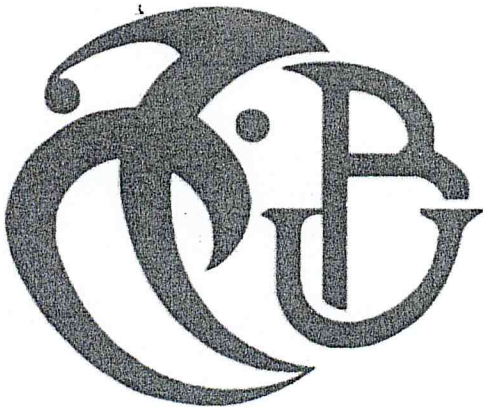
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université Saad Dahlab, Blida
USDB

Faculté des Sciences
Département Informatique

Mémoire pour l'Obtention
Du Diplôme de Master en Informatique
Option : Ingénierie du Logiciel

Sujet:

Réalisation d'une Plateforme HW/SW sur FPGA Dédiée au Contrôle Embarqué

Présenté par:
Med Lamine Berrandjia

Promoteurs :
- Mr Abdelkrim Kamel Oudjida
- Mme Nadjia Benblidia

Soutenu devant le jury :
ABED HAFIDA : Président
Ayeg mahdia : Examinateur
Benghara Maamar : Examinateur

Organisme d'accueil : Centre de Développement des Technologies Avancées

- 2010/2011-

Gloire à Dieu

A mes chers Parents

A ma famille

Remerciements

Je tiens à remercier la direction du Centre de Développement des Technologies Avancées (CDTA) ainsi que le département informatique de l'Université Saad Dahlab de Blida pour tous les efforts qui ont été déployés afin de nous inscrire en Master 2 Recherche.

Je tiens aussi à remercier Mr Oudjida, Chargé de Recherche au CDTA, et Mme Benblidia, Maître de Conférences à l'Université de Blida, d'avoir bien accepté de m'encadrer dans ce projet de Master.

Mes remerciements vont aussi à tous ceux qui m'ont aidé de près ou de loin à concrétiser ce travail.

Résumé

Dans ce Mémoire, nous présentons la mise au point d'une plateforme embarquée sur FPGA dédiée au contrôle et à la régulation des systèmes dynamiques. A cet effet, nous allons montrer à travers les chapitres de ce mémoire les différentes étapes que nous avons suivies pour développer les trois parties essentielles d'un système de contrôle ou de régulation, en l'occurrence : la partie de commande, la partie d'acquisition et la partie de contrôle. Cette dernière est réalisée par un PID (Proportionnel Intégral Dérivé) implémenté sous forme d'un IP. Comme cas pratique permettant la validation de notre système, nous avons implémenté ce dernier sur la plateforme Memec V2MB1000 qui contrôle un dispositif de régulation de température. Une Application software sur PC qui permet de modifier la consigne et de visualiser l'état du système à contrôler a été aussi développée.

Mots-clés— Capteur; Contrôle; Contrôle Embarqué(CE); Régulation; Field Programmable Gate Array (FPGA) ; Proportionnel Dérivé Intégral (PID) ; Microblaze ; Modulation de Largeur d'impulsion (PWM); On-Chip Peripheral Bus ; Propriété Intellectuelle (IP); Système Embarqué.

Abstract

In this thesis, an FPGA platform dedicated for embedded control and regulation of dynamic systems is presented. We will show the necessary steps to integrate the three essentials parts of any control system: Input interface, output interface and the controller. This latter is implemented as a PID (Proportional, Derivative, and Integral). As a practical case, we have implemented our system in Memec V2MB1000 platform and developed an experimental device in order to it in a real environment. We have also developed a user software application to facilitate the utilization of our platform and to display the state of the controlled process.

Keywords—Control; Embedded Control; Field Programmable Gate Array (FPGA); PID (Proportional, Derivative, and Integral); Embedded System; Intellectual Property (IP); Microblaze; On-Chip Peripheral Bus; Pulse Width Modulation (PWM); Regulation; Sensor.

ملخص

هذه المذكرة تناقش إنشاء نظام مكون من مسطحة من نوع مصفوفة البوابات المنطقية القابلة للبرمجة في الميدان لتعديل والتحكم في الأنظمة المتغيرة. لقد قمنا بإدخال أجزاء التحكم والإكتساب والمراقبة على نظامنا. إن جزء المراقبة مبني على طريقة العنصر تحكم تناسبي تكاملي تفاضلي. من أجل التحقق من فعالية النظام، قمنا بوضع هذا الأخير في مسطحة من أجل مراقبة سيره في وسط حقيقي للتحكم في درجة حرارة جهاز تجريبي. لقد قمنا أيضا بتطوير برنامج يسمح بتغيير عوامل التجربة ورسم نتائج هذه الأخيرة.

الكلمات الدالة: مراقبة، تعديل، نظام مضمن، مصفوفات البوابات المنطقية القابلة للبرمجة في الميدان، عنصر تحكم تناسبي تكاملي تفاضلي، مكشاف.

Liste des Figures

Liste des Tableaux

Liste des Abréviations

ASIC : Application-Specific Integrated Circuit

BMA : Booth Multiplication Algorithm

CAN : Convertisseur Analogique Numérique

CNA : Convertisseur Analogique Numérique

COTS : Commercial Off-The-Shelf

DSP : Digital-Signal-Processors

EDK : Embedded Design Kit

FPGA : Field Programmable Gate Array

GNU : GNU's Not UNIX

HDL : Hardware Description Language

HW/SW : Hardware / Software

IP : Intellectual Property

JTAG : Joint Test Action Group

LMB : Local Memory Bus

LQG : Linear Quadratic Gaussian

MAC : Multiply And Accumulate

MBMA : Modified Booth Multiplication Algorithm

ODMAC: Optimized Double Multiply And Accumulate

OPB : On-Chip Peripheral Bus

PWM : Pulse Width Modulation

PID : Proportionnel Intégral Dérivé

RMRMA : Recursive Multibit Recording Multiplication Algorithm

RISC : Reduced Instruction Set Computer

UML : Unified Modeling Language

Introduction Générale

De nos jours, l'utilisation des systèmes de contrôle numérique est de plus en plus croissante (Appareils électroménagers, Voitures, Médecine, Industrie, Robotique,...) contraignant ainsi les concepteurs à proposer des solutions efficaces et à moindre coût. Dans le domaine des systèmes embarqués tels que la robotique ou bien l'industrie spatiale et l'aéronautique des contraintes plus difficiles sont imposées aux concepteurs. Effectivement, ces derniers doivent développer des solutions visant à optimiser les ressources disponibles (limitées) ainsi que la consommation de puissance qui représente un facteur clef de qualification des systèmes embarqués.

Il existe principalement quatre solutions pour l'intégration de contrôleurs numériques :

- Composants électroniques de type COTS (Commercial Off-The-Shelf). Cela comprend les microprocesseurs à usage général ainsi que les microcontrôleurs;
- Processeurs de type DSP (Digital-Signal-Processors) comme le TMS320C55x et le TMS320C64x, considérés respectivement comme étant le processeur réclamant le moins d'énergie et le processeur le plus performant du commerce ;
- Circuits reprogrammables de type FPGA (Field-Programmable Gate-Array), tels que ceux de Xilinx ou ceux d'Altera FPGA;
- Circuits de type ASIC (Application-Specific Integrated Circuit) qui sont des puces conçues et fabriquées selon une technologie donnée pour des solutions spécifiques ou à usage général.

Il est démontré que la solution ASIC représente l'approche qui donne les meilleurs résultats par rapport aux autres solutions (en termes de surface, consommation de puissance et vitesse). Cependant, la méthodologie de conception des circuits de type ASIC passe par plusieurs étapes aussi longues les unes que les autres et qui impliquent une augmentation du temps de développement et par conséquent du coût. D'un autre coté, les FPGAs exigent moins de temps de développement et peuvent être exploités à des fins de prototypage rapide et dans certains cas ils peuvent être utilisés comme solution finale.

Ce projet de Master s'insère dans le cadre du protocole de recherche 2011-2013, inscrit sous le contrat de recherche 21/CRSOC/DMN/CDTA/2011 au Centre de Développement des Technologies Avancées, et dont l'intitulé est : Contrôle Embarqué pour des Applications de Micromanipulation. L'objectif est de développer des contrôleurs numériques avancés (LQG, SSV, PID pour comparaison) sur FPGA pour la commande d'une micropince.

Le développement de telles solutions sur FPGA passe par plusieurs phases qui garantissent des propriétés telles la maintenabilité et la réutilisation. La première phase consiste à coder le contrôleur en langage HDL (VERILOG ou VHDL) en utilisant les spécifications établies à partir du cahier de charges. La deuxième phase permet d'effectuer un test fonctionnel du contrôleur dans l'environnement Simulink de Matlab. La dernière phase consiste à intégrer le contrôleur dans un système embarqué sur FPGA et de le valider définitivement dans un environnement réel.

Dans ce mémoire, nous nous intéresserons à la dernière phase du développement du contrôleur. Notre tache consiste à intégrer ce dernier dans une plateforme HW/SW de contrôle embarqué sur FPGA. Tout système de contrôle est composé essentiellement de trois parties : la partie de contrôle, la partie de commande et la partie d'acquisition. Nous exposons à travers les chapitres de ce mémoire les différentes étapes entreprises pour l'intégration et le test des différentes parties de notre système de contrôle. Enfin, un dispositif expérimental permettant la validation définitive de notre système dans un environnement réel doit être développé.

Ce Mémoire est organisé comme suit : dans le premier chapitre, nous commençons par une introduction aux systèmes embarqués et aux circuits FPGA. Le deuxième chapitre introduira les systèmes de contrôle et exposera le contexte et la problématique du projet. Dans le chapitre suivant, nous présentons les différentes étapes d'intégration des parties constituant notre système de contrôle embarqué. Le quatrième chapitre sera consacré à la conception et la réalisation d'un dispositif expérimental permettant le test de notre plateforme dans un environnement réel. Le dernier chapitre traitera les tests qui ont été réalisés et les résultats expérimentaux obtenus. Enfin nous terminons par une conclusion et des perspectives.

Introduction aux FPGA et aux Systèmes Embarqués

Ce chapitre constitue une introduction aux systèmes embarqués sur FPGA et au contrôle et la régulation de processus physiques. Nous commençons tout d'abord par une brève présentation des FPGAs (Définition, domaines d'application, avantages et inconvénients,...) et au flot de conception matériel / Logiciel de Xilinx. Nous présentons aussi les outils de conception permettant le développement de solutions embarquées sur FPGA. Ensuite nous passons aux systèmes embarqués (Définition et domaines d'utilisation) et plus particulièrement à ceux implémentés sur FPGA.

I. Introduction aux FPGAs

I.1. Définition et Domaines d'Applications

Un FPGA (**F**ield-**P**rogrammable **G**ate **A**rray) est un dispositif semi-conducteur contenant des composants à logique programmable (Blocs logiques) et des interconnexions programmables (routage programmable). Les blocs logiques peuvent exécuter n'importe quelle fonctionnalité depuis les fonctions logiques de base (comme : AND, OR, XOR, NOT) jusqu'aux fonctions complexes combinatoires. Les FPGAs comprennent aussi des éléments de mémoire (bascules, blocs complets de mémoire). Quelque soit la fonction logique, elle peut être réalisée par l'interconnexion des blocs logiques. La figure I.1.b représente une vue d'ensemble d'un FPGA [1].

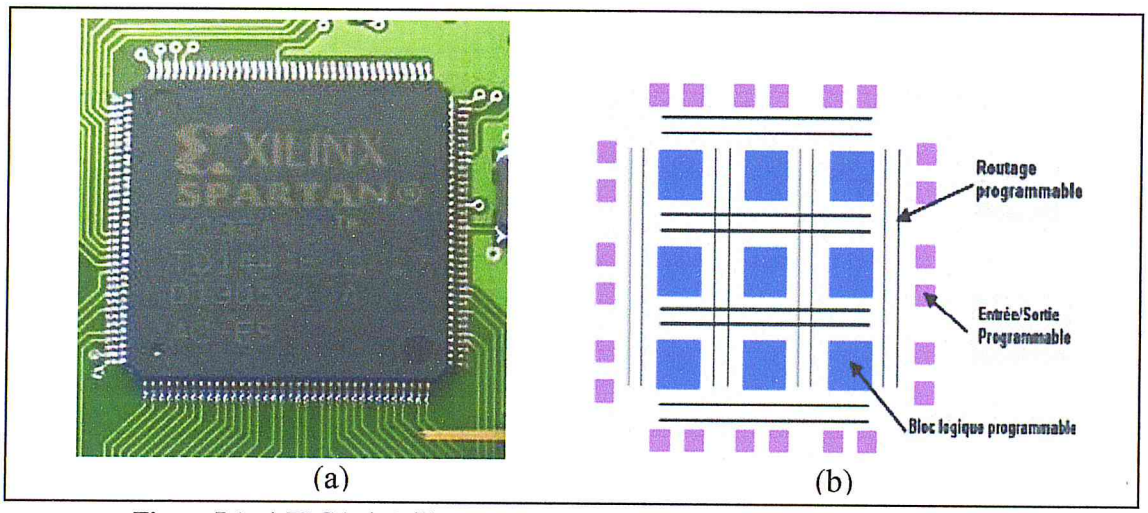


Figure I.1. a) FPGA de Xilinx avec 400 000 portes b) Schéma d'ensemble d'un FPGA

La principale caractéristique des FPGAs est la reconfiguration. Cette dernière peut être même effectuée en cours de fonctionnement. On parlera alors de reconfiguration dynamique. Cette propriété est très intéressante car elle permet de développer des systèmes autonomes. Les FPGAs permettent aussi un prototypage rapide en comparaison avec la solution ASIC (**A**pplication-**S**pecific **I**ntegrated **C**ircuit), mais en contre partie ils sont moins rapides

et consomment plus d'énergie. Le tableau I.1 présente une comparaison entre les caractéristiques d'un FPGA par rapport à un ASIC.

Tableau I.1. Comparaison entre FPGA est ASIC [2].

FPGA	ASIC
• Flexibilité de programmation ; • Délais de développement court ; • Seulement pour applications numériques ; • Consommation de puissance plus élevée.	• Conception fixe ; • Délais de développement long ; • Applications numériques, analogiques, mixtes ; • Faible consommation de puissance.

Donc pour des fins de recherche ou bien de test (Prototypage rapide), les FPGAs constituent un excellent choix. Parmi les fabricants de tels circuits reprogrammables, on trouve Abound Logic, Achronix, Actel, Altera, Atmel, Cypress, Lattice Semiconductor, Nallatech, QuickLogic, SiliconBlue, Tabula Inc., Tier Logic et Xilinx. Dans ce qui suit nous nous intéressons à l'architecture des FPGAs de Xilinx et plus précisément à la famille VIRTEX.

I.2. FPGAs Virtex

En 1998, Xilinx a introduit une nouvelle famille de FPGAs nommée Virtex et depuis plusieurs produits basés sur cette classe ont été mis sur le marché tels que : Virtex II, Virtex-E, Virtex 4 ou bien Virtex 5. La famille Virtex 5 est actuellement la plus utilisée car elle représente une solution à faible coût tout en délivrant de hautes performances.

Un FPGA Virtex est composé essentiellement de quatre éléments qui sont [2]:

- Configurable Logic Bloc : il représente l'élément reconfigurable principal d'un Virtex. Il fournit des composants permettant l'exécution de fonctionnalités de logique combinatoire et séquentielle ;
- Block SelectRAM (RAM) : représente des modules mémoire de stockage ;
- Des Multiplieurs 18-bit x 18-bit ;
- Blocks DCM (Digital Clock Manager) : ces blocks permettent la distribution, la multiplication et la division du signal d'horloge,…

La figure I.2 [3] présente l'architecture interne d'un CLB qui contient quatre Slices :

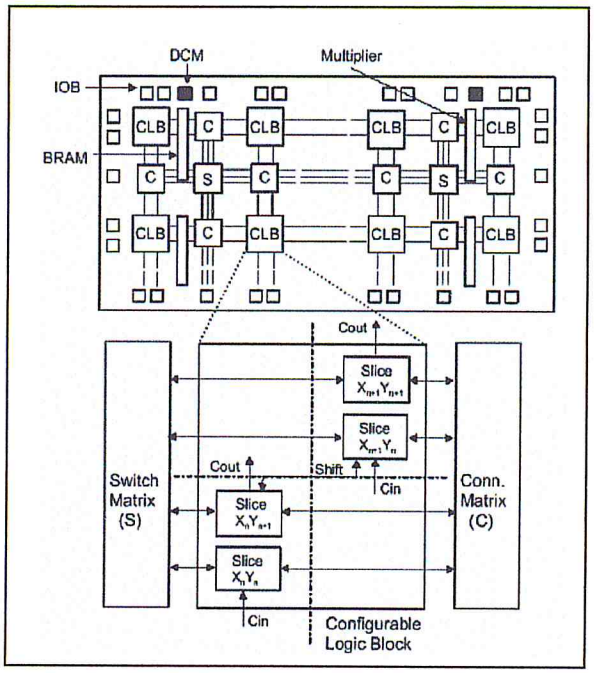


Figure I.2. Architecture d'un FPGA Virtex.

Un Slice (voir figure I.3) est une petite unité de reconfiguration qui contient :

- Deux générateurs de fonction (F & G) qui peuvent être configurés comme Look-Up Table (LUT), ou bien comme registres à décalage,… ;
- Deux Bascules-D ;
- Des portes logiques arithmétiques ;
- Multiplexeurs larges ;
- …

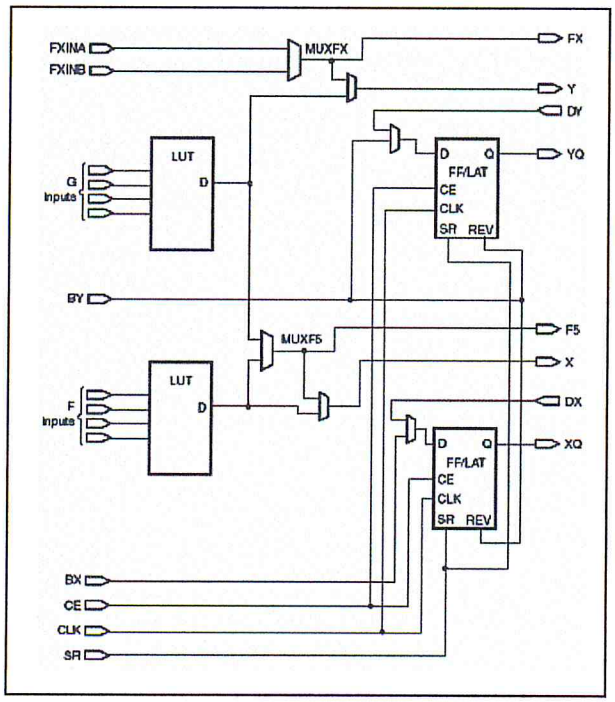


Figure I.3. Architecture interne d'un Slice.

[3] contient plus de détails concernant l'architecture des FPGAs Viretx. Dans la section suivante nous allons présenter les plateformes d'évaluation FPGA et en particulier la plateforme MEMEC V2MB1000 qui a été utilisée dans notre projet.

I.3. Plateformes d'Evaluation FPGA

Une plateforme d'évaluation FPGA est un circuit imprimé contenant un circuit FPGA, un processeur (éventuellement), des périphériques d'E/S permettant l'interaction avec l'environnement externe. Ces plateformes permettent non seulement de tester de façon rapide et efficace des prototypes de test mais peuvent aussi être utilisées comme solution finale après validation. Pour cela, elles sont équipées de mémoire ROM leurs permettant de garder de façon définitive la solution finale.

Pour notre cas, nous avons utilisé la plateforme d'évaluation MEMEC V2MB1000 [4] basée sur le FPGA XC2V1000-4FG456C de Xilinx (famille Virtex II). Cette plateforme est dotée de plusieurs types de périphériques (RS232, Leds, Boutons Switch,…) qui peuvent être utilisés selon les besoins du concepteur et la nature de l'application à développer. La figure I.4 représente une vue globale de la plateforme V2MB1000.

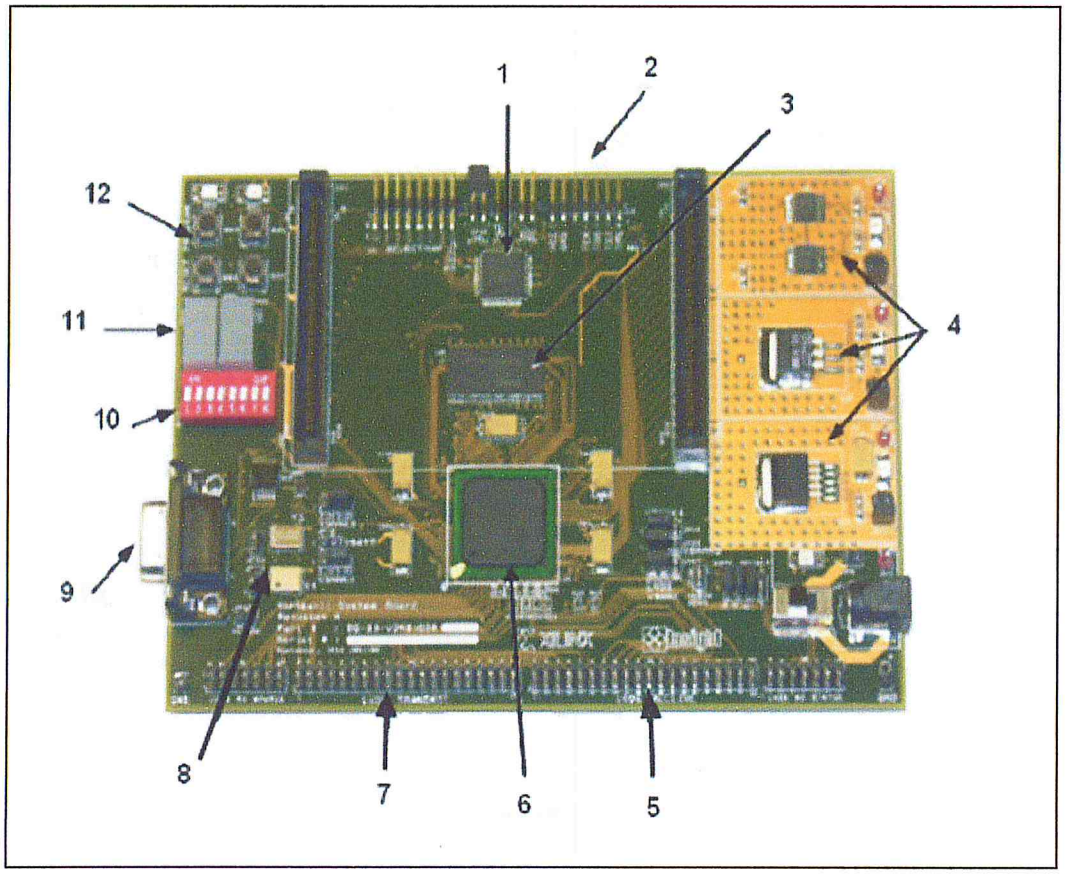


Figure I.4. Plateforme d'évaluation MEMEC V2MB1000.

Tels que :

- **1** : circuit PROM qui permet de conserver définitivement une conception donnée ;
- **2** : est un port JTAG qui permet la programmation du FPGA ;
- **3** : mémoire DDR 16M x 16 ;
- **4** : sources d'alimentation ;
- **5** : est un port de réception LVDS ;
- **6** : est un FPGA Virtex II, XC2V1000-4FG456C ;
- 7 : est un port d'émission LVDS ;
- 8 : deux horloges (25MHZ et 100MHZ) ;
- 9 : est un port de communication RS232 ;
- 10 : DIP Switches ;
- 11 : deux afficheurs sept-segments ;
- 12 : boutons Switches.

Le FPGA est programmé à partir d'un ordinateur en utilisant un câble JTAG et une application appropriée (voir figure I.5). Cette dernière fait partie d'un ensemble d'outils (voir la section suivante) permettant le développement de solutions sur FPGA, depuis les spécifications du cahier de charges jusqu'au test final sur la plateforme d'évaluation. Dans ce Mémoire nous nous intéressons aux outils de développement Xilinx pour FPGA.

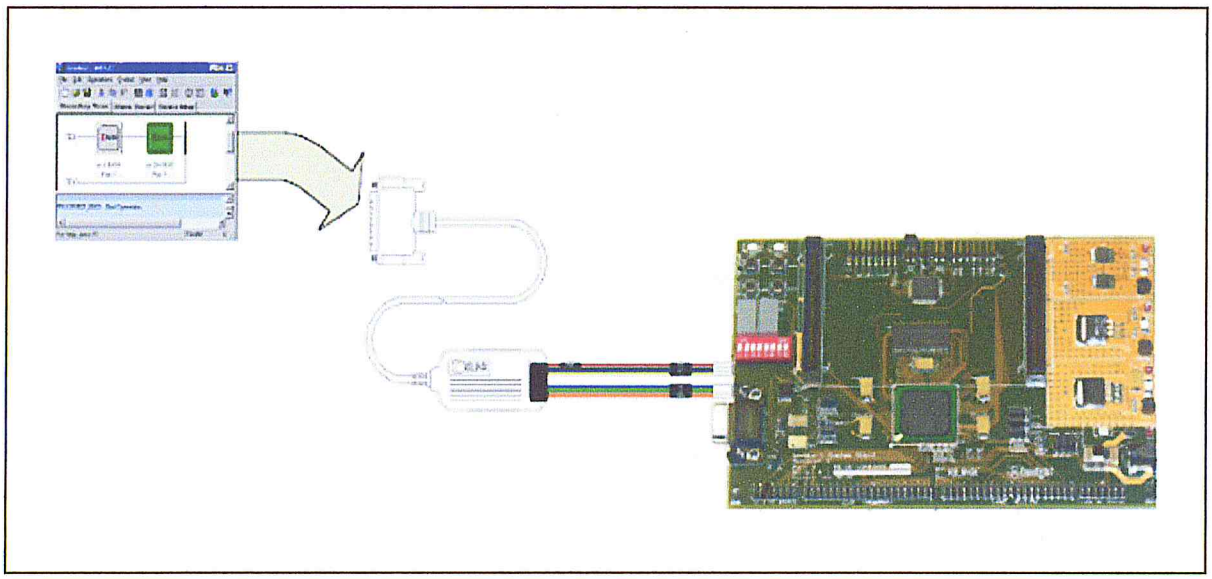


Figure I.5. Programmation d'un FPGA avec un câble JTAG en utilisant l'application IMPACT.

I.4. Outils Xilinx pour FPGA

Le flot de conception Xilinx pour FPGA contient une multitude d'outils adaptés aux besoins du concepteur. Ces outils permettent d'accomplir ce qu'on appelle le CoDesign, c'est-à-dire associer d'un coté le développement Hardware en utilisant des langages HDL (Hardware Description Language) tels que Verilog ou bien VHDL et de l'autre coté le développement de software embarqué en utilisant des langages de programmation classiques tels que C ou bien C++. La figure I.6 est un schéma synoptique représentant les différentes étapes de développement d'une solution HW/SW en utilisant le flot de conception de Xilinx.

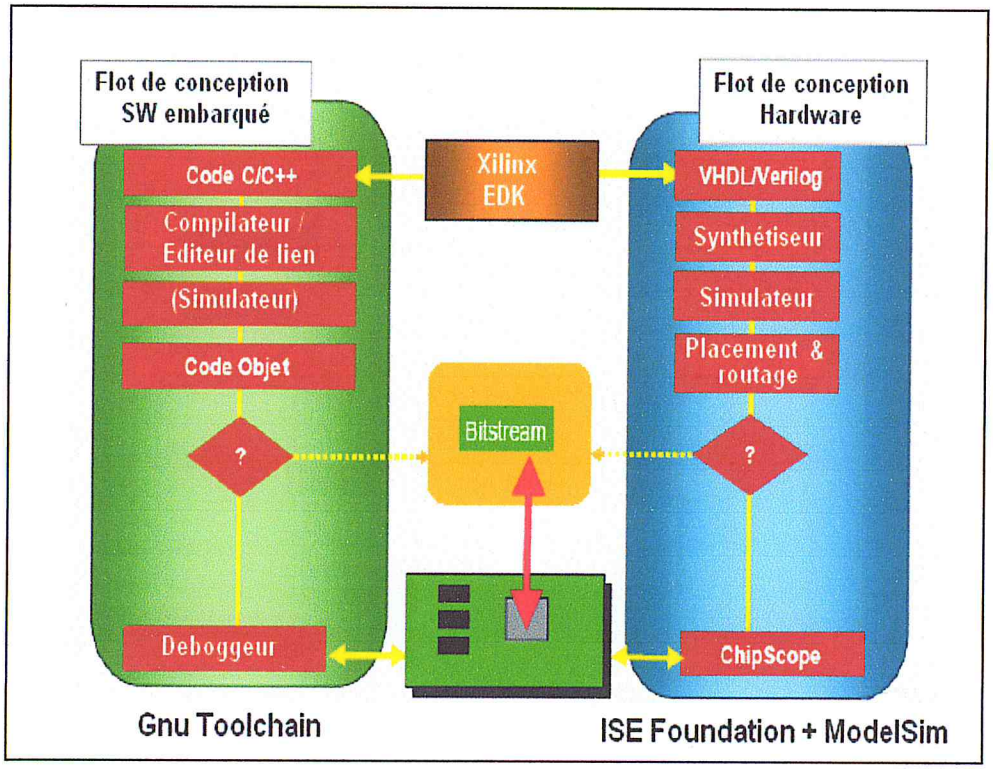


Figure I.6. Flot de conception HW/SW de Xilinx.

