REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE NATIONALE POLYTECHNIQUE

Département d'Automatique

Projet de fin d'études

Pour l'obtention du diplôme

D'Ingénieur d'Etat et du diplôme de Master en Automatique

THÈME

Implémentation des techniques MLI sur un circuit FPGA

Etudié par :

DJAAFRI Noureddine
REBAI Aissa

Proposé et dirigé par :

Pr. E.M. BERKOUK
Pr. C. LARBES

Juin 2012

Laboratoire de Commande des Processus
Ecole Nationale Polytechnique, 10, AV. Hassen Badi, El-Harrach, Algérie

Remerciements

Nous remercions le bon Dieu de nous avoir accordé toute la patience, le courage, la volonté et la motivation qui nous ont permis d'achever ce travail.

Nous exprimons notre profonde gratitude, notre grand respect et notre sincère reconnaissance à notre promoteur le Pr. BERKOUK de l'Ecole Nationale Polytechnique pour avoir assumé la lourde responsabilité de nous encadrer, de nous avoir orienté et conseiller tout au long de ce travail ainsi pour la confiance qu'il nous a accordée .

Nous remercions également notre co-promoteur le Pr. LARBES de l'Ecole Nationale Polytechnique pour ses précieux conseils, son suivi, sa disponibilité et son aide.

Nous remercions chaleureusement les membres du jury pour l'honneur qu'ils nous ont fait en acceptent d'évaluer notre projet.

Nous souhaitons aussi remercier tous les enseignants de l'Ecole Nationale Polytechnique d'Alger, et en particulier, Nos professeurs d'Automatique qui nous ont encadrés auparavant et tous nos enseignants pour les connaissances qu'ils nous ont transmis, leur disponibilité et leurs efforts.

Que tous ceux qui ont contribué de près ou de loin à la réalisation de ce modeste travail trouvent ici l'expression de notre sincère gratitude.

Dédicaces

Je dédie ce modeste travail à ma chère mère qui a été de tout temps, la plus proche, qui n'a jamais ménagé leur effort, leur encouragement et leur soutien avec abnégation et patience

Sans oublier mes Grandes mères.

A la mémoire de mon père.

A mes frères et ma sœur.

A l'ensemble des amis que j'ai connu pendant mes études et à ceux qui ont prodigué leurs vifs conseils, encouragements et témoigné de leur amitié.

Aissa

A ma très chère mère,

A mes frères mes sœurs que j'adore,

A toute ma famille,

A mon ami Sofiane, ainsi qu'à tous mes amis,

A vous tous,

A la mémoire de mon très cher père.

Noureddine

ملخص:

الهدف من هذا العمل هو دراسة و إنجاز ذراع محول مستمر ـ متناوب. طرق التحكم المستعملة هي تقنية تعديل عرض النبضة (MLI)، وتقنية عرض النبضة مع حذف توافقيات الإشارة والتحكم في المركبة الأساسية التي وضعها كل من باتل و هوفت.

لذلك قمنا بمحاكاة التقنيات المستعملة باستعمال البرنامج PSIM . بعد ذلك قمنا ببرمجة هذه التقنيات على دارة مبرمجة من نوع FPGA Xilinx باستعمال لغة VHDL وبطاقة التطوير Memec Design. في الأخير تم إنجاز الدارة الكاملة للذراع، ثم تجريبها والتعليق على النتائج المحصل عليها.

كلمات مفتاحية: تعديل عرض النبضة المبرمجة، تعديل عرض النبضة، محول مستمر ـ متناوب، دارة FPGA، لغة VHDL، باتل، هوفت.

Résumé :

L'objectif de ce travail est l'étude et la réalisation d'un bras d'onduleur. Les commandes utilisées sont : une MLI calculée On-Line (en temps réel) à élimination d'harmoniques et asservissement du fondamental, et une commande MLI engendrée triangulo-sinusoïdale. Dans ce mémoire nous avons commencé par simuler le système à réaliser dans l'environnement PSIM, puis nous avons implémenté l'algorithme MLI on-line et la commande engendrée sur un circuit FPGA XC2V1000 de XILINX, en utilisant le langage VHDL et la carte de développement Memec Design V2MB1000. Ensuite nous avons réalisé le bras de l'onduleur, Enfin nous avons présenté des tests sur ce bras et des discussions des résultats obtenus.

Mots clés : MLI engendrée, MLI calculée, onduleur, circuits FPGA, langage VHDL, Xilinx, Patel et Hoft.

Abstract :

The aim of this work to design and implement an arm of inverter. The technique used is an On-Line harmonic elimination PWM (HEPWM), with fundamental control and a generated technique PWM. They are a scalar techniques which allows generating high-quality output spectra.

We have used a co-simulation based on PSIM to improve the performances of the techniques used. Then the V2MB1000 Memec Design development card was used to implement the On-Line algorithm in a XILINX FPGA (XC2V1000), using the VHDL language.

Finally, we have built the arm of the inverter. The test of global system and the discussion of the results obtained are given.

Keywords: PWM, HEPWM, harmonics, FPGA, VHDL, Voltage source inverter.

Table des matières

Table des figures

Liste des tableaux

Liste et sigle des abréviations

T : La période de la tension de sortie

E : La tension d'entrée de l'onduleur

v_0 : La tension de sortie instantanée

v_r : La tension de référence

i_0 : Le courant instantanée de sortie

V_0 : La valeur efficace de la tension de sortie

V_1 : La valeur efficace du fondamental

V_n : La valeur efficace du n^{ime} harmonique

f_0 : La fréquence de la tension de sortie

f_p : La fréquence du porteuse

f_r : La fréquence du signal de référence

P_{01} : La puissance de sortie

m : L'indice de modulation

r : Le taux de modulation (coefficient de réglage)

a_n : Coefficient de série de Fourier

b_n : Coefficient de la série de Fourier

δ_m : Largeur de l'impulsion m

α_m : L'angle de commutation de la m^{ime} impulsion

$E^{(k)}$: La matrice Jaccobienne

$\Delta\alpha^{(k)}$: Vecteur d'erreur

α_s : Angle de séparation

VHDL : Very high speed integrated circuit Hardware Description Language
FPGA : Field Programmable Gate Array
MLI : Modulation de Largeur d'Impulsion
PWM : Pulse With Modulation
MLIU : Modulation de Largeur d'Impulsion Uniforme
HF : Facteur d'harmonique
THD : Distorsion totale d'harmonique
LOH : Harmonique le plus bas ordre
ASI : Alimentation Sans Interruption
MAS : Machine Asynchrone
GTO : Gate Turn Off
IGBT : Insulated Gate Bipolar Transistor
NPC : Neutral Point Clamped
PLD : Programmable Logic Device
CPLD : Complex Programmable Logic Device
PSD : Programmable System Devices
PLA : Programmable Logic Array
ASIC : Application Specific Integrated Circuit
SRAM : Static Random Access Memory
LCA : Logic Cells Arays
IOB : Input Output Bloc
CLB : Configurable Logic Bloc
RAM : Random Access Memory
ROM : Read Only Memory
TTL : Transistor-Transistor Logic
CMOS : Complementary Metal Oxide Semiconductor
MOS : Metal Oxide Semiconductor
EPROM : Erasable Programmable Read Only Memory
EEPROM : Electrically Erasable Programmable Read Only Memory
LUT : Look Up Table
RTL : Registre Translate Level

Introduction Générale

Une des branches de l'électronique en pleine expansion est l'électronique de puissance qui traite et contrôle l'énergie électrique ainsi que sa conversion en d'autres formes d'énergie afin de fournir des tensions et des courants aux différents types de charges selon les applications. On distingue fondamentalement les conversions suivantes : alternatif/continu, continu/alternatif, alternatif/alternatif, et la conversion continu/continu.

L'un des principaux problèmes liés aux convertisseurs statiques, et qui dépend de la stratégie de commande utilisée, est celui de la présence inévitable des harmoniques dans l'onde de sortie. Les harmoniques ont des effets néfastes sur le moteur, ils provoquent l'échauffement excessif, les pulsations du couple et la saturation du circuit magnétique.

Dans un onduleur à Modulation Largeur d'Impulsions (en Anglo-Saxon, Pulse With Modulation), au lieu de de former chaque alternance d'une tension de sortie avec un seul créneau rectangulaire , on la forme de plusieurs créneaux de largeurs convenables. Les schémas des onduleurs restent les mêmes, c'est la commande des interrupteurs qui est modifiée : la fréquence des commutations est supérieure à la fréquence des grandeurs de sortie.

La multiplication du nombre des impulsions formant chacune des alternances d'une tension de sortie présente deux avantages importants :

- Elle repousse vers les fréquences plus élevées les harmoniques de la tension de sortie, ce qui facilite le filtrage.
- Elle permet de faire varier la valeur du fondamental de la tension de sortie, même avec les onduleurs à deux interrupteurs par phase.

L'essor de la Modulation de Largeur d'Implusion est lié aux progrès sur les semi-conducteurs de puissance ; l'augmentation du nombre des commutations entrainerait des pertes excessives si on n'avait pas réussi à réduire les pertes à chacune des commutations.

Le plus souvent on détermine en temps réel les instants de fermeture et d'ouverture des interrupteurs à l'aide d'une électronique de commande analogique ou numérique ou en faisant simultanément appel à ces deux techniques. Pour cette détermination on utilise les intersections d'une onde de référence et d'une onde de modulation. Dans certain applications, on calcule et on mémorise au préalable les instants de commande, les interrupteurs sont ensuite commandés par un microprocesseur.

Le but de notre projet est d'implémenter trois commandes MLI sur une carte FPGA. La première c'est la commande MLI engendrée basée sur l'intersection entre un signal de référence et un signal porteuse, la deuxième c'est une commande MLI basée sur le calcul des angles exactes de commutations par la résolution d'un système d'équation non linéaire, et on termine par une commande MLI avec asservissement du fondamental et élimination des harmoniques sélectives. Dans cette technique les angles de commutation sont approximées à l'aide d'un algorithme on-line.

Chapitre 1

Généralités sur les onduleurs

1.1 Introduction

Les onduleurs constituent une fonction incontournable de l'électronique de puissance, présente dans les domaines d'applications les plus variés. La forte évolution de cette fonction s'est appuyée, d'une part, sur le développement de composants à semi-conducteurs entièrement commandables, puissants, robustes et rapides, d'autre part, sur l'utilisation quasi-généralisée des techniques dites « modulation de largeur d'impulsion ». ces dernières s'appuyant sur les performances en fréquence de découpage permises par les premiers.

1.2 Définition des onduleurs

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

L'onduleur est dit autonome s'il peut fixer lui-même la fréquence et la valeur efficace de sa tension de sortie. Un redresseur commandé tout thyristors peut fonctionner en onduleur. Ce type d'onduleur est dit " non autonome " ou encore " assisté " car il ne permet de fixer ni la fréquence ni la valeur efficace des tensions du réseau alternatif dans lequel il débite [23].

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

1.3 Divers types d'onduleurs

Comme nous avons cité précédemment il y a deux types principales d'onduleurs qui sont les onduleurs autonomes de tension et de courant.

1.3.1 Les onduleurs de tension

On appelle onduleurs de tension un onduleur qui est alimenté par une source de tension continue, c'est-à-dire par une source d'impédance interne négligeable. La tension n'est pas affectée par les variations du courant qui la traverse. La source continue impose la tension à l'entrée de l'onduleur et donc à sa sortie.

Le courant à la sortie et donc le courant à l'entrée dépendent de la charge placée du coté alternatif. Cette charge peut être quelconque à la seule condition qu'il n'agisse pas d'une autre source de tension (capacité ou f.e.m alternative) directement branchée entre les bornes de sortie.

1.3.2 Les onduleurs de courant

Un onduleur de courant (souvent appelé commutateur de courant) est alimenté par une source de courant continue. La source continue impose le courant à l'entrée du convertisseur et donc à sa sortie. La tension à la sortie et donc la tension à l'entrée dépendent de la charge placée du coté alternatif [5].

1.4 Les Différents schémas des onduleurs de tension

1.4.1 Les onduleurs de tension monophasés en demi-pont

La topologie de l'onduleur monophasé de tension en demi pont est présentée selon la figure 1.1.Il utilise deux interrupteurs bidirectionnels en courant unidirectionnels en tension et une source de tension à point milieu. Les interrupteurs utilisés sont des composants électroniques de puissance commandables tels que le transistor bipolaire, le GTO, l'IGBT, etc ...

Le principe de fonctionnement de cet onduleur est le suivant [26] : l'interrupteur $tr1$ se met à conduire pendant une demi période soit $T/2$, tandis que $tr2$ est bloqué ; alors la tension instantanée aux bornes de la charge est $E/2$. Mais si au contraire $tr2$ conduit et $tr1$ bloqué pendant $T/2$, la tension instantanée aux bornes de la charge vaut $-E/2$. La loi de commande doit être faite de sorte que les deux interrupteurs ne conduisent pas en même temps. La figure 1.2-a montre l'évolution du courant pour une charge RL, tandis que la figure 1.2-b montre l'évolution du courant de sortie et le temps de conduction des composants pour une charge purement inductive. Le courant dans ce type de charge ne peut pas changer immédiatement avec la variation de la tension de sortie ; si au temps $T/2$ l'interrupteur $tr1$ se bloque, le courant dans la charge ne s'annule pas immédiatement, continuera à circuler dans la diode $D2$ et la charge jusqu'à ce qu'il atteigne zéro. Ce principe est similaire quand $tr2$ est bloqué au temps T ; pendant ce temps, le courant circule dans la partie supérieure de l'onduleur, c'est à dire dans la diode $D1$ et la charge. Ces diodes $D1$ et $D2$ sont connues sous le nom de diodes de réaction ou de retour car elles conduisent l'énergie vers la source. Chaque interrupteur conduit pendant $T/4$; la tension inverse à ses bornes vaut E. Si nous considérons que la tension de sortie est égale à $E/2$ pendant une demi période, nous pouvons trouver la valeur efficace de cette tension comme suit :

$$V_0 = (\frac{2}{T}\int_0^{\frac{T}{2}} \frac{E^2}{4} dt)^{\frac{1}{2}}$$

La tension de sortie instantanée peut être exprimée en série de Fourier comme :

$$v_0 = \sum_{n=1}^{+\infty} \frac{2E}{n\pi} sin(n\omega t) \qquad n = 1, 3, 5, ...$$

Où $\omega = 2\pi f_0$ est la fréquence de la tension de sortie en rad/s. Pour n = 1, l'équation précédente de v_0 donne la valeur efficace de la composante fondamentale comme :

$$V_1 = \frac{2E}{\sqrt{2}\pi} = 0.45E$$

Il faut noter que pour une charge purement inductive, un seul transistor conduit pour $t = T/2$ ou (90^o). En fonction du facteur de puissance de la charge, la période de conduction d'un transistor pourra varier de 90^o à 180^o.

Le courant instantané i_0 pour une charge RL peut être trouvé d'après :

$$i_0 = \sum_{n=1}^{+\infty} \frac{2E}{n\pi\sqrt{R^2 + (nL\omega)^2}} sin(n\omega t - \theta_n) \qquad n = 1, 3, 5, ...$$

Où :

$$\theta_n = \tan^{-1}(n\frac{L\omega}{R})$$

Si I_{01} est la valeur efficace de la fondamentale du courant de charge, la puissance de sortie (pour n=l) est :

$$P_{01} = V_1 I_{01}\cos(\theta_1) = I_{01}^2 R = [\frac{2E}{\sqrt{2}\pi\sqrt{R^2 + (L\omega)^2}}]^2$$

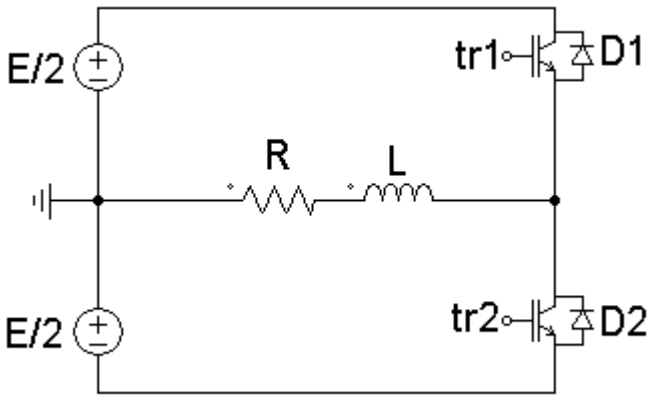


FIGURE 1.1 – Onduleur de tension monophasé en demi-pont

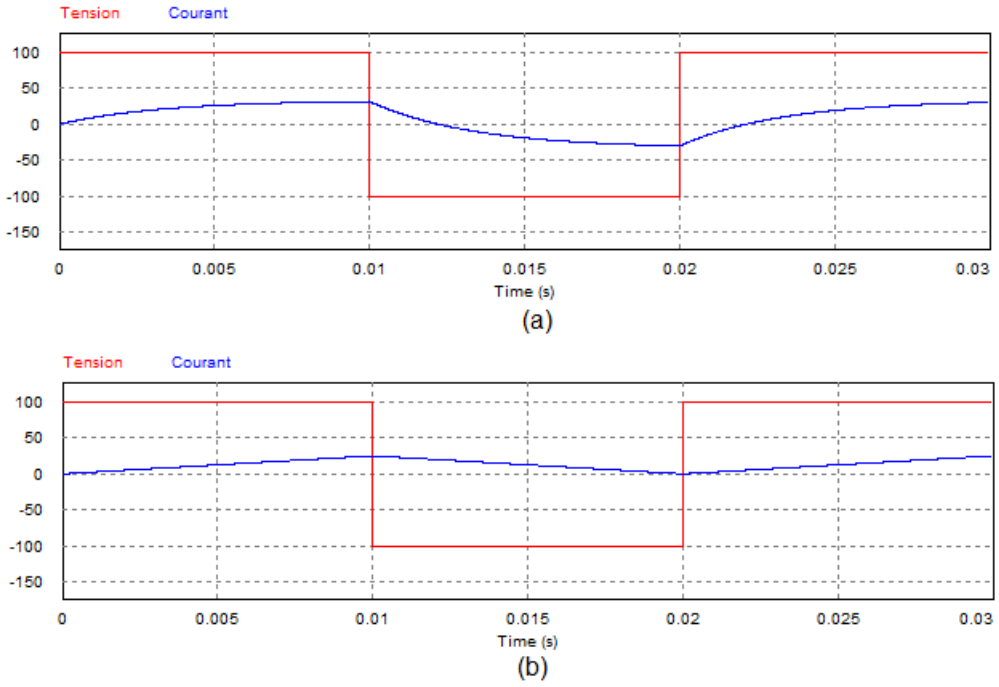


FIGURE 1.2 – courant pour une charge RL et pour une charge purement inductive

1.4.2 Les onduleurs de tension monophasés en pont

La structure de base de l'onduleur de tension en pont se présente à la figure 1.3. Il se compose de quatre interrupteurs à semi conducteurs avec des diodes antiparallèles. La charge est branchée entre les deux bras de l'onduleur aux points milieux.

Le principe de fonctionnement de cet onduleur est le Suivant [4] : quand $tr1$ et $tr2$ sont fermés, $tr3$ et $tr4$ sont ouverts, la charge est connectée à la tension E, alors la tension de sortie v_o à la charge est maintenant égale à E, Cette valeur est aussi obtenue si $tr3$ et $tr4$ sont fermés, $tr1$ et $tr2$ sont ouverts, la tension à la charge sera égale à $-E$. Ainsi, l'onduleur de tension en pont peut fournir trois niveaux de tension :$E, 0, -E$; ce qui n'est pas le cas pour l'onduleur monophasé en demi-pont, qui donnait deux niveaux de tension $+E/2, -E/2$. Cette caractéristique est un avantage de l'onduleur monophasé en pont, car cela permet de faire varier et régler la tension de sortie v_o de $\pm E$.

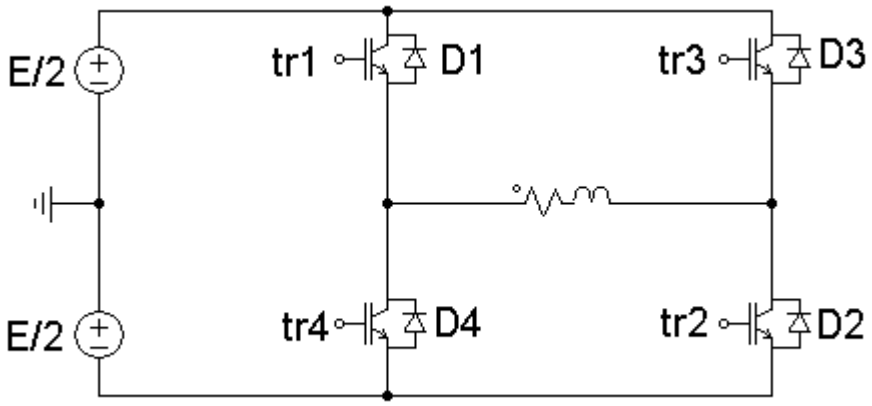


FIGURE 1.3 – Onduleur de tension monophasé en pont

Il faut aussi remarquer que la tension inverse maximale du blocage aux bornes des interrupteurs et des diodes de retour est la même pour les deux types d'onduleurs, si la tension E à la même valeur.

Cependant, avec les conditions égales pour les deux onduleurs, la puissance délivrée par l'onduleur en pont est quatre fois plus importante et sa composante fondamentale harmonique est deux fois plus élevée que celle de l'onduleur en demi pont. La tension de sortie efficace v_o de l'onduleur à un créneau par alternance est égale à :

$$V_o = \sqrt{\frac{2}{T}\int_0^{\frac{T}{2}} E^2 dt} = E$$

Et l'expression de la série de Fourier est :

$$v_o = \sum_{n=1}^{+\infty} \frac{4E}{n\pi} sin(n\omega t) \qquad n = 1, 3, 5, ...$$

où $\omega = 2\pi f_0$ est la fréquence de la tension de sortie en rad/s. Pour n = 1, l'équation précédente de v_0 donne la valeur efficace de la composante fondamentale comme :

$$V_1 = \frac{2\sqrt{2}E}{\pi} = 0.90E$$

1.4.3 Les onduleurs triphasés

Trois onduleurs monophasés en demi-ponts peuvent être connectés en parallèle pour constituer un onduleur triphasé de tension (figure 1.4).

Les onduleurs monophasés sont utilisés pour des applications de faible puissance, alors que les onduleurs triphasés couvrent la gamme des moyennes et des fortes puissances. L'objectif de cette topologie est de fournir une source de tension triphasée, dont l'amplitude, la phase et la fréquence sont contrôlables [4] .

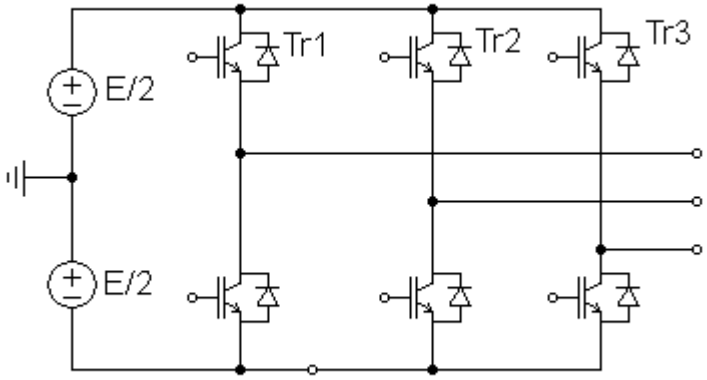


FIGURE 1.4 – Les onduleurs de tension triphasés

Il est facile de généraliser ce principe à un onduleur p-phasé en pont qui comporterait donc p cellules. Dans cette configuration, la cellule de commutation peut donc être considérée comme une phase de l'onduleur, la composante alternative de sa tension de sortie constituant une tension simple du système polyphasé [17].

1.4.4 Les onduleurs multi-niveaux

Les onduleurs multi-niveaux triphasés dont la topologie est basée sur la mise en série d'onduleurs partiels par phase (figure 1.5). Généralement, les tensions continues alimentant les onduleurs partiels. Le bras d'onduleur à n niveau permet de régler les échanges d'énergie entre *n-1* sources de tension et une source de courant (charge)[22].

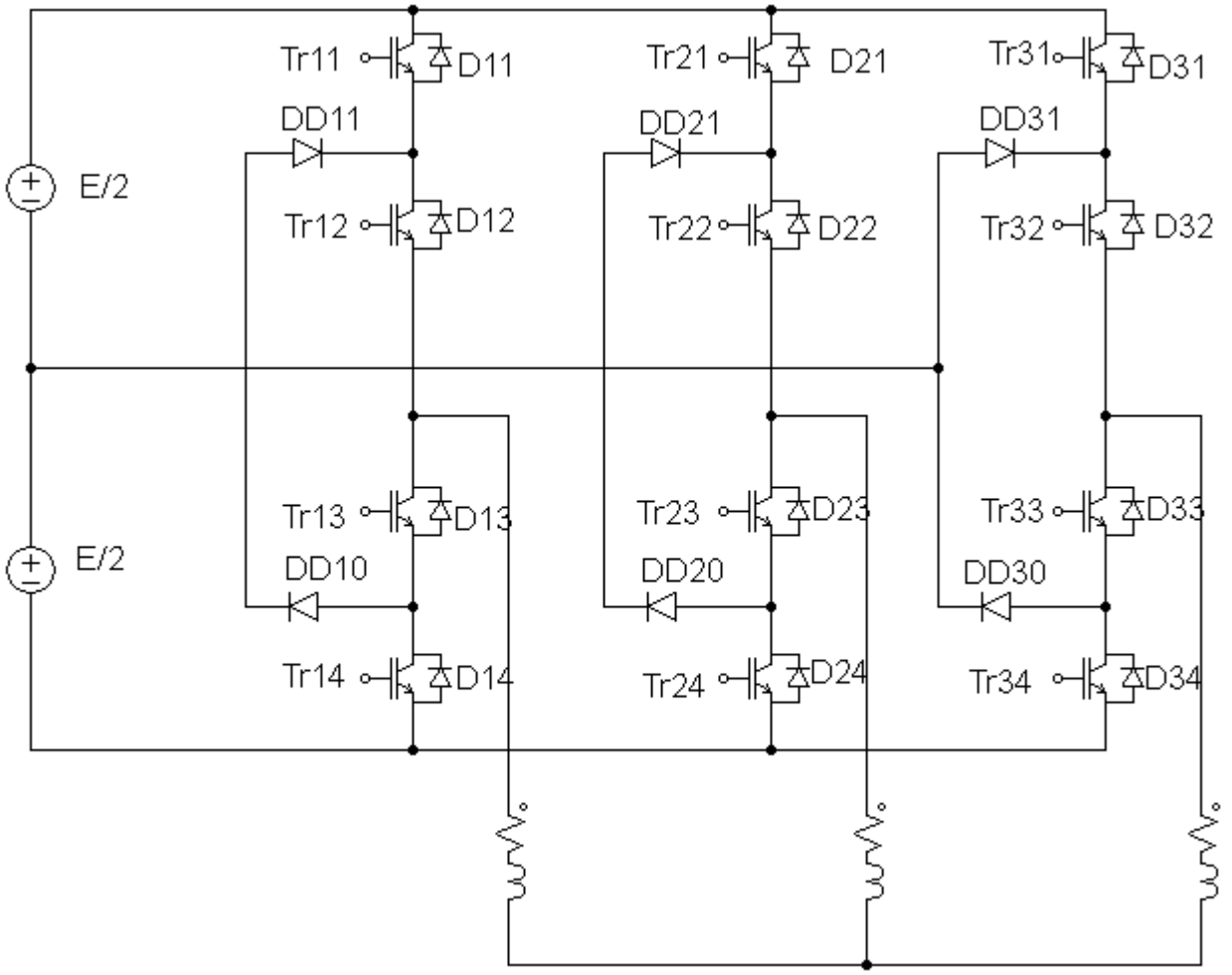


FIGURE 1.5 – Onduleur à 3 niveaux à structure NPC [22]

1.5 Caractérisation de la tension fournie par l'onduleur

Les onduleurs sont caractérisés par la qualité de la tension alternative qu'ils fournissent à leurs sorties. Celle-ci est évaluée par trois paramètres : le facteur d'harmonique, la distorsion totale d'harmonique et l'harmonique de plus bas ordre.

1.5.1 Le facteur d'harmonique (HF)

Il représente la mesure de la contribution individuelle d'une harmonique (par exemple celle du n^{ime} harmonique).

$$HF_n = \frac{V_n}{V_1}$$

Avec : V_1 :valeur efficace du fondamental.

V_n :valeur efficace du n^{ime} harmonique.

1.5.2 La distorsion totale d'harmonique (THD)

Elle représente la mesure de rapprochement dans le forme entre l'onde de tension et son fondamental :

$$THD = \frac{\sqrt{\sum_{n=2}^{+\infty} V_n^2}}{V} = \frac{\sqrt{V^2 - V_1^2}}{V} < 1;$$

Ou encore :

$$THD = \frac{V}{V_1} > 1$$

1.5.3 L'harmonique le plus bas ordre (LOH)

C'est celui dont la fréquence est la plus proche du fondamental et dont l'amplitude est supérieure ou égale à 3% de celle du fondamental.

1.6 Le problème de filtrage

La tension en sortie du convertisseur continu/alternatif n'est pas sinusoïdale. En effet, les semiconducteurs travaillant en commutation, la tension de sortie sera toujours constituée de morceaux de tension continue.

Cette tension non sinusoïdale peut être considérée comme la somme d'un fondamental (que l'on souhaite) et de tensions de fréquences multiples de celle du fondamental, les harmoniques (que l'on ne souhaite pas). Ces tensions harmoniques provoquent la circulation de courants harmoniques.

L'objectif du filtrage dépend du système considéré [18] :

– Dans le cas des alimentations sans interruption, on souhaite une tension analogue à celle délivrée par le réseau donc sinusoïdale. On va donc filtrer la tension avec des condensateurs. L'impédance en alternatif d'un condensateur étant $Z_c = \frac{1}{C\omega}$, on voit que pour les harmoniques de tension de rang croissants, cette impédance est de plus en plus faible.

– Dans le cas des variateurs de vitesse pour MAS, on souhaite que le courant soit sinusoïdal pour éviter les couples harmoniques générateurs de pertes et de vibrations. On va donc lisser le courant avec des inductances. L'impédance en alternatif d'une inductance étant $Z_l = L\omega$, on voit que pour les harmoniques de courants de rang croissants, cette impédance est de plus en plus grande. Dans le cas d'un MAS, l'inductance propre du stator suffit généralement a assurer un filtrage convenable.

1.7 Domaines d'application

Les deux grands domaines applications classiques des onduleurs de tension sont les alimentations de secours et la commande des moteurs alternatifs. Ils sont également caractéristiques de deux grandes familles, respectivement celle des systèmes à fréquence fixe et celle des systèmes à fréquence variable.

1.7.1 Domaine des fréquences fixes

Dans ce premier domaine, l'onduleur de tension est destiné à palier les défaillances ou même l'absence d'un réseau alternatif en recréant la tension correspondante à partir d'une batterie. Le schéma de principe d'une telle application est donné par la figure 1.6

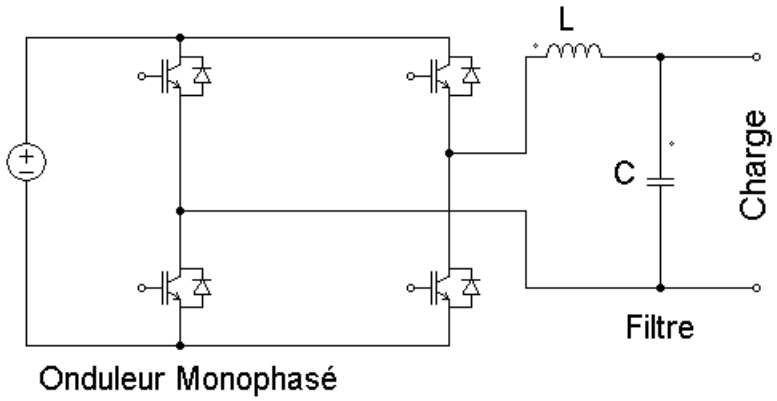


FIGURE 1.6 – Schéma de principe d'un onduleur pour alimentation de secours

Cette application requiert généralement la fourniture d'une tension de sortie très pure, donc sans les harmoniques de découpage, d'où la nécessité d'un filtre de sortie.

1.7.2 Domaine des fréquences variables

Dans ce deuxième domaine, l'onduleur est généralement alimenté par l'intermédiaire d'un réseau alternatif. La source continue est donc un redresseur suivi d'un filtre d'entrée qui peut avoir une double fonction, à savoir éliminer les composantes harmoniques de courant issues de l'onduleur, et les composantes harmoniques de tension dues au redressement.

Une autre différence notable par rapport au cas précèdent réside dans le domaine de fonctionnement beaucoup plus étendu tant en fréquence fondamentale (de quelques hertz à quelques centaine de hertz) qu'en amplitude[17]. Le domaine d'applications des onduleurs de tension le plus connu est sans doute celui de la variation de vitesse des machines à courant alternatif. Le schéma de principe d'une telle application est donné par la figure 1.7.

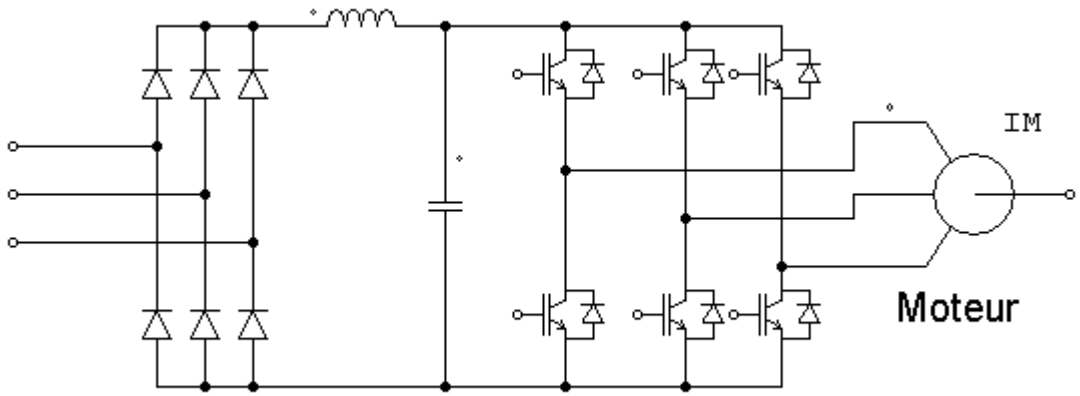


FIGURE 1.7 – Schéma de principe d'un onduleur pour la conduite d'un moteur asynchrone

1.8 Commande des onduleurs par techniques M.L.I

1.8.1 Introduction

L'objectif principal de techniques de commande, appliquées aux onduleurs, est de permettre l'obtention d'ondes de tension alternatives, d'amplitude et de fréquence fondamentale réglables, en éliminant ou en repoussant le plus loin possible les composantes harmoniques parasites résultant du découpage. Quelle que soit la forme de l'onde alternative recherchée (le plus souvent sinusoïdale), l'établissement de la stratégie de commande devra, prendre en compte la façon dont ces techniques

vont pouvoir s'insérer dans les boucles de contrôle et de régulation qui sont obligatoirement présentes dans toutes les applications des onduleurs.

1.8.2 Technique MLI multiple

Lorsqu'on veut réduire le contenu harmonique, on utilise plusieurs impulsions dans chacune des alternances de la tension de sortie. Cette technique est connue sous le nom de MLI multiple. La génération des signaux de commande pour permettre la conduction et le blocage des transistors est montrée sur la figure 1.8 obtenue en comparant un signal de référence avec une porteuse triangulaire. La fréquence du signal de référence règle la fréquence de sortie f_0 et la fréquence f_p du signal porteuse détermine le nombre d'impulsions durant la période p. Le taux de modulation contrôle l'amplitude de la tension de sortie. Ce type de modulation est également connue sous le nom de Modulation en Largeur d'Impulsions Uniforme (MLIU «Uniform Pulse Width Modulation »). Le nombre d'impulsions par période est :

$$p = \frac{f_p}{f_0} = m$$

Où $m = \frac{f_p}{f_0}$ est l'indice de modulation. La variation de taux de modulation r de 0 à 1 fait varier la largeur d'impulsion de 0 à π/p et la tension de sortie de 0 à E. La tension de sortie d'un onduleur en pont est donnée par la figure 1.8-b pour une MLI uniforme. Dans cette technique la largeur de chaque impulsion est constante tel que : $0 < \delta < \frac{\pi}{2p}$[22].

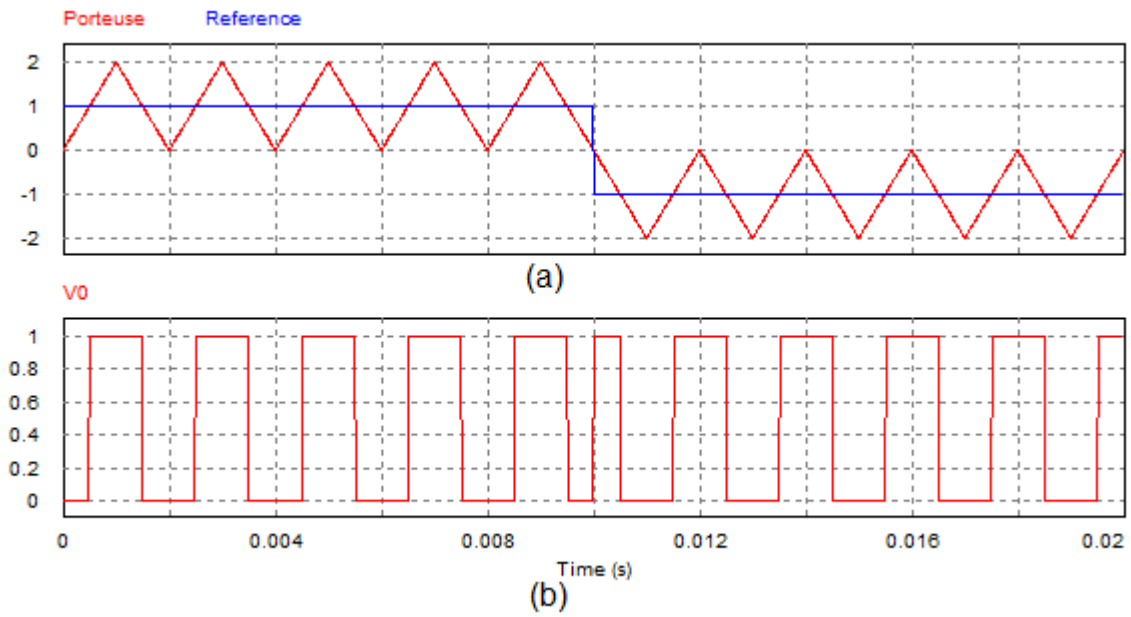


FIGURE 1.8 – MLI multiple

1.8.3 Technique MLI engendrée

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

1.8.4 Technique MLI vectorielle

La technique de modulation vectorielle est une technique numérique pour laquelle l'objectif est de générer une onde MLI de tension $v(t)$ à la sortie de l'onduleur dont la valeur moyen sur chaque période de découpage T est égale à celle de tension sinusoïdale de référence $v_r(t)$ sur cette période. Ceci est effectué à chaque période d'échantillonnage en sélectionnant les états appropriés des interrupteurs parmi la table d'excitation de l'onduleur à deux niveaux et en déterminant la durée d'application de chacun des états. La sélection des états et la détermination des durées sont basées

sur la transformation triphasée (abc)-biphasée (α, β).

