الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Projet de Fin d'Études

présenté par

AZZA ABDELAZIZ

&

OUAZENE HAMZA

pour l'obtention du diplôme de Master en Génie Electrique option Automatique

Thème

ETUDE ET IMPLEMENTATION SUR FPGA DE LA COMMANDE PID

Proposé par : Mr. KARA KAMEL & Mr. AIT SAHED OUSSAMA

Juin 2012

Dédicaces

Je dédie ce travail aux gens qui m'ont soutenu et aidé à préparer ce travail:

Mes parents

Mes frères, surtout Houcine et Nedjwa

Ma nièce Aya

ma famille,

mes enseignants,

mes amis,

mes collègues,

et...

à toute personne qui s'intéresse à la science.

Azza abdelaziz

Remerciements

Nous remercions dieu de nous avoir donnée patience et courage pour accomplir ce travail.

Nous remercions nos parents et nos frères et sœurs pour leur soutient, leur amour et leur encouragement.

Nous tenons à remercier notre encadreur Mr. K.Kara qui s'est toujours montré à l'écoute et très disponible tout au long du travail, ainsi que son aide et le temps qu'il nous a consacré afin que notre travail soit bien fait et bien présenté.

Nos remerciements s'adressent également Mr. A.S.Oussama pour ces conseils et ces encouragements.

Nous remercions aussi tous nos amis et nos proches qui se demandaient sur le moindre avancement du travail.

Enfin, nous n'oublions pas à remercier tous le staff du département d'électronique à l'université de BLIDA.

Dédicaces

A mes parents et à toute ma grande famille OUAZENE.

A mon frère Sadam.

A ma chère sœur et son mari HOUCINE.

A mes neveux Adam Mohammed Islam, Sidra et Besma.

A mon cher ami Zegay Mohammed et mes amis du campus.

A tous mes Enseignants ,de l'école primaire à l'université.

A mon ami et mon enseignant OSSAMA.

A mon cher enseignant Mr. Guessoum Abderrezak.

A tous mes amis de Souk Lethnin et de Damous.

A mon cheikh Mourad Attassi.

Ouazene Hamza

ملخص: تعرض هذه المذكرة تثبيت قانون التحكم PID على دارة قابلة للبرمجة FPGA (Spartan 3-E). وتعرض أيضا تطبيقا لهذا القانون من أجل التحكم في سرعة محرك ذي تيار مستمر. قبل التثبيت قمنا بمحاكاة التركيب باستعمال البرامج : ISE 12.3 وISim. تم التحقق من عمل التركيب و فعالية التحكم من خلال عدة تجارب أجريناها في المخبر بإدماج الدارة في حقل تجارب.

كلمات المفاتيح: الدارة القابلة للبرمجة FPGA Spartan-3E,قانون التحكم PID, لغة برمجة VHDL,التحكم في محرك ذو تيار مستمر.

Résumé : Ce mémoire présente l'implémentation du régulateur PID sur la carte de développement Sapartan-3E en utilisant le langage VHDL. Les performances de l'algorithme implémenté sont mises en évidence en considérant l'asservissement en vitesse d'un moteur à courant continu. Les outils utilisés pour l'implémentation et la simulation des différents modules sont le logiciel Xilinx ISE 12.3 et le simulateur ISim. Le mémoire présente aussi les résultats des simulations et des tests pratiques réalisés au cours de la mise au point de ce projet.

Mots clés : FPGA, VHDL, Carte de développement Spartan-3E, le contrôleur PID, Modulation en Largeur d'Impulsion (MLI), commande en vitesse d'un MCC.

Abstract: This report outlines the PID controller implementation on the starter kit Spartan-3E using VHDL language. To highlight the performance of the implemented algorithm, the speed control of a DC motor application is considered. The tools that are used for implementing and testing the different Modules necessary for building the PID controller are Xilinx ISE 12.3 and ISim simulator tools. The report also presents the simulation and the test results that are carried out to illustrate and test the designed modules.

Keywords: FPGA, VHDL, Spartan-3E FPGA kit board, Proportional Integral Derivative (PID) controller, Pulse Width Modulation(PWM), DC Motor speed Control.

Conclusion Générale

L'objectif de ce travail a été de faire l'étude du régulateur PID et de mettre en évidence la contribution et l'apport des FPGA dans la commande des systèmes. Pour ce faire, le travail présenté dans ce mémoire était principalement attaché à l'implémentation du contrôleur PID numérique. Il existe plusieurs approches pour l'algorithme du PID numérique, nous avons choisi un algorithme simple que nous avons appliqué pour la commande en vitesse du MCC. Les paramètres de ce dernier ont été tirés par identification. La synthèse par placement des pôles a été mise en œuvre et les paramètres du PID ont été calculés.

Pour mettre en œuvre le système de commande en vitesse du MCC par le régulateur PID, nous sommes passés par une vue générale sur les techniques de génération des signaux PWM.

Les circuits FPGA et leur configuration ont fait l'objet d'une étude un peu approfondie dans les deux derniers chapitres. La carte de développement Spartan-3E et ses modules, le VHDL, l'ISE et ses outils de développement ont été définies avant de passé à la synthèse du système de commande. Un flot de développement bien déterminé a été suivi pour réaliser cette synthèse d'une manière correcte.

Les outils de développement ont facilité la description en VHDL, la vérification et l'implémentation du régulateur PID et des modules secondaires (générateur des PWM, Mesureur de vitesse).

Un banc d'essais composé principalement d'un ordinateur, d'un circuit d'isolation, d'une carte de puissance, d'un moteur à courant continu et de la carte de développement Spartan-3E, a été réalisé afin de permettre une manipulation facile lors des essais pratique. Ce banc a été largement exploité.

Les résultats de simulation nous ont encouragé de passer vers la pratique, et d'aborder quelques essais qui ont permet la validation de la bonne fonctionnalité du système de commande, à base du régulateur PID numérique, ainsi réalisé.

Ce même banc d'essais peut être exploité dans le futur pour l'implémentation d'une variété d'algorithmes de commande, voir : La commande auto-ajustable, la commande prédictive, la commande floue, la commande neuro-floue…etc.

Enfin, en tant que automaticiens chercheurs, ce projet nous a permet de consolider les notions acquises au cours de notre formation à propos de la commande numérique des systèmes et plus précisément, l'implémentation des algorithmes numériques sur un système embarqués tel que FPGA.

Sommaire

Chapitre 2

Génération des signaux PWM

Chapitre 3

Les Circuits Programmables

FPGAs

Chapitre 4

Implémentation et Essais Pratiques

Notations et Symboles

$G_c(s)$	Fonction de transfert du contrôleur
$G_a(s)$	Fonction de transfert du moteur
$G_a(s)$	Fonction de transfert associée à la consigne
W(s)	La consigne
Y(s)	La sortie
E(s)	L'erreur
U(s)	La commande
K_p	Gain proportionnel
K_i	Gain intégral
K_d	Gain dérivation
K_0	Gain statique du processus
K_{cr}	Gain critique
K_n	Gain normalisé
K	Gain relatif
T_i	Temps d'intégration
T_d	Temps de dérivation
T_t	Constante de temps de remise à zéro dynamique de l'intégrateur
L	Retard apparent
τ	Temps mort relatif
T	Constante de temps apparente
R	Résistance
Ω	Vitesse de rotation du rotor
f	Coefficient de frottement
K_c	Constante de couple
K_e	Constante de vitesse

PWM	Pulse Width Modulation
MLI	Modulation en largeur d'impulsion
α	Rapport cyclique
V_{s0}	La tension moyenne
FPGA	Field Programmable Gate Array
CPLD	Complex Programmable Logic Device
PLD	Programmable Logic Device
EPLD	Erasable Programmable Logic Device
PAL	Programmable Array Logic
GAL	Generic Array Logic
MOS	Metal Oxyde Semiconductor
SRAM	Static Read Access Memory
CLB	Configurable Logic Bloc (bloc logique configurable)
IOB	Inptu Output Bloc (bloc d'entrées sorties)
LUT	Luck Up Table
LCA	Logic Cell Array
VHDL	Very high speed integrated circuit Hardware Description Language
DSP	Digital Signal Processor
RTL	Register transfer Level

Liste des figures et tableaux

Chapitre 1

La régulation PID

Chapitre 2

Génération des signaux PWM

Chapitre 3

Les Circuits Programmables FPGAs

Chapitre 4

Implémentation et Essais pratique

Conclusion Générale

L'objectif qui a été donné à ce travail est l'implémentation du régulateur PID dans un circuit FPGA ensuite l'étude des performances de l'algorithme implémenté en considérant la commande en vitesse d'un moteur à courant continu. De ce fait, le travail présenté dans ce mémoire est principalement consacré à la réalisation de cette implémentation. La mise au point de cette tâche passe nécessairement par la synthèse et la numérisation du PID. En effet, après avoir identifié le modèle discret du moteur, nous avons choisi d'utiliser une méthode de synthèse discrète basée sur la technique de placement de pôles pour déterminer les paramètres du PID à implémenter.

L'implémentation réalisée a été divisée en plusieurs modules à savoir : le module de génération des signaux PWM, le module de division de fréquence, le module de mesure de vitesse et le module PID. Chacun de ces modules a été réalisé et testé séparément des autres en faisant des simulations et des essais pratiques. Nous notons que cette approche modulaire, nous a énormément facilité la tâche de synthèse et de test étant donné que chaque module a été considéré indépendamment de l'autre. La concordance des résultats de simulation et des résultats pratiques obtenus pour chaque module et pour l'ensemble des modules interconnectés ont bien montré la réussite de l'implémentation considérée. L'avantage essentiel que nous pouvons tirer de cette implémentation est l'exécution rapide de l'algorithme de commande. Ceci est principalement dû au parallélisme qu'offre ce type de circuits numériques. Il est clair que ce gain en terme de temps de calcul permettra l'implémentation en temps réel des algorithmes de commande complexes et de considérer la commande des systèmes rapides.

Toutes les difficultés que nous avons rencontrées lors de la réalisation de ce travail sont liées à la non disponibilité du matériel d'une part et à la non maîtrise de l'implémentation des algorithmes sur les circuits FPGA. Nous tenons à noter ici que la majorité des projets, relatifs à l'implémentation sur FPGA, disponibles au niveau de la bibliothèque de notre faculté et que nous avons eu l'occasion de consulter sont arrêtés à la phase de simulation.

En tant qu'automaticiens, ce projet nous a permet de consolider les notions acquises au cours de notre formation à propos de la commande numérique des systèmes et plus précisément, l'implémentation des algorithmes numériques sur un système embarqués à base de circuits FPGA.

Le travail que nous avons réalisé pourra faire l'objet de plusieurs améliorations dans le futur. En particulier nous pouvons proposer d'exploiter les ports de communication pour lire les mesures à partir

d’un ordinateur pour permettre le suivi en temps réel l’évolution de la vitesse au cours du temps. Une autre amélioration qui peut être aussi considéré est l’exploitation de l’afficheur LCD de la carte pour afficher la vitesse. Une suite à ce travail qui pourra être aussi envisagé est l’implémentation de d’autres algorithmes plus complexes et la considération des applications réelles.

Chapitre 2
Génération des signaux PWM

1. Introduction

La Modulation en Largeur d'Impulsions (MLI) en anglais Pulse Width Modulation (PWM) est une technique de pilotage des convertisseurs statiques servant d'interface entre une charge (machine électrique) et son dispositif d'alimentation (hacheurs ou onduleurs). C'est une technique utilisée pour la conversion d'énergie ayant ses bases dans le domaine des télécommunications (traitement du signal).
Les techniques de MLI ont été l'objet de recherches intensives [23] [24] [25] [26], un nombre important de stratégies, différentes de par leurs concepts et leurs performances, ont été développées. La génération des signaux PWM est devenue une partie intégrante de la majorité des systèmes embarqués. Le choix d'une technique dépend du type de machine à commander, de la gamme de puissance, des semi-conducteurs utilisés dans le convertisseur (hacheurs ou onduleurs) et de la simplicité d'implantation de l'algorithme de commande. Ce sont finalement des critères de coût et de performances qui vont déterminer ce choix. L'avancement des méthodes de synthèse et le développement de l'électronique des semi-conducteurs l'ont mené à devenir très populaire. De nombreuses techniques PWM sont utilisées pour obtenir une source ayant un voltage et une fréquence variables.

2. Les techniques PWM

Il existe deux types fondamentaux de techniques PWM : techniques analogiques et techniques numériques. Dans les techniques analogiques, on a une porteuse et une modulante. Ces deux signaux sont comparés à l'aide d'un comparateur. La sorties de ce comparateur est la sortie PWM désirée. Il y a fondamentalement quatre techniques analogiques :

a. Le signal PWM sinusoïdal
b. Le signal PWM sinusoïdal modifié
c. Le signal PWM à modulante à un seul niveau
d. Le signal PWM à modulante multi-niveaux

D'après [23], [25] et [26] les inconvénients de ces techniques sont la sensibilité au bruit, la variation de la température, ou le changement de la tension. Aussi, elles subissent des changements dus aux variations des valeurs des composants. Les techniques numériques sont plus convenables pour la synthèse des générateurs PWM. Elles sont très flexibles et moins sensitive aux bruits environnementaux [23]. Aussi, elles sont faciles à construire et peuvent être implémenté d'une manière très facile. La plupart des techniques numériques utilisent des circuits à base d'un compteur et d'un comparateur. Ces techniques feront l'objet de notre étude.

3. Les avantages de la synthèse à base d'un FPGA :

Un FPGA offre une bonne méthode pour la synthèse d'un générateur des signaux PWM. Elles sont, en fait, des interconnexions entre différents blocs logiques et leur synthèse est faite de manière à permettre de simples futures modifications. Nous n'avons qu'à changer les interconnections entre ces blocs logiques. La faculté d'être reprogrammé rend l'implémentation sur des circuits FPGA plus favorable [27].

4. Principe d'un signal PWM

Il consiste à alterner rapidement entre deux états distincts du système afin d'obtenir en moyenne un signal analogique. Comme il est illustré dans la figure (2.1), un signal PWM peut être obtenu en comparant un signal quelconque avec un autre signal triangulaire. En effet, si le signal comparé est au-dessus du signal triangulaire, la sortie est à l'état haut. Dans le cas contraire, la sortie est mise à l'état bas.

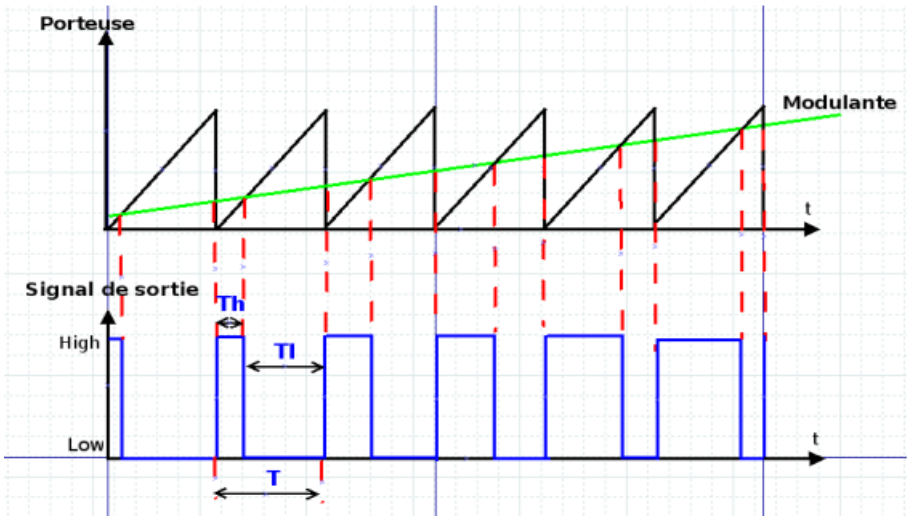


Fig.2.1 : Principe du PWM

Le signal continu généré à partir des deux valeurs distinctes prend la valeur moyenne sur les intervalles de temps. On précise que dans la figure (2.1) :

- Th : représente la durée du temps pendant laquelle la sortie est à l'état haut.

- TI : représente la période de temps pendant laquelle la sortie est à l'état bas.

Un signal PWM se caractérise par :

- la période PWM :

$$T = Th + TI \tag{2.1}$$

- le rapport cyclique (Duty cycle) :

$$\alpha = \frac{Th}{T} \tag{2.2}$$

La fréquence du signal PWM est toujours constante par contre le rapport cyclique peut varier. De plus, le signal PWM est un signal carré qui varie entre deux valeurs extrêmes mais les systèmes sur lesquels on applique un signal PWM sont en général des filtres passe bas (moteurs, œil humain,...), ils ne ressentent qu'à sa valeur moyenne.

5. Modes du PWM

Il existe deux modes du PWM :

- edge-aligned : Le signal est asymétrique par rapport au début de la période. Ce signal s'obtient en comparant un signal constant avec un signal triangulaire possédant une seule pente. Nous trouvons le mode left edge-aligned (à chaque début de période, on est à l'état haut) et le mode right edge-aligned (à chaque fin de période, on est à l'état haut) (Fig.2.2).
- center-aligned : Ce signal est symétrique par rapport au début d'une période. Il s'obtient en comparant un signal constant avec un signal triangulaire à deux pentes (Fig.2.3).

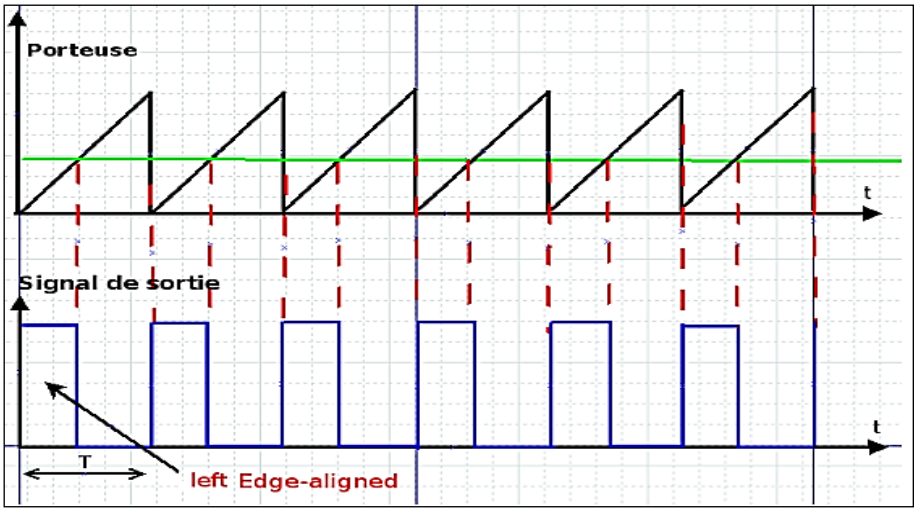


Fig.2.2 : Edge-aligned

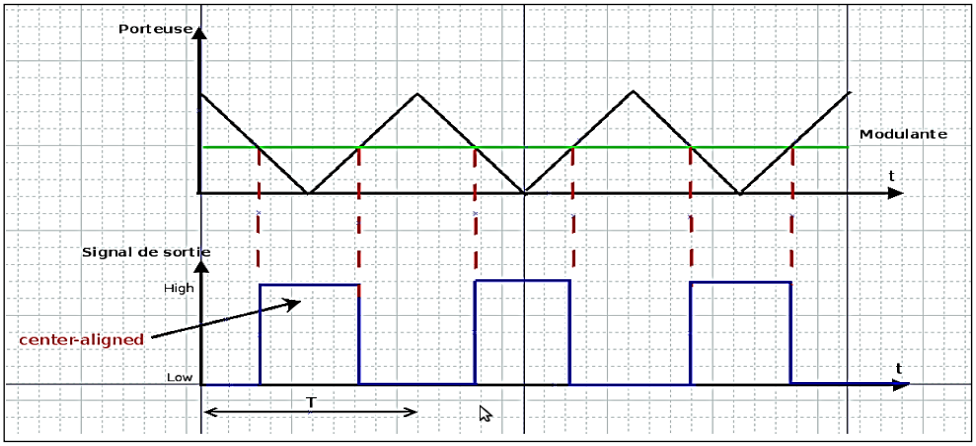


Fig.2.3 : Center-aligned

6. Techniques numériques pour la génération des signaux PWM

Comme nous l'avons mentionné au deuxième paragraphe, les techniques numériques sont plus favorables pour la réalisation d'une commande sophistiquée que les techniques analogiques. Il existe plusieurs techniques numériques dépendant de la disposition et du type de compteurs utilisés. Dans ce paragraphe, trois topologies de base pour un générateur de signaux PWM sont présentées. Il existe, le :

- Générateur des signaux PWM à base de compteur haute fréquence,
- Générateur des signaux PWM à base de compteur,
- Générateur des signaux PWM à base de compteur en cascade.

a. Générateur des signaux PWM à base de compteur haute fréquence :

Cette architecture a été proposée par E. Koutroullis, A. Dollas et K. kalaitzakis [27]. Dans cette architecture, la sortie de N bits d'un compteur haute fréquence est comparée avec la sortie d'un registre de N bits contenant la valeur désirée du rapport cyclique à l'aide d'un comparateur. La sortie du comparateur prend l'état '1' quand les deux valeurs sont égales. Cette sortie de comparateur est utilisée pour la mise à 1 d'une bascule RS. Le signal de débordement du comparateur est utilisé pour la remise à zéro de la bascule RS. La sortie de la bascule RS représente la sortie PWM désirée. Le signal de débordement est aussi utilisé pour charger une nouvelle valeur du rapport cyclique dans le registre. L'avantage de cette méthode est sa capacité de générer une sortie PWM de haute fréquence, ce qui n'est pas possible dans un générateur à base de compteur. La figure (2.4) montre le schéma bloc de cette architecture.

b. Générateur des signaux PWM à base de compteur

Cette architecture, proposée par A.P. Dancy, R. Amirtarajah et A.P. Chandrakasan [28], est utilisée pour les sources de faible énergie. Le signal de sortie PWM est mis à '1' avant le signal d'horloge il reste à '1' jusqu'à la remise à zéro quand la sortie du compteur atteint la valeur du rapport cyclique.

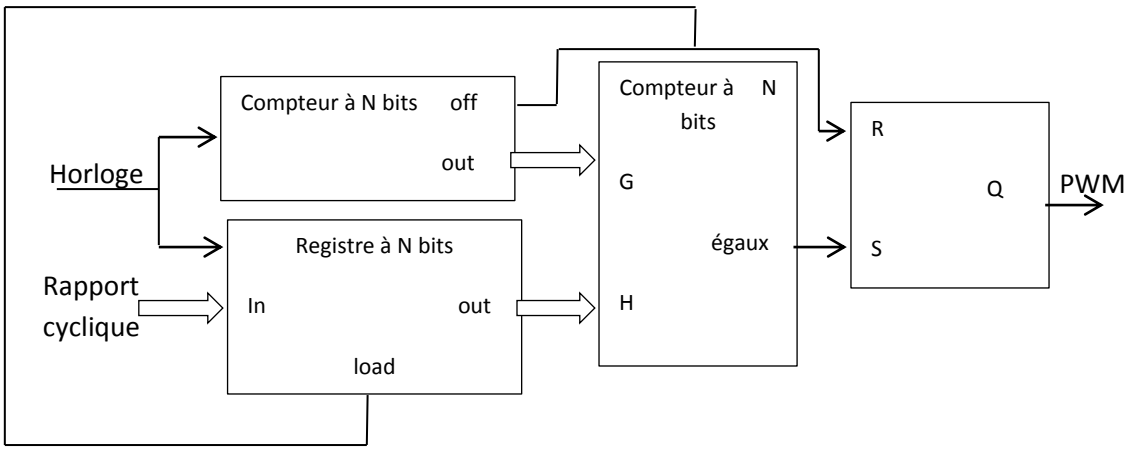


Fig.2.4 : Architecture du générateur des signaux PWM proposé par K. Kalaitzakis et al.

c. Générateur des signaux PWM à base de compteur en cascade

Dans cette architecture deux compteurs à 4 bits sont mets en cascade pour former un compteur à 8 bits. Puis, on applique le signal d'horloge sur les deux compteurs. On relie la sortie de chaque compteur à l'une des deux entrées d'un comparateur à 8 bits. On introduit le rapport cyclique dans l'autre entrée du comparateur. La sortie du comparateur est mise à '1' quand ses deux entrées sont égales. Ce comparateur est utilisé pour la remise à zéro de la bascule RS. Les bits de débordement sont utilisés pour la mise à 1de la bascule. La sortie de la bascule RS fournit le signal PWM.

7. Application :

Dans ce paragraphe, nous allons voir l'une des applications les plus connus des signaux PWM. Cette application consiste à commander les interrupteurs d'un hacheur à quatre quadrants dit pont en H afin de piloter un moteur à courant continu.

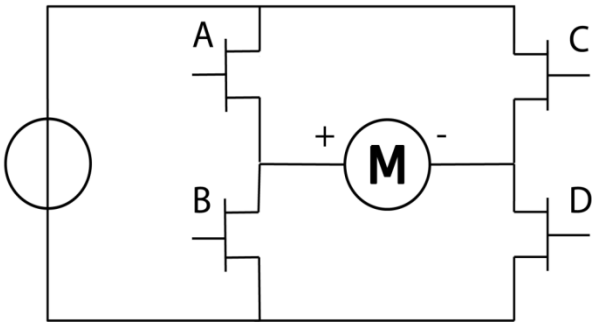


Fig.2.5 : Le pont en H

Pour un hacheur à quatre quadrants, il suffit de commander les interrupteurs en utilisant un signal PWM pour chaque bras. Quand le premier signal PWM attaque le premier bras avec une tension de αU, le deuxième signal PWM attaque le deuxième bras par une tension de $(1-\alpha)U$. La formule régissant le fonctionnement du hacheur quatre quadrant est donnée par :

$$V_{s0} = (2\alpha - 1)U \tag{2.3}$$

Où :

V_{s0} : La tension moyenne

U : La tension d'entrée

α : Le rapport cyclique

Dans cette application, nous avons attaqué le hacheur par une séquence de commandes (modulante) (Fig.2.6) qui sera comparée avec la sortie du compteur en dents de scie (porteuse) (Fig.2.7). Cette comparaison engendre un rapport cyclique variable (Fig.2.8). Les deux signaux PWM résultants sont représentés par les figures (2.9) et (2.10).

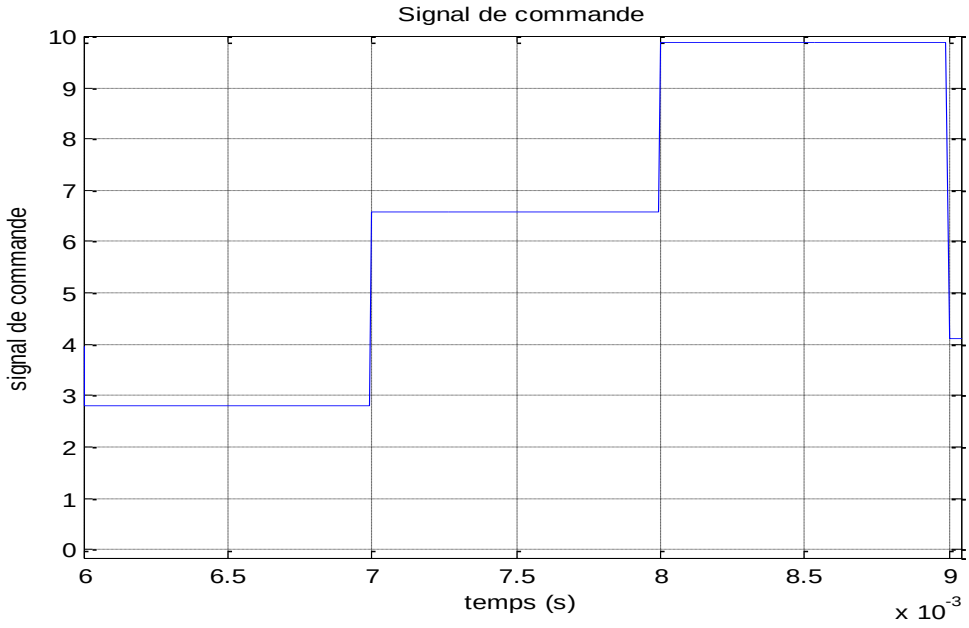


Fig.2.6 : Le signal de commande (modulante)

Remarque :

Dans cette application, nous voulons obtenir un signal PWM de 10 KHz. Pour cela nous allons choisir une porteuse de fréquence égale à 10KHz. La porteuse est représentée par les valeurs que prend un compteur comptant jusqu'à cent et échantillonné avec une fréquence de 1MHz.

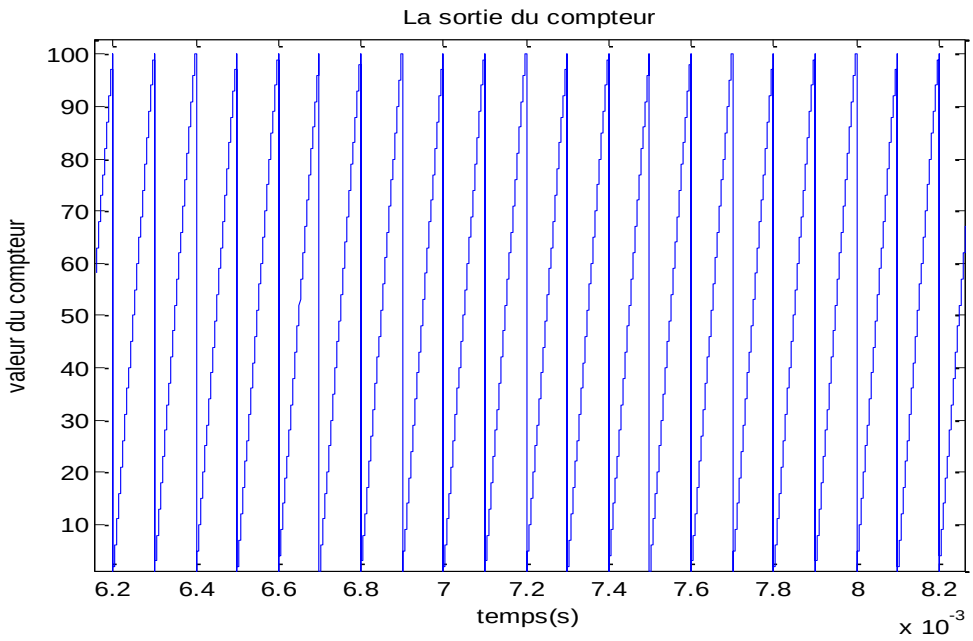


Fig.2.7 : La sortie du compteur (porteuse)

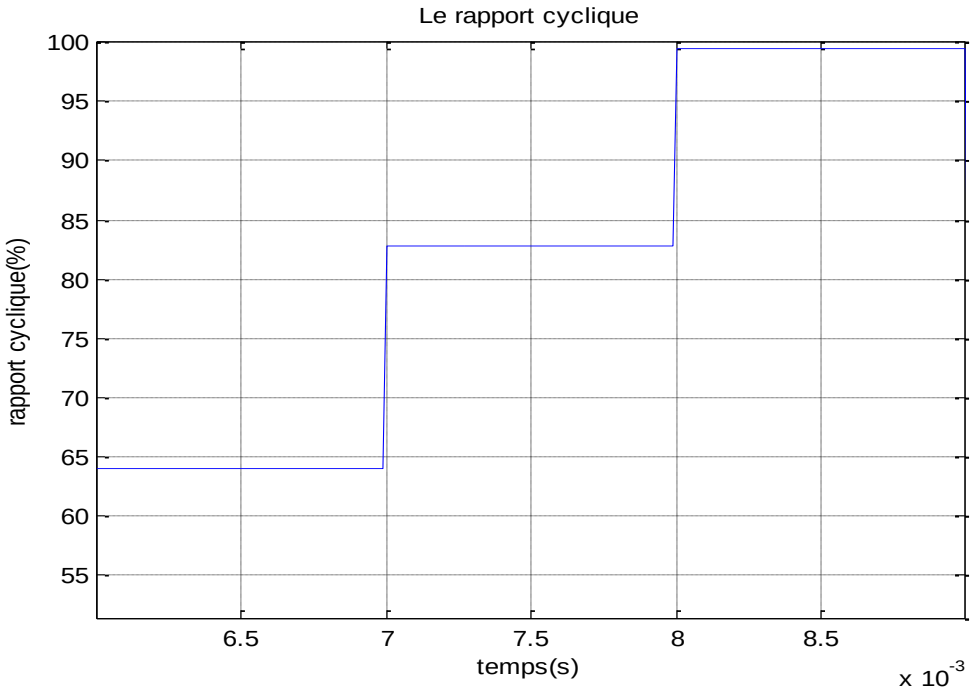


Fig.2.8 : Le rapport cyclique engendré

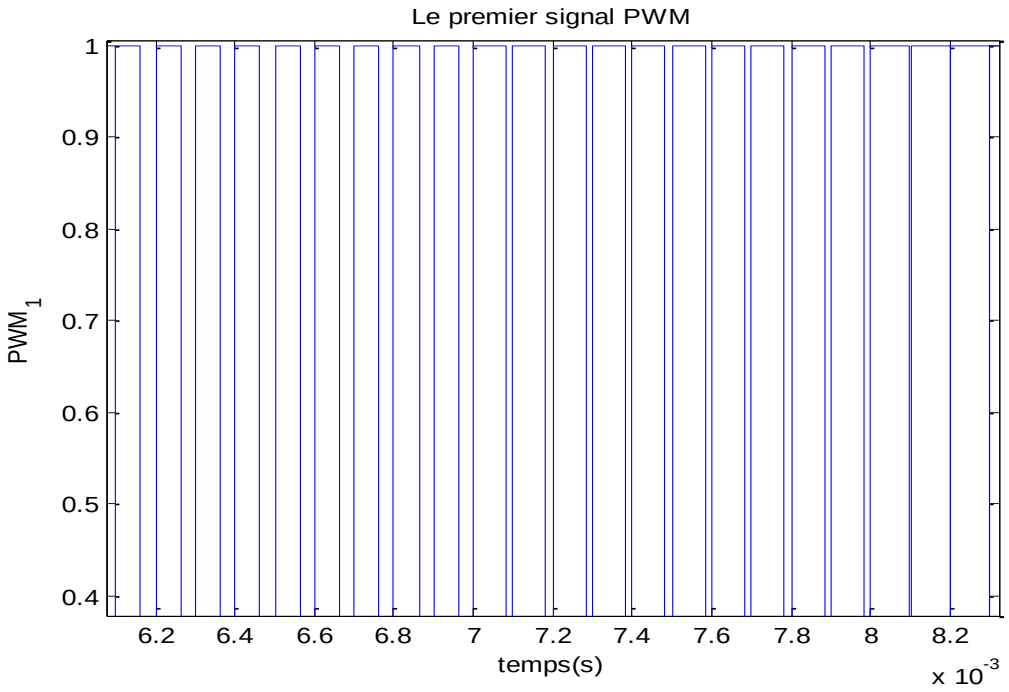


Fig.2.9 : La première sortie PWM

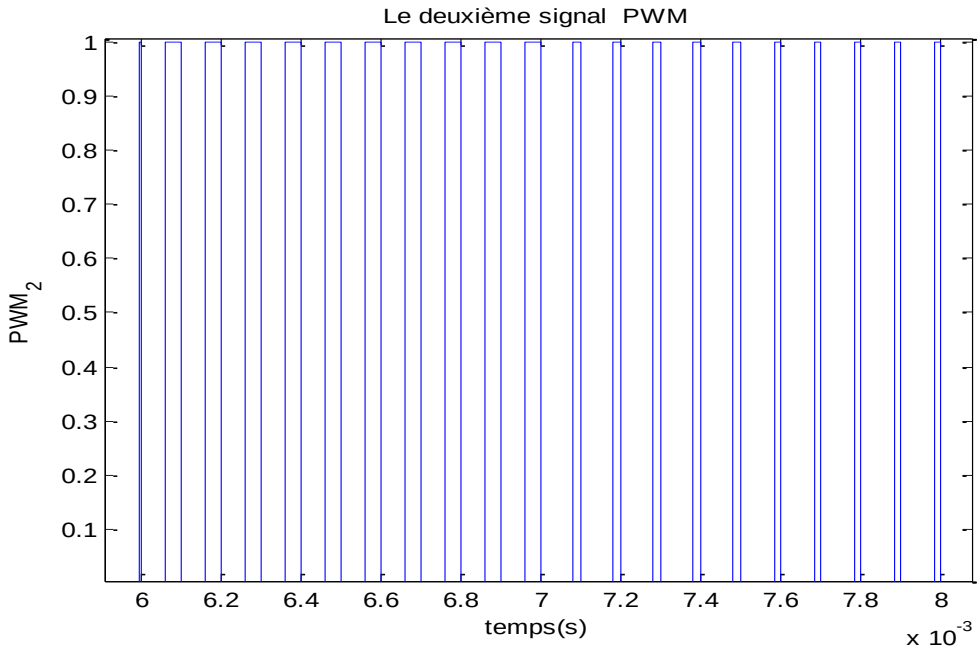


Fig.2.10 : La deuxième sortie PWM

Remarque :

Les deux signaux PWM résultants sont complémentaires. Chaque signal a une fréquence constante de 10KHz et un rapport cyclique variable. Si on prend l'exemple du premier signal, son rapport cyclique augmente graduellement ce qui implique une augmentation de la tension moyenne en sortie du hacheur provoquant une augmentation de la vitesse du moteur à courant continu.

8. Conclusion :

Dans ce chapitre, après avoir définir un signal PWM et expliqué ces différents types, nous avons présenté les différentes techniques utilisés pour sa génération. Ensuite, une application de ce type de signaux a été considérée.

Références

[23] Rahim N.A. and Islam Z., "Field Programmable Gate Array-Based Pulse-Width Modulation for Single Phase Active Power Filter"; American Journal of Applied Sciences, Vol.6 (2009): pp. 1742-1747

[24] Retif J.M., Allard B., Jorda X. and Perez A, "Use of ASIC"s in PWM techniques for power converters", Proceedings of the International Conference on Industrial Electronics, Control and Instrumentation, IEEE Xplore Press, Maui, HI, USA.,(1993) pp: 683-688.

[25] Mohd. Shafie Bakar et al « Analysis of various PWM controls on single-phase Z-source inverter» article, Proceedings of 2010 IEEE Student Conference on Research and Development.

[26] Dariusz Czarkowski « Solving the Optimal PWM Problem for Single-Phase Inverters» article, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS; April 2002.

[27] Koutroulis E., Dollas A. and Kalaitzakis K., "High-frequency pulse width modulation implementation using FPGA and CPLD ICs", Journal of Systems Architecture, Vol.52 (2006): pp. 332–344

[28] Dancy A.P., Amirtharajah R. and Chandrakasan A.P., "High-Efficiency Multiple-Output DC–DC Conversion for Low-Voltage Systems", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 3, June 2000: pp.252-263

Introduction Générale

Les contrôleurs PID (Proportionnel, intégrateur, dérivateur) ont été largement utilisés dans les dernières cinq décennies à cause de leur simplicité, robustesse, efficacité et applicabilité dans une large gamme de systèmes. Malgré de nombreuses approches qui ont apparait dans la littérature, il est estimé que, de nos jours, les contrôleurs PID sont utilisés dans plus de 95% des processus industriels [1]. Pour plusieurs décennies le contrôleur PID numérique a été beaucoup utilisé pour la commande numérique en temps réel. Le PID est largement utilisé dans les entrainements, la robotique, la commande de la température et l'électronique de puissance. D'une manière générale, le PID est un outil très important pour le synthétiseur de commande numérique des systèmes embarqués en temps réel. Il est souvent implémenté soit en hardware en utilisant des composants analogiques ou en logiciel en utilisant des systèmes numériques.

L'émergence des circuits logiques programmables tels que FPGA (Field Programmable Gate Array) et des langages de description matérielle permet d'avoir des dimensions supplémentaires pour les PID numériques (le parallélisme, la taille de bus programmable) …etc. Avec l'augmentation de la complexité du moteur et les applications de la commande en vitesse, il devient apparent qu'un FPGA offre des avantages significatifs par rapport aux autres circuits de traitement numérique des signaux [2].

Avec un FPGA, les calculs qui doivent prendre un grand temps au microprocesseur quand ils sont implémentés en logiciel peuvent être accélérés par le hardware. Les interfaces de pilotage de moteurs comme la modulation en largeur d'impulsion (PWM : Pulse Width Modulation) peuvent être développées facilement, rapidement et avec un coût réduit. Parce qu'il peut être entièrement configuré, le même FPGA peut être utilisé pour différents gammes de produits [3]. La famille Spartan 3E des FPGAs a été conçue de manière à satisfaire les besoins en termes de grande densité et de coût bas pour le développement des applications. Les FPGAs modernes et leurs capacités distinguables ont été annoncés par les vendeurs des FPGAs [4]. En outre, quelques articles des journaux spécialisés ont adressé les avantages de l'utilisation de ces puissantes puces [5] [6]. Dans les dernières quatre années, les deux familles Spartan 2 et 3 de Xilinx ont été largement utilisés avec succès dans une grande

variété d'applications incluant les onduleurs [7] [8], la communication [9] [10], les processeurs [11], et le traitement d'image. L'implémentation du contrôleur PID en utilisant un DSP (Digital Signal Processor) est ancien est bien connu [12] [13], alors que moins de travail peut être trouvé dans la littérature sur la manière dont on implémente un contrôleur PID sur une cible FPGA.

Le logiciel de développement ISE de Xilinx permet la création d'une interface à travers les boutons poussoirs, les commutateurs et les LEDs de la carte de développement de telle façon à ce que l'utilisateur peut changer la vitesse de référence du moteur à courant continu, lire la vitesse mesurée en sortie et remettre à zéro le système tout entière.

L'objectif de notre travail est l'implémentation de la commande PID sur la carte de développement Spartan-3E. Pour mettre en évidence les performances de commande de l'algorithme implémenté, la commande en vitesse d'un moteur à courant continu est considérée dans ce travail.

Le mémoire est organisé en quatre chapitres :

- Le premier chapitre est consacré à l'étude théorique de la commande PID. Différentes méthodes de synthèse y sont présentés et spécialement la synthèse par placement des pôles.
- Le deuxième chapitre résume les différentes techniques de génération des signaux PWM. La simulation d'un signal PWM ayant un rapport cyclique variable est présentée en fin du chapitre.
- Le troisième chapitre présente les différents types des circuits logiques programmables. Ensuite, il donne une vue globale sur l'architecture interne des FPGAs. Le langage de description matérielle VHDL, le flot et les outils de développement y sont aussi discutés.
- Le quatrième chapitre traite la description du banc d'essai ainsi que ces différentes parties. Puis une description de chaque module développé au cours de notre travail est donnée avec les résultats de simulations et ceux des essais pratiques propres à chaque module.

Dans la dernière partie de ce mémoire, nous présentons les conclusions et les perspectives qui feront l'objet des travaux futur.

Références

[1]. K. J. Astrom and T. H. Hagglund, New Tuning Methods far PID Controllers, in *Proc. of 3rd European Conference*, 1995, pp. 2456-2462.

[2]. Shouling He and Xuping Xu, Hardware/Software Co design Approach for an ADALINE Based Adaptive Control System, *Journal of Computers*, Vol. 3, No. 2, February 2008, pp. 29-36.

[3]. Craig Hackney, PGA Motor Control Reference Design, *Application Note: Spartan and Virtex FPGA Families, Xilinx* XAPP808 Vol. 1.0, September 16, 2005.

4]. Mohamed Abdelati, FPGA-Based PID Controller Implementation, The Islamic University of Gaza, Palestine.

5]. Anthony Cataldo, Low-priced FPGA options set to expand, *Electronic Engineering Times Journal,* No. 1361, 2005, pp. 38-45.

[6]. Gordon Hands, Optimized FPGAs vs. dedicated DSPs, *Electronic Product Design Journal,* Vol. 25, No. 12, December 2004.

Chapitre 1
La régulation PID

1. Introduction

Le régulateur le plus utilisé dans l'industrie est le régulateur PID (proportionnel intégral dérivé). Il permet de régler à l'aide de ses trois actions les performances d'un système en boucle fermée. La prédominance incontestée de ce type de régulateurs provient, outre de la simplicité de sa structure et le nombre restreint de paramètres à ajuster, des performances qu'il peut offrir aux systèmes en boucle fermée, satisfaisant très souvent les cahiers des charges, si ses paramètres sont choisis judicieusement. Plusieurs méthodes de synthèse d'un régulateur PID, offrant une complexité et des performances très variables, sont disponibles dans la littérature. Cependant, un PID peut être considéré comme un outil dont l'ajustement peut être accompli en considérant indépendamment l'effet produit par chacun de ses paramètres. Ceci a conduit à des règles et formules empiriques d'ajustement très utiles.

Les régulateurs PID répondent à plus de 95% des besoins industriels et le nombre de régulateurs installés dans une usine pétrolière, par exemple, se compte par milliers. Malheureusement, malgré l'expérience acquise au fil des ans, les valeurs choisies pour les paramètres des actions P, I et D ne sont pas toujours satisfaisantes, ni adaptées au processus à régler.

En 1942, Ziegler et Nichols [14] ont proposé deux démarches permettant de trouver facilement les paramètres optimums pour une installation donnée. Au fil des ans, les propositions de Ziegler et Nichols ont été adaptées ou modifiées selon les besoins.

En 1963, Horowitz [15] a ajouté un degré de liberté supplémentaire au régulateur PID afin de mieux contrôler les dépassements obtenus lors d'une réponse indicielle. Ce nouveau degré de liberté consiste, en particulier, à ne réinjecter vers le terme proportionnel qu'une partie du signal de sortie.

Au début des années 1990 et dans le but de fournir des règles d'ajustement simples mais plus performantes que celles de Ziegler-Nichols, Aström [16] et ses collaborateurs ont analysé le comportement dynamique d'un grand nombre de processus. Cette analyse a conduit à l'établissement de tableaux servant aux calculs des paramètres des actions P, I et D à partir de mesures simples.

Dans ce chapitre, après avoir présenté les régulateurs PID classiques et modifiés nous présentons les différentes méthodes de synthèse : par compensation des pôles, celles de Ziegler et Nichols, et celles de Aström et Hägglund [17] [28]. Une application de la synthèse par placement des pôles est présentée en fin de chapitre.

2. Correction des systèmes :

Le rôle le plus important des correcteurs est la commande des processus pour avoir une sortie désirée. Le correcteur est sélectionné de façon à ce que les nouveaux caractéristiques du système en boucle fermée répondent à des critères de performances exigés. Les critères les plus importants dans une commande sont les marges de stabilité, le dépassement, le temps de réponse et l'erreur en régime permanent (l'erreur statique).

On peut distinguer deux approches essentielles pour la correction des systèmes, la première est basée seulement sur la valeur désirée à la sortie du processus (commande en boucle ouverte) et l'autre sur la valeur désirée mais aussi sur la sortie du système réellement obtenue (commande en boucle fermée).

2.1 Correction en boucle ouverte

Dans une correction en boucle ouverte, la grandeur de commande u ne dépend que du signal de référence w, et le correcteur ne reçoit pas d'informations sur la sortie réelle du système. Ce cas serait intéressant, si on sait qu'une commande quelconque appliquée au système va toujours engendrer la même réponse indépendamment du fait que les perturbations sont présentes ou non, ou de leurs importances (dans les cadres tolérés). L'avantage immédiat de cette approche réside dans le fait qu'un tel correcteur sera simple à concevoir en plus d'un coût très intéressant. Mais dans la majorité des cas réels l'image de la sortie est nécessaire pour le calcul de la commande adéquate. Ceci nous guide à l'introduction d'une boucle de retour qui donne cette image à un comparateur qui sert à calculer l'erreur.

Une application où on peut trouver ce type de correcteurs est la commande des moteurs pas à pas. Dans ce type de moteurs, pour chaque impulsion reçue par le moteur, ce dernier va tourner avec un angle spécifique quelle que soit sa charge (dans la limite de son fonctionnement). Donc suivant le nombre d'impulsions on connaît exactement sa position, sans la nécessité d'un capteur. On peut aussi le trouver dans les systèmes où les perturbations sont négligeables, où le contrôle précis n'est pas nécessaire.

2.2 Correction en boucle fermée

Le schéma classique d'une correction en boucle fermée est présenté dans la figure (1.1). Le régulateur, dont la fonction de transfert est désignée par $G_c(s)$, est situé en amont du système à régler $G_a(s)$.

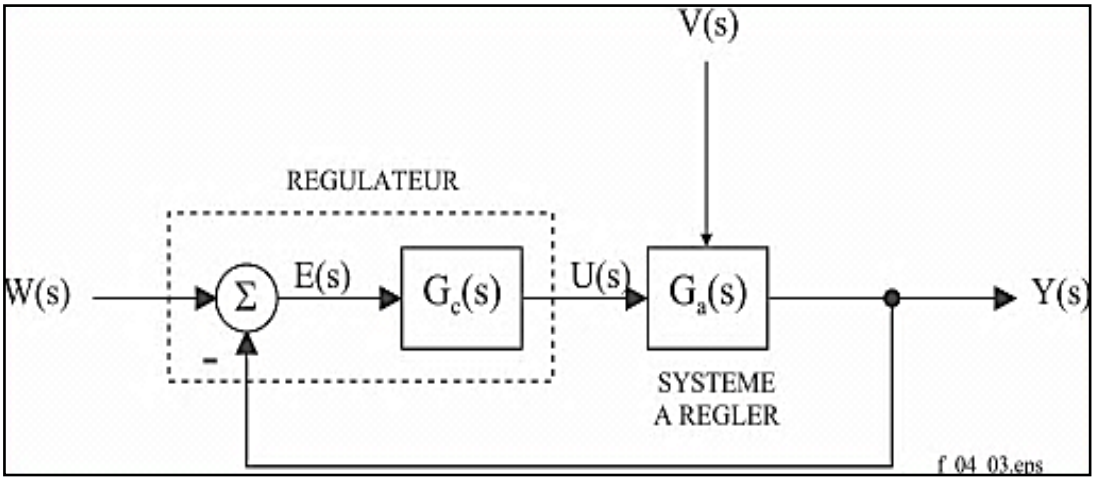


Fig.1.1 : Schéma fonctionnel d'un système asservi mono-variable (correction en boucle fermée).

Le système à régler $G_a(s)$ comprend, outre le processus, l'amplificateur de puissance, l'actionneur, le capteur et l'électronique de traitement de la mesure associée [19]. Appliquée au système à régler, la commande u(t) provoque donc une modification de la grandeur réglée y(t). Le régulateur en tenant compte pour former u(t), on constate que y(t) apparaît :

- à l'origine de l'action entreprise par le régulateur ;
- comme conséquence de cette action [19].

3. Description des régulateurs PID

Un régulateur PID remplit essentiellement trois fonctions :

- Il fournit un signal de commande u(t) en tenant compte de l'évolution du signal de sortie y(t) par rapport à la consigne w(t).
- Il élimine l'erreur statique grâce au terme intégrateur.
- Il anticipe les variations de la sortie grâce au terme dérivateur.

3.1 Le régulateur PID

Le régulateur PID classique relie directement le signal de commande u(t) au signal d'écart e(t). Sa description temporelle est la suivante :

$$u(t) = K_p.\left(e(t) + \frac{1}{T_i}.\int_{-\infty}^{t} e(\tau)\,d\tau + T_d.\frac{de(t)}{dt}\right) \quad (1.1)$$

avec l'écart défini comme suit :

$$e(t) = w(t) - y(t) \quad (1.2)$$

Sa fonction de transfert s'écrit :

$$G_c(s) = \frac{U(s)}{E(s)} = K_p.\,(1 + \frac{1}{sT_i} + sT_d) \quad (1.3)$$

Cette combinaison des termes P, I et D est aussi désignée sous le nom de forme parallèle ou non-interactive (Fig.1.2).

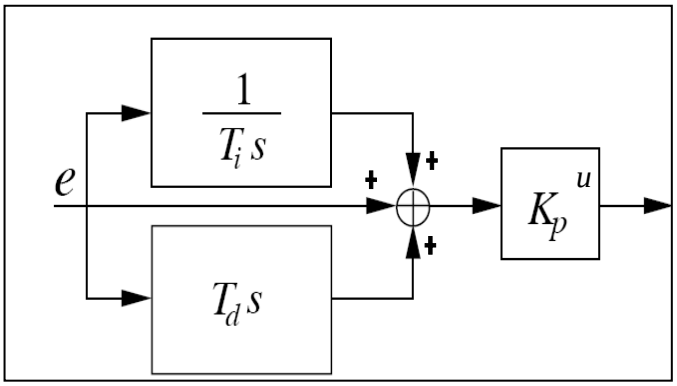


Fig.1.2: Schéma bloc d'un PID classique

3.2. Le régulateur PID modifié :

Les modifications que l'on souhaite apporter au régulateur PID sont de deux types :

- Afin de diminuer l'importance des dépassements tout en conservant un faible temps de réglage, on réduit l'effet de la consigne sur le terme proportionnel. L'écart est alors défini par :

$$e_p(t) = b.w(t) - y(t) \quad (1.4)$$

Suivant le dépassement souhaité, le paramètre b de pondération est choisi entre 0 et 1.

- Afin d'éviter de fortes amplitudes du signal de commande lors de brusques variations de la consigne, on diminue ou on annule l'effet de la consigne sur le terme dérivé :

$$e_d(t) = c.w(t) - y(t) \tag{1.5}$$

Généralement, le paramètre c est choisi nul [16].

