REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE RECHERCHE SCIENTIFIQUE

DEPARTEMENT D'ELECTRONIQUE

Projet de Fin d'Etudes

pour l'obtention du diplôme

d'Ingénieur d'Etat en Electronique

Thème

Implémentation d'une technique MLI en temps réel sur un circuit FPGA

Proposé et dirigé par :

Mr . C.LARBES(ENP)

Mr. N. CHIKHI (CDTA)

Etudié par :

Mr GUELLAL amar

Année universitaire 2006/2007

بسم الله الرحمن الرحيم

ملخص:

إن تقنية تغير عرض الدفع (MLI) "المبرمجة" التي وضعها كل من باتل و هوفت تمثل حلا ممتازا 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 تحسب مباشرة زوايا التبديل مع اشتغال المحرك و بدقة متساوية مع دقة تقنية باتل و هوفت,هذه الدقة تسمح بالحذف الكلي للمركبات التوافقية الأولى المختارة من أجل الحذف, و ضبط المركبة الأساسية.

كلمات مفاتيح: بطاقة "FPGA " , برنامج "VHDL ", تقنية "MLI " , التحاكي, باتل, هوفت

Résumé :

La technique de modulation de largeur d'impulsion (MLI) calculée 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 projet est d'implémenter sur un circuit FPGA un* nouvel algorithme MLI temps réel *qui présente une précision de calcul pratiquement égale à celle de la technique de Patel et Hoft permettant ainsi une* élimination totale *des premiers harmoniques sélectionnés et un asservissement du fondamental.

Mots clés : Circuit FPGA, langage VHDL, MLI, simulation, on-line, Patel, Hoft.

Abstract :

The calculated pulse width modulation (PWM) technique of Harmonic Elimination and Voltage Control is an attractive alternative for speed control of an induction motor. However, its application has been limited by the fact that the switching angles cannot be calculated online in real-time applications.Our project aim is the implementation of new PWM algorithm based on real-time calculation of the switching angles. 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 circuit, VHDL language, PWM, simulation, on-line, Patel, Hoft.

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 tiens aussi à exprimer mes plus vifs remerciements à Monsieur N.CHIKHI du Laboratoire de Microélectronique du Centre de Développement des Techniques Avancées (CDTA) pour ses éclaircissement.

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

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.

Dédicace

Je dédie ce travail à :

Mes parents,

Mes frères et mes soeurs,

Toute ma famille,

Ainsi qu'à toute mes amis.

Amar

Table de matière

Liste des figures

Liste des tableaux :

Introduction

Introduction

Dans le domaine de l'industrie, La vitesse variable est assurée généralement par le moteur à courant continu. Mais celui-ci présente plusieurs inconvénients : coût élevé, présence d'un collecteur, entretien fréquent.

Pour les éviter, 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, encombrement faible.

Ce moteur est alimenté à partir d'un onduleur de tension variable en tension et en fréquence, avec le rapport V/f constant à Modulation de Largeur d'Impulsion (MLI).

Mais ce fonctionnement est accompagné de *pulsations de couple* et d'*échauffement* du moteur dus à la présence d'*harmoniques* dans le signal de sortie de l'onduleur de tension MLI. Ces pulsations de couple deviennent gênantes aux faibles vitesses surtout. Pour pallier à ces inconvénients, on doit éliminer correctement les harmoniques. Autrement dit, on doit calculer les angles de commutation avec une plus grande précision.

Le but de notre projet est d'implémenter une commande MLI avec asservissement du fondamental et élimination harmonique sélective, telle que définie par la technique de Patel et Hoft, les angles de commutation sont approximées à l'aide d'un *nouvel algorithme 'on-line'*.

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, on s'intéresse à la technique MLI '*programmée*', avec *élimination harmonique sélective* et *asservissement du fondamental*, de Patel et Hoft [3]. On calcule les angles exacts de commutation par une méthode numérique (Newton-Raphson) [5]. Puis on approxime ces angles exacts à l'aide d'un *algorithme 'on-line'* .

Dans le troisième chapitre, on aborde les circuits FPGA et on s'intéresse à leur architecture et leurs caractéristiques. On fait une description de la carte de développement Virtex-II de MEMEC qu'on utilise pour l'implémentation, et on termine le chapitre par une explication du langage de description VHDL.

Au dernier chapitre on aborde notre but qui est l'implémentation de la commande MLI avec approximation des angles de commutation sur la carte VERTEX-II, on présente les programmes VHDL, on les simule à l'aide du ModelSim puis on les visualise sur un oscilloscope.

Chapitre 1

Introduction générale

1- LE MOTEUR ASYNCHRONE

1.1- Introduction

Le moteur électrique est une machine électrique qui transforme de l'énergie électrique en énergie mécanique.

Il existe différents types de moteurs électriques :

- Moteur à courant continu : il est utilisé en vitesse variable dans une large gamme de puissances.
- Moteur synchrone : il est utilisé dans la gamme grandes puissances.
- Moteur asynchrone : il est utilisé en moyennes et faibles puissances.

Le choix du type de moteur est en fonction du domaine d'application.

1.2- Définition et caractéristiques [3]

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

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

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)

1.3 Inversion du sens de rotation du moteur [3]

Le sens de rotation du moteur asynchrone triphasé s'inverse en inversant le sens de rotation du flux tournant statorique. Il suffit pour cela de permuter les entrées de deux phases. L'inversion du sens du flux tournant pendant la rotation du moteur entraîne d'abord le freinage puis l'arrêt et enfin la rotation en *sens inverse* du moteur.

2- Les onduleurs

Les onduleurs de tension sont des convertisseurs statiques qui servent principalement 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'ensemble des techniques et méthodes agissant directement sur "les éléments actifs" de l'onduleur et permettant l'amélioration de la forme d'onde de sortie sont appelées communément les ***stratégies de commande***..

Le schéma de principe d'un onduleur de tension monophasé est donné par la figure 1

2.1- Définition

Un onduleur est un convertisseur statique assurant la conversion continu- alternatif par une séquence adéquate de commande des semi-conducteurs.

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.

Le schéma de principe d'un onduleur de tension monophasé est donné par la figure 1.1

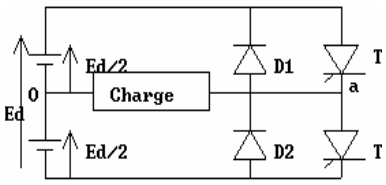


Figure 1.1 Schéma de l'onduleur demi-pont

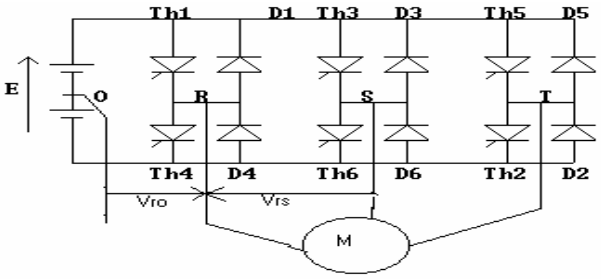


Figure 1.2 : Schéma d'un onduleur triphasé

2.2- La commande des onduleurs

2.2.1- Introduction

Pour commander un onduleur, la solution la plus largement employée consiste à utiliser les intersections d'une onde de référence ou modulante, généralement sinusoïdale, avec une onde de modulation ou porteuse, généralement triangulaire, d'où son appellation de « modulation sinus-triangle ou MLI engendrée ». On peut également utiliser une bascule à hystérésis commandée par la différence entre une onde de référence (ou son intégrale) et l'intégrale de tension en créneaux à la sortie de l'onduleur ; c'est la « modulation en delta » (ou en sigma-delta). On peut également générer la séquence des signaux de commande des interruptions de façon à suivre au mieux le vecteur défini par les composants de Clarke du système de tensions qu'on veut produire : c'est la « modulation vectorielle ». Dans certaines applications, on calcule au préalable, sur la base d'un critère d'optimisation, les instants de commande, c'est la « modulation calculée ou MLI calculée ».

2.2.2- TECHNIQUES MLI

On distingue principalement deux types de techniques MLI : la MLI 'engendrée' (triangulo-sinusoidale) et la MLI 'programmée' (calculée).

a- Technique MLI engendrée (triangulo-sinusoidale) [2]

La modulation dite triangulo-sinusoidale, compare la tension de sortie recherchée, dite tension de référence ou *tension modulatrice* Uo, à un signal triangulaire symétrique, appelé *porteuse* Up. La fréquence de la porteuse Up est *multiple* de celle de l'onde modulatrice Uo (onde fondamentale).

La tension Uo est *inférieure ou égale* à la tension Up. Un comparateur change d'état à chaque intersection des deux signaux superposés Up et Uo(figures 1.3 et 1.4).

Ce changement d'état se traduit par la génération des impulsions de commutation aux deux redresseurs Th1 et Th2 de l'onduleur demi-pont monophasé. La tension de sortie VRO ne peut avoir instantanément que deux valeurs :+E/2 ou -E/2 [2]. La tension VRO prend la valeur +E/2

lorsque Th1 conduit (Th2 est alors bloqué) et prend la valeur -E/2 lorsque Th2 conduit (Th1 est alors bloqué). Le redresseur Th1 conduit lorsque la modulatrice U_0 est supérieure à la porteuse U_p. Le redresseur Th2 conduit lorsque la modulatrice U_0 est inférieure à la porteuse.

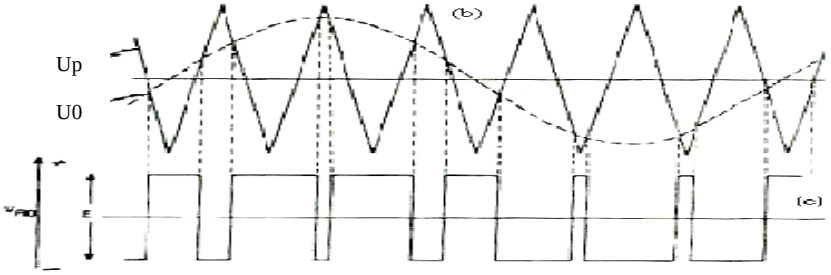


Figure 1.3 Principe de la modulation triangulo-sinusoïdale

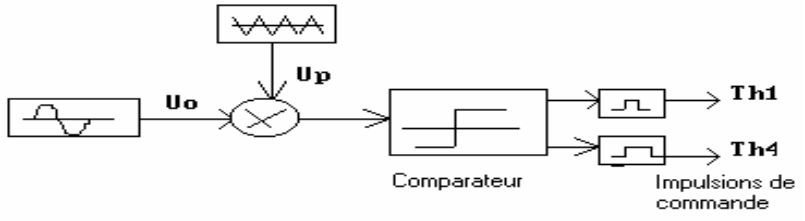


Figure 1.4 Schéma synoptique de la loi de modulation triangulo-sinusoidale

.

Ce principe de modulation s'applique aussi pour un montage triphasé(figure 1.5)

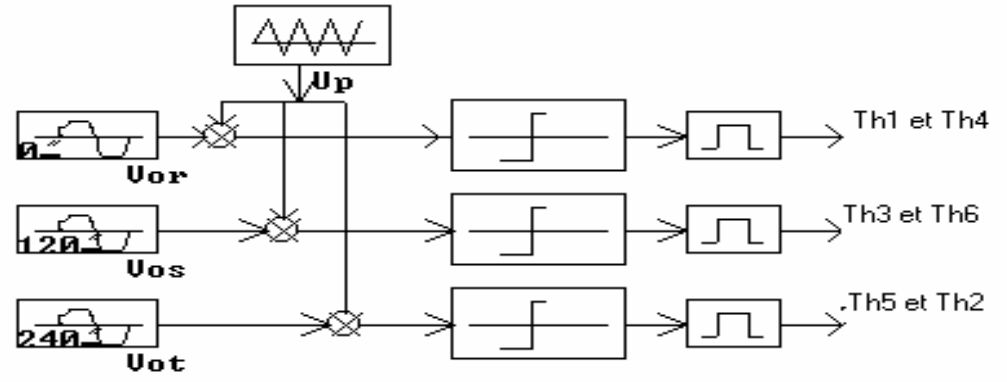


Figure 1.5 Schéma synoptique de la loi de modulation triangulo-sinusoidale pour un montage triphasé.

b- Technique MLI programmée: [4]

Dans ce cas, les impulsions qui commandent la commutation des redresseurs de l'onduleur MLI sont générées par un circuit numérique.

La MLI programmée optimise une fonction particulière, comme par exemple la minimisation des pertes, la réduction des fluctuations du couple ou encore l'élimination sélective d'harmonique.

Cette dernière technique est considérée comme la plus efficace pour l'obtention de résultats performants.

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

3- Réglage de la vitesse d'un moteur asynchrone

Pour faire varier la vitesse du moteur asynchrone, il faut faire varier la fréquence de ses tensions d'alimentation : il faut donc l'alimenter par un onduleur. En même temps que la fréquence, il faut faire varier la valeur efficace des tensions d'alimentation.

3.1- Utilisation d'un onduleur à U/F constant

On se propose de fournir un couple *maximal et constant* des faibles vitesses à la vitesse nominale.

Considérons la relation suivante: **[1]**

$$T\max = \frac{3pV^2}{4\pi f(Rs + \sqrt{Rs^2 + [2\pi f(Ls + Lre)]^2}} \qquad (1.5)$$

avec Rs et Ls respectivement 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 la fréquence de la tension d'alimentation.

Sachant que, dans l'équation (1.5), les paramètres Rs, Ls, p et Lre sont constants, alors le couple Tmax est fonction seulement des variations de V et f.

En technique MLI, la tension d'alimentation et la fréquence doivent varier simultanément. Pour atteindre le but proposé ci-dessus, le rapport V/f doit varier en fonction de la fréquence, de la facon suivante :

Pour des fréquences en dessous de la fréquence nominale Fr, *le rapport V/f doit être maintenu constant* pour que *le flux totalisé reste approximativement constant*. Dans ce cas on obtient un couple de sortie maximum Tm constant (voir figure 1.6).

$Rs << 2\pi f(Ls+Lre)$ (1.6)

La tension d'entrée du moteur V devrait être à la valeur maximale Vr quand f est égale à la fréquence nominale Fr.

Pour des valeurs faibles de f, l'équation (1.6) n'est plus valable car Rs prend des valeurs relativement importantes. On doit donc diminuer *V moins vite que f*.

Pour des fréquences supérieures à la fréquence nominale Fr, la tension V est maintenue constante à la valeur nominale Vr. Dans ce cas, le couple est proportionnel à $1/f^2$ et la puissance est proportionnelle à 1/f (figure 1.6).

En fait dans notre étude, on s'intéresse surtout à la région située en dessous de la fréquence nominale fr.

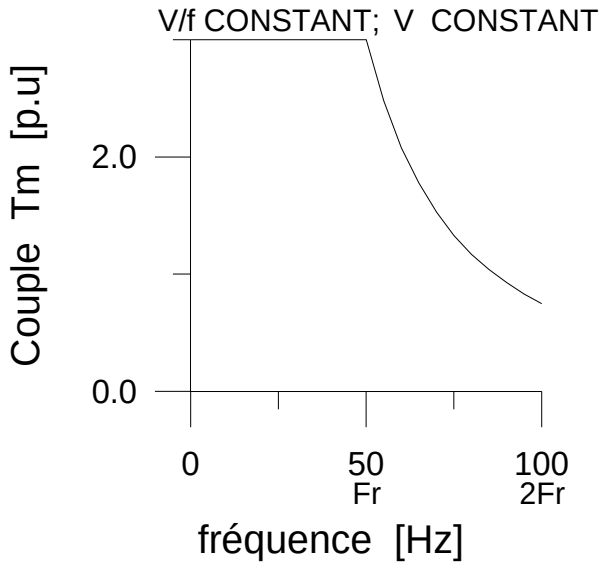


Figure 1.6 Cette courbe décrite le couple moteur Tm en fonction de la fréquence (avec le rapport V/f constant puis avec V constant et égal à Vr).