La théorie du transformation triphasée-biphasée dit que les grandeurs triphasées dans la sommes des composantes est nulles dans le référentiel fixe (a,b,c) peuvent être représentée par une grandeur biphasée dans un repère fixe (α, β). par exemple le vecteur $[v_{ref}]_{abc} = [v_a, v_b, v_c]$ peut être représenté par le vecteur $[v_{ref}]_{\alpha\beta} = [v_\alpha, v_\beta]$ en utilisant la transformation :

$$v_\alpha = \frac{2}{3}(v_a - \frac{1}{2}(v_b + v_c))$$
$$v_\beta = \frac{\sqrt{3}}{3}(v_b - v_c)$$

De même cette transformation est appliquée aux huit états de l'onduleur à deux niveaux qui correspond à huit vecteurs dans le repère (α, β)(figure 1.9)[22].

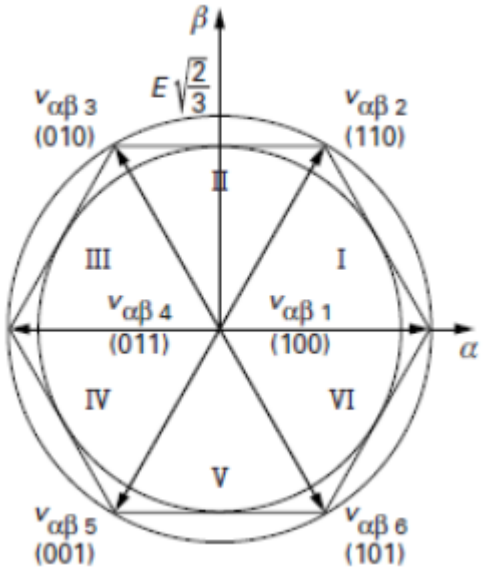


FIGURE 1.9 – Les états d'un onduleur deux niveaux dans le repère (α, β)[18]

Les durées d'application des états des interrupteurs sont déterminées par la relation vectorielle suivante :

$$\overrightarrow{v}_{ref}T = \overrightarrow{v}_i T_i + \overrightarrow{v}_{i+1} T_{i+1} + \overrightarrow{v}_z T_z$$

1.8.5 Technique MLI à élimination des harmoniques (calculée)

La philosophie des MLI calculées est sensiblement différente. Elles sont utilisées lorsque le rapport entre la fréquence de découpage et la fréquence fondamentale est faible, ce qui est fréquent

en forte puissance. Dans ce cas, il y a présence de composantes harmoniques, que l'on cherche à éliminer ou à minimiser en exploitant au mieux le nombre d'impulsions disponibles sur la période fondamentale. Pour atteindre cet objectif, on détermine a priori des formes d'ondes optimisées.

La technique MLI « programmée » est basée l'algorithme de calcul des angles de commutation de Patel et Hoft. Dans cette technique, il est possible d'asservir le fondamental de la tension MLI et d'annuler les amplitudes des *m-1* premiers harmoniques où *m* est le nombre d'angles de commutation de la tension MLI de sortie ou nombre de commutations par quart-d'onde.

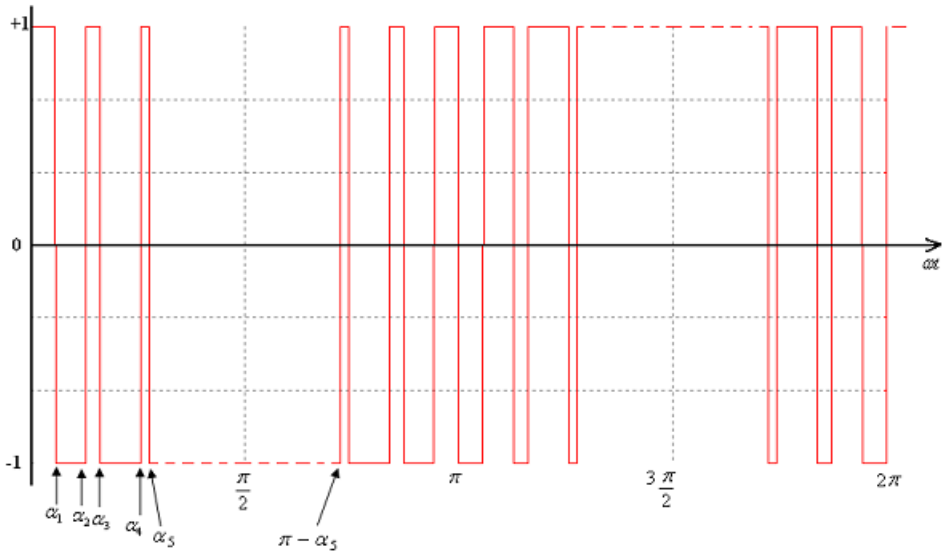


FIGURE 1.10 – MLI calculée

1.8.6 Technique MLI delta

une onde triangulaire est utilisée pour osciller à l'intérieur d'une fenêtre définie $\triangle v$ comme l'enveloppe d'une onde sinusoïdale de référence v_r. La fonction de commutation de l'onduleur, identique à la tension de sortie v_o est générée à partir de la verticale de l'onde triangulaire v_c comme le montre la figure 1.11 .Cette technique de commande est aussi connue sous le nom de « modulation d'hystérésis ». Si la fréquence de l'onde modulée change en maintenant la pente de l'onde triangulaire constante, le nombre d'impulsions et les largeurs des impulsions de l'onde modulante changent aussi.

La fondamentale de la tension de sortie peut être au-dessus de $1E$, et dépend de l'amplitude maximale V_r, et la fréquence f_r de la tension de référence. La modulation delta peut commander le rapport de tension par rapport à la fréquence qui est une caractéristique désirable en contrôle des moteurs à courant alternatif[6].

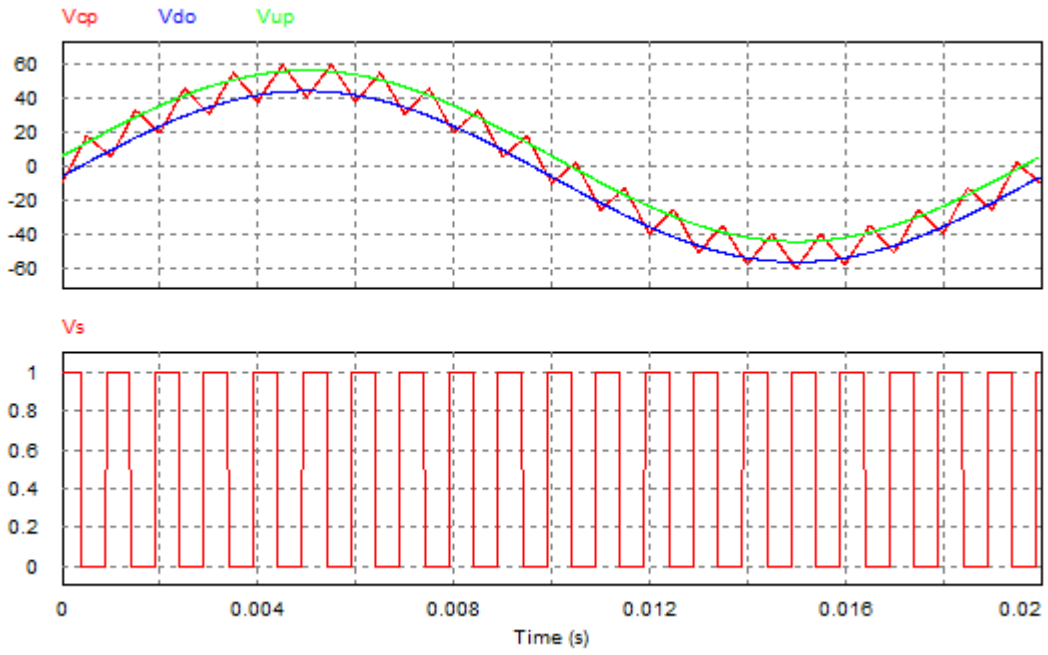


FIGURE 1.11 – MLI delta

1.9 Conclusion

Dans ce chapitre, nous avons présenté les principes des différents schémas des onduleurs de tension ainsi quelque stratégies de commande des onduleurs.

La technique MLI présente l'une des techniques les plus utilisées, elle permet d'obtenir un bon spectre de sortie en augmentant l'indice de modulation mais ça nécessite un espace mémoire considérable pour une implémentation numérique.

Les deux techniques MLI engendrée et programmée avec asservissement du fondamental et élimination d'harmoniques objet de ce mémoire sont présentées au chapitre suivant.

Chapitre 2

Description des techniques MLI engendrée et à élimination des harmoniques

2.1 Technique de la MLI engendrée

2.1.1 Principe

Au lieu de maintenir la largeur de toutes les impulsions constantes, comme dans le cas de la MLI uniforme, dans ce cas, la largeur de chaque impulsion varie en fonction de l'amplitude d'une onde sinusoïdale évaluée au centre de la même impulsion. Le facteur de distorsion et les harmoniques sont réduits significativement. Les signaux de commande sont montrés sur la figure 2.1-a et sont générés en comparant un signal de référence sinusoïdale avec une onde porteuse triangulaire de fréquence f_p. Ce type de modulation est communément utilisé dans les applications industrielles. La fréquence du signal de référence f_r détermine la fréquence f_0 de l'onduleur ; alors que l'amplitude maximale contrôle V_r l'indice de modulation m qui à son tour détermine la tension efficace de sortie V_0. Le nombre d'impulsions par demi cycle dépend de la fréquence de l'onde porteuse. La tension instantanée de sortie de la figure 2.1-b montre que deux transistors d'une même branche ne peuvent conduire à la fois.

2.1.2 Caractérisation de la modulation

Si la référence est sinusoïdale, deux paramètres caractérisent la commande :l'indice de modulation m et le taux de modulation r.

D'ordinaire la modulation est synchrone, c'est-à-dire que la fréquence f_p est un multiple entier de f_r. La tension de sortie est alors vraiment périodique et bien de fréquence $f_0 = f_r$.
Mais dans certains cas la modulation est asynchrone, notamment quand à fréquence de modulation f_p donnée on fait varier de façon continue la fréquence de référence[5].

En modulation synchrone, si m est impair, l'alternance négative de la tension de sortie reproduit au signe près son alternance positive. Le développement en série de Fourier de la tension de sortie ne comporte que des harmoniques impairs.
Au contraire si m est pair, on trouve dans le développement en série de la tension de sortie une composante continue, des harmoniques pairs et impairs. Lorsque l'onduleur est à sortie monophasée, on choisit donc des valeurs impaires de m[5].

2.1.3 Étude de la tension de sortie

La tension efficace de sortie peut être variée en variant l'indice de modulation m. On peut observer que la zone de chaque impulsion correspond approximativement à la zone au dessus de l'onde sinusoïdale entre la moitié des points adjacents de la fin de la période au début des signaux de commande. Si δ_m est la largeur de la m^{ime} impulsion, la tension efficace de sortie peut être écrite sous la forme :

$$V_0 = E\sqrt{\sum_{m=1}^{p} \frac{\delta_m}{\pi}}$$

Où :p est le nombre d'impulsions.
Ainsi le coefficient de la série de Fourier de cette tension est :

$$b_n = \sum_{m=1}^{p} \frac{2E}{n\pi} sin\frac{n\delta_m}{2}[sin(n(\alpha_m + \frac{\delta_m}{2})) - sin(n(\pi + \alpha_m + \frac{\delta_m}{2}))]n = 1, 3, 5, ...$$

Cette technique réduit le facteur de distorsion mieux que la MLI multiple. Elle élimine toutes les harmoniques inférieures ou égales à $2p - 1$. Pour $p = 5$, l'harmonique de rang le petit est le neuvième. Toute fois, la tension de sortie contient des harmoniques. Cette modulation repousse ces

harmoniques dans le domaine des hautes fréquences autour de la fréquence de commutation f_c et ces multiples[1].

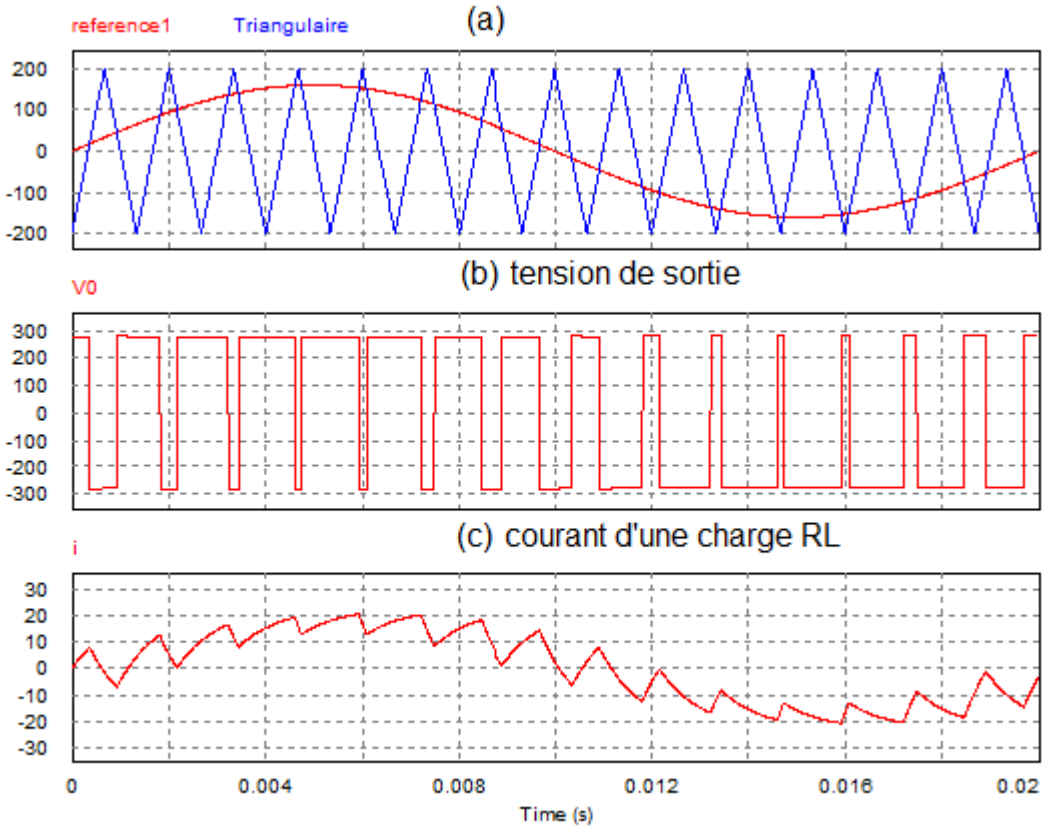


FIGURE 2.1 – MLI engendrée pour un onduleur monophasé

2.1.4 Contrôle de tension d'un onduleur triphasé

Un onduleur triphasé peut être considéré comme étant trois onduleurs monophasés déphasés de 120^o. Ainsi, les techniques que ces derniers utilisent, sont applicables aux onduleurs triphasés. Par exemple, la génération des signaux de commande avec une MLI sinusoïdale est montrée sur la figure 2.2-a. On remarque que les trois ondes de référence sinusoïdales ont déphasées de 120^o entre elles.

Une onde porteuse est comparée avec le signal de référence de la phase correspondante pour générer le signal de commande de cette phase.

La tension de sortie comme l'indique la figure 2.2-b est générée en éliminant la condition que deux dispositifs de commutation de la même branche ne peuvent conduire en même temps.

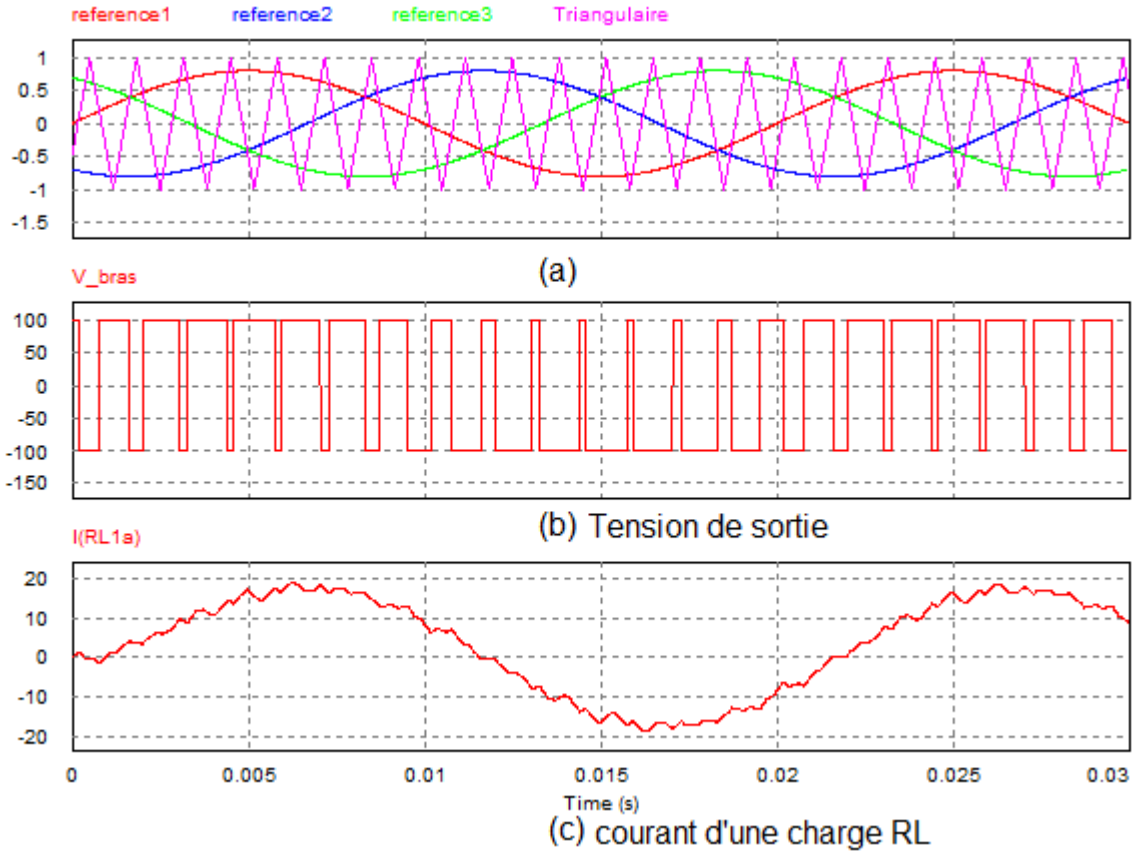


FIGURE 2.2 – Onduleur MLI engendrée triphasée

2.1.5 MLI sinusoïdale modifiée

Selon la caractéristique de la MLI sinusoïdale, les largeurs des impulsions s'approchent de l'amplitude maximale de l'onde sinusoïdale pour ne pas changer significativement avec la variation de l'indice de modulation. Cela est dû à la caractéristique d'une onde sinusoïdale et la technique de MLI sinusoïdale peut être modifiée en appliquant l'onde sinusoïdale durant le début et la fin d'un intervalle de 60^o par demi cycle ; c'est à dire 0 à 60^o et de 120^o à 180^o. Ce type de modulation est connu sous le nom de MLI sinusoïdale modifiée. La composante fondamentale est ainsi augmentée et les caractéristiques des harmoniques sont améliorées. Il réduit le nombre de commutations des dispositifs de puissance et réduit également les pertes dues aux commutations. La figure 2.3 montre le principe de cette modulation[27].

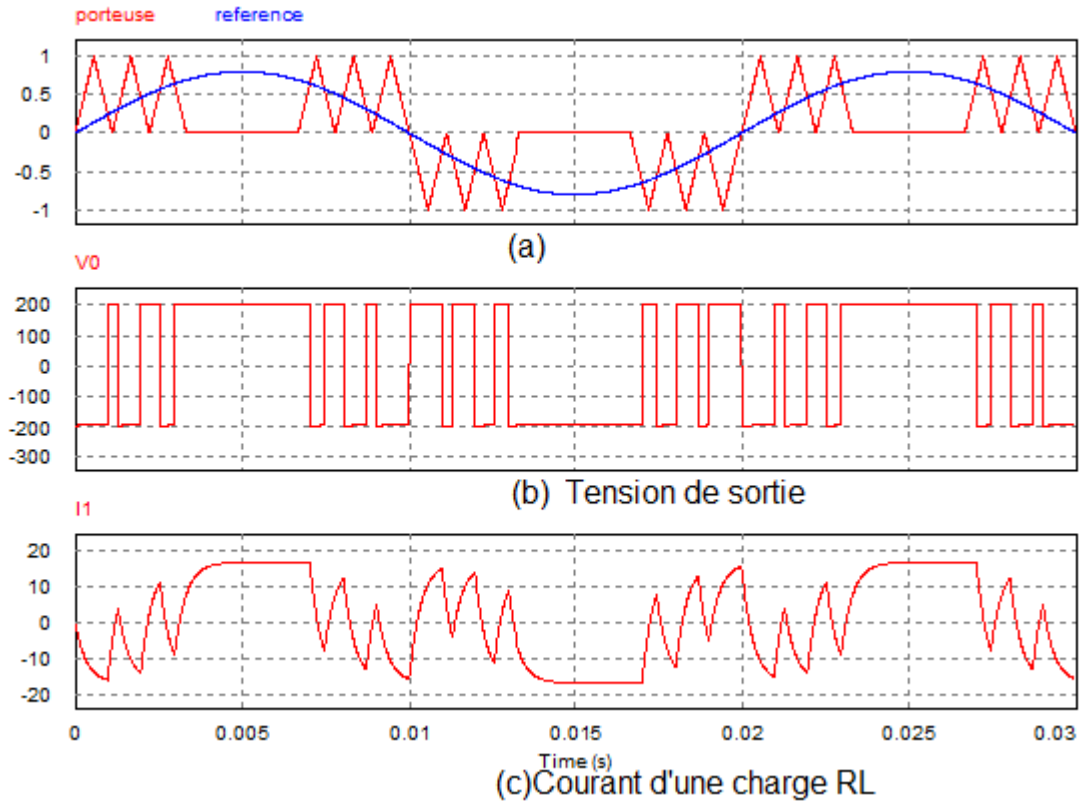


FIGURE 2.3 – MLI modifiée

2.2 Description de la technique de PATEL et HOFT[13]

Considérons un onduleur demi-pont monophasé.

Soit la tension de sortie à deux états de l'onduleur en demi-pont de la figure 2.4. Les angles de commutation impairs $\alpha 1, \alpha 3, \alpha 5, ...$ définissent des transitions négatives, tandis que les angles de commutation pairs $\alpha 2, \alpha 4, \alpha 6, ...$ 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 :

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

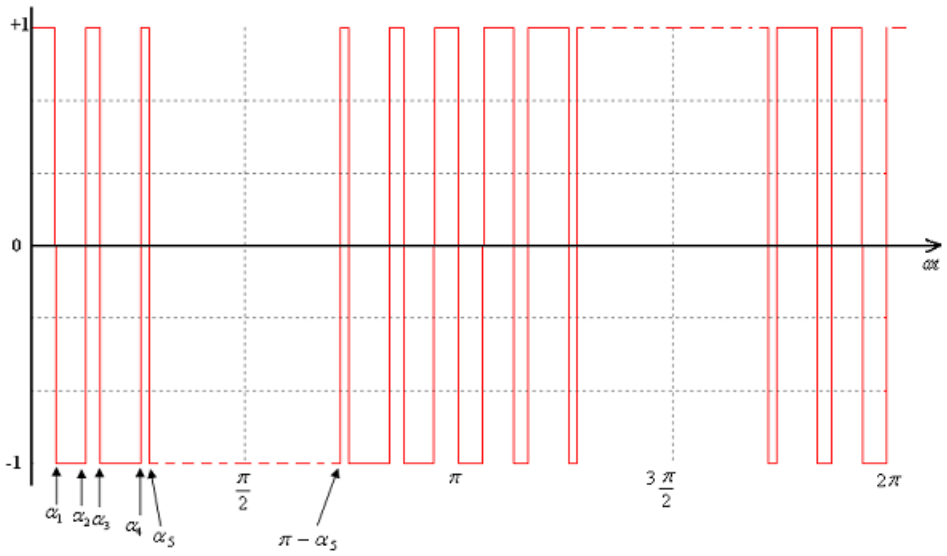


FIGURE 2.4 – graphe d'une tension MLI calculée

Les coefficients a_n et b_n sont donnés par :

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(\alpha)d\alpha$$

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(\alpha)sin(n\alpha)d\alpha \qquad n = 1, 2, 3, ... \tag{2.2}$$

$$b_n = \frac{1}{\pi}\int_0^{2\pi} f(\alpha)cos(n\alpha)d\alpha$$

D'autre part comme f(α) présente une symétrie par rapport a $\frac{\pi}{2}$ et anti-symétrie par rapport a π i.e

$$\begin{cases} f(\alpha) = f(\pi - \alpha) \\ \\ f(\pi + \alpha) = -f(\alpha) \end{cases}$$

Le calcul des valeurs de a_0, a_n et b_n nous donne :

$$a_0 = \frac{1}{\pi}\int_0^{2\pi} f(\alpha)d\alpha = a_0 = \frac{1}{2\pi}[\int_0^{\pi} f(\alpha)d\alpha + \int_{\pi}^{2\pi} f(\alpha)d\alpha]$$

On effectue un changement de variable :

$y = \pi - \alpha \rightarrow d\alpha = -dy$ et $\alpha = \pi - y$.

$\Rightarrow \int_{\pi}^{2\pi} f(\alpha)d\alpha = \int_{0}^{\pi} f(\pi - y)(-dy) = -\int_{0}^{\pi} f(y)dy$

$\Rightarrow a_0 = 0$.

Pour calculer a_n on effectue le changement de variable :

$y = \alpha - \pi \rightarrow d\alpha = dy$.

$\Rightarrow \int_{\pi}^{2\pi} f(\alpha)sin(n\alpha)d\alpha = \int_{0}^{\pi} f(\pi + y)sin(n\pi + ny)(dy) = -\int_{0}^{\pi} f(y)sin(n\pi + ny)dy$

On voit bien que :

$$sin(n\pi + ny) = \begin{cases} sin(ny) & \text{si } n \text{ est pair} \\ -sin(ny) & \text{si } n \text{ est impair} \end{cases}$$

Donc :

$$\int_{\pi}^{2\pi} f(\alpha)sin(n\alpha)d\alpha = \begin{cases} -\int_{0}^{\pi} f(y)sin(n\pi + ny)dy \Rightarrow a_n = 0 & \text{si } n \text{ est pair} \\ \int_{0}^{\pi} f(y)sin(n\pi + ny)dy \Rightarrow a_n = \frac{2}{\pi}\int_{0}^{\pi} f(\alpha)sin(n\alpha)d\alpha & \text{si } n \text{ est impair} \end{cases}$$

Finalement :

$a_n = \frac{2}{\pi}\int_{0}^{\pi} f(\alpha)sin(n\alpha)d\alpha$ avec n impair.

Pour b_n, on effectue le même changement de variable on trouve :

$b_n = \frac{2}{\pi}\int_{0}^{\pi} f(\alpha)cos(n\alpha)d\alpha$ avec n impair.

On peut simplifier les deux expressions par un autre changement de variable : $\alpha = \pi - y$. On trouve :

$$a_n = \frac{4}{\pi}[\int_0^{\frac{\pi}{2}} f(\alpha)sin(n\alpha)d\alpha + \int_{\frac{\pi}{2}}^{\pi} f(\alpha)sin(n\alpha)d\alpha]$$

$$= \frac{2}{\pi}[\int_0^{\frac{\pi}{2}} f(\alpha)sin(n\alpha)d\alpha + \int_{\frac{\pi}{2}}^{0} f(\pi - y)sin(n\pi - ny)(-dy)]$$

$$= \frac{2}{\pi}[\int_0^{\frac{\pi}{2}} f(\alpha)sin(n\alpha)d\alpha + \int_0^{\frac{\pi}{2}} f(y)sin(ny)dy]$$

$$\Rightarrow a_n = \frac{2}{\pi}\int_0^{\frac{\pi}{2}} f(\alpha)sin(n\alpha)d\alpha \qquad \text{avec } n \text{ impair.}$$

Pour b_n :

$$b_n = \frac{2}{\pi}[\int_0^{\frac{\pi}{2}} f(\alpha)cos(n\alpha)d\alpha + \int_{\frac{\pi}{2}}^{\pi} f(\alpha)cos(n\alpha)d\alpha]$$

$$= \frac{2}{\pi}[\int_0^{\frac{\pi}{2}} f(\alpha)cos(n\alpha)d\alpha + \int_{\frac{\pi}{2}}^{0} f(\pi - y)cos(n\pi - ny)(-dy)]$$

$$= \frac{2}{\pi}[\int_0^{\frac{\pi}{2}} f(\alpha)sin(n\alpha)d\alpha - \int_0^{\frac{\pi}{2}} f(y)cos(ny)dy]$$

$$\Rightarrow b_n = 0 \qquad \forall n$$

Donc on obtient finalement les expressions des coefficients a_0, a_n et b_n :

$$a_0 = 0$$

$$a_n = \frac{4}{\pi}\int_0^{\frac{\pi}{2}} f(\alpha)sin(n\alpha)d\alpha \qquad n = 1, 3, 5, 7, ... \tag{2.3}$$

$$b_n = 0 \qquad \forall n$$

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

$$a_n = \frac{4}{\pi}[\int_0^{\alpha_1}(1)sin(n\alpha)d\alpha + ... + \int_{\alpha_{m-1}}^{\alpha_{m-2}}(-1)^{m-1}sin(n\alpha)d\alpha + \int_{\alpha_m}^{\frac{\pi}{2}}(-1)^m sin(n\alpha)d\alpha$$

$$= \frac{4}{n\pi}[(1 - cos(n\alpha_1)) + (cos(n\alpha_2) - cos(n\alpha_1)) + ... + (-1)^m(cos(n\alpha_m) - cos(n\alpha_{m-1})$$

$$+(-1)^m(cos(n\alpha_m) - cos(\frac{n\pi}{2})]$$

Donc :

$$a_n = \frac{4}{n\pi}[1 + 2\sum_{k=1}^{m}(-1)^k cos(n\alpha_k)] \qquad n = 1, 3, 5, ... \tag{2.4}$$

On considère une alimentation unité,i.e E_d/2=1.

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

$$V_{ao}(t) = f(wt) = \sum_{n=1}^{+\infty} a_n sin(n\omega t)$$

Le système d'équation 2.4 possède m variables inconnues $\alpha_1, \alpha_2, ..., \alpha_m$ 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.

2.3 calcul des valeurs exactes des angles de commutation par la méthode de Newton-Raphson

2.3.1 Description

La relation 2.4 est un système de m équations non linéaires à m inconnues $\alpha_1, ..., \alpha_m$. On assigne une valeur déterminée r, appelée taux 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.

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

Le système 2.4 s'écrit encore :

$$\begin{aligned}
a_1 &= \frac{4}{\pi}[1 + 2\sum_{k=1}^{m}(-1)^k cos(\alpha_k)] = -r \\
a_5 &= \frac{4}{5\pi}[1 + 2\sum_{k=1}^{m}(-1)^k cos(5\alpha_k)] = 0 \\
a_7 &= \frac{4}{7\pi}[1 + 2\sum_{k=1}^{m}(-1)^k cos(7\alpha_k)] = 0 \\
&\vdots \\
&\vdots \\
a_n &= \frac{4}{n\pi}[1 + 2\sum_{k=1}^{m}(-1)^k cos(n\alpha_k)] = 0
\end{aligned} \tag{2.5}$$

Ces amplitudes sont normalisées i.e la tension d'alimentation continue est supposée égale à l'unité. On doit signaler que la valeur du taux de modulation r assignée au fondamental est un indice sans

dimension. Pour obtenir la valeur correspondante en volt, il faut multiplier r par $E_d/2$, 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 r. C'est pourquoi on assigne une valeur négative $-r$ au fondamental[28]. Ce qui correspond à un déphasage de π du fondamental.
En résumé, on a un système de forme générale :

$$\begin{aligned}
f_1(\alpha_1,...,\alpha_m) &= \frac{4}{\pi}[1+2\sum_{k=1}^{m}(-1)^k cos(\alpha_k)] + r = 0 \\
f_2(\alpha_1,...,\alpha_m) &= \frac{4}{5\pi}[1+2\sum_{k=1}^{m}(-1)^k cos(5\alpha_k)] = 0 \\
f_3(\alpha_1,...,\alpha_m) &= \frac{4}{7\pi}[1+2\sum_{k=1}^{m}(-1)^k cos(7\alpha_k)] = 0 \\
&\vdots \\
&\vdots \\
f_m(\alpha_1,...,\alpha_m) &= \frac{4}{n\pi}[1+2\sum_{k=1}^{m}(-1)^k cos(n\alpha_k)] = 0
\end{aligned} \tag{2.6}$$

2.3.2 Résolution du système non linéaire par la méthode de Newton-Raphson

Notons : $\alpha^* = (\alpha_1^*, \alpha_2^*, ..., \alpha_m^*)$ le vecteur solution du système non linéaire 2.6[28] :

$f_i(\alpha) = 0$ avec $i = 1 : m$ et : α=(α_1,...,α_m).

Si chaque fonction f_i est continue et continûment différentiable, alors on peut la développer en série de Taylor d'ordre 1 dans le voisinage d'un estimé $\alpha^{(k)}$ (obtenu à la $k$ième itération) proche de α^*.

On obtient :

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

$$= f_i(\alpha^{(k)}) + \sum_{j=1}^{m} [\frac{\partial f_i(\alpha)}{\partial \alpha_j}]_{\alpha=\alpha^{(k)}} . (\alpha_j^* - \alpha_j^{(k)}) = 0$$

Le système s'écrit donc :

$$f_i(\alpha^{(k)}) + \sum_{j=1}^{m} [\frac{\partial f_i(\alpha)}{\partial \alpha_j}]_{\alpha=\alpha^{(k)}} . (\alpha_j^* - \alpha_j^{(k)}) = -f_i(\alpha^*) \tag{2.7}$$

Définissons la matrice des dérivées premières : $E^{(k)} = (E_{ij}^{(k)})$

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

D'où :

$$E^{(k)} = \frac{8}{\pi} \begin{pmatrix} sin(\alpha_1) & -sin(\alpha_2) & \cdots & -(-1)^m sin(\alpha_m) \\ 5sin(5\alpha_1) & -5sin(5\alpha_2) & \cdots & -5(-1)^m sin(5\alpha_m) \\ \vdots & \vdots & \ddots & \vdots \\ nsin(n\alpha_1) & -nsin(n\alpha_2) & \cdots & -n(-1)^m sin(n\alpha_m) \end{pmatrix}$$

Définissons le vecteur d'erreur : $\triangle\alpha^{(k)} = [\triangle\alpha_1^{(k)}, \triangle\alpha_2^{(k)}, ..., \triangle\alpha_m^{(k)}]^t$ avec : $\triangle\alpha_j^{(k)} = \alpha_j^* - \alpha_j^{(k)}$.

Soit le vecteur : $F^{(k)} = [F_1^{(k)}, F_2^{(k)}, ..., F_m^{(k)}] \quad$ avec : $F_i^{(k)} = -f_i(\alpha^{(k)})$

Alors le système 2.7 s'écrit sous la forme matricielle suivante :

$$E^{(k)} . \triangle\alpha^{(k)} = F^{(k)} \tag{2.8}$$

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

Le système 2.8 est un système linéaire de Cramer que l'on peut résoudre par l'algorithme de Gauss. Une fois le vecteur $\triangle\alpha^{(k)}$ déterminé, on obtient un meilleur estimé $\alpha^{(k+1)}$ de α^* par la relation : $\alpha^{(k+1)} = \alpha^{(k)} + \triangle\alpha^{(k)}$.

On continue jusqu'à ce que : $| \alpha^* - \alpha^{(k)} | \to 0$.

Les figures 2.5, 2.6 et 2.7 donnent, à titre d'exemple, les graphes des angles de commutation exacts calculés pour m égal à 3 ,5 et 7.

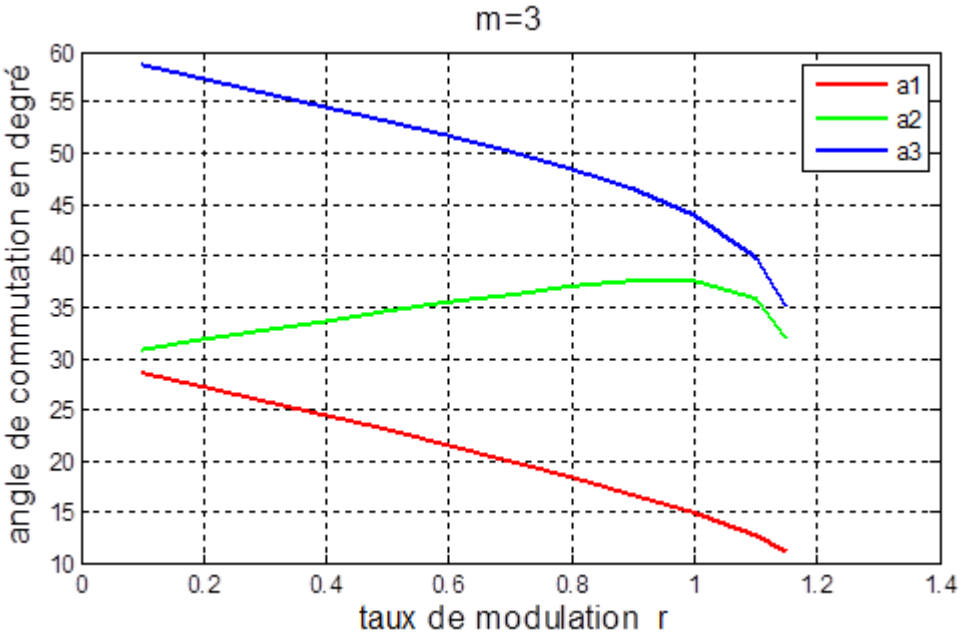


FIGURE 2.5 – Courbes des angles de commutation exacts pour m égal à 3.

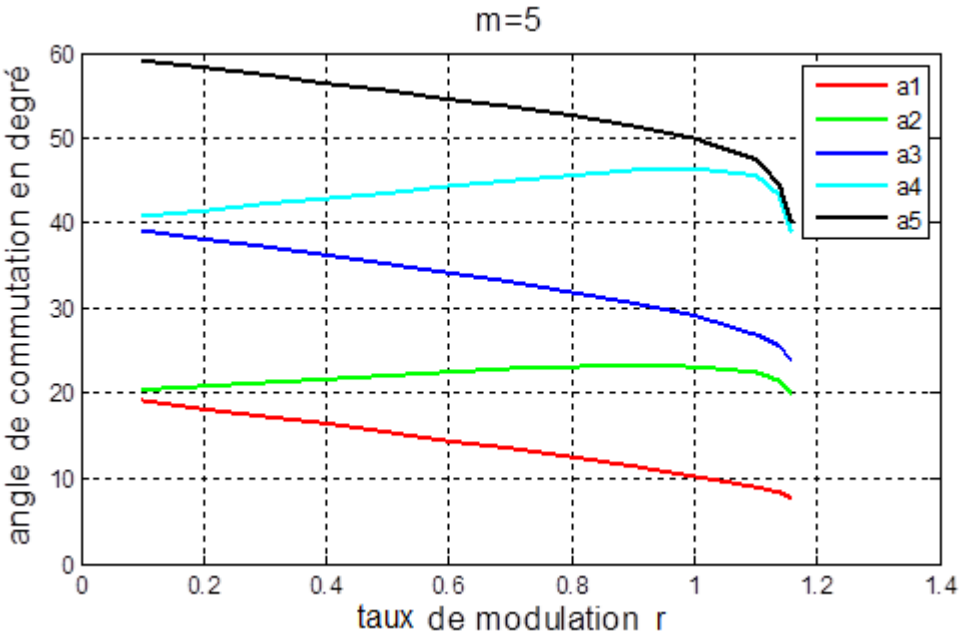


FIGURE 2.6 – Courbes des angles de commutation exacts pour m égal à 5.

