République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud MAMMERI, Tizi-Ouzou

Faculté de Génie Electrique et d'Informatique
Département d'Automatique

MEMOIRE DE FIN D'ETUDES

En vue de l'obtention du diplôme

MASTER ACADEMIQUE EN AUTOMATIQUE
OPTION : COMMANDE DES SYSTEMES

Thème

Commande d'un robot en position à base d'une carte FPGA.

Proposé par : M[r].MELLAH Rabah

Présenté par : AFETTOUCHE Malik

Dirigé par : M[r].MELLAH Rabah

Soutenu le : / /2013

Promotion 2013

Remerciements

En premier lieu je remercie le bon DIEU le tout puissant et le créateur de m'avoir facilité le chemin et qui ma donné durant toutes ces années la santé, le courage à la fois en moi même pour arriver à ce jour.

Je tiens à remercier vivement mon promoteur M^{r} MELLAH RABAH pour ses conseils et ses suggestions qui m'ont permis de mener à bonne fin mon bon travail.

Je tiens aussi à remercier tout ce qui m'a apporté leur assistance et leur encouragement.

Dédicaces

Je dédie ce modeste travail à :

A mes chers parents.

A mes chères sœurs.

A mes amis du village.

A mes amis du l'université.

A toute ma famille.

A toute la promotion Automatique 2013.

Résumé :

Les travaux de recherche présentés dans ce mémoire ont porté sur la réalisation d'une commande d'un robot a trois axes a base d'une carte FPGA (Field Programmable Gate Array) et son implantation. Dans ce contexte, ce travail est scindé en 3 chapitres principaux :

- Le premier chapitre est consacrée au traitement des circuits logiques programmables selon leur ordre chronologique d'événements jusqu'à l'invention des FPGA et nous nous intéressons particulièrement aux FPGA de la compagnie XILINX de la famille VIRTEX-5.
- Le deuxième chapitre, traite l'outil indispensable pour la programmation d'un FPGA qui est le langage VHDL. Ce langage est présenté en détail et on finira ce chapitre par donner un aperçu sur le logiciel de développement ISE.
- Le troisième chapitre a débuté par une description du robot ensuite des généralités sur les moteurs pas à pas puis on a fait un petit aperçu sur notre application en citant les différents procédés utilisé pour la conception matérielle et logiciel et en finira par les résultat de simulation.

Et enfin, nous terminons notre travail par une conclusion générale.

Liste des figures :

NOMENCLATURE

Sigles utilisés

PLD : circuits logiques programmables (programmable logic device).

EPLD : Erasable Programmable Logic Device.

CPLD : Complex programmable logic device.

FPGA : Field programmable Gate Array ou encore réseau de cellules logiques programmables.

CLB : Blocs logiques configurables.

IOB : Blocs d'entrées/sorties programmables.

LUT : Look-Up Table.

VHDL : Very High Speed Integrated Circuit Hadware Description Language.

Sommaire

Sommaire

Chapitre II : utilisation du langage VHD

CHAPITRE III : Application d'un programme VHDL à la commande en position d'un robot à 3 axes.

Introduction générale

Introduction générale :

La robotique est une science importante qui ne cesse d'évoluer. Elle est le domaine ou se réunis plusieurs spécialités en vue de réaliser une entité pourvue de possibilités d'intégrer avec son environnement, par une intervention minimale d'un opérateur humain ou de l'éliminer carrément.

C'est au siècle dernier que la robotique à amorcé des projets de recherche sur la robotique industriel, car dans les années 80 les bras manipulateurs industriels sont très semblables à ceux d'aujourd'hui appart que ceux d'aujourd'hui sont plus développés et ils apportent plus de précision et rapidité…

Les progrès de la technologie robotique son apportés par les développements scientifiques, spécifique de l'électronique et de l'informatique et aussi d'automatique, mathématique, mécanique et matériaux.

Les robots actuels sont très répondus dans l'industrie en particulier dans la construction automobile et chez la plupart des fabricants d'ordinateur. Ils sont capables d'effectuer des taches répétitives avec une vitesse et précision importante, ils sont même utilisés dans des chaines industrielles de fabrication et de montage et surtout on les emploie dans des milieux difficilement supportés par l'homme (condition extrêmes de température ou de pression, radioactivité élevée, etc.…). Car ces robot son sophistiqués et aussi dotés d'une intelligence artificielle.

Dans notre travail nous avons proposé une commande d'un robot constitué de trois moteurs pas à pas. Il se déplace dans les trois axes X, Y, Z, cela à base d'une carte FPGA(Field Programmable Gate Arrayou réseau de cellules logiques programmables) de type VITREX-5 que nous avons programmé par le langage VHDL(Very High Speed Integrated Circuits Hardware Description Langage)..

On a scindé ce mémoire en trois chapitres :

Le premier chapitre est consacrée au traitement des circuits logiques programmables selon leur ordre chronologique d'événements jusqu'à l'invention des FPGA et nous nous intéressons particulièrement aux FPGA de la compagnie XILINX de la famille VIRTEX-5.

Le deuxième chapitre, traite l'outil indispensable pour la programmation d'un FPGA qui est le langage VHDL. Ce langage est présenté en détail et on finira ce chapitre par donner un aperçu sur le logiciel de développement ISE.

Le troisième chapitre a débuté par une description du robot ensuite des généralités sur les moteurs pas à pas puis on a fait un petit aperçu sur notre application en citant les différents procédés utilisé pour la conception matérielle et logiciel et en finira par les résultat de simulation.

Et enfin, nous terminons notre travail par une conclusion générale.

Chapitre 1

Les circuits logiques programmables

1. Introduction : [5][4][3]

Durant ces dernières années l'évolution de l'industrie a connu une évolution gigantesque grâce à la concurrence entre les fabricants.

Actuellement, la logique programmable est la plus utilisés par les microcontrôleurs (µc) et microprocesseurs(µp) mais l'inconvénient de ces circuit et c'est qu'on peut seulement les programmer selon le programme existant dans une mémoire, l'architecture interne est celle proposé par le fabriquant, tout comme les entrées/sorties, et aussi sans oublie le nombre de ces circuit nécessaire qui peut être important, ce qui avait pour conséquence un prix important, une mise en œuvre complexe et un circuit imprimé de taille.

Pour diminuer le cout de fabrication, de développement et de maintenance Les PLD ont subi une évolution technologique au fil du temps depuis l'apparition de premier PAL (programmable array logique réseau logique programmable) jusque à l'aboutissement des premiers circuits intégrés reconfigurables de type FPGA (Field Programmable Gate Arrayou réseau de cellules logiques programmables) par la société XILINX en 1985.

Ces circuits FPGAs sont capables de réaliser plusieurs fonctions logiques complexes avec la souplesse et la flexibilité apportée par la logique programmable est ça dans un seul circuit, et ils sont les circuits logiques programmables les plus performants qui existent en ce moment.

2. Définition des PLDs : [5]

Un circuit programmable est un assemblage d'opérateurs combinatoires (les opérateurs combinatoires génériques qui interviennent dans les circuits programmables proviennent soit des mémoires (réseaux logiques) soit des fonctions standard (multiplexeurs et OU exclusif)) et de bascules dans lequel la fonction réalisée n'est pas fixée lors de la fabrication. Il contient potentiellement la possibilité de réaliser toute une classe de fonctions, plus ou moins large suivant son architecture. La programmation du circuit consiste à définir une fonction parmi toutes celles qui sont potentiellement réalisables. Comme dans toute réalisation en logique câblée, une fonction logique est définie par les interconnexions entre des opérateurs combinatoires et des bascules, et par les équations des opérateurs combinatoires. Ce qui est programmable dans un circuit concerne donc les interconnexions et les opérateurs.

3. La structure de base d'un PLD : [7]

La structure générale d'un circuit PLD est représentée sur la figure 1.1 :

- Un bloc d'entrée qui permet de fournir au bloc combinatoire l'état de chaque entrée et de son complément.
- Un ensemble d'opérateurs « ET » sur lesquels viennent se connecter les variables d'entrée et leurs compléments.
- Un ensemble d'opérateurs « OU » sur lesquels les sorties des opérateurs « ET » sont connectées.
- Un bloc de sortie.
- Un bloc d'entrée-sortie, qui comporte un porte 3 état et une broche d'entrée sortie.

Le bloc combinatoire programmable est formé de matrices « ET » et de matrices « OU » car toute fonction logique combinatoire peut être écrite comme somme de produit, d'interconnexions de ces matrices qui doivent être programmables par un logiciel, utilisant des fusibles qui seront grillés lors de la programmation.

Le bloc de sortie est souvent appelé marco-cellule que l'on nomme OLMC (abréviation anglaise de Output Logic Macro Cell signifiant macro-cellule logique de sortie). Cette macro-cellule comporte :

- Une porte OU exclusif, une bascule D.
- Des multiplexeurs qui permettent de définir différentes configuration et un dispositif de bouclage sur la matrice ET.
- Des fusibles de configuration (dans les FPGAs on utilise plutôt des cellules de commande des points de connexion).

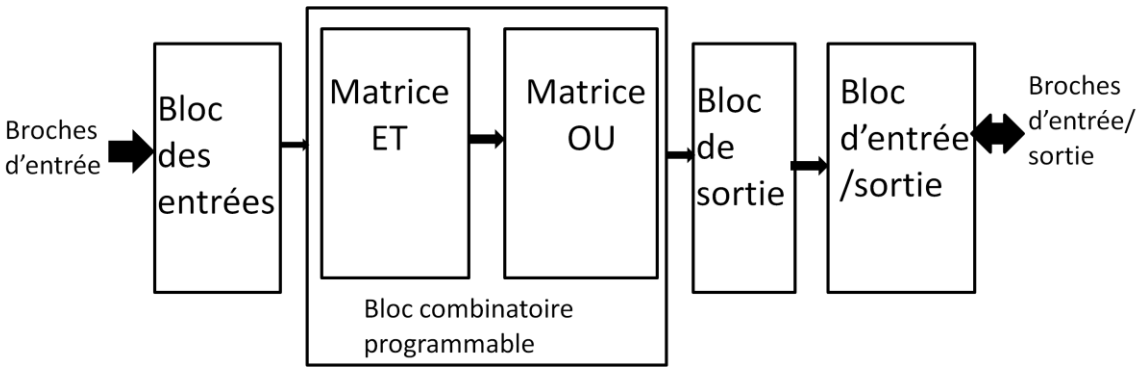


Figure1.1 : La structure de base d'un PLD

4. Les différentes familles de PLD : [6]

La classification des PLDs peut se révéler délicate et difficile, les différences de technologie se doublent de différences d'architectures. La classification suivante n'a que pour objectif de mettre en lumière de grands points de repère. Néanmoins, on peut les classer suivant leurs structures internes à savoir : le nombre d'entrées, de sorties, de connexions programmables et le niveau d'intégration. Le tableau ci-dessous représente certains de ces familles :

Type	Nombre de Porte intégré	Matrice ET	Matrice OU	Effaçable
PAL	10 à 100	Programmable	Fixe	Non
GAL	10 à 100	Programmable	Fixe	Electriquement
EPLD	100 à 3000	Programmable	Fixe	U-V
FPLA	2000 à 3000	Programmable	programmable	Eclectiquement
FPGA	Plus de 50.000	Programmable	programmable	Électriquement

Tableau 1.1 : Tableau récapitulatif des différentes familles des PLD

4.1. Les PALs : [7]

Les PALs (Programmable Array logique ou réseau logique programmable) ce sont les premiers circuits intégrés programmables pour réaliser des fonctions logiques et ils ont fait un grand succès sur le marché. Ces circuits sont des composons relativement simples le fait qu'ils dérivent des PROM (Programmable Real Only Mémoire)qui ont une mémoire morte à lecture seule et programmable une fois. Ce type de circuit a été développé par le constructeur AMD à base des matrices ‹‹ET››programmables et des matrices ‹‹OU›› fixes.

La programmation de ces PAL s'effectue par la destruction de ces fusibles et une fois programmé on ne peut plus les effaces, lors de l'achat ces fusibles sont vierges.

On distingue deux sous familles :

4.1.1. Les PALs combinatoires (simple): [3]

Ce type de PAL possède une architecture interne la plus simple car il possède des portes logiques. Comme tout les autres PALS ils contiennent des pins dédiées uniquement en entré et d'autre en entré/sortie associe avec un buffer.

4.1.2. **Les PALs séquentielles :**

Parmes les types de PAL séquentielle on distingue trois types qui sont :

a. Les PALs à registre (FPLD): [3]

Ce type de PAL dispose d'une bascule D en sortie car tout signal sortant sera passé par cette bascule. Pour cette raisons que certain boitier on des sortie combinatoires et des sortie séquentielles (à registre). On notera que le signal d'horloge est le même pour toutes les bascules de ce composant.

b. Les PALs asynchrone registre : [3]

Ce type de circuit est presque le même avec les PALs à registre mais la seule différence réside dans le signal l'horloge de ces bascules.

Les bascules de circuit n'ont pas le même signal d'horloge car chaque bascule à sont propre signal d'horloge issu d'un terme produit de la matrice.

Ces circuits, contrairement aux PALs à registres, ils ont muni d'un multiplexeur qui permet de shunter le registre. Les deux entrées AP (Preset Asynchrone) et AR (Reset Asynchrone) sont simultanément utilisées pour pilôter ce multiplexeur. Ces deux signaux issus d'un terme produit sont actifs à 1 ; et on notera la présence d'un XOR se comportant comme un inverseur programmable, et son signal de commande est une valeur statique.

c. Les PALs versatiles (VPAL): [3]

Ce type de structure représente les PALs les plus évolués, la structure de ça sortie est dit versatile et configurable. Le mode de fonctionnement est obtenus en utilisant deux multiplexeurs qui utilisent comme signal de sélection les points de programmation S0 et S1. Le tableau ci-dessous décrit les quartes différentes possibilités :

S1	S0	Configuration de la sortie	Retour
0	0	Registre, actif à 0	Registre
0	1	Registre, actif à 1	Registre
1	0	Combinatoire, actif à 0	Entrée/sortie
1	1	Combinatoire, actif à 1	Entrée/sortie

Cette structure de circuit a un signal d'horloge qui est commun pour l'ensemble du composant et il est issu d'une broche spécifique. Ce signal d'horloge ne peut pas être inversé et son front actif est toujours le front montant. Ce PAL dispose d'un signal AR (Asynchrone Reset) qui initialise la bascule à 0 de manière asynchrone et d'un signal SP (Synchronours Preset) qui permet de charger la bascule à 1 sur le front montant de l'horloge.

4.2. Les GALs : [1]

L'apparition des GALs a éliminé l'inconvénient majeur des 1er PAL qui sont programmables une seule fois, leurs développements par la société LATTICE à donné naissance aux GAL (Réseau logique générique) qui sont constitué à base des transistors CMOS et ils sont effaçables et programmable électriquement. Ils sont équipés des macro-cellules qui sont programmable et qui permet aussi de replacer n'importe quel PAL.

Etant donné que ces circuits sont constitués de transistors CMOS, leur consommation est beaucoup plus faible que les PAL à fusibles bipolaires . Ils sont aussi plus souples d'utilisation et ils sont dotés d'un bit de sécurité qui peut être activé lors de la programmation pour empêchant la lecture du contenu du circuit, et Ce bit se remis à zéro seulement en effaçant complètement le GAL.

4.3.Les EPLDs : [1]

Les EPLDs c est des circuits logiques programmables éclectiquement et effaçable, soit par ultraviolets ou électrique, et ils ont été inventés par la firme américaine ALTERA qui les a introduits pour la première fois en 1984. Ces circuits, comme les PAL et les GAL, font appel à la notion de macro-cellule qui permet, par programmation, de réaliser de nombreuses fonctions logiques de base. Ils sont caractérisés par leur densité d'intégration qui est nettement supérieure à celle offerte par les PALs et une vitesse de fonctionnement égale, ou du moins comparable, à celle des PALs bipolaires. En plus de ces deux caractéristiques, les EPLDs possèdent un atout majeur qui réside dans la possibilité d'effacement .

Depuis leurs mises sur le marché en 1984, ALTERA n'a cessé de travailler et de développer son produit et propose maintenant une gamme de produits très diversifiés et offrant, en outre, la possibilité d'effacement électrique. On trouve en effet :

Les EPLDs à usage général ou EPLD classique de la série EP qui ont pour principale fonction le remplacement de la logique conçue avec un ou plusieurs PAL ou bien encore avec plusieurs dizaines de boîtiers TTL classiques. Ces circuits existent en différentes versions avec boîtiers disposant de 20 à 65 pattes. Leurs délais de propagation typiques sont de l'ordre de 12ns et des versions pouvant fonctionner jusqu'à 100 MHZ. Ces circuits ne sont effaçables qu'aux ultraviolets.

Les EPLDs de la série MAX 5000 qui possèdent les mêmes fonctions que leurs homologues de la série EP mais avec des densités d'intégration plus élevées. En effet, ils permettent d'intégrer des réalisations utilisant de 20 à 25 PAL classiques ou bien encore des centaines de boîtiers TTL ordinaires et une architecture d'interconnexion différente qui est MAX (Multiple Array Matric) ; qui ne sont pas effaçables qu'aux ultraviolets.

Les EPLDs de la série MAX 7000 qui sont plus récents et plus denses que les MAX 5000. Ces circuits généralisent la méthode d'interconnexion originale employée dans les Max 5000 et apportent, en plus, la possibilité d'effacement électrique.

Les EPLDs de la série MAX 9000 qui sont encore plus dense et plus riche en terme de possibilités d'interconnexion que les précédents avec un effacement électrique et une programmation en circuit sous une tension unique de 5 volts.

4.4. Les CPLD : [1]

Ce sont des circuits composés de plusieurs PALs élémentaires reliés entre eux par une zone d'interconnexion (matrice d'interconnexion). Sa physionomie est généralement très structurée. Un certain nombre de macro- cellules de base sont regroupées pour former des blocs logiques. Grâce à leurs structures, ils peuvent atteindre des vitesses de fonctionnement élevées (plusieurs centaines de MHz). Ces circuits ont une capacité en nombre de portes et en possibilités de configuration très supérieure à celle des PALs.

