

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M. Ing.

PAR
Simon NOIROT

COMMANDE PAR FPGA :
DE LA MODÉLISATION À L'IMPLÉMENTATION

MONTRÉAL, LE 6 JUIN 2012

©Tous droits réservés, Simon Noirot, 2012

©Tous droits réservés

Cette licence signifie qu'il est interdit de reproduire, d'enregistrer ou de diffuser en tout ou en partie, le présent document. Le lecteur qui désire imprimer ou conserver sur un autre media une partie importante de ce document, doit obligatoirement en demander l'autorisation à l'auteur.

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Christian Belleau, directeur de mémoire
Département de génie mécanique à l'École de technologie supérieure

M. Jean-François Boland, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Maarouf Saad, membre du jury
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 14 MAI 2012

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à remercier M. Christian Belleau d'avoir accepté de me diriger dans le cadre mon mémoire. Ses conseils avisés et sa disponibilité m'ont permis d'apprendre énormément. Il a su me guider tout en me laissant libre dans mes choix et m'a fourni des ressources essentielles qui m'ont permis de m'épanouir dans mon projet.

Je tiens à saluer le département de génie électrique de l'INSA de Lyon qui a rendu possible mon cursus en double diplôme au sein de l'ETS. Je remercie également l'ÉTS pour la qualité de la formation que j'y ai reçue et pour m'avoir accordé une bourse interne.

Je remercie aussi tout ceux qui ont partagé ma vie au cours de ces derniers mois. D'abord, mon colocataire de toujours, Cyril, avec qui j'ai vécu cette aventure. Il reste, je m'en vais, ainsi va la vie. Une pensée également à Romain et Nicolas pour tous les bons moments passés ensemble. Ensuite, Rémi, avec qui nous avons fait des parties de foot endiablées et avons dévoré des tonnes de bagels !

Je remercie ma mère, que j'aime, et qui m'a toujours soutenu et encouragé au cours de mes études.

Enfin, je souhaite remercier Elise pour son soutien et son amour et, avec qui nous avons pris le pari un peu fou de vivre notre histoire malgré le temps et la distance.

COMMANDE PAR FPGA : DE LA MODÉLISATION À L'IMPLEMENTATION

Simon NOIROT

RÉSUMÉ

Dans un monde où s'accroît la complexité des systèmes électromécaniques, des applications de plus en plus performantes sont exigées. La commande de tels systèmes doit donc pouvoir répondre à ces attentes. L'évolution des technologies numériques permet aujourd'hui de disposer de composants efficaces, reconfigurables. Le développement au cours de ces dernières années d'outils logiciels intégrés à haut niveau d'abstraction permet à un nombre croissant de concepteurs d'utiliser des circuits numériques de pointe. Les FPGA profitent de ces évolutions et sont des candidats de choix pour la réalisation de modules de commande performants.

Après avoir défini un système d'étude et en avoir donné sa modélisation mathématique, il est possible de passer à la phase de développement. En bénéficiant de l'outil HDL Coder intégré au sein de Matlab/Simulink, il est possible d'obtenir du code HDL à partir de schémas de modélisation de haut niveau. Dans un premier temps, un régulateur de type PID est proposé. Ensuite une commande plus élaborée est définie. La commande moderne, par ses calculs matriciels importants, permet de bénéficier de la puissance de calcul des FPGA. Enfin, une démarche de conception à base de modèles est proposée.

La méthodologie de conception à base de modèles couplées à la mise en place d'une bibliothèque de composants réutilisables permet de disposer de modèles à la fois simulable et synthétisable dans un FPGA. L'implémentation d'un PID, en utilisant l'arithmétique distribuée, permet d'obtenir des résultats intéressants. L'implantation d'un PID au sein d'un FPGA peut se réaliser rapidement et aisément comme peuvent le proposer d'autres méthodes, tel xPC Target. La mise en place de l'arithmétique distribuée au sein d'une commande dans l'espace d'état afin d'économiser les ressources du FPGA à disposition a montré ses limites. En effet, l'accumulation d'erreurs pourtant faibles ne permet pas de disposer d'un système stable. La raison est due, notamment, à un système d'étude peut propice au calcul numérique à point fixe. Il a également été montré que dans le cadre de la commande dans l'espace d'état, la résolution des convertisseurs jouait un rôle primordial. Une résolution trop faible produisant une erreur statique sur l'estimation des états provoquant la divergence de la boucle de commande. Enfin il a été possible de mettre en œuvre une bibliothèque de composants réutilisables et flexibles qui, utilisée dans une conception à base de modèles, permet de réduire le temps de mise sur le marché de façon significative.

Mots-clés : FPGA, Matlab/Simulink, HDL Coder, PID, commande moderne, conception à base de modèles, bibliothèque, commande

COMMANDE PAR FPGA : DE LA MODÉLISATION À L'IMPLÉMENTATION

Simon NOIROT

ABSTRACT

In a world where electro-mechanical systems are more and more complex has come an increasingly need of performances for this kind of applications. The evolution of digital technologies gives a set of components that can meet these requirements. Over the last years the development of electronic computer-aided design softwares has made FPGAs good candidates to enhanced the performances of controllers.

Once the model is described mathematically, the development process can start. It is possible to use an high level graphical model to generates HDL code by using the Matlab/Simulink Toolbox HDL Coder. First a classical PID controller is developped. Then a modern control technique is designed. FPGAs'computational ability is used to perform the required matrix multiplications. We also propose the model-based design as an other modeling approach.

The model-based design methodology as well as the development of a reusable and flexible components' library give models that can be both simulated and synthetized. A PID controller with distributed arithmetic gives great performances and presents a time to market equivalent to xPC Target for instance. The use of distributed arithmetic in the state space context has shown its limits. One reason for that result is that the system use is not « computational-friendly ». Also it has been shown that the resolution of converters has a great impact on the system stability and more particularly on the states estimation. Finally a library has been developped that enables the end-user to focus on the system level design. The massive use of flexible and reusable components associated with model-based design reduces significantly the time to market.

Keywords : FPGAs, Matlab/Simulink, HDL Coder, PID, modern control, model-based design, library, control

TABLE DES MATIÈRES

	Page
INTRODUCTION	1
CHAPITRE 1 REVUE DE LITTÉRATURE	9
1.1 Field Programmable Gate Array (FPGA).....	9
1.2 Notions fondamentales	10
1.2.1 Historique.....	10
1.2.2 Concepts.....	12
1.2.3 Constitution interne.....	14
1.2.4 Démarche de conception	17
1.3 Conception de systèmes à base de FPGA.....	18
1.3.1 Comment utiliser les FPGA ? Quels sont les outils à notre disposition ?	19
1.3.2 Domaine d'utilisation.....	21
CHAPITRE 2 CONCEPTION DES SYSTEMES DE COMMANDE	25
2.1 Notions fondamentales	25
2.1.1 La transformée de Laplace.....	25
2.1.2 La transformée en Z.....	27
2.1.3 Modélisation des systèmes physiques, fonction de transfert	28
2.1.4 Représentation des fonctions de transfert	30
2.2 Dynamique et placement des pôles	32
2.3 Contrôleur Proportionnel-Intégral-Dérivé (PID).....	38
2.3.1 Descriptif du PID	38
2.3.2 Discrétisation du contrôleur PID.....	39
2.4 Commande dans l'espace d'état	45
2.4.1 Représentation de systèmes dans l'espace d'état.....	45
2.4.2 Commande dans l'espace d'état.....	50
2.4.2.1 Observateur d'état.....	51
2.4.2.2 Régime établi	54
CHAPITRE 3 TRAITEMENT NUMÉRIQUE	57
3.1 La gestion numérique des nombres	57
3.2 Le traitement des opérations numériques	60
3.2.1 Opérations et fonctionnement	60
3.2.2 Introduction à l'arithmétique distribuée.....	63
3.2.3 Implémentation et taille mémoire	64
3.2.4 Quantification du gain de l'AD.....	65
CHAPITRE 4 DESCRIPTION DU SYSTEME.....	67
4.1 Le processus à commander (Banc d'essai).....	68
4.2 Dynamique interne à la carte d'évaluation.....	70
4.2.1 Description du système de commande.....	73

4.2.2	Convertisseurs A/N, N/A et conditionnement du signal	74
4.2.3	Transfert total	77
4.2.4	Contraintes et limitations dynamiques	78
4.3	Modélisation du système dans l'espace d'état.....	81
CHAPITRE 5 IMPLÉMENTATION DU MODULE PID.....		85
5.1	L'AD appliquée au régulateur PID filtré	85
5.2	Mise en place des blocs fonctionnels.....	87
5.2.1	L'interface Simulink	88
5.2.2	Implémentation du système de commande type PID	91
5.2.2.1	Action proportionnelle	94
5.2.2.2	Action intégrale.....	96
5.2.2.3	Action dérivée.....	100
5.3	Simulation des actions du PID.....	101
5.3.1	Simulation de l'action proportionnelle.....	101
5.3.2	Simulation de l'action intégrale	102
5.3.3	Simulation de l'action dérivée.....	103
5.4	Simulation du système en boucle fermée	105
5.5	Résultats expérimentaux.....	107
CHAPITRE 6 IMPLÉMENTATION DE LA COMMANDE DANS L'ESPACE D'ÉTAT		113
6.1	Simulation du modèle simplifié.....	113
6.2	Simulation du système à commande discrète	117
6.2.1	Discrétisation du procédé.....	117
6.2.2	Système de commande discret	118
6.2.3	Système de commande synthétisable	121
6.2.4	Système de commande à AD	126
6.2.4.1	Construction de l'observateur par AD	127
6.2.4.2	Tests en boucle fermée.....	132
6.3	Résumé	134
CHAPITRE 7 VERS UNE CONCEPTION À BASE DE MODÈLES.....		135
7.1	Mise en œuvre du MBD	135
7.1.1	MBD pour la commande par PID	135
7.1.2	MBD pour la commande dans l'espace d'état	137
7.2	Mise en place de la bibliothèque de composants.....	141
7.2.1	Modules de gestion des interfaces.....	142
7.2.2	Les organes de commande	142
7.3	Résumé	143
CHAPITRE 8 DISCUSSION		145
8.1	Commande PID par FPGA	145
8.2	Commande dans l'espace d'état par FPGA	147
CONCLUSION.....		151

ANNEXE I	UTILISATION DES OUTILS LOGICIELS	153
ANNEXE II	COMPLÉMENTS SUR L'ESPACE D'ÉTAT	159
ANNEXE III	CIRCUIT D'INSTRUMENTATION	165
ANNEXE IV	GESTION DES TYPES NUMÉRIQUES (POINT FIXE)	171
ANNEXE V	PRÉSENTATION DE LA BIBLIOTHÈQUE	173
BIBLIOGRAPHIE	177

LISTE DES TABLEAUX

	Page
Tableau 3.1	Addition avec retenue anticipée61
Tableau 5.1	Description des étapes de l'AD.....95
Tableau 5.2	Exemple de calcul décimal par AD99
Tableau 5.3	Comparaison des grandeurs caractéristiques de la réponse entre PID continu et à AD..... 106
Tableau 5.4	Résultats de la synthèse du PID implémenté avec AD issus du logiciel ISE107
Tableau 5.5	Utilisation des ressources du FPGA pour l'implémentation des modules d'entrées/sorties issus du logiciel ISE.....108
Tableau 5.6	Comparaison des grandeurs caractéristiques109
Tableau 5.7	Comparaison des caractéristiques entre implémentation via xPC Target et FPGA.....110
Tableau 6.1	Temps caractéristiques de la commande continue 116
Tableau 6.2	Temps caractéristiques du système à commande discrète.....121
Tableau 6.3	Comparaison des temps caractéristiques entre deux précisions de commande synthétisable.....125
Tableau 6.4	Résultats de synthèse pour la commande synthétisable avec précision (43,35) issus du logiciel ISE.....125
Tableau 6.5	Utilisation des ressources du FPGA pour différentes implémentations.....132

LISTE DES FIGURES

		Page
Figure 1.1	Principe d'un tableau logique (ou <i>Logic Array</i>)	12
Figure 1.2	Structure d'un bloc logique configurable (CLB).....	14
Figure 1.3	Détail d'une tranche	15
Figure 1.4	Points d'Interconnexions Programmables	17
Figure 2.1	Schéma d'une boucle de commande à retour unitaire	30
Figure 2.2	Position des pôles dans le plan de Laplace.....	34
Figure 2.3	Réponse impulsionnelle en fonction de la position des pôles	35
Figure 2.4	Positions des pôles dans le plan des Z	38
Figure 2.5	Principe d'intégration par les différences avant.....	41
Figure 2.6	Représentation d'état d'un système (A,B,C)	46
Figure 2.7	Représentation d'état avec retour d'état	50
Figure 2.8	Système de commande dans l'espace d'état muni d'un observateur	52
Figure 2.9	Construction d'un observateur	54
Figure 3.1	Représentation numérique à point fixe sur N bits	57
Figure 3.2	Implémentation d'une gestion CLA dans un FPGA de type Spartan-3.....	61
Figure 3.3	Temps et fréquence équivalente des calculs effectués via la gestion CLA	63
Figure 3.4	Utilisation des LUT en fonction de la taille de la multiplication	66
Figure 4.1	Boucle de commande globale.....	67
Figure 4.2	Traitement de la consigne.....	68
Figure 4.3	Banc d'essai	69

Figure 4.4	Représentation de la formation de l'erreur numérique et analogique	72
Figure 4.5	Représentation physique et abstraite du système	73
Figure 5.1	Implémentation d'une fonction basique sous Simulink	88
Figure 5.2	Choix du type d'un signal	90
Figure 5.3	Schéma Simulink du régulateur PID complet destiné à être synthétisé	91
Figure 5.4	Schéma Simulink du cœur du régulateur PID	93
Figure 5.5	Schéma Simulink de l'Action proportionnelle avec AD (« K_P »).....	96
Figure 5.6	Principe du registre PISO	96
Figure 5.7	Schéma Simulink de l'action intégrale avec AD (« K_I »)	97
Figure 5.8	Principe du registre SISO	97
Figure 5.9	Schéma Simulink de l'action dérivée avec AD (« K_D_filtr »).....	100
Figure 5.10	Schéma de simulation.....	101
Figure 5.11	Schéma de simulation pour le bloc intégral.....	102
Figure 5.12	Courbes de simulation pour l'action intégrale	103
Figure 5.13	Schéma de Simulation blocs "dérivé"	104
Figure 5.14	Résultats de simulation pour l'action dérivée.....	104
Figure 5.15	Schéma Simulink du système muni d'un PID à AD	105
Figure 5.16	Résultats de simulation pour deux types PID modélisés (continu et AD).....	106
Figure 5.17	Réponse simulée et expérimentale du système.....	108
Figure 5.18	Réponse du système pour deux types d'implémentation de PID (xPC et FPGA).....	111
Figure 6.1	Schéma de simulation de la commande par retour et observateurs d'état.....	113

Figure 6.2	Description de l'observateur continu	114
Figure 6.3	Estimation des états pour le modèle continu	115
Figure 6.4	Tension de sortie et de commande du système modélisé en continu	116
Figure 6.5	Schéma Simulink du système à commande discrète	119
Figure 6.6	Implémentation de l'observateur discret.....	120
Figure 6.7	Tension de sortie et de commande du système à commande discrète.....	120
Figure 6.8	Schéma Simulink du système pour une commande synthétisable	122
Figure 6.9	Schéma Simulink de l'observateur d'état synthétisable	123
Figure 6.10	Retour d'état synthétisable.....	123
Figure 6.11	Comparaisons des tensions de sortie et de commande du système à commande synthétisable pour deux précisions	124
Figure 6.12	Schéma Simulink de l'observateur à AD	127
Figure 6.13	Structure de l'observateur à AD.....	128
Figure 6.14	Détails de l'AD sur une ligne.....	129
Figure 6.15	Erreur d'estimation de l'observateur à AD.....	130
Figure 6.16	Erreur d'estimation pour l'observateur (58,50).....	131
Figure 6.17	Evolution de la différence de la commande par retour état u_k	132
Figure 6.18	Réponse du système pour retour d'état et observateur de type (58,50)	133
Figure 7.1	Schéma Simulink du système complet à commande PID	136
Figure 7.2	Schéma d'implémentation de la commande dans l'espace d'état	137
Figure 7.3	Erreur d'estimation de l'observateur en fonction du modèle du CNA	139
Figure 7.4	Réponse du système pour un retour d'état sur états réels et états estimés avec CNA réel	140

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

Abréviations

SysGen	System Generator for DSP
μC	Microcontrôleur
μP	Microprocesseur

Acronymes

ABEL	Advanced Boolean Expression Language
AD	Arithmétique distribuée
ASIC	Application-Specific Integrated Circuits
BTCAM	Bit-True Cycle-Accurate Models
CA	Circuit d'adaptation
CAD	Computer Aided Design
CAN	Convertisseur Analogique/Numérique
CI	Circuit intégré
CLA	Carry-Lookahead Adder
CLB	Configurable Logic Blocks
CNA	Convertisseur Numérique/Analogique
CPLD	Complex Programmable Logic Devices
CPU	Central Processing Unit
DSP	Digital Signal Processors
ED	Equation différentielle
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programmable Read-Only Memory

XXII

EMF	Embedded Matlab Function
EPROM	Erasable Programmable Read-Only Memory
FPGA	Field Programmable Gate Array
HDL	Hardware Description Languages
ISE	Integrated Synthesis Environnement
LC	Logic Cell
LUT	Look-Up Table
LTI	Linear Time-Invariant
MBD	Model-Based Design
MOSFET	Metal-Oxyde Semiconductor Field Effect Transistors
OTP	One Time Programmable
PAR	Place And Route
PID	Proportionnel-Intégral-Dérivé
PIP	Points d'interconnexions programmables
PISO	Parallel-In Serial-Out
PLD	Programmable Logic Devices
PROM	Programmable Read Only Memory
RAM	Random Access Memory
ROM	Read-Only Memory
RTL	Register-Transfert Level
SISO	Serial-In Serial-Out
SoC	System-on-a-Chip
SRAM	Static Random Access Memory

TTM	Time To Market
VHDL	Very High Speed Integrated Circuits Hardware Description Languages
VHSIC	Very High Speed Integrated Circuits
VLSI	Very Large Scale Integration

INTRODUCTION

En 1947, les laboratoires Bell présentent un nouveau composant qui initie alors le développement de l'Électronique tout au long du XXème siècle, le transistor (Maxfield, 2004). Les années 1960 voient, quant à elles, l'émergence des semi-conducteurs et des circuits intégrés qui inaugurent une nouvelle discipline : l'électronique numérique¹ (Allard, 2009). Une dizaine d'années plus tard, le premier microprocesseur (μ P) fait son apparition² (Intel, 2012). Puis, les techniques d'industrialisation se sont améliorées rendant possibles des gravures toujours plus fines, permettant de disposer d'une puissance de calcul toujours grandissante (Burghartz et al., 2006). A ce titre, la conjecture de Moore (1965), quant au doublement du nombre de transistors³ sur une même puce tous les dix-huit mois s'est révélée exacte jusqu'au début du XXIème siècle. Ainsi le développement des moyens électroniques a permis de disposer de systèmes à très grande échelle d'intégration (Jullien et Bayoumi, 1991). On évoque alors la technologie *Very Large Scale Integration* (VLSI). Parallèlement, se sont développés des systèmes électromécaniques de plus en plus complexes dont leur commande a dû répondre à des attentes de plus en plus exigeantes (Monmasson, 2007). C'est ainsi qu'on a pu constater une explosion de la demande des systèmes embarqués toujours plus performants, et ce, dans tous les domaines comme le montrent Paiz et Pormann (2007) ou Barlas et Moallem (2008).

Un des représentants de la technologie VLSI est le *Field Programmable Gate Array* (FPGA) qui est apparu dans les années 1980 mais dont l'utilisation n'a véritablement débuté qu'à partir des années 1990 (Maxfield, 2004). Un FPGA est un circuit intégré disposant d'un nombre important de blocs logiques configurables reliés entre eux par des interconnexions programmables. Ils permettent la réalisation de fonctions complexes (Xilinx, 2012a) en bénéficiant d'une architecture permettant la parallélisation des calculs, alliée à la flexibilité propre aux circuits numériques programmables (Rahul, Pramod et Vasantha, 2007),

¹ On parle aussi d'électronique digitale.

² Intel présente le 4004 en 1971.

³ Moore parlait initialement de composants, ne précisant pas le terme transistor

(Mingyao, Wuhua et Xiangning, 2010). Ces composants sont venus occuper l'espace vacant entre d'un côté les circuits ultra-performants, les *Application-Specific Integrated Circuits* (ASIC), et de l'autre, les *Programmable Logic Devices* (PLD). Les ASIC sont capables de réaliser des fonctions extrêmement complexes mais la mise en place se révèle coûteuse (dans le temps et économiquement). Les PLD qui, s'ils ont l'avantage de bénéficier d'un faible coût de mise en œuvre dû à leur simplicité, ne peuvent être utilisés pour des applications complexes (Maxfield, 2004). Toutefois, les FPGA n'auraient pas pu connaître leur essor actuel sans le développement d'outils d'aide à la conception directement emprunté aux ASIC (Monmasson, 2007). En effet, l'implémentation des FPGA se fait au moyen de langages de description matérielle, ou en anglais, *Hardware Description Languages* (HDL). Il y a deux principaux langages HDL : le Verilog et le *Very High Speed Integrated Circuits* (VHSIC) HDL, communément appelé VHDL. En outre, ces langages, comme leur nom l'indique, décrivent le fonctionnement au niveau électronique. Dès lors, ces langages apparaissent comme l'antonymie complémentaire à la réalisation de fonctions complexes, qui nécessitent, de fait, un certain niveau d'abstraction. On voit alors clairement apparaître un besoin croissant d'outils d'intégration efficaces gommant les différences entre niveaux de modélisation fonctionnelle et description matérielle (Barlas et Moallem, 2008; Monmasson, 2007). Ces outils permettant la description de haut niveau ont subi d'importantes recherches à partir de la fin des années 1990. Même si comme le montre Maxfield (2004), les recherches ont toujours poussé dans le sens d'une augmentation du niveau d'abstraction. Ce fut, d'ailleurs, un axe de recherche très tôt comme l'indique Brown (1996). Shanblatt (2005) regrettait toujours le manque d'un outil intégré complet permettant la conception, la simulation et l'implémentation d'un système au sein d'un FPGA.

La firme Xilinx propose dès le début des années 2000, (Ownby et Mahmoud, 2003), un module complémentaire à Matlab/Simulink, *System Generator for DSP* (SysGen), permettant de faire le lien entre le modèle du système et le système physique (Lazaro et al., 2006). SysGen permet de construire des modèles graphiques de systèmes à partir d'une bibliothèque de composants (mémoires, blocs logiques, additionneurs...) et de générer un code VHDL synthétisable (Lazaro et al., 2006). Toutefois, l'absence concrète de moyen d'implémentation

limitait cette pratique. En effet, le processus de conception était entrecoupé dans le sens où le *continuum* de conception était à certains moments rompu. Typiquement, la simulation nécessitait le concours d'une application tierce, un simulateur VHDL, fonctionnant en parallèle. On parle de cosimulation. Ainsi, l'environnement de conception initial ne se suffit pas à lui-même.

Depuis Mathworks (2007) *via* son logiciel Matlab/Simulink a proposé une solution plus intégrée permettant la démarche de conception, simulation, validation et implémentation⁴ au sein d'un même environnement. En effet, la boîte à outils *HDL Coder* permet de générer du code HDL, Verilog ou VHDL, à partir de modèles Simulink. Avec la mise en place de la génération de code HDL *via* Matlab/Simulink, une étape a été franchie. En effet, HDL Coder permet de disposer de modèles vrais au bit et cycle près. On parle, en anglais, de *Bit-True Cycle-Accurate Models* (BTCAM) (Mathworks, 2007). Une telle génération permet de disposer de modèles de simulation reflétant le comportement du code VHDL, ce qui rend possible la création de bancs de test (ou en anglais *Testbenchs*) plus efficaces toujours dans le même environnement.

Ce type d'outil permet d'utiliser une démarche de conception à base de modèles ou *Model-Based Design* (MBD) en anglais. Il s'agit alors de construire un modèle simulation permettant le réglage de la fonctionnalité et des performances du système (Zhixun et al., 2011).

Par ailleurs, l'utilisation combinée du logiciel de Mathworks et de SysGen dote le concepteur de moyens efficaces pour le design d'un système complexe requérant un fort niveau d'abstraction. En effet, les éléments disponibles au sein de SysGen sont dédiés aux FPGA de la même firme et sont donc optimisés pour ceux-ci. Cependant, le coût relativement élevé d'un tel module complémentaire (minimum de 1000\$CAN⁵) peut être un frein à son

⁴ Il convient de nuancer. Nous le verrons plus tard, l'implémentation si elle est facilitée, n'est pas directement possible *via* Matlab/Simulink.

⁵ http://www.xilinx.com/onlinestore/design_resources.htm

utilisation dans le cas de petites compagnies s'étant déjà offertes une licence Mathworks. C'est pourquoi, dans le cadre de ce travail, nous nous focaliserons sur la boîte à outils HDL Coder, en laissant de côté SysGen mais en n'oubliant pas que ce module fut le pionnier dans le domaine.

La conception, ainsi de plus en plus intégrée, permet l'élaboration de systèmes toujours plus complexes. Si les FPGA sont longtemps restés dans le domaine des télécommunications et du traitement du signal comme le souligne Monmasson (2007) c'est parce que ces domaines requièrent des débits toujours plus importants. D'ailleurs, cela s'illustre lorsque l'on explore le contenu des modules précédemment cités dont les bibliothèques sont orientées vers ces domaines. On le comprend d'autant plus que le parallélisme des FPGA permet d'obtenir des vitesses de calcul jusqu'à cent fois supérieures à celle des *Digital Signal Processors* (DSP) (Zhengwei, Carletta et Veillette, 2005). Néanmoins, les télécommunications et le traitement du signal ne constituent pas des domaines réservés. En effet, peu à peu l'intérêt des concepteurs de régulateurs et de systèmes embarqués envers les FPGA n'a eu cesse de croître au cours des dernières années comme le démontre Monmasson (2007).

Ainsi, la plus grande intégration des outils au sein d'un même environnement (permettant d'effectuer toutes les tâches de la conception à l'implémentation) combinée à la prise en compte de hauts niveaux d'abstraction ont, aujourd'hui, rendu possible la conception de systèmes complexes et performants à base de FPGA tout en réduisant le temps de développement (Ownby et Mahmoud, 2003), (Paiz et Pormann, 2007) et (Rahul, Pramod et Vasantha, 2007) . Cette dynamique s'applique aux développements de contrôleurs et des systèmes embarqués qui bénéficient de quatre facteurs :

- la réduction des coûts,
- la sécurité,
- la tolérance aux contraintes (temporelles, de consommation),
- une meilleure qualité du contrôle.

La réduction des coûts est liée à trois facteurs (Monmasson, 2007) : une architecture dédiée à une tâche spécifique, la réduction du temps de mise sur le marché (TTM pour *time to market*) et l'intégration, à terme, totale d'un système embarqué complet (entrées/sorties analogiques) sur un *System-On-a-Chip* (SoC). La sécurité des informations est aussi assurée via l'encryptage des données (Maxfield, 2004) dont le type diffère selon les distributeurs et le type de FPGA. La tolérance aux contraintes est, elle, assurée par la nature même d'un FPGA. En effet, ce dernier permet un parallélisme accru du calcul réduisant le temps de traitement, et permettant d'avoir de longue période de veille (peu de circuiterie active, donc peu de consommation). Enfin, ce type de contrôleur accroît ostensiblement la qualité du contrôle (par rapport à une autre solution numérique) puisque l'on se rapproche d'une dynamique analogique (par la rapidité du traitement) tout en conservant une grande flexibilité.

Il reste alors à développer des composants dédiés à la création de régulateurs au sein d'une librairie intégrée à Matlab/Simulink. Cela permettrait à des personnes non-initiées aux FPGA (et à l'électronique de façon générale) de pouvoir se doter d'organes de commande performants en utilisant un environnement largement répandu (Matlab/Simulink). Ce projet s'inscrit dans la continuité de disposer d'outils à haut niveau d'abstraction mais respectueux des contraintes matérielles. Ces dernières devront être prises en compte lors de la conception des éléments de la bibliothèque dédiée à la commande de systèmes.

Les avantages énoncés précédemment sont toutefois à nuancer un peu. Si l'efficacité d'un régulateur formé d'un FPGA est indéniable, sa mise en pratique comporte quelques zones plus complexes requérant, malgré un développement à haut niveau d'abstraction, des considérations matérielles. La façon dont sont gérés les nombres est un exemple. Ensuite, il faut savoir ce que les opérations que l'on souhaite effectuer impliquent au sein du FPGA. Si une addition ou une soustraction ne demande que peu de ressources, une multiplication est très gourmande en termes d'espace (Chan, Moallem et Wang, 2007) (Gupta, Khare et Singh, 2009) . Dès lors, il est nécessaire de comprendre ce qui se passe à bas niveau pour pouvoir être en mesure de développer des systèmes de haut niveau dont le fonctionnement demeure le même à haut comme à bas niveau.

La commande de systèmes repose sur trois principaux objectifs selon Nise (2011) : assurer la stabilité du système à commander, disposer d'une dynamique conforme au cahier des charges et avoir une réponse en régime établi conforme aux spécifications. Ainsi, le correcteur le plus simple à mettre en place serait un régulateur proportionnel. Dans les faits, cette simple implémentation implique une multiplication. Il est alors préférable de s'affranchir des contraintes matérielles en utilisant une méthode de calcul qui n'utilise pas de multiplication. Chan, Moallem et Wang (2004; 2007) et Gupta, Khare et Singh (2009), entre autres, ont travaillé sur des correcteurs, souvent de type Proportionnel-Intégral-Dérivé (PID), en utilisant des architectures sans multiplicateurs. Ces structures de calculs ont été présentées pour la première fois par Peled et Bede (1973) avant que White (1989) n'établisse un article référence, sur ce que l'on appelle l'Arithmétique Distribuée (AD). Dans leur travaux Chan, Moallem et Wang (2004; 2007) ont montré que la mise en place de l'AD permet de réduire la consommation de 40% et de diminuer l'occupation mémoire. Toutefois, les compagnies de FPGA (Xilinx, Altera) ont fait évoluer leurs produits et aujourd'hui ceux-ci disposent de multiplicateurs qui permettent un développement encore plus rapide si le système à commander est simple (Lazaro et al., 2006).

Nous allons donc chercher à savoir, si oui ou non, l'AD peut rester une bonne méthode de calcul dans certains cas. Nous poursuivrons, après avoir construit un compensateur de type simple, en développant un régulateur plus complexe qui permette de profiter des FPGA et de leurs performances (Scrofano, Seonil et Prasanna, 2002). Parallèlement à cela, nous insérerons les éléments de commande conçus afin de constituer une bibliothèque Matlab/Simulink de composants synthétisables et implémentables sur un FPGA et facilement utilisables par des concepteurs n'ayant que des connaissances limitées sur le FPGA.

Afin de mener à bien ces objectifs, nous allons, dans un premier temps, nous intéresser aux FPGA et leur fonctionnement. Ce premier point nous permettra de toujours garder à l'esprit le lien à tisser entre haut et bas niveau d'abstraction (à l'échelle des bits, des registres).

Nous effectuerons ensuite un bref rappel sur les systèmes asservis qui nous permettra de disposer des bases nécessaires à l'étude critique de nos résultats car ne perdons pas de vue que l'on souhaite disposer d'un système de commande. Les systèmes de commande à l'épreuve seront :

- une commande type PID, la plus largement répandue dans l'industrie (Yu, 1999),
- une commande moderne munie d'un observateur d'état.

Le choix de ces deux commandes est simple. La première, la plus basique, la plus répandue, est également la plus usitée dans la littérature liée à la commande par FPGA. La seconde permet de disposer d'une commande plus complexe, tout en disposant d'un cadre théorique bien défini (Rugh, 1991), où le nombre d'opérations (et de multiplications) dépend de la dimension du système. Nous chercherons donc à savoir si ce qu'ont mis en évidence Scrofano, Seonil et Prasanna (2002), en termes d'efficacité de la multiplication matricielle, se révèle pertinent dans le domaine de la commande de systèmes.

Nous nous intéresserons ensuite à la façon dont sont traités les nombres dans un système numérique. Nous nous focaliserons plus particulièrement sur les nombres à virgules fixes puisqu'ils sont directement synthétisables en VHDL.

A la suite, nous tâcherons de montrer que l'utilisation de logiciels à haut niveau d'abstraction se prête particulièrement bien à la conception de systèmes de commande embarquée. En effet, nous tenterons de mettre en évidence les avantages de tels outils afin de réduire le temps de mise sur le marché. Enfin, dans le but de poursuivre une démarche de conception à bases de modèles performantes nous verrons qu'il sera nécessaire de passer par la création d'une bibliothèque de composants dédiés à la commande.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Field Programmable Gate Array (FPGA)

Un FPGA est un circuit intégré (CI), qui est défini comme une matrice de blocs logiques configurables (*Configurable Logic Blocks*, CLB) reliés entre eux par des interconnexions entièrement reconfigurables (Monmasson, 2007). Ces blocs et interconnexions peuvent être programmés par l'utilisateur *via* des cellules mémoire statiques (Xilinx, 2011a).

Les FPGA sont des appareils programmables disposant de quelques centaines de milliers à quelques millions de portes logiques configurables. Il est important de noter que dans le cas d'un FPGA, on parle de configuration plus que de programmation. En effet, les données que l'utilisateur charge dans le FPGA affectent directement l'architecture matérielle de ce dernier, il ne s'agit pas d'instructions purement logicielles.

Les FPGA sont des circuits logiques configurables *in situ* (*field-programmable*). Ils sont venus remplir, à partir des années 1980, le vide que comportait le spectre des circuits logiques. Nous avons d'un côté les PLD qui, s'ils devenaient de plus en plus complexes (on parle de *Complex PLD* (CPLD)), ne permettaient pas de réaliser des fonctions de haut niveau, de l'autre les ASIC munis d'une architecture fixe. Ces derniers sont conçus pour des applications spécifiques et ne sont pas configurables.

Ainsi, étant reconfigurables à souhait, les FPGA peuvent servir à plusieurs applications ce qui représente un sérieux avantage.

Les FPGA sont donc des circuits logiques qui se configurent. La mise en œuvre de ceux-ci est radicalement différente de la programmation à laquelle nous pouvons d'ordinaire être habituée : celle des microprocesseurs (μ P) ou microcontrôleurs (μ C). En effet, ces derniers fonctionnent de façon séquentielle : tâche 1 puis tâche 2 (tâche 1 et tâche 2 indépendantes)

Cela vient du fait même de leur constitution : lecture du registre d'instruction, traitement de l'instruction etc.

Les FPGA, quant à eux, jouissent d'un parallélisme total si bien que dans l'exemple précédent la tâche 1 et la tâche 2 s'effectuent simultanément. Le parallélisme leur confère une grande rapidité en comparaison des μP et μC . Toutefois, cela a pour effet de considérablement modifier la façon dont on « programme » ces composants (le séquençement par exemple). Il existe deux principaux langages pour configurer les FPGA ce sont des langages dits de description matériel, *Hardware Description Language* (HDL) dont les deux largement répandus sont le Verilog et le VHDL.

Pour résumer, les FPGA sont des circuits numériques dotés d'une grande multitude de portes logiques programmables par l'utilisateur permettant la réalisation de fonctions de traitement dédiés à un besoin.

1.2 Notions fondamentales

Nous allons à présent revenir sur les aspects fondamentaux des FPGA à savoir leur place dans l'histoire de l'électronique ainsi que les phénomènes physiques liés à leur technologie. Enfin, nous verrons comment ces circuits peuvent servir à la conception de systèmes de haut niveau grâce à des outils qui n'ont eu cesse de se développer.

1.2.1 Historique

L'apparition des premiers FPGA, par la firme Xilinx, remonte à 1985 avec le XC2064 (Santo, 2009). Mais ce n'est qu'à partir des années 1990 qu'ils commencent à susciter un intérêt plus grand. Pour mieux comprendre le phénomène, il est nécessaire de revenir quelques années en arrière.

Les FPGA s'inscrivent, en plein, dans la grande Histoire de l'Electronique et ont pour point de départ l'apparition du premier transistor en 1947. Les transistors sont le fruit de recherches intenses, et en 1962 la technologie *Metal-Oxyde Semiconductor Field Effect Transistors* (MOSFET) émerge et permet la conception de circuits moins gourmands en énergie. Parallèlement, le concept de CI se développe et Jack Kilby, de Texas Instrument réussit en 1958 à créer le premier circuit intégré composé de cinq composants réunis sur le même morceau de semi-conducteur (Maxfield, 2004). Au cours des années 1960 se sont développées des gammes de CI, notamment chez Texas Instrument, proposant une petite dizaine de portes logiques.

Les premiers CI programmables arrivent sur le marché au début des années 1970 sous forme de *Programmable Read Only Memory* (PROM). En parallèle, se développent le premier microprocesseur (μ P), et les mémoires de type *Static Random Access Memory* (SRAM). Le développement des mémoires (PROM, EPROM, E2PROM) permet la réalisation de CPLD toujours plus complexes. Ainsi, la société Altera mélange, en 1984, la technologie EPROM avec les MOSFET permettant de disposer d'un circuit numérique programmable ne consommant pas trop d'énergie (Maxfield, 2004). Le concept de tableau logique (*logic array*) fait son apparition. Le principe réside, comme le résume la Figure 1.1, en la programmation de liens logiques à l'aide d'opérations simples (AND, OR, XAND, XOR...). Typiquement, sont présents en entrée plusieurs signaux avec leur complémentaire, on dresse alors la matrice qui permet de connecter tous les signaux. Puis, lors de la programmation, on décide des nœuds que l'on souhaite activer afin de réaliser la fonction désirée. Ainsi les CPLD ont nécessité des moyens plus évolués de mise en œuvre à mesure que leur taille grandissait. On voit alors apparaître les premiers langages de description matérielle (HDL) qui serviront de base au VHDL et au Verilog, à ce titre on peut citer le langage *Advanced Boolean Expression Language* (ABEL) introduit par Data I/O en 1983.

Par ailleurs, le développement des ASIC s'effectue aussi en utilisant le principe de *logic array*. Si la conception des premiers circuits ASIC se faisait à la main, à l'aide de masques, la technologie, évoluant, et la finesse de gravure, permettant d'avoir plusieurs millions de

composants sur une même puce, rendent l'élaboration d'outils d'aide à la conception nécessaire. Ainsi l'émergence des langages HDL, VHDL et Verilog, avec la mise en place de standards IEEE (1988; 1994), a permis le développement d'outils d'aide à la conception électronique (*Computer Aided Design, CAD*) de hautes performances (Monmasson, 2007). On dénomme ce type de logiciel par *Electronic Design Automation (EDA)*. Ils permettent la conception et l'implémentation des CI complexes.

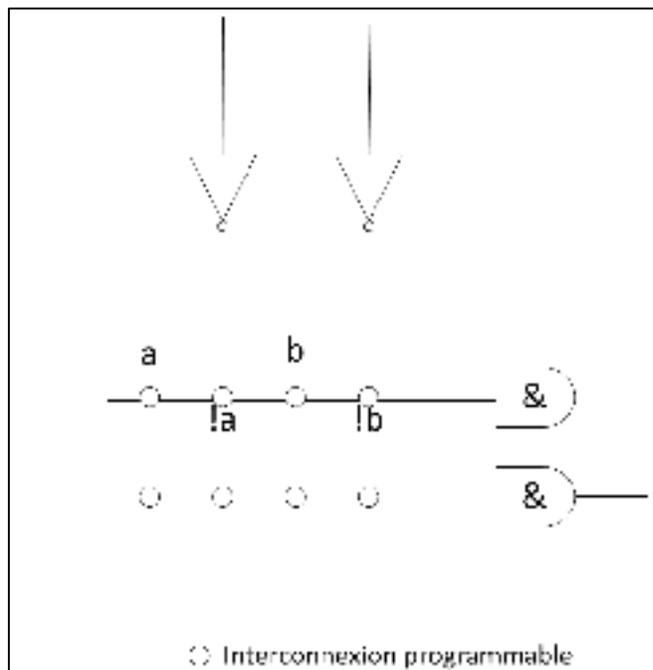


Figure 1.1 Principe d'un tableau logique (ou *Logic Array*)

1.2.2 Concepts

Les concepts fondamentaux des FPGA sont très bien résumés par Maxfield (2004). L'élément clef de la compréhension réside dans la méthode de configuration des FPGA. En effet, les techniques de programmation ont considérablement évolué passant d'un système programmable une unique fois à un système capable de supporter un nombre infini de programmation.

Nous pouvons voir les interconnexions de la Figure 1.1 comme des interrupteurs. Au début de l'ère du numérique la technique des fusibles (*fuses*) était utilisée. La méthode consiste à choisir de façon définitive les connexions que l'on souhaite effectuées en « grillant » le fusible qui nous est inutile. Nous pouvons également envisager le principe inverse : l'anti-fusible (*anti-fuses*). On dispose d'anti-fusibles sur chacune de nos lignes, à l'état initial les anti-fusibles sont dotées d'une telle résistance que le circuit est vu comme ouvert. Lorsque l'utilisateur programme l'appareil configurable, il injecte une impulsion de courant sur l'un des anti-fusibles qui devient passant. Ce phénomène est rendu possible par la nature isolante que présente le silicium à l'état amorphe et son pouvoir conducteur lorsqu'il est sous forme cristalline. Dans les deux cas présentés, les appareils ne sont programmable qu'une seule fois, on parle de *One Time Programmable* (OTP).

Ces techniques ont été utilisées pour réaliser les premières mémoires de type *Read-Only Memory* (ROM). Toutefois, lors des phases de conception, il devenait fastidieux de sans cesse changer de ROM pour modifier les valeurs. Pour résoudre ce problème, Intel, en 1971, présente l'*Erasable Programmable Read-Only Memory* (EPROM) qui permet d'effacer la mémoire et d'y inscrire de nouvelles valeurs. Le principe de cette mémoire est quasi-similaire au MOSFET, y ajoutant seulement une seconde grille, appelée grille flottante, qui permet de maintenir le transistor à l'état passant lorsque le signal de commande a disparu. Pour effacer la valeur en mémoire, il suffit de « décharger » la grille flottante. Pour cela, une source de rayonnement ultraviolet est nécessaire. Toutefois, les mémoires EPROM ne permettent pas d'effacer les données stockées de façon précise, l'ensemble de la mémoire doit être effacé. C'est ensuite au tour de l'*Electrically Erasable Programmable Read-Only Memory* (EEPROM ou E2PROM) d'arriver sur le marché. L'E2PROM permet d'effacer précisément les données grâce à un signal électrique. La constitution physique de l'E2PROM est très proche de sa petite sœur, l'EPROM, c'est uniquement l'épaisseur, plus fine, de l'oxyde isolant qui permet à un transistor de fournir l'énergie nécessaire à la décharge de la grille flottante.

Cependant, la solution la plus répandue pour la configuration des FPGA ne réside pas dans des cellules mémoires mortes mais dans des mémoire vives, *Random Access Memory* (RAM) (Trimberger, 1993; Xilinx, 2011b). Il s'agit d'une cellule appelée *Static RAM* (SRAM). Cette cellule permet de changer sa valeur à tout moment, mais le principal problème réside dans le fait que les données sont perdues lorsque l'alimentation est coupée.

1.2.3 Constitution interne

Nous avons évoqué précédemment qu'un FPGA est constitué d'une matrice de CLB. Ces blocs sont eux-mêmes composés de quatre tranches (*Slices*) qui sont elles même formées par deux cellules logiques (*Logic Cell, LC*) (Xilinx, 2011b). La Figure 1.2 présente la structure d'un CLB.

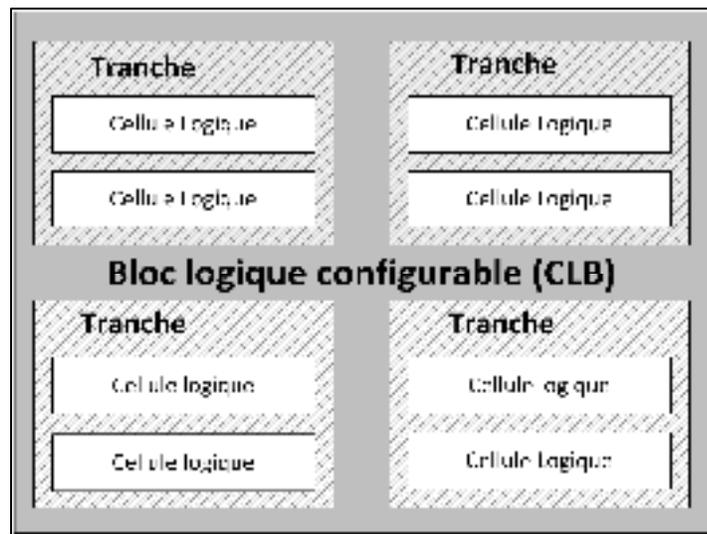


Figure 1.2 Structure d'un bloc logique configurable (CLB)

La LC est la plus petite structure présente au sein d'un FPGA, elle est composée d'une table de conversion (*Look-Up Table, LUT*) à quatre entrées, d'un multiplexeur, d'un registre et d'une gestion de la retenue (Xilinx, 2011b) comme le rappelle la Figure 1.3. La gestion de la retenue (*carry* en anglais) est interne à chaque LC mais une chaîne de gestion est mise en place à travers tous les blocs quelque soit leur niveau hiérarchique (Xilinx, 2011b), de sorte

que les performances des fonctions logiques soient améliorées (Trimberger, 1993). Par ailleurs, la LUT à quatre entrées, présente dans chaque LC, est composée de seize cellules SRAM. L'utilisation de la SRAM pour construire les LUT permet, de fait, une grande flexibilité. En effet, lors de l'implémentation, le bloc formant la LUT peut être instancié sous différentes formes : une LUT standard à quatre entrées, un bloc 16×1 bit RAM (on parle de RAM distribuée) ou un registre. Cela permet de disposer d'une architecture matérielle adaptée à la fonction souhaitée. Par ailleurs, les blocs d'interconnexions présents entre chaque CLB reposent sur des points d'interconnexions programmables (PIP). Ces PIP sont formés par un transistor contrôlé par une cellule mémoire de type SRAM. Ainsi, la sortie d'un CLB peut être routée vers un autre bloc de traitement *via* un PIP à l'état passant. La Figure 1.4 en rappelle le principe.