La description temporelle du régulateur PID modifié est alors la suivante :

$$u(t) = K_p.\left(e_p(t) + \frac{1}{T_i}.\int_{-\infty}^{t} e(\tau)\,d\tau + T_d.\frac{de_d(t)}{dt}\right) \tag{1.6}$$

Tenant compte des définitions des trois écarts, le signal de commande u(t) s'écrit :

$$u(t) = K_p\left(bw(t) + \frac{1}{T_i}\int_{-\infty}^{t} w(\tau)\,d\tau + cT_d\frac{dw(t)}{dt}\right) - K_p\left(y(t) + \frac{1}{T_i}\int_{-\infty}^{t} y(\tau)\,d\tau + T_d\frac{dy(t)}{dt}\right) \tag{1.7}$$

dont la transformée de Laplace vaut :

$$U(s) = K_p\left(b + \frac{1}{sT_i} + csT_d\right)W(s) - K_p\left(1 + \frac{1}{sT_i} + sT_d\right)Y(s) \tag{1.8}$$

Le régulateur PID ainsi modifié fait apparaître deux fonctions de transfert associées à la consigne pour l'une, et au signal réglé pour l'autre :

$$G_{ff}(s) = \frac{U(s)}{W(s)} = K_p(b + \frac{1}{sT_i} + c\,sT_d) \tag{1.9}$$

$$G_c(s) = \frac{U(s)}{Y(s)} = K_p.(1 + \frac{1}{sT_i} + sT_d) \tag{1.10}$$

Le schéma fonctionnel d'un processus réglé à l'aide de ce régulateur est présenté dans la figure (1.3). On dit que ce régulateur possède deux degrés de libertés.

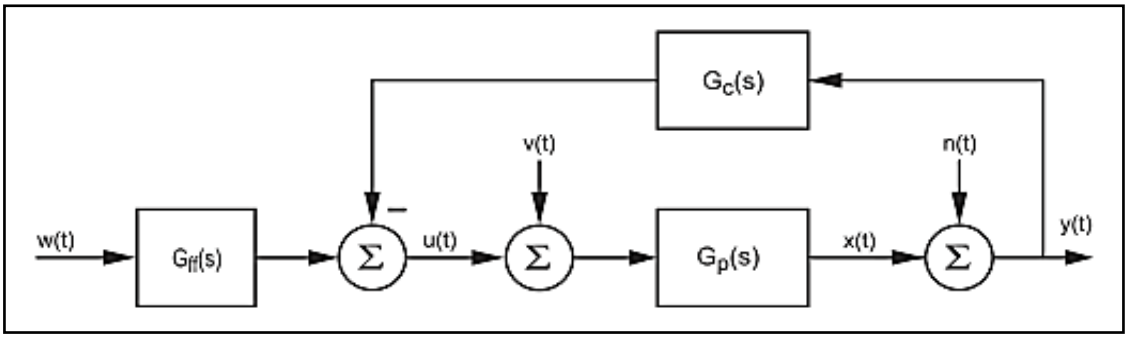


Fig.1.3: Schéma fonctionnel d'un processus réglé par un PID modifié

D'un point de vue pratique, deux autres modifications sont fréquemment apportées au régulateur PID :

- Afin de diminuer l'influence du bruit hautes-fréquences, on limite le gain HF du dérivateur en remplaçant le terme sT_d par la fonction de transfert suivante :

$$G_d(s) = \frac{sT_d}{1 + sT_d/N} \tag{1.11}$$

Généralement, on choisit N entre 5 et 20.

- En présence du terme intégral et dans le cas où l'écart tarde à s'annuler, la composante intégrale peut croître sans cesse et entraîner la saturation du convertisseur analogique numérique ou de l'actionneur commandant le processus. Puis, lors du changement de signe de l'écart, il faudra attendre que la composante intégrale décroisse suffisamment pour ne plus saturer les éléments qui se trouvent en aval. Pour éviter ce temps perdu, on introduit une limitation sur la valeur maximum que peut atteindre le terme intégral. On notera qu'avec cette modification, le régulateur ne peut plus être considéré comme linéaire.

Ce phénomène est dit « WIND-UP », et il apparait souvent quand la consigne subit de grands changements où il peut être la conséquence de grandes perturbations ou un mal fonctionnement des équipements. Aussi, le wind-up peut apparaitre quand on a des sélectionneurs utilisés avec quelques contrôleurs pour la commande d'un seul actionneur. Différentes méthodes existent dans la littérature de contrôle pour la résolution de ce problème [16] [21]. L'une de ces méthodes, est dite « calcul rétroactif et poursuite ». Son principe est comme suit :

Quand la sortie se trouve saturée, l'intégrale est recalculée de manière à ce que sa nouvelle valeur donne une sortie à la limite de saturation. C'est avantageux non pas pour la remise à zéro instantané de l'intégrateur mais pour sa remise à zéro dynamique avec une constante de temps T_t. La figure (1.4) montre un schéma bloc du contrôleur PID avec un anti-windup basé sur le calcul rétroactif.

Le système a un chemin de retour supplémentaire généré par la mesure de la sortie actuelle de l'actionneur et formant ainsi le signal d'erreur e_s comme la différence entre la sortie du contrôleur v et celle de l'actionneur u. le signal e_s est renvoyé vers l'entrée de l'intégrateur à travers le gain $1/T_t$. Le signal est nul quand il n'y a pas de saturation. Ainsi, elle n'aura aucun effet sur l'opération normale quand l'actionneur n'est pas saturé. Quand l'actionneur est saturé, le signal e_s est différent de zéro. Le chemin de retour normal autour du processus est brisé parce que son entrée se voit constante. Toutefois, il existe un chemin de retour autour de l'intégrateur. A cause de ceci, la sortie de l'intégrateur est guidée vers une valeur qui rend son entrée nulle. Ceci, empêche le phénomène du wind-up d'avoir lieu. Le temps pris par la sortie du contrôleur pour être mise à zéro par le gain de retour $1/T_t$, où T_t peut-être interprété comme la constante de temps, qui détermine la rapidité de la remise à zéro de l'intégrale. Ce temps est dit temps de poursuite.

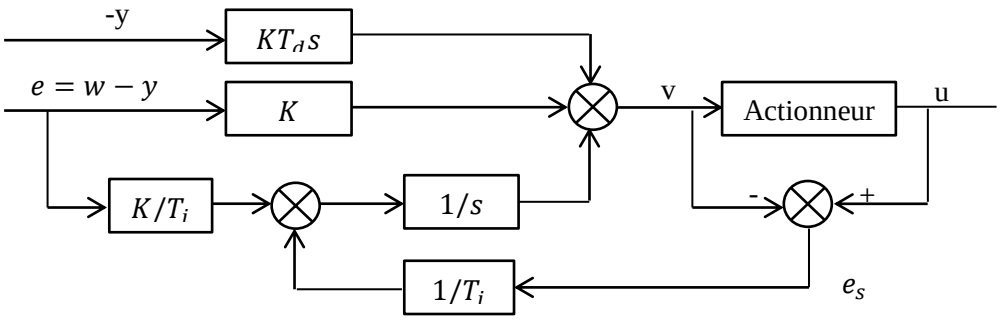


Fig.1.4 Schéma bloc du régulateur PID avec anti-windup

Remarque :

Fréquemment, la sortie de l'actionneur ne peut pas être mesurée. Le schéma du contrôleur avec anti-windup ainsi décrit, peut être appliqué à l'aide d'un modèle mathématique de l'actionneur saturé.

3.3 Le PID numérique

Afin d'implémenter le correcteur dans un calculateur numérique, il faut discrétiser les résultats obtenus en continu pour avoir la version numérique de la fonction du transfert. Donc, on doit discrétiser le terme intégrateur et le terme dérivateur présents dans l'équation (1.10).

Dans le cas où la période d'échantillonnage T_e est petite et le bruit dans le signal de sortie est bien filtré, on peut remplacer la dérivée par une différence du premier ordre :

$$\frac{de}{dt} \approx \frac{e(k) - e(k-1)}{T_e} = \frac{\Delta e(k)}{T_e} \tag{1.12}$$

Il existe une autre méthode pour discrétiser la dérivée, mais nous nous intéressons qu'à la méthode ainsi définie. Pour le terme intégral, il existe trois approximations. La première consiste à remplacer ce terme par une somme des rectangles inférieurs (FRM : Forward Rectangular Method), cela va donner :

$$\int_0^t e(\tau)\, d\tau \approx\ T_e \sum_{i=1}^{k} e(i-1) \tag{1.13}$$

Donc, l'équation du PID discrétisée devient :

$$u(k) = K_p \left\{ e(k) + \frac{T_e}{T_i} \sum_{i=1}^{k} e(i-1) + \frac{T_d}{T_e} [e((k) - e(k-1)] \right\} \tag{1.14}$$

Si le terme intégral est remplacé par une somme des rectangles supérieurs (BRM : Backward Rectangular Method) :

$$\int_0^t e(\tau)\, d\tau \approx\ T_e \sum_{i=1}^{k} e(i) \tag{1.15}$$

L'équation du PID discrétisé devient :

$$u(k) = K_p \left\{ e(k) + \frac{T_e}{T_i} \sum_{i=1}^{k} e(i) + \frac{T_d}{T_e} [e((k) - e(k-1)] \right\} \tag{1.16}$$

La dernière approximation utilise la méthode des trapézoïdes (TRAP : Trapezoid method), elle est la plus précise des trois. Le terme intégral est donné par :

$$\int_0^t e(\tau)\, d\tau \approx\ T_e \sum_{i=1}^{k} \frac{e(i) + e(i-1)}{2} \tag{1.17}$$

Ce qui donne l'équation suivante pour le PID discrétisé :

$$u(k) = K_p \left\{ e(k) + \left[\frac{e(0)+e(k)}{2} + \sum_{i}^{k-1} e(i)\right] + \frac{T_d}{T_e} [e((k) - e(k-1)] \right\} \tag{1.18}$$

Si la période d'échantillonnage est suffisamment petite, on ne remarque pas de différences (apparentes) entre les trois équations. En pratique l'équation (1.16) est la plus utilisée. Les trois équations (1.14, 1.16 et 1.18) sont de nature non-récursive, il faut connaître toutes les valeurs de l'erreur depuis le

début de l'opération de contrôle, ce n'est pas pratique. C'est pour cette raison qu'on utilise les versions récursives de ces trois équations [20]. La version récursive de l'équation (1.16) est :

$$u(k) = \Delta u(k) + u(k-1) \tag{1.19}$$

$$\Delta u(k) = K_p \left\{ e(k) - e(k-1) + \frac{T_e}{T_i} e(k) + \frac{T_d}{T_e} [e(k) - 2e(k-1) + e(k-2)] \right\} \tag{1.20}$$

Pour avoir une seule expression du PID discret pour les trois méthodes, on utilise l'équation suivante :

$$u(k) = q_0\ e(k) + q_1\ e(k-1) + q_2\ e(k-2) + u(k-1) \tag{1.21}$$

Donc pour implémenter l'équation (1.21) dans un calculateur, il faut remplacer les paramètres q_0, q_1 et q_2 par leurs valeurs numériques calculées à l'avance. Les paramètres q_0, q_1 et q_2 sont donnés dans ce tableau :

Paramètres du PID	FRM	BRM	TRAP
q_0	$K_p(1 + \frac{T_d}{T_e})$	$K_p(1 + \frac{T_e}{T_i} + \frac{T_d}{T_e})$	$K_p(1 + \frac{T_e}{2T_i} + \frac{T_d}{T_e})$
q_1	$-K_p(1 - \frac{T_e}{2T_i} + \frac{2T_d}{T_e})$	$-K_p(1 + \frac{2T_d}{T_e})$	$-K_p(1 + \frac{T_e}{2T_i} + \frac{2T_d}{T_e})$
q_2	$K_p \frac{T_d}{T_e}$	$K_p \frac{T_d}{T_e}$	$K_p \frac{T_d}{T_e}$

Tab.1.1 Expression des différents paramètres du PID pour les différentes méthodes d'approximation de l'intégral

4. Synthèse des correcteurs :

La synthèse d'un correcteur consiste à déterminer ses paramètres. Il existe plusieurs méthodes pour la faire. Les méthodes décrites ci-après sont : la synthèse par placement de pôles, la synthèse par la méthode de Ziegler et Nichols (temporelle et fréquentielle) et la synthèse par la méthode de Aström et Hägllund (temporelle et fréquentielle).

4.1 Synthèse par placement de pôles :

La méthode de synthèse considérée est basée sur le choix prédéterminé des pôles du système en boucle fermée, en d'autres termes le choix du polynôme caractéristique. Ce choix nous permet de sélectionner les caractéristiques du système à l'avance (stabilité, dépassement, amortissement, … etc.). Dans certains cas, le choix exclusif des pôles n'est pas suffisant pour avoir les caractéristiques nécessaires. Le numérateur du correcteur sera aussi choisi pour avoir des zéros désirés en boucle fermée, car les zéros eux aussi ont un rôle important dans la dynamique des systèmes. Considérons le schéma de la figure (1.5).

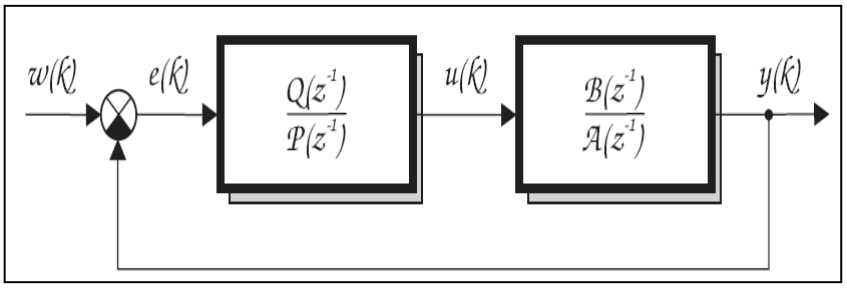


Fig.1.5 Schéma bloc pour la boucle de correction utilisée

Le polynôme caractéristique de la boucle fermée est déjà prédéterminé (calculé pour avoir les performances désirées), donc il suffit de calculer le polynôme caractéristique de la boucle fermée de la figure (1.5) en fonction des inconnues $Q(z^{-1})$ et $P(z^{-1})$, et d'égaliser les équations obtenues, pour avoir les expressions de $Q(z^{-1})$ et $P(z^{-1})$ [20].

D'après la figure (1.5), on peut écrire la fonction de transfert du processus, comme :

$$G_a(z) = \frac{Y(z)}{U(z)} = \frac{B(z^{-1})}{A(z^{-1})} = \frac{b_1 z + b_2}{z^2 + a_1 z + a_2} \tag{1.22}$$

La fonction de transfert du correcteur est donnée par :

$$G_c(z) = \frac{U(z)}{E(z)} = \frac{Q(z^{-1})}{P(z^{-1})} = \frac{q_0 + q_1 z^{-1} + q_2 z^{-2}}{(1 - z^{-1})(1 - \gamma z^{-1})} \tag{1.23}$$

La fonction de transfert du système en boucle fermée s'obtient de la manière suivante :

$$G_w(z) = \frac{Y(z)}{W(z)} = \frac{B(z^{-1})Q(z^{-1})}{A(z^{-1})P(z^{-1}) + B(z^{-1})Q(z^{-1})} \tag{1.24}$$

Le polynôme caractéristique du système en boucle fermée $D(z^{-1})$ est égal à :

$$D(z^{-1}) = A(z^{-1})P(z^{-1}) + B(z^{-1})Q(z^{-1}) \tag{1.25}$$

En imposant pour $D(z^{-1})$ l'expression suivante :

$$D(z^{-1}) = 1 + d_1\, z^{-1} + d_2\, z^{-2} \tag{1.26}$$

Avec

$$d_1 = -2\exp(-\xi\ \omega_n\ T_0)\cos\left(\omega_n\ T_0\sqrt{1-\xi^2}\right) \qquad pour \quad \xi \le 1$$

$$d_1 = -2\exp(-\xi\ \omega_n\ T_0)\cosh\left(\omega_n\ T_0\sqrt{1-\xi^2}\right) \qquad pour \quad \xi > 1$$

$$d_2 = \exp(-2\ \xi\ \omega_n\ T_0)$$

Où :

ξ : Coefficient d'amortissement dans l'équation caractéristique *D (z^{-1})*.

ω_n : Fréquence naturelle dans l'équation caractéristique *D (z^{-1})*.

En résolvant l'équation (1.25), nous aurons :

$$\begin{bmatrix} b_1 & 0 & 0 & 1 \\ b_2 & b_1 & 0 & a_1 - 1 \\ 0 & b_2 & b_1 & a_2 - a_1 \\ 0 & 0 & b_2 & -a_2 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \gamma \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{1.27}$$

Avec :

$$x_1 = d_1 + 1 - a_1$$
$$x_2 = d_2 + a_1 - a_2$$
$$x_3 = a_2$$
$$x_4 = 0$$

Où :

α : est un réel positif inférieur à 1. Il est utilisé pour contrôler la rapidité du système.

ω : est un réel qui vérifie la relation $\alpha^2 + \omega^2 < 1$. Il est utilisé pour varier le dépassement du système.

La loi de commande, à implémenter, peut être déduite à partir de l'équation aux différences, régissant le correcteur PID numérique, suivante :

$$u(k) = q_0\ e(k) + q_1\ e(k-1) + q_2\ e(k-2) + (1-\gamma)\ u(k-1) + \gamma\ u(k-2) \tag{1.28}$$

Les paramètres a_1, a_2, b_1 et b_2 sont calculés à partir du système d'équation (1.27). Ceci permettra, par la suite, de trouver les paramètres q_0, q_1, q_2 et γ de cette loi de commande.

4.2. La synthèse par la réponse indicielle et la réponse fréquentielle :

Il existe d'autres méthodes de synthèse qui se basent sur la réponse du processus, selon qu'elle soit temporelle ou fréquentielle. Avant d'entamer ces deux méthodes on présente ici les réponses d'un processus apériodique :

Réponse indicielle

La réponse indicielle d'un processus apériodique est illustrée par la figure (1.6). On y a mis en évidence les instants t_1, t_2, t_3 qui nous permettent de définir les grandeurs suivantes :

- le retard apparent :

$$L = t_1 \tag{1.29}$$

- la constante de temps apparente :

$$T = t_2 - t_1 \tag{1.30}$$

- la pente de la tangente au point d'inflexion :

$$p = \frac{y(\infty)}{t_3 - t_1} = \frac{a}{L} \tag{1.31}$$

- le temps mort relatif :

$$\tau = \frac{L}{L+T} = \frac{t_1}{t_2} \tag{1.32}$$

- Le rapport entre la valeur asymptotique y(∞) et l'amplitude E du saut appliqué en entrée détermine le gain statique k_0 du processus :

$$K_0 = \frac{y(\infty)}{E} \tag{1.33}$$

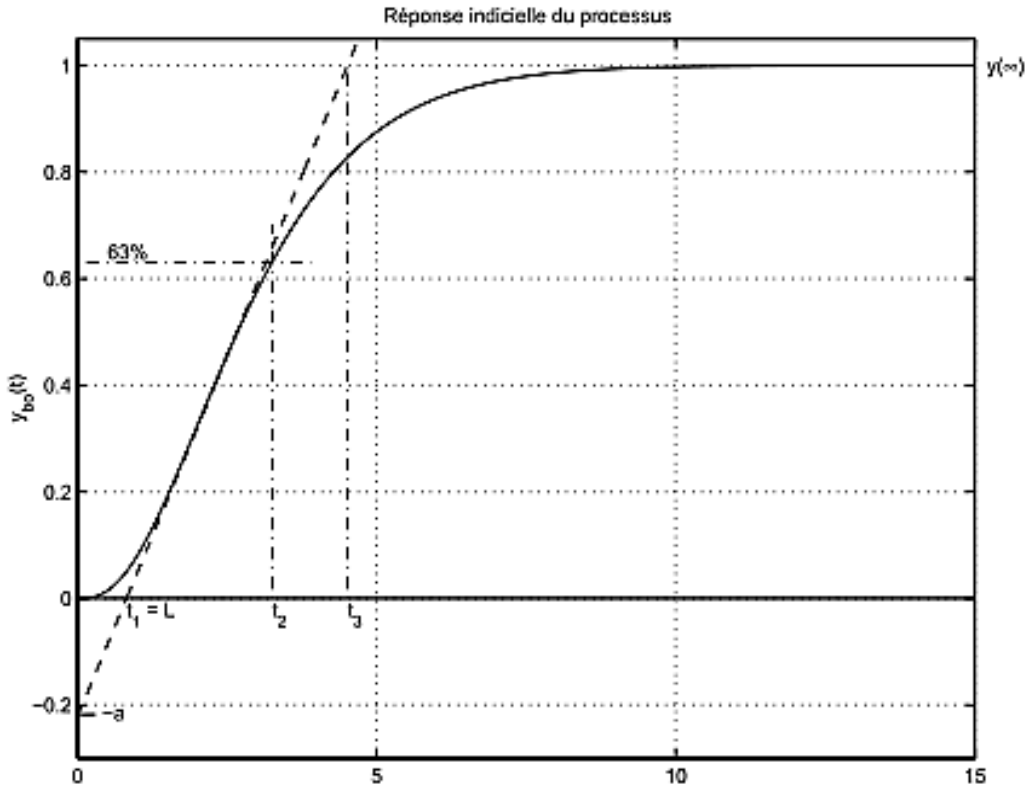


Fig.1.6:Réponce indicielle d'un processus apériodique

La réponse fréquentielle

La réponse fréquentielle d'un processus apériodique est illustrée par la figure (1.7). Sur cette réponse, on définit les grandeurs suivantes :

- la pulsation ω_π pour laquelle la phase vaut -180 °
- le gain G_π correspondant à cette pulsation
- le gain critique K_{cr} qu'il faut introduire dans le système bouclé pour le rendre instable :

$$K_{cr} = \frac{1}{G_\pi} \tag{1.34}$$

- le gain relatif :

$$k = \frac{G_\pi}{G(0)} = \frac{1}{K_{cr} K_0} \tag{1.35}$$

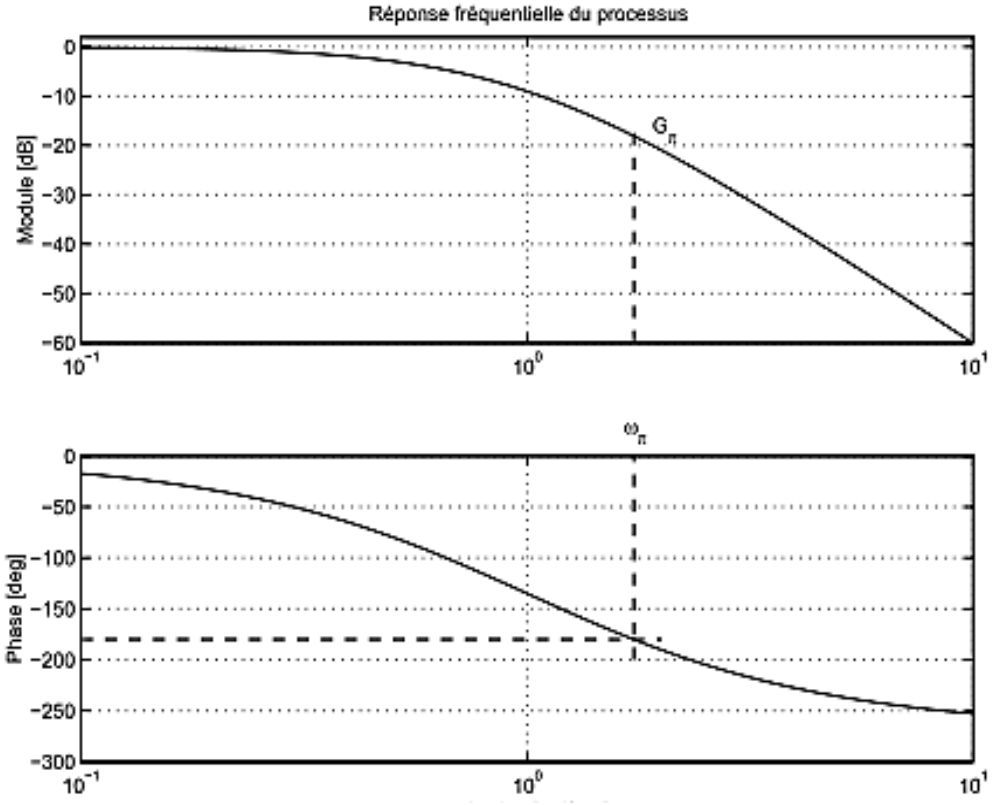


Fig.1.7 : Réponse fréquentielle d'un processus apériodique d'ordre 3

4.2.1 Méthodes de Ziegler et Nichols (ZN)

En 1942, Ziegler et Nichols [14] ont proposé deux approches heuristiques basées sur leur expérience et quelques simulations pour ajuster rapidement les paramètres des régulateurs P, PI et PID. La première méthode nécessite l'enregistrement de la réponse indicielle en boucle ouverte, alors que la deuxième demande d'amener le système bouclé à sa limite de stabilité.

a. Méthode de la réponse indicielle

Pour obtenir les paramètres du régulateur PID, il suffit d'enregistrer la réponse indicielle du processus seul (c'est-à-dire sans le régulateur), puis de tracer la tangente au point d'inflexion de la courbe. On mesure ensuite sa pente p, le retard apparent L correspondant au point d'intersection de la tangente avec l'abscisse et le gain $K_0 = y_\infty/E$ (Fig.1.6). On peut alors calculer les coefficients du régulateur choisi à l'aide du tableau (1.2). Généralement, les gains K_p proposés par Ziegler-Nichols sont trop élevés et conduisent à un dépassement supérieur à 20%. Il ne faut donc pas craindre de réduire K_p d'un facteur 2 pour obtenir une réponse satisfaisante.

Type	K_p	T_i	T_d
P	$1/(pLK_{0)} = 1/(aK_0)$		
PI	$0.9/(pLK_{0)} = 0.9/(aK_0)$	$3L$	
PID	$1.2/(pLK_{0)} = 1.2/(aK_0)$	$2L$	$0.5L$

Tab.1.2: Paramètres PID obtenus à partir d'une réponse indicielle (ZNt)

b. Méthode du point critique

Cette méthode est basée sur la connaissance du point critique du processus. Expérimentalement, on boucle le processus sur un simple régulateur proportionnel dont on augmente le gain jusqu'à amener le système à osciller de manière permanente ; on se trouve ainsi à la limite de stabilité. Après avoir relevé le gain critique K_{cr} du régulateur et la période d'oscillation T_{cr} de la réponse, on peut calculer les paramètres du régulateur choisi à l'aide du tableau (1.3). Ici également, les valeurs proposées conduisent à un temps de montée relativement court malheureusement assorti d'un dépassement élevé. Cette situation n'étant pas toujours satisfaisante, on peut être amené à corriger les coefficients proposés et, en particulier, à diminuer le gain.

On notera que les paramètres T_i et T_d proposés par les deux méthodes de Ziegler-Nichols sont dans un rapport constant égal à 4. Le régulateur possède donc deux zéros confondus valant :

$-1/(2Td) = -2/Ti$.

Type	K_p	T_i	T_d
P	$0.5K_{cr}$		
PI	$0.4K_{cr}$	$0.8T_{cr}$	
PID	$0.6K_{cr}$	$0.5T_{cr}$	$0.125T_{cr}$

Tab.1.3: Paramètres PID obtenus à partir du point critique (ZNf)

4.2.2 Méthodes de Aström et Hägglund (AH)

Au début des années 1990, Aström et ses collaborateurs ont testé une nouvelle approche afin de pallier les inconvénients de la méthode de Ziegler et Nichols. Ils ont, pour cela, analysé et simulé le comportement d'un grand nombre de systèmes à comportement non-oscillant en recherchant à les caractériser de manière simple. Pour des processus sans intégration, ils ont constaté que leur comportement dynamique pouvait être caractérisé à l'aide du temps mort relatif τ ou du gain relatif k définis par les équations (1.32) et (1.35).

Pour des processus avec intégration, ces deux caractéristiques ne suffisent pas pour déterminer les paramètres d'un régulateur PID. Une information supplémentaire est nécessaire.

Dans ce qui suit, on ne présentera que les résultats concernant des systèmes sans intégration. Pour les processus contenant une intégration, plus de détails et informations peuvent être trouvés dans l'ouvrage de Aström et Hägglund [16].

a. Critère M_s pour l'ajustage des paramètres

De manière à pouvoir mesurer la qualité des réponses temporelles et les comparer sur la base d'un critère objectif, Aström a défini un critère d'ajustage des paramètres du régulateur. Ce critère consiste à observer la fonction de sensibilité du processus réglé et à la limiter à un maximum valant 1.3 (+3dB) ou 2 (+6dB) [16]. Le maximum M_s de la fonction de sensibilité est défini comme suit :

$$M_s = max_{0<\omega<\infty} \left| \frac{1}{1+G_a(jw).G_c(jw)} \right| \tag{1.36}$$

où $G_a(j\omega)$ est la réponse fréquentielle du processus et $G_c(j\omega)$ celle du régulateur. Dans le diagramme de Nyquist, ce maximum Ms possède une interprétation intéressante. En effet, son inverse $1/M_s$ est la plus courte distance entre la courbe de Nyquist de $G_{a(jw)}G_c(jw)$ et le point critique -1. Cette distance peut être prise comme une mesure de la robustesse car elle nous indique de combien le processus peut changer sans causer d'instabilité.

b. Méthode de la réponse indicielle

L'approche de AH est similaire à celle de ZN si ce n'est qu'on y utilise un paramètre supplémentaire : la constante de temps apparente T. Celle-ci correspond au temps nécessaire pour que la réponse indicielle atteigne le 63% de sa valeur asymptotique moins le temps mort apparent L (Fig.1.6). Comme pour la méthode de Ziegler-Nichols, on définit deux paramètres qui serviront au calcul des termes P, I et D :

- le gain normalisé

$$K_n = K_0 \frac{L}{T} \tag{1.37}$$

- le temps mort relatif

$$\tau = \frac{L}{L+T} \tag{1.38}$$

Contrairement à Ziegler et Nichols qui proposaient simplement trois coefficients pour déterminer les termes du régulateur PID, les multiples essais conduits par Aström l'ont amené à relier les paramètres du régulateur au temps mort relatif τ au travers de fonctions ayant la forme :

$$f(\tau) = \alpha_0 . \exp(\alpha_1 \tau + \alpha_2 \tau^2) \tag{1.39}$$

Les paramètres α0, α1, α2 servant au calcul des paramètres K_p , T_i , T_d ont été obtenus par ajustage de courbes au sens des moindres carrés. Aström a alors pu construire les tableaux de paramètres pour les régulateurs PI et PID (Tab.1.4). L'expérience de AH a montré que les valeurs de T_i et T_d peuvent être calculées à partir de L ou de T. La valeur de cette dernière étant généralement plus précise que la précédente, on travaille de préférence avec elle.

PI	$M_s = 1.4$			$M_s = 2.0$		
	$\alpha 0$	$\alpha 1$	$\alpha 2$	$\alpha 0$	$\alpha 1$	$\alpha 2$
$K_n K_p$	0.29	-2.7	3.7	0.78	-4.1	5.7
T_i/L	8.9	-6.6	3.0	8.9	-6.6	3.0
T_i/T	0.79	-1.4	2.4	0.79	-1.4	2.4
b	0.81	0.73	1.9	0.44	0.78	-0.45
PID	$M_s = 1.4$			$M_s = 2.0$		
	$\alpha 0$	$\alpha 1$	$\alpha 2$	$\alpha 0$	$\alpha 1$	$\alpha 2$
$K_n K_p$	3.8	-8.47	7.3	8.4	-9.6	9.8
T_i/L	5.2	-2.5	-1.4	3.2	-1.5	-0.93
T_i/T	0.46	2.8	-2.1	0.28	3.8	-1.6
T_d/L	0.89	-0.37	-4.1	0.86	-1.9	-0.44
T_d/T	0.077	5.0	-4.8	0.076	3.4	-1.1
b	0.40	0.18	2.8	0.22	0.65	0.051

Tab.1.4 Paramètres des régulateurs PI et PID obtenus à partir d'une réponse indicielle

c. Méthode du point critique

L'approche de Aström-Hägglund est également similaire à celle de Ziegler-Nichols. Ils utilisent un paramètre supplémentaire : le gain statique K0 du processus. Comme pour la méthode de ZN, deux paramètres serviront pour le calcul des termes P, I et D :

- le gain relatif

$$k = \frac{G_\pi}{G(0)} = \frac{1}{K_{cr}.K_0} \tag{1.40}$$

- la période d'oscillation critique

$$T_{cr} = \frac{2\pi}{w_\pi} \tag{1.41}$$

De la même manière que pour la méthode temporelle, Aström relie les paramètres du régulateur au gain relatif k au travers de fonctions ayant la forme :

$$f(k) = \alpha_0.\exp(\alpha_1 k + \alpha_2 k^2) \tag{1.42}$$

Les paramètres a_0, a_1 et a_2 servant au calcul des termes T_i et T_d ont été obtenus par ajustage de courbes au sens des moindres carrés. Aström a alors pu construire les tableaux de paramètres pour les régulateurs PI et PID (Tab.1.5).

PI	$M_s = 1.4$			$M_s = 2.0$		
	$\alpha 0$	$\alpha 1$	$\alpha 2$	$\alpha 0$	$\alpha 1$	$\alpha 2$
K_p/K_{cr}	0.053	2.9	-2.6	0.13	1.9	-1.3
T_i/T_{cr}	0.90	-4.4	2.7	0.90	-4.4	2.7
b	1.1	-0.0061	1.8	0.48	0.40	-0.17
PID	$M_s = 1.4$			$M_s = 2.0$		
	$\alpha 0$	$\alpha 1$	$\alpha 2$	$\alpha 0$	$\alpha 1$	$\alpha 2$
K_p/K_{cr}	0.33	-0.31	-1.0	0.72	-1.6	1.2
T_i/T_{cr}	0.76	-1.6	-0.36	0.59	-1.3	0.38
T_d/T_{cr}	0.17	-0.46	-2.1	0.15	-1.4	0.56
b	0 .58	-1.3	3.5	0.25	0.56	-0.12

Tab.1.5: Paramètres des régulateurs PI et PID obtenus à partir du point critique

5. Le PID commercial :

Les PID commerciaux se diffèrent dans leurs structure de la loi de commande (standard, série , parallèle), le paramétrage, la limitation du gain à hautes fréquences (filtrage), et comment la consigne est introduite. Pour savoir comment ajuster un contrôleur PID il faut connaitre la structure et le paramétrage de l'algorithme de commande. Malheureusement, cette information est, souvent, indisponible dans le manuel du contrôleur. Les trois structures utilisées dans les contrôleurs PID commerciaux sont discutées dans le livre d'Astrom et Hüggland [16]. On rappelle que les paramètres a et b servent à pondérer la consigne. La limitation du gain à hautes fréquences du terme dérivation est exprimée par le choix du paramètre N (voir le paragraphe 3.2). Une comparaison entre quelques PID commerciaux est montrée dans le tableau suivant :

Contrôleur	**Structure**	**b**	**C**	**Limitation du gain N**	**Temps d'échantillonnage (Te)**
Allen Bradelly PLC5	Standard, parallèle	1.0	1.0	*pas*	*Dépend de la charge*
Bailey Net 90	Série, parallèle	0.0 ou 1.0	0.0 ou 1.0	10	0.25
Toshiba TOSDIC 200	série	1.0	1.0	3.3-10	0.2
Turnbull TCS 6000	série	1.0	1.0	*pas*	0.036-1.56
Yokogawa SLPC	standard	0.0 ou 1.0	0.0 ou 1.0	10	0.1S

Tab.1.6: Propriétés des algorithmes PID dans quelques contrôleurs commerciaux

Remarque :

On remarque que les paramètres du contrôleur se différent d'un constructeur à un autre, mais surtout l'absence du filtre pour le gain de l'action D dans quelque contrôleurs.

6. Application :

Dans ce paragraphe nous considérons la commande en vitesse d'un moteur à courant continu par un régulateur PID. Les paramètres de ce dernier sont déterminés en utilisant la synthèse par placement de pôles.

6.1. Modélisation du MCC :

Un moteur à courant continu est une machine électrique. Il s'agit d'un convertisseur électromécanique permettant la conversion bidirectionnelle d'énergie entre une installation électrique parcourue par un courant continu et un dispositif mécanique, d'où l'énergie électrique est transformée en énergie mécanique. Dans la suite nous considérons le moteur à excitation séparée. Son schéma équivalent est donné par la figure suivante.

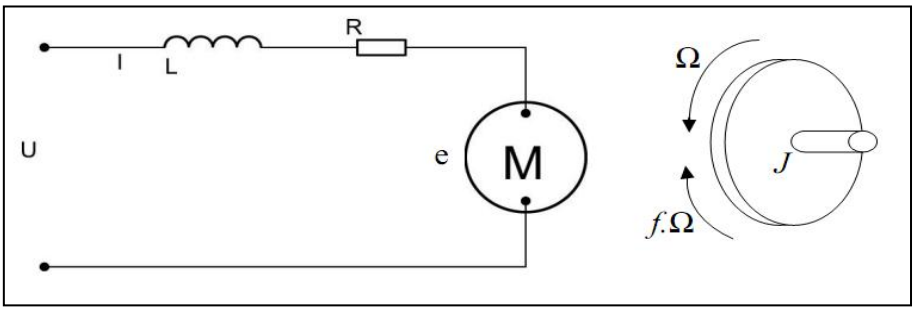


Fig.1.8 : Schéma équivalent d'un moteur à courant continu

Un moteur électrique à courant continu est régit par les équations physiques découlant de ses caractéristiques électriques, mécaniques et magnétiques. D'après la loi de Newton, combiné à des lois de Kirchhoff, on peut écrire les équations différentielles de premier ordre suivantes :

$$u(t) = Ri(t) + L\frac{di(t)}{dt} + e(t) \tag{1.43}$$

$$e(t) = K_e\Omega(t) \tag{1.44}$$

D'après le principe fondamental de la dynamique on a :

$$J\frac{d\Omega}{dt} = C_u - C_r \tag{1.45}$$

$$C_u = K_c i(t) - C_p \tag{1.46}$$

$$C_r = f\,\Omega(t) \tag{1.47}$$

u(t) : Tension appliquée au moteur — C_p *:* Couple de pertes

e(t) : Force contre électromotrice — C_u : Couple moteur généré

i(t) : Intensité traversant le moteur — C_r *:* Couple résistant

$\Omega(t)$ *:* Vitesse de rotation du rotor — K_e *:* Constante de vitesse

f : Coefficient de frottement visqueux — *J :* Moment d'inertie du rotor

K_C : Constante de couple

En pratique, on peut confondre les constantes K_c et K_e et poser $K_c \approx K_e = K$

Le passage dans le domaine de Laplace permet d'écrire :

$$U(s) = \Omega(s)\left(\frac{(R+Ls)(f+Js)}{K} + K\right) + \frac{R+Ls}{f+Js}\,Cp \tag{1.48}$$

On suppose que le moment du couple de pertes (qui est vu comme une perturbation) soit négligeable devant le moment du couple électromagnétique $(K_c i(t))$. On peut alors prendre C_p nul pour simplifier le système.

On obtient donc :

$$U(s) = \Omega(s)\left(\frac{(R+Ls)(f+Js)}{K} + K\right) \tag{1.49}$$

La fonction de transfert recherchée H(s) est entre la tension entrant dans le moteur U(s) et la vitesse de sortie Ω (s) [22].

$$H(s) = \frac{\Omega(s)}{U(s)} = \frac{K}{(R+Ls)(f+Js)+K^2} \tag{1.50}$$

Ou encore :

$$H(s) = \frac{\frac{K}{JL}}{s^2 + \frac{RJ+Lf}{JL}s + \frac{Rf+K^2}{JL}} \tag{1.51}$$

Il faut noter que si le moteur est équipé de réducteur mécanique on doit ajouter le moment d'inertie de ce dernier pour pouvoir modéliser tout le système tel que :

$$\mathrm{J} = \mathrm{J_m} + \mathrm{J_r} \tag{1.52}$$

J_m: Moment d'inertie rapporté au rotor du moteur

J_r : Moment d'inertie rapporté à l'axe du réducteur

Aussi la vitesse rapportée sur l'axe du réducteur est donnée par :

$$\Omega_\mathrm{r}(\mathrm{s}) = \Omega(\mathrm{s}).\,\mathrm{ratio} \tag{1.53}$$

$\Omega_\mathrm{r}(s)$: vitesse rapportée sur l'axe du réducteur

$ratio$: Rapport de réduction

6.2. Identification des paramètres du moteur utilisé :

Vu que nous n'avons pas les valeurs numériques des paramètres du constructeur, nous avons procédé à l'identification du modèle du moteur à courant continu utilisé. Pour cela nous avons utilisé le signal d'excitation donné par la figure (1.9), l'algorithme des moindres carrés récursifs (RLS : Recursive Least Squares) et le modèle donné par l'équation (1.22). L'évolution des valeurs des paramètres de ce modèle est illustrée par la figure (1.10). Nous obtenons le modèle numérique suivant :

$$H(z) = \frac{0.18\,z + 1.01}{z^2 - 1.024\,z + 0.09} \tag{1.54}$$

Ce modèle va servir dans la simulation et l'implémentation.

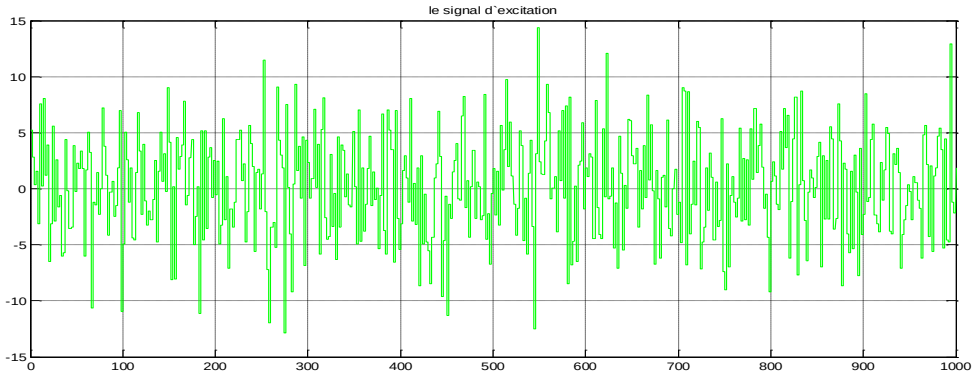


Fig.1.9 (a) :Le signal d'excitation utilisé pour l'identification

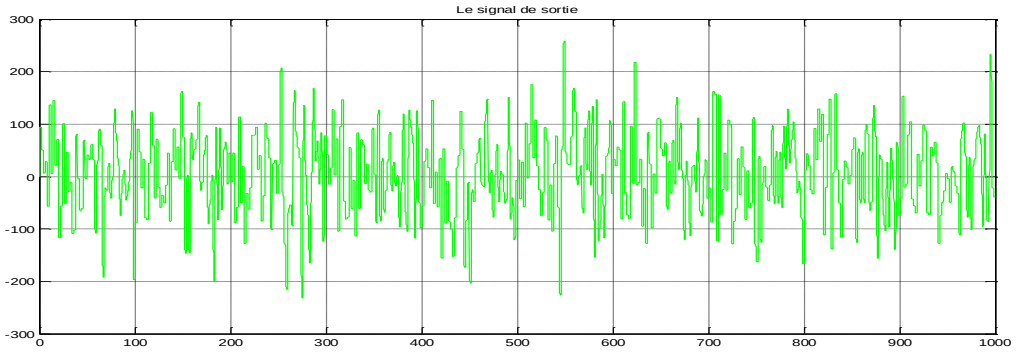


Fig.1.9 (b) :le signal de sortie correspondant au signal d'excitation

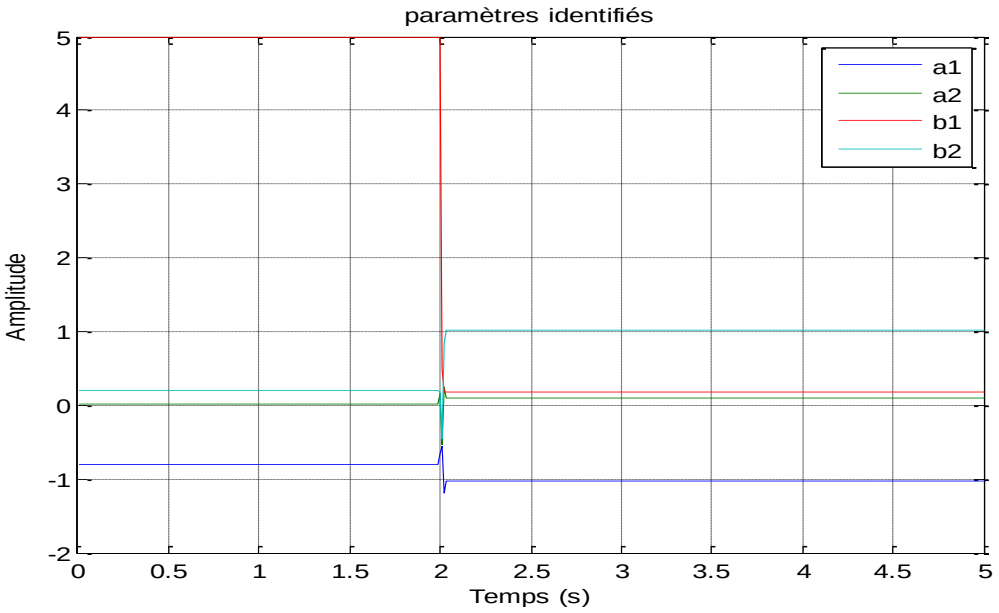


Fig.1.10 :Paramètres du moteur identifiés par l'algorithme RLS

6.3. Mise en œuvre du régulateur et résultats de simulation

Dans ce qui suit nous allons présenter quelques résultats de la commande en vitesse de la MCC. La figure suivante constitue le schéma de base pour cette commande :

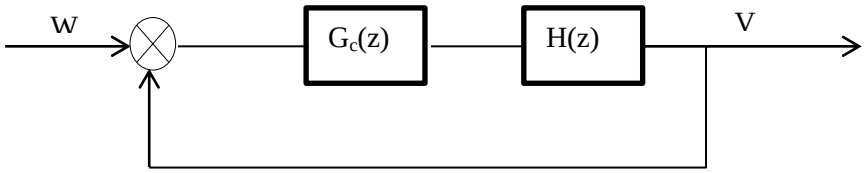


Fig.1.11**:**Schéma bloc de la commande en vitesse du MCC

On distingue le régulateur PID désigné par sa fonction de transfert Gc(z), et le moteur par H(z)

a. Calcul de La loi de commande

Les paramètres de la loi de commande donnée par l'équation (1.28) doivent être calculés de telle manière à satisfaire aux performances exigées. Ils peuvent être calculés à partir des équations (1.27) et (1.28). Si, par exemple, les performances exigées sont celles d'un système du deuxième ordre de pulsation propre $\omega_n = 10\ \mathrm{rad/s}$ et de coefficient d'amortissement $\varepsilon = 0.7$ nous obtenons, alors, pour la période d'échantillonnage $\mathrm{Te} = 10\mathrm{ms}$:

$$u(k) = 0.0739e(k) - 0.0795e(k-1) + 0.0134e(k-2) + 0.8493u(k-1) + 0.1507u(k-2) \quad (1.55)$$

Remarque :

Pour garantir la stabilité du système il faut que :

$$\begin{cases} \omega_n > 0 \\ \varepsilon > 0 \end{cases} \quad (1.56)$$

b. Résultats de Simulation

Nous considérons ici la simulation de la commande du moteur sans charge. En premier lieu on fixe la pulsation propre et on varie le facteur d'amortissement (simulation 1,2), puis en varie la pulsation propre (simulation 3). Le signal de référence est la séquence : 0, 30, 80, 180, 80, 0 (tr/mn).

Simulation 1 :

Les résultats de la première simulation sont donnés par la figure (1.12).

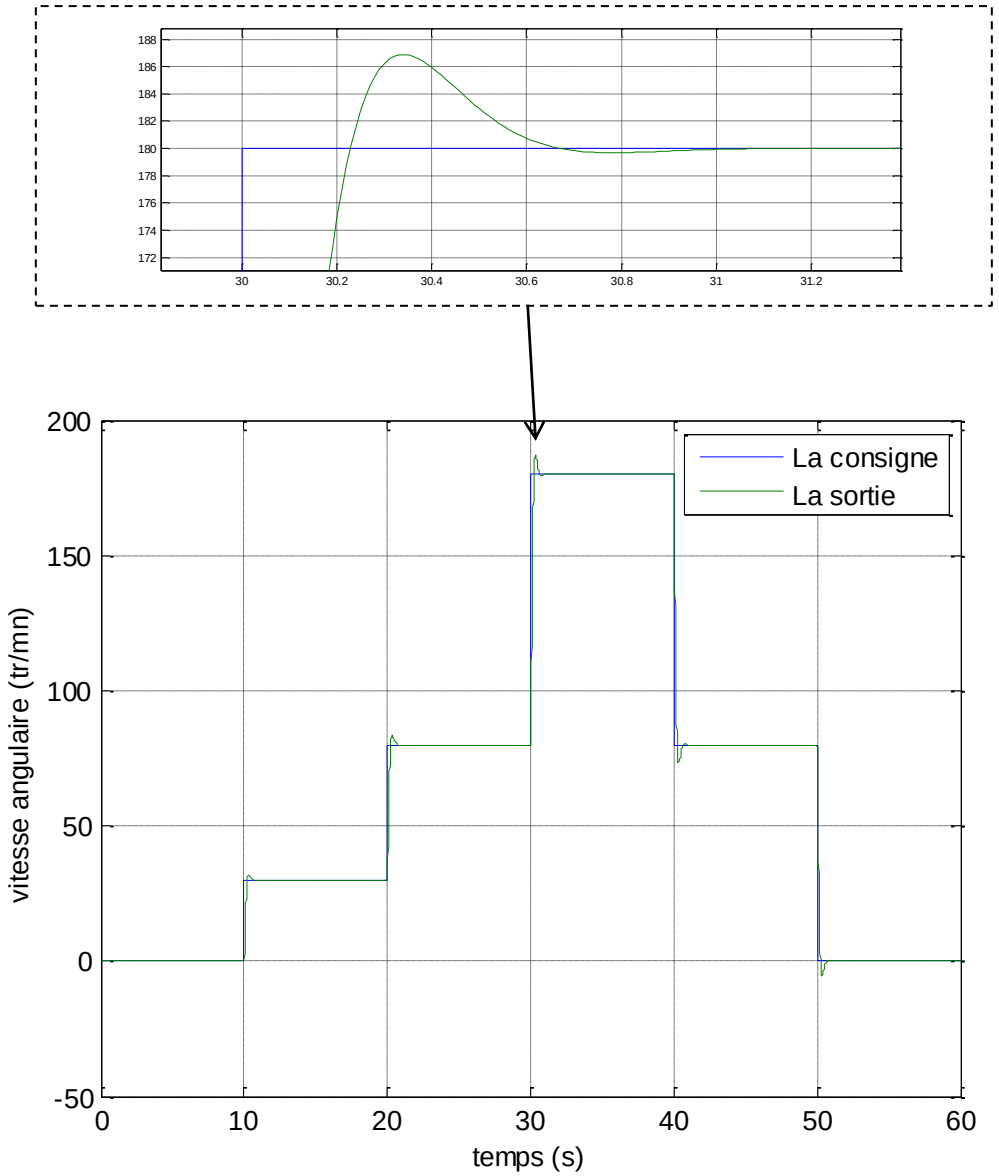


Fig.1.12 : Tracés de la sortie et de la référence pour$\omega_n = 10\ rad/s$ et $\varepsilon = 0.7$

Nous pouvons remarquer ici que la porsuite de la référence est presque parfaite, avec un dépassement égal à 6.87% et un temps de réponse proche de 500 ms. La loi de commande est calculée pour $\omega_n = 10\ \mathrm{rad/s}$ et $\varepsilon = 0.7$.

Simulation 2 :

Nous calculons la loi de commande pour $\omega_n = 10\ \mathrm{rad/s}$ et $\varepsilon = 0.2$. Les résultats obtenus sont donnés par la figure (1.13). Nous constatons ici que la diminution du facteur d'amortissement provoque une dégradation considérable des performances, avec un dépassement égal à 33% et un temps de réponse proche de 1.12 s.