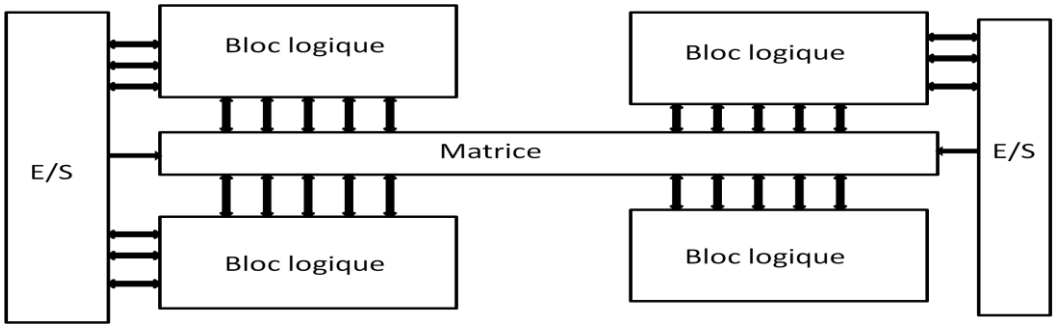


Figure 1.2 : physionomie d'un CPLD

4.5. Les FPGAs: [15]

Les premiers circuits intégrés reconfigurables de type FPGA (Field Programmable Gate Array ou réseau de cellules logiques programmables) ont été commercialisés par la société XILINX en 1984.

Un FPGA est un circuit logique reprogrammable. À l'aide de blocs logiques préconstruits et de ressources de routage programmables, c'est un circuit configurable afin de mettre en œuvre des fonctionnalités matérielles personnalisées, sans avoir jamais besoin d'utiliser une maquette ou un fer à souder.

5. Etude des FPGAs :

5.1.Généralité : [3]

Les premiers circuits intégrés reconfigurables de type FPGA (Field Programmable Gate Array) ont été commercialisés par la société XILINX en 1985. Depuis cette date d'autres fabriquant sont apparus sur un marché comme ALTERA, ACTEL, Texas Instrument …qui n'a pas cessé de croître. Au fur et à mesure que la complexité des FPGAs s'est développée, leurs possibilités d'emploi se sont accrues, jusqu'à concurrencer sérieusement les circuits spécifiques pour de petits volumes de production (jusqu'à quelques milliers de composants). C'est pourquoi on envisage de plus en plus de développer l'utilisation des FPGA dans le cadre des applications aéronautiques et spatiales.

Ils apportent une grande facilité d'emploi et permettent une réduction du coût de développement des électroniques embarquées. Les FPGAs reconfigurables à base de SRAM (Static Random Access Memory) apportent en outre la possibilité de faire évoluer in situ la fonctionnalité de ces circuits, ce qui ouvre des perspectives de maintenance à distance que n'autorisent pas les circuits spécifiques.

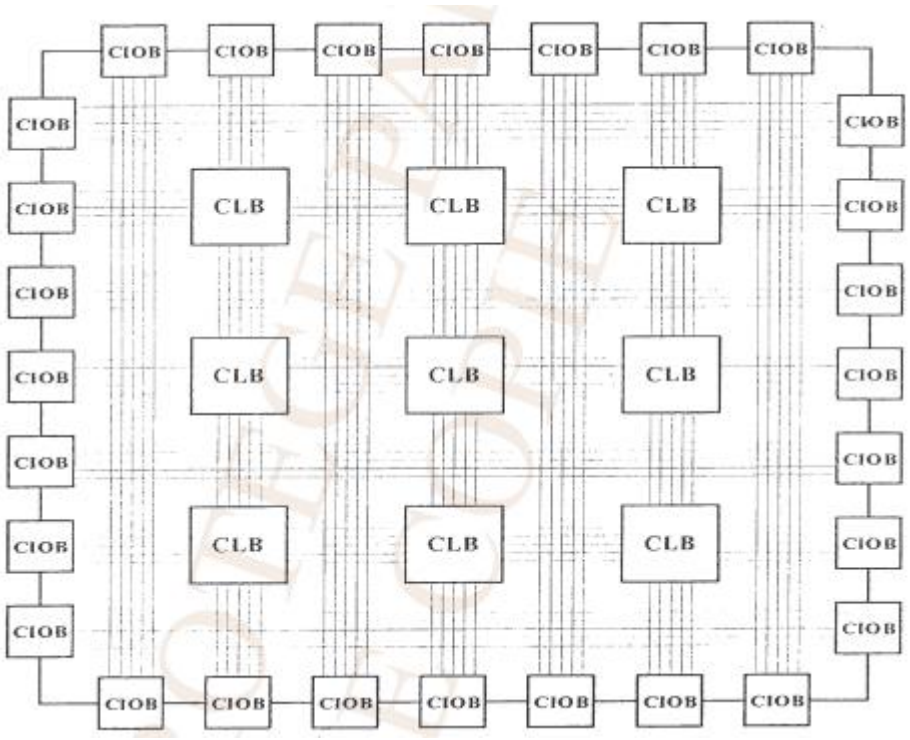


Figure 1.3 : Architecture interne des FPGAs

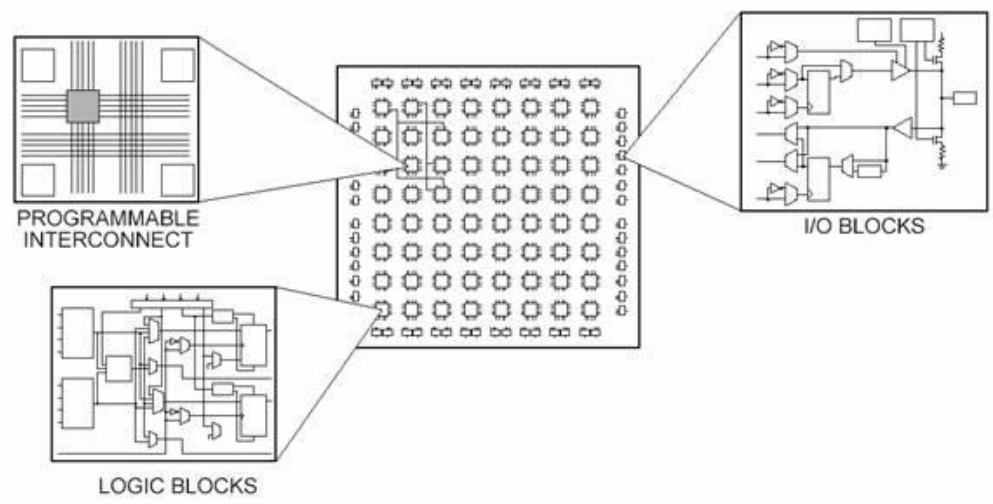


Figure 1.4 Les différents secteurs d'un FPGA

5.2. Les différents types de FPGA : [3]

Les FPGA ont une architecture qui diverse et on distingue deux types de FPGA qui sont :

-Les FPGAs de type SRAM, appelé LCA chez XILINX, FLEX chez ALTERA.

-Les FPGAs à anti fusibles proposé entre autre, par Texas Instrument et ACTEL.

Ces deux technologies différent essentiellement dans la façon dont leurs interconnections sont réalisées.

5.2.1. Les FPGAs du type anti fusible : [1] [2]

Ces circuits sont commercialisé pour la première fois en 1990 et ils sont différent des autres FPGAs car ceux là sont pas effaçables. Ils sont proposés pour l'essentiel par Texas Instrument.

Ces circuits ne sont pas reprogrammables, le fait qu'il utilise des connexions de type ROM, dans les modifications sont irréversibles. D'où le nom d'anti-fusible puisque dans l'état initiale d'un fusible (couche isolante) est présent et il n'y a pas de contacte. Pour établir ce contact, il faut détruire le fusible ce qui est contraire au fonctionnement habituel d'un fusible.

L'architecture globale de ces FPGAs sont généralement beaucoup moins complexe que celle des FPGAs de type SRAM, et on constate aussi que sont architecture est analogue à celle des LCA, avec un certain nombre de blocs d'entrées/sorties répartis tout autour de la puce. Des blocs logiques placés en matrice au centre de celle ci et des lignes d'interconnexions, mais la similitude s'arrête là. En effet, alors que les LCAs faisaient appel à des cellules logiques de base relativement complexes, les CLB, des FPGAs à anti-fusibles utilisant des cellules très simples. Leurs avantages par rapport aux autres circuits c'est qu'ils peuvent être nombreux sur la puce, ce qui accroit la souplesse d'interconnexions internes et facilite le routage. Comme référence, la famille TCP 10XX et TCP 12XX de TEXAS INSTRUMENT.

5.2.2. Les FPGA du type SRAM :[3] [2]

La structure de base d'un FPGA de type SRAM est complexe. Un point de connexion entre les différentes cellules est un ensemble de transistors MOS de commutation commandés par des cellules de mémoire vive (RAM).

Ces circuits FPGA type SRAM sont constitués de blocs logiques élémentaires et de réseaux d'interconnexions pouvant être configurés par l'utilisateur. Celui-ci a la possibilité d'implanter une fonctionnalité donnée dans ces circuits sans avoir à se préoccuper des différentes étapes de sa fabrication. La complexité des circuits implantés dans les FPGAs leur permet aujourd'hui de concurrencer les circuits spécifiques pour de petits volumes de production (jusqu'à quelques milliers d'unités). Ils sont donc très bien adaptés à une

utilisation pour des applications de type spatial. Le recours aux ASIC (Application Specific Integrated Circuits) impose en effet des temps de développement et de fabrication de l'ordre de plusieurs mois et ce pour un coût élevé. Les FPGAs autorisent la réalisation de circuits ayant une complexité équivalente en des temps et pour des coûts plus réduits. En outre, de nombreux FPGA permettent une reconfiguration à volonté et donc une évolution des circuits au cours de leur vie. Ce qui nous amène à nous intéresser de plus près à leur architecture.

6. Etude de la structure générale d'un circuit FPGA :

6.1.Les cellules logiques de base : [1][2]

Généralement en distingue deux types de cellule de base qui sont :

6.1.1. Les cellules d'entrées-sorties :

Ils constituent l'interface entre les branches de sortie du circuit et les CLB. Ils sont présents sur toute la périphérie du circuit FPGA car ces cellules sont des intermédiaires par lesquelles les données transitent depuis les blocs logiques internes jusqu'aux ressources externes et vice versa. Chaque bloc IOB contrôle une broche du composant et peut être défini en entrée, en sortie, en entrée/sortie ou être inutilisé qui peut prendre les états suivant 0, 1 ou haute impédance.

Le rôle principal des interfaces d'entrés/sorties est de transmettre et de recevoir des données.

Néanmoins l'interface d'entrée/sortie peut être dotée d'options telles que des registres, impédances et buffers.

Chaque fabricant a sa propre appellation pour désigner l'interface d'entrées/sorties mais la fonction reste toujours la même.

Altera, les nomme IOE Input Output Element. L'IOE remplie toujours son rôle d'interface d'entrés/sorties, elle dispose d'une résistance de rappel pull-up et un temporisateur du signal.

Chez XILINX, les interfaces d'entrées/sorties sont nommées IOB pour Input Output Blocks. L'IOB est constitué de registres, de diviseurs de tension s, des résistances de rappel pull up et autres ressources spécifiques.

6.1.2. Les macro-cellules :

Ces cellules logique sont appelé aussi par :

-Soit CLB (configurable logique bloc), c'est la dénomination adoptée par XILINX.

-Soit LC (cellule logique), c'est le nom choisi par CYPRESS.

-Soit LE (élément logique), c'est l'appellation d'ALTERA.

Ces macro-cellules sont plus nombreuses, et il n'y a pas de matrice ET et OU .la macro-cellule est constitué d'une partie combinatoire et une partie séquentielle.

- La partie combinatoire sur laquelle en peut réaliser les fonctions de complexité moyenne car les constructeurs ont proposé pour chacun une ou plusieurs solutions 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 fonction à l'aide d'un multiplieur.
  La synthèse de fonction combinatoire à l'aide de mémoire vive. Dans ce dernier cas, on dit aussi réalisation de fonction logique par LUT (look-up ou table d'observation).
- La partie séquentielle à une ou deux bascules généralement de type D. il est rare de trouver des macro-cellules uniques pourvues de la partie combinatoire.

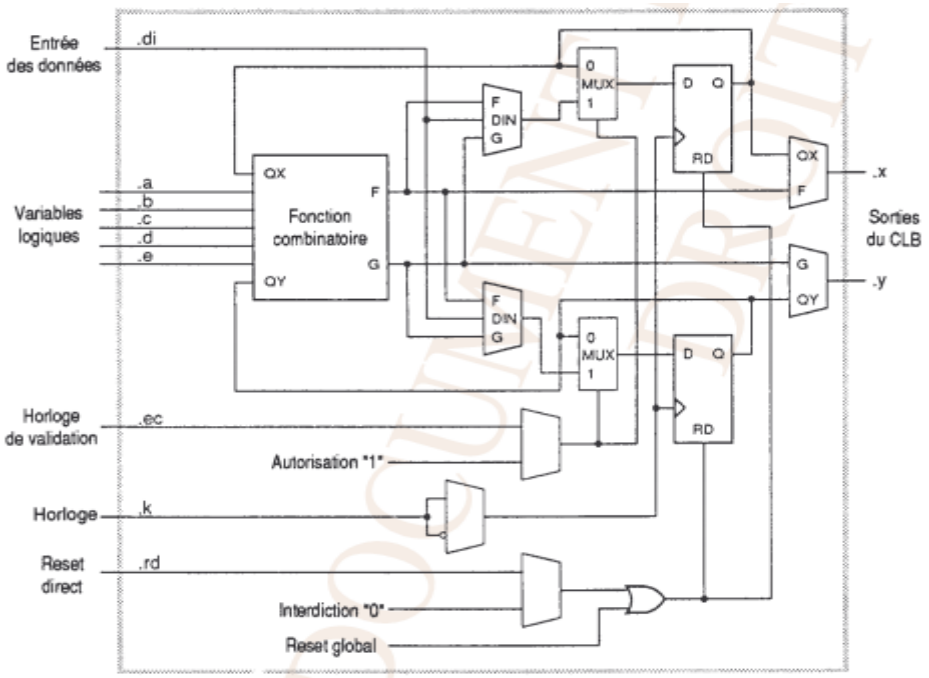


Figure1.5 : structure détaillée d'un CLB

6.2. Réseau d'interconnexion :

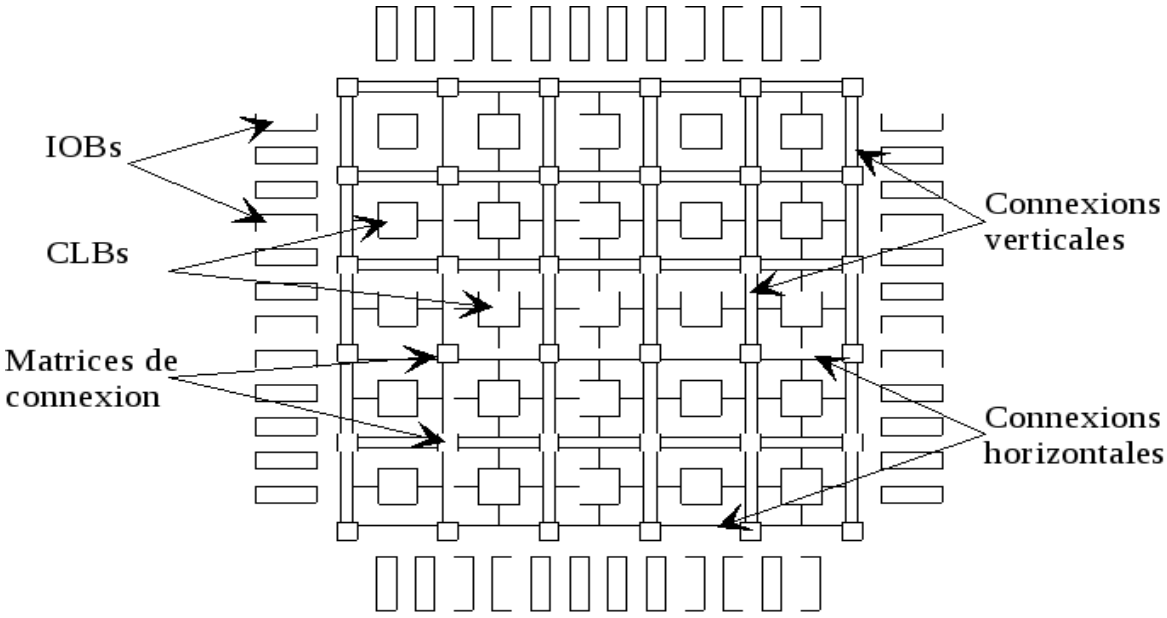
Ce réseau d'interconnexion permet de connecté une CLB avec une autre CLB ou avec une cellule d'entrée/sortie, et pour cella il existe un ensemble de lignes horizontales et verticales et un ensemble de points de connexion. On distingue plusieurs types de ligue qui sont définies par leur longueur relative et qui sont :

-Les interconnexions ou lignes segmentées à usage générale, de longueur la plus courte.

-les lignes directes ou interconnexions directe, de longueur double des lignes courtes.

-les lignes longues.

Chaque CLB est entourée de ces lignes et des points de connexion et tout ça est détaillé sur les figures suivantes :



F IGURE 1.6 : Concept architectural de base des FPGAs

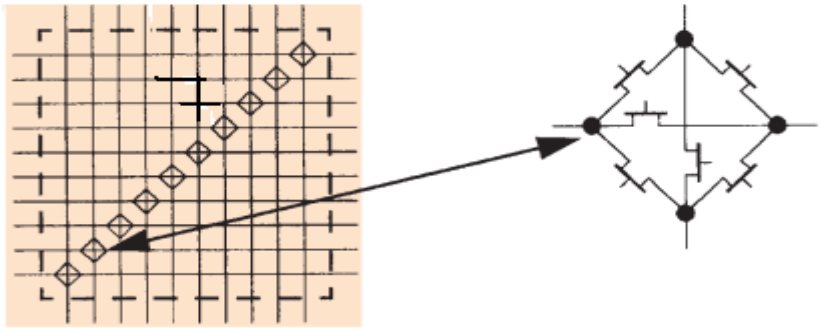


Figure 1.7: Les points d'interconnexion

6.2.1. Les interconnexions à usage générale :

Sont des verticaux et horizontaux qui encadre chaque CLB et qui peuvent être reliés entre eux par une matrice de commutation car sont rôle est de raccorder les segments entre eux selon diverse configuration.il assure aussi la commutation des signaux d'une voie sur l'autre.

6.2.2. Les interconnexions directes :

Ces interconnexions permettent d'établir des chemins entre les CLBs adjacents et les cellules entrées/sorties avec un maximum d'efficacité en terme de vitesse et d'occupation de circuit.

