REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l`Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri de Tizi-Ouzou

Faculté de génie électrique et informatique
Département d'électrotechnique
Spécialité : Entraînements Electriques

Mémoire de fin d'étude
en vue de l'obtention du diplôme
de Master en Entraînements Electriques

Thème

Implémentation d'un algorithme de commande d'un dispositif électromagnétique sur FPGA.

Proposé par :

Pr. MOHELLEBI Hassane

Réalisé par :

Mr. RAHEM Rabah

Dirigé par :

Mr. OULD OUALI SamyHassani

Mr. ACHOUR Hakim

Année universitaire 2011/2012

Remerciements

Je veux remercier ici tous ceux qui, de près ou de loin, ont contribués à l'aboutissement de ce travail. Qu'ils trouvent dans ce mémoire toute ma reconnaissance. Aussi, je voudrais citer :

Je remercie vivement Monsieur MOHELLEBI Hassane, Professeur à l'université Mouloud Mammeri de Tizi-ouzou, de m'avoir proposé et de m'avoir suivi durant ce travail de si près, avec une qualité remarquable.J'ai grandement apprécié ses compétences et qualités scientifiques ainsi que son intelligence.

Je tiens aussi à remercier Monsieur OULD OUALI Samy Hassani, Maitre de conférence classe B, d'avoir dirigé ce travail, avec une grande disponibilité et une attention minutieuse accordés à l'avancement des travaux.

Je tiens aussi à remercier :

- **Monsieur DAOUI Mhamed, Maitres de conférences classe B, département Informatique.**
- **Monsieur ACHOUR Hakim, Maitres de conférences classe A, département Automatique.**
- **Monsieur BERCHICHE Madjid, responsable du laboratoire maquette FGEI.**

Je saisis aussi cette occasion pour remercier l'ensemble des enseignants du département d'électrotechnique option Entraînements électriques ayant consacré leurs vies sur la voie noble de l'enseignement.

Dédicace

Je dédié ce modeste travail à :

- **Mes parents avec lesquels j'ai le plaisir de partager ces moments, et je les remercié pour m'avoir guidé tout ce chemin ;**
- **Ma sœur et mes frères ;**
- **Mes oncles et tantes ;**
- **Tous mes amis(es) ;**
- **A tous les enseignants qui ont contribué à ma formation, tous cycles confondus, chacun à son nom.**

Sommaire

Sommaire

Chapitre II : Langage de description VHDL

Liste de figures

Chapitre I

Chapitre II

Liste de figures

Liste de tableaux

Chapitre I

Chapitre III

Nomenclature

HDL : Hardware Description Language (langage de description matériel).

VHDL: Very high Speed Integrated Circuit Hardware Description Language.

VHDL-AMS: Very High Speed Integrated Circuits Hardware Description Language For Analog And Mixed Signal.

RTL: Register Transfer Level.

FPGA: Field Programmable Gate Array (réseau de cellules logiques programmables).

ASIC : Circuit logique programmable lors de sa fabrication.

PAL: Programmable Array Logique (réseau logique programmable)

GAL: Generique Array Logic (réseau logique générique)

EPLD: Erasable Programmable logique Device,

EEPROM ou E2PROM : Electrical Erasable Programmable Read Only Memory (Mémoire effaçable électriquement à lecture seule).

RAM : Random Acces Memory (mémoire à lecture aléatroire).

OLMC: Out Put Logic Macro Cell

MAX: Multiple Array matriX

OE: Output Enable (validation des sorties).

CLB: Configurable Logic Bloc.

LUT: Lookup Table.

IOB: Input Output Bloc.

LE: Logic Element.

LAB: Logic Array Bloc.

NETLIST : Les interconnexions entre les composants qui interviennent avant le routage.

NGD: XILINX Native Database.

Nomenclature

NGC: XILINX Specific file.

TCL: Tool Command Language

ISE: Integrated Software Environment.

JTAG: Joint Test Action Group

JDEC: Joint Electron Device Council.

BITSTREAM: Fichier de programmation des FPGA au format JDEC.

I2C ou IIC: Inter Integrated Circuits.

SDA: Serial Data.

SCL: Serial Clock.

Introduction générale

Introduction générale

Les progrès réalisés en micro-électroniques permettent aujourd'hui d'intégrer plusieurs centaines de millions de transistors sur un seul circuit, cette capacité d'intégration augmentant exponentiellement selon la loi de Moore.

Ces évolutions technologiques ont particulièrement favorisé la famille des circuits programmables FPGA (Field Programmable Gate Array) qui sortent aujourd'hui de leur niche de marché originale pour se poser comme un alternatif autre circuit dédiés ASIC (Application Specific Integrated Circuit), ces dernières années on assiste ainsi à l'émergence de nouvelles générations de circuits programmables, qui offrent des densités de plusieurs dizaines de millions de portes logiques, et intègrent des cœurs de processeurs hautes performance. [22]

Grace à ces avancées spectaculaires de ces circuits, des nombreuses applications complexes qui relèvent de différents domaines et particulièrement la commande, ont pu voir le jour [23].

L'objectif de ce mémoire est d'implémenter un algorithme de commande d'un dispositif électromagnétique sur un circuit FPGA.

Dans ce contexte ce mémoire est scindé en quatre chapitres :

- Le premier chapitre est devisé en deux parties, la première est un rappel sur la logique combinatoire et séquentielle comme étant la base des circuits programmables, la deuxième c'est des généralités sur les circuits ASIC, en particulier les FPGA.
- Dans le deuxième chapitre on donne une présentation détaillée du langage VHDL et on clôturera ce chapitre par donner un aperçu sur le logiciel de développement Active HDL.
- Le troisième chapitre comporte deux parties, en premier lieu, on modélisera le dispositif électromagnétique, la deuxième portera sur la réalisation de circuit de commande à programmer en langage VHDL.
- Enfin, dans le dernier chapitre on étalera sur la simulation et l'implémentation de circuit de commande.

CHAPITRE I

Circuits logiques programmables

Introduction

Dans la course aux performances, la complexité toujours croissante des systèmes a mené à des degrés d'intégrations très élevés. La logique programmable est née du besoin simultané d'intégration et d'optimisation des coûts de développement et d'industrialisation. Elle permet, en outre, de réduire les temps de mise sur le marché des produits et de gérer à moindre coût les évolutions d'un système.

Dans les années 70, on pouvait considérer d'une part, les méthodes de développement des composants VLSI qui nécessitaient la maitrise des technologies et faisant l'objet d'un métier spécifique très pointu et d'autre part la conception des cartes numériques réservé aux ingénieurs plus généralistes. Les composants utilisés étaient ceux disponibles sur le marché. Leur fonction était définie par le constructeur et l'utilisation d'un nombre important de composants de natures différentes s'avérait souvent nécessaire. La diversité et la quantité de composants utilisés entrainaient des mises au point longues et coûteuses. La moindre erreur de conception impliquait une nouvelle refonte du circuit imprimé.

Dans les années 80 apparurent les composants programmables par l'utilisateur. La densité du composant, c'est-à-dire sa capacité à remplacer un certain nombre de composants dédiés était réduite (quelque centaines de portes). On obtenait tout de même un degré supplémentaire d'intégration mais l'avantage principale résidait dans l'obtention de composants génériques, de nature homogène, apportant simultanément une souplesse de conception et permettent de gérer les évolutions.

Comme une suite aux circuits logiques programmables, au milieu des années 80 sont apparus les FPGAs. Ces composants programmables permettent un degré d'intégration très élevé (quelque millier à quelque million en 2007), ces circuits amènent toute la souplesse de la logique programmable par l'utilisateur et une densité jusqu'alors réservé aux ASICs.

Donc, dans ce chapitre on va donner un petit rappel sur la logique combinatoire et séquentielle, et ensuite un aperçu de manière générale sur les circuits logiques programmables.

I.1. Rappel sur la logique

I.1.1. Notion de système séquentiel

Les circuits combinatoires sont tels qu'à une combinaison des variables d'entrée, il correspond une et une seul combinaison des variables de sortie.

Dans les circuits séquentiels au contraire, l'état des sorties est fonction non seulement de la valeur binaire des entrées mais aussi de l'état antérieur des sorties. Or l'état antérieur des sorties dépondait du précédant état des entrées, donc la valeur des sorties dépond de l'ordre dans lequel se succèdent les états des entrées. [6]

Pour expliquer la notion de système séquentiel, considérons la commande d'un moteur M à partir de deux boutons poussoirs « a » (arrêt) et «m» (marche).

0) Pour a=0 et m=0 => M=0
1) Pour m activé, m=1 => démarrage de M, M=1
2) Pour m relâché, m=0 => continue de tourner, M=1
3) Pour a activé, a=1 => arrêt de M, M=0
4) Pour a relâché, a=0 => M reste arrêt, M=0

On constate que le fonctionnement du moteur dépond de la valeur des variables d'entrée « a » et « m » et également de l'état passé du moteur ou état interne. Traduisant les différentes étapes de fonctionnement du moteur par une table de vérité :

	A	m	M
0)	0	0	0
1)	0	1	1
2)	0	0	1
3)	1	0	0
4)	0	0	0

Tableau I-1 table de vérité de fonctionnement du moteur.

Pour la même combinaison d'entrée (ligne 2 et ligne 4) corresponde à deux comportements distincts du moteur, le problème n'est donc possible que si une troisième variables d'entrée secondaire « y » qui permettra de différencier ces deux états.

L'information sur le passé du système, nécessaire pour élaborer ses évolutions futures est appelée état interne du système séquentiel. La table représentant l'évolution du moteur

M(t+1) doit faire apparaitre les entrées « a » et « m », mais également l'état du moteur au temps t, M(t).

Cette tableest donnée par le tableau suivant :

A	M	M(t)	M(t+1)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Tableau I-2fonctionnement global d'un système séquentiel.

L'évolution de M n'étant pas spécifié pour m=a=1, nous donnons la priorité à l'arrêt du moteur. Sur la base de ces définitions, on peut considérer que le fonctionnement global d'un système séquentiel peut être décrit par le schéma général suivant :

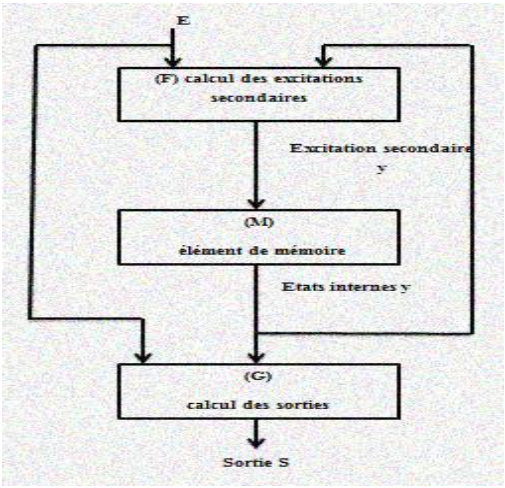


Figure I-1 : Fonctionnement global d'un système séquentiel.

Avec : E : vecteur des variables d'entrées primaires.

S : vecteur des variables de sorties.

y : vecteur des états internes ou variables d'entrées secondaires.

Y : vecteur des variables d'excitation ou excitation secondaires, elles caractérisent les états suivants.

Le système évolue à partir de chaque état interne vers un autre état interne sous l'effet des actions d'entrées.

- Le bloc F détermine la fonction état suivant à partir des variables d'entrée primaires et secondaires.
- Le bloc G réalise la fonction des sorties à partir des entrées et des états internes.

Ces deux blocs (F, G) sont des circuits combinatoires :

$$Y(t) = F[e(t), y(t)]$$
$$S(t) = G[e(t), y(t)] \quad \text{(I-1)}$$

- Le bloc M est chargé de la mémorisation de l'état interne. Il transforme l'état suivant en état présent et représente donc l'évolution interne du système. Il peut se composer
  - De bascules RS, D, JK,
  - De registres,
  - De mémoires,
  - De simples ports logiques dont le retard assure la fonction mémoire.

Pour la synthèse d'un système séquentiel, il faut introduire autant d'éléments de mémorisation que de variables internes.

L'étude des circuits séquentiels repose sur la théorie des automates finis. Un automate fini est caractérisé par :

- Son entrée E.
- Sa réponse (sortie) S.
- Son état Q.

Le comportement d'un automate est déterminé si l'on connait soit :

- Ses fonctions de transfert,
- Les tables de transitions ou états qui donnent les valeurs des fonctions de transferts
- Les diagrammes d'état ou transitions, où les états sont représentés par des ronds et les

transitions entre états par des flèches allant de l'état initial à l'état final.

I.1.2. Classification des systèmes séquentiels

Les systèmes séquentiels peuvent être différenciés en considérant :

- Leur mode de fonctionnement,
- La forme des signaux rencontrés dans le système.

I.1.2.1. Mode de fonctionnement : synchrone-asynchrone

Très souvent, l'une des variables d'entrée d'un système séquentiel est une variable logique passant successivement de l'état « 0 » à l'état « 1 » et de l'état « 1 » à l'état « 0 » d'une façon périodique dans le temps.

a) Système synchrone

Un système séquentiel est dit synchrone s'il ne peut évoluer que sur un ordre extérieur à lui-même : ses évolutions internes sont contrôlées.

En absence d'ordre, le système reste figé où il se trouvait. Dans un système synchrone, les variables d'entrées, sorties et internes sont observées à des intervalles de temps définis et commandés par la fréquence du signal d'horloge extérieur au système : le fonctionnement est asservi à l'horloge.

b) Système asynchrone

Un système séquentiel est dit asynchrone s'il peut évoluer sans ordre extérieur : des modifications en entrée provoquent immédiatement l'adaptation des états de sortie. Même s'il est doté d'une entrée d'horloge, il peut exécuter des ordres en dehors des impulsions d'horloge.

I.1.2.2. Commandes ou signaux des systèmes séquentiels

On peut trouver deux types de commandes ou de signaux :

- Un signal de niveau est un signal dont la durée est supérieure au temps de réponse du système étudié.

- Un signal impulsionnel est un signal dont la durée est inférieure au temps de réponse du système considéré.

I.1.3. Fonctions logiques élémentaires

I.1.3.1. les opérateurs logiques

Comme dans l'algèbre classique il y a des opérateurs tel que (+), (-), (/)… qui assurent la liaison entre les variables, de même il existe pour la logique binaire des opérateurs qui assurent la liaison entre les variables logiques, on les nomme : portes logiques. On donnera ici les opérateurs les plus utilisés :

a) Opérateur inversible

La porte NON est également connue par le nom INVERSEUR elle inverse le niveau de logique d'entrée à la sortie. Elle est représentée par la figure (I-2), elle n'a qu'une seule entrée et une seule sortie. La bulle dans le symbole dénote l'inversion (sans elle, le symbole représentera une porte qui ne change pas le niveau de logique ; dans la norme d'IEEE/ANSI, la bulle est remplacée par un triangle). [13]

A	F
0	1
1	0

Tableau I-3: Table de vérité de porte OR.

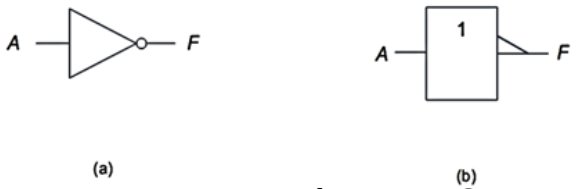


Figure I-2-a et I-2-b : Porte NON.

b) Opérateurs OR et NOR

Cet opérateur s'applique au moins à deux variables d'entrées ayant comme sortie la somme de ces entrées. A comme représentation algébrique A+B=F. Le rendement est HAUT quand une ou plus de n'importe quelle des entrées sont HAUTES et le rendement est seulement BAS quand toutes les entrées sont BASSES.

La prote NOR est fondamentalement une OR avec le rendement inversé. La figure (I-3) montre le symbole logique de cette porte. Algébriquement, l'opération peut être définie par la représentation : $\mathbf{F = \overline{A + B}}$. Plusieurs portes NOR peuvent être employées pour mettre en application les portes AND, OR ou NON.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Tableau I-4 table de vérité d'une porte OR.

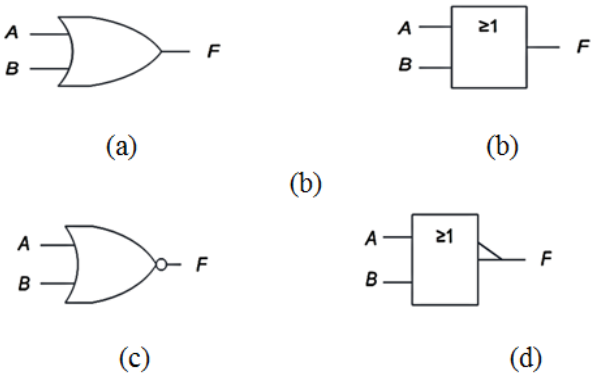


Figure I-3 : Portes logiques : (a) et (b) porte OR (c) et (d) porte NOR.

c) Opérateurs AND et NAND

Cet opérateur s'applique sur au moins deux variables d'entrées ayant comme sortie le produit de ces entrées. Sa représentation algébrique est la suivante : A*B=F. Le symbole traditionnel représenté sur le schéma de la figure (I-4-a), est généralement utilisé dans des manuels. Cependant, le symbole d'IEEE/ANSI suivant les indications de la figure (I-4-b) gagne la popularité et a l'avantage de contenir des symboles de qualification à l'intérieur du symbole logique qui décrit le fonctionnement de la porte.

Le rendement est BAS (FAUX) quand une ou plus de n'importe quelles des entrées sont BASSES (FAUX) et le rendement est seulement HAUT (VRAI) quand toutes les entrées sont HAUTES (VRAI).

Le symbole de la porte NAND est montré sur les figures (I-4-a) et (I-4-b), sa table de vérité est l'inverse de celle de la porte AND. Algébriquement, l'opération peut être définie par : $F = \overline{AB}$.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Tableau I-5 : Table de vérité de porte OR.

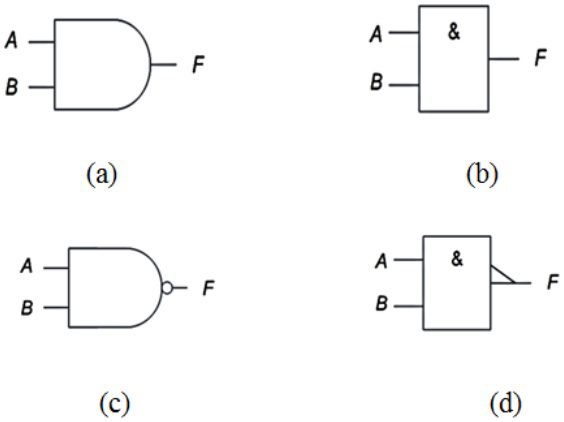


Figure I-4 (a) et (b) : Porte AND. (c) et (d) : Porte NAND.

I.1.4. Différents types de bascules

Pour les systèmes asynchrones ce sont les retards de propagation qui permettent en quelque sorte la mémorisation fugitive de l'état, par contre pour les systèmes synchrones où ce sont des dispositifs électroniques qui sont utilisés comme cellule des mémorisations, on les appelle bascules.

Donc, dans les systèmes séquentiels, le circuit de base est la bascule : c'est un opérateur susceptible de changer d'état sur commande et de conserver le nouvel état jusqu'à l'apparition d'une nouvelle commande.

Une bascule a pour rôle de mémoriser l'état d'une variable logique. Elle possède deux états stables (0 et 1) et constitue ainsi une mémoire binaire élémentaire.

On distingue les bascules à fonctionnement synchrones et les bascules à fonctionnement asynchrone :

- Dans une bascule asynchrone, les ordres appliqués sur les entrées provoquent immédiatement en sortie le changement d'état correspondant.
- Sur une bascule synchrone, une entrée supplémentaire est destinée à recevoir des impulsions d'horloge. Le circuit n'est opérationnel que pondant la durée de l'impulsion d'horloge en dehors de ce temps, la sortie reste figée quel que soit les états des entrés.

  Le déclenchement d'une bascule synchrone se fait en deux temps :

  - Application des états logiques souhaités sur les entrées,
  - Application du signal d'exécution H (horloge).

Il existe quatre principaux types de bascules : RS, JK, D et T :

I.1.4.1. La bascule RS

C'est la bascule de base, elle comporte deux entrées R et S, et deux sorties (Q et $\bar{Q}$), R reçoit le signal de la mise à zéros de la sortie Q et S reçoit le signal de la mise à un de la sortie Q. cette bascule est réalisable à partir de deux porte NOR ou de deux portes NAND.

R	S	Q^-	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	Φ
1	1	1	Φ

Tableau I-6 : Table de vérité de la bascule RS.

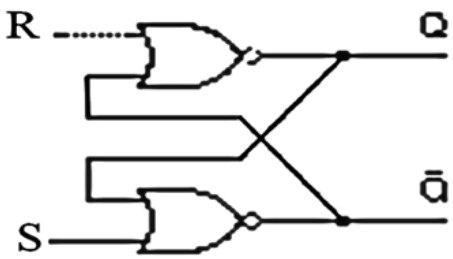


Figure I-5 : Schéma d'une bascule RS.

NB : R=S=1, le fonctionnement de la bascule n'est pas spécifié (une condition interdite) d'où Q=Φ. Si la bascule est réalisée avec des portes NAND, la condition interdite est alors le cas de R=S=0,

On peut synchroniser la bascule RS par adjonction d'un signal CLK, ce signal sert à valider les données R et S. Le basculement ou non de la bascule n'est possible que lorsque l'horloge est 1, on peut ajouter des entrées supplémentaires prioritaires permettant de forcer la bascule à 1 (preset) et 0 (clear), elle est représentée par la figure suivante [14] :

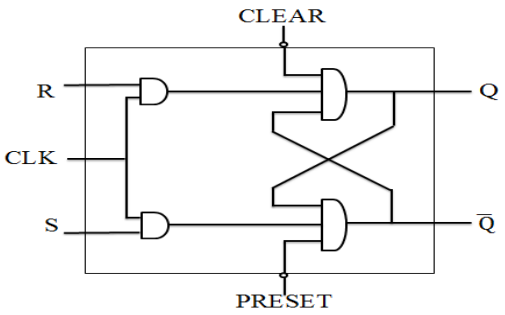


Figure I-6 : Bascule RS synchrone.

I.1.4.2. La bascule JK

La bascule JK est une bascule RS plus élaborée et ne possédant plus d'indétermination, la combinaison (J, K) = (1, 1) est maintenant applicable. La bascule JK s'obtient à partir d'une bascule RS en faisant :

$S = J\overline{Q^-}$ $\quad R = KQ^-$

Ce bouclage interne fait que la bascule JK ne peut être utilisée en mode asynchrone, la bascule JK comporte deux entrée J et K, et deux sorties Q et $\bar{Q}$.

J	K	Q^-	Q	remarque
0	*0*	*0*	0	Etat mémoire
0	*0*	*1*	1	Etat mémoire
0	1	0	0	Mise à zéros
0	1	1	0	Mise à zéros
1	0	0	1	Mise à un
1	0	1	1	Mise à un
1	1	0	1	Etat précédent inversé
1	1	1	0	Etat précédent inversé

Tableau I-7 : Table de vérité de la bascule JK.

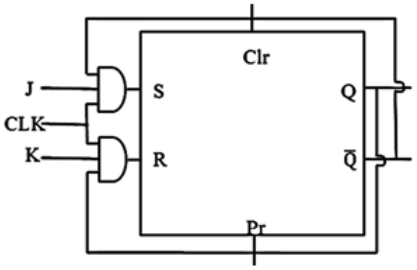


Figure I-7 : Schéma d'une bascule JK.

I.1.4.3. La bascule D

La bascule D est une bascule suiveuse : l'entrée D (donnée) est recopiée sur la sortie. Ce circuit est donc inutile en mode asynchrone. La bascule D a une seule entrée logique D(Data), une entrée horloge et deux sorties (Q et$\bar{Q}$).

En pratique, on trouve deux types de bascule :

- Bascule de stockage type D (commande par niveau).
- Bascule à commande par front.

On peut obtenir la bascule D à partir d'une bascule RS avec R=$\bar{D}$ et S=D, comme le montre le schéma donné par la figure suivante :

D	Q
0	0
1	1

Tableau I-8 : table de vérité de la bascule D.

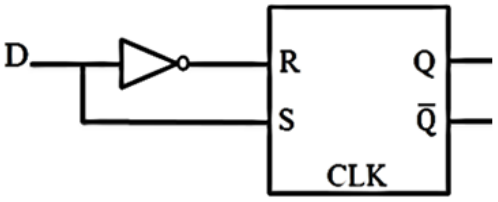


Figure I-8 : Schéma d'une bascule D.

I.1.4.4. La bascule T

Cette bascule dispose d'une seule commande d'entrée T et de deux sorties Q et $\overline{Q}$. Elle change d'état à chaque impulsion de la commande : le seul mode de fonctionnement est le basculement. Cette bascule n'est pas commercialisée, elle est fabriquée à l'aide des autres bascules.

Exemple de réalisation à partir d'une bascule D :

T	Q^-	Q
0	0	0
0	1	1
1	0	1
1	1	0

Tableau I-9 : Table de vérité de la bascule T.

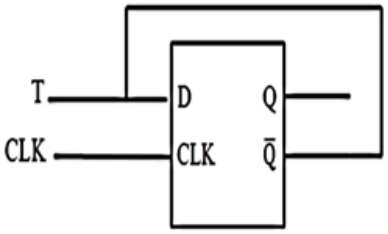


Figure I-9 : Schéma d'une bascule T.