L'outil EDK (Embedded Design Kit) [5] de Xilinx permet de développer des solutions sur FPGA en faisant du Co-Design. Le flot de conception Hardware permet d'avoir en finalité des modules qui seront implémentés sur FPGA et qui feront office de circuits électroniques (Contrôleur de périphérique, Acquisition, Convertisseurs, …). Le flot de conception software sert d'un autre coté à développer les applications permettant l'exploitation de la partie Hardware. Après avoir effectué plusieurs tests de vérification du bon fonctionnement de la partie Hardware ainsi que de la partie Software, on obtient en finalité un fichier appelé **Bitstream**. Ce dernier est obtenu après plusieurs étapes (Synthèse, placement, routage, ...) et est utilisé pour programmer en dernier lieu le FPGA. Il faut savoir que la conception et le développement de la partie Hardware suit une méthodologie bien définie sur plusieurs phases

jusqu'à son implémentation effective sur FPGA ou bien sous forme d'ASIC (circuit intégré) en passant par des étapes de vérification très rigoureuses assurant ainsi son bon fonctionnement [6][7]. L'outil de simulation Modelsim [8] est aussi très utilisé durant l'étape de vérification. Il permet non seulement la simulation du fonctionnement d'un module mais aussi la simulation du fonctionnement de toute une plateforme FPGA. Les étapes de synthèse et de placement/routage sont accomplies par EDK de façon transparente en utilisant les outils fournis avec ISE Foundation (tel que XST pour la synthèse). Ce dernier est un Kit de développement pour FPGA fournissant tous les outils nécessaires depuis l'écriture du code HDL jusqu'au test Hardware. Quant à la partie software, EDK utilise une suite d'outils de programmation de GNU (GNU Toolchain) qui contient essentiellement plusieurs compilateurs, Bintuils (Editeur de liens, assembleur,…) ainsi que des outils nécessaires au débogage et à la vérification du code. Dans ce mémoire nous nous intéressons particulièrement à l'outil EDK et à l'intégration de modules et d'applications embarquées sur FPGA.

II. Introduction aux Systèmes Embarqués

II.1. Définition et Domaines d'Applications

Un système embarqué peut être défini comme un système autonome intégrant des logiciels et de l'électronique et éventuellement des parties mécaniques ou autres. Il est dédié à une tâche bien précise. Les ressources dont il dispose sont limitées. Cette limitation est généralement d'ordre spatial (taille limitée) et énergétique (source d'énergie limitée).

Malgré que les caractéristiques des systèmes embarqués diffèrent selon l'application visée, nous pouvons néanmoins en citer les propriétés les plus communes :

- Ils sont généralement conçus pour exécuter une tache bien précise ;
- Ils doivent satisfaire des contraintes très difficiles (consommation d'énergie, surface, rapidité,…) ;
- Ils doivent interagir avec leur environnement et calculer les résultats d'un traitement en temps réel.

Aujourd'hui ces systèmes sont très présents dans notre vie quotidienne et leur production dépassent de loin celle des ordinateurs personnels. Les domaines dans lesquels on trouve des systèmes embarqués sont de plus en plus nombreux :

- Transport : Automobile, Aéronautique (avionique), etc ;
- Astronautique : Fusée, Satellite artificiel, Sonde spatiale, etc ;
- Militaire : Missiles, Avions de chasse, etc ;
- Télécommunication : Set-top box, Téléphonie, Routeur, Pare-feu, Serveur de temps, Téléphone portable, etc ;
- Electroménager : Télévision, Four à micro-ondes ;
- Impression : Imprimante multifonctions, Photocopieur, etc ;
- Informatique : Disque dur, Lecteur de disquette, etc ;
- Multimédia : Console de jeux vidéo, Assistant personnel ;
- Guichet automatique bancaire (GAB) ;
- Equipement médical ;
- Automate programmable industriel, Contrôle-commande ;
- Métrologie.

Figure I.7. Exemples d'utilisation des systèmes embarqués.

Un système embarqué est constitué d'un processeur exécutant des applications softwares (applications dédiées, système d'exploitation, …). Dans la section suivante nous nous intéressons aux différents types de processeurs et plus particulièrement à ceux utilisés dans les systèmes embarqués sur FPGA.

II.2. Processeur

Il existe deux types de processeurs utilisés dans les systèmes embarqués :

- Hardcore : ce type désigne des circuits intégrés (ASIC) qui sont fabriqués et dédiés à une application bien précise. Ce genre de processeurs ne peut être reconfiguré après sa fabrication. Il existe plusieurs genres de processeurs Hardcores qui sont adaptés à des applications spécifiques tels que : les DSPs (Digital Signal Processor) pour les applications de traitement du signal, les GPUs (Graphic Processing Units) pour les applications de visions et de traitement d'images, etc ;

- Softcore : ce type désigne les processeurs qui sont décrits entièrement en langage HDL (Hardware Description Language) tels que VHDL ou bien Verilog et qui sont généralement implémentés dans une logique reconfigurable (FPGA). Ce genre de processeurs présente l'énorme avantage d'être reconfigurable c'est-à-dire qu'il peut être adapté selon les besoins de l'application visée (Performance, consommation de puissance,...). L'autre avantage réside dans le fait que n'importe quel nombre de processeurs Softcores peut être intégré sur FPGA (Dans la limite des capacités de ce dernier bien sûr) et que son coût est nettement inférieur à celui d'un processeur Hardcore. Cependant, ce dernier est bien plus performant.

Notons que plusieurs familles de FPGA intègrent un ou plusieurs processeurs Hardcores, comme c'est le cas pour le Virtex II-Pro qui contient deux processeurs PowerPC.

Dans ce mémoire nous nous sommes limités à l'utilisation du processeur softcore *Microblaze* [9] de Xilinx pour les raisons suivantes :

- La plateforme matérielle utilisée (MEMEC V2MB1000) contient un FPGA (Virtex II, XC2V1000-4FG456C) qui ne comprend pas de processeur Hardcore;

- Facilité d'utilisation et de configuration du processeur Microblaze à l'aide de l'outil EDK de Xilinx;

- Microblaze peut être intégré dans un système complexe qui contient plusieurs périphériques d'E/S et est supporté par plusieurs systèmes d'exploitation embarqués tels que : uClinux, Xilinx Xilkernel, FreeRTOS, etc.

Dans ce qui suit, nous présentons une vue d'ensemble des processeurs Softcores existants ainsi que sur l'architecture du processeur Microblaze et des systèmes embarqués basés sur ce dernier.

II.2.1. Vue d'Ensemble sur les Processeurs Softcores Existants

Il existe plusieurs processeurs Softcores qui peuvent être utilisés dans les systèmes embarqués sur FPGA. Pour les FPGAs de Xilinx, on distingue essentiellement trois processeurs largement utilisés :

- **Microblaze :** est un processeur RISC 32 bits développé par Xilinx. Il a été conçu et optimisé spécialement pour les FPGAs de Xilinx. Microblaze est distribué avec EDK, cependant son code source n'est pas accessible ;
- **LEON :** est un processeur RISC 32 bits compatible avec l'architecture de SPARC V8. Il est hautement configurable et son code source est accessible sous la licence GNU GPL ;
- **OpenRisc :** est un processeur RISC 32 bits. OpenRisc est gratuit (Open Source) et peut être obtenu sur le site de OpenCores (Code Verilog) sous la licence GNU GPL.

Le tableau I.2 [10] présente une comparaison succincte entre ces trois processeurs.

Tableau I.2. Comparaison entre Microblaze, LEON et OpenRisc.

Processeur	Avantages	Inconvénients
Microblaze	• Optimisé pour les FPGAs de Xilinx ; • Utilisation se ressources minimisée ; • Bien documenté • Facilité d'ajout de périphériques utilisateur.	• Ne contient pas de MMU (Memory Managment Unit) ; • Code source non disponible.
LEON	• Hautes performances ; • Code source disponible.	• Utilise beaucoup de ressources sur FPGA.
OpenRisc	• Bonne Performances ; • Code source disponible.	• Utilise beaucoup de ressources sur FPGA ; • Pas bien documenté ; • Outils de développement pauvres.

II.2.2. Architecture de Microblaze

Microblaze est un processeur RISC 32 bits doté d'un pipeline à trois étages. Il contient 32 registres à usage général, une UAL et un riche ensemble d'instructions pour les applications embarquées. Il est à noter que Microblaze est une propriété de Xilinx et que son code source n'est pas disponible. La figure I.8 représente un schéma synoptique de l'architecture interne d'un processeur Microblaze.

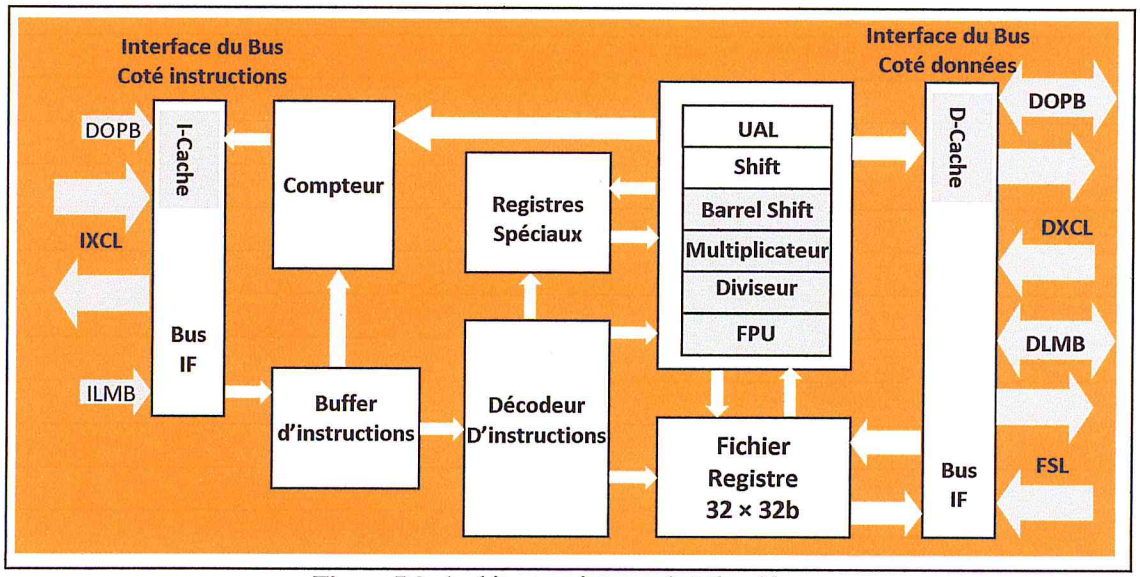


Figure I.8. Architecture interne de Microblaze.

Les principales caractéristiques de Microblaze sont :

- 32 registres à usage général ;
- Pipeline à trois étages ;
- Supporte le bus OPB (On-Chip Peripheral Bus). Ce dernier est une interface à usage général qui est conforme à la technologie CoreConnect de IBM. Cette technologie permet aussi de développer des périphériques pour des systèmes se basant sur le processeur PowerPC ;
- Connexion directe à On-Chip Memory (BRAM) à travers le bus LMB (Local Memory Bus) ;
- Possibilité d'ajout d'un Co-Processeur utilisant une connexion FSL (Fast Simplex Link) ;
- Peut travailler avec une fréquence pouvant atteindre 200 Mhz sur Virtex 4 et 170 Mhz sur Virtex 2 ;
- Multiplicateur et diviseur Hardware réduisant le temps d'exécution des opérations.

II.2.3. Système Embarqué sur FPGA

Nous avons vu qu'un système embarqué est un système autonome intégrant des logiciels et du matériel ce qui signifie la présence d'un processeur et des périphériques connectés à ce dernier à travers un Bus. Nous avons vu aussi que les FPGAs permettent l'intégration des processeurs Softcores qui sont moins chers, configurables mais aussi moins performants que les processeurs dédiés. Dans notre cas, nous utilisons la plateforme MEMEC V2MB1000 qui ne contient pas de processeur Hardcore à l'instar de la Virtex 2 Pro. Nous sommes donc contraints à utiliser le processeur Microblaze.

Notre système embarqué est constitué du processeur Microblaze connecté à travers le bus OPB aux différents périphériques d'E/S. Nous utilisons comme mémoire centrale, la BRAM présente sur le FPGA. Cette dernière est reliée à Microblaze à travers le Bus LMB (Local Memory Bus). La figure I.9 représente un schéma synoptique d'un système embarqué sur FPGA ayant Microblaze comme processeur.

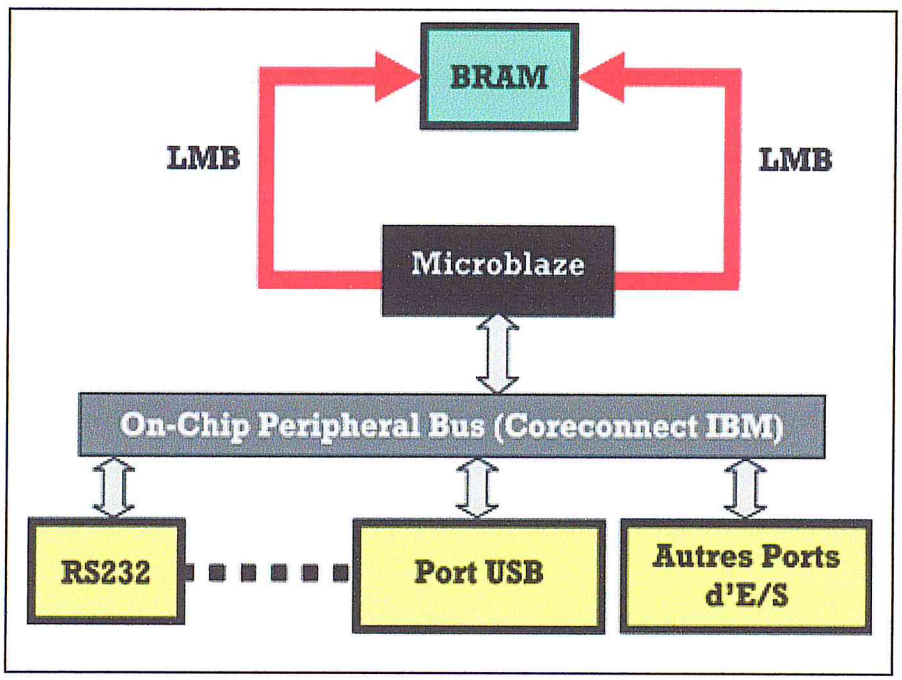


Figure I.9. Schéma synoptique d'un système embarqué sur FPGA basé sur Microblaze.

Ce genre de système peut être développé facilement en utilisant EDK. Ce dernier dispose d'une base de données contenant les sources HDL (en Verilog ou en VHDL) de plusieurs modules qui peuvent être soit des contrôleurs de périphériques d'E/S (RS232, USB,...) soit un contrôleur d'interruption ou tout simplement un module dédié à une tâche bien précise (traitement d'image, traitement de la voie, ...). Ces modules doivent être impérativement compatibles-OPB afin qu'ils puissent être gérés à travers des applications embarquées.

III. Conclusion

Dans ce chapitre nous avons présenté une introduction aux circuits FPGA. Ces derniers permettent un prototypage rapide et une conception à la fois de la partie matérielle et logicielle du système à concevoir. Nous avons exposé en particulier les FPGAs de la famille Virtex de Xilinx ainsi que le flot de conception qui contient un ensemble d'outils adaptés au développement matériel et logiciel. Nous avons aussi présenté les systèmes embarqués et leurs domaines d'application. Nous nous sommes intéressés plus particulièrement aux systèmes embarqués sur FPGA. Dans le chapitre suivant, nous allons présenter une introduction aux systèmes de contrôle et exposer la problématique et le contexte de notre projet tout en situant notre tache exacte.

Introduction aux Systèmes de Contrôle

Dans ce chapitre nous présentons une introduction aux systèmes de contrôle. Nous exposons de façon brève le principe de fonctionnement d'un système de contrôle et de régulation et ses principales parties ainsi que le fonctionnement du régulateur PID (Proportionnel, Intégral, Dérivé). Nous finirons par présenter la problématique de notre projet et dans quel contexte s'insère ce dernier.

I. Introduction aux Systèmes de Contrôle

Dans de nombreux systèmes et produits actuels, il est indispensable de maintenir des grandeurs physiques (Température, vitesse, débit,…) à des valeurs déterminées dans les installations industrielles ou bien simplement dans des appareils de tous les jours (Four, climatiseurs,…) et cela en dépit des variations internes ou externes influant sur ces dernières. Dans le cas où ces perturbations sont minimes, un simple réglage dit en boucle ouverte est suffisant pour maintenir le processus à contrôler à la valeur désirée. Cependant, dans la plupart des cas, le réglage en boucle ouverte ne permet pas l'obtention des résultats attendus car ce dernier est trop brut ou bien instable. Une comparaison en permanence entre la valeur mesurée du paramètre à régler et la valeur souhaitée s'impose pour avoir un meilleur contrôle du processus. La figure II.1 représente le fonctionnement d'un système de contrôle et de régulation en boucle fermée et en boucle ouverte.

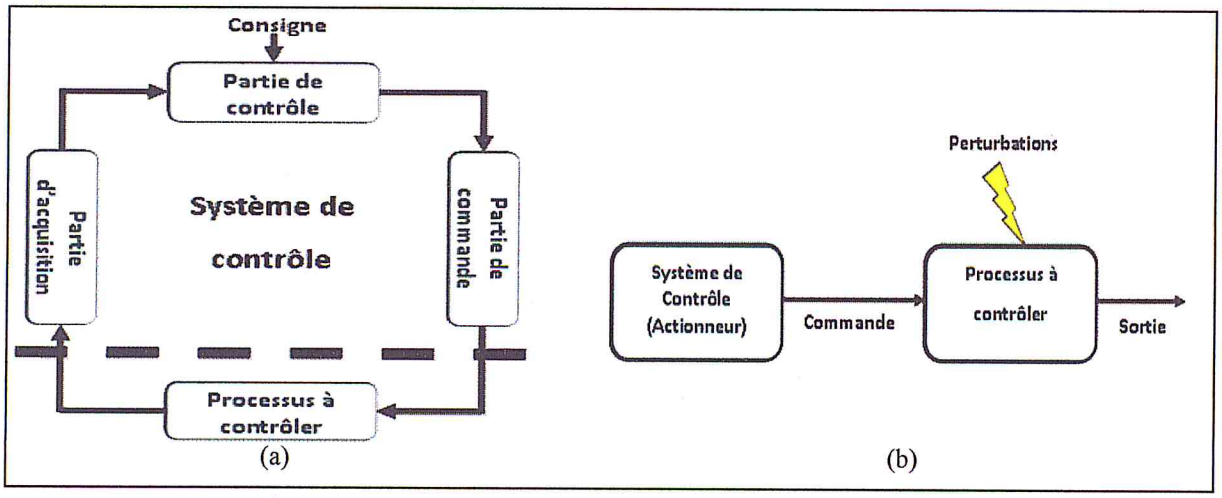


Figure II.1. Types des systèmes de contrôle (a) Contrôle en boucle fermée (b) contrôle en boucle ouverte.

En général un système de Contrôle ou de régulation en boucle fermée est composé essentiellement de trois parties :

- **Partie d'acquisition:** cette partie permet de récupérer la valeur de l'état du processus à contrôler, comme la température, la vitesse, la pression, etc ;

- **Partie de contrôle:** cette partie permet de calculer la valeur de la commande à appliquer sur le processus afin de satisfaire la consigne imposée par l'utilisateur;**Partie de commande:** cette partie permet d'appliquer la commande calculée par la partie de contrôle sur le processus à contrôler.

Le contrôle d'un processus donné peut être effectué soit en régulation soit en asservissement. Dans le cas de contrôle d'un processus en régulation, la valeur de la consigne est maintenue constante et le système de régulation doit maintenir l'état du processus malgré les perturbations externes. Dans le milieu industriel, l'aspect de régulation est primordial car les valeurs des consignes sont en général constantes. La figure II.2 montre le comportement d'un processus en régulation.

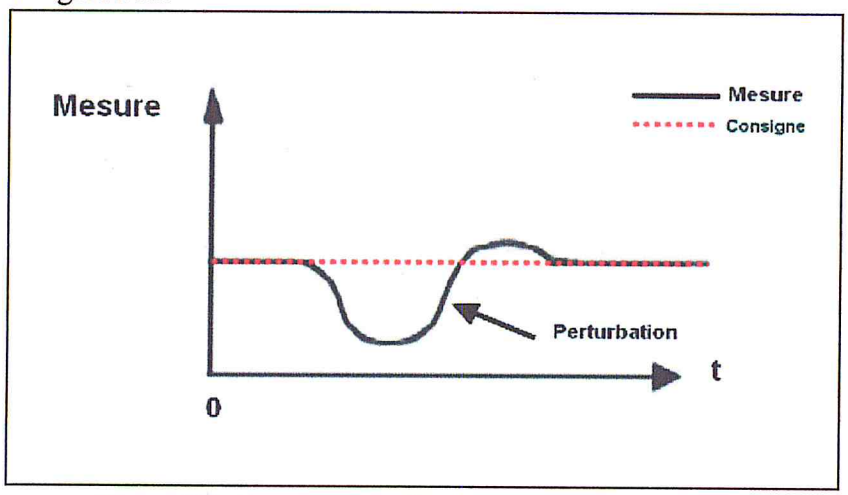


Figure II.2. Comportement d'un processus en régulation.

Dans le cas de contrôle d'un processus en asservissement, la valeur de la consigne est modifiée par l'opérateur ce qui induit un changement du point de fonctionnement du processus à contrôler. La figure II.3 représente le comportement d'un processus en asservissement.

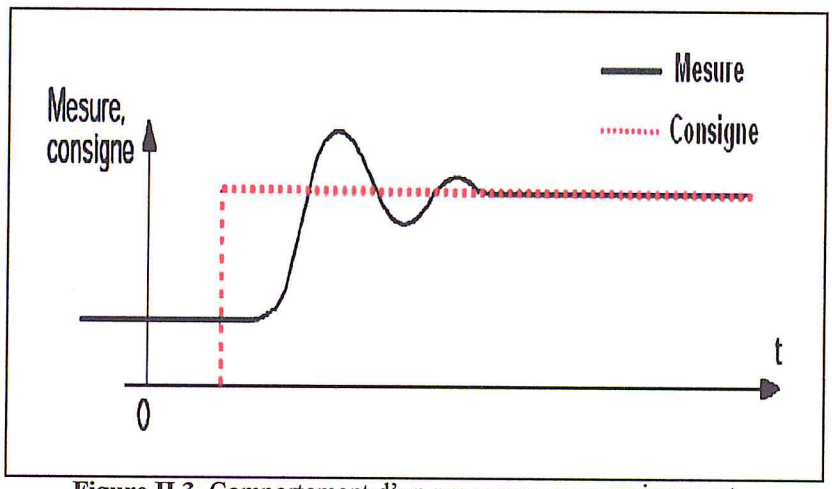


Figure II.3. Comportement d'un processus en asservissement.

I.2. Régulateur PID (Proportionnel Intégral Dérivé)

En industrie, le régulateur PID (Proportionnel Intégral Dérivé) représente le standard le plus utilisé (voir Annexe A). Il permet de contrôler ou de réguler en boucle fermée à l'aide de ses trois actions (P, I et D) les performances d'un grand nombre de processus industriels (température, débit, vitesse, déplacement, …). Un régulateur PID remplit essentiellement les trois fonctions suivantes :

- Il fournit un signal de commande en tenant compte de l'état du processus par rapport à la consigne de l'utilisateur ;
- Il élimine l'erreur statique en utilisant l'action Intégral ;
- Il anticipe les variations de l'état du processus en utilisant l'action Dérivé.

La figure II.4 présente le principe de fonctionnement d'un régulateur PID.

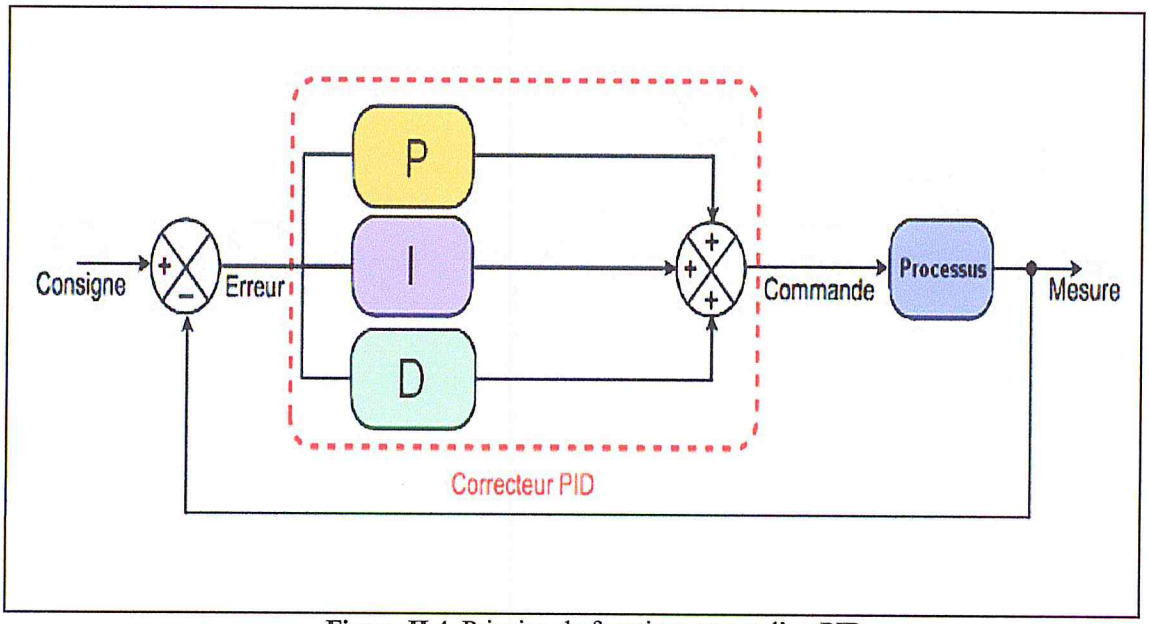


Figure II.4. Principe de fonctionnement d'un PID.

Toutes les équations régissant le fonctionnement des trois actions P, I et D sont expliquées et détaillées dans [11] (voir annexe A). Le réglage d'un PID consiste à déterminer les coefficients des actions P, I et D afin d'obtenir une réponse adéquate du processus et de la régulation. L'objectif est d'être robuste, rapide et précis. Pour cela, il faut limiter le ou les éventuels dépassements (overshoot). Le critère de robustesse représente le critère le plus important et le plus critique. En effet, ce dernier doit garantir un fonctionnement correct même si l'état du processus subit des variations externes. Le critère de rapidité dépend du temps de monté et de celui d'établissement du régime stationnaire (voir figure II.5). Quant au critère de précision, il dépend de l'erreur statique.

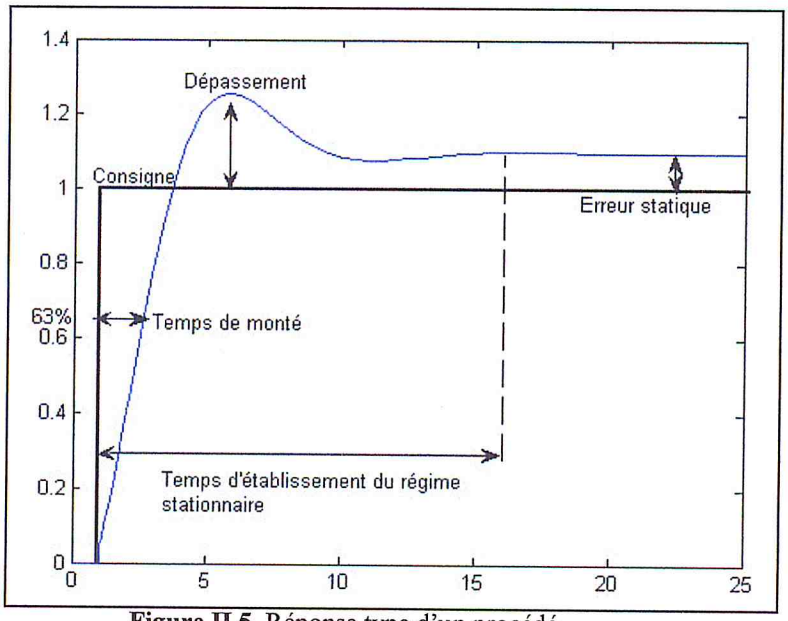


Figure II.5. Réponse type d'un procédé.

II. Contexte et Problématique

Ce travail de Master s'intègre dans le cadre du protocole de recherche 2011/2013 au Centre de développement des Technologies Avancées inscrit dans le projet dont l'intitulé est Contrôle embarqué pour des applications de micromanipulation. Notre but consiste à intégrer les contrôleurs développés dans un système embarqué en les dotant d'une interface leur permettant de communiquer avec le processeur du système ainsi que d'un ensemble de pilotes qui permettent l'accès à ces contrôleurs à travers une application embarquée.

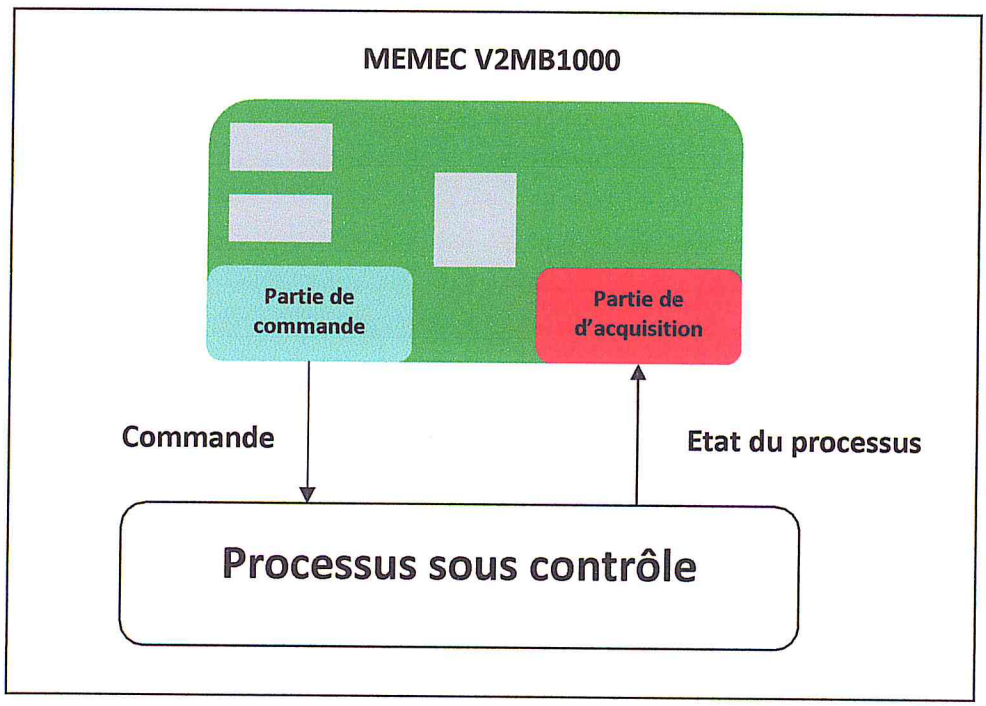


Figure II.6. Schéma synoptique de notre système de contrôle embarqué.