6.2.3. Les longues lignes :

Sont des longues lignes verticales et horizontales qui n'utilisent pas de matrice de commutation. Elles parcourent toutes la longueur et la largeur du circuit. Elles permettent aussi de transporté les signaux qui parcourt un long trajet. Elles égalisent les délais entre les signaux de façon à permettre un décalage minimum entre deux points distants de la ligne. Ces lignes conviennent pour transporter les signaux d'horloge.

6.3. Les éléments de mémorisation : [8]

Actuellement, Les FPGAs sont utilisés pour des applications plus importantes qui demandent souvent des capacités de stockage (par exemple les applications du traitement vidéo). La nécessité d'intégrer des blocs de mémoire directement dans l'architecture des FPGAs est vite devenue capitale. De cette façon, les temps d'accès à la mémoire sont diminués puisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit.

6.4. Les éléments de contrôle et d'achèvement des horloges : [8]

L'horloge est un élément essentiel pour le bon fonctionnement d'un système électronique.

Les circuits FPGA sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routage spécialement adaptées au transport d'horloges sur de longues distances. Les circuits FPGAs disposent des éléments d'asservissement des horloges (des PLL ou des DLL) afin d'avoir la même horloge dans tout le circuit (synchronisation des signaux). Ces éléments permettent de créer à partir d'une horloge d'autres horloges à des fréquences multiples de la fréquence de l'horloge incidente.

6.5. Les éléments de routage : [8]

Les éléments de routages sont les composants les plus importants dans les FPGAs. En fait, ces éléments représentent la plus grosse partie du silicium consommée sur la puce du circuit. Ces ressources sont composées de segments (de longueurs différentes) qui permettent de relier entre eux les autres éléments via de s matrices de connexions. Le routage de ces ressources est un point critique du développement d'une application sur un FPGA. Ces éléments sont très importants puisqu'ils vont déterminer la vitesse et la densité logique du système. Par exemple, les matrices de routage sont physiquement réalisées grâce à des transistors de cellules SRAM, qui ont une résistance et une capacité, ce qui entraîne l'existence de constantes de temps. Et ce routage et montré dans la figure suivante :

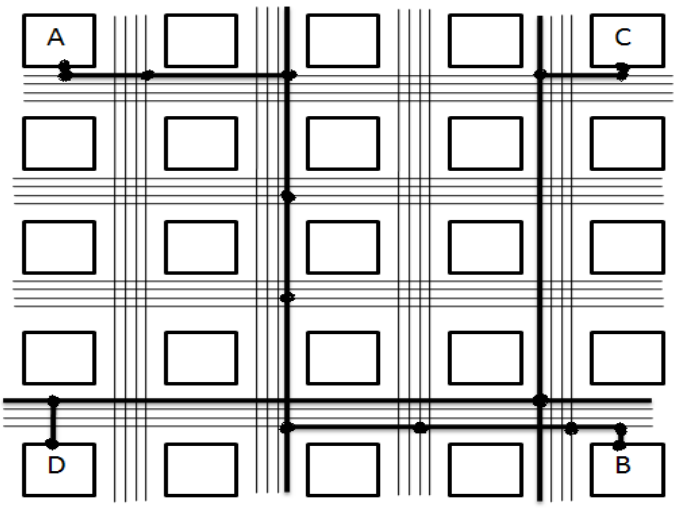


Figure 1.8: routage d'un FPGA

7. Etude des FPGAs de la famille Xilinx :[1][9]

7.1. Présentation et généralités sur les FPGAs de la famille Xilinx:

Le plus gros constructeur du marché est XILINX qui a introduit la série XC2000 (de 600 à 1500 portes) entre 1984 et 1985 avec des fréquences de fonctionnement pouvant atteindre les 420MHZ. Depuis, d'autres séries sont apparues comme les XC3000, XC4000, XC5200 et XC6200, ainsi que les XC9500 et la mise sur le marché du 1er FPGA XILINX en 1985.Puis l'apparition en 1992 du premier FPGA du constructeur ALTERA qui est le concurrent le plus important de XILINX, avec un type de circuits assez différent le FLEX 8000 (15 000 portes max).Rapidement l'exploitation de la technologie EEPROM un an plus tard en 1993 puis le lancement du VIRTEX II /XILINX (jusqu'à 10 millions de portes) en 2001 et des FPGAs de capacités supérieures à 50 millions de portes fonctionnant à des fréquences dépassant les 500 MHz en 2005.

Le principe des FPGAs de XILINX est de stocker la configuration dans une mémoire vive statique SRAM. Aujourd'hui des blocs des fonctionnalités supplémentaires dans quelques versions évoluées sont ajoutés et dédiées à des applications spécifiques. Ces fonctionnalités supplémentaires qui sont : Mémoire RAM, Petits multiplieurs, Blocs DSP, Cœurs de processeurs RISC et arbres de distribution d'horloge pour générer différent domaines d'horloge pour un bon synchronisme en cas d'FPGA a grande capacité.

Dans le cas du Virtex-II 1000, la matrice de logique programmable est de 40 × 32 = 1280 CLB, soit 5120 slices, soit enfin 10240 cellules logiques. Les LUT sont constitués de quatre slices. Les bascules dans chaque slice sont initialisées par défaut à valeur '0' et chaque bascule bénéficie de broches de contrôle. Chaque table LUT permet la conception d'une mémoire synchrone 16×1 bits. Les deux LUT d'une tranche offrent une mémoire synchrone 16×2 bits, 32×1 bits ou 16×1 bits. Une LUT fonctionne également comme un registre à décalage de seize bits.

Le tableau suivant dresse l'historique des FPGA chez Xilinx :

FPGA (SRAM) couteux	FPGA faible coût
XC2000(1984)	
XC3000(1987)	

XC4000(1991)	Spartan(1998) Spartan-XL(1999)
Virtex (1999)	Spartan-2(2000)
Virtex-E(2000)	Spartan-2E(2002)
Virtex-II(2001)	Spartan-3(2004)
Virtex-II(2003)	
Virtex-4(2005)	
Virtex-5(2006)	
Virtex-6(2009)	Spartan-6(2009)

7.2.La famille VIRTEX-5 :

La famille Virtex-5 de Xilinx est un circuit logique programmable très flexible proposé comme alternative aux ASICs. Ils sont fabriqués avec la technologie 65nm. Proposé sous en quatre sous séries qui sont:

Virtex-5 LX: logique haute performance (expédition maintenant).

Virtex-5 LXT: logique haute performance avec une connectivité série (à venir dans la seconde moitié de l'année 2006).

Virtex-5 SXT: DSP haute performance avec une connectivité série (à venir dans la seconde moitié de l'année 2006).

Virtex-5 FXT: traitement à la connectivité en série (à venir dans la première moitié de 2007) .

7.2.1. Nomenclature d'un circuit de VIRTEX-5

Voici un exemple de référence Virtex-5 et le détail de sa nomenclature

Figure 1.9: le circuit FPGA virtex-5

7.2.2. Les IOB :

Toute la famille Virtex-4 dispose de la technologie d'interface entés/sorties haute performances et peut supporter une grande variété de standards d'interfaces. Elle dispose d'un contrôle des sorties et d'un contrôle numérique d'impédance (Digitally Controlled Impedance DCI). Chaque IOB (input, output bloc) dispose de l'entré, de la sortie et de la sélection de la sortie en trois états. Le nombre maximum d'I/O utilisateur est de 600. Ces composants possèdent aussi des DSP de 32 à 192 DSP 550 MHz 25 x 18-bit MAC, jusqu'à 12 DCM (Digital Clock Manager), jusqu'à 6 PLL (Phase Lock Loop).

7.2.3. Les blocs logiques de base de la Virtex-5, Configurable Logic Block (CLB) :

Les blocs logiques programmables (CLB) constituent les principales ressources logiques pour la mise en œuvre des circuits combinatoires et synchrones. Chaque CLB est composé de six cellules logiques (Slices).

Chaque CLB est constitué de 6 slices. Chaque slice à six entrées comme le montre la figure1.10, et elle contient un générateur de fonctions, une logique de traitement de la retenue (carry logic), des portes logiques arithmétiques, un large multiplexeur et deux éléments de stockages, 256bits de RAM distribuée au lieu de 64 bits dans la Virtex-4, un registre de 168 bits au lieu de 64 bits dans la Virtex-4. Le nombre de blocs de RAM varie entre 32 et 288 blocs. Chaque bloc RAM a une taille de 36Kbits (18Kbits pour la Virtex-4) avec une largeur de bus de données maximales de 36 bits. Ces mémoires sont doubles ports (Dual Port). Et Ils offrent une densité supérieure à 12 M portes logiques équivalente. Ils peuvent atteindre une fréquence de fonctionnement de 550 MHz.

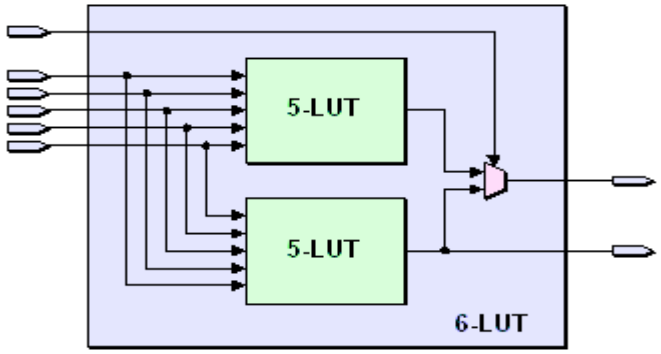


Figure 1.10: la caractéristique de la famille Virtex-5 de avec 6 entrées pour les LUTs

7.2.4. Les interconnexions :

Toute la famille Virtex-5 utilise le même schéma d'interconnexion et ils ont le même accès à la matrice globale d'interconnexion. La distribution de l'horloge est partagée de sorte à améliorer les performances des circuits à grande vitesse.

8. Conclusion :

Dans cette partie nous avons vu l'évolution des circuits logique programmables qui ont conduit à l'apparition des FPGAs. Au départ, nous avons présenté le début d'apparition des circuits programmables(PLD) puis nous avons étudié les FPGAs en détaille est spécialement la famille Xilinx, ce qui nous a permis de conclure que la technologie FPGA s'inscrit au sommet de l'évolution des composants logiques et le besoin croissant de composants plus performants, plus économiques et disponibles en grandes quantités avec un faible cout est les grands axes du progrès qui sont disponibles dans les FPGAs. Les FPGAs ouvrent de grandes perspective en matière de contrôle en temps réel. La réalisation d'un système de contrôle en temps réel nécessite aussi une bonne maîtrise des outils de modélisation et de simulation, ainsi qu'une bonne maîtrise de l'informatique temps réel lors de la phase d'implantation et l'ensemble de ces point seront traiter au cours des prochains chapitres et particulièrement détaillé et appliqué sur des moteurs pas à pas qui est la cible de notre d'application sur les FPGAs.

Chapitre 2

Utilisation du langage VHDL

1. Introduction :

L'abréviation VHDL signifier VHSIC Hardware Description Language (VHSIC: Very High Speed Integrated Circuit). Ce langage a été écrit durant les années 70**,** par le département de la défense américaine destiné à modéliser les circuits intégrés complexes. Au début, ce langage était uniquement destiné à décrire les circuits intégrés déjà conçus et devait permettre de réaliser des documentations techniques facilement interprétables par certaines personnes. Aujourd'hui, la finalité de ce langage a bien changée, puisque il est essentiellement utilisé à concevoir et modéliser les circuits, non plus dans un but de documentation, mais de simulation. Car on la étendu en lui rajoutant des extensions pour permettre la conception (synthèse) de circuits logiques programmables (P.L.D. Programmable Logic Device).

Auparavant pour décrire le fonctionnement d'un circuit électronique programmable les techniciens et les ingénieurs utilisaient des langages de difficile (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 ainsi que les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits PLD, en créant des langages plus faciles qui sont VHDL et VERILOG.

Ces langages permettent au code écrit d'être portable, de façon qu'une description écrite pour un circuit puisse être facilement utilisée pour un autre circuit. Ceci permet de matérialiser les structures électroniques d'un circuit. En effet les instructions écrites dans ces langages se traduisent par une configuration logique de portes et de bascules qui est intégrée à l'intérieur des circuits PLDs. C'est pour cela qu'on préfère parler de description VHDL ou VERILOG que de langage.

Dans ce chapitre nous nous intéresserons seulement à VHDL et aux fonctionnalités de base de celui-ci lors des phases de conception ou synthèse (c'est à dire à la conception de PLD).

2. Historique : [6]

Au début des années 80, le département de la défense américaine(DOD) désire un système de description formelle et standard des circuits dans le cadre de son programme VHSIC, car à cette époque, les fournisseurs du DOD avaient chacun son propre HDL ce qui limitait l'échange des designs. C'est pour quoi, le DOD a décidé de définir un langage de spécification. Il a ainsi mandaté des sociétés pour établir un langage. Parmi ces langages proposés, le DOD a retenu le langage VHDL qui fut ensuite normalisé par IEEE (Institute of Electical and Electronics Engineers) en vue de satisfaire les objectifs suivants :

- La spécification par la description de circuits et de systèmes.

- La simulation afin de vérifier la fonctionnalité du système.

- La conception afin de tester une fonctionnalité identique mais décrite avec des solutions d'implémentations de différents niveaux d'abstraction.

En 1993, une nouvelle normalisation par l'IEEE du VHDL a permis d'étendre le domaine d'utilisation du VHDL vers:

- La synthèse automatique de circuit à partir des descriptions.

- La vérification des contraintes temporelles.

- La preuve formelle d'équivalence de circuits.

En 2001 nouvelle révision (VHDL 2001 ou IEEE 1076 – 2001).

En 2006 nouvelle version (VHDL 2006 ou IEEE 1076 – 2006).

3. Definition du language VHDL : [7][10]

Le VHDL est un langage de description matériel destiné à représenter le comportement ainsi que l'architecture d'un système électronique indépendamment d'un fournisseur d'outils. Ainsi, techniquement, il est incontournable car c'est un langage puissant, moderne et qui permet une excellente lisibilité, une haute modularité et une meilleure productivité des descriptions. Il permet de mettre en œuvre les nouvelles méthodes de conception.

En outre, le développement du l'ensemble synthétisable du langage VHDL est de mieux en mieux défini. Cependant, la majorité des outils de synthèse sont compatibles avec cette

norme, ce qui reflète un véritable standard pour la synthèse automatique avec le langage VHDL.

4. La structure d'une description VHDL : [7]

La description VHDL est composée de 2 parties qui sont indissociables :

- L'entité (ENTITY).
- L'architecture (ARCHITECTURE).

4.1. Declaration des bibliothèques : [11][13]

Toute description VHDL utilisée pour la synthèse à besoin de bibliothèque. Tout d'abord la librairie principale qui est en générale IEEE (institut of Electrical and Electronics Engineers) . Elle contient les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques…

Ensuite il vient le mot clé ‹ use › qui fait indiqué le package de la librairie, après on écrit le nom du package. Enfin le .all qui indique l'utilisation de tout ce qui se trouve dans ce package. Elle est déclarée comme l'exemple suivant :

```
Library ieee.std_logique_1164.all ;
Use ieee.std.numeric_std.all;
Use ieee.std_logique_unsigned.all ;
```

4.2. Déclaration de l'entité :[11][12]

Tout programme en VHDL est défini par une déclaration d'entité. Cette entité permet de décrire l'interface avec l'environnement extérieur. On y retrouve donc un nom d'entité, et la liste des signaux, avec leurs caractéristiques (nom, mode, type).

La déclaration d'entité peut être partagée par plusieurs entités de conceptions, dont chacune possède une architecture différente car elle peut être vue comme une classe d'entité de conception, et présente les mêmes interfaces.

Elle permet de définir le nom de l'entité et aussi les entrées, sorties et entrées/sorties qui sont utilisées, et c'est l'instruction ‹‹port ›› qui les a définit.

La syntaxe générale de l'entité :

```
Entity Nom_de_l'entité is
        Port (Nom_entrée_1 : in type _du_signal ;
```

```
        Nom_entrée_2 : in type_du_signal ;
          …..
        Nom_sortie_1 : out type_du_signal ;
        Nom_E_S_1 : inout type _du_signal
    );

End Nom_de_l_entite ;
```

Remarque :

Il ne faut jamais maitre un point virgule après la dernière définition de signal de l'instruction port mais il faut la maitre après la fermeture de la parenthèse de cette instruction.

4.2.1. Le signal d'entré/sortie :

Un signal est défini par son :

- Nom.
- Mode: in : pour un signal en entrée.

  Out : pour un signal en sortie.

  Inout : pour un signal entrée sortie.

  Buffer : pour un signal en sortie mais utilisé en comme entré dans la description.

- Type :

  Std_logique : pour un signal.

  Std_logic_vector(x down to x0): pour un bus qui est compose de plusieurs signaux, avec x correspond au MSB et x0 correspond au LSB.

Remarque :

Les signaux std_logic peuvent prendre différentes valeur qui sont :

- '1' ou 'H' : pour le niveau haut.
- '0' ou 'L' : pour le niveau bas.
- 'Z' : pour l'état de haut impédance.

- '_' : quelconque, c.à.d. n'importe quel valeur.

4.3. Les architectures :[7][6]

L'architecture décrit la vue interne du modèle ; elle 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 est établit à travers les instructions les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire, séquentiel ou les deux (séquentiel et combinatoire) en même temps.

En VHDL la boîte noire est nommé entité (entity) car l'entité doit toujours être associée avec au moins une description de son contenu, de son implémentation qui est l'architecture. Une architecture est un ensemble de processus qui s'exécutent en parallèle.

Une architecture fait toujours référence à une entité. Elle est définie par un nom, que l'on pourra choisir pour expliciter la façon dont on code.

A la suite de la déclaration de l'architecture, on définira les déclarations préalables (signaux internes, composants).

Ensuite, viens le mot clé « begin ». A sa suite, on trouvera le code qui décrit le fonctionnement de l'architecture.

La description d'une architecture peut prendre trois formes :

- Comportementale
- Structurelle
- Mixte

La description qu'elle soit structurelle, comportementale, ou mixte se fait de la manière suivante :

Architecture description of nom_entité is

{Partie déclarative}

Begin

{Partie descriptive}

End description ;

4.3.1. Description comportementale : [7] [6]

Ce type de description décrit le fonctionnement du circuit à réaliser et le simuler en fonction des équations logiques qui relient les entrés aux sorties. Cette description est représentée dans la syntaxe suivante :

ARCHITECTURE comportementale of circuit is

-Partie déclarative.

BEGIN

-partie descriptive.