I.2. Circuits numériques

I.2.1. Introduction

L'histoire des circuits intégrés a commencé après l'invention du transistor en 1947 par les laboratoires Bell. Relativement, l'apparition des premiers circuits intégrés est située au cours des années 1958 et 1959 grâce à un jeune ingénieur du nom de Jack Kilby de la firme Texas Instruments qui a intégré sur un même substrat de silicium plusieurs éléments électroniques (transistors, résistances, capacités) ce qui est qualifié à cette époque comme étant le premier circuit intégré. Juste après, dans les années 60, et plus précisément en 1965, un des fondateurs de la compagnie Intel nommé Gordon Moore a fait une étude concernant l'évolution du secteur des circuits intégrés, ce qui a lui permis de prédire que le taux d'intégration des transistors dans ces circuits double tous les deux ans. A nos jours, ce constat reste vérifié sur les circuits FPGA qui n'ont pas échappé à cette loi. Jusqu'au début des années 80 et même à une époque plus récente, la conception d'un système sur puce (SOC) n'était accessible qu'aux firmes et sociétés spécialisées à cause de la complexité des circuits et des fonctions à intégrer qui demandent divers efforts et compétences. Par conséquence des coûts élevés et cette technologie est inaccessible au grand public.

Aujourd'hui, l'avènement des dernières générations des FPGAs a permis de mettre la technologie SOC à la portée d'un public nettement plus large. Ceci est particulièrement depuis que les FPGAs sont proposés à un prix très faible et raisonnable. Ce prodigieux essor a été rendu possible grâce aux progrès concernant les technologies de fabrication des transistors et les méthodes de conception assistée par ordinateur (CAO). Le rôle des FPGA est d'intégrer des circuits logiques complexes. Ces circuits sont susceptibles d'être reconfigurés (Architecture programmée modifiable) partiellement ou entièrement suivant l'application. A cet effet, ce premier chapitre s'inscrit dans un contexte qui traite les FPGA et leurs positions au sein des autres systèmes digitaux.

I.2.2. Facteur d'évolution des circuits numériques

Les circuiteries numériques (où la porte logique représente l'unité de base) reposent sur les trois aspects des circuits logiques qui sont : le combinatoire, le séquentiel et l'hybride des deux. Des statistiques qui ont été faites ces dernières années ont montrés qu'au bout de 10 ans (1995 à 2004), le prix d'une porte logique a été divisé par 200 d'où l'intérêt économique du numérique. A cet effet, le prix d'une porte logique est divisé et se réduit de 40% par an. Le diagramme suivant montre ces statistiques d'évolution des coûts durant ces dernières années:

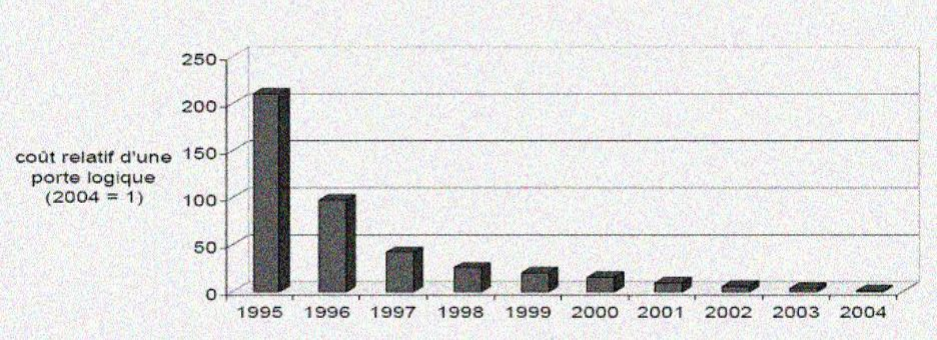


Figure I-10 : Diagramme d'évolution des coûts ces dernières années. [15]

Cette évolution est le fruit d'un ensemble de facteurs qui sont récapitulés comme suit

- Besoins croissants en circuits spécialisés.
    - Produits de plus en plus complexes.
    - Contraintes (performance, coût…).
- Evolution rapide de la technologie.
    - Généralisation des « Systèmes sur Puce ».
    - Espace de conception trop grand.
- Evolution des outils de conception "haut-niveau".
    - Produire une architecture à partir d'un algorithme.
    - Exploration automatique de l'espace de conception.
    - Outils d'estimations (performances, surface, etc.).

I.2.3. ASICs

Un ASIC (pour : Application Specific Integrated Circuit) est un circuit intégré (microélectronique) spécialisé. En général, il regroupe un grand nombre de fonctionnalités unique et/ou sur mesure.

Les ASICs fournissent précisément la fonctionnalité nécessaire pour une tâche spécifique. Ils sont conçus pour une tâche bien précise cela leur permet d'être plus petits, moins chers, plus rapide et de consommer moins de puissance qu'un processeur programmable.

I.2.3.1. Classification des ASICs

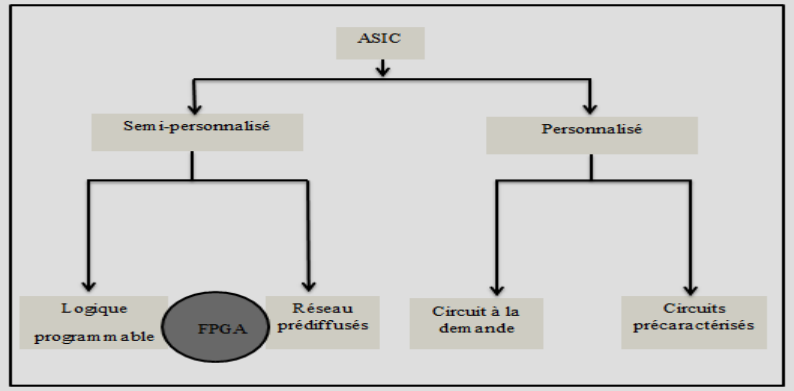


Figure I-11:Classification des ASICs.

a) Les circuits semi-personnalisés

Les semi personnalisés sont des réseaux prédéfinis de transistors ou fonctions logiques qui nécessitent une personnalisation de l'utilisateur pour réaliser la fonction désirée. Cette famille comprend :

- Les réseaux logiques programmables,
- Les réseaux pré-diffusés.

a).1. Les réseaux logiques programmables

Ce composant ne nécessite aucune étape technologique supplémentaire pour être personnalisé. Nous y trouvons les PALs/PLDs, ce sont des circuits standards programmables par l'utilisateur grâce à des différents outils de développement. La programmation consiste à

établir des connexions en imposant un courant supérieur aux courants normaux (claquage de fusibles ou de jonctions). Les circuits logiques programmables incluent un grand nombre de solutions, toute basées sur des variantes de l'architecture des portes ET/OU.

Nous y trouvons :

- PAL (Programmable Array Logic) matrice ET programmable, matrice OU figée,
- PLA (Programmable Logic Array) matrice ET ou la matrice OU programmable,
- EPLD (Erasable PLD) effaçable par rayons ultraviolet, ils peuvent être reprogrammés,
- EEPLD (Electrically Erasable PLD) programmable et effaçable électriquement. Les limites de l'architecture des PLD réside dans le nombre des bascules, le nombre des signaux entrée/sortie, la rigidité du plan logique ET, OU et des interconnexions. Précisons que ces composants très souples d'emploi sont limités à des fonctions numériques et adaptés à des productions de petites séries et ne présentent aucune garantie quant à la confidentialité.

a).2. Les pré-diffusés

Les réseaux pré-diffusés sont des circuits partiellement pré-fabriqués. L'ensemble des éléments (transistors, diodes, résistances, capacités, etc.) est déjà implanté sur les circuits suivant une certaine topologie, mais les éléments ne sont pas connectés entre eux (sauf au niveau diffusion). La réalisation des connexions dans le but de définir la fonction souhaitée est la tâche du concepteur, pour cela il dispose de bibliothèques de macro-cellules et d'outils logiciels d'aide à la conception. A partir de cette liste d'interconnexions (netlist) le fondeur n'aura que quelques étapes technologiques à effectuer pour achever le circuit, c'est-à-dire le dépôt d'une ou plusieurs couches de métallisation.

Cette technique est intéressante sur le plan de la conception et de la fabrication, par contre elle présente l'inconvénient de ne pas permettre une optimisation en terme de densité de composants puisque les éléments de base sont pré implantés et pas forcément utilisés et que leur positionnement a priori n'est pas forcement optimal pour le but recherché.

b) Les circuits personnalisés

Ce sont des circuits non préfabriqués. Pour chaque application on optimise le circuit intégré, ce qui conduit à la création de son propre composant. Cette famille comprend :

- Les circuits à la demande,
- Les circuits pré-caractérisés.

b).1. Les circuits à la demande

Les solutions circuits à la demande (qu'on appelle full custom en anglais) présentent l'avantage d'autoriser une meilleure optimisation de placement puisque celui-ci n'est pas prédéfini. On dispose d'une bibliothèque de modèle mathématique de comportement et via un « compilateur de silicium » logiciel très sophistiqué on peut concevoir toute l'architecture en faisant une validation logiciel (simulation logique) puis dans une avant dernière étape, on déduit le schéma des divers masque de fabrication.

Toutes les opérations de la conception à la fabrication, sont effectuées de façons spécifiques adaptées aux exigences de l'utilisateur. L'ensemble des critères techniques est au choix de concepteur, que soit la taille du composant, le nombre de broches, le placement du moindre transistor. C'est l'ASIC le plus optimisé car aucune contrainte ni lui est imposée. Le placement des blocs fonctionnels et les routages des interconnexions, même si ces opérations sont assistées par ordinateurs, sont effectués avec beaucoup plus d'interventions manuelles pour atteindre l'optimisation au niveau de chaque transistor. Cependant, les phases de mise au point sont longues et onéreuse, il va de soi que la rentabilisation des investissements de développement nécessite un fort volume de production.

b).2. Les circuits pré-caractérisés

Pour la réalisation de circuits pré-caractérisés on dispose d'une bibliothèque de circuits élémentaires que le fondeur sait fabriquer et dont il peut garantir les caractéristiques et on les associe pour la réalisation de circuit à la demande. Ici encore on utilisera en fabrication des masques personnalisés pour chacune des couches diffusées et de métallisations.

Le concept est très semblable de celui des circuits à la demande, la seule différence réside dans la réalisation du schéma puisque l'on accède à une bibliothèque de cellules prédéfinies générant de très nombreuses fonctions élémentaires ou élaborées. Cette dernière

constitue un véritable catalogue dans lequel le concepteur se sert pour constituer son schéma. Il existe trois types de cellules,

- Les cellules standards (standard cells) correspondent à la logique classique.
- Les méga-cellules (mega-cells) peuvent être des blocs de types microprocesseur, périphériques,
- Les cellules compilées (compilables cells) dont les blocs RAM ou ROM. Il est nécessaire de personnaliser la diffusion, et par conséquent de créer tous les masques.

Cependant un avantage évident en découle : alors qu'il est impossible avec les pré-diffusés d'utiliser à 100% le réseau des cellules ou portes, ce qui se traduit par une perte de silicium, les pré-caractérisés permettent d'exploiter complètement la surface du circuit.

I.2.4. Les technologies de mémorisation

Voici d'une manière générale, l'arborescence des mémoires disponibles à nos jours : [15]

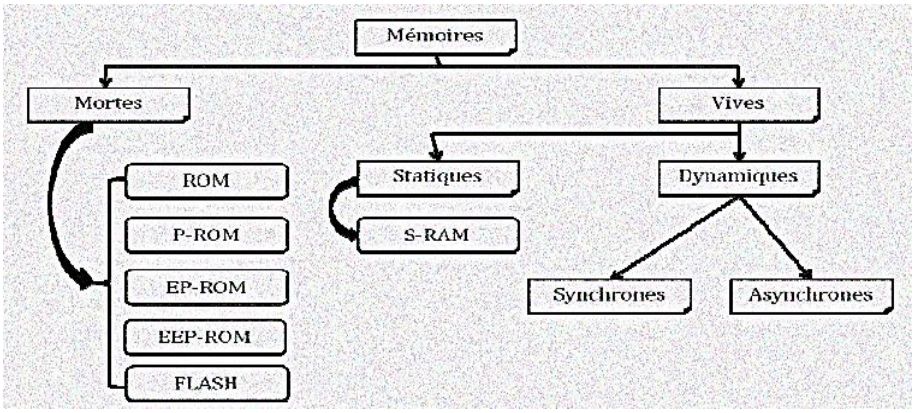


Figure I-12 : Les différents types de mémoires.

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

- Les ROM (Read Only Memory): Mémoires figées par le concepteur à lecture seule et non modifiables.
- Les P-ROM (Programmable Read Only Memory): Mémoires programmables une fois par l'utilisateur avec un équipement spécialisé (tableau de fusibles).

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

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

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

- Les S-RAM (Static Random Mémory):Mémoires volatiles avec cellule de base à plusieurs transistors (accès rapide, consommation plus, coûteux). La volatilité correspond à l'indisponibilité de l'information lorsqu'il n'y a pas d'alimentation.

- Les RAM dynamiques: Mémoires volatiles qui nécessitent rafraichissement périodique de l'information afin de la conserver avec cellule de base à un transistor (densité forte, accès lent).

I.2.5. Circuits logiques programmables

Il y a de cela quelques années, la réalisation des montages électroniques impliquait l'utilisation d'un nombre important de circuits intégrés logiques, ce qui avait pour conséquence une mise en œuvre complexe, un prix de revient important et un circuit imprimé de taille.

Pour diminuer et minimiser ces circonstances, ces dernières années ont vu l'apparition d'un nouveau type de circuit intégré à savoir, les circuits logiques programmables (PLD ; programmable logic device). Ces derniers ont pour origine le développement des mémoires utilisées en informatique. Ce type de produit est caractérisé par sa simplicité de mise en œuvre et sa capacité d'intégrer, dans un seul circuit, plusieurs fonctions logiques programmables.

La plupart des circuits numériques programmables sont issues de la célèbre architecture « Architecture Von-Neumann » proposée par John Von Neumann en 1945 et porte son nom. Ensuite une autre architecture qui vienne pour compenser les lacunes de la précédente dans certains domaines et afin d'améliorer la cadence de calcule où le facteur temps d'exécution est le plus important. Cette architecture nommée « Architecture Harvard » qui porte le nom de

l'université américaine qui l'a proposée sachant que parallèlement l'architecture « Von-Neumann » n'est pas figée mais en évolution. Mais le besoin croissant de composants très rapides a orienté les chercheurs à développer une autre solution qui sera complètement différente des deux précédentes architectures. Cette solution réside dans le mode de programmation qui est devenu architectural à logique câblée inversement aux deux premières architectures précédentes où la programmation est séquentielle.

D'une manière générale, il existe deux alternatives ou solutions qui sont:

- Une solution logicielle : Elle est nommée aussi solutions programmables du type « Processeur » où un traitement séquentiel relativement lent et programmation dépendante des composants (DSP, Microprocesseur et Microcontrôleur…).
- Une solution matérielle: Elle est nommée aussi solutions programmables du type «Logique » où un traitement parallèle en temps réel et une programmation architecturale avec un langage de description matériel HDL (Méthodologie de conception CAO) indépendante du composant (ASIC et FPGA…).

Les caractéristiques de ces circuits sont :

- Les circuits du type DSP/Microprocesseurs : Un rapport performance/coût faible, un temps de conception très court et une grande souplesse d'utilisation.
- Les circuits du type spécialisé ASIC : Très performants mais avec un cycle de conception long et une architecture figée.
- Les circuits du type FPGA: Des performances proches des ASIC, un coût unitaire intermédiaire et un cycle de conception moyen et une architecture modifiable.

Voici un tableau récapitulatif de comparaison des différentes solutions numériques :

	ASIC	FPGA	DSP
Performances	Très élevées	Elevées	Faibles
Taille	Faible	Moyenne	Elevée
Consommation	Faible	Moyenne	Très élevées
Intégration	Système sur puce	Système sur puce	Composants supplémentaires
Souplesse	Fonctions figées	Reconfigurable	Programmable
Mise en œuvre	Complexe	Complexité moyenne	Complexité moyenne
Coût du composant	Très élevé	Moyen	Faible

Tableau I-10 : Comparaison de différentes solutions numériques.

Les fréquences de fonctionnement du mode séquentiel dépassent aujourd'hui 2 GHz, la réduction du temps de cycle ne suffira pas à compenser les insuffisances de ce mode de fonctionnement. La fréquence d'horloge ou de fonctionnement des FPGA est relativement faible devant les microprocesseurs et les DSP et ne dépasse pas quelques centaines de Mégahertz, mais cette faiblesse est largement compensée et même surpassée grâce au parallélisme de traitement. L'exploitation du parallélisme est une technique en pleine expansion dans les circuits numériques FPGA qui sont une alternative des DSP.

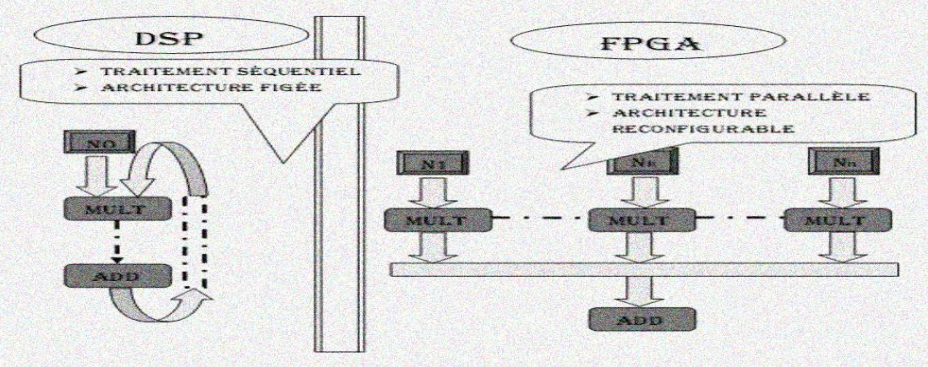


Figure I-13 :Schéma comparatif d'un DSP et d'un FPGA.

I.2.5.1. Définition des PLDs

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

I.2.5.2. Structure des PLDs

La plupart des circuits PLDs suivent la structure suivante (figure I-14)

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

Le deuxième et le troisième ensemble forment chacun ce qu'on appelle une matrice. Les interconnexions de ces matrices doivent êtres programmables, et ceci est réalisé par des fusibles qui sont grillés lors de la programmation. 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 : [4]

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

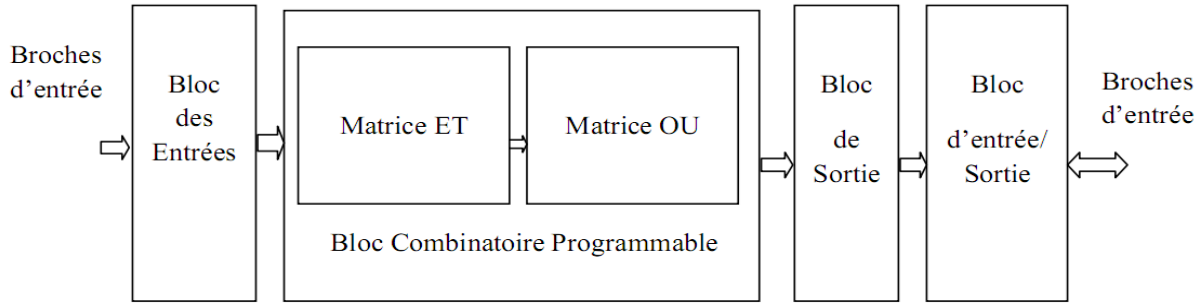


Figure I-14 : Structure de base des PLDs.

I.2.5.3. Les différentes familles de PLDs

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

Type	Nombre de porte intégré	Matrice ET	Matrice OU	Effaçable
PAL	10 à 100	Programmable	Fixe	Non
GAL	10 à 100	Programmable	Fixe	Électriquement
EPLD	100 à 3000	Programmable	Fixe	Par UV
FPLA	2000 à 3000	Programmable	Programmable	Électriquement
FPGA	Plus de 50 000	Programmable	Programmable	

Tableau I-11 : Différentes familles des PLDs.

a) Les PALs (Programmable Array Logique)

Les PALs ont eu un grand succès dès leur première parution sur le marché, ce fut les premiers circuits logiques programmables.

Un PAL est un composant relativement simple, dérivé des PROM (Programmable Read Only Mémoire, mémoire morte à lecture seule programmable une fois). Les ingénieurs de MMI ont combiné la technologie à fusibles (utilisée pour les mémoires PROM) avec des portes ET, OU pour réaliser des fonctions logiques. La compréhension de la cellule de base d'un PAL suffit car c'est la même qui se répète sur l'étendue de la capacité de celui-ci (voir la Figure(I.14)). La cellule de base se compose d'un buffer d'entrée qui dispose de l'information et de son complément (tous les PALs sans exception disposent d'un certain nombre d'entrées qui aboutissent toutes, sous forme directe et inversé, sur la matrice de fusible de programmation), suivit de la matrice à fusibles puis de portes ET (considérées en entrée) puis suivit de portes OU en sortie [16].

La programmation d'un PAL s'effectue par la destruction de fusibles, à l'aide d'un programmateur dédié en appliquant des tensions de programmation requises. L'inconvénient majeur des PALs, c'est qu'une fois programmée, ils ne sont plus effaçables (fusible détruit) c'est contraignant en cas d'erreur de programmation ou de mise à jour. Les PALs existent sous 4 d'architectures (dans l'ordre de leur évolution) :

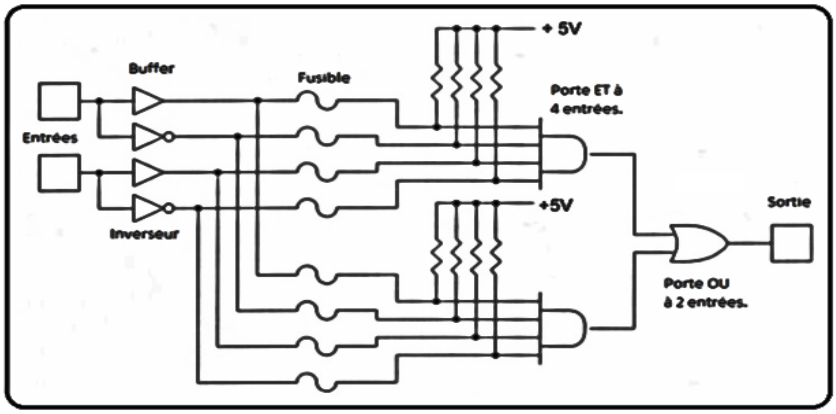


Figure I-15 : Cellules de base d'un PLD (architecture combinatoire).

- Les PALs combinatoires : possède l'architecture la plus simple. Comme dans tout type de PALs, certaines broches sont dédiées uniquement aux entrées. D'autres bidirectionnelles, sont associés à un buffer de sortie trois états.

- Les PALs à registres : ils disposent en sortie d'une bascule D. Tout signal de sortie passe obligatoirement par cette bascule. Pour cette raison, certains boîtiers sont proposés avec un certain nombre de sorties à registre (synchrone) et d'autres de type combinatoire.

- Les PALs asynchrones à registres : Ce type de PALs constitue une variante du type évoqué précédemment. Une première différence réside dans la méthode de distribution du signal d'horloge des bascules. Contrairement aux PALs à registres, un multiplexeur permet de shunter le registre. Les deux entrées AP (PRESET Asynchrone) et AR (RESET Asynchrone) sont simultanément utilisées pour piloter ce multiplexeur. A noter la présence d'une XOR se comportant comme un inverseur programmable. Le signal de commande de cette XOR a une valeur statique, il ne s'agit ni d'un signal global ni d'un signal issu de la matrice. C'est à partir de là que le concept de la « macro cellule » programmable a fait son apparition. Cette ressource d'inversion permet de coder une fonction sans tenir compte de la polarité du signal de sortie.

- Les PALs versatiles (VPAL) : ils constituent une évolution des PALs très significative. Ils proposent des macros cellules très évoluées, du type de cellules rencontrées au sein de composants plus complexes (CPLD, FPGA). Le mode de fonctionnement est obtenu par l'utilisation de deux multiplexeurs qui utilisent comme

signal de sélection les points de programmation S0 et S1. En combinaison de ces deux points de programmation, on obtient différentes configurations de la sortie.

b) Les GALs (Generique Array Logic)

Les GALs ne sont rien d'autre (d'un point de vue architectural) que des PALs reprogrammables. D'un point de vue technologique au lieu d'utiliser des transistors bipolaires, ils ont utilisé des transistors MOS FET pouvant être régénérés. Cette possibilité de régénération des fusibles sans pour autant restreindre la durée de vie du composant.

Après étude des besoins du marché LATTICE SEMICONDUCTOR se fixe quatre objectifs pour la mise au point des GALs :

- Offrir des produits ayant des vitesses de travail comparable à celle des PALs bipolaires, tout en étant testable à 100%.
- Permettre un remplacement, au moins fonctionnel, mais idéalement broche pour broche, des PALs bipolaires dans n'importe quelle application.
- Offrir une consommation beaucoup plus faible que les PALs bipolaires d'une complexité équivalente.
- Proposer une plus grande souplesse de configuration des entrées/ sorties que les PALs bipolaires.

