République Algérienne Démocratique et Populaire Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DU GENIE ELECTRIQUE ET D' INFORMATIQUE

DEPARTEMENT *D'ELECTRONIQUE*

MEMOIRE DE FIN D'ETUDES

En vue de l'obtention du diplôme de

MASTER ACADEMIQUE EN ELECTRONIQUE

OPTION : INSTRUMENTATION

Thème

Conception de commandes d'une Imprimante 3D à base d'une carte FPGA Altera DE2

Proposé et deriger par : **Mme.CHIBANE**

Jurys :

Presidante : **Mme.Ait Abdelmalek**

Examinateur : **Mr.Khalifa**

Présenter par :

FOUNAS RAYANE

GUECHIDA AMAR

Promotion 2024

Remeriements

Remerciements

Nous tenons à exprimer tout d'abord, nos profonds remerciements et louanges à ***Dieu***

Qui nous a guidé sur le droit chemin et nous a donné le courage et la volonté d'achever ce modeste travail.

Nos remerciements s'adressent à l'ensemble des jurys, de nous avoir honorés en acceptant d'évaluer ce travail

Au terme de notre travail, nous tenons à exprimer toute nos reconnaissances et notre profond respect à nôtres promotrice Madame ***Chibane Loundja .*** *Pour son aide, ces précieux conseils et l'orientation dont on a pu bénéficier.*

Nos sincères remerciements, pour ***Mme ait Abdelmalek et Mr Khati****,* ***Mr BENAMANE****, pour leurs aides précieuses dans la réalisation de ce travail.*

Nos remerciements vont également aux personnels du laboratoire du département d'électronique.

Nos remerciements vont également à ***Mr HAMID KLALECHE*** *chef du service moyens générale et maintenance de la faculté génie électrique et informatique.*

Dédicace

Dédicaces

En signe de respect et de reconnaissance, je dédie ce modeste travail à :

A mes parents, *pour les valeurs que vous m'avez transmises avec tant de patience et d'attention ; pour votre affection, votre soutien moral et votre tolérance. Que ce travail soit témoin de reconnaissance éternelle de mon respect et mon amour pour vous.*

Que Dieu vous bénit.

A mes très chères sœurs, *pour votre amour et encouragements.*

A mon adorable frère :

Khider pour tous ce que vous m'apportez, votre grand amour et soutien.

A ma tante paternelle Malha.

A mon binôme : RAYANE

A mes amis : *KAMEL Et Mounia.*

.

AMAR.

Dédicaces

En signe de respect et de reconnaissance, je dédie ce modeste travail à :

A mes parents, *pour les valeurs que vous m'avez transmises avec tant de patience et d'attention ; pour votre affection, votre soutien moral et votre tolérance. Que ce travail soit témoin de reconnaissance éternelle de mon respect et mon amour pour vous.*

Que Dieu vous bénit.

A mes sœurs, *pour ton amour et encouragements.*

A mon binôme : AMAR

A mes amis : *Ghani Et Kamel.*

A mes amis de la fac

RAYANE

Liste des Figures

Liste Des Figures

Liste Des Figures

Liste Des Figures

LISTE DES TABLES

Sommaire

Sommaire

Sommaire

Sommaire

Introduction Générale

Introduction Générale

L'innovation technologique dans le domaine de l'électronique a transformé de nombreux secteurs industriels, en particulier avec l'avènement des FPGA (Field Programmable Gate Array) et des microcontrôleurs. Ce projet s'inscrit dans cette dynamique de transformation en explorant l'intégration de ces deux technologies pour le contrôle précis des moteurs pas à pas, qui jouent un rôle crucial dans diverses applications telles que les imprimantes 3D, la robotique, et les systèmes d'automatisation.

Les FPGA sont des circuits intégrés qui peuvent être programmés après fabrication pour réaliser des fonctions logiques spécifiques. Ils offrent une flexibilité et une reprogrammabilité inégalées, permettant aux ingénieurs de concevoir et d'implémenter des systèmes complexes sans les contraintes des circuits intégrés spécifiques. Les microcontrôleurs, quant à eux, sont des circuits intégrés compacts utilisés pour contrôler d'autres dispositifs électroniques. Leur facilité d'utilisation et leur large adoption dans les projets d'électronique en font un choix idéal pour les interfaces utilisateur et les contrôles basiques.

Le moteur pas à pas est un composant essentiel dans les systèmes où un positionnement précis est requis. Contrairement aux moteurs conventionnels, les moteurs pas à pas avancent par des pas discrets, permettant un contrôle fin et précis de leur position et de leur vitesse. Ces caractéristiques les rendent indispensables dans les imprimantes 3D, où la précision du mouvement est cruciale pour la qualité de l'impression. Cependant la commande efficace de ces moteurs nécessite des systèmes de contrôle sophistiqués capables de gérer les dynamiques complexes et de répondre aux exigences de performance élevées.

Les FPGA (Field-Programmable Gate Arrays) se sont révélés être une technologie prometteuse pour la réalisation de système de contrôle avancés. Grace à leur architecture reprogrammable et à leur capacité à exécuter des opérations parallèles, les FPGA offrent une flexibilité et une performance supérieure par rapport aux microcontrôleurs et aux processeurs numériques traditionnels. Ils permettent une personnalisation materielle qui peut être adaptée précisément aux besoins spécifiques de l'application, tout en offrant des délais de réponse rapides et une grande fiabilité.

L'objectif de cette thèse est de développer un système de commande pour un moteur pas à pas bipolaire à aiment permanent pour une imprimante 3D en utilisant une carte Arduino comme génératrice de signaux jouant le rôle du contrôleur d'une l'imprimante 3D et une carte FPGA pour l'implémentation du programmes de fonctionnement du moteur pas à pas. Pour y arriver

nous avons modélisé théoriquement notre système, conçu des algorithmes de commande et mis en œuvre pratiquement notre réalisation sur FPGA, en passant par la validation expérimentale de notre système.

Pour mener à bien notre travail, nous l'avons structuré comme suit :

Dans le premier chapitre nous avons effectué une étude détaillée de la technologie FPGA en passant en revue leur architecture interne, leurs fonctionnalités, les langages de description matérielle(HDL) utilisés pour les programmer.

Le deuxième chapitre présente une étude théorique de la littérature existante sur les moteurs pas à pas et leurs méthodes de commande. L'étude des technologies utilisées dans les divers types d'imprimantes 3D et le rôle crucial du moteur pas dans le contrôle des mouvements des axes avec précision et synchronisation sont aussi présentés et discutés.

Dans le troisième et dernier chapitre nous avons procédé à la réalisation pratique de notre système en passant par la réalisation du programme de commande du moteur pas à pas et la visualisation des signaux relatifs en utilisant le logicielle « Quartus II 9.1sp2 Web Edition » puis nous avons généré les signaux de contrôle à partir d'une carte Arduino que nous avons programmé pour jouer le rôle d'un contrôleur d'une imprimante 3D . Nous avons clôturé cette thèse par une conclusion générale sur les résultats obtenus avec les perspectives intéressantes qui peuvent être envisagées dans le futur.

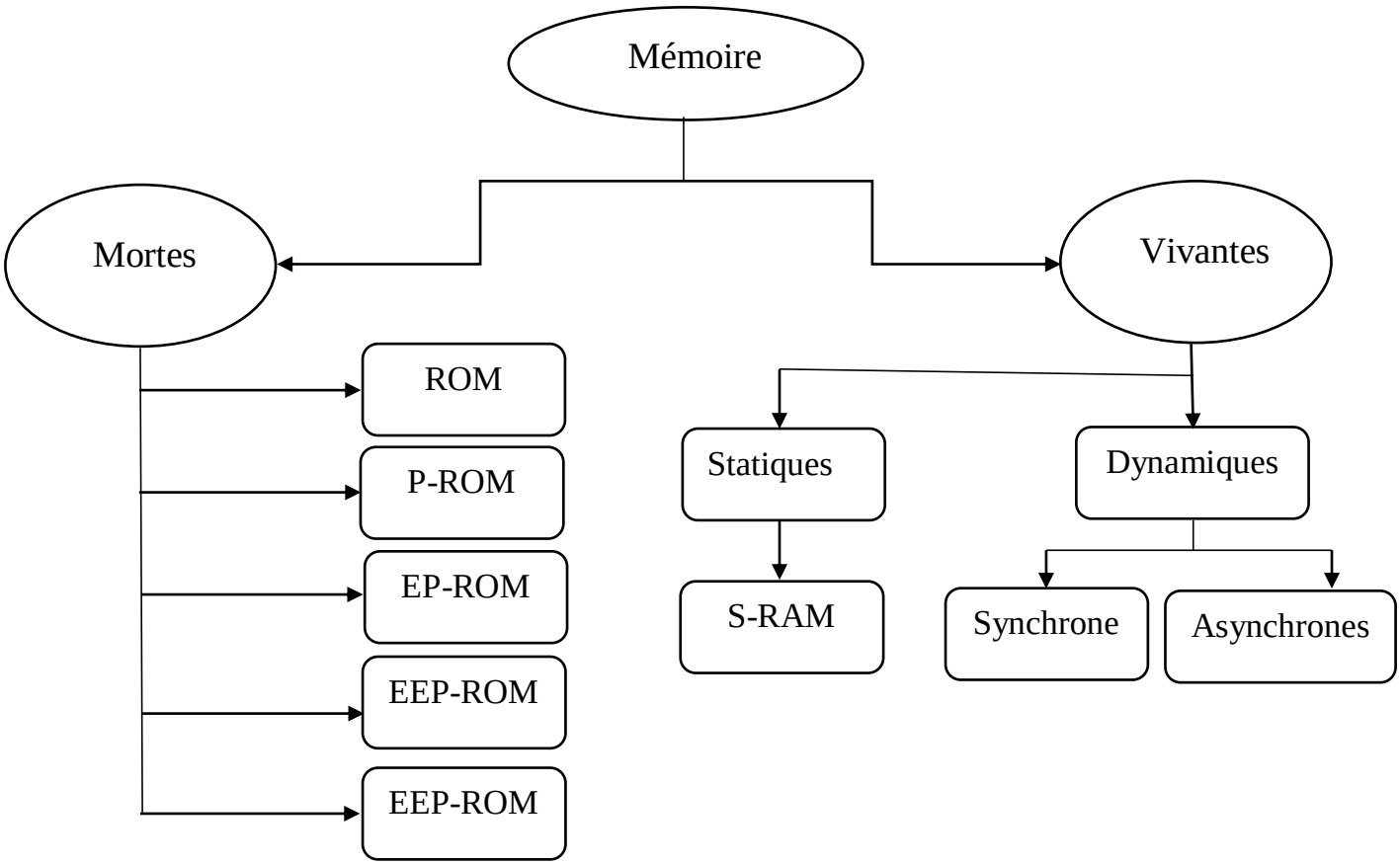
Chapitre I

I.1. Préambule :

Au fil du temps, les circuits intégrés permettent d'atteindre des très hautes performances pour une faible consommation d'énergie comparativement à des solutions logicielles. Cela s'explique par la spécialisation des fonctionnalités intégrées dans le circuit tel est le cas des circuits FPGA. Dans ce premier chapitre nous avons réalisé une étude approfondie relative aux circuits FPGA en passant en revue leur technologies, leurs différents types, leurs avantages et inconvénients ainsi que les langages de configuration et de programmation utilisés.

I.2. Les techniques de mémorisation :

Ce schéma représente les différents types de mémoires **[1]**



Figure(I.1) : Les différents types de mémoires.

L'ensemble des caractéristiques de ces mémoires sont récapitulées comme suit :

Les ROM (Read Only Memory) : Mémoires figées par concepteur à lecteur seul et non modifiables.

Les P-Rom (Programmable Read Only Memory) : Mémoire programmable une fois par l'utilisateur.

Les EP-ROM (Erasable Programmable Read Only Memory) : Mémoires programmables électriquement et effaçables par des rayon-violets au bout d'un certain temps (quelques minutes).

Les EEP-ROM (Electrically Erasable Programmable Read Only Memory) : Mémoires programmables électriquement à lecture seul, effaçables électriquement (quelques millisecondes).

Les FLASHs : Elles sont une version plus évoluée des EEP-ROM avec avantage d'être plus faciles à programmer et à effacer.

Les S-RAM (Static Random Memory) : Mémoires volatiles avec cellule de base à plusieurs transistors (accès rapide, consommation plus coûteux).

Les RAM dynamiques : Mémoires volatiles qui nécessitent rafraichissement périodique de l'information afin de la conserver avec cellules de base à un transistor.

I.3. Les circuits logiques programmables :

Les familles des circuits logiques programmables et reprogrammables sont apparues depuis les années 70 avec des noms très divers suivant les constructeurs. **[2]** **La figure(I.2)** donne une classification possible des circuits numériques en précisant où se situe chaque type de ces circuits dans cette classification

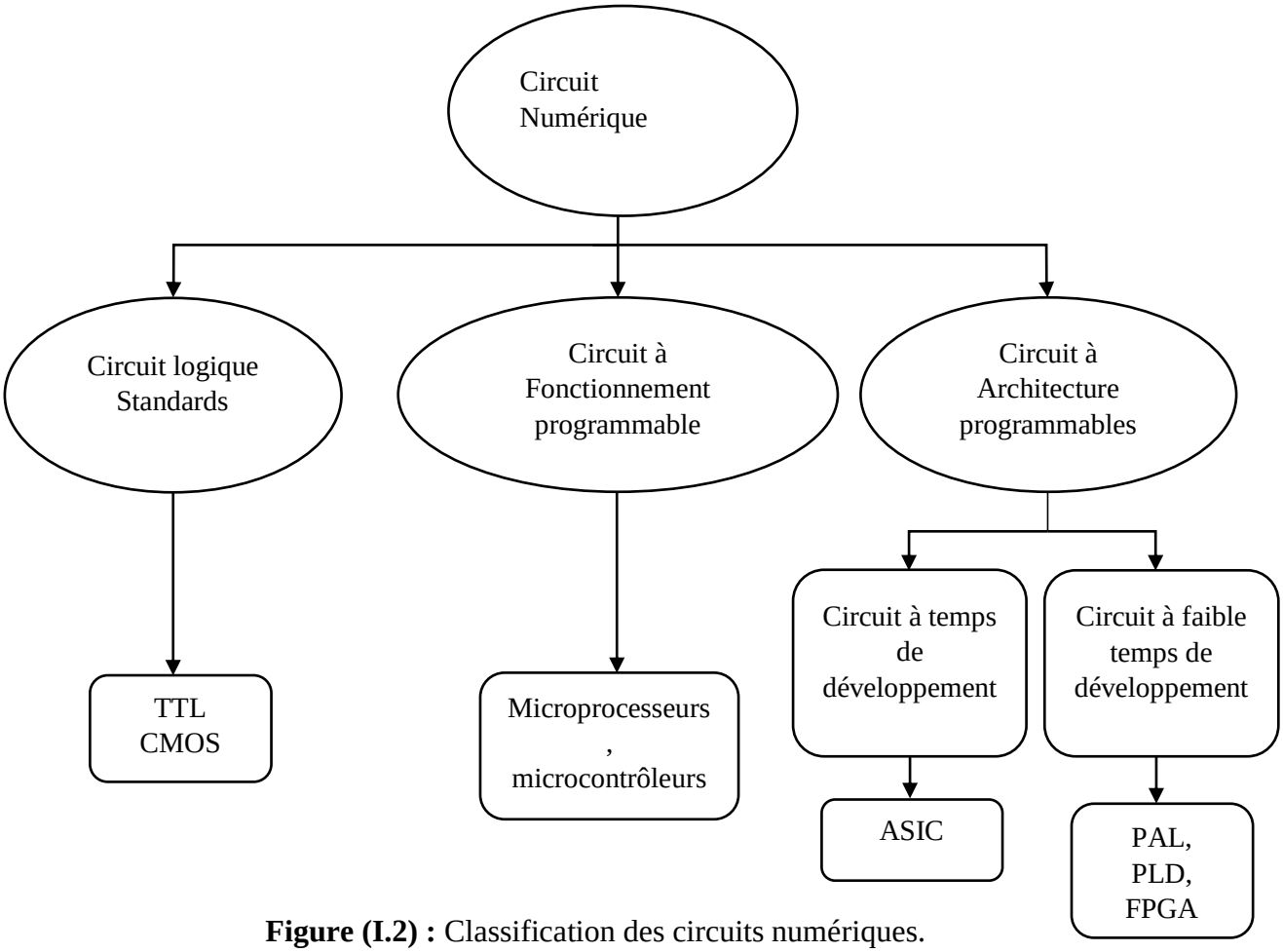


Figure (I.2) : Classification des circuits numériques.

La technologie de la logique câblée est utilisée pour créer des circuits simples moins chers en reliant des circuits logiques traditionnels tels que les TTL (Transistor-Transistor Logic) ou les CMOS (Complementary Metal-Oxide-Semiconductor). Pour des réalisations plus complexes, on utilise des circuits logiques programmables comme les FPLDs (Field-Programmable Logic Devices) tels que les PLD (Programmable Logic Devices), les FPGA (Field-Programmable Gate Arrays) ou les CPLD (Complex Programmable Logic Devices). Ces technologies permettent de réaliser des circuits logiques en configurant des matrices de portes logiques et de bascules selon les besoins spécifiques du projet.

I.4. Les circuits à architecture programmable :

I.4.1. Les ASICs (Application Specific Integrated Circuit) [3]:

Par définition, les circuits ASIC regroupent tous les circuits dont la fonction peut être personnalisée d'une manière ou d'une autre en vue d'une application spécifique, par opposition aux circuits standards dont la fonction est définie et parfaitement décrite dans le catalogue des composants.

Le concept ASIC par définition assure une optimisation maximale du circuit à réaliser. Nous disposons alors d'un circuit intégré correspondant réellement à nos propres besoins. La personnalisation du circuit donne une confidentialité au concepteur et une protection industrielle. Enfin, ce type de composant augmente la complexité du circuit, sa vitesse de fonctionnement et sa fiabilité.

Dans l'approche des circuits du type ASIC, l'inconvénient majeur réside dans le fait du passage obligatoire chez le fondeur ce qui implique des frais et un temps de développement élevés du circuit.

I.4.2. Les circuits logiques programmables (Circuit à faible temps de développement) [4] :

Les circuits logiques programmables et reprogrammables architecturalement sont classifiés en trois grandes familles les PLD, CPLD, el FPGA. Le schéma suivant les classifie ainsi que leurs technologies :

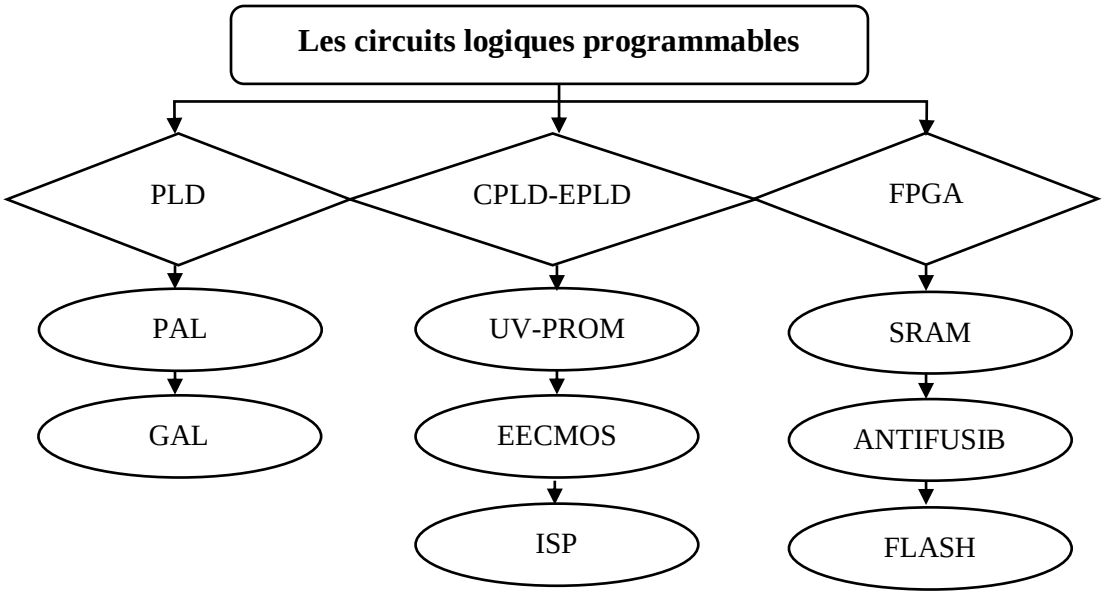


Figure (I.3) : le diagramme de différents types de circuits logiques

I.4.2.1. Les PLD (programmable logic Device) :

Est un circuit programmable qui contient des 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 des bascules dont lesquelles la fonction n'est pas défini lors de la fabrication. Suivant son architecture il peut réaliser toute une classe de fonction il est de la famille des circuits programmables qui comprend les PAL et les GAL.

- **PAL (programmable Array Logic) :** circuits logiques programmables dans lesquels seules les fonctions ET sont programmables, les fonctions OU ne le sont pas.
- **GAL (Generic Array logic) :** circuits logiques reprogrammables à technologie CMOS.

- **Structure de base d'un PLD :**

  La plupart des circuits PLDs suivent la structure suivante **(Figure (I.4)** :

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

Le deuxième et le troisième ensemble forment une matrice les interconnections entre ces matrices est maintenu par des fusibles qui les rends programmables. Lorsqu'un PLD est vierge, toutes les connexions sont assurées.

Le bloc de sortie est souvent appelé macro-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 configurations et un dispositif de rebouclement sur la matrice ET.

- Des fusibles de configuration (dans les FPGAs on utilise plutôt des cellules de commande des points de connexion)

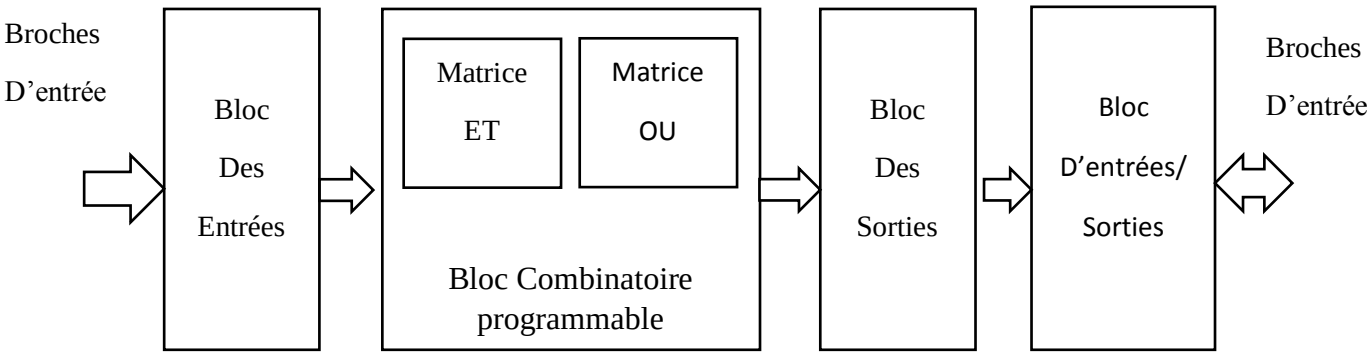


Figure (I.4) : La structure de base d'une PLDs.

I.4.2.2. CPLD ou EPLD (Erasable Programmable Logic Device) : Circuits logiques reprogrammables, qui sont des circuits effaçables par rayons ultraviolets, ils peuvent être reprogrammés.

Ce sont des circuits composés de plusieurs PALs élémentaires reliés entre eux par une zone d'interconnexion (matrice d'interconnexion) [28] [29]. 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) [31] [32]. Ces circuits ont une capacité en nombre de portes et en possibilités de configuration très supérieure à celle des PALs.

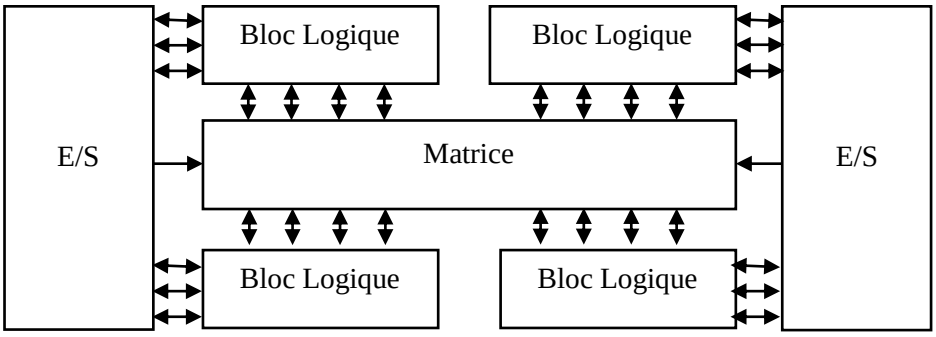


Figure (I.5) : Physionomie d'un CPLD.

I.4.2.3. FPGA (Field Programmable Gate Array) : Ces circuits sont une évolution des CPLD.

I.5. Présentation de FPGA :

Un FPGA est un circuit à densité d'intégration très élevée dont l'architecture est formée d'un ensemble de blocs logiques programmables que l'utilisateur peut les interconnecter afin de réaliser les fonctions désirées. Ces circuits comparés aux CPLDs utilisent des modules logiques beaucoup plus réduits, mais beaucoup plus nombreux. [5]

Les circuits FPGAs se composent de plusieurs types de ressources matérielles. Les principaux éléments formant un FPGA sont [6]:

- Les blocs logiques programmables ;
- Les blocs d'entrées/sorties ;
- Le réseau d'interconnexion ;

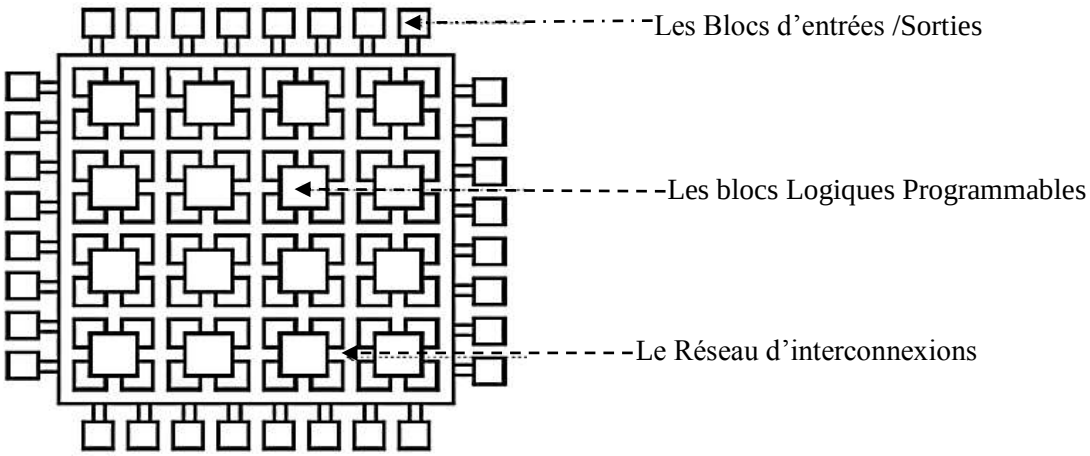


Figure (I.6) : Architecture générale d'un FPGA

Les éléments formant le circuit FPGA sont disposés par groupes et par collons, comme présente la figure ci-dessus.

Les interconnexions de ces modules ne sont pas centralisées comme les CPLDs. La physionomie du réseau de routage est vue comme une multitude de segments métalliques (lignes) pouvant être reliés entre eux ou connectés en entrée ou en sortie des blocs logiques. On distingue plusieurs types de lignes définies par leur longueur relative, on trouve :

- Les interconnexions à usage général qui sont composées de segments verticaux et horizontaux qui entourent chaque bloc logique configurable CLB et qui peuvent être reliés entre eux par une matrice de commutation.

- Les lignes directes fournissant des chemins entre les CLB adjacents et entre les CLB et les cellules d'entrée-sorties.
- Les lignes longues qui sont des lignes verticales et horizontales qui n'utilisent pas de matrice de commutation. Elles parcourent toutes les zones d'interconnexion. Elles sont utilisées pour véhiculer les signaux qui doivent parcourir de long trajet. Ces lignes conviennent pour véhiculer les signaux d'horloge.

L'ensemble de points interconnectées est appelé PIP (abréviation anglaise de Programmable Interconnect Points) et chaque point de connexion peut être réalisé selon deux techniques définissant deux classes de FPGAs (FPGAs à SRAM et FPGAs à ANTIFUSIBLE)

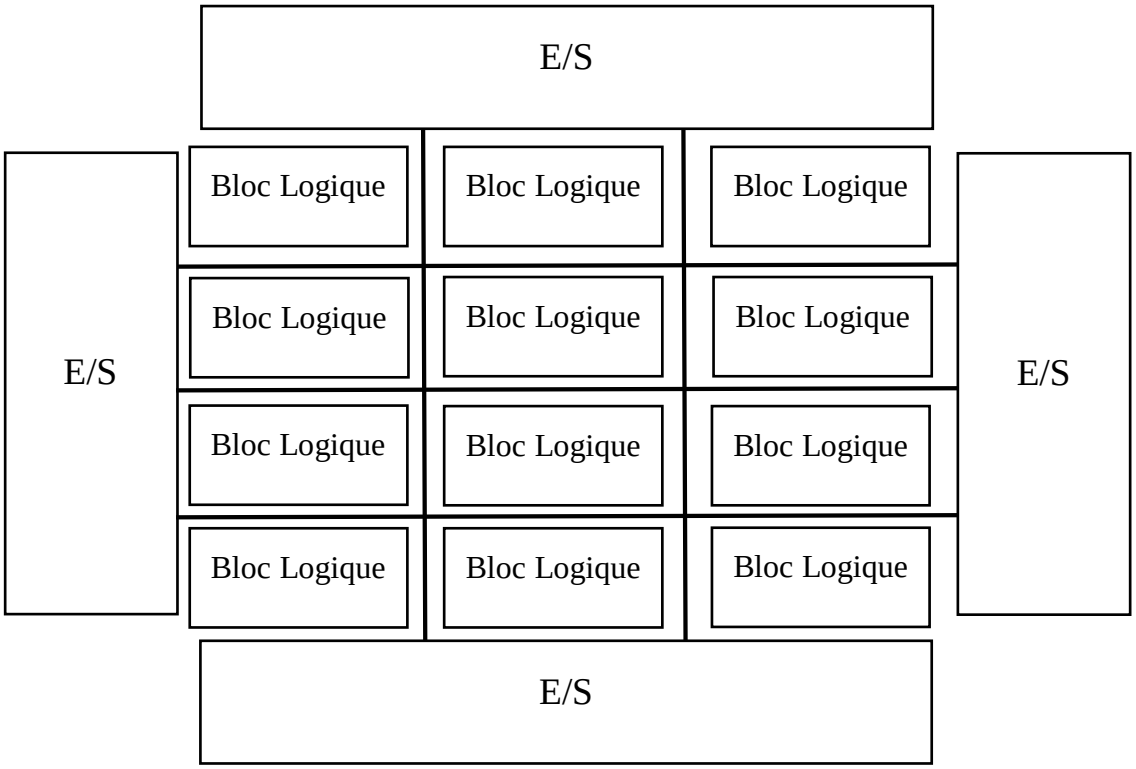


Figure (I.7) : Physionomie d'un FPGA.