Nous visons ainsi le développement d'une solution HW/SW embarquée sur FPGA pour les systèmes de contrôle. Il s'agit ici en premier lieu, d'intégrer les parties d'acquisition et de commande sur la plateforme V2MB1000 en réutilisant et en adaptant des modules à notre cas. Cette tache est essentielle car elle permet au système de contrôle d'avoir les données nécessaires sur le processus à contrôler et de modifier l'état de ce dernier en appliquant une commande donnée. L'étape suivante consiste à intégrer la partie de contrôle au sein de notre système de contrôle embarqué. Cette partie est implémentée sous forme d'un module Hardware (Propriété Intellectuelle: IP) réalisant le travail d'un PID. La dernière étape consiste à valider définitivement notre système dans un environnement réel. Il s'agit de contrôler et de réguler la température d'un dispositif expérimental composé d'un tube contenant un ventilateur, une lampe halogène et un capteur. Le développement d'une application sur PC permettant à l'utilisateur d'interagir avec le système et de visualiser l'état du dispositif expérimental est aussi nécessaire.

Intégration du Système de Contrôle Embarqué

Dans ce chapitre, nous présentons les différentes étapes permettant l'intégration de notre système de contrôle embarqué sur FPGA. Dans le chapitre précédent, nous avons vu que tout système de contrôle est constitué de trois parties: la partie d'acquisition, la partie de commande et la partie de contrôle. En premier lieu nous allons créer à l'aide de l'outil EDK un système de base avec un fonctionnement minimal, contenant un processeur Microblaze et quelques périphériques. Ensuite nous allons procéder à l'intégration, au développement des pilotes appropriés et au test des modules jouant le rôle de la partie de commande, d'acquisition et de contrôle.

I. Configuration du Système Matériel de Base

Cette étape consiste à créer à l'aide de l'outil EDK un système embarqué de base [12] composé du processeur Microblaze relié à plusieurs modules ou périphériques à travers le bus OPB (On Chip Peripheral Bus) [13]. Notre système est constitué des éléments suivants:

- Microblaze;
- Une BRAM interne reliée à Microblaze à travers le bus LMB (Local Memory Bus);
- Un contrôleur d'interruptions;
- Un contrôleur du port série RS232;
- MDM (Microblaze Debug Module): c'est un module de débogage pour Microblaze;

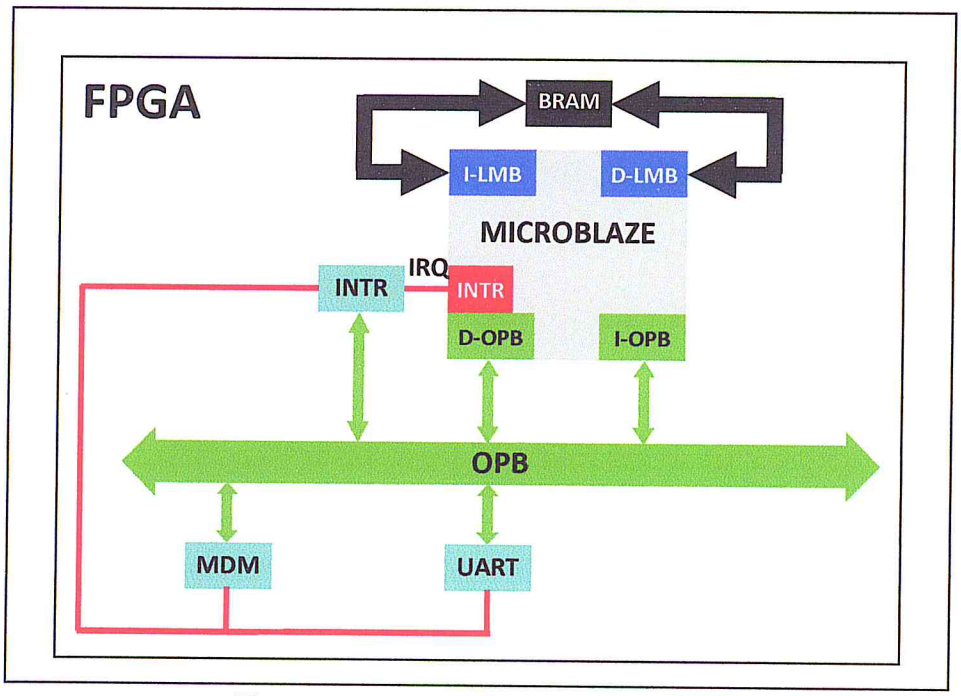


Figure III.1. Architecture du système de base.

Par analogie avec l'ordinateur personnel qui a comme entrée standard le clavier et comme sortie standard l'écran, le port RS232 constitue pour notre système embarqué l'entrée et la sortie standard. L'utilisation de ce port se fait au moyen du logiciel HyperTerminal ou tout autre logiciel équivalent (dans notre cas **Tera Term Pro**). Effectivement, ce dernier permet non seulement d'afficher les données provenant du port RS232 mais aussi d'en envoyer.

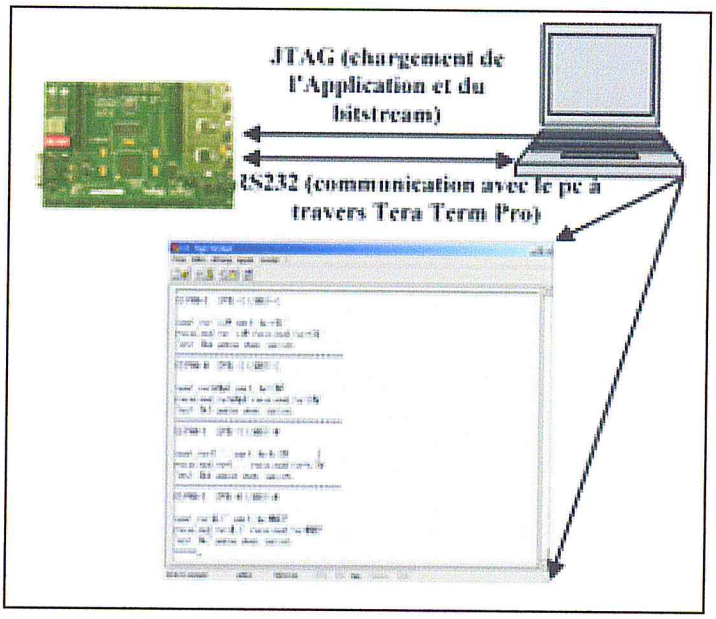


Figure III.2. Procédure de communication entre la plateforme et un PC.

L'étape suivante consiste à générer le Bitstream correspondant à notre configuration et à exécuter l'application de test. Cette dernière permet de vérifier le bon fonctionnement des différents périphériques faisant partie de notre système de base. Le résultat d'exécution de l'application sera affiché à travers le logiciel Tera Term Pro comme le montre la figure III.3.

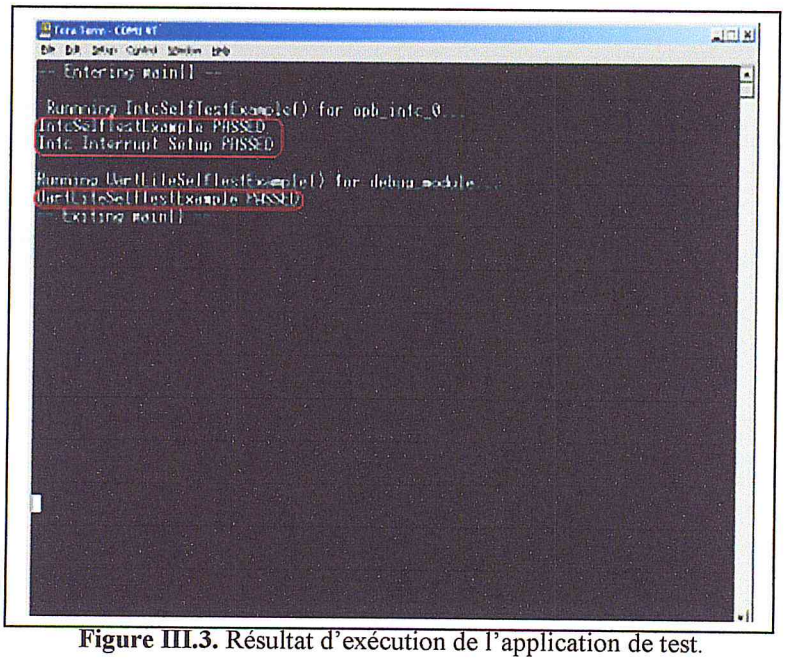


Figure III.3. Résultat d'exécution de l'application de test.

II. Intégration de la Partie Commande

Dans cette étape, nous allons procéder à l'intégration de la partie de commande sur la plateforme MEMEC V2MB1000. Cette partie permet au système d'appliquer une commande sur le processus à contrôler afin d'amener l'état de ce dernier à une situation proche de la consigne imposée par l'utilisateur. Cette fonctionnalité peut être accomplie soit par un Convertisseur Numérique/Analogique (CNA) soit par un PWM (Modulation de largeur d'impulsion) [14]. Dans notre cas, la plateforme MEMEC V2MB1000 ne contient pas initialement un CNA, néanmoins ce dernier peut être remplacé par un module HDL au niveau du FPGA. A cet effet, nous utiliserons le module OPB Delta Sigma Digital To Analog existant au niveau de EDK. Ce dernier permet un contrôle en continu d'un processus donné. La deuxième solution consiste à utiliser un PWM qui permet de contrôler un processus donné en mode discret. Cette fonctionnalité peut être réalisée par la réutilisation du module OPB Timer-Counter qui peut être utilisé en mode PWM.

II.1. OPB Delta Sigma Numérique/Analogique

Un convertisseur Numérique/Analogique permet de convertir une valeur numérique en un niveau de tension qui lui correspond. Cette fonctionnalité peut être effectuée par le module OPB-CNA delta sigma Softcore nécessite l'utilisation d'un filtre passe bas externe [15]. Ce dernier est un circuit simple composé d'une résistance reliée en série à une capacité (voir figure III.4). Les principales caractéristiques de ce module de conversion sont :

- Taille configurable de la donnée à convertir (2 jusqu'à 16 bits) ;
- Utilisation en mode interruption ;
- 16 entrées pour le FIFO ;
- Interface OPB (OnChip peripheral Bus) 32 bits.

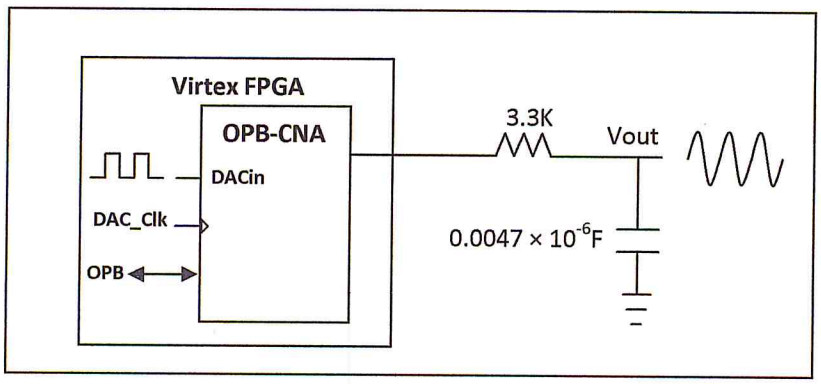


Figure III.4. Module de conversion Numérique/Analogique.

Dans ce qui suit, nous donnons brièvement une description des signaux entrées/sorties, des interruptions, des paramètres ainsi que des registres faisant partie du module OPB-CNA qui permettent sa configuration.

Les signaux d'E/S du module OPB-CNA sont divisés en deux catégories :

- Les signaux du OPB-CNA : ces signaux sont en relation directe avec le fonctionnement interne du OPB-CNA. Il y a trois signaux d'E/S qui sont :
  - **Dac_clk_en :** ce signal d'entrée permet de synchroniser la conversion du CNA avec le signal d'horloge du système ;
  - **Read_en :** ce signal d'entrée permet de lire une nouvelle valeur à partir du FIFO afin de la convertir en un niveau de tension;
  - **Dac_out :** ce signal de sortie représente la valeur du nombre binaire convertie lue à partir du FIFO.
- Les signaux OPB : ces signaux permettent à une application embarquée d'accéder soit en lecture soit en écriture à un registre du module OPB-CNA. On peut citer essentiellement les signaux suivants:
  - **OPB_Clk :** ce signal d'entrée est connecté directement à l'horloge du système ;
  - **OPB_Rst :** ce signal d'entrée est connecté au signal système qui permet la remise à zéro du système ;
  - **IP2INTC_Irpt :** ce signal de sortie représente l'interruption système du module OPB-CNA ;
  - **OPB_RNW :** ce signal d'entrée est mis à '1' par le processeur dans le cas d'une opération de lecture d'un registre ou bien à '0' dans le cas d'une écriture.

Le module OPB-CNA peut être ajusté par l'utilisateur en modifiant plusieurs paramètres notamment :

- **C_NUM_DAC_BITS :** ce paramètre représente la taille en bit du OPB-CNA qui varie entre 2 et 16 ;
- **C_BASE_ADRESS:** ce paramètre représente l'adresse du premier registre du OPB-CNA.

Le module OPB-CNA contient plusieurs registres adressables, accessibles en Lecture/Ecriture. L'adresse de base du premier registre est spécifiée dans le paramètre **C_BASE_ADRESS**. Ces registres permettent d'activer et d'initialiser le module, d'avoir des informations sur l'état du FIFO ou bien de définir le comportement du module en utilisant les interruptions. Le tableau III.1 regroupe les registres les plus importants du module OPB-CNA.

Tableau III.1. Quelques registres du CNA.

Registre	Accès	Fonction	
		Bits	Description
Registre de Contrôle (CR)	Lecture/Ecriture	0-29	Réservé
		30	Remise à zéro du FIFO '1' : remet à zéro le FIFO. '0' : Opération normale.
		31	Activation du CNA '1' : active le CNA. '0' : désactive et remet le CNA à zéro.
Registre d'occupation du FIFO (OCCY)	Lecture	0-26	Réservé
		27-31	Nombre d'éléments dans le FIFO.
Registre d'interruption programmable (PIRQ)	Lecture /Ecriture	0-26	Réservé
		27-31	Une interruption est générée si la valeur de ce registre est supérieure à celle du registre OCCY

Le fonctionnement du OPB-CNA peut être régit en mode interruption ou en mode scrutation (Polling mode). Le module OPB-CNA possède deux interruptions qui permettent la gestion du nombre de données présentes au niveau du FIFO :

- **FIFO EMPTY :** cette interruption est mise à '1' lorsque le FIFO est vide ;
- **FIFO PIRQ :** cette interruption est mise à '1' lorsque le nombre de données au niveau du FIFO est supérieur à la valeur présente dans le registre PIRQ.

Les pilotes (interface entre l'application et le matériel) qui permettent l'utilisation du module OPB-CNA sont fournis avec EDK [15]. Ces derniers permettent:

- L'initialisation et la remise à zéro ;
- Lancer ou bien arrêter la conversion ;
- Activer ou bien désactiver les interruptions ;
- Consulter l'état du FIFO (Nombre de données);
- Ajouter des données au FIFO.

Pour tester le fonctionnement du module OPB-CNA, nous utiliserons l'oscilloscope numérique Tektronix TDS3054 [16]. Ce dernier sera relié à la sortie du filtre passe bas à l'aide d'une sonde (voir figure III.5).

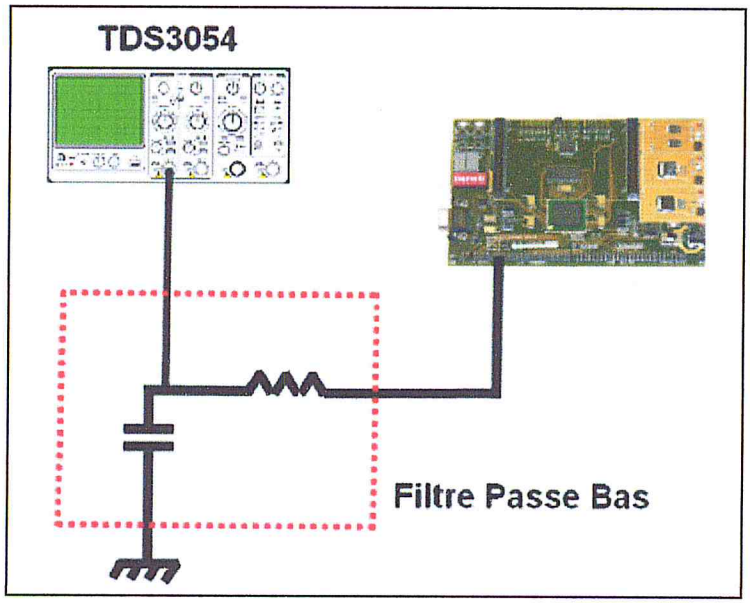


Figure III.5. Montage de test du OPB-CNA

Le test consiste à générer à l'aide d'un programme embarqué sur la plateforme MEMEC V2MB1000 plusieurs types de signaux analogiques. En respectant le fonctionnement du OPB-CNA décrit dans [15], la conversion d'une valeur située dans le FIFO en un niveau de tension se fait à chaque fois que le signal **READ_EN** est mis à '1'. Cette opération (mise à '1' du signal READ_EN) doit se faire chaque 256 tops d'horloge laissant ainsi le temps au module OPB-CNA de convertir la valeur courante avant d'entamer la valeur suivante. Le programme embarqué extrait d'un tableau la valeur à convertir chaque 256 tops d'horloge et l'écrit dans le FIFO. La figure III.6 présente les signaux qui ont été générés avec le programme embarqué et les valeurs du tableau correspondantes.

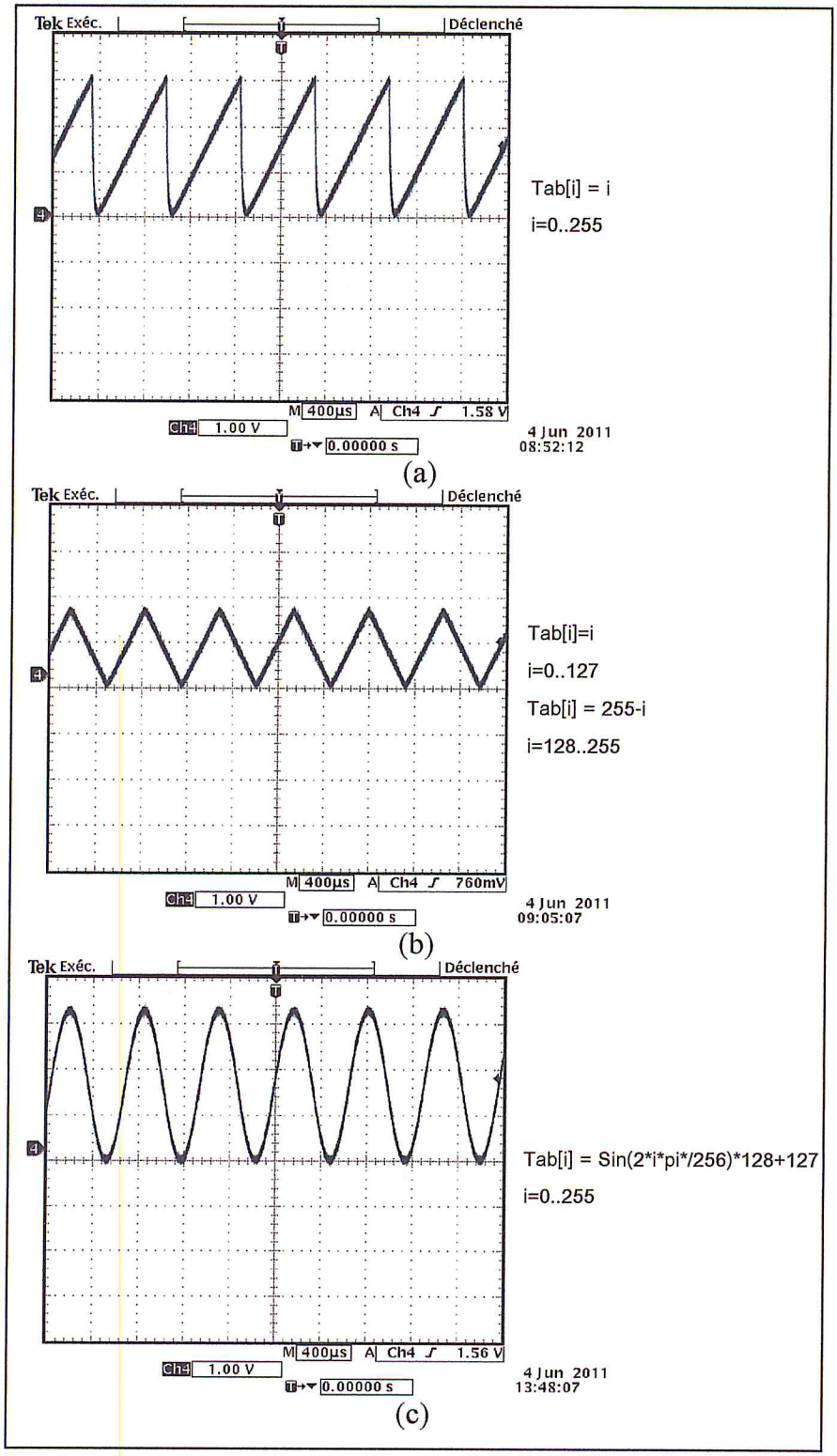


Figure III.6. Test de génération de signaux analogiques (a) Dents de scie (b) Triangulaires (c) Sinusoïdaux.

Il est clair que l'utilisation d'un OPB-CNA peut s'avérer très utile, cependant ce dernier n'est pas adapté au contrôle de certains dispositifs tels que les transistors MOSFET qui se surchauffent rapidement avec un contrôle en continu. Dans la section suivante nous présentons une autre méthode qui permet de commander des dispositifs en mode discret tout

en permettant un ajustement précis du fonctionnement voulu.

II.2. Modulation de Largeur d'Impulsion (PWM)

La technique PWM est couramment utilisée pour synthétiser des signaux continus à l'aide de circuits au fonctionnement tout ou rien, ou plus généralement à états discrets. La fonction PWM possède deux propriétés importantes à gérer qui définissent son fonctionnement : la période et le rapport cyclique. Ce dernier est défini par la relation suivante :

$$R = (t1 / T) \times 100$$

Tel que T représente la période et t1 est la durée pendant laquelle le signal est mis à '1'.

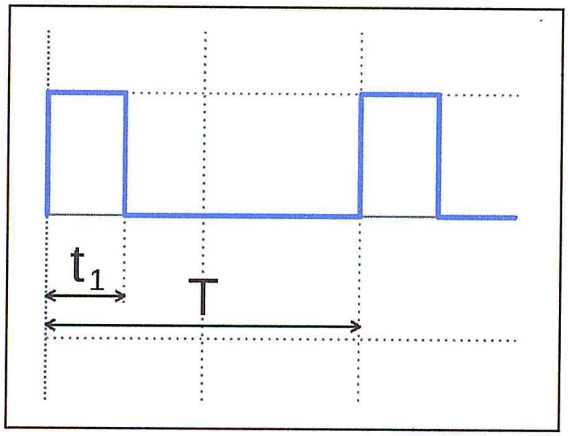


Figure III.7. Période et rapport cyclique d'un PWM.

Dans notre cas, cette fonction est réalisée par le module OPB Timer-Counter [17]. Ce dernier contient deux compteurs configurables (jusqu'à 32 bits). Il peut être utilisé en 3 modes, entre autres le mode PWM. Le tableau suivant présente les différents registres présents sur le module OPB Timer-Counter :

Tableau III.2. Registres du OPB Timer-Counter.

Registre	Offset	Taille	Accès	Description
TCSR0	0x00	32 bits	Lecture/Ecriture	Registre de contrôle du premier compteur.
TLR0	0x04	32 bits	Lecture/Ecriture	Registre de chargement
TCR0	0x08	32 bits	Lecture	Registre compteur
TCSR1	0x10	32 bits	Lecture/Ecriture	Registre de contrôle du deuxième compteur.
TLR1	0x14	32 bits	Lecture/Ecriture	Registre de chargement
TCR1	0x18	32 bits	Lecture	Registre du compteur

Les registres TCSR0 et TCSR1 sont des registres de contrôle. Ils permettent la configuration du mode de fonctionnement du module ainsi que l'activation ou bien la désactivation des interruptions. Le registre TCR0 (respectivement TCR1) contient la valeur courante du premier (respectivement du deuxième) compteur du module. Les registres TLR1 et TLR0 permettent la définition de la période et du rapport cyclique du PWM tel que :

- Le registre TLR0 permet de spécifier la valeur de la période du PWM comme suit:

$$PERIOD\_PWM = (TLR0 + 2) * OPB\_CLK\_PERIOD \quad (1)$$

- Le rapport cyclique est calculé à partir de la valeur du registre TLR1 d'après le Datasheet [17] comme suit:

$$PWM\_HIGH\_TIME = (TLR1 + 2) * OPB\_CLK\_PERIOD \quad (2)$$

En sachant que le rapport cyclique est défini comme suit :

$$R = (PWM\_HIGH\_TIME / PERIOD\_PWM) * 100$$

En utilisant les équations (1) et (2) on obtient :

$$R = [(TLR1 + 2) / (TLR0 + 2)] * 100 \quad (3)$$

Tel que (1 / OPB_CLK_PERIOD) représente la fréquence à laquelle fonctionne le système.

Si on veut obtenir un signal PWM avec une fréquence et un rapport cyclique donné, il suffit simplement de calculer les valeurs correspondantes de TLR0 et TLR1 en utilisant les équations (1) et (3). De l'équation (1) on obtient:

$$TLR0 = (PERIOD\_PWM / OPB\_CLK\_PERIOD) - 2 \quad (4)$$

On sait que PERIOD_PWM = 1/FREQ_PWM et que OPB_CLK_PERIOD = 1/FREQ_OPB et en les remplaçant dans l'équation (4) on obtient :

$$TLR0 = ((1 / FREQ\_PWM) * FREQ\_OPB) - 2 \quad (5)$$

De même, on utilise l'équation (3) pour calculer la valeur à mettre dans le registre TLR1 pour obtenir un rapport cyclique donné :

$$TLR1 = [(R / 100) * (TLR0 + 2)] - 2 \quad (6)$$

Par exemple si on veut obtenir un signal PWM avec une fréquence de 20 Hz et un rapport cyclique de 60% en sachant que la fréquence de fonctionnement du système est de 50 MHz, il suffit d'utiliser les équations (5) et (6) pour calculer les valeurs de TLR0 et TLR0 correspondantes :

$TLR0 = ((1 / 20) * 50 * 10^6) - 2 = 2500 * 10^3 - 2 = 2499998$

$TLR1 = ((60 / 100) * (2499998 + 2)) - 2 = 1499998$

Il faut noter que le mode PWM n'est pas pris en charge par les pilotes fournis avec l'outil EDK. Dans ce qui suit nous présentons les pilotes qui ont été développés afin de prendre en charge ce mode de fonctionnement.

II.2.1. Développement des Pilotes pour le Mode PWM

Les pilotes représentent une interface entre l'application (software) et le matériel (Hardware), offrant la possibilité d'interaction entre ces deux derniers. Ils permettent un accès bas niveau aux registres internes du module soit en lecture soit en écriture (Figure III.8). L'accès à ces registres permet la gestion du mode interruption, l'activation ou la désactivation du module, ou bien le lancement d'une fonctionnalité donnée, etc.

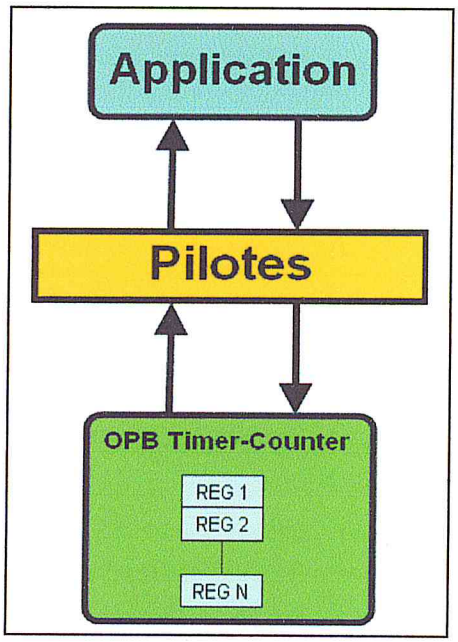


Figure III.8. Accès bas niveau aux registres à travers les pilotes.

Les pilotes sont écrit en respectant le datasheet du module OPB Timer-Counter qui contient toutes les informations nécessaires à propos des registres existants ainsi que la signification de chaque bit de ces derniers. Par exemple, pour basculer vers le mode PWM il faut effectuer un certains nombre d'opérations sur les bits des registres TCSR0 et TCSR1 :

- Mettre le bit 29 des registres TCSR0 et TCSR1 à '1' ;
- Mettre le bit 31 des registres TCSR0 et TCSR1 à '0' ;
- Enfin activer le PWM en mettant le bit 22 des registres TCSR0 et TCSR1 à '1'.