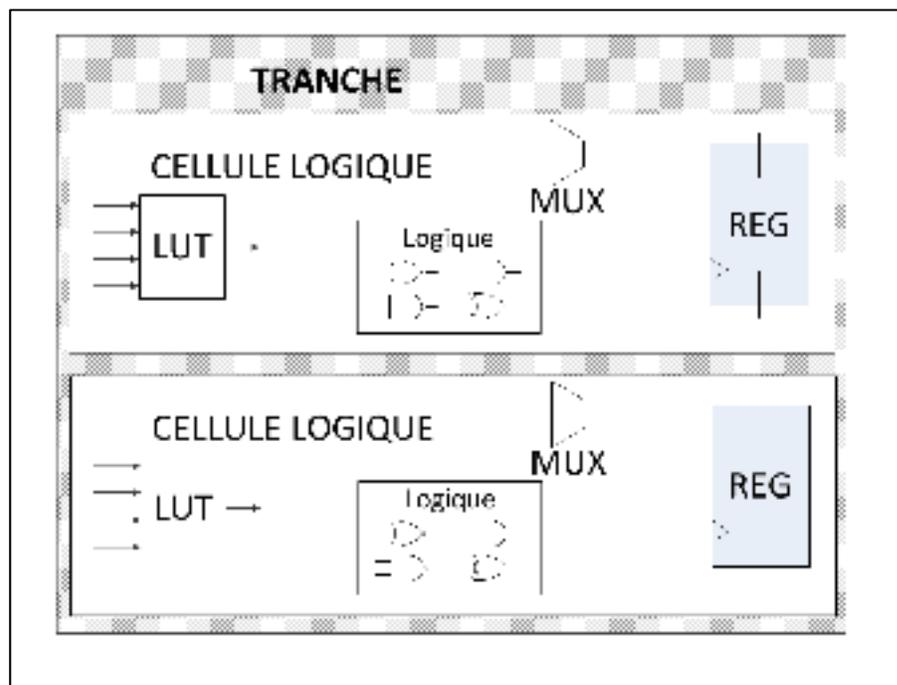


Figure 1.3 Détail d'une tranche

A la grande souplesse de configuration des FPGA, s'ajoute des éléments permettant la gestion de fonction plus complexes. Typiquement, des blocs de RAM embarqués sont insérés dans la matrice formée par les CLB et les PIP. Ces blocs se distinguent des modules de RAM

distribuée par une plus grande capacité. En outre, les FPGA peuvent embarquer des processeurs. On distingue deux types de processeurs pouvant être mis en place : les *soft core* et *hard core*. Les processeurs *hard core* sont des processeurs déjà câblés qui sont insérés (physiquement) dans la structure matricielle des FPGA. En revanche, les processeurs de type *soft core* sont construits à l'aide des blocs présents dans le FPGA. Le terme *soft* fait référence à la partie « logicielle ». En effet, un processeur *soft core* est créé à l'aide d'un langage de description. Ainsi la fonction de processeur *soft* est menée à bien par une partie du FPGA qui se comporte « comme » un processeur. La possibilité de disposer d'un processeur permet d'élargir encore un peu plus le spectre d'utilisation des FPGA leur permettant le traitement de fonctions complexes et séquentielles. Enfin, les nouvelles générations de FPGA⁶ embarquent systématiquement des blocs multiplicateurs qui permettent la gestion d'opérations 18×18 bits signés (Rahul, Pramod et Vasantha, 2007; Xilinx, 2009).

⁶ Au moins depuis avril 2003 avec la sortie du Spartan 3 de Xilinx.

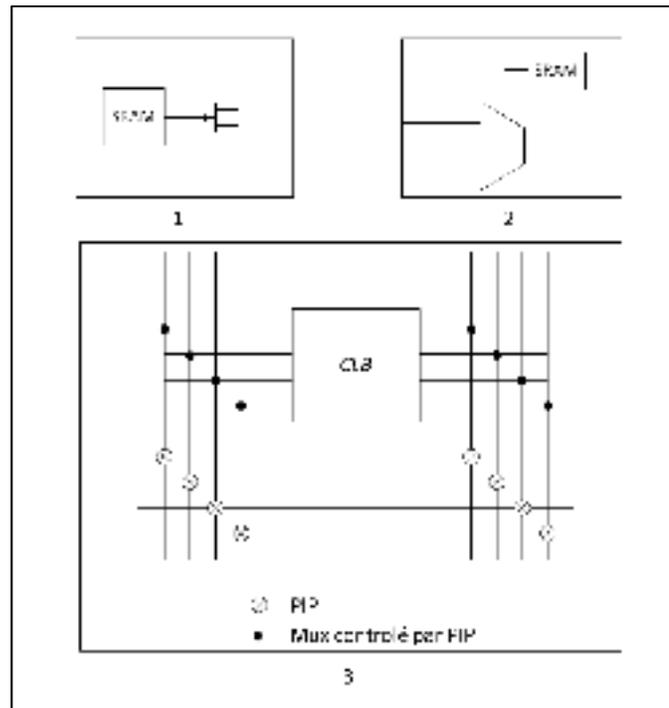


Figure 1.4 Points d'Interconnexions Programmables
 1) Structure d'un PIP
 2) Multiplexeur contrôlé par une cellule mémoire
 3) Connexions des CLB
 Adaptée de Trimberger (1993, p. 3)

1.2.4 Démarche de conception

Riesgo (1999) présente une démarche de conception, ou parfois appelée *flot*⁷ de conception comme le fait Derrien (2002), de haut en bas (*top-down*) qui consiste à partir du niveau d'abstraction le plus élevé pour finir à l'implémentation physique et électronique.

Les niveaux d'abstraction présentés par Derrien (2002), Maxfield (2004), Zerbe et Andelkovic (2004), Cong et *al.* (2011) sont, du plus haut au plus bas :

- le niveau « système » qui permet de définir les attentes du circuit à implémenter ;
- le niveau « comportemental » qui représente la description algorithmique ;

⁷ Le terme *flot* fait directement sens puisqu'en anglais on utilise le mot *flow*

- le niveau « structurel » (*Register-Transfert Level*, RTL) qui décrit le système à l'aide de primitives logiques ;
- le niveau « physique » qui prend en compte l'appareil programmable choisi et rend compte de la fonction d'un point de vue électrique.

Quelque soit le niveau d'abstraction choisi pour implémenter la fonction désirée, le flot de conception se résume en quatre étapes comme le rappelle Derrien (2002) :

- la synthèse qui permet de passer d'une description abstraite à une architecture implémentable. Notons que la phase de synthèse est réalisée à l'aide d'un outil logiciel ;
- l'allocation des ressources qui permet faire correspondre le résultat de la synthèse aux ressources disponibles sur le FPGA ;
- le placement et routage (PAR) qui permet d'associer chaque ressource nécessaire à un bloc logique configurable tout en programmant les interconnexions ;
- la génération du fichier de configuration qui est le fichier chargé dans le FPGA afin d'obtenir la configuration requise.

1.3 Conception de systèmes à base de FPGA

A la lumière de ce que sont les FPGA, et de leurs routines d'implémentation, il convient désormais de s'attarder sur leur domaine d'application et de regarder en quoi ils peuvent présenter un avantage dans la commande des systèmes.

Nous venons de voir que les FPGA embarquent de plus en plus de fonctionnalités (processeurs, multiplicateurs, mémoires...). Il reste à définir cependant leur domaine d'utilisation. Nous verrons d'abord les outils logiciels à la disposition du concepteur. Nous nous focaliserons ensuite sur l'utilisation des FPGA dans les boucles de commande.

1.3.1 Comment utiliser les FPGA ? Quels sont les outils à notre disposition ?

Dans leurs premières heures, les FPGA disposaient d'un champ d'application réduit. En effet, ils étaient utilisés dans une logique de glue permettant d'interfacer plusieurs PLD au moyen de décodeurs d'adresses ou de multiplexeurs (Derrien, 2002). Ce domaine d'utilisation réduit était directement induit par des outils logiciels d'implémentation limités (Maxfield, 2004; Monmasson, 2007). Les concepteurs devaient alors placer chaque transistor si bien que la mise en place de systèmes complexes était fastidieuse pour ne pas dire impossible. Ainsi, un effort conséquent a été fourni pour doter le concepteur de moyens efficaces. Très vite, les logiciels alors sur le marché proposent une implémentation graphique à l'aide de schématiques intégrant des portes logiques élémentaires (XOR, NAND, registres...). Toutefois, ce niveau d'abstraction reste limité. En effet, comme le souligne Derrien (2002), le concepteur doit à la fois faire la description structurelle (ou RTL) et comportementale. Cela revient à dire qu'à l'aide d'une seule couche d'abstraction, on doit pouvoir décrire la structure basse et électronique (les portes logiques) tout en assurant le bon comportement de la fonction implémentée. On comprend donc que ce genre d'outils limite considérablement l'évolution des fonctions implémentées, puisque le schéma est figé et la modification d'un paramètre du circuit doit se répercuter sur l'ensemble des paramètres associés⁸ (Maxfield, 2004).

La mise en place, très tôt, de HDL standardisés IEEE (1988; 1996) a permis de s'affranchir du problème de la représentation de plusieurs couches d'abstraction au sein d'une même interface de description. En effet, les langages VHDL (IEEE, 1988; 1999; 2007) et Verilog (IEEE, 1996; 2005) se trouvent à un plus haut niveau d'abstraction et permettent la description comportementale de la fonction à implémenter sans le souci de la définition structurelle. Par ailleurs, Maxfield (2004) introduit une nuance entre les deux langages : le VHDL dispose d'un niveau d'abstraction légèrement supérieur à ce que propose le Verilog

⁸ On peut par exemple citer la modification de la taille d'une variable de travail où les mémoires, registres et portes logiques par lesquels passe la variable modifiée doivent être également révisés.

(on peut citer les différences entre les types abstraits et fixes). Ainsi, les recherches sont toujours allées dans le sens d'une conception se faisant à plus haut niveau d'abstraction comme le soulignait très tôt Brown (1996).

A partir de la fin des années 1990, l'augmentation du niveau d'abstraction passait par la possibilité de concevoir un système, non pas à l'aide de HDL mais grâce au langage C (Cong et al., 2011). Toutefois, Shanblatt (2005) remarquait qu'un outil intégré à un logiciel de haut niveau comme Simulink permettant de passer de la description d'un système à une implémentation sur FPGA manquait. Les premières ébauches de travail dans ce sens ont pourtant démarré très tôt. Typiquement, Ličko (2003) présentait différentes façons d'effectuer un prototypage rapide à l'aide de Simulink. Mais d'où venait ce besoin d'environnements de conception de haut niveau ? Brettle (2009) et Cong et al. (2011) expliquent que l'utilisation d'environnements de conception de haut niveau largement usités dans le monde scientifique permet de toucher plus d'ingénieurs. En effet, les récents progrès ont permis de bâtir un pont entre deux mondes : les spécialistes des circuits numériques et les concepteurs de systèmes (quelque soit le domaine). Mais les environnements de développements largement utilisés dans le monde de l'ingénierie embarquant des fonctionnalités dédiés aux FPGA attendent la fin des années 2000 pour émerger comme le montrent Han (2007), Brettle (2009) ou le présente Auger (2008).

La compagnie The Mathworks (2007) présente une boîte à outils, HDL Coder, permettant la génération de code HDL synthétisable à partir de schéma Simulink. De la même façon Brettle (2009) présente l'utilisation du module FPGA du logiciel LabView de la société National Instruments. Ces avancées permettent en effet de s'affranchir des contraintes importantes liées aux langages HDL au profit d'un environnement de développement largement répandu. Aujourd'hui les EDA embarquent donc des modules de génération automatique de codes HDL (VHDL ou Verilog) destinés à être synthétisés.

Devant la complexité grandissante des systèmes à concevoir, Zhixun et al.(2011) rappellent que l'approche directe au niveau « système » lors de la conception permet de se focaliser sur

la fonctionnalité. On parle alors de conception à base de modèles (MBD). Le MBD utilise les modèles du système comme des spécifications tout au long du processus de développement et répond aux contraintes imposées à la conception de systèmes complexes et hétérogènes (Rongqing et al., 2011).

La méthodologie de conception descendante permet grâce aux développements des outils logiciels d'avoir des fonctionnements garantis. Toutefois, il subsiste une rupture dans le *continuum* des EDA. En effet, aujourd'hui nous avons toujours deux phases distinctes lors de la conception de systèmes numériques. D'abord, nous l'avons vu, une étape de conception se réalisant dans un environnement présentant un fort niveau d'abstraction, permet de disposer d'un code HDL synthétisable. Ce code doit ensuite être traité via un second environnement de travail. Ce deuxième ensemble permet un traitement successif correspondant aux étapes de synthèse, d'allocation des ressources, de Placement et routage (PAR pour *Place and Route*) et de génération de fichier de configuration décrite dans la partie 1.2.4. Dans notre travail, nous avons utilisé *Integrated Synthesis Environment* (ISE) proposé par Xilinx qui propose une suite de logiciels permettant la synthèse, le PAR et l'implémentation. Une présentation plus complète de cet environnement est proposée en ANNEXE I. Cet environnement de travail est lui aussi de haut niveau et les étapes du flot de conception sont presque transparentes pour l'utilisateur final. Celui-ci doit simplement s'assurer d'effectuer les tâches dans le bon ordre.

Au final, nous avons retracé l'historique de mise en place de systèmes de haut niveau de conception, en indiquant les raisons et montrant que cela répondait à un réel besoin d'abstraction. En connaissant à présent les bases des FPGA et les moyens de les utiliser, il nous reste à définir leurs domaines d'application.

1.3.2 Domaine d'utilisation

La migration technologique au profit des FPGA s'opère selon quatre facteurs selon Paiz (2007) :

- la performance algorithmique ;
- la flexibilité ;
- les coûts ;
- la consommation.

Il convient néanmoins de développer ces quatre points à la lumière de ce que nous avons établi précédemment. Tout d'abord, la performance algorithmique est garantie *via* le calcul parallèle qui réduit le temps de traitement. La flexibilité est assurée par la reconfigurabilité induite par les cellules SRAM et par l'utilisation d'EDA permettant de disposer de systèmes matériels et/ou logiciel. Typiquement, Patel et Moallem (2010) évoque la possibilité d'embarquer des systèmes multiprocesseurs disposant de systèmes d'exploitation temps réel capable de réserver certaines tâches critiques à une exécution purement matérielle. La réduction des coûts est, quant à elle, rendue possible par plusieurs facteurs comme la réduction du temps de développement ou le rapport coût/performances. En effet, l'augmentation du niveau d'abstraction des EDA a permis la constitution de blocs réutilisables (bibliothèque) écourtant la phase de conception. Par ailleurs, l'émergence des aides à la conception permet de réduire le TTM permettant une plus grande viabilité économique des solutions à base de FPGA. Le rapport coût/performance est utilisé afin de choisir entre DSP et FPGA. En effet, ce rapport permet de savoir combien de DSP sont nécessaires afin de remplir la mission d'un seul FPGA. Les prix des DSP sont souvent moins dispendieux mais ce rapport permet de tirer une conclusion quant au choix qui doit être réalisé. Enfin, la question de la consommation a été traitée dans la littérature, notamment Scrofano, Seonil et Prasanna (2002) qui ont mis en évidence à la fois l'efficacité numérique et énergétique des FPGA par rapport aux DSP dans le cadre de multiplications matricielles.

Toutefois si tous les facteurs énoncés expliquent une tendance globale, il convient de présenter les éléments qui font des FPGA des solutions de choix dans le cas de systèmes de commande.

Les FPGA ont vu une croissance de leur utilisation au sein de systèmes embarqués de commande. Leur faible consommation et le temps de traitement très rapide en fait des candidats de choix. Monmasson (2007) explique que la vitesse d'opération des FPGA en fait des éléments « quasi analogiques » en présentant les avantages des éléments analogiques (délai réduit) tout en évitant les inconvénients (dérives des paramètres). Leur qualité de système numérique leur permet de disposer d'une grande flexibilité. C'est d'ailleurs ce que souligne Han (2007) en précisant que les FPGA dans les systèmes de commande permettent des ajustements rapides *in situ*. Par ailleurs, l'auteur note également la fiabilité du processus de conception. C'est ainsi que l'utilisation des FPGA à des fins de commande de systèmes s'est répandu en électrotechnique (Li, 2008), équipement médical (Ruei-Xi, Liang-Gee et Lilin, 2000), robotique (Sanchez-Solano et al., 2007) ou électro-hydraulique (Stubkier et al., 2011).

Si les DSP et les FPGA occupent deux champs d'application différents, ceux-ci se recouvrent quant on traite la question des systèmes électriques. Si les solutions logicielles sont privilégiées (i.e. l'emploi de DSP), c'est, selon Monmasson (2007), en raison de la récente émergence des solutions matérielles qui continuent « d'effrayer » les concepteurs traditionnellement habitués aux solutions logicielles. Toutefois, les mœurs évoluent et, Zhang et al. (2010) proposent un enseignement basé sur les logiciels de haut niveau.

En outre, l'efficacité du calcul permet de disposer d'un temps d'inactivité conséquent entre le traitement de deux échantillons. Ce laps de temps peut être mis au service d'une autre activité comme la commande d'un second système. En effet, Garcia et al. (2004) ont mis en place un contrôleur multiple (commande d'un convertisseur DC-DC à 16 phases) au sein d'un même FPGA permettant une plus grande maîtrise de la consommation. Typiquement, ce type d'implémentation constitue un marché de niche pour les FPGA et est particulièrement adapté au domaine aéronautique où la fiabilité et l'énergie forment deux composantes essentielles.

Au final, tous ces éléments rendent les FPGA intéressants dans l'optique de systèmes de commande embarquée. Nous avons vu comment les FPGA étaient conçus et mis en œuvre. Nous avons également essayé de comprendre les efforts récents pour aider le concepteur à

l'aide d'EDA très évolués. Cela nous a permis de définir un domaine d'application cohérent. Nous avons, en outre, pu montrer que les caractéristiques des FPGA (matériels et logiciels) convergeaient vers l'idée d'en disposer comme de systèmes de commande.

CHAPITRE 2

CONCEPTION DES SYSTEMES DE COMMANDE

Nous allons, dans cette partie, nous focaliser sur la commande des systèmes. Dans un premier temps, nous rappellerons les notions et concepts qui nous permettront ensuite de poser les bases de la construction de régulateurs. Nous décrirons ensuite deux types de contrôleurs : le PID, largement répandu dans l'industrie, et la commande moderne.

2.1 Notions fondamentales

Nous nous proposons de poser un certain nombre de bases théoriques liées à la commande. D'abord, un rappel mathématique succinct sur la transformée de Laplace nous permettra d'appliquer cette théorie aux systèmes dynamiques en introduisant la notion de *fonction de transfert*. Nous verrons ensuite comment nous les représentons schématiquement avant de présenter les enjeux sous-jacents à la dynamique des systèmes.

2.1.1 La transformée de Laplace

Bensoussan (2008) rappelle la théorie de la transformation de Laplace. La transformée, notée $F(s)$, d'un signal continu $f(t)$ est définie par :

$$F(s) = \int_0^{+\infty} f(t)e^{-st} dt \quad (2.1)$$

Avec s la variable complexe définie par :

$$s = \sigma + i \omega \quad (2.2)$$

Avec $(\sigma, \omega) \in \mathbb{R}^2$

Comme le rappelle Astangul (2008) l'équation (2.1) converge pour $\sigma > \sigma_0$. La transformée de Laplace, $F(s)$, de la fonction $f(t)$ peut également se noter par :

$$F(s) = \mathcal{L} [f(t)] \quad (2.3)$$

Il existe également une transformation inverse notée \mathcal{L}^{-1} , qui permet de passer d'un signal décrit dans l'espace de Laplace à un signal temporel. La transformée inverse de Laplace se réalise à l'aide d'une décomposition en éléments simples qui permet de représenter une fonction complexe en une somme de fonctions beaucoup simples permettant d'obtenir un signal dans le temps de façon plus commode (notamment à l'aide de tables).

La transformée de Laplace présente des propriétés tout à fait intéressantes qui nous sont utiles dans le domaine de la commande. Typiquement on peut présenter la propriété de dérivation qui nous servira dans la suite de notre travail. Cherchons la transformée de Laplace de la dérivée de la fonction $f(t)$, notée $f'(t)$.

Soit $Re(s) > \sigma_0$

$$\mathcal{L} [f'(t)] = \int_0^{+\infty} f'(t)e^{-st} dt \quad (2.4)$$

$$\mathcal{L} [f'(t)] = sF(s) - f(0) \quad (2.5)$$

De façon plus générale, la dérivée n-ième d'une fonction $f(t)$, notée $f^{(n)}(t)$, s'exprimera dans l'espace de Laplace par :

$$\mathcal{L} [f^{(n)}(t)] = s^n F(s) - \sum_{i=0}^{n-1} s^{n-1-i} f^{(i)}(0) \quad (2.6)$$

2.1.2 La transformée en Z

Dans le cas de signaux discrets on utilise une autre transformation dite en Z. La définition de la transformée en Z monolatérale, notée $F(z)$, d'un signal discret $f(n)$ est :

$$F(z) = \sum_{n=0}^{+\infty} f(n)z^{-n} \quad (2.7)$$

On peut également noter cela par :

$$\mathcal{Z}[f(n)] = F(z) \quad (2.8)$$

Une fonction continue, notée $f(t)$, soumise à échantillonnage, de période T_e s'écrit alors $f_e(t)$ tel que :

$$f_e(t) = \sum_{n=0}^{+\infty} f(nT_e)\delta(t - nT_e) \quad (2.9)$$

On peut alors effectuer la transformée de Laplace de l'expression précédente (Nise, 2011) :

$$F_e(s) = \mathcal{L}[f_e(t)] = \sum_{n=0}^{+\infty} f(nT_e)e^{-nT_e s} \quad (2.10)$$

En posant :

$$z = e^{T_e s} \quad (2.11)$$

On retrouve la définition (2.7) :

$$F(z) = \sum_{n=0}^{+\infty} f(nT_e)z^{-n} \quad (2.12)$$

Par ailleurs, la transformation en Z présente une propriété intéressante qui permet d'exprimer la transformation en Z pour un décalage de k échantillons :

$$Z[f(n - k)] = z^{-k}F(z) \quad (2.13)$$

2.1.3 Modélisation des systèmes physiques, fonction de transfert

Les systèmes physiques se modélisent à l'aide d'équations différentielles (ED). Comme l'indique Nise (2011) beaucoup de systèmes peuvent être décrits à l'aide d'ED linéaires et invariantes dans le temps. Ces systèmes répondent à l'appellation LTI *pour Linear Time-Invariant* et se présentent sous la forme :

$$\sum_{i=0}^n b_i y^{(i)}(t) = \sum_{j=0}^m a_j r^{(j)}(t) \quad (2.14)$$

Avec $y^{(i)}$ et $r^{(j)}$ qui représentent respectivement la dérivée i-ème de y(t) et j-ème de r(t).

Nous avons vu qu'il était possible de décrire des dérivées d'ordre n à l'aide de fonction de Laplace (équation (2.6)). On se propose d'appliquer le résultat sur l'équation (2.14) :

$$\begin{aligned}
& \left(\sum_{i=0}^n b_i s^i \right) Y(s) - \sum_{i=0}^{n-1} s^{n-1-i} y^{(i)}(0) \\
&= \left(\sum_{j=0}^m a_j s^j \right) R(s) - \sum_{j=0}^{m-1} s^{m-1-j} r^{(j)}(0)
\end{aligned} \tag{2.15}$$

En formant alors le rapport entre la sortie, y , et l'entrée, r , et considérant les conditions initiales nulles, on peut définir une fonction de transfert, notée $G(s)$:

$$G(s) = \frac{Y(s)}{R(s)} = \frac{\sum_{j=0}^m a_j s^j}{\sum_{i=0}^n b_i s^i} \tag{2.16}$$

Dans le domaine discret, les systèmes ne sont pas représentés par des équations différentielles mais par des équations aux récurrences (ou équations aux différences) telles que :

$$\sum_{i=1}^n b_i y(i) = \sum_{j=1}^m a_j r(j) \tag{2.17}$$

On peut alors réécrire l'expression (2.17) à l'aide de la propriété (2.13), en considérant les conditions initiales nulles, et on a :

$$\begin{aligned}
& b_1 z^{-(n-1)} Y(z) + \dots + b_{n-1} z^{-1} Y(z) + b_n Y(z) \\
&= a_1 z^{-(m-1)} R(z) + \dots + a_m R(z)
\end{aligned} \tag{2.18}$$

$$(b_1 z^{-(n-1)} + \dots + b_{n-1} z^{-1} + b_n) Y(z) = (a_1 z^{-(m-1)} + \dots + a_m) R(z) \tag{2.19}$$

$$\frac{Y(z)}{R(z)} = \frac{(a_1 z^{-(m-1)} + \dots + a_m)}{(b_1 z^{-(n-1)} + \dots + b_{n-1} z^{-1} + b_n)} \tag{2.20}$$

On définit alors la fonction de transfert discrète comme le rapport entre une sortie discrète et une entrée discrète :

$$G(z) = \frac{Y(z)}{R(z)} \quad (2.21)$$

2.1.4 Représentation des fonctions de transfert

On se propose de décrire un système constitué d'un processus (ou parfois appelé procédé) que l'on souhaite commander selon des performances spécifiées dans le cahier des charges. Soit une boucle de régulation représentée par la Figure 2.1 où G représente le procédé à commander (modélisant le processus physique) et C le correcteur (ou contrôleur, ou régulateur).

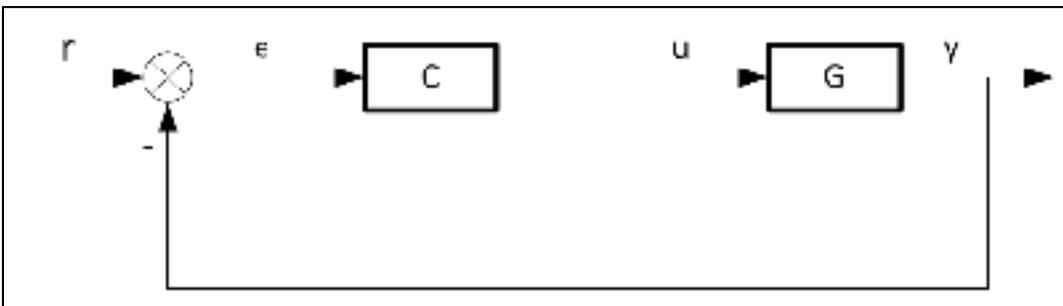


Figure 2.1 Schéma d'une boucle de commande à retour unitaire

La Figure 2.1 présente une boucle de régulation simple à retour unitaire. Les signaux r , e , u et y sont des signaux temporels représentant respectivement : la consigne, l'erreur, la commande et la sortie. Afin d'alléger les calculs, on ne se place pas dans le domaine temporel, mais dans le domaine de Laplace⁹ tel que :

$$R(s) = \mathcal{L}[r(t)], \quad E(s) = \mathcal{L}[e(t)], \quad U(s) = \mathcal{L}[u(t)], \quad Y(s) = \mathcal{L}[y(t)]$$

⁹ Le produit de convolution devient alors un simple produit

On peut alors définir la fonction de transfert (qui exprime un transfert entre une entrée et une sortie). Typiquement, C représente un transfert, dans le sens où il s'agit du rapport d'une sortie sur une entrée que l'on peut expliciter par (en se référant à la Figure 2.1) :

$$C(s) = \frac{U(s)}{E(s)} \quad (2.22)$$

La fonction de transfert de la boucle ouverte que l'on notera H_{BO} :

$$H_{BO} = C(s)G(s) \quad (2.23)$$

Ainsi la fonction de transfert en boucle fermée, notée H , correspondant au transfert de la consigne à la sortie s'exprime par :

$$H = \frac{H_{BO}}{1 + H_{BO}} = \frac{Y(s)}{R(s)} \quad (2.24)$$

$$H = \frac{C(s)G(s)}{1 + C(s)G(s)} \quad (2.25)$$

L'asservissement est la discipline qui veille à donner à H la dynamique souhaitée (Retif, 2008).

La commande est le domaine qui s'intéresse au compensateur C permettant au système total, H , d'avoir un comportement dynamique conforme au cahier des charges. Gardons en tête l'expression (2.22) qui présente toute fonction de transfert comme une fraction rationnelle. Le correcteur C permet alors au concepteur de choisir les pôles et les zéros qui sont respectivement les valeurs pour lesquelles le dénominateur et le numérateur de H s'annulent. Par ailleurs, notons que lors de l'étude des systèmes on s'intéresse à deux types de réponse :

- la réponse impulsionnelle qui est la réponse d'un système pour une entrée équivalente à une impulsion de Dirac¹⁰.
- la réponse indicielle (qui est plus commode à réaliser dans la pratique) qui donne la sortie d'un système soumis à un échelon en entrée.

Dans le domaine discret, il s'agit du même type de représentation.

2.2 Dynamique et placement des pôles

Le concepteur en choisissant la fonction de transfert H (via le compensateur C) impose une dynamique au système. En effet, si l'expression du transfert s'exprime dans le domaine de Laplace, on peut effectuer la transformée inverse afin d'obtenir une caractéristique temporelle du système.

Tout d'abord, notons qu'une fonction de transfert dispose d'un ordre. Celui-ci est égal au degré du polynôme présent au dénominateur, il s'agit également de l'ordre de l'équation différentielle. Toutefois, il est possible de se limiter à l'étude des systèmes du deuxième ordre. En effet, comme le rappelle Odet (2009) tout système d'ordre élevé peut se ramener à des systèmes d'ordre 1 et 2 en cascade. De plus, Nise (2011) souligne le fait qu'un système d'ordre élevé peut s'approximer par un deuxième ordre, sous certaines conditions. Nous allons évoquer ces conditions d'approximation mais avant nous décrirons les systèmes du deuxième ordre.

Les systèmes du deuxième ordre permettent de modéliser différents types de réponses des systèmes. Outre la vitesse de la réponse, la forme de celle-ci peut varier selon les différents paramètres en présence. Un système du deuxième ordre représenté par l'équation (2.16) (où $n = 2$), peut se représenter dans l'espace de Laplace par (au gain statique près) :

¹⁰ Dans l'espace de Laplace, la transformée d'un Dirac donne la valeur 1. Ainsi, pour obtenir la réponse impulsionnelle d'un système, il suffit de faire la transformée inverse de Laplace de la fonction de transfert.

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (2.26)$$

Cette formulation permet de faire apparaître deux paramètres essentiels : le coefficient d'amortissement ζ et la pulsation naturelle ω_n . On peut factoriser le dénominateur de l'expression précédente afin de faire apparaître la pulsation naturelle amortie ω_d :

$$G(s) = \frac{\omega_n^2}{(s + \sigma + j\omega_d)(s + \sigma - j\omega_d)} \quad (2.27)$$

Avec $\omega_d = \omega_n\sqrt{1 - \zeta^2}$ et $\sigma = -\zeta\omega_n$

Les pôles $s_{1,2}$ étant égaux à :

$$s_{1,2} = -\zeta\omega_n \pm j\omega_n\sqrt{1 - \zeta^2} \quad (2.28)$$

La Figure 2.2 présente la position des pôles dans le plan complexe pour différentes valeurs du coefficient d'amortissement ζ . On note que l'amortissement inférieur à 0 est purement conceptuel. La Figure 2.3 présente les réponses impulsionnelles associées à chacun des pôles présents dans la Figure 2.2.

La réponse impulsionnelle d'un tel système est donnée par :

$$g(t) = y(t) = \frac{\omega_n^2}{\omega_d} e^{-\zeta\omega_n t} \sin(\omega_d t) \quad (2.29)$$

La réponse indicielle, qui correspond au produit de $G(s)$ par $U(s) = \frac{1}{s}$ donne :

$$y(t) = g(t) * u(t) = 1 - \frac{\omega_n}{\omega_d} e^{-\zeta\omega_n t} \sin(\omega_d t + \phi) \quad (2.30)$$

$$\phi = \arccos \zeta$$

Cela, illustre la nécessité de disposer de pôles à partie réelle négative s'il l'on souhaite disposer d'un système stable *i.e.* un système qui dans le cas :

- de la réponse impulsionnelle : tend vers 0 quand le temps tend vers l'infini ;
- de la réponse indicielle : tend vers 1 quand le temps tend vers l'infini.

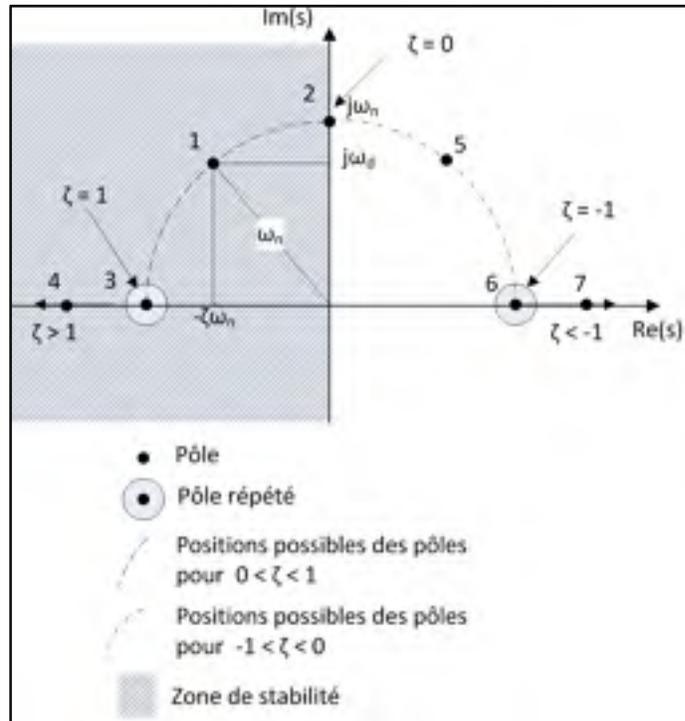


Figure 2.2 Position des pôles dans le plan de Laplace pour différentes valeurs de l'amortissement
 Adaptée de Bensoussan (2008, p. 81)

On peut, à l'étude des amortissements ζ , classer les différentes réponses produites par les systèmes du deuxième ordre, comme l'indique la Figure 2.3 :

- régime non-amorti : si $\zeta = 0$, alors les pôles sont purement imaginaires. Cela implique des oscillations constantes ;
- régime sous-amorti : si $0 < \zeta < 1$, alors les pôles conjugués sont à parties réelles négatives avec une partie imaginaire. La réponse converge avec quelques oscillations ;
- régime critique : si $\zeta = 1$, les pôles sont répétés et ne disposent pas de partie imaginaires, il n'y a pas de dépassement ;

- régime sur-amorti : si $\zeta > 1$, alors on dispose de pôles purement réels et négatifs, il n'y a pas de dépassement ;
- régime instable : $\forall \zeta < 0$.

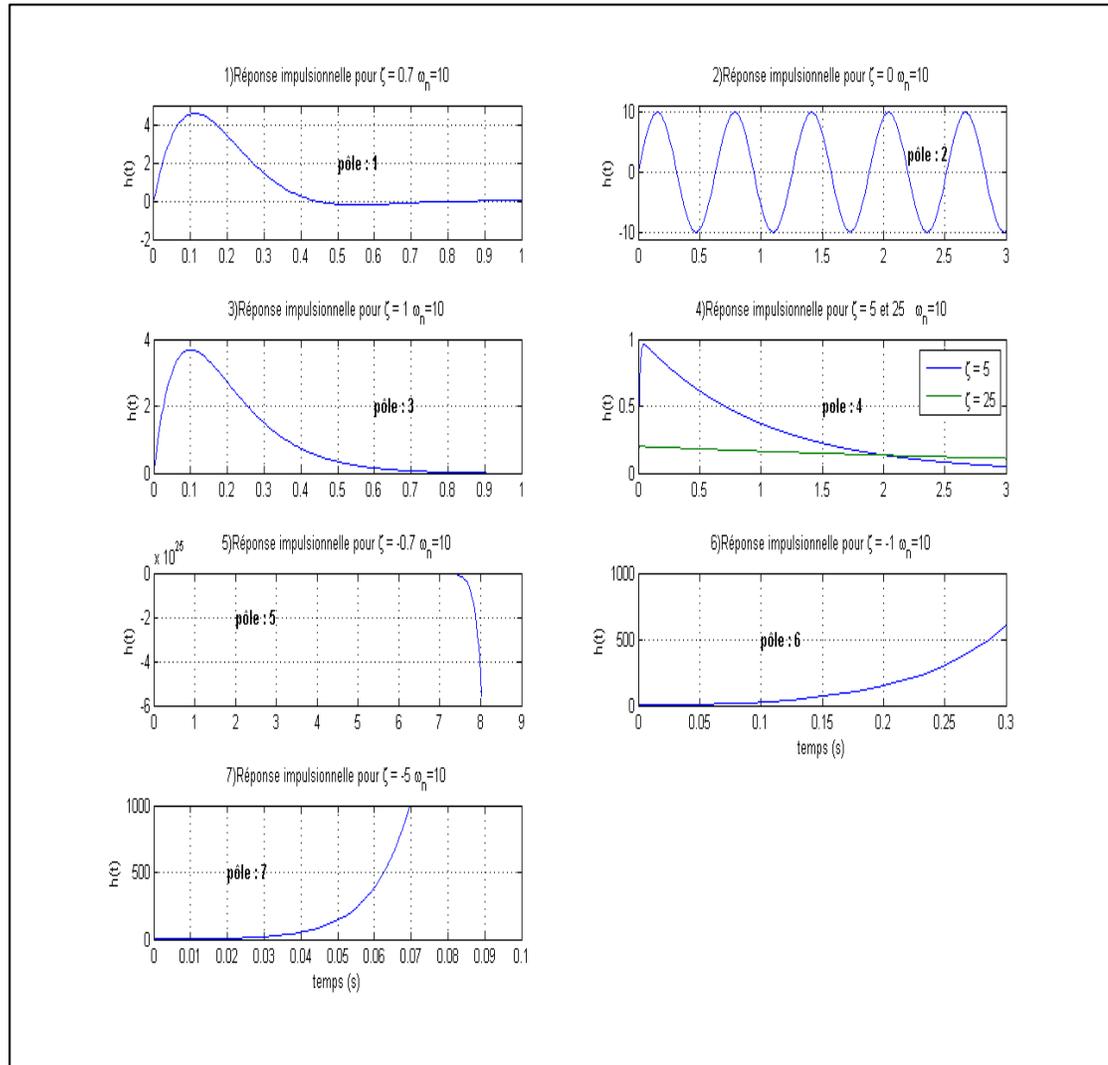


Figure 2.3 Réponse impulsionnelle en fonction de la position des pôles

- 1) La réponse d'un système sous-amorti
- 2) D'un système non-amorti
- 3) D'un système amorti
- 4) D'un système sous-amorti
- 5, 6, 7) D'un système instable

Notons que la Figure 2.3 donne les allures des réponses impulsionnelles des pôles représentés (et numérotés) sur la Figure 2.2.

Le cas du régime sous amorti est intéressant puisqu'il permet de définir des caractéristiques de réponse. Citons :

- le pourcentage de dépassement, $\%d$, exprime l'erreur relative entre la valeur finale et la valeur maximale atteinte et exprimée en pourcentage ;
- le temps d'établissement ou de réponse, T_r , exprime le temps nécessaire au système pour qu'il se stabilise autour de 2% de la valeur finale ;
- le temps de montée, T_m , exprime le temps que le système met pour passer de 10% à 90% de la valeur finale ;
- le temps de dépassement, T_d , exprime le temps auquel survient le dépassement maximum.

Revenons à présent sur les systèmes d'ordres plus élevés. Ceux-ci peuvent, comme nous l'avons indiqué, s'approximer (dans certains cas) par des modélisations du deuxième ordre. Les pôles d'une fonction de transfert d'ordre élevé peuvent être différents et à ce titre induire chacun une dynamique propre. En effet, pour revenir à une expression temporelle à partir d'une fonction de transfert, nous passons par une décomposition en éléments simples (ou parfois appelés fractions partielles). Ainsi, le signal temporel correspond à la somme de la contribution de chacun des pôles. Bensoussan (2008) rappelle d'ailleurs la forme de la solution à une équation différentielle d'ordre n dont les pôles ne sont pas répétés :

$$y(t) = \underbrace{\sum_{i=1}^k C_i e^{\sigma_i t}}_{\text{pôles réels}} + \underbrace{\sum_{j=1}^{n-k} C_j e^{(\sigma_j + j\omega_{d_j})t}}_{\text{pôles complexes}} \quad (2.31)$$

Cela a pour effet, dans certains cas de faire apparaître des pôles dominants (ou modes dominants). En effet, les pôles à partie réelle négative faible (en valeur absolue) imposent la dynamique du système puisque les pôles dont la partie réelle négative est plus grande (en valeur absolue) conduisent à une convergence très rapide du terme exponentielle et, ont donc une contribution très faible sur le comportement du système. Une règle de conception

couramment utilisée fixe la valeur de la partie réelle des pôles faibles à 5 fois la valeur des pôles associés aux modes dominants (Nise, 2011) afin de pouvoir les négliger.

Par ailleurs, il peut être intéressant de regarder ce qu'il devient des résultats précédents lorsque l'on travaille dans un environnement numérique. Ainsi le traitement numérique implique la discrétisation qui passe par une transformée en Z . Le passage de la variable s à la variable z de l'espace des Z est donné pour une période d'échantillonnage T_E par :

$$z = e^{sT_E} \quad (2.32)$$

La convergence et la notion de stabilité impose que $z < 1$. En d'autres termes, les pôles des fonctions de transfert exprimés en Z sont compris à l'intérieur du cercle unité. Rappelons que le plan des Z est un plan imaginaire et la relation précédente (2.32) permet d'illustrer cela puisque s est une variable imaginaire et que l'exponentielle se divise en deux termes :

$$\begin{aligned} |z| &= r \\ \arg(z) &= \phi \end{aligned} \quad (2.33)$$

Avec

$$\begin{aligned} r &= e^{-\sigma T_E} \\ \phi &= \omega_d T_E \end{aligned} \quad (2.34)$$

La Figure 2.4 présente les lieux des pôles dans le plan des Z . Les positions indiquées correspondent aux équivalents de la Figure 2.2

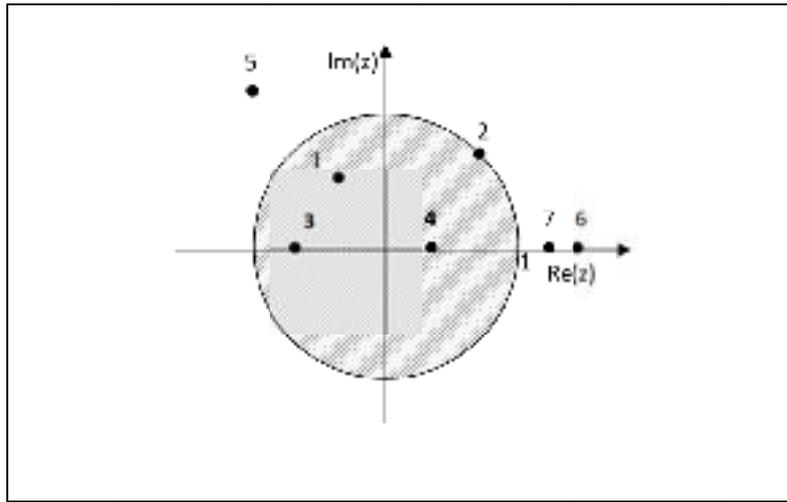


Figure 2.4 Positions des pôles dans le plan des Z

2.3 Contrôleur Proportionnel-Intégral-Dérivé (PID)

Les contrôleurs Proportionnel-Intégral-Dérivé sont le type de contrôleurs largement répandu dans l'industrie (Besançon-Voda et Gentil, 1999), (Åström et Hägglund, 2001), (Chan, Moallem et Wang, 2007). Ce type de régulateurs allie simplicité et performance (Besançon-Voda et Gentil, 1999).

2.3.1 Descriptif du PID

Le principe du correcteur PID est d'agir sur l'erreur (formée par la différence entre la consigne et la sortie) au moyen de trois actions :

- une action proportionnelle s'exprimant par le gain k_p ;
- une action intégrale se manifestant par le gain k_i ;
- une action intégrale filtrée caractérisée par le gain k_d et le paramètre du filtre f_d .

On peut alors donner l'expression de la forme parallèle d'un tel contrôleur dans le domaine de Laplace :

$$C(s) = k_p + \frac{k_i}{s} + \frac{k_d f_d s}{s + f_d} \quad (2.35)$$

$$\begin{aligned} C(s) &= \frac{(k_p(s + f_d)s + k_i(s + f_d) + k_d f_d s)}{s(s + f_d)} \\ &= \frac{k_p s^2 + (k_p f_d + k_i + k_d f_d)s + k_i f_d}{s(s + f_d)} \end{aligned}$$

Ainsi un correcteur PID *via* ses trois actions permet de régler les performances (temps de réponse, amortissement) d'un système, de type deuxième ordre, en boucle fermée (Besançon-Voda et Gentil, 1999). Un filtre sur l'action dérivée est mis en place afin d'avoir un transfert causal ce qui dans la pratique s'exprime par une atténuation du bruit de l'erreur. En effet, en dérivant un signal bruité on obtiendrait un signal essentiellement composé de bruit.

2.3.2 Discrétisation du contrôleur PID

Souvent, nous l'avons vu le processus à contrôler se place dans le domaine continu, analogique ou les grandeurs physiques des signaux sont réelles (appartenant à l'ensemble des réels \mathbb{R}). Ainsi, il faut prendre garde lors de la conception d'un correcteur numérique de la présence des convertisseurs analogique/numérique et numérique/analogique.

La méthodologie à suivre est la suivante dans le cas où l'on connaît le transfert du processus à contrôler (Besançon-Voda et Gentil, 1999) :

- définir une nouvelle fonction de transfert du processus en prenant en compte les convertisseurs (domaine de Laplace) ;
- calculer le PID en fonction des spécifications (domaine de Laplace) ;
- implémenter le PID numériquement.

Dans notre travail nous allons uniquement nous concentrer sur le dernier point. En effet, nous posons pour cela deux hypothèses :

- qu'une fonction de transfert du processus à contrôler est connue dans le domaine de Laplace ;
- que les coefficients du correcteur PID sont connus.

Dans la pratique, nous nous basons sur le travail de (Gagnon, 2011) qui a démontré dans ses travaux de maîtrise la puissance de la modélisation acausale (hypothèse 1 : le modèle est connu) et a pu obtenir grâce au logiciel Matlab/Simulink des paramètres de PID conformes aux spécifications requises (hypothèse 2 : les coefficients sont connus).

Il nous reste alors l'implémentation du PID numérique. On peut, pour cela utiliser l'approximation des différences avant (*forward difference* dans la littérature anglaise) dont on résume le principe concisément.

