MINISTEREDEL'ENSEIGNEMENTS UPERIEURET DELARECHERCHESCIENTIFIQUEUNIVE RSITE« Abbès LAGHROUR »DE KHENCHELAFACULTEDESSCIENCES ETDE TECHNOLOGIE

Département de Génie Industriel

N°de série:……

Mémoiredefin d'étude

Pourl'obtention du diplôme de Master(L.M.D)

Filière: Génie industriel

Spécialité:automatique et informatique industriel

Thème

Implémentation sur FPGA d'un système de commande basé sur la technique RPWM(application quartus)

Réalisé par:

- NABTI Fouad
- BENNOURI Fouad

Membres de jury

- Mr. DJAMAI Djmoui	Président
- Mr. BOUMAARAF Abdelaali	Rapporteur
-Mr. SAHOUR Abdelhakim	Rapporteur
- Mr. KHALIL Mokhtar	Examinateur

Année universitaire 2021/2022

Implémentation sur FPGA d'un système de commande basé sur la technique RPWM(application quartus)

Résumé :

Au bout d'un semestre de travail environ, nous avons essayé de réaliser un objectif de très grande importance du point de vue technologique. Il s'agissait d'envisager la possibilité d'employer les techniques les plus récentes en matière de traitement du signal et les outils les plus développés en matière de langage de programmation.

L'implémentation sur FPGA d'un système de commande basé sur la technique RPWM(application quartus)

Le système réalisé comprend un ensemble de modules programmés de manière à ce qu'ils fonctionnent en interaction entre eux et assurer chacun une fonction individuelle pour contribuer tous ensemble, dans une action collective qui constitue l'objectif de notre simulation.

Les résultats ont été vérifiés par module selon leurs fonctions respectives.

Espérant que d'autres projets futurs permettront de procéder à une implémentation réelle qui servira aux finalités escomptées.

Dédicace:

A mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études.

A ma femme mon soutien dans la vie et l'ame de mon cœur.

A mes enfants, anas, reyhane, yakob, Khalil, la lumière de ma vie

A mes chères sœurs pour leurs encouragements permanents, et leur soutien moral

A mes chers frères, pour leur appui et leur encouragement.

A toute ma famille pour leur soutien tout au long de mon parcours universitaire.

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fuit de votre soutien infaillible.

.Merci d'être toujours là pour moi

BENNOURI fouad

Dédicace:

A mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études.

A mes chères sœurs pour leurs encouragements permanents, et leur soutien moral.

A mes chers frères, , pour leur appui et leur encouragement.

A toute ma famille pour leur soutien tout au long de mon parcours universitaire.

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fuit de votre soutien infaillible.

Merci d'être toujours là pour moi.

NABTI Fouad

Remerciements

Je tiens à remercier, en tout premier lieu, Monsieur BOUMAARAF Abdelâali, rapporteur de notre travail qui m'a toujours encouragé depuis le début et sans lui je n'aurais jamais eu la volonté de commencer ce MASTER ni la force de l'achever.

Je tiens à remercier spécialement Dr. SAHOUR Abdelhakim,

Je tiens à remercier les membres du jury pour leur présence, pour leur lecture attentive de ce mémoire, ainsi que pour les remarques qu'ils m'adresseront lors de cette soutenance afin d'améliorer notre travail.

Bennouri fouad

Nabti fouad

SOMMAIRE

Liste des figures:

Liste des tableaux:

Listes des Acronymes et Symboles

Acronymes

M.L.I. Modulation en Largeur d'Impulsion

F.P.G.A. Field Programmable Gate Array

R.P.W.M. Repeated Pulse Width Modulatio

M.L.I.R. Modulation en Largeurd'ImpulsionRépétée, Very high speed integrated circuit

V.H.D.L. Hardware Description Language

U/f. Tension sur fréquence

P.W.M. Pulse Width Modulation

INTRODUCTION GENIRALE :

Introduction genirale :

Le domaine des télécommunications et des techniques de modulation numérique ont connu un développement considérable durant les dernières décades. Cette croissance technologique accrue a fait de la modulation numérique un moyen de plus en plus utilisé dans les commandes des systèmes industriels à travers les convertisseurs de puissance. La Modulation en Largeur d'Impulsion (M.L.I.) (En Anglais Pulse Width Modulation)) est l'une des techniques qui illustrent l'apport important en matière d'optimisation du fonctionnement et d'augmentation du rendement des machines de puissance, en agissant sur les signaux de commande.

Dans ce travail nous proposons une Implémentation sur FPGA d'un système de commande basé sur la technique RPWM(application quartus), c'est la M.L.I. Répétée (En Anglais Repeated Pulse Width Modulation (R.P.W.M.)) qui optimise encore mieux la M.L.I. classique .

Les parties composant l'environnement de cette conception sont décrites en quatre chapitres distincts ; Le premier chapitre c'est FPGA et le VHDL. objet du deuxième chapitre est un passage obligatoire pour donner une interprétation des techniques de la M.L.I. qui donnent une initiation à la M.L.I. Répétée ou R.P.W.M. Le chapitre trois réservé à la simulation qui constitue l'objet même de ce travail. Finalement un quatrième et dernier chapitre c'est limplimentation de la technique RPWM a l'aide de logiciel quartus.

Une partie que nous avons jugée aussi importante a été mentionnée en annexe, c'est le langage de description V.H.D.L. qui est des plus récents dans la conception des systèmes programmables regroupant performance et flexibilité.[6][7][8][16]

CHAPITRE I : LES FPGA ET LE VHDL

Chapitre I : les FPGA et le VHDL

I. 1.Introduction

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.

Suite à la révolution technologique du 20eme siècle et l'implantation de l'électronique dans tous les domaines de la vie, l'évolution des circuits électronique était très remarquable et une nécessité d'avoir des circuits puissant et spécialisé pour des applications bien précises a été apparait. De plus en plus ces applications demandent que ces circuits soient programmables, cela à donner la naissance d'une famille des circuits programmable comme les FPGAs.

Les FPGAs, apparus en 1985 sous l'initiative de l'entreprise XILINX, ce sont des composants électroniques entièrement reconfigurables pouvant être reprogrammés afin d'accélérer notablement certaines phases de calculs.

La programmation de l'architecture d'un FPGA, bien qu'il soit possible de programmer en schématique, se fait principalement en langage VHDL (ou Verilog). Nous utilisons les FPGA XILINX (VERTEX) et l'outil de synthèse produit par le même fabriquant, à savoir ISE.

Dans la première partie de ce chapitre on présente les circuits FPGAs ainsi que son architecture interne, on présente leurs avantages et leurs inconvénients. Dans la deuxième partie nous abordent le langage[3]

I.2.FPGA

I.2.1-Definition FPGA[3][2]

Pour pallier les défauts des CPLD[5], les **FPGA** (*field-programmable gatearray*, réseau de portes programmables *in situ*) ont été développés dans les années 2000.

Dansl'esprit, cesont:

- des "CPLD" où les cellules peuvent contenir :
    - des portes logiques librement interconnectables entre elles

    - des fonctions élémentaires (bascules flip-flop, LUT, Multiplexeurs, boucles à verrouillage de phase)
    - des fonctions de traitement "précablées" (DSP, protocoles de communication comme le PCIe, voire des fonctionnalités de microprocesseurs, de mémoire, etc.)
    - néanmoins, l'architecture et la conception interne des FPGA diffèrent radicalement de celle des CPLD)
- des puces où la performance est un point important, aussi les vitesses de traitement des FPGA (bande passante) surclassent les CPLD. Ceci est notamment dû à l'utilisation de SRAM à la place d'EEPROM comme mémoire interne)
- La technologie des FPGA fait que leur programmation est volatile et doit dont d'être stockée dans une ROM ou mémoire flash. Un FGPA doit donc charger son programme à chaque mise en route. Les dernières puces intègrent désormais une ROM qui garde en mémoire le programme à lancer à chaque démarrage

I.2.2-les circuits programmables FPGAs[3]

Les FPGA, sigle anglais qui signifie « **F**ield **P**rogrammable **G**ates **A**rrays » traduit en français par réseau de portes programmables, les FPGAs peuvent être programmé ou reprogrammé pour la fonctionnalité ou l'application requise après fabrication, ils constituent une évolution des circuits logiques programmables **CPLDs**,
L'FPGA est constitué de blocs logiques programmables, d'interconnexions reconfigurables et de blocs d'entrée / sortie. Les blocs logiques utilisés dans un réseau de portes programmables sur site peuvent être constitués d'éléments de mémoire tels que des bascules ou des blocs de mémoire. Les blocs logiques sont capables d'effectuer des fonctions de calcul simples à complexes. Les réseaux de portes programmables sur le terrain sont à bien des égards similaires aux puces de mémoire programmables en lecture seule. Cependant, contrairement aux puces mémoires programmables en lecture seule, qui sont limitées à des centaines de portes, un réseau de portes programmables sur le terrain peut supporter plusieurs milliers de portes. Une autre caractéristique importante des réseaux de portes programmables sur le terrain est la possibilité de reprogrammation, contrairement aux circuits intégrés

spécifiques à l'application qui sont fabriqués pour des tâches spécifiques.
Un réseau de portes programmable par l'utilisateur peut aider les utilisateurs d'ordinateurs à adapter les capacités des microprocesseurs aux besoins individuels spécifiques. En fait, les ingénieurs utilisent des réseaux de portes programmables sur le terrain pour concevoir des circuits intégrés spécialisés. D'autresavantages de l'utilisation de réseaux de portes programmables sur site comprennent un cycle de vie plus prévisible grâce à l'élimination des capacités de plaquettes, des réseaux potentiels, un temps de mise sur le marché plus rapide par rapport aux autres options et un cycle de conception simple.
Ils offrent la possibilité de réaliser des fonctions numériques plus ou moins complexes.
Les deux plus grands constructeurs de FPGA sont XILINX et ALTERA

I.2.3.Les Circuits Logiques:

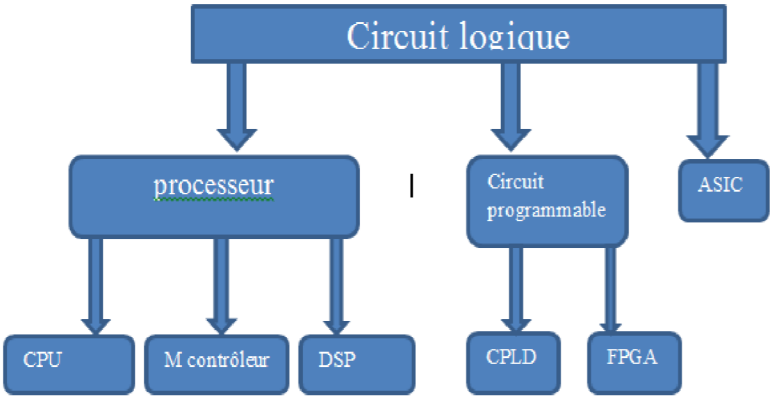


Figure 1.1:Les Circuits Logiques

CPU : Computer Processing Unit
DSP : Digital Signal Processor
CPLD : Complex Programmable Logic Device
FPGA : Field Programmable Gate Array
ASIC : Application Specific Integrated Circuit

I.2-3.1.Comparaison entre les circuits ASIC et FPGA[3]

Les **FPGA** constituent l'autre option majeure de l'accélération matérielle. Il s'agit de circuits composés de cellules qui, contrairement au **CPU**, peuvent être reprogrammées après fabrication. Les utilisateurs peuvent donc attribuer différentes fonctions aux cellules et redéfinir les interconnexions.

ASIC

*Développement long

*Coût de fabrication (enaugmentation)

*Full custom : Performancesmaximales

*Fabrication grande série

FPGA

*Développement rapide

*Coût à l'unité (endiminution)

*Contraint par la technologiedu FPGA

*Prototypage rapide

I.2.4.Architecture des FPGA[3]

La plupart des grands FPGA modernes sont fondés sur des cellules SRAM aussi bien pour le routage du circuit que pour les blocs logiques à interconnecter.
Un bloc logique est de manière générale constitué d'une table de correspondance (LUT ou *lookup table*) et d'une bascule (*flip-flop* en anglais). La LUT sert à implémenter des équations logiques ayant généralement 4 à 6 entrées et une sortie. Elle peut toutefois être considérée comme une petite mémoire, unmultiplexeur ou un registre à décalage. Le registre permet de mémoriser un état (machine séquentielle) ou de synchroniser un signal (*pipeline*).
Les blocs logiques, présents en grand nombre sur la puce (de quelques milliers à quelques millions en 2007) sont connectés entre eux par une matrice de routage configurable. Ceci permet la reconfiguration à volonté du composant, mais occupe une place importante sur le silicium et justifie le coût élevé des composants FPGA. La topologieest dite « Manhattan », en référence aux rues à angle droit de ce quartier de New York.

Les densités actuelles ne permettent plus un routage manuel, c'est donc un outil de placement-routage automatique qui fait correspondre le schéma logique voulu par le concepteur et les ressources matérielles de la puce. Comme les temps de propagation dépendent de la longueur des liaisons entre cellules logiques, et que les algorithmes d'optimisation des placeurs-routeurs ne sont pas déterministes, les performances (fréquence max.) obtenues dans un FPGA sont variables d'un design à l'autre. L'utilisation des ressources est par contre très bonne, et des taux d'occupation des blocs logiques supérieures à 90 % sont possibles.

Comme la configuration (routage et LUT) est faite par des points de mémoire volatile, il est nécessaire de sauvegarder le design du FPGA dans une mémoire non volatile externe, généralement une mémoire Flash série, compatible « JTAG ». Certains fabricants se distinguent toutefois par l'utilisation de cellules EEPROM pour la configuration, éliminant le recours à une mémoire externe, ou par une configuration par anti-fusibles (la programmation par une tension élevée fait « claquer » un diélectrique, créant un contact). Cette dernière technologie n'est toutefois pas reconfigurable.

Quelques fonctionnalités particulières disponibles sur certains composants :

- blocs de mémoire supplémentaires (hors des LUT), souvent double-port, parfois avec mécanisme de FIFO ;
- multiplieurs câblés (coûteux à implémenter en LUT), voire blocs multiplieur-accumulateur pour traitements DSP ;
- cœur de microprocesseur enfoui (dit *hard core*) comme des architectures PowerPC ou ARM ;
- blocs PLL pour synthétiser ou resynchroniser les horloges ;
- reconfiguration partielle, même en cours de fonctionnement ;
- chiffrement des données de configuration ;
- sérialiseurs/désérialiseurs dans les entrées-sorties, permettant des liaisons série haut-débit ;

- impédance contrôlée numériquement dans les entrées-sorties, évitant de nombreux composants passifs sur la carte ;
- couche MAC Ethernet ;
- couches matérielles

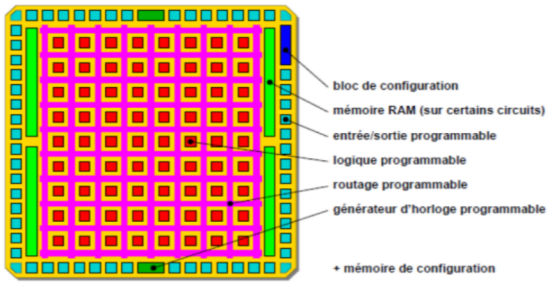


Figure 1.2:mémoire de configuration

I.2.4.1.les circuits programmables FPGAs[3]

la structure interne d'un FPGA de type matrice symétrique

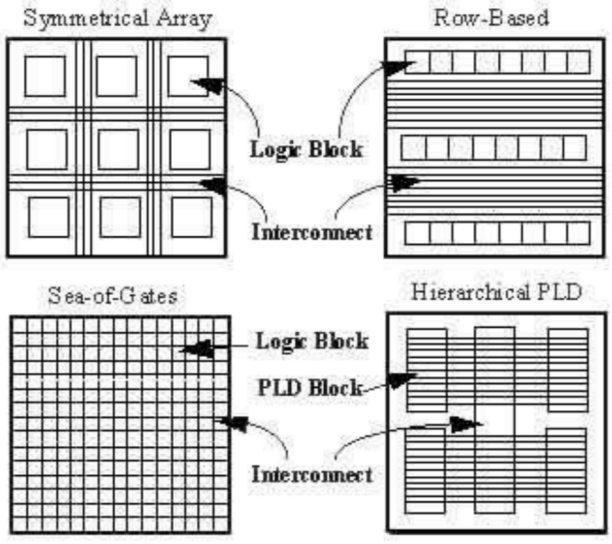


Figure 1.3 :structure interne d'un FPGA de type matrice symétrique

voici la structure interne d'un FPGA de type matrice symétrique. Il s'agit de l'architecture que l'on retrouve dans les FPGA de la série XC4000 de chez Xilinx.

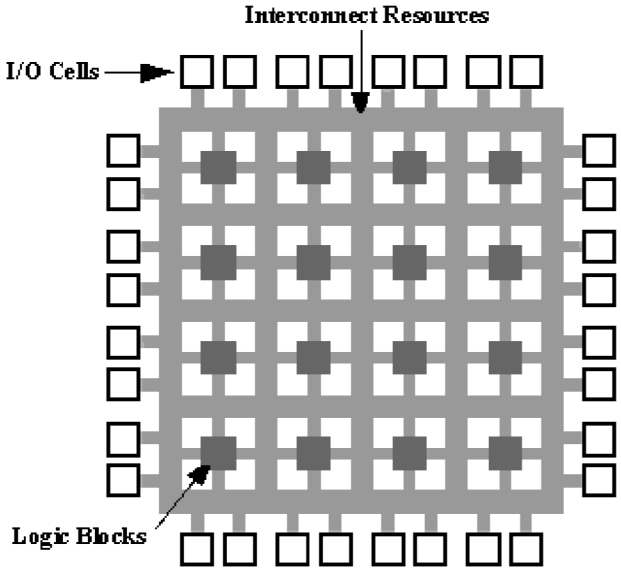


Figure 1.4 :structure interne d'un FPGs

L'utilisateur peut programmer la fonction réalisée par chaque cellule (appelée CLB par Xilinx: Configurable Logic Block):

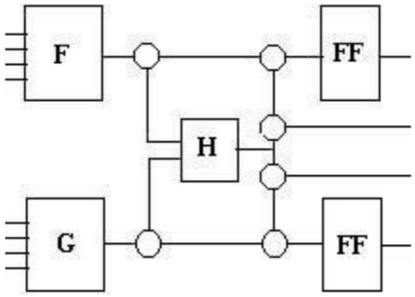


Figure 1.5:Schéma bloc d'une cellule

On programme aussi les interconnexions entre les cellules:

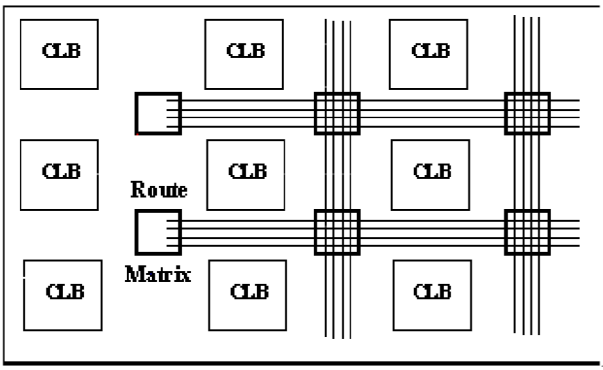


Figure 1.6:Les interconnexions entre les cellule d'un FPGA

I.2.5.Programmation[3]

La programmation des PLD modernes (FPGA / CPLD) passe généralement par un compilateur basé sur un langage de programmation de type langage de description matériel (ou « HDL » pour Hardware Description Language) comme le ABEL. Pour faciliter la programmation, il existe aussi des langages de plus haut niveau. Les deux plus connus sont le VHDL ("V" pour "*Very high speed*") et le VERILOG. Il est également possible d'utiliser OpenCL[6], plutôt qu'un VHDL, ou bien des langages de plus haut niveau que les HDL.

Les autres langages sont ensuite traduits synthétiseurs de leurs FPGA dans Yosys plutôt que d'en créer de nouveaux dans un des deux HDL — Verilog ou VHDL — avant de pouvoir être synthétisés pour le circuit. Le synthétiseur est généralement propre à chaque fabriquant, cependant, aujourd'hui, l'utilisation du logiciel libre Yosys, à l'instar de GCC pour la compilation en langage machine, tend à unifier la synthèse pour tous les types de PLD. Certains constructeurs, tels Renesas ou Cologne Chip Design, ont préféré implémenter directement les

I.2.6.domaines d'application des FPGAs[3]

- Informatique : Périphériques spécialisés.
- Machinerie industrielle : Contrôleur pour machines.
- Télécommunications : Traitement d'images, Filtrage.
- Instrumentation : Équipement médical, Prototypage.
- Transport : Contrôle d'avions et métros.
- Aérospatiale: Satellites, Radar, la détection ou la surveillance……etc

I.2.7-Les technologies FPGA[3]

1- Performances

2- Coût

3.-Fiabilité

4.-Temps de mise sur le marché

5- Maintenance à long terme

I.2.8.les blocs logiques CLBs (Configurable Logic Blocs)[3]

Sont des éléments déterminants des performances du FPGA et contenant les fonctions logiques combinatoires et séquentielles.

La partie combinatoire permet de réaliser des fonctions arithmétiques et logiques de complexité variable. Il est en effet possible d'utiliser plusieurs méthodes de synthèse dont les principales sont : la synthèse de fonctions à 4 ou 5 variables avec des portes classiques ET, OU et NON, la synthèse de fonctions combinatoires à l'aide de mémoires vives. Dans ce dernier cas, on dit aussi réalisation de fonctions logique par LUT (Look-Up Table) signifiant table de réalisation (ou d'observation).

La partie séquentielle comporte en règle générale une ou deux bascules de type D.

Suivant le fabricant du circuit, ces blocs contiennent un nombre différent de portes logiques et de bascules à l'intérieur d'un bloc comme le montre la figure **1.7**

Il existe 4 types de blocs logiques :

Les macro-cellules.

Les blocs à multiplexeurs.

Les LUT

Les cellules symétriques.

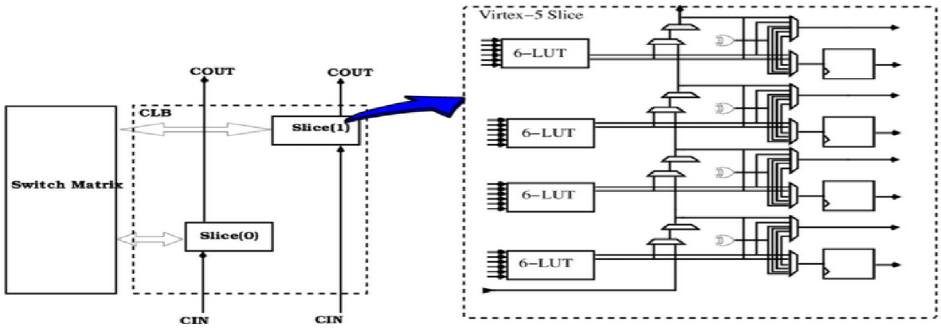


Figure 1.7: Blocs logique configurable(CLBs) Virtex-5 contenant deux slices**.**

I.2.8.1 les cellules d'entrées/sorties IOBs (Input Output Blocs)[3]

Ces blocs permettent d'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Pour adapter les signaux suivants :

-Alimentation.

-Signaux d'horloge.

-Signaux de configuration du FPGA.

-Signaux de test.

Les IOBs permettent l'interconnexion de la logique interne aux ports d'entrées et de sorties du FPGA. Les IOBs ont leur propre mémoire de configuration, elles stockent les standards de tension et la direction des ports. Ces blocs sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être reconfiguré en entrée ou en sortie, en signaux bidirectionnels ou être inutilisé (hauteimpédance).

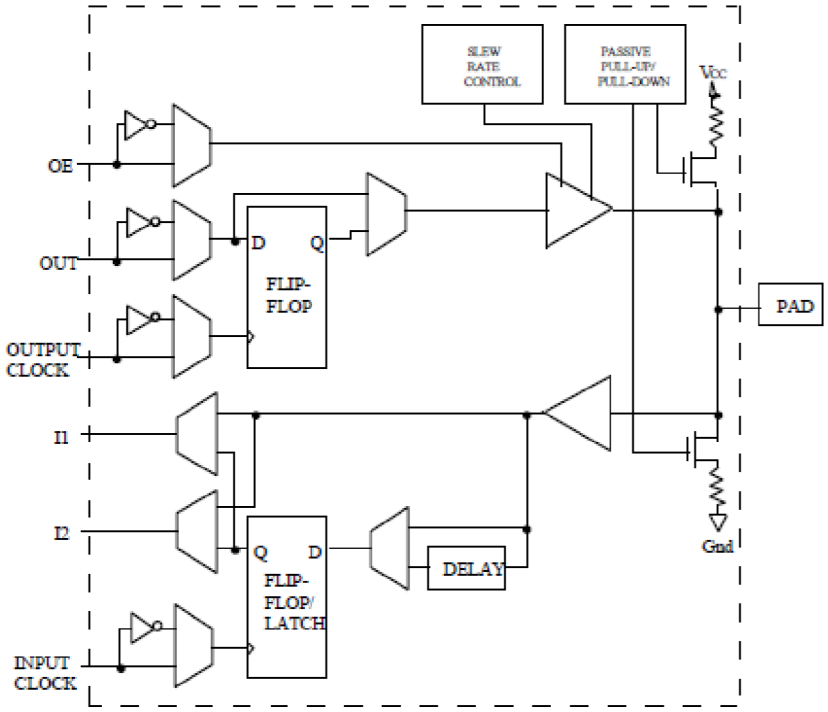


Figure 1.8 :exemple de Cellule I/O (IOB) de la famille X4000.

I.2.8.2.configuration en entrée[3]

Premièrement, 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 descendent. De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (program controlled multiplexer). Un bit positionné dans une case mémoire commande ce dernier. **[1]**

I.2.8.3 Configuration en sortie:[3]

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.

I.2.9.Les interconnexions[3]

Les ressources d'interconnexion au sein d'un FPGA permettent la connexion arbitraire des CLB et des IOB. 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 comme le montre la Figure **1.9**

– Les interconnexions directes : Ces interconnexions permettent l'établissement des liaisons entre les CLB et les IOB. Il est possible aussi de connecter directement certaines entrées d'un CLB aux sorties d'un autre.

– Les longues lignes : Ce sont de longs segments métallisés parcourant toute la longueur et la largeur du FPGA, 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.

– Les matrices d'interconnexion : Ce sont des aiguilleurs situés à chaque intersection. Leur rôle est de raccorder les longues lignes entre elles selon diverses configurations.

Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre CLB sur

le FPGA pour assurer la communication des signaux. Pour éviterl'affaiblissement des signaux traversant les longues lignes, des buffers sont implantés dans chaque matrice d'interconnexion.

Les trois blocs présentés jusqu'ici sont interconnectés ensemble dans le dispositif pour créer une infrastructure de communication composée d'un réseau de communication et d'IOBs autour des CLBs. Des cellules de mémoire liées à chaque bloc détiennent les caractéristiques principales, de telle sorte que les interconnexions entre l'infrastructure de communication, les normes de tension des entrées-sorties d'un IOB, et les équations soient commandées par des valeurs particulières stockées dans une mémoire. Toutes ces configurations sont stockées dans des SRAM qui sont volatiles : lorsque le composant est mis sous tension, toute sa configuration est perdue et il doit être redémarré avec une nouvelle configuration. Habituellement, une machine externe se charge de télécharger la configuration sur le FPGA via une de ses interfaces de configuration, et envoie une commande de démarrage pour signaler que la configuration a eu lieu. Certaines cartes ont une mémoire ROM d'intégrée. Elle permet de stocker la configuration, de sorte qu'elle puisse ensuite être téléchargée sur le FPGA. Dans ce cas, les données de configuration sont copiées sur la mémoire SRAM de configuration du FPGA au démarrage de celui-ci.

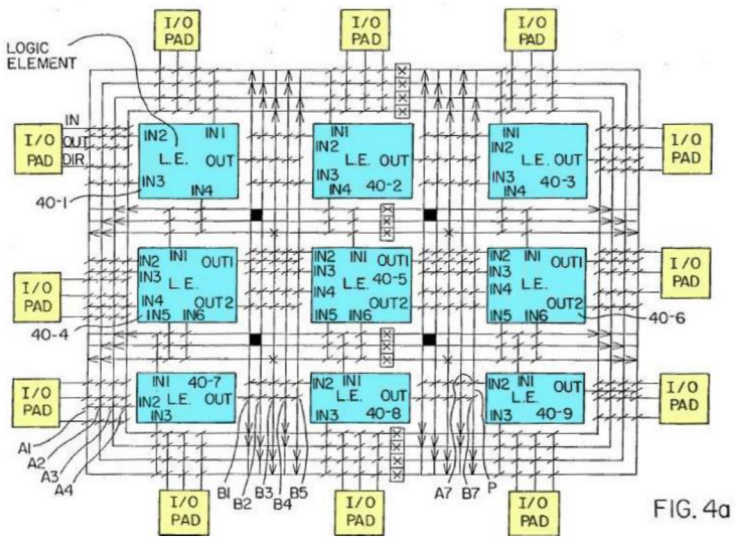


Figure 1.9: Interconnexion interne d'un FPGA.

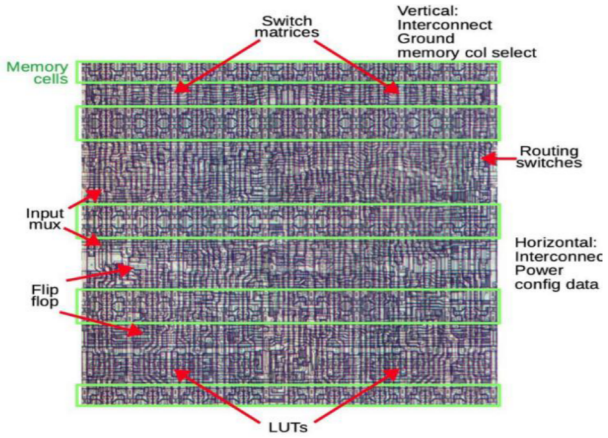


Figure 1.10: Une tuuilefpga montrant les neoruds fonctionnels

I.2.10.Technologies de programmation des FPGA[3]

Il existe trois types d'FPGAs reprogrammables suivant la technologie de mémorisation pour répondre aux différentes applications :

-**ANTIFUSIBLE** (la plus ancienne, configurable une seule fois).

-**FLASH** (non-volatile).

-**SRAM** (volatile, la plus utilisée, représente plus de 80 % du marché).

