REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE Dr. YAHIA FARES DE MEDEA



FACULTE DES SCIENCES ET DE LA TECHNOLOGIE

DEPARTEMENT DE GENIE ELECTRIQUE ET DE L'INFORMATIQUE

PROJET DE FIN D'ETUDE

En vue de l'obtention du diplôme de MASTER en génie électrique

Option :Automatique des systèmes

Thème

Acquisition, traitement et affichage de données liées à un processus industriel via le circuit numérique reconfigurable FPGA

Réalisé par :

- ✓ *BOULARIAH Aboubakre*
- ✓ *YKHLEF CHAOUCH Hamza*

Dirigé par :

Dr. MESSAI. A

Promotion 2013

Remerciement

Nous tenons tout d'abord à remercier le bon dieu Allah pour nous avoir donné le courage, la patience et la force nécessaires pour venir au bout de ce travail.

Nous remercions très chaleureusement nos familles pour leur patience et soutien.

Nos vifs remerciements s'adressent tout particulièrement à notre encadreur monsieur Dr.MESSAI Adnan qui nous a activement aidés dans ce projet sans oublier le Mr. MEKKI hamza pour son aide durant toute la durée du notre stage.

Nous remercions tous les personnels de La division études et développement d'instrumentation nucléaire (DEDIN).

Nous remercions tous les enseignants du département de génie électrique et informatique.

Nous n'oublierons pas tous ceux nous aidés de près ou de loin tout le long de notre étude.

BOULARIAH Aboubakre et YKHLEF CHAOUCH Hamza

Septembre 2013

DEDICACE

Je dédie ce modeste travail à :

Mes très chers parents pour leur encouragement durant la période de mes études,

Toute ma famille,

Tous les camarades du groupe d'automatique des systèmes,

Tous mes amis,

BOULARIAH Aboubakre

{Boubakre10@yahoo.fr}

DEDICACE

Je dédie ce modeste travail à :

Mes très chers parents,

Toute La famille YKHLEF CHAOUCH,

Tous mes amis.

YKHLEF CHAOUCH hamza

ملخص:

يهدف هذا المشروع إلى اقتناء، معالجة وعرض بيانات متعلقة بنظام صناعي باستخدام رقاقة رقمية قابلة للبرمجة من نوع FPGA. التصميم المحقق من أجل هذا المشروع تم محاكاته، توليفه وتركيبه على لوحة التطوير Memec Virtex-4 والتي تستعمل FPGA من نوع XC4VSX35-10FF668C لشركة Xilinx مع استعمال لوحة التطوير P240.

كلمات مفتاحية: P240 analog module , FPGA, Analog to Digital Converter (ADC), Acquisition Virtex-4™mb, VHDL,

Résumé :

Ce projet vise à l'acquisition, le traitement et l'affichage des données liées à un processus industriel via le circuit numérique reprogrammable FPGA. L'architecture réalisé dans le cadre de ce projet a été simulée, synthétisée et mise en application dans un kit de développement de Memec Virtex-4™mb utilise le FPGA XC4VSX35-10FF668C de Xilinx avec l'utilisation du module analogique P240.

Mot clé : Acquisition, convertisseur analogique-numérique(CAN), FPGA, Le module analogique P240, VHDL, Virtex-4™mb,

Abstract:

This project aims to the acquisition, processing and display of data related to industrial processes using the digital circuit reprogrammable FPGA. The architecture developed for this project was simulated, synthesized and implemented in a development kit Memec Virtex-4 ™ MB uses XC4VSX35-10FF668C Xilinx FPGA with the use of P240 analog module.

Key words: Acquisition, Analog to Digital Converter (ADC), FPGA, P240 analog module, VHDL, Virtex-4™mb.

SOMMAIRE

Résumé ..... *i*
Sommaire ..... *ii*
Liste des figures ..... *vi*
Liste des tableaux ..... *viii*
Introduction générale *1*

Chapitre I: Description de la plateforme matérielle

I.1 description du module analogique P240 3
I.1.1 Introduction 3
I.1.2 Description générale du module 3
I.1.3 Caractéristiques téchniques 4
I.1.3.1 Entrée analogique 4
I.1.3.2 Sortie analogique 4
I.1.3.3 Synchronisation 4
I.1.4 Description fonctionnel 4
I.1.5 Entrées analogiques 6
I.1.5.1 Circuit d'entrée analogique 6
I.1.5.2 Filtre d'Anti-repliement 8
I.1.5.3 Convertisseur analogique-numérique (CAN) 9
I.1.6 Sorties analogiques 10
I.1.6.1 Convertisseur numérique-analogique (CNA) 10
I.1.6.2 Filtre passe-bas de reconstruction analogique du DAC 11
I.1.7 Architecture d'horloge 12
I.1.7.1 Le circuit Tampon d'horloge (CDCP180) 12
I.1.7.2 Système d'horloge de l'ADC 12
I.1.7.3 Système d'horloge du DAC 13
I.1.7.4 Système d'horloge recommandé pour le module analogique P-240 13
I.1.7.5 Source du signal d'horloge pour le module analogique P-240 14
I.1.8 Alimentation électrique 15
I.1.9 Conclusion 16
I.2 Présentation des circuits FPGAs 16
I.2.1 Introduction 16
I.2.2 Les ASICs (Application Specific Integrated Circuit) 17
I.2.2.1 Avantages et inconvénients de l'utilisation d'ASIC 17

I.2.3 Les PLD (Programmable Logic Device) 18
I.2.4 Les FPGAs 18
I.2.4.1 Architecture interne des circuits FPGAs 19
I.2.4.2 Les IOB (Input Output Bloc) 21
I.2.4.3 Les différents types d'interconnexion 23
I.2.5 Notre choix 25
I.2.6 Description du kit de développement virtex-4 MB 26
I.2.6.1 Vue d'ensemble 26
I.2.6.2 Le kit de système du Virtex-4 MB 26
I.2.6.3 Description fonctionnelle 27
I.2.6.4 Conclusion 36

Chapitre II : description des outils de développement logiciel

II.1 Le langage « VHDL » 37
II.1.1 Introduction 37
II.1.2 Structure d'une description VHDL 37
II.1.3 Déclaration des bibliothèques 38
II.1.4 Déclaration de l'entité et des entrées / sorties (I/O) 39
II.1.4.1 le nom du signal 39
II.1.4.2 Le sens du signal 40
II.1.4.3 Le type 40
II.1.5 Déclaration de l'architecture correspondante à l'entité (description du fonctionnement) 40
II.1.6 Les instructions de base (mode « concurrent »), logique combinatoire 41
II.1.6.1 Les opérateurs 41
II.1.6.2 Les instructions du mode « concurrent » 43
II.1.7 Les instructions du mode séquentiel 44
II.1.7.1 Définition d'un PROCESS 44
II.1.7.2 Les Types 46
II.1.8 Exemple du programme en VHDL fait durant le stage 48
II.2 L'outil de développement ISE de Xilinx 52
II.2.1 Introduction et aperçu 52
II.2.2 Etapes d'ISE design flow 52
II.2.2.1 Entrée du design 52
II.2.2.2 Synthèse 52
II.2.2.3 Implémentation 52

II.2.2.4 Vérification 53
II.2.2.5 Configuration du circuit 53
II.2.3 Familiarisation et procédure de réalisation d'un projet avec ISE 53
II.2.3.1 Créer un nouveau projet 53
II.2.3.2 Créer un fichier source HDL 56
II.2.3.3 Simulation d'un fichier source HDL 58
II.2.4 Conclusion 60

Chapitre III: Implémentation de différents modules sur la carte FPGA

III.1 Introduction 61
III.2 Implémentation hardware des modules d'acquisition, de traitement et d'affichage de données 61
III.3 Une unité d'acquisition et de traitement de données 61
III.4 Une unité d'affichage de données 62
III.4.1 Le schéma fonctionnel de l'afficheur LCD 16x2 63
III.4.2 Le rôle de différentes broches 63
III.5 Description de l'architecture interne des modules conçus 64
III.5.1 Description du circuit globale 64
III.5.2 Architecture des différents sous-modules composant le module global 67
III.5.2.1 Description du bloc buffer1 67
III.5.2.2 Description du bloc buffer2 68
III.5.2.3 Description du bloc compl2_1 68
III.5.2.4 Description du bloc compl2_2 69
III.5.2.5 Description du bloc filtre_1 69
III.5.2.6 Description du bloc filtre_2 70
III.5.2.7 Description du bloc dec_1 70
III.5.2.8 Description du module dec_2 71
III.5.2.9 Description du bloc dec_ 3 71
III.5.2.10 Description du bloc dec_4 72
III.5.2.11 Description du bloc lcd16x2_ctrl 72
III.6 conclusion 73

Chapitre VI: Les résultats pratiques

IV.1 Les résultats de simulation et expérimentaux 74
IV.2 Les instruments utilisés lors de la validation expérimentale 74
IV.2.1 source des signaux d'entrée 74

IV.2.2 le kit d'acquisition de données P240 74
IV.2.3 le kit de développement Virtex-4 MB 74
IV.2.4 L'oscilloscope 74
IV.2.5 le multimètre 74
IV.3 les résultats de simulation 75
IV.3.1 Résultats de simulation pour le circuit global 75
IV.3.2 Simulation du fonctionnement du buffer d'entrée du module global 75
IV.3.3 Résultats de simulation pour le module complément à deux 76
IV.3.4 Résultats de simulations obtenus pour le filtre du traitement numérique 76
IV.3.5 Résultats de simulation obtenus pour le Décodeur 1 77
IV.3.6 Résultats de simulations obtenus pour le Décodeur 2 77
IV.4 Résultats obtenus lors de la phase des tests pratiques 78
IV.4.1 l'affichage des données acquises en ASCII 78
IV.4.1 l'affichage en binaire 80
IV.5 Conclusion 81
Conclusion générale 82
bibliographie 84
Annexe A 85
Annexe B 86

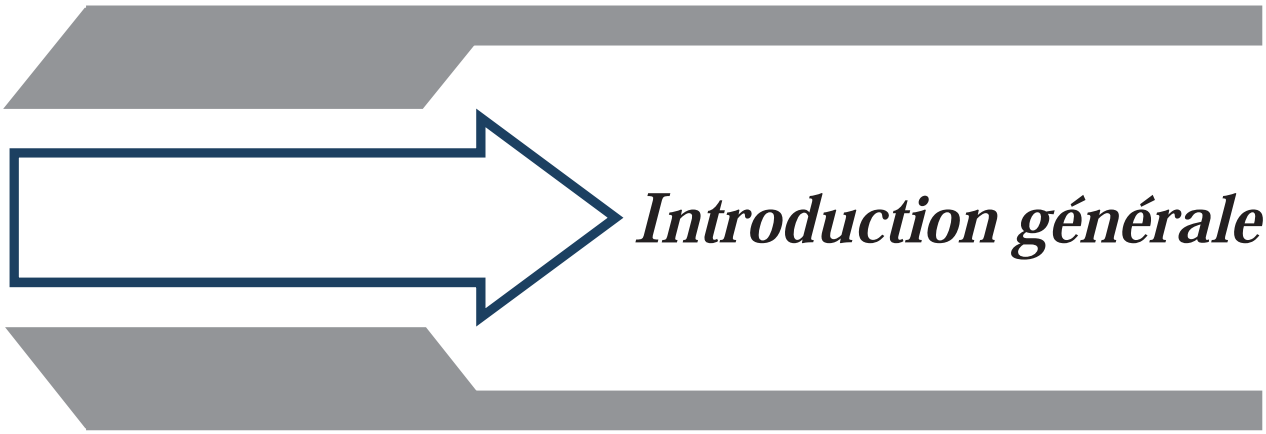
Liste des figures

Figure I.1 : Vue supérieure du module analogique P240 5
Figure I.2 : Schéma bloc fonctionnel simplifié du module analogique P-240 5
Figure I.3 : Section d'entrée analogique du module P-240 6
Figure I.4 : Schéma bloc interne de l'amplificateur THS4509 7
Figure I.5 : Circuit analogue d'entrée du module P240 (condensateur d'accouplement à C.A.non montré) 7
Figure I.6 : Filtre Tchebychev d'Anti-repliement du 7ème ordre 8
Figure I.7 : Courbe de réponse du filtre Anti-repliement 8
Figure I.8 : Schéma bloc interne de l'ADC ADS5500 9
Figure I.9 : Chronogramme de l'opération de conversion au sein de l'ADC ADS5500 9
Figure I.10 : Partie de la sortie analogique du module P-240 10
Figure I.11: Schéma bloc interne du DAC 11
Figure I.12 : Le circuit Tampon d'horloge 12
Figure I.13 : Couplage AC, entrée différentielle de l'horloge pour l'ADC 12
Figure I.14 : Système d'horloge recommandé 14
Figure I.15 : Circuit générateur d'horloge à base de « ICS8482 » sur le Kit FPGA 15
Figure I.16 : Classification des circuits numériques 18
Figure I.17 : Architecture interne du FPGA 20
Figure I.18 : Structure d'une cellule SRAM 21
Figure I.19 : Schéma d'une cellule logique (XC4000 de Xilinx) 21
Figure I.20 : Schéma d'un bloc d'entrée/sortie (IOB) 22
Figure I.21 : Connexions à usage général et détail d'une matrice 24
Figure I.22 : Les interconnexions directes 24
Figure I.23 : Les longues lignes 25
Figure I.24 : Le kit de développement Virtex 4 26
Figure I.25 : Schéma fonctionnel de la plate-forme de développement du Virtex-4 MB 28
Figure I.26 : L'interface SPI-4.2 29
Figure I.27 : Interface DDR SDRAM 30
Figure I.28: Interface du flash 30
Figure I.29: Sources d'horloge sur le conseil du Virtex-4mb 31
Figure I.30: ICS8442 synthétiseur d'horloge 32
Figure I.31: L'interface Ethernet 10/100 33

Figure I.32: USB 2.0 to RS232 Serial Interface 34
Figure I.33: L'interface RS232 34
Figure I.34: Vertex-4MB Development Board JTAG chain 35
Figure II.1 : Un décodeur 1 parmi 4 38
Figure II.2 : Les différent sens d'un signale 40
Figure II.3 : Résultat de simulation 51
Figure II.4 : La fenêtre principale de l'ISE 53
Figure II.5 : La fenêtre New Project Wizard 54
Figure II.6 : La fenêtre New Project Wizard – Device Properties 55
Figure II.7 : La fenêtre New Project Wizard – Project Summary 55
Figure II.8 : ISE 56
Figure II.9 : La fenêtre New Source Wizard 56
Figure II.10 : La fenêtre New Source Wizard – Define Module 57
Figure II.11 : La fenêtre New Source Wizard - Summary 57
Figure II.12 : ISE avec notre fichier source crée 58
Figure II.13 : ISE avec notre fichier source « AZERTY.vhd » 59
Figure II.14 : Les résultats de simulation 60
Figure III.1 : La chaine d'acquisition utilisée 61
Figure III.2 : Les paramètres du filtre utilisé 62
Figure III.3 : Schéma fonctionnel de l'afficheur LCD16x2 63
Figure III.4 : L'architecture de circuit globale et leur sous modules 66
Figure IV.1 : Résultat de simulation du circuit global 75
Figure IV.2 : Résultat de simulation du module buffer 75
Figure IV.3 : Résultat de simulation du module comp2 76
Figure IV.4 : Résultat de simulation du filtre numérique 76
Figure IV.5 : Résultat de simulation du module dec1 77
Figure IV.6 : Résultat de simulation du module dec2 77
Figure IV.7 : Exemple 1 pour le test pratique. 78
Figure IV.8 : Exemple 2 pour le test pratique 78
Figure IV.9 : Exemple 3 pour le test pratique 79
Figure IV.10 : Exemple 4 pour le test pratique 79
Figure IV.11 : Exemple1 pour l'affichage en binaire 80
Figure IV.12 : Exemple 2 pour l'affichage en binaire 80

Liste des tableaux

Tableau I.2 : Signaux D'interface du LCD 33
Tableau I.3 : Les pins d'affectation des LEDs 35
Tableau I.4 : Tensions entrée-sortie de banc 36
Tableau II.1 : Les opérateurs logiques 42
Tableau II.2 : Les opérateurs arithmétiques 42
Tableau II.3 : Les opérateurs relationnels 43
Tableau II.4 : Les assignations sélectives et conditionnelleS 45
Tableau III.1: Description des signaux Entrées/Sorties du circuit global 65
Tableau III.2 : Description des signaux Entrées/Sorties du module buffer_1 67
Tableau III.3 : Description des signaux Entrées/Sorties du module buffer_2 68
Tableau III.4 : Description des signaux Entrées/Sorties du module compl2_1 68
Tableau III.5 : Description des signaux Entrées/Sorties du module compl2_2 69
Tableau III.6 : Description des signaux Entrées/Sorties du module filter_1 69
Tableau III.7 : Description des signaux Entrées/Sorties du module filter_2 70
Tableau III.8 : Description des signaux Entrées/Sorties du module dec_1 70
Tableau III.9 : Description des signaux Entrées/Sorties du module dec_2 71
Tableau III.10 : Description des signaux Entrées/Sorties du module dec_3 71
Tableau III.11 : Description des signaux Entrées/Sorties du module dec_4 72
Tableau III.12 : Description des signaux Entrées/Sorties du module Lcd16x2_ctrl 72



Introduction générale

Introduction générale

Le traitement numérique du signal s'oppose au traitement analogique du signal, sur lequel se basait tous les anciens systèmes électroniques et qui manipule les signaux à leur état naturel, c'est-à-dire sous forme d'ondes (signaux analogiques). Le traitement numérique consiste à manipuler les signaux sous forme de suites d'échantillons numériques obtenus à partir de signaux physiques réels au moyen de convertisseurs analogique/numériques (CAN), leur appliquer les traitements mathématiques appropriés, puis éventuellement les restituer sous leur forme originelle à l'aide d'un convertisseur numérique-analogique (CNA). Les systèmes électroniques actuels, à l'image des téléphones mobiles, intègrent souvent les deux types de traitement, on parle alors de système mixte analogique/numérique. L'acquisition et le traitement du signale sont aujourd'hui des techniques très largement utilisée et dont chacun voit bien la nécessité dans plusieurs domaines.

Nous notons par ce début que la quasi-totalité du travail décrit lors de ce mémoire (acquisition et traitement de données) est effectué à la base d'une carte électronique munie d'un circuit FPGA Virtex4. Celle-ci est également dotée d'une interface d'acquisition de données développée par la firme ***Avnet*** et commercialisée sous l'appellation : Analog module P-240.

La carte FPGA évoquée en haut (***Virtex-4^{TM} MB Development Kit***) est un kit de développement FPGA ordinaire qui contient les fonctionnalités généralement offertes par la quasi-totalité des kits qu'on trouve souvent dans le commerce de l'équipement électronique. Mais, l'interface d'acquisition, est un module spécifique, non répandu, qui requiert une attention et un soin particuliers pour une utilisation optimale de l'ensemble (Carte FPGA & Module d'interfaçage).

Pour ce qui est des filtres de traitement, implémentés au sein du circuit FPGA, nous avons utilisé des outils softwares connus tels les langages : matlab, simulink, Xilinx ISE 14.1, ModelSim de mentor graphics, … etc.

Par conséquent, et pour décrire au mieux le travail réalisé, nous avons jugé utile que le présent mémoire sera subdivisé en quatre chapitres essentiels :

- Lors du premier chapitre, nous donnerons une description plus au moins exhaustive sur de la plateforme matérielle utilisée pour la concrétisation de ce travail, à savoir le module analogique P240, ainsi que la carte FPGA (virtex4).
- Quant au deuxième chapitre, on abordera la description des outils software utilisés au cours de notre travail. Il s'agira, d'une présentation du langage de programmation : VHDL,

ainsi que une brève présentation sur les modes d'utilisation de l'outil de développement de Xilinx : ISE 14.1.

- Au cours du troisième chapitre, on détaillera le travail réalisé pour concrétiser les tâches d'acquisition et de traitement de données au sein du circuit FPGA.
- Dans le quatrième chapitre on exhibera les résultats de simulation ainsi que les résultats des testes pratiques obtenus au terme de notre travail.
- On achèvera ce manuscrit par une conclusion générale, au cours de laquelle on dressera un bilan sur les travaux déjà réalisés et les développements futurs capables d'étoffer un tel travail.

CHAPITRE I

Choix de la plateforme matérielle

Ce chapitre est subdivisé en deux parties distinctes. La première partie étant réservée au module d'acquisition de données, alors que la deuxième traitera la description du circuit FPGA.

I.1 DESCRIPTION DU MODULE ANALOGIQUE P240 [1]

I.1.1 Introduction

Pour la mise en œuvre du travail qui nous a été assigné, nos premiers efforts dans ce contexte ont tous été détournés vers le module d'interfaçage P-240. Après une recherche bibliographique approfondie en terme de quantité de documents consultés et charge horaire allouée, il s'est avéré que la documentation technique traitant les aspects pratique visant à l'exploitation de ce module sont vraiment rares voir introuvables. Ceci est dû au fait que le module en question a été produit par **Avenet** en 2006 et n'a été commercialisé que durant une petite période et à petite quantité à travers le monde entier (quelques prototypes seulement). Une situation délicate, qui a fait en sorte que les aspects techniques et les exemples d'application sont non disponibles soit à travers le net ou sur des documents imprimés. Donc, à part un datasheet descriptif et généraliste, fourni par le constructeur, et qui reste assez loin pour donner des informations techniques valables aux yeux d'un utilisateur donné.

Devant une situation aussi confuse que celle-ci, nous avons décidé d'initier un travail visant à étudier et caractériser ce module d'interfaçage. Pour cela, nous avons adopté une stratégie qui consiste en l'exploration technique détaillée du module et ce à travers la documentation technique des différents constituants (Circuits ADC, Circuit DAC, Générateurs d'horloge, amplificateur d'instrumentation, …, etc.) séparément. Cette étape sera par la suite suivie par une application effective des connaissances ainsi acquises sur l'équipement considéré.

I.1.2 Description générale du module

Le module analogique P-240 est une module d'interfaçage spécialement conçu pour l'acquisition de données à la carte : ***Virtex-4^{TM} MB Development Kit.*** Il est doté de quatre voies analogiques indépendantes, deux entrées (IN1 et IN2 sur la figure I.1) et deux sorties (CH1 et CH2 sur la figure I.1). Les canaux analogiques d'entrée sont identiques du point de vue conception et sont configurés en entrées différentielles. Les 02 convertisseurs A/N : *ADS5500* 14-bits, 125 MSPS de Texax instrument, transforment les signaux analogiques entrants en 14-bits de données pour le FPGA situé sur la carte de base. La bande passante des entrées analogique de 750 MHz de ces composants les rend idéaux pour des applications de la haute fréquence intermédiaire (IF) ou des signaux sous-échantillonnés dans la bande UHF. Un DAC double canal à interpolation, de Texas Instruments, 16 bits, 500 MSPS, 2x à 8x, génère des sorties analogiques. Le DAC5687 offre

des blocs de traitement des signaux optionnels incluant des filtres numériques d'interpolation, un mélangeur complexe avec un oscillateur à commande numérique 32 bits et la modulation en quadrature, tous programmables via une interface série. Les sorties du CNA couplées par transformateur de 50 ohms, incluent des étages analogiques de filtre à reconstruction.

Le module analogique P-240 d'Avnet Electronics Marketing, avec le plein appui sur les outils « System Generator » de Xilinx et Simulink pour DSP de MathWorks, simplifie le développement d'applications et aide les concepteurs à répondre rapidement et efficacement à leurs besoins en matière de R&D en électronique numérique.

I.1.3 Caractéristiques techniques

I.1.3.1 Entrée analogique

- (2) ADC, 14 bits, 125 Msps.
- Entrées 50 Ohm, single-ended (référenciée à la masse).
- filtre RLC passe-bas d'anti-repliement du 7ème ordre.

I.1.3.2 Sortie analogique

- DAC double canal à interpolation 2x-8x, 16 bits, 500 Msps.
- référence interne et externe.
- Sorties 50 Ohm couplées avec transformateurs.
- filtre de reconstruction RLC passe-bas du 5ème ordre.

I.1.3.3 Synchronisation

- Entrée d'horloge LVDS (*Low Voltage Differential Signalling*) externe.
- l'entrée d'horloge couplée par transformateur.

I.1.4 Description fonctionnelle

Lors de cette section, on abordera la description fonctionnelle du module analogique P-240. La figure (I.1) montre un aperçue réel du module, alors que la figure (I.2) fournit un schéma bloc fonctionnel simplifié.

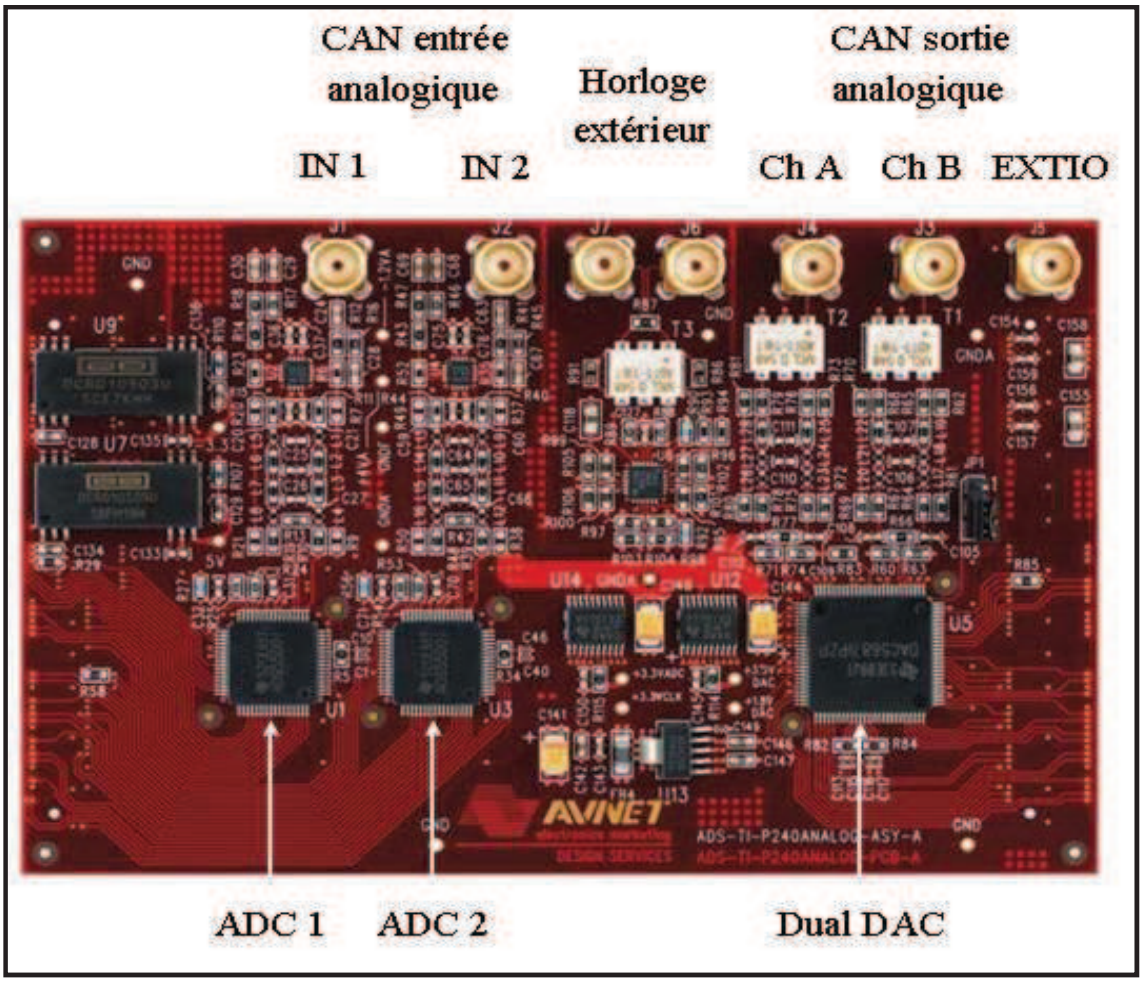


Figure I-1 : *Vue supérieure du module analogique P240*

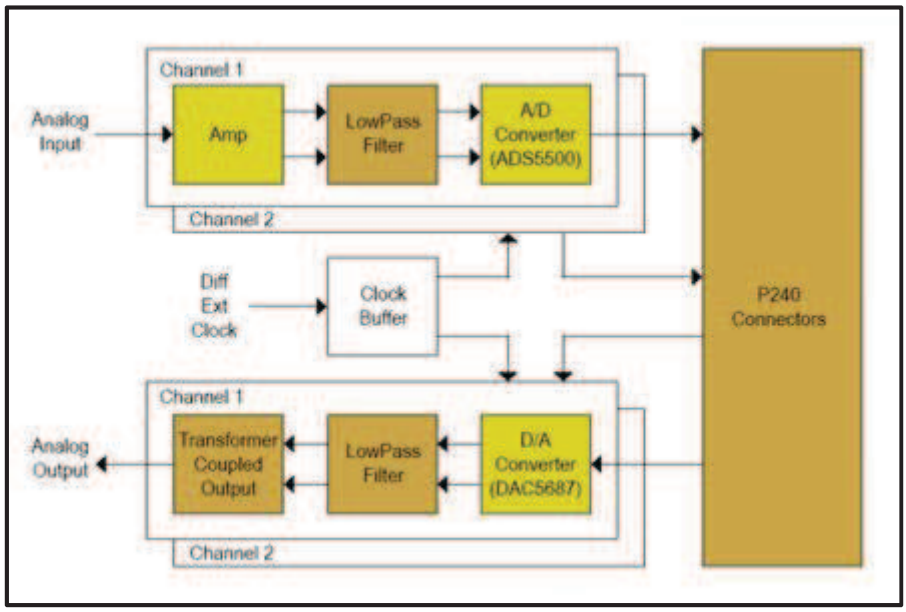


Figure I-2 : *Schéma bloc fonctionnel simplifié du module analogique P-240*

I.1.5 Entrées analogiques

La figure (I.3) illustre la section des deux entrées analogiques du module P-240.

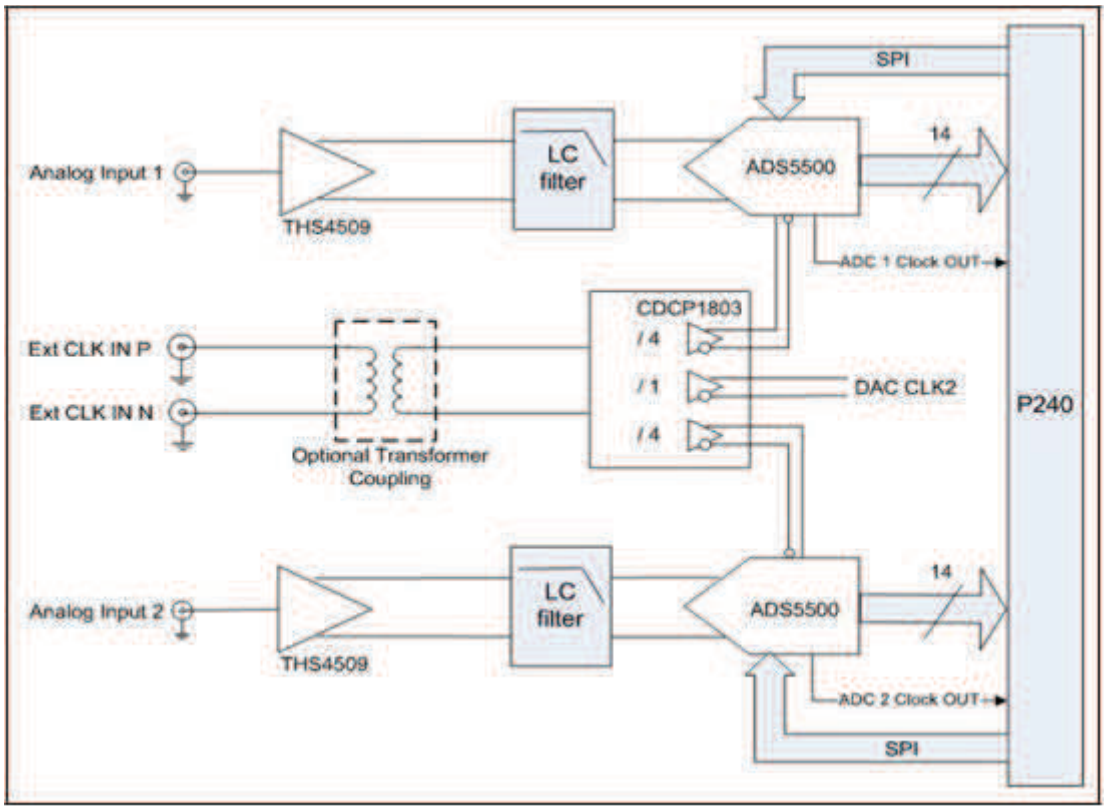


Figure I-3 : *Section d'entrée analogique du module P-240*

I.1.5.1 Circuit d'entrée analogique

Chacune des deux entrées analogiques accepte une source de signal single-ended (référencié à la masse) avec une impédance de sortie nominale de 50 ohms. Un couplage AC pour chaque entrée est assuré par un condensateur de 220 nF (A noter que ce condensateur n'est pas représenté sur la figure I-5). L'étage d'entrée utilise un amplificateur différentiel à large bande : *THS4509* (figure I-4). Celui-ci convertit l'entrée référenciée à la masse à une entrée différentiel, définit la tension du mode commun correspondant à l'entrée de l'ADC *ADS5500* et fournit un gain de 10 dB par défaut dans une configuration préalable. Cet étage d'entrée actif permet de fixer diverses valeurs du gain en fonction du niveau du signal d'entrée qui doit correspondre à la gamme d'entrée de l'ADC *ADS5500*, maximisant ainsi le rapport de signal sur bruit SNR. Le gain est une fonction de l'impédance de la source du signal, et d'une résistance de contre-réaction de 348 ohms. D'autres valeurs de gains peuvent être réglées en modifiant ces valeurs de résistance.