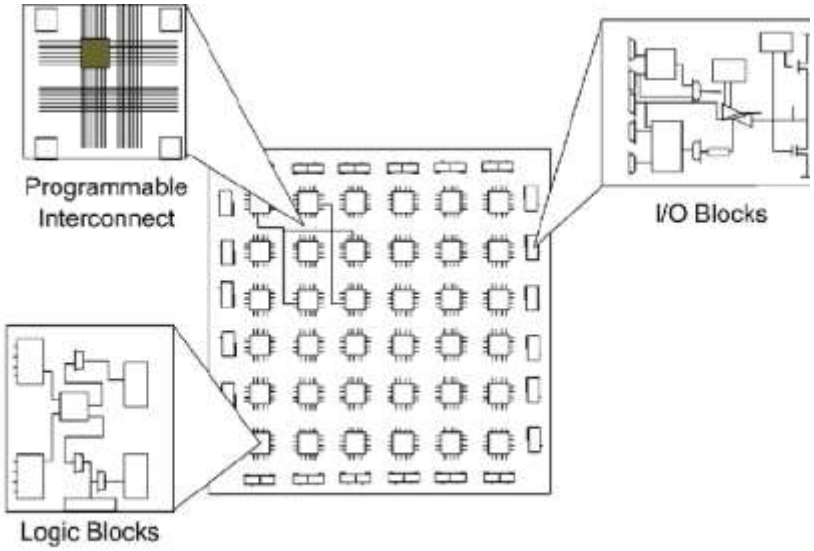


Figure (I.8) : Structure générale d'une FPGA.

I.6. Principaux fondeurs de FPGA :

Les améliorations sur les cartes FPGA ne cessant jamais, et les fabricants faisant leurs mieux pour rendre cette technologie encore plus petit sur l'échelle de taille et avoir plus de fonctionnalité. L'ensemble des principes fondeurs qui conçoivent ce type de circuits sont : Altera, Actel, Atmel, Cypress, Lattice, Minc, Quiclogic, Xilinx et d'autres.

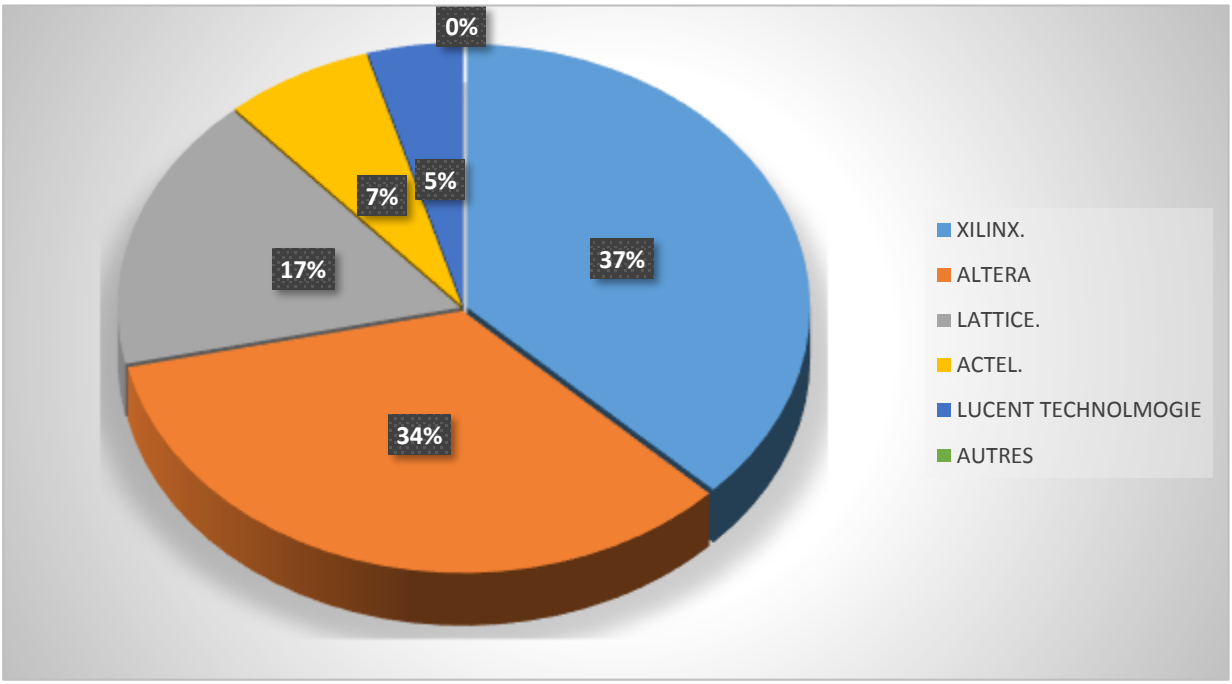


Figure (I.9) : Statistique de marchée occupée par les vendeurs des FPGAs

I.7. Classification des FPGAs :

Les FPGA ont une architecture qui diverse et on distingue deux types de FPGA qui sont : [9]

- Les FPGAs de type SRAM, appelé LCA chez XILINX, FLEX chez ALTERA.
- Les FPGAs à base d'EEPROM ou FLASH (LATTICE et ACTEL)
- Les FPGAs à anti fusibles proposé entre autre, par Texas Instrument et ACTEL.

I.7.1. Les FPGAs de type SRAM :

La structure de base d'un FPGA de type SRAM est complexe. Le point de connexion entre les différentes cellules est un ensemble de transistors MOS de commutation commandés par des cellules de mémoire vive (RAM). [9] [8]

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.

I.7.2. Les FPGAs à base d'EEPROM :

Cette technologie garde sa configuration, mais un nombre limité de configurations avec une configuration plus lente par rapport à SRAM [10]

I.7.3. Les FPGAs à anti fusibles :

Les points de connexions sont de type ROM, c'est-à-dire que la modification du point est irréalisable. Pour comprendre le mécanisme de connexion sans rentrer dans les détails des semi-conducteurs, on considère que le point de connexion est le point de rencontre de deux segments conducteurs ou lignes conductrices. Le nom ANTI-FUSIBLE vient du fait que l'état initial du fusible ou la couche isolante est présent et il n'y a pas de contact pour l'établir, il faut détruire le fusible ce qui est contradictoire au fonctionnement habituel d'un fusible. [7] [8]

I.8. Critères du choix des FPGAs :

Les puces FPGA sont utilisées pour des applications intermédiaires en puissance de calcul et en volume de pièces entre les ASIC (processeurs complètement dédiés) et les Processeurs généralistes qui exécutent un logiciel : [11]

- Les ASIC se justifient pour des volumes de pièces très élevés, car la mise au point d'un ASIC coute très cher. Non seulement il faut le programmer, mais il faut faire une conception physique de la puce, avec tous les masques de gravure. Cela crée des coûts fixes très élevés.
  Les ASIC peuvent éventuellement être justifiés par des contraintes de performances très élevées, même en faible volume, mais le prix n'est acceptable que très rarement sur des faibles volumes.
- Les FPGA permettent d'obtenir des puces spécialisées pour une application, qui sont plus efficaces que des puces généralistes de calcul ou de traitement du signal. Les FPGA ne sont pas moins chers que les processeurs généralistes (évidement cela dépend), mais reviennent beaucoup moins cher que les ASIC pour des petits nombre de pièces produites car il n'est pas nécessaire d'amortir la conception et la mise en place d'une production physique de puces spécifique, car le FPGA reste générique.
- Les puces programmables classiques, de calcul ou de traitement du signal, sont les plus polyvalentes mais cela limite leurs performances. Cependant, le choix entre un FPGA et un processeur classique n'est pas toujours aisé.

I.9. Différent domaines d'applications des FPGAs :

Les FPGA (Field-Programmable Gate Array) sont des circuits intégrés de type logique programmable. Ils contiennent des milliers de cellules identiques, assurant chacun une fonction de base identique (mémoire et logique), et qui peuvent être interconnectées à volonté. [12]

Donc, il s'agit d'un circuit logique dont les applications sont celles de l'électronique digitale ou numérique. Comme le circuit est programmable, ses fonctions (ses interconnections internes) peuvent être modifiées (reprogrammées) en cas de nécessité … contrairement aux circuits dédicacés (ASIC acronyme de l'anglais Application-Specific Integrated Circuit) qui sont des circuits intégrés spécialisés dont le contenu est figé une fois pour toutes. Il est donc

plus cher mais plus souple : il sera utilisé en développement ou pour de "petites" séries (moins de 10.000 pièces par an).

Les capacités actuelles des FPGA sont telles qu'on peut y implanter un microprocesseur. On peut donc y réaliser aussi bien une logique parallèle (plus rapide) qu'une logique séquentielle (plus complexe) ou les deux simultanément. Tout ensemble de circuits logiques peut être remplacé par un (ou plusieurs) circuits FPGA.

I.10. Application du FPGA :

Les FPGA sont utilisés dans de nombreux domaines, notamment l'aérospatiale, l'automobile, les télécommunications, la finance, la défense et bien d'autres. Dans l'aérospatiale, par exemple, les FPGA sont utilisés pour la navigation, le traitement des signaux, la gestion de l'énergie et les calculs embarqués. Dans l'automobile, les FPGA sont utilisés pour la vision par ordinateur, le traitement des signaux audio et vidéo, ainsi que pour la gestion du moteur et des systèmes de sécurité. Dans les télécommunications, les FPGA sont utilisés dans les systèmes de routage, les modems, les réseaux sans fil et les stations de base. La flexibilité et les hautes performances des FPGA en font le premier choix pour ces applications où les exigences changent rapidement et où la vitesse de traitement est critique. [13]

I.11. Les familles architecturales des FPGAs :

Un facteur important dans l'évolution des systèmes électroniques modernes est l'apparition de nouvelles architectures basées sur la programmation de circuits matériels tels que les composants programmables. Les récentes évolutions des différentes familles autorisent aujourd'hui l'intégration de systèmes de plus en plus complexes avec des contraintes de performances de plus en plus fortes. D'autre part, la flexibilité offerte par ce type de technologie fait des FPGAs (Field programmable gate arrays) une cible architecturale promise à un bel avenir. L'évaluation des performances d'une application sur une technologie reconfigurable est un problème peu étudié à ce jour. Jusqu'à présent, les chercheurs ont principalement porté leurs efforts sur l'amélioration des architectures afin de les rendre plus performantes et ainsi constituer une réelle alternative aux ASICs (Application specific integrated circuits) [14].

EXEMPLES DE FAMILLES DE FPGA :

On peut trouver defrent types des fpgas par exemples : [15]

- FPGA Altera
- FPGA Xilinx.
- FPGA Actel/Microsemi
- FPGA Achronix

La plupart des constructeurs proposent différentes familles de circuits FPGA. Tous les circuits d'une même famille ont les mêmes caractéristiques physiques et fonctionnalités et ne diffèrent que par le nombre de composants disponibles. Au sein d'une famille, les FPGA sont disponibles en plusieurs tailles. Dans les petits circuits, les différents blocs ne sont présents qu'en petites quantités à la surface du circuit, alors que dans les circuits plus grands, ils sont répliqués en grand nombre. Le coût de la puissance de traitement (calcul, stockage et communications) et des circuits augmente également avec la taille, souvent de manière non linéaire.

Chaque famille possède ses propres caractéristiques qui la rendent particulièrement adaptée à certains types d'applications. Par exemple, il existe des familles pour les performances de traitement élevées (High-End Family) et d'autres séries pour les appareils à faible coût et à faible consommation (Low-Cost and Low-Power Family). Les familles hautes performances offre des capacités de traitement très élevées (jusqu'à 2 millions de portes équivalentes et 1 à 6 GMAC·s – 1 ou milliard d'accumulations multipliées par seconde). Pour les applications où les coûts de production sont élevés, il existe des séries à faible coût avec des capacités de traitement plus modestes (centaines de milliers de portes équivalentes) et des prix bien inférieurs. Le nombre croissant de séries intermédiaires (Mid-Range Family) permet d'affiner le choix entre les performances du circuit et le prix.

Au sein d'une même famille, il existe généralement plusieurs variantes disponibles. Chaque variante prend en charge une catégorie spécifique d'applications en intégrant plusieurs blocs d'un type spécifique. Certaines variantes ont des blocs logiques plus configurables, d'autres ont plus de blocs de calcul DSP ou offrent un nombre plus élevé de blocs d'entrée/sortie (par exemple pour les applications de

télécommunications). Enfin, certains constructeurs proposent des variantes spécifiques pour des secteurs bien précis, comme le secteur aéronautique, spatial ou encore le marché automobile [15].A savoir que les deux plus grands constructeurs de FPGA dans le monde sont : XLINX et ALTERA, ci-dessous (Fig.I.10 et FIG.I.II) nous présentons respectivement les photographies de ces deux derniers :

Figure (I.10) : FPGA de **Xilinx** (modèle Spartan XC3S400) avec 400 000 portes.

Figure (I.11) : FPGA, Altera, EP4CE6E22C8N, Cyclone IV E, 6272 Cellules, 392 Blocs, EQFP 144.

I.12.Programmation et configuration des circuits FPGAs :

La programmation d'un FPGA consiste à spécifier comment ses composants internes doivent être configurés pour exécuter les fonctions souhaitées. Cette opération est souvent réalisée à l'aide de langages de description du matériel (HDL), tels que VHDL ou Verilog.

I.12.1. Le HDL (Hardware Description Language) [16] :

Est une instance d'une classe de langage informatique ayant pour but la description formelle d'un système électronique.

Il peut généralement :

Décrire le fonctionnement du circuit, Décrire sa structure, Et servir à vérifier sa fonctionnalité par simulation ou preuve formelle.

À la différence d'un langage de programmations logicielles, la syntaxe et la sémantique d'un HDL incluent des notations explicites pour exprimer le temps et la concurrence qui sont les attributs principaux du matériel.

I.12.1.1.But du HDL :

Le but des HDL est double :

1. La simulation

L'un des objectifs des HDL est d'aboutir à une représentation exécutable d'un circuit, soit sous forme autonome, soit à l'aide d'un programme externe appelé simulateur. Cette forme exécutable comporte :

- Une description du circuit à simuler,
- Un générateur de stimuli (vecteurs de test),
- Un dispositif implémentant la sémantique du langage et l'écoulement du temps.

Il existe deux types de simulateurs :

1/ à temps discret, généralement pour le numérique,

2/ à temps continu pour l'analogique.

Remarque :

Des HDL existent pour ces deux types de simulations

2. La synthèse

En n'utilisant qu'un sous-ensemble d'un HDL, un programme spécial appelé synthétiseur peut transformer une description de circuit en une netlist de portes logiques ayant le même comportement que le circuit de départ. Le sous-ensemble du langage utilisé à ce propos est alors dit synthétisable.

I.12.1.2. Les Principaux Langages HDLs:

ABEL "Advanced Boolean Expression Language", un langage propriétaire développé par Data I/O Corporation, maintenant possédé par Lattice;

AHDL ("Altera HDL", langage propriétaire d'Altera pour la programmation de leurs FPGA) langage propriétaire essentiellement structurel proche d'ABEL ;

Verilog qui mélange description structurelle et algorithmique ;

VHDL légèrement plus abstrait que Verilog qui est inspiré de ADA ;

SystemC utilisant le C++ et qui permet de modéliser les interactions logiciel/matériel ;

Confluence, langage déclaratif GPL pouvant générer du Verilog, du VHDL, des modèles exécutables en C et des modèles de vérification formelle ;

CUPL langage propriétaire de Logical Devices; [16]

I.12.2. VHDL (VHSIC Hardware Description Language):

Le langage de description VHDL est ensuite devenu une norme IEEE en 1987. Révisée en 1993 pour supprimer quelques ambiguïtés et améliorer la portabilité du langage, cette norme est vite devenue un standard en matière d'outils de description de fonctions logiques.

A ce jour, on utilise le langage VHDL pour : [16]

- Concevoir des ASIC
- Programmer des composants programmables du type PLD, CPLD et FPGA.
- Concevoir des modèles de simulations numériques ou des bancs de tests. [17]

I.12.2.1. But du langage vhdl :

Le but des langages de description de matériel tels que VHDL est de faciliter le développement

Décrire des circuits numériques en fournissant une méthode rigoureuse de description de leurs fonctionnements et l'architecture de circuit requise.

L'étape suivante consiste à synthétiser cette description matérielle pour obtenir un composant

Utiliser des éléments logiques spécifiques (portes logiques, registres, bascules) pour exécuter la fonction requise

Ceux-ci seront placés directement sur le Transistors (dans le cas des ASIC) ou composants programmables basés sur FPGA.

VHDL a une double fonctionnalité (simulation et synthèse) et n'est qu'une partie de VHDL sont synthétisables, les autres n'existent que pour faciliter la simulation (écriture de modèles comportement et banc de test).

I.12.2.2. Environnement du VHDL :

- **plateforme : PC/Station**
- Éditeurs de texte.
- Compilateur.
- Synthèse.
- Simulateurs.
- Interfaces graphiques (courbes stimulés & résultats).
- bibliothèques : IEEE.
- Paquetages IEEE.
- Paquetages standards.
- Paquetages propriétaires
- **Notion de projets**

Un analyseur (transformation du code VHDL en éléments logiques de base).

Un Place et Route (placement et connection des éléments logiques dans le composant).

Bibliothèque communes (IEEE) et bibliothèques utilisateur (WORK).

Remarque :

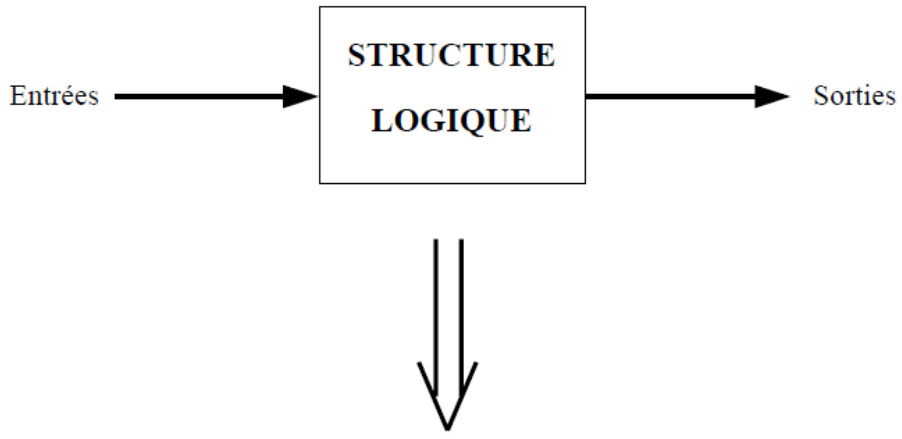
Beaucoup de logiciels prennent en charge toutes ces fonctionnalités :
Xillinx ISE, Altera Quartus, Lattice ISP Lever, Altium Designer, etc...

Une description VHDL : est un ensemble de fichiers d'extension. Vhd permettant : Édition des fichiers VHDL, Faire référence aux bibliothèques et paquetages nécessaires (IEEE, WORK), Compilation, Simulation, Synthèse et implémentation sur les ASICs ou les FPGAs. [16]

I.12.2.3. Les unités de compilation VHDL :

L'analyse d'un modèle VHDL peut s'effectuer sur des parties du code ou "unités de compilation". Il existe 5 types d'unités de compilation [16] :

- Entité (vue externe)
- Architecture (vue interne)
- Configuration (couple : entité-architecture)
- Paquetage (déclarations globales, …)
- Corps du paquetage (sous-programmes, …



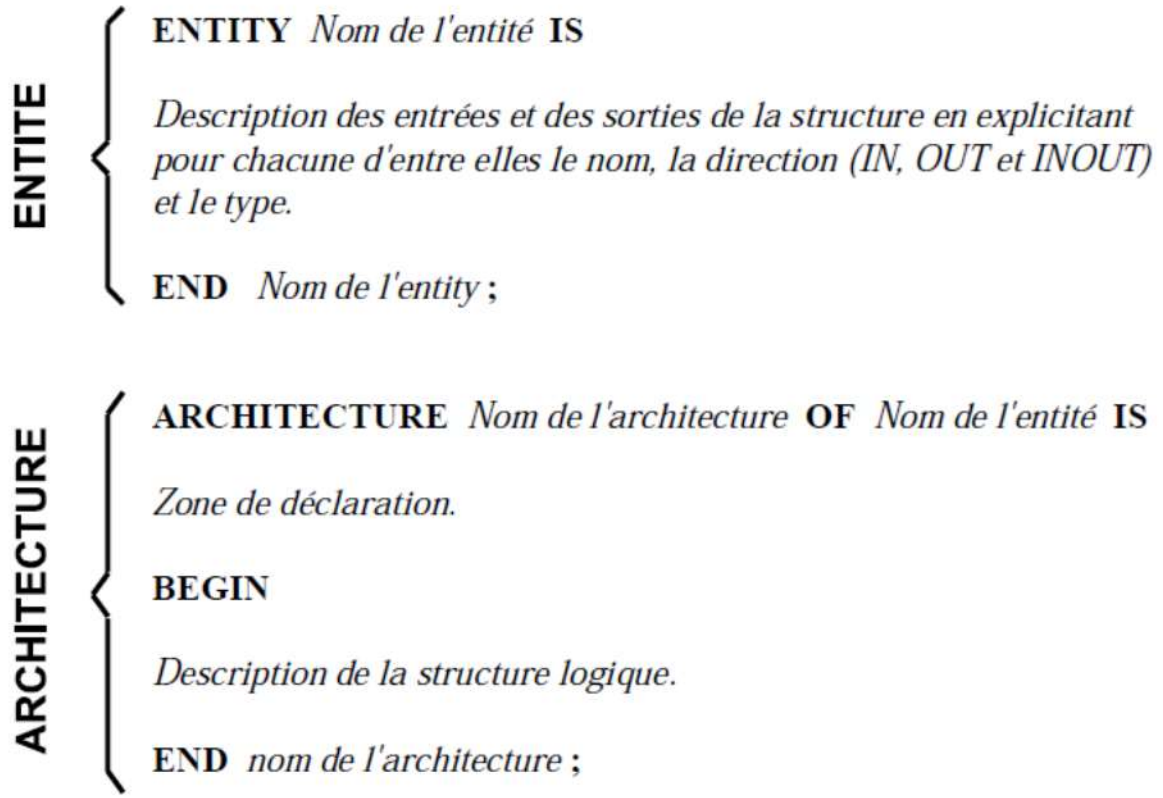


Figure (I.12) : Structure logique d'un programme VHDL [17]

- **Circuit configurable :**

Un circuit configurable est un circuit programmé et chargé par différentes données, où les interconnections d'un FPGA sont programmées afin de donner un fonctionnement spécifique à un tel circuit [18].

- **Circuit reconfigurable :**

C'est le même principe de configuration, sauf que cette fois on reconfigure le circuit FPGA une deuxième fois pour l'utiliser dans une autre fonction comme on peut garder la dernière configuration du circuit. On peut même effacer cette configuration et on reconfigure le circuit une nouvelle fois [18].

- **Circuit partiellement reconfigurable :**

Un dispositif ou un circuit est défini comme partiellement reconfigurable (dans la littérature on trouve aussi la terminologie Run Time Reconfiguration RTR globale) s'il est possible de le reconfigurer sélectivement, tandis que l'état de repos du reste du dispositif est inactif, mais il conserve son information configurée. Encore, il ne semble pas y avoir n'importe quel dispositif sur le marché qui soit partiellement reconfigurable, mais non aussi dynamiquement reconfigurable, La reconfiguration partielle permet de rendre un FPGA effectif, multiple fonctions, et change de fonctions pendant le fonctionnement du système [18]

.

- **Circuit dynamiquement reconfigurable :**

Un circuit FPGA est reconfigurable dynamiquement (dans la littérature on trouve la terminologie Run Time Reconfiguration RTR locale) s'il peut être partiellement reconfiguré durant son fonctionnement, c.-à-d. une partie du circuit correspondant à certaines fonctions logique et leur interconnections peut être changée sans affecter le fonctionnement de la logique restante. On peut aussi parler de reconfiguration dynamique dans le cas où plusieurs circuits FPGAs sont connectés entre eux et il s'agit de reconfigurer un seul composant FPGA tout en maintenant les autres circuits en fonctionnement [18].

.

I.13. Avantages et Inconvénients des FPGAs :

➢ Avantage :

- Performance.
- Délai de mise sur le marché.
- Coût peu élevé.
- Stabilité.
- Maintenance à long terme.
- Reprogrammable.

➢ Inconvénients :

- Prix unitaire trop élevé pour les très grandes séries.
- Performance électriques inférieures aux puces spécialisées (notamment en fréquence).
- Faible taux d'utilisation du circuit.
- Performances non optimisées.

Remarque :

Dans notre travail nous utilisons la carte **FPGA ALTERA.**

I.14. Architecteur de la carte FPGA :

Cette carte dispose des éléments suivants :

- Un FPGA Altera Cyclone II 2C35
- Un circuit de mémoire Flash série Altera EPCS16
- Une interface USB-Blaster pour la programmation et le contrôle des API utilisateur, les
- modes de programmation JTAG et Active Serial (AS) sont supportés
- 512-Kbyte de SRAM
- 8-Mbyte de SDRAM
- 4-Mbyte de mémoire Flash (1 Mbyte sur certaines cartes)
- Un connecteur SD Card
- 4 boutons poussoirs
- 18 interrupteurs
- 18 LEDs utilisateur rouges
- 9 LEDs utilisateur vertes
- Un oscillateur 50-MHz et un oscillateur 27-MHz comme sources d'horloges
- Un CODEC 24-bit CD-quality avec jacks entrée ligne, sortie ligne et microphone-in.
- Un CNA VGA (10-bit high-speed triple DACs) avec connecteur VGA-out
- Un décodeur TV (NTSC/PAL) avec connecteur TV-in
- Un contrôleur 10/100 Ethernet avec connecteur
- Un contrôleur USB Hôte/Esclave avec connecteurs USB type A et type B
- Un E/R RS-232 avec connecteur 9-broches
- Un connecteur PS/2 souris/clavier
- Un E/R IrDA
- 2 connecteurs d'extension 40-broches avec diodes de protection

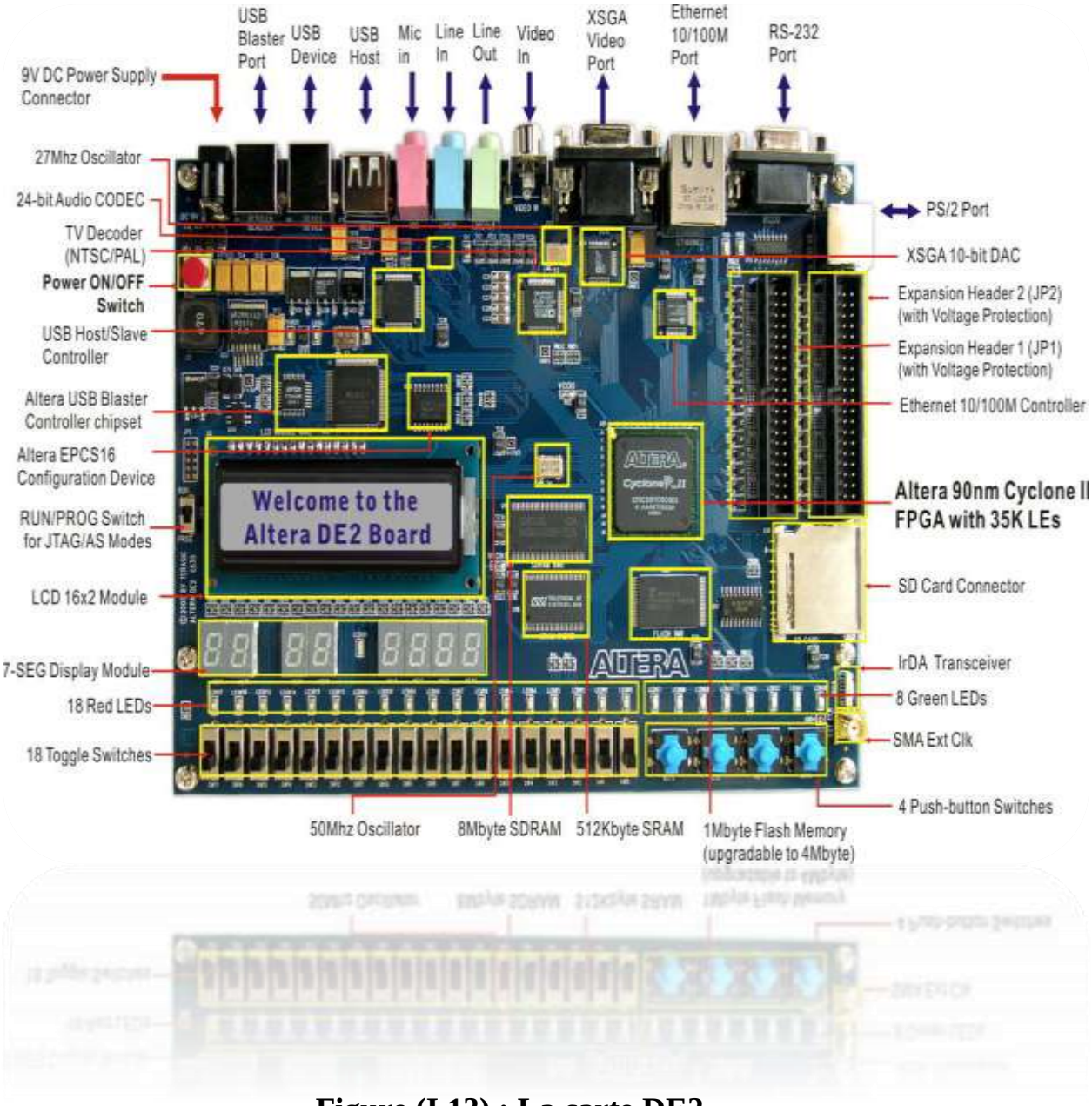


Figure (I.13) : La carte DE2.

Discussion :

Cette partie est portée sur les généralités sur les circuits logiques programmables où on s'est intéressé plus particulièrement au circuit FPGA ainsi qu'a ses propres caractéristiques.

Chapitre II

II.1. Préambule :

Bien que les premiers moteurs pas à pas remontent aux années 1930, Ceux-ci ne sont vraiment développés que vers 1970 grâce au développement conjugué de l'électronique de puissance et, surtout, grâce à l'apparition de l'électronique numérique à forte intégration.

Le moteur pas à pas est un convertisseur électromécanique qui assure la transformation d'un signal électrique impulsionnel en un déplacement mécanique (angulaire ou linaire). Sa structure de base se présente sous la forme de deux pièces séparées mécaniquement, le Stator et le Rotor. L'interaction électromagnétique entre ces deux parties assure la rotation.