Le tableau III.3 présente les principaux pilotes qui ont été développés en C embarqué. Ces derniers permettent le passage au mode PWM ainsi que la spécification de la période et du rapport cyclique.

Tableau III.3. Pilotes permettant l'initialisation et l'utilisation du mode PWM.

Nom du Pilote	Description
PWM_Set_TLR0 (TLR1)	Permet d'écrire une valeur dans le registre TLR0 (TLR1).
PWM_Disable_MDT_TCSR0 (TCSR1)	Mettre le bit 31 du registre TCSR0 (TCSR1) à '0'. Cette fonction doit être exécutée avant l'activation du mode PWM.
PWM_Enable_MDT_TCSR0 (TCSR1)	Mettre le bit 31 du registre TCSR0 (TCSR1) à '1'.
PWM_Disable_PWM0_TCSR0 (TCSR1)	Mettre le bit 22 du registre TCSR0 (TCSR1) à '0'. Cette fonction désactive le mode PWM du module OPB Timer-Counter.
PWM_Enable_PWM0_TCSR0 (TCSR1)	Mettre le bit 22 du registre TCSR0 (TCSR1) à '1'. Cette fonction active le mode PWM du module OPB Timer-Counter.
PWM_Disable_GENT_TCSR0 (TCSR1)	Mettre le bit 29 du registre TCSR0 (TCSR1) à '0'.
PWM_Enable_GENT_TCSR0 (TCSR1)	Mettre le bit 29 du registre TCSR0 (TCSR1) à '1'. Cette fonction doit être exécutée avant l'activation du mode PWM.
PWM_Disable_ENALL_TCSR0 (TCSR1)	Mettre le bit 21 du registre TCSR0 (TCSR1) à '0'. Cette fonction permet d'activer les Timers du module OPB Timer-Counter.
PWM_Enable_ENALL_TCSR0 (TCSR1)	Mettre le bit 21 du registre TCSR0 (TCSR1) à '1'. Cette fonction permet de désactiver les Timers du module OPB Timer-Counter.
PWM_Disable_INTR_TCSR0 (TCSR1)	Mettre le bit 25 du registre TCSR0 (TCSR1) à '0'. Cette fonction permet de désactiver les interruptions.
PWM_Enable_INTR_TCSR0 (TCSR1)	Mettre le bit 25 du registre TCSR0 (TCSR1) à '1'. Cette fonction d'activer l'effet des interruptions.
PWM_Clear_INTR_TCSR0 (TCSR1)	Mettre le bit 23 du registre TCSR0 (TCSR1) à '1'. Cette fonction permet de remettre à zéro le signal d'interruption.

Pour le test du module OPB Timer-Counter en mode PWM, nous avons procédé de la même manière adoptée pour le test du OPB-CNA. La figure III.9 présente le résultat de génération de différents signaux PWM sur l'oscilloscope.

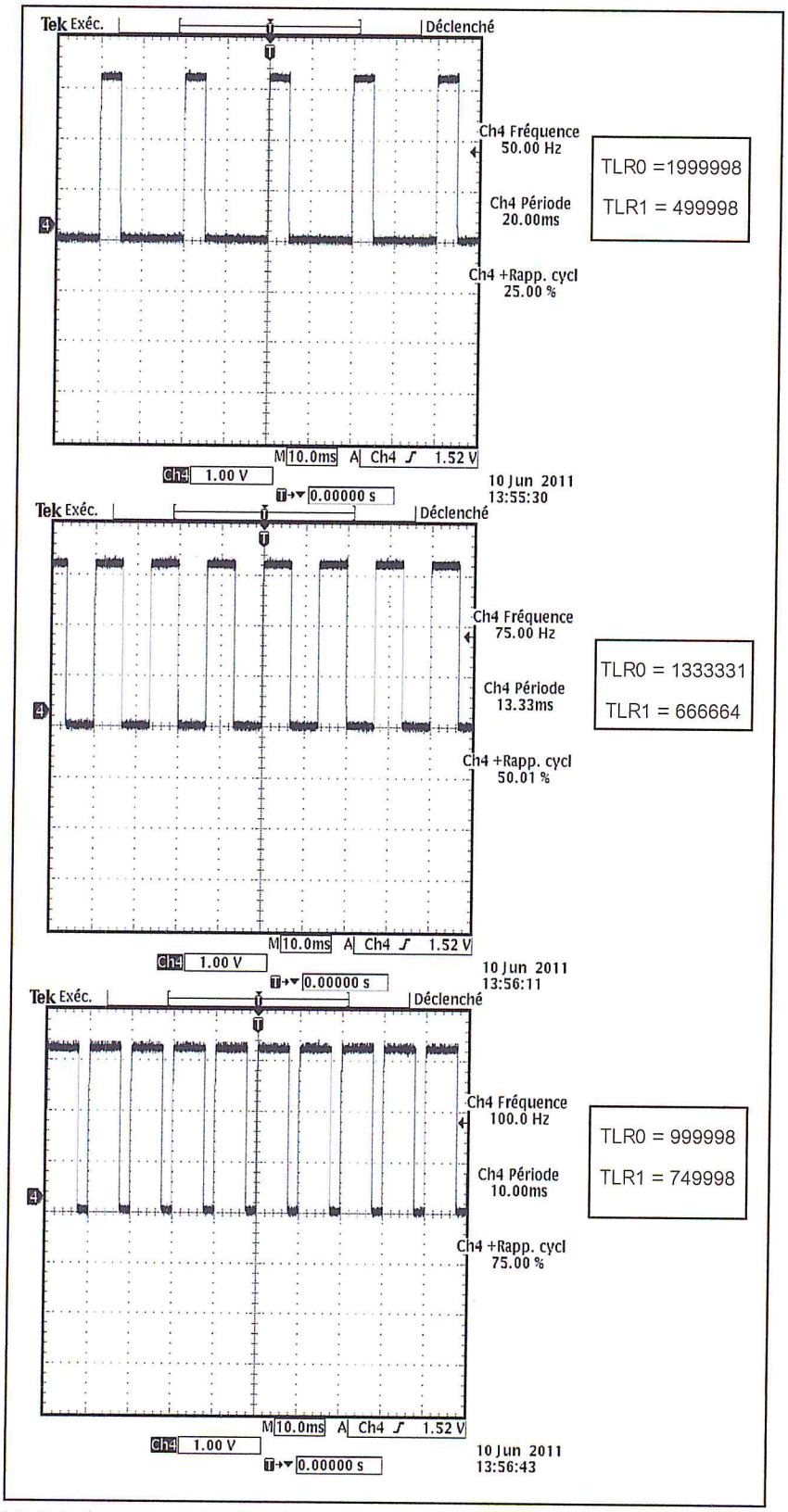


Figure III.9. Résultats de génération de signaux PWM avec différentes périodes et rapports cycliques.

III. Intégration de la Partie Acquisition

La partie d'acquisition (ou de mesure) fournit à la partie de contrôle les données issues d'un capteur permettant de mesurer l'état ou la situation du processus à contrôler. Cette fonction est réalisée par un Convertisseur Analogique/Numérique (CAN). Dans notre cas, la plateforme Memec V2MB1000 ne possède pas un circuit jouant le rôle d'un CAN. Néanmoins, une solution alternative consiste à réutiliser le module OPB Delta Sigma Analog-to-Digital [18]. Ce dernier est un module qui permet de réaliser la conversion Analogique/Numérique. Les principales spécifications de ce module sont :

- Compatible avec le Bus OPB 32 bits ;
- Fréquence de fonctionnement pouvant atteindre 100 Mhz ;
- 16 entrées pour le FIFO;
- Résolution paramétrable (8 où 10 bits).

Toutefois, ce module nécessite l'utilisation d'une certaine électronique externe à la plateforme V2MB1000 (Figure III.10). Ce dispositif électronique est constitué des composants suivants :

- Un filtre passe-bas à la sortie du signal de référence;
- Un comparateur qui a comme entrée le signal de référence et le signal analogique provenant du système à contrôler.

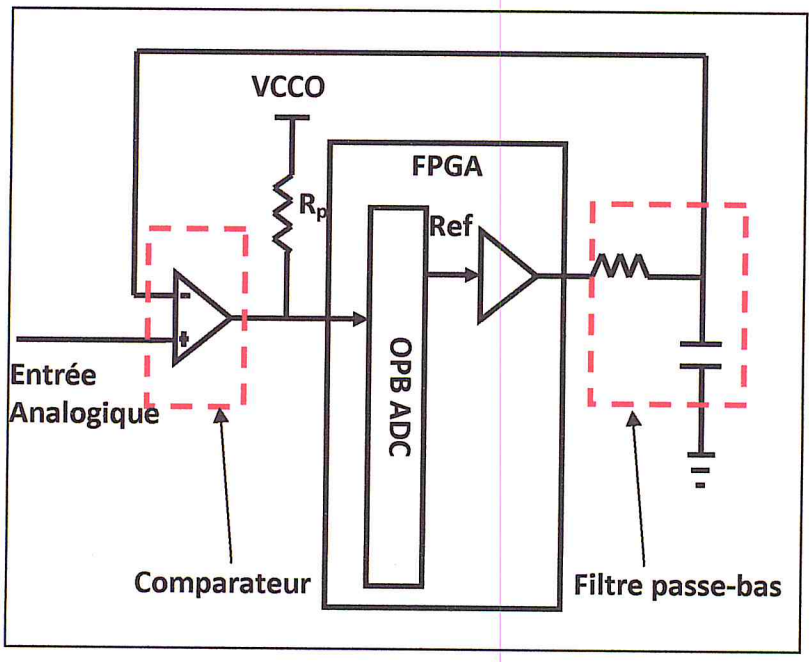


Figure III.10. Réalisation d'un CAN à l'aide d'un OPB-CAN.

Le module OPB-CAN convertit un niveau de tension en une valeur numérique non signée dont le minimum est 0 qui correspond à une tension de 0 volts et le maximum est 2^n-1 qui correspond à une tension de 3.3 volts (tension maximum lue par la plateforme V2MB1000), tel que *n* représente la résolution du OPB-CAN. Le tableau III.4 présente les différents registres présents sur le module OPB-CAN :

Tableau III.4. Registres du OPB-CAN.

Registre	Offset	Accès	Description
Registre de Contrôle (CR)	0x01C	Lecture/Ecriture	Registre de contrôle du CAN.
FIFO	0x104	Lecture	File d'attente FIFO pour les données converties.
Registre d'occupation du FIFO (OCCY)	0x108	Lecture	Nombre d'éléments dans le FIFO.
Registre d'état du FIFO (IPIPR 1 bit)	0x020	Lecture	'1' : Le FIFO contient des données; '0' : Le FIFO est vide.
Registre d'interruption (IPIER 1 bit)	0x028	Lecture/Ecriture	'1' : Interruption du FIFO activée; 0 : Interruption du FIFO désactivée.

Pour le test de la partie d'acquisition nous avons utilisé un synthétiseur de signaux Keithly 3940 [19] dont la sortie est connectée à l'amplificateur opérationnel LM741 [20] jouant le rôle de comparateur. L'autre entrée du comparateur est connectée au signal de référence provenant de la plateforme V2MB1000. Le synthétiseur de signaux est utilisé pour générer un signal analogique en dents de scie. Les valeurs lues par le module d'acquisition sont envoyées à travers le RS232 et affichées sur le PC par logiciel Tera Term Pro. La figure III.11 représente le schéma synoptique du montage réalisé pour le test.

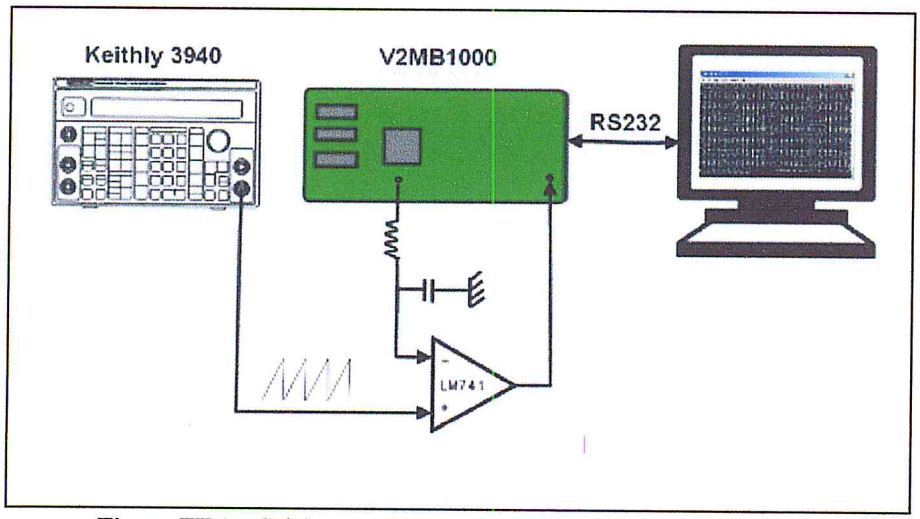


Figure III.11. Schéma synoptique du montage de test du OPB-CAN.

L'amplitude du signal analogique généré par le synthétiseur de signaux a été réglée pour quelle soit comprise dans l'intervalle [0 volts, 3.3 volts]. Ce dernier représente la plage des valeurs de tensions pouvant être lues par la plateforme V2MB1000. Nous avons configuré notre OPB-CAN pour une résolution de 10 bits, donc les valeurs lues seront comprises entre 0 et 1023 (2^{10}-1). Tel que 0 correspond à une tension d'entrée de 0 volts et 1023 correspond à la tension maximale de 3.3 volts. Le signal en dents de scie varie linéairement dans le temps entre les valeurs 0 volt et 3.3 volt, donc le résultat sur le PC doit correspondre à un affichage des valeurs en ordre croissant entre 0 et 1023. La figure III.12 représente le résultat du test affiché par le logiciel Tera Term Pro, qui valide le bon fonctionnement de la partie d'acquisition.

Le synthétiseur de signaux qui a permis de générer le signal analogique sera remplacé par la suite par un capteur dont la sortie sera connectée à l'entrée du dispositif d'acquisition. Ce capteur permettra au système de contrôle embarqué de connaitre l'état du processus.

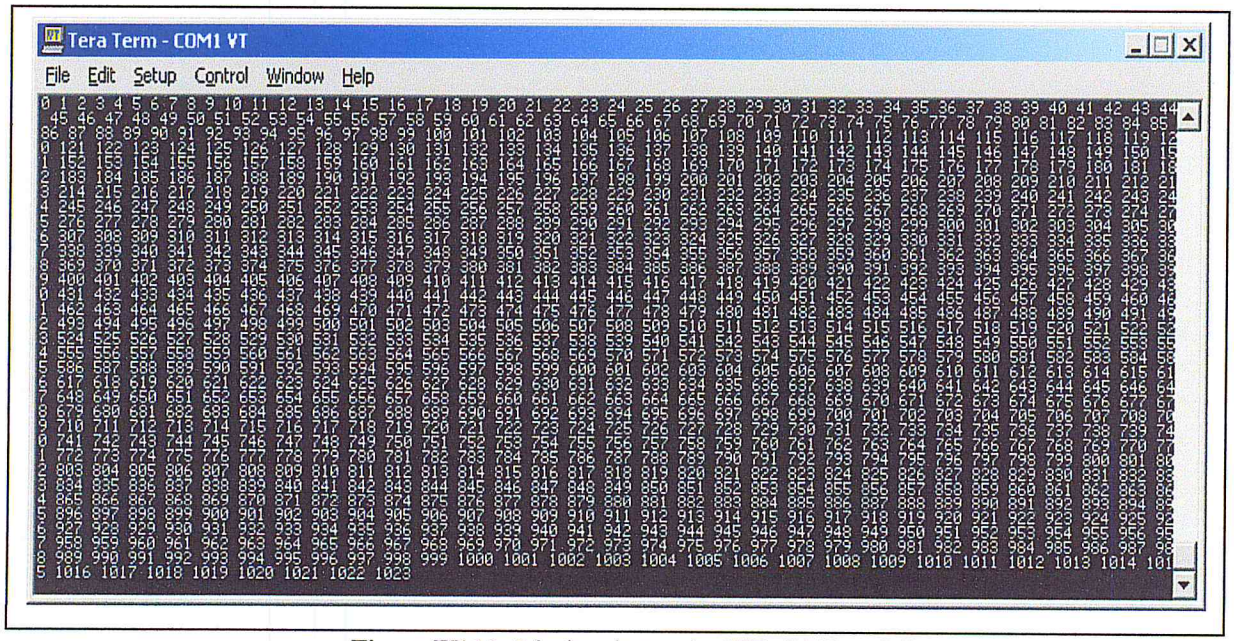


Figure III.12. Résultat du test du OPB-CAN sur PC.

IV. Intégration de la Partie de Contrôle

Cette partie représente le cœur du système de contrôle (ou de régulation) embarqué. En effet, elle permet de calculer la commande (qui sera appliquée par la partie de commande) permettant de modifier l'état du processus à contrôler. Ce calcul tient compte des données issues de la partie d'acquisition et de la consigne imposée par l'utilisateur du système. La partie de contrôle a été développée sous forme d'un module hardware (IP: Intellectual Property) écrit en langage HDL, et ensuite a été intégrée dans notre système. Dans ce qui suit

nous allons résumer les étapes de développement et de test qui ont permis d'abord la vérification fonctionnelle et par la suite l'intégration de cette partie.

IV.1. Développement et Test Fonctionnel

IV.1.1. Notion d'IP et Méthodologie de Conception

La partie de contrôle a été réalisée en utilisant la méthodologie de développement d'IP (Propriété Intellectuelle) [21] qui garantit un certain niveau de qualité avant le déploiement final du produit. Un IP est un module équipé de capacités de réutilisation (IP = module + capacité de réutilisation) qui peut être acheté, re-exploité et ajusté pour une application particulière. En plus d'être réutilisable, un module doit être :

- Configurable : le module est conçu pour résoudre un problème général ;
- Portable : le module est conçu indépendamment de la technologie ;
- Lisible : le module doit être bien documenté sur la gamme d'applications, interfaces, contraintes,... ;
- Debuguable : le module doit être vérifié avec un degré élevé de confiance.

Ces propriétés nous permettent donc de classifier les IPs par leur niveau de qualité comme suit (voir aussi Figure III.13):

- Fonctionnel : représente le niveau minimum qu'un module doit atteindre pour pouvoir être déployé ;
- Maintenable : par rapport au niveau fonctionnel, un bloc maintenable serait aussi fonctionnel, mais également bien documenté (schémas, codage propre et clair, codes commentés, spécifications, sommaires, ...) ;
- Réutilisable : ce niveau est un enrichissement par rapport au niveau précédent.

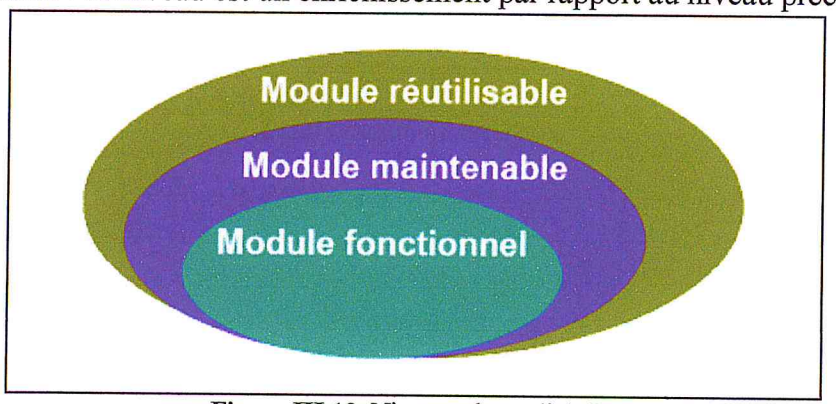


Figure III.13. Niveaux de qualité d'un IP.

Le passage d'un niveau à un autre peut se faire mais en faisant un effort supplémentaire de conception.

Pour garantir toutes ces propriétés, la méthodologie de développement d'IPs doit suivre un chemin bien tracé depuis la spécification des besoins jusqu'au test de validation final dans un environnement réel. La figure III.14 représente le flot de conception d'un IP qui passe par les étapes suivantes :

- Spécification des besoins : cette étape permet d'élaborer un cahier des charges après avoir effectué une étude du marché sur les produits commerciaux déjà existants ;
- Spécification fonctionnelle : cette étape permet de spécifier ce que l'IP doit faire et pas comment le faire. Elle ne contient pas les détails d'implémentations mais une description exacte sur les futurs caractéristiques de l'IP ;
- Conception architecturale : dans cette étape l'IP est décrit sous forme de blocs et d'interconnexions entre ces derniers. Les détails d'implémentation de chaque bloc sont omis dans cette étape ;
- Conception détaillée : dans cette étape on s'intéresse aux détails d'implémentation de chaque bloc établi dans l'étape précédente ;
- Vérification et validation : dans cette étape deux genres de test sont effectués :
  - Test fonctionnel : ce test permet de vérifier d'abord le fonctionnement de chaque bloc, ensuite de l'IP complet par la suite [22];
  - Test au niveau système : ce test permet de valider le fonctionnement de l'IP dans un système embarqué (Processeur + Bus + Périphériques).

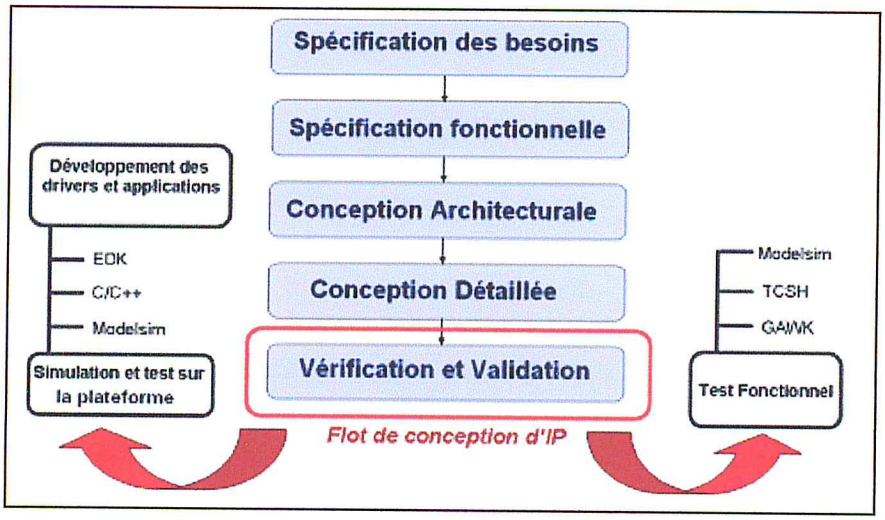


Figure III.14. Flot de conception d'IPs.

Dans notre cas nous nous intéressons à la dernière étape qui consiste à intégrer l'IP dans un système embarqué, d'écrire les différents pilotes permettant l'utilisation du module et enfin de procéder aux différents tests de vérification de validation dans un environnement réel sur la plateforme MEMEC V2MB1000.

Il faut noter que l'IP jouant le rôle de la partie de contrôle a été déjà développé et a subi une vérification fonctionnelle conjointement sur Modelsim et dans l'environnement Simulink sur Matlab [23]. Avant d'aborder l'intégration de l'IP de contrôle, nous allons d'abord présenter une brève description de l'architecture interne de ce dernier ainsi que le test fonctionnel sur Simulink et Modelsim.

IV.1.2. Description et Test Fonctionnel

La fonctionnalité de contrôle ou bien de régulation est réalisée par un PID. Ce dernier représente le moyen de contrôle en boucle fermée le plus utilisé dans l'industrie grâce à sa simplicité et à sa robustesse. La figure III.15 représente un système en boucle fermée typique utilisant un contrôleur PID où y(k), uc(k) et u(k) représentent respectivement la donnée issue du capteur, la consigne de l'utilisateur et la commande calculée par le PID au $k^{ème}$ instant d'échantillonnage.

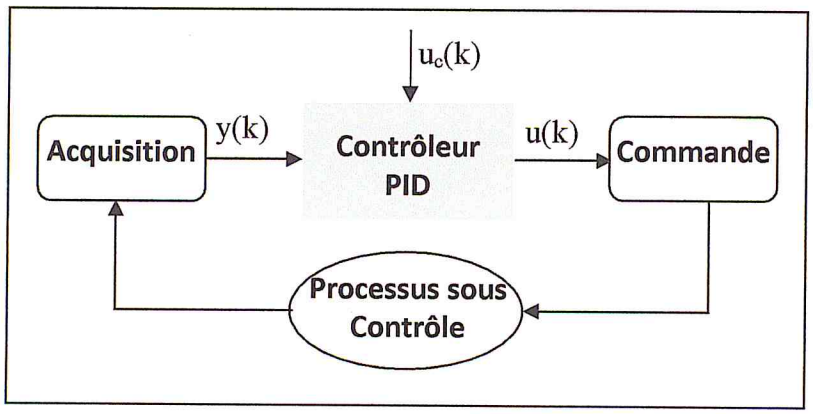


Figure III.15. Système de contrôle en boucle fermée utilisant un PID.

La commande calculée par le PID qui doit être appliquée sur le processus à contrôler est définie comme suit [11]:

$$u(k) = P(k) + I(k) + D(k)$$

Tels que :

$$P(k) = A \cdot u_c(k) + B \cdot y(k), I(k) = I(k-1) + C \cdot e(k-1) \text{ et } D(k) = D \cdot D(k-1) + E \cdot f(k)$$

P représente l'action proportionnelle, I l'action intégrale et D l'action dérivé à un instant K.

Avec : $e(k-1)=u_c(k-1)-y(k-1)$ et $f(k)=y(k)-y(k-1)$.

Tel que e représente l'erreur entre la consigne spécifiée par l'utilisateur (u_c) et l'état actuel du processus à contrôler (y) à un instant k. Le tableau III.5 présente les équations permettant le calcul des coefficients A, B, C et D.

Tableau III.5. Equations de calcul des coefficients.

Coefficients	
A	$K_p b$
B	$-K_p$
C	$K_p \frac{T_s}{T_i}$
D	$\frac{T_d}{T_d + N T_s}$
E	$-\frac{K_p T_d N}{T_d + N T_s}$

Tels que K_p représente le gain proportionnel ; T_i et T_d sont respectivement les temps du dérivé et celui de l'intégral ; N est la valeur maximale du gain dérivé ; b permet de limiter l'effet de l'action proportionnel et T_s est le temps d'échantillonnage dans lequel l'état du processus à contrôler est transféré à la partie de contrôle.

La représentation de la commande, de la mesure et de la consigne au niveau de l'IP PID est en virgule fixe. Cette représentation permet d'une part d'économiser les ressources utilisées au niveau du FPGA mais aussi de réduire la consommation de puissance de ce dernier. Le codage d'un nombre réel en virgule fixe est présenté par la figure III.16. Un nombre codé en virgule fixe est composé d'une partie entière, d'une partie fractionnaire et d'un bit de signe. En augmentant la taille de la partie fractionnaire on augmente par la même occasion la précision du calcul.

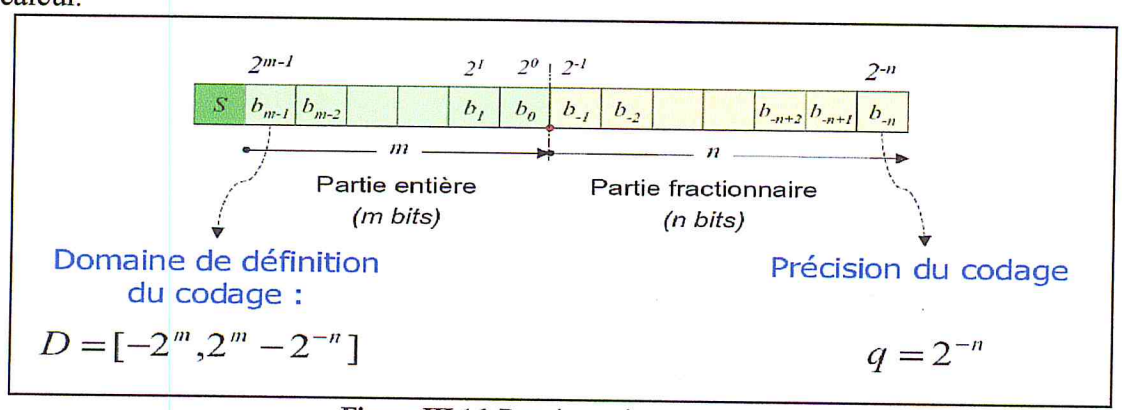


Figure III.16. Représentation en virgule fixe.

Dans ce qui suit nous donnons à titre d'informations les détails d'implémentation de l'IP PID. Une implémentation intuitive et parallèle du PID requiert en total 7 additionneurs/soustracteurs et 5 modules de multiplication. Dans la conception digitale le nombre total de portes logiques utilisées est linéairement lié à la taille du mot pour un additionneur et quadratiquement pour un multiplieur. Cette façon de faire accroit considérablement le nombre de ressources utilisées sur le FPGA, augmentant ainsi la consommation de puissance. Il est donc clair que l'effort d'optimisation de la consommation de puissance doit être concentré sur l'implémentation sérielle de la fonction MAC (Multiply And Accumulate X×Y) [24]. Une variante optimisée de cette dernière qui permet de calculer X×Y+T×Z (ODMAC : Optimized Double Multiply And Accumulate) a été implémentée de telle façon à optimiser le nombre de ressources utilisées et la consommation de puissance. Ces deux fonctions sont basées sur l'algorithme de multiplication RMRMA (Recursive Multibit Recording Multiplication Algorithm) [25] qui est une amélioration de l'algorithme de multiplication de Booth (BMA) [26] et de l'algorithme de multiplication de Booth modifié (MBMA) [27]. La figure III.17 représente l'architecture de l'IP PID.