Chapitre 2

TECHNIQUE MLI CALCULEE

1-DESCRIPTION de la TECHNIQUE de PATEL et HOFT

1.1- DESCRIPTION [3]

Considérons un onduleur demi-pont monophasé figure 1.1.

Soit la tension de sortie à deux états de l'onduleur demi-pont de la Figure 2.1. 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:

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

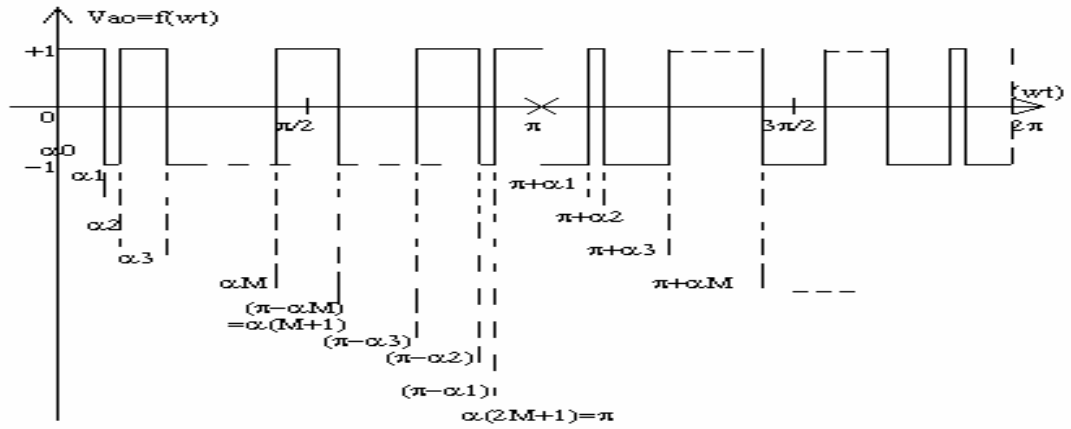


Figure 2.1 graphe d'une tension MLI calculée

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 i.e

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

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 coefficiens a_n et b_n sont alors donnés par :

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

Remplacons 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 (2.8) 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 dans la section 2.

1.2- Remarque sur les angles de commutation:

Considérons la figure 2.1. On a :

$$\alpha = \alpha_0 = 0,\ \text{Th1 conduit},\ \text{Th2 bloqué}$$
$$\alpha = \alpha_1 \quad ,\ \text{Th2 conduit},\ \text{Th1 bloqué}$$
$$\alpha = \alpha_2 \quad ,\ \text{Th1 conduit},\ \text{Th2 bloqué}$$
$$\alpha = \alpha_3 \quad ,\ \text{Th2 conduit},\ \text{Th1 bloqué}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\alpha = \alpha_M \quad ,\ \text{Th2 conduit},\ \text{Th1 bloqué}$$

Donc lorsque Th1 conduit (et Th2 bloqué) on a f(wt)=+1, et lorsque Th2 conduit (et Th1 bloqué) on a f(wt)= -1; autrement dit les thyristors Th1 et Th2 conduisent *alternativement* pour connecter le point a aux phases positive et négative, Figure 2.1.

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

2.1- DESCRIPTION

La relation (2.8) 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 résoud 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 obrenu par la méthode du gradient. Mais, celle-ci étant lente (convergence linéaire)[6], on utilisera plutôt l'algorithme 'on-line' de Taufik, Mellitt et Goodman [5] 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 [10]. 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[5]. 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:

$$\begin{aligned}
a_1 &= \frac{4}{n}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(\alpha_k)\right] = -im \\
a_5 &= \frac{4}{5\pi}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(5\alpha_k)\right] = 0 \\
a_7 &= \frac{4}{7\pi}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(7\alpha_k)\right] = 0 \\
a_{11} &= \frac{4}{11\pi}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(11\alpha_k)\right] = 0 \\
a_{13} &= \frac{4}{13\pi}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(13\alpha_k)\right] = 0 \\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
a_n &= \frac{4}{n\pi}\left[1 + 2\sum_{k=1}^{m}(-1)^k \cos(n\alpha_k)\right] = 0
\end{aligned} \qquad (2.9)$$

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 1.16. 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.

2.2- Estimation initiale de la solution: [5]

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.

2.3- Résolution du système non linéaire par la Méthode de Newton-Raphson. [6]

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 f_i 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 :

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

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

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

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:

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

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

Définissons la matrice des dérivées premières :

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

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

D'où :

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

Définissons le *vecteur erreur* :

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

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:

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

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[6].

.

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

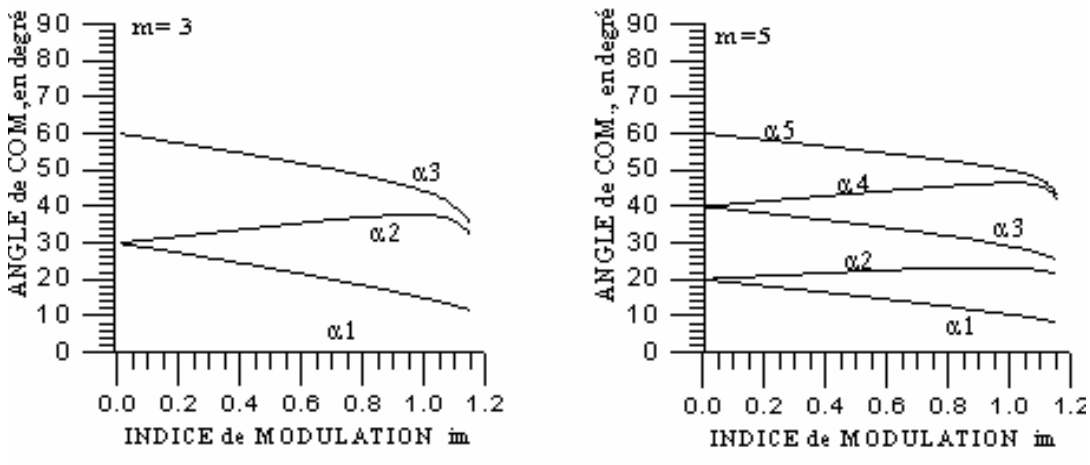


Figure 2.2 Courbes des angles de commutation exacts pour m égal à 3 et 5.

3- Le NOUVEL ALGORITHME 'ON-LINE'

3.1- Approximation des angles exacts [7]

Il faut entendre par '*exact*' une grandeur calculée en utilisant les valeurs exactes des angles de commutation et par '*approximé*' une grandeur calculée en utilisant les valeurs approximées des angles de commutation. Les valeurs exactes sont calculées par un système d'équations non linéaires. Les valeurs approximées sont calculées par le nouvel algorithme.

La figure 2.2 montre des courbes représentant les valeurs exactes des angles de commutation α_k en fonction de l'indice de modulation im, pour un indice m égal à 3 et 5. Pour un indice de modulation im égal à zéro, l'angle α_s, appelée angle de *séparation*, est donné par la formule approchée:

$$\alpha_s = \frac{2 \times 60°}{(m+1)} \qquad \text{m impair} \qquad (2.15)$$

Pour m fixé, les courbes de rang k impair présentent une pente négative et sont 'presque' parallèles dans la majeure partie de l'intervalle de variation [0,1.15] de l'indice de modulation im, à l'exception des valeurs extrêmes de cet intervalle. De même les courbes de rang k pair présentent cette fois une pente positive et sont 'presque' parallèles dans les mêmes conditions de variation de l'indice de modulation im que précédemment. Cette caractéristique nous amènent à approximer les courbes correspondantes à k impair différemment de celles correspondantes à k pair. De plus la forme de ces courbes laisse penser que les fonctions qui

approximent les courbes *exactes* devraient être la combinaison d'une fonction linéaire et d'une autre fonction non linéaire de l'indice im et dont l'expression reste à déterminer.
L'idée, donc, consiste à approximer la valeur exacte de l'angle de commutation en approximant chaque courbe exacte en *deux étapes.*

Dans une première étape, on approxime la partie entière de la valeur exacte de l'angle de commutation par une fonction linéaire de l'indice im.

Dans une seconde étape, on cherche à approximer soit la partie fractionnaire de la valeur exacte, soit la différence entre la valeur exacte et la valeur approximative (trouvée dans la première étape).

La mise en oeuvre de ces deux étapes d'approximation nécessite la conception et la mise au point d'un programme d'approximation basé sur la méthode des M*oindres Carrés.* Ce programme calcule les coefficients A_0, A_1,...., Ap (p<n) de la *fonction d'approximation* suivante:

$$\Phi(x) = A_0 . \phi_0(x) + \ldots + A_p . \phi_p(x) \qquad (2.16)$$

permettant l'approximation de la fonction f(x) connue *empiriquement* en (n+1) points et qui prend les valeurs b_0, b_1, ..., b_n aux abscisses a_0, a_1, ..., a_n.

Les fonctions élémentaires d'approximation $\phi_0, \phi_1, \ldots, \phi_p$ sont *choisies à l'avance*.

3.1.1- Cas k impair

Soit m fixé et soit α_k la valeur exacte de l'angle de commutation:

a- Première étape d'approximation de α_k:

On cherche une fonction d'approximation linéaire de la forme:

$$\Phi(im) = A_0 + A_1' * im \qquad (2.18)$$

qui approxime la partie entière des valeurs exactes de chaque courbe $\alpha_k(im)$ avec im variant de 0 à 1.15 et k impair. On a:

$$A_0 = \alpha_k(0) = \alpha_s = (k+1)\frac{60°}{(m+1)} \qquad (2.19)$$

On suppose : $$A_1' = A_1 \times \frac{60°}{(m+1)} \qquad (2.20)$$

On remplace dans l'équation (4.3). On trouve la nouvelle fonction d'approximation:

$$\Phi(im) = \frac{60°}{(m+1)}(k+1) + A_1 \times (\frac{60°}{(m+1)} im) \qquad (2.21)$$

Notons $[\alpha_k]$, la partie entière de α_k. Pour calculer A_1, il faut mettre l'équation (2.21) sous la forme:

$$\Phi'(im) = [\alpha_k] - \frac{60°}{(m+1)}(k+1) = A_1 \times (\frac{60°}{(m+1)} im)$$

On cherche donc à approximer la différence suivante:

$$[\alpha_k] - \frac{60°}{(m+1)}(k+1) \qquad (2.22)$$

Par la fonction d'approximation suivante:

$$\Phi'(im) = A_1 \times (\frac{60°}{(m+1)} \times im) \qquad (2.23)$$

La fonction d'approximation élementaire est donc:

$$\phi_0(im) = \frac{60°}{(m+1)} im \qquad (2.24)$$

Le problème maintenant est de déterminer le coefficient A_1 pour chacune les courbes représentant les valeurs exactes des angles de commutation en fonction de m et k impair.

Pour chaque valeur de m et de k, on fait varier l'indice de modulation im de 0.1 à 1.1 par pas de 0.1. Pour chaque valeur de l'indice im, on calcule la différence (2.22).

On répète le calcul en faisant varier m de 3 à 11 par valeurs impaires et pour chaque valeur de m on fait varier k de 1 à m par valeurs impaires.

Le coefficient A1 est pratiquement *constant* et *égal à moins un (-1)* lorsque k varie de 1 à m, par valeurs impaires, quelque soit la valeur de m. L'équation (2.21) devient:

$$\Phi(\text{im}) = \frac{60°}{(\text{m}+1)}(k+1) + (-1) \times \frac{60°}{(\text{m}+1)} \times \text{im} \qquad (2.25)$$

b-Seconde étape d'approximation de α_k:

Après l'approximation de la partie entière de l'angle 'exact' de commutation, dans la première étape d'approximation, on va effectuer maintenant l'approximation de la *différence* d(im) entre la valeur exacte et la valeur approximée, donnée par l'équation (2.25), de la partie entière de l'angle de commutation. La différence d(im) est définie par:

$$\text{d(im)} = \alpha_k(\text{im}) - \Phi(\text{im}) = \alpha_k(\text{im}) - \left[\frac{60°}{(m+1)}(k+1) + (-1) \times \frac{60°}{(m+1)} \times im\right] \qquad (2.26)$$

Pour étudier la forme de d(im), on va calculer d(im) pour m égal à 9, par exemple, et pour k variant de 1 à 9 par valeurs impaires en faisant varier im de 0.1 à 1.1 par pas de 0.1. La figure 2.3 représente la fonction d(im).

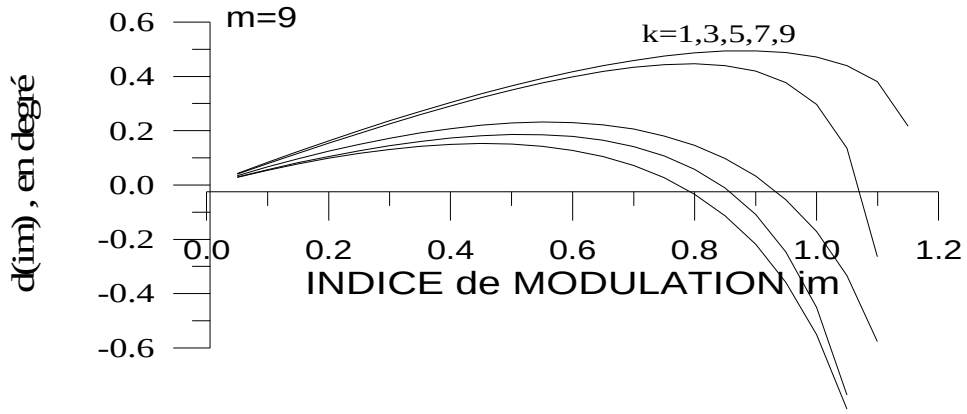


Figure 2.3 : Courbes représentant d(im) en fonction de im, avec m égal à 9 et k variant de 1 à m.

Cependant en observant ces courbes on constate qu'elles ont chacune une forme *linéaire* en début d'intervalle, passent par un maximum puis *décroissent rapidement* en fin d'intervalle.

Le problème maintenant est de rechercher comment effectuer l'approximation des courbes de la figure 2.3 à l'aide de fonctions simples et avec une plus grande précision ?

Pour répondre à cette question, considérons les fonctions suivantes:

$f(x)= a_1.x$ $a_1>0$ $x \varepsilon [0,1.15]$

$g(x)= a_2.x^p$ $a_2>0$ $p>0$ $x \varepsilon [0,1.15]$

Considérons maintenant la différence d(x) définie par:

$$d(x)= f(x)-g(x) = (a_1.x- a_2.x^p) \quad (2.27)$$

Calculons d(x) pour x variant dans l'intervalle [0,1.15] et traçons les courbes correspondantes avec les paramètres suivants:

a_1=0.5, 1, 1.5 a_2=1 p=4

Comparons les courbes des figures 2.3 et 2.4. Elles présentent une grande *similitude* entre elles. Il est tout à fait naturel d'utiliser une fonction de la forme d(x), donnée par l'équation (2.27), pour faire l'approximation, avec une *bonne précision,* de la fonction d(im) donnée par l'équation (2.26).

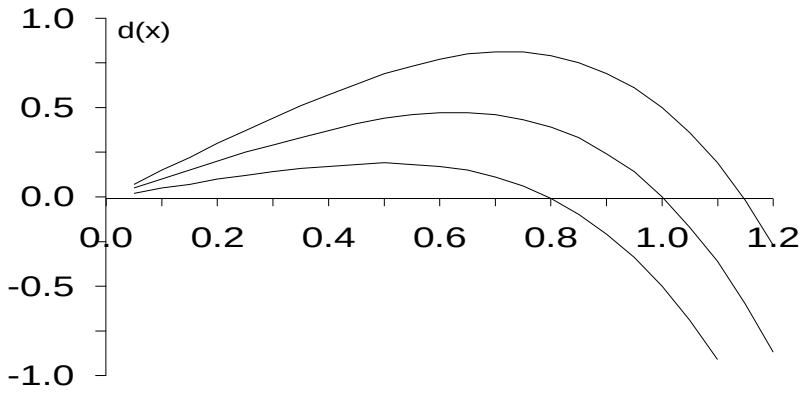


Figure 2.4 Courbes représentant la fonction d(x) pour: a2=1, p=4 et a1=0.5, 1, 1.5.