Le calcul d'une intégrale selon les différences avant est illustré par la Figure 2.5 et s'exprime comme :

$$I_{kT_E} = I_{(k-1)T_E} + T_E Y_{(k-1)T_E} \quad (2.36)$$

$$I(z) = z^{-1}(I(z) + T_E Y(z))$$

$$I(z) = \frac{T_E}{z-1} Y(z)$$

Ainsi le lien entre la variable s de Laplace et la discrétisation est :

$$s = \frac{z-1}{T_E} \quad (2.37)$$

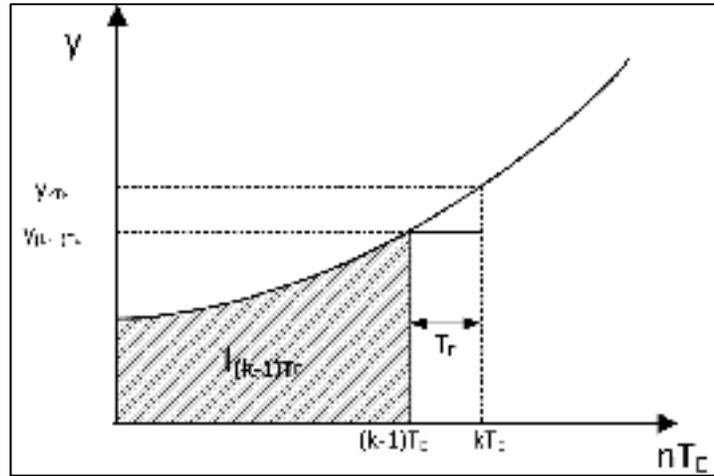


Figure 2.5 Principe d'intégration par les différences avant

Ainsi, le transfert du correcteur PID numérique associé est :

$$C(z) = k_p + \frac{k_i T_E}{z-1} + \frac{k_d f_d (z-1)}{T_E \left(\frac{z-1}{T_E} + f_d \right)} \quad (2.38)$$

$$C(z) = k_p + \frac{k_i T_E}{z-1} + \frac{k_d f_d z - k_d f_d}{z-1 + f_d T_E} \quad (2.39)$$

En mettant au même dénominateur on a :

$$C(z) = \frac{k_p (z-1)(z-1 + f_d T_E) + k_i T_E (z-1 + f_d T_E) + k_d f_d z - k_d f_d}{(z-1)(z-1 + f_d T_E)} \quad (2.40)$$

$$C(z) = \frac{k_p z^2 + (k_p f_d T_E - 2k_p + k_i T_E + k_d f_d) z + \dots}{(z-1)(z-1 + f_d T_E)} \quad (2.41)$$

$$\underline{\underline{\dots k_p + k_i f_d T_E^2 - k_p f_d T_E - k_i T_E - k_d f_d}}$$

On peut identifier l'expression précédente à un transfert plus général :

$$C = \frac{az^2 + bz + c}{z^2 + dz + e} \quad (2.42)$$

Il est alors relativement aisé d'identifier les paramètres a, b, c, d, e :

$$a = k_p \quad (2.43)$$

$$b = k_p f_d T_E - 2k_p + k_i T_E + k_d f_d \quad (2.44)$$

$$c = k_p + k_i f_d T_E^2 - k_p f_d T_E - k_i T_E - k_d f_d \quad (2.45)$$

$$d = f_d T_E - 2 \quad (2.46)$$

$$e = 1 - f_d T_E \quad (2.47)$$

On constate, à partir de l'équation (2.41), la présence d'un pôle en 1 et d'un second pôle réglé par le coefficient du filtre sur l'action dérivée et par la période d'échantillonnage. Notons p_1, p_2 les deux pôles tels que:

$$p_1 = 1 - f_d T_E$$

$$p_2 = 1$$

Soit un contrôleur C disposant de deux pôles $p_1 = 1$ et p_2 et disposant de deux zéros, z_1, z_2 .

Soit un processus G soumis au régulateur C tel que :

$$G = \frac{G_{num}}{G_{den}}$$

Alors d'après le théorème de la valeur finale (de la transformée en Z), le régime permanent de la réponse indicielle (entrée R , un échelon) du système en boucle fermée H est :

$$y(\infty) = \lim_{z \rightarrow 1} \left(\frac{z-1}{z} \right) H(z)R(z)$$

$$\begin{aligned}
&= \lim_{z \rightarrow 1} \left(\frac{z-1}{z} \right) H(z) \left(\frac{z}{z-1} \right) \\
&= \lim_{z \rightarrow 1} H(z) \\
&= \lim_{z \rightarrow 1} \left(\frac{\frac{(z-z_1)(z-z_2)G_{num}}{(z-1)(z-p_2)G_{den}}}{1 + \frac{(z-z_1)(z-z_2)G_{num}}{(z-1)(z-p_2)G_{den}}} \right) \\
&= \lim_{z \rightarrow 1} \left(\frac{\frac{(z-z_1)(z-z_2)G_{num}}{(z-1)(z-p_2)G_{den}}}{\frac{(z-1)(z-p_2)G_{den} + (z-z_1)(z-z_2)G_{num}}{(z-1)(z-p_2)G_{den}}} \right) \\
&= \lim_{z \rightarrow 1} \left(\frac{(z-z_1)(z-z_2)G_{num}}{(z-1)(z-p_2)G_{den}} \times \frac{(z-1)(z-p_2)G_{den}}{(z-1)(z-p_2)G_{den} + (z-z_1)(z-z_2)G_{num}} \right) \\
&= \lim_{z \rightarrow 1} \left(\frac{(z-z_1)(z-z_2)G_{num}}{(z-1)(z-p_2)G_{den} + (z-z_1)(z-z_2)G_{num}} \right) \\
&= \lim_{z \rightarrow 1} \left(\frac{(z-z_1)(z-z_2)G_{num}}{(z-z_1)(z-z_2)G_{num}} \right) \\
&= 1
\end{aligned}$$

La position du pôle p_2 nous apprend donc qu'il n'y a pas d'erreur statique sur un système muni d'un PID filtré.

On peut alors écrire la loi de commande générale à partir de l'expression (2.42) :

$$U(z) = aE(z) + bz^{-1}E(z) + z^{-2}cE(z) - z^{-1}dU(z) - z^{-2}eU(z) \quad (2.48)$$

On est alors en mesure d'écrire l'équation aux récurrences suivante :

$$u(k) = ae(k) + be(k - 1) + ce(k - 2) - du(k - 1) - eu(k - 2) \quad (2.49)$$

L'expression précédente (2.49) donne la formule générale d'un régulateur PID discret. Il reste cependant possible d'en donner une autre méthode d'implémentation, à partir de l'équation (2.39) :

Posons :

$$P(z) = k_p E(z) \quad (2.50)$$

$$I(z) = \frac{k_i T_E}{z - 1} E(z) \quad (2.51)$$

$$D(z) = \frac{k_d f_d z - k_d f_d}{z - 1 + f_d T_E} E(z) \quad (2.52)$$

De telle sorte que :

$$U(z) = P(z) + I(z) + D(z) \quad (2.53)$$

On obtient alors trois expressions aux récurrences pour chacune des actions P, I et D.

$$P(k) = k_p e(k) \quad (2.54)$$

$$I(k) = k_i T_E e(k - 1) + I(k - 1) \quad (2.55)$$

$$D(k) = k_d f_d e(k) - k_d f_d e(k - 1) + (1 - f_d T_E) D(k - 1) \quad (2.56)$$

Ce type d'implémentation permet une plus grande flexibilité dans le contrôle puisque l'on profite réellement de l'architecture parallèle du régulateur.

2.4 Commande dans l'espace d'état

Avant de développer les principes de la commande moderne, il convient d'effectuer un bref rappel sur les systèmes d'état. En effet, le cadre choisi pour développer la commande moderne s'inscrit dans l'espace d'état qui permet de s'assurer d'un cadre théorique largement connu dans la littérature, permettant un meilleur formalisme pour une meilleure compréhension (Rugh, 1991), (Bensoussan, 2008). Comme nous l'avons précédemment fait, nous déroulerons d'abord des rappels dans le domaine continu avant de nous focaliser sur notre domaine d'intérêt, à savoir le domaine discret.

2.4.1 Représentation de systèmes dans l'espace d'état

Il est possible de représenter un système régi par des équations différentielles à coefficients constants par le système suivant :

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \quad (2.57)$$

Où

x est le vecteur de dimension n constitué des variables d'état

u est le vecteur d'entrée de dimension p

y est le vecteur de sortie de dimension r

A est une matrice de taille $n \times n$

B est une matrice de taille $n \times p$

C est une matrice de taille $r \times n$

D est une matrice de taille $r \times p$ qui n'agit pas sur la dynamique du système. $D = 0$.

Le système décrit par l'équation (2.57) représente un système linéaire, autrement dit, les éléments des matrices sont constants dans le temps. La Figure 2.6 donne à voir une représentation typique d'un système dans l'espace d'état.

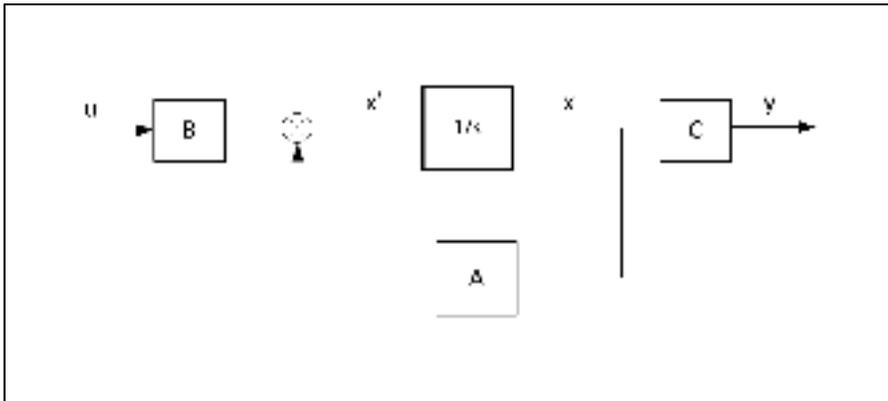


Figure 2.6 Représentation d'état d'un système (A,B,C)

On peut toutefois appliquer la représentation précédente aux systèmes décrits par des équations différentielles non linéaires en linéarisant autour de points de fonctionnement.

A partir de l'équation (2.57), écrivons :

$$\dot{x} = f(x, u) \quad (2.58)$$

Nous avons alors (Bensoussan, 2008), (Gauthier, 2010) :

$$\Delta \dot{x} = \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial u} \Delta u \quad (2.59)$$

Par ailleurs, on est en mesure de donner la fonction de transfert du système décrit par la représentation d'état (transfert de l'entrée vers la sortie) (ANNEXE II) :

$$\frac{Y(s)}{U(s)} = G(s) = C(sI - A)^{-1}B + D \quad (2.60)$$

Les pôles de la fonction de transfert sont les solutions de l'équation caractéristique définissant les valeurs propres de la matrice A . Nous nous trouvons dans un cas similaire à celui évoqué dans la partie 2.2 et un système (A, B, C, D) est stable si les valeurs propres de

la matrice A ont des parties réelles négatives (pour assurer la convergence de l'exponentielle dans le domaine temporel).

Notons, en outre, qu'il est possible de donner une représentation dans l'espace d'état d'un système décrit par sa fonction de transfert. Bensoussan (2008) et Nise (2011) le rappellent : Soit un procédé décrit par l'équation (2.16), que l'on peut réécrire de façon normalisée par :

$$G(s) = \frac{\sum_{j=0}^m \tilde{a}_j s^j}{\sum_{i=0}^n \tilde{b}_i s^i} \quad (2.61)$$

Où

$$\tilde{a}_j = \frac{a_j}{b_n}, \text{ pour } j \in \llbracket 0 ; m \rrbracket \quad (2.62)$$

$$\tilde{b}_i = \frac{b_i}{b_n}, \text{ pour } i \in \llbracket 0 ; n \rrbracket$$

Alors sa représentation dans l'espace d'état est donné par (dont le développement est présenté en ANNEXE II) :

$$A = \begin{pmatrix} -\tilde{b}_{n-1} & -\tilde{b}_{n-2} & \cdots & \tilde{b}_1 & \tilde{b}_0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \quad (2.63)$$

$$B = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad (2.64)$$

$$C = (\tilde{a}_m \quad \tilde{a}_{m-1} \quad \cdots \quad \tilde{a}_0) \quad (2.65)$$

$$D = 0 \quad (2.66)$$

Il existe encore deux autres propriétés fondamentales pour l'étude des systèmes d'état : la commandabilité et l'observabilité.

- La commandabilité : un système est commandable s'il existe une commande u permettant à partir d'un état initial $x(t_0)$ d'aboutir à un état désiré $x(t_1)$. Le critère associé est le suivant : Soit la matrice de commandabilité \mathbb{C} de taille $n \times n.p$ définie par :

$$\mathbb{C} = (B \ AB \ \dots \ A^{n-1}B) \quad (2.67)$$

Alors le système (A, B, C, D), à n variables d'état, est commandable si :

$$\text{rang}(\mathbb{C}) = n \quad (2.68)$$

- L'observabilité : un système est observable au temps t_0 si l'état associé $x(t_0)$ peut être connu à partir de la sortie $y(t_1)$, avec $t_1 \geq t_0$. Le critère d'observabilité est le suivant : Soit la matrice d'observabilité \mathbb{O} de taille $r.n \times n$ définie par :

$$\mathbb{O} = \begin{pmatrix} C \\ CA \\ \dots \\ CA^{n-1} \end{pmatrix} \quad (2.69)$$

Alors le système (A, B, C, D) est observable si :

$$\text{rang}(\mathbb{O}) = n \quad (2.70)$$

Notons qu'un système décrit par les équations (2.63)-(2.66) dispose d'une matrice de commandabilité de rang n . La démonstration est présentée en ANNEXE II.

Par ailleurs, le système ABCD présenté par l'expression (2.62) a une solution. On décompose la solution en deux solutions distinctes : la solution homogène (pour une entrée nulle) et la solution générale.

La solution homogène se présente sous la forme d'une exponentielle de matrice tel que :

$$x(t) = e^{At}x(0) \quad (2.71)$$

La solution générale quant à elle s'exprime par :

$$x(t) = e^{At}x(0) + C \int_0^t e^{A(t-t')}Bu(t')dt' \quad (2.72)$$

A présent, regardons les systèmes d'états discrets. Ceux-ci peuvent s'écrire comme suit :

$$\begin{cases} x(k+1) = A_d x(k) + B_d u(k) \\ y(k) = C_d x(k) + D_d u(k) \end{cases} \quad (2.73)$$

Avec un système initial continu composé des matrices A, B, C et D. Les relations entre les matrices continues et discrètes sont les suivantes (ANNEXE II) :

$$A_d = \mathcal{L}^{-1}[(sI - A)^{-1}]_{t=T_E} \quad (2.74)$$

$$B_d = \mathcal{L}^{-1} \left[\frac{1}{s} (sI - A)^{-1} B \right]_{t=T_E}$$

$$C_d = C$$

$$D_d = D$$

T_E : la période d'échantillonnage

On peut également présenter une équivalence entre les matrices continues et discrètes par une approximation. Si la période d'échantillonnage T_E est suffisamment petite il est possible d'écrire d'après Taylor :

$$A_d \cong I + AT_E \quad (2.75)$$

$$B_d \cong T_E B$$

Ces résultats sont très pratiques car ils permettent de ne pas utiliser de transformées inverses de Laplace, ni de calculer d'exponentielles de matrices. Pour plus de détails sur cette dernière approximation, vous pouvez vous référer à l'ANNEXE II. Notons que le logiciel Matlab

utilise l'algorithme de Van Loan (1978) qui permet d'utiliser des opérations matricielles élémentaires à partir de matrices triangulaires.

Pour les systèmes discrets, les formules (2.67) à (2.70) sont identiques, il suffit de remplacer A par A_d et B par B_d .

Au final, toutes les relations énoncées permettent la description d'un système dans l'espace d'état. Cependant, il nous reste à voir comment élaborer une stratégie de commande dans une représentation d'état.

2.4.2 Commande dans l'espace d'état

Afin de contrôler les processus décrits dans l'espace d'état, on peut utiliser un retour d'état, il s'agit de boucler le système afin d'en contrôler sa dynamique à l'aide d'un gain K . La Figure 2.7 présente un tel système.

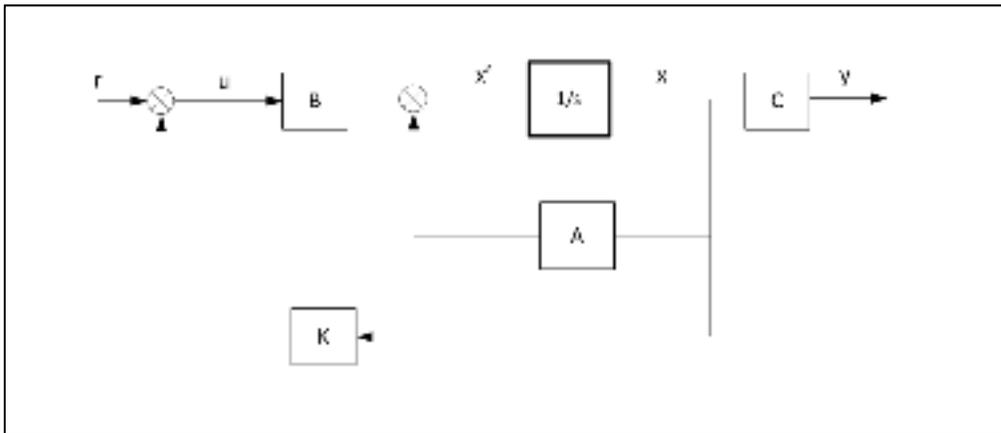


Figure 2.7 Représentation d'état avec retour d'état

Ainsi, en bouclant on agit sur la commande u passée *via* le gain K et le système (2.57), qui doit être commandable, devient :

$$\begin{cases} \dot{x} = (A - BK)x + Br \\ y = Cx + Du \end{cases} \quad (2.76)$$

On dispose alors d'une nouvelle matrice $\mathcal{A} = A - BK$ qui permet *via* le choix du vecteur K de spécifier une dynamique (par une stratégie de placement de pôles, puisque l'on cherche à régler les valeurs propres de la matrice \mathcal{A}). Cependant dans la pratique, les états d'un système ne sont pas toujours connus. C'est pourquoi on peut mettre en place un observateur d'état (ou estimateur) qui permet d'approximer les états afin de pouvoir appliquer une commande par retour d'état, par exemple.

2.4.2.1 Observateur d'état

Souvent, les états ne sont pas directement accessibles mais nécessitent une reconstitution à partir des signaux d'entrée et de sortie. L'observateur d'état tente donc de reformer les états qui ne sont pas directement mesurables. C'est pourquoi on parle aussi d'estimateur d'état. En effet, une nouvelle variable de travail apparaît \hat{x} qui représente l'estimation de l'état x . Si on est capable d'estimer correctement les états d'un système alors on peut appliquer un retour d'état directement à partir des estimations comme l'indique la Figure 2.8 . Notons qu'un système ne peut se doter d'un observateur que s'il est observable, autrement dit si l'on est capable de connaître un état à un instant t à partir d'une sortie à un instant t' , avec $t < t'$.

L'équation d'un observateur est :

$$\dot{\hat{x}} = A\hat{x} + Bu + H(y - C\hat{x}) \quad (2.77)$$

$$\dot{\hat{x}} = (A - HC)\hat{x} + Bu + Hy \quad (2.78)$$

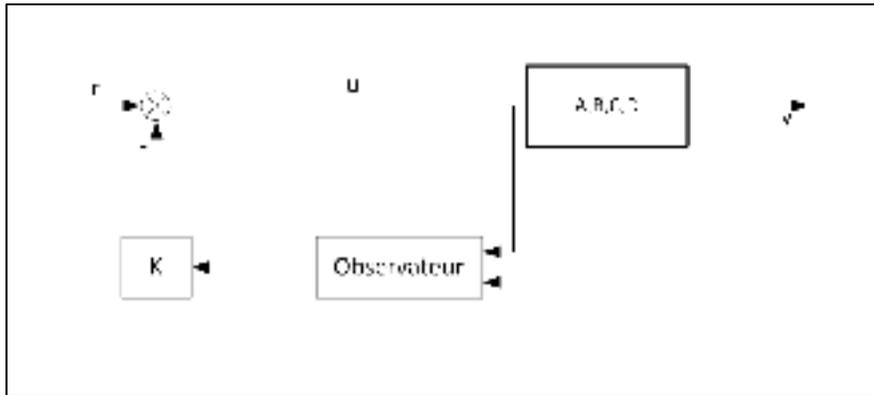


Figure 2.8 Système de commande dans l'espace d'état muni d'un observateur

Nous l'avons vu, nous souhaitons disposer de bonnes estimations des états \hat{x} . Pour cela, il est possible de définir l'erreur e telle que :

$$e = x - \hat{x} \quad (2.79)$$

On peut également s'intéresser à la dynamique de l'erreur définie par :

$$\dot{e} = \dot{x} - \dot{\hat{x}} \quad (2.80)$$

En injectant les équations (2.57) et (2.78) dans l'équation (2.80) on a

$$\dot{e} = (A - HC)e \quad (2.81)$$

On constate alors que la dynamique de l'erreur est dépendante des valeurs propres de la matrice $(A - HC)$. Cette dernière, via H permet d'ajuster la dynamique de convergence de l'erreur. Notons que l'on souhaite une convergence très rapide par rapport à celle des retours d'état par exemple. En effet, une convergence très rapide de l'erreur implique que l'estimation est rapidement correcte. Ainsi le retour d'état u_K s'applique sur de bonnes valeurs et la commande est plus fiable. La commande u_K s'écrit :

$$u_K = K\hat{x} \quad (2.82)$$

Il nous reste à montrer comment mettre en pratique la construction de l'observateur. On peut considérer l'équation (2.78) comme une représentation d'état. On peut poser que :

$$\dot{\hat{x}} = A\hat{x} + BY \quad (2.83)$$

Avec

$$A = A - HC \quad (2.84)$$

$$B = [B \quad H]$$

$$Y = \begin{bmatrix} u \\ y \end{bmatrix}$$

Une vue de l'implémentation est donnée par la Figure 2.9.

À présent, nous nous proposons de décrire la mise en place d'un observateur discret. Il suffit de reprendre les expressions précédentes en les indiquant :

$$\hat{x}(k+1) = (A_d - HC_d)\hat{x}(k) + B_d u + Hy \quad (2.85)$$

Où H est une matrice choisie afin de faire converger rapidement l'erreur d'estimation. Dans l'espace en Z , il convient alors de faire en sorte que le polynôme caractéristique de $(zI - (A_d - HC_d))$ ait des solutions dans le plan de convergence de Z et si possible que celles-ci soit proches de 0.

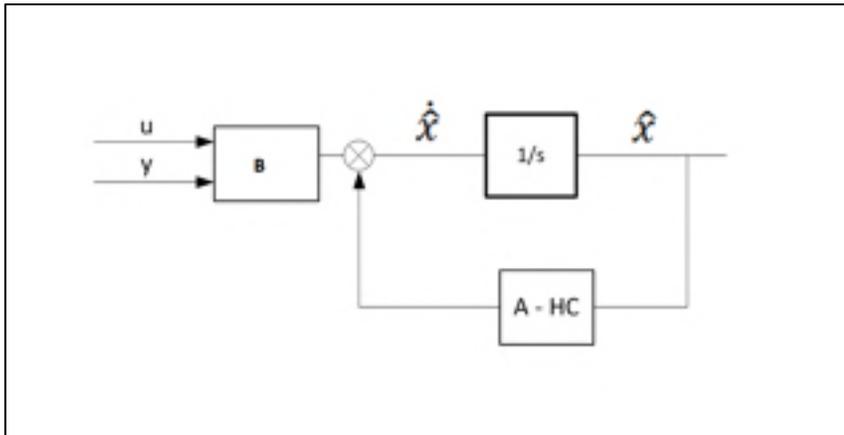


Figure 2.9 Construction d'un observateur

2.4.2.2 Régime établi

L'équation (2.57) décrit la dynamique d'un système. En effet, le terme dérivé, \dot{x} , implique la notion de variation, de dynamique. En régime établi on souhaite avoir en sortie la valeur passée en consigne.

Ecrivons le système (2.57) en régime établi, autrement dit pour un système qui ne varie pas dans le temps :

$$\begin{aligned} 0 &= Ax + Br & (2.86) \\ y_{\infty} &= Cx \end{aligned}$$

On peut exprimer le vecteur d'état par :

$$x = -A^{-1}Br \quad (2.87)$$

La sortie en régime établi, y_{∞} , s'exprime :

$$y_{\infty} = -CA^{-1}B r \quad (2.88)$$

Ainsi si l'on souhaite disposer de la valeur y_∞ en sortie, il faut avoir comme consigne :

$$r = (-CA^{-1}B)^{-1} y_\infty \quad (2.89)$$

La matrice A présente dans les équations (2.86)-(2.89) peut être remplacée par la matrice $(A - BK)$ dans le cas d'un retour d'état.

CHAPITRE 3

TRAITEMENT NUMÉRIQUE

3.1 La gestion numérique des nombres

Les ingénieurs utilisent, aujourd'hui, les systèmes numériques pour simuler des systèmes qui seront implantés, après validation, dans le monde analogique. Ainsi, ceux-ci ont besoin de modéliser un environnement analogique, décrit par des nombres réels (au sens de l'appartenance au corps des réels, noté \mathbb{R}) à l'aide de système numérique qui (pour l'heure) n'utilise toujours que des 0 et des 1.

Les nombres peuvent être codés selon deux représentations : en virgule flottante et en point fixe. Une représentation à point fixe, en numérique, sur N bits est donnée par la Figure 3.1.

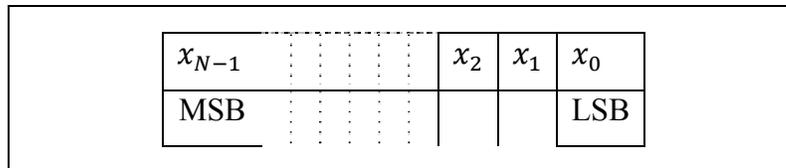


Figure 3.1 Représentation numérique à point fixe sur N bits

Les valeurs des bits $(x_i)_{0 \leq i \leq N-1}$ de la Figure 3.1 sont comprises en 0 et 1.

Il existe deux types de nombres ainsi codés les signés et non signés (en anglais : *signed* et *unsigned*). La différence entre ces deux types réside dans le fait que l'on tient en compte le signe du nombre ou non. Afin de tenir compte du signe, sans disposé d'un bit supplémentaire (ou bit de signe), on utilise le complément à deux.

Ainsi : $0010\ 1101|_2 = 45|_{10}$ « devient » $1101\ 0011|_2 = -45|_{10}$ (ou $211|_{10}$ en non signé) ; où $x|_B$ indique la valeur du nombre x dans la base B . Tout cela représente une vision simple puisqu'on ne tient pas compte de la présence de la virgule.

La mise à l'échelle est alors un passage obligé afin d'éviter des dépassements (*Overflow* et *Underflow*) et des erreurs de quantification. La boîte à outils *Simulink Fixed-Point* permet à l'utilisateur de choisir l'échelle en fonction de la virgule ou selon toute autre échelle qu'il jugera pertinente.

Les nombres à virgules fixes sont représentés de manière affine :

$$x \cong \tilde{x} = aq + B \quad (3.1)$$

x : représente la valeur réel

\tilde{x} : représente la valeur réel approximée

a : représente la pente (coefficient directeur de la loi affine)

B : représente l'ordonnée à l'origine

q : représente la valeur stockée codant x

La pente a se décompose en un facteur d'ajustement, F , compris entre 1 et 2 et un facteur décrit par une puissance de deux, 2^E . Cette puissance négative de deux (E) représente le nombre de chiffre à la droite de la virgule telle que : $a = F \cdot 2^E$

Lors de la quantification, on représente x en non signé tel que :

$$x = a \cdot \left(\sum_{i=0}^{N-1} x_i 2^i \right) + B \quad (3.2)$$

Un nombre signé se représente :

$$x = a \cdot \left(-x_{N-1} 2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i \right) + B \quad (3.3)$$

Exemple :

1	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

Prenons $F = 1$, $B = 0$ et cherchons à représenter un *non-signé*.

$$\begin{aligned} x &= 2^{-3}(2^7 + 2^5 + 2^4 + 2^2 + 2^0) \\ &= 22,625 \end{aligned}$$

Si l'on souhaite à présent traduire la valeur définie sur l'octet par un *signé*:

$$\begin{aligned}
 x &= 2^{-3}(-2^7 + 2^5 + 2^4 + 2^2 + 2^0) \\
 &= -9,375
 \end{aligned}$$

Le complémentaire à deux de l'octet présenté est :

0	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---

$$\begin{aligned}
 x &= 2^{-3}(0 \times (-2^7) + 2^6 + 2^3 + 2^1 + 2^0) \\
 x &= 9,375
 \end{aligned}$$

Dans la suite de ce travail, nous considérerons les nombres signés exclusivement.

Nous prendrons également $B = 0$ et $a = F \cdot 2^E$ avec $F = 1$ on a :

$$x = 2^E \left(-x_{N-1} 2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i \right) \quad (3.4)$$

L'équation (3.4) donne une représentation générale des nombres signés. Pour des raisons de commodité, imaginons que le bit de poids fort (celui affecté du signe «-») n'est plus indicé (N-1) mais (0) ce qui constitue une représentation tout aussi correcte. Nous aurions :

$$x = 2^E \left(-x_0 2^{N-1} + \sum_{i=1}^{N-1} x_i 2^{(N-1)-i} \right) \quad (3.5)$$

Pour faciliter les développements théoriques futurs, on choisit de travailler avec un nombre disposant de $(N - 1)$ termes après la virgule, soit :

$$-1 \leq x < 1 \quad (3.6)$$

Cela implique $E = -(N - 1)$ si bien que :

$$x = 2^E \left(-x_0 2^{N-1} + \sum_{i=1}^{N-1} x_i 2^{(N-1)-i} \right) \quad (3.7)$$

$$\begin{aligned}
x &= 2^{-(N-1)} \left(-x_0 2^{N-1} + \sum_{i=1}^{N-1} x_i 2^{((N-1)-i)} \right) \\
&= -x_0 2^{N-1} \times 2^{-(N-1)} + \sum_{i=1}^{N-1} x_i 2^{((N-1)-i)} \cdot 2^{-(N-1)} \\
&= -x_0 2^0 + \sum_{i=1}^{N-1} x_i 2^{((N-1)-i)} \cdot 2^{-(N-1)} \\
x &= -x_0 + \sum_{i=1}^{N-1} x_i 2^{-i} \tag{3.8}
\end{aligned}$$

Le passage de l'expression (3.5) à l'expression (3.8) permet d'établir une base de raisonnement commune avec la littérature et notamment avec (White, 1989).

Toutefois, il est important de noter que la plage dynamique des nombres à virgules fixes est inférieure à celle des nombres à virgules flottantes (Mathworks, 2007). Cependant la représentation en virgule flottante n'est pas prise en compte par le synthétiseur de Xilinx pour les anciennes générations de FPGA (Xilinx, 2011a) et la structure même des FPGA n'est pas optimisée pour les opérations en virgules flottantes (Monmasson, 2007). C'est pourquoi nous avons choisi la représentation à point fixe qui permet également une économie de ressources.

3.2 Le traitement des opérations numériques

3.2.1 Opérations et fonctionnement

L'addition numérique peut se réaliser au moyen d'additionneurs complets. Pour calculer la somme de deux termes de n bits, on requiert n additionneurs. Un élément clef est la gestion de la retenue qui permet de propager les retenues, si elles existent, à travers les n étages. Toutefois, ce type d'additionneur est très lent. Une amélioration peut résider dans le fait

d'implémenter une gestion de retenue anticipée (CLA, pour *Carry-Lookahead Adder*) (Hennessy et Patterson, 1995). Le guide utilisateur du FPGA Spartan 3A de la firme Xilinx (2011b) propose le détail du fonctionnement des ressources dédiées à l'arithmétique. Une gestion CLA permet de déterminer si le signal de retenue doit être propagé ou généré selon le Tableau 3.1.

Tableau 3.1 Addition avec retenue anticipée

Entrée A	Entrée B	Propagé (P)	Généré (G)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

L'implémentation de la fonction logique s'effectue alors avec un multiplexeur et deux portes XOR. Le multiplexeur est piloté par le signal P. Si $P = 0$, alors $Cout = A + B$ (A OU B). Dans le cas, où $P = 1$, le signal de retenue d'entrée (en provenance d'un bit de poids plus faible) est propagé vers Cout. L'implémentation présente dans un FPGA de type Spartan 3 est présentée sur la Figure 3.2.

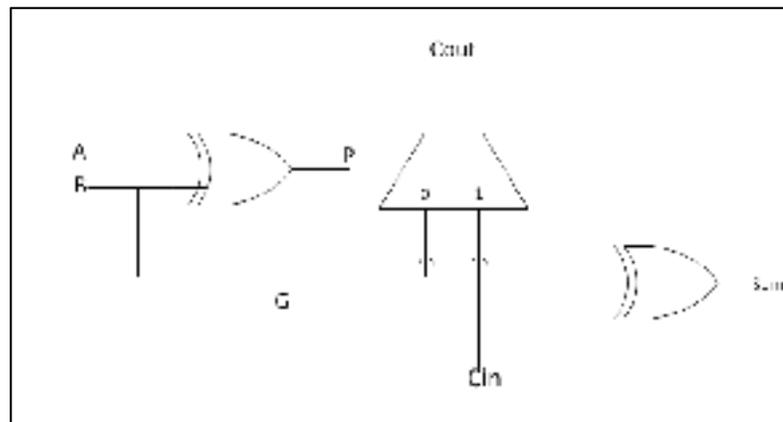


Figure 3.2 Implémentation d'une gestion CLA dans un FPGA de type Spartan-3
Tirée de Xilinx (2011, p. 277)

Notons que le multiplexeur et la porte XOR de sortie (vers Sum) sont réalisés à l'aide d'une circuiterie dédiée à la chaîne de gestion de la retenue et ne nécessite pas les ressources présente dans une LC (LUT, multiplexeur et bascule). Ce type d'implémentation est très efficace car il bénéficie d'une architecture dédiée et les temps indiqués pour réaliser une addition sont indiqués par la Figure 3.3.

Si l'addition est optimisée au sein des FPGA que ce soit en terme de ressources (via circuit dédié CLA) permettant d'obtenir des fréquences de calcul importantes (cf. Figure 3.3), il est intéressant de regarder comment la multiplication est implémentée.

La multiplication binaire est très simple puisqu'une porte ET suffit. Cependant, il est rare que l'on souhaite effectuer une multiplication de deux mots de 1 bit. Ainsi pour gérer les multiplications plus complexes, les FPGA utilisent des produits partiels (ce qui revient à multiplier tous les bits d'un opérande par ceux de l'autre facteur. On se retrouve alors avec un nombre conséquent de portes logique ET, afin d'effectuer chaque multiplication bit à bit, et d'additionneurs qui somment les résultats partiels. D'ailleurs, le guide d'utilisateur (Xilinx, 2011b) rappelle qu'une multiplication de n bits par m bits requiert (conformément à la description de la Figure 1.2) :

$$n \times m \text{ LUT} \quad (3.9)$$

ou encore

$$\frac{n \times m}{2} \text{ Tranches} \quad (3.10)$$

Toutefois, il est possible de réduire le nombre de tranches occupées par deux en combinant au sein d'une même LUT une porte ET (de la multiplication) avec la première porte XOR de l'additionneur représenté sur la Figure 3.2. Néanmoins, même si le nombre de LUT peut être réduit, la multiplication est une opération très gourmande en termes de ressources. En effet, Chan, Moallem et Wang (2004; 2007) et Chander, Agarwal et Gupta (2010) mettent en évidence, dans le cadre de l'implémentation d'un régulateur PID (respectivement dédié à la

commande d'un système de ventilation et à un convertisseur DC-DC), que l'utilisation de multiplications réalisées à l'aide des ressources arithmétiques consomme plus d'espace sur le FPGA qu'une implémentation réalisée par l'arithmétique distribuée (AD) tout en consommant plus d'énergie. Chan, Moallem et Wang (2007) quantifient le gain en terme d'espace occupé à hauteur de 80%.

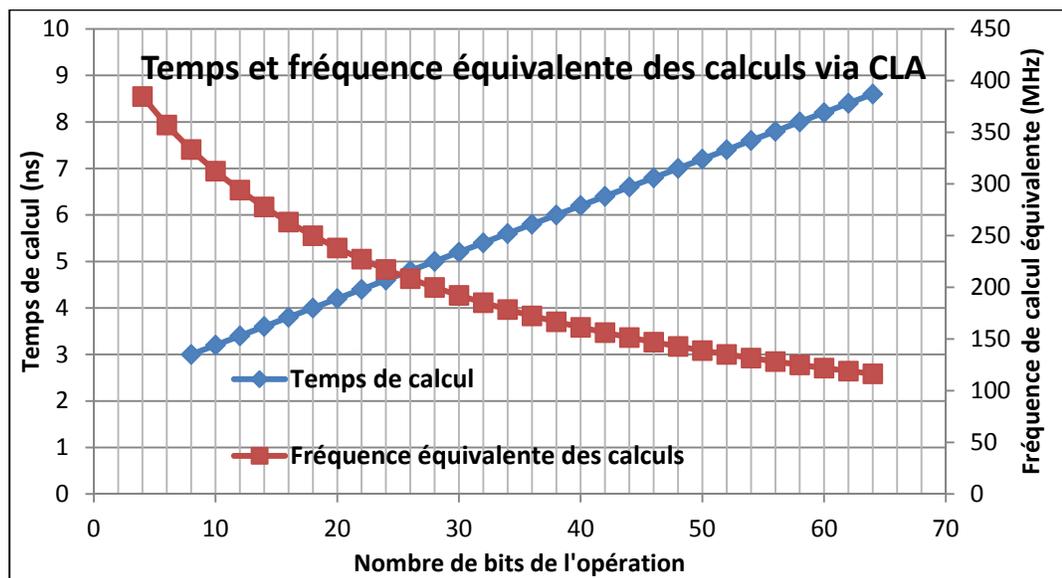


Figure 3.3 Temps et fréquence équivalente des calculs effectués via la gestion CLA
Adaptée de Xilinx (2011, p. 290)

3.2.2 Introduction à l'arithmétique distribuée

Nous venons de voir que la multiplication constitue un défi important à relever pour les systèmes numériques. Ce défi comporte deux enjeux majeurs qui sont l'espace nécessaire à la réalisation de ce type d'opération et le temps de calcul. Une solution proposée au cours d'une conférence à la fin des années 1960 par Pelen et Liu a mis en évidence une méthode appelée Arithmétique Distribuée (AD) ou *Distributed Arithmetic*, en anglais). White (1989) propose un article détaillé sur le principe de l'AD en proposant une méthodologie appliquée aux filtres du deuxième ordre à réponse impulsionnelle infinie. L'AD traite d'une méthode efficace pour effectuer une somme de produits.

Soit la sortie, y , définie comme une somme de K-produits :

$$y = \sum_{k=1}^K A_k x_k \quad (3.11)$$

A_k : des coefficients fixes

x_k : le k-ième vecteur d'entrée défini en point fixe par l'équation (3.8) :

$$x_k = -x_{k0} + \sum_{i=1}^{N-1} x_{ki} 2^{-i} \quad (3.12)$$

On peut alors réécrire l'équation (3.11), à l'aide de l'équation (3.12) ce qui donne :

$$y = \sum_{k=1}^K A_k \left(-x_{k0} + \sum_{i=1}^{N-1} x_{ki} 2^{-i} \right) \quad (3.13)$$

En modifiant l'ordre des sommations, on trouve :

$$y = \sum_{i=1}^{N-1} \left(\sum_{k=1}^K A_k x_{ki} \right) 2^{-i} + \sum_{k=1}^K A_k (-x_{k0}) \quad (3.14)$$

Cette dernière expression représente le calcul par AD d'après White (1989).

3.2.3 Implémentation et taille mémoire

Si on se focalise à présent sur le terme :

$$\sum_{k=1}^K A_k x_{ki}$$

Celui-ci indique la somme sur K i.e. le nombre de vecteurs en présence pour un même bit (pour un bit i fixé). En d'autres termes, le terme x_{ki} représente le i -ème bit du vecteur x_k .

Nous l'avons vu chaque x_{ki} ne peut prendre par définition que des valeurs de 0 ou 1 (car représentant un bit). Par ailleurs, il y a 2^K valeurs possibles pour ce terme. Plutôt que de calculer successivement ces valeurs, la solution proposée par l'AD est de stocker les valeurs possible en ROM puis à l'aide d'un accumulateur de calculer au fur et à mesure les valeurs si bien qu'aux termes des N cycles nécessaires (i décrivant les valeurs de 0 à $N - 1$), nous trouvons le bon résultat.

N'oublions pas la présence du terme comprenant le bit de poids fort. Celui-ci étant un bit de signe, il faut doubler la taille de la ROM, soit 2×2^K . Pour 4 vecteurs d'entrée, on doit ainsi avoir une ROM de 32 mots.

On peut envisager de réduire par deux la taille mémoire nécessaire. Pour cela on peut définir un signal de contrôle T_s tel que celui-ci prenne la valeur 1 quand le bit de signe se présente et la valeur 0 le reste du temps. En utilisant le bit de signe (à savoir 0 ou 1) via le signal T_s on est en mesure de réduire par deux la taille de la ROM en utilisant un additionneur/Soustracteur (si $T_s = 1$, on soustrait, sinon, on additionne).

Il existe encore une autre façon de réduire la taille nécessaire en LUT, en partitionnant le système (Mathworks, 2007), (White, 1989). C'est cette méthode que nous utiliserons pour l'implémentation de l'algorithme du PID. En effet, en utilisant le principe des actions séparées en considérant notre correcteur comme purement parallèle nous verrons qu'il est possible de réduire le nombre nécessaire de LUT.

3.2.4 Quantification du gain de l'AD

Afin de connaître le gain obtenu en utilisant l'AD par rapport à une multiplication réalisée au sein du FPGA (sans multiplicateurs embarqués), nous avons tracé le nombre requis de LUT en fonction de la taille de la multiplication. Nous avons effectué des multiplications de $n \times n$.

La Figure 3.4 propose les résultats obtenus. On observe que la multiplication utilisant les ressources du FPGA consomme conformément à l'équation (3.9) n^2 LUT. L'AD propose une consommation réduite et linéaire des ressources.

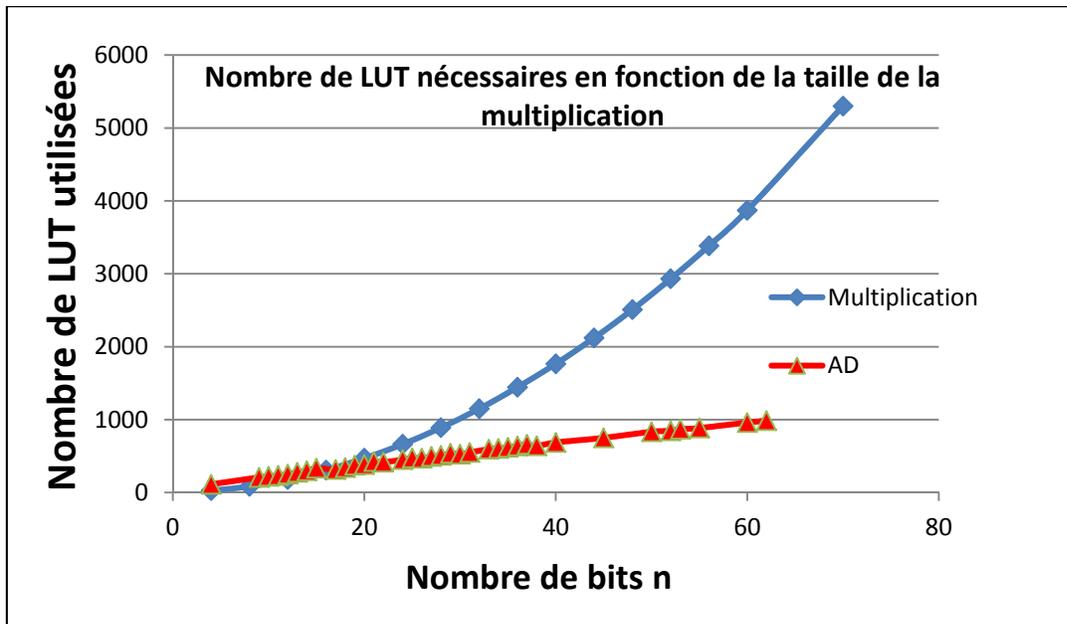


Figure 3.4 Utilisation des LUT en fonction de la taille de la multiplication

CHAPITRE 4

DESCRIPTION DU SYSTEME

Avant de rentrer dans des considérations techniques et précises de notre application, il convient de s'attarder sur le système à l'étude. Nous allons donc présenter le système dans son ensemble : du processus à commander au régulateur en passant par les circuits d'interfaçage. On peut, pour une compréhension générale de l'enjeu se référer à la Figure 2.1. Toutefois, cette figure ne rend pas compte de la réalité. Dans la pratique, il faut ajouter plusieurs interfaces au modèle élémentaire. En effet, il faut tenir compte de la présence des convertisseurs Analogique/Numérique (CAN) et Numérique/Analogique (CNA) ainsi que du gain du capteur (K_{pot}) et des circuits d'adaptation (CA1 et CA2) des niveaux de tension. La Figure 4.1 donne à voir une modélisation plus complète et plus fidèle de la réalité.

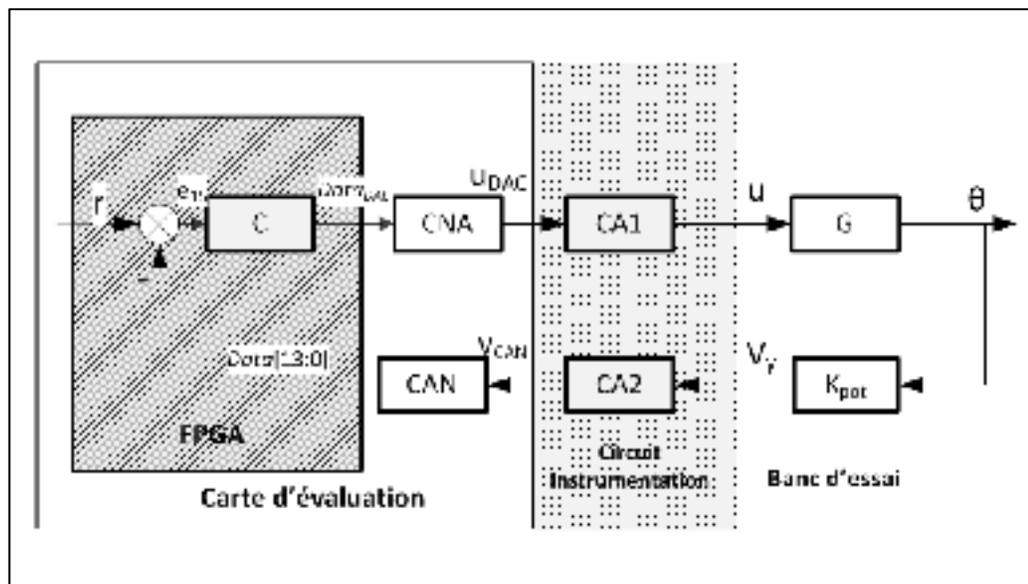


Figure 4.1 Boucle de commande globale

Ainsi on retrouve le système de base avec un régulateur (C) et un procédé (G) en tenant compte des différents traitements opérés sur les signaux. On distingue également où se situent physiquement les éléments de la boucle de commande. Le régulateur est implémenté

dans le FPGA, comme la consigne qui est une constante programmée directement dans le FPGA (ou *Hard Coded*). La consigne subit elle aussi un traitement. En effet, la sortie et la consigne doivent être homogènes.