La source de signal est supposée avoir une impédance de sortie de 50 ohms. Le signal d'entrée est appliqué aux bornes d'une résistance de 69,3 ohms et un condensateur de 0,22 pF lié à la masse, en combinaison avec l'impédance d'entrée du circuit amplificateur. Un condensateur de 0,22 pF et une résistance de 49,9 Ohm est inséré à la terre à travers la résistance de 69,8 ohms et le condensateur de 0,22 sur l'entrée négative pour équilibrer le circuit.

Une alimentation asymétrique de ***4V*** et ***-1V*** est utilisée pour régler les tensions en mode commun que ce soit en entrée ou en sortie de manière à approximer la mi- alimentation pendant le réglage de la tension du mode commun à l'entrée de l'ADC *ADS5500* à une valeur typique de 1,55V. Cela garanti le maintien d'une marge de sécurité quant à l'alimentation maximale appliquée aux transistors internes de l'amplificateur *THS4509.*

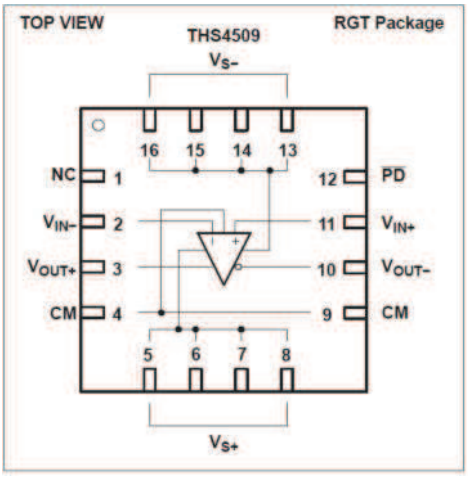


Figure I-4 : *Schéma bloc interne de l'amplificateur THS4509*

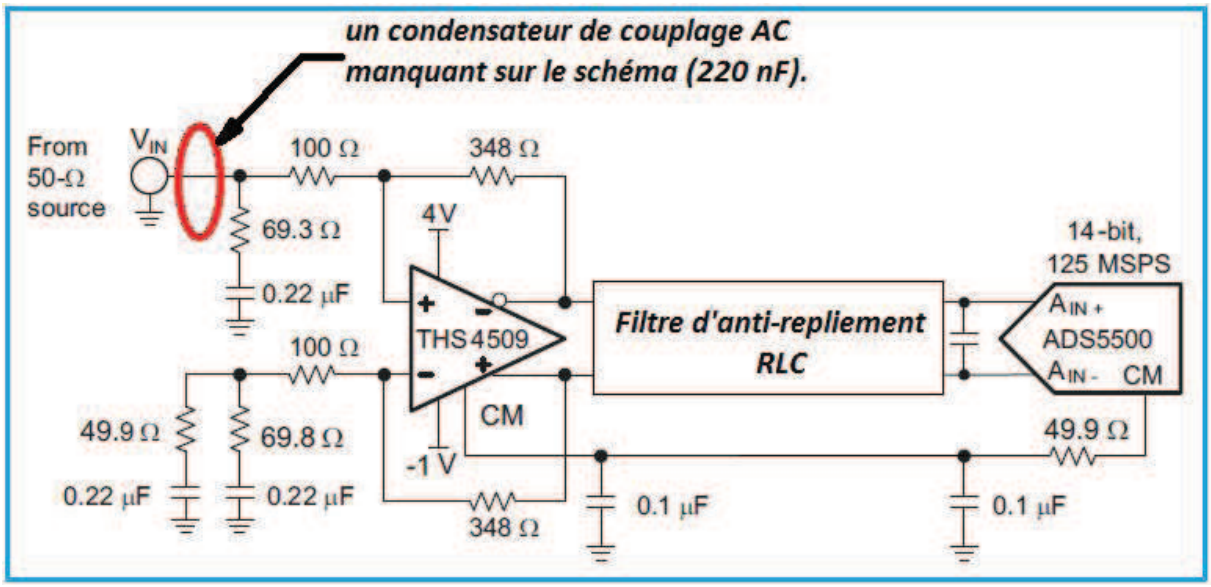


Figure I-5 : *Circuit électrique de l'entrée analogique du module P-240 (condensateur de couplage AC non montré)*

I.1.5.2 Filtre d'Anti-repliement

Le filtre d'anti-repliement placé entre la sortie différentielle de l'amplificateur *THS4509* et l'ADC *ADS5500* est un filtre passe-bas du 7ème ordre avec une réponse Chebyshev de 3dB. Avec les valeurs des composants apparaissant sur la figure (I-6) ; la fréquence de coupure à 3 dB est d'environ 45 MHz. Le filtre accomplie une atténuation de 60 dB à 100 MHz. L'utilisateur peut modifier la réponse du filtre passe-bas en changeant les valeurs des composants. A noter que l'impédance d'entrée de *1'ADS5500* est de 6,6 KOhms en parallèle d'une charge capacitive de 4pF.

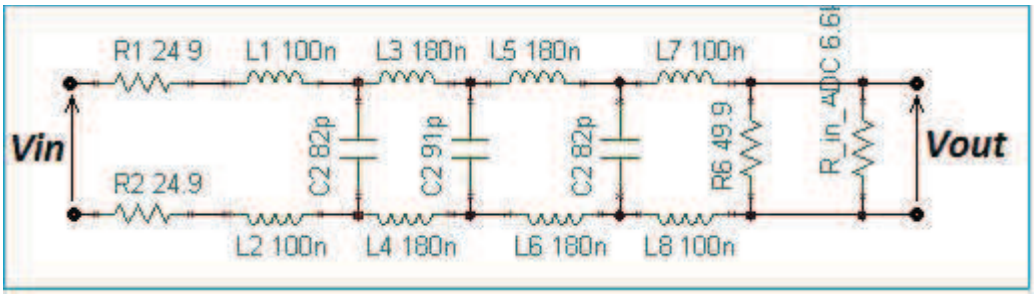


Figure I-6 : *Filtre Tchebychev d'Anti-repliement du 7ème ordre*

L'atténuation DC à travers le filtre anti-repliement est de 6 dB. Par conséquent, le gain total à partir de l'entrée analogique jusqu'à l'entrée de l'ADC est de 4 dB: 10 dB à travers l'amplificateur différentiel *THS4509*, 6 dB à travers le filtre d'anti-repliement, tout en supposant une source d'entrée du signal de 50 ohms.

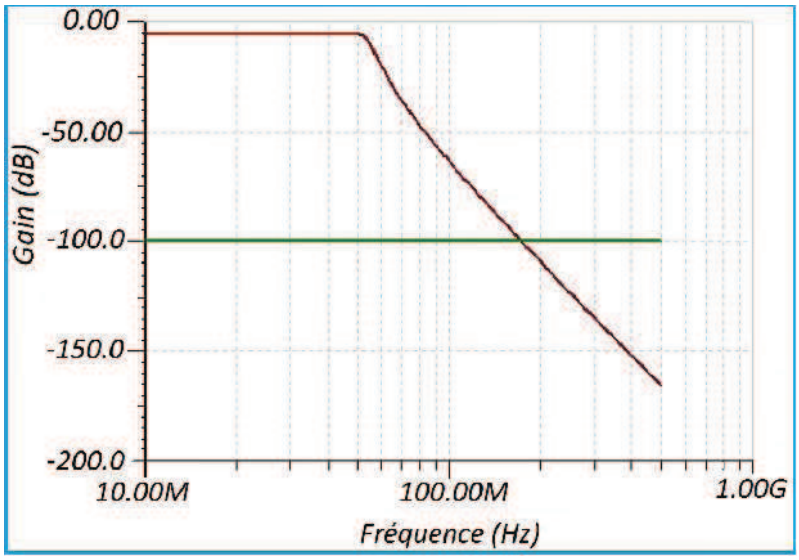


Figure I-7 *: Courbe de réponse du filtre Anti-repliement*

Dans le cas des applications sous-échantillonnées, les utilisateurs peuvent utiliser un filtre passe-bande externe. Dans ce cas, le filtre passe-bas, déjà présent sur la carte, peut être contourné en utilisant des connexions de résistances 0 Ohms prévus sur le module analogique P-240.

I.1.5.3 Convertisseur analogique-numérique (CAN) :

Le module P-240 analogue est muni de 02 convertisseurs analogique-numérique : *ADS5500* 14-bit, 125 MSPS de Texas Instruments.

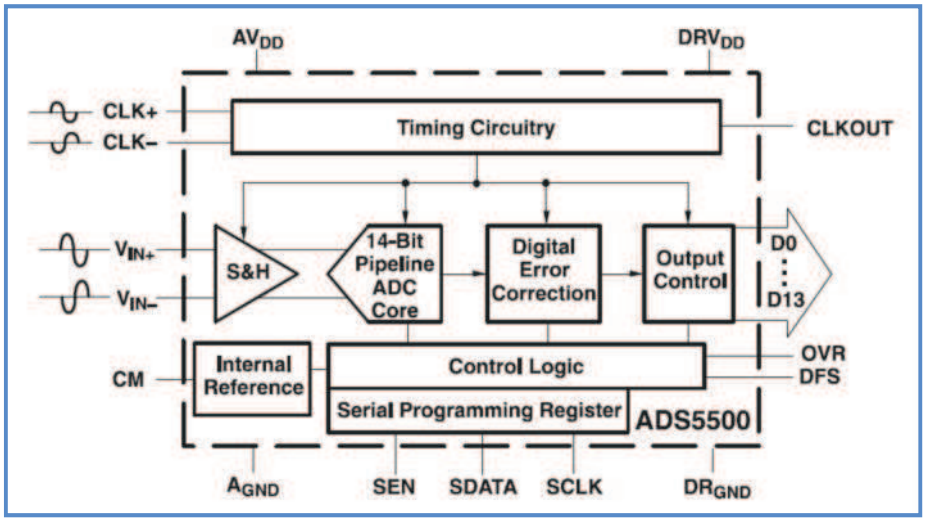


Figure I-8 : *Schéma bloc interne de l'ADC ADS5500*

La sortie de l'ADC sur les 14 bits: ***D13 (MSB) ... D0 (LSB)*** et les données sont de format complément à deux. Elles sont disponibles sur chaque front montant de l'horloge. Dans le cas du module analogique P-240, le format des données à la sortie de l'ADC est prédéfini par câblage hardware et ce en fixant la tension ***VDFS*** au niveau de la pin-DFS (40) de l'ADC *ADS5500*.

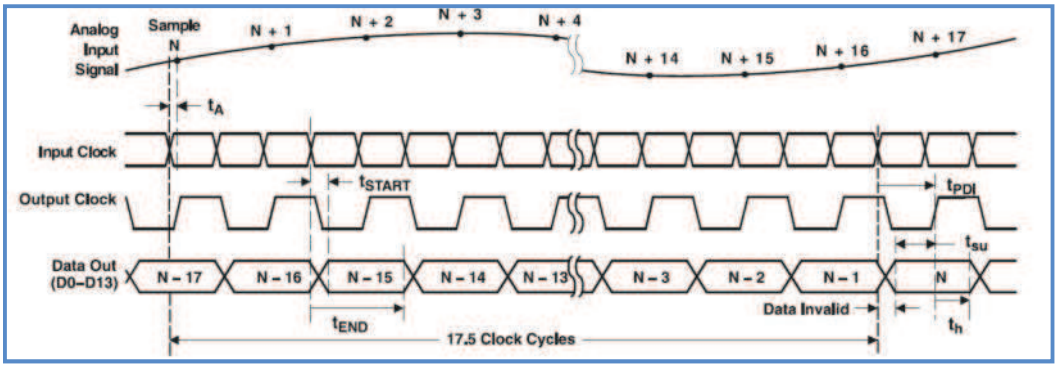


Figure I-9 *: Chronogramme de l'opération de conversion au sein de l'ADC ADS5500*

L'ADS5500 génère et transmet au circuit FPGA sa propre horloge (CLKOUT au niveau de la Pin-43). C'est une horloge synchronisée avec les données converties à la sortie et qui est utilisée en tant que horloge du domaine ADC au sein du circuit FPGA. En conséquence, aucune horloge distincte, générée le circuit tampon d'horloge CDCP1803 n'est acheminé vers le FPGA. Le signal de CLKOUT a la même fréquence que l'horloge d'entrée à l'ADC, et ce jusqu'à un maximum de 125 MHz.

I.1.6 Sorties analogiques :

La figure (I-10) montre la partie liée à la sortie analogique du module P-240.

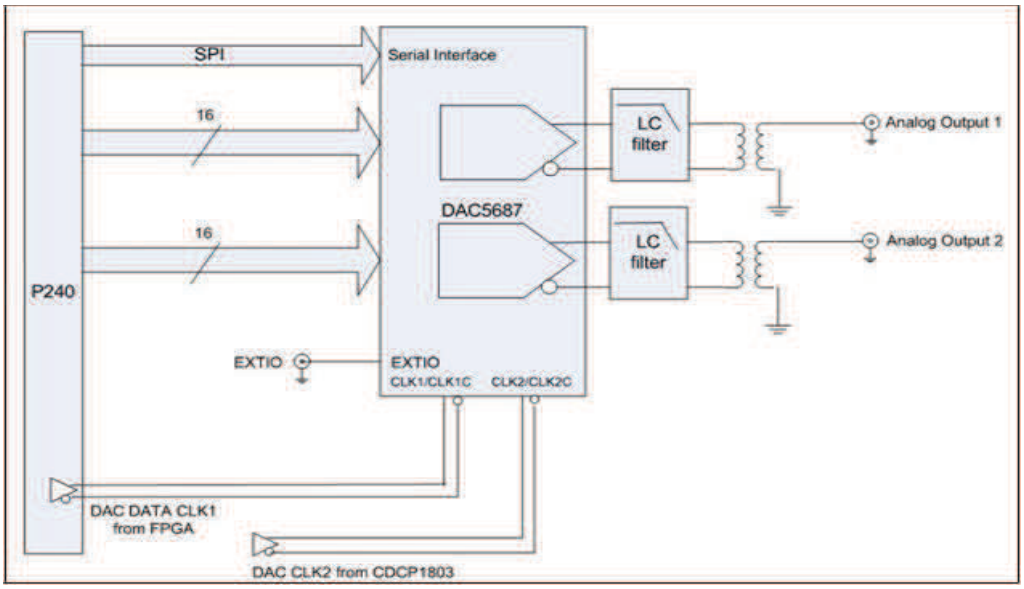


Figure I-10 : *Partie de la sortie analogique du module P-240*

I.1.6.1 Convertisseur numérique-analogique (CNA) :

Le module analogique P-240 inclut un convertisseur numérique-analogique à double canal.et à filtres d'interpolation 2x-8x : ***TI DAC5687*** de 16 bits, 500-Msps.

Le DAC5687 intègre des filtres numériques à interpolation 2x, 4x et 8x, un oscillateur à commande numérique (NCO), un multiplicateur d'horloge interne et une référence de tension sur puce (on-chip). Il a été spécialement conçu pour recevoir des données à faible taux d'échantillonnage en entrée, et un taux d'échantillonnage important des données à la sortie émanant du circuit FPGA.

L'envoi des données synchrones depuis la source vers le DAC à un taux inférieur minimise les erreurs de synchronisation qui peuvent surgir suite à une distorsion du signal quand celui-ci se propage depuis le circuit FPGA vers le DAC en passant à travers les connecteurs d'interfaçage du module P-240.

Le *DAC5687* peut être configuré via son interface série pour choisir des formats de données manipulables telles que : complément à deux ou décalage binaire : ***DA15 (MSB) ... DA0 (LSB) et DB15 (MSB) ... DB0 (LSB)***.

Le schéma bloc fonctionnel du *DAC5687* est illustré à la figure ci-après.

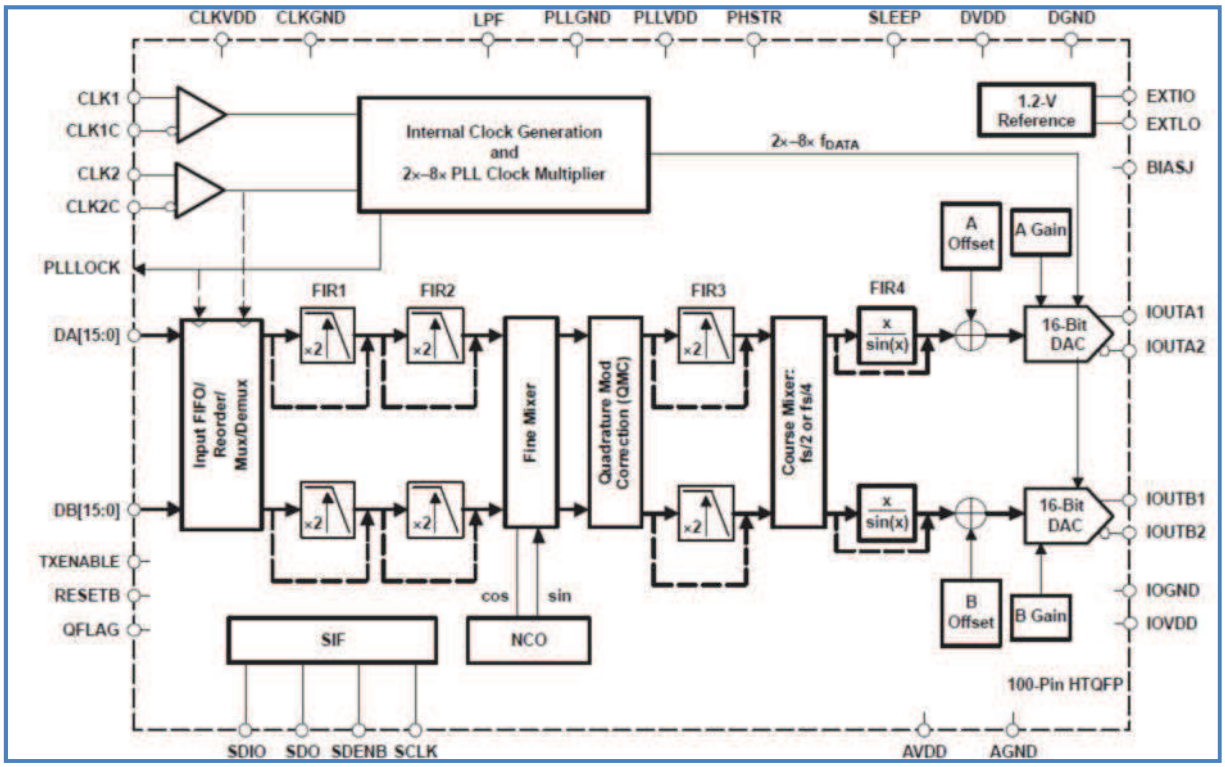


Figure I-11 *: Schéma bloc interne du DAC*

I.1.6.2 Filtre passe-bas de reconstruction analogique du DAC

Chaque sortie du DAC incorporé au module P-240 est dotée d'un filtre de reconstruction analogique passe-bas, de type Bessel et du 5ème ordre avec une fréquence de coupure à -3dB de 62,5 MHz. Cela représente un compromis entre la complexité du filtrage analogique, ondulation de la bande passante, et le rejet suffisant de l'image à 500 MHz, en supposant un taux d'échantillonnage de données à l'entrée du DAC de 125 Msps, avec interpolation 4x à 500 Msps avant la conversion analogique. L'atténuation du filtre est de 70 dB à 500 MHz. La réponse en phase est à peu près linéaire lorsqu'elle est tracée sur une échelle linéaire en cas de retard de groupe en plateau.

Les filtres de reconstruction du DAC peuvent être contournés en utilisant des connexions de résistances de « 0 Ohms » prévus sur le module analogique P-240. Un transformateur RF à rapport de transformation « 01:01 » est utilisé en aval du filtre de reconstruction sur chaque sortie analogique et ce pour piloter un câble de 50 ohms à terminaison double.

I.1.7 Architecture d'horloge

I.1.7.1 Le circuit Tampon d'horloge (CDCP180)

Les signaux d'horloge injectés aux : 2 ADCs (ADS5500) et CLK2 du double DAC5687 sont synchronisés à une source d'horloge commune, distribuée à travers le circuit tampon d'horloge ***CDCP1803*** doté d'une sortie en norme LVPECL (**L**ow-**V**oltage **P**ositive/**P**seudo **E**mitter-**C**oupled **L**ogic). Le CDCP1803 garanti une faible distorsion, ce qui minimise le bruit de phase dans les sorties des deux types de convertisseur de données présents sur le module.

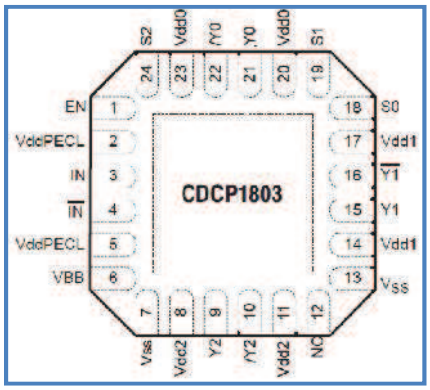


Figure I-12 : *Le circuit Tampon d'horloge.*

Le CDCP1803 est câblé sur le module analogique P240 de manière à fournir une horloge divisée par 4 à chacun des deux ADC : *ADS5500*, et une horloge divisée par 1 au convertisseur DAC *DAC5687* : CLK2. Le circuit tampon d'horloge *CDCP1803* est compatible avec tous les standards de signaux différentiels: LVDS, HSTL, CML, VML, SSTL-2 et single-ended: LVTTL / LVCMOS.

I.1.7.2 Système d'horloge de l'ADC

Chaque signal d'horloge divisé par 4 issu du circuit CDCP1803 est injecté (couplage : AC) à l'entrée d'horloge d'un *ADS5500* ADC comme étant un signal d'horloge différentiel, ce qui réduit la sensibilité au bruit en mode commun. Voir figure ci-dessous.

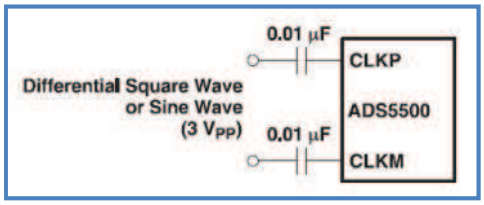


Figure I-13 : *Couplage AC, entrée différentielle de l'horloge pour l'ADC.*

Chaque convertisseur ADC : *ADS5500* génère et transmet au FPGA son propre signal d'horloge (CLKOUT, la broche 43 sur le circuit ADS5500) synchronisé avec les données de sortie en cours de conversion, qui peuvent être utilisés en tant que signal d'horloge au sein du circuit FPGA. Lorsque les deux *ADS5500* sont utilisés, CLKOUT de l'un des deux ADC peut être utilisé comme horloge interne au sein de l'FPGA. En conséquence, aucun autre signal d'horloge généré par le circuit tampon d'horloge CDCP1803 n'est acheminé vers le circuit FPGA. Le signal de CLKOUT émanant du circuit ADS5500 a une fréquence maximale de 125 MHz.

I.1.7.3 Système d'horloge du DAC

Le circuit DAC5687 dispose de 3 modes d'horloge. Ces trois modes sont mis en évidence via les registres internes du DAC5687 accessibles via le port d'interface série. Le mode préféré est le mode : Dual-Clock, dans lequel les données numériques accèdent au DAC d'une manière synchrone avec l'horloge CLK1, et sont envoyés à l'étage de sortie à une fréquence d'échantillonnage cadencée par CLK2 et ce avant l'opération finale de la conversion analogique. Le mode Dual-Clock peut utiliser le FIFO interne de circuit *DAC5687* pour passer du signal d'horloge CLK1 à CLK2. Avec le FIFO, aussi longtemps que CLK1 et CLK2 sont verrouillé en fréquence, les 2 signaux d'horloge peuvent être asynchrones. En pilotant CLK2 à partir d'une source d'horloge à faible distorsion, toute instabilité dans le signal CLK1 est sans rapport avec la première.

Le mode Dual-Clock est activée à travers les bits du registre PLLVDD = 0V et dual_clk=1. Le PLL/VCO interne du DAC5687 n'est pas recommandé pour une utilisation sur le module analogique P-240, car il se traduit généralement par un bruit de phase élevé comme si une horloge générée à l'extérieur est utilisée. En mode Dual-Clock, le taux d'échantillonnage du signal de sortie du DAC5687 est réglé par l'horloge à CLK2 (du CDCP1803) et le taux de transfert de données d'entrée par CLK1 du circuit FPGA.

I.1.7.4 Système d'horloge recommandé pour le module analogique P-240

Le système d'horloge recommandée pour le module analogique P240 utilise le FIFO interne du circuit *DAC5687* en mode Dual-Clock, une horloge différentielle à faible distorsion fournie sur CLK2/CLK2C au DAC à une fréquence maximale de 500 MHz. Les données envoyées au DAC sur 16 bits générées dans le circuit FPGA émanent du module : « Xilinx Virtex-4 MB baseboard » à travers les connecteurs du module P240 aux entrées DAC_DATA_A / B du DAC DAC5687, accompagnées par un signal d'horloge différentielle transmis aux entrées CLK1/CLK1C.

L'horloge d'entrée du DAC CLK1 est obtenue de l'un des deux signaux émanant des sorties d'horloge du circuit ADS5500 CLKOUT LVCMOS. Ces signaux sont acheminés vers le FPGA en compagnie des données générées à la sortie de l'ADC. N'importe quelle distorsion sur le signal d'horloge CLK1 du circuit DAC5687 n'est pas critique, car la mémoire FIFO du DAC est utilisée pour passer d'un CLK1 propre à un signal d'horloge CLK2 propre. Le *DAC5687* doit être configuré pour effectuer une interpolation 4X, car l'horloge d'échantillonnage *ADS5500* est une version divisée par 4 de l'horloge d'entrée dans la mémoire du circuit tampon d'horloge ***CDCP1803***.

Le système d'horloge recommandée est illustré sur la figue ci-dessous avec une horloge différentielle à faible distorsion connectée aux 02 connecteurs EXT via des fiches SMA sur le module analogique P-240. Afin de minimiser la distorsion, ce signal d'horloge différentiel n'est pas acheminé à travers le circuit FPGA.

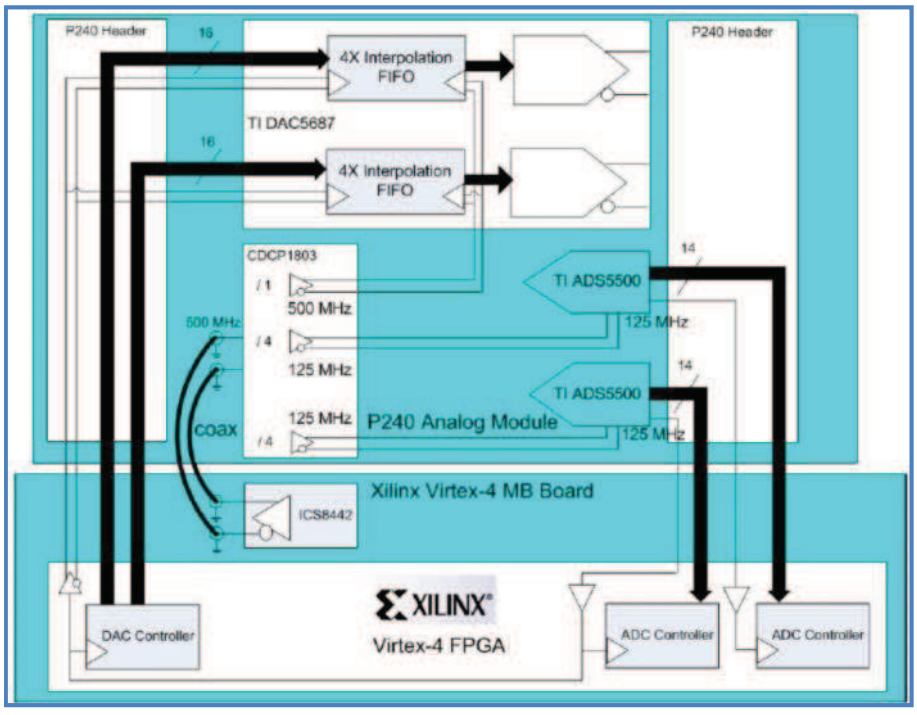


Figure I-14 *: Système d'horloge recommandé*

I.1.7.5 Source du signal d'horloge pour le module analogique P-240

Le module analogique P-240 prend son signal d'horloge à travers les 02 connecteurs SMA femelles : **Clock EXT**, qui présente une charge de 50 du transformateur de couplage à la source. Afin de préserver les performances spécifiées du convertisseur *DAC5687*, la source d'horloge

devrait avoir une très faible distorsion. En utilisant une horloge avec un rapport cyclique de 50% on aura une performance dynamique optimale.

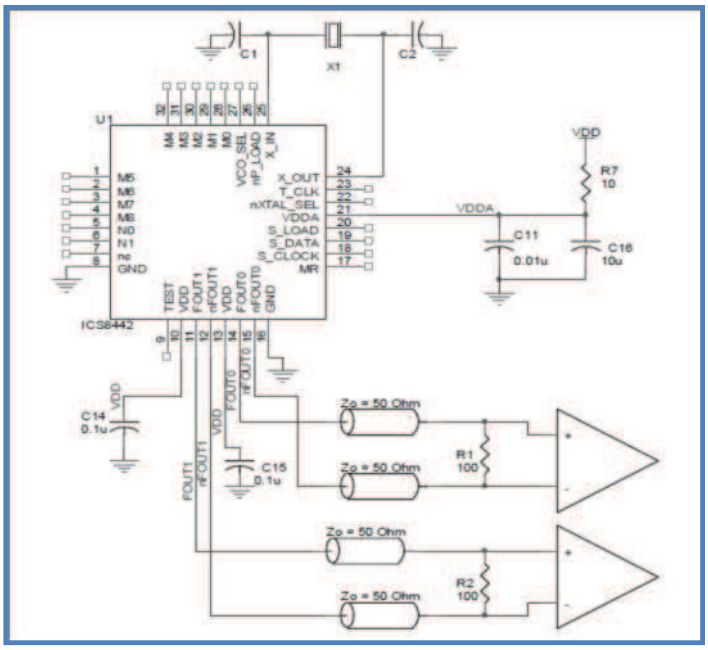


Figure I-15 : *Circuit générateur d'horloge à base de « ICS8482 » sur le Kit FPGA.*

Un signal d'horloge externe typique est de 300 mV, de forme sinusoïdale et de 0V d'offset, ou signal carré à rapport cyclique de 50%, appliquée au connecteur SMA EXT CLKIN, couplé au transformateur d'entrée pour régler la tension du mode commun correctement pour le pilote d'horloge : ***CDCP1803***. Des Plots de soudure sont fournis pour contourner le transformateur d'adaptation avec des résistances 0 Ohm, si désiré.

La carte « Xilinx Virtex-4 MB baseboard » possède un synthétiseur d'horloge programmable avec sortie différentielle LVDS connectée aux connecteurs SMA : ***ICS8442***, présentés dans la figure ci-dessus. C'est une excellente source d'horloge adaptée pour le module analogique P-240.

I.1.8 Alimentation électrique

L'alimentation est fournie au module analogique P-240 via +5 V et +3.3 V fournis par la carte FPGA à travers les connecteurs JX1 et JX2. Plusieurs convertisseurs DC-DC et régulateurs de tension génèrent les lignes de tension appropriées pour les différents périphériques du module.

