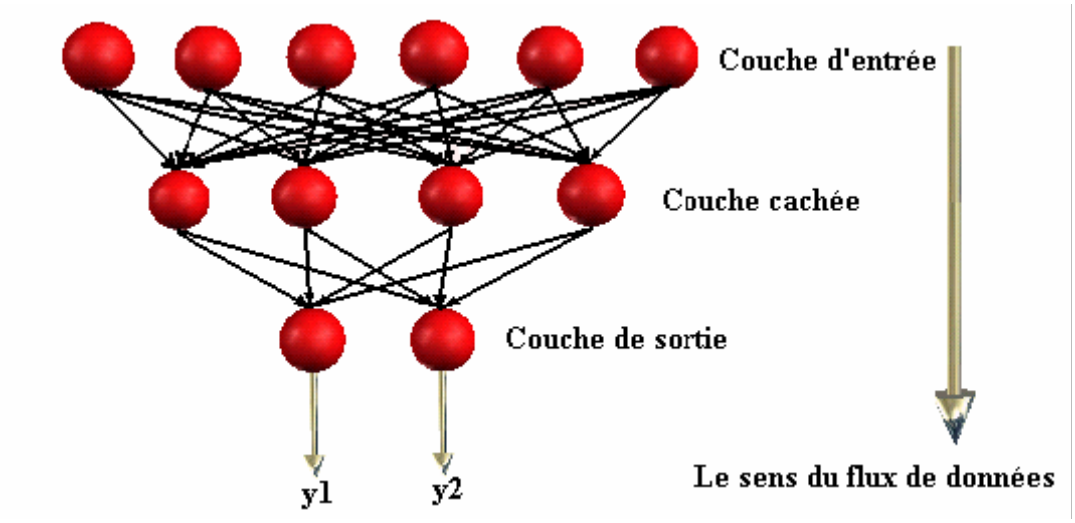


Figure 3.3 Topologie d'un réseau multicouche (MLP).

III.1.6.2 Réseau à connexions locales

Il s'agit d'une structure multicouche, conserve une certaine topologie. Chaque neurone entretien des relations avec un nombre réduit et localisé de neurones de la couche avale, figure 3.4 Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique.

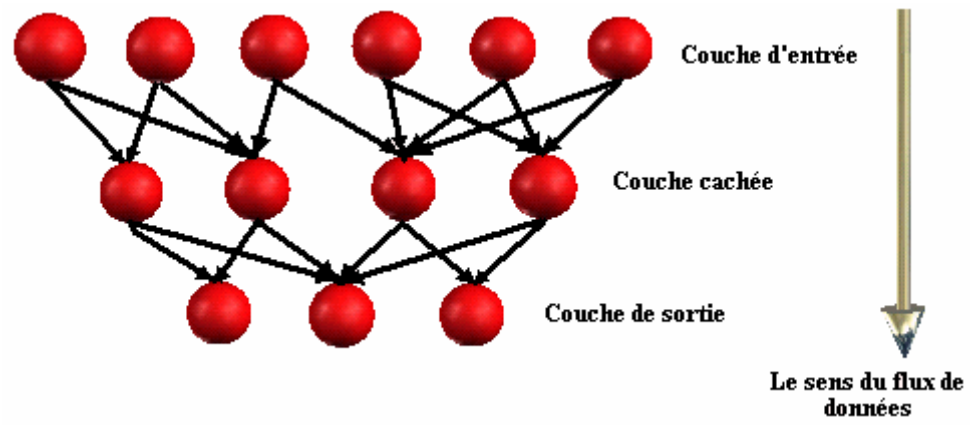


Figure 3.4 Réseau à connexion local.

III.1.6.3 Réseau à connexions récurrentes

Les connexions récurrentes, ramènent l'information en arrière par rapport au sens de propagation, défini dans un réseau multicouche. Ces connexions sont le plus souvent locales (figure 3.5). Les réseaux récurrents sont appelés aussi réseaux dynamiques. Cela est dû à la caractéristique dynamique de ces réseaux, ils évoluent dans le temps car la sortie actuel y(t) de ces réseaux dépendent aussi des valeurs de sorties précédente y(t-1). La notion du temps dans ces réseaux est importante.

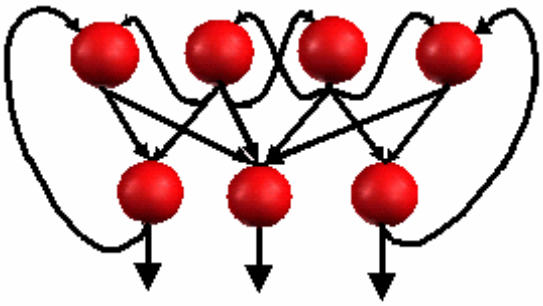


Figure 3.5 Réseau à connexions récurrentes.

III.1.6.4 Réseau à connexion complète

C'est la structure d'interconnexion la plus générale figure 3.6 Chaque neurone est connecté à tous les neurones du réseau et à lui-même. Le réseau à connexion complète appelé encore le modèle de Hopfield fut présenté en 1982. Ce modèle est basé essentiellement sur le principe des mémoires associatives.

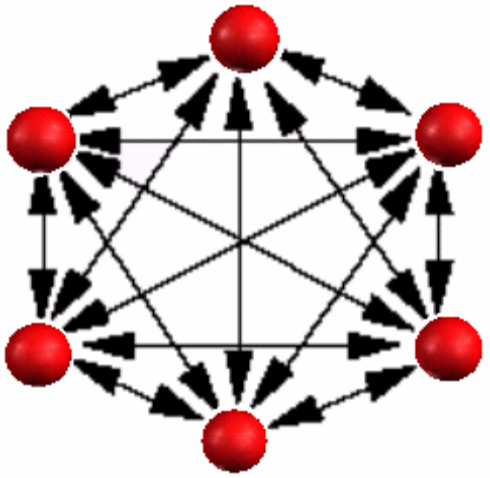


Figure 3.6 Réseau à connexion complète.

III.1.7 Apprentissage d'un réseau multicouche

L'apprentissage est vraisemblablement la propriété la plus intéressante des réseaux neuronaux. La plupart d'entres eux font appel à des règles d'apprentissage sur des données pour ajuster les poids des connexions synaptiques. En d'autres termes les réseaux de neurones sont généralement élaborés à partir d'exemples (***comme un enfant apprend à reconnaître un chien d'un chat à partir d'exemples de l'espèce***). Durant cette phase d'apprentissage, l'état du réseau de neurones évolue suivant une loi de minimisation du coût de sa sortie, jusqu'à l'obtention du comportement désiré **[19]**.

En conséquence, le but de l'apprentissage est l'estimation des poids synaptiques, pour remplir au mieux la tâche à laquelle le réseau est destiné **[22]**.

III.1.7.1 Apprentissage supervisé

Un ***apprentissage supervisé*** est lié à la disponibilité d'exemples, ou de modèles de réponses. L'ensemble des exemples utilisés pour l'apprentissage se présente sous forme de couples **(x, y*)** ou **y*** est la réponse désirée du réseau à l'entrée **x**.

Lorsque le réseau apprend à mimer une fonction multi variable, il utilise des exemples d'entrée/sortie. Ses poids sont ajustés sous l'influence d'un signal d'erreur qui représente la différence entre la sortie estimée par le réseau et la sortie désirée par le modèle.

Les caractéristiques de l'algorithme neuronal lui permettent ensuite de généraliser et de répondre à une entrée quelconque, même si elle n'appartient pas au jeu d'apprentissage. Le réseau réalise une approximation de la fonction qui lui a servi de ***professeur*** **[19]**.

III.1.7.2 Apprentissage non supervisé

L'hypothèse d'existence d'un maître qui supervise l'apprentissage n'est pas toujours possible. Dans ce cas, on ne connaît pas les sorties désirées **y*** des exemples d'apprentissage.

Le réseau doit donc apprendre de lui-même et on parle ***d'apprentissage non supervisé***. Ce type d'apprentissage permet au réseau d'extraire des propriétés contenues implicitement dans un ensemble de données, représentées par des vecteurs de grande dimension, et l'on cherche à les regrouper, selon des critères de ressemblance qui sont inconnus *a priori*.

III.1.7.3 Caractéristique de l'algorithme d'apprentissage supervisée

Un réseau de neurones est conçu pour réaliser une tâche que le concepteur définit par un ensemble d'apprentissage **D** (base de données). Chaque élément de cet ensemble est appelé exemple d'apprentissage et se présente sous la forme d'un couple ***(x, y*)*** ou **x** est une valeur d'entrée du réseau et ***y**** la valeur désirée correspondante pour les sorties des neurones de sortie.

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.

Les valeurs des poids synaptiques sont, en général, déterminées par un processus algorithmique mettant en oeuvre l'ensemble d'apprentissage.

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****.

W* est donc la solution d'un problème d'optimisation, consistant à minimiser une fonction de coût : ***E(W, D)***.

III.1.8 La méthode du gradient

La plupart des méthodes d'optimisation non linéaires sont basées sur la même stratégie.

On choisit une valeur initiale ***W(0)*** de la matrice ***W***, puis on utilise un processus itératif dans lequel on tente d'optimiser la fonction ***E***. À chaque itération, cette optimisation implique deux étapes **[17]**

- Le choix de la direction dans laquelle on va chercher la valeur suivante ***W(t+1)***.
- Et le déplacement η le long de cette direction.

III.1.8.1 Rétro propagation du gradient

L'algorithme du rétro propagation du gradient est très connu et le plus utilisé dans les applications des réseaux de neurones. À chaque itération, on retire un exemple d'apprentissage ***(x, y*)*** et on calcule une nouvelle estimation de ***W(t)***. Cette itération est réalisée en deux phases :

a- Propagation

À chaque itération, un élément de l'ensemble d'apprentissage **D** 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, les calculs s'effectuent dans le sens inverse. À la sortie du réseau, on forme le critère de performance **J** en fonction de la sortie réelle du système et sa valeur désirée. Puis, on évalue le gradient de **J** par rapport aux différents poids en commençant par la couche de sortie et en remontant vers la couche d'entrée.

III.1.8.2 Calcul du gradient

Pour un exemple ***i*** d'un ensemble d'observation ***E***, la fonction de coût des moindres carrés est égale à la somme, sur les N_2 valeurs de l'ensemble d'observation, de carrés des écarts entre la sortie du modèle (sortie du réseau de neurones = y_i) et la sortie désirée (grandeur mesurée notée y_i^{des}). On cherche à minimiser, à chaque étape de mise à jour, le critère suivant :

$$J = \frac{1}{2}\sum_{K=1}^{N_2}\left(y_i^{des} - y_i\right)^2 \qquad (3.1)$$

y_i^{des} : La composante ***i*** de la sortie désirée du système.

y_i : La composante ***i*** de la sortie calculée du système.

N_2 : Le nombre d'exemples (de valeurs) dans la base d'apprentissage.

Le problème consiste à déterminer les poids ***W*** de toutes les couches qui minimisent le critère de performance **J.**

La mise à jour de ***W*** se fait selon la règle de delta :

$$\Delta W = -\eta \frac{\partial J}{\partial W} \tag{3.2}$$

$$W_{ij}^{(k)}(t+1) = W_{ij}^{(k)}(t) - \Delta W \tag{3.3}$$

Ce qui revient à déterminer les variations du critère de performance **J** par rapport aux variations des poids.

III.1.8.3 Evaluation de la couche de sortie

Calcul de $\frac{\partial J}{\partial W_{ij}}$ selon la figure 3 .7

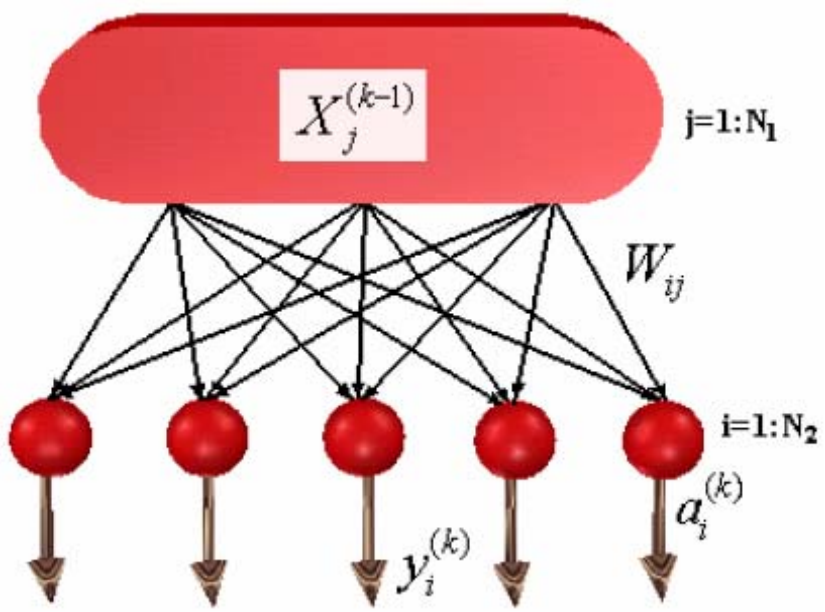


Figure 3.7 Représentation de la couche de sortie d'un réseau de neurones.

Où **N2** est le nombre de neurones dans la couche ***k*** (couche de sortie), **N1** le nombre de neurones dans la couche ***(k -1)*** (la dernière couche cachée).

$$\frac{\partial J}{\partial W_{ij}^{(k)}} = \frac{\partial J}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial a_i^{(k)}} \frac{\partial a_i^{(k)}}{\partial W_{ij}^{(k)}} \tag{3.4}$$

$$J = \frac{1}{2} \sum_{i=1}^{N_2} \left(y_i^{des} - y_i\right)^2 \Rightarrow \frac{\partial J}{\partial y_i^{(k)}} = -(y_i^{des} - y_i^{(k)}) \tag{3.5}$$

$$y_i^{(k)} = g\left(a_i^{(k)}\right) \Rightarrow \frac{\partial y_i^{(k)}}{\partial a_i^{(k)}} = g'\left(a_i^{(k)}\right) \quad (3.6)$$

$$a_i^{(k)} \sum_{j=1}^{N_1} W_{ij} \cdot X_j^{(k-1)} \Rightarrow \frac{\partial a_i^{(k)}}{\partial W_{ij}} = X_j^{(k-1)} \quad (3.7)$$

De l'équation (3.5) et (3.6) :

$$Err_i^{(k)} = \frac{\partial J}{\partial a_i^{(k)}} = -\left(y_i^{des} - y_i^{(k)}\right).\ g'\left(a_i^{(k)}\right) \quad (3.8)$$

Où $Err_i^{(k)}$: l'erreur du $i^{ème}$ neurone dans la $k^{ème}$ couche.

Et de l'équation (3.7) et (3.8) : $$\frac{\partial J}{\partial W_{ij}^{(k)}} = Err_i^{(k)} \cdot X_j^{(k-1)} \quad (3.9)$$

III.1.8.4 Evaluation des couches cachées

Calcul $\frac{\partial J}{\partial W_{ij}}$ de selon la figure 3.8

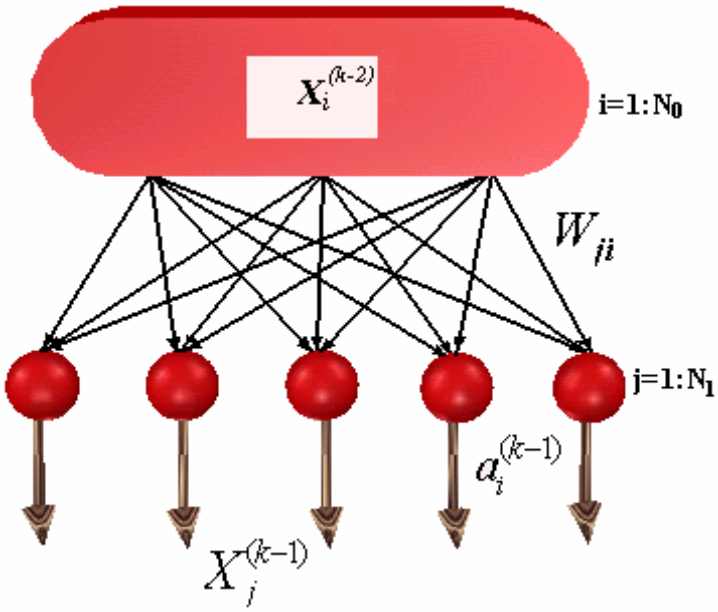


Figure 3.8 Représentation d'une couche cachée d'un réseau de neurones.