END comportementale

Ce type de description à deux moyennes de représentations qui sont :

a. Sous forme de flow de données(DATAFALOW):

Dans ce type de description comportementale DATAFALOW, on modélise le circuit par un ensemble d'équations logiques et arithmétiques, car elle consiste à décrire chaque sortie par une équation en fonction des entrées.

Example:

ARCHITECTURE XOR of circuit is

BEGIN

S1 <= IN1 XOR IN2;

S2 <= IN1 XOR IN3;

END XOR;

b. Sous forme d'instruction séquentielle:

Sous cette forme, le contenu est décrit de façon algorithmique en utilisant les structures des langages de programmation, à savoir : les déclarations séquentielles suivantes :

La structure alternative:

```
If <condition> THEN
 Séq_stat
ELSIF <condition> THEN
 Séq_stat ;
END IF ;
```

La structure d'aiguillage :

```
CASE <identificateur> IS
 WHEN choix= séq_stat ;
 WHEN others= séq_stat ;
END CASE;
```

Remarque:

Séq_stat désigne une ou plusieurs déclarations séquentielles.

4.3.2. Description structurelle :[7][6]

Dans ce type de description, les interconnexions des composants préalablement décrits sont énoncées. Cette description est la transcription directe d'un schéma. Elle se compose de trois rubriques qui sont :

- **Déclaration des signaux internes destinés à interconnecter les composants :**

Cette déclaration se fait par le mot clé <SIGNAL> accompagné par le nom et le type du signal utilisé.

- **Déclaration des composants utilisés :**

Cette étape permet de lister les composants utiles pour la construction de l'entité. Elle se fait de la manière suivante :

```
COMPONET Composant I
    Port (A: in bit, B: out bit);
END COMPONET;
```

- **Représentation de la réalisation des différentes interconnexions des composants déclarés dans la partie déclarative :**

Chaque connexion est définie de la manière suivante :

```
Comp 1 : composant port map (….) ;
```

Où :

Comp1 : représente l'affectation du mapping.

Composant : est le nom de la cellule qui entre dans la constitution de l'entité.

Port map : est le mot clé qui réalise la connexion entre les différents niveaux, signaux utilisés, internes et externes.

4.3.3. Description Mixte :[7][6]

Elle regroupe les deux descriptions décrites précédemment. À chaque entité peut être associée à une ou plusieurs architectures mais au moment de l'exécution (Simulation, synthèse…) seulement une architecture et un seule est utilisée.

Cette dernière est spécifiée par le mécanisme de package.

L'architecture dépend implicitement de l'entité à laquelle elle est associée ; tous les objets définis dans l'entité sont connus par l'architecture et ils sont vus comme des signaux qui peuvent être lus ou écrits à cet endroit, cela se fait par le biais d'instructions concurrentes énumérées au niveau du corps de l'architecture. Cette architecture comporte aussi une partie déclarative où peuvent figurer un certain nombre de déclarations (de signaux, de composants……. Etc.) Internes à l'architecture.

5. Les instructions concurrentes et séquentielles:[13][6]

5.1. Les instructions concurrentes :

Programme concurrent: le programme dans lequel les instructions s'exécutent de manière simultanée. Il n'y a aucun ordre d'exécution de ces événements. VHDL est un programme à exécution concurrente, de telle sorte que pour une description VHDL toutes les instructions sont évaluées et affectent les signaux de sortie en même temps. L'ordre dans lequel elles sont écrites n'a aucune importance. En effet la description génère des structures électroniques, c'est la grande différence entre une description VHDL et un langage informatique classique.

Dans un système à microprocesseur, les instructions sont exécutées les une après les autres.

Avec VHDL il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une bonne description, cela n'est pas toujours évident.

5.1.1. L'affectation simple :

Dans une description VHDL, c'est certainement l'opérateur le plus utilisé. En effet il permet de modifier l'état d'un signal en fonction d'autres signaux et/ou d'autres opérateurs.

Exemple avec des portes logiques : S1 <= E2 and E1 ;

5.1.2. L'affectation conditionnelle :

Cette instruction modifie l'état d'un signal suivant le résultat d'une condition logique entre un ou des signaux, valeurs, constantes.

```
SIGNAL <= expression when condition
[Else expression when condition]
[Else expression];
```

Remarque :

L'instruction [else expression] n'est pas obligatoire mais elle est fortement conseillée, elle permet de définir la valeur du SIGNAL dans le cas où la Condition n'est pas remplie. On peut mettre en cascade cette instruction

5.1.3. L'affectation selective :

Cette instruction permet d'affecter différentes valeurs à un signal, selon les valeurs prises par un signal dit de sélection.

```
With SIGNAL_DE_SELECTION  select
SIGNAL <=   expression  when valeur_de_selection,
[Expression when valeur_de_selection,]
[Expression when others];
```

Remarque:

L'instruction [expression when others] n'est pas obligatoire mais fortement conseillée, elle permet de définir la valeur du SIGNAL dans le cas où la condition n'est pas remplie.

Example N°1:

```
-- Multiplexeur 4 vers 1
With SEL   select
   S2 <= E1 when "00",
E2 when "01",
E3 when "10",
E4 when "11",
'0' when others;
```

Remarque:

When others est nécessaire car il faut toujours définir les autres cas du signal de sélection pour prendre en compte toutes les valeurs possibles de celui-ci.

En conclusion, les descriptions précédentes donnent le même schéma, ce qui est rassurant. L'étude des deux instructions montre toute la puissance du langage VHDL pour décrire un circuit électronique, en effet si on avait été obligé d'écrire les équations avec des opérateurs de base pour chaque sortie, on aurait eu les instructions suivantes :

```
S2 <= (E1  and  not ( SEL (1)) and  not ( SEL (0)))  or (E2  and  not  SEL (1)  and ( SEL (0)) or  ( E3  and  SEL (1) and  not ( SEL (0)))  or  ( E4  and  SEL (1) and  SEL (0));
```

L'équation logique ci-dessus donne aussi le même schéma, mais elle est peu compréhensible, c'est pourquoi on préfère des descriptions de plus haut niveau en utilisant les instructions VHDL évoluées.

5.1.4. Les opérateurs :

a) Opérateur de concaténation : &

Cet opérateur permet de joindre des signaux entre eux.

Exemple :

```
-- Soit A et B de type 3 bits et S1 de type 8 bits
--  A = "001"  et B ="110"
S1 <= A & B & "01" ;
-- S1 prendra la valeur suivante après cette affectation
-- S1 = "001110 01"
```

b) Opérateurs logique :

Opérateur	VHDL
ET	And
NON ET	Nand
OU	Or
NON OU	Nor
OU EXCLUSIF	Xor
NON OU EXCLUSIF	Xnor
NON	Not
DECALAGE A GAUCHE	Sll
DECALAGE A DROITE	Srl
ROTATION A GAUCHE	Rol
ROTATION A DROITE	Ror

Exemples :

```
S1 <= A   sll  2 ;  -- S1 = A décalé de 2 bits à gauche.

S2 <= A   rol  3 ;  -- S2 = A avec une rotation de 3 bits à gauche
S3 <= not (R); -- S3 = R
```

c) Opérateurs arithmétiques :

Opérateur	VHDL
ADDITION	+
SOUSTRACTION	-
MULTIPLICATION	*
DIVISION	/

Remarque N°1 :

Pour pouvoir utiliser les opérateurs ci-dessus il faut rajouter les bibliothèques suivantes au début du fichier VHDL :

```
Use ieee.numeric_std.all;
Use ieee.std_logic_arith.all;
```

Exemples:

```
S1 <= A  –  3 ; -- S1 = A – 3
 -- On soustrait 3 à la valeur de l'entrée / signal A
S1 <= S1  +  1 ; -- On incrémente de 1 le signal S1
```

Remarque N°2 :

Attention l'utilisation de ces opérateurs avec des signaux comportant un nombre de bits important peut générer de grandes structures électroniques.

Exemples :

```
S1 <= A  *  B ; -- S1 = A multiplié par B : A et B sont codés sur 4 bits
S2 <= A  /  B ; -- S2 = A divisé par B : A et B sont codés sur 4 bits
```

d) Opérateurs relationnels :

Ils permettent de modifier l'état d'un signal ou de signaux suivant le résultat d'un test ou d'une condition. En logique combinatoire ils sont souvent utilisés avec les instructions :

- when … else …

- with …. Select ….

Voir ci-dessous:

Opérateur	VHDL
Egal	=
Non égal	/=
Inférieur	<
Inférieur ou égal	<=
Supérieur	>
Supérieur ou égal	>=

5.2. Les instructions séquentielles :

Programme séquentiel : programme se déroulant de manière séquentielle. Toutes les opérations ont lieu les unes après les autres de manière ordrée (comme dans un programme informatique par exemple, ADA, C, C++, Pascal, ...).

Système séquentiel : système numérique dans lequel les valeurs des sorties sont déterminées par des informations précédemment mémorisées dans des bascules, et par les valeurs “actuelles” (aux temps de propagation près) des entrées. Les informations à mémoriser sont enregistrées dans des bascules en synchronisme avec un signal d’horloge. Un événement à une entrée ne se répercute pas forcément de façon “immédiate” (aux temps de propagation près) sur les sorties. Il n’influencera les informations mémorisées que lors du prochain “coup” d’horloge.

Les instructions séquentielles doivent être utilisées uniquement dans une zone de description séquentielle. Nous utiliserons principalement ces instructions à l’intérieur de l’instruction process. Ces instructions peuvent aussi être utilisées dans des procédures et des fonctions.

5.2.1. L’affectation simple :

La syntaxe générique d'une affectation simple est la suivante:

```
Signal_1 <= Signal_2 fonction_logique Signal_3;
```

Remarque:

L'affectation ne modifie pas la valeur actuel du signal. Celle-ci modifie les valeurs futures. Le temps n'évolue pas lors de l'évaluation d'un process.

L'affectation sera donc effective uniquement lors de l'endormissement du process.

L'instruction when ... else n'est pas utilisable à l'intérieur d'un process. C'est une instruction concurrente. L'instruction séquentielle correspondante est l'instruction conditionnelle (if then else).

5.2.2. L'instruction conditionnelle :

Cette instruction est très utile pour décrire à l'aide d'un algorithme le fonctionnement d'un système numérique. La syntaxe générique de l'instruction conditionnelle est la suivante:

```
If  Condition_Booléenne_1 then
--Zone pour instructions séquentielles
elsif Condition_Booléenne_2 then
--Zone pour instructions séquentielles
elsif Condition_Booléenne_3  then
  ...
else
--Zone pour instructions séquentielles
end if ;
```

Nous reprenons la description du détecteur de priorité (3 entrées et une sortie sur 2 bits) pour démontrer l'utilisation de l'instruction conditionnelle.

```
Architecture Comport of Detect_Priorite is
  Begin
    Process (Entree1, Entree2, Entree3)
      Begin
       Priorite <= "00";  -- Valeur par defaut
      if (Entree3 = '1') then
          Priorite <= "11";
       elsif (Entree2 = '1') then
        Priorite <= "10";
```

```
    elsif (Entree1 = '1') then
     Priorite <= "01";
    --else  pas nécessaire, déjà valeur par défaut
    --Priorite <= "00";
   end   if;
 end process ;

end Comport ;
```

5.2.3. L'instruction de choix :

La syntaxe générique d'une instruction de choix est la suivante:

```
case Expression  is
when  Valeur_1 =>  --Zone pour instructions séquentielles
when  Valeur_2 =>  --Zone pour instructions séquentielles
  ....
when others     => --Zone pour instructions séquentielles
end case;
```

Il est possible de définir plusieurs valeurs de la manière suivante:

```
when  Val_3 | Val_4 | Val_5  =>     --Zone pour instructions séquentielles
```

Si le type utilisé pour l'expression est ordonné (exemple : Integer, Natu-ral, ..), il est possible de définir des plages:

```
when  0  to  7   =>  --Zone pour instructions séquentielles
ou
when  31 downto 16  => --Zone pour instructions séquentielles
```

Il est important de noter que le type Std_Logic_Vector n'est pas un ensemble de valeurs ordonnées. C'est un type discret. Il est donc impossible d'utiliser les plages to et downto.

Nous allons montrer l'utilisation de l'instruction case avec la description d'un démultiplexeur 1 à 4. Ce circuit dispose d'une entrée de sélection Sel de 2 bits, une entrée à commuter Val_In et de 4 sorties.

```
Architecture Comport of DMUX1_4 is

    begin

      process (Sel, Val_In)

      begin
            Y0 <= '0'; --Valeur par défaut
            Y1 <= '0'; --Valeur par défaut
            Y2 <= '0'; --Valeur par défaut
            Y3 <= '0'; --Valeur par défaut
         case Sel is
            when "00" => Y0 <= Val_In;
            when "01" => Y1 <= Val_In;
            when "10" => Y2 <= Val_In;
            when "11" => Y3 <= Val_In;
            when  others => Y0 <= 'X';--pour simulation
                                    Y1 <= 'X';--pour simulation
                                    Y2 <= 'X';--pour simulation
                                       Y3 <= 'X';--pour simulation
      end case ;
      end process ;
      end  Comport;
```

Remarque:

Le cas « others » n'est jamais utilisé pour les combinaisons logiques de Sel.

Mais le type « Std_Logique » comprend 9 états ('X', 'U' , 'H', etc.). Le vecteur Sel de 2 bits comprend donc 81 combinaisons et pas seulement 4! Il est indispensable de prévoir ces combinaisons dans la description. Lors de la simulation si Sel est différent d'une des 4 combinaisons logiques ("00", "01", "10","11"), les sorties sont affectées avec l'état 'X' pour indiquer qu'il y a un problème dans le fonctionnement du démultiplexeur.

6. Définition de process : [11]

Un process est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement c'est à dire les unes à la suite des autres.

Un process peut être définie par un label, mais ce n'est pas obligatoire. Il permet d'effectuer des opérations sur les signaux en utilisant l'instruction standard de la programmation structurée comme dans les systèmes à microprocesseurs.

L'exécution d'un process est déclenchée par un ou des changements d'états de signaux logiques. Le nom de ces signaux est défini dans la liste de sensibilité lors de la déclaration du process.

Exemple en utilisant le process :

```
Entity MUX is
        Port ( A, B, SEL : in std_logic;
        OP : out std_logic
        );
End MUX;
Architecture TEST of MUX is
        Begin

        MUXPROCESS: process (A, B, SEL)

        Begin

        If (SEL = '1') then
                OP <= A;
        Else
                OP <= B;
        End if;
    End process MUXPROCESS;
End TEST;
```

7. Les fonctions et procédures : [10][12]

7.1. Rôle, principe et fonctionnement :

Le langage VHDL permet l'utilisation et la création de fonctions ou de procédures que l'on peut appeler à partir d'une architecture. Le rôle de ces fonctions ou procédures est de permettre, au sein d'une description, la création d'outils dédiés à certaines tâches pour un type déterminé de signaux (voir chapitre suivant concernant les types de signaux). Cette notion d'objets est typique aux langages évolués de programmation.

Une fonction reçoit donc des paramètres d'entrées et renvoie un paramètre de sortie. Dans l'exemple précédent, la fonction reçoit trois paramètres A, B et C et retourne un paramètre S. Le mot clé RETURN permet d'associer au paramètre de sortie une valeur. Une fonction peut donc contenir plusieurs RETURN,

7.2. La déclaration des fonctions et procédures :

Dans les exemples qui précèdent, les fonctions ou procédures ont été insérées dans des architectures. Or, dans ce cas précis, les fonctions ou procédures en question ne sont accessibles que dans l'architecture dans laquelle elles ont été décrites. Pour les rendre accessibles à partir de plusieurs architectures, il est nécessaire de les déclarer au sein de packages et de déclarer en amont de l'architecture vouloir utiliser ce package. En reprenant l'exemple de la création d'une fonction NONET et en insérant cette fonction au sein d'un package, voici ce que l'on obtient :

8. Les attributs : [5] [7]

Les attributs sont des propriétés ou des caractéristiques associées à un objet ou à un type. Ils se divisent en deux grandes catégories ; les attributs prédéfinis et les attributs définis par l'utilisateur.

La désignation d'un attribut fait intervenir un préfixe et un suffixe séparés par une apostrophe.

La syntaxe de déclaration de ces attributs est comme suit :

Nom_var' nom_attribut ;

9. Paquetage : [5][7]

Pour ce langage, il existe une unité de conception dont le rôle est de permettre le regroupement de données, de variables, de fonctions et de procédés. Cette unité est une sorte de bibliothèque d'objets (constantes, variables, fonctions ou procédures……etc.) que l'on pourra appeler à partir de plusieurs descriptions. Cette unité est particulièrement intéressante lorsqu'on utilise au sein d'un projet contenant plusieurs entités de ressources identiques.

Un paquetage est constitué de deux parties principales :

- La première est la partie de déclaration ; elle contient principalement des déclarations de types, de constantes, de composants et de sous- programmes destinés à être utilisés par d'autres unités de conception.

Package identificateur is

Déclaration de types, de fonctions, de composants, d'attributs.

End Package ;

- Le corps d'un paquetage contient le corps des sous-programmes déclarés dans la partie visible et des déclarations locales qui n'ont pas vocation à être visibles de l'extérieur.

Package body identificateur is

Corps des sous programme déclarés.

End Package body;

10. Les différences avec un langage de programmation : [6]

La différence majeure avec un langage informatique ("C" ou le "C++") est la simultanéité des actions décrites. Pour maitriser ce langage et faire la différence avec les autres langages, il faut se familiariser avec la notion d'instruction concurrente et d'instruction séquentielle. Par défaut tout ces instructions sont concurrentes, et pour quelles soient séquentielles il faut l'introduire la notion de "process". Le langage VHDL a pour objectif de décrire l'état de la structure matérielle d'un système car ce matérielle a une structure figée dont l'état logique évolue au cours du temps par contre les autres langages de programmation ont un objectif différent est qui est de décrire l'exécution d'un programme. Mais la différence la plus importante réside dans le fait qu'un programme est séquentiel alors qu'une description matériel est parallèle car c'est un problème d'interconnexion du code .il est possible de décrire du matériel avec un langage de programmation en modifiant la sémantique du langage.

Avec ce langage, un système est structuré en composants qui fonctionne en parallèle et en permanence. Par contre, un autre langage de programmation est structuré en sous-programme qui a une durée d'exécution limité dans le temps avec un début en une fin. Il existe aussi une différence qui concerne le support de l'information : car dans un système matériel les composants sont interconnectés avec des signaux qui sont des connexions permanentes et concurrentes, des sorties de canaux par lesquels transitent les informations. La

notion de signal est la base de la description matérielle des structures de donnés qui sont le support de l'information en transit dans le système, car un signal est affecté en permanence. par contre pour les autres langages de programmation les variables supportent l'information et elles sont affectées de manière ponctuelle dans le temps.