I.1.9 Conclusion

A travers l'étude menée, nous avons pu dévoiler les concepts de bases qui se cachent derrière un bon fonctionnement du module adressé lors de ce rapport. Le savoir acquis qui a été exploité avec succès pour faire démarrer le système (interface d'acquisition de données + carte FPGA).

En supprimant le condensateur de couplage (AC) à l'entrée du convertisseur (non mentionnées sur les schémas électriques fournis par le constructeur de l'équipement), nous avons pu acquérir des données DC (format complément à deux) sur FPGA et qu'on a d'ailleurs affiché sur l'afficheur LCD disponible dur le kit de développement FPGA.

On note par cette fin, que le transfert de données numérisées entre le convertisseur ADC et le circuit FPGA se fait en mode rafale sans signaux supplémentaires de hand-shaking. Ce qui veut dire qu'à chaque impulsion d'horloge, on a une nouvelle donnée au niveau du circuit du traitement (FPGA). Une prouesse technologique qui peut être exploitée avec sucées lors du traitement des impulsions nucléaires via l'équipement en cours d'étude.

I.2 PRESENTATION DES CIRCUITS FPGAS

I.2.1 Introduction

Les progrès technologiques continus dans le domaine des circuits intégrés ont permis la réduction des coûts, de la consommation, et c'est maintenant un lieu commun d'affirmer que les circuits intégrés spécifiques d'une application ont permis une réduction de la taille des systèmes numériques ainsi que la réalisation de circuits de plus en plus complexes, tout en améliorant leurs performances et leur fiabilité. Aujourd'hui les techniques de traitement numérique occupent une place majeure dans tous les systèmes électroniques modernes grand public, professionnels ou de défense.

De plus, les techniques de réalisation de circuits spécifiques, tant dans les aspects matériels (composants reprogrammables, circuits pré caractérisés et bibliothèques de macro fonctions) que dans les aspects logiciels (placement routage, synthèse logique) font désormais de la microélectronique une des bases indispensables pour la réalisation de systèmes numériques performants. Elle impose néanmoins une méthodologie de développement en CAO (Conception Assistée par Ordinateur) très structurée.

I.2.2 Les ASICs (Application Specific Integrated Circuit) [2]

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

Les ASIC peuvent être classés en plusieurs catégories selon leur niveau d'intégration, en fait un ASIC est défini par sa structure de base (réseau programmable, cellule de base, matrice, etc.). Sous le terme ASIC deux familles sont regroupées, les semi personnalisés (avec des réseaux prédéfinis) et les personnalisés (non préfabriqué, qu'on optimise pour créer son propre composant).

I.2.2.1 Avantages et inconvénients de l'utilisation d'ASIC [2]

D'une manière générale l'utilisation d'un ASIC conduit à de nombreux avantages provenant essentiellement de la réduction de la taille des systèmes. Il en ressort :

- Réduction du nombre des composants sur le circuit imprimé. La consommation et l'encombrement s'en trouvent considérablement réduits. ·
- Le concept ASIC par définition assure une optimisation maximale du circuit à réaliser. Nous disposons alors d'un circuit intégré correspondant réellement à nos propres besoins.
- La personnalisation du circuit donne une confidentialité au concepteur et une protection industrielle.
- Enfin, ce type de composant augmente la complexité du circuit, sa vitesse de fonctionnement et sa fiabilité.

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

Les ASICs sont généralement plus convenables pour les productions en série de conceptions déjà vérifiées et non pour les prototypes.

I.2.3 Les PLD (Programmable Logic Device) [2]

Ce sont des chips qui peuvent être programmés pour se comporter comme une conception arbitraire. Un PLD peut être programmé pour une implémentation aussi simple qu'une opération en logique combinatoire comme il pourrait l'être pour des conceptions beaucoup plus importantes.

I.2.4 Les FPGAs

Les FPGAs (Field Programmable Gate Arrays ou "réseaux logiques programmables") sont des composants VLSI (Very Large Scale Integration) de type PLD entièrement reconfigurables ce qui permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs. L'avantage de ce genre de circuit est sa grande souplesse qui permet de les réutiliser à volonté dans des algorithmes différents dans un temps très court. Le progrès de ces technologies permet de faire des composants toujours plus rapides et à plus haute intégration, ce qui permet de programmer des applications importantes.

Un FPGA possède une architecture générique qui consiste en un ensemble de blocs logiques et séquentiels configurables et d'un ensemble d'interconnections programmables.

Les circuits FPGAs ne sont pas optimisés pour une application bien déterminée, par conséquent ils consomment plus d'énergie que les ASICs. Par contre ils sont beaucoup plus simples à programmer et à reprogrammer, ce qui raccourcit les cycles de conception et permet de suivre l'évolution de l'application pour laquelle, ils ont été conçus.

Les FPGAs sont plus convenables pour les prototypes et pour les productions en série limitées qui ne sont pas de la qualité des ASICs.

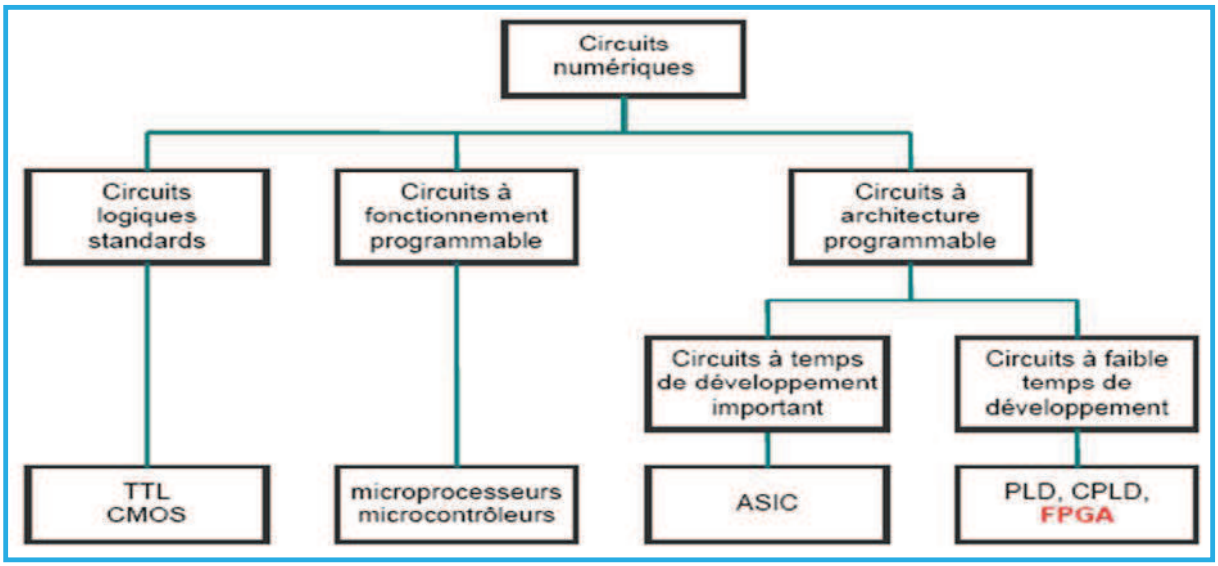


Figure I.16: *Classification des circuits numériques*

Les FPGA sont utilisés dans de nombreuses applications, on en cite dans ce qui suit quelques-unes :

- prototypage de nouveaux circuits,
- fabrication de composants spéciaux en petite série,
- adaptation aux besoins rencontrés lors de l'utilisation,
- systèmes de commande à temps réel,
- DSP (Digital Signal Processing),
- imagerie médicale.

I.2.4.1 Architecture interne des circuits FPGAs [3]

Les circuits FPGA possèdent une structure matricielle de deux types de blocs (ou cellules). Des blocs d'entrées/sorties et des blocs logiques programmables.

Le passage d'un bloc logique à un autre se fait par un routage programmable, Certains circuits FPGA intègrent également des mémoires RAM, des multiplieurs et même des noyaux de processeurs.

Actuellement deux fabricants mondiaux se disputent le marché mondial des FPGA : Xilinx et Altera.

De nombreux autres fabricants, de moindre envergure, proposent également leurs propres produits avec des technologies et des principes organisationnels différents. Dans ce qui suit, on va faire une description de l'architecture utilisée par Xilinx, car c'est sur des circuits Xilinx que

Nous allons va implémenter notre programme. L'architecture retenue par Xilinx se présente sous forme de deux couches :

- une couche appelée circuit configurable.
- une couche réseau mémoire SRAM (Static Random Access Memory).

La couche dite (circuit configurable) est constituée d'une matrice de blocs logiques configurables (CLB) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles.

Tout autour de ces blocs logiques configurables, nous trouvons des blocs d'entrées/sorties (IOB) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs (Figure I.17).

La programmation du circuit FPGA, appelé aussi LCA (Logic Cells Arrays), consistera en l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ servant à

interconnecter les éléments des CLB et des IOB, afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux.

Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM.

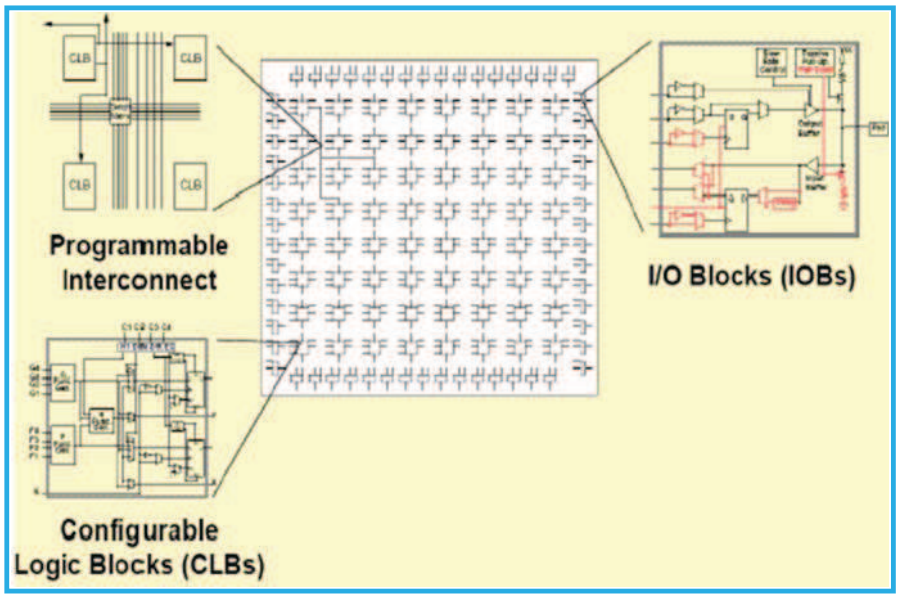


Figure I.17: *Architecture interne du FPGA*

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 à partir de la ROM.

Ainsi on conçoit aisément qu'un même circuit puisse être exploité successivement avec des ROMs 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.

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 (LUT) 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 LUT (Look Up Table) est un élément qui dispose de quatre entrées, il existe donc 24 = 16 combinaisons différentes de ces entrées.

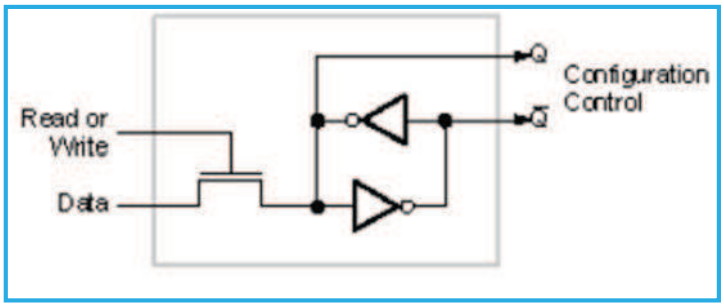


Figure I.18: *Structure d'une cellule SRAM*

L'idée consiste à mémoriser la sortie correspondant à chaque combinaison d'entrée dans une petite table de 16 bits, la LUT devient ainsi un petit bloc générateur de fonctions.

La (Figure I.19) ci-dessous montre le schéma simplifié d'un CLB de la famille XC4000 de Xilinx.

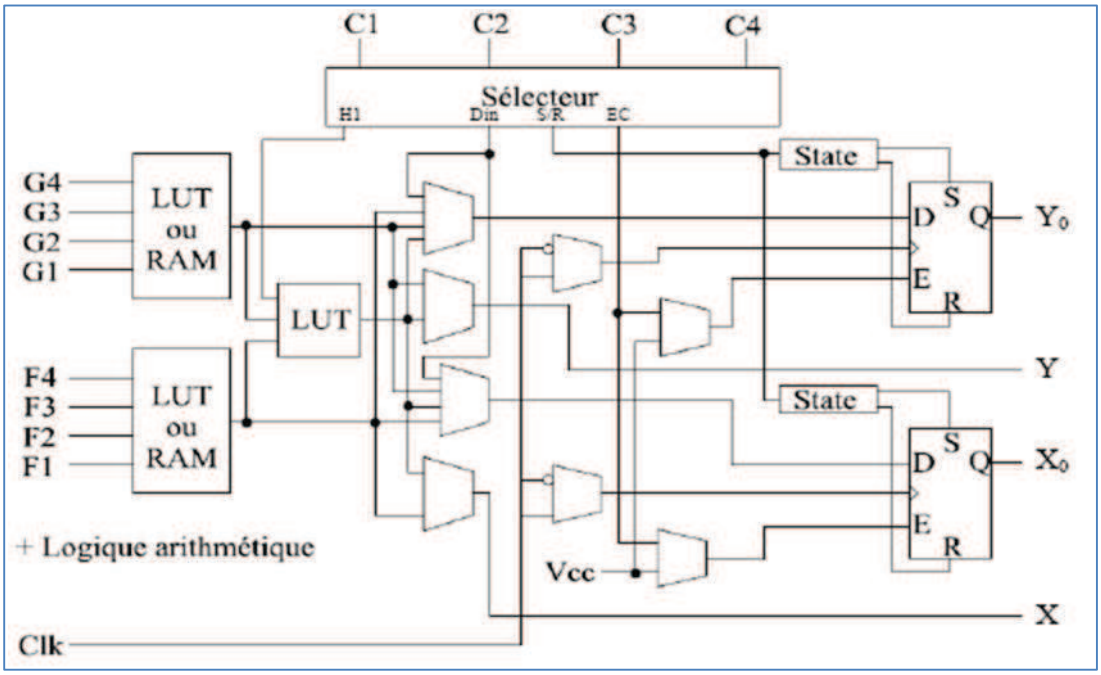


Figure I.19: *Schéma d'une cellule logique (XC4000 de Xilinx)*

I.2.4.2 Les IOB (Input Output Bloc) [4]

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 I.20) présente la structure de ces blocs.

a) Configuration en entrée

Premièrement, le signal d'entrée traverse un buffer qui, selon sa programmation, peut détecter soit des seuils TTL soit des seuils CMOS. Il peut être routé directement sur une entrée directe de la logique du circuit FPGA ou sur une entrée synchronisée.

Cette synchronisation est réalisée à l'aide d'une bascule de type D, le changement d'état peut se faire sur un front montant ou descendant.

De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (program controlled multiplexer). Un bit positionné dans une case mémoire commande ce dernier.

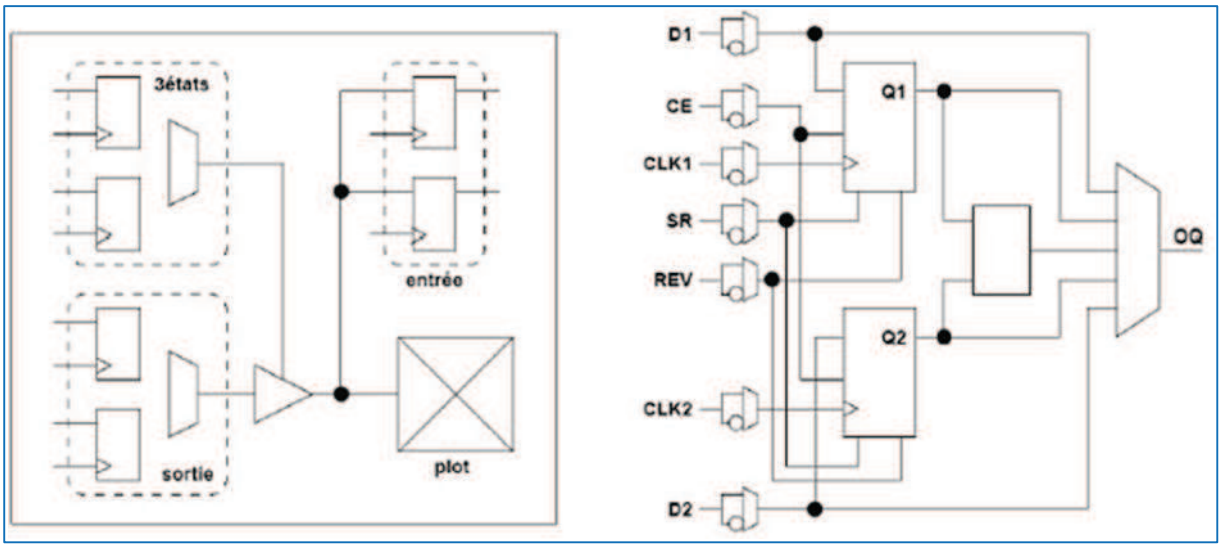


Figure I.20 : *Schéma d'un bloc d'entrée/sortie (IOB)*

b) Configuration en sortie

Nous distinguons les possibilités suivantes :

- inversion ou non du signal avant son application à l'IOB,
- synchronisation du signal sur des fronts montants ou descendants d'horloge,
- signaux en logique trois états ou deux états, le contrôle de mise en haute impédance et la réalisation des lignes bidirectionnelles sont commandés par le signal de commande (Out Enable), lequel peut être inversé ou non.

Chaque sortie peut délivrer un courant de 12mA.

Ainsi toutes ces possibilités permettent au concepteur de connecter au mieux une architecture avec les périphériques extérieurs.

I.2.4.3 Les différents types d'interconnexion [3]

Les connexions internes dans les circuits FPGA sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB.

Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM (Random Access Memory).

Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les blocs d'entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible.

Pour parvenir à cet objectif, Xilinx propose trois sortes d'interconnexion selon la longueur et la destination des liaisons. Nous disposons :

- D'interconnexions à usage général,
- D'interconnexions directes,
- de longues lignes.

a) Les interconnexions à usage général

Des aiguilleurs appelés aussi (switch matrix) sont situés à chaque intersection.

Leur rôle est de raccorder les segments entre eux selon diverses configurations, ils assurent ainsi la communication des signaux d'une voie vers l'autre.

Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre CLB.

Pour éviter que les signaux traversant les grandes lignes ne soient affaiblis, nous trouvons généralement des buffers implantés en haut et à droite de chaque matrice de commutation (Figure I.21).

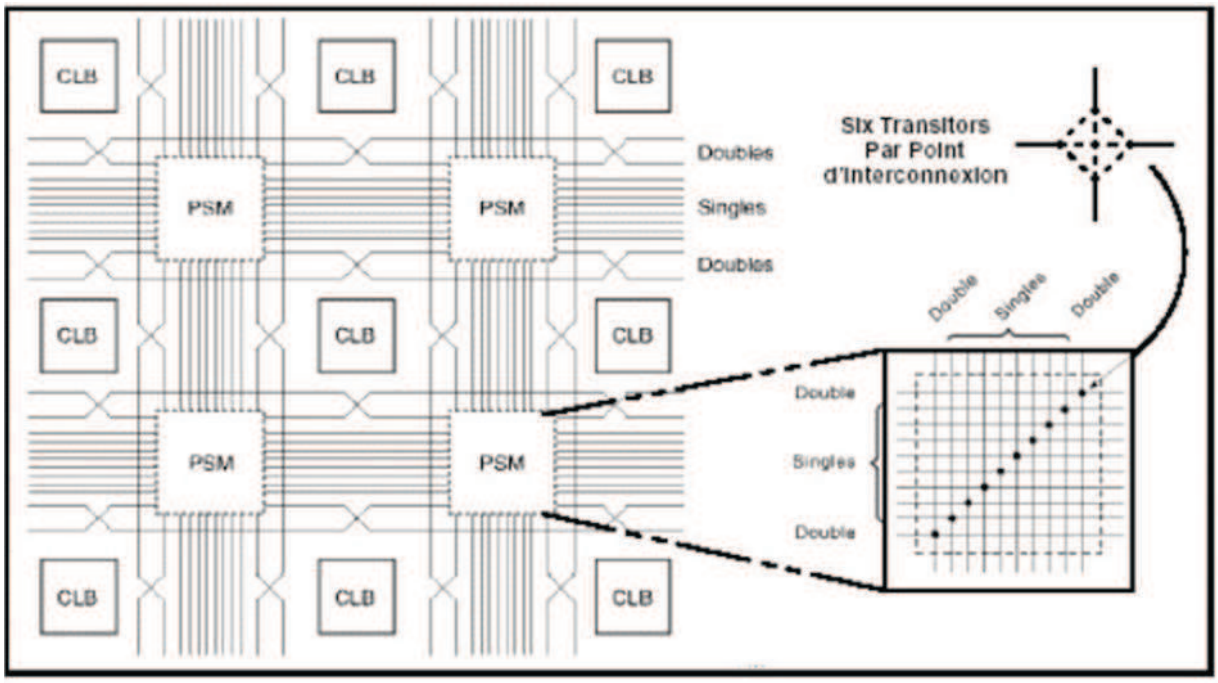


Figure I.21: *Connexions à usage général et détail d'une matrice*

b) Les interconnexions directes :

Ces interconnexions permettent l'établissement de liaisons entre les CLB et les IOB avec un maximum d'efficacité en termes de vitesse et d'occupation du circuit.

De plus, il est possible de connecter directement certaines entrées d'un CLB aux sorties d'un autre (Figure I.22).

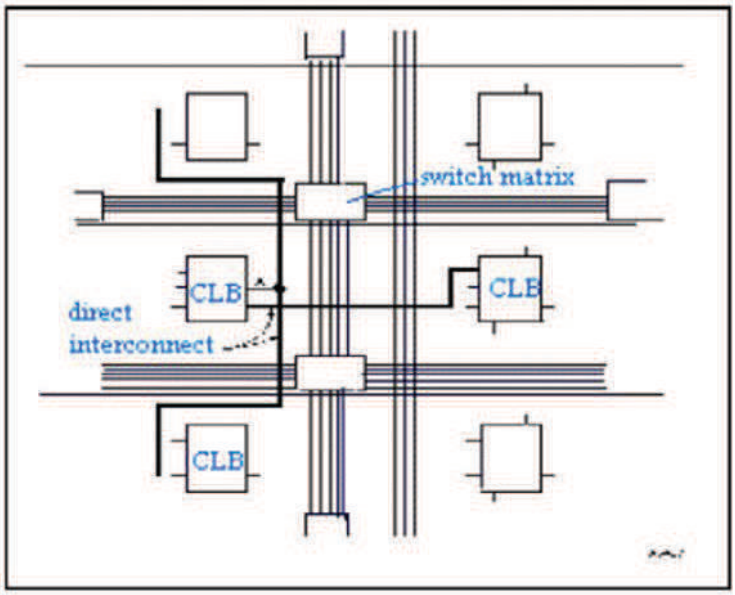


Figure I.22 : *Les interconnexions directes*

c) Les longues lignes

Les longues lignes sont de longs segments métallisés parcourant toute la longueur et la largeur du composant, elles permettent éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments dans le but d'assurer un synchronisme aussi parfait que possible.

De plus, ces longues lignes permettent d'éviter la multiplicité des points d'interconnexion (Figure I.23).

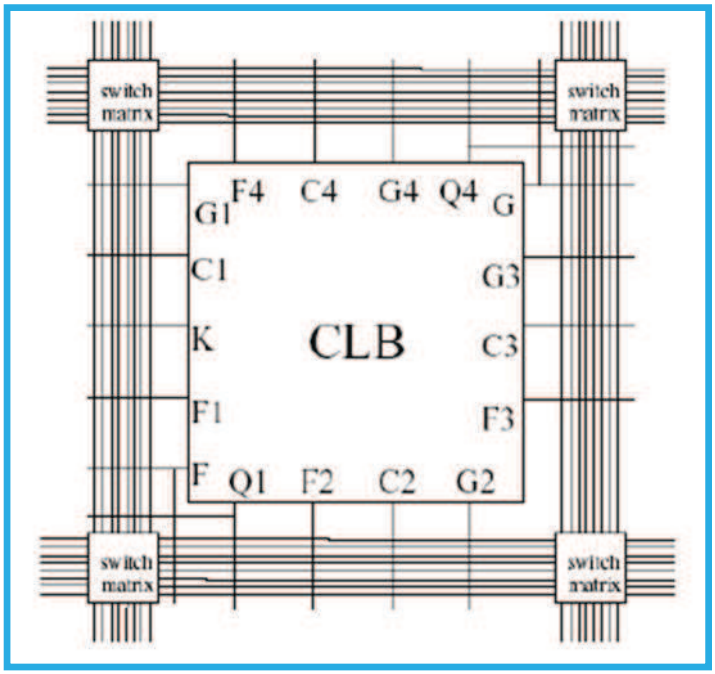


Figure I.23: *Les longues lignes*

I.2.5 Notre choix

Dans le cas de notre projet de fin d'étude, on va utiliser une carte FPGA du constructeur Xilinx, implanté sur une carte virtex4.

I.2.6 Description du kit de développement Virtex-4 MB [4]

I.2.6.1 Vue d'ensemble

Le kit de développement de Memec Virtex-4™ MB fournit une plate-forme complète de développement pour concevoir et vérifier des applications basées sur la famille de Xilinx Virtex-4 FPGA. Ce kit permet également aux concepteurs de mettre en œuvre des applications DSP tout une flexibilité extrême via l'utilisation des noyaux d'IP et des modules adaptés aux besoins de l'utilisateur. Le circuit FPGA Virtex-4, muni d'un noyau de processeur de Xilinx/MicroBlaze, est très utile pour les applications basées sur le prototypage des processeurs, permettant ainsi aux équipes de conception du logiciel d'implémentation d'accéder à une plate-forme de matériel avant d'arriver aux produits finis/cible.

I.2.6.2 le kit de système du Virtex-4 MB

Le Kit de développement 'Virtex-4MB' (figure I.24) utilise le circuit FPGA : Xilinx XC4VSX35-10FF668C. Il inclut également 64MB de RAM "la DDR SDRAM", 4MB de mémoire flash, LVDS de 16 bits qui transmettent et reçoivent des données sur des ports prévus à cet effet. On trouve aussi, une source programmable d'horloge de LVDS, un port USB-RS232 et des circuits additionnels d'assistance aux utilisateurs pour développer un système complet. Le Kit supporte également la norme de module d'expansion de Memec P240, permettant aux modules spécifiques à l'application d'expansion d'être facilement ajoutés.

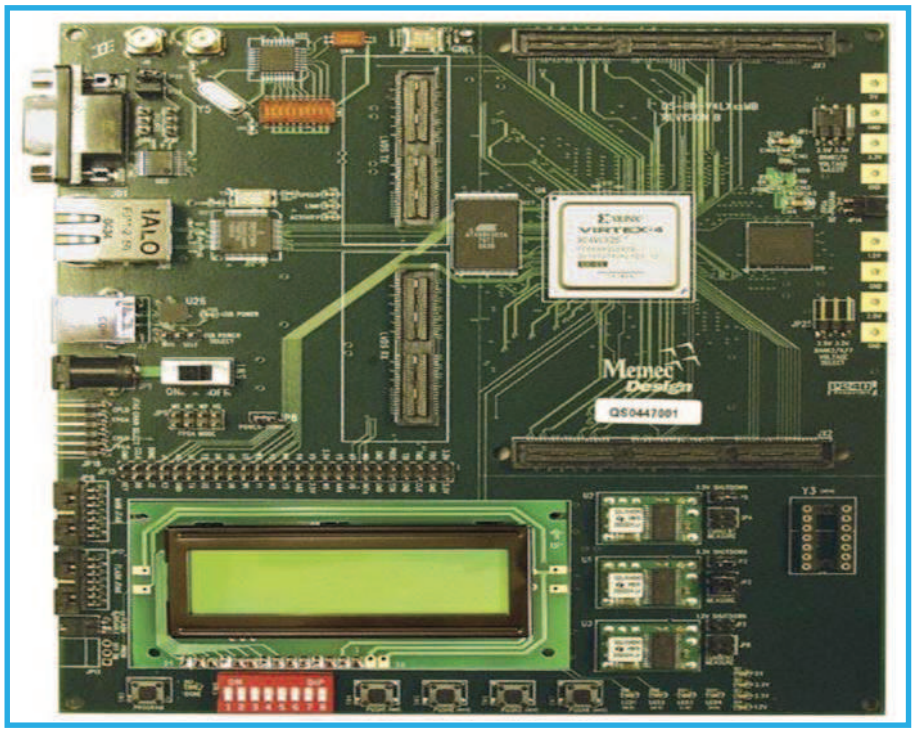


Figure I.24: *Le kit de développement Virtex 4*

I.2.6.3 Description fonctionnelle

Un schéma fonctionnel à haut niveau de la plate-forme de développement de Virtex-4™ MB est montré ci-dessous (figure I.25). Il est suivi d'une courte description de chaque sous-section. Une liste de dispositifs constituant ce module est énumérée ci-dessous :

- Circuit FPGA : Xilinx XC4SX35-10FF668,
- 64MB de mémoire principale sous forme : DDR SDRAM,
- 4MB de mémoire flash,
- Des entités LVDS de 16 bits transmettant et recevant les données via des interfaces,
- 10/100 Ethernet PHY,
- Source programmable d'horloge de LVDS (25-700 mégahertz),
- Sorties d'horloge de l'utilisateur LVDS par l'intermédiaire des connecteurs différentiels de SMA,
- Oscillateur à bord de 100MHz LVTTL,
- Douille à bord d'oscillateur de LVTTL (oscillateurs 4/8-Pin),
- Connecteurs P240,
- Panneau d'afficheur LCD, 16 x 02,
- Mémoire flash 32MB pour la configuration de FPGA,
- Connecteurs JTAG programmant/port de configuration,
- Connecteur de module de System de norme ACE™,
- Port RS-232,
- Quatre LEDs utilisateurs,
- Quatre commutateurs " bouton poussoir" réservés pour l'utilisateur,
- Des switches à positions multiples de 8 positions chacun,
- Pont de commutation entre USB-RS232

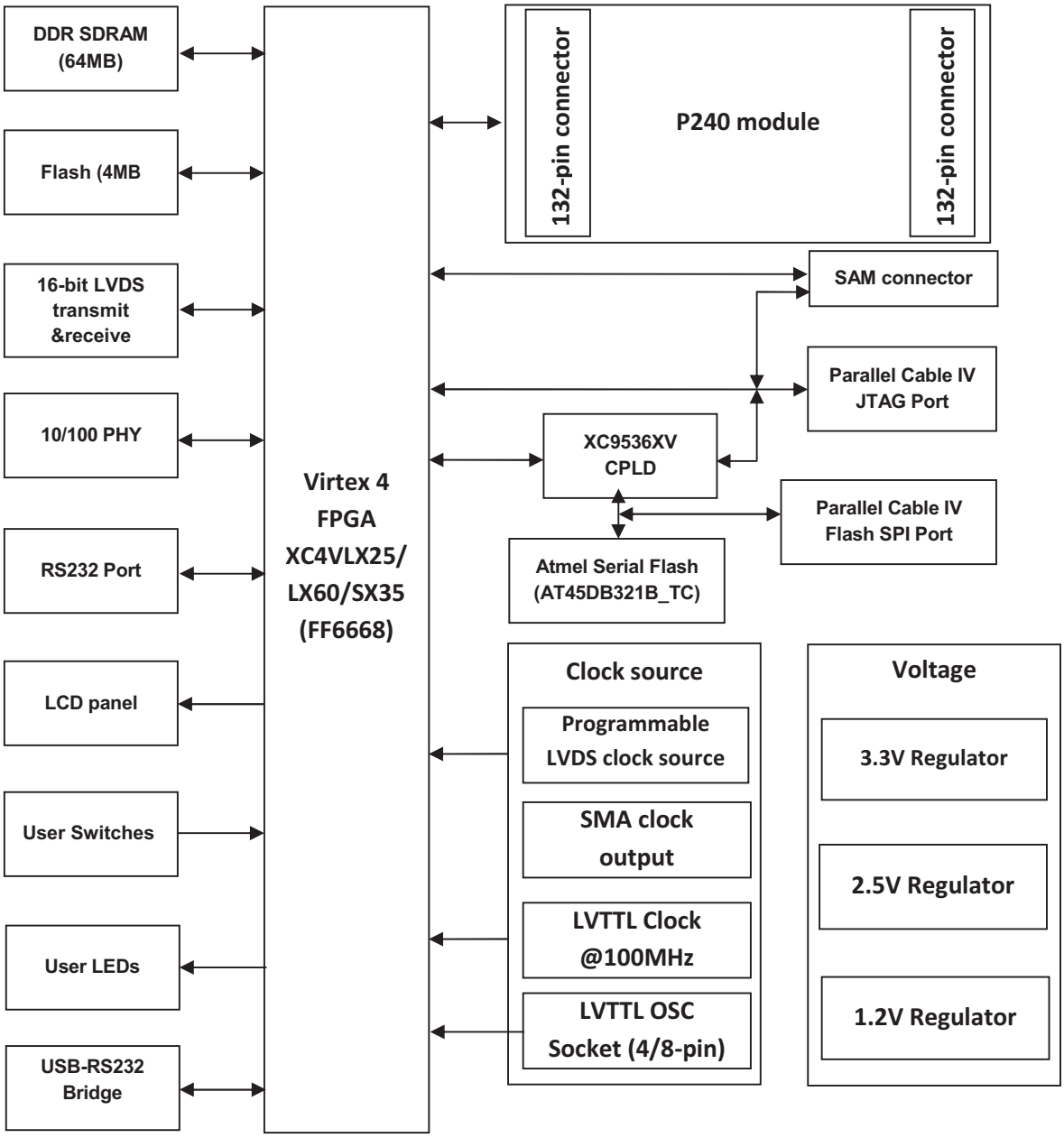


Figure I.25 : *Schéma fonctionnel de la plate-forme de développement du Virtex-4 MB*

a) Interface de LVDS

Le kit de développement du Virtex-4mb fournit des connecteurs à grande vitesse de LVDS soutenant une interface SPI-4.2. Cette interface se compose de 36 paires de signaux de type LVDS (72 signaux de FPGA) et 6 signaux asymétriques. En plus de l'interface SPI-4.2, l'interface LVDS est conçue pour supporter une interface : XSBI LVDS de 16 bits @644Mbps pour soutenir une interface 10GbE sur la plate-forme de développement du Virtex-4 MB. Les sections suivantes fournissent une courte description de l'interface de LVDS sur ce kit de développement.

b) Interface SPI-4.2

Le kit de développement du Virtex-4 MB fournit un SPI-4.2 par l'intermédiaire d'une interface électrique de 16 bits LVDS parallèle. La figure I.26 montre l'interface SPI-4.2 sur le kit. La transmission et la réception des signaux via l'interface du SPI-4.2 sont mises en application l'utilisation des signaux de LVDS tandis que les signaux de contrôle de flux de statut sont mis en application par l'utilisation des signaux asymétriques de LVTTL.