I.2.10.1 Technologie à base d'anti-fusibles (ACTEL)[3]

Les points de connexions sont du type ROM, c'est-à-dire que la modification du

point est irréversible. Pour comprendre le mécanisme de connexion sans rentrer dans les détails des semi-conducteurs, on considère que le point de connexion est le point de rencontre de deux segments conducteurs ou lignes conductrices. Le nom antifusible
vient du fait que l'état initial du fusible ou la couche isolante est présente et il n'y a pas de contact pour l'établir, il faut détruire le fusible ce qui est contradictoire au fonctionnement habituel d'un fusible. Des composants moins génériques mais plus petits et plus rapides ont été développés figure **1.11**.

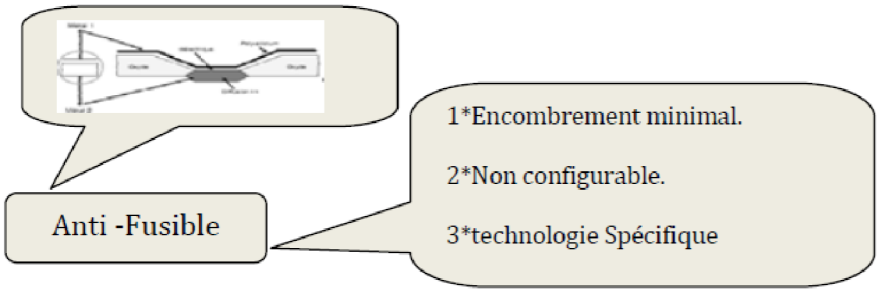


Figure1.11 : Caractéristiques des technologies ANTI-FUSIBLES.

I.2.10.2.Technologie à base d'EEPROM ou FLASH (LATTICE et ACTEL)[3]

Cette technologie garde sa configuration mais un nombre limité de configuration avec une configuration plus lente par rapport à SRAM comme le montre la figure **I.12**

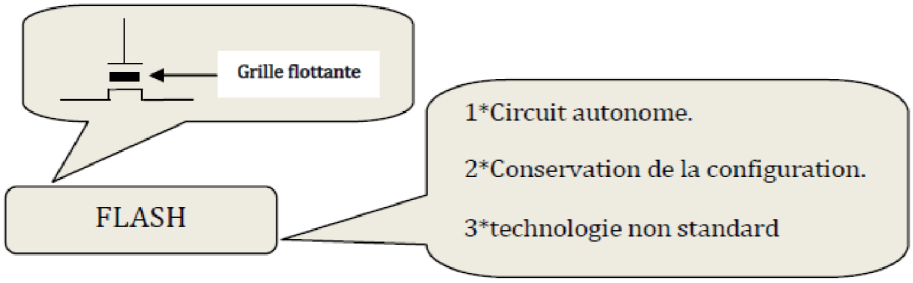


Figure I.12: Caractéristiques des technologies FLASH.

I.2.10.3.Technologie à base de RAM (XILINX et ALTERA)[3]

Cette technologie permet d'avoir une reconfiguration rapide des FPGAs. Les points de connexions sont des ensembles de transistors commandés. L'inconvénientmajeur de cette technologie c'est qu'elle nécessite beaucoup de place et il est nécessaire de sauvegarder le design du FPGA dans une autre mémoire Flash. La figure **I.13**

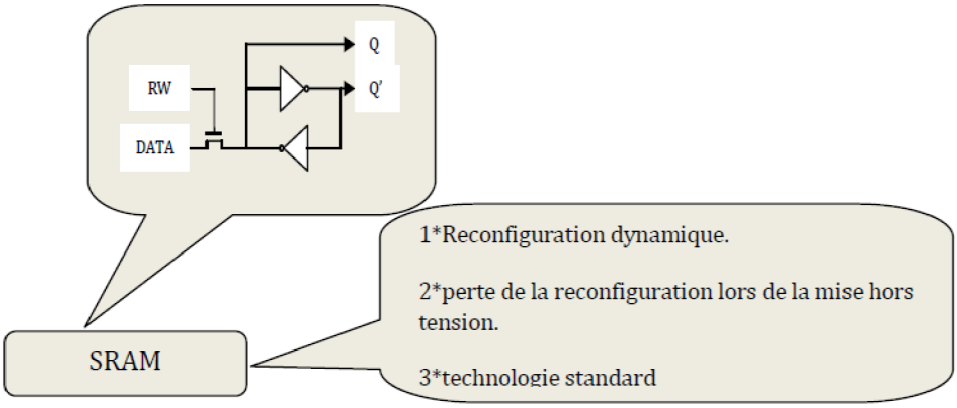


Figure 1.13: Caractéristiques des technologies SRAM.

I.2.11.Avantages et inconvénients des FPGA[3]

Les avantages et les inconvénients des FPGA sont multiples dont on cite :

Les Avantages

- Technologie « facile » à maîtriser.
- Reprogrammable.
- Temps de développement réduit.
- **I**déal pour le prototypage (rapide).
- Coût peu élevé.
- Parallélisme de traitement.
- Flexibilité et la possibilité de réduire
- **F**ortement les délais de développement et commercialisation.
- La reconfiguration, parfois en temps réel.

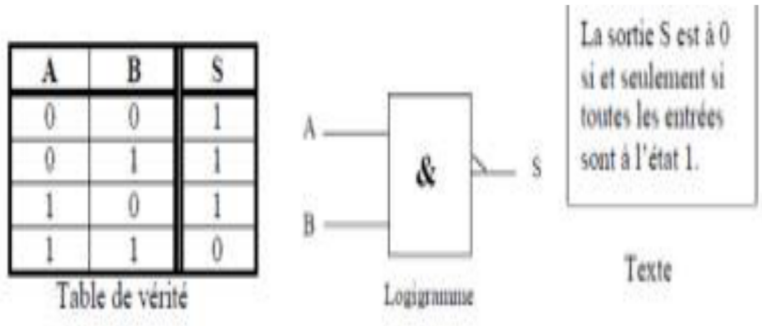
Les Inconvénients

- Performances no optimisées.
- **T**emps de réponse long par rapport aux **ASIC.**

I.3.VHDL[17]

I.3.1.Introduction[17]

Pour décrire une fonction logique combinatoire ou séquentielle, il existe plusieurs représentations : Table de vérité ou table de transitions, Logigrammes, chronogrammes,diagramme d'états, textes etc..



A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

Figure :1.14:fonction logique combinatoire ou séquentielle

Le langage VHDL permet de décrire le comportement d'un circuit logique. Ce langage est utilisé pour développer les circuits logiques programmables. A partir d'un fichier VHDL, un outil de développement informatique nous permettra la programmation d'un circuit logique programmable.

Exemple : la fonction NAND précédente sera décrite comme suit en VHDL:

```
Library IEEE ;
USE ieee.std_logic_1164.all ;
USE work.std_arith.all ;
entity PORTE is
Port ( A, B : in std_logic ;
S : out std_logic ) ;
end PORTE
architecture ARCH_PORTE of porte is
begin
S<= A nandB ;
end ARCH_PORTE ;
```

L'abréviation**VHDL**signifie**VHSIC** **H**ardware **D**escription **L**anguage (**VHSIC** :**V**ery**H**igh

Speed **I**ntegrated**C**ircuit). Ce langage a été écrit dans les années 70 pour réaliser la simulationde circuits électroniques. On l'a ensuite étendu en lui rajoutant des extensions pour permettrela conception (synthèse) de circuits logiques programmables (**P.L.D. P**rogrammable **L**ogic**D**evice).

Auparavant pour décrire le fonctionnement d'un circuit électronique programmable les techniciens et les ingénieurs utilisaient des langages de bas niveau (**ABEL, PALASM,ORCAD/PLD**,..) ou plus simplement un outil de saisie de schémas.

Actuellement la densité de fonctions logiques (portes et bascules) intégrée dans les **PLDs**est telle (plusieurs milliers de portes voire millions de portes) qu'il n'est plus possible d'utiliser:
les outils d'hier pour développer les circuits d'aujourd'hui.
Les sociétés de développement et les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits. Ils ont donc créé des langages dits de haut niveau à savoir **VHDL** et**VERILOG**. Ces deux langages font abstraction des contraintes technologies des circuits**PLDs**. Ils permettent au code écrit d'être portable, c'est à dire qu'une description écrite pourun circuit peut être facilement utilisée pour un autre circuit.

I.3.2. Relation entre une description VHDL et les circuits logiques programmables.

L'implantation d'une ou de plusieurs descriptions VHDL dans un PLD va dépendre de l'affectation que l'on fera des broches d'entrées / sorties et des structures de base du circuit logique programmable. L'affectation peut se faire de plusieurs manières :

I.3.2.1.L'affectation automatique.[17]

On laisse le synthétiseur et le Fitter du fabriquant (« Fondeur » : Xilinx, Lattice, Altéra,Cypress...») du circuit implanter la structure correspondant à la description VHDL. Les numéros de broches seront choisis de façon automatique.

I.3.2.2.L'affectation manuelle.[17]

On définit les numéros de broches dans la description VHDL ou sur un schéma bloc définissant les liaisons entre les différents blocs VHDL ou dans un fichier texte propre au fondeur. Les numéros de broches seront affectés suivant les consignes données.

I.3.2.2.Structure d'une description VHDL simple.[17]

Une description **VHDL** est composée de 2 parties **indissociables** à savoir :

- ***L'entité*** (***ENTITY***), elle définit les entrées et sorties.

- ***L'architecture*** (***ARCHITECTURE***), elle contient les instructions **VHDL** permettant deréaliser le fonctionnement attendu.

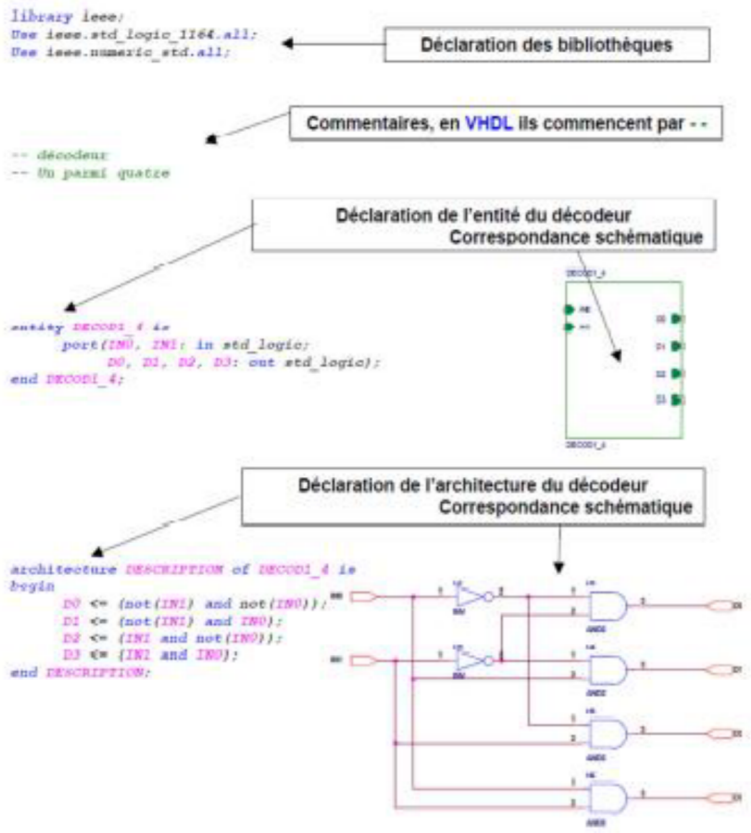


Figure :1.15:**Un décodeur 1 parmi 4.**

I.3.3.1. Déclaration des bibliothèques.[17]

Toute description **VHDL** utilisée pour la synthèse a besoin de bibliothèques. L'**IEEE** (**I**nstitut of **E**lectrical and **E**lectronics**E**ngineers) les a normalisées et plus particulièrement la bibliothèque**IEEE1164**. Elles contiennent les définitions des types de signaux électroniques,des fonctions et sous programmes permettant de réaliser des opérations arithmétiques etlogiques,...

Library ieee;

Use ieee.std_logic_1164.all;

Use ieee.numeric_std.all;

***Use ieee.std_logic_unsigned.all*;**

-- cette dernière bibliothèque est souvent utilisée pour l'écriture de compteurs

La directive ***Use*** permet de sélectionner les bibliothèques à utilizer

I.3.3.2. Déclaration de l'entité et des entrées / sorties (I/O).[17]

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

Syntaxe:

entity NOM_DE_L_ENTITE is

port (Description des signaux d'entrées /sorties …);

end NOM_DE_L_ENTITE;

Exemple :

entity SEQUENCEMENT is

port (

CLOCK : in std_logic;

RESET : in std_logic;

Q : out std_logic_vector(1 downto 0)

);

end SEQUENCEMENT;

Remarque : Après la dernière définition de signal de l'instruction ***port*** il ne faut jamais mettre de point virgule.

L'instruction port :

Syntaxe:

NOM_DU_SIGNAL : sens type;

Exemple:

CLOCK: in std_logic;

BUS : out std_logic_vector (7 downto 0);

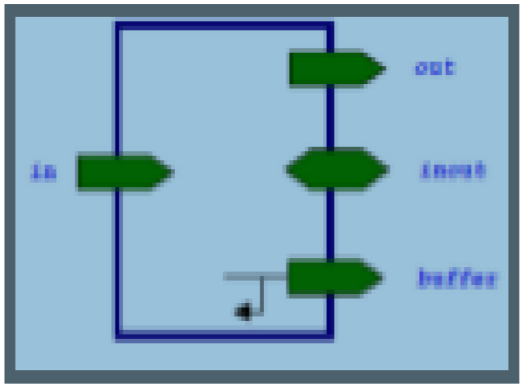
On doit définir pour chaque signal : le NOM_DU_SIGNAL, le sens et le type.

I.3.3.3. Le NOM_DU_SIGNAL.[17]

Il est composé de caractères, le premier caractère doit être une lettre, sa longueur est quelconque, mais elle ne doit pas dépasser une ligne de code. **VHDL** n'est pas sensible à la «casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.

I.3.3.4. Le SENS du signal.[17]

- ***in*** : pour un signal en entrée.
- ***out*** : pour un signal en sortie.
- ***inout***: pour un signal en entrée sortie
- ***buffer*** : pour un signal en sortie mais utilisé comme entrée dans la description.



4

Figure:1.16:Le SENS du signal.

I.3.4.domaine d'utilisations du langage VHDL[17]

- Concevoir des ASICs.
- Programmer des composants programmables du type PLD, CPLD, FPGA.
- Concevoir des modèles de simulations numériques ou des bancs de tests.

I.3.5.Le *TYPE*

Le ***TYPE*** utilisé pour les signaux d'entrées / sorties est :

- le***std_logic***pour un signal.
- le***std_logic_vector***pour un bus composé de plusieurs signaux.

Par exemple un bus bidirectionnel de 5 bits s'écrira :

LATCH :inoutstd_logic_vector (4 downto 0) ;

Où ***LATCH(4)*** correspond au **MSB** et ***LATCH(0)*** correspond au **LSB**.

Les valeurs que peuvent prendre un signal de type ***std_logic***sont :

- ***'0' ou 'L'*** : pour un niveau bas.
- ***'1' ou 'H'*** : pour un niveau haut.
- ***'Z'*** : pour état haute impédance.
- ***'-'*** : Quelconque, c'est à dire n'importe quelle valeur.

I.3.6. Déclaration de l'architecture correspondante à l'entité :[17]

description du fonctionnement.[17]

L'architecture décrit le fonctionnement souhaité pour un circuit ou une partie du circuit.En effet le fonctionnement d'un circuit est généralement décrit par plusieurs modules

VHDL.Il faut comprendre par module le couple *ENTITE/ARCHITECTURE*. Dans le cas de simples**PLDs**on trouve souvent un seul module.

L'architecture établit à travers les instructions les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire, séquentiel voire les deux

séquentiel et combinatoire.

Exemples :

```
-- Opérateurs logiques de base
entity PORTES is
port (A,B :in std_logic;
Y1,Y2,Y3,Y4,Y5,Y6,Y7:outstd_logic);
end PORTES;
architecture DESCRIPTION of PORTES is
begin
Y1 <= A and B;
Y2 <= A or B;
Y3 <= A xor B;
Y4 <= not A;
Y5 <= A nand B;
Y6 <= A nor B;
Y7 <= not(A xor B);
```

```
end DESCRIPTION;
-- Décodeurs 7 segments
entity DEC7SEG4 is
port (DEC :in std_logic_vector(3 downto 0);
SEG:outstd_logic_vector(0 downto 6));
end DEC7SEG4;
architecture DESCRIPTION of DEC7SEG4 is
begin
SEG <= "1111110" when DEC = 0
else "0110000" when DEC = 1
else "1101101" when DEC = 2
else "1111001" when DEC = 3
else "0110011" when DEC

else "1011011" when DEC = 5
else "1011111" when DEC = 6
else "1110000" when DEC = 7
else "1111111" when DEC = 8
else "1111011" when DEC = 9
else "-------";
end DESCRIPTION;
```

I.3.7. La vérification d'une conception VHDL[17]

Il existe différentes méthodes pour développer des systèmes numériques incluent lesprocessus d'optimisation, de vérification et de validation des systèmes. Les tests de développement du système sont principalement effectués en utilisant deux méthodes.

-Premièrement la fonctionnalité et les performances du système sont observées et réglées à l'aide d'un logiciel outils de simulation. Le niveau de confiance obtenu à partir de la simulation logicielle est spécifique à l'application et dépend de la disponibilité et de la précision du système.

-Deuxièmement, la conception est déployé sur une plate-forme cible pour vérifier la fonctionnalité et mesurer la performance dans des conditions plus réalistes. Cette approche est également connue sous le nom de test de matériel.

Dans ce travail nous nous examinons deux méthodes pour la vérification : la Cosimulationet FPGA in the loop la figure **I.17**.

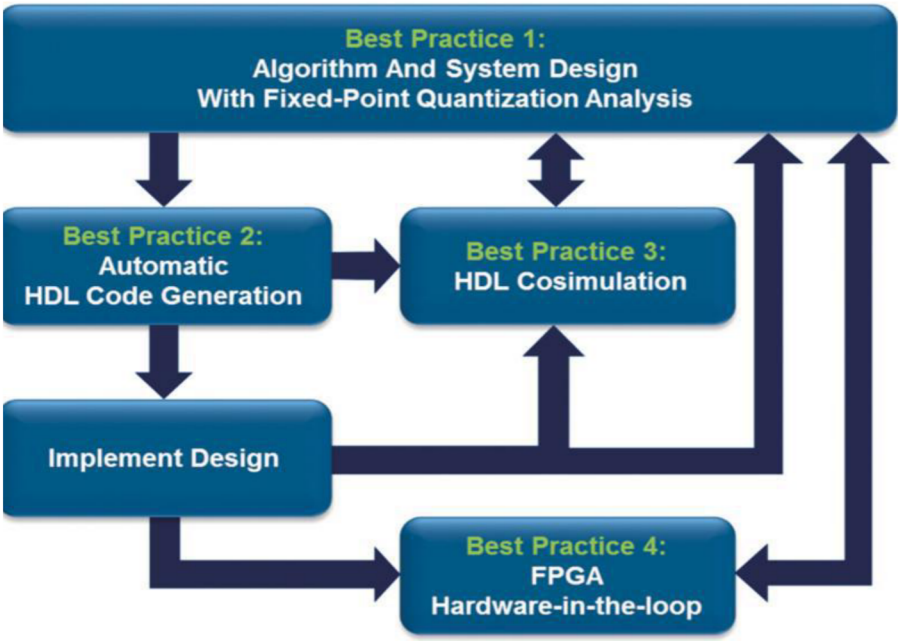


Figure: I.17:Principe de la Co-simulation et de FPGA in the loop.

I.3.8. La Co-simulation[17]

La Co-simulation est basée sur la communication entre deux simulateurs liés à deux logiciels différents, l'un numérique et l'autre analogique. Les modèlessont conjointement exécutés par ces deux simulateurs, chaque simulateur modélisant une partie spécifique du circuit à concevoir ou de son environnement. Une interface de Co-simulation permet l'échange de données entre les deux simulateurs tout en respectant les contraintes de types et de tailles mais surtout respectant la synchronisation temporelle des deux simulateurs.

Comme exemple la Co-Simulation entre MODELSIM/MATLAB est montré dans la Figure**I.18**.

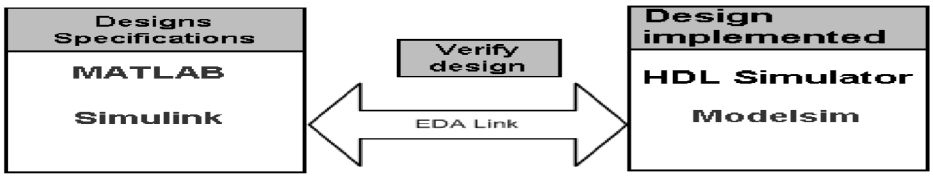


Figure :I.18: la Co-simulation entre matlab et modelsim.

I.3.9.Avantages et Inconvénients de la Co-simulation[17]

-La Co simulation

-Les avantages

- Bibliothèques riches en composants analogiques ou de puissance.

-Les inconvénients

- Deux simulateurs sont nécessaires et doivent être maîtrisés.

-Temps de simulation important.

I.4.Conclusion[17]

Dans la première partie de ce chapitre on a discuté les circuits logiques programmablesFPGAs et on a abordé leurs principes de fonctionnement ainsi que leurs différents composants constituent ces circuits, leurs avantages et inconvenants. Dans la deuxième partie, de ce chapitre on a vu les langages HDL qui sont utiliser pour programmer les circuits FPGAs, et en particulier le langage VHDL, Nous avons également abordé la chose importante incorporée dans le processus de vérification qui est la Co-simulation et FPGA-in-the-loop.

CHAPITRE:II LES DIFFERENTES TECHNIQUES DE LA M.L.I

Chapitre:II LES DIFFERENTES TECHNIQUES DE LA M.L.I

II.1. Introduction:[1][2]

La technique de modulation de largeur d'impulsion (Pulse Width Modulation PWM) consiste à générer un signal carré avec un rapport cyclique modulé en fonction d'un signal de commande. Le signal généré peut servir à commander un circuit de puissance à découpage (pont en H), associé à un filtrage passe-bas inductif, pour générer une onde sinusoïdale ou d'une autre forme. La technique est utilisée dans les onduleurs monophasés, diphasés ou triphasés . Le même principe est utilisé dans les amplificateurs Audio de classe D.

Cette page montre comment générer des signaux par MLI avec un Arduino Due 32 bits. Pour faire la même chose avec un Arduino 8 bits (Uno, Mega ou autre), consulter Génération d'un signal par modulation de largeur d'impulsion. L'arduino Due permet d'atteindre des fréquences plus de 10 fois supérieures à un Arduino 8 bits. L'objectif est d'alimenter un haut parleur (ou une autre charge inductive) avec un signal audio, par l'intermédiaire d'un pont en H piloté par le signal PWM.

II.2.Définition:[1][2]

On reste en numérique, les signaux ont toujours une valeur LOW ou HIGH et le principe est de construire un signal qui est alternativement LOW et HIGH et de répéter très vite cette alternance. La DEL est donc alternativement allumée et éteinte mais le cycle est tellement rapide que la persistance rétinienne nous donne l'illusion d'une DEL allumée en permanence. Nous avons déjà rencontré ce phénomène dans « La programmation, qu'est ce que c'est » où, en tentant de faire clignoter une DEL, on obtenait un allumage permanent mais d'une luminosité moindre.

Prenons par exemple une période de 10ms, soit une fréquence de 100Hz. Si la DEL est allumée pendant 5ms et éteinte pendant 5ms, comme sur la figure ci-dessous, l'impression sera une luminosité de 50% de la luminosité maximum.

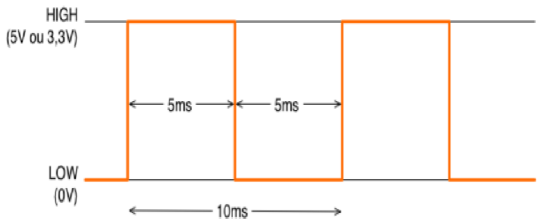


Figure:2.1:PMW à 50%
La fréquence est de 100Hz, le rapport cyclique de 50%

Si la DEL est allumée pendant 1ms et éteinte pendant 9ms, l'impression sera une luminosité de 10% comme sur la figure ci-

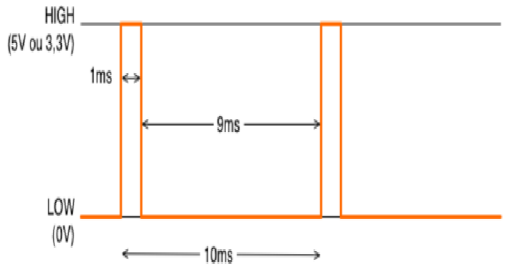


Figure:2.2:PWM à 10%
La fréquence est de 100Hz et le rapport cyclique de 10%.

Le pourcentage de temps passé à l'état HIGH sur la période du signal est appelé le *rapport cyclique*. Il varie donc de 0%, le signal est tout le temps LOW, à 100%, le signal est tout le temps HIGH.

II.3.ConfigurationPWM:[1][2]

Pour illustrer comment **configurer** un **PWM**, vous allez faire varier l'intensité d'une led. Le principe va consister à envoyer le signal d'un **PWM** sur une led avec une fréquence suffisamment élevée pour que l'œil ne soit pas capable de distinguer les moments où la led est allumée ou éteinte.

II.4.calculer le PWM :[1][2]

La « valeur » de la **PWM**est sur un octet. Le maximum est donc de 255, ce qui correspond à 100% de rapport cyclique. Pour fixer le rapport cyclique à n%, il suffit de faire une règle de 3 : n/100 = p/255. Donc p = 255 x n / 100.

II.5.changer la fréquence du PWM :[1][2]

La modification de la **fréquence** se fait facilement en changeant la valeur du "prescaler" du compteur qui divise l'horloge. Grâce à l'environnement de programmation simplifié, un**PWM** s'obtient sur une Arduino classique avec la fonction analogWrite utilisé sur une patte numérique Digital 3, 5, 6, 9, 10 ou 11.

II.6. Modulation de largeud'impulsion:[1][2]

II.6.1.fonctionnement de la modulation de largeur d'impulsion (MLI):

La figure suivante montre le fonctionnement de la modulation de largeur d'impulsion (MLI). Une porteuse triangulaire est comparée à un signal de consigne, par exemple une sinusoïde. Le signal de consigne doit avoir une fréquence bien plus petite que la porteuse. Le signal de sortie est au niveau haut (disons 5 V) lorsque la consigne est supérieure à la porteuse, au niveau bas (0 V) dans le cas contraire. On considère le cas d'un signal de consigne à valeurs positives. Pour traiter un signal alternatif, il suffira de lui appliquer un décalage

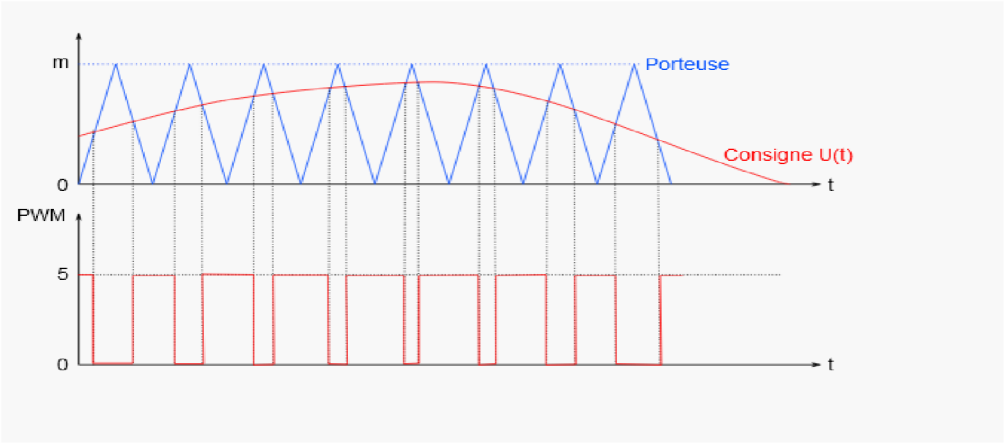


Figure:2.3:fonctionnement de la modulation de largeur d'impulsion (MLI).

Le signal PWM obtenu doit subir un filtrage passe-bas pour en extraire le signal de consigne. Pour comprendre le principe de cette restitution, considérons le cas d'une consigne constante égale à $U(t)=U_0$. Le signal PWM est alors un signal carré dont le rapport cyclique est $\alpha=U_0/m$, où m est la valeur maximale de la porteuse. La moyenne de ce signal carré est précisément égale à U_0.

Lorsque la consigne est lentement variable par rapport à la porteuse, il faudra appliquer un filtrage passe-bas pour restituer les variations de basses fréquences de la consigne. En pratique, le signal PWM est utilisé pour commander un circuit de puissance travaillant en commutation, et le filtrage passe-bas est assuré par une bobine en série avec la charge. Pour commander un pont en H, il faudra aussi disposer du signal complémentaire, obtenu avec une porte NON. Le générateur PWM de l'Arduino Due nous permettra d'obtenir directement ce signal complémentaire.

II.6.2.Les usages les plus fréquents:[1][2]

- La conversion numérique-analogique
- Les amplificateurs de classe D
- Les alimentations à découpage
- Les variateurs de vitesse
- Les onduleurs
- Les redresseurs
- Plus généralement tous les dispositifs d'électronique de puissance utilisant des composants en commutation à base de semi-conducteurs de type MOSFET, IGBT, GTO. En effet, fondamentalement, les cellules de commutation n'ont que deux états possibles, l'état haut ou l'état bas. Cette structure intrinsèque rend naturelle l'utilisation de PWM pour leur commande. Il est aussi possible de faire de la transmission de données par cette méthode.

II.7.Les types de MLI:[1][2]

II.7.1.La MLI numérique:

Le principe est de créer un signal logique (valant 0 ou 1), à fréquence fixe mais dont le rapport cyclique est contrôlé numériquement, la valeur moyenne de ce signal étant une grandeur analogique, égale au produit du rapport cyclique par l'amplitude maximale du signa**[1][2]**

II.7.1.2.Principe de réalisation:

Généralement on réalise ce type de fonction à l'aide d'un microcontrôleur.
Les microcontrôleurs modernes (ARM Cortex M, PIC, etc.) disposenttous de périphériques dédiées à la génération de PWM.**[1][2]**

Pour les microcontrôleurs plus anciens (8051, 68HC11), ilest possible de détourner le fonctionnement d'un TIMER pour le transformer en générateur de PWM. Enfin en cas d'utilisation d'un microprocesseur, la PWM est généralement émulée.