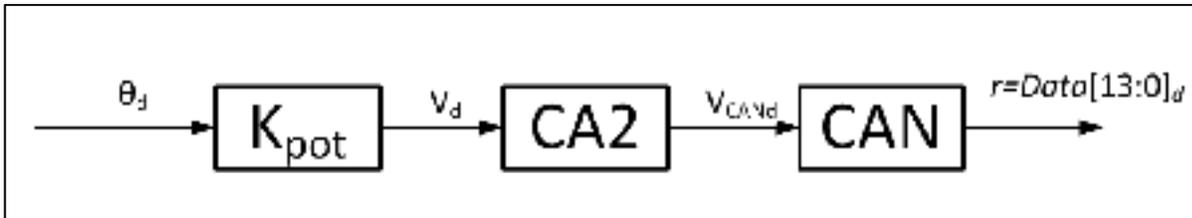


Figure 4.2 Traitement de la consigne

La Figure 4.2 illustre le traitement subi par la consigne afin de disposer d'un signal d'erreur cohérent pouvant être traité numériquement.

Le terme carte d'évaluation sur la Figure 4.1 représente la carte à notre disposition comprenant le FPGA ainsi que plusieurs modules d'entrées/sorties (E/S) tels les CAN et CNA. Notons que le circuit d'instrumentation ne sera pas traité dans le corps du texte mais en ANNEXE III. Maintenant que nous avons connaissance du système dans sa globalité, il nous reste à le décrire plus en détail.

4.1 Le processus à commander (Banc d'essai)

Le processus, que nous cherchons à commander, est un système électro-hydraulique. Celui-ci a été décrit en précision par Gagnon (2011). Il s'agit d'un bras oscillant piloté par un vérin hydraulique linéaire à double effet. Ce vérin est contrôlé par une servovalve qui permet de convertir un signal électrique en un débit. Le système est présenté sur la Figure 4.3.

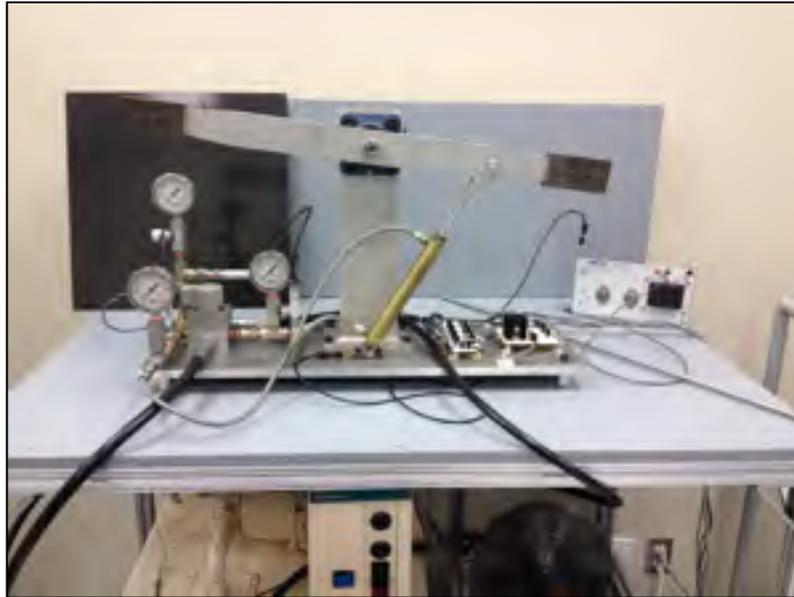


Figure 4.3 Banc d'essai

Les trois organes du système (servovalve, vérin et bras) peuvent être décrits analytiquement. Toutefois, la présence d'écoulements fluidiques et le mouvement du bras introduisent des éléments non-linéaires. Gagnon (2011) a montré ces comportements et a linéarisé le système autour d'un point de fonctionnement afin de disposer de fonctions de transfert. Le transfert total exprime le rapport entre la sortie et l'entrée du système. Dans notre cas, il s'agit du rapport entre l'angle du bras, θ , et la tension de commande appliquée à la servovalve, u .

La tension appliquée sur la servovalve doit être telle que :

$$-2,5V \leq u \leq 2,5V \quad (4.1)$$

Pour la suite de ce travail, nous considérerons que l'angle de rotation du bras, θ , est compris entre :

$$\frac{5\pi}{6} \leq \theta \leq \frac{7\pi}{6} \quad (4.2)$$

La fonction de transfert totale du système, G , d'ordre 4, est donnée par Gagnon (2011) :

$$G = \frac{\theta(s)}{U(s)} = \frac{\mathcal{N}}{s(s^2 + bs + c)(es + 1)} \quad (4.3)$$

$$G = \frac{5,014.10^{-8}}{1.914.10^{-16}s^4 + 6.511.10^{-14}s^3 + 2,822.10^{-10}s^2 + 8,817.10^{-8}s} \quad (4.4)$$

On peut, en outre, donner une représentation normalisée de l'expression précédente. Pour cela, il convient de diviser l'ensemble des coefficients de la fonction de transfert par le coefficient de la plus grande puissance présente au dénominateur. On a alors :

$$G = \frac{2,6198.10^8}{s^4 + 340,1558 s^3 + 1,4745.10^6 s^2 + 4,6065.10^8 s} \quad (4.5)$$

Notons que nous ne nous attarderons pas sur la description de cette fonction de transfert. Le détail des coefficients se trouve dans Gagnon (2011).

Par ailleurs, Gagnon (2011) a montré que la fonction de transfert du capteur, noté K_{pot} , est un gain. En effet, le capteur de position est constitué d'un potentiomètre de $5k\Omega$ à un tour. Gagnon (2011) donne la valeur de K_{pot} :

$$K_{pot} = 0,7968 V.rad^{-1} \quad (4.6)$$

En plus du CAN de la carte d'évaluation, la tension de sortie est acquise à l'aide d'une carte d'acquisition National Instrument de type NI-6229 qui dispose d'un CAN de 16 bits. Il nous est alors possible de récupérer les données à l'aide du logiciel LabView.

4.2 Dynamique interne à la carte d'évaluation

La carte à notre disposition est un kit d'évaluation Spartan 3-A FPGA Start Kit de la société Xilinx (Xilinx, 2006). Le FPGA embarqué sur la carte d'évaluation est un Spartan 3-A contenant 700000 portes logiques disposant de 372 E/S utilisables par l'utilisateur. La carte

d'évaluation comporte plusieurs éléments d'interfaces tels des boutons poussoirs ou des diodes électroluminescentes (DEL). En outre, celle-ci dispose de :

- CAN : Le LTC1407A-1 de Linear Tech, dispose de deux canaux et muni d'un préamplificateur LTC6912-1 qui permet de régler la plage de conversion. La communication avec ces deux éléments se fait par une liaison SPI. De plus, la conversion s'effectue en signé sur 14 bits (Xilinx, 2007) ;
- CNA : Le LTC2624 Quad DAC de Linear Tech comporte quatre canaux. La conversion est effectuée à partir de données définies sur 12 bits non signé. Notons que la communication avec le CNA s'effectue également par une liaison SPI.

Notons que le CAN et le CNA utilisent les mêmes lignes présentes sur le bus SPI de la carte d'évaluation. Il est alors nécessaire de gérer la synchronisation de ces deux éléments.

Pour comprendre les mécanismes internes au FPGA, on se propose d'étudier la dynamique relative à l'action proportionnelle (cas le plus simple) avant de généraliser aux actions intégrale et dérivée. Nous l'avons rappelé, l'objectif est de délivrer une commande, u , proportionnelle à l'erreur, e . On doit donc avoir :

$$u = K \times e \quad (4.7)$$

K représente le gain proportionnel permettant d'obtenir une dynamique voulue.

e désigne l'erreur analogique.

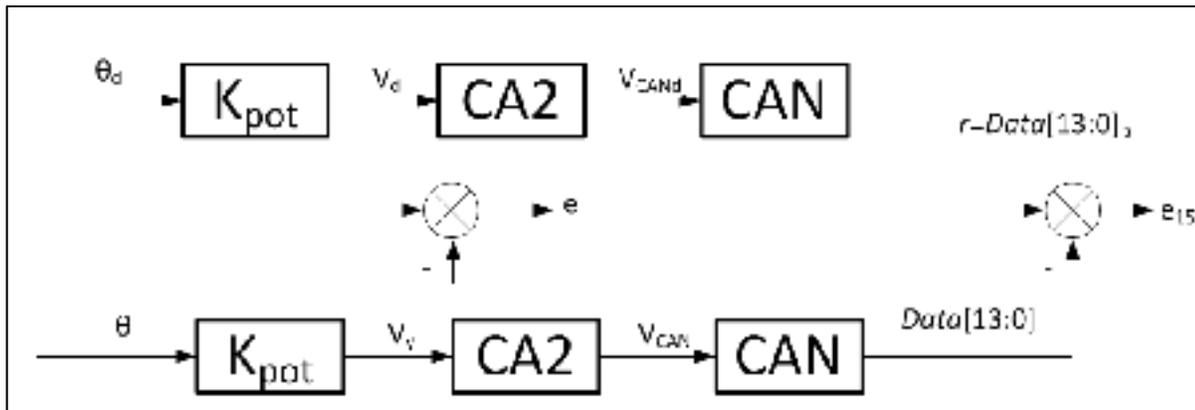


Figure 4.4 Représentation de la formation de l'erreur numérique et analogique

Afin de respecter l'équation (4.7), il est nécessaire de s'intéresser à l'intégralité du chemin que parcourent les signaux. Dans le système réel, toutefois, l'erreur analogique, e , est une représentation abstraite et n'existe pas physiquement. La Figure 4.4 permet de se représenter la formation de l'erreur numérique, e_{15} , qui est utilisée dans le processus de commande et l'erreur analogique e qui permet une représentation plus commode à la compréhension. On peut alors donner une représentation montrant le lien entre la représentation abstraite et physique du système via la Figure 4.5. On comprend à l'aide de cette figure que le correcteur (C) implémenté dans le FPGA devra prendre en considération les différentes étapes de traitement afin qu'il soit équivalent à la représentation abstraite du régulateur idéal (K). Nous allons développer nos calculs sur la dualité évoquée par la Figure 4.5.

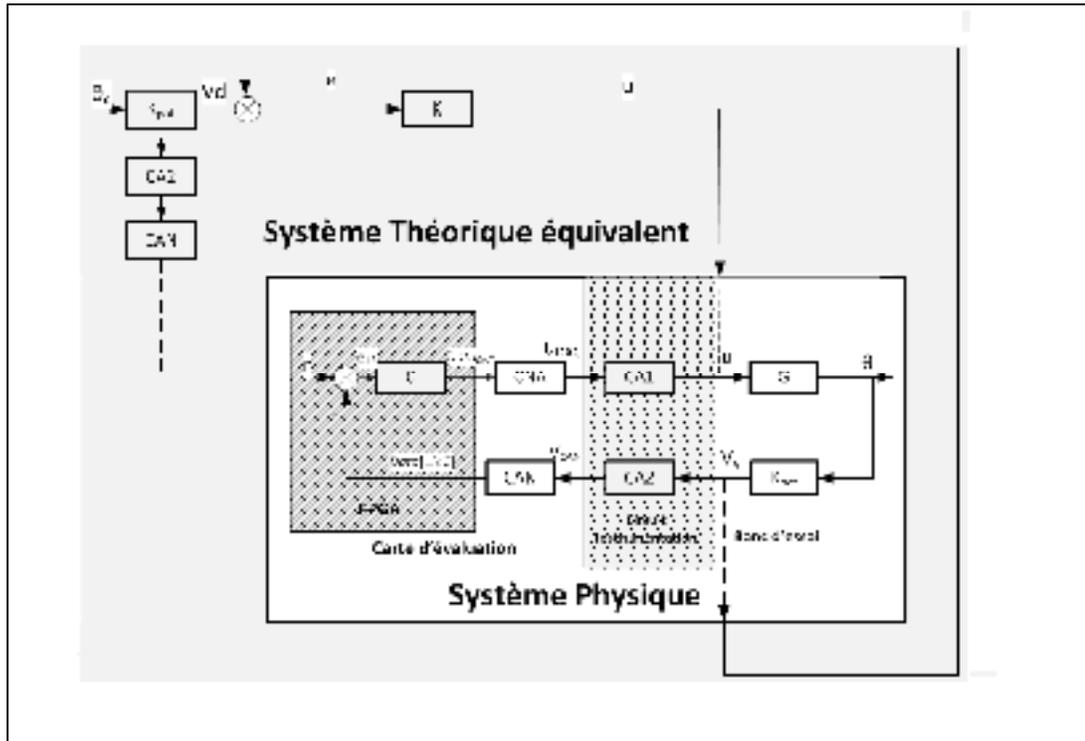


Figure 4.5 Représentation physique et abstraite du système

4.2.1 Description du système de commande

Le capteur de position angulaire permet de transposer la valeur de l'angle en une tension électrique, V_y , selon les équations (4.2) et (4.6):

$$V_y \in [2,08 ; 2,9]V \quad (4.8)$$

De la même façon, la consigne angulaire, θ_d , peut indiquer des angles compris dans la plage de l'équation (4.2). En appliquant le gain de l'équation (4.6) on dispose d'une tension de consigne, V_r , et non plus d'une consigne angulaire. L'erreur analogique, e , peut alors être bornée:

$$\min(V_r - V_y) \leq e \leq \max(V_r - V_y) \quad (4.9)$$

4.2.2 Convertisseurs A/N, N/A et conditionnement du signal

La sortie, θ , transformée en une tension V_y est acquise grâce au CAN, qui fournit au FPGA une valeur signée sur 14 bits. La consigne est, quant à elle, directement codée dans le programme et, correspond à une valeur signée sur 14 bits, afin d'assurer la cohérence de l'ensemble. L'erreur numérique est donc définie sur 15 bits (signés) afin d'éviter tout dépassement (ou *overflow*). On écrira la valeur de l'erreur numérique définie sur 15 bit par : e_{15} .

Dans notre cas, nous disposons d'un CAN qui dispose d'une plage d'entrée définie par :

$$v_{CAN} \in [0,4 ; 2,9]V \quad (4.10)$$

On dispose alors d'un premier circuit de conditionnement du signal qui permet de passer de la tension de sortie (Equation (4.8)) à la plage d'entrée du CAN (Equation 4.10). (Le détail du calcul se trouve en ANNEXE III) :

$$v_{CAN} = 2,976 \times V_y - 5,79 \quad (4.11)$$

Regardons à présent le transfert (tension d'entrée v_{CAN} à la valeur numérique $data[13:0]$) réalisé par le CAN, d'après (Xilinx, 2007) :

$$data[13:0] = \frac{1,65 - v_{CAN}}{1,25} \times 8192 \quad (4.12)$$

En combinant les deux précédentes équations (4.11)-(4.12) on peut écrire :

$$data[13:0] = \frac{-3,47 \times V_y + 8,47}{1,25} \times 8192 \quad (4.13)$$

A partir des équations (4.8) et (4.9), il est possible d'exprimer l'erreur analogique théorique¹¹ et de réaliser l'encadrement suivant :

$$-0,82 \leq e \leq 0,82 \quad (4.14)$$

Comme l'erreur numérique est signée et définie sur 15 bits, un autre encadrement est possible :

$$-16383 \leq e_{15} \leq 16382 \quad (4.15)$$

On peut alors exprimer l'erreur numérique en fonction de l'erreur analogique par :

$$e_{15} = \alpha_{CAN} \times e \quad (4.16)$$

On déduit alors de (4.14)-(4.16) que :

$$\alpha_{CAN} = -19979 \quad (4.17)$$

Ne perdons pas de vue l'équation (4.7) et cherchons la commande à fournir au système. Soit un gain proportionnel défini sur M bits et noté K_{PM} alors la commande sera définie sur (15+M-1) bits tel que :

$$u_{15+M-1} = K_{PM} \times e_{15} \quad (4.18)$$

Il est à présent intéressant de s'intéresser à l'autre bout de la chaîne de traitement. Le CNA à disposition sur la carte d'évaluation est un convertisseur non signé sur 12 bits qui donne une tension telle que :

¹¹ On précise « théorique » puisque la seule grandeur physique mesurée, mesurable et connue est la tension de sortie. La consigne, elle, est implémentée avec sa valeur numérique. On extrapole sa supposée valeur analogique à partir de (4.11), ce qui permet de définir une erreur analogique (qui est virtuelle).

$$0 \leq u_{DAC} \leq 3,3V \quad (4.19)$$

Or le système que l'on souhaite contrôler accepte une tension de commande définie par (2.1). Il est nécessaire de mettre en place un circuit d'adaptation (CA1) (dont le détail est donné en ANNEXE III) :

$$u = 1,5152u_{DAC} - 2,5 \quad (4.20)$$

L'expression de u_{DAC} d'après (Xilinx, 2007) est :

$$u_{DAC} = 8,0586.10^{-4} \times Data_{DAC} \quad (4.21)$$

Où, $Data_{DAC}$ s'exprime par :

$$Data_{DAC} = u_{12} + 2047 \quad (4.22)$$

Où u_{12} représente la commande numérique signée sur 12 bits.

Il nous reste à établir le lien entre les équations (4.18) et (4.21) :

$$u_{12} = (u_{15+M-1}) \gg (15 + M - 1 - 12) \quad (4.23)$$

où \gg représente un décalage vers la droite.

L'équation (4.23) peut aussi s'écrire :

$$u_{12} = \frac{(u_{15+M-1})}{2^{(15+M-1-12)}} \quad (4.24)$$

4.2.3 Transfert total

Nous nous trouvons à présent en mesure d'exprimer le transfert entre l'erreur analogique, e , et la tension de commande, u , illustré dans l'équation (4.7).

Des équations (4.19) et (4.20), on a :

$$u = 1,5152 \times 8,0586.10^{-4} \times Data_{DAC} - 2,5 \quad (4.25)$$

A partir des équations (4.18), (4.21) et (4.24) on peut exprimer l'équation (4.25) par :

$$u = 1,5152 \times 8,0586.10^{-4} \left(\frac{K_{PM} \times e_{15}}{2^{(15+M-1-12)}} + 2047 \right) - 2,5 \quad (4.26)$$

Puis à l'aide de l'équation (4.16), il vient :

$$u = 1,5152 \times 8,0586.10^{-4} \left(\frac{\alpha_{CAN} \cdot K_{PM} \times e}{2^{(15+M-1-12)}} + 2047 \right) - 2,5 \quad (4.27)$$

En développant l'équation (4.27), on se rend compte que l'expression se simplifie et qu'il vient :

$$u = \left(1,5152 \times 8,0586.10^{-4} \times \frac{\alpha_{CAN} \cdot K_{PM}}{2^{(15+M-1-12)}} \right) \times e \quad (4.28)$$

En identifiant les équations (4.28) et (4.7), on peut alors écrire que le gain proportionnel désirée K s'exprime par :

$$K = 1,5152 \times 8,0586.10^{-4} \times \frac{\alpha_{CAN} \cdot K_{PM}}{2^{(15+M-1-12)}} \quad (4.29)$$

On peut en déduire le « gain statique du système », G_{sys} tel que :

$$K = G_{sys} \times \frac{K_{PM}}{2^{(15+M-1-12)}} \quad (4.30)$$

Avec

$$G_{sys} = 24,4 \quad (4.31)$$

Cela implique que le gain réellement implémenté dans le système numérique sera :

$$K_{PM} = \frac{K}{G_{sys}} \times 2^{(15+M-1-12)} \quad (4.32)$$

On voit ainsi que la valeur du gain qui sera codée dépend du nombre de décalages effectués.

4.2.4 Contraintes et limitations dynamiques

Le résultat énoncé par (4.28) implique certaines limitations sur la dynamique de notre système. Nous nous proposons, ici, de les énoncer et, tentons d'y apporter des solutions.

En effet, la tension de commande $u \in [-2,5 ; 2,5]V$ implique que l'équation (4.7) ne peut être respectée sans provoquer la saturation de l'actionneur. Ainsi la plage dynamique du contrôleur est, de fait, réduite. En prenant la tension de commande maximum :

$$u_{max} = 2,5 V \quad (4.33)$$

De (4.7) on tire :

$$e_{max} = \frac{u_{max}}{K} \quad (4.34)$$

Et on a pour $K = 10$:

$$e_{max} = 0,250 V \quad (4.35)$$

L'erreur analogique maximale possible afin de disposer du comportement dynamique souhaité doit être $e \in [-0,25 ; 0,25]V$. Cette plage représente une erreur numérique sur 15 bits telle que :

$$e_{15} \in [-5056 ; 5056] \quad (4.36)$$

A partir de (4.2), en prenant une consigne angulaire $\theta_d = \pi rad$, on obtient une tension de consigne selon (4.6) qui vaut $V_r = 2,5 V$ et la plage de mouvement du bras sera :

$$y \in [-0,314 ; 0,314]rad \quad (4.37)$$

Ce qui correspond en degré à :

$$y \in [-18 ; 18]^\circ \quad (4.38)$$

Concrètement, dans notre application cela nous est acceptable. En effet, le modèle du système physique est linéarisé autour du point $\theta = \pi rad$. Ainsi notre modèle peut être considéré comme vrai autour du point choisi pour de faibles variations (Gagnon, 2011).

Nous avons pu démontrer qu'il ne suffisait pas d'implanter la valeur de notre régulateur théorique directement dans le FPGA pour bénéficier de la commande souhaitée. Ce sont les différents traitements que reçoit le signal entrant qui le distord. Nous avons raisonné, ici, avec l'exemple d'un simple correcteur proportionnel. Cependant, ne perdons pas de vue que l'on souhaite mettre en place, dans un premier temps, un régulateur PID. Ainsi, les équations (4.23)-(4.25), doivent être réécrites.

Avec l'équation (4.23), nous avons un gain proportionnel k_p , lors de son implémentation nous ne passerons pas la valeur de k_p mais celle définie par l'équation (4.32). Bien entendu les actions intégrale et dérivée sont elles-aussi touchées par l'ajustement nécessaire des coefficients (équation (4.32)). En effet, même dans le cas où ce sont les valeurs précédentes qui agissent, comme l'indiquent les équations (4-24)-(4.25), le gain interne, G_{sys} , doit être pris en compte. En ne tenant pas compte de ce phénomène, la dynamique demandée ne peut pas être assurée. Pour prouver ce que nous disons, prenons un exemple simple. L'équation (4.24) nous donnait :

$$I(k) = k_i T_E e(k-1) + I(k-1)$$

Imaginons que cette équation correspond au correcteur implanté. Nous aurions un compensateur I uniquement. Cela revient à dire que le terme $I(k)$ correspond directement à la commande présente dans le FPGA. Pour faire le parallèle avec les calculs précédents donnons l'équivalence suivante :

$$I(k) \equiv u_{15+M-1}$$

Faisons abstraction du terme en (k-1), ce qui se passe lors du premier échantillon traité, on se trouve alors dans le cas d'un contrôleur proportionnel. La prise en compte du gain interne s'avère donc nécessaire. A présent imaginons, que le terme k-1 entre en jeu. Celui-ci introduit une dynamique (celle de l'intégrale). On voit alors que si le terme $I(k-1)$ n'est pas affecté par le gain, G_{sys} , la dynamique voulue ne pourra être obtenue.

Par ailleurs, notons que nous effectuerons deux tests de notre régulateur. Le premier est un test de l'action proportionnelle uniquement afin de vérifier la validité du gain interne, G_{sys} . Le second test présente un PID complet dont les coefficients ont été trouvés par Gagnon (2011). Nous pourrions alors comparer nos résultats afin de savoir si la commande implantée *via* un FPGA est efficace et conforme aux attentes et spécifications.

4.3 Modélisation du système dans l'espace d'état

Après avoir décrit le système et en particulier en ayant donné la fonction de transfert du processus à commander, il nous reste à représenter celui-ci dans l'espace d'état. Il est possible à partir de la fonction de transfert d'un système d'en donner une description dans un système (A, B, C, D) comme rappelé par les expressions (2.68)-(2.71). Pour donner la représentation dans l'espace d'état, nous utiliserons la fonction de transfert donnée par l'équation (4.5) :

$$A = \begin{pmatrix} -340,1558 & -1,4745 \times 10^6 & -4,6065 \times 10^8 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (4.39)$$

$$B = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (4.40)$$

$$C = (0 \quad 0 \quad 0 \quad 2,6198 \times 10^8) \quad (4.41)$$

On peut alors donner la matrice d'observabilité décrite par l'équation (2.69) :

$$\mathbb{O} = \begin{pmatrix} 0 & 0 & 0 & 2,6198 \times 10^8 \\ 0 & 0 & 2,6198 \times 10^8 & 0 \\ 0 & 2,6198 \times 10^8 & 0 & 0 \\ 2,6198 \times 10^8 & 0 & 0 & 0 \end{pmatrix} \quad (4.42)$$

Le rang de la matrice d'observabilité est de 4. Le système est donc complètement observable.

On se propose à présent de regarder la matrice de commandabilité définie par l'expression (2.67) :

$$\mathbb{C} = \begin{pmatrix} 1 & -340,1558 & -1,3588 \times 10^6 & 5,0309 \times 10^8 \\ 0 & 1 & -340,1558 & -1,3588 \times 10^6 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.43)$$

La matrice est obtenue à l'aide du logiciel Matlab via la commande *ctrb*. On décide de regarder le rang de cette matrice. Le logiciel de calcul numérique nous renvoie un rang égal à 3. Or nous avons démontré qu'un système défini sous forme canonique de commandabilité est doté d'une matrice de commandabilité de rang maximal. Pour expliciter ce phénomène il est nécessaire de vérifier le conditionnement de notre matrice \mathbb{C} . En regardant le conditionnement à l'aide d'une 2-norme matricielle subordonnée, on obtient :

$$\text{cond}(\mathbb{C}) = \|\mathbb{C}^{-1}\|_2 \|\mathbb{C}\|_2 = 2,3176 \times 10^{17} \quad (4.44)$$

On sait que le bon-conditionnement d'une matrice passe par une valeur proche de 1. L'expression précédente illustre le mauvais conditionnement générant une erreur lors du calcul du rang. Notons par ailleurs que le conditionnement de la matrice d'observabilité est unitaire.

Il convient alors d'utiliser un autre moyen de représenter notre système dans l'espace d'état. L'utilisation de la fonction Matlab *ss* permet la conversion d'une fonction de transfert en une représentation en ayant au préalable appliquée une transformation de similarité (Mathworks, 2007). La transformation répond à :

$$\tilde{A} = T^{-1}AT \quad (4.45)$$

Celle-ci repose sur le fait que les normes des colonnes et des lignes, de la matrice \tilde{A} , soient aussi proches que possible. De plus, la matrice de transformation T est une matrice diagonale

dont les éléments sont des puissances de deux. L'effet d'une telle transformation est d'obtenir une matrice des vecteurs propres de \tilde{A} avec un meilleur conditionnement que celle de A . Le mauvais conditionnement se retrouve alors porté par la matrice T . Moore (1981) a montré que ce type de représentation, « balancée » facilitait la mise en œuvre numérique.

Avec cette transformation la représentation de notre système dans l'espace d'état est :

$$A = \begin{pmatrix} -3.4015 & -1440 & -878,6 & 0 \\ 1024 & 0 & 0 & 0 \\ 0 & 512 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (4.46)$$

$$B = \begin{pmatrix} 16 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (4.47)$$

$$C = (0 \quad 0 \quad 0 \quad 24,8523) \quad (4.48)$$

On obtient alors un système (A, B, C) qui dispose d'une matrice de commandabilité :

$$\mathbb{C} = \begin{pmatrix} 16 & -5442,5 & -2,174 \times 10^7 & 8,0495 \times 10^9 \\ 0 & 16384 & -5,5731 \times 10^6 & -2,2262 \times 10^{10} \\ 0 & 0 & 8,389 \times 10^6 & -2,8534 \times 10^9 \\ 0 & 0 & 0 & 8,389 \times 10^6 \end{pmatrix} \quad (4.49)$$

Cette matrice triangulaire supérieure, à éléments diagonaux non nuls, présente un rang plein. Le système est donc complètement commandable. Notons par ailleurs, que la fonction Matlab *rank* indique un rang maximal. La transformation de similarité a donc permis de s'affranchir des difficultés numériques.

La matrice d'observabilité du système est :

$$\mathbb{O} = \begin{pmatrix} 0 & 0 & 0 & 24,8523 \\ 0 & 0 & 24,8523 & 0 \\ 0 & 12724 & 0 & 0 \\ 1,3030 \times 10^7 & 0 & 0 & 0 \end{pmatrix} \quad (4.50)$$

Le rang de cette matrice est plein, le système est donc complètement observable.

CHAPITRE 5

IMPLÉMENTATION DU MODULE PID

On se propose, à présent, de regarder plus en détails l'implémentation concrète d'un régulateur de type PID filtré utilisant l'AD à l'aide du logiciel Simulink. Avant de décrire plus en profondeur le système, il est nécessaire d'appliquer l'AD sur le régulateur PID numérique.

5.1 L'AD appliquée au régulateur PID filtré

Nous l'avons vu au paragraphe 2.3, il est possible de décrire l'implémentation de plusieurs façons. Nous allons ici utiliser les formules (2.50)-(2.52) qui permettent de réduire la taille de la mémoire nécessaire lors de l'implémentation *via* l'AD en partitionnant le système comme proposé par White (1989) ou Mathworks (2007).

Ainsi en appliquant le principe de l'arithmétique distribuée à (2.50)–(2.52) et en considérant les entrées et sorties définies sur N bits.

D'abord, réécrivons les expressions (2.50)–(2.52) afin d'alléger les notations : nous écrivons $x(k)$ qui représentait le k -ième échantillons du signal x , nous le noterons à présent x_k . D'où :

$$P_k = k_p e_k \quad (5.1)$$

$$I_k = k_i T_E e_{k-1} + I_{k-1} \quad (5.2)$$

$$D_k = k_d f_d e_k - k_d f_d e_{k-1} + (1 - f_d T_E) D_{k-1} \quad (5.3)$$

En appliquant le principe de l'AD, on a :

$$P_k = \sum_{i=0}^{N-1} (k_p e_{ki}) 2^i \quad (5.4)$$

$$I_k = \sum_{i=0}^{N-1} (k_i T_E e_{(k-1)i} + I_{(k-1)i}) 2^i \quad (5.5)$$

$$D_k = \sum_{i=0}^{N-1} (k_d f_d e_{ki} - k_d f_d e_{(k-1)i} + (1 - f_d T_E) D_{(k-1)i}) 2^i \quad (5.6)$$

Avec (e_{ki}) représentant le i -ème bit du k -ième échantillon de e i.e :

$$e_{ki} = \begin{cases} 0 \\ 1 \end{cases} \quad (5.7)$$

Des trois expressions précédentes (5.4)-(5.6), on remarque que l'on dispose uniquement de 4 vecteurs d'entrée. Il serait donc possible de combiner (5.4)-(5.6) afin d'obtenir une somme de quatre produits telle que :

$$u_k = A_1 e_k + A_2 e_{k-1} + A_3 I_{k-1} + A_4 D_{k-1} \quad (5.8)$$

$$A_1 = k_p + k_d f_d ; \quad A_2 = k_i T_E - k_d f_d ; \quad A_3 = 1 \quad \text{et} \quad A_4 = 1 - f_d T_E$$

L'écriture s'en trouverait réduite mais souvenons-nous que la taille nécessaire en mémoire est imposée par la loi 2×2^K , avec K le nombre de produits. Dans ce cas, il nous faudrait disposer de 32 mots en mémoire. En ajoutant de la circuiterie, il nous serait possible de diminuer à 16 mots la taille de la ROM. Toutefois, Chan, Moallem et Wang (2004) propose de profiter de l'architecture parallèle pour réduire la taille mémoire nécessaire. En effet, cela

s'explique par le fait qu'au lieu d'une mémoire ROM unique regroupant tous les coefficients, nous disposons de trois ROM distinctes associées à chacune des actions P, I et D.

Mathématiquement, cela vient de :

Soient $a, b \in \mathbb{N}^*$,

$$2^a + 2^b \leq 2^{a+b} \quad (5.9)$$

Nous aurons donc les occupations mémoires suivantes d'après (5.4)-(5.6) :

$$ROM(P) = \begin{cases} 0 \\ k_p \end{cases} \quad (5.10)$$

$$ROM(I) = \begin{cases} 0 \\ k_i T_E \\ 1 \\ k_i T_E + 1 \end{cases} \quad (5.11)$$

$$ROM(D) = \begin{cases} 0 \\ 1 - f_d T_E \\ -k_d f_d \\ -k_d f_d + 1 - f_d T_E \\ k_d f_d \\ k_d f_d + 1 - f_d T_E \\ 0 \\ 1 - f_d T_E \end{cases} \quad (5.12)$$

Soit un total de 14 mots qui doivent être stockés en mémoire.

5.2 Mise en place des blocs fonctionnels

Après avoir vu le mode d'implantation théorique d'un régulateur de type PID filtré à l'aide de l'AD, nous allons à présent nous focaliser sur l'implémentation pratique de celui-ci.

D'abord, nous décrirons l'environnement dans lequel nous allons travailler, puis nous rentrons au cœur du système afin de montrer la mise en pratique de la théorie.

5.2.1 L'interface Simulink

La compagnie Mathworks (2007) a développé une boîte à outils, HDL Coder, permettant de générer du code HDL à partir de schéma Simulink. La documentation associée est fournie et complète. On trouve de nombreux exemples et fichiers de démonstration qui permettent une prise en main rapide du module. La majorité des blocs usuels est convertible.

Pour réaliser une fonction synthétisable sur un FPGA via Simulink, il est nécessaire de connaître quelques éléments de base. Pour présenter le fonctionnement de HDL Coder, on se propose de présenter le cas d'un exemple simple.

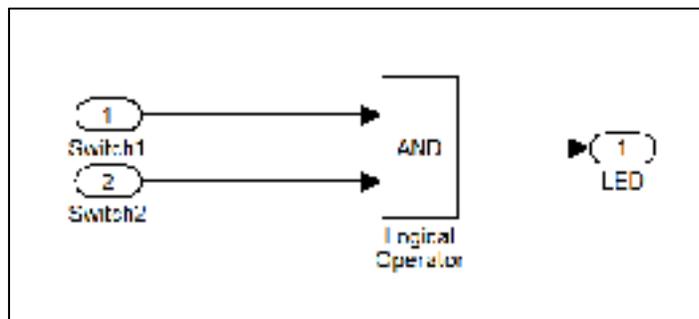


Figure 5.1 Implémentation d'une fonction basique sous Simulink

Sur la Figure 5.1, on distingue deux types fondamentaux de blocs : *in port* et *out port*. Concrètement lors de la génération du code HDL, ces ports vont indiquer les E/S du système. L'exemple présenté par la Figure 5.1 est extrêmement basique : il s'agit d'une porte ET dont les deux entrées sont des interrupteurs et la sortie une diode électroluminescente (DEL). Une fois le fichier HDL généré, il est possible de le synthétiser via ISE puis de l'implanter sur le FPGA. Il nous faudra alors écrire un fichier de contraintes utilisateur (ucf) qui permet, en outre, d'assigner les E/S dans le code aux E/S physiques et présentes sur la carte.

Outre les blocs standards de la bibliothèque Simulink synthétisables, un bloc particulier requiert notre attention : la fonction embarquée Matlab (*Embedded Matlab Function*, EMF). Celle-ci permet de disposer d'un bloc entièrement programmable par l'utilisateur et entièrement synthétisable pour peu que l'on prenne garde à quelques éléments. Lors d'une simulation Matlab, une EMF est exécutée à chaque période de simulation. Elle permet donc un travail en temps réel. De plus, toutes les variables présentes dans une EMF sont écrasées à chaque itération à moins qu'elles ne soient déclarées persistantes (*persistent*). On utilisera les EMF, notamment pour implémenter les machines d'état. Ces blocs présentent un intérêt majeur : créer des fonctions bas niveau à l'aide d'un langage évolué tel que le langage Matlab. D'ailleurs, ce qui finit de prouver leur efficacité est l'existence d'une librairie Simulink dédiée à la génération de code HDL et uniquement réalisée à l'aide de EMF (*eml_hdl_design_patterns*). Parmi les éléments les plus puissants de cette bibliothèque se trouve un cœur de processeur (*Central Processing Unit*, CPU) RISC à accumulateur, 8 bits, exclusivement réalisé à l'aide d'EMF. Les EMF se structurent de la même façon qu'un « composant » dans le langage VHDL. En effet, l'utilisateur définit les entrées et les sorties de la fonction. Notons qu'il est possible de passer une variable présente dans l'espace de travail de Matlab (*Workspace*) en paramètre d'une EMF. Notons toutefois que le paramètre doit être fixe et ne peut être variable sous peine de voir échouer la génération de code HDL. La paramétrisation des EMF les rend flexibles et donc aisément modifiables.

Enfin l'élément essentiel à la réalisation de fonctions Simulink pouvant être embarquées sur un FPGA est l'utilisation de la boîte à outils Point Fixe (*Fixed-point toolbox*). En effet, à chaque fois qu'une nouvelle variable apparaît dans Simulink (que ce soit dans un bloc standard, ou une variable déclarée dans une EMF) il est nécessaire de déclarer son type. En effet, Matlab, par défaut, travaille en virgule flottante. Pour cela, on distingue deux cas. Le **type standard** : le choix se fait par un double clic sur le bloc puis en choisissant le type de la variable parmi les plus courants comme le montre la Figure 5.2. Reprenons l'exemple donné dans la Figure 5.1. Puisque nous décrivons des interrupteurs, « switch1 » et « switch2 » sont booléens. De la même façon, le type de la sortie « Led » est binaire. Le **type personnalisé** : qui permet de personnaliser le type que l'on souhaite attribuer à une variable ou signal, en

invoquant le constructeur d'objet numérique à point fixe, $fi()$ (Mathworks, 2007). Par exemple la déclaration de 'MaVariable' en un type personnalisé se fait par : `MaVariable = fi(valeur, signé, NbBitTotal, NbBitDecimal)`. Ainsi la variable `Variable_Test = fi(5, 1, 8, 2)` est codée en binaire par : 000101,00. Notons également, qu'il est possible de personnaliser le type dans les boîtes de dialogue des blocs Simulink standards. Il suffit d'écrire le type à point fixe souhaité selon le gabarit suivant : `fixdt(signé, NbBitTotal, NbBitDecimal)`.

La gestion des types numériques est la plus grande difficulté de l'implémentation et une attention particulière doit y être consacrée lors de la phase de conception. Une équivalence plus complète sur la gestion des types numériques est présentée en ANNEXE IV. Il reste alors à trouver, comme souvent en électronique, le compromis qui convient à la situation : Espace occupé vs. précision/performances.

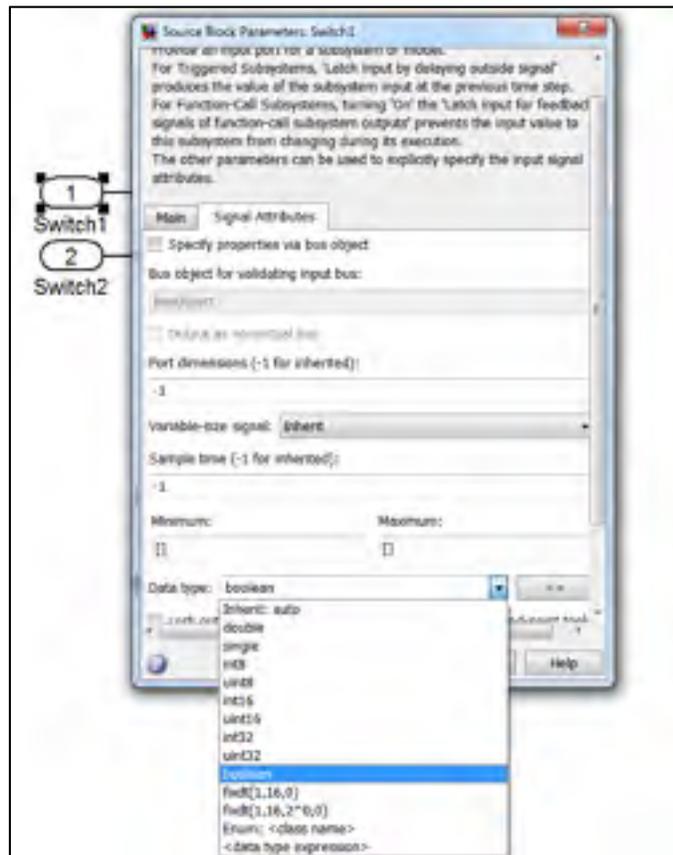


Figure 5.2 Choix du type d'un signal

5.2.2 Implémentation du système de commande type PID

L'implémentation se réalise à l'aide du logiciel Simulink et répond aux exigences décrites précédemment et les valeurs des coefficients du régulateur sont :

$$k_p = 26 \quad (5.13)$$

$$k_i = 1,4$$

$$k_d = 0.122$$

$$f_d = 22$$

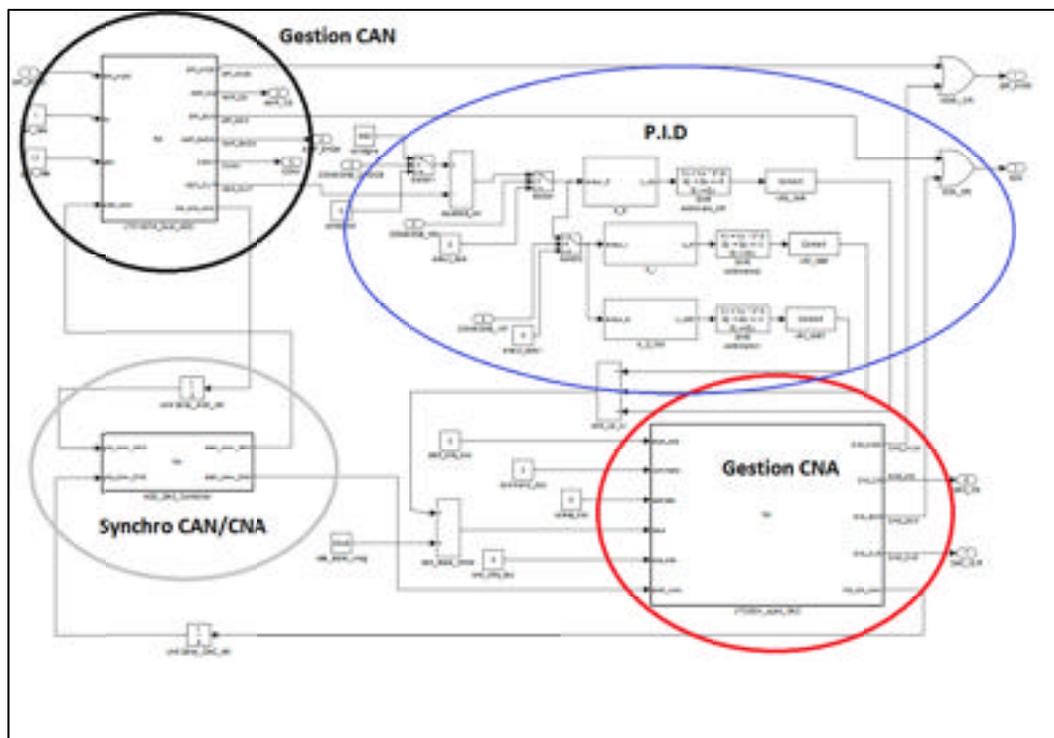


Figure 5.3 Schéma Simulink du régulateur PID complet destiné à être synthétisé

La Figure 5.3 présente le système complet qui est voué à être embarqué sur un FPGA. On distingue plusieurs éléments :

- un contrôleur de CAN (coin supérieur gauche),
- un contrôleur de CNA (coin inférieur droit),
- un gestionnaire de priorité (coté gauche) qui permet de contrôler la fréquence d'échantillonnage. Pour ce travail nous avons :

$$f_e = 40,8 \text{ kHz} \quad (5.14)$$

- des blocs logiques au centre.

Il convient, toutefois, de s'attarder sur les blocs logiques. Ceux-ci sont formés, de blocs Simulink traditionnels et de blocs créés. C'est véritablement cette partie du circuit qui constitue le cœur de la fonction du régulateur PID. Puisqu'on y trouve les trois actions : P, I et D comme l'indique la Figure 5.4.

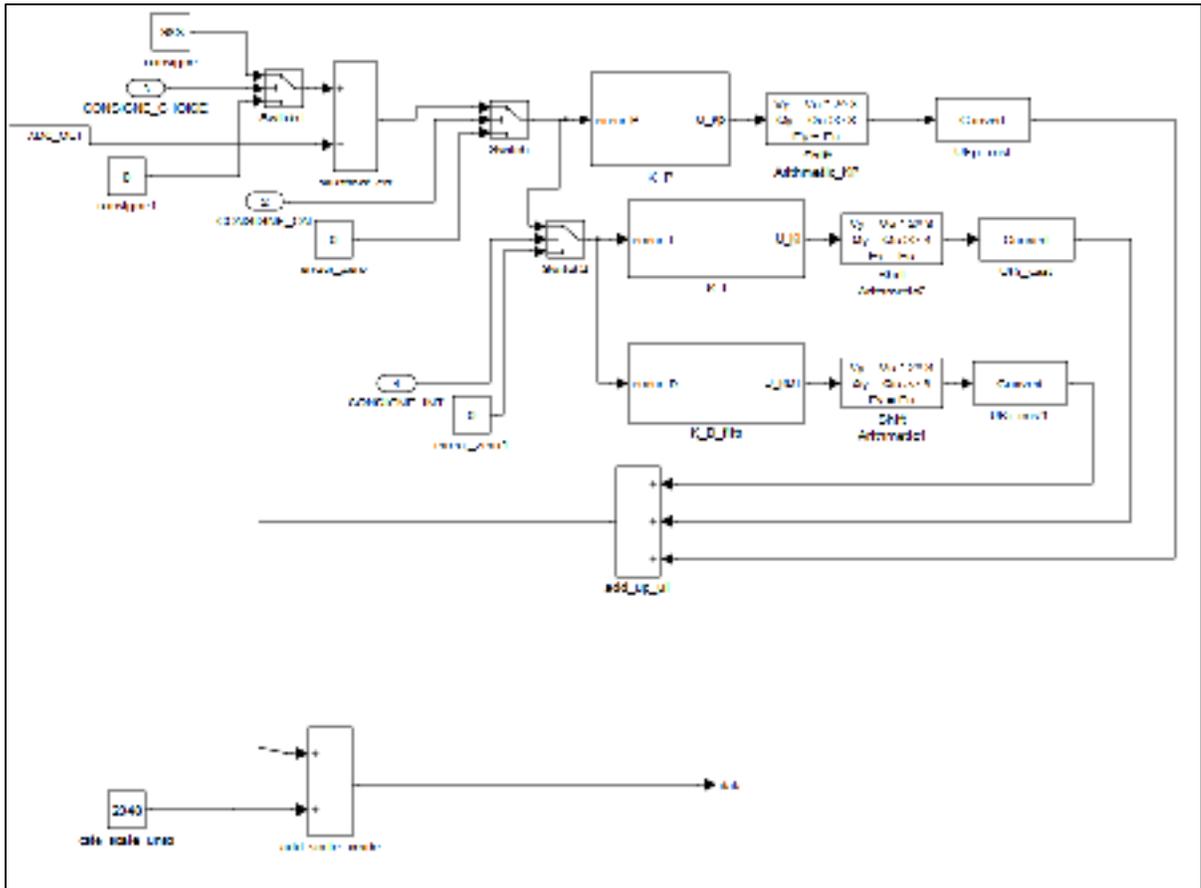


Figure 5.4 Schéma Simulink du cœur du régulateur PID

On distingue l'entrée du système à gauche (« ADC_OUT ») et la sortie (« data »). Concrètement, l'entrée de la Figure 5.4 correspond au flux de données qui arrive en provenance du CAN qui acquiert la sortie du système. Quant à la sortie de la Figure 5.4, elle se dirige vers le CNA, il s'agit de la commande que l'on fournit à l'actionneur. En haut de la même figure, on distingue un bloc Simulink (Constant) qui mentionne « consigne ». Le signal d'erreur se forme alors via le soustracteur (« soustrac_err »). Ce signal va alors passer dans les blocs suivants : « K_P », « K_I » et « K_D_filtr ». Ce sont ces derniers qui réalisent le calcul de la commande *via* la méthode de l'AD. Il convient donc de les expliciter.