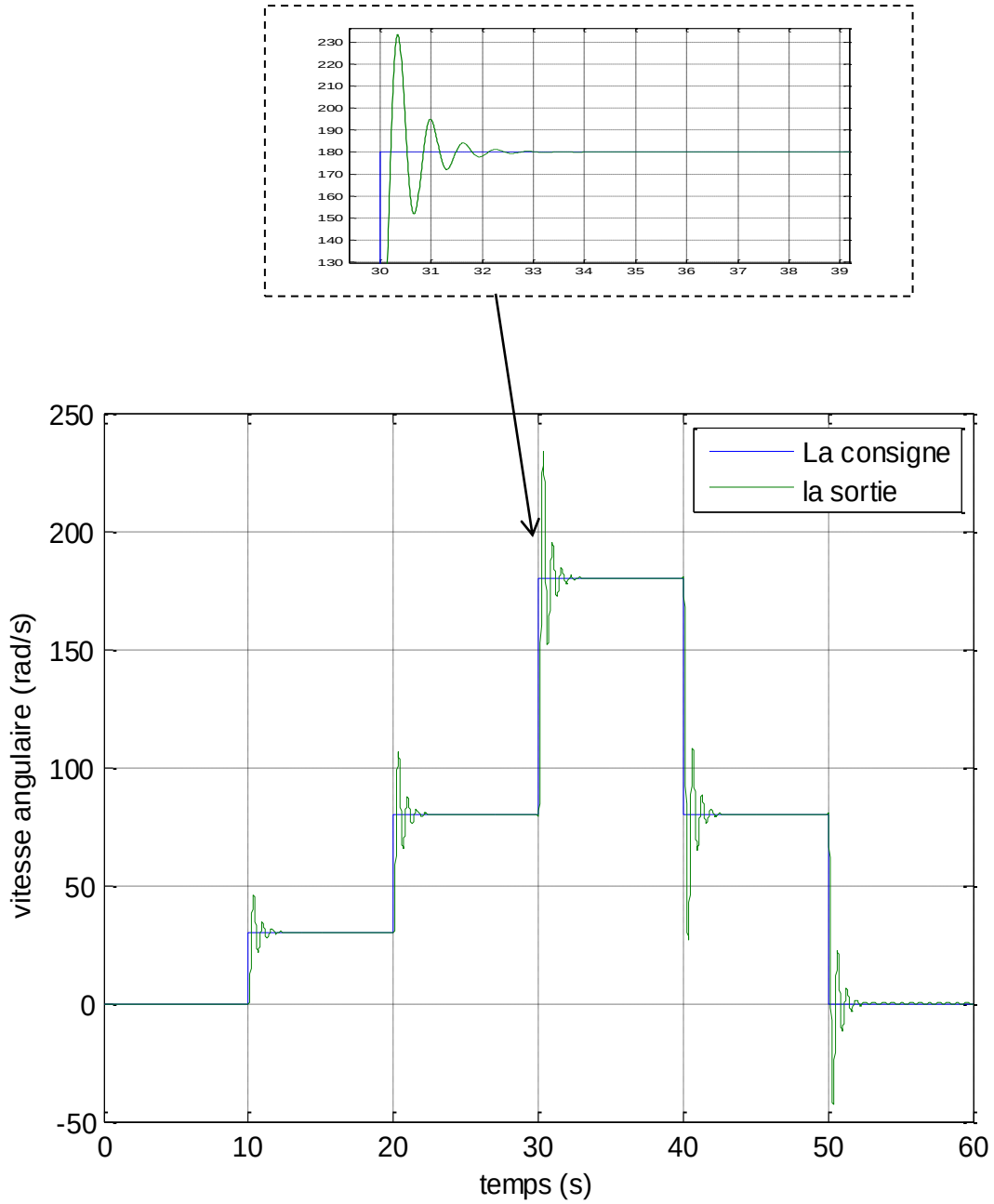


Fig.1.13 : Tracés de la sortie et de la référence pour $\omega_n = 10\, rad/s$ et $\varepsilon = 0.2$

Simulation 3 :

Dans cette simulation nous calculons la loi de commande pour $\omega_n = 50\, rad/s$ et $\varepsilon = 0.7$. Les résultats obtenus sont donnés par la figure (1.14). Nous constatons ici que le changement de la pulsation propre a provoqué une dégradation considérable des performances illustrée par un dépassement égal à 70% et un temps de réponse proche de $330ms$

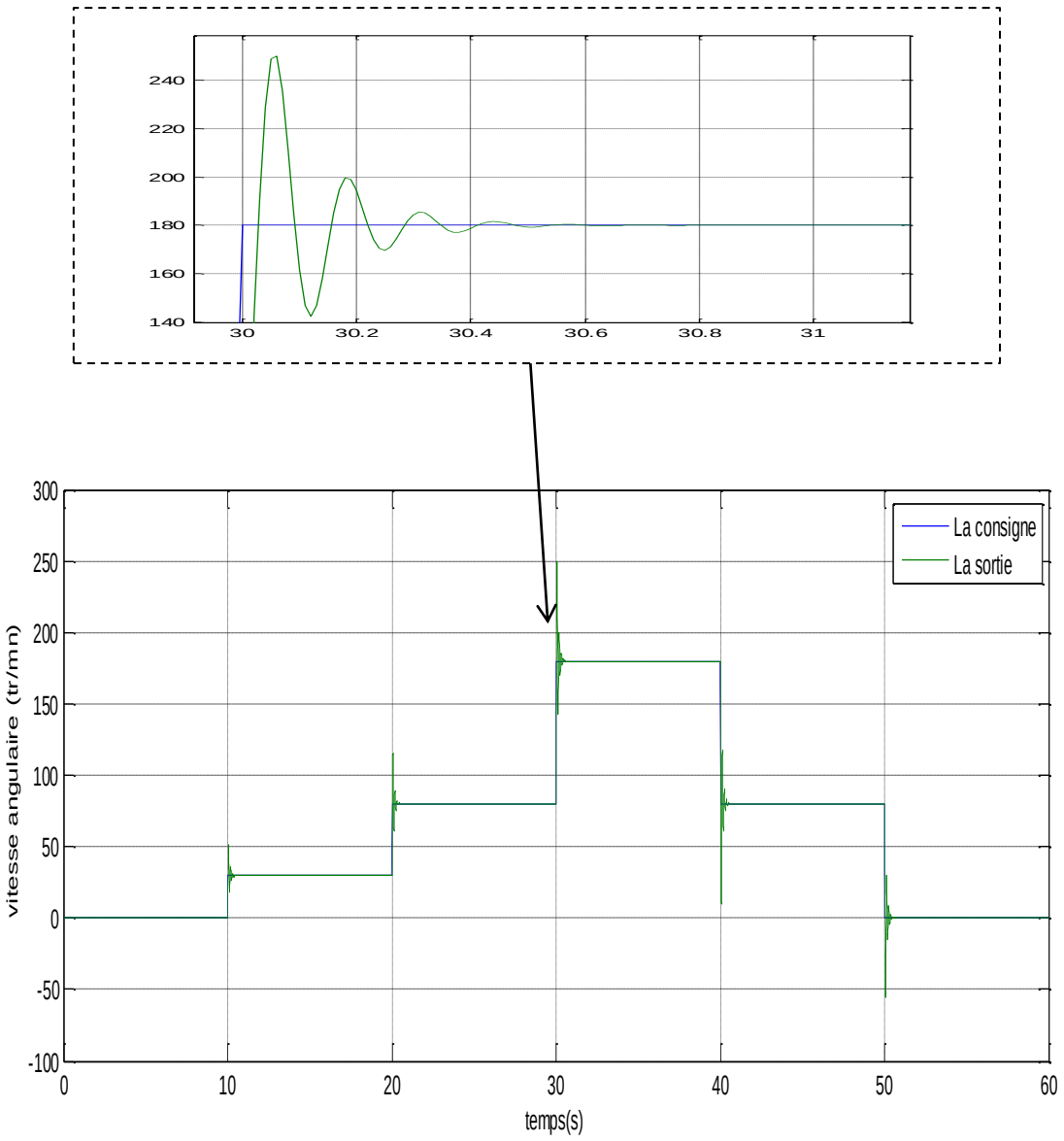


Fig.1.14 : Tracés de la sortie et de la référence pour $\omega_n = 50\, rad/s$ et $\varepsilon = 0.2$

Le tableau suivant regroupe les différents résultats de simulation obtenus :

Simulation	Dépassement (%)	Temps de réponse (ms)
$\boldsymbol{\omega_n = 10\, rad/s}$ **et** $\boldsymbol{\varepsilon = 0.7}$	6.87	440
$\boldsymbol{\omega_n = 10\, rad/s}$ **et** $\boldsymbol{\varepsilon = 0.2}$	33	1120
$\boldsymbol{\omega_n = 50\, rad/s}$ **et** $\boldsymbol{\varepsilon = 0.7}$	70	330

Tab.1.7 : Les performances tirées de chaque simulation

7. Conclusion :

Dans ce chapitre, nous avons présenté une étude des correcteurs PID. En fait, nous avons présenté les différentes formes de ces régulateurs analogiques, les transformations qui permettent d'obtenir leur version numérique et les différentes méthodes de synthèse qui puissent être utilisées pour déterminer leurs paramètres.

Un des intérêts du régulateur PID, qui explique sa popularité dans le milieu industriel, est sans conteste la possibilité de le régler sans connaissance approfondie du système. En effet, on dispose de méthodes empiriques, fondées uniquement sur la réponse temporelle, ou fréquentielle du système, selon une procédure expérimentale, comme la méthode d'oscillation de Ziegler-Nichols, permettant dans la majorité des cas d'aboutir à des performances acceptables. L'application considérée et les résultats de simulation obtenus montrent l'efficacité et la simplicité de ce type de correcteurs.

Références

[14] J.G. Ziegler, N.B. Nichols : « Optimum settings for automatic controllers. »Trans. ASME, 64, pp. 759-768, 1942.

[2] I.M Horowitz (1963) : Synthesis of Feedback Systems. Acaddemic Press, New-York.

[3] K.J. Aström, T. Hägglund: PID Controllers : Theory, Design and Tuning, Instrument Society of America, 2nd edition, 1995.

[4] K.J. Aström, T. Hägglund, C.C. Hang, W.K. Ho: « Automatic Tuning and Adaptation for PID Controllers - A Survey. », Control Engineering Practice, 1:4, pp.699-714, 1993.

[5] K.J. Aström, T. Hägglund (1984): «Automatic Tuning of Simple Regulators with Specification on Phase and Amplitude Margin. », Automatica, 20, pp. 645-651, 1984.

[6] Michel Etique : « Régulation automatique », mars 2006, Yverdon- les Bains.

[7]N.Bobal J.Bohn,J.Fessel,J.Machacek, « Digital Self-tuning Contollers »,Edition Wiley ,2010

[8] Michael A. Johnson Mohammad H. Moradi « PID Control New Identification and Design Methods», Springer-Verlag London Limited 2005

[9] B. Deforge & Q. David «Asservissement en position d'un axe linéaire», projet d'automatique, 2008.

Chapitre 3

Les Circuits Programmables

FPGAs

1. Introduction

Le développement d'un système complexe sur *FPGA* est un processus compliqué qui consiste à faire plusieurs transformations complexes et à optimiser les algorithmes qui vont être implémenté. Afin d'automatiser quelques tâches, des outils logiciels ont été introduite sur le marché par les différents constructeurs des *FPGAs* [33]. Nous utilisons la version *Xilinx ISE 12.3* pour la synthèse, la simulation et l'implémentation, et nous utilisons le *System Generator* pour la simulation. Dans ce chapitre, nous donnons une brève description des circuits *FPGAs* et de la carte de développement *SPARTAN-3E,* nous introduisons les outils de développement *ISE* et *System Generator* et nous présentons une vue globale du langage de description *VHDL* et ses règles de base.

2. Les FPGAs

2.1.Les circuits logiques programmables

Un circuit logique programmable (*PLD : Programmable Logic Device*) est un assemblage d'opérateurs logiques combinatoires et de bascules dans lequel la fonction réalisée n'est pas fixée lors de la fabrication. Il contient potentiellement la possibilité de réaliser toute une classe de fonctions, plus ou moins large suivant son architecture.

L'apparition de ce type de circuit s'est d'abord faite avec les circuits logiques programmables simples de type *PAL* (*Programmable Array Logic*), qui se programment comme des mémoires non volatiles de type *ROM* et sont utilisés pour implémenter des fonctions combinatoires simples (décodeurs d'adresse, contrôleurs de bus,…). Avec les développements de la micro-électronique il y a eu apparition des différents familles de circuits programmables : les *CPLD* (*Complex Logic*

Programmable Device), puis les *FPGA* (Field Programmable Gate Arrays), introduits par la société *Xilinx* en 1985. Les circuits de type *FPGA* les plus récents offrent désormais l'équivalent des millions de portes logiques programmables.

Il existe plusieurs familles de *PLDs* qui sont différenciées par leur structure interne, la plus ancienne et la plus connue est la famille des *PAL*.

a. Les P.A.L

Les *PAL*s (*Programmable Array Logic*) ou réseau logique Programmable sont les premiers circuits programmables à être utilisés pour réaliser des fonctions logiques. Ils possèdent des matrices « *ET* » programmables et des matrices « *OU* » fixes. La programmation de ces circuits s'effectue par destruction de fusibles. Une fois programmés, on ne peut plus les effacer. La structure de base de ce *PLD* est présentée par le schéma suivant:

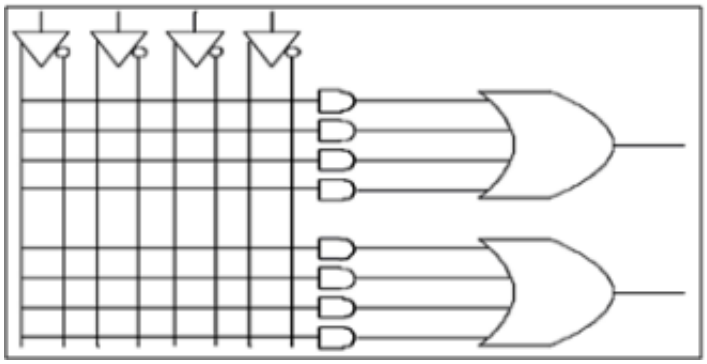


Fig.3.1 Structure de base d'un PAL

b. Les P.A.L. effaçables (E.P.L.D.)

Les *EPLDs* (*Erasable Programmable Logic Device)*, ou circuit logique programmable et effaçable peuvent être effacés par ultraviolet ou électriquement. Ils sont encore appelés *PALCMOS*.

c. Les G.A.L.

Les GAL (Generic Array Logic) ou encore réseau logique générique. Leur fonctionnement est identique aux *PALCMOS*, ils sont programmables et effaçables électriquement.

d. Les C.P.L.D.

Les *CPLDs* (*Complex Programmable Logic Device*) sont composés de plusieurs *PALs* élémentaires reliés entre eux par une zone d'interconnexion. Grâce à cette architecture, ils permettent d'atteindre des vitesses de fonctionnement élevées (plusieurs centaines de *MHz*). La figure suivante montre la structure générale d'un *CPLD*.

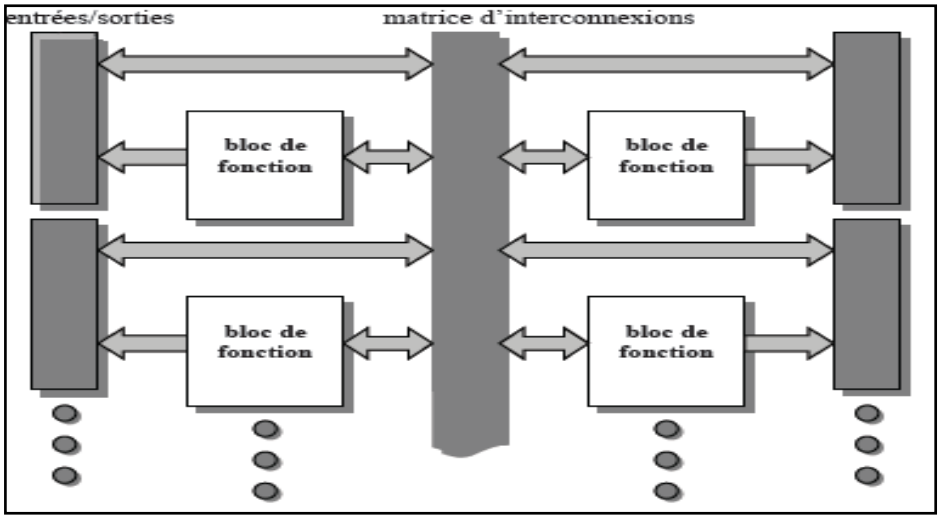


Fig. 3.2 : Structure générale d'un CPLD

e. Les F.P.G.A.

Un *FPGAs* (*Field Programmable Gate Array)* ou réseau de portes programmable par l'utilisateur. C'est un ensemble de blocs logiques élémentaires que l'utilisateur peut interconnecter pour réaliser les fonctions logiques de son choix. On distingue deux types :

- *FPGA* à anti-fusible : programmables électriquement par l'utilisateur, non effaçables

-*FPGA* à *SRAM* : ou encore *LCA* (Logic Cell Array) à base des cellules *SRAM* pour configurer les connexions entre les blocs logiques.

La figure suivante illustre un résumé graphique des familles des circuits logiques programmables.

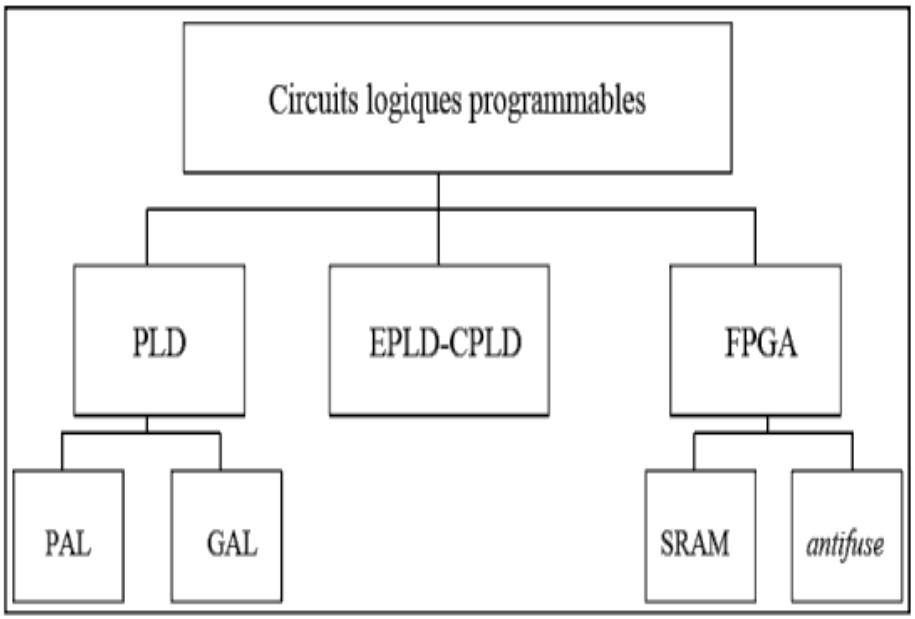


Fig. 3.3 : Les circuits logiques programmables

2.2. Les technologies d'interconnexion

Le premier critère de choix d'un circuit programmable est la technologie utilisée pour matérialiser les interconnexions:

a. Fusibles

La connexion par fusibles, la première méthode employée, est en voie de disparition. On ne la rencontre plus que dans quelques circuits de faible densité et de conception ancienne.

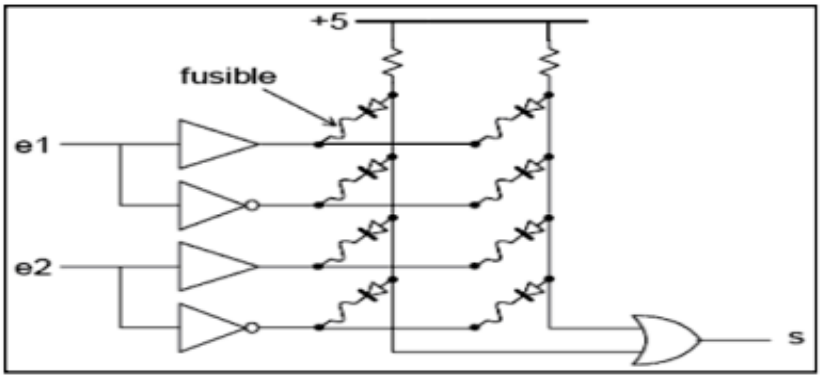


Fig. 3.4 PLD élémentaire à fusibles

La figure (3.4) en illustre le principe ; toutes les connexions sont établies à la fabrication. Lors de la programmation le circuit est placé dans un mode particulier par le programmateur, mode dans lequel des impulsions de courant sont aiguillées successivement vers les fusibles à détruire. Pour programmer un circuit, il faut transférer dans le programmateur une table qui indique par un chiffre binaire l'état de chaque fusible.

b. MOS à grille flottante

Les transistors MOS sont des interrupteurs, commandés par une charge électrique stockée sur leur électrode de grille. Si, en fonctionnement normal, cette grille est isolée, elle conserve sa charge éventuelle éternellement. Il reste au fondeur à trouver un moyen de modifier cette charge, pour programmer l'état du transistor. Le dépôt d'une charge électrique sur la grille isolée d'un transistor fait appel à un phénomène connu sous le nom d'effet tunnel : un isolant très mince soumis à une différence de potentiel suffisamment grande est parcouru par un courant de faible valeur, qui permet de déposer une charge électrique sur une électrode normalement isolée. Ce phénomène, réversible, permet de programmer et d'effacer une mémoire. La figure (3.5) montre la structure du PLD élémentaire précédent, dans lequel les fusibles sont remplacés par des transistors à grille isolée (technologie FLASH).

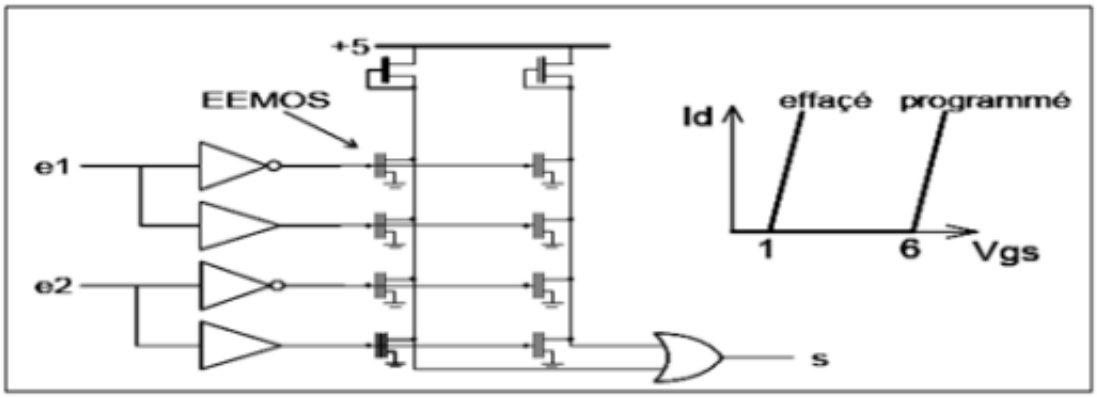


Fig. 3.5 : PLD simple à MOS

c. Mémoires statiques (SRAM)

Dans les circuits précédents, la programmation de l'état des interrupteurs, conservée en l'absence de tension d'alimentation, fait appel à un mode de fonctionnement électrique particulier. Dans les technologies à mémoire statique, l'état de chaque interrupteur est commandé par une cellule mémoire classique à quatre transistors (plus un transistor de programmation), dont le schéma de principe est celui de la figure suivante :

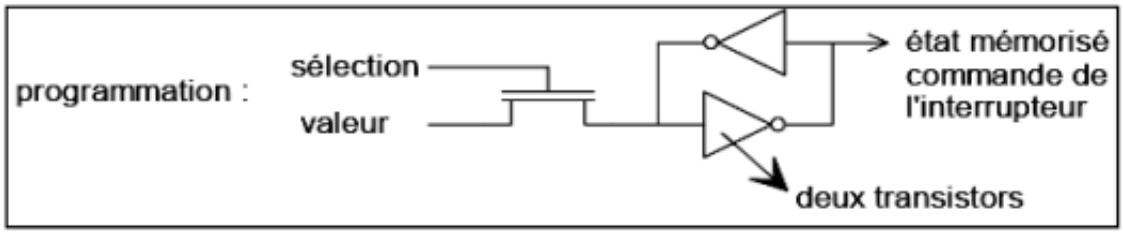


Fig.3.6 Cellule SRAM

d. Anti-fusibles

L'inverse d'un fusible est un anti-fusible. Le principe est, à l'échelle microscopique, celui de la soudure électrique par points. Un point d'interconnexion est réalisé au croisement de deux pistes conductrices (métal ou semi-conducteur selon les procédés de fabrication), séparées par un isolant de faible épaisseur. Une surtension appliquée entre les deux pistes provoque un perçage définitif du diélectrique.

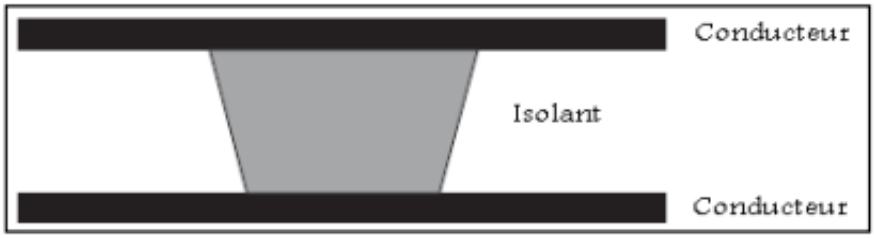


Fig.3.7 Anti-fusible

Les circuits à anti-fusibles partagent, avec ceux à SRAM, le sommet de la gamme des circuits programmables en vitesse et en densité d'intégration. Il est clair que ces circuits (à anti-fusibles) ne sont programmables qu'une fois.

2.3. Architecture interne des FPGAs

Les circuits FPGA possèdent une structure matricielle de deux types de blocs (ou cellules). Des blocs d'entrée-sortie et des blocs logiques programmables. Le passage d'un bloc logique à un autre se fait par un routage programmable. Certains circuits FPGAs intègrent également des mémoires RAM, des multiplieurs et même des noyaux de processeur [33]. Actuellement deux fabricants mondiaux se disputent le marché des FPGAs : Xilinx et Altera. Nous ferons une description de l'architecture utilisée par Xilinx, qui se présente sous forme de deux couches :

- Une couche appelée circuit configurable.
- Une couche appelée réseau mémoire *SRAM*.

La couche dite "circuit configurable" est constituée d'une matrice de blocs logiques configurables CLBs (Configurable Logic Blocs) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs entrées-sorties IOBs (Input Output Blocs) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs (Fig. 3.8).

La programmation du circuit FPGA consistera par le biais de l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ pour interconnecter les éléments des CLBs et des IOBs afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM.

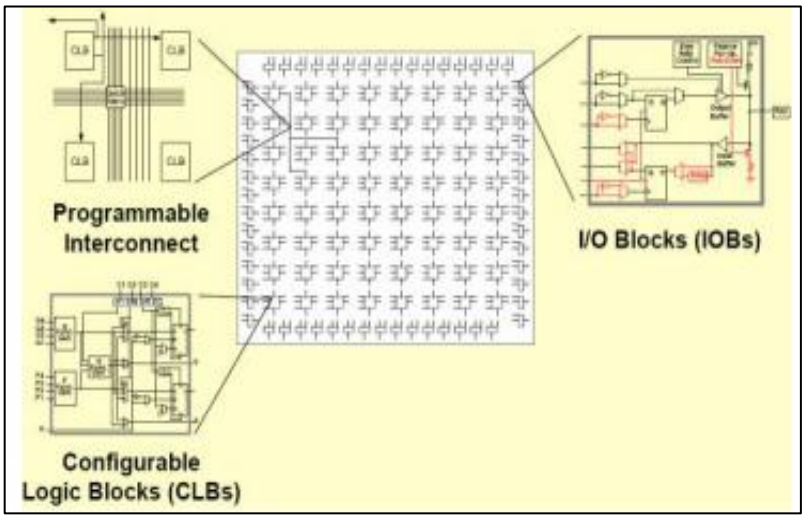


Fig. 3.8 : Architecture interne d'uu *FPGA*

a. Les CLBs (Configurable Logic Blocs)

Les blocs logiques configurables sont les éléments déterminants des performances du circuit FPGA. Chaque CLB est un bloc de logique combinatoire composé de générateurs de fonctions à quatre entrées et d'un bloc de mémorisation/synchronisation composé de bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB. La figure (3.9) présente la structure d'un bloc logique configurable CLB de la technologie Xilinx. Cette structure comporte une LUT (Look-up Table) de 4 bits qui permet de réaliser n'importe quelle fonction combinatoire de quatre variables logiques. Ce LUT peut être aussi configuré comme étant une mémoire RAM (16×1) ou un registre de décalage de taille 16 bits. Elle comporte aussi un multiplexeur utilisé pour l'implémentation de grand multiplexeurs et une bascule D avec toutes ses entrées de contrôle (horloge, reset, enable) [33].

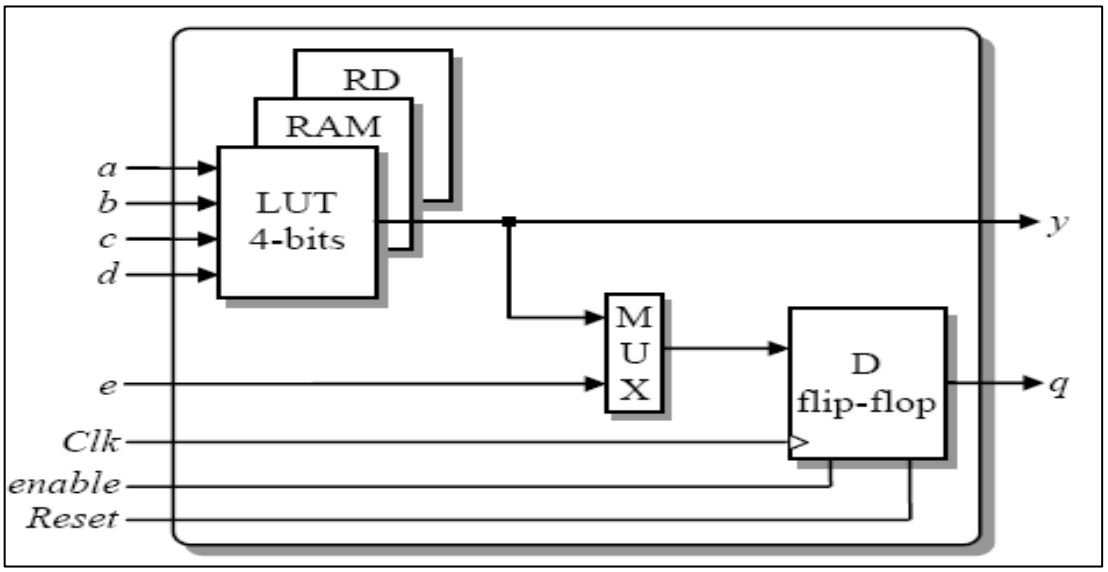


Fig. 3.9 Structure d'une cellule logique.

b. Les IOBs (Input Output Blocs)

Ces blocs d'entrée-sortie permettent l'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signal bidirectionnel ou être inutilisé (état haute impédance). La figure (3.10) présente la structure de ces blocs.

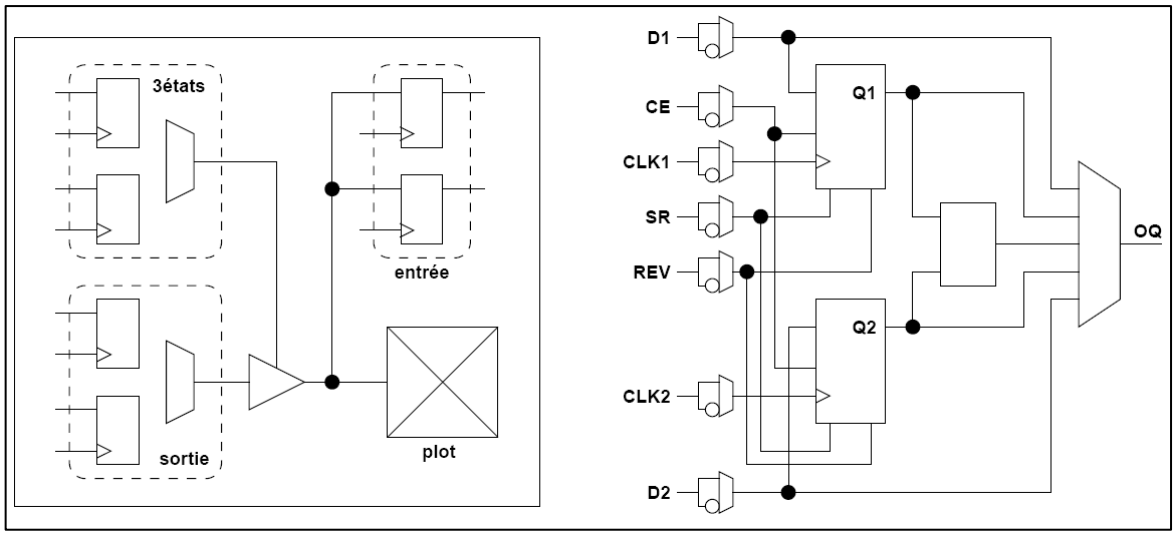


Fig. 3.10 Schéma d'un bloc d'entrée/sortie (IOB).

c. Réseau mémoire SRAM

La programmation d'un circuit FPGA est volatile, la configuration du circuit est donc mémorisée sur la couche réseau SRAM et stockée dans une ROM externe. Un dispositif interne permet à chaque mise sous tension de charger la SRAM interne (Fig. 3.11) à partir de la ROM. Ainsi on imagine aisément qu'un même circuit puisse être exploité successivement avec des ROM différentes puisque sa programmation interne n'est jamais définitive. On voit tout le parti que l'on peut tirer de cette souplesse en particulier lors d'une phase de mise au point. Une erreur n'est pas rédhibitoire, mais peut aisément être réparée.

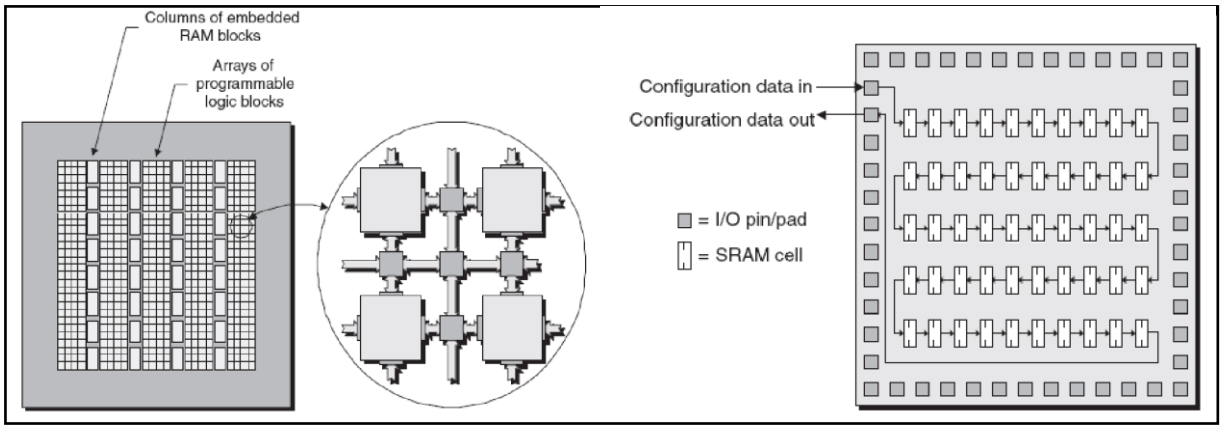


Fig. 3.11: Réseau mémoire *SRAM*.

d. Architecture des connexions

L'architecture d'un FPGA est la façon dont les commutateurs et les segments programmables de câblage sont placés pour permettre l'interconnexion des blocs logiques. Il y a habituellement un

compromis entre la flexibilité et la densité, puisque plus il y a d'interconnections possibles dans un FPGA, plus il est flexible. En contrepartie une plus grande surface est perdue pour les connexions et la configuration. Les architectures de connexion évoluées incluent une vue hiérarchique des interconnexions, avec des boites de commutateurs (switch boxes) où les files verticaux et horizontaux se croisent et peuvent être reliés ensemble. Les files sortantes d'un bloc logique peuvent entrer dans les boites commutateurs pour arriver aux connexions de plus haut niveau.

Cette architecture, connue sous le nom de modèle à iles, est représentée dans la figure (3.12), c'est la plus utilisée dans les dispositifs commerciaux.

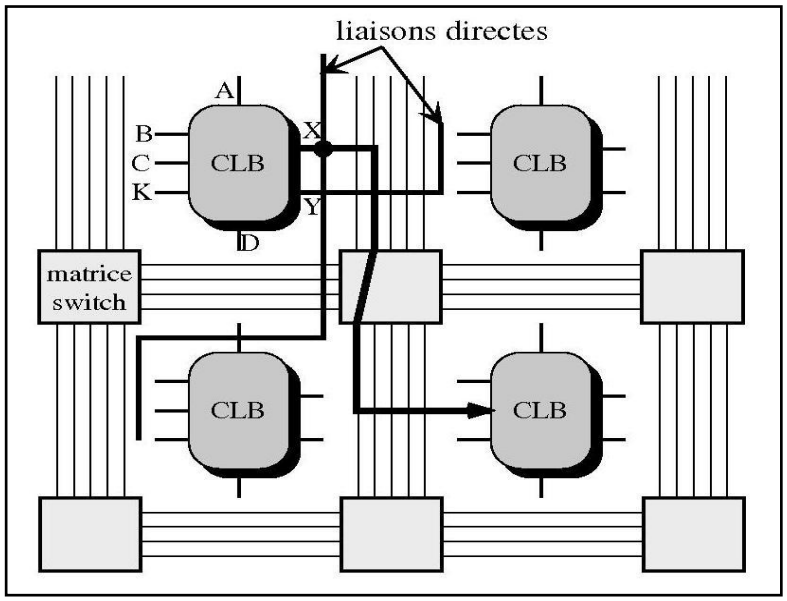


Fig. 3.12 Exemple d'architecture de connexions.

2.4. Configuration et programmation des circuits FPGAs

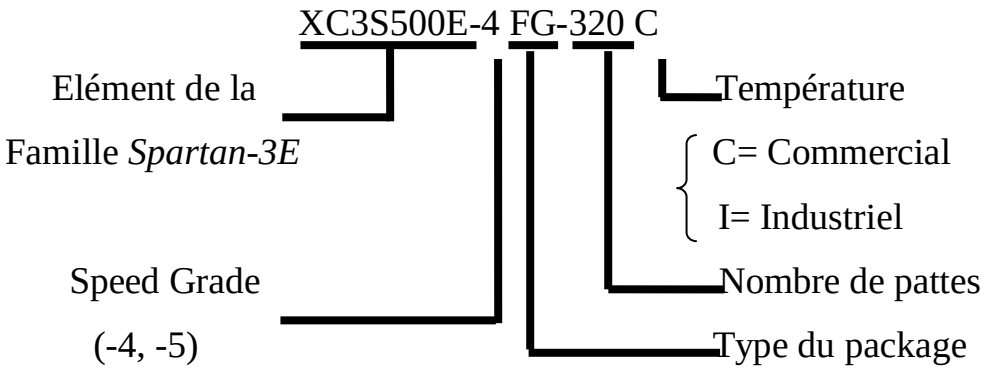
La configuration est le processus de charger des données spécifiques à une conception dans un ou plusieurs FPGAs pour définir l'opération fonctionnelle des blocs interne ainsi que leurs interconnexions. Le temps de chargement des données dépend du mode de configuration sélectionné. Dans tous les cas le chemin des données de la configuration est soit via un port série pour minimiser le besoin de pins soit via un bus à 8 bits pour maximiser les performances ou pour des interfaces plus simples avec le processeur[35].

Pour programmer un FPGA, il faut commencer par décrire la conception en utilisant un langage de description matériel (tel que VHDL, Verilog,..). Le synthétiseur va ensuite générer la liste d'interconnexion (netlist) qui permettra de simuler le placement de tous les composants au sein du FPGA et d'effectuer le routage entre les différentes cellules logiques.

La mise au point de cette configuration s'effectue en deux temps : Une première étape purement logicielle va consister à dessiner puis simuler logiquement le circuit fini. Dans la seconde étape, on effectuera une simulation matérielle en configurant un circuit réel. On pourra alors vérifier si le fonctionnement réel correspond bien à l'attente du concepteur, et si besoin, identifier les anomalies liées généralement à des temps de transit réels légèrement différents de ceux supposés lors de la simulation logicielle, ce qui peut conduire à des états instables voir même erronés.

2.5. Nomenclature du FPGA

Le client (le développeur) doit savoir comment choisir le FPGA le plus adéquat pour le développement de son application. Pour cela, les constructeurs suivent une nomenclature spécifiant la famille, le type du package et d'autre informations spécifique au FPGA vendu. Les circuits FPGA suivent la nomenclature suivante :



Speed grade : la vitesse du composant selon la technologie.
Type de package : c'est une collection de déclarations (types, signaux, constantes) et sous programmes utilisés fréquemment par plusieurs concepteurs.

3. La carte de développement *Spartan-3E*

Xilinx présente à ces clients deux grandes générations : Virtex (II, V, VII,…) et Spartan (2, 3,6). La génération Spartan-3 comporte deux familles : Spartan-3A et Spartan-3E [35]. La carte de développement Spartan-3E de Digilent met de l'avant le FPGA Spartan-3E (XC3S500E) de Xilinx et ces caractéristiques uniques. Cette carte offre un environnement de conception très adaptée pour le prototypage d'applications variées dont celles des systèmes numériques à usage général et des systèmes embarqués. Cette carte est de plus idéale pour les applications de traitement vidéo et de traitement de signal en général.

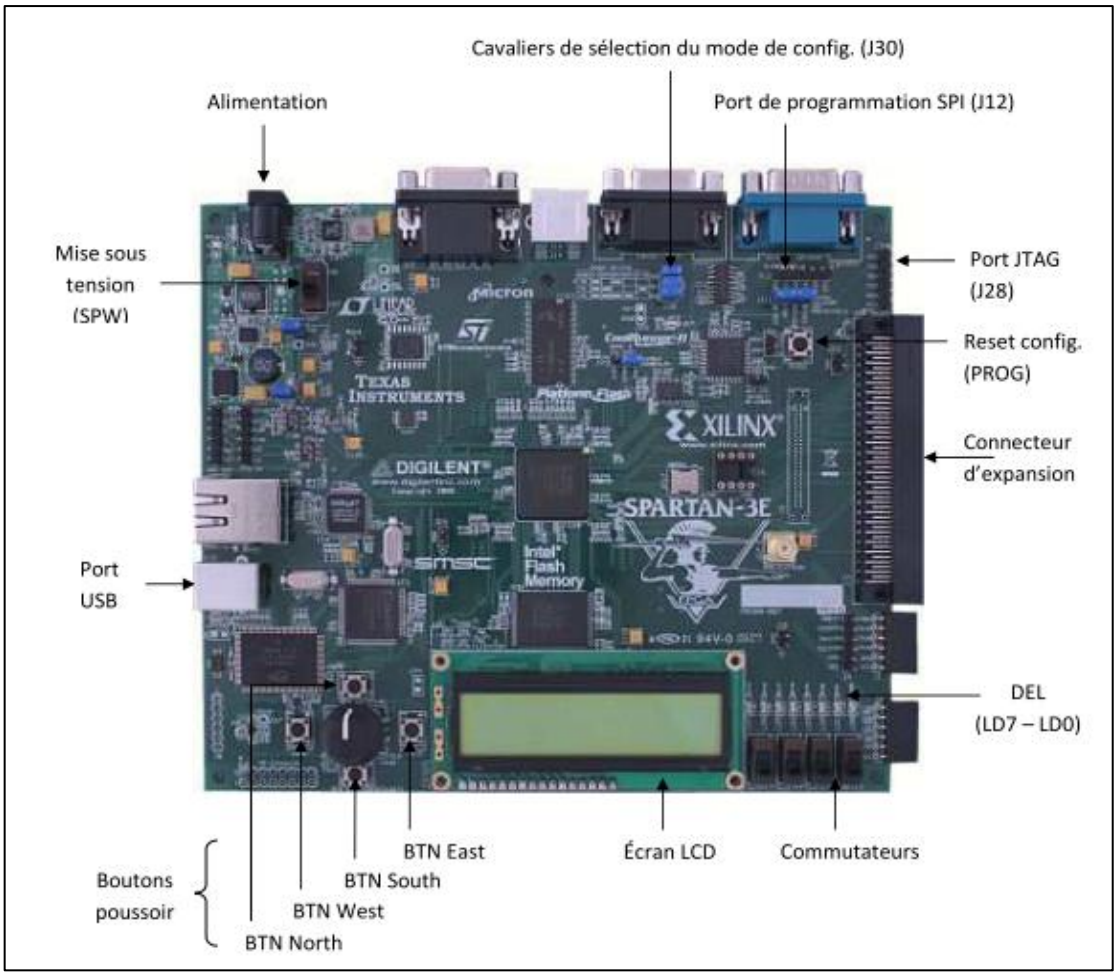


Fig. 3.13 : La carte *Spartan-3E* et ses principaux modules embarqués

La carte *Spartan-3E* (Fig3.6) regroupe entre autre

- Un *FPGA XC3S500E Spartan-3*^{E}
- Un *CPLD Coolrunner II*
- Une mémoire *PROM* de *4 Mbit*
- Une mémoire *Flash sérielle* de *16 Mbit*
- Une mémoire *Flash parallèle* de *128 Mbit*
- Une mémoire *SDRAM DDR* de *512 Mbit*
- Un écran *LCD*, un port *PS/2*, un port *VGA*
- Une prise *Ethernet 10/100*
- Deux ports *RS-232*
- Un port de configuration *USB*
- Deux convertisseurs de données
- Un oscillateur à *50MHz*

- Un connecteur d'expansion *Hirose FX2* permettant *40 entrées/sorties* génériques
- Des *DEL*, des boutons poussoirs et des commutateurs.

La carte supporte trois modes de configuration à la mise sous tension, soit le mode Master-Slave utilisant le PROM 4 Mbit, un mode SPI utilisant la mémoire Flash sériel et un mode BPI utilisant la mémoire Flash parallèle. La carte peut aussi être programmée directement à l'aide d'un port JTAG lorsqu'elle est sous tension [34].

4. Le VHDL

Le VHDL (Very high speed integrated circuit Hardware Description Language) est un langage de description matériel. Au début, Il a été sponsorisé par le département de la défense des Etats Unis, et adopté après par le IEEE (Institute of Electrical and Electronics Engineering) [35]. Le VHDL a été au cours des conceptions électronique dès son déclaration officielle par le IEEE en 1987. Pendant plus de 25 ans, l'industrie de la conception électronique automatisée s'est développé et l'utilisation du VHDL est allé du document parlant des concepts initiaux de la conception vers l'implémentation et la vérification fonctionnelle [36]. Le VHDL est conçu de manière à nous permettre de modéliser des systèmes complexes décrits à des niveaux d'abstractions très différents. De plus, le VHDL est un langage modulaire et hiérarchique. Un système complexe peut être divisé en plusieurs blocs, chaque bloc peut à son tour être divisé en plusieurs sous blocs et ainsi de suite. L'interface entre les différents blocs se fait par des "liens de communication". De la même manière, un modèle VHDL peut être composé de plusieurs sous-modules, chaque sous-module peut être divisé à son tour en d'autres sous-modules et ainsi de suite. Le VHDL a d'abord été proposé comme un langage de modélisation et il a graduellement été utilisé comme un langage de synthèse. Cependant, il faut noter que les possibilités de modélisation avec VHDL dépassent de loin les besoins de la synthèse. Certaines caractéristiques abstraites de VHDL n'ont pas d'équivalent dans la logique digitale. Autrement dit, la synthèse ne supporte qu'un sous ensemble des possibilités de VHDL. Parmi les avantages du langage VHDL nous insistons sur les aspects suivants :

- Cycle de conception plus court.
- Description plus compacte, moins d'erreurs.
- Exploration de l'espace de design, re-design.
- Maîtrise de la complexité.
- Description portable (indépendante de la technologie).

4.1. Description Générale

Un module VHDL élémentaire est appelé entité. Il est principalement composé de deux éléments de base : l'énoncé entity et l'énoncé architecture. L'énoncé entity décrit l'interface entre l'entité et le monde extérieur (i.e. les signaux d'entrée-sortie, les broches de la puce). L'énoncé architecture décrit quant à lui son contenu interne, son comportement. Notons ici qu'une entité VHDL peut avoir plusieurs architectures différentes. Ceci nous permet de générer des vues différentes de la même entité à des niveaux d'abstraction différents.