Dans tous les cas, le principe du périphérique ou du programme d'émulation consiste à générer un signal périodique à assez haute fréquence (généralement supérieure à 20KHz pour éviter d'être dans la gamme audible, bien que cela ne soit pas obligatoire) appelé généralement porteuse (mais l'appellation est impropre) et à générer des impulsions à 1, dont le rapport entre la durée et la période, est égal à la valeur moyenne souhaitée par rapport à la valeur maximum du signal

$$\frac{Ton}{T} = \frac{N}{Nmax} \qquad (2.1)$$

où Ton est la durée de l'impulsion, T la période du signal, N la valeur du signal à l'instant t (analogique ou numérique) et Nmax la valeur maximum que peut prendre N.

Dès lors la tension moyenne générée par le signal de PWM (variant entre E et 0) devient

$$V_{moy} = \frac{1}{T}\left(\int_0^{T_{on}} E\,dt + \int_{T_{on}}^{T} 0\,dt\right) = \frac{1}{T}\cdot E\cdot T_{on} = E\cdot\frac{T_{on}}{T} = E\cdot\frac{N}{N_{max}} \qquad (2.2)$$

II.7.2.MLI « intersective » ou MLI analogique:**[1][2]**

C'est le plus classique. Elle consiste à comparer le modulateur au triple support en général. La valeur du signal de sortie est(1) si le modulateur est plus grand que la porteuse, (0)sinon ; Ainsi, le signal de sortie change d'état à chaque jonction entre la modulation et la porteuse

Cette méthode s'étend bien à l'application analogique : il suffit d'un générateur triangulaire et d'un comparateur. Il existe de nombreux circuits intégrés personnalisés

Les sous-espèces sont classées de plusieurs manières :

Echantillonnage analogique ou numérique, selon que le modulateur et le comparateur sont continus ou discrets Avec un support triangulaire centré ou dentelé (gauche ou droite);Asynchrone ou synchrone, selon que le modulateur et l'émetteur sont complètement multifréquences ou non.

Nous portons une attention particulière au lecteur à ne pas confondre la synchronisation PWM avec les moteurs synchrones ou asynchrones.

En fait, même s'ils portent le même nom, au moins leurs définitions sont très différentes.

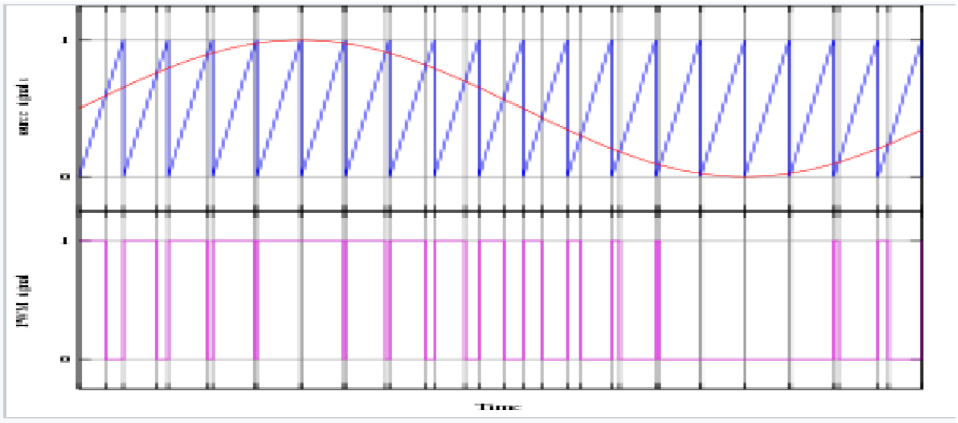


Figure:2.4:La mli a porteuse en dents de scie.

II.7.3.MLI Par injection d'harmoniques d'ordre 3:[1][2]

La technique décrite ci-après est utile pour la commande de transformateurs de tension car elle consiste à produire une tension sinusoïdale en sortie dudit transformateur.

Les PWM à injection de troisième harmonique sont des variantes simples mais efficaces du PWM classique (appelé SPWM pour PWM sinusoïdal) décrit précédemment.

Leur abréviation dans la littérature est appelée THIPWM et les deux principalement utilisées sont THIPWM1/6 et THIPWM1/4.

Historiquement THIPWM1/6 a été décrit pour la première fois par Buja en 1975 afin d'augmenter la région linéaire de l'indice de modulation et d'augmenter la qualité harmonique des solutions.

L'indice de modulation est toujours classique (avec SPWM) entre 0 et 1.

On voit que l'indice de modulation augmente de 15% grâce à THIPWM1/6 et de 12% grâce à THIPWM1/4.

Cette deuxième méthode malgré une région linéaire moins large a l'avantage de réduire le taux de distorsion harmonique par rapport à la variante 1/6.

II.7.4.MLI aléatoire:[1][2]

Le principe du PWM aléatoire (Random PWM ou RPWM) est de générer un grand nombre d'harmoniques afin de réduire le bruit perçu.Plus précisément, le spectre de fréquences est réparti sur toutes les fréquences.Pour ce faire plusieurs méthodes ont été décrites dans la littérature.mais la technique la plus simple reste de faire varier aléatoirement la fréquence de la porteuse triangulaire entre une fréquence maximale et une fréquence minimale définie par l'utilisateur. Cette technique permet ainsi de ne pas avoir des pics de puissance sur des harmoniques précis mais de les étaler sur tout le spectre.

Cette configuration apporte ce qui semble être un paradoxe car mathématiquement, la puissance des harmoniques est en valeur absolue plus élevée. Mais l'oreille humaine n'est pas sensible à la puissance du bruit en tant que telle, elle est sensible à la puissance de certaines fréquences. Ainsi, étaler le spectre ne crée plus un bruit stridant mais un sentiment d'écoulement de sable fin bien plus agréable. Il est à noter que les RPWM ne peuvent fonctionner qu'à une fréquence de commutation élevée par rapport à la fréquence électrique afin de respecter la consigne de tension.

II.7.5.MLI Discontinue:[1][2]

Le principe des MLI discontinue (Discontinuous PWM or DPWM en anglais) a été introduit pour la première fois par Depenbrock en 1977 en introduisant ce qui sera appelé plus tard la DPWM1. Le principal avantage des DPWM est de permettre de réduire les pertes par commutations des convertisseurs d'électronique de puissance en empêchant périodiquement une phase de commuter. Ce qui a pour effet de diminuer les pertes par commutation. Cependant, les mauvaises performances dans les faibles indices de modulation (problèmes d'impulsion trop étroites et la mauvaise qualité de la forme d'onde de courant) ainsi que la complexité de mise en œuvre de cette première stratégie ont limité l'application de cette méthode. Néanmoins, d'autres travaux plus récents ont conduit à d'autres mises en œuvre équivalentes et à des simplifications des stratégies DPWM.

II.7.6.MLI « Vecteur spatial »:[1][2]

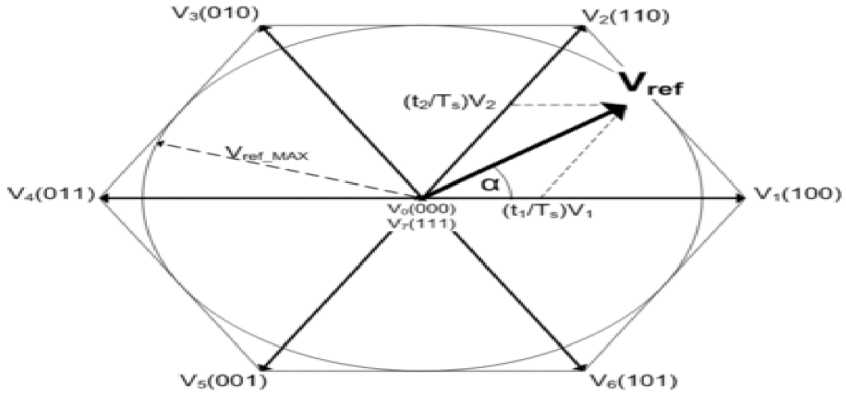


Figure:2.5:vecteur spatial

La MLI dite space vector (vecteur spatial, abrégé en SVM dans la littérature) est surtout applicable aux variateurs de vitesse triphasés sans neutre.Elle consiste à considérer globalement le système triphasé, et à lui appliquer une transformée de Concordia pour se ramener dans le plan (Vα, Vβ). Le système triphasé de tensions à générer pour la durée d'échantillonnage en cours peut alors être représenté comme un unique vecteur dans ce plan (voir aussi commande vectorielle).Ce vecteur n'est pas directement réalisable par les interrupteurs du variateur, mais on peut chercher les trois configurations les plus proches (situées sur les sommets et au centre de l'hexagone), et les appliquer successivement pendant une fraction adéquate de la période d'échantillonnage, de façon à obtenir en moyenne le vecteur recherché.En modulation sinusoïdale, elle donne des résultats similaires (mais néanmoins meilleurs) à la MLI intersective à porteuse triangulaire centrée. Néanmoins, elle peut être plus facile à implanter dans un microcontrôleur, et, disjointe d'harmonique 3, elle permet de maximiser la puissance disponible, ce qui justifie son usage.

II.7.7.MLI « précalculée »:[1][2]

Les MLI précalculées aussi appelées MLI hors ligne ou Optimal Pulse pattern (OPP) en anglais est surtout utilisée lorsque, du fait d'une fréquence porteuse faible, on a besoin d'optimiser le spectre du signal généré. Le motif du signal de sortie est prédéterminé (hors ligne) et stocké dans des tables qui sont ensuite lues en temps réel. De fait, ces MLI sont toujours synchrones (la fréquence porteuse est exactement multiple de la fréquence de la modulante), en effet l'optimisation réalisée en amont

génère des angles de commutation bien précis qui ne peuvent être réalisés que de manière synchrone. De plus, le synchronisme est une condition nécessaire pour avoir un spectre harmonique constante et éviter la générations d'harmoniques plus faibles que le fondamental qui sont difficiles à filtre. En pratique, ce type de MLI ne peut être réalisé qu'en numérique, ou en analogique grâce à une carte FPGA

II.8. Techniques de commande de MLI aux répetition:

II.8.1.Principe de la commande:[1][2]

Le signal de référence de commande est obtenu par une conversion de signal sinusoïdal en rapport cyclique dont il est divisé en m segments ou chaque segment est convertit en une forme numérique en n bits pour représenter l'état de fermeture et d'ouverture des interrupteurs du pont

(ton, toff) .

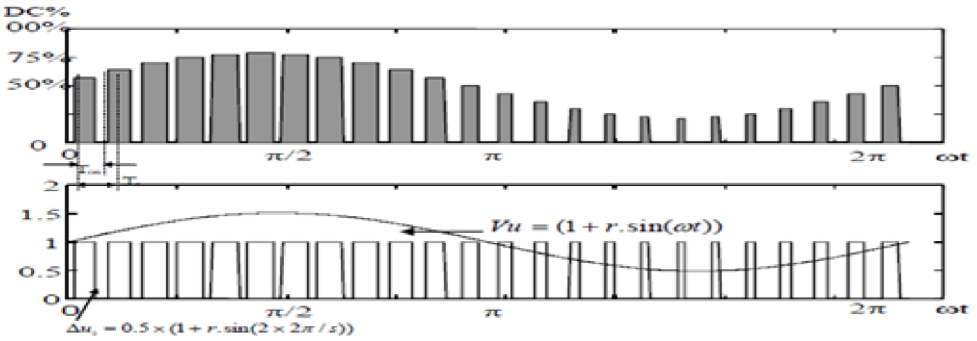


Figure: 2.6: Principe de la commande PWM utilisé.

La tension de phase en un point M.L.I. du pont est donnée par :

Vu=Vdc.Δu

Δu: le rapport cyclique de la phase u (% de mise en ouverture)

Δu=ton/Ts

Ts: la duré d'un segment (période d'échantillonnage).

La tension du moteur est maximale quand le rapport cyclique Δu varie de 0% à 100% (lepourcentage de modulation en profondeur K=100%) et elle est minimale (nulle) quand le rapport cyclique Δu ne varie pas c.à.d. fixé à 50% (la profondeur K= 0

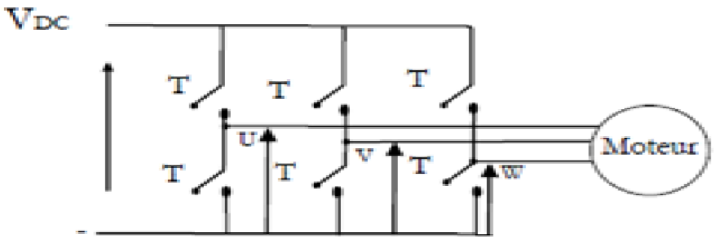


Figure: 2.7: Schéma de principe du convertisseur.

II.8.1.2. Etude mathématique de M.L.I:[1][2]

Les coefficients de série de Fourrier de la tension de branche sont donnés par les deux formules suivantes

$$a_n = \frac{E}{n\pi}\left(1 + \sum_{i=1}^{m-1}(-1)^i \times \cos(n \times \theta_i)\right)$$

$$b_n = \frac{E}{n\pi}\left(\sum_{i=1}^{m-1}(-1)^{i+1} \times \sin(n \times \theta_i)\right)$$

a b n n,=Coefficients de Fourrier

θi=Les instants de commutation

m =Nombre d'instants de commutation par période

n =Rang des harmoniques

E =Valeur de la tension DC appliquée.

$$\begin{cases} \theta_{2p} = k \times 2\pi / 24 \\ \theta_{2p+1} = \theta_{2p} + \pi / 24(\sin k \times 2\pi / 24 + 1) \end{cases}$$

Avec k ∈N

Dans notre cas d'utilisation d'un pont triphasé de puissance, nous utilisons les équations des tensions de lignes (tension entre deux points M.L.I. du pont) qui sont données par les équations 5 et 6

$$a_n = \frac{\sqrt{3}E}{n\pi}\left(1 + \sum_{i=1}^{m-1}(-1)^i \times \cos(n \times \theta_i)\right) \quad ou \quad n \neq 3 \times k$$

$$b_n = \frac{\sqrt{3}E}{n\pi}\left(\sum_{i=1}^{m=1}(-1)^{i+1}\times\sin(n\times\theta_i)\right) \quad ou \quad n \neq 3\times k$$

Dans la figure 2.3 on présente le spectre de l'onde PWM pour les deux fréquences 10 Hz et45 Hz pour un nombre de segments (échantillon) égale à 24 donc m =48.

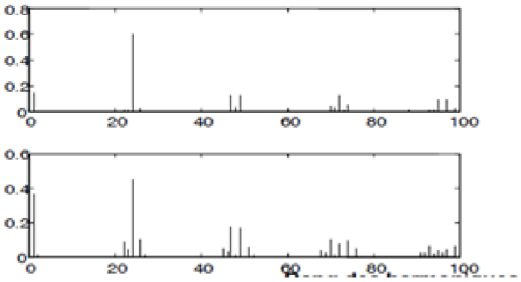


Figure: 2.8: Spectre de signal modulé en PWM pour f=10 et 45 Hz.

II.8.2.3.Commande par hystérésis:[1][2]

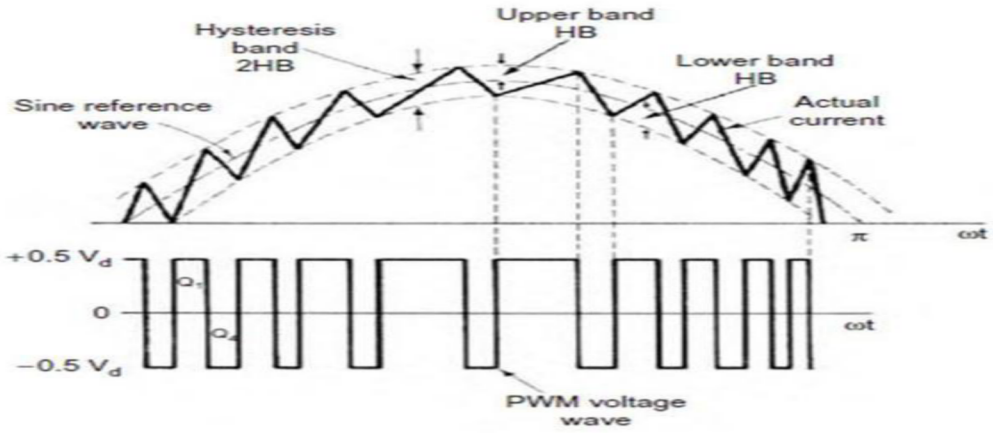


Figure:2.9:principe du régulateur de courant à bande d'hysrésis.

Cette méthode consiste à élaborer le signal MLI directement à partir de la grandeur à contrôler, par des décisions de type tout ou rien. Son nom de "Hystérésis", vient du fait que le choix de la commande se décide grâce à plusieurs cycle d'hystérésis (une pour chaque phase). En effet, en pratique, une comparaison entre le courant de commande et le courant de sortie est faite. Ensuite cette différence, au delà d'un certain seuil fait changer la ou les cellules de commutation de l'état haut à l'état bas et vice versa.

Les avantages sont la très grande simplicité et le temps de réponse minimal aux perturbations. L'inconvénient majeur est l'absence de contrôle de la fréquence de commutation des transistors et sa forte non-linéarité intrinsèque, ce qui rend .

II.9.LA M.L.I. REPETEE:|6||7||8||15|

II.9.1. DEFINITIONT : C'estune technique basée sur la répétition des segments de données d'un signal de référence haché, dont chaque segment est l'image en rapport cyclique (pourcentaged'utilisation ou de mise en ouverture) de la tension de sortie (conversion tension alternative en rapport cyclique). aussi, un nouveau concept de génération de la M.L.I. répétée, basé sur une description en V.H.D.L. et une mémoire contenant les données pré introduites, cette description assurent également toutes les parties qui assurent les synchronisations et les traitements nécessaires.

II.9.2.Principe de variation de la fréquence du signal : :[6][7][8][15]

Pour produire un signal de basse fréquence (relativement à la fréquence de référence) on produit une séquence de répétition de chaque segment n fois ; calculer de telle façon on obtient la fréquence exigée par le système. La figure 4 présente, en pourcentage du rapport cyclique, une onde répétée 3 fois.

La variation de la fréquence est obtenue par la variation de la fréquence de signal de modulation. La fréquence du signal de commande est calculée par la formule suivante:

$F = n \times m \times r \times f$

n: nombre de bits.

m: nombre de segments.

r: nombre de répétitions.

f: fréquence de génération de données.

Pour définir les rapports cycliques des 6 interrupteurs de l'étage de puissance on calcule les données pour (T1, T3, T5) et les interrupteurs d'opposition sont complémentaire

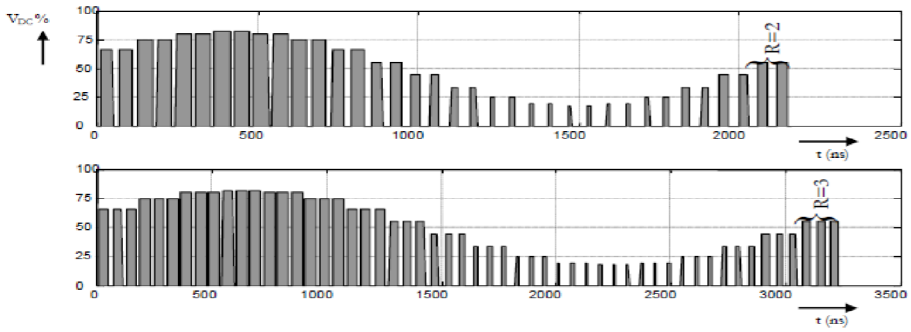


Figure 2.10. L'onde de commande avec répétition

Spectre des différents signaux générés par répétition à partir d'un même signal de référence On présente dans la figure 5 le spectre d'une onde répétée 2 fois en (a) et 3 fois en (b). On remarque pour chaque répétition de signal de référence on obtient un décalage de rang des harmoniques par 24 sans aucune modification dans la forme du spectre, ce qui nous donne une bonne amélioration dans la réponse spectrale

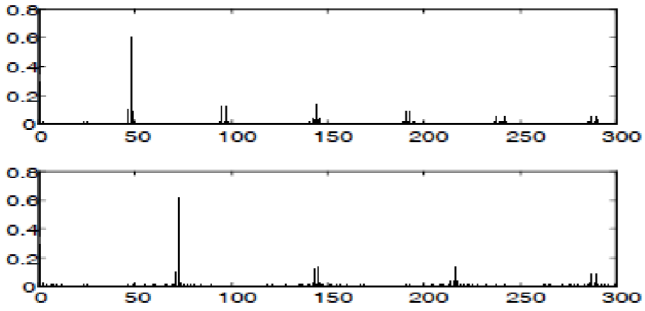


Figure 2.11. Spectre de signal de commande et un signal répété

Pour bien comprendre l'aspect de répétition des signaux M.L.I. (**M.R.L.I**) et leur influence sur le spectre on a développé les formules qui calculent les coefficients de Fourier et les programmés par Matlab et les comparés à celle de la simulation afin de valoriser la formule mathématique obtenue.

II.9.3.Répétition R fois Rappelons la formule calculant les coefficients de Fourier de la tension de branche d'un signal de commande: :[6][7][8][15]

$$\begin{cases} a_n = \dfrac{E}{n\pi}\left(1+\sum_{i=1}^{m-1}(-1)^i \times \cos(n\times\theta_i)\right) \\ b_n = \dfrac{E}{n\pi}\left(\sum_{i=1}^{m-1}(-1)^{i+1} \times sin(n\times\theta_i)\right) \end{cases}$$

a b n n,=coefficient de Fourrier
θ*i*=les instants de commutation

m =nombre d'instant de commutation par période

n =rang des harmoniques

E =valeur de la tension DC appliquée.

$$\begin{cases} \theta_{2p} = k \times 2\pi / 24 \\ \theta_{2p+1} = \theta_{2p} + \pi / 24(\sin k \times 2\pi / 24 + 1) \end{cases} \text{ avec } k \in N$$

Pour une répétition de segment, de ce même signale, de R fois avec un rapport cyclique

Variable θi on peut calculer les coefficients de Fourier de la façon suivante :

$$a_n = \frac{E}{n\pi}\left(\sum_{i=0}^{m-1}\sum_{k=0}^{R-1} \cos\left(n.\frac{2.\pi}{S.R}(R.i+k)\right) - \cos\left(n.(\frac{2.\pi}{S.R}(R.i+k)+\theta_i)\right)\right) \quad c \qquad (2.9)$$

$$b_n = \frac{E}{n\pi}\left(\sum_{i=0}^{m-1}\sum_{k=0}^{R-1} \sin\left(n.\frac{2.\pi}{S.R}(R.i+k)\right) - \sin\left(n.(\frac{2.\pi}{S.R}(R.i+k)+\theta_i)\right)\right) \quad c$$

$$a_n = \frac{E}{n\pi}\left(\sum_{i=0}^{m-1}\sum_{k=0}^{R-1} \cos\left(n.\frac{2.\pi}{S.R}(R.i+k)\right) - \cos\left(n.(\frac{2.\pi}{S.R}(R.i+k)+\frac{\pi}{S.R}(1+r.\sin(\frac{2\pi}{S}i))\right)\right)$$

$$b_n = \frac{E}{n\pi}\left(\sum_{i=0}^{m-1}\sum_{k=0}^{R-1} \sin\left(n.\frac{2.\pi}{S.R}(R.i+k)\right) - \sin\left(n.(\frac{2.\pi}{S.R}(R.i+k)+\frac{\pi}{S.R}(1+r.\sin(\frac{2\pi}{S}i))\right)\right)$$

On peut simplifier la représentation mathématique des a_n et b_n comme suit

$$\begin{cases} a_n = \frac{E}{n\pi}\left(1 + \sum_{k=1}^{R-1}\cos(\frac{k.2.\pi}{S\times R}) + \sum_{i=1}^{m-1}\sum_{k=0}^{R-1} (-1)^i \times \cos\left(n.(\frac{k.2.\pi}{S\times R}.+\theta_i)\right)\right) \\ b_n = \frac{E}{n\pi}\left(\sum_{k=1}^{R-1}\sin(\frac{k.2.\pi}{S\times R}) + \sum_{i=1}^{m-1}\sum_{k=0}^{R-1} (-1)^{i+1} \times \sin\left(n.(\frac{k.2.\pi}{S\times R}.+\theta_i)\right)\right) \end{cases}$$

avec:

$$\begin{cases} \theta_{2p} = 2.p \times \frac{2\pi}{S} \\ \theta_{2p+1} = \theta_{2p} + \frac{\pi}{S\times R}.(1 + r \times \sin((2p+1)\times 2\pi/S)) \end{cases} \text{ avec } k \in N$$

II.9.3.1.Résultats par simulation: :[6][7][8][15]

Sur la figure 2 on présente le spectre simulé d'un signal PWM répété 2 et 3 fois et de la même manière on présente sur la figure 3 du spectre calculé.

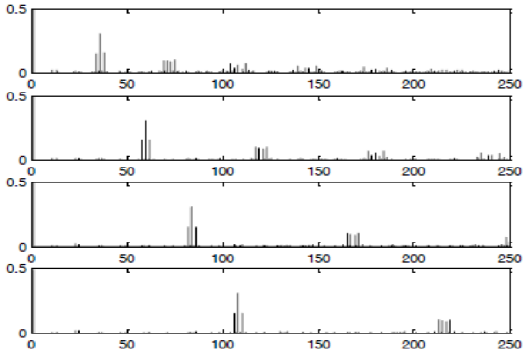


Figure 2.12. Spectre d'un signal répété obtenu par calcul 1 répétition (b) 2 répétitions (c) 3 répétitions)

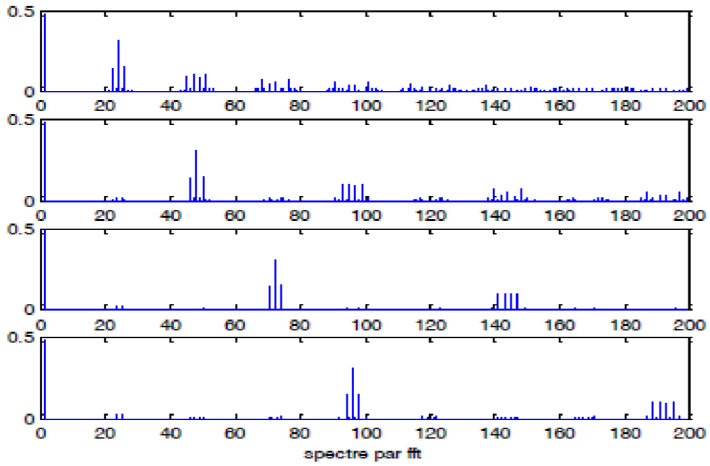


Figure 2.13. Spectre d'un signal répété obtenu par simulation.

II.9.4.Répétition alternée : :[6][7][8][15]

Aussi pour améliorer le pas d'incrémentation à la plage de 50 Hz on propose une nouvelle procédure de répétition alternée c.à.d. répété les segments de signal M.L.I. alterné avec deux valeurs différentes (pour les segments pairs et impairs).

Pour une répétition alternée des segments de R1 et R2 les coefficients de Fourier se calculent par les expressions suivantes : R=(R1+R2)

$$a_n = \frac{E}{n\pi}\left\{\sum_{i=0}^{m/2-1}\left(\sum_{k=0}^{R_1-1}(-1)^i\cos(n.\theta_i)+\sum_{k=0}^{R_2-1}(-1)^i\cos\left(n.(\frac{2.\pi}{S.R}R_1+\theta_i)\right)\right)\right\}$$

$$b_n = \frac{E}{n\pi}\left\{\sum_{i=0}^{m/2-1}\left(\sum_{k=0}^{R_1-1}(-1)^{i+1}\sin(n.\theta_i)+\sum_{k=0}^{R_2-1}(-1)^{i+1}\sin\left(n.(\frac{2.\pi}{S.R}R_1+\theta_i)\right)\right)\right\}$$

II.9.4.1.Résultats par simulation : :[6][7][8][15]

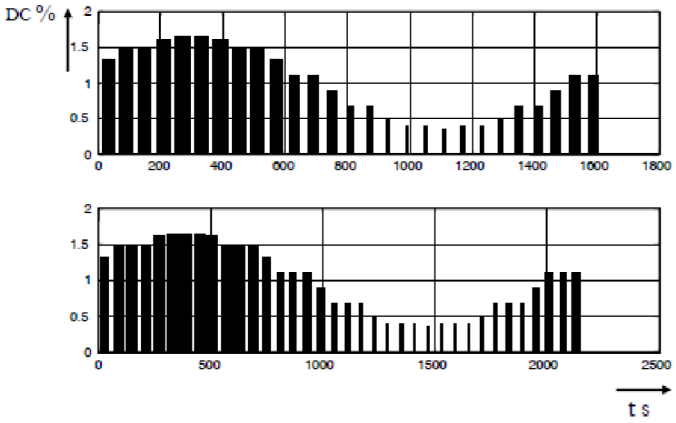


Figure 2.14. Signal de commande avec répétition alternée.

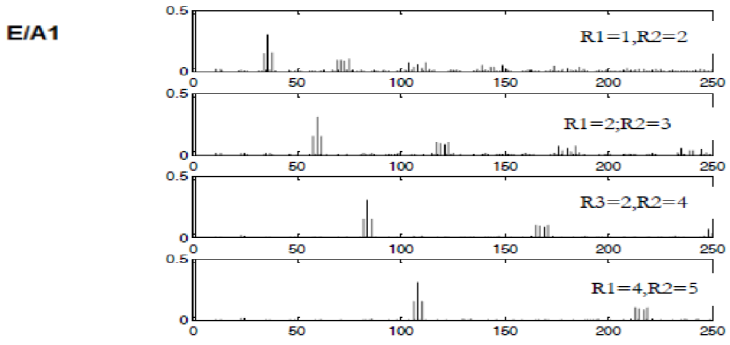


Figure: 2.15: Spectre d'un signal alternativement répété obtenu par formule.

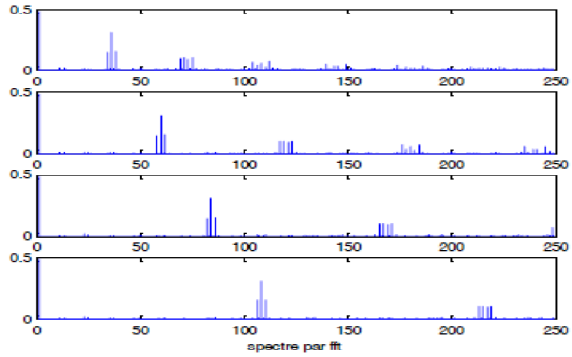


Figure :2.16: Spectre d'un signal alternativement répété obtenu par simulation.

On remarque que l'application de la répétition alternée provoque une légère influence sur le spectre avec l'apparition des sous porteuses multiples de nombre de segments divisé par deux. Mais vu leurs valeurs négligeables on peut considérer cette procédure comme une nouvelle méthode de génération des signaux PWM et pourrait être utilisée pour une grande précision de la fréquence, selon l'équation ;.