Où **N0** est le nombre de neurones de la couche ***(k-2)***.

$$\frac{\partial J}{\partial W_{ji}^{(k-1)}} = \frac{\partial J}{\partial X_j^{(k-1)}} \frac{\partial X_j^{(k-1)}}{\partial a_j^{(k-1)}} \frac{\partial a_j^{(k-1)}}{\partial W_{ji}^{(k-1)}} \quad (3.10)$$

$$\frac{\partial J}{\partial X_j^{(k-1)}} = \sum_{i=1}^{N_2} \frac{\partial J}{\partial a_i^{(k)}} \frac{\partial a_i^{(k)}}{\partial X_j^{(k-1)}} \Rightarrow \frac{\partial J}{\partial X_j^{(k-1)}} = \sum_{i=1}^{N_2} Err_i^{(k)} . W_{ji}^{(k)} \quad (3.11)$$

$$X_j^{(k-1)} = g\left(a_j^{(k-1)}\right) \Rightarrow \frac{\partial X_j^{(k-1)}}{\partial a_j^{(k-1)}} = g'\left(a_j^{(k-1)}\right) \quad (3.12)$$

$$a_j^{(k-1)} = \sum_{i=1}^{N_0} W_{ji}^{(k-1)} . X_i^{(k-2)} \Rightarrow \frac{\partial a_j^{(k-1)}}{\partial W_{ji}^{(k-1)}} = X_i^{(k-2)} \quad (3.13)$$

De l'équation (3.11) et (3.12) on aura :

$$Err_j^{(k-1)} \equiv \frac{\partial J}{\partial a_j^{(k-1)}} = \left[\sum_{k=1}^{N_2} Err_i^{(k)} . W_{ji}^{(k)}\right] g'\left(a_j^{(k-1)}\right) \quad (3.14)$$

$$\frac{\partial J}{\partial W_{ji}^{(k-1)}} = Err_j^{(k-1)} . X_i^{(k-2)} \quad (3.15)$$

III.1.8.5 Modification des paramètres du réseau en fonction du gradient de J

Dans l'étude précédente, nous avons vu comment évaluer le gradient simple de la fonction de coût ***J*** par rapport aux paramètres du réseau de neurones, à chaque itération du processus d'apprentissage. Une fois que l'on dispose de cette évaluation, on effectue une modification des poids selon l'équation (3.3), afin d'approcher d'un minimum de la fonction de coût ***J*** dans l'espace des poids, pour cela il faut que la condition suivante doit être vérifiée

$$\frac{\partial J}{\partial W_{ij}} = 0 \quad (\forall i, j)$$

Le paramètre η dans l'équation (3.2) est appelé pas du gradient ou pas d'apprentissage.

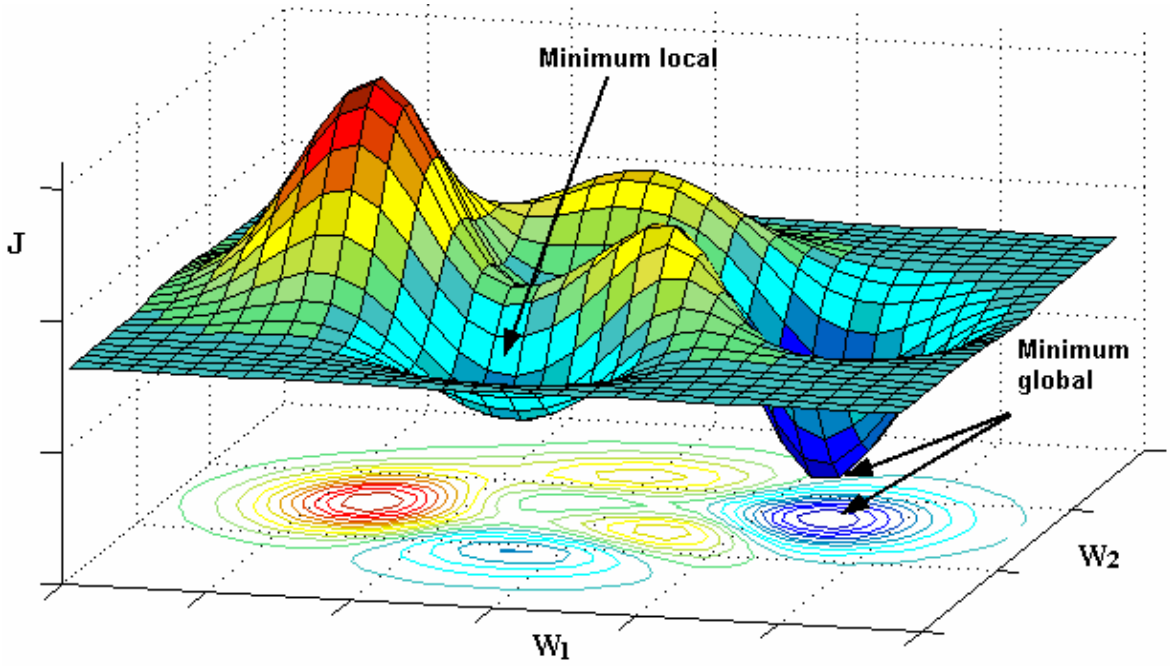


Figure 3.9 Représentation de la fonction de coût J d'un neurone à deux entrées pondérées W_1 et W_2.

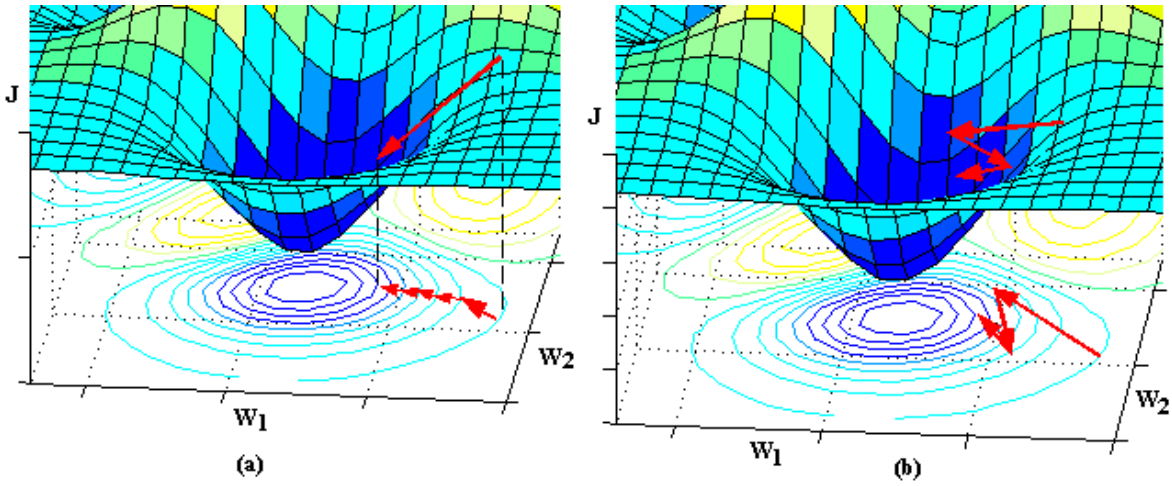


Figure 3.10 Minimisation de la fonction de coût J par la méthode du gradient.

(a) Pas du gradient est petit, convergence lente, le minimum global peut être atteint.

(b) Pas du gradient est grand, convergence rapide, le minimum global est rarement atteint.

Cette méthode du gradient de premier ordre est simple, mais elle présente de nombreux inconvénients, entres autres :

I. Si le pas du gradient est trop petit, la décroissance du coût ***J*** est très lente figure 3.10(a). Si le pas est trop grand, le coût ***J*** peut augmenter ou osciller, cette situation est illustrée sur la figure 3.10 (b), qui représente une forme 3D d'une fonction de coût ***J*** ayant des minimum locaux et un minimum global, où sa projection sur le plan de base formé par

deux variables (poids W_1 et W_2) donne un contour de plusieurs niveaux de la fonction de coût.

II. Au voisinage d'un minimum de la fonction de coût, la pente de la descente est faible, ce qui revient à dire que le gradient de cette fonction tend vers zéros c-à-d

$\frac{\partial J}{\partial W_{ij}} \approx 0$.

À ce niveau, multiplier cette grandeur par le pas d'apprentissage η , n'améliorera pas considérablement le résultat, et donc l'évolution du vecteur des paramètres du réseau lors de la mise à jour par l'équation (3.3) devient très lente. Il en va de même si la fonction du coût présente des « plateaux » où sa pente est très faible. Ces plateaux peuvent être très éloignés d'un minimum, et, il est impossible de savoir si une évolution très lente du gradient est due au fait que l'on est au voisinage d'un minimum, ou que l'on se trouve sur un plateau de la fonction de coût.

III. Si la courbure de la surface de coût varie beaucoup, la direction du gradient peut être très différente de la direction qui mènerait vers le minimum ; c'est le cas si le minimum recherché se trouve dans une « ***vallée*** » longue et étroite (les courbes de niveau sont des ellipsoïdes allongées au voisinage du minimum (figure 3.10). Pour pallier le premier inconvénient, de très nombreuses méthodes ont été proposées avec des succès divers. Les méthodes de recherche unidimensionnelle, fondées sur des principes solides, sont recommandées. Pour faire face aux deux autres problèmes, on utilise des méthodes de gradient du second ordre qui, au lieu de modifier les coefficients uniquement en fonction du gradient de la fonction de coût, utilisent les dérivées secondes de cette dernière.
La méthode la plus célèbre du second ordre c'est l'algorithme de *Levenberg-Marquart*, il faut noter que ces méthodes ne sont pas spécifiques aux réseaux de neurones ; ce sont des méthodes très générales d'optimisation, qui existaient bien avant l'introduction des réseaux de neurones.

III.1.8.6 Choix d'une structure neuronale

Un problème qu'on doit impérativement résoudre avant d'utiliser un réseau de neurones est la définition de sa structure. Pour une topologie multicouche MLP, le nombre de neurones d'entrée/sortie du réseau de neurones est imposé par la structure de fonctionnement globale où il sera inséré, tandis que le nombre de couches cachées ainsi que leurs nombres de neurones correspondant à chaque couche, ne sont pas limités. L'objectif final étant une réalisation matérielle embarquable, alors pour diminuer le temps de calculs, il est nécessaire de développer une architecture neuronale aussi petite que possible.

III.1.8.7 Centrage des données

Avant tout apprentissage, il est indispensable de normaliser et de centrer toutes les données de la base d'apprentissage, afin qu'ils soient actifs, en moyenne sur la partie linéaire de la fonction sigmoïde figure 3.11

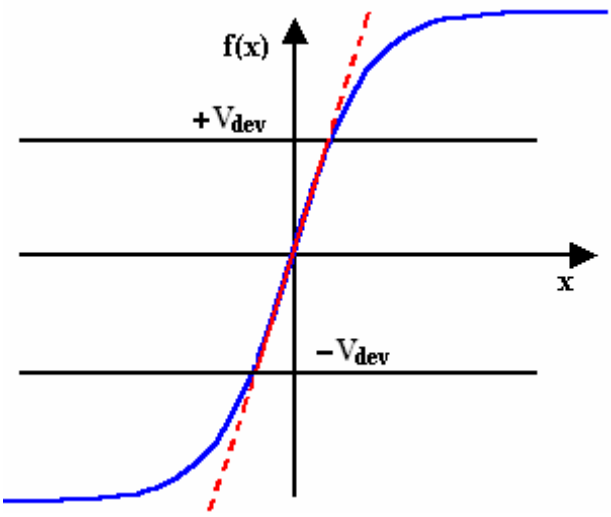


Figure 3.11 Centrage et normalisation des données de la base d'apprentissage.

En effet, si des entrées ont des grandeurs très différentes, celles qui sont « petites » n'ont pas d'influence sur l'apprentissage.

En pratique, il est donc recommandé, d'appliquer pour chaque vecteur d'entrée ***V*** la normalisation suivante :

Soit : V_{min} ,V_{max} et V_{moy} respectivement le minimum, le maximum et la moyenne de la variable ,V considérée. On définit :

$$V_{dev} = Max\,(V_{moy} - V_{min}\,,\, V_{max} - V_{moy}) \qquad (3.16)$$

Les variables V sont alors centrées/réduites par :

$$V = scal * \frac{(V - V_{moy})}{2 * V_{dev}} \qquad (3.17)$$

III.1.9 Conclusion

Un réseau de neurones artificiels est une modélisation mathématique de la merveilleuse machine qui constitue le cerveau. La capacité de ce dernier d'accomplir des opérations très complexes à partir des éléments de base que constituent les neurones a fasciné les chercheurs, qui ont d'abord commencé par modéliser le neurone. Dans ce modèle, un neurone accomplit la somme pondérée des potentiels d'actions qui lui parviennent, puis s'active si celle-ci dépasse un certain seuil, en transmettant une réponse dont la valeur correspond à son activation.

Les réseaux de neurones artificiels deviennent un nouveau moyen de traitement de l'information, bien que la plupart de ces réseaux soient souvent exploités à travers des simulations sur des ordinateurs, qui ne sont que des outils d'analyse. Le principe de traitement dans les réseaux de neurones est tout à fait différent de traitement algorithmique des calculateurs classiques. Un réseau de neurones fonctionne sans programme, n'exécute pas d'instructions de façon séquentiel et ne possède pas une mémoire pour y stocker les codes d'instructions ou les données. Il doit être considéré comme un « Processeur Parallèle » et non pas comme une nouvelle technique de programmation.

Après cette étude on remarque que les réseaux de neurones possèdent plusieurs propriétés qui leurs permettent d'être des candidats naturels lors de l'identification des systèmes. Toutes les informations du système peuvent être stockées dans un réseau, et ceci grâce aux propriétés suivantes : la non linéarité, le parallélisme, l'implantation hardware, l'apprentissage et la généralisation, et ils sont naturellement multi-variables donc directement applicables aux systèmes MIMO « M*ulti-Input Multi-Output* ». Ces propriétés rendent les réseaux de neurone souhaitables en identification des systèmes non linéaires.

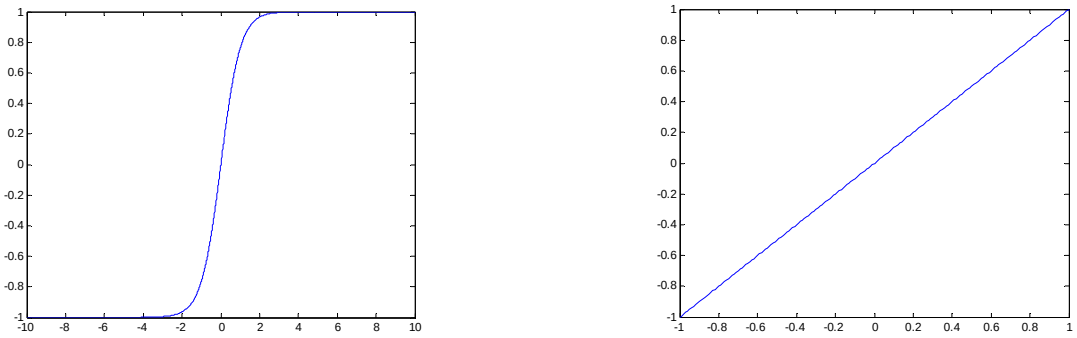
III.2 MLI neuronale

III.2.1 Introduction

Le but de notre projet et de trouver un réseau multicouche qui permet de générer des angles de commutation d'un signal MLI proches des angles exactes de notre base de données, ce réseau sera implémenté sur la carte FPGA pour générer les signaux de commande d'un onduleur MLI triphasé.

Notre réseau est composé d'une couche d'entrée, une couche cachée et une couche de sortie.

Pour les fonctions d'activation on a choisi la fonction « tansig » entre la couche d'entrée et la couche cachée, et une fonction linéaire entre la couche cachée et la couche de sortie (figure 3.12).



a-La fonction tansig b-La fonction linéaire

Figure 3.12 : les fonctions d'activation utilisées

III.2.2 Architecture

L'entrée de notre réseau c'est l'indice im qui varie de 0 à 1, on souhaite que notre commande génére 100 vitesses, donc il faut que notre indice im varie par pas de 0.01.

Le nombre d'angles de commutation par quart de période m dépend de im selon le tableau 2.1.

D'après le tableau il y a six valeurs de m, pour chaque valeur on construit un réseau. Pour créer ces réseaux sur MATLAB on utilise la fonction « newff », la figure 3.13 montre l'architecture de notre réseau.