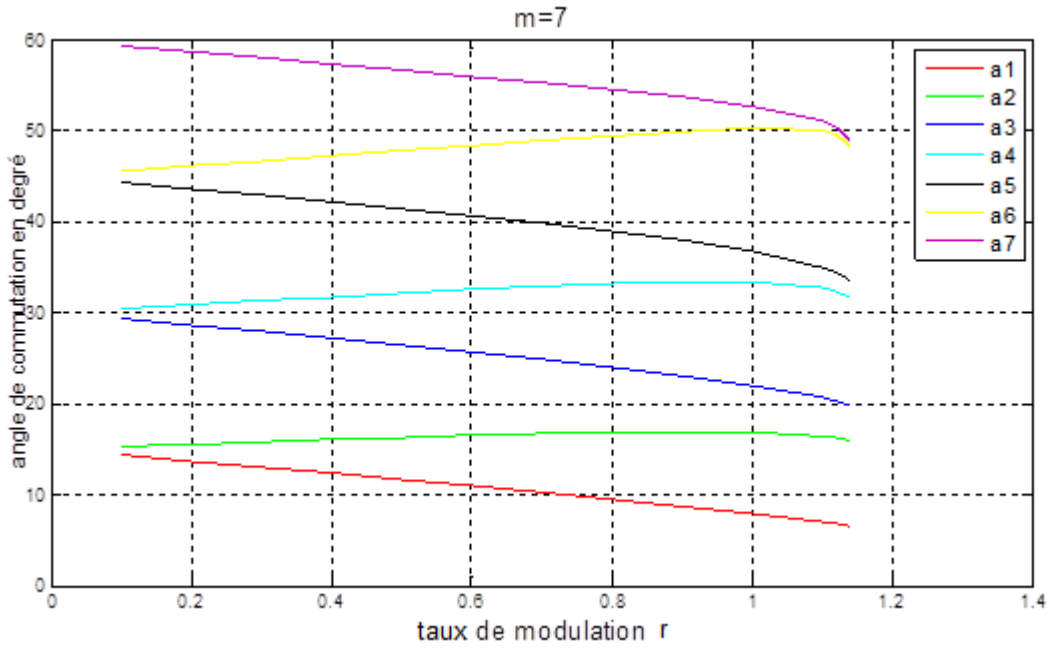


FIGURE 2.7 – Courbes des angles de commutation exacts pour m égal à 7.

2.4 Description de l'algorithme 'on-line'

Les équations utilisées pour calculer les angles de commutation sont non linéaires et transcendantes, donc leur solution ne peut pas être faite on-line. Ces angles sont calculés off-line par des méthodes numériques bien connues, ensuite stockés dans des mémoires ; mais avec une large gamme de variation du taux de modulation r et l'interpolation requise demande une énorme puissance de calcul et rend le système plus coûteux.

Ce problème a poussé les chercheurs d'élaborer des algorithmes permettant un calcul on-line de ces angles. Dans les paragraphes qui suivent on va élaborer un algorithme MLI calculée on-line, c'est un algorithme basé sur la méthode des moindres carrées.

2.4.1 Approximation des angles exacts

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.

D'après les figures 2.5, 2.6 et 2.7 et Pour un taux de modulation r égal à zéro, l'angle α_s , appelée angle de séparation, est donné par la formule approchée[14] :

$$\alpha_s = \frac{2 \times 60^o}{m+1} \qquad \text{avec m impair}$$

Pour m fixe, 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] du taux de modulation r, 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 du taux de modulation r 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 du taux r 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 donne une forme générale des angle de commutation en fonction de k, m, r et $\Delta\alpha_k$. Dans une seconde étape, on cherche à approximer $\Delta\alpha_k$ en fonction de k et m.

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

$$\phi(x) = A_0\phi_0(x) + ... + A_p\phi_p(x) \tag{2.9}$$

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 $\phi_0, \phi_1, ..., \phi_p$ sont choisies à l'avance[28].

2.4.2 Cas k impair

1. **Première étape d'approximation de α_k :** Soit α_k la valeur exacte de l'angle de commutation et α_s l'angle de séparation pour m et k données et pour r fixe, on définit [14] :

$$\Delta\alpha_k = \frac{60^o(k+1)}{m+1} - \alpha_k \tag{2.10}$$

La variation normalisée de $\Delta\alpha_k$ noté Δ_k est donnée par :

$$\Delta_k = \frac{\Delta\alpha_k}{\alpha_s} = \frac{(m+1)\Delta\alpha_k}{2\times 60^o}$$

On remplace dans l'équation 2.10 :

$$\alpha_k = \frac{60^o(k+1)}{m+1} - \frac{2\times 60^o}{m+1}\Delta_k$$

Par conséquent on cherche d'approximer α_k par une fonction linéaire de la forme :

$$\phi_{k,m}(r) = A_0 + A_1 r$$

Avec : $A_0 = \frac{60^o(k+1)}{m+1}$
$A_1 = C_0\frac{2\times 60^o}{m+1}\Delta_k$ Avec C_0 paramètre à définir

Pour chaque valeur de m et de k, on fait varier le taux de modulation r de 0.1 à 1.1 par pas de 0.1. Pour chaque valeur du taux r, on calcul la déférence. Le coefficient C_0 est pratiquement constant et égal à -1.25.
Donc :

$$\phi_{k,m}(r) = \frac{60^o(k+1)}{m+1} - [\frac{2\times 60^o}{m+1}\times\frac{\Delta_k\times r}{0.8}]$$

2. **Deuxième étape d'approximation de α_k :** Après l'approximation linéaire de l'angle 'exact' de commutation, dans la première étape d'approximation, on va effectuer maintenant l'approximation de la pente Δ_k de la fonction $\phi(k,r)$ en fonction de k et m.

Pour étudier la forme de Δ_k, on va calculer Δ_k pour des valeur de m et k variant par valeurs impaires et La figure 2.8 représente la fonction $\Delta_k(k,m)$.

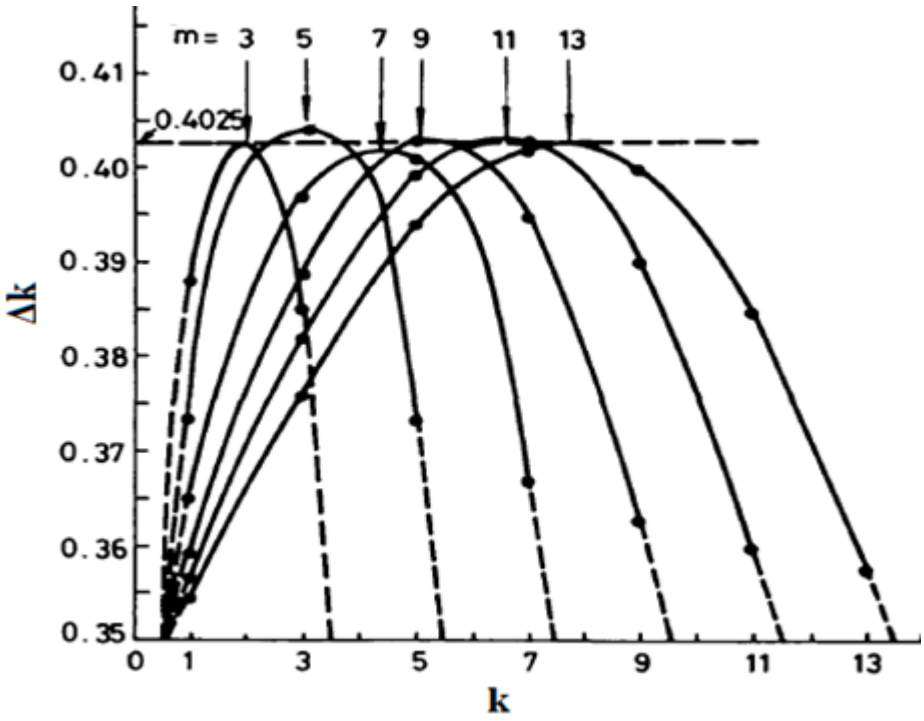


FIGURE 2.8 – variation de Δ_k pour k impair[14]

Cependant, en observant ces courbes on constate qu'elles ont chacune une forme non linéaire parabolique. Elles diminuent en fonction de m pour même k et passent par un maximum fixe quelque soit m et en remarque que $\Delta_k(k,m)$ est pratiquement égale à $\Delta_k(m+1-k,m)$ c'est-à-dire elles présentent une symétrie par rapport à $\frac{(m+1)}{2}$.

Le problème maintenant est de rechercher comment effectuer l'approximation des courbes de la figure 2.8 à l'aide de fonctions simples et avec une plus grande précision.
Pour répondre à cette question, considérons la fonction suivante[15] :

$$\phi_2(k,m) = \frac{C_1}{m^2} \times (k - \frac{m+1}{2})^2 + C_2$$

On fait varier k (impair) pour chaque m et on calcul les paramètres C_1 et C_2. Les paramètres précédents sont choisis en minimisant le critère quadratique des moindres carrés. On trouve : $C_1 = -0.21, C_2 = 0.4025$.

Finalement la formule approximée des angles de commutation pour k impair est donnée par :

$$\hat{\alpha}_k = \frac{60^o(k+1)}{m+1} - [\frac{2 \times 60^o}{m+1}\frac{\hat{\Delta}_k \times r}{0.8}] \tag{2.11}$$

$$\hat{\Delta}_k = \frac{-0.21}{m^2} \times (k - \frac{m+1}{2})^2 + 0.4025$$

2.4.3 Cas k pair

Pour le cas pair on suive le même raisonnement, la forme de l'angle de commutations a la même forme que le cas impair avec une pente positive :

$$\hat{\alpha}_k = \frac{60^o k}{m+1} + [\frac{2 \times 60^o}{m+1}\frac{\Delta_k \times r}{0.8}]$$

Pour étudier la forme de Δ_k pour le cas pair, on va calculer Δ_k pour des valeur de m et k variant par valeurs paires et La figure 2.9 représente la fonction $\Delta_k(k, m)$.

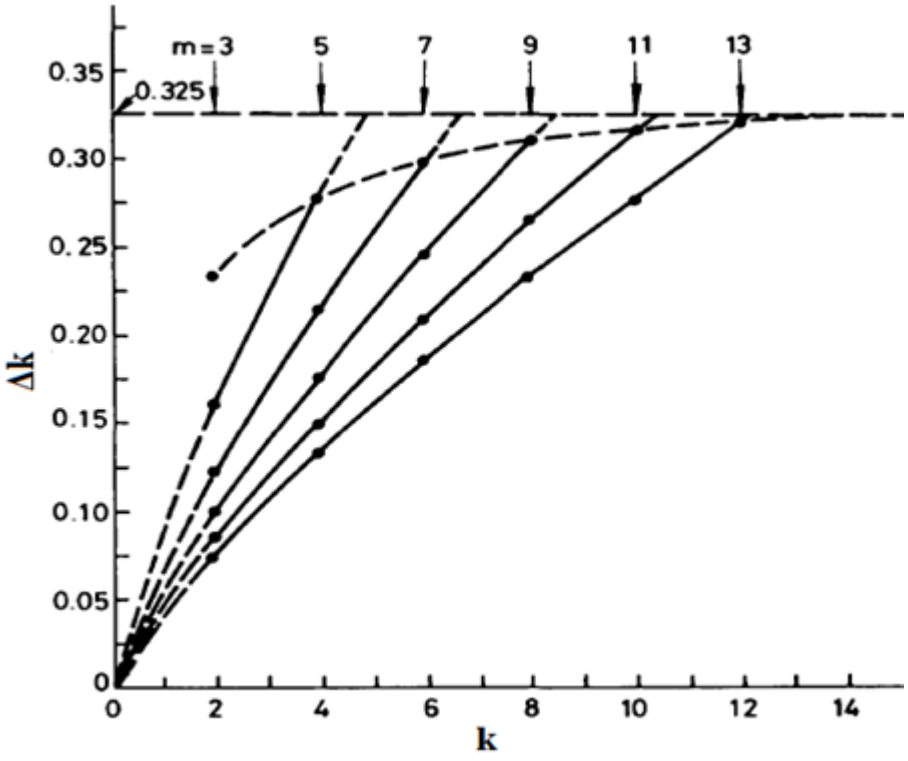


FIGURE 2.9 – variation de Δ_k pour k pair[14]

On observe que toutes les courbes $\Delta_k(k,m)$ passer par l'origine et croissant avec k, et diminuant avec m pour même k.

Pour l'approximation des courbes de la figure 2.9 on pose la fonction paramétrique suivante [15] :

$$\phi_3(k,m) = \frac{C_4}{(m-1)^2} \times (k - C_5\frac{m-1}{2})^2 + C_6 - \frac{k}{m^3}$$

Pour déterminer les paramètres C_4, C_5 et C_6 on fait varier k par valeur pair pour chaque m. Ces paramètres sont déterminés en minimisant le critère quadratique des moindres carrées. on trouve : $C_4 = -0.082, C_5 = 2.482$ et $C_6 = 0.505$

Finalement la formule approximée pour les angles de commutation pour k pair est donnée par :

$$\hat{\alpha}_k = \frac{60^o k}{m+1} + [\frac{2 \times 60^o}{m+1}\frac{\hat{\Delta_k} \times r}{0.8}] \tag{2.12}$$

$$\hat{\Delta_k} = \frac{-0.082}{(m-1)^2} \times (k - 2.482\frac{m-1}{2})^2 + 0.505 - \frac{k}{m^3}$$

2.5 Précision de l'algorithme

Les figures 2.10 et 2.11 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 7, dans tout l'intervalle de variation [0, 1] du taux r. Les figures montrent que l'erreur moyenne est de l'ordre de dix centièmes $\frac{10}{100}$ de degré pour m égal à 5 et qu'elle diminue lorsque m augmente et atteint huit centième $\frac{8}{100}$ de degré pour m égal à 7. Il faut signaler que l'erreur moyenne est calculée sur tout l'intervalle [0 ;0.8] de variation du taux r et qu'elle est beaucoup plus faible pour de faibles valeurs de r.

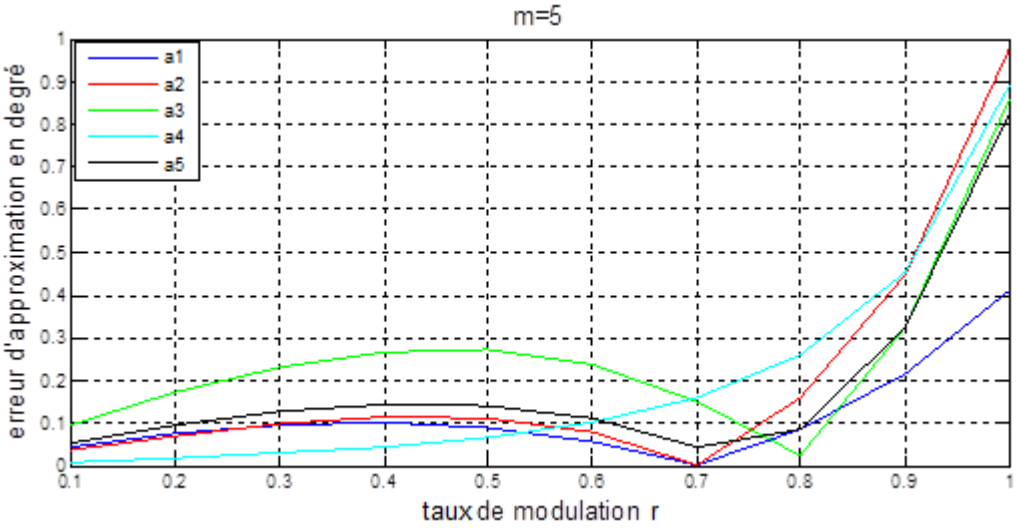


FIGURE 2.10 – erreur d'approximation pour $m = 5$

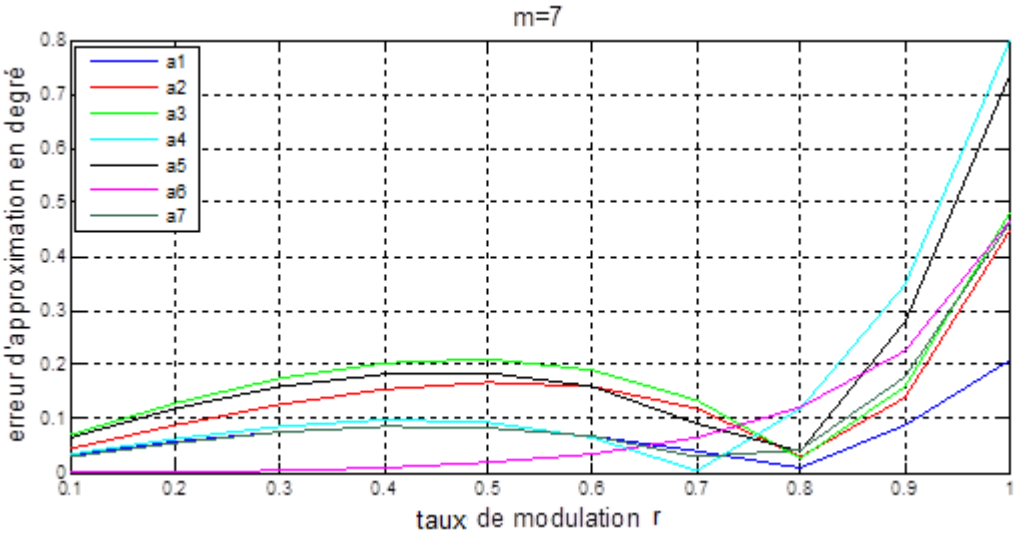


FIGURE 2.11 – erreur d'approximation pour $m = 7$

Le modèle proposé est précis pour une plage de variation de r de 0 à 0.8. Pour les valeurs plus grandes, il faut ajouter au modèle des termes correctifs[15].

2.6 Conclusion

Ce chapitre a été consacré pour la description des deux techniques MLI engendrée et à élimination des harmoniques. Pour la technique MLI calculée, nous avons développé un algorithme on-line qu'on va implémenter ensuite sur la carte FPGA.

Dans le prochain chapitre, on va voir les circuits FPGA. On va décrire l'architecture interne et les types ainsi la configuration de ces circuits.

Chapitre 3

Les circuits FPGA

3.1 Introduction

Les composants logiques programmables sont traditionnellement regroupés au sein de deux familles : les CPLD et les FPGA. Les premiers sont des PLD (Programmable Logic Device) complexes, car composés de plusieurs macro cellules programmables simples, réparties autour d'une matrice d'interconnexion.

Les FPGA (Field Programmable Gate Array) présentent une mer de modules logiques de petite taille, noyés dans un canevas de routage. Les CPLD (Complex Programmable Logic Device) sont composés de blocs de logique combinatoire et de bascules. La confusion ayant pu régner par le passé entre des CPLD et des FPGA, c'est à présent le mode d'interconnexion qui tend à déterminer le type de PLD. Cette distinction est d'ailleurs la plus significative pour l'utilisateur. En effet, les temps de propagation étant plus prévisibles pour les CPLD du fait de leur schéma d'interconnexion, c'est cette propriété qui permettra de faire un choix quant au type de circuit logique à privilégier pour une application donnée[11].

Parmi les autres types de composants programmables, les matrices d'interconnexion destinées à aiguiller des signaux de natures diverses ne seront pas traitées, de même que les SPLD ou PAL de taille réduite. Les composants exotiques tels que les PSD (Programmable system devices) de STMicroelectronics, qui intègrent quelques macro cellules dans un microcontrôleur ou bien une puce mémoire système configurable, n'offrent pas non plus de critère de comparaison. Un aspect

incontournable de la logique programmable concerne les outils de développements, bien plus abordables que ceux destinés à la conception des ASICs. N'importe quel intégrateur peut répondre au besoin d'un composant logique spécifique pour un coût supportable. L'ajout de fonctions standard, implantées rapidement au sein du PLD, est également facilité l'aide des bibliothèques de propriété intellectuelle largement diffusées.

3.2 Critères de performances

Dans le monde des circuits numériques les chiffres évoluent très vite. Cette impression de mouvement nous oblige de définir des critères de choix pour les circuits programmables.

3.2.1 Puissance de calcul

Elle dépend de nombre des portes équivalentes, architecture des cellules et le nombre d'entrées sorties.

3.2.2 Vitesse de fonctionnement

Les comportements dynamiques des circuits programmables présentent des différences marquantes. Une analyse et une simulation post synthèse, qui prend en compte les paramètres dynamiques des cellules, permet réellement de prévoir les limites de fonctionnement d'un circuit.

3.2.3 Capacité de mémoire

Les circuits programmables contiennent des mémoires pour stocker leur configuration. La plupart des familles récentes offrent à l'utilisateur la possibilité d'utiliser certaines de ces mémoires en tant que telles. Les capacités de mémorisations atteignent quelques dizaines de kilo bits.

3.2.4 Routabilité

Certains circuits garantissent une routabilité complète : toute fonction intégrable dans le circuit pourra être modifiée sans modification du câblage externe. Le routage influe sur les performances dynamiques de la fonction finale[12].

3.3 Les circuits FPGA

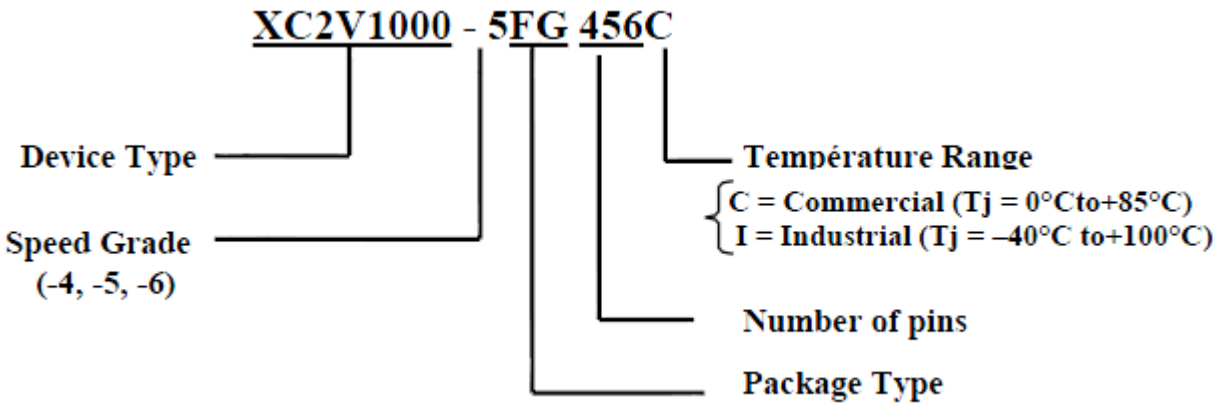
3.3.1 Définition

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 [10].

3.3.2 Nomenclature des circuits FPGA

Les circuits FPGA suivent la nomenclature suivante, selon un exemple donné[25] :



3.3.3 Domaines d'application des circuits FPGA

Les utilisations sont nombreuses, on en cite :

– Prototypage de nouveaux circuits (nouveaux microprocesseurs pour ordinateur, par exemple).

- Fabrication de composants spéciaux en petite série.
- Dans les systèmes embarqués de commande temps réel : l'avionique ; Chaque carte de commande de vol d'un Airbus emploie plusieurs FPGA 'Actel'. Les routeurs de réseau informatique emploient plusieurs FPGA ; les systèmes de l'informatique industrielle aussi.
- Adaptation aux besoins rencontrés lors de l'utilisation (par exemple dans des satellites ou des sondes spatiales) [25].

3.4 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.

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 dont 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 :

1. Une couche appelée circuit configurable.
2. Une couche réseau mémoire SRAM.

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 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 par le biais de l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ 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[23].

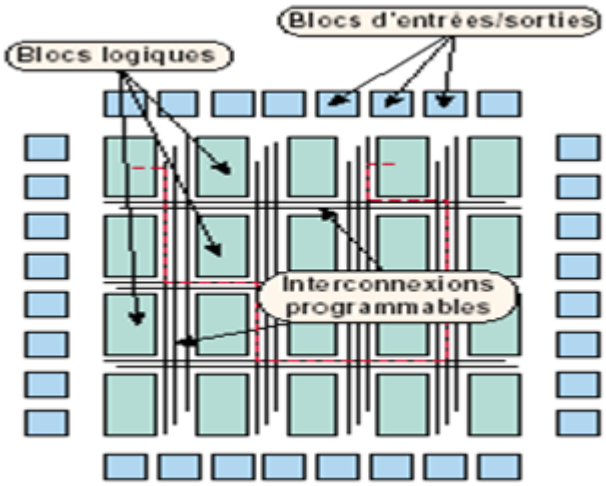


FIGURE 3.1 – Architecture interne d'un FPGA[23]

Les circuits FPGA du fabricant Xilinx utilisent deux types de cellules de base :

- les cellules d'entrées/sorties appelés IOB (input output bloc).
- les cellules logiques appelées CLB (configurable logic bloc).

Ces différentes cellules sont reliées entre elles par un réseau d'interconnexions configurable.

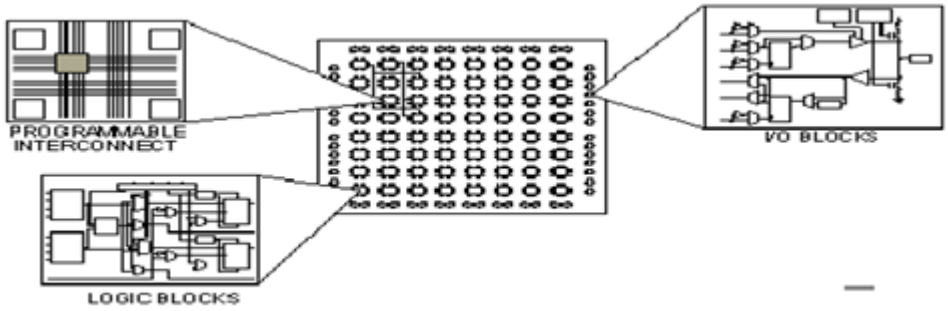


FIGURE 3.2 – réseaux d'interconnexions configurable[27]

3.4.1 Les CLB (configurable logic bloc)

Les blocs logiques configurables sont les éléments déterminants les 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 figure ci-dessous, nous montre le schéma d'un CLB[27].

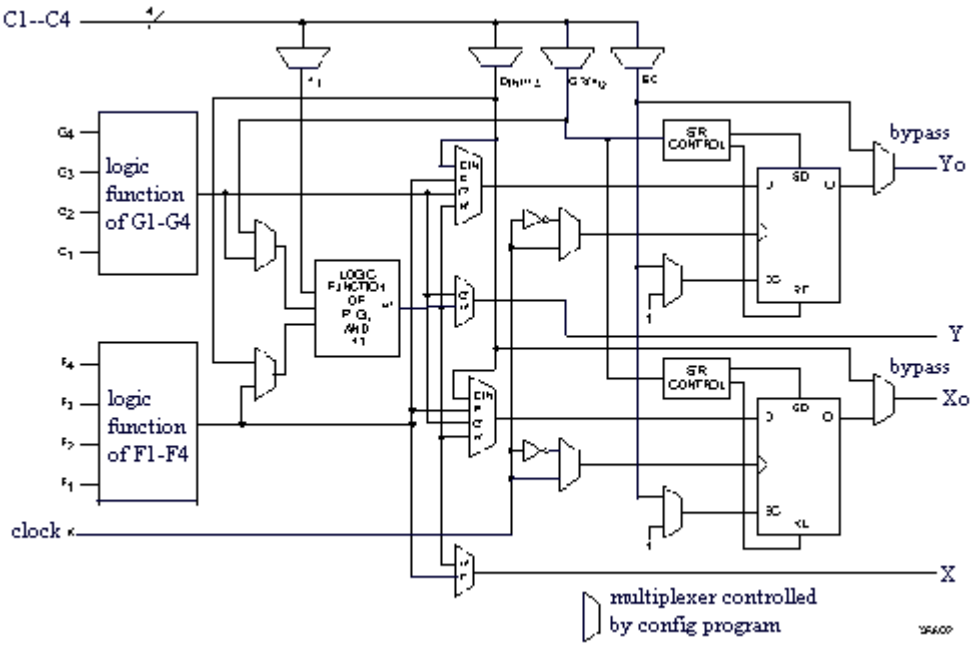


FIGURE 3.3 – Cellules logiques(CLB)[27]

D'abord le bloc logique combinatoire qui possède deux générateurs de fonctions F' et G' à quatre entrées indépendantes (F1...F4, G1...G4), lesquelles offrent aux concepteurs une flexibilité de développement importante car la majorité des fonctions aléatoires à concevoir n'excède pas quatre variables. Les deux fonctions sont générées à partir d'une table de vérité câblée inscrite dans une zone mémoire, rendant ainsi les délais de propagation pour chaque générateur de fonction indépendants de celle à réaliser. Une troisième fonction H' est réalisée à partir des sorties F' et G' et d'une troisième variable d'entrée H1 sortant d'un bloc composé de quatre signaux de contrôle H1, Din, S/R, Ec. Les signaux des générateurs de fonctions 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 [23] :

1. Deux fonctions indépendantes à quatre entrées indépendantes.
2. Ou une seule fonction à cinq variables.
3. Ou deux fonctions,une à quatre variables et une autre à cinq variables.

L'intégration de fonctions à nombreuses variables diminue le nombre de CLB nécessaires, les délais de propagation des signaux et par conséquent augmente la densité et la vitesse du circuit.

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 XQ et YQ. Un mode dit de " verrouillage " exploite une 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 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.

Un mode optionnel des CLB est la configuration en mémoire RAM de 16×2bits ou 32×1bit. 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 D0, D1 (RAM 16×2bits) d'entrée et le signal de validation d'écriture WE. Le contenu de la cellule mémoire (D0 et D1) 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.

3.4.2 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).

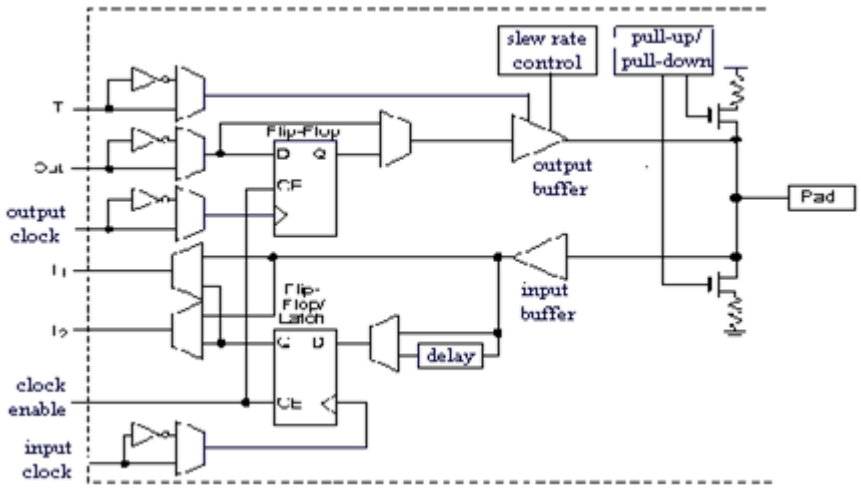


FIGURE 3.4 – Input Output Block (IOB)[10]

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

1. Seuil d'entrée TTL ou CMOS.
2. Slew-rate programmable.
3. Buffer de sortie programmable en haute impédance.
4. Entrées et sorties directes ou mémorisées.
5. Inverseur programmable

3.4.3 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 ns 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 [10].

3.4.4 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.

3.4.5 Les interconnexions[25]

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. **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.

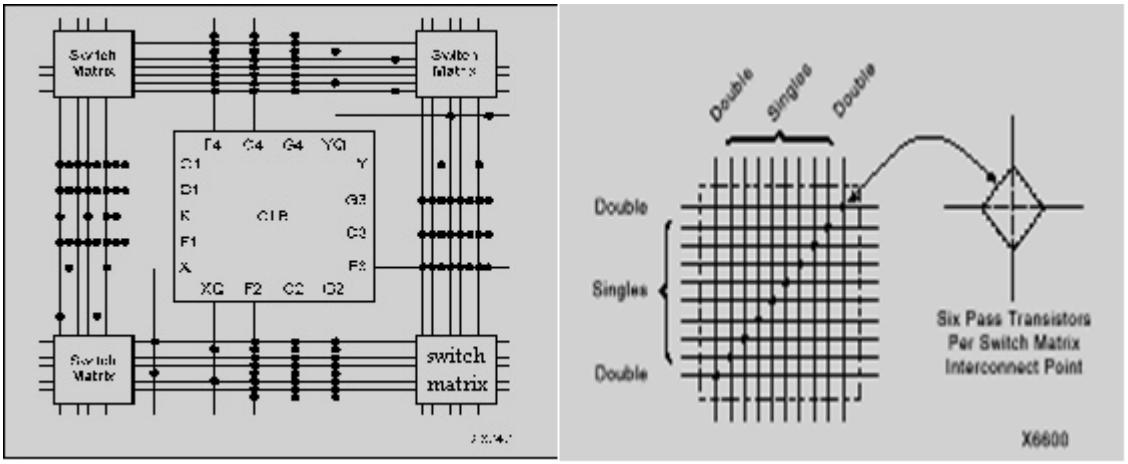


FIGURE 3.5 – Connexions à usage général et matrice de commutation[25]

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.

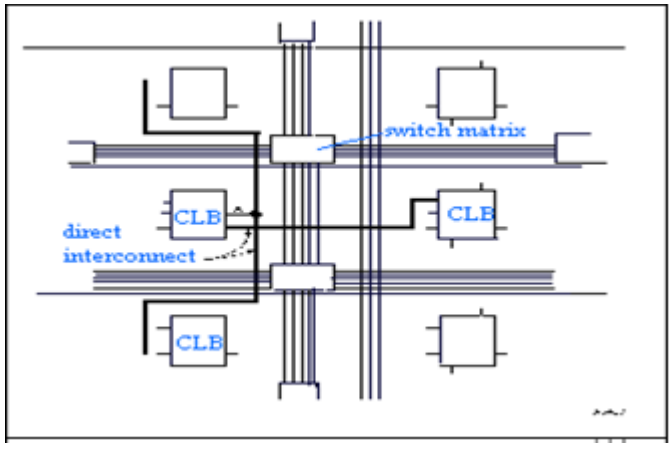


FIGURE 3.6 – Les interconnexions directes[25]

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.

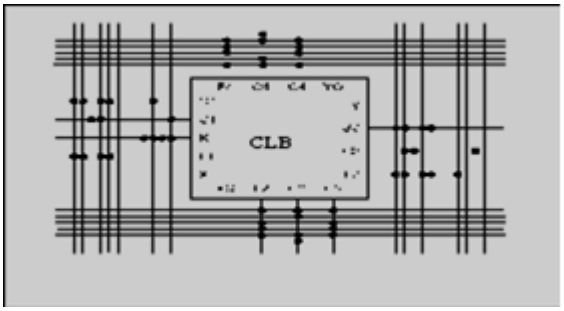


FIGURE 3.7 – Les longues lignes[25]

3.5 Configuration des FPGA

Les circuits FPGA se configurent selon plusieurs modes à titre d'exemple on cite les plus utilisés par XILINX [23] :

1. Le mode série qui utilise une EEPROM série caractérisée par sa capacité considérable et faible rapidité.
2. Le mode parallèle 8 bits utilisant une EPROM classique qui est plus rapide.
3. Le mode parallèle asynchrone : le circuit FPGA se comporte comme un périphérique de microprocesseur ce qui lui offre une possibilité de reconfiguration partielle intelligente.

Dans chaque mode, plusieurs FPGA peuvent être chaînés en mode maître/esclave.

3.6 Types des FPGA

Il existe plusieurs types de FPGA selon le mode de programmation choisi : anti-fusible, EPROM et SRAM[23].

- La technologie anti-fusible permet une programmation unique du circuit.
- La technologie EPROM permet de programmer plusieurs fois le circuit mais nécessite l'application de rayons UV sur la puce.

– La technologie SRAM offre la possibilité de programmer plusieurs fois le circuit sans l'utilisation d'artifices extérieurs. On parle alors de FPGASRAM (Symétrique). Dans cette technologie, les éléments configurables sont connectés à des mémoires SRAM (static-RAM) de taille d'un bit. Le circuit est programmé par écriture d'un 0 ou d'un 1 dans chaque SRAM. Un FPGA est ainsi constitué de plusieurs milliers de mémoires SRAM connectées aux blocs logiques, aux blocs de routage local et aux blocs de routage global. Ainsi, un interrupteur programmable est configuré en position fermée (ouverte) par l'écriture d'un 1(0) dans la SRAM correspondante.

Une LUT (Lock Up Table) est un bloc combinatoire programmable composé de n entrées E0...En-1, d'une sortie Z et d'un tableau vertical de 2n cellules SRAM connectées à un multiplexeur 2n vers 1. La LUT est capable de réaliser toutes les fonctions possibles à n entrées. Pour cela, chacune de ses cellules SRAM est programmée selon la table de vérité de la fonction donnée.

3.7 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.

3.8 Méthodologie de développement d'un projet sur FPGA[28][25]

Notre application a été réalisée en utilisant le langage VHDL. C'est un langage de description hardware de haut-niveau, qui permet de matérialiser les structures électroniques d'un circuit.

En effet, les instructions écrites dans ce langage se traduisent par une configuration logique faite de portes et de bascules, qui est ensuite intégrée à l'intérieur du circuit cible.

La description VHDL se fait d'une manière hiérarchique, où chaque module peut soit être décrit complètement, soit faire appel à des sous-modules déjà décrits. Un code écrit en VHDL est por-

table, c'est-à-dire qu'une description écrite pour un circuit peut être utilisée pour un autre. Mais pour obtenir un design avec des performances optimales, il vaut mieux décrire le comportement des algorithmes souhaités en tenant compte de la technologie et de l'architecture du circuit FPGA cible (le XC2V1000-4FG456C dans notre cas).
La réalisation d'une application sur circuit FPGA impose une méthodologie de développement structurée.

La première étape est la description des algorithmes en VHDL, une vérification des erreurs s'impose alors. Ensuite c'est l'étape de synthèse, dans laquelle les lignes de code VHDL sont traduites en schémas électroniques équivalents composés de fonctions logiques de base. L'étape suivante est le placement et routage, où les schémas électroniques sont transformés en routes et connexions sur le FPGA cible. Enfin, c'est la génération des fichiers binaires de programmation et leur chargement sur le FPGA cible à travers le port JTAG.

Il est possible de disposer de fichiers VHDL après 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. Cependant, c'est la simulation comportementale qui est la plus souvent utilisée.

Après le chargement du design sur le FPGA, on doit réaliser des tests en laboratoire en appliquant des contraintes réelles, afin de valider le bon fonctionnement du dispositif dans la pratique.

3.8.1 Logiciel et outil utilisé

Chaque étape du développement d'un projet sur FPGA nécessite un outil logiciel différent. Nous allons présenter brièvement l'outil et logiciel utilisés dans ce projet qui est Xilinx ISE 9.2i. Ce logiciel offre un environnement convivial et regroupe tous les outils nécessaires à la simulation et la réalisation des différentes étapes du projet. Le code VHDL des différents modules est saisi dans un éditeur de texte. L'étape de synthèse est réalisée par l'outil XST, l'étape de placement et routage par l'outil FPGA Editor et l'étape de configuration du composant par l'outil iMPCAT. Xilinx ISE permet ainsi de suivre l'évolution du projet étape par étape.