11. Les avantages du langage VHDL : [5]

Le langage VHDL offre deux avantages majeurs en plus de la simplicité lors de la conception :

- Simulation :

Le but de la modélisation, c'est la simulation. Il existe deux points importants pour la simulation :

La fidélité: UN modèle doit être aussi précis que possible dans son champ d'application.

L'efficacité: le modèle doit pouvoir être simulé le plus rapidement possible et doit être portable, réutilisable et facile à maintenir.

Pour simuler un modèle, il faut disposer du modèle, bien sûr, mais aussi de stimuler, c'est-à-dire de la description des signaux d'entrées du modèle au cours du temps. Ces stimuler sont appelés les TESTSBENCHES en Anglais.

Les langages fonctionnels de description de matériel permettent de simplifier la conception en analysant automatiquement les résultats en cours de simulation.

Un environnement de simulation complet comprend : un générateur de stimuli, un modèle de l'objet à simuler et un vérificateur automatique des résultats. Ces trios composants sont modélisés à l'aide du même langage.

- Synthèse :

Les langages fonctionnels de description matérielle servent aussi à concevoir. Il ne s'agit plus de modéliser en vue de la simulation, mais de décrire les objets qui seront véritablement fabriqués. Si les considérations de vitesse d'exécution en simulation existent toujours (la description sera simulée avant d'alimenter le synthétiseur, afin de vérifier que la fonction décrite est bien la fonction désirée) elles ne sont plus prioritaires. Ce qui compte le plus, c'est l'efficacité du code au sens du synthétiseur. En effet, pour que ce dernier produise

la description structurelle la plus économique possible (et donc la surface de silicium la plus petite possible).

12. Environnement ISE : [5]

L'environnement ISE est un logiciel de programmation de la famille XILINX. Il est défini comme étant un environnement intégré de développement de systèmes numériques ayant pour objectif une implantation matérielle sur FPGA de la famille XILINX.

ISE intègre différents outils permettant de passer à travers tout le flot de conception d'un Système numérique. En effet il dispose:

-D'éditeur de textes, de schémas et de diagramme d'états.

- D'un compilateur VHDL.

-D'un outil de simulation.

-D'outils pour la gestion des contraintes temporelles.

- D'outils pour la synthèse.

-D'outils pour la vérification.

-D'outils pour l'implantation sur FPGA.

Les étapes d'implémentation d'un circuit numérique à l'aide de cet environnement sont présentées dans l'annexe.

13. Conclusion :

Dans ce deuxième chapitre on a pu faire une présentation détaillée du le langage VHDL, ainsi qu'une petite comparaison avec les langages informatiques, et ces avantages, et pour enfin conclure par un aperçu sur le logiciel de développement ISE qui est l'outil pour l'implantation sur FPGA.

Chapitre 3

Application d'un programme VHDL à la commande en position d'un robot à 3 axes.

1. Introduction :

Dans ce chapitre, nous décrivons les différentes parties réalisées pour la mise en œuvre d'une stratégie de commande par la carte FPGA d'un robot manipulateur à trois axes entrainés par des moteurs pas à pas. Pour ce faire, nous commençons par une brève description du robot qui peut servir à plusieurs applications comme par exemple la réalisation automatique des cartes électroniques, car en plaçant une perceuse sur la tête du robot en peut percer avec une grande précision sur la carte électronique. Puis, on donne une description des circuits de commande en puissance des moteurs pas à pas. Enfin, nous donnons l'organigramme de commande du robot programmé en VHDL ainsi que les résultats de simulation obtenus.

2. Description du robot :

Le robot représenté dans la figure ci-dessous est un robot pour lequel on peut commander la position dont les trois directions qui sont X, Y et Z c-ad c est un robot dans la structure est cartésienne et qui possède trois degrés de liberté. Il est constitué de trois moteur pas a pas car chaque axe a sont propre moteur pour le faire translater. Les longueurs des axes sont comme suit : l'axe des X a 500mm et l'axe des Y a 450mm et l'axe des a Z 250mm. Ce robot peut faire plusieurs taches : soudage par point, perçage, …

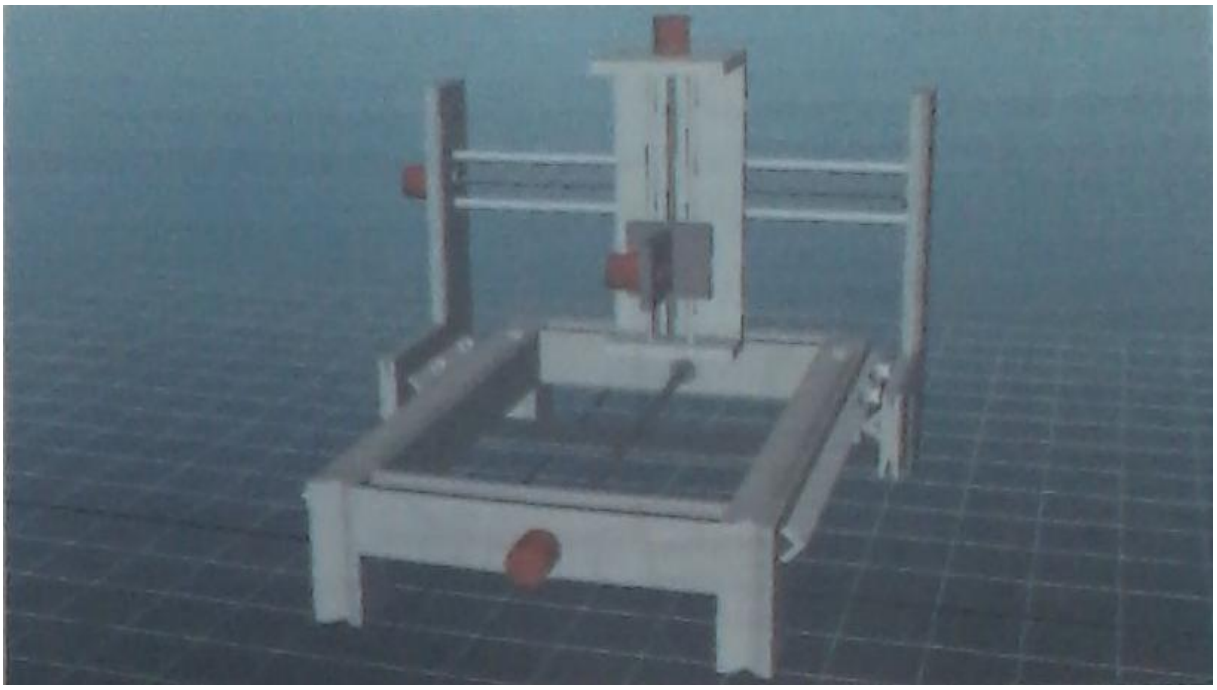


Figure3.1 : la structure du robot

3. Les moteurs pas à pas :

3.1. Généralité sur les moteurs pas à pas : [16][14]

Le moteur pas à pas est un actionneur électromécanique principalement utilisé pour le positionnement en raison de sa grande précision, le fait qu'il joue le rôle d'un convertisseur électromécanique destiné à transformer le signal électrique (impulsion ou train d'impulsions de pilotage) en déplacement (angulaire ou linéaire) mécanique. Il peut tourner avec une

précision de l'ordre du centième de millimètre, en fournissant un couple important avec une faible inertie. De plus, il est très fiable et requiert peu de maintenance puisque sans balai. Son aptitude à fournir une bonne précision en contrôle de vitesse ou de position, combinée à leur petites taille et à leur faible cout, font de ce moteur un actionneur apprécie dans de nombreuses applications telles que les imprimantes, les machines textiles, les machines-outils ou encore dans différents robots industriels.

Le moteur pas à pas est moteur qui tourne dont le rotor se déplace d'un angle élémentaire qui s'appelé pas et cella en fonction des impulsions électriques reçues dans ses bobinage car l'alimentation de chaque bobinage du moteur par une tension particulière provoque l'apparition d'un courant qui engendre un champ magnétique de direction précise. Le changement séquentiel des tensions particulières à chaque bobinage permet de déplacer la position du champ statorique selon une résolution élémentaire appelée pas. Autrement dit, toutes configurations des tensions aux bornes des bobinages correspondent à un déplacement de la position stable du rotor. Une série bien déterminée de commutations de tensions entraîne un déplacement sur un nombre correspondant de pas. La succession des configurations d'alimentation, à une fréquence donnée, impose un champ statorique tournant avec une résolution d'un micro pas, d'un demi pas ou d'un pas entier.

Pour le principe de commande de la rotation ou de la vitesse d'un moteur pas à pas peut se faire sans asservissement : il n'est pas nécessaire de contrôler le résultat qui correspond exactement aux ordres donnés à condition de respecter certaine limite de fonctionnement. Ce mouvement par pas est appelé mouvement incrémental. Pour avoir une bonne résolution dans le positionnement, le moteur pas à pas doit avoir un pas assez faible, c'est un paramètre essentiel de la machine. On peut également caractériser cette résolution par le nombre de pas par tour «N», qui doit évidement être élevé $N = 2\pi/\propto p$.

On trouve deux types de moteur pas à pas qui sont soit aiment permanent ou à réluctance variable. Nous verront par la suite que les moteur à aiment permanent subdivise en deux catégorie qui sont les moteur unipolaire et les moteur bipolaire.

Les moteurs les plus couramment rencontré sur le marché présente des pas qui varie de 24 pas jusqu'à 400 pas (c.à.d. l'ongle de déplacement varie entre 15° et 0.9°). Ils sont alimentés avec une valeur de tension qui dépend du moteur et son utilisation car il varie entre quelque volts et une dizaine de volts et son courant aussi suit le même principe, de quelques mA pour les petits moteurs jusqu'à plusieurs ampères pour les moteurs plus grands. Ils sont présentés avec différente taille qui varie entre 1cm et plus d'une dizaine de centimètres et cella dépend de l'application à faire. Car les plus petits sont destinés à faire des tâches qui ne nécessitent pas un couple important par exemple les disques dure, les têtes de lecteur CD mais par contre pour un déplacement d'un bras manipulateur qui nécessitera un couple important et donc un moteur avec un diamètre élevé. La figure suivante montre deux sortes de moteur pas à pas :

Figure 3.2 : moteur pas à pas

3.2. Les différents types de moteur pas à pas :

Dans ce type de moteur on peut distinguer trois catégories technologiques :

- Moteur à réluctance variable**.**
- Moteur à aimant permanent.
- Moteur hybride.

3.2.1. Moteur à réluctance variable : [14][18]

Le moteur à réluctance variable est caractérisé par une structure dentée au niveau du rotor et du stator. Le nombre de bobines dans le stator et le type de connexion déterminent le nombre de phases du moteur. Le rotor comporte moins de dents qu'il n'y a de pôles au stator et il n'est pas aimanté, comme le montre la figure 3.3.

Le rotor en fer doux, son mouvement est indépendant du sens d'alimentation des différentes phases: le choix de la séquence d'alimentation détermine son sens de rotation.

La rotation de ce moteur pas à pas à reluctance variable, est engendrée par la réaction entre un champ magnétique statorique (partie fixe) et un rotor saillant (partie mobile) qui conduit à une disposition alignée de la partie saillante avec le pôle créé par le champ magnétique.

On utilise dans ce type de moteur la propriété d'une pièce en matériau magnétique doux placée dans un champ magnétique, peut prendre une position de sorte que la reluctance soit minimale, ce qui provoque un couple de rappel.

On note que le calcul de nombre de pas s'effectue de la manière suivante :

-statorique : $s = \frac{360}{8} = 45°$

-rotorique : $r = \frac{360}{6} = 60°$

$\longrightarrow$ par tour : Np=$\frac{360}{r-s}$ =$\frac{360}{60-45}$ =24 pas

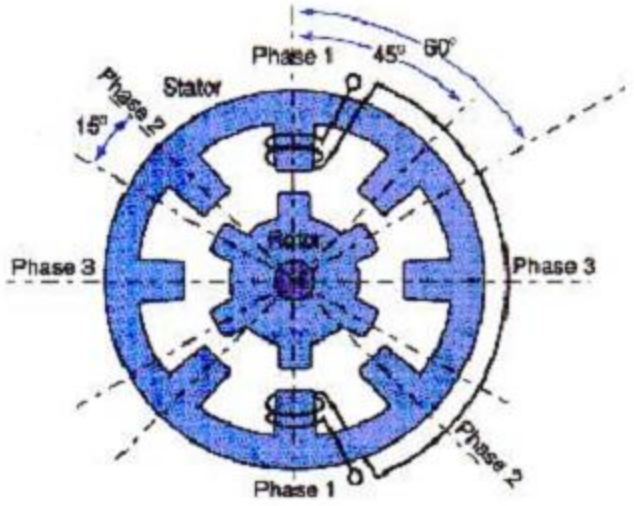


Figure3.3 : moteur à réluctance variable

3.2.2. Moteur à aimant permanent : [18][14]

Il s'agit du type de moteur le plus utilisé dans les applications autres qu'industriel. Car il s'agit d'un moteur qui possède un fort couple, et de basse vitesse, ce qui le rend idéal pour plusieurs applications. Malgré sa construction qui engendre des pas relativement grands, il est compensé par sa simplicité qui lui donne un avantage lorsqu'il s'agit d'une production à grande échelle et à faible coût.

Les moteurs pas à pas à aiment permanent sont ceux utilisés dans la réalisation présentées dans ce projet. Ces moteurs sont composés d'une partie fixe qui est le stator et d'une partie mobile appelé le rotor constitué par un aiment permanent.

Ce type de moteur présente une résolution plus faible due a la discute de loger l'aimant, sa construction est plus complexe, mais il possède un fort couple moteur proportionnel au courant et le sens de rotation dépend de l'ordre d'alimentation des bobines et du sens du courant.

a. Le mode unipolaire :

Ce mode de commande n'utilise en effet qu'une seule bobine par phase. Elle se fait entre une extrémité et le point milieu de bobinage. Cela permet néanmoins de simplifier l'électronique de pilotage vu que l'on travail dans le domaine numérique.

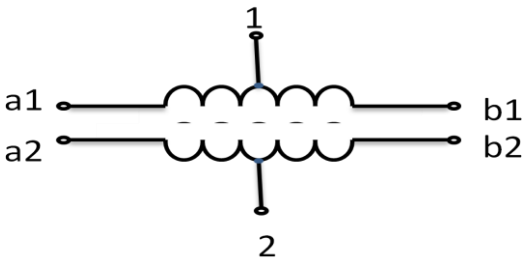


Figure3.4 : Mode unipolaire

b. Le mode bipolaire :

Ce type mode permet d'augmenter le couple de moteur. Il possède des enroulements au niveau du stator qui n'ont pas de point milieu, et chaque borne de l'enroulement est alimentée successivement par une polarité positive ou négative en inversant les polarités des enroulements statique c.à.d. quant inverse le sens de rotation.

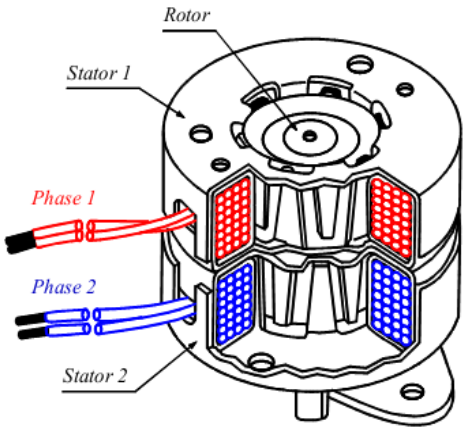


Figure3.5 : moteur pas à pas bipolaire

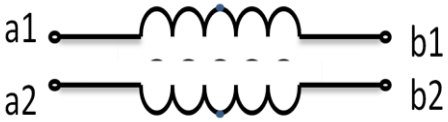


Figure3.6 : mode bipolaire

3.2.3. Moteur hybride : [18][14]

Ce type de moteur sont des moteur à reluctance variable ayant un aiment permanent au rotor. Il combine les avantages des deux moteurs précédents (moteur à aiment permanent et moteur à réluctance variable). Il est le plus utilisé dans la gamme de puissance de moyenne à élevée (jusqu'à 50 Nm). Le rotor dente, sous forme de deux roues polaires (un disque Nord et un disque Sud) décalées d'une dent (correspondant à l'angle électrique), est associe à un aimant axial. Et sans oublié que le couple de reluctance est négligeable. On utilise dans ce moteur les deux principes évoques précédemment. Le rotor du moteur prend la position présentant la résistance magnétique minimale en fonction de l'excitation du stator et du flux magnétique de l'aimant permanent du rotor.

Ce type de moteur présente les avantages du moteur à aimant permanent avec un couple moteur élève et les avantages du moteur à reluctance variable avec un nombre important de pas par tour. Cependant, il faut préciser que son inertie et ses pertes fer sont relativement importantes.

3.3. Configurations interne des bobines d'un moteur pas à pas:[15]

3.3.1. Les moteurs à 6 fils :

Avec ce type de moteur à 6 fils, on a le choix d'une commande que ce soit bipolaire, ou unipolaire. Pour faire une commande bipolaire il suffit d'ignorer les deux fils aux points milieux des deux bobines d'excitation. Pour le deuxième cas qui est la commande unipolaire qui est réalisé en relient les 6 fils à l'alimentation.

Dans notre projet on à utilisé ces type de moteur avec le mode bipolaire.

3.3.2. Les moteurs à 4 fils :

Ce type de moteur agit comme s'il ne possède que deux bobine, dont chacune est alimentée de telle manière à ce que le moteur soit commandé en pas ou en demi-pas.

A tout moment, donc le moteur à la moitie ou la totalité de ses bobines qui sont alimentées, ce qui a comme avantage de lui donner plus de force. Par contre, il est plus complexe de contrôler un moteur bipolaire, au niveau de l'interface de puissance.

3.4. La comparaison entre les trois types de moteur pas à pas:[18]

3.4.1. pour les moteurs à réluctance variable :

-Les fréquences de fonctionnement peuvent être élevées (une grande vitesse de fonctionnement).

-bonne résolution.

-construction simple mais délicate.

-leur faible pas (ongle de pas est compris entre 5° et 30°).

3.4.2. Pour les moteurs à aimant permanent :

- Un fort couple.
- Une fréquence de commutation faible.
- Un couple de maintien non négligeable.

3.4.3. Pour les moteurs hybrides :

Ils développent des couples plus importants que les moteurs à réluctance variable et permettent d'atteindre des fréquences limites en arrêt et démarrage, beaucoup plus élevées que les moteurs à aiment permanent.