Dans cette première partie de l'étude nous présenterons les principales caractéristiques des différents types de moteur pas à pas, et leurs principes de fonctionnement ainsi son rôle dans les imprimantes 3D.

II.2. Les Moteurs Pas A Pas :

Les moteurs pas à pas sont des moteurs spéciaux, composés simplement d'un stator réunissent des pièces polaires et des bobinages, et utilisés pour commander avec grande précision le déplacement et la position d'un objet.

Comme leur nom l'indique, ces moteurs tournent par incrément discret. Chaque incrément de rotation est provoqué par une impulsion de courant fournie à l'un des enroulements du stator

Le moteur pas à pas est l'organe de positionnement et de vitesse travaillant généralement en boucle ouverte.

Les moteurs pas à pas sont très utilisés dans toutes les applications mécaniques ou l'on doit contrôler simplement la position ou la vitesse d'un système en boucle ouverte. Ces moteurs sont par exemple utilisés dans les imprimantes jet d'encre ou laser, pour positionner les têtes d'impression ou pour l'avancée du papier.

II.2.1. Les Déférents types des moteurs pas à pas :

Les moteurs pas à pas actuellement disponibles peuvent être classés en fonction du phénomène physique qui est à l'origine du couple. On distingue trois types principaux de moteurs [19] :

II.2.1.1. Moteurs à aimant permanent : [20] Il utilise le principe de l'action d'un champ magnétique sur un aimant (M.P), La construction des moteurs pas à pas à aimants permanents obéit aux règles générales suivantes :

- Le stator, le plus souvent à pôles saillants, est assemblé à partir de tôles magnétiques en fer silicium, isolées par oxydation ou par un vernis.
- L'entrefer entre plots et aimant(s) est maintenu aussi petit que possible.
- Le rotor est constitué soit entièrement par un aimant permanent cylindrique dans lequel on usine une ouverture pour passer l'arbre, soit par une carcasse en fer le plus souvent feuilletée sur laquelle sont frettés les aimants

II.2.1.2. Moteurs à reluctance variable : [20] Il utilise le principe du flux maximum (M.R.V), Les circuits magnétiques du rotor et du stator sont assemblés à partir de tôles magnétiques de haute perméabilité (fer-silicium ou même fer-cobalt). Certaines machines, destinées au positionnement ou a rotation très lente, peuvent avoir un rotor ou/et un stator en fer massif.

II.2.1.3. Moteur hybrides : Les moteurs pas à pas hybrides réunissent, au moins en partie, les avantages des moteurs pas à pas à réluctance variable et à aimants permanents, à savoir :

- un grand nombre de pas par tour.
- une fréquence propre mécanique importante.
- un couple massique élevé.
- un amortissement interne important.
- une mémoire de position.

Dans sa configuration de base le moteur pas à pas hybride comporte un stator en fer feuilleté à plots saillants et deux couronnes rotoriques dentées en matériau ferromagnétique, géométriquement identiques et réunies par un aimant permanent cylindrique magnétisé axialement. Les lignes de champs de l'aimant se ferment à travers les dents du rotor [20]

Dans notre étude, nous considérons un moteur à aimant permanent.

II.2.2. Moteurs à aimant permanent :

Les moteurs pas à pas à aimant permanent sont dérivés des moteurs synchrones à aimant permanent classique, ce sont des moteurs dont le rotor est constitué d'un aimant bipolaire ou d'un aimant multipolaire. [21]

II.2.2.1. Constitution :

Le moteur pas à pas à aimant permanent est composé de deux parties [22] :

Rotor : qui est la partie mobile constitué par un aimant en ferrite ayant une perméabilité faible, la reluctance du circuit faible, la reluctance du circuit magnétique ne varie pas quand l'aimant tourne.

Stator : qui est la partie fixe comporte des pôles électromagnétiques A, B, A`, B` dont on peut fixer la polarité selon le sens du courant dans les bobines.

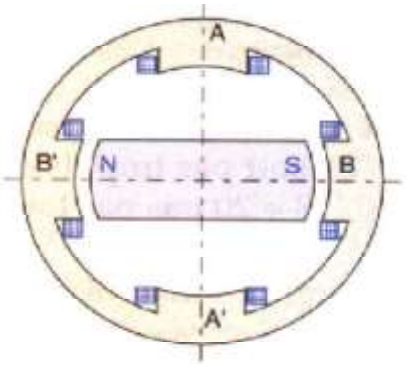


Fig. II.1 : Un moteur à aimant permanent

II.2.2.2. Principe de fonctionnement :

Au repos, le rotor aimant N.S. se place en face d'une paire de dents du stator. Quand on alimente les bobines, le rotor se place en face des bobines alimentées de telles façon que le flux qui le traverse soit maximal. En alimentant successivement les phases (A-A`), (B-B`), (A`-A), (B`-B) on obtient quatre positions stables.

La rotation dans le sens inverse est obtenue en inversant la séquence de commutation des phases de rotor.

II.2.2.2.1. Moteur à aimant permanent bipolaire :

Le courant de commande est bidirectionnel et l'avance d'un pas s'effectue par une séquence de commutation des enroulements statorique [23].

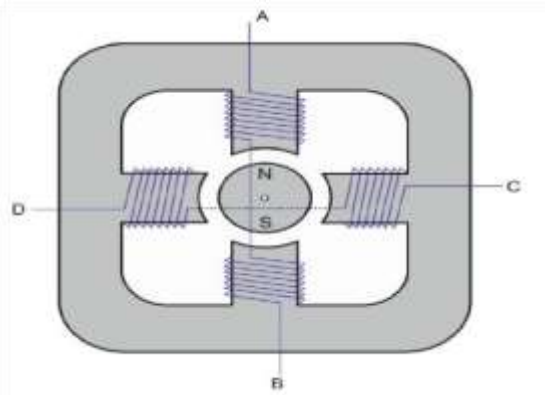


Figure II.2 : Représentation schématique d'un moteur bipolaire

Pour ce type de moteur nous avons trois possibilités de commande :

II.2.2.2.1.1. Fonctionnement à pas complet

La première consiste à alimenter les enroulements en suivant la séquence A vers B/C vers D/B vers A/D vers C (BA est les mêmes enroulements que AB mais alimenté par un courant de polarité inverse). Par la suite nous simplifierons la notation pour une meilleure correspondance avec les chronogrammes des phases en indiquant uniquement la phase qui est alimentée par un courant "positif" Soit A B C D.

Cette séquence est connue sous le nom de " one phase on full step" (traduisez phase par phase ou une phase à la fois en pas entier). A tout moment il n'y a qu'une seule phase d'alimentée et nous travaillons en mode pas entier.

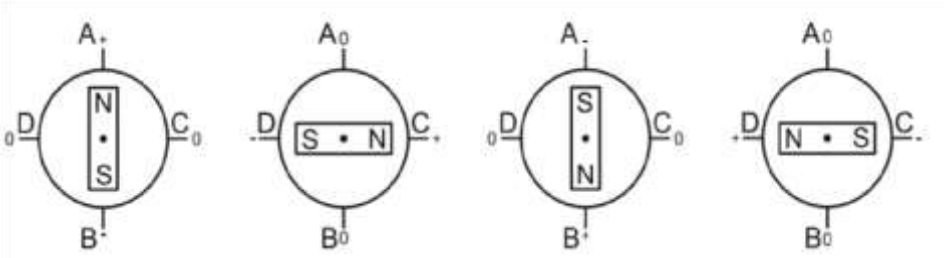


Figure II.3 : Fonctionnement à pas complet

II.2.2.2.1.2. Fonctionnement avec couple maximal :

La seconde possibilité est d'alimenter une paire de phase en même temps de façon à ce que le rotor se positionne entre deux pôles.

Appelé "two-phase-on full step" (deux phases à la fois en pas entier) ce mode de commande est celui qui procure le couple le plus élevé.

La séquence sera donc AC/CB/BD/DA.

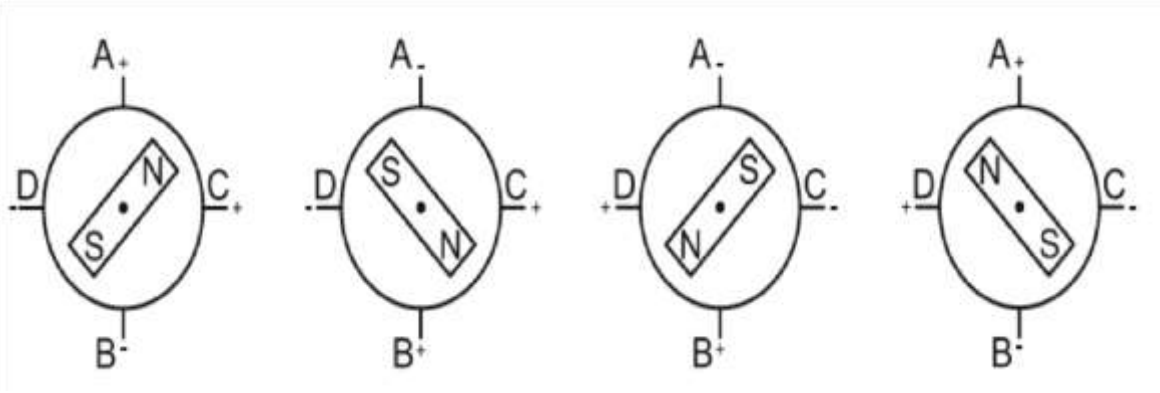


Figure II.4 : Fonctionnement avec couple maximal

II.2.2.2.1.3. Fonctionnement à demi-pas :

La troisième option est un mélange des deux premières puisque l'on alimente tour à tour le moteur sur une phase puis deux puis une … cette séquence connue sous le nom de mode demi-pas procure affectivement une division par 2 de l'angle d'avance d'un pas mais aussi un couple moins régulier.

La séquence qui en découle est la suivante : A/AC/C/CB/B/BD/D/DA.

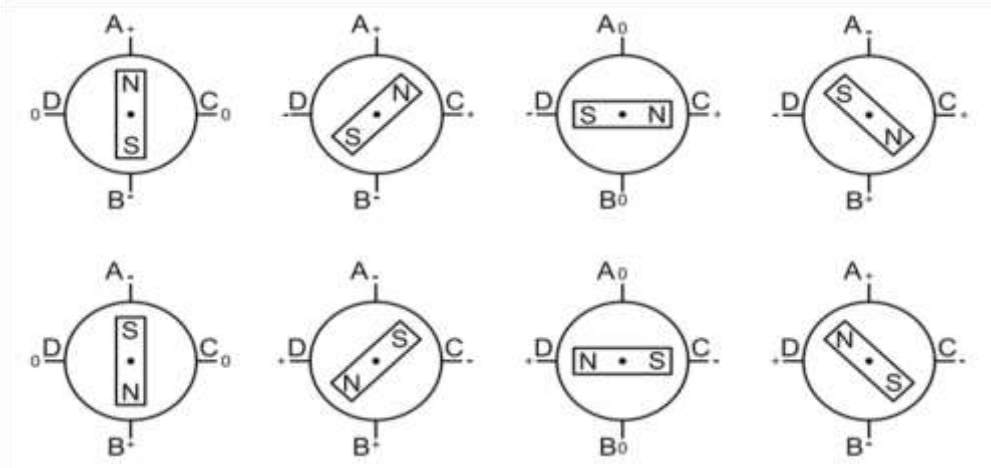


Figure II.5 : Fonctionnement à demi-pas

II.2.2.2.2. Moteur à aimant permanent unipolaire

Les moteurs unipolaires se différencient par le fait qu'ils sont à double bobinage. Le double bobinage est utilisé pour l'inversion du flux statorique et le moteur se commande de la même manière qu'un bipolaire excepté qu'un seul transistor pour chaque enroulement suffit dans l'étage de puissance (soit quatre Darlington pour un moteur ou un réseau de 4 transistors- voir ULN 2075B).

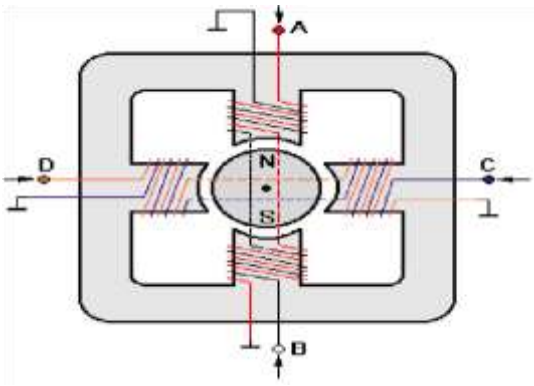


Figure II.6 : Représentation schématique d'un moteur unipolaire

Simplement, les moteurs unipolaires sont plus chers car leur fabrication réclame un double bobinage. De plus, pour une taille donnée, ce type de moteur à un couple plus faible à cause des enroulements qui sont plus fins.

Il fût une époque où les moteurs unipolaires étaient intéressants pour les concepteurs parce qu'ils simplifiaient l'étage de commande électronique. Maintenant, grâce aux circuits de commande (push pull monolithique) du genre L298, les moteurs bipolaires sont devenu populaires et d'une utilisation courante.

Tous les moteurs à aimant permanent souffrent des oscillations (et des harmoniques qui s'ensuivent) générées par le rotor qui limitent la vitesse de rotation.

Quand des accélérations et des vitesses plus élevées sont nécessaires on utilisera de préférence les moteurs à reluctance variable [22].

II.2.2.2.2.1. Fonctionnement d'un moteur pas-a-pas unipolaire

On partira du principe que la rotation d'un moteur pas à pas s'effectue en 4 étapes, dans la réalité, le moteur est constitué d'une succession d'alternance de pôles : ainsi, l'axe du modèle dont nous disposons dans notre réalisation fait un tour complet en 48 pas (un pas correspond donc à 360°/48 = 7,5°).

Dans les schémas, la flèche noire représente l'aiguille d'une boussole qui serait disposé en place et lieu du rotor ; elle indique l'orientation du champ magnétique (elle pointe vers le nord, qui attire donc le pôle Sud du rotor) et se décale alors d'un quart de tour à chaque étape [24].

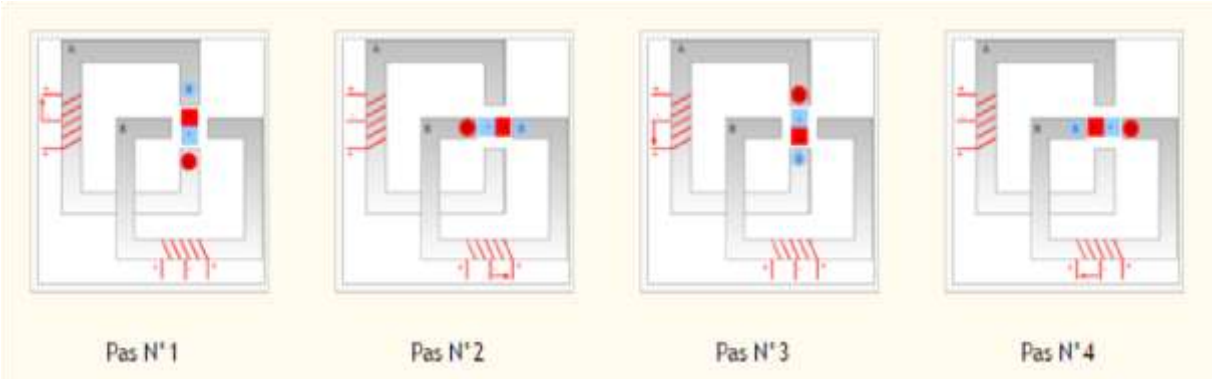


Figure II.7 : fonctionnement d'un moteur pas-a-pas unipolaire

II.2.2.2.2.2. Etats successifs des phases du moteur unipolaire :

La table de vérité B.2.2.1 résume les états successifs des différentes phases : l'état logique indique si la phase est alimentée (''1'') ou non (''0'').

Numéro de l'étape :	1	2	3	4
Bobinage 1, phase 1	0	1	1	0
Bobinage 1, phase 2	1	0	0	1
Bobinage 2, phase 1	1	1	0	0
Bobinage 2, phase 2	0	0	1	1

Table II.1 : Table Etat successifs des phases

II.2.2.2.2.3. fonctionnement du moteur unipolaire en mode demi-pas :

Le moteur de notre schéma effectue une rotation en quatre pas. Il se caractérise par un fonctionnement dit ''par pas'' il existe également un mode de fonctionnement par ''demi pas'' : il consiste a intercaler entre deux étapes, une période au cours de laquelle l'on coupe l'alimentation du bobinage du stator dont l'alimentation s'apprête à changer de sens (elle passe donc par zéro) : durant cette nouvelle étape .le rotor tourne d'un demi pas (45°) en s'alignant table de vérité des phases est représentée en table suivent :

Numéro de l'étape :	1	2	3	4	5	6	7	8
Bobinage 1, phase 1	0	0	1	1	1	0	0	0
Bobinage 1, phase 2	1	0	0	0	0	0	1	1
Bobinage 2, phase 1	1	1	1	0	0	0	0	0
Bobinage 2, phase 2	0	0	0	0	1	1	1	0

Table II.2 : Etat successifs des phases lors de l'utilisation des demi-pas.

Le mouvement s'effectué à la suite d'une inversion du champ magnétique en alimentant l'une ou l'autre des phases d'un bobinage a point milieu ; seule une moitié du bobinage et donc utilisée a un instant donné. Un autre type de moteur. Dit moteur à deux phases, permet d'obtenir un couple plus important : son principe consiste à utiliser un bobinage sans point milieu, et à faire circuler le courant dans un sens ou dans l'autre…

II.2.3. La Différence entre ces trois types de moteurs :

Types de moteurs	Moteur à aimant permanant	Moteurs à reluctance variable	Moteur hybride
Résolution (nombre de tour /pas	Moyenne	Bonne	Elevée
Couple moteur	Elevée	Faible	Elevée
Sens de rotation	Il dépend : − Du sens du courant pour le moteur bipolaire. − De l'ordre d'alimentation des bobines.	Il dépend uniquement : − De l'ordre d'alimentation des bobines	Il dépend : − Du sens du courant − De l'ordre d'alimentation des bobines.
Fréquence de travaille	Faible	Grande	Grande

Tableau II.3 : Comparaison des trois de moteurs pas à pas

II.3. Les imprimantes 3D :

II.3.1. Présentations :

II.3.1.1 Définition de l'impression 3D :

L'impression 3D est une technique de fabrication dite additive qui procède par ajout de matières, contrairement aux techniques procédant par retrait de matière comme l'usinage. L'impression 3D permet de réaliser des objets usuels, des pièces détachées ou encore des prototypes destinés aux essais. Le point de départs est un fichier informatique représentant l'objet en trois dimensions, décomposé en tranches. Ces informations sont envoyées à une imprimante 3D qui va réaliser la fabrication par ajout de couches successives. [25].

II.3.1.2. Méthodes d'impression :

II.3.1.2.1. dépôt de fil FDM (Fused Deposition Modeling) :

Il existe beaucoup de méthodes différentes d'impression 3D. La plus répandue dans le commerce est la méthode dite du « **dépôt de fil FDM (Fused Deposition Modeling)** ».

Comme son nom l'indique, cette technique repose sur un dépôt de filament qui s'effectue en couches successives très fines. Chauffé à haute température, ce filament, le plus souvent en **« acide polylactique » ou PLA**, va atteindre son point de fusion et sera déposé par une buse sur le plateau de l'**imprimante 3D**. Le **diamètre** de ce filament varie selon le diamètre de la buse et permet d'obtenir un objet plus ou moins détaillé (avec une qualité d'impression variable en somme). **Plus le diamètre est faible, plus l'objet sera de qualité mais mettra du temps à s'imprimer.**

Ainsi, ce filament très fin sort de la buse et vient se placer sur ceux déjà déposés. Les **deux filaments fusionnent** entre eux, ce qui donne au final un objet en un seul bloc. Cette technique d'impression 3D est déposée par l'entreprise **Stratasys**, qui l'a inventée. [26]

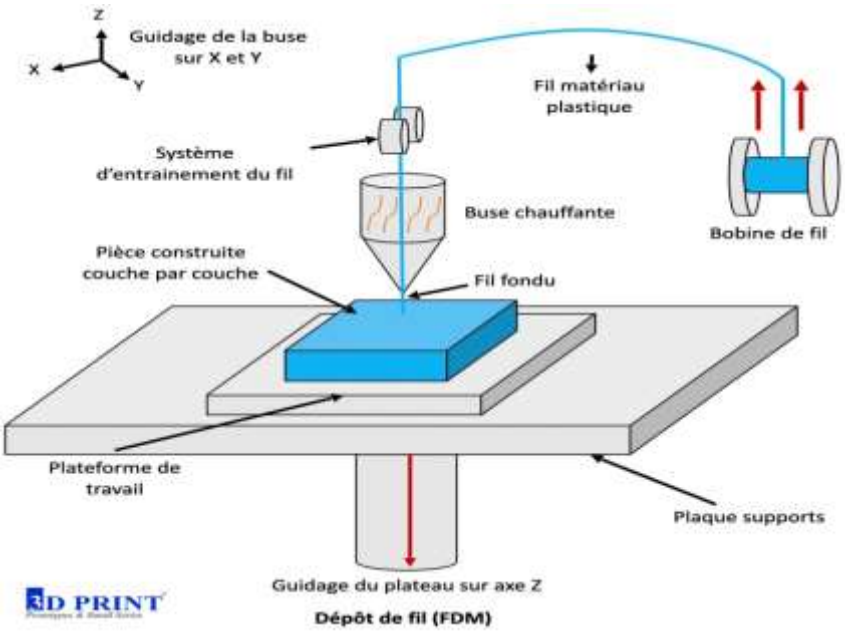


Figure II.8 : Procédé de Fused Deposition

II.3.1.2.2. Stéréolithographie (SLA, stereolithograph apparatus) :

La **stéréolithographie** (SLA, stereolithograph apparatus) est également répandue et commune. Le principe général est identique à la **méthode FDM**, si ce n'est l'utilisation d'ultra-violets dans un liquide plastique monomère. Entre chaque couche de résine déposée, une **lampe ultra-violette**, masquée par un écran LCD dessinant la forme de la couche à imprimer, va flasher la résine pour la traiter. La conséquence est un durcissement de la résine qui sera alors prête à accueillir la seconde couche et ainsi de suite. Son proche cousin est le DLP (Digital Light Processing ou Traitement Numérique de la Lumière) qui utilise non pas les UV mais une ampoule inactinique (comme dans les labos photo argentiques en résumé). [26],

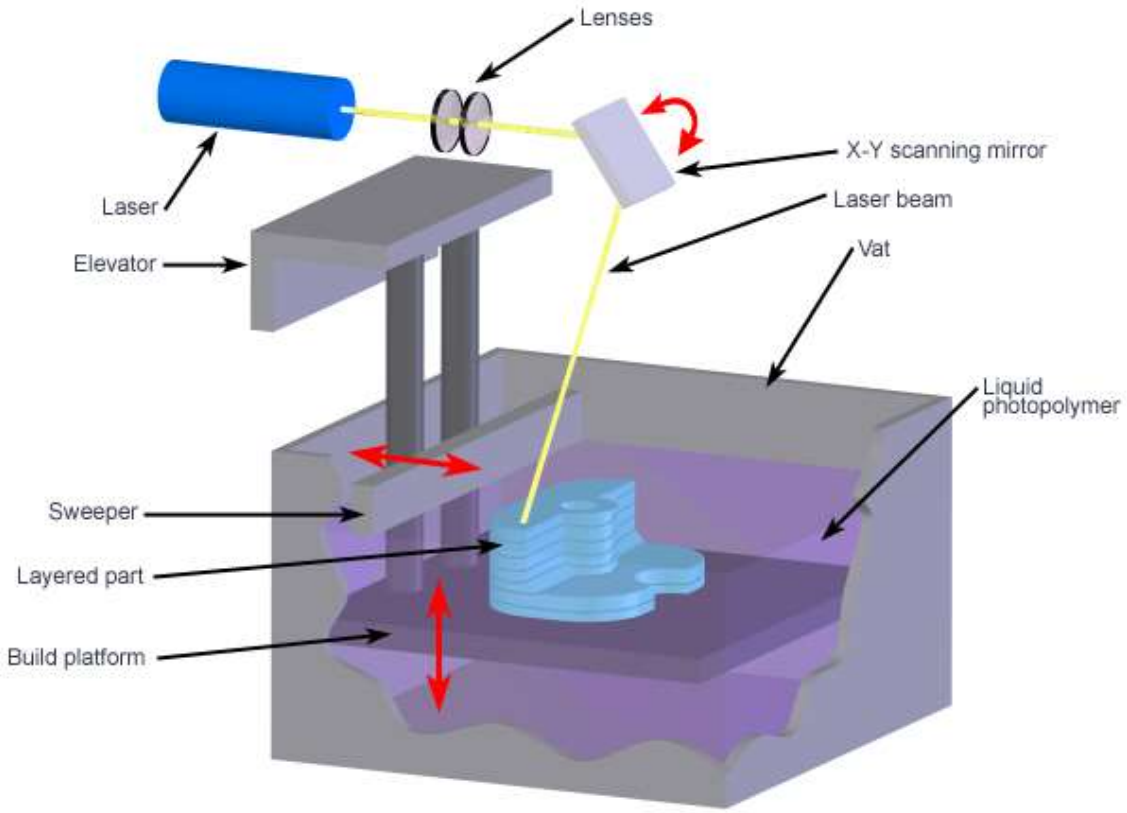


Figure II.9 : Procédé de Stéréolithographie

II.3.1.2.3. SLS (Selective Laser Sintering) ou « Frittage Sélectif Laser » :

Très proche de la **stéréolithographie,** la **SLS** (Selective Laser Sintering) ou **« Frittage Sélectif Laser** » est une technique utilisée dans les imprimantes 3D de grande taille (destinées aux industries). Au lieu de résine, la matière employée est une **poudre**. Entre chaque couche, un laser solidifie la poudre appliquée et la fixe aux couches précédentes par frittage (la poudre chauffe sans entrer en fusion et se soude à la couche inférieure). Une fois les premières couches soudées par ce procédé, une nouvelle couche de poudre est étalée et le processus continue jusqu'à ce que la pièce soit achevée. [26]

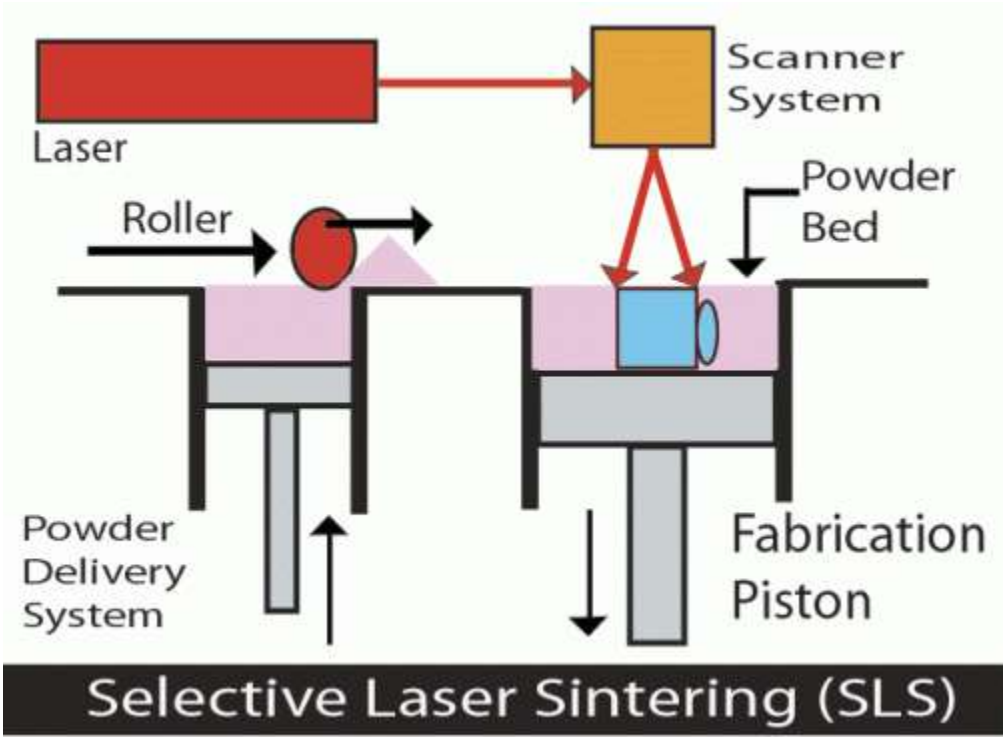


Figure II.10 : Procédé de Selective Laser Sintering

II.3.2 mode de fonctionnement :

L'impression 3D fonctionne selon plusieurs procédés, qui diffèrent selon la technologie et le type d'imprimante 3D utilisée. On peut classer ces procédés dans trois grands groupes : [27]

– Le dépôt de matière

– La solidification par la lumière

– L'agglomération par collage

Ces trois procédés fonctionnent selon le même principe de base, c'est à dire superposer des couches de matières selon les coordonnées XYZ d'un fichier 3D. La différence se situe sur la manière dont sont déposées et traitées ses couches, ainsi que le type de matériau utilisé.

Pour la plupart des procédés employés l'utilisateur a besoin :

– d'une imprimante 3D
– de consommable (filament, poudre…)
– d'un fichier 3D (le plus souvent au format STL ou OBJ)
– d'un logiciel de slicing pour trancher le fichier et transmettre les indications à l'imprimante

– d’un ordinateur

La manière d’exporter les fichiers vers l’imprimante diffère selon les marques et les modèles : câble USB, Wi-Fi ou carte SD.