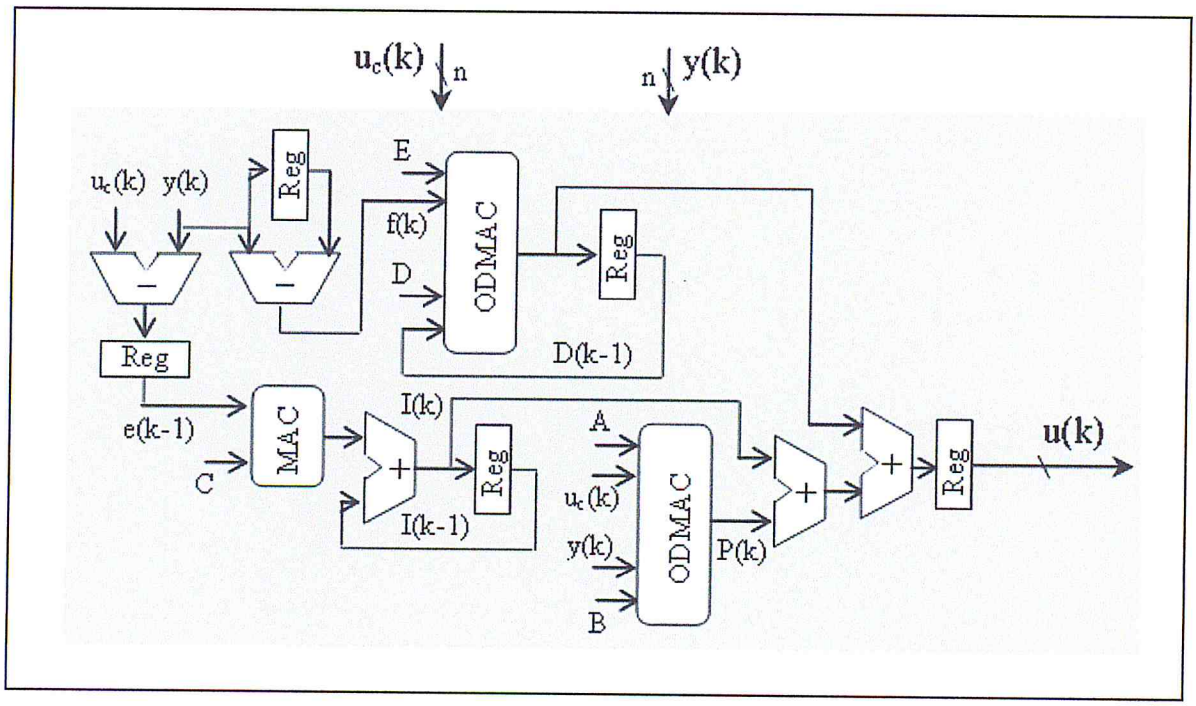


Figure III.17. Architecture de l'IP PID.

Les entrées de l'IP sont la consigne imposée par l'utilisateur du système ($u_c(k)$) et la mesure représentant l'état de ce dernier à un instant k (y(k)). La sortie du module n'est que la commande (u(k)) à appliquer sur le processus à contrôler. Les coefficients A, B, C, D et E qui servent à paramétrer l'IP PID sont implémentés sous forme de registres dont les valeurs sont calculées en utilisant les équations du tableau II.5 et les paramètres K_p, T_i, T_d, T_s, N et b.

L'étape suivante (test et vérification fonctionnelle) consiste à vérifier le bon fonctionnement de notre IP PID. Cela permet dans un premier temps de valider l'IP avant de procéder à son intégration au sein d'un système embarqué et finalement de le tester dans un environnement réel. Pour ce faire, l'IP PID a été intégré dans Simulink sous forme d'un Bloc HDL dont les entrées sont stimulées par des vecteurs de test. Ces derniers sont générés par Simulink et transmis à l'outil Modelsim qui effectue la simulation HDL, qui a comme résultat la commande calculée par l'IP PID qui représente la sortie de ce dernier. Le transfert de données entre les deux outils Simulink et Modelsim se fait d'une manière transparente avec l'outil LINK FOR MODELSIM. Cette façon de faire est appelée Co-Simulation HDL (voir figure III.18).

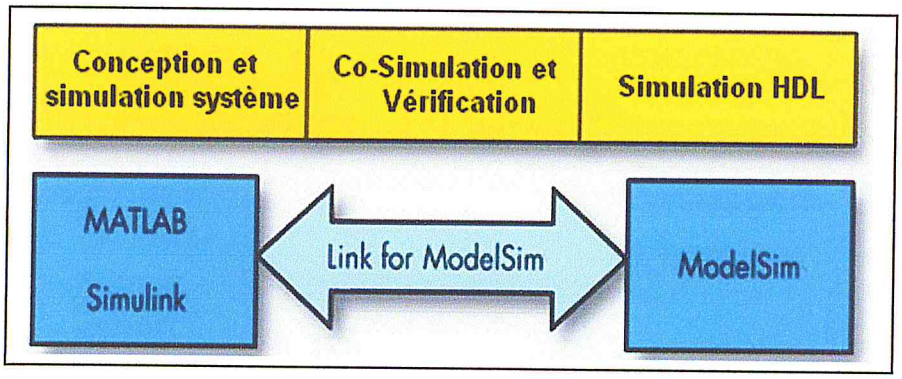


Figure III.18. Interface de Co-Simulation Simulink/ModelSim.

Afin de tester le bon fonctionnement de notre IP PID, deux environnements virtuels ont été mis en place. Le premier possède comme module de contrôle notre IP PID et le second un Bloc PID idéal (se trouve au sein de la bibliothèque de Simulink) qui servira comme référence pour notre PID. La figure III.19 représente les environnements crées sous Simulink.

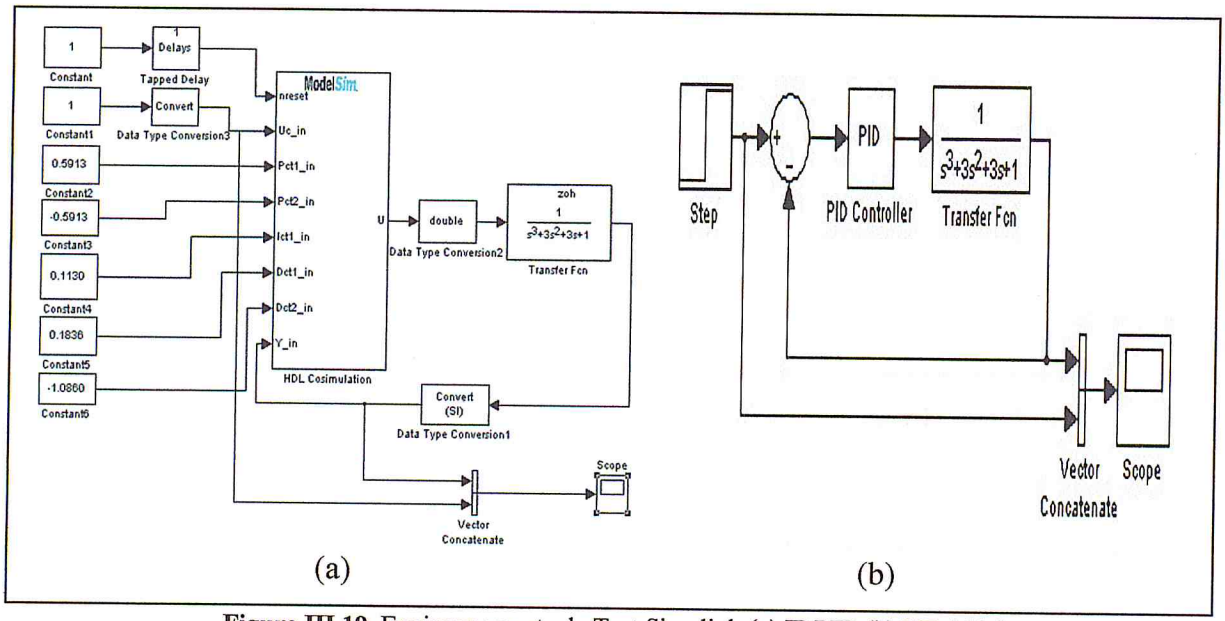


Figure III.19. Environnements de Test Simulink (a) IP PID (b) PID idéal.

Le processus à contrôler est représenté sous forme d'une fonction de transfert [28] qui permet de modéliser son comportement et dont la sortie représente son état qui sera affiché avec la consigne (sur la courbe des résultats) et injecté au module de contrôle (PID idéal et IP PID) pour le calcul de la prochaine commande. Nous avons ensuite procédé à plusieurs types de tests pour valider le fonctionnement de l'IP PID. Le premier test consiste à vérifier l'impact de la taille de la partie fractionnaire sur la qualité de la régulation par rapport à celle effectuée par le PID idéal. Le deuxième test consiste à vérifier la stabilité de l'IP PID face à des perturbations externes (test de stabilité). Le dernier test permet de vérifier la capacité de l'IP PID à suivre une consigne variable dans le temps (test de poursuite). La figure III.20 présente les résultats obtenus des différents tests cités ci-dessus.

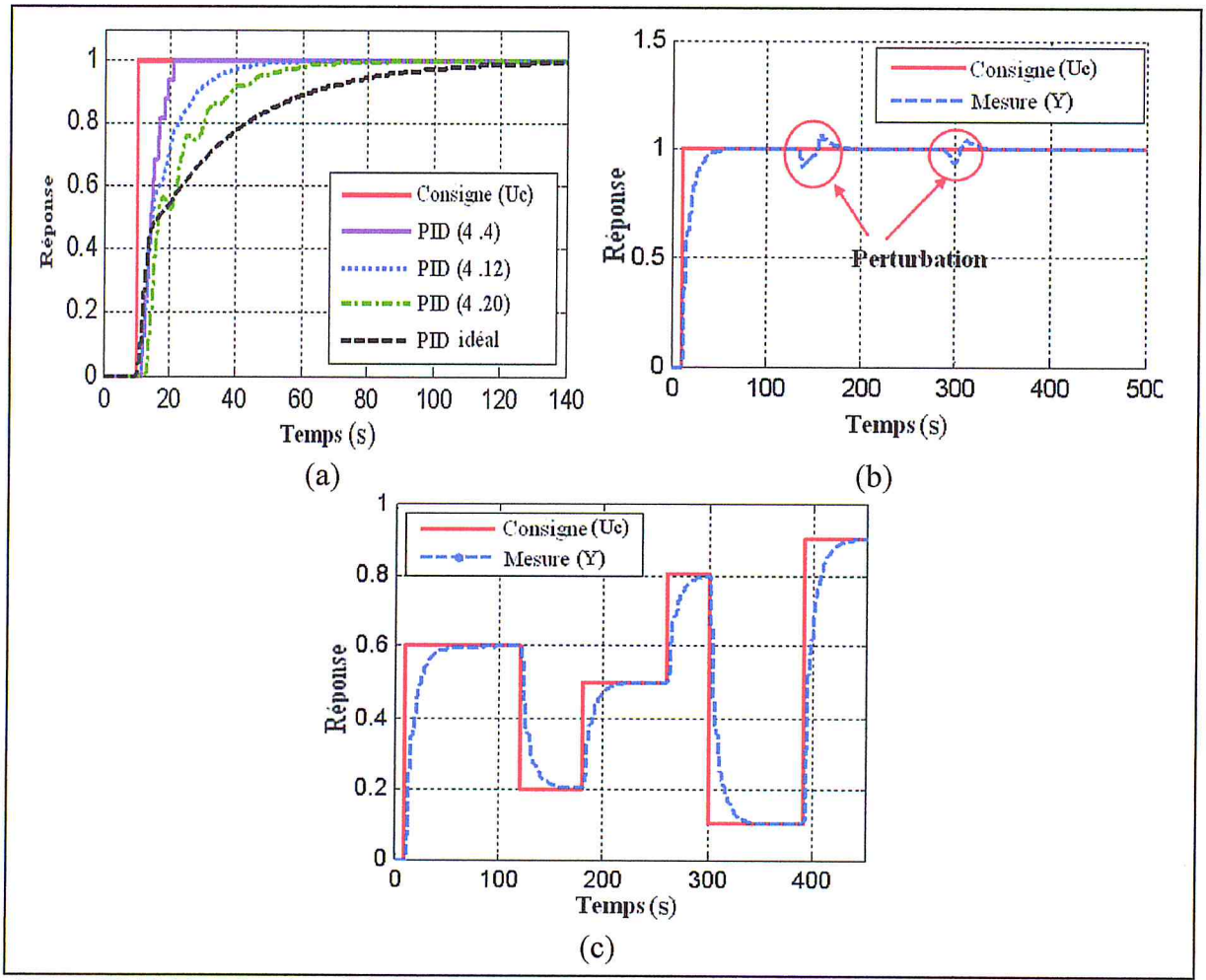


Figure III.20. Tests fonctionnel de l'IP PID (a) Impact de la précision (b) Test de stabilité (c) Test de poursuite.

On remarque d'après la figure III.20.a que plus on augmente la taille de la partie factionnaire plus la précision du calcul augmente et plus le résultat se rapproche de celui du PID idéal. Pour le test de stabilité qui consiste à exercer des séries de perturbations, on remarque que l'IP PID permet à l'état du processus de se stabiliser sur la valeur de la consigne après un certain temps.

Enfin pour le test de poursuite, on remarque d'après la figure III.20.c, que l'IP PID est capable de poursuivre une consigne avec amplitude et durée variables dans le temps. Dans la section suivante, nous allons présenter les étapes d'intégration de notre IP PID dans le système de contrôle ainsi que les pilotes qui ont été développés pour utiliser et accéder ce dernier.

IV.2. Intégration et Développement des Pilotes

Nous avons vu que notre système de contrôle embarqué est constitué d'un processeur (Microblaze) relié à travers le bus OPB à plusieurs modules et périphériques dont notre IP PID fait partie. Les parties de commande et d'acquisition disposent déjà d'une interface leurs permettant d'être exploitées par une application embarquée puisque nous les avons réutilisées et adaptées à notre cas. Il s'agit donc de développer une interface (ou bien Wrapper) [29] [30] qui permet au processeur de communiquer avec notre IP en utilisant le protocole du bus OPB. Cette interface gère les signaux en provenance du processeur vers notre IP ainsi que les données qui doivent soit être lues à partir des registres internes ou bien écrites vers ces derniers. Dans la figure II.1, nous avons présenté un système de base ne contenant que les modules nécessaires pour un fonctionnement minimal. La figure III.21 représente le même système après y avoir intégré les parties de contrôle, de commande et d'acquisition.

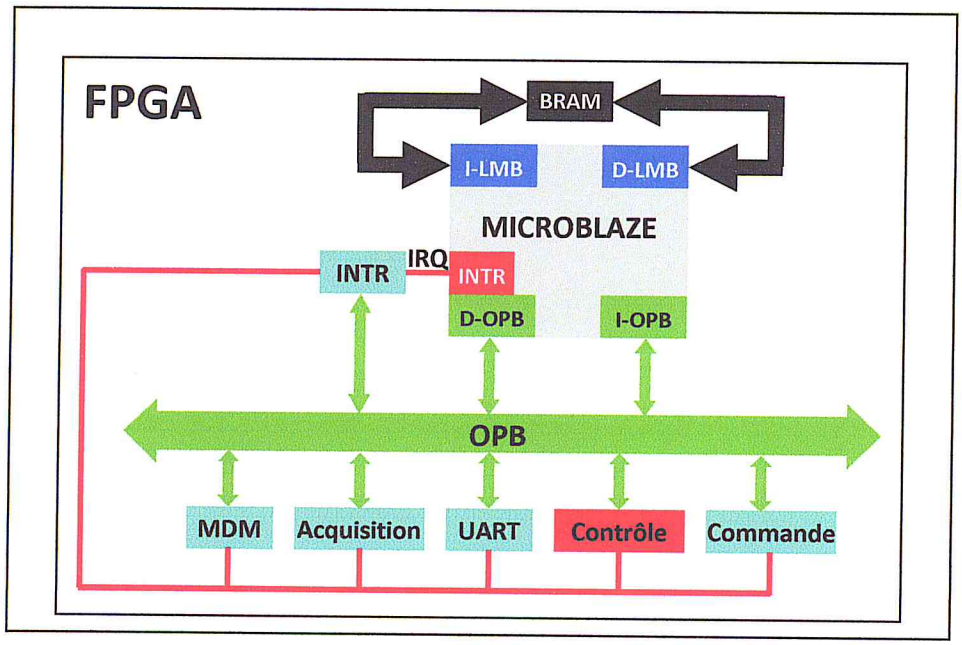


Figure III.21. Système de contrôle embarqué.

Notre IP PID est constitué de deux types de registres, ceux utilisés pour la configuration et ceux des entrées/sorties. Les registres du premier type contiennent les valeurs des coefficients A, B, C, D et E, tandis que les registres du deuxième type contiennent la commande, la mesure

et la consigne imposée par l'utilisateur du système. L'interface ou le Wrapper à développer doit pouvoir interpréter les signaux du bus OPB venant de la part du processeur qui sont soit des opérations de lecture soit d'écriture dans les registres internes de l'IP. Les signaux OPB qui doivent être traités par l'interface de l'IP PID sont :

- OPB_CLK : ce signal représente l'horloge du système. C'est un signal de synchronisation ;
- OPB_RESET : ce signal permet une remise à zéro (RESET) du système ;
- OPB_DIN : représente le bus de données en entrée. Il permet de véhiculer les données à écrire dans un registre donné ;
- OPB_DOUT : représente le bus de données en sortie. Il permet de véhiculer les données lues à partir d'un registre donné ;
- OPB_CS (CS pour Chip Select): ce signal indique que le processeur veut sélectionner un périphérique donné pour effectuer une opération de lecture ou bien d'écriture ;
- OPB_RW : ce signal indique le type d'opération à effectuer ('0' pour écriture et '1' pour lecture) ;
- OPB_ACK : est un signal d'acquittement qui indique au processeur la fin d'une opération de lecture ou d'écriture ;
- OPB_ADR : représente le bus d'adresse. Ce dernier sert à véhiculer l'adresse du registre à lire ou bien à modifier.

Il faut savoir que chaque périphérique possède une plage d'adresse ou bien un intervalle entre l'adresse de base (Base_Address : adresse de début) et l'adresse haute (High_Address : fin). Les adresses des registres internes d'un périphérique donné sont situées par rapport à l'adresse de base, par exemple :

$$ADDR\_REG = BASE\_ADDRESS\_PERIPH + OFFSET$$

Telle que ADDR_REG est l'adresse d'un registre donné, BASE_ADRESS_PERIPH est l'adresse de base du périphérique et OFFSET est le déplacement par rapport à cette dernière. Pour effectuer une opération de lecture le processeur doit procéder comme suit :

- Mettre dans OPB_ADR l'adresse du registre à lire ;
- Mettre le signal OBP_RW à '1' ;
- Mettre le signal OPB_CS à '1'.

De l'autre coté, l'interface doit vérifier si le signal OPB_CS concerne bien notre IP et cela en examinant si l'adresse contenue dans OPB_ADR appartient à l'intervalle d'adresses [IP_Base_Adress, IP_High_Adress]. Si oui, l'interface demande à l'IP d'effectuer une lecture (le signal OPB_RW est à '1') du registre concerné en mettant sa valeur dans OPB_DOUT. Enfin l'interface met le signal OPB_ACK à '1' (pour 1 cycle d'horloge) qui indique au processeur la fin de l'opération de lecture. De même pour une opération d'écriture le processeur doit procéder comme suit :

- Mettre dans OPB_ADR l'adresse du registre à modifier ;
- Mettre dans OPB_DIN la valeur à écrire dans le registre concerné ;
- Mettre le signal OPB_RW à '0' ;
- Mettre le signal OPB_CS à '1'.

Comme pour l'opération de lecture, l'interface doit vérifier si le signal OPB_CS concerne bien notre IP de la même manière que la fois précédente. Si oui, l'interface demande à l'IP d'effectuer une écriture (le signal OPB_RW est à '0') dans le registre dont l'adresse est spécifiée dans OPB_ADR en utilisant la valeur mise dans OPB_DIN. A la fin de l'opération l'interface met le signal OPB_ACK à '1' (pour 1 cycle d'horloge) qui indique au processeur la fin de l'opération. La figure III.22 illustre l'intégration de l'IP PID ainsi que les principaux signaux OPB à gérer.

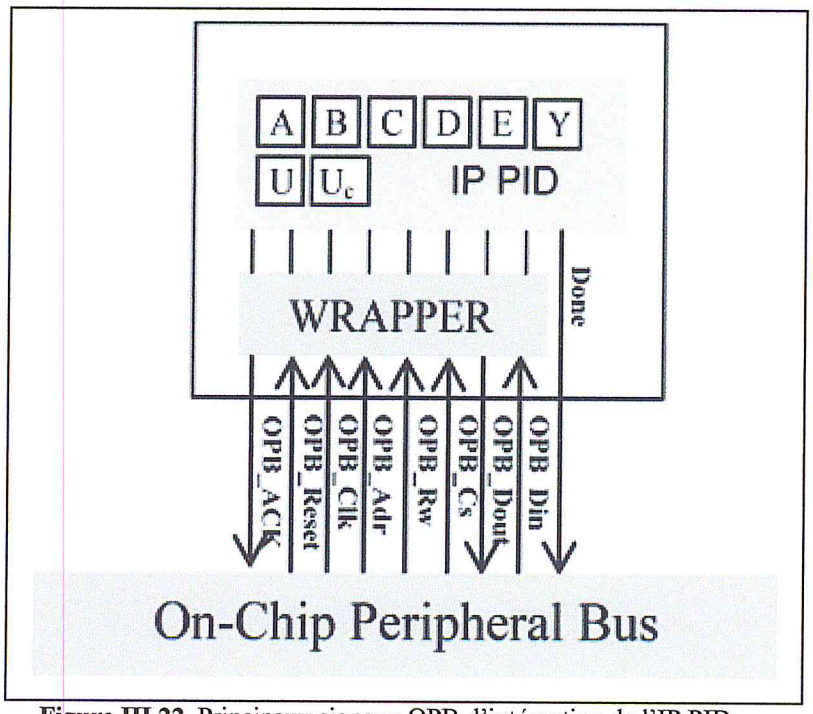


Figure III.22. Principaux signaux OPB d'intégration de l'IP PID.

Le signal **Done** est une interruption qui indique que la commande à appliquer sur le processus a été calculée et peut être lue depuis le registre U. Le traitant (ou bien Handler) de cette interruption doit lire la commande calculée et appliquer cette dernière en utilisant le PWM (Partie de commande) afin de changer l'état du processus à contrôler. La figure III.23 présente le code VHDL de l'interface OPB du IP PID.

```
entity PID_8_8 is
 generic
 (

  C_BASEADDR              : std_logic_vector ;
  C_HIGHADDR              : std_logic_vector ;
  C_OPB_AWIDTH              : integer          := 32;
  C_OPB_DWIDTH              : integer          := 32;
  C_USER_ID_CODE            : integer          := 3;
  C_FAMILY              : string           := "virtex2p"
 );
 port
 (
  ------------------------------------------------------------
        done: out std_logic;
  ------------------------------------------------------------
  OPB_Clk                 : in  std_logic;
  OPB_Rst                 : in  std_logic;
  Sl_DBus                 : out std_logic_vector(0 to C_OPB_DWIDTH-1);
  Sl_errAck               : out std_logic;
  Sl_retry                : out std_logic;
  Sl_toutSup              : out std_logic;
  Sl_xferAck              : out std_logic;
  OPB_ABus                : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
  OPB_BE                  : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
  OPB_DBus                : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
  OPB_RNW                 : in  std_logic;
  OPB_select              : in  std_logic;
  OPB_seqAddr             : in  std_logic
 );
end entity PID_8_8;

architecture IMP of PID_8_8 is

component PID is
 port
 (
  clk: in  std_logic;
  nreset: in  std_logic;
  done: out std_logic;
  din: in  std_logic_vector(0 to 15);
  dout:out  std_logic_vector(0 to 31);
  adr:in  std_logic_vector(0 to 2);
  rw:in  std_logic;
  cs:in std_logic;
  ack_for_OPB:out std_logic
 );
end component PID;
Suite ...
```

Paramètres de l'IP :
- Adresse haute
- Adresse de base
- FPGA cible

Interruption de l'IP

Signaux OPB

Déclaration de l'IP PID

```
---------- Décodage de l'adresse Envoyée Par le processeur ----------
signal ii_xfer_Ack:  std_logic;
signal ii_CS_1:      std_logic;
signal OPB_ABus_reg: std_logic_vector(OPB_ABus'range);
signal OPB_DBus_reg: std_logic_vector(OPB_DBus'range);
signal OPB_RNW_reg:  std_logic;
signal my_Select:    std_logic;
signal my_Ack:       std_logic;
signal my_DBus:      std_logic_vector(0 to 31);
signal NOT_OPB_RST:std_logic;
begin
address_decode: block
      component pselect
           generic (
                 C_AB   : integer := 9;
                 C_AW   : integer := 32;
                 C_BAR  : std_logic_vector(0 to 31) );
           port (
                A     : in  std_logic_vector(0 to C_AW-1);
                AValid : in  std_logic;
                CS    : out std_logic);
      end component;
      function Addr_Bits (x, y : std_logic_vector(0 to C_OPB_AWIDTH-1)) return integer is
            variable addr_nor : std_logic_vector(0 to C_OPB_AWIDTH-1);
       begin
            addr_nor := x xor y;
            for i in 0 to C_OPB_AWIDTH-1 loop
                 if addr_nor(i) = '1' then return i; end if;
            end loop;
            return(C_OPB_AWIDTH);
       end function;
       constant C_AB :    integer:= Addr_Bits(C_HIGHADDR, C_BASEADDR);
       signal  ii_CS_x:  std_logic;
       signal  ii_CS_2:  std_logic;
       signal  ii_CS_1_2: std_logic;
       begin
       pselect_I : pselect    generic map (
            C_AB  => C_AB,
            C_AW  => OPB_ABus'length,
            C_BAR => C_BASEADDR)
       port map (
            A     => OPB_ABus,
            AValid => OPB_select,
            CS    => ii_CS_x);
       CS_1_FF : FDR port map (
            Q => ii_CS_1,
            C => OPB_Clk,
            D => ii_CS_X,
            R => ii_xfer_Ack);
       CS_2_FF: FDR port map (
            Q => ii_CS_2,
            C => OPB_Clk,
            D => ii_CS_1,
            R => ii_xfer_Ack);
      ii_CS_1_2<= ii_CS_1 and not ii_CS_2;
      my_Select<= ii_CS_1_2;
  end block;
Suite...
```

Instanciation du décodeur d'adresse

```
from_OPB: block
      begin
      WRDBUS_FF_GENERATE: for i in 0 to C_OPB_DWIDTH-1 generate
           WRDBUS_FF_I: FDR port map (
                Q => OPB_DBus_reg(i),
                C => OPB_Clk,
                D => OPB_DBus(i),
                R => OPB_Rst);
      end generate;
      ABUS_FF_GENERATE: for i in 0 to C_OPB_AWIDTH-1 generate
           ABUS_FF_I: FDR port map (
                Q => OPB_ABus_reg(i), C => OPB_Clk, D => OPB_ABus(i), R => OPB_Rst);
      end generate;
      RNW_FF_I: FDR port map (
           Q => OPB_RNW_reg, C => OPB_Clk, D => OPB_RNW, R => OPB_Rst);
end block;
to_OPB: block
      signal read_reg_rst: std_logic;
      signal toutSup:     std_logic;
      signal toutSup_rst:  std_logic;
      begin
      read_reg_rst<= not(my_Ack)
      or          not(ii_CS_1)
      or          not(OPB_RNW_reg);
      RDBUS_FF_GENERATE : for i in 0 to C_OPB_DWIDTH-1 generate
           RDBUS_FF_I : FDRE port map (
                Q  => Sl_DBus(i), C  => OPB_Clk, D  => my_DBus(i),
                R  => read_reg_rst, CE => OPB_BE(i/8));
      end generate;
      XFER_ACK_I: FDR port map (
           Q => ii_xfer_Ack, C => OPB_Clk, D => my_ACK, R => OPB_Rst);
      Sl_xferAck <=  ii_xfer_Ack;
      Sl_errAck  <= '0';
      Sl_retry   <= '0';
      toutSup    <= '1';
      toutSup_rst<= not(ii_CS_1)
      or          ii_xfer_Ack;
      TOUTSUP_FF: FDR port map (
           Q  => Sl_toutSup, C  => OPB_Clk, D  => toutSup, R  => toutSup_rst);
end block;
PID_8_8_inst: component PID
 port map
 (
  clk       => OPB_Clk,
  nreset    => NOT_OPB_RST,
  done      => done,
  din       => OPB_DBus_reg(16 to 31),
  dout      => my_DBus(0 to 31),
  adr       => OPB_ABus_reg(29 to 31),
  rw        => OPB_RNW_reg,
  cs        => my_Select,
  ack_for_OPB => my_Ack
 );
NOT_OPB_RST <= not OPB_Rst;
end IMP;
```

Traitement des signaux entrants

Traitement des signaux sortants

Instanciation de l'IP PID

Figure III.23. Interface OPB de l'IP PID.

Après avoir développé l'interface OPB de notre IP PID, l'étape suivante consiste à mettre au point un ensemble de pilotes qui permettent l'utilisation de ce dernier à travers une application software embarquée. Ces pilotes effectuent essentiellement des opérations de lecture ou bien d'écriture sur les registres internes de l'IP. Ceci est réalisé en utilisant deux fonctions Xilinx de base :

- XIo_Out16 (Xuint32 Adr, Xint16 Val) : cette fonction permet d'écrire la valeur de la variable Val (de taille 16 bits) dans le registre dont l'adresse est spécifiée par la variable Adr (l'adresse est codée sur 32 bits pour le bus OPB) ;
- (Xint16) XIo_In16 (Xuint32 Adr) : cette fonction permet de retourner la valeur du registre (de taille 16 bits ou moins) dont l'adresse est spécifiée par la variable Adr ;

Le type de données Xuint32 signifie entier non signé sur 32 bits et Xint16 est le type d'entiers signés sur 16 bits. Les deux fonctions précédentes existent pour des registres de 8 et 32 bits (XIo_Out8, XIo_In8 pour 8 bits et XIo_Out32, XIo_In32 pour 32 bits). Prenons le cas d'un IP PID dont les données manipulées par ce dernier sont représentées en virgule fixe sur 16 bits (7 bits pour la partie entière et 8 bits pour la partie factionnaire et un bit de signe). Le tableau III.6 présente les registres de l'IP ainsi que les offsets d'adresses assignés à ces derniers.