4.2. Quelques règles de base

Pour illustrer les règles de base du VHDL considérons l'exemple d'un comparateur à deux bits, décrit, ci-dessus, à l'aide de deux comparateurs à un bit.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL ;
3.  entity comp2 is
4.  Port ( a : in  STD_LOGIC_VECTOR (1 downto 0) ;
5.          b : in  STD_LOGIC_VECTOR (1 downto 0) ;
6.         aeqb : out  STD_LOGIC);
7.  end comp2 ;
8.  architecture  arch_struct of comp1 is
9.  signal  e0: std_logic  ;
10. signal  e1: std_logic  ;
11. begin
12. --instantiation de deux comparateur à 1 bit
13. unit0 : entity work.comp1(comp1_arch)
14. port map (i0=>a(0) , i1=>b(0) , eg=>e0);
15. unit1:entity work.comp1(comp1_arch)
16. port map (i0=>a(1) , i1=>b(1) , eg=>e1);
17. --a et b sont égaux si et seulement si ils sont égaux bit à bit
18. aeqb <= e0 and e1;
19. end arch_struct;
```

Exemple d'une description VHDL

Voyons maintenant quelques règles qu'un développeur doit suivre pour décrire son système d'une manière correcte :

a. La bibliothèque et le package

Les deux premières lignes :

1. **library** IEEE;
2. **use** IEEE.**STD_LOGIC_1164**.ALL ;

appelle le package *std_logic_1164* de la bibiothèque *IEEE*.

Pour des raisons de flexibilité les types logiques des ports sont souvent définis au préalable dans des bibliothèques. En VHDL, les bibliothèques sont spécifiées par le mot clé library. Pour avoir accès à une partie de la bibliothèque on utilise l'énoncé use. On a donc accès dans l'entité comp2 à tous les types définis dans le package std_logic_1164 de la libraire IEEE. Cela nous permet entre autres d'utiliser le type std_logic.

b. Déclaration de l'entité :

La déclaration de l'entité

3. **entity** comp2 **is**
4. **Port (** a : in STD_LOGIC_VECTOR (1 downto 0) ;
5. b : in STD_LOGIC_VECTOR (1 downto 0) ;
6. aeqb : out STD_LOGIC**);**
7. **end** comp2 ;

spécifie essentiellement les signaux d'entrées-sorties du circuit. La première ligne indique que le nom du circuit est comp2, et la section port spécifie les signaux d'entrées-sorties, leur direction (entrée, sortie, bidirectionnel). La syntaxe générale pour la déclaration d'une entité est illustrée par la figure suivante :

```
entity <nom de l'entité> is
port(
        <Nom du port1> : <Direction> <Type Logique>;
        <Nom du port1> : <Direction> <Type Logique>;
                                                ...;
                                                );
end <nom de l'entité>;
```

Syntaxe d'une déclaration d'entité

c. Types de données

Le VHDL est un langage fortement typique, seules les valeurs définies peuvent être utilisées pour un objet. Les données peuvent être de type *std_logic* ou l'une de ses variantes. Le package

std_logic_1164 contient neuf valeurs. Parmi ces valeur le '0' logique le '1' logique et le 'z' pour 'haute impédance'.

Un signal dans un circuit numérique contient, fréquemment, plusieurs bits. Le type de données std_logic_vector, définit comme un tableau d'éléments de std_logic est utilisé pour ce but. La déclaration d'un vecteur est comme suit : nom_port : direction std_logic_vector (MSB downto LSB).

d. L'énoncé architecture

La déclaration de l'architecture

```
8.  architecture  arch_struct of comp1 is
9.  signal  e0: std_logic  ;
10. signal  e1: std_logic  ;
11. begin
12. --instantiation de deux comparateur à 1 bit
13. unit0 : entity work.comp1(comp1_arch)
14. port map (i0=>a(0) , i1=>b(0) , eg=>e0);
15. unit1:entity work.comp1(comp1_arch)
16. port map (i0=>a(1) , i1=>b(1) , eg=>e1);
17. --a et b sont égaux si et seulement si ils sont égaux bit à bit
18. aeqb <= e0 and e1;
19. end arch_struct;
```

décrit le fonctionnement du circuit. Le *VHDL* permet d'introduire plusieurs énoncés d'architectures pour la même entité. L'énoncé est identifié par le nom *arch_struct* (architecture_structurelle). Elle peut contenir d'autres options supplémentaires comme les signaux internes, les constantes,…etc. Deux signaux internes ont été déclarés dans cet énoncé :

```
9.  signal  e0: std_logic  ;
10. signal  e1: std_logic  ;
```

Remarque :

Tout comme dans la plupart des langages de programmation, le VHDL permet d'introduire des commentaires. Les commentaires sont introduits en plaçant deux tirets consécutifs "--" et se terminent dans tous les cas à la fin de la ligne. Evidemment les commentaires sont ignorés par le compilateur VHDL.

La syntaxe générale pour la déclaration de l'énoncé architecture est illustré par la figure suivante :

```
architecture <nom de l'architecture> of <nom de  l'entité> is

--déclaration des signaux internes;

begin

<instructions>… ;

…

end <nom de l'architecture>
```

Syntaxe d'une déclaration de l'architecture

L'énoncé principal, inclus entre begin et end, contient des instructions concurrentes. A l'inverse d'un programme en C, où l'exécution des instructions est séquentielle, les séquences concurrentes s'exécutent d'une manière analogue au fonctionnement de différentes parties d'un circuit, en parallèle. Le signal à gauche d'une instruction peut être considéré comme la sortie de la partie, et l'expression à droite spécifie le fonctionnement du circuit et les signaux d'entrée (ligne.18) [33].

Remarque :

La notion d'architecture structurelle consiste au fait que nous avons fait appel à deux comparateurs à un bit pour décrire l'architecture du comparateur à deux bits. Le comparateur à un bit doit être déjà conçu pour pouvoir l'appeler en utilisant l'instruction : port map. L'appel d'un circuit conçu auparavant se fait comme suit :

```
unité :  e n t i t y  nom_bibiothèque.nom_entité
(nom_architecture)

p o r t  map(

signal_formel => signal_actuel,

…

) ;
```

Syntaxe d'un appel d'un circuit développé auparavant

Où :

unité : le nouveau identifiant donné au composant pour être utilisé plus qu'une fois selon le besoin.

nom_bibliothèque : la bibliothèque de travail. La bibliothèque par défaut est **WORK.**

e. L'énoncé process

Pour faciliter la modélisation des systèmes, le VHDL contient plusieurs instructions séquentielles qui s'exécutent en séquence. Comme leur comportement est différent de celui des instructions concurrentes, les instructions séquentielles sont encapsulées à l'intérieur d'un process. Le process lui-même est une instruction concurrente. Il peut être considéré comme étant une boite noire ayant un comportement décrit par des instructions séquentielles. Un process mal codé mène fréquemment inutilement à une implémentation plus complexe ou ne peut pas être synthétisé. Les instructions les plus utilisées dans un process sont : if et case [33].

5. Le flot de développement

Le flot simplifié de développement d'un système sur cible FPGA est montré par la figure (3.14). Pour plus de simplicité de lecture, nous poursuivons les termes utilisés dans la documentation de Xilinx. La partie gauche du flot est le processus de programmation et de la finition, dans lequel un système est transformé d'une description en texte HDL vers la configuration d'un dispositif de niveau cellulaire qui sera téléchargée sur la cible FPGA. La partie droite est le processus de validation, qui vérifie si le système a atteint les performances exigées ou non.

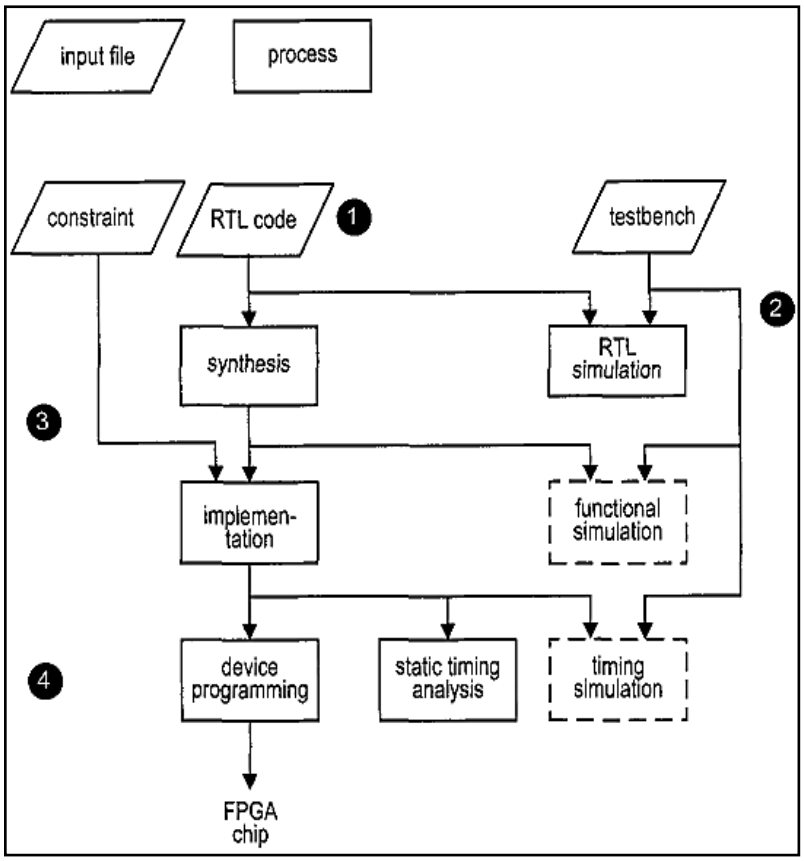


Fig. 3.14 : Le flot de développement sur une cible FPGA

Les principales étapes sont:

- Concevoir le système et dériver le fichier HDL : Nous aurons besoins d'écrire un fichier séparé de contraintes pour spécifier certaines contraintes d'implémentation.

- Ecrire un testbench en HDL et performer une simulation RTL : Le terme RTL tient au fait que le code HDL est fait dans un niveau comportemental.

- Réaliser la synthèse et l'implémentation : Le processus de synthèse est souvent dit : synthèse logique, dans lequel le logiciel transforme le code HDL à des composants génériques, comme les portes logiques simples et les bascules. Le processus d'implémentation passe par trois petits processus: traduction, mapping, placement et routage. Le processus de traduction fusionne plusieurs fichiers en un netlist. Le processus de mapping qui est, généralement, connu sous le nom de : technolgy mapping (mapping technologique), fait la liaison entre les portes génériques, dans le netlist crée, les cellules logiques du FPGA et le IOBs. Le processus de placement et de routage réalise la conception physique à l'intérieur de la puce FPGA. Il place les cellules dans leurs locations physiques et définit le chemin que doit prendre chaque signal. Dans le flot de Xilinx, static timing analysis (analyse statique du temps), qui détermine les différents paramètres temps, comme le temps de retard maximal et la fréquence maximale de l'horloge, est réalisé à la fin du processus d'implémentation.

- Générer et télécharger le fichier de programmation (généralement, le bitstream) : Dans ce processus un fichier de configuration est généré en accord avec le netlist final. Le fichier est téléchargé vers une cible FPGA. Le circuit physique peut alors être vérifié.

La simulation fonctionnelle peut être faite après la synthèse, et la simulation temporelle peut être réalisée après l'implémentation. La simulation fonctionnelle utilise le netlist synthétisé pour remplacer la description comportementale et vérifier l'exactitude de la synthèse. La simulation temporelle utilise le netlist final, avec des données détaillées sur le temps, pour faire la simulation. A cause de la complexité du netlist final, les deux simulations peuvent prendre un temps considérable. Si la description de la conception est écrite d'une manière convenable et efficace, le code HDL sera synthétisé et implémenté correctement. Dans ce cas nous n'aurons besoins qu'à utiliser la simulation comportementale pour vérifier l'exactitude du code HDL et à utiliser l'analyse statique du temps pour examiner les informations sur le temps. Les deux autres simulations peuvent alors être enlevées du flot de développement [33].

6. L'outil de développement ISE 12.3

L'outil de développement *Xilinx ISE* permet de réaliser les étapes essentielles suivante Créer un projet et le code *HDL (VHDL* ou *Verilog)*.

- Créer un *testbench* (banc d'essai) et réaliser une simulation *RTL*.
- Ajouter un fichier de contraintes.
- synthétiser et implémenter le code.
- Générer et télécharger le ficher de configuration vers la cible *FPGA*.

Remarque :

La première étape n'est nécessaire que pour simuler le fonctionnement du processus développé. Elle peut être surpassée [33].

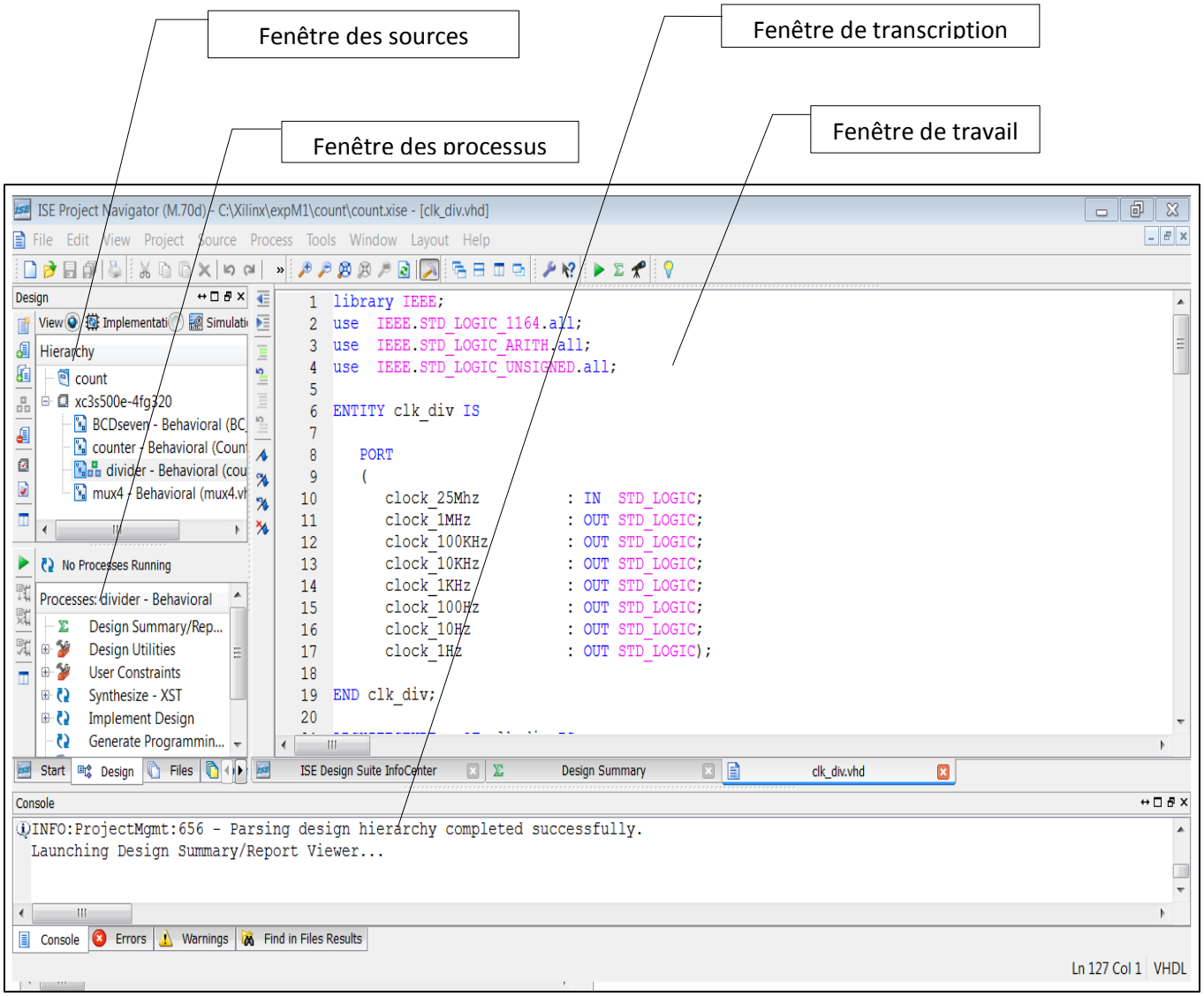


Fig 3.15 : L'environnement de l'outil de développement *Xilinx ISE 12.3*

La fenêtre par défaut de l'ISE est montrée par la figure (3.15). Elle est divisée en quatre sous fenêtres :

- Fenêtre des sources : affiche hiérarchiquement les fichiers inclus dans un projet.

- Fenêtre de processus : affiche les processus qui sont valables pour la source sélectionnée.
- Fenêtre de transcription : affiche les messages de l'état, des erreurs et des avertissemenst.
- Fenêtre de travail : contient des fenêtres pour plusieurs documents (code *HDL* Schématique,…etc.) qui peuvent être vus ou modifiés.

Chaque fenêtre peut être réduite, agrandie ou fermée. La disposition par défaut peut être restaurée en sélectionnant : layout → load default layout. La référence [37] donne en détaille les différentes manipulations possibles pour développer une application en VHDL ou en schématique.

7. System Generator

System Generator est un outil de traitement numérique des signaux de la firme Xilinx qui s'intègre à l'environnement Simulink pour la modélisation des systèmes en schéma bloc. L'expérience précédente avec les méthodes de conception de type RTL (Register Transfer Level) des FPGAs Xilinx ne sont pas demandés pour pouvoir utiliser System Generator.

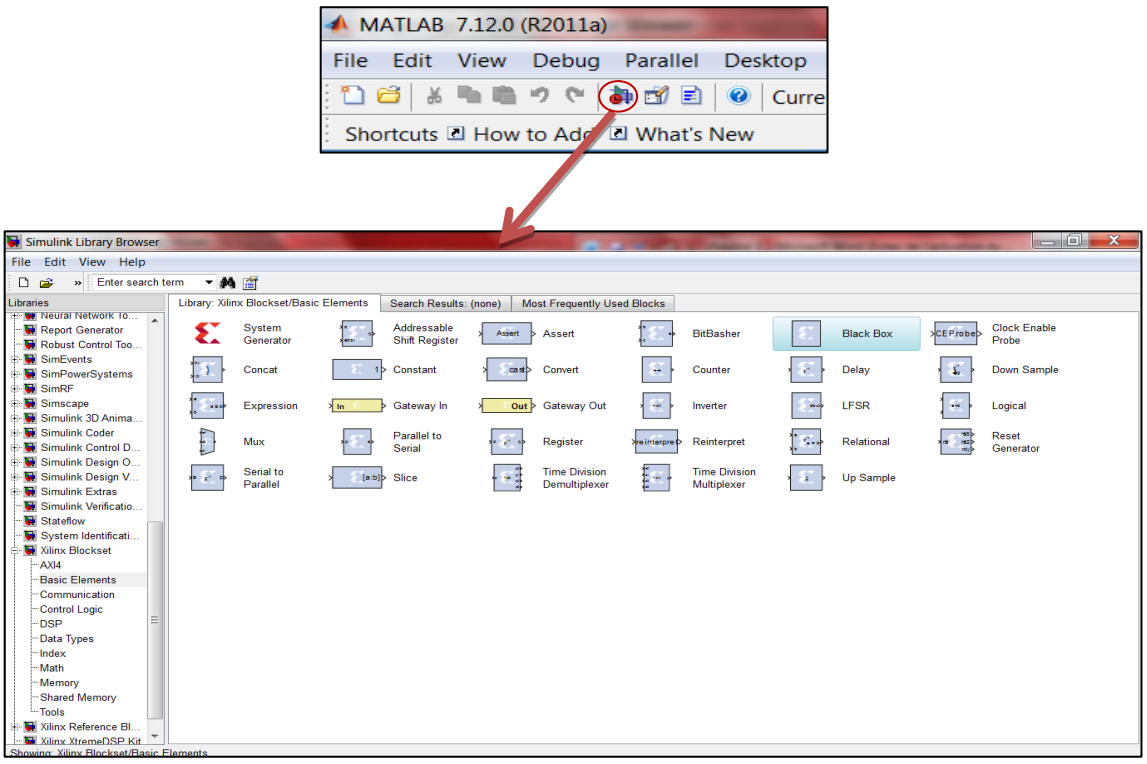


Fig. 3.16 : *Xilinx Blockset* dans l'environnement *Simulink*

La conception peut être faite dans l'environnement Simulink ce qui permet une manipulation facile. La synthèse, le placement et le routage sont automatiquement performés pour générer le fichier de programmation [38].

Plus de 90 blocs de construction DSP (Digital Signal Processor) sont fournis dans le jeu de blocs DSP de Xilinx pour Simulink (Fig3.16). Ces blocs comprennent les blocs de construction commune DSP tels que des additionneurs, des multiplicateurs et des registres. D'autres blocs sont également inclus tels que les blocs FFT (Fast Fourier Transformer) et les filtres.

7.1. Définition de quelques blocs

a. Le bloc System Generator

Le bloc System Generator permet de contrôler les paramètres du système et de la simulation. Tous les modèles Simulink contenant les blocs de Xilinx doivent contenir au moins un bloc System Generator. Une fois ce dernier est ajouté à un modèle, il est possible de spécifier la manière dont la génération de code et la simulation doivent être effectués (Fig3.17).

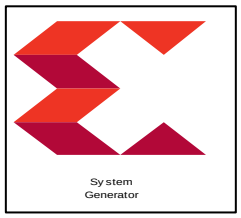


Fig.3.17: Le bloc *System Generator*.

En double cliquant sur le block System Generator, on peut choisir la cible FPGA, le mode de compilation et paramétrer la simulation.

Remarque:

En contrepartie des blocs de Simulink les blocs de System Generator nécessitent une configuration assez délicate. Il faut marquer ici que le type de données en entrée et en sortie doit être bien spécifié. C'est la transition du System Generator vers le Simulink ou l'inverse qui cause plus de difficultés lors de la modélisation des systèmes dans Simulink. Pour permettre cette transition deux autres blocs sont obligatoirement utilisés : le Gateway in et le Gateway out.

b. Black Box

Le Black Box de Xilinx, est répertorié dans les bibliothèques suivantes : Xilinx Blockset, Basic elements. Il nous permet d'introduire des fonctions qui n'existent pas dans la bibliothèque du System Generator par des scripts VHDL. Nous l'avons utilisé pour simuler le fonctionnement des circuits décrits en VHDL au cours de notre travail.

8. Conclusion

A travers ce chapitre, nous avons, en premier lieu, décrit les caractéristiques essentielles des FPGAs en abordant leur architecture interne et leur configuration. En second lieu nous nous sommes intéressés au langage de description matérielle VHDL et aux outils qui permettent de décrire et synthétiser une conception suivant le flot de développement donné dans ce chapitre.

Les circuits FPGA présentent de nombreux avantages :

Le premier avantage est la souplesse de programmation qui permet l'emploi conjoint des solutions logicielles et matérielles, ce qui permet de multiplier les essais, d'optimiser de diverses manières l'architecture développée, de vérifier à divers niveaux de simulation la fonctionnalité de cette architecture.

Le second avantage est évidemment la nouvelle possibilité de reconfiguration dynamique d'un circuit ce qui permet d'une part, une évolutivité assurant la possibilité de couvrir des nouveaux besoins sans nécessairement modifier l'architecture dans sa totalité.

Pour les livres : noms des auteurs, « titre du livre », nom de l'éditeur ville et pays, année d'édition

Références

[1] Pong P.Chu, FPGA Prototyping by VHDL Examples, Edition Wiley USA, 2008.

[2] UG230: Spartan-3E Starter Kit User Guide, Xilinx, 2011.

http://www.xilinx.com/support/documentation/userguides/ug230.pdf

[3]UG332: Spartan-3 Generation Configuration User Guide,Xilinx, 2009.

http://www.xilinx.com/support/documentation/userguides/ug332.pdf

[4] Doglas L.Perry ,VHDL Programming by Example, Mc Graw Hill 4th edition,USA,2002.

[5] UG695 (v 12.3) : ISE In-Depth Tutorial, Xilinx, September 21,2010.

www.xilinx.com/products/boards/s3astarter/reference_designs.htm

[6] System Generator for DSP, release 10.1, Xilinx, 2008.

Chapitre 3
Les Circuits Programmables FPGAs

1. Introduction

Le développement d'un système complexe sur *FPGA* est un processus compliqué qui consiste à faire plusieurs transformations complexes et à optimiser les algorithmes qui vont être implémenté. Afin d'automatiser quelques tâches, des outils logiciels ont été introduite sur le marché par les différents constructeurs des *FPGAs* [29]. Nous utilisons la version *Xilinx ISE 12.3* pour la synthèse, la simulation et l'implémentation, et nous utilisons le *System Generator* pour la simulation. Dans ce chapitre, nous donnons une brève description des circuits *FPGAs* et de la carte de développement *SPARTAN-3E,* nous introduisons les outils de développement *ISE* et *System Generator* et nous présentons une vue globale du langage de description *VHDL* et ses règles de base.

2. Les FPGAs

2.1. Les circuits logiques programmables

Un circuit logique programmable (*PLD : Programmable Logic Device*) est un assemblage d'opérateurs logiques combinatoires et de bascules dans lequel la fonction réalisée n'est pas fixée lors de la fabrication. Il contient potentiellement la possibilité de réaliser toute une classe de fonctions, plus ou moins large suivant son architecture.

L'apparition de ce type de circuit s'est d'abord faite avec les circuits logiques programmables simples de type *PAL* (*Programmable Array Logic*), qui se programment comme des mémoires non volatiles de type *ROM* et sont utilisés pour implémenter des fonctions combinatoires simples (décodeurs d'adresse, contrôleurs de bus,…). Avec les développements de la micro-électronique il y a eu apparition des différents familles de circuits programmables : les *CPLD* (*Complex Logic Programmable Device*),

puis les *FPGA* (Field Programmable Gate Arrays), introduits par la société *Xilinx* en 1985. Les circuits de type *FPGA* les plus récents offrent désormais l'équivalent des millions de portes logiques programmables.

Il existe plusieurs familles de *PLDs* qui sont différenciées par leur structure interne, la plus ancienne et la plus connue est la famille des *PAL*.

a. Les P.A.L

Les *PAL*s (*Programmable Array Logic*) ou réseau logique Programmable sont les premiers circuits programmables à être utilisés pour réaliser des fonctions logiques. Ils possèdent des matrices « *ET* » programmables et des matrices « *OU* » fixes. La programmation de ces circuits s'effectue par destruction de fusibles. Une fois programmés, on ne peut plus les effacer. La structure de base de ce *PLD* est présentée par le schéma suivant:

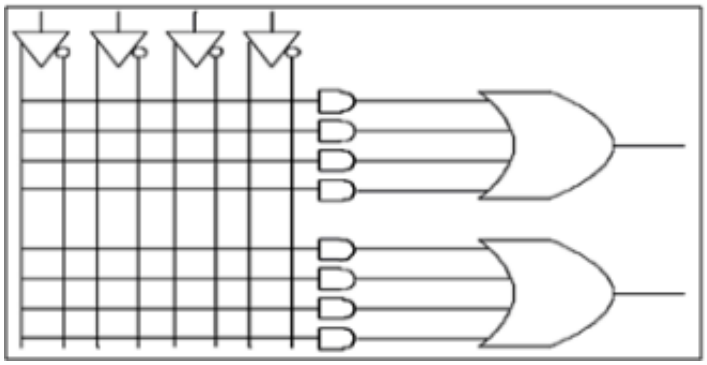


Fig. 3.1 Structure de base d'un PAL

b. Les P.A.L. effaçables (E.P.L.D.)

Les *EPLDs* (*Erasable Programmable Logic Device),* ou circuit logique programmable et effaçable peuvent être effacés par ultraviolet ou électriquement. Ils sont encore appelés *PALCMOS*.

c. Les G.A.L.

Les GAL (Generic Array Logic) ou encore réseau logique générique. Leur fonctionnement est identique aux *PALCMOS,* ils sont programmables et effaçables électriquement.

d. Les C.P.L.D.

Les *CPLDs* (*Complex Programmable Logic Device*) sont composés de plusieurs *PALs* élémentaires reliés entre eux par une zone d'interconnexion. Grâce à cette architecture, ils permettent d'atteindre des vitesses de fonctionnement élevées (plusieurs centaines de *MHz*). La figure suivante montre la structure générale d'un *CPLD*.

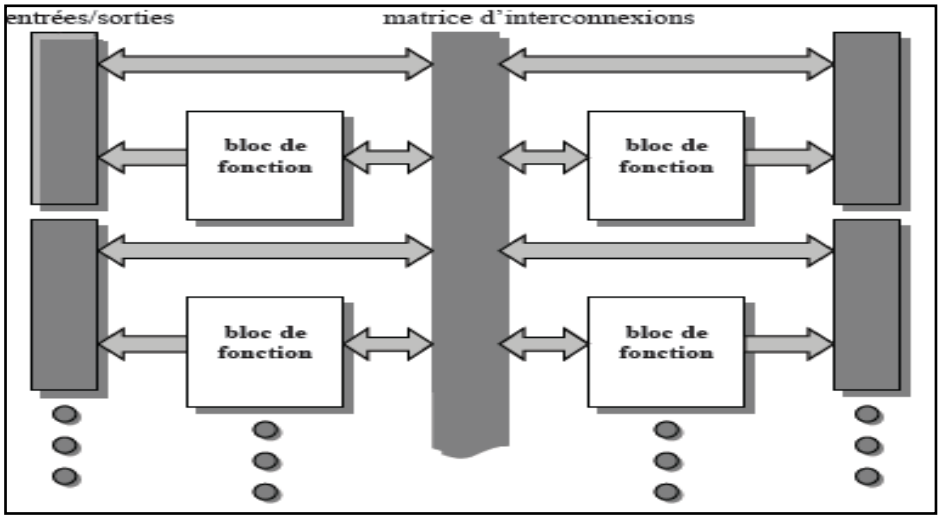


Fig. 3.2 : Structure générale d'un CPLD

e. Les F.P.G.A.

Un *FPGAs* (*Field Programmable Gate Array)* ou réseau de portes programmable par l'utilisateur. C'est un ensemble de blocs logiques élémentaires que l'utilisateur peut interconnecter pour réaliser les fonctions logiques de son choix. On distingue deux types :

- *FPGA* à anti-fusible : programmables électriquement par l'utilisateur, non effaçables

-*FPGA* à *SRAM* : ou encore *LCA* (Logic Cell Array) à base des cellules *SRAM* pour configurer les connexions entre les blocs logiques.

La figure suivante illustre un résumé graphique des familles des circuits logiques programmables.

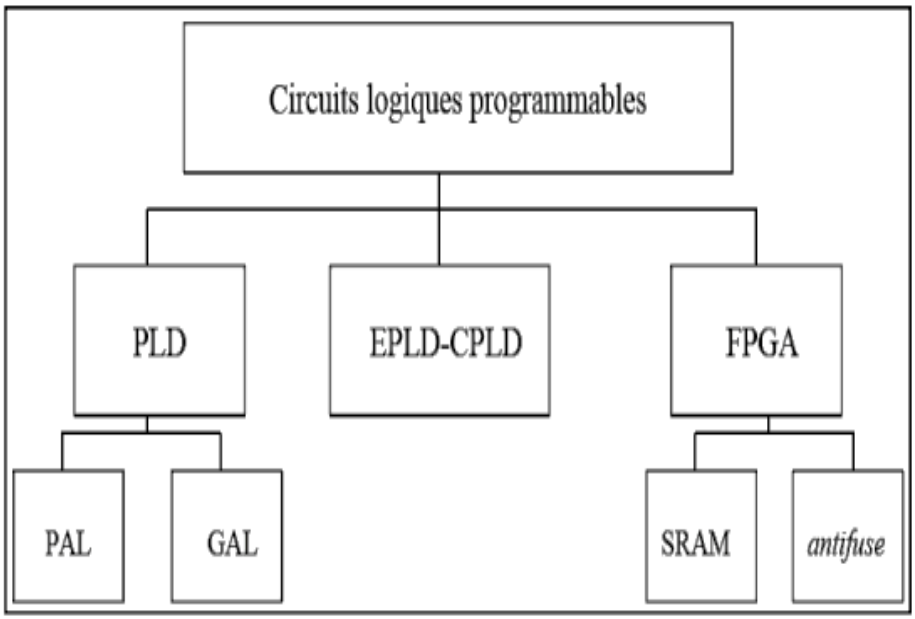


Fig. 3.3 : Les circuits logiques programmables

2.2. Les technologies d'interconnexion

Le premier critère de choix d'un circuit programmable est la technologie utilisée pour matérialiser les interconnexions:

a. Fusibles

La connexion par fusibles, la première méthode employée, est en voie de disparition. On ne la rencontre plus que dans quelques circuits de faible densité et de conception ancienne.

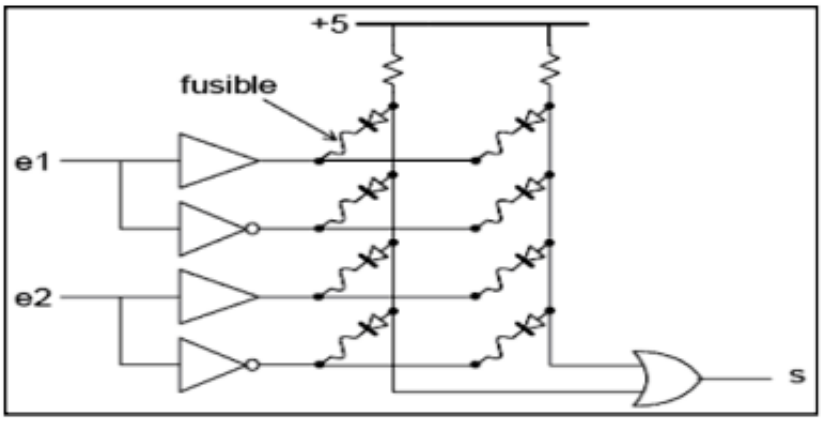


Fig. 3.4 PLD élémentaire à fusibles

La figure (3.4) en illustre le principe ; toutes les connexions sont établies à la fabrication. Lors de la programmation le circuit est placé dans un mode particulier par le programmateur, mode dans lequel des impulsions de courant sont aiguillées successivement vers les fusibles à détruire. Pour programmer un circuit, il faut transférer dans le programmateur une table qui indique par un chiffre binaire l'état de chaque fusible.

b. MOS à grille flottante

Les transistors *MOS* sont des interrupteurs, commandés par une charge électrique stockée sur leur électrode de grille. Si, en fonctionnement normal, cette grille est isolée, elle conserve sa charge éventuelle éternellement. Il reste au fondeur à trouver un moyen de modifier cette charge, pour programmer l'état du transistor. Le dépôt d'une charge électrique sur la grille isolée d'un transistor fait appel à un phénomène connu sous le nom d'effet tunnel : un isolant très mince soumis à une différence de potentiel suffisamment grande est parcouru par un courant de faible valeur, qui permet de déposer une charge électrique sur une électrode normalement isolée. Ce phénomène, réversible, permet de programmer et d'effacer une mémoire. La figure (3.5) montre la structure du *PLD* élémentaire précédent, dans lequel les fusibles sont remplacés par des transistors à grille isolée (technologie *FLASH*).

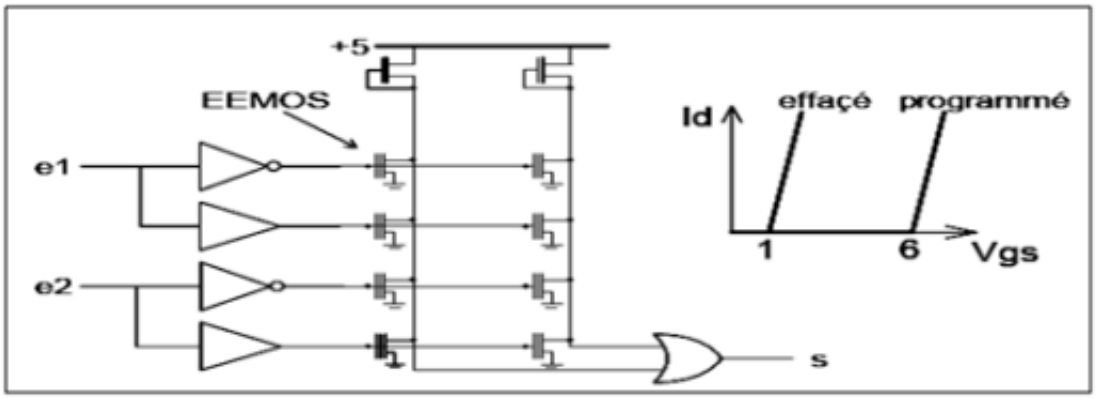


Fig. 3.5 : PLD simple à MOS

c. Mémoires statiques (SRAM)

Dans les circuits précédents, la programmation de l'état des interrupteurs, conservée en l'absence de tension d'alimentation, fait appel à un mode de fonctionnement électrique particulier. Dans les technologies à mémoire statique, l'état de chaque interrupteur est commandé par une cellule mémoire classique à quatre transistors (plus un transistor de programmation), dont le schéma de principe est celui de la figure suivante :

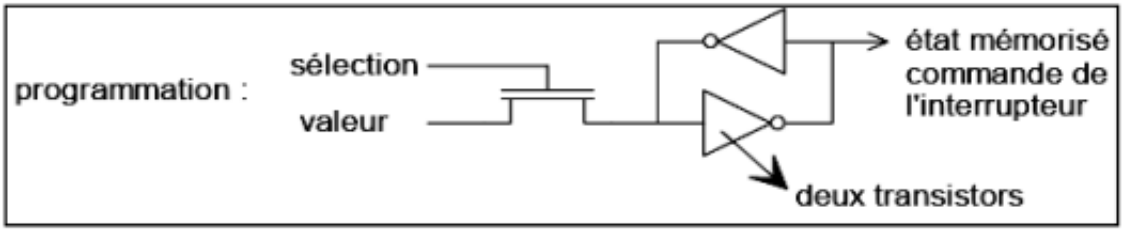


Fig.3.6 Cellule SRAM

d. Anti-fusibles

L'inverse d'un fusible est un anti-fusible. Le principe est, à l'échelle microscopique, celui de la soudure électrique par points. Un point d'interconnexion est réalisé au croisement de deux pistes conductrices (métal ou semi-conducteur selon les procédés de fabrication), séparées par un isolant de faible épaisseur. Une surtension appliquée entre les deux pistes provoque un perçage définitif du diélectrique.

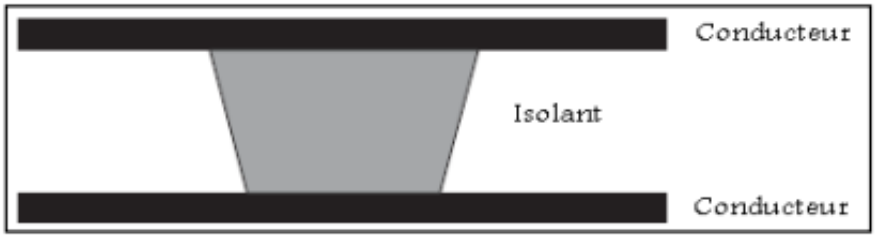


Fig.3.7 Anti-fusible

Les circuits à anti-fusibles partagent, avec ceux à SRAM, le sommet de la gamme des circuits programmables en vitesse et en densité d'intégration. Il est clair que ces circuits (à anti-fusibles) ne sont programmables qu'une fois.

2.3. Architecture interne des FPGAs

Les circuits *FPGA* possèdent une structure matricielle de deux types de blocs (ou cellules). Des blocs d'entrée-sortie et des blocs logiques programmables. Le passage d'un bloc logique à un autre se fait par un routage programmable. Certains circuits *FPGAs* intègrent également des mémoires *RAM*, des multiplieurs et même des noyaux de processeur [29]. Actuellement deux fabricants mondiaux se disputent le marché des *FPGAs* : *Xilinx* et *Altera*. Nous ferons une description de l'architecture utilisée par *Xilinx*, qui se présente sous forme de deux couches :

- Une couche appelée circuit configurable.
- Une couche appelée réseau mémoire *SRAM*.

La couche dite "*circuit configurable*" est constituée d'une matrice de blocs logiques configurables *CLBs* (Configurable Logic Blocs) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs entrées-sorties *IOBs* (Input Output Blocs) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs (Fig. 3.8).

La programmation du circuit *FPGA* consistera par le biais de l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ pour interconnecter les éléments des *CLBs* et des *IOBs* afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire *SRAM*.

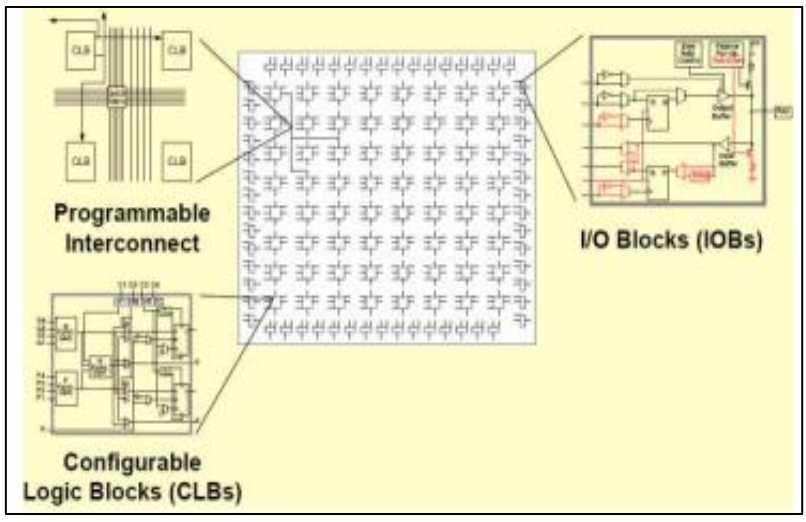


Fig. 3.8 : Architecture interne du *FPGA*

a. Les CLBs (Configurable Logic Blocs)

Les blocs logiques configurables sont les éléments déterminants des performances du circuit *FPGA*. Chaque *CLB* est un bloc de logique combinatoire composé de générateurs de fonctions à quatre entrées et d'un bloc de mémorisation/synchronisation composé de bascules *D*. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du *CLB*. La figure (3.9) présente la structure d'un bloc logique configurable *CLB* de la technologie *Xilinx*. Cette structure comporte une *LUT* (*Look-up Table*) de 4 bits qui permet de réaliser n'importe quelle fonction combinatoire de quatre variables logiques. Ce *LUT* peut être aussi configuré comme étant une mémoire *RAM (16×1)* ou un registre de décalage de taille 16 bits. Elle comporte aussi un multiplexeur utilisé pour l'implémentation de grand multiplexeurs et une bascule *D* avec toutes ses entrées de contrôle (horloge, reset, enable) [29].

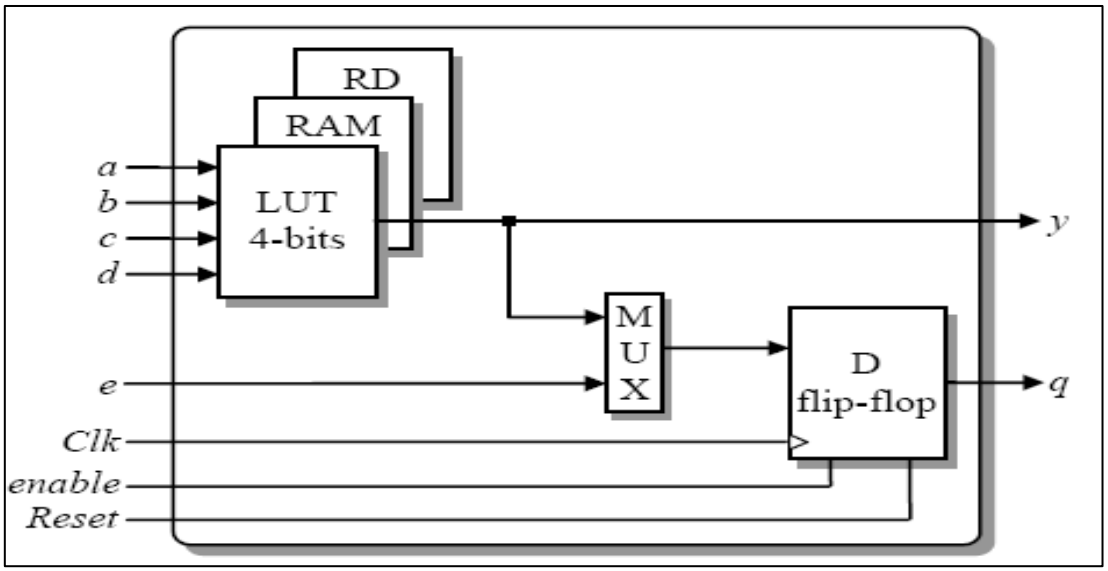


Fig. 3.9 Structure d'une cellule logique.

b. Les IOBs (Input Output Blocs)

Ces blocs d'entrée-sortie permettent l'interface entre les broches du composant *FPGA* et la logique interne développée à l'intérieur du composant. Ils sont présents sur toute la périphérie du circuit *FPGA*. Chaque bloc *IOB* contrôle une broche du composant et il peut être défini en entrée, en sortie, en signal bidirectionnel ou être inutilisé (état haute impédance). La figure (3.10) présente la structure de ces blocs.

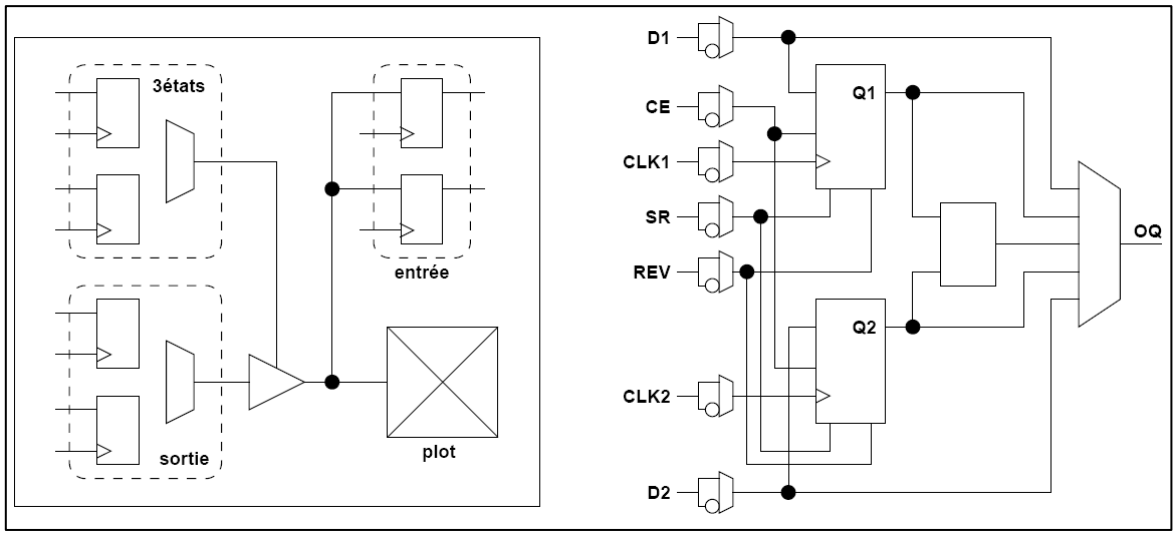


Fig. 3.10 Schéma d'un bloc d'entrée-sortie (IOB).

c. Réseau mémoire SRAM

La programmation d'un circuit *FPGA* est volatile, la configuration du circuit est donc mémorisée sur la couche réseau *SRAM* et stockée dans une *ROM* externe. Un dispositif interne permet à chaque mise sous tension de charger la *SRAM* interne (Fig. 3.11) à partir de la *ROM*. Ainsi on imagine aisément qu'un même circuit puisse être exploité successivement avec des *ROM* différentes puisque sa programmation interne n'est jamais définitive. On voit tout le parti que l'on peut tirer de cette souplesse en particulier lors d'une phase de mise au point. Une erreur n'est pas rédhibitoire, mais peut aisément être réparée.

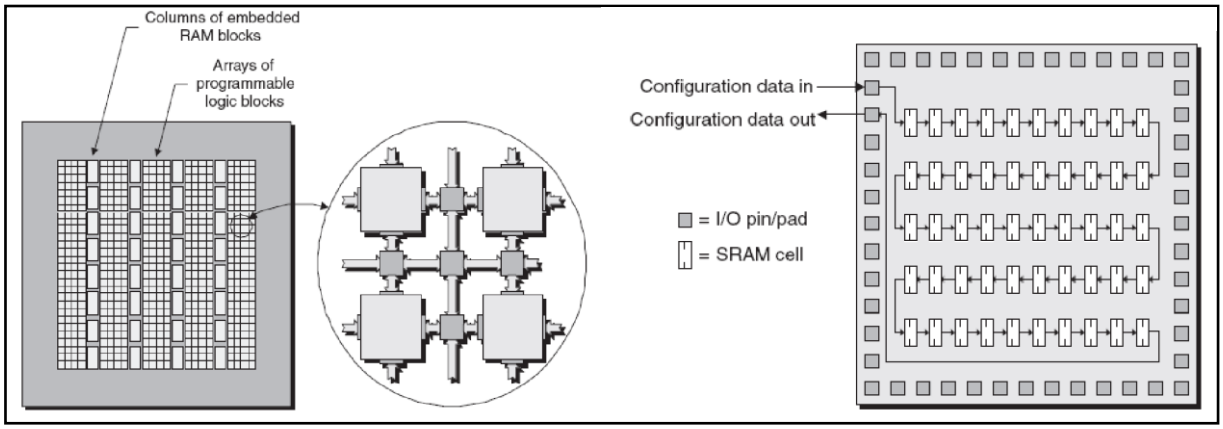


Fig. 3.11: Réseau mémoire *SRAM*.

d. Architecture des connexions

L'architecture d'un *FPGA* est la façon dont les commutateurs et les segments programmables de câblage sont placés pour permettre l'interconnexion des blocs logiques. Il y a habituellement un compromis entre la flexibilité et la densité, puisque plus il y a d'interconnections possibles dans un

FPGA, plus il est flexible. En contrepartie une plus grande surface est perdue pour les connexions et la configuration. Les architectures de connexion évoluées incluent une vue hiérarchique des interconnexions, avec des boites de commutateurs (*switch boxes*) où les files verticaux et horizontaux se croisent et peuvent être reliés ensemble. Les files sortantes d'un bloc logique peuvent entrer dans les boites commutateurs pour arriver aux connexions de plus haut niveau.

Cette architecture, connue sous le nom de modèle à iles, est représentée dans la figure (3.12), c'est la plus utilisée dans les dispositifs commerciaux.

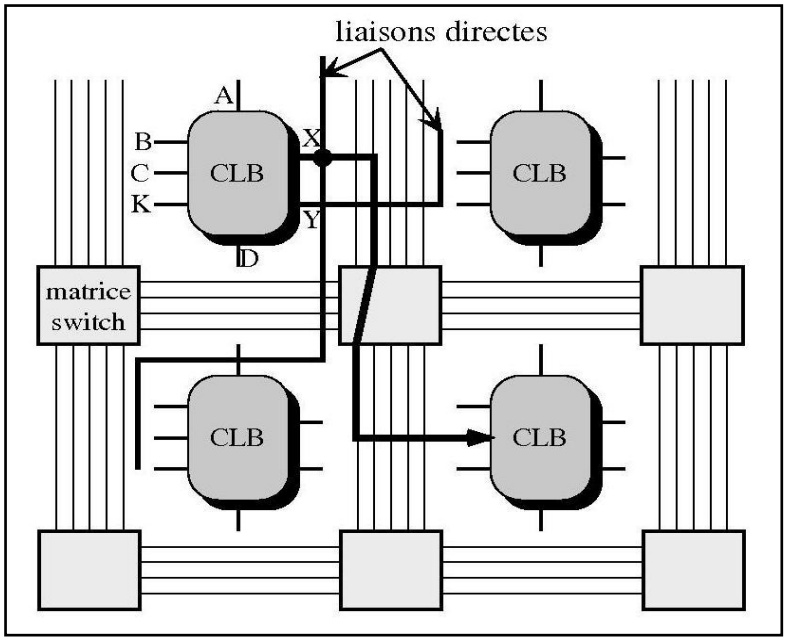


Fig. 3.12 exemple d'architecture de connexions.

2.4. Configuration et programmation des circuits FPGAs

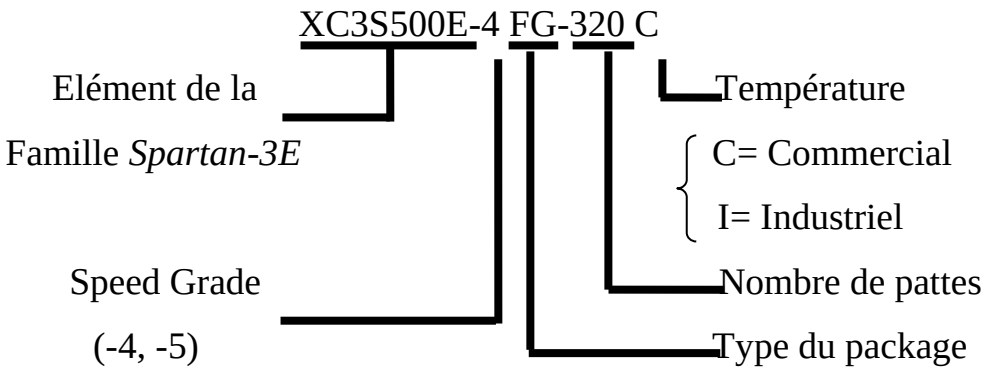
La configuration est le processus de charger des données spécifiques à une conception dans un ou plusieurs *FPGAs* pour définir l'opération fonctionnelle des blocs interne ainsi que leurs interconnexions. Le temps de chargement des données dépend du mode de configuration sélectionné. Dans tous les cas le chemin des données de la configuration est soit via un port série pour minimiser le besoin de pins soit via un bus à 8 bits pour maximiser les performances ou pour des interfaces plus simples avec le processeur [31].

Pour programmer un *FPGA*, il faut commencer par décrire la conception en utilisant un langage de description matériel (tel que *VHDL*, *Verilog*,..). Le synthétiseur va ensuite générer la liste d'interconnexion (*netlist*) qui permettra de simuler le placement de tous les composants au sein du *FPGA* et d'effectuer le routage entre les différentes cellules logiques.

La mise au point de cette configuration s'effectue en deux temps : Une première étape purement logicielle va consister à dessiner puis simuler logiquement le circuit fini. Dans la seconde étape, on effectuera une simulation matérielle en configurant un circuit réel. On pourra alors vérifier si le fonctionnement réel correspond bien à l'attente du concepteur, et si besoin, identifier les anomalies liées généralement à des temps de transit réels légèrement différents de ceux supposés lors de la simulation logicielle, ce qui peut conduire à des états instables voir même erronés.

2.5. Nomenclature du FPGA

Le client (le développeur) doit savoir comment choisir le *FPGA* le plus adéquat pour le développement de son application. Pour cela, les constructeurs suivent une nomenclature spécifiant la famille, le type du package et d'autre informations spécifique au *FPGA* vendu. Les circuits *FPGA* suivent la nomenclature suivante :



Speed grade : la vitesse du composant selon la technologie.
Type de package : c'est une collection de déclarations (types, signaux, constantes) et sous programmes utilisés fréquemment par plusieurs concepteurs.

3. Carte de développement *Spartan-3E*

Xilinx présente à ces clients deux grandes générations : *Virtex (II, V, VII,…)* et *Spartan (2, 3,6).* La génération *Spartan-3* comporte deux familles : *Spartan-3A* et *Spartan-3E* [31]. La carte de développement *Spartan-3E* de *Digilent* met de l'avant le *FPGA Spartan-3E (XC3S500E)* de *Xilinx* et ces caractéristiques uniques. Cette carte offre un environnement de conception très adaptée pour le prototypage d'applications variées dont celles des systèmes numériques à usage général et des systèmes embarqués. Cette carte est de plus idéale pour les applications de traitement vidéo et de traitement de signal en général.

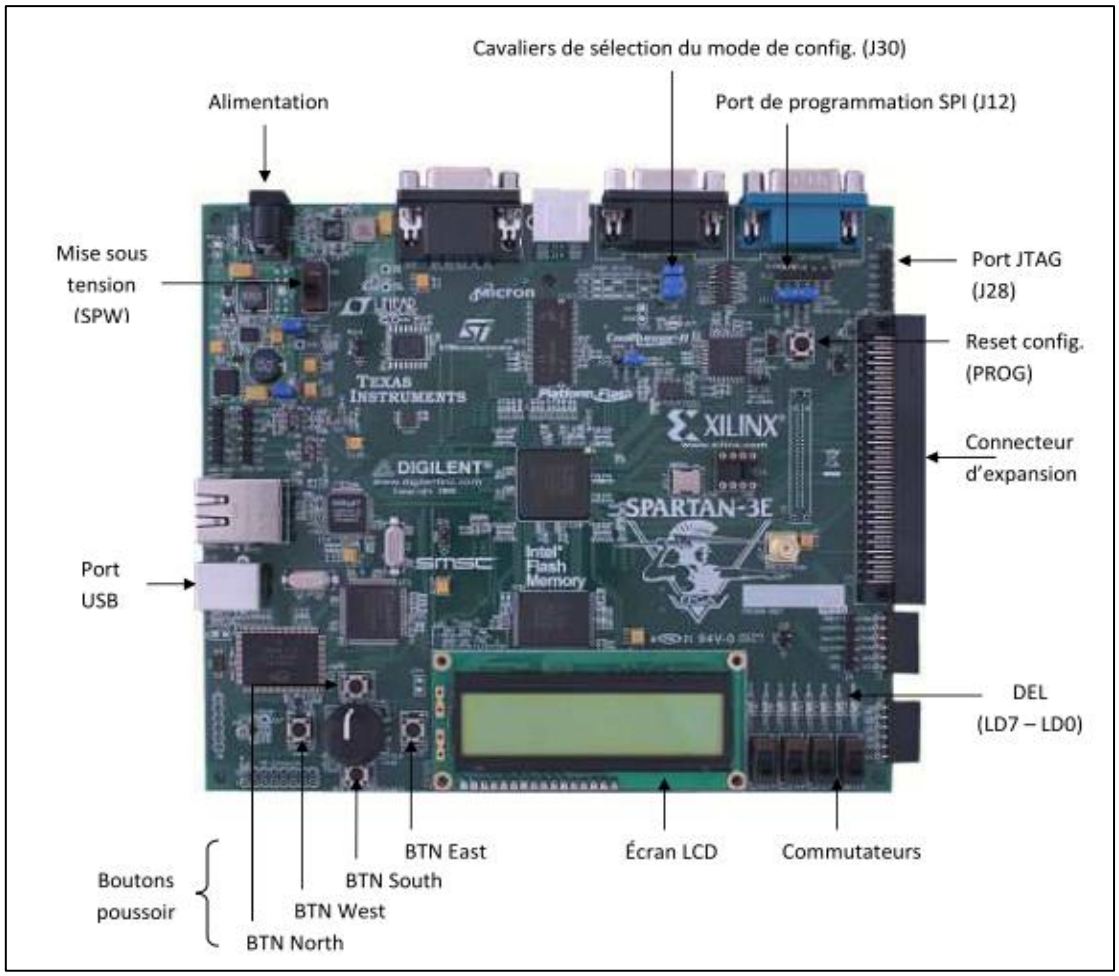


Fig. 3.13 : La carte *Spartan-3E* et ses principaux modules embarqués

La carte *Spartan-3E* (Fig3.6) regroupe entre autre

- Un *FPGA XC3S500E Spartan-3*^{E}
- Un *CPLD Coolrunner II*
- Une mémoire *PROM* de *4 Mbit*
- Une mémoire *Flash sérielle* de *16 Mbit*
- Une mémoire *Flash parallèle* de *128 Mbit*
- Une mémoire *SDRAM DDR* de *512 Mbit*
- Un écran *LCD*, un port *PS/2*, un port *VGA*
- Une prise *Ethernet 10/100*
- Deux ports *RS-232*
- Un port de configuration *USB*
- Deux convertisseurs de données

- Un oscillateur à *50MHz* un connecteur d'expansion *Hirose FX2* permettant *40 entrées/sorties* génériques
- Des *DEL*, des boutons poussoirs et des commutateurs.

La carte supporte trois modes de configuration à la mise sous tension, soit le mode *Master-Slave* utilisant le *PROM 4 Mbit*, un mode *SPI* utilisant la mémoire *Flash sériel* et un mode *BPI* utilisant la mémoire *Flash parallèle*. La carte peut aussi être programmée directement à l'aide d'un port *JTAG* lorsqu'elle est sous tension [30].

4. Le VHDL

Le *VHDL* (Very high speed integrated circuit Hardware Description Language) est un langage de description matériel. Au début, Il a été sponsorisé par le département de la défense des Etats Unis, et adopté après par le *IEEE* (Institute of Electrical and Electronics Engineering) [31]. Le *VHDL* a été au cours des conceptions électronique dès son déclaration officielle par le *IEEE* en *1987*. Pendant plus de *25 ans*, l'industrie de la conception électronique automatisée s'est développé et l'utilisation du *VHDL* est allé du document parlant des concepts initiaux de la conception vers l'implémentation et la vérification fonctionnelle [32]. Le *VHDL* est conçu de manière à nous permettre de modéliser des systèmes complexes décrits à des niveaux d'abstractions très différents. De plus, le *VHDL* est un langage modulaire et hiérarchique. Un système complexe peut être divisé en plusieurs blocs, chaque bloc peut à son tour être divisé en plusieurs sous blocs et ainsi de suite. L'interface entre les différents blocs se fait par des "liens de communication". De la même manière, un modèle *VHDL* peut être composé de plusieurs sous-modules, chaque sous-module peut être divisé à son tour en d'autres sous-modules et ainsi de suite. Le *VHDL* a d'abord été proposé comme un langage de modélisation et il a graduellement été utilisé comme un langage de synthèse. Cependant, il faut noter que les possibilités de modélisation avec *VHDL* dépassent de loin les besoins de la synthèse. Certaines caractéristiques abstraites de *VHDL* n'ont pas d'équivalent dans la logique digitale. Autrement dit, la synthèse ne supporte qu'un sous ensemble des possibilités de *VHDL*. Parmi les avantages du langage *VHDL* nous insistons sur les aspects suivants :

- Cycle de conception plus court.
- Description plus compacte, moins d'erreurs.
- Exploration de l'espace de design, re-design.
- Maîtrise de la complexité.
- Description portable (indépendante de la technologie).

4.1. Description Générale

Un module *VHDL* élémentaire est appelé entité. Il est principalement composé de deux éléments de base : l'énoncé *entity* et l'énoncé *architecture*. L'énoncé entity décrit l'interface entre l'entité et le monde extérieur (i.e. les signaux d'entrée-sortie, les broches de la puce). L'énoncé architecture décrit quant à lui son contenu interne, son comportement. Notons ici qu'une entité *VHDL* peut avoir plusieurs architectures différentes. Ceci nous permet de générer des vues différentes de la même entité à des niveaux d'abstraction différents.

4.2. Quelques règles de base

Pour illustrer les règles de base du VHDL considérons l'exemple d'un comparateur à deux bits, décrit, ci-dessus, à l'aide de deux comparateurs à un bit.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL ;
3.  entity comp2 is
4.  Port ( a : in  STD_LOGIC_VECTOR (1 downto 0) ;
5.           b : in  STD_LOGIC_VECTOR (1 downto 0) ;
6.          aeqb : out  STD_LOGIC);
7.  end comp2 ;
8.  architecture  arch_struct of comp1 is
9.  signal  e0: std_logic  ;
10. signal  e1: std_logic  ;
11. begin
12. --instantiation de deux comparateur à 1 bit
13. unit0 : entity work.comp1(comp1_arch)
14. port map (i0=>a(0) , i1=>b(0) , eg=>e0);
15. unit1:entity work.comp1(comp1_arch)
16. port map (i0=>a(1) , i1=>b(1) , eg=>e1);
17. --a et b sont égaux si et seulement si ils sont égaux bit à bit
18. aeqb <= e0 and e1;
19. end arch_struct;
```

Exemple d'une description VHDL

Voyons maintenant quelques règles qu'un développeur doit suivre pour décrire son système d'une manière correcte :

a. La bibliothèque et le package

Les deux premières lignes :

1. **library** IEEE;
2. **use** IEEE.**STD_LOGIC_1164**.ALL ;

appelle le package *std_logic_1164* de la bibiothèque *IEEE.*

Pour des raisons de flexibilité les types logiques des ports sont souvent définis au préalable dans des bibliothèques. En *VHDL*, les bibliothèques sont spécifiées par le mot clé ***library***. Pour avoir accès à une partie de la bibliothèque on utilise l'énoncé ***use***. On a donc accès dans l'entité *comp2* à tous les types définis dans le package *std_logic_1164* de la libraire *IEEE*. Cela nous permet entre autres d'utiliser le type *std_logic*.

b. Déclaration de l'entité :

La déclaration de l'entité

3. **entity** comp2 **is**
4. **Port (** a : in STD_LOGIC_VECTOR (1 downto 0) ;
5. b : in STD_LOGIC_VECTOR (1 downto 0) ;
6. aeqb : out STD_LOGIC**);**
7. **end** comp2 ;

spécifie essentiellement les signaux d'entrées-sorties du circuit. La première ligne indique que le nom du circuit est *comp2*, et la section ***port*** spécifie les signaux d'entrées-sorties, leur direction (entrée, sortie, bidirectionnel). La syntaxe générale pour la déclaration d'une entité est illustrée par la figure suivante :

```
entity <nom de l'entité> is
port(
        <Nom du port1> : <Direction> <Type Logique>;
        <Nom du port1> : <Direction> <Type Logique>;
                                                ...;
                                                );
end <nom de l'entité>;
```

Syntaxe d'une déclaration d'entité

c. Types de données

Le *VHDL* est un langage fortement typique, seules les valeurs définies peuvent être utilisées pour un objet. Les données peuvent être de type *std_logic* ou l'une de ses variantes. Le package *std_logic_1164* contient neuf valeurs. Parmi ces valeur le *'0'* logique le *'1'* logique et le *'z'* pour 'haute impédance'.