5.2.2.1 Action proportionnelle

La Figure 5.5 Schéma Simulink de l'Action proportionnelle avec AD (« K_P ») présente l'implémentation d'un correcteur proportionnel réalisé par AD. Comme nous l'avons évoqué, l'entrée du système « K_P » est l'erreur. Le premier bloc est un registre « Entrée Parallèle-Sortie Série » ou, en anglais, *Parallel In Serial Out* (PISO). Ce registre est fondamental puisqu'il permet un flux bit à bit sur lequel repose l'AD. Par ailleurs, nous avons implémenté dans ce bloc un compteur d'indice qui permet de synchroniser l'ensemble du système tout en tenant compte du bit de signe qui est le bit de poids fort (indiqué N-1). La sérialisation effectuée par le registre PISO est résumée par la Figure 5.6 Principe du registre PISO. Ensuite, le bloc « Proportionnel_KP26 » est constitué d'une LUT et d'un délai. Cet agencement permet lors de la génération du code HDL et de la synthèse d'instancier directement une ROM (Mathworks, 2007). Typiquement, les valeurs en présence dans la LUT correspondent à ce qui est présenté par les équations (5.10) et (5.13). Ainsi, pour un gain proportionnel désiré d'une valeur de 26, l'équation (4.32) nous donne le gain à implémenter. Pour un décalage de 8 bits après calcul, on doit implémenter la valeur 207 dans la LUT. L'application numérique de l'expression (5.10) devient :

$$ROM(P) = \begin{cases} 0 \\ 207 \end{cases} \quad (5.15)$$

Enfin le dernier bloc est un accumulateur. Il permet d'accumuler (d'additionner et de décaler) les valeurs successives issues de la LUT en fonction de l'indice du bit traité. On note que, tant que le dernier bit de la valeur en traitement n'est pas arrivé, on bloque la sortie à 0 pour la première itération et à la valeur précédente pour les suivantes. Par ailleurs, les ports non utilisés sur la Figure 5.5 illustrent des ports de débogage qui nous ont servi lors du développement du système.

On peut présenter un exemple simple afin de faciliter la compréhension. Imaginons une entrée codée sur 5 bits (valeur égale à 9 en décimal) et une valeur multiplicative définie sur 4

bits et stockée dans une LUT (valeur décimale égale à 5). Bien entendu le résultat de la multiplication donnera un résultat égal à 45.

$$9|_{10} = 01001|_2 \text{ et } 5|_{10} = 0101|_2$$

Le registre PISO permet de disposer de la valeur 9 sous la forme sérialisée. Le flux de bits issu du registre PISO est chronologiquement le suivant : 1 – 0 – 0 – 1 – 0.

Ainsi, la valeur présente dans la LUT sera passée à l'accumulateur à chaque fois que le bit issu du PISO est égal à un. La taille de l'accumulateur est définie par la taille des deux facteurs soit $5 + 4 - 1 = 8$. Le Tableau 5.1 illustre le processus de calcul *via* l'AD sur un exemple simple : le produit de 9 par 5. Cela montre, que l'implémentation d'un registre PISO est nécessaire afin de créer un flux sérialisé de bits et que le décompte du nombre de bit est important dans le calcul.

Tableau 5.1 Description des étapes de l'AD

Bit envoyé par PISO	Index du bit PISO	Valeur transmise à l'accumulateur	Valeur Accumulateur
1	0	0101 ₂	00000101 ₂ = 5 ₁₀
0	1	0000 ₂	00000000 ₂ << 1 +00000101 ₂
0	2	0000 ₂	00000000 ₂ << 2 +00000101 ₂
1	3	0101 ₂	00000101 ₂ << 3 +00000101 ₂ = 00101101 ₂
0	4	0000 ₂	= 00101101 ₂ = 45 ₁₀

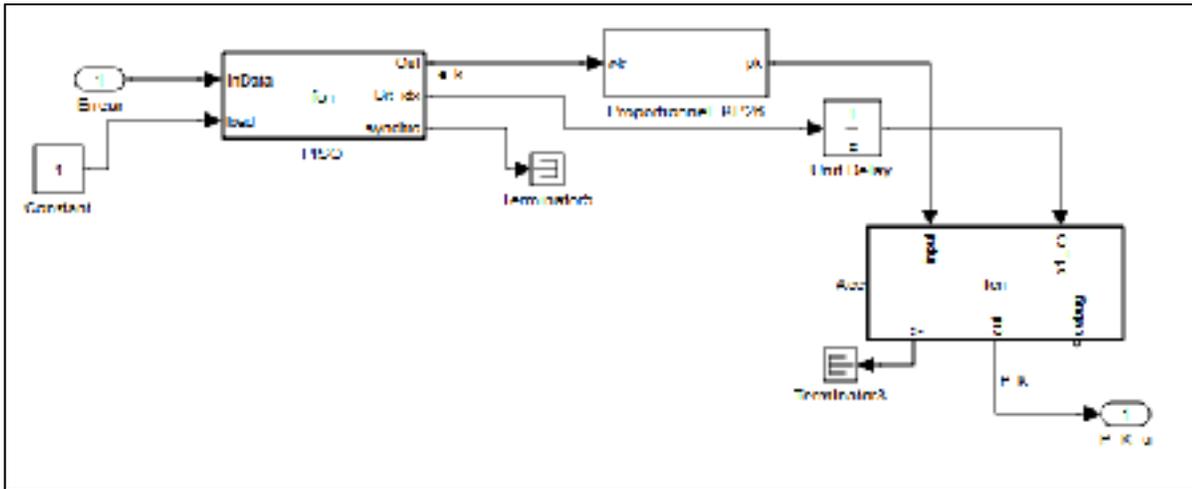


Figure 5.5 Schéma Simulink de l'Action proportionnelle avec AD (« K_P »)

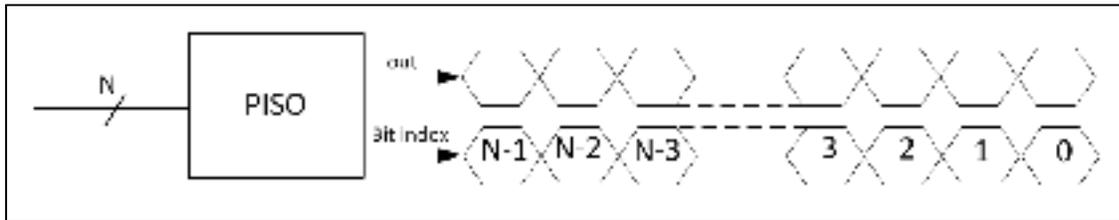


Figure 5.6 Principe du registre PISO

5.2.2.2 Action intégrale

Par rapport à la Figure 5.5, la Figure 5.7 se densifie. En effet, nous avons dû mettre en place un autre registre PISO (ici « PISO_Int »), pour sérialiser la valeur précédente de l'intégrale afin de boucler le système conformément à l'équation (5.5). On remarquera également un nouveau bloc « SISO register » pour *Serial In Serial Out*. Ce bloc permet de constituer un délai. L'erreur sérialisée par le PISO passe bit à bit dans le SISO. Une fois la valeur complète de l'erreur acquise par le SISO, celui-ci la renvoie bit à bit ce qui à pour but d'introduire un délai égal au nombre de bits sur lequel est défini l'erreur. Ainsi nous sommes en mesure de disposer du signal d'erreur à sa valeur précédente (e_{k-1}) conformément à l'expression (5.5). La Figure 5.8 présente un schéma de principe du registre SISO.

implémenter différent de ceux présentés puisqu'il faut tenir compte de la dynamique interne comme nous l'avons montré (équation (4.32)).

Nous l'avons également précisé, la fréquence d'échantillonnage de notre système est de 40800 Hz, soit une période de 24,51 μ s. Ainsi dans le cas de l'action intégrale nous devons implémenter les gains suivants :

$$ROM(I) = \begin{cases} 0 \\ k_i T_E \rightarrow 0,0000112 \\ 1 \rightarrow 0,3239 \\ k_i T_E + 1 \rightarrow 0,3239112 \end{cases} \quad (5.16)$$

Ces gains sont inférieurs à un. On choisit de les définir sur 28 bits dont 27 après la virgule *i.e.* (28,27). Les mots stockés dans la LUT sont également signés. Nous allons donc traiter 27 digits après la virgule. Il faut alors augmenter la taille de notre signal d'erreur (15,0) devenant (42, 27). Ainsi, à la sortie du registre PISO, le flux de bits de l'erreur a une taille de 42.

Focalisons-nous à présent sur l'accumulateur. Celui-ci est défini sur (42,27). Toutefois afin de prendre en compte les valeurs décimales, il est nécessaire d'adapter le traitement réalisé au sein de l'accumulateur. Pour tous les bits après la virgule, le décalage s'effectue à droite. Une fois 27 bits traités, nous repassons dans la procédure standard de décalage vers la gauche. Afin de mieux comprendre les phénomènes mis en jeu, nous proposons un exemple simple de multiplication de deux valeurs décimales.

Soient $9,75|_{10} = 01001,11|_2$ et $0,5|_{10} = 0,1000|_2$

La valeur 9,75 est définie par (7,2), la valeur 0,5 par (5,4). Etant donné que l'un des facteurs dispose de 4 chiffres après la virgule, il convient d'augmenter le terme 9,75. Il vient :

$9,75|_{10} = 01001,1100|_2$, la valeur 9,75 est définie par (9,4).

En passant dans le registre PISO, il vient chronologiquement la suite de 9 bits suivante :

0 – 0 – 1 – 1 – 1 – 0 – 0 – 1 – 0.

La taille de l'accumulateur est définie par (9,4). Le résultat que l'on doit obtenir est 4,875. Le Tableau 5.2 donne à voir la démarche de calcul par AD pour deux valeurs décimales. Nous avons choisi de prendre un exemple simple afin d'illustrer notre propos sur le cas concret de l'action intégrale mais disposant de mots de grande taille (42,27).

Tableau 5.2 Exemple de calcul décimal par AD

Bit envoyé par PISO	Index du bit PISO	Valeur transmise à l'accumulateur	Valeur Accumulateur
0	1	$00000,0000 _2$	$00000,0000 _2$
0	2	$00000,0000 _2$	$00000,0000 _2$
1	3	$0,1000 _2 \gg 2$ $= 00000,0010 _2$	$00000,0010 _2$
1	4	$0,1000 _2 \gg 1$ $= 00000,0100 _2$	$00000,0110 _2$
1	5	$0,0010 _2$ $= 00000,1000 _2$	$00000,1110 _2$
0	6	$00000,0000 _2$	$00000,1110 _2$
0	7	$00000,0000 _2$	$00000,1110 _2$
1	8	$0,1000 _2 \ll 3$ $= 0100,0000 _2$	$00100,1110 _2$
0	9	$00000,0000 _2$	$00100,1110 _2$ $= 4,875 _{10}$

5.2.2.3 Action dérivée

Enfin la Figure 5.9 donne à voir l'implémentation de l'action dérivée filtrée. Il n'y a pas de nouveau bloc utilisé. Le principe reste le même que dans les deux actions précédentes. Toutefois, la taille, de la LUT est plus grande, puisque nous avons 3 entrées.

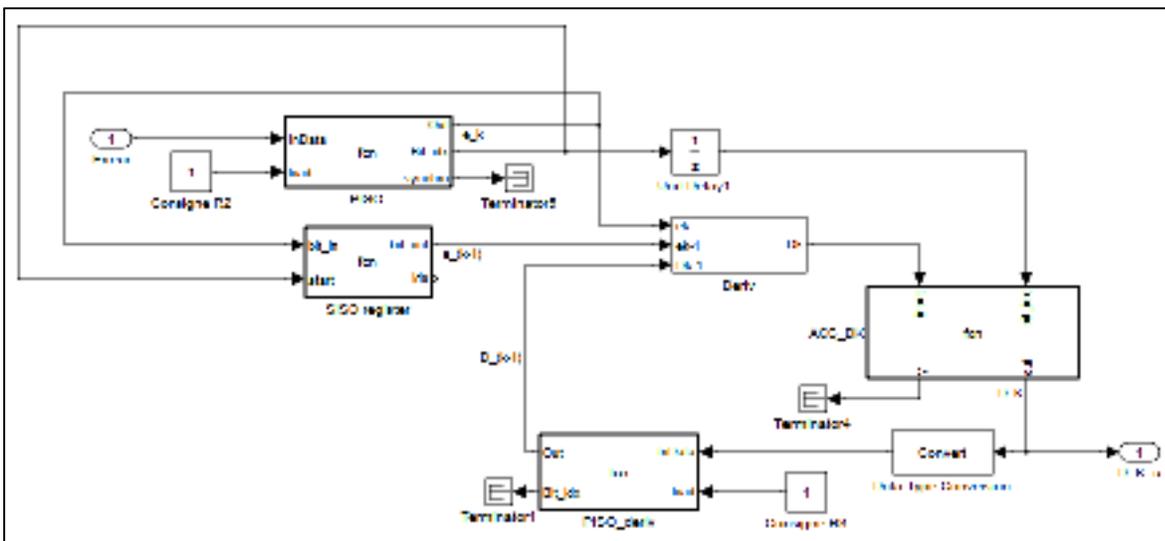


Figure 5.9 Schéma Simulink de l'action dérivée avec AD (« K_D_filtr »)

Les gains à implémenter dans le cas de l'action dérivée sont, conformément aux équations (4.32) et (5.12), les suivants :

$$ROM(D) = \begin{cases} 0 \rightarrow 0 \\ 1 - f_d T_E \rightarrow 0,3238 \\ -k_d f_d \rightarrow -0,8696 \\ -k_d f_d + 1 - f_d T_E \rightarrow -0,5458 \\ k_d f_d \rightarrow 0,8696 \\ k_d f_d + 1 - f_d T_E \rightarrow 1,1934 \\ 0 \rightarrow 0 \\ 1 - f_d T_E \rightarrow 0,3238 \end{cases} \quad (5.17)$$

5.3 Simulation des actions du PID

Dans cette partie, nous nous proposons de présenter les résultats de simulation des différentes actions. En effet, avant de passer à l'implémentation, nous avons dû vérifier que nos trois modules P, I et D fonctionnaient correctement. Ainsi afin de valider le fonctionnement de nos modules nous avons simulé notre bloc créé et un bloc simulink déjà existant.

5.3.1 Simulation de l'action proportionnelle

Afin de valider l'implémentation du module proportionnelle, nous avons comparé le résultat d'une simple multiplication réalisée à l'aide de notre bloc à base d'AD, et le bloc « gain » présent nativement dans Simulink. La Figure 5.10 Schéma de simulation présente le schéma de la simulation effectuée.

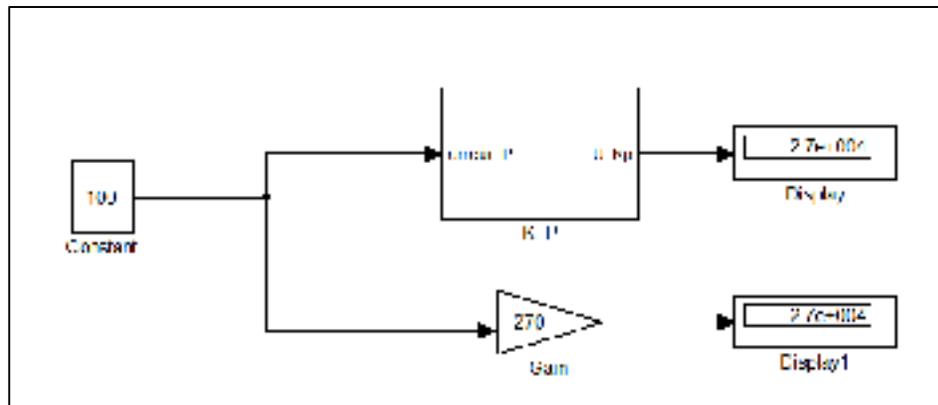


Figure 5.10 Schéma de simulation

Nous avons implémenté un gain de 270 dans chacun des deux blocs. Les deux résultats sont conformes aux attentes. Le module proportionnel permet de multiplier un signal d'entrée selon un gain spécifié (mis en place dans la LUT).

5.3.2 Simulation de l'action intégrale

La simulation mise en place est représentée par la Figure 5.11. On y trouve le bloc implémenté à l'aide de l'AD. Le bloc qui se trouve en dessous est un intégrateur discret présent dans la librairie Simulink. On se propose donc de vérifier que pour une même entrée, les deux blocs se comportent de la même façon. Par ailleurs, les blocs « rate transition » qui entourent le bloc issu de Simulink, permettent d'adapter notre modèle. En effet, le bloc que nous avons implémenté à l'aide de l'AD dispose de registre de 42 bits. Ainsi, la sérialisation inhérente à l'AD implique une actualisation du résultat tous les 42 temps de simulation. Afin d'être en mesure de comparer les résultats, nous indiquons que le bloc « *discrete-time integrator* » a une fréquence d'échantillonnage de 42 fois la période de simulation. On se retrouve donc avec deux systèmes équivalents. On note, en outre, que le gain choisi pour le test est unitaire *i.e.* $k_i = 1$.

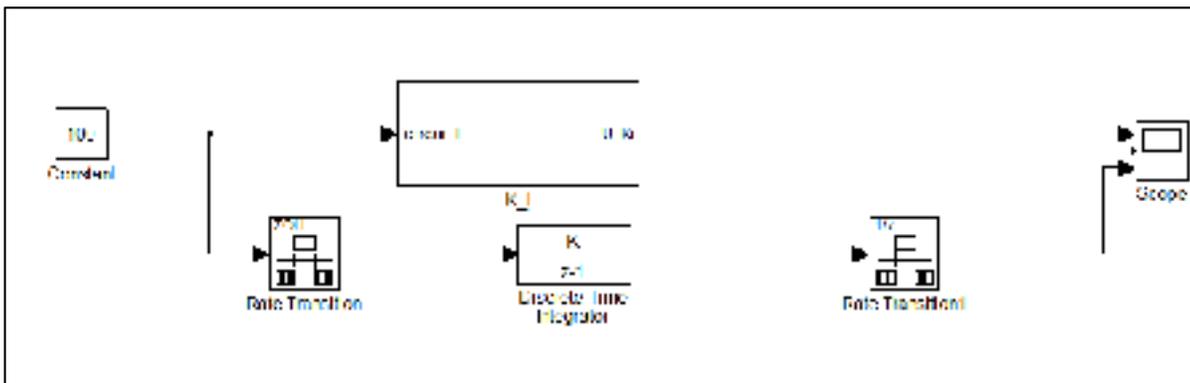


Figure 5.11 Schéma de simulation pour le bloc intégral

Il nous est alors possible de tracer les résultats de la simulation (Figure 5.12). L'entrée de notre système est une constante, donc un intégrateur devrait donner, en sortie, une rampe. On constate que les deux courbes sont très proches de la rampe. On observe également que le bloc intégral qui utilise l'AD sature pour une valeur de 16383. Ce qui correspond à une définition sur 15 bits signés. Par ailleurs, l'autre bloc ne sature pas puisqu'il utilise le type « double » *i.e.* en points flottants. C'est d'ailleurs à cela que l'on peut imputer les différences

entre les deux courbes. En effet, en utilisant l'AD, nous avons une précision limitée par le nombre de digits situés derrière la virgule.

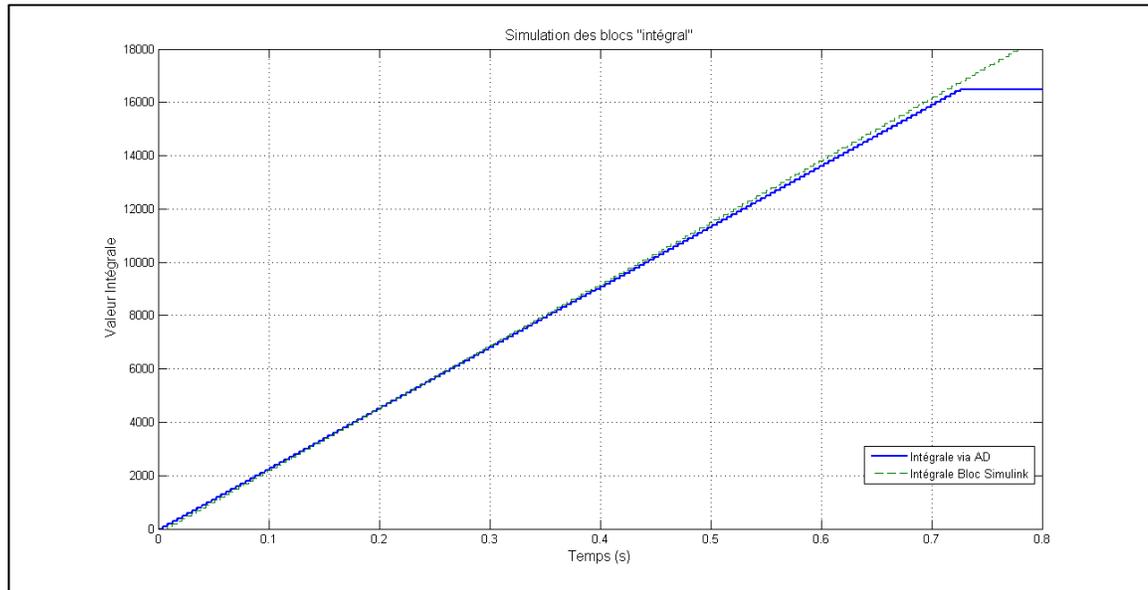


Figure 5.12 Courbes de simulation pour l'action intégrale

5.3.3 Simulation de l'action dérivée

L'action dérivée, nous l'avons vu, n'est pas uniquement un simple dérivateur mais un filtre qui permet de ne pas générer trop de bruit sur le système. Nous comparerons la simulation du bloc dérivée à architecture AD avec le résultat de simulation d'une fonction de transfert discrète équivalente. Le troisième terme de l'équation (2.39) sert à l'implémentation de cette fonction de transfert. Comme dans le cas précédent de l'intégrale, nous avons positionné des blocs permettant d'adapter le temps de simulation. Le bloc dérivé à AD dispose de mots d'une longueur de 27 bits. La sérialisation implique que la sortie du bloc ne s'actualise que tous les 27 temps de simulation. La Figure 5.13 présente l'implémentation de test en simulation.

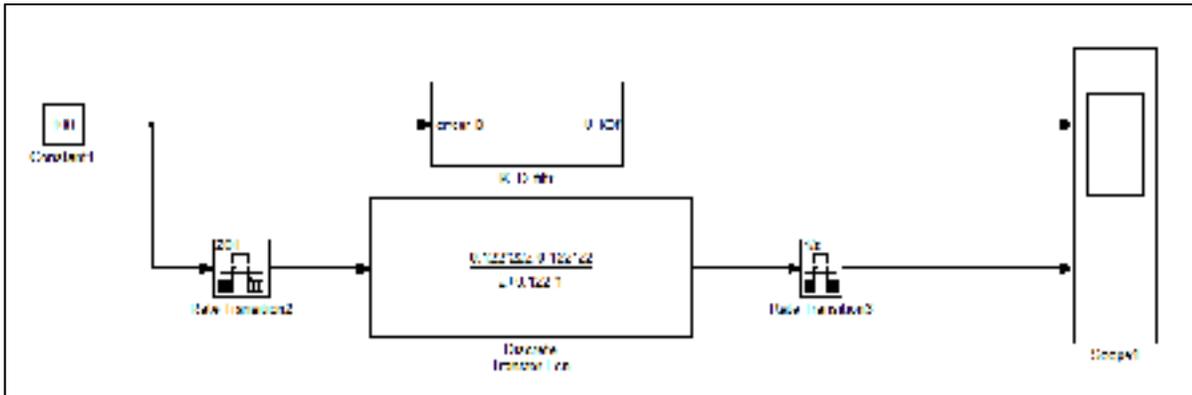


Figure 5.13 Schéma de Simulation blocs "dérivé"

Nous avons pu obtenir les résultats présentés par la Figure 5.14 :

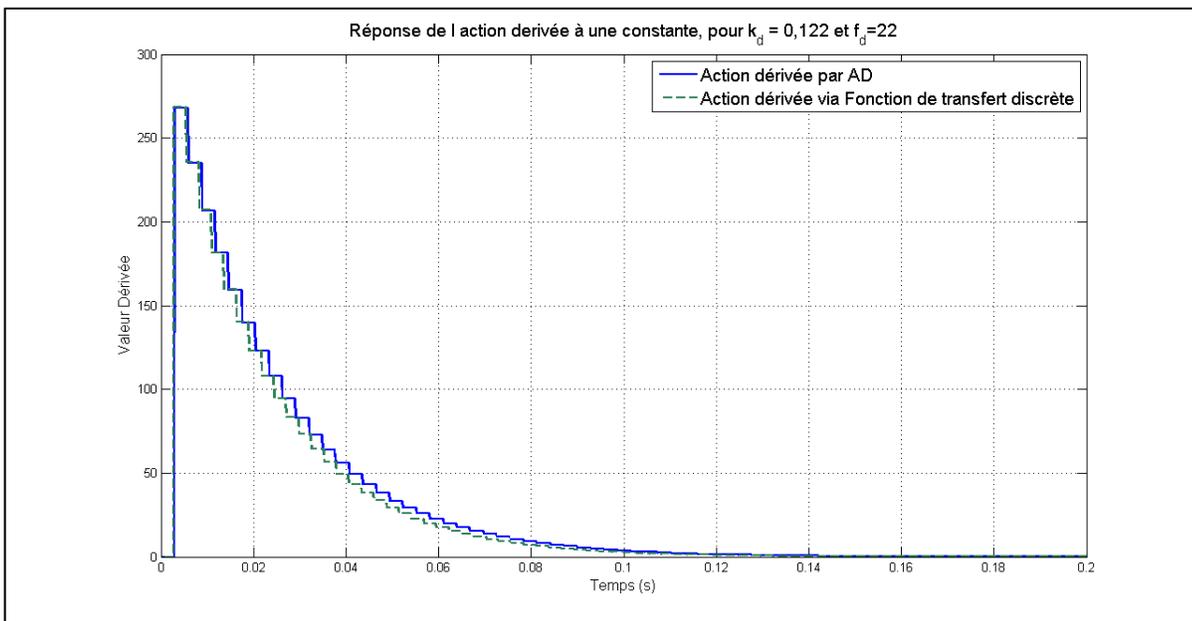


Figure 5.14 Résultats de simulation pour l'action dérivée

On constate que la valeur initiale est la même pour les deux systèmes. On distingue que les réponses sont très proches. Les différences entre les deux courbes s'expliquent par la nature des variables utilisées pour le calcul. Nous disposons dans le bloc soumis à l'AD de variables

à virgules fixes avec une taille limitée. Le bloc fonction de transfert utilise quant à lui des variables à virgules flottantes (*double*).

Nous avons pu démontrer que les blocs implémentés à l'aide de l'AD ont un comportement cohérent. D'ailleurs nous avons été en mesure de reproduire leur dynamique à l'aide de blocs usuels.

5.4 Simulation du système en boucle fermée

Dans cette partie, nous allons montrer nos résultats de simulation pour le système en boucle fermée.

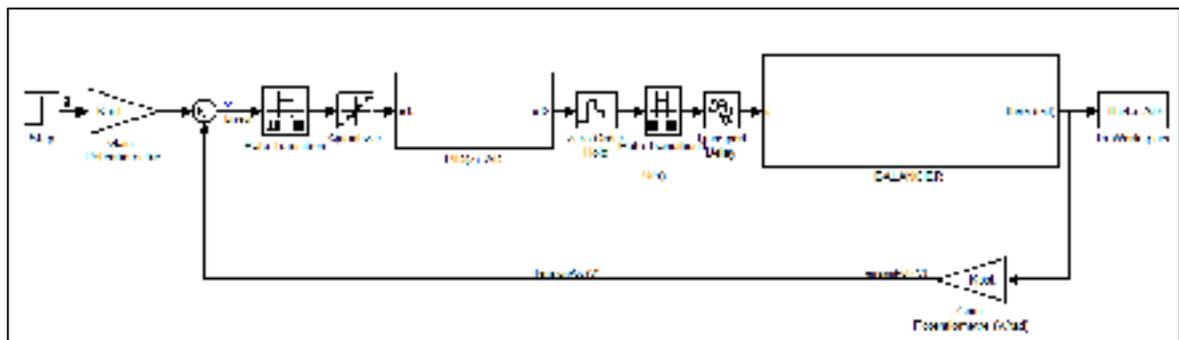


Figure 5.15 Schéma Simulink du système muni d'un PID à AD

La Figure 5.15 présente le schéma de simulation du système en boucle fermée. Le correcteur mis en place est un PID réalisé à l'aide de l'AD. Gagnon (2011) avait mis en place cette simulation en utilisant un PID continu (à l'aide d'un bloc Simulink). L'introduction des blocs « rate transition » permet de simuler le système réel. En effet, ils permettent d'avoir deux périodes de simulation distinctes. Une plus rapide pour le PID implémenté dans le FPGA, et l'autre, plus lente, correspondant au balancier. Notons également que les gains implémentés dans le PID à AD sont ceux correspondant à l'expression (5.13). En effet, dans le schéma de simulation proposé on ne tient pas compte de la présence des convertisseurs (CAN et CNA) et circuits d'adaptation (CA1 et CA2), la formule (4.32) est donc ignorée.

Les résultats de la simulation issus de la Figure 5.15 sont présentés par la Figure 5.16 . On compare les réponses du modèle de simulation utilisé par Gagnon (2011) (PID continu) au nôtre. On constate des allures de courbes sensiblement égales. Le Tableau 5.3 présente les différents temps mis en jeu et les écarts relatifs des deux modèles. Pour rappel, nous avons donné la définition de ces grandeurs dans la partie 2.2.

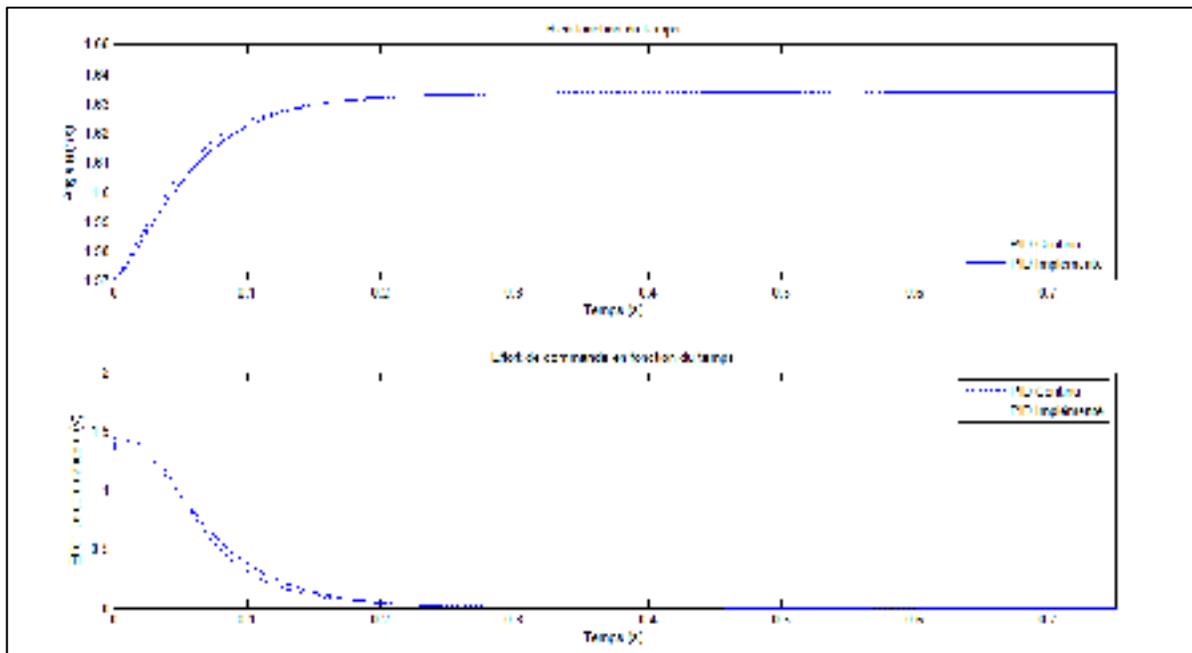


Figure 5.16 Résultats de simulation pour deux types PID modélisés (continu et AD)

Tableau 5.3 Comparaison des grandeurs caractéristiques de la réponse entre PID continu et à AD

	PID Continu	PID à AD	Erreur relative (%)
Temps de montée T_m (s)	0,114	0,1162	1,93
Temps de réponse T_r (s)	0,2337	0,2056	11,9
Temps de dépassement T_d (s)	X	0,6081	X
Dépassement %d (%)	X	0,1	X

Le Tableau 5.3 met en évidence de légères différences obtenues lors de la simulation du système à l'aide d'un régulateur continu et d'un PID à AD. Toutefois, notons que les aspects dynamiques sont acceptables. Le temps de montée est relativement proche dans les deux cas. L'erreur relative dépasse 10% pour le temps de réponse, on peut imputer cela au fait que le PID à AD traite des valeurs numériques finies, en point fixe, alors que le PID continu dispose d'une plage de valeurs numériques plus importante. En effet, il est codé à l'aide du type *double* défini en virgule flottante sur 64 bits. De plus, l'effort de commande est plus grand pour le PID à AD entre 0,05 et 0,15s ce qui réduit le temps de réponse.

5.5 Résultats expérimentaux

Nous avons alors mis en œuvre notre module PID au sein du FPGA. La Figure 5.3 nous a indiqué le schéma synoptique destiné à la synthèse dans notre FPGA. Les résultats des étapes de synthèse et d'implémentation du correcteur sont présentés par le Tableau 5.4 **Erreur ! Source du renvoi introuvable.** Ainsi, le PID utilisant l'AD occupe moins de 20% des tranches du FPGA.

Tableau 5.4 Résultats de la synthèse du PID implémenté avec AD issus du logiciel ISE

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	577	11,775	4%
Number of Output LUTs	2,196	11,775	18%
Number of occupied Slices	1,147	5,888	19%
Number of Slices containing only related logic	1,147	1,147	100%
Number of Slices containing unrelated logic	0	1,147	0%
Total Number of Input LUTs	2,236	11,775	19%
Number used as logic	2,196		
Number used as a route-thru	22		

Si l'on décide d'implémenter au sein du FPGA le système représenté par la Figure 5.3 sans la partie PID, à savoir uniquement les modules d'E/S on obtient l'utilisation des ressources présentées par la Tableau 5.5 :

Tableau 5.5 Utilisation des ressources du FPGA pour l'implémentation des modules d'entrées/sorties issus du logiciel ISE

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	
Number of Slice Flip Flops	23	11,776	0%	
Number of Output LUTs	256	11,776	2%	
Number of occupied Slices	128	6,080	2%	
Number of Slices containing only related logic	128	128	100%	
Number of Slices containing unrelated logic	0	128	0%	
Total Number of 4-input LUTs	256	11,776	2%	
Number used as logic	256			
Number used as carry bits	0			

La mise en œuvre du régulateur PID par AD sur le montage expérimental donne les résultats présentés par la Figure 5.17.

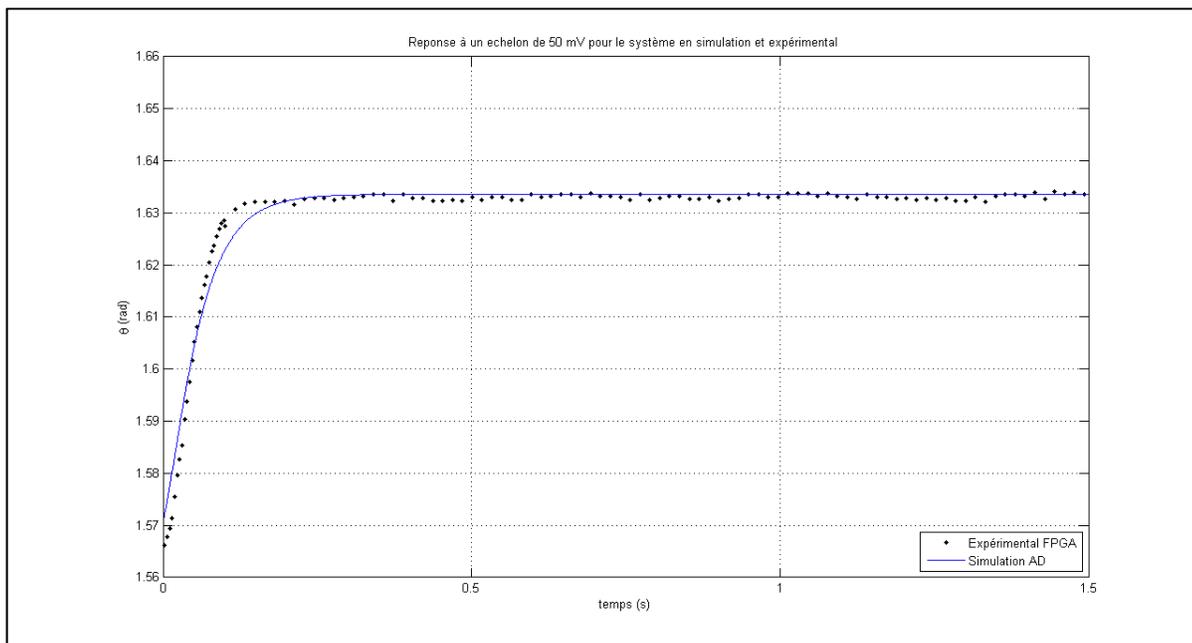


Figure 5.17 Réponse simulée et expérimentale du système

Tableau 5.6 Comparaison des grandeurs caractéristiques entre le PID à AD simulé et implémenté

	PID AD implémenté	PID AD simulé	Erreur relative (%)
Temps de montée T_m (s)	0,072	0,1162	38
Temps de réponse T_r (s)	0,203*	0,2056	1,2
Temps de dépassement T_d (s)	0,667**	0,6081	9,7
Dépassement %d (%)	1,9**	0,1	1800**

Le Tableau 5.6 montre les temps caractéristiques de la réponse expérimentale et simulée. Il est toutefois important de nuancer plusieurs éléments se trouvant dans le tableau.

Le temps de réponse pour le module implémenté (*) a été déduit en effectuant une moyenne glissante (ou mobile) de la réponse (sur dix échantillons). En effet, la réponse étant bruité il n'était pas possible de définir ce temps. Notons toutefois que le temps ainsi présenté permet de donner une idée de la valeur du temps de réponse mais le moyennage agit sur la dynamique. Ainsi, on considérera seulement ce temps comme indicatif.

Par ailleurs, les temps de dépassement et pourcentage de dépassement (**) trouvés ne sont que donnés à titre indicatif. En effet, le bruit sur la réponse ne permet pas de disposer de résultats fiables.

L'échelon de tension (consigne) a une valeur de 50 mV. Celui-ci a pour effet d'avoir en sortie une variation angulaire de 0,0628 rad. La bande à $\pm 2\%$ définie autour du régime établi est donc de 2mV ou encore ± 1 mV. On comprend alors que le temps de réponse est difficilement caractérisable puisqu'une variation supérieure (en valeur absolue) à 1mV autour de la valeur de régime statique suffit à sous entendre que le système n'est pas en régime établi.

Nous pouvons avancer les raisons suivantes pour le bruit trouvé :

- la résolution des convertisseurs notamment le CNA, qui dispose d'une résolution de 0,8 mV (d'après l'expression (4.21)) ;
- les circuits d'adaptation (CA1 et CA2) dont les jeux de résistances disposent d'une tolérance à 5% ;
- le balancier qui peut avoir un jeu axial (pivot) ;
- la précision du capteur angulaire.

Par ailleurs, il peut être intéressant de regarder l'implémentation du régulateur PID à AD sur un FPGA comparativement à celui proposé par Gagnon (2011) qui a mis en place un correcteur à l'aide de XPC Target¹². Le Tableau 5.7 propose de comparer ces résultats. On remarque que les temps de montée sont relativement proches. Nous avons gardé le temps de réponse présent dans le Tableau 5.6 puisqu'il permet, comme nous l'avons souligné, de donner une indication. Par ailleurs, les termes liés au dépassement ne figure plus dans les données puisque nous avons vu que les données obtenues n'étaient pas suffisamment fiables. Les allures des réponses sont proposées à la Figure 5.18

Tableau 5.7 Comparaison des caractéristiques entre implémentation via xPC Target et FPGA.
Adapté de Gagnon (2011, p 109)

	PID à AD FPGA	PID xPC Target (Gagnon, 2011)	Erreur relative (%)
Temps de montée T_m (s)	0,072	0,066	9
Temps de réponse T_r (s)	0,203	0,153	32
Temps de dépassement T_d (s)		0,136	
Dépassement %d (%)		1,1	

¹² Il s'agit d'une boîte à outils de Matlab/Simulink qui permet la génération d'un code C à partir d'un modèle Simulink qui peut être utilisé par un PC qui démarre sur le code et joue le rôle de correcteur.

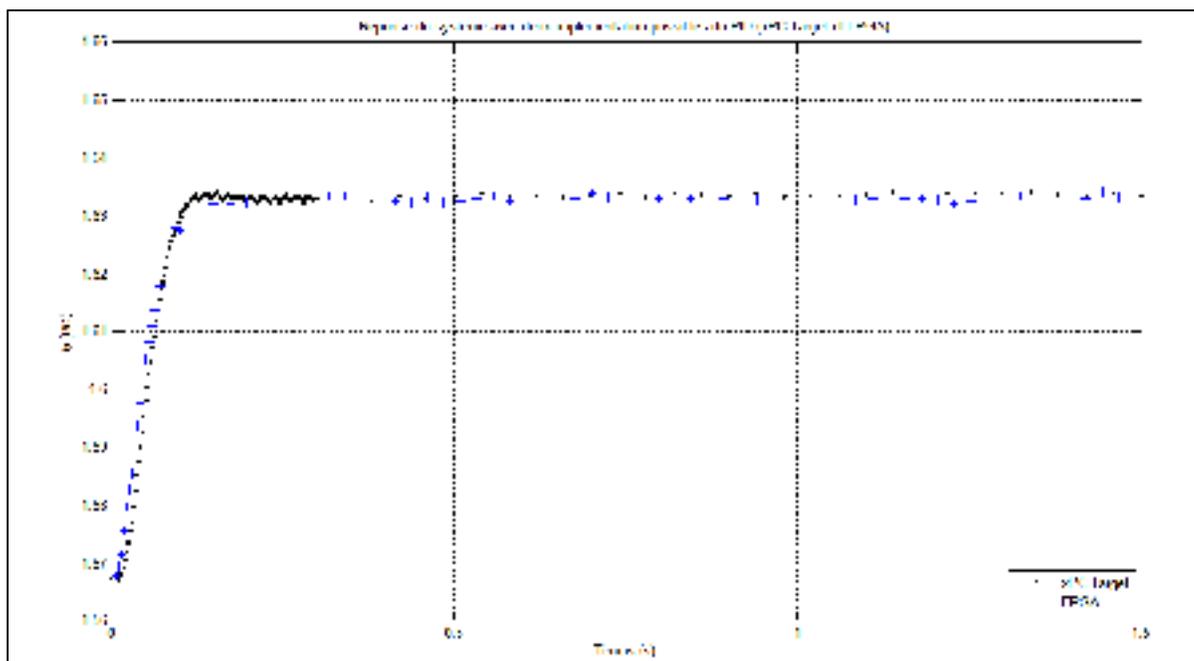


Figure 5.18 Réponse du système pour deux types d'implémentation de PID (xPC et FPGA)
Adaptée de Gagnon (2011, p 109)

CHAPITRE 6

IMPLÉMENTATION DE LA COMMANDE DANS L'ESPACE D'ÉTAT

Nous allons à présent décrire la conception d'une commande dans l'espace d'état. La démarche retenue consiste à présenter les modèles par complexité croissante. Nous partons d'un modèle Simulink de base jusqu'à une modélisation plus complète du système embarqué.

6.1 Simulation du modèle simplifié

Afin de valider notre représentation, nous avons modélisé de façon élémentaire notre système. La modélisation s'effectue à l'aide de blocs Simulink existants. On considère l'ensemble du système comme continu. La Figure 6.1 donne le schéma de simulation.

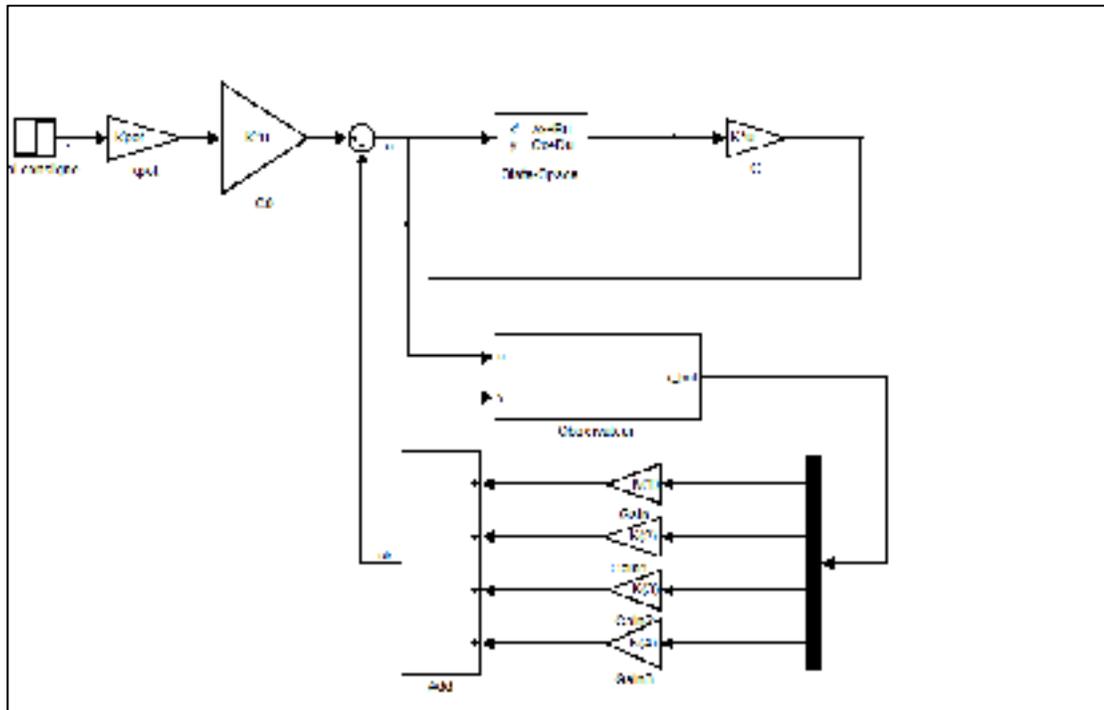


Figure 6.1 Schéma de simulation de la commande par retour et observateurs d'état

Bien entendu, le système à commander est décrit par le système (ABCD) présenté par les expressions (4.46)-(4.48). Sur la Figure 6.1 on distingue le bloc observateur, qui permet de sortir les états estimés. Le bloc observateur est représenté à la Figure 6.2. Sa construction se réalise à l'aide d'une représentation d'état comme présenté à l'équation (2.78).