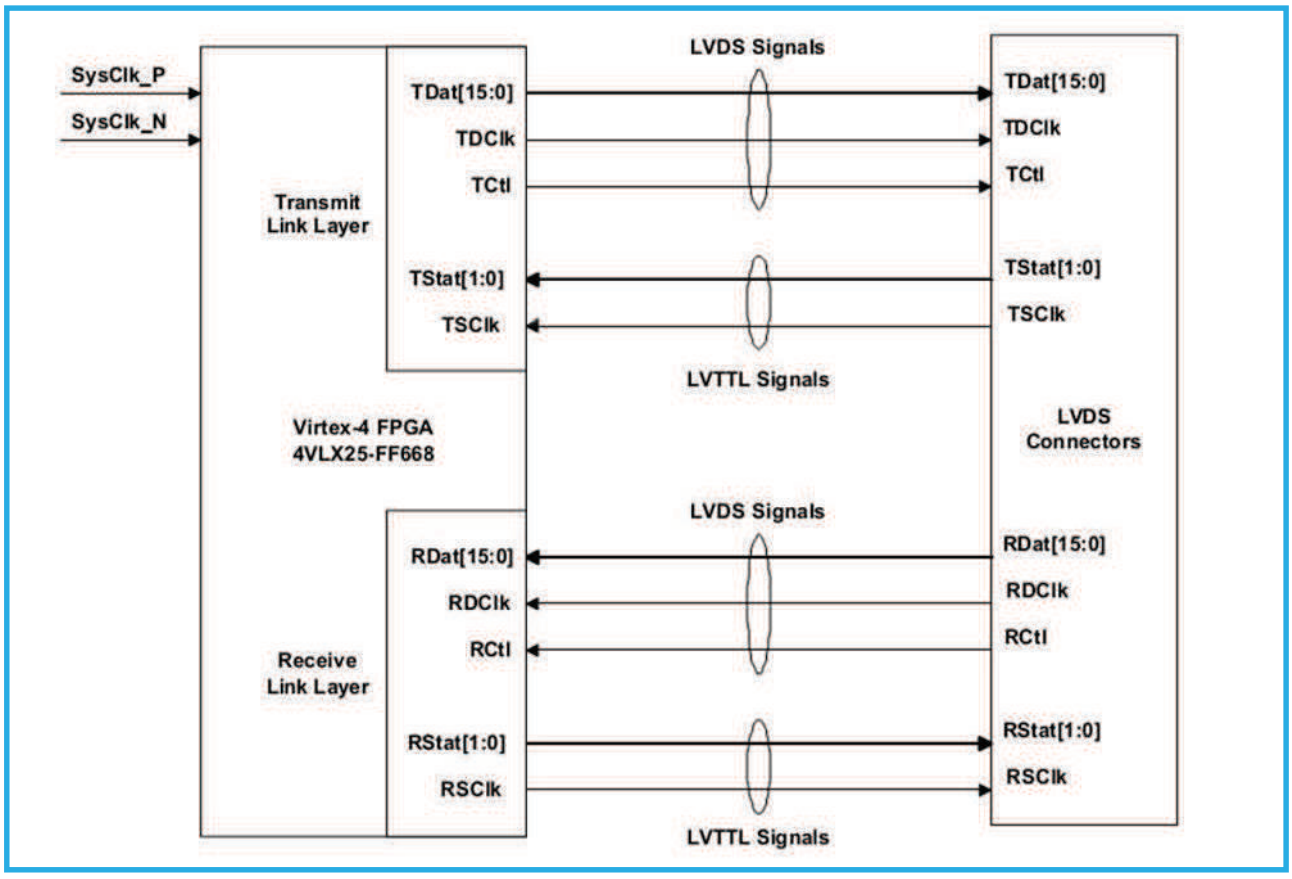


Figure I.26 : *L'interface SPI-4.2*

c) DDR SDRAM

Le kit de développement de Virtex-4™ MB inclus aussi 64MB de la mémoire de la DDR SDRAM. Un schéma fonctionnel de cet interface est montré ci-dessous (figure I.27).

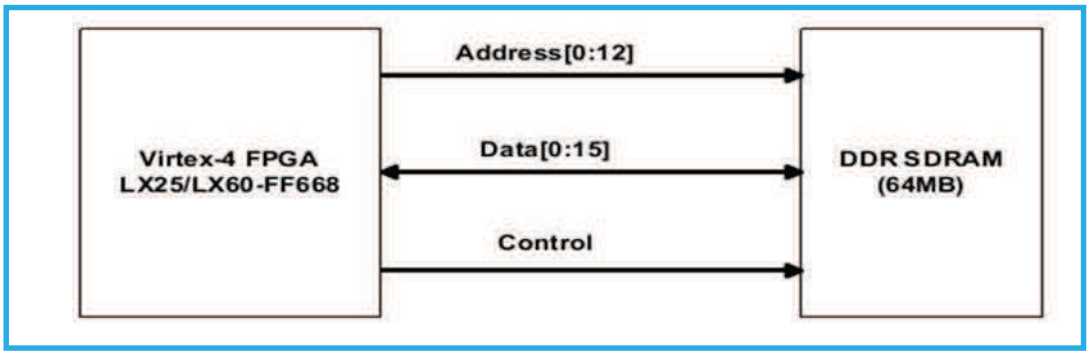


Figure I.27 : *Interface DDR SDRAM*

d) Mémoire Flash :

Le kit de développement de Virtex-4™ mb renferme 4MB de la mémoire Flash (x16). Un schéma fonctionnel à niveau élevé de l'interface est montré ci-dessous (figure I.28).

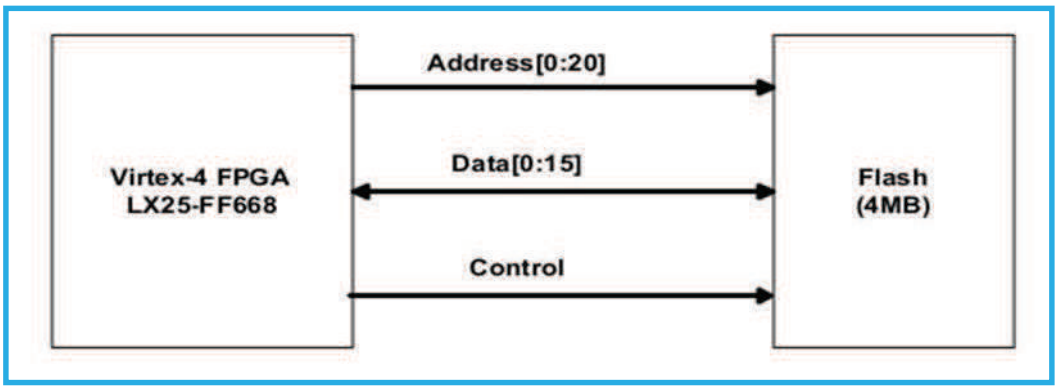


Figure I.28 : *Interface du flash*

e) Sources d'horloge :

Le circuit de génération d'horloge du kit du Virtex-4 MB fournit toutes les horloges nécessaires pour un processeur de Micro Blaze, les unités d'E/S Situées sur le kit, aussi bien que la mémoire de la DDR SDRAM. Généralement les sources d'horloge sur le kit sont groupées dans deux catégories ; sources différentielles et asymétriques d'horloge. Les sources d'horloge différentielle sont principalement employées par l'interface de LVDS, alors que les sources asymétriques d'horloge sont employées par la section de processeur.

Un oscillateur 100MHz à bord fournit les signaux d'horloge de système à la section de processeur. Cette horloge 100Mhz sera employée par tous les périphériques du Virtex-4 à lecture digitale pour produire de diverses signaux horloges de la carte. En plus des entrées d'horloge ci-dessus, une douille de connexion est fournie sur le kit et peut être employée pour fournir des signaux d'horloge de norme LVTTL au circuit FPGA par l'intermédiaire d'un circuit oscillateur de 8 ou 4 pins. La figure (I.29) montre les ressources d'horloge sur le kit de développement du Virtex-4 MB.

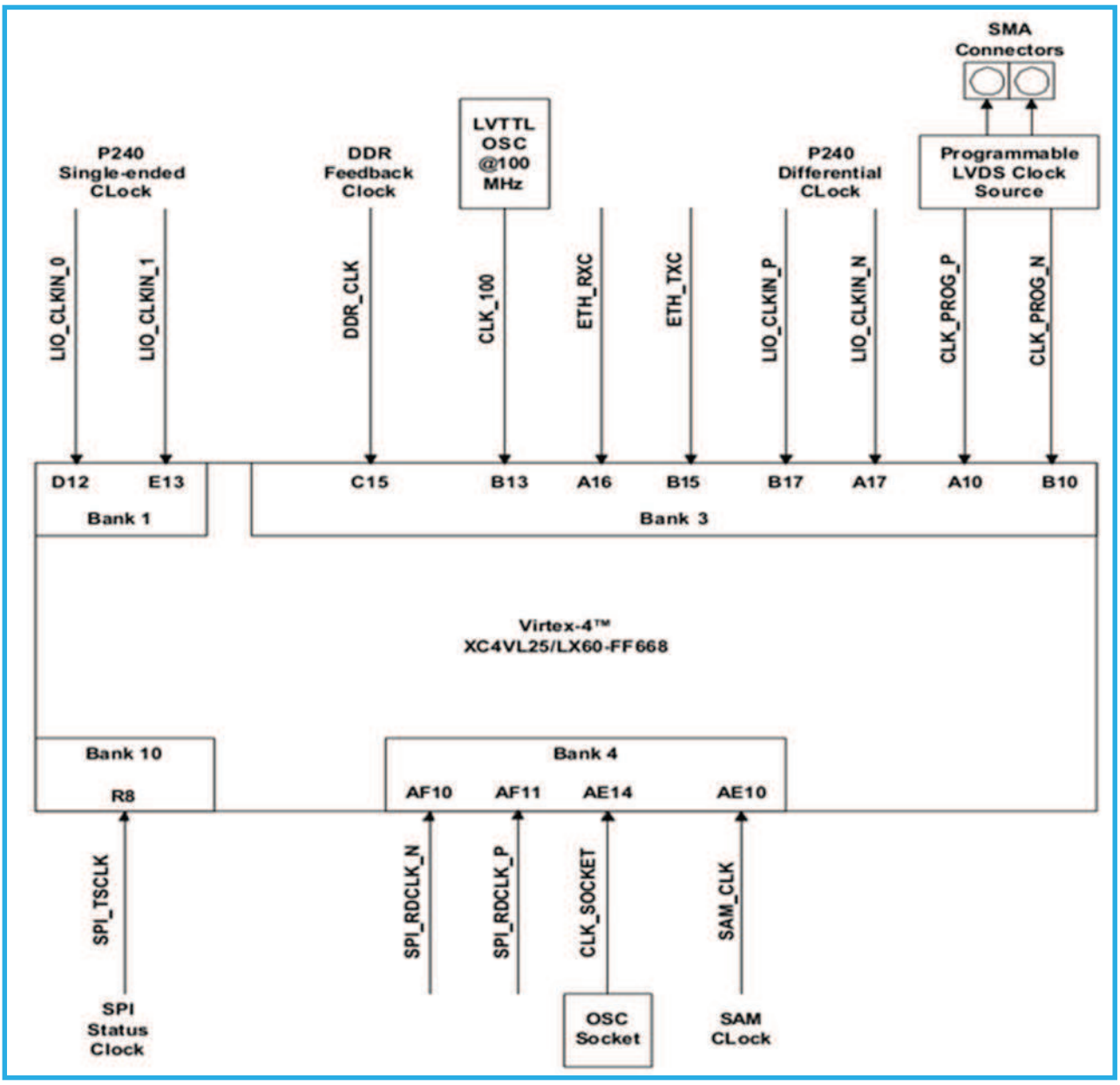


Figure I.29 : *Sources d'horloge sur le conseil du Virtex-4mb*

e.1) Source d'horloge programmable :

Un synthétiseur programmable d'horloge de LVDS est utilisé sur le kit de développement du Virtex-4 MB pour produire des signaux d'horloge de référence à l'interface de LVDS. Le port de sortie différentiel est également bien adapté pour des applications basées sur les techniques DSP.

e.2) Synthétiseur programmable d'horloge d'ICS8442 LVDS

La conception de kit de développement du Virtex-4 MB utilise le synthétiseur d'horloge : ICS8442 LVDS (figure I-30) pour générer de diverses fréquences du signal d'horloge. Une liste des caractéristiques de ce dispositif ICS8442 est montrée ci-dessous.

- Gamme de fréquences : 25MHz à 700MHz
- Période de RMS : 2.7ps (typique)
- Rapport Cycle-à-cycle : 27ps (typique)
- Élévation de rendement et temps de chute : 650ps (maximum)

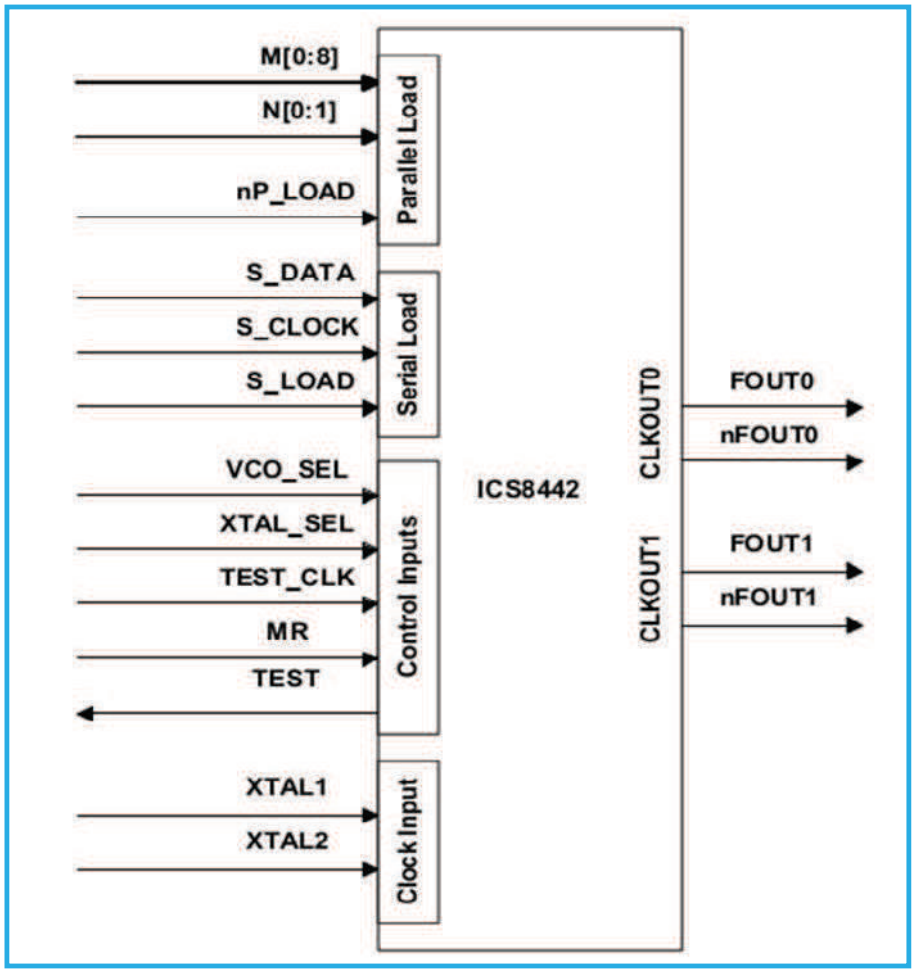


Figure I.30: *ICS8442 synthétiseur d'horloge*

e.3) 10/100 Ethernet

Le kit de développement de Virtex-4 MB renferme aussi un port d'Ethernet de 10/100 pour la connexion réseau. Un schéma fonctionnel de l'interface d'Ethernet de 10/100 est montré dans la figure suivante.

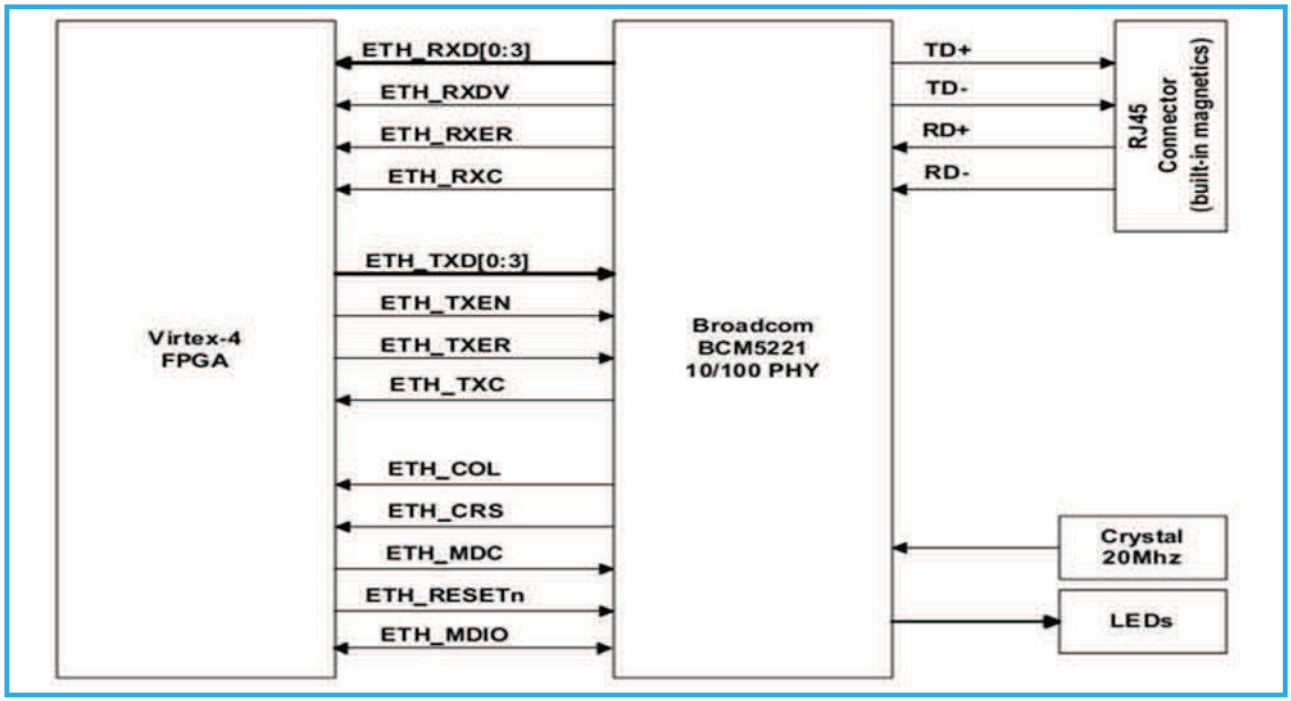


Figure I.31 : *L'interface Ethernet10/100*

f) Afficheur à cristaux liquides LCD :

Le kit de développement du Virtex-4 MB fournit une interface de 8 bits à un afficheur LCD 2x16 (MYTECH MOC-16216B-B). Le tableau suivant montre les signaux d'interface d'affichage de cet afficheur LCD.

Nom de signale	Description	Virtex-4 Pin
D0	LCD Data Bit0	K25
D1	LCD Data Bit1	P19
D2	LCD Data Bit2	AC10
D3	LCD Data Bit4	AB10
D4	LCD Data Bit4	AF12
D5	LCD Data Bit5	AE12
D6	LCD Data Bit6	AC17
D7	LCD Data Bit7	AB17
EN	LCD Enable Signal	L19
RW	LCD Write Signal (this signal is connected to logic"0" on	
RS	LCD Register Select Signal	N21

Tableau I.1 : *Signaux D'interface du LCD*

g) USB 2.0 au port RS232 :

Le kit de développement du Virtex-4 MB met à la disposition de l'utilisateur un port d'USB 2.0. Ce fait est réalisé par l'utilisation du contrôleur de pont CP2101 USB à UART. Le FPGA peut être connecté au CP2102 comme UART simple. L'interface d'UART au CP2102 peut fonctionner aux vitesses s'étendant de 300 à 921.600 bauds.

Le CP2102 est un circuit contrôleur intégrant le pont USB à UART, en fournissant une solution simple pour des communications périodiques d'USB tout en utilisant un minimum de composants et d'espace de carte. Le CP2102 inclut un sous-contrôleur à toute vitesse de fonction d'USB 2.0, l'émetteur récepteur d'USB, l'oscillateur, l'EEPROM, et le bus de données périodiques asynchrone (UART) avec un plein de signaux de commande de modem dans un paquet MLP-28 compact de 5mm x 5mm.

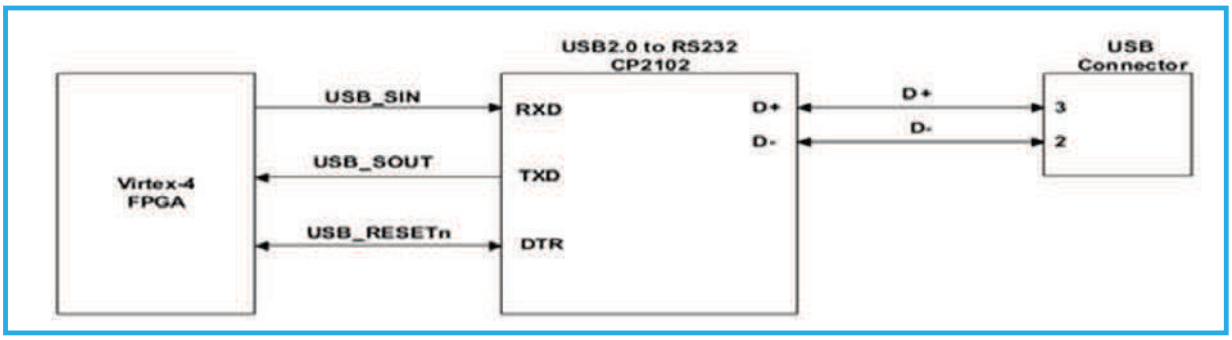


Figure I.32: *USB 2.0 to RS232 Serial Interface*

h) Liaison RS-232 :

Le kit de développement du Virtex-4 MB fournit une interface RS232 manipulant des signaux RX et TX et à des pullovers pour relier les signaux de RTS et de CTS. La figure suivante montre l'interface RS-232 au Virtex-4 LX25/LX60/SX35 FPGA.

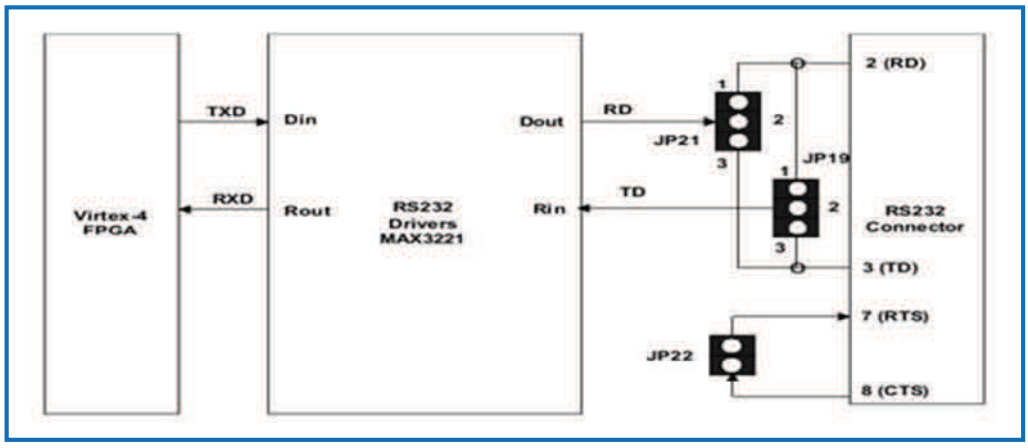


Figure I.33 : *L'interface RS232*

i) LEDs pour Utilisateurs :

Le kit de développement du Virtex-4 MB dispose quatre LEDs pour l'utilisateur qui peuvent être allumées en conduisant le signal de LEDs à la logique « 0 ». La table suivante montre les LEDs prévues pour l'utilisateur leur connexion aux pins du circuit FPGA.

Désignation LED	LED	Virtex-4 Pin
DS9	LED1	N25
DS10	LED2	V25
DS11	LED3	L26
DS12	LED4	K26

Tableau I.2: *Les Pin d'affectation des LEDs*

j) Circuits régulateurs de tension :

La figure suivante montre les schémas électriques des régulateurs de tension qui sont utilisés au sein du Kit employés sur le conseil de développement de Virtex4 MB. Suivant les indications de cette figure, un connecteur est utilisé pour fournir la tension 5.0V principale au kit. Cette source de tension est fournie à tous les régulateurs à bord pour produire des tensions dérivées de 1.2V, 2.5V, et 3.3V.

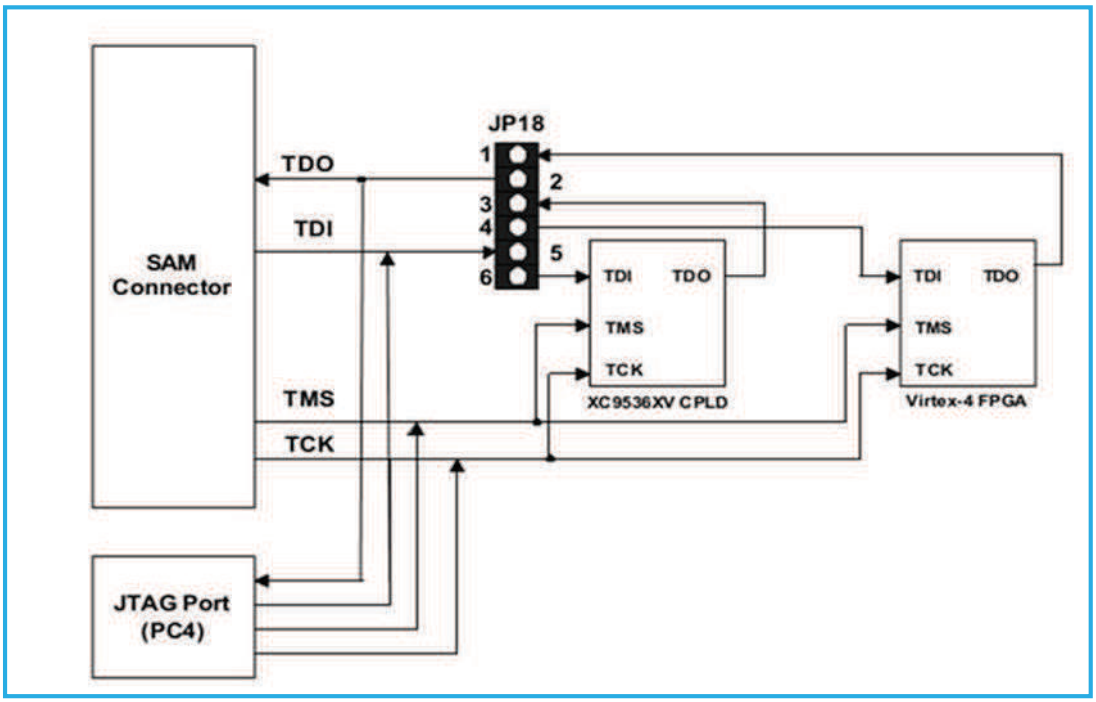


Figure I.34: *Vitex-4MB Development Board JTAG Chain*

k) Banc de tension d'entrée-sortie :

La table suivante montre les tensions d'entrée-sortie du banc Virtex-4 disponibles sur le module de développement Virtex-4 MB.

Bank	I/O Voltage
0	2.5V
1	2.5V/3.3V
2	2.5V/3.3V
3	2.5V
4	2.5V
5	2.5V/3.3V
6	2.5V
7	2.5V/3.3V
8	2.5V
9	2.5V
10	3.3V

Tableau I.3 : *Tensions entrée-sortie de banc*

I.2.6.4 Conclusion

Dans ce chapitre nous avons donné la description matérielle que nous allons utiliser dans notre projet de fin d'étude, pour cela on a présenté ce chapitre en deux partie :

Lors de la première partie, nous avons détaillé une description technique du module analogique P240. En particulier, nous avons mis l'accent sur le convertisseur analogique numérique et numérique analogique, les filtres, l'entrées et les sorties analogiques …

Au cours de la deuxième partie, on a donné un petit résumé sur les circuits FPGAs et leur architecture, ensuite nous avons présenté le kit de développement de Memec Virtex-4™mb et ses différents constituants.

Le franchissement de cette étape nous a dotés du background, côté hardware, nécessaire à l'aboutissement de notre travail.

CHAPITRE II

Description des outils de développement logiciel

Parallèlement au chapitre précédent, nous aborderons dans ce chapitre la description détaillée des outils de développement software qui ont été utilisé pour la mise en ouvre du travail à réaliser.

II.1 Le langage «VHDL» [5]

II.1.1 Introduction

Dans un passé proche, pour décrire le fonctionnement d'un circuit électronique programmable, les techniciens et les ingénieurs utilisaient des langages de bas niveau tels : (**ABEL, PALASM, ORCAD/PLD**,..) ou plus simplement un outil de saisie de schémas.

Actuellement la densité de fonctions logiques (portes et bascules) intégrées dans les **PLDs** est telle (plusieurs milliers de portes voir millions de portes) qu'il n'est plus possible d'utiliser les outils d'hier pour développer les circuits d'aujourd'hui.

Les sociétés de développement et les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits. Ils ont donc créé des langages dits de haut niveau à savoir le langages de programmation hardwares : **VHDL** et **VERILOG**.

Le langage **VHDL** *(**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage)* est un langage de description du matériel qui permet de synthétiser des fonctions logiques complexes. Un usage commun du VHDL est de décrire comment un matériel doit être implémenté, de telles descriptions peuvent être traduites en une Netlist qui décrit comment sont interconnectés et configurés les blocs formant le circuit FPGA.

Ils permettent au code écrit d'être portable, c'est à dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit. Il faut avoir à l'esprit que ces langages dits de haut niveau permettent de matérialiser les structures électroniques d'un circuit.

En effet les instructions écrites dans ces langages se traduisent par une configuration logique de portes et de bascules qui sont intégrées à l'intérieur des circuits **PLDs**.

II.1.2 Structure d'une description VHDL

Une description VHDL est composée de 2 parties indissociables à savoir :

- *L'entité (ENTITY),* elle définit les entrées et sorties.
- *L'architecture (ARCHITECTURE),* elle contient les instructions VHDL permettant de réaliser le fonctionnement attendu.