Tableau III.6. Description détaillée des registres de l'IP PID.

Registre	Offset	Taille	Accès	Description
U	0x00	32 bits	Lecture/Ecriture	[0..15] : contient la consigne. [16..31]: contient la commande.
Y	0x01	16 bits	Ecriture	Ce registre contient la mesure.
Pct1	0x02	16 bits	Ecriture	Ce registre contient la valeur du paramètre A.
Pct2	0x03	16 bits	Ecriture	Ce registre contient la valeur du paramètre B.
Ict1	0x04	16 bits	Ecriture	Ce registre contient la valeur du paramètre C.
Dct1	0x05	16 bits	Ecriture	Ce registre contient la valeur du paramètre D.
Dct2	0x06	16 bits	Ecriture	Ce registre contient la valeur du paramètre E.

Les pilotes (ou drivers) à développer sont associés à chaque opération qui concerne un registre donné. Dans notre cas, il s'agit de mettre au point un ensemble de huit pilotes. Le tableau III.7 présente une liste descriptive de ces derniers.

Tableau III.7. Tableau récapitulatif des pilotes développés.

Nom du Pilote	Registre	Opération	Description
`Set_Uc(Adr, Val)`	U	Ecriture	Permet de mettre à jour la valeur de la consigne imposée par l'utilisateur.
`Set_Y(Adr, Val)`	Y	Ecriture	Permet de mettre à jour la valeur de la mesure.
`Set_Pct1(Adr, Val)`	Pct1	Ecriture	Permet de modifier la valeur du paramètre A.
`Set_Pct2(ADR, Val)`	Pct2	Ecriture	Permet de modifier la valeur du paramètre B.
`Set_Ict1(Adr, Val)`	Ict1	Ecriture	Permet de modifier la valeur du paramètre C.
`Set_Dct1(Adr, Val)`	Dct1	Ecriture	Permet de modifier la valeur du paramètre D
`Set_Dct2(Adr, Val)`	Dct2	Ecriture	Permet de modifier la valeur du paramètre E.
`Get_U(Adr)`	U	Lecture	Permet de récupérer la valeur de la commande.

Le paramètre Adr est codé en entier non signé sur 32 bits et représente la somme de l'adresse de base de l' IP PID et de l'offset du registre à lire ou à modifier. Le paramètre Val est codé en virgule fixe sur 16 bits. Afin d'illustrer l'utilisation des deux fonctions Xilinx de base qui ont permis l'écriture des pilotes, nous donnons ci-dessous (Figure III.24) les sources en C embarqué des fonctions Set_Y et Get_U.

```
/* Lecture de la commande */
Xint16 get_U(Xuint32 Base)
{
  return(XIo_In16(Base + offset_U));
}
/* Mise à jour de la mesure */
void set_Y(Xuint32 Base,Xint16 value)
{
  XIo_Out16( Base + offset_Y,value);
}
```

Figure III.24. Utilisation des fonctions d'E/S Xilinx de base.

La variable Base représente l'adresse de base de l'IP, offset_U et offset_Y représentent respectivement les déplacements par rapport à l'adresse de base des registres U e Y et le paramètre value représente la valeur en virgule fixe sur 16 bits à écrire dans le registre Y. Avant de paramétrer notre IP PID c.à.d. calculer les valeurs des paramètres A, B, C, D et E et les écrire sur les registres correspondants, il faut d'abord les convertir en virgule fixe. Ce type

n'étant pas disponible (il n'existe pas de type virgule fixe en C/C++), il va falloir convertir ces derniers du type float (réel en virgule flottante) en virgule fixe et puis ensuite les écrire sur les registres correspondants.

Avant de procéder au test final dans un environnement réel, il faut s'assurer du bon fonctionnement des pilotes. Ceci peut être fait par une simulation du système de contrôle embarqué en utilisant l'outil ModelSim avec l'outil EDK de Xilinx. La simulation permet entre autres de déboguer plus rapidement le module PID ainsi que les pilotes softwares qui ont été développés précédemment. La figure III.25 présente la fenêtre principale de l'outil ModelSim avec les différents composants de notre système de contrôle embarqué. Les composants entourés d'un cercle rouge sont des éléments clés de notre système (Processeur, RS232, Partie de commande d'acquisition et de contrôle).

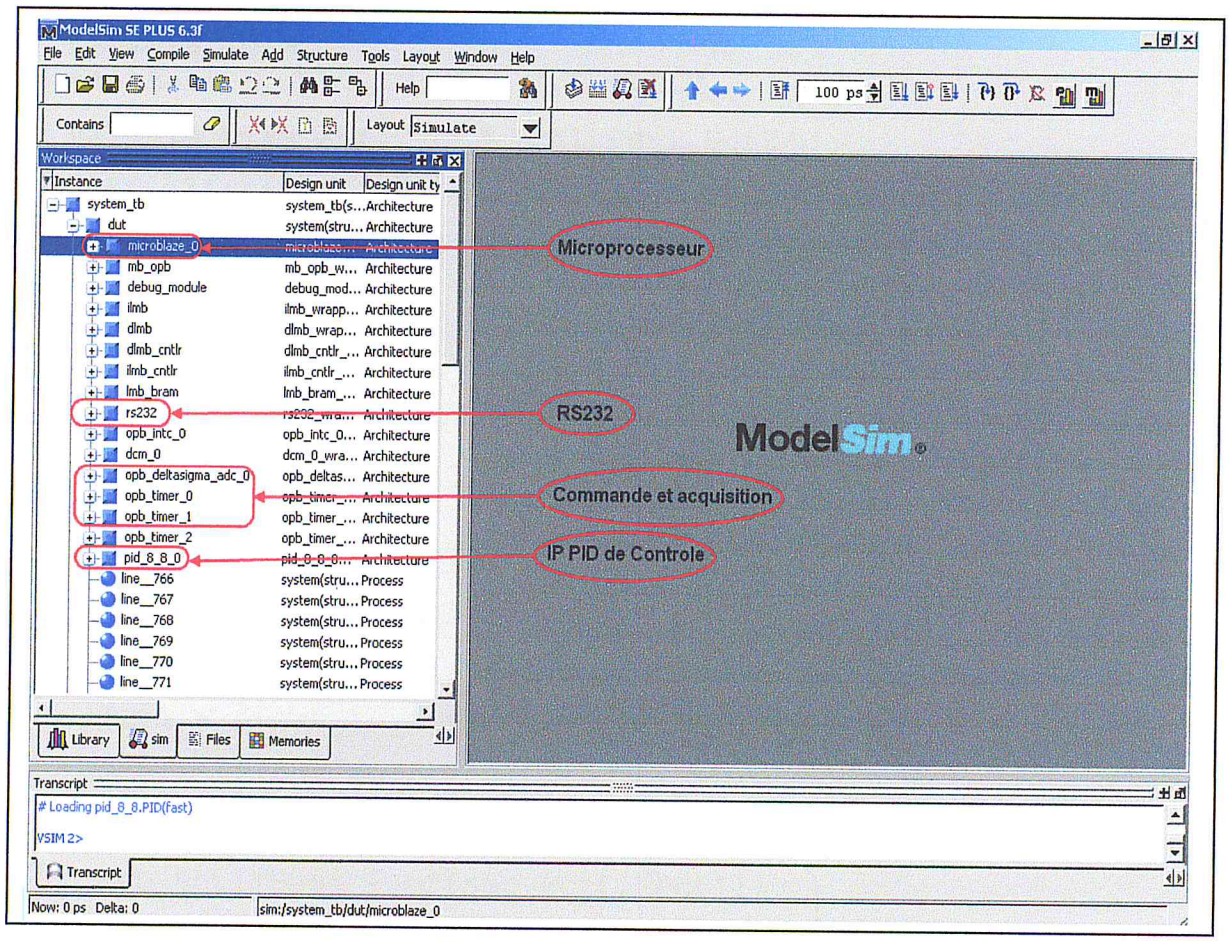


Figure III.25. Simulation du système de contrôle embarqué.

Le test consiste à écrire une simple application embarquée en C qui modifie la valeur des registres Pct1, Pct2, Ict1, Dct1 et Dct2 et de vérifier le résultat de simulation sur ModelSim. Les valeurs des registres précédents du IP PID sont en premier calculées en utilisant les équations du tableau II.5 et les paramètres K_p, T_i, T_d, T_s, N et b, ensuite ils sont convertis en virgule fixe et finalement écrit dans les registres correspondants. La figure III.26

La première partie du pseudo code permet de calculer les valeurs des paramètres A, B, C, D et E en utilisant les équations du tableau II.5. La deuxième partie convertit les valeurs obtenues des paramètres précédents en virgule flottante en des variables en virgule fixe sur 16 bits en utilisant la fonction **`float_to_fixed`**. La troisième et dernière partie consiste à écrire les valeurs converties des paramètres dans les registres correspondants en utilisant les pilotes associés à ces derniers.

Enfin, il ne reste plus qu'à simuler le comportement du système pour s'assurer du bon fonctionnement des pilotes développés. Cette simulation prend en compte non seulement le comportement du coté Hardware du système mais aussi le coté Software qui est représenté par l'application embarquée. La figure III.27 présente le résultat de simulation du système de contrôle embarqué dont l'application est celle équivalente au pseudo code de la figure II.24. On remarque bien que les pilotes testés sont fonctionnels puisque les registres ont pris les valeurs des variables Pct1, Pct2, Ict1, Dct1 et Dct2. On remarque aussi que les signaux OPB_CS et OBP_ACK sont toujours activés respectivement avant et après la fin de l'opération.

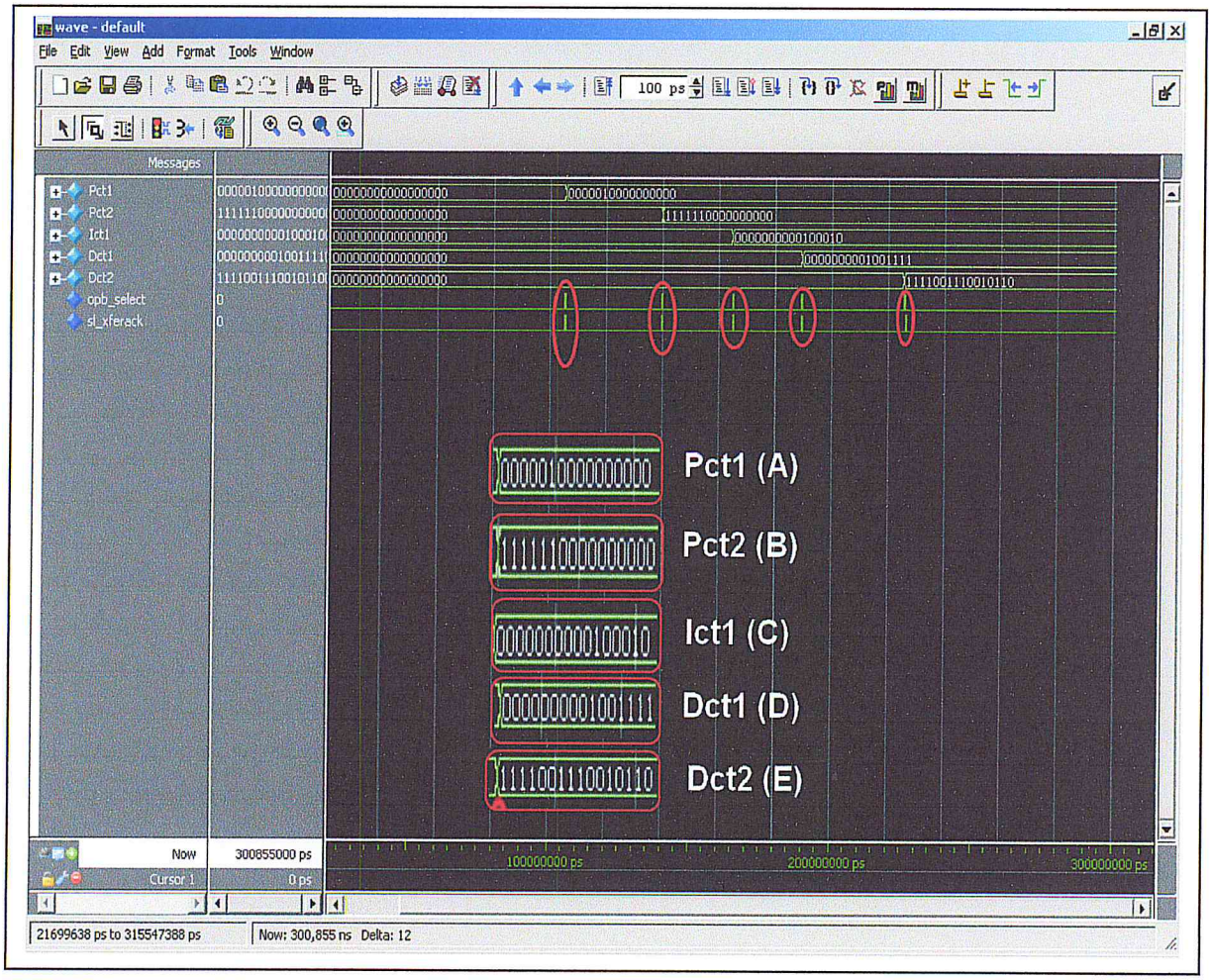


Figure III.27. Simulation Hardware/Software du système.

IV. Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'intégration des trois parties essentielles de notre système de contrôle. Nous avons vu en premier lieu comment construire un système embarqué de base constitué du processeur Microblaze relié à travers le bus OPB à un ensemble de périphériques (tels que le contrôleur d'interruptions et le contrôleur du port RS232). Ensuite, nous nous sommes penchés sur l'intégration de la partie de commande dont la tâche consiste à changer l'état du processus à contrôler. Nous avons vu que cette partie peut être intégrée soit sous forme d'un Convertisseur Digital/Analogue (en réutilisant le module OPB-DAC) ou bien sous forme d'un PWM (en réutilisant la module OPB Timer-Counter). Pour ce dernier, nous avons dû développer les pilotes nécessaires pour son exploitation à travers une application embarquée. L'étape suivante consiste à intégrer la partie d'acquisition qui fournit à la partie de contrôle l'état du processus à contrôler. Cette fonctionnalité a été réalisée par un Convertisseur Analogue/Digital (en réutilisant le module OPB CAN). Ensuite, nous avons présenté d'une manière succincte l'architecture interne de l'IP PID et son test fonctionnel avec Simulink sur Matlab. Enfin, nous nous sommes intéressés à l'intégration de l'IP PID dans un système embarqué ainsi qu'au développement des ses pilotes softwares. Nous avons aussi vu, comment procéder à un test préliminaire de simulation permettant de valider le comportement de l'IP et par la même occasion les pilotes softwares de ce dernier.

Le chapitre suivant sera consacré à la conception et à la réalisation d'un dispositif expérimental de contrôle et de régulation de température. Ce dernier nous permettra de valider le fonctionnement de notre système de contrôle embarqué dans un environnement réel.

Conception et Réalisation d'un Dispositif Expérimental

Dans ce chapitre nous exposons la conception et la réalisation d'un dispositif expérimental pour valider le fonctionnement de notre système de contrôle embarqué dans un environnement réel. Nous commençons d'abord par donner une description globale du dispositif à réaliser en utilisant notamment le langage de modélisation UML afin de mieux comprendre le fonctionnement et l'interaction entre les composants du système. Enfin, nous finirons par détailler la description donnée précédemment en mettant l'accent sur le matériel utilisé ainsi que sur le fonctionnement interne du système.

I. Description et Conception Globale

I.1. Description Globale du Dispositif

L'objectif de ce dispositif expérimental est de reproduire le comportement d'un système de contrôle et de régulation en boucle fermée. Nous avons vu précédemment que ce dernier est constitué de trois parties principales lui permettant de contrôler ou de réguler un processus donné. La partie d'acquisition est reliée à un capteur qui fournit l'état du processus à contrôler. En utilisant cette information (état du processus) et la valeur de la consigne imposée par l'utilisateur, la partie de contrôle calcule la commande et la transmet à la partie de commande qui l'applique sur le processus.

Nous venons ici de résumer le fonctionnement global de tout système de contrôle ou de régulation en boucle fermée. Notre objectif est de contrôler la température d'un dispositif expérimental afin de valider le fonctionnement de notre système de contrôle embarqué. Ce dernier dispose déjà des trois parties lui permettant d'effectuer le contrôle, il ne reste plus qu'à le relier au dispositif à contrôler. Ce dernier est constitué d'un tube qui enferme une lampe halogène et un capteur qui fournit la température de cette dernière à la partie d'acquisition de notre système. Il est constitué aussi d'un ventilateur qui permet de faire baisser rapidement la température du dispositif. La partie de commande est constituée dans notre cas de deux PWMs qui contrôlent chacun soit la lampe soit le ventilateur. Le principe de fonctionnement repose sur le contrôle de l'intensité de la lampe ou bien de la vitesse de rotation du ventilateur faisant ainsi varier la température du dispositif. Enfin, une application tournant sur un PC permet à l'utilisateur de visualiser la progression de l'état du dispositif (Température de la lampe) et de spécifier une consigne donnée ou bien de modifier la fréquence de fonctionnement des PWMs. Ceci est possible en reliant le PC à la plateforme FPGA MEMEC V2MB1000 en utilisant un câble RS232 qui véhicule les données entre ces

derniers. La figure IV.1 représente un schéma synoptique du dispositif expérimental proposé.

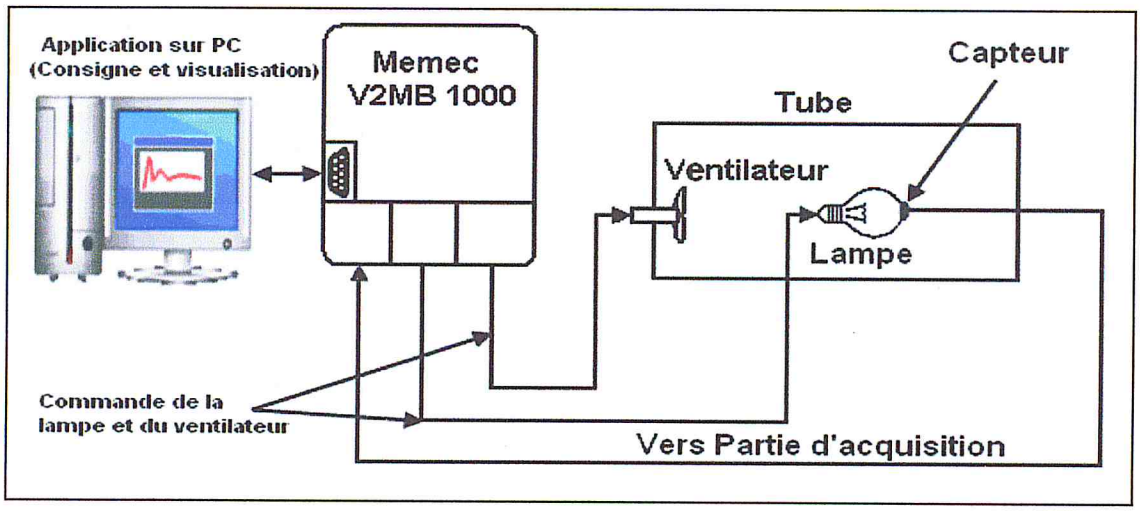


Figure IV.1. Schéma synoptique du dispositif expérimental.

I.2. Conception Préliminaire

Pour illustrer et mieux comprendre les interactions entre notre système de contrôle embarqué et les différents éléments externes, nous utiliserons les diagrammes offerts par UML (Unified Modeling Language) [32]. Nous commençons en premier lieu par établir un diagramme de cas d'utilisation afin de délimiter les fonctionnalités de notre système et ses interactions avec les différents composants du dispositif expérimental et l'utilisateur du système. D'après la description faite précédemment, l'utilisateur doit pouvoir :

- Spécifier la consigne à travers une application tournant sous un PC ;
- Visualiser la progression de l'état du dispositif sur la même application ;
- Modifier la fréquence de fonctionnement des PWMs ;

L'autre interaction concerne le transfert de données entre le capteur et notre système et plus précisément avec la partie d'acquisition qui fournit la température acquise (par le capteur) à la partie de contrôle. Enfin, le système applique la commande calculée par la partie de contrôle soit sur la lampe (augmenter la température) soit sur le ventilateur (baisser la température) en utilisant la partie de commande. D'après les informations de cette analyse on peut établir un diagramme de cas d'utilisation dont les acteurs sont :

- L'utilisateur du système ;
- L'application PC est un acteur secondaire permettant à l'utilisateur de spécifier la consigne ou bien de modifier la fréquence de fonctionnement des PWMs de la partie;

- Le capteur de température ;
- La lampe et le ventilateur.

Les cas d'utilisation peuvent aussi être résumés comme suit :

- Changer la consigne : ce cas d'utilisation est initié par l'utilisateur du système en utilisant l'application PC;
- Visualisation de la température par l'utilisateur en se servant de l'application ;
- Modifier la fréquence du PWM : ce cas est initié aussi par l'utilisateur du système en utilisant l'application PC;
- Acquisition : ce cas est initié par le capteur de température qui transmet la donnée à la partie d'acquisition;

La figure IV.2 représente le diagramme des cas d'utilisation correspondant aux interactions entre notre système de contrôle et les acteurs cités précédemment.

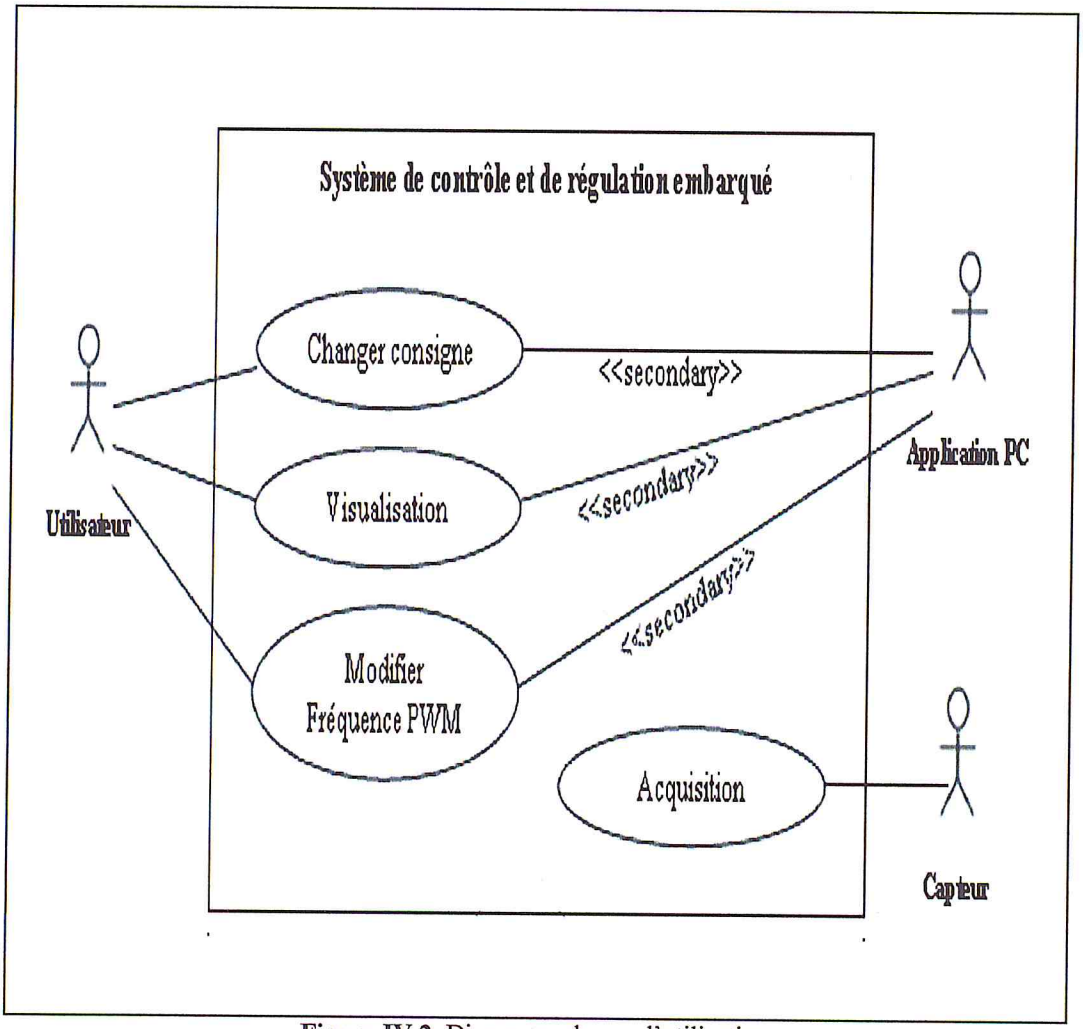


Figure IV.2. Digramme de cas d'utilisation.

Nous avons vu que le diagramme de cas d'utilisation permet de délimiter les frontières de notre système du point de vue fonctionnalités et interactions avec les éléments externes. Afin de mettre l'accent sur les composants de notre système ainsi que ceux du dispositif expérimental, nous nous proposons d'élaborer un diagramme de classes modélisant ces derniers. En tenant compte de la description faite précédemment ainsi que du diagramme de cas d'utilisation qui a été établi, on peut déjà extraire les classes et les relations suivantes:

- Le système de contrôle qui se compose de la partie de contrôle, de commande et d'acquisition ;
- Un port RS232 pour le transfert de données ;
- Une application PC permettant l'interaction entre l'utilisateur et le système de contrôle ;
- Un dispositif expérimental se composant d'une lampe, un ventilateur et un capteur ;
- La fonctionnalité de la partie de contrôle est réalisée par l'IP PID ;
- La fonctionnalité de la partie de commande est réalisée par un PWM ;
- La fonctionnalité de la partie d'acquisition est réalisée par le module OPB-CAN.
- La partie de commande du système de contrôle contient deux PWMs, chacun commandant soit la lampe soit le ventilateur.
- Le module OPB-CAN est relié au capteur qui lui transmet les données représentant la température du dispositif.

La figure IV.3 présente le diagramme de classes modélisant le système de contrôle et le dispositif expérimental et les liens qui les relient à l'utilisateur.

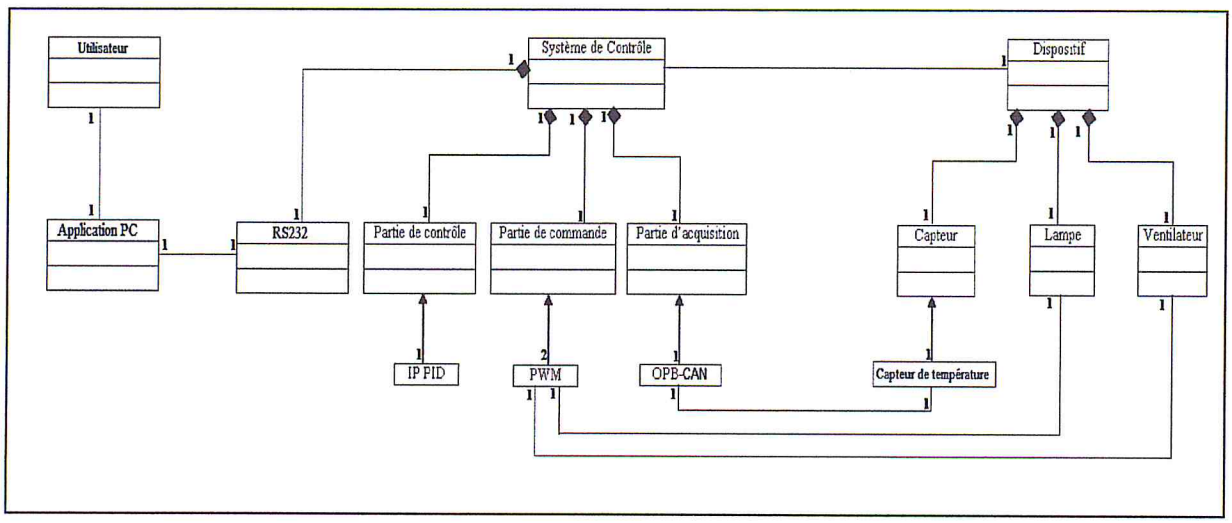


Figure IV.3. Diagramme de classes modélisant le système et le dispositif.

II. Description Détaillée du Dispositif

Dans cette section nous allons détailler un peu plus le fonctionnement de notre système de contrôle embarqué ainsi que celui du dispositif expérimental. Nous présentons aussi les composants matériels qui ont été utilisés pour réaliser ce dernier.

II.1. Capteur et Partie d'Acquisition

Nous avons vu que le module OPB-CAN permet de convertir un niveau de tension en entrée en une valeur numérique correspondante. Nous savons aussi que la plateforme FPGA utilisée ne peut lire que des niveaux de tension entre 0 volts et 3.3 volts. Nous allons utiliser le même montage que celui présenté sur la figure II.11 mais en remplaçant le synthétiseur de signaux par un capteur de température. Pour ce faire, nous avons choisi le capteur LM35 [33] qui fournit en sortie un niveau de tension qui correspond à une température donnée. Il faut noter que dans le cas de ce capteur, la relation température-tension est totalement linéaire (1.5 volts correspond à une température de 150°C et pour une tension de 0 volts on a 0°C). En tenant compte de la résolution du module OPB-CAN qui est de 10 bits (les valeurs lues seront comprises entre 0 et 1023) et de la tension maximale lue par la plateforme MEMEC V2MB1000 qui est de 3.3 volts, le calcul de la température correspondante à une valeur lue par le module OPB-CAN se fait comme suit :

$$T = ((can\_val) * 3.3) / 1023) * 100$$

Tel que can_val est la valeur lue par le module OPB-CAN à un moment donné. Il faut calculer en premier la valeur du niveau de tension qui correspondant à la valeur lue par le module OPB-CAN et ensuite multiplier le résultat obtenu par 100 afin de trouver la température correspondante. La figure IV.4 présente le schéma du montage nécessaire au fonctionnement du capteur et du module OPB-CAN.