F=2f/b(b×s(R1+R2)

II.9.4.2.Facteur de perte: :[6][7][8][15]

Le facteur de perte est un des principaux indices de performance dans la stratégie de commande PWM dont la solution optimale est obtenue avec la minimisation de cette quantité. La formule de cette dernière est la suivante :

$$\sigma = \sqrt{\sum_{n=2}^{Q} \left(\frac{V_n}{n}\right)}$$

Q=10N; N: nombre de commutation par 1/4 de période.

Pour notre cas Q=5xSxR.

On remarque bien, que le facteur de perte diminue avec l'augmentation du nombre de répétition et aussi ce facteur s'améliore avec l'augmentation de l'équerre entre les valeurs des deux répétitions, des segments paire et impaire. Donc pour optimiser la commande PWM avec répétition alternée on procède à un choix entre le facteur de perte et la valeur des harmoniques parus.

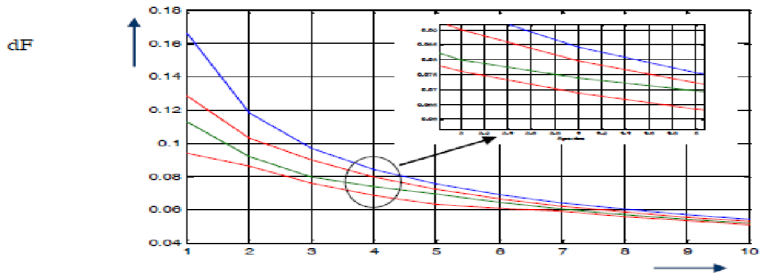


Figure :2.17: Facteur de perte en fonction du nombre de répétition.

II.9.5.Répétition variable: :[6][7][8][15]

On présente dans cette partie un développement en sérié de Fourier pour un cas général ; chaque segment i est répété par une valeur Ri différente des autres segments dans le but d'améliorer le pas de variation de fréquence et plus de précision dans la génération des signaux:

$$\begin{cases} a_n = \dfrac{E}{n\pi}\left(1+\sum_{i=1}^{m-1}\sum_{k=1}^{R_i-1}(-1)^i.\cos(n.\theta_i)\right) \\ b_n = \dfrac{E}{n\pi}\left(\sum_{i=1}^{m-1}\sum_{k=0}^{R_i-1}(-1)^{i+1}.\sin(n.\theta_i)\right) \end{cases}$$

avec:

$$\begin{cases} \theta_{2p} = \dfrac{2.\pi}{S.R}\left(\sum_{j=0}^{m-1} R_j + R_i.i + k\right) \\ \theta_{2p+1} = \theta_{2p} + \dfrac{\pi}{S.R}\left(1 + r.\sin\left(\dfrac{2\pi}{S}i\right)\right) \end{cases}$$

Donc la fréquence de signal généré ce calcule de la manière suivante:

$$F = \frac{f}{b \times \sum_{i=1}^{m} R_i}$$

Pour montrer l'utilité de cette procédure on effectue sur le signal de référence une répétition de quatre segments, présentent une double symétrie afin d'obtenir un signal de moindre fréquence et de maintenir le rapport cyclique du signal sinusoïdal constant comme le montre la figure 18.

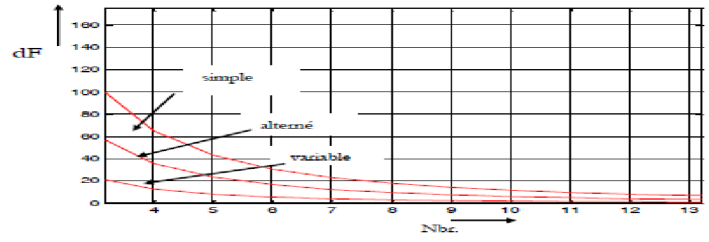


Figure :2.18: Variation du pas d'incrémentation en fonction du nombre de répétition.

Sur le tableau n° 1 on présente les formules calculent les variations minimales de la fréquence qui peut être obtenue entre deux signaux générés pour les 3 cas étudiés.

	Répétition Simple	Répétition Alternée	Répétition variable
F_1	$\frac{f}{n.m.r}$	$\frac{f}{n.m.r}$	$\frac{f}{n.m.r}$
F_2	$\frac{f}{n.m.(r+1)}$	$\frac{2.f}{n.m.(2.r+1)}$	$\frac{f}{n.(m.r+4)}$
ΔF	$\frac{f}{n.m.r.(r+1)}$	$\frac{f}{n.m.r.(2.r+1)}$	$\frac{4.f}{n.m.r.(m.r+4)}$

Tableau 2.1 : Formule de variation minimale de fréquence pour chaque technique.

II.9.6.Répétition alternée d'amplitude variable: :[6][7][8][15]

Pour générer des signaux de commande à amplitude intermédiaire sans recoure à desdonnées supplémentaires on procède à une génération alternée des données des signaux de référence les plus proches avec une profondeur variable rm.

II.9.6.1.Principe de la technique d'amplitude variable: :[6][7][8][15]

Le principe de génération des signaux de commande d'amplitude variable basé sur lagénération alternée des segments pair et impair des deux signaux de commande de référence obtenus par l'échantillonnage des deux signaux d'amplitudes différentes comme le montre la figure

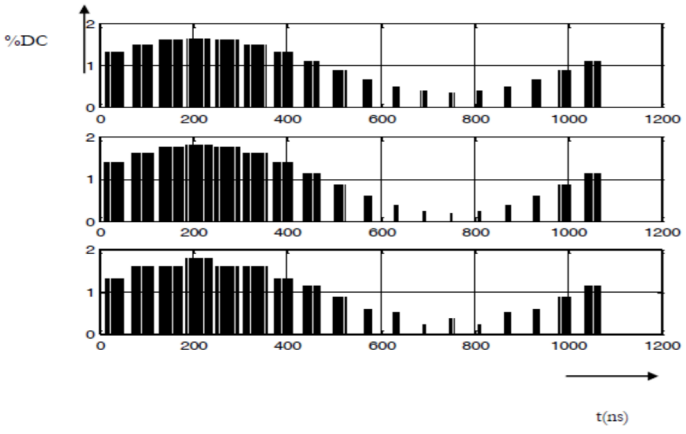


Figure:2.19: Exemple d'un signal de commande.

obtenu à partir de deux signaux de référence a amplitude différente (a) et (b)

Etude mathématique .

Le développement en série de Fourier du signal composé peut être obtenu par la formule suivante; développée à partir de l'équation de la technique R.P.W.M. (chapitre2)

$$a_n = \frac{E}{n\pi}\left\{\sum_{i=0}^{m/2-1}\left(\sum_{k=0}^{R1-1}\cos\left(n.\frac{2.\pi}{S.R}(R.(2.i)+k)\right)\right.\right.$$
$$-\cos\left(n.\left(\frac{2.\pi}{S.R}(R.(2.i)+k)+\frac{\pi}{S.R}(1+r_1.\sin(\frac{2\pi}{S}(2.i)))\right)\right)$$
$$+\sum_{k=0}^{R2-1}\cos\left(n.\frac{2.\pi}{S.R}(R_1+R.(2.i)+k)\right)$$
$$\left.\left.-\cos\left(n.\left(\frac{2.\pi}{S.R}(R_1+R.(2.i)+k)+\frac{\pi}{S.R}(1+r_2.\sin(\frac{2\pi}{S}(2.i+1)))\right)\right)\right\}$$

$$b_n = \frac{E}{n\pi}\left\{\sum_{i=0}^{m/2-1}\left(\sum_{k=0}^{R1-1}\sin\left(n.\frac{2.\pi}{S.R}(R.(2.i)+k)\right)\right.\right.$$
$$-\sin\left(n.\left(\frac{2.\pi}{S.R}(R.(2.i)+k)+\frac{\pi}{S.R}(1+r_1.\sin(\frac{2\pi}{S}(2.i)))\right)\right)$$
$$+\sum_{k=0}^{R2-1}\sin\left(n.\frac{2.\pi}{S.R}(R_1+R.(2.i+1)+k)\right)$$
$$\left.\left.-\sin\left(n.\left(\frac{2.\pi}{S.R}(R_1+R.(2.i+1)+k)+\frac{\pi}{S.R}(1+r_2.\sin(\frac{2\pi}{S}(2.i+1)))\right)\right)\right\}$$

II.9.6.2.Résultats par simulation: :[6][7][8][15]

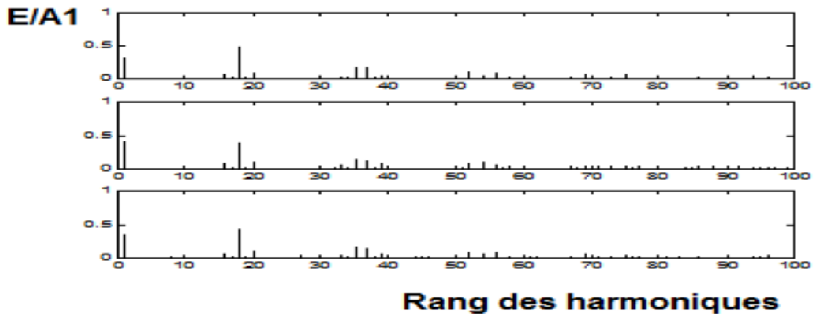


Figure:2.20:Spectre des signaux de commande.

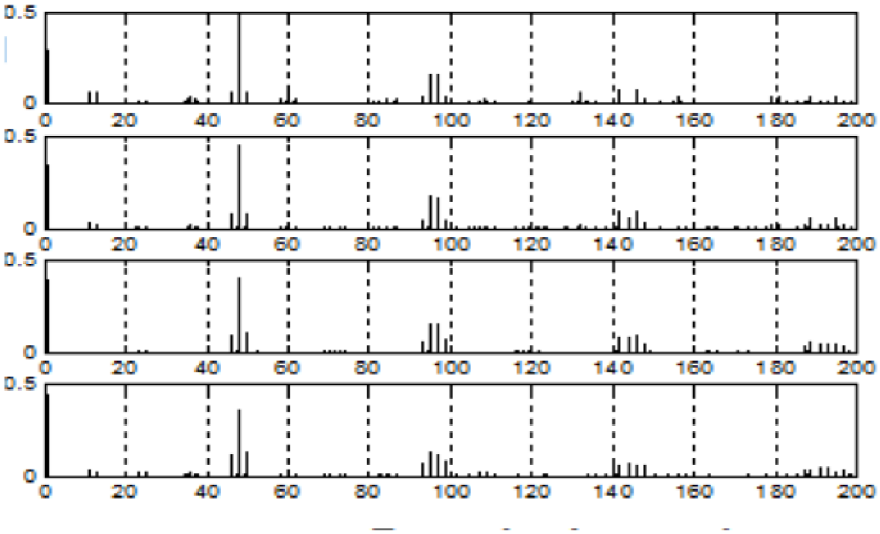


Figure :2.21:. Spectre des signaux composés pour différent Δr.

(a) =0.2, (b)Δr=0.1, (c)Δr=0, (d) Δr=-0.1

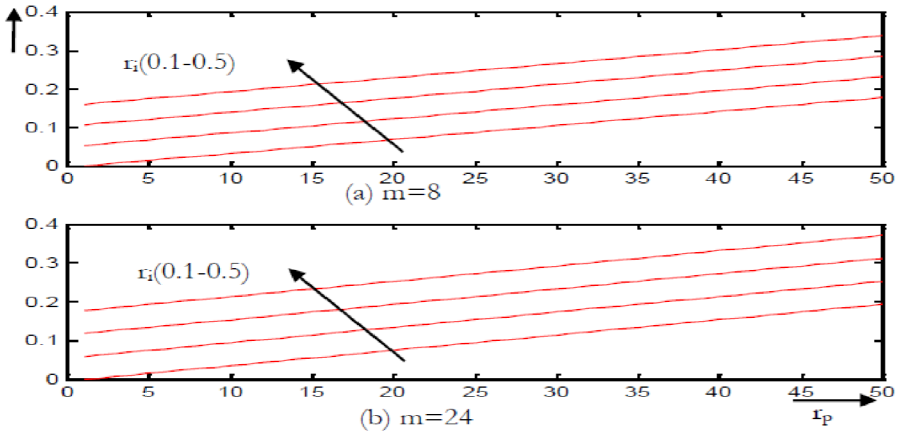


Figure :2.22: Valeur de l'amplitude du fondamental pour quatre valeurs différentes de ri en fonction de rp pour m=8 et 24.

On présente sur la figure 3 le spectre des signaux de commande composés à partir de deuxsignaux d'amplitude différente r1 et r2 dont Δr et la différence entre les .deux dernières amplitudes

La valeur du fondamental du signal composé pour un ri donnée en fonction de rp peut êtrecalculée de la fonction de la droite

$$A_1 = a.R_p + b.R_i$$

$$a = 0.592, \quad b = 0.3959$$

r_1:Amplitude de signal qui forme les segments pairs.

r_p:Amplitude de signal qui forme les segments impairs.

Dont la constante 'a' dépend du nombre de segments ou échantillon comme la montre sur lafigure 5.

La valeur du fondamental dépend de la différence du nombre des répétitions des segmentspairs et impairs.

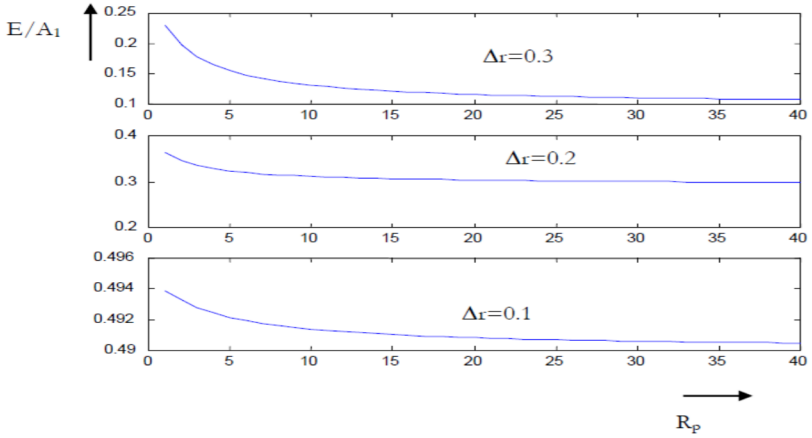


Figure :2.23: Amplitude du fondamental en fonction du nombre de répétition des segments pairs (Rp) pourtrois différentes valeurs de Δr.

(a) r_1=0.5 r_2=0.1 (b) r_1=0.5 r_2=0.3 (c) r_1=0.5 r_2=0.5

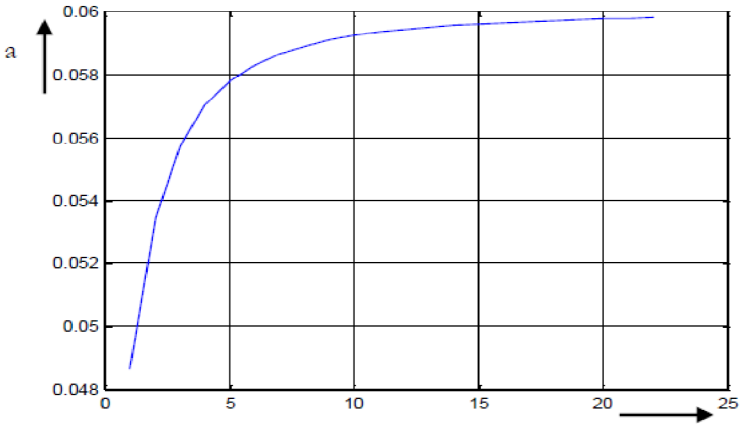


Figure:2.24: Valeur de 'a' en fonction du nombre des segments S=2-24.

$$a = \frac{a_1 + bx}{1 + cx + dx^2}$$

$$a_1 = 0.05, \qquad b = 0.05, \qquad c = 0.08, \qquad d = 0.0004.$$

Pour un cas d'un signal de référence alterné de déférentes amplitudes des segments paire etimpaire, la valeur du fondamental peut être calculée de l'équation 3 dont le coefficient 'a' dépend dedu rapport r_p et r_i comme le montre la figure 5 dont, en présente 'a' en fonction du rp-ri avec la valeur de r_p est constante

Facteur de perte

Pour valoriser cette technique de génération on présente sur la figure 8 le facteur de perte enfonction de la différence d'amplitude entre les segments pairs et les segments impairs

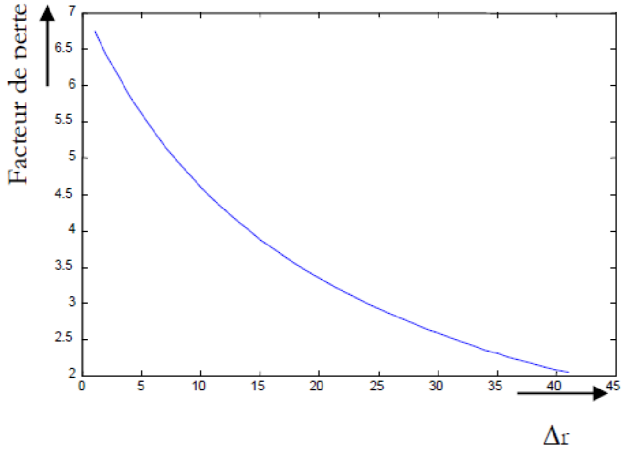


Figure:2.25:Facteur de perte en fonction .

On remarque bien que facteur de perte diminue quand l'équerre d'amplitude entre les deux signaux associés augmente. Donc pour optimiser la commande R.P.W.M. on procède à un choix entre le facteur de perte et la valeur des harmoniques parus après l'application de la répétitionalternée d'amplitude variabl

Application des résultats de l'étude

Dans le tableau 2.1., nous donnons quelques valeurs de fréquence, que nous avons calculéeset qui sont utilisé dans les tests de convertisseur, ainsi que le nombre de répétition pour les deuxsegments, le gain en mémoire obtenu par notre structure et les pas de variation de fréquence pourles deux cas (répétition simple et répétition alternée). On remarque bien que la nouvelle techniqueproposée nous permet

d'accéder aux fréquences intermédiaires sans le changement de la fréquencede modulation qui reste 31.26 KHz avec un pas de variation amélioré de 200%

Fréquence (HZ)	Nb. de répétitions segments pairs	Nb. de répétitions segments impairs	Espace mémoire généré (Ko)	Espace mémoire réel (octets)	Gain en mémoire	Pas de variation précédent	Pas de variation amélioré
44,90	29	29	22,272	768	96,55	-	-
45,69	29	28	21,888	768	96,49	-	0,79
46,50	28	28	21,504	768	96,43	1,60	0,82
47,35	28	27	21,120	768	96,36	-	0,85
48,23	27	27	20,736	768	96,30	1,72	0,88
49,14	27	26	20,352	768	96,23	-	0,91
50,08	26	26	19,968	768	96,15	1,85	0,94
51,06	26	25	19,584	768	96,08	-	0,98
52,08	25	25	19,200	768	96,00	2,00	1,02
53,15	25	24	18,816	768	95,92	-	1,06
54,25	24	24	18,432	768	95,83	2,17	1,11
55,41	24	23	18,048	768	95,74	-	1,15
56,61	23	23	17,664	768	95,65	2,36	1,20
57,87	23	22	17,280	768	95,56	-	1,26
59,19	22	22	16,896	768	95,45	2,57	1,32

Tableau2.2. Récapitulation des résultats de l'étude .

II.9.7.Méthode d'utilisation de MLI: :[6][7][8][15]

Les données sont une séquence de bits stockées en mémoire. Chaque bit (1,0) présente l'état de l'interrupteur (ON, OFF). Chaque segment présente les données de base de chaque cycle[

La séquence de données de 32 bits définie par un cycle de base nous donne une sensibilité de 3.125%

II.9.8.Circuit de commande électronique: :[6][7][8][15]

II.9.8.1.article de bou: :[6][7][8][15]

Le circuit électronique de commande est conçu pour valider l'étude théorique et générer des signaux de commande PWM répétés avec la méthode de combinaisons CRPWM. Cette technique nécessite une génération de signaux de commande autonomes à très haute fréquence de l'ordre de 1MHz, pour balayer une fréquence de champ acceptable. Dans ce cas, la génération du signal du microcontrôleur devient incontournable. Nous avons donc opté pour un adressage séquentiel système indépendant du microcontrôleur pouvant assurer le balayage de données prédéfinies, stockées dans l'EPROM sous forme numérique, avec répétition autonome sans l'intervention du microcontrôleur. Qui quand a mesure, teste, numérote et assure les signaux de commande pour aller d'un segment à l'autre .avec les incréments et décréments de profondeurs

Ce concept a l'avantage d'assurer la génération de signaux de courte longueur, la réduction des lignes d'adresse entre le microcontrôleur et la mémoire de données et la réduction du temps alloué par le microcontrôleur. Par conséquent, un système plus efficace (DSP, FPGA, ...) pour la génération de ce type de commande, est Inutile

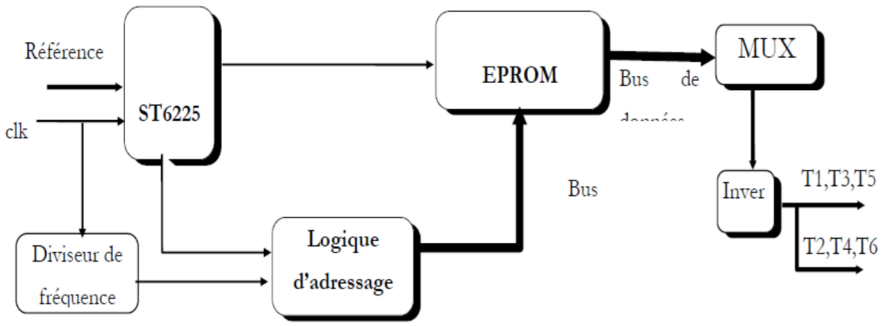


Figure:2.26: Leschéma du circuit de commande

II.9.9.Conclusion: :[6][7][8][15]

Dans ce travail, nous avons présenté un état des lieux de la commande PWM avec une combinaison signaux concrétisée par un matériel optimisé dans le choix des composants, gain en mémoire pouvant atteindre 100%, gain en temps réservé par le microcontrôleur pouvant atteindre 50 % et la sensibilité de la génération du signal qui est d'une micro-seconde. Aussi, la nouvelle technique proposée CRPWM a amélioré le pas de variation de fréquence notamment dans l'intervalle .de 50Hz qui se réduit à 0.02Hz.

CHAPITRE III:

SIMULATION

Chapitre III: simulationt

III.1.Introduction:

L'objectif est de concevoir un outil de commande adapté de manière à optimiser le rendement d'une machine de puissance alimentée à partir d'une source d'énergie solaire variable. La source d'énergie délivre une valeur de tension qui risque de varier importunément selon les moments de la journée, les saisons de l'année, le temps qu'il fait…

Le principe basé sur la technique R.P.W.M. est de générer des signaux à fréquences relatives aux nombres de répétitions des segments obtenus lors de la modulation. Ces valeurs de fréquences obtenues par répétition des segments influencent directement la fréquence du signal alternatif appliqué au moteur, donc sur la constante V/f et par là sur le couple moteur.

L'outil conçu est composé de différents blocs interconnectés de manière à recevoir une valeur en binaire transmise à partir d'un Convertisseur Analogique Numérique représentant la tension d'alimentation et de générer à la sortie des trains d'impulsions adaptées pour être appliqués sur les gâchettes des thyristors (Onduleur.(

Les différents blocs sont traduits en modules décrits sous le langage V.H.D.L., de manière à ce que chaque module assure une fonction intermédiaire dans le système. **:[6][7][8][15]**

III.2.schéma général:

schéma synoptique de la conception est représenté par la figure suivante:

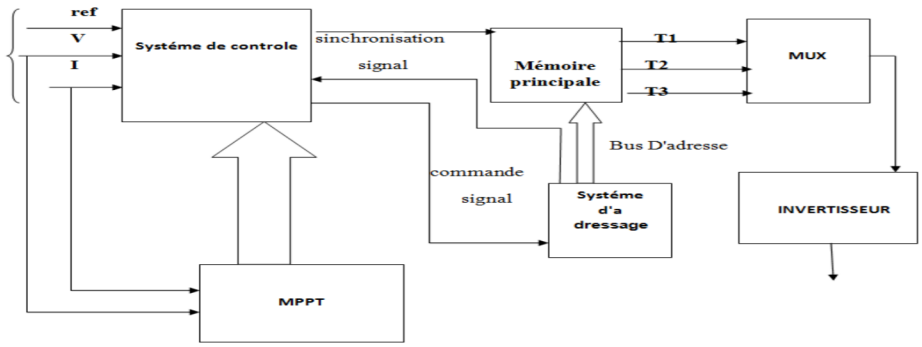


Figure :3.1: schéma général

Il reçoit la valeur en binaire lui parvenant du Convertisseur Analogique / Numérique et la compare à une valeur de référence fixée dans notre cas à 10010110 (300V). Trois cas de figure sont à envisager ;

III.3 .Différents modules de la conception:

La conception est composée des modules suivants:

III.3.1.Comparateur Consigne / Référence:

Il reçoit la valeur en binaire lui parvenant du Convertisseur Analogique / Numérique et la compare à une valeur de référence fixée dans notre cas à 10010110 (300V). Trois cas de figure sont à envisager

- Cosigne > Référence : - Envoyer un signal mono-bit sur le Compteur / Décompteur pour le lancer en sens comptage
- Consigne < Référence : - Envoyer un signal mono-bit sur le Compteur / Décompteur pour le lancer en sens décomptage.
- Consigne = Référence : - Envoyer un signal mono-bit sur le Compteur / Décompteur pour qu'il garde son état. **:[6][7][8][15]**

III.3.2.Cmpteur d'adressage:

Il permet le balayage de l'espace Mémoire Principale pour en extraire les valeurs des segments selon la fréquence préférée (de f0 à f39). C'est un compteur à 10 bits .compartimenté en deux compteurs en cascade de 5 bits chacun**:[6][7][8][15]**

III.3.3.Compteu r / Décompteur:

Il permet de pointer sur l'espace de la Mémoire Principale relatif à la fréquence voulu. Il est actionné d'une part à partir du Comparateur Consigne / Référence pour le sens .de comptage et d'autre part par les dix bits du Compteur d'adressage à '1' à la fois Il génère aussi l'adresse de la case mémoire de répétition contenant le nombre de .répétitions à transmettre au décompteur de nombre de répétitions**:[6][7][8][15]**

III.3.4.Mémoire principale:

C'est une mémoire à écriture / lecture (RAM) de mots de 6 bits, qui contient les valeurs de tous les segments. Le nombre total de segments est de 32 (de 32 bits chacun). Donc, pour une valeur de fréquence on a un nombre de cases mémoires égal à (32x32). Si on considère 40 valeurs de fréquences différentes on aura une Mémoire .Principale de taille (32x32x40) = 40960**:[6][7][8][15]**

III.3.5.Mémoire de répétition :

C'est aussi une mémoire à écriture / lecture (RAM) mais de mots de 8 bits, qui contient les valeurs représentant le nombre de répétitions relativement aux différentes .fréquences

III.3.6.Décompteur de nombre répétitions :

C'est un compteur à 8 bits décroissant qui sert à décrémenter la valeur lui parvenant de la mémoire de répétition (représentant le nombre de répétitions) jusqu'à atteindre la valeur 00000000, pour ensuite actionner le premier compartiment du Compteur d'adressage en se combinant avec les 5 bits de son deuxième compartiment pour le .lancer en comptage**:[6][7][8][15]**

III.3.7.Multiplexeur de commande :

C'est un Multiplexeur 2 Entrées / 1 Sortie à 3 bits. Il permet d'envoyer en ordre 3 par 3 les 6 bits prélevés de la Mémoire Principale vers les gâchettes des thyristors de .l'onduleur**:[6][7][8][15]**

III.4.Signaux d'Entrée / Sortie des différents modules:

Les interactions entre les différents modules de la conception sont résumées dans le tableau swivant:

Modules	Signaux reçus	Sources	Signaux générés	Destinations
Comparateur Consigne / Référence	-Entrée 8 bits.	Convertisseur Analogique Numérique	Signal mono-bit pour le sens de comptage	Compteur / Décompteur
Compteur d'adressage	-Signal mono-bit sur le 1er compartiment. -Horloge sur le 2ème compartiment.	- AND logique entre 5 bits du 2ème compartiment et le résultat du NOR logique à la sortie du Décompteur.	- 10 bits d'adressage vers la mémoire principale.	Mémoire principale.
Compteur / Décompteur	-signal mono-bit pour le sens de comptage. -Signal mono-bit qui sert d'horloge.	Comparateur Consigne / Référence. -AND logique des 10 bits du	-Bus de 6 bits pour le passage d'une fréquence à une autre.	-Mémoire principale.

		compteur d'adressage.		
Mémoire principale	-Bus de données de 6 bits. -Bus d'adresse de 10 bits. -Bus d'adresse de 6 bits.	-Ecriture mémoire. -Compteur d'adressage. Compteur / Décompteur.	-Bus de données de 6 bits.	Multiplexeur de commande.
Mémoire de répétition	-Bus de données de 8 bits. -Bus d'adresse de 6 bits.	Ecriture mémoire. Compteur / Décompteur.	Bus de données de 8 bits.	-Décompteur.
Décompteur Nombre de répétitions	-Bus de données de 8 bits. -Horloge.	-Mémoire de répétition.	-Bus de données de 8 bits.	-NOR logique des 8 bit ensuite AND logique avec 5 bits du 2ème compartiment pour aller vers 1er compartiment Compteur d'adressage.
Multiplexeur de commande	-Bus de données de 6 bits. -Un signal mono-bit de sélection.	-Mémoire principale. -Signal mono-bit (MSB) du Compteur / Décompteur.	-Un bus de 3 bits. -Les trois bits inversés.	-Les gâchettes des thyristors T_1-T_2-T_3. Les gâchettes des thyristors T_1'-T_2'-T_3'.

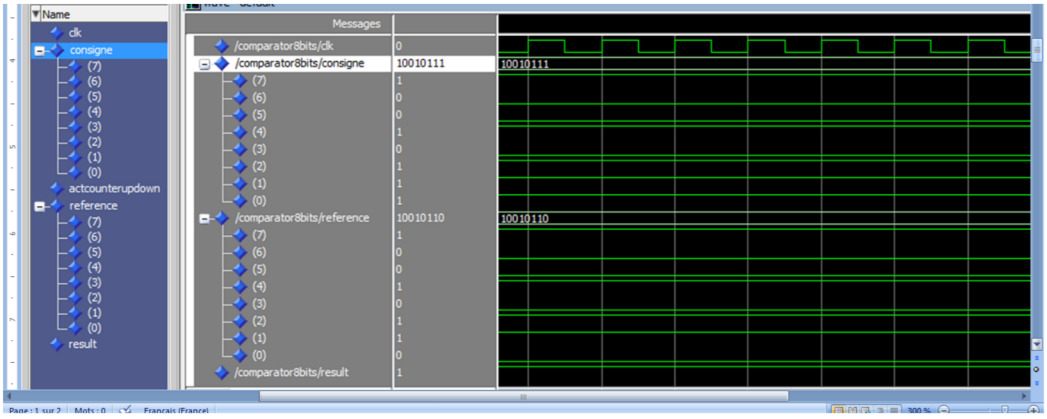
Tableau:3.1: Signaux d'entrée / sortie des différents modules.