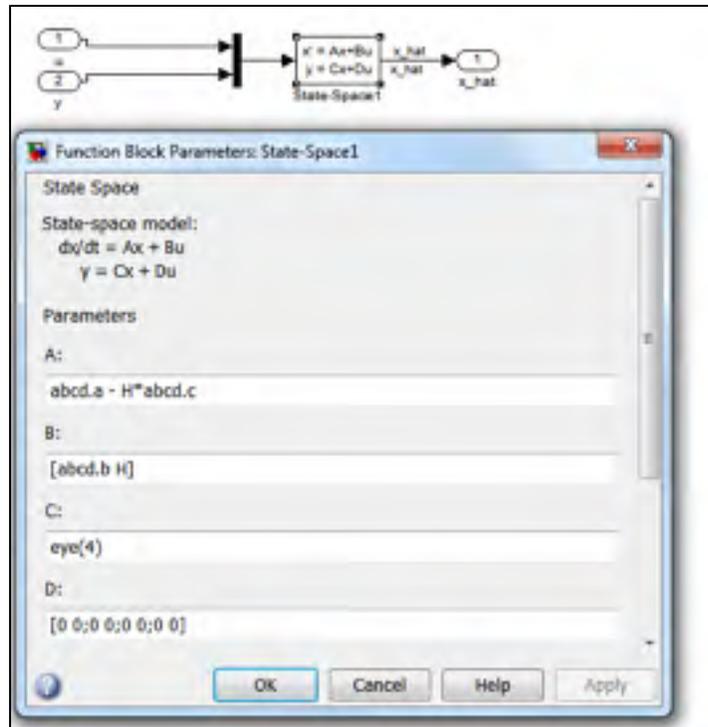


Figure 6.2 Description de l'observateur continu

Le retour d'état est appliqué sur ces états observés. Notons également la mise en place d'un gain (noté G_0) qui permet conformément à l'expression (2.89) d'avoir une sortie conforme à la consigne.

Il faut alors définir les valeurs des matrices H (observateur) et K (retour d'état).

Les pôles de l'observateur sous forme conjuguée sont :

$$p_{obsv_{1,2}} = -500 \pm 10j \quad (6.1)$$

$$p_{obsv_{3,4}} = -300 \pm 5j$$

Ces pôles permettent à l'observateur une estimation rapide des états. La Figure 6.3 montre l'estimation des états. Nous avons vu que les états du système ne correspondent pas réellement à une grandeur physique. Toutefois, on note que l'état x_4 est à rapprocher de la tension de sortie (à la matrice C près) ou encore à l'angle θ (à la matrice C et gain du capteur de position près). Il est donc cohérent de trouver l'allure de l'état x_4 présenté dans la Figure 6.3. Par ailleurs, les autres états (x_1, x_2, x_3) sont les dérivées successives de l'état x_4 comme nous l'avons montré (2.4.1). Ainsi, en régime établi, la position et donc la tension est stabilisée. Ce qui implique que les variations (dérivés) sont nulles. Sur la Figure 6.3 on constate que les états x_1, x_2 et x_3 convergent vers 0. Nous avons donc un observateur fonctionnel.

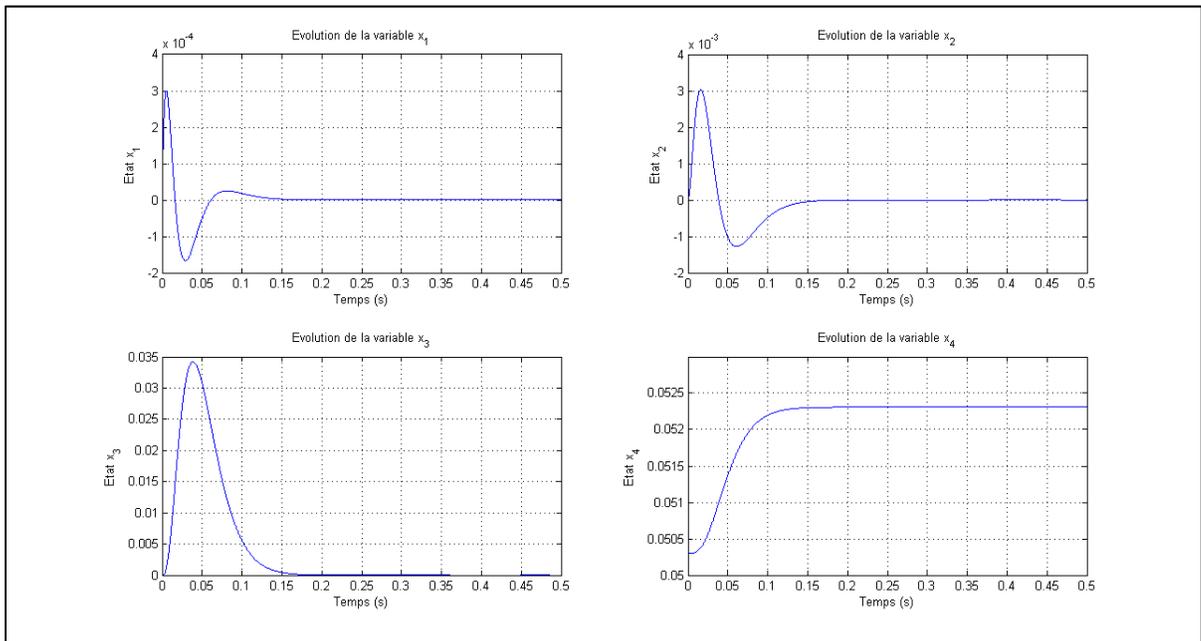


Figure 6.3 Estimation des états pour le modèle continu

Pour le retour d'état on choisit des pôles dont la valeur réelle est 5 fois inférieure à celle des pôles de l'observateur :

$$p_{K_{1,2}} = -100 \pm 10j \quad (6.2)$$

$$p_{K_{3,4}} = -60 \pm 5j$$

La réponse du système est présentée à la Figure 6.4.

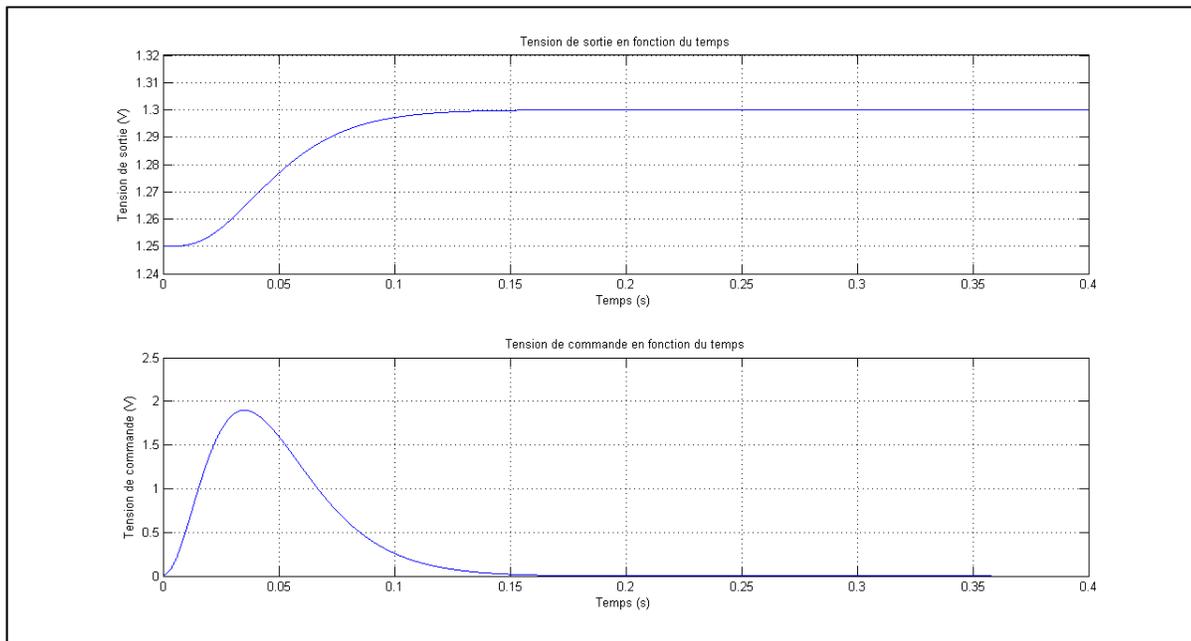


Figure 6.4 Tension de sortie et de commande du système modélisé en continu

Le système présente une réponse sans dépassement dont les temps caractéristiques sont donnés dans le Tableau 6.1. Par ailleurs, on constate que la tension de commande est inférieure à 2V, l'actionneur n'est donc pas saturé.

Tableau 6.1 Temps caractéristiques de la commande continue

	Commande continue
Temps de montée T_m (s)	0,04
Temps de réponse T_r (s)	1,29
Temps de dépassement T_d (s)	X
Dépassement %d (%)	X

Cependant, la commande continue ne peut s'implémenter directement dans un appareil numérique qui traite nécessairement des valeurs discrètes. Il faut alors discrétiser le module de commande, à savoir l'observateur et le retour d'état.

6.2 Simulation du système à commande discrète

Nous l'avons vu les éléments proposés dans la précédente simulation étaient tous continus. Il ne faut alors pas perdre de vue l'objectif d'implémentation au sein d'un régulateur numérique. Le passage de l'organe de commande du continu au discret ne se fait pas de façon triviale. En effet, il est d'abord nécessaire de disposer d'un modèle discret d'un procédé à commander. Il faut alors discrétiser le système (ABCD) continu.

6.2.1 Discrétisation du procédé

Nous avons évoqué dans la partie (2.4.1) la discrétisation d'un modèle décrit dans l'espace d'état. Nous avons indiqué la fréquence d'échantillonnage (équation (5.14)). Nous opérons la discrétisation avec le logiciel Matlab par la commande *c2d*. On obtient le système discret suivant :

$$A_d = \begin{pmatrix} 0,991 & -0,3353 & -0,0214 & 0 \\ 0,025 & 0,9996 & -2,69 \cdot 10^{-4} & 0 \\ 1,5703 \cdot 10^{-4} & 0,0125 & 1 & 0 \\ 1,28 \cdot 10^{-9} & 1,54 \cdot 10^{-7} & 2,45 \cdot 10^{-5} & 1 \end{pmatrix} \quad (6.3)$$

$$B_d = \begin{pmatrix} 3,905 \cdot 10^{-3} \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$C_d = C$$

Notons que l'approximation par série de Taylor en $o(T_E^2)$ définie à l'expression (2.75) donne des résultats identiques à ceux présentés à l'expression (2.74) à 10^{-3} près. Toutefois, afin de disposer d'un modèle le plus précis possible nous conserverons les résultats obtenus par Matlab.

6.2.2 Système de commande discret

A présent, nous disposons d'un modèle discrétisé. Ce modèle va servir à mettre en place l'observateur d'état et le retour d'état. En utilisant la relation (2.11) de passage entre le domaine de Laplace et des Z , on peut définir les pôles discrets de l'observateur et du retour d'état. Les pôles de l'observateur, sous forme conjuguée, sont :

$$\begin{aligned} p_{obsv_{d_{1,2}}} &= 0,9878 \pm 0,0002j \\ p_{obsv_{d_{3,4}}} &= 0,9927 \pm 0,0001j \end{aligned} \quad (6.4)$$

Les pôles du système muni du retour d'état, sous forme conjuguée, sont :

$$\begin{aligned} p_{Kd_{1,2}} &= 0,9985 \pm 0,0002j \\ p_{Kd_{3,4}} &= 0,9976 \pm 0,0001j \end{aligned} \quad (6.5)$$

Le système simulé est présenté à la Figure 6.5.

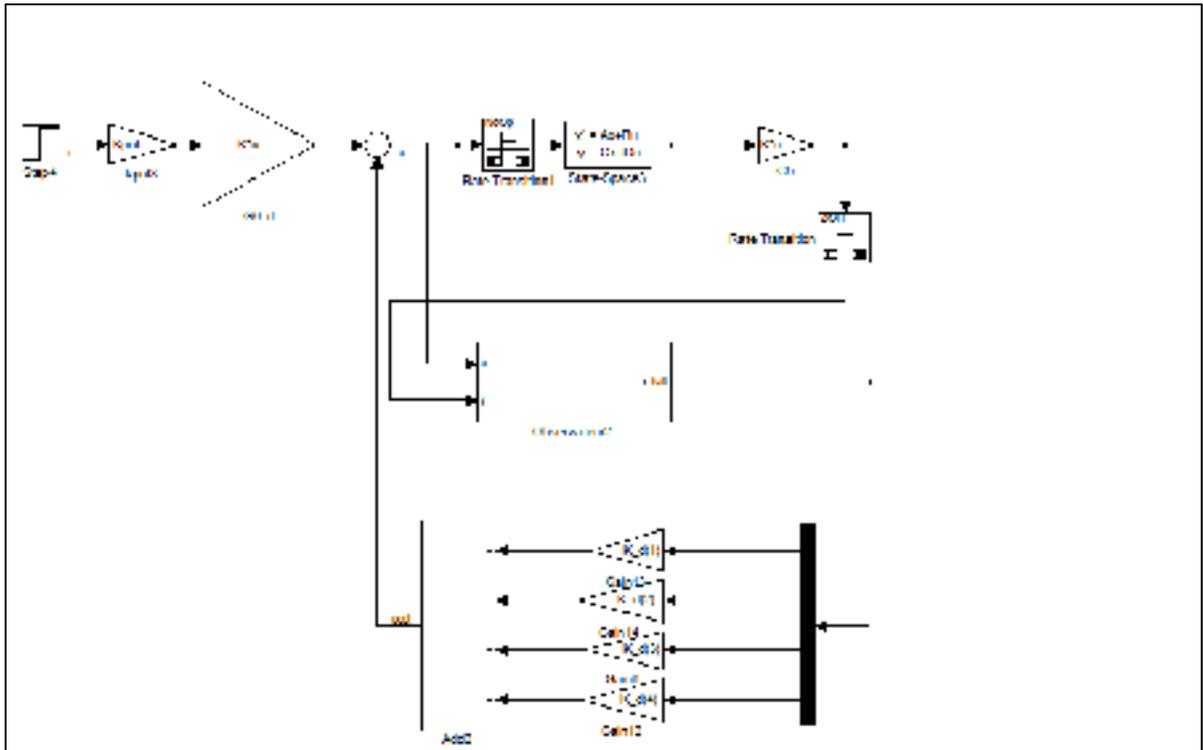


Figure 6.5 Schéma Simulink du système à commande discrète

Notons que le gain statique G_{0d} est modifié par rapport à la première simulation. En effet, il faut prendre en compte la matrice de gains K_d . La matrice K de l'expression (2.83) est donc remplacée par la matrice des gains K_d .

Le bloc observateur est décrit à la Figure 6.6 . Pour le modéliser, nous avons utilisé un bloc Simulink de représentation dans l'espace d'état pour un système discret. La matrice C est ajustée sur une matrice identité ce qui permet de sortie les états du système. En effet, notre choix d'implémentation ne fait pas intervenir la sortie estimée mais uniquement les états observés¹³.

¹³ A partir de l'expression (2.77) il est possible de donner une autre implémentation faisant intervenir l'erreur d'estimation de la sortie. Toutefois, cette implémentation fait intervenir un produit matriciel et une soustraction en plus.

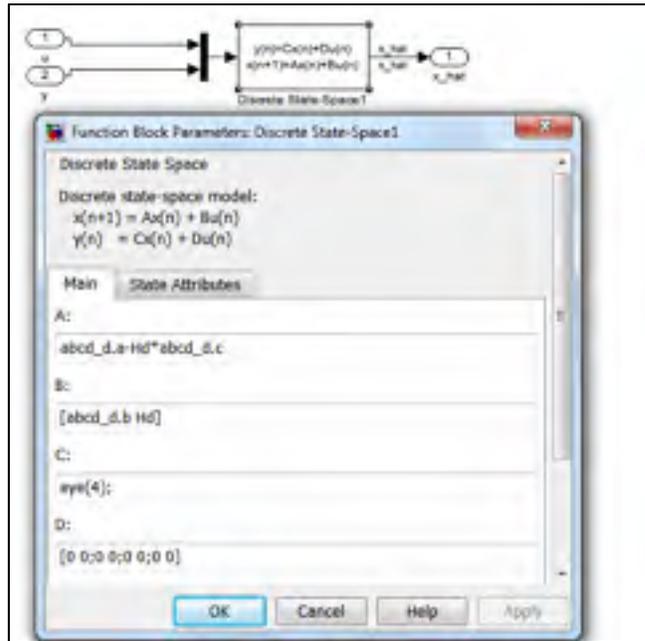


Figure 6.6 Implémentation de l'observateur discret

Nous obtenons alors la réponse du système présenté par la Figure 6.7 . Le Tableau 6.2 donne les caractéristiques de la réponse.

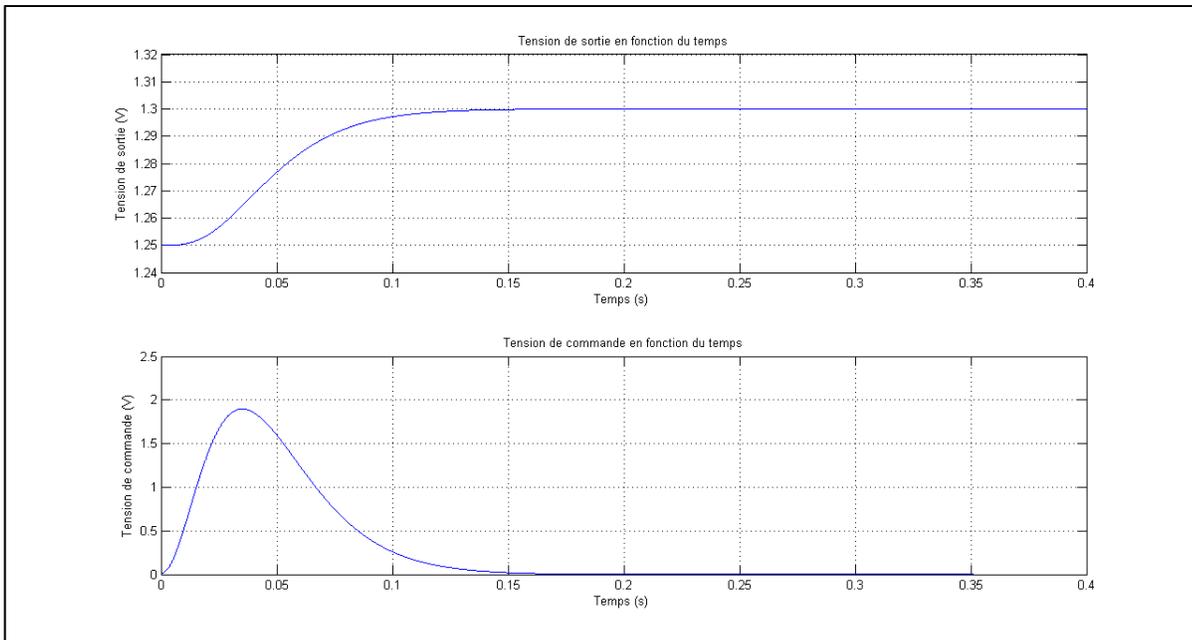


Figure 6.7 Tension de sortie et de commande du système à commande discrète

Tableau 6.2 Temps caractéristiques du système à commande discrète

	Commande continue
Temps de montée $T_m(s)$	0,04
Temps de réponse $T_r(s)$	1,29
Temps de dépassement $T_d(s)$	
Dépassement %d (%)	

On constate que les Tableau 6.1 et Tableau 6.2 donnent les mêmes résultats. Le passage au système à commande discrète est donc réalisé avec succès. On a pu vérifier que les formes d'ondes des tensions de commande et de sortie sont équivalentes dans les deux cas. À présent il est temps de porter notre effort sur la réalisation d'un système de commande discret mais destiné à être synthétisé sur un FPGA.

6.2.3 Système de commande synthétisable

Un critère important pour un système destiné à être synthétisé est qu'il doit être défini dans une arithmétique à point fixe. Ainsi pour modéliser des valeurs définies en point fixe il nous faut utiliser la boîte à outils de Simulink, *Fixed-point Toolbox*. Le schéma de simulation est proposé par la Figure 6.8. On dispose toujours d'un système dont la commande est modélisée par une représentation d'état continue. Cependant, à la différence du modèle précédent, l'observateur d'état est implémentable sur un système numérique puisque qu'il traite des valeurs en point fixe. Le retour d'état par la matrice de gains K s'effectue également en point fixe. C'est d'ailleurs à cet effet que l'on trouve dans la Figure 6.8 un certain nombre de blocs de conversion permettant d'assurer la compatibilité entre les systèmes continu et discret.

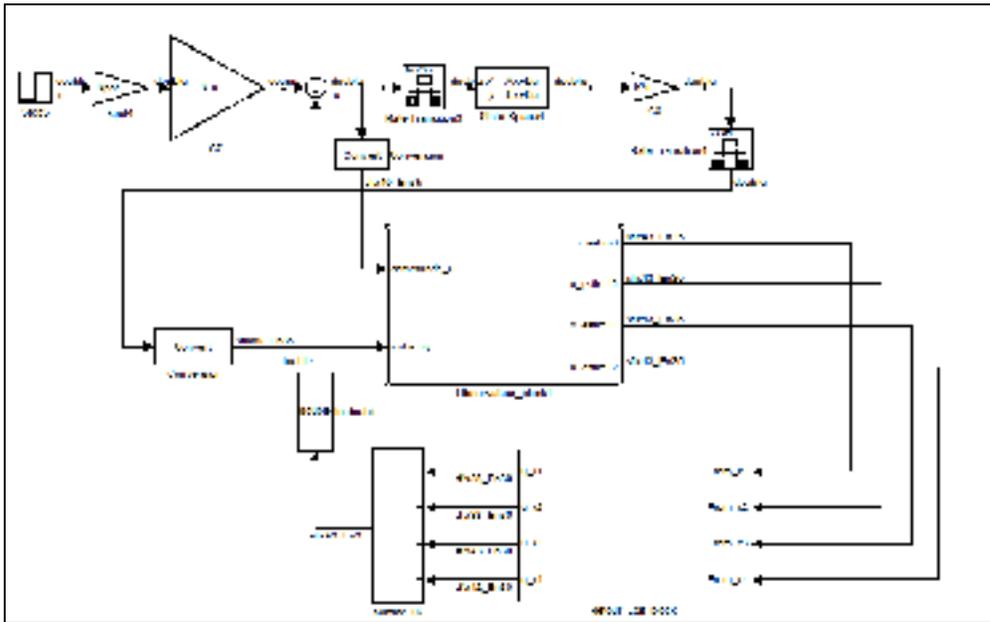


Figure 6.8 Schéma Simulink du système pour une commande synthétisable

L'observateur est décrit dans la Figure 6.9. Celle-ci montre un système muni des deux mêmes entrées que les observateurs montrés précédemment. Toutefois, les entrées (u et y) sont discrétisée et codées en point fixe. La sortie du bloc présente les quatre états. Le bouclage des sorties sur les entrées fait partie de l'algorithme pour un système discret (décrit par l'expression (2.76)). Le bloc « *observateur_emf* » est une fonction embarquée Matlab. Au sein de cette fonction, nous définissons des constantes (les coefficients des matrices) et effectuons les multiplications matricielles. Celles-ci sont réalisées ligne à ligne puisque les matrices ne sont pas gérées par HDL Coder (Matlab/Simulink, 2007). Nous avons donc quatre sommes de six produits, ainsi pour l'état estimé \hat{x}_i nous avons :

$$\hat{x}_i(k) = a_{i1}\hat{x}_1(k-1) + a_{i2}\hat{x}_2(k-1) + a_{i3}\hat{x}_3(k-1) + a_{i4}\hat{x}_4(k-1) + b_i u + h_i y \quad (6.6)$$

où les coefficients $(a_{ij})_{\substack{1 \leq i \leq 4 \\ 1 \leq j \leq 4}}$ représentent la matrice $\mathcal{A}_{obv} = A_d - H_d C$.

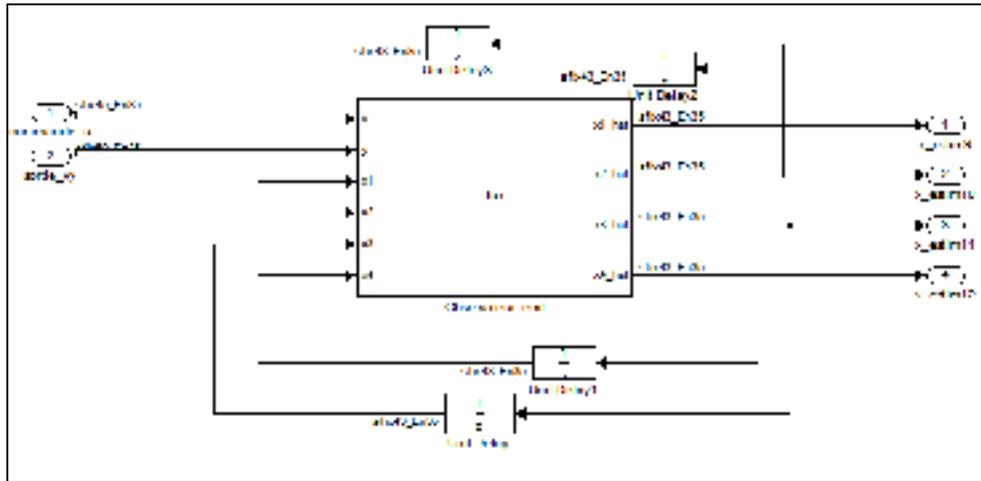


Figure 6.9 Schéma Simulink de l'observateur d'état synthétisable

La Figure 6.10 présente le retour d'état synthétisable. Nous avons défini les gains de la matrice K en point fixe ainsi que les sorties.

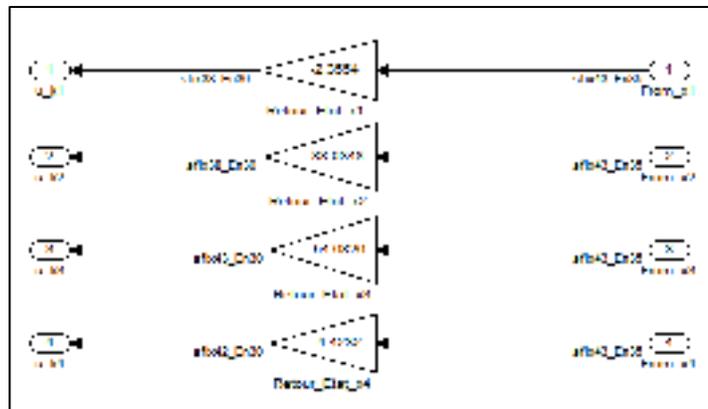
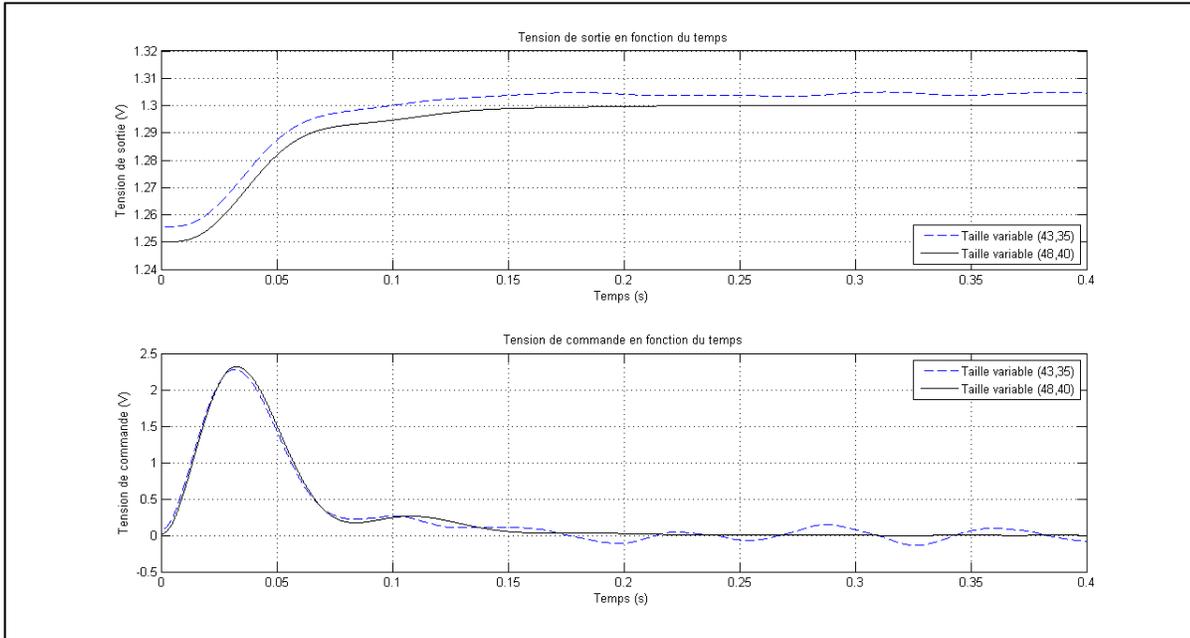


Figure 6.10 Retour d'état synthétisable

On dispose ainsi d'un système synthétisable à partir d'un code HDL généré à l'aide de HDL Coder. Pour tester cette implémentation, nous avons effectué deux simulations. La première (qui correspond aux figures présentées) met en jeu des variables et constantes à point fixe définies sur 43 bits dont 35 après la virgule, que l'on note (43,35). La deuxième simulation met en jeu des variables et des constantes à point fixe définies en (48,40), soit 40 bits après la

virgule. On se propose de comparer les résultats de simulation entre les deux précisions offertes. La Figure 6.11 présente ces résultats.



On constate que pour une précision de 35 bits après la virgule les signaux en jeu sont fortement bruités. On observe également un bruit sur les signaux codés avec 40 bits après la virgule. Toutefois, l'ajout de 5 bits permet de diminuer fortement l'imprécision du système. Ajoutons que pour le système le moins précis, il y a une erreur statique.

Tableau 6.3 Comparaison des temps caractéristiques entre deux précisions de commande synthétisable

	Commande synthétisable (43,35)	Commande synthétisable (48,40)	Erreur relative (%)
Temps de montée T_m (s)	0,041	0,04	2,5
Temps de réponse T_r (s)	1,305	1,29	1,2
Temps de dépassement T_d (s)	X	X	X
Erreur statique (%)	2	X	X

A présent nous souhaitons vérifier la synthèse d'un tel module de commande. Le Tableau 6.4 présente le résultat de synthèse obtenu pour la commande utilisant 35 bits après la virgule.

Tableau 6.4 Résultats de synthèse pour la commande synthétisable avec précision (43,35) issu du logiciel ISE

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(%)
Number of Slice Flip Flops	177	11,770	1%	
Number of Input LUTs	17,810	11,770	151%	OVERWRITTEN
Number of occupied Slices	10,257	5,888	171%	OVERWRITTEN
Number of Slices containing only related logic	10,257	10,257	100%	
Number of Slices containing unrelated logic	0	10,257	0%	
Total Number of Input LUTs	20,277	11,770	171%	OVERWRITTEN
Number used as logic	17,810			
Number used as route thru	2,567			
Number of MULT18K1010s	9	20	45%	

La synthèse de la commande la moins précise produit un dépassement des capacités du FPGA. Notons que nous avons 28 produits à effectuer. Nous avons laissé l'outil de synthèse gérer et optimiser les ressources. Ainsi, 9 des 20 multiplicateurs sont requis. Les LUT et les tranches ne sont pas suffisamment nombreuses pour implanter la commande. En effet, plus de 20000 LUT sont requises tandis que l'on en dispose d'un peu plus de 11000. Bien entendu, la synthèse du module de commande à la précision (48,40) eut générée une demande de ressources supérieures.

Il convient alors d'employer une méthode d'implémentation facilitant la gestion des ressources. Le principe d'AD est un bon candidat puisqu'on a vu qu'il permettait d'économiser les ressources.

6.2.4 Système de commande à AD

On se propose maintenant d'implanter un calcul par AD au sein de l'observateur afin de réduire l'utilisation des ressources induites par le calcul de 24 produits (4 sommes de 6 produits).

Nous avons pu le voir précédemment, l'AD permet d'utiliser des opérations élémentaires afin de réaliser des sommes de produits. Cette technique induit un temps de calcul plus long dû à la sérialisation des données. Dans le système présenté ci-après, nous avons implémenté l'AD à partir de la commande précédente définie à la précision (43,35) et (48,40). Nous en avons définie une nouvelle précision (58,50). Il y a ainsi un flux de 43 bits (respectivement 48 et 58 bits) en série. En tenant compte des délais, dus à la construction du modèle, la latence entre le début de l'algorithme et le résultat est de 45 coups d'horloge (respectivement 50 et 60 coups d'horloge). La Figure 6.12 présente la modèle de simulation de l'observateur.

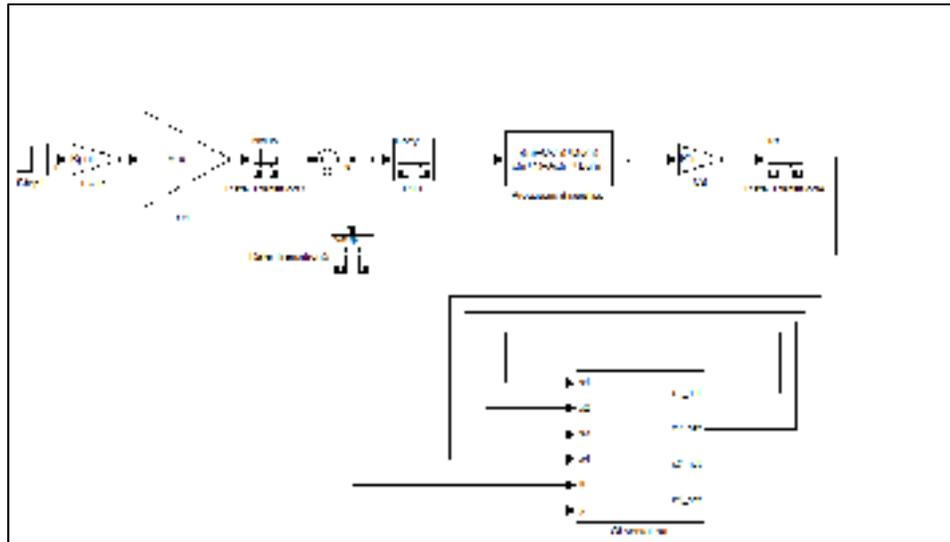


Figure 6.12 Schéma Simulink de l'observateur à AD

Les blocs de transition (Rate Transition) permettent de passer d'une période de simulation à une autre. On dispose ainsi d'éléments s'actualisant à des fréquences différentes. Par ailleurs, nous avons mis en place une représentation d'état discrète pour le processus à commander. En effet, ce bloc nous permet une plus grande liberté pour la gestion des périodes de simulation. Ainsi pour une période de simulation notée, T_s , nous avons une seconde période de simulation, \tilde{T}_s , présente dans le modèle telle que :

$$\tilde{T}_s = \text{latence} \times T_s \quad (6.7)$$

Le bloc observateur va donc disposer de la période T_s tandis que le reste des éléments s'actualise 45 (respectivement 50 et 60) fois moins rapidement (\tilde{T}_s).

6.2.4.1 Construction de l'observateur par AD

Décrivons maintenant la construction de l'observateur à AD. Comme réalisé dans la commande précédente, on décompose le calcul matriciel sur chaque ligne conformément à l'expression (5.78). Le bloc observateur se décompose en plusieurs étages. À l'entrée, on

convertit toutes les variables en jeu selon un format à point fixe. Ensuite on met en place le calcul par AD. Notons que l'on a deux traitements par AD par ligne. Le premier effectue le calcul pour les variables d'état, le second calcul la somme des produits mise en jeu par l'entrée et la sortie. Cela permet de réduire la taille des LUT conformément à la formule (5.9). Enfin en sortie, on somme les résultats des deux calculs afin d'obtenir l'estimation de l'état. Le détail de ce bloc est montré à la Figure 6.13.

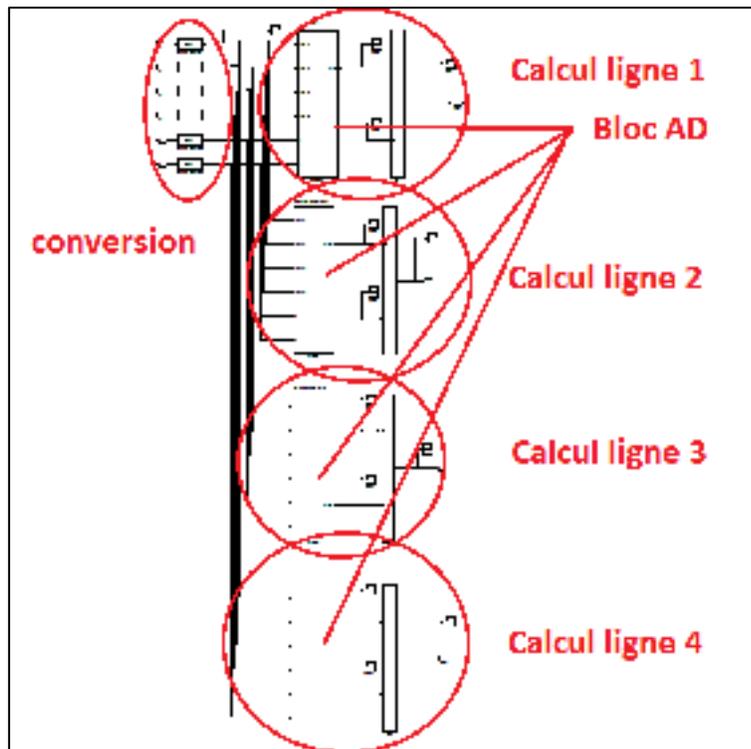


Figure 6.13 Structure de l'observateur à AD

Il convient de montrer l'intérieur du bloc AD référencé sur la figure précédente. Celui-ci ressemble plus ou moins à celui implémenté dans le cas du PID. Toutefois, il est plus complexe puisqu'il y a deux calculs par AD sur chacune des lignes qui s'effectuent en parallèle comme l'indique la Figure 6.14. On retrouve en effet les caractéristiques de l'implémentation du calcul par AD. Nous avons en premier lieu la sérialisation des données par les registres PISO. Le flux de bits permet alors l'adressage de la table d'interpolation (LUT) par concaténation. La valeur correspondant à l'adresse est alors envoyée à l'accumulateur qui permet le calcul par décalages successifs. Notons que les valeurs

présentent dans la LUT sont générées à l'aide d'un script Matlab. Cela permet de disposer d'un modèle réutilisable. Nous aurons l'occasion de revenir sur cela dans le chapitre à venir.

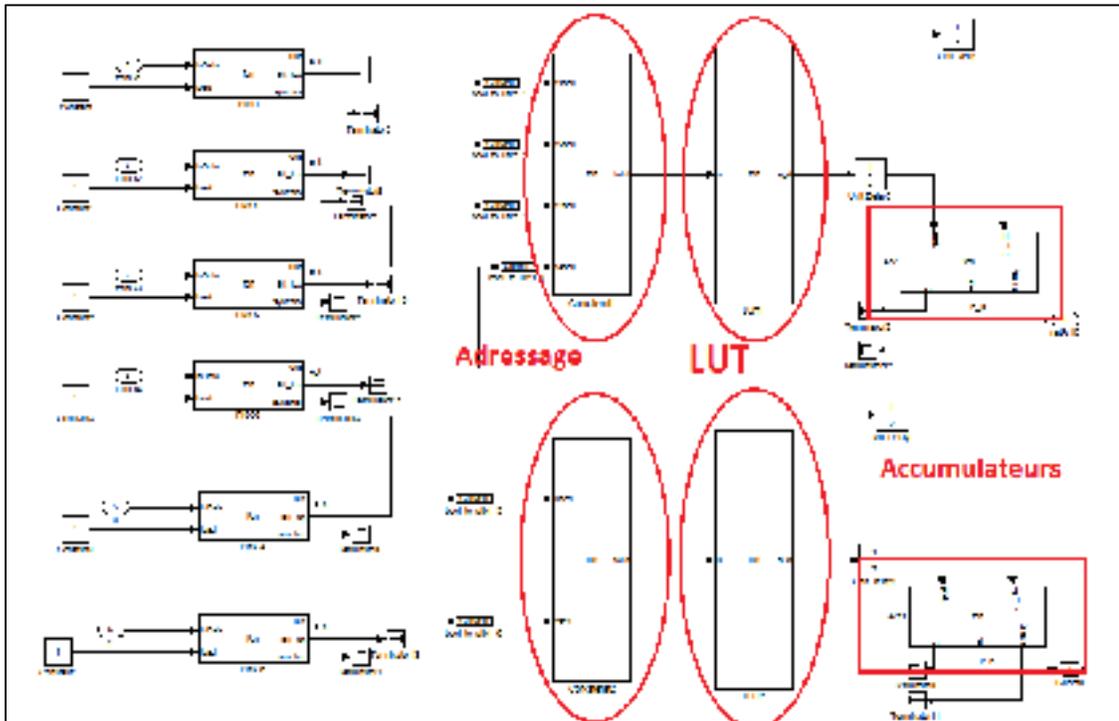


Figure 6.14 Détails de l'AD sur une ligne

On se propose de regarder l'erreur d'estimation, définie par la formule (2.79) pour les trois implémentations (43,35), (48,40) et (58,50). La Figure 6.15 présente l'allure des résultats.

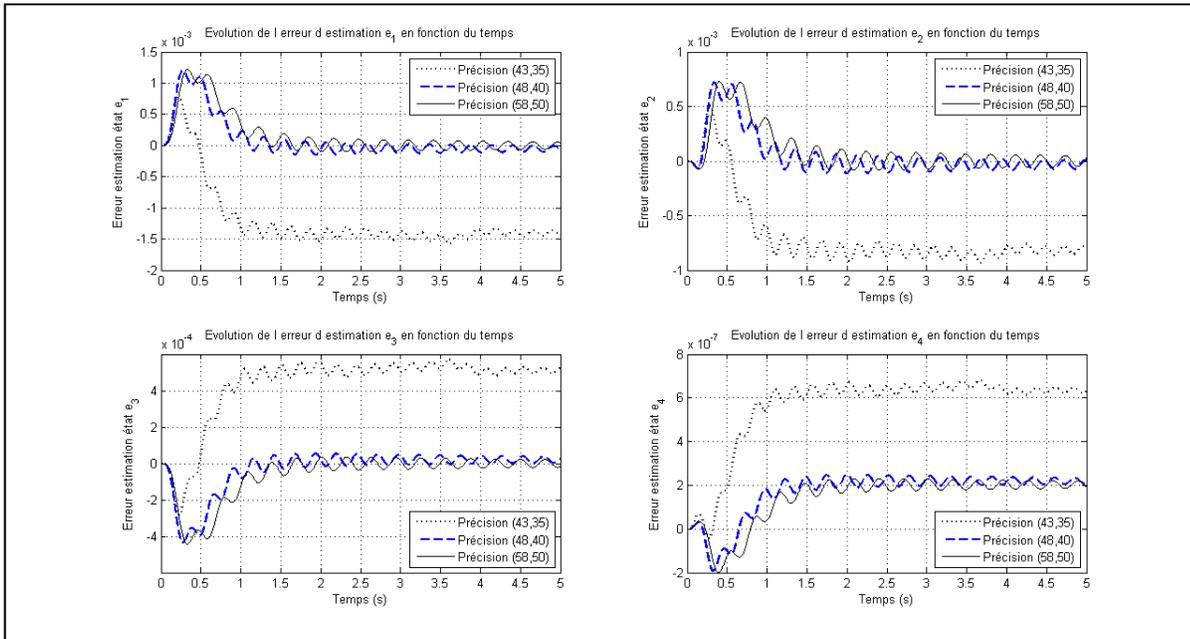


Figure 6.15 Erreur d'estimation de l'observateur à AD

On constate que pour une plus grande précision des calculs (implémentation (58,50)) on observe une meilleure convergence de l'estimation. En effet pour l'observateur (43,35), il y a une erreur statique importante tandis que pour le (58,50) l'erreur statique semble inférieure. On peut désormais éliminer l'observateur (43,35) car l'estimation n'est pas assez précise. La Figure 6.16 propose d'observer l'évolution de l'erreur de l'estimation pour la précision de (58,50) pour un temps de simulation plus long. On constate que l'erreur tend vers 0 et vérifie la plus grande qualité d'estimation de cet observateur.

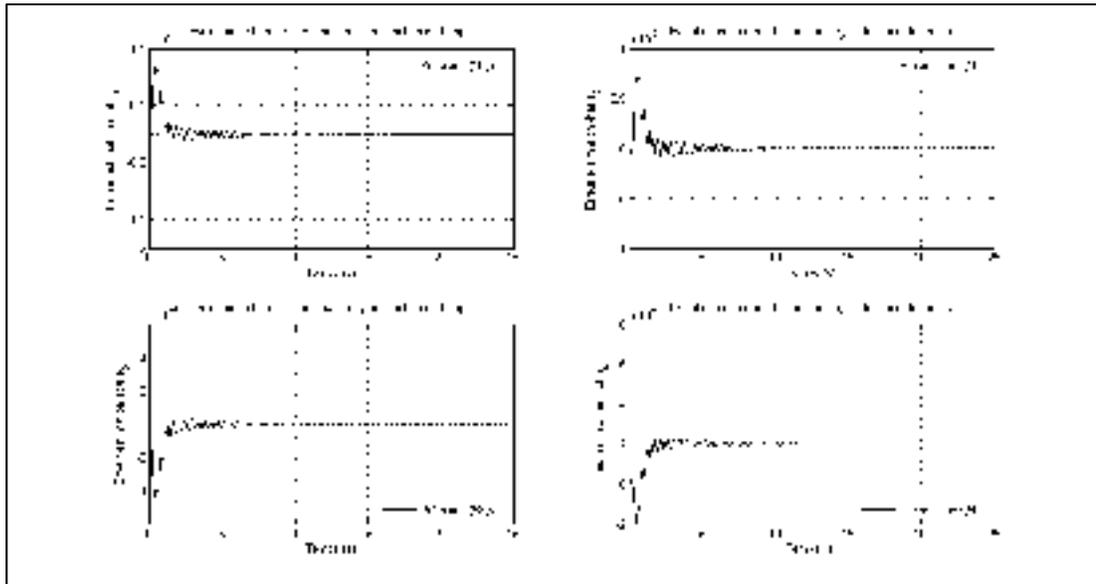


Figure 6.16 Erreur d'estimation pour l'observateur (58,50)

Nous avons donc pu observer que l'observateur implémenté selon le principe d'AD converge vers la valeur des états. Toutefois, la précision limitée et inhérente liée à la nature de codage (point fixe) implique des erreurs sur les états estimés, que cela se manifeste par une erreur statique et/ou une ondulation autour de la valeur estimée. Nous avons par ailleurs pu noter, ce qu'il était possible de deviner, à savoir que l'estimation est d'autant plus bonne que la précision des calculs est importante.