3.8.2 Implémentation d'un projet sur circuit FPGA

L'implémentation d'un projet sur le circuit FPGA suit ces étapes :

1. **Description VHDL :**c'est la modélisation de l'architecture et l'écriture des différents codes VHDL des différents composants constituant l'architecture ;
2. **Compilation syntaxique :**vérifie l'existence d'erreurs dans les instructions du code VHDL ;
3. **Compilation logique :**vérifie la faisabilité décrite dans les programmes ;
4. **Simulation fonctionnelle :**permet de tester le bon fonctionnement de l'architecture moyennant l'outil de simulation Xilinx ISE 9.2i.
5. **Phase de synthèse :**permet de passer d'un niveau d'abstraction élevé 'VHDL' vers un niveau d'abstraction plus bas qui est le niveau (RTL : Registre Translate Level) adapté au langage machine. Le résultat de la synthèse est un fichier dont l'extention est **'.edif'**.
6. **Phase de placement routage :**elle passe par 3 étapes :
   - **Translation :**permet de convertir le fichier **'.edif'** vers un fichier d'extention **'.NGD'** dont il est comprhénsible par l'outil de placement routage ;
   - **Mapping :**planification de l'implémentation des différentes parties de l'architecture vis-à-vis des ressources des circuits FPGA que nous ciblons dans notre application ;
   - **Placement et routage :**il s'agit d'établir les différentes interconnexions entre les composants integrés dans le circuit.
7. **Génération du fichier '. bit' :**après le placement routage, l'outil nous génére un fichier dont l'extention '.bit' est un fichier binaire, téléchargeable, qui permet de configurer un 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 base, 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.

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

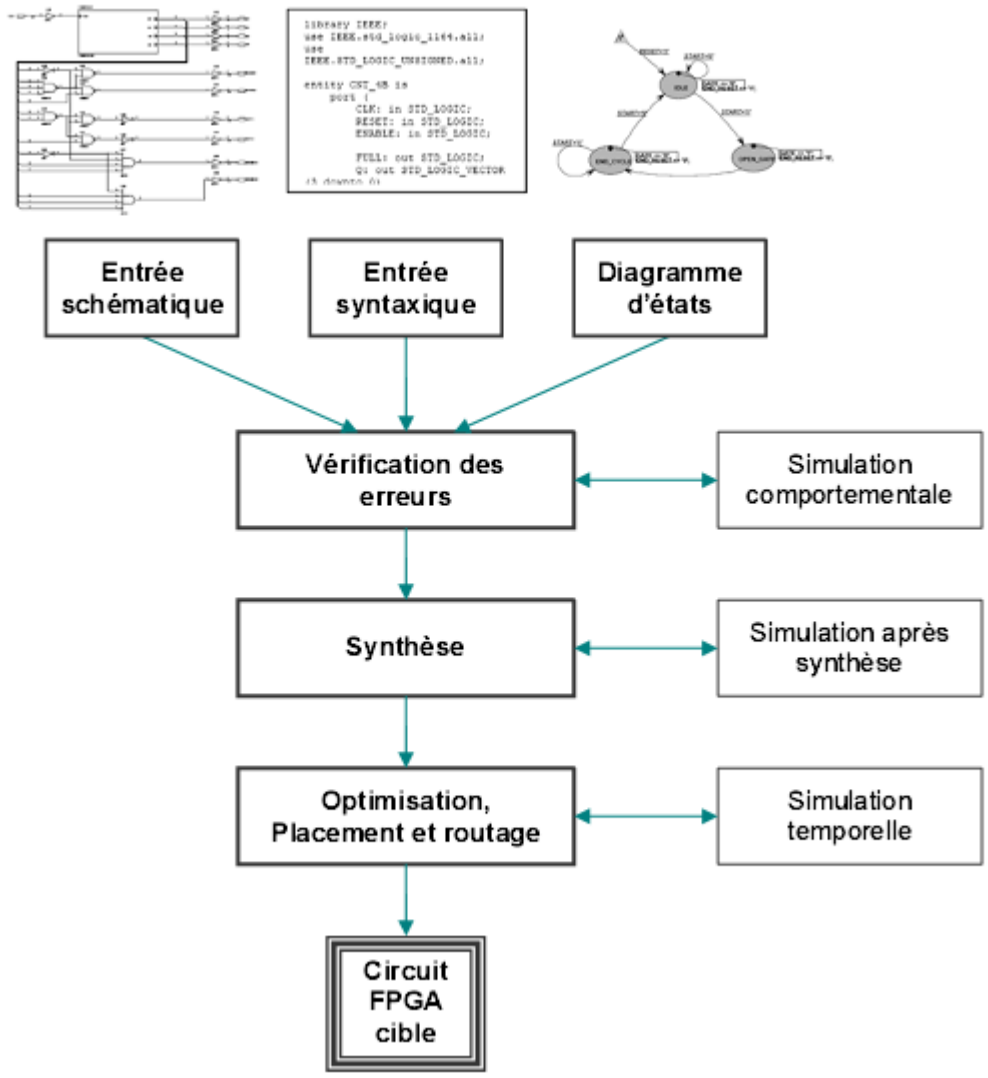


FIGURE 3.8 – Organisation fonctionnelle de développement d'un projet sur circuit FPGA[25].

3.9 Conclusion

Ce chapitre a été consacré à l'étude des circuits programmables FPGA. On a expliqué les différentes étapes de développement d'une implémentation d'un projet autour d'un circuit FPGA.

Le prochain chapitre sera consacré à la description du langage VHDL utilisé pour la programmation de ces circuits. Nous allons voir aussi quelque notions sur le saisie d'un code VHDL et les instructions les plus utilisées.

Chapitre 4

Le langage de description VHDL

4.1 Introduction

Depuis quelques années, la logique discrète disparait au profit de la logique intégrée et de l'utilisation de circuits programmables plus ou moins complexes pour réaliser les fonctions numériques. La méthodologie de conception a suivi et a aussi évolué :

- Quand on faisait de la logique câblée avec des circuits discrets, on simplifiait les équations à la main.
- Les premiers circuits programmables réalisaient des fonctions à partir d'équations que les outils simplifiaient automatiquement.
- Puis, pour les circuits complexes, on est passé à la saisie directe de logigrammes hiérarchiques utilisant des bibliothèques de fonctions complexes.
- Maintenant, on utilise des langages textuels de description de matériel *(Hardware Description Languages)*.

Il existe essentiellement deux langages : le VERILOG et le VHDL. Ces deux langages sont proches et il existe d'ailleurs des outils de traduction automatique pour passer de l'un à l'autre. Notons que ces langages permettent également de décrire les fichiers de simulation pour valider les circuits (*testbench* en anglais)[12].

On parlera ici uniquement du VHDL synthétisable, c'est-à-dire du VHDL destiné à la réalisation effective de circuit intégré et non limité à de la modélisation.

4.2 Caractéristiques de VHDL

VHDL est l'acronyme de *Very high speed integrated circuit Hardware Description Language*. C'est un langage qui a été normalisé IEEE en 1987 (il est appelé depuis VHDL-87), ce qui lui conféra une bonne portabilité dans les outils des divers fabricants. Il existe de nombreuses révisions (VHDL-93, VHDL-2000, VHDL-2002 ...), mais on se limitera à l'usage du VHDL-93.

Pour résumer, c'est un langage [2] :

- **portable** :il est normalisé, donc la syntaxe est standardisée dans les outils ; la simulation est la même quel que soit l'outil ;
- **hiérarchique** :il permet la réutilisation des sources ;
- **autonome** :il permet de simuler le circuit décrit sans utiliser d'autres langages pour décrire les stimuli. Il permet aussi de décrire des modèles.

4.3 Structure d'un code source VHDL

Un code source VHDL est constitué de trois parties : les bibliothèques, la description de l'entité, et la description de l'architecture.

4.3.1 Bibliothèques

En VHDL, on parle plutôt de package, le terme bibliothèque désignant l'ensemble des packages.

1. **Bibliothèque standard :** C'est la bibliothèque qui est inhérente au langage.Elle contient un package intéressant pour la manipulation des fichiers :le package **std_textio**.
2. **Bibliothèque normalisée IEEE :**Dans la bibliothèque IEEE, on trouve les packages suivants :
   - **std_logic_1164 :**il contient la description des types **std_logic** et **std_logic_vector** ;
   - **numeric_std :**il contient la définition des types **signed** et **unsigned**.
3. **Autres bibliothèques :**Il existe d'autres bibliothèques, qui ne sont pas normalisées, bien que couramment utilisées. Citons notamment le package **std_logic_arith**, créé par Synopsys, qui

est tellement courant que certains le considèrent à tort comme normalisé. Il vaut mieux ne pas mélanger des packages de diverses provenances car on risque d'avoir des erreurs de compilation et en synthèse. La bibliothèque IEEE permet de tout faire. Elle est supportée par tous les outils. Il est conseillé de n'utiliser que celle-ci pour éviter les problèmes [8].

4.3.2 Généralités sur la syntaxe

Un certain nombre de règles sont à respecter lors de l'écriture du code :

- Les noms ne doivent pas être des mots clés. Ils ne doivent pas commencer par un chiffre.
- Les instructions se terminent par un point-virgule ;
- Les commentaires commencent par – et se terminent en fin de ligne.
- Jamais de caractères accentués.
- Les bits simples sont écrits entre ' '*(simple quote)*.
- Les variables sur plusieurs bits sont écrites entre " " *(double quote)*.

4.3.3 Description d'entité

L'entité correspond à la description des liens avec l'extérieur du circuit. Elle comprend :

- Les paramètres génériques.
- Les entrées et les sorties.
- Les entrées/sorties dans le cas de signaux bidirectionnels.

On y précise également les types de ces liens.

Dans l'entité, on utilisera uniquement le type **std_logic** pour les données sur 1 bit, et le type **std_logic_vector** pour les données sur plusieurs bits. Les paramètres génériques permettent de définir des valeurs modifiables au plus haut niveau, sans modifier le code. Ils servent souvent à définir la dimension des bus, mais ce n'est pas limitatif.

On écrit souvent un code généralisé (donc facilement réutilisable) qu'on utilise avec une valeur spécifique. On peut modifier cette valeur sans toucher au code écrit en utilisant la hiérarchie.

Exemple :
```
entity zz is
generic (n : natural := 4) ; - - 4 valeur par défaut
port(
   en : in std_logic ; - - entrée sur un bit
   a,b : in std_logic_vector(n-1 downto 0) ; - - donnée en entrée
   s : out std_logic_vector(n-1 downto 0) ) ; - - et en sortie sur n bits
end entity ;
```

4.3.4 Généralités sur l'architecture

Elle se décompose en deux parties :

- Une zone de déclaration où on crée tous les signaux internes.
- Une zone de description où on décrit la logique du circuit.

Exemple :
```
architecture arch of zz is - - l'architecture arch est associée à l'entité zz
signal sig1 : std_logic ;
signal sig2 : std_logic_vector(n-1 downto 0) ; - - on utilise le générique défini dans l'entité
begin
- - zone de description de la logique
end architecture ;
```

4.3.5 Signaux et opérateurs

1. **signaux et constantes**

Les signaux sont les équipotentielles internes, les fils de liaison entre les diverses instructions de l'architecture. Ils doivent tous être nommés et typés [2]. Les constantes sont des signaux ayant une valeur fixe. On ne pourra donc que lire ces valeurs. On fixe le type et on écrit sa valeur précédée de :=.

Exemple :
```
signal s1 : std_logic_vector (7 downto 0) := x"FF" ;- - valeur initiale de simulation.
constant c1 : std_logic_vector (3 downto 0) := x"6" ;- - constante sur 4 bits.
```

2. **Opérateurs arithmétiques**

Les opérations arithmétiques ne sont pas toutes synthétisables, et encore moins applicables à tous les types.

Opération		Type concerné	Synthètisable
+	addition	integer	oui
-	soustraction	signed	oui
*	multiplication	unsigned	pas toujours
/	division		non
**	puissance		non

TABLE 4.1 – Les opérateurs arithmétiques

Exemple :

```
entity op_arith is
Port ( a : in integer ;
       b : in integer ;
       s1 : out integer ;
       s2 : out integer) ;
end op_arith ;
architecture Behavioral of op_arith is
begin
s1<= a + b ;
s2<= a - b ;
end Behavioral ;
```

3. **opération de concaténation**

Elle permet de regrouper plusieurs bits et mots en un seul et même mot. L'opérateur de concaténation se note **&**.

Exemple :

```
entity ex is
Port ( e : out STD_LOGIC_VECTOR (3 downto 0) ;
       f : out STD_LOGIC_VECTOR (3 downto 0)) ;
end ex ;
architecture Behavioral of ex is
constant a :std_logic :='1' ;
constant b :std_logic :='0' ;
constant c :std_logic :='1' ;
constant d :std_logic :='0' ;
begin
e<=a & b & c & d ;
f<= b & a & d & c ;
end Behavioral ;
```

4.4 Type de données

En plus des types classiques que l'on rencontre dans les langages informatiques, on trouve aussi des types spécifiques au VHDL [2].

4.4.1 Types communs

bit	(0, 1) peu utilisé car trop limité en simulation
Boolean	(TRUE, FALSE) résultat des tests. Peu utilisé comme tel
natural	ne pas utiliser sans borner les nombres (pour la synthése)
integer	
real	ne jamais utiliser car non synthétisable
character	parfois pratique en simulation (non synthétisable)

TABLE 4.2 – Les types communs

4.4.2 Types std_logic et std_logic_vector

C'est le type fondamental du VHDL. Il est décrit dans le package std_logic_1164. Ce type a été créé pour matérialiser tous les cas possibles en simulation, tant électriques que fonctionnels ou logiques.Il comporte 9 états.

Type	Sens anglais	Signification physique	Utilisation
U	Uninitialized	signal non initialisé	débogage
X	Conflict	court circuit	débogage
0	0	niveau bas fort (liaison directe)	très courante
1	1	niveau haut fort (liaison directe)	très courante
Z	High impedance	3 états en mode isolement fort	très courante
H	High	niveau haut résistif faible (liaison résistive)	rarement utilisés : spécifiques à des modélisations particuliéres
L	Low	niveau bas résistif faible (liaison résistive)	
W	Weak	mode isolement faible	
-	Do not care	sans importance	

TABLE 4.3 – Les états du std_logic[2]

4.4.3 Types signed et unsigned

Ils sont décrits dans le package **numeric_std**. Ils font le lien entre un **std_logic_vector** et sa valeur numérique décimale associée. Le type **unsigned** associe la valeur décimale du code binaire naturel contenu dans le vecteur. Le type **signed** associe la valeur décimale du code complement à 2 contenu dans le vecteur.Ils se déclarent de la même façon que le **std_logic_vector**.

Exemple : **signal** u : **unsigned** (4 **downto** 0) ; *–décimal sur 4 bits (donc de 0 à 15)*

4.4.4 Conversion de types

Entre vecteur et valeur numérique associée à ce vecteur, on doit pouvoir passer de l'un à l'autre. On dispose de fonctions de conversion qui permettent ces opérations :

vec1 <= **std_logic_vector** (valnum1)

valnum2 <= **unsigned** (vec2)

Ces conversions sont indispensables pour réaliser des opérations arithmétiques simples.

4.4.5 Type time

Le temps n'est bien sûr pas synthétisable. Mais, pour faire de la simulation ou des modèles logico-temporels, on a besoin de références temporelles. En VHDL, l'unité de temps est basée sur la seconde qui s'écrit sec.

Symbol	Valeur	signification
hr	60 min	heure
min	60 sec	minute
sec	1000 ms	seconde
ms	10^{-3} sec	milliseconde
us	10^{-6} sec	microseconde
ns	10^{-9} sec	nanoseconde
ps	10^{-12} sec	picoseconde
fs	10^{-15} sec	femtoseconde

TABLE 4.4 – Les symboles de temps en VHDL

4.4.6 Type énuméré

Dans certains cas, il est pratique de créer son propre type, ce qui permet de manipuler ses propres objets.

Exemple : **type** nom_du_type **is** nom1, nom2, nom3, nom4 ; – *liste finie*
signal sig : nom_du_type ; – *création d'un signal de ce type*

4.4.7 Tableaux

Ils sont utiles pour créer des modèles de mémoires. Il faut d'abord créer la structure, puis un signal fondé sur cette structure pour pouvoir l'utiliser[8].

Exemple : - -*création du type*
type memoire : **array** (0 **to** 4) **of std_logic_vector** (7 **downto** 0) ;
- - *mémoire morte de 5 octets*
constant rom : memoire := (x"a0", x"1f", x"35", x"9b", x"ff") ;

4.4.8 Enregistrement

Équivalents des structures de C, les enregistrements permettant de rassembler des objets de types différents dans une même organisation, et de repérer chaque élément par son nom [8].

Exemple :
```
type nom_de_mois is (jan,fev,mars,avr,mai,juin,juil,aout,sept,oct,nov,dec) ;
type date is
record
    jour : integer range 0 to 31 ;
    mois : nom_de_mois ;
    année :integer range 1000 to 3000 ;
end record
```

4.4.9 Attributs

Cette fonctionnalité permet de récupérer des informations sur un vecteur, ou sur un objet de type énuméré. Elle est introduite par ' collé après le nom du signal, suivi du nom de l'attribut. Voici les plus courants qui s'appliquent sur les std_logic_vector[12] .

Exemple :
```
signal a : std_logic_vector (7 downto 0) ;
signal b : std_logic_vector (1 to 5) ;
```

'range	plage entière	a**'range=** 7 **downto** 0	b**'range=** 1 **to** 5
'left	borne gauche	a**'left=** 7	b**'left=** 1
'right	borne droite	a**'right=** 0	b**'right=** 5
'high	borne haute	a**'high=** 7	b**'high=** 5
'low	borne basse	a**'low=** 0	b**'low=** 1

TABLE 4.5 – Les attributs[12]

4.5 Fonctions rising_edge et falling_edge

Ces fonctions sont très utiles. Elles indiquent l'apparition d'un front sur un signal de type std_logic. Elles vont donc permettre de décrire une bascule D. Elles renvoient l'information TRUE

ou FALSE, directement exploitable dans un test [2].

Exemple : rising_edge(h) - -renvoie TRUE si le front est montant
falling_edge(h) - -renvoie TRUE si descendant

4.6 Instructions concurrentes

Ces instructions servent le plus souvent à relier les process entre eux. Elles permettent de manipuler des signaux. On peut également construire de petites structures logiques. On distingue essentiellement deux instructions concurrentes dérivées de l'affectation [11].

4.6.1 Instruction d'affectation

Elle permet de recopier un signal sur un autre : a <= b ;
les signaux sont de même dimension, les signaux sont de même type.

4.6.2 Affectation conditionnelle

C'est la description d'un choix. Souvent cela traduit un multiplexeur.

Exemple : sig <= a **when** sel='0' **else** b ;

4.6.3 Affectation sélective

Avec cette instruction, on doit traiter tous les cas possibles.

Exemple : **with** k **select** - - *k std_logic_vector (1 downto 0)*
s <= a1 **when** "00", - - *donc 9×9 = 81 cas possibles*
a2 **when** "01" ;
a3 **when others** ; - - *pour tous les autres cas possibles*

4.6.4 Affectation implicite

En VHDL, toute description non complète engendre une mémorisation.Quand on écrit :
s <= x **when** a=='1' ;
On engendre une mémorisation implicite quand a='0', ce qui correspond à un D latch ayant le signal a comme entrée d'horloge.

4.7 Instructions séquentielles

Ce sont les descriptions les plus fréquentes. Elles permettent de faire des descriptions algorithmiques. On les rencontre uniquement dans les process. Les instructions y sont séquentielles, elles s'exécutent les unes après les autres [11].

4.7.1 Process

C'est la structure de base de la description séquentielle[12].

Exemple : **process** (sig1, sig2,...)- - *liste de sensibilité*
- - *zone de déclaration*
begin
- - *instructions séquentielles*
end process ;

Il s'exécute uniquement à chaque variation d'un des signaux de la liste de sensibilité. En dehors de cela, il reste en veille. Un process est comme une instruction concurrente. Ainsi, plusieurs process peuvent s'exécuter en même temps.

4.7.2 Liste de sensibilité

Dans cette liste de sensibilité, on trouvera :

1. L'horloge et le signal d'initialisation asynchrone en cas de process synchrone.
2. Toutes les entrées pour la description d'un bloc combinatoire.

4.7.3 Process et signaux

Tous les process sont concurrents entre eux. Ils s'exécutent en parallèle. Il ne faut pas oublier que le langage VHDL engendre des portes logiques. Or, toutes ces portes existeront physiquement toutes en même temps dans le circuit réalisé.

Pour que cela puisse fonctionner en simulation, il faut gérer correctement les différentes informations qui circulent dans le circuit par les signaux.

4.7.4 Variables

La variable est un « signal » particulier local au process et elle n'est pas visible ni accessible à l'extérieur du process. Elle est déclarée dans le process. Sa manipulation est délicate, car elle évolue instantanément dans le process lors de son exécution. On peut se passer de variable dans la plupart des cas[2].

Exemple : **process** (sig1, sig2,...)
variable var1 : **integer range** 0 **to** 31 ; - -*déclaration d'une variable locale*
begin – *écriture du driver d'entrée sig1(en entrée)*
var1 := sig1 ; - - *on écrit sur la variable*
var1 := var1 + 1 ; - - *on calcule immédiatement la nouvelle valeur*
sortie <= var1 ; - - *on écrit sur le driver du signal de sortie*
end process ; - - *mise a jour du signal sortie*

Si sig1=2 au début du process, on aura sortie=3 en fin de process.

4.7.5 Instructions séquentielles

On retrouve l'affectation et essentiellement des instructions de test et d'aiguillage. Certaines sont synthétisables, d'autres ne le sont pas.

1. **Affectation**

   On retrouve la même instruction qu'en mode concurrent :a <= b ;
   L'affectation pour une variable s'écrit :a := b ;

2. **Instruction de test if, then, else**

   Dans cette instruction, **else** est optionnel.

   – **L'instruction de test simple :**On décrit un seul test de la forme si... sinon :

   Exemple : **if** condition ;
   then instruction_10 ; instruction_11 ; - - *suite d'instructions*
   else instructions_20 ; instruction_21 ; - - *suite d'instructions*
   end if ;

– **Les tests multiples :** On décrit des tests de la forme si...sinon si...sinon... :

Exemple :
```
if condition_1 ;
then instruction_10 ; instruction_11 ; instruction_12 ;
elsif condition_2 ;
then instructions_20 ; instruction_21 ; instruction_22 ;
else instructions_30 ; instruction_31 ; instruction_32 ;
end if ;
```

– **Les tests imbriqués :**Ce sont des tests de la forme si... sinon(si... sinon(si... sinon..)...).. :

Exemple :
```
if condition_1 ;
then instruction_10 ; instruction_11 ; instruction_12 ;
if condition_2 ;
then instructions_20 ; instruction_21 ; instruction_22 ;
else instructions_30 ; instruction_31 ; instruction_32 ;
end if ;
end if ;
```

3. **Instruction case**

C'est l'instruction de test exhaustif. Le test doit donc étudier tous les cas possibles.Elle est bien adaptée à la traduction directe de table de vérité.

Cette instruction donne le meilleur résultat en synthèse. Elle est à préférer à l'instruction if avec de nombreuses imbrications.

Exemple : sig1 est un std_logic_vector (1 downto 0)
```
case sig1 is - - on teste tous les cas de sig1
when "00" => a <= '0' ; - - sig1="00"
when "10" => b <= "010" ; – sig1="10"
when "11" => c <= '0' ; – sig1="11"
when others => null ; - - pour tous les autres cas
end case ;
```

4. **Instructions de boucles**

   Elles permettent de faire des algorithmes combinatoires, ou de décrire des algorithmes dans les fonctions. Il faut les utiliser avec des variables car on doit avoir des variations immédiates des valeurs dans les calculs à chaque pas de la boucle.

   Ces instructions sont assez délicates à utiliser. On risque de ne pas obtenir de bons résultats en synthèse comme en simulation d'ailleurs[2].

   – **Instruction while :**Cette instruction n'est pas toujours supportée par les outils de synthèse.

   Exemple : **while** condition **loop** - - *la boucle s'exécute tant que la condition est vraie*
   instruction_1 ; instruction_2 ;
   end loop ; - - *fin de boucle*

   – **Instruction for, loop :**Moins ambiguë que la boucle while, elle n'en reste pas moins délicate à utiliser pour la synthèse. L'index de boucle (ci-dessous j) est implicite et n'a pas besoin d'être déclaré[2].

   Exemple : **for** j **in** 5 **to** 10 **loop** - - *les bornes sont strictement croissantes*
   a(j) := a(j) + j ; - - *calcul sur des variables*
   exit when a(j) = 30 ; - - *permet la sortie anticipée*
   b(j) <= a(j) ; - - *mise à jour du signal b*
   end loop ;

4.8 Programmation modulaire

4.8.1 Les modules

VHDL, comme tous les langages évolués, permet de subdiviser un algorithme complexe en modules plus simples, compilables et testables séparément, que l'on peut ranger dans des librairies et importer a volonté.

VHDL décrit des circuits, des architectures matérielles. Un circuit complexe peut être obtenu en assemblant des blocs fonctionnels plus simples, blocs internes a une architecture ou composants,

pour lesquels des politiques différenciées de synthèse (placement et routage, compromis vitesse-surface lors de l'optimisation) peuvent être établies [11].

Le langage offre donc deux outils complémentaires de constructions modulaires [12] :

- Un outil purement logiciel, qui passe par la constitution de sous-programmes, de paquetages et de librairies.
- Un outil matériel, nomme traditionnellement construction hiérarchique.

4.8.2 Les sous-programmes

VHDL connait deux catégories de sous-programmes : les procédures et les fonctions. Les sous-programmes sont des modules de code séquentiel ; ils utilisent donc le même jeu d'instructions que les processus.

La création et l'emploi d'un sous-programme passent par trois opérations distinctes :

- Sa création proprement dite, ou définition, consiste a écrire l'algorithme qui constitue le corps du sous-programme.
- Sa déclaration consiste à rendre connu d'un module qui l'utilise le prototype du sous-programme, c'est-à-dire son nom, ses arguments d'appels et, dans le cas d'une fonction, le type de la valeur retournée.
- Son utilisation dans une instruction établit un lien entre les paramètres formels, utilises dans la définition et la déclaration, et les paramètres réels qui appartiennent au module appelant.

Les procédures et les fonctions diffèrent principalement par les sens de transfert des informations, alors que les premières autorisent des paramètres en entrée et en sortie, les secondes n'ont que des paramètres qu'en entrée, et renvoient un résultat utilisable dans une expression.

La définition du corps d'un sous-programme comporte, comme à l'habitude, deux parties : une partie déclarative ou sont définis les objets (données, types ...) locaux et le corps proprement dit qui contient une suite d'instructions séquentielles[12].

La syntaxe d'un sous-programme est : **procedure** nom [(liste_des_paramètres_formels)]

function nom [(liste_des_paramètres_formels)] **return** type

is

- -zone déclarative

Begin

- -zone d'instructions séquentielles

end [**procedure** | **function**] [nom] ;

Exemple :

On va réaliser la somme logique de deux éléments a et b,le programme est le suivant :

procedure	fonction
library IEEE ;	**library IEEE** ;
use IEEE.STD_LOGIC_1164.ALL ;	use IEEE.STD_LOGIC_1164.ALL ;
use IEEE.STD_LOGIC_ARITH.ALL ;	use IEEE.STD_LOGIC_ARITH.ALL ;
use IEEE.STD_LOGIC_UNSIGNED.ALL ;	use IEEE.STD_LOGIC_UNSIGNED.ALL ;
entity ex_pro **is**	**entity** ex_pro **is**
Port (a : in **STD_LOGIC** ;	Port (a : in **STD_LOGIC** ;
b : in **STD_LOGIC** ;	b : in **STD_LOGIC** ;
s1 : out **STD_LOGIC**) ;	s1 : out **STD_LOGIC**) ;
end ex_pro ;	**end** ex_pro ;
architecture Behavioral **of** ex_pro **is**	**architecture** Behavioral **of** ex_pro **is**
procedure somme(**signal** a1 :in **std_logic** ;	**function** somme(**signal** a1 :in **std_logic** ;
signal b1 :in **std_logic** ;	**signal** b1 :in **std_logic**)
signal s :out **std_logic**)**is**	**return** std_logic **is**
begin	variable s :std_logic ;
s<=a1 or b1 ;	**begin**
end procedure somme ;	s :=a1 or b1 ;
begin	return s ;
somme(a,b,s1) ;	**end function** somme ;
end Behavioral ;	**begin**
	s1<=somme(a,b) ;
	end Behavioral ;

le résultat de simulation de ce programme est :

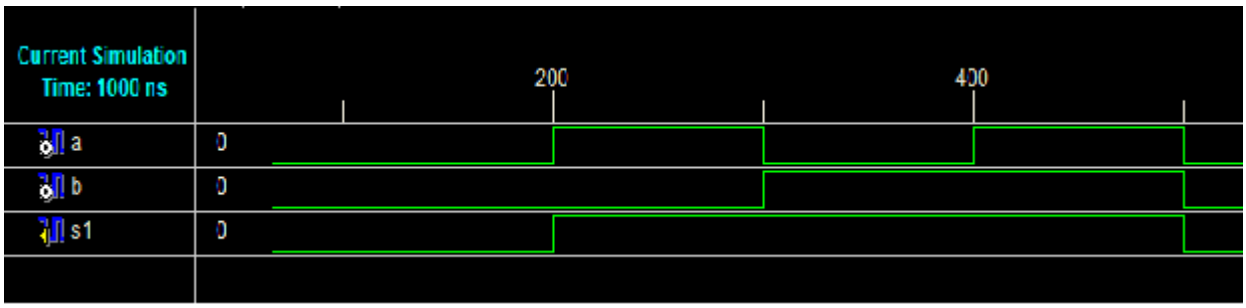


FIGURE 4.1 – simulation d'une somme logique

4.9 Conclusion

Dans ce chapitre nous avons vu quelque notions sur le langage de description des circuits électroniques qui est le VHDL.

Les instructions utilisées en VHDL ne sont pas tous synthétisables par exemple la division et la puissance des nombres. Donc pour utiliser ces instructions il faut les programmer. même certain types ne sont pas prédéfinit comme les nombres réels. Donc le langage VHDL synthétisable est une partie du langage VHDL de modélisation et de description.

Dans le chapitre suivant, nous allons implémenter les deux techniques MLI engedrée et calculée sur une carte FPGA de Xilinx.

Chapitre 5

Implémentation des techniques MLI sur un circuit FPGA

5.1 Introduction

Dans ce chapitre, on va exploiter les résultats qui ont été trouvés précédemment pour programmer deux techniques MLI, engendrée et calculée, en langage VHDL puis les implémenter sur la carte Memec Design V2MB1000 ayant Virtex-II de Xilinx. On termine par l'étude des caractéristiques de la tension de sortie de l'onduleur en utilisant le logiciel PSIM.

5.2 Description du circuit de puissance

Pour valider les résultats de simulation, nous avons émulé un bras d'onduleur à l'aide d'un amplificateur opérationnel et un circuit d'isolation de la carte FPGA. La figure 5.1 donne le schéma du circuit utilisé.

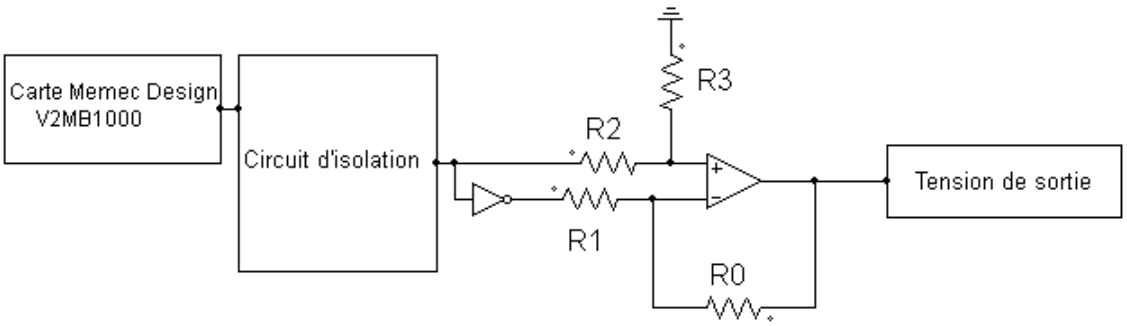


FIGURE 5.1 – Circuit de commande d'un bras d'onduleur

Pour que l'amplificateur opérationnel joue le rôle d'un comparateur il faut que les rapport $\frac{R0}{R1} = \frac{R2}{R3}$. Le gain de l'ampli est $\frac{R0}{R1}$. Dans notre application on a pris un gain $\frac{R0}{R1} = 2$.

Les résultats expérimentaux sont visualisés sur un oscilloscope numérique, puis récupérés sur ordinateur par une carte d'acquisition Instrunet, afin de les interpréter et les comparer avec les résultats obtenus en simulation.

5.3 Implémentation d'une commande MLI engendrée

5.3.1 Introduction

Nous allons maintenant implémenter la commande MLI engendrée qui a été décrite précédemment, sur un circuit FPGA. Pour cela on utilise le langage VHDL. Dans une première étape on génère le signal sinus, et dans l'étape suivante on génère le signal triangulaire après on fait la comparaison entre les deux signaux ; dans la dernière étape on simule notre programme en utilisant le logiciel ISE de Xilinx et on termine par le test de notre commande sur la carte de développement Memec Design V2MB1000.

5.3.2 Programme de la commande

1. **Génération du signal de référence**

Nous faisons générer par le FPGA un signal de référence numérique sinusoïdal d'amplitude constante, dont on choisit la fréquence de sinusoïde à partir de la fréquence d'horloge du circuit. Nous utilisons pour cela un compteur associé à une ROM, dans laquelle est stocké le motif de sinus.

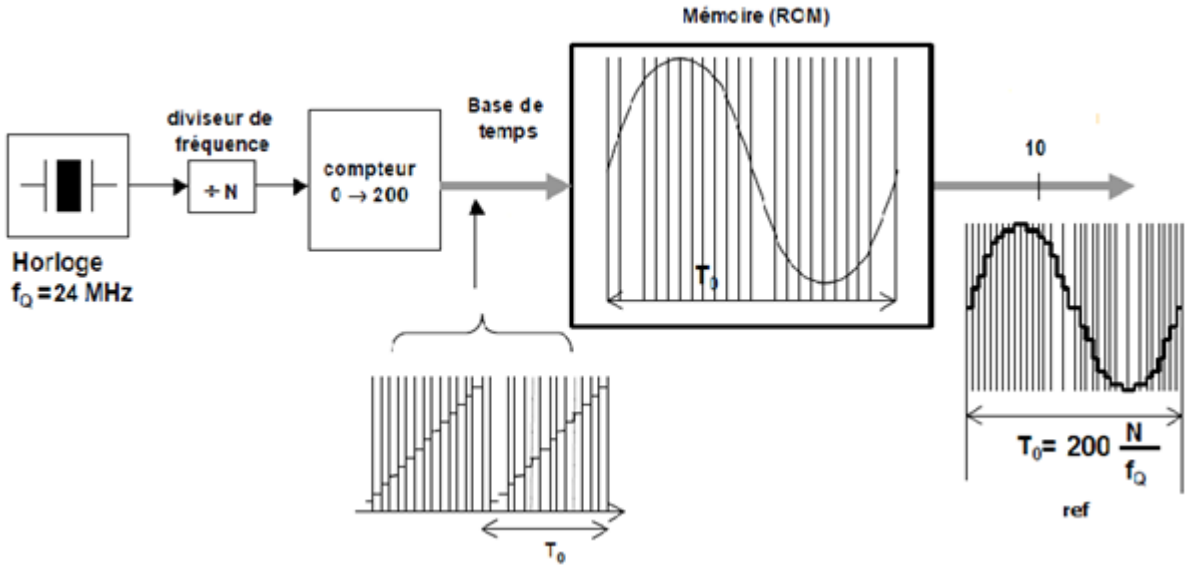


FIGURE 5.2 – principe de génération d'un signal sinusoïdal

En choisissant le rapport de division de fréquence entre le signal d'horloge à $24MHz$ et le compteur, nous pouvons modifié la fréquence de la sinusoïde. Sachant que la fréquence apparente de découpage de la tension de sortie est 200 fois moindre que celle du quartz, nous pouvons obtenir une fréquence de référence qui vaut :

$$f_0 = \frac{f_q}{200 \times N}$$

Avec : f_q : fréquence du signal de quartz

f_0 : fréquence apparente de découpage de la tension de sortie

N : rapport de division de fréquence

2. **Diviseur de fréquence**

Dans notre programme on a besoin de travailler avec des fréquences de synchronisation différentes pour générer des sinusoïdes et des porteuses à fréquences variables, mais sur la carte de développement il existe deux fréquences $f_1 = 24MH$, $f_2 = 100MHz$, donc pour résoudre le problème on utilise un diviseur de fréquence à coefficient variable. L'organigramme du programme du diviseur de fréquence est donné par la figure 5.3.

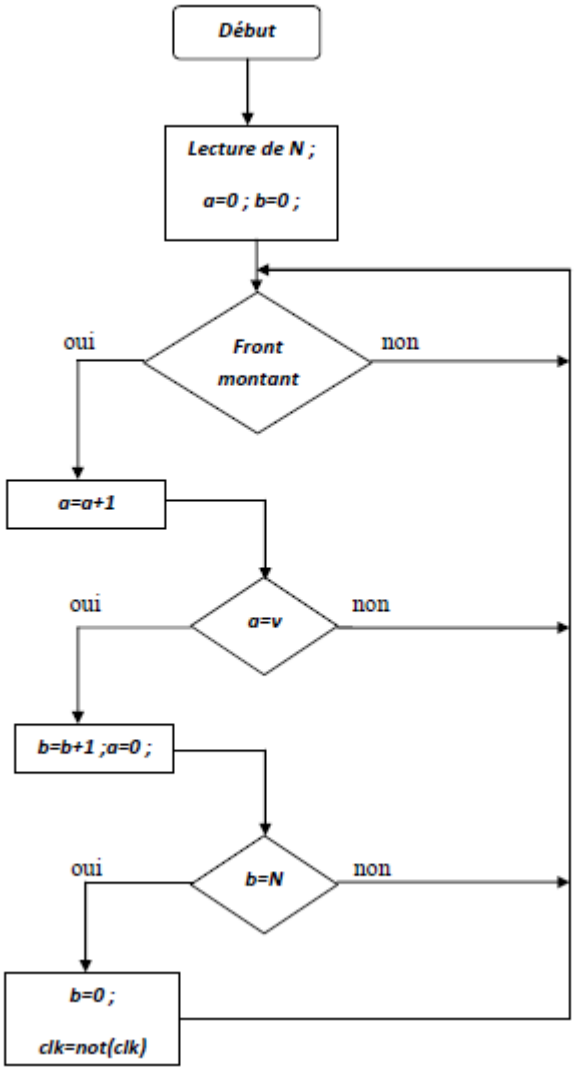


FIGURE 5.3 – Organigramme d'un diviseur de fréquence

à la sortie du programme, la fréquence résultante clk est donnée par :

$$f_r = \frac{f_q}{2 \times V \times N}$$

Avec : f_q la fréquence de l'horloge du circuit. En modifiant N et V, on peut régler la fréquence de sortie.

3. **Génération de la porteuse**

Dans notre cas on utilise une porteuse triangulaire à amplitude constante, pour la générer numériquement on utilise un compteur-décompteur avec un pas fixe et une fréquence variable. L'organigramme de ce programme est donné par la figure 5.4.