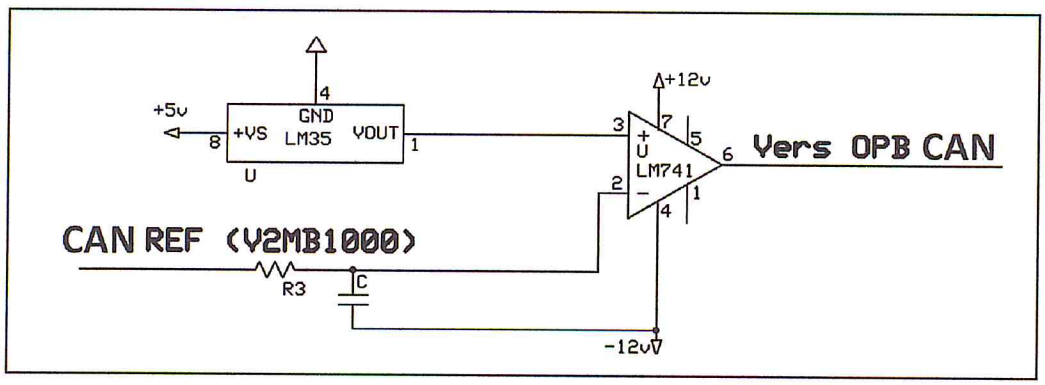


Figure IV.4. Montage nécessaire au fonctionnement du capteur et du module OPB-CAN.

II.2. Commande d'une Lampe et d'un Ventilateur

Comme nous l'avons précisé précédemment, le principe de fonctionnement du contrôle de la température du dispositif repose soit sur le contrôle de l'intensité de la lampe halogène (12 volts, 21 Watts) soit sur le contrôle de la vitesse de rotation du ventilateur (12 volts, 1.68 Watts). Ceci est réalisé par deux PWMs. Ces derniers n'agissent pas directement sur la lampe ou bien sur le ventilateur mais sur des transistors NMOS IRF540 [34] qui contrôlent la puissance consommée par ces derniers. En fonction du rapport cyclique du PWM le transistor fait passer une quantité de courant correspondante. Prenons par exemple le cas de la lampe 12 volts qui consomme en plein régime 21 Watts ce qui correspond à un courant de 1.75 A. En appliquant un signal PWM avec un rapport cyclique de 50% sur le transistor, la consommation de puissance et le courant sont réduis de moitié. On peut donc régler facilement et d'une manière précise la cadence de fonctionnement de la lampe ou bien celle du ventilateur.

Notons que la tension du signal PWM en sortie de la plateforme V2MB1000 est insuffisante (3.3 volts) pour actionner le transistor IRF 540. Ce problème peut être résolu en amplifiant la tension du signal en utilisant un amplificateur opérationnel en tant qu'amplificateur non inverseur (voir Annexe B). Pour ce faire nous avons utilisé un amplificateur opérationnel (LM741) avec deux résistances R1 et R2 qui permettent de fixer le facteur d'amplification. Deux montages d'amplification sont nécessaires : l'un pour le transistor contrôlant la lampe et l'autre pour celui qui contrôle le ventilateur (voir Figure IV.5).

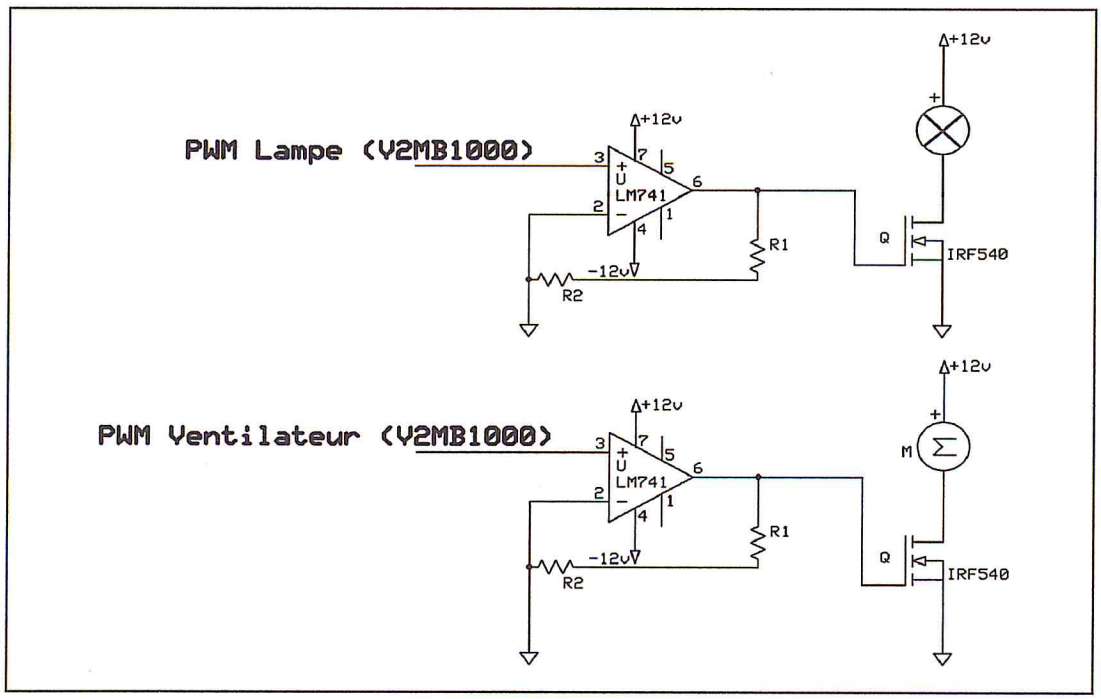


Figure IV.5. Montages de contrôle de la lampe et du ventilateur.

II.3. Fonctionnement Interne du Système

Nous avons vu que notre système de contrôle est composé de plusieurs IPs (ou bien modules) : processeur, contrôleur RS232, contrôleur d'interruptions, IP PID, etc. Ces derniers ne sont pas suffisants pour le fonctionnement du système. En effet une partie software est nécessaire pour gérer les différentes interruptions existantes telles que l'interruption **Done** de l'IP PID qui indique que la commande a été calculée. Cette commande représente en fait le rapport cyclique du signal PWM qui commande soit la lampe soit le ventilateur. Ces derniers fonctionnent en exclusion mutuelle. Si la valeur de la commande calculée par le PID est positive alors le PWM de la lampe est activé, dans le cas contraire c'est celui du ventilateur sera activé.

La commande du PID est calculée chaque T_s. Ceci peut être effectué en ajoutant un autre module OPB Timer-Counter jouant le rôle d'un compteur. Ce dernier générera une interruption dont le traitant (ou bien Handler) a pour mission de lire la valeur de l'état du processus, de l'écrire dans le registre Y de l'IP PID et de l'envoyer aussi vers l'application PC pour y être affichée. Quant au traitant de l'interruption **Done**, il permet de récupérer la commande calculée par le PID et d'activer soit le PWM commandant la lampe (commande positive) soit celui qui commande le ventilateur (commande négative).

Un autre traitant d'interruption pour la réception de données à travers le port RS232 est nécessaire. En effet, si l'utilisateur décide de modifier la consigne ou bien la fréquence de fonctionnement des PWMs, la donnée correspondante est envoyée depuis l'application PC vers la plateforme V2MB1000 en utilisant le port RS232. Puis le traitant d'interruption vérifie d'abord quel changement l'utilisateur veut effectuer et enfin on procède au changement de la consigne ou bien à la modification de la fréquence de fonctionnement des PWMs. Pour ce faire, l'application PC doit envoyer en plus de la donnée elle-même (fréquence ou consigne), le type de l'opération à effectuer (modification de la fréquence ou bien de la consigne).

Une étape d'initialisation au début est nécessaire pour calculer les paramètres de l'IP PID et les transcrire dans les registres correspondants. Cette étape compte aussi l'association de chaque traitant à son interruption correspondante, la définition des priorités des interruptions et l'activation de ces dernieres au niveau du contrôleur d'interruptions. Il faut aussi initialiser les modules de commande PWM en définissant la fréquence de fonctionnement de ces derniers, et vider la file d'attente des données en entrée des modules RS232 et OPB-CAN. La figure IV.6 présente l'organigramme du fonctionnement interne de notre

système de contrôle embarqué.

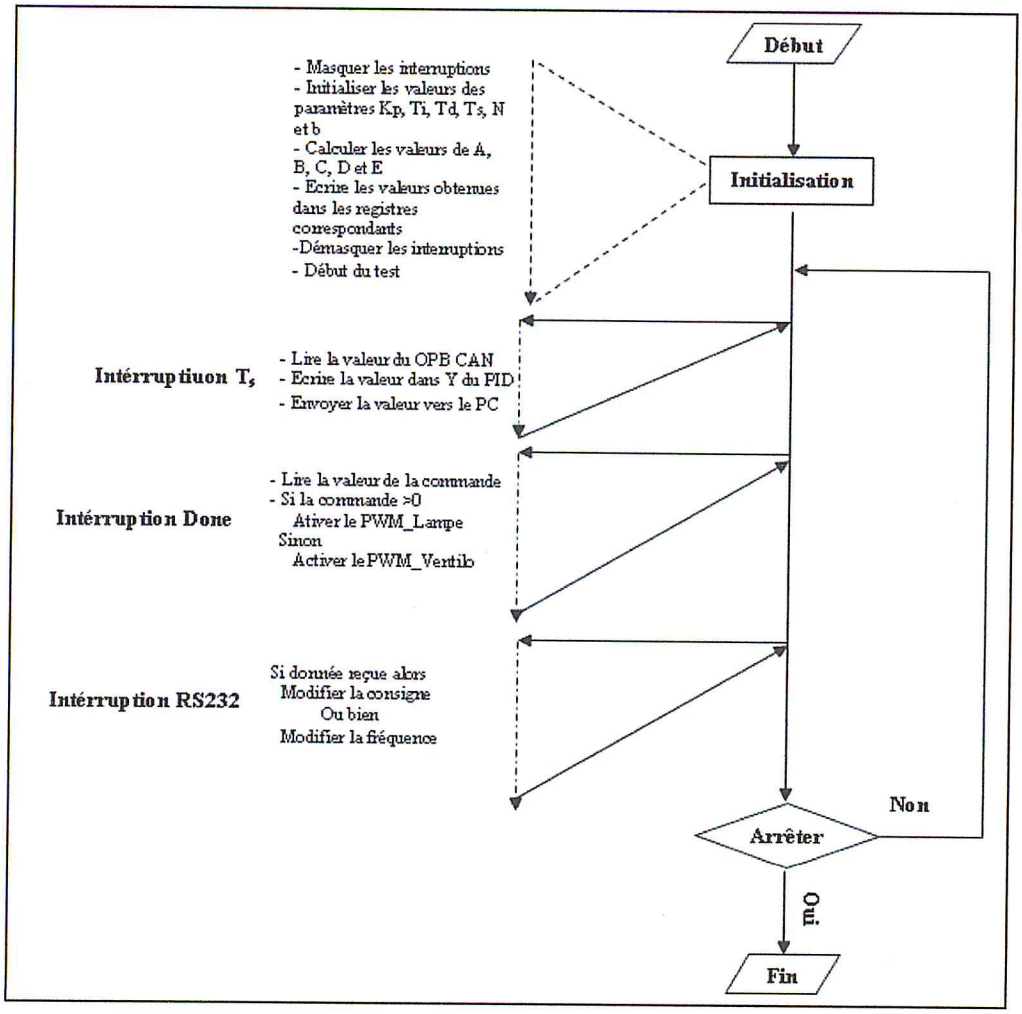


Figure IV.6. Organigramme de fonctionnement du système de contrôle.

II.4. Interaction et Visualisation des données

L'interaction entre l'utilisateur et le système de contrôle embarqué est très importante. Elle se limite dans notre cas à la spécification de la consigne ou bien à la modification de la fréquence de fonctionnement des PWMs de la partie de commande. La visualisation de la température du dispositif en temps réel est aussi un autre service qu'il faut impérativement offrir à l'utilisateur.

L'application sur le PC offre ces deux fonctionnalités à l'utilisateur. En effet, cette dernière est déployée sur un PC qui est relié à l'aide d'un câble RS232 à la plateforme FPGA MEMEC V2MB1000. Cette dernière envoie toutes les T_s la valeur de la température actuelle du dispositif. Elle affiche aussi les valeurs reçues sous forme d'une courbe qui montre l'évolution de la température du dispositif. Nous avons développé cette application en utilisant l'environnement Microsoft Visual Studio 2008. La figure IV.7 présente la fenêtre

principale de l'application développée et les fonctionnalités de base offertes à l'utilisateur, tels que :

- 1 : permet de saisir la consigne imposée par l'utilisateur ;
- 2 : permet de lancer la réception des données envoyées par la plateforme V2MB1000;
- 3 : permet d'arrêter la réception des données ;
- 4 : permet de modifier la consigne ;
- 5 : permet de quitter l'application ;
- 6 : est une zone utilisée pour dessiner en temps réel la courbe représentant l'évolution de la température en fonction du temps;
- 7 : permet de modifier la fréquence de fonctionnement du PWM (lampe ou ventilateur);
- 8 et 9 : affichent respectivement la valeur de la température actuelle et du temps écoulé.

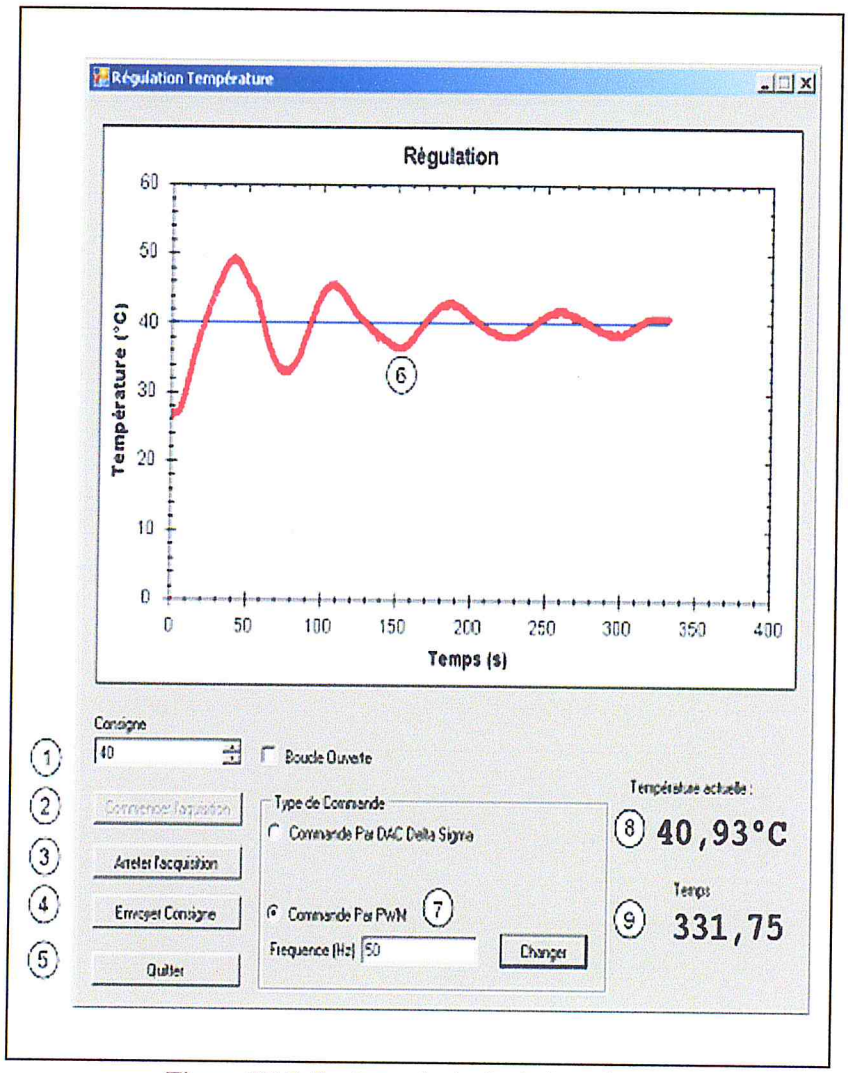


Figure IV.7. Fenêtre principale de l'application.

III. Conclusion

Dans ce chapitre nous avons présenté un dispositif expérimental qui permet de valider le fonctionnement de notre système de contrôle embarqué dans un environnement réel. Nous avons choisi comme test le contrôle et la régulation de la température. En premier lieu, nous avons établi une conception préliminaire du dispositif à développer à travers une description globale ne contenant aucun détail sur la réalisation, notamment avec l'utilisation du diagramme de cas d'utilisation. Après cela, nous avons établi une liste des différents composants à utiliser et des relations qui existent entre ces derniers.

Ensuite, nous nous sommes intéressés aux détails de réalisation du dispositif tels que le choix et le fonctionnement du capteur, la commande des composants modifiant la température du dispositif (Lampe et ventilateur) ainsi que le mode de fonctionnement interne de l'application embaquée gérant les différentes interruptions et modules (RS232, OPB-CAN, PWM, IP PID,...) de notre système.

L'interaction avec l'utilisateur et la visualisation des résultats sont des fonctionnalités primordiales pour l'utilisation et l'exploitation de notre système. Pour ce faire nous avons développé une application tournant sous un PC relié à la plateforme FGPA à travers le port RS232 facilitant la modification de la consigne et de la fréquence de fonctionnement des PWMs et permettant l'affichage de l'évolution de la température de notre dispositif en temps réel.

Tests et Résultats Expérimentaux

Dans ce chapitre nous exposons l'ensemble des tests que nous avons adoptés pour valider le fonctionnement de notre système de contrôle embarqué dans un environnement réel. Ce dernier est constitué d'un dispositif expérimental qui est relié aux parties de commande et d'acquisition de notre système de contrôle et de régulation embarqués. Ces tests ont pour objectif en premier lieu d'examiner l'impact de la taille de la partie fractionnaire sur la précision et la qualité de la régulation. Nous allons aussi tester la réaction de notre système de contrôle face aux perturbations externes et sa capacité à poursuivre une consigne avec amplitude et durée variables dans le temps.

I. Test de Fonctionnalité

La figure V.1 présente une photo prise du montage réalisé en laboratoire, tels que :

- 1 : représente la plateforme FPGA MEMEC V2MB1000 ;
- 2 : est le montage contenant les deux amplificateurs de tension pour les PWMs de la lampe et du ventilateur ainsi que le comparateur utilisé par le module OPB-CAN ;
- 3 : représente le dispositif à contrôler. Il est constitué d'un tube contenant un ventilateur, une lampe halogène et un capteur de température (LM35).
- 4 : est un PC relié à la plateforme FPGA par un câble RS232. Ce PC contient l'application utilisée pour interagir avec l'utilisateur (Modification de la consigne et de la fréquence des PWMs) et afficher l'état du dispositif en temps réel.

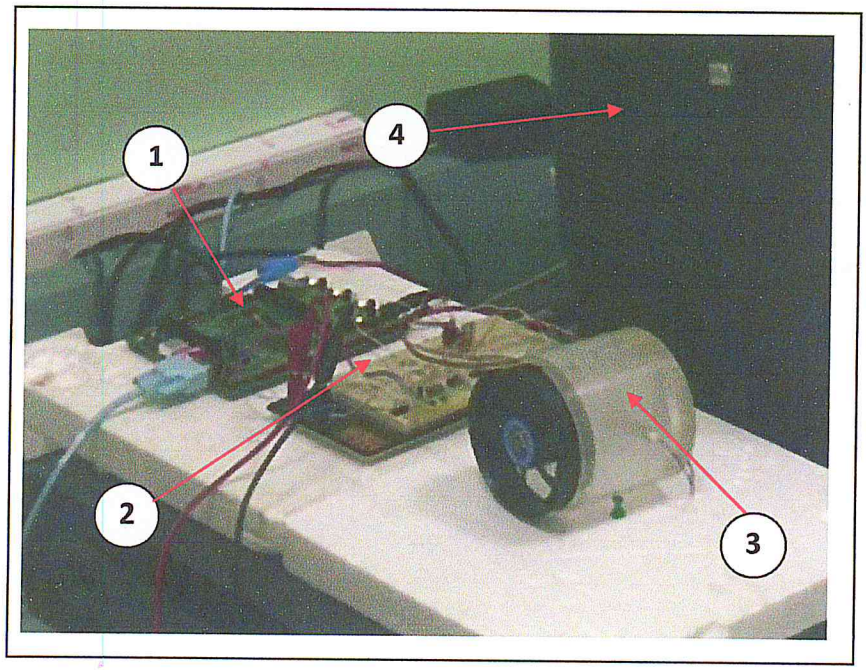


Figure V.1. Photo du montage expérimental réalisé.

L'alimentation du montage électronique ainsi que celle du dispositif expérimental (Ventilateur et Lampe) est assurée par une alimentation stabilisée (12V, -12V, 2.5A). Le capteur de température LM35 quant à lui est alimenté par une tension de 5V.

Dans un premier lieu, nous allons imposer une consigne à notre système et nous verrons si ce dernier la satisfait sans prendre en compte la précision et la rapidité de la régulation. Pour ce test nous avons utilisé un PID_8_8 (8 bits pour la partie fractionnaire et 7 bits pour la partie entière et un bit de signe) ainsi que les paramètres suivants : K_p=14.705, T_d=3.125, T_i=12.5 et T_s=0.05. Ces valeurs ont été déterminées d'une manière empirique en procédant à plusieurs tests. Cependant, il existe plusieurs méthodes (Annexe A) qui permettent de déterminer ces paramètres et qui exigent un certain bagage en théorie du contrôle (Méthodes d'identification). La figure V.2 (prise à partir de l'application) présente le résultat obtenu de la régulation de température du dispositif expérimental avec les paramètres précédents.

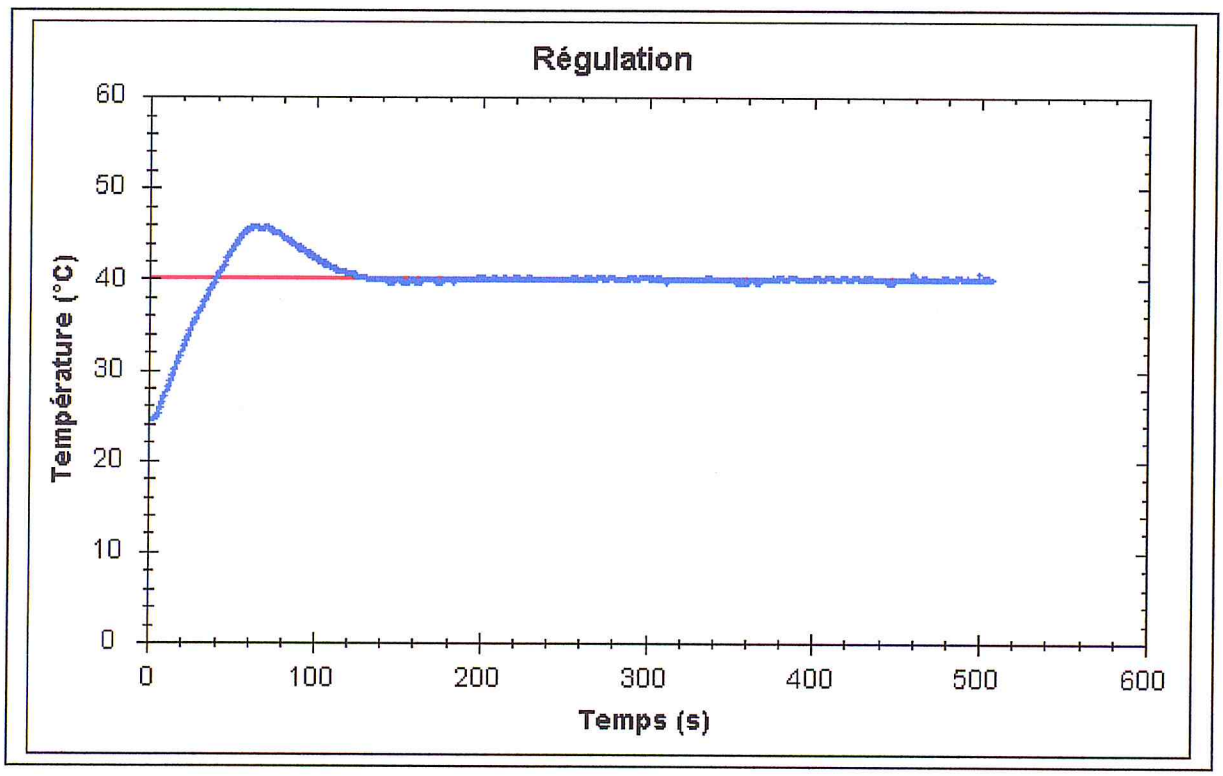


Figure V.2. Régulation de la température avec un PID_8_8.

La courbe rouge représente la consigne que nous avons imposée à notre système et la courbe bleue représente l'évolution de la température du dispositif en fonction du temps. Nous remarquons que la température du dispositif dépasse dans un premier temps la consigne avant de se stabiliser autour de cette dernière avec une petite erreur statique de l'ordre de ±0.3°C. La vitesse de la régulation et le dépassement dépendent des paramètres K_p, T_i et T_d.

II. Impact de la Précision

Ce test vise à montrer l'impact de la précision sur la qualité de la régulation en augmentant ou bien en diminuant la taille de la partie fractionnaire des données manipulées par l'IP PID. Afin d'avoir un point de comparaison ou bien de référence, nous avons développé une application embarquée jouant le rôle d'un PID. Cette dernière a été implémentée en utilisant les mêmes équations qui ont servi à développer l'IP PID. Contrairement à ce dernier, cette application manipule des données réelles avec une taille de 32 bits (Type Float en C). Nous avons aussi utilisé plusieurs versions de notre IP PID qui se diffèrent par la taille de la partie fractionnaire. La figure V.3 présente le résultat du test effectué.

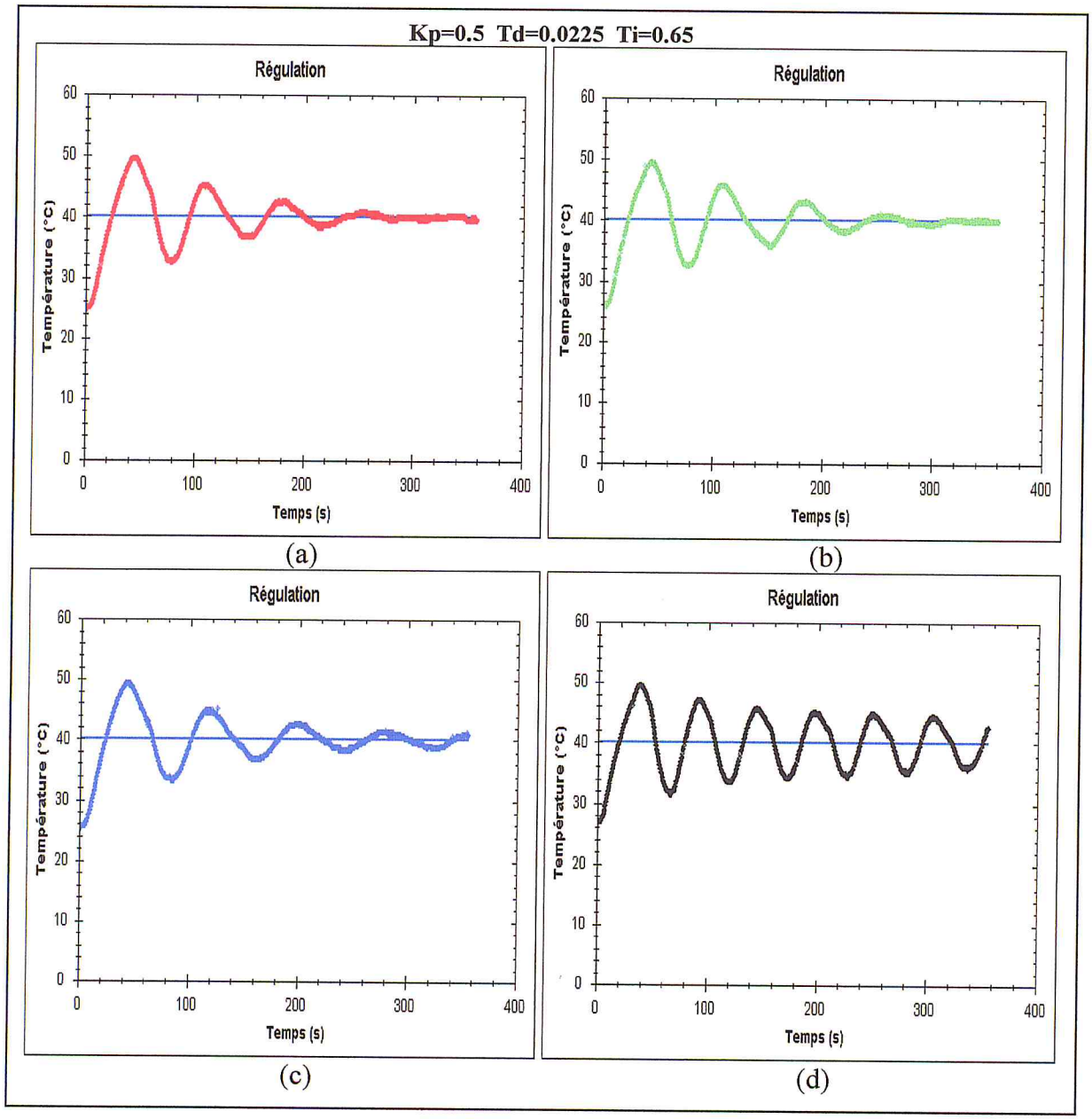


Figure V.3. Impact de la précision sur la qualité de la régulation (a) avec PID software (b) avec un PID_8_8 (c) avec un PID_8_6 (d) avec un PID_8_4.