En outre ils sont dotés d'un grand nombre de pas par tour. Leur puissance utile est supérieure à celle développé par les deux autre types moteurs (pas angulaire est comprise entre 0.9° et 15°).

4. Commande d'un robot par une carte FPGA:

4.1. Description d'un système de commande des machines électriques :

Vu la complexité et la diversité des systèmes de commande de machines électriques, il est difficile de définir d'une manière universelle une structure générale pour de tels systèmes.

Cependant, en ayant une réflexion par rapport aux éléments les plus communément rencontrés dans ces systèmes, il est possible de définir au mieux une structure générale d'un système de commande de machines électriques comme le montre la figure suivante :

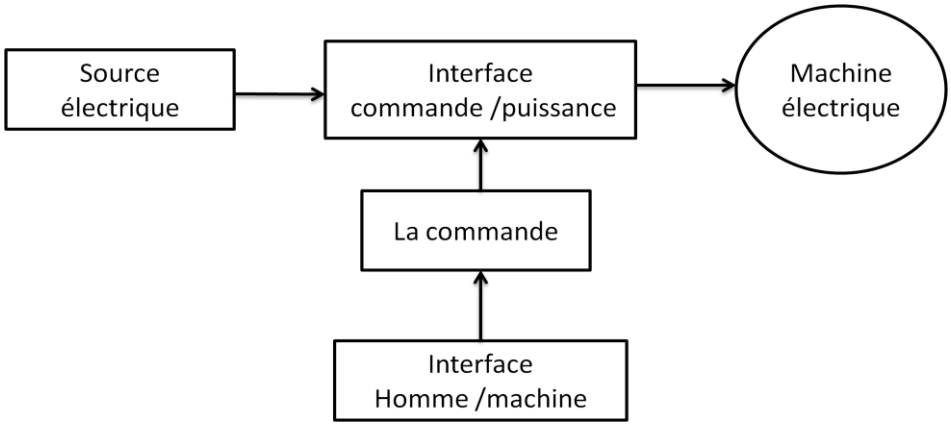


Figure3.7 : Structure générale de commande de machines électriques (moteur pas à pas)

4.2. La structure générale de commande de machines électriques est essentiellement constituée de quatre parties :

- Partie Puissance : Cette partie elle-même inclue quatre éléments. Le premier étant une source électrique. Cette dernière peut être fournie via des batteries, des générateurs, un réseau électrique (monophasé ou triphasé)… Elle peut aussi contenir des composants d'électrotechnique et d'électronique de puissance tels que les transformateurs, les autotransformateurs, les ponts redresseurs (commandés ou non commandés), les filtres capacitifs… Le deuxième élément est c'est l'interface commande /puissance. Il s'agit de dispositifs d'électronique de puissance incluant des interrupteurs de puissance (IGBTs, Thyristors, transistors de puissance…) qui sont commandés à travers des signaux de commande à l'ouverture et/ou à la fermeture. Le rôle de l'interface commande /puissance d'un convertisseur commandé est de convertir l'énergie électrique fournie par la source électrique d'une forme à un e autre à travers la commande d'interrupteurs de puissance. Par exemple, il est possible de trouver l'interface commande /puissance qui est un convertisseur continu-continu tels que les hacheurs, des convertisseurs continu-alternatif tels que les onduleurs, des convertisseurs alternatif-continu tels que les redresseurs (commandés ou non commandés), des convertisseurs alternatif-alternatif tels que les gradateurs… Le troisième

élément est une machine électrique. Cette dernière constitue une charge électromécanique qui est alimentée via l'énergie électrique fournie à la sortie du l'interface commande /puissance (convertisseur commandé). La machine électrique permet de convertir l'énergie électrique qu'elle reçoit en une énergie mécanique sous forme de couple. Plusieurs types de machines sont utilisés dans l'industrie selon l'application considérée et les performances souhaitées. Par exemple, on y trouve les moteurs pas à pas, les machines à courant continu, les machines asynchrones, les machines synchrones … Le quatrième et dernier élément qui constitue la partie puissance est la charge mécanique. Cette dernière utilise l'énergie mécanique délivrée par la machine électrique pour remplir une fonctionnalité donnée.

- Partie Interface (Puissance/Commande) : cette partie permet d'assurer le bon fonctionnement des machine électriques. Elle est constituée d'un élément essentiel qui est le circuit de puissance L293D. Cette partie assure le traitement électronique des signaux électriques échangés entre la partie puissance et la partie commande. Elle est constituée d'éléments tels que les capteurs électriques (capteurs de tension, de courant…), les capteurs mécaniques (couple, vitesse, position…), l'électronique de filtrage des perturbations, l'électronique de conversion analogique numérique et de conversion numérique analogique, l'électronique de pilotage des interrupteurs de puissance du convertisseur commandé…

- Partie Commande : Cette partie assure le contrôle le fonctionnement de la machine électrique (Contrôle de la vitesse et de la position). Ce contrôle étant assuré par un algorithme de commande qui est implanté sur cible numérique. Le contenu algorithmique de la partie commande dépend du cahier de charge de l'application considérée et des performances souhaitées. La partie commande acquiert dans un sens les signaux électriques générés par la partie interface et dans un autre, elle envoie les signaux de commande vers le convertisseur commandé.

- Partie Interface (Homme/Machine) : Cette partie permet le contrôle de l'état du système à travers un échange bidirectionnel d'informations entre le manipulateur et le système commandé. Elle permet dans un sens d'envoyer les consignes de référence (consignes de position et de vitesse) vers la partie commande.

L'objectif de cette partie est d'assurer un contrôle simple et transparent de l'état de la machine électrique.

5. Description de la carte de commande (FPGA) :

De nos jours, les FPGA offrent la possibilité d'utiliser des blocs dédiés tels que les mémoires RAM, les multiplieurs câblés, les interfaces PCI et les cœurs processeurs. La conception des architectures de commande s'effectue en utilisant les outils CAO. La saisie est effectuée graphiquement ou via un langage de description matériel de haut niveau, nommé également langage HDL (Hardware Description Language). Deux langages HDL sont les plus couramment utilisés, à savoir le VHDL (Very high speed integrated Hardware Description Language) et le Verilog. Ces deux langages sont standardisés et offrent au concepteur différents niveaux de description, et surtout l'avantage d'être portables et compatibles avec

toutes les technologies FPGA précédemment introduites. La figure suivante démontre les étapes principales de la programmation de la carte FPGA.

Cependant, les exigences de contrôle modernes dans le domaine de commande de machines électriques dépassent les capacités de calcul offertes par ces solutions. Et bien que les nouveaux multiprocesseurs et les nouveaux DSP de hautes performances puissent résoudre ce problème, ils présentent l'inconvénient d'avoir un coût élevé.

La migration du mode de fonctionnement séquentiel des solutions logicielles au mode de fonctionnement parallèle des solutions matérielles est un nouveau degré de liberté offert aux concepteurs qui s'est avéré bénéfique dans le domaine de commande de machines électriques et qui a permis de répondre aux exigences de contrôle modernes.

Parmi les nouvelles solutions matérielles, les composants FPGA ont été utilisés avec succès dans différentes applications liées à la commande de machines électriques. En effet, ils ont été utilisés pour le contrôle des convertisseurs de puissance tels que les onduleurs de tension triphasé, les convertisseurs alternatif/continu, les convertisseurs multi niveaux, les filtres actifs,… Les FPGA ont aussi été utilisés pour le contrôle des machines asynchrones, des machines synchrones, des machins à réluctance variable …

Ainsi, grâce aux caractéristiques propres des FPGA, il est possible de :

- Améliorer les performances de contrôle: La rapidité de calcul des FPGA permet une augmentation de la bande passante des boucles de régulation et une meilleure résolution temporelle.

- Implanter des algorithmes complexes: Avec l'avancement technologique, l'augmentation d'intégration des composants FPGA ne cesse d'augmenter. De nos jours, la densité des composants FPGA peut atteindre l'équivalent de 10 millions de portes logiques avec des fréquences de commutation de l'ordre de 500 MHz. Ceci permet l'implantation d'algorithmes de contrôle complexes dans leur intégralité avec un faible délai de temps de calcul.

- Réaliser des reconfigurations dynamiques : Le parallélisme inhérent des composants FPGA offre la possibilité de faire tourner plusieurs algorithmes de commande en parallèle et de reconfigurer entre eux selon des critères bien définis. La reconfiguration dynamique entre les algorithmes de commande permet de sélectionner les algorithmes appropriés selon les points de fonctionnements. Elle peut être utile aussi pour assurer une continuité de fonctionnement en cas de défauts (capteurs, interrupteurs, …).

- Renforcer la confidentialité : L'architecture de contrôle implanté sur cible FPGA n'est pas facilement duplicable.

5.1. Disposition du la carte FPGA dans le circuit :

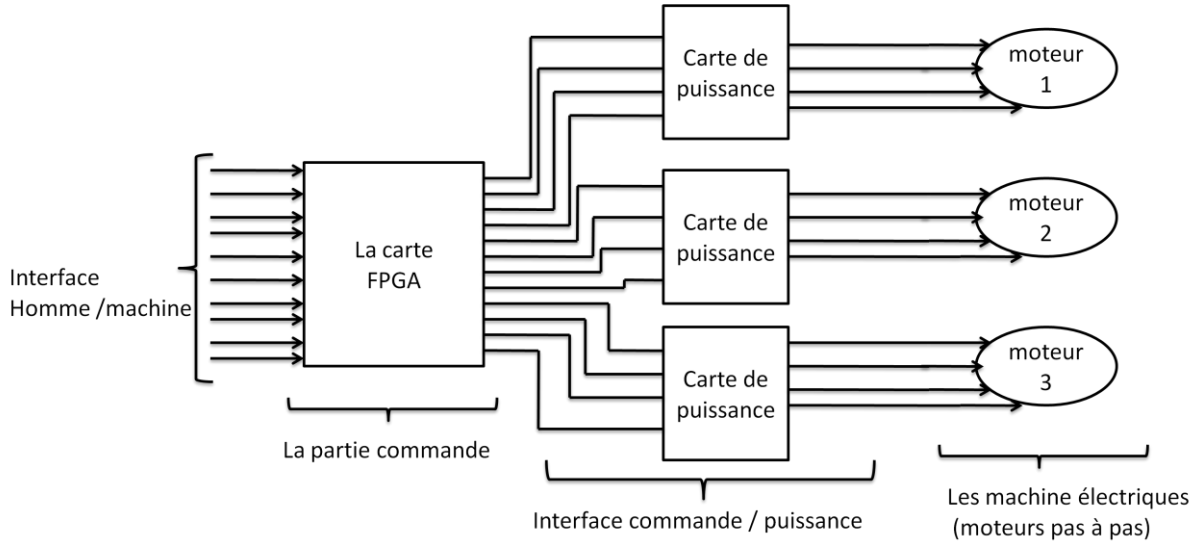


Figure 3.8: description globale de circuit du système

6. La commande des moteurs pas a pas:

La commande qui a été effectué est la commande en position des moteurs pas à pas via les circuits de puissance qui sont représenté ci-dessus. Ainsi, on peut contrôler la vitesse des moteurs pas à pas en commandant l'horloge du système.

6.1. Les cartes de puissance utilisée pour la commande des moteurs pas à pas :

Chaque carte est constituée d'un circuit de puissance qui est L293D et quatre condensateurs de filtrage, deux diodes, deux régulateurs l'un est de 5volts et l'autre est de 9volts.

6.2. Présentation du circuit L293D :

Le circuit L293D est constitué de deux « ponts H ».un pont H est un circuit électronique qui permet, à l'aide d'entrées numériques, d'inverser la polarité aux bornes d'un moteur ou d'une bobine.

Pour valider les quatre sortie pour alimenté le moteurs pas à pas bipolaire il faut tout d'abord alimenter les deux entrées ENABLE1 et ENABLE2. Si on applique respectivement, les niveaux 0 et1 sur input1 et input2, la sortie sera 0 volts sur output1 et +Vs sur output2. Inversement, si on applique respectivement, les niveaux 1 et 0sur input1 et input2, la sortie sera +Vs sur output1 et 0 volts sur output2 (la même chose pour le deuxième étage avec les inputs 3 et 4).

6.3. Bloc d'alimentation du circuitL293D:

Ce bloc d'alimentation est constitué de deux alimentations nécessaires pour alimenter le circuit intégré L293D. La première est de valeur 5 volts et elle est destinée à alimenter la partie logique. La deuxième est une alimentation de la partie de puissance de valeur 9 volts pour alimenter les moteurs pas à pas. Ceci est obtenu a partir d'une alimentions de 12volts, réglé par des régulateur de 9volts et des régulateurs de 5volts. Comme le montre la figure suivante :

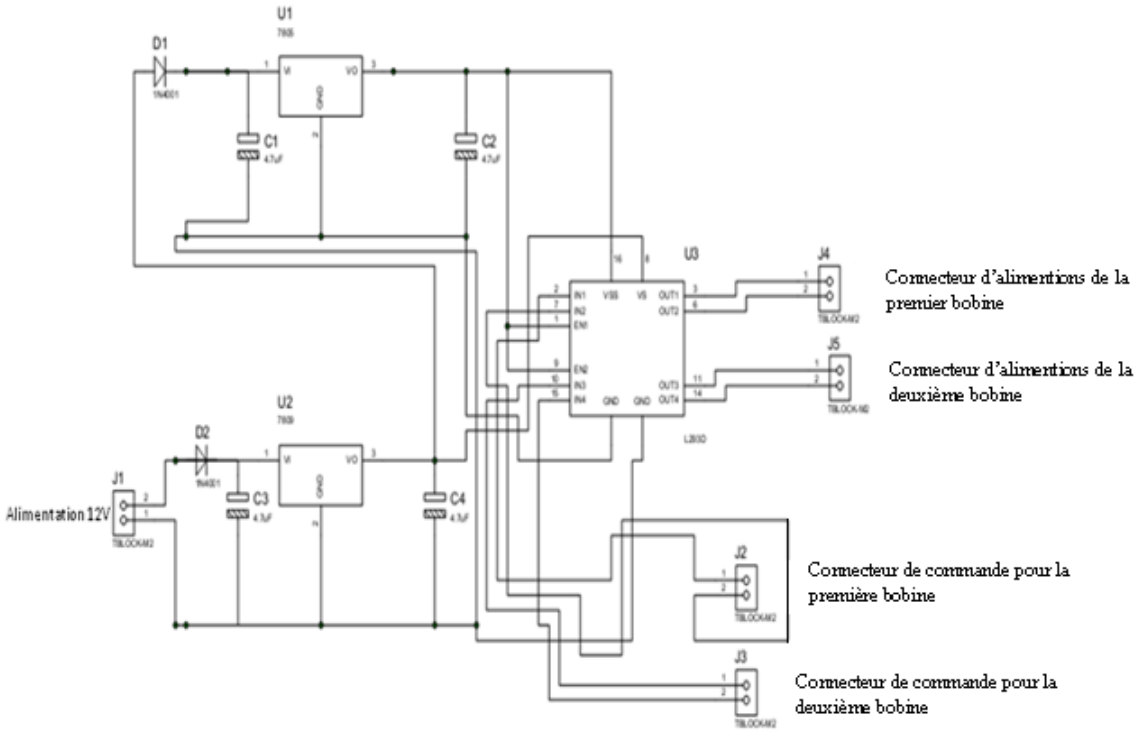
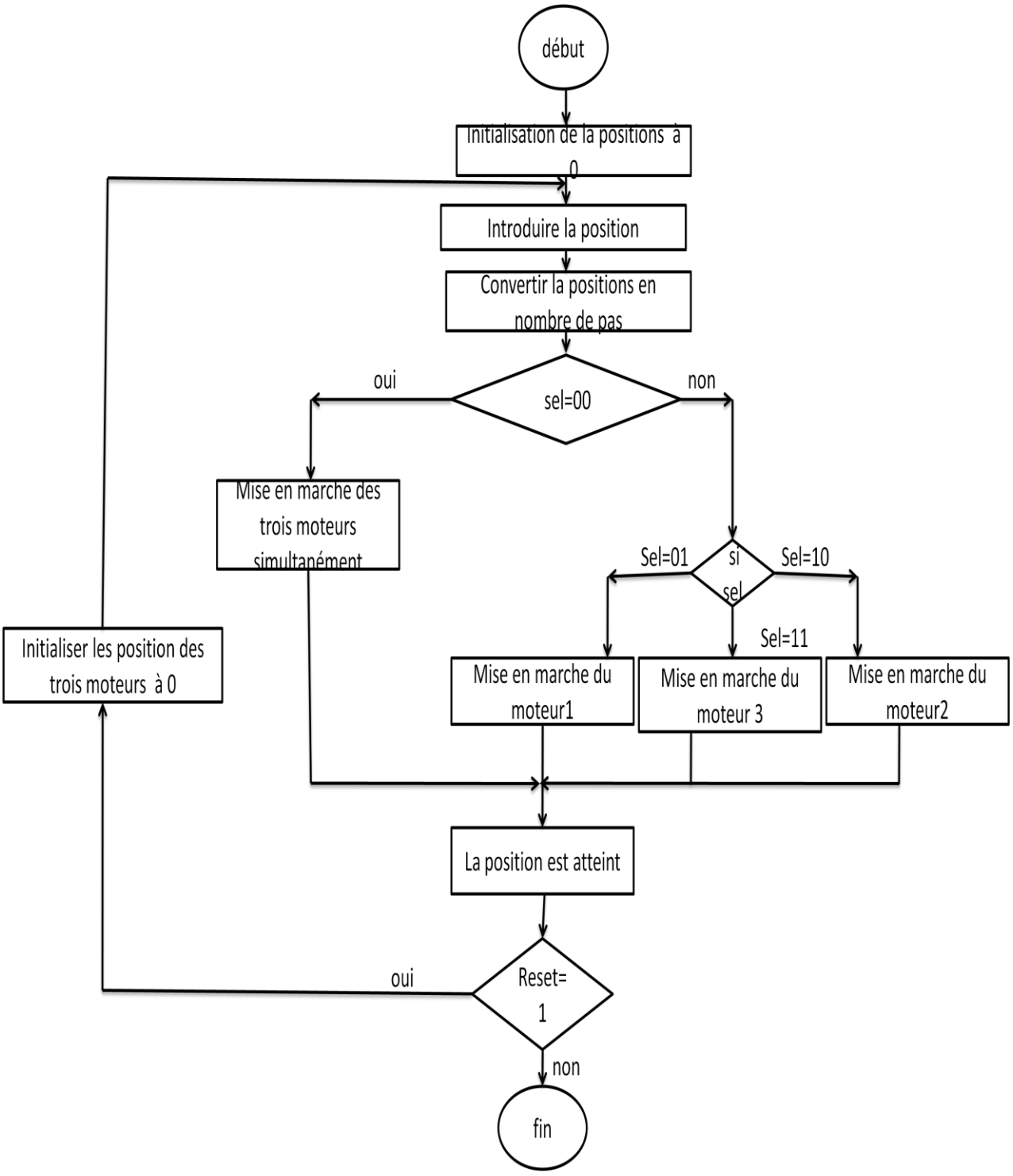