Un signal dans un circuit numérique contient, fréquemment, plusieurs bits. Le type de données *std_logic_vector*, définit comme un tableau d'éléments de *std_logic* est utilisé pour ce but. La déclaration d'un vecteur est comme suit : *nom_port* : *direction std_logic_vector (MSB downto LSB).*

d. L'énoncé architecture

La déclaration de l'architecture

```
8.  architecture  arch_struct of comp1 is
9.  signal  e0: std_logic  ;
10. signal  e1: std_logic  ;
11. begin
12. --instantiation de deux comparateur à 1 bit
13. unit0 : entity work.comp1(comp1_arch)
14. port map (i0=>a(0) , i1=>b(0) , eg=>e0);
15. unit1:entity work.comp1(comp1_arch)
16. port map (i0=>a(1) , i1=>b(1) , eg=>e1);
17. --a et b sont égaux si et seulement si ils sont égaux bit à bit
18. aeqb <= e0 and e1;
19. end arch_struct;
```

décrit le fonctionnement du circuit. Le *VHDL* permet d'introduire plusieurs énoncés d'architectures pour la même entité. L'énoncé est identifié par le nom *arch_struct* (architecture_structurelle). Elle peut contenir d'autres options supplémentaires comme les signaux internes, les constantes,…etc. Deux signaux internes ont été déclarés dans cet énoncé :

```
9.  signal  e0: std_logic  ;
10. signal  e1: std_logic  ;
```

Remarque :

Tout comme dans la plupart des langages de programmation, le *VHDL* permet d'introduire des commentaires. Les commentaires sont introduits en plaçant deux tirets consécutifs "--" et se terminent dans tous les cas à la fin de la ligne. Evidemment les commentaires sont ignorés par le compilateur *VHDL*.

La syntaxe générale pour la déclaration de l'énoncé architecture est illustré par la figure suivante :

```
architecture <nom de l'architecture> of <nom de  l'entité> is

--déclaration des signaux internes;

begin

<instructions>… ;

…

end <nom de l'architecture>
```

Syntaxe d'une déclaration de l'architecture

L'énoncé principal, inclus entre ***begin*** et ***end***, contient des instructions concurrentes. A l'inverse d'un programme en C, où l'exécution des instructions est séquentielle, les séquences concurrentes s'exécutent d'une manière analogue au fonctionnement de différentes parties d'un circuit, en parallèle. Le signal à gauche d'une instruction peut être considéré comme la sortie de la partie, et l'expression à droite spécifie le fonctionnement du circuit et les signaux d'entrée (ligne.18) [29].

Remarque :

La notion d'architecture structurelle consiste au fait que nous avons fait appel à deux comparateurs à un bit pour décrire l'architecture du comparateur à deux bits. Le comparateur à un bit doit être déjà conçu pour pouvoir l'appeler en utilisant l'instruction : ***port map***. L'appel d'un circuit conçu auparavant se fait comme suit :

```
unité :  e n t i t y  nom_bibiothèque.nom_entité
(nom_architecture)

p o r t  map(

signal_formel => signal_actuel,

…

) ;
```

Syntaxe d'un appel d'un circuit développé auparavant

Où :

unité : le nouveau identifiant donné au composant pour être utilisé plus qu'une fois selon le besoin.

nom_bibliothèque : la bibliothèque de travail. La bibliothèque par défaut est **WORK.**

e. L'énoncé process

Pour faciliter la modélisation des systèmes, le *VHDL* contient plusieurs instructions séquentielles qui s'exécutent en séquence. Comme leur comportement est différent de celui des instructions concurrentes, les instructions séquentielles sont encapsulées à l'intérieur d'un *process*. Le *process* lui-même est une instruction concurrente. Il peut être considéré comme étant une boite noire ayant un comportement décrit par des instructions séquentielles. Un *process* mal codé mène fréquemment inutilement à une implémentation plus complexe ou ne peut pas être synthétisé. Les instructions les plus utilisées dans un *process* sont : *if* et *case* [29].

5. Le flot de développement

Le flot simplifié de développement d'un système sur cible *FPGA* est montré par la figure (3.14). Pour plus de simplicité de lecture, nous poursuivons les termes utilisés dans la documentation de *Xilinx*. La partie gauche du flot est le processus de programmation et de la finition, dans lequel un système est transformé d'une description en texte *HDL* vers la configuration d'un dispositif de niveau cellulaire qui sera téléchargée sur la cible *FPGA*. La partie droite est le processus de validation, qui vérifie si le système a atteint les performances exigées ou non.

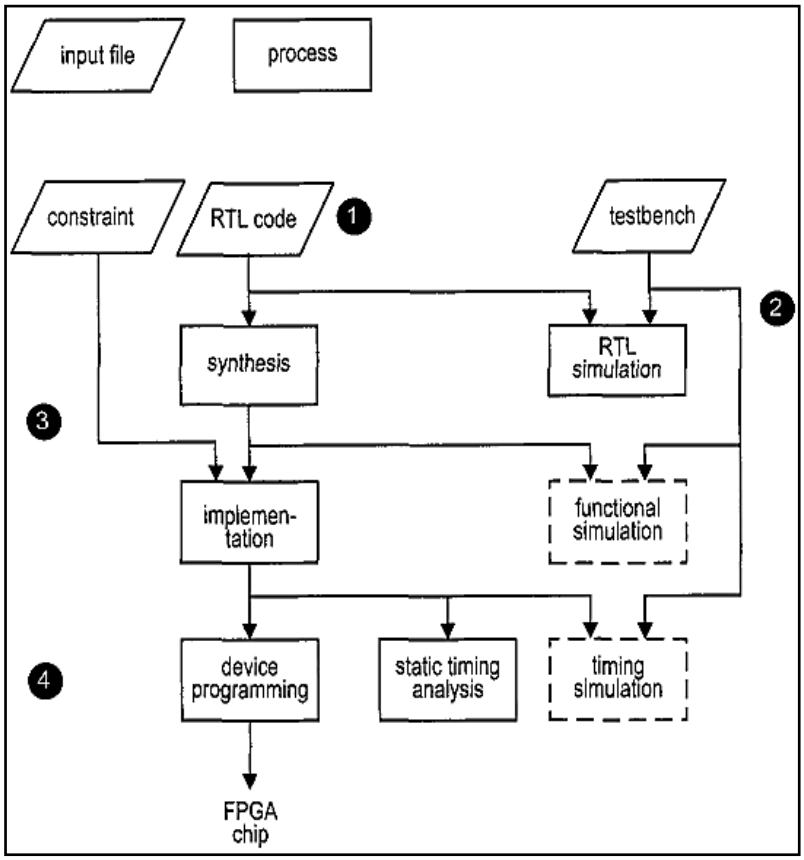


Fig. 3.14 : Le flot de développement sur une cible FPGA

Les principales étapes sont:

- **Concevoir le système et dériver le fichier HDL :** Nous aurons besoins d'écrire un fichier séparé de contraintes pour spécifier certaines contraintes d'implémentation.

- **Ecrire un testbench en HDL et performer une simulation RTL :** Le terme *RTL* tient au fait que le code *HDL* est fait dans un niveau comportemental.

- **Réaliser la synthèse et l'implémentation :** Le processus de synthèse est souvent dit : synthèse logique, dans lequel le logiciel transforme le code *HDL* à des composants génériques, comme les portes logiques simples et les bascules. Le processus d'implémentation passe par trois petits processus: *traduction*, *mapping*, *placement et routage*. Le processus de traduction fusionne plusieurs fichiers en un *netlist*. Le processus de *mapping* qui est, généralement, connu sous le nom de : *technolgy mapping* (mapping technologique), fait la liaison entre les portes génériques, dans le *netlist* crée, les cellules logiques du *FPGA* et le *IOBs*. Le processus de placement et de routage réalise la conception physique à l'intérieur de la puce *FPGA*. Il place les cellules dans leurs locations physiques et définit le chemin que doit prendre chaque signal. Dans le flot de *Xilinx*, *static timing analysis* (analyse statique du temps), qui détermine les différents paramètres temps, comme le temps de retard maximal et la fréquence maximale de l'horloge, est réalisé à la fin du processus d'implémentation.

- **Générer et télécharger le fichier de programmation (généralement, le *bitstream*) :** Dans ce processus un fichier de configuration est généré en accord avec le *netlist* final. Le fichier est téléchargé vers une cible *FPGA*. Le circuit physique peut alors être vérifié.

La simulation fonctionnelle peut être faite après la synthèse, et la simulation temporelle peut être réalisée après l'implémentation. La simulation fonctionnelle utilise le *netlist* synthétisé pour remplacer la description *comportementale* et vérifier l'exactitude de la synthèse. La simulation temporelle utilise le *netlist* final, avec des données détaillées sur le temps, pour faire la simulation. A cause de la complexité du *netlist* final, les deux simulations peuvent prendre un temps considérable. Si la description de la conception est écrite d'une manière convenable et efficace, le code *HDL* sera synthétisé et implémenté correctement. Dans ce cas nous n'aurons besoins qu'à utiliser la simulation comportementale pour vérifier l'exactitude du code *HDL* et à utiliser l'analyse statique du temps pour examiner les informations sur le temps. Les deux autres simulations peuvent alors être enlevées du flot de développement [29].

6. Outil de développement ISE 12.3

L'outil de développement *Xilinx ISE* permet de réaliser les étapes essentielles suivante Créer un projet et le code *HDL (VHDL* ou *Verilog)*.

- Créer un *testbench* (banc d'essai) et réaliser une simulation *RTL*.
- Ajouter un fichier de contraintes.

- synthétiser et implémenter le code.
- Générer et télécharger le ficher de configuration vers la cible *FPGA*.

Remarque :

La première étape n'est nécessaire que pour simuler le fonctionnement du processus développé. Elle peut être surpassée [29].

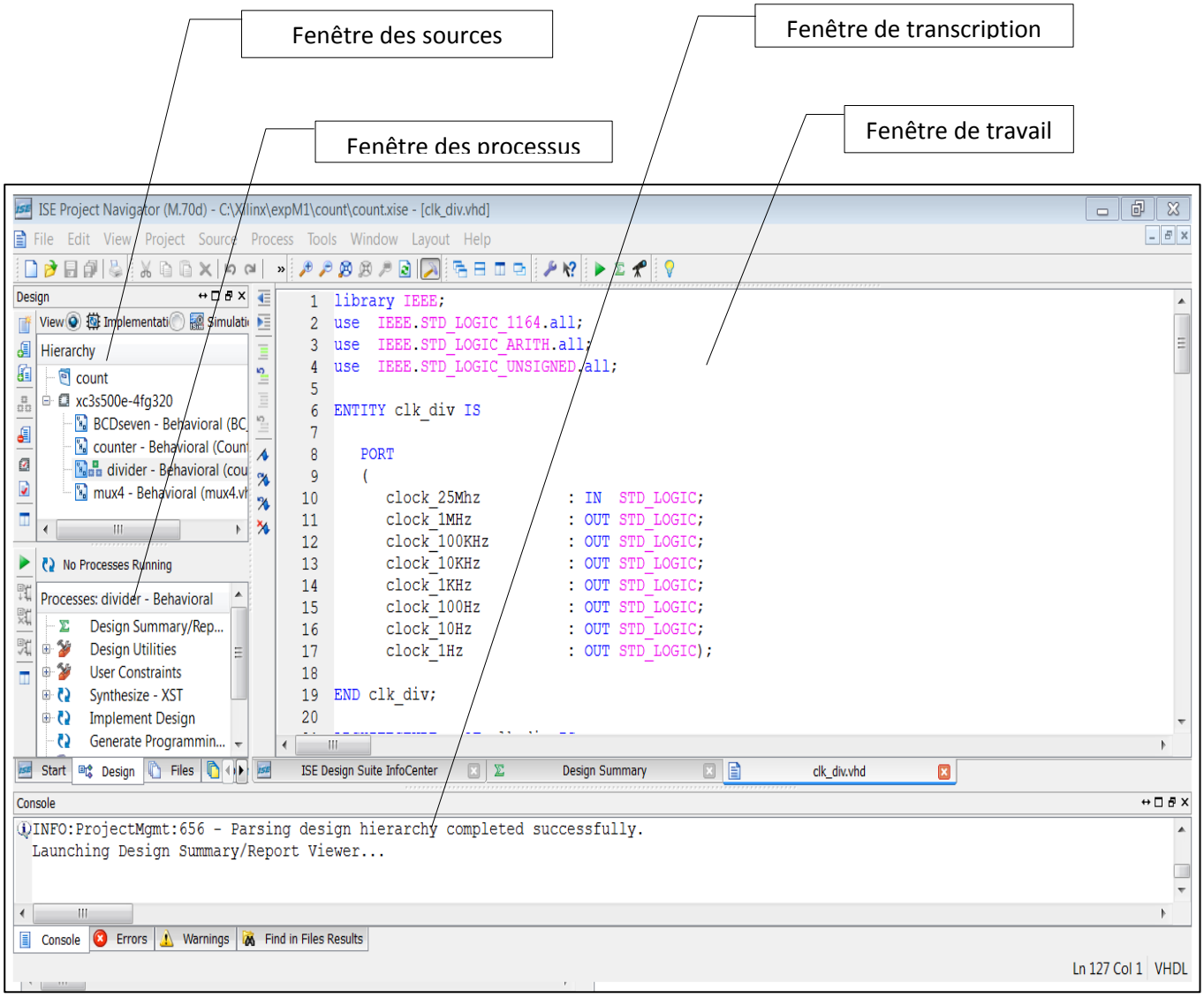


Fig 3.15 : l'environnement de l'outil de développement *Xilinx ISE 12.3*

La fenêtre par défaut de l'*ISE* est montrée par la figure (3.15). Elle est divisée en quatre sous fenêtres :

- Fenêtre des sources : affiche hiérarchiquement les fichiers inclus dans un projet.
- Fenêtre de processus : affiche les processus qui sont valables pour la source sélectionnée.
- Fenêtre de transcription : affiche les messages de l'état, des erreurs et des avertissemenst.

- Fenêtre de travail : contient des fenêtres pour plusieurs documents (code *HDL* Schématique,…etc.) qui peuvent être vus ou modifiés.

Chaque fenêtre peut être réduite, agrandie ou fermée. La disposition par défaut peut être restaurée en sélectionnant : *layout* → *load default layout*. La référence [33] donne en détaille les différentes manipulations possibles pour développer une application en *VHDL* ou en schématique.

7. System Generator

System Generator est un outil de traitement numérique des signaux de la firme *Xilinx* qui s'intègre à l'environnement *Simulink* pour la modélisation des systèmes en schéma bloc. L'expérience précédente avec les méthodes de conception de type *RTL* (Register Transfer Level) des *FPGAs Xilinx* ne sont pas demandés pour pouvoir utiliser *System Generator*.

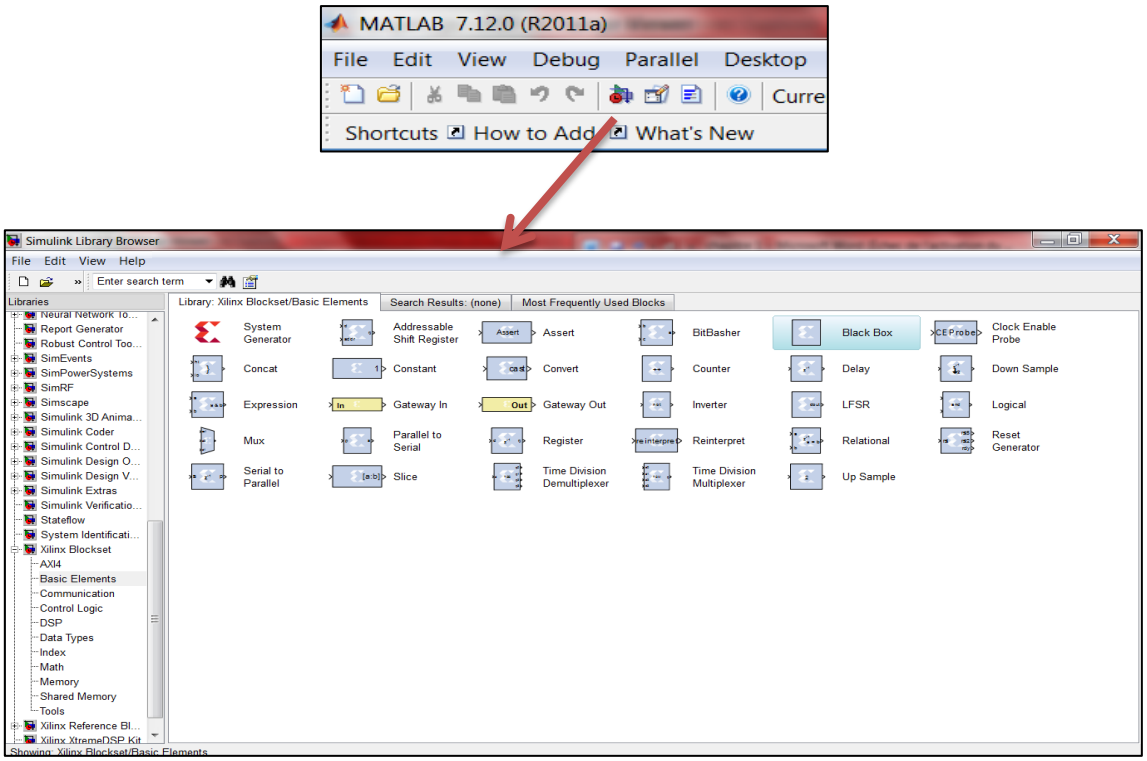


Fig. 3.16 : *Xilinx Blockset* dans l'environnement *Simulink*

La conception peut être faite dans l'environnement *Simulink* ce qui permet une manipulation facile. La synthèse, le placement et le routage sont automatiquement performés pour générer le fichier de programmation [34].

Plus de 90 blocs de construction *DSP* (Digital Signal Processor) sont fournis dans le jeu de blocs *DSP* de *Xilinx* pour Simulink (Fig3.16). Ces blocs comprennent les blocs de construction commune *DSP*

tels que des additionneurs, des multiplicateurs et des registres. D'autres blocs sont également inclus tels que les blocs *FFT* (Fast Fourier Transformer) et les filtres.

7.1. Définition de quelques blocs

a. Le bloc System Generator

Le bloc *System Generator* permet de contrôler les paramètres du système et de la simulation. Tous les modèles *Simulink* contenant les blocs de *Xilinx* doivent contenir au moins un bloc *System Generator*. Une fois ce dernier est ajouté à un modèle, il est possible de spécifier la manière dont la génération de code et la simulation doivent être effectuées (Fig3.17).

Fig. 3.17 : Le bloc *System Generator*.

En double cliquant sur le block *System Generator*, on peut choisir la cible *FPGA*, le mode de compilation et paramétrer la simulation.

Remarque:

En contrepartie des blocs de *Simulink* les blocs de *System Generator* nécessitent une configuration assez délicate. Il faut marquer ici que le type de données en entrée et en sortie doit être bien spécifié. C'est la transition du *System Generator* vers le *Simulink* ou l'inverse qui cause plus de difficultés lors de la modélisation des systèmes dans *Simulink.* Pour permettre cette transition deux autres blocs sont obligatoirement utilisés : le *Gateway in* et le *Gateway out.*

b. Black Box

Le *Black Box* de *Xilinx*, est répertorié dans les bibliothèques suivantes : *Xilinx Blockset*, *Basic elements*. Il nous permet d'introduire des fonctions qui n'existent pas dans la bibliothèque du *System Generator* par des scripts *VHDL*. Nous l'avons utilisé pour simuler le fonctionnement des circuits décrits en *VHDL* au cours de notre travail.

8. Conclusion

A travers ce chapitre, nous avons, en premier lieu, décrit les caractéristiques essentielles des *FPGAs* en abordant leur architecture interne et leur configuration. En second lieu nous nous sommes intéressés au langage de description matérielle *VHDL* et aux outils qui permettent de décrire et synthétiser une conception suivant le flot de développement donné dans ce chapitre.

Les circuits *FPGA* présentent de nombreux avantages :

Le premier avantage est la souplesse de programmation qui permet l'emploi conjoint des solutions logicielles et matérielles, ce qui permet de multiplier les essais, d'optimiser de diverses manières l'architecture développée, de vérifier à divers niveaux de simulation la fonctionnalité de cette architecture.

Le second avantage est évidemment la nouvelle possibilité de reconfiguration dynamique d'un circuit ce qui permet d'une part, une évolutivité assurant la possibilité de couvrir des nouveaux besoins sans nécessairement modifier l'architecture dans sa totalité.

Chapitre 4

Implémentation et essais pratiques

1. Introduction

La commande en vitesse du moteur à courant continu via la carte de développement Spartan-3E nécessite la mise en œuvre d'une carte d'interface et une carte de puissance. Différents circuits sont nécessaires pour la génération des signaux PWM et la mesure de vitesse.

Dans ce chapitre, après avoir donné une brève description des différentes parties du banc d'essais, nous envisageons la synthèse de chaque module sous l'environnement ISE, accompagnée d'un résumé sur le nombre d'éléments utilisés. Enfin, les résultats de simulation et ceux des essais pratiques sont présentés et commentés.

2. Constitution du banc d'essais

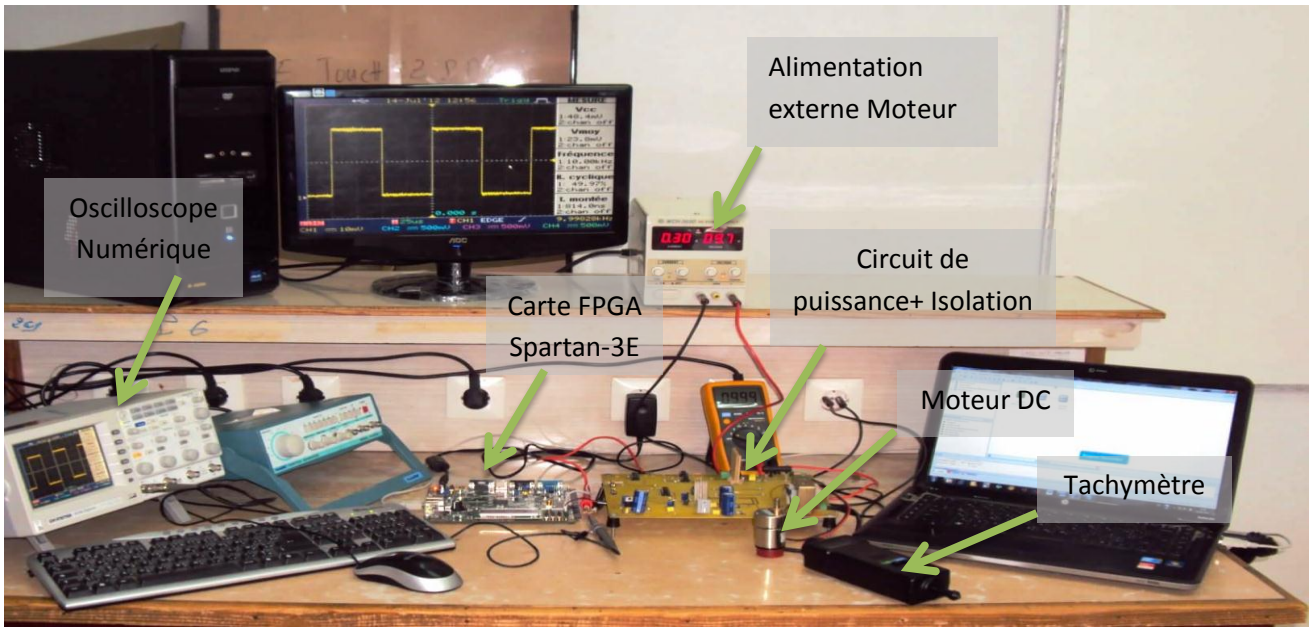


Fig.4.1 : Vue général du dispositif expérimental

Le banc d'essai que nous avons réalisé, illustré dans la figure (4.1) est constitué des éléments suivant :

- La carte de développement Spartan-3E
- Un micro-ordinateur
- Un circuit d'isolation
- Un circuit de puissance
- Un oscilloscope numérique
- Une alimentation externe pour le moteur
- Une alimentation stabilisée de 5V
- Un tachymètre

Ce dispositif va nous permettre de synthétiser, simuler, tester et valider le système réalisé au cours de notre travail. Le système réalisé est illustré par son schéma synoptique suivant :

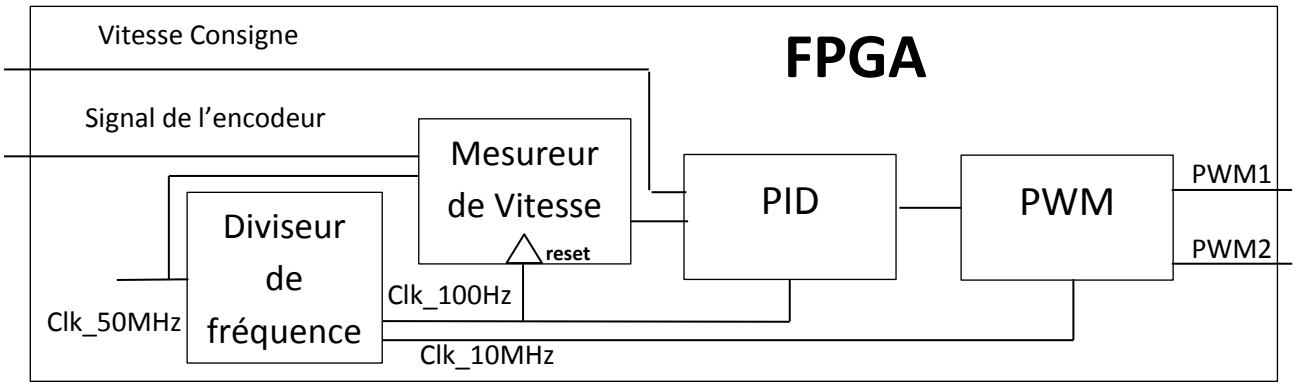


Fig. 4.2 : Schéma synoptique du système réalisé

3. Description des éléments du banc d'essais

3.1. La carte de développement FPGA SPARTAN-3E

La carte FPGA Spartan-3E, décrite dans le chapitre précédent, constitue le circuit numérique de commande. Nous allons décrire les modules utilisés lors de nos expériences.

a. Le circuit FPGA XC3S500E

Le XC3S500E possède cinq éléments fonctionnels fondamentaux :

- Des CLBs
- Des IOBs
- Des blocs RAM
- Des blocs multiplieurs
- Des DCMs.

b. Les périphériques

- Port USB

La carte FPGA Spartan-3E contient un port USB standard de type B. Ce port est dans la partie gauche de la carte juste à côté du port Ethernet. Une fois que la carte est connectée au PC d'une manière correct via ce port, une LED verte s'allume indiquant son bon fonctionnement.

- Sources d'horloge

La carte FPGA Spartan-3E possède une horloge de 50MHz, avec un rapport cyclique compris entre 40% et 60%. Pour introduire une horloge externe, le signal d'entrée doit être connecté au connecteur SMA.

Fig.4.3 : Les horloges de la carte SPARTAN-3E

- L'expansion de connecteurs (broches d'Entrées/Sorties)

La carte FPGA Spartan-3E offre une variété de connecteurs pour une interface facile et flexible avec les circuits externes. La carte possède les expansions d'entrées/sorties suivants :

- Un connecteur Hirose possédant 100 broches dont 43 entrées/sorties sont à usage général.
- Trois modules périphériques de 6 broches chacun.
- Un autre connecteur sans extensions physiques sur la carte.

Nous nous sommes servis des modules périphériques de 6 broches J1, J2 et J4. Un exemple de connexions entre le module J1 et le FPGA est illustré dans la figure suivante :

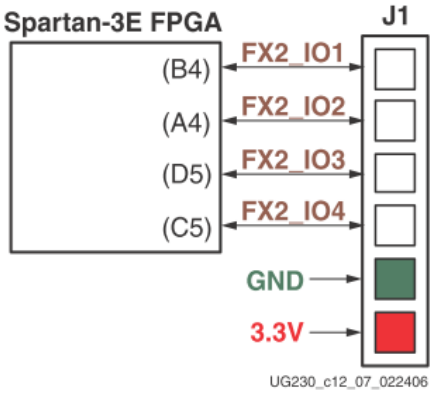


Fig.4.4 : Les connections entre le FPGA et le module périphérique J1

Remarque :

Ces modules servent comme entrées/sorties (IO1, IO2,…). Chaque module possède 4 broches entrée/sortie, un GND et un Vcc de 3.3V.

- Les commutateurs et les boutons poussoirs

La carte FPGA Spartan-3E possède quatre commutateurs localisés dans le côté droite de la carte. Ces commutateurs sont étiquetés SW3 à l'extrême gauche jusqu'à SW0 à l'extrême droite, comme le montre la figure (4.5). Ils vont nous permettre d'introduire la consigne codée en binaire. Un commutateur dans la position HIGH introduit une tension de 3.3V. Par contre un commutateur en position LOW introduit une tension de 0 V.

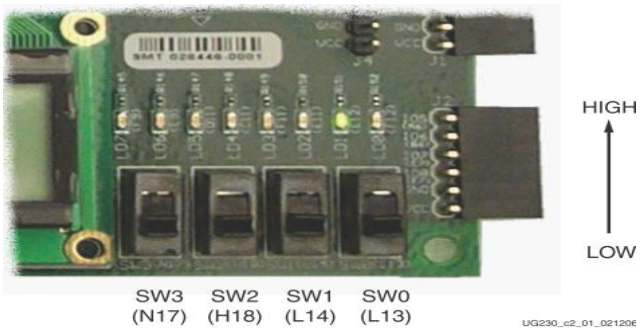


Fig.4 .5 : Les commutateurs

La carte possède aussi quatre boutons poussoirs qui servent comme entrées. Un exemple de leur utilisation est l'introduction d'un signal 'reset' pour la remise à zéro d'un processus.

- Les LEDs

La carte FPGA Spartan-3E a huit LEDs situés au-dessus des commutateurs comme le montre la figure (4.6). Les LEDs sont etiquetés LED7 jusqu'à LED0. Ces LEDs servent à la lecture en binaire ou en hexadécimal des résultats obtenus lors des expériences.

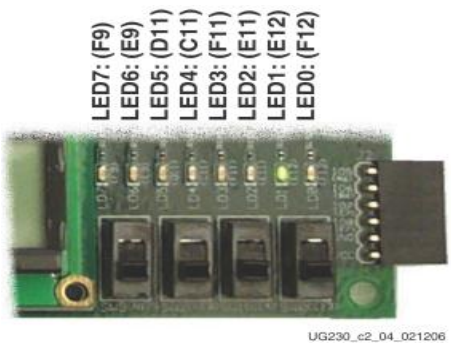


Fig.4.6 : Les LEDs

3.2. La carte de puissance

a. Circuit de puissance

Le circuit de puissance est à base du circuit intégré L6203, ce dernier est un hacheur à quatre quadrants (annexe A5), alimenté avec une tension pouvant atteindre les 48V et un courant de 5A. Notre moteur peut absorber un courant maximal de 2A, donc pour garantir le fonctionnement attendu du moteur, le courant demandé par ce dernier doit être disponible à tout moment, plus particulièrement si nous devons introduire des charge assez importantes.

La fréquence de commutation utilisée dans notre projet est de l'ordre de 10 KHz. Le schéma électrique du circuit de puissance est illustré par la figure suivante :

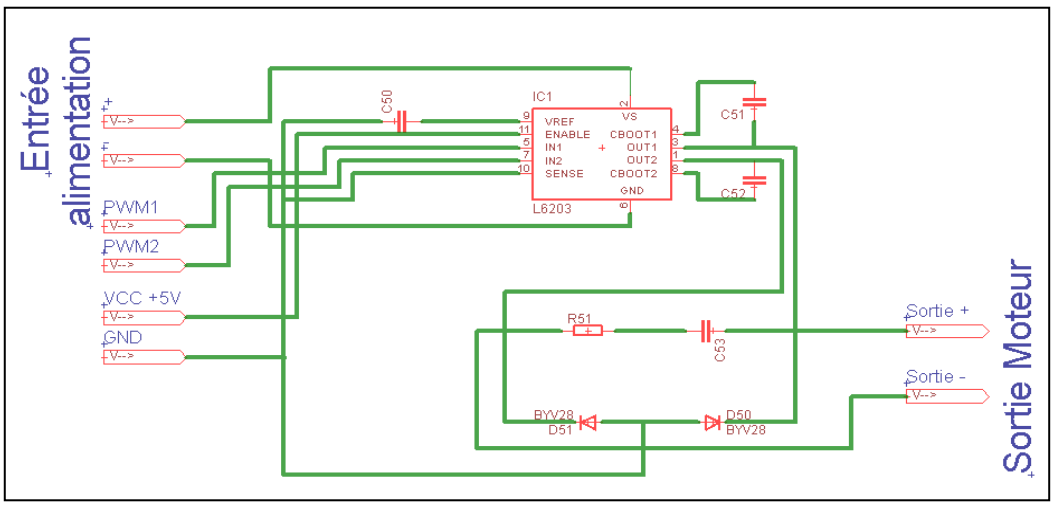


Fig.4.7: Schéma électrique de la carte de puissance

b. Circuit d'isolation

Vu que le FPGA est très sensible aux perturbations extérieures (par exemple une pin qui est mise par erreur à la masse), un circuit d'isolation entre la carte de développement et la carte de puissance est utilisé. Notre choix s'est porté sur des opto-coupleurs de la série 6N137 (annexe B2). La figure 4.10 illustre le schéma électrique utilisé.

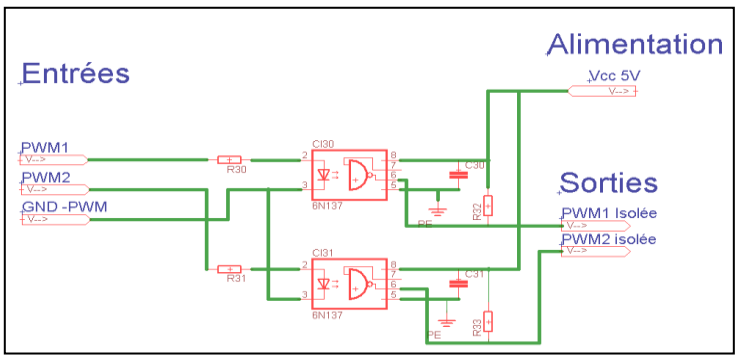


Fig. 4.8 : Schéma électrique du circuit d'isolation

c. Alimentation 3.3V

Pour pouvoir alimenter le circuit intégré HEF4050B, une alimentation de 3.3V à partir du régulateur de tension LM317 (annexe A2) est nécessaire et représenté par le schéma électrique suivant :

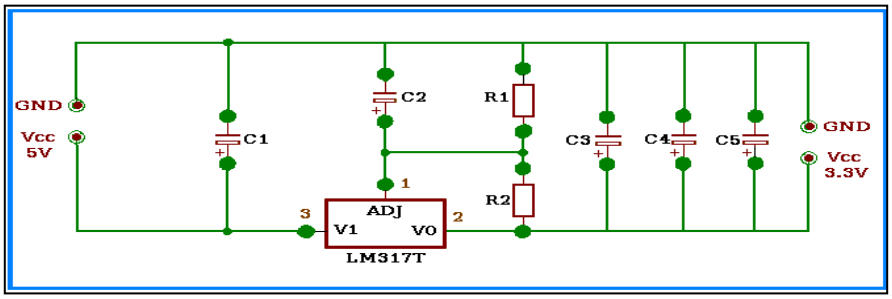


Fig. 4.9 : Schéma électrique de l'alimentation 3.3V

d. L'alimentation de 5V

La plupart des circuits électriques utilisés sont à base de circuits intégrés qui fonctionnent avec une tension de +5V. Pour cette raison, un circuit est nécessaire pour avoir cette tension à partir du secteur. Son schéma électrique est le suivant.

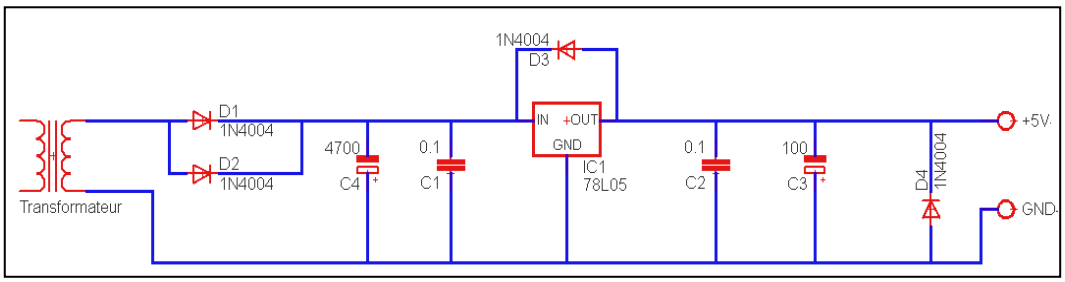


Fig.4.10 : Schéma électrique du circuit d'alimentation stabilisée de + 5V

Le circuit de puissance, le circuit d'isolation et le l'alimentation de 5V sont implantés sur la même carte représentée par la figure suivante :

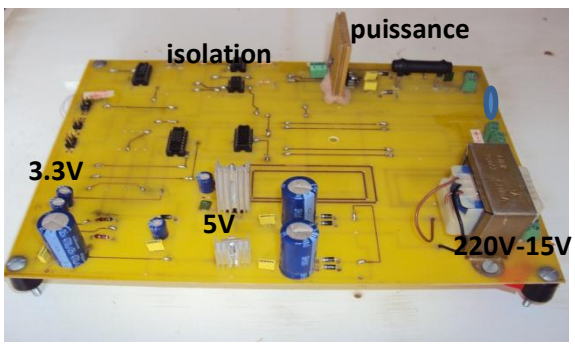


Fig. 4.11 : Photo de la carte de puissance

3.3. Le Tachymètre

Le tachymètre utilisé travail selon deux modes de mesures : mesure optique et mesure par contact. Il va nous permettre de valider les résultats de mesures obtenus par le module de mesure synthétisé.

4. Résultats de simulations et essais pratiques

Afin de maitre en œuvre le système qui permet d'asservir la vitesse du moteur à courant, en utilisant un régulateur PID, nous avons mis au point plusieurs modules VHDL. Le fonctionnement de chaque module a été simulé sous l'environnement ISE, en utilisant l'outil ISim, puis validé par des essais pratiques.

4.1. Diviseur d'horloge

Comme il a été mentionné dans le paragraphe précédent, la carte Spratan-3E possède une horloge de 50 MHz et un connecteur qui permet d'introduire une horloge externe. Comme nous avons besoins d'une horloge de 10MHz et une autre de 100Hz nous avons pensé à concevoir deux diviseurs de fréquences, un pour diviser l'horloge interne et l'autre pour diviser l'horloge externe. Le schéma synoptique de la figure suivante introduit le principe de ce diviseur :

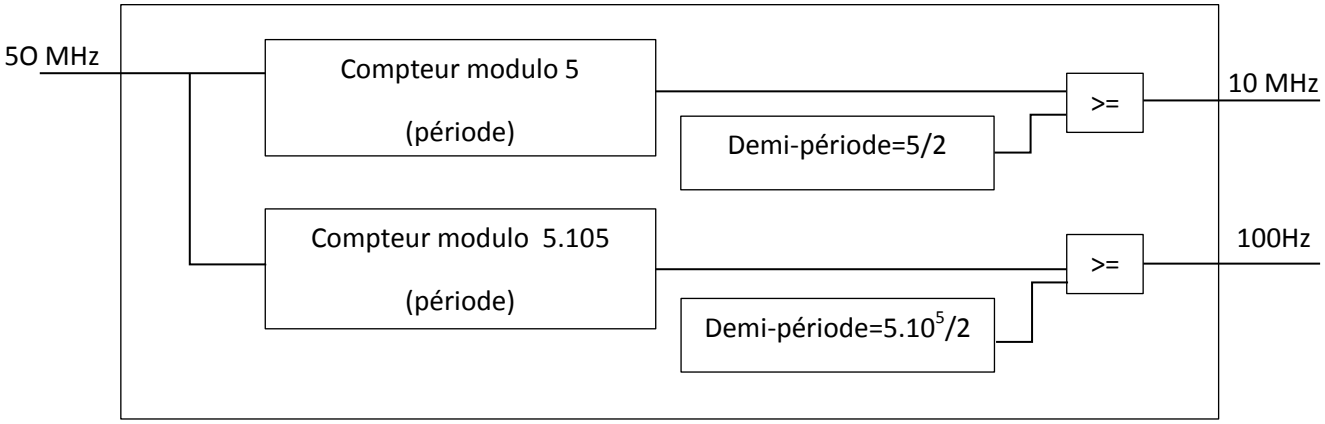


Fig.4.12 : Schéma synoptique du diviseur d'horloge

La figure suivante présente le schématique du diviseur conçu :

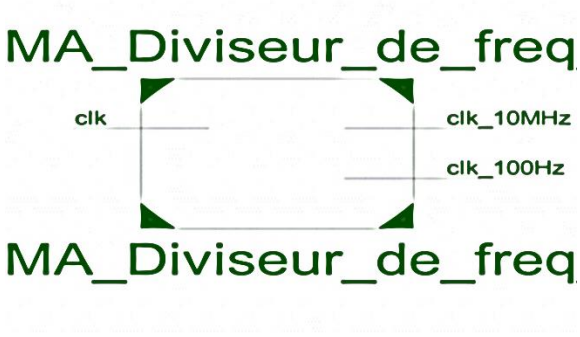
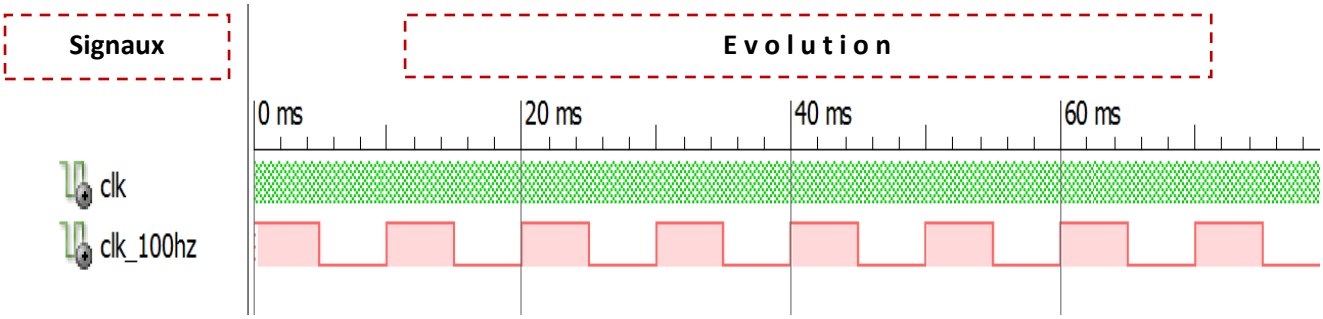


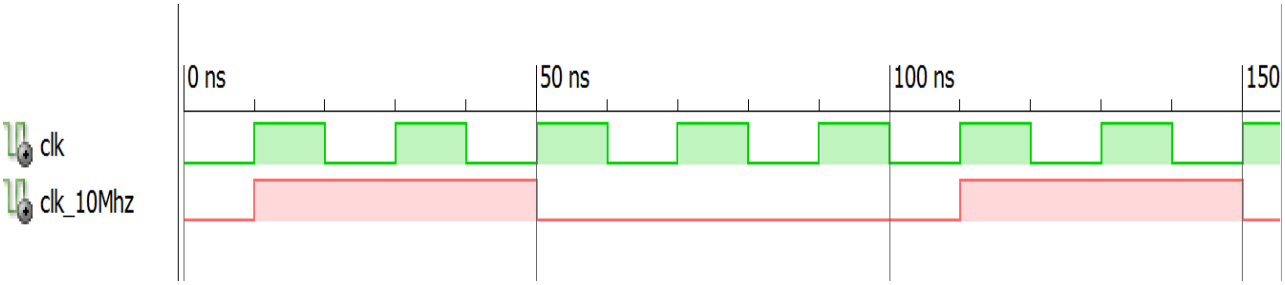
Fig. 4.13 : Schématique du diviseur de fréquence

a. Simulation sous ISim

- Division de l'horloge interne



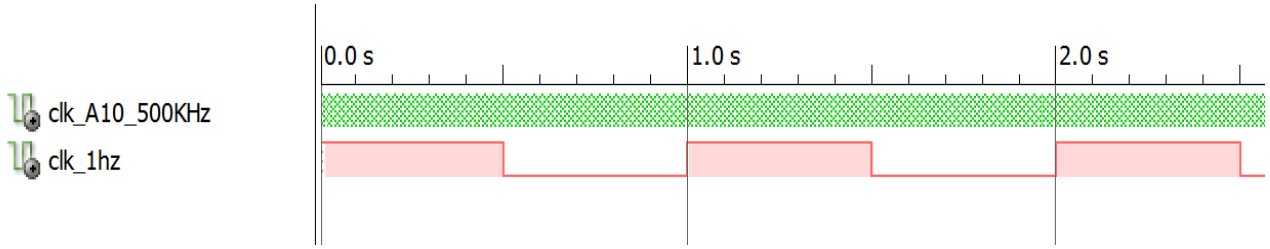
(a) Synthèse d'une horloge de 100 Hz



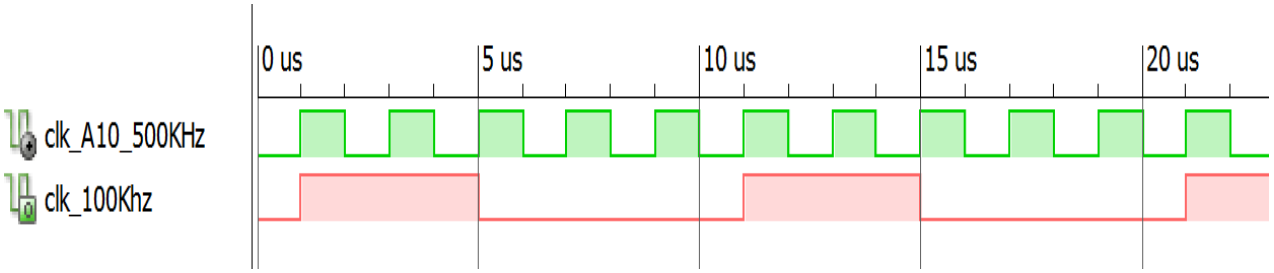
(b) Synthèse d'une horloge de 10 MHz

Fig. 4. 14 : Résultats de la division de l'horloge interne

- Division de l'horloge externe



(a) Synthèse d'une horloge de 1 Hz

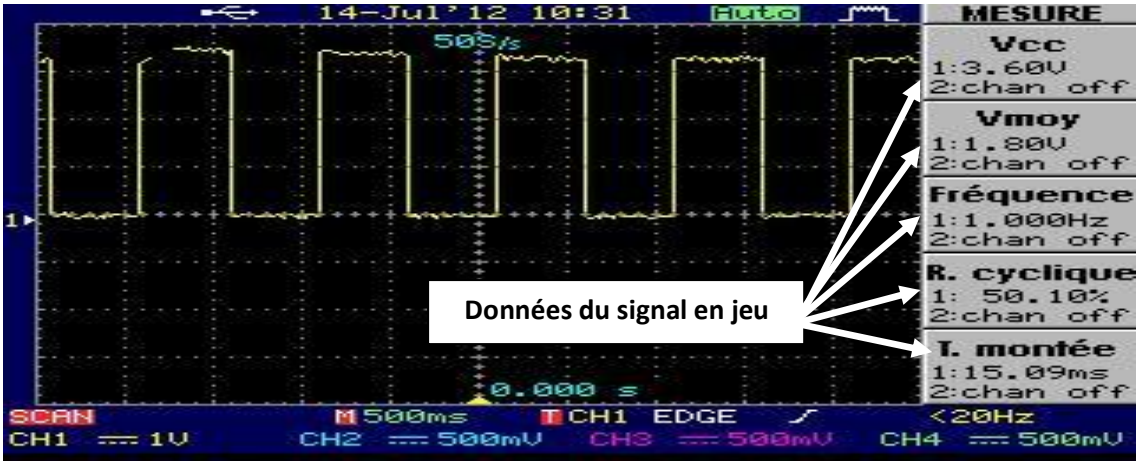


(b) Synthèse d'une horloge de 100 KHz

Fig. 4.15 : Résultats de la division de l'horloge externe

c. Résultats pratiques

- Division de l'horloge interne



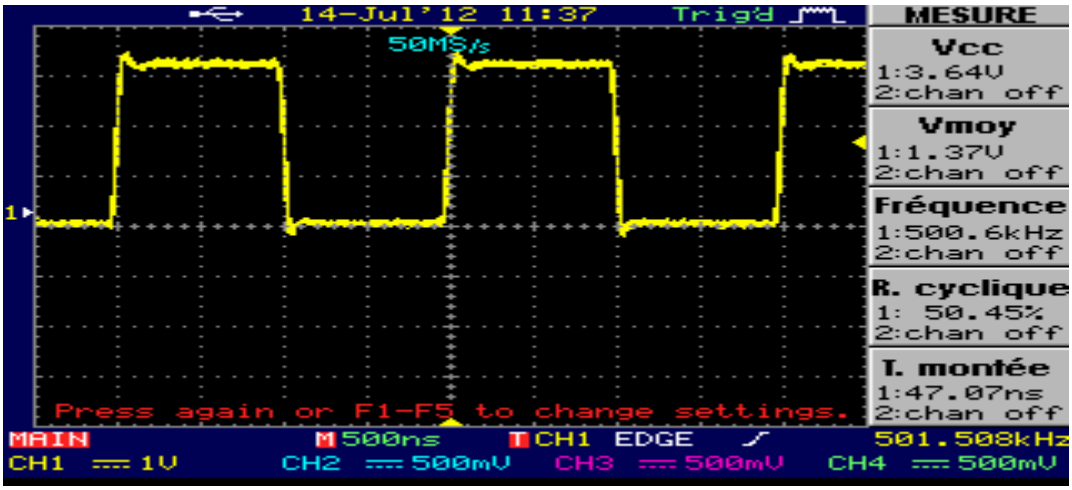
(a) Synthèse d'une horloge de 1 Hz

(b) Synthèse d'une horloge de 100 Hz

(c) Synthèse d'une horloge de 1 MHz

Fig. 4.16 : Résultats pratiques de la génération des horloges nécessaires à partir de l'horloge interne

- **Division de l'horloge externe**



(a) Signal de l'horloge externe

(b) Synthèse d'une horloge de 1 Hz

(c) Synthèse d'une horloge de 100 KHz

Fig. 4.17 : Résultats pratiques de la génération des horloges nécessaires à partir de l'horloge externe

Remarque :

Ces deux modules vont servir dans la génération des signaux d'horloge pour les autres modules (générateur de signaux PWM et module de mesure de la vitesse).