Donc la fonction d'approximation de la fonction d(im) sera une fonction de la forme :

$$\Phi(\text{im}) = A_0.\text{im} - A_1.(im)^p \qquad \text{avec } A_0 \text{ , } A_1 \text{ , p réels} \qquad (2.28)$$

Les fonctions élémentaires d'approximation sont donc:

$$\phi_0(im) = im \quad ; \qquad \phi_1(im) = -(im)^p \qquad (2.29)$$

Le problème maintenant est de déterminer les trois coefficients A_0, A_1 et p. On fixe p et pour chaque valeur de m et k on fait varier l'indice de modulation im de 0.1 à 1.1 par pas de 0.1. Pour chaque valeur de l'indice im, on calcule la différence (2.26). On a donc 11 données numériques par courbe à approximer. Le coefficient p est choisit de façon que l'erreur moyenne entre d(im) et $\Phi(im)$ soit minimale. Ces coefficients vont nous permettre de construire la fonction d'approximation (2.28) qui va approximer l'ensemble des 11 données numériques citées ci-dessus.

On répète le calcul en faisant varier m de 3 à 11 par valeurs impaires et pour chaque valeur de m on fait varier k de 1 à m par valeurs impaires.

Quelques résultats sont donnés, à titre d'exemple, dans le tableau 2.1.

On a donc:

$$\text{d(im)} = \alpha_k(\text{im}) - \left[\frac{60°}{(m+1)}\left[(k+1) - im\right]\right]$$

$$\Phi(\text{im}) = A_0.\text{im} - A_1.(\text{im})^p \qquad p > 0$$

$$\text{d(im)} \cong \Phi(\text{im})$$

Des trois équations précédentes on déduit l'équation générale d'approximation des valeurs exactes des angles de commutation $\alpha_k(\text{im})$ pour *k impair*:

$$\alpha_k(\text{im}) \cong \frac{60°}{(m+1)}(k+1) + \left[A_0 - \frac{60°}{(m+1)} \right] \times \text{im} \; - \; A_1 \times (\text{im})^p \qquad (2.30)$$

Avec m=3,5,..,21; k=1,3,..,m et im ∈ [0,1.15].

Tableau 2.1 : Exemple de coefficients A0,A1,p de l'équation (2.30) pour k impair et m égal à 3, 5 et 7.

m	k	p	A0	A1	emax	Emoy
3	1	4	1.1659	1.3675	0.0540	0.0290
3	3	8	1.2320	2.2860	0.1500	0.0960
5	1	4	1.0514	0.7118	0.027	0.015
5	3	6	0.4619	1.4311	0.076	0.045
5	5	9	0.9653	1.1071	0.091	0.054
7	1	4	0.9042	0.4530	0.018	0.009
7	3	5	0.5112	1.0020	0.045	0.023
7	5	7	0.3514	1.1006	0.063	0.040
7	7	10	0.7667	0.5854	0.054	0.033

3.1.2- Cas k pair

De l'équation (2.30) on déduit la forme générale de l'équation générale d'approximation des angles de commutation 'exacts' pour *k pair* :

$$\alpha_k(im) = \frac{60°}{(m+1)} \times k + A_0 \times im - A_1 \times im^p \qquad (2.31)$$

Avec m=3,5,..,21 ; k=2,4,..,m-1 ; im ∈ [0,1.15].

De l'équation (2.31) on déduit la différence:

$$\alpha_k(\text{im}) - \frac{60°}{(m+1)}(k+1) \cong A_0 \times \text{im} \; - \; A_1 \times (\text{im})^p \qquad (2.32)$$

Les fonctions élémentaires d'approximation sont donc:

$$\phi_0(im) = im \qquad (2.33)$$

$$\phi_1(im) = -(im)^p \qquad (2.34)$$

Le problème maintenant est de déterminer les trois coefficients A_0, A_1 et p en fonction de m et k pair. On fixe p et pour chaque valeur de m et k on fait varier l'indice de modulation im de 0.1 à 1.1 par pas de 0.1. Pour chaque valeur de l'indice im, on calcule la différence (2.32). On a donc 11 données numériques par courbe à approximer. Le coefficient p est choisit de façon que *l'erreur moyenne* entre d(im) et $\Phi(im)$ soit *minimale*. Ces coefficients vont nous permettre de construire la fonction d'approximation (2.27) qui va approximer l'ensemble des 11 données numériques citées ci-dessus.

On répète le calcul en faisant varier m de 3 à 11 par valeurs impaires et pour chaque valeur de m on fait varier k de 1 à m par valeurs paires.

Les résultats du Tableau 2.2 montrent que l'erreur moyenne, notée emoy, est de l'ordre du centième de degré (1/100°). Donc, la technique d'approximation utilisée donne une plus grande précision de calcul par rapport aux algorithmes MLI 'on-line' connus aussi bien pour les angles pairs que impairs.

Tableau2.2 Exemple de coefficients A0,A1,p de l'équation (4.17) pour k pair et m égal à 3, 5, 7 .

m	k	p	A0	A1	emax	emoy
3	2	10	9.0872	1.5766	0.093	0.038
5	2	6	4.4472	1.3392	0.084	0.026
5	4	11	7.1389	0.7694	0.062	0.027
7	2	6	2.5970	0.8099	0.028	0.017
7	4	7	4.3641	1.0538	0.074	0.028
7	6	12	5.6821	0.4038	0.037	0.018

3.2- Précision de l'Algorithme:

Les figures 2.5 et 2.6 représentent *l'erreur en valeur absolue*, en degré, entre la valeur *exacte* de l'angle de commutation et sa valeur *approximée* par l'algorithme pour m égal à 5 et 15, dans tout l'intervalle de variation [0, 1.15] de l'indice im. Les figures montrent que l'erreur moyenne est de l'ordre de trois centièmes (3/100) de degré pour m égal à 5 et qu'elle diminue lorsque m augmente et atteint un centième (1/100) de degré pour m égal à 15. Il faut signaler que l'erreur moyenne est calculée sur tout l'intervalle [0;1.15] de variation de l'indice im et qu'elle est beaucoup plus faible pour de faibles valeurs de im.

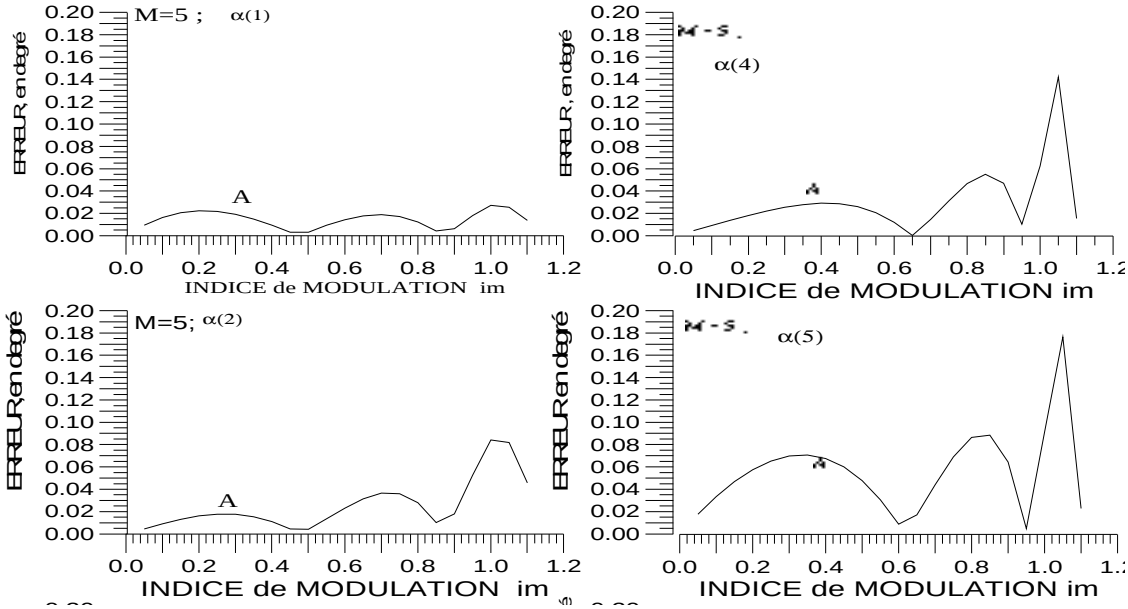


Figure 2.5 Graphe représentant l'erreur angulaire entre la valeur exacte et la valeur approximée pour m égal à 5.

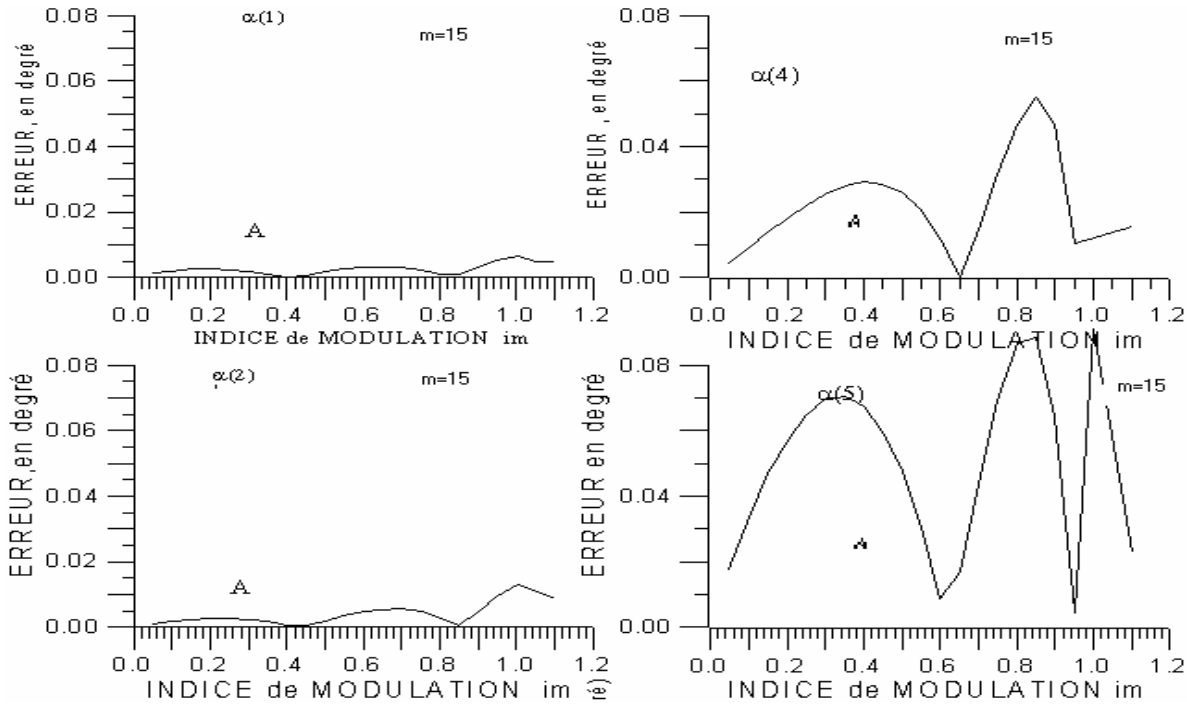


Figure 2.6 Graphe représentant l'erreur angulaire entre la valeur exacte et la valeur approximée des angles de commutation pour m égal à 15.

Chapitre 3

Les circuits FPGA et le langage de description VHDL

1- Les circuits FPGA

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, professionnel 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.

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.

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.

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(figuire3.1).

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.

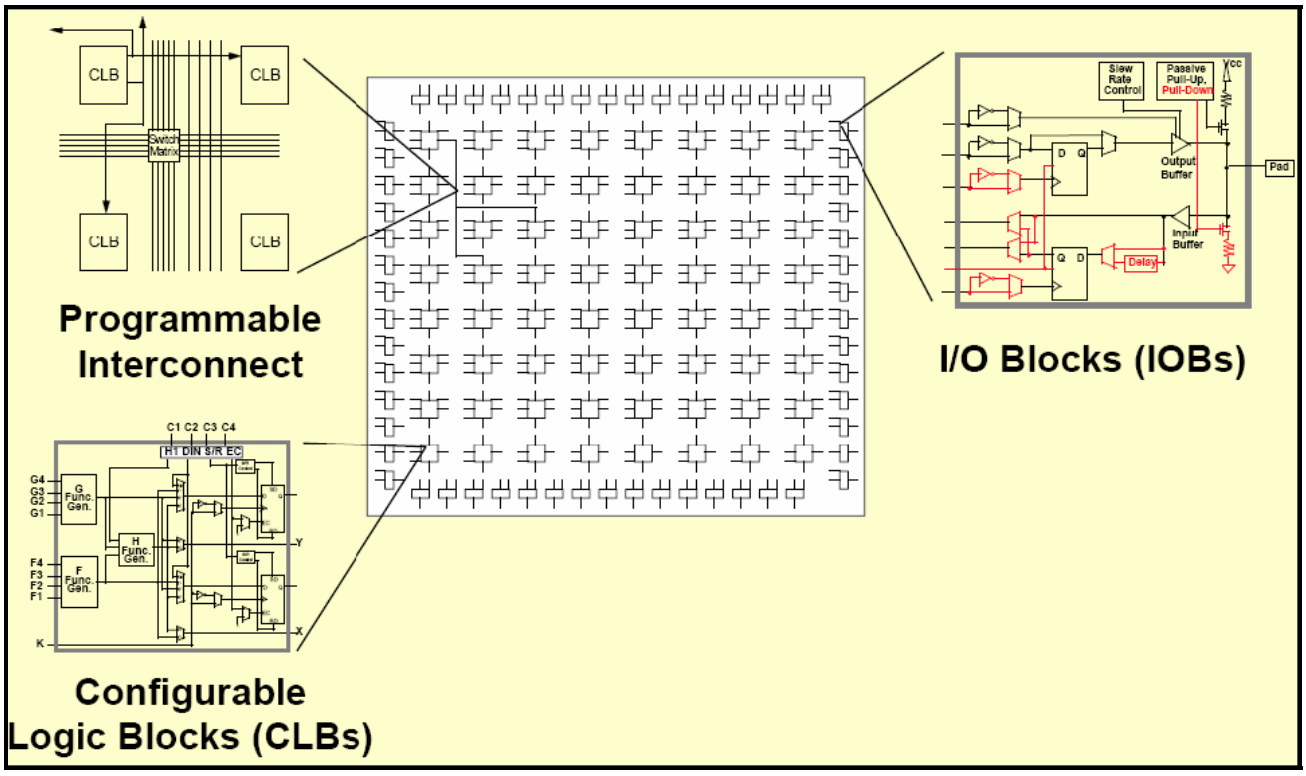


Figure 3.1 Architecture interne du FPGA

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.

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.

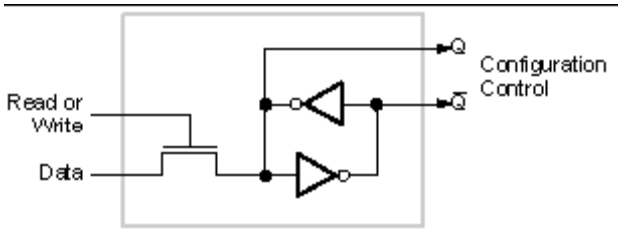


Figure 3.2 Structure d'un cellule SRAM

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.