C'est ainsi que LATTICE à pallier aux inconvénients majeurs des PALs pour donner naissance à un composant lui imposant une forte concurrence.

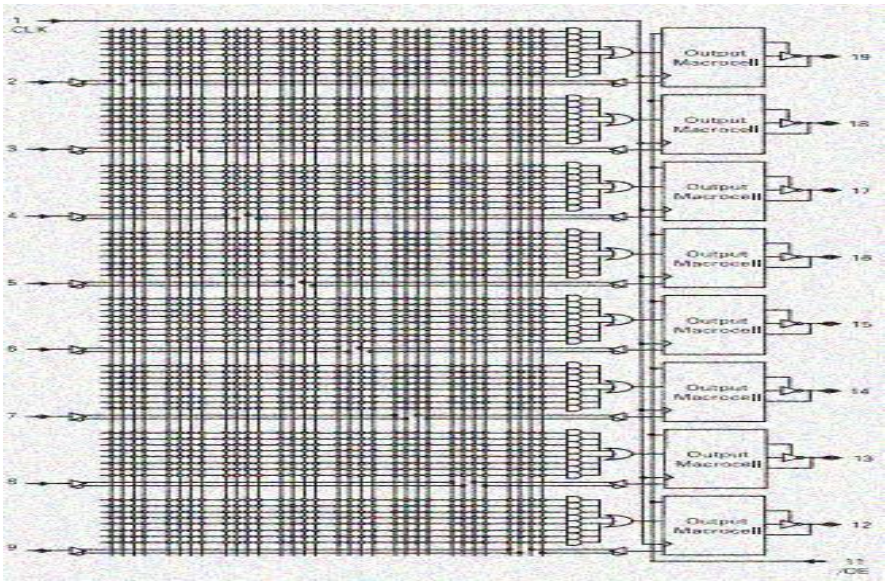


Figure I-16: GAL 16V8 chip. [11]

c) Les EPLDs (Erasable Programmable Logique Device):

Les EPLDs sont des circuits réalisés en technologie CMOS, présentant l'avantage de faible consommation électrique, mais qui augmente en fonction de la fréquence. Ils disposent d'une macro cellule plus évolué que celles des PALs, en plus ils sont effaçables.

L'introduction des EPLDs telle que l'a voulu ALTERA visait deux buts distincts:

- Permettre une densité d'intégration nettement supérieure à celle offerte par les PALs et aussi proche que possible que celle permise par les réseaux de portes programmable.
- Fonctionner à une vitesse, si non égal, du moins comparable à celle des PAL bipolaires et en tout cas nettement supérieure à celle des portes traditionnels.

Les différentes familles d'EPLDs :

- Les EPLDs classique de la série EP. Ces circuits existent en différentes versions avec des boîtiers disposant de 20 à 68 pins. Effaçable à l'UV.
- Les EPLDs de la série MAX 5000, qui ont la même fonction que leur homologue de la série EP, avec toutefois une densité d'intégration plus élevée et une architecture d'interconnexion différente, effaçable à l'UV.
- La série MAX 7000 plus dense que MAX 5000 en plus effaçable électriquement.
- La série MAX 9000 plus dense, effaçable électriquement en plus programmable sur circuit avec la tension unique de 5v.

d) Les CPLDs (Complex programmable logic device)

Ce sont des circuits composés de plusieurs PALs élémentaires reliés entre eux par une zone d'interconnexion (matrice d'interconnexion). Sa physionomie est généralement très structurée. Un certain nombre de macro-cellules de base sont regroupées pour former des blocs logiques. Grâce à leurs structures, ils peuvent atteindre des vitesses de fonctionnement élevées (plusieurs centaines de MHz). Ces circuits ont une capacité en nombre de portes et en possibilités de configuration très supérieure à celle des PALs. [4]

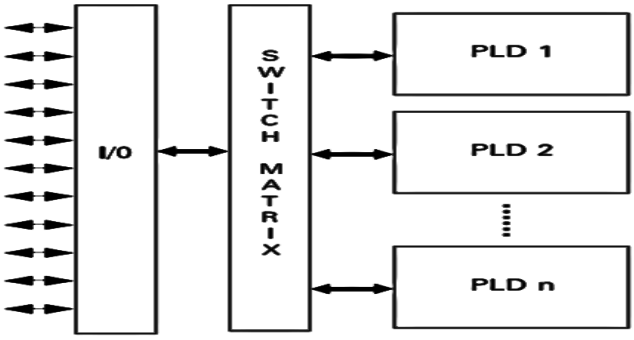


Figure I-17 : Structure des CPLDs.

e) Les FPGAs

Les FPGA sont des composants VLSI (Very Large Scale Integration). Ils sont programmables par l'utilisateur et essentiellement constitués de trois parties : [17]

- Une matrice de blocs logiques configurables CLB (Configurable Logic Bloc).
- Des blocs d'entrée/sortie configurables.
- Un réseau d'interconnections programmables.

La figure (I-18) présente l'architecture générique d'un FPGA.

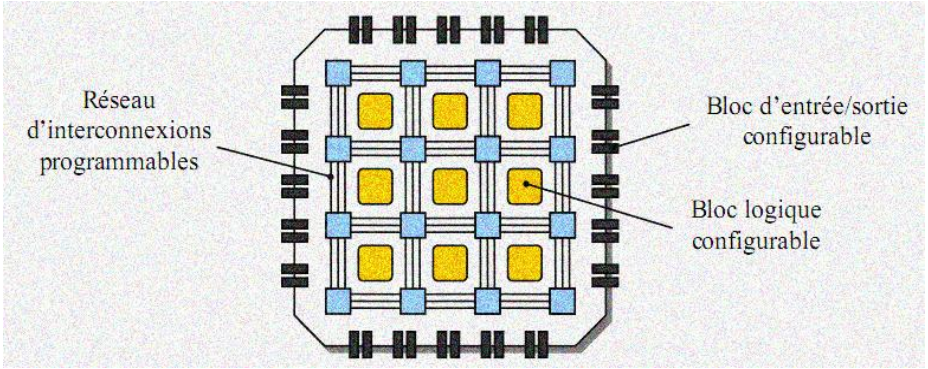


Figure I-18 : Architecture générale des FPGAs.

Les FPGAs constituent une évolution des circuits logiques programmables CPLDs. Ils se composent d'une matrice de blocs logiques programmables entourée de blocs d'entrées/sorties, et l'ensemble est relié par un réseau d'interconnexion programmable. Ces circuits comparés aux CPLDs utilisent des modules logiques beaucoup plus réduits mais beaucoup plus nombreux. 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. Selon l'information véhiculée, on distingue plusieurs types de lignes définies par leur longueur relative. On trouve en effet :

- Les interconnexions à usage général qui sont composées de segments verticaux et horizontaux qui entourent chaque 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-sortie.

- 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 des points de connexion est appelé PIP (abréviation anglaise de Programmable Interconnect Points) et chaque point de connexion peut être réalisé selon deux techniques qui définissent deux classes de FPGAs (FPGAs à SRAM et à anti-fusible).

L'évolution des circuits logiques programmable, depuis la création des PALs qui présentaient l'avantage de réduire l'encombrement et de créer des fonctions logiques personnalisé. Puis vient l'étape des EPLDs qui présentent l'avantage de l'écriture électrique mais en ayant recours à un programmateur (un appareil qui permet d'injecter le routage du circuit via un fichier de programmation) mais effaçable à l'UV (ultraviolet) pour évoluer vers les CPLDs qui sont effaçable électriquement. Puis l'avènement des FPGAs qui représentent une technologie qui permet de reprogrammer le circuit in situ (c'est-à-dire sur circuit ou sur site). L'avantage majeur que présentent les FPGAs est leur grande flexibilité. En effet, la structure interne du circuit FPGA peut être changée sans avoir à modifier la structure globale de la maquette. Cet avantage est très apprécié par les concepteurs de cartes électroniques vu que ça leurs permet de faire des prototypage rapide et de moindre coût en comparaison aux ASICs pour lesquels il faut des mois pour réaliser un prototype sans avoir de certitude qu'il puisse être opérationnel en plus de ça la moindre erreur nécessite de refaire le travail depuis le début. Un coût de reviens important et une durée de développement étendu ce qui doit être minimisé dans les milieux industriels. Il y aussi l'avantage de la mise à jour du circuit face à des bugs ou l'ajout de nouvelles fonctionnalités. [16]

Xilinx, Altera et Quick logic sont les pionniers dans le domaine des FPGAs, et plusieurs autres compagnies produisent les FPGAs. Toutes ces compagnies se partagent le même concept architectural. Il se divise en trois parties : Interfaces d'entrées/Sorties (I/O interface), les blocs logiques de base (Basic Logic Building Blocks) et les interconnexions (voir figure (I.18)).

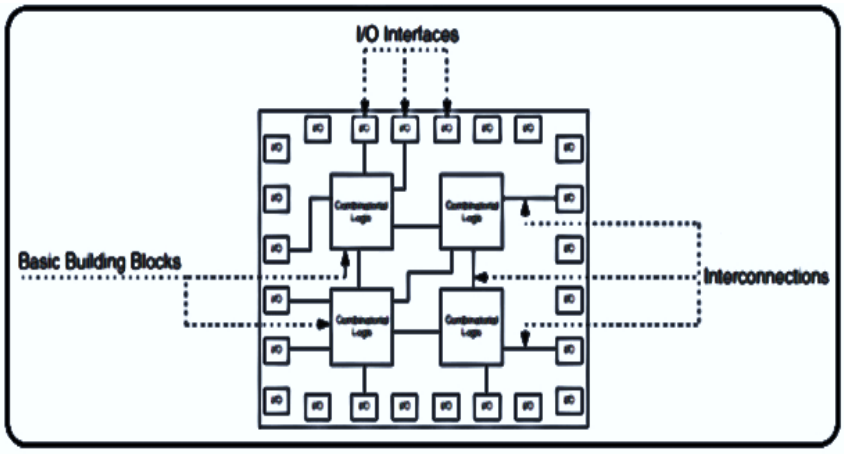


Figure I-19 : Concept architecturel de base des FPGAs.

e).1. Interfaces d'entrées/Sorties (I/O interfaces)

Les interfaces d'entrées/Sorties se présentent comme les intermédiaires par lesquelles les données transitent depuis les blocs logiques internes jusqu'aux ressources externes et vice versa. L'interfaçage des signaux peut être : unidirectionnel, bidirectionnel, à deux états ou trois états (0, 1 ou haute impédance) voir même obéir à un standard d'entrées/sorties. Voici quelques-uns de ces standards :

- GTL (gunning transceiver logic).
- HSTL (high-speed transceiver logic).
- LVCMOS (low-voltage CMOS).
- LVTTL (low-voltage transistor-transistor logic).
- PCI (peripheral component interconnect)…

Le rôle principal des interfaces d'entrées/sorties est de transmettre et de recevoir des données. Néanmoins l'interface d'entré/sorties peut être dotée d'options telles que des registres, impédances et buffers.

Chez Xilinx, les interfaces d'entrées/sorties sont nommées IOB pour Input Output Blocks. L'IOB est constitué de registres, de diviseurs de tensions, des résistances de rappel pull up et autres ressources spécifiques. La Figure (I-19) est un exemple d'IOB simplifié de Xilinx. [16]

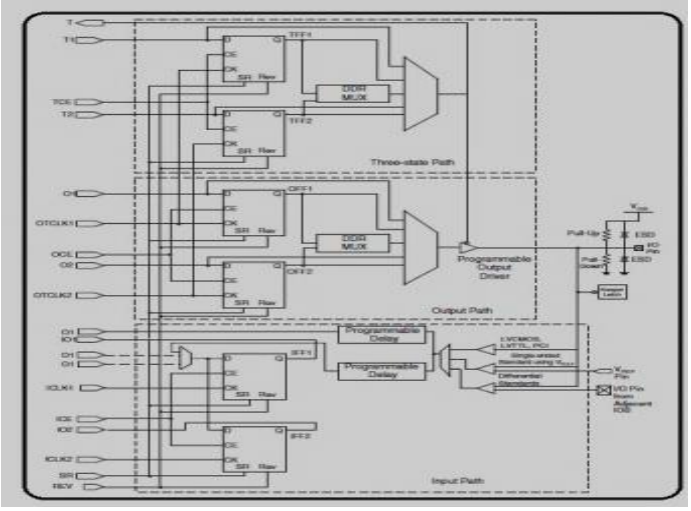


Figure I-20 : Structure simplifiée d'un IOB (bloc entrées/sorties de FPGA Xilinx). [16]

e).2. Les blocs logiques de base (basic logic building blocks)

Les blocs logiques de base, sont des blocs logiques configurés ou optimisés de sorte à réaliser des fonctions logiques. Xilinx, nomme les blocs logiques de base CLB Configurable Logic Blocks. Chaque CLB contient des slices et chaque slice a des LUTs Look up Tables. Voir la figure (I-20).

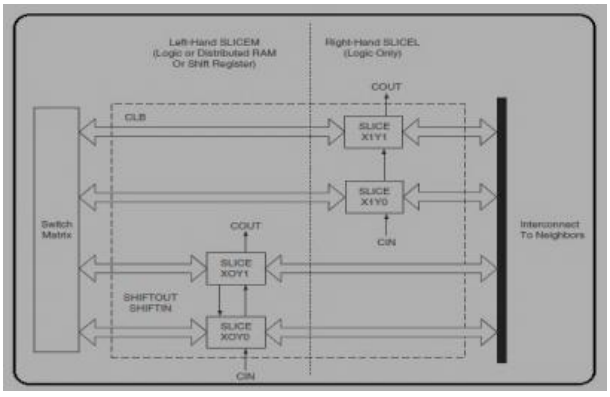


Figure I-21 : Constitution d'un CLB et la disposition des slices (famille Virtex).

e).3. Les Interconnexions

Les interconnexions sont très importantes car ce sont elles qui transmettent le signal d'un point à un autre. Il existe plusieurs types de lignes :

- Les interconnexions à usage général : sont composées de segments verticaux et horizontaux qui entourent chaque bloc logique de base et qui peuvent être reliés entre eux par une matrice de communication.
- Les lignes directes : fournissant des chemins entre les blocs logiques de base adjacents et entre les blocs logiques de base et les interfaces d'entrées/sorties.
- Les lignes longues : qui sont des lignes verticales et horizontales qui n'utilisent pas de matrices de communication. 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.

I.2.6. Les différentes technologies de réalisation des FPGAs

Il y a plusieurs constructeurs de composants FPGA tels que Actel, Xilinx et Altera. Ces constructeurs utilisent différentes technologies pour la réalisation des FPGAs. Parmi ces technologies, celles qui assurent une reprogrammation des FPGAs sont les plus intéressantes étant donné qu'elles permettent une grande flexibilité de conception. Les différentes technologies reprogrammables des FPGA sont les suivantes : [17]

- La technologie EPROM : Cette technologie utilise des transistors EPROM (Erasable Programmable Read-Only Memory). Son principal désavantage est l'opération de reconfiguration qui nécessite l'utilisation d'une source ultraviolette.

- La technologie EEPROM : Cette technologie utilise des transistors EEPROM (Electrically Erasable Programmable Read-Only Memory). Par rapport à la technologie EPROM, elle présente l'avantage de pouvoir être reprogrammée électriquement.

- La technologie Static Ram (SRAM) : Pour cette technologie, les connexions sont réalisées en rendant les transistors passants. Cette technologie permet une reconfiguration rapide du circuit FPGA. Cependant, son principal inconvénient est la surface nécessaire pour la SRAM.

- La technologie FLASH : Cette technologie est limitée en nombre de reconfigurations et possède un temps de reconfiguration plus long par rapport à la technologie SRAM. Cependant, l'avantage de cette technologie est qu'elle garde sa configuration même si

l'alimentation est enlevée. Par conséquent, un FPGA à base de technologie Flash déjà programmé est prêt à fonctionner dès sa mise sous tension.

I.2.7. Critères de choix des circuits programmables FPGAs

Les FPGAs sont développés récemment grâce aux progrès de la technologie VLSI, l'apparition de ce type de circuits est une révolution des systèmes digitaux et ouvrants des perspectives de traitement numérique inaccessibles auparavant. Typiquement, un circuit FPGA haute densité peut contenir jusqu'à plusieurs millions d'éléments programmables. Pour réussir une application à base d' FPGA et afin d'obtenir un système plus performant, consommant un minimum de puissance, il est nécessaire de respecter un certain nombre de règles comme :

- Bien connaître les caractéristiques du FPGA ciblé pour assurer son adéquation avec les besoins du projet.
- Elaborer une méthodologie de conception.
- Maîtriser les outils d'implémentation et de choisir des outils de synthèse de qualité.

La conception sur les circuits FPGA est un challenge dans lequel l'objectif est de trouver le bon compromis entre densité, flexibilité et performances temporelles.

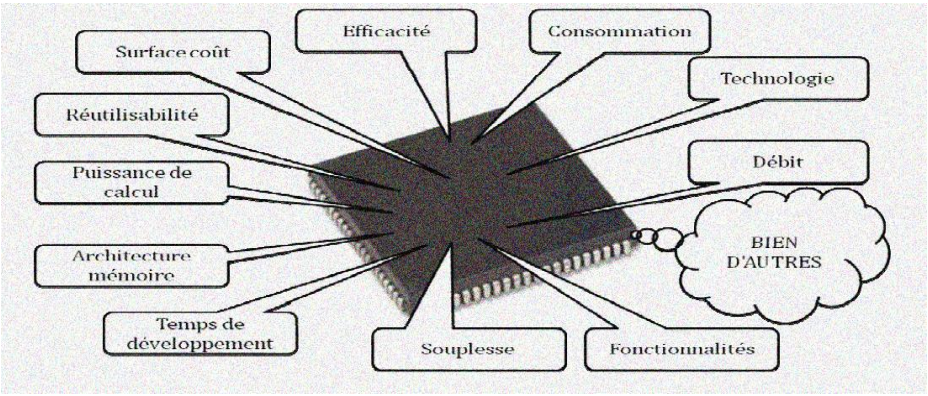


Figure I-22 : Critère de choix des circuits FPGAs.

I.2.8. Différent domaines d'application des FPGA

Les FPGA ont fait révolutionner certain domaines de contrôle numérique et de plus en plus utilisés pour intégrer des architectures numériques complexes. Ils sont devenus les plus populaires en matière d'implantation et de prototypage des circuits numériques après leur

apparition sur le marché en 1984. La clé maîtresse de leurs réussites est l'aspect de programmation de ces derniers. Leurs utilisations actuelles couvrent les deux domaines : civil et militaire. Parmi ces applications nous citons :

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

I.2.9. Avantages d'utilisation des FPGA

Les circuits programmables fournissent une puissance de calcul importante. Des algorithmes, traditionnellement considérés comme une séquence d'instructions, peuvent être analysés pour déterminer un parallélisme possible. La possibilité de reconfiguration sur ces circuits permet une minimisation et une adaptation spécifique à chaque algorithme. [20]

Dans un circuit programmable, on peut implanter des algorithmes au niveau matériel avec un parallélisme au niveau calcul, ce qui conduit à des performances supérieurs par rapport à des processeurs DSP.

De nombreux algorithmes nécessitent la multiplication. L'architecture du composant programmable contient des Look-Up Tables (LUTs) qui simplifient la multiplication. Ces tables contiennent des valeurs pré-calculées, éliminant ainsi l'ajout d'une cellule multiplieur.

Les principaux avantages des circuits FPGA sont donc :

- Un faible coût pour une production à faible unité, ou pour le développement de prototypes car ces circuits sont reprogrammables indéfiniment.
- Un temps de fabrication du circuit négligeable par rapport à un circuit ASIC développé chez un fondeur.

Par contre, un circuit FPGA devient d'un coût plus important qu'un circuit ASIC pour une production de plus grande série.

Les FPGA peuvent servir dans trois types d'applications :

- Le prototypage rapide : Un FPGA permet de transférer une nouvelle application sur silicium en quelques heures alors que la réalisation des circuits intégrés (ASICs) nécessite des semaines voire des mois. Ceci présente l'avantage de pouvoir évaluer plusieurs options d'architecture de réalisation pour une application donnée.

- L'émulation du matériel : Ceci est particulièrement efficace pour des applications pour lesquelles la simulation logique s'avère inutilisable à cause d'évènements survenant en temps réel. Un FPGA peut servir comme circuit de prototypage bon marché dans la mesure où le temps de conception est une tache critique.

- L'accélération de fonctions : Un FPGA peut fournir une aide précieuse pour le développement de différentes fonctions sur un composant reconfigurable. Par exemple, afin de satisfaire au mieux les besoins de communication entre plusieurs processeurs parallèles, les composants sont configurés selon l'interface à utiliser.

Conclusion

Ce premier chapitre a porté en premier lieu sur des généralités sur la logique combinatoire et la logique séquentielle, puis en second lieu sur les circuits numériques particulièrement sur les circuits logiques programmables, en donnant un petit historique de ces circuits et leurs évolutions sur le marché. Les FPGAs sur lesquels on a donné splus d'importance appartiennent à cette catégorie, en indiquant leurs avantages et leurs spécifications. Ce qui nous a permis de conclure que la technologie FPGA s'inscrit au sommet de l'évolution des composants logiques et le besoin croissant de composants plus performants, plus économiques et disponibles en grandes quantités sont les grands axes du progrès qui sont disponibles dans les FPGA.

CHAPITRE II

Langage de description VHDL

Introduction

Au cours des vingt dernières années, les méthodes de conception des fonctions numériques ont subi une évolution importante. Dans les années soixante-dix, la majorité des applications de la logique câblée étaient construites autour des circuits intégrés standards, souvent pris dans la famille TTL. Au début des années quatre-vingt apparurent, parallèlement, les premiers circuits programmables par l'utilisateur du côté des circuits simples et les circuits intégrés spécifiques (ASICs) pour les fonctions complexes fabriquées en grande série. La complexité de ces derniers a nécessité la création d'outils logiciels de haut niveau qui sont à la description structurelle (schémas au niveau des portes élémentaires). A l'heure actuelle, l'écart de complexité entre circuits programmables et ASICs s'est restreint : on trouve une gamme continue de circuits qui sont des héritiers des premiers PALs (programmable array logic), équivalents de quelques centaines de portes, à des FPGAs (Field programmable gate array) ou des LCAs (Logic cell array) de quelques dizaines de milliers de portes équivalentes. Les outils d'aide à la conception se sont unifiés ; un même langage, VHDL par exemple, peut être employé quels que soient les circuits utilisés, des PALs aux ASICs[3].

II.1. Définition de VHDL

Le VHDL est un langage de description matériel destiné à représenter le comportement ainsi que l'architecture d'un système électronique. Son apparition remonte aux années 80 suite à l'initiative du département de la défense des Etats-Unis guidé par le souci de développer un langage normalisé de design et de documentation. C'est ainsi que ce département parraina un projet de développement du VHDL (Very High Speed Integrated Circuit Hadware Description Language) qui sera standardisé en 1987 par l'Institut of Electrical and Electronics Engineers (IEEE) sous la norme IEEE 1076.[4]C'est un langage de haut niveau : il fait abstraction de l'objet pour lequel il est utilisé. Il est utilisé (VHDL) en particulier pour développer les circuits logiques programmables. Un outil de développement permet la programmation d'un circuit logique programmable à partir d'un fichier VHDL. Paradoxalement le VHDL qui est un langage relativement ancien créé pour la modélisation, semble s'imposer aujourd'hui comme standard de description (destiné à la synthèse). Tous les constructeurs de composants programmables possèdent aujourd'hui des outils intégrant les ressources VHDL. Ce langage est largement employé par les concepteurs. Pour ces différentes raisons, le VHDL est considéré comme un

des maillons indispensables dans la chaine de développement des composants programmables[8].

II.2. Un langage commun

Le principe majeur de VHDL est de fournir un langage commun, depuis la description au niveau système jusqu'au circuit, à tous les niveaux de description le langage est le même. Créer un langage qui permette à la fois de valider une conception au niveau système, de construire les programmes de test utiles à tous les niveaux et génère automatiquement des schémas d'implémentation sur le silicium, est évidemment un propos ambitieux. VHDL est donc beaucoup plus qu'une simple traduction textuelle de la structure interne d'un circuit.

VHDL est un langage informatique qui décrit des circuits. Les programmes de test simulent l'environnement d'un montage, ils bénéficient de la souplesse de l'informatique pour recréer virtuellement toutes les situations expérimentales possibles et imaginables, même sicelles-ci seraient difficiles à réaliser en pratique.