4.2. Générateur des signaux PWM

Le signal PWM est la grandeur qu'à travers laquelle le correcteur peut agir sur le processus. D'où son importance capitale dans le système de commande. Pour générer ce signal nous avons développé un module générateur de signaux PWM basé sur l'architecture suivante :

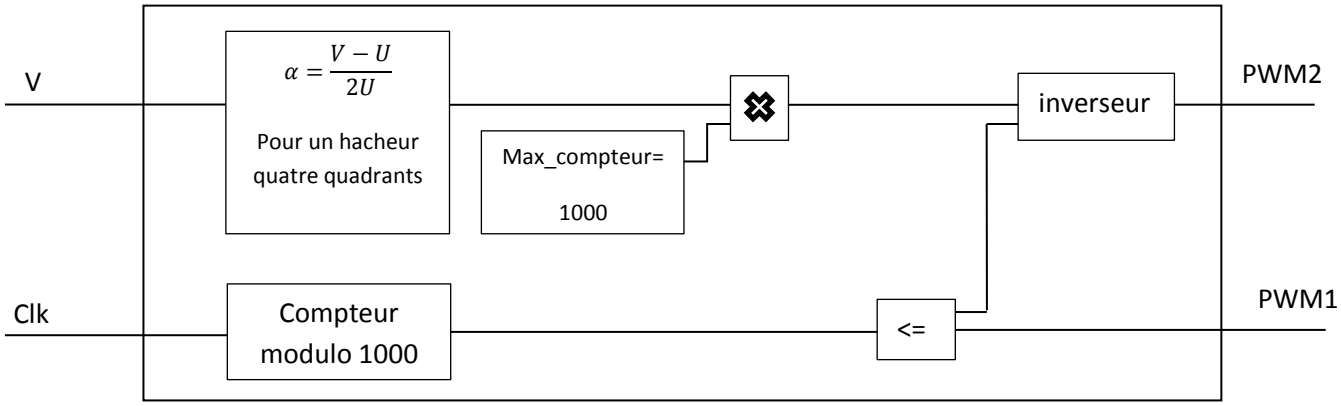


Fig.4.18: Schéma synoptique du générateur des signaux PWM

La représentation schématique du module est donnée par la figure suivante :

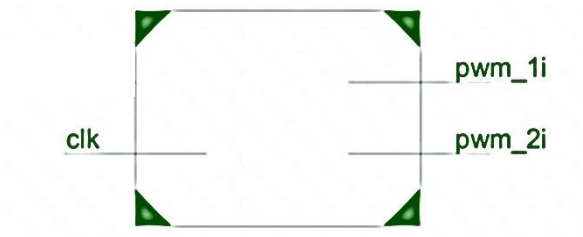


Fig. 4. 19: Représentation schématique du générateur des signaux PWM

a. Simulation sous ISim

Les résultats de la simulation du fonctionnement du générateur des signaux PWM, pour trois valeurs du rapport cyclique, α=50%, α=75% et α=95% sont donnés dans les figures 4.20 à 4.22 :

- **Pour α=50%**

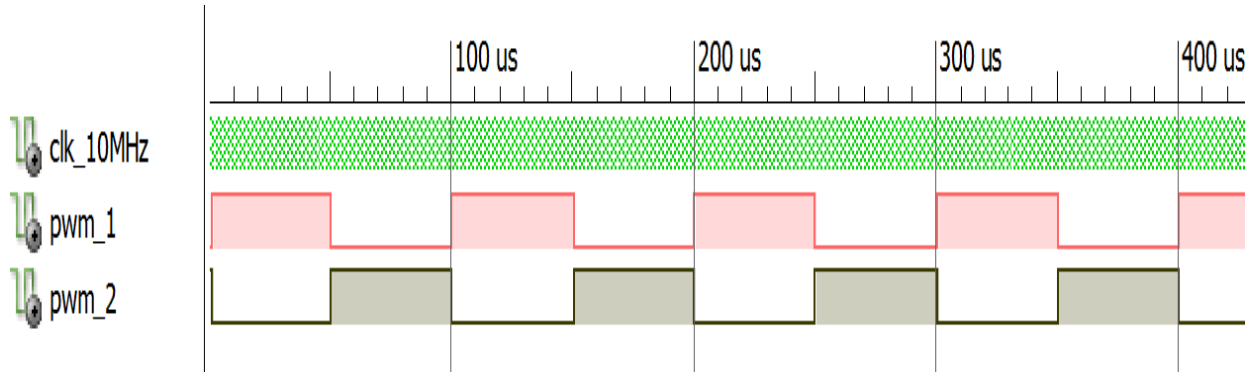


Fig. 4.20: Chronogrammes des signaux PWM1 et PWM2 pour α=50%

- **Pour α=75%**

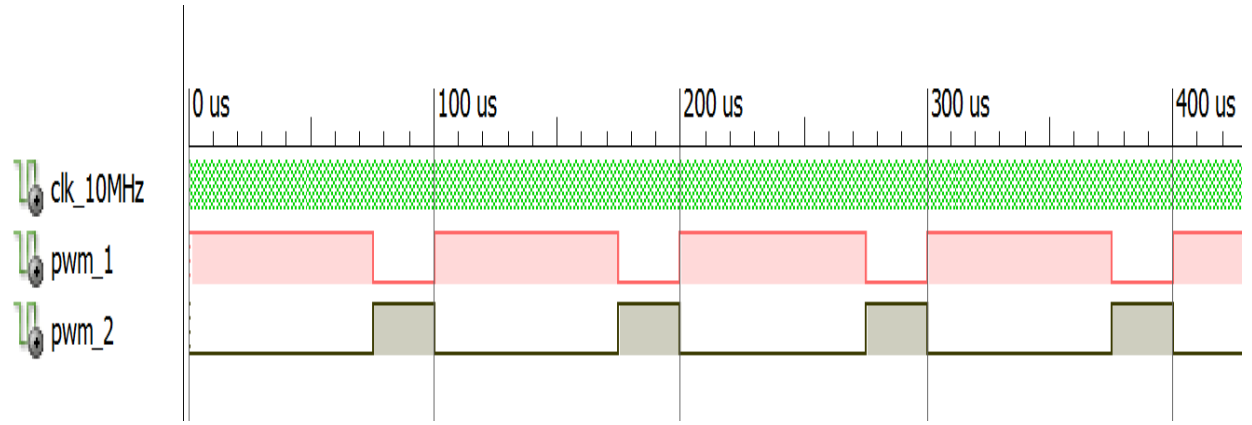


Fig. 4.21 : Chronogrammes des signaux PWM1 et PWM2 pour α=75%

- **α=95%**

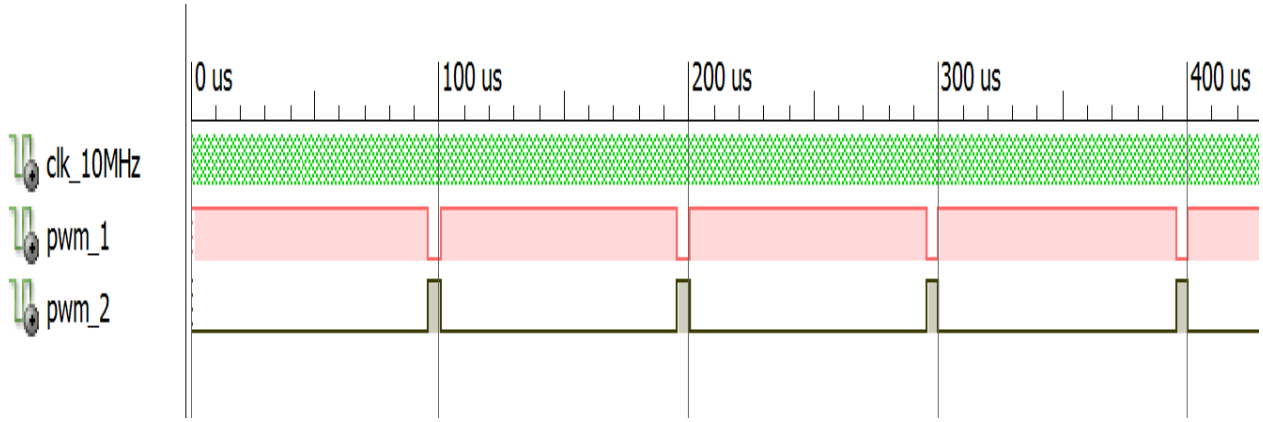


Fig. 4.22 : Chronogrammes des signaux PWM1 et PWM2 pour α=95%

b. Résultats pratiques

Pour s'assurer du bon fonctionnement du module, nous avons effectués des essais pour différents rapports cycliques. Les mesures, prises par un oscilloscope numérique, sont données dans les figures 4.23 à 4.25.

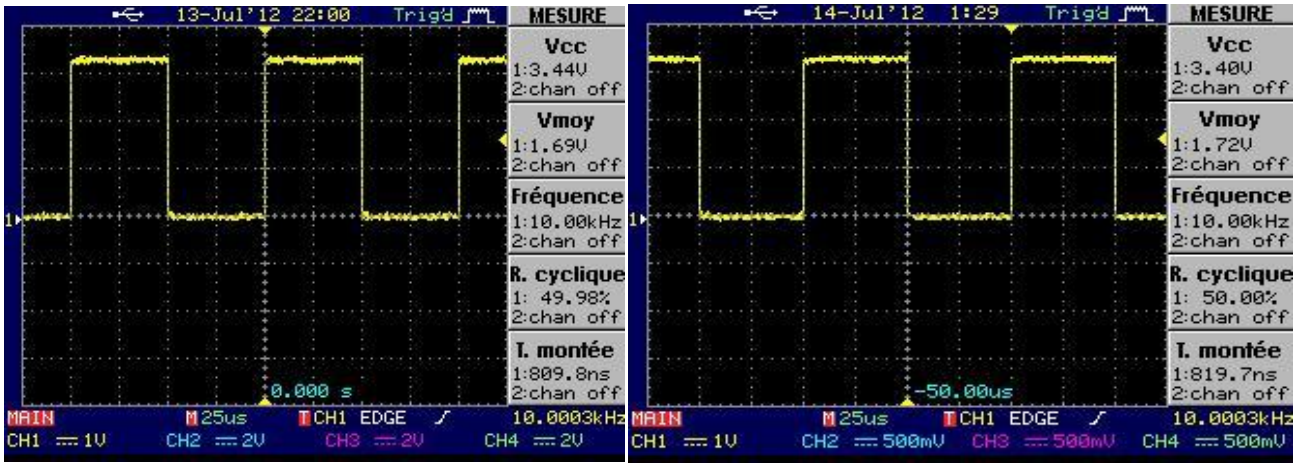


Fig.4.23 : Les signaux PWM1 et PWM2 pour un rapport cyclique α=50%

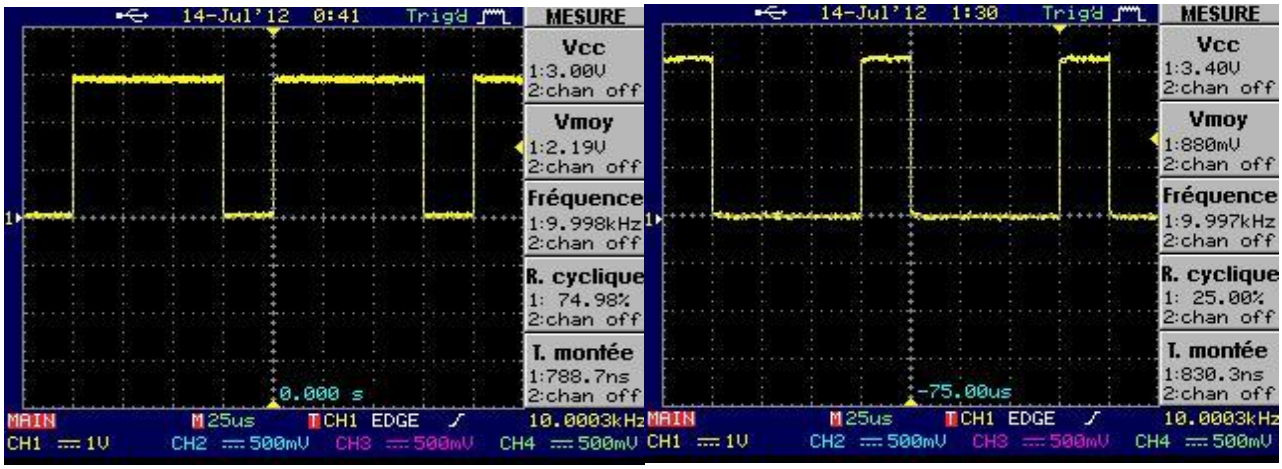


Fig.4.24 : Les signaux PWM1 et PWM2 pour un rapport cyclique α=75%

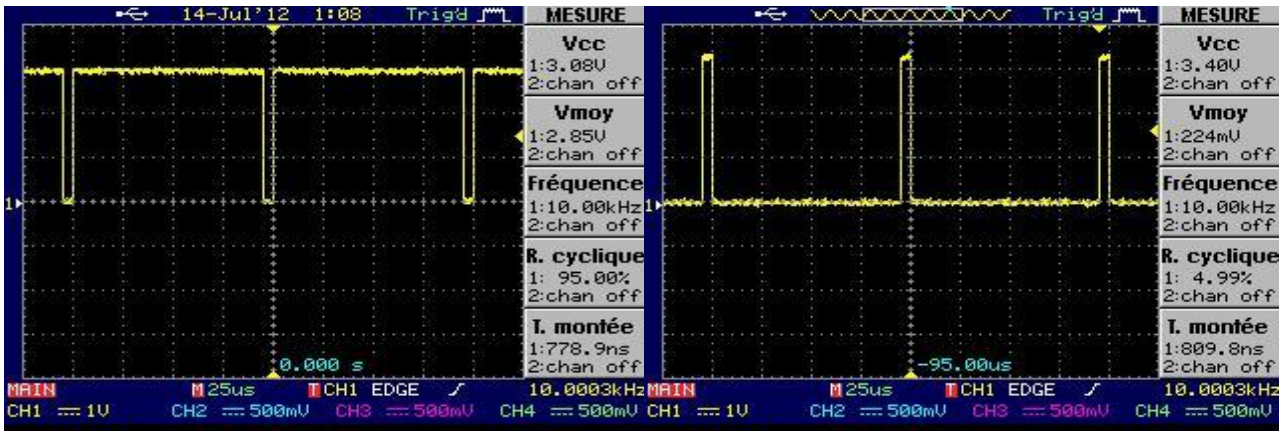


Fig.4.25 : Les signaux PWM1 et PWM2 pour un rapport cyclique α=95%

Remarque :

Nous remarquons bien que les résultats des essais pratiques coïncident avec ceux obtenus par simulation. Ceci, montre le bon fonctionnement du module de génération des signaux PWM.

4.3. Mesure de vitesse

Obtenir la valeur réelle de la vitesse de rotation du moteur, est une tâche très essentielle pour la commande en vitesse du moteur à courant continu. Ce dernier possède un encodeur qui génère un signal carré dont le nombre d'impulsions par unité de temps détermine la vitesse de rotation du moteur. Pour mesurer cette vitesse nous avons conçu un module VHDL dont le schéma synoptique est donné dans la figure suivante :

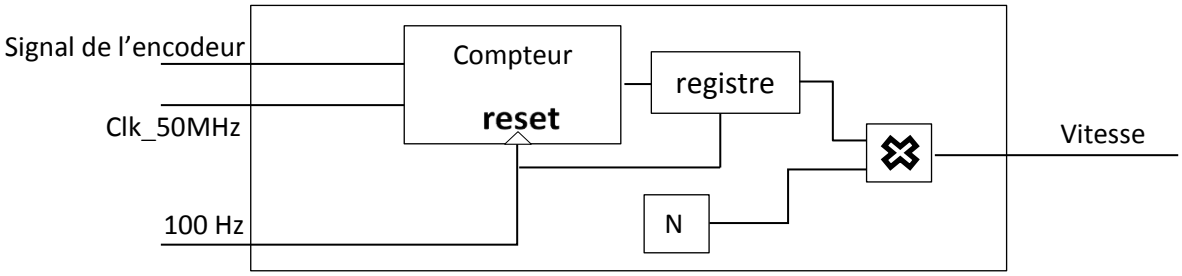


Fig.4.26 : Schéma synoptique du module de mesure de vitesse

Le facteur de multiplication N sert au calcul selon la relation suivante :

$$\frac{1\ impulsion}{T_e} \rightarrow N\ (tr/mn)$$

Où : T_e représente la période d'échantillonnage.

Dans notre cas, la période d'échantillonnage est égale à 10 ms. Autrement dit, si le compteur reçoit une impulsion chaque période d'échantillonnage (10 ms), la sortie du compteur sera multiplié par N (dans notre cas N=9,6 tr/m). La représentation schématique qui résume les entrées/sorties de ce module est donnée dans la figure suivante :

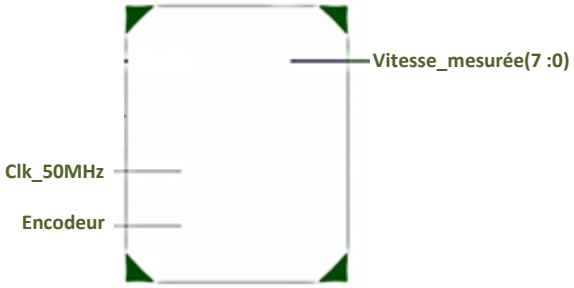


Fig. 4.27: Représentation schématique du module de mesure de vitesse

Le fonctionnement de ce module a été analysé par simulation et pratiquement en lui appliquant les signaux générés par l'encodeur lorsque le moteur est sous tension. Les trois cas qui correspondent à une vitesse de : 200 tr/mn, 96 tr/mn et 57 tr/mn ont été examinés. La valeur de la vitesse est récupérée en binaire en utilisant les LEDs de la carte de développement.

a. Simulation sous ISim

A chaque fréquence du signal de l'encodeur correspond une valeur bien déterminée de la vitesse qui sera calculée et donnée en sortie du module mesure de vitesse.

- **Pour 200 tr/mn**

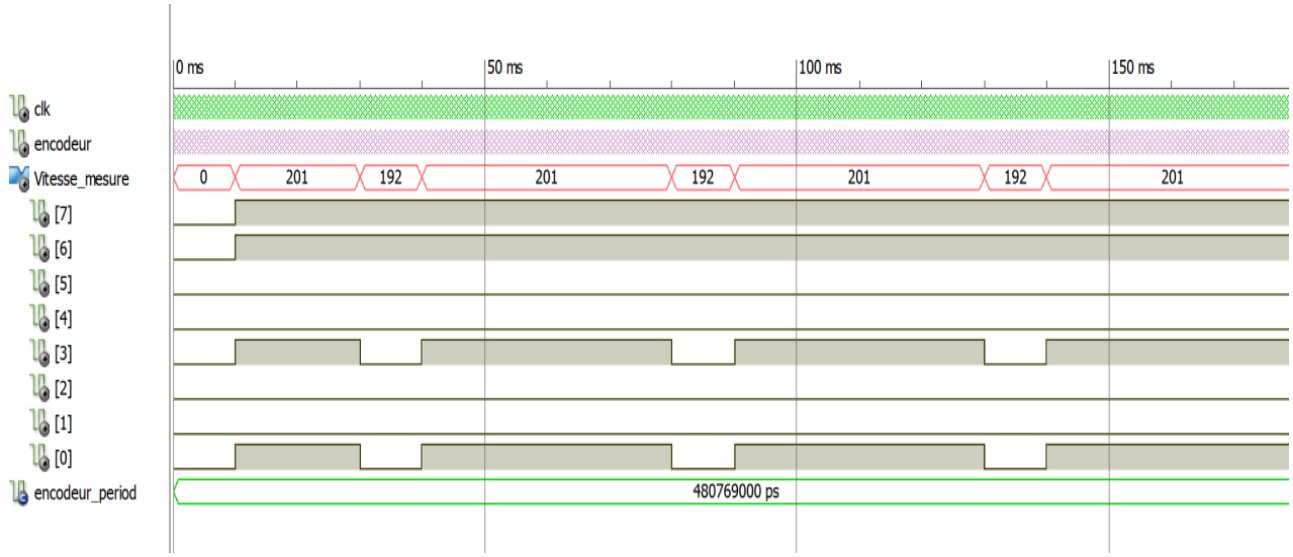


Fig. 4.28: Chronogrammes du signal provenant de l'encodeur et de la vitesse mesurée (200 tr/mn)

- **Pour 96 tr/mn**

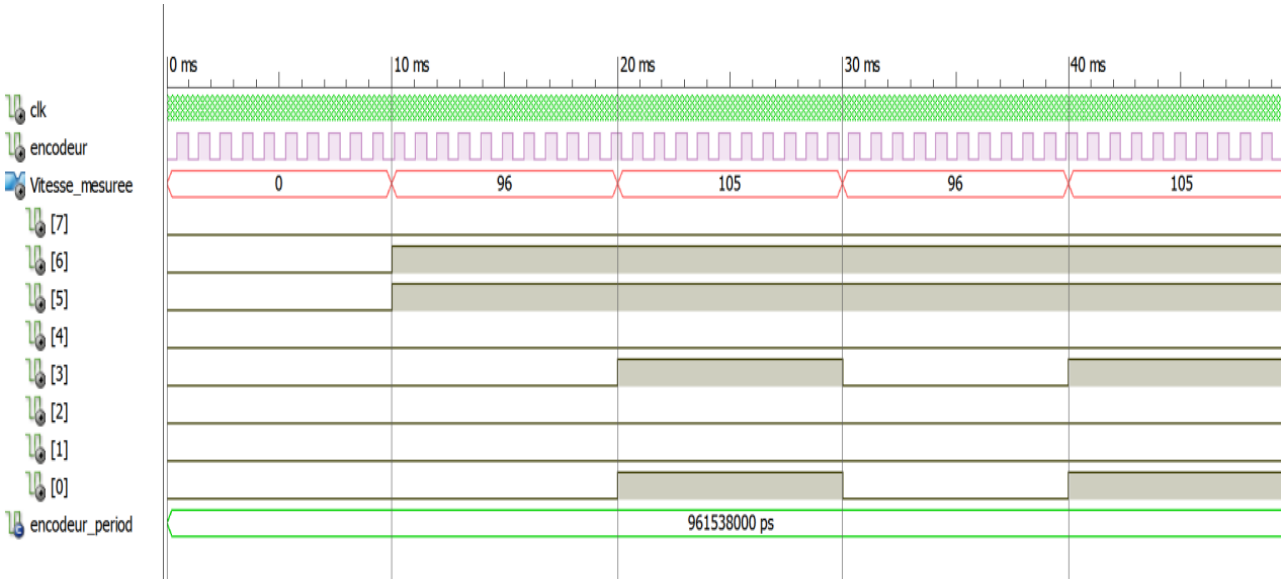


Fig. 4.29 : Chronogrammes du signal provenant de l'encodeur et de la vitesse mesurée (96 tr/mn)

• **Pour 57 tr/mn**

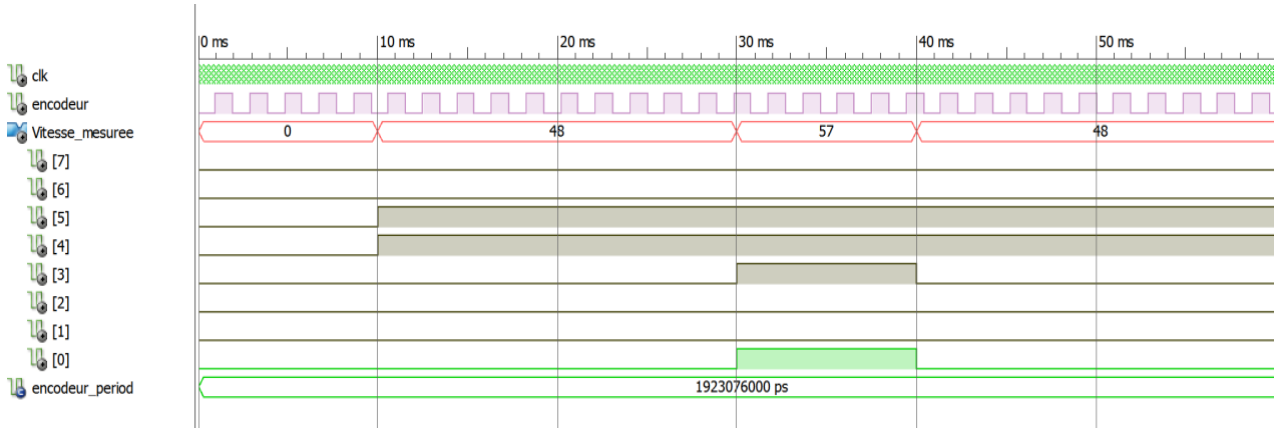


Fig. 4.30 : Chronogrammes du signal provenant de l'encodeur et de la vitesse mesurée (96 tr/mn)

b. Résultats pratiques

Pour vérifier le bon fonctionnement du module mesure de vitesse nous avons utilisé un tachymètre externe.

• **Pour 200 tr/mn**

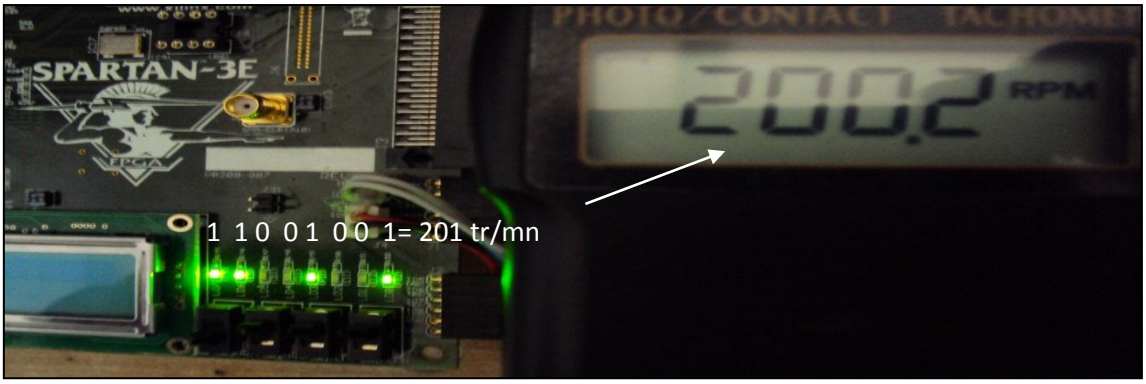


Fig. 4.31 : La vitesse mesurée (200 tr/mn)

• **Pour 96 tr/mn**

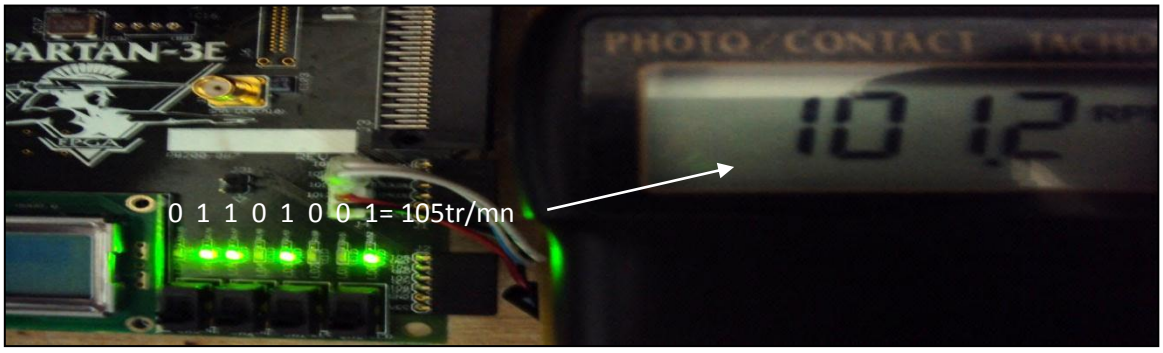


Fig. 4.32 : La vitesse mesurée (96 tr/mn)

• **Pour 57 tr/mn**

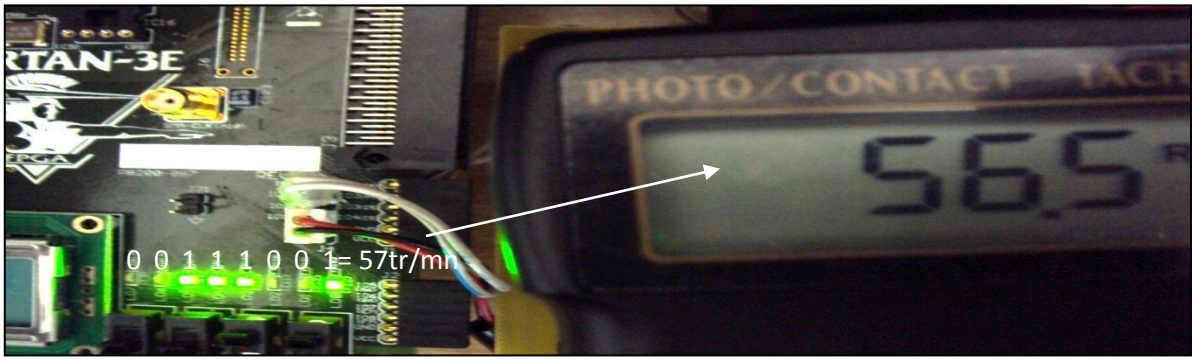


Fig. 4.33 : La vitesse mesurée (57 tr/mn)

Remarque :

Les résultats de simulation et pratiques montre qu'il y a un petit écart entre la vitesse mesurée et la vitesse réelle. Ceci, est dû à la mauvaise réception de quelques impulsions du signal de l'encodeur. Cet écart est très faible et la précision du module développé peut être jugé acceptable.

4.4. Le module PID

Le module PID développé est donné par sa représentation schématique suivante :

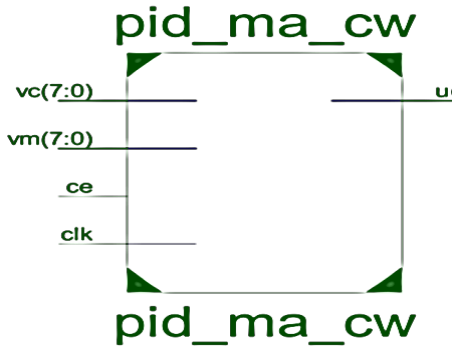


Fig. 4.34 : Représentation Schématique du module PID

Outre que le signal d'horloge clk, le module PID reçoit en entrée la consigne vc et la vitesse mesurée vm qui sont codés sur 8 bits. Le fonctionnement de ce module a été analysé par simulation en utilisant une entrée de consigne carrée d'amplitude 150 tr/min.

• **Résultats de simulation sous ISim**

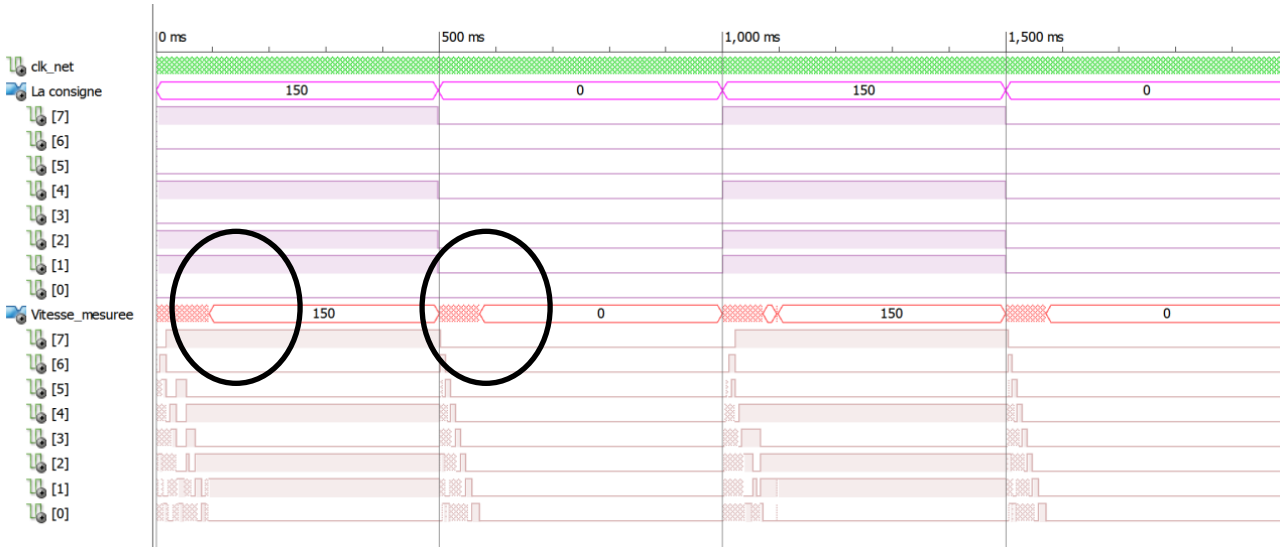


Fig. 4.35 : Chronogrammes de la consigne et de la vitesse mesurée

Pour obtenir l'ordre de grandeur du temps de réponse nous avons fait un zoom sur les parties encerclées en noir. Les résultats sont présentés dans les figures suivantes :

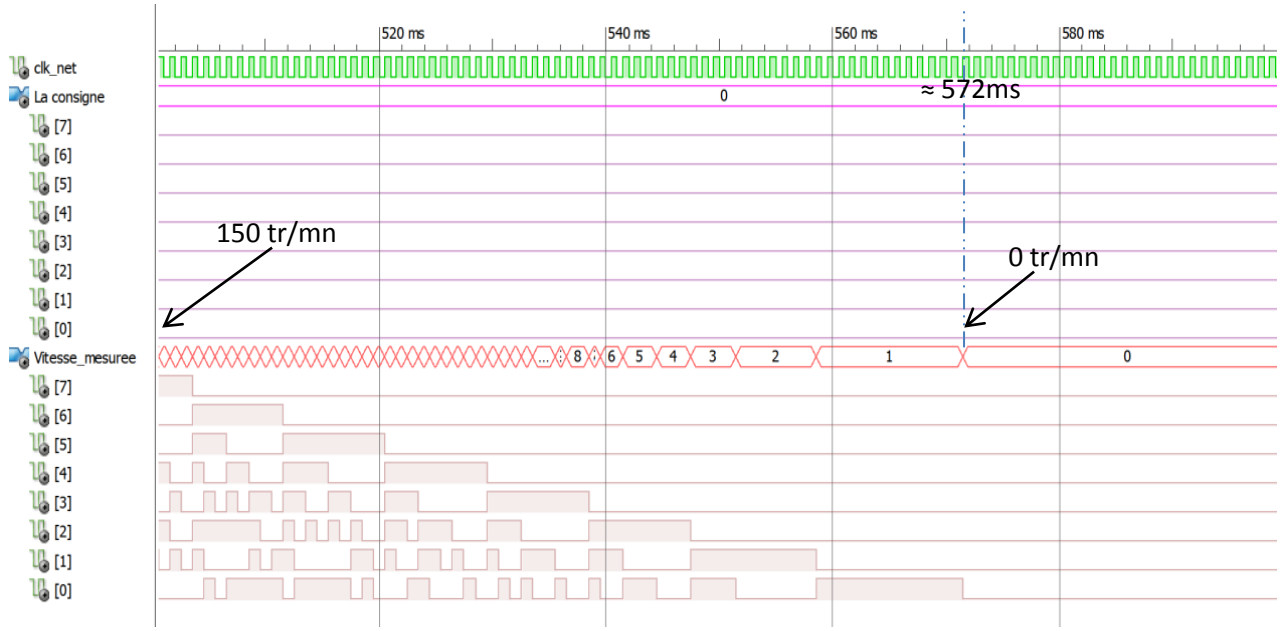


Fig. 4.36 : Zoom sur les chronogrammes (passage de 150 tr/mn vers 0 tr/mn)

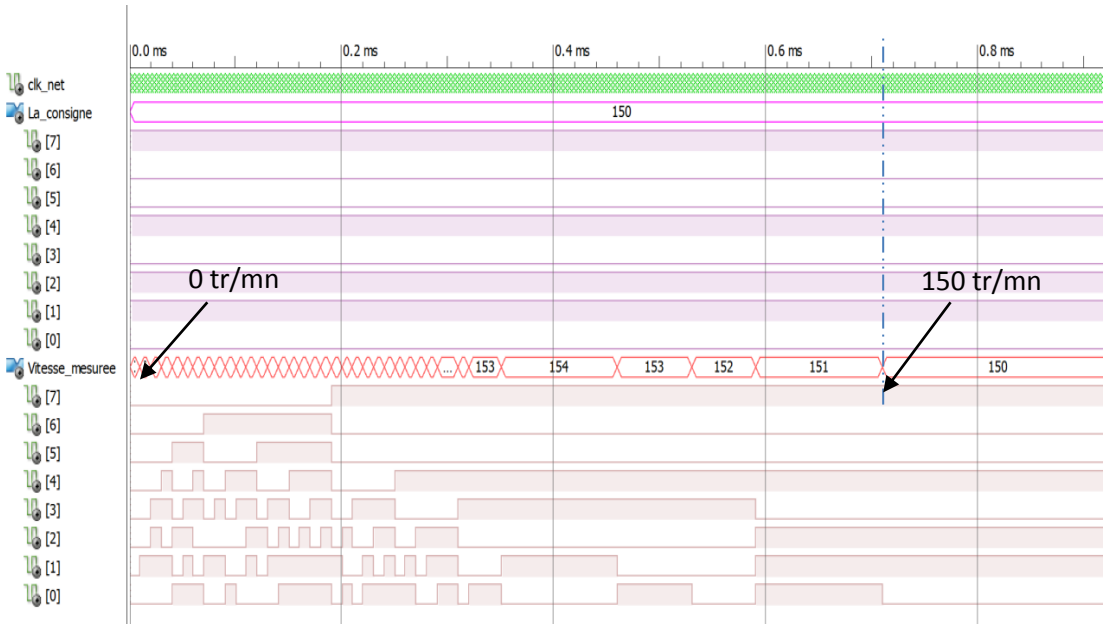


Fig. 4.37 : Zoom sur les chronogrammes (passage de 0 tr/mn vers 150 tr/mn)

Remarque :

Ces résultats montrent le bon fonctionnement du module PID développé.

5. Application à la commande du moteur

Dans ce paragraphe, après avoir testé le bon fonctionnement des modules élémentaires, le fonctionnement de l'ensemble des modules représentés par le schéma synoptique de la figure (4.2) a été soumis à des tests pratiques. La consigne est codée sur trois bits et elle est introduite à l'aide des trois commutateurs (SW2, SW1, SW3) de la carte de développement. La valeur binaire obtenue à l'aide des commutateurs est multipliée par une constante égale à 30 pour former la valeur de la cosigne en tr/mn. La vitesse de rotation du moteur est récupérée sous forme binaire à l'aide des LEDs de la carte et mesurée avec le tachymètre. Les figures suivantes montrent les résultats obtenus pour différentes valeurs de la consigne.

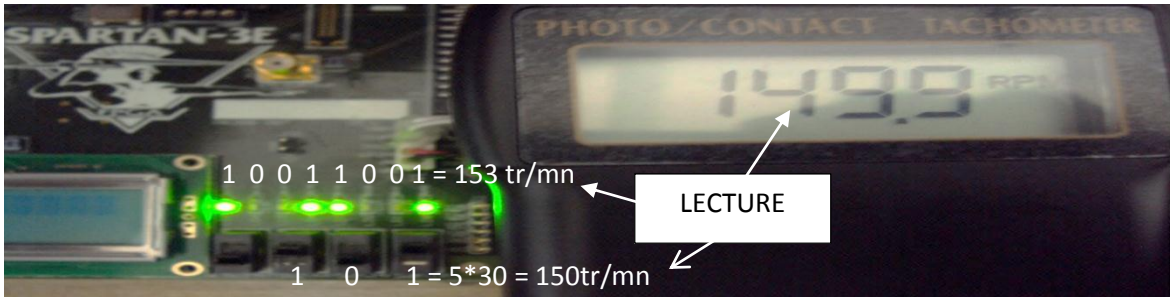


Fig. 4.38. : Résultat de la commande (consigne = 150 tr/mn)

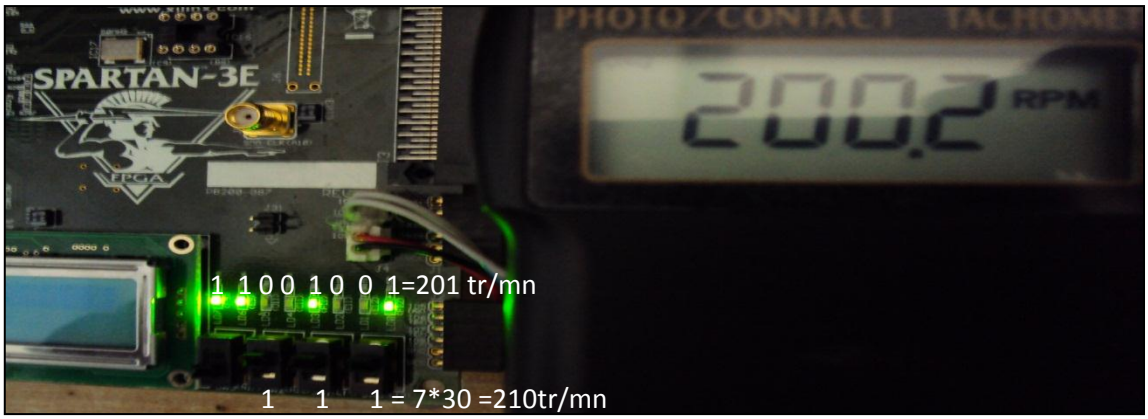


Fig. 4.39 : Résultat de la commande (consigne = 210 tr/mn)

Le tableau suivant donne un récapitulatif des résultats trouvés :

Disposition des commutateurs	Consigne (tr/mn)	Vitesse mesurée sur LEDs (tr/mn)	Vitesse mesurée sur Tachymètre
101 = 5	5*30=150	10011001 = 153	149.9
111= 7	7*30=210	11001001 = 201	200.2

Tab. 4.1 : Résumé des résultats pratique de la commande PID

6. Résumés des résultats de la synthèse

pfe_mama_bit_cw Project Status (06/16/2012 - 01:32:35)			
Project File:	pfe_mama_bit_cw.xise	**Parser Errors:**	No Errors
Module Name:	pfe_mama_bit_cw	**Implementation State:**	Programming File Generated
Target Device:	xc3s500e-4fg320	**Errors:**	No Errors
Product Version:	ISE 12.3	**Warnings:**	No Warnings
Design Goal:	Balanced	**Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**Timing Constraints:**	All Constraints Met
Environment:	System Settings	**Final Timing Score:**	0 (Timing Report)

Tab. 4.2: Rapport de synthèse du projet réalisé

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	197	9,312	2%	
Number used as Flip Flops	189			
Number used as Latches	8			
Number of 4 input LUTs	507	9,312	5%	
Number of occupied Slices	351	4,656	7%	
Number of Slices containing only related logic	351	351	100%	
Number of Slices containing unrelated logic	0	351	0%	
Total Number of 4 input LUTs	650	9,312	6%	
Number used as logic	507			
Number used as a route-thru	143			
Number of bonded IOBs	17	232	7%	
Number of BUFGMUXs	3	24	12%	
Number of MULT18X18SIOs	16	20	80%	
Average Fanout of Non-Clock Nets	1.53			

Performance Summary			[-]
Final Timing Score:	0 (Setup: 0, Hold: 0, Component Switching Limit: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	sam. 16. juin 01:22:02 2012	0	0	3 Infos (0 new)
Translation Report	Current	sam. 16. juin 01:32:00 2012	0	0	0
Map Report	Current	sam. 16. juin 01:32:15 2012	0	0	7 Infos (0 new)
Place and Route Report	Current	sam. 16. juin 01:32:30 2012	0	0	0
Power Report					
Post-PAR Static Timing Report	Current	sam. 16. juin 01:32:33 2012	0	0	4 Infos (0 new)
Bitgen Report	Current	sam. 16. juin 01:32:50 2012	0	0	0

Secondary Reports		[-]
Report Name	**Status**	**Generated**
Post-Synthesis Simulation Model Report	Current	sam. 16. juin 01:24:14 2012
Physical Synthesis Report	Out of Date	sam. 16. juin 01:32:14 2012
WebTalk Report	Current	sam. 16. juin 01:32:50 2012
WebTalk Log File	Current	sam. 16. juin 01:32:58 2012

Tab.4.3 : Résumé du taux d'utilisation des composants dans le système conçu

7. Conclusion

Après avoir décrit le banc d'essais et ses différentes parties, nous avons présenté les différents résultats, de simulation et de test pratique, pour chaque module. Les résultats de simulations présentés au cours de ce chapitre à l'aide de l'outil ISim ainsi que les résultats pratiques pour chaque module synthétisé, à savoir : Le module diviseur de fréquence, le générateur des signaux PWM et le régulateur PID implémentés sur le circuit FPGA Spartan-3E, montrent le bon fonctionnement du système implémenté. Un rapport de synthèse du projet implémenté a été donné en fin de ce chapitre. Une lecture rapide de ce rapport nous montre que le taux des LUTs et des multiplieurs dédiés utilisés est faible.

Implémentation

Et Essais pratiques

1. Introduction

La commande en vitesse du moteur à courant continu via la carte de développement Spartan-3E nécessite la mise en œuvre d'une carte d'interface et une carte de puissance. Différents circuits sont nécessaires pour la génération des signaux PWM et la mesure de vitesse.

Dans ce chapitre, après avoir donné une brève description des différentes parties du banc d'essais, nous envisageons la synthèse de chaque module sous l'environnement ISE, accompagnée d'un résumé sur le nombre d'éléments utilisés. Enfin, les résultats de simulation et ceux des essais pratiques sont présentés et commentés.

2. Constitution du banc d'essais

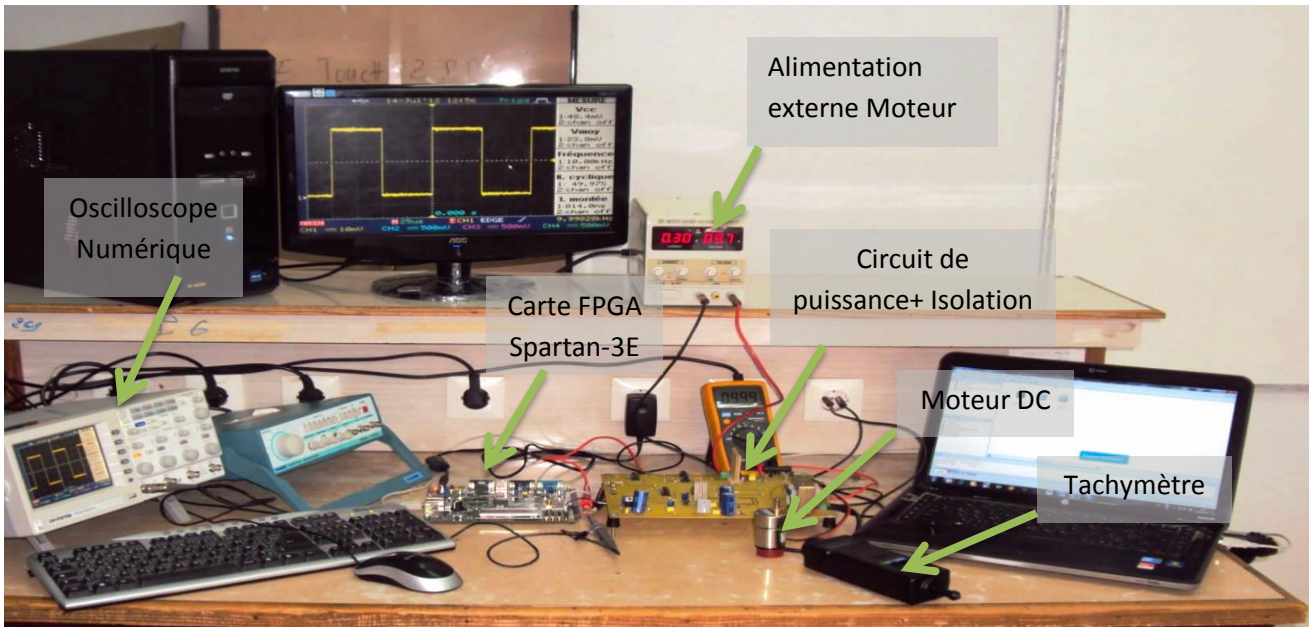


Fig.4.1 : Dispositif expérimental du banc d'essais

Le banc d'essai que nous avons réalisé est décrit par la figure (4.1) et est constitué des éléments suivant :

- La carte de développement Spartan-3E
- Un micro-ordinateur
- Un circuit d'isolation
- Un circuit de puissance
- Un Oscilloscope numérique
- Une alimentation externe pour le moteur
- Une alimentation stabilisée de 5V
- Un Tachymètre

Ce dispositif va nous permettre de synthétiser, simuler, tester et valider le système réalisé au cours de notre travail. Le système réalisé est illustré par son schéma synoptique suivant :

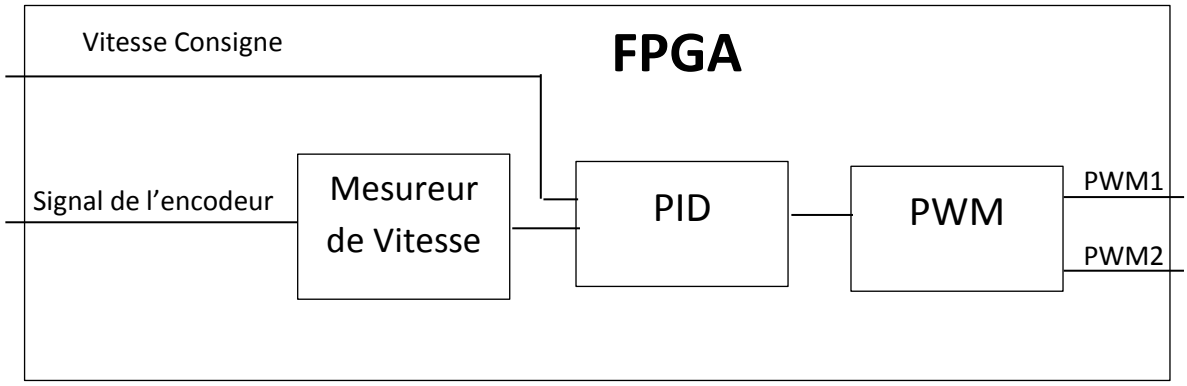


Fig. 4.2 : Schéma synoptique du système réalisé

3. Description des éléments du banc d'essais

3.1.La carte de développement FPGA SPARTAN-3E

La carte FPGA Spartan-3E, décrite dans le chapitre précédent, constitue le circuit numérique de commande. Nous allons décrire les modules utilisés lors de nos expériences.

a. Le circuit FPGA XC3S500E

Le XC3S500E possède cinq éléments fonctionnels fondamentaux :

- Des CLBs
- Des IOBs
- Des blocs RAM
- Des blocs multiplieurs
- Des DCMs

b. Les périphériques

- **Port USB**

La carte FPGA Spartan-3E contient un port USB standard de type B. Ce port est dans la partie gauche de la carte juste à côté du port Ethernet. Une fois le câble est bien installé et la carte est connectée au PC d'une manière correct, une LED verte s'allume, indiquant le bon fonctionnement.

- **Sources d'horloge**

La carte FPGA Spartan-3E possède une horloge de de 50MHz, avec un rapport cyclique compris entre 40% et 60%. Pour introduire une horloge externe, le signal d'entrée doit être connecté au connecteur SMA.

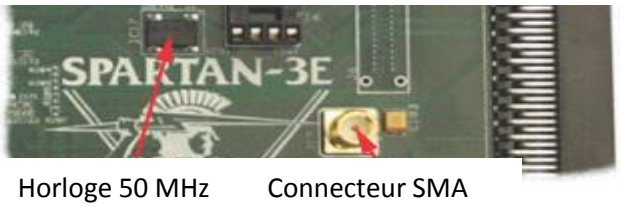


Fig.4.3 : Les horloges de la carte SPARTAN-3E

- **Les connecteurs d'expansion (broches d'Entrées/Sorties)**

La carte FPGA Spartan-3E offre une variété de connecteurs d'expansion pour une interface facile et flexible avec les circuits externes. La carte possède les expansions d'E/S suivants :

- Un connecteur Hirose possédant 100 broches avec avec 43 E/S utilisable à usage général concepteur.
- Trois modules périphériques de 6 broches.
- Un autre connecteur sans extensions physiques sur la carte.

Nous nous sommes servis des modules périphériques de 6 broches J1, J2 et J4. L'exemple des connexions entre le module J1 et le FPGA.