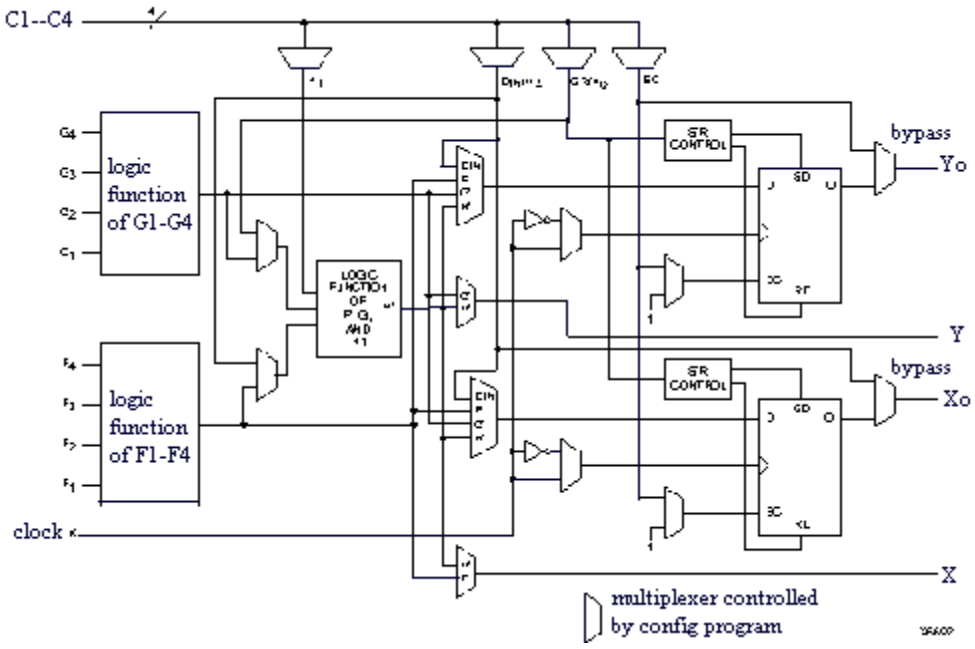


Figure 3.3 Cellules logiques (CLB)

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.

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).

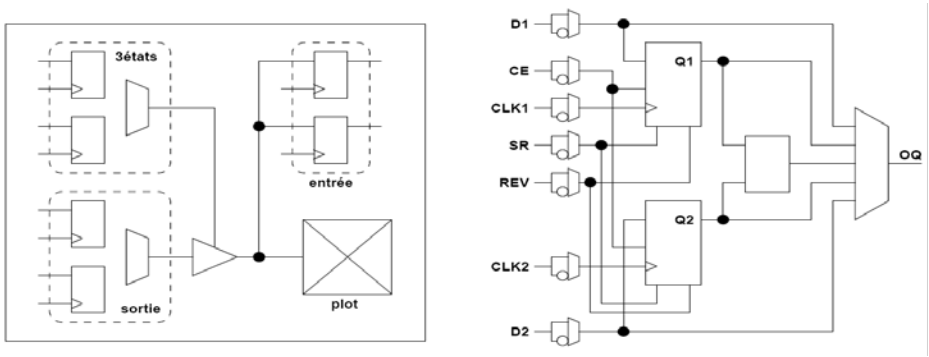


Figure 3.4: Input Output Block (IOB)

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

1.5- Les configurations en entrée et en sortie

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.

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.

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.

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.

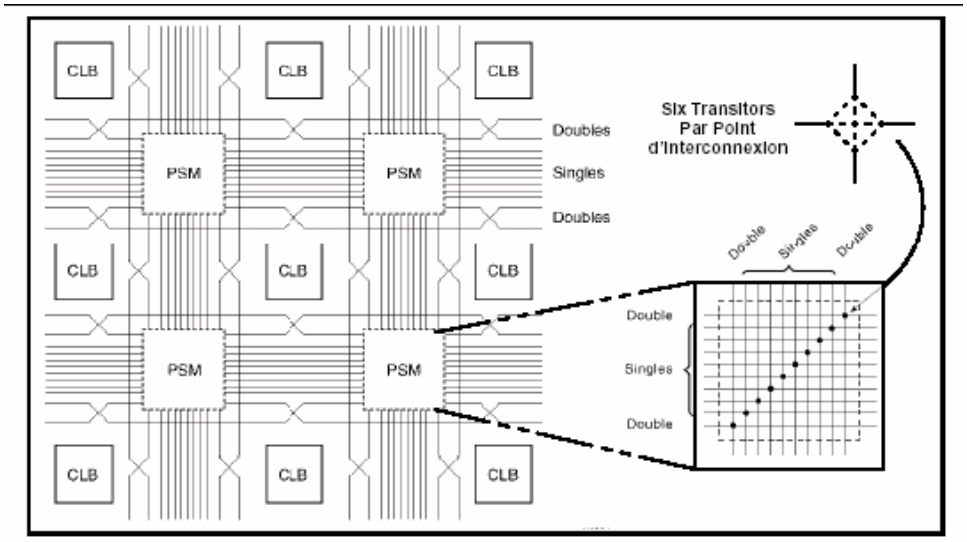


Figure 3.5 : Connexions à usage général et matrice de commutation

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.

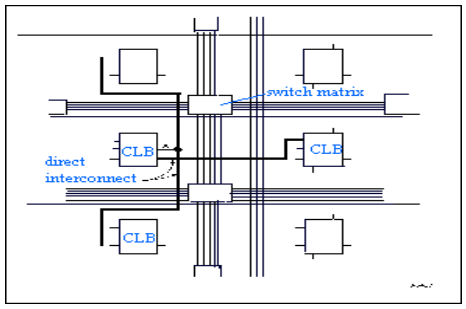


Figure 3.6 : Les interconnexions directes

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 lignes permettent d'éviter la multiplicité des points d'interconnexion.

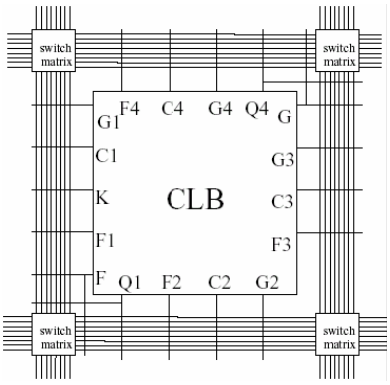


Figure 3.7 : Les longues lignes

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.

2- La carte de développement Virtex-II V2MB1000 de Memec Design [8]

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.

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 p8résents 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.

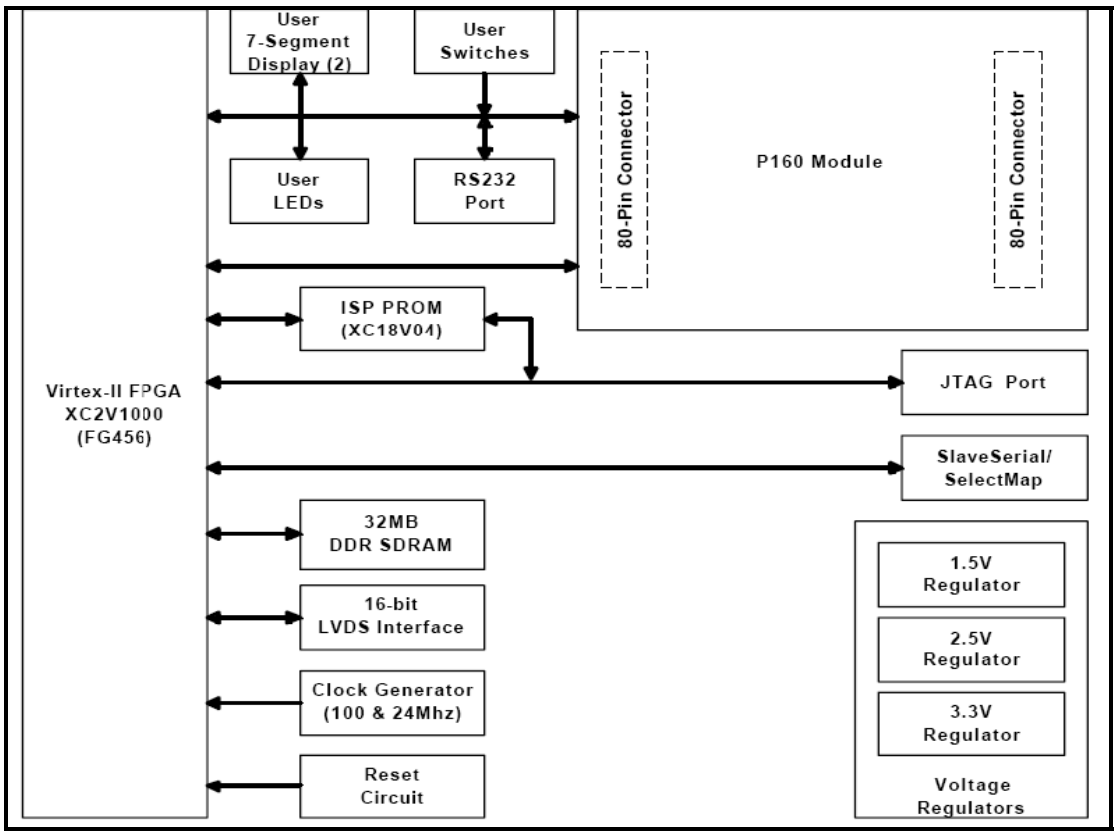


Figure 3.10 : Diagramme de la carte de développement

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.

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.

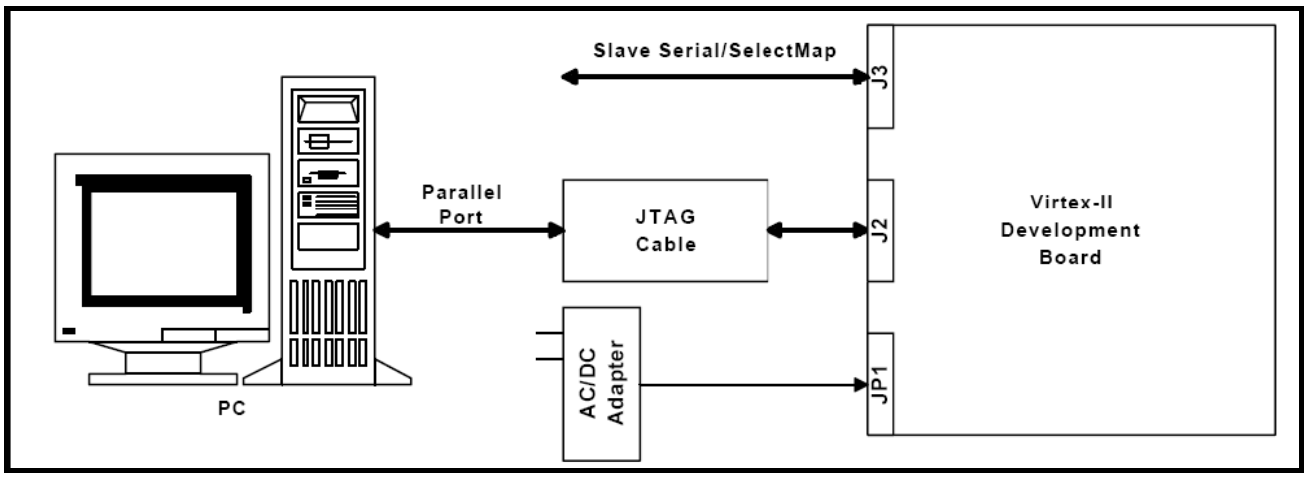


Figure3.9 : Chargement du programme sur la carte

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.

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.

3- Langage de description VHDL

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 du 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.

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.

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.

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 esais 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.

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.

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.

3.8- 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.