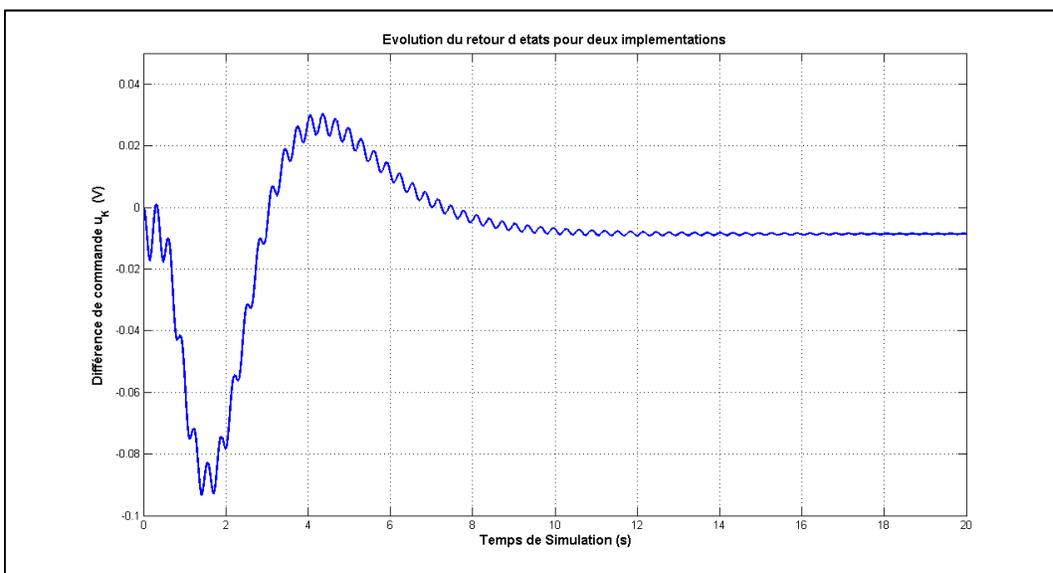
Il reste maintenant à regarder l'implémentation de l'observateur à AD au sein du FPGA. On se propose de regarder l'utilisation des ressources nécessaire dans chacun des trois cas simulés : (43,35), (48,40), (58,50). Le Tableau 6.5 présente les ressources utilisées en fonction de l'implémentation. On observe que les solutions viables pour une implémentation sont (48,40) et (43,35). Ainsi, comparativement à l'utilisation requise dans le cadre d'un observateur utilisant des multiplications, nous avons pu économiser des ressources et rendre ce dernier implémentable au sein du FPGA. On montre aussi qu'il est possible grâce à l'AD d'utiliser des variables plus grandes afin d'assurer une meilleure précision des calculs. Il reste maintenant à regarder le système en boucle fermée. Pour cela, on place le retour d'état sur les estimations des états.

Tableau 6.5 Utilisation des ressources du FPGA pour différentes implémentations

	(78,70)		(58,50)		(48,40)		(43,35)	
	Quantité	%	Quantité	%	Quantité	%	Quantité	%
Bascules	3838	32	2943	24	2461	20	2239	19
LUT	19729	167	15043	127	11393	96	10911	92
Tranches	9869	167	7523	127	5783	98	5607	95

6.2.4.2 Tests en boucle fermée

Il convient à présent de regarder le comportement en boucle fermée du système utilisant l'observateur construit par AD. D'abord, on se propose de comparer la commande envoyée par l'ensemble observateur (58,50) et retour d'état à celle fournie par un observateur discret tel que mis en place à la Figure 6.6 (précision de type *double*) avec retour d'état. La Figure 6.17 propose l'évolution de la différence de la commande u_k .

Figure 6.17 Evolution de la différence de la commande par retour état u_k

La Figure 6.17 présente l'écart entre la commande du retour d'état « théorique » et celui obtenu à partir de l'observateur à AD. On distingue une erreur n'excédant pas 80 mV en valeur absolue. On décide alors de fermer la boucle afin de voir si la différence de commande a un impact majeur. La Figure 6.18 présente la réponse du système en boucle fermée avec un observateur de type (58,50).

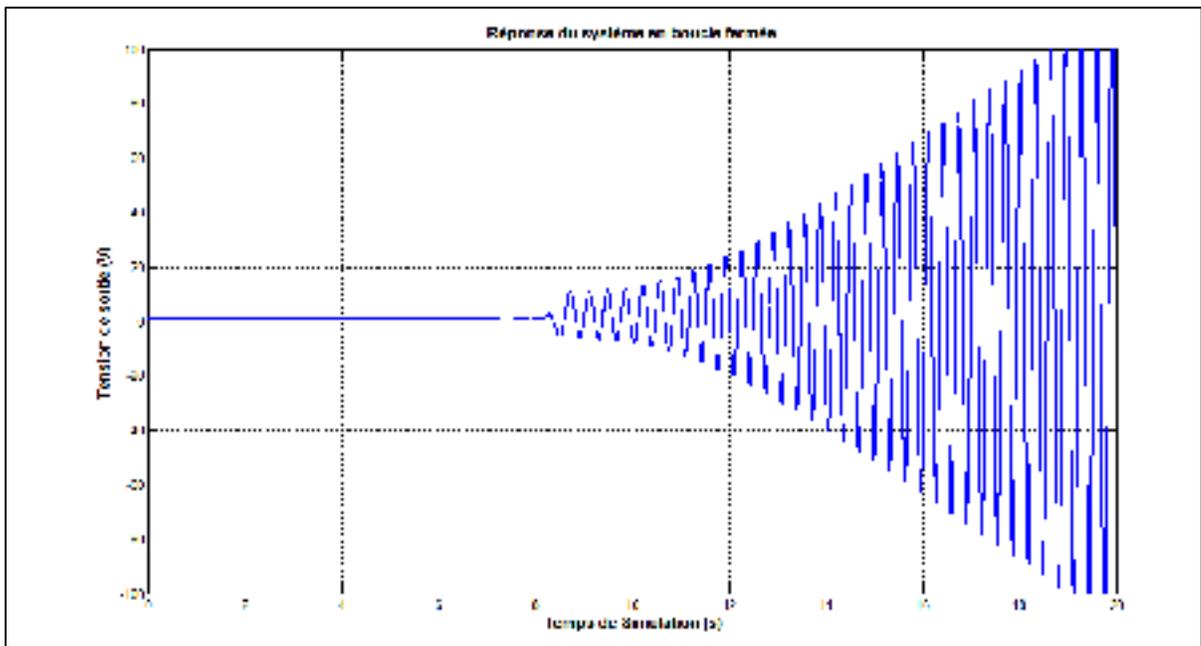


Figure 6.18 Réponse du système pour retour d'état et observateur de type (58,50)

On peut observer que le système en boucle fermée diverge. La raison à cela est la nature imparfaite des états estimés. En effet, en appliquant un retour d'état sur des états mal-estimés induit une commande mal-appropriée. Nous avons vu que les états x_1 , x_2 et x_3 tendent vers 0 en régime établi¹⁴. Alors les efforts de commande générés par ces états tendent vers 0. Dans le cas où les états sont mal estimés, et ne sont pas nuls en régime établi, l'effort de commande prend en considération l'action de ces états. Il en résulte une dynamique modifiée qui implique la divergence du système. Si la Figure 6.17 présentait une faible erreur entre la

¹⁴ Puisqu'ils représentent les dérivées successives de la position qui, en régime établi, est constante.

commande engendrée par l'observateur théorique et à AD, celle-ci est suffisante à déstabiliser le système.

6.3 Résumé

Dans cette partie, nous avons déroulé notre méthode de conception. D'abord nous nous sommes focalisés sur la fonctionnalité du système. Afin de vérifier nos calculs théoriques nous avons utilisé un système « parfait », purement continu. Nous avons ensuite affiné notre analyse en utilisant un organe de commande discret. Dans les deux cas, les résultats ont confirmé nos attentes. Nous sommes alors passés à la conception d'un module de commande et plus particulièrement d'un observateur synthétisable. En effet, la nature des valeurs définies dans un FPGA est à point fixe. Nous avons montré que notre système de commande à point fixe était fonctionnel en simulation. Cependant, lors de l'implémentation au sein du FPGA nous avons constaté que notre régulateur demandait trop de ressources. Il a donc fallu tenter une autre approche. L'AD propose une économie des ressources lorsque des sommes de produits sont nécessaires. Toutefois, cette méthode introduit une latence correspondant à la taille des variables dans le résultat. Il a donc fallu en tenir compte dans nos simulations. A la suite de simulations réalisées avec succès, nous avons décidé de passer à l'implémentation de l'observateur afin de vérifier que le gain en termes de ressources était effectif. L'AD nous a permis de disposer d'un observateur disposant de variables de travail plus importantes en nécessitant moins de ressources. Finalement, nous avons regardé le comportement du système en boucle fermée. La présence d'erreurs, aussi minimes soient-elles, lors de l'estimation a induit la divergence du système. Ces erreurs sont dues à l'accumulation d'erreurs lors du calcul par AD. L'algorithme fonctionnant par décalage successifs, il est aisé de comprendre que les erreurs s'accumulent. Notons, par ailleurs que le système utilisé pour nos tests dispose d'un conditionnement relativement pauvre. En effet, nous avons vu que la première représentation dans l'espace d'état n'était pas viable et qu'il a fallu reconditionner le système. Pour conclure, nous avons été en mesure d'implémenter un algorithme d'observateur par AD convergent mais pas assez à la vue des données du problème pour pouvoir passer à la réalisation.

CHAPITRE 7

VERS UNE CONCEPTION À BASE DE MODÈLES

Nous avons, aux CHAPITRE 5 et CHAPITRE 6, pu montrer le processus de conception pour la mise en place d'un PID et d'une commande dans l'espace d'état. Dans les deux cas, nous avons répondu à une même démarche. D'abord, nous avons simulé notre système de façon simplifiée en utilisant des éléments analogiques. Ensuite, nous avons discrétisé notre module de commande avant d'utiliser une implémentation répondant aux contraintes matérielles via une commande discrète à point fixe. Afin de disposer d'un modèle de simulation encore plus fidèle à la réalité, il est possible de passer à une modélisation plus complète.. Nous allons, ici pousser la conception à base de modèles (MBD) à son paroxysme. En effet, cette méthodologie propose de simuler un système directement implémentable en tenant compte des éléments matériels. Le but étant d'éviter toute surprise entre la simulation et le prototype réduisant ainsi le TTM.

7.1 Mise en œuvre du MBD

Nous allons mettre en œuvre le MBD pour nos deux types de commande, et en voir les avantages et intérêts. Ensuite, nous verrons que ce type de modélisation n'est possible qu'à l'aide d'un catalogue de fonctions prédéfinies permettant une flexibilité d'implémentation.

7.1.1 MBD pour la commande par PID

On se propose ici de mettre en place le MBD complet pour nos commandes. D'abord, nous allons voir comment mettre en place, dans le cas du PID, ce type de modélisation. Dans le CHAPITRE 5 nous avons utilisé uniquement des modèles simplifiés puisqu'on ne trouvait pas de trace des convertisseurs (CAN et CNA) et des circuits d'adaptation (CA1 et CA2). Le modèle proposé est présenté à la Figure 7.1 . Le bloc PID à AD tient dorénavant compte de

tous les éléments du système ainsi les gains implémentés correspondent aux expressions (5.15)-(5.17).

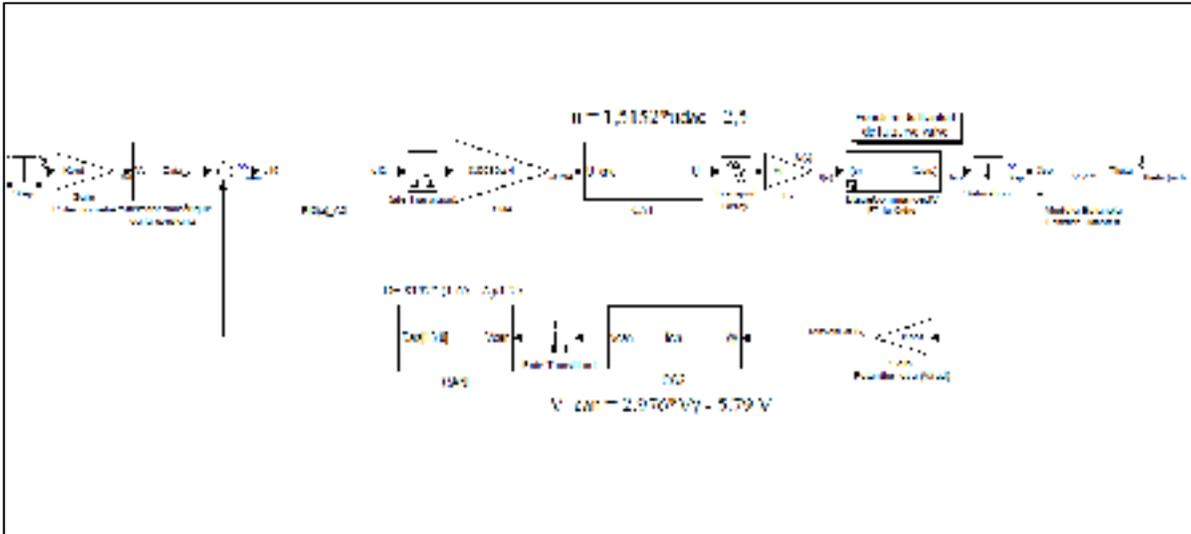


Figure 7.1 Schéma Simulink du système complet à commande PID

Les résultats de simulations sont exactement les mêmes que ceux présentés par la Figure 5.17 et le Tableau 5.3 Comparaison des grandeurs caractéristiques. En effet, la seule différence entre le modèle simulé à la partie (5.4) et celui de la Figure 7.1 repose sur la prise en compte des éléments CA1 et CA2 qui agissent sur la valeur du gain à implémenter au sein du FPGA comme indiqué par la formule (4.32). Le phénomène de modélisation complète et totale peut apparaître coûteux en termes de temps de conception (ajout de tous les éléments du système), mais le contrepoint est un gain de fiabilité dans les résultats de simulation ainsi que la réduction du temps de passage de la conception à la mise en œuvre pratique. Ainsi, sur le modèle de la Figure 7.1, le bloc PID, qui représente l'organe à implanter au sein de notre régulateur, peut directement être synthétisé à partir du modèle de simulation. Cela est rendu possible par la prise en compte de tous les éléments de l'environnement (dans notre cas les CAN et CNA et les circuits d'instrumentation CA1 et CA2).

7.1.2 MBD pour la commande dans l'espace d'état

À présent, nous souhaitons effectuer une modélisation complète pour le système dans l'espace d'état. Nous avons vu au CHAPITRE 6 comment implémenter un observateur d'état permettant d'effectuer un retour à l'aide des estimations de ceux-ci. Dans chacun des cas de figure nous avons travaillé avec deux signaux en entrée de l'observateur : la tension de commande et la tension de sortie. Ces tensions, lors du traitement numérique, ne sont pas directement accessibles. En effet, la tension de sortie est conditionnée par un circuit avant d'être convertie à l'aide d'un CAN. La tension de commande est quant à elle fournie par le FPGA sur 12 bits non signé qui, via le CNA, renvoie une tension. Le schéma de cette structure a été présenté à la Figure 4.1.

Si l'on souhaite disposer du même type d'observateur (et plus généralement de commande par retour d'état) dans un cas concret il nous fait donc obtenir les grandeurs physiques mise en jeu. La Figure 7.2 présente le nouveau système.

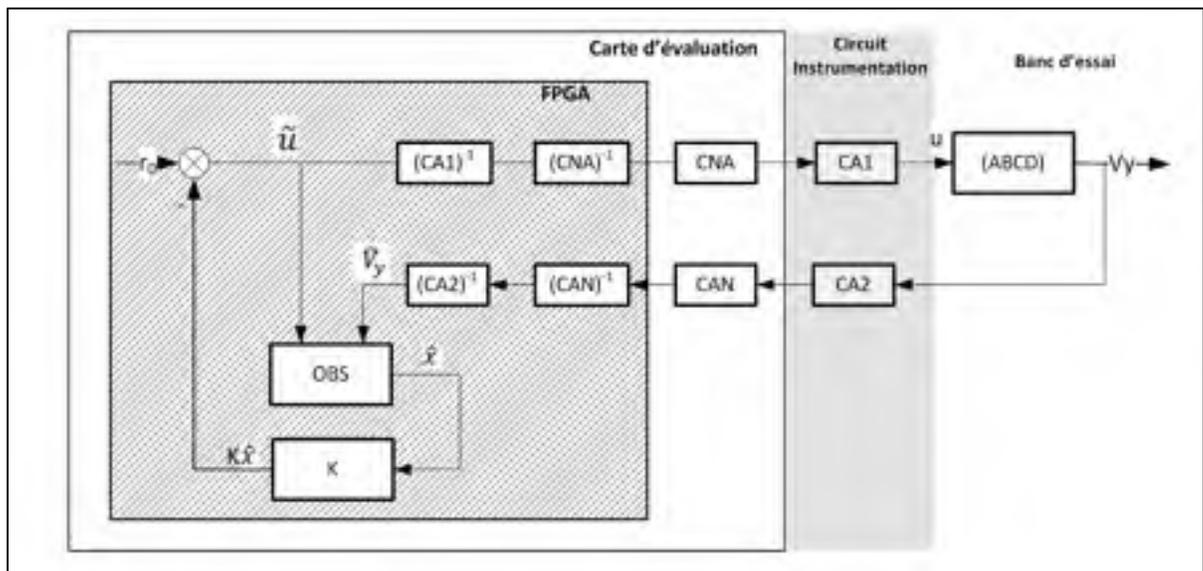


Figure 7.2 Schéma d'implémentation de la commande dans l'espace d'état

On retrouve dans la Figure 7.2 les mêmes éléments que ceux présents dans la Figure 4.1. Toutefois le système apparaît augmenté d'un certain nombre de blocs. D'abord, comme nous

pouvions nous y attendre, on constate la présence de l'observateur et du retour d'état K permettant de fournir la commande par retour d'état $K\tilde{x}$. Les entrées de l'observateur sont : \tilde{u} et \tilde{V}_y . Ces notations indiquent que les grandeurs sont représentées à l'aide de valeurs à point fixe. Celles-ci sont les images des valeurs de tension de commande u et de sortie V_y analogiques. Afin d'obtenir ces images, il est nécessaire de mettre en place les blocs $(CNA)^{-1}$, $(CAN)^{-1}$, $(CA1)^{-1}$ et $(CA2)^{-1}$. Ces blocs permettent d'inverser les traitements reçus par les signaux u et V_y . La description de ces blocs est donnée à la suite.

D'après l'expression (4.11) on peut donner la représentation en point fixe du CAN, \tilde{v}_{CAN} correspondant au circuit $(CAN)^{-1}$:

$$\tilde{v}_{CAN} = 1,65 - \frac{1,25 \times data[13:0]}{8192} \quad (7.1)$$

Le circuit $(CA2)^{-1}$ s'exprime à partir des équations (4.12) et l'image de la tension de sortie représentée codée en point fixe, \tilde{V}_y se note :

$$\tilde{V}_y = \frac{\tilde{v}_{CAN} + 5,79}{2,976} \quad (7.2)$$

Le circuit $(CA1)^{-1}$ donne la représentation en point fixe de la tension en sortie du CNA conformément à l'expression (4.20) :

$$\tilde{u}_{CNA} = \frac{\tilde{u} + 2,5}{1,5152} \quad (7.3)$$

Enfin le bloc $(CNA)^{-1}$ permet d'obtenir la valeur $Data_{CNA}$ à transmettre au CNA (expression (4.21)) :

$$Data_{CNA} = \frac{\tilde{u}_{CNA}}{8,0586 \cdot 10^{-4}} \quad (7.4)$$

Nous effectuons maintenant la simulation du système complet. Pour commencer, nous allons regarder la convergence de l'observateur. Nous distinguons deux cas :

- la commande envoyée par le FPGA au CNA, u_{CNA} , ne tient pas compte de la nature à point fixe. On dispose alors d'un CNA « parfait » ;
- la commande envoyée par le FPGA au CNA, \tilde{u}_{CNA} , est une valeur non signée sur 12 bits.

La Figure 7.3 présente l'erreur d'estimation de l'observateur pour les deux cas de figure. Notons que l'observateur utilisé pour cette simulation est celui mis en œuvre dans la partie (6.2.3).

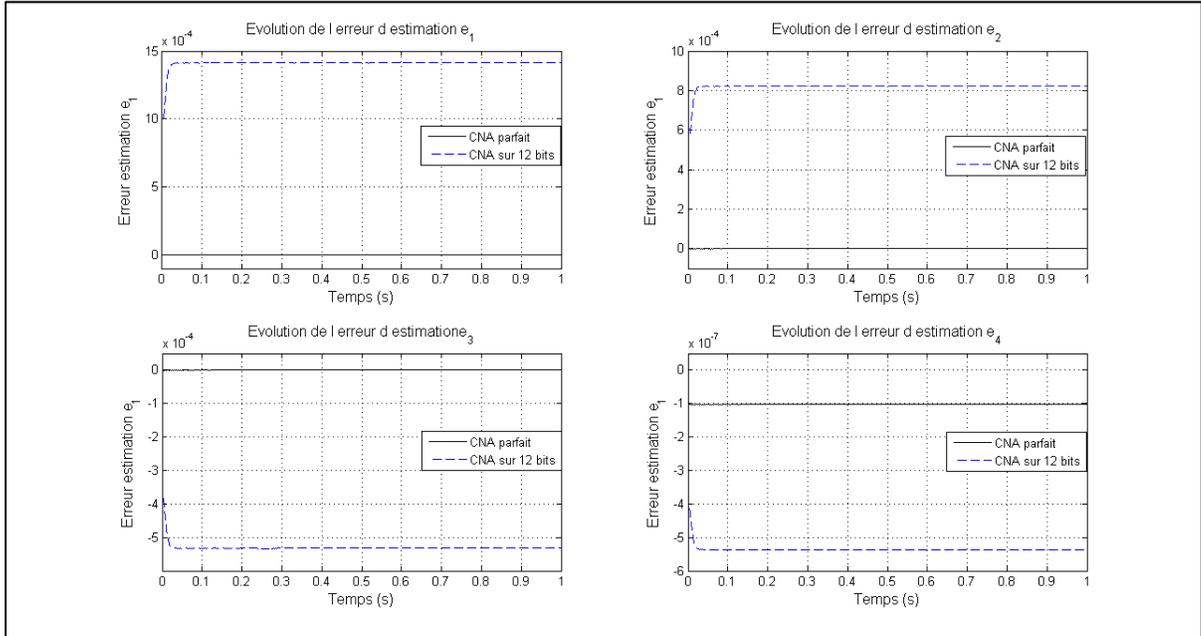


Figure 7.3 Erreur d'estimation de l'observateur en fonction du modèle du CNA

On constate une erreur d'estimation statique pour un CNA réel. En effet, le CNA à notre disposition convertit une valeur non signée sur 12 bits en une tension comprise entre 0 et 3,3V. Ainsi, la résolution de celui-ci est d'environ 0,8 mV. Nous constatons d'ailleurs sur la Figure 7.3 que l'ordre de grandeur de l'erreur d'estimation est de 10^{-4} . Il nous reste à vérifier le comportement du système en boucle fermée.

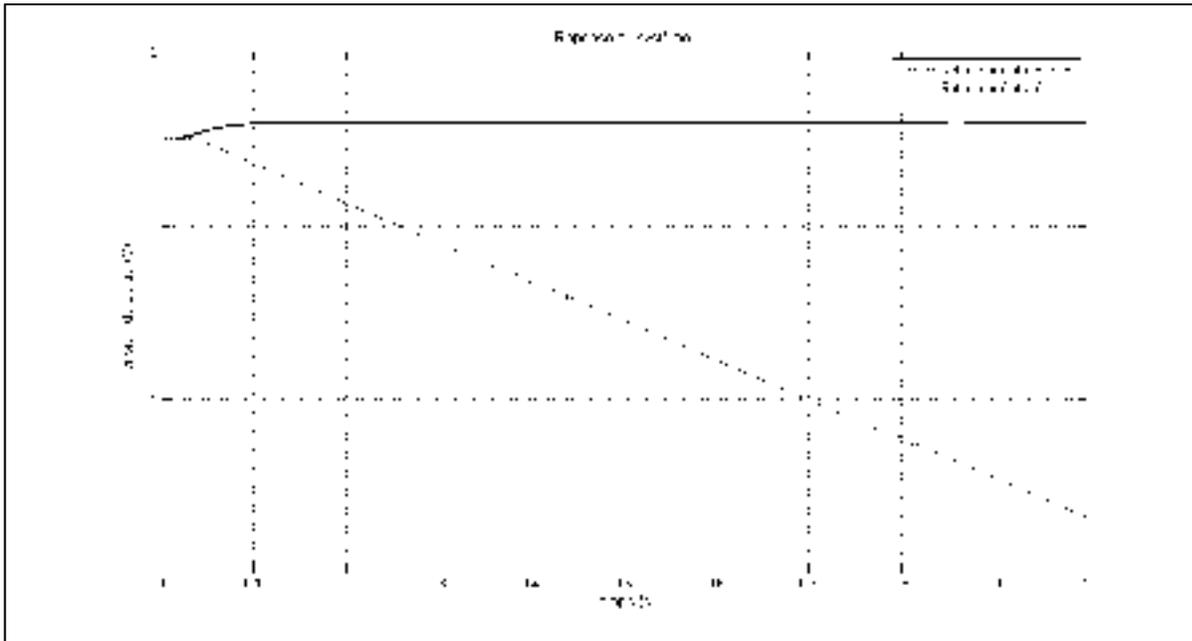


Figure 7.4 Réponse du système pour un retour d'état sur états réels et états estimés avec CNA réel

La Figure 7.4 montre le comportement du système en boucle fermée pour le système muni d'un CNA réel (défini sur 12 bits). On constate que le système diverge lorsque l'on applique un retour d'état à l'aide des états estimés. Le système est stable dans le cas où le retour d'état se fait au moyen des états réels. Ainsi les résultats présentés par les Figure 7.3 et Figure 7.4 permettent de conclure à la non viabilité de cette commande. En effet, le CNA ne permet pas d'obtenir une précision suffisante lors de l'estimation des états et induit une commande non appropriée.

Au final, le MBD s'il est efficace pour disposer d'un modèle simulable et implémentable, peut être relativement long à mettre en place. En effet, la méthode (Nicolescu et Mosterman, 2009) repose sur la description la plus complète du système à l'étude (Rongqing et al., 2011). C'est pourquoi, comme l'indique Perry (2009), il est nécessaire de disposer d'éléments déjà implémentés à haut niveau d'abstraction dont le fonctionnement à bas niveau a été vérifié. Ainsi, il est naturel, à la suite de notre démarche, de se porter sur la mise en place d'une bibliothèque de composants.

7.2 Mise en place de la bibliothèque de composants

Très tôt, il y a eu la volonté de disposer de composant réutilisable afin de réduire le TTM (Charaabi, Monmasson et Slama-Belkhodja, 2002). Toutefois, les bibliothèques de composants sont le plus souvent très générales -blocs élémentaires- (Matlab/Simulink, 2007; Xilinx, 2012b) ou dédiés aux domaines historiques des FPGA (Maxfield, 2004).

Ainsi la mise en place d'éléments de commande au sein d'une bibliothèque de composants répond à plusieurs attentes :

- disposer d'éléments de commande prêts à être synthétisé et implémentés ;
- avoir des modules de commande permettant de concevoir des systèmes en utilisant une démarche de conception à base de modèles (MBD) ;
- permettre un accès plus facile à l'utilisation des FPGA dans les systèmes de commande embarqués ;
- combler le manque de ce type de bibliothèque.

La bibliothèque est présentée en ANNEXE V. Nous nous proposons, ici, de regarder l'architecture de cette dernière et les grandes lignes de son fonctionnement.

De façon générale, les éléments peuvent se trouver seuls, ou intégrés au sein de module préconçus. Cela permet à l'utilisateur final d'avoir le choix dans la solution à adopter. Typiquement s'il désire rapidement passer au prototypage à l'aide d'un système connu, il peut utiliser un modèle complet prêt à la synthèse. En revanche, s'il est en phase de conception de son système, il peut souhaiter disposer d'un élément en particulier facilement adaptable à ses besoins.

La bibliothèque mise en place se structure hiérarchiquement. D'abord, nous trouvons deux rubriques :

- 1) Les modules de gestion des interfaces
- 2) Les organes de commande

7.2.1 Modules de gestion des interfaces

Les modules de gestion des interfaces sont ceux présents dans la Figure 5.3. La rubrique se scinde en deux entités :

- 1) Les éléments disponibles individuellement,
- 2) Des assemblages directement réutilisables.

On dispose ainsi d'une liberté de choix lors de la conception des systèmes en pouvant n'utiliser qu'un seul des éléments ou un montage prédéfini.

Les éléments présents dans cette rubrique sont les suivants :

- un module de gestion du CAN LTC1407A-1 de Linear Tech ;
- un module de gestion du CNA LTC2624 Quad Dac de Linear Tech
- un gestionnaire des priorités CAN/CNA.

7.2.2 Les organes de commande

Nous avons pu développer au cours de notre travail deux méthodes de commande qui sont elles-mêmes décomposées en sous parties :

- 1) PID ;
 - a) Les blocs de base : actions P, I et D ;
 - b) Les structures PI, PD et PID.
- 2) Commande moderne.
 - a) Les blocs de base : observateur, retour d'état ;
 - b) Montages complets observateur et retour d'état.

L'utilisateur dispose alors d'un ensemble complet de blocs fonctionnels prêts à l'usage.

Notons que l'on propose au sein de la bibliothèque plusieurs types d'implémentation : multiplication avec ou sans AD, commande moderne pour des systèmes de dimension trois ou quatre.

Par ailleurs afin d'assurer la flexibilité de notre bibliothèque, un script associé à chacune des méthodes de commande est proposé. En bénéficiant de la paramétrisation des EMF (partie 5.2.1), on peut alors proposer à l'utilisateur de modifier les éléments des blocs de commande à l'aide d'un script. Ainsi, il n'aura pas à ouvrir le bloc sélectionné afin d'apporter les modifications pour son système. Cela présente également l'avantage de ne requérir que peu de connaissances en programmation chez l'utilisateur. Typiquement, le script proposé pour la commande moderne permet à l'utilisateur de définir son système soit à l'aide d'une fonction de transfert ou d'une définition du système (A, B, C, D) directement. Ensuite l'utilisateur choisit la position de ses pôles et le script génère automatiquement les matrices H et K . Si jamais il a choisi d'utiliser l'AD pour son système, le script permet de remplir les LUT automatiquement sinon les gains pour les multiplications sont directement implantés.

7.3 Résumé

Au final, nous avons développé une démarche de conception par MBD pour des modules de commande dédiées aux FPGA. Dans le premier cas, partie (7.1.1), nous avons mis en place cette méthode de conception pour le plus courant des régulateurs, le PID. Nous avons pu avec succès modéliser le système entier en tenant compte de l'ensemble des éléments présents dans le montage contrairement à ce que nous avons pu réaliser au CHAPITRE 5. Cela nous a permis d'aboutir à un schéma de simulation muni d'un régulateur directement synthétisable et implémentable.

Dans un second temps, nous avons cherché à appliquer le MBD à un organe de commande plus complexe. En effet, une commande dans l'espace d'état requiert un nombre important de calculs (matriciels). Par ailleurs, la démarche MBD nous a montré les limites de notre système. Une simple modélisation d'un élément du système (le CNA) a permis de voir que la mise en œuvre pratique s'avérait difficile à opérer. En effet, le principe du MBD repose sur une modélisation de haut niveau mais permet d'affiner le modèle au fur et à mesure de la

conception (Rongqing et al., 2011) afin de réduire les incertitudes du passage à la réalisation pratique. De ce point de vue, cette méthode s'est avérée efficace puisqu'elle nous a permis de constater que la résolution du CNA utilisé n'était pas suffisante pour contrôler convenablement notre système à l'aide de la commande moderne.

Nous avons ensuite mis en exergue la nécessité de disposer d'un ensemble de composants existants afin de profiter au mieux de la démarche de conception à base de modèles. Notre bibliothèque permet donc l'utilisation des éléments développés au cours de notre travail. Sa structure hiérarchisée se prête à une plus grande flexibilité de mise en place. Par ailleurs, la flexibilité d'utilisation est accrue par une prise en compte automatisée des contraintes matérielles (tailles des variables, méthodes de multiplication).

CHAPITRE 8

DISCUSSION

Dans ce chapitre, nous proposons un regard critique sur notre travail. Nous avons déjà pu donner des résultats munis d'une première analyse, le plus souvent exclusivement factuelle. Il est nécessaire d'aller plus loin et de dépasser le premier degré d'analyse en inscrivant nos résultats dans une étude comparative de la littérature.

8.1 Commande PID par FPGA

A mesure de l'augmentation du niveau d'abstraction des logiciels d'aide à la conception électronique, s'est accrue l'utilisation de circuits numériques reconfigurables au sein des systèmes embarqués (Paiz et Pormann, 2007). Monmasson (2007) fait du FPGA un candidat de choix pour la conception de modules de commande embarqués. Très tôt, les premiers régulateurs ont été implémentés à l'aide de FPGA profitant de logiciel de conception graphique (Charaabi, Monmasson et Slama-Belkhodja, 2002; Ruei-Xi, Liang-Gee et Lilin, 2000) avant de se concentrer sur la réalisation pratique en évoquant des méthodes efficaces d'implémentation prenant en compte les contraintes matérielles (Chan, Moallem et Wang, 2004; Sanchez-Solano et al., 2007; Zhengwei, Carletta et Veillette, 2005), et toujours d'actualité (Baghel et Shaik, 2011).

Ainsi dans la continuité des travaux cités, nous avons cherché à développer un module de commande PID par FPGA au moyen d'une méthode d'implémentation permettant une économie des ressources disponibles à l'aide de l'AD. Dans un premier temps, nous avons obtenu, en simulation, un comportement équivalent à une implémentation par un autre moyen comme l'a présenté la Figure 5.16. Toutefois, le Tableau 5.3 nous indiquait une certaine erreur entre la réponse présentée par le système analogique théorique et celui disposant d'un régulateur muni d'une gestion des nombres à point fixe.

Ensuite, nous avons cherché à savoir si la mise en œuvre d'un tel correcteur se révélait pertinente. Nous avons alors comparé nos résultats expérimentaux avec ceux de Gagnon (2011) qui utilisait un module de commande réalisé à l'aide de xPC Target. La Figure 5.18 présente les allures comparées des réponses du système pour les deux types de régulateur. Nous avons pu obtenir des résultats quasi-semblables comme le rappelle le Tableau 5.7. La principale source d'erreur sur notre système à FPGA repose sur la précision de notre CNA et sur le traitement en point fixe des variables de calcul. En effet, Gagnon (2011) disposait d'un CNA de 16 bits sur la carte National Instrument NI-6229. Par ailleurs, le traitement des opérations à l'aide de xPC Target s'effectue en virgule flottante, réduisant encore les imprécisions.

Par ailleurs, cette première phase de travail a été décomposée en deux parties. Nous nous sommes d'abord concentrés sur la simulation pour juger de la viabilité avant de se pencher sur l'implémentation. Nous avons dû retravailler notre correcteur qui ne disposait pas des mêmes paramètres en fonction de la phase de travail. Cela est illustré par l'équation (X.XX) qui présente le gain statique d'implémentation qu'il est nécessaire de prendre en compte. Nous sommes ensuite passés à une nouvelle forme de modélisation : la conception à base de modèles (MBD). Cette démarche de conception permet de réduire le TTM en fusionnant les étapes de simulation et d'implémentation (Mosterman, 2007). Nous avons en outre mis en place des composants réutilisables au sein d'une bibliothèque. Dans l'optique d'une conception à base de modèles, le fait de disposer d'éléments réutilisables et flexibles est absolument nécessaire (Kirby et Kang, 2008).

Au final, en appliquant la démarche de conception à l'aide de modèles extrêmement précis et en utilisant une bibliothèque de composants, nous sommes en mesure de disposer d'une méthode de développement rapide et facile à implanter. On obtient un temps de conception et de mise en pratique équivalent à ce que l'on peut obtenir à l'aide de xPC Target. En outre, on peut alors profiter des avantages des FPGA comme le sont : la reconfigurabilité, la rapidité de calcul, l'utilisation au sein de systèmes embarqués.

8.2 Commande dans l'espace d'état par FPGA

Après avoir réalisé avec succès un module PID simulable, synthétisable et implémentable dans un FPGA, nous avons cherché à mettre en place une méthode de commande plus élaborée. La commande moderne repose sur du calcul matriciel. Dès lors, cela met en jeu un grand nombre d'opérations et plus particulièrement de multiplications. Nous avons toutefois pu montrer à l'aide d'un simple PID que l'AD permettait d'effectuer des multiplications en économisant les ressources.

Nous avons dans un premier temps modélisé notre système dans l'espace d'état en mettant en évidence un problème initial de conditionnement numérique (partie 4.3). Toutefois, Moore (1981) a proposé une méthode permettant d'équilibrer un système après avoir montré que les difficultés numériques pouvaient venir du système de coordonnées d'un système (A, B, C, D) . Le logiciel Matlab (Mathworks, 2007) permet le passage d'une fonction de transfert en une représentation d'état. Sous réserve d'utiliser la fonction `ss`, la représentation d'état obtenue est équilibrée. Nous avons alors pu commencer la conception de notre système de commande. Notre système, SISO, n'est pas doté d'états directement accessibles. Il a donc fallu mettre en œuvre un observateur d'état (Bensoussan, 2008). Dans notre cas, le système est de dimension quatre. L'observateur d'état doit alors effectuer quatre sommes de six produits soit 24 multiplications. En un premier temps, nous avons montré que le système était réalisable puisque les résultats de simulation à l'aide d'une commande idéale indiquent un comportement attendu. Ensuite, nous avons cherché à disposer d'un module synthétisable capable d'être embarqué dans un FPGA. Pour cela, il a fallu passer à la mise en place d'une gestion à point fixe des valeurs traitées. Un observateur utilisant des multiplicateurs a d'abord été testé. Si celui-ci s'est révélé fonctionnel, nous avons pu constater que son implémentation n'était pas possible car elle demandait trop de ressources, comme l'indique la **Erreur ! Source du renvoi introuvable.** Nous avons alors adopté l'AD qui permet une économie des ressources. Nous avons alors pu constater que l'observateur implémenté avec cette méthode était d'autant plus précis que la taille des variables mises en jeu augmentait. Toutefois, les accumulations d'erreurs inhérente à l'algorithme ont introduit une erreur

statique rémanente qui malgré sa faiblesse a produit la divergence du système. Nous avons alors pu toucher du doigt le problème évoqué par Kelly (1997). Ce dernier évoque le problème de système se prêtant peu à la mise en œuvre pratique dû à une sensibilité (de quantification notamment) trop importante. Cependant, nous avons pu montrer que l'algorithme à AD se révélait particulièrement efficace pour l'implémentation sur le FPGA. À l'étude de la **Erreur ! Source du renvoi introuvable.** et du Tableau 6.5, on peut dire que l'AD nous a permis d'économiser 46% des ressources nécessaires pour l'utilisation de multiplication.

Enfin, nous avons cherché à aller plus loin dans la modélisation du système en adoptant une procédure de description complète du système. Par rapport aux simulations effectuées au CHAPITRE 6, nous avons tenu compte de l'implémentation concrète du système comme le résume la Figure 7.2. Nous nous sommes aperçu que la modélisation précise du CNA a provoqué la non-viabilité de notre système de commande. En effet, dans cette partie nous avons souhaité voir si l'observateur synthétisable (mais non implémentable pour causes de ressources) décrit dans la partie 6.2.3 permettait toujours d'avoir un comportement convergent dans un système décrit en détail. Les Figure 7.3 et Figure 7.4 nous ont permis de conclure que le CNA présent sur la carte d'évaluation n'était pas suffisamment précis si bien que l'estimation des états comportait une erreur statique. Celle-ci malgré sa faible grandeur (de l'ordre de 10^{-4}) suffit à déstabiliser le système lors de l'application d'un retour d'état sur les valeurs estimées.

En parallèle de la conception à base de modèles, nous avons développé une bibliothèque de composants réutilisables et flexibles permettant à un utilisateur de minimiser le temps de conception en se focalisant sur le système en général et non sur les détails d'implémentation. La flexibilité des composants de la bibliothèque est rendue possible à l'aide de la paramétrisation des EMF évoquée dans la partie 5.2.1. Ainsi un script permet, après que l'utilisateur ait fourni ses spécifications, de disposer d'un modèle directement simulable et synthétisable. L'utilisateur n'a alors pas à rentrer dans le détail des blocs mis en jeu, ce qui lui permet de se concentrer sur les performances de son système en ne perdant pas de temps

sur la gestion des types numériques, ou la méthode de calcul, par exemple qui sont des composantes situées à bas niveau. Notons toutefois, que pour le moment, le script génère uniquement des résultats pour un système complètement observable et commandable.

CONCLUSION

Dans ce travail, nous avons proposé d'utiliser les FPGA comme organe de commande au sein de systèmes embarqués. D'abord, nous avons rappelé ce qu'était un FPGA, leur place dans le spectre des composants numériques, les concepts qu'ils mettent en jeu et les moyens mis à notre disposition pour les utiliser. Cela nous a permis de voir que les FPGA présentent un important nombre de caractéristiques les rendant intéressants dans une optique de commande de systèmes embarqués. Ensuite, nous avons revu des éléments théoriques de commande. Nous avons pu mettre en parallèle les comportements analogiques et discrets et donner les liens de passage d'une représentation à une autre. Puis nous avons évoqué la façon dont sont traités les nombres dans les FPGA avec une définition des valeurs à point fixe. Nous avons par la suite proposé la description du système utilisé pour mener à bien nos expériences. La description du modèle s'est réalisée sous deux formes : par une fonction de transfert et par une représentation d'état. Nos modèles étant à disposition, nous avons alors pu commencer la mise en œuvre de systèmes de commande à base de FPGA. Profitant de la boîte à outils de Matlab/Simulink *HDL Coder*, il nous a été possible de concevoir des modules synthétisables à l'aide de l'interface graphique de Simulink.

Dans un premier temps, nous nous sommes concentrés sur la mise en place d'un PID. Ce dernier représente le régulateur le plus présent dans l'industrie. Il constituait donc un passage obligé. Nous avons pu montrer que l'utilisation d'un FPGA était envisageable et pouvait même se révéler efficace pour la mise en place d'un tel régulateur. Nous avons porté nos efforts sur la méthode de calcul en utilisant un algorithme évoqué dans la littérature comme permettant de limiter l'utilisation des ressources. En un deuxième temps, nous avons proposé une méthode de commande plus évoluée. La commande moderne qui demande un nombre important de multiplications matricielles nous est apparue comme un défi intéressant. Toutefois, la nécessité d'utiliser un algorithme peu coûteux en termes de ressources a introduit des erreurs de calculs rédhibitoires. Nous avons alors pu appréhender la forte dépendance de ce mode de commande aux variations du système et au conditionnement. Enfin dans un dernier temps, nous avons porté nos efforts sur une méthode de conception à

base de modèles. Cette dernière se révèle particulièrement efficace dans le cas de la réalisation de systèmes complexes. En effet, on peut alors disposer de modèle à la fois simulables et synthétisables (et implémentables). Cette méthodologie n'est cependant qu'efficace qu'en présence d'une bibliothèque de composants réutilisables et flexibles permettant à l'utilisateur final de rester à un haut niveau d'abstraction. C'est la raison pour laquelle nous avons réuni au sein d'une bibliothèque les éléments développés au cours de notre travail. En utilisant, une automatisation des valeurs mises en jeu liée à la paramétrisation des EMF nous disposons d'éléments facilement adaptables à tous les systèmes pouvant bénéficier de commandes modernes et régulateurs PID.

Pour finir nous nous proposons d'établir une liste non exhaustive des éléments pouvant faire l'objet de travaux futurs :

- Amélioration de l'électronique :
 - mettre en œuvre des convertisseurs CAN et CNA de plus grande résolution ;
 - utiliser des circuits d'instrumentation (CA1 et CA2) plus performants (tolérance des résistances à $\pm 1\%$ et meilleurs amplificateurs opérationnelles) ;
 - utilisation d'un FPGA de nouvelle génération permettant un plus grand nombre de multiplicateurs.
- Amélioration de la gestion numérique :
 - utiliser des nombres en virgules flottantes, ce qui implique d'agir sur le logiciel de synthèse de Xilinx ;
 - appliquer la technique de transformation creuse pour la commande dans l'espace d'état ce qui permet de réduire le nombre d'opérations et contribue à une meilleure sensibilité (Kelly, 1997).
- Amélioration de la bibliothèque :
 - ajout de nouveaux modes de commande notamment non linéaire en profitant de la rapidité de calcul des FPGA pour faire de la linéarisation en temps réel, par exemple ;
 - prise en compte dans l'espace d'état des systèmes non complètement observables et commandable ;
 - mise en place d'observateurs d'ordre réduit.

ANNEXE I

UTILISATION DES OUTILS LOGICIELS

Nous nous proposons ici de décrire le fonctionnement de la suite logicielle pour notre travail et plus particulièrement les programmes proposés par la société Xilinx et réunis en même environnement de travail nommé ISE pour *Integrated Synthesis Environment*, où environnement de synthèse intégré. Nous allons présenter l'interface utilisateur et les bases nécessaires et suffisantes à l'utilisation de cette suite logicielle. Bien entendu, ce qui suit n'est absolument exhaustif et pour des éléments plus approfondis, le concepteur/lecteur pourra se référer aux guides d'utilisateurs des programmes présents dans la bibliographie.