On remarque sur la figure V.3 la dégradation de la qualité de régulation au fur et à mesure que la taille de la partie fractionnaire est réduite. Alors que le PID software et l'IP PID_8_8 se sont stabilisés sur la consigne, le PID_8_6 ne l'est pas encore et le PID_8_4 est rentré dans une phase d'oscillation autour de cette dernière. On arrive donc à obtenir presque la même qualité de régulation que celle du PID software en utilisant un PID_8_8. Il faut noter que l'utilisation des nombres réels à virgule flottante dans une application embarquée induit non seulement une utilisation supplémentaire des ressources système mais aussi une augmentation de la consommation de la puissance.

III. Test de Stabilité

Ce test consiste à vérifier la capacité de notre système à reprendre la régulation après avoir été perturbé par un phénomène externe. Nous allons laisser le système fonctionner jusqu'à ce qu'il se stabilise sur la consigne, ensuite nous exercerons une perturbation en soufflant de l'air froid sur la lampe, et enfin nous observerons le comportement du système face à cette situation. La figure V.4 montre le résultat du test de stabilité.

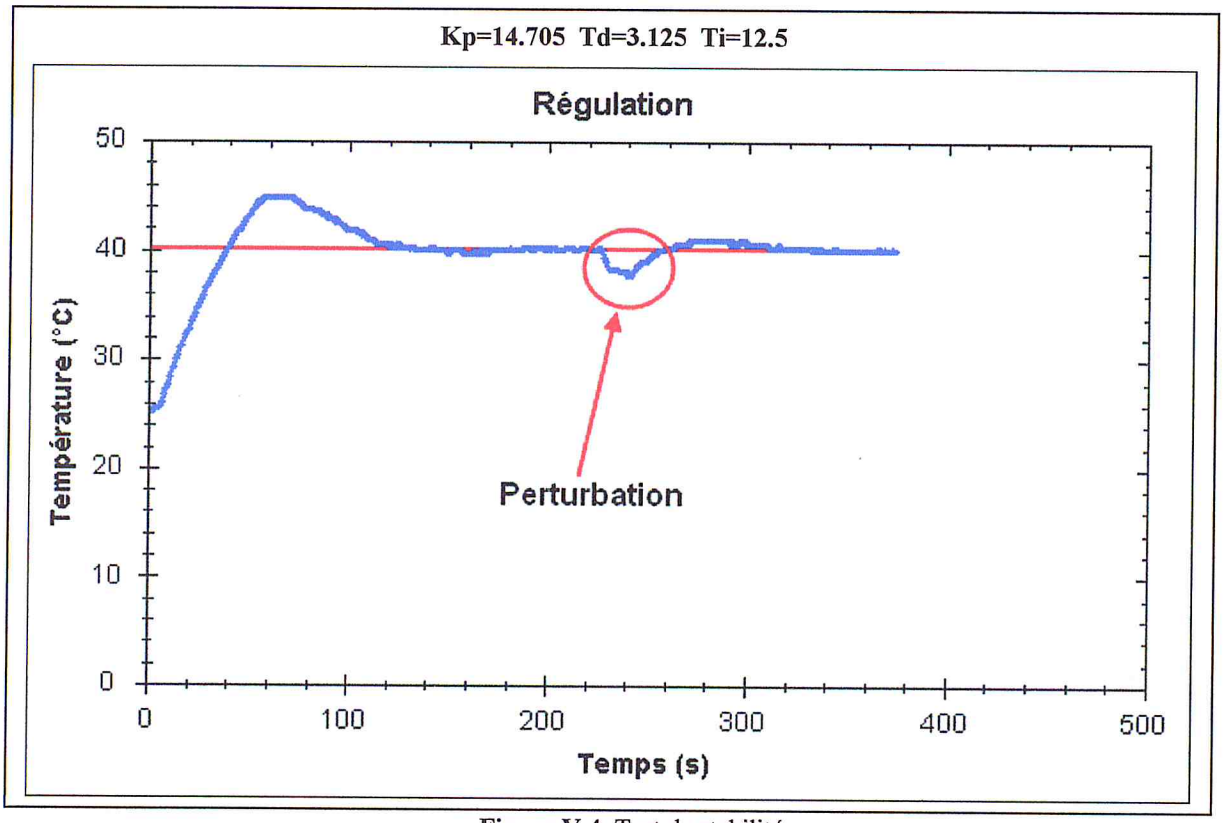


Figure V.4. Test de stabilité.

On remarque bien que notre système reprend la régulation du dispositif et se stabilise sur la consigne après un certain temps.

IV. Test de Poursuite

L'objectif de ce test est de vérifier la capacité de notre système à poursuivre la régulation de la température du dispositif avec une consigne variable dans le temps (Durée et amplitude). Pour ce faire, nous avons procédé au changement de la consigne après chaque fois que la température du dispositif soit stabilisée. La figure V.5 présente le résultat de ce test.

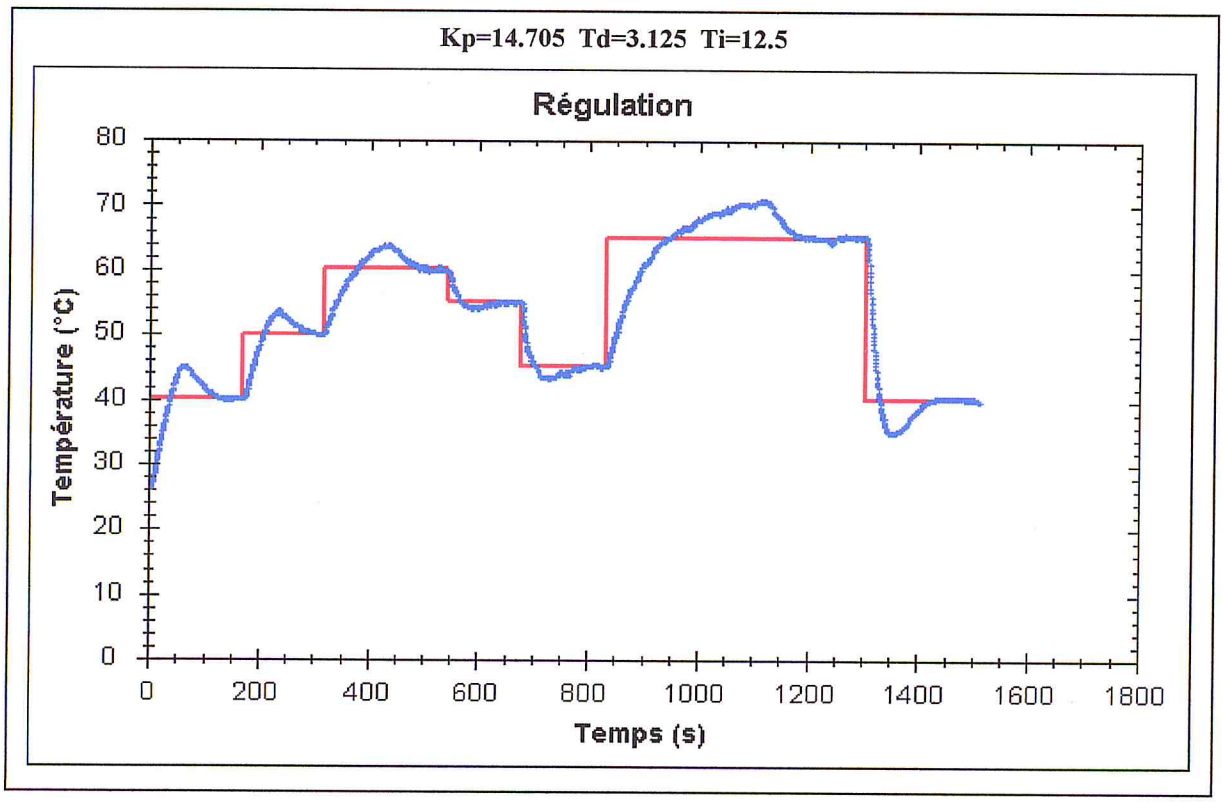


Figure V.5. Test de poursuite de consigne avec amplitude et durée variables dans le temps.

On remarque d'après la figure V.5, que notre système arrive à chaque fois à se stabiliser sur la consigne après la modification de cette dernière.

IV. Conclusion

Dans ce Chapitre nous avons présenté les différents tests qui ont été menés pour valider le fonctionnement de notre système de contrôle embarqué dans un environnement réel. En premier lieu, nous avons procédé à un test simple visant à vérifier que le système est bien fonctionnel et qu'il effectue correctement la régulation demandée. Dans le deuxième test nous avons observé l'impact de la précision sur la qualité de la régulation. Ensuite, nous avons procédé à un test de stabilité afin de vérifier la capacité à résister aux perturbations externes. Enfin, un test de poursuite avec une consigne variable dans le temps a été effectué.

Conclusion et Perspectives

Dans ce Mémoire, nous avons présenté la mise au point d'une plateforme FPGA HW/SW pour le contrôle et la régulation embarqués de processus physiques. Après une brève introduction générale, nous avons exposé l'état de l'art des circuits FPGA et leurs domaines d'application ainsi que l'avantage de leur utilisation par rapport à d'autres solutions déjà existantes sur le marché. Nous avons pris comme cas particulier, les FPGAs et le flot d'outils de Xilinx. Ensuite, nous nous sommes intéressés aux systèmes embarqués en général et à ceux implémentés sur FPGA en particulier. Pour ce faire nous avons établi une étude succincte sur les processeurs Softcores existants sur le marché et nous nous sommes intéressés plus particulièrement au processeur Microblaze et les systèmes embarqués basés sur ce dernier. Après cela, dans le deuxième chapitre nous avons introduit la notion de contrôle et de régulation de processus physiques ainsi que le régulateur PID. Nous avons terminé ce deuxième chapitre par exposer la problématique du projet et la solution proposée.

Dans le troisième chapitre, nous avons abordé les différentes étapes suivies pour intégrer les parties de notre système de contrôle embarqué. Nous avons tout d'abord réalisé un système embarqué basique que nous avons testé à travers une application vérifiant le bon fonctionnement de ses composants. Dans l'étape suivante nous avons procédé à l'intégration de la partie de commande qui permet de changer l'état du processus à contrôler. Nous avons vu que la fonctionnalité de cette partie peut être réalisée par un convertisseur numérique/analogique (en intégrant le module OPB-DAC) ou bien par un PWM (en intégrant le module OPB Timer-Counter). Notre choix s'est porté sur ce dernier car il permet un contrôle précis du fonctionnement voulu. L'intégration du module OPB Timer-Counter a nécessité le développement d'un ensemble de pilotes softwares pour l'exploitation de ce dernier à partir d'une application embarquée. Ensuite, nous sommes passés à l'intégration et au test de la partie d'acquisition qui fournit l'état du processus à contrôler. La fonctionnalité de cette partie est réalisée par le module OPB-CAN qui nécessite un certain accompagnement électronique externe à la plateforme FPGA. Enfin, l'intégration de la partie de contrôle est la plus importante. Cette partie a été implémentée sous forme d'un IP PID dont la représentation interne des données est en virgule fixe. Cette représentation optimise les ressources utilisées et la consommation de puissance sur FPGA. Nous avons aussi développé une interface OPB et un ensemble de pilotes softwares pour notre IP PID.

Dans le quatrième chapitre, nous avons présenté la conception et la réalisation d'un dispositif expérimental pour valider le fonctionnement de notre système dans un environnement réel. Nous avons choisi le contrôle et la régulation de température d'un dispositif constitué d'un

tube contenant un ventilateur et une lampe halogène munie d'un capteur. Pour mieux comprendre le fonctionnement de notre système et de ses composants ainsi que les liens qui les relient, nous avons utilisé le diagramme de cas d'utilisation et le diagramme de classes d'UML. Nous avons aussi présenté une application qui permet à l'utilisateur d'interagir avec le système de contrôle embarqué et de visualiser l'évolution de la température du dispositif en temps réel.

Enfin, dans le dernier chapitre nous avons procédé à plusieurs tests visant à valider le fonctionnement de notre système. Ces tests sont similaires à ceux effectués sur Simulink. En premier, nous avons testé la capacité du système à satisfaire une consigne donnée. Ensuite, nous avons observé à l'aide du deuxième test l'impact de la taille de la partie factionnaire (précision) sur la qualité de la régulation. Nous avons aussi observé la stabilité du système de contrôle face aux perturbations externes. Dans le dernier test, nous avons mis à l'épreuve la capacité de notre système à suivre une consigne variable (Durée et amplitude) dans le temps (test de poursuite).

Comme perspective, nous visons à utiliser le système conçu pour contrôler des processus plus rapide que la régulation de la température tels que contrôle de la vitesse ou bien du déplacement. Ceci peut être accompli en utilisant notre système pour contrôler des dispositifs rapides tels que des bras mécaniques robotisés ou bien la vitesse de déplacement d'un véhicule ou encore la manipulation de micro-pinces. Ces dernières font l'objet de la prochaine étape de notre projet de recherche (Le projet de Master s'intègre dans le cadre du protocole de recherche 2011-2013, CDTA).

Annexes

Annexe A

Régulateur PID

Définition

Un régulateur PID (pour « **P**roportionnel **I**ntégral **D**érivé ») est un dispositif de contrôle permettant d'effectuer une régulation ou un contrôle en boucle fermée d'un système industriel. C'est le régulateur le plus utilisé dans l'industrie, et il permet de contrôler un grand nombre de procédés.

Principe général

L'erreur observée est la différence entre la consigne et la mesure. Le PID permet trois actions en fonction de cette erreur :

- Une action **Proportionnelle** : l'erreur est multipliée par un gain P ;
- Une action **Intégrale** : l'erreur est intégrée sur un intervalle de temps *s*, puis divisée par un gain *Ti ;*
- Une action **Dérivé** : l'erreur est dérivée suivant un temps *s*, puis multipliée par un gain *Td.*

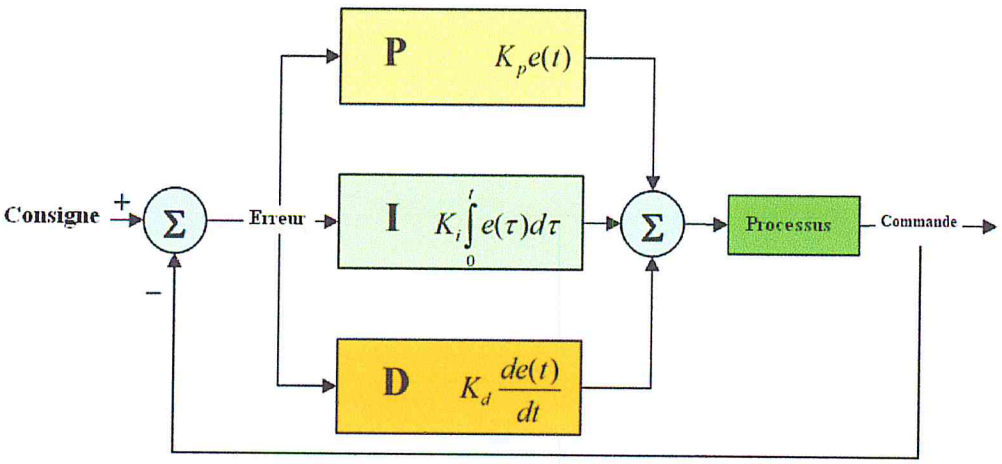


Schéma synoptique d'un régulateur PID

Les différents paramètres à trouver sont P, Td et Ti pour réguler le procédé ayant pour fonction de transfert. Il existe de nombreuses méthodes pour trouver ces paramètres. Cette détermination de paramètres est communément appelée synthèse. La procédure expérimentale pour calculer ces paramètres s'appelle aussi procédure d'identification paramétrique.

Les paramètres du PID influencent la réponse du système de la manière suivante :

- **G** : Lorsque *G* augmente, le temps de montée (*rise time*) est plus court mais il y a un dépassement plus important. Le temps d'établissement varie peu et l'erreur statique se trouve améliorée.

- **Ti** : Lorsque $\frac{1}{Ti}$ augmente, le temps de montée est plus court mais il y a un dépassement plus important. Le temps d'établissement au régime stationnaire s'allonge mais dans ce cas on assure une erreur statique nulle. Donc plus ce paramètre est élevé, plus la réponse du système est ralentie.

- **Td** : Lorsque *Td* augmente, le temps de montée change peu mais le dépassement diminue. Le temps d'établissement au régime stationnaire est meilleur. Pas d'influences sur l'erreur statique. Si ce paramètre est trop élevé dans un premier temps il stabilise le système en le ralentissant trop mais dans un deuxième temps le régulateur anticipe trop et un système à temps mort élevé devient rapidement instable.

Pour ces trois paramètres, le réglage au-delà d'un seuil trop élevé a pour effet d'engendrer une oscillation du système de plus en plus importante menant à l'instabilité.

L'analyse du système avec un PID est très simple mais sa conception peut être délicate, voire difficile, car il n'existe pas de méthode unique pour résoudre ce problème. Il faut trouver des compromis, le régulateur idéal n'existe pas. En général on se fixe un cahier des charges à respecter sur la robustesse, le dépassement et le temps d'établissement du régime stationnaire. Les méthodes de réglage les plus utilisées en théorie sont la méthode de Ziegler-Nichols, la méthode de P. Naslin (polynômes normaux à amortissement réglable), la méthode du lieu de Nyquist inverse (utilise le diagramme de Nyquist).

Dans la pratique, les professionnels utilisent soit l'identification par modèle de Broïda pour les systèmes stables ou le modèle intégrateur retardé pour les systèmes instables soit la méthode par approches successives, qui répond à une procédure rigoureuse : on règle d'abord l'action P seule pour avoir un dépassement de 10 à 15% puis l'action dérivé de façon à "raboter" au mieux le dépassement précédent, enfin on ajuste si nécessaire l'action intégrale en se fixant un dépassement final compris entre 5 et 10%. Dans environ 15% des cas les performances d'un PID peuvent devenir insuffisantes en raison de la présence d'un retard trop important dans le

modèle du procédé. On fait alors appel à d'autres algorithmes de réglage (notamment : régulateur PIR ou à modèle interne ou à retour d'état).

Pseudo-Code

Voici une simple boucle d'un programme qui implémente l'algorithme de régulation d'un PID:

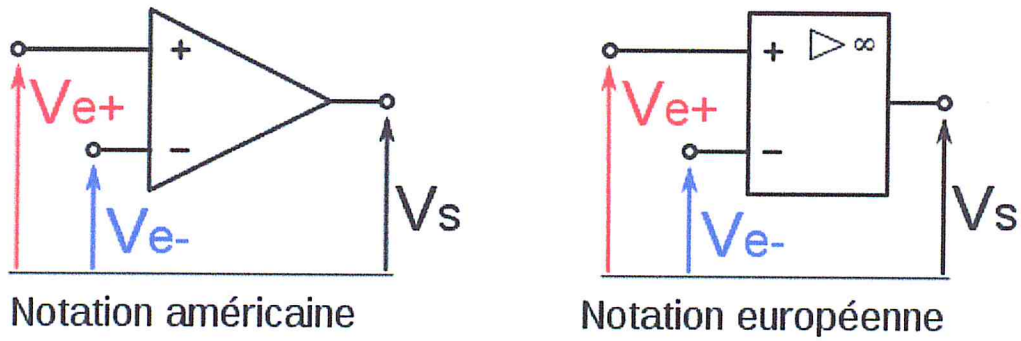
```
Erreur_prec = consigne - etat_acvtuel
integrale = 0
Début:
  erreur = consigne - etat_actuel
  integrale = integrale + (erreur*Ts)
  derivee = (erreur - error_prec)/Ts
  commande = (Kp*erreur) + (Ki*integrale) + (Kd*derivative)
  erreur_prec = erreur
  Attendre(Ts)
  goto Début
```

Annexe B

Montage d'un Amplificateur de tension

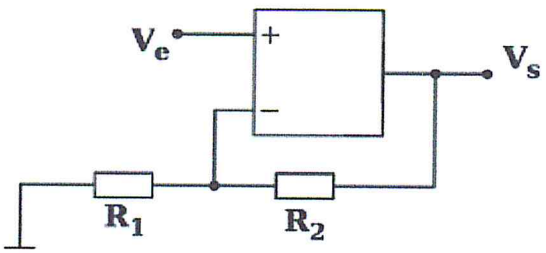
Définition

Un amplificateur opérationnel (aussi dénommé ampli-op ou ampli op, AO, AOP, ALI ou AIL) est un amplificateur différentiel : c'est un amplificateur électronique qui amplifie une différence de potentiel électrique présente à ses entrées. Il a été initialement conçu pour effectuer des opérations mathématiques dans les calculateurs analogiques : il permettait de modéliser les opérations mathématiques de base comme l'addition, la soustraction, l'intégration, la dérivation et d'autres. Par la suite, l'amplificateur opérationnel est utilisé dans bien d'autres applications comme la commande de moteurs.



Symbole électrique d'un amplificateur opérationnel

Montage en Amplificateur de tension



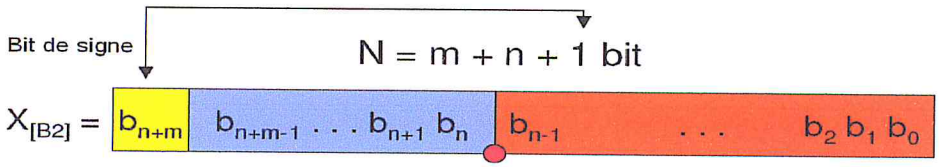
Montage d'un amplificateur non-inverseur

Ce Montage permet d'amplifier un niveau de tension en entrée (V_e) suivant la formule suivante :

$$Vs = Ve \times (1+(R1/R2))$$

Annexe C

Conversion Réel en virgule flottante → Réel en virgule fixe



Représentation d'un réel en virgule fixe.

Un nombre en virgule fixe est constitué d'une partie entière (m bits), d'une partie fractionnaire (n bits) et d un bit de signe.

La représentation en virgule fixe est le complément à deux de la représentation en virgule flottante multipliée par 2^N.

$$X_{[Base\ 10]} \times 2^n = \left[-2^{N-1} \times b_{N-1} + \sum_{i=0}^{N-2} 2^i b_i \right]$$

Représentation en virgule fixe

Exemple :

$9.6 \rightarrow 9.6 \times 2^4 = 153.6 = (\text{arrondie})\ 154 = (0 \mid 1001 \mid 1010)_{[Base\ 2]}$

$-12.4 \rightarrow 12.4 \times 2^4 = -198.4 = (\text{arrondie})\ -198 = (1 \mid 0011 \mid 1010)_{[Base\ 2]}$

Fonction de conversion en C :

```
1  :int float_to_fixed(float real_float,float pow,float pow2)
2  :{
3  :  float partie_fractionnaire;
4  :  int partie_entiere;

5  :  real_float=real_float*pow;
6  :  partie_entiere=(int)real_float;
7  :  partie_fractionnaire = real_float - (int)partie_entiere;

8  :  if(real_float>0)
9  :  {
10 :   if(partie_fractionnaire >0.5)
11 :    partie_entiere ++;
12 :  }
13 :  else
14 :  {
15 :   if(partie_fractionnaire <-0.5)
16 :     partie_entiere --;
17 :   partie_entiere =pow2 + partie_entiere;
18 :  }
19 : return(partie_entiere);
20 :}
```

- Les variables pow1 et pow2 représentent respectivement 2^n et 2^N ;
- La $5^{ème}$ ligne multiplie le nombre flottant par 2^n ;
- La $6^{ème}$ et $7^{ème}$ ligne sépare la partie fractionnaire de la partie entière ;
- Les lignes (10,11) et (15,16) permettent d'arrondir la partie entière ;
- La ligne 17 permet de calculer le complément à deux de la partie entière dans le cas ou le nombre flottant serait négatif.

Références Bibliographiques

[1] Wikipedia FPGA, http://en.wikipedia.org/wiki/FPGA

[2] Arnaud Lagger, "Self Reconfiguration plateform for cryptographic application", Master theisis, 2006, Swiss federal institute of technology lausanne.

[3] Xilinx, "Virtex-II complete datasheet", *Product Specification*, no. 31, Mars 2005. Available from: http://www.xilinx.com/partinfo/ds031.pdf

[4] Xilinx, "Virtex-II™ V2MB1000 Development Board User's Guide". www.cs.lth.se/EDA385/HT06/doc/restricted/V2MB_User_Guide_3_0.pdf

[5] Xilinx, "EDK Concepts, Tools, and Techniques". http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/edk_ctt.pdf

[6] A.K. Oudjida, M. Liem, D. Benamrouche, A. Liacha, M. Goudjil & R. Tiar, "Front-End IP-Development: Basic Know-How", Journal *"Technologies Avancées, CDTA"*, No. 20, Juillet 2008, Algérie.

[7] Ludovic Loiseau, "Methodology Design Reuse," *Miranda Technologie Inc*, 2001, Canada.

[8] Mentor Graphics, "ModelSim SE User's Manual". http://www.model.com/resources/default.asp?sendto=/support/documentation/se/pdf_6.3j/modelsim_se_user.pdf

[9] Xilinx, "MicroBlaze Processor Reference Guide". http://www.xilinx.com/support/documentation/sw_manuals/edk91i_mb_ref_guide.pdf

[10] D. Mattsson and M. Christensson "Evaluation of synthesizable CPU cores", Chalmers, University Of Technology, Décembre 2004.

[11] K. Åström, T. Hägglund, "PID Controllers: Theory, Design, and Tuning," by the Instrument Society of America, Research Triangle Park, NC, USA, 2nd Edition, ISBN: 1-55617-516-7, Copyright 1995.

[12] M.L. Berrandjia, A.K. Oudjida, A. Liacha, R. Tiar, "Développement d'Applications Embarquées sous uClinux : Exemple Pratique sur Memec V2mb1000 ", *2end Proceedings of the International Conference on Systems & Processing Information ICSIP'2011*, p. 84, May 15-17, 2011, Guelma, Algérie.

[13] IBM Corp, "On-Chip Peripheral Bus, Architecture Specifications", Version 2.1, Copyright, Avril 2001.

[14] Netrino, " Introduction to Pulse Width Modulation (PWM)". http://www.netrino.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation.

[15] Xilinx, "Xilinx Device Drivers Documentation". http://www.cs.washington.edu/lab/facilities/hwlab/pdfs/xilinxdoc/xilinx_drivers.pdf

[16] Tektronix, "Tektronix TDS3054 technical specifications". http://www.gsamart.com/sp.cfm/DIGOSC/TEK/TDS3054.html

[17] Xilinx, "OPB Timer/Counter". http://www.xilinx.com/support/documentation/ip_documentation/opb_timer.pdf.

[18] Xilinx, "OPB Delta-Sigma Analog To Digital Converter (ADC)". http://www.xilinx.com/support/documentation/ip.../opb_deltasigma_adc.pdf.

[19] Keithley, "Model 3940 Multifunction Synthesizer Service Manual Rev. A". http://www.keithley.fr/data?asset=966

[20] Texas instruments, "µA741, µA741Y GENERAL-PURPOSE OPERATIONAL AMPLIFIERS". http://www.datasheetcatalog.org/datasheet/texasinstruments/ua741.pdf.

[21] A.K. Oudjida, M. Liem, D. Benamrouche, A. Liacha, M. Goudjil & R. Tiar, "Front-End IP-Development: Basic Know-How", Journal "*Technologies Avancées, CDTA*", No. 20, Juillet 2008, Algérie.

[22] R.Tiar, A.Liacha, A.K.Oudjida & M.L. Berrandjia, "Procédure Automatique de Vérification D'IPs", *International Conference on Systems and Information Processing*, p 31, 2-4 Mai, 2009, Guelma, Algérie.

[23] J. Lazaro et al, "Simulink/Modelsim Simulable VHDL PID Core for Industrial SoPC Multiaxis Controllers", *Proceedings of the IEEE 32nd Annual Conference on Industrial Electronics (IECON)*, pp. 3007-3011, 2006.

[24] Y.H. Seo, and D.W. Kim, "A New VLSI Architecture of Parallel Multiplirer-Accumulator Based on Radix-2 Modified Booth Algorithm", *IEEE Trans. on VLSI Systems*, vol. 18, N° 2, Février, 2010.

[25] A.K.Oudjida, N.Chaillet, A.Liacha, M.Hamerlain, M.L.Berrandjia, "High Speed and Low-Power PID Structures for Embedded Applications", Proceedings of the *21th edition of the International Workshop on Power and Timing Modeling, Optimization and Simulation PATMOS*, pp. 257-266, 26-29 Septembre, 2011, Madrid, Espagne.

[26] A. D. Booth, "A Signed Binary Multiplication Te:chnique," *Quarterly J. Mech. Appl. Math.*, Vol. 4, part 2, pp. 236-240,1951.

[27] O.L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proceedings of the IRE*, Vol. 49(1), pp. 67-91, Janvier 1961.

[28] Bernd Girod, Rudolf Rabenstein, Alexander Stenger, "Signals and systems", 2nd ed., Wiley, 2001, ISBN 0471988006, p. 50

[29] Xilinx, "Designing Custom OPB Slave Peripherals for MicroBlaze". http://www.xilinx.com/ipcenter/processor_central/microblaze/doc/opb_tutorial.pdf

[30] M.L. Berrandjia, A.K. Oudjida, A. Liacha, R. Tiar, "Intégration de l'IP SPI-Slave dans l'Environnement SoC Xilinx-EDK et Test Physique avec la Carte d'Evaluation Memec V2MB1000", *Proceedings of the International Conference on Applied Informatics ICAI'09*, pp. 87-92, ISBN: 978-9947-0-2763-9, Novembre 15-17, 2008, Bordj Bou Arréridj, Algérie.

[31] D. MENARD, "Méthodologies de conversion automatique en virgule fixe pour les applications de traitement du signal". http://hal.inria.fr/tel-00609159/PDF/menard-4.pdf.

[32] Grady Booch, James Rumbaugh, Ivar Jacobson, "Le guide de l'utilisateur UML", Editions Eyrolles, 2000, ISBN 2-212-09103-6.

[33] National Semiconductor, "LM35, Precision Centigrade temperature Sensors". http://www.national.com/ds/LM/LM35.pdf.

[30] Fairchaild Semiconductors, "IRF540N MOSFET". http://www.datasheetcatalog.org/datasheet/fairchild/IRF540N.pdf.