III.5.Résultats de la simulation :

Les résultats de la simulation sont donnés par module sous forme d'image écran afin .de les illustrer au mieux

III.5.1.Comparateur Consigne / Référence:

Le module reçoit un bus de données lui parvenant du convertisseur analogique / numérique. La valeur consigne reçue sera comparées à la valeur numérique de référence (300Volts). Selon le résultat de la comparaison, un signal mono-bit sera envoyé et servira de commande au sens de comptage du Compteur / Décompteur.



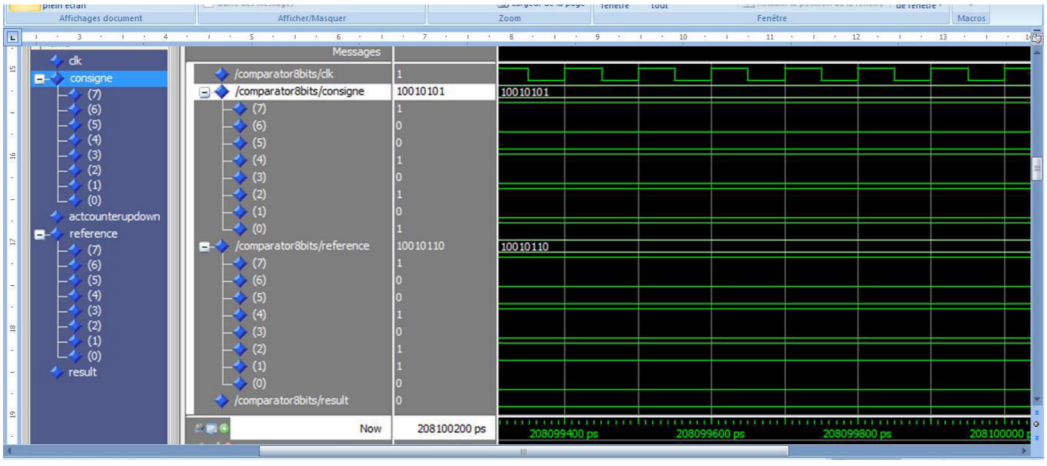


Figure:3.2:resultats de simulation Comparateur Consigne / Référence.

III.5.2.Compteur d'adressage ompteur:C'est un compteur à deux compartiments qui servira au pointage de l'adresse principale contenant les segments.

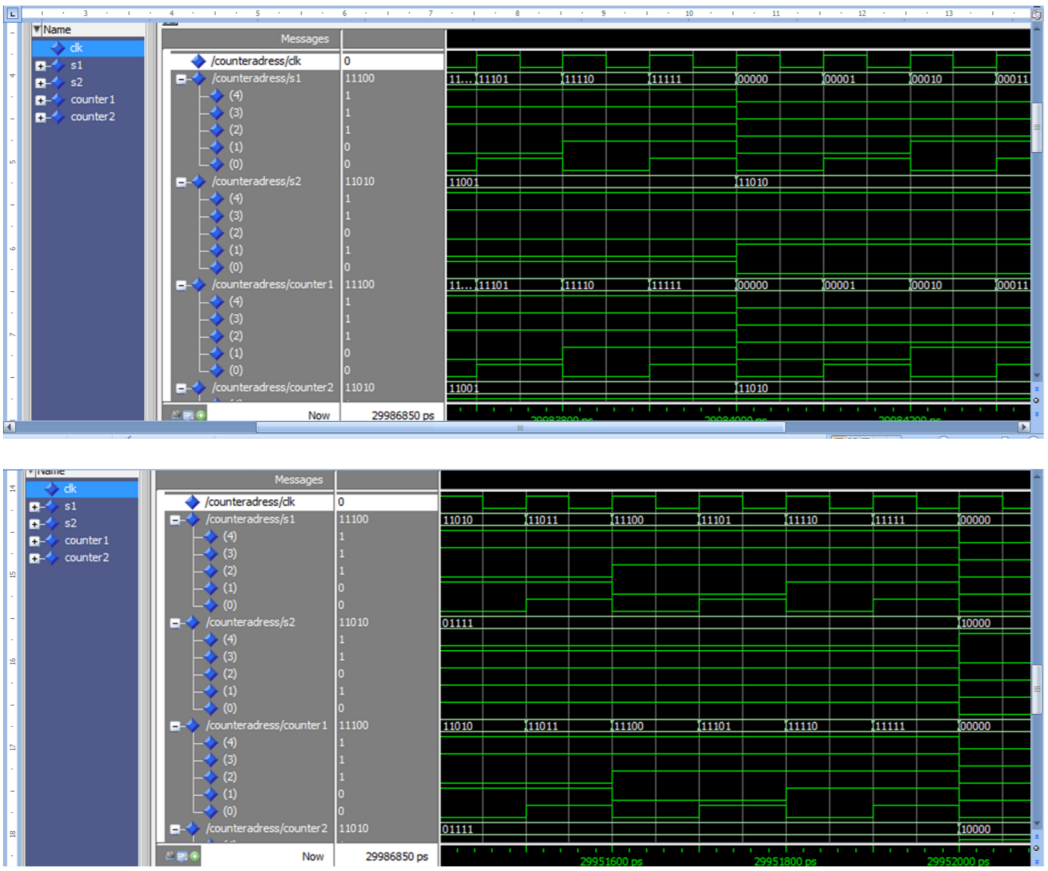
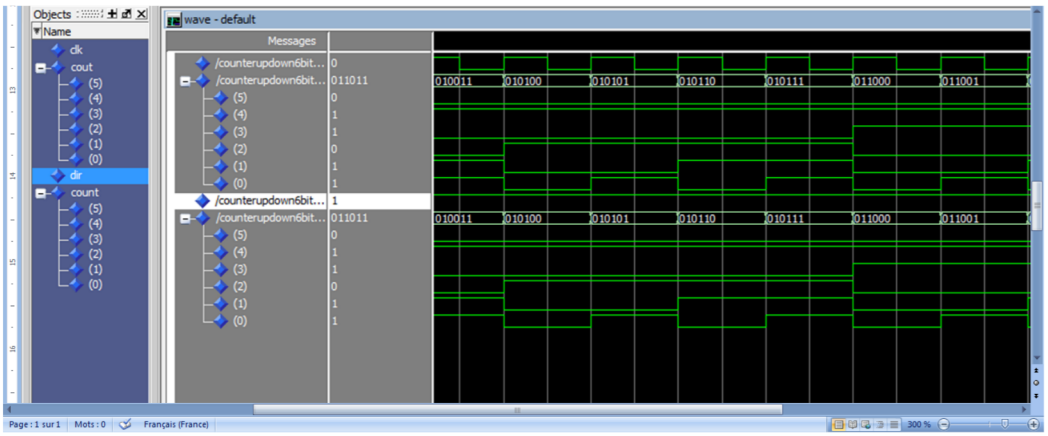


Figure:3.3: resultats de simulation Compteur d'adressage .

III.5.3.Compteur /Décompteur:

Ce module est un compteur en deux sens (Up / Down) qui sert de complément au compteur d'adressage pour pointer les blocs d'adresse dans la mémoire principale pour passer d'un espace mémoire à un autre selon la valeur des fréquences de répétition.



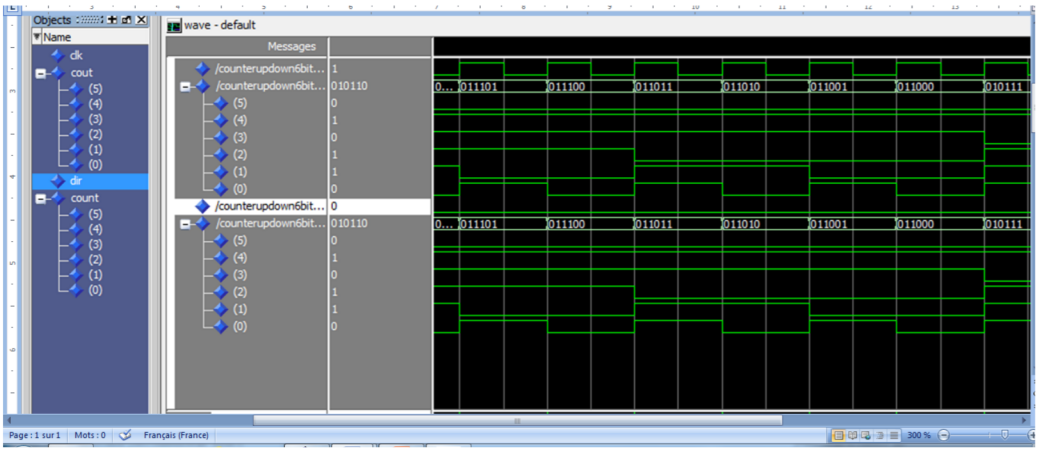
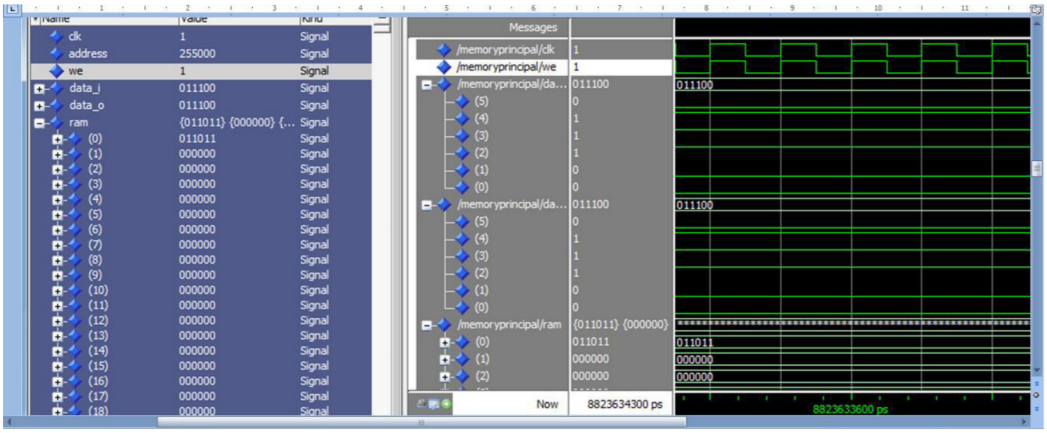
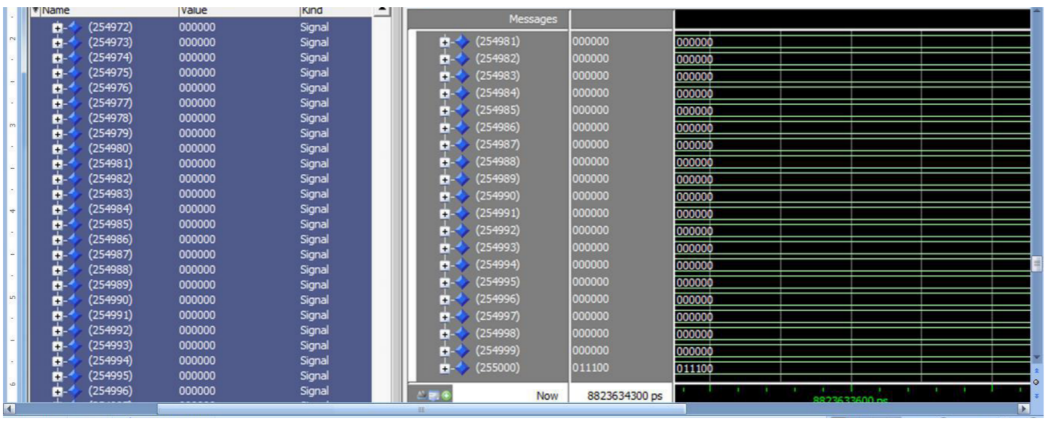


Figure: 3.4:resultats de simulation Compteur /Décompteur:

III.5.4.Mémoire principal:

C'est un espace mémoire contenant les valeurs des segments représentant les échantillons du signal. C'est une mémoire à écriture / lecture pour permettre d'y stocker les valeurs numériques des segments et de l'en extraire au moment voulu.





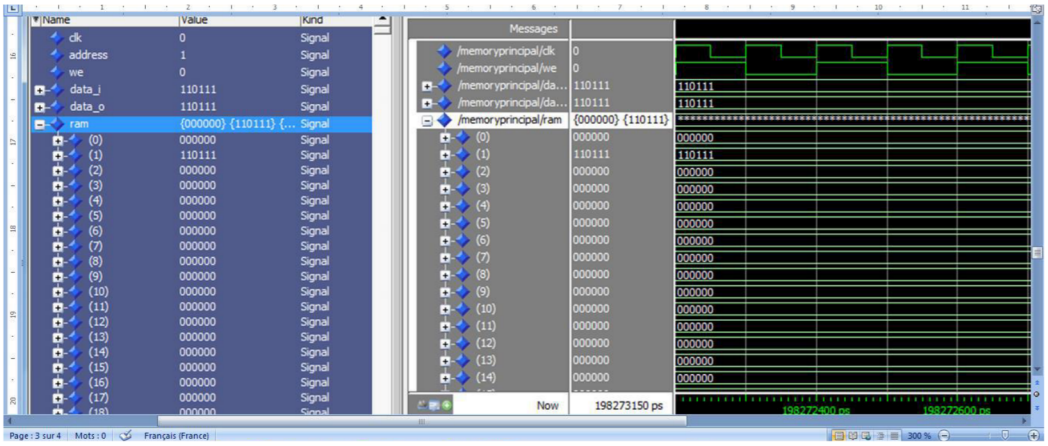


Figure:3.5:resultats de simulationMémoire principale.

III.5.5.Mémoire de répétition:

C'est un bloc mémoire ordinaire qui contient des valeurs numériques relatives aux nombres de répétitions emmagasinées au préalables et qui seront pointées par le résultat de la comparaison Consigne / Référence à travers le Compteur / Décompteur dont la sortie qui servira d'entrée adresse.

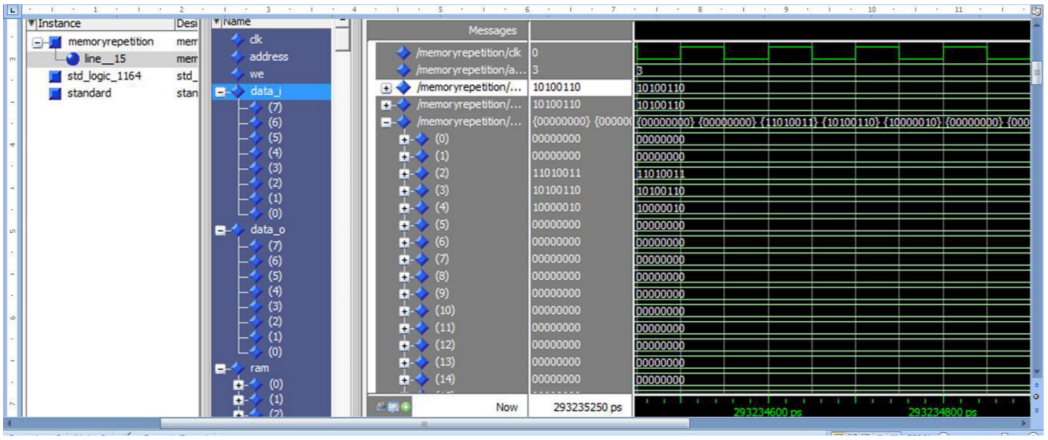


Figure: 3.6: resultats de simulation Mémoire de répétition .

III.5.6.Décompteur de nombre de répétition:

Ce module est un décompteur qui part d'une valeur de répétition issue de la mémoire et la décrémente jusqu'à la valeur zéro pour envoyer un signal de commande au compteur d'adressage mémoire principale à travers une porte logique OU – Inversée.

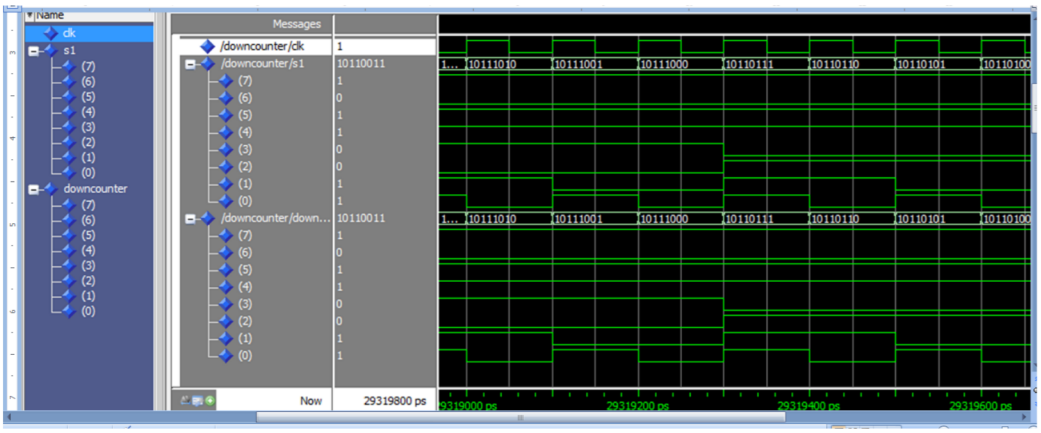
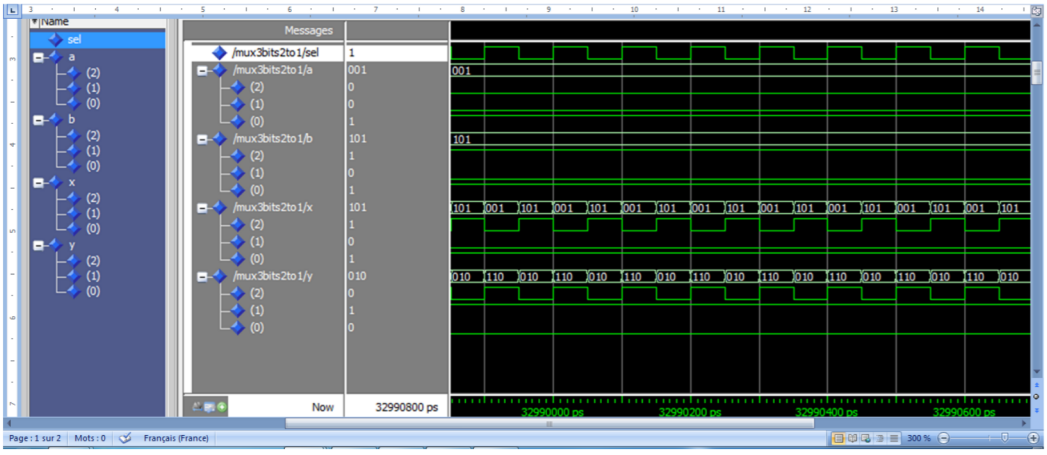
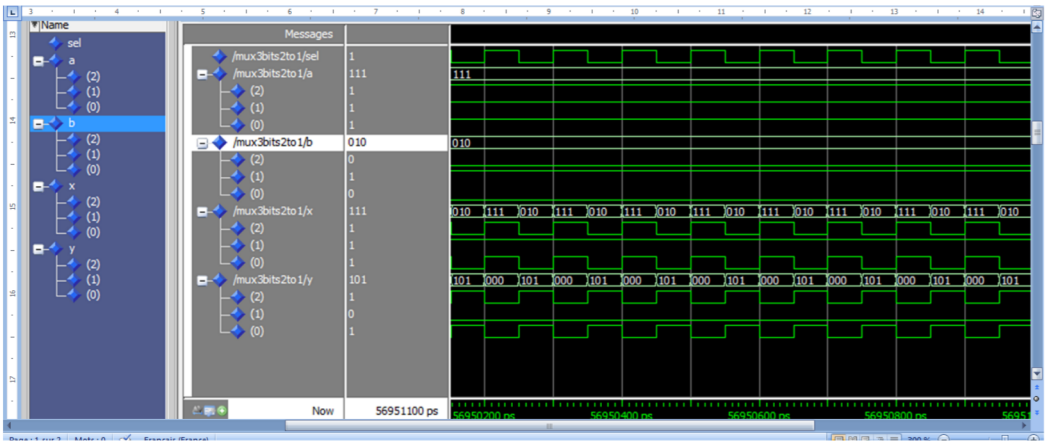


Figure: 3.7: resultats de simulation Décompteur de nombre de répétitions.

III.5.7Multiplexeur de commande .

C'est un boitier multiplexeur à deux entrées de trois bits chacune. Il reçoit un bus de six bits de la mémoire principale et sert à la sélection de trois bits sur les six pour les envoyer alternativement sur trois gâchettes des thyristors de l'onduleur et leurs inverses sur les trois autres gâchettes.





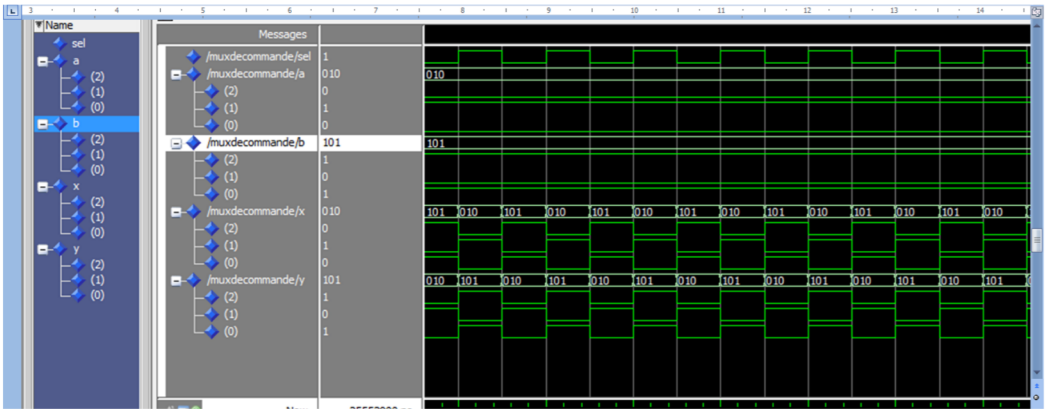


Figure: 3.8: resultats de simulation Multiplexeur de commande .

III.5.8.PROGRAMME PRINCIPAL:

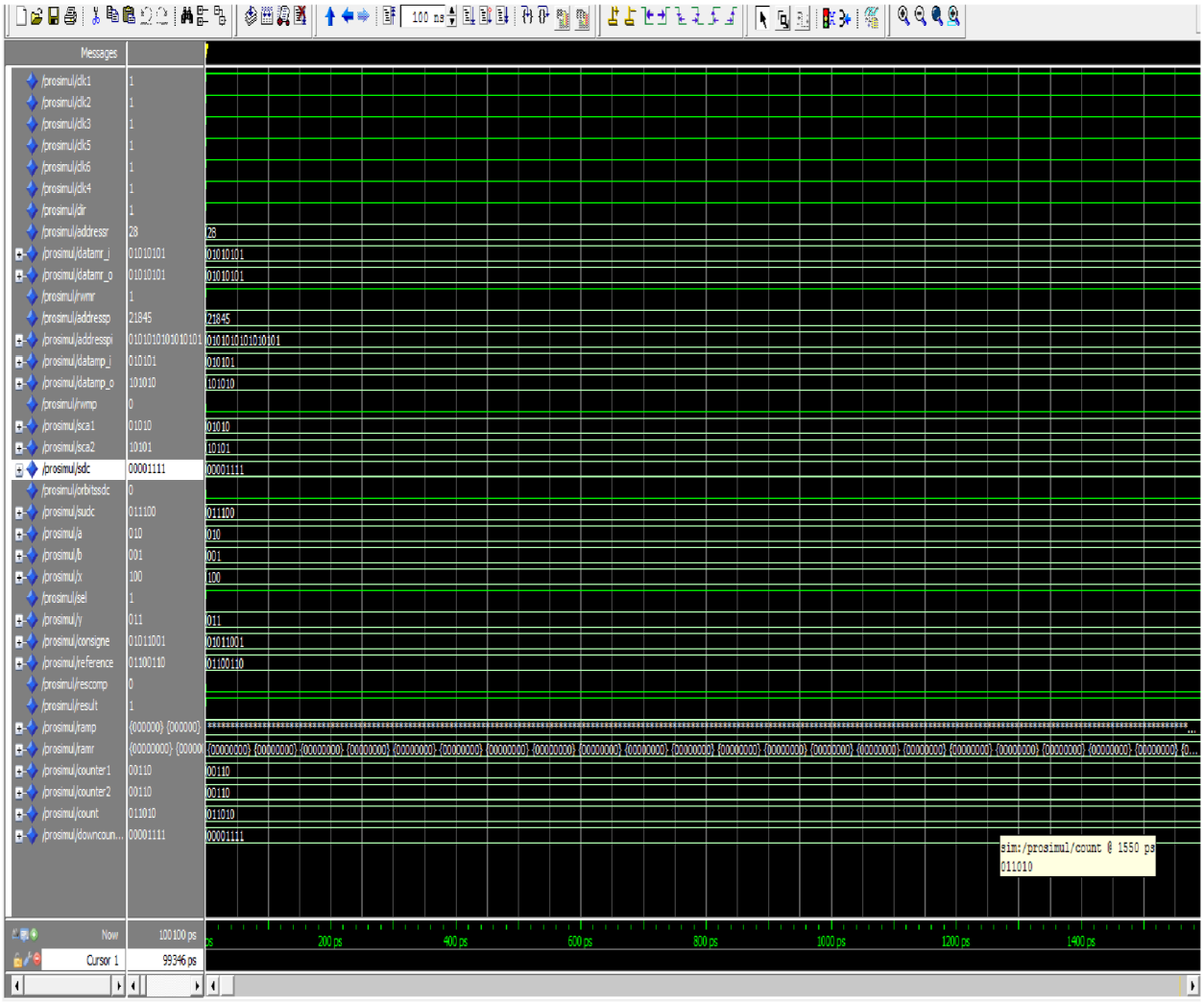


Figure: 3.9: resultats de simulation programme principal .

III.6.Conclusions:

On constate bien que les signaux obtenus répondent au fonctionnement de leurs modules respectifs. Ce qui rendrait la simulation de la conception faisable

Il est toutefois intéressant de dire que pour aboutir à la réalisation de ce travail nous avons été contraints de passer par certaines étapes et de réflexions, à citer par exemple Avoir essayé d'utiliser les machines d'états (MOORE et/ou MEALY), avec toutes les études nécessaires Avoir envisagé de passer à l'implémentation (sur FPGA), avec toutes les recherches pour le choix du FPGA qui conviendrait .Espérons que ça sera aussi utile qu'intéressant d'exploiter ces résultats.

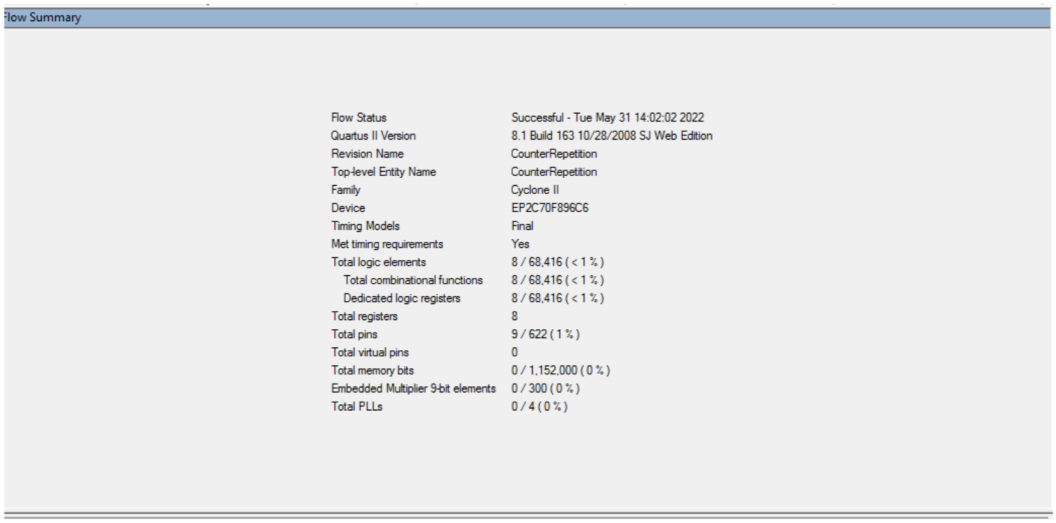
CHAPITRE: IV IMPLÉMENTATION DE SYSTÈME DE COMMANDE SUR QUARTUS

Chapitre IV: Implémentation de système de commande sur QUARTUS:

IV.1.INTRODUCTION:

La dernière étape consiste à programmer le composant, c'est à dire implanter la description dans la cible matérielle (FPGA dans notre cas). Il faut pour cela que le projet ait auparavant été compilé.