Lancer le logiciel *Xilinx ISE*. L'icône porte le nom « Project Navigator ». Dans le menu « file » choisir « New Project ». La fenêtre suivante apparaît :

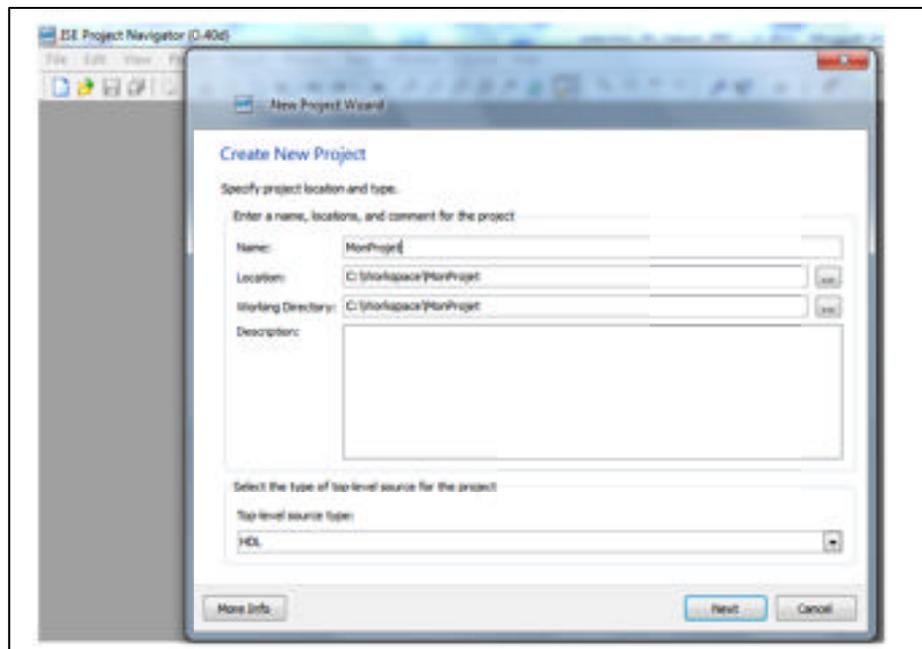


Figure-A I-I Création d'un nouveau projet dans ISE

Inscrivez le nom de votre projet. Ici, nous avons choisi d'appeler notre projet « MonProjet ». Notez que votre projet doit se trouver dans le répertoire « Workspace » lui-même situé à la racine de votre disque. En d'autres termes les chemins d'accès à vos projets doivent être de la forme : « C:\Workspace\MonProjet ». Cliquez sur « Next ». Une deuxième fenêtre apparaît. Celle-ci vous permet de régler les propriétés de votre projet.

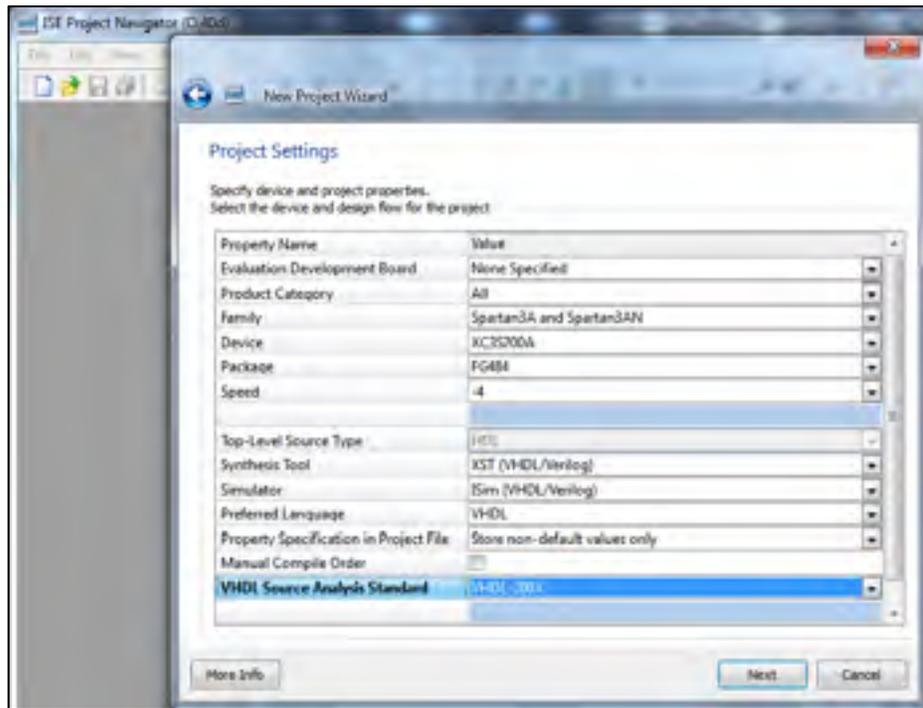


Figure-A I-II Propriétés d'un nouveau projet

Si vous utilisez le Spartan 3-A Starter Kit, veuillez choisir les propriétés suivantes :

- Family : *Spartan3A and Spartan3AN*
- Device : *XC3S700A*
- Package : *FG484*
- Speed : *-4*
- Synthesis Tool : *XST*
- VHDL Source Analysis Standard : *VHDL-200X*

Cliquez sur « Next ». Une troisième fenêtre apparaît. Celle-ci récapitule l'ensemble des options de votre projet en rappelant le répertoire choisi pour le projet. Vous venez de finir la création de votre nouveau projet. Il ne reste qu'à y insérer vos fichiers sources VHDL généré à partir de HDL Coder. Dans la fenêtre principale d'ISE, apparaît un volet « design » situé sur la gauche de celle-ci. Ce volet vous permet de gérer facilement vos fichiers sources et de procéder aux étapes de la synthèse, du PAR ou de la génération du fichier de configuration.

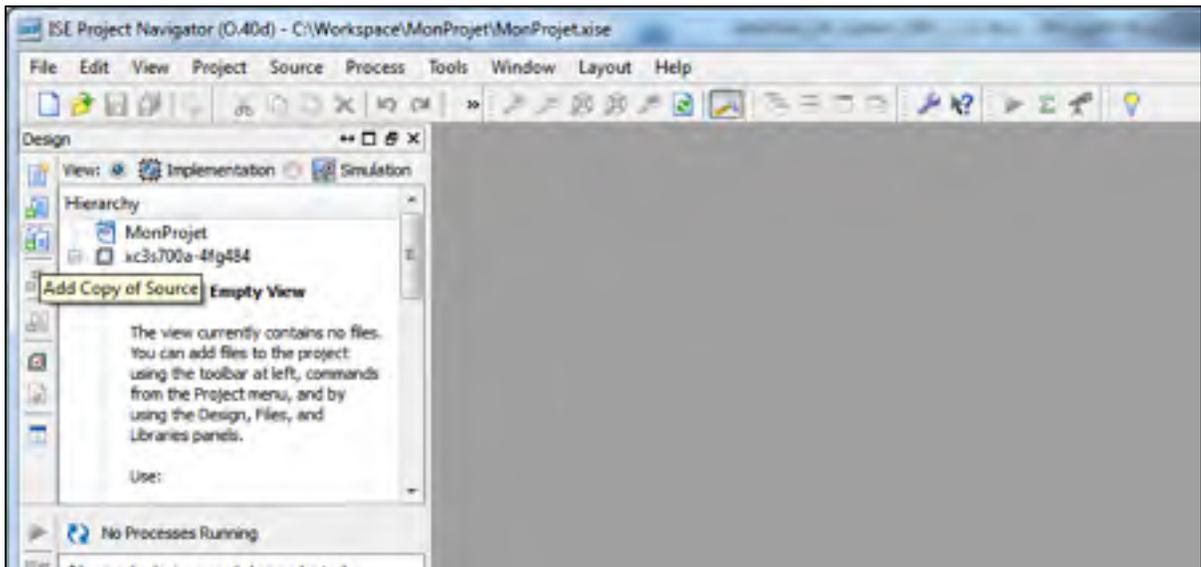


Figure-A I-III Insertion des fichiers sources

Afin d'ajouter les codes sources à votre projet, sélectionner « *Add Copy of Source* » dans le volet « *Design* ». Une nouvelle fenêtre s'ouvre. Parcourez les répertoires afin de vous situez dans le répertoire où vous avez généré les fichiers sources avec HDL Coder. Vous pouvez sélectionner l'ensemble de vos sources en même temps. Pour sélectionner l'ensemble des fichiers sources faites « *Ctrl+A* ». Cliquez ensuite sur ouvrir.

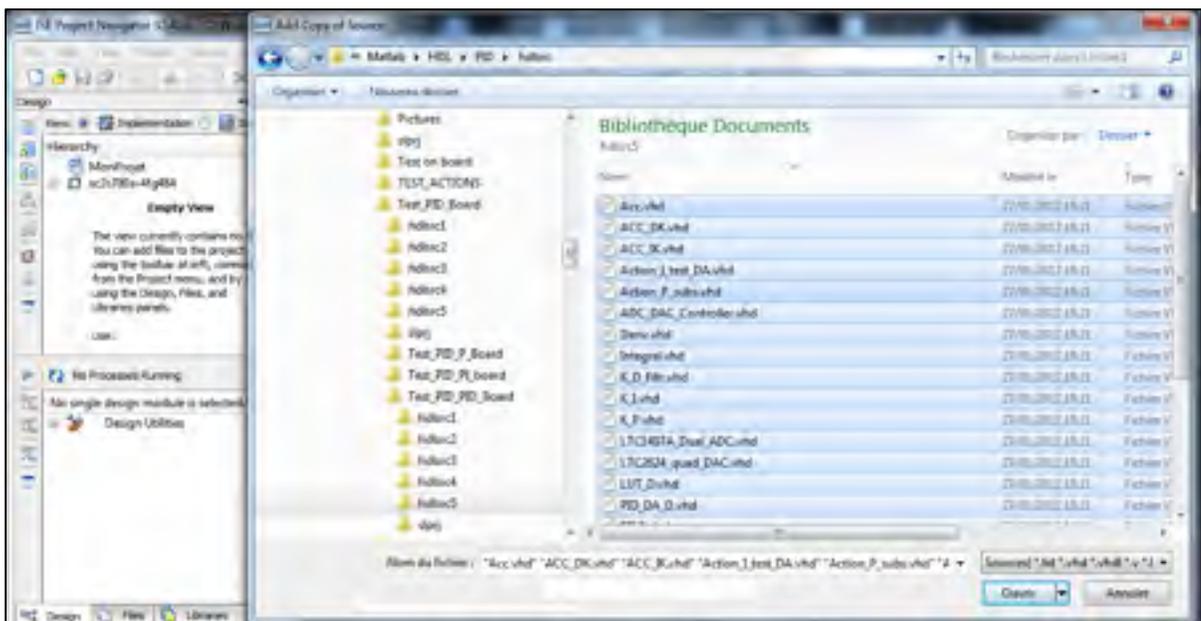


Figure-A I-IV Sélection des fichiers sources

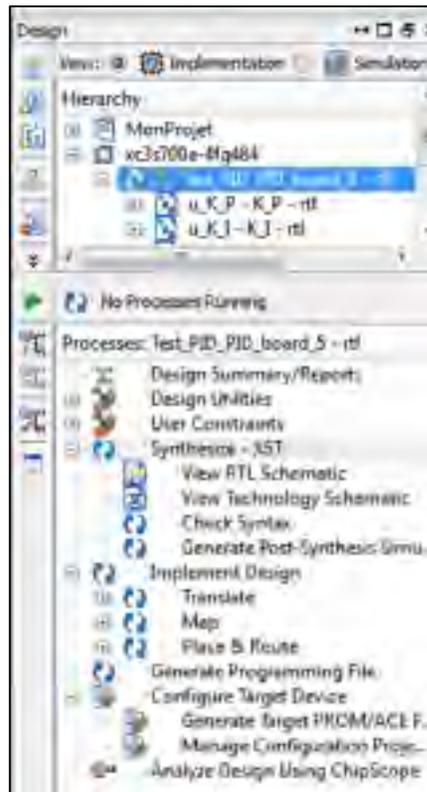


Figure-A I-VI Volet Gauche

Dans le logiciel « iMPACT », il faut mettre en place notre FPGA. D'abord, il faut relier la carte d'évaluation. Allez dans « Edit », choisissez « Launch Wizard ». Laissez l'option de reconnaissance automatique par JTAG. Cliquez sur OK.

Sur la fenêtre principale vous devriez voir apparaître un schéma. Celui-ci représente le FPGA. Il vous suffit de cliquer droit et de sélectionner « Assign New Configuration File ». Choisissez le répertoire de votre projet, le fichier de configuration (*.bit) doit s'y trouver. Cliquez sur OK sur toutes les fenêtres qui s'ouvrent. Une jauge apparaît enfin indiquant l'état de téléchargement du programme dans le FPGA.

ANNEXE II

COMPLÉMENTS SUR L'ESPACE D'ÉTAT

Dans cette annexe, nous nous proposons d'approfondir les notions d'espace d'état présentées dans le corps du document. Nous présenterons notamment quelques preuves et démonstrations sur ce qui a été énoncé.

PASSAGE D'UNE REPRÉSENTATION D'ÉTAT A UNE FONCTION DE TRANSFERT

Un système décrit par l'équation (2.57) est transposable dans l'espace de Laplace :

$$sX(s) - sx(0) = AX(s) + BU(s)$$

Si bien que le système devient à l'aide la relation précédente :

$$\begin{aligned} X(s) &= (sI - A)^{-1}x(0) + (sI - A)^{-1}BU(s) \\ Y(s) &= C(sI - A)^{-1}x(0) + (C(sI - A)^{-1}B + D)U(s) \end{aligned}$$

Ainsi pour des conditions initiales, $x(0)$, nulles on a :

$$Y(s) = (C(sI - A)^{-1}B + D)U(s)$$

Et la définition de la fonction de transfert, G , du système :

$$\frac{Y(s)}{U(s)} = C(sI - A)^{-1}B + D$$

On peut également remonter à la réponse temporelle, $y(t)$, donnée par :

$$y(t) = \mathcal{L}^{-1}[C(sI - A)^{-1}]x(0) + \mathcal{L}^{-1}(C(sI - A)^{-1}B + D)U(s)$$

PASSAGE D'UNE ÉQUATION DE TRANSFERT À UNE REPRÉSENTATION D'ÉTAT

Nous avons exprimé une fonction de transfert standardisée (dont le coefficient du terme à la puissance la plus élevée est unitaire) :

$$G(s) = \frac{Y(s)}{U(s)} = \frac{\sum_{j=0}^m \tilde{a}_j s^j}{\sum_{i=0}^n \tilde{b}_i s^i}$$

avec $m \leq n$

En d'autres termes nous avons :

$$\left(\sum_{i=0}^n \tilde{b}_i s^i \right) Y(s) = \left(\sum_{j=0}^m \tilde{a}_j s^j \right) U(s)$$

En développant :

$$(s^n + \tilde{b}_{n-1} + \dots + \tilde{b}_1 s + \tilde{b}_0)Y(s) = (\tilde{a}_m s^m + \dots + \tilde{a}_0)U(s)$$

Définissons un sous-système X tel que :

$$(s^n + \tilde{b}_{n-1}s + \dots + \tilde{b}_1s + \tilde{b}_0)X(s) = U(s)$$

L'expression précédente est à exprimer dans le temps (sous la forme d'une ED) :

$$\left(\frac{d^n}{dt^n} + \tilde{b}_{n-1} \frac{d^{n-1}}{dt^{n-1}} + \dots + \tilde{b}_0 \right) x(t) = u(t)$$

On peut alors exprimer la dérivée n-ième de $x(t)$ en fonction des autres dérivées de l'expression et de u :

$$\frac{d^n x(t)}{dt^n} = -\tilde{b}_{n-1} \frac{d^{n-1}}{dt^{n-1}} x(t) - \dots - \tilde{b}_0 x(t) + u(t)$$

On pose alors un vecteur modélisant les états, le vecteur d'état x , tel que :

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

En posant que :

$$x_n = x(t)$$

Et que les termes répondent à

$$x_k = \frac{d^{(n-k)} x(t)}{dt^{(n-k)}} = \dot{x}_{k+1}$$

Cette dernière formulation implique que la dérivée du premier état x_1 , noté \dot{x}_1 vaut :

$$\dot{x}_1 = \frac{d}{dt} x_1 = \frac{d}{dt} \left(\frac{d^{n-1} x(t)}{dt^{n-1}} \right) = \frac{d^n x(t)}{dt^n}$$

Or nous avons vu que l'on pouvait donner une représentation du terme dérivé n-fois par les autres dérivés et l'entrée :

$$\begin{aligned} \dot{x}_1 &= -\tilde{b}_{n-1} \frac{d^{n-1}}{dt^{n-1}} x(t) - \dots - \tilde{b}_0 x(t) + u(t) \\ &= -\tilde{b}_{n-1} x_1 - \dots - \tilde{b}_0 x_n + u(t) \end{aligned}$$

Avec la représentation vectorielle de x il est facile de conclure sur la constitution des matrices A et B :

$$A = \begin{pmatrix} -\tilde{b}_{n-1} & -\tilde{b}_{n-2} & \dots & \tilde{b}_1 & \tilde{b}_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Avec

$$\dot{x} = Ax + Bu$$

Il nous reste à définir la sortie du système précédent et l'on voit que :

$$y(t) = \left(\tilde{a}_m \frac{d^m}{dt^m} + \dots + \tilde{a}_0 \right) x(t)$$

$$y = C \cdot x$$

Avec

$$C = (0 \quad \dots \quad \tilde{a}_m \quad \dots \quad \tilde{a}_0)$$

FORME DE COMMANDABILITE

Revenons sur la structure dite de commandabilité. Nous avons rappelé la façon dont se construit la matrice de commandabilité \mathbb{C} .

Pour rappel, nous avons un système de la forme :

$$A = \begin{pmatrix} -\tilde{b}_{n-1} & -\tilde{b}_{n-2} & \dots & \tilde{b}_1 & \tilde{b}_0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Donc la matrice \mathbb{C} s'écrit :

$$\mathbb{C} = (B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B)$$

La matrice \mathbb{C} est alors de rang de plein. Ainsi le système décrit par une forme canonique de commandabilité est complètement commandable.

Preuve

Soit la matrice de commandabilité définie par l'équation (2.XX). Celle-ci peut s'écrire comme :

$$\mathbb{C} = ({}^0c \quad {}^1c \quad \dots \quad {}^{n-1}c) \quad (8.1)$$

Où les éléments ${}^k c$ pour $k \in \llbracket 0; n-1 \rrbracket$ représentent les colonnes de la matrice \mathbb{C} et répondent à la définition récurrente suivante :

$${}^k c_i = \sum_j a_{ij} {}^{k-1} c_j \quad (8.2)$$

On se propose de montrer que la matrice de commandabilité est de type triangulaire supérieure avec une diagonale unitaire. Autrement dit :

$$\begin{aligned} {}^k c_i &= 1 \text{ pour } i = k + 1 \\ &= 0 \text{ pour } i > k + 1 \end{aligned}$$

Raisonnons par récurrence :

Initialisation

$${}^0 c = B = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Donc

$$\begin{aligned} {}^0 c &= 1 \text{ pour } i = j = 1 \\ &= 0 \text{ pour } i > j = 1 \end{aligned}$$

$${}^1 c_i = \sum_j a_{ij} {}^0 c_j = \sum_j a_{ij} b_j$$

La structure de la matrice A impose que sur une ligne, i , hormis la première, le seul élément atteint est l'élément de la diagonale unitaire inférieure localisé à la colonne $j = i - 1$, réciproquement l'élément non nul et unitaire de la matrice A est localisé à la ligne $i = j + 1$ pour $i > 1$, i.e $a_{21} = 1$. Par ailleurs, $b_j = 1$ pour $j = 1$ donc le terme :

$$a_{21} b_1 = 1$$

De plus $b_j = 0$ pour tout $j > 1$ donc le produit $(a_{ij} b_j)$ sera nul pour tout $j > i - 1$. Donc on a :

$$\begin{aligned} {}^1 c_i &= 1 \text{ pour } i = 2 \\ &= 0 \text{ pour } i > 2 \end{aligned}$$

Au rang k :

$${}^k c_i = \sum_j a_{ij} {}^{k-1} c_j$$

La structure de la matrice A impose que sur une ligne i , hormis la première, le dernier élément atteint est l'élément de la diagonale inférieure à la colonne $j = i - 1$.

De plus ${}^{k-1} c_j = 0$ pour tout $j > k$ donc le produit $(a_{ij} {}^{k-1} c_j)$ sera nul pour toutes les lignes i tel que $i > k + 1$.

En outre, le terme ${}^{k-1} c_j = 1$ pour $j = k$. De plus la structure de la matrice A impose que le seul élément atteint sur une ligne i est situé la colonne $j = i - 1$ i.e $a_{k+1, k} = 1$. Donc à la ligne $i = k + 1$:

$$\begin{aligned}
{}^k c_{k+1} &= \sum_j a_{k+1 j} {}^{k-1} c_j \\
&= a_{k+1 k} {}^{k-1} c_k \\
&= 1
\end{aligned}$$

Au final :

$$\begin{aligned}
{}^k c_i &= 1 \text{ si } i = k + 1 \\
&= 0 \text{ si } i > k + 1
\end{aligned}$$

Au rang $k+1$:

$${}^{k+1} c_i = \sum_j a_{ij} {}^k c_j$$

La structure de la matrice A impose que sur une ligne hormis la première, le seul élément atteint est l'élément de la diagonale inférieure situé à la colonne $j = i - 1$ i.e $a_{k+2 k+1} = 1$ et comme ${}^k c_{k+1} = 1$ on a

$${}^{k+1} c_{k+2} = 1$$

De plus ${}^k c_j = 0$ pour tout $j > k + 1$ donc le produit $(a_{ij} {}^k c_j)$ sera nul pour toutes les lignes i tel que $i > k + 2$. Autrement dit :

$$\begin{aligned}
{}^{k+1} c_i &= 1 \text{ si } i = k + 2 \\
&= 0 \text{ si } i > k + 2
\end{aligned}$$

Nous avons donc montré par récurrence la forme de la matrice de commandabilité \mathbb{C} qui est triangulaire supérieure avec des éléments diagonaux unitaires. Le rang d'une telle matrice est alors nécessairement plein. ■

DISCRÉTISATION D'UN SYSTEME CONTINU

Un système continu décrit par les matrices A , B , C et D peut être discrétisé. Bensoussan (2007) propose une démonstration :

A partir de l'expression (A-III.X) on est en mesure d'écrire que dans un système discret l'entrée u garde sa valeur pendant une période d'échantillonnage T_E .

$$x(T) = \mathcal{L}^{-1}[(sI - A)^{-1}]_{t=T_E} x(0) + \mathcal{L}^{-1} \left[\frac{1}{s} (sI - A)^{-1} B \right]_{t=T_E} u(0)$$

La formule précédente donne la réponse de la variable d'état à une entrée u constante (donc présence du terme $1/s$).

Donc globalement la réponse à l'instant $k+1$ est donné par :

$$x(k + 1) = \mathcal{L}^{-1}[(sI - A)^{-1}]_{t=T_E} x(k) + \mathcal{L}^{-1} \left[\frac{1}{s} (sI - A)^{-1} B \right]_{t=T_E} u(k)$$

Au regard de la formulation :

$$x(k+1) = A_d x(k) + B_d u(k)$$

On identifie :

$$A_d = \mathcal{L}^{-1}[(sI - A)^{-1}]_{t=T_E}$$

$$B_d = \mathcal{L}^{-1}\left[\frac{1}{s}(sI - A)^{-1}B\right]_{t=T_E}$$

Toutefois une autre approximation est possible. La solution générale en continu est donnée par :

$$x(t) = e^{At}x(0) + \int_0^t e^{T_E(t-t')}Bu(t')dt'$$

En considérant que l'entrée u est constante entre kT_E et $(k+1)T_E$ et que $x(kT)$ est une valeur initiale, on peut écrire :

$$x(k+1) = e^{AT_E}x(kT_E) + \int_{kT_E}^{(k+1)T_E} e^{A(t-t')}Bu(t')dt'$$

Or l'intégrale vaut en $t = (k+1)T_E$:

$$\int_{kT_E}^{(k+1)T_E} e^{A(t-t')}Bu(t')dt' = \int_0^{T_E} e^{At}Bdt$$

En posant

$$B_d = \int_0^{T_E} e^{At}Bdt$$

$$A_d = e^{AT_E}$$

Pour une période d'échantillonnage suffisamment petite, il est possible de donner une approximation de A_d et B_d à l'aide de la série de Taylor.

$$e^{AT_E} = I + AT + \frac{A^2T^2}{2!} + \dots + \frac{A^nT^n}{n!}$$

$$\int_0^{T_E} e^{At}Bdt = A^{-1}[e^{AT_E} - 1]B = \left(T + \frac{AT^2}{2!} + \dots + \frac{A^{n-1}T^n}{n!}\right)B$$

ANNEXE III

CIRCUIT D'INSTRUMENTATION

Nous l'avons vu (figure 4.1), nous avons mis en place un circuit d'instrumentation permettant l'interface entre la carte d'évaluation et le banc de test. Ce circuit permettant d'adapter les signaux en provenance du CNA vers la servovalve et ceux provenant du capteur de position vers le CAN.

Adaptation Capteur vers CAN :

Nous avons donné dans le corps du document l'équation permettant l'adaptation de tension au niveau du CAN :

$$v_{CAN} = 2,976 \times V_y - 5,79$$

Il convient à présent de l'expliciter. Rappelons que la tension de sortie est comprise dans l'intervalle $[2,08 ; 2,92]V$ tandis que le CAN dispose d'une plage d'entrée comprise entre $[0,4 ; 2,9]V$.

On cherche la fonction linéaire f_{CAN} telle que :

$$f_{CAN} : V_y \in [2,08 ; 2,92]V \rightarrow v_{CAN} \in [0,4 ; 2,9]V$$

La forme de f_{CAN} est de type affine :

$$v_{CAN} = a_{CAN} V_y + b_{CAN}$$

On trouve alors :

$$a_{CAN} = \frac{2,9 - 0,4}{2,92 - 2,08} = 2,976$$

$$b_{CAN} = -5,79$$

Pour réaliser dans la pratique cette implémentation, nous utilisons un montage à l'aide d'amplificateur opérationnel (AOP). Nous réalisons un circuit non inverseur à offset comme indiqué dans la figure-A-IV-1.

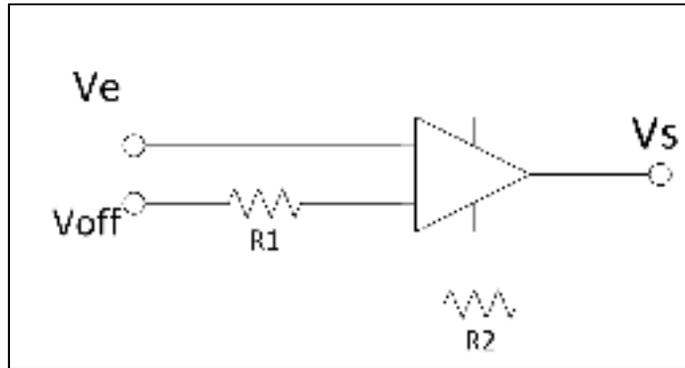


Figure-A III-I Montage AOP Non Inverseur avec offset

La tension de sortie s'exprime par :

$$V_s = \left(\frac{R_2}{R_1} + 1 \right) V_e - \frac{R_2}{R_1} V_{off}$$

Dans notre cas on identifie :

$$a_{CAN} = \frac{R_2}{R_1} + 1$$

$$b_{CAN} = \frac{R_2}{R_1} V_{off}$$

On a donc :

$$\frac{R_2}{R_1} = 1,976 \Rightarrow R_2 = 1,976 R_1$$

$$V_{off} = \frac{b_{CAN}}{1,976} = 2,93 V$$

Afin de générer le signal référence, V_{off} , nous utilisons un pont diviseur de tension muni d'un suiveur. Disposant d'une tension $V_{cc} = 5V$ sur la carte d'évaluation, nous effectuons le montage décrit par la Figure-A III-II

$$V_{off} = \frac{R_{inf}}{R_{inf} + R_{sup}} V_{cc}$$

Prenons $R_{inf} = 10k\Omega$:

$$R_{sup} = \frac{R_{inf}(V_{cc} - V_{off})}{V_{off}}$$

$$R_{sup} = 7,06k\Omega$$

L'ajout du suiveur permet de ne pas écrouler l'alimentation en ponctionnant trop de courant (ce qui rendrait le pont diviseur inutile).

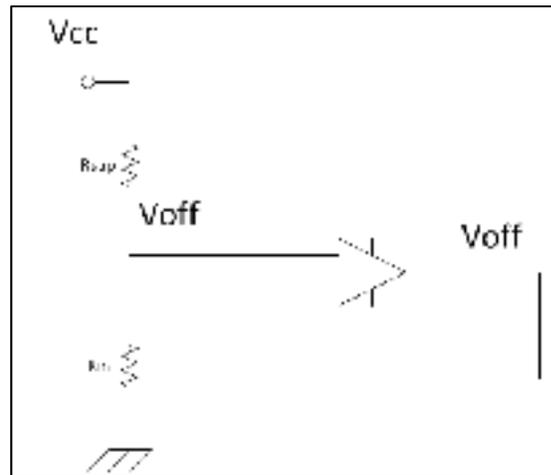


Figure-A III-II Montage diviseur de tension avec AOP en suiveur

Passons à présent sur le circuit nécessaire à l'adaptation du signal de commande transmis par CNA vers la servovalve. Nous avons donné (équation ())

$$u = 1,5152u_{DAC} - 2,5$$

Le CNA donne une tension de sortie comprise entre 0 et 3,3V. La servovalve, elle, nécessite une tension comprise en $\pm 2,5$. Raisonnant de la même façon que pour le CAN :

$$f_{CNA} : u_{CNA} \in [0 ; 3,3]V \rightarrow u \in [-2,5 ; 2,5]V$$

$$u = a_{CNA}u_{CNA} + b_{CNA}$$

$$a_{CNA} = \frac{5}{3,3} = 1,5152$$

$$b_{CNA} = -2,5$$

En procédant de la même façon, un AOP monté en non inverseur avec une tension d'offset générée à l'aide d'un diviseur de tension associé à un montage suiveur. Nous avons :

$$R_4 = 0,5152 R_3$$

$$V_{off2} = 4,85V$$

En prenant une résistance $R_{inf2} = 10k$, nous trouvons une résistance $R_{sup2} = 310k$.

Notons que nous avons utilisé pour le montage pratique des résistances disposant de valeurs à $\pm 5\%$. C'est pourquoi nous avons inséré des potentiomètres afin d'affiner les valeurs des résistances.

Tableau-A III-I Données expérimentales et théoriques CA2

VCNA (V)	u_experim (V)	u_théorique (V)	VCNA (V)	u_experim (V)	u_théorique (V)
0	-2,496	-2,5	1,7	0,081	0,0758
0,3	-2,04	-2,0454	1,8	0,233	0,2274
0,4	-1,888	-1,8939	1,888	0,362	0,3607
0,5	-1,736	-1,7424	1,995	0,527	0,5228
0,6	-1,584	-1,5909	2,101	0,6877	0,6834
0,7	-1,434	-1,4394	2,2	0,842	0,8334
0,8	-1,28	-1,2878	2,365	1,087	1,0834
0,9	-1,13	-1,1363	2,5	1,29	1,288
1	-0,977	-0,9848	2,6	1,448	1,4395
1,1	-0,829	-0,8333	2,7	1,603	1,591
1,2	-0,676	-0,6818	2,8	1,748	1,7426
1,3	-0,524	-0,5302	2,898	1,896	1,891
1,4	-0,372	-0,3787	2,9	1,897	1,8941
1,5	-0,218	-0,2272	3	2,05	2,0456
1,6	-0,067	-0,0757	3,1	2,202	2,1971
1,648	0,002	-0,003	3,2	2,358	2,3486
1,65	0,003	0,0001	3,298	2,496	2,4971
1,685	0,056	0,0531			

Par ailleurs, pour valider le montage, nous avons réalisé des tests sur chacun des deux circuits d'adaptation. La Figure-A III-III montre que le montage expérimental est efficace. Nous avons un écart moyen entre la valeur théorique et expérimentale de l'ordre de 5mV, à ce titre le montage nous permet satisfaisant.

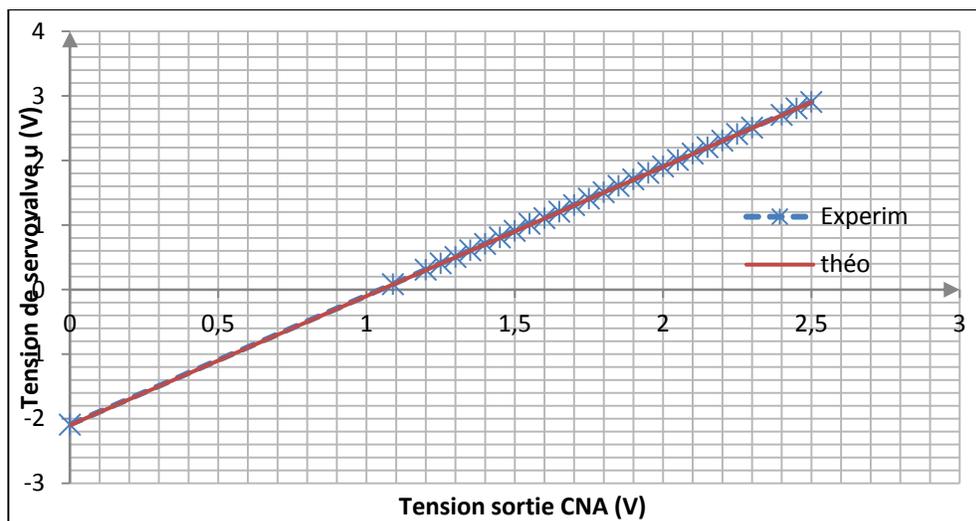


Figure-A III-III Tension de commande en fonction de la tension CNA

Tableau-A III-II Résultats expérimentaux et théoriques CA1

Vcapter	VCAN_Exp	V_CAN_theo
0	-5,86	-5,79
1,8	-0,443	-0,4332
2,083	0,401	0,409008
2,1	0,452	0,4596
2,3	1,063	1,0548
2,4	1,348	1,3524
2,45	1,505	1,5012
2,502	1,651	1,655952
2,55	1,807	1,7988
2,6	1,955	1,9476
2,705	2,268	2,26008
2,903	2,865	2,849328
3,007	3,18	3,171

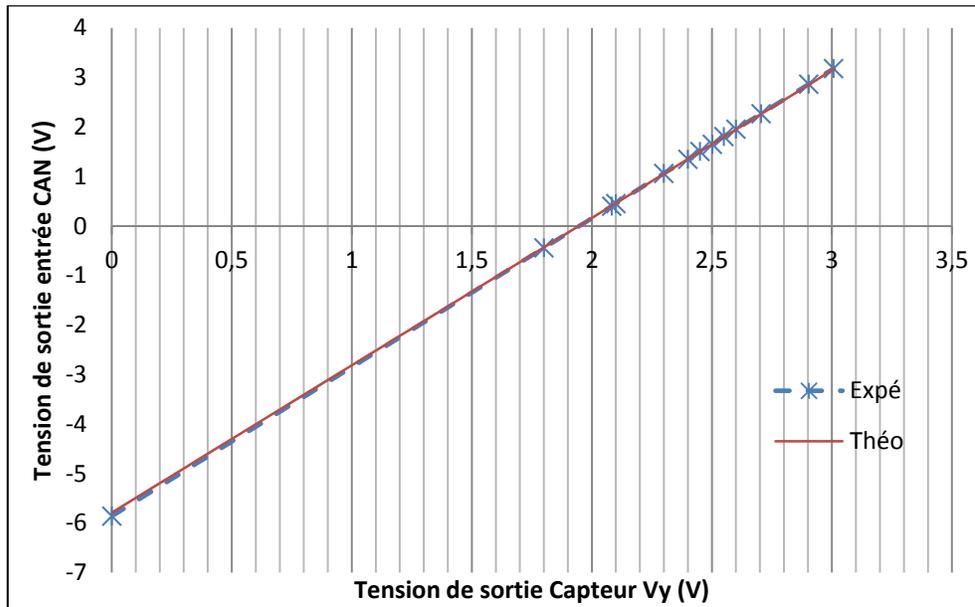


Figure-A III-IV Tension CAN en fonction de la tension de sortie (CA1)

Le Tableau-A III-II et la Figure-A III-IV montrent que le montage expérimental est correct. De plus on note un écart moyen à la valeur théorique inférieur à 10mV.

ANNEXE IV

GESTION DES TYPES NUMÉRIQUES (POINT FIXE)

On se propose de résumer la description des types numériques au sein de Simulink et de voir les résultantes en VHDL. La gestion des types numérique est l'élément essentiel à ne pas négliger lors de la conception.

Tableau-A IV-I Types générés à partir de HDL Coder sur la base d'un bloc Simulink « Constant » avec pour valeur 12 en décimal

Simulink	VHDL	Valeur
uint8	<code>unsigned(7DOWNT00);-- uint8</code>	<code>to_unsigned(12,8);</code>
int8	<code>signed(7DOWNT00);-- int8</code>	<code>to_signed(12,8);</code>
fixdt(1,8,0)	<code>signed(7DOWNT00);-- int8</code>	<code>to_signed(12,8);</code>
fixdt(1,8)_signed	<code>signed(7DOWNT00);-- sfix8_En3</code>	<code>to_signed(96,8);</code>
fixdt(0,8)_unsigned	<code>unsigned(7DOWNT00);-- ufix8_En4</code>	<code>to_unsigned(192,8);</code>
boolean	<code>std_logic</code>	
fixdt(1,8,1)	<code>signed(7DOWNT00);-- sfix8_En1</code>	<code>to_signed(24,8);</code>

A partir du tableau précédent et de ce qui a été vu, regardons les valeurs de la constante « 12 » lors de la génération du code VHDL et plus particulièrement pour des types : `fixdt(x,8)` et `fixdt(1,8,1)`.

<code>fixdt(1,8)_signed</code>	<code>signed(7DOWNT00);-- sfix8_En3</code>	<code>to_signed(96,8);</code>
--------------------------------	--	-------------------------------

Dans le code on voit la conversion qui s'effectue par :

`to_signed(96,8);`

De plus on distingue le commentaire suivant : `-- sfix8_En3`

Donc il s'agit d'un nombre signé disposant de trois chiffres après la virgule.

$$96|_{10} = 0110\ 0000|_2$$

En incluant la virgule on obtient :

$$01100,000|_2 = 2^{-3} \cdot (2^5 + 2^6)|_{10} = 12$$

<code>fixdt(0,8)_unsigned</code>	<code>unsigned(7DOWNT00);-- ufix8_En4</code>	<code>to_unsigned(192,8);</code>
----------------------------------	--	----------------------------------

De la même façon, le code est cette fois transcrit comme :

$$12|_{10} = 1100,0000|_2 = 2^{-4}(2^7 + 2^6)|_{10}$$

<code>fixdt(1,8,1)</code>	<code>signed(7DOWNT00);-- sfix8_En1</code>	<code>to_signed(24,8);</code>
---------------------------	--	-------------------------------

De la même manière, ici 12 est codé sur 8 bits avec une décimale.

Ainsi les valeurs issues du tableau prennent leur sens et leur origine peut être expliquée.
Dans l'aide on trouve :

a = fixdt(Signed, WordLength)

a = fixdt(Signed, WordLength, FractionLength)

a = fixdt(Signed, WordLength, TotalSlope, Bias)

a = fixdt(Signed, WordLength, SlopeAdjustmentFactor,
FixedExponent, Bias)

a = fixdt(DataTypeNameString)

ANNEXE V

PRÉSENTATION DE LA BIBLIOTHÈQUE

La bibliothèque créée est insérée au sein de l'explorateur de bibliothèques de Simulink.

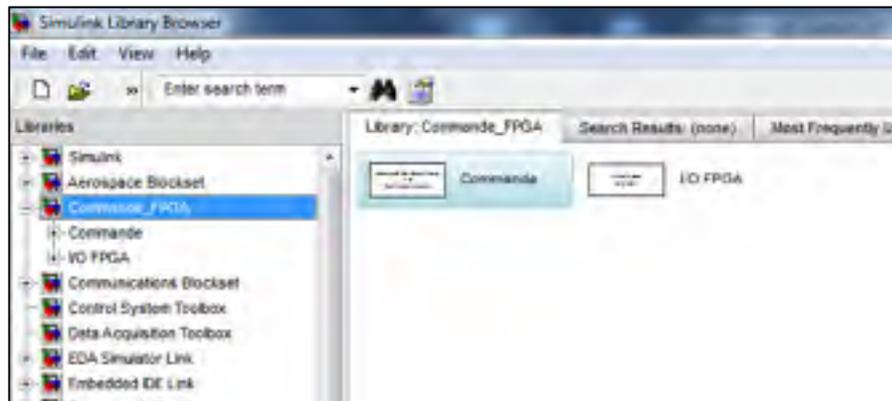


Figure-A V-I La bibliothèque dans Simulink

La Figure-A V-I montre que la bibliothèque créée se trouve dans Simulink. Cela permet de disposer des éléments facilement lors de la conception de modèles.

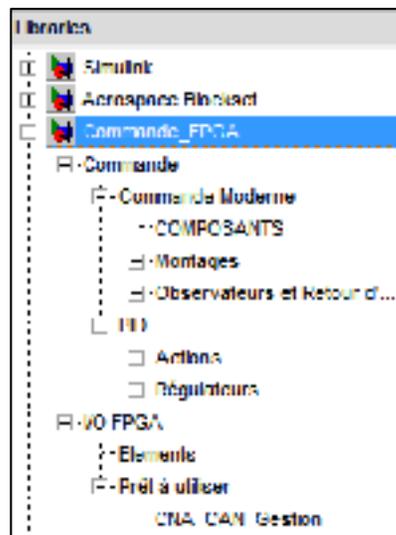
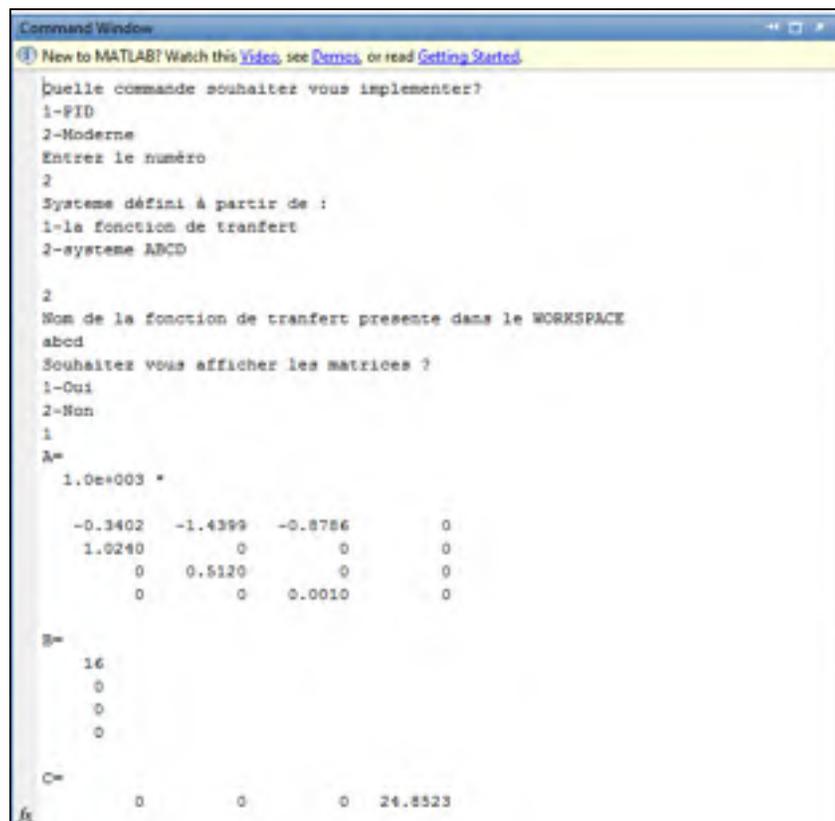


Figure-A V-II Arborescence de la bibliothèque

La Figure-A V-II présente l'arborescence de la bibliothèque créée. On y distingue deux principales rubriques comme présenté dans la partie (7.2). Il suffit alors de choisir un composant afin de l'insérer dans le modèle. Dans chaque partie, des modèles préconstruits sont disponibles. Cela permet de concevoir rapidement de nouveaux systèmes.

La bibliothèque doit être utilisée avec un script. Celui-ci permet de définir le système à l'étude. Le script s'opère dans la fenêtre des commandes de Matlab et est fait de telle sorte que l'utilisateur peut paramétrer son système. En effet, une série de questions est posée et permet de définir le type de commande que le concepteur souhaite mettre en place.



```

Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

[Quelle commande souhaitez vous implementer?
1-FID
2-Moderne
Entrez le numéro
2
Systeme défini à partir de :
1-la fonction de transfert
2-systeme ABCD

2
Nom de la fonction de transfert presente dans le WORKSPACE
abcd
Souhaitez vous afficher les matrices ?
1-Oui
2-Non
1
A=
 1.0e+003 *
   -0.3402   -1.4399   -0.8786     0
    1.0240     0         0         0
     0     0.5120     0         0
     0         0     0.0010     0

B=
 16
 0
 0
 0

C=
 0     0     0 24.8523

```

Figure-A V-III Exemple d'utilisation du script

La Figure-A V-III illustre l'interface utilisateur du script (mode console). Une fois le système décrit il est nécessaire de définir les tailles des variables codées en point fixes. Pour illustrer l'ajustement de la taille des variables par l'utilisateur, on propose de regarder l'exemple de

l'observateur par AD. En se basant sur les Figures 6.14 et 6.15, on distingue plusieurs éléments :

- Blocs de conversion : convertit de *double* en point fixe (*taille, ftaille*) ;
- PISO : sérialise des éléments codées en point fixe (*taille,ftaille*) ;
- Adressage : définit par la taille du système ;
- LUT : valeurs stockée en point fixe (*tailleLUT ; ftailleLUT*) ;
- Accumulateur : calcule par décalages successifs (*tailleAcc,ftailleAcc*).

Le code source suivant permet la définition par l'utilisateur des tailles de variables mises en jeu.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Variables observateur %%%%%%%%%
% Taille Totale
      taille = input('NB bits conv,PISO');
% Taille Fraction
      ftaille = input('NB bits apres virgule conv, PISO');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% ETAGE 2 %%%%%%%%%
%%%%%%%% Constantes LUT %%%%%%%%%
% Taille LUT Totale
      tailleLUT = input('NB bits LUT');
% Taille Fraction LUT
      ftailleLUT = input('NB bits apres virgule LUT');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% ETAGE 3 %%%%%%%%%
%%%%%%%% Tailles accumulateurs %%%%%%%%%
% Taille Totale Acc
      tailleAcc = (taille-ftaille)+(tailleLUT-ftailleLUT)-1;
% Taille fraction totale Acc
      ftailleAcc = ftailleLUT;

```

On a au final un modèle fiable car les erreurs de gestion des types sont minimisées et facilement modifiables en fonction des résultats de simulation.

BIBLIOGRAPHIE

- Allard, Bruno. 2009. *Electronique Digitale*. Coll. « Notes de cours ». Lyon: INSA Lyon Génie électrique.
- Aslangul, Claude. 2008. *Mathématiques pour physiciens*. Coll. « Notes de cours ». <http://www.phys.ens.fr/enseign/fip/cours/FIPCoursMathsCIA_Ch01.pdf>.
- Åström, K. J., et T. Hägglund. 2001. « The future of PID control ». *Control Engineering Practice*, n° 9, p. 1163-1175.
- Auger, D. 2008. « Programmable hardware systems using model-based design ». In *Programmable Hardware Systems, 2008 IET and Electronics Weekly Conference on* (8-9 Oct. 2008). p. 1-12.
- Baghel, S., et R. Shaik. 2011. « FPGA implementation of Fast Block LMS adaptive filter using Distributed Arithmetic for high throughput ». In *Communications and Signal Processing (ICCSP), 2011 International Conference on* (10-12 Feb. 2011). p. 443-447.
- Barlas, T., et M. Moallem. 2008. « Next generation of embedded controllers: developing FPGA-based reconfigurable controllