UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences de l'Ingénieur

Département d'Electronique

MEMOIRE DE MAGISTER

En électronique

Spécialité : Contrôle

Simulateur dynamique des systèmes électromécanique à base d'FPGA

Par

TOUHAMI Rahim

Devant le jury composé de :

M.BOUNEKHLA	Professeur	USD-Blida	Président
A. NESBA	Maitre de conférences A	ENS-Kouba	Examinateur
A. FERDJOUNI	Maitre de conférences A	USD-Blida	Examinateur
A.GUESSOUM	Professeur	USD-Blida	Promoteur
M. MAAMOUN	Maitre de conférences B	USD-Blida	Invité

Blida, Mai 2013

Résumé

L'objectif du présent travail est d'analyser et de mettre en évidence la contribution des circuits reconfigurables en particulier les circuits FPGA dans la modélisation et l'accélération de la simulation des systèmes dynamiques. Pour mettre en évidence cette contribution, le modèle de la machine asynchrone est pris comme exemple. En effet, au cours de ce travail, toutes les étapes utiles pour la modélisation de la machine asynchrone, ainsi que l'implémentation de ce modèle sur la cible FPGA.

Mots clés : FPGA, Machine asynchrone

Abstract

The objective of this work is to analyze and highlight the contribution of reconfigurable FPGA especially to accelerate the modeling and simulation of dynamic systems, to highlight, this contribution it almost as an example model of induction motor, in did,, we will represent all steps taken to implement the model of induction motors on the circuit FPGA.

Keywords: FPGA**,** Asynchronous machine

مـــلـــخـــــص

الهدف من هذا العمل هو تحليل و إبراز مساهمة الدارات المعادة التشكيل بالخصوص AGPF لا سيما في تسريع نمذجة و محاكاة الأنظمة الديناميكية، لتسليط الضوء على هذه المساهمة، نعطي كمثال نموذج : "الآلة المتزامنة"، ومحتوى هذا العمل يمثل جميع الخطوات المتبعة التي اتخذت لتنفيذ نموذج الآلة المتزامنة على دارة FPGA

كلمات مفتاحية : الآلة المتزامنة, FPGA

REMERCIEMENTS

Je tiens tout d'abord à exprimer ma profonde reconnaissance et mes sincères remerciements à mes deux directeurs de mémoire, Pr A.Guessoum et Dr M.Maamoun, pour leurs grandes qualités d'encadrants et pour la confiance qu'ils m'ont témoignée.

Je tiens à témoigner toute ma gratitude à Mr. BOUKERBOUB Ahmed pour tous les conseils et l'aide précieuse qu'il m'a apportés.

J'adresse également des remerciements très particuliers aux membres du jury, qui m'ont fait l'honneur de participer au jugement de ce travail.

Je réserve une pensée toute particulière à mes chers parents qui m'ont toujours soutenu et guidé tout au long de mes études.

Je remercie également DOUADI Nour EL Houda, LOUGHIMA Sofiane, OUNISSI Oussama, ABDELLI Lotfi et YAHYAOUI Walid pour toute l'aide qu'ils mon apportée.

À toute personne qui a contribué - de près ou de loin - à la réalisation de ce travail, je dis merci.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

INTRODUCTION

Depuis plus d'un siècle, la simulation des systèmes électroniques aussi bien dans le domaine industriel que dans celui de la recherche scientifique constitue une étape indispensable dans le test et l'optimisation de tout circuit électronique complexe avant, pendant et après sa réalisation.

Au début du siècle dernier, pour effectuer une simulation, on représentait le modèle du système étudié par un ensemble d'éléments (résistances, condensateurs, inductances, etc.) assemblés pour modéliser les différents équipements. Cette méthode comportait beaucoup d'inconvénients, comme l'entretien permanant, le long changement de topologie et le résultat non précis. Au fil des années, avec l'invention du transistor, il ait été possible de développer des ordinateurs. De nos jours, l'avancement technologique, l'évolution des techniques de communication de données et l'énorme progrès des outils de calcul, ont permis d'effectué des simulations des systèmes, avec un temps de calcul proportionnel à la complexité du système simulé et inversement proportionnel à la longueur du pas de calcul ainsi qu'à la puissance de l'ordinateur utilisé.

Actuellement on peut distinguer deux types de simulation, la simulation logicielle « offline ou software » et la simulation matérielle « hardware ».

La simulation logicielle utilise des solutions programmables de type Processeur, où son traitement est séquentiel et relativement lent, sa programmation dépend du composant (DSP « Digital Signal Processor », Microprocesseur et Microcontrôleur…). La simulation matérielle utilise des solutions programmables du type Logique, où son traitement est parallèle en temps réel et une programmation architecturale qui ne dépend pas du composant (ASIC « Application Specific Integrated Circuits » et FPGA « Field Programmable Gate Array » …).

Problématique actuelle de la simulation des systèmes électromécaniques :

L'électromécanique est l'association des techniques de l'électricité et de la mécanique. Les applications des systèmes électromécaniques sont très nombreuses, on peut citer l'industrie, les produits électroménagers et l'automobile etc.

Une machine électrique est un dispositif électromécanique permettant la conversion d'énergie électrique en travail ou énergie mécanique. Ce processus est réversible et peut servir à produire de l'électricité. Les machines électriques produisant une énergie mécanique à partir d'une énergie électrique sont appelées des moteurs. On peut distinguer plusieurs types de moteurs, parmi eux, le moteur asynchrone.

La robustesse, le faible coût, les performances et la facilité d'entretient font l'intérêt du moteur asynchrone (MAS) dans de nombreuses applications industrielles. L'absence du découplage naturelle entre le stator et le rotor donne au moteur asynchrone un modèle dynamique non linéaire qui est à l'opposé de la simplicité de sa structure, et de ce fait on aura la difficulté de sa commande.

La technologie moderne des systèmes d'entraînement exige de plus en plus un contrôle continu et précis de la vitesse du couple, tout en garantissant la stabilité, la rapidité et le rendement le plus élevé possible. Afin de répondre à ces critères de performances toujours croissants, des algorithmes de commande de plus en plus complexes ont été développés.

L'approche classique de programmation séquentielle qui est une solution logicielle est considérée insuffisante pour la commande en temps réel car les exigences de temps d'exécution ne cessent d'augmenter. En effet, dans la phase de simulation des algorithmes de commande, les délais de temps de calcul détériorent les performances de contrôle en termes de rapidité de correction. Par exemple, le contrôle comporte un nombre élevé d'opérations mathématiques avec une grande précision (un grand nombre de bits pour la partie entière et fractionnaire des opérandes), ce qui le rend trop lent (plusieurs jours) à simuler par des logiciels analogiques comme Matlab/Simulink.

Les nouvelles solutions numériques matérielles telles que les FPGA peuvent être aussi considérées comme des solutions appropriées afin d'améliorer les performances de contrôle. Le parallélisme inhérent de ces nouvelles solutions numériques, ainsi que leurs grandes capacités de calcul font que les délais de temps de calcul sont négligeables en dépit de la complexité des algorithmes à implémenter. Par ailleurs, par rapport aux solutions logicielles, les solutions matérielles offrent au concepteur un accès à la partie architecture matérielle.

Le présent travail fixe comme objectif d'analyser et de mettre en évidence la contribution des circuits reconfigurables en particulier les circuits FPGA dans l'accélération de simulation des systèmes dynamiques comparée aux simulateurs "offline" ; comme (MATLAB/SIMULINK, SPICE, etc.….) ; ainsi que le développement d'un simulateur dynamique à temps réel, en implémentant le modèle de la machine asynchrone sur un circuit FPGA. Ce mémoire comporte quatre chapitres.

Le premier chapitre est consacré à la présentation des circuits programmables en particulier les circuits FPGA. Ainsi qu'une introduction au langage de description matériel (HDL « Hardware Description Language ») que l'on utilise pour l'implémentation du modèle de la machine asynchrone sur le circuit FPGA.

Le deuxième chapitre est consacré à l'étude de la machine asynchrone et le sa modélisation sous forme d'un système d'équations différentiel.

Le troisième chapitre est consacré à la présentation des différentes étapes de la programmation et la conception du modèle du la MAS avec les logiciels MATLAB et XILINX. Ainsi qu'une présentation des différentes étapes de la simulation et résultats obtenus avec les logiciels MATLAB, MODELSIM et SYSTEM GENERATOR.

Enfin, nous terminons par une conclusion générale sur l'ensemble de cette étude.

CHAPITRE 1
FPGA ET CALCUL NUMERIQUE

1.1.Introduction :

Les composants programmables, apparus début 70, ont dû attendre le milieu des années 80 pour que la technologie permette l'intégration d'architectures complexes et performantes. Ils permettent de réaliser des fonctions logiques combinatoires (décodage d'adresses, par exemple) ou séquentielles (machines d'état, filtres numériques, ...) très rapides. Exploitant les technologies les plus performantes du moment, leur architecture doit sans cesse évoluer pour offrir les meilleures performances, en termes de vitesse, de capacité et d'utilisation.

Les circuits FPGA (Field Programmable Gate Array, ou ce qui se traduit par « réseaux de portes programmables sur site ») sont certainement les circuits reprogrammables ayant le plus de réussite. Ce sont des circuits entièrement configurables par programmation et qui permettent d'implémenter les fonctions logiques. De plus, ils ne sont pas limités à un mode de traitement séquentiel de l'information tels que les microprocesseurs, et en cas d'erreur, ils sont reprogrammables électriquement sans avoir à extraire le composant de son environnement.

Ross Freeman, Bernie Vonderschmitt et Jim Barnett, fondateurs de la compagnie XILINX (1984) introduisent en 1985 sur le marché le premier FPGA, le XC2064, et offrent ainsi une alternative aux précédentes approches.

1.2.Les circuits programmables:

Un circuit programmable est un assemblage d’opérateurs logiques combinatoires et de bascules dont la fonction réalisée n’est pas fixée lors de sa fabrication. Il contient potentiellement la possibilité de réaliser toute une classe de

fonctions plus ou moins large selon son architecture. On peut les programmer électriquement une seule fois (fusibles), ou plusieurs fois (reconfigurables).

La plupart des circuits programmable sont issus de la célèbre architecture **Von-Neumann** proposée par John Von Neumann, ou l'architecture **Harvard** qui porte le nom de l'université américaine qui la proposée [13].

On retrouve souvent l'architecture Von-Neumann dans les microprocesseurs et les microcontrôleurs (Motorola 68XXX, Intel 80X86) du fait qu'elle est très souple pour la programmation. Dans cette architecture, le programme et les données sont enregistrés sur la même mémoire. Chaque instruction contient la commande de l'opération à effectuer et l'adresse de la donnée à utiliser, donc pour exécuter une instruction il faut souvent plusieurs cycles d'horloge. L'architecture Harvard est utilisée généralement dans des microprocesseurs du type DSP (Digital Signal Processor). Elle nécessite deux fois plus de bus de données, d'adresses, et donc plus de broches sur la puce.

Le besoin croissant de composants très rapides a orienté les chercheurs à développer une autre solution différente des deux précédentes architectures. Cette solution réside dans le mode de programmation qui est devenu architecturale à logique câblée.

Ces circuits se distinguent par deux solutions de programmation :

- logicielle:

  Elle est nommée aussi solution programmable de type Processeur, où son traitement est séquentiel et sa programmation dépend du composant (DSP, Microprocesseur et Microcontrôleur…).

- matérielle:

  Elle est nommée aussi solution programmable du type logique, où son traitement est parallèle en temps réel et une programmation architecturale qui ne dépend pas du composant (ASIC et FPGA…).

La Figure 1.1 ci-dessous nous permet de bien voir la différence entre la solution logicielle qui effectue un traitement séquentiel et la solution matérielle qui effectue un traitement parallèle.

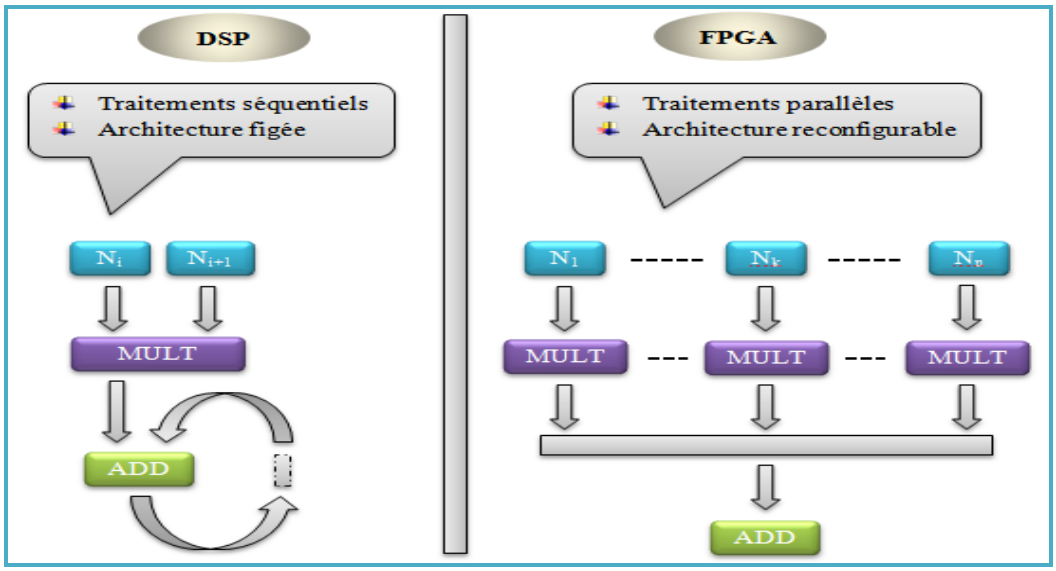

Figure 1.1 : Schéma comparatif d'un DSP et d'un FPGA.

On constate que pour multiplier n valeurs puis les additionnées avec un DSP on utilise un multiplieur et un additionneur (voir Figure 1.1 coter gauche), ceci va prendre plusieurs top d'horloge et rend la tâche longue. Avec un FPGA le travail se fait en deux top d'horloge, le premier top on multiplie les n valeurs, le deuxième top d'horloge on les additionne (voir Figure 1.1 coter droit).

Les caractéristiques de ces circuits sont récapitulées dans le tableau suivant [12]:

	ASIC	FPGA	DSP
Performances	Très élevées	Elevées	Faibles
Taille	Faible	Moyenne	Elevées
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	Très élevé	Moyen	Faible

Tableau 1.1 : Comparaison des différentes solutions numériques.

Avant de détailler les circuits logiques programmables, on doit bien distinguer entre les différents types de mémoires.

1.2.1.Les mémoires :

Une mémoire est un ensemble d'éléments mémoires binaires (ou cases mémoires). On distingue deux types de mémoires, les mémoires mortes ou non volatiles (l'information écrite à l'intérieur de chacune des cases mémoire n'est pas affectée lors de l'ouverture du circuit d'alimentation), c'est le cas des ROM, PROM, EPROM et EEPROM, les mémoires vives ou volatiles, c'est le cas des RAM.

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

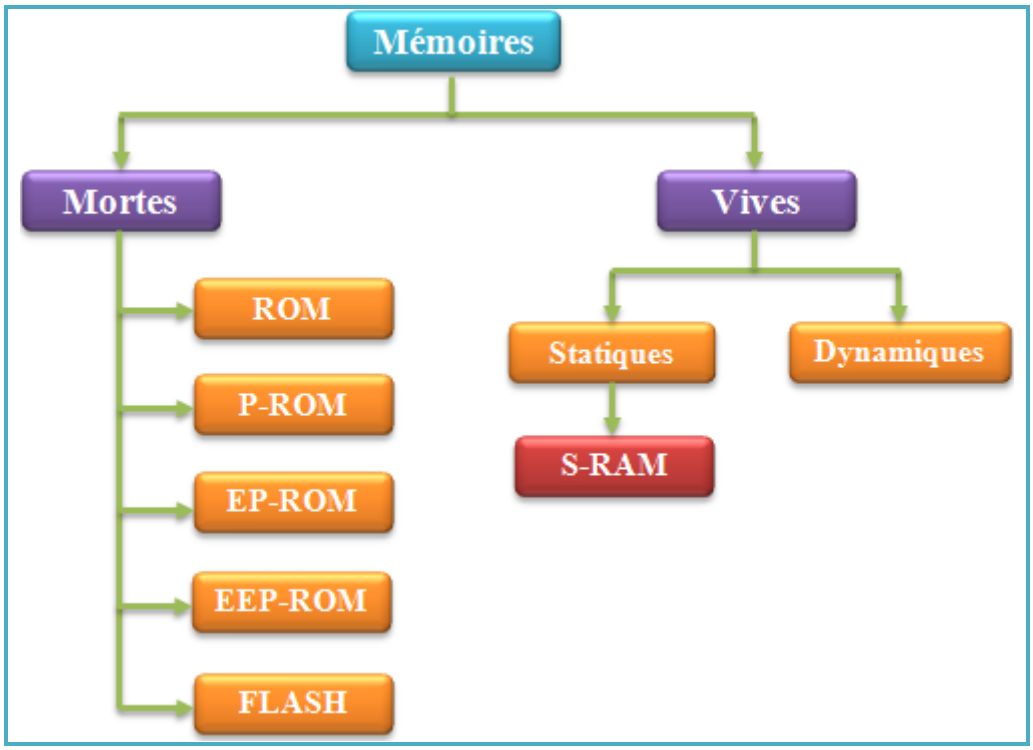

Figure 1.2 : Les différents types de mémoires.

L'ensemble des caractéristiques de ces mémoires sont récapitulées comme suit [12,17] :

- 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 de quelques minutes. On les appelle aussi des UVP-ROM.

- Les EEP-ROM (Electrically Erasable Programmable Read Only Memory): Mémoires programmables électriquement à lecture seule, effaçables électriquement, elles sont plus faciles à effacer que les EPROM**.**

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

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

- Les RAM dynamiques: Mémoires volatiles qui nécessitent rafraichissement périodique de l'information afin de la conserver.

1.2.2.Les circuits logiques programmables :

Les circuits logiques programmables et reprogrammables architecturalement sont classifiés en trois grandes familles les SPLD, CPLD, FPGA. L'arborescence suivante illustre les différents types suivant la technologie utilisée [16, 17, 14].

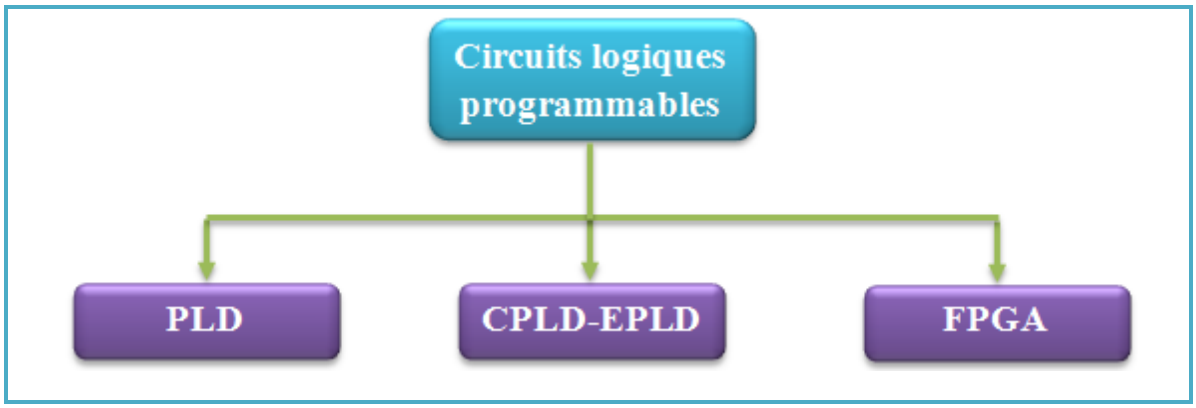

Figure 1.3 : Diagramme des différents types de circuits logiques programmables.

1.2.2.1.Les circuits SPLD (Simple Programmable Logic Device):

Ils sont composés d'une grille de portes ET et d'une grille de portes OU, ainsi qu'un bloc d'entrée et un bloc de sortie. Dans ces circuits, les connexions sont préexistantes, les différentes lignes étant reliées par des fusibles ou des transistors. En brûlant certains de ces fusibles, ou en programmant les transistors, il est toujours possible de réaliser différentes fonctions logiques.

Dans cette famille, on distingue deux types de circuits:

- PAL (Programmable Array Logic): Circuits logiques programmables dans lesquels seules les fonctions ET sont programmables, les fonctions OU ne le sont pas.

- GAL (Generic Array Logic): Circuits logiques PAL reprogrammables à technologie CMOS.

1.2.2.2.Les CPLD ou EPLD (Complex ou Erasable Programmable Logic Device):

Ces circuits sont le résultat de l'évolution des PLD, ils permettent l'implémentation de systèmes nettement plus complexes, composés d'éléments de base programmables du type SPLD, connectés entre eux par un réseau d'interconnexions relativement simple [14].

Dans cette famille, on distingue deux types de circuits :

- Circuits à EPROM : ce sont des circuits à base de mémoires EPROM.
- Circuits à EEPROM : ce sont des circuits à base de mémoires EEPROM.

1.2.2.3.Les ASIC :

Un ASIC (Application-Specific Integrated Circuit) est un circuit intégré à application spécifique, on distingue deux catégories :

Circuit semi-spécifique (certaines étapes de fabrication ont été faites avant la conception) et le circuit spécifique (toutes les étapes de fabrication seront réalisées après la conception).

1.3.Les circuits logiques programmables du type FPGA :

L'évolution des CPLD a donné naissance aux circuits FPGA, récemment, ils intègrent également des mémoires entières, des multiplieurs et même des noyaux de processeurs.

Les FPGA ou encore réseau de cellules logique programmables sont des circuits assimilables aux ASIC, à la déférence qu'ils sont programmables par l'utilisateur. Ces circuits sont des ensembles de blocs logiques élémentaires (plus d'un million de portes) que l'utilisateur peut interconnecter afin de réaliser les fonctions logiques de son choix.

1.3.1.Architecture d'un circuit FPGA :

Le circuit FPGA est un réseau de blocs logiques programmables **(CLB)** organisées dans une structure matricielle et de blocs d'entrée/sortie **(IOB)** placés sur la périphérie du circuit FPGA**,** le tout est interconnectées par un réseau d'interconnections programmables. Le modèle de FPGA que nous utilisons comme cible est un Virtex5 **XC5VLX50 -3**, produit par Xilinx.

La figure suivante simplifie la représentation de l'architecture interne d'un FPGA [3, 19, 20].

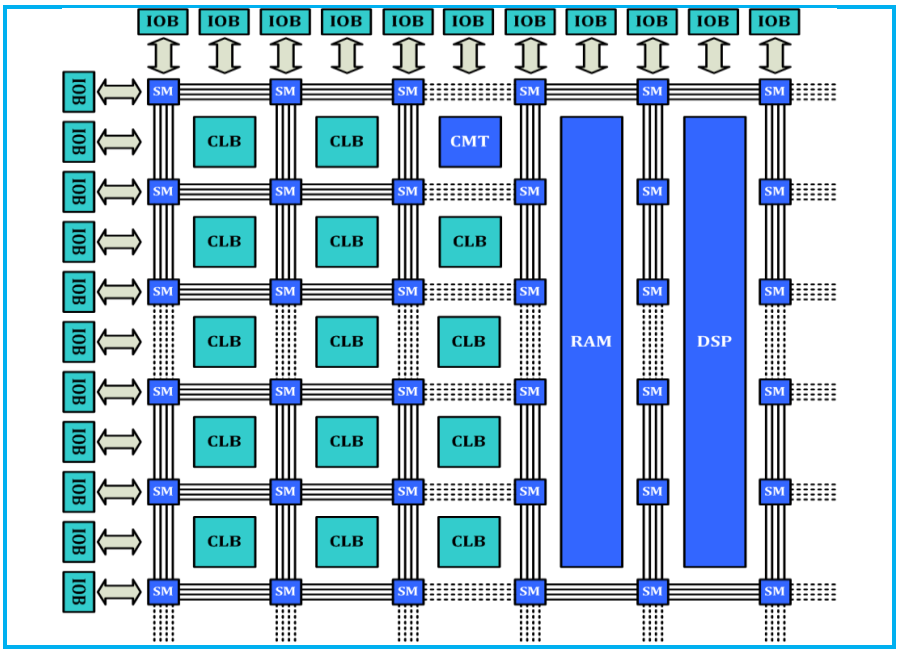

Figure 1.4 : Architecture interne d'un FPGA

Le composant Virtex-5 **XC5VLX50 -3 FFG 676 C** contient une matrice de 120X30 **CLB** et 560 **IOB**. Ce FPGA contient aussi plusieurs blocs dédiés pour différentes utilisations, on site : 48 blocs **RAM** de 36kb de type FIFO, 06 blocs de gestion d'horloge **(CMT)**, et 48 blocs de traitement du signal **(DSP48E)** [24], [23]**.**

1.3.1.1.Bloc d'Entrées/Sorties (IOB) :

Ces blocs entrée/sortie (***Input/Output Blocks***) permettent l'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé (haute impédance).

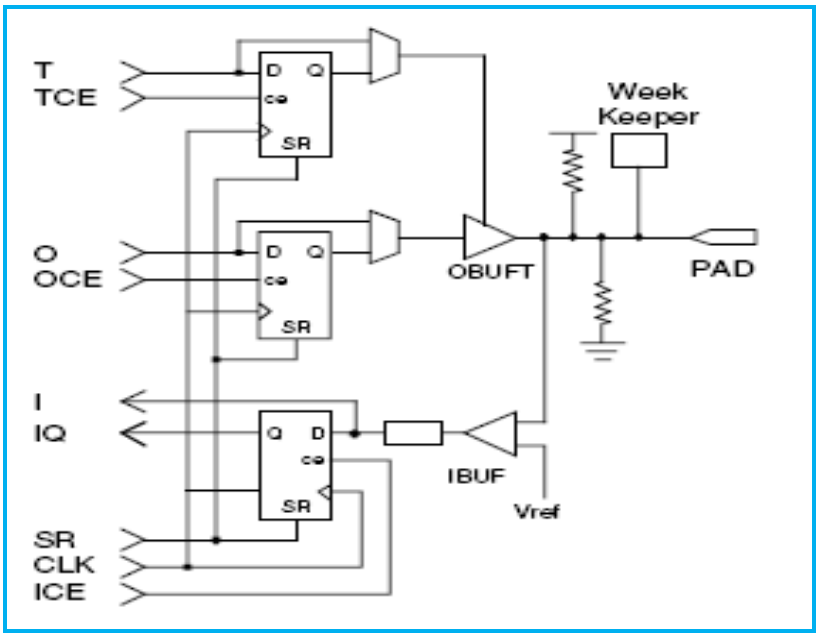

Figure 1.5 : Diagramme de bloc IOB de base

1.3.1.2.Bloc Logique Configurable (CLB) :

Le bloc logique configurable (***Configurable Logic Blocks***) est l'élément de base d'un composant FPGA, il est constitué de deux ***slices*** sans connections directes entre eux.

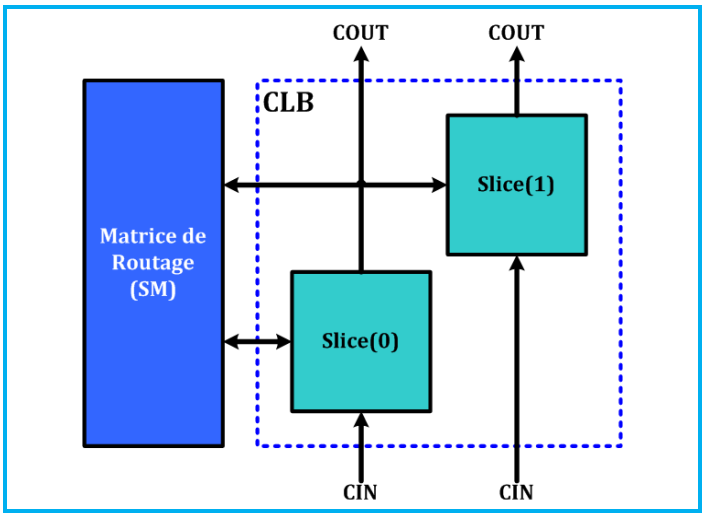

Figure 1.6 : Architecture d'un CLB

Un ***slices*** est de manière générale constitué de quatre tables de correspondance **LUT** (***Look-Up-Table***), de quatre bascules D et de quelques multiplexeurs. Elles sont utilisées pour réaliser des fonctions logiques, arithmétiques, décalages et ROM. Elles peuvent être considérées aussi comme une RAM distribuée.

La LUT est une petite mémoire sert à implémenter des équations logiques de 6 entrées et une sorties. Elle peut toutefois être considérée comme, un multiplexeur ou un registre à décalage. Le registre permet de mémoriser un état (machine séquentielle) ou de synchroniser un signal (pipeline) [25].

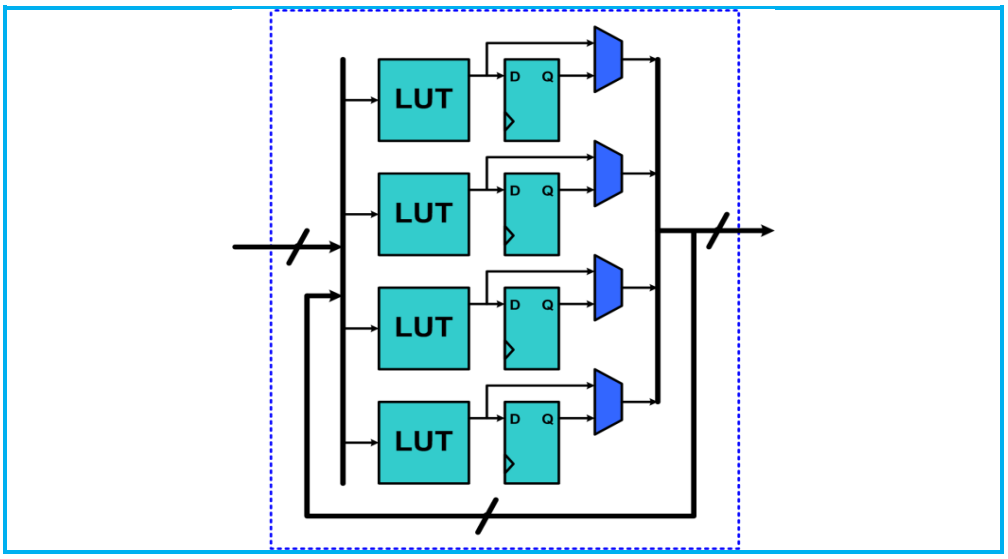

Figure 1.7 : Représentation simplifié d'un slice de Virtex5

1.3.1.3. Bloc de mémoire vive (RAM Bloc) :

Un FPGA de famille Virtex5 contient 48 **RAM** bloc ***(Random Access Memory)*** de 36kb organisées en colonnes. Chacune de ces RAM est constituée de deux blocs de 18 kb contrôlé indépendamment. Ces blocs peuvent être utilisés séparément, comme ils peuvent être utilisés en cascade pour former des RAM de plus grande taille, des ROM ou des mémoires FIFO synchrones ou asynchrones.

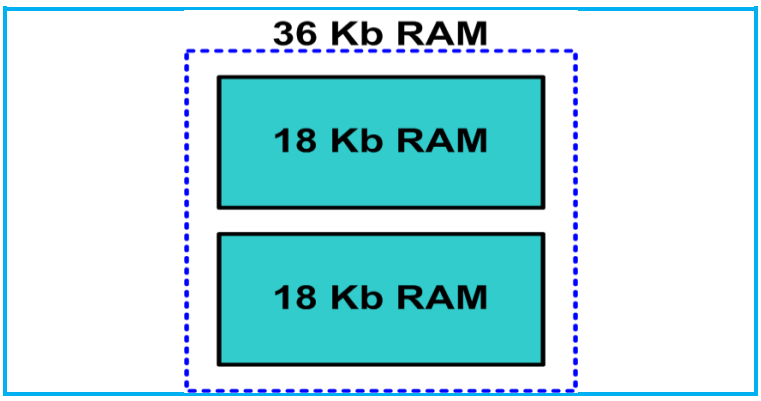

Figure 1.8 : Représentation d'une RAM bloc

1.3.1.4. Cellule de gestion d'Horloges (CMT) :

Les blocs **CMT** (***Clock Management Tile***) dans la famille Virtex5 assurent une gestion souple et très performante des horloges dans un circuit FPGA. Chaque CMT contient deux blocs **DCM** et un bloc **PLL**.

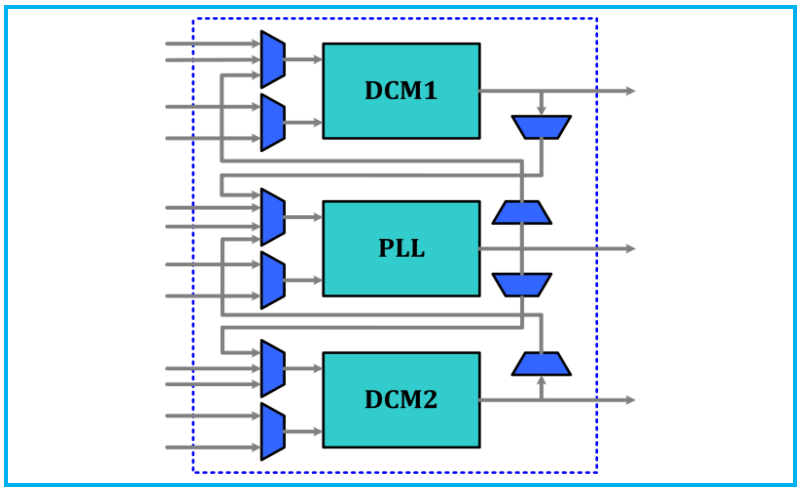

Figure 1.9 : Diagramme de bloc CMT de Virtex5

❖ Gestionnaire d'Horloge Digitale (DCM) :

Les blocs DCM (***Digital Clock Manager***) des FPGA sont utilisés pour générer des signaux d'horloge internes au FPGA parfaitement en phase avec les horloges sources venant de l'extérieur ou bien déphasé de 90°, 180°, ou 270° de la version originale.

Les DCM contiennent des unités **DLL** (***delay-locked loop***) qui éliminent les délais de distribution du signal d'horloge dans les endroits éloignés de la source. Ils sont également capables de générer un grand nombre de fréquences d'horloge différentes, à travers un ensemble de diviseurs/multiplicateurs de fréquence.

❖ Boucle à Verrouillage de Phase (PLL) :

Le **PLL** (***Phase-Locked Loop***) est utilisé principalement comme un synthétiseur pour une large bande de fréquences, il se servit aussi avec le DCM pour filtrer tous les signaux d'horloge internes et externes [26].

1.3.1.5. Bloc de traitement de signal (DSP48E) :

La Virtex5 contient une deuxième génération de blocs DSP c'est la **DSP48E**. Ces blocs peuvent réaliser plusieurs fonctions mathématiques indépendantes pour le traitement des signaux comme la multiplication, multiplication addition, multiplication accumulation …
L'architecture simplifiée d'un bloc DSP48E peut être représentée par la figure suivante :

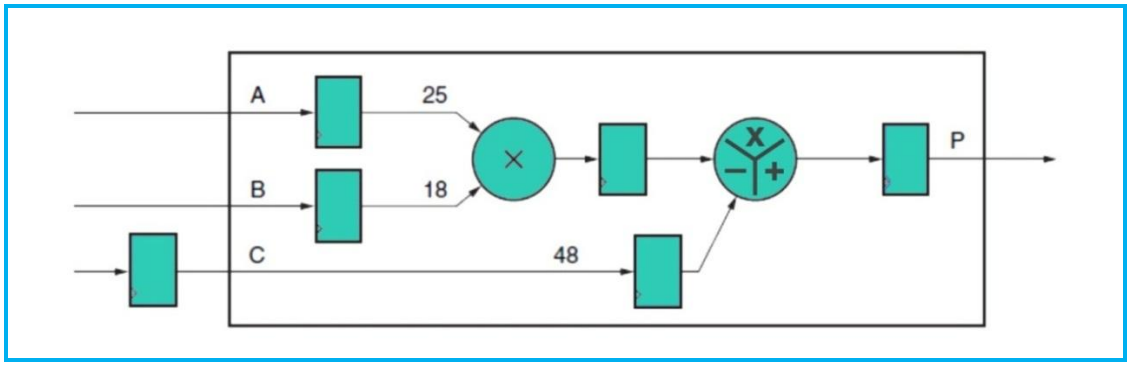

Figure 1.10 : Diagramme de bloc simplifié d'une cellule DSP48E de Virtex5

1.3.1.6. Réseau d'interconnexions :

Le réseau d'interconnexions permet la réalisation des connexions (le routage) entre les différentes tranches au sein d'un même CLB, entre les différents CLBs, les CMTs, les multiplieurs, les DSP48E, la mémoire RAM et les IOBs utilisés pour la construction d'un circuit donné.

Au sein d'un même CLB, les différentes tranches peuvent communiquer via des connexions rapides. Une matrice de routage (***SM***) est associée à chaque CLB, IOB, CMT, à la mémoire RAM, au multiplieur et aux blocs DSP48E.

Le rôle de cette matrice est de réaliser une connexion programmée entre les conducteurs "entrants" et "sortants". La programmation de la connexion est établie à l'aide d'un mot de mémoire défini par la configuration. Grâce aux ***SM***s toute ressource peut donc accéder à toute autre ressource, quel que soit l'endroit où elle se trouve dans le circuit FPGA.

Les conducteurs "entrants" et "sortants" d'une matrice SM sont disposés entre les lignes et les colonnes des CLBs.

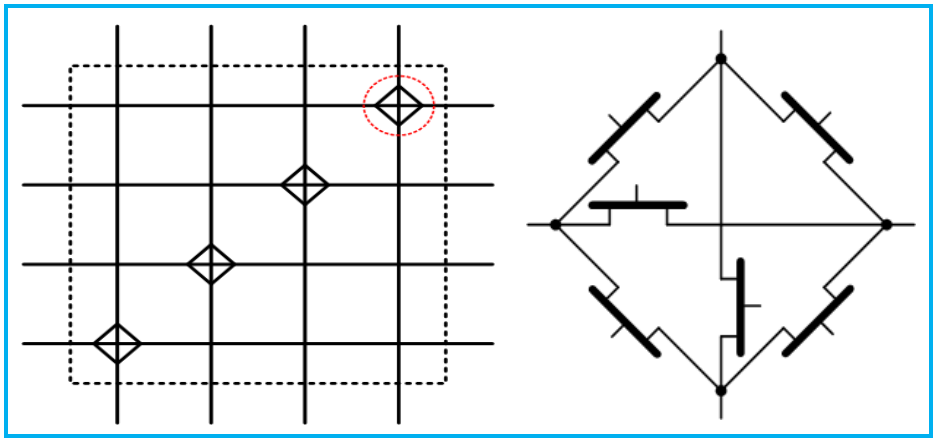

Figure 1.11 : Matrice de routage (SM)

1.3.2.La configuration des FPGA :

Les techniques de conception CAO (**C**onception **A**ssistée par **O**rdinateur**)** sont aujourd'hui très éprouvées et largement employées afin de concevoir des circuits électroniques nécessaires à mettre en pratique les connaissances algorithmiques. L'approche moderne pour la conception des circuits (logiques) électroniques et la manière d'introduire une fonctionnalité sur un support physique sont confiées aux outils CAO [12, 15, 17]**.**

Dans le cas des FPGA, Les outils CAO sont utilisés pour générer un fichier de configuration (bit-Stream) à partir d'une description de haut niveau conçu à l'aide d'un éditeur schématique ou d'un outil de traitement de texte. Les principaux rôles confiés aux outils CAO sont :

- La description,
- La simulation,
- La synthèse,
- Le placement et le routage.

L'un de ces outils est nommé XILINX ISE, il est dédié à la CAO des circuits numériques accompagné de son simulateur intégré ISE SIMULATOR. Le développement d'une application sur FPGA par XILINX ISE suit l'enchainement des étapes suivantes:

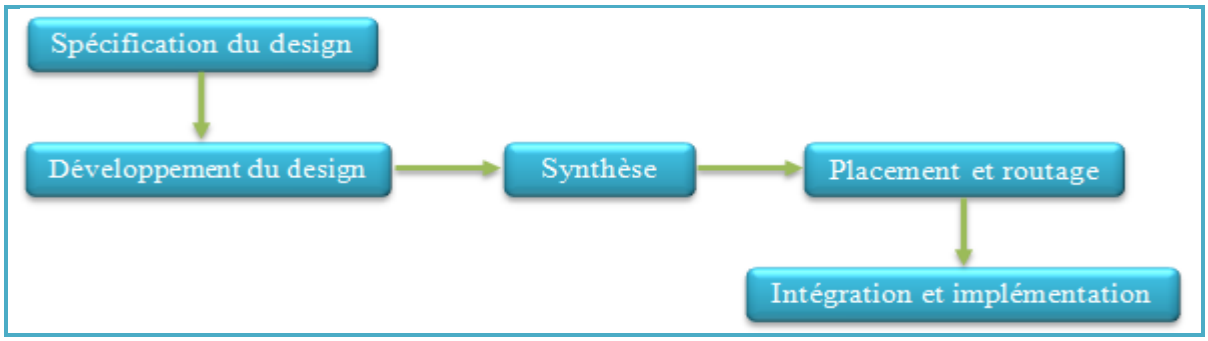

Figure 1.12 : Etapes de conception sur FPGA.

1.3.2.1.Spécification du design :

La spécification du design consiste à définir :

- Le nombre de broches d'entrée-sortie et leur localisation dans la puce FPGA.
- La spécification de la fréquence d'horloge du système.
- La spécification de la mémoire requise pour l'application.

1.3.2.2.Développement du design :

La spécification du design consiste à définir :

- La saisie de la description RTL « Register Transfer Level » (VHDL, Verilog).

- La simulation (Pré et Post synthèse) : La simulation permet de vérifier le comportement d'une description avant ou après implémentation dans le composant cible. Elle représente une étape essentielle qui nous fera gagner du temps lors de la mise au point sur la carte. Il faut juste noter qu'un projet peut être simulé même s'il n'est pas synthétisable. On distingue trois types de simulation qui sont utilisées à différentes étapes :
  - Simulation comportementale : Lors de l'étape de simulation comportementale, on valide l'application indépendamment de l'architecture et des temps de propagation du futur circuit.
  - Simulation fonctionnelle : La phase de simulation après synthèse valide l'application sur l'architecture du circuit cible. Elle ne tient pas compte des capacités de liaison dues au routage entre les différentes cellules. Elle permet donc de vérifier uniquement la validité du circuit par rapport au cahier des charges du point de vue fonctionnel et non du point de vue temporel.
  - Simulation temporelle : Il s'agit de vérifier la fonctionnalité du circuit en prenant en compte par un calcul estimatif les longueurs d'interconnexion et les retards apportés par les capacités parasites liées au routage. Elle vérifie donc que la fonctionnalité n'a pas été modifiée par l'introduction des délais de propagation et reste conforme au cahier des charges.

1.3.2.3.Synthèse :

La synthèse est le processus qui convertit la représentation du design à partir du code HDL fourni pour produire une représentation au niveau porte logique. Elle s'occupe de déterminer quelles sont les structures susceptibles de répondre au cahier des charges étudié et de produire un code booléen unique sous forme d'un fichier.

1.3.2.4.Placement et routage :

A partir des fichiers de synthèse, l'outil de conception procède au placement et routage. Un algorithme de routage est sensé de faire l'aiguillage des données qu'il reçoit vers leurs destination par action sur les nœuds de routage ce

qui est équivalent à définir les chemins qui relient l'ensemble des CLB contenus dans la fonction désirée. Ces algorithmes de routage sont différents d'un concepteur à un autre. Plusieurs traitements sont nécessaires pour obtenir un fichier de configuration :

- **Partitionnement** : Les équations logiques sont partitionnées en un autre ensemble équivalent d'équations. Chaque équation de ce nouvel ensemble peut être implantée dans un seul bloc logique du composant cible FPGA.
- **Placement** : Des blocs logiques sont sélectionnés dans la matrice et affectés au calcul des nœuds du réseau booléen.
- **Routage** : Les ressources d'interconnexion sont affectées à la communication de l'état des nœuds du réseau vers les différents blocs logiques qui en ont besoin.
- **Génération des données numériques de configuration** : Les informations abstraites de routage, de placement et les équations implantées dans les blocs sont transformées en un ensemble de valeurs numériques, qui seront chargées sur le composant FPGA.

1.3.2.5.Implémentation :

L'implémentation est la réalisation proprement dite qui consiste à mettre en œuvre l'algorithme sur l'architecture du circuit configurable cible, c'est-à-dire à compiler, charger, puis lancer l'exécution sur un ordinateur ou calculateur. C'est une étape de programmation physique et de tests électriques qui clôture la réalisation du circuit.

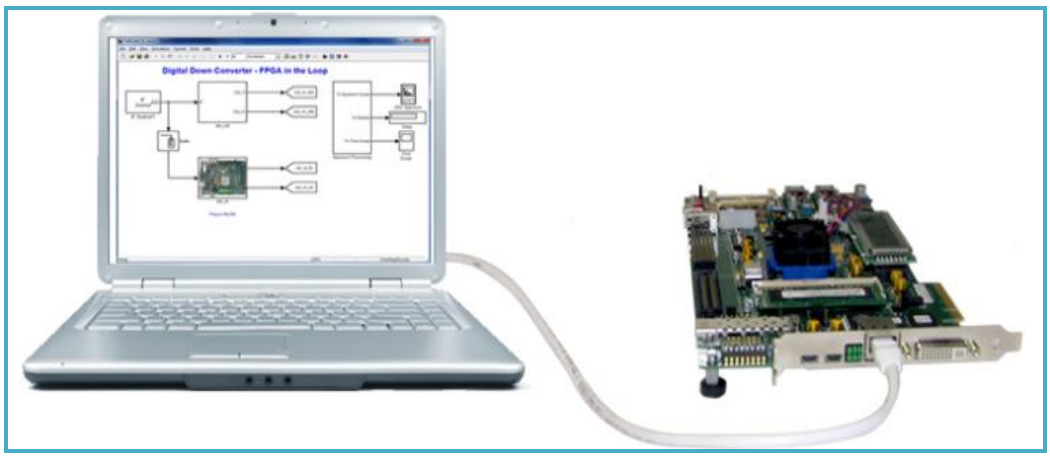

Figure 1.13 : envoie des données sur FPGA.

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.

1.3.3.Différents fondeurs et domaines d'application des FPGA :

Les fabricants des FPGA ne cessent pas d'améliorer leurs produits par l'efficacité et la puissance. L'ensemble des principaux fondeurs (Firmes) qui conçoivent ce type de circuits sont : Actel ,Altera , Atmel ,Cypress, Lattice, Minc , QuicLogic, Xilinx et d'autres [12].

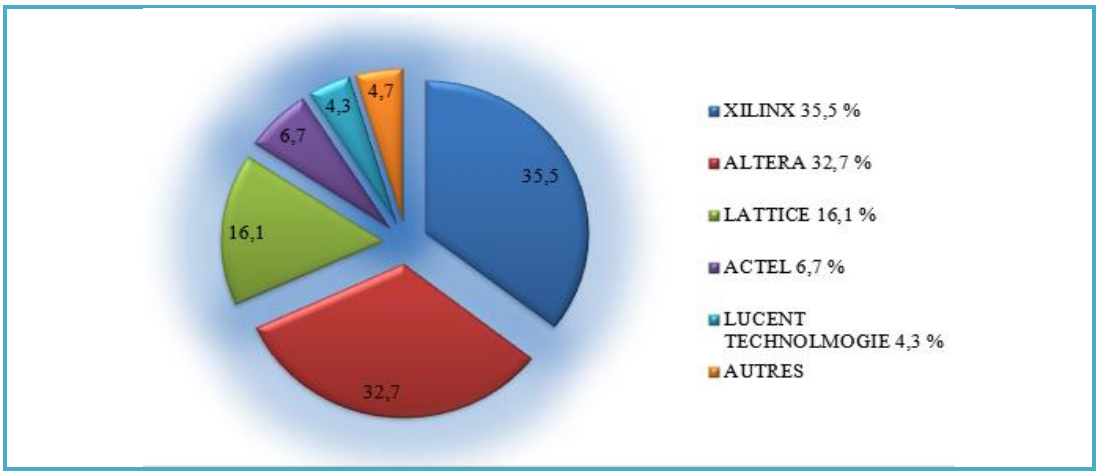

Figure 1.14 : Statistiques du marché occupé par les vendeurs d'FPGA.

Les circuits FPGA fabriqués par la société Xilinx occupent une grande partie du marché à cause de leur diversité ainsi que leurs performances. Ils sont divisés en deux gammes :

- FPGA hautes performances : gamme **Virtex** : Virtex, Virtex2, Virtex2Pro, Virtex4, Virtex5…
- FPGA pour la fabrication en grande série: gamme **Spartan**: Spartan, Spartan2, Spartan2E, Spartan3, Spartan3E, Spartan3A, Spartan3AN…

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**.** Leurs utilisations actuelles couvrent plusieurs domaines. 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.

1.4.Conclusion :

Dans ce chapitre nous avons présenté en premier temps les différents types de circuits programmables ainsi que leurs différentes architectures. Ensuite une description du circuit FPGA de Xilinx a été présentée. On a donné également les étapes principales de l'implémentation du circuit dans une cible FPGA, ainsi que les différents fondeurs et domaines d'application de ces circuits.

Ceci nous a permis de se familiarisé avec la technologie FPGA, qui s'inscrit au sommet de l'évolution des composants logiques.

CHAPITRE 2
MODELISATION DE LA MACHINE ASYNCHRONE

2.1.Introduction :

Depuis son invention par *NICOLA TESLA*, le moteur asynchrone (MAS) ou moteur à induction a attiré une grande attention dans le monde de l'industrie, et devenant l'actionneur le plus important parmi les machines tournantes de nos jours.

Le moteur asynchrone est une machine à courant alternatif sans contact physique entre le stator et le rotor [2]. Le terme asynchrone résulte du fait que la vitesse du moteur n'est pas nécessairement proportionnelle à la fréquence des courants qui le traversent. Il est caractérisé par sa simplicité de conception, de fabrication, d'entretien et de robustesse. De plus, il est peu coûteux avec un excellent rendement [1]. Cette simplicité s'accompagne par une complexité de contrôle dû à son non linéarité et à son caractère fortement couplé de ses variables d'état liées aux interactions électromagnétiques entre le stator et le rotor.

2.2.Description du moteur asynchrone:

Le moteur asynchrone triphasé est constitué principalement du stator et du rotor [2].

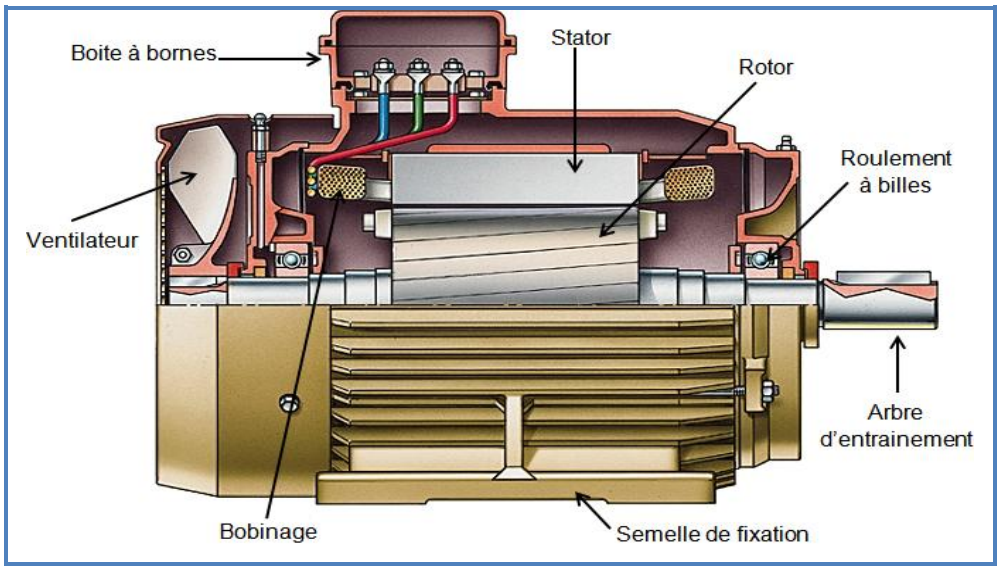

Figure 2.1 : le moteur asynchrone.

2.2.1.Le stator :

Le stator est une pièce construite en matériau ferromagnétique, c'est la partie statique liée à la carcasse, il contient un bobinage triphasé constitué de trois enroulements identiques correctement disposés le long d'un entrefer de telle sorte que leurs axes forment des angles de 120^0 entre eux.

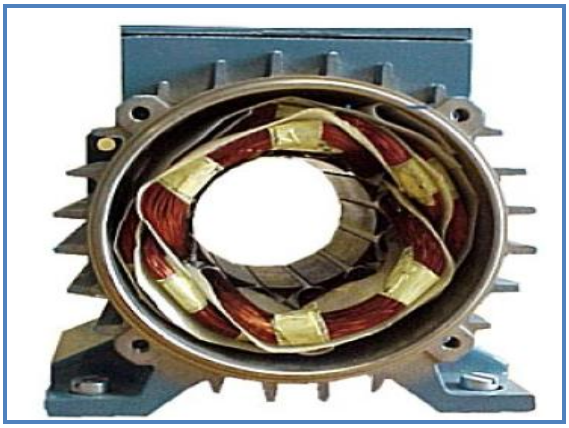

Figure 2.2 : le stator.

2.2.2.Le rotor **:**

Le rotor est la partie rotative du moteur asynchrone. Il est placé à l'intérieur du stator, et constitué d'un empilage de tôles d'acier formant un cylindre claveté sur l'arbre du moteur. On distingue deux types de rotor :

2.2.2.1.Rotor à cage :

Constitué par un empilement de tôles percées de trous, dans les quelles, on loge des barres conductrices, ces barres sont court-circuitées à leurs extrémités par des couronnes conductrices, ce qui constitue une véritable cage d'écureuil.

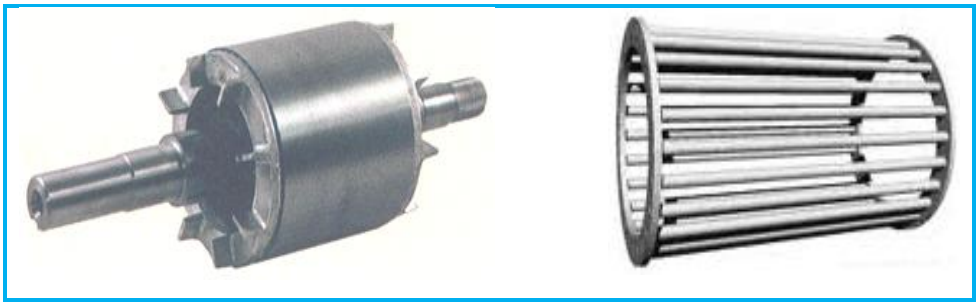

Figure 2.3 : Rotor à cage.

2.2.2.2.Rotor bobiné :

Le rotor bobiné est muni d'encoches dans lesquelles viennent se loger des enroulements.

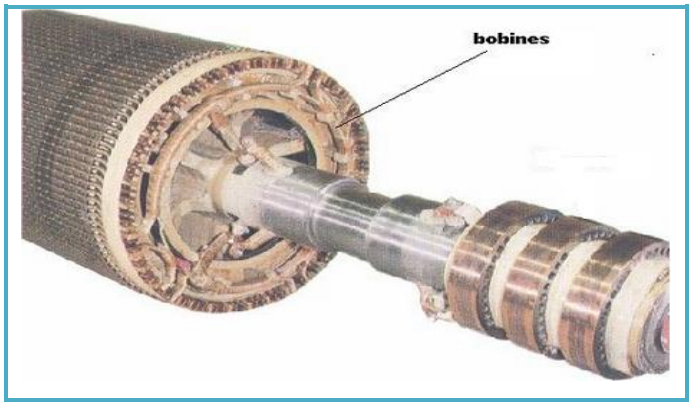

Figure 2.4 : Rotor bobiné

2.2.3.Entrefer **:**

C'est l'espace entre le stator et le rotor. Il doit être très étroit pour éviter les pertes de flux.

2.3.Principe de fonctionnement :

Le principe de fonctionnement de la plupart des machines tournantes à courant alternatif s'appuie sur la création d'un champ magnétique tournant auquel est soumise une partie libre en rotation. En particulier dans le cas du moteur asynchrone, c'est ce champ qui est responsable de la rotation de la machine.

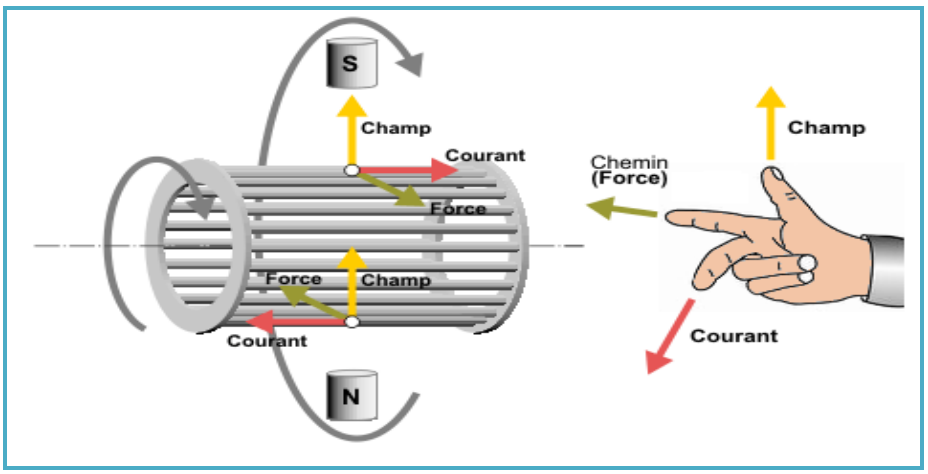

Figure 2.5 : champ magnétique tournant.

Les courants statoriques créent un champ magnétique tournant dans le stator. La fréquence de rotation de ce champ est imposée par la fréquence des courants statoriques, c'est-à-dire que sa vitesse de rotation est proportionnelle à la fréquence de l'alimentation électrique. La vitesse de ce champ tournant est appelée vitesse de synchronisme. L'enroulement au rotor est donc soumis à des variations de flux (du champ magnétique). Une force électromotrice induite apparaît qui crée des courants rotoriques. Ces courants sont responsables de l'apparition d'un couple qui tend à mettre le rotor en mouvement afin de s'opposer à la variation de flux (loi de Lenz), le rotor se met donc à tourner pour tenter de suivre le champ statorique. La différence de vitesse entre le rotor et le champ statorique est appelée vitesse de glissement.

2.4.Modélisation du moteur asynchrone a cage d'écureuil :

L'étude et l'analyse du moteur asynchrone consiste à obtenir un modèle mathématique représentatif de son fonctionnement, qui permet de prévoir le comportement de ce système et l'évolution de ses variables d'état sous l'action d'un événement particulier avec une prise en compte de toutes les simplifications possibles et leurs influences sur les résultats de synthèse [4].

2.4.1.Hypothèses simplificatrices :

La modélisation du moteur asynchrone s'appuie sur certain nombre d'hypothèses [5, 6, 7, 8 ,22] :

- Parfaite symétrie de construction.
- La répartition de la force magnétomotrice est sinusoïdale.
- Il n'y a pas d'effet de peau.
- Le stator est considéré comme lisse et l'entrefer constant (l'effet des encoches est négligé).
- Nous supposons que nous travaillons en régime non saturé.

Par conséquence à ces simplifications, les flux sont additifs, les inductances propres sont constantes et une variation sinusoïdale pour les inductances mutuelles en fonction de l'angle électrique de leurs axes de rotation.

Dans ce qui suit nous présentons les relations mathématiques permettant la modélisation du moteur asynchrone triphasé dans le repère triphasé ainsi que le repère diphasé.

2.4.2.Repère triphasé :

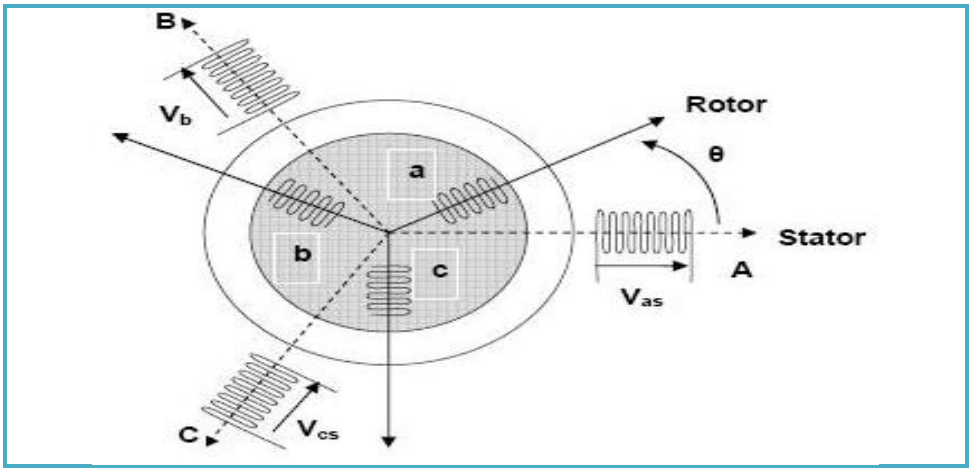

Figure 2.6 : Représentation de la MAS dans un repère triphasé.

2.4.2.1.Equations électriques:

La figure 2.7 représente les enroulements des trois phases statoriques et des trois phases rotoriques dans l'espace, où les phases rotoriques sont court-circuitées sur elle-même, et θ représente l'angle électrique entre la phase (A) statorique et la phase (a) rotorique. Ces enroulements peuvent être représentés comme indiqué en figure 2.8.

La loi des mailles s'exprime par la relation : [8]

$$V = Ri - e$$

$$= Ri + \frac{d\varphi}{dt}$$

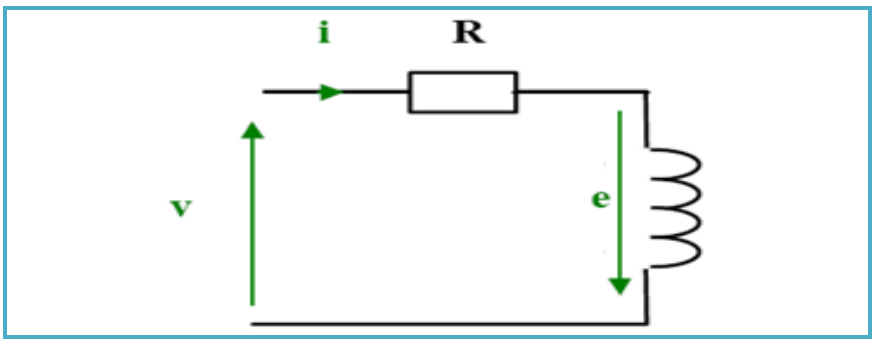

Figure 2.7 : Modèle d'une phase avec force électromotrice.

En appliquant la loi d'Ohm généralisée ainsi que la loi Lenz-Faraday à chaque enroulement de la machine, nous obtenons :

❖ Les tensions statoriques

$$\begin{cases} v_{as} = \dfrac{d\varphi_{as}}{dt} + R_s i_{as} \\ v_{bs} = \dfrac{d\varphi_{bs}}{dt} + R_s i_{bs} \\ v_{cs} = \dfrac{d\varphi_{cs}}{dt} + R_s i_{cs} \end{cases} \qquad \textbf{(2.1)}$$

Tel que, d'après la figure 2.8, on a :

- V représente soit v_{as}, v_{bs} ou v_{cs} : les tensions statorique sur les axes ABC.
- i représente soit i_{as}, i_{bs} ou i_{cs} : les courants statorique sur les axes ABC.
- φ représente soit φ_{as}, φ_{bs} ou φ_{cs} : les flux statorique sur les axes ABC.
- R représente R_s: la résistance statorique sur les axes ABC.

❖ Les tensions rotoriques

$$\begin{cases} v_{ar} = \dfrac{d\varphi_{ar}}{dt} + R_r i_{ar} = 0 \\ v_{br} = \dfrac{d\varphi_{br}}{dt} + R_r i_{br} = 0 \\ v_{cr} = \dfrac{d\varphi_{cr}}{dt} + R_r i_{cr} = 0 \end{cases} \qquad \textbf{(2.2)}$$

Tel que, d'après la figure 2.8, on a :

- V représente soit v_{ar}, v_{br} ou v_{cr} : les tensions rotoriques sur les axes abc.
- i représente soit i_{ar}, i_{br} ou i_{cr} : les courants rotoriques sur les axes abc.
- φ représente soit φ_{ar}, φ_{br} ou φ_{cr} : les flux rotoriques sur les axes abc.
- R représente R_r: la résistance rotorique sur les axes abc.

Les tensions rotorique sont nulles, du fait que le rotor du moteur asynchrone à cage est fermé sur lui-même (court-circuité).

2.4.2.2.Les équations des flux magnétiques :

Les équations des flux en fonction des courants s'obtiennent à partir des différentes inductances (propres et mutuelles), dont certaines dépendent du temps par l'intermédiaire de l'angle électrique θ.

- Les équations statoriques :

$$\begin{bmatrix}\varphi_{as}\\ \varphi_{bs}\\ \varphi_{cs}\end{bmatrix} = [L_{ss}]\begin{bmatrix}i_{as}\\ i_{bs}\\ i_{cs}\end{bmatrix} + [L_{sr}]\begin{bmatrix}i_{ar}\\ i_{br}\\ i_{cr}\end{bmatrix} \quad \textbf{(2.3)}$$

φ_{as},φ_{bs} et φ_{cs} sont les flux statorique sur les axes ABC.

- Les équations rotoriques :

$$\begin{bmatrix}\varphi_{ar}\\ \varphi_{br}\\ \varphi_{cr}\end{bmatrix} = [L_{rr}]\begin{bmatrix}i_{ar}\\ i_{br}\\ i_{cr}\end{bmatrix} + [L_{sr}]\begin{bmatrix}i_{as}\\ i_{bs}\\ i_{cs}\end{bmatrix} \quad \textbf{(2.4)}$$

φ_{ar},φ_{br} et φ_{cr} sont les flux rotorique sur les axes abc.
Les matrices des inductances sont données par :

$$[L_{ss}] = \begin{bmatrix}l_s & l_{ss} & l_{ss}\\ l_{ss} & l_s & l_{ss}\\ l_{ss} & l_{ss} & l_s\end{bmatrix} \quad \textbf{(2.5)}$$

$$[L_{rr}] = \begin{bmatrix}l_r & l_{rr} & l_{rr}\\ l_{rr} & l_r & l_{rr}\\ l_{rr} & l_{rr} & l_r\end{bmatrix} \quad \textbf{(2.6)}$$

$$[L_{sr}] = L_{sr}\begin{bmatrix}cos(\theta) & cos(\theta + \frac{2\pi}{3}) & cos(\theta - \frac{2\pi}{3})\\ cos(\theta - \frac{2\pi}{3}) & cos(\theta) & cos(\theta + \frac{2\pi}{3})\\ cos(\theta + \frac{2\pi}{3}) & cos(\theta - \frac{2\pi}{3}) & cos(\theta)\end{bmatrix} \quad \textbf{(2.7)}$$

$[L_{rs}] = [L_{sr}]^t$

$[L_{ss}]$,$[L_{rr}]$: Matrices des inductances.

l_s : Inductance propre d'une phase statorique.

l_r : Inductance propre d'une phase rotorique.

l_{ss} : Inductance mutuelle entre deux phases statoriques.

l_{rr} : Inductance mutuelle entre deux phases rotoriques.

l_{sr} : Le maximum de l'inductance mutuelle entre une phase statorique et une phase rotorique.

2.4.2.3.L'équation mécanique :

Comme la somme des couples à l'arbre est équivalente au couple inertiel, il s'ensuit :

$$J\frac{d\Omega}{\mathrm{dt}} = Cem - C_r - f_v\Omega \qquad \textbf{(2.8)}$$

Avec :

$$Cem = p\begin{bmatrix} i_{as} \\ i_{bs} \\ i_{cs} \end{bmatrix}^t \frac{d[L_{sr}]}{dt}\begin{bmatrix} i_{ar} \\ i_{br} \\ i_{cr} \end{bmatrix} \qquad \textbf{(2.9)}$$

Cem : Couple électromagnétique.

C_r : Couple résistant.

f_v : Coefficient de frottement visqueux.

J : Moment d'inertie.

p : Nombre de paires de pôles.

2.4.3.Repère diphasé :

2.4.3.1.Application de la transformation de PARK sur la machine asynchrone :

La variation des coefficients de la matrice $[L_{sr}]$ en fonction de temps par l'intermédiaire de l'angle (θ) rend difficile l'utilisation du modèle pour un calcul donné, cette difficulté se résout en application sur le système d'équation une transformation qui rend les paramètres du modèle indépendants du temps, celle-ci dite transformation de Park.

Cette dernière, On peut la comprendre comme une transformation des trois enroulements de la MAS à seulement deux enroulements, comme le montre la figure (2.9).

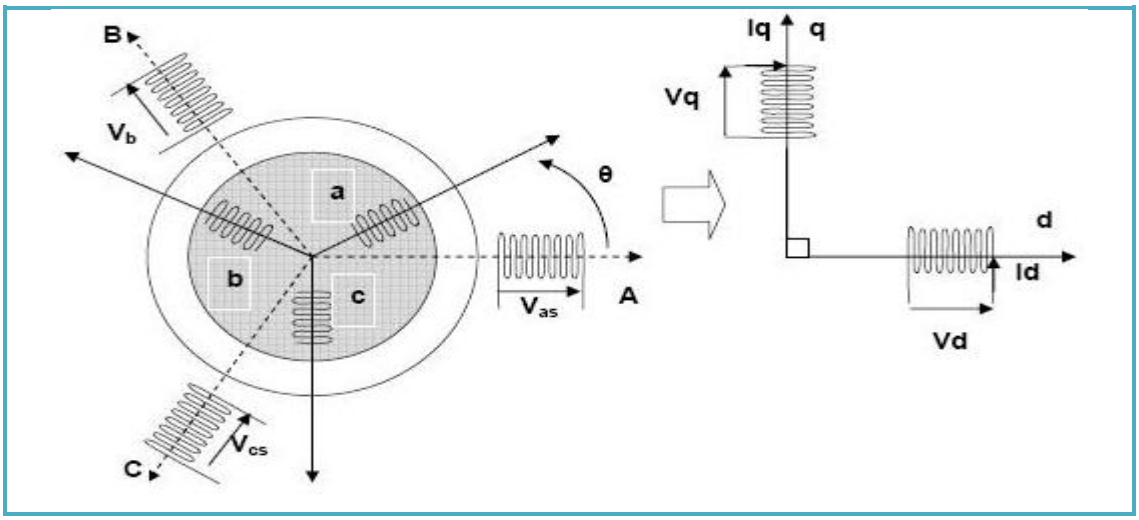

Figure 2.8 : Représentation de la MAS dans les repères triphasé et diphasé.

Cette transformation permet de faire le passage d'un système triphasé (*a, b, c*) réelle vers un système (*d, q, H*) fictif constitué des grandeurs diphasées équivalentes (*d, q*) et d'une composante homopolaire (*H*), qui n'est pas prise en compte dans le modèle usuel de la machine.

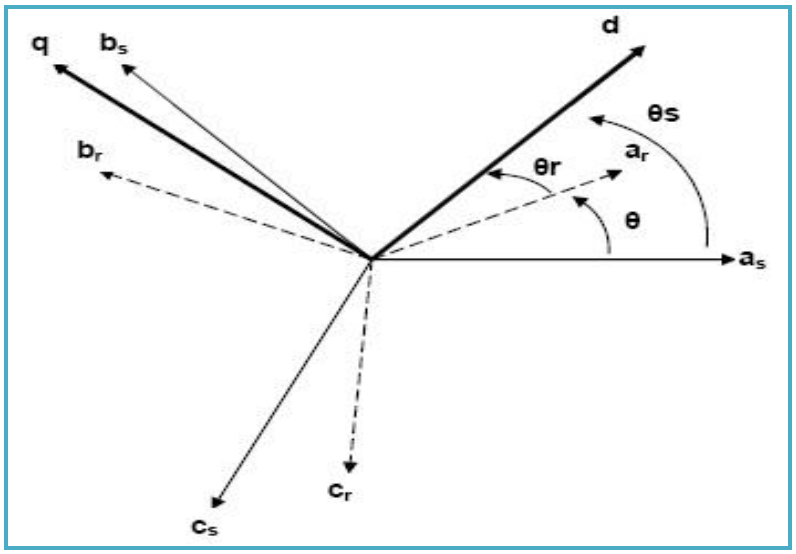

Figure 2.9 : représentation des axes de la machine.

La matrice de transformation de Park s'exprime par :

$$T_{33} = \sqrt{\frac{2}{3}}\begin{bmatrix} \cos(\Psi) & \cos\left(\Psi - \frac{2\pi}{3}\right) & \cos\left(\Psi - \frac{4\pi}{3}\right) \\ -\sin(\Psi) & -\sin\left(\Psi - \frac{2\pi}{3}\right) & -\sin\left(\Psi - \frac{4\pi}{3}\right) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \quad \textbf{(2.10)}$$

Ψ: Le déplacement angulaire du nouveau référentiel par rapport au référentiel réel.

Cette matrice est orthogonale, donc sa matrice inverse est la matrice transposée.

$$[T]^{-1} = [T]^{t} = \sqrt{\frac{2}{3}}\begin{bmatrix} \cos(\Psi) & -\sin(\Psi) & \frac{1}{\sqrt{2}} \\ \cos\left(\Psi - \frac{2\pi}{3}\right) & -\sin\left(\Psi - \frac{2\pi}{3}\right) & \frac{1}{\sqrt{2}} \\ \cos\left(\Psi - \frac{4\pi}{3}\right) & -\sin\left(\Psi - \frac{4\pi}{3}\right) & \frac{1}{\sqrt{2}} \end{bmatrix} \quad \textbf{(2.11)}$$

L'alimentation triphasée étant équilibrée, la somme des courants $\sum i_{a,b,c}$ est donc nulle. Le courant et le flux homopolaires s'annulent. Par conséquent, cet axe n'est pas pris en compte dans la transformation et la matrice de passage de Park avec sa version modifiée se résume dans l'équation (2.12) [2]:

$$T_{23} = \sqrt{\frac{2}{3}}\begin{bmatrix} \cos(\Psi) & \cos\left(\Psi - \frac{2\pi}{3}\right) & \cos\left(\Psi - \frac{4\pi}{3}\right) \\ -\sin(\Psi) & -\sin\left(\Psi - \frac{2\pi}{3}\right) & -\sin\left(\Psi - \frac{4\pi}{3}\right) \end{bmatrix} \quad \textbf{(2.12)}$$

En supposant que le référentiel diphasé tourne avec une vitesse ω_s bien déterminé, faisant un angle θ_s avec l'axe statorique *Sa*, pris comme référence. L'application du modèle de Park sur le système d'équations donne :

- Pour les tensions :

$$[v_{dq}] = [T_{23}][v_{abc}]$$

$$\begin{cases} v_{ds} = R_s I_{ds} + \dfrac{d\varphi_{ds}}{dt} - \dfrac{d\theta_s}{dt}\varphi_{qs} \\ v_{qs} = R_s I_{qs} + \dfrac{d\varphi_{qs}}{dt} + \dfrac{d\theta_s}{dt}\varphi_{ds} \end{cases} \quad \textbf{(2.13)}$$

$$\begin{cases} v_{dr} = R_r I_{dr} + \dfrac{d\varphi_{dr}}{dt} - \dfrac{d\theta}{dt}\varphi_{qr} = 0 \\ v_{qr} = R_r I_{qr} + \dfrac{d\varphi_{qr}}{dt} + \dfrac{d\theta}{dt}\varphi_{dr} = 0 \end{cases} \quad \textbf{(2.14)}$$

❖ Pour les flux :

$$[\varphi_{dq}] = [T][\varphi_{abc}]$$

$$\begin{cases} \varphi_{ds} = L_s i_{ds} + L_m i_{dr} \\ \varphi_{qs} = L_s i_{qs} + L_m i_{qr} \\ \varphi_{dr} = L_m i_{ds} + L_r i_{dr} \\ \varphi_{qr} = L_m i_{qs} + L_r i_{qr} \end{cases} \quad \textbf{(2.15)}$$

Avec :

$L_s = l_s - l_{ss}$: Inductance propre cyclique du stator.

$L_r = l_r - l_{rr}$: Inductance propre cyclique du rotor.

$L_m = \frac{3}{2} L_{sr}$: Inductance mutuelle cyclique entre stator et rotor.

Les angles θ_r, θ_s et θ sont liés par :

$$\theta = \theta_s - \theta_r \quad \textbf{(2.16)}$$

❖ Pour L'équation mécanique :

On applique la transformé de PARK sur le couple électromotrice :

$$C_{em} = \left[[T]^{-1}[i_{sdq}]\right]^t \frac{d}{dt}\left([L_{sr}][T]^{-1}[i_{rdq}]\right) \quad \textbf{(2.17)}$$

En la simplifiant, on trouve :

$$Cem = p\frac{L_m}{L_r}(\varphi_{dr} I_{qs} - \varphi_{qr} I_{ds}) \quad \textbf{(2.18)}$$

En utilisant l'équation (2.18) dans (2.8), on trouve:

$$J\frac{d\Omega}{\mathrm{dt}} = p\frac{L_m}{L_r}\left(\varphi_{dr} I_{qs} - \varphi_{qr} I_{ds}\right) - C_r - f_v\Omega \quad \textbf{(2.19)}$$

Avec $\Omega = \frac{d\theta_r}{dt}$: Vitesse de rotation de la machine.

2.4.3.2.Choix du vecteur d'état et du référentiel :

Concernant le choix du vecteur d'état, les variables qu'on veut observer sont les courants statoriques, les flux rotoriques et la vitesse mécanique. Donc le vecteur d'état choisit est : X=$\left[I_{ds} \quad I_{qs} \quad \varphi_{dr} \quad \varphi_{qr} \quad \Omega\right]^T$.

Concernant le choix du référentiel, le repère diphasé orthonormé (*d,q*) peut être fixe ou tournant par rapport aux armatures de la machine. Judicieusement, il existe trois systèmes d'axes de référence ayant des spécificités distinctes et intéressantes [2, 9].

- Repère fixe lié au stator ou : $\omega_s = 0.$ $\omega_r = \omega.$
- Repère fixe lié au rotor ou : $\omega_s = \omega,$ $\omega_r = 0.$
- Repère lié au champ tournant qui dépend des deux vitesses : ω_s, ω_r. Ou $\omega_s = \frac{d\theta}{dt}$: Pulsation du référentiel d-q.

2.4.3.3.Modèle de la machine asynchrone lié au stator :

Ce modèle s'obtient en assignant une vitesse nulle au référentiel biphasé ($\omega_s = 0$) et en alignant l'axe d de ce dernier avec l'axe statorique Sa. Dans ce référentiel, on peut voir l'évolution et les propriétés des grandeurs statorique. En substituant les courants statorique dans les équations 2.13 et 2.15 par les flux rotoriques (en utilisant l'équation 2.4), le modèle résultant s'exprime par :

$$\begin{cases} \frac{dI_{ds}}{dt} = -\gamma I_{ds} + \alpha\beta\varphi_{dr} + p\beta\Omega\,\varphi_{qr} + aV_{ds} \\ \frac{dI_{qs}}{dt} = -\gamma I_{qs} + \alpha\beta\varphi_{qr} - p\beta\Omega\,\varphi_{dr} + aV_{qs} \\ \frac{d\varphi_{dr}}{dt} = \alpha L_m I_{ds} - \alpha\varphi_{dr} - p\Omega\,\varphi_{qr} \\ \frac{d\varphi_{qr}}{dt} = \alpha L_m I_{qs} - \alpha\varphi_{qr} + p\Omega\,\varphi_{dr} \\ \frac{d\Omega}{dt} = \mu(\varphi_{dr} I_{qs} - \varphi_{qr} I_{ds}) - \frac{C_r}{J} - \frac{f_v}{J}\Omega \end{cases} \qquad \textbf{(2.20)}$$

Avec :

$$\alpha = \frac{R_r}{L_r} \quad \beta = \frac{L_m}{\sigma L_s L_r} \quad \gamma = \frac{L_m^2 R_r}{\sigma L_s L_r^2} + \frac{R_s}{\sigma L_s} \quad \mu = \frac{pL_m}{JL_r} \quad \sigma = 1 - \frac{L_m^2}{L_s L_r} \quad a = \frac{1}{\sigma L_s}$$

2.5.Conclusion :

Ce chapitre a été consacré à l'étude de la machine asynchrone dans le but de la modélisée en un système d'équations. En premier lieu nous avons modélisé la machine asynchrone dans le repère triphasé, puis à l'aide de la transformation de PARK on a passé à un repère diphasé équivalent. Cette représentation nous a permis de réduire l'ordre du système en isolant la composante homopolaire.

Le modèle d'état choisi dans ce travail pour décrire le moteur asynchrone est celui d'un système d'équations multi-variable non linéaire. Le choix des sorties est lié à la nature de l'étude effectuée sur la machine. Dans ce cas le choix du vecteur d'état dont les variables qu'on veut observer sont les courants statoriques, les flux rotoriques et la vitesse rotorique.

Toutes ces informations concernant les moteurs asynchrones et les circuits FPGA ainsi que le langage de description VHDL nous sont utiles dans le chapitre suivant, dont on va décrire les étapes de conception de notre simulateur dynamique.

CHPITRE 3

CONCEPTION DU MODELE DE LA MAS SUR CIRCUIT FPGA

L'objectif de ce chapitre est, premièrement, créer un programme du modèle de la MAS étudié dans le deuxième chapitre, avec le logiciel MATLAB, afin de simuler son comportement, ainsi prélever les résultats obtenus.

Deuxièmement, créer un programme du modèle de la MAS à l'aide du langage de description VHDL avec le logiciel XILINX, et vérifier sa fonctionnalité. Puis, en utilisant le logiciel MODELSIM on effectue une simulation du comportement de son architecture.

En fin, nous comparons les résultats obtenus à l'aide du logiciel MATLAB avec ceux acquis de l'architecture proposé à l'aide du logiciel XILINX par l'utilisation des outils du SystemGenerator, afin de vérifier si ces deux résultats coïncident parfaitement.

3.1.Introduction :

La machine asynchrone est un système multi-variable, non linéaire, fortement couplé à dynamique rapide et à paramètres variant dans le temps. Elle comporte aussi plusieurs avantages par rapport aux autres machines électrique, parmi lesquels nous pouvons citer : robustesse, entretien moins fréquent et faible coût. Ceci dis, la machine asynchrone est de loin la plus utilisée dans les applications requérant la variation de vitesse. Cependant, pour profiter des nombreux avantages de cette machine, des efforts énormes ont été déployés pour maitriser son comportement et l'aligner suivant les exigences de l'application souhaitée. Dans ce sens, différentes schémas et algorithmes de commande ont été proposés, néanmoins, ces solution, malgré leur performance deviennent très compliqués et très encombrantes lorsqu'on cherche à résoudre le maximum de problèmes de la machine asynchrone.

En utilisant les solutions logicielles, on fait dans la plupart des cas un compromis entre la complexité des algorithmes, la précision des résultats et la rapidité. Cependant, si ce compromis peut amener à des solutions moins couteuses et exploitables industriellement, il provoque une carence de performance qu'on peut récupérer si on dispose de hardware plus sophistiqué.

Avec l'avancement technologique dans le domaine de la micro-électronique, de nouvelles solutions de conception matérielles, telle que les FPGA sont disponibles et peuvent être utilisées comme cibles numériques pour l'implémentation des algorithmes. Le parallélisme inhérent de ces nouvelles solutions numérique ainsi que leurs grandes capacités de calcul font que les délais de temps de calcul sont négligeables en dépit de la complexité des algorithmes a implémentés. Par ailleurs, par rapport aux solutions logicielles, les solutions matérielles offrent au concepteur un accès à la partie architecture matérielle, puisque c'est le concepteur lui-même qui assure sa conception.

On peut simuler le modèle de la MAS avec une solution logicielle (software) comme on peut le faire avec une solution matérielle (hardware). Mais lorsqu'on ajoute des onduleurs, des régulateurs, intégrateurs et boucle de routeur, la simulation en software dure un temps important, et en augmentant la précision, la simulation devient de plus en plus longue (plusieurs jours). Afin d'éviter ce problème on implémente le modèle de la MAS sur carte FPGA. Rappelons que l'objectif de ce mémoire, est de mettre en évidence la contribution des circuits reconfigurables FPGA dans l'accélération de simulation des systèmes dynamiques.

Pour implémenter le modèle de la machine asynchrone sur le circuit FPGA on a utilisé le langage de description VHDL.

Dans ce chapitre, nous décrivons les différentes étapes pour implémenter le modèle de la machine asynchrone sur une carte FPGA.

3.2.Travail sous MATLAB :

MATLAB contient une interface graphique puissante, ainsi qu'une grande variété d'algorithmes scientifiques, de plus c'est un langage simple et très efficace. Il est beaucoup plus concis que les anciens langages (C, Pascal, Fortran, Basic …).

Pour la résolution du modèle de la machine asynchrone avec MATLAB, on a choisi d'utilisé une méthode numérique, à savoir « RUNGE KUTTA ».

3.2.1.Méthode de Runge-Kutta :

Les méthodes de Runge-Kutta sont des méthodes d'analyse numérique d'approximation de solutions d'équations différentielles. Elles ont été nommées ainsi en l'honneur des mathématiciens Carl Runge et Martin Wilhelm Kutta lesquels élaborèrent la méthode en 1901.

Ces méthodes reposent sur le principe de l'itération, c'est-à-dire qu'une première estimation de la solution est utilisée pour calculer une seconde estimation, plus précise, et ainsi de suite.

Il existe trois méthodes de Runge-Kutta qui sont nommées RK1, RK2 et RK4. La dernière méthode (RK4) est celle qui nous intéresse.

- La méthode de Runge-Kutta d'ordre quatre (RK4) :

  Considérons le problème suivant :

  $y' = f(y),$ tel que : $y(0) = y_0$

  La méthode RK4 est donnée par l'équation :

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

  Où *h* est le pas de l'itération.

$k_1 = f(y_n),$ $k_2 = f(y_n + \frac{h}{2}k_1),$ $k_3 = f(y_n + \frac{h}{2}k_2),$ $k_4 = f(y_n + hk_3)$

L'idée est que la valeur suivante (y_{n+1}) est approchée par la somme de la valeur actuelle (y_n) et du produit de la taille de l'intervalle (h) par la pente estimée. La pente est obtenue par une moyenne pondérée de pentes :

- k_1 est la pente au début de l'intervalle.
- k_2 est la pente au milieu de l'intervalle, en utilisant la pente k_1 pour calculer la valeur de *y.*
- k_3 est de nouveau la pente au milieu de l'intervalle, mais obtenue cette fois en utilisant la pente k_2 pour calculer *y.*
- k_4 est la pente à la fin de l'intervalle, avec la valeur de *y* calculée en utilisant k_3.

Dans la moyenne des quatre pentes, un poids plus grand est donné aux pentes au point milieu.

$$pente = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

La méthode RK4 est une méthode d'ordre 4, ce qui signifie que l'erreur commise à chaque étape est de l'ordre de h^5, alors que l'erreur totale accumulée est de l'ordre de h^4.

3.2.2.Application de méthode RK4 au modèle de la MAS :

On reprend le système d'équations 1.22, et on pose :

$$\alpha\beta = C_1 \qquad p\beta = C_2 \qquad \alpha L_m = C_3$$

$$p = C_4 \qquad \frac{C_r}{J} = C_5 \qquad \frac{f_v}{J} = C_6$$

On obtient :

$$\begin{cases} \frac{dI_{ds}}{dt} = -\gamma I_{ds} + C_1\varphi_{dr} + C_2\Omega\,\varphi_{qr} + aV_{ds} \\ \frac{dI_{qs}}{dt} = -\gamma I_{qs} + C_1\varphi_{qr} - C_2\Omega\,\varphi_{dr} + aV_{qs} \\ \frac{d\varphi_{dr}}{dt} = -\alpha\varphi_{dr} + C_3 I_{ds} - C_4\Omega\,\varphi_{qr} \\ \frac{d\varphi_{qr}}{dt} = -\alpha\varphi_{qr} + C_3 I_{qs} + C_4\Omega\,\varphi_{dr} \\ \frac{d\Omega}{dt} = -C_6\Omega + \mu(\varphi_{dr} I_{qs} - \varphi_{qr} I_{ds}) - C_5 \end{cases} \qquad (3.1)$$

En appliquant la méthode RK4 on trouve :

- Pour I_{ds}:

$$Ids_{n+1} = Ids_n + \frac{h}{6}(k_{Ids_1} + 2k_{Ids_2} + 2k_{Ids_3} + k_{Ids_4}) \quad (3.2)$$

Avec :

$$\begin{cases} k_{Ids_1} = -\gamma Ids_n + C_1\varphi dr_n + C_2\Omega_n\,\varphi qr_n + aV_{ds} \\ k_{Ids_2} = -\gamma(Ids_n + \frac{k_{Ids_1}}{2}) + C_1\varphi dr_n + C_2\Omega_n\varphi qr_n + aV_{ds} \\ k_{Ids_3} = -\gamma(Ids_n + \frac{k_{Ids_2}}{2}) + C_1\varphi dr_n + C_2\Omega_n\varphi qr_n + aV_{ds} \\ k_{Ids_4} = -\gamma(Ids_n + \frac{k_{Ids_3}}{2}) + C_1\varphi dr_n + C_2\Omega_n\varphi qr_n + aV_{ds} \end{cases} \quad (3.3)$$

- Pour I_{qs} :

$$Iqs_{n+1} = Iqs_n + \frac{h}{6}(k_{Iqs_1} + 2k_{Iqs_2} + 2k_{Iqs_3} + k_{Iqs_4}) \quad (3.4)$$

Avec :

$$\begin{cases} k_{Iqs_1} = -\gamma Iqs_n + C_1\varphi qr_n - C_2\Omega_n\varphi dr_n + aV_{qs} \\ k_{Iqs_2} = -\gamma(Iqs_n + \frac{k_{Iqs_1}}{2}) + C_1\varphi qr_n - C_2\Omega_n * \varphi dr_n + aV_{qs} \\ k_{Iqs_3} = -\gamma(Iqs_n + \frac{k_{Iqs_2}}{2}) + C_1\varphi qr_n - C_2\Omega_n\varphi dr_n + aV_{qs} \\ k_{Iqs_4} = -\gamma(Iqs_n + \frac{k_{Iqs_3}}{2}) + C_1\varphi qr_n - C_2\Omega_n\varphi dr_n + aV_{qs} \end{cases} \quad (3.5)$$

- Pour φ_{dr}:

$$\varphi dr_{n+1} = \varphi dr_n + \frac{h}{6}(k_{\varphi dr_1} + 2k_{\varphi dr_2} + 2k_{\varphi dr_3} + k_{\varphi dr_4}) \quad (3.6)$$

Avec :

$$\begin{cases} k_{\varphi dr_1} = -\alpha\varphi dr_n + C_3 Ids_n - C_4\Omega_n\varphi qr_n \\ k_{\varphi dr_2} = -\alpha(\varphi dr_n + \frac{k_{\varphi dr_1}}{2}) + C_3 Ids_n - C_4\Omega_n\varphi qr_n \\ k_{\varphi dr_3} = -\alpha(\varphi dr_n + \frac{k_{\varphi dr_2}}{2}) + C_3 Ids_n - C_4\Omega_n\varphi qr_n \\ k_{\varphi dr_4} = -\alpha(\varphi dr_n + \frac{k_{\varphi dr_3}}{2}) + C_3 Ids_n - C_4\Omega_n\varphi qr_n \end{cases} \quad (3.7)$$

- Pour φ_{qr}:

$$\varphi qr_{n+1} = \varphi qr_n + \frac{h}{6}\left(k_{\varphi qr_1} + 2k_{\varphi qr_2} + 2k_{\varphi qr_3} + k_{\varphi qr_4}\right) \quad (3.8)$$

Avec :

$$\begin{cases} k_{\varphi qr_1} = -\alpha\varphi qr_n + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \\ k_{\varphi qr_2} = -\alpha(\varphi qr_n + \frac{k_{\varphi qr_1}}{2}) + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \\ k_{\varphi qr_3} = -\alpha(\varphi qr_n + \frac{k_{\varphi qr_2}}{2}) + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \\ k_{\varphi qr_4} = -\alpha(\varphi qr_n + \frac{k_{\varphi qr_3}}{2}) + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \end{cases} \quad (3.9)$$

- Pour Ω**:**

$$\Omega_{n+1} = \Omega_n + \frac{h}{6}(k_{\Omega_1} + 2k_{\Omega_2} + 2k_{\Omega_3} + k_{\Omega_4}) \quad (3.10)$$

Avec :

$$\begin{cases} k_{\Omega_1} = -C_6\Omega_n + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \\ k_{\Omega_2} = -C_6(\Omega_n + \frac{k_{\Omega_1}}{2}) + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \\ k_{\Omega_3} = -C_6(\Omega_n + \frac{k_{\Omega_2}}{2}) + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \\ k_{\Omega_4} = -C_6(\Omega_n + \frac{k_{\Omega_3}}{2}) + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \end{cases} \quad (3.11)$$

3.2.3.Programmation et simulation numérique du modèle de la MAS :

La simulation est un moyen efficace et économique utilisé pour faire des préliminaires et /ou comparatives, tant au stade de développement (conception), qu'au cours du fonctionnement normal des systèmes. Pour ce faire, l'environnement MATLAB offre plusieurs opportunités, en particulier pour les systèmes électriques et électroniques. A cet effet, ce logiciel a été exploité pour simuler le modèle de la MAS.

La programmation se fait en décrivant en langage MATLAB le modèle de la MAS représenté par le système d'équations (3.1) après lui avoir appliqué la méthode Runge-Kutta d'ordre 4. Donc on programme les systèmes d'équations de (3.1) jusqu'à (3.11). Les paramètres de la machine sont indiqués dans l'appendice B.

Les figures 3.1, 3.2, 3.3, 3.4 et 3.5 illustrent les résultats de la simulation obtenus en considérant un pas de simulation de 10^{-3} *s* sous MATLAB.

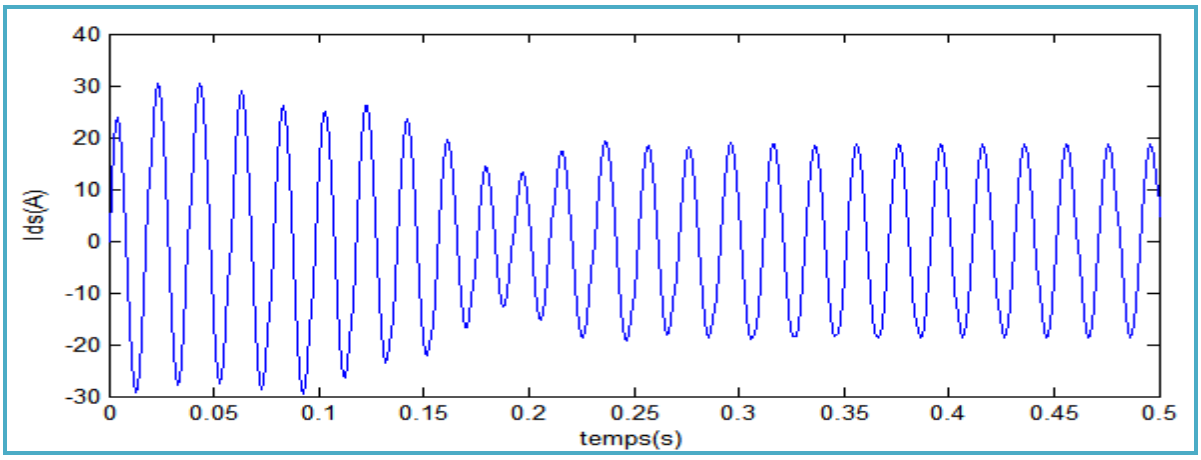

Figure 3.1 : le courant statorique Ids

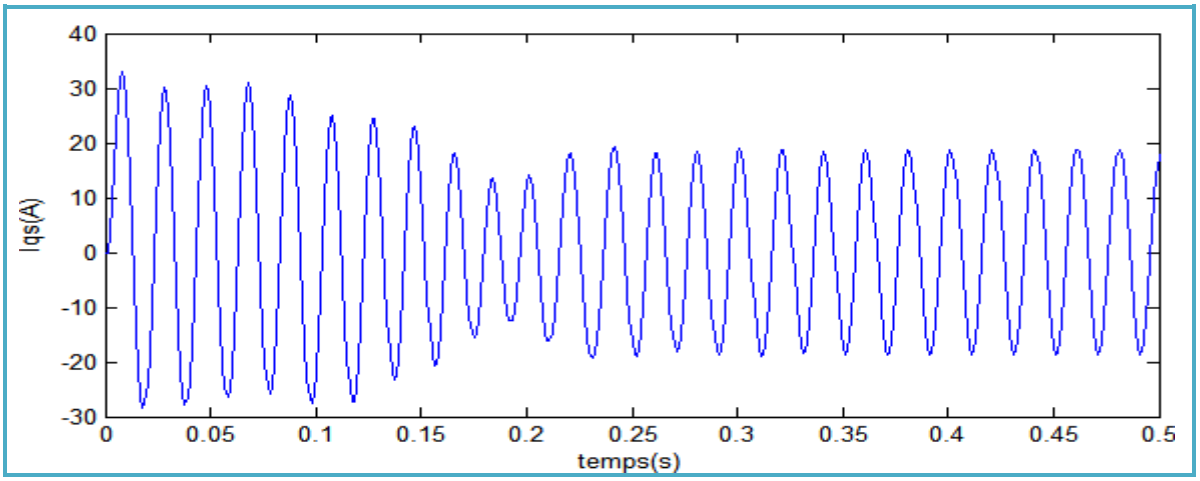

Figure 3.2 : le courant statorique Iqs

- On reconnait le classique appel de courant statorique au démarrage égal à 5 fois environ le courant nominal (régime transitoire). Après ils se stabilisent et prennent leur valeurs correspondantes au régime permanent (Figure 3.1 et 3.2).

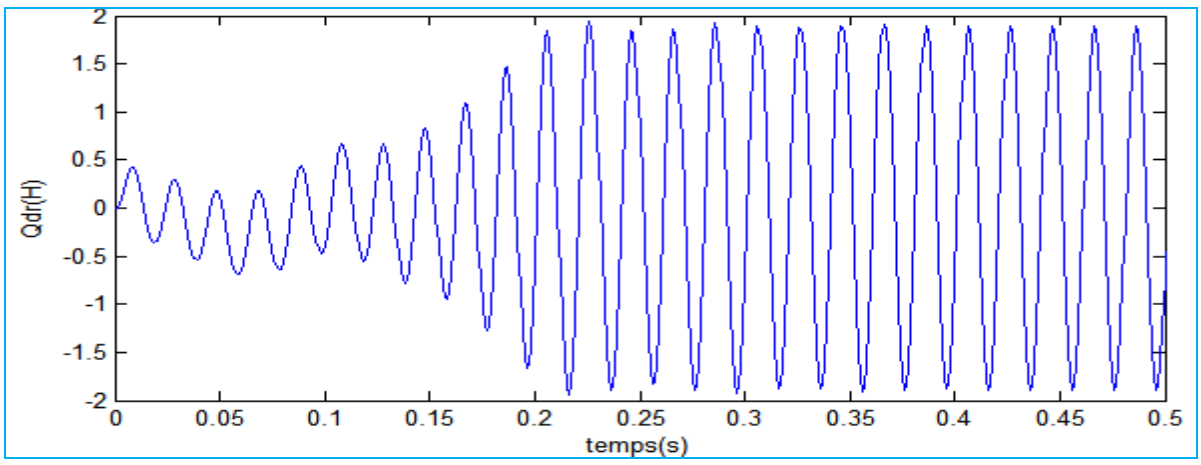

Figure 3.3 : le flux rotorique φdr

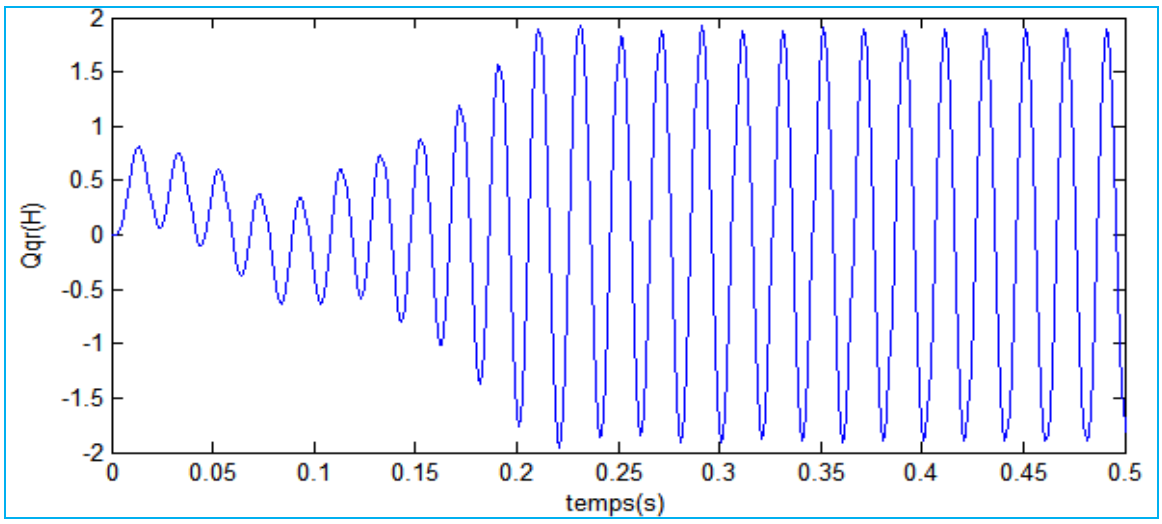

Figure 3.4 : le flux rotorique φqr

- Contrairement aux courants statoriques, on remarque une faible amplitude pendant le régime transitoire pour les flux rotoriques, après ces derniers prennent une forme sinusoïdale (Figure 3.3 et 3.4).

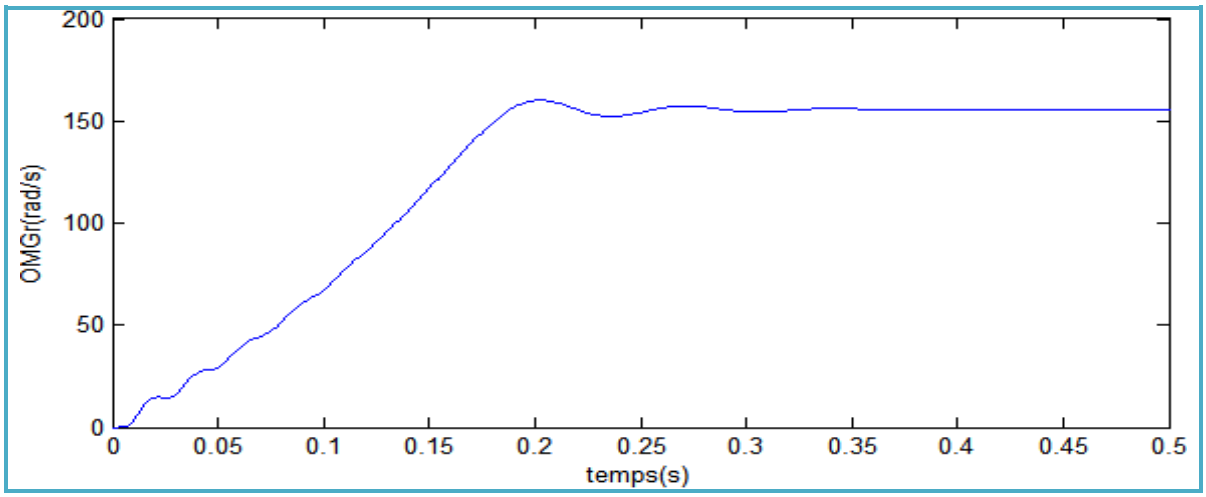

Figure 3.5 : la vitesse Ω

- La courbe de la vitesse présente des oscillations dans les premiers instants de démarrage avec un accroissement presque linéaire, après un temps d'environs 0.3 s, la vitesse de rotation s'établit à une valeur proche de la vitesse de synchronisme qui a une valeur de 156 rad/s puisque le moteur possède deux paires de pôles (Figure 3.5).

Afin de connaitre le temps pris par MATLAB pour effectuer la simulation, on utilise la commande Tic Toc. Pour ce faire, il suffit d'insérer la commande Tic au début

de la simulation (cela partira le chronomètre) et de placer Toc lorsque l'on veut que le temps arrête. MATLAB affichera alors le temps pris entre le Tic et le Toc.
Le temps pris par MATLAB pour effectuer une seul itération de la simulation du model de la MAS est égale à 0,2277 ms avec un ordinateur dont le processeur est un Core2Duo d'une fréquence de 1,66 GHz, et la RAM est égale 3 Go.

3.3.Travail sous XLINX ISE (Integrated Software Environment) :

XILINX ISE est un logiciel de programmation des produits Xilinx (CPLD, FPGA Spartan et Virtex, …). Il permet :

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

A travers ce logiciel, on va proposer et réalisé une architecture du modèle de la MAS avec le langage de description VHDL, en veillons à l'optimisé et utilisé que les ressources nécessaire à nos besoins.

Pour arriver à cela, on doit réaliser une bonne représentation des données et leurs différents codages. Aussi, on doit bien gérer les opérations arithmétiques, telle que l'addition, la soustraction et la multiplication. Tout ceci sera bien détailler dans les paragraphes suivants.

3.3.1.Représentation binaries des données:

Deux types de données sont utilisées dans ce projet ; les constantes et les variables. Les constantes sont les paramètres de la machine asynchrone ainsi que la matrice de transformation de Park. Les variables sont des grandeurs qui représentent les variations des différents signaux telles que les tensions, les courants, les flux et la vitesse, ainsi que les signaux intermédiaire.

Ces deux types de données sont soit des nombres entiers, soit des réels. On se qui concerne les nombres entiers, leur codage est simple. Par contre pour les nombres réels on doit prendre un choix concernant leur codage. Soit on choisit un format à virgule fixe ou un format a virgule flottante (appendice C).

Dans ce mémoire, on a opté pour représenter les données avec un format à virgule fixe, du fait de sa simplicité de manipulation et sa consommation moins importantes des ressources en termes de surface et d'énergie.

Ce type de représentation est référencé par le format $s[(m+n)/Qn]$ lorsqu'il s'agit d'une représentation signée, et par le format $u[(m+n)/Qn]$ lorsqu'il s'agit d'une représentation non signée. Tel que le nombre de bits total est égal à $(m+n)$, où les *m bits* les plus significatifs représentent la partie entière tandis que les *n bits* restants représentent la partie fractionnaire. Ce format est symbolisé dans la figure suivante.

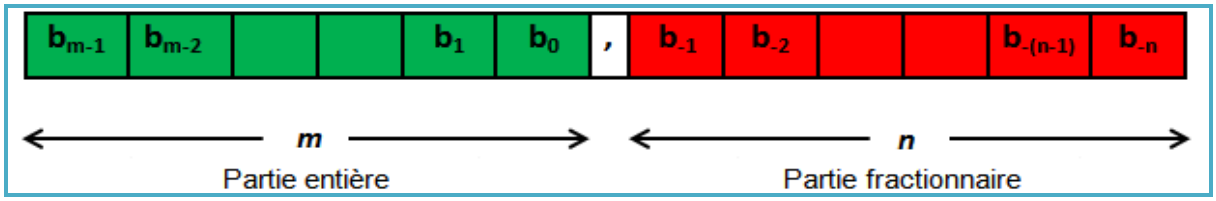

Figure 3.6: Représentation binaires des données.

3.3.2.Codages des données :

Afin de bien exploiter la carte FPGA, du point de vue rapidité et ressources consommées, on a choisi de coder chacune des données d'après ses besoin en nombres de bits en s'assurant qu'il ne dépasse pas 18 bits (la bibliothèque de XILINX ne contient que des multiplieurs (18*18) bits), de manière à avoir le codage le plus fidèle possible à la valeur réelle. Cependant, les valeurs des données que soit les constantes ou les variables sont de type réel (des nombres à virgule), et le langage VHDL ne traite que des nombres entiers. Donc il faut bien les manipuler dans la description du programme en effectuant les opérations arithmétiques.

3.3.3.Les opérations arithmétiques :

On peut évidemment effectuer quatre opérations arithmétiques fondamentales (addition, soustraction, multiplication et division) non seulement dans le système décimal mais aussi dans les autres systèmes numériques et en particulier dans le système binaire ; les règles du système décimal seront valables pour ces opérations.

Voici une représentation en langage de description VHDL des deux opérations les plus utilisé dans ce mémoire.

3.3.3.1.L'addition :

L'opération d'addition de deux représentations binaires s'effectue de façon similaire à l'addition décimale, c'est à dire en additionnant digit par digit les deux représentations alignées sur la virgule et en reportant une éventuelle retenue sur la colonne suivante. Pour additionner deux nombres binaires A et B, les signaux A et B doivent avoir la même longueur (nombre de bits) et donnent un résultat de même longueur plus un bit de retenue supplémentaire. En utilisant le langage de description VHDL pour additionner deux nombres A et B, on doit réaliser un programme qui satisfait les quatre possibilités :

- A et B positifs.
- A positif et B négatif.
- A négatif et B positif.
- A et B négatifs.

Par exemple, soit A sur 15 bits, B sur 12 bits et la sortie C sur 16 bits. Voici une représentation en code VHDL de l'addition:

```
Entity add1512_16 is
        Port (  A: in STD_LOGIC_VECTOR (14 downto 0);
                B: in STD_LOGIC_VECTOR (11 downto 0);
                C: out STD_LOGIC_VECTOR (15 downto 0);
                CLK: in std_logic);
End add1512_16;
Architecture Behavioral of add1512_16 is
Begin
Process (CLK)
    Begin
    If (CLK= '1' and CLK'event) then
If A(14) = '0' and B(11) = '0' then C <= signed('0'&A) + signed("0000"&B);
Elsif A(14) = '1' and B(11) = '1' then C <= signed('1'&A) + signed("1111"&B);
Elsif A(14) = '1' and B(11) = '0' then C <= signed('1'&A) + signed("0000"&B);
Elsif A(14) = '0' and B(11) = '1' then C <= signed('0'&A) + signed("1111"&B);
End if; End if;
End process;
End Behavioral;
```

D'après cette représentation, on remarque qu'on a réalisé quatre opérations de test sur les deux nombres à additionner (A et B) du fait qu'ils n'ont pas la même taille (nombres de bits). En plus on a effectué une extension de signe en ajoutant des bits du côté gauche (la partie entière), afin que leurs tailles soit égales à celle du résultat (C).

3.3.3.2.La multiplication :

Le processus de multiplication binaire est en fait beaucoup plus simple que le processus de multiplication décimale. Pour chaque digit du multiplicateur égal à 1, un produit partiel non nul est formé, ce produit est constitué du multiplicande décalé d'un certain nombre de positions de manière à aligner son digit de poids le plus faible avec le 1 correspondant du multiplicateur. Le produit final est obtenu par addition de tous les produits partiels.

Comme la bibliothèque de XILINX ne contient que des multiplieurs (18*18) bits, on a choisi de codées les données de manière à ne pas dépasser 18 bits.

En utilisant le langage de description VHDL pour multiplier deux nombres A et B, tel que A sur 15 bits, B sur 12 bits et la sortie C sur 27 bits. Voici une représentation en code VHDL de la multiplication :

```
Entity mult_1512_27 is
        Port ( A: in STD_LOGIC_VECTOR (14 downto 0);
               B: In STD_LOGIC_VECTOR (11 downto 0);
               C: Out STD_LOGIC_VECTOR (26 downto 0);
              CLK: in std_logic);
End mult_1512_27 ;
Architecture Behavioral of  mult_1512_27   is
Begin
        Process (CLK)
        Begin
        If (CLK= '1' and CLK'event) then  C <= Signed (A) * signed (B);
        End if;
        End process;
End Behavioral;
```

D'après cette représentation, on constate que la sortie C du multiplieur contient 27 bits. Ceci dit, si le résultat C est multiplié par un autre nombre, on doit lui effectuer une troncature afin que sa taille ne dépasse pas 18 bits.

La troncature consiste a éliminé des bits sans pour autant changer la valeur réelle. On a trois types de troncature :

- On élimine des bits du cotée droit (la partie fractionnaire).
- On élimine des bits du cotée gauche (la partie entière), ce qui revient a éliminé seulement les bits d'extension de signe.
- On élimine des bits des deux cotées (effectuer les deux précédentes éliminations).

Cette méthode permet de gagner en rapidité et en ressources consommés, mais son inconvénient majeur est la perte de précision.

Après avoir donné une représentation binaire des données et mettre en évidence les différentes opérations arithmétiques, on va détailler toutes les étapes de la réalisation de notre architecture avec le logiciel XILINX ISE.

3.3.4.Schéma synoptique :

Le synoptique du système global (Figure 3.7) est constitué d'un module pour générer l'alimentation triphasée et de trois modules pour établir le modèle de la MAS. Ces modules sont :

- Module de la génération de la tension triphasée.
- Module de la génération de la tension Vdqs.
- Module Runge-Kutta d'ordre 4 : il est composé de :
  - Module de calcul des pentes ($\sum k_i$).
  - Module de la Mémorisation et mise à jour de X_n.

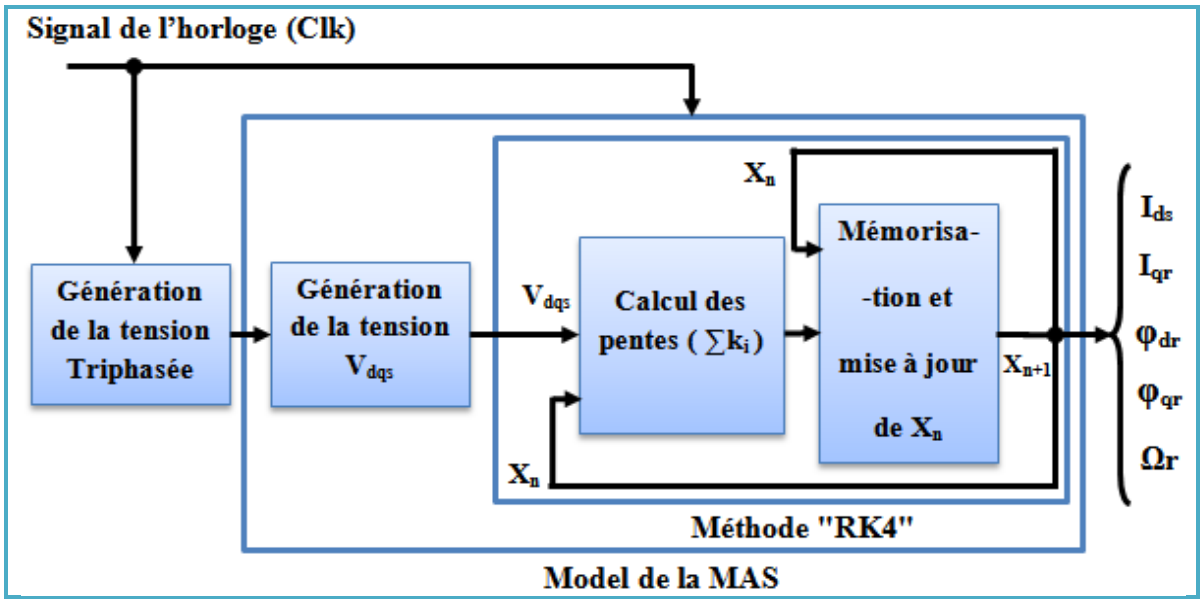

Figure 3.7 : Schéma synoptique.

3.3.5.Module de la génération de tension Triphasée :

Ce module sert à générer les trois tensions d'alimentation triphasées Va, Vb et Vc diphasées de 120° entre eux. Voici leurs équations respectives :

$$\begin{cases} \text{Va} = 220\sqrt{2}\,sin(\omega t) \\ \text{Vb} = 220\sqrt{2}\,sin(\omega t + {}^{2\pi}/_{3}) \\ \text{Vc} = 220\sqrt{2}\,sin(\omega t - {}^{2\pi}/_{3}) \end{cases} \qquad (3.12)$$

L'équation (3.12) montre que les trois tensions triphasées contiennent des fonctions sinusoïdes. Donc pour générer les trois tensions triphasées il faut d'abord générer les fonctions sinusoïdes.

3.3.6.Synthèse ou (génération) des fonctions sinusoïdes :

L'implémentions du modèle de notre machine, implique que son intégration doit être optimisée le plus possible pour économiser les ressources du circuit FPGA, ainsi les exploiter dans une autre réalisation, par exemple pour la commande vectorielle. L'utilisation d'une ROM représente une solution très avantagée pour générer une onde sinusoïdale.

Dans ce cas, les valeurs de sinus sont échantillonnées, puis codées en binaire et enfin stockées dans une mémoire. Les valeurs sont organisées tel que chaque valeur correspond à une adresse bien spécifie pour y accéder directement.

3.3.6.1.La taille de la ROM :

La taille de la ROM dépend du pas d'échantillonnage de la sinusoïde. Pour chaque pas, il existe un nombre de points ou de valeurs :

- Le pas $= 10^{-3} \Rightarrow 1000$ points
- Le pas $= 10^{-4} \Rightarrow 10\,000$ points
- Le pas $= 10^{-5} \Rightarrow 100\,000$ points

D'autre part, l'utilisation d'une ROM pour produire une sinusoïde est intéressante, mais cette solution devient encombrante lorsque le nombre de points à sauvegarder devient important, donc il faut que la manipulation de la ROM soit intelligente. Suivant le pas d'échantillonnage et le nombre de points on peut distinguer deux cas :

- Pour un pas d'échantillonnage assez grand, le nombre de points à mémorisés est réduit, donc on peut penser à stocker toute l'onde sinusoïde sur la ROM. Par exemple : pour le pas de 10^{-3}s, il est possible d'utiliser une ROM de 1000 adresses pour stocker les 1000 points résultants.
- Pour un très petit pas d'échantillonnage, le nombre de points devient important, la ROM exigée devient par conséquent importante (pour un pas de 10^{-4} on a 10000 points).

Donc, comme un signal sinusoïdal est un signal périodique alors la solution consiste à mémoriser une partie (période) de la sinusoïde et l'exploiter pour générer le reste de l'onde.

Avec un pas de simulation de 10^{-3} et une fréquence de 50 Hz, on déduit qu'à chaque période on a 21 valeurs, tel qu'à chaque pas de simulation on a une valeur.

3.3.6.2.Génération du signal sinusoïdale:

La génération du signal sinusoïdal nécessite une reproduction d'une période de 21 valeurs comprises entre [-1 ; 1] de ce signal. Voici les étapes pour générer un signale sinusoïdale :

- Créer une ROM de 21 mots de 9 bits dont le plus à gauche est le bit signe.
- Créer un compteur synchrone, pour l'incrémentation du bus d'adresse (ADR).
- Le bus de données (DATA) représente la sortie du signal sinusoïdal.

Les valeurs du sinus sont stockées dans une ROM de type LUT. La figure 3.8 représente le synoptique du Bloc fonctionnel générateur du signal sinusoïdal.

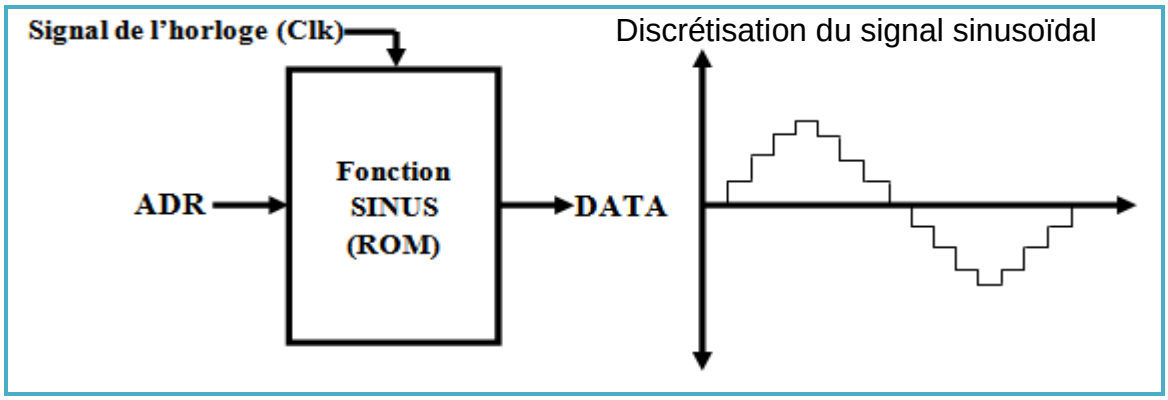

Figure 3.8 : Génération de la sinusoïde.

Après avoir généré la sinusoïde, on peut maintenant générer les tensions triphasées Va, Vb et Vc. Pour cela on doit disposer de trois ROM, chacune d'elles correspond à une équation de Va, Vb ou Vc. comme il est montré dans la figure suivante :

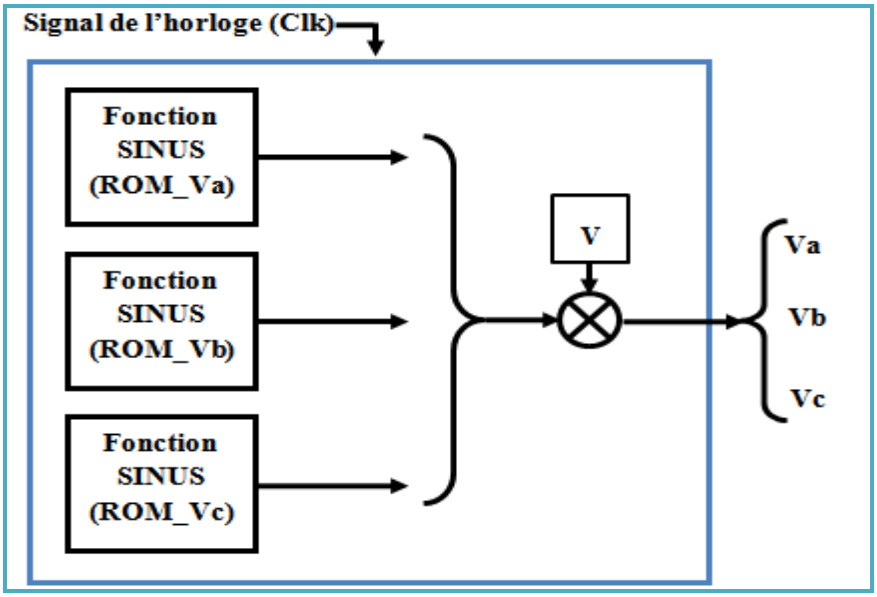

Figure 3.9 : Module de la génération de tension Triphasée.

Avec : $V = 220\sqrt{2}$.

3.3.7.Module de la génération des tensions Vds et Vqs :

Ce module sert à générer les deux tensions statoriques dans le référentiel (dq). Rappelons que la matrice de transformation de PARK s'exprime par :

$$T = \sqrt{\frac{2}{3}}\begin{bmatrix} \cos(\Psi) & \cos\left(\Psi - \frac{2\pi}{3}\right) & \cos\left(\Psi - \frac{4\pi}{3}\right) \\ -\sin(\Psi) & -\sin\left(\Psi - \frac{2\pi}{3}\right) & -\sin\left(\Psi - \frac{4\pi}{3}\right) \end{bmatrix}$$

Posons $\Psi = 0$, on trouve :

$$T = \sqrt{\frac{2}{3}}\begin{bmatrix} 1 & \cos\left(-\frac{2\pi}{3}\right) & \cos\left(-\frac{4\pi}{3}\right) \\ 0 & -\sin\left(-\frac{2\pi}{3}\right) & -\sin\left(-\frac{4\pi}{3}\right) \end{bmatrix}$$

$$= \sqrt{\frac{2}{3}}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}$$

Donc on a :

$$\begin{cases} v_{ds} = \sqrt{\frac{2}{3}}\left[Va + Vb\cos\left(-\frac{2\pi}{3}\right) + Vc\cos\left(-\frac{4\pi}{3}\right)\right] \\ v_{qs} = \sqrt{\frac{2}{3}} * \left[-Vb\sin\left(-\frac{2\pi}{3}\right) - Vc\sin\left(-\frac{4\pi}{3}\right)\right] \end{cases} \quad (3.14)$$

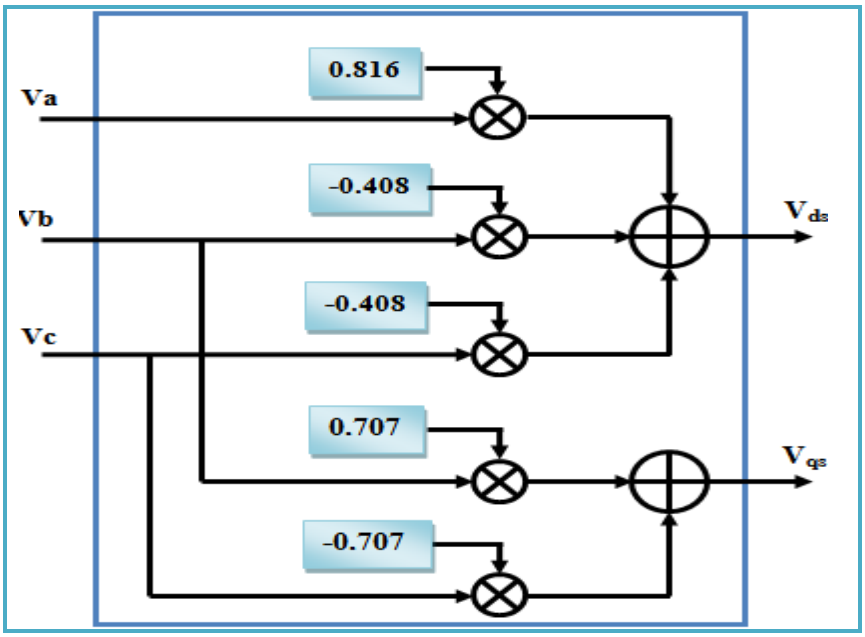

Figure 3.10 : Module de la génération des tensions Vdqs

3.3.8.Module Runge-Kutta d'ordre 4 (RK4):

Ce module sert a représenté la méthode de résolution itérative de Runge-Kutta d'ordre 4. Il se compose de deux parties :

3.3.8.1.Module de calcul des pentes ($\sum k_i$) :

Ce module représente l'équation suivante :

$$pente = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

Pour cela, on reprend les systèmes d'équations de (3.1) jusqu'à (3.11) afin de réaliser ses modules interne.

- **Pour** I_{ds}: équations (3.1) et (3.2).

$$Ids_{n+1} = Ids_n + \frac{h}{6}\left(k_{Ids_1} + 2k_{Ids_2} + 2k_{Ids_3} + k_{Ids_4}\right)$$
$$= Ids_n + h(pente\_Ids).$$

La figure 3.11 représente le calcul de « $pente\_Ids$ ».

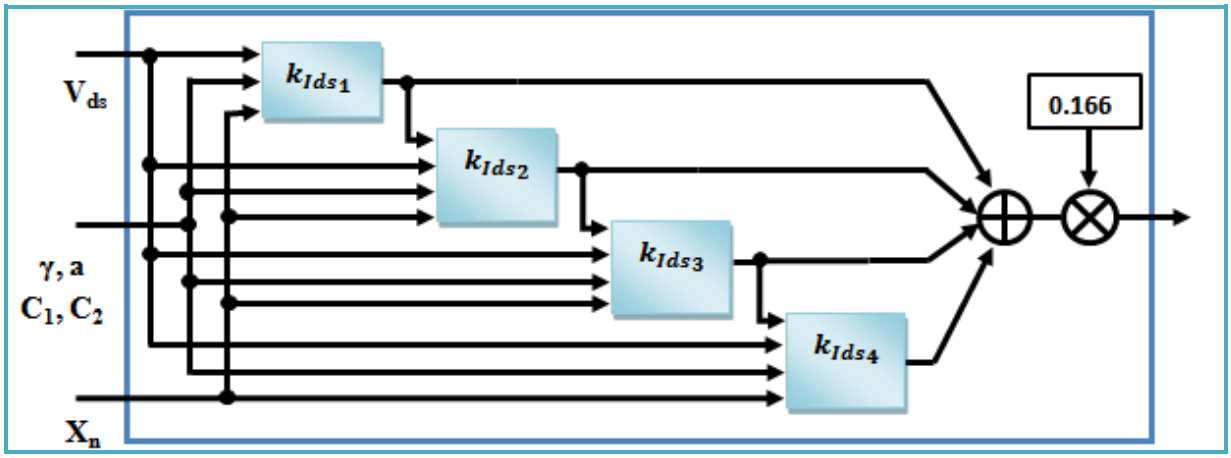

Figure 3.11 : Représentation de $pente\_Ids$.

Avec **:**

$$\begin{cases} k_{Ids_1} = -\gamma Ids_n + C_1\varphi dr_n + C_2\Omega_n\,\varphi qr_n + aV_{ds} \\ k_{Ids_2} = -\gamma(Ids_n + \frac{k_{Ids_1}}{2}) + C_1\varphi dr_n + C_2\Omega_n\varphi qr_n + aV_{ds} \\ k_{Ids_3} = -\gamma(Ids_n + \frac{k_{Ids_2}}{2}) + C_1\varphi dr_n + C_2\Omega_n\varphi qr_n + aV_{ds} \\ k_{Ids_4} = -\gamma(Ids_n + \frac{k_{Ids_3}}{2}) + C_1\varphi dr_n + C_2\Omega_n\varphi qr_n + aV_{ds} \end{cases}$$

La figure 3.12 représente le calcul de « k_{Ids_1} ».

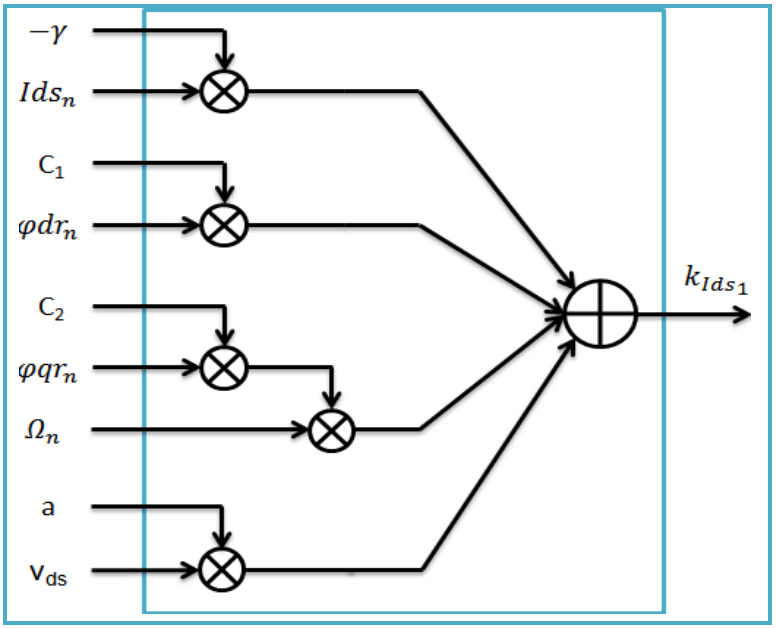

Figure 3.12 : Représentation de k_{Ids_1}.

La représentation de (k_{Ids_2}, k_{Ids_3} et k_{Ids_4}) ce fait d'après leur équations respectives, de la même manière que k_{Ids_1}.

- **Pour I_{qs} :** équations (3.1) et (3.2).

$$Iqs_{n+1} = Iqs_n + \frac{h}{6}\left(k_{Iqs_1} + 2k_{Iqs_2} + 2k_{Iqs_3} + k_{Iqs_4}\right)$$

$$= Iqs_n + h(pente\_Iqs)$$

La figure 3.13 représente le calcul de « $pente\_Iqs$ ».

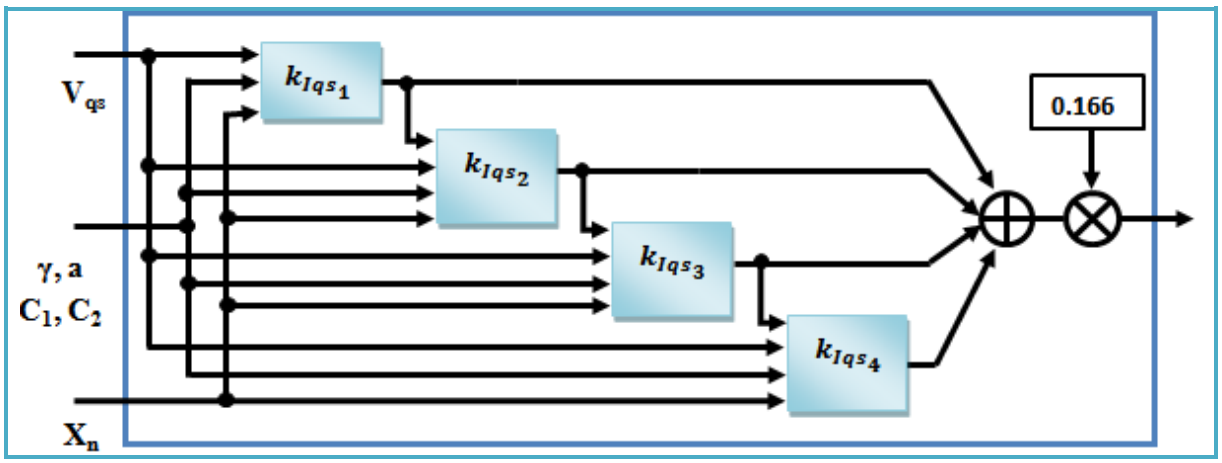

Figure 3.13 : Représentation de pente_Iqs.

Avec :

$$\begin{cases} k_{Iqs_1} = -\gamma Iqs_n + C_1\varphi qr_n - C_2\Omega_n\varphi dr_n + aV_{qs} \\ k_{Iqs_2} = -\gamma(Iqs_n + \frac{k_{Iqs_1}}{2}) + C_1\varphi qr_n - C_2\Omega_n * \varphi dr_n + aV_{qs} \\ k_{Iqs_3} = -\gamma(Iqs_n + \frac{k_{Iqs_2}}{2}) + C_1\varphi qr_n - C_2\Omega_n\varphi dr_n + aV_{qs} \\ k_{Iqs_4} = -\gamma(Iqs_n + \frac{k_{Iqs_3}}{2}) + C_1\varphi qr_n - C_2\Omega_n\varphi dr_n + aV_{qs} \end{cases}$$

La figure 3.14 représente le calcul de « k_{Iqs_1} ».

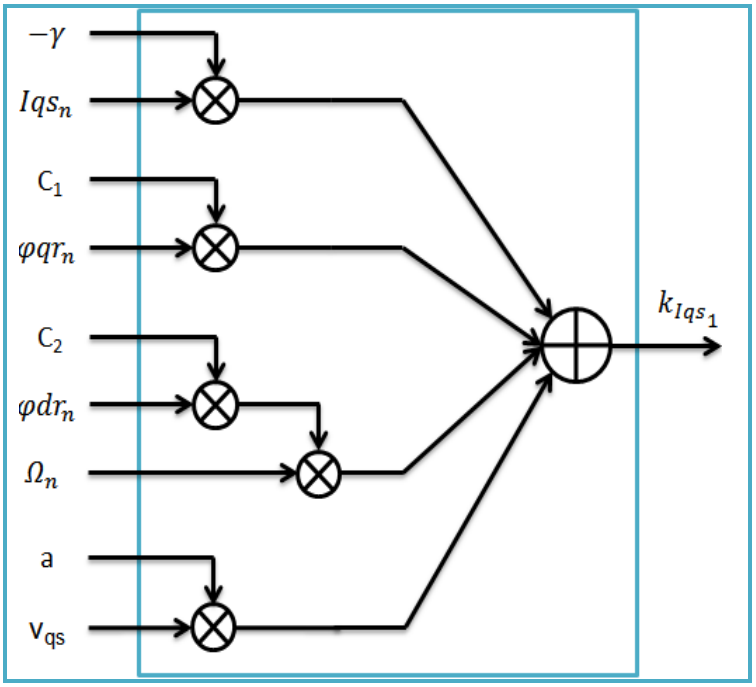

Figure 3.14 : Représentation de k_{Iqs_1}.

La représentation de (k_{Iqs_2}, k_{Iqs_3} et k_{Iqs_4}) ce fait d'après leur équations respectives, de la même manière que k_{Iqs_1}.

- Pour φ_{dr} :

$$\varphi dr_{n+1} = \varphi dr_n + \frac{h}{6}\left(k_{\varphi dr_1} + 2k_{\varphi dr_2} + 2k_{\varphi dr_3} + k_{\varphi dr_4}\right)$$
$$= \varphi dr_n + h(pente\_\varphi dr)$$

La figure 3.15 représente le calcul de « $pente\_\varphi dr$ ».

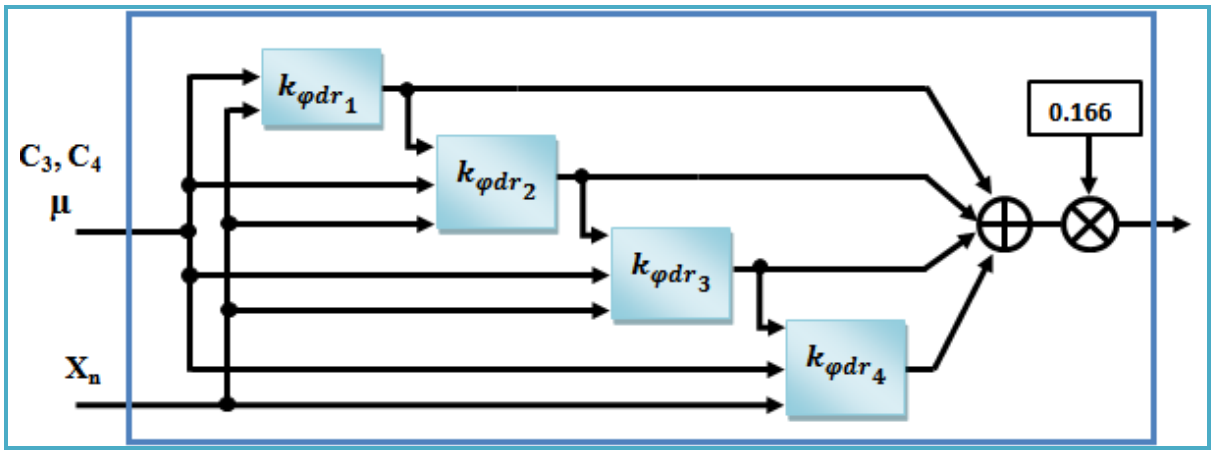

Figure 3.15 : Représentation de pente_φdr.

Avec :

$$\begin{cases} k_{\varphi dr_1} = -\alpha\varphi dr_n + C_3 Ids_n - C_4\Omega_n\varphi qr_n \\ k_{\varphi dr_2} = -\alpha(\varphi dr_n + \frac{k_{\varphi dr_1}}{2}) + C_3 Ids_n - C_4\Omega_n\varphi qr_n \\ k_{\varphi dr_3} = -\alpha(\varphi dr_n + \frac{k_{\varphi dr_2}}{2}) + C_3 Ids_n - C_4\Omega_n\varphi qr_n \\ k_{\varphi dr_4} = -\alpha(\varphi dr_n + \frac{k_{\varphi dr_3}}{2}) + C_3 Ids_n - C_4\Omega_n\varphi qr_n \end{cases}$$

La figure 3.16 représente le calcul de « $k_{\varphi dr_1}$ ».

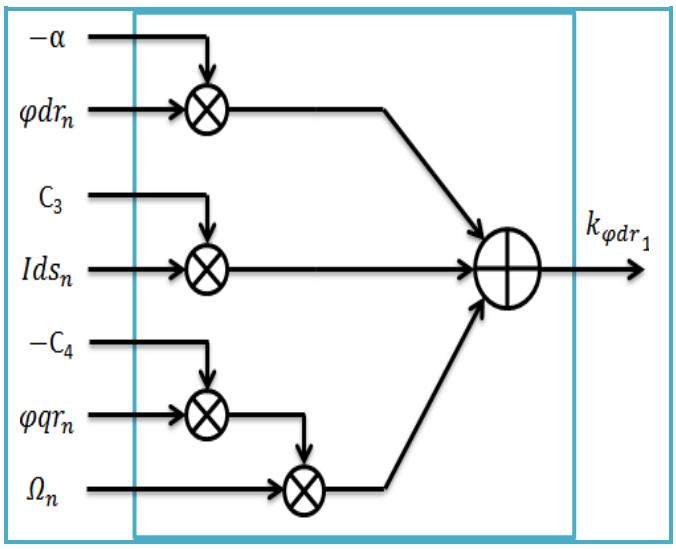

Figure 3.16 : Représentation de $k_{\varphi dr_1}$.

La représentation de ($k_{\varphi dr_2}$, $k_{\varphi dr_3}$ et $k_{\varphi dr_4}$) ce fait d'après leur équations respectives, de la même manière que $k_{\varphi dr_1}$.

❖ Pour φ_{qr}:

$$\varphi qr_{n+1} = \varphi qr_n + \frac{h}{6}(k_{\varphi qr_1} + 2k_{\varphi qr_2} + 2k_{\varphi qr_3} + k_{\varphi qr_4})$$
$$= \varphi qr_n + h(pente\_\varphi qr)$$

La figure 3.17 représente le calcul de « $pente\_\varphi qr$ ».

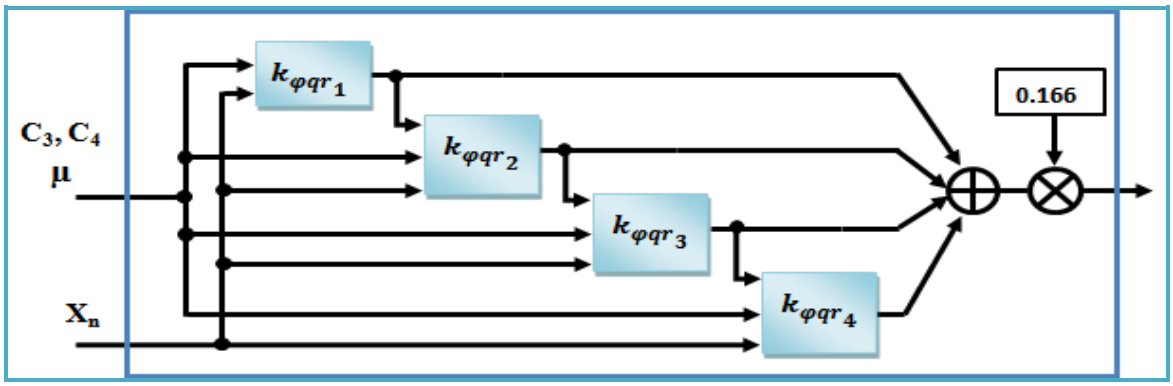

Figure 3.17 : Représentation de pente_φqr.

Avec :

$$\begin{cases} k_{\varphi qr_1} = -\alpha\varphi qr_n + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \\ k_{\varphi qr_2} = -\alpha(\varphi qr_n + \frac{k_{\varphi qr_1}}{2}) + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \\ k_{\varphi qr_3} = -\alpha(\varphi qr_n + \frac{k_{\varphi qr_2}}{2}) + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \\ k_{\varphi qr_4} = -\alpha(\varphi qr_n + \frac{k_{\varphi qr_3}}{2}) + C_3 Iqs_n + C_4\Omega_n\varphi dr_n \end{cases}$$

La figure 3.18 représente le calcul de « $k_{\varphi qr_1}$ ».

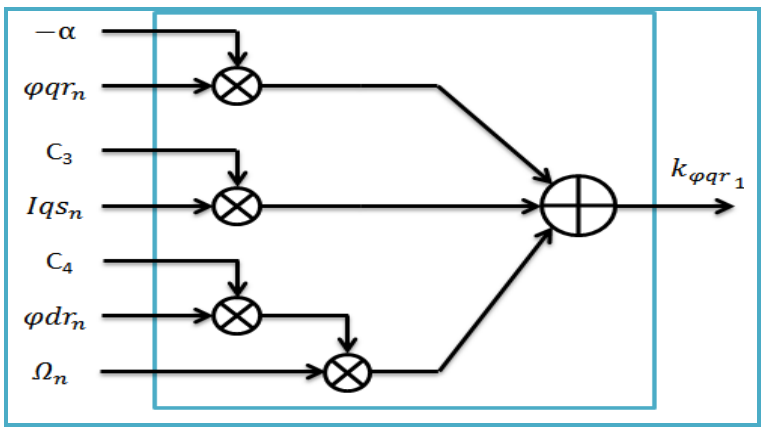

Figure 3.18 : Représentation de $k_{\varphi qr_1}$.

La représentation de ($k_{\varphi qr_2}$, $k_{\varphi qr_3}$ et $k_{\varphi qr_4}$) ce fait d'après leur équations respectives, de la même manière que $k_{\varphi qr_1}$.

❖ Pour Ω:

$$\Omega_{n+1} = \Omega_n + \frac{h}{6}(k_{\Omega_1} + 2k_{\Omega_2} + 2k_{\Omega_3} + k_{\Omega_4})$$
$$= \Omega_n + h(pente\_\Omega)$$

La figure 3.19 représente le calcul de « $pente\_\Omega$ ».

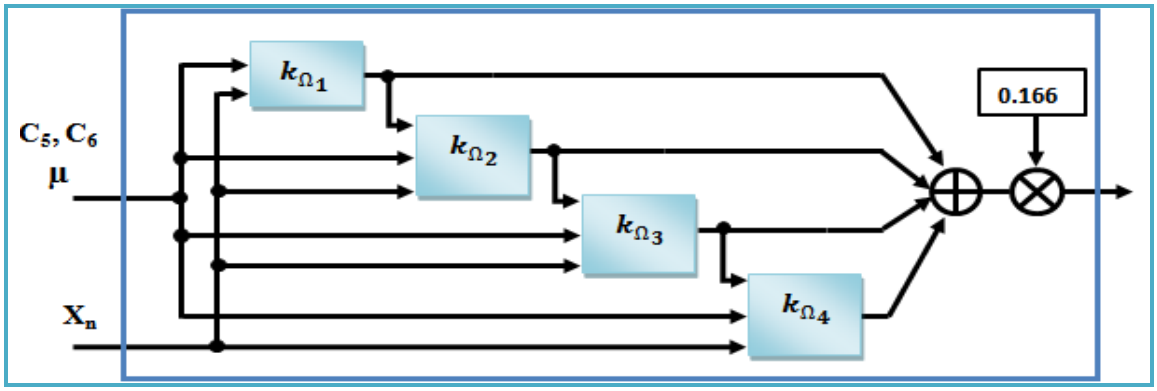

Figure 3.19 : Représentation de pente_Ω.

Avec :

$$\begin{cases} k_{\Omega_1} = -C_6\Omega_n + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \\ k_{\Omega_2} = -C_6(\Omega_n + \frac{k_{\Omega_1}}{2}) + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \\ k_{\Omega_3} = -C_6(\Omega_n + \frac{k_{\Omega_2}}{2}) + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \\ k_{\Omega_4} = -C_6(\Omega_n + \frac{k_{\Omega_3}}{2}) + \mu(\varphi dr_n Iqs_n - \varphi qr_n Ids_n) - C_5 \end{cases}$$

La figure 3.20 représente le calcul de « k_{Ω_1} ».

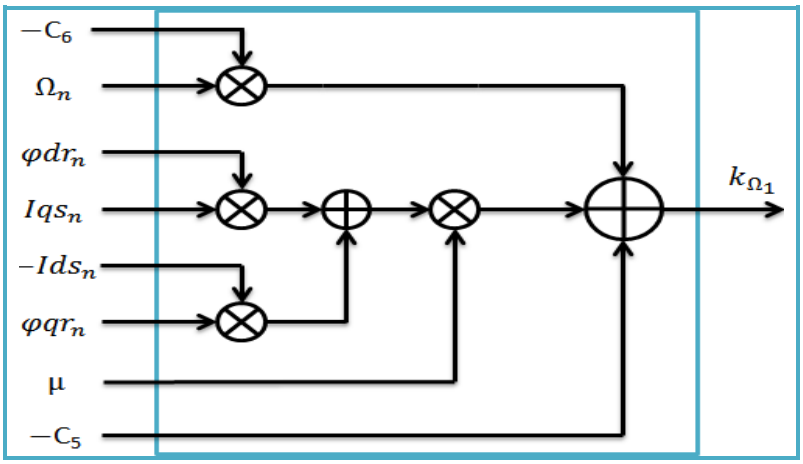

Figure 3.20 : Représentation de k_{Ω_1}.

La représentation de (k_{Ω_2}, k_{Ω_3} et k_{Ω_4}) ce fait d'après leur équations respectives, de la même manière que k_{Ω_1}.

3.3.8.2.Module de la Mémorisation et la mise à jour de X_n :

Ce module est responsable de la mise à jour et de la mémorisation de X_n, il est composé de bascules D, ainsi d'additionneurs.

Les figure 3.21, 3.22 représentent les synoptiques du bloc Mémorisation et du bloc de la mise à jour de X_n.

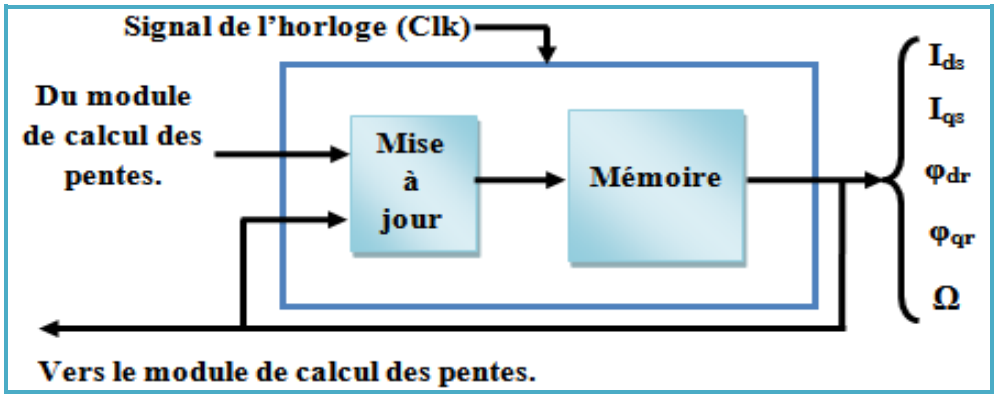

Figure 3.21 : Module de la Mémorisation et la mise à jour de X_n

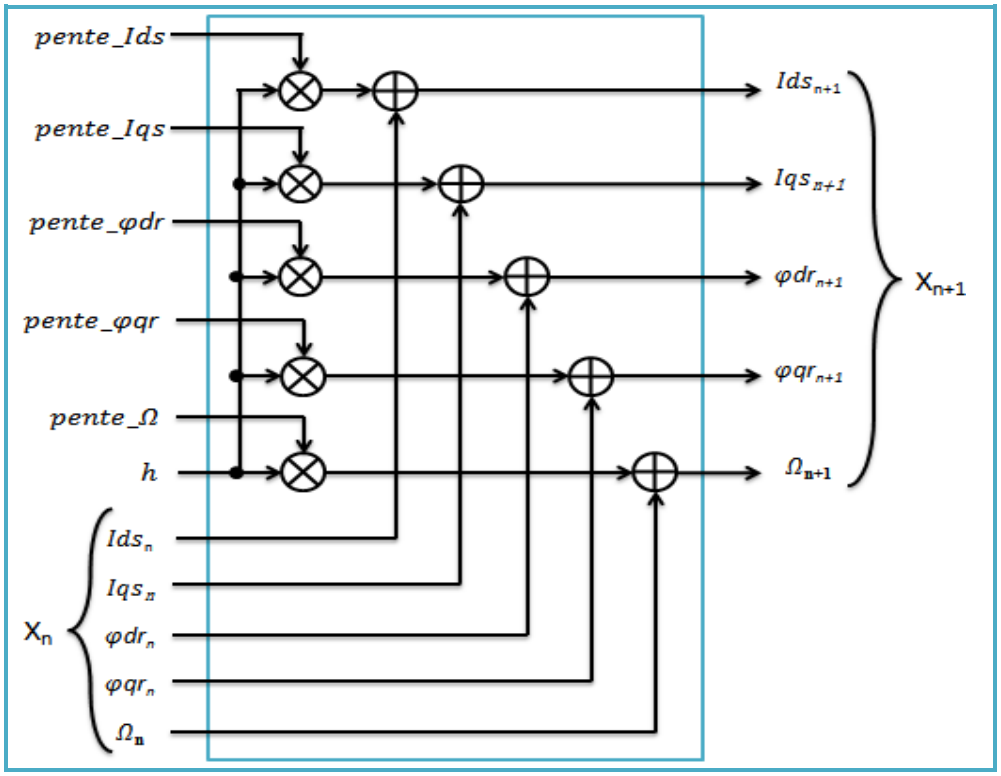

Figure 3.22 : Bloc de la mise à jour.

Le vecteur d'état X_{n+1} va être mémorisé dans le bloc mémoire, la sortie de ce dernier est relié à la sortie.

3.4.Présentation des résultats :

❖ Résultat de simulation avec MODELSIM:

Les simulations comportementale et fonctionnelle de la description ont été effectuées à l'aide du logiciel **« ModelSim Simulator »** de **MentorGraphics.**

La figure 3.23 représente les résultats obtenus après une simulation fonctionnelle de l'architecture proposée pour le modèle de la MAS dans le repère diphasé (dq).

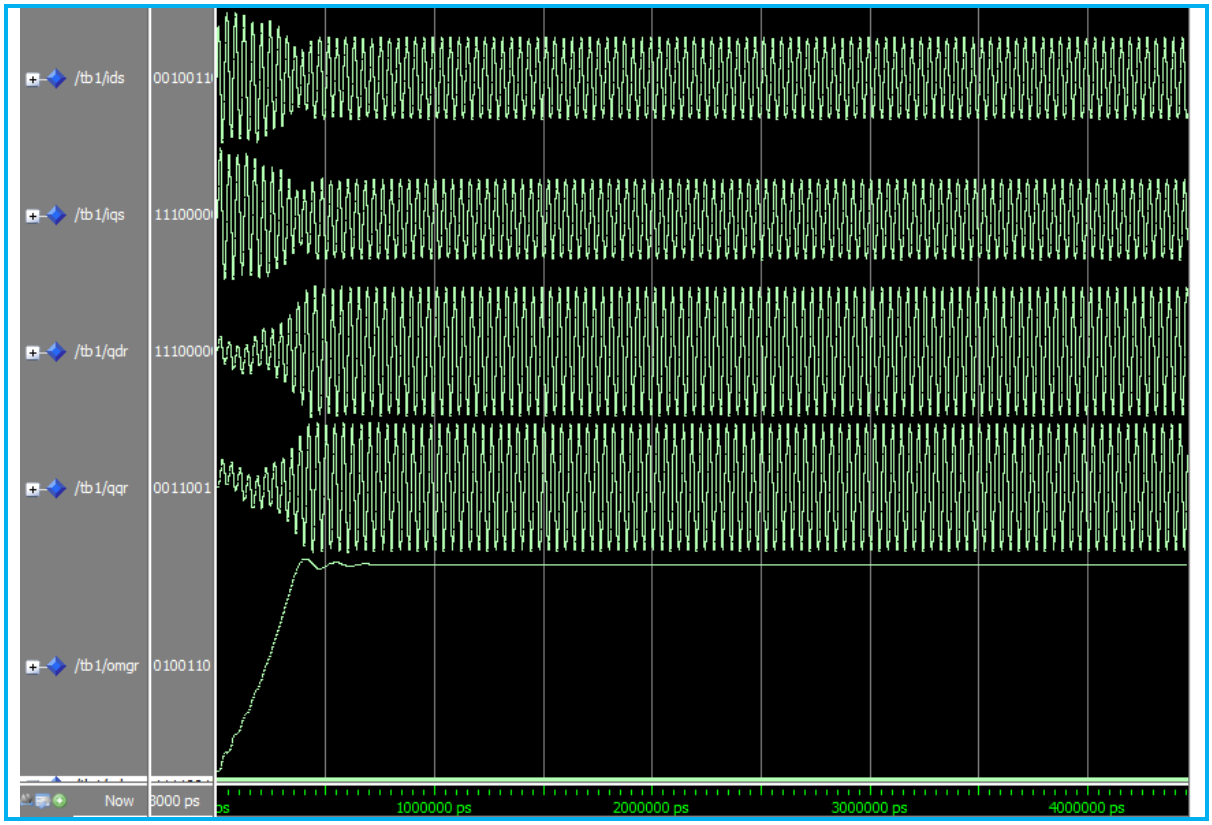

Figure 3.23 : Simulation fonctionnelle du modèle de la MAS avec MODELSIM

Les deux premières courbes représentent les courants statoriques Ids et Iqs. Les deux suivantes courbes représentent les flux rotoriques φdr et φqr. Et la dernière courbe représente la vitesse mécanique.

Les signaux obtenus par cette simulation ont des formes similaires aux signaux obtenus avec le logiciel MATLAB. Pour le vérifier, l'utilisation des outils du **SystemGenerator** permet de comparer les résultats obtenus par l'architecture proposée avec ceux qui proviennent de la simulation du **Matlab**.

❖ Résultats de simulation avec SystemGenerator :

Avec cette simulation, on peut implémenter une description HDL dans un bloc (appelé « black box » ou boite noir) dans l'environnement **Matlab/Simulink** pour qu'on puisse lui appliquer des signaux d'entré et lire ou visualiser ses signaux de sortie. Cette technique permet de comparer les résultats obtenus par ce bloc (en précision finie) avec ceux provenant d'une simulation **Matlab/Simulink** (en précision infinie) pour vérifier la bonne fonctionnalité de la description.

L'utilisation des blocs du **SystemGénérator** développé par **Xilinx** avec des blocs ordinaires du Simulink facilite l'établissement d'une relation simple entre le Simulateur HDL (**ModelSim** ou **ISE Simulator**) et le Simulink.

La figure 3.24 représente l'implémentation de la description VHDL du model de la MAS dans un model Simulink en vue de sa simulation.

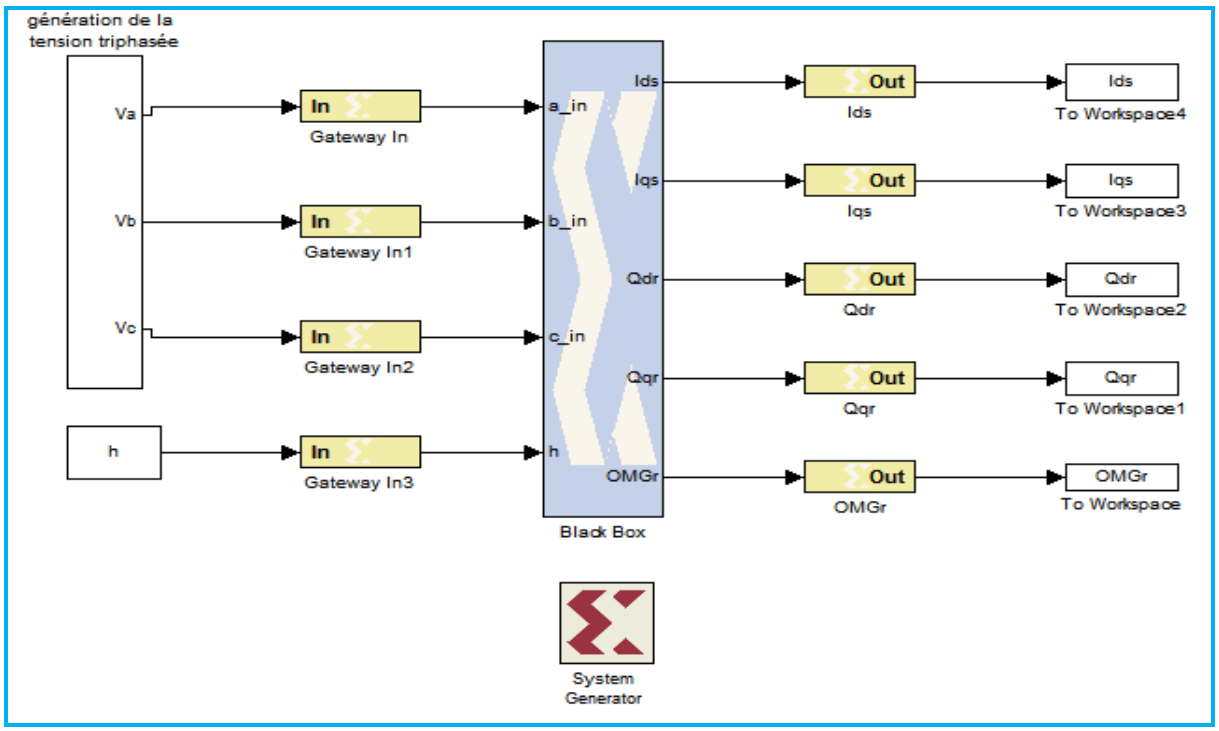

Figure 3.24 : Schéma bloc de simulation avec SystemGenerator

Les figures 3.25, 3.27, 3.29, 3.31, 3.33 représentent les résultats obtenus par la simulation avec SystemGenerator.

La figure 3.25 représente la superposition des courbes du courent statorique Ids issues du logiciel MATLAB et celle du circuit FPGA (black box).

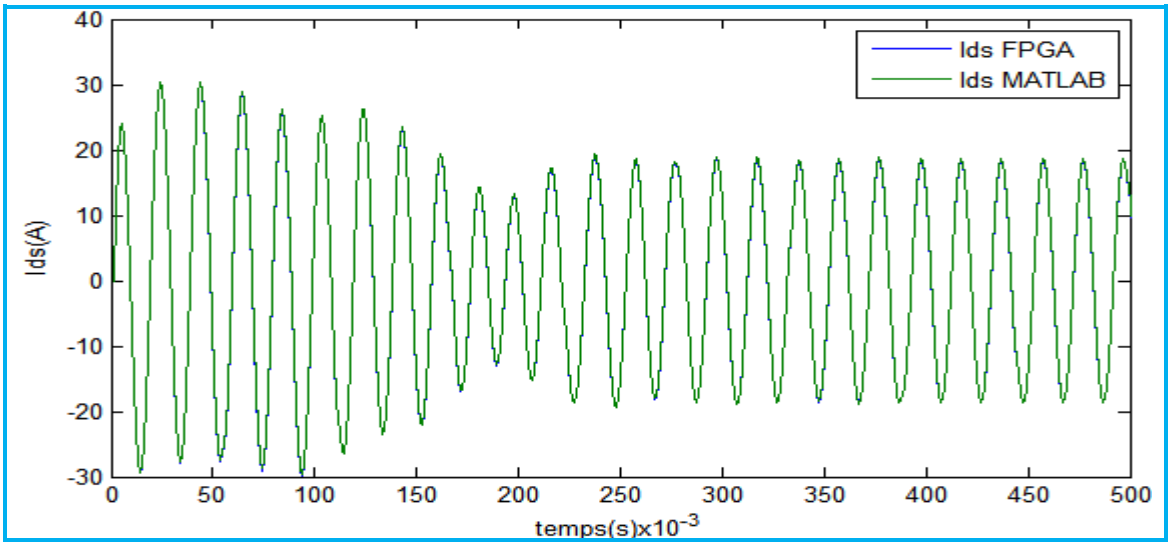

Figure 3.25 : superposition des courbes du courent statorique Ids.

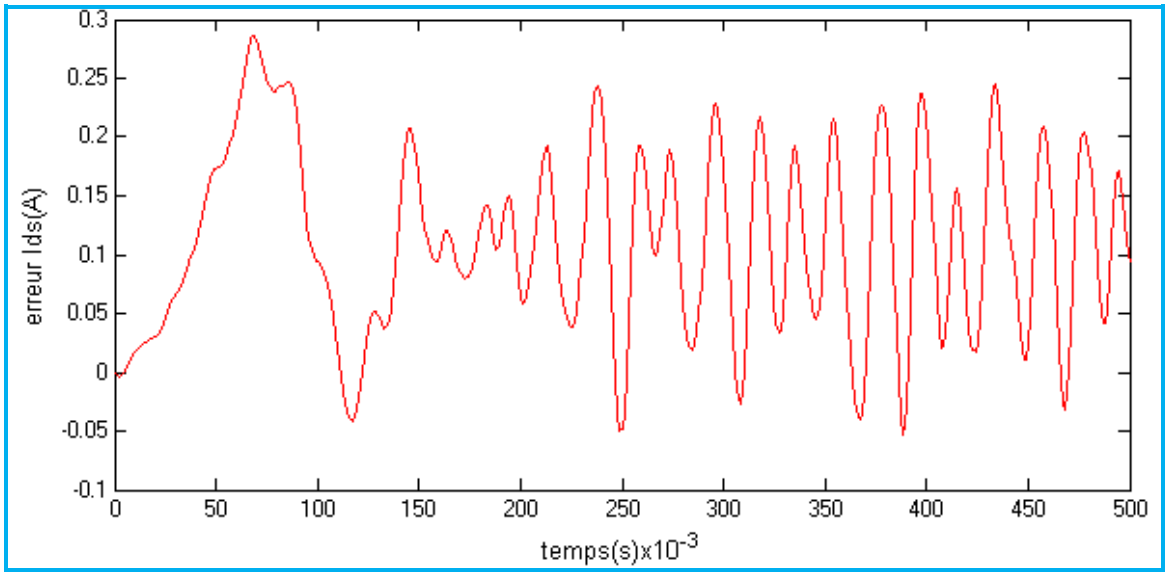

Figure 3.26 : l'erreur entre les deux courbes du courent statorique Ids

La figure 3.26 représente l'erreur entre les deux courbes, on constate qu'elle est entre -0.1 et 0.3, ce qui représente une erreur inférieur 0.66%.

La figure 3.27 représente la superposition des courbes du courent statorique Iqs issues du logiciel MATLAB et celle du circuit FPGA (black box).

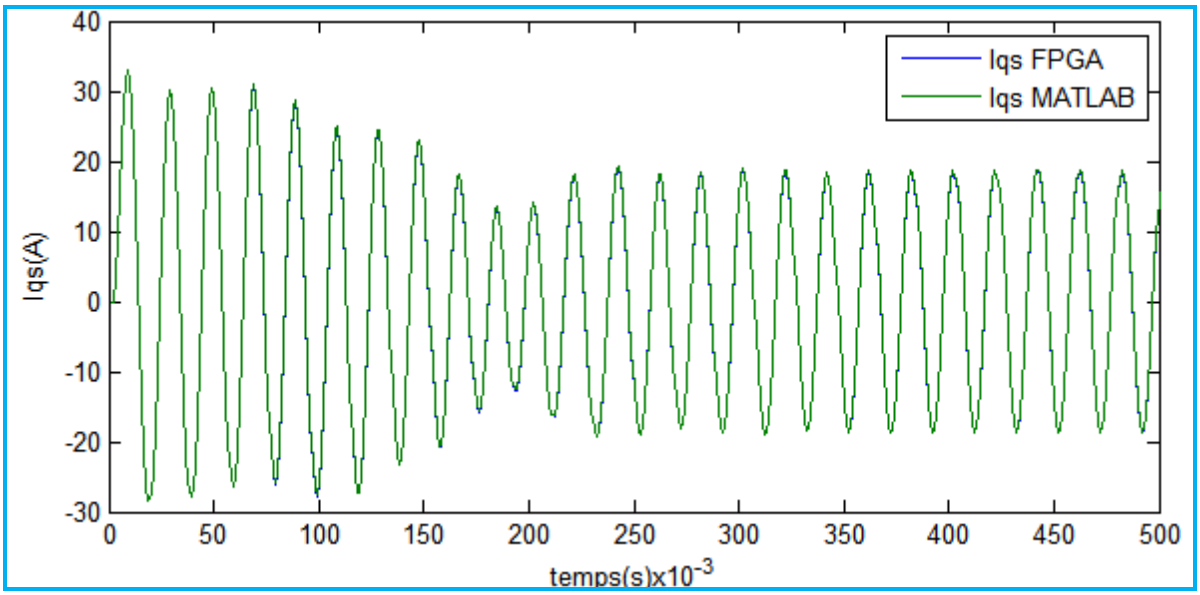

Figure 3.27 : superposition des courbes du courent statorique Iqs

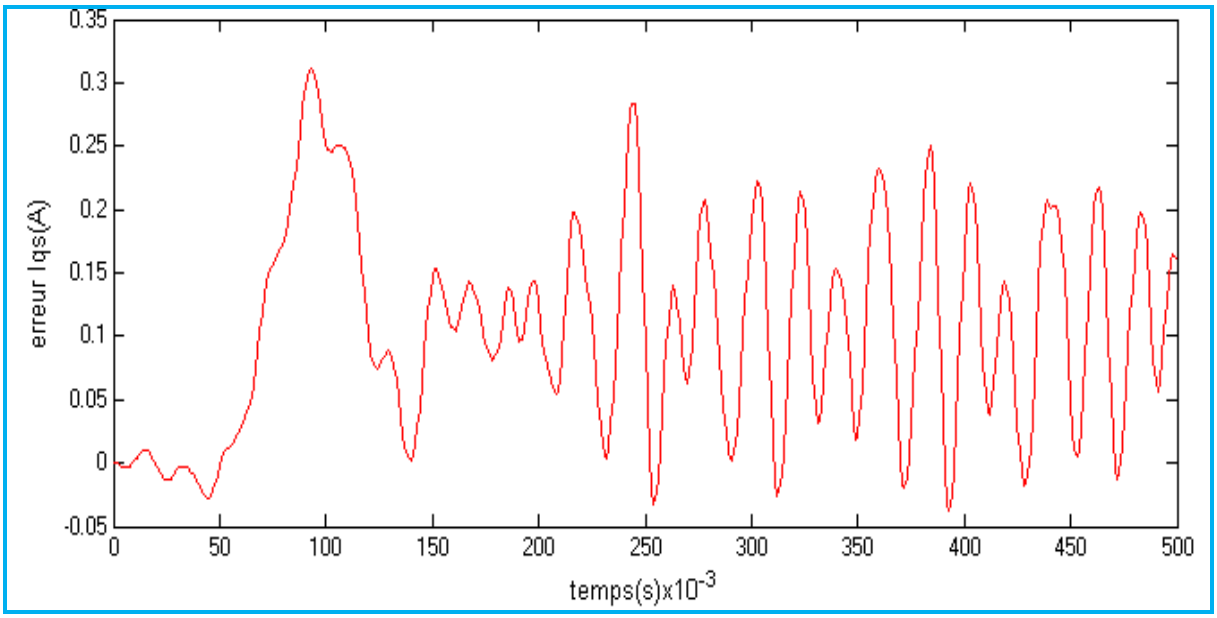

Figure 3.28 : l'erreur entre les deux courbes du courent statorique Iqs

La figure 3.28 représente l'erreur entre les deux courbes, on constate qu'elle est entre -0.05 et 0.35, ce qui représente une erreur inférieur 0.66%.

La figure 3.29 représente la superposition des courbes du flux rotorique Qdr issues du logiciel MATLAB et celle du circuit FPGA (black box).

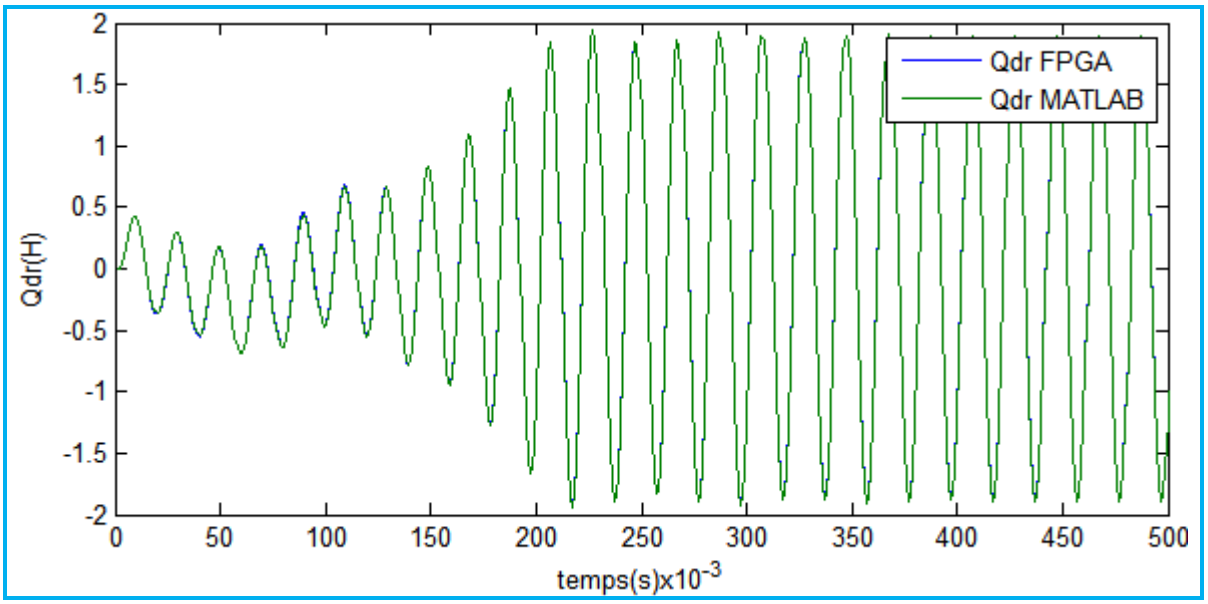

Figure 3.29 : superposition des courbes du flux rotorique Qdr

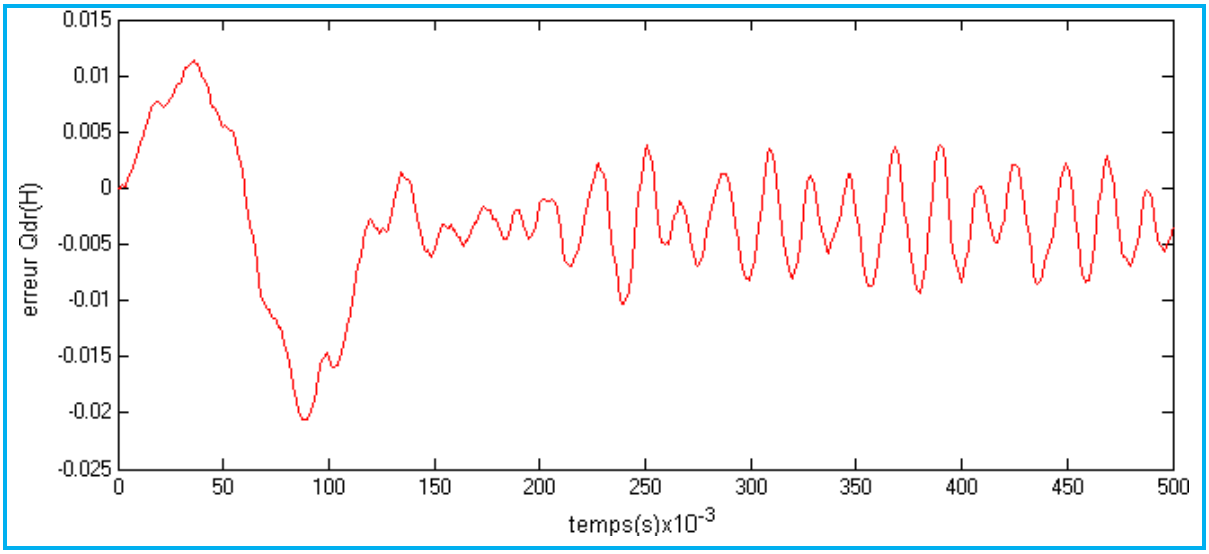

Figure 3.30 : l'erreur entre les deux courbes du flux rotorique Qdr

La figure 3.30 représente l'erreur entre les deux courbes, on constate qu'elle est entre -0.025 et 0.015, ce qui représente une erreur inférieur 1%.

La figure 3.31 représente la superposition des courbes du flux rotorique Qqr issues du logiciel MATLAB et celle du circuit FPGA (black box).

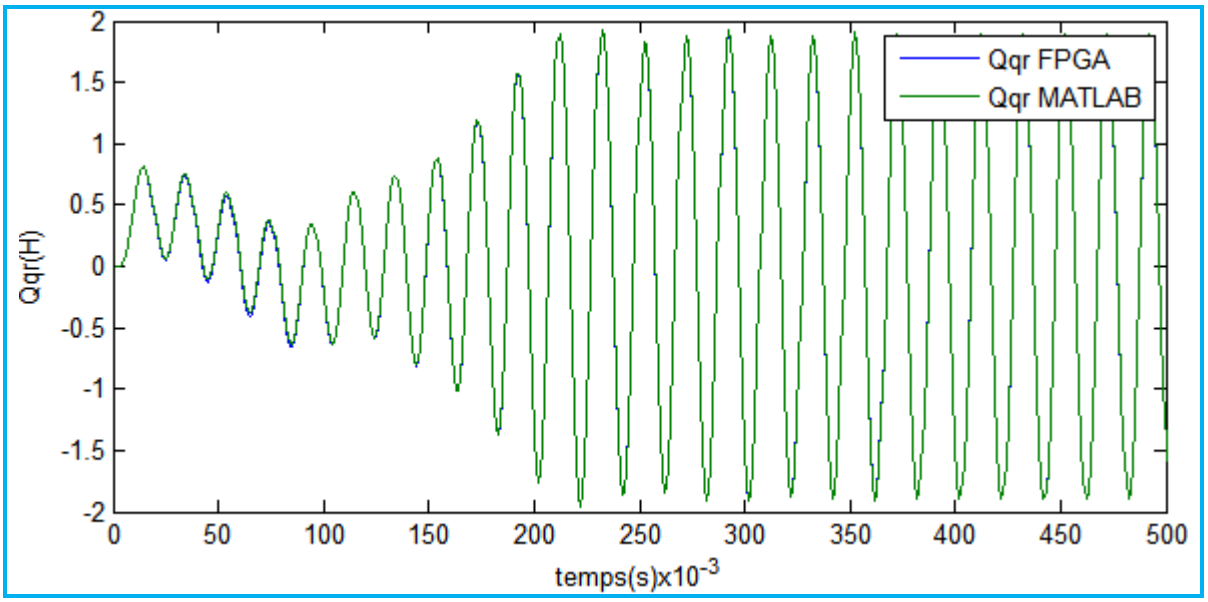

Figure 3.31 : superposition des courbes du flux rotorique Qqr

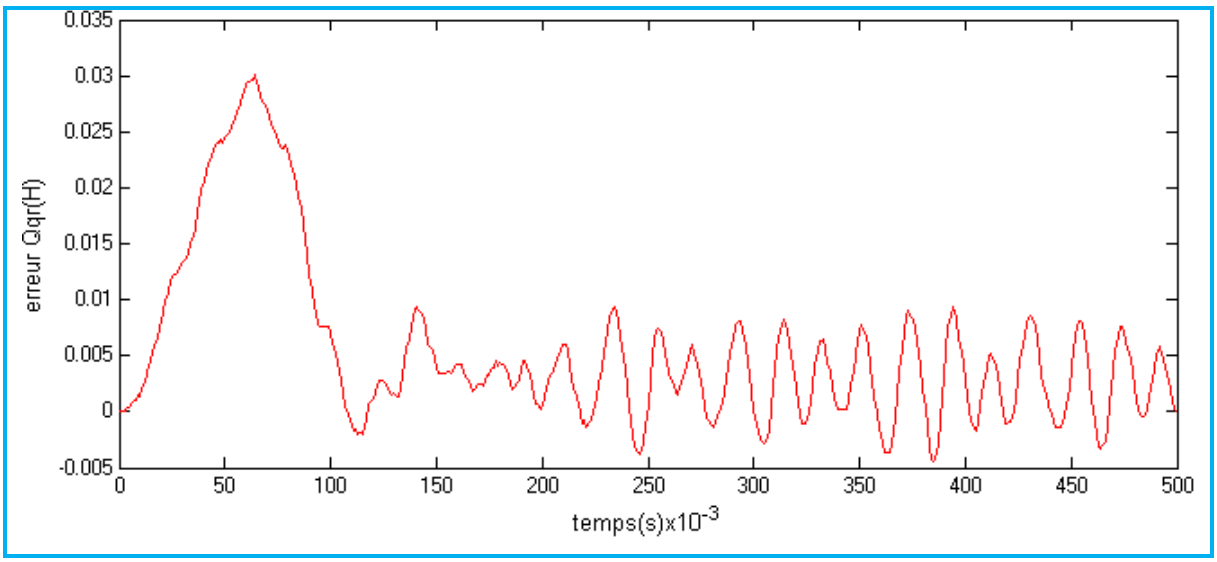

Figure 3.32 : l'erreur entre les deux courbes du flux rotorique Qqr

La figure 3.32 représente l'erreur entre les deux courbes, on constate qu'elle est entre -0.005 et 0.035, ce qui représente une erreur inférieur 1%.

La figure 3.33 représente la superposition des courbes de la vitesse Ω issues du logiciel MATLAB et celle du circuit FPGA (black box).

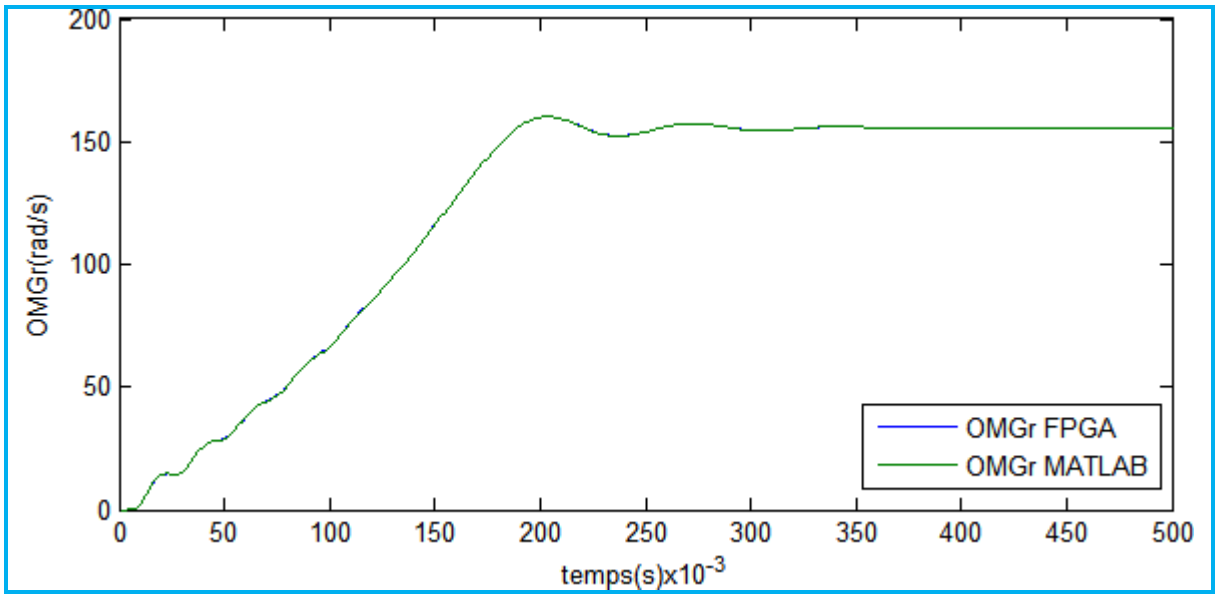

Figure 3.33 : superposition des courbes de la vitesse Ω

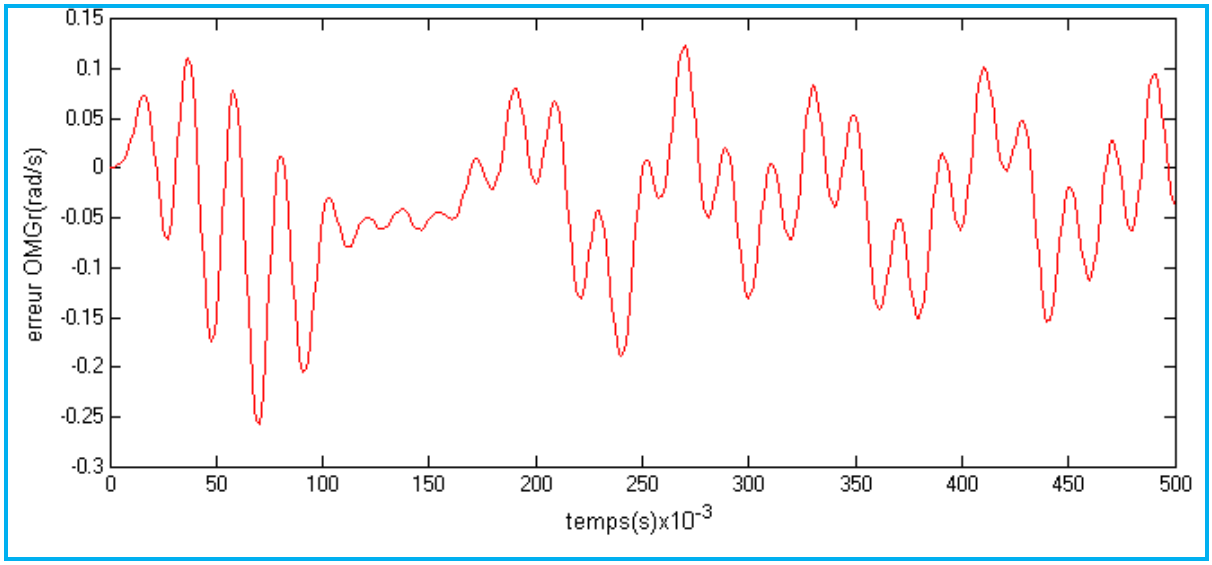

Figure 3.34 : l'erreur entre les deux courbes de la vitesse Ω

La figure 3.34 représente l'erreur entre les deux courbes, on constate qu'elle est entre -0.3 et 0.15, ce qui représente une erreur inférieur 0.3%.

❖ Résultats de synthèse :

L'outil ISE de Xilinx peut être utilisé pendant toutes les étapes de la procédure de développement d'un projet FPGA. Son interface est représentée par la figure suivante :

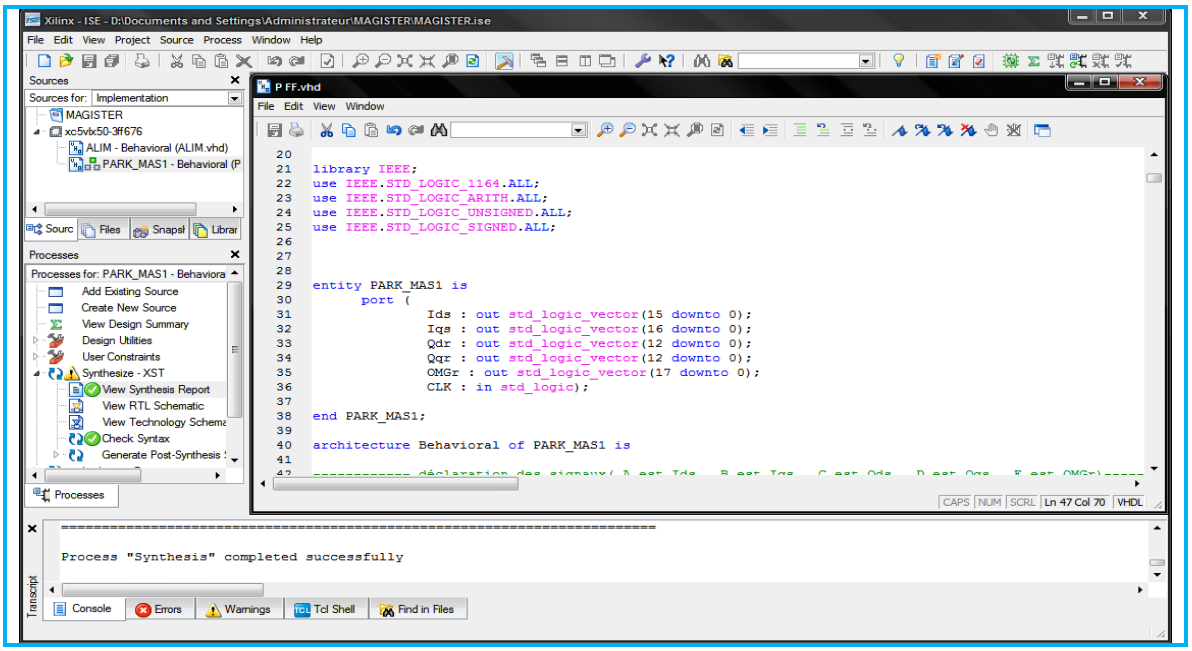

Figure 3.35 : Interface du logiciel de XILINX ISE

Après la synthèse, on peut visualiser les schémas RTL des différents blocs de notre architecture.

La figure 3.36 représente le schéma RTL du bloc général. A l'entrée, on trouve le signal d'horloge (CLK) ainsi que les tensions triphasées. A sa sortie on trouve les courants statoriques (Ids et Iqs), les flux rotorique (Qdr et Qqr) et la vitesse mécanique (Ω).

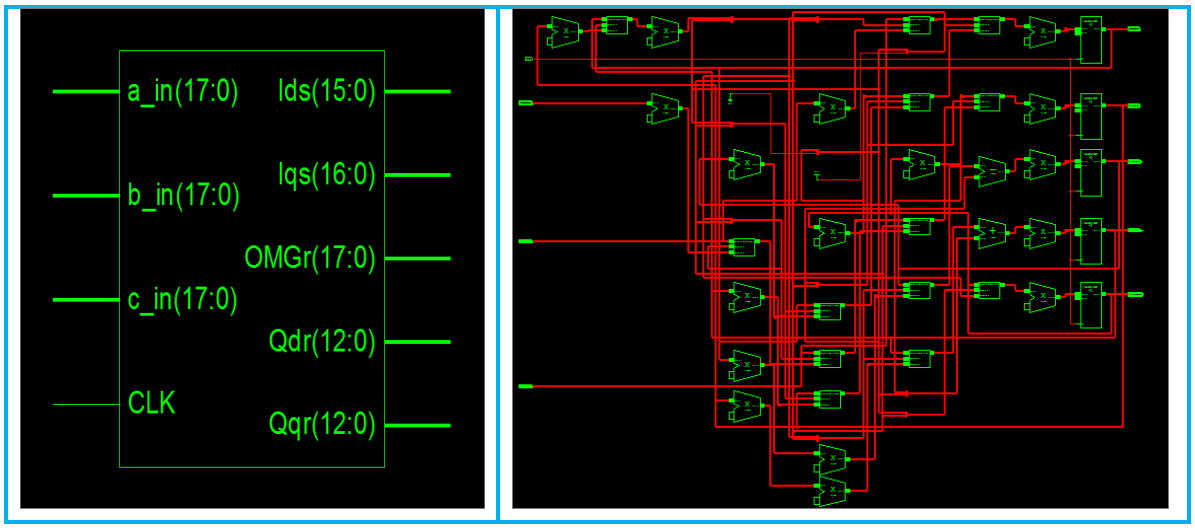

Figure 3.36 : schéma RTL du bloc général

Ce bloc est constitué des différents modules étudiés dans les paragraphes précédents, qui sont :

- ❖ Module de la génération de la tension Vdqs.
- ❖ Module Runge-Kutta d'ordre 4.

Concernant le Module de la génération de la tension triphasée voici sa représentation du schéma RTL.

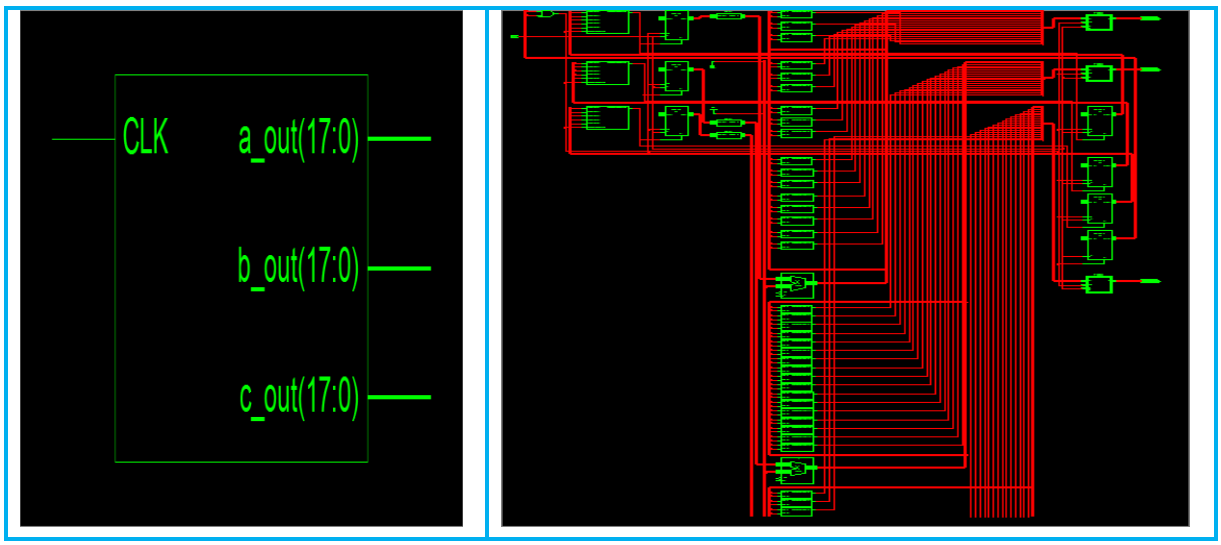

Figure 3.37 : schéma RTL du bloc Module de la génération de la tension triphasée

L'outil ISE nous a donné aussi un rapport de synthèse détaillé. Ce rapport peut être résumé par les deux figures suivantes:

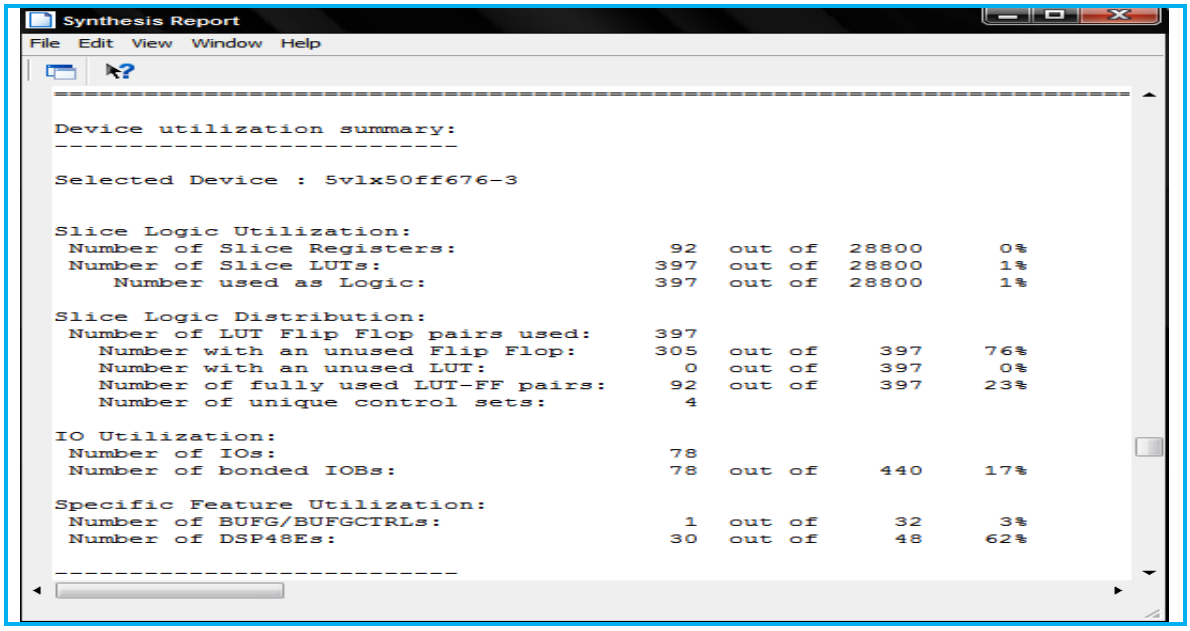

Figure 3.38 : Rapport de synthèse – Utilisation des ressources

A partir du rapport de synthèse (figure 3.38), on remarque qu'on n'a pas utilisé beaucoup de slices. 360 slices ont été complètement exploitées, 92 slices ont été utilisées comme des registres et 397 ont été utilisés comme des LUT. On a utilisé également 17% des blocs d'entrée/sortie, 3% des Buffers et 62% des slices DSP.

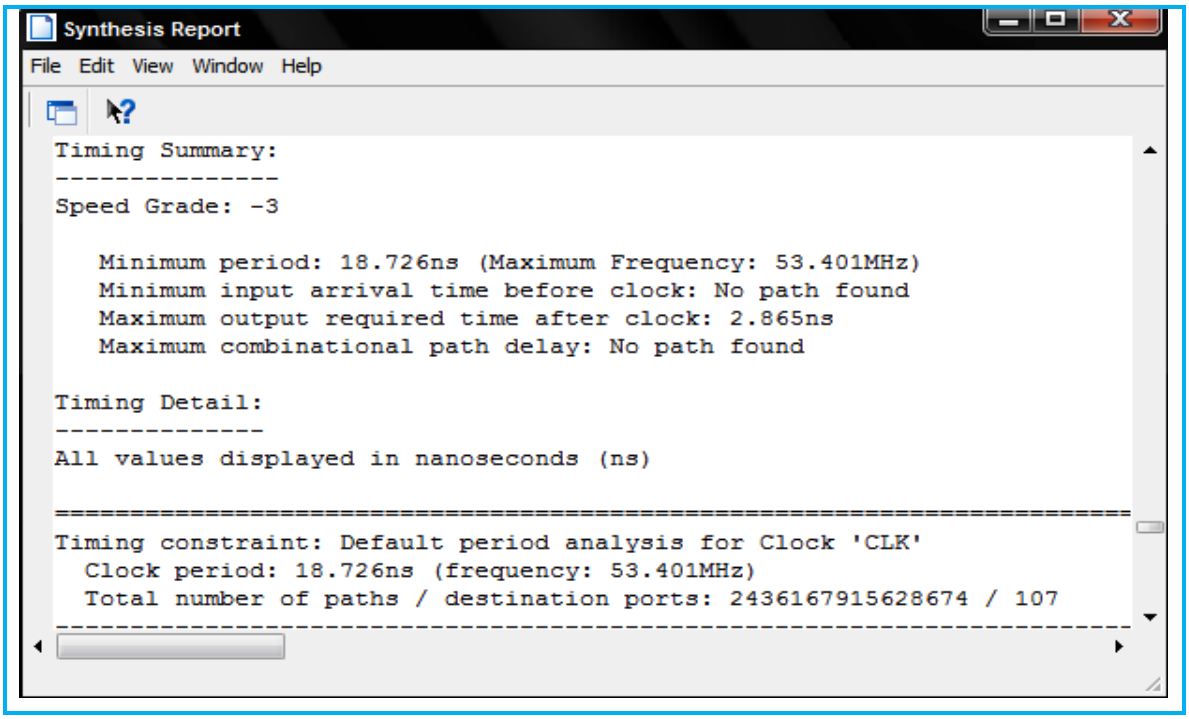

Figure 3.39 : Rapport de synthèse – Rapport temporel

La figure 3.39 montre que la fréquence maximale du fonctionnement de notre architecture est de 53,401 MHz, ce qui représente une période minimale de 18,726 ns. Ce temps représente la durée minimale de propagation des signaux entre l'entrée et la sortie.

Étant que l'objectif de ce mémoire, est de mettre en évidence la contribution des circuits reconfigurables FPGA dans l'accélération de simulation des systèmes dynamiques par rapport aux simulateurs software comme MATLAB. On doit alors comparé le bilan temporel des deux simulations.

D'après la simulation réalisée avec le logiciel MATLAB, on constate que la durée nécessaire pour effectuer une itération est de 0,2277 ms. En revanche, d'après la simulation réalisée avec le logiciel XILINX, la durée nécessaire pour effectuer une

itération avec un circuit FPGA est de 18,726 ns. De ce fait, on peut dire qu'avec un circuit FPGA on a gagné environ 12160 fois plus en rapidité.

3.5.Conclusion :

Dans ce chapitre nous avons en premier lieu résolut le modèle mathématique de la MAS représenté par un système d'équations différentielle fortement couplé et non linéaire, avec la méthode itérative de Runge-Kutta d'ordre 4. Ensuite, en utilisant le logiciel MATLAB, nous avant simuler le comportement du model de la MAS, ainsi prélever les résultats acquis.

Puis, à l'aide du langage de description VHDL nous avons créé un programme du modèle de la MAS avec le logiciel XILINX, et vérifier sa fonctionnalité. Puis, en utilisant le logiciel MODELSIM, on a effectué une simulation du comportement de son architecture.

Ensuite, on a utilisé les outils du SystemGénérator, et comparé les résultats obtenus à l'aide du logiciel MATLAB (précision infini) avec ceux acquis de l'architecture proposé à l'aide du logiciel XILINX (précision fini). On a constaté que les résultats sont quasi identiques, et que la différence entre eux est due aux troncatures effectuées lors des opérations de multiplications (rappelons que l'on utilise que les multiplieurs 18x18 contenues dans le circuit FPGA).

Finalement, à travers le bilan temporel effectué dans chaque simulation, on peut déduire que la simulation en utilisant un circuit FPGA (solution matériel) est beaucoup plus avantageuse par rapport à l'utilisation d'un simulateur « offline » comme MATLAB (solution logiciel).

CONCLUSION GENERALE

Actuellement, La simulation logicielle est de loin la plus utilisé dans le domaine de l'électrique, mais les solutions numériques matérielles telles que les FPGA peuvent être aussi considérées comme des solutions appropriées pour améliorer les performances de contrôle. Le parallélisme inhérent de ces nouvelles solutions numériques, ainsi que leurs grandes capacités de calcul font que les délais de temps de calcul sont négligeables en dépit de la complexité des algorithmes à implémenter. Par ailleurs, par rapport aux solutions logicielles, les solutions matérielles offrent au concepteur un accès à la partie architecture matérielle, ce qui est un atout précieux pour un électronicien.

Dans ce mémoire nous avons analysé et mis en évidence la contribution des circuits reconfigurables en particulier les circuits FPGA dans l'accélération de simulation des systèmes dynamiques comparée aux simulateurs "offline" ; comme (MATLAB/SIMULINK, SPICE, etc.….) ; ainsi que le développement d'un simulateur dynamique temps réel, en implémentant le modèle de la machine asynchrone sur circuit FPGA.

Pour cela, nous avons résolu le modèle mathématique de la MAS représenté par un système d'équations différentielle, avec la méthode itérative « Runge-Kutta d'ordre 4 ». Ensuite, nous avant simuler le comportement du model de la MAS, ainsi prélever les résultats acquis en utilisant le logiciel MATLAB.

Puis, à l'aide du langage de description VHDL nous avons créé un programme du modèle de la MAS avec le logiciel XILINX. Après, on a effectué une simulation du comportement de son architecture en utilisant le logiciel MODELSIM.

Afin de comparé les résultats obtenus à l'aide du logiciel MATLAB avec ceux acquis de l'architecture proposé à l'aide du logiciel XILINX, on a utilisé les

outils du SystemGenerator. On a constaté que les résultats sont quasi identique, et que la différence entre eux est très négligeable.

Finalement, à travers ce mémoire, on peut déduire que la simulation en utilisant un circuit FPGA (solution matériel) est beaucoup plus intéressante par rapport à l'utilisation d'un simulateur « offline » comme MATLAB (solution logiciel) en termes de rapidité en gagnant un temps considérable pendant la simulation.

APPENDICE A
LISTE DES SYMBOLES

MAS	Machine Asynchrone
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Arrays
ASIC	Application Specific Integer Circuit
$\mathbf{P}$	Nombre de paires de pôles
$\boldsymbol{\theta}$, $\boldsymbol{\omega}$, $\boldsymbol{\Omega}$	Angle, Pulsation et vitesse mécaniques
$\boldsymbol{\theta_r}, \boldsymbol{\omega_r}, \boldsymbol{\Omega_r}$	Angle, Pulsation et vitesse Rotoriques
$\boldsymbol{\theta_s}, \boldsymbol{\omega_s}, \boldsymbol{\Omega_s}$	Angle, Pulsation et vitesse Statoriques
$\boldsymbol{C_{em}}, \boldsymbol{C_r}$	Couple électromagnétique et couple de charge
$\boldsymbol{v_{sa,sb,sc}}, \boldsymbol{v_{ra,rb,rc}}$	Tensions Triphasés Statoriques et Rotoriques
$\boldsymbol{i_{sa,sb,sc}}, \boldsymbol{i_{ra,rb,rc}}$	Courants Triphasés Statoriques et Rotoriques
$\boldsymbol{\varphi_{sa,sb,sc}}, \boldsymbol{\varphi_{ra,rb,rc}}$	Flux Triphasés Statoriques et Rotoriques
$\boldsymbol{v_{ds,qs}}$, $\boldsymbol{v_{dr,qr}}$ $\boldsymbol{v_{dr,qr}}$	Tensions de Park
$\boldsymbol{i_{ds,qs}}$, $\boldsymbol{i_{dr,qr}}$	Courants de Park
$\boldsymbol{\varphi_{ds,qs}}$, $\boldsymbol{\varphi_{dr,qr}}$	Flux de Park
$\boldsymbol{l_s}$ **et** $\boldsymbol{l_r}$	Inductances propres statorique et rotorique.
$\boldsymbol{l_{ss}}$ **et** $\boldsymbol{l_{rr}}$	Inductances mutuelles entre deux phases statoriques ou rotoriques
$\boldsymbol{l_{sr}}$	Inductance mutuelle maximale entre une phase statorique et une phase rotorique
$\boldsymbol{L_s}$, $\boldsymbol{L_r}$	Inductances Cycliques Statorique et Rotorique
$\boldsymbol{L_m}$	Inductance Mutuelle Cyclique entre Stator et Rotor
$[\mathbf{T}]$	Matrice de transformation de Park
$\boldsymbol{R_s}$, $\boldsymbol{R_r}$	Résistances statorique et rotorique.
$\mathbf{J}$	Moment d'inertie de la MAS
$\mathbf{f}$	Frottement Totale de la MAS et de la charge

APPENDICE B
PARAMETRES DE LA MACHINE ASYNCHRONE UTILISÉE

La machine utilisée lors des simulations dans ce mémoire est une machine asynchrone à cage d'écureuil standard. Ses caractéristiques principales sont les suivantes:

Puissance nominale (Pn)	1.5 KW
Tension nominale (Vn)	220/380 V
Courant nominale (In)	6.7/3.7 A
Nombre de pole (P)	2
La vitesse de rotation (Nn)	1450 TR/MN
Fréquence nominale (Fn)	50 HZ

Tableau B.1 : caractéristiques de MAS.

Paramètres électriques :

Résistance statorique (Rs)	4.85 Ω
Résistance rotorique (Rr)	3.81 Ω
Inductance cyclique du stator (Ls)	0.274 H
Inductance cyclique du rotor (Lr)	0.274 H

Tableau B.2 : Paramètres électriques.

Paramètres mécaniques :

Moment de d'inertie du rotor	0.031 $Kg.m^2$
Coefficient de frottement visqueux	0.0114 SI

Tableau B.3 : Paramètres mécaniques.

APPENDICE C
REPRESENTATIONS A VIRGULE FIXE

Une représentation avec un format à virgule fixe est caractérisée par l'association d'un nombre fini de bits. En ce qui suit sont expliquées les différentes manières de représentation d'un nombre à virgule fixe. En effet, il est possible d'avoir 2^n combinaisons différentes en associant les n bits du code binaire d'un nombre à virgule fixe. Cependant, la signification de la combinaison des *n* bits dépend de son interprétation.

C.1. Représentation entière non signée :

La représentation entière non signée interprète chaque code binaire comme un entier positif. Un code binaire à *n* bits ($b_{n-1}b_{n-2}\ldots b_1b_0$) est interprété comme étant un entier égal à :

$$b_{n-1}2^{n-1}+b_{n-2}2^{n-2}+\ldots+b_12+b_0$$

Dans ce cas les entiers qui peuvent être codés avec ces *n* bits appartiennent à l'intervalle $[0 , 2^{n-1}]$.

C.2. Représentation entière signée :

Avec une représentation entière signée, le bit le plus significatif indique le signe de l'entier représenté avec le code binaire. Le bit le plus significatif est appelé dans ce cas bit de signe. Si le bit de signe est '0', alors le code binaire représente un entier positif, tandis qu'un entier négatif possède un bit de signe égal à '1'. Avec une représentation entière signée, un code binaire à *n* bits ($b_{n-1}\, b_{n-2}\ldots b_1b_0$) peut représenter des entiers compris dans l'interval $[(-2)^{n-1}, 2^{n-1}-1]$. Ce code binaire est interprété comme étant un entier égal à :

$$-b_{n-1}2^{n-1}+b_{n-2}2^{n-2}+\ldots+b_12+b_0$$

C.3. Représentation fractionnaire :

Une représentation fractionnaire est utilisée pour coder des réels compris entre -1 et 1. Avec une représentation fractionnaire signée, un code binaire à *n* bits peut représenter 2^n réels équidistants de $2^{-(n-1)}$ et compris dans l'interval

$[\frac{-2^{n-1}}{2^{n-1}} = -1, \frac{-2^{n-1}-1}{2^{n-1}} = 1 - 2^{n-1}]$. Avec ce type de représentation, un code binaire à *n* bits ($b_{n-1}\, b_{n-2} \ldots b_1 b_0$) est interprété comme suit :

$$\frac{-b_{n-1}2^{n-1}+b_{n-2}2^{n-2}+\cdots+b_1 2+b_0}{2^{n-1}} = -b_{n-1}+b_{n-2}2^{-1}+\cdots+b_1 2^{-(n-2)}+b_0 2^{-(n-1)}$$

Cette représentation est référencée comme étant un format *s*[n/*Q*(n-1)]. Dans ce cas, *s* indique qu'il s'agit d'une représentation signée, *n* représente le nombre de bits total et l'indice (*n*-1) de *Q* représente le nombre de bits après la virgule.

C.5. Représentation a virgule fixe :

A partir des représentations mentionnées dans les paragraphes précédents, il est possible de considérer une représentation plus généralisée donnée par la figure suivante. Cette représentation est divisée en deux parties : une partie entière et une partie fractionnaire. Ce type de représentation est référencé par le format *s* [(*m*+*n*)/*Qn*] lorsqu'il s'agit d'une représentation signée et par le format *u* [(*m*+*n*)/*Qn*] lorsqu'il s'agit d'une représentation non signée. Ce type de format indique que le nombre de bits total est égal à (*m*+*n*), où les *m* bits les plus significatifs représentent la partie entière tandis que les *n* bits restants représentent la partie fractionnaire.

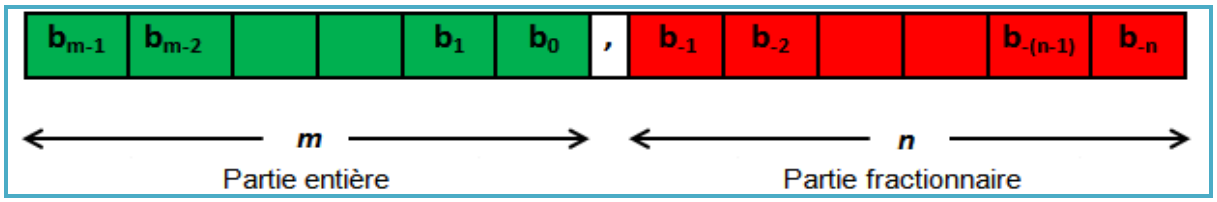

Figure C.1: Représentation binaires des données.

Si le bit le plus significatif *bm*-1 n'est pas interprété comme étant un bit de signe (représentation non signée), le format *u* [(*m*+*n*)/*Qn*] peut représenter $2^{(m+n)}$ réels équidistants de 2^{-n} et compris dans l'interval [0, 2^m -2^{-n}]. Avec ce type de représentation, un code binaire à (*m*+*n*) bits est interprété comme suit :

$$b_{m-1}2^{m-1} + b_{m-2}2^{m-2} + \ldots + b_1 2 + b_0 + b_{-1}2^{-1} + b_{-2}2^{-2} + \ldots + b_{-(n-1)}2^{-(n-1)} + b_{-n}2^{-n}$$

Par exemple, avec un format 11/*Q*8 non signé, le code binaire du réel 2.75 est "01011000000".

APPENDICE D
PRESENTATION DU LANGAGE DE DESCRIPTION VHDL

Le langage utiliser dans la description du circuit dans notre travail est le VHDL “(**V**ery High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage”.
Ce langage a été développé dans les années 80 par le DOD (Département of Défense) des Etats-Unis ; l’objectif était de disposer d’un langage commun avec les fournisseurs pour décrire les circuits complexes [21].

Auparavant pour décrire le fonctionnement d’un circuit électronique programmable, les techniciens et les ingénieurs utilisaient des langages de bas niveau ou plus simplement un outil de saisie de schémas. Actuellement la densité de fonctions logiques (portes et bascules) intégrées dans les PLD (plusieurs milliers voire millions de portes) est telle qu’il n’est plus possible d’utiliser les outils d’hier pour développer les circuits d’aujourd’hui.

Les sociétés de développement et les ingénieurs ont voulu s’affranchir des contraintes technologiques des circuits. Ils ont donc créé des langages dits de haut niveau à savoir VHDL et VERILOG. Ces deux langages font abstraction des contraintes technologies des circuits PLD. Ils permettent au code écrit d’être portable, c'est-à-dire qu’une description écrite pour un circuit peut être facilement utilisée pour un autre circuit.

En 1987, une première version du langage est standardisée par l'IEEE (Institute of Electrical and Electronics Engineers) sous la dénomination IEEE Std. 1076-1987 (VHDL 87). Une évolution du VHDL est normalisée en 1993 (IEEE Std. 1076-1993 ou VHDL-93) ; cette nouvelle version supprime quelques ambiguïtés de la version 87, et surtout met à disposition de nouvelles commandes. Actuellement, tous les outils dignes de ce nom supportent le VHDL-93. En 1997, de nouvelles libraires ont été normalisées de manière à ajouter des fonctions pour la synthèse (conception) de circuits logiques programmables PLD, et plus particulièrement les FPGA [11].

D.1. Structure d'une description VHDL:

La structure typique d'une description VHDL est composée de 3 parties indissociables qui sont : la bibliothèque, l'entité et l'architecture.

- la bibliothèque : Pour augmenter la richesse fonctionnelle du langage, VHDL utilise les objets de bibliothèques préalablement compilées permettant d'introduire de nouveaux types et fonctions.
- L'entité : L'entité (ENTITY) est vue comme une boite noire avec des entrées et des sorties caractérisées par des paramètres. Elle représente une vue externe de la description.
- L'architecture : L'architecture (ARCHITECTURE) contient les instructions VHDL permettant de décrire et de réaliser le fonctionnement attendu. Elle représente la structure interne de la description.

D.1.1. Déclaration des bibliothèques :

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques. L'IEEE 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, etc. La directive use permet de sélectionner les bibliothèques à utiliser.

Example:

```
Library IEEE;
Use IEEE.std_logic_1164. ALL;
Use IEEE.Numeric_std. ALL;
Use IEEE. std_logic_unsigned. ALL;
```

D.1.2. Déclaration de l'entité et des entrées/sorties :

Cette opération permet de définir le nom de la description VHDL ainsi que les entrées et sorties utilisées, l'instruction qui les définit est « port ».
Le nom du signal est composé d'une chaîne de caractères dont le premier est une lettre. Le langage VHDL n'est pas sensible à la « casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.

Le signal peut être défini en entrée (In), en sortie (Out), en entrée/sortie (InOut) ou en buffer, c'est-à-dire qu'il est en sortie mais il peut être utilisé comme entrée à l'intérieur de la description. L'affectation des broches d'entrées/sorties se fait par la définition d'attributs supplémentaires qui dépendent du logiciel de développement utilisé.

Voici comment on peut déclarer une entité en langage VHDL :

```
ENTITY Nom de l'entité IS
Description des entrées et des sorties de la structure en explicitant pour chacune d'entre elles le nom, la direction (IN, OUT et INOUT) et le type.
END Nom de l'entité ;
```

D.1.3. Déclaration de l'architecture correspondante à l'entité :

L'architecture décrit le fonctionnement souhaité pour un circuit ou une partie du circuit. En effet le fonctionnement d'un circuit est généralement décrit par plusieurs modules VHDL. Il faut comprendre par module le couple entité/architecture. Dans le cas de simples PLDs on trouve souvent un seul module. L'architecture établit les relations entre les entrées et les sorties à travers les instructions. On peut avoir un fonctionnement purement combinatoire, séquentiel voire les deux : séquentiel et combinatoire.

Voici comment on peut déclarer une architecture en langage VHDL :

```
ARCHITECTURE Nom de l'architecture OF Nom de l'entité IS
Zone de déclaration.
BEGIN
Description de la structure logique.
END nom de l'architecture ;
```

D.2. Les avantages du VHDL :

Les principaux avantages du langage description VHDL sont :

- Spécification : le langage VHDL est très bien adapté à la modélisation des systèmes numériques complexes grâce à son niveau élevé d'abstraction. Le

partitionnement en plusieurs sous-ensembles permet de subdiviser un modèle complexe en plusieurs éléments prêts à être développés séparément.

- La simulation : le VHDL est également un langage de simulation, la notion de temps a été introduite. Des modules destinés uniquement à la simulation peuvent être crées et utilisés pour valider le fonctionnement logique ou temporel du code VHDL.
- La synthèse logique : les logiciels de synthèse permettent de traduire la description VHDL en logique. Il est ainsi possible d'intégrer la description dans un composant programmable (CPLD, FPGA) ou dans un circuit ASIC [18].
- La portabilité : la 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 bien sûr, que la description en question puisse s'intégrer dans le composant choisi et que l'outil utilisé possède une entrée VHDL).

REFERENCES

[1] : Idir A.H., Belmehdi A. and Chikouche D., « Recherche de Signatures de Défaut de la Machine à Induction en Vue de Diagnostic », 4th International Conference on Computer Integrated Manufacturing, (2007).

[2] : Doumbia L, Roy G, Rajagopalan V, «An Integrated Solution for Simulating Electrical Drive Systems with Matlab/Simulink», International Symposium on Industrial Electronics, 1997.

[3] : S.Brown and J.Pose FPGA and CPLD Architectures. A tutorial IEEE Design&Test of computers Summer 1996.

[4]. Hilairet M., « Application des outils du traitement du signal à la commande des machines tournantes », Thèse de doctorat, Université de Nantes, France, (2001).

[5]: Ghanes M. « Observation et commande de la machine asynchrone sans capteur mécanique », Thèse de doctorat, université de Nantes, France, (2005).

[6]: Chaouch S., « Commande vectorielle robuste d'une machine à induction sans capteur de vitesse », Thèse de Doctorat, université de Batna, (2005).

[7] : M. Mahmoudi, « Modélisation et Commande Vectorielle de la Machine Asynchrone » Document cours, Ecole Nationale Polytechnique, Alger.

[8] : Jean-Pierre Caron, Jean-Paul Hautier, Préface de Jean Faucher, «Modélisation et Commande de la Machine Asynchrone», Livre, Editions Technip, Paris, France, (1995).

[9]. Malrait F., « Problèmes d'identification et d'observabilité du moteur à induction pour la variation de vitesse industrielle sans capteur », Thèse de doctorat, Ecole Nationale Supérieure des Mines de Paris, France, (2001).

[10]: Nicolas HERVÉ, Daniel MÉNARD et Olivier SENTIEYS, « Synthèse d'architecture sur FPGA sous contrainte de précision des calculs » IRISA – Université de Rennes 1, France, (2005).

[11] : BENMOSBAH Amine et MECHERAOUI Choukri Adel « Implémentation sur FPGA des méthodes MPPT : "P&O" et "floue optimisée par les Algorithmes Génétiques ». Ecole national polytechnique.

[12] : Zahir Ait Ouali « Application des FPGA à la commande d'un moteur asynchrone », Mémoire de MAGISTER, Université MOULOUD MAMMERI de TIZI-OUZO.

[13] : Mark B, « Complete Digital Design», McGraw-Hill Companies, Inc., USA, (2003), 460p.

[14] : Maamoun M, « Nouvelles architectures d'interfaçage graphique dans les systèmes à microprocesseur », Thèse de Doctorat, ENP, Alger.

[15] : OUNISSI Oussama, « implémentation sur un circuit FPGA du filtre de KALMAN (FK) pour la machine asynchrone triphasée », mémoire de Magister, université SAAD DAHLAB, Blida, (2012).

[16] : N. JULIEN « Circuits logiques programmables » Université de Bretagne Sud – Lorient, Fance, (1999).

[17] : Etienne Messerli, Yves Meyer « Electronique Numérique, Systèmes combinatoires », Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud, France, (2010).

[18] : http://www.vetopsy.fr/sens/visu/ret1_struc.php.

[19] : J-M DUTERTRE « Circuits Reconfigurables Robustes » Thèse de doctorat, Université de Montpellier II, Oct. 2002.

[20] : D. MYLNEK « Design of VLSI Systems »Ecole polytechnique Fédérale de Lausane (EPFL) 2001 http://lsiwww.epfl.ch/LSI2001/teaching/ webcourse/toc.html.

[21] : D.SMITH « HDL Chip Design : A pratical Guide for Designing, Synthesis &Simulating Asics &FPGAs using VHDL or Verilog » Doone Pubns .ISBN : 0965193438.

[22] : H.Tamrabet, «Robustesse d'un Contrôle Vectoriel de Structure Minimale d'une Machine Asynchrone » Thèse de magister, université de Batna 2006.

[23] : Sasao T, Kubota H « Stability Analysis of Sensor-less Induction Motor Drives with Stator Resistance Adaptation using Estimation Error Index », Proceeding of International Conference on Electrical Machines and Systems 2007.

[24] : « Virtex-5 FPGA », User Gide (UG190, v5.3), by Xilinx, (2010).

[25] : Ahmed T., Kundarewich P.D., Anderson J.H., Taylor B.L., and Aggarwal R., « Architecture-Specific Packing for Virtex-5 FPGAs », FPGA'08, Monterey, California, USA, (2008).

[26] : Chekired F., « Etude et implémentation d'une commande MPPT neurofloue sur FPGA », Thèse de Magister, ENP, Alger, (2008).

[27] : Mohamed Wissem NAOUAR., « Commande numérique à base de composants FPGA d'une machine synchrone » Thèse de doctorat, l'ecole nationale d'ingenieurs de Tunis et l'universite de cergy pontoise France, (2007).