IV.2.Compteur de Repetition



Compilation rapport Compteur de Repetition

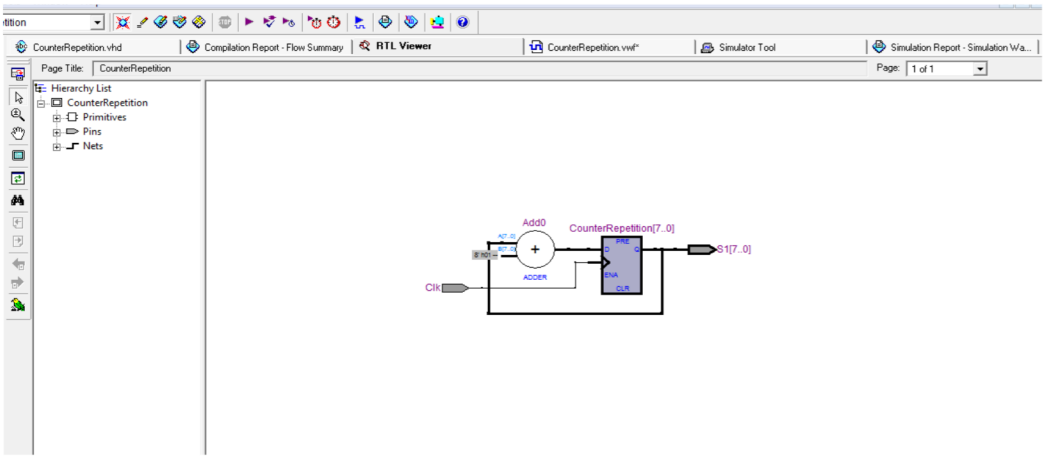


Figure: 4.1:Compteur de Repetition:

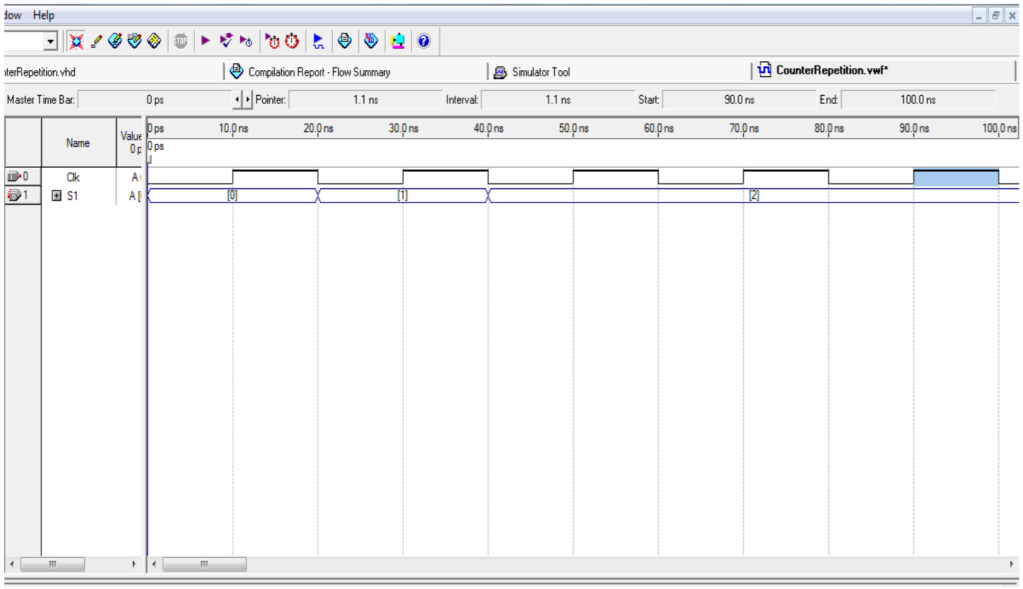


Figure: 4.2:simulationt QUARTUS Compteur de Repetition:

IV .3 . Memoire de repetition

Flow Summary

Flow Status	Successful - Tue May 31 18:11:06 2022
Quartus II Version	8.1 Build 163 10/28/2008 SJ Web Edition
Revision Name	memoryrepitition
Top-level Entity Name	memoryrepitition
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	0 / 68,416 (0 %)
Total combinational functions	0 / 68,416 (0 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	50 / 622 (8 %)
Total virtual pins	0
Total memory bits	4,104 / 1,152,000 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Compilation rapport Mémoire de repetition

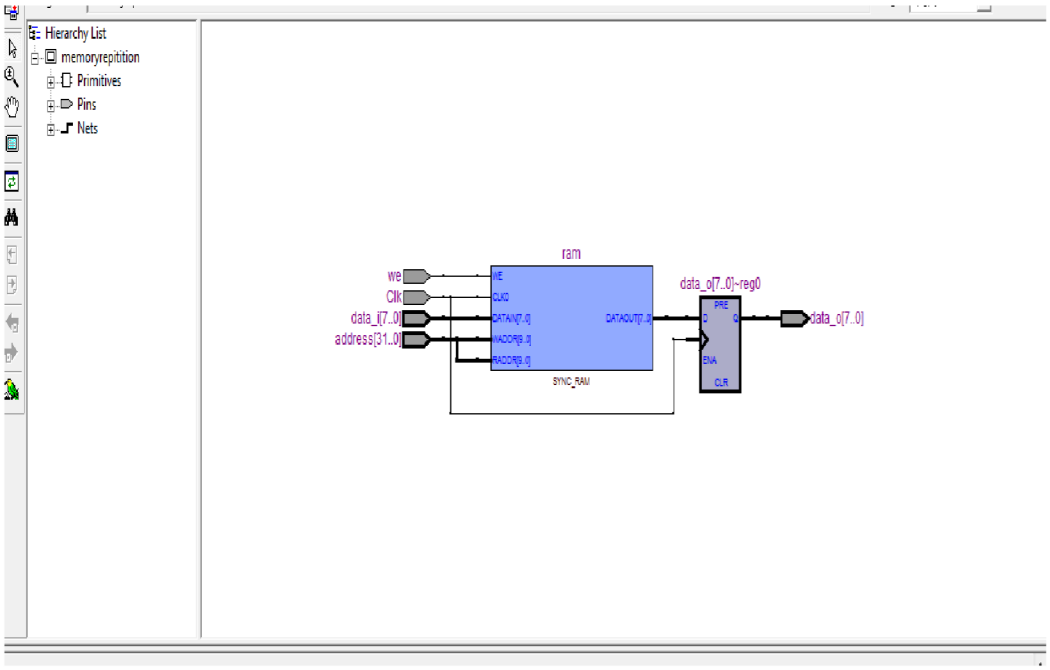


Figure: 4.3. Mémoire de repetition:

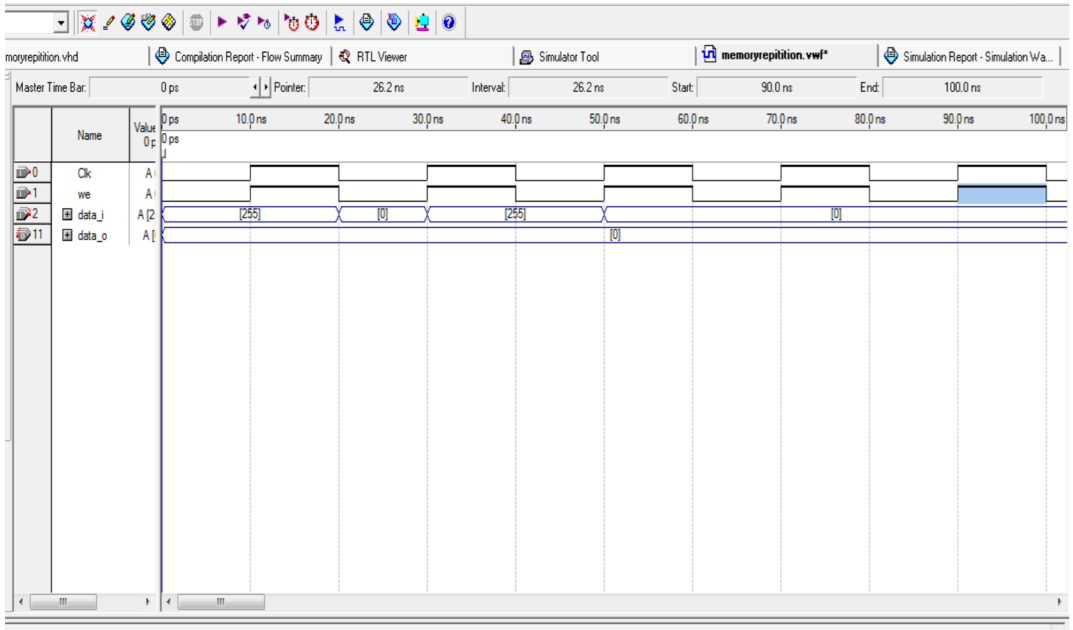
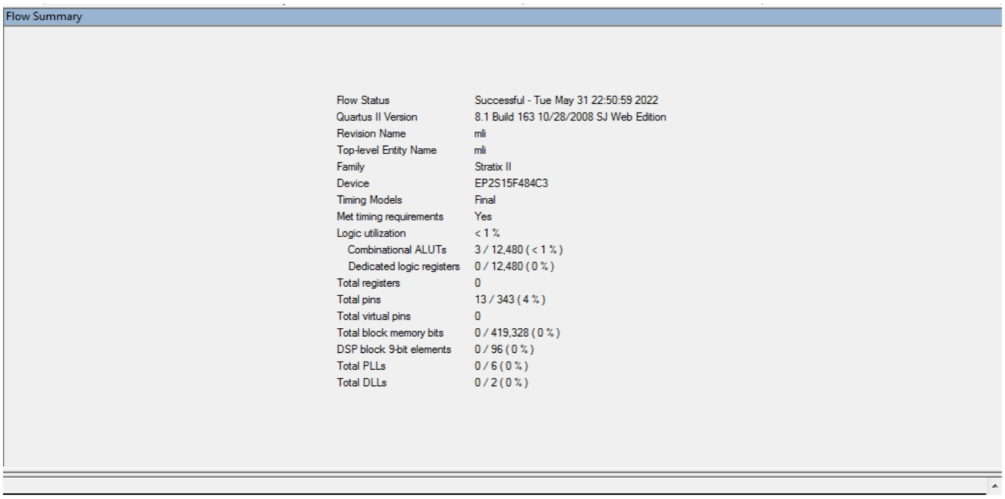


Figure: 4.4:simulation QUARTUS Mémoire de repitition:

IV .4.Multiplexeur de commande:



Compilation rapport Multiplexeur de commande

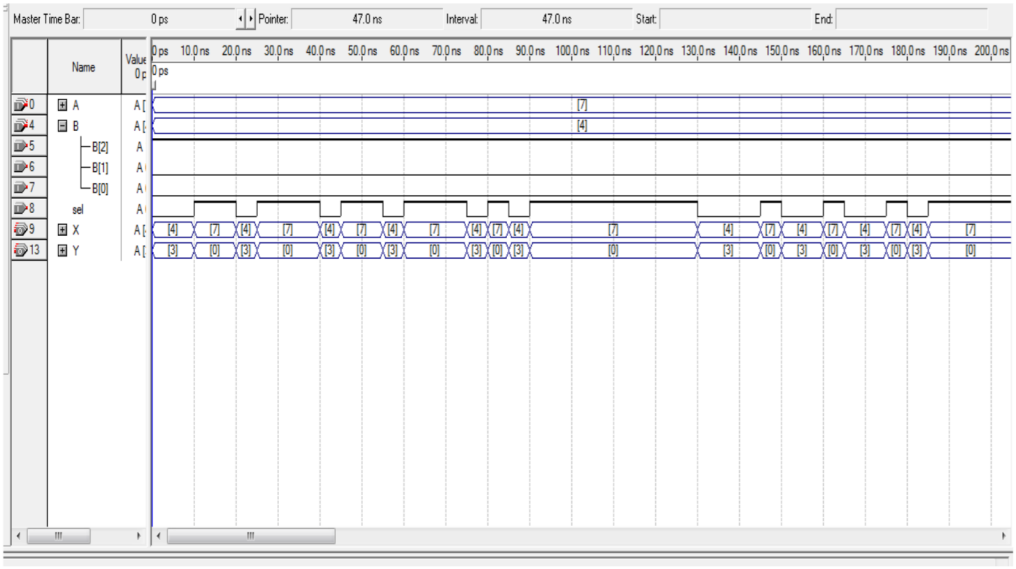


Figure: 4.5:simulation QUARTUS Multiplexeur de commande:

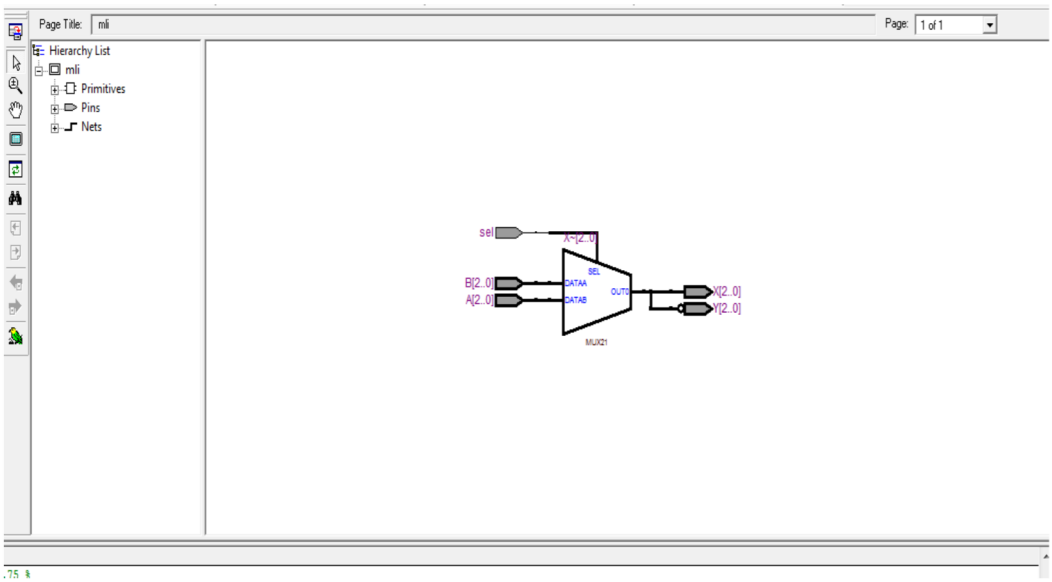
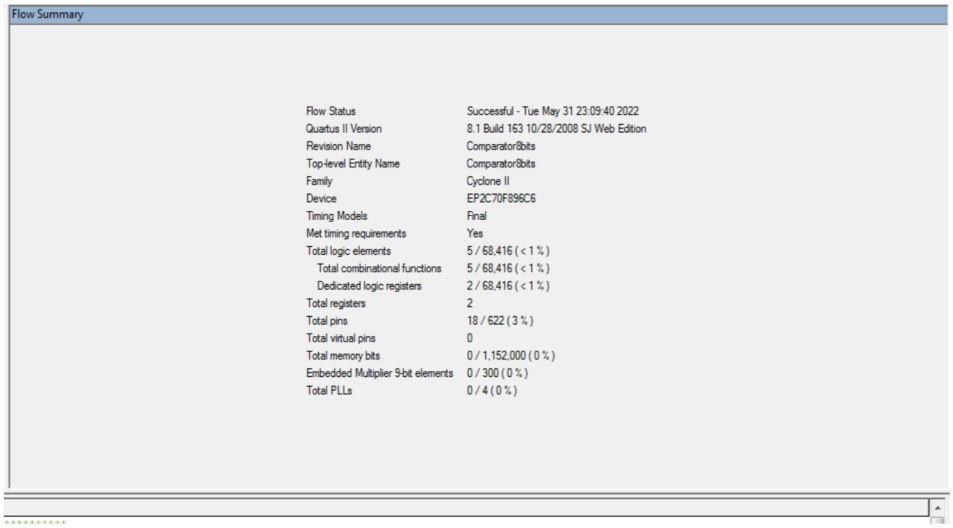


Figure: 4.6:Multiplexeur de commande:

IV .5.Comparateur 8bits :



Compilation rapport Comparateur 8bits

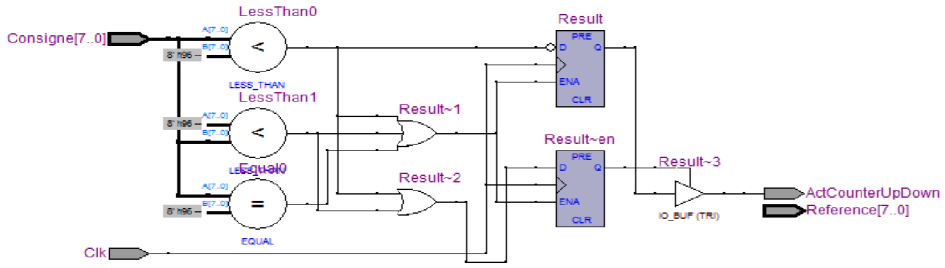


Figure: 4.7.Comparateur 8bits:

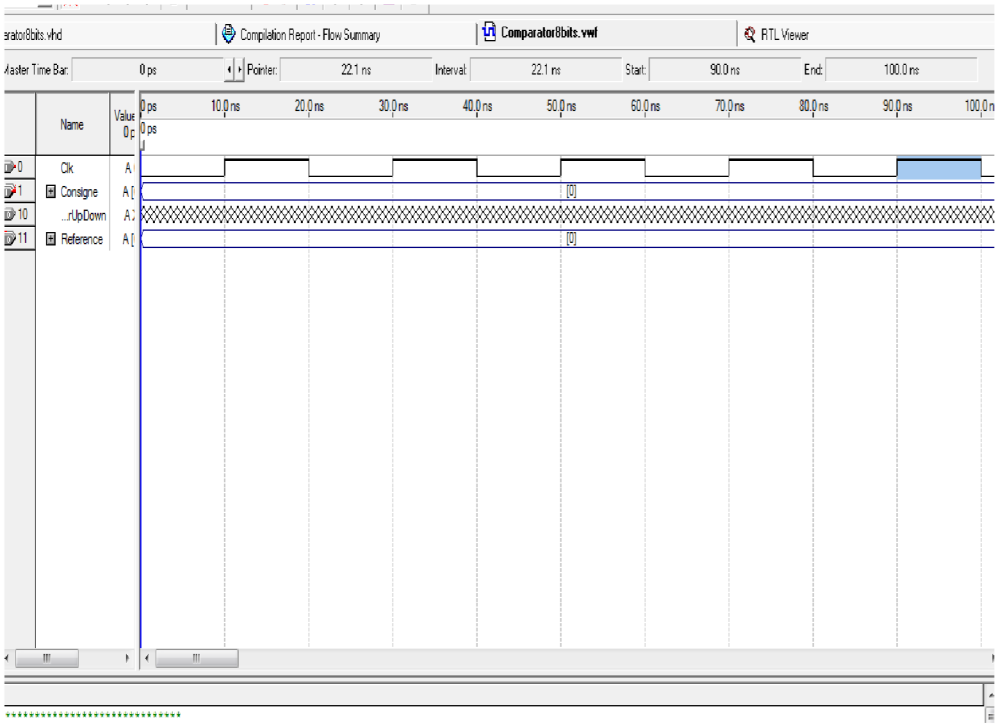
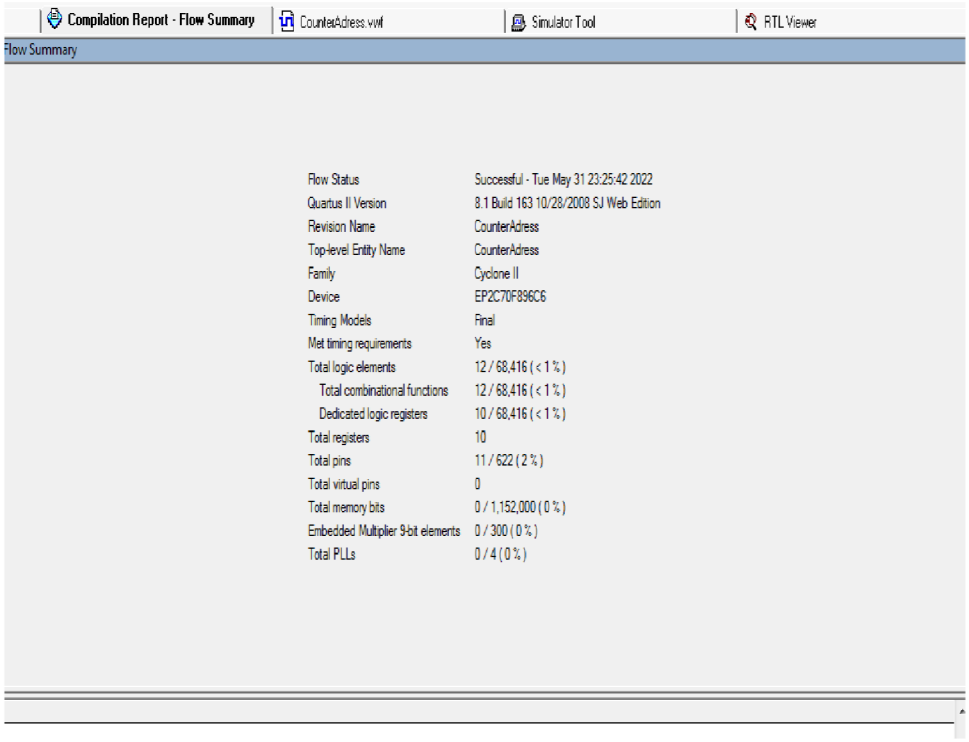


Figure: 4.8:simulation QUARTUSComparateur 8bits:

IV .6.Compteur D'adressage:



Compilation rapport Compteur D'adressage

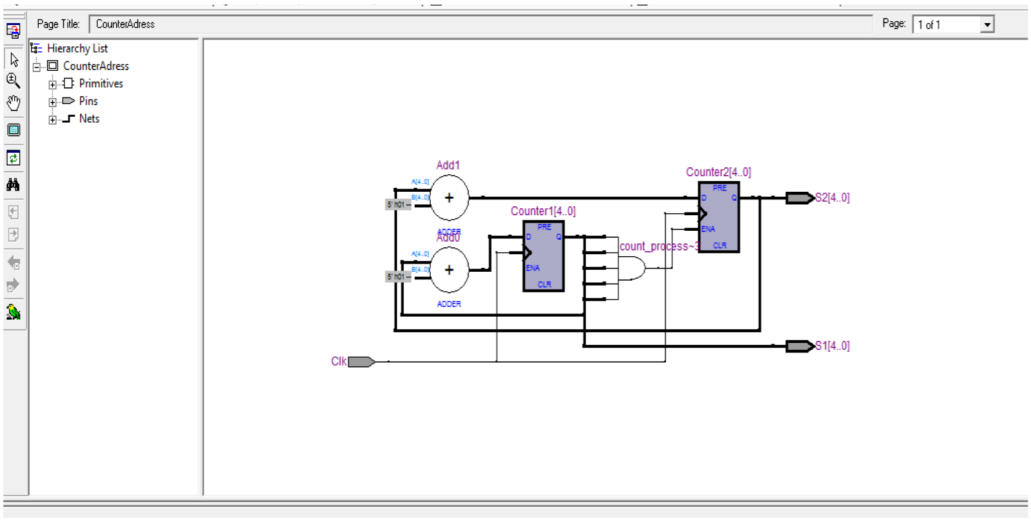


Figure: 4.9: Compteur D'adressage

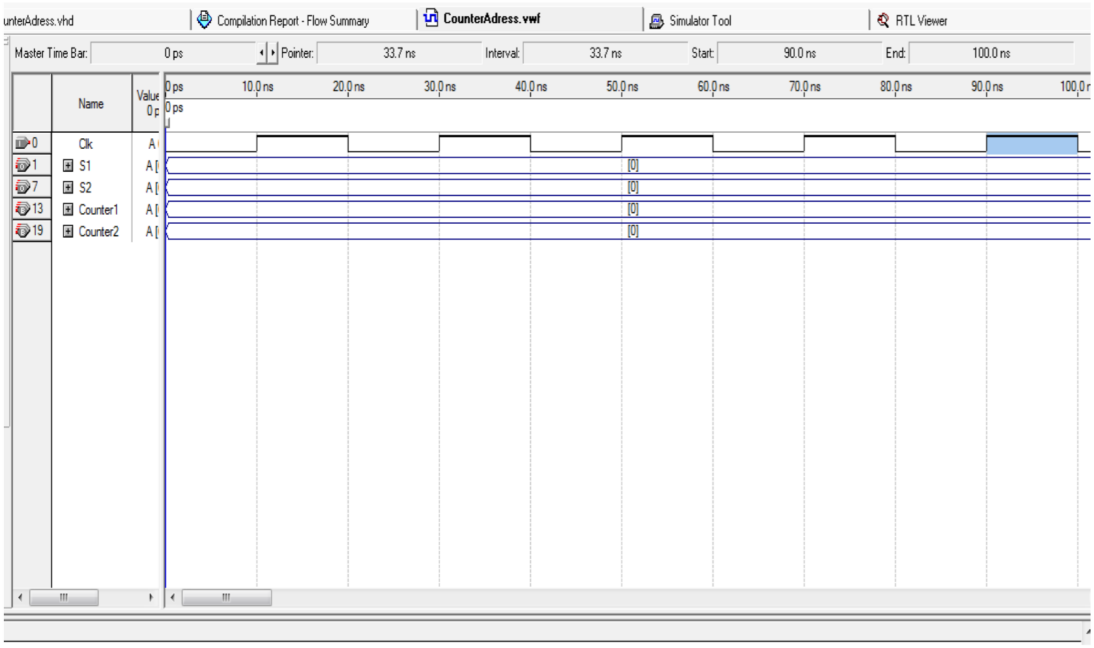
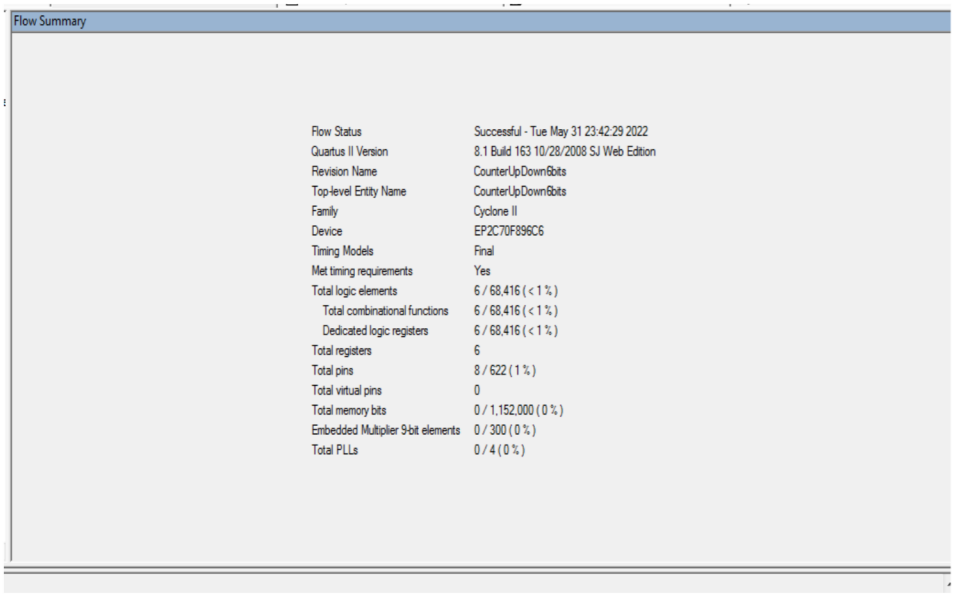


Figure: 4.10:simulation QUARTUS Compteur D'adressage

IV .7.Compteur / Décompteur:



Compilation rapport Compteur / Décompteur

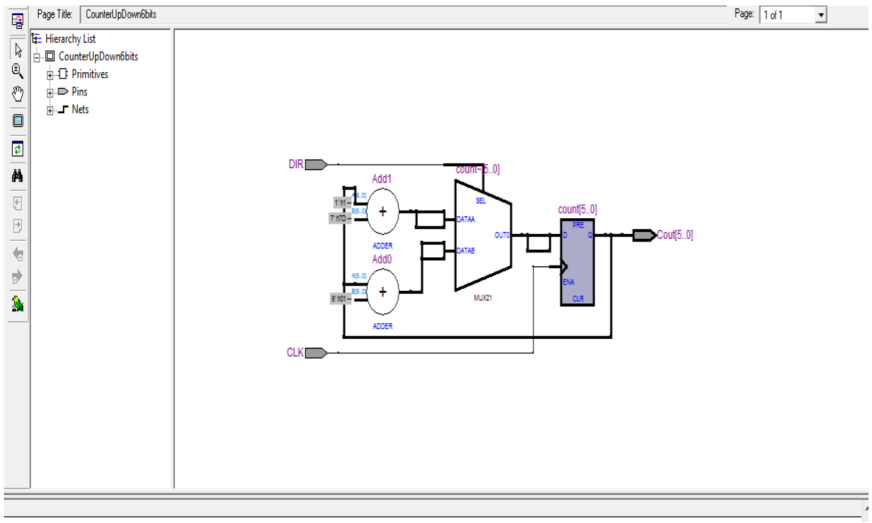


Figure: 4.11: Compteur / Décompteur

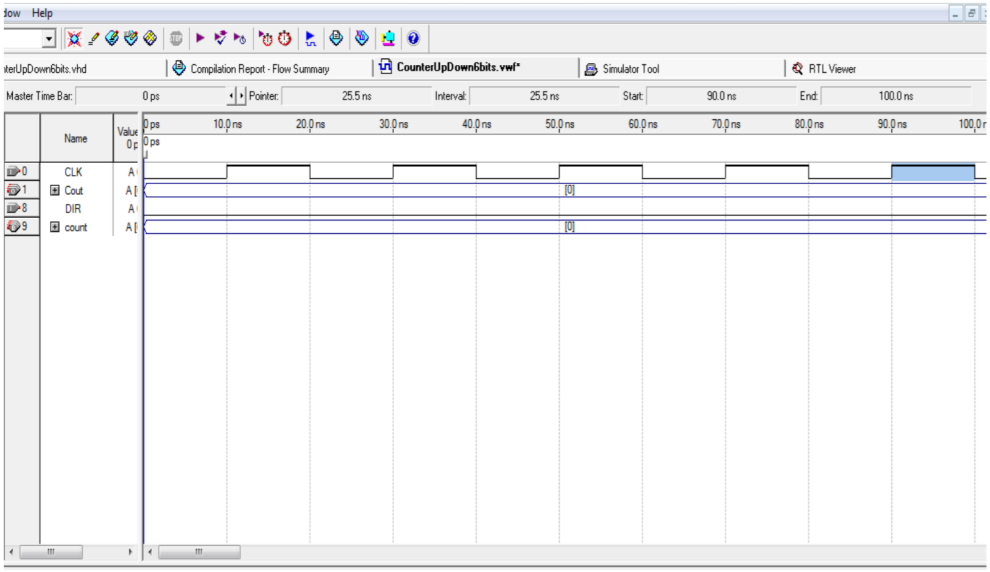
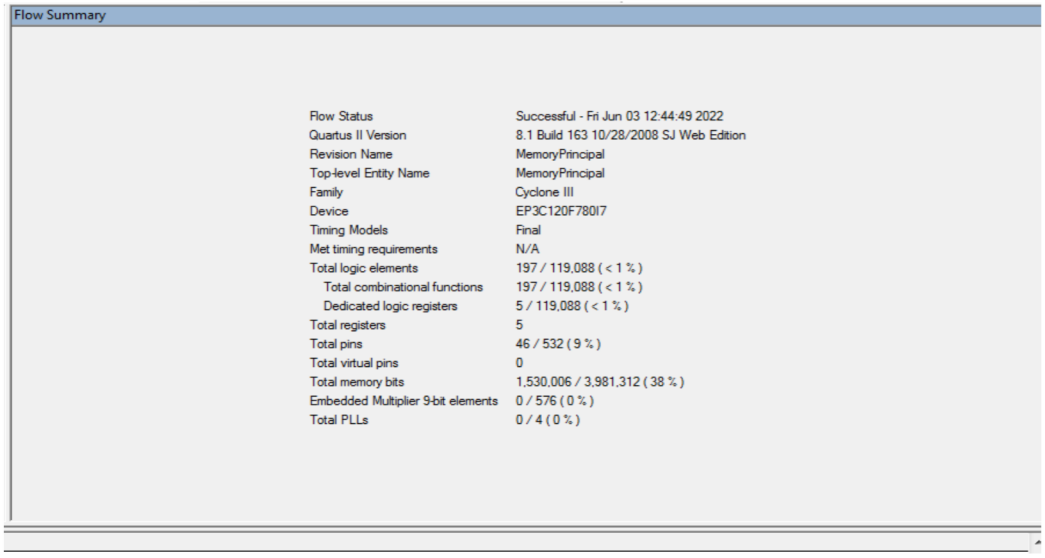