Exemple *:* Un décodeur 1 parmi 4.

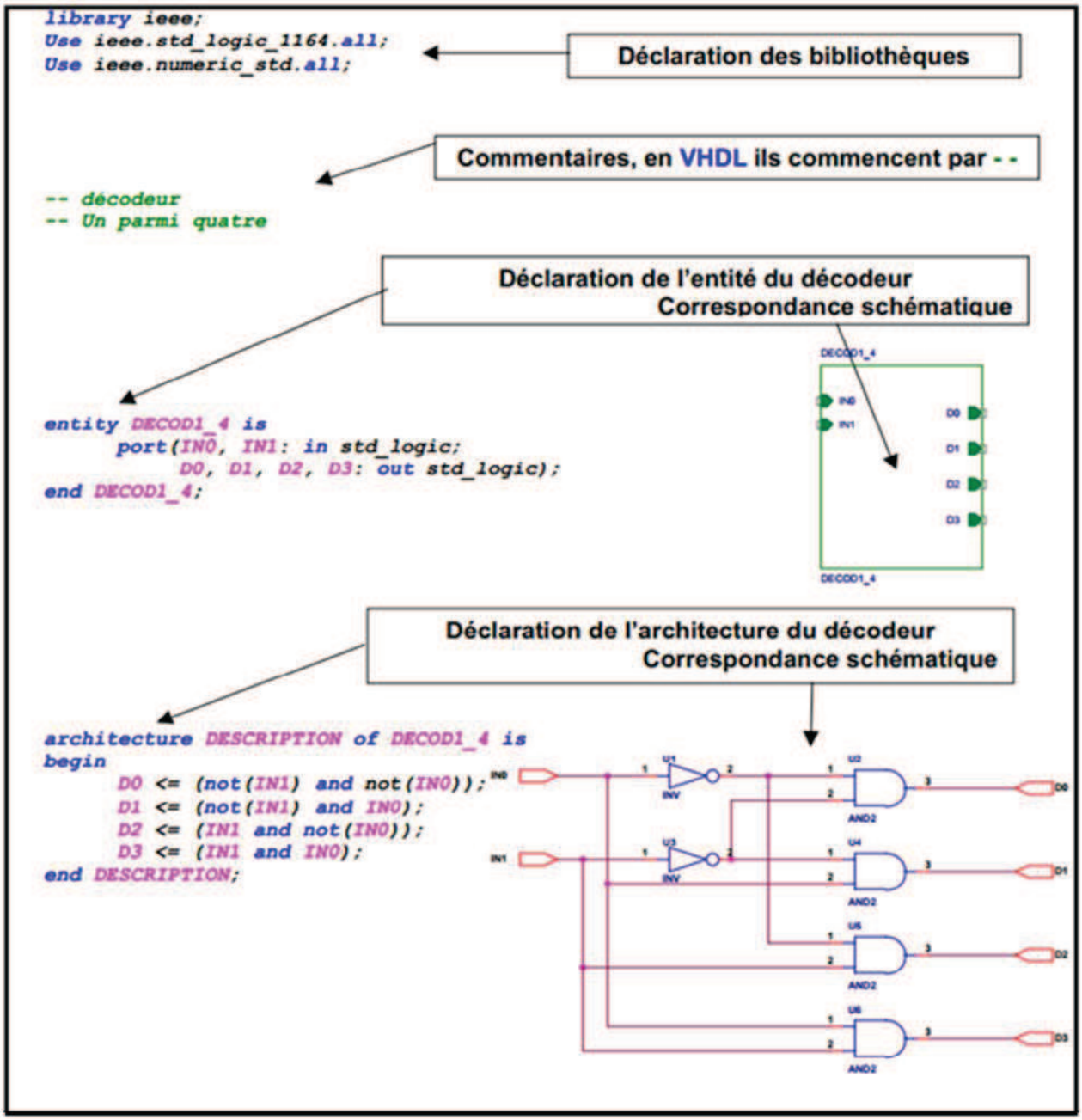


Figure II.1 *: Un décodeur 1 parmi 4*

II.1.3 Déclaration des bibliothèques

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

Syntaxe *:*

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

La directive ***Use*** permet de sélectionner les bibliothèques à utiliser.

II.1.4 Déclaration de l'entité et des entrées / sorties (I/O)

Elle permet de définir le nom de la description VHDL ainsi que les entrées et sorties utilisées, L'instruction qui les définit est ***port***.

Syntaxe:

```
Entity NOM_DE_L_ENTITE is
Port (Description des signaux d'entrées /sorties ...);
End NOM_DE_L_ENTITE;
```

Remarque : Après la dernière définition du signal de l'instruction port il ne faut jamais mettre de point-virgule.

L'instruction ***port***.

Syntaxe:

```
NOM_DU_SIGNAL : sens type;
```

On doit définir pour chaque signal : le *NOM_DU_SIGNAL*, le ***sens*** et le ***type***.

II.1.4.1 le nom du signal

Il est composé de caractères, le premier caractère doit être une lettre, sa longueur est quelconque, mais elle ne doit pas dépasser une ligne de code.

Le VHDL n'est pas sensible à la « casse », c'est à dire qu'il ne fait pas la distinction entre les majuscules et les minuscules.

II.1.4.2 Le sens du signal

- *In* : pour un signal en entrée.
- *Out* : pour un signal en sortie.
- *Inout*: pour un signal en entrée sortie
- *buffer* : pour un signal en sortie mais utilisé entrée dans la description.

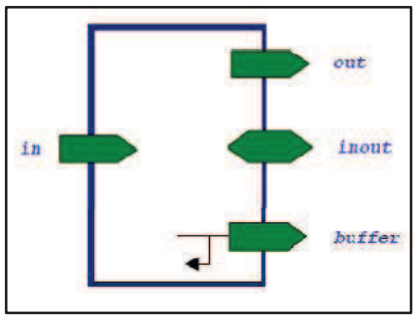


Figure II.2:*Les différent sens d'un signale*

II.1.4.3 Le type

Le type utilisé pour les signaux d'entrées / sorties est :

- le *std_logic* pour un signal.
- le *std_logic_vector* pour un bus composé de plusieurs signaux.

Par exemple un bus bidirectionnel de 5 bits s'écrira :

LATCH: inout std_logic_vector (4 downto0);

Où *LATCH (4)* correspond au MSB et *LATCH(0)* correspond au LSB. Les valeurs que peuvent prendre un signal de type *std_logic* sont :

- *'0' ou 'L'* : pour un niveau bas.
- *'1' ou 'H'* : pour un niveau haut.
- *'Z'* : pour état haute impédance.
- *'-'* : Quelconque, c'est à dire n'importe quelle valeur.

II.1.5 Déclaration de l'architecture correspondante à l'entité (description du fonctionnement)

L'architecture décrit le fonctionnement souhaité pour un circuit ou une partie du circuit. En effet le fonctionnement d'un circuit est généralement décrit par plusieurs modules **VHDL**. On entend par module le couple *ENTITE/ARCHITECTURE*. Dans le cas de simples **PLDs, o**n trouve souvent un seul module.

L'architecture établit à travers les instructions les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire, séquentiel, voir les deux au même temps.

II.1.6 Les instructions de base (mode « concurrent »), logique combinatoire

Qu'est-ce que le mode « concurrent » ? Pour une description VHDL toutes les instructions sont évaluées et affectent les signaux de sortie en même temps. L'ordre dans lequel elles sont écrites n'a aucune importance.

En effet la description génère des structures électroniques, c'est la grande différence entre une description VHDL et un langage informatique classique.

Dans un système à microprocesseur, les instructions sont exécutées les unes à la suite des autres.

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

II.1.6.1 Les opérateurs

a) L'affectation simple <=

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

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

Les valeurs numériques que l'on peut affecter à un signal sont les suivantes :

- '1' ou 'H' pour un niveau haut avec un signal de 1 bit.
- '0' ou 'L' pour un niveau bas avec un signal de 1 bit.
- 'Z' pour un état haute impédance avec un signal de 1 bit.
- '-' pour un état quelconque, c'est à dire '0' ou '1'. Cette valeur est très utilisée avec les instructions : *when ... else* et *with .... Select ....*
- Pour les signaux composés de plusieurs bits on utilise les guillemets " … ".

b) Opérateur de concaténation: &

Cet opérateur permet de joindre des signaux entre eux.

Exemple:

Soit A et B de type 3 bits et S1 de type 8 bits ; A = "001" et B ="110"

S1 <= A & B &"01" ; après cette affectation S1 = "001110 01".

c) Opérateurs logiques

Opérateur	VHDL
ET	and
NON ET	nand
OU	or
NON OU	nor
OU EXCLUSIF	xor
NON OU EXCLUSIF	xnor
NON	not
DECALAGE A GAUCHE	sll
DECALAGE A DROITE	srl
ROTATION A GAUCHE	rol
ROTATION A DROITE	ror

Tableau II.1: *Les opérateurs logiques*

Exemple : *S1 <= A sll 2 ; -- S1 = A décalé de 2 bits à gauche.*

d) Opérateurs arithmétiques

Opérateur	VHDL
ADDITION	+
SOUSTRACTION	-
MULTIPLICATION	*
DIVISION	/

Tableau II.2 : *Les opérateurs arithmétiques*

Exemple : *S1 <= A – 3 ; -- S1 = A – 3 -- On soustrait 3 à la valeur de l'entrée / signal A*

e) Opérateurs relationnels

Ils permettent de modifier l'état d'un signal ou des signaux suivant le résultat d'un test ou d'une condition.

En logique combinatoire ils sont souvent utilisés avec les instructions:

- *when … else …* - *with …. Select ….*

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

Tableau II.3 : *Les opérateurs relationnels*

II.1.6.2 Les instructions du mode « concurrent »

a) Affectation conditionnelle

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

Syntax :

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

Remarque : l'instruction *[else expression]* n'est pas obligatoire mais elle fortement conseillée. Elle permet de définir la valeur du SIGNAL au cas où la condition n'est pas remplie. On peut mettre en cascade cette instruction.

Exemple :

```
-- Structure évoluée d'un multiplexeur 4 vers 1
S2 <= E1 when (SEL="00»)
Else E2 when (SEL="01")
Else E3 when (SEL="10»)
Else E4 when (SEL="11")
Else'0';
```

b) Affectation sélective

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

Syntax:

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

Remarque :

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

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

Exemple :

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

II.1.7 Les instructions du mode séquentiel

II.1.7.1 Définition d'un PROCESS

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

Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standards de la programmation structurée comme dans les systèmes à microprocesseurs.

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

Syntaxe :

```
[Nom_du_process :] process (Liste_de_sensibilité_nom_des_signaux)
Begin
-- instructions du process
End process [Nom_du_process] ;
```

Remarque: Le nom du process entre crochet est facultatif, mais il peut être très utile pour repérer un process parmi d'autres lors de phases de mise au point ou de simulations.

L'assignation conditionnelle

```
if condition then
    instructions
[elsif condition then instructions]
[else instructions]
end if ;

Exemple:
if (RESET='1') then SORTIE <= "0000";
end if ;
```

L'assignation sélective

```
case signal_de_slection is
when valeur_de_sélection => instructions
[when others => instructions]
end case;

Exemple:
case SEL is
        when "000" => S1 <= E1;
        when "001" => S1 <= '0';
        when "010" | "011" => S1 <='1';
        -- La barre | permet de réaliser
        -- un ou logique entre les deux
        -- valeurs "010" et "011"
        when others => S1 <= '0';
end case;
```

Tableau II.4 : *Les assignations sélectives et conditionnelles*

Exemple: Déclaration d'une bascule D.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
Entity BASCULED is
port (D,CLK : in std_logic;
        S: out std_logic);
End BASCULED;
Architecture DESCRIPTION of BASCULED is
begin
PRO_BASCULED :process (CLK)
    begin
        if (CLK'event and CLK ='1') then
```

```
            S <= D;
        end if;
    end process PRO_BASCULED;
end DESCRIPTION;
```

Commentaire :

- Seul le signal *CLK* fait partie de la liste de sensibilité. D'après les règles de fonctionnement énoncées précédemment, seul un changement d'état du signal *CLK* va déclencher le process et par conséquent évaluer les instructions de celui-ci.
- L'instruction *if (CLK'event and CLK='1') then* permet de détecter un front montant du signal CLK. La détection de front est réalisée par l'attribut *event* appliqué à l'horloge *CLK*. Si on veut un déclenchement sur un front descendant, il faut écrire l'instruction suivante : *if (CLK'event and CLK='0').*
- Les bibliothèques IEEE possèdent deux instructions permettant de détecter les fronts montants) *rising_edge (CLK)* ou descendants *falling_edge (CLK).*
- Si la condition est remplie alors le signal de sortie S sera affecté avec la valeur du signal d'entrée D.

II.1.7.2 Les Types

Le VHDL étant un langage typé tous les paramètres utilisés doivent avoir un type défini. Le type définit le format des données et l'ensemble des opérations applicables à ces données ; on distingue quatre catégories de type :

a) Le type scalaire :

Il définit les données qui n'ont pas de structure interne ; ce type est définit par un domaine de définition ou par une liste de valeurs, permettant de limiter la taille des données lui appartenant. Ce type regroupe les entiers et les flottants.

Exemple :

```
Type octet is range 0 to 255 ;
```

b) Le type structuré :

Il permet de structurer les données au moyen de tableaux. Un tableau est un ensemble d'éléments de même type.

Exemple :

```
Subtype natural is integer range 0 to integer'high;
Type bit_vector is array (natural range <>) of bit;
```

c) Le type pointeur :

Ce type ne correspond à aucun objet synthétisable dans le circuit. La syntaxe qui permet la déclaration de ce type est donnée comme suit :

Exemple :

```
Type nom_type is access type_pointe ;
```

d) Le type fichier :

Le fichier représente une séquence d'éléments de même type. Pour déclarer un fichier on utilise la syntaxe :

Exemple :

```
Type nom_de_type is file of type_des_éléments;
```

e) Les constantes :

Les constantes permettent de paramétrer sous forme littérale les variables non modifiables au cours de l'exécution du programme.

Exemple :

```
Constant age: integer :=25 ;
```

f) Les signaux :

Ils correspondent aux canaux de communication du circuit, donc ils ont un rôle central dans le langage.

Exemple :

```
Signal temps : std_logic_vector (dimension 1 downto 0);
```

g) Les variables :

Ils servent au stockage des valeurs nécessaires au bon déroulement d'un algorithme.

Exemple *:*

```
Variable tmp: integer range 0 to 18 ;
Tmp <= a+b ;
If tmp > 9 then
Tmp <= tmp -10;
end if;
```

II.1.7 Exemple du programme en VHDL développé durant le stage

Le but de cet exemple est d'afficher la phrase **« univ de medea 2013»** sur l'afficheur LCD16X2, cet exemple a un seul entrée 'clk' et quatre sortie 'lcd_e', 'lcd_rs', 'lcd_rw' et 'lcd_db'

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lcd16x2_ctrl_demo is
   Port (clk : in  STD_LOGIC;
lcd_e : out  STD_LOGIC;
lcd_rs : out  STD_LOGIC;
lcd_rw : out  STD_LOGIC;
lcd_db : out  STD_LOGIC_VECTOR (7 downto 4));
end lcd16x2_ctrl_demo;
architecture Behavior of lcd16x2_ctrl_demo is
signal timer: natural range 0 to 100000000 := 0;
signal switch_lines :std_logic := '0' ;
signal line1 :std_logic_vector (127 downto 0) ;
signal line2 :std_logic_vector (127 downto 0) ;
signal rst :std_logic ;
signal line1_buffer :std_logic_vector (127 downto 0) ;
signal line2_buffer :std_logic_vector (127 downto 0) ;
```

```
begin
DUT : entity work.lcd16x2_ctrl
generic map (clk_period_ns => clk_period_ns)
port map (
clk  => clk,
rst => rst,
lcd_e => lcd_e,
lcd_rs => lcd_rs,
lcd_rw => lcd_rw,
lcd_db => lcd_db,
line1_buffer => line1_buffer,
line2_buffer => line2_buffer);
rst <= '0';
line1 (127 downto 120) <= X"00";
line1 (119 downto112) <= X"55"; -- U
line1 (111 downto 104) <= X"6E"; -- n
line1 (103 downto96) <= X"69"; -- i
line1 (95 downto 88) <= X"76"; -- v
line1 (87 downto 80) <= X"00";
line1 (79 downto 72) <= X"64"; -- d
line1 (71 downto 64) <= X"65";-- e
line1 (63 downto 56) <= X"00";
line1 (55 downto 48) <= X"4D"; -- M
line1 (47 downto 40) <= X"65"; -- e
line1 (39 downto 32) <= X"64"; -- d
line1 (31 downto 24) <= X"65"; -- e
line1 (23 downto16) <= X"61"; -- a
line1 (15 downto 8) <= X"00";
line1(7 downto 0) <= X"00";
line2 (127 downto 120) <= X"00";
line2 (119 downto 112) <= X"00";
line2 (111 downto 104) <= X"00";
line2 (103 downto 96) <= X"00";
line2 (95 downto 88) <= X"00";
```

```
line2 (87 downto 80) <= X"00";
line2 (79 downto 72) <= X"32"; -- 2
line2 (71 downto 64) <= X"30"; -- 0
line2 (63 downto 56) <= X"31"; -- 1
line2 (55 downto 48) <= X"33"; -- 3
line2 (47 downto 40) <= X"00";
line2 (39 downto 32) <= X"00";
line2 (31 downto 24) <= X"00";
line2 (23 downto 16) <= X"00";
line2 (15 downto 8) <= X"00";
line2 (7 downto 0) <= X"00";
line1_buffer <= line2 when switch_lines ='1' else line1;
line2_buffer <= line1 when switch_lines ='1' else line2;
process (clk)
begin
if rising_edge (clk) then
if timer =0 then
timer <= 100000000;
switch_lines <= not switch_lines;
else
timer <= timer-1;
end if;
end if;
end process;
end architecture behavior;
```

Et voici les résultats de la simulation avec l'éditeur ISE (Xilinx) :

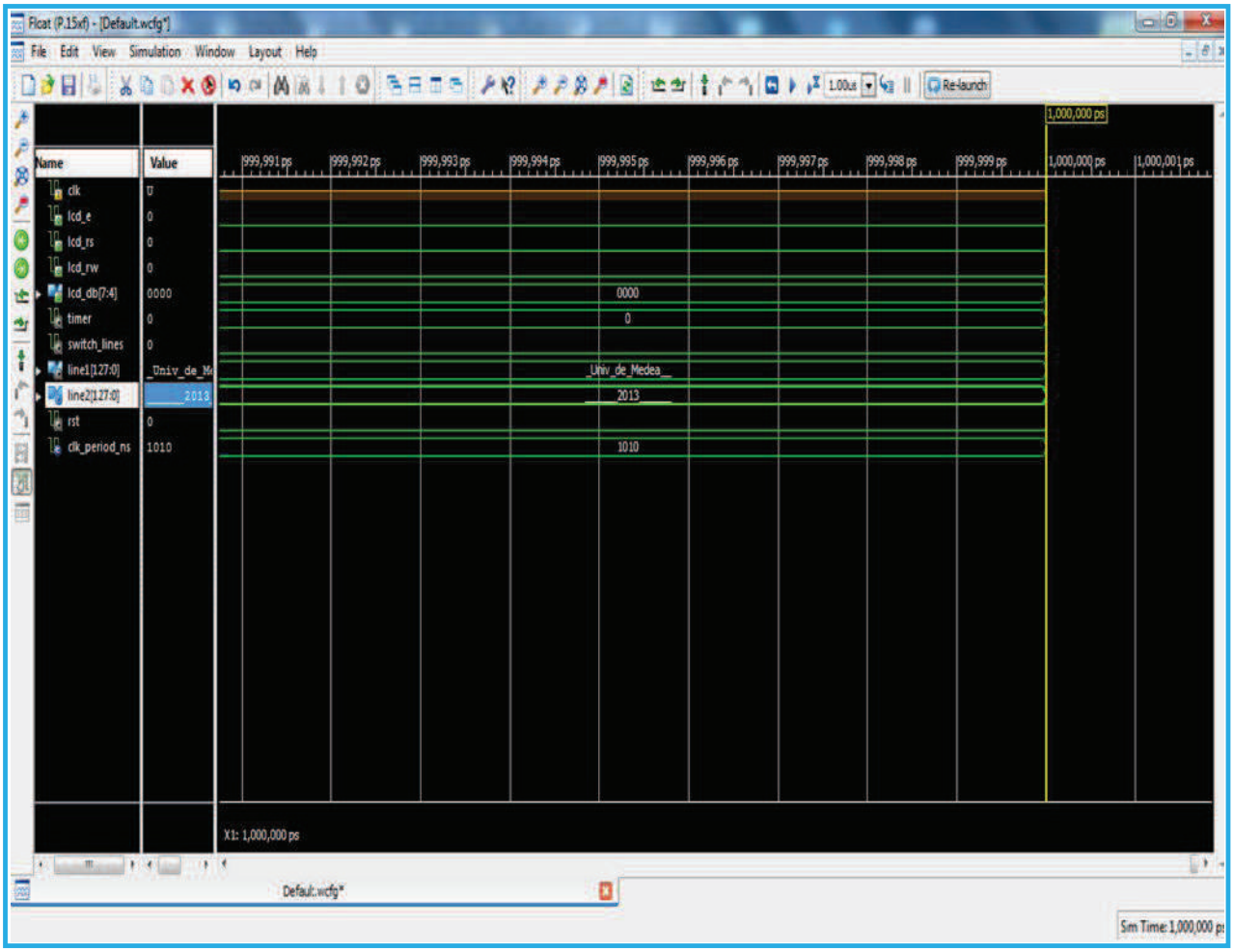


Figure II.3 : *Résultat de simulation.*

II.2 L'outil de développement ISE de Xilinx

II.2.1 Introduction et aperçu

ISE (Integrated Software Environment) est un logiciel de design de Xilinx qui nous permet de faire entrer votre circuit (comme design) à travers design entrée (Xilinx device programming). L'ISE Project Navigator va vous guider à développer et à suivre l'évolution de votre design à travers les différentes étapes dans le ISE design flow.

II.2.2 Etapes d'ISE design flow [6]

II.2.2.1 Entrée du design

C'est la première des étapes dans l'ISE design flow. Durant cette étape, vous créer vos fichiers sources basés sur les objectifs de votre circuit. Vous pouvez créer un fichier de haut niveau utilisant un langage de description matériel (Hardware Description Language - HDL-) comme VHDL, VERILOG ou ABEL sinon en utilisant un schématique. Comme vous pouvez utiliser plusieurs formats pour les fichiers sources de bas niveau dans votre design.

II.2.2.2 Synthèse

Après le design entry nous passons à la synthèse. Durant cette étape, les fichiers VHDL et VERILOG ou autres deviennent des fichiers Netlist qui est acceptés comme des entrées pour l'étape de l'implémentation.

Notons aussi que si on travaille avec un fichier EDIF ou NGC/NGO synthétisé on peut sauter les deux premières étapes et aller directement à l'implémentation.

II.2.2.3 Implémentation

Après la synthèse, on exécute design implémentation, qui convertit le design (fichier) logique en un format de fichier physique qui pourra être chargé dans le circuit cible. A partir du navigateur de projets (Project Navigator) on peut exécuter le processus d'implémentation en une seule étape ou en plusieurs. Les processus de l'implémentation varient suivant que vous ciblez un FPGA (Field Programmable Gate Array) ou un CPLD (Complex Programmable Logic Device).

II.2.2.4 Vérification

On peut vérifier le fonctionnement de notre design ou circuit en différents points dans le design flow. On peut utiliser un logiciel de simulation (ModelSim, Xilinx ISE Simulator,...) pour vérifier le fonctionnement et le timing de votre design ou une portion de celui-ci. Le simulateur interprète le code VHDL ou Verilog en fonctionnement du circuit et donne des résultats logiques pour le HDL décrit afin de déterminer l'opération correcte du circuit. La simulation nous permet de créer et vérifier les fonctions complexes dans un temps relativement court. On peut aussi faire une vérification directe sur le circuit après l'avoir programmer.

II.2.2.5 Configuration du circuit

Après avoir générer un fichier de programmation, on configure notre circuit. Durant la configuration, on génère des fichiers de configuration et chargez les fichiers de programmation dans le circuit à partir de votre ordinateur central.

II.2.3 Familiarisation et procédure de réalisation d'un projet avec ISE

II.2.3.1 Créer un nouveau projet

D'abord, on va voir comment se présente la fenêtre principale du navigateur de projets.

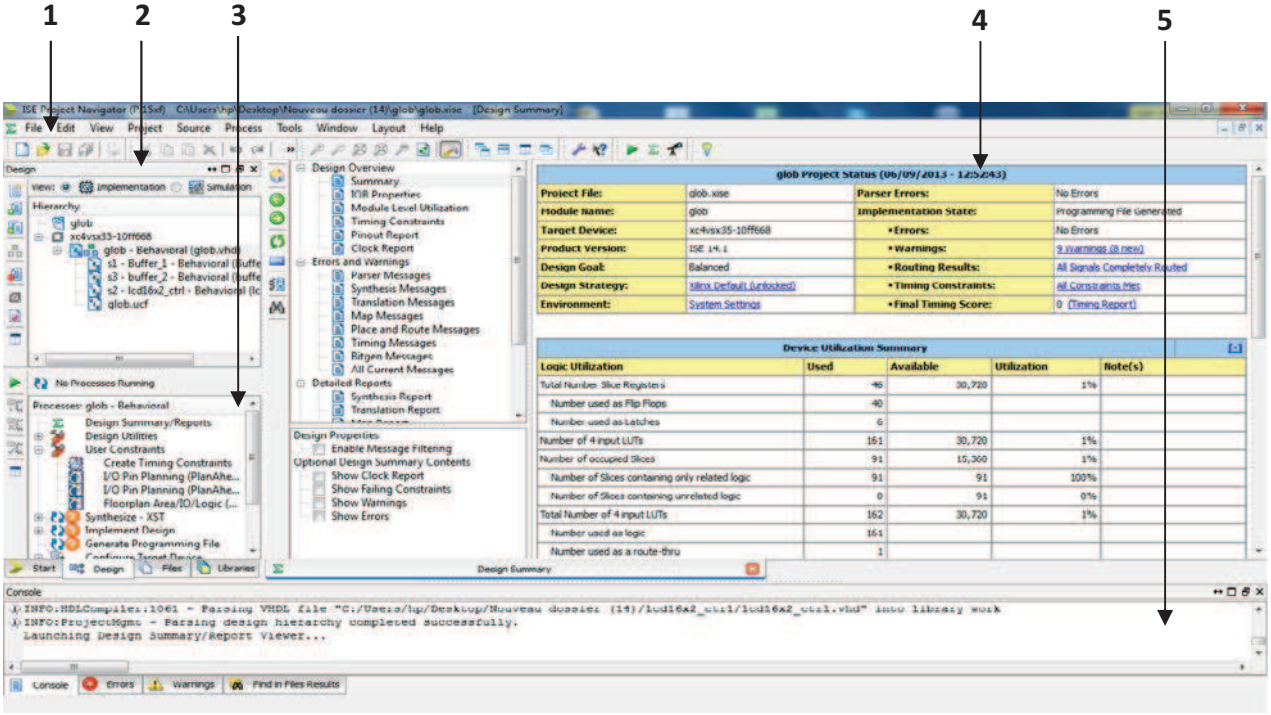


Figure II.4: *La fenêtre principale de l'ISE*

1) Barre d'outils,
2) Fenêtre des sources,
3) Fenêtre des Processes,
4) Workspace,
5) Fenêtre Transcript,

Donc, pour créer un projet on va à ***File => New Project***. Une fenêtre New Project Wizard s'ouvre, on tape dedans le nom de notre projet et son emplacement. Et préciser le type de notre fichier source haut niveau (dans notre cas c'est HDL).

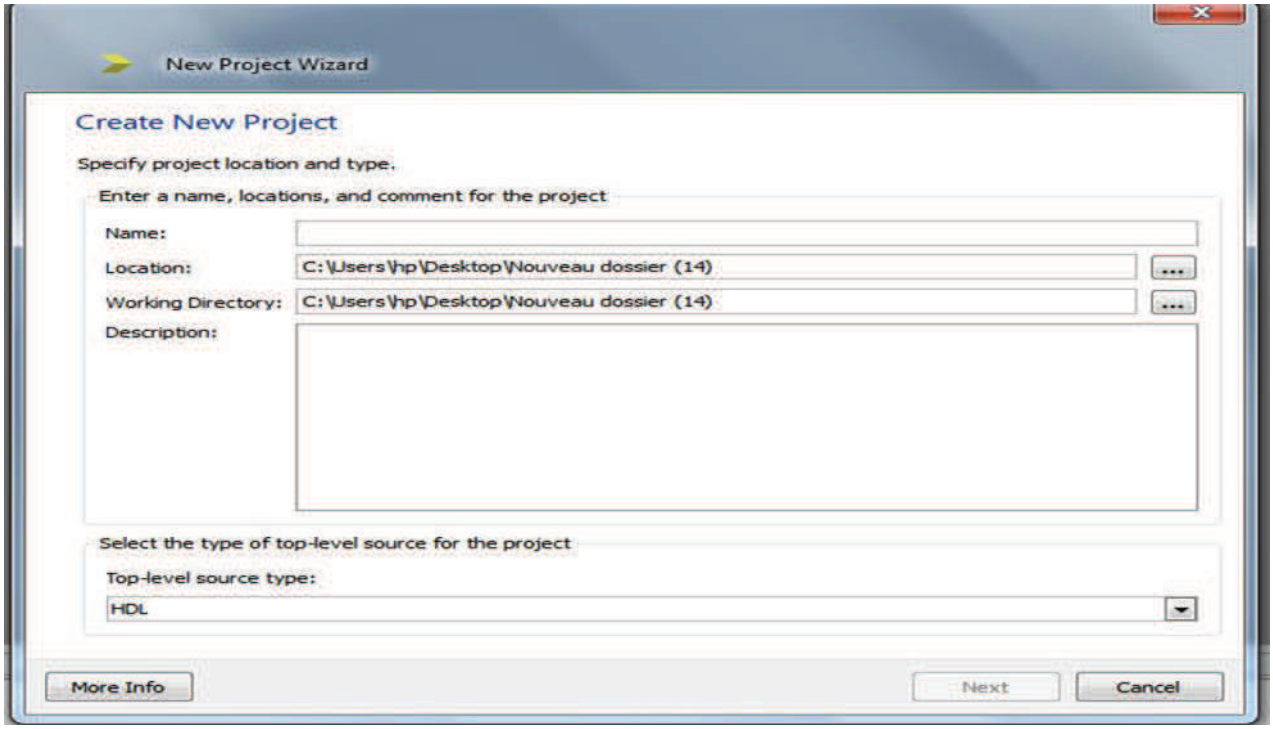


Figure II.5: *La fenêtre New Project Wizard*

On Appuie sur ***Next.***

Une autre fenêtre s'ouvre et nous permet de choisir les spécifications de notre circuit tel que: Famille, référence, boitier, vitesse, type du fichier source, outil de synthèse, le simulateur et le langage préféré pour la description matérielle, etc.