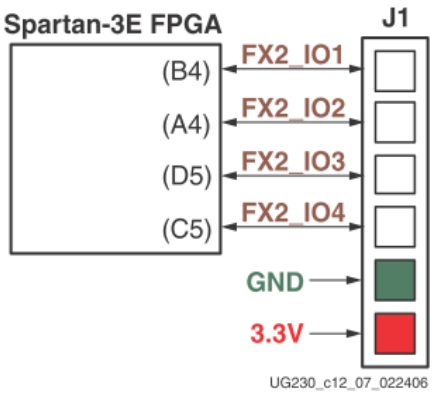


Fig.4.4 : Les connections entre le FPGA et le module périphérique J1

Remarque :

Ces modules servent comme entrées /sorties (IO1, IO2,…). Chaque module a 4 broches entrée/sortie, un GND et un Vcc de 3.3V.

• Les commutateurs et le bouton poussoirs

La carte FPGA Spartan-3E possède quatre commutateurs localisés dans le côté droite de la carte. Ces commutateurs sont étiquetés SW3 à l'extrême gauche jusqu'à SW0 à l'extrême droite comme le montre la figure (4.3). Ils vont nous permettre d'introduire la consigne codée en binaire. Un commutateur dans la position HIGH introduit 3.3V (un '1' logique). Par contre un commutateur en position LOW introduit un '0' logique.

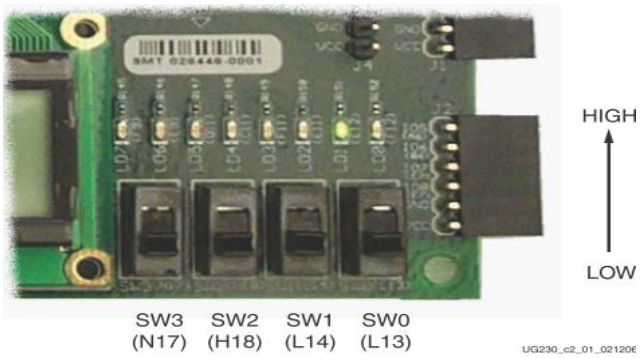


Fig.4 .5 : Les commutateurs

La carte possède aussi quatre boutons poussoirs sui servent comme entrées. Un exemple de leur utilisation est l'introduction d'un signal 'reset' pour la remise à zéro d'un processus.

• Les LEDs

La carte FPGA Spartan-3E a huit LEDs situés au-dessus des commutateurs comme le montre la figure (4.4). Les LEDs sont etiquetés LED7 jusqu'à LED0. Ces LEDs servent à la lecture en binaire ou en hexadécimal des résultats obtenus lors des expériences.

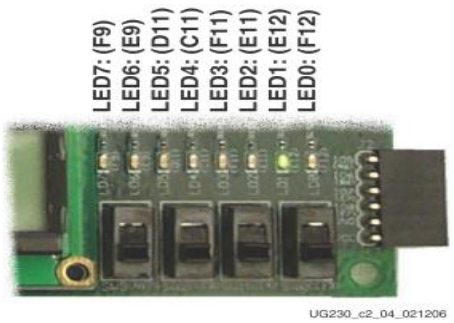


Fig.4.6 : Les LEDs

3.2. La carte de puissance

a. Circuit de puissance

Notre circuit de puissance est à base du circuit intégré L6203, ce dernier est un hacheur à quatre quadrants (annexe B1), alimenté avec une tension pouvant atteindre les 48V et un courant de 5A. Notre moteur peut absorber un courant maximal de 2A, donc pour garantir le fonctionnement attendu du moteur, le courant demandé par ce dernier doit être disponible à tout moment, plus particulièrement si nous devons introduire des charge assez importants.

La fréquence de commutation utilisée dans notre projet est de l'ordre de 10 KHz. Le schéma électrique du circuit de puissance réalisé est illustré par la figure suivante :

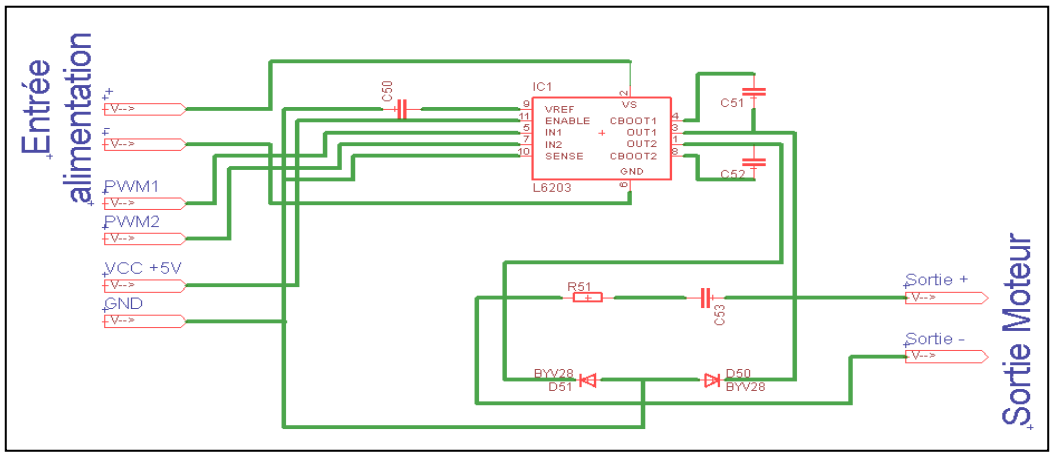


Fig.4.7: Schéma électrique de la carte de puissance

b. Circuit d'isolation

Vu que le FPGA est très sensible aux perturbations extérieures (par exemple une pin qui est mise par erreur à la masse), un circuit d'isolation entre la carte de développement et la carte de puissance est utilisé. Notre choix s'est porté sur des opto-coupleurs de la série 6N137 (annexe B2). La figure 4.10 illustre le schéma électrique utilisé.

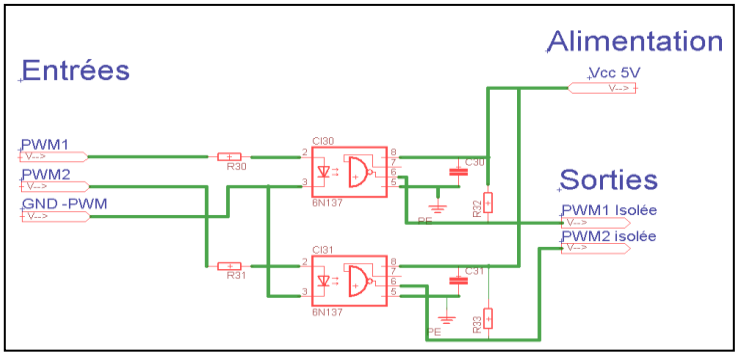


Fig. 4.8 : Schéma électrique du circuit d'isolation

c. Alimentation 3.3V

Pour pouvoir alimenter le circuit intégré HEF4050B, une alimentation de 3.3V à partir du régulateur de tension LM317 (annexe A2) est nécessaire et représenté par le schéma électrique suivant :

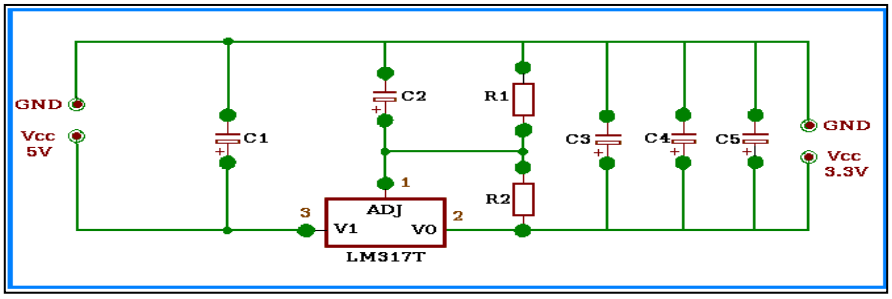


Fig. 4.9 : Schéma électrique de l'alimentation 3.3V

d. L'alimentation 5V

La plupart des circuits électriques utilisés sont à base de circuits intégrés qui fonctionnent avec une tension de +5V. Pour cette raison, un circuit est conçu pour avoir cette tension à partir du secteur. Son schéma électrique est le suivant.

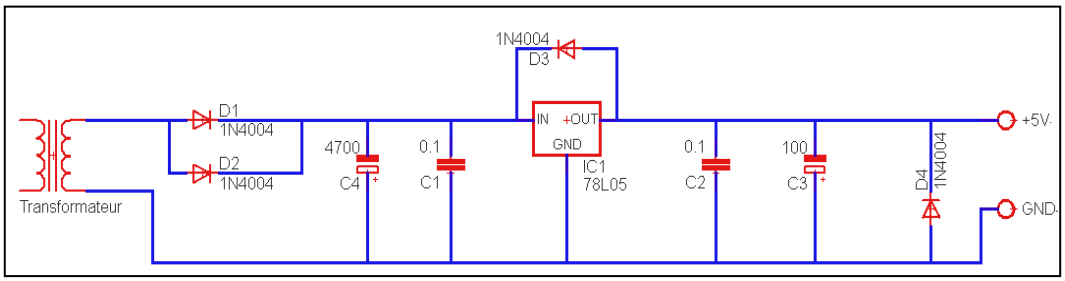


Fig.4.10 : Schéma électrique du circuit d'alimentation stabilisée de + 5V

Le circuit de puissance, le circuit d'isolation et le l'alimentation de 5V sont implantés sur la même carte représentée par la figure suivante :

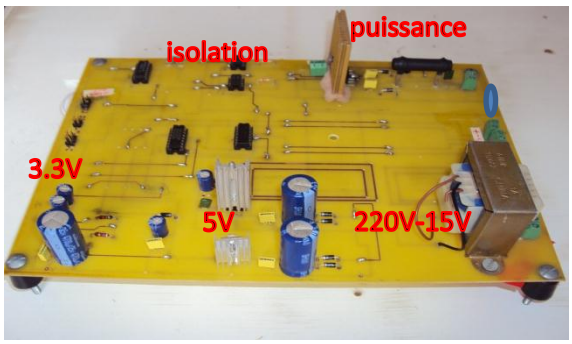


Fig. 4.11 : Photo de la carte de puissance

3.3.Le Tachymètre

Le tachymètre utilisé travail selon deux modes de mesures : mesure optique et mesure par contact. Il va nous permettre de valider les résultats de mesures obtenus par le module de mesure synthétisé.

4. Résultats de simulations et Essais pratiques

Afin de pouvoir tester le bon fonctionnement du processus commandé nous sommes passés par la synthèse de plusieurs modules en utilisant le langage VHDL. Le fonctionnement de chaque module a été simulé sous l'environnement ISE en utilisant ISim, puis validé par des essais pratiques.

4.1.Diviseur d'horloge

Comme c'est mentionné au paragraphe précédent, la carte Spratan-3E possède une horloge de 50 MHz et un connecteur qui permet d'introduire une horloge externe. Comme nous avons besoins d'une horloge de 10MHz et une autre de 100Hz nous avons pensé à concevoir deux diviseur d'horloge, un pour diviser l'horloge interne et l'autre pour diviser l'horloge externe. Le schéma synoptique de la figure suivante résume le principe de ce diviseur :

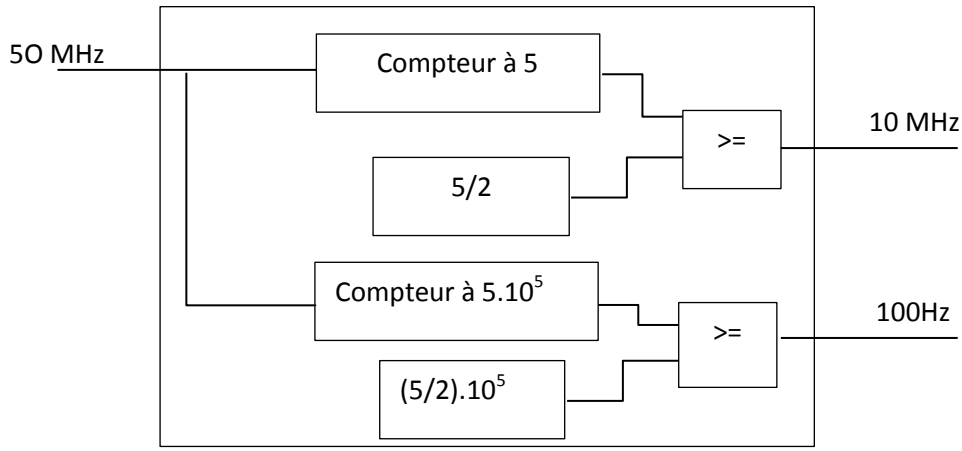


Fig.4.12: Schéma synoptique du diviseur d'horloge

La figure suivante présente le schématique du diviseur conçu :

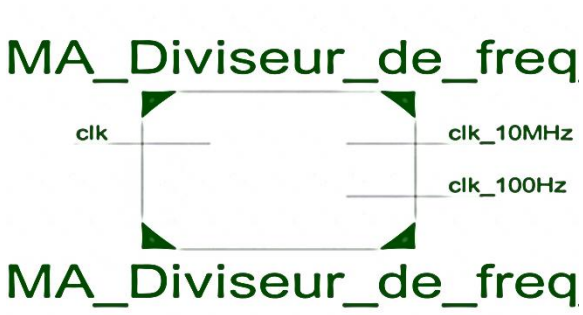
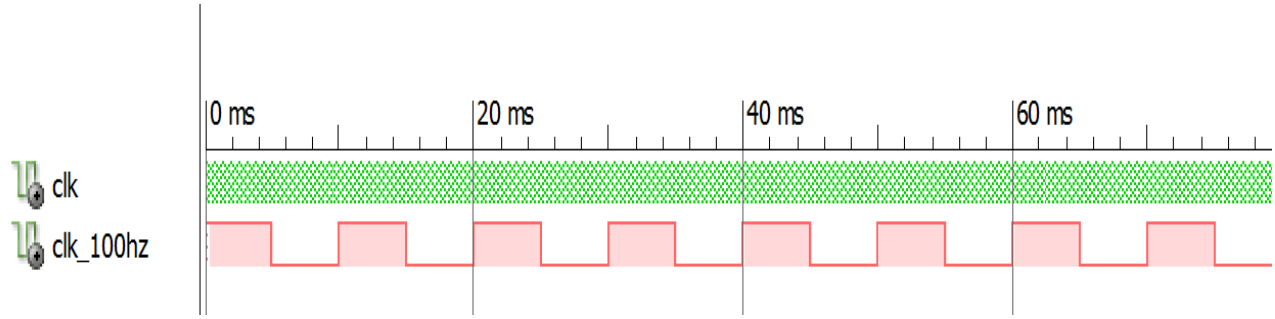


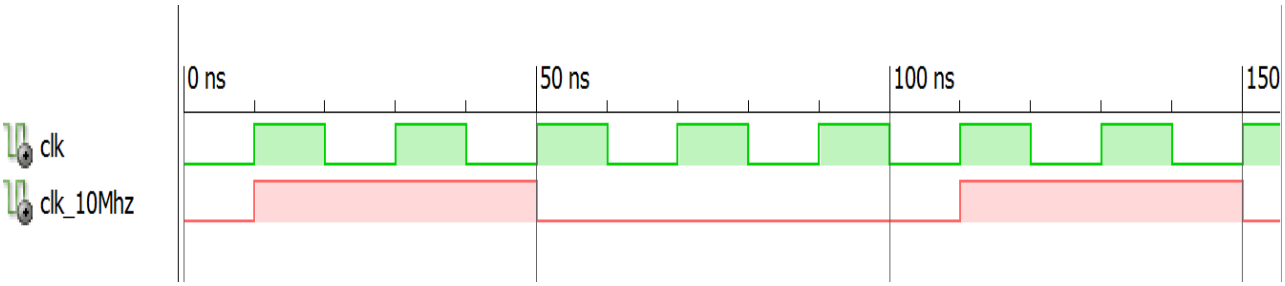
Fig. 4.13 : Schématique du diviseur de fréquence

a. Simulation sous ISim

- **Division de l'horloge interne**



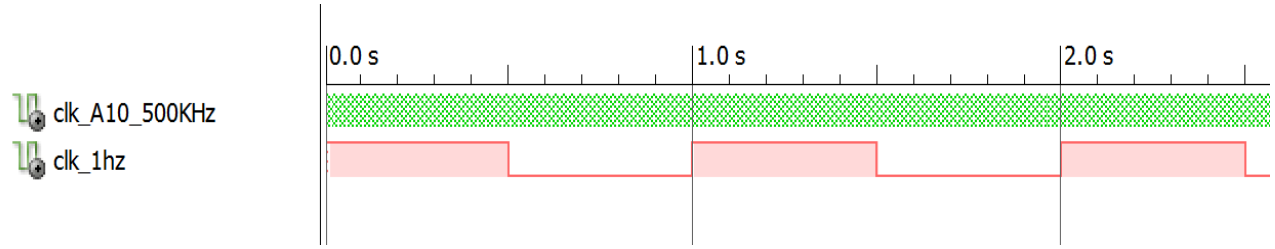
(a) Etablissement d'une horloge de 100 Hz



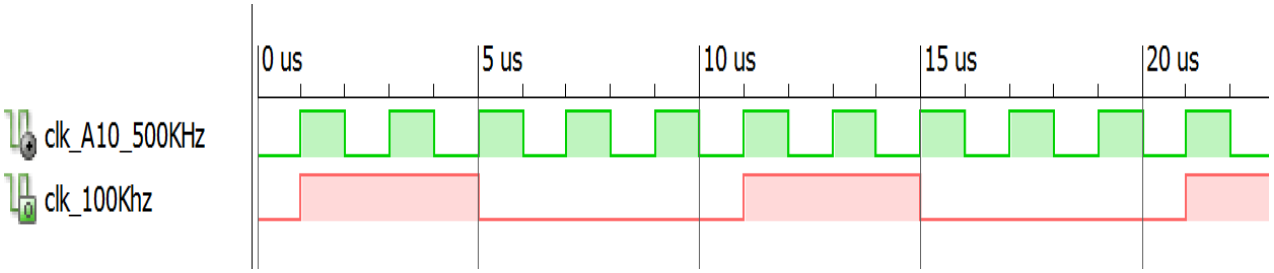
(b) Etablissement d'une horloge de 10 MHz

Fig. 4. 14: Résultats de la division de l'horloge interne

- **Division de l'horloge externe**



(a) Etablissement d'une horloge de 1 Hz



(b) Etablissement d'une horloge de 100 KHz

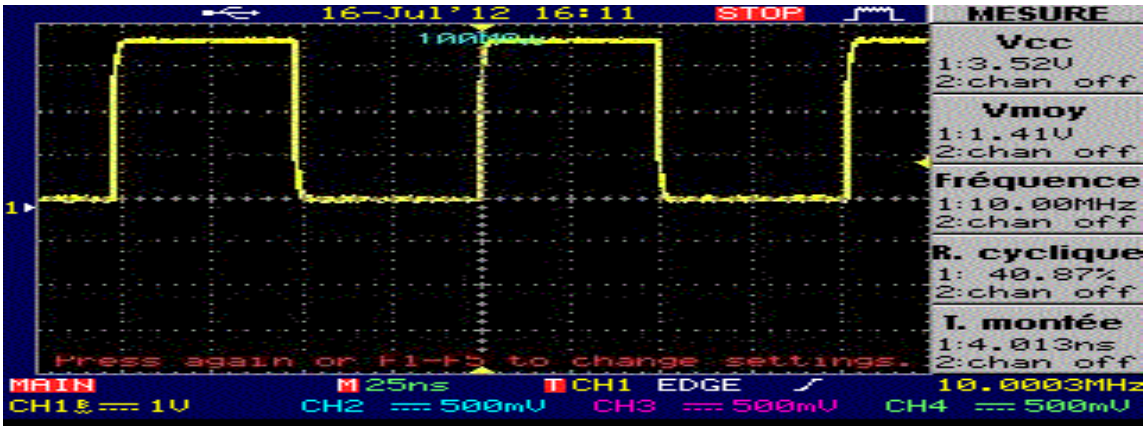
Fig. 4.15 : Résultats de la division de l'horloge externe

b. Pratique

- **Division de l'horloge interne**

(a) Etablissement d'une horloge de 1 Hz

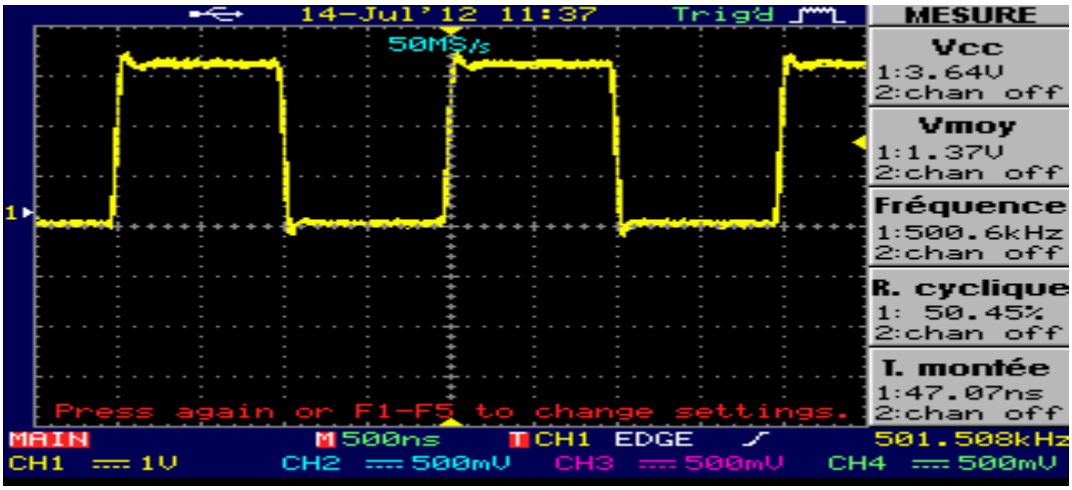
(b) Etablissement d'une horloge de 100 Hz



(c) Etablissement d'une horloge de 10 MHz

Fig. 4.16 : Résultats pratiques de la division de l'horloge interne

- **Division de l'horloge externe**



(a) Signal de l'horloge externe en jeu

(b) Etablissement d'une horloge de 1 Hz

(c) Etablissement d'une horloge de 100 KHz

Fig. 4.17 : Résultats pratiques de la division de l'horloge externe

Remarque :

En analysant le fonctionnement des deux modules de division d'horloges, lors de la simulation et des essais pratiques, on voit bien une analogie presque parfaite. Donc, on s'assure du bon fonctionnement de ces deux modules.

Ces deux modules vont établir les horloges pour les autres modules (générateur de PWM et Module de mesure de la vitesse).

4.2.Générateur des signaux PWM

Le PWM est la grandeur qu'à travers le correcteur peut agir sur le processus. D'où son importance capitale dans le système de commande. Pour générer ce signal nous avons synthétisé un module générateur de PWM basé sur l'architecture suivante :

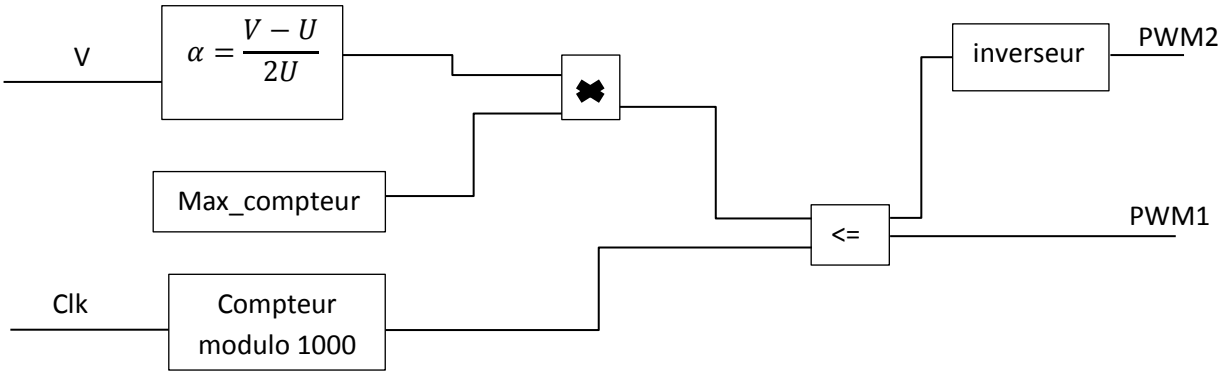


Fig.4.18: Schéma synoptique du générateur des signaux PWM

La représentation schématique du module est donnée par la figure suivante

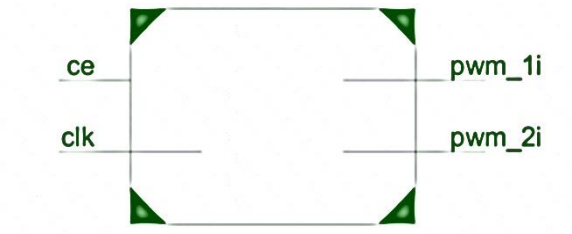


Fig. 4.19 : Représentation schématique du générateur des signaux PWM

a. Simulation sous ISim

Les résultats de la simulation du fonctionnement du générateur des signaux PWM qui a été effectuée pour trois valeur différentes du rapport cyclique, α=50%, α=75% et α=95% sont :

- **Pour α=50%**

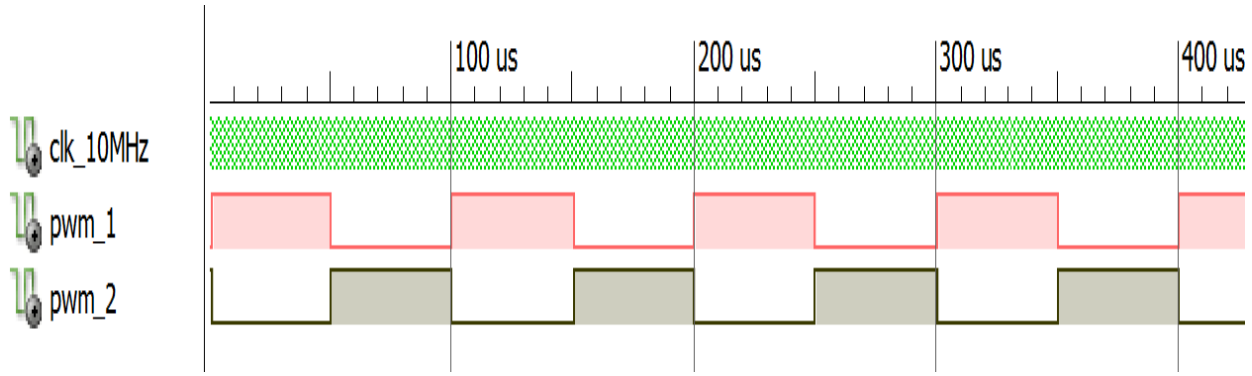


Fig. 4.20: Chronogrammes des signaux PWM1 et PWM2 pour α=50%

- **Pour α=75%**

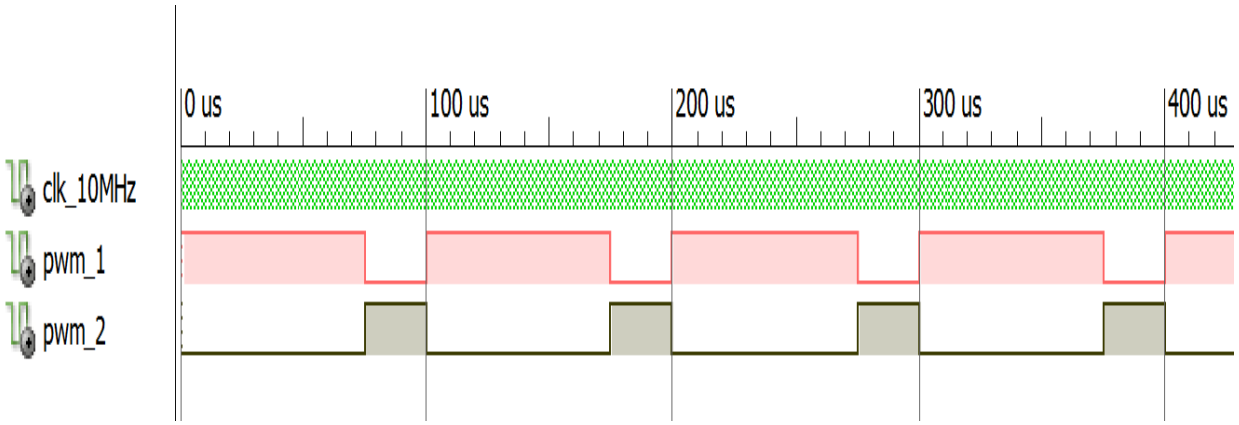


Fig. 4.21: Chronogrammes des signaux PWM1 et PWM2 pour α=75%

- **α=95%**

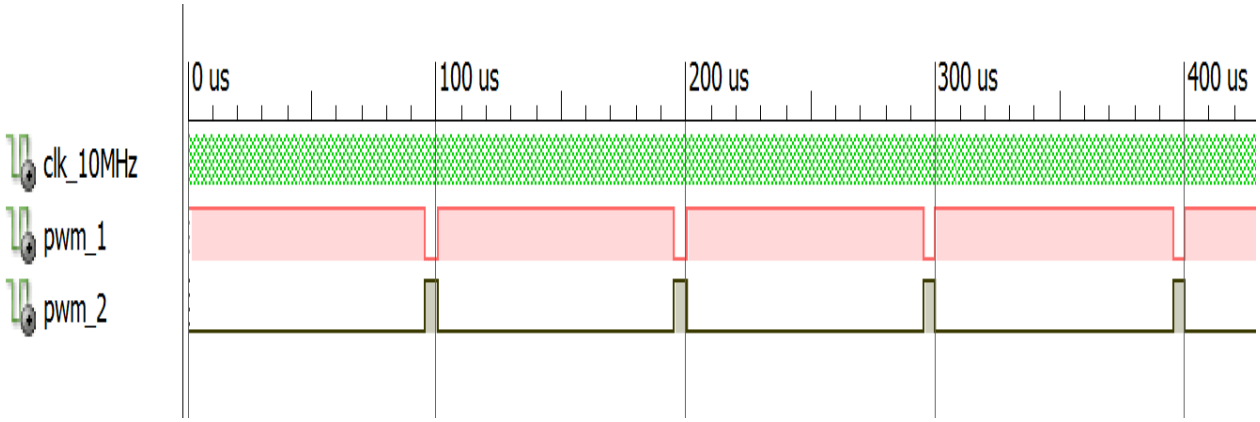


Fig. 4.22 : Chronogrammes des signaux PWM1 et PWM2 pour α=95%

b. Pratique

Pour s'assurer du bon fonctionnement du module, nous avons effectués quelques essais pour différents rapports cycliques. Les mesures ont été prises par un oscilloscope numérique.

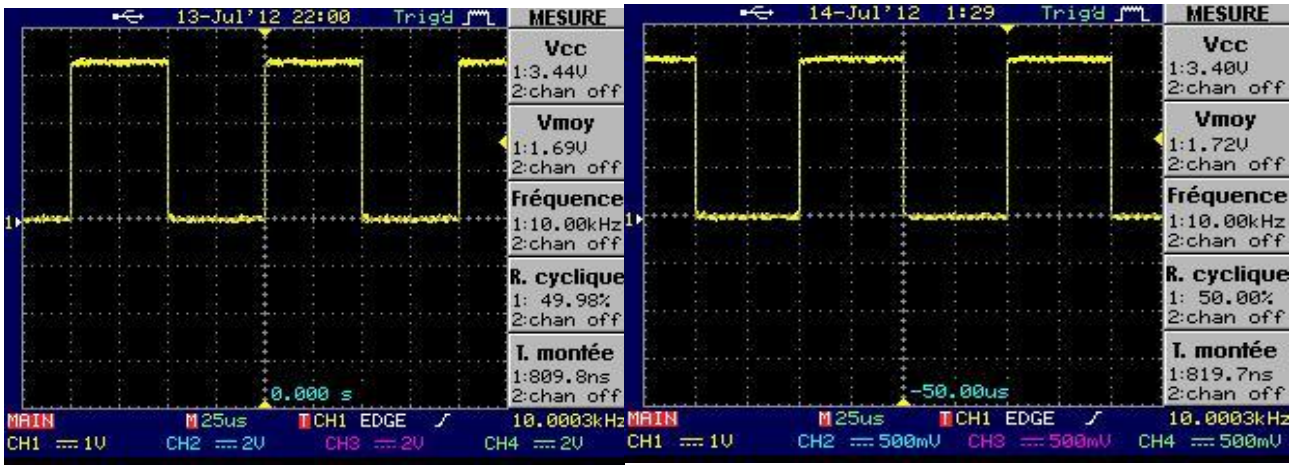


Fig.4.23 : Les signaux PWM1 et PWM2 pour un rapport cyclique α=50%

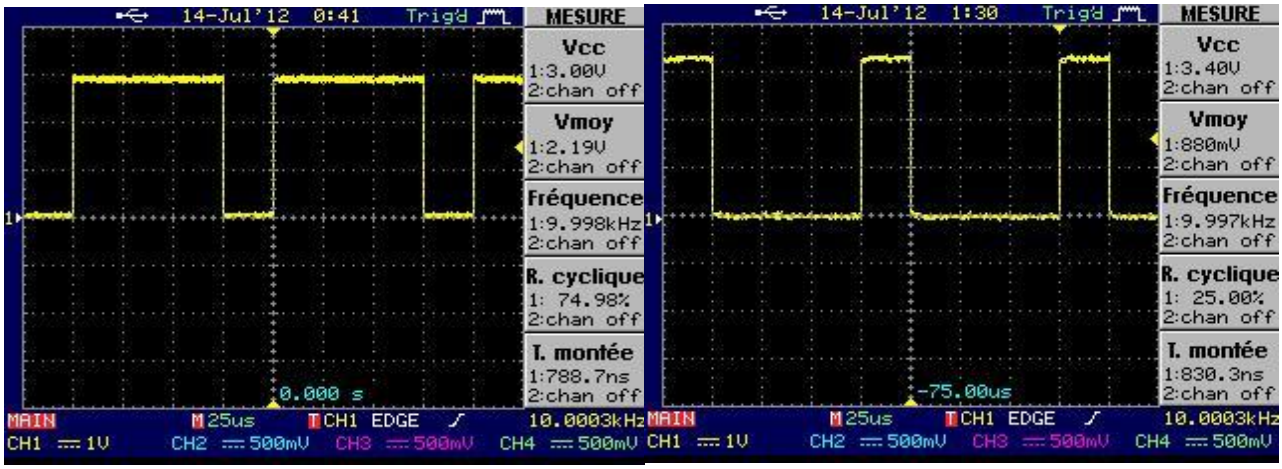


Fig.4.24: Les signaux PWM1 et PWM2 pour un rapport cyclique α=75%

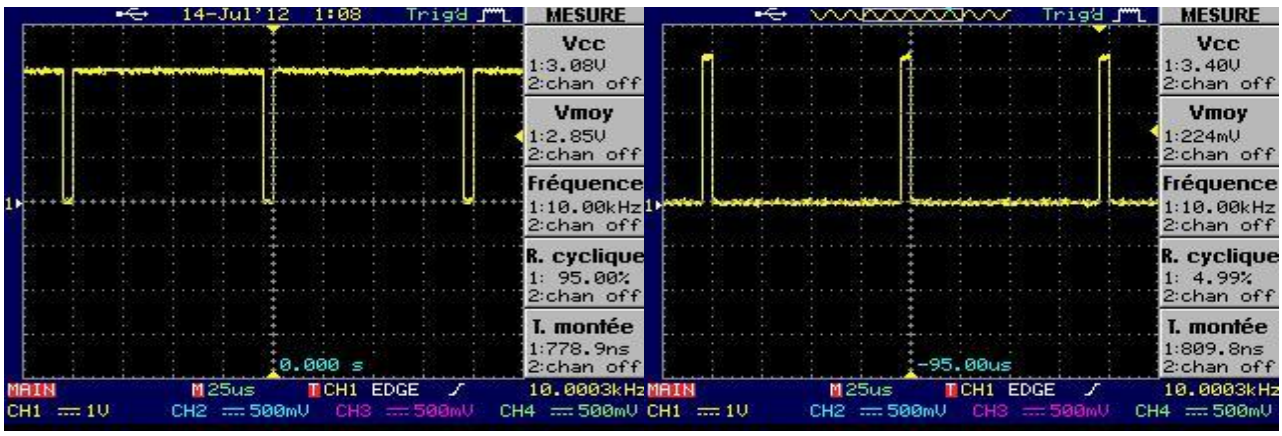


Fig.4.25 : Les signaux PWM1 et PWM2 pour un rapport cyclique α=95%

Remarque :

Chaque PWM est accompagné par son complément. Prenez le dernier essai, PWM1 a un rapport cyclique α=95% et PWM2 a un rapport cyclique de α=5%.

Les résultats de simulation et ceux de la pratique prouvent le bon fonctionnement du module 'générateur des signaux PWM' synthétisé.

Les résultats des essais pratiques sur le module de génération des signaux PWM sont démontrent son bon fonctionnement.

4.3.Mesureur de vitesse

Avoir l'image réelle de la vitesse de rotation du moteur, est une tâche très essentielle pour la commande en vitesse du moteur à courant continu. Ce dernier possède un encodeur qui donne un certain nombre d'impulsions par minutes. La vitesse peut être calculée en se basant sur le nombre de ces impulsions. Afin de pouvoir récupérer avoir son image réel à chaque impulsion, Nous avons conçu un module servant au calcul la vitesse. Le schéma synoptique de ce module est donné dans la figure suivante :

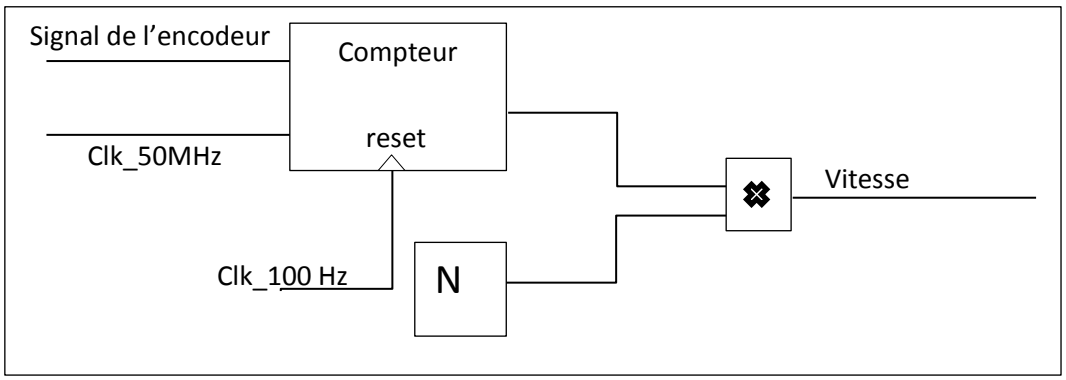


Fig.4.26: Schéma synoptique du module servant au calcul de la vitesse

Le facteur de multiplication N sert au calcul selon la relation suivante :

$$\frac{1\ impulsion}{T_e} \rightarrow N \quad \text{(tr/mn)}$$

Où :

T_e: Représente la période d'échantillonnage.

Dans notre cas le temps d'échantillonnage est égal à 10 ms. Autrement dit si le compteur arrive reçoit une impulsion chaque période d'échantillonnage (10 ms), le facteur N la sortie du compteur sera multiplié par N (dans notre cas N=9,6).La représentation schématique qui résume les entrées/sorties de ce module est donnée dans la figure suivante :

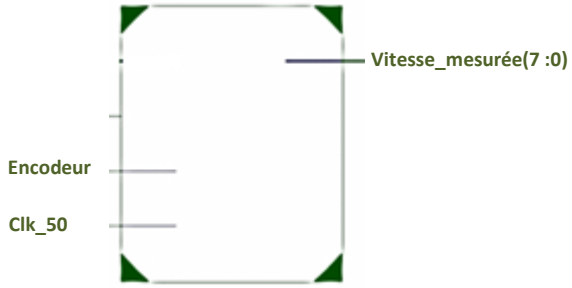


Fig. 4.27 : Représentation schématique du mesureur de vitesse

Pour démontrer son bon fonctionnement on l'expose à des tests de simulation et des essais pratiques. En introduisant une tension aux bornes du moteur, le moteur tourne, l'encodeur envoie un signal qui va être récupéré par le mesureur de vitesse, la vitesse mesurée sera calculée et donnée en sortie codée en binaire sur huit bit.

a. Simulation sous ISim

Pour chaque fréquence du signal de l'encodeur une vitesse bien déterminée sera calculée et donnée en sortie du mesureur de vitesse. Pour simuler son fonctionnement on a choisi d'avoir trois cas différents : 200 tr/mn, 96 tr/mn et 58 tr/mn.

- **Pour 200 tr/mn**

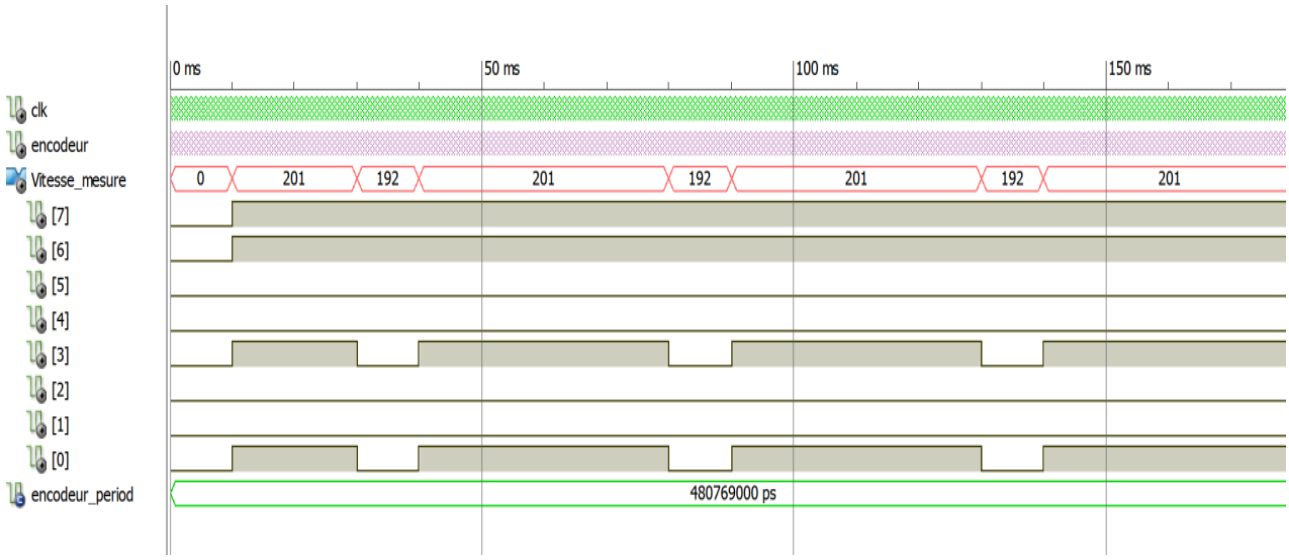


Fig. 4.28: Chronogrammes du signal provenant de l'encodeur et de la vitesse mesurée (200 tr/mn)

Pour avoir une vitesse de 200 tr/mn le mesureur doit recevoir une impulsion chaque période de 480769000 ps.

• **Pour 96 tr/mn**

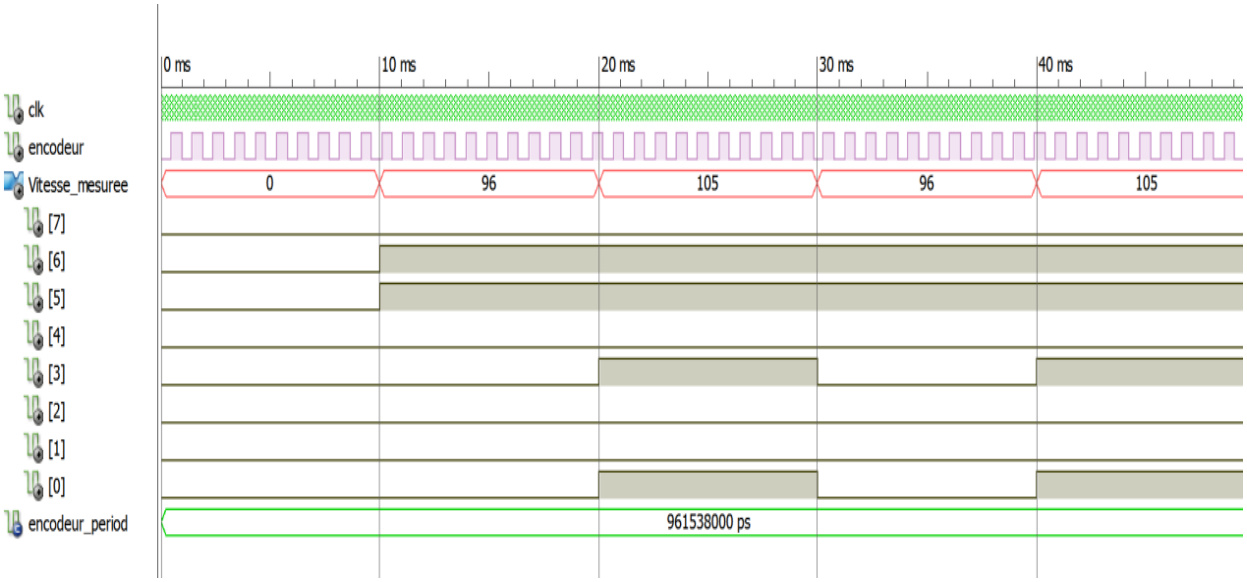


Fig. 4.29 : Chronogrammes du signal provenant de l'encodeur et de la vitesse mesurée (96 tr/mn)

Pour avoir une vitesse de 96 tr/mn le mesureur doit recevoir une impulsion chaque période de 961538000 ps.

• **Pour 57 tr/mn**

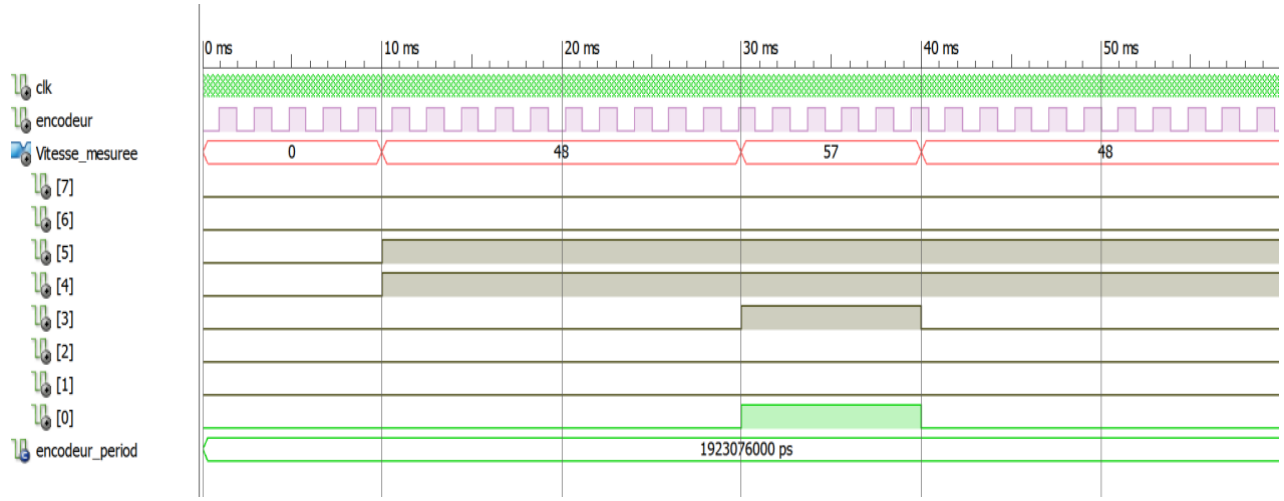


Fig. 4.30 : Chronogrammes du signal provenant de l'encodeur et de la vitesse mesurée (57 tr/mn)

Pour avoir une vitesse de 57 tr/mn le mesureur doit recevoir une impulsion chaque période de 1923076000 ps.

b. Résultats pratiques

• Pour 200 tr/mn

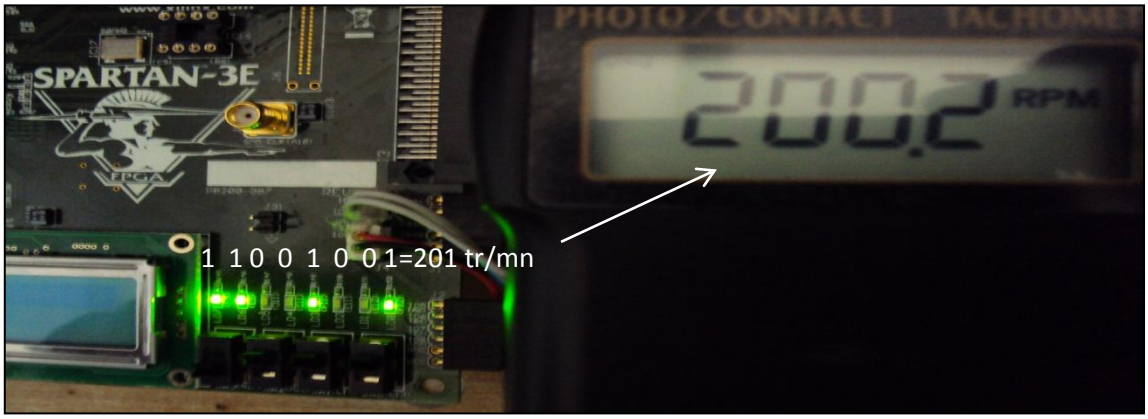


Fig. 4.31: La vitesse mesurée (200 tr/mn)

• Pour 96 tr/mn

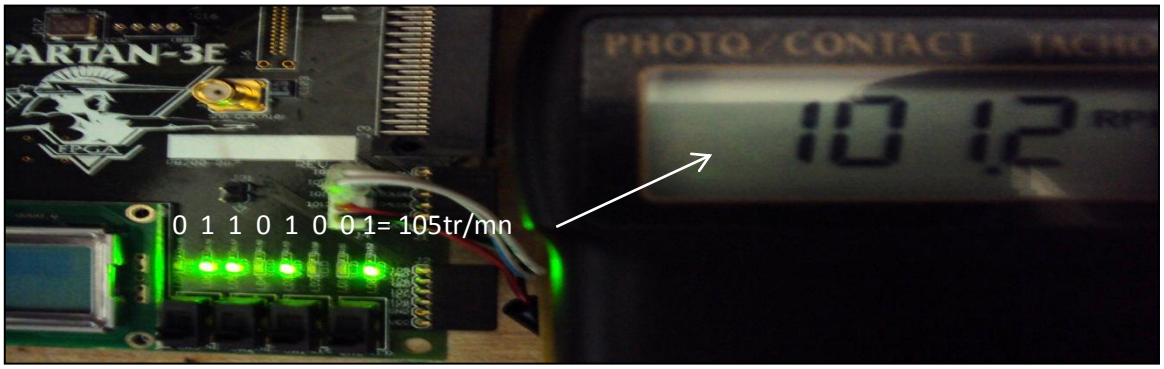


Fig. 4.32 : La vitesse mesurée (96 tr/mn)

• Pour 57 tr/mn

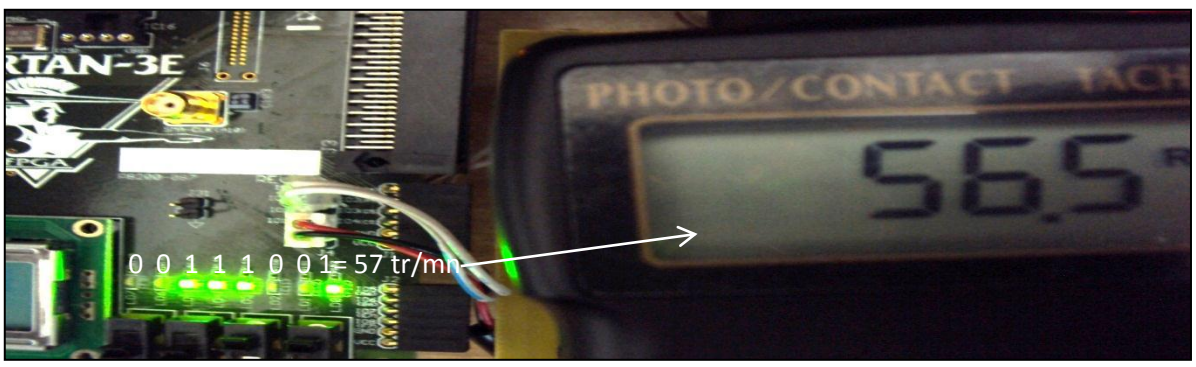


Fig. 4.33 : La vitesse mesurée (57 tr/mn)

Remarque :

Dans la simulation ainsi que les essais pratiques il existe un petit décalage dans les mesures du à la mauvaise réception de quelques impulsion. Le choix de la période d'échantillonnage est à l'origine de cette faible erreur. Le module de mesure de vitesse peut être jugé fonctionnel à une erreur près.

4.4. Le module PID

Le module PID synthétisé est représenté par son schématique suivant :

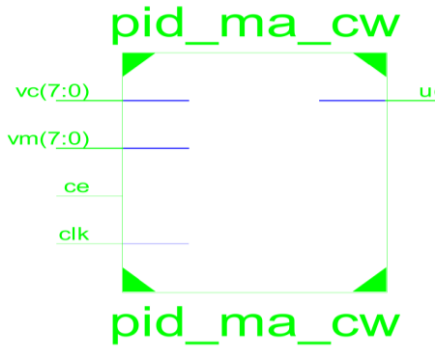


Fig. 4.34 : Représentation Schématique du module PID

Outre que le signal d'horloge clk, il a en entrée la consigne vc et la vitesse mesurée vm codés sur 8 bits. Le fonctionnement de ce module est simulé pour une consigne carrée (alternative de 0 ; 150).

- **Simulation sous ISim**

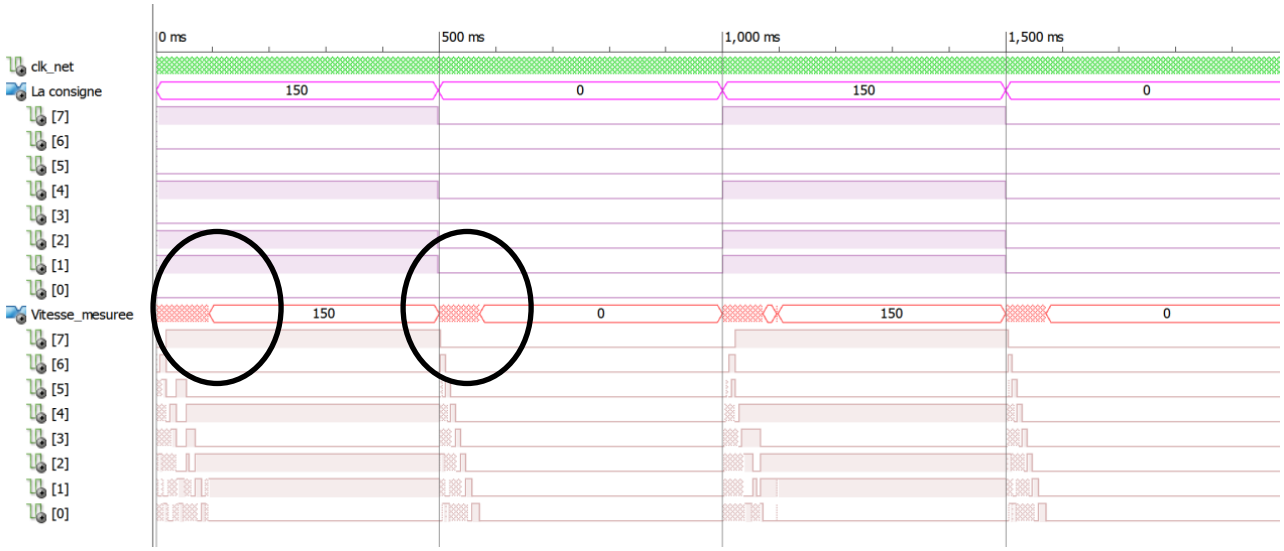


Fig. 4.35: Chronogrammes de la consigne et de la vitesse mesurée

Pour avoir une idée sur le temps de réponse nous avons fait un zoom sur les parties encerclées en noir. Les résultats sont présentés dans ces figures :

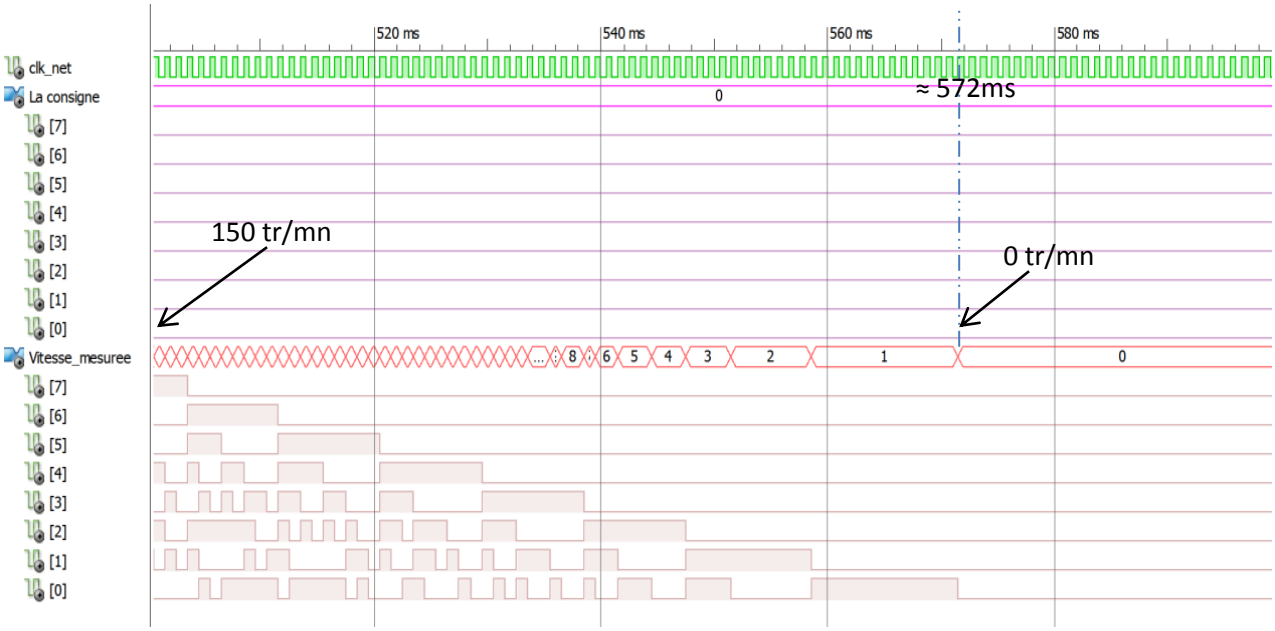


Fig. 4.36 : Zoom sur les chronogrammes (passage de 150 tr/mn vers 0 tr/mn)

Ce zoom permet de voir que le temps de réponse du système est inférieure à 80 ms

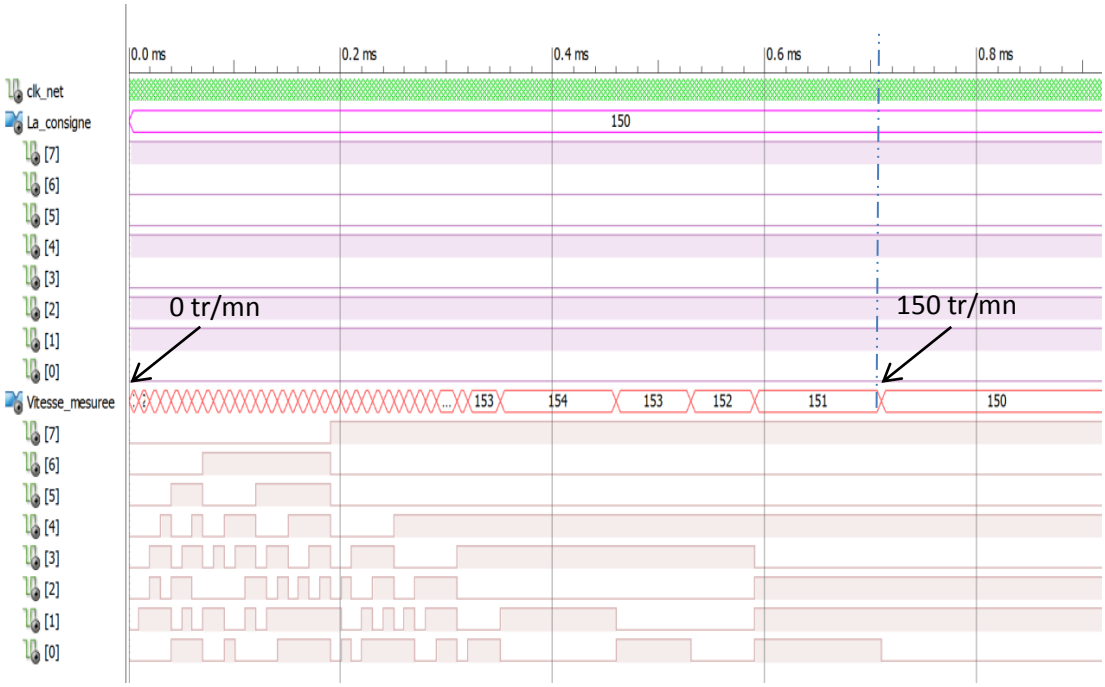


Fig. 4.37 : Zoom sur les chronogrammes (passage 0 tr/mn vers 150 tr/mn)

Ce zoom permet de remarquer un temps de réponse inférieure à 100 ms et un dépassement de3% .

Remarque :

Ces résultats preuve le bon fonctionnement du module PID synthétisé d'où sa validation pour être utilisé dans cette commande. (ie : Il fera partie du système réalisé (Fig. 4.2).

4.5.Commande par PID

Dans ce paragraphe, après avoir testé le bon fonctionnement des modules élémentaires, le fonctionnement de l'ensemble des modules représentés par le schéma synoptique de la figure (4.) va être simulé puis mis sous tests pratiques. Le système sera jugé fonctionnel, une fois on arrive à atteindre la consigne introduite par l'utilisateur. Cette vitesse doit être lisible sur les LEDs de la carte et sur le tachymètre. Nous allons introduire différent consignes à l'aide des commutateurs. La consigne est codée sur trois bits (SW2, SW1, SW3). La valeur décimale doit être multipliée par 30 pour avoir la consigne en tr/mn.

- **Résultats pratiques**

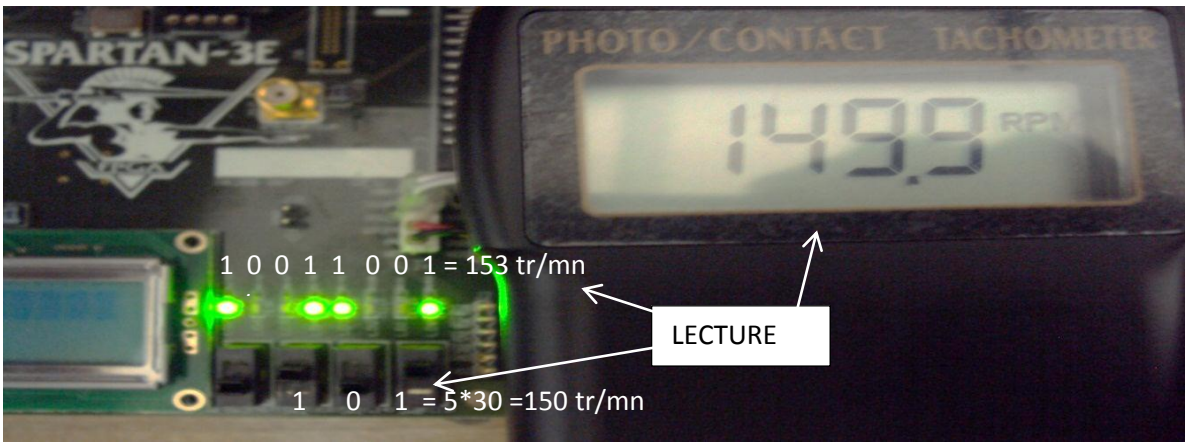


Fig. 4.38. : Résultat de la commande (consigne = 150 tr/mn)

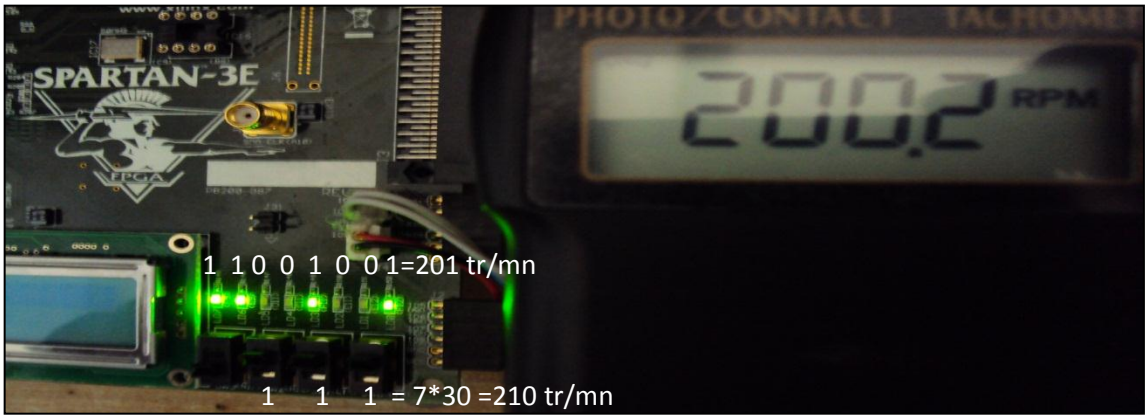


Fig. 4.39 : Résultat de la commande (consigne = 210 tr/mn)

Le tableau suivant donne un récapitulatif des résultats trouvés :

Expérience	Disposition des commutateurs	Consigne (tr/mn)	Vitesse mesurée sur LEDs (tr/mn)	Vitesse mesurée sur Tachymètre
a	101 = 5	5*30=150	10011001 = 153	149.9
b	111= 7	7*30=210	11011011 = 201	200.2

Tab.4.1 : Résumé des résultats pratiques de la commande par PID

Remarque :

La consigne et la vitesse sont codées en binaire. Voir la figure (4.38).

5. Résumés des résultats de la synthèse

pfe_mama_bit_cw Project Status (06/16/2012 - 01:32:35)			
Project File:	pfe_mama_bit_cw.xise	**Parser Errors:**	No Errors
Module Name:	pfe_mama_bit_cw	**Implementation State:**	Programming File Generated
Target Device:	xc3s500e-4fg320	• **Errors:**	No Errors
Product Version:	ISE 12.3	• **Warnings:**	No Warnings
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Tab.4.2: Résumé du projet réalisé

Remarque :

Dans le résumé du haut nous remarquons que les modules ont été synthétisés avec succès. Le nombre des erreurs est égal à zéro. Il est de même pour le nombre des alarmes. Les signaux sont complètement acheminés. Toutes les contraintes assignées ont été affectées aux emplacements physiques du circuit FPGA.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	197	9,312	2%	
Number used as Flip Flops	189			
Number used as Latches	8			
Number of 4 input LUTs	507	9,312	5%	
Number of occupied Slices	351	4,656	7%	
Number of Slices containing only related logic	351	351	100%	
Number of Slices containing unrelated logic	0	351	0%	
Total Number of 4 input LUTs	650	9,312	6%	
Number used as logic	507			
Number used as a route-thru	143			
Number of bonded IOBs	17	232	7%	
Number of BUFGMUXs	3	24	12%	
Number of MULT18X18SIOs	16	20	80%	
Average Fanout of Non-Clock Nets	1.53			

Performance Summary			[-]
Final Timing Score:	0 (Setup: 0, Hold: 0, Component Switching Limit: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	sam. 16. juin 01:22:02 2012	0	0	3 Infos (0 new)
Translation Report	Current	sam. 16. juin 01:32:00 2012	0	0	0
Map Report	Current	sam. 16. juin 01:32:15 2012	0	0	7 Infos (0 new)
Place and Route Report	Current	sam. 16. juin 01:32:30 2012	0	0	0
Power Report					
Post-PAR Static Timing Report	Current	sam. 16. juin 01:32:33 2012	0	0	4 Infos (0 new)
Bitgen Report	Current	sam. 16. juin 01:32:50 2012	0	0	0

Secondary Reports		[-]
Report Name	**Status**	**Generated**
Post-Synthesis Simulation Model Report	Current	sam. 16. juin 01:24:14 2012
Physical Synthesis Report	Out of Date	sam. 16. juin 01:32:14 2012
WebTalk Report	Current	sam. 16. juin 01:32:50 2012
WebTalk Log File	Current	sam. 16. juin 01:32:58 2012

Date Generated: 06/16/2012 - 01:34:25

Tab.4.3 : Résumé du taux d'utilisation des composants dans le système synthétisé

6. Conclusion

Après avoir décrit le banc d'essais et ses différentes parties, nous avons présenté les différents résultats de tests pour chaque module. Les résultats de simulations présentés au cours de ce chapitre à l'aide de l'outil ISim ainsi que les résultats pratiques pour chaque module synthétisé, à savoir : Le module diviseur de fréquence, le générateur des PWM et le contrôleur PID implémentés sur cible FPGA Spartan-3E, prouvent la bonne fonctionnalité du système conçu (i.e : la commande en vitesse du moteur à courant continu par un PID numérique). Un résumé sur le projet est donné en fin de chapitre. Une lecture rapide de ce résumé nous montre une faible utlisation des LUTs et l'utilisation de 80% des multiplieurs dédiés. Donc la carte Spartan-3E nous a facilité les différentes tâches lors de la synthèse de ce système de commande.

Références bibliographiques

[1]. K. J. Astrom and T. H. Hagglund, New Tuning Methods for PID Controllers, in *Proc. of 3rd European Conference*, 1995, pp. 2456-2462.

[2]. Shouling He and Xuping Xu, Hardware/Software Co-design Approach for an ADALINE Based Adaptive Control System, *Journal of Computers*, Vol. 3, No. 2, February 2008, pp. 29-36.

[3]. Craig Hackney, PGA Motor Control Reference Design, *Application Note: Spartan and Virtex FPGA Families, Xilinx* XAPP808 Vol. 1.0, September 16, 2005.

[4]. Mohamed Abdelati, FPGA-Based PID Controller Implementation, The Islamic University of Gaza,Palestine.

[5]. Anthony Cataldo, Low-priced FPGA options set to expand, *Electronic Engineering Times Journal*,No. 1361, 2005, pp. 38-45.

[6]. Gordon Hands, Optimized FPGAs vs. dedicated DSPs, *Electronic Product Design Journal,* Vol. 25, No.12, December 2004.

[7]. R. Jastrzebski, A. Napieralski, O. Pyrhonen and H. Saren, Implementation and simulation of fast inverter control algorithms with the use of FPGA circuit, Nanotechnology Conference and Trade Show, 2003, pp. 238-241.

[8]. Lin. F. S, Chen. J. F, Liang. T. J, Lin. R. L and Kuo, Y. C, Design and implementation of FPGA-based single stage photovoltaic energy conversion system, in Proceedings of IEEE Asia-Pacific Conference on Circuits and Systems, Taiwan, December 2004, pp. 745-748.

[9]. Bouzid Aliane and Aladin Sabanovic, Design and implementation of digital bandpass FIR filter in FPGA, Computers in Education Journal, Vol. 14, 2004, pp. 76-81.

[10]. M. Canet, F. Vicedo, V. Almenar and J. Valls, FPGA implementation of an IF transceiver for OFDM-based WLAN, IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation, USA, 2004, pp. 227-232.

[11]. Xizhi Li and Tiecai Li, ECOMIPS: An economic MIPS CPU design on FPGA, in Proceedings of the 4th IEEE International Workshop on System-on-Chip for Real-Time Applications, Canada, 2004, pp. 291-294.

[12]. H. D. Maheshappa, R. D. Samuel and A. Prakashan, Digital PID controller for speed control of DC motors, IEEE Technical Review Journal, Vol. 6, No. 3, 1989, pp. 171-176.

[13]. J. Tang, PID controller using the TMS320C31 DSK with on-line parameter adjustment for real-time DC motor speed and position control, IEEE International Symposium on Industrial Electronics, Vol. 2, 2001, pp. 786-791.

[14]. J.G. Ziegler, N.B. Nichols : « Optimum settings for automatic controllers. »Trans. ASME, 64, pp. 759-768, 1942.

[15]. I.M Horowitz (1963) : Synthesis of Feedback Systems. Acaddemic Press, New-York.

[16]. K.J. Aström, T. Hägglund: PID Controllers : Theory, Design and Tuning, Instrument Society of America, 2nd edition, 1995.

[17]. K.J. Aström, T. Hägglund, C.C. Hang, W.K. Ho: « Automatic Tuning and Adaptation for PID Controllers - A Survey. », Control Engineering Practice, 1:4, pp.699-714, 1993.

[18]. K.J. Aström, T. Hägglund (1984): «Automatic Tuning of Simple Regulators with Specification on Phase and Amplitude Margin. », Automatica, 20, pp. 645-651, 1984.

[19]. Michel Etique : « Régulation automatique », mars 2006, Yverdon- les Bains.

[20]. N.Bobal J.Bohn,J.Fessel,J.Machacek, « Digital Self-tuning Contollers »,Edition Wiley ,2010

[21]. Michael A. Johnson Mohammad H. Moradi « PID Control New Identification and Design Methods», Springer-Verlag London Limited 2005

[22]. B. Deforge & Q. David «Asservissement en position d'un axe linéaire», projet d'automatique, 2008.

[23] Rahim N.A. and Islam Z., "Field Programmable Gate Array-Based Pulse-Width Modulation for Single Phase Active Power Filter"; American Journal of Applied Sciences, Vol.6 (2009): pp. 1742-1747

[24] Retif J.M., Allard B., Jorda X. and Perez A, "Use of ASIC"s in PWM techniques for power converters", Proceedings of the International Conference on Industrial Electronics, Control and Instrumentation, IEEE Xplore Press, Maui, HI, USA.,(1993) pp: 683-688.

[25] Mohd. Shafie Bakar et al « Analysis of various PWM controls on single-phase Z-source inverter» article, Proceedings of 2010 IEEE Student Conference on Research and Development.

[26] Dariusz Czarkowski « Solving the Optimal PWM Problem for Single-Phase Inverters» article, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS; April 2002.

[27] Koutroulis E., Dollas A. and Kalaitzakis K., "High-frequency pulse width modulation implementation using FPGA and CPLD ICs", Journal of Systems Architecture, Vol.52 (2006): pp. 332–344

[28] Dancy A.P., Amirtharajah R. and Chandrakasan A.P., "High-Efficiency Multiple-Output DC–DC Conversion for Low-Voltage Systems", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 3, June 2000: pp.252-263

[29] Pong P.Chu, FPGA Prototyping by VHDL Examples, Edition Wiley USA, 2008.

[30] UG230: Spartan-3E Starter Kit User Guide, Xilinx, 2011.

http://www.xilinx.com/support/documentation/userguides/ug230.pdf

[31]UG332: Spartan-3 Generation Configuration User Guide,Xilinx, 2009.

http://www.xilinx.com/support/documentation/userguides/ug332.pdf

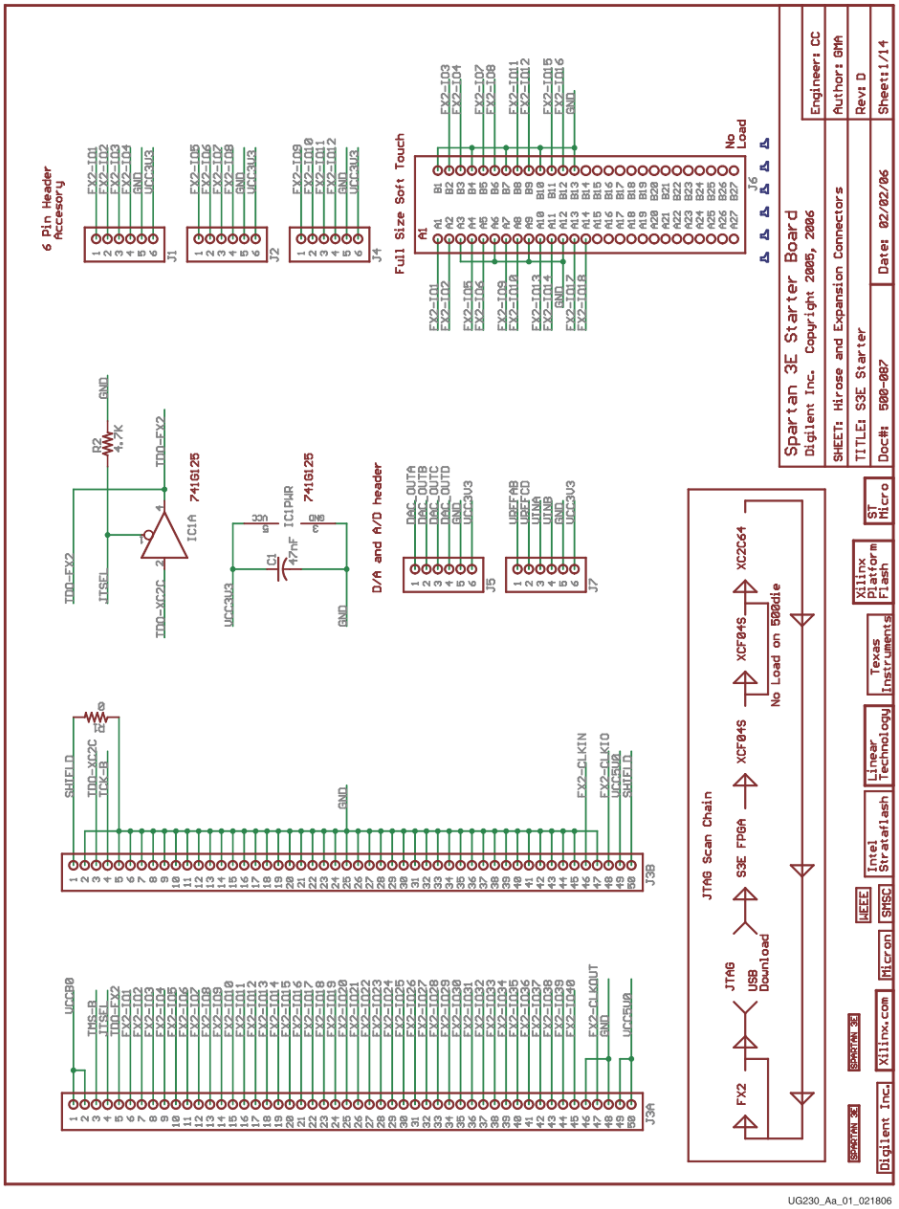
[32] Doglas L.Perry ,VHDL Programming by Example, Mc Graw Hill 4th edition,USA,2002.

[33] UG695 (v 12.3) : ISE In-Depth Tutorial, Xilinx, September 21,2010.

www.xilinx.com/products/boards/s3astarter/reference_designs.htm

[34] System Generator for DSP, Release 10.1, Xilinx, 2008.

A1 (Schématique J6)



UG230_Aa_01_021806

Figure A-1: **Schematic Sheet 1**

A2 (Régulateur de tension)

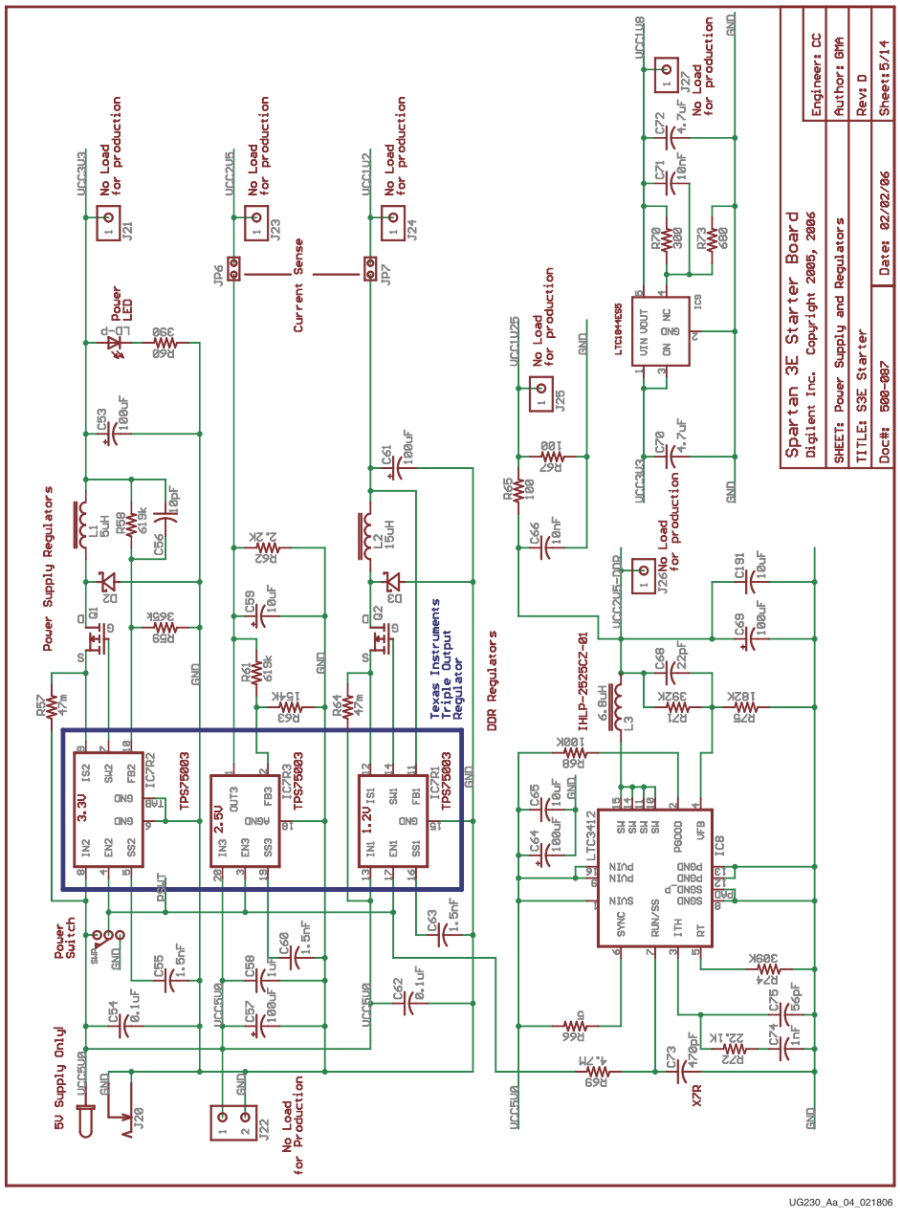
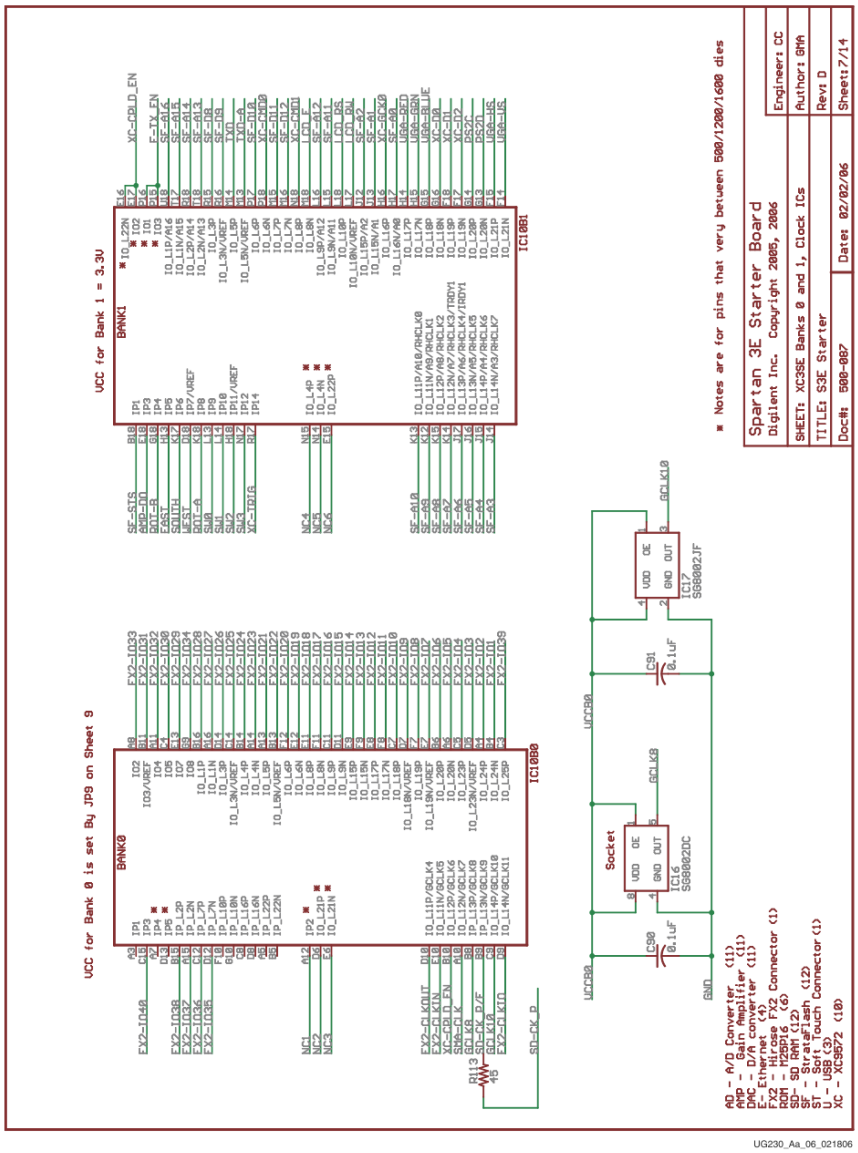


Figure A-4: **Schematic Sheet 5**

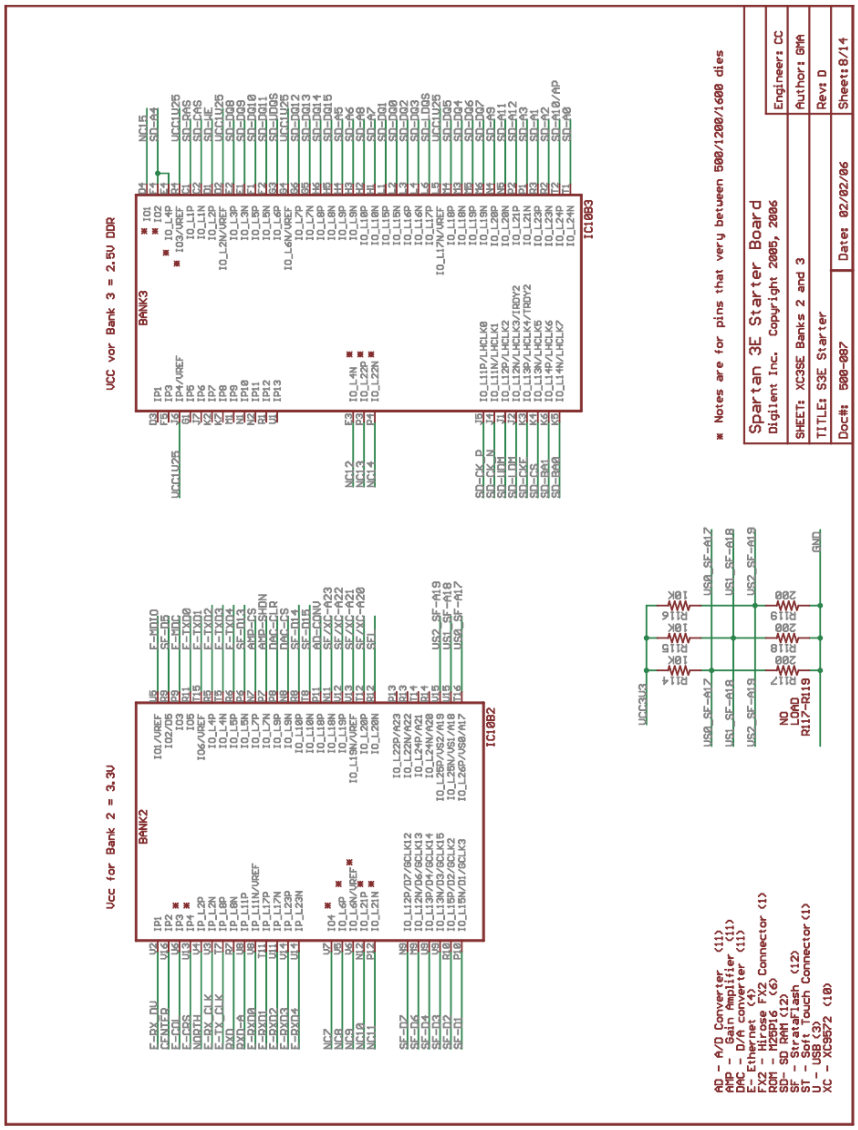
A3 (Bancs d’E/S))



UG230_Aa_06_021806

Figure A-6: **Schematic Sheet 7**

A4 (Bancs d'E/S)



UG230_Aa_07_021806

Figure A-7: **Schematic Sheet 8**

A5 (découplage de l'alimentation)

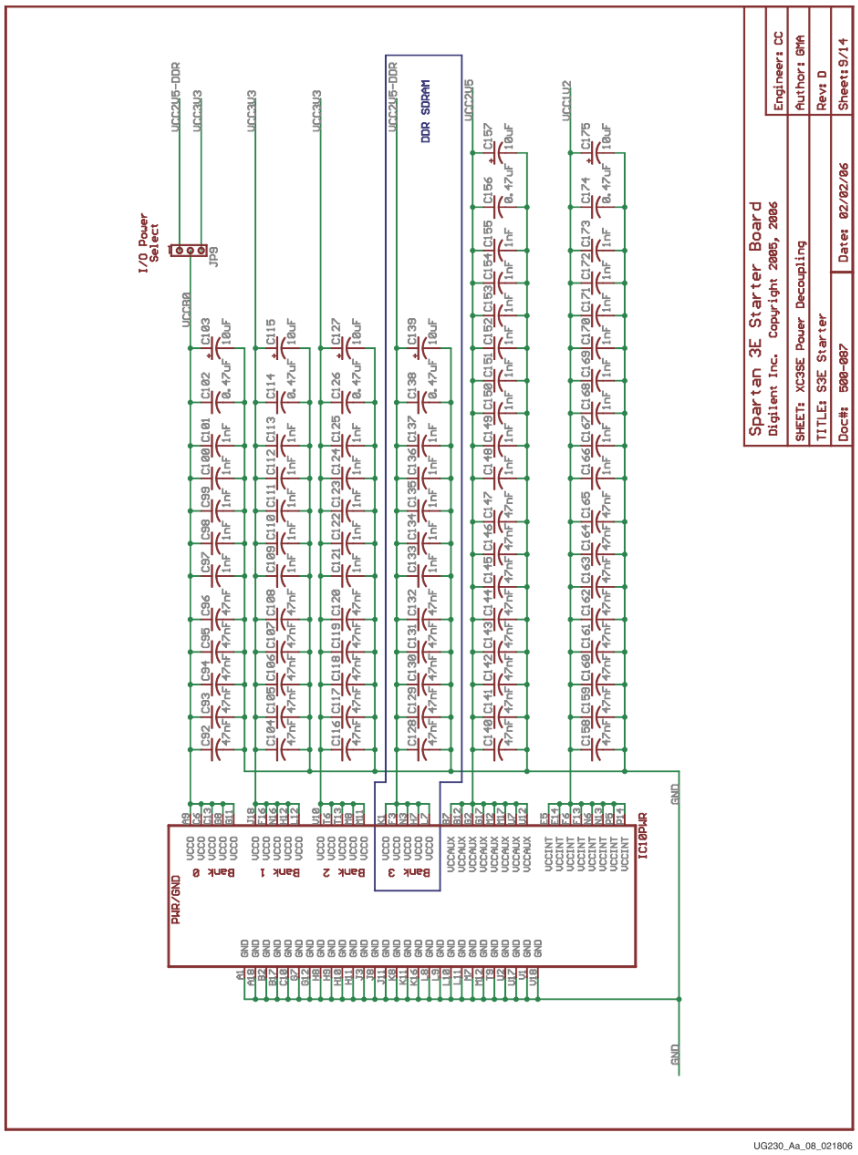
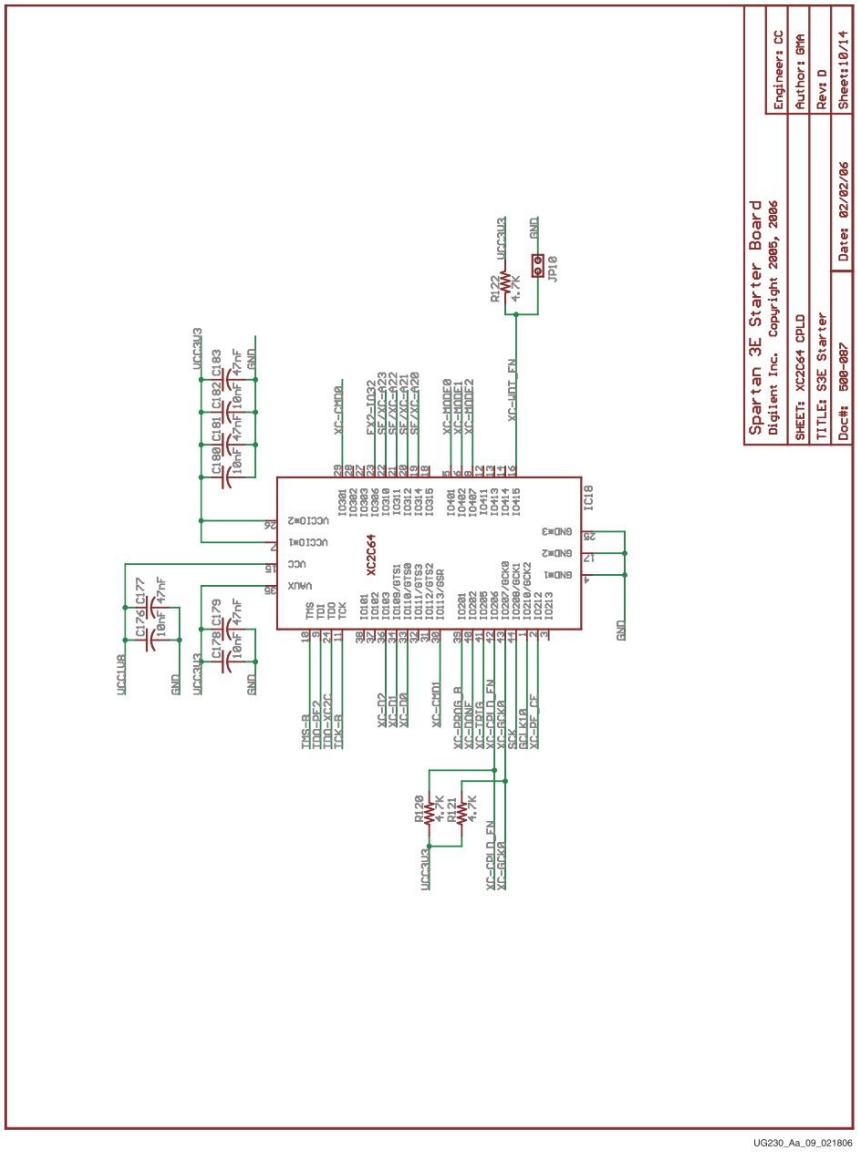


Figure A-8: **Schematic Sheet 9**

A6 (Le CPLD SC2C64A CoolRunner-II)



UG230_Aa_09_021806

Figure A-9: **Schematic Sheet 10**

A7 (Boutons poussoirs,Commutateurs)

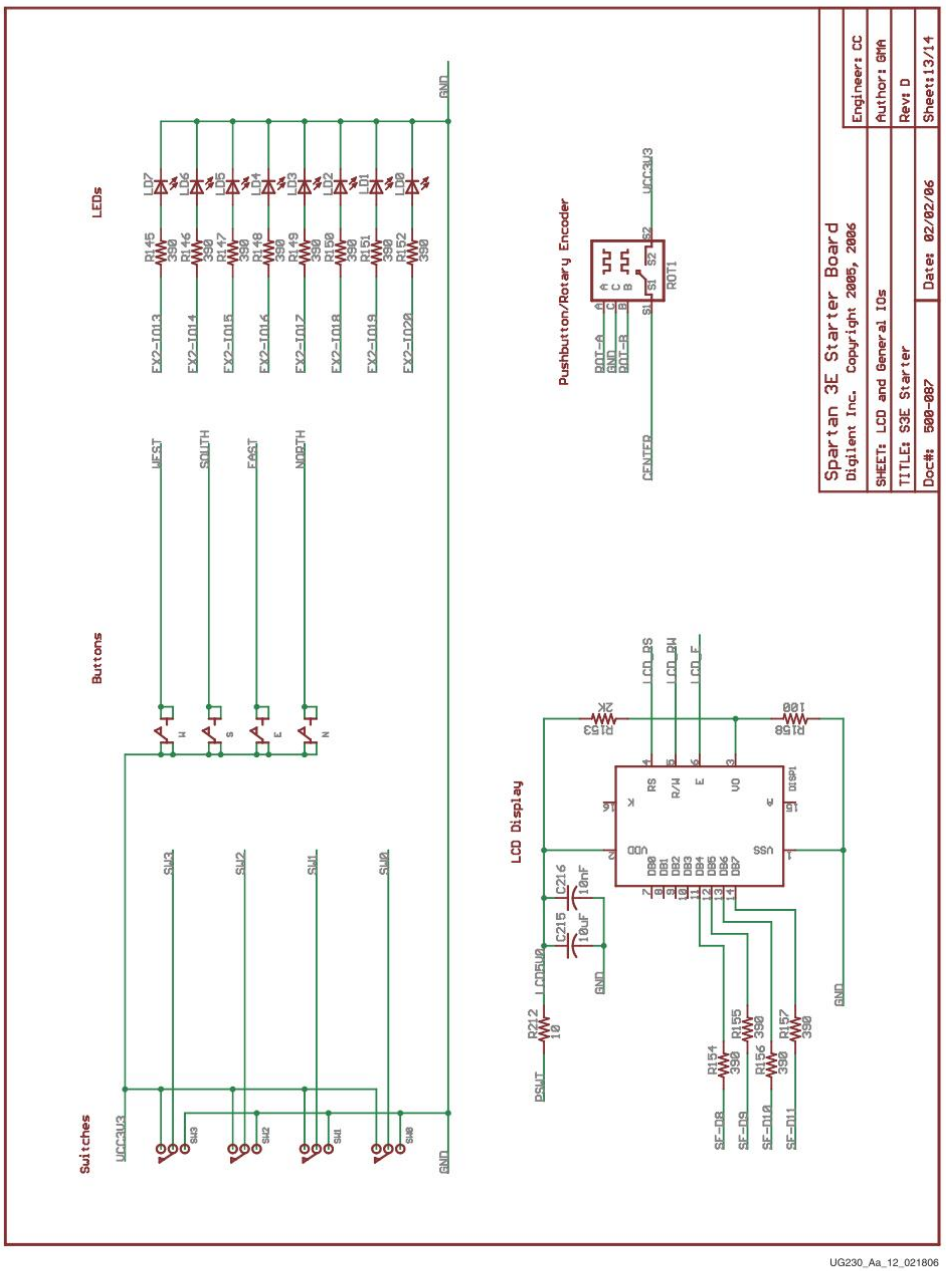
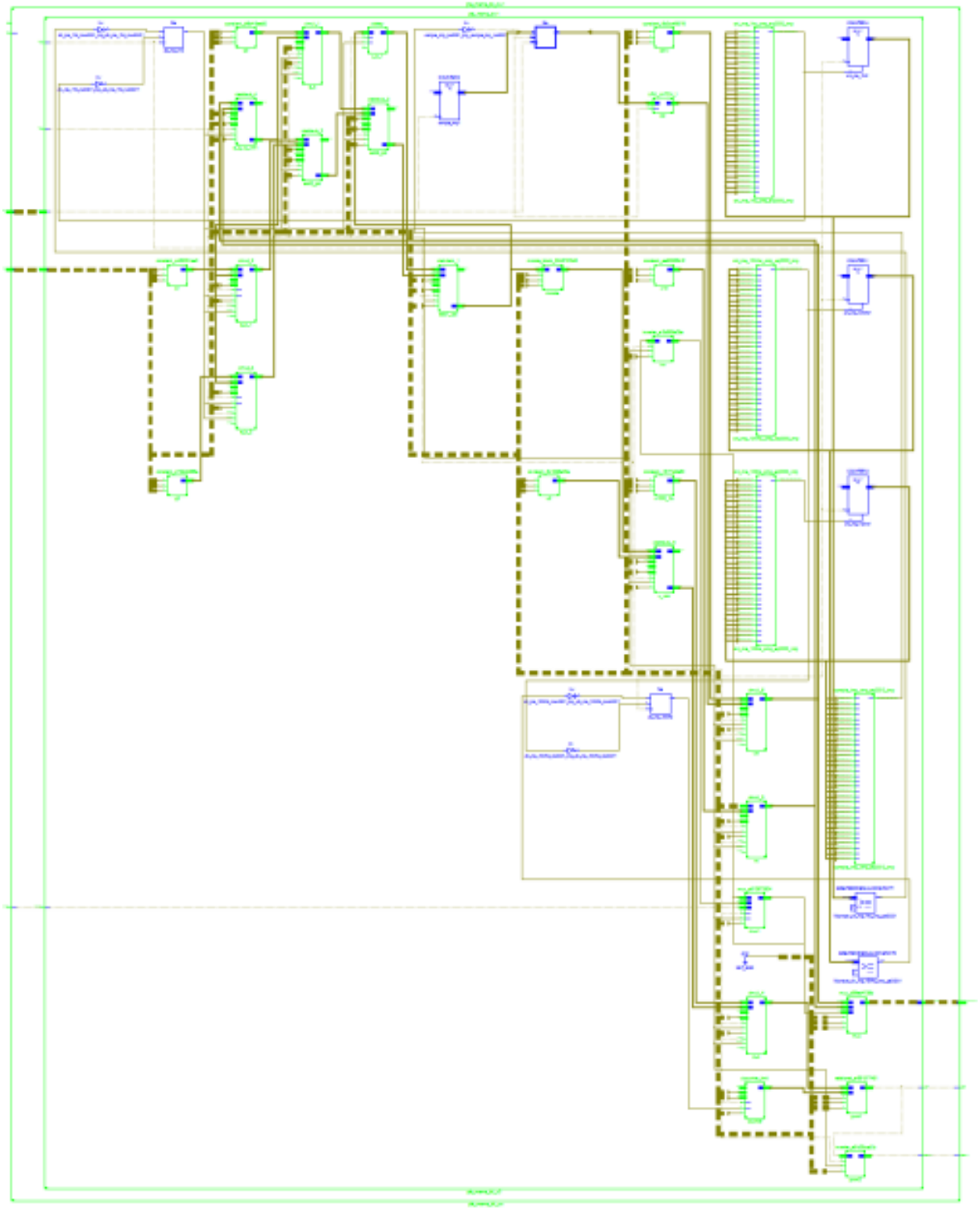


Figure A-12: **Schematic Sheet 13**

A8 (Schéma RTL du système synthétisé)



B1

FAIRCHILD
SEMICONDUCTOR®

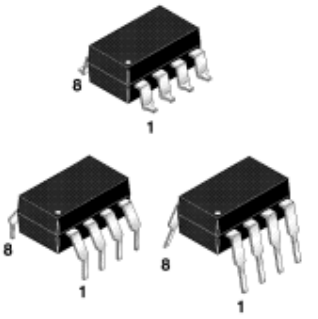
HIGH SPEED-10 MBit/s LOGIC GATE OPTOCOUPLERS

SINGLE-CHANNEL
6N137
HCPL-2601
HCPL-2611

DUAL-CHANNEL
HCPL-2630
HCPL-2631

DESCRIPTION

The 6N137, HCPL-2601/2611 single-channel and HCPL-2630/2631 dual-channel optocouplers consist of a 850 nm AlGaAS LED, optically coupled to a very high speed integrated photodetector logic gate with a strobable output. This output features an open collector, thereby permitting wired OR outputs. The coupled parameters are guaranteed over the temperature range of -40°C to +85°C. A maximum input signal of 5 mA will provide a minimum output sink current of 13 mA (fan out of 8).
An internal noise shield provides superior common mode rejection of typically 10 kV/µs. The HCPL- 2601 and HCPL- 2631 has a minimum CMR of 5 kV/µs.
The HCPL-2611 has a minimum CMR of 10 kV/µs.

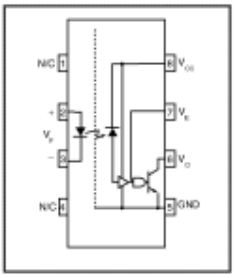


FEATURES

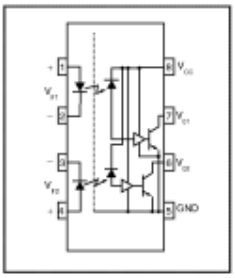
- Very high speed-10 MBit/s
- Superior CMR-10 kV/µs
- Double working voltage-480V
- Fan-out of 8 over -40°C to +85°C
- Logic gate output
- Strobable output
- Wired OR-open collector
- U.L. recognized (File # E90700)

APPLICATIONS

- Ground loop elimination
- LSTTL to TTL, LSTTL or 5-volt CMOS
- Line receiver, data transmission
- Data multiplexing
- Switching power supplies
- Pulse transformer replacement
- Computer-peripheral interface



6N137
HCPL-2601
HCPL-2611



HCPL-2630
HCPL-2631

TRUTH TABLE
(Positive Logic)

Input	Enable	Output
H	H	L
L	H	H
H	L	H
L	L	H
H	NC	L
L	NC	H

A 0.1 µF bypass capacitor must be connected between pins 8 and 5.
(See note 1)

B2

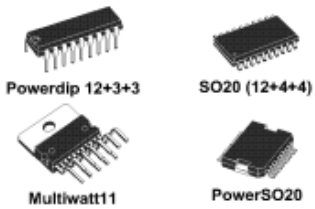
SGS-THOMSON MICROELECTRONICS

L6201
L6202 - L6203

DMOS FULL BRIDGE DRIVER

- SUPPLY VOLTAGE UP TO 48V
- 5A MAX PEAK CURRENT (2A max. for L6201)
- TOTAL RMS CURRENT UP TO L6201: 1A; L6202: 1.5A; L6203/L6201PS: 4A
- $R_{DS\ (ON)}$ 0.3 Ω (typical value at 25 °C)
- CROSS CONDUCTION PROTECTION
- TTL COMPATIBLE DRIVE
- OPERATING FREQUENCY UP TO 100 KHz
- THERMAL SHUTDOWN
- INTERNAL LOGIC SUPPLY
- HIGH EFFICIENCY

MULTIPOWER BCD TECHNOLOGY



ORDERING NUMBERS:
L6201 (SO20)
L6201PS (PowerSO20)
L6202 (Powerdip18)
L6203 (Multiwatt)

DESCRIPTION

The I.C. is a full bridge driver for motor control applications realized in Multipower-BCD technology which combines isolated DMOS power transistors with CMOS and Bipolar circuits on the same chip. By using mixed technology it has been possible to optimize the logic circuitry and the power stage to achieve the best possible performance. The DMOS output transistors can operate at supply voltages up to 42V and efficiently at high switching speeds. All the logic inputs are TTL, CMOS and µC compatible. Each channel (half-bridge) of the device is controlled by a separate logic input, while a common enable controls both channels. The I.C. is mounted in three different packages.

BLOCK DIAGRAM