Figure: 4.12:simulation QUARTUS Compteur Décompteur

IV .8.Memoire Principale:



Compilation rapport memoire principale

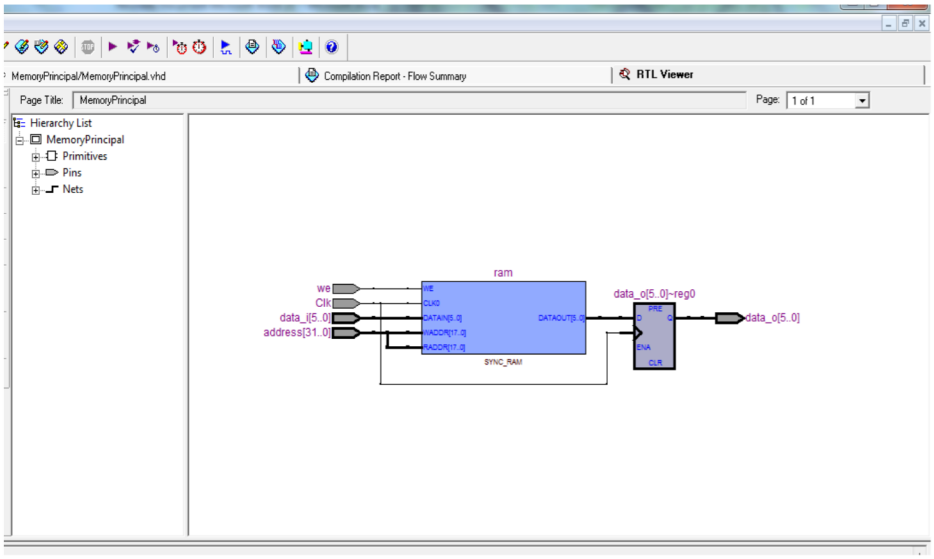
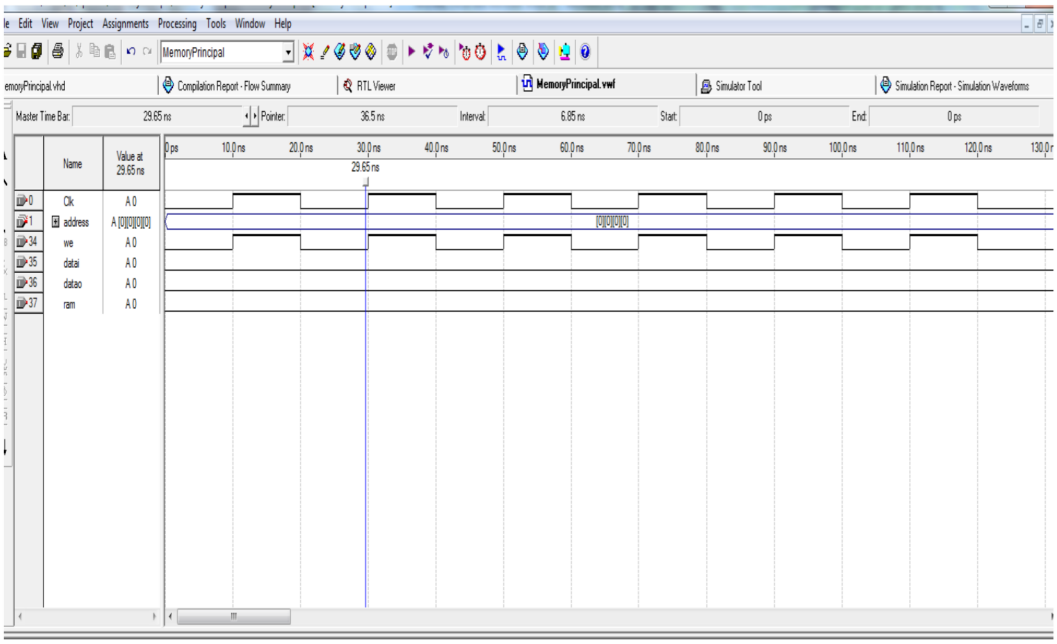


Figure: 4.13: memoire principale



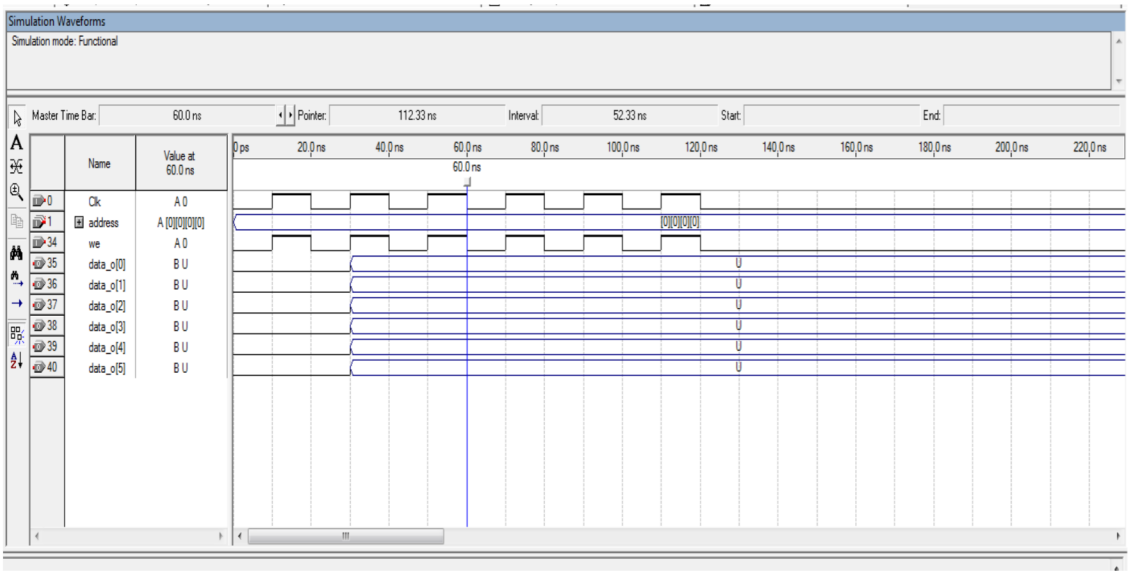


Figure: 4.14:simulation QUARTUS memoire principale

IV .9.simulation programme principale

Flow Status	Successful - Mon Jun 06 22:12:31 2022
Quartus II Version	8.1 Build 163 10/28/2008 SJ Web Edition
Revision Name	ProSimul
Top-level Entity Name	ProSimul
Family	Cyclone III
Device	EP3C120F780I7
Timing Models	Final
Met timing requirements	N/A
Total logic elements	81 / 119,088 (< 1 %)
Total combinational functions	80 / 119,088 (< 1 %)
Dedicated logic registers	29 / 119,088 (< 1 %)
Total registers	29
Total pins	172 / 532 (32 %)
Total virtual pins	0
Total memory bits	393,728 / 3,981,312 (10 %)
Embedded Multiplier 9-bit elements	0 / 576 (0 %)
Total PLLs	0 / 4 (0 %)

Compilation rapport programme principale

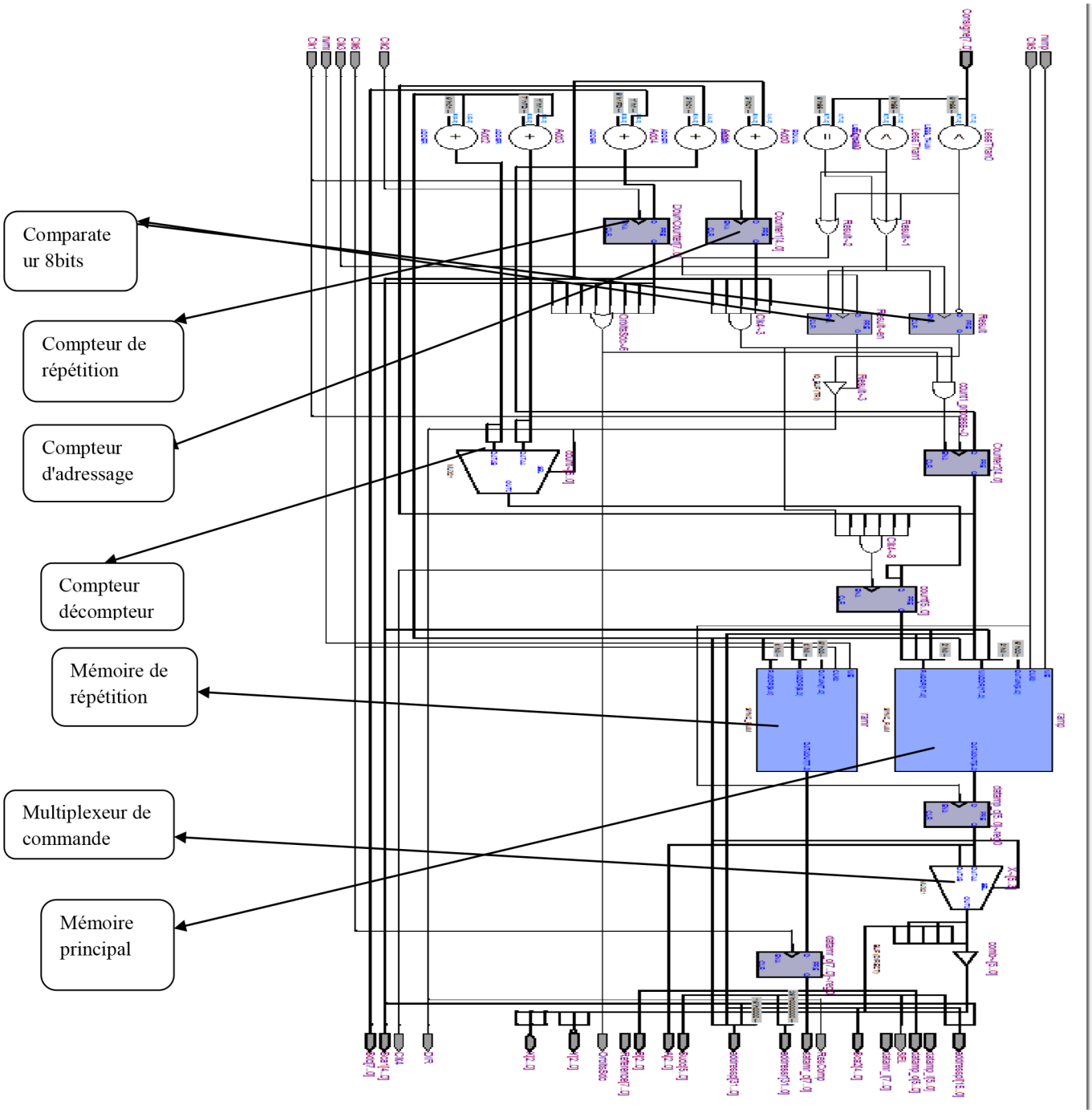


Figure: 4.15: programme principale

IV .10.conclusion:

Dans ce chapitre,nous avons implémenté sur la carte FPGA le modulateur de la technique RPWM configuraable à l'aide de l'outil ALTERA ,le langage utilse dans la conception et quartus ,en remarque que les résultats de l'implimentation son reèle.

CONCLUSION GÉNERALE

Conclusion génerale:

L'objectif de ce mémoire était limplementation d'un système de commande d'un onduleur basè sur la technique RPWM..

Nous avons commance par aborder quelque topologie de generateur RPWM numerique les plus repandues .

Nous avans choisi FPGA comme platforme materielle et avons presente brievement leur archetecture la conception a swivie la proche buttom-up utile pour les petite conception comme la presente.

Chaque bloc est alors decrit on longage VHDL et simule .

Ona ensuite ressemble les different bloc pour realiser le modulateur complet final et l'on aussi verifier par simulation.

Toutes les simulation etant satisfaisantes.

Finalement en passe a l'implimentation sur logiciel quartus

Le logiciel Quartus II est le numéro un en termes de performances et de productivité pour les conceptions CPLD, FPGA et ASIC, offrant le chemin le plus rapide pour convertir votre concept en réalité.

L'implémentation sur FPGA devrait toutefois être faite en considérant les contraintes pratiques, à savoir;

- La disponibilité des équipements.
- La bonne étude des boitiers qui conviennent.
- Le temps de réponse des boitiers.
- Les durées de transite des signaux.
- La technologie des boitiers en matière de réponse aux signaux.
- La synchronisation entre les différentes horloges (système séquentiel.(
- L'adaptation des horloges avec les autres signaux de validation et de lancement des actions.
- Etc…
- Du point de vue R.P.W.M., on devra exploiter tous les aspects théoriques, tels que;
- La variation de la fréquence en fonction du nombre de répétitions et de l'horloge du F.P.G.A.

- Les différentes techniques proposées (répétition alternée, répétition variable, répétition composée.
- Comme autre perspective, on pourrait aussi étudier la possibilité d'utiliser les D.S.P., outils qui permettent de minimiser l'espace mémoire et d'obtenir une bonne génération des signaux.

ANNEXE

Annexe:

C'est quoi V.H.D.L. [10][11]

V.H.D.L. vient de l'expression VHSIC Hardware Description Langage où VHSIC vient de Very high speed integrated circuits.

V.H.D.L. est un langage de description de circuits qui est en utilisation depuis les années 80 du vingtième siècle et qui a été adopté par le standard IEEE.

V.H.D.L. peut être utilisé pour deux objectifs;

1.Synthèse

2.Simulation

V.H.D.L. est similaire à tout langage de programmation en termes de style et de syntaxe. Mais à la différence que V.H.D.L. inclut plusieurs constructions de circuits spécifiques.

Terminologie V.H.D.L.

VHDL (Hardware Description Language(VHDL est un type de langage de programmation qui est utilisé pour la conception de circuits logiques programmables. HDL fournit la capacité de configurer et définir le comportement de circuits en termes de logiciel. V.H.D.L. comme Verilog sont les plus communs des HDLs.

Modélisation comportemental des circuits

C'est la définition du comportement du model en terme de relation entre les entrées / sorties. La structure interne du circuit n'est pas spécifiée expressément mais plutôt laissée à la synthèse. La focalisation est seulement sur la fonction du model.

Modélisation structurelle des circuits

C'est la définition des connexions entre les entrées et les sorties du composant de manière explicite. D'où la structure du model est définie par le concepteur. Cette méthode permet de modeler la conception, ce qui la rend très utile pour des conceptions étendues et complexes.

Dans les conceptions V.H.D.L., la méthode commune utilisée en pratique est de modeler des sous-composants comme comportementaux et ensuite utiliser la modélisation structurelle pour les connecter les uns aux autres de manière croissante.

Synthèse

C'est le passage du V.H.D.L. à l'implémentation des circuits.

Process

C'est l'unité de base d'un développement en langage V.H.D.L..

Débit de conception en V.H.D.L.

En langage V.H.D.L. le débit de conception est composé de 3 étapes:

Description (Coding) : C'est dans cette phase que le V.H.D.L. est rédigé (écrit). Le concepteur aura à transcrire la description à partir d'un brouillant ou d'une idée.

Simulation (Simulation) : Dans cette phase, la description V.H.D.L. préparée est testée avant d'être téléchargée sur un FPGA. Par la simulation certaines erreurs possibles seront évitées et/ou minimisées.

Synthèse (Synthesis) : Dans cette phase, la description V.H.D.L. est translatée dans le circuit, càd reconvertie en RTL. Ce process est assuré par des programmes de synthèse (synthétiseurs). Chose qui peut être faite manuellement, ce qui demanderait des connaissances avancées et un travail laborieux pour des conceptions importantes.

Syntaxe et architecture générale V.H.D.L. [09][10][11]

Quelques règles générales de syntaxe V.H.D.L.

- - Les descriptions V.H.D.L. sont classées en deux catégories;
- Descriptions synthétisables : Celles-là peuvent être complètement synthétisées et téléchrgées dans des FPGA pour implémentation.
- Descriptions non-synthétisables : Celles-là sont uniquement sujettes à la compilation et aux tests. Quoiqu'elles soient très faciles à la simulation, elles ne peuvent pas être translatées dans des circuits. De là elles ne peuvent pas être téléchargées dans les FPGA.

Donc, les concepteurs doivent faire attention au choix de la description pour implémentation ou bien simulation.

b) - V.H.D.L. a ses propres mots réservés, comme par Exemple : if, while, and, when, process, with," etc.

c) - Les instructions V.H.D.L. peuvent être de deux types;

- Concurrentes : Travaillent en parallèles comme des circuits.
- Séquentielles : Travaillent de manière séquentielle comme des langages de programmation ordinaires (exemple C, Java, etc.(.

d) - Les descriptions V.H.D.L. se terminent par (;), exemple : A<=3;

e) - V.H.D.L. n'est pas sensible à la casse. Exemple : "temps" et "TEMPS" représentent la même chose pour le programme de synthèse.

f) - Les espaces vides ou pleins sont ignores en V.H.D.L.. On peut donc utiliser les deux écritures pour formuler la description;

A <= C and D ; A <= CandD;

g) - Tout ce qui vient après le signe "--" est considéré comme commentaire et ignore par le programme de synthèse. Les commentaires sont des explications courtes utilisées afin d'éclaircir la description aux autres et aussi pour les utilisations futures.

A <= C + D; -- affecte à A l'addition de C et D L'expression en italique gras est donc ignorée par le programme de synthèse.

Architecture générale V.H.D.L. [09][10]

Une description V.H.D.L. typique consiste en quatre sections:

a) -. Entity

Entity définit l'interface de la description avec son environnement extérieur. Dans cette section les Entrées/Sorties sont définies. Elle est utilisée pour déterminer la relation du modèle avec son interface. Cette section montre les limites extérieures, les entrées et les sorties du modèle. Les ports aussi sont définis dans cette construction

La déclaration de Port : Les Entrées/Sortie sont déclarées dans cette section.

Trois types de Ports peuvent être définis avec V.H.D.L: .

- In : C'est le port qui détermine les signaux venant de l'extérieur du modèle.
- Out : C'est le port qui détermine les signaux allant vers l'extérieur du modèle.
- Inout : C'est le port bidirectionnel qui peut être utilisé à la fois en entrée et en sortie. Utilisé spécialement dans les applications mémoires.

La déclaration de Generic : Dans cette section, certains paramètres de l'Entity sont transférés dans les composants. L'information demandée peut être inscrite dans le mode de conception de cette manière.

b) -. Architecture

Cette construction est utilisée pour définir la fonctionnalité de la structure. Dans cette section, on retrouve les relations entre les ports définies dans l'Entity.

Chaque Architecture doit être déclarée dans une Entity. Une Entity peut contenir plus d'une Architecture. Toutes les expressions définies dans une Architecture sont traitées en parallèle.

c) Configuration

Elle détermine comment tous les sous-composants sont combinés pour devenir une description et comment les blocks sont connectés entre eux.

d) –Package

Package est une construction qui regroupe les déclarations dans le but de les utiliser dans différentes descriptions.

C'est une collection de déclarations qui peuvent être utilisées dans plus d'une description. Constantes, types de données, composants, fonctions et sous-routines peuvent être définis en Package, ainsi ça pourrait être utilisé dans plus d'une description.

Le concepteur définit les déclarations partagées dans le Package. Ainsi le concepteur pourra utiliser ces déclarations dans tous les modules et sous-descriptions. Donc le concepteur est soulagé du fait de répéter les mêmes expressions des fois et des fois.

La construction du Package

1.La déclaration de PACKAGE : Cette section est obligatoire (mandatée). Les déclarations de types et sous-programmes sont inclus dans cette section.

2.Le corps PACKAGE : Cette section est optionnelle. Elle contient les définitions des sous-programmes.

Définir un Package en V.H.D.L. est contraire à l'usage habituel d'un package. Il serait plus approprié de le définir dans une Library prédéfinie ou bien dans une Library créée spécialement.

Utilisation de PACKAGE

L'expression "Use" permet l'utilisation des déclarations (définies dans le Package) dans les

e) Library

Les Libraries sont des classeurs qui contiennent un ou plusieurs Packages.

- Library Source Une Library Source contient:
- Un Package Standard (Standart package.(
- Les Packages qui sont définis par IEEE (The packages which are defined by IEEE.(
- Les Libraries qui sont créées par les fabricants (fournisseurs) des FPGA.
- Certaines Libraries des unités qui sont référencées dans les conceptions elles mêmes.

Library de travail

C'est la Library qui contient la description V.H.D.L. du domaine d'application de la conception.

IEEE Library

IEEE Library contient les packages suivants:

- IEEE.STD_LOGIC_1164 : Contient le type std_logic et les fonctions annexes.
- IEEE.STD_LOGIC_ARITH : Contient les fonctions arithmétiques.
- IEEE.STD_LOGIC_UNSIGNED : Contient les fonctions arithmétiques non signées.
- IEEE.STD_LOGIC_SIGNED : Contient les fonctions arithmétiques signées.

Library Standard

Elle contient les Packages suivants:

C'est une Library construite, donc pas nécessairement une référence en V.H.D.L..

Les mots réservés V.H.D.L.

En V.H.D.L., Les mots spécifiques qui sont réservés ne peuvent pas être utilisés pour d'autres usages que ceux qui leur sont attribués. Ils sont listés dans le tableau suivant:

ABS	ELSE	NAND	SELECT
ACCESS	ELSIF	NEW	SEVERITY
AFTER	END	NEXT	SIGNAL
ALIAS	ENTITY	NOR	SHARED
ALL	EXIT	NOT	SLA
AND		NULL	SLL
ARCHITECTURE	FILE		SRA
ARRAY	FOR	OF	SRL
ASSERT	FUNCTION	ON	SUBTYPE
ATTRIBUTE		OPEN	
	GENERATE	OR	THEN
BEGIN	GENERIC	OTHERS	TO
BLOCK	GROUP	OUT	TRANSPORT
BODY	GUARDED		TYPE
BUFFER		PACKAGE	
BUS	IF	PORT	UNAFFECTED
	IMPURE	POSTPONED	UNITS
CASE	IN	PROCEDURE	UNTIL
COMPONENT	INERTIAL	PROCESS	USE
CONFIGURATION	INOUT	PURE	
CONSTANT	IS		VARIABLE
		RANGE	
DISCONNECT	LABEL	RECORD	WAIT
DOWNTO	LIBRARY	REGISTER	WHEN
	LINKAGE	REM	WHILE
	LITERAL	REPORT	WITH
	LOOP	ROL	
		ROR	XNOR
	MAP	RETURN	XOR
	MOD		

Les opérateurs V.H.D.L.

- Opérateurs arithmétiques
- Opérateurs logiques
- Opérateurs relationnels

Les fonctions arithmétiques et booléennes sont définies uniquement pour les types de données Standard Package.

Pour les autres types de données, des fonctions spéciales qui sont définies par une Library IEEE peuvent être utilisées dans les opérateurs arithmétiques et booléens..

Types de données définies en STD Library

Chaque type de données peut être trnsformé en different autres types en utilisant des fonctions.

a) -Bit

Ce sont les 0 et les 1 logiques. Pour ne pas les confondre avec les nombres entiers (0 et 1), ils sont déclarés entre guillemets (‘0’ et ‘1’).

STD_LOGIC

C’est le type de données le plus communément utilisé dans le langage V.H.D.L.. La Library IEEE est placée dans le package STD_LOGIC_1164. Quoiqu’il doit avoir huit valeurs différentes, trois seulement d’entre elles sont le plus souvent utilisées ; ‘’1’’, ‘’0’’ et ‘’Z.’’

’1‘:Logique (Logic)

’0‘:Logique (Logic)

‘X’ : Inconnu (Unknown)

‘U’ : Non utilise (Uninitialized)(Undifined)

‘Z’ : Haute impédance (High impedance)

‘H’: Logique haute (Logic high)(Weak 1)

‘L’ : Logique basse (logic low)(Weak 0)

‘W : Logique inconnue (Logic unknown)(Weak unknown)

‘D’ ou bien ‘-‘ : Indifférent (don’t care)

STD_LOGIC_VECTOR et BIT_VECTOR)

Il est utilisé pour décrire des données TD_LOGIC multiple. Ce type est défini sous forme de tableau dont les éléments sont des bits. La taille du tableau est définie dans la partie déclaration. Les valeurs des tableaux sont déclarées entre doubles guillemets.

VECTOR

Les tableaux à une seule dimension qui consistent en des éléments logiques sont appelés VECTIORs. Les tableaux qui sont constitués de BITs et STD_LOGIC sont respectivement appelés BIT_VECTOR et STD_LOGIC_VECTOR.

b) -Boolean

Ce sont les valeurs booléennes. Ils prennent les valeurs « vrai » ou « faux ». Les expressions conditionnelles (<, >, <=, >=, =) sont utilisées conjointement avec les opérateurs. Ils sont définis dans le PACKAGE STD.

c) -Integer

Ce sont les nombres entiers positifs et négatifs. Leur rang et leur classement sont fixés selon l'utilisation. Si on attrobue une valeur hors du rang défini on obtient un message erreur.

Ils sont utilisés pour les opérateurs standards comme les opérations arithmétiques et les comparaisons.

d) -Natural

Ce sont les nombres entiers supérieurs ou égaux à zéro.

e) -Positive

Ce sont les nombres entiers supérieurs à zéro.

f) -Character

Ce sont les caractères ASCII. Ils sont définis dans le Package STD. Il y'en a 256 comprenant coprenant également "=", "<", ">", "<=", ">=" et."≠"

g) _TIME

Ce sont les types temps (ps=picoseconde, ns=nanoseconde, µs=microseconde, ms=M.L.I.lisecond, sec=seconde, min=minute, hr=heure). Ils sont écrits entre doubles guillemets. Ils ne sont pas synthétisés, mais utilisés pour donner un message lors de la simulation.

h) -REAL

Ce sont les nombres reels.

i) -STRING (POSITIVE)

Ce sont des chaines ou tableaux de caractères. La taille et le rang sont déterminés dans la partie déclaration. Les types STRING sont utilisés dans les opérations de "CONCATENATION", "AGGREGATE", "SLICE."

Il est possible de comparer des STRINGs en utilisant les opérateurs "=", "<", ">", "<=", ">=" et."≠"

Ils sont écrits entre doubles guillements. Ils ne sont pas synthétisés, mais utilisés pour donner un message lors de la simulation.

j) -DELAY_LENGTH

Il représente la valeur de temps (TIME) qui est supérieure ou égale à zero.

k) -COMPOSITE TYPE

C'est le type de données ou d'objets qui consistent en de multiples elements. Il y en a deux types « ARRAY » et « RECORD ». Les éléments dans RECORD peuvent être de différents types. Toutefois, tous les éléments de ARRAY doivent être de même type. Les éléments de ARRAY sont tous indexés.

l) -SIGNED-UNSIGNED

Ces deux importants types sont défines dans le Package STD_LOGIC_ARITH. Ces types fournissent l'accès individuel à chaque bit dans différentes valeurs numériques, à la différence de INTEGER. Avec cette propriété, ils sont siM.L.I.aires au type BIT_VECTOR.

Toutefois, ces types de signaux et d'opérations variables sont définis comme atant numériques dans le Package STD_ARITH.

Unsigned

Celà veut dire une valeur numérique non signée. Le bit le plus à gauche est le bit le plus significatif (MSB.(

Signed

Le bit le plus à gauche est le bit de signe. Où '1' indique que c'est un nombre négatif et '0' pour un nombre positif.

m) -FLOATING POINT TYPE

C'est un type de nombres réels défini dans un rang spécifique. Il peuvent être utilisés dans les opérations arithmétiques.

- Ils ne sont pas synthétisables.
- Ils ne sont pas généralement utilises.
- S'il est nécessaire d'opérer entre un nombre entire et un nombre reel, une "CONVERSION" doit être utilisée.

Types de données définies dans le package IEEE.STD_LOGIC_1164

a) - Constant

C'est un objet qui ne peut pas être changé une fois sa valeur déterminée. Il est utilisé pour améliorer la lecture de la description.

b) - SGNAL

Signal est un lien physique (fil, câble) qui permet la communication entre les Processes dans l'Architecture. Il peut être déclaré dans une Entity, une Architecture ou un Package quelconques.

Le symbole d'affectation(=>)

Le symbole d'affectation (<=) est utilisé pour donner une nouvelle valeur au signal. Il est siM.L.I.aire à l'opérateur (=) qui est utilise dans les autres langages de programmation comme C/C++, Java.…

Toutes les valeurs:

Reg<="1111;"

Reg<=x"F"; (hexadecimal(

L'affectation à un seul bit:

Reg(2)<='0;'

Affectation multi-bits (Trabche de bits: (

Reg(1 to 2)<="10;"

c) - COMPONENT

Un composant est une partie d'une Entity / Architecture. Il affecte un sous-système dans une autre Architecture. Tout simplement, un composant permet d'inclure des modules créés avec des descriptions structurelles dans des descriptions V.H.D.L. principales.

Types de données définies par l'utilisateur

- **Tableaux**

Les sous-éléments d'un tableau sont tous du même type. Chaque élément d'un tableau a un index (index number) dans un rang. Il y a un seul index pour un tableau à une seule dimension. L'index est d'un rang supérieur au tableau pour les tableaux multidimensionnels.

L'utilisateur peut définir des types de données avec leurs noms et leurs valeurs. La description peut alors être plus claire. Ces types sont utilisés surtout dans les machines d'états limitées.

Affectation de signaux concurrents

Il y a trois formes d'affectation conditionnelle de signaux:

- Affectation basique de signaux (Basic signal assignment(
- Affectation conditionnelle de signaux (Conditional signal assignment(
- Affectation sélective de signaux (Selected signal assignment(

a) - Affectation basique

Syntaxe:

signal_name <= expression

b) - Affectation conditionnelle

Syntaxe:

Signal_name<= expression when condition else

expression when condition else

expression;

c) - Affectation sélective de signaux

Syntaxe:

With choice_expression select value <= expression when choice,

Expression whenchoice,

Expression when others;

Process

Quoique Process est un état concurrent, il contient un ensemble d'états séquentiels. Dans cet état tous les Processes restent en exécution jusqu'à ce qu'ils rencontrent une liste sensible (Sensitivity list) ou bien un état d'attente (wait statement.(

Syntaxe:

Label: Process (Sensitivity list(

contant declaration

type declaration

variable declaration

Begin

--sequential statement

--sequential statement

end process;

Toutes les expressions dans un Process sont exécutées une à une de haut en bas. Donc, seule la dernière valeur affectée reste valable.