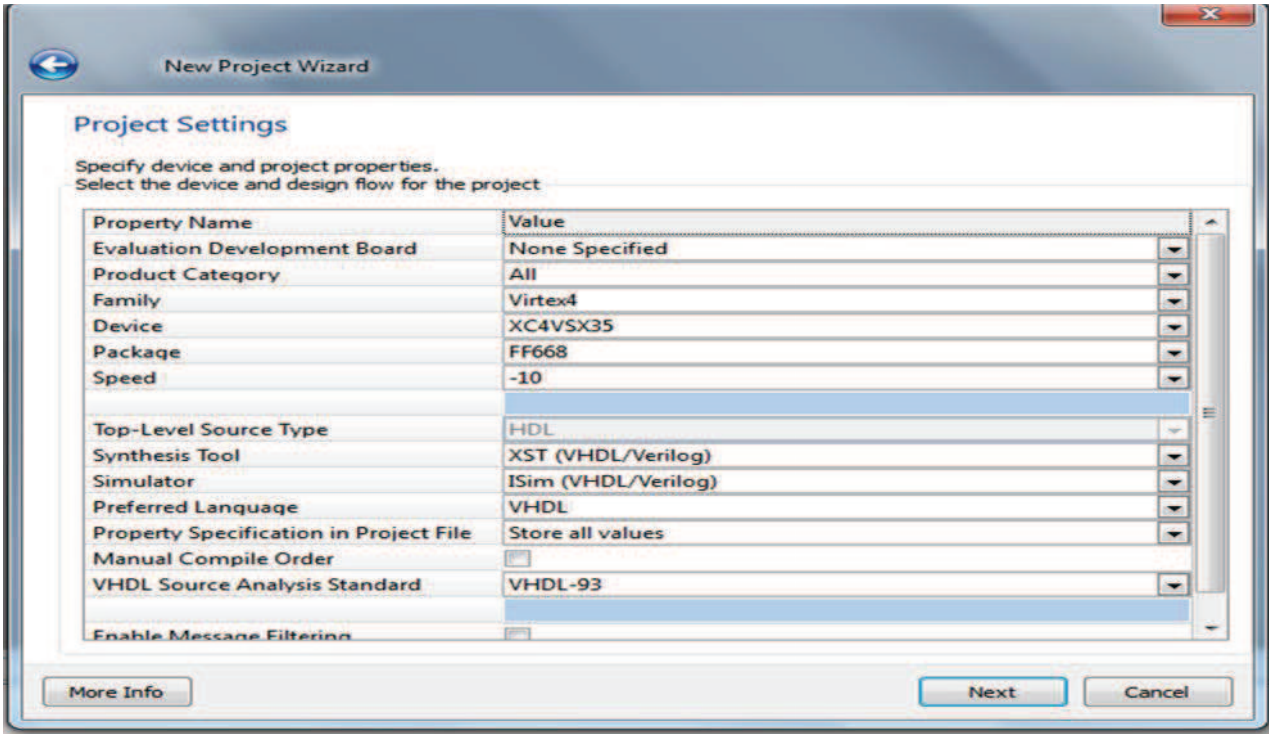


Figure II.6: *La fenêtre New Project Wizard – Device Properties*

On Appuie sur ***Next***.

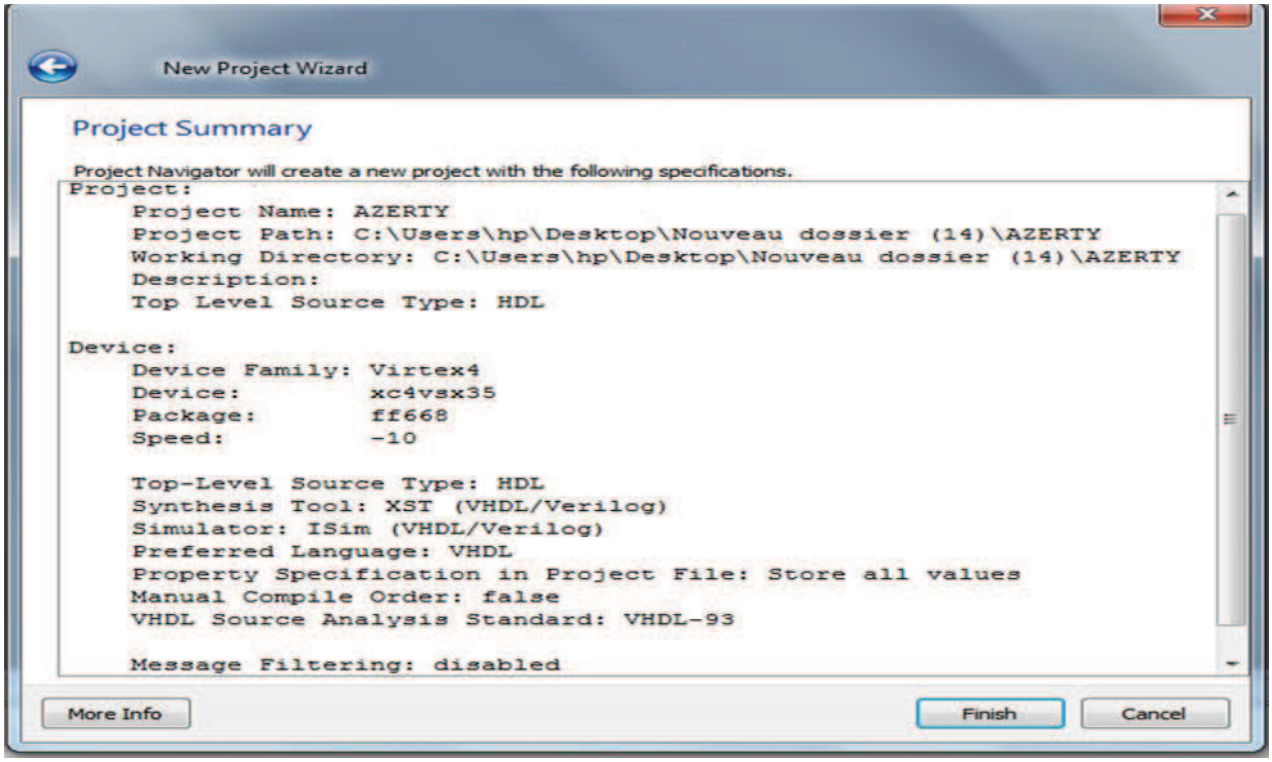


Figure II.7: *La fenêtre New Project Wizard – Project Summary*

A la fin nous aurons un résumé des paramètres de notre projet. Appuyez sur Finish. Voici à quoi va ressembler notre navigateur de projets.

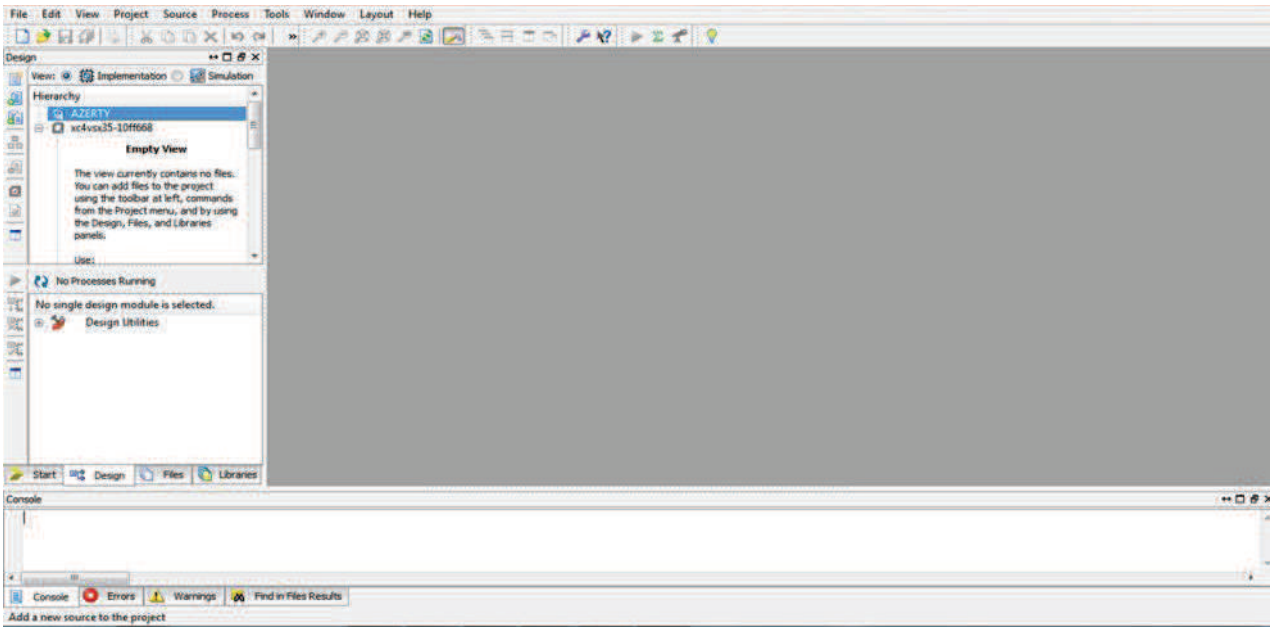


***Figure II.8**: ISE*

II.2.3.2 Créer un fichier source HDL

On fait un clic droit sur votre circuit qui se trouve dans la fenêtre Sources et choisissez **New source => VHDL Module** et on tape le nom de notre fichier source.

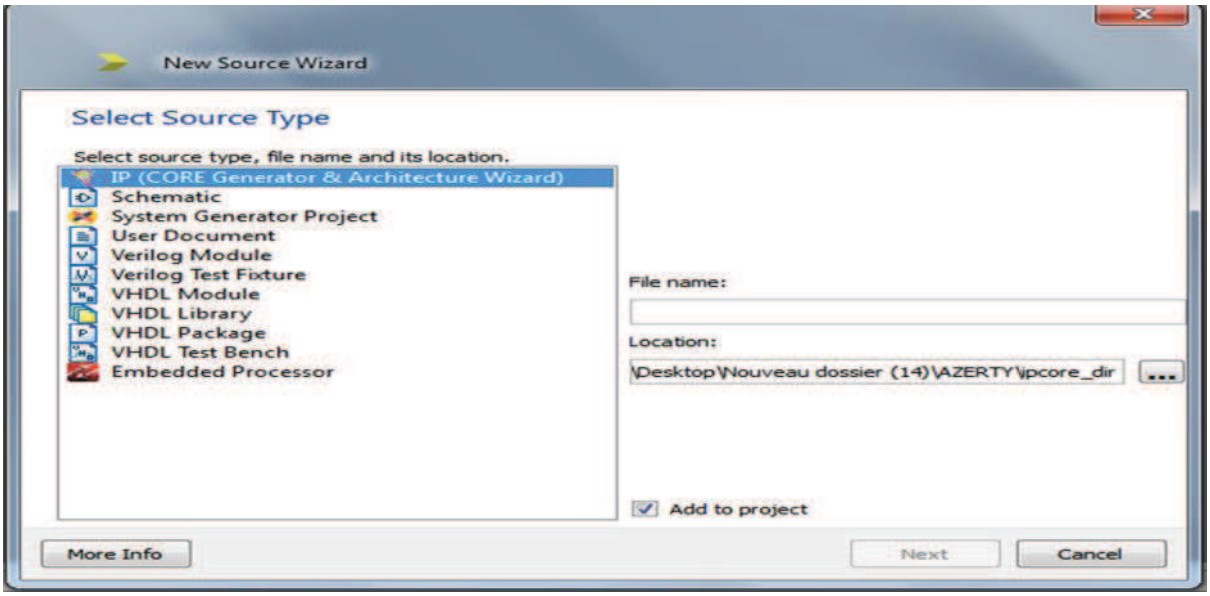


***Figure II.9**: La fenêtre New Source Wizard*

Appuyez sur ***Next***

La fenêtre suivante va nous permettre de donner la structure de notre système en définissant les ports (nom, taille, …) et en donnant un nom pour notre système et pour l'architecture.

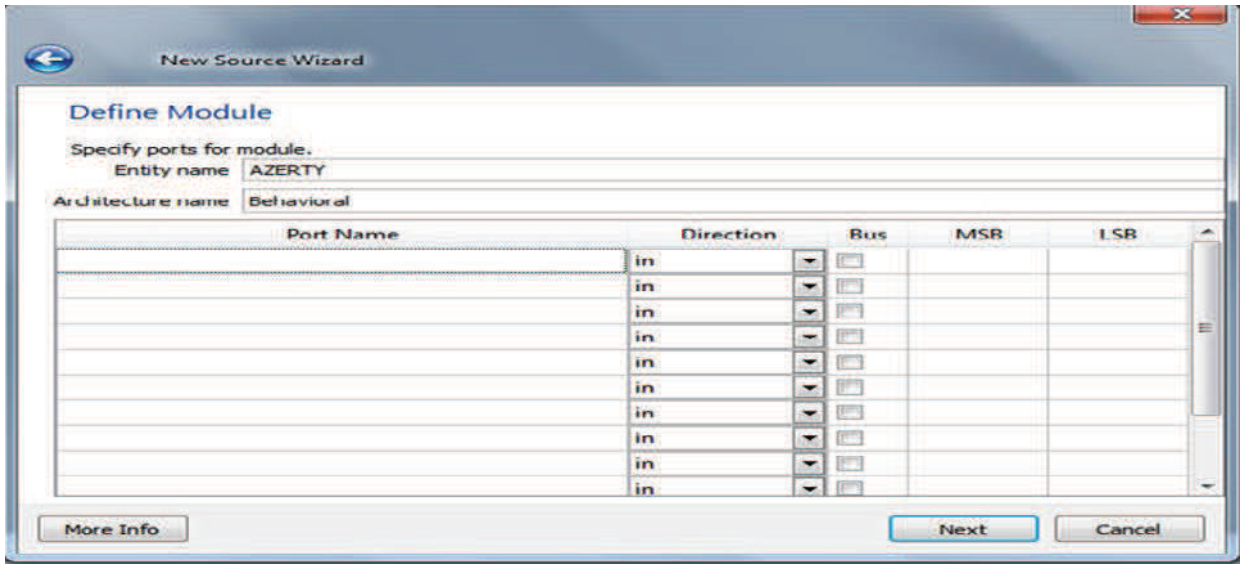


***Figure II.10**: La fenêtre New Source Wizard – Define Module*

On Appuie sur ***Next***.

Un petit résumé de notre fichier source s'affiche.

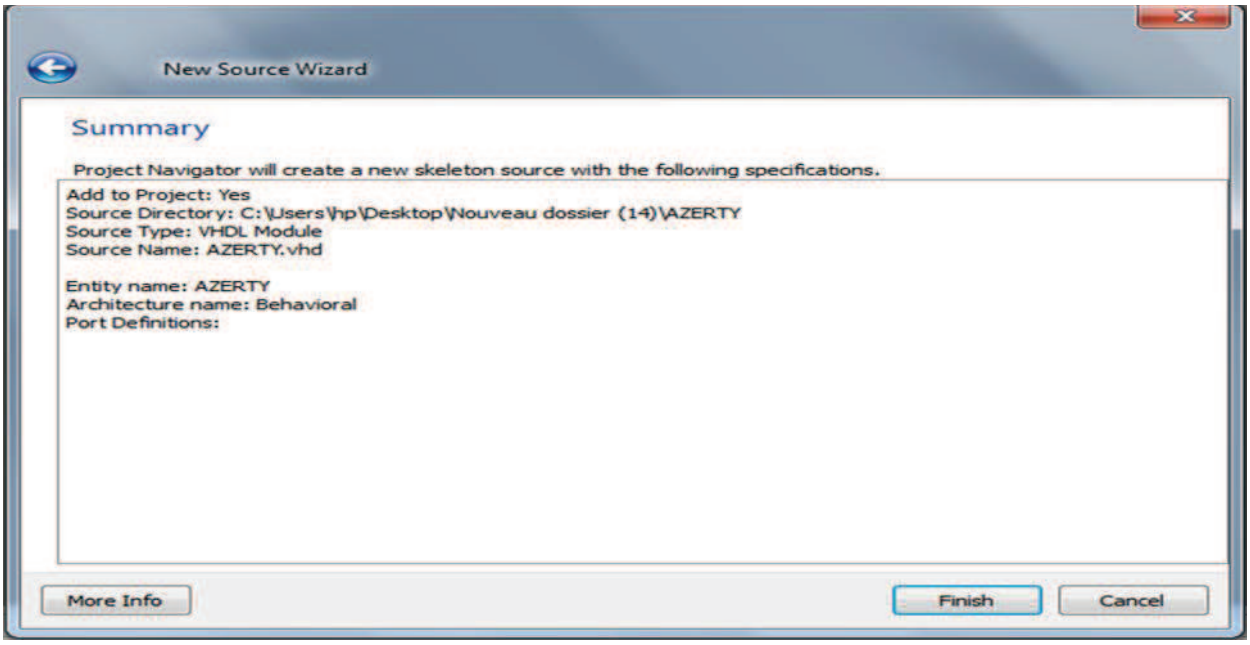


***Figure II.11**: La fenêtre New Source Wizard - Summary*

A la fin on aura ça

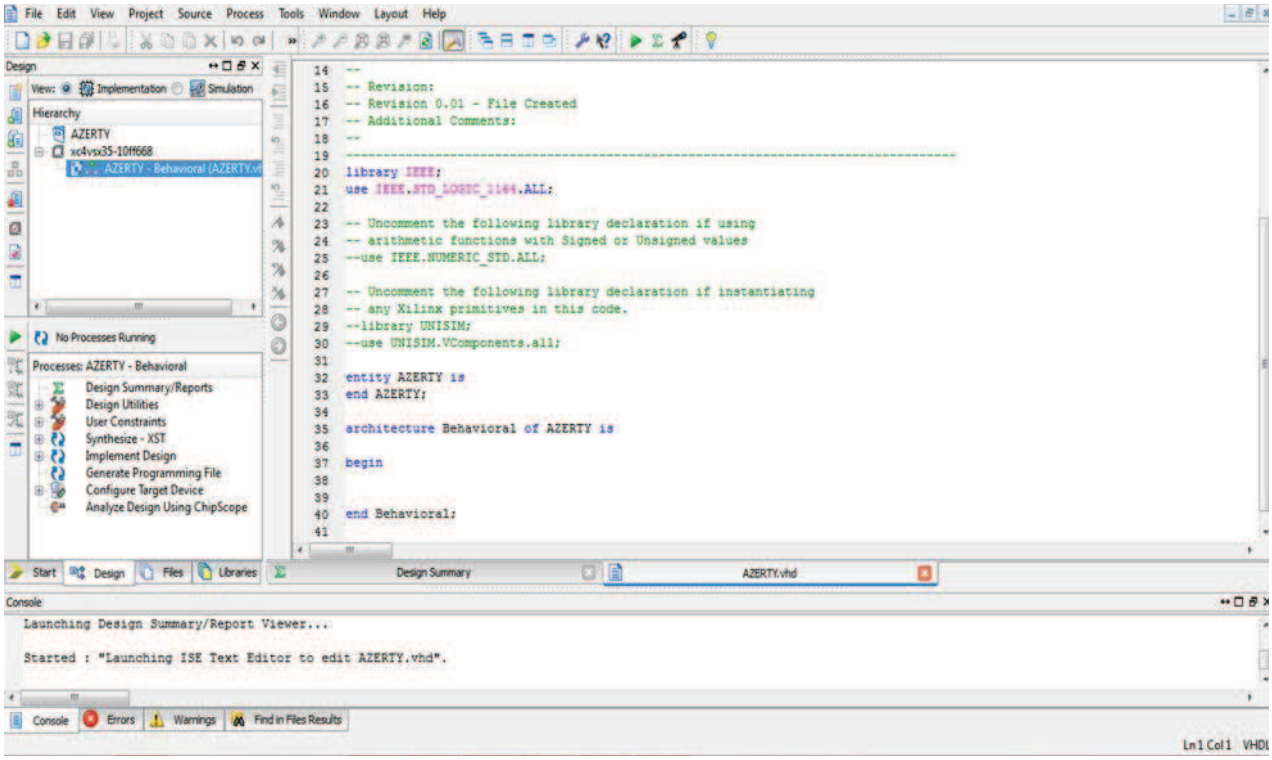


Figure II.12:*ISE avec notre fichier source crée*

II.2.3.3 Simulation d'un fichier source HDL

Pour la simulation, on va faire un petit programme de multiplication qui donnera à la sortie le résultat de la multiplication a*b.

Pour faire cette opération, on va ajouter une commande qui se trouvera dans l'architecture après l'instruction begin.

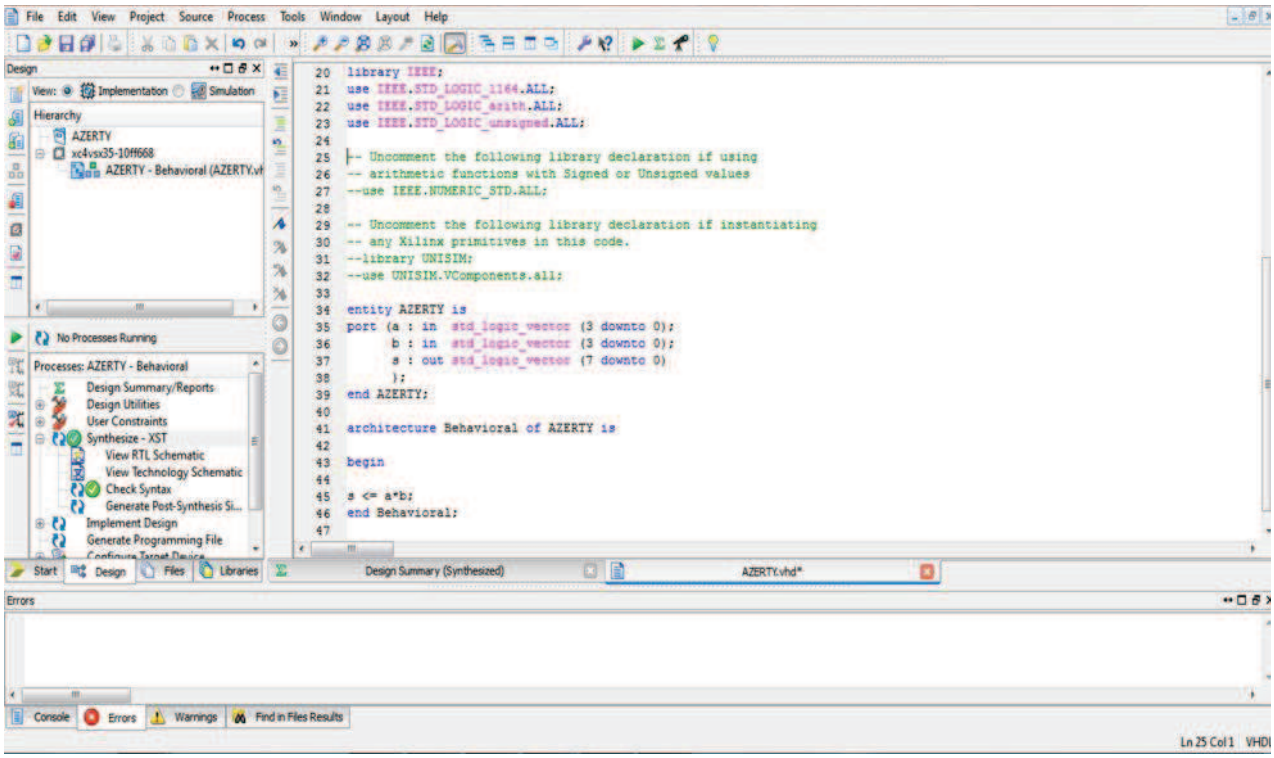


Figure II.13: *ISE avec notre fichier source « AZERTY.vhd »*

Avant la simulation, on doit d'abord tester la syntaxe de notre programme. Pour le faire, on enregistre les derniers changements faits et on va à la fenêtre **sources => Synthesize XST => Check syntax (double clique)**.

Si toute marche bien on aura un message qui informe que la vérification est terminée avec succès.

Pour la simulation, on doit maintenant créer un fichier source de type Test Bench Waveform de la même manière que la création d'un fichier VHDL Module :

Fenêtre Sources, clique droit dans cette fenêtre et choisissez **New source => Test Bench Waveform** (lui donner un nom).

Après avoir validé et donné quelques configurations (temps d'exécution, fronts d'horloge, période,…) on appuie sur finish. A ce stade, une fenêtre va apparaître et nous donner la main pour faire entrer les valeurs de nos entrées (ici a et b).

Remarque : Au cas où on n'apprécie pas l'affichage en hexadécimal, on peut le changer en décimal avec un clic droit sur la zone de la valeur et cocher Décimal.

On fait entrer maintenant nos valeurs et ensuite on change dans la fenêtre sources le type de nos sources de implémentation à Bihavioral simulation et enregistrons. On va avoir notre

fichier source test bench. Pour la simulation, on le sélectionne et on va vers la fenêtre **Processes => Xilinx ISE Simulator (votre simulateur) => Simulate Bihavioral Model (double clique).**

Voici le résultat de notre simulation :

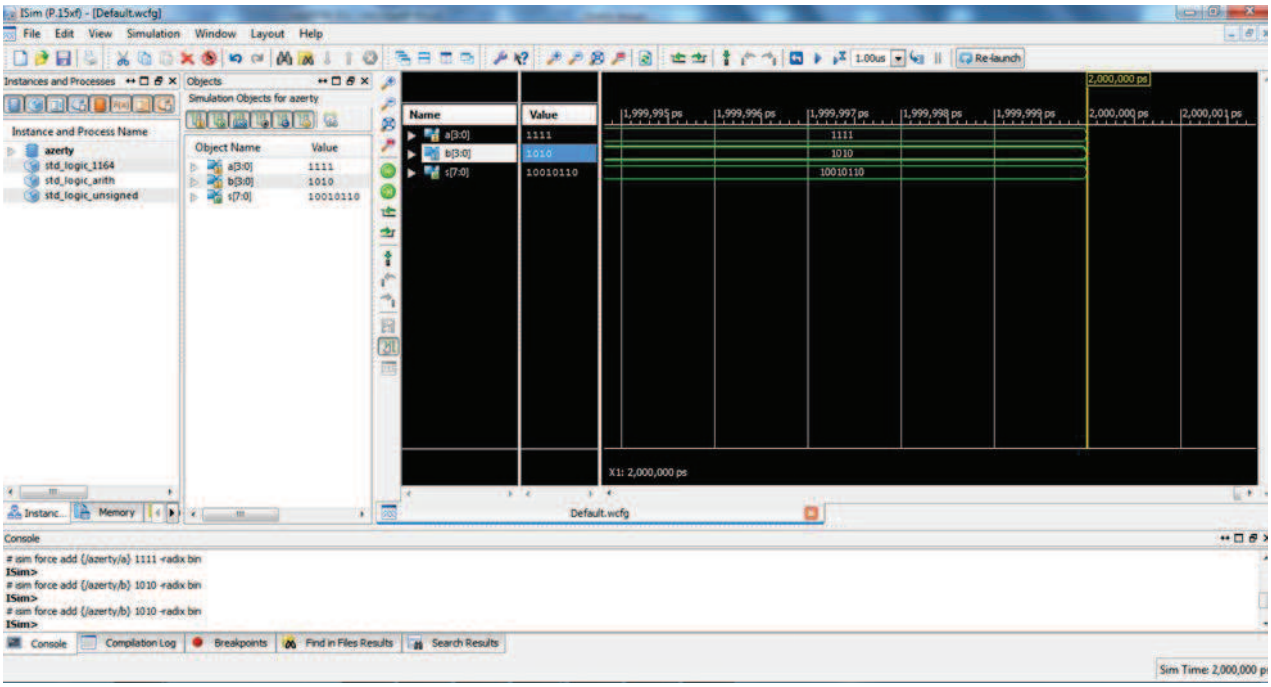


Figure II.14: *Les résultats de simulation*

II.2.4 Conclusion

Dans ce chapitre nous avons donné une description des modes d'utilisation des outils logiciels qu'on aura à utiliser dans ce travail.

Au cours d'une première partie, nous avons passer en revu les principales caractéristiques du langage de programmation hardware : VHDL. Nous avons souligné les concepts de base liés aux procédures de déclaration et les syntaxes des instructions les plus utilisées tout en donnant un exemple du programme en VHDL fait durant le stage avec son simulation.

Ensuite nous avons effectué un bref survol sur les modes d'utilisation du logiciel ISE (Integrated Software Environment) de Xilinx. Les notions évoquées couvraient les étapes de design flow, comment créer un nouveau projet et comment faire la simulation d'un fichier source VHDL et en fin on a donnée un petit exemple de multiplication de deux nombres binaire avec sa simulation.

CHAPITRE III

Implémentation de différents modules sur la carte FPGA

III.1 Introduction

Lors de ce chapitre, nous passons à la description effective du travail que nous avons réalisé pour la mise en œuvre finale de la tâche qui nous a été assignée dans le cadre d'un PFE. Nous décrirons l'implémentation hardware, sur circuit FPGA, des différents blocs fonctionnels requis pour l'acquisition et le traitement des données obtenues via les deux entrées analogiques disponibles sur le module d'acquisition P-240.

III.2 Implémentation hardware des modules d'acquisition, de traitement et d'affichage de données

Conformément au schéma bloc de la figure ci-dessous, on voit bien que les données (véhiculées par les signaux électriques générés par la source de tension) sont transportés à l'unité de traitement et l'affichage de données (qui est le circuit FPGA dans notre cas) via le module d'acquisition analogique représenté par les entités fonctionnelles : Amplificateur du signal, filtrage d'anti-repliement, convertisseur analogique numérique CAN.

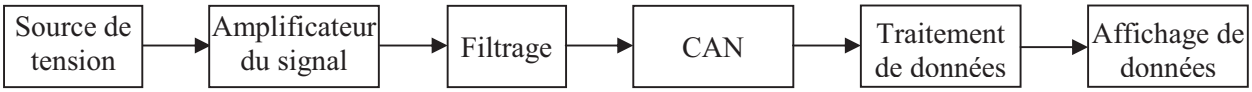


Figure III.1 *la chaine d'acquisition utilisée*

Donc, la réalisation des tâches d'acquisition, de traitement et d'affichage de données se traduira par l'implémentation sur FPGA des modules suivants :

III.3 Une unité d'acquisition et de traitement de données

Ce module électronique est implémenté au sein du circuit FPGA. Il réalise la tâche d'acquisition de données via des buffers d'entrés et fait ensuite le traitement de toutes les données venant du monde extérieur tout en passant par le convertisseur analogique numérique placé sur le module analogique P-240.

A noter ici, que dans un premier temps, le traitement visé ne concernait qu'un simple filtrage numérique (un filtre passe-band dans ce cas de figure III.2). Le filtre en question a été synthétisé à l'aide de l'outil de synthèse des filtres numériques (Filter Tools) offert par le fameux logiciel Matlab. Une fois le filtre synthétisé, par un choix adéquat de paramètres de ce filtre (voir figure ci-dessous):

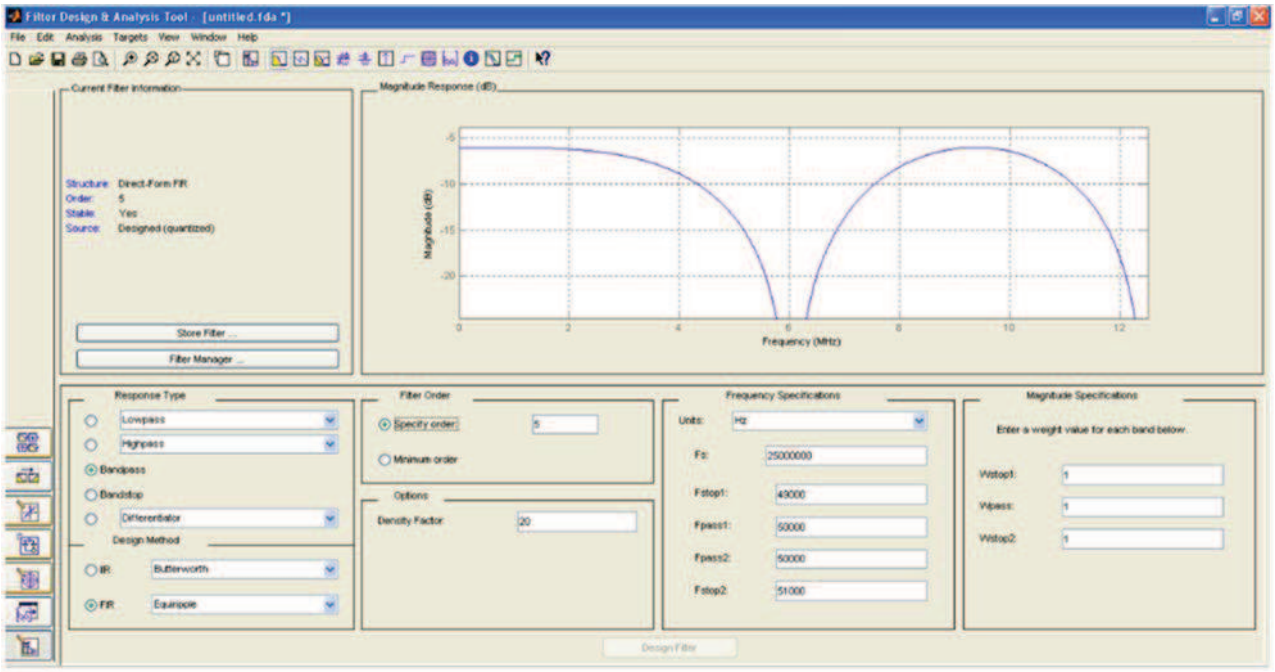


Figure III.2 : *les paramètres du filtre utilisé*

Nous avons utilisé un utilitaire de conception intégré aux nouvelles versions de Matlab (HDL Co-design) pour générer automatiques le code VHDL correspondant au filtre qui a été synthétisé en dessus. Pour générer un code VHDL à partir Matlab on clique sur generate HDL de l'onglet Target dans la barre de menu.

Le code ainsi obtenu est inséré dans une fenêtre des modules dans le logiciel ISE pour donner naissance à un filtre numérique passe-bas en sa version d'implémentation hardware.

III.4 Une unité d'affichage de données

A la sortie de l'unité précédente, figurera l'unité de formatage et d'affichage de données. Elle nous a permet de voir les résultats de l'acquisition et de traitement sur un afficheur LCD 16x2.

L'affichage en caractères ASCII des entités binaires obtenues à la sortie de l'unité de traitement se fait en ajoutant un bloc de conversion binaire/décimal.

A noter ici que l'affichage en binaire ne nécessite aucune conversion, c'est-à-dire après la conversion analogique numérique et le traitement numérique, on fait directement l'affichage, après une certaine adaptation pour un afficheur LCD 16x2, cela nous permet de voire la valeur réelle en binaire.

III.4.1 Le schéma fonctionnel de l'afficheur LCD 16x2

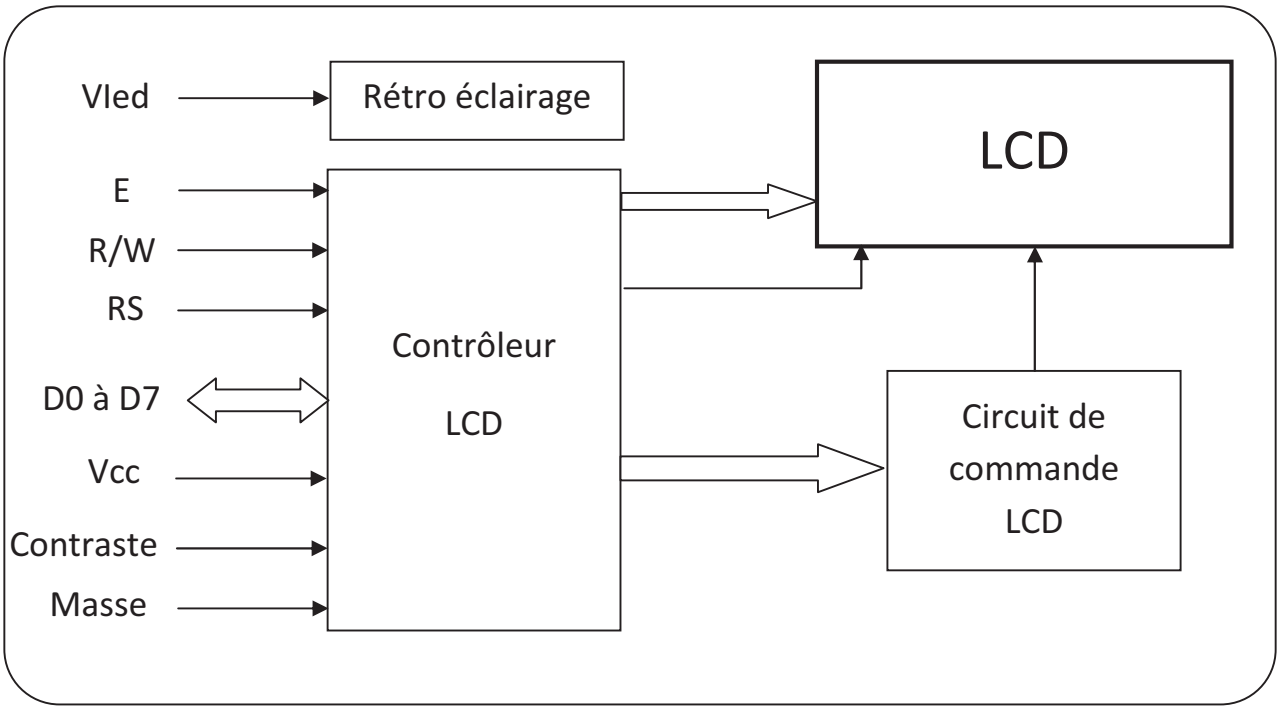


Figure III. 3 : *schéma fonctionnel de l'afficheur LCD16x2*

III.4.2 Le rôle de différentes broches

- **Vcc, Mass :** Pin réservée à l'alimentation de l'afficheur LCD, il s'alimente en 0V – 5V.
- **Contraste :** entrée permettant de régler le contraste de l'afficheur LCD. Il faut appliquer une tension continue réglable entre (0V et 5V) à l'aide d'un potentiomètre.
- **VIed :** différence de potentiel permettant de commander le rétro éclairage.
- **E :** entrée de validation **(ENABLE)**, elle permet de valider les données sur un **front descendant.** Lorsque E=0 alors le bus de données est à l'état haute impédance.
- **RS : registre selecte** cette entrée permet d'indiquer à l'afficheur si l'on souhaite réaliser une commande **(RS=0)** par des instructions spécifique ou écrire une données (envoi du cod du caractère à afficher) sur le bus **(RS=1).**
- **R/W :** entrée de lecture **(R/W=1)** et d'écriture **(R/W=0)**. Lorsqu'on commande l'afficheur LCD il faut se placer en écriture.
- **D7 … D0 : bus de données bidirectionnel,** il permet de transférer les instructions ou les données à l'afficheur LCD.

III.5 Description de l'architecture interne des modules conçus

Dans cette partie, il sera question de décrire l'architecture interne des différents modules fonctionnels implémentés au sein du circuit FPGA. Ceci sera fait à travers l'examen des différents blocs VHDL de tout le projet.

Rappelons à ce début que pour tous les modules, nous avons utilisé un signal d'horloge (clk) pour activer les différentes opérations ainsi que pour la synchronisation entre les différents modules.

III.5.1 Description du circuit globale

En vue de l'extérieur, le circuit global renferme deux entrées : Celles-ci correspondent au sorties des deux convertisseurs CAN 16 bits implantés sur le module analogique P-240. Avant de passer à l'unité de traitement proprement dite, les signaux électriques acquis à travers ces deux entrées (constituent les données numériques à acquérir et à traiter) de 16 bits chacun transitent par deux blocs : buffer1 et buffer2. Ces deux derniers sont utilisés pour l'acquisition synchrone des deux signaux. Ces deux derniers, subiront par la suite un traitement (filtrage numérique) avant d'être affichés sur un afficheur LCD. Les signaux de données et de commande de ce dernier représentent les signaux de sortie du module global.

Notons ici que les données fournies par les convertisseurs ADC sont sous forme d'entités binaire en code compléments à deux. C'est pour ces raison que nous avons également implémenté des blocs de décodage pour faire la conversion entre les différents codes et du binaire en ASCII. Pour cela nous avons divisé les valeurs acquises de 14 bits chacune en deux parties distinctes: Une partie entière composée de 04 bits du poids le plus forts et une partie fractionnaire composée de 9 bits.

Les entrées et les sorties du module global ainsi que leurs descriptions sont montrés dans le tableau suivant :

Lcd16x2_ctr_demo			
Port		**Description**	**Schéma bloc du module**
Les entrées	Clk	Signal d'horloge (freq = 125 MHZ)	
	Clk_adc1	le signal d'horloge utilisé par l'ADC1 (1bit)	
	Clk_adc2	le signal d'horloge utilisé par l'ADC2 (1bit)	
	Sortie_adc1	la donnée convertie venant de l'ADC 1(14 bits)	
	Sortie_adc2	la donnée convertie venant de l'ADC 2 (14 bits)	
Les sorties	Lcd_db	le bus de donnée de l'afficheur LCD (4bits)	
	Lcd_e	le signal enable pour l'afficheur LCD (1 bits)	
	Lcd_rs	le signal registre select (1 bits)	
	Lcd_rw	le signal read/write (1 bits)	
	Oe_adc1	le signal enable pour l'ADC1 (oe_adc1 <= '1')	
	Oe_adc2	le signal enable pour l'ADC2 (oe_adc2 <= '1')	
	Rs_adc1	le signal reset pour l'ADC1 (1 bits)	
	Rs_adc2	le signal reset pour l'ADC2 (1 bits)	

Tableau III.1: *Description des signaux Entrées/Sorties du circuit global*

La figure suivante montre l'architecture du circuit global et les sous-modules qui le composent.

Le schéma RTL de cette architecture est présenté dans (ANNEX A)

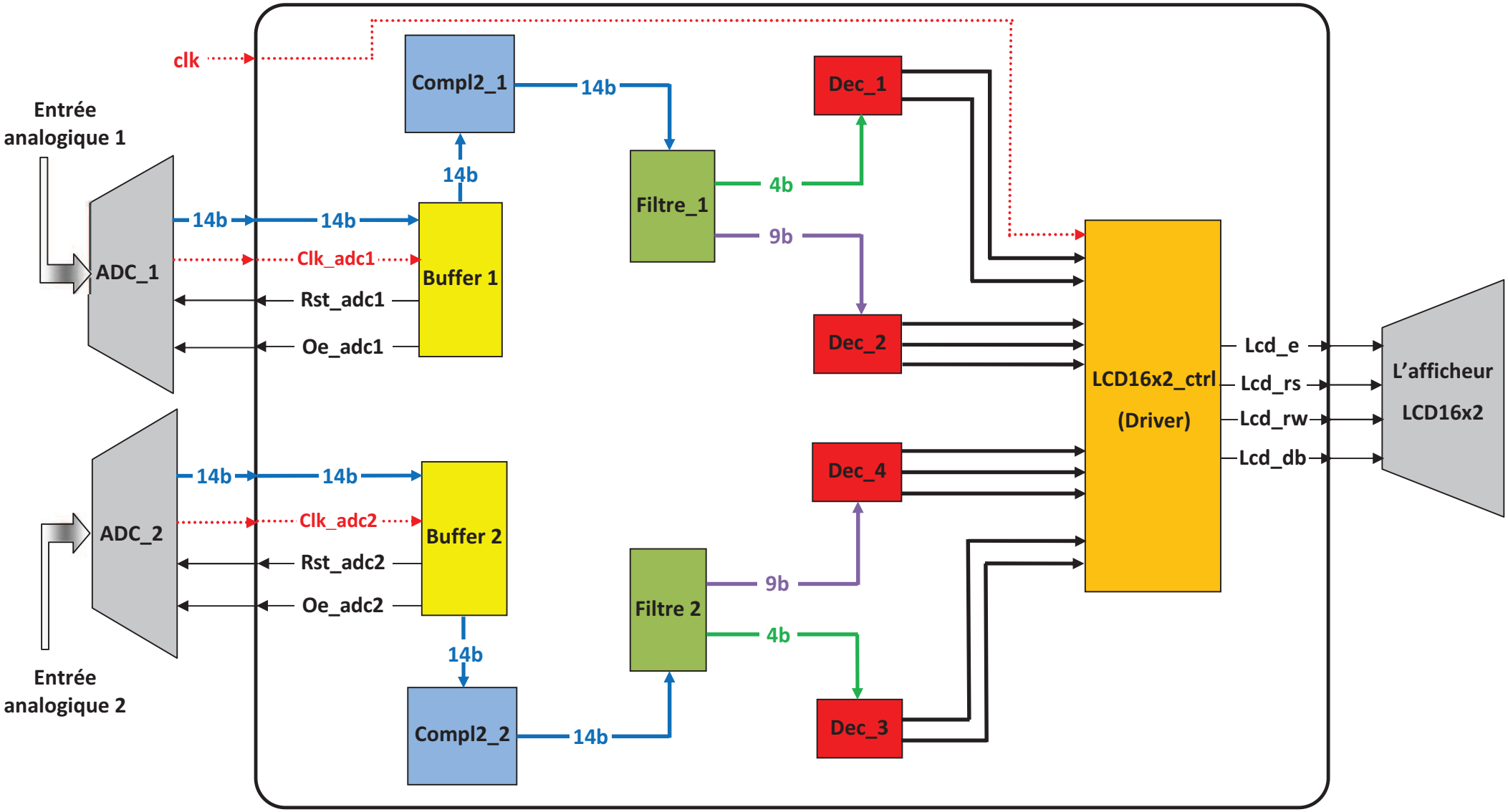


Figure III.4: l'architecture de circuit globale et leur sous modules

III.5.2 Architecture des différents sous-modules composant le module global

Le module global inclus les différents sous-modules suivants :

1. Le module : buffer1,
2. Le module : buffer2,
3. Le module : compl2_1,
4. Le module : compl2_2,
5. Le module : filtre_1,
6. Le module : filtre_2,
7. Le module : dec_1,
8. Le module : dec_2,
9. Le module : dec_3,
10. Le module : dec_4,
11. le module : lcd16x2_ctrl.

III.5.2.1 Description du bloc buffer1

Nous avons réalisé ce bloc dans le but de faire entrer les signaux (les données) électriques convertis par le premier convertisseur (ADC1) depuis le module P-240 vers le circuit FPGA. A chaque front montant de l'horloge liée au convertisseur, celui-ci fourni une donnée (signal sur 14 bits) sur ces pins de sortie. Pour acquérir ces données sur FPGA d'une manière synchrone (vis-à-vis des signaux de contrôle manipulé par celui-là, on utilisera les données stockées momentanément dans ce type de buffer. Les signaux d'entrées/sorties (Données et signaux de contrôle) de ce module sont illustrés dans le tableau suivant :

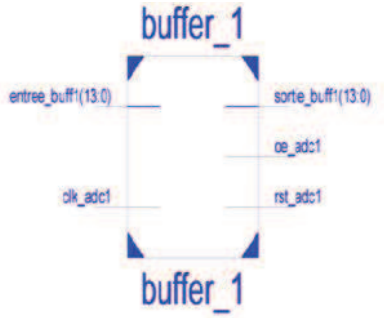
Buffer_1			
Ports		**Description**	**Schéma bloc du module**
Les entrées	Entree_buffer1	Entrée venant du convertisseur ADC1 (14bits)	
	Clk_adc1	Signal d'horloge (freq = 25 MHZ)	
Les sorties	Sortie_buff1	La sortie de buffer (14bits)	
	Oe_adc1	Out enable (1bit)	
	Rst_adc1	Remise à zéro de buffer (1bit)	

Tableau III.2: *Description des signaux Entrées/Sorties du module buffer_1*

III.5.2.2 Description du bloc buffer2

Ce bloc réalise la même fonction que celle décrite en haut pour les signaux issus du deuxième convertisseur ADC2. Les signaux d'entrées/sorties de ce module sont illustrés dans le tableau suivant :

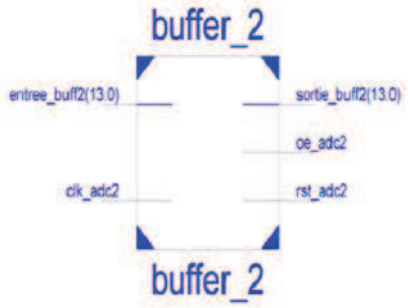
Buffer_2			
Ports		**Description**	**Schéma bloc du module**
Les entrées	Entree_buffer1	Entrée venant du convertisseur ADC2 (14bits)	
	Clk_adc2	Signal d'horloge (freq = 25 MHZ)	
Les sorties	Sortie_buff2	La sortie de buffer (14bits)	
	Oe_adc2	Out enable (1bit)	
	Rst_adc2	Remise à zéro de buffer (1bit)	

Tableau III.3: *Description des signaux Entrées/Sorties du module buffer_2*

III.5.2.3 Description du bloc compl2_1 :

Ce bloc a pour but de convertir la valeur binaire issue de l'ADC1 (codée en complément à deux sur 14 bits), en code binaire simple. Les signaux d'entrées/sorties de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

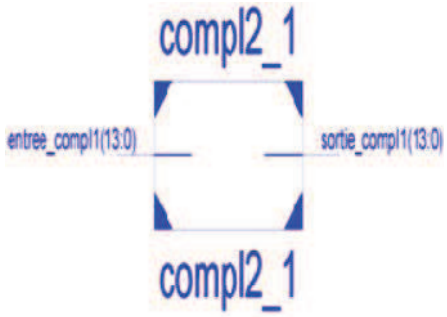
Compl2_1			
Ports		**Description**	**Schéma bloc du module**
Les entrées	Entree_compl1	L'entrée de complément (14bits)	
Les sorties	Sortie_compl1	La sortie de complément (14bits)	

Tableau III.4: *Description des signaux Entrées/Sorties du module compl2_1*

III.5.2.4 Description du bloc compl2_2

La même chose comme le module coml2_1, ce module fait la conversion de complément à deux en binaire simple mais au niveau des signaux de sortie du convertisseur CAN2.

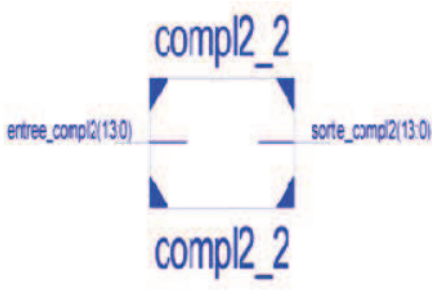
Compl2_2			
Ports		**Description**	**Schéma bloc du module**
Les entrées	Entree_compl2	L'entrée de complément (14bits)	
Les sorties	Sortie_compl2	La sortie de complément (14bits)	

Tableau III.5: *Description des signaux Entrées/Sorties du module compl2_2*

III.5.2.5 Description du bloc filtre_1

Ce module est un filtre numérique, qui a pour objectifs de filtrer le premier signal issu du premier convertisseur ADC1 et codée sur 14bits. Les signaux d'entrées/sortis de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

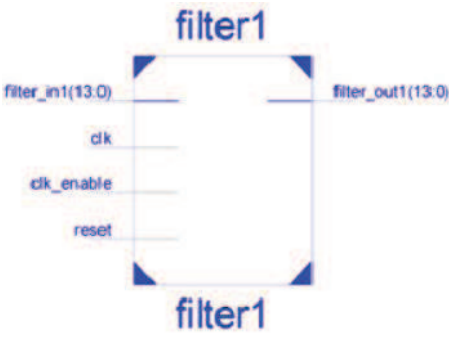
filter_1			
Ports		**Description**	**Schéma bloc du module**
Les entrées	Filter_in1	L'entrée de filter1 (14bits)	
	Clk	Signal d'horloge	
	Clk_enable	Signal d'horloge enable	
	Reset	Signal reset	
Les sorties	Filter_out1	La sortie de filter1 (14bits)	

Tableau III.6: *Description des signaux Entrées/Sorties du module filter_1*

III.5.2.6 Description du bloc filtre_2

Ce module est un filtre numérique, qui a pour objectifs de filtrer notre deuxième signal codé sur de 14 bits. Les signaux d'entrées/sorties de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

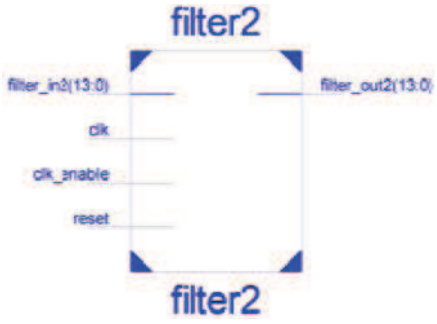
filter_2			
Ports		**Description**	**Schéma bloc du module**
Les entrées	Filter_in2	L'entrée de filter2 (14bits)	
	Clk	Signal d'horloge	
	Clk_enable	Signal d'horloge enable	
	Reset	Signal reset	
Les sorties	Filter_out2	La sortie du filter2 (14bits)	

Tableau III.7: *description des signaux Entrées/Sorties du module filter_2*

III.5.2.7 Description du bloc dec_1

Ce bloc fonctionnel a pour objectif de faire la conversion de la donnée courante, issue du premier convertisseur ADC1, du binaire classique vers le décimal. Pour cela, ce module considère 4bits, comme entrée, de la valeur totale, cette valeur va représenter la valeur entière de la deuxième valeur affichée. Les signaux d'entrées/sorties de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

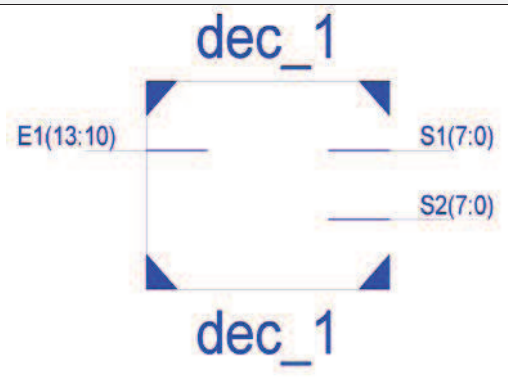
Dec_1			
Ports		**Description**	**Schéma bloc du module**
Les entrées	E1	L'entrée du décodeur (4bits)	
Les sorties	S1	La sortie 1 du décodeur (8bits)	
	S2	La sortie 2 du décodeur (8bits)	

Tableau III.8: *Description des signaux Entrées/Sorties du module dec_1*

III.5.2.8 Description du module dec_2

Ce bloc à pour objectifs de faire la conversion binaire décimal de la première valeur de CAN1. Pour cela, ce module prend en charge 8 bits de la valeur totale considérée, cette valeur va représenter la valeur fractionnaire de la première valeur affichée. Les signaux d'entrées/sortis de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

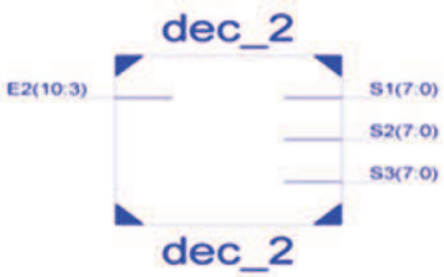
Dec_2			
Ports		**Description**	**Schéma bloc du module**
Les entrées	E2	L'entrée du décodeur (8bits)	
Les sorties	S1	La sortie 1 du décodeur (8bits)	
	S2	La sortie 2 du décodeur (8bits)	
	S3	La sortie 3 du décodeur (8bits)	

Tableau III.9: *Description des signaux Entrées/Sorties du module dec_2*

III.5.2.9 Description du bloc dec_ 3

Ce bloc à pour objectifs de faire la conversion binaire décimal de la deuxième valeur de ADC2. Pour cela ce module considère 4bits de la valeur totale considérée, cette valeur va représenter la valeur entière de la deuxième valeur affichée. Les signaux d'entrées/sortis de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

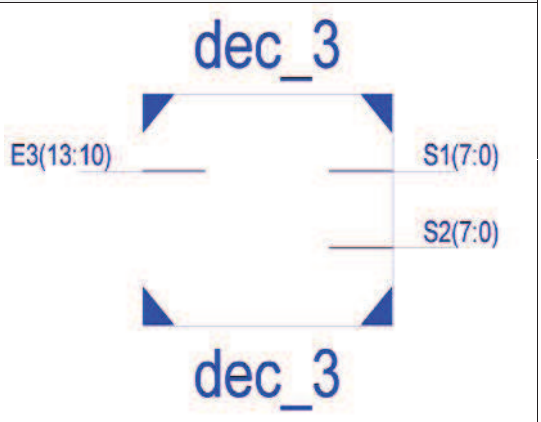
Dec_3			
Ports		**Description**	**Schéma bloc du module**
Les entrées	E3	L'entrée du décodeur (4bits)	
Les sorties	S1	La sortie 1 du décodeur (8bits)	
	S2	La sortie 2 du décodeur (8bits)	

Tableau III.10: *Description des signaux Entrées/Sorties du module dec_3*

III.5.2.10 Description du bloc dec_4

Ce bloc à pour objectifs de faire la conversion binaire décimal de la première valeur de CAN2. Pour cela, ce module prend en charge 8 bits de la valeur totale considérée, cette valeur va représenter la valeur fractionnaire de la première valeur affichée. Les signaux d'entrées/sortis de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

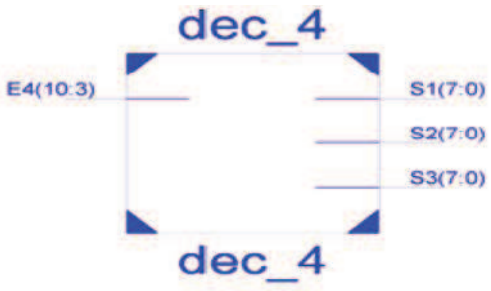
Dec_4			
Ports		**Description**	**Schéma bloc du module**
Les entrées	E4	L'entrée de décodeur (8bits)	
Les sorties	S1	La sortie 1 de décodeur (8bits)	
	S2	La sortie 2 de décodeur (8bits)	
	S3	La sortie 3 de décodeur (8bits)	

Tableau III.11: *Description des signaux Entrées/Sorties du module dec_4*

III.5.2.11 Description du bloc lcd16x2_ctrl

Ce bloc représente le driver hardware de notre afficheur LCD16x2. C'est le dernier module de la chaîne du traitement considéré. Les signaux d'entrées/sortis de ce module ainsi que le schéma bloc sont illustrées dans le tableau suivant :

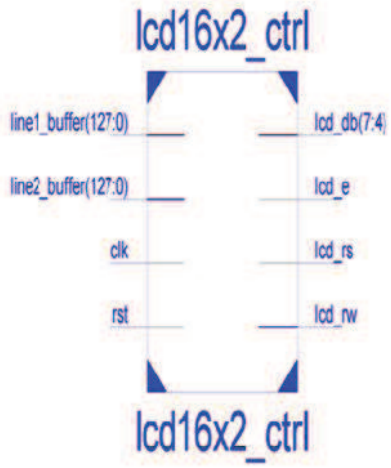
Lcd16x2_ctrl			
Ports		**Description**	**Schéma bloc du module**
Les entrées	Line1_buffer	Line buffer 1	
	Line2_buffer	Line buffer 2	
	Clk	Signal d'horloge (freq = 25 MHZ)	
	rst	Le signal reset	
Les sorties	Lcd_db	le bus de donnée de l'afficheur LCD (7bits)	
	Lcd_e	le signal enable pour l'afficheur LCD (1 bits)	
	Lcd_rs	le signal registre select (1 bits)	
	Lcd_rw	le signal read/write	

Tableau III.12: *Description des signaux Entrées/Sorties du module Lcd16x2_ctrl*

III.6 conclusion

Dans ce chapitre, nous avons donné une description détaillée de l'architecture globale des différents modules d'acquisition, de traitement et d'affichage développée pendant ce travail en utilisant les outils de développement software.

CHAPITRE IV

Les résultats pratiques

IV.1 Les résultats de simulation et expérimentaux

Après avoir franchi l'étape de conception et de réalisation des différentes parties du travail demandé, nous passons lors de ce chapitre à la phase de la validation expérimentale. Mais, avant de passer à l'expérimentation proprement dite, nous avons jugé utile d'utiliser les outils de simulation software pour accomplir une validation préliminaire de toutes les entités considérées. Comme outil de simulation logicielle, nous avons utilisé le simulateur : Isim de ISE de Xilinx 14.1 et le ModelSim 6.5 de la firme Mentor Graphics.

IV.2 Les instruments utilisés lors de la validation expérimentale

IV.2.1 source des signaux d'entrée

C'est un générateur de fonction qui fournit des signaux de fréquence différentes et amplitudes qui varient entre 0 et 3,2 V. Nous avons injecté ces signaux-là dans les deux entrées analogiques du kit P240.

IV.2.2 Le kit d'acquisition de données P240

Ce kit permet d'acquérir deux signaux analogiques via deux entrées à double canal. Il contient deux convertisseurs analogiques numériques que nous avons utilisés pour la conversion des deux signaux électriques considérés.

IV.2.3 Le kit de développement Virtex-4 MB

Ce kit fournit une plate-forme complète de développement pour concevoir et vérifier des applications de traitement numérique des signaux basées sur la famille de Xilinx Virtex-4 FPGA. En plus du circuit FPGA Virtex-4, ce kit contient aussi un afficheur LCD 16x2 que nous avons utilisé pour voire notre valeur convertie.

IV.2.4 L'oscilloscope

On a utilisé l'oscilloscope seulement pour voir la forme notre signal analogique à l'entrée du module P-240**.**

IV.2.5 Le multimètre

Utilisé pour visualiser et comparer la valeur réelle de tension injectée et la tension affichée sur l'afficheur LCD.

IV.3 les résultats de simulation

IV.3.1 Résultats de simulation pour le circuit global

Le résultat de simulation du circuit global est résumé dans la figure suivante :

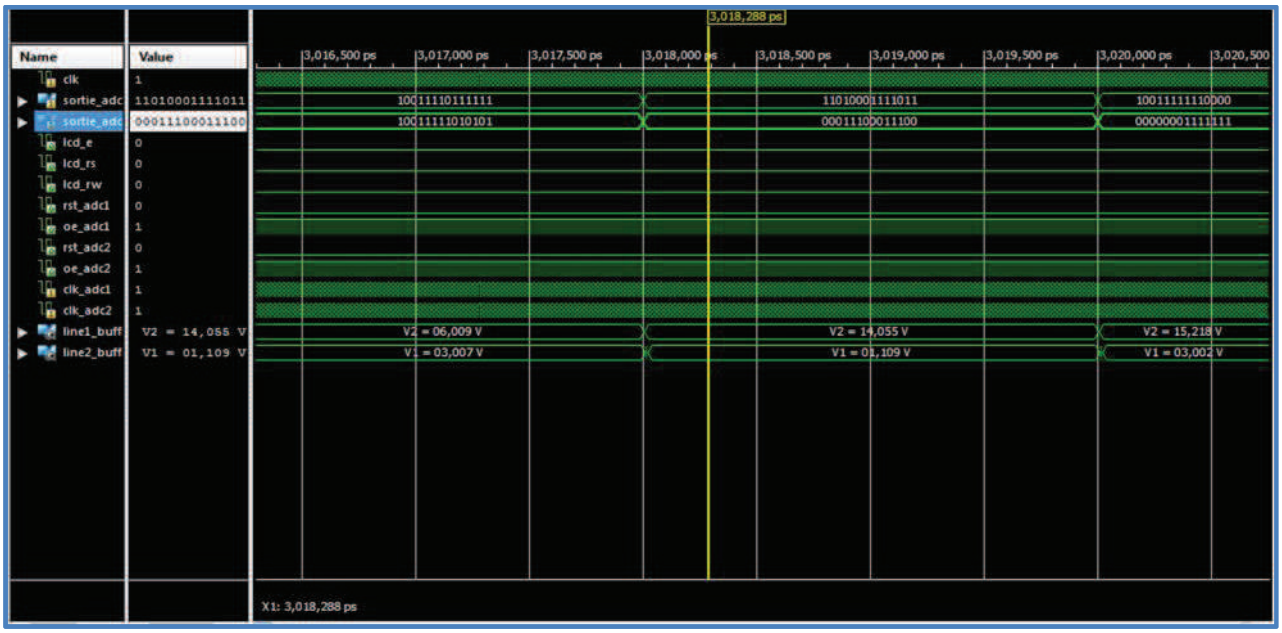


Figure IV.1 : *la simulation du circuit global*

Dans ce chronogramme de simulation de notre circuit global on voit clairement que nous avons injecté des valeurs binaires quelconques à l'entrée du module et nous avons obtenu des valeurs codées en réelles dans la sortie de celui-ci. A noter ici que l'effet du filtrage n'apparait pas et il s'agissait simplement de vérifier la réponse du système global quant à la variation de la donnée à son entrée.

IV.3.2 Simulation du fonctionnement du buffer d'entrée du module global

La simulation de ce module est représentée dans la figure suivante :

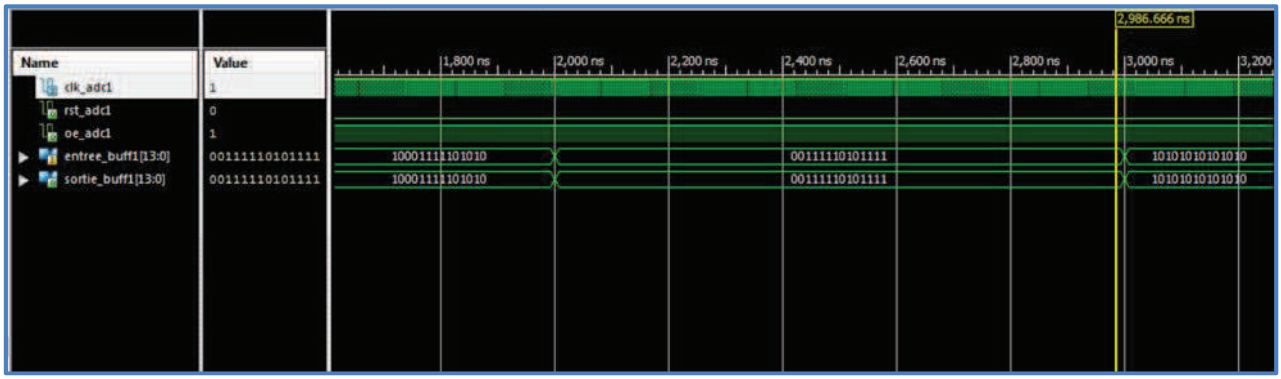


Figure IV.2: *Résultat de simulation du module buffer*

Dans ce schéma on peut constater que si la valeur est disponible au niveau de la sortie de notre convertisseur dans ce moment la notre valeur va transmettre immédiatement au bloc buffer.

IV.3.3 Résultats de simulation pour le module complément à deux

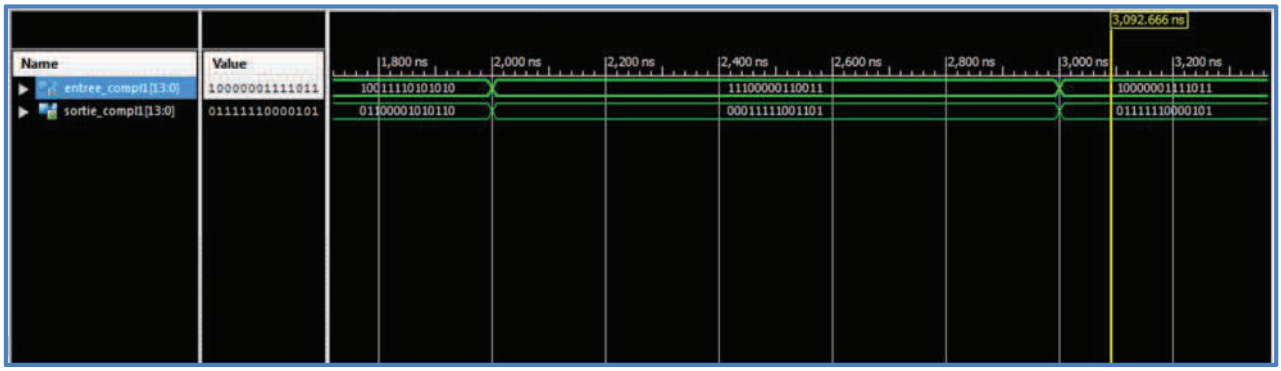


Figure IV.3: *Résultat de simulation du module comp2*

Cette simulation montre que ce bloc permet effectivement de convertir une entité binaire présente à son entrée (codée en binaire complément à deux) en une autre entité binaire (codée en binaire classique. Une telle conversion est indispensable dans la chaîne d'acquisition considérée et ce suite au faite que les ADC utilisés dans le module P-240 fournissent des données binaires codées en complément à deux.

IV.3.4 Résultats de simulations obtenus pour le filtre du traitement numérique

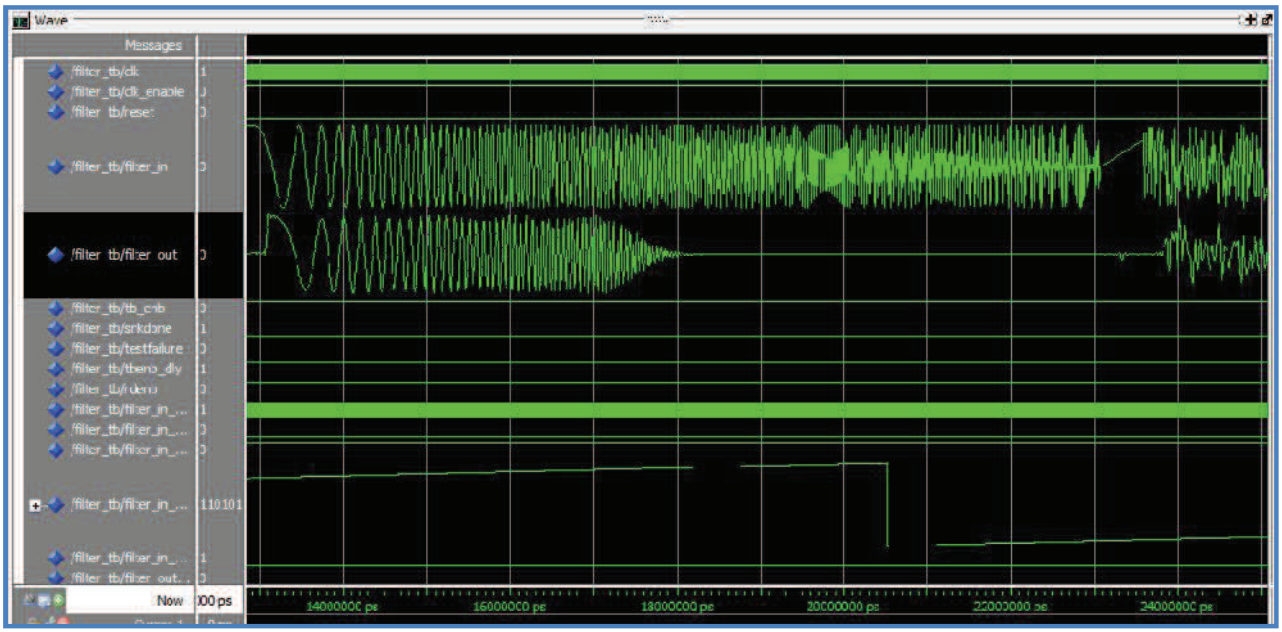


Figure IV.4*: Résultat de simulation du filtre numérique*

On injecte à l'entrée du filtre un signal de fréquence variable (filter_in), on voit bien que la partie correspondant à une haute fréquence a été éliminée complètement en sortie (filter_out) par effet de filtrage passe-bas.

IV.3.5 Résultats de simulation obtenus pour le Décodeur 1

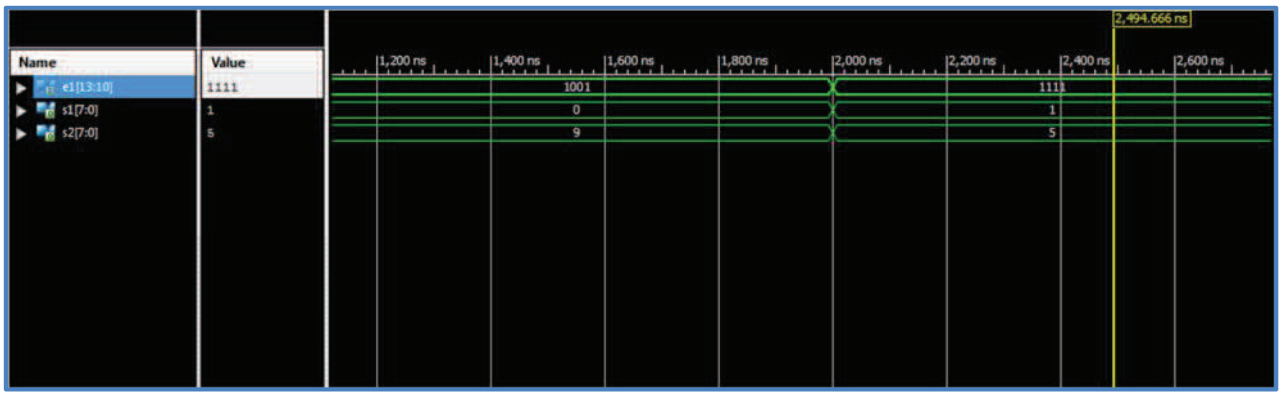


Figure IV.5 : *La simulation du module dec1*

Cette simulation montre que ce module fait la conversion binaire décimale, donc nous avons données des valeurs on binaire et la sorties est en décimale, donc ce module donne la valeur réelle à afficher c'est-à-dire la valeur qui va afficher avant la virgule.

IV.3.6 Résultats de simulations obtenus pour le Décodeur 2

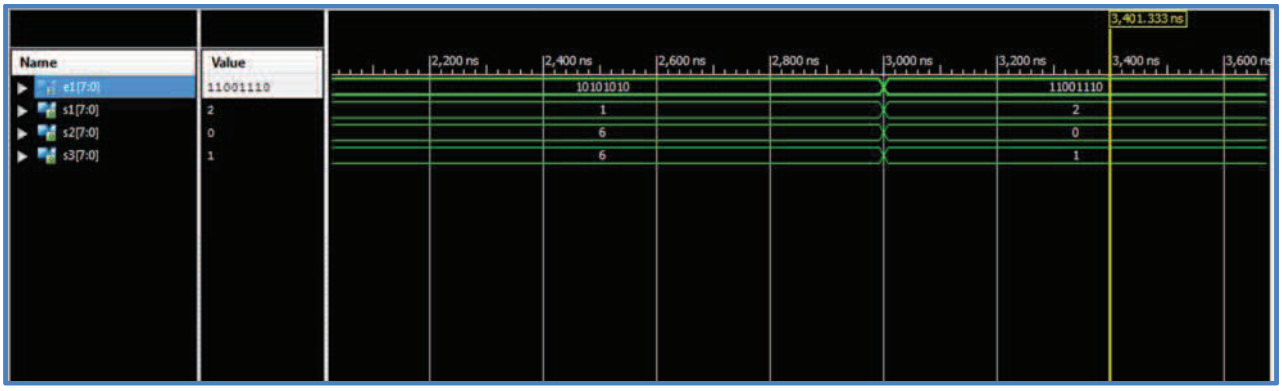


Figure IV.6 : *Résultat de simulation du module dec2*

Cette simulation montre que ce module fait la conversion binaire décimale, donc nous avons données des valeurs on binaire et la sorties est en décimale, donc ce module donne la valeur fractionnelle à afficher c'est-à-dire la valeur qui va afficher après la virgule.

IV.4 Résultats obtenus lors de la phase des tests pratiques

IV.4.1 l'affichage des données acquises en ASCII

Pour valider cette fonctionnalité, nous avons considéré plusieurs opérations de tests :

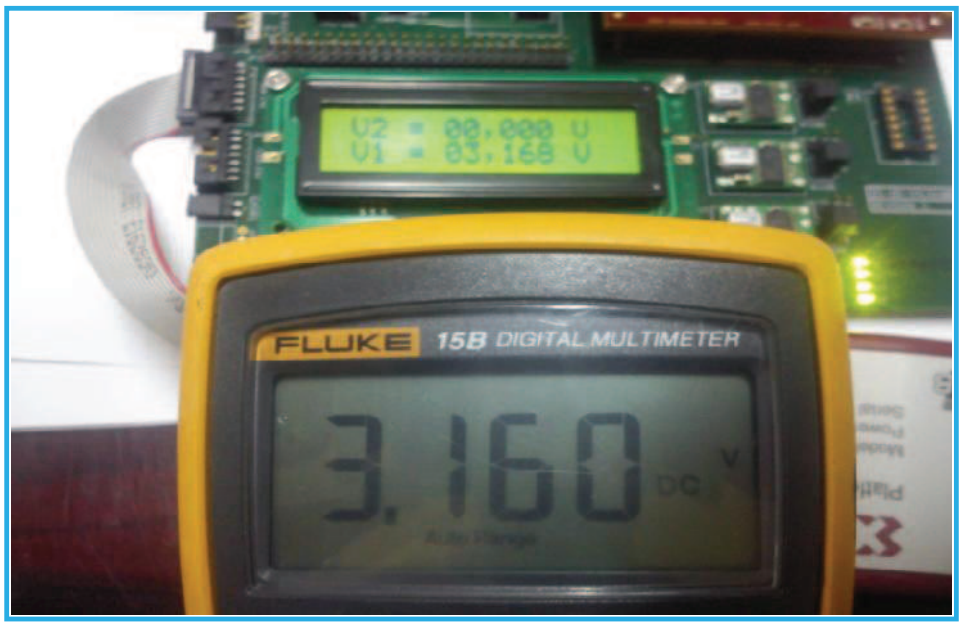


Figure IV.7: *Exemple 1 pour le test pratique*

On voit bien que la valeur affichée par notre système (sur V1) correspond parfaitement à celle mesurée à l'entrée du convertisseur analogique/numérique ADC1. Un autre aspect, qui ne peut être visualisé à travers cette même figure, est que lorsqu'on fait varier la tension d'entrée du convertisseur (source de tension variable), la valeur affichée sur la carte FPGA, suit à merveille celle indiquée sur le multimètre.

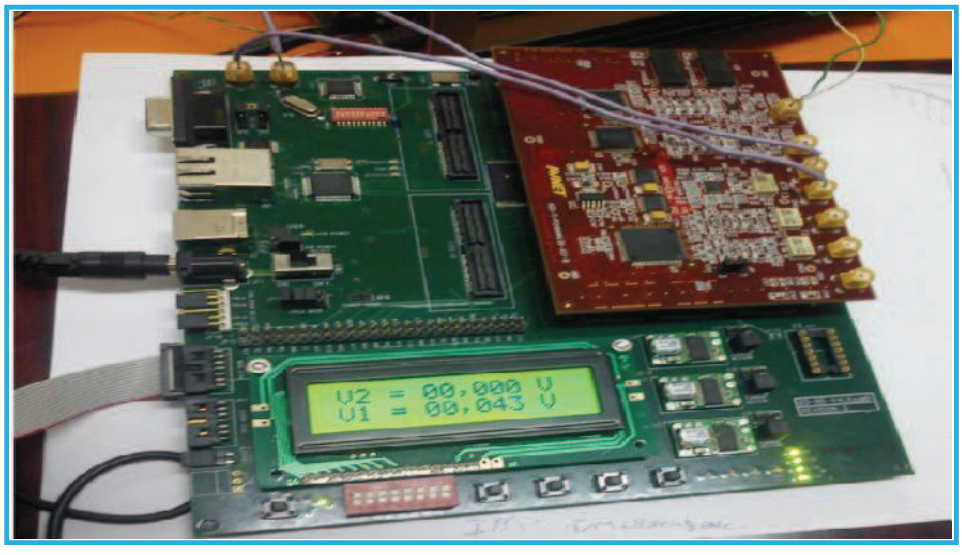


Figure IV.8 : *Exemple 2 pour le test pratique*

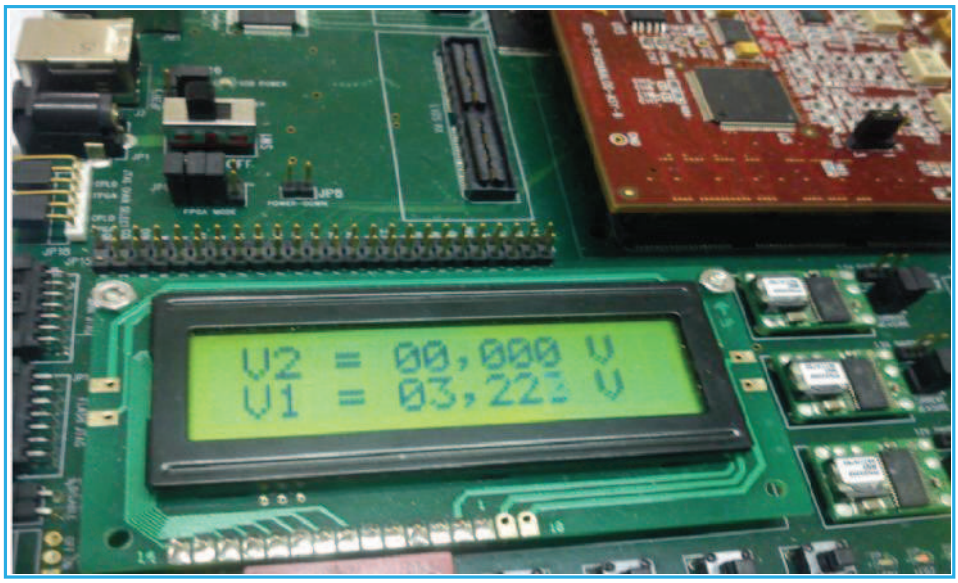


Figure IV.9 : *Exemple 3 pour le test pratique*

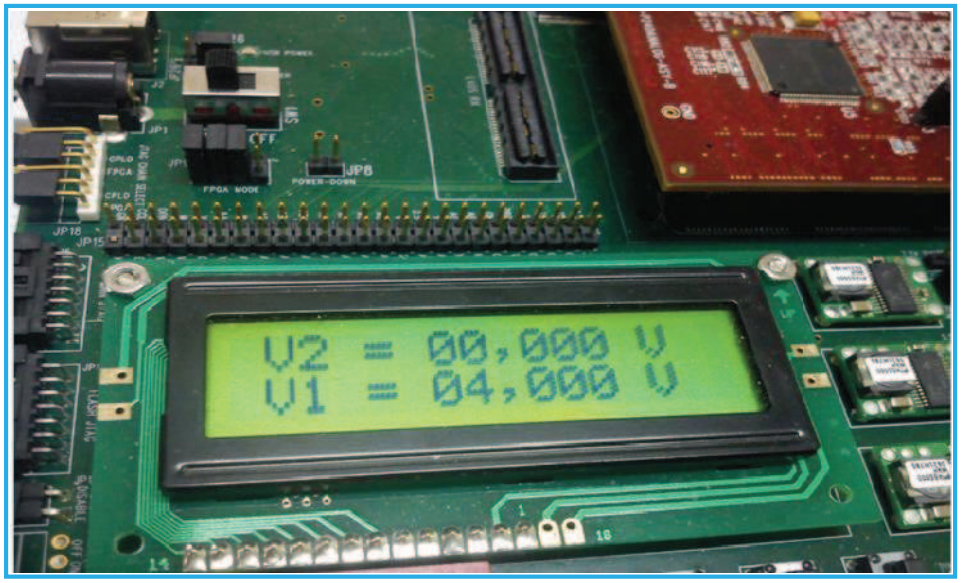


Figure IV.10 : *Exemple 4 pour le test pratique*

Nous avons donné dans les exemples précédents quelque valeur réelle que nous avons obtenue lors des tests pratiques au niveau du laboratoire, ces valeurs sont obtenues en variant la valeur de la tension d'entrée.

IV.4.2 l'affichage en binaire

Pour l'affichage des données en binaire (codées en binaire classique), on a pris deux exemples :

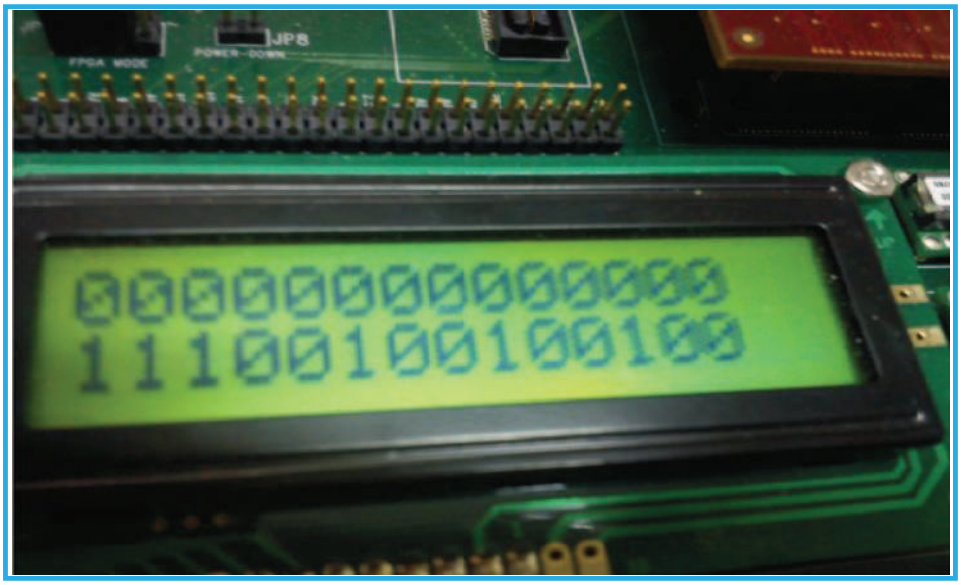


Figure IV.11 : *Exemple1 pour l'affichage en binaire*

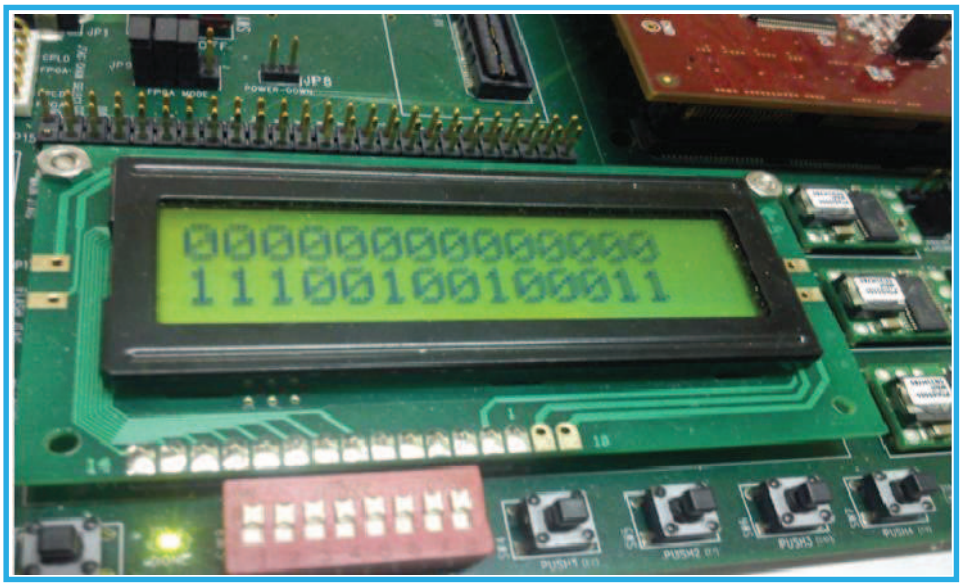
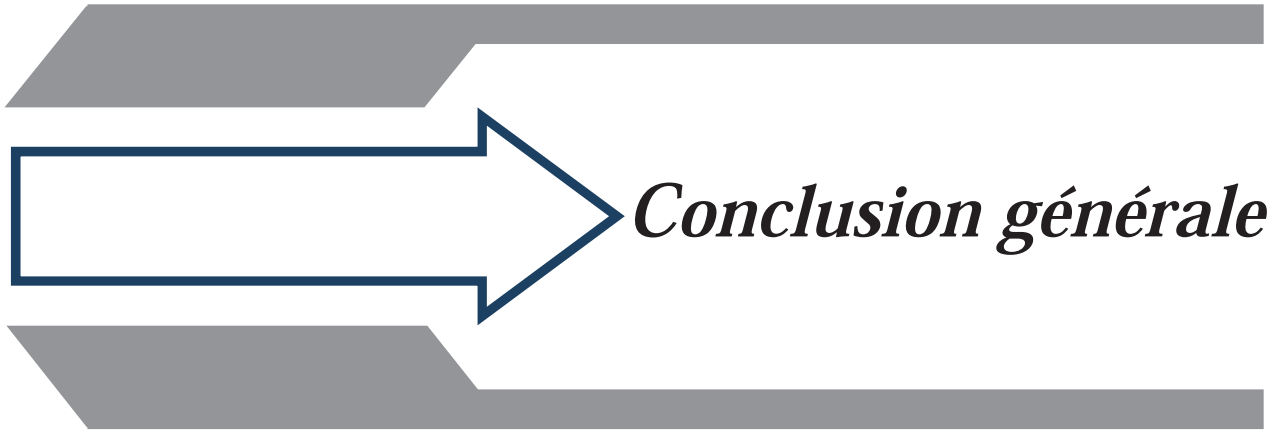


Figure IV.12 : *Exemple 2 pour l'affichage en binaire*

Dans ces deux exemples, on a procédé de la même manière que précédemment sauf que le module de conversion binaire/ASCII a été éliminé de la chaîne de traitement considérée.

IV.5 Conclusion

Dans ce chapitre nous avons donné les résultats des tests de simulation et des tests pratiques qu'on a faits au niveau de laboratoire d'électronique. Les résultats obtenus confirment bien l'aboutissement de notre travail. Et c'est ainsi qu'on affirme que l'implémentation des différents modules d'acquisition, de traitement et d'affichage de données sur la carte VERTX-4MB ont été mené à terme.



Conclusion générale

Conclusion Générale

Le travail réalisé dans le cadre de ce PFE concernait en premier lieu : l'étude et la compréhension plus au moins approfondie des caractéristiques technique d'un support hardware donnée comprenant un Kit pédagogique « Virtex-4MB » bâtit à la base du circuit numérique programmable de type FPGA et une interface d'acquisition de données associée au Kit considéré. En deuxième lieu, ce matériel devait être utilisé pour la conception et l'implémentation hardware d'une chaîne d'acquisition, de traitement et d'affichage de données « deux voies analogiques ». Pour arriver à concrétiser un tel système, nous avons tout d'abord étudié et analysé le support hardware mis à notre disposition. Et nous avons utilisé les logiciels de développement et de simulation tels : Matlab, ISE, Modelsim, …, pour implanter les différents modules et sous-modules requis dans un contexte pareil.

Au cours des différentes phases de réalisation de ce travail, nous nous sommes investis à fond dans le domaine de l'acquisition de données dans le contexte d'un processus industriel. Pour cela, nous avons utilisé des outils de développement (Hardware & Software) très développés fabriquée par des firmes de renommée mondiale telles : Xilinx, Mentor Graphics, Memec, … etc.

Les résultats obtenus lors de la phase des tests, nous ont permis de confirmer la validité du travail réalisé. On tient également à préciser que les modules ainsi développés peuvent d'ores et déjà être utilisés pour l'acquisition et le traitement de données dans plusieurs processus industriels.

Nous mentionnons aussi par cette fin, que la réalisation de notre projet de fin d'étude au sein du Centre de Recherche Nucléaire de Birine, nous a été très bénéfique sur plusieurs plans à savoir : On a pu développer un esprit de travail d'équipe et de consultation continue, chose qui a accéléré l'aboutissement de notre travail. On a appris à lire et à tirer profit, des documents techniques et des data-sheets de plusieurs équipements et composants électroniques. On s'est également familiarisé avec d'autres outils de développements (que ça soit hard ou soft) présents au sein du laboratoire, qui nous ont été jusqu'ici méconnus. Parmi les concepts et les notions qu'on a pu acquérir, on note bien : La technique d'acquisition de données qui constitue la colonne vertébrale de tous les systèmes de commande industriels contemporains.

On note également que la tâche ainsi réalisée, n'est qu'une partie d'un travail énorme qui consiste en la rénovation de l'instrumentation de mesure et les techniques de commande autour

du réacteur Es-salam. Elle sera suivie par une implémentation de la partie des circuits de commande liés aux actionneurs agissants suite aux variations des données déjà acquises.

Bibliographie

[1] fiche technique de module analogique P240. Version 1.0 15/05/2006, www.Ti.com, téléchargé le 26 Février 2013.

[2] Farida BOUNKAR, Sabrina DJELLOULI. *Système temps réel embarqué pour commander un robot mobile, chapitre III.* Projet de fin d'étude, école national polytechnique Alger ; département de génie électrique juin 2005

[3] M. BIREM Merwan, M. NAïT MEZIANE Mohamed. *Initiation aux circuit FPGA de la famille Virtex II concept de base et architecture pour le traitement de signal; chapitre I.* Rapport de stage école national polytechnique Alger ; département d'électronique 2008/2009

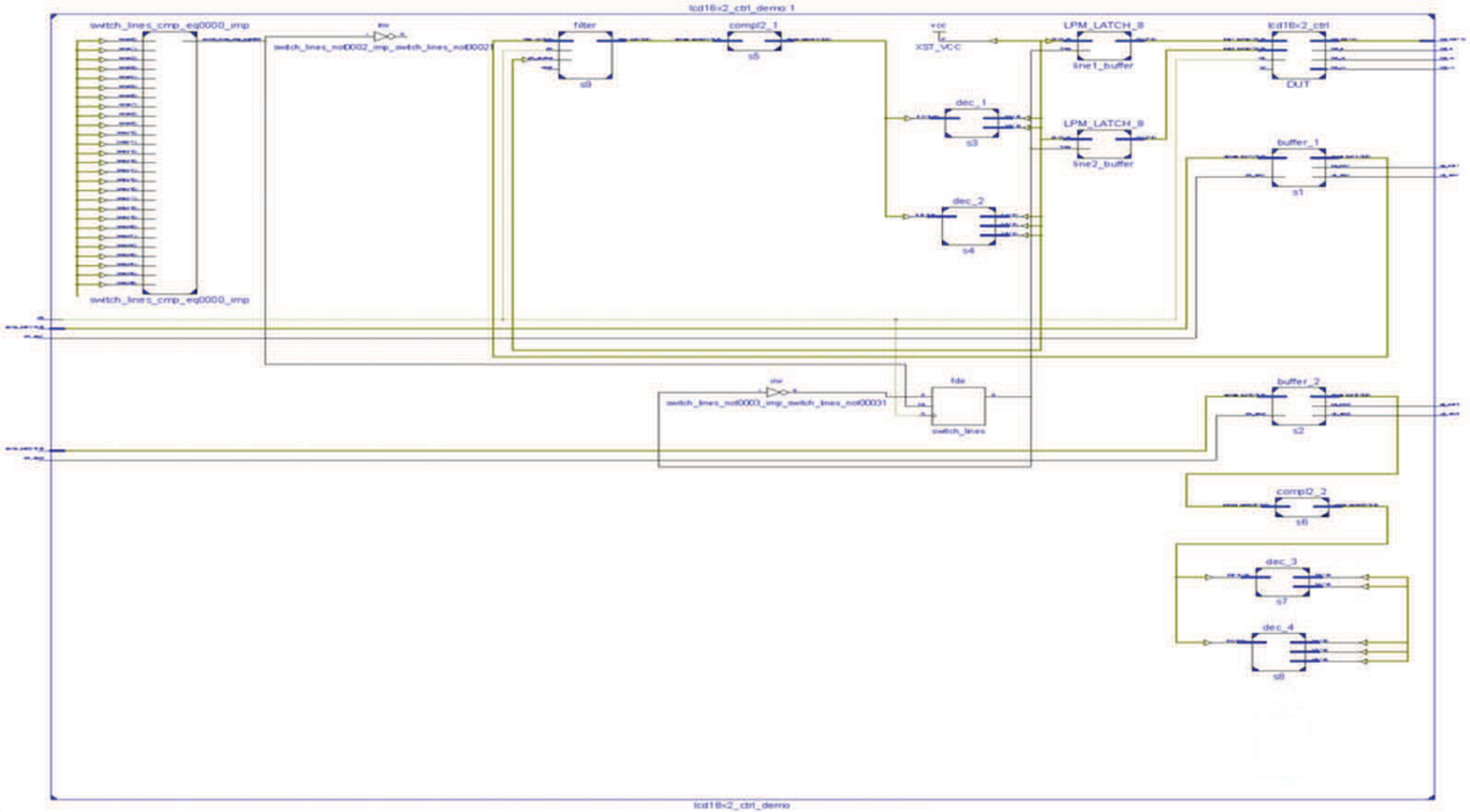
[4] fiche technique *Virtex-4™ MB Development Board User's Guide*.Version 3.0 Décembre 2005, www.xilinx.com, téléchargé le 26 Février 2013.

[5] Philippe LECARDONNEL & Philippe LETENNEUR. *Introduction à la Synthèse logique V.H.D.L*, Lycée Julliot de la Morandière – GRANVILLE – 2001

[6] M.BIREM Merwan, M. NAïT MEZIANE Mohamed. *Initiation aux circuits FPGA de la famille Virtex II concept de base et architecture pour le traitement de signal; chapitre III.* Rapport de stage école national polytechnique alger; département d'électronique 2008/2009

[7] http://acronyms.thefreedictionary.com Consulté le 07 septembre 2013.

ANNEXE A (schéma RTL)



ANNEXE B

Termes et définition [7]

Terme	Définition
DAC	Digital-to-Analog Converter
ADC	Analog to Digital Converter
RLC	Resistance Bobine Condensator
MSPS	Mega-Samples Per Second
LVDS	Low Voltage Differential Signalling
AC	Alternating current
DC	Direct current
SNR	Signal to Noise Ratio
FPGA	Field Programmable Gate Array
NCO	Numerically Controlled Oscillator
LVPECL	Low Voltage Positive Emitter Coupled Logic
HSTL	High Speed Transceiver Logic
CML	Current Mode Logic
LVTTL	Low Voltage Transistor Logic
LVCMOS	Low Voltage Complementary Metal Oxide Semiconductor
FIFO	First In First Out
VCO	Voltage Controlled Oscillator
CLOCK EXT	Clock Extérieur
ASIC	Application-Specific Integrated Circuit
PLD	Programmable Logic Device
SRAM	Static Random Access Memory
CLB	Configurable Logic Block
LCA	Logic Cells Array
IOB	Input/Output Bloc
ROM	Read Only Memory
LUT	Look Up Table
TTL	Transistor Transistor Logic
CMOS	Complementary Metal Oxide Semiconductor
LCD	Liquid Cristal Display
JTAG	Joint Test Action Group

IEEE	Institute of Electrical and Electronics Engineers
LED	Light Emitting Diode
USB	Universal Serial Bus
XSBI	10 Gigabyte Sixteen-Bit Interface
RTS	Request To Send
CTS	Clear To Send
EEPROM	Electrically Erasable Programmable Read Only Memory
HDL	Hardware Description Language
VHDL	Very High Speed Integrated Circuit Hardware Description Language
CPLD	Complex Programmable Logic Device
ISE	Integrated Software Environment
FDIF	Electronic Design Interchange Format
BIT	binary digit