II.3. Structure d'une description VHDL

La description d'un système en VHDL est en fait toujours structurelle au niveau le plus élevé. Le principe de la hiérarchisation est exactement le même que pour une entrée schématique. On réalise des composants que l'on interconnecte entre eux. Ces composants peuvent être eux même composés de composants. Dans une entrée schématique, on distinguera parfois un sous-schéma et un composant. Dans le principe, il s'agit toujours d'un sous ensemble rappelable plusieurs fois dans le niveau supérieure de la hiérarchie. Comme dans une entrée schématique, où il est nécessaire de réaliser d'une part le schéma de composants et d'autre part le symbole physique contenant les broches de connexions et les attributs du composant, en VHDL, un composant sera défini par une zone appelée ENTITY correspondant au symbole et une zone appelée ARCHITECTURE correspondant au schéma. Toutes description VHDL sera composée d'au moins un composant et aura la structure syntaxique minimale suivante [8] :

Structure d'une description VHDL

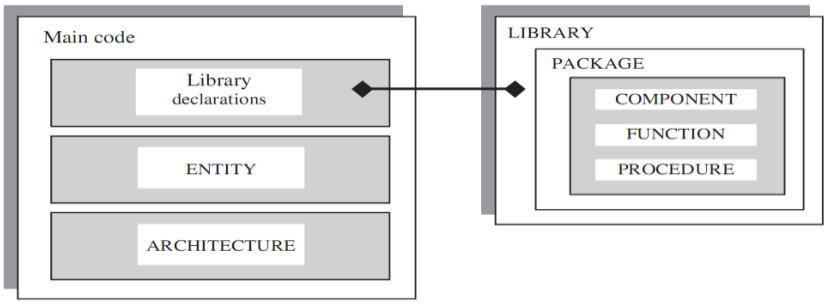


Figure II-1 : Les unités fondamentales d'une description VHDL.

```
ENTITY nom_du_composant IS
PORT(signal,[singnal],…… :direction type ;
      signal,[singnal],…… :direction type ;
           …
           signal,[singnal],…… :direction type ;
END nom_du_composant ;
ARCHITECTURE nom_architecture_du_composant OF nom_du_composant IS
      [déclaration des composants]
      [déclaration des types]
      [déclaration des signaux]
      …
BEGIN
            « zone de description du composant »
END nom_architecture_du_composant ;
```

Un opérateur élémentaire, un circuit intégré, une carte électronique ou un système complet est complètement défini par des signaux d'entrées et de sorties et par la fonction réalisée de façon interne. Ce double aspect (signaux de communication et fonction) se retrouve à tous les niveaux de la hiérarchie d'une application. L'élément essentiel de toute description en VHDL, nommé design entity dans le langage, est formé par le couple entity-architecture, qui décrit l'apparence externe d'une unité de conception et son fonctionnement interne[5].

La description VHDL d'un circuit combinatoire est composée de deux parties :

- l'entité (entity) qui permet de définir les entrées et les sorties du composant, leurs noms et leurs types ;
- l'architecture (architecture) qui permet de définir le comportement du circuit.

II.3.1. ENTITY (boite noire :une entité)

La déclaration d'entité décrit l'interface entre le monde extérieur et une unité de conception : signaux d'entrées et de sorties et, éventuellement, paramètres génériques (constantes dont la valeur est fixée par l'environnement de l'unité considérée). La déclaration d'entité peut également contenir des instructions concurrentes passives, c'est-à-dire qui ne contiennent aucune affectation de signaux ; de telles instructions ne génèrent pas de circuit lors du processus de synthèse.

Une même entité peut être associée à plusieurs architectures différentes. Elle décrit alors une classe d'unités de conception qui présentent au monde extérieur le même aspect, avec des fonctionnements internes différents. Le même circuit peut, par exemple, être décrit de façon purement fonctionnelle lors de la conception, et de façon structurelle pour contrôler le bon respect des règles temporelles de ses constituants physiques.

Syntaxes

```
entity_declaration : :=
entity identifier  is
[ generic ( generic_list ) ; 1 [
port (port_list ) ; ]
entity_declarative_part
[begin
entity_statement_part ]
end [ entity ] [ identifier ] ;
```

La zone déclarative (entity_declarative_part) et la zone d'instructions (entity_statement_part) contiennent des informations qui sont connues de toutes les architectures qui partagent la même entité. Nous n'en détaillerons pas ici les contenus, certains exemples illustreront leur utilisation éventuelle. Notons cependant qu'en ce qui concerne les instructions, seules des instructions « passives », c'est-à-dire qui ne provoquent aucun changement de signal, sont légales dans l'entité. Il est, par exemple, possible de vérifier à cet endroit que des spécifications électriques concernant les signaux d'entrées. Rappelons que, dans ces descriptions syntaxiques, les mots réservés du langage sont indiqués en caractères gras et que les éléments entre crochets ne sont pas obligatoires.

Exemple : Une déclaration d'entité pour un multiplexeur de quatre voies vers une voie :

```
entity mux4_1 is
port( e0,e1,e2,e3 : in bit ;
sel : in bit_vector(1 downto 0) :
```

```
sort : out bit ) ;
end mux4_1 ;
```

Une déclaration d'entité pour un multiplexeur de dimension arbitraire 1 :

```
entity muxN_1 is
generic ( dimension : integer := 4) ;
port( entree : in bit_vector(dimension-1 downto 0) :  sel : in
integer range 0 to dimension-1 ; sort : out bit )
nd muxN_1 ;
```

Une déclaration d'entité pour un registre bidirectionnel de dimension arbitraire:

```
entityregistre_N is
generic ( dimension : integer := 8 ) ;
port(hor, direction : in bit :
donnee :inoutstd_logic_vector(0 to dimension -1));
endregistre_N ;
```

Donc dans cette partie on donne la description des entrées et sorties de la structure, leurs noms, leurs types et leurs modes.

L'entité précise

- le nom du circuit,
- les ports d'entrée-sortie :
  - leur nom,
  - leur direction (in, out, inout),
  - leur type (bit, bit_vector, integer, std_logic, …)
- les paramètres éventuels pour les modèles génériques

II.3.2. ARCHITECTURE (Son contenu : une architecture)

Le fonctionnement interne d'un module, son corps, est précisé par une architecture associée à l'entité qui décrit l'aspect extérieur de ce module. Une architecture porte un nom, ce qui autorise la création de plusieurs architectures différentes pour la même déclaration d'entité. Une unité de conception est la réunion d'une entité et d'une architecture.

L'architecture est divisée en deux parties : une zone déclarative et une zone d'instructions. Ces instructions sont concurrentes, elles s'exécutent en parallèle. Cela signifie que les instructions d'une architecture peuvent être écrites dans un ordre quelconque, le fonctionnement ne dépend pas de cet ordre d'écriture. Les différentes parties d'un circuit coexistent et agissent simultanément ; il peut être un peu surprenant pour les programmeurs habitués des langages procéduraux comme C ou PASCAL. Le corps de l'architecture est

inclus entre lesdeux mots clés « BEGIN » et « END ». La syntaxe générale d'une architecture estreprésentée comme suit :[8]

```
architecturearchitecture_name of entity_nameis
architecture_declarative_part
begin
architecture_statement_part
end [ architecture ] [ architecture_name ]
```

La zone déclarative permet de définir des types, des objets (signaux, constantes) locaux au bloc considéré. Elle permet également de déclarer des noms d'objets externes (composants d'une librairie, par exemple) utilisés dans le corps de l'architecture.

Exemples : l'exemple suivant correspond à l'exemple de déclaration d'entité donné précédemment.

```
--Un multiplexeur de quatre voies vers une voie :
architectureessai of mux4_1 is
constant zero : bit_vector(1 downto 0) := "00" ;
constant un : bit_vector(1 downto 0) := "01" ;
constant deux : bit_vector(1 downto 0) := "10" ;
constanttrois : bit_vector(1 downto 0) := "11" ;
begin
process (e0,e1,e2,e3,sel)
begin
casesel is
when zero  => sort <= e0 ;
when un  => sort <= el :
when deux  => sort <= e2 ;
whentrois => sort <= e3 ; end
case ;
end process ;
endessai ;
```

II.4. Déclaration des bibliothèques

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques. L'IEEE (Institut of Electrical and Electronics Engineers) les a normalisées et plus particulièrement la bibliothèque **IEEE1164**. Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques,...[9]

Les bibliothèques permettent à plusieurs Concepteurs de travailler ensemble sur le même projet et rendent le langage indépendant du Système d'Exploitation de la machine

hôte.La création de la bibliothèque ne fait pas partie du langage VHDL. Chaque outil VHDL a donc ses propres règles de création. Par exemple avec le simulateur ModelSim il faut créer la bibliothèque avec cette commande : [10]

- vlib BIB

La bibliothèque par défaut est work. Work est aussi le nom symbolique de la bibliothèque dans laquelle sont rangés les résultats. La bibliothèque STD est une bibliothèque standard fournie avec le langage. Elle contient des définitions des types et des fonctions de base (integer, bit, boolean …) dans le paquetage STANDARD et des fonctions sur les caractères dans le paquetage TEXTIO.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
```

Cette dernière bibliothèque est souvent utilisée pour l'écriture de compteurs. La directive USE permet de sélectionner les bibliothèques à utiliser. Les bibliothèques STD et le WORK montré ci-dessus sont rendus évident par défaut, tellement là n'est aucun besoin de les déclarer ; seulement la bibliothèque d'IEEE doit être explicitement écrite.Cependant, la dernière (WORK) est seulement nécessaire quand le type de données de STD_LOGIC (ou STD_ULOGIC) est utilisé dans la conception.

std_logic_1164 :Les systèmes à valeurs multiples indique par STD_LOGIC (8 niveaux) et de STD_ULOGIC (9 niveaux) logique. [11]

std_logic_arith de _ : Indique les types de données SIGNÉS et NON SIGNÉS et les opérations relatives d'arithmétique et de comparaison. Il contient également plusieurs fonctions de conversion de données, qui permettent à un type d'être converti en des autres :conv_integer(p), conv_unsigned(p, b), conv_signed(p, b), conv_std_logic_vector(p, b) [11].

std_logic_signed :Contient les fonctions qui permettent à des opérations avec des données de STD_LOGIC_VECTOR d'être effectuées comme si les données étaient du type SIGNÉ[11].

std_logic_unsigned : Contient les fonctions qui permettent à des opérations avec des données de STD_LOGIC_VECTOR d'être effectuées comme si les données étaient du type NON SIGNÉ.

Les bibliothèques permettant de ranger les résultats et de les réutiliser, certaines sont par défaut WORK ET STD, d'autres sont standards IEEE, d'autres sont personnelles [10].

II.5. Description comportementale

Ce type de description spécifie le comportement du composant ou du circuit à réaliser au moyen d'instruction séquentielle ou sous forme de flow de données (DATAFLOW) constituant un process. Dans ce type de descriptions, il y a plusieurs niveaux.Nous allons voir les principales descriptions :

- Sous forme de flow de données :

Dans ce type de description comportementale, on modélise le circuit par un ensemble d'équations logiques et arithmétiques; c'est-à-dire on décrit chaque sortie par une équation en fonction des entrées. [4]

- Sous forme d'instruction séquentielle:

Sous cette forme, le contenu est décrit de façon algorithmique en utilisant les structures des langages de programmation, Pour les circuits séquentiels, il faudra faire appel à un niveau plus abstrait, appelé RTL, à savoir : les déclarations séquentielles suivantes [4] :

- La structure alternative :

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

- La structure d'aiguillage :

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

Séq_stat désigne une ou plusieurs déclarations séquentielles.

II.6. Description structurelle

La description en VHDL d'un circuit se fait souvent de façon structurelle :

- pour faciliter la compréhension : il est plus simple de séparer le circuit en blocs plus simples, ayant des fonctions bien identifiées. Ces blocs pourront alors être décrits sous forme comportementale, ou bien à leur tour les séparer en blocs encore plus simples.
- pour faciliter la synthèse : la synthèse est un processus lent (en termes de temps et de calcul). Plus un bloc est gros et complexe, plus sa synthèse prendra du temps. Il vaut mieux travailler sur des blocs plus petits, plus simples à synthétiser et rassembler le tout à la fin [10].

Dans ce type de description, on énonce les interconnexions des composants préalablement décrits. Cette description est la transcription directe d'un schéma. Elle se compose de trois rubriques, à savoir : [4]

- Déclaration des signaux internes destinés à interconnecter les composants : Cette déclaration se fait par le mot clé <SIGNAL> accompagné par le nom et le type du signal utilisé
- Déclaration des composants utilisés : cette étape permet de lister les composants utiles pour la construction de l'entité. Elle se fait de la manière suivante :

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

- Représentation de la réalisation des différentes interconnexions des composants déclarés dans la partie déclarative : Chaque connexion est définie de la manière suivante :

Comp 1 : composant port map (….) ;

Où :

Comp1 : représente l'affectation du mapping.

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

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

II.7. Processus

« Un processus est une instruction concurrente (N.D.T deux instructions concurrentes sont simultanées) qui définit un comportement qui doit avoir lieu quand ce processus devient actif. Le comportement est spécifié par une suite d'instructions séquentielles exécutées dans le processus. » [3]

Que cela signifie-t-il ?

Trois choses :

1. Les différentes parties d'une réalisation interagissent simultanément, peu importel'ordre dans lequel un câbleur soude ses composants, le résultat sera le même. Le langage doit donc comporter une contrainte de « parallélisme » entre ses instructions. Cela implique des différences notables avec un langage procédural comme le C.

Exemple :

```
a <= b ;
c <= a + d ;
```

et

```
c <= a + d ;
a <= b ;
```

Ces deux manières d'écrire représentent la même chose en VHDL, ce qui est notablement différent de ce qui se passerait en C.

2. L'algorithmique fait grand usage d'instructions séquentielles pour décrire le mode. VHDL offre cette facilité à l'intérieur d'un processus explicitement déclaré. Dans le corps d'un processus il sera possible d'utiliser des variables, des boucles, des conditions, dont le sens est le même que dans les langages séquentiels. Même les affectations entre signaux sont des instructions séquentielles quand elles apparaissent à l'intérieur d'un processus. Seul sera visible de l'extérieur le résultat final obtenu à la fin du processus.

3. Les opérateurs séquentiels, surtout synchrones, mais pas exclusivement eux, comportent « naturellement » la notion de mémoire, qui est le fondement de l'algorithmique traditionnelle. Les processus sont la représentation privilégiée de ces opérateurs. Mais attention, la réciproque n'est pas vraie, il est parfaitement possible de décrire un

opérateur purement combinatoire par un processus, le programmeur utilise alors de cet objet la seule facilité d'écriture de l'algorithme.

Syntaxe générale

```
[étiquette : ]process [ (liste de sensibilité) ]
partie déclarative optionnelle : variables notamment
begin
corps du processus.
instructions séquentielles
end process[ étiquette ] ;
```

II.9. Paquetage

En VHDL, il existe une unité de conception dont le rôle est de permettre le regroupement de données, de variables, de fonctions et de procédés. Cette unité est une sorte de bibliothèque d'objets (constantes, variables, fonctions ou procédures……etc.) que l'on pourra appeler à partir de plusieurs descriptions. Cette unité est particulièrement intéressante lorsqu'on utilise au sein d'un projet contenant plusieurs entités de ressources identiques [4].Un paquetage permet de rassembler des déclarations et des sous programmes, utilisés fréquemment dans une application, dans un module qui peut être compilé à part, et rendu visible par l'application au moyen de la clause use [3].

Un paquetage est constitué de deux parties : la déclaration, et le corps (body).

- La déclaration contient les informations publiques dont une application a besoin pour utiliser correctement les objets décrits par le paquetage: essentiellement des déclarations, des définitions de types, des définitions de constantes et de sous-programmes destinés à être utilisés par d'autres unités de conception. ...etc.
- Le corps, qui n'existe pas obligatoirement, contient le code des fonctions ou procédures définies par le paquetage, s'il en existe.

Le paquetage PAQ est déclaré avec le mot clé package. Des nouveaux types "type" et sous types "subtype" qu'on peut déclarer dans le paquetage. On définit un type MOT qui est un vecteur de 8 bits, un type RAM qui est un tableau de MOTS et un sous type MRAM qui est une RAM de 1024 MOTS. Des constantes "constant " N_BITS et REL. N_BITS est un entier > 0 "positive" et REL est un entier naturel "NATURAL". [10]

II.10. Les attributs

Les attributs sont des propriétés ou des caractéristiques associées à un objet ou à un type. Ils se divisent en deux grandes catégories ; les attributs prédéfinis et les attributs définis par l'utilisateur. La désignation d'un attribut fait intervenir un préfixe et un suffixe séparés par une apostrophe.

Attributs sur les types

L'exemple suivant illustre les principaux attributs de type : [10]

```
type COULEUR is (BLEU, ROUGE, VERT);
COULEUR 'LEFT renvoie BLEU
COLEUR'RIGHT renvoie VERT
COULEUR'POS (BLEU) renvoie 0
COULEUR'VAL(0) renvoie BLEU
COULEUR'SUCC(BLEU) renvoie ROUGE
COULEUR 'PRED(ROUGE) renvoie BLEU
```

La syntaxe de déclaration de ces attributs est comme suit :

Nom_var' nom_attribut ;

II.11. Description Mixte

Elle regroupe les deux types de descriptions décrites précédemment. A chaque entité peut être associée une ou plusieurs architectures, mais, au moment de l'exécution (simulation, synthèse…..), seulement une architecture et une seule est utilisée. Cette dernière est spécifiée par le mécanisme de package.

L'architecture dépend implicitement de l'entité à laquelle elle est associée ; tous les objets définis dans l'entité sont connus par l'architecture et ils sont vus comme des signaux qui peuvent être lus ou écrits à cet endroit, cela se fait par le biais d'instructions concurrentes énumérées au niveau du corps de l'architecture. Cette architecture comporte aussi une partie déclarative où peuvent figurer un certain nombre de déclarations (de signaux, de composants… Etc.) internes à l'architecture [4].

II.12. Les Objets utilisés en VHDL

Lors de la programmation en VHDL, on utilise différents objets qu'on peut subdiviser en trois familles à savoir : les constantes, les variables et les signaux [4].

II.12.1. Les constantes

Ce sont des objets bien connus des langages de programmation. Leur valeur, une fois définie pendant l'initialisation, reste inchangée par l'exécution. Néanmoins, lorsqu'elles sont déclarées dans un sous-programme, les constantes sont recalculées à chaque appel du sous-programme mais restent constantes pendant l'exécution du sous-programme. Elles sont déclarées par le mot clés « constant ».

II.12.2. Les variables

La valeur des variables est immédiatement mise à jour lorsqu'elles sont assignées. Ces variables peuvent être déclarées soit à l'intérieur d'un sous-programme soit à l'intérieur d'un processus (process) ; elles sont locales à leur bloc de déclaration. Les variables ne peuvent apparaître que dans le domaine de programmation séquentielle. La syntaxe de déclaration et d'affectation des variables est comme suit :

```
            Déclaration : VARIABLE <nom_var> : TYPE ( := expression
initiale) ;
            Affectation : nom_var   valeur ;
```

II.12.3. Les signaux

Les signaux sont les éléments de base de toute description. Ils sont déclarés dans la zone de spécification des constituants du composant. Ils nomment les connexions entre les entités du composant.

Les signaux ne sont pas des contenants comme les variables. Ils sont des objets permanents et ils ont des liens fixés avec d'autres signaux. Les signaux possèdent une histoire, en effet, ils ont un passé accessible par l'utilisation d'attributs, un présent c'est-à-dire une valeur courante et un futur consistant à un ensemble de couples (valeurs, temps) projetés et gérés par le noyau du simulateur.

La syntaxe de déclaration et d'affectation du signal est comme suit:

Déclaration : SIGNAL<nom_var> : TYPE (:= expression initiale) ;

Affectation : nom_var   valeur ;

Le signal dispose d'une caractéristique principale qui réside dans la sémantique de son assignation, en effet, il peut être affecté par le résultat d'une expression pouvant utiliser les valeurs courantes d'autres signaux (comme par exemple « A <= B and C » A, B et C étant

des signaux). Mais cette assignation n'implique pas que la valeur courante du signal soit mise à jour immédiatement. En fait, la valeur de l'expression est stockée dans le pilote du signal en tant que valeur projetée de ce signal. Elle ne deviendra seulement la valeur courante du signal que lorsque sera rencontrée une instruction de synchronisation (instruction « wait »). De plus, ce mécanisme d'assignation peut utiliser des retards comme le montre l'exemple suivant :

« A <= B and C after 5ms ».

La différence entre variables et signaux est que les premières n'ont pas d'équivalent physique dans le schéma, contrairement aux seconds. Certains outils de synthèse ne respectent malheureusement pas cette distinction. On notera qu'il est possible d'affecter la valeur d'une variable à un signal, et inversement, pourvu que les types soient compatibles.

II.13. Instruction concurrentes et séquentielles

II.13.1. Les instructions concurrentes

Les instructions concurrentes sont décrites directement dans le corps de l'architecture. L'objectif de ces relations étant d'affecter des valeurs à des composants ou de réaliser des connexions, alors elles n'ont pas d'ordre d'exécution. Elles sont effectuées en parallèle.

On distingue trois types d'instructions concurrentes :

- Instruction logique classique : Elle traduit une simple interconnexion entre deux équipotentielles et cela en utilisant l'assignation inconditionnelle.

La syntaxe de déclaration est :

Nom_du_signal<= expression.

- Instruction conditionnelle du type When……else : Cette instruction permet de spécifier une expression en fonction d'une ou plusieurs conditions. Et cette dernière doit être une expression logique de choix.

La syntaxe de déclaration de cette instruction est :

```
Nom_signal<= source_1 when condition_1 else
source_2 when condition_2 else
         .
.
source_n;
```

- Instruction sélective de type With-select-when: Cette instruction spécifie une valeur à évaluer et précise les différents cas donnant lieu à une affectation pour un signal. Elle est semblable à l'instruction précédente avec en plus une précision du signal sur lequel vont porter les signaux.

La syntaxe de déclaration de cette instruction est :

```
                With expression select
    Nom_signal<=valeur_1 when select_1;
          valeur_2 when select_2;
.
 .
                              valeur_sinon when others;
```

II.13.2.Les instructions séquentielles

Ce type d'instructions est interne aux processus, aux procédures et aux fonctions (sous-programmes). Elles constituent les outils de base des descriptions comportementales. Ces instructions sont, pour la plupart, inspirées des langages classiques de programmation.

- Instruction séquentielle du type if-then-else: Cette instruction permet la réalisation de boucle conditionnelle. En effet, elle permet d'exécuter des blocs d'instruction en fonction des résultats d'évaluation des conditions booléennes.

La syntaxe générale de déclaration de cette instruction est :

```
IF condition_1 THEN
Instructions séquentielles ;
ELSEIF condition_2 THEN
Instructions séquentielles ;
.
ELSE
Instructions séquentielles ;
 ENDIF ;
```

- Instruction case-then : Cette instruction permet de spécifier une des instructions à effectuer en fonction de la valeur de l'expression testée. L'expression doit être de type discret. Toutes les valeurs possibles de l'expression testée doivent être traitées et apparaître une fois et une seule dans la suite des alternatives.

La syntaxe de déclaration de cette instruction est :

```
        CASE Expression IS
        WHEN choix_1   Instructions séquentielles ;
  WHEN choix_2   Instructions séquentielles ;
        .
  WHEN OTHERS   Instructions séquentielles ;
  END CASE;
```

- Instruction wait : L'instruction « WAIT » provoque la suspension du process ou de la procédure dans lesquels elle apparaît. Avec l'utilisation d'une telle instruction, l'exécution des traitements est bloquée tant que l'expression spécifiée n'est pas évaluée et qu'aucun événement ne survient sur l'un des signaux de la liste de sensibilité.

  La syntaxe de déclaration est :

```
WAIT[ON liste de signaux][UNTIL Expression booléenne][FOR Expression temps]
```

- Instruction return : Cette instruction est réservée aux sous-programmes. A son exécution, le sous-programme est immédiatement suspendu, la main étant rendue à l'appelant.

La syntaxe générale pour une fonction est :

```
    Return Valeur ;
    Et pour une procédure elle est :
  Return ;
```

- Instruction nulle :

La syntaxe de cette instruction est :

Null ;

La sémantique de cette instruction est claire, l'exécution passe à la ligne suivante. Cette instruction est très précieuse pour écrire certaines instructions de sélection (case) quand tous les cas doivent être envisagés.

- Boucles : Elles permettent de répéter des séquences d'instruction soit de façon indexée à un indice (boucle for), soit de façon continue jusqu'à l'occurrence d'une condition de sortie (exit). Chaque déroulement s'appelle une itération.

  - Boucle FOR-LOOP :

Elle répète la séquence d'instruction un certain nombre de fois. La syntaxe de cette boucle est :

```
     [Etiquette :] : FOR indice IN min TO max LOOP
  Liste d'instruction
     END LOOP [Etiquette] ;
```

  - Boucle WHILE-LOOP : La syntaxe générale de cette boucle est :

```
     [Etiquette :] : WHILE condition LOOP
     Liste d'instruction
    END LOOP [Etiquette] ;
```

  - Boucle simple : La syntaxe générale est :

```
     [Etiquette :]   LOOP
     Liste d'instruction
     END LOOP [Etiquette] ;
```

Dans cette boucle, il est nécessaire d'utiliser l'instruction de sortie de boucle exit pour ne pas rester indéfiniment dans la boucle.

La syntaxe de cette instruction est :

```
          Exit [Etiquette] [WHEN Expression]
```

Dans le même registre, l'instruction « NEXT » permet de passer à l'itération suivante dans une boucle de type FOR. En effet, elle permet d'arrêter l'itération en cours et de se positionner de manière à exécuter l'itération suivante.

La syntaxe générale de cette instruction est :