II.3.3.Le principe de fonctionnement de l’imprimante 3D :

En superposant ou en adjoignant des particules ou des filaments d’un matériau entre eux, il est possible de donner forme à un ensemble. Il est là, tout le secret de l’impression 3D. Il suffit de s’imaginer que l’on joue avec des pièces de LEGO de taille minuscule et qu’on les agence de manière à donner une forme précise à une nouvelle création. Il est alors possible d’utiliser des modèles fournis par le fabricant ou sur internet, mais aussi de créer ses propres modèles sur son ordinateur via des logiciels spécialisés en modélisation 3D. Il faut ensuite charger une bobine de fil à l’emplacement prévu à cet effet sur l’imprimante 3D. Cette bobine agit comme l’encre d’une imprimante classique, c’est elle qui constitue la matière de l’objet en cours de création. Une fois la bobine est mise en place, la tête d’impression chauffante aspire ce fil pour le faire fondre progressivement, puis créer des couches successives sur le plateau de verre qui s’abaisse petit à petit. C’est l’accumulation de ces couches tout en hauteur qui forme l’objet final. Les matériaux employés dans les impressions 3D sont généralement des thermoplastiques qui fusionnent entre eux lorsqu’ils sont chauffés, ou d’autres substances qui durcissent lorsqu’elles sont exposées à des lasers. [28]

II.3.4. Le rôle du moteur pas à pas au sein d’une impriment 3D :

Le rôle du moteur pas à pas dans une imprimante 3D est crucial pour contrôler les mouvements des axes, notamment l'extrudeur et le plateau chauffant. Ces moteurs permettent une précision et une efficacité dans le positionnement des éléments lors de l'impression. Ils sont contrôlés par des pilotes pas à pas qui influent sur des aspects tels que la puissance, la vitesse, la précision, la température et le volume sonore de l'imprimante. De plus, les moteurs pas à pas sont essentiels pour assurer des mouvements précis et synchronisés, garantissant la qualité des impressions 3D réalisées. [29]

Le moteur pas à pas joue un rôle important dans une imprimante 3D, y compris dans la phase de semi-assemblage et tout au long du processus d'impression. Voici comment il contribue [29] :

- **Positionnement précis** : Le moteur pas à pas contrôle les mouvements des axes de l'imprimante 3D, tels que l'axe X, Y et Z, ainsi que parfois d'autres axes pour des fonctionnalités avancées. Ces mouvements doivent être extrêmement précis pour obtenir des impressions de haute qualité.
- **Extrusion du filament** : Dans le cas des imprimantes 3D à filament (FDM/FFF), le moteur pas à pas contrôle également l'extrusion du filament à travers la buse chauffée. Il doit réguler la vitesse à laquelle le filament est poussé dans la buse pour assurer un débit uniforme et une couche d'impression constante.
- **Calibration et compensation** : Les imprimantes 3D nécessitent souvent une calibration régulière pour garantir des impressions précises. Le moteur pas à pas est utilisé lors de ces processus de calibration pour ajuster avec précision les paramètres tels que le débit du filament et le niveau du lit d'impression.
- **Contrôle de la vitesse et de l'accélération** : Lors de l'impression, le moteur pas à pas est responsable de contrôler la vitesse et l'accélération des mouvements de l'extrudeuse et des axes. Cela garantit une impression efficace et évite les problèmes tels que le décollement des couches ou les vibrations excessives.
- **Gestion des reprises après incident :** En cas d'interruption imprévue, comme une coupure de courant, le moteur pas à pas est souvent utilisé pour positionner l'extrudeuse et les axes dans leur dernier emplacement connu. Cela permet à l'imprimante de reprendre l'impression là où elle s'est arrêtée, minimisant ainsi les déchets et les défauts.

En résumé, le moteur pas à pas est essentiel à la fois pour le positionnement précis des composants de l'imprimante 3D et pour le contrôle de l'extrusion du filament, ce qui garantit des impressions de haute qualité et de fiabilité.

II.3.5. Les modes de contrôle des moteurs pas à pas d'une imprimante 3D)

Les principales méthodes de contrôle des moteurs pas à pas dans les imprimantes 3D sont : [30]

Contrôle en boucle ouverte :

Les moteurs fonctionnent sans capteur de position, le contrôleur envoie simplement un certain nombre d'impulsions pour faire tourner le moteur d'un angle donné.

C'est une méthode simple et peu coûteuse, mais elle ne permet pas de corriger les erreurs de positionnement.

Contrôle en boucle fermée :

Des capteurs de position, comme des codeurs incrémentaux, mesurent la position réelle du moteur.

Le contrôleur compare la position réelle à la position désirée et ajuste le nombre d'impulsions en conséquence pour corriger les erreurs.

Cette méthode est plus complexe et coûteuse mais permet d'obtenir une meilleure précision.

Microétapes :

Le contrôleur divise chaque pas du moteur en plusieurs sous-pas plus petits.

Cela permet d'obtenir une résolution de positionnement plus élevée et un mouvement plus fluide.

Les microétapes sont généralement utilisées en combinaison avec un contrôle en boucle fermée.

Contrôle par firmware :

Le firmware de l'imprimante 3D contient les instructions et les algorithmes nécessaires pour contrôler les moteurs pas à pas. Il peut être personnalisé ou mis à jour pour prendre en charge de nouvelles fonctionnalités ou améliorations de performances.

Contrôle par logiciel :

Les utilisateurs peuvent contrôler les moteurs pas à pas à l'aide de logiciels de trancheurs qui convertissent les modèles 3D en instructions compréhensibles par l'imprimante. Ces logiciels permettent de personnaliser les paramètres d'impression et d'envoyer les commandes de contrôle aux moteurs pas à pas.

En combinant ces différents modes de contrôle, les imprimantes 3D peuvent produire des impressions précises et de haute qualité tout en maximisant l'efficacité et la fiabilité du processus d'impression.

Dans notre présent travail nous avons pu reproduire une partie des signaux de commande qui sont habituellement émis sous forme de langage logiciel spécifique aux imprimantes 3D venant de leur contrôleur et transmis aux moteurs pas à pas associés. Nous avons pu générer des signaux de commande à l'aide d'une carte Arduino Uno (R3) que nous avons programmé de sorte à envoyer des impulsions de commande nécessaires pour contrôler un seul moteur pas à pas via le circuit FPGA DE2 que nous avons utilisé.

Il existe plusieurs types de carte Arduino, ci-dessous nous nous intéresserons à celle que nous avons utilisé dans ce présent travail :

II.3.6. Types de cartes Arduino :

Il existe de nombreux types de cartes Arduino disponibles sur le marché, mais toutes les cartes ont un point commun : elles peuvent être programmées à l'aide de l'IDE Arduino. Les raisons pour différents types de cartes sont différentes exigences d'alimentation, options de connectivité, leurs applications, etc. Les cartes Arduino sont disponibles en différentes tailles, formes. Les cartes Arduino connues et fréquemment utilisées sont Arduino UNO, Arduino Mega, Arduino Nano, Arduino Micro et Arduino Lilypad [13].

II.3.6.1. Arduino Uno (R3) :

L'Uno est une énorme option pour l'Arduino initial. Il se compose de 14 broches d'entrées/sorties numériques, où 6 broches peuvent être utilisées en tant que sorties de modulation de largeur d'impulsion (PWM), 6 entrées analogiques, un bouton de réinitialisation, une prise d'alimentation, une connexion USB. Il comprend tout ce qui est nécessaire pour tenir le microcontrôleur en place. Connectez-le simplement à un PC à l'aide d'un câble USB et donnez le matériel nécessaire pour commencer à utiliser un adaptateur CA/CC ou une batterie [31].

Figure II.11 : Arduino UNO

II.3.6.1.1. Structure de l'Arduino Uno :

La principale raison d'utiliser Arduino Uno est que l'UNO est la meilleure carte pour se lancer dans l'électronique et le codage, elle est la carte la plus robuste sur lequel on peut commencer à bricoler. L'ONU est la carte la plus utilisé et documenté de toute la famille Arduino & Génuine.

Venir à l'origine d'Arduino Uno "Uno" signifie un en italien et a été choisi pour marquer la sortie d'Arduino Software (IDE) 1.0. La carte Uno et la version 1.0 du logiciel Arduino (IDE) étaient les versions de référence d'Arduino, qui ont maintenant évolué vers de nouvelles versions. La carte Uno est la première d'une série de cartes USB Arduino et le modèle de référence pour la plate-forme Arduino.

II.3.6.1.2. Schématique de la carte Arduino :

A Partir du schéma ci-dessus, nous pouvons obtenir l'aperçu schématique de la conception des cartes Arduino. Et la photo suivante spécifie clairement le schéma des broches d'Atmega168 IC, qui est l'essentiel pour contrôler le fonctionnement de l'Arduino Uno Board. Voir le mappage entre les broches Arduino et les ports ATmega328P. La cartographie des Atmega8, 168 et 328 est identique [32].

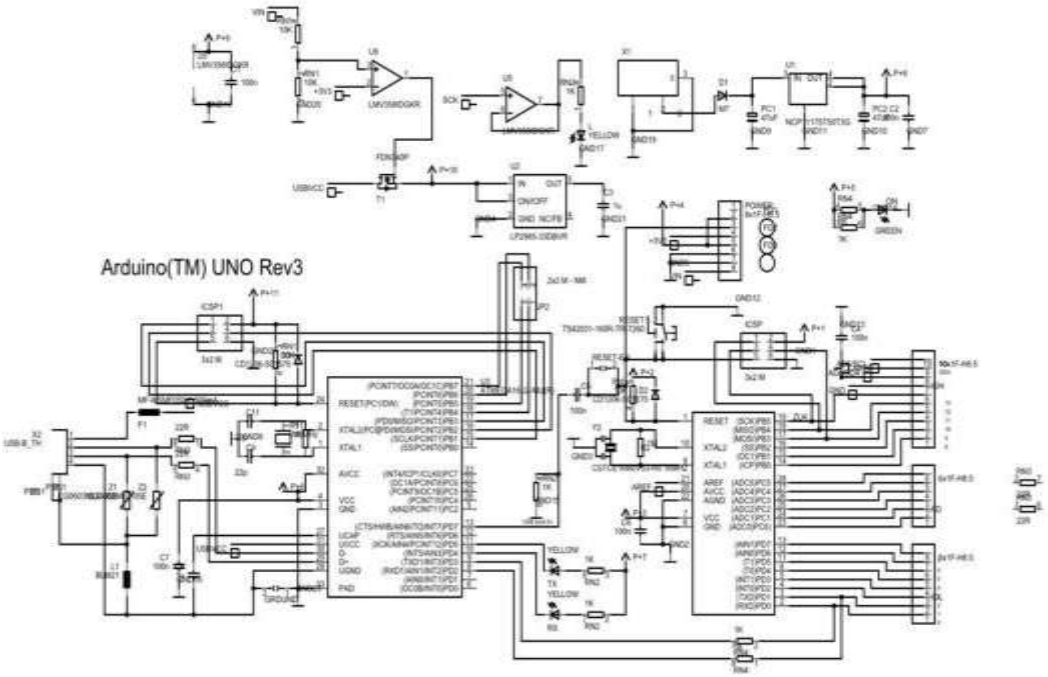


Figure II.12 : schématique de la carte Arduino

II.3.6.1.3. Entrées et Sorties :

Chacune des 14 broches numériques de l'Uno peut être utilisée comme entrée ou sortie à l'aide des fonctions pinMode(), analogWrite(), analogread(), digitalWrite () et digitalRead (). Ils fonctionnent à 5 volts. Chaque broche peut fournir ou recevoir 20 mA dans les conditions de fonctionnement recommandées et possède une résistance de tirage interne (déconnectée par défaut) de 20 à 50 000 ohms. Un maximum de 40 mA est la valeur qui ne doit pas être dépassée sur une broche d'entrée / sortie pour éviter des dommages irréversibles au microcontrôleur. De plus, certaines broches ont des fonctions spécialisées : Série : 0 (RX) et 1 (TX). Utilisé pour recevoir (RX) et transmettre (TX) des données sérient TTL. Ces broches sont connectées aux broches correspondantes de la puce série ATmega8U2 USB-to-TTL. Interruptions externes : 2 et 3.

Ces broches peuvent être configurées pour déclencher une interruption sur une valeur basse, un front montant ou descendant ou une modification de la tension. Valeur. Voir la fonction attachInterrupt () pour plus de détails. PWM: 3, 5, 6, 9, 10 et 11. Fournissez une sortie

PWM 8 bits avec la fonction analogWrite ().SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ces broches prennent en charge la communication SPI à l'aide de la bibliothèque SPI.LED: 13. Une DEL intégrée est pilotée par la broche numérique 13.

Lorsque la broche a la valeur HIGH, elle est allumée, lorsque la broche est LOW, elle est éteinte. Le Uno a 6 entrées analogiques, appelées A0 à A5, chacune offrant une résolution de 10 bits (soit 1024 valeurs différentes). Par défaut, ils mesurent la masse à 5 volts, mais il est possible de modifier l'extrémité supérieure de leur plage à l'aide de la broche AREF et de la fonction analogReference(). Il existe deux autres broches sur la carte: AREF. Tension de référence pour les entrées analogiques. Utilisé avec analogReference() [32].

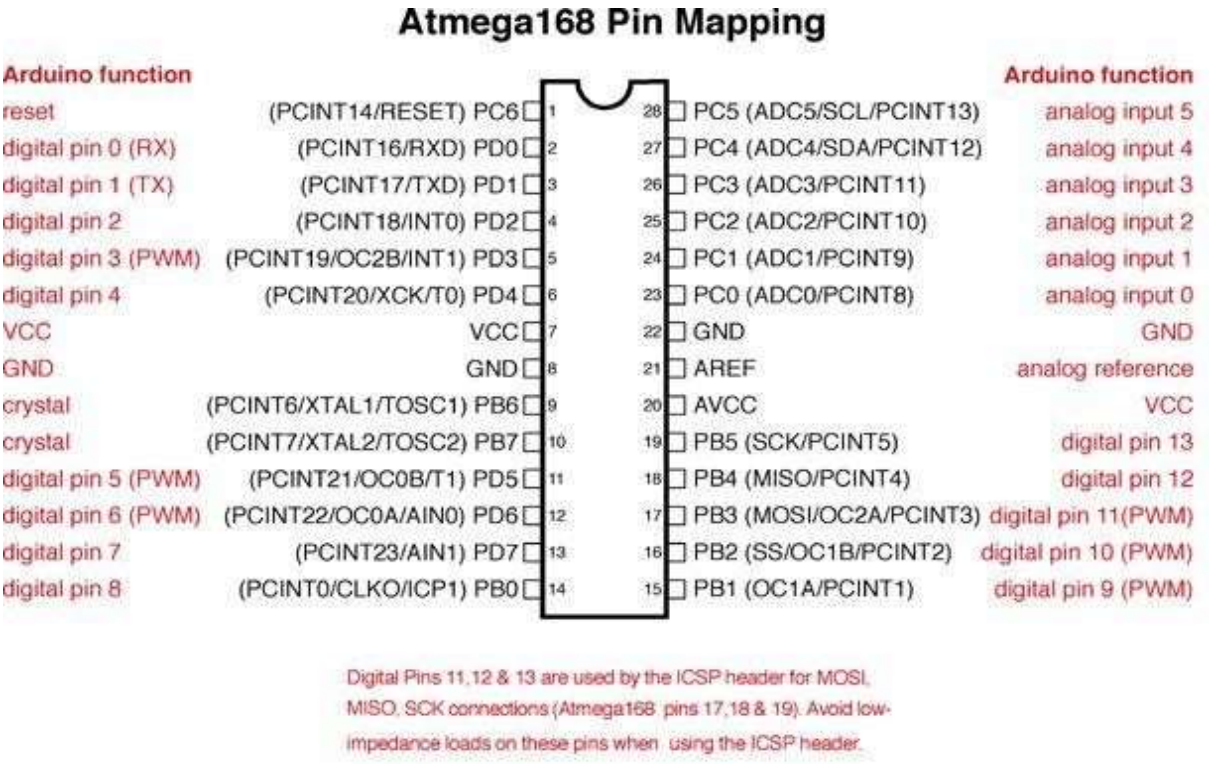


Figure II.13 : Entrées et Sorties de microcontrôleur (Arduino)

II.3.6.1.4. Programmation de l'Arduino :

En ce qui concerne sa partie programmation, la principale chose que nous devons avoir en tête est le langage de programmation que nous allons utiliser pour la programmation.

Le langage Arduino est simplement un ensemble de fonctions C/C++ pouvant être appelées à partir d'un code. L'esquisse subit des modifications mineures (par exemple, la génération automatique de prototypes de fonctions) et est ensuite transmis directement à un compilateur C/C++. Et pour le compilateur, voici le nouveau terme appelé Carte de développement intégré Arduino (Arduino IDE). L'Arduino / Genuino Uno peut être programmé avec le (logiciel Arduino (IDE)). Sélectionnez "Arduino / Genuino Uno dans le menu Outils> Carte (en fonction du microcontrôleur de la carte utiliser).

L'ATmega328 de l'Arduino / Genuino Uno est préprogrammé avec un chargeur de démarrage qui nous permet de télécharger du nouveau code sans l'aide d'un programmateur matériel externe.

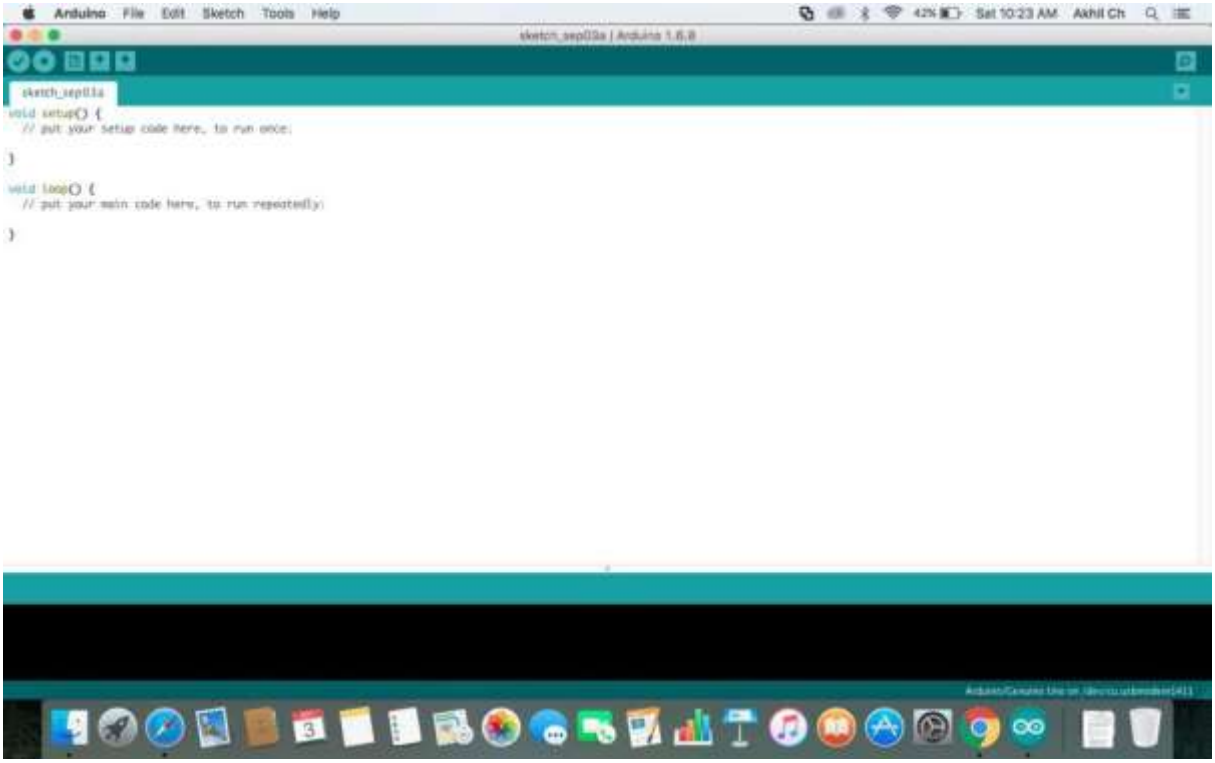


Figure II.14 : Exemple de démarrage de program Arduino

II.3.6.1.5. Les tensions de référence :

La carte Arduino fournit des ports permettant d'accéder à certaines tensions de référence. GND est la référence de la carte Arduino par rapport à laquelle toutes les différences de tension sont mesurées. Si la carte est reliée à l'ordinateur par un câble USB, cette tension est celle de la terre.

Les ports 5V et 3V3 donnent accès aux tensions de 5 V et de 3.3 V. Ces tensions sont normalement régulées et précises. Une exception : quand la carte est branchée sur un port USB sans alimentation externe, le port 5 V ne provient plus de la carte Arduino mais directement du câble USB, la tension de référence 5 V n'est alors plus aussi bien régulée. VIN est la tension de l'alimentation externe, quand il y en a une. Attention : si on relie directement le port 5 V au port GND (ou le port 3V3 au port GND, ou le port 5V au port 3V3), on provoque un court-circuit qui endommagera la carte.

II.4. Intégration d'une interface de communication entre le FPGA et le microcontrôleur principal de l'imprimante 3D :

Pour permettre au FPGA de recevoir des instructions de mouvement depuis le microcontrôleur principal de l'imprimante 3D, on a les principales étapes à suivre : [33]

II.4.1.Choix du protocole de communication :

Les protocoles les plus adaptés sont :

II.4.1.1. SPI (Serial Peripheral Interface) : communication série synchrone simple et rapide [34]

Le bus SPI est une liaison série synchrone qui opère en mode "full duplex" - émission / réception simultanée, La méthode d'accès est du type maître / esclave et c'est toujours le maître qui a l'initiative des échanges : quand le maître sélectionne l'esclave et génère l'horloge, les données sont échangées dans les deux direct ions, simultanément.

Principe :

- Le maître ne tient pas compte de la donnée reçue dans le cas d'un échange "écriture seule" ou alors il envoie un octet sans importance (0xFF) dans le cas d'un échange "lecture seule".
- La communication avec un esclave de type CODEC par exemple (coder-decoder), permet d'exploiter pleinement les capacités du bus SPI, avec un flot de données bidirectionnel.

Principe de communication entre le maitre et l'esclave ;

- Le maître génère l'horloge et sélectionne l'esclave avec qui il veut communiquer.
- L'esclave répond aux requêtes du maître.
- A chaque coup d'horloge le maître et l'esclave s'échangent un bit.
- Après huit coups d'horloges le maître a transmis un octet à l'esclave et vice-versa.
- La vitesse de l'horloge est réglée selon des caractéristiques propres aux périphériques.

II.4.1.2. I2C (Inter-Integrated Circuit) : [35] Communication série synchrone avec adressage, plus flexible que SPI.

Le bus I^2C (BUS I^2C est une marque déposée par PHILIPS composants) constitue un moyen simple de relier (ou de mettre en réseau) des composants ou des sous-ensembles électroniques. Ce bus permet un échange bidirectionnel d'informations à une fréquence pouvant atteindre 100 Kbits/s. Il fonctionne selon le mode de transmission série et n'utilise que deux conducteurs actifs : SDA (pour Sérial Data) et SCL (pour Sérial CLock).

Caractéristiques :

- Le bus I2C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement trois fils : Un signal de donnée (SDA), un signal d'horloge (SCL), et un signal de référence électrique (Masse).
- Ceci permet de réaliser des équipements ayants des fonctionnalités très puissantes (En apportant toute la puissance des systèmes microprogrammes) et conservant un circuit imprimé très simple, par rapport un schéma classique (8bits de données, 16 bits d'adresse + les bits de contrôle).

- Les données sont transmises en série à 100Kbits/s en mode standard et jusqu'à 400Kbits/s en mode rapide. Ce qui ouvre la porte de cette technologie à toutes les applications où la vitesse n'est pas primordiale.
- De nombreux fabricants ayant adopté le système, la variété des circuits disponibles disposant d'un port I2C est énorme : Ports d'E/S bidirectionnels, Convertisseurs A/N et N/A, Mémoires (RAM, EPREM, EEPROM, etc...), Circuits Audio (Egaliseur, Contrôle de volume, ...) et autre drivers (LED, LCD, ...)
- Le nombre de composants qu'il est ainsi possible de relier est essentiellement limité par la charge capacitive des lignes SDA et SCL : 400 pF

Principe :

Afin de d'éviter les conflits électriques les Entrées/Sorties synchrones SDA et SCL sont de type "Collecteur Ouvert"

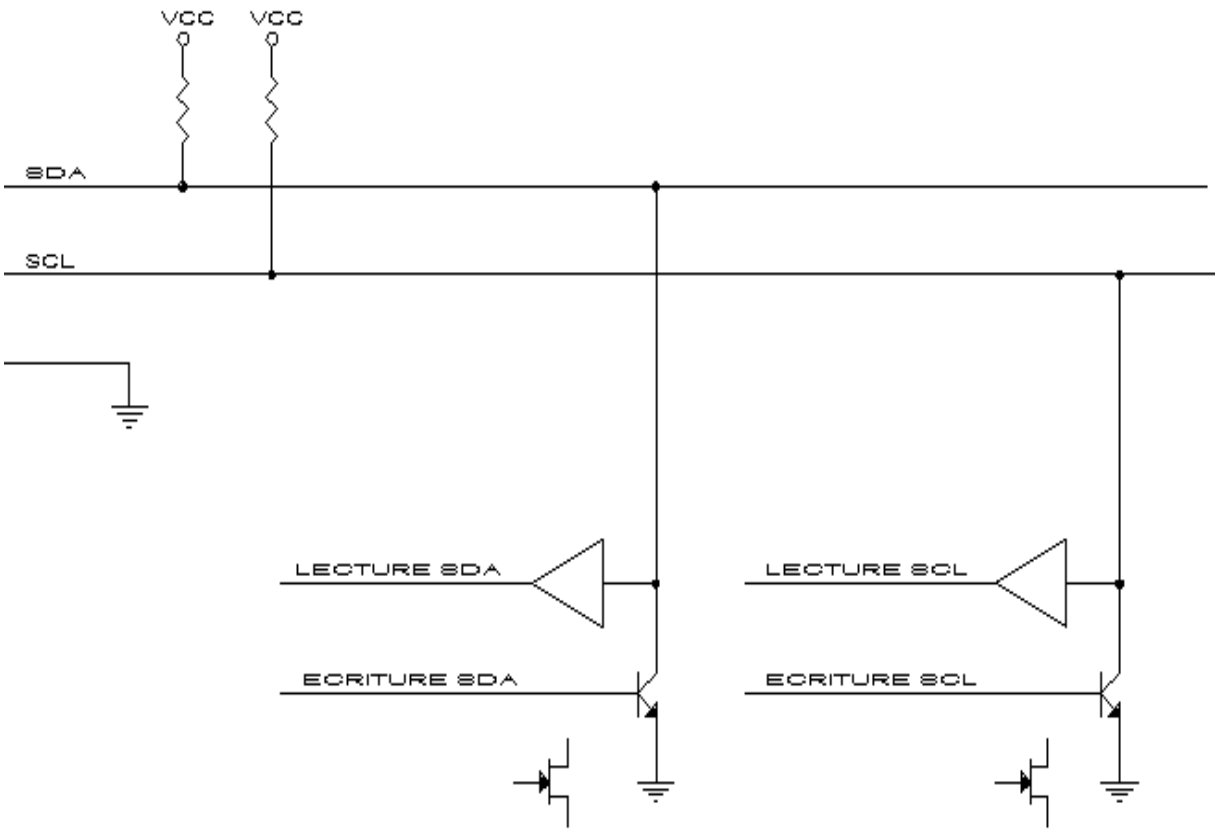


Figure II.15 : Structure d'E/S d'un module I2C.

Plusieurs circuits pouvant être branché en même temps sur le même bus, il a été nécessaire d'instaurer un protocole entre eux, afin d'éviter les problèmes dues à une prise de parole simultanée de différents modules.

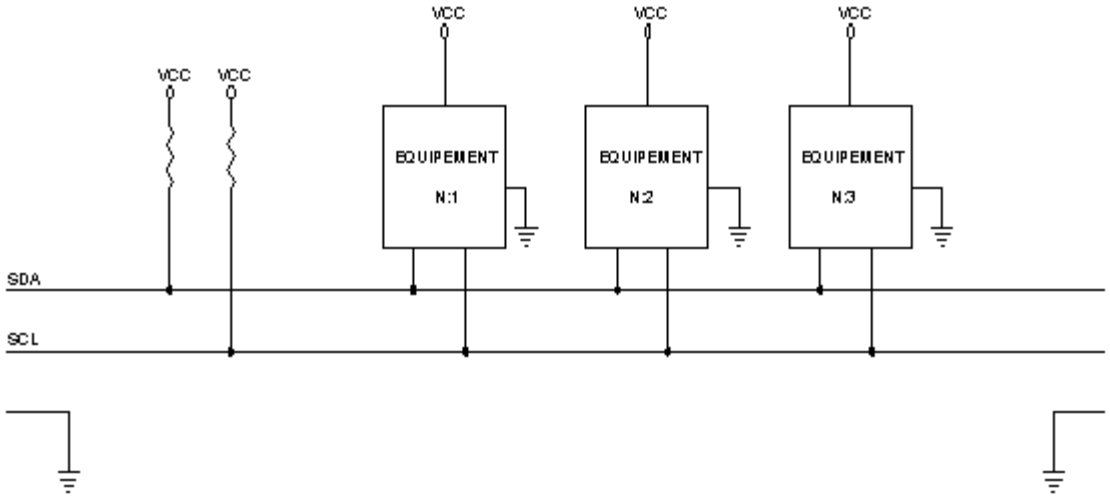


Figure II.16 : Protocole I2C

Le bus I2C supporte deux vitesses de transmission :
le mode standard et le mode rapide.

Caractéristiques des lignes SDA et SCL :

- Les niveaux SDA et SCL dépendent de la tension d'alimentation, comme pour le circuit CMOS. Le niveau haut est défini pour une tension dépassant 0,7 x VCC tandis que le niveau bas est défini pour une tension qui reste inférieure à 0,3 x VCC.
- Le courant maximum circulant dans les étages de sorties des lignes SDA et SCL est fixé à 3mA.
- Les spécifications du bus I2C indiquent que le temps de monté des signaux doit rester inférieur à 1uS dans le mode standard.
  Par ailleurs la capacité maximum qui charge les lignes SDA et SCL ne doit pas dépasser 400pF, pour respecter les temps de descente.

II.4.1.3. UART (Universal Asynchronous Receiver-Transmitter):

UART, abréviation de Universal Asynchronous Receiver/Transmitter, est un protocole de communication série largement adopté utilisé pour transférer des données entre des appareils électroniques. Il s'agit d'une méthode de communication fondamentale dans l'industrie électronique depuis des décennies, permettant aux appareils d'échanger des informations sans avoir besoin d'un signal d'horloge partagé. Au lieu de cela, UART utilise des bits de démarrage et d'arrêt pour encadrer les données transmises, permettant aux appareils de se synchroniser pendant la communication. [36]

Fonctionnement :

Le protocole UART fonctionne de manière asynchrone, ce qui signifie qu'il ne nécessite pas de signal d'horloge dédié pour synchroniser la transmission des données. Au lieu de cela, il utilise des bits de départ (Start bits) et de fin (stop bits) pour indiquer le début et la fin de chaque paquet de données.

Lorsqu'un microcontrôleur envoie des données via l'UART, il convertit les données parallèles en une forme série et les transmet via la ligne de transmission (TX). Le périphérique récepteur convertit ensuite les données série en données parallèles pour les utiliser. Cette conversion série-parallèle permet une transmission efficace des données entre les périphériques.

Principe et caractéristiques :

- L'émetteur-récepteur asynchrone universel (UART) est similaire au protocole I2C. Ces interfaces ont un débit de données maximal d'environ 5 Mbps. Les périphériques UART sont également faciles à utiliser, car aucune horloge n'est envoyée entre eux, tout est asynchrone.
- Notez que l'horloge interne (système) de chaque périphérique UART doit fonctionner à un multiple donné de la fréquence en bauds (en d'autres termes, chaque bit est échantillonné N fois). Seuls deux fils sont utilisés pour la communication entre un contrôleur et un périphérique en aval.
- Le format des données, les niveaux des signaux et le débit en bauds d'un périphérique UART sont configurables avec un circuit de pilotage externe. Malheureusement, cela signifie également qu'il existe peu de règles strictes et rapides pour le routage des périphériques UART.