3.8.1- 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;
```

3.8.2- 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.

3.9- 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.

3.9.1- 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.

3.9.2- 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 standard 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).

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 3.10 donne les différentes étapes nécessaires au développement d'un projet sur circuit FPGA.

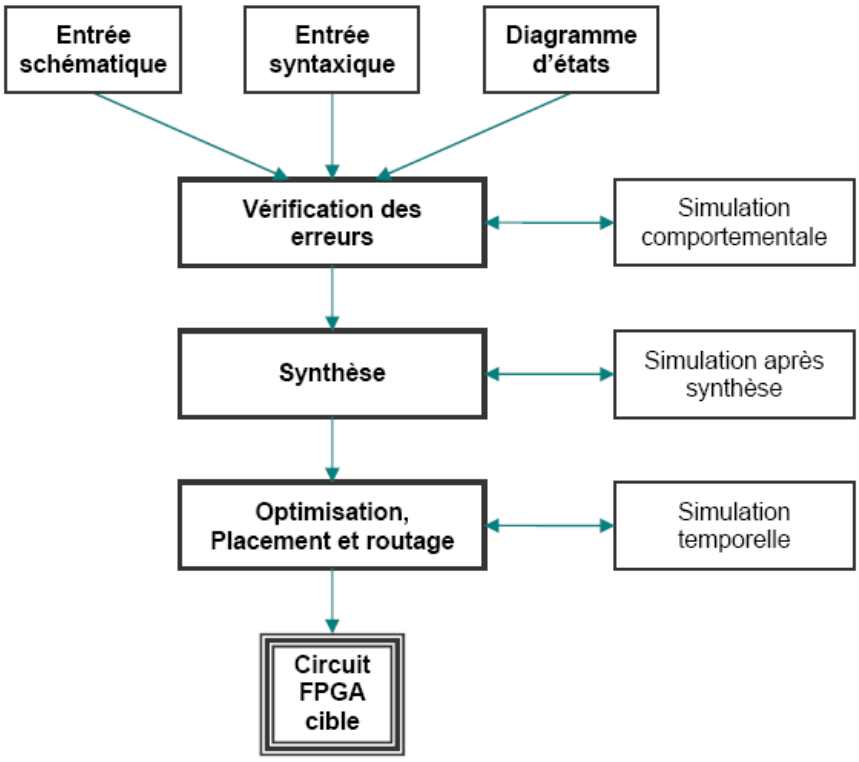


Figure 3.10 : Organisation fonctionnelle de développement d'un projet sur circuit FPGA

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 3.11 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.

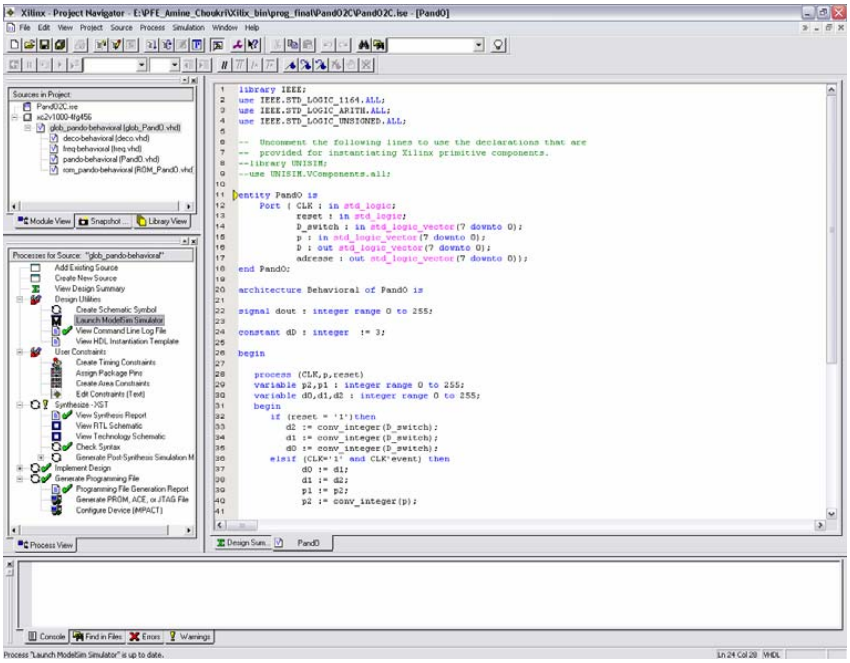


Figure 3.11 : Vue d'ensemble du logiciel « Xilinx Project Navigator »

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.

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 (figure3.12).

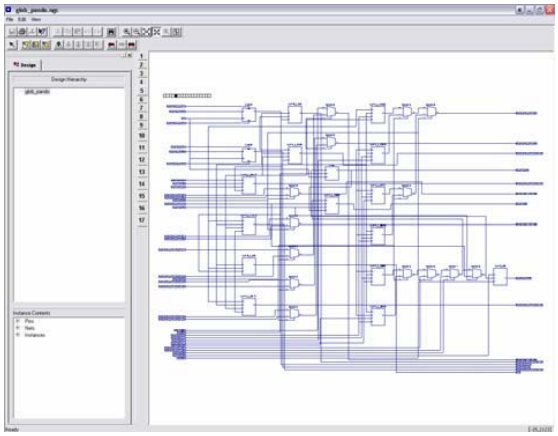


Figure 3.12 : Aperçu de l'outil « View RTL Schematic »

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 3.13 montre un aperçu de l'outil d'assignation des broches d'entrées/sorties.

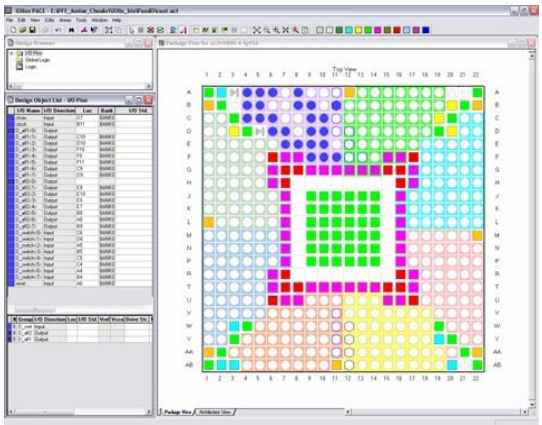


Figure 3.13 : Aperçu de l'outil d'affectation des broches d'entrées/sorties

3.4- Simulation

Le simulateur utilisé est le « ModelSim Simulator » (figure 3.14). 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.

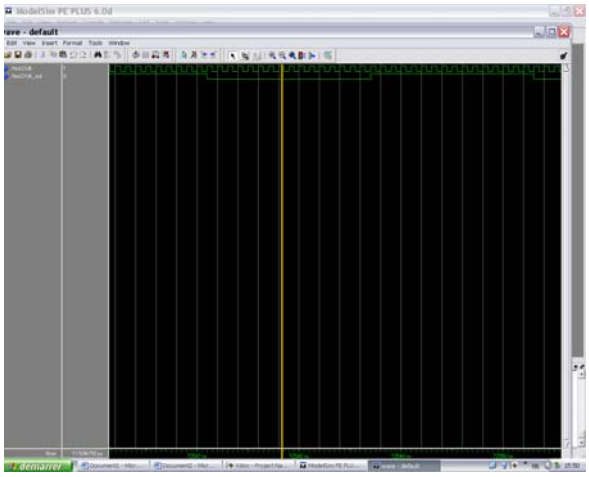


Figure 3.14 : Présentation du simulateur « ModelSim Simulator »

3.5- 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 3 .15 donne un aperçu de l'outil de placement et routage « FPGA Editor » qui permet de visualiser et d'éditer le circuit routé.

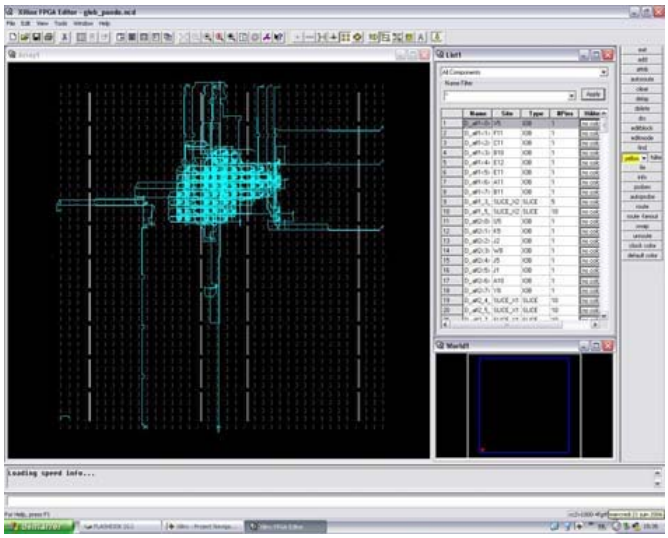


Figure 3.15 : Aperçu de l'outil « FPGA Editor »

3.6- 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.

Chapitre 4

Implémentation de la commande ‘ON-LINE’ sur une FPGA

1- Introduction

Le but de notre projet est d’implémenter la commande on-line qui a été décrite dans le chapitre II sur un circuit FPGA. Pour cela on utilise le langage VHDL. Dans une première étape on programme une fonction qui calcul les angles de commutation, et dans l’étape suivante on réalise un programme qui à partir des angles de commutation donne les six signaux de commande ; et dans la dernière étape on propose un programme pour tester notre commande sur la carte de développement VERTEX-II. Et avant le passage d’une étape à la suivante on simule le programme en utilisant le model SIM de XLINX.

2- Le calcul des angles de commutation

2.1- Description

Dans le langage VHDL il n’existe pas une bibliothèque standard qui définit et compile les nombres réels, pour cela on travaille avec les nombres entiers qui sont prédéfinis dans le langage.

Pour faire les différentes opérations arithmétiques et logiques sur ces nombre on fait l’appelle des bibliothèques suivantes « *std_logic_arith Package, std_logic_1164 Package* ».

Pour calculer les angles de commutation on utilise les deux équations (2.30) pour les k impaires et (2.31) pour les k paires.

2.2- Le programme

2.2.1- Les angles de commutaion

Puisque on travaille avec les nombres entiers on calcul d'abord β_k définit par :

$$\beta_k = 10^5 (m+1)\alpha_k \qquad (3.1)$$

On fait la multiplication par (m+1) car dans la bibliothèque « *std_logic_arith Package* » il n'existe pas une fonction qui fait la division des nombres entiers, et on fait la multiplication par 10^5 pour augmenter la précision de calcul. Et on remplace im qui varie de 0 à 1 par pas de 0.01 par $\text{d} = 100\,\text{im}$. Donc les deux équations (2.30) , (2.31) deviennent

Pour k impaire

$$\beta_k(\text{im}) \cong 60000\left[100 \times (k+1) - d\right] + (m+1)\left(B_0 \times d - B1 \times (\text{im})^{\text{p}}\right) \qquad (3.2)$$

Pour k paire

$$\beta_k(\text{im}) \cong 6 \times 10^6 k + (m+1)\left(B_0 \times d - B1 \times 100 \times (\text{im})^{\text{p}}\right) \qquad (3.3)$$

Avec B0=1000A0 et B1 = 1000A1.

Pour calculer une division entière on utilise la soustraction et le comptage, et pour calculer une puissance de p on utilise la multiplication (p-1) fois.

La fonction a comme entrées les variables suivants :

d :est la commande d = 100 im

m : est le nombre des angles

k : est l'indice de l'angle

B0 : est déterminés à partir des tableaux (2.1) et (2.2)

B1: est déterminés à partir des tableaux (2.1) et (2.2)

p : est déterminés à partir des tableaux (2.1) et (2.2)

s: pour k paire s = '0', si non s '1'

et comme sortie c qui représente l'angle α_k en degrés multiplie par 100

a- Organigramme:

L'organigramme de l'algorithme précèdent est décrit par la figure 4.1.

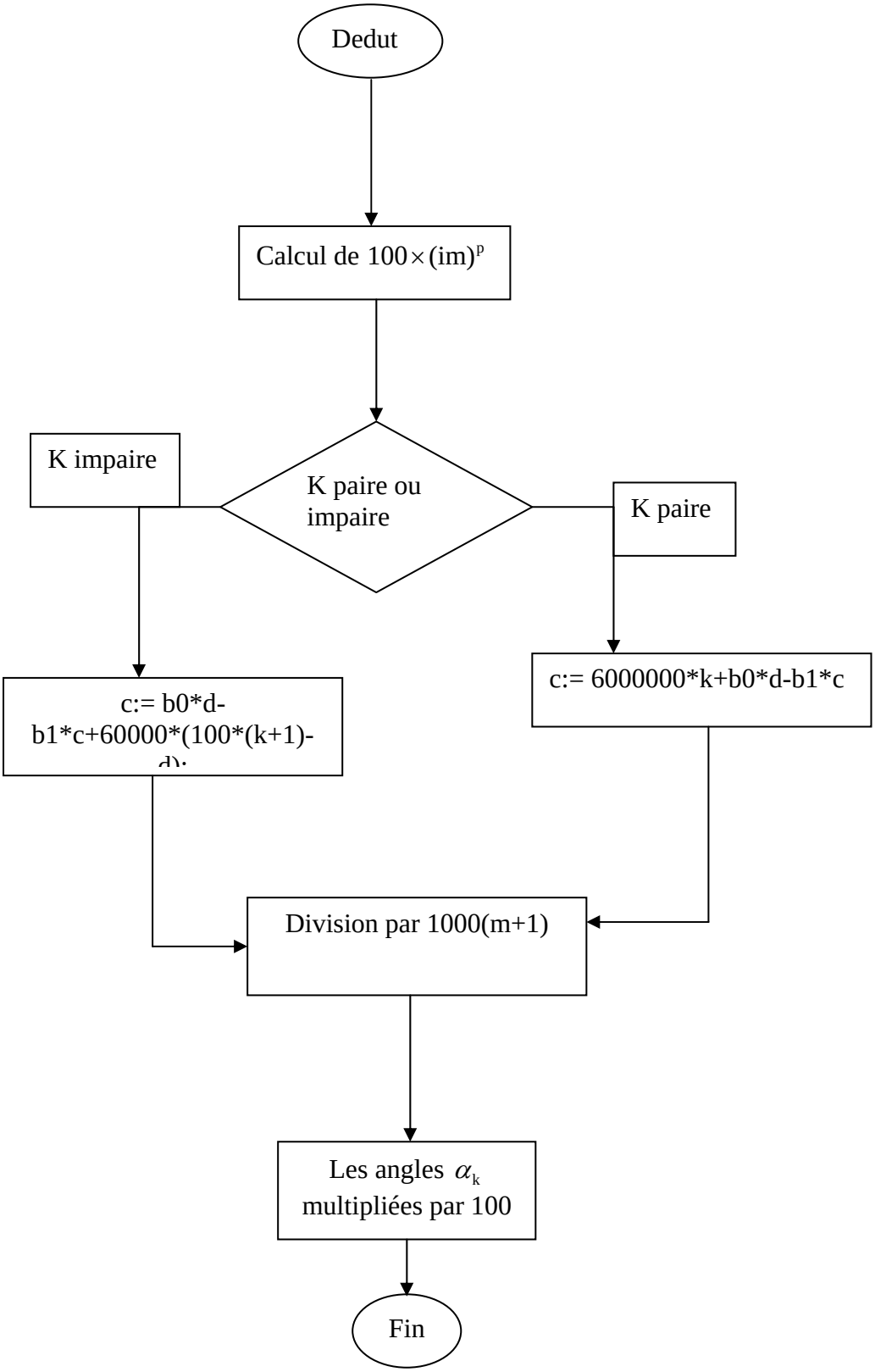


Figure 4.1 :L’organigramme da la fonction qui calcul les angles de commutation

b- Programme

On donne dans le tableau suivant une explication du programme qui calcule les angles de commutation :

La fonction qui calcule les angles de commutation