Après l'exécution de ce Process, la valeur de ''b'' devient 6 et celle de ''a'' devient 3 tant que c'est sa dernière valeur attribuée.

Liste sensible (Sensitivity List)

S'il y a une liste sensible (Sensitivity liste) dans un Process, le Process suspendu devient actif au

moment où l'un des signaux listés change. Process peut contenir plus d'une expression d'attente (Wait expression)

Une liste sensible et une expression d'attente ne peuvent pas être utilisées dans un même Process.

Ces deux expressions sont identiques. Dans les deux Processes, la valeur de ''input1 and input2'' est affectée à la sortie ''output''. Le Process d'affectation est exécuté à la suite de tout changement survenant sur input1 ou input2.

Une Architecture peut contenir plus d'un Process. Dans une même Architecture, tous les Processes sont exécutés de manière concurrentielle.

Affectations conditionnelles

a) - if-then

If condition then

Statements

elsifcondition then

statements

else

statements

end if;

b) - Case

Case choice_expression is

When choice=> assignment;

When choice =>assignment;

When choice =>assignment;

end case;

c) - Loops

- Loop
- Une boucle (Loop) infinie reste active aussi longtemps qu'il n'y ait pas d'expression de sortie.
- While Loop
- While loop est utilisé quand un état est attribué avec une condition.
- For Loop
- For loop est utilisé dans une gamme de paramètres en boucle.
- FOR-LOOP représente les états séquentiels qui vont être répétés.
- Les expressions sous LOOP doivent continuer à être exécutées aussi longtemps que les conditions sont satisfaisantes. Ensuite le premier état après la fin de la boucle est abordé.
- FOR-LOOP est utilisé dans les Processes, Fonctions, Procédures pour les états séquentiels.
- Un autre cas de FOR-LOOP est le FOR-GENERATE.
- End loop;
- Des expressions de sortie (Exit) et de passage (Next) peuvent être utilisées dans une boucle (Loop.(
- d) - Wait
- WAIT est utilisé pour suspendre une opération ou bien un Process. Un changement de signal ou bien une période de temps peuvent être utilisés comme condition à 'WAIT.'
- Wait on signal_name
- L'exécution attend l'avènement d'un changement quelconque dans le signal.
- Wait unti condition_expression
- L'exécution attend de rencontrer une condition.
- Wait for time_expression

L'exécution attend la durée de l'intervalle de temps indiqué dans ''time_expression''. Il n'est pas sujet à une synthèse. Il n'est utilisé que pour un but de simulation.

e) - When-Else

WHEN est utilisé avec ELSE pour choisir entre des état conditionnels.

f) - With

WITH est utilisé pour choisir entre des états conditionnels concurrents.

g) - FOR GENERATE (Etats concurents)

Label: for variable in bottom_limit to upper_limit [upper_limit downto bottom_limit] generate

Begin

-statements

-statements

end generate;

Déclaration de VARIABLE

Les variables peuvent être déclarées dans les Processes, les fonctions et les procédures. VARIABLE est un objet qui peut avoir différentes valeurs.

Affectation des variables

Cela veut dire remplacer la valeur d'une VARIABLE par une nouvelle valeur. Le symbole " := " est utilisé pour l'affectation des variables. Une telle affectation est effectuée instantanément. Aucune temporisation ne survient.

Déclaration de SIGNAL

Les SIGNALs sont des liaisons physiques entre les blocs du hardware à concevoir.

Affectation de SIGNAL

Différentes manières sont utilisées pour affecter un signal.

Toutes valeurs

Reg:=”1111;”

Reg:=x”F”;(hexadecimal)

Affectation d'un seul bit

Reg(2):=’0;’

Affectation multi-bits

Reg(1 to 2):=”10;”

Simple guillemets (‘ ’) pour l’affectation d’un seul bit.

Double guillemets (" ") pour l’affectation multi-bits.

ACCESS

C’est un type qui permet l’accès à un objet, en utilisant un identificateur (Identifier/Allocator)

Type identifier;

Le type Access permet de changer la donnée pour laquelle la valeur est au préalable inconnue et qui doit apparaitre seulement lors de la simulation. Pour ce faire, un identificateur (Allocator) est utilisé comme pointeur.

Indication de sous-types : C’est le type d’objet qui est marqué par une valeur. Il peut être un scalaire quelconque, composé ou un autre type d’accès à l’exception du type fichier (any scalar,composite, or other type of access, except File type)

Seulement des objets variables peuvent être utilises comme types Access.

La valeur par défaut pour Access est NULLE.

Le type Access permet des listes dynamiques d’objets qui sont créés durant la simulation. Ils sont essentiellement des enregistrements qui consistent en des éléments à accès. Ces éléments peuvent être de différents types déclarés au préalable. Pour ce faire, un type incomplet est utilisé en annonçant en premier lieu les variables ensuite les déclarer plus tard.

Une déclaration incomplète doit être complétée dans la même partie déclarative avec le même nom plus tard. Ce nom est utilisé seulement ici.

AGGREGATE

C’est un Process d’Affectation d’une ou plusieurs valeurs à des éléments d’enregistrements ou de tableaux (matrices). L’Affectation est faite en mettant une virgule entre les éléments sous des parenthèses.

ALIAS

Il permet de grouper un signal qui a été défini avec des noms différents.

ALLOCATOR

Ce sont des Processes qui créent des variables et des objets anonymes auxquels on accèdera plus tard dans la description.

ARCHITECTURE

Cette section décrit le fonctionnement et la structure interne de l'Entity. Elle est directement liée à l'Entiy. Elle est définie selon que l'Entiy de la decription est comportementale, structurelle ou débit de données.

ARRAY

C'est un tableau composé d'une série de sous-éléments qui sont tous du même type. Chaque élément a un index dans un tableau avec un certain rang. Un tableau à une dimension a un index et un tableau multidimentionnel a un nombre d'index égal à sa dimension.

ASSERT

C'est une expression qui vérifie l'exactitude d'un état conditionnel et remets un message erreur si c'est faut.

ATRIBUTE Prédéfini

L'utilisateur peut utiliser un ATTRIBUTE prédéfini. Les ATTRIBUTEs prédéfinis sont les expressions qui formulent la fonction, le type et le rang d'un objet.

ATTRIBUTE défini par l'utilisateur

L'utilisateur peut définir son propre ATTRIBUTE. Dans ce cas il est possible de collecter des informations additionnelles ou objets dans la description.

BLOCK

Il est utilisé pour dissocier le programme en de petits blocs. Ce qui le rend plus compréhensible.

CASE

Il est utilisé pour sélectionner des Processes alternatifs selon les valeurs possibles d'une variable.

COMMENT

Pour rendre la description facilement compréhensible, on a besoin d'émettre des commentaires et des explications qui lui sont liés, dans des endroits particuliers. Cette section n'affecte en rien la description.

COMPONENT

Il représente la paire Entity/Archoitecture. Component represents the entity-architecture pair. Il désigne un sous-système qui sera appelé à être utilisé

momentanément dans une autre Architecture de manière hiérarchique. It appoints a sub-system that is to be called in another architecture (instantiate) hierarchically.

CONCATENATION

Cela consiste à relier deux tableaux unidimentionnels de même type, entre eux en utilisanr un

opérateur '&'. L'effet de cette liaison est de mettre la valeur du côté gauche du signe '&' au début dans le résultat.

CONFIGURATON

C'est une structure qui détermine comment des sous composants veinnent ensemble constituer une description entière, les blocs sont connectés les uns aux autres.

CONSTANT

C'est un objet dont la valeur est déterminée une seule fois et ne pourra pas être changée plus tard. Les CONSTANTs peuvent être déclarés directement ou bien peuvent être des sous-éléments d'autres CONSTANTs.

DELAY

C'est une fonction de temporisation.

ENTITY

ENTITY définit l'interface entre la description et son environnement extérieur. Les ports d'Entrée/Sortie sont déclarés dans cette partie.

Il y a une seule ENTITY dans une description V.H.D.L.. Il pourrait y avoir plus d'une ARCHITECTURE ou CONFIGURATION, dans le but de maitriser facilement la description.

ENUMERATION

Les valeurs classées sont listées dans le type ENUMERATION. Les éléments sous le type ENUMERATION doivent être différents les uns des autres. Toutefois, des éléments de mêmes nomination peuvent être utilisés sous des types d'ENUMERATION différents. Cela est appélé surcharge.

Surcharge dans le type ENUMERATION

Dans une description V.H.D.L., si un élément est utilisé sous plus d'une ENUMERATION une surcharge apparait (annoncée.(

Afin de ne pas causer d'erreurs ou d'absurdités, on doit spécifier le type d'ENUMERATION qu'on va utiliser dans la description.

EVENT

C'est un attribut qui retourne des informations sur les signaux, concernant le changement de leur valeur ou bien valeur prévue.(

Si la valeur d'un signale présente un EVENT (sa valeur a changé) dans le cycle de simulation courant, cet attribut change vers 'VRAI.'

EVENT est un attribut V.H.D.L. important et communément utilisé.

N'importe quel changement sur un signal peut être détecté en utilisant EVENT.

Son rôle est basé sur la vérification de n'importe quekl changement sur les flancs du signal horloge.

EXIT

Il est utilisé pour sortir d'une boucle. Si cet état contient une condition, cela veut dire que la sortie de la boucle dépend de cette condition.

FILE

FILE est un objet qui peut être utilisé comme un type de donnée V.H.D.L. après sa déclaration. Ce type aide à créer des fichiers sous le système d'un ordinateur et d'y accéder.

FUNCTION

FUNCTION est un sous-programme qui consiste en des états de renvoi de valeurs. FUNCTION contient uniquement des paramètres d'entrée. Chaque FUNCTION renvoie seulement une valeur.

GENERATE

Il est possible de regrouper des composants (COMPONENTs) siM.L.I.aires ensemble sous un seul composant (COMPONENTs), de les définir et de les répéter en utilisant GENERATE.

a) - FOR GENERATE

Label : for parameter in range generate

---declarations

Begin

--concurrent_statements

end generate Label;

b) - IF GENERATE

Label : if condition generate

---declarations

Begin

--concurrent_statements

end generate Label;

GENERIC

GENERIC est une structure V.H.D.L. optionnelle qui est utilisée pour transférer quelques paramètres de composants à ENTITY.

Les composants qui ont la même structure de base comme les bus et les signaux, peuvent être utilisés dans différentes conceptions en utilisant GENERIC.

GENERICs peuvent être déclarés dans des BLOCKs, des ENTITIES ou bien des COMPONENTs.

A la différence des CONSTANTs, les GENERICs peuvent être déclarés dans des COMPONENTS ou bien des CONFIGURATIONs et être changés de l'extérieur.

generic (generic_arayüz_listesi; (

GROUP

Cette structure V.H.D.L. est utilisée pour regrouper des ENTITIES. Des ATTRIBUTEs sont utilisés dasn les ENTITIES. Les ATTRIBUTEs définis par l'utilisateur peuvent être reliés à toutes ENTITIES indépendamment. Ainsi chaque ENTITY a son propre ATTRIBUTE.

Déclaration d'intérimaire de GROUP

Afin de déclarer un intérimaire de GROUP, les GROUPs qui sont liés à des ENTITIES dans une certaine classe. Ces classes peuvent contenir une ENTITY, ARCHITECTURE, CONFIGURATION, PROCEDURE, FUNCTION, PACKAGE, TYPE, SUBTYPE, CONSTANT, SIGNAL, VARIABLE, COMPONENT, LABEL, LITERAL, UNITS, GROUP ou FILE. Chaque classe d'ENTITY est une classe d'ENTITY dans un GROUP.

Déclaration de GROUP

Les ENTITIES sont connectées par leurs propriétés dans la déclaration de GROUP. Cette section

consiste en des identificateurs, nom d'intérimaire de GROUP, et liste de GROUP. Un identificateur représente un GROUP, nom d'intérimaire de GROUP représente une déclaration de tâche pour GROUP, liste de GROUP représente les ENTITIES qui appartiennent à ce GROUP.

Attributs de GROUPs

La spécification d'un ATTRIBUT de GROUP est réalisée de manière siM.L.I.aire que les autres spécifications d'ATTRIBUTs. En premier lieu, l'ATTRIBUT d'un TYPE donné est déclaré et spécifié pour le GROUP donné dans sa partie Déclaration qui contient la déclaration du GROUP particulier.

GUARD

GUARD est une expression logique qui a un TYPE BOULEAN dans le BLOCK. L'expression GUARD dans une STRUCTURE est utilisée pour contrôler l'Affectation du signal courant dans le block. Les Affectations de signaux sont traitées aussi longtemps que la condition de GUARD écrite du côté droit est vraie. Dans le cas où elle est fausse, aucune action n'est prise.

L'expression GUARD se réfère au signal qui prend place dans la déclaration de block sous le nom du GROUP. Le signal GUARD peut être vu seulement dans le block en question.

Etat IF

L'état IF est une structure qui dépend d'une ou plusieurs conditions.

LIBRARY

library library_name; use library_name:package_name.all;

IEEE LIBRARY

library IEEE;

IEEE PACKAGES

use IEEE.math_complex.all;

use IEEE.math_real.all;

use IEEE.numeric_bit.all;

use IEEE.numeric_std.all;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_misc.all;

use IEEE.std_logic_signed.all;

use IEEE.std_logic_textio.all;

use IEEE.std_logic_unsigned.all;

use IEEE.STD_LOGIC_ARITH.all;

NEXT

Cette expression est utilisée pour compléter un des états répétés dans une boucle. Elle est utilisé en conjonction avec une expression conditionnelle.

L'état NEXT est utilisé pour passer au PROCESS itératif suivant. Pas pour attendre la fin de la boucle.

"Next" statement is used to move to the next iterative process, notwithstanding ending the loop.

next;

NULL

Cette expression veut dire rien à traiter. Tant que l'expression NULL ne fait rien, l'état suivant est traité. En général, elle est utilisée avec l'état CASE, quand l'utilisateur veut que le programme ne fait rien pour une condition spécifiée.

SHIFTING OPERATORS

Un opérateur SHIF est utilisé pour des PROCESS de décalage appliqué à un tableau de type BIT et BOOLEAN.

PACKAGE

PACKAGE est une unité de regroupement de déclarations à utiliser dans différentes conceptions.

Déclaration de PACKAGE

package package_name is

package_declaration

end package package_name;

Le corps de PACKAGE

Des sous programmes sont déclarés et des valeurs de constantes déférées dans cette section.

PHYSICAL TYPE

Le type PHYSICAL veut dire que le GROUP de valeurs Numériques indique une quantité. Ça consiste en une valeur entière et une unité de mesure.

PORT

Il y a trois types de PORTs communément les plus utilisés.

a) _ INPUT

Port_name: in type;

Data: in STD_LOGIC;

b) - OUTPUT

Port_name: out type;

Output : out STD_LOGIC_VECTOR(3 downto 0;(

Output : out STD_LOGIC_VECTOR(0 to 3;(

c) - INOUT

Ce type de PORT peut être utilisé à la fois en entrée et en sortie. Il est utile pour les bus de données dans les applications de stockage de bases de données comme les RAMs par exemple.

Port_name : inout type;

PROCEDURE

PROCEDURE est siM.L.I.aire à FUNCTION dans le fonctionnement. Mais PROCEDURE ne retourne pas les valeurs comme FUNCTION. Chaque entrée et sortie peut être définie dans PROCEDURE. Cette caractéristique fait la différence entre les deux.

Déclaration de PROCEDURE

procedure procedure_name (inputs seperated with commas: in type;

outputs seperated with comması : out type;(

Le corps de PROCEDURE

procedure procedure_name (inputs seperated with commas: in type;

outputs seperated with commas : out type;(

--constant and variable declarations

Begin

--procedure_body

end procedure_name;

PROCESS

PROCESS est un état où des PROCESS séquentiels indépendants sont définis dedans.

RANGE

RANGE est utilisé pour déterminer des sous ensembles de valeurs numériques.

RECORD

RECORD est un type de données composé qui consiste en des éléments de types différents. C'est la différence la plus importante avec ARRAY.

REPORT

L'expression REPORT est siM.L.I.aire à l'expression ASSERT. Elle est utilisée pour montrer un message à l'utilisateur dans le but de l'aider durant le débogage. La différence avec ASSERT est d'utiliser un état conditionnel pour le message.

RESUME

Les PROCESSes peut être suspendus avec l'expression WAIT. Ces PROCESSes suspendus peuvent être rappelés quand les conditions définies deviennent vraies. Si le PROCESS n'est pas reporté les opérations suspendues deviennent actives dans le même intervalle de simulation.

RETURN

L'expression RETURN est utilisée pour attribuer le résultat d'une PROCEDURE ou bien FUNCTION.

Un sous programme (FUNCTION ou bien PROCEDURE) peut être arrêté en utilisant l'expression RETURN dedans. Le sous programme reprend inconditionnellemnt.

SLICING

La méthode SILICING est utilisée pour diviser des tableaux en parties. Alors, les parties consécutives dans un tableau à une dimension, sont dites ''SLICEs.''

SUBTYPE

Il divise les TYPEs principaux en des sous ensembles.

TYPE

TYPE est un ensemble de valeurs et opérations. Chaque objet V.H.D.L. doit avoir un TYPE. Les opérations et les affectations ne s'effectuent pas sans TYPE.

USE

L'expression USE rend la déclaration utilisable selon les choix.

Règles de codage V.H.D.L.

Comme dans tous les langages de programmation, il y a beaucoup de règles de codage V.H.D.L.. Ces règles sont nécessaires pour aviter les erreurs en écrivant un programme. Les règles de codage s'applique plus et spécialement sur les identificateurs. Les noms d'objets comme ENTITY, PROCESS, FUNCTION, PROCEDURE sont appelés identificateurs. Il y a deux genres d'identificateurs ; Basiques et Etendus.

basic_identifier ::= {letter / digit / underscore{

extended_identifier ::= \graphical characters\

NB:

- Les identificateurs de base ne peuvent pas prendre plus d'une ligne.
- Les identificateurs de base consistent en des nombres et digits.
- Les identificateurs de base commencent par un caractère.
- Les caractères spéciaux ne sont pas permis dans les identificateurs de base. Seulement les caractères de l'alphabet latin sont autorisés.
- L'espace n'est pas permis dans les identificateurs.
- Les mots réservés V.H.D.L. ne peuvent pas être utilisés comme identificateurs.
- Les identificateurs étendus sont supportés par le standard 93 (V.H.D.L. 93 Standard). Tous les mots écrit entre les caractères ‘’’ sont considérés comme identificateurs étendus.
- Les mots réservés V.H.D.L., les espaces, les caractères non ASCII sont permis dans les identificateurs étendus.
- Les espaces et les lignes vides sont insignifiants en V.H.D.L.

GUIDE QUARTUS II

Quartus II est un logiciel de CAO destiné à la conception de circuits logiques mettant enœuvre des composants programmables du constructeur Altera. La programmation d'un circuit se décompose en 4 phases :

1. Saisie de la description du circuit. Pour cela 3 outils sont utilisables :

• description sous la forme d'un schéma électronique,

• description textuelle en utilisant un langage de programmation (VHDL ou

AHDL),

• description sous la forme de chronogramme.

2. Vérification de la description par compilation du circuit,

3. Vérification du bon fonctionnement de la description par simulation logico-temporelle. A ce

stade, il peut-être nécessaire de reprendre à la 1ère phase,

4. Programmation du circuit physique.

L'objectif de cette annexe est de vous guider dans vos premiers pas sous Quartus II. Vous

apprendrez à créer un projet, compiler, assigner les broches de votre CLP, simuler puis programmer

votre CLP pour tester votre projet.

I. CREATION D'UN PROJET

Pour le logiciel Quartus II, un projet consiste en un ensemble de fichiers de conception, de fichiers

d'assignation, de fichiers de simulation, d'options de configuration et d'informations sur le projet. Le module Création d'un Projet du didacticiel vous guidera à travers les étapes qui sont nécessaires à la création de votre projet.

Avant de commencer, créez un répertoire « TP_quartus »sous votre compte. Puis créez un sous répertoire du nom de votre projet sous « TP_quartus ». Attention à ne pas mettre d'espace dans vos noms de répertoire !!

Pour créer un nouveau projet, suivez ces étapes :

1. Sélectionnez New Project Wizard... dans le menu File.

2. Il se peut qu'une page d'introduction apparaisse la première fois que vous ouvrez le guide New Project

Wizard; cliquez sur Next pour passer à la première page du guide.

3. Entrez le nom du répertoire dans lequel vous désirez travailler ou choisissez le répertoire avec [...]

4. Entrez le nom du projet.

Note : Le nom du fichier top-level est par défaut le même

que le nom du projet que vous venez d'entrer. Vous pouvez

modifier ce nom à votre guise.

5. Cliquez sur Next pour faire apparaître la page Add Files. Puisque c'est un nouveau projet,

il n'existe pas encore de fichier à rattacher. Cependant, si vous voulez utiliser des composants ou des fichiers créés précédemment, vous pouvez les ajouter en indiquant leur nom ou en utilisant [...] pour sélectionner les fichiers et ensuite cliquez sur Add

6. Cliquez sur Next. Vous voilà maintenant à la page Family & Device Settings. Assurez-

vous de choisir la famille CYCLONE II et la cible EP2C35F672C6.

Si vous filtrez les noms de cible en fixant le nombre de broches à 672 et en choisissant lavitesse la plus rapide, le circuit apparaît alors en premier dans la liste.

7. Cliquez sur Next. Maintenant, la page EDA Tool Setting apparaît. Cette page permet de spécifier si on utilisera les outils par défaut de Quartus II ou si l'on fera appel à des outils externes.

A cette étape, il faudra simplement spécifier que l'on utilisera l'outil de simulation Modelsim-Altera.

8. Cliquez sur Next. La page Summary montre les options que nous avons choisi afin deconfigurer le projet.

9. Cliquez sur Finish. Le projet est maintenant créé. Le nom de l'entité top-level du projet apparaît dans le signet Hierarchies dans la fenêtre du navigateur du projet

II. DESCRIPTION DU CIRCUIT

Vous allez maintenant créer la description du circuit.

Cettedescription sera définie comme l'entité top-level du projet.

Vous utiliserez principalement des descriptions sous forme deschéma, de machine d'état ou des descriptionscomportementales en VHDL.

Pour créer une nouvelle description, suivez ces étapes

1. Choisissez New dans le menu File.

2. Sélectionnez Block Diagram/Schematic File de

l'onglet Device Design Files pour une descriptionsous forme de schéma.

OU

2. Sélectionnez State Machine File de l'onglet Device

Design Files pour une description sous forme de machine d'état.

OU

2. Sélectionnez VHDL File de l'onglet Device Design

Files pour une description en VHDL.

3. Cliquez sur OK. Une nouvelle fenêtre d'édition de s'ouvre.

4. Choisissez Save As du menu File.

5. Sélectionner le répertoire où vous désirez sauvegarder le fichier.

6. Dans le champ File name, entrez le nom du bloc que vous voulez créer.

7. Assurez-vous que la case Add file to current project est cochée.

8. Cliquez sur Save pour sauvegarder le fichier et l'ajouter à votre projet.

Description sous forme de schéma :

Ce type de description est utilisé :

si la synthèse a été au préalable entièrement faite « à la main » et que l'on dispose d'un logigramme du système ou de la fonction à synthétiser

si l'on souhaite connecter entre elles plusieurs fonctions déjà synthétiser au préalable.

III. COMPILATION

Le Compilateur de Quartus II est constitué d'une série de modules qui vérifient s'il n'y a pas d'erreurs dans le design, synthétisent la logique et génèrent les fichiers pour la simulation, l'analyse temporelle et la programmation du composant.

1. Compilez le projet en choisissant la commande Start compilation du menu Processing ou en cliquant sur .

2. Si vous recevez un message indiquant que la compilation est réussie et qu'il n'y a pas eu d'erreurs, cliquez sur OK sinon corriger votre projet et recommencer une compilation.

Remarque :

Durant la compilation, la fenêtre Compilation Report apparaît automatiquement. Le rapport procure des informations détaillées sur la compilation. La section Summary permet par exemple d'obtenir de l'information sur :

• Le statut final de la compilation

• Les requis temporels

• Le nom des entités compilées

• Le nombre total de cellules logiques, broches, mémoires et de PLL utilisés

IV. ASSIGNATION DES BROCHES

L'assignation des signaux aux broches d'entrées/sortie du FPGA peut se faire après la

simulation mais il est à noter que cette assignation pourrait avoir une influence sur le comportement du circuit.

Suivez ces étapes afin d'assigner les signaux aux broches de votre FPGA :

1. Après avoir compilé le projet avec succès, cliquer sur Pin dans le menu Assignments.

2. Dans la section du bas, associez des broches du FPGA aux entrées/sorties de votre projet.

V. SIMULATION

La simulation vous permet de vous assurer que votre projet répond correctement à vos

attentes. La simulation s'effectuera en plusieurs étapes :

création d'un fichier Vector Waveform File (.vwf),

configuration du simulateur

exécution de la simulation

analyse du résultat.

Vous pouvez créer un fichier .vwf, qui sera utilisé comme stimuli pour le simulateur, avec l'éditeur Waveform de Quartus II. Afin de créer un fichier .vwf, suivez ces étapes :

1. Choisissez New dans le menu File.

2. Cliquez sur l'onglet Verification/Debugging Files et sélectionnez University Program VWF.

3. Cliquez sur OK. L'éditeur Waveform s'ouvre.

4. Pour modifier le temps de fin de la simulation, choisissez End Time du menu Edit.

5. Pour ajouter des entrées et des sorties à votre fichier de simulation, faire un clic droit dans la colonne name puis sélectionnez Insert Node or Bus.

6. Si vous signaux n'apparaissent pas, cliquer sur Node Finder, sélectionnez Pins : all dans la liste Filter.

7. Cliquez sur Start afin de lancer la recherche.

8. Dans la liste de signaux trouvés, sélectionnez les signaux et déplacer-les dans la colonne Name de la fenêtre VWF.

9. Fermer l'outil Node Finder.

10. Affectez à chaque entrée des stimuli en sélectionnant un intervalle temporel puis en lui attribuant une valeur.

11. Sauvegardez votre fichier.

Il faut savoir que le logiciel Quartus II autorise deux types de simulation : la simulation temporelle et la simulation fonctionnelle.

Lors d'une simulation fonctionnelle, le simulateur simule le comportement du design sans tenir compte des contraintes temporelles. Pour la simulation temporelle, le simulateur utilise les informations temporelles estimées ou réelles sur circuit. Ce type de simulation vous permet de vérifier les paramètres temporels tels que les temps de maintient, temps de placement et de détecter les glitchs (passage de la sortie à un niveau non désiré lorsque plusieurs entrées changent d'état en même temps), en plus du fonctionnement logique du circuit.

Pour effectuer la simulation, choisissez Run Functionnal Simulation ou Run Timing

Simulation dans le menu Simulation suivant le type de simulation désiré.

VI. PROGRAMMATION

Le programmeur vous permet d'utiliser les fichiers générés par le compilateur afin de

programmer et/ou de configurer les composants Altera qui sont supportés par le logiciel Quartus II.

Suivez ces étapes afin d'effectuer la programmation :

1. Cliquez sur .

2. Configurer le mode de programmation en cliquant sur hardware setup....

3. Choisissez USB-Blaster en mode JTAG.

3. Cochez la case Program/Configure.

4. Cliquez sur Start.

Bibliographie:

Bibliographie:

[1] https://fr.wikipedia.org/wiki/Modulation_de_largeur_d%27impulsion(MLI)

[2] https://www.locoduino.org/spip.php?article47 dif

[3]http://proxacutor.free.fr/glossaire.htm fpgint

[4]https://fr.wikipedia.org/wiki/Modulation_de_largeur_d%27impulsion

[5]https://fr.wikipedia.org/wiki/Circuit_logique_programmable

[6] A. Boumâaraf, M.D. Draou, S. Chikhi. Nouveau concept de la commande PWM destiné au système de pompage photovoltaïque. Revue des énergies renouvelables, vol. 5, no2, p. 139, 2002, [Online]. Available: [www.cder.dz/download/Art5-2_6.pdf

[7]A. Boumâaraf, T. Mohamadi, M.D. Draou. Optimization of the Repeated PWM Command Applied to the Variable Voltage Variable Frequency (VVVF) Converter. International Journal of Computer Applications, vol. 24, no. 2, p.01-12, Jun. 2011. http://www.ijcaonline.org/archives/volume24/ number2/2926-3869

[08]Abdelâali Boumâaraf , Tayeb Mohamadi & Nadhir Messai, ''Improving of the Generation Method of Repeated PWM Based on the Signals Combinations Applied to a PV Pumping system'' TMREES15, International Conference on Technologies and Materials for Renewable Energy, Environment and Sustainability, Beirut Lebanon, 17-22 04 2015

[09]Manuel FPGACenter 2010-2013, 12 mars 2015, : ,http://fpgacenter.com/fpga/index.php

[10] Manuel FPGACenter 2010-2013, 12 mars 2015, : ,http://fpgacenter.com/V.H.D.L./index.php

[11] [Manuel FPGACenter 2010-2013, 12 mars 2015, http://fpgacenter.com/V.H.D.L._dic/access.php,

[12]BAGHLI L., Modélisation et commande de la machine asynchrone, Cours, IUFM de LORAINE, Université Henri POINCARE-Nancy 1, 2005.

[13]BLASCHKE, F.; The principal of field orientation as applied to the new transvector closed-loop ontrol system for rotating-field machines, Siemens Review, XXXIX, n°5, pp. 217-220, 1972.

[14]BAGHLI L., Contribution à la commande de la machine asynchrone, utilisation de la logique floue, des réseaux de neurones et des algorithmes

génériques, Thèse de Doctorat, Université Henri POINCARE, Janvier 1999, [En ligne]. Disponible http://www.baghli.com/dl/these_baghli.pdf

[15] Stefan Laurentiu CAPITANEANU, Optimisation de la fonction MLI d'un onduleur de tension deux-niveaux, Thèse présentée pour obtenir le titre de Docteur de l'Institut National Polytechnique de Toulouse, Spécialité : Génie Electrique, Soutenance : le 28 novembre 2002.

[16] GOURMAT Laïd,Simulation de la commande d'un moteur asynchrone

par la technique R.P.W.M. sous V.H.D.L. Thèsepour obtenir Master 2015Soutenu le ..09/06/2015

[17] cour friha mastar 2 automatique et informatique industrial 2022

Mots Clés :

FPGA,V.H.D.L, Commande, Onduleur, M.L.I., M.L.I. Répétée.