Le choix dépendra des besoins en termes de débit, de complexité et des interfaces disponibles sur le FPGA et le microcontrôleur.

II.4.1.4.Comparaison entre SPI, I2C et UART : [35]

CARACTÉRISTIQUES	UART	SPI	I2C
Type de protocole	Asynchrone	Synchrone	Synchrone
Nombre de lignes	2 (TX, RX)	4 (MOSI, MISO, SCLK, SS)	2 (SDA, SCL)
Vitesse de transmission	Variable, débit fixe	Haute	Variable, débit fixe
Communication duplex	Simplex, semi-duplex	Full-duplex	Bidirectionnelle
Nombre de périphériques	2 (maître et esclave)	Plusieurs	Jusqu'à 127
Complexité	Simple	Moyenne	Simple
Utilisation des broches	Moins	Plus	Moins
Exemples d'utilisation	Communication série	Cartes SD, afficheurs	Capteurs

Table II.4 : Comparaison entre les trois protocoles

II.4.2. Implémentation de l'interface de communication sur le FPGA :

- Développer un module RTL (Verilog ou VHDL) pour gérer le protocole de communication choisi.
- Connecter ce module aux ports d'entrée/sortie du FPGA correspondants.

- Créer une logique de décodage des instructions de mouvement reçues depuis le microcontrôleur.

II.4.3. Intégration côté microcontrôleur :

- Programmer le microcontrôleur pour qu'il envoie les instructions de mouvement via le protocole de communication choisi.
- Synchroniser l'envoi des instructions avec le FPGA pour garantir un fonctionnement stable.

II.4.4. Tests et validation :

- Effectuer des tests de communication entre le FPGA et le microcontrôleur.
- Vérifier le bon fonctionnement de la réception et de l'exécution des instructions de mouvement sur le FPGA.

En suivant ces étapes, nous pourrons intégrer une interface de communication efficace entre le FPGA et le microcontrôleur principal de l'imprimante 3D, permettant ainsi au FPGA de recevoir et d'exécuter les instructions de mouvement.

II.5. Génération de signaux de commandes depuis la carte FPGA :

Pour générer les signaux de commandes depuis la carte FPGA ALTERA DE2 on aura besoin d'étudier les états du moteur pas à pas à aimant permanant bipolaire pour ainsi générer ces signaux de commandes.

II.5.1.Les signaux pas (step) :

Les états du moteur (sens horaire)

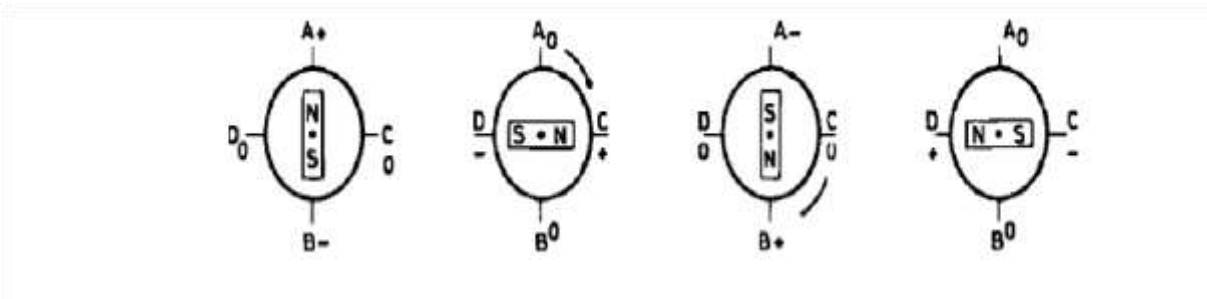


Figure II.17 : États du moteur pour effectuer un pas

Effectuer un pas dans le sens horaire :

Pole / Phase	A	B	C	D
Phase1	1	0	0	0
Phase2	0	0	1	0
Phase3	0	1	0	0
Phase4	0	0	0	1

Table II.5 : Les phases pour effectuer un pas

Pour contrôler le moteur en mode pas complet on va générer les signaux A, B, C, D depuis la carte FPGA.

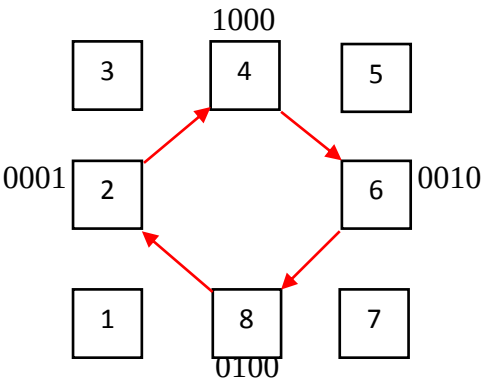


Figure II.18 : L'état machine mode pas complet.

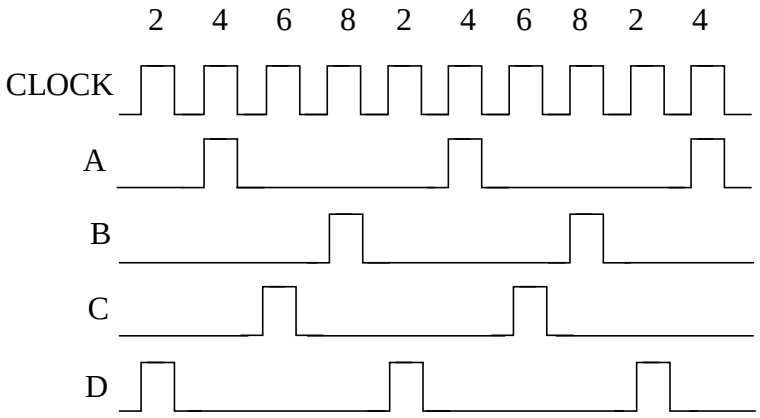


Figure II.19 : Chronogramme des sorties A, B, C, D et signal d'horloge mode pas complet.

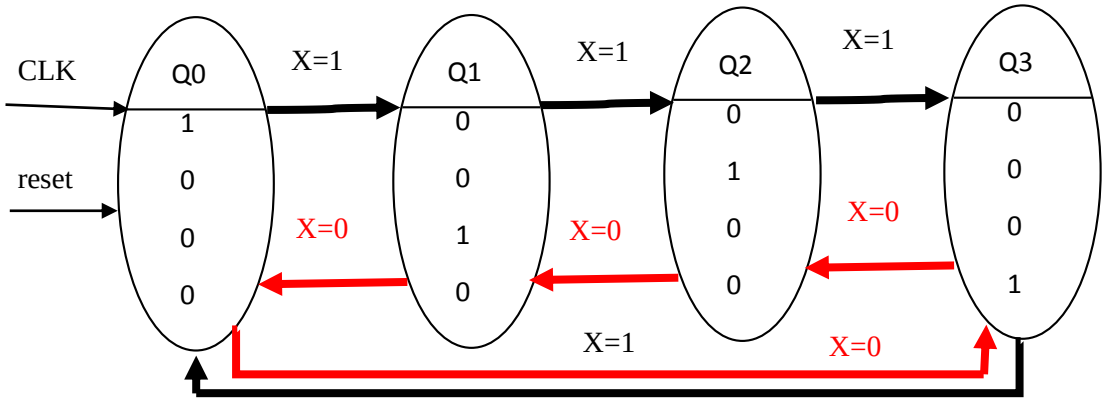


Figure II.20 : Schéma machine à états mode pas complet

Dans le cas de fonctionnement en demi-pas :

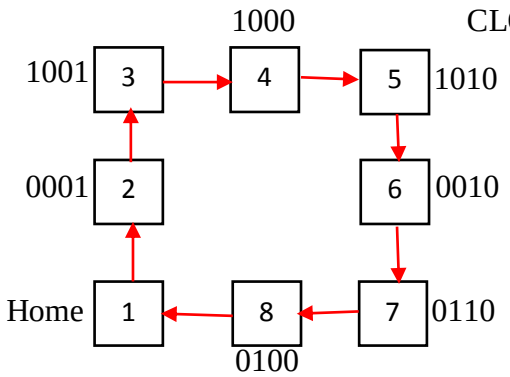


Figure II.21 : L'état machine mode demis pas

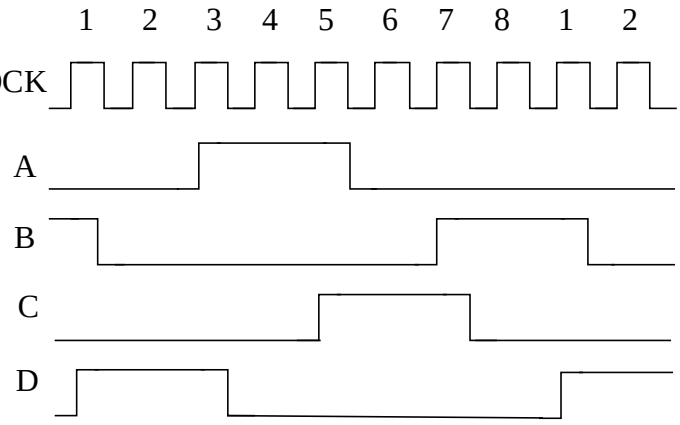


Figure II.22 : Chronogramme signal d'horloge des sorties A, B, C, D mode demis pas

Schéma machine à états :

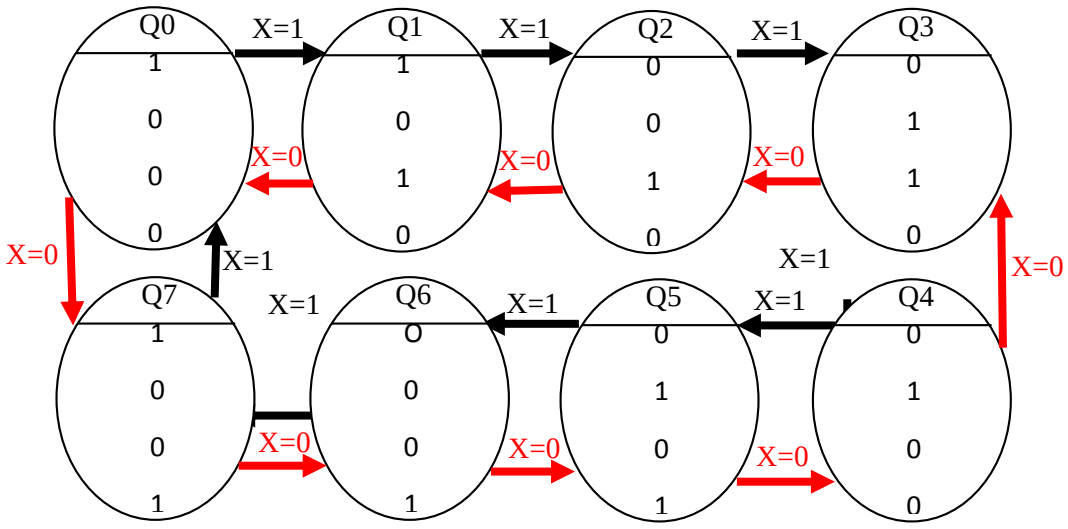


Figure II.23 : Schéma machine à états mode demi-pas

II.5.2. Génération de signaux d'activation et de désactivation :

On va configurer un état dont on envoie un signal logique avec A= 0, B=0, C=0, D=0.

Ça veut dire qu'il y'aura pas de courant qui passe de la carte FPGA au moteur, on va définir cette état de désactivation si reset=1 avec un switch sinon si reset=0 c'est l'état d'activation.

II.5.3. Génération de signaux de direction :

Le changement de direction veut dire que le moteur va tourner dans le sens anti horaire on va associer le sens horaire a un switch qui définit la direction de rotation du moteur on va l'appeler X si X=0, le moteur va tourner dans le sens horaire sinon si X= 1, le moteur va tourner dans le sens anti horaire.

II.6. Discussion :

Cette étude montre l'importance des moteurs pas à pas dans les applications de haute précision telles que les imprimantes 3D. La combinaison des technologies FPGA et Arduino permet de créer des systèmes de contrôle robustes et flexibles. Le choix des protocoles de communication et l'implémentation précise des signaux de commande sont essentiels pour le bon fonctionnement des moteurs et, par extension, de l'imprimante 3D.

Chapitre III

III.1. Préambule :

La réalisation de notre système s'est déroulée en deux parties principales, la première partie nous l'avons consacré à la commande du moteur pas à pas et la visualisation de ses signaux par carte FPGA en utilisant le « logicielle Quartus II 9.1sp2 Web Edition », et la deuxième partie nous l'avons consacré à la génération les signaux de contrôle à partir d'une carte Arduino que nous avons programmé et qui joue le rôle du contrôleur d'une imprimante 3D .Nous avons aussi utilisé la carte pilote L298N pour l'interfaçage carte FPGA / moteur.

III.2. Partie 01 : Commande par carte FPGA Altera DE2 d'un moteur pas à pas à aimant permanant bipolaire « mode pas complet ».

Dans cette partie de ce chapitre nous présentons les différents composants que nous avons utilisés pour la réalisation de notre système pratiquement.

III.2.1. Diviseur d'horloge : Pour commander le moteur pas à pas avec la carte FPGA on aura besoin d'un diviseur d'horloge car la fréquence de la carte FPGA est très élevée et le moteur ne pourra pas gérer cette fréquence,

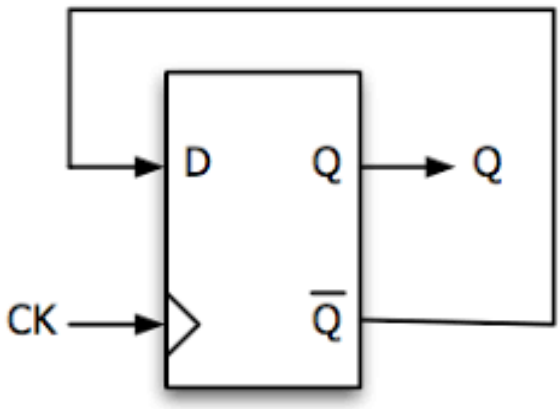


Figure III.1 : Diviseur d'horloge

III.2.2. Carte pilote L298N : Pour la communication entre FPGA et le moteur on utilisera une carte pilote L298N pour but de protection.

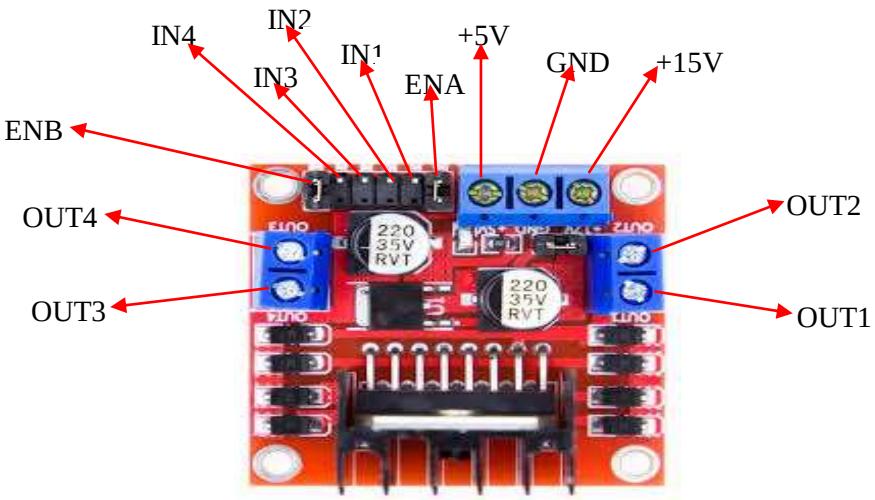


Figure III.2 : Carte pilote de moteur L298N

III.2.2.1. Définition de la Carte Pilote de Moteur L298N :

La carte pilote de moteur L298N est un module électronique qui permet de contrôler la vitesse et le sens de rotation des moteurs à courant continu (DC) et des moteurs pas à pas. Elle utilise le circuit intégré L298N, un double pont en H, pour gérer les courants élevés nécessaires au fonctionnement des moteurs.

III.2.2.2. Rôle de la Carte Pilote de Moteur L298N :

Le rôle principal de la carte pilote L298N est de faciliter le contrôle des moteurs électriques dans divers projets de robotique et d'automatisation. Elle permet :

1. **Le Contrôle de la Direction** : Permet de faire tourner les moteurs dans les deux sens (avant et arrière).
2. **Le Contrôle de la Vitesse** : En utilisant la modulation de largeur d'impulsion (PWM), elle permet de régler la vitesse des moteurs.
3. **L'Alimentation de Puissance** : Fournit suffisamment de courant aux moteurs pour un fonctionnement optimal, souvent jusqu'à 2A par canal avec un maximum de 4A pour les deux canaux.

III.2.2.3. Fonctionnement de la Carte Pilote de Moteur L298N :

Le fonctionnement de la carte L298N repose sur l'utilisation d'un pont en H, qui est un circuit composé de quatre interrupteurs (transistors ou MOSFET) permettant de changer la direction du courant traversant le moteur.

1. **Entrées Logiques** : La carte dispose de plusieurs broches d'entrée qui se connectent à un microcontrôleur (comme un Arduino ou un Raspberry Pi). Ces entrées sont utilisées pour contrôler les interrupteurs internes du pont en H.
    - **IN1 et IN2** : Contrôlent le sens de rotation du moteur connecté au canal A.
    - **IN3 et IN4** : Contrôlent le sens de rotation du moteur connecté au canal B.
2. **Entrées PWM** : Les broches ENA et ENB permettent de contrôler la vitesse des moteurs par modulation de largeur d'impulsion.
3. **Alimentation** : La carte nécessite une alimentation externe pour les moteurs (généralement entre 5V et 35V) et une alimentation logique (5V) pour les circuits de commande.

III.2.2.4. Diagramme de Fonctionnement :

Un moteur pas à pas bipolaire a deux enroulements (A et B). Chaque enroulement doit être alimenté dans les deux sens pour permettre au moteur de tourner correctement. Cela nécessite l'utilisation de deux ponts en H, que le L298N peut fournir puisqu'il a deux canaux.

III.2.2.5. Séquence de Commande

Pour faire tourner un moteur pas à pas bipolaire, une séquence spécifique de signaux doit être appliquée aux enroulements. Voici la séquence de base (en mode pas complet) pour faire tourner le moteur dans un sens :

Étape	IN1	IN2	IN3	IN4
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0

Table III.I : séquence de base (en mode pas complet) pour faire tourner le moteur dans un sens

Explication de la Séquence

1. **Étape 1 :**
    - Enroulement A est alimenté de IN1 (HIGH) à IN2 (LOW).
    - Enroulement B est alimenté de IN3 (HIGH) à IN4 (LOW).
2. **Étape 2 :**
    - Enroulement A reste alimenté de IN1 (HIGH) à IN2 (LOW).
    - Enroulement B est inversé, alimenté de IN4 (HIGH) à IN3 (LOW).
3. **Étape 3 :**
    - Enroulement A est inversé, alimenté de IN2 (HIGH) à IN1 (LOW).
    - Enroulement B reste alimenté de IN4 (HIGH) à IN3 (LOW).
4. **Étape 4 :**
    - Enroulement A est alimenté de IN2 (HIGH) à IN1 (LOW).
    - Enroulement B est inversé, alimenté de IN3 (HIGH) à IN4 (LOW).

III.2.3. Moteur pas à pas à aimant permanant bipolaire EM-546 :

La figure ci-dessous (Fig.III.3) représente la photographie du moteur pas à pas bipolaire à aiment permanent que nous avons utilisé dans notre réalisation

Figure III.3 : Moteur EM-546

III.2.3.1. Spécifications du Moteur EM-546 :

L'OKI EM-546 est un moteur pas à pas bipolaire avec les spécifications suivantes :

Tension : 12 V

Pas par révolution : 48 pas, ce qui équivaut à un angle de pas de 7,5 degrés par pas.

Configuration de la bobine : Bipolaire, nécessitant une séquence de conduite spécifique pour un fonctionnement correct.

Dimensions physiques :

Diamètre : 42 mm

Hauteur : 16 mm

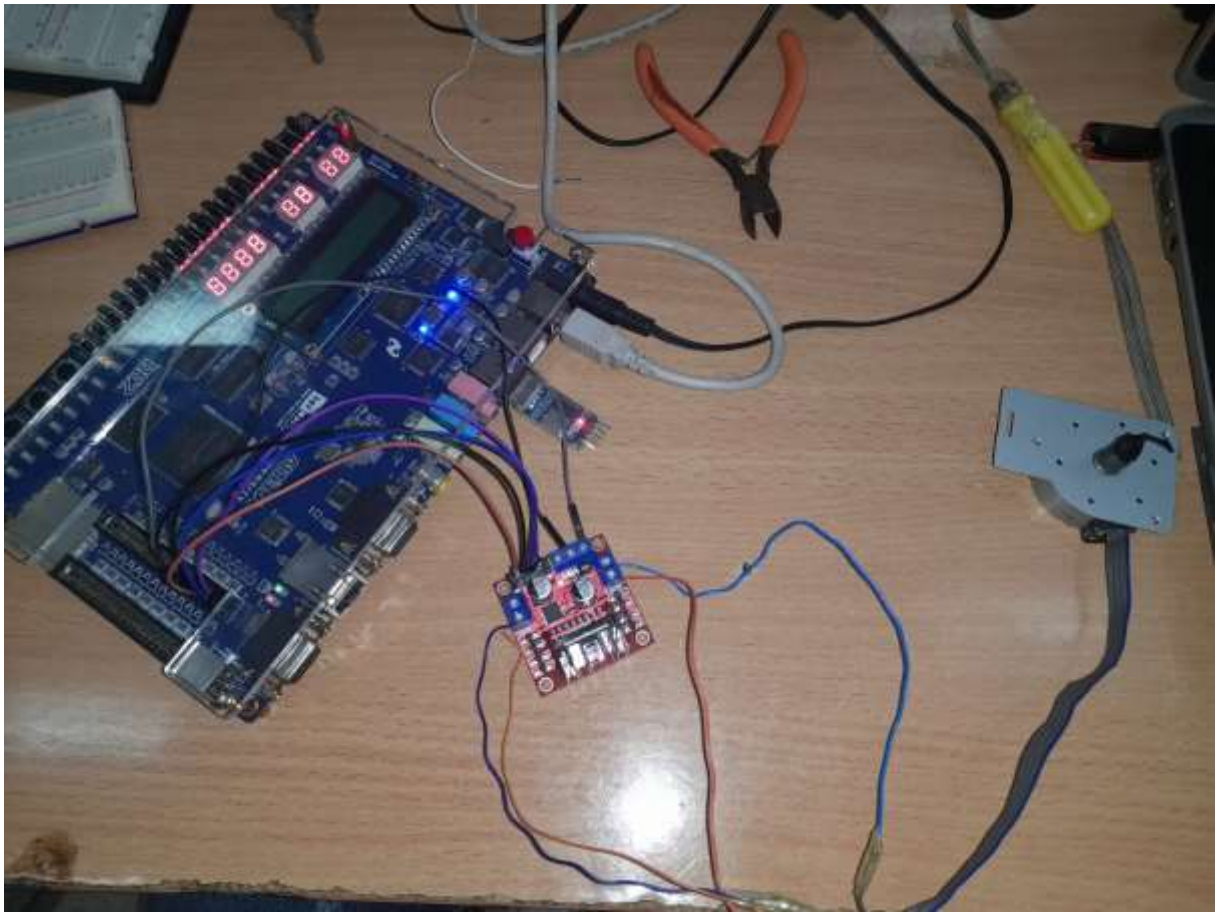
Diamètre de l'arbre : 3 mm

Le moteur est couramment utilisé dans les applications nécessitant un positionnement précis, comme dans les imprimantes ou les petites machines CNC. Il fonctionne efficacement en mode

micro pas mais nécessite une gestion minutieuse de la séquence de conduite pour éviter les étapes inégales (IndustryArena) (Forums Adafruit) (Datasheet4U).

III.2.4. Projet et schéma de branchement :

La figure en dessous représente la photographie du système de pilotage du moteur pas à pas branché aux sorties du module L298N d'une part et aux pins de sorties du circuit FPGA DE2 que nous avons utilisé.



Dans la figure ci-dessous (III.4) nous représentons schématiquement les différents composants utilisés dans notre réalisation, les enroulements du moteur pas à pas bipolaires sont mis en évidence ainsi que le sens de rotation et les différentes positions prises par le rotor dudit moteur.

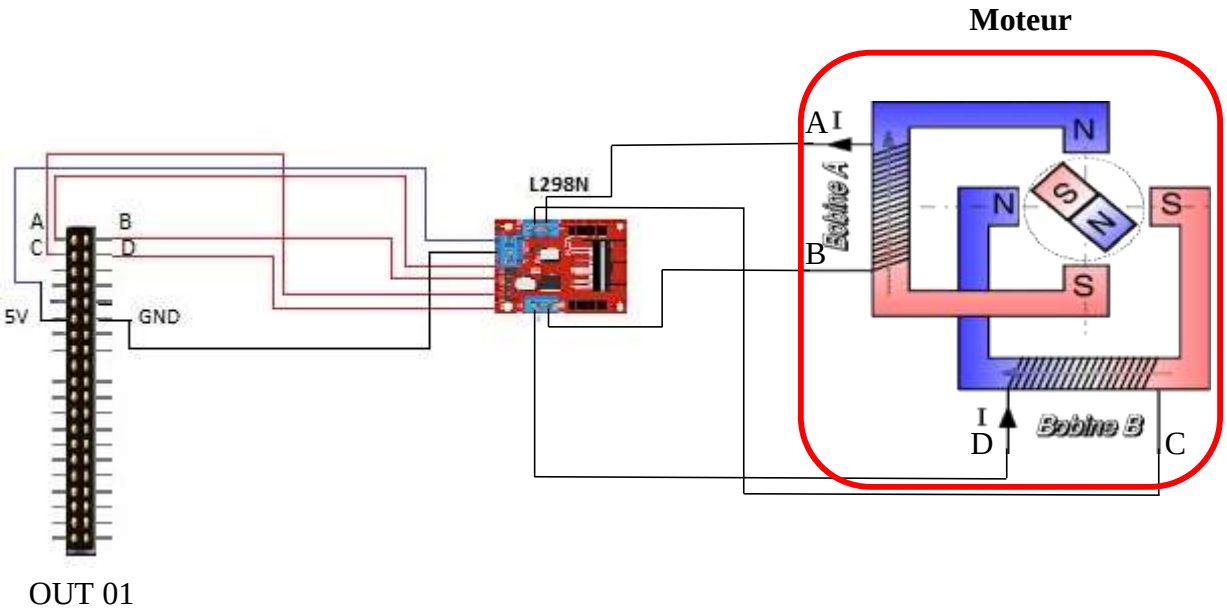
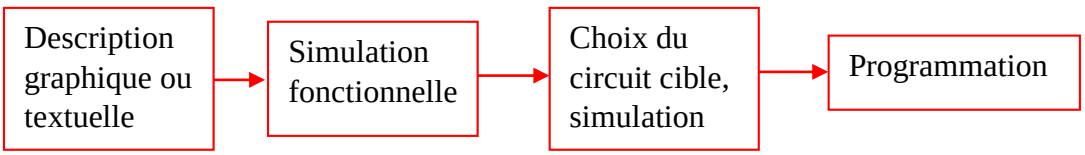


Figure III.4 : Schéma de réalisation

III.2.5. Programmation sur Quartus :

Dans ce présent travail nous avons utilisé le logiciel **Quartus version « II 9.1sp2 Web Edition »** qui permet entre autres, la description d'un projet (système numérique), sa compilation, sa simulation logique et temporelle, son analyse temporelle et la programmation d'un circuit cible (CPLD ou FPGA).

La programmation sur le logicielle **Quartus II 9.1sp2 Web Edition** se fait on 4 étapes



La description du système numérique (logique) peut être faite à l'aide d'une des entrées suivantes :

- **Editeur de texte :** pour l'utilisation du langage VHDL ou VERILOG.

- **Editeur graphique :** permet d'utiliser les composants prédéfinis des bibliothèques fournies par le logiciel.

• **Editeur de chronogrammes :** avec lequel on représentera l'évolution temporelle et celle attendue des sorties.

Pour une meilleure compréhension des différentes étapes nécessaires pour pouvoir effectuer correctement notre programme, nous avons pris le soin de les représenter toutes ci-dessous sous forme de capture d'écran.

Etape 1 :

- Entrer dans le logiciel Quartus

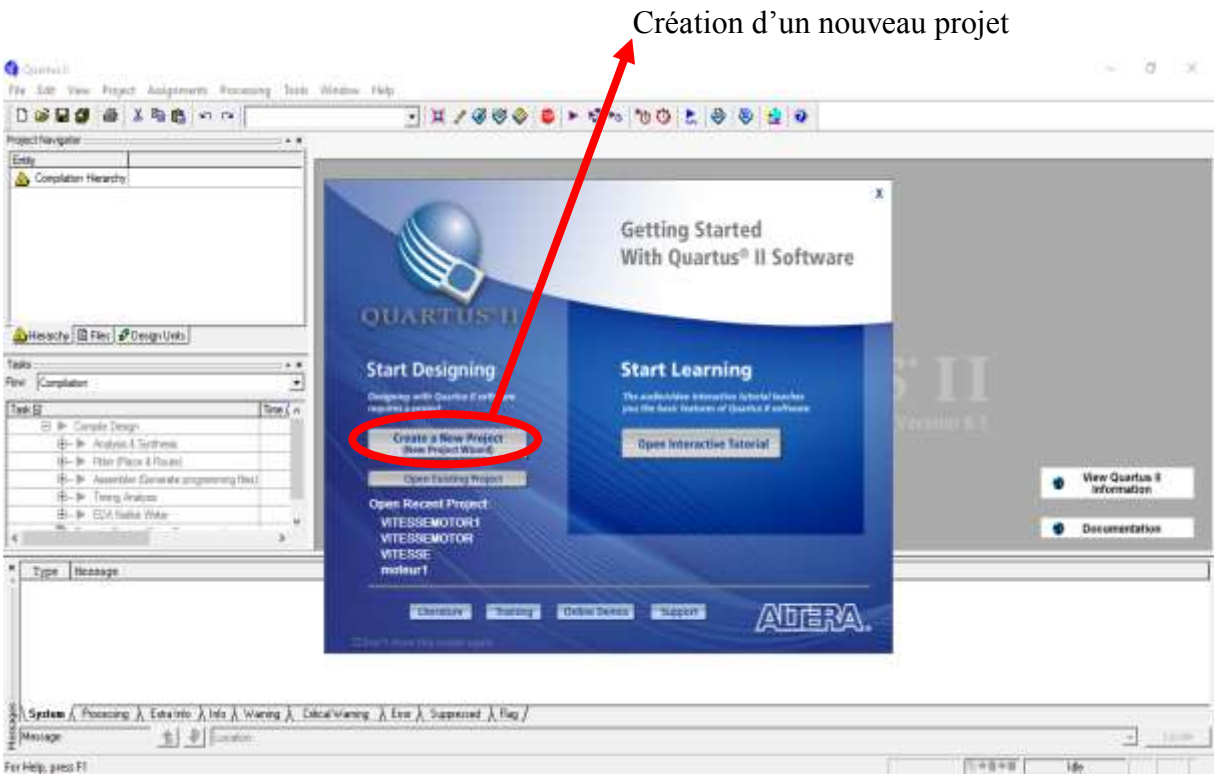


Figure III.8 : Entrée sur logicielle QUARTUS

Etape 2

- On donne un nom au projet par exemple VITESSEMOTOR ainsi l'emplacement de notre projet sur pc on va créer un dossier qui porte le même nom du projet

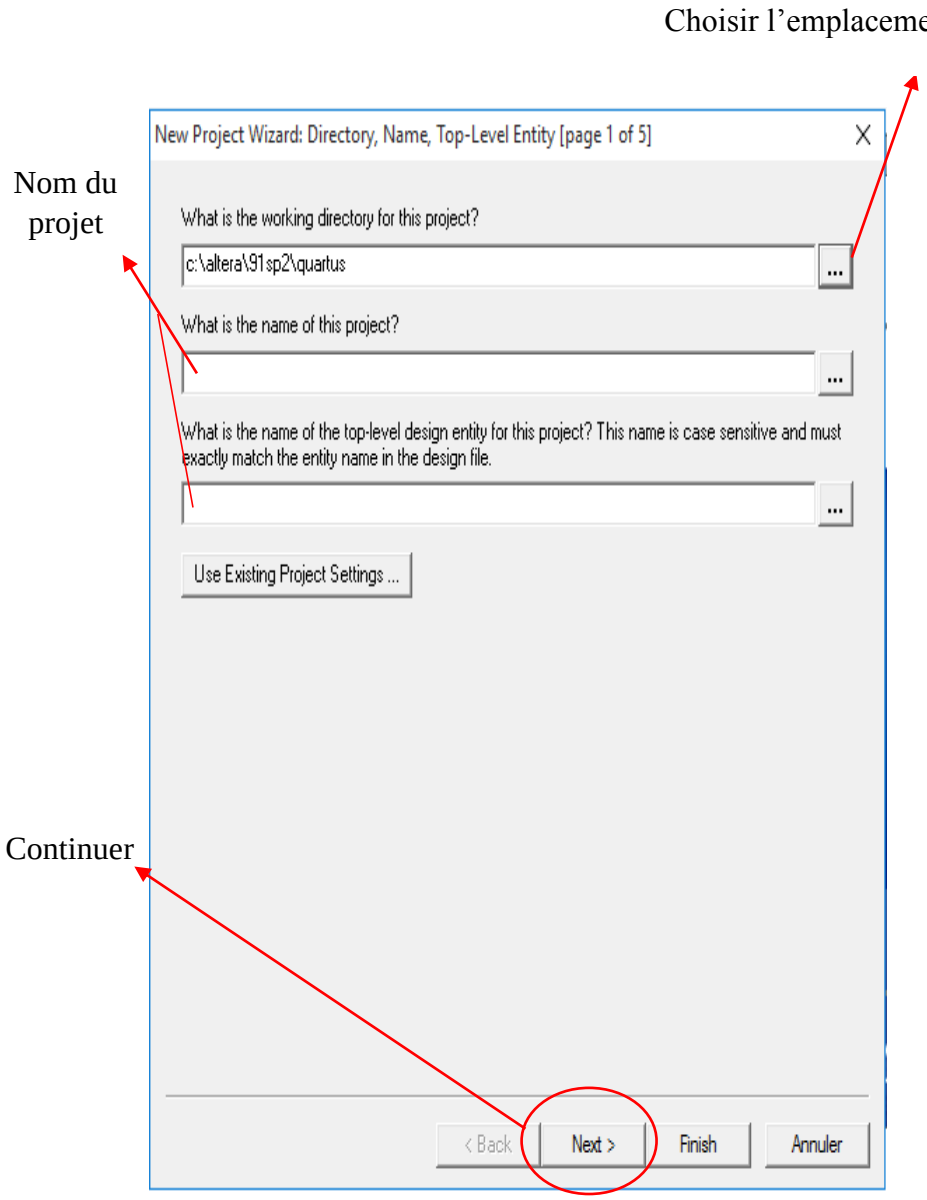


Figure III.9 : Nommassions du projet et de son emplacement

Etape 3 :

- Donner les informations de notre carte qu'on va utiliser pour téléversement du programme qu'on va créer dans notre cas on utilisera une carte de la famille cyclone II avec une référence de carte EP2C35F672C6

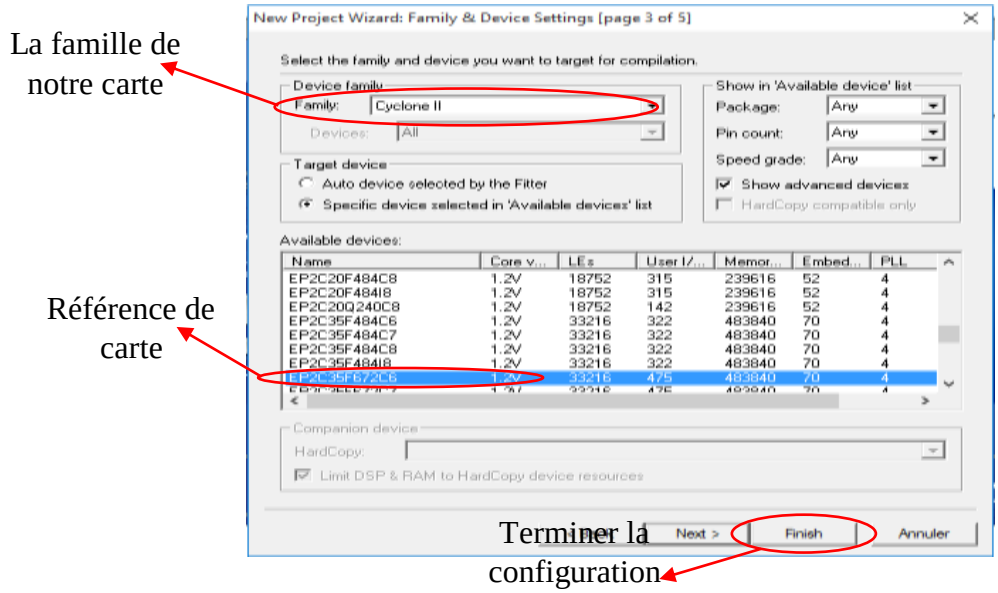


Figure III.10 : Insertion de la famille et référence de la carte

Etape 4 :

- Aperçu du logiciel après configuration

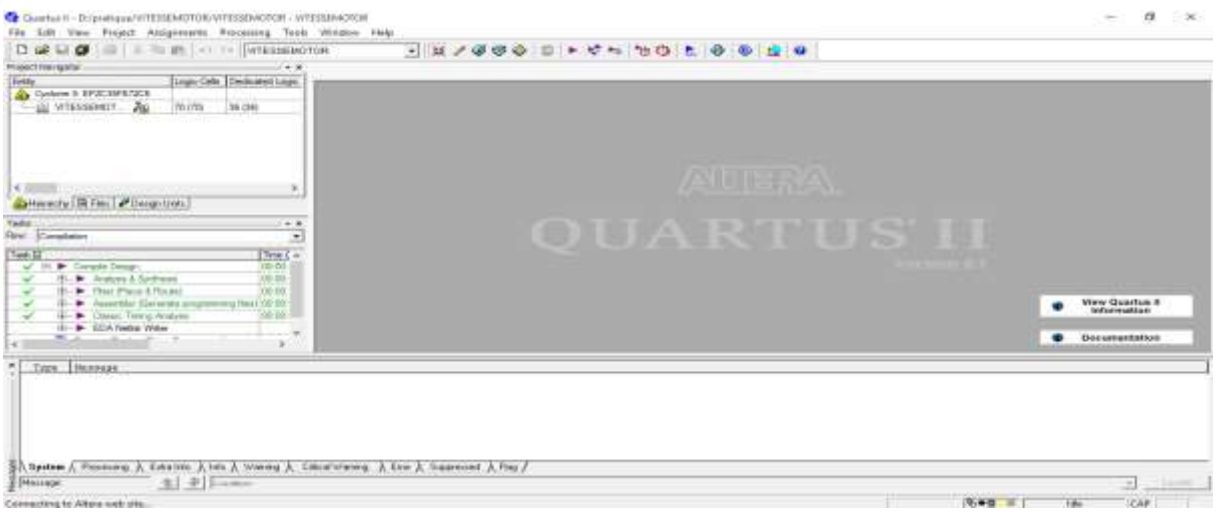


Figure III.11 : Aperçu du logicielle QUARTUS après configuration

Etape 5 :

- Création du projet vhdl

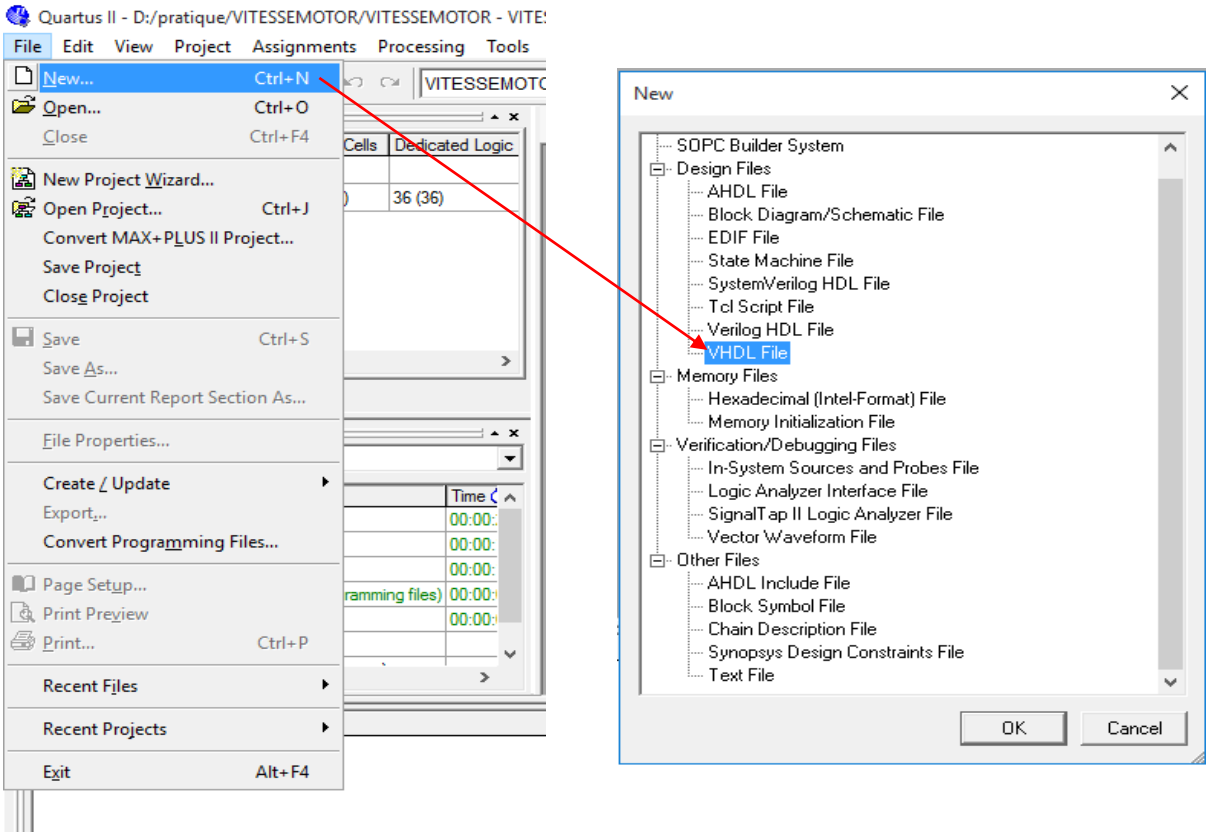


Figure III.12 : Création du projet vhdl

Etape 6 :

- Aperçu de la fenêtre vhdl code

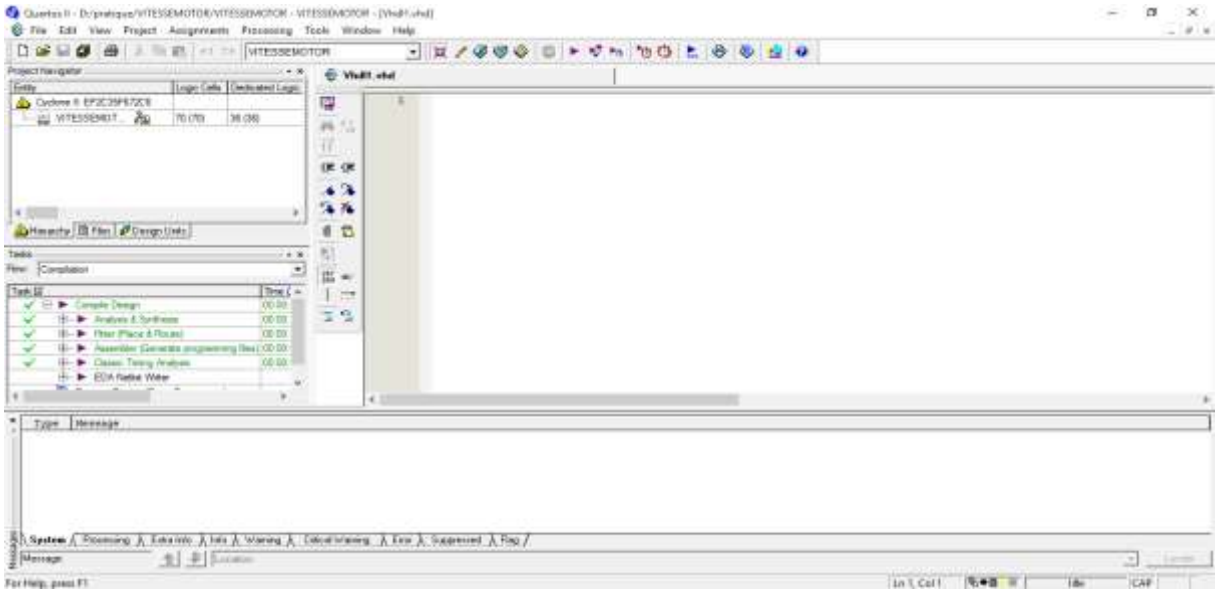


Figure III.13 : Aperçu de la fenêtre vhdl code

Etape 7 :

- Dans cette étape on doit avoir une bonne maitrise du langage VHDL pour écrire le code le plus adapter, et correcte pour le bon fonctionnement de notre projet. puis compiler pour savoir s'il y a des fautes dans notre programme.

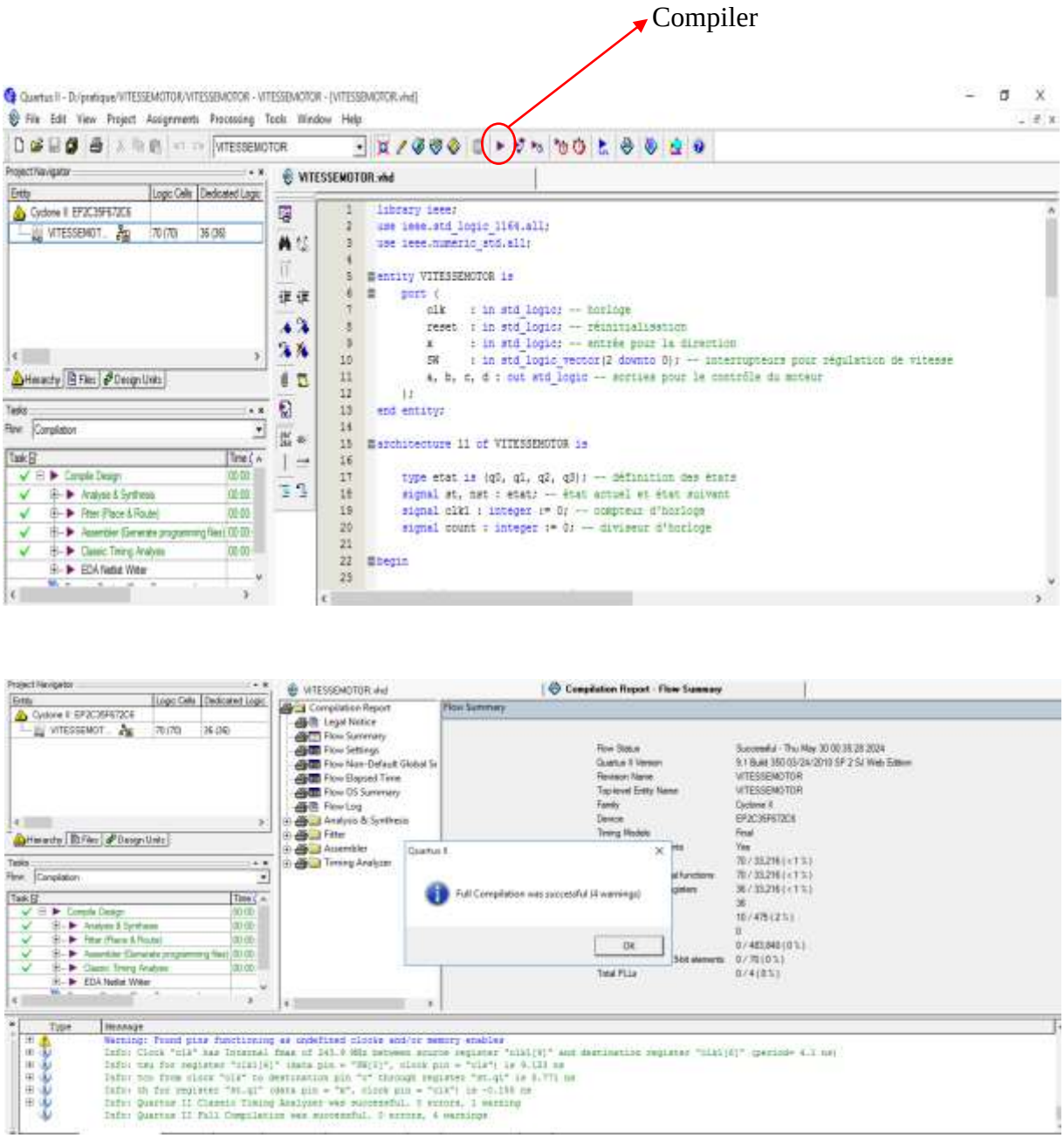


Figure III.14 : Code VHDL et compilation sur logicielle

Apres compilation faut pas avoir d'erreurs dans le code sinon ça va pas marcher

III.2.6. Assignement des broches sur « Quartus II 9.1sp2 Web Edition » :

Ouvrir la fenêtre d'assignement sur Quartus et affecter l'assignement aux broches qu'on utilise

Dans notre cas on a :

	DIRECTION	Location	NOM
a	Sortie	D25	GPIO [0]
b	Sortie	J22	GPIO [1]
c	Sortie	E26	GPIO [2]
d	Sortie	E25	GPIO [3]
x	Entrée	N25	SWITCH(0)
reset	Entrée	N26	SWITCH(1)
clk	Entrée	N2	CLOCK
Sw(0)	Entrée	V2	SWITCH(0)
Sw(1)	Entrée	V1	SWITCH(0)
Sw(2)	Entrée	U4	SWITCH(0)

Table III.2 : Les entrées et les sorties et leurs assignements

Etape 1 : Entrer à l'assignement sur Quartus II

Figure III.15 : Entrer à l'assignement sur Quartus

Etape 2 : Affectation des locations sur assignement

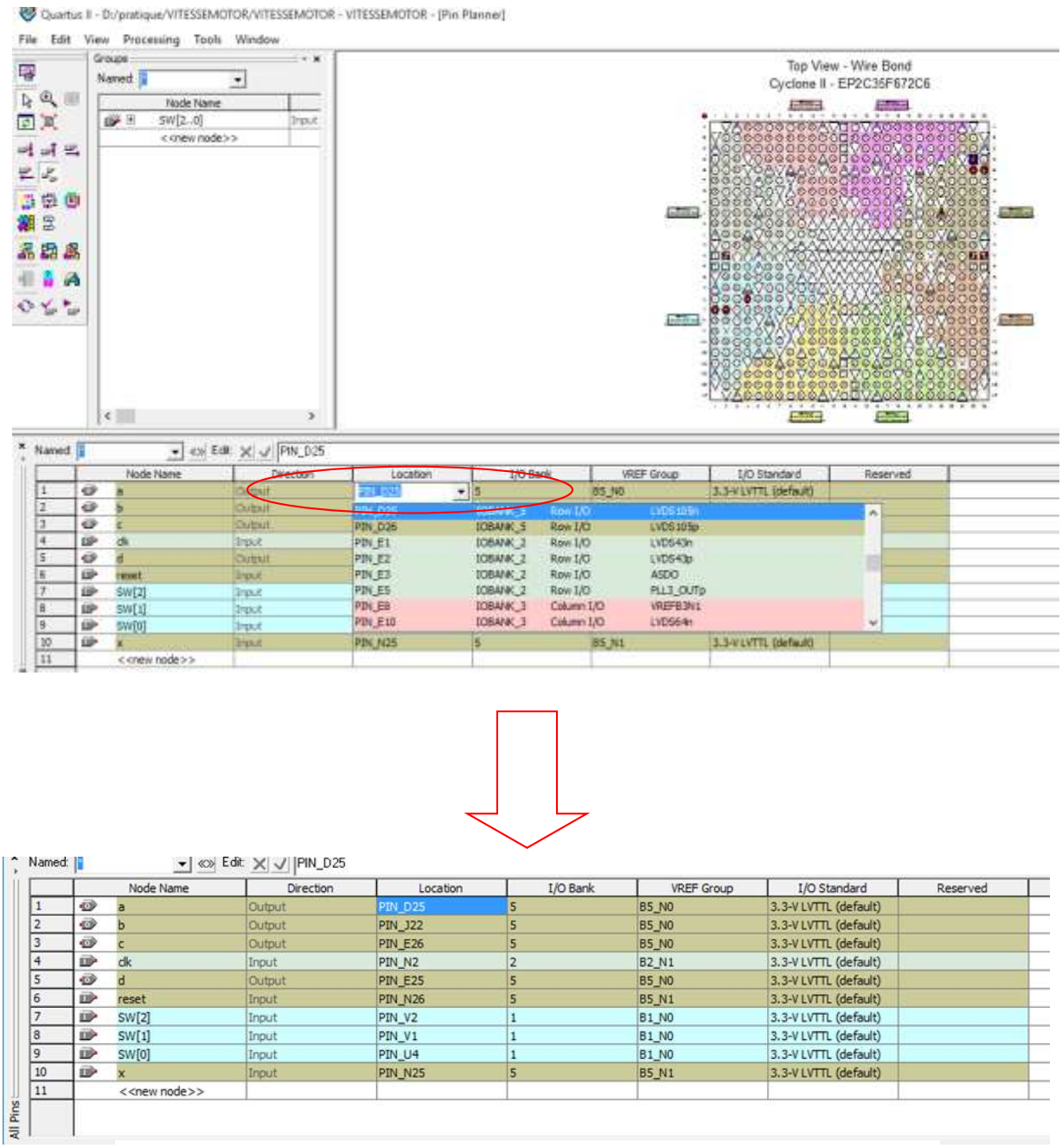


Figure III.16 : Affectation des locations sur assignement

III.2.6. Compilation et téléversement vers FPGA Altera DE2

Pour passer au téléversement du programme faut compiler le programme avec l'assignement puis allumer la carte FPGA et relier la carte au pc avec un câble blaste puis suivre les étapes suivantes

Etape 1 : Aller sur tool puis programmer et on va avoir cette fenêtre

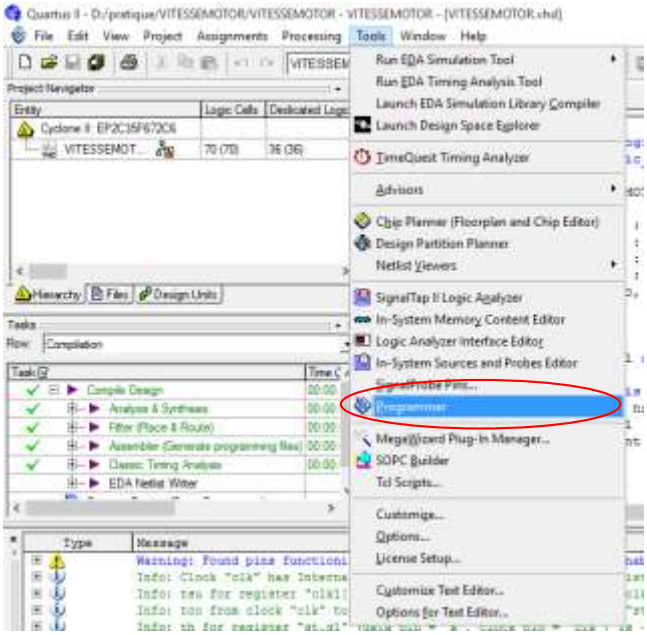


Figure III.17 : Ouverture de fenêtre de téléversement

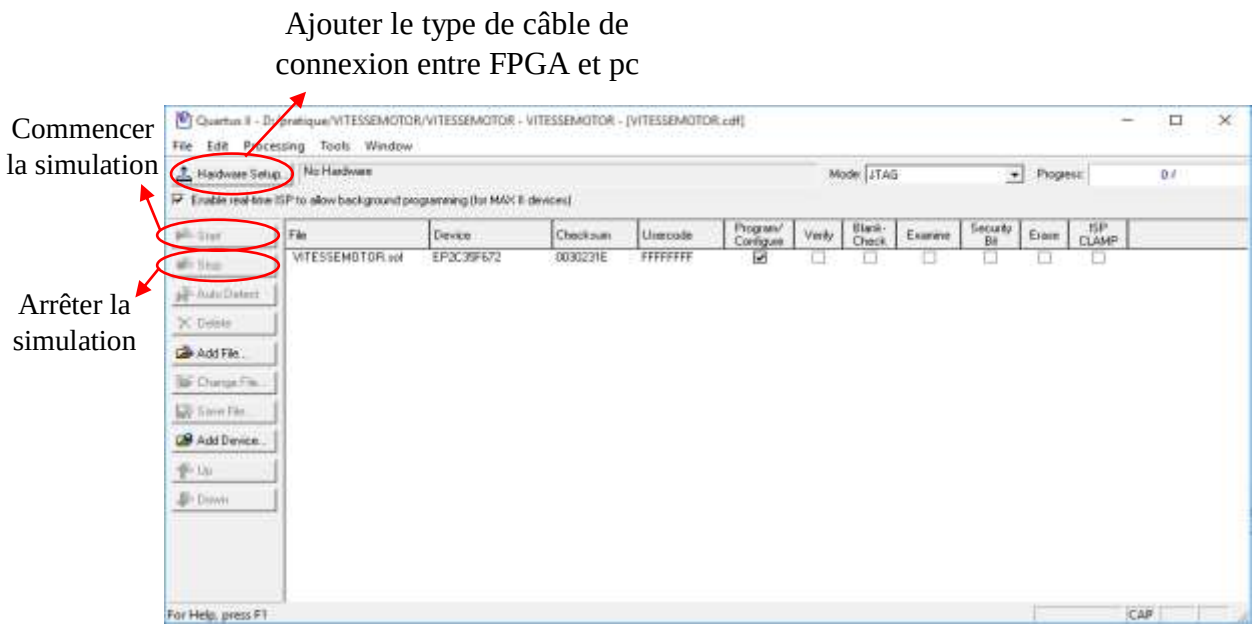


Figure III.18 : Téléversement vers la carte FPGA

III.2.7. Visualisation des signaux de commandes :

Pour visualiser les signaux de commandes générer depuis la carte FPGA on utilisera le logicielle « ModelSim-Altera 6.5b (Quartus II 9.1) Starter Edition » suivant ces étapes

Etape 1 : Création de projet

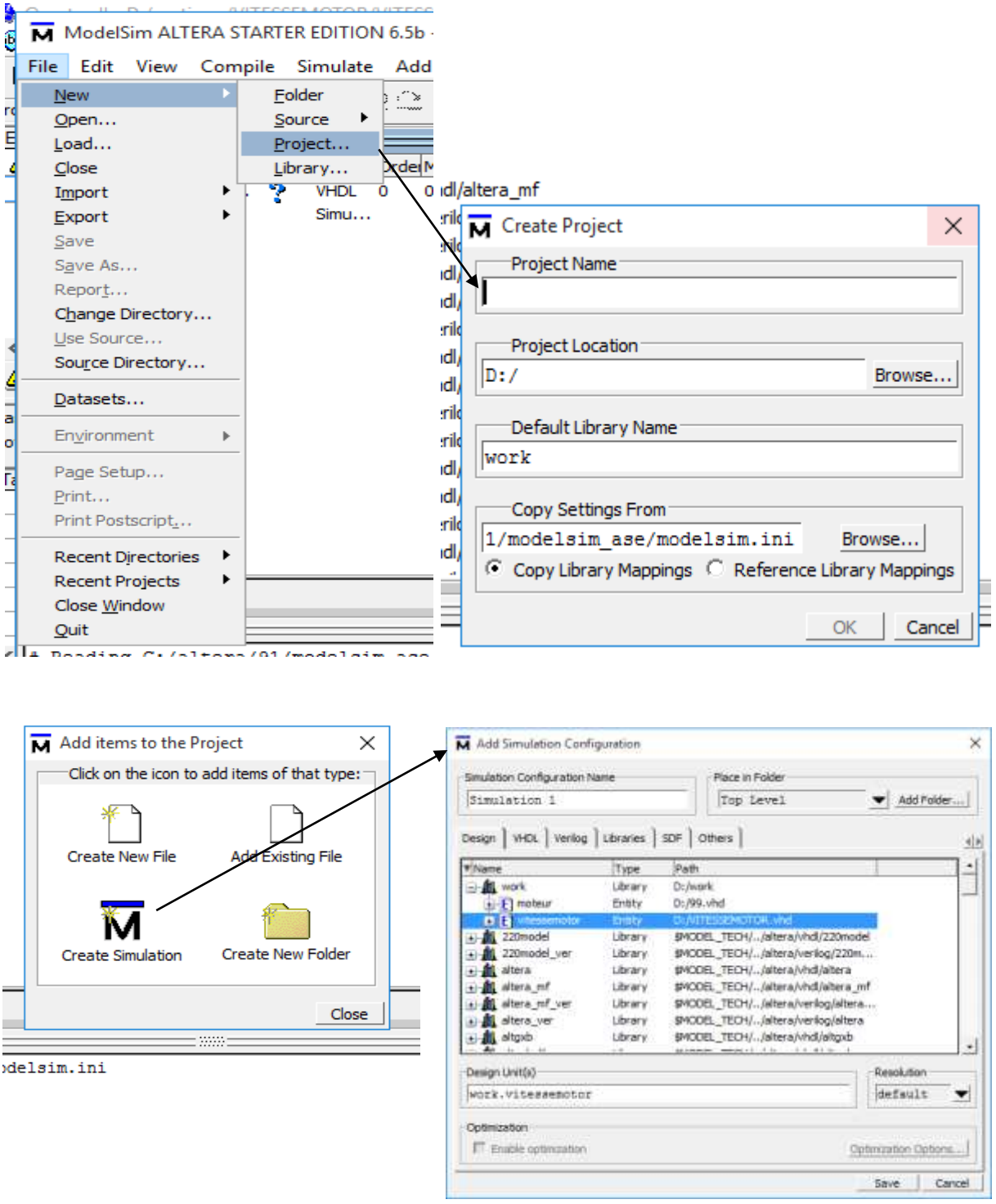


Figure III.19 : Création du projet vhdl et simulation

Etape 2 : Création de projet du projet VHDL et programme nommé VITESSEMOTOR :