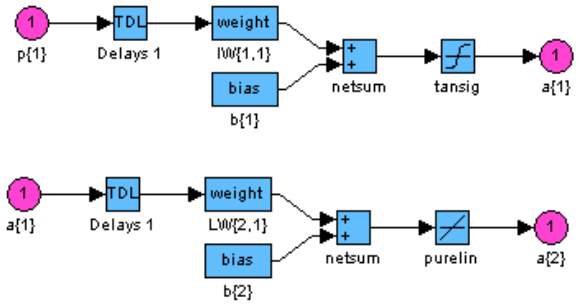


Figure 3.13 L'architecture du réseau utilisé sous Simulink

Où :

p : représente l'entrée de notre réseau im

IW{1,1} : une matrice qui contient les poids entre la couche d'entrée et la couche cachée

LW{2,1} : une matrice qui contient les poids entre la couche cachée et la couche de sortie

b{1} :une matrice qui contient les seuils entre la couche d'entrée et la couche cachée

b{2} :une matrice qui contient les seuils entre la couche cachée et la couche de sortie

a{2}:représente la sortie de notre réseau

III.2. 3 L'Apprentissage

Cette étape permet de calculer les paramètres de nos six réseaux qui sont les poids (weights) et les seuils (bias).

Pour créer notre réseau sous MATLAB on a utilisé la fonction « newff » avec la base de données des angles de commutation exacts calculés dans le chapitre précédent.

Pour lancer l'apprentissage sous MATLAB on utilise la fonction « train ». Le tableau 3.1 donne un exemple des poids et seuils calculés par le programme pour m=7.

IW{1,1}	LW{2,1}	b1{1}	b2{2}
-0,94864364	1,10442596	0,49285027	0,48368996
	-1,10135659		0,53339166
	1,10418357		0,48638628
	-1,10467793		0,52793608
	1,10404012		0,48674086
	-1,10467119		0,52109015
	1,10421175		0,48426608

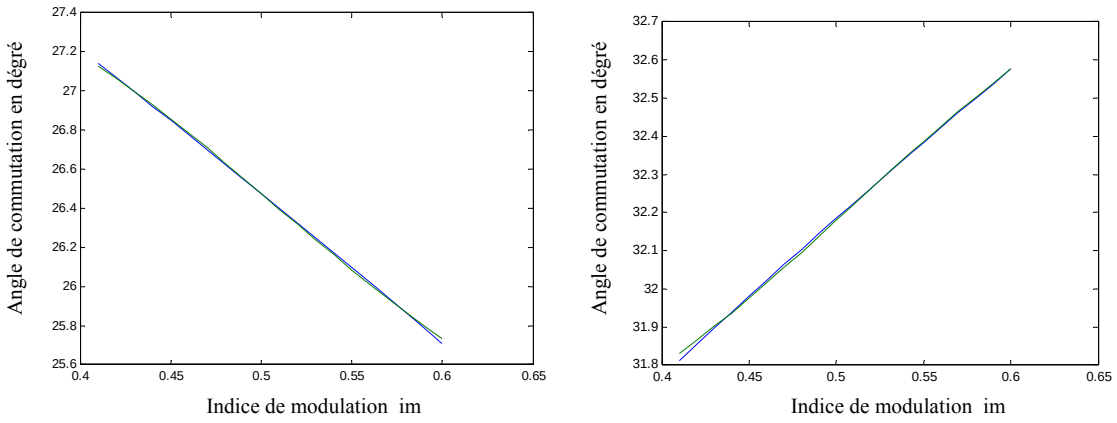
Tableau 3.1 : Les poids et les seuils pour m=7

III.2.4 Simulation

Avant l'implémentation de notre commande sur la carte FPGA on la simule sur MATLAB et PSIM, des résultats de simulation sont présentés dans le tableau 3.2 et sur les figurer 3.14, 3.15, 3.16.

angles exacts	Angles approximés	Erreur
11,67103009	11,6724438	-0,00141372
16,297481	16,2928275	0,00465346
26,47641812	26,4757616	0,00065649
32,1851373	32,1792789	0,0058584
41,45152938	41,4505927	0,00093673
47,86367067	47,8596118	0,00405887
56,671649	56,6726525	-0,00100348

Tableau 3.2 Angles exacts et approximés pour im=0 .5 et m=7

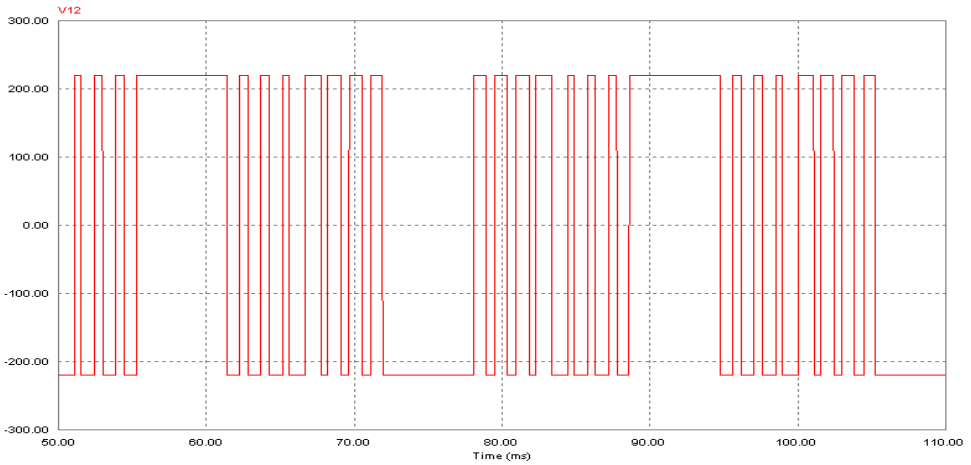


a b

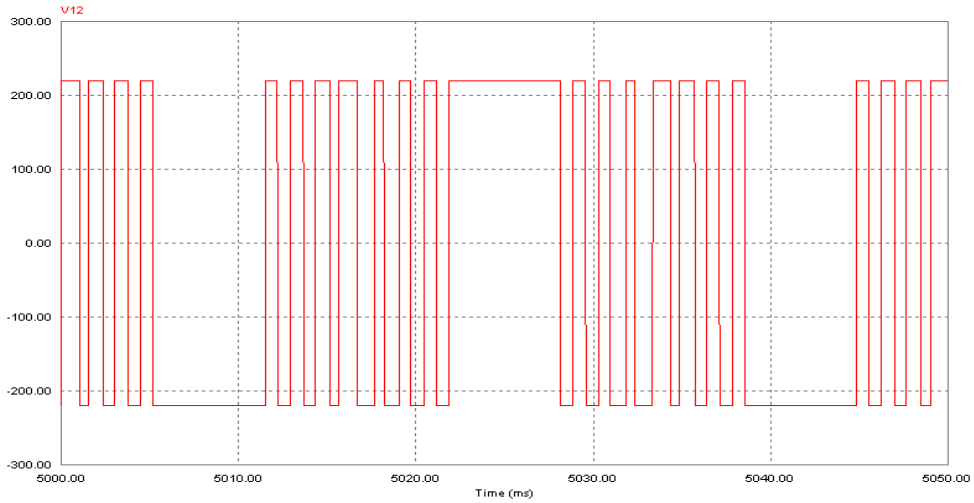
Figure 3.14 Comparaison entre l'angle exact et approximé pour m=7

a- 3 ème angle

b- 4 ème angle



a



b

Figure 3.15 Exemple de tension simple obtenues en simulation

a- im=0.5 m=7

b- im=0.6 m=7

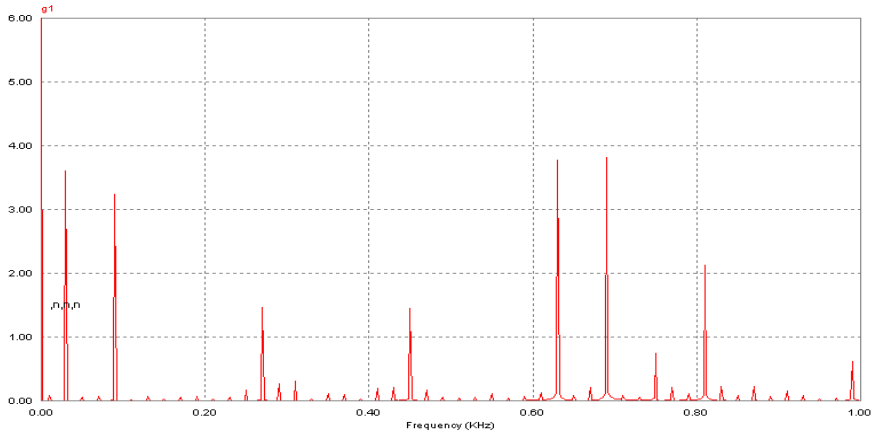


Figure 3.16 Spectre de la tension simple obtenue en simulation pour im=0.6 m=7

III.2.5 Conclusion

D'après le tableau 3.2 et la figure 3.14 on voit bien que les angles de commutation calculés par le réseau neuronale sont très proche des angles exacts et l'erreur maximale est inferieur à 0.006.

D'après la figure 3.15 on remarque que la tension est périodique de fréquence 30 Hz pour im=0.6 et 25 pour im=0.6, et en voit bien l'existence de sept angles de commutation par chaque quart de période. L'analyse spectrale (figure 3.16) montre l'élimination des harmonique 5, 7, 11, 13, 17et 19 pour m=7.

Donc on peut dire que notre algorithme MLI neuronale présente une grande précision dans le calcul des angles de commutation et il donne une efficacité dans l'élimination des harmoniques voulus.

Chapitre 4

IMPLEMENTATION DE L'ALGOTITHME MLI ON-LINE NEURONALE SUR FPGA

IV.1 Les circuits FPGA

IV.1.1 Introduction

L'électronique moderne se tourne de plus en plus vers le numérique qui présente de nombreux avantages sur l'analogique : grande insensibilité aux parasites, reconfigurabilité, facilité de stockage de l'information etc.

Aujourd'hui les techniques de traitement numérique occupent une place majeure dans tous les systèmes électroniques modernes grand public, professionnels ou de défense. De plus, les techniques de réalisation de circuits spécifiques, tant dans les aspects matériels (composants reprogrammables, circuits précaractérisés et bibliothèques de macrofonctions) que dans les aspects logiciels (placement-routage, synthèse logique) font désormais de la microélectronique une des bases indispensables pour la réalisation de systèmes numériques performants. Elle impose néanmoins une méthodologie de développement très structurée.

IV.1.2 Définition

Les FPGA (Field Programmable Gate Arrays ou "réseaux logiques programmables") sont des composants entièrement reconfigurables ce qui permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs. L'avantage de ce genre de circuit est sa grande souplesse qui permet de les réutiliser à volonté dans des algorithmes différents en un temps très court.

Le progrès de ces technologies permet de faire des composants toujours plus rapides et à plus haute intégration, ce qui permet de programmer des applications importantes. Cette technologie permet d'implanter un grand nombre d'applications et offre une solution d'implantation matérielle à faible coût pour des compagnies de taille modeste pour qui, le coût de développement d'un circuit intégré spécifique implique un trop lourd investissement.

IV.1.3 Application des FPGA

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

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

IV.1.4 Architecture des FPGA

Les circuits FPGA sont constitués d'une matrice de blocs logiques programmables entourés de blocs d'entrée sortie programmable. L'ensemble est relié par un réseau d'interconnexions programmable(figuire4.1).

Les FPGA sont bien distincts des autres familles de circuits programmables tout en offrant le plus haut niveau d'intégration logique.

Il existe actuellement plusieurs fabricants de circuits FPGA, Xilinx et Altera sont les plus connus, et plusieurs technologies et principes organisationnels. L'architecture, retenue par Xilinx, se présente sous forme de deux couches :

- une couche appelée circuit configurable.
- une couche réseau mémoire SRAM.

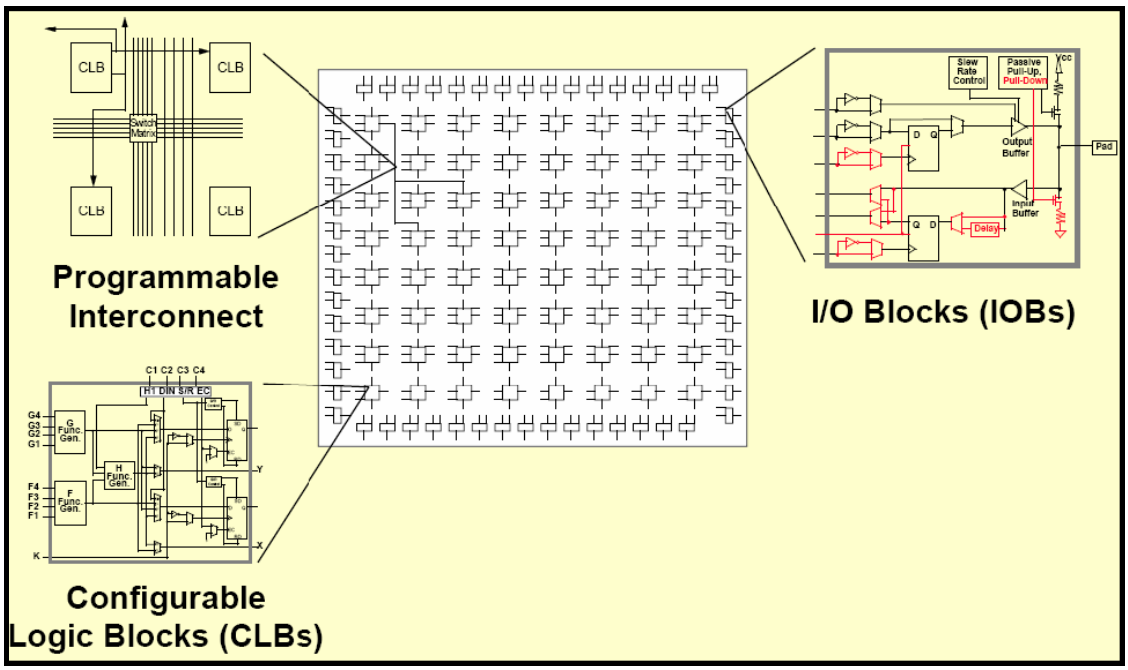


Figure 4.1 Architecture interne du FPGA

IV.1.4.1 Circuit configurable

La couche dite « circuit configurable » est constituée d'une matrice de blocs logiques configurables (CLB) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs d'entrées/sorties (IOB) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs.

La programmation du circuit FPGA, appelé aussi LCA (Logic Cells Arrays), consistera en l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ servant à interconnecter les éléments des CLB et des IOB, afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM.

IV.1.4.2 Réseau mémoire SRAM

La programmation d'un circuit FPGA est volatile, la configuration du circuit est donc mémorisée sur la couche réseau SRAM et stockée dans une ROM externe. Un dispositif interne permet à chaque mise sous tension de charger la SRAM interne à partir de la ROM. Ainsi on conçoit aisément qu'un même circuit puisse être exploité successivement avec des ROM différentes puisque sa programmation interne n'est jamais définitive. On voit tout le parti que l'on peut tirer de cette souplesse en particulier lors d'une phase de mise au point. Une erreur n'est pas rédhibitoire, mais peut aisément être réparée.

La mise au point d'une configuration s'effectue en deux temps : Une première étape purement logicielle va consister à dessiner puis simuler logiquement le circuit fini. Dans la seconde étape, on effectuera une simulation matérielle en configurant un circuit réel. On pourra alors vérifier si le fonctionnement réel correspond bien à l'attente du concepteur, et si besoin est identifier les anomalies liées généralement à des temps de transit réels légèrement différents de ceux supposés lors de la simulation logicielle, ce qui peut conduire à des états instables voire même erronés.

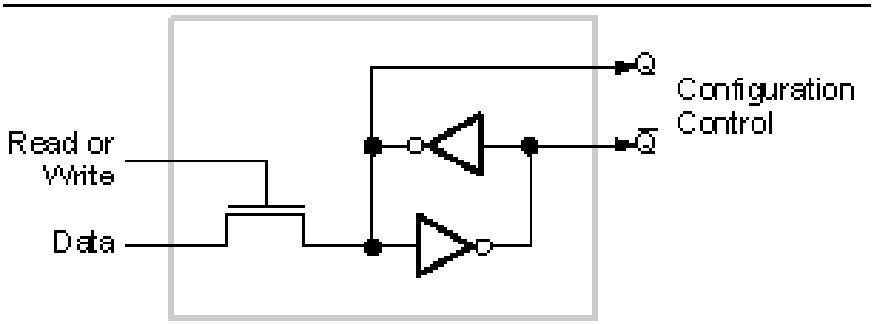


Figure 4.2 Structure d'un cellule SRAM

IV.1.4.3 Les CLB (configurable logic bloc)

Les blocs logiques configurables sont les éléments déterminants des performances du FPGA. Chaque bloc est composé d'un bloc de logique combinatoire composé de deux générateurs de fonctions à quatre entrées et d'un bloc de mémorisation synchronisation composé de deux bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB.