```
function cal3  ( b0: in integer; b1 : in integer;  p : in integer ; d : in integer
                  ;k : in integer;m : in integer;s : in bit)  return integer is
--declaration des variables
variable c : integer:=1;  variable i : integer:=2;  variable g : integer:=0;
variable e : integer:=0; variable f : integer:=0;
begin
c:=d;
-- calcul d'une puissance de p
for h in 1 to 14  loop
 if i<=p then
    i:=i+1; c:=c*d;
    for  j in 1 to 200 loop
          if c>100 then  e:=e+1; c:=c-100;end if ;
    end loop;
    if c >50 then c:=e+1;else c:=e;end if;e:=0;end if;
end loop;
--pour k paire
if s='0' then
c:= 6000000*k+b0*d-b1*c;
--pour k impaire
else if s='1' then
c:= b0*d-b1*c+60000*(100*(k+1)-d);
end if;end if;
--la division par(m+1)
g:=1000*(m+1);
for l in 1 to 9000 loop
if c>=g then c:=c-g; f:=f+1; end if;
end loop;
c:=f;
return c;
end cal3;
```

2.2.2- Les instants temporels équivalents

Pour programmer les signaux de commande, il faut transformer ces angles en degrés alpha (i) à ses instants temporels équivalents T (i).

On a :

$$alpha(i)= 2*180*f*T(i)$$

$$T(i)=alpha(i)/(2*180*f) \quad (3.4)$$

Pour avoir le rapport V/f constant, on prend :

V/Vn =im (3.5)

et f/fn =im (3.6)

D’où V/f=(Vn*im)/(fn*im)=Vn/fn=constante (3.7)

Des équations (3.5) à (3.7), on déduit:

f=V*(fn/Vn)=(im**Vn)*(fn/Vn)=im*fn

L’équation (3.4) devient:

T(i)=alpha(i)/(2*180*im*fn) i=1,(4m+2) (3.8)

avec T(i) en seconde, fn égale à 50 Hz et im l’indice de modulation.

Donc les instants de commutation sur un quart de période sont donnés par l’équation (3.8) à T seconde près, T étant la période temporelle.

Finalement on trouve :

T(i)=(B(i)*100)/(360*d*fn) (3.9)

Ou’ :

B(i) sont données par les deux équations (3.2) et (3.3).

d = 100 im

a- Organigramme:

L’organigramme de l’algorithme précèdent est décrit par la figure 4.2.

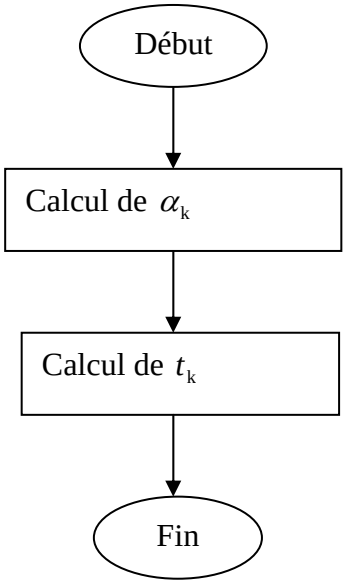


Figure 4.2 : organigramme de la fonction qui calcul les instants temporels

2.3- Les résultats de simulation

On fait un programme pour tester les deux fonctions qui calculent les angles de commutation et les instants temporels équivalents ; on prend dix valeur de im, pour $i_m \le 0.5$ on prend m=7, pour $0.5 < i_m \le 0.8$ on prend m=5 et pour $0.8 < i_m \le 1$, puis on le simule par le MODEL SIM, les résultats de simulation sont montrés dans le tableau 4.1 et sur les figures 4.3 et 4.4.

im	*Les angles de commutation trouvées par le programme*						
0.1	14.26	15.03	29.25	30.05	44.25	45.07	59.25
0.2	13.52	15.06	28.51	30.10	43.50	45.14	58.51
0.3	12.78	15.09	27.76	30.16	42.76	45.21	57.77
0.4	12.04	15.12	27.02	30.21	42.01	45.28	57.03
0.5	11.30	15.16	26.27	30.27	41.27	45.35	56.29
0.6	14.08	20.43	34.03	40.71	54.09	0	0
0.7	13.09	20.49	33.02	40.83	53.10	0	0
0.8	12.09	20.53	31.99	40.64	52.10	0	0
0.9	16.53	31.90	46.53	0	0	0	0
1.0	14.94	31.87	44.73	0	0	0	0

Tableau 4.1 : les angles de commutation calculées par le programme

im	*Les angles de commutation trouvées par le programme*						
0.1	14.34	15.26	29.30	30.43	44.28	45.56	59.32
0.2	13.68	15.51	28.60	30.87	43.57	46.13	58.65
0.3	13.01	15.77	27.90	31.30	42.85	46.70	57.98
0.4	12.35	16.03	27.19	31.74	42.13	47.27	57.30
0.5	11.67	16.28	26.47	32.17	41.41	47.84	56.63
0.6	14.53	22.60	34.21	44.28	54.56	0	0
0.7	13.56	22.95	33.15	44.98	53.63	0	0
0.8	12.54	23.20	31.99	45.64	52.62	0	0
0.9	16.65	37.62	48.60	0	0	0	0
1.0	14.79	37.51	43.94	0	0	0	0

Tableau 4.2 : les angles de commutation calculées par l'ordinateur

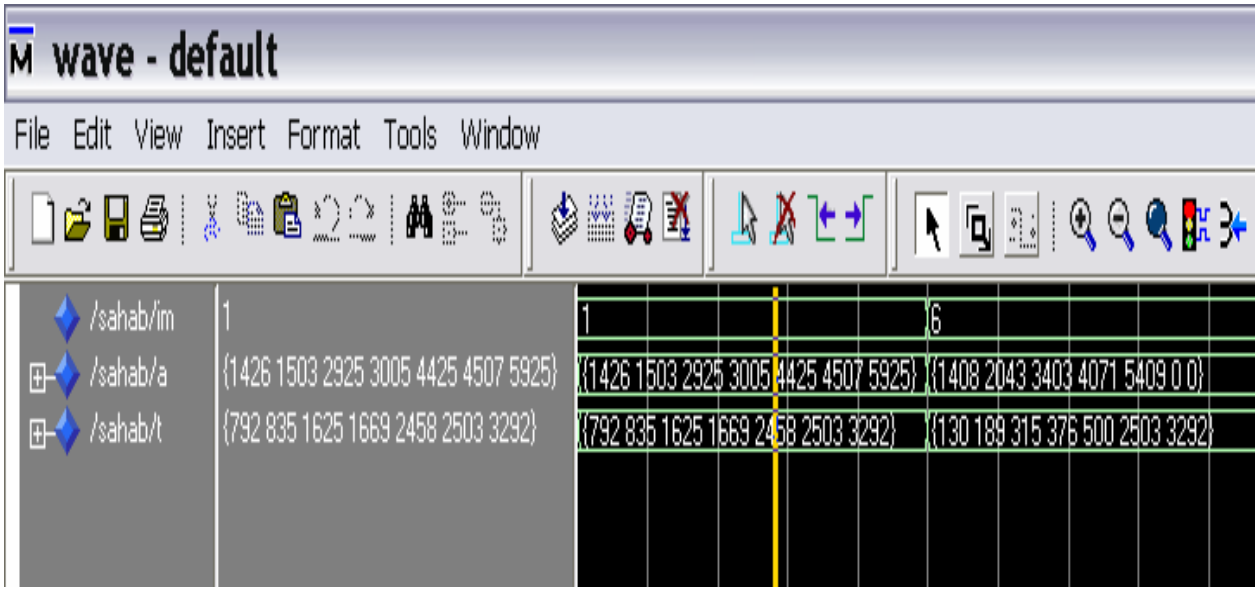


Figure 4.3 : simulation pour im =0.1 et im=0.6

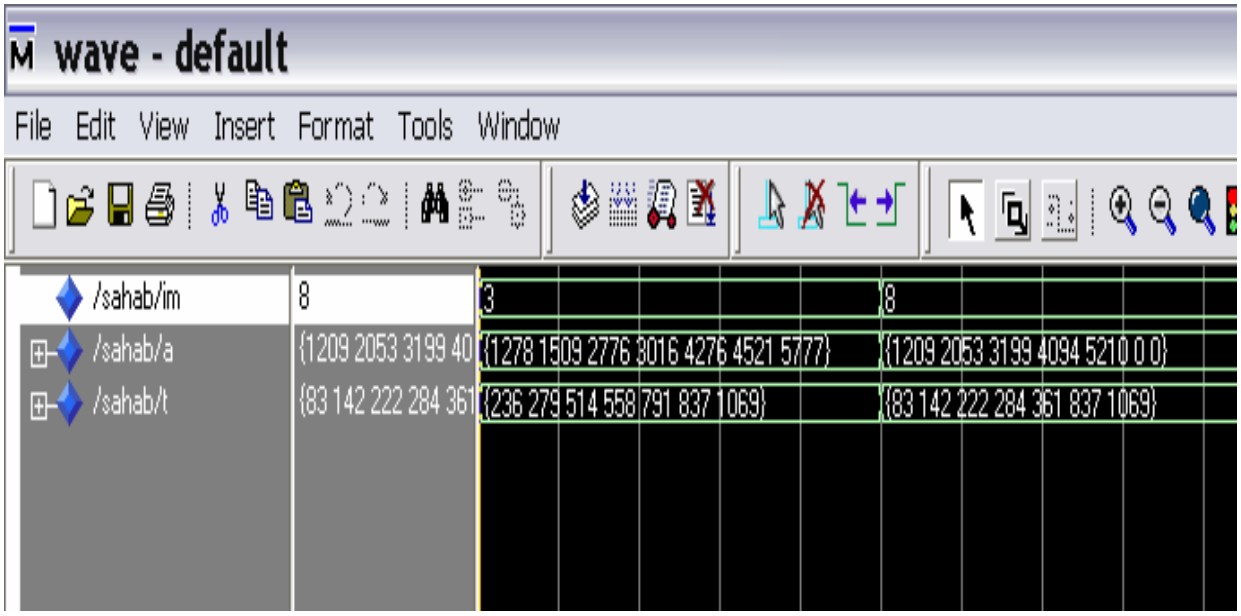


Figure 4.4 simulation pour im =0.3 et im=0.8

2.4- Interprétation des résultats

A partir des résultats des tableaux 4.1 et 4.2 on voit que les angles calculés par notre programme sont très proches à celles calculées par l’ordinateur en utilisant les deux équations (2.30) pour les k impaires et (2.31) pour les k paires.

.Pour augmenter la précision du programme il faut augmenter la précision de calcul de la division et de la puissance.

3-programmation de la commande

3.1- Description

Dans la partie précédente on a programmé une fonction qui à partir des données A0, B0, p des tableaux 2.1 et 2.2 calcule les angles de commutation et les instants temporels équivalents, dans le programme suivant on programme les six signaux de commande.

On a comme entrées une horloge pour la synchronisation, et une commande d qui représente la variation de im (d=100im), pour d<50 on prend 7 angles de commutation (m=7), pour 50<d<70 on prend 5 angles (m=5) et pour 70<d<100 on prend 3 angles (m=3). Ces angles représentent le quart de la période ; pour le deuxième quart on a une symétrie par rapport au premier quart et pour les deux derniers quarts on a une antisymétrie par rapport aux deux premiers (figure 2.1).

3.2- Le programme

3.2.1- les signaux de commande

On mémorise les valeurs de B0,B1,p dans une mémoire on prend en considération que B0, B1<10000 et p < 20 ; et pour chaque entrée d =100im on calcule les angles de commutation α_k en utilisant la fonction cal3 et on déduit les valeurs temporelles équivalent t_k en utilisant la fonction cal6 .

Pour commander l’onduleur triphasé on a besoin de six signaux de commande deux à deux complémentaires s1, s2, s3, s4, s5, s6.

A partir de t_k trouvées pour une commande d donnée, on fait un programme qui donne le signal s1; et on déduit :

s2 qui est le compliment de s1,s3 qui est déphasé de 120° par rapport à s1, s4 qui est le compliment de s3, s5 qui est déphasé de 240° par rapport à s1, s6 qui est le compliment de s4

a- Organigramme:

L’organigramme de l’algorithme précèdent est décrit par la figure 4.5.

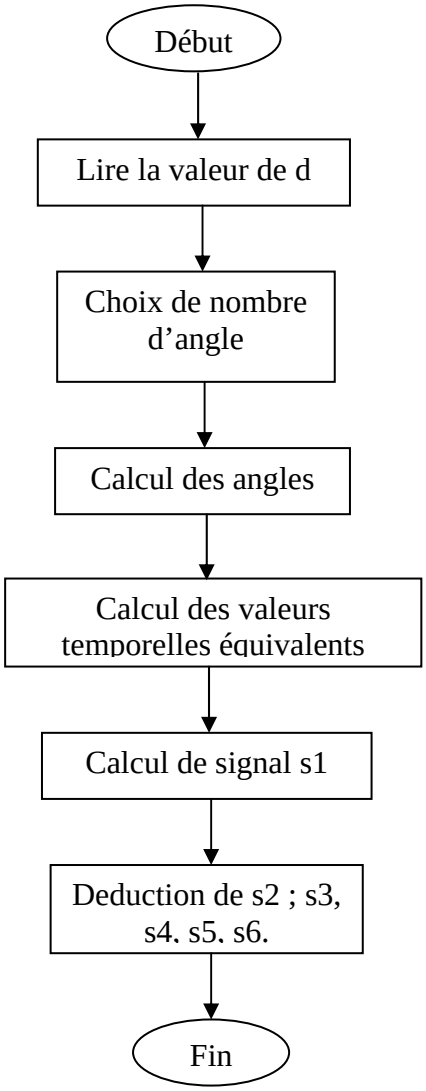


Figure 4.5 : L’organigramme du programme qui calcul les six signaux de commande.

b- Programme

On donne dans les tableaux suivant une explication d’une partie du programme qui calcule les signaux de commande:

Une partie du programme qui fait le calcul de t_k pour d=10 (im=0.1)
if d=10 then
-- on choisit 7 angles de commutation
for k in 1 to 7 loop
if (k mod 2)=0 then
-- pour k impaire
t(k)<=cal6(ma(k).v2,ma(k).v3,ma(k).v1,10,K,7,'0');

```
else
--pour k paire
t(k)<=cal6(ma(k).v2,ma(k).v3,ma(k).v1,10,K,7,'1');
end if ;
end loop;
```

Une partie du programme qui fait le calcul de s1(clkout) et s2(clkoutt)

```
-- calcul de s2 qui rst le compliment de s1
clkoutt<=not clkout;
-- Pour les deux premiers quarts deux période de s1
if n ='1'then
if l =t(1)+1 then clkout <='0';c:='0';
else if l =t(2)+1 then clkout<='1';c:='1'  ;
else if l =t(3)+1 then clkout <='0';c:='0';
else if l =t(4)+1 then clkout<='1';c:='1' ;
else if l =t(5)+1 then clkout<='0';c:='0' ;
else if l =t(6)+1 then clkout<='1'; c:='1';
else if l =t(7)+1 then clkout<='0';c:='0';
else if l=1 then clkout <='1';c:='1';
else clkout<= c;
end if;end if;end if;end if;end if;end if;end if;end if;
else
--pour les deux derniers quarts de période
if l =t(1)+1 then clkout <='1';c:='1' ;
else if l =t(2)+1 then clkout<='0';c:='0'  ;
else if l =t(3)+1 then clkout <='1';c:='1' ;
else if l =t(4)+1 then clkout<='0';c:='0' ;
else if l =t(5)+1 then clkout <='1';c:='1' ;
else if l =t(6)+1 then clkout<='0';c:='0'  ;
else if l =t(7)+1 then clkout <='1';c:='1' ;
else if l=1 then clkout<='0';c:='0'  ;
```

```
else clkout<= c;
end if; end if; end if; end if; end if; end if; end if; end if;
end if;
.
.
.
if im=6 then
-- l  représente le quart de période
if l< 833 then  l:=l+1; else l:=1; g:=g+1 ; end if ;
.
.
End if;
if g=3 then n:=not n; g:=1 ; end if; end if;
```

3.2.2- Les résultats de simulation

Les résultats de simulation par le MODEL SIM sont montrés sur les figures (4.6), (4.7), (4._).

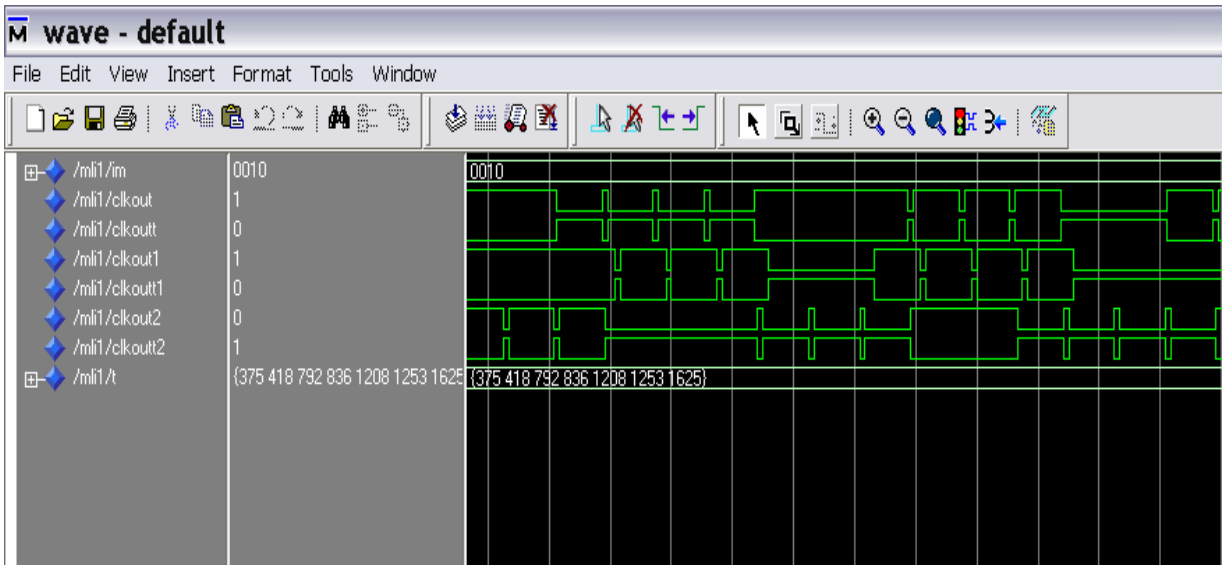


Figure 4.6 : Résultat de simulation pour im =0.20 et m=7.

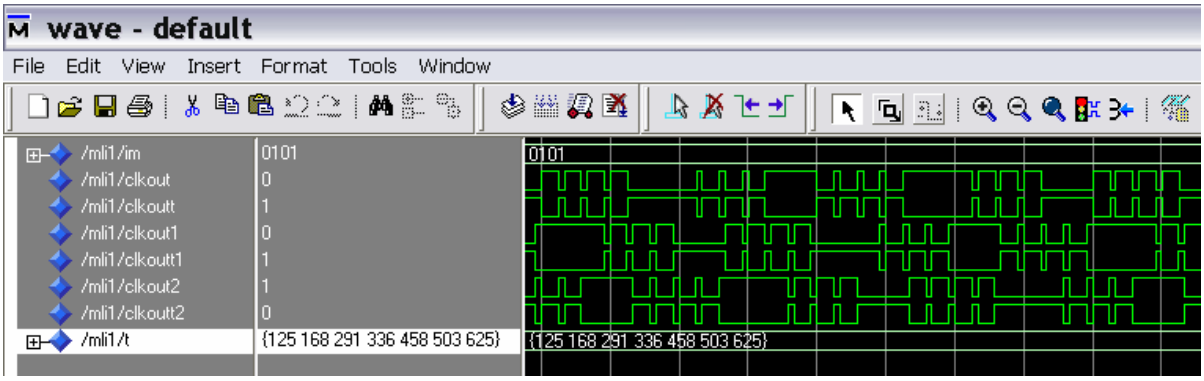


Figure 4.7 : Résultat de simulation pour im=0.50 et m=7.

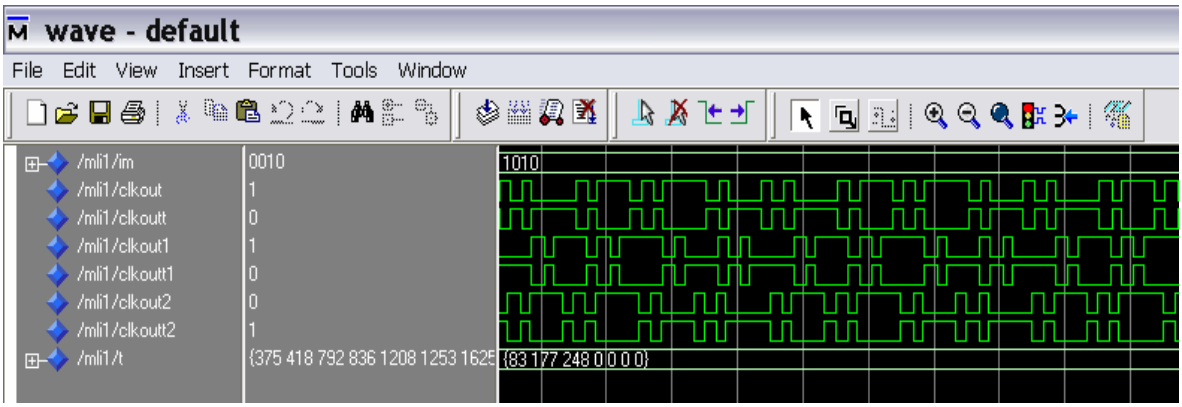


Figure 4.8 : Résultat de simulation pour im=1.00 et m=3.

3.3- Interprétation des résultats

A partir des résultats de simulation, on voit bien :

- m commutation par quart d’onde,
- Les signaux sont deux à deux complémentaire,
- s1et s3 sont déphasés d’un tiers de période (même pour s 3 et s4).

4- Implémentation de la commande sur la carte de développement VIRTEX-II

4.1- Introduction

Pour implémenter et tester notre commande sur la carte de développement VIRTEX-II, on propose le programme qui est structuré dans la figure 4.9. Il est composé de trois blocks, un diviseur de fréquence, un block pour calculer les valeurs temporelles t_k et un block qui donne les six signaux de commande.

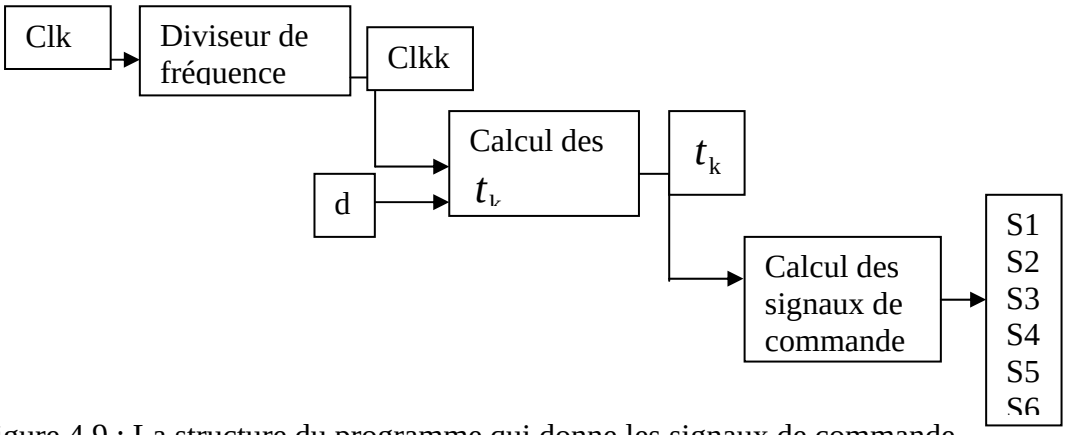


Figure 4.9 : La structure du programme qui donne les signaux de commande

4.2- Le diviseur de fréquence

4.2.1- Description

Dans notre programme on a utilisé une fréquence f = 0.1 MH, mais sur la carte de développement il existe deux autres fréquences f1 = 24 MH, f2 = 100MH, donc pour tester notre programme on a besoin d'un diviseur de fréquence. On choisit par exemple la fréquence f2 et on fait une division sur 1000.

a- Organigramme:

L'organigramme de l'algorithme précèdent est décrit par la figure 4.10.

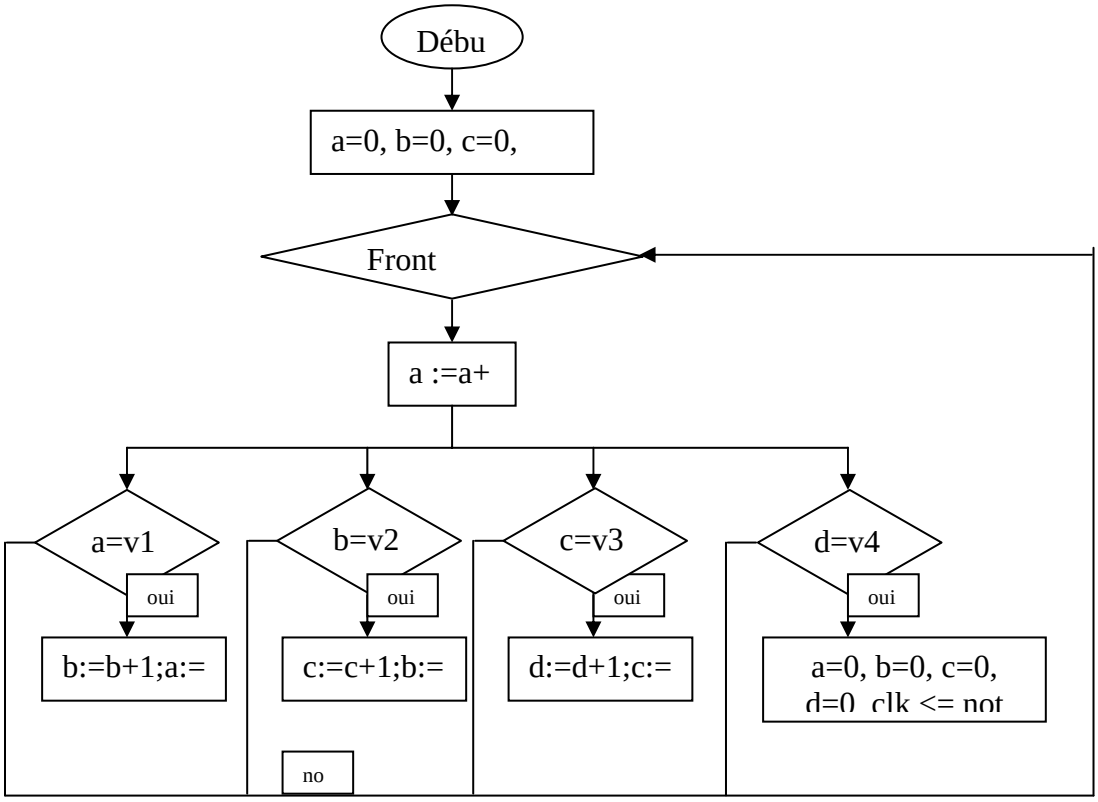


Figure 4.10 : L'organigramme du programme de diviseur de fréquence

b-Le programme

le tableau suivant donne une explication du programme

Le programme d’un diviseur de fréquence

-- l’entrée clk de fréquence fclk
p1: PROCESS (clk)
-- définition des variables internes
variable a:integer:=0 ; variable d:integer:=0 ; variable c:integer:=0 ; variable
b:integer:=0 ;
BEGIN
if(clk'event AND clk='1') THEN a:= a+1; if (a=10) then b:=b+1;a:=0;end if ;
if (b=10) then c:=c+1;b:=0;end if;if (c=5) then d:=d+1;c:=0;end if;
if (d=1) then a:=0;b:=0;d:=0;c:=0;clkk<=not clkk; end if;end if;
END PROCESS p1;
-- la sortie clkk de fréquence fclkk = fclk / (2*a*b*c*d)

c- Simulation

La figure 4.11 montre une simulation d’un diviseur de fréquence par 16

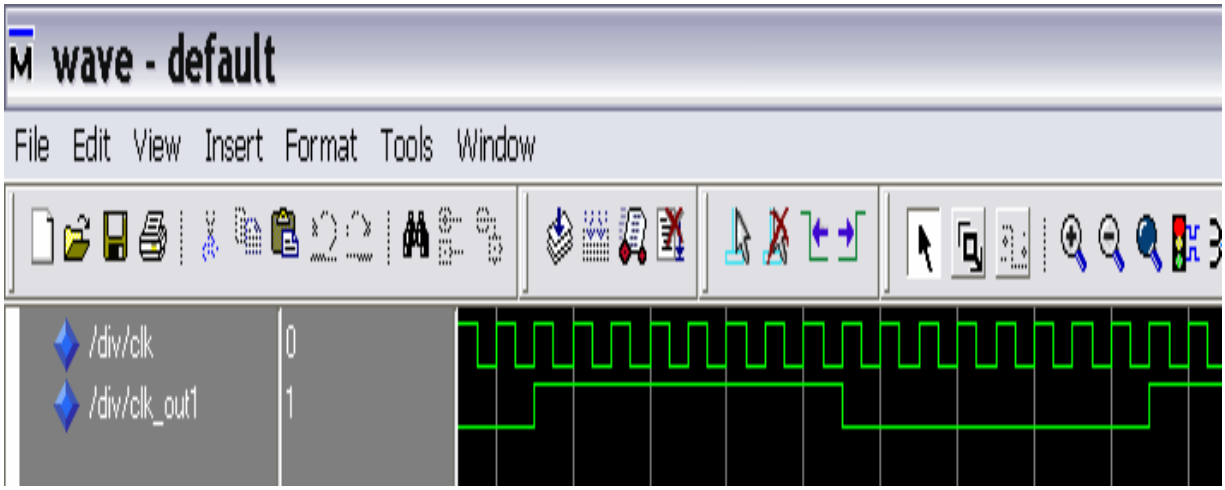


Figure 4.11 : Résultat de simulation d’un diviseur de fréquence sur 16

4.3- Implémentation

Pour implémenter notre commande on choisit sur la carte comme horloge de synchronisation l’horloge qui est de fréquence 24 MHz, et comme commande ‘d’ l’interface (DIP switch) qui contient 8 entrées exploitables par l’utilisateur qui peuvent être mis statiquement à un état haut ou bas, et on dirige les six signaux de sortie vers l’interface LDVS de transmission pour qu’elles peuvent être visualisés sur un oscilloscope voire la figure 4.12, et le tableau 4.3.

Les horloges	Pin
Clk	A11 (24 MHz)
La commande ‘d’	
D1	B4
D2	A4
D3	C4
D4	C5
Les signaux de sortie	
S1	H2
S2	J2
S3	F4
S4	K1
S5	G4
S6	E3

Tableau 4.3 : Affectation des broches d’entrées sorties

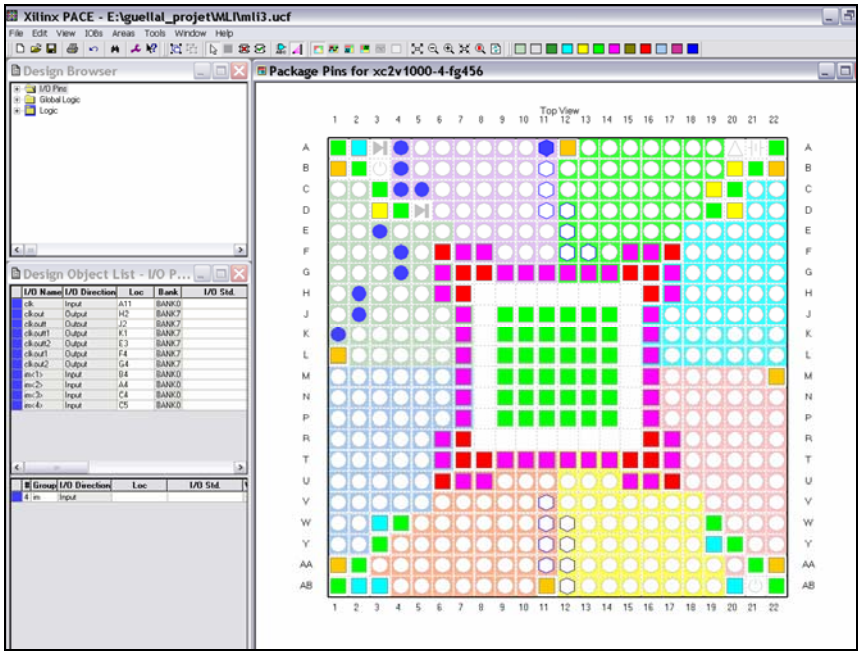


Figure 4.12 : Aperçu de l’outil d’affectation des broches d’entrées sorties

4.4- Visualisation sur l’oscilloscope

Pour vérifier le fonctionnement de notre commande on fait la visualiser sur un oscilloscope pour plusieurs valeurs de ‘d’.
Les figures (4.13), (4.14), (4.15), (4.16) montre la visualisation de notre commande pour im=0.1, im=0.3, im=0.5, et im=0.8.

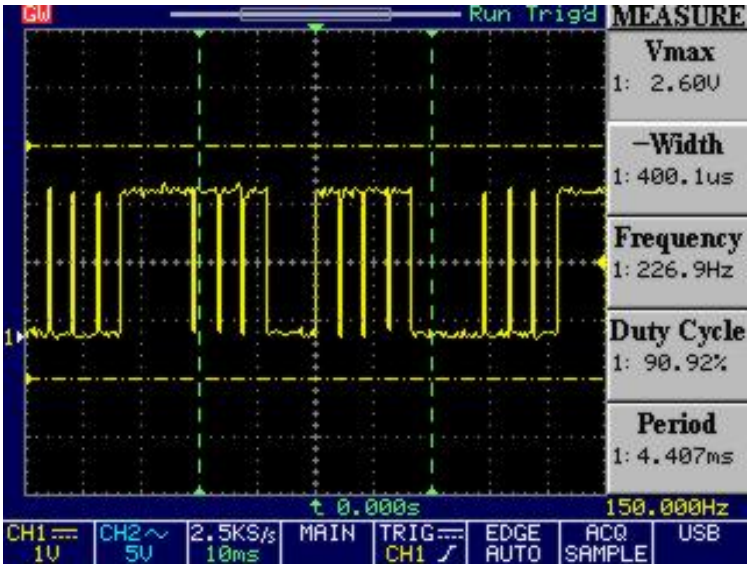


Figure 4.13 : Visualisation de la commande s1 pour im=0.2 et m=7

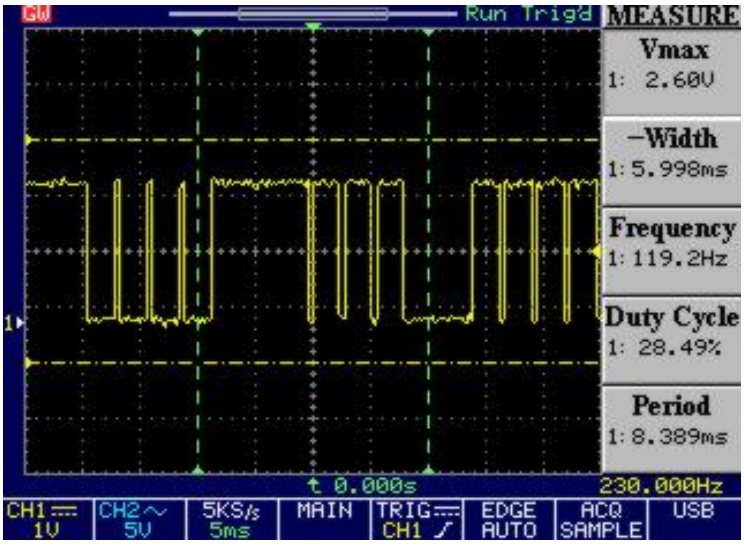


Figure 4.14 : Visualisation de la commande s1 pour im=0.3 et m=7

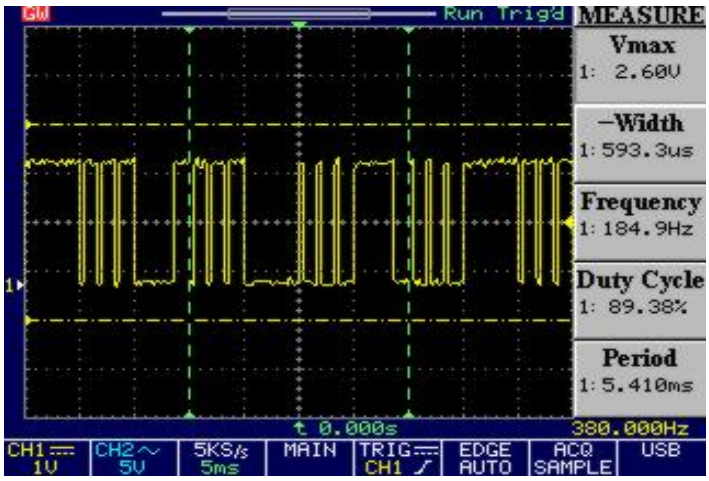


Figure 4.15 : Visualisation de la commande s1 pour im=0.5 et m=7

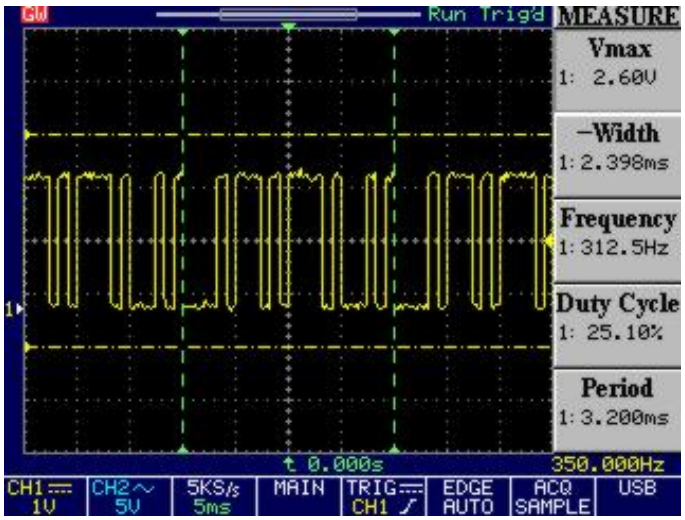


Figure 4.16 : Visualisation de la commande s1 pour im=1.0 et m=3

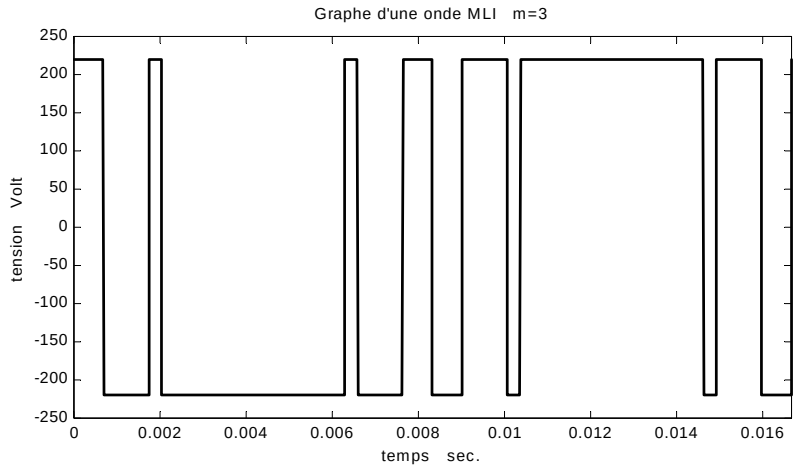


Figure 4.18 : Graphe d’une tension MLI avec m égal à 3

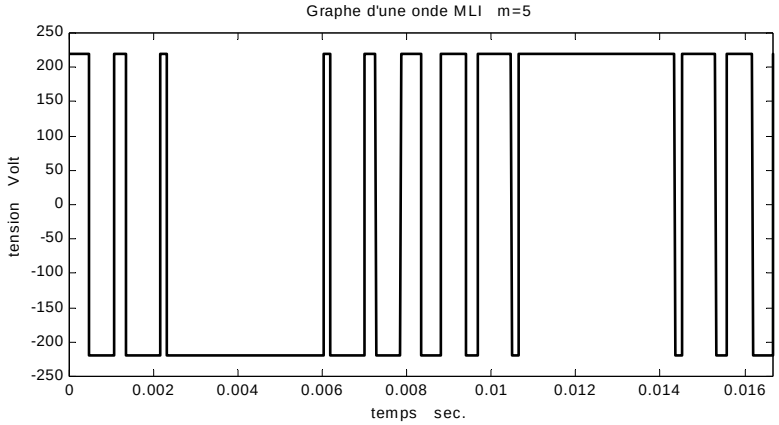


Figure 4.19 : Graphe d’une tension MLI avec m égal à 5.

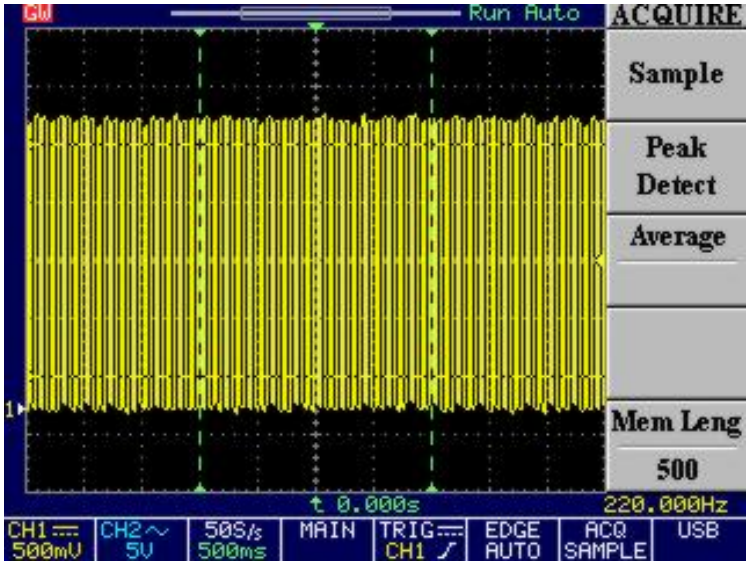


Figure 4.20 : Visualisation de la commande s1 pour im=0.1 et m=23

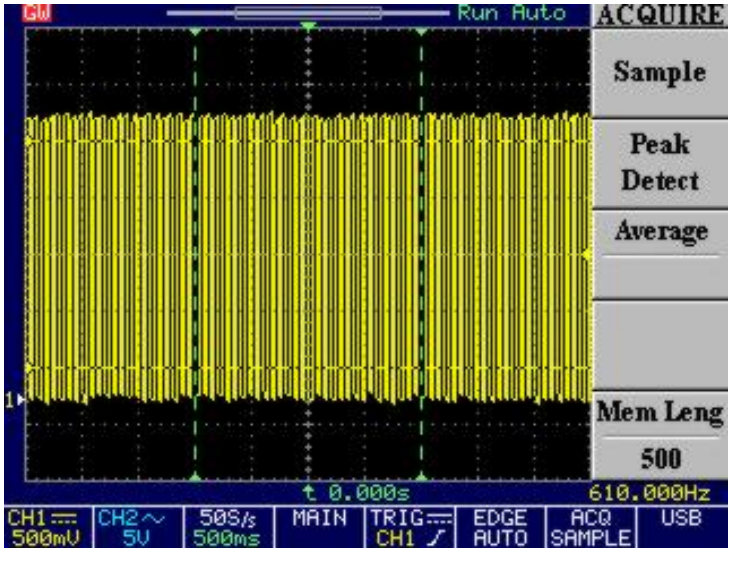