```
NEXT [Etiquette] [WHEN Expression] ;
```

II.14. Différence entre le VHDL et un langage de programmation

La différence essentielle avec un langage informatique est la simultanéité des actions décrites. Pour cela il faudra se familiariser avec la notion d'instruction concurrente (simultanée) et d'instruction séquentielle. Par défaut toutes les instructions sont concurrentes, pour qu'elles soient séquentielles il faut qu'elles apparaissent dans un process. Un langage de programmation est réalisé pour décrire une exécution d'un programme, alors qu'un langage comme le VHDL a pour objectif de décrire l'état de la structure matérielle d'un système.

Un système est structuré en composant fonctionnant en permanence, par contre, un langage de programmation est structuré en sous-programmes qui ont une durée de vie (exécution) limitée dans le temps avec un début et une fin. La même différence existe en ce qui concerne le support de l'information : dans un langage de programmation les variables supportent l'information et elles sont affectées de manière ponctuelle dans le temps, alors que pour un système matériel les composants sont interconnectés avec des signaux qui sont des connections permanentes et concurrentes, des sorties de canaux par lesquels transitent les informations. La notion du signal est la base de la description matérielle des structures de données qui sont le support de l'information en transite dans le système. Un signal est donc affecté en permanence [12].

II.15. Les avantages du langage VHDL

Un langage de description hardware comme le VHDL permet de décrire et de synthétiser rapidement des circuits contenants l'équivalent de 5 à 20 milles portes logiques (voir plus), un travail équivalent avec les équations booléennes prendrait quelques mois de travail [12].

- Puissance et flexibilité :

Le VHDL est un langage de haut niveau, moderne, puissant permettant l'écriture du code décrivant des circuits logiques complexes. Il offre en plus la possibilité de créer des bibliothèques de composants réutilisables d'un projet à l'autre ; l'utilité est grande car le code

de chaque composant peut être validé par simulation, la recherche des erreurs dans un composant complexe est donc simplifiée.

- Design indépendant du circuit :

Le VHDL permet de créer une fonction sans se préoccuper du composant dans laquelle elle sera implémentée. Il ne faut pas donc devenir intimement familier avec l'architecture d'un circuit pour l'utilisation des ressources et les performances.

- Portabilité

Portabilité des descriptions VHDL, c'est-à-dire, possibilité de cibler une description VHDL dans le composant ou la structure que l'on souhaite en utilisant l'outil que l'on veut (en supposant que la description en question puisse s'intégrer dans le composant choisi et que l'outil utilisé possède une entrée VHDL).

Conclusion

Dans ce chapitre une présentation plus au moins détaillée du langage VHDL est fournie, et de manière générale on a vu :

- Les bibliothèques
  - permettant de ranger les résultats et de les réutiliser,
  - certaines sont par défaut WORK ET STD, d'autres sont standards IEEE, d'autres sont personnelles.

- L'entité
  - Elle décrit l'interface du circuit avec le monde extérieur,
  - Il faut spécifier les ports, leur sens et leur type,
  - Les paramètres d'un circuit générique sont déclarés dans l'entité.

- L'architecture
  - Décrit l'intérieur du circuit, en structurel (schéma) ou en comportemental ou les 2,
  - Avant de les utiliser dans le corps de l'architecture, il faut déclarer les objets et les types (signaux, composants, constantes, fonctions, …).

CHAPITRE III

Modélisation du système et réalisation de circuit de commande

Introduction

Dans ce travail on réalisera un système de commande d'un lanceur électromagnétique. D'abord c'est quoi un lanceur électromagnétique, il s'agit d'un dispositif destiné à projeter des objets, sorte de canons électriques.

Il existe deux catégories des lanceurs [7] :

- Canon à rails
- Lanceur à induction

Dans notre cas nous intéressons au lanceur à induction, ce dernier est composé de deux parties :

- Un inducteur (partie fixe) constitué d'un enroulement triphasé alimenté par un système de tensions variables.
- Un induit (partie mobile), représente le projectile [7]

La partie fixe alimentée sous une tension alternative crée un champ d'induction magnétique B, alors que la partie mobile est assimilable à une spire mise en court-circuit, elle est le siège des courants induits qui ont pour effet d'éloigner la pièce de la partie fixe (inducteur).

III.1. Modélisation du système

Dans le but d'étudier notre système, on l'assimile à un circuit RLC donné par la figure suivante:

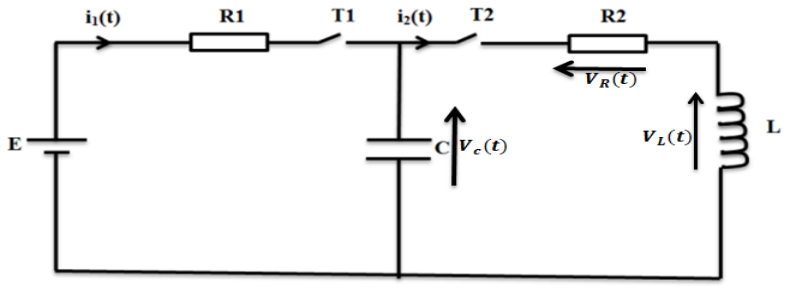


Figure III-1 : Schéma électrique équivalent d'une phase du système.

E : tension d'alimentation.

$i_1(t)$: courant de charge.

R_1 : résistance de charge.

T_1 : interrupteur qui contrôle la charge du condensateur.

C : condensateur.

$i_2(t)$: courant de la décharge.

R_2 : résistance du circuit de puissance.

L : inductance du circuit de puissance.

T_2 : interrupteur qui contrôle la décharge du condensateur.

N.B : les interrupteurs T_1 et T_2 ne conduisent pas simultanément.

III.1.1. Phase de la charge du condensateur

Pour charger le condensateur on ferme l'interrupteur T_2 et T_1 restera ouvert, donc on obtient le circuit suivant :

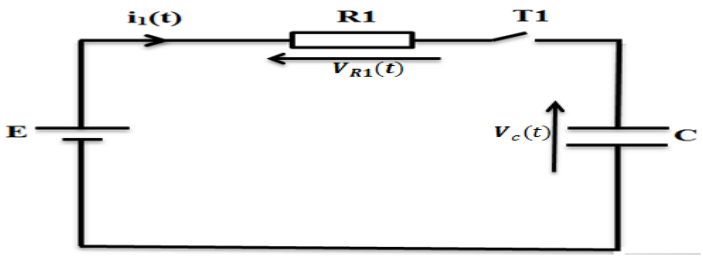


Figure III-2 : Circuit de la charge du condensateur.

De la loi des mailles on obtient l'équation suivante

$$E = R_1 i_1(t) + V_C(t) \quad \text{(III-1)}$$

On a:

$$i_1(t) = c\frac{dV_c(t)}{dt} \quad \text{(III-2)}$$

En remplaçant l'expression (III-2) dans l'expression (III-1) on aura:

$$E = R_1 c\frac{dV_c(t)}{dt} + V_C(t) \quad \text{(III-3)}$$

L'équation obtenue est une équation différentielle du premier ordre. En utilisant la transformée de LAPLACE pour résoudre l'expression (III-3) on obtient :

$$\frac{E}{P} = R_1 C[PV_C(p) - V(0)] + V_C(p) \quad \text{(III-4)}$$

Avec $V_C(0)=0$ (car le condensateur est initialement déchargé).

Donc on obtient l'expression suivante :

$$V_C(p) = \frac{E}{p(R_1 Cp+1)} \quad \text{(III-5)}$$

Par identification on obtient :

$$V_C(p) = \frac{E}{p} + \frac{E}{\left(p+\frac{1}{R_1 C}\right)} \quad \text{(III-6)}$$

La solution de l'équation est donnée par l'équation suivante :

$$V_C(t) = E\left(1 - \exp\left(\frac{-t}{\tau}\right)\right) \quad \text{(III-7)}$$

Tel que : $$\tau = R_1 C \quad \text{(III-8)}$$

τ : Constante du temps, est définit comme étant le temps au bout duquel le condensateur est chargé à 63% de la valeur finale de la tension (V(∞)).

A partir de l'équation de la tension de charge on peut définir celle du courant de charge comme suite :

On a de l'expression (III-1) :

$$E = R_1 i_1(t) + V_C(t) \quad \text{(III-1)}$$

Donc :

$$i_1(t) = \frac{E - V_C(t)}{R_1} \quad \text{(III-9)}$$

$$i_1(t) = \frac{E}{R_1}\exp\left(\frac{-t}{\tau}\right) \quad \text{(III-10)}$$

III.1.2. Phase de la décharge du condensateur

A la fin de la première phase on aura les condensateurs complétement chargés, donc on peut passer à la phase suivante, alimenter le circuit de puissance en déchargeant les trois

condensateurs avec un déphasage de (2π/3) entre la première et la deuxième phase et de même entre la deuxième et la troisième.

On va étudier une seule phase (la première dans cet exemple), on assimile celle-ci à un circuit RLC avec :

$R=R_2$: résistance du circuit de puissance.

L : inductance du circuit de puissance.

On ouvre T_1 et T_2 fermé, on obtient le circuit RLC donné par la figure suivante :

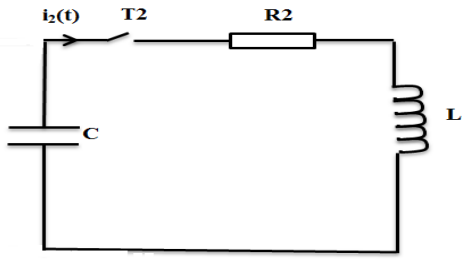


Figure III-3 : Circuit de la décharge.

Le circuit ne comporte qu'une seule maille, donc à partir de la loi des mailles on obtient l'équation des tensions suivantes :

$$V_C(t) + R_2 i_2(t) + L\frac{di_2(t)}{dt} = 0 \qquad \text{(III-11)}$$

$$i_2(t) = -C\frac{dV_c(t)}{dt} \qquad \text{(III-12)}$$

En remplaçant (III-12) dans (III-11) on aura :

$$V_C(t) + C\frac{dV_c(t)}{dt} + LC\frac{d^2V_c(t)}{dt^2} = 0 \qquad \text{(III-13)}$$

Ce qui donne une équation différentielle de deuxième ordre en Vc(t). C'est l'équation différentielle que nous allons résoudre. En utilisant la transformée de LAPLACE :

$$V_C(p) = V_C(t_1)\frac{p+\frac{R_2}{L}}{p^2+\frac{R_2}{L}p+\frac{1}{LC}} \qquad \text{(III-14)}$$

La résolution de l'équation différentielle (III-12) s'effectue en écrivant l'équation caractéristique :

$$p^2 + \frac{R_2}{L}p + \frac{1}{LC} = 0 => p^2 + 2zwp + w^2 = 0 \quad \text{(III-15)}$$

Avec : $w = \frac{1}{\sqrt{LC}}$ la pulsation propre.

$z = \frac{R_2}{2Lw}$ Coefficient d'amortissement.

De l'équation (III-15) on aura :

$$(p + zw)^2 + w^2(1 - z^2) = 0 \quad \text{(III-16)}$$

Selon la valeur de z on aura trois cas passibles :

- $1\text{-}z^2<0$: l'équation admet deux racines réelles,
- $1\text{-}z^2=0$: l'équation admet une racine double,
- $1\text{-}z^2>0$: l'équation admet deux racines complexes,

Dans notre cas, nous s'intéresserons au troisième cas. Cas du régime pseudopériodique Ce cas correspond à un faible amortissement, ou un grand facteur de qualité. Les 2 solutions de l'équation caractéristique (III-16) sont complexes :

L'expression (III-14) peut s'écrire comme suite :

$$V_C(t) = E\frac{(p+2zw)}{(p+zw)^2+w^2(1-z^2)} \quad \text{(III-17)}$$

$$V_C(t) = E\frac{p+zw}{(p+zw)^2+w^2(1-z^2)} + E\frac{zw}{(p+zw)^2+w^2(1-z^2)} \quad \text{(III-18)}$$

En utilisant la transformée inverse de LAPLACE, nous obtiendrons la tension $V_C(t)$ aux bornes du condensateur :

$$V_C(t) = E\left[C_1\cos\left(w_p t + \varphi\right)\right]\exp(-\alpha t) \quad \text{(III-19)}$$

Avec : $$C_1 = \sqrt{1 + (\frac{\alpha}{w_p})^2} \quad \text{(III-20)}$$

$$\varphi = arctg(\frac{-\alpha}{w_p}) \quad \text{(III-21)}$$

Pseudo pulsation

$$w_p = \sqrt{w^2 - \alpha^2} \quad \text{(III-22)}$$

Pseudo période

$$T_p = \frac{2\pi}{w_p} \quad \text{(III-23)}$$

Facteur d'amortissement

$$\alpha = zw \quad \text{(III-24)}$$

La pseudo-période de ce signal représente le temps qui sépare 2 passages à 0 de la tension où de l'intensité,

L'expression du courant de décharge est donnée par l'équation suivante :

$$i_2(t) = EC\ (w_p + \frac{\alpha^2}{w_p}) sin(w_p t) \exp\ (w_p t) \quad \text{(III-25)}$$

III.2. Réalisation du circuit de commande

III.2.1. Schéma synoptique de la commande

La commande de ce système est manuelle, on utilisera trois boutons poussoirs à l'entrée de la commande représentant trois signaux de commande (e1, e2, e3), à la sortie par contre on aura quatre signaux (s1, s2, s3, s4). Un schéma synoptique de la commande est donné par la figure III-4.

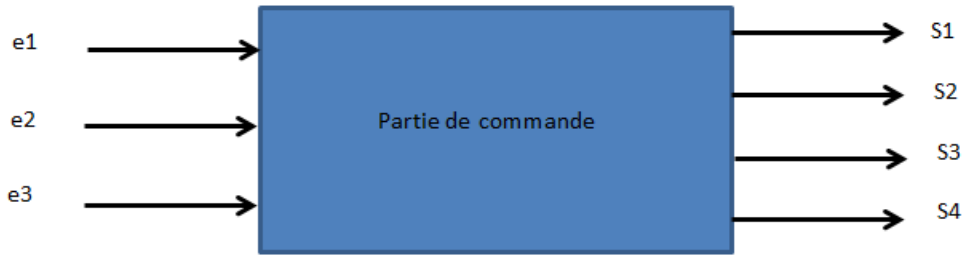


Figure III-4 : Schéma synoptique de la commande.

Le fonctionnement de cette commande doit passer par quatre étapes essentiales,

- Etape 1 : les trois condensateurs sont déchargés.
- Etape 2 : la charge des trois condensateurs simultanément.
- Etape 3 : fin de la charge des trois condensateurs.
- Etape 4 : la décharge des trois condensateurs avec un déphasage de $\frac{2\pi}{3}$ entre le premier

et le deuxième et même déphasage entre le deuxième et le troisième.

Ces étapes se terminent par la décharge des trois condensateurs dans le circuit de puissance du lanceur électromagnétique et retour à l'étape 1 comme le montre le chronogramme suivant :

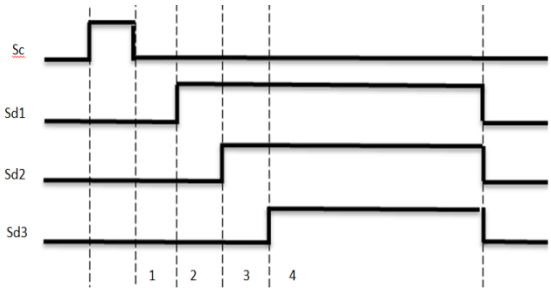


Figure III-5 : Chronogramme des signaux de commande.

Avec : Sc : signal de commande de la charge des trois condensateurs.

Sd1 : signal de commande de la décharge du premier condensateur.

Sd2 : signal de commande de la décharge du deuxième condensateur.

Sd3 : signal de commande de la décharge du troisième condensateur.

Du chronogramme représenté dans la figure III-5, on obtiendra la séquence suivante (000-001-011-111) qui indique les différents états que va prendre la sortie du système de commande pour les différentes étapes de fonctionnement du système. Donc pour contrôler la décharge des condensateurs on réalisera un compteur synchrone qui nous donne les mêmes séquences. En utilisera pour la réalisation de ce compteur des bascules (JK) avec une entrée clr (remise à zéro) et une entrée pr (mise à zéro).

La bascule JK est une bascule RS plus élaborée et ne possédant pas d'indétermination, la combinaison (J, K)=(1,1) est maintenant applicable. La bascule JK s'obtient à partir d'une bascule RS en faisant [6] :

$$S = J\overline{Q} \qquad et \qquad R = KQ$$

III.2.2. Réalisation du circuit de commande de la décharge des condensateurs

Un compteur est un dispositif affichant certains états sur une commande extérieure, appelée impulsion de progression ou d'avancement. L'emploi des compteurs ne se limitepas aux systèmes arithmétiques, ils sont utilisés lorsqu'on désire définir une succession d'états.

On appelle un compteur modulo N= 2^n, un compteur décrivant la succession des nombres binaires compris entre (0) et (N-1), c'est-à-dire la suite des chiffres d'une base N traduite en binaire [6].

La synthèse d'un compteur consiste à déterminer les fonctions de commande des bascules dans les deux structures (synchrone et asynchrone), autrement dit à déterminer toutes les liaisons à effectuer entre les entrée et les sorties des bascules.

Le problème posé sur la synthèse des compteurs est la suivante : à un instant (t), la sortie d'une bascule étant Q_t, quelles sont les valeurs à appliquer aux entrées pour qu'à l'impulsion d'horloge suivante la sortie prenne la valeur Q_{t+1}, imposée par le cycle de comptage désiré ? [6]

Pour réaliser un compteur synchrone, on procède de la manière suivante :

- Définir la capacité maximale du compteur, c'est-à-dire le plus grand nombre binaire que le compteur peut afficher. Ceci permet de déterminer le nombre d'étages donc le nombre des bascules à utiliser.
- Choisir le code à utiliser. Les sorties du compteur occupent des états dépendant du code utilisé.
- Choisir le type des bascules.
- Etablir le tableau des états présents-états futurs pour toutes les bascules.
- Déduire la table de transition pour chaque bascule.
- A l'aide des tables de vérité réduites des bascules utilisées, déterminer les valeurs des entrées de chacune des bascules qui définissent l'état du compteur à chaque impulsion d'horloge.
- Déduire les équations logiques des entrées de chaque bascule en fonction des sorties des différentes bascules en utilisant les diagrammes de KARNAUGH.
- Faire le schéma du compteur.

Dans notre cas, le compteur à réaliser est un compteur synchrone modulo 8, donc N=8=2^3, d'où l'utilisation de trois bascules JK. Le compteur à réaliser passera par quatre étapes donc par quatre états :

Etat 1 : toutes les bascules sont à l'état zéro.

Etat 1 = $\overline{Q_A Q_B Q_C}$

Etat 2 : la première bascule est à un, la deuxième et la troisième sont à l'état zéro.

Etat 2 = $Q_A \overline{Q}_B \overline{Q}_C$

Etat 3 : la première et la deuxième bascule sont à un, la troisième est à zéro.

Etat 3 = $Q_A Q_B \overline{Q}_C$

Etat 4 : toutes les bascules sont à l'état un.

Etat 4 = $Q_A Q_B Q_C$

Dans le tableau III-1 sont représentés les différents états selon les séquences données précédemment.

Séquence	Etats de la sorties		
	Q_A	Q_B	Q_C
000	0	0	0
100	1	0	0
110	1	1	0
111	1	1	1

Tableau III-1 : Etats à la sortie des bascules.

Connaissant les sorties des bascules présentes et futures, on peut élaborer les expressions desentrées J et K, qui sont représentées dans le tableau III-2 :

Q_A	Q_B	Q_C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	1	Φ	0	Φ	0	Φ
1	0	0	Φ	0	1	Φ	0	Φ
1	1	0	Φ	0	Φ	0	1	Φ
1	1	1	Φ	1	Φ	1	Φ	1

Tableau III-2 Tableau des états-présents et états-futures et table de transition.

Pour obtenir les expressions des sorties J et K des bascules, on utilisera la méthode de simplification de KARNAUGH.

- Expression de J_A :

Q_C \ Q_AQ_B	00	01	11	10
0	1	Φ	Φ	Φ
0	Φ	Φ	Φ	Φ

Tableau III-3 Expression de J_A

- Expression de K_A :

Q_C \ Q_AQ_B	00	01	11	10
0	Φ	Φ	1	0
0	Φ	Φ	1	Φ

Tableau III-4 Expression de K_A

- Expression de J_B :

Q_C \ Q_AQ_B	00	01	11	10
0	0	0	Φ	1
0	Φ	Φ	Φ	Φ

Tableau III-5 Expression de J_B

- Expression de J_B :

Q_C \ Q_AQ_B	00	01	11	10
0	Φ	Φ	0	Φ
0	Φ	Φ	1	0

Tableau III-6 Expression de K_B

- Expression de J_C :

Q_AQ_B / Q_C	00	01	11	10
0	0	Φ	1	0
0	Φ	Φ	Φ	Φ

Tableau III-7 Expression de J_C

- Expression de K_C :

Q_AQ_B / Q_C	00	01	11	10
0	Φ	Φ	Φ	Φ
0	Φ	Φ	1	Φ

Tableau III-8 Expression de K_C

On obtient les expressions suivantes :

$J_A = 1$; $K_A = Q_C$;

$J_B = Q_A$; $K_B = Q_C$;

$J_C = Q_B$; $K_C = 1$;

Ce compteur est synchrone, donc il sera commandé par un signal d'horloge d'où l'utilisation d'une commande de décharge qu'on nommera M_d à l'entrée de l'horloge, donc CLK =H*M_d. Mais une fois l'état (111) à la sortie du compteur est atteint, un front montant du signal d'horloge remettra le compteur à zéro, d'où l'incertitude d'une décharge complète des condensateurs, pour remédier à ce problème, on doit annuler le signal de l'horloge CLK. Si onreliera lasomme destrois sorties $\overline{Q_A}, \overline{Q_B}$ et $\overline{Q_C}$ à l'horloge, à l'état (111) on aura $\overline{Q_A} + \overline{Q_B} + \overline{Q_C} = 0$, donc le signal CLK=0.

Donc le signal de l'horloge sera composé de M_d (commande de la décharge), le signal H (horloge principale) et la somme ($\overline{Q_A} + \overline{Q_B} + \overline{Q_C}$).

$$CLK = (\overline{Q_A} + \overline{Q_B} + \overline{Q_C}) M_d H.$$

Le fonctionnement de la commande de décharge des condensateurs sera assuré par le compteur réalisé précédemmentd'où le fonctionnement suivant :

Initialement : $Q_A = Q_B = Q_C = 0$

- Si la commande de décharge est désactivée

M_d=0, donc CLK sera nul, pas de changement d'état du compteur($Q_A = Q_B = Q_C = 0$).

- Si la commande de décharge est activée M_d=1
    - Au premier front montant du signal d'horloge $Q_A = 1\ et\ Q_B = Q_C = 0$.
    - Au deuxième front montant du signal d'horloge $Q_A = Q_B = 1\ et\ Q_C = 0$.
    - Au troisième front montant du signal d'horloge $Q_A = Q_B = Q_C = 1$.

A cette étape, on a sur la sortie du compteur$\overline{Q_A} + \overline{Q_B} + \overline{Q_C} = 0$, donc CLK est nul. Par mesure de sécurité on reliera l'inverse de la somme ($\overline{Q_A} + \overline{Q_B} + \overline{Q_C}$) à l'entrée P_r (mise à un des bascules JK qui s'active à un).

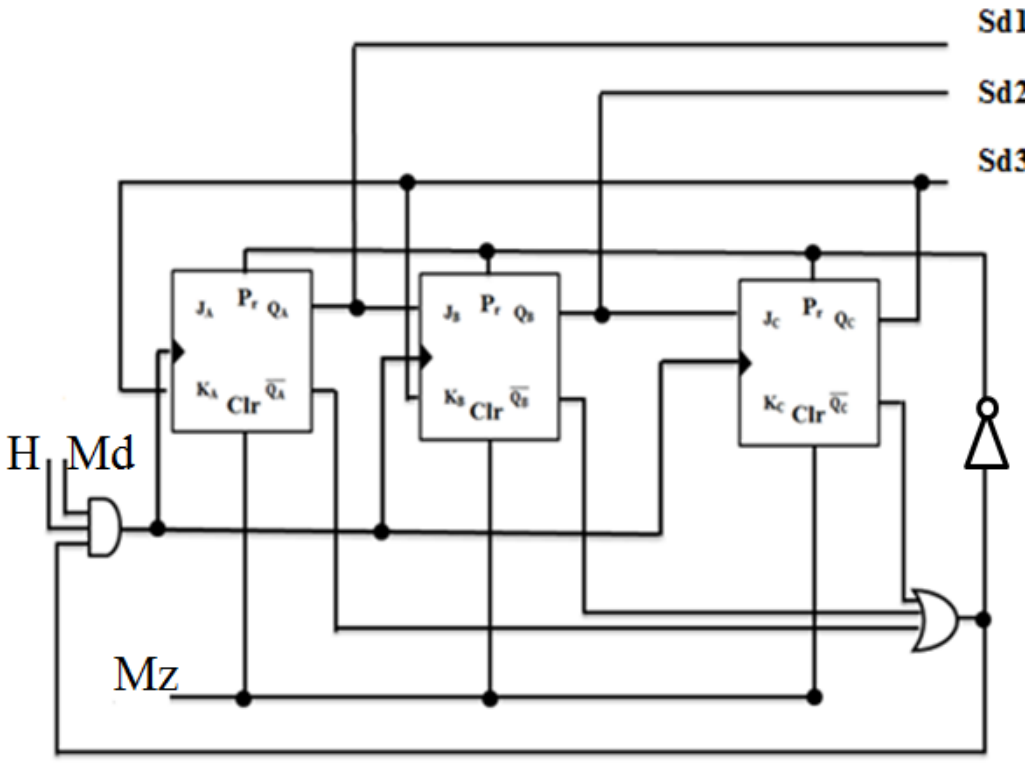


Figure III-6 : Circuit de commande de la décharge des condensateurs.

III.2.3.Réalisation du circuit de commande de la charge des condensateurs

Pour recharger les condensateurs, on désactive la commande de décharge, on remet les bascules JK (le compteur) à zéro. Pour cela on utilise un bouton poussoir qui sera relié aux entrées Clr (remise à zéro) de chaque bascule JK.

Les trois condensateurs se charge simultanément donc leurs contrôles s'effectuent par une seule bascule. On utilisera un bouton poussoir qu'on nommera Mc, pour arrêter la charge on utilisera le bouton de remise à zéro (Mz), donc on peut avoir Mc=Mz=1 (interdiction), de là l'utilisation d'une bascule RS à la commande à la commande de la charge des condensateurs est évidente.

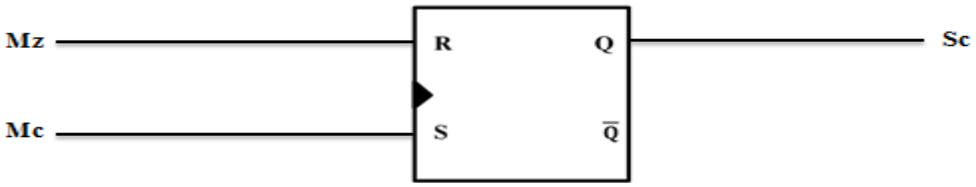


Figure III-7 : Circuit de commande de la charge des condensateurs.

En fin, la charge et la décharge seront contrôlées par les états de sorties des deux circuits donnés par les figures III-6 et III-7, comme indiqué dans le tableau suivant :

Q_A	Q_B	Q_C	Q
0	0	0	1
1	0	0	0
1	1	0	0
1	1	1	0

Tableau III-9 Les états de sorties du circuit de commande.

Il suffit de lier ces deux circuits pour assurer la commande de la charge et la décharge des trois condensateurs. Par mesure de sécurité on reliera le produit des sorties $\overline{Q_A}, \overline{Q_B}$ et $\overline{Q_C}$ à l'entrée S de la bascule RS.

$$S=\overline{Q_A Q_B Q_C} Mc.$$

$$R=Mz.$$

Le schéma de commande est donné par la figure III-8.

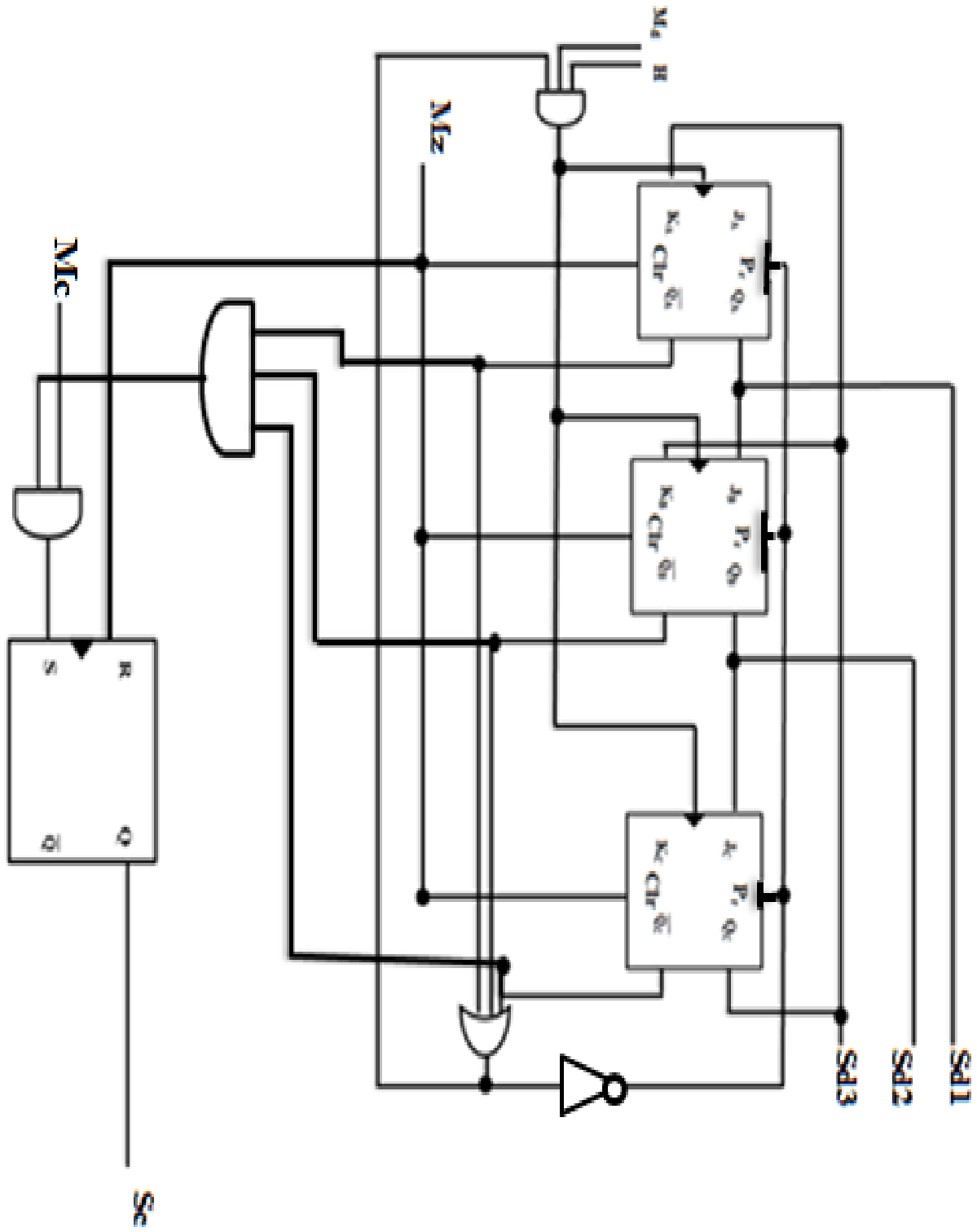


Figure III-8 : circuit de commande.

Conclusion

Dans ce troisième chapitre on a réaliséla modélisation du système, il est composé de deux parties essentielles :

- La première partie, sous forme d'un circuit RC, en alimentant le condensateur il se charge à travers la résistance (R1), donc celui-ci représente le circuit de charge du condensateur.
- La deuxième partie, sous forme d'un circuit RLC, si on ouvre l'interrupteur le condensateur se décharge en alimentant le circuit de puissance du système. Donc cette partie représente le circuit de décharge.

Le but de cette modélisation est de réaliser le circuit de la commande de l'ouverture et de la fermeturedes interrupteurs. Pour cela on a réalisé une commande avec trois entrées (Mz : remise à zéro, Mc : charge des condensateurs, Md : décharge des condensateurs) et quatre sorties (Sc : commande les interrupteurs des circuits de charge, Sd1, Sd2 et Sd3 : commandent les interrupteurs des circuits de décharge).

CHAPITRE IV

Réalisation et résultats expérimentaux

Introduction

Ce quatrième et dernier chapitre représente la partie expérimentale de notre projet qui consiste à la réalisation d'un circuit de commande d'un dispositif électromagnétique (lanceur électromagnétique). Ce système de commande est réalisé à l'aide d'un Field programmable GateArray (FPGA) qui a trois entrées de commande (mise à zéro, charge des condensateurs, décharge des condensateurs) et donne en sortie quatre signaux de commande de l'ouverture et fermeture des interrupteurs (transistors) pour alimenter le circuit de puissance. Le schéma de la commande réalisé est donné dans le chapitre III.

Mais, avant de donner les résultats de simulation du programme à implémenter dans le circuit FPGA, on commence par la simulation (sur MATLAB) de charge et décharge des condensateurs dans les trois phases du dispositif.

IV.1. Simulation des circuits de charge et de décharge

IV.1.1. Visualisation des courants et des tensions de charge

Soient les expressions de courant et de tension obtenues précédemment (chapitre III) :

$$V_C(t) = E(1 - \exp\left(\frac{-t}{\tau}\right)) \quad \text{(III-7)}$$

$$i_1(t) = \frac{E}{R_1}\exp(\frac{-t}{\tau}) \quad \text{(III-9)}$$

Les paramètres du circuit de charge :

R1 : 20 Ω,

C : 150.10^{-6} F,

E1 : 350V,

Visualisation de la tension de charge

La courbe de la tension de charge est donnée par la figue suivante :

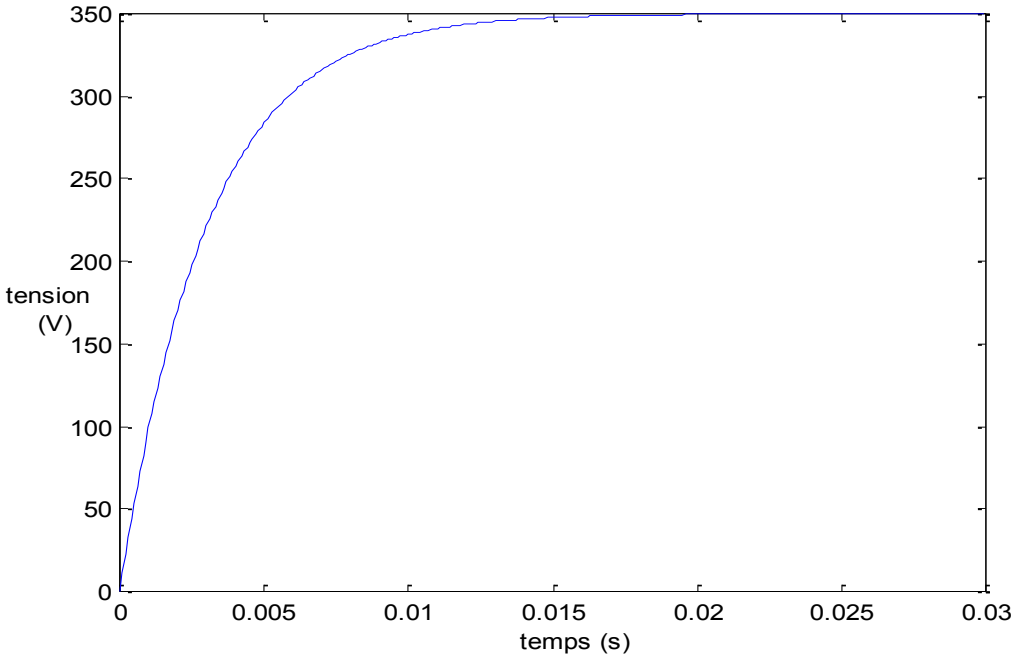


Figure IV. 1 : Tension de charge du condensateur

Visualisation du courant de charge :

La courbe du courant de charge est donnée par la figue suivante :

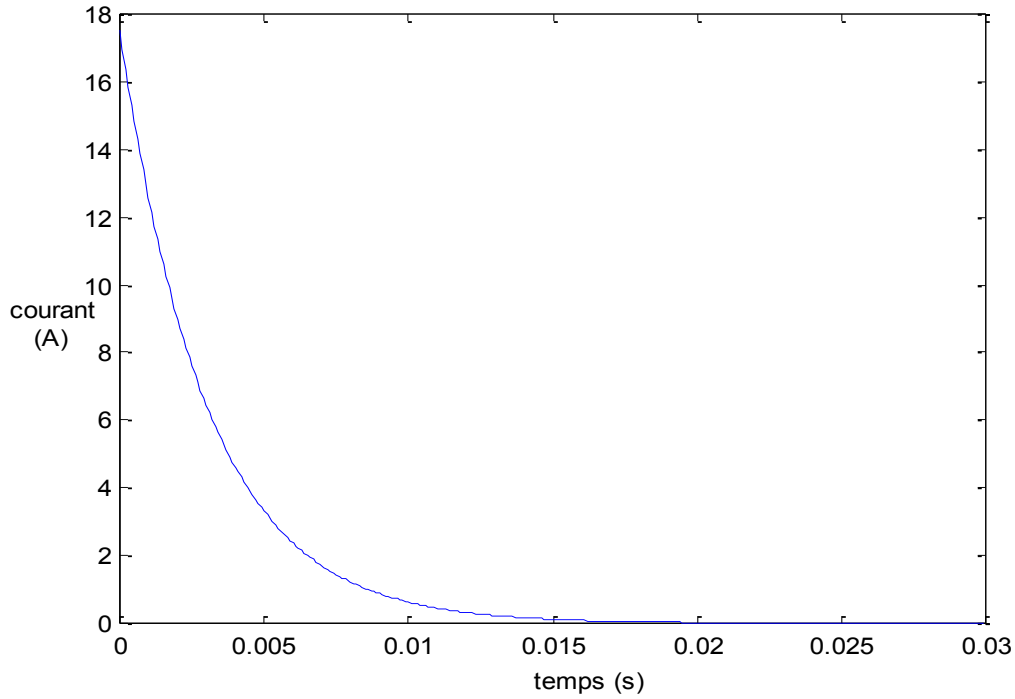


Figure IV-2 : Courant de charge du condensateur

Comme les trois condensateurs se chargent en même temps, on aura les mêmes caractéristiques pour les trois phases, donc elles seront superposées l'une sur l'autre.

L'équation qui régit le circuit de charge, est une équation différentielle du premier ordre, donc le système est dit de premier ordre, ce qui explique la forme des courbes obtenues données par les figures (IV-1 et IV-2). Les courbes de la tension et du courant prennent respectivement une forme exponentielle croissante et décroissante, elles ont la même constante de temps $\tau=R_1C$, qui est définit aussi comme étant le temps nécessaire pour que les caractéristiques atteignent 63% de la valeur finale, cette constante caractérise la rapidité du système. On peut définir aussi le temps de réponse du système, qui ne peut être que $Tr=3\tau$, au bout de ce temps, le système aura atteint 95% de la valeur finale.

$Vc(\infty)=E$, la valeur finale de la tension.

$I_1(\infty) = 0$,la valeur finale du courant.

T= R1C=0.003 s, constante de temps.

$Tr=3*\tau$= 0.009 s, temps de réponse.

IV.1.2. Visualisation des tensions de décharge

Soit l'expression de la tension obtenue précédemment :

$$V_C(t) = E[C_1\cos(w_p t + \varphi)]\exp(-\alpha t) \qquad \text{(III-18)}$$

L'alimentation des trois phases du lanceur électromagnétique s'effectue avec des tensions déphasées entres elles de $(2\pi/3)$ entre la première et la deuxième phase, et de $(4\pi/3)$ entre la deuxième phase et la troisième phase. Pour cela, les trois condensateurs seront alimentés comme suite :

Le premier condensateur : E_1=350 V
Le deuxième condensateur : E_2=-350 V
Le troisième condensateur : E_3=350 V.

Dans ce programme on imposera le courant maximal dans le circuit de charge, donc la résistance du circuit de charge sera calculée pour ne pas dépasser le courant maximal.

Visualisation des tensions de décharge :

Les courbes des tensions de décharge sont données par la figure suivante :

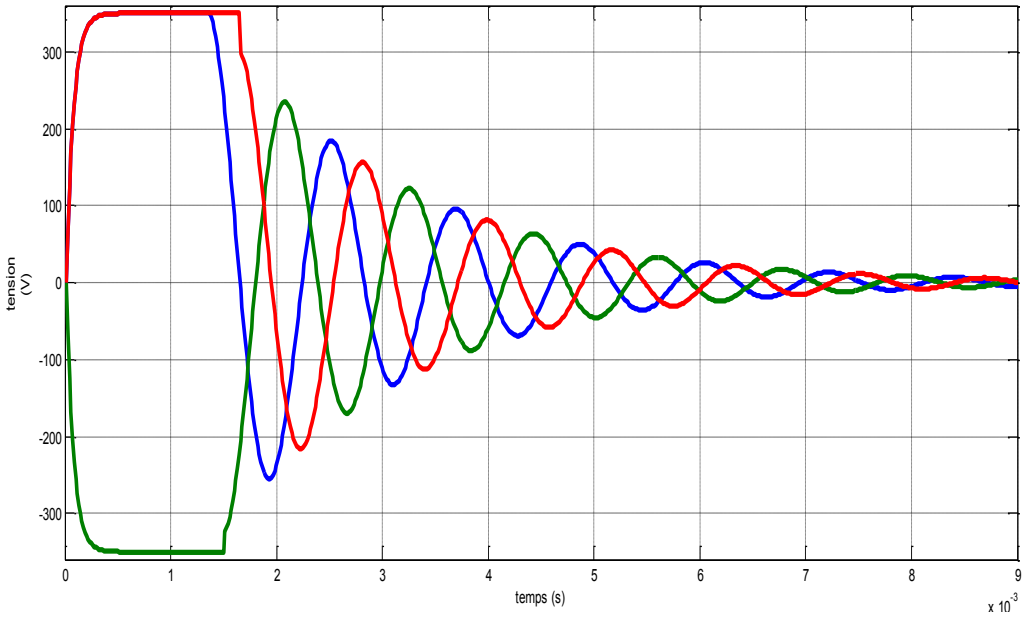


Figure IV-3: Tensions de décharge.

Le déchargement des trois condensateurs s'effectue avec un déphasage de (2π/3) entre le premier et le deuxième condensateur, et de même entre le deuxième et le troisième condensateur, entre le premier condensateur et le troisième le déphasage est de (4π/3).

On voit qu'au bout d'un certain temps, les tensions s'annulent, et ceci est dû à la présence de la résistance R_2, qui provoque la dissipation de l'énergie stockée dans les condensateurs par effet Joule et un amortissement plus au moins rapide.

Les tensions fournissent alternativement des valeurs négatives et positives, elles oscillent. Les tensions subirent des oscillations libres amorties dont la valeur maximale décroît et intervient à intervalles de temps égaux. On dit que le régime des oscillations est pseudopériodique. Les courbes de la tension sont sous amorties, car le taux d'amortissement est inférieur à un (α=0.1033), ce régime est le produit d'une exponentielle décroissante et d'une sinusoïde. La décroissance des amplitudes de la tension est due à l'énergie dissipée par effet Joule dans le circuit de décharge. Il y a transfert d'énergie du condensateur vers la bobine et réciproquement de façon alternative.

IV.2. Programmation en VHDL

Dans ce travail on a utilisé le logiciel Quartus II, ce logiciel permet de compiler, de synthétiser, de simuler le fonctionnement du circuit avant implémentation sur un FPGA. Plus de détails sur le fonctionnement de celui-ci sont donnés dans l'annexe.

Ce circuit de commande est composé d'un compteur modulo_8 pour commander la décharge des condensateurs et d'une bascule RS à base des portes OR pour commander la charge des condensateurs. Un schéma simplifié du circuit à programmer est donné par la figure suivante :

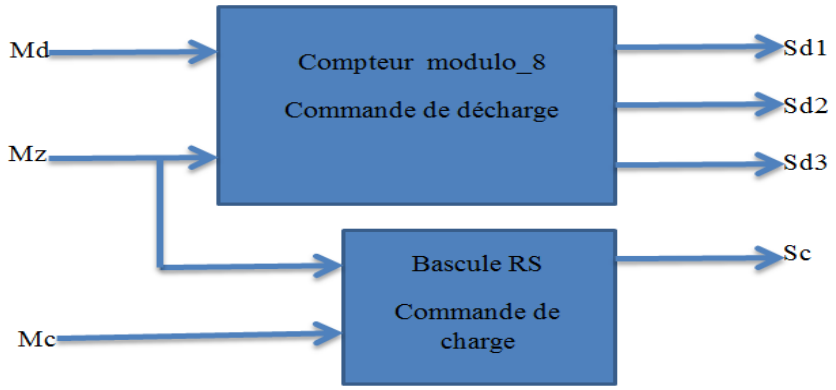


Figure IV-4 : Schéma simplifié du circuit de commande.

Md : commande de la décharge ;

Mc : commande de la charge ;

Mz : commande de remise à zéro ;

Sc : sortie de commande de la fermeture des transistors du circuit de charge ;

Sd_1, Sd_2, Sd_3 : sorties de commande de la fermeture des transistors du circuit de décharge.

IV.2.1. Description du code VHDL du circuit de commande

Le code VHDL utilisé pour générer les signaux de commande de charge et décharge des condensateurs est constitué de deux parties principales, la première permet de générer le signal de commande de la charge des trois condensateurs simultanément, la deuxième partie est celle de la commande de la décharge des trois condensateurs avec un déphasage de $(2\pi/3)$.

IV.2.1.1. Circuit de la commande de charge

Il est à base d'une bascule RS, commandée par deux signaux d'entrée, la mise à zéro de ce circuit (Mz) et la commande de charge des condensateurs (Mc).

Les résultats de la simulation de cette bascule sont donnés par la figure suivante :

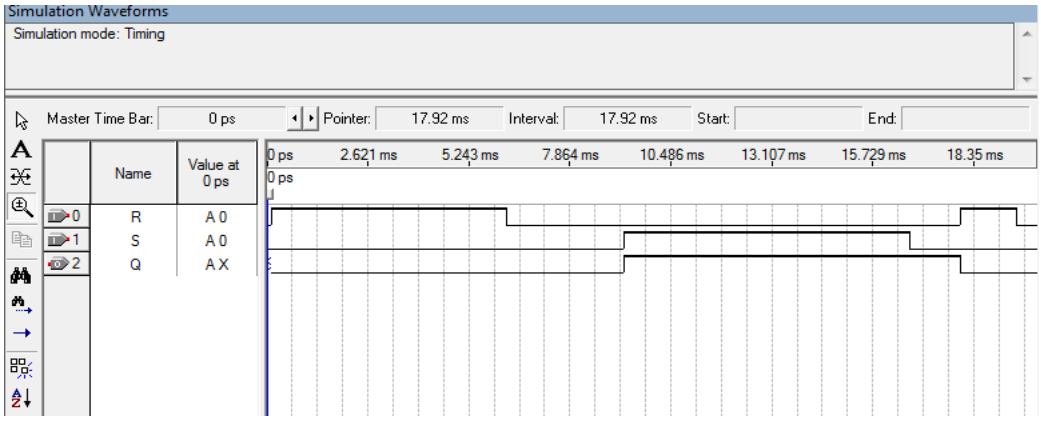


Figure VI-5: Simulation du fonctionnement de la bascule RS.

Dans cette figure l'entrée est liée au signal de commande de la charge des condensateurs (Mc) et l'entrée R représente le signal de la mise à zéro de ce circuit. La sortie de cette bascule (Q) est le signal de commande de la fermeture des transistors du circuit de charge. On voit que si (Mz=1) donc (R=1) le sortie sera (Q=0) donc la mise à zéro, pour enclencher la charge des condensateurs il faut avoir (S=1) donc activer la commande de charge (Mc=1).

IV.2.1.2. Circuit de la commande de la décharge

Ce circuit est composé de trois entités à base d'une bascule JK qui est représentée sous forme d'un compteur modulo_8.

Les résultats de la simulation du fonctionnement de la bascule JK sont donnés par la figure suivante :

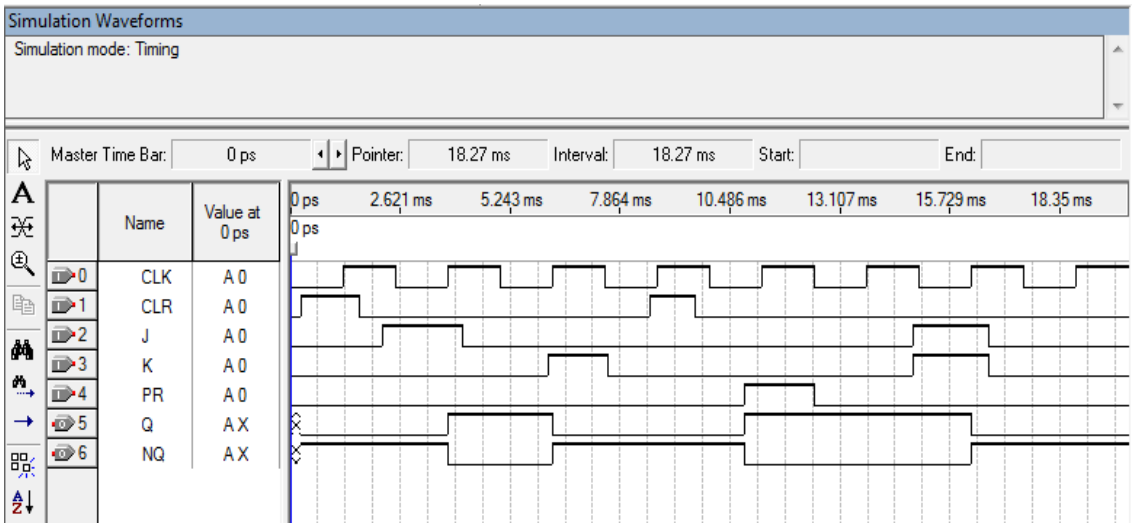


Figure IV-6 : Simulation du fonctionnement de la bascule JK.

Cette bascule a deux modes de fonctionnement, le premier est le mode asynchrone à l'aide des entrées asynchrones (CLR : mise à zéro, PR : mise à un), le deuxième mode de fonctionnement est le mode synchrone en fonction de l'horloge à l'entrée (CLK) et l'état des entrées (J, K), on obtient les états de sorties (Q).

IV.2.1.3. Schéma du circuit de commande

Le schéma de ce circuit est donné par la figure suivante :

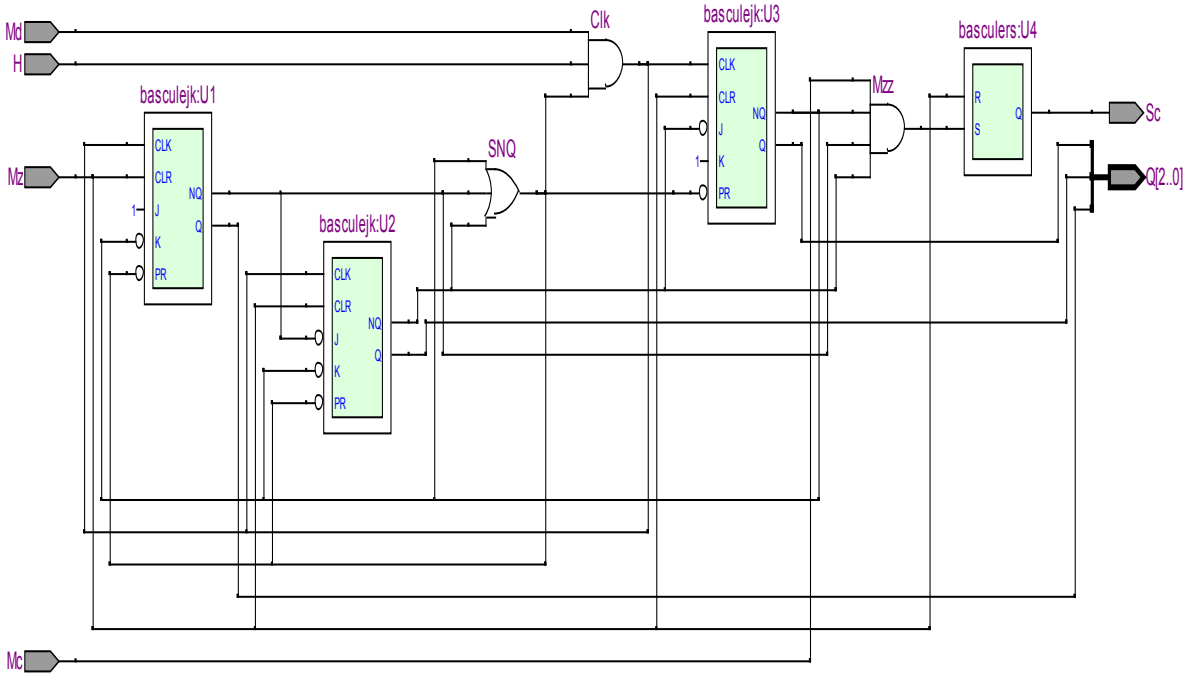


Figure IV-7 : Circuit de commande.

De la figure (IV-7) on voit que le système est commandé par les trois entrées (Md, Mz, Mc) qui sont des bottons poussoirs activés par l'utilisateur, et une horloge (H) qui sera générée automatiquement.

Avant de lancer la simulation de ce circuit, on doit d'abord choisir une fréquence d'horloge (f_H). La période du signal de l'horloge sera égale au temps représentant le déphasage entre les tensions de décharge.

Le déphasage est égal à :

$$dep = \frac{2*\pi}{3} \quad \text{(IV-1)}$$

Donc, le temps correspondant sera :

$$t_{dep} = \frac{dep}{w_p} \quad \text{(IV-2)}$$

D'où :

$$t_{dep} = 148.71 * \mu S$$

Et donc la période du signal d'horloge sera :

$$T_H = 1.2\ mS$$

La fréquence de l'horloge :

$$f_H = 851\ Hz$$

Nous allons chercher à valider notre conception en fonctionnel, pour cela, on utilise le simulateur Active HDL. Il faut, au préalable à la simulation, de définir des vecteurs de test et les appliquer comme forces sur les entrées du circuit à simuler, et définir le temps de simulation qui dépendra de la fréquence du signal d'horloge.La simulation de ce circuit donne les résultats suivants :

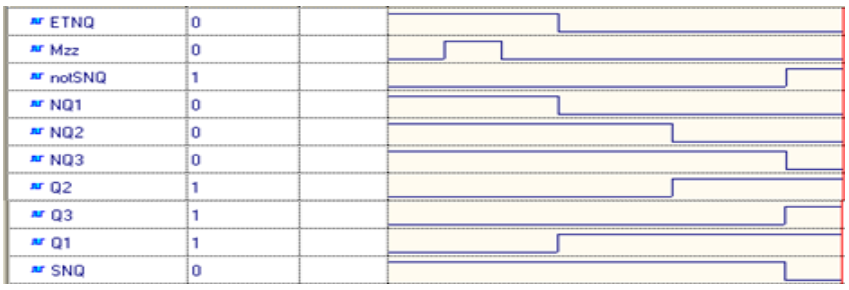


Figure IV-8:Résultats de simulation des signaux.

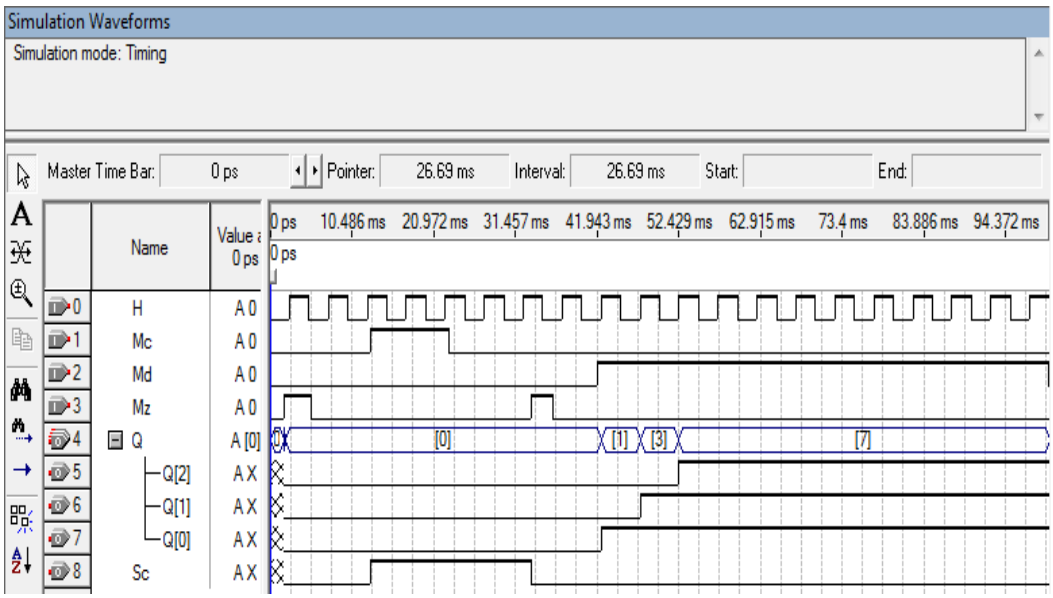


Figure IV-9 : Simulation de fonctionnement de circuit de commande.

A partir de la figure (IV-9), on voit que lorsque (Mz=1) toutes les sorties sont mises à zéro, la première étape est de charger les condensateurs, lorsque (Mc=1) on aura (Sc=1), puis on active (Mz) pour remettre (Sc) à zéro, dans ce cas on peut lancer la décharge des condensateurs en activant la commande (Md), et la décharge se déclenche automatiquement en fonction du signal de l'horloge (H). Ce circuit s'active sur les fronts montants du signal d'horloge.

A la troisième impulsion du signal d'horloge toutes les sorties de commande sont activées (Sd1=Sd2=Sd3=1) à part la commande de charge (Sc=0), en ce moment les sorties (NQ1=NQ2=NQ3=0).

On a :

$$CLK = H \text{ AND } Mc \text{ AND } SNQ$$

Et comme :

$$SNQ = NQ1 \text{ OR } NQ2 \text{ OR } NQ3$$

Donc le signal (CLK) s'annulera après l'activation des sorties de commande de décharge. Pour maintenir les sorties de commande de décharge activées on relira le signal (notSNQ figure (IV-8)) à l'entrée asynchrone (PR : mise à un) des bascules JK.

$$notSNQ = NOT\ (SNQ) = 1$$

IV.2.2. Implémentation sur la carte FPGA

Pour programmer le composant, il faut impérativement :

- que la DE02 soit reliée au PC par câble USB
- qu'elle soit sous tension (bouton rouge)
- que le commutateur RUN / PROG soit positionné sur RUN

L'implémentation (transfert de la séquence vers la puce) se fait par le biais d'un câble USB selon l'un de deux modes :

a- JTAG (Joint Test Action Group)

b- AS (Active Serial)

En mode JTAG, le fichier de configuration est envoyé directement au FPGA. Le design demeurera sur la puce tant que le kit de développement sera sous tension.

En mode AS, le fichier est transféré dans une puce de configuration qui contient une mémoire de type Flash. Le design sera alors conservé dans cette puce secondaire même lorsque le kit sera mis hors tension et sera ainsi transféré automatiquement vers le FPGA lors de la mise sous tension.

Sur la plaquette, l'interrupteur SW19, situé à gauche de l'écran LCD, contrôle le mode d'implémentation.

Avant de lancer l'implémentation, il est nécessaire d'affecter les entrées sorties aux numéros de broches de notre circuit cible de sorte à être compatible avec la carte de développement. L'assembleur convertit automatiquement le dispositif, sous forme d'un ou plusieurs fichiers programmeur (.pof) ou SRAM (.sof) pour le périphérique cible (carte Altera).

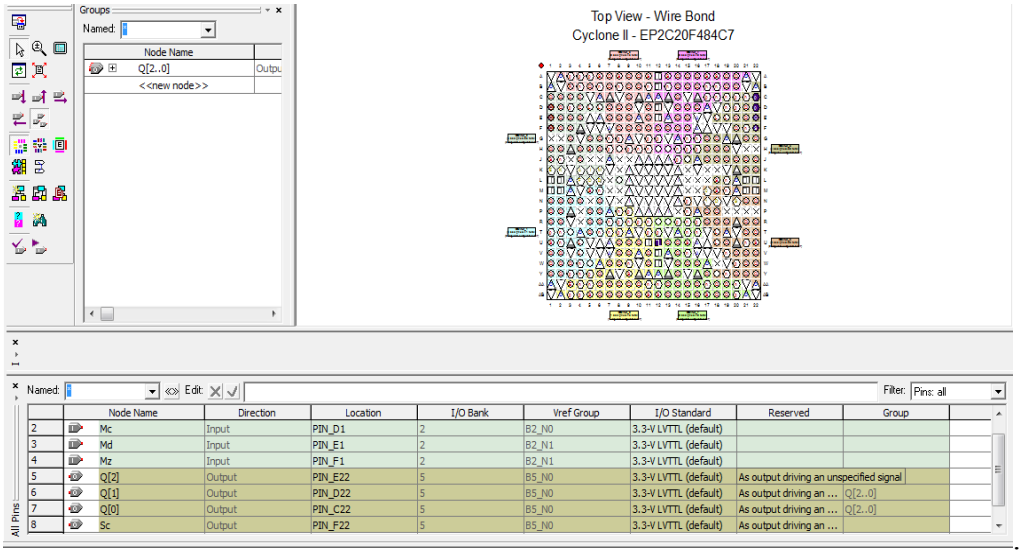


Figure IV-10 Choix des entrées et sorties sur la carte.

Pour implémenter ce programme sur la carte, sélectionner l'icône de programmation, La fenêtre suivante s'ouvre :

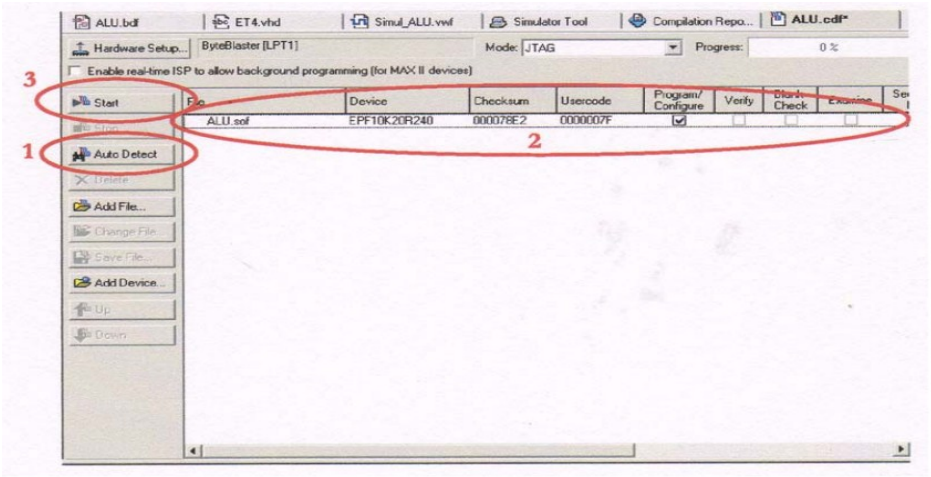


Figure IV-11 Implémentation du programme sur la carte Altera.

La programmation du circuit dans ce travail se fait avec le protocole (mode) JTAG (Joint Test Action Group).

- On vérifie que la connexion entre le PC (USB) et la carte est opérationnelle.
- On met sous tension la carte Altera
- On lance le programmateur par Tools →Programmer, puis on clique sur le bouton Autodetect pour sélectionner le ficher (.sof)
- In suffit de cliquer sur Start pour transférer le programme dans la carte.

Conclusion générale

Conclusion générale

L'objectif de ce travail et de mettre en évidence un circuit qui permettra de commander un système électromagnétique, donc notre travail a porté sur l'implémentation d'une commande d'un dispositif électromagnétique (lanceur électromagnétique) sur un FPGA.

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

- En premier lieu, nous avons vu les différentes caractéristiques des circuits logiques programmables en particulier les FPGA, ces circuits sont la solution idéale pour l'implémentation des circuits de grande densité. De cette façon on a vu les différents atouts des circuits FPGA.
- Pour la programmation des circuits FPGA, différents langages comportementaux qui sont le fruit de perfectionnement des descriptions matérielles. Parmi ces langages, ils existent deux, qui sont les plus utilisés : le VHDL et le VERILOG. Dans ce travail on a utilisé le premier langage, pour sa simplicité en plus de sa puissance.

Avant d'aller à la programmation, on doit d'abord définir les différents composants de circuit de commande, une fois toute est défini, en utilisant le langage VHDL pour programmer ce circuit, en suite on implémente celui-ci sur le FPGA après qu'il soit validé par la simulation.

Les résultats expérimentaux obtenus nous ont permis de démontrer le bon fonctionnement de notre circuit mis en œuvre et de saisir l'importance de l'utilisation des circuits logiques programmables FPGA de par les nombreux avantages qu'ils offrent.

Annexes

Annexes

I) Normes IEEE

Voici la liste des normes IEEE pour le langage VHDL. Nous les avons classées selon leur date d'édition. [18]

1987 Standard IEEE Std 1076-1987 (VHDL 87)	1996 Standard IEEE Std 1076.2 (Mathematical Packages)
1993 Standard IEEE Std 1164-1993 (Std_Logic_1164)	1997 Standard IEEE Std 1076.3 (Numeric_Bit et Numeric_Std)
1993 Standard IEEE Std 1076-1993 (VHDL 93)	1999 Standard IEEE Std 1076.6 (Standard for VHDL Register Transfer Level Synthesis)
1994 Approbation ANSI (ANSI/IEEE Std 1076-1993)	2000 : Standard IEEE Std 1076-2000 (VHDL 2000)
1995 Standard IEEE Std 1076.4 (Vital_Primitive et Vital_Timing)	2002 : Standard IEEE Std 1076-2002 (VHDL 2002)

II) Les mots réservés du VHDL.

Abs	Entity	nor	select
access	exit	not	severity
after	file	null	signal
alias	for	of	shared
all	function	on	sla
and	generate	open	sll
architecture	generic	or	sra
array	group	others	srl
assert	guarded	out	subtype
attribute	if	package	then
begin	impure	port	to
block	in	postponed	transport
body	inertial	procedure	type
buffer	inout	process	unaffected
bus	is	pure	units
case	label	range	until
component	library	record	use
configuration	linkage	register	variable
constant	loop	reject	wait
disconnect	map	rem	when
downto	mod	report	while
else	nand	return	with
elsif	new	rol	xnor
end	next	ror	xor

III) Les opérateurs du langage VHDL.

Tous les opérateurs existants en VHDL se trouvent dans la liste ci-dessous dans l'ordre décroissant de leur priorité: [18]

syntaxe VHDL	nom	classe	priorité	exemple
abs	valeur absolue	op. divers	1	abs A
not	non logique	op. divers	1	Not A
**	puissance	op. divers	1	A**B
*	multiplication	op. de multiplication	2	A*B
/	division	op. de multiplication	2	A/B
mod	modulo	op. de multiplication	2	A mod B
Rem	remainder	op. de multiplication	2	A rem B
+	plus	op. de signe	3	+ A
-	moins	op. de signe	3	- A
+	addition	op. d'addition	4	A + B
-	soustraction	op. d'addition	4	A - B
&	concaténation	op. d'addition	4	- D1&D0 - "vh"&"dl"
Sll	déc. log. gauche	op. décalages et rotation (VHDL93)	5	Vecteur sll 3
Srl	déc. log. droite	op. décalages et rotation (VHDL93)	5	Vecteur srl 1
Sla	déc. arith. gauche	op. décalages et rotation (VHDL93)	5	Nombre sla 2
Sra	déc. arith. droite	op. décalages et rotation (VHDL93)	5	Nombre sra 4
Rol	rotation gauche	op. décalages et rotation (VHDL93)	5	Valeur rol 2
ror	rotation droite	op. décalages et rotation (VHDL93)	5	Valeur ror 4
=	équivalent à	op. relationnels	6	when A=B
/=	différent de	op. relationnels	6	when A/=B
<	plus petit	op. relationnels	6	when A<B
<=	plus petit ou égal	op. relationnels	6	when A<=B
>	plus grand	op. relationnels	6	when A>B
>=	plus grand ou égal	op. relationnels	6	when A>=B
and	et	op. logiques	7	A and B
Or	ou	op. logiques	7	A or B
nand	non et	op. logiques	7	A nand B
nor	non ou	op. logiques	7	A nor B
xor	ou exclusif	op. logiques	7	A xor B
xnor	non ou exclusif	op. logiques (VHDL93)	7	A xnor B

IV) Exemple de programme en langage VHDL

Le programme suivant est un exemple en VHDL de la bascule RS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;
entity RS_ASYNC is
port (R,S :in std_logic;
      Q,NQ :out std_logic);
end RS_ASYNC;
architecture ARCH_RS_ASYNC of RS_ASYNC i
signal X :std_logic;
begin
  X <= '0' when R='1' and S='0'
  else '1' when R='0' and S='1'
  else  X  when R='0' and S='0'
  else '-';
  Q <= X;
  NQ<=not X ;
end ARCH_RS_ASYNC;
```

Programme de la bascule RS synchrone en langage VHDL :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;
entity RS_SYNC is
port (CLK,R,S,CLR,Pr :in std_logic;
      Q,NQ:out std_logic);
end RS_SYNC;
architecture ARCH_RS_SYNC of RS_SYNC is
signal X :std_logic;
begin
  process(CLK,CLR,Pr)
   begin
   if CLR='1' then X<='0'
   else if Pr='1' then X<='1'
   else if (CLK'event and CLK='1') then
     if R='1' and S='0' then X <= '0';
     else if R='0' and S='1' then X <= '1';
     else if R='0' and S='0' then X <= X;
     else X <= '-';
     end if;
  end if;
  end if;
  end if;
  end if;
  end if;
```

```
  end if;
  end process;
  Q <= X;
  NQ <= not X;
end ARCH_RS_SYNC;
```

Le programme en langage VHDL de cette bascule est le suivant :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;
entity JK_FF is
port (CLK,Pr,CLR,J,K :in std_logic;
      Q,NQ :out std_logic);
end JK_FF;
architecture ARCH_JK_FF of JK_FF is
signal X :std_logic;
begin
  process(CLK,PR,CLR)
   begin
   if CLR='1' then X<='0';
   else if PR='1' then X <= '1';
   else if (CLK'event and CLK='1') then
     if K='1' and J='0' then X <= '0';
     elsif K='0' and J='1' then X <= '1';
     elsif K='1' and J='1' then X <= not X;
     else X <= X;
     end if;
   end if;
   end if;
   end if;
   end if;
  end process;
  Q <= X;
  NQ <=not X;
end ARCH_JK_FF;
```

Programme en langage VHDL de la bascule D :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;
entity D_FF is
port (CLK,PR,CLR,D :in std_logic;
      Q,NQ:out std_logic);
end D_FF;
```

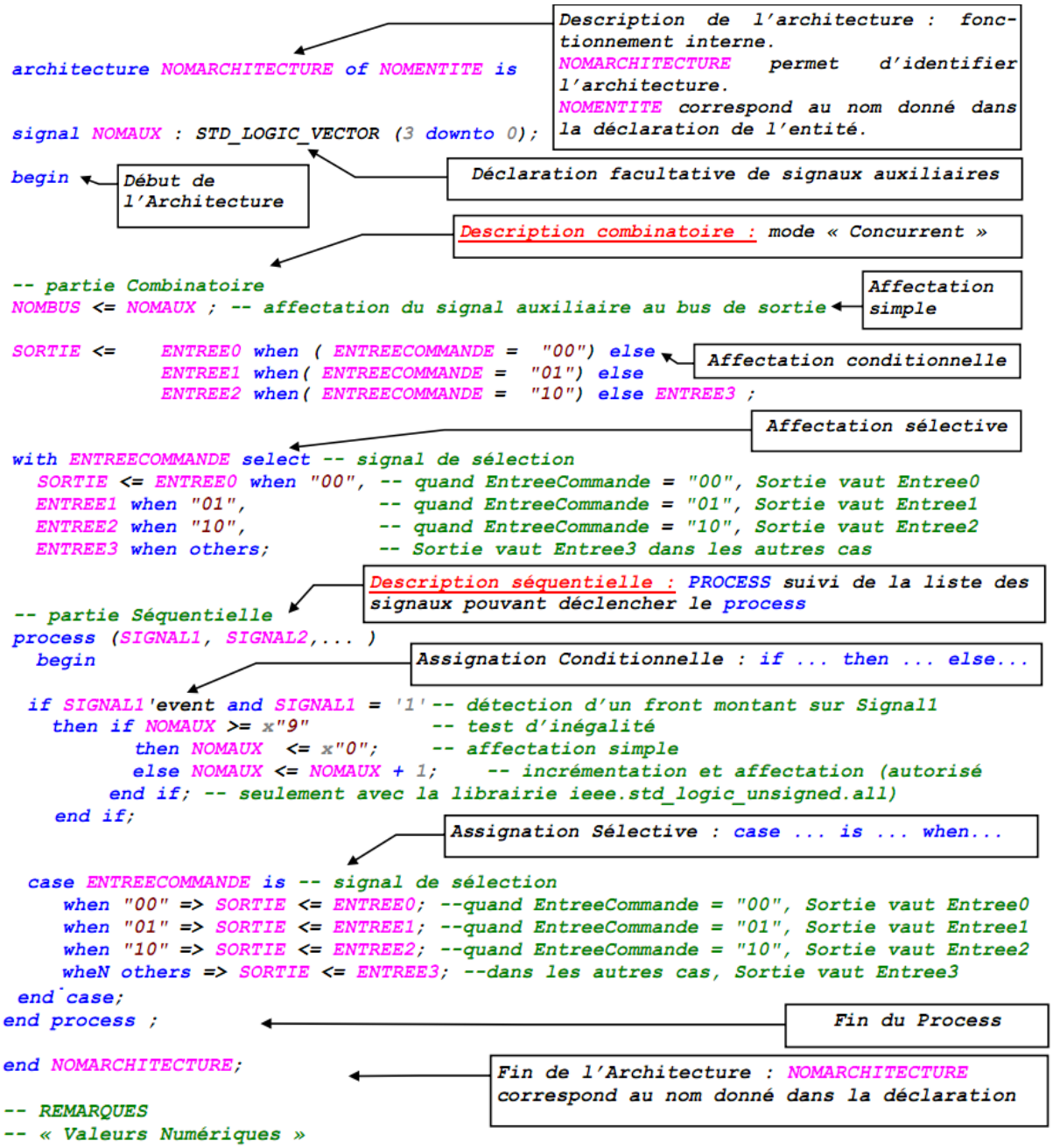
```
architecture ARCH_D_FF of D_FF is
signal X :std_logic;
begin
  process(CLK,PR,CLR)
   begin
   if CLR='1' then X<='0';
   else if PR='1' then X <= '1';
   else if (H'event and H='1') then X <= D;
   end if;
   end if;
   end if;
   end process;
  Q <= X;
  NQ <=not X;
end ARCH_D_FF;
```

Le programme de la bascule T en langage VHDL :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;
entity T_FF is
port (CLK,CLR,PR,T :in std_logic;
      Q,NQ :out std_logic);
end T_FF;
architecture ARCH_T_FF of T_FF is
signal X :std_logic;
begin
  process(CLK,PR,CLR)
   begin
   if PR='1' then X <= '1';
   else if CLR='1' then <='0';
   elsif (CLK'event and CLK='1') then
     if T='1' then X <= not X;
     else X <= X;
     end if;
     end if;
     end if;
     end if;
  end process;
  Q <= X;
  NQ <=not X;
end ARCH_T_FF;
```

V) Structure d'un programme en VHDL


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

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


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

 -- En plus des lignes précédentes, il est nécessaire de spécifier le brochage
 -- retenu pour le circuit utilisé.
 -- La syntaxe suivante est propre à OrCad Express pour un SPLD
 -- La définition du brochage doit respecter les possibilités du circuit :
 --       IN ≠ CLOCK    IO ≠ OUT


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


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


end NOMENTITE;
```




```
architecture NOMARCHITECTURE of NOMENTITE is

signal NOMAUX : STD_LOGIC_VECTOR (3 downto 0);

begin

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

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

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

-- partie Séquentielle
process (SIGNAL1, SIGNAL2,... )
  begin

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

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

end NOMARCHITECTURE;

-- REMARQUES
-- « Valeurs Numériques »
--  '1' : binaire pour un bit
--  "11" : binaire pour un bus
--  x"11" : heXadécimal pour un bus
--  11 : décimal (ne peut pas s'appliquer à tous les types de signaux)

-- « Tests »
-- = égal    /=  différent              <  inférieur                  >  supérieur
-- >=  supérieur ou égal                <=  inférieur ou égal
```

VI) Altera cyclone II FPGA

Le panneau DE1 comporte une puce puissante du cyclone ® II FPGA. Des composants de la plus haute importance sur le panneau sont branchés aux chevilles de cette puce, permettant à l'usager de configurer les connexions entre les divers composants comme

désirés. Pour des expériences simples, le panneau DE2 inclut un nombre suffisant des commutateurs, LEDs, et afficheur sept segments. Pour des expériences plus avancées, il y a SRAM, SDRAM, et une mémoire Flash. Pour les expériences qui exigent un compilateur et des simples interfaces d'E/S, il est facile d'instancier le compilateur de Nios II d'Altera et d'employer des normes d'interface telles que RS-232 et PS/2. Pour les expériences qui impliquent le son ou les signaux vidéo, il y a les connecteurs normaux fournis sur le panneau. Pour les grands projets, il est possible emploient la carte mémoire SD.

En conclusion, il est possible de brancher d'autres tableaux conçus pour l'utilisateur au panneau DE1 au moyen de deux en-têtes d'expansion. Le logiciel équipé de panneau DE1 comporte les outils de modèle d'édition web de Quartus II. Il inclut également un programme de moniteur simple qui permet à l'étudiant de contrôler de diverses parties du panneau d'une façon facilement compréhensible. Il y a également plusieurs applications qui expliquent l'installation du panneau DE1.

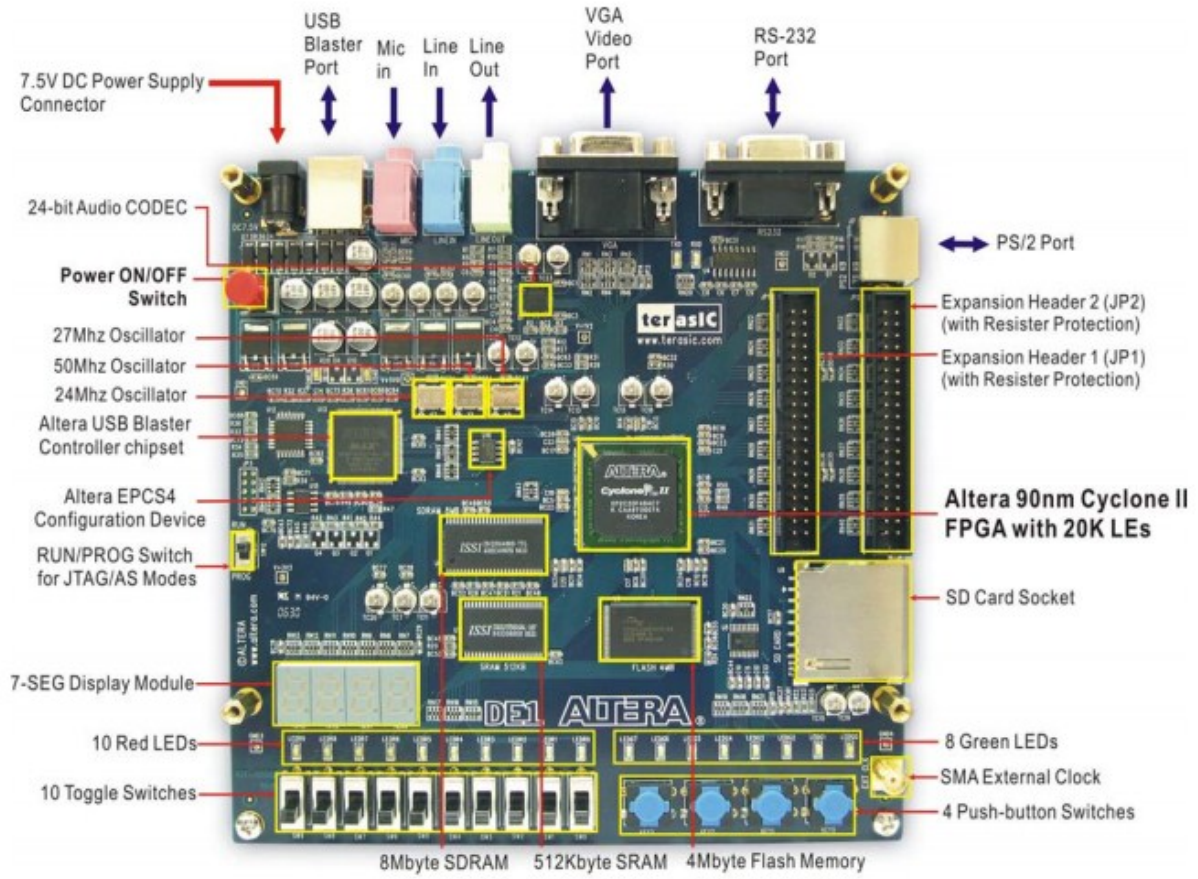


Figure (1) circuit FPGA Altera Cyclone® II 2C20.

Le panneau DE1 est expédié dans un module qui inclut toutes les pièces nécessaires pour son fonctionnement. Les parties essentielles sont l'adaptateur d'alimentation de 9 volts et le câble USB. Il y a également une couverture protectrice de plexiglass qui peut être utilisée

dans l'environnement de laboratoire pour protéger le panneau contre des dommages physiques accidentels.

Pour commencer, il faute brancher l'adaptateur de 9 volts. Employer le câble USB pour brancher le connecteur USB extrême gauche (celui le plus étroitement à l'interrupteur d'alimentation) sur le panneau DE1 à un port USB sur un ordinateur qui exécute le logiciel de Quartus II. Mettre en marche l'interrupteur d'alimentation sur le panneau DE1.

L'ordinateur identifiera le matériel branché à son port USB, mais il ne pourra pas l'effectuer si le pilote n'est pas installé. Le panneau DE1 est programmé en employant le mécanisme de l'USB-Sableuse d'Altera. Si le gestionnaire d'USB-Sableuse n'est pas déjà installé, New HardwareWizard sur le schéma suivant apparaîtra :

Figure (2) The new Hardware Wizard.

Le panneau offre une riche collection de caractéristiques qui le rendent approprié pour l'usage dans l'environnement des laboratoires pour des cours d'université et des applications, pour une série de projets de modèle, aussi bien que pour le développement des systèmes digitaux sophistiqués.

Avec le FPGA Cyclone® avancé, et son option mémoire flexible, et ces nombreuses unités d'E/S Avancées, le panneau DE1 est une plateforme idéale pour la mise en place de beaucoup de types de systèmes digitaux. Les professionnels de bureau d'études apprécieront la richesse des exemples de modèle équipés de panneau, et auront plaisir à expérimenter avec sonore, vidéo. Le panneau DE1 est le moyen idéal pour mettre en œuvre des applications embarquées de ce type qui comportent le processeur embarqué d'Altera Nios II.

Les spécifications de ce circuit sont :

- FPGA : FPGA Cyclone II EP2C20F484C7 et EPCS4 dispositif de configuration séquentielle.
- Unités d'E/S :
  - Interface d'USB intégré pour le port de la configuration RS-232 de FPGA.
  - VGA DAC resistor network (4096 colors).
  - Port PS/2 pour brancher une sourie ou un clavier.
  - Line-in, Line-out, microphone-in (24-bit audio CODEC).
  - Expansion headers (76 signal pins).
- Mémoire:
  - 8-MB SDRAM, 512-KB SRAM, 4-MB Flash.
  - Carte mémoire SD.
- Commutateurs, LEDs, étalages, et horloges :
  - 10 interrupteurs à levier.
  - 4 commutateurs de bouton poussoir.
  - Dix LEDs rouges et six LEDS vertes.
  - Quatre afficheurs sept segments.
  - oscillateurs 27 MHz et 50 MHz, entrée d'horloge externe de SMA.

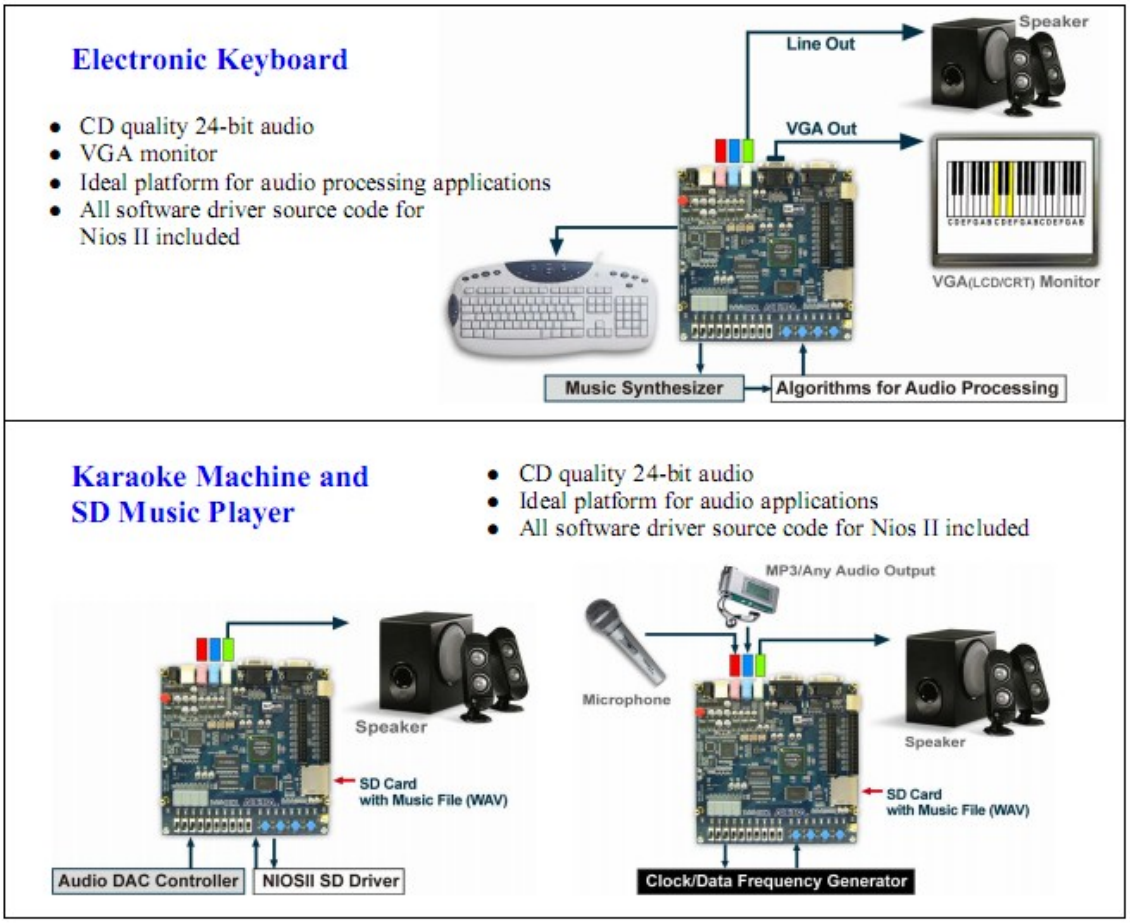


Figure (3) différent exemple d'utilisation de FPGA Cyclone II.

VII) QUARTUS II

Le système de Quartus II inclut le support complet pour toutes les méthodes populaires d'écriture d'une description du circuit désiré dans un système de DAO. Dans notre cas on s'intéresse à la méthode d'entrée de modèle en VHDL, dans laquelle l'usager spécifie le circuit désiré dans le langage de description de matériel VHDL. Mais aussi, il existe d'autres langages de description de matériel comme le Verilog et l'autre est basé sur la définition de circuit désiré sous forme de schéma.

La dernière opération dans le procédé de modèle implique est de configurer le circuit conçu dans un dispositif réel FPGA. Pour montrer comment ceci est fait, on le suppose que l'usager a accès à un Altera DE2 branché à un ordinateur qui a le logiciel de Quartus II installé. Un lecteur qui n'a pas accès au panneau DE2 trouvera toujours le cours d'instruction utile pour apprendre comment programmer le FPGA et comment la tâche de configuration est effectuée.

Le logiciel Quartus II rend facile de mettre en application un circuit logique désiré en utilisant un dispositif logique programmable, tel qu'un FPGA. Les étapes à suivre sont illustrée dans la figure suivante :

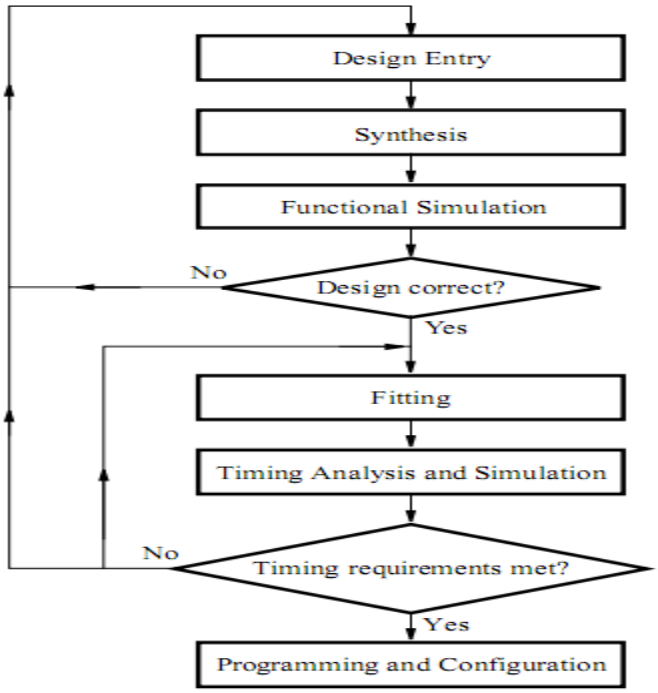


Figure (4) Les étapes de création d'un projet en VHDL.

- Entrée de modèle : le circuit désiré est spécifié au moyen d'un schéma de principe, ou en employant un langage de description de matériel, tel que VHDL.
- Synthèse : le modèle écrit est synthétisé dans un circuit qui comprend les éléments logiques fournis dans le FPGA.
- Simulation fonctionnelle : le circuit synthétisé est testé pour vérifier son exactitude fonctionnelle.
- Ajustement : l'outil de programmation détermine l'emplacement des éléments logiques défini dans la netlist dans les éléments logiques réels de FPGA.
- Analyse de calage : des délais de propagation le long des divers chemins dans le circuit adapté s'analysent pour fournir un signe du rendement prévu du circuit.
- Simulation de calage : le circuit adapté est testé pour vérifier son exactitude fonctionnel et calage.
- Programmation et configuration : le circuit conçu est mis en application dans un FPGA en programmant les commutateurs de configuration qui configurent les éléments logiques et établissent les connexions désirées de câblage.

Commencer avec Quartus II

Chaque circuit logique étant conçu avec le logiciel de Quartus II s'appelle un projet. Le logiciel travaille sur un projet à la fois et maintient toute l'information pour ce projet dans un répertoire simple dans le système de fichiers. Pour commencer un modèle neuf de circuit logique, la première opération est de produire un répertoire pour retenir ses fichiers.

L'étalage donné par la figure (5), se compose de plusieurs fenêtres qui permettent d'accéder à toutes les caractéristiques de logiciel de Quartus II. La plupart des commandes fournies par le logiciel de Quartus II peuvent être consultées en employant un ensemble de menus qui sont situés au-dessous de la barre de titre.

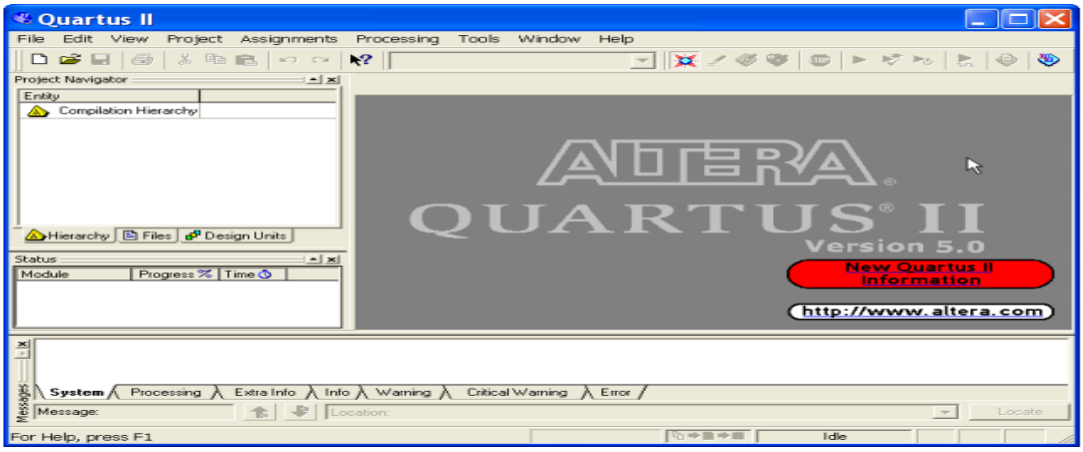


Figure (5) page de démarrage de Quartus II.

Pour commencer à travailler sur un modèle neuf, d'abord nous devons définir un projet de modèle neuf. Le logiciel de Quartus II rend la tâche de création facile. Pour produire un projet neuf on suit les étapes suivantes :

- Choisir *file > New Project Wizard* pour atteindre la fenêtre dans la figure (6). Vous pouvez sauter cette fenêtre dans des projets suivants en contrôlant le cadre *Don't show me this introduction again*. Choisir *Next* pour obtenir la fenêtre représentée sur la figure (7).

Figure (7) création de nouveau projet.

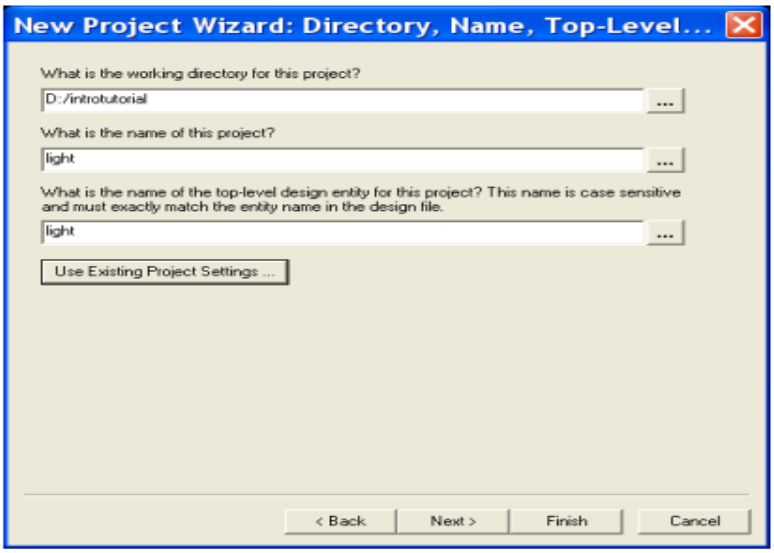


Figure (7) le nom de projet.

- Nous devons spécifier le type d'appareil dans lequel le circuit conçu sera mis en application. Choisir CycloneTM II comme famille de périphérique cible. Nous pouvons laisser le logiciel de Quartus II choisissons un appareil spécifique dans la famille, ou nous pouvons choisir le dispositif explicitement figure.

VIII) Programme de circuit de commande

Le programme de la bascule JK :

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity JK_ent is
        port(
                J : in STD_LOGIC;
                K : in STD_LOGIC;
                CLK : in STD_LOGIC;
                CLR : in STD_LOGIC;
                PR : in STD_LOGIC;
                Q : out STD_LOGIC;
                NQ : out STD_LOGIC
          );
end JK_ent;

architecture JK_arch of JK_ent is

signal etat:STD_LOGIC;
begin
        process(CLR,PR,CLK)
        begin
```

```
                if CLR ='1'
                        then etat<='0';
                else if  PR='1'
                        then etat<='1';
                else if CLR='1' and PR='1'
                        then etat<=etat;
                else if J='1' and K='0' and CLK='1' and CLK'event then
                        etat<='1';
                        else if J='0' and K='1' and CLK='1' and CLK'event then
                        etat<='0';
                else if J='1' and K='1' and CLK='1' and CLK'event then
                        etat<=not etat;
                end if;
                end if;
                end if;
                end if;
                end if;
                end if;
                end process;
                Q<=etat;
                NQ<=not etat;
end JK_arch;
```

Le programme de la bascule RS qu'on a utilisé dans ce travail est le suivant :

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity RS_ent is
       port(
             R : in STD_LOGIC;
             S : in STD_LOGIC;
             Q : out STD_LOGIC
           );
end RS_ent;
architecture RS_arch of RS_ent is
   signal etat1:STD_LOGIC;
begin
      process (R,S)
begin
    if R='1' and S='0'  then
            etat1<='0';
      else  if S='1' and R='0' then
            etat1<='1';
            else if S='0' and R='0' then
                  etat1<=etat1;
      else if R='1' and S='1' then
            etat1<=0;
      end if;
      end if;
      end if;
      end if;
      end process;
Q<=etat1;
end RS_arch;
```

Le programme de circuit de commande en langage VHDL est le suivant :

```
library IEEE;
use IEEE.std_logic_1164.all;
entity circuitcom is
  port(
      H : in STD_LOGIC;
      Mc : in STD_LOGIC;
      Md : in STD_LOGIC;
      Mz : in STD_LOGIC;
      Sc : out STD_LOGIC;
      Q : out STD_LOGIC_VECTOR(2 downto 0)
  );
end circuitcom;
architecture circuitcom of circuitcom is
component JK_ent
  port (
      CLK : in STD_LOGIC;
      CLR : in STD_LOGIC;
      J : in STD_LOGIC;
      K : in STD_LOGIC;
      PR : in STD_LOGIC;
      NQ : out STD_LOGIC;
      Q : out STD_LOGIC
  );
end component;
component RS_ent
  port (
      R : in STD_LOGIC;
      S : in STD_LOGIC;
      Q : out STD_LOGIC
  );
end component;
----    Constants    -----
constant DANGLING_INPUT_CONSTANT : STD_LOGIC := '1';
---- Signal declarations used on the diagram ----
signal Clk : STD_LOGIC;
signal ETNQ : STD_LOGIC;
signal Mzz : STD_LOGIC;
signal notSNQ : STD_LOGIC;
signal NQ1 : STD_LOGIC;
signal NQ2 : STD_LOGIC;
signal NQ3 : STD_LOGIC;
signal Q2 : STD_LOGIC;
signal Q3 : STD_LOGIC;
signal Q1 :STD_LOGIC;
signal SNQ : STD_LOGIC;
---- Declaration for Dangling input ----
signal Dangling_Input_Signal : STD_LOGIC;
begin
----  Component instantiations  ----
U1 : JK_ent
port map(
                    CLR => Mz,
                    CLK => Clk,
```

```
      J => Dangling_Input_Signal,
      K => Q3,
      NQ => NQ1,
      PR => notSNQ,
      Q => Q(0)
  );
notSNQ <= not(SNQ);
Clk <= SNQ and H and Md;
U2 : JK_ent
  port map(
      CLK => Clk,
      CLR => Mz,
      J => Q1,
      K => Q3,
      NQ => NQ2,
      PR => notSNQ,
      Q => Q(1)
  );
U3 : JK_ent
  port map(
      CLK => Clk,
      CLR => Mz,
      J => Q2,
      K => Dangling_Input_Signal,
      NQ => NQ3,
      PR => notSNQ,
      Q => Q(2)
  );
U4 : RS_ent
  port map(
      Q => Sc,
      R => Mz,
      S => Mzz
  );
Q3 <= not(NQ3);
Q2 <= not(NQ2);
Mzz <= Mc and ETNQ;
ETNQ <= NQ1 and NQ2 and NQ3;
SNQ <= NQ1 or NQ2 or NQ3;
Q1 <=not NQ1;
---- Dangling input signal assignment ----
Dangling_Input_Signal <= DANGLING_INPUT_CONSTANT;
end circuitcom;
```