La LUT (Look Up Table) est un élément qui dispose de quatre entrées, il existe donc 2^4 = 16 combinaisons différentes de ces entrées. L'idée consiste à mémoriser la sortie correspondant à chaque combinaison d'entrée dans une petite table de 16 bits, la LUT devient ainsi un petit bloc générateur de fonctions. La figure ci-dessous montre le schéma simplifié d'un CLB de la famille XC4000 de Xilinx.

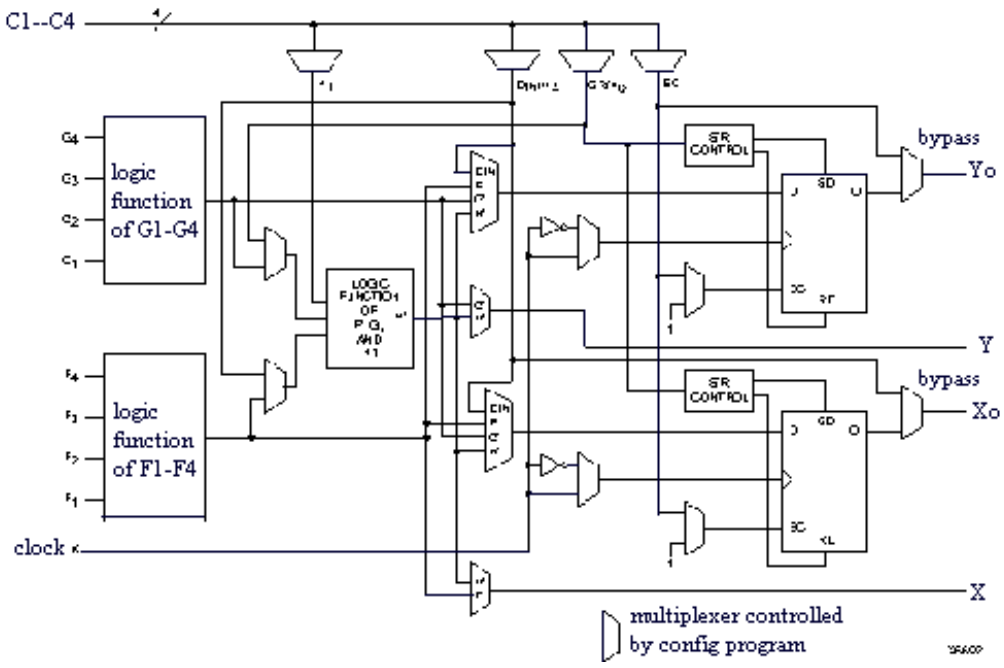


Figure 4.3 Cellules logiques (CLB)

Dans cette famille, la cellule de base contient deux LUT à 4 entrées qui peuvent réaliser deux fonctions quelconques à 4 entrées. Une troisième LUT peut réaliser une fonction quelconque à 3 entrées à partir des sorties des deux premières LUT (F' et G' qui deviennent H2 et H3) et d'une troisième variable d'entrée H1 sortant du bloc « sélecteur ». Le bloc sélecteur contient 4 signaux de contrôle : 3 signaux dédiées pour les registres : une donnée "Din", un signal de validation "Ec" et une remise à un ou à zéro asynchrone "S/R", et le $4^{ème}$ signal représente l'entrée H1 de la LUT à 3 entrées.

Les signaux des générateurs de fonction peuvent sortir du CLB, soit par la sortie X pour les fonctions F' et G', soit Y pour les fonctions G' et H'. Ainsi un CLB peut être utilisé pour réaliser :

- deux fonctions indépendantes à 4 entrées indépendantes ;
- ou une seule fonction à 5 variables ;
- ou deux fonctions, une à 4 variables et une autre à 5 variables.

Les sorties de ces blocs logiques peuvent être appliquées à des bascules au nombre de deux ou directement à la sortie du CLB (sorties X et Y). Chaque bascule présente deux modes de fonctionnement : un mode « flip-flop » avec comme donnée à mémoriser, soit l'une des fonctions F', G', H' ; soit l'entrée directe Din. La donnée peut être mémorisée sur un front montant ou descendant de l'horloge (Clk). Les sorties de ces deux bascules correspondent aux sorties du CLB X_0 et Y_0.

Un mode dit de « verrouillage » exploite l'entrée S/R qui peut être programmée soit en mode SET, mise à 1 de la bascule, soit en Reset, mise à zéro de la bascule. Ces deux entrées coexistent avec une autre entrée, qui n'est pas représentée sur la figure III.4, et qui est appelée le global Set/Reset. Cette entrée initialise le circuit FPGA à chaque mise sous tension, à chaque configuration, en commandant toutes les bascules au même instant soit à '1', soit à '0'. Elle agit également lors d'un niveau actif sur le fil RESET lequel peut être connecté à n'importe quelle entrée du circuit FPGA.

L'idée de cette architecture consiste à pouvoir modifier le contenu des mémoires des LUT en cours de fonctionnement. En effet, les LUT ne sont rien d'autre que des petites RAM qui étaient configurées au démarrage ; on peut alors les utiliser comme des petites mémoires de 16x1 bits. Un mode optionnel des CLB est donc la configuration en mémoire RAM de 16x2 bits ou 32x1 bit. Les entrées F1 à F4 et G1 à G4 deviennent des lignes d'adresses sélectionnant une cellule mémoire particulière. La fonctionnalité des signaux de contrôle est modifiée dans cette configuration : les lignes H1, Din et S/R deviennent respectivement les deux données D_0 et D_1 d'entrée (RAM 16x2bits) et le signal de validation d'écriture WE. Le contenu de la cellule mémoire (D_0 et D_1) est accessible aux sorties des générateurs de fonctions F' et G'. Ces données peuvent sortir du CLB à travers ses sorties X et Y ou alors en passant par les deux bascules.

L'intégration de fonctions à nombreuses variables diminue le nombre de CLB nécessaires et les délais de propagation des signaux ; par conséquent, elle augmente la densité et la vitesse du circuit. Le plus large circuit de cette famille (le circuit XC40250) dispose d'un réseau de 92x92 cellules de base et est équivalent à environ 250.000 portes logiques.

Xilinx propose également des composants "haute densité" avec les familles Virtex (4 millions de portes) et Virtex II (6 millions de portes). La famille Virtex apporte plusieurs nouveautés par rapport à la famille XC4000 :

- l'adjonction de blocs mémoires de 4 Kbits au coeur de la logique et même de plus larges mémoires dans la famille "Extended Memory".
- l'utilisation de boucles à verrouillage de phase améliorées : DLL (Digital Delay Locked Loop) qui permettent de synchroniser une horloge interne sur une horloge externe, de travailler en quadrature de phase et de multiplier ou diviser la fréquence.
- La présence d'un anneau de connexions autour du circuit pour faciliter le routage des entrées-sorties.

• La compatibilité avec de nombreux standards de transmission de données et de niveaux logiques.

L'architecture des cellules logiques est toujours basée sur des LUT à 4 entrées configurables également en petites mémoires RAM 16 bits. De plus, de la logique a été ajoutée pour permettre la synthèse de plus larges LUT comme une combinaison des LUT existantes. Le plus large circuit de cette famille (le circuit XCV3200E) contient un réseau de 104x156 cellules logiques et est équivalent à environ 4 millions de portes logiques.

Finalement, la famille Virtex II Pro améliore encore un peu le modèle précédent :

• Des blocs de mémoire de 18 Kbits.

• Des multiplieurs signés de 18x18 bits vers 36 bits.

IV.1.4.4 Les IOB (input output bloc)

Les blocs entrée/sortie permettent l'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé (haute impédance).

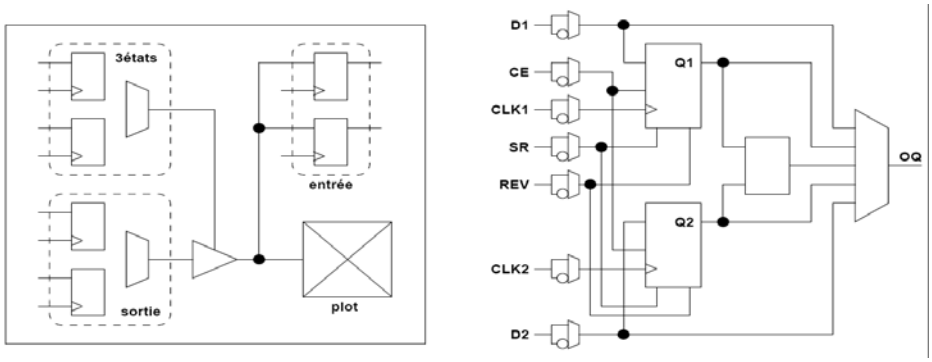


Figure 4.4: Input Output Block (IOB)

Les Ports d'entrée/sortie totalement programmables sont généralement composés de :

• Seuil d'entrée TTL ou CMOS

• Slew-rate programmable

• Buffer de sortie programmable en haute impédance

• Entrées et sorties directes ou mémorisées

• Inverseur programmable

IV.1.5 Les configurations en entrée et en sortie

IV.1.5.1 La configuration en entrée

Le signal d'entrée traverse un buffer qui selon sa programmation peut détecter soit des seuils TTL ou soit des seuils CMOS. Il peut être routé directement sur une entrée directe de la logique du circuit FPGA ou sur une entrée synchronisée. Cette synchronisation est réalisée à l'aide d'une bascule de type D, le changement d'état peut se faire sur un front montant ou descendant. De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (program controlled multiplexer). Un bit positionné dans une case mémoire commande ce dernier.

IV.1.5.2 La configuration en sortie

Nous distinguons les possibilités suivantes :
- inversion ou non du signal avant son application à l'IOB.
- synchronisation du signal sur des fronts montants ou descendants d'horloge.
- mise en place d'un " pull-up " ou " pull-down " dans le but de limiter la consommation des entrées/sorties inutilisées,
- signaux en logique trois états ou deux états. Le contrôle de mise en haute impédance et la réalisation des lignes bidirectionnelles sont commandés par le signal de commande Out Enable lequel peut être inversé ou non. Chaque sortie peut délivrer un courant de 12mA. Ainsi toutes ces possibilités permettent au concepteur de connecter au mieux une architecture avec les périphériques extérieurs.

IV.1.6 Les interconnexions

Les connexions internes dans les circuits FPGA sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM. Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible. Pour parvenir à cet objectif, Xilinx propose trois sortes d'interconnexions selon la longueur et la destination des liaisons.

IV.1.6.1 Les interconnexions à usage général

Ce système fonctionne en une grille de cinq segments métalliques verticaux et quatre segments horizontaux positionnés entre les rangées et les colonnes de CLB et de l'IOB. Des aiguilleurs appelés aussi matrices de commutation sont situés à chaque intersection. Leur rôle est de raccorder les segments entre eux selon diverses configurations, ils assurent ainsi la communication des signaux d'une voie sur l'autre. Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre. Pour éviter que les signaux traversant les grandes lignes ne soient affaiblis, nous trouvons généralement des buffers implantés en haut et à droite de chaque matrice de commutation.

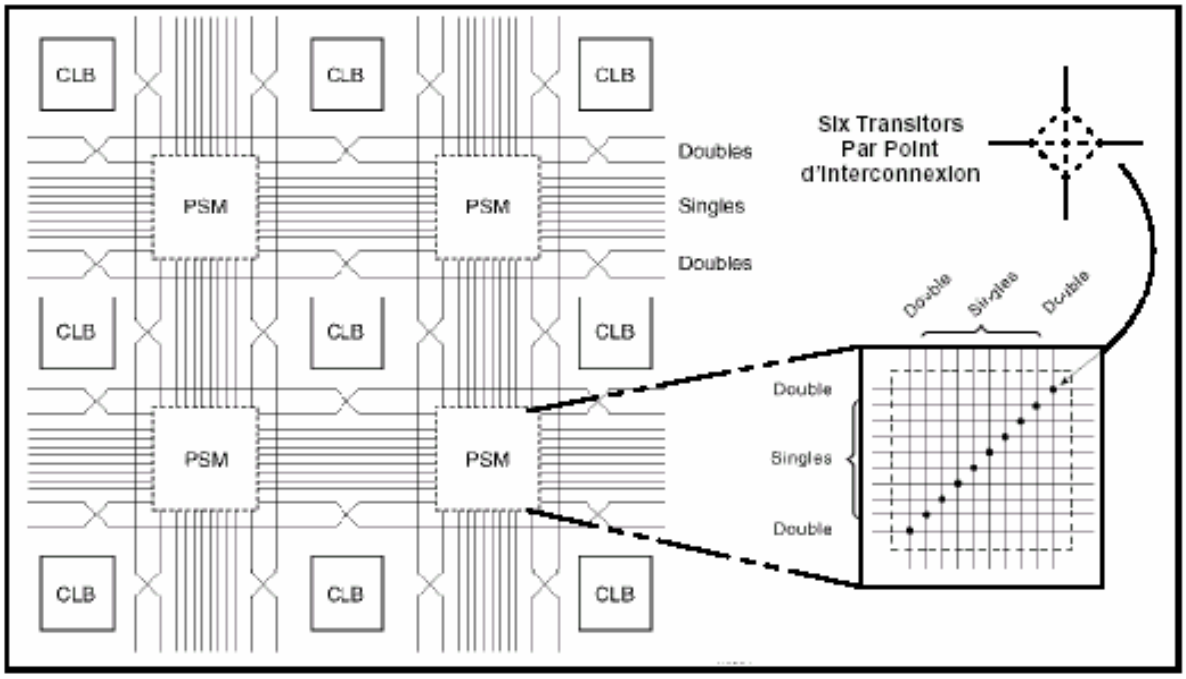


Figure 4.5 : Connexions à usage général et matrice de commutation

IV.1.6.2 Les interconnexions directes

Ces interconnexions permettent l'établissement de liaisons entre les CLB et les IOB avec un maximum d'efficacité en terme de vitesse et d'occupation du circuit. De plus, il est possible de connecter directement certaines entrées d'un CLB aux sorties d'un autre.

Pour chaque bloc logique configurable, la sortie X peut être connectée directement aux entrées C ou D du CLB situé au-dessus et les entrées A ou B du CLB situé au-dessous. Quant à la sortie Y, elle peut être connectée à l'entrée B du CLB placé immédiatement à sa droite. Pour chaque bloc logique adjacent à un bloc entrée/sortie, les connexions sont possibles avec les entrées I ou les sorties O suivant leur position sur le circuit.

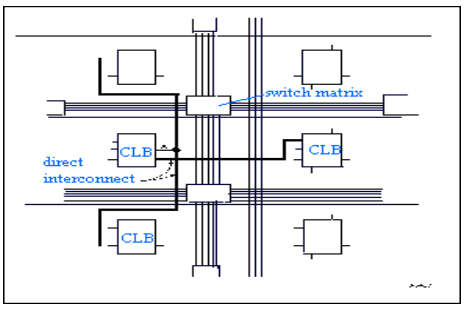


Figure 4.6 : Les interconnexions directes

IV.1.6.3 Les longues lignes

Les longues lignes sont de longs segments métallisés parcourant toute la longueur et la largeur du composant, elles permettent éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments dans le but d'assurer un synchronisme aussi parfait que possible. De plus, ces longues permettent d'éviter la multiplicité des points d'interconnexion.