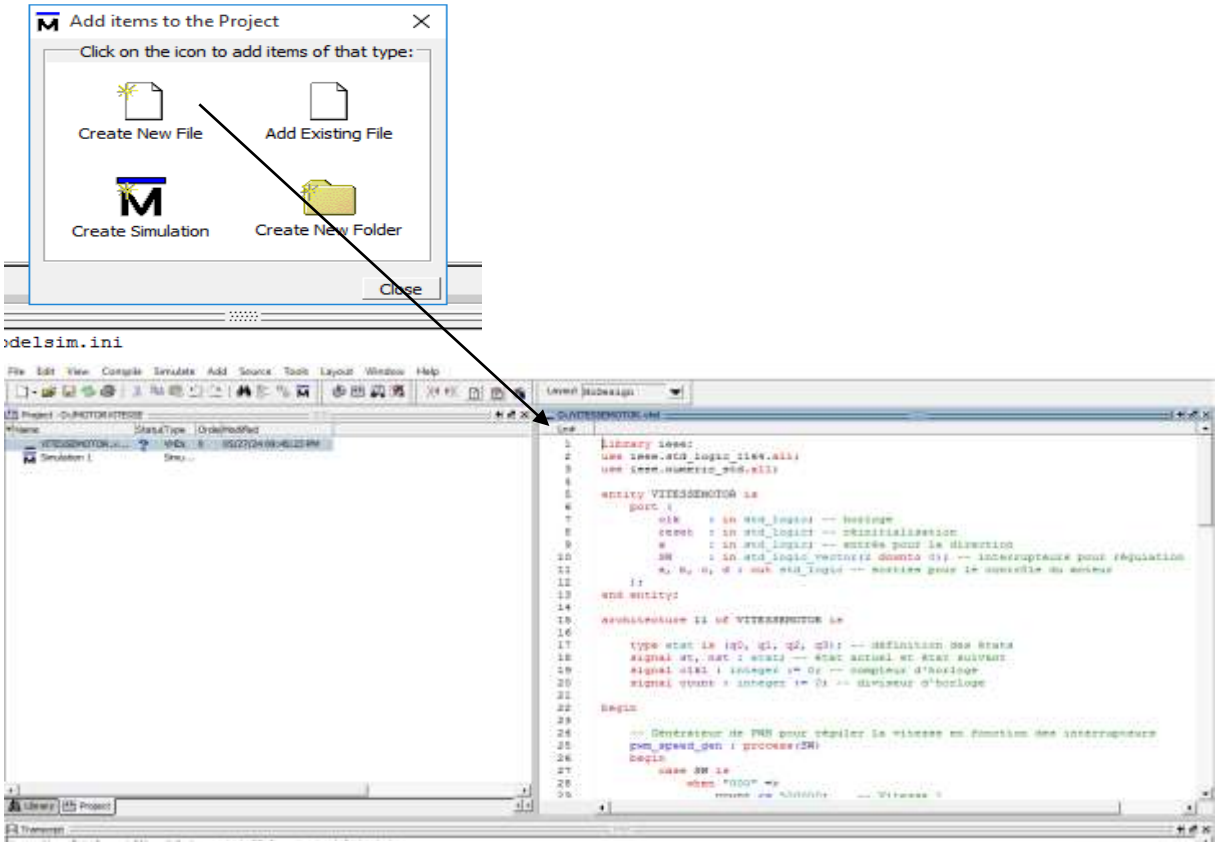


Figure III.20 : Création du projet VHDL

Etape 3 : Compilation du projet :

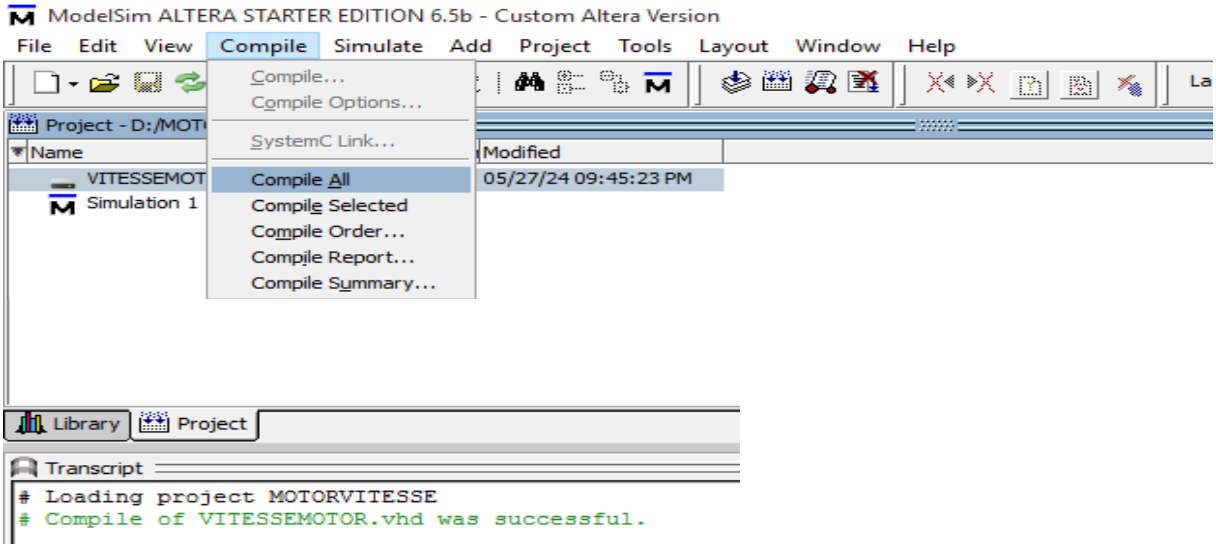


Figure III.21 : Compilation du projet

Etape 4 : Création du projet de simulation et visualisation :

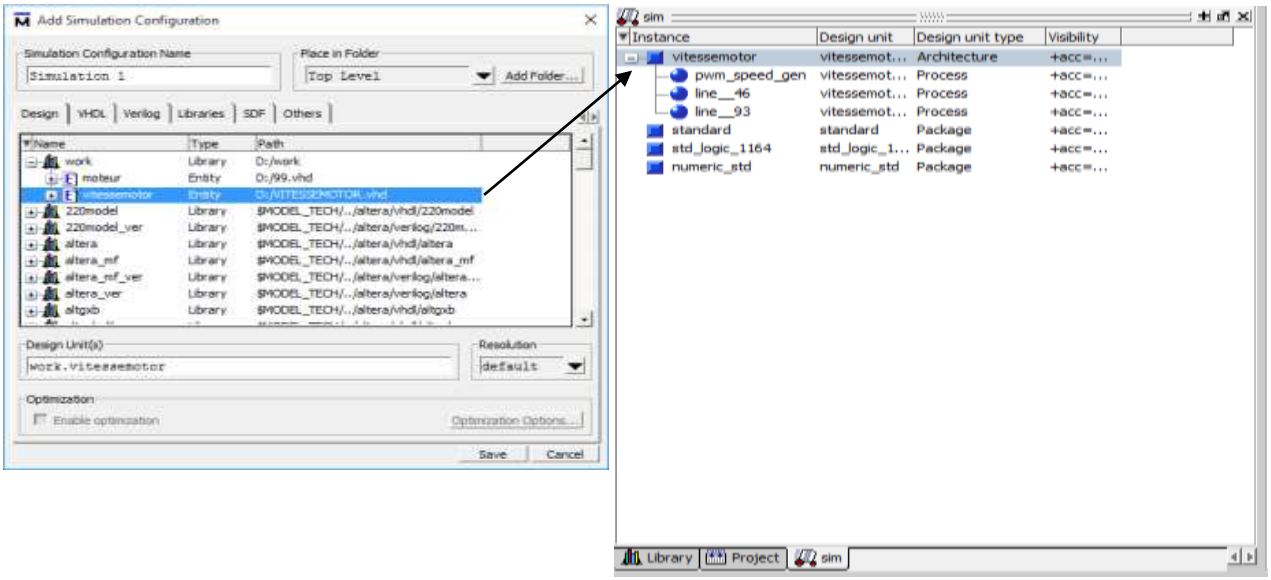


Figure III.22 : Création du projet simulation et visualisation

Etape 5 : Visualisation des signaux :

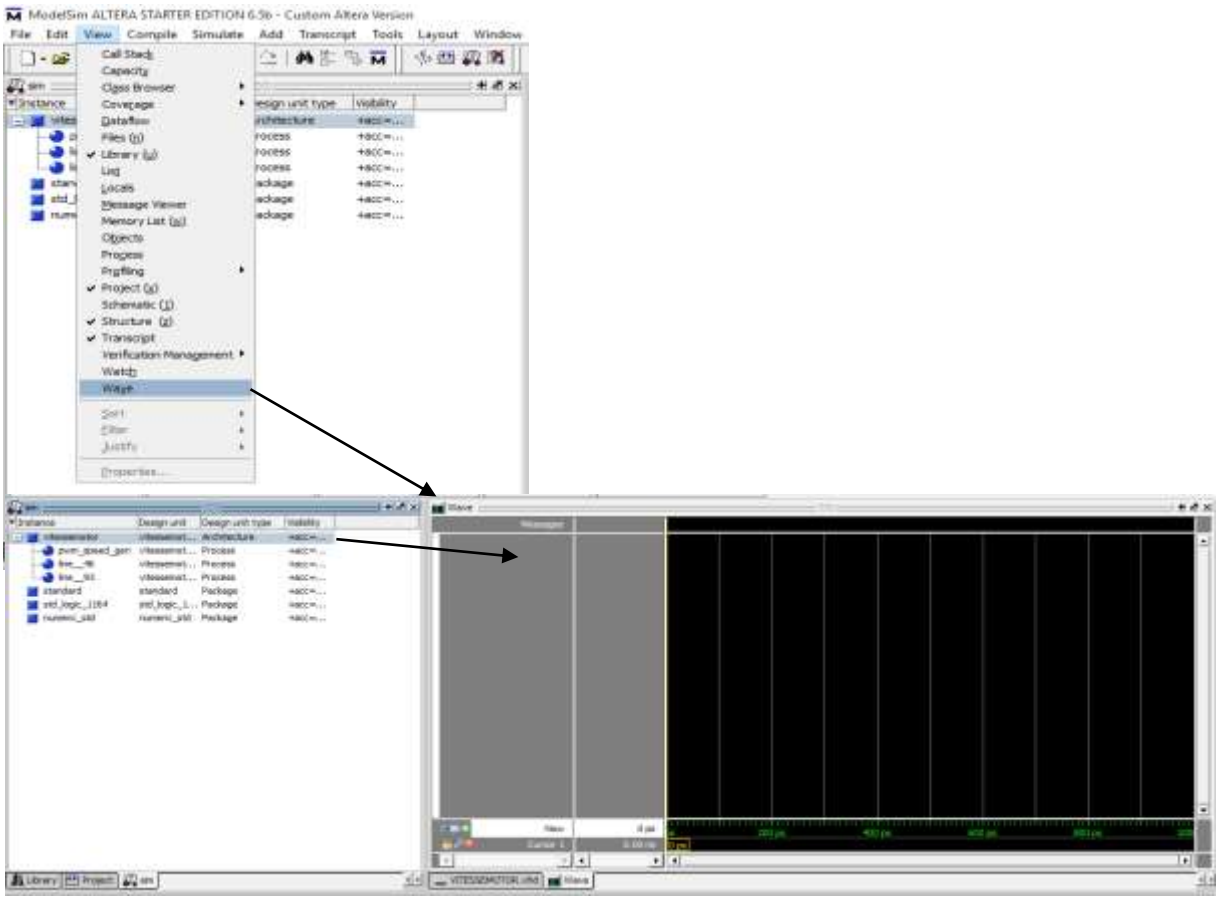


Figure III.23 : Visualisation des signaux

Etape 6 : Remplissage de données :

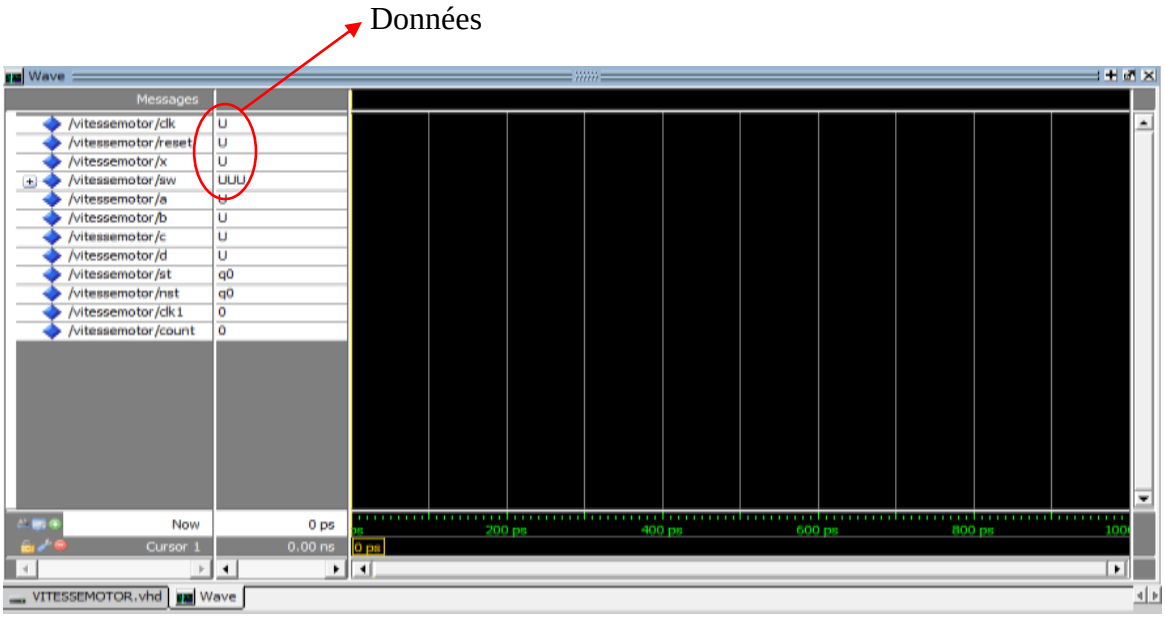


Figure III.24 : Remplissage de données d'entrées sur Model-sim Altera

CLK :

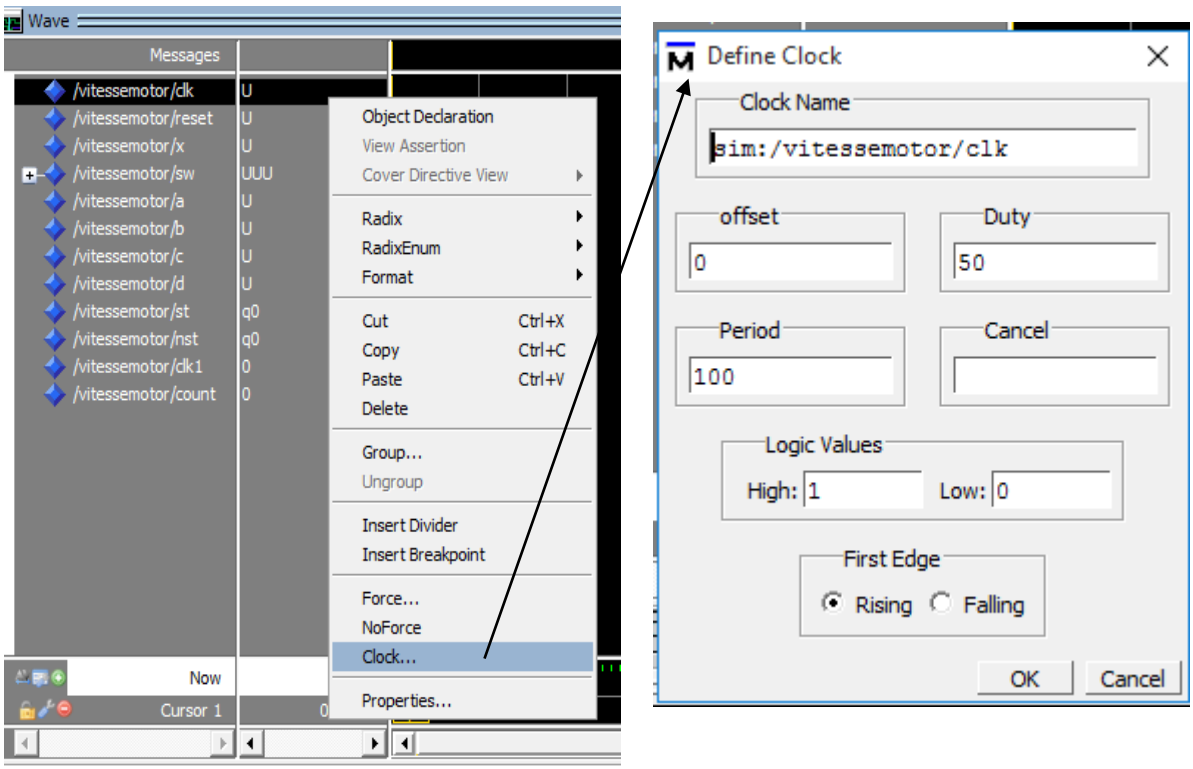


Figure III.25 : Signal d'horloge CLK générer par la carte FPGA

Reset : ‘0’ : Activation, ‘1’ : Désactivation

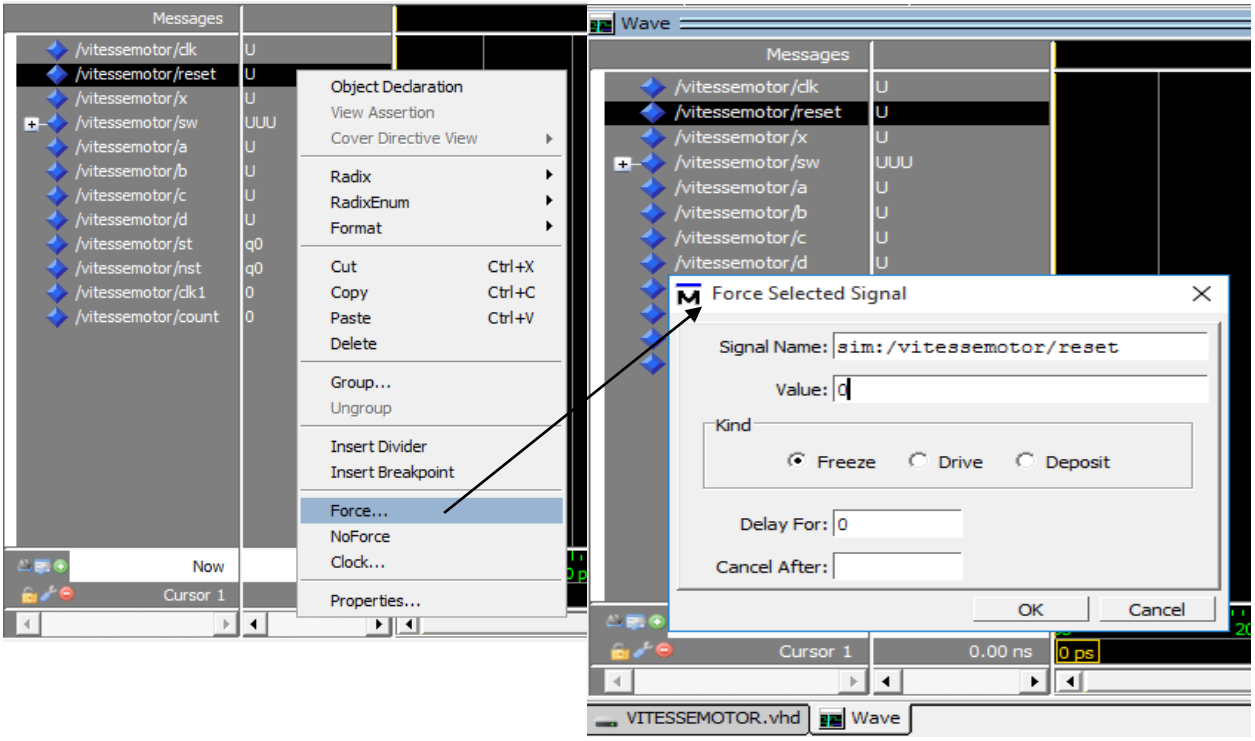


Figure III.26 : Signal d’activation/ Désactivation Reset

X : ‘0’ : Sens antihoraire, ‘1’ : Sens horaire

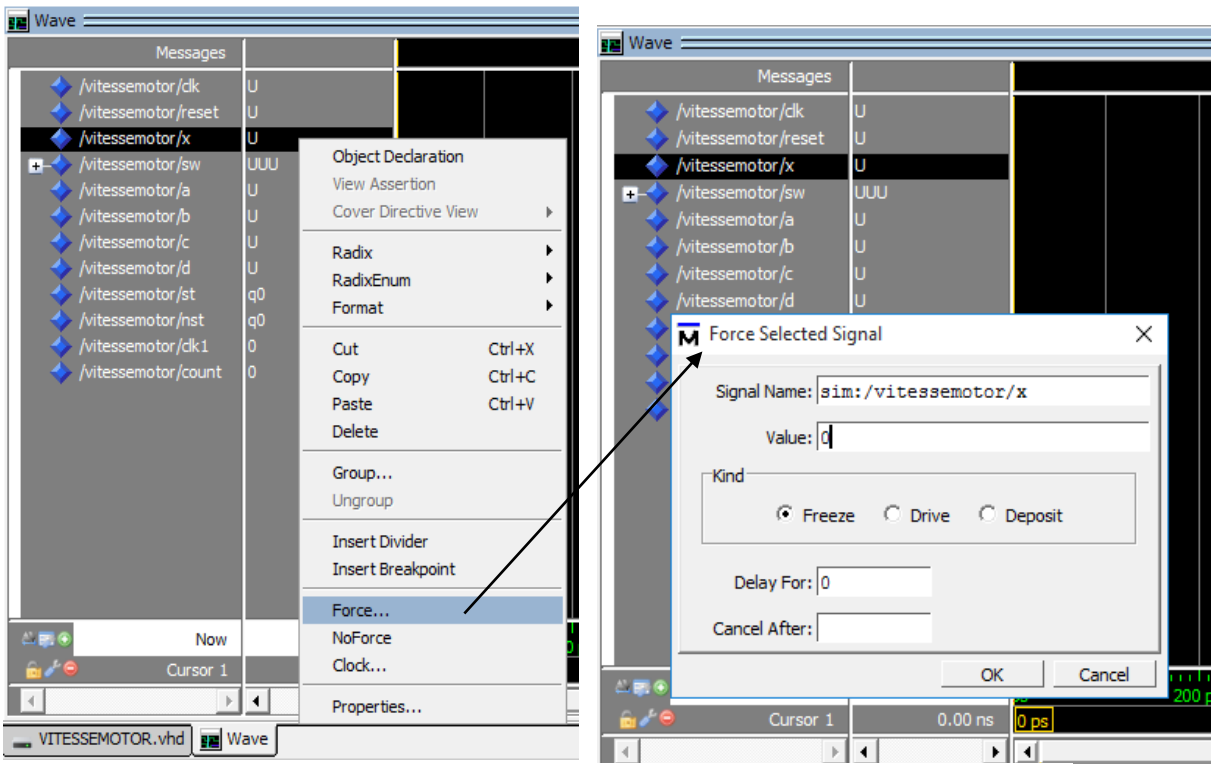


Figure III.27 : Signal de direction X

Sw : valeur correspondante à la vitesse voulue

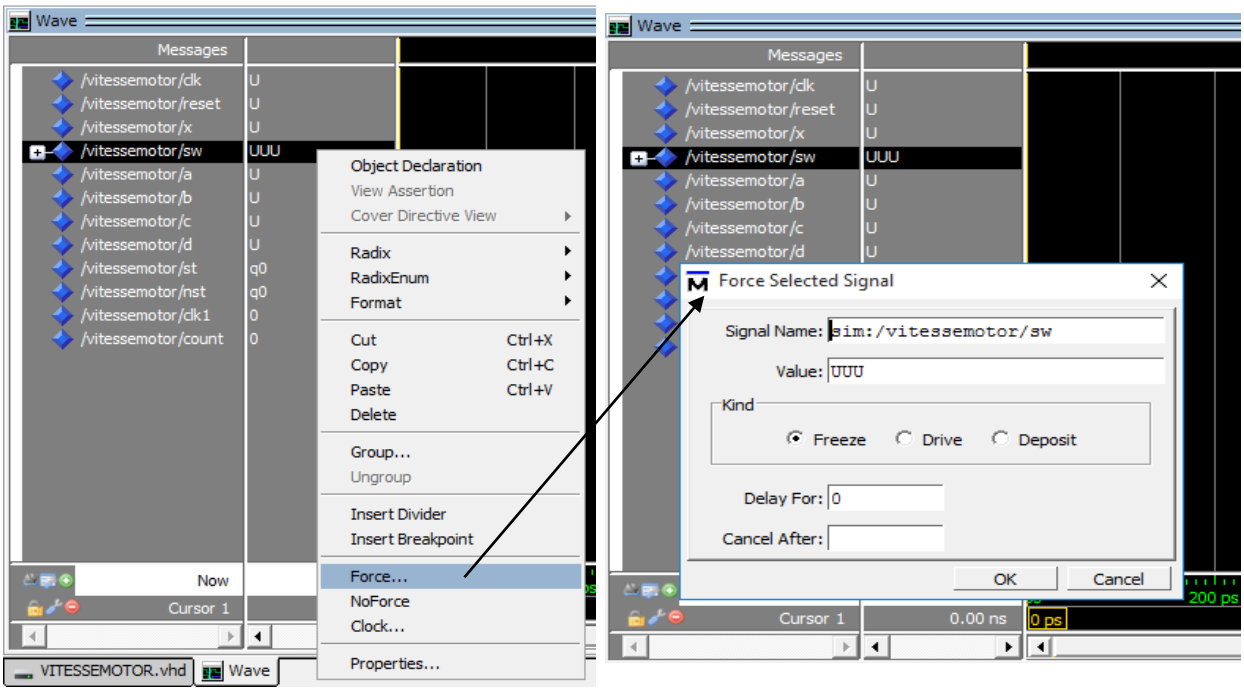


Figure III.28 : Signal de régulation de vitesse Sw

Visualisation des signaux de commandes :

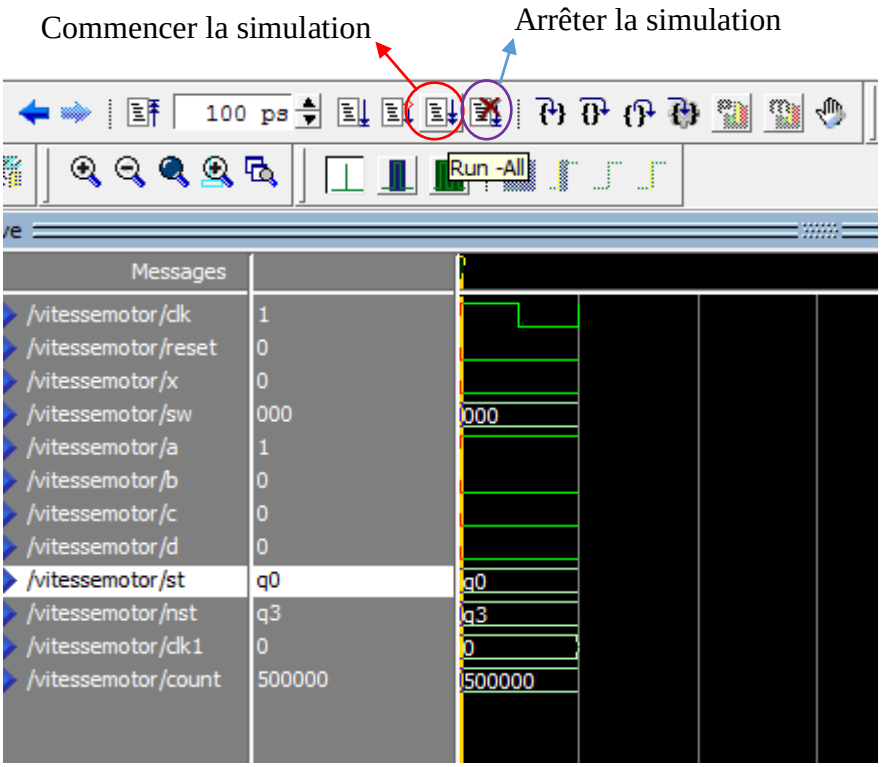


Figure III.29 : Commencer la simulation sur logicielle

Résultats des signaux simulés :

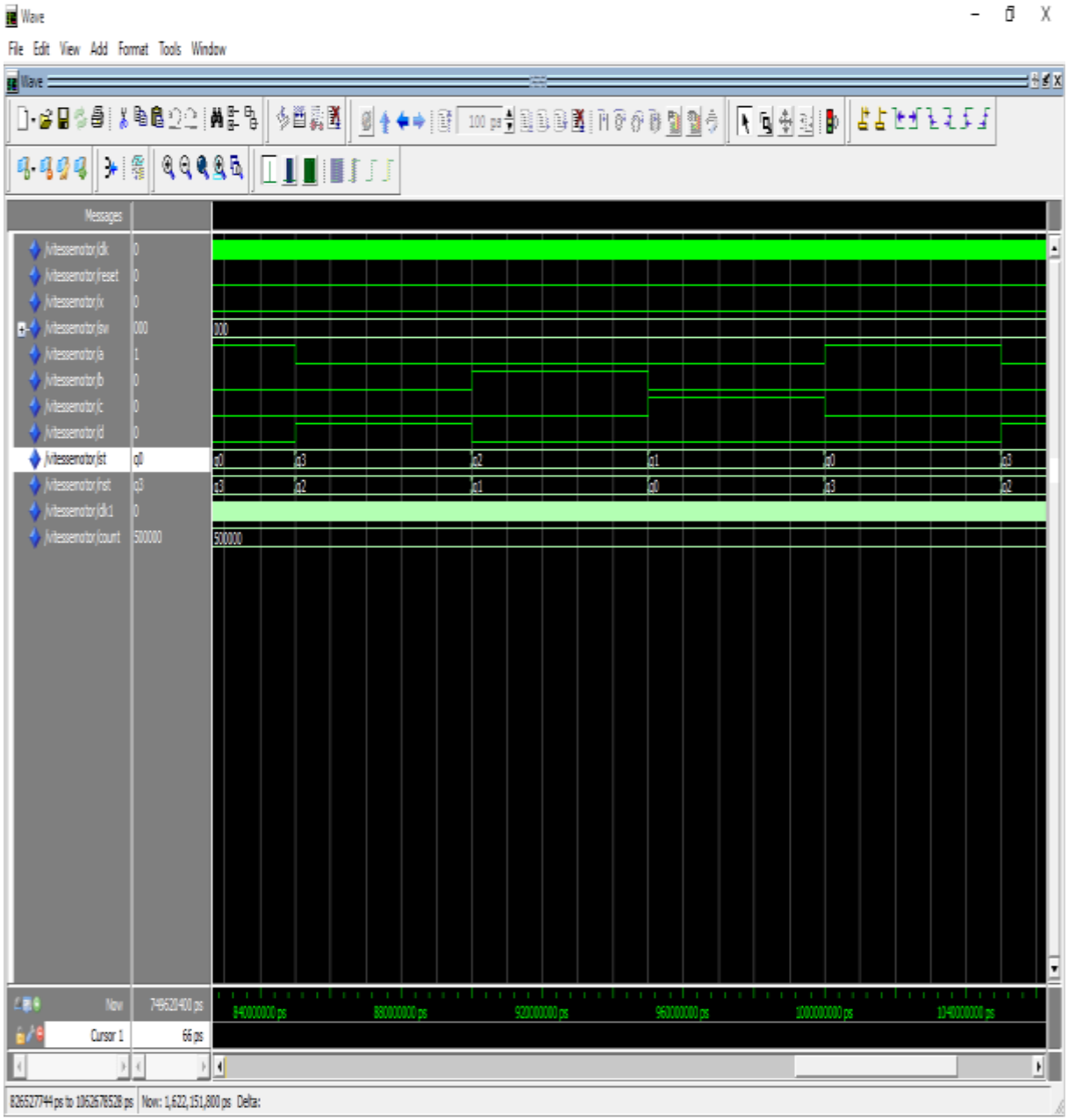


Figure III.30 : Résultats des signaux simulés

Résultats et remarques :

Signal clk, clk1 :

- **Observation** : **clk1** apparaîtra comme un compteur incrémentant à chaque cycle de **clk** jusqu'à atteindre **count - 1**, puis se réinitialisant.