Figure 3.9: Le circuit de la carte de puissance des moteurs pas à pas

7. Organigramme de commande :

L'organigramme suivant est programmé avec le langage VHDL et qui a pour but d'atteindre une position qui est commandé par une carte FPGA et cela sur les trois axes car il suffit d'introduire les trois positions X, Y, Z et la position est atteint par les trois moteurs. Et la figure suivante a décrit le système réalisé :

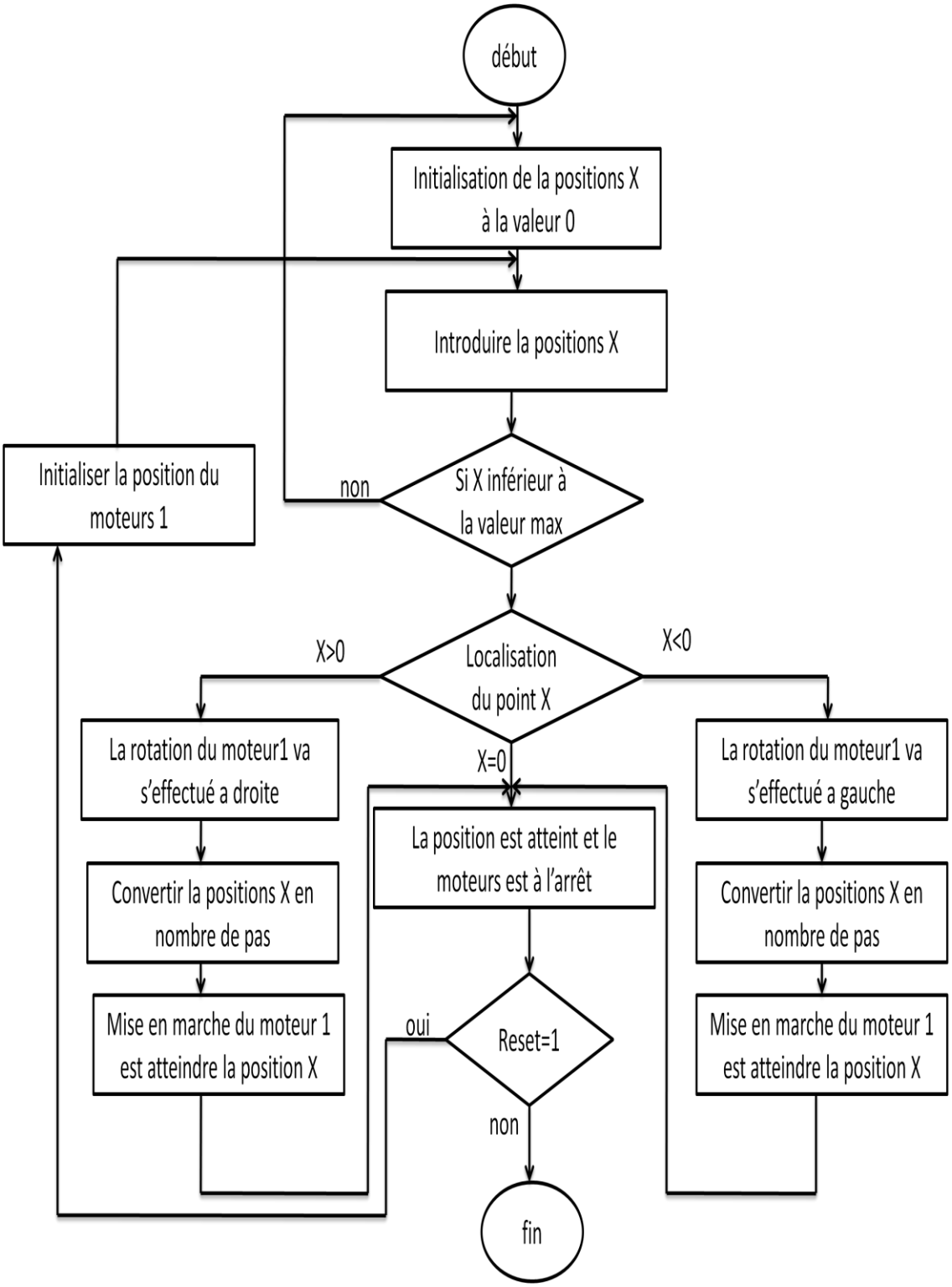
7.1. Organigramme du fonctionnement global de système:



Remarque :

Dans cette application on a utilisé trois moteurs qui fonctionnent soit simultanément ou un par un, et leur organigramme est le même pour chaque moteur.

7.2. Organigramme du premier moteur :



8. Résultats de simulation:

La procédure de simulation est effectuée en utilisant le logiciel XILINX ise. L'objectif de cette étape est de :

- Vérifier la fonctionnalité de l'algorithme de contrôle lorsqu'il est inséré dans l'application considérée.

8.1. Description des signaux :

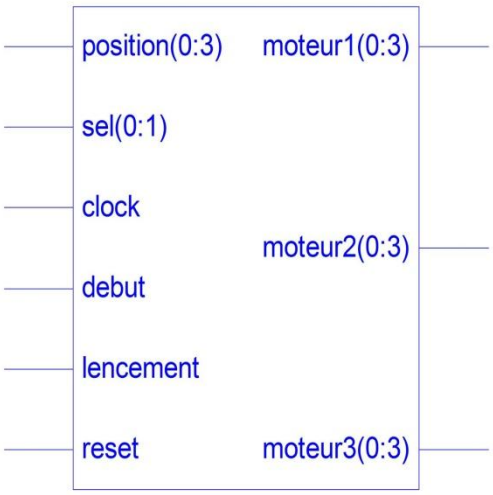


Figure3.10 : illustration des entrées/sortie

Nom	Direction	Description
Debut	Entrée	Bit de démarrage de système.
position[0 :3]	Entrée	Bus de position.
lencement	Entrée	Le passage d'une position à une autre.
Sel[0 :1]	Entrée	Le sélecteur de position
Clk	Entrée	horloge du système.
Reset	Entrée	Reset du système.
Moteur1[0 :3]	Sortie	
Moteur2[0 :3]	Sortie	
Moteur3[0 :3]	Sortie	

8.2. simulation :

La figure suivante montre les résultats de la simulation faite sur le logiciel de XILINX **:**

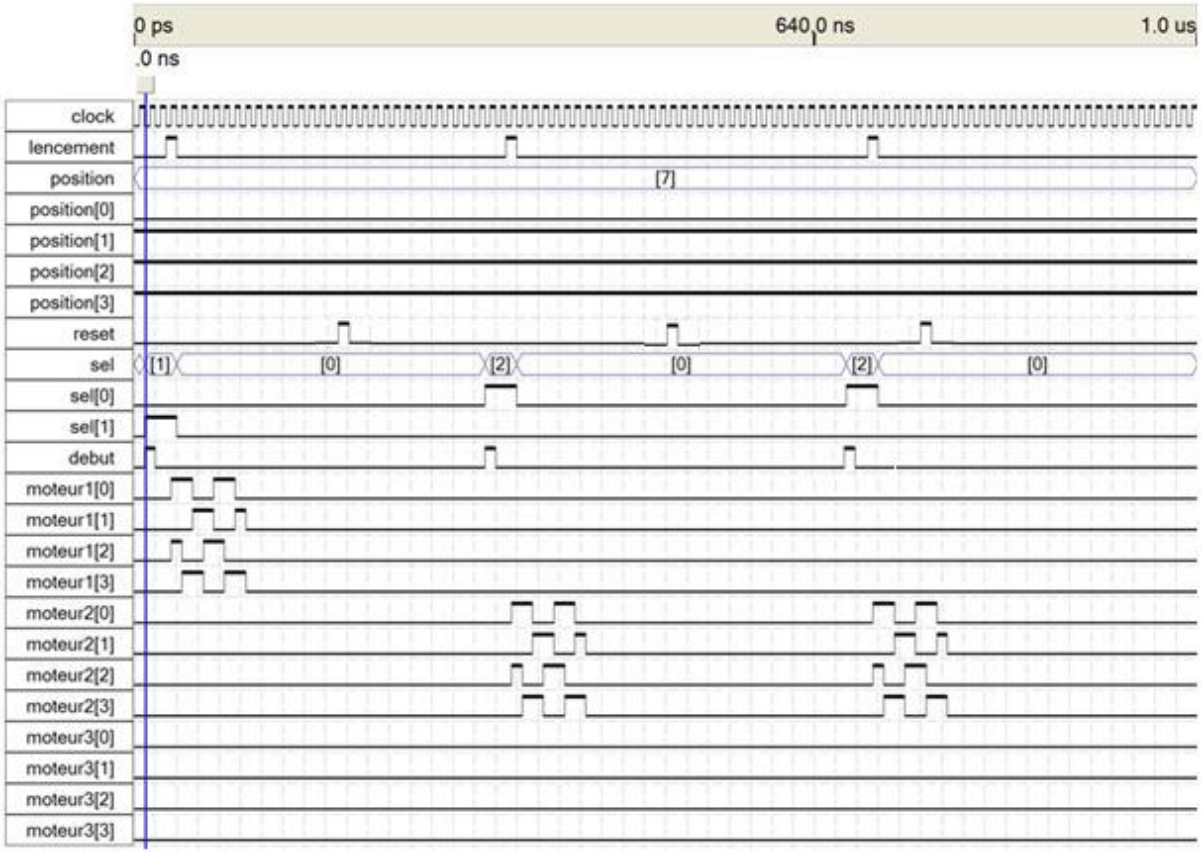


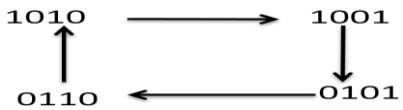
Figure3.11 : résultat de simulation

9. interprétation des résultats de simulation :

Dans cette simulation le changement d'état s'effectue à chaque front d'horloge et cette horloge peut été générer par un circuit interne ou externe (RC) car c'est l'horloge qui contrôle la vitesse d'exécution du système.

Cette carte de commande FPGA génère des signaux de commande pour faire tourner les moteurs pas à pas et cella avec des séquences qui sont :

-séquences de commande pour tourner un moteur à droite :



-séquences de commande pour tourner un moteur à gauche :

10. Conclusion :

Ce chapitre a pour but d'exposer une commande par le langage VHDL, pour cela nous avons passé par un bref rappelles sur les moteurs pas à pas, leurs différents types, ainsi que leur dispositifs de commande particulièrement: la carte FPGA de la famille VIRTEX-5 et les cartes de puissance. Finalement une simulation sur les systèmes embarqués a été considérée pour illustrer notre travail.

Conclusion générale

Conclusion générale :

Nous constatons que l'implémentation dans le domaine de la conception d'une commande sur les circuits numériques à architecture reconfigurable FPGA est complexe, car elle nécessite non seulement une maitrise des technologies relatives aux FPGA mais aussi une très bonne connaissance de l'application.

Pour parvenir à réussir l'implémentation du circuit, on a suivi la méthode scientifique en commençant par la recherche bibliographique, l'étude des documents et l'assimilation des informations et par-dessus toute l'assimilation de la méthodologie propre à l'implémentation des circuits numériques sur un FPGA. Puis viens la partie expérimentale qui est la réalisation du circuit en langage VHDL, on teste le circuit en le simulant ce qui est tout à fait acceptable puis viens la dernière étape qui est la réalisation qui se matérialise par l'implémentation qu'on n'a pas pu réaliser à cause du manque de temps et de matériels. C'est pour cela que nous avons procédé comme suit :

En premier lieu nous nous sommes intéressés à étudier les circuits logiques programmables, le FPGA s'est révélé le circuit logique idéal pour l'implémentation d'un circuit d'une grande densité. Un choix fondé sur les principaux atouts des FPGA à savoir la flexibilité, la fiabilité, la performance, le temps d'élaboration d'un prototype opérationnel et la maintenance à long terme.

Puis nous avons étudié en général l'utilisation du langage VHDL, car les langages comportementaux sont le fruit du perfectionnement des descriptions matérielles ; car le VHDL est un langage puissant qui permet de suivre la conception d'un circuit logique à chaque étape de son élaboration c'est-à-dire la conception, la simulation et la synthèse du fichier de programmation le BITSTREAM.

Grace au travail réalisé, on a pu atteindre notre but et satisfaire le cahier des charges, mais cela ne veut pas dire qu'il est complet. Ainsi, nous proposons que le robot réalisé soit la base de toute une série d'améliorations que nous n'avons pas eu la chance de faire par manque de temps et de matériel.

Annexe

I- Types des signaux :

Comme cité dans le premier chapitre, le langage VHDL est un langage fortement typé. Il possède quatre familles de types à savoir ; les types scalaires, les types composites, les types accès et les types fichiers

I-1- Les types scalaires :

Le type scalaire représente les types énumérés, entiers, flottants et les types physiques.

La base de définition de tous types est l'énumération des éléments contenus. Toutes

les déclarations commencent par le mot-clé « TYPE ». Afin de déclarer ce type de données, il

suffit d'indiquer les valeurs symboliques (identificateurs ou caractères) qu'il peut prendre.

La syntaxe générale de déclaration est :

TYPE <nom_type> IS (val1, val2……);

Par exemple :

TYPE BIT IS ('0','1')

TYPE BOOLEAN IS ('FALSE', TRUE')

I-1-1- Types Entiers :

Les types entiers peuvent être vus comme des types énumérés. En effet, ils représentent une suite énumérable. L'ensemble des opérations arithmétiques est défini sur tous ces types**.**

La déclaration d'un type entier se fait par indication de son intervalle de variation par le mot-clé « RANGE ». L'exemple suivant déclare un type énuméré entier variant de 1 à 12 bornes comprises.

TYPE profondeur_de_la_ pile IS RANGE 1 to 12.

La syntaxe générale de déclaration est :

TYPE <nom_type> IS RANGE min to max.

I-1-2- Types physiques

Ils sont considérés comme des entiers associés à une unité physique.

I-1-3- Types flottants

De manière tout à fait identique aux types entiers, la déclaration d'un type flottant se fait en spécifiant ses bornes (flottantes aussi). L'exemple suivant déclare un type flottant variant de 1.89 à 6.07

TYPE <Nom_flottant> IS RANGE 1.89 to 6.07**.**

I-2- Les types composites :

Comme tout langage de haut niveau, VHDL permet à l'utilisateur de structurer ses

données au moyen des types composites constitués d'éléments scalaires ou eux-mêmes composites, tous de même type ou de types différents. Il existe deux catégories de types composites, les tableaux (array) et les enregistrements (records).

I-2- -1- Tableaux :

Le tableau est une collection d'éléments, tous de même type, repérés par les valeurs d'indices.

I-2-2-Enregistrements :

Ils permettent de rassembler les objets de types différents dans une même organisation et de repérer chaque élément par son nom.

La syntaxe de déclaration est :

TYPE <record_name > IS RECORD.

Elément_déclaration

END RECORD ;

I-3-Les types accès ou pointeurs :

La syntaxe générale de déclaration est:

TYPE <nom_type> IS ACCESS <type_pointe> ;

I-4-Les types fichiers :

Un fichier est une séquence d'objets, de même type, contenus dans un fichier du

système hôte. Au cours de la déclaration d'un type fichier, des procédures d'accès sont déclarées implicitement : FILE_OPEN (…..) ; FILE_CLOSE (…..) ; READ (…..) ; WRITE (…..) et une fonction indique la fin d'un fichier, END FILE(…).

La syntaxe de déclaration est comme suit :

Type <nom_de_type> IS FILE of <type_des_éléments> ;

I- Les attributs :

II-1- Attributs prédéfinis :

Ces attributs sont classés suivant la nature de leur préfixe et la catégorie de l'attribut lui-même ; valeur, type, fonction ou signal.

II-1-1- Attributs de type ou de sous-type :

- **Type de base :**

T'base retourne le type de base de son préfixe. Pour un type donné, le type de base est le type lui-même. Pour un sous-type, le type de base est le type à l'origine du sous-type.

- **Bornes des types scalaires :**

Ces attributs s'appliquent aux types ou aux sous-types scalaires et retournement une de leurs bornes limites. Ces attributs sont :

T'Left ; retourne la borne de gauche du type préfixe.

T'Right ; retourne la borne de droite du type préfixe.

T'Hight ; retourne la plus grande des deux limites du type ou du sous-type préfixe.

T'Low ; retourne la plus petite des deux limites du type ou du sous-type préfixe.

- **Conversion valeur-position :**

Ils permettent la conversion entre variable de position d'un type et la valeur dans le type. Ils sont valables pour des types entiers, énumérés ou physiques.

T'Position (X); retourne la variable de position pour la valeur X.

T'Valeur (X) ; représente la fonction inverse. X représente une position et la fonction retourne la valeur selon la définition du type et du sous-type.

- **Déplacement de position**

Ces quatre attributs permettent d'incrémenter ou de décrémenter une position. C'est la valeur dans le type ou le sous-type qui est retournée.

T'Succ (X) ; retourne le successeur de X dans le type de base.

T'Pred(X) ; retour le prédécesseur de X dans le type de base.

T' LeftOf(X) ; retourne la valeur gauche de X.

T'RighOf (X) ; retourne la valeur droite de X.

II-1-2- Attributs de tableau

Dans ces attributs le préfixe doit être un objet de type tableau, un sous type tableau, un pointeur ou un alias. Ils possèdent un paramètre optionnel (N) permettant de faire porter l'attribut sur une dimension particulière du tableau.

- **Bornes d'index de tableau :**

A'Left (N) ; retourne la borne gauche du Néme index de A.

A'Right (N) ; retourne la borne droite du Néme index de A.

A'Low (N) ; retourne la borne inférieur du Néme index de A.

- **Intervalle de variation d'index de tableau :**

Ces attributs sont utilisés lors de l'écriture des boucles ou pour spécifier à partir d'un type tableau d'autres types ou sous-types.

A'Range (N) ; retourne l'intervalle de variation de la dimension N de A. Dans ce cas,

l'ordre prédéfini est inchangé.

A'Reverse_Range (N) ; retourne l'intervalle de variation de la dimension N de A. L'ordre prédéfini est inversé.

A'Length (N) ; retourne le nombre d'éléments du Néme index de A.

II-1-3- Attributs de signal :

Dans cette catégorie d'attributs, le préfixe doit être un signal. Le type du résultat peut être signal ou fonction.

- **Attribut signal :**

Ces attributs retournent un signal. Les deux premiers sont sensibles à des événements et les deux autres aux transactions portant sur le signal préfix. Les trois premiers possèdent un paramètre optionnel de type « Time » qui ne peut être négatif. Ces attributs ne peuvent être utilisés à l'intérieur de fonctions ou de procédures.

S'Delayed(T) ; est un signal de même type que S dont les valeurs sont retardées de T unités de temps.

S'Stable (T) ; est un signal de type booléen qui est vrai si S n'a pas changé de valeur dans l'intervalle de temps écoulé T.

S'Quiet (T) ; est un signal de type booléen qui est vrai si S n'a pas connu de transaction dans l'intervalle de temps écoulé T.

S'Transaction (T) ; est un signal de type bite qui change de valeur à chaque transaction sur S. Cet attribut permet aussi de transformer une transaction en événement

- **Attribut fonction**

Etant donné que les attributs de signal ne peuvent pas être placés dans un sous-programme, il existe des attributs de fonction.

S'Event ; est un booléen, vrai quand le signal auquel il est appliqué a changé depuis le dernier cycle de test.

S'Active ; est un booléen, vrai quand le signal auquel il est appliqué subit une transaction.

S'Last-Event ; Retourne le Laps de temps écoulé depuis le changement du signal S.

S'Last-Active ; Retourne le Laps de temps écoulé depuis la dernière transaction.

S'Last-Value ; Retourne la valeur qu'avait S juste avant le dernier événement le concernant.

II-1-4- Attributs de bloc

Le préfixe pour ce type d'attribut doit être le nom d'une architecture ou l'étiquette identifiant un bloc.

B'Behavior ; est un booléen vrai si et seulement si B ne contient d'instanciation de composant.

B' Structure ; est un booléen vrai pour une structure pure. Le bloc B ne doit contenir aucune affectation de signal concurrente ou dans un processus.

II-2- Attributs définis

L'utilisateur peut définir ses propres attributs sous réserve qu'ils correspondent à des constantes. La syntaxe générale de déclaration est :

Attribut-déclaration ::=

Attribute identifier :type_mark :

Une valeur lui est associée par une spécification d'attribut :

Attribute_specification ::=

Attribute identifier of entity_specification is expr :

Entity_specification::=

Entity_name_list:entity_class;

II- **Syntaxe résumée du langage VHDL.**

Ces deux pages présentent de manière très condensée la syntaxe du langage VHDL. Il est fortement conseillé de se reporter au document complet pour plus de détails.

Commentaires relatifs au fichier VHDL

```
-- Nom des auteurs
-- Description du rôle du fichier VHDL
-- Circuit logique programmable utilisé
```

Définition des librairies utilisées

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
-- librairie additionnelle pour les opérations arithmétiques
```

Description de l'entité : interface externe. NOMENTITE permet d'identifier l'entité

Début de la description des signaux

```
entity NOMENTITE is
-- Description des signaux d'entrée et de sortie port (
    ENTREE          : in std_logic;                 -- signal simple
    SORTIE          : out std_logic;                -- signal simple
    ENTREESORTIE    : inout std_logic;              -- signal simple
    BUFFER          : buffer std_logic;             -- signal simple
    BUS :       in std_logic_vector (3 downto 0);       -- signaux regroupés en un bus
    A,B :       in std_logic_vector (3 downto 0);       -- 2 bus 4 bits en entrée
    C,D,E,F     : out std_logic -- 4 signaux de même type PAS DE POINT VIRGULE !
        );
```

Fin de la description des signaux

```
-- En plus des lignes précédentes, il est nécessaire de spécifier le brochage
-- retenu pour le circuit utilisé.
```

-- La syntaxe suivante est propre à OrCad Express pour un SPLD
-- La définition du brochage doit respecter les possibilités du circuit :
-- IN ≠ CLOCK IO ≠ OUT

Affectation automatique des broches d'un SPLD

```
-- Affectation automatique
attribute pldtype : string;                    -- définition de l'attribut « Type de broche » attribute
pldtype of HORLOGE : signal is "CLOCK";                    -- ne pas confondre avec IN
attribute pldtype of ENTREE : signal is "IN";
attribute pldtype of SORTIE : signal is "OUT";
attribute pldtype of ENTREESORTIE : signal is "IO"; -- ne pas confondre avec INOUT
attribute pldtype of VALIDATION : signal is "EN";
attribute pldtype of A,B,C : signal is "IN"; -- signaux simples ou bus de même type attribute pldtype
of D,E,F : signal is "IO"; -- signaux simples ou bus de même type
```

Affectation manuelle des broches d'un SPLD

```
-- Affectation manuelle
attribute pldpin : string;                     -- définition de l'attribut « Numéro de broche »
attribute pldpin of NOMSIGNAL: signal is "1";              -- une seule broche
attribute pldpin of NOMBUS: signal is "23,22,21,20";              -- LSB en premier !
```

```
end
NOMENTITE;
```

Fin de l'entité : NOMENTITE correspond au nom donné dans la déclaration

```
architecture NOMARCHITECTURE of NOMENTITE is

signal NOMAUX : STD_LOGIC_VECTOR (3 downto 0);

begin

-- partie Combinatoire
NOMBUS <= NOMAUX ; -- affectation du signal auxiliaire au bus de sortie

SORTIE <=     ENTREE0 when ( ENTREECOMMANDE =      "00")
else
              ENTREE1 when( ENTREECOMMANDE =
                                                   "01") else
              ENTREE2 when( ENTREECOMMANDE =       "10") else
              ENTREE3 ;

with ENTREECOMMANDE select -- signal de sélection
   SORTIE <= ENTREE0 when "00", -- quand EntreeCommande = "00", Sortie vaut Entree0
   ENTREE1 when "01",           -- quand EntreeCommande = "01", Sortie vaut Entree1
   ENTREE2 when "10",           -- quand EntreeCommande = "10", Sortie vaut Entree2
   ENTREE3 when others;         -- Sortie vaut Entree3 dans les autres cas

-- partie Séquentielle
process (SIGNAL1, SIGNAL2,... )
   begin
  if SIGNAL1'event and SIGNAL1 = '1'-- détection d'un front montant sur Signal1 then if
     NOMAUX >= x"9"                -- test d'inégalité
              then NOMAUX <= x"0";      -- affectation simple
              else NOMAUX <= NOMAUX + 1;   -- incrémentation et affectation (autorisé end if; --
          seulement avec la librairie ieee.std_logic_unsigned.all)
     end if;

  case ENTREECOMMANDE is -- signal de sélection
       when "00" => SORTIE <= ENTREE0; --quand EntreeCommande = "00", Sortie vaut Entree0
when "01" => SORTIE <= ENTREE1; --quand EntreeCommande = "01", Sortie vaut Entree1 when "10"
=> SORTIE <= ENTREE2; --quand EntreeCommande = "10", Sortie vaut Entree2 wheN others =>
SORTIE <= ENTREE3; --dans les autres cas, Sortie vaut Entree3--      "11" : binaire pour un bus
                                                              --      x"11" : heXadécimal pour un bus


  end case;
end process ;

end NOMARCHITECTURE;

-- REMARQUES :Fin de l'Architecture :
```

Description de l'architecture : fonctionnement interne.
NOMARCHITECTUREpermet d'identifier l'architecture.
NOMENTITE correspond au nom donné dans la déclaration de l'entité.

Début de l'Architecture

Déclaration facultative de signaux auxiliaires

Description combinatoire : mode « Concurrent »

Affectation simple

Affectation conditionnelle

Affectation sélective

Description séquentielle : PROCESS suivi de la liste des signaux pouvant déclencher le process

Assignation Conditionnelle : if ... then ... else...

Assignation Sélective : case ... is ... when...

Fin du Process

--NOMARCHITECTURE correspond au
--nom donné dans la déclaratio

IV- Etape d'implémentation sous environnement ISE :

L'implémentation d'un circuit numérique se fait suivant les étapes suivantes :

IV-1- Lancement d'ISE 9.2i et création du projet :

Cette étape consiste à lancer le Navigateur ISE et à créer le projet à réaliser. Cela se fait comme suit :

- On clique deux fois sur l'icône sur le bureau de travail où on choisit :

Start→ Programs→ XilinxISE →Project Navigator.

- Si des messages apparaissent, on sélectionne cancel

- On choisit **File →New Project.**

- On donne un nom pertinent au projet à réaliser.

- Etant donné qu'on désire réaliser un design à base d'un code HDL, on choisit **HDL** comme **Top_Level Source Type.**

- Ensuite on clique sur **Next.**

- On choisit les paramètres du circuit programmable utilisé à savoir :

Family : permet de choisir la famille du circuit logique programmable.

Device : permet de sélectionner le sous-type (nombre de portes logiques).

Package : définit le type de boîtier.

- On appuie sur **Next.**

- Enfin, on appuie sur **Next** deux autres fois pour les deux autres fenêtres et sur **Finish.**

IV-2- Description du circuit numérique à réaliser avec un fichier VHDL :

Cette étape permet de réaliser un fichier source qui renferme les données descriptives ou de configuration du circuit à réaliser. La construction de ce fichier se fait suivant les instructions suivantes :

- On sélectionne le menu **Project → new Source.**

- On sélectionne **VHDL Module** comme source et on entre le mot du schéma à réaliser.

- On vérifie que l'option **add to Project** est cochée.

- On clique sur **Next.**

- On déclare les ports entrée/sortie du design à réaliser.

- On clique sur **Next** et ensuite sur **Finish**. Une ébauche du fichier apparait avec la description de l'entité et de l'architecture.

- Dans la description de l'architecture, en dessous de l'énoncé begin, on insère les fonctions réalisant notre schéma ou circuit.

IV- 3- Compilation :

Une fois le fichier VHDL édité, il est conseillé de vérifier la syntaxe du design dans le but de trouver les erreurs de syntaxe ou de typographie :

- On vérifie que **synthesis / implementation** est sélectionné dans la liste déroulante de la fenêtre **sources.**

- On sélectionne le fichier VHDL **VHDL_didacticiel** pour afficher les processus liés dans la fenêtre **processes.**

- On clique sur le « + » à côté de **synthesize-XST.**

- On double-clique sur le processus **check syntax.** Si tout va bien, un crochet vert apparaît. Sinon :

- On consulte les messages d'erreurs dans la console au bas de l'écran.

- On corrige les erreurs s'il y a lieu, puis on ferme le fichier VHDL.

IV- 4- Simulation du design :

Cette étape permet de vérifier que le design décrit précédemment fonctionne de la façon prévue par les spécifications. Cette simulation est une simulation comportementale.

Afin de réaliser cette dernière, il faut créer un banc d'essai contenant les stimuli d'entrée.

- On sélectionne le fichier à simuler **(VHDL_didacticiel).**

- On crée un nouveau banc d'essai en sélectionnant **Project_ New Source.**

- Dans la fenêtre qui s'ouvre, on sélectionne **Test Bench WaveForm** comme source, et on donne un nom au fichier.

- On clique sur **Next.**

- Une page apparait montrant quel fichier source est associé au banc d'essai; il s'agit de notre fichier à simuler.

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

- Une fenêtre permettant d'effectuer une initialisation temporelle (détermination des paramètres de l'horloge et de certaines contraintes temporelles) apparaît.

- On clique enfin sur **Finish** pour terminer cette initialisation temporelle.

- Dans la fenêtre **sources,** on sélectionne **behavioral simulation.**

- On assure que le banc d'essai est bien sélectionné dans la fenêtre **sources.**

- Dans la fenêtre **processes**, on déroule l'outil **xilinx ise simulator.**

- On double-clique sur **Simulate Behavioral Model.**

- On corrige notre design si nous obtenons des erreurs.

IV-5- Synthèse et implémentation du design :

IV-5-1- Description de la synthèse et de l'implémentation :

La synthèse d'un circuit consiste à traduire la description du circuit à réaliser en blocs disponibles dans la technologie utilisée. Par exemple, pour un circuit décrit avec un code VHDL et qui doit être réalisé sur FPGA, le processus de synthèse convertit et regroupe les portes logiques en composantes réalisables sur le FPGA choisi.

L'implémentation du circuit se fait en quatre sous étapes :

- La transformation **(mapping)** : elle consiste à regrouper les composants obtenus lors de la synthèse dans des blocs spécifiques du FPGA.
- la disposition **(placement)** qui consiste à choisir des endroits spécifiques sur le FPGA où disposer les blocs utilisés, et choisir les pattes du FPGA correspondant aux ports d'entrée et de sortie .
- le routage **(routing)** : établir des connexions électriques entre les blocs utilisés.
- la configuration **(configuration)** : convertir toute cette information en un fichier pouvant être téléchargé sur le FPGA pour le programmer.

IV-5-2- Ports d'entrée et de sortie :

Lors de l'étape de disposition de l'implémentation, il faut assigner des pattes spécifiques du FPGA à des ports d'entrée et de sortie de son design.

L'assignation des ports se fait par l'entremise d'un fichier de contraintes avec l'extension «.ucf » (pour user constraints file).

- Dans la fenêtre **sources**, on sélectionne **Synthesis/Implementation**

- Dans la fenêtre **Processes,** on déroule le menu **User Constraints** puis on double-clique sur **Assign Package Pins.**
- Dans la fenêtre suivante, on clique sur **Yes** afin de créer automatiquement le fichier d'assignation (.ucf) des ports du FPGA.
- Une fenêtre d'assignation de pattes va s'ouvrir. On fait entrer, dans la section Loc du menu **Design Object List – I/O Pins,** les numéros des pattes reliant le FPGA au design.
- Afin de sauver les allocations effectuées, on sélectionne **File_Save.** Puis on sélectionne **XST Default ;** pour le type de sauvegarde. Et enfin, on clique sur **Ok.**
- On ferme la fenêtre d'assignation des pattes.
- Dans la fenêtre **Sources,** on sélectionne le fichier VHDL **VHDL_didacticiel–Behavioral (didactiel.vhdl).**
- Dans la fenêtre **Processes,** on double-clique sur **Generate Programming File.** Après une attente de quelques minutes, on obtient un schéma indiquant que les étapes de synthèse ont bien été effectuées.

IV- 6-Programmation du FPGA et tests :

La démarche pour la programmation d'un circuit logique programmable est :

- On double-clique sur **Configure Device (IMPACT).**
- Si un message d'avertissement s'affiche on clique sur **Cancel.**
- On sélectionne **Configure Devices using Boundary-Scan (JTAG).**
- On clique sur **Finish.**
- On sélectionne le fichier de programmation créé **(VHDL_didacticiel.bit** pour un design basé sur un fichier VHDL**).**
- On clique sur **Ok** si un message d'avertissement apparaît.

Référence Bibliographique

Bibliographies:

[1] : Christian Tavernier ; circuits logiques programmables; Dunod, Paris, 1996.

[2] : Alexandre NKETSA; circuit logique programmable Mémoires, PLD, CPLD, FPGA; 1998.

[3] : LAURENT DUTRIEUX et DIDIER DEMIGNY ; logique programmable Architecture des FPGA et CPLD méthodes de conception le langage VHDL ; édition Eyrolles 1997.

[4] : Jean-Max DUTERTRE ; THESE sur Circuits Reconfigurables Robustes Pour obtenir le grade de DOCTEUR DE L'UNIVERSITE MONTPELLIER II ; Le 30 octobre 2002.

[5] : NACHEF TOUFIK ; mémoire sur implantation d'une instruction sur un FPGA Pour obtenir le diplôme Magister en Electronique à UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU ; 2011.

[6] : NAAK HOURIA, HACID HASSINA et MOUSOUNI FADHILA; mémoire sur implantation d'un variateur de vitesse à base d'un circuit FPGA Pour obtenir le diplôme d'ingénieur d'état en électronique à UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU ; 2007.

[7] : Mr CHIBAH AREZKI ; mémoire sur CONCEPTION D'UN CONTROLEUR D'ETAGE DE PUISSANCE PAR FPGA Pour obtenir le diplôme de Magister en Electrotechnique Option: Entrainements Electriques à UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU ; 2011.

[8] : Ahmed BEN ATITALLAH ; THESE sur Etude et Implantation d'Algorithmes de Compression d'Images dans un Environnement Mixte Matériel et Logiciel, POUR OBTENIR LE GRADE DE DOCTEUR à L'UNIVERSITE BORDEAUX 1; 2007.

[9] :www.xilinx.com.

[10] : D. HOUZET, L. BARRANDON ; CONCEPTION DE CIRCUIT EN VHDL ET VHDL-AMS ; 2006.

[11]: ROMAIN BERNY ; INTRODUCTION AU LANGAGE VHDL POUR LA SYNTHESE ; 2005.

[12] : JACQUES WEBER, MAURICE MEAUDRE ; le langage VHDL cours et exercices ; 2001.

[13] : Philippe LECARDONNEL & Philippe LETENNEUR ; Introduction à la Synthèse logique V.H.D.L ; 2001.

[14] : Frédéric Nollet ; thèse sur LOIS DE COMMANDE PAR MODES GLISSANTS DU MOTEUR PAS A PAS pour obtenir le grade de DOCTEUR délivré conjointement par l'Ecole Centrale de Lille et l'Université des Sciences et Technologies de Lille ; le 7 décembre 2006.

[15] : HADID MALIK et HAMI SOFIANE ; mémoire sur la conception et réalisation d'une machine extrudeuse à commande numérique en temps réel pour obtenir le diplôme d'ingénieur d'état en automatique à l'université MOULOUD MAMMERI DE TIZI OUZOU; 2010.

[16] : SALHI FARID et REZKI DAHMANE ; mémoire sur la commande à distance d'un chariot mobile avec évitement d'obstacle pour obtenir le diplôme d'ingénieur d'état en automatique à l'université MOULOUD MAMMERI DE TIZI OUZOU; 2010.

[17]: Luc Yin; Rapport de Stage SUR L'Étude et Réalisation d'une commande moteur pas-à-pas; 2008.

[18]: CHAIB AHMED et MAHFOUF SAID; mémoire sur l'étude et réalisation d'un bras articulé commandé par un micro ordinateur pour obtenir le diplôme D.E.UA en électronique à l'université MOULOUD MAMMERI DE TIZI OUZOU; 2008.