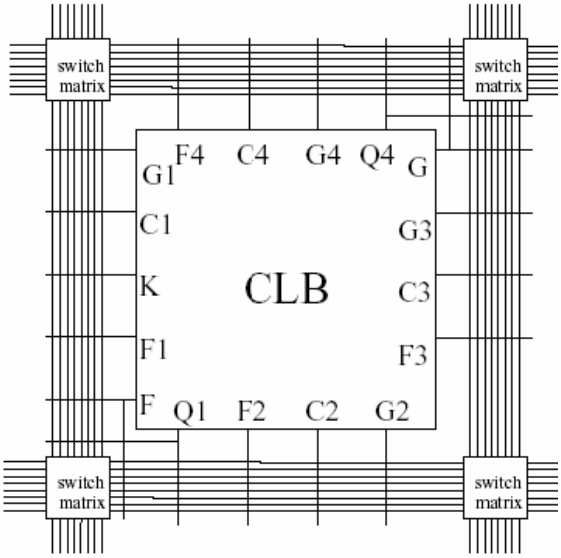


Figure 4.7 : Les longues lignes

IV.1.7 les outils de développement des FPGA

Les outils de développement permettent au concepteur de programmer le circuit à partir de la description de la fonction à réaliser. Cette description peut être textuelle dans ce cas on utilise généralement des langages de développement : le Verilog et le VHDL. Cette description peut aussi être graphique et ce au moyen de chronogrammes, de graphes d'état et de symboles de fonction.

Pour développer notre application on utilise une carte VERTEX-II qui serra décrite dans le paragraphe suivante, et on utilise le langage de description VHDL qui serra décrite dans le paragraphe 3.

IV.2 La carte de développement Virtex-II V2MB1000 de Memec Design

Le kit de développement Virtex-II V2MB1000 de Memec Design, qu'on a utilisé pour développer notre application, fournit une solution complète de développement d'applications sur la famille Virtex-II de Xilinx. Il utilise le circuit « FPGA XC2V1000-4FG456C » qui appartient à la famille Virtex-II de Xilinx et qui est équivalent à 1 million de portes logiques. La haute densité d'intégration des portes ainsi que le nombre important d'entrées/sorties disponibles à l'utilisateur permettent d'implémenter des systèmes complets de solutions sur la plate forme FPGA. La carte de développement inclue aussi une mémoire 16M x 16 DDR, deux horloges, un port série RS-232 et des circuits de support additionnels. Une interface LVDS est disponible avec un port de transmission 16-bit et un port de réception 16-bit, en plus de signaux d'horloge, d'état et de contrôle pour chacun de ces ports. La carte supporte également le module d'expansion Memec Design P160, qui permet d'ajouter facilement des modules pour des applications spécifiques.

La famille FPGA Virtex-II possède les outils avancés pour répondre à la demande à des applications de haute performance. Le kit de développement Virtex-II fournit une excellente plateforme pour explorer ces outils, l'utilisateur peut alors utiliser toutes les ressources disponibles avec rapidité et efficacité.

IV.2.1 Description de la carte de développement

La carte de développement Virtex-II contient le circuit FPGA **XC2V1000-4FG456C**. Ce circuit fait partie de la famille Virtex-II, qui est une famille de circuits développés pour des applications haute performance telles que les télécommunications, l'imagerie et les applications DSP. Il possède 456 broches dont 324 peuvent être utilisées en entrées/sorties. Il se compose d'une matrice de 40x32 CLB et il contient un total de 10.240 LUT et 10.240 bascules « flip-flop ». Sa capacité maximale en Select RAM est de 163.840 bits **[9]**.

La carte contient également une mémoire DDR de 32MB. Elle présente 2 générateurs d'horloges internes, générant des signaux d'horloge à 100MHz (CLK.CAN2) et 24MHz (CLK.CAN1).

Elle contient aussi circuit de remise à zéro « Reset » activé par un bouton poussoir (SW3), un bouton poussoir « PROGn » pour initialiser la configuration et charger le contenu de la PROM dans le circuit FPGA (SW2) ainsi que deux boutons poussoirs (SW5 et SW6) qui peuvent être utilisés pour générer des signaux actifs.

Deux afficheurs 7 segments à cathode commune sont présents sur la carte, et peuvent être utilisés durant la phase de test et de debugging. Il existe aussi 8 entrées exploitables par l'utilisateur (DIP switch) qui peuvent être mis statiquement à un état haut ou bas.

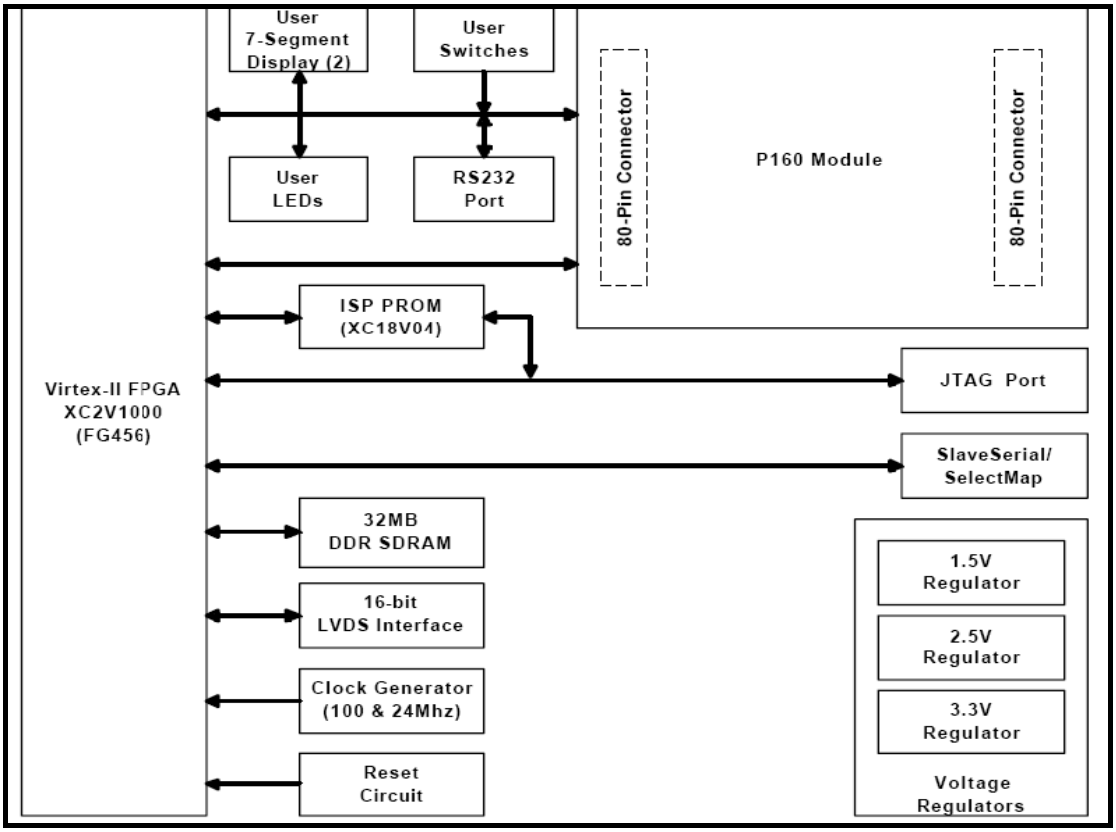


Figure 4.8 : Diagramme de la carte de développement

La carte possède une interface RS232, une interface JTAG pour programmer l'ISP PROM et configurer le circuit FPGA ainsi qu'un connecteur de câble parallèle IV qui peut aussi être utilisé pour configurer le FPGA via la configuration Maître/Esclave.
Il existe également des régulateurs internes de tension qui génèrent, à partir de l'alimentation principale de 5,0V, des tensions internes d'alimentation à 1,5V, 2,5V et 3,3V. Les entrées/sorties sont regroupées dans 8 différents groupes, et chaque groupe peut être configuré pour opérer dans le mode 2,5V ou 3,3V.

La carte de développement peut être configurée pour travailler en mode Master Serial, Slave Serial, Master SelectMap, Slave SelectMap ou JTAG selon la position des jumpers M0, M1, M2 et M3.

IV.2.2 Chargement du programme sur la carte de développement

La carte de développement Virtex-II supporte plusieurs méthodes de configuration de son circuit FPGA. Le port JTAG peut être utilisé directement pour configurer le FPGA, ou pour programmer l'ISP PROM. Une fois l'ISP PROM programmée, elle peut être utilisée pour configurer le FPGA. Le port SelectMap/Slave Serial sur cette carte peut aussi être

utilisé pour configurer le FPGA. La figure suivante montre l'installation pour toutes les configurations de modes supportés par la carte de développement Virtex-II.

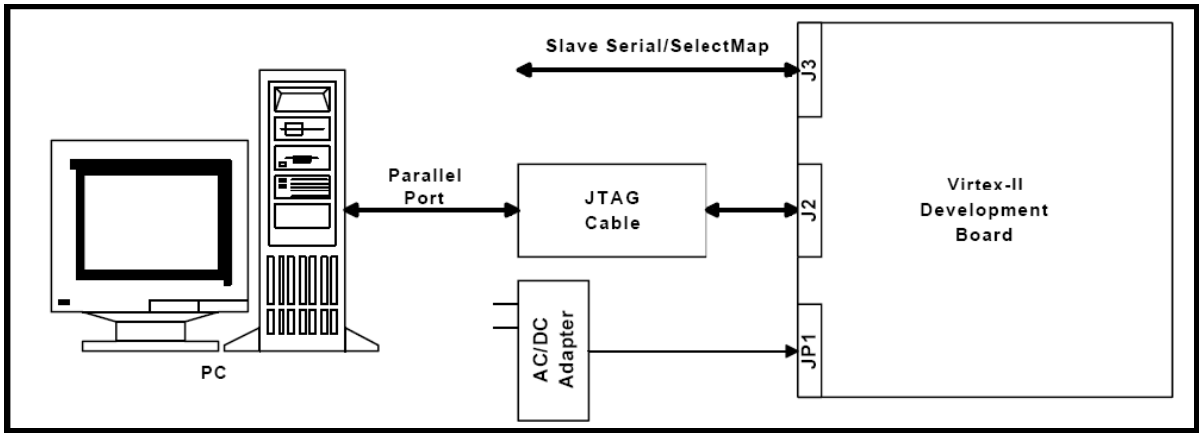


Figure4.9 : Chargement du programme sur la carte

IV.2.3 Utilisation de l'interface JTAG

Le câble Memec Design JTAG est connecté d'un coté à la carte de développement, et de l'autre au port série du PC. On utilise alors l'outil de programmation du JTAG de Xilinx (iMPACT) pour charger le programme binaire soit directement sur le circuit FPGA en mode JTAG, soit sur l'ISP PROM en mode Master Serial ou Master SelectMap, dans ce dernier cas il faut appuyer sur le bouton poussoir PROGn (SW2) pour initialiser la configuration dans le circuit FPGA.

IV.2.4 Utilisation de l'interface Slave Serial

Dans ce mode, une source externe fournit la configuration bitstream et la configuration clock au circuit FPGA Virtex-II. Là aussi, le programme peut être directement chargé sur le circuit FPGA, ou bien sur l'ISP PROM, et dans ce cas il faut utiliser le bouton PROGn pour initialiser le FPGA.

IV.3 Langage de description VHDL

IV.3.1 Bref historique

Le VHDL - HSIC (Very High Speed Integrated Circuit) Hardware Description Language , a été demandé par le DOD (Département de la défense américain) pour décrire les circuits complexes, de manière à établir un langage commun avec ses fournisseurs. C'est un langage, standard IEEE 1076 depuis 1987, qui aurait dû assurer la portabilité du code pour différents outils de travail (simulation, synthèse pour tous les circuits et tous les

fabricants). Malheureusement, ce n'est pour l'instant pas le cas, bien que plusieurs vendeurs aient tendance à se rapprocher du VHDL utilisé par Synopsis.

Une mise à jour du langage VHDL s'est faite en 1993 (IEEE 1164) et en 1996, la norme 1076.3 a permis de standardiser la synthèse VHDL.

IV.3.2 Utilité du VHDL :

Le VHDL est un langage de spécification, de simulation et également de conception. Contrairement à d'autres langages (CUPL, ABEL) qui se trouvaient être en premier lieu des langages de conception, VHDL est d'abord un langage de spécification. La normalisation a d'abord eu lieu pour la spécification et la simulation (1987) et ensuite pour la synthèse (1993). Cette notion est relativement importante pour comprendre le fonctionnement du langage et son évolution. Grâce à la normalisation, on peut être certain qu'un système décrit en VHDL standard est lisible quel que soit le fabricant de circuits. Par contre, cela demande un effort important aux fabricants de circuits pour créer des compilateurs VHDL adaptés à et autant que possible optimisés pour leurs propres circuits.

IV.3.3 Spécification :

Etabli en premier lieu pour de la spécification, c'est dans ce domaine que la norme est actuellement la mieux établie. Il est tout-à-fait possible de décrire un circuit en un VHDL standard pour qu'il soit lisible de tous. Certains fabricants (de circuits ou de CAO) adaptent ce langage pour donner à l'utilisateur quelques facilités supplémentaires, au détriment de la portabilité du code. Heureusement, il y a une nette tendance de la part des fabricants à revoir leurs positions et à uniformiser le VHDL. Le rapprochement se fait, comme précité, autour du VHDL de Synopsis. Il est donc probable que l'on s'approche d'un vrai standard VHDL et non plus d'un standard théorique. Il y aura toujours des ajouts de la part des fabricants, mais il ne s'agira plus d'une modification du langage (aussi légère soit elle), mais de macros offertes à l'utilisateur pour optimiser le code VHDL en fonction du circuit cible (en vue de la synthèse).

Cette possibilité de décrire des circuits dans un langage universel est aussi très pratique pour éviter les problèmes de langue. De longues explications dans une langue peuvent ainsi être complétées par du code VHDL pour en faciliter la compréhension.

IV.3.4 Simulation :

Le VHDL est également un langage de simulation. Pour ce faire, la notion de temps, sous différentes formes, y a été introduite. Des modules, destinés uniquement à la

simulation, peuvent ainsi être créés et utilisés pour valider un fonctionnement logique ou temporel du code VHDL. La possibilité de simuler avec des programmes VHDL devrait considérablement faciliter l'écriture de tests avant la programmation du circuit et éviter ainsi de nombreux essais sur un prototype qui sont beaucoup plus coûteux et dont les erreurs sont plus difficiles à trouver. Bien que la simulation offre de grandes facilités de test, il est toujours nécessaire de concevoir les circuits en vue des tests de fabrication, c'est-à-dire en permettant l'accès à certains signaux internes.

IV.3.5 Conception :

Le VHDL permet la conception de circuits avec une grande quantité de portes. Les circuits actuels comprennent, pour les FPGA par exemple, entre 500 et 1000'000 portes et ce nombre augmente très rapidement. L'avantage d'un langage tel que celui-ci par rapport aux langages précédents de conception matérielle est comparable à l'avantage d'un langage informatique de haut niveau (Pascal, ADA, C) vis-à-vis de l'assembleur. Ce qui veut aussi dire que malgré l'évolution fulgurante de la taille des circuits, la longueur du code VHDL n'a pas suivi la même courbe. Cependant, ce langage étant conçu en premier lieu pour de la spécification, certaines variantes du langage ne sont pour l'instant pas utilisables pour la conception. Il faut noter que lorsqu'il s'agit de concevoir quelque chose en VHDL, il ne faut pas le faire tête baissée. Le VHDL, bien que facilement accessible dans ses bases, peut devenir extrêmement compliqué s'il s'agit d'optimiser le code pour une architecture de circuit. C'est pour cette raison que de plus en plus de fabricants offrent des macros, gratuites pour les fonctions sans grandes difficultés et payantes pour les autres. Donc avant de concevoir une ALU, un processeur RISC, une interface PCI ou d'autres éléments de cette complexité, il peut être judicieux de choisir un circuit cible en fonction des besoins et d'acheter la macro offerte par le constructeur. Il est bien évident qu'il faudra évaluer les besoins (performance du code nécessaire, quantité de pièces à produire) et le coût d'une telle macro.

IV.3.6 La Synthèse :

Deux étapes particulières apparaissent dans l'utilisation du VHDL, celle de la spécification et de la synthèse. En VHDL, il convient de bien distinguer ces deux étapes, car le code écrit pour l'une ou l'autre n'est pour l'instant pas complètement identique. Avant de commencer l'apprentissage du VHDL, il est donc nécessaire de les définir. La spécification est la partie d'un développement qui consiste à valider par la simulation ce qui a été demandé par le mandataire. Elle permet de corriger un cahier des charges

incorrect ou de compléter celui-ci. La spécification en VHDL permet de créer une sorte de maquette qui a, vu de l'extérieur, le comportement du système désiré. La synthèse permet de générer automatiquement à partir du code VHDL un schéma de câblage permettant la programmation du circuit cible. La description du code pour une synthèse est, en théorie, indépendante de l'architecture du circuit. En pratique, le style utilisé aura une influence sur le résultat de la synthèse, influence liée au type de circuit et au synthétiseur utilisé. Le code devra être optimisé pour un type circuit. Si le besoin d'optimisation n'est pas très important, cette partie peut se dérouler très rapidement, alors qu'au contraire, si le besoin d'optimisation est grand, les subtilités pour coder de manière adéquate en VHDL obligeront le concepteur à y consacrer beaucoup de temps. D'ailleurs, ce n'est souvent plus en VHDL que l'on optimisera, car ce langage se révèle, pour l'instant en tout cas, peu adapté à cet usage. Il faudra travailler à un niveau plus proche du circuit ou acheter des macros pré-existantes.

IV.3.7 Structure d'une description VHDL simple

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

● **L'entité**

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

● **L'architecture**

L'architecture (ARCHITECTURE) contient les instructions VHDL permettant de décrire et de réaliser le fonctionnement attendu. Elle représente la structure interne de la description.

Les entités et architectures sont des unités de conception dites primaire et secondaire. Un couple entité-architecture donne une description complète d'un élément ; ce couple est appelé modèle.

IV.3.7.1 Déclaration des bibliothèques

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

Exemple : `Library ieee;`

```
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
```

IV.3.7.2- Déclaration de l'entité et des entrées/sorties

Cette opération permet de définir le nom de la description VHDL ainsi que les entrées et sorties utilisées, l'instruction qui les définit est **port** :

Exemple :
```
entity ana is
port (DEC :in std_logic_vector(3 downto 0);
SEG:out std_logic_vector(6 downto 0)
);
end ana ;
```

Le nom du signal est composé d'une chaîne de caractères dont le premier est une lettre. Le langage VHDL n'est pas sensible à la « casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.

Le signal peut être défini en entrée (**in**), en sortie (**out**), en entrée/sortie (**inout**) ou en **buffer**, c'est-à-dire qu'il est en sortie mais il peut être utilisé comme entrée à l'intérieur de la description. L'affectation des broches d'entrées/sorties se fait par la définition d'attributs supplémentaires qui dépendent du logiciel de développement utilisé.

IV.3.8 Les deux modes de travail en VHDL

Le VHDL utilise deux modes de fonctionnement : le mode combinatoire (ou concurrent) et le mode séquentiel. Chacun de ces modes est utilisé dans des cas bien précis.

IV.3.8.1 Le mode combinatoire

En mode combinatoire (ou concurrent), toutes les instructions d'une description VHDL sont évaluées et affectent les signaux de sortie en même temps, l'ordre dans lequel les instructions sont écrites n'a donc aucune importance. En effet la description génère des structures électroniques, c'est la grande différence entre une description VHDL et un langage informatique classique.

Dans un système à microprocesseur, les instructions sont exécutées les unes à la suite des autres. Avec VHDL il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description, cela n'est pas toujours évident.

IV.3.8.2 Le mode séquentiel

Le mode séquentiel utilise les process dans lesquels le temps est une variable essentielle. Un process est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement, c'est-à-dire les unes à la suite des autres. Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standards de la programmation structurée comme dans les systèmes à microprocesseurs.

Dans le mode séquentiel, on distingue :

• **Le mode séquentiel asynchrone,** dans lequel les changements d'état des sorties peuvent se produire à des instants difficilement prédictibles (retards formés par le basculement d'un nombre souvent inconnu de fonctions),

• **Le mode séquentiel synchrone,** dans lequel les changements d'état des sorties se produisent tous à des instants quasiment connus (un temps de retard après un front actif de l'horloge).

IV.4 Etapes nécessaires au développement d'un projet sur FPGA

Le développement en VHDL nécessite l'utilisation de deux outils : le simulateur et le synthétiseur. Le premier va nous permettre de simuler notre description VHDL avec un fichier de simulation appelé « test-bench » ; cet outil interprète directement le langage VHDL et il comprend l'ensemble du langage. L'objectif du synthétiseur est très différent : il doit traduire le comportement décrit en VHDL en fonctions logiques de bases, celles-ci dépendent de la technologie choisie ; cette étape est nommée « synthèse ». L'intégration finale dans le circuit cible est réalisée par l'outil de placement et routage. Celui-ci est fourni par le fabricant de la technologie choisie.

Le langage VHDL permet d'écrire des descriptions d'un niveau comportemental élevé. La question est de savoir si n'importe quelle description comportementale peut être traduite en logique ?

Avec les outils actuels, il est possible de disposer de fichiers VHDL à chaque étape. Le même fichier de simulation (test-bench) est ainsi utilisable pour vérifier le fonctionnement de la description à chaque étape de la procédure de développement. La figure 4.10 donne les différentes étapes nécessaires au développement d'un projet sur circuit FPGA.

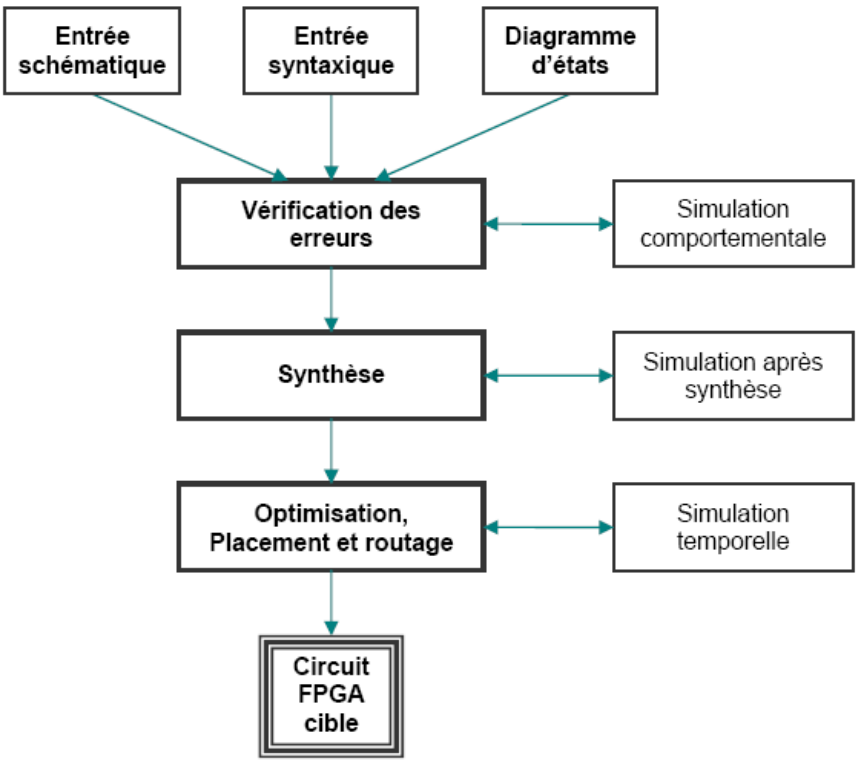


Figure 4.10 : Organisation fonctionnelle de développement d'un projet sur circuit FPGA

IV.4 .1 Saisie du texte VHDL

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

La figure 4.11 montre comment se présente le logiciel «ISE Xilinx Project Navigator».

La saisie du texte se fait sur la partie droite de l'écran, on voit en haut à gauche la hiérarchie du projet, et en bas à gauche les nombreux outils nécessaires tout au long du développement du projet.

Il faut commencer par crée un projet, ensuite inclure des fichiers sources dans lesquels il faut saisir le texte VHDL désiré. On peut inclure autant de sources qu'on veut dans un projet.

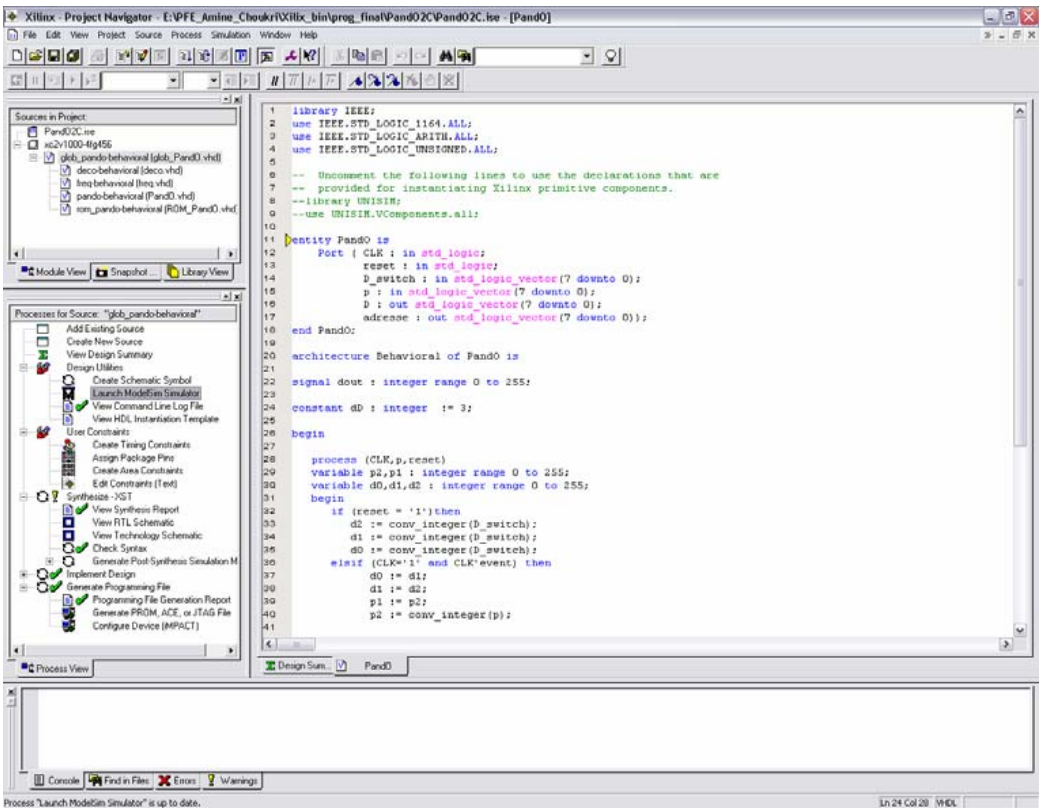


Figure 4.11 : Vue d'ensemble du logiciel « Xilinx Project Navigator »

IV.4.2 Vérification des erreurs

Cette étape est effectuée en appuyant sur le bouton « check syntax ». Elle permet de vérifier les erreurs (errors) de syntaxe du texte VHDL et d'afficher les différentes alarme « warnings » liées au programme, par exemple des signaux déclarés mais non utilisés dans le programme. S'il y'a des erreurs dans le programme, il ne peut pas être synthétisé, mais la présence d'alarmes n'empêche pas de poursuivre normalement les autres étapes du développement.

Cette étape permet donc de valider la syntaxe du programme et de générer la « netlist », qui est un fichier contenant la description de l'application sous forme d'équations logiques.

IV.4.3 Synthèse

La synthèse permet de réaliser l'implémentation physique d'un projet. Le synthétiseur a pour rôle de convertir le projet, en fonction du type du circuit FPGA cible utilisé, en portes logiques et bascules de base. L'outil « View RTL Schematic » permet de visualiser les schémas électroniques équivalents générés par le synthétiseur (figure4.12).

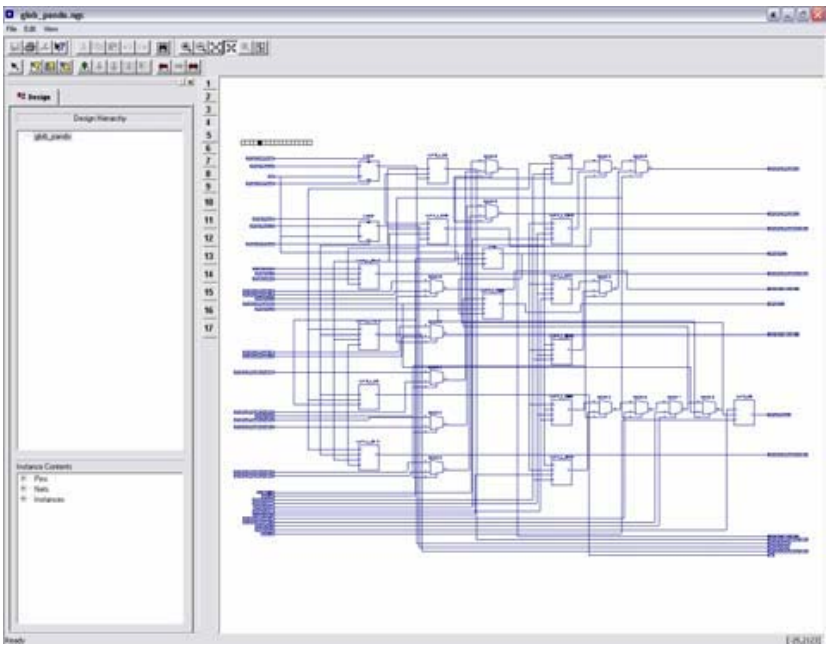


Figure 4.12 : Aperçu de l'outil « View RTL Schematic »

De plus, le synthétiseur permet à l'utilisateur d'imposer des contraintes de technologie (User constraints) : par exemple fixer la vitesse de fonctionnement (Create Timing Constraints), délimiter la zone du circuit FPGA dans laquelle le routage doit se faire (Create Area constraints) ou affecter les broches d'entrées/sorties (Assign Package Pins). La figure 4.13 montre un aperçu de l'outil d'assignation des broches d'entrées/sorties.

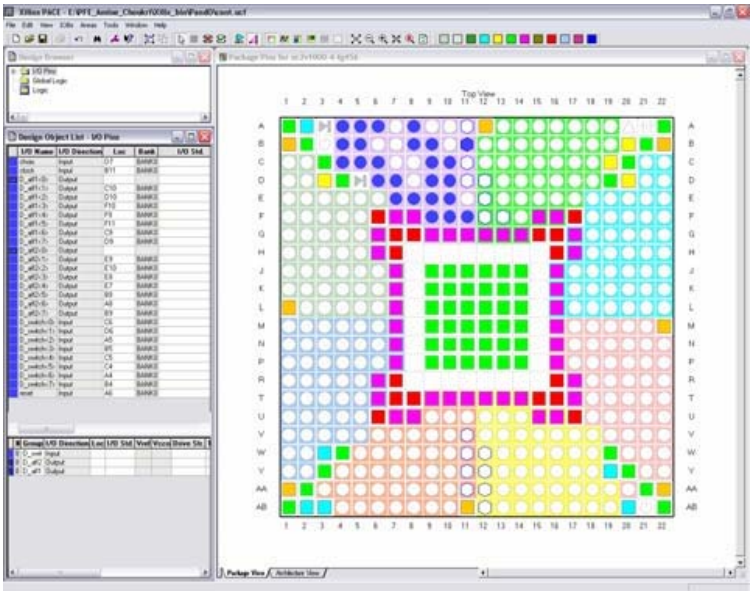


Figure 4.13 : Aperçu de l'outil d'affectation des broches d'entrées/sorties

IV.4.4- Simulation

Le simulateur utilisé est le « ModelSim Simulator » (figure 4.14). La simulation permet de vérifier le comportement d'un design avant ou après implémentation dans le composant cible. Elle représente une étape essentielle qui nous fera gagner du temps lors de la mise au point sur la carte. Il faut juste noter qu'un projet peut être simulé même s'il n'est pas synthétisable.

Lors de l'étape de simulation comportementale, on valide l'application indépendamment de l'architecture et des temps de propagation du futur circuit cible. La phase de simulation après synthèse valide l'application sur l'architecture du circuit cible, et enfin la simulation temporelle prend en compte les temps de propagation des signaux à l'intérieur du circuit FPGA cible.

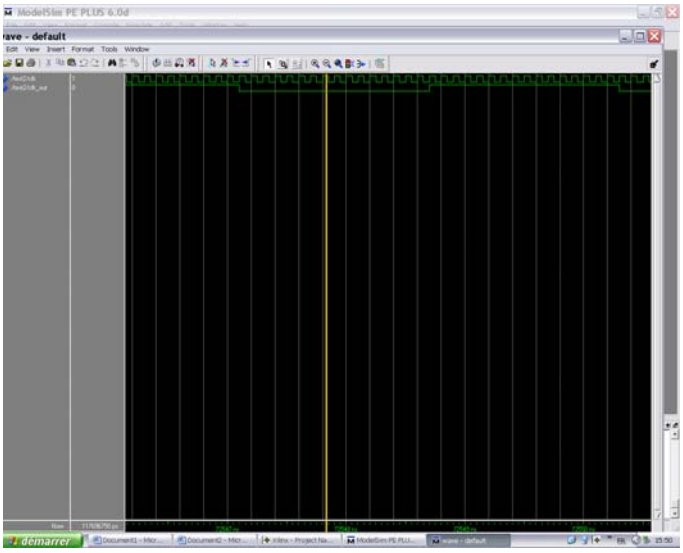


Figure 4.14 : Présentation du simulateur « ModelSim Simulator »

IV.4.5- Optimisation, placement et routage

Pendant l'étape d'optimisation, l'outil cherche à minimiser les temps de propagation et à occuper le moins d'espace possible sur le circuit FPGA cible. Le placement et routage permet de tracer les routes à suivre sur le circuit afin de réaliser le fonctionnement attendu. La figure 4.15 donne un aperçu de l'outil de placement et routage « FPGA Editor » qui permet de visualiser et d'éditer le circuit routé.

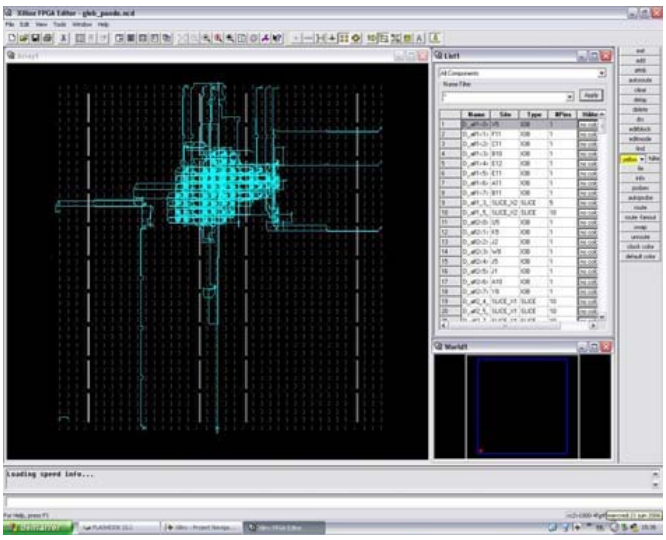


Figure 4.15 : Aperçu de l'outil « FPGA Editor »

IV.4.6- Programmation du composant et test

Dans cette dernière étape (Generate Programming Files), on génère le fichier à charger sur le circuit FPGA à travers l'interface JTAG. Une fois le programme chargé sur le circuit, on peut tester et visualiser les résultats directement sur la carte de développement Virtex-II à travers les nombreuses interfaces qu'elle offre, soit directement sur les deux afficheurs 7 segments, soit à travers l'interface RS232 ou bien à travers l'interface LVDS 16 bits.

IV.5 Implémentation de la commande sur FPGA

IV.5.1 Introduction

Le but de notre projet est d'implémenter la commande MLI on-line Neuronale, qui a été décrite dans la partie précédente, sur un circuit FPGA. Dans une première étape on propose une fonction qui calcul les angles de commutation, et dans l'étape suivante on fait un programme qui à partir des angles de commutation génère les trois signaux de commande. Dans la dernière étape on propose un programme pour tester notre commande sur la carte de développement Memec Design V2MB1000.

IV.5.2 Le calcul des angles de commutation

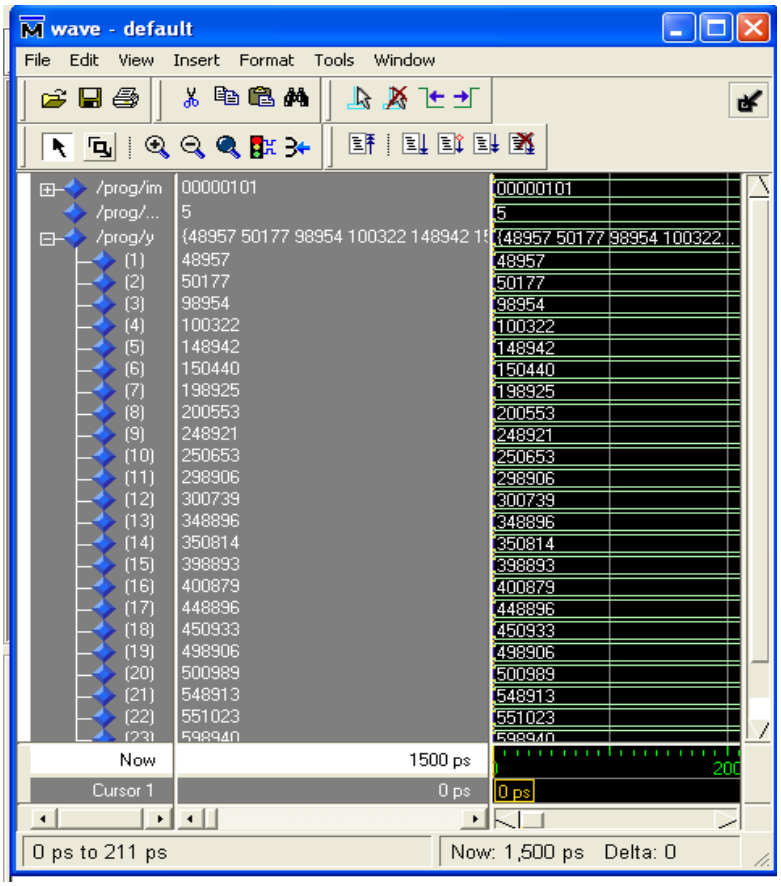


Figure 4.16 Exemple d'angles de commutation calculés sur FPGA pour im=0.05 et m=23

angles exacts	Angles approximés	Erreur
4,893218114	4,8957	-0,00248189
5,017466166	5,0177	-0,00023383
9,893002362	9,8954	-0,00239764
10,03224181	10,0322	4,1813E-05
14,8919143	14,8942	-0,0022857
15,04508678	15,044	0,00108678
19,8904682	19,8925	-0,0020318
20,05640871	20,0553	0,00110871
24,88899896	24,8921	-0,00310104
25,06644501	25,0653	0,00114501
29,88773909	29,8906	-0,00286091
30,07534046	30,0739	0,00144046
34,88685759	34,8896	-0,00274241
35,0831866	35,0814	0,0017866
39,88648131	39,8893	-0,00281869
40,09004372	40,0879	0,00214372
44,88670766	44,8896	-0,00289234
45,09595409	45,0933	0,00265409
49,88761259	49,8906	-0,00298741
50,10095028	50,0989	0,00205028
54,88925591	54,8913	-0,00204409
55,1050609	55,1023	0,0027609
59,89168523	59,894	-0,00231477

Tableau 4.1 Angles exacts et approximés (sur FPGA) pour im=0 .05 et m=23

D'après le tableau 4.1 et la figure 4.3 on voit bien que les angles de commutation calculés par le programme sont très proche des angles exacts et l'erreur maximale est inferieure à 0.003.

IV.5.3 Programmation de la commande MLI on-line neuronale

IV.5.3.1 Schéma block

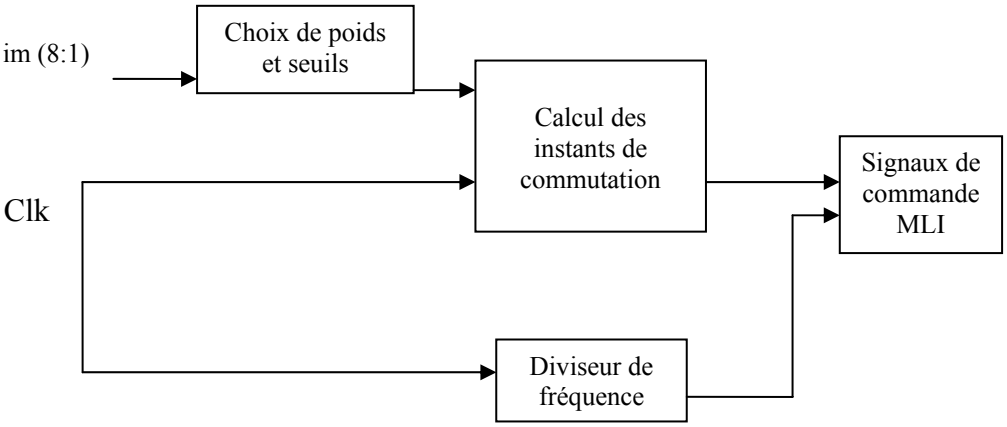


Figure 4.17 Schéma synoptique du programme

IV.5.3.2 Résultats de simulation

L'outil de simulation utilisé est MODELSIM; les figures 4.18, 4.19 et 4.20 donnent quelques exemples des résultats obtenus.

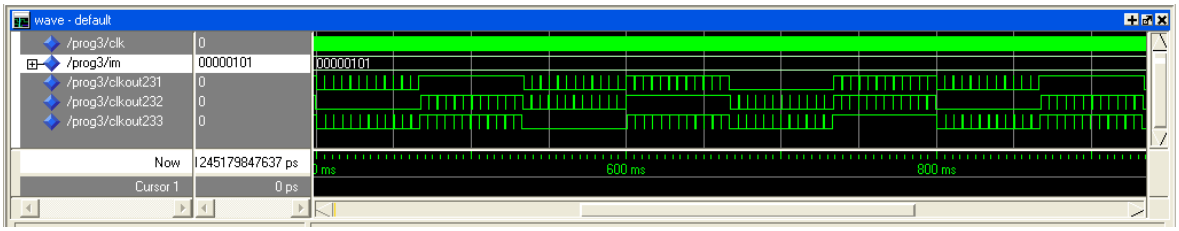


Figure 4.18 Les signaux MLI obtenus par simulation pour im=0.05 m=23

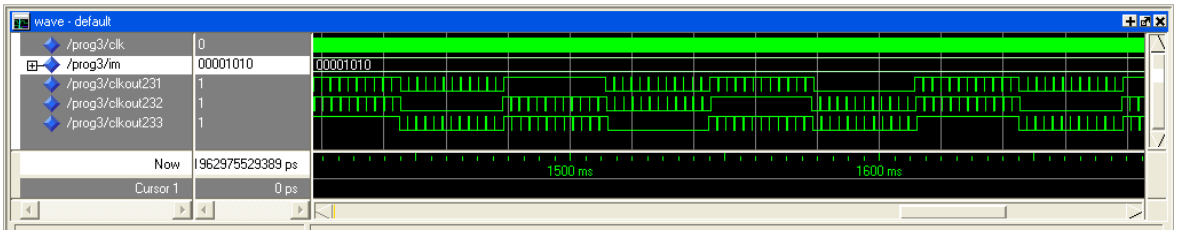


Figure 4.19 Les signaux MLI obtenus par simulation pour im=0.1 m=23

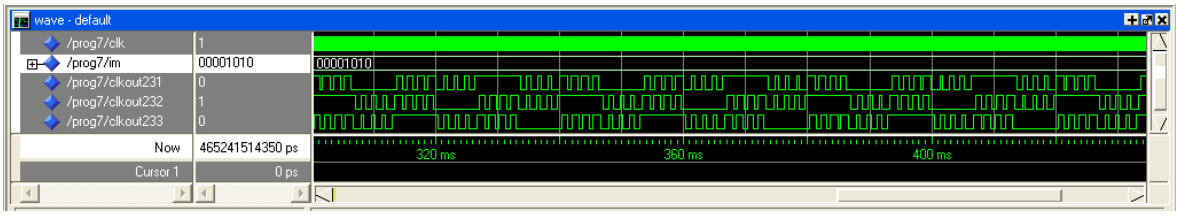
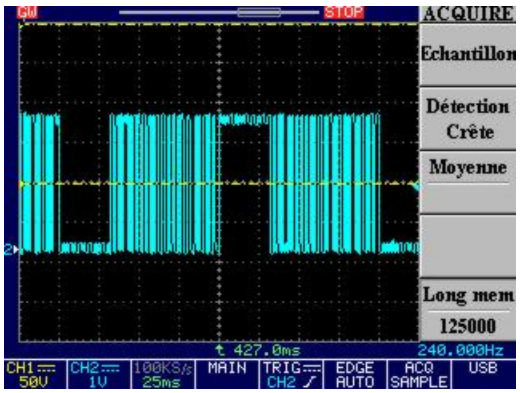


Figure 4.20 Les signaux MLI obtenus par simulation pour im=0.5 m=7

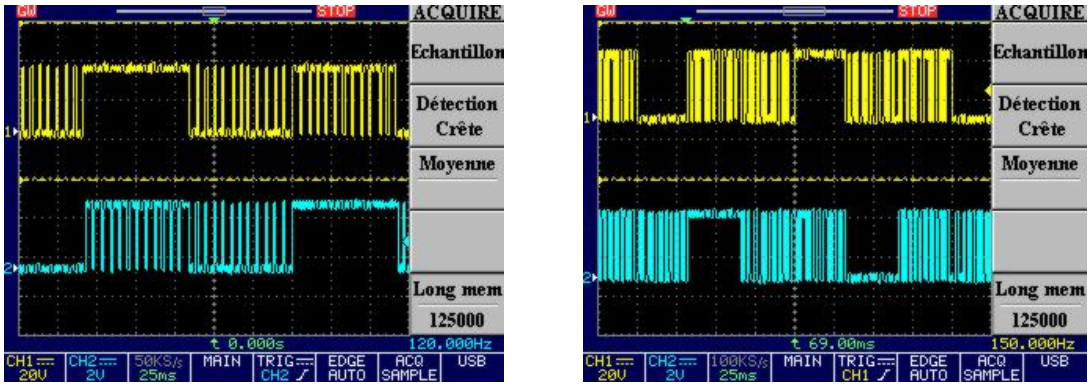
IV.5.4 Implémentation sur la carte de développement

Pour vérifier le fonctionnement de notre commande on l'implémente sur la carte de développement Memec Design V2MB1000. Les deux figures 4.21 et 4.22 montrent la visualisation de la commande sur un oscilloscope pour plusieurs valeurs de im.



a- im=0.1 b- im=0.05

Figure 4.21 : Visualisation de la commande s1 m=23



a- im=0.05 b- im=0.1

Figure 4.22 : Visualisation des commandes s1et s2 m=23

D'après les figures de visualisation on peut vérifier que l'on a bien 'm' commutations par quart-d'onde et que la forme est celle d'une tension Modulée en Largeur d'Impulsion. On voit bien le déphasage de $2\pi/3$ entre les deux signaux.

Conclusion

Conclusion

L'objectif de notre étude est *la commande de vitesse d'un moteur asynchrone triphasé* dans toute sa gamme de vitesses, de zéro à la vitesse nominale du moteur. L'objectif principal de ce mémoire est d'implémenter sur un circuit FPGA une commande MLI calculée, basée sur le principe des réseaux de neurone, avec élimination sélective d'harmoniques et asservissement du fondamental.

En effet, l'alimentation sinusoïdale variable en tension et en fréquence nécessaire à la commande de vitesse d'un moteur asynchrone est un cas idéal. Pour atteindre cet objectif on utilise généralement une commande MLI programmée avec élimination sélective d'harmonique et asservissement du fondamental de Patel et Hoft. Mais l'utilisation d'une telle commande en fonctionnement temps réel (on-line), comme l'exige la commande de vitesse, est impossible à cause d'un temps de calcul des angles très élevé. La technique de Patel et Hoft est de ce fait une technique 'off-line'.

Il fallait donc mettre au point un nouvel algorithme 'on-line' pour pallier cet inconvénient. C'est ce qui a été fait dans notre étude, où nous avons proposé une nouvelle méthode basée sur le principe des réseaux de neurone pour approximer les angles de commutation exacts. Il a été montré que la méthode proposée présentait un taux d'harmoniques négligeable par élimination sélective des harmoniques. Ce résultat est pratiquement le même que celui de l'algorithme de Patel et Hoft.

Dans le présent travail nous avons démontré la possibilité de générer une commande MLI calculée, avec élimination sélective d'harmonique et asservissement du fondamental, on-line en se basant sur le principe des réseaux de neurone et sur les capacités des circuits FPGA.

Nous avons aussi essayé une stratégie de commande permettant de varier la vitesse du moteur sur toute la gamme de variation tout en optimisant l'élimination des harmoniques en faisant varier le nombre d'angles par période de la commande MLI.

Notre nouvelle méthode 'on-line', utilisée en commun avec la stratégie de commande, permet donc de réaliser une commande de vitesse d'un moteur asynchrone triphasé en temps réel et avec un taux d'harmoniques minimal et constant dans toute la gamme de vitesses, des faibles vitesses à la vitesse nominale.

Annexe

A- Les angles exacts calculés par le programme MATLAB

a- pour m=23

L'angle	la valeur de Im									
	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
α1	4,978681083	4,957302192	4,93594599	4,914584695	4,89321811	4,87184607	4,85046838	4,82908486	4,80769532	4,78629957
α2	5,003520365	5,006986043	5,01047959	5,013973069	5,01746617	5,02095855	5,0244499	5,02793987	5,0314281	5,03491425
α3	9,978658384	9,957242798	9,93584362	9,914430254	9,89300236	9,87155958	9,85010156	9,82862792	9,80713829	9,78563229
α4	10,00649491	10,01292173	10,0193706	10,02581085	10,0322418	10,038663	10,0450737	10,0514735	10,0578617	10,0642378
α5	14,97845106	14,95683599	14,9352193	14,91357888	14,8919143	14,8702251	14,8485107	14,8267708	14,8050048	14,7832122
α6	15,00907298	15,01808634	15,0271043	15,03610471	15,0450868	15,0540499	15,0629934	15,0719165	15,0808186	15,0896989
α7	19,97816557	19,95628021	19,9343743	19,91243718	19,8904682	19,8684669	19,8464327	19,8243652	19,8022637	19,7801276
α8	20,01133986	20,02263586	20,0339183	20,04517611	20,0564087	20,0676154	20,0787955	20,0899482	20,1010728	20,1121687
α9	24,97787124	24,95570707	24,9335074	24,91127151	24,888999	24,8666892	24,8443417	24,8219558	24,799531	24,7770666
α10	25,01334541	25,02666305	25,0399527	25,05321353	25,066445	25,0796465	25,0928172	25,1059566	25,1190639	25,1321384
α11	29,97761589	29,9552091	29,9327574	29,91026758	29,8877391	29,8651714	29,842564	29,8199163	29,7972277	29,7744976
α12	30,01511997	30,0302254	30,045294	30,06033257	30,0753405	30,0903171	30,105262	30,1201745	30,1350539	30,1498996
α13	34,97743401	34,95485425	34,9322262	34,90956085	34,8868576	34,864116	34,8413354	34,8185155	34,7956555	34,772755
α14	35,01668257	35,03335997	35,0499977	35,06660675	35,0831866	35,0997368	35,1162568	35,1327462	35,1492044	35,1656309
α15	39,97735133	39,95469413	39,9319908	39,90925336	39,8864813	39,8636742	39,8408316	39,8179529	39,7950377	39,7720854
α16	40,01804567	40,03609178	40,0541009	40,07208495	40,0900437	40,1079768	40,1258839	40,1437645	40,1616183	40,1794448
α17	44,97738748	44,95476884	44,9321107	44,90942381	44,8867077	44,8639619	44,8411861	44,8183798	44,7955426	44,772674
α18	45,01921788	45,03843893	45,05763	45,07680175	45,0959541	45,1150867	45,1341994	45,153292	45,172364	45,1914153
α19	49,9775577	49,95511009	49,9326329	49,91013385	49,8876126	49,8650688	49,8425022	49,8199123	49,7972989	49,7746615
α20	50,02020582	50,04041582	50,060606	50,08078416	50,1009503	50,1211042	50,1412459	50,1613751	50,1814918	50,2015958
α21	54,97787399	54,95574331	54,9335947	54,91143232	54,8892559	54,8670652	54,8448601	54,8226402	54,8004053	54,7781551
α22	55,02101523	55,04203543	55,0630476	55,08405608	55,1050609	55,1260621	55,1470596	55,1680535	55,1890438	55,2100305
α23	59,97834698	59,95668912	59,9350263	59,91335837	59,8916852	59,8700068	59,8483228	59,8266332	59,8049379	59,7832367

b- pour m=19

L'angle	la valeur de Im									
	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.20
α1	5,71795975	5,69226361	5,66655703	5,640839748	5,61511149	5,58937199	5,56362098	5,53785817	5,5120833	5,48629606
α2	6,05435969	6,05928277	6,06420047	6,069112239	6,07401749	6,07891563	6,08380606	6,08868817	6,09356131	6,09842485
α3	11,7158264	11,6898505	11,6638487	11,63782062	11,6117656	11,5856832	11,5595727	11,5334337	11,5072654	11,4810674
α4	12,0987716	12,1076496	12,1165071	12,12534327	12,1341572	12,1429479	12,1517146	12,1604561	12,1691716	12,17786
α5	17,7112655	17,6847974	17,65829	17,63174254	17,6051543	17,5785245	17,5518525	17,5251374	17,4983784	17,4715748
α6	18,1365833	18,1488288	18,1610419	18,17322141	18,1853664	18,1974758	18,2095485	18,2215834	18,2335795	18,2455355
α7	23,7062842	23,6793123	23,6522924	23,62522346	23,5981048	23,5709355	23,5437147	23,5164415	23,489115	23,4617343
α8	24,1693264	24,1845072	24,1996487	24,21474993	24,2298099	24,2448274	24,2598016	24,2747313	24,2896153	24,3044526
α9	29,7020939	29,6747204	29,6472953	29,61981758	29,5922865	29,5647012	29,5370606	29,509364	29,4816102	29,4537983
α10	30,1977933	30,2155514	30,2332689	30,25094487	30,2685785	30,2861687	30,3037146	30,3212152	30,3386695	30,3560763
α11	35,6994894	35,6718873	35,6442351	35,61653181	35,5887767	35,5609688	35,5331072	35,5051911	35,4772195	35,4491913
α12	36,2224168	36,2424313	36,2624089	36,28234885	36,3022504	36,3221128	36,3419351	36,3617167	36,3814567	36,401154
α13	41,6990183	41,671407	41,6437515	41,61605081	41,5883043	41,5605113	41,5326708	41,504782	41,4768441	41,4488561
α14	42,2434428	42,2654109	42,2873499	42,30925934	42,3311387	42,3529873	42,3748047	42,3965902	42,4183433	42,4400633
α15	47,7010681	47,6736973	47,6462917	47,61885067	47,5913736	47,5638597	47,5363086	47,5087193	47,4810912	47,4534236
α16	48,2610204	48,2846472	48,308256	48,33184643	48,3554183	48,3789713	48,402505	48,4260193	48,4495136	48,4729877
α17	53,705916	53,6790548	53,6521708	53,62526374	53,5983331	53,5713784	53,5443992	53,5173951	53,4903654	53,4633097
α18	54,2752545	54,3002483	54,3252372	54,35022115	54,3752001	54,4001741	54,425143	54,4501068	54,4750654	54,5000189
α19	59,7137606	59,6876896	59,6616095	59,63552021	59,6094214	59,5833129	59,5571944	59,5310657	59,5049267	59,4787769

Annexe

B- Les poids et les seuils (weights et bias)

a- Pour m=23

IW	LW	B1	B2
2,119081229	-0,624273841	-0,96673742	0,493942625
	0,555771455		0,527681231
	-0,624284801		0,494233139
	0,598257223		0,508190976
	-0,62436211		0,49454875
	0,602887108		0,507706069
	-0,624458717		0,49479674
	0,616997798		0,507861428
	-0,624549277		0,494951632
	0,61982503		0,507803713
	-0,624620687		0,495012561
	0,621423158		0,507658241
	-0,62466636		0,494987922
	0,622438348		0,507446933
	-0,624682869		0,494889267
	0,623126309		0,507189818
	-0,624667946		0,494728873
	0,623611524		0,50690376
	-0,624619201		0,494518844
	0,623961942		0,506603144
	-0,624533132		0,494270889
	0,624216972		0,506300659
	-0,624402295		0,493994229

b- Pour m=19

IW	LW	B1	B2
-1,93639557	0,66680183	0,87904227	0,496335775
	-0,58300756		0,50461373
	0,66686146		0,496750045
	-0,64725229		0,505682996
	0,66695748		0,497090014
	-0,65798689		0,50591958
	0,66704783		0,49729371
	-0,66197327		0,505847958
	0,66711452		0,497363661
	-0,66397191		0,505612701
	0,66714965		0,497316044
	-0,66512513		0,505283286
	0,66714873		0,497168674
	-0,66584929		0,504901761
	0,66710736		0,496938387
	-0,66632781		0,504497499
	0,66701902		0,496640985
	-0,66665242		0,504093277
	0,66687316		0,496291852

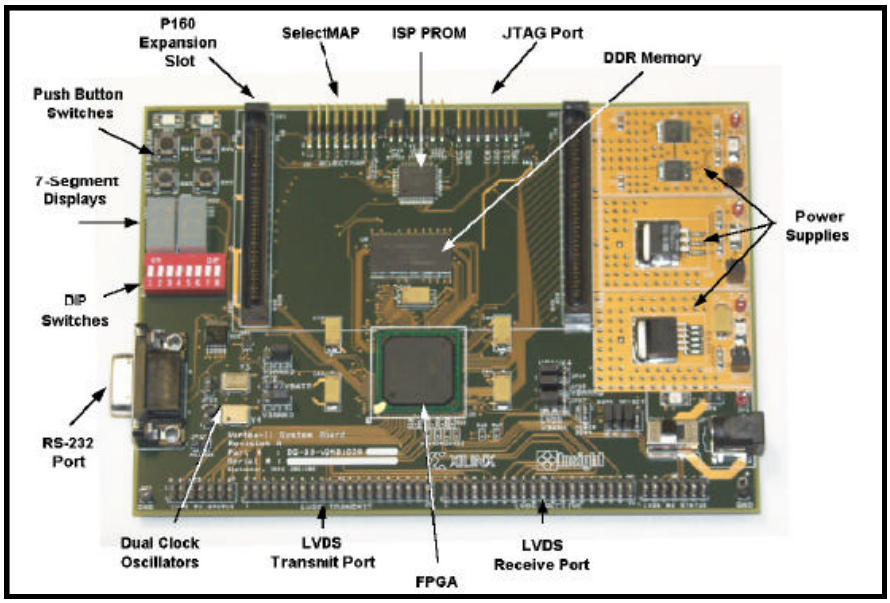
c- Pour m=15

IW	LW	B1	B2
0,63530328	-1,70353593	-0,00687112	-0,00789978
	1,62011397		0,02201248
	-1,70359701		-0,00511665
	1,68392255		0,02442334
	-1,70370426		-0,0033261
	1,69537361		0,02333988
	-1,70378208		-0,00261487
	1,69967038		0,0208627
	-1,70381935		-0,00286106
	1,70169957		0,01774437
	-1,70381365		-0,0039485
	1,7027521		0,01441134
	-1,7037536		-0,00577321
	1,70331315		0,01116179
	-1,70360786		-0,0082235

d- Pour m=7

IW	LW	B1	B2
-0,94864364	1,10442596	0,49285027	0,48368996
	-1,10135659		0,53339166
	1,10418357		0,48638628
	-1,10467793		0,52793608
	1,10404012		0,48674086
	-1,10467119		0,52109015
	1,10421175		0,48426608

C- Carte de développement « Memec Design Virtex-II »



Bibliographie

[1] H.S.Patel and R.G.Hoft,"Generalized techniques of harmonic elemination and voltage control in thyristor inverters: Part I- "Harmonic Elemination", IEEE Trans.Ind.App., vol.1A-9, pp. 310-317 No. 3,May/June 1973.
[2] M'hamed BOUNEKHLA, " Contribution à l'identification paramétrique de la machine asynchrone ", Thèse de Doctorat d'Etat ès-Sciences ENP 2004.
[3] Lotfi BAGHLI, "Modélisation et commande de la machine asynchrone", Notes de cours, IUFM de Lorraine, UHP 2005.
[4] E.Acha, V.G.Agelidis,O.Anaya-Lara,"Power Electronics Control in Electrical Systems" Newnes 2002.
[5] R.KABOUCHE,"Etude et élaboration d'une technique MLI à élimination d'harmoniques optimale", Mémoire de Magister en Electronique, Ecole Nationale Polytechnique, Alger, 2002.
[6] J.R. Espinoza,"Inveters" In: M.H.Rashid,"Power Electronic Handbook", Academic Press, USA 2001, pp 225-269.
[7] H.Foch, F.Forest et T.Meynard, "Onduleurs de tension. Structures. Principes. Applications" Techniques de l'ingénieur, vol. D31, N0 D3176, Novembre 1998.
[8] E.M.BERKOUK,"Contribution à la conduite des machines asynchrones monophasées et triphasées alimentées par des convertisseurs directs et indirects .Application gradateurs et onduleurs multiniveaux."Thèse de doctorat. C.N.A.M., Paris 1992.
[9] P.N.Enjeti, P.D.Ziogas and J.F.Lindsay,"Programmed PWM Techniques to Eliminate Harmonics: A Critical Evaluation", IEEE Trans.Ind.App., vol.26, NO 2, pp. 302-316, MARS/AVRIL 1990.
[10] J.P. Louis, et C. Bergmann, 'Command numérique. Systèmes triphasés: régime permanent', Techniques de l'ingénieur, vol. D3III, N°D3642, novembre 1996.
[11] N.A.Azli and L.Y.Chan, "Development of Equations Through Trajectories Linearization for an HEPWM Inverter ", First international PECon 2006.
[12] J.A.TAUFIQ,Prof.B.MELLITT And C.J.GOODMAN, 'Novel algorithme for Generating near optimal PWM waveforms for AC traction drives', IEE Procceedinges, Vol.133,PT.B,No2,March 1986,pp 85-94.
[13] Eric Monmasson, Marcian N. Cirstea,"FPGA Design Methodology for Industrial Control Systems- A Review", *IEEE Trans. Ind. Electron.*, vol. 54, no. 4,Aout 2007.
[14] Juan J- Rodrequez-Andina, "Features, Design Tools, and Application Domains of FPGAs", IEEE Trans. Ind. Electron., vol.54, No.4, Aout 2007.
[15] J P Deschamps, G JAntoine Bioul,"Synthesis of Arithmetic Circuits FPGA, ASIC, And Embedded Systems" JOHN WILEY & SONS, 2006.
[16] Khider M, " commande de vitesse en temps réel d'un moteur asynchrone triphasé " mémoire de magister, Ecole Nationale Polytechnique, Alger, 2003.
[17] Abdelmoumene Toudeft, Méthodes connexionnistes pour la commande des systèmes non linéaires : application à la régulation des rivières. Thèse de doctorat de l'université Paris 6. décembre1998.
[18] A.GOURDIN et M.BOUMAHRAT, 'Méthodes Numériques Appliquées', OPU, seconde édition , 1991.
[19] Claude Touzet, Les réseaux de neurones artificiels «introduction au connexionisme». 'Laboratoire d'Etudes et Recherche en Informatique', ISBN 2 - 906 899 - 78X. Juillet 1992.
[20] T. Kohonen, Self organization and associative memory. Springer Series in information sciences, Vol.8, Springer-Verlag, 1984; and second edition, 1987.
[21] W.S. McCulloch et W. PITTS, A logical calculus for the ideas immanent in nervous activity. Dans Bulletin of Mathematical Biophysics. Vol. 5, pp. 115-137, 1943.
[22] Isabelle Rivals, Modélisation et commande de processus par réseaux de neurones : application au pilotage d'un véhicule autonome. Thèse de doctorat de l'université paris 6. Janvier 1995.
[23] Juan J- Rodrequez-Andina, "Features, Design Tools, and Application Domains of FPGAs", IEEE Trans. Ind. Electron., vol.54, No.4, Aout 2007.
[24] PONG P. CHU. "RTL Hardware Design Using VHDL" USA:John Wiley & Sons 2006, 669p.

[25] Dragomir Milojevic, "Implémentation des filtres non-linéaires de rang sur des architectures universelles et reconfigurables", Thèse de Doctorat en Sciences Appliquées, Université Libre de Bruxelles, 2004.
[26] Claude Guex, "introduction au VHDL", Ecole d'ingénieurs de Canton de Vaud Juin 98
[27] Volnei A Pedroni, "Circuit Design With VHDL" MIT Press 2004.
[28] S.Berto, A.Paccagnella, M.Ceschia,"Potentials and Pitfalls of FPGA Application in Inverter Drives - a Case Study"in Proc. IEEE ICIT 2003, pp.500-505.
[29] SCHULTZ W, ”Drive techniques for high side N channel MOSFET”, Revue PCIM, Juin 1987, pp. 34-40.
[30] Blasckhe,F., "The principle of field oriented as rotationg-field machines,” Siemens Review XXXIX, n0 5, pp. 217-220, 1972.