Signal reset :

- **Observation** : Quand le signal **reset** passe à '1' pour réinitialiser le système, puis revenir à '0' pour permettre le fonctionnement normal.

Signal x (Direction) :

- **Observation** : Le signal **x** doit changer en fonction des besoins de direction du moteur (avant ou arrière).
- **Remarque** : Lorsque **x** change, les transitions d'état reflètent correctement ce changement. Par exemple, lorsque **x** passe de '0' à '1', les états doivent progresser dans l'ordre **q0** -> **q1** -> **q2** -> **q3**, et inversement lorsque **x** passe de '1' à '0'.

Signal SW (Interrupteurs de Vitesse) :

- **Observation** : Le signal **SW** change pour modifier la vitesse du moteur.
- **Remarque** : chaque configuration de **SW** (par exemple, **"000"**, **"001"**, **"010"**, **"100"**) modifie correctement le délai (**count**) entre les transitions d'état. Une modification de **SW** doit se traduire par un changement observable dans la fréquence de transition des états **a**, **b**, **c**, **d**.

Sorties a, b, c, d :

- **Observation** : Les sorties **a**, **b**, **c**, **d** suivant un cycle de séquences correct en fonction des états (**q0**, **q1**, **q2**, **q3**).
- **Remarque** : Les séquences des sorties sont conformes aux attentes pour chaque état. Par exemple, dans l'état **q0**, **a** doit être '1' et les autres sorties doivent être '0'.

Organigramme de fonctionnement du programme :

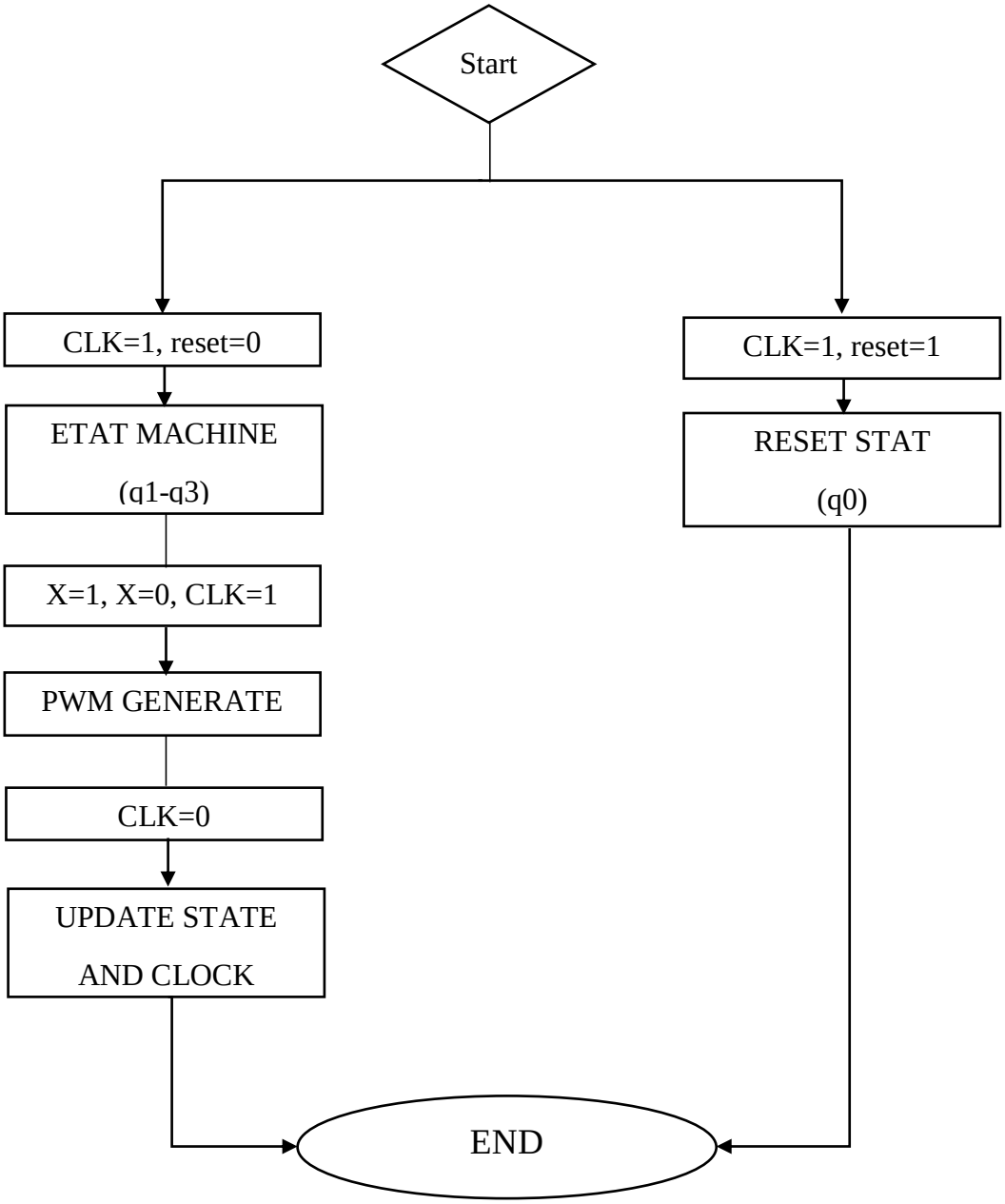


Figure III.4 : Organigramme de fonctionnement du programme

Cet organigramme représente les différentes étapes du programme :

1. **START** : Point de départ du programme.
2. **clk = 1** : Attend que le signal d'horloge soit actif.
3. **reset = 1** : Attend que le signal de réinitialisation soit activé.
4. **Reset State (q0)** : Initialise l'état du système à q0 lorsque reset = 1.

5. **State Machine (q1-q3)** : Gère la logique de transition entre les états q1, q2 et q3 en fonction de l'entrée x.
6. **PWM Generator** : Génère le signal PWM en fonction de la configuration des interrupteurs SW.
7. **Update States & Clock** : Met à jour les états de la machine et le compteur d'horloge en fonction des conditions.
8. **END** : Fin du programme.

Le programme qu'on a utilisé pour commander la vitesse et la direction, Activation /Désactivation du moteur pas à pas à aimant permanant bipolaire est dans l'Annexe A.

III.3. Partie 2 : intégration de l'interface de communication :

Après avoir étudié les différentes méthodes de communication et les principes fondamentaux de l'Arduino, nous allons maintenant aborder l'intégration de ces interfaces dans notre projet. Cette section se concentrera sur la manière d'incorporer efficacement l'interface de communication.

Pour développer le code Arduino qui communiquera avec le FPGA via UART, on doit configurer l'Arduino pour pouvoir envoyer les signaux de contrôles nécessaires au fonctionnement du circuit FPGA. Ces commandes seront reçues par le module UART sur le FPGA et interprétées pour contrôler la vitesse du moteur.

Pour faire une adaptation entre la carte FBGA et ARDUINO on va utiliser un convertisseur ou bien un module RS-485

Le programme qu'on a utilisé pour génération des signaux de commande de vitesse avec adaptation UART avec la carte Arduino est dans l'annexe B.

III.3.2. LE RS-485 :

Est un standard de communication série largement utilisé dans les applications industrielles pour la transmission de données sur de longues distances et dans des environnements bruyants. Conçu pour offrir une communication fiable et robuste, il permet la connexion de plusieurs périphériques sur un même bus de communication.

Figure III.5 : Module de communication RS-485

III.3.2.1. LE ROLE DU RS-485 :

Le rôle du RS-485 dans les systèmes de communication est crucial, en particulier dans les environnements industriels et les réseaux distribués. Voici quelques-uns de ses rôles principaux :

1. **Transmission de données robuste :** RS-485 offre une communication fiable même sur de longues distances et dans des environnements bruyants ou exposés à des interférences électromagnétiques. Cela en fait un choix populaire pour les applications industrielles où la fiabilité est primordiale.
2. **Connectivité multi-nœuds :** RS-485 permet de connecter plusieurs périphériques sur un même bus de communication. Cela signifie qu'un seul câble peut relier plusieurs équipements, ce qui simplifie le câblage et réduit les coûts d'installation.
3. **Flexibilité de topologie :** Le RS-485 prend en charge plusieurs topologies de réseau, notamment les configurations point à point, multipoints et multidrop. Cela permet une grande flexibilité dans la conception des systèmes de communication en fonction des besoins spécifiques de l'application.
4. **Communication bidirectionnelle :** RS-485 permet la transmission bidirectionnelle de données, ce qui signifie que les périphériques connectés peuvent à la fois recevoir et envoyer des données sur le bus de communication. Cela facilite la mise en œuvre de

systèmes de contrôle et de surveillance où les données doivent être échangées entre plusieurs appareils.

5. **Compatibilité avec divers protocoles :** RS-485 est utilisé avec une variété de protocoles de communication, tels que Modbus, Profibus et BACnet, ce qui le rend compatible avec de nombreux équipements industriels et systèmes de contrôle.

III.3.3.Adaptation entre fbga et le rs- 485 :

Une interface de communication est souvent nécessaire pour gérer le flux de données entre le FBGA et le RS-485. Cela peut impliquer l'utilisation d'un contrôleur de communication série, tel qu'une UART (Universal Asynchronous Receiver-Transmitter),

La figure III.7 représente la photographie du système qu'on a réalisé pour générer les signaux de régulation de vitesse depuis la carte Arduino.

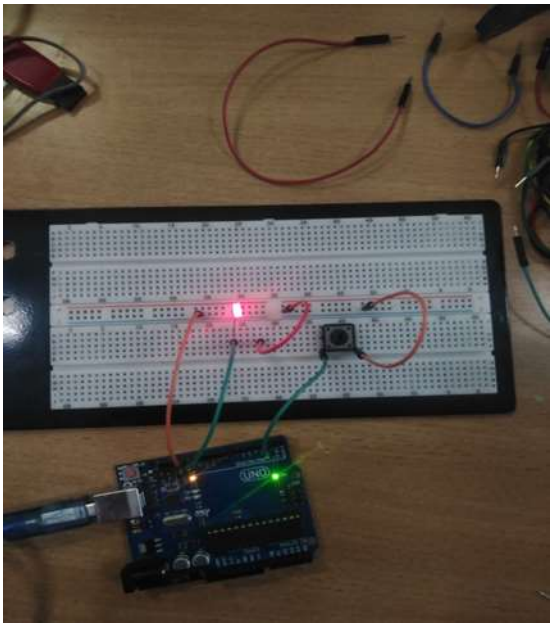


Figure III.6 : Réalisation de circuit de commande via Arduino

III.3.3.1. Explication du fonctionnement du programme :

Ce programme est conçu pour contrôler la vitesse d'un moteur en fonction de l'état d'un bouton poussoir, et il communique également la vitesse sélectionnée via une connexion série (UART). Voici une explication étape par étape du programme :

1. **Définition des broches :**
   - `buttonPin` est défini comme la broche numérique à laquelle le bouton poussoir est connecté (dans ce cas, la broche 2).
   - `ledPin` est défini comme la broche numérique à laquelle la LED est connectée (dans ce cas, la broche 13).
2. **Variables :**
   - `buttonState` stocke l'état actuel du bouton poussoir (HIGH ou LOW).
   - `lastButtonState` stocke l'état précédent du bouton poussoir pour détecter les changements d'état.
   - `motorSpeed` stocke la vitesse actuelle du moteur (un nombre entier de 0 à 5).
3. **Fonction setup() :**
   - Initialise la communication série avec une vitesse de transmission de 9600 bauds.
   - Définit `buttonPin` comme une entrée (INPUT) pour lire l'état du bouton poussoir.
   - Définit `ledPin` comme une sortie (OUTPUT) pour contrôler la LED.
4. **Fonction loop() :**
   - Lit l'état du bouton poussoir et le stocke dans `buttonState`.
   - Vérifie si l'état du bouton poussoir a changé depuis la dernière itération de la boucle.
   - Si le bouton est enfoncé (état HIGH) et qu'il y a eu un changement, la vitesse du moteur est incrémentée de 1 et la nouvelle vitesse est envoyée via la fonction `sendMotorSpeed()`. La LED est également allumée.
   - Si le bouton est relâché (état LOW), la LED est éteinte.

- Met à jour `lastButtonState` avec l'état actuel du bouton pour la prochaine itération.

5. **Fonction sendMotorSpeed(int speed) :**
   - Cette fonction prend la vitesse du moteur en entrée et envoie une chaîne de caractères correspondant à cette vitesse via la communication série.
   - Les différentes vitesses sont représentées par des motifs de bits uniques :
     - 0 : "00000001"
     - 1 : "00000010"
     - 2 : "00000011"
     - 3 : "00000100"
     - 4 : "00000101"
     - 5 : "00000110"
   - Si une vitesse invalide est reçue, la fonction envoie la vitesse par défaut (0).

Ce programme utilise la communication série pour envoyer des informations sur la vitesse du moteur à un autre périphérique, tandis que le bouton poussoir est utilisé pour contrôler la vitesse du moteur localement.

Résumé

1. Matériel :

- On va utiliser des câbles de connexion pour établir les connexions entre l'Arduino et le FPGA et un convertisseur de niveau logique si nécessaire pour adapter les tensions entre l'Arduino et le FPGA.

2. Code Arduino :

- on va utiliser l'interface UART pour envoyer des commandes au FPGA.

3. Code FPGA :

- Utilisation un module UART pour recevoir les commandes de l'Arduino.

- Intégration des commandes dans la logique de contrôle du moteur pour ajuster la vitesse du moteur.

Le programme qu'on a utilisé pour FPGA Réceptrice des commandes de la carte Arduino est dans l'annexe C.

III.3.4. Fonctionnement du projet :

Le but de ce projet est d'envoyer des signaux de commandes depuis la carte ARDUINO qu'on va la prendre comme IMPRIMANTE 3D ces signaux vont être générer vers la carte FPGA qui contrôlera le moteur pas à pas à aimant permanant bipolaire dans notre cas on a pu générer un signal de commande de vitesse avec un bouton poussoir.

III.4. Discussion :

Dans ce chapitre nous avons pu contrôler en mode pas complet : la vitesse, la direction ainsi que l'activation / désactivation du moteur pas à pas à aimant permanant bipolaire que nous avons utilisé via la carte FPGA DE2, aussi nous avons pu visualiser les signaux de commandes générés depuis la carte FPGA. Pour la deuxième partie de ce travail qui consiste à générer les signaux de commandes depuis une carte ARDUINO UNO R3 vers la carte FPGA, nous avons réussi le développement du programme associé pour la génération des signaux de commandes du moteur sauf que pour leur mise en pratique nous n'avons pas pu s'en procurer le module de communication RS-485.

Annexe A

Programme commande du moteur pas à pas avec la carte FPGA DE2 :

```
entity VITESSEMOTOR is
   port (
      clk    : in std_logic; -- horloge
      reset  : in std_logic; -- réinitialisation
      x      : in std_logic; -- entrée pour la direction
      SW     : in std_logic_vector(2 downto 0); -- interrupteurs pour régulation de vitesse
      a, b, c, d : out std_logic -- sorties pour le contrôle du moteur
   );
end entity;

architecture ll of VITESSEMOTOR is

   type etat is (q0, q1, q2, q3); -- définition des états
   signal st, nst : etat; -- état actuel et état suivant
   signal clk1 : integer := 0; -- compteur d'horloge
   signal count : integer := 0; -- diviseur d'horloge

begin

   -- Générateur de PWM pour réguler la vitesse en fonction des interrupteurs
   pwm_speed_gen : process(SW)
   begin
      case SW is
         when "000" =>
            count <= 500000;    -- Vitesse 1
         when "001" =>
            count <= 300000;    -- Vitesse 2
         when "010" =>
            count <= 200000;    -- Vitesse 3
```

```
        when "100" >
            count <= 150000;    -- Vitesse 4
        when others =>
            count <= 500000;    -- Valeur par défaut
    end case;
end process;

-- Logique de contrôle des états
process (x, st)
begin
    case st is
        when q0 =>
            a <= '1';
            b <= '0';
            c <= '0';
            d <= '0';
            if x = '1' then
                nst <= q1;
            else
                nst <= q3;
            end if;
        when q1 =>
            a <= '0';
            b <= '0';
            c <= '1';
            d <= '0';
            if x = '1' then
                nst <= q2;
            else
                nst <= q0;
```

```
            end if;
        when q2 =>
            a <= '0';
            b <= '1';
            c <= '0';
            d <= '0';
            if x = '1' then
                nst <= q3;
            else
                nst <= q1;
            end if;
        when q3 =>
            a <= '0';
            b <= '0';
            c <= '0';
            d <= '1';
            if x = '1' then
                nst <= q0;
            else
                nst <= q2;
            end if;
    end case;
end process;

-- Processus d'horloge pour la mise à jour des états
process (reset, clk)
begin
    if reset = '1' then
        st <= q0;
        clk1 <= 0;
```

```
        elsif rising_edge(clk) then
            clk1 <= clk1 + 1;
            if clk1 = count - 1 then -- diviseur d'horloge pour ajuster la vitesse
                st <= nst; -- mise à jour de l'état
                clk1 <= 0;
            end if;
        end if;
    end process;
end ll;
```

Annexe B

Programme Arduino génération des signaux de commande de vitesse avec adaptation UART :

```
const int buttonPin = 2;

const int ledPin =  13;


int buttonState = 0;

int lastButtonState = 0;

int motorSpeed = 0;


void setup() {

  Serial.begin(9600);

  pinMode(buttonPin, INPUT);

  pinMode(ledPin, OUTPUT);

}


void loop() {

  buttonState = digitalRead(buttonPin);


  if (buttonState != lastButtonState) {

    if (buttonState == HIGH) {

      motorSpeed = (motorSpeed + 1) % 6;

      sendMotorSpeed(motorSpeed);

      digitalWrite(ledPin, HIGH);

    } else {
```

```
      digitalWrite(ledPin, LOW);
    }
    lastButtonState = buttonState;
  }

  delay(50);
}

void sendMotorSpeed(int speed) {
  switch (speed) {
    case 0:
      Serial.println("00000001");
      break;
    case 1:
      Serial.println("00000010");
      break;
    case 2:
      Serial.println("00000011");
      break;
    case 3:
      Serial.println("00000100");
      break;
    case 4:
      Serial.println("00000101");
```

```
    break;
  case 5:
    Serial.println("00000110");
    break;
  default:
    Serial.println("00000001");
    break;
}
```

Annexe C

Programme HDL adaptation avec la communication via UART :

Module UART Receiver:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uart_rx is
    Port (
        clk       : in std_logic;
        reset     : in std_logic;
        rx        : in std_logic;
        data_out  : out std_logic_vector(7 downto 0);
        data_ready : out std_logic
    );
end uart_rx;

architecture Behavioral of uart_rx is
    constant baud_rate       : integer := 9600;
    constant clk_freq        : integer := 50000000; -- fréquence d'horloge en Hz
    constant baud_period     : integer := clk_freq / baud_rate;

    signal rx_shift_reg      : std_logic_vector(7 downto 0);
    signal bit_counter       : integer range 0 to 7 := 0;
```

```
    signal baud_counter      : integer := 0;
    signal receiving         : std_logic := '0';
    signal data_ready_reg    : std_logic := '0';


begin
    process(clk, reset)
    begin
        if reset = '1' then
            rx_shift_reg <= (others => '0');
            bit_counter <= 0;
            baud_counter <= 0;
            receiving <= '0';
            data_ready_reg <= '0';
        elsif rising_edge(clk) then
            if baud_counter = baud_period - 1 then
                baud_counter <= 0;
                if receiving = '1' then
                    if bit_counter = 7 then
                        receiving <= '0';
                        data_ready_reg <= '1';
                    else
                        bit_counter <= bit_counter + 1;
                        rx_shift_reg(bit_counter) <= rx;
                    end if;
```

```
            elsif rx = '0' then -- start bit detected
                receiving <= '1';
                bit_counter <= 0;
            end if;
        else
            baud_counter <= baud_counter + 1;
        end if;
    end if;
end process;

data_out <= rx_shift_reg;
data_ready <= data_ready_reg;
end Behavioral;
```

Intégration du module UART avec le contrôle du moteur :

Voici comment intégrer le module `uart_rx` avec le code de contrôle du moteur :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity VITESSEMOTOR is
    port (
```

```
        clk, reset, rx : in std_logic; -- entrées
        SW : in std_logic_vector(2 downto 0); -- switches pour la régulation de vitesse
        a, b, c, d : out std_logic -- sorties
    );
end entity;

architecture ll of VITESSEMOTOR is
    signal d1, d2, d3, q1, q2, q3 : std_logic;
    signal data_out : std_logic_vector(7 downto 0);
    signal data_ready : std_logic;
    signal x : std_logic := '0';
    signal count : integer := 0;

    component uart_rx
        port (
            clk : in std_logic;
            reset : in std_logic;
            rx : in std_logic;
            data_out : out std_logic_vector(7 downto 0);
            data_ready : out std_logic
        );
    end component;

begin
```

```
uart_rx_inst : uart_rx
    port map (
        clk => clk,
        reset => reset,
        rx => rx,
        data_out => data_out,
        data_ready => data_ready
    );

process(data_ready)
begin
    if data_ready = '1' then
        case data_out is
            when "00000001" => x <= '1';
            when "00000010" => x <= '0';
            when others => null;
        end case;
    end if;
end process;

pwm_speed_gen : process(SW)
begin
    case SW is
        when "000" => count <= 500000; -- Vitesse 1
```

```
        when "001" => count <= 300000; -- Vitesse 2
        when "010" => count <= 200000; -- Vitesse 3
        when "011" => count <= 150000; -- Vitesse 4
        when "100" => count <= 200000; -- Vitesse 5
        when "101" => count <= 15625; -- Vitesse 6
        when others => count <= 500000; -- Valeur par défaut
    end case;
end process;

process(st)
begin
    case st is
        when q0 =>
            a <= '1';
            b <= '0';
            c <= '0';
            d <= '0';
            if x = '1' then nst <= q1; else nst <= q3; end if;
        when q1 =>
            a <= '0';
            b <= '0';
            c <= '1';
            d <= '0';
            if x = '1' then nst <= q2; else nst <= q0; end if;
```

```
        when q2 =>
            a <= '0';
            b <= '1';
            c <= '0';
            d <= '0';
            if x = '1' then nst <= q3; else nst <= q1; end if;
        when q3 =>
            a <= '0';
            b <= '0';
            c <= '0';
            d <= '1';
            if x = '1' then nst <= q0; else nst <= q2; end if;
    end case;
end process;

process(reset, clk)
begin
    if reset = '1' then
        st <= q0;
        clk1 <= 0;
    elsif rising_edge(clk) then
        clk1 <= clk1 + 1;
        if clk1 = count - 1 then
            st <= nst; -- calcul de l'état futur
```

```
            clk1 <= 0;
        end if;
    end if;
  end process;


end ll;
```

Conclusion Générale

Conclusion Générale

Tout au long de ce projet on a exploré en profondeur l'utilisation de la technologie FPGA (Field Programmable Gate Array) pour la commande de moteurs pas à pas, avec une application spécifique aux imprimantes 3D. Nous avons examiné les caractéristiques des FPGA, les différents types de moteurs pas à pas, et l'intégration de ces technologies dans des systèmes de contrôle avancés.

Dans le cadre de cette thèse, nous avons pu exploré les différentes étapes nécessaires à la réalisation d'un système de contrôle d'un moteur pas à pas à aimant permanent, pour une imprimante 3D, en utilisant une carte « FPGA DE2 » en utilisant le logiciel « Quartus II 9.1sp2 Web Edition », et aussi nous avons pu générer des signaux de contrôle à partir d'une carte « Arduino uno » en remplacement du contrôleur habituellement utilisé dans les imprimante 3D pour la génération es instructions nécessaires aux contrôle des moteurs pas à pas associés.

Nous avons pu développer des algorithmes de commande robustes et l'avons implémenté en utilisant des langages de description matérielle telle que Le « VHDL ».

Les résultats expérimentaux ont validé la capacité de notre système à commander le moteur pas à pas avec la précision désirée.

Cette thèse ouvre la voie à plusieurs perspectives de recherche, telle que le contrôle de plusieurs moteurs pas à pas en temps réel avec plus de précision en utilisant le contrôle des moteurs pas à pas par micro pas.

Bibliographie

Bibliographie

[1] Mihai Bogdan Luca. << Apports du chaos et des estimateurs d'états pour la transmission sécurisée de l'information>>. Thèse de doctorat. Laboratoire d'Electronique et Systèmes de Télécommunications LEST - UMR CNRS 6165. Université de Bretagne Occidentale - Ecole Doctorale SMIS(2006)

[2] M. Walid, M.I.Moctar. Implémentation d'un modulateur OFDM sur un circuit FPGA. Mémoire de Fin d'Etudes, Electronique. Alger : Ecole Nationale Polytechnique (ENP), 2007.

[3] B.Farida, D.Sabrina. Système temps réel embarque pour commander un robot mobile. Mémoire de Fin d'Etudes, Automatique. Alger : Ecole Nationale Polytechnique (ENP), 2005.

[4] Eduardo Sanchez, << Les circuits logiques programmables>>, Ecole Polytechnique Fédérale de Lausanne. [8] Eduardo Sanchez,<< Circuits reconfigurables: Les FPGAs>>.

[5] D.Houssey. Implémentation d'une commande MPPT floue sur FPGA. Mémoire de Fin d'Etudes, Electronique. Alger : Ecole Nationale Polytechnique (ENP), 2006.

[6] B. Zeidman, "DesigningwithFPGAs and CPLDs", CMP Books, 2002.

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

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

[9] LAURENT DUTRIEUX et DIDIER DEMIGNY ; logique programmable Architecturedes FPGA et CPLD méthodes de conception le langage VHDL ; édition Eyrolles 1997.

[10] T.L.Carrol and L.M.Pocora, Synchronisation in chaotic systems, Physicals review and letters, 1990.

[11] Pierre Moller. Anciennement Ancien DG d'entreprise technbologique

[12] Paul Sente. Master en Ingénieurs électriciens et Électronique, Université catholique de Louvain (Diplôme obtenu en 1978) .

[13] Community.fs.com.

[14] S Bilavarn - 2002 - hal.science

[15] Olivier SENTIEYS, Arnaud TISSERAND. Techniques de l'Ingénieur.

[16] Dr. MOHAMMED ZAKARYA BABA-AHMED

[17] le langage de description VHDL, T. BLOTIN Lycée Paul-Eluard 93206 SAINT-DENIS.

[18] **:**N.R.Galagher, « Système de communication hout débit sécurisé par chaos et intensité », Thèse doctorat, Université de Franche-Comté, (2006).

[19] M. Kercha, "Commande par mode de glissement d'un Moteur pas à pas à Aimant Permanent", Mémoire de Magister, Université de Batna, 2005.

[20] TRAITE D 'ELECTRICITE ELECTROMECANIQUE Volume IX Auteur : Prof. Marcel Jufer ISBN 2-88074-285-4

[21] O. Safi et N. Arkoub, "Commande par modes glissants d'un moteur pas à pas", Mémoire d'Ingénieur d'état, Université Mouloud Mammeri de Tizi Ouzou, 2005.

[21] F. Nollet, "Lois de commande par modes glissants du moteur pas à pas", Thèse de Doctorat, Ecole Centrale de Lille, 2006.

[22] http://www.positron-libre.com/electronique/moteur-pas-a-pas/sequence-commande moteur-pas-a-pas PHP.

[23] Takashi Kenjo and Akira Sugawara, «stepping motors and their microprocessor controls » Oxford University Press, ISBN 0-19-859385-6.

[24] https://www.futura-sciences.com/tech

[25] https://www.materiel.net/guide-achat/g24

[26] https://www.primante3d.com/principe/

[27] S Ali Ahmed, A Hadid - 2015 - ummto.dz

[28] https://www.imprimeren3d.net/

[29] https://www.electronicshub.org/stepper-motor-control-techniques/

[30] https://www.electronicshub.org/arduino-introduction/#Introduction

[31] https://www.instructables.com/id/Hardware-Structure-of-ARDUINO-UNO/

[32] https://www.fpga4student.com/2017/12/how-to-interface-mouse-with-FPGA.html?m=1

[33] http://electroniqueveynes.free.fr/IMG/pdf/SPI_Eleves.pdf

[34] http://ressource.electron.free.fr/bts/b5/83c552/i2c.htm

[35] www.moussasoft.com/les-protocoles-de-communication-spi-i2c-et-uart/