Figure 4.21 : Visualisation de la commande s1 pour im=0.1 et m=63

4.5- Conclusion

Dans ce chapitre on a proposé un programme pour implémenter la commande MLI ‘on line’ sur un circuit FPGA.

Avant d’implémenter ce programme sur la carte on l’a simulé par le ModelSim, et pour vérifier le fonctionnement de notre commande on l’a implémenté sur un circuit FPGA VIRTEX-II et on a visualisé les résultats sur un oscilloscope.

D’après les figures de simulation et de visualisation, on voit que les résultats sont très proches, et 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 (figure 4.18, 4.19).

Notre programme nous a permet d’aller jusqu’à 63 angles de commutation par quart d’onde (figures 4.20, 4.21)

Conclusion

Conclusion générale

L'objectif de notre étude est *la commande de vitesse du moteur asynchrone triphasé* dans toute la gamme de vitesses, de zéro à la vitesse nominale du moteur.

L'objectif principal de ce mémoire est d'implémenter une commande MLI calculée, avec élimination d'harmoniques sélective et asservissement du fondamental, sur un circuit FPGA. Pour cela un *nouvel algorithme 'on-line' est utilisé pour obtenir* les angles de commutation approximés.

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 remédier à ce problème on utilise généralement une alimentation MLI programmée avec élimination d'harmonique sélective et asservissement du fondamental de Patel et Hoft. Mais l'utilisation d'une telle alimentation 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 une précédente étude, par approximation des angles de commutation exacts calculés par la méthode de Newton-Raphson. Il a été montré que le nouvel algorithme 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 on a démontré la possibilité de générer une MLI calculée avec élimination d'harmonique sélective et asservissement du fondamental on-line en se basant sur cet algorithme d'approximation et sur les capacités des circuits FPGA.

On a même essayé une stratégie de commande permettant de varier la vitesse sur toute la gamme de variation tout en optimisant l'élimination des harmoniques ceci a été fait en variant le nombre d'angles de la MLI.

Le nouvel algorithme 'on-line', utilisé 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.

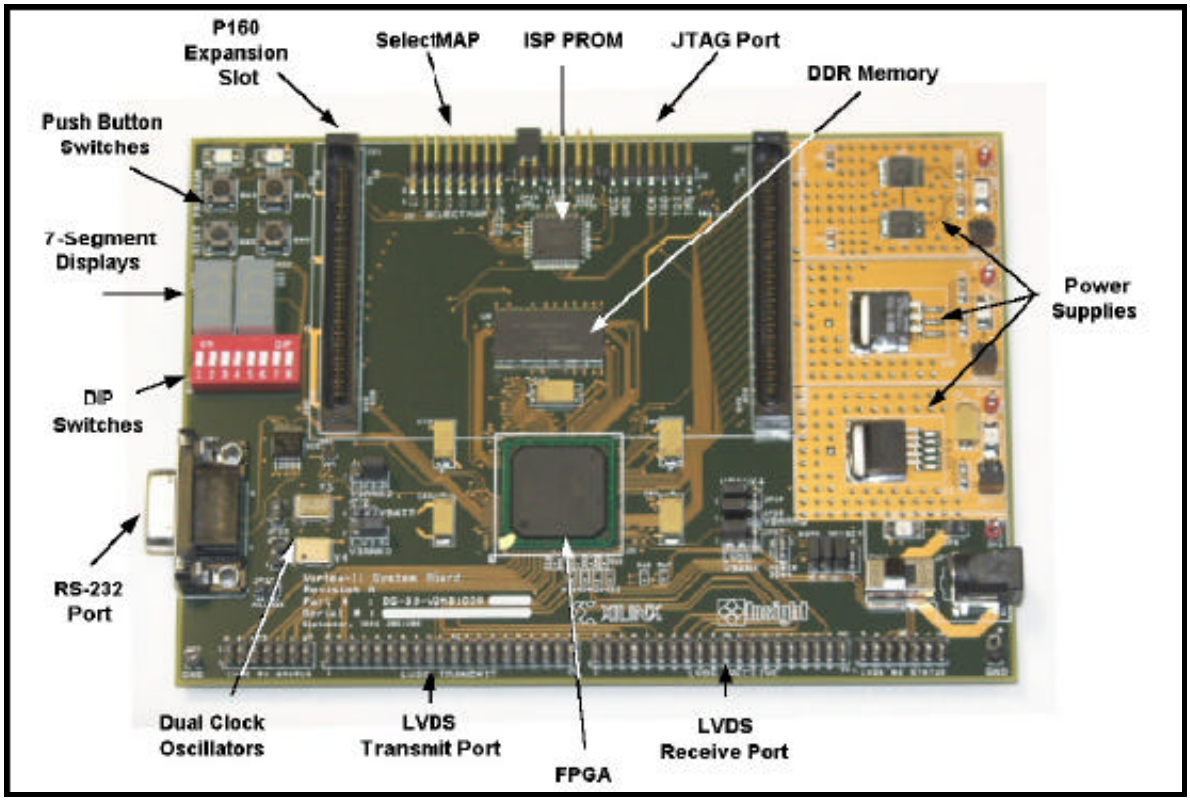
Ceci a pour conséquences une réduction importante des pulsations de couple aux faibles vitesses et de l'échauffement du moteur.

La difficulté qu'on a rencontré dans la programmation c'est que le langage VHDL ne possède pas une bibliothèque qui compile les nombres réels donc on a été obligé de travailler avec les nombres entiers. On a trouvé aussi des difficultés avec la division et la puissance entières.

On a trouvé que le circuit FPGA nous permet de faire et de développer des applications importantes avec des vitesses rapides. Pour l'interface XILINX il nous a permis de vérifier nos programmes et il nous donne le fichier « .bit » qui sera implémenté dans le circuit FPGA. Le ModelSim aussi nous a permis de simuler nos programmes avant d'être implémentés. Ainsi que la carte de développement VIRTEX-II de MEMEC nous a permis d'implémenter notre application et de vérifier son fonctionnement.

En fin, ce projet nous a permet de comprendre la façon de développer un projet sur les circuits FPGA.

Annexes



Carte de développement « Memec Design Virtex-II »

REFERENCES BIBLIOGRAPHIQUES

[1] C.C.CHAN and W.C.LO, 'Control stategy of PWM inverter drive system for electric vehicles.', IEEE Transactions on Industrial Electronics, vol .IE-34, Novembre 1987, pp447-56.

[2] R. CHAUPRADE et F. MILSANT, 'Commande électronique des moteurs à courant alternatif', Editions Eyrolles, 1980

[3] G. SEGUIER et F. NITELET, 'Electronique Industrielle', Technique et Documentation, 1977.

[4] H.asmukh S.PATEL And Richard G.HOFT, 'Generalised Techbiques of Harmonic Elimination and Voltage Contrôle in Thyristor Inverters', IEEE Transactions on Industry Application,
Part I: Harmonic Elimination, Vol.IA-9, No3, May/June 1973, pp 310-17.
Part II: Voltage Control Techniques, Vol.IA-10, No5, September/October 1974. pp 666-73.

[5] 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.

[6] A.GOURDIN et M.BOUMAHRAT, 'Méthodes Numériques Appliquées', OPU, seconde édition , 1991.

[7] M. KHIDER, *commande de vitesse en temps réel d'un moteur asynchrone triphasé,* Mémoire de Magister, Ecole Nationale Polytechnique, Algérie, 2003.

[8] MEMEC Design, *Virtex-II V2MB1000 Development Bord user's Guide,* http://legacy.memec.com/solutions/refernce/xilinx, Décembre 2002.

[9] XILINX, *Virtex-II Platform FPGAs : Complete Data Sheet*, http://www.xilinx.com, Mars 2005.

[10] Guy SEGUIER,'L'électronique de puissance', éditions DUNOD.