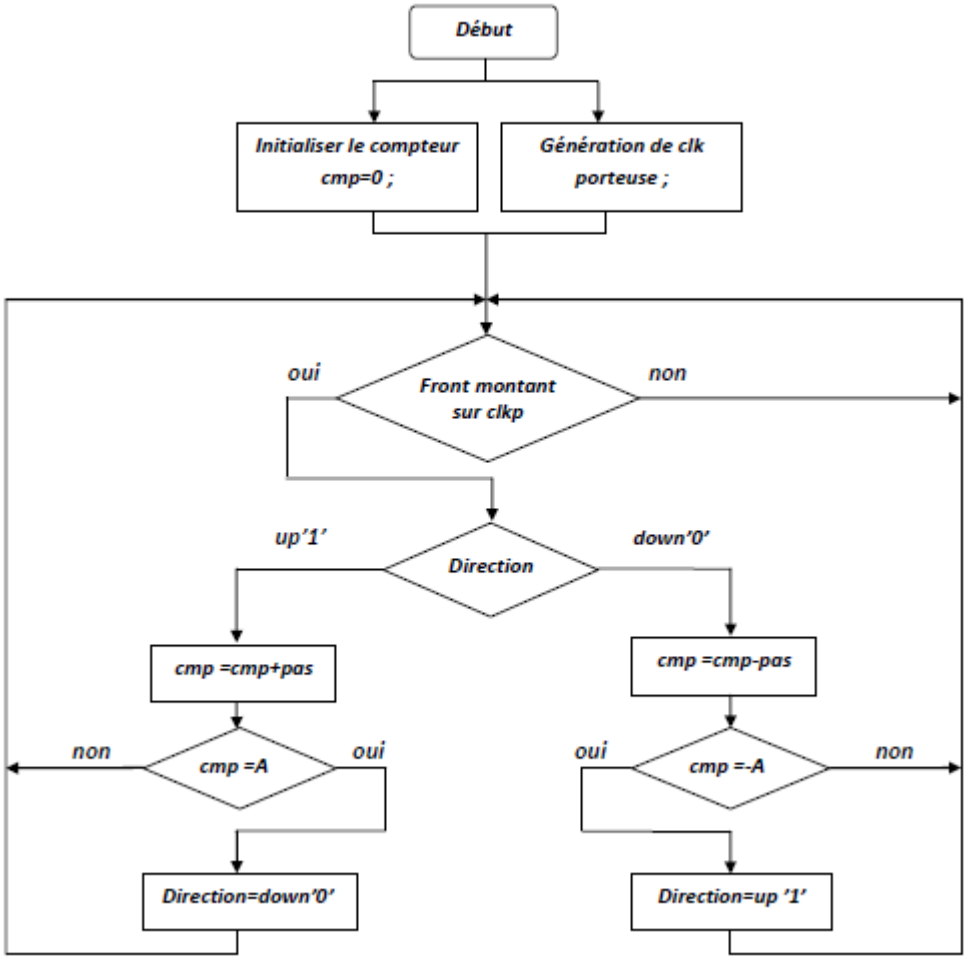


FIGURE 5.4 – Organigramme de génération d'un signal triangulaire

A fin du sous programme on obtient comme résultat une porteuse triangulaire échantillonnée d'amplitude A et une fréquence :

$$f_p = \frac{f_q}{2 \times 4 \times \frac{A}{pas} \times N}$$

Le paramètre A est l'amplitude de la porteuse triangulaire égal à l'unité. Le choix du pas et du coefficient du diviseur de la fréquence dépend de l'indice de modulation m.

4. **Vue générale du programme**

Comme le langage VHDL offre des outils élémentaire seulement pour la programmation des circuit FPGA (+, -, *, not,and) pour des types bien définis (logique, binaire, entier) certaines opérations ne sont pas réalisables tel que la multiplication des flottantes ; alors il faut augmenter l'échelle de travail pour ne pas avoir de telles erreurs.

Pour notre programme on va lire le taux de modulation r et l'indice de modulation m et on génère les commandes des transistors de l'onduleur.

L'organigramme du programme d'une commande MLI engendrée est montré par la figure 5.5.

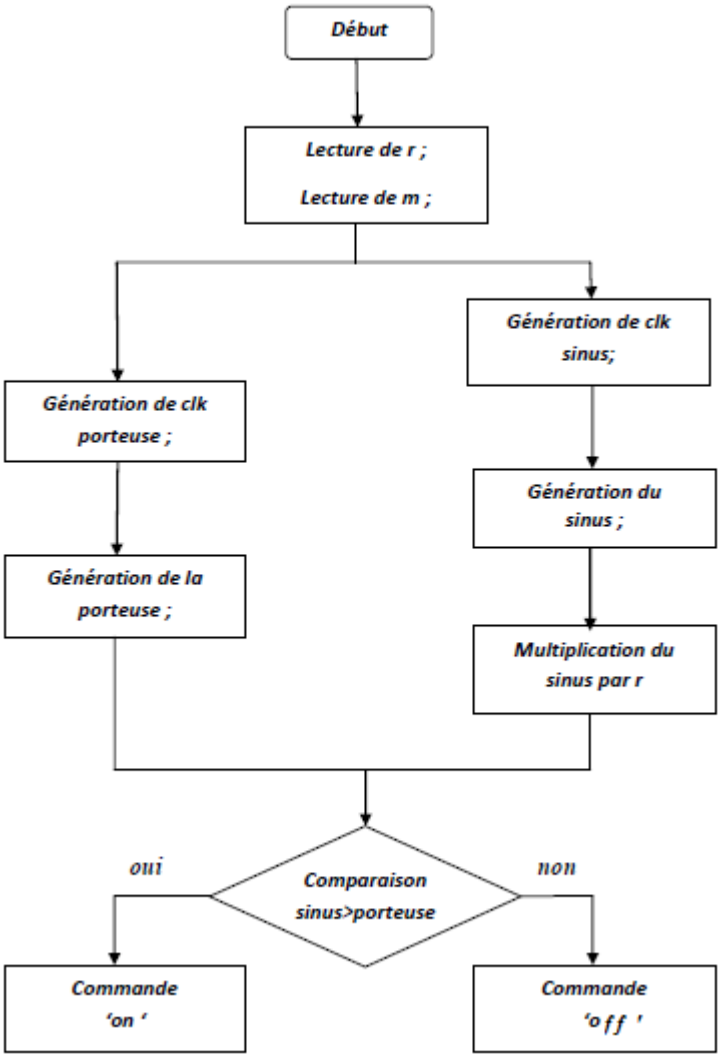


FIGURE 5.5 – Organigramme d'une commande MLI engendrée

5.3.3 Résultats de simulation

Pour vérifier le fonctionnement de notre programme ; on le simule par Xilinx ISE 9.2i pour des valeurs différentes du coefficient de réglage r et de l'indice de modulation m (figures 5.6 à 5.8).

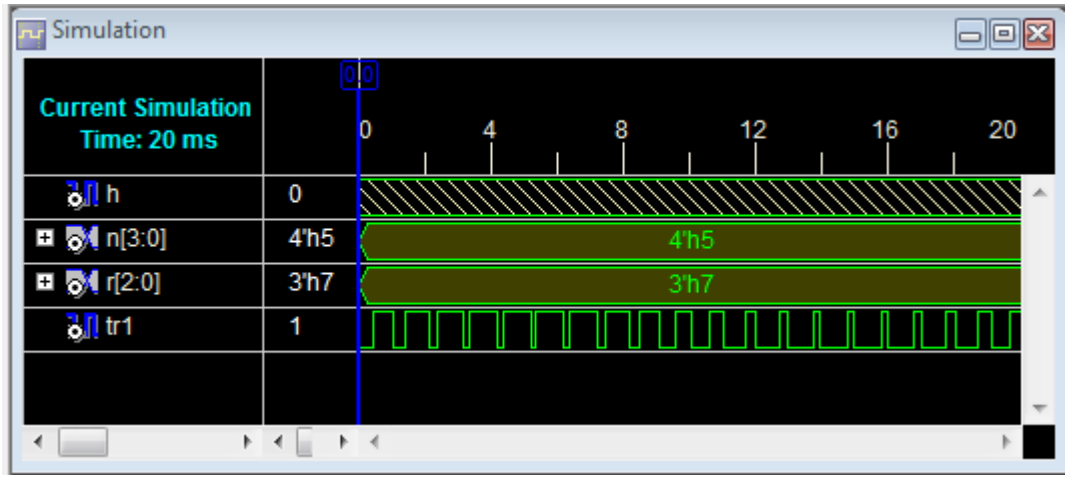


FIGURE 5.6 – Résultat de simulation pour $f_r = 50Hz$, $m = 20$ et $r = 0.7$

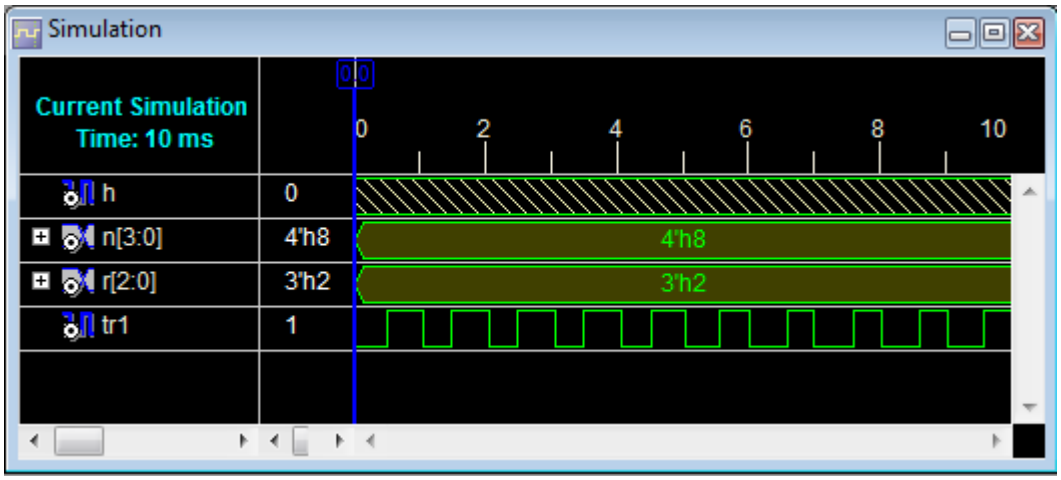


FIGURE 5.7 – Résultat de simulation pour $f_r = 100Hz$, $m = 10$ et $r = 0.2$

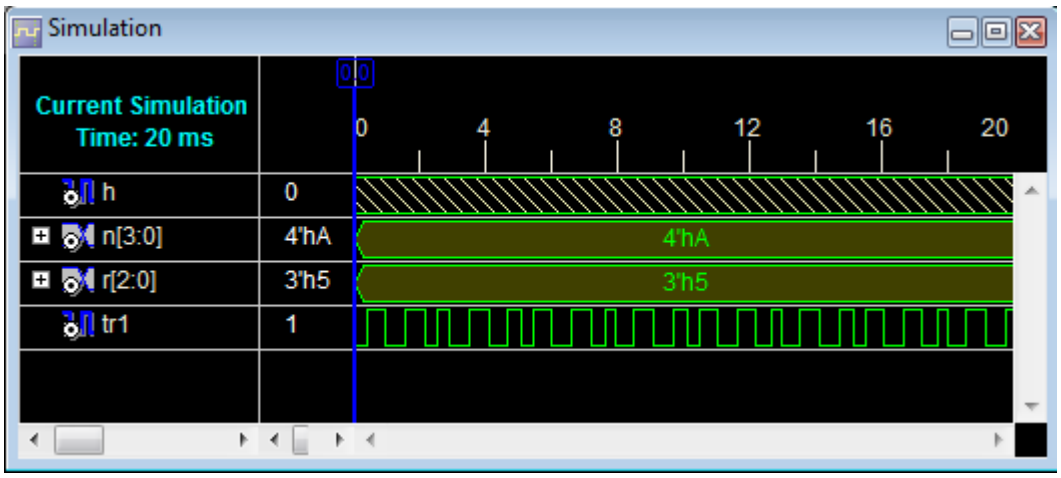


FIGURE 5.8 – Résultat de simulation pour $f_r = 60Hz$, $f_p = 1kHz$ et $r = 0.5$

5.3.4 Implémentation de la commande

Pour implémenter la commande on choisit sur la carte comme valeur d'horloge de synchronisation de fréquence $24MHz$, et comme interface de commande l'interface (DIP switch) qui contient huit entrées exploitables par l'utilisateur qui peuvent être mises statiquement à un état haut ou bas, les entrées sont l'indice de modulation codé sur 4 bits et le coefficient de réglage r codé sur 3 bits. de plus une entrée $remise$ qui fait la remise à zero de tous les paramètres du programme.

Le tableau 5.1 résume les affectations des broches entrée-sortie.

L'entrée/sortie	h	remise	n(3)	n(2)	n(1)	n(0)	r(2)	r(1)	r(0)	tr
Pin	a11	b4	a4	c4	c5	b5	a5	d6	c6	f3

TABLE 5.1 – Affectation des broches d'entrées-sorties

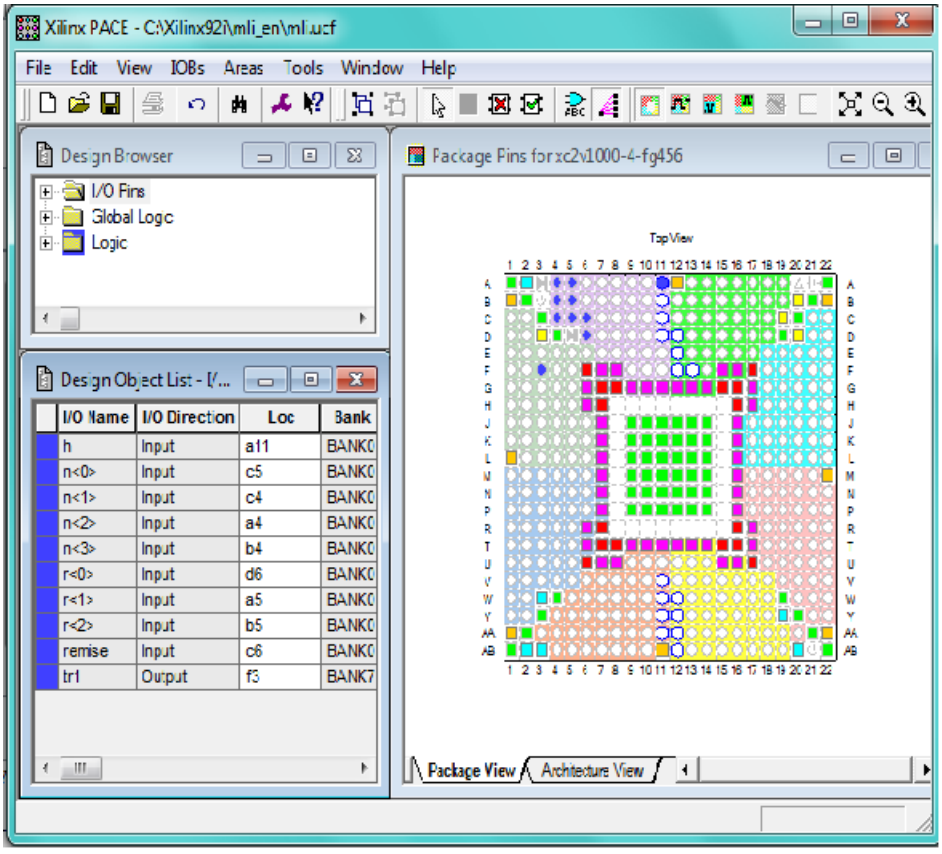


FIGURE 5.9 – Aperçu de l'outil d'affectation des broches d'entrées-sorties

5.3.5 Visualisation des résultats sur l'osciloscope

Comme nous l'avons mentionné précédemment, il y a une différence entre la simulation et l'implémentation ; car il y a des cas où on peut simuler le programme mais on ne peut pas l'implémenter.

Pour vérifier que la carte est bien programmée, on visualise la sortie sur un osciloscope en changeant les paramètres de la commande (figures 5.10 et 5.11).

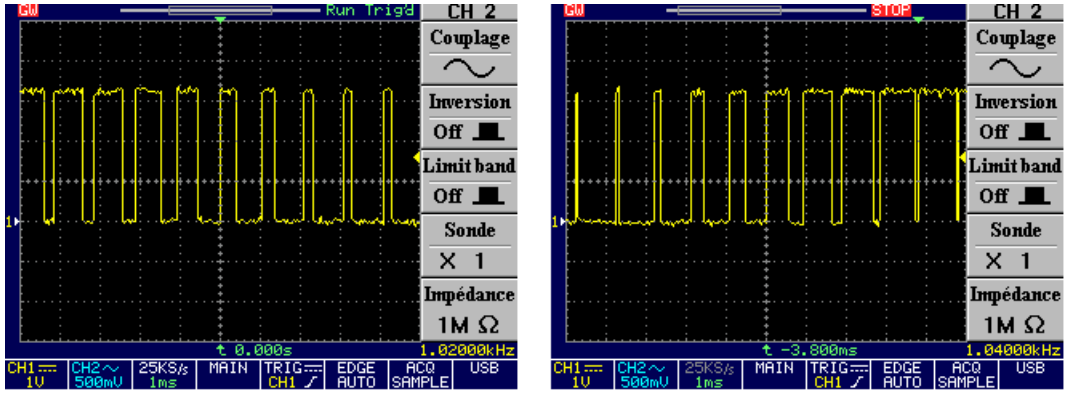


FIGURE 5.10 – Commande MLI engendrée pour $m = 20, \quad f_r = 50Hz$ et $r_1 = 0.6; \quad r_2 = 0.9$

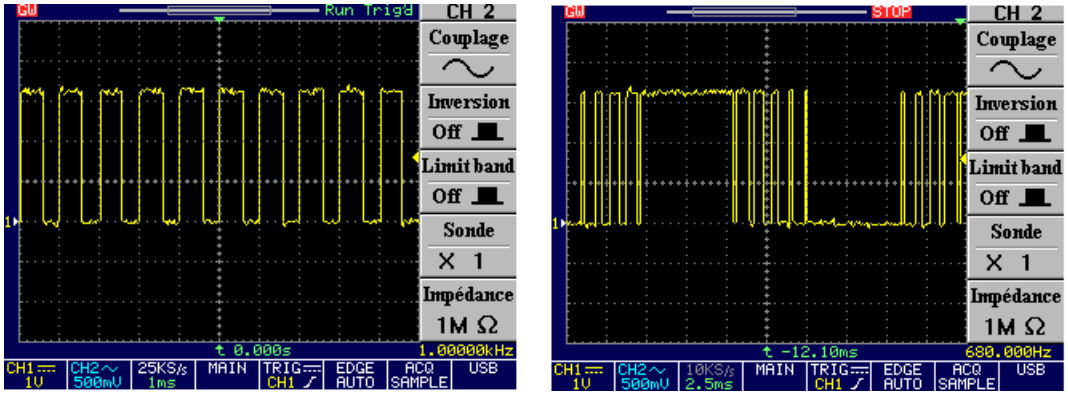


FIGURE 5.11 – Commande MLI engendrée pour $m = 20, \quad f_r = 50Hz$ et $r_1 = 0.2; \quad r_2 = 1.2$

5.3.6 Étude du spectre

Pour étudier les spectres des tensions de sorties, on fait une comparaison entre les résultats obtenus par simulation et les résultats expérimentaux en utilisant le logiciel PSIM.

1. **Résultats de simulation**

La figure 5.12 représente un onduleur en demi-pont alimentant une charge RL. Il est à noter que le récepteur est une résistance $R = 3Ohm$ en série avec une inductance $L = 10mL$, avec une tension d'entrée $E = 200V$.

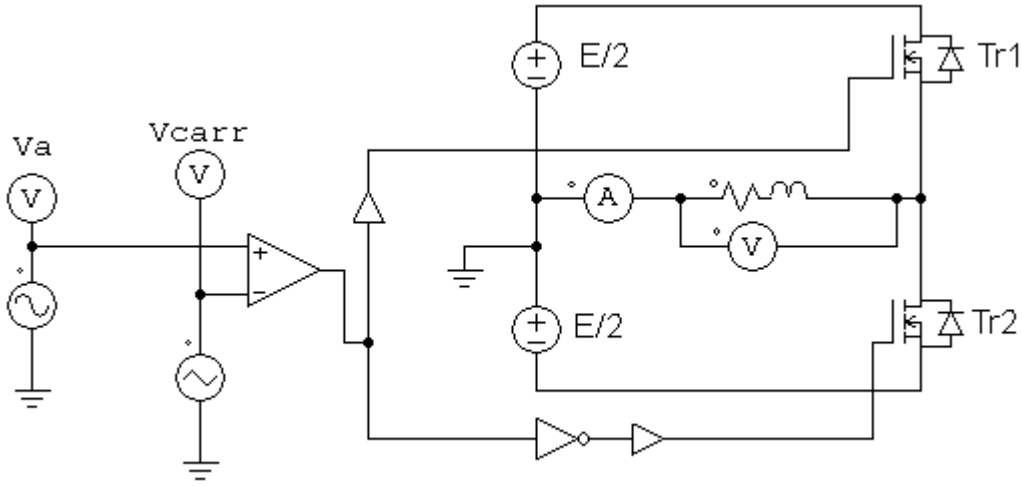


FIGURE 5.12 – Onduleur en demi-pont sous PSIM

Le signal de commande de l'interrupteur $Tr1$ et la forme de la tension de sortie sont tels indiqués à la figure 5.13. L'indice de modulation $m = 30$ et le taux de modulation $r = 0.8$.

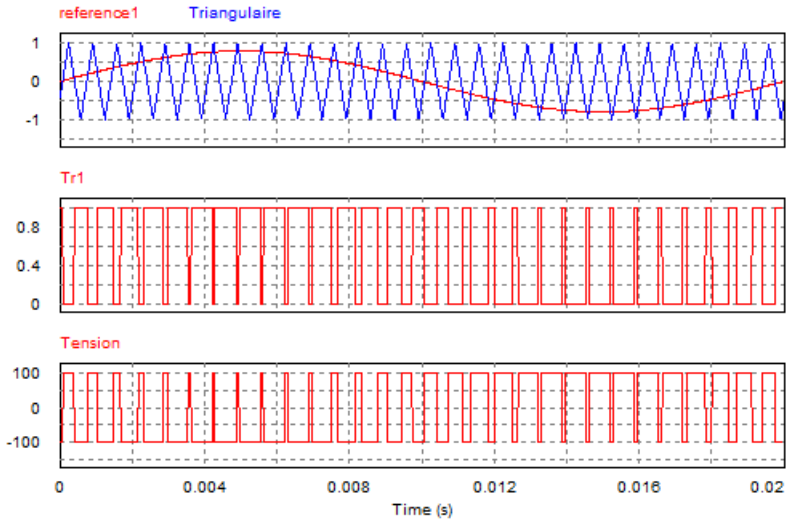


FIGURE 5.13 – Commande MLI engendrée de l'onduleur en demi-pont $m = 30$ et $r = 0.8$

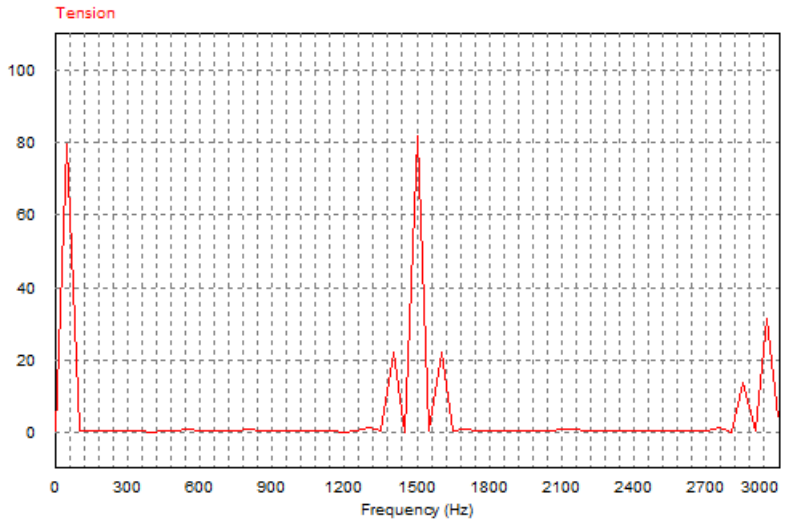


FIGURE 5.14 – Spectre de la tension de sortie $m = 30$ et $r = 0.8$

D'après l'allure du spectre de la figure 5.14, on voit que les harmoniques apparaissent aux fréquences : 1400, 1500 et 1600 Hz. Donc : en général ils apparaissent à la fréquence normalisée f_h par paquet centrée autour de m et ces multiples, explicitement on a : $h = l \times m \pm k$ où k doit être paire pour l impaire et inversement.

2. **Résultats expérimentaux**

La tension de sortie de l'onduleur et son spectre pour $m = 30$ et $r = 0.8$ sont donnés par la figure 5.15. D'après les résultats obtenus, on voit que les spectres dans les deux cas (simulation et expérimental) sont pratiquement les mêmes.

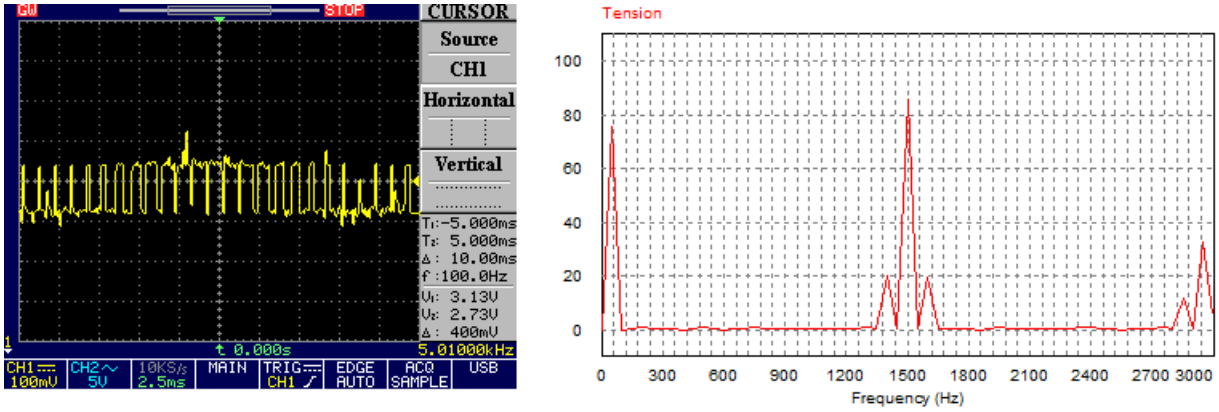


FIGURE 5.15 – Tension et spectre de sortie d'un onduleur en demi-pont pour $m = 30$ et $r = 0.8$

3. **Interprétation des résultats**

Dès que m est suffisamment grand (supérieur ou égal à 9) l'amplitude du fondamental de la tension de sortie est pratiquement égale à la tension de référence multiplié par r.

$$V_1 = r\frac{E}{2}$$

Par action sur r, on ne peut faire croître V_1 depuis zéro que jusqu'à un maximum correspondant à l'annulation de certain créneaux de la tension de sortie v à cause de la disparition de certaines intersections entre les ondes de référence et de modulation si $r > 1$.

La modulation triangulo-sinusoïdal ne réduit pas le taux harmoniques, celui-ci est indépendant de m et très important puisque :

$$\sqrt{\sum_{n=2}^{+\infty} v_n^2} = \frac{E}{2}\sqrt{1-\frac{r^2}{2}} \quad \Rightarrow THD = \frac{\sqrt{\sum_{n=2}^{+\infty} v_n^2}}{V} = \sqrt{1-\frac{r^2}{2}}$$

Mais elle permet de pousser les harmoniques vers les fréquences élevées. Si m est impair et multiple de trois, les harmoniques de rang 3 sont supprimés et la première famille d'harmoniques apparait à la fréquence $(m-2)f_r$ et $(m+2)f_r$.

5.3.7 Conclusion

L'étude du spectre de la tension de sortie montre que l'on obtient un fondamental dont la fréquence et l'amplitude dépendent de celles de la référence. Les harmoniques sont d'amplitudes importantes mais de fréquences proches de celle de la porteuse donc très élevées. Le filtrage est donc très facile. On peut même vérifier que le courant de la charge est très proche d'une sinusoïde.

5.4 Implémentation d'une commande MLI calculée off-line

5.4.1 Introduction

Ce paragraphe concerne la programmation et l'implémentation d'un programme off-line basé sur la technique MLI à élimination des harmoniques. Pour la programmation il faut d'abord calculer les angles exacts de commutation pour des m et r différents et ensuite on génère les signaux de commande des commutateurs.

5.4.2 Programme d'une commande MLI off-line

Le calcul des angles de commutation se fait par la résolution de système non linéaire à l'aide de la méthode de Newton-Raphson décrite dans le chapitre 2. En utilisant une mémoire on stocke dans l'ordre ces résultats sur la carte. Pour la génération des signaux de commande, un compteur est utilisé. Chaque fois que les valeurs stockées sont atteintes par le compteur le signal de commande commute.

Le tableau 5.2 donne à titre d'exemple les valeurs exactes des angles de commutation pour $m = 5$.

r	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
α_1	19.12	18.23	17.33	16.41	15.47	14.52	13.54	12.53	11.48
α_2	20.45	20.90	21.35	21.78	22.20	22.58	22.92	23.18	23.30
α_3	39.09	38.16	37.21	36.24	35.24	34.20	33.10	31.92	30.61
α_4	40.92	41.44	42.16	42.88	43.59	44.29	44.96	45.60	46.13
α_5	59.13	58.25	57.36	56.45	55.53	54.57	53.58	52.53	51.37

TABLE 5.2 – Les angles de commutation exactes pour $m = 5$

L'organigramme complet du programme est illustré dans la figure 5.16.

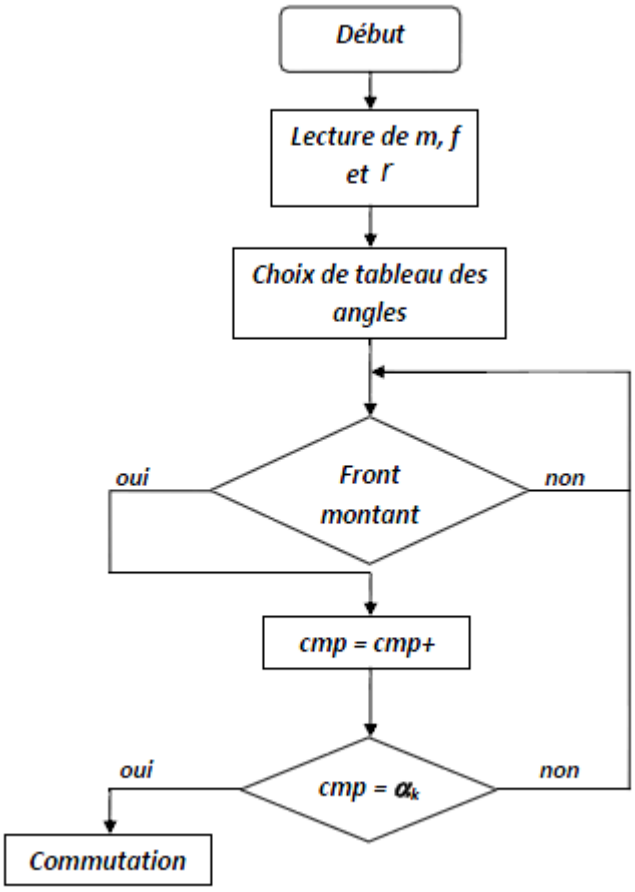


FIGURE 5.16 – Organigramme d'une MLI off-line

5.4.3 Résultats de simulation

Dans notre cas, on a étudié que deux valeurs de m ; $m = 3$ et $m = 5$, et pour des valeurs de r variant de 0.6 jusqu'à 1.

Les figures 5.17, 5.18 et 5.19 montrent les résultats de simulation de notre programme pour une fréquence $f = 50Hz$.

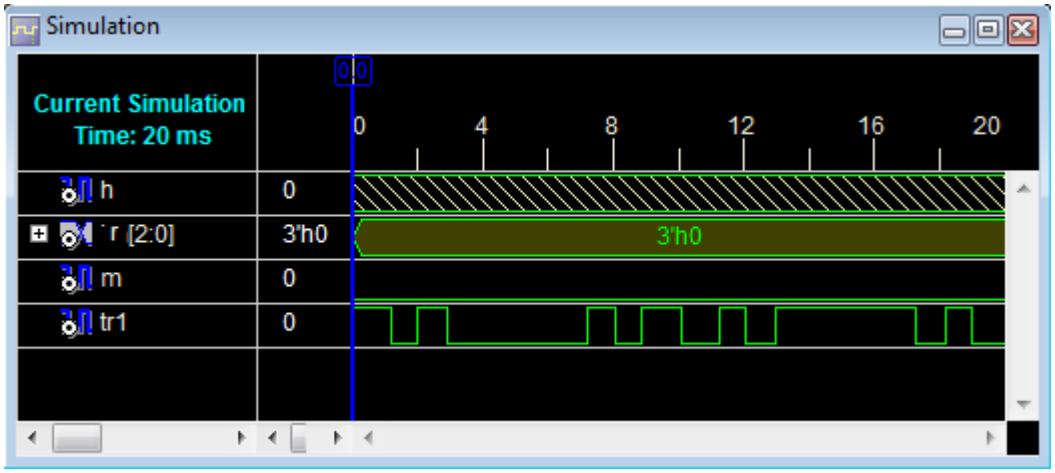


FIGURE 5.17 – MLI calculée pour $m = 3$ et $r = 0.6$

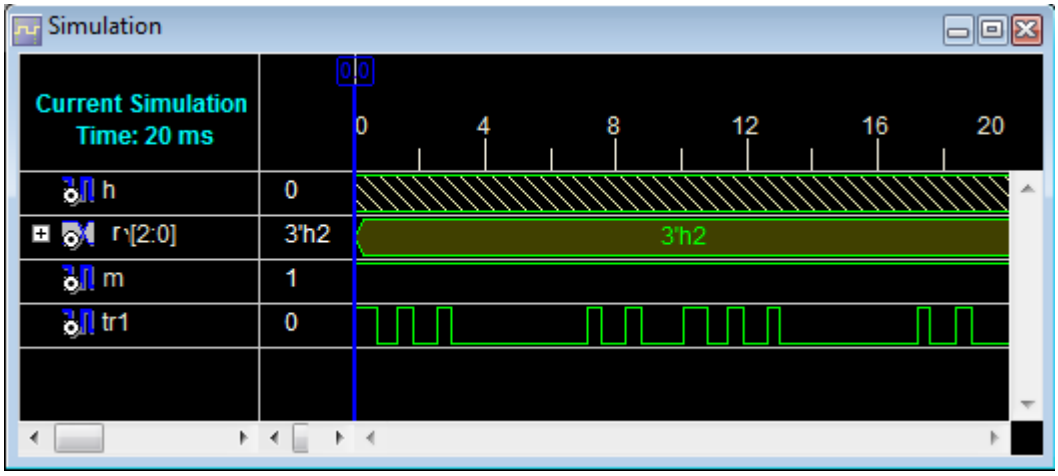


FIGURE 5.18 – MLI calculée pour $m = 5$ et $r = 0.8$

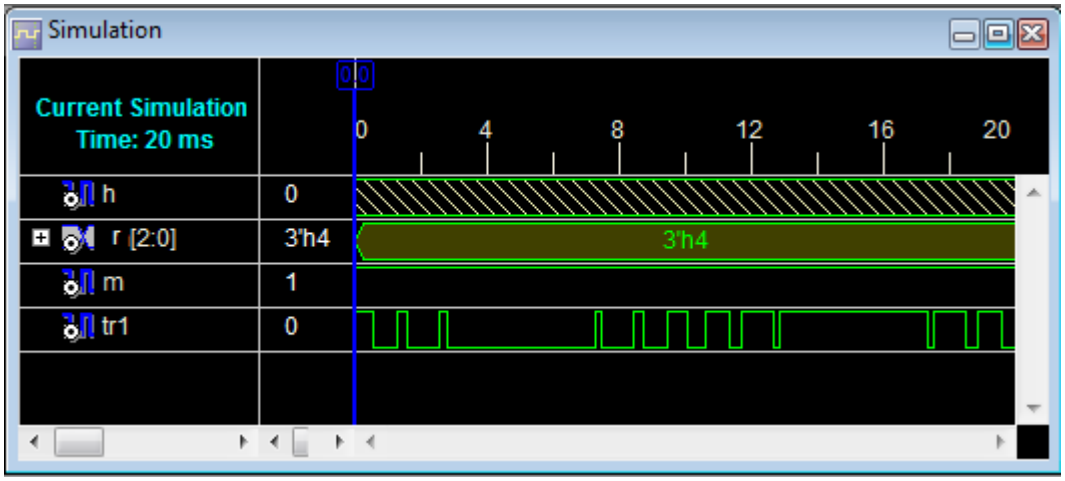


FIGURE 5.19 – MLI calculée pour $m = 5$ et $r = 1$

5.4.4 Implémentation de la commande

Pour cette commande les entrées sont la fréquence, le taux de modulation r et le nombre de commutation m. Les deux premiers sont codés sur 3 bits, et le m est codé sur un seul bit.

Le tableau 5.3 résume les affectations des broches entrée-sortie.

L'entrée/sortie	h	f(2)	f(1)	f(0)	r(2)	r(1)	r(0)	m	tr
Pin	a11	b4	a4	c4	c5	b5	a5	d6	f3

TABLE 5.3 – Affectation des broches d'entrées-sorties

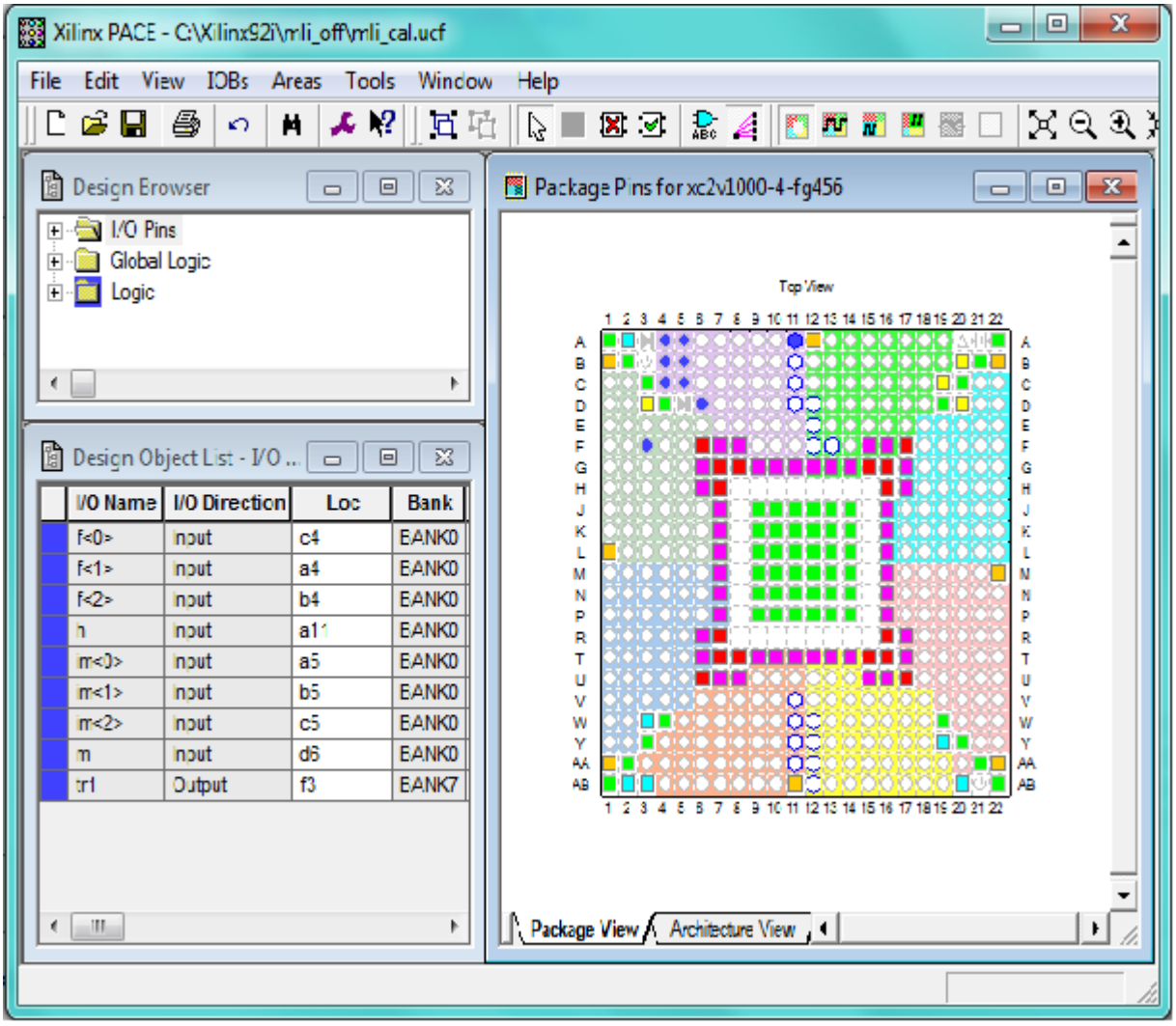


FIGURE 5.20 – Aperçu de l'outil d'affectation des broches d'entrées sorties

5.4.5 Visualisation des résultats sur l'osciloscope

Pour vérifier le fonctionnement de la commande on la visualise pour plusieurs valeurs de m, f et r (figures 5.21, 5.22 et 5.23).

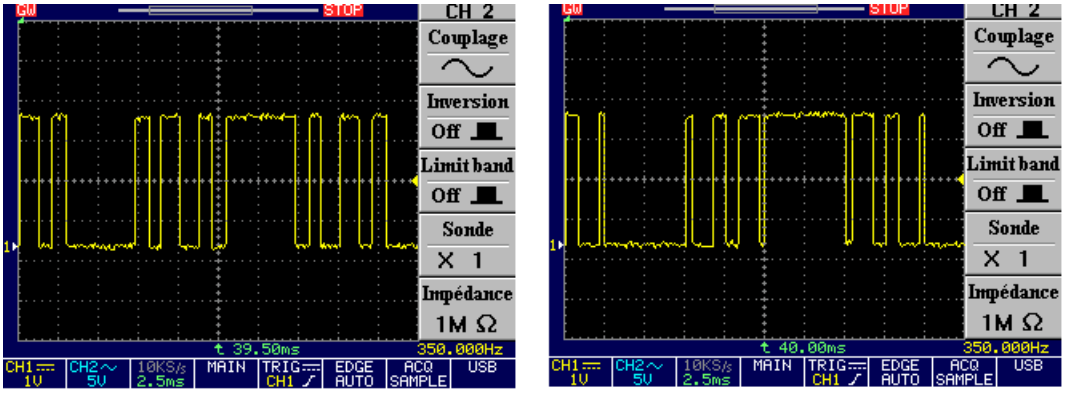


FIGURE 5.21 – Commande MLI calculée pour $m = 3, \quad f_r = 50Hz$ et $r_1 = 0.6; \quad r_2 = 1$

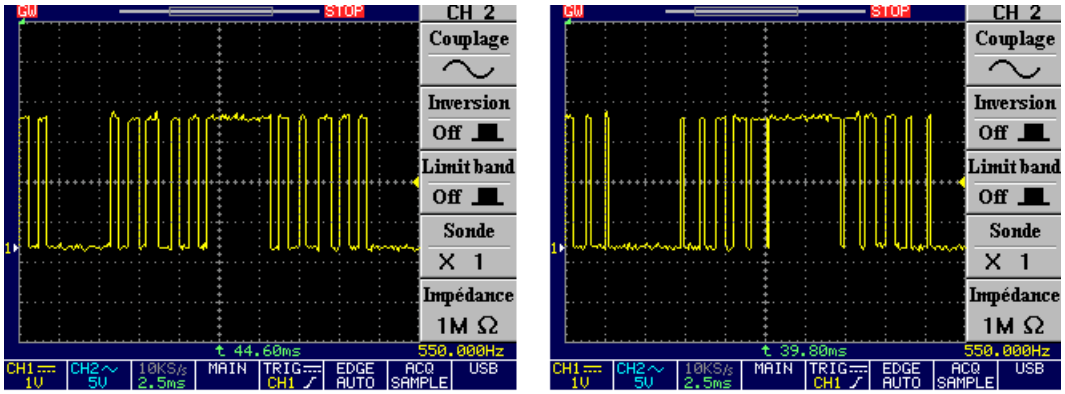


FIGURE 5.22 – Commande MLI calculée pour $m = 5, \quad f_r = 50Hz$ et $r_1 = 0.6; \quad r_2 = 1$

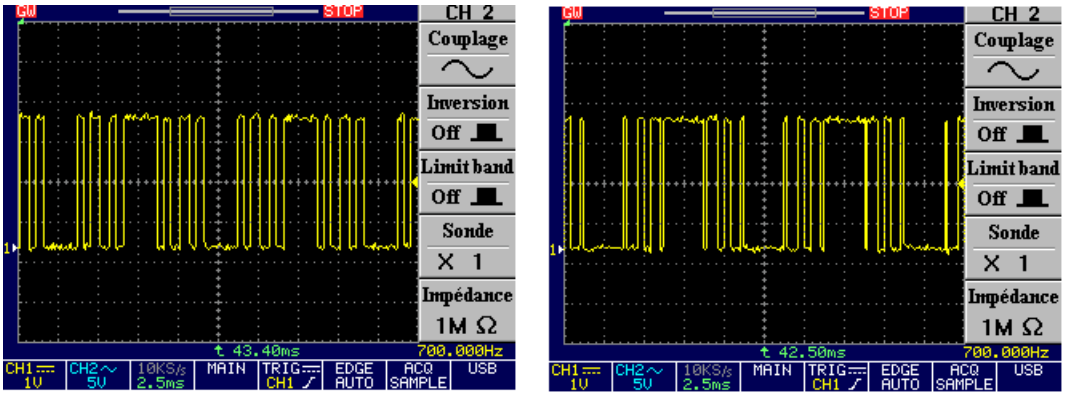


FIGURE 5.23 – Commande MLI calculée pour $m = 3, \quad f_r = 100Hz$ et $r_1 = 0.6; \quad r_2 = 1$

5.5 Implémentation d'une commande MLI calculée 'on-line'

5.5.1 Introduction

Dans cette section on va essayer d'améliorer le programme précédent ; donc on va implémenter l'algorithme on-line qui a été développé dans le chapitre 2 pour un calcul en temps réel des angles de commutation. 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 signaux de commande pour les transistors en utilisant des fonctions qui réalisent certaines opérations arithmétiques comme la division et la puissance.

5.5.2 Programme de calcul des angles de commutation

Dans le chapitre 2, nous avons développé les formules pour calculer les valeurs approximée des angles de commutation pour k impair et k pair (les équations 2.11, 2.12).

Dans le langage VHDL il n'existe pas une bibliothèque standard qui définit et compile la division et la puissance. Donc on réalise des sous programmes qui font ces opérations, et comme on travaille avec les nombres entiers puisque les nombres réels ne sont pas prédéfinis, on multiple l'échelle de travail par 10^6 pour l'augmentation de la précision.

La figure 5.24 montre l'organigramme pour le calcul des angles de commutations.

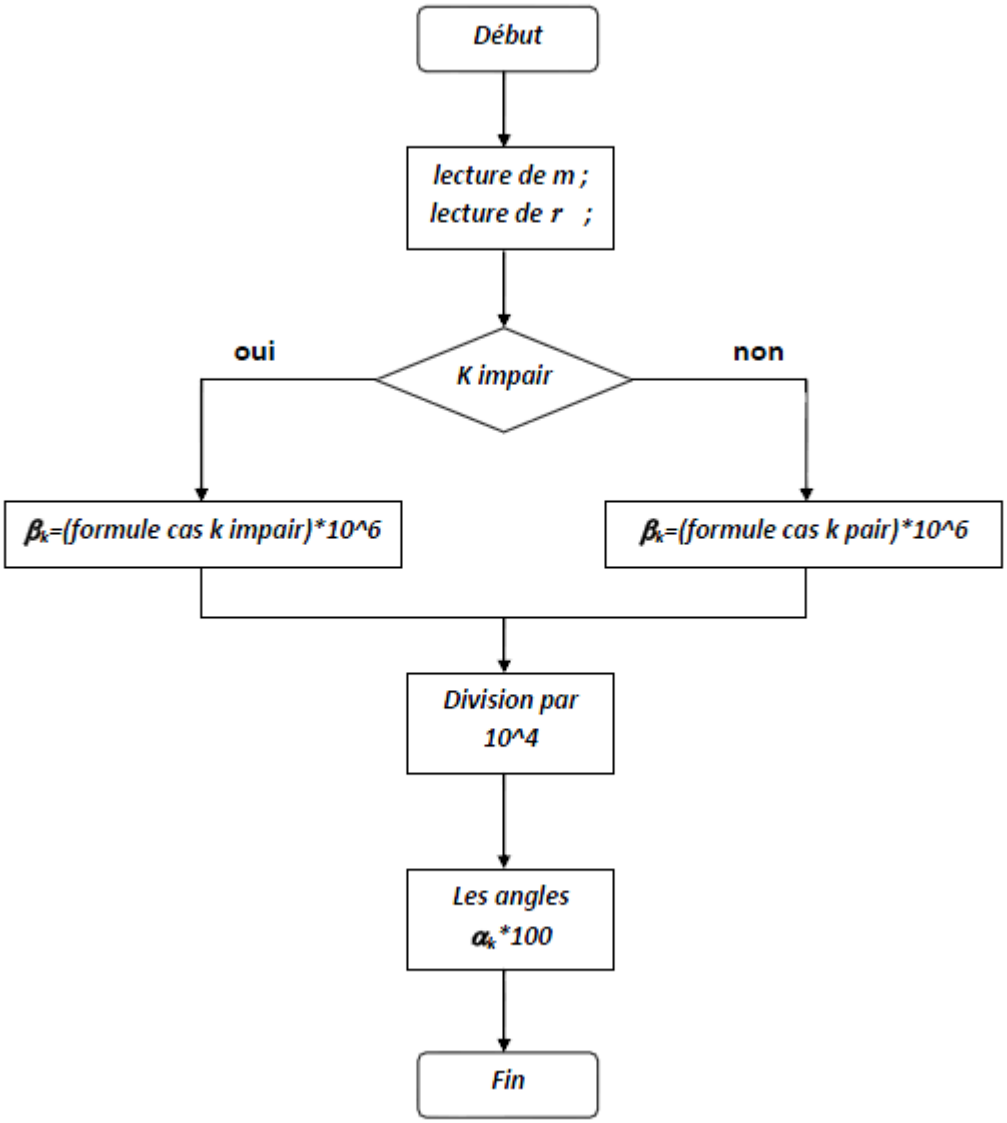


FIGURE 5.24 – Organigramme de calcul des angles de commutations

Pour le calcul des formules des angles α_k dans le cas k pair ou impair, on a besoin de l'appel des deux fonctions de division et de puissance. La figure 5.25 montre les organigrammes de ces fonctions.

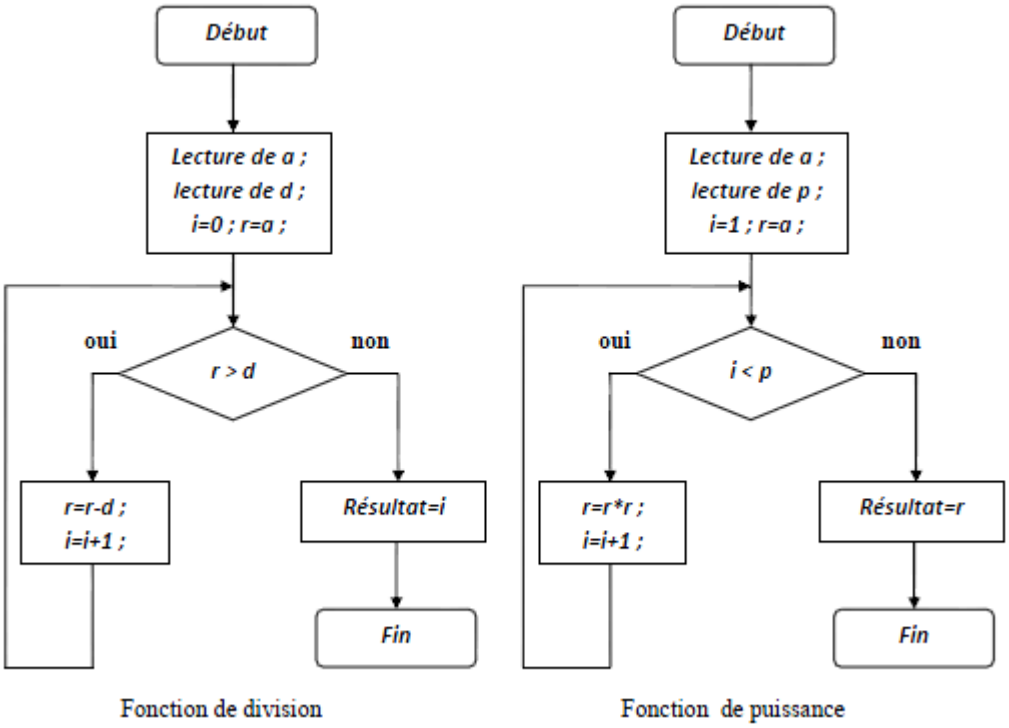


FIGURE 5.25 – Organigrammes des fonctions de division et de puissance

5.5.3 Résultats de simulation

Pour valider le programme, on calcule les valeurs approximées des angles de commutations pour plusieurs valeurs de r et on les compare avec les valeurs exactes. Le tableau 5.4 donne les angles trouvés par le programme pour $m = 5$.

r	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
α_1	19.08	18.15	17.23	16.31	15.39	14.46	13.54	12.62	11.70
α_2	20.41	20.83	21.25	21.69	22.08	22.40	22.92	23.34	23.75
α_3	38.99	37.98	36.98	35.97	34.97	33.96	32.95	31.95	30.94
α_4	40.73	41.46	42.19	42.93	43.66	44.32	45.12	45.86	46.59
α_5	59.08	58.15	57.23	56.31	55.39	54.46	53.54	52.62	51.70

TABLE 5.4 – Les angles de commutation approximés pour $m = 5$

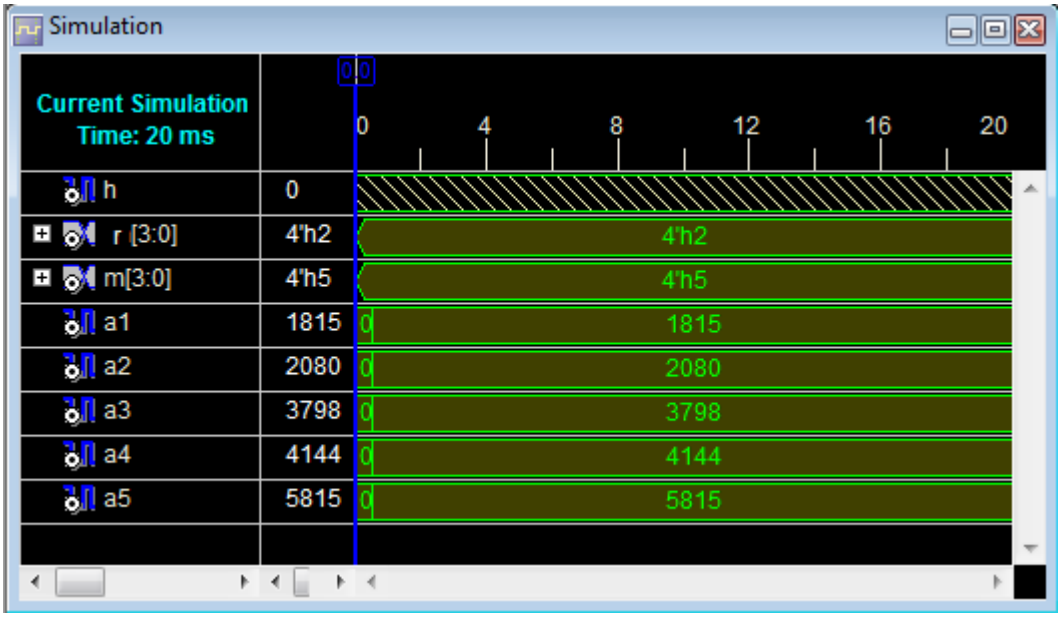


FIGURE 5.26 – Les angles de commutation pour $m = 5$ et $r = 0.2$

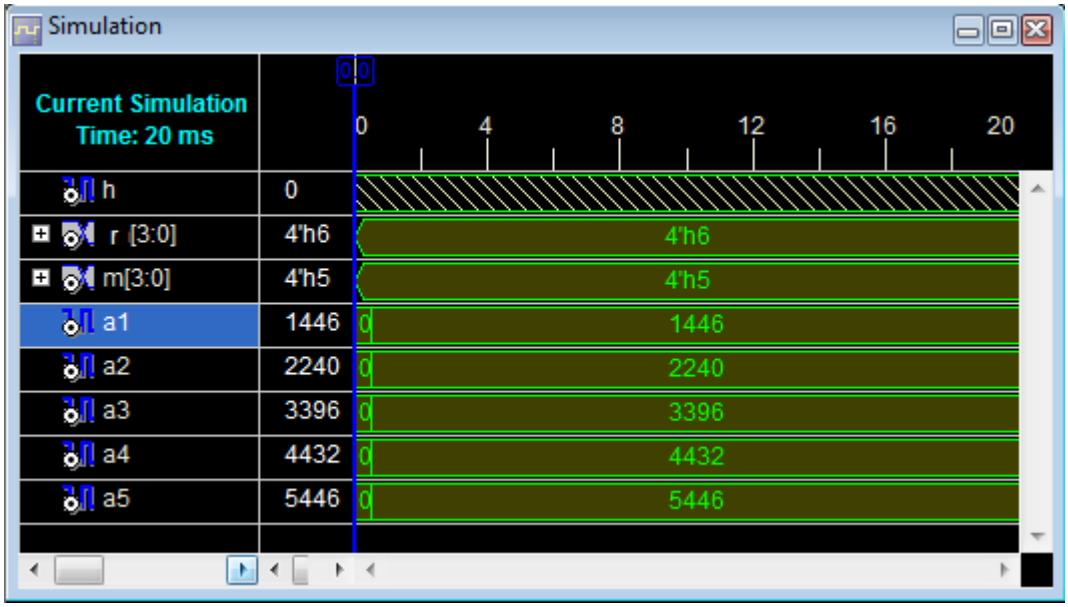


FIGURE 5.27 – Les angles de commutation pour $m = 5$ et $r = 0.6$

D'après les résultats obtenus et par comparaison, on voit que les valeurs des angles de commutations (approximées et exactes) sont très proches. Ce qui valide le programme. Pour augmenter la precision des résultats , il faut augmenter la precision de calcul de la division et de la puissance. Mais, nous sommes limités par la taille du résultat, parce que il faut que le résultat soit sur 32 bits au maximum.

5.5.4 Programme de génération des signaux de commande

Après le calcul des angles de commutation, nous traitons maintenant le calcul des instants de commutation à partir des angles pour obtenir à la sortie de l'onduleur un signal sinusoïdal à une fréquence donnée. Pour cela on utilise un diviseur de fréquence avec un coefficient variable et un compteur pour la génération de la commande. La figure 5.28 donne l'organigramme du programme.

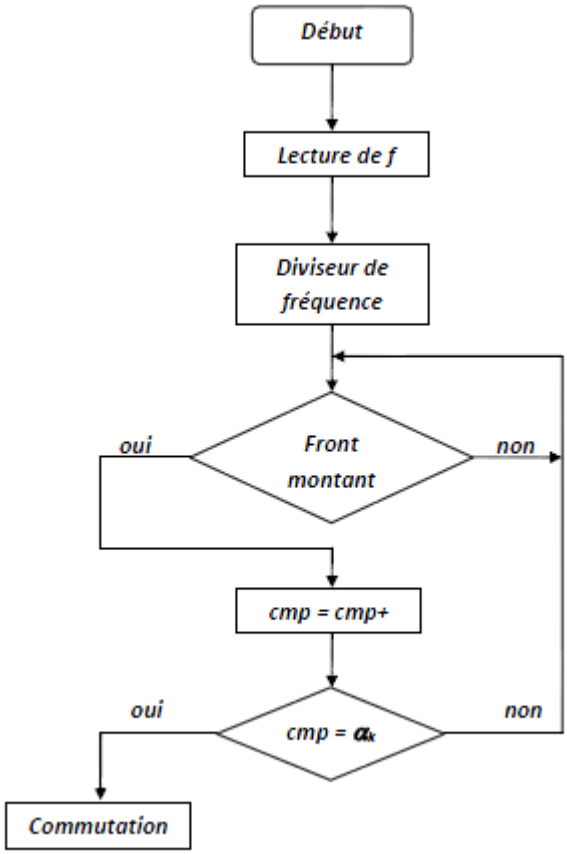


FIGURE 5.28 – Organigramme de génération des signaux de commande

La fréquence de découpage obtenue dépend du pas et de la fréquence de comptage calculée par le diviseur de fréquence et la précision prise pour les angles de commutation. Pour notre cas on a pris la précision de deux chiffres après la virgule.

$$f_s = \frac{f_c \times pas}{36000}$$

Avec f_s : la fréquence de signal à la sortie de l'onduleur.

f_c : la fréquence de comptage.

5.5.5 Résultats de simulation

Les résultats de simulation par Xilinx ISE 9.2i sont montrés par les figures 5.29, 5.30 et 5.31.

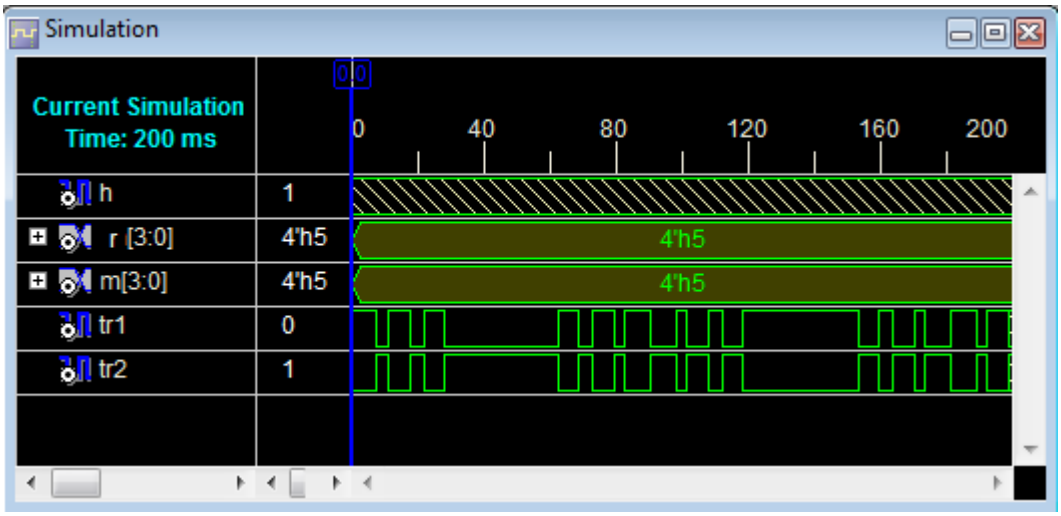


FIGURE 5.29 – Résultat de simulation pour $m = 5$ et $r = 0.5$

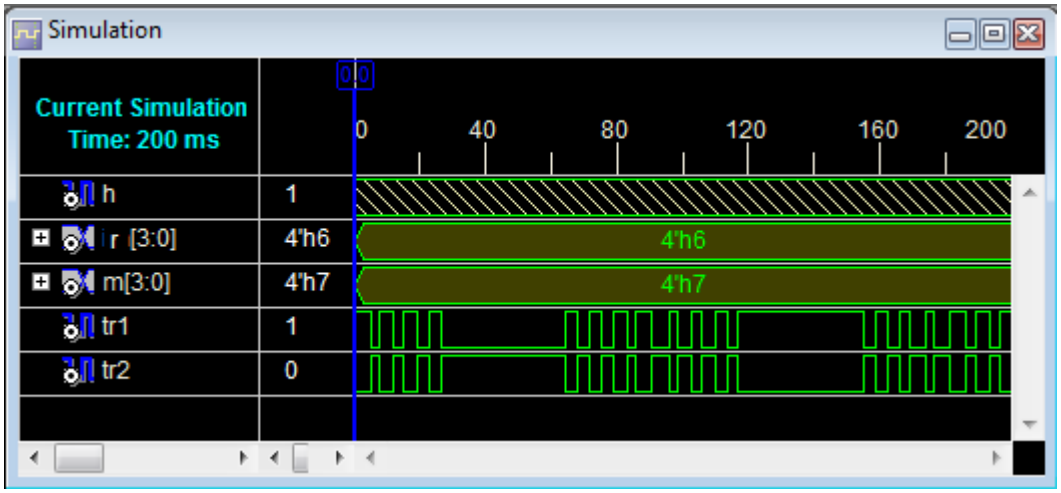


FIGURE 5.30 – Résultat de simulation pour $m = 7$ et $r = 0.6$

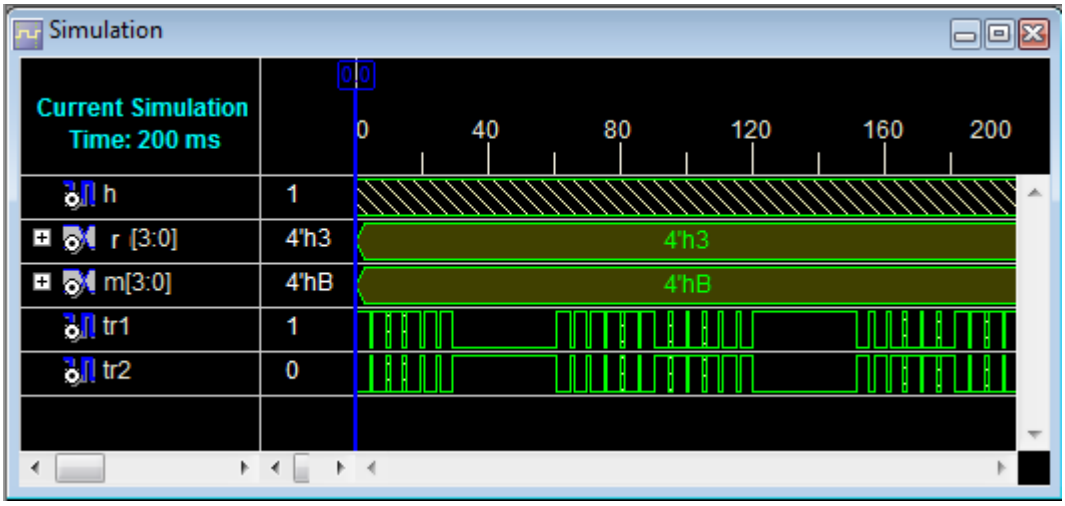


FIGURE 5.31 – Résultat de simulation pour $m = 11$ et $r = 0.3$

5.5.6 Implémentation de la commande

Comme pour le cas engendrée et off-line ; pour implémenter la commande sur la carte, on choisit la fréquence de l'horloge de $24MHz$; dans ce cas les entrées sont le taux de modulation r codé sur 4 bits, et le nombre de commutations m codé sur 4 bits.

Le tableau 5.5 résume les affectations des broches entrée sortie.

L'entrée/sortie	h	r(3)	r(2)	r(1)	r(0)	m(3)	m(2)	m(1)	m(0)	tr
Pin	a11	b4	a4	c4	c5	b5	a5	d6	c6	f3

TABLE 5.5 – Affectation des broches d'entrées sorties

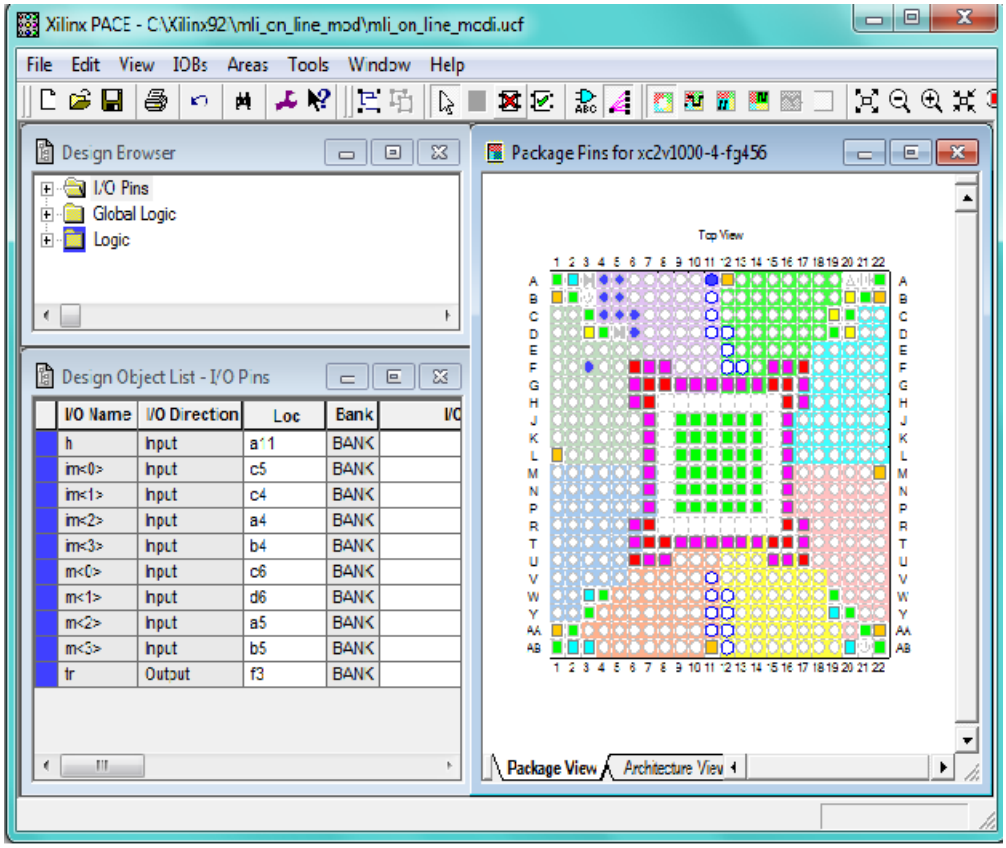


FIGURE 5.32 – Aperçu de l'outil d'affectation des broches d'entrées sorties

5.5.7 Visualisation des résultats sur l'osciloscope

Pour vérifier le fonctionnement de la commande on la visualise pour plusieurs valeurs de m et r (figures 5.33 et 5.34).

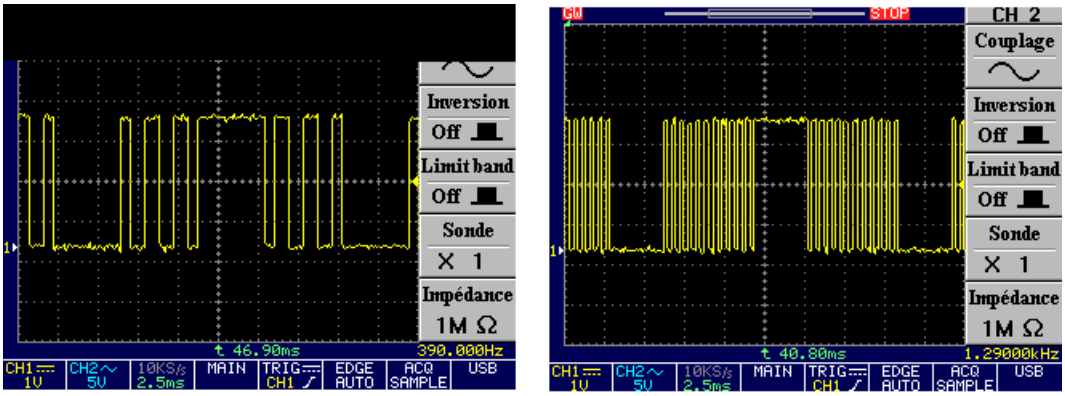


FIGURE 5.33 – Résultat d'implémentation pour $m = 3, \quad 11$ et $r = 0.8$

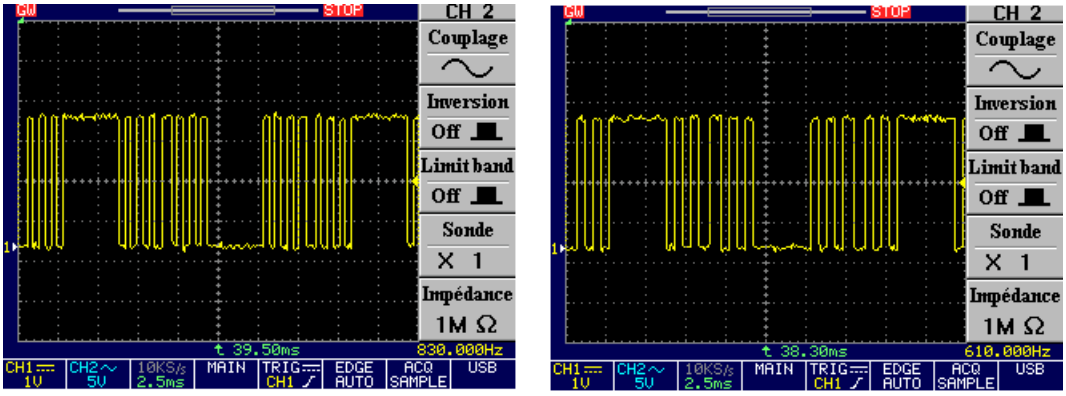


FIGURE 5.34 – Résultat d'implémentation pour $m = 5, \quad 7$ et $r = 0.6$

5.5.8 Étude du spectre

L'objectif de cette partie est l'étude du spectre de la tension de sortie du bras de l'onduleur en demi-pont de la figure 5.12 en comparant les résultats expérimentaux avec ceux obtenus par simulation.

1. **Résultats de simulation**

Pour $r = 0.8$, on a choisi 5 angles de commutation, on est dans la fréquence de $50Hz$. Les commandes des deux transistors sont données par la figure 5.35.

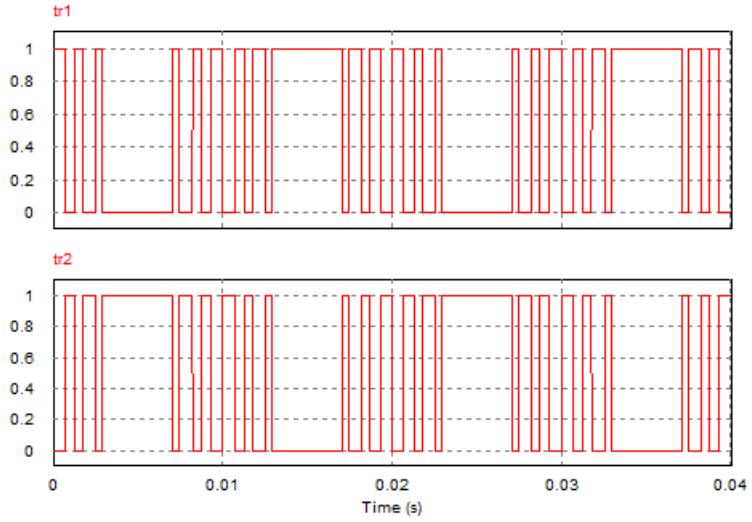


FIGURE 5.35 – Commandes des transistors

Comme il est clair sur la figure 5.35, le signal de commande présente 5 commutations par quart de période, avec une double symétrie par rapport au quart et à la demi-période.

Pour vérifier l'élimination des harmoniques désirés, nous avons tracé le spectre de la tension de sortie figure 5.36.

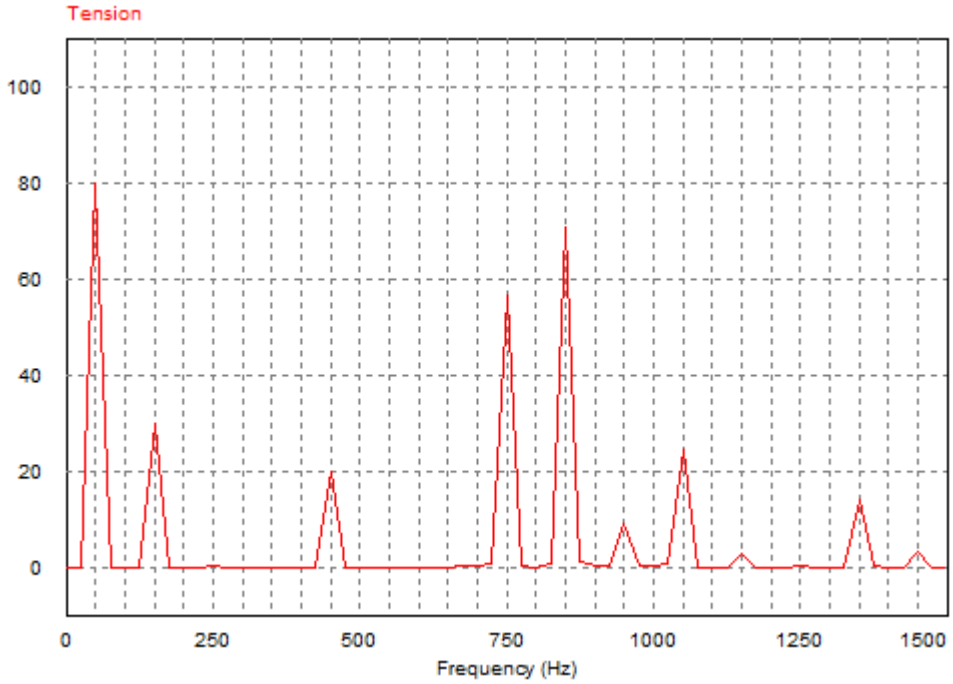


FIGURE 5.36 – Spectre de la tension de sortie

On remarque bien l'élimination des harmoniques 5,7,11 et 13, le premier harmonique non éliminé est celui de rang 17 (850 Hz). On remarque aussi la présence de l'harmonique trois et ces multiples car notre onduleur n'est pas triphasé. Les figures 5.37 et 5.38 donnent les résultats de simulation pour un onduleur triphasé avec $m = 7$ et $r = 0.5$.

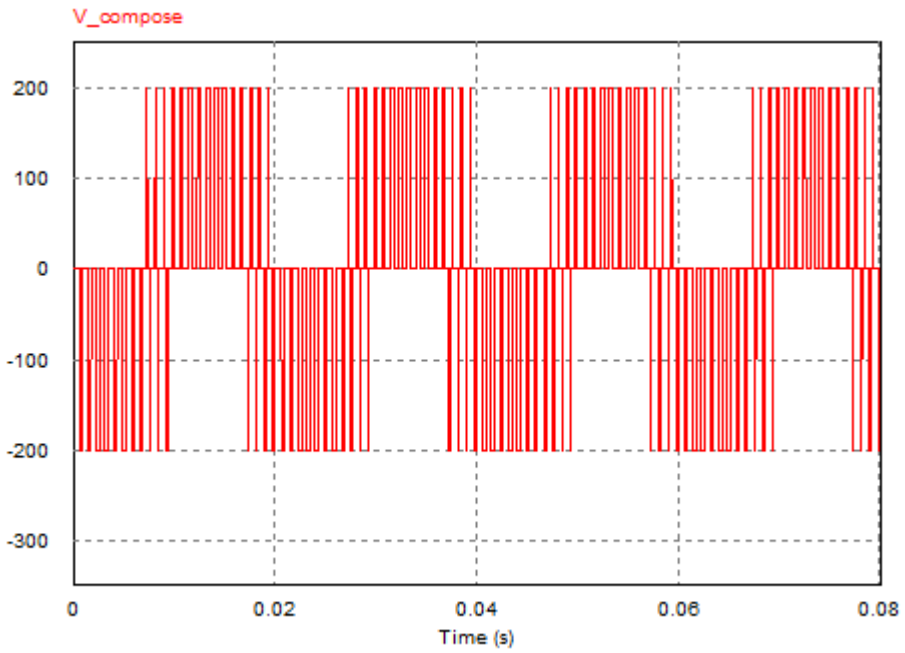


FIGURE 5.37 – Tension composée d'un onduleur triphasé $m = 7$ et $r = 0.5$

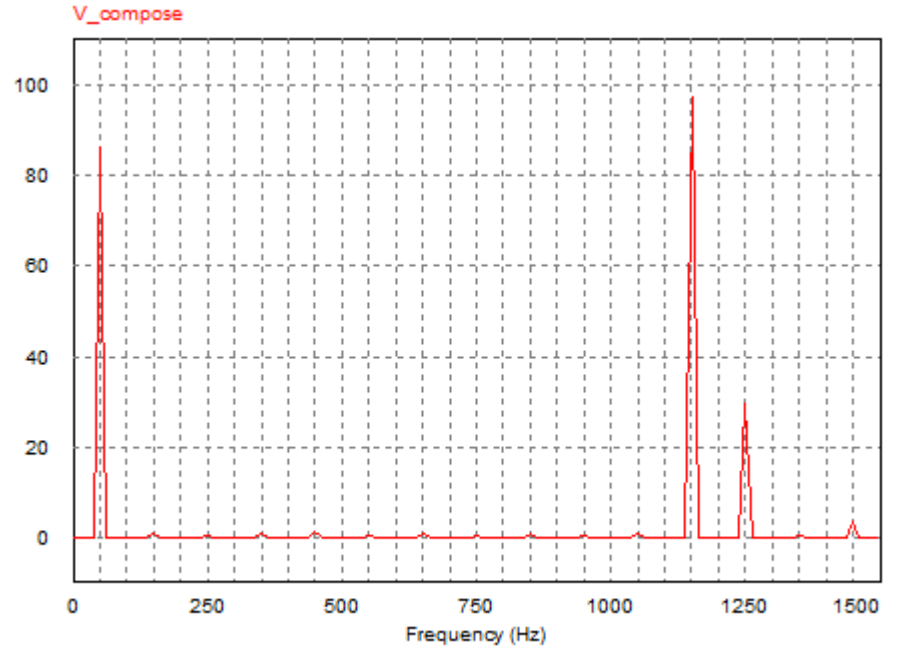


FIGURE 5.38 – Spectre de la tension composée d'un onduleur triphasé $m = 7$ et $r = 0.5$

2. **Résultats expérimentaux**

Comme dans le cas de la commande engendrée, nous avons appliqué la commande 'on-line' sur un bras d'onduleur. La tension de sortie de l'onduleur pour $r = 0.8$ et $m = 5$ est donnée par la figure 5.39.

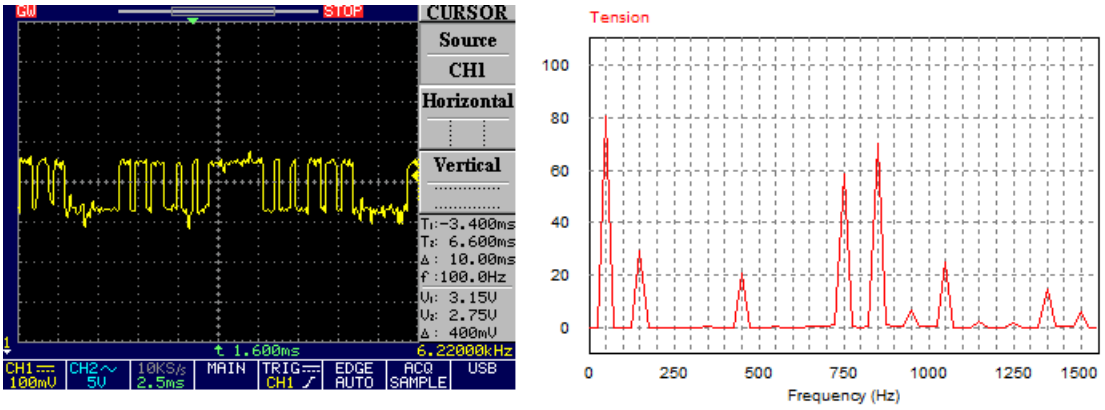


FIGURE 5.39 – Tension et spectre de sortie du bras de l'onduleur pour $m = 5$ et $r = 0.8$

3. **Interprétation des résultats**

D'après l'allure du spectre représentée dans la figure 5.39, on voit la suppression des cinq premiers harmoniques non multiples de trois. et que le premier harmonique non éliminé c'est celui de rang 17 (850 Hz).

La forme d'onde du signal de commande est très proche des résultats théoriques et on remarque bien qu'on a m commutations par quart de période.

Le contenu spectral du signal de sortie de l'onduleur et pratiquement le même que la méthode de Patel et Hoft où les harmoniques pairs et d'ordre trois sont éliminés et $m - 1$ harmoniques impairs non multiples de trois sont encore éliminés.

L'amplitude du fondamental est pratiquement égale au taux de modulation r multiplié par $E/2$, où E est la tension d'alimentation de l'onduleur.

5.5.9 Conclusion

En comparant les résultats obtenus dans la pratique avec ceux obtenus par simulation on peut dire que l'algorithme implémenté donne de très bons résultats pour l'élimination des harmoniques.

Cette stratégie de commande permet d'augmenter la fréquence des premiers harmoniques et donc facilite le filtrage. Il est facile de faire varier l'amplitude du fondamental. Cette méthode est surtout utilisée dans les variateurs des vitesses des machines tournantes.

5.6 Conclusion

Dans ce chapitre nous avons implémente deux techniques MLI, on a commencé par la technique MLI engendrée, ensuite nous avons implémenté la technique MLI calculée, dans un premier temps nous avons implémenté un programme off-line. En vue de l'insuffisance de ce programme, nous l'avons amélioré par un programme on-line.

Dans les deux commandes MLI, les simulations des programmes effectuées donnent de bons résultats. De même l'algorithme MLI on-line a été correctement implémenté sur le circuit FPGA. Les résultats obtenus en pratique sont acceptables en comparaison avec ceux obtenus en simulation.

Conclusion Générale

Le but de notre étude est d'implémenter deux commandes MLI sur un circuit FPGA. la stratégie MLI triangulo-sinusoïdale et MLI calculée avec élimination d'harmoniques sélective et asservissement du fondamental pour un onduleur triphasé.

L'objectif commun de toutes les modulations de largeur d'impulsion est d'offrir un contrôle continu (en valeur moyenne) d'un système à découpage fondamentalement discontinu. Lorsque l'on travaille en faible puissance on peut facilement effectuer le découpage à des fréquences suffisamment élevées, une modulation engendré est suffisante pour éloigner les harmoniques vers des fréquences très élevées. Ils sont généralement filtrés par la charge alimentée par l'onduleur qui a un comportement de type filtre passe bas. Mais à des puissances élevées cette solution n'est plus envisageable. La fréquence de découpage est alors déterminée à la suite d'un compromis entre les pertes par commutation dans les composants électronique et les perturbations engendrées par le découpage. Pour permettre d'assurer un tell compromis de la manière la plus satisfaisante possible, on étudie la stratégie de modulation de largeur d'impulsion calculée basée sur le développement de Patel et Hoft.

Mais l'utilisation d'une telle alimentation en fonctionnement temps réel (on-line) 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 pra-

tiquement le même que celui de l'algorithme de Patel et Hoft.

Lors du passage à l'implémentation nous avons choisi d'utiliser des architectures reconfigurables FPGA, qui donnent de bonnes performances pour les algorithmes avec des contraintes temporelles. L'absence des bibliothèques traitants les nombres réels et les fonctions mathématiques élémentaire tel que les fonctions sinus, division et la puissance nous a conduits à faire quelques approximations, malgré cela, les résultats obtenus son acceptables.

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. Ainsi que la carte de développement VIRTEX-II de MEMEC nous a permis d'implémenter notre application et de vérifier son fonctionnement.

En perspectives, nous proposons l'amélioration de la précision de cet algorithme en lui ajoutant des bibliothèques traitant les nombres en virgule flottante. Et développer des algorithmes on-line pour des onduleurs multi-niveaux pour améliorer la qualité de signal de sortie et faciliter l'implémentation.

Bibliographie

[1] M. Pinard. *"Convertisseurs et électronique de puissance"*.Edition DUNOD, 2007.

[2] B. Grabowski, C. Ripoll et Coll. *"Aide-mémoire Electronique"*. 5e Edition DUNOD, 2008.

[3] G. Chateigner et M. Boes. *"Manuel de génie électrique"*. Edition DUNOD, 2007.

[4] Guy Séguier, *"Électronique de puissance"*, 7e Edition DUNOD, 1999.

[5] G. Séguier, F. Labrique, *"Les convertisseurs de l'électronique de puissance"*, Volume 4, Lavoisier 1989.

[6] Eric. Monmasson, *"Commande rapprochée de convertisseur statique 1"*, Lavoisier, 2009.

[7] J.C. Bouisson. *"Processeurs concevoir son microprocesseur"*. Edition ELLIPSES.

[8] V. Pedroni. *"Digital electronics and design with VHDL"*. Edition ELSEVIER, 2008.

[9] R. Woods et J.Mc. Allister. *"FPGA based implementation of signal processing systems"*. Edition WILEY, 2008.

[10] S.Kilts. *"Advanced FPGA design"*. Edition WILEY, 2007.

[11] P.Nouel. *"langage VHDL et conception des circuits"*. Ecole Nationale Supérieure d'Electronique Informatique Radiocommunication de Bordeaux(ENSEIRB), France, 2003.

[12] J. Weber et M. Meaudre. *"Le langage VHDL cours et exercices"*. 2e Edition DUNOD, 2003.

[13] 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.

[14] 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.

[15] Z .Salam S.I.Safie L.J.Yiap, *"An On-Line Harmonics Elimination PWM Scheme for Three-phase Voltage Source Inverters"*, Fifth International Conference Power Electronics and Drive Systems IEEE PEDS, Vol.1, 17-20 Nov 2003.

[16] Z.Salam and C.T.Lynn, *"Algorithm for Near Optimal Harmonics Elimination PWM Based on Quadratic Approximation Method"* , IECON 02, Seville, Spain, Nov 2002.

[17] H.Foch, F.Forest et T.Meynard,*"Onduleurs de tension. Structures. Principes. Applications"*. Techniques de l'ingénieur, vol. D31, N^o D3 176, Novembre 1998.

[18] H.Foch, F.Forest et T.Meynard,*"Onduleurs de tension. Mise on œuvre"*. Techniques de l'ingénieur, vol. D31, N^o D3 177, Aout 2000.

[19] E.Monmasson, L.Idkhajine, I.Bahri, M-W-Naouar, L.Charaabi, *"Design methodology and FPGA-based controllers for Power Electronics and drive applications"*, 2010 5th IEEE Conference on Industrial Electronics and Applicationsis.

[20] L.Idkhajine, E.Monmasson, *"Design Methodology for Complex FPGA-based Controllers - Application to an EKF Sensorless AC Drive"*, IEEE ICEM 2010. Rome.

[21] F. Pereira, L. Gomes, L. Redondo, *"FPGA Controller for Power Converters with integrated Oscilloscope and Graphical User Interface"*, IEEE Proceedings of the 2011 International Conference on Power Engineering, Energy and Electrical Drives.

[22] V. Parkhi, S. Shilaskar, M. Tirmare and M. Jog, *"FPGA Implementation of PWM Control Technique for Three Phase Induction Motor Drive"*, 2008 IEEE DOI 10.1109/ICETET.2008.115.

[22] A.Das, K. Sivakumar, R.Ramchand, C.Patel, and K. Gopakumar, *"A Pulsewidth Modulated Control of Induction Motor Drive Using Multilevel 12-Sided Polygonal Voltage Space Vectors"*, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 56, No. 7, JULY 2009.

[23] A. Guellal, *"Implémentation d'une technique MLI en temps réel sur un circuit FPGA"*. Mémoire d'ingénieur en Electronique, Ecole Nationale Polytechnique, Alger, Algérie, 2007.

[24] C. Benboucham. *"contribution a l'implémentation des réseaux de neurones artificiels sur hardware reconfigurable FPGA"*. thèse de doctorat en Automatique, Ecole Nationale Polytechnique, Alger, Algérie, 2008.

[25] F. Chkired. *"Etude et implémentation d'une commande MPPT neuro-floue sur FPGA"*. thèse de Magistère en Electronique, Ecole Nationale Polytechnique, Alger, Algérie, 2008.

[26] M. Keita, *"Techniques de commande des convertisseurs"*, thèse Master École de Technologie Supérieure du Québec, Canada 1999.

[27] D. Bendib, *"Etude et réalisation d'une commande MLI on-line sur circuit FPGA"*, thèse de Magister en Électronique, Ecole Nationale Polytechnique, Alger, Algérie, 2009.

[28] A. Quarteroni, R. Sacco, F. Saleri. *"Méthodes Numériques"*. Édition Springer, 2007.

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

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

Annexe A : Programmation des systèmes automatisés par les circuits FPGA

Dans cette section nous allons voir comment utiliser les cartes FPGA dans les systèmes automatisés. Pour cela nous allons faire l'étude sur un chariot avec retour qui est un système très utilisé dans l'industrie.

Description du système

Le chariot est commandé par deux moteurs $M1$ et $M2$, a, b et c sont des capteurs de presence.

En appuyant sur un bouton poussoir m, et si le chariot est en a, on demarre le moteur $M1$ jusqu'au point c en passant par le point b.

Arrivé au point c, on lance une temporisation T de $1min$, ensuite on rebrousse le chemin avec $M2$ jusqu'au a.

Le voyant Va est allumé quand le chariot se trouve au point a, le voyant Vb s'allume au point b et le voyant Vc s'allume au point c.

Le grafcet du système est donné par la figure 40.

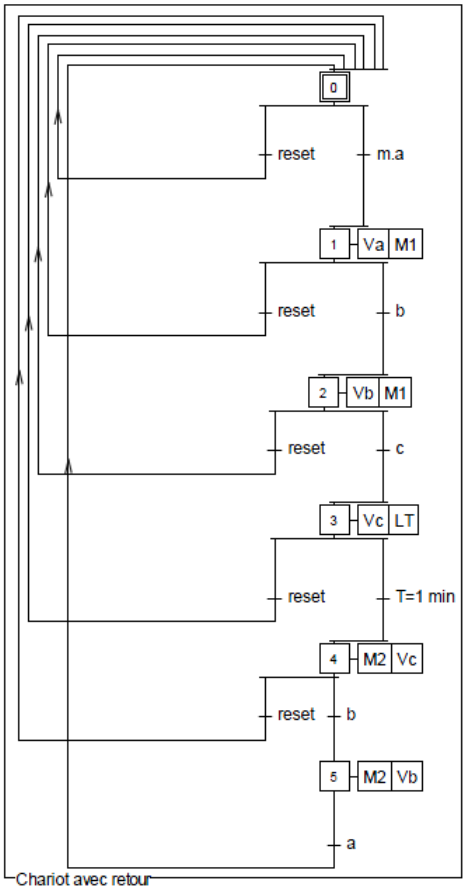


FIGURE 40 – Grafcet d'un chariot aller-retour

Les équations des sorties de notre système sont :

$$M1 = X1 + X2$$
$$M2 = X4 + X5$$
$$Va = X1$$
$$Vb = X2 + X5$$
$$Vc = X3 + X4$$

Programme

Le logiciel Xilinx ISE permet l'édition des programmes sous forme d'un schéma. Donc dans ce cas nous utilisons un schéma pour la programmation en utilisant les equations et la réalisation RS du système.

Résultats de simulation

Le résultat de simulation du programme par Xilinx 9.2i sont donnée par la figure 41.

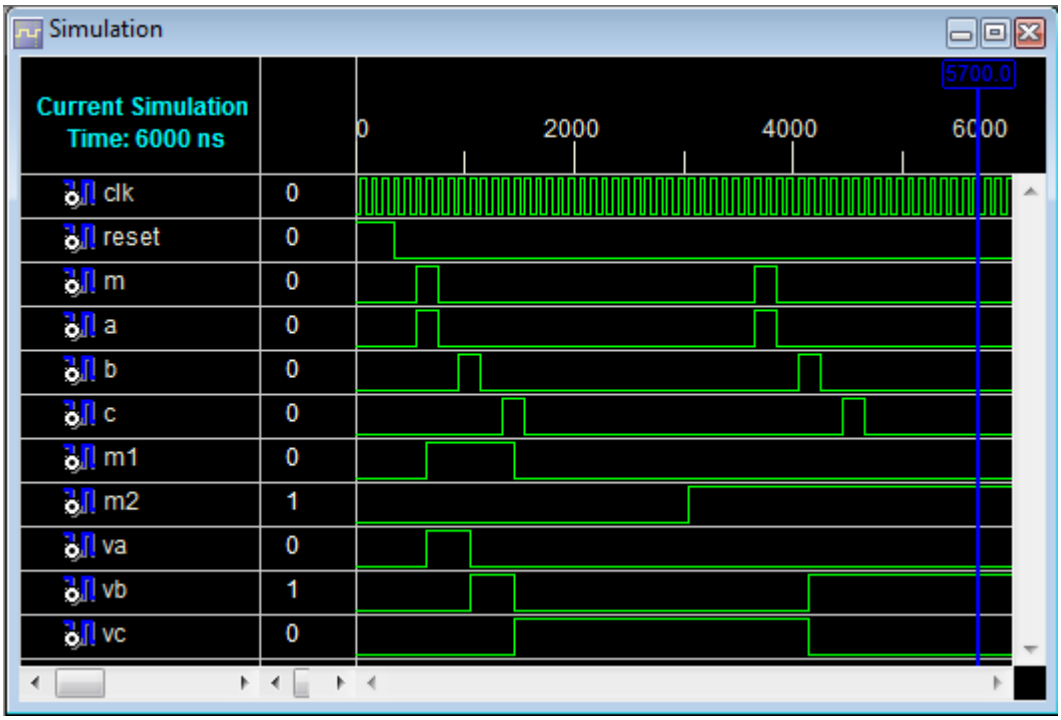


FIGURE 41 – Programme d'un chariot aller-retour

Annexe B : Utilisation du logiciel Xilinx ISE

Introduction

Il s'agit d'un logiciel de programmation des produits Xilinx (CPLD , FPGA spartan et virtex,...).Il permet :

- la saisie de projets d'implantation (sous forme de schéma logique, de machine d'états ou en langage VHDL, verilog, ABEL,...).
- la simulation du fonctionnement d'un projet d'implantation.
- la transcription en fichier JDEC et l'implantation sur un produit Xilinx.

Généralités

Démarrer le logiciel Xilinx

Pour démarrer Xilinx double-clic sur le navigateur :

On peut démarrer Xilinx a partir du menu Démarrer par : **Démarrer** → **Tous les programmes** → **Xilinx ISE 9.2i** → **Project Navigator.**

Help de Xilinx

Pour ouvrir le Help, on fait l'une ou l'autre de ce qui suit :

– On presse **F1** pour obtenir l'aide sur un outil spécifique ou sur une fonction qu'on a choisi.

– On Lance le **ISE Help Contents** du menu de **Help**.

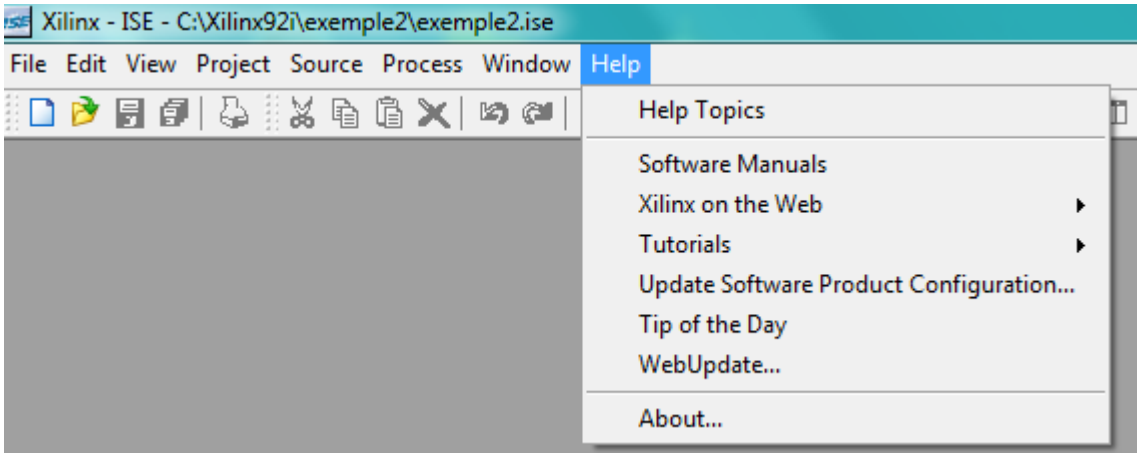


FIGURE 42 – Help de ISE

Création d'un nouveau projet

Pour créer un nouveau projet il faut suivre les étapes suivantes :

1. On choisit **File** → **New Project...** le **New Project Wizard** apparaît.
2. On tape le nom du projet dans la case **Project Name**.
3. On vérifie que le type **HDL** est sélectionné dans la liste de **Top-Level Source Type**.
4. On clique **Next** pour se déplacer à la page de propriétés de dispositif **Device Properties**.
5. On complète les propriétés de notre dispositif.
6. Quand la table est complète, nos propriétés de projet ressembleront à ce qui suit :

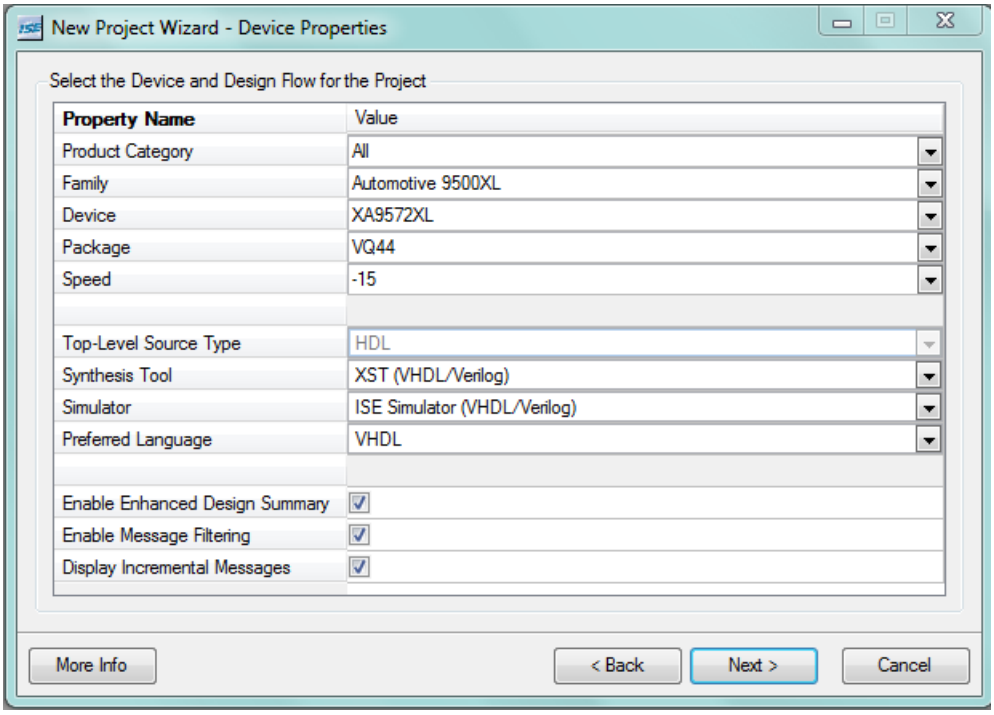


FIGURE 43 – Propriétés de dispositif de projet

7. Cliquer **Next** pour terminer la création de votre projet.

Création d'un nouveau fichier source VHDL

Pour créer un nouveau fichier source on doit suivre les étapes suivants :

1. On clique le bouton **New Source** dans la fenêtre **New Project Wizard**.
2. On choisit **VHDL Module** comme type de source.
3. On saisit le nom de fichier (par exemple :compt).
4. On vérifie que la case **Add to project** est cochée.
5. On clique **Next**.
6. On déclare les entrées et les sorties du compteur en complétant l'information gauche comme montré ci-dessous :

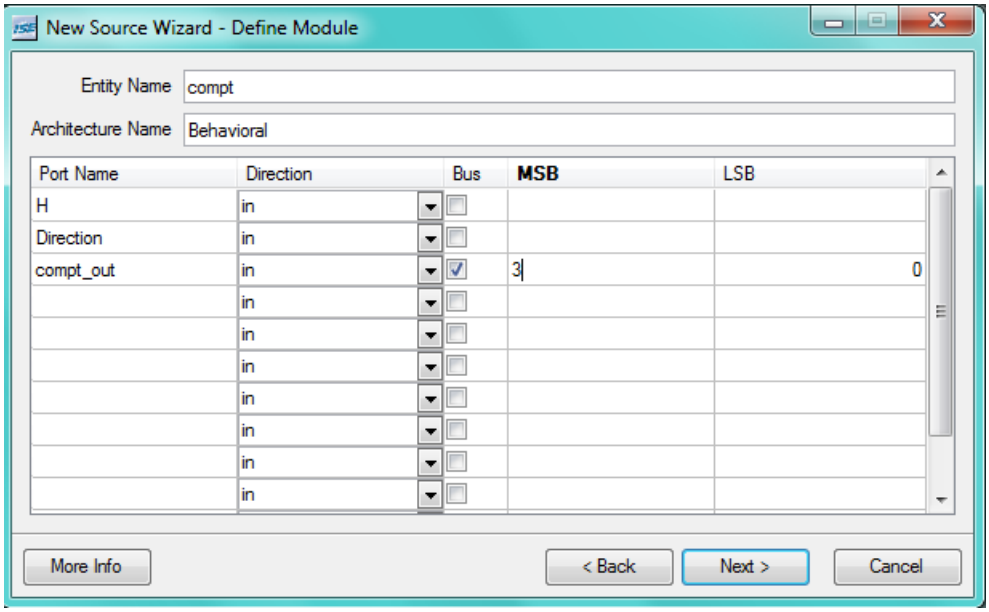


FIGURE 44 – Définition du module

7. On clique **Next**, puis **Finish** dans la nouvelle fenetre **New Source Wizard - Summary**.

8. On clique **Next**, puis **Next**, puis **Finish**.

Le fichier source contenant les affichages de paires d'entité/architecture dans Workspace est comme montré ci-dessous :

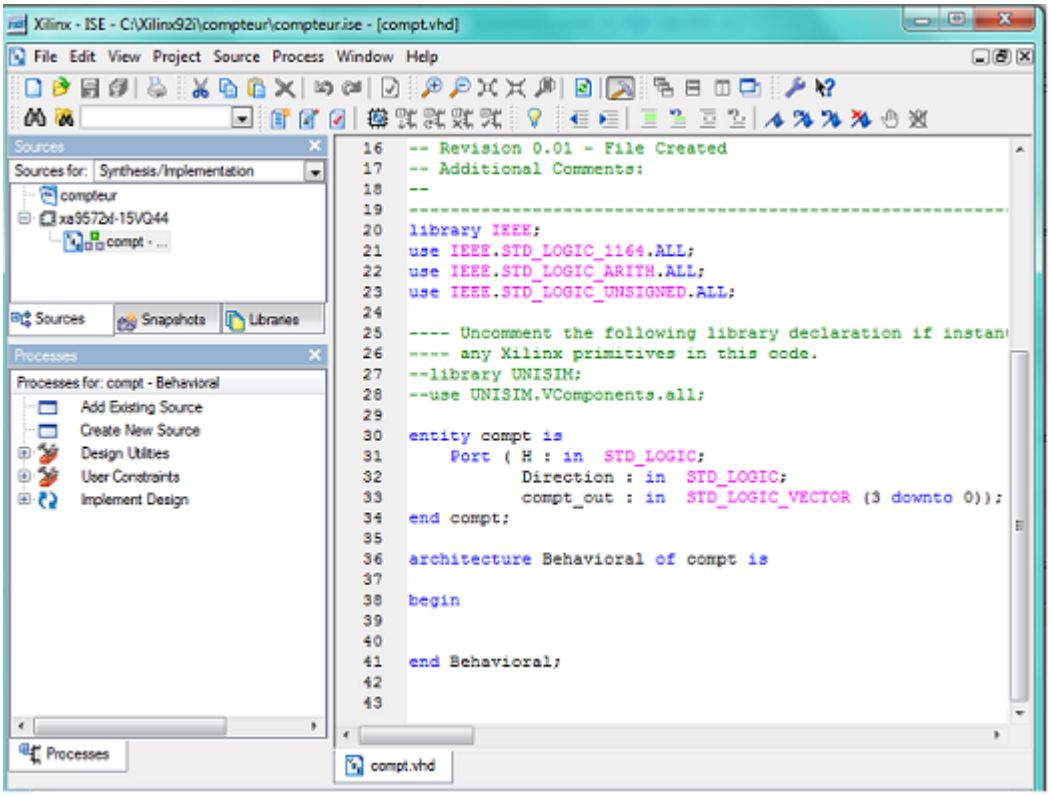


FIGURE 45 – Nouveau projet dans ISE

Utilisation de Language Templates

La prochaine étape, est d'ajouter la description comportementale du compteur. Pour faire ceci on peut utiliser un code prédéfinit par l'ISE pour un simple compteur.

1. On place le curseur juste au-dessous du **Begin** de l'architecture du compteur.
2. On choisit **Edit** → **Language Templates..**.
3. Double-clique sur : **VHDL** → **Synthesis Constructs** → **Coding Examples** → **Counters** → **Binary** → **Up/Down Counters** → **Simple Counter**.
4. On choisit **Edit** → **Use in File**.
5. On ferme la fenêtre **Language Template**.

Édition finale du fichier source VHDL

1. On ajoute la déclaration suivante du signal pour assurer le retour du compteur vers zero, après **Architecture** et avant **Begin**.
   signal count_int : **std_logic_vector**(3 **downto** 0) := "0000" ;
2. On adapte le fichier source du compteur en remplaçant les noms des signaux d'entrée et de sortie par les noms utilisés dans le code comme suit :
   – On remplace toutes les occurrences de "CLOCK" par "H".
   – On remplace toutes les occurrences de "count_direction" par "Direction".
   – On remplace toutes les occurrences de "count" par "count_int".
3. On ajoute la ligne suivante après **end process** :COUNT_OUT <= count_int ;
4. On sauvegarde le fichier par : **File** → **save**.

Vérification des erreurs

Quand les fichiers sources sont complets, on vérifie la syntaxe de la conception pour trouver les erreurs.

1. On vérifie que **Behavioral Simulation** est choisie parmi la liste dans la fenêtre **sources**.
2. On sélectionne **compt.vhd**.
3. On clique sur le symbol '+' de **Xilinx ISE simulator** dans la fenêtre **Processes**.
4. Double-clic sur **Check Syntax**. S'il y a des erreurs il faut les corriger.
5. On ferme le fichier *compt.vhd*

Simulation

vérification de la fonctionnalité

Avant la simulation, on crée le **test bench wavform** pour varier les entrées du compteur. Pour faire ceci :

1. On sélectionne le fichier *compt.vhd* dans la fenêtre des sources.
2. On crée un nouveau source Test Bench par : **Project** $\rightarrow$ **New Source**.
3. Dans la fenêtre **New Source Wizard**, On sélectionne **Test Bench WavForm** et saisir le nom de fichier qui est *compt_tbw*.
4. On clique **Next**, **Next**, puis **Finish**.
5. Avant que la fenêtre s'ouvre, il faut définir la fréquence d'horloge, les retards s'il y a des retards et la longueur initiale de Test Bench.

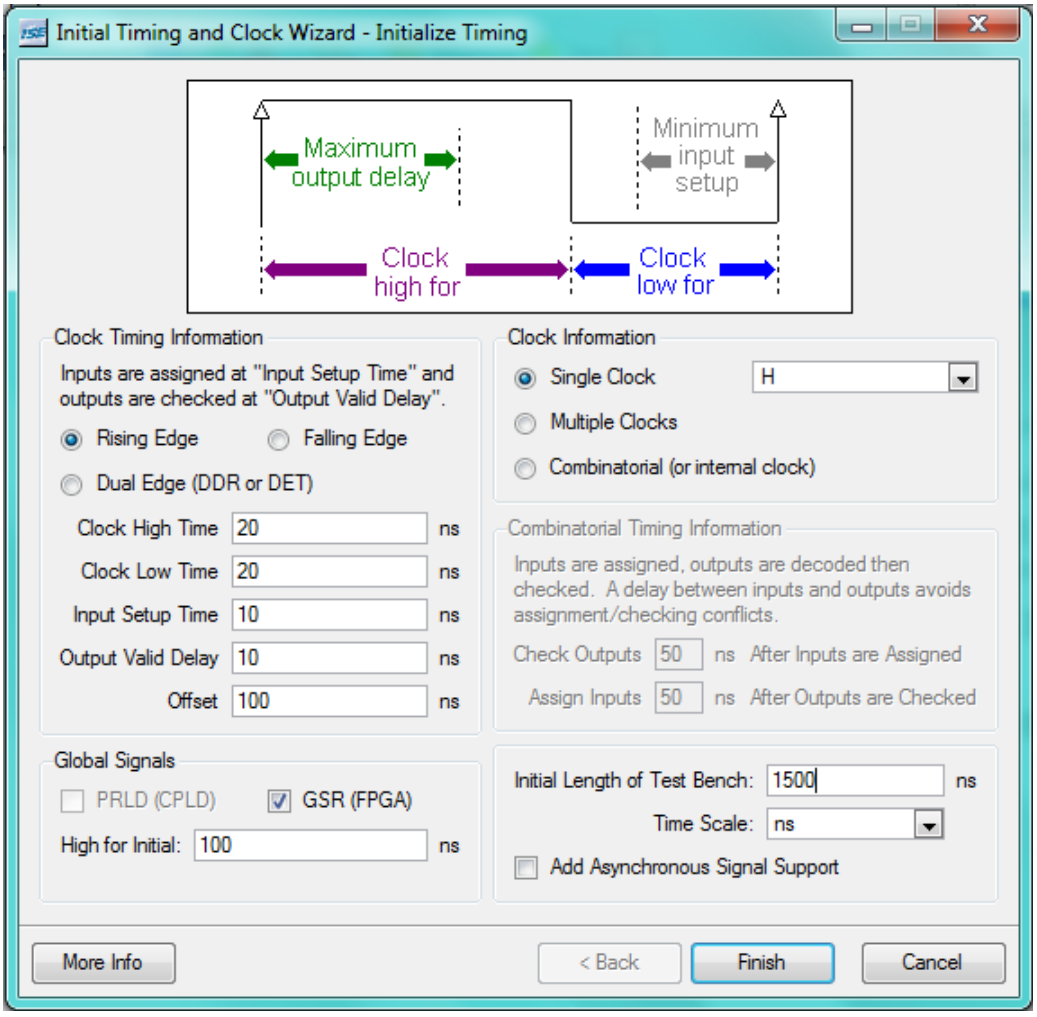


FIGURE 46 – Initialisation de synchronisation

6. On clique **Finish** pour terminer l'initialisation.

7. On choisit la valeur de signal *Direction* de 300 ns a 900 ns égal a '1'.

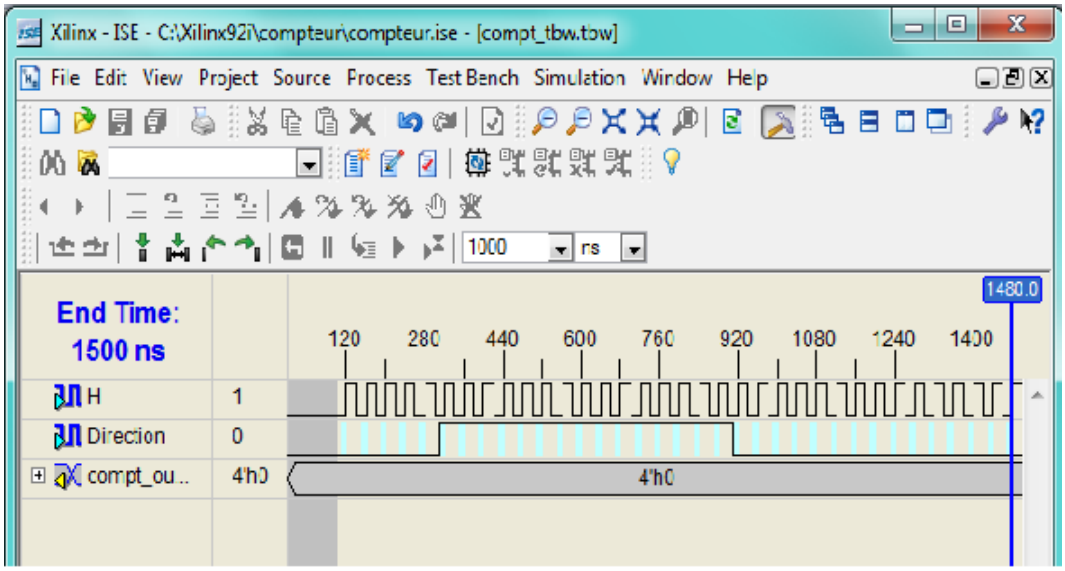


FIGURE 47 – Test Bench WaveForm

8. On sauvegarde.

9. Dans la fenêtre des sources, on choisit **Behavioral Simulation**.

10. Dans la fenêtre **Processes**, on clique le symbol '+' de **Xilinx ISE Simulator**.

11. Double-clic sur **Simulate Behavioral Model**. Le résultat de simulation est :

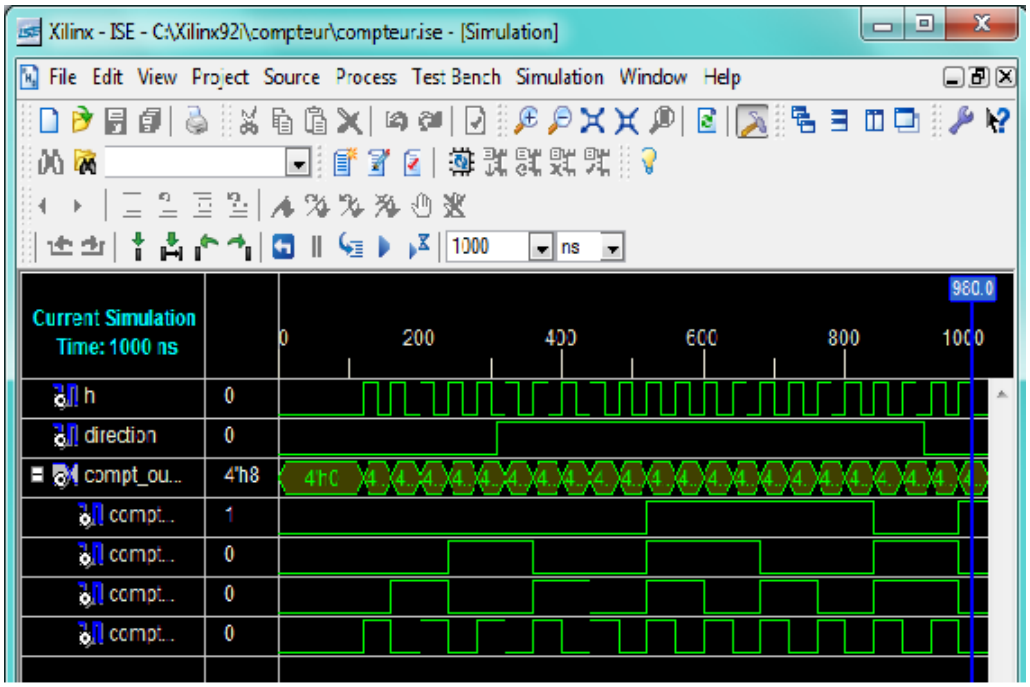


FIGURE 48 – compteur decompteur 4 bits

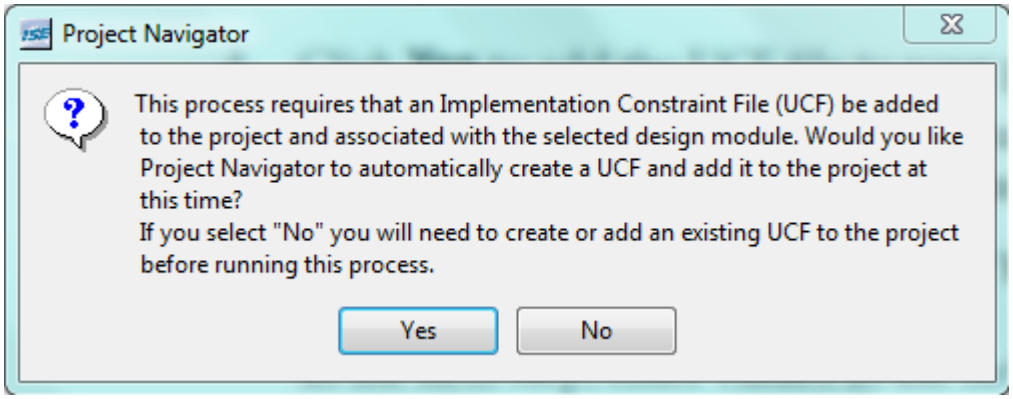
12. On ferme la fenêtre de simulation.

Création des contraintes de synchronisation

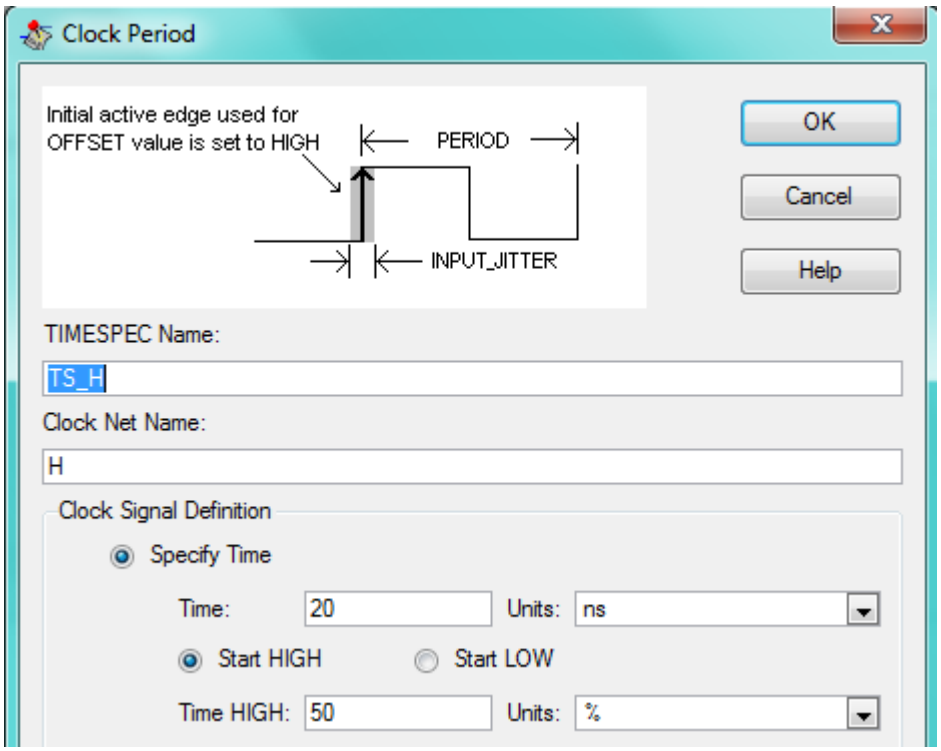
Pour contraindre la conception faire ce qui suit :

1. On sélectionne **Synthesis/Implementation** dans la fenêtre des **Sources**.
2. On sélectionne le fichier *compt.vhd*
3. Dans la fenêtre **Processes** cliquer le signe '+' de **User contraints**, puis double-clic sur **Create Timing Contraints**.

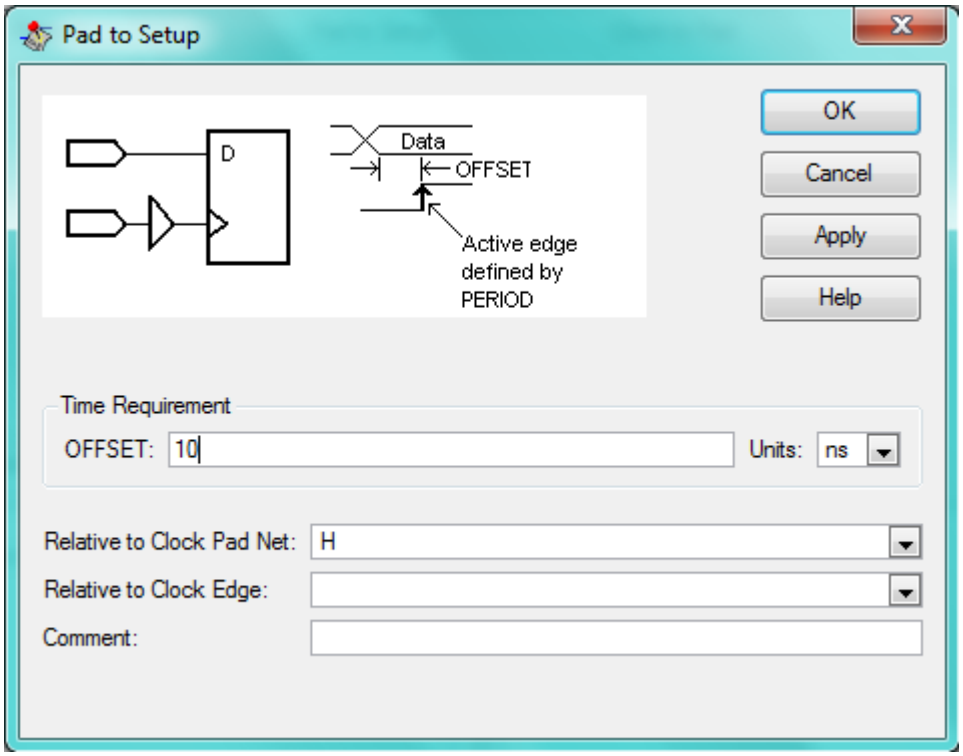
   ISE démarre la synthèse et traduit les étapes et crée automatiquement un fichier de contraintes de l'utilisateur (UCF). Le message suivant va apparaitre :



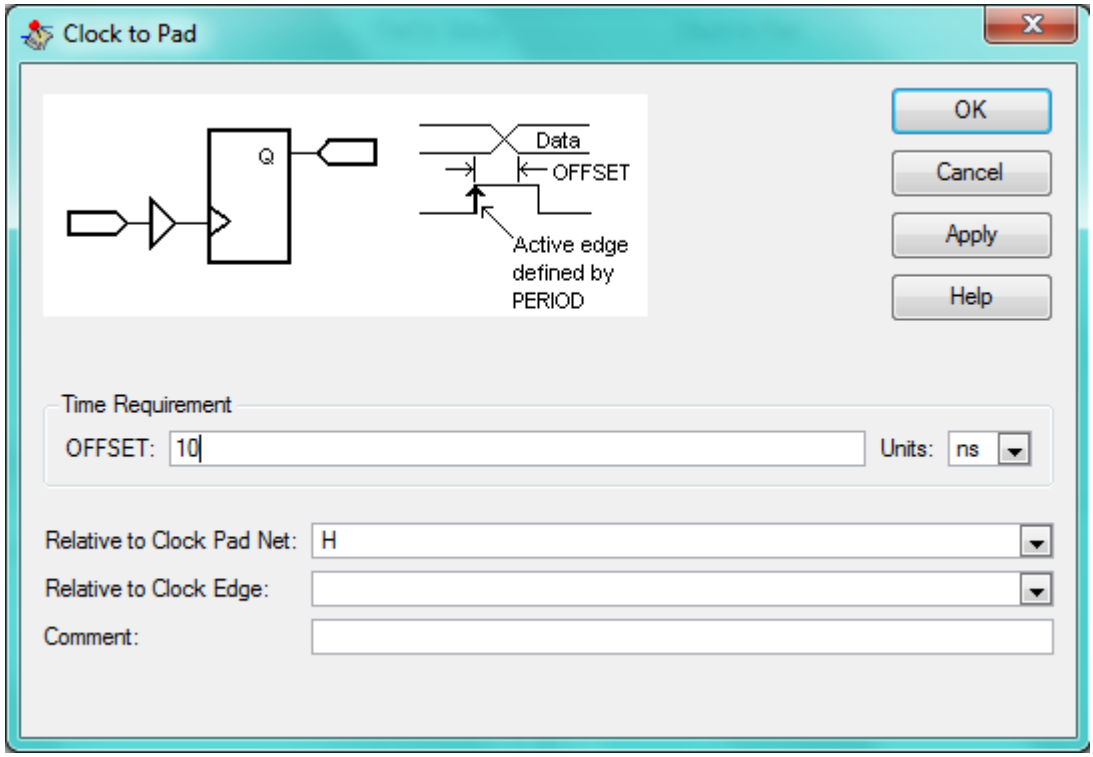
4. On clique **Yes** pour ajouter le fichier UCF au projet.
5. On clique sur **Global**, puis *H*.
6. Double-clic sur **Period** et on écrit 40 ns dans **Time**.



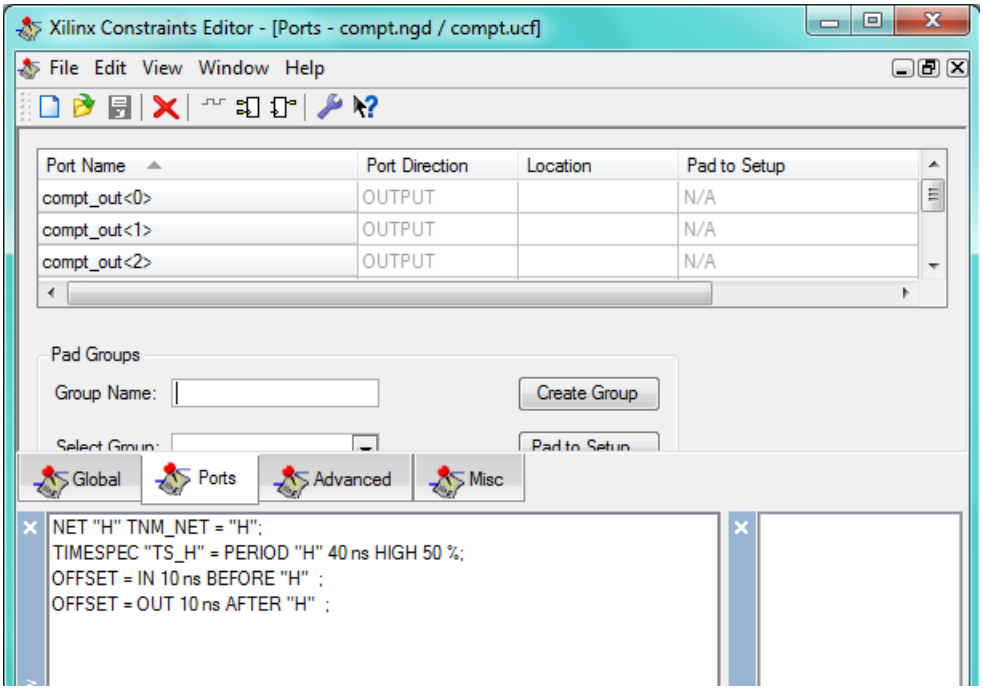
7. On clique **OK**.
8. Double-clic sur **Pad to Setup**, puis on remplit le champ **OFFSET** par 10 ns.



9. On clique **OK**.
10. Double-clic sur **Clock to Pad**, puis on remplit le champ **OFFSET** par 10 ns.



11. On clique **OK**.Les contraintes sont affichées dans l'étiquette (lecture/écriture) de contraintes, comme montré ci-dessous :



12. On sauvegarde les contraintes et on ferme la fenêtre **Contraints Editor**.

Implémentation du projet et vérification des contraintes

Implémentation du projet

1. On sélectionne le fichier source *compt.vhd* dans la fenêtres **Sources**.
2. On ouvre le résumé de la conception par double-clic sur **View Design Summary**.
3. Double-clic sur **Implement Design**.

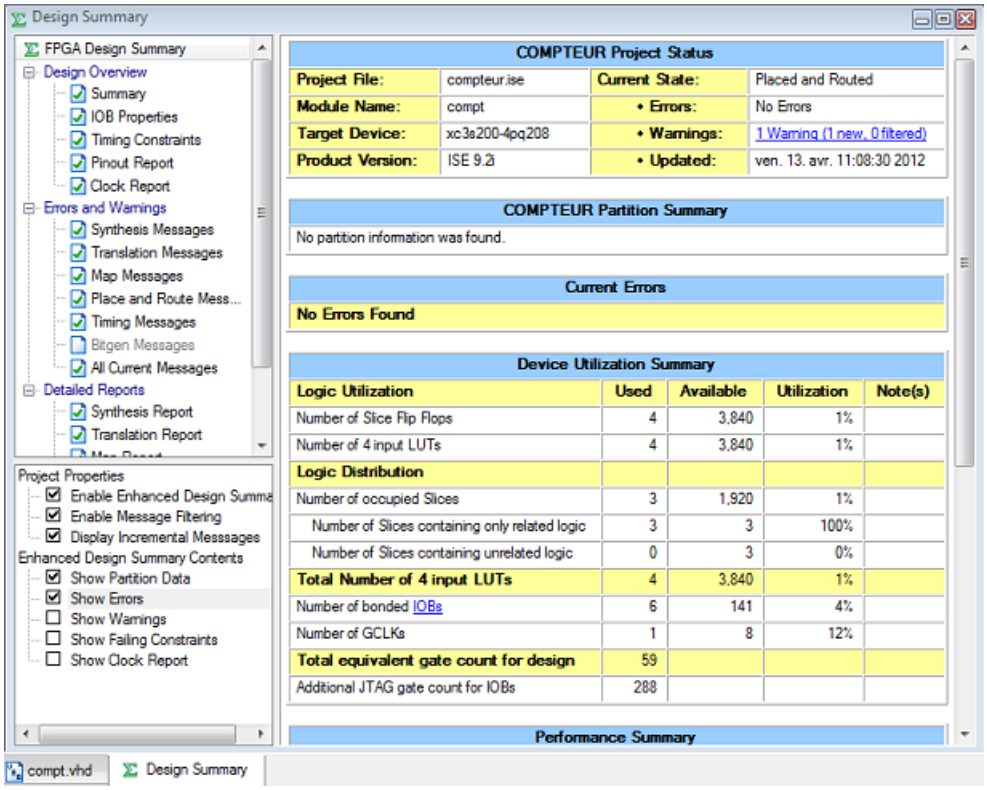


FIGURE 49 – Résumé de la conception

4. On clique **All Constraints Met** pour vérifier que la conception répond aux exigences de synchronisation définies.

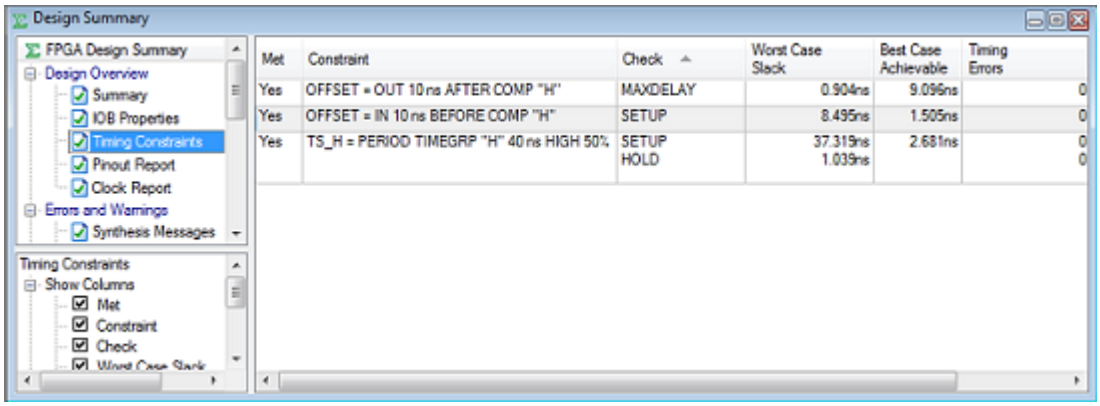


FIGURE 50 – Toutes les contraintes de synchronisation

5. On ferme la fenêtre **Design Summary**.

Attribution des contraintes d'endroit des broches

Il faut spécifier les endroits des broches pour les ports de la conception de sorte qu'ils soient reliés correctement sur le panneau de démarrage du circuit a programmer.

1. On vérifie que le fichier source *compt.vhd* est sélectionné dans la fenêtres **Sources**.
2. Double-clic sur **Assign Package Pins** trouvé dans **User contraints**.
3. On sélectionne **Package View**.
4. Dans la fenêtre **Design Object's**, on saisie un endroit de broche dans la colonne d'endroit en utilisant l'information suivante :
   - Le port d'entrée d'horloge **H** se relie à la broche **T9** de FPGA.
   - Le port **compt_out(0)** se relie à **K12**.
   - Le port **compt_out(1)** se relie à **P14**.
   - Le port **compt_out(2)** se relie à **L12**.
   - Le port **compt_out(3)** se relie à **N14**.
   - Le port d'entrée **Direction** se relie à **K13**.

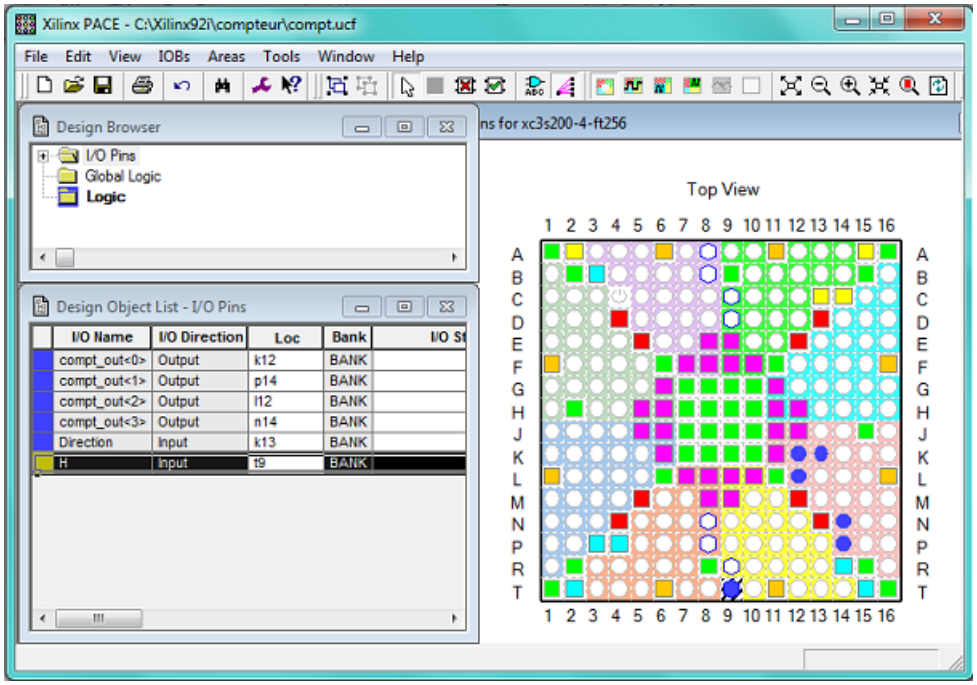


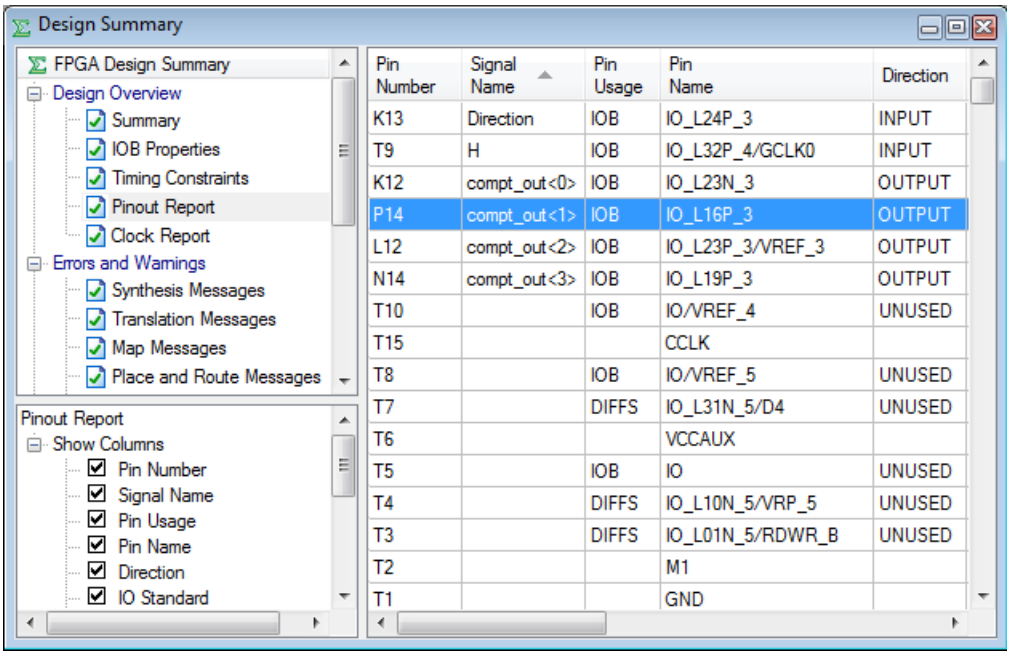
FIGURE 51 – Aperçu des broches d'entrées-sorties

5. Sauvegarder et fermer la fenêtre **Xilinx PACE**.

Reimplémentation de la conception et vérification des endroits des broches

Pour faire ceci :

1. Double-clic sur **View Design Summary**.
2. On sélectionne **Pinout Report**. On voit bien que les broches d'entrées et de sorties ne sont pas les même que les broches choisis.
3. Double-clic sur **Implement Design**.
4. On sélectionne **Pinout Report**. On peut vérifier que les broches sont corrects.



5. On ferme **Design Summary**.

Chargement de la conception dans le circuit

C'est la dernière étape dans l'implémentation de la conception. On doit suivre les étapes suivantes :

1. On relie le câble d'alimentation de 5V a l'entrée d'alimentation du circuit.
2. On relie le câble de téléchargement entre le PC et le circuit.
3. On sélectionne **Synthesis/Implementation** dans la fenêtre **Processes**.
4. On sélectionne le fichier source *compt.vhd* dans la fenêtre **Sources**.
5. On clique sur '+' de **Generate Programming File**.
6. Double-clic sur **Configure Device (iMPACT)**.
7. Dans la zone de dialogue bienvenue, on choisit **Configure devices using Boundary-Scan (JTAG)**.
8. On clique **Finish**.

Annexe C : Description de la carte Memec Design V2MB1000

Les familles des FPGA de Xilinx

La tendance des dernières générations est de cibler certains créneaux porteurs du marché, comme la solution "bas coût" ou à l'opposé la solution "haute performance" : la taille, le type et le nombre de cellules qui varient suivant les familles de composants. Ainsi, la panoplie présentée par le fabricant montre sa volonté de couvrir tous les segments de marché. Parmi les générations des composants FPGA de XILINX on site :
XC2000, XC3000, XC4000, XC5200, XC6200, XC8100, SPARTAN, VIRTEX, VIRTEX-II, VIRTEX-II Pro, Virtex-IV, Virtex-V.

FPGA de la famille Xilinx Virtex-II

Les FPGA du constructeur Xilinx sont divisés en deux gammes :

1. FPGA hautes performances : gamme " Virtex ",
2. FPGA pour la fabrication en grande série (Low Cost) : gamme " Spartan ".

Dans la gamme Virtex, on trouve plusieurs familles dont les performances et la complexité augmentent avec chaque nouvelle génération. On les classe par ordre chronologique dans ce qui suit :

1. Virtex : sortie en Novembre 1998,
2. Virtex-II : sortie en Janvier 2001,
3. Virtex-II Pro : sortie en Mars 2002,

4. Virtex-IV : sortie en Septembre 2004,
5. Virtex-V : sortie en Mai 2006.

Dans notre application nous étudions l'architecture détaillée du circuit Vertex-II, afin d'implémenter notre commande MLI.

La famille Virtex-II est le premier jeu de plate-forme FPGA, elle a été conçue pour réaliser des conceptions à faible ou grande densité d'intégration et exige des performances élevées (elle peut atteindre jusqu'à 10 million de portes logiques). Elle offre une densité variante de 40K à 8M portes logiques et peut atteindre une fréquence de fonctionnement de 420MHz. Son architecture est divisée en deux types de cellules de base comme le montre la figure 52.

1. Les cellules d'entrées/sorties appelées IOB (input output bloc),
2. Les cellules logiques appelées CLB (configurable logic bloc).

Ces différentes cellules sont reliées entre elles par un réseau d'interconnexions configurables.

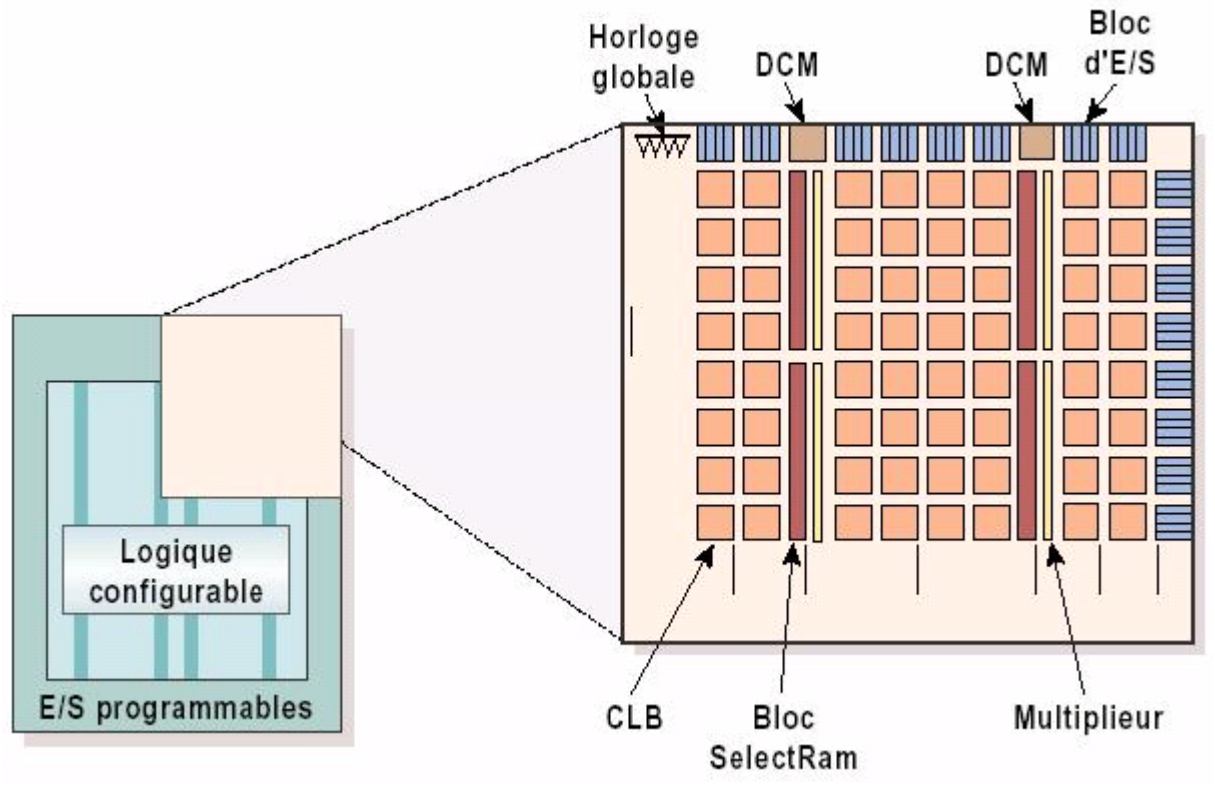


FIGURE 52 – Architecture interne de la famille VIRTEX-II

Développement d'un projet sur circuit FPGA

La carte de développement Memec Design V2MB1000

Le kit de développement Memec Design V2MB1000 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 cette famille 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. La figure 53 présente une photo de la carte de développement Memec Design V2MB1000.

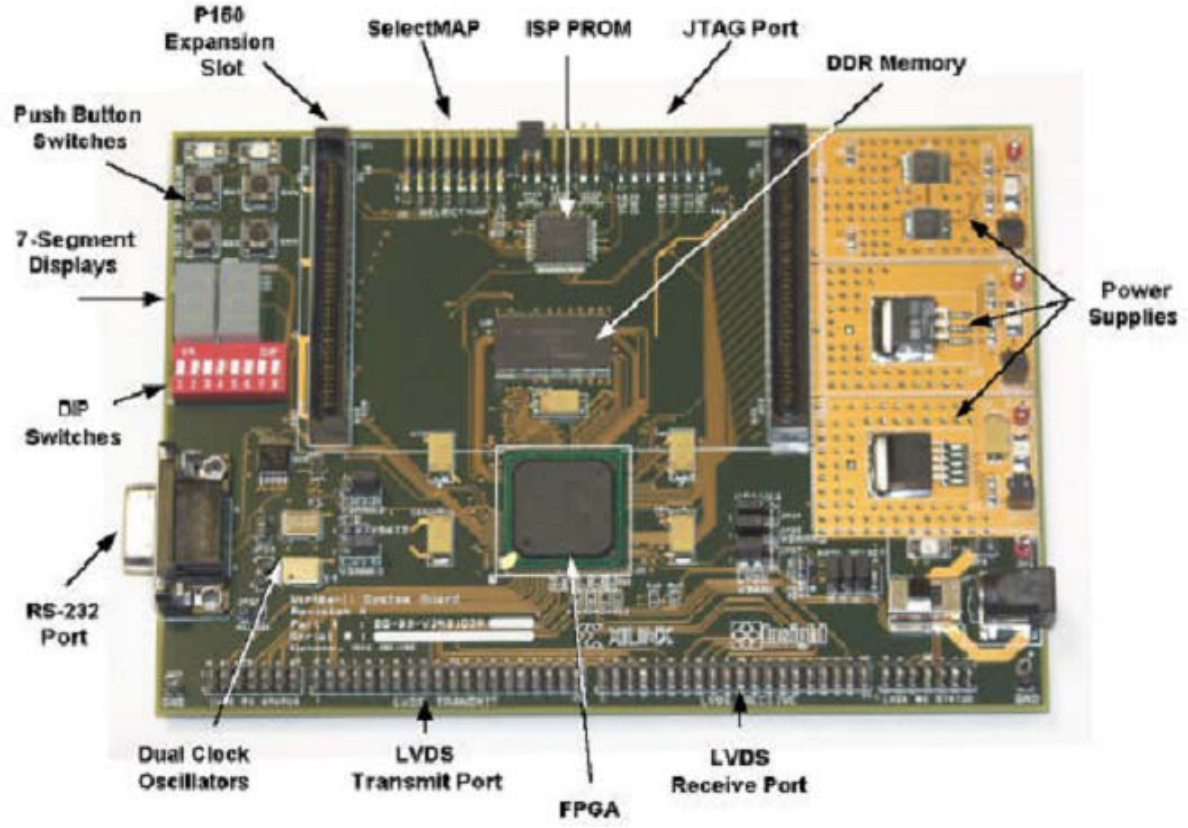


FIGURE 53 – Carte de développement "Memec Design V2MB1000"

.

Description de la carte de développement Memec Design Virtex-II V2MB1000

Un diagramme simplifié de la carte de développement Memec Design Virtex-II V2MB1000 est illustré à la figure 54.

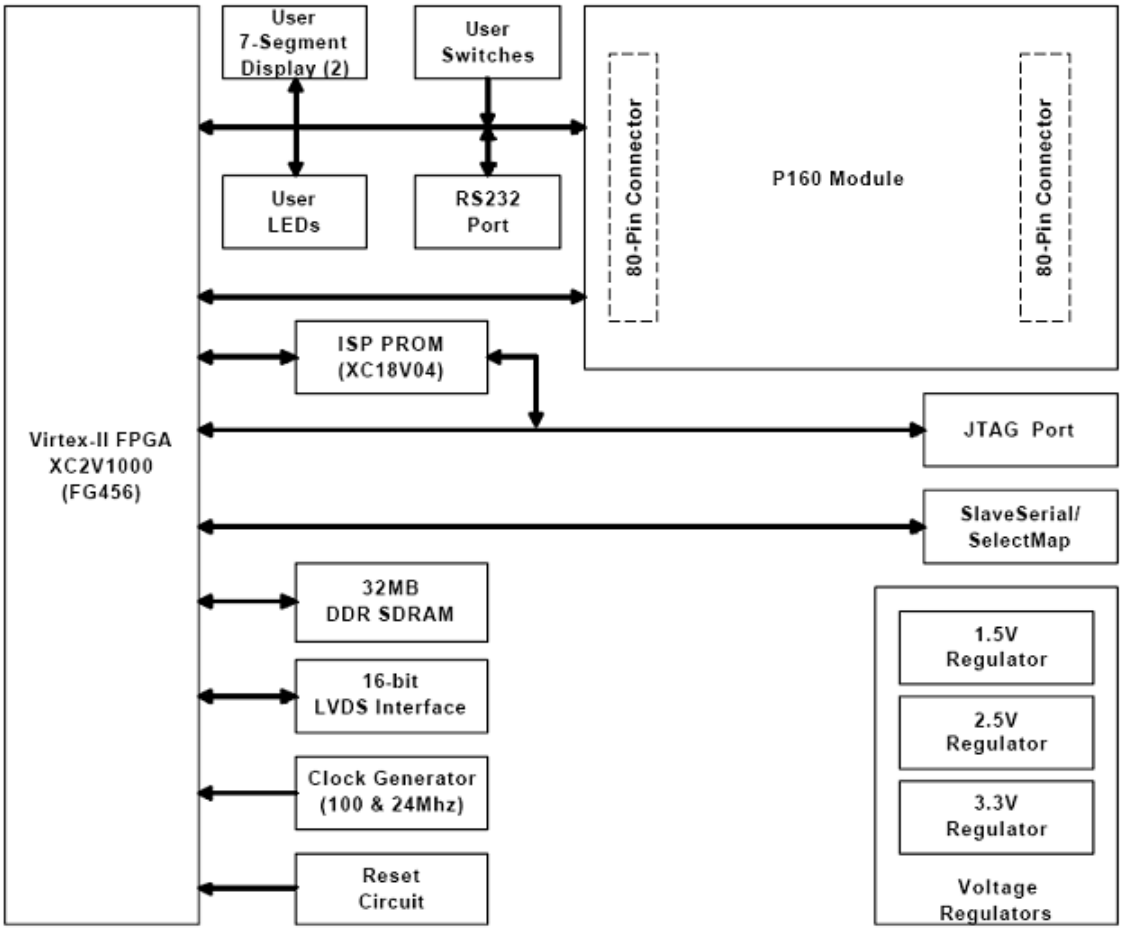


FIGURE 54 – Diagramme de la carte Memec Design Virtex-II V2MB1000.

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 10240 LUT et 10240 bascules "flip-flop". Sa capacité maximale en Select RAM est de 163840 bits.

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), un troisième signal d'horloge externe est disponible et peut être utilisé si on a besoin. Elle contient aussi un circuit de remise à zéro "Reset" activé par un bouton poussoir (SW3), un bouton poussoir "PROGn" pour initialiser la configuration et charger le contenu de la PROM dans le circuit FPGA

(SW2) ainsi que deux boutons poussoirs (SW5 et SW6) qui peuvent être utilisés pour générer des signaux actifs.
Deux afficheurs 7 segments à cathode commune sont présents sur la carte, et peuvent être utilisés durant la phase de test et de debugging.
Il existe aussi 8 entrées exploitables par l'utilisateur (DIP switch) qui peuvent être mis statiquement à un état haut ou bas.
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 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.

Chargement du programme sur la carte de développement

La carte de développement Memec Design V2MB1000 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 être aussi utilisé pour configurer le FPGA. La figure suivante montre l'installation pour toutes les configurations de modes supportés par la carte de développement Memec Design Virtex-II V2MB1000.

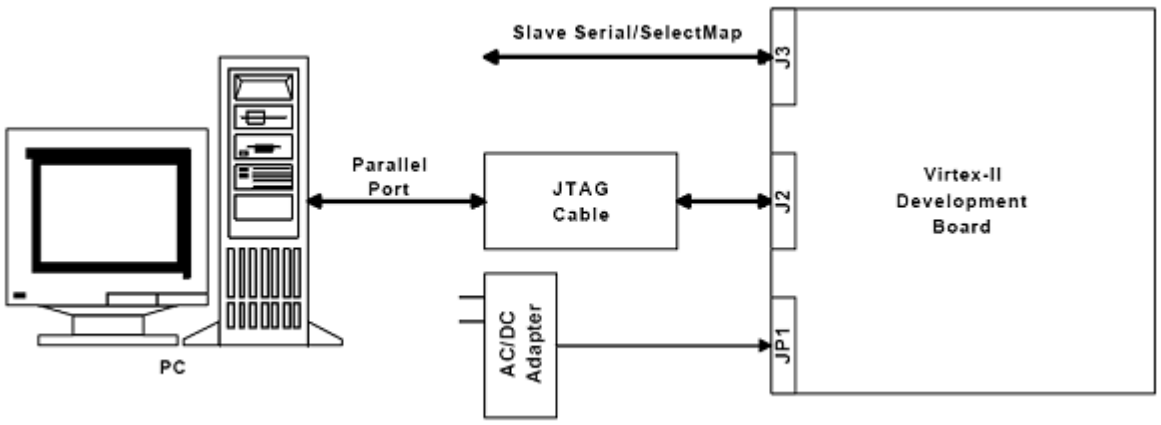


FIGURE 55 – Chargement du programme sur la carte FPGA

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 parallèle 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.

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.

Annexe D : Description de l'optocoupleur HCPL2200

Afin de pouvoir utiliser les signaux de sortie de la carte FPGA, qui sont de faible amplitude, et de protéger la carte on est obligé de concevoir un étage d'isolation entre cette partie et l'étage de puissance. La solution est d'utiliser des optocoupleurs, qui permettent la transmission optique de la commande éloignée. La particularité ici est de trouver des optocoupleurs faible courant et compatibles avec les signaux LVDS, à condition qu'ils puissent délivrer un courant suffisant aux drivers.

Le circuit utilisé pour notre cas est le HCPL2200 de la société HEWLETT PACKARD, ce circuit a les caractéristiques suivantes :

- Compatible avec les signaux LSTTL, TTL, et CMOS Logique.
- Un faible courant d'entrée (1.6 mA).
- Temps de montée (rise time) $t_r = 55ns$.
- Temps de descente (fall time) $t_f = 15ns$.

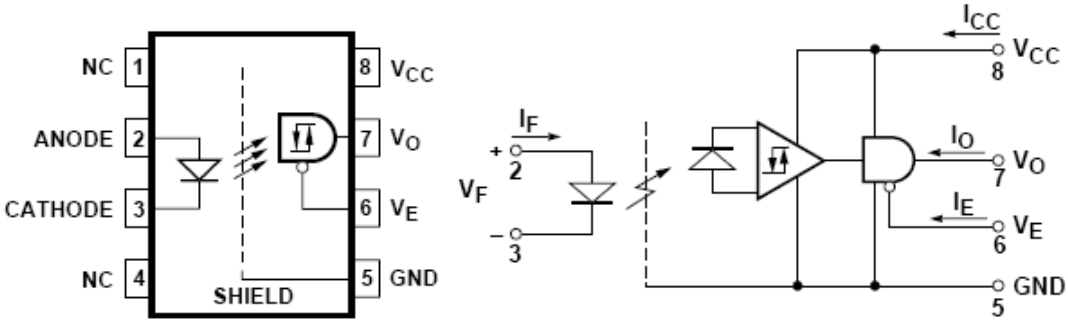


FIGURE 56 – L'optocoupleur HCPL2200 et son schéma interne

Une résistance est ajoutée à l'entrée 2 de l'optocoupleur afin de limiter le courant d'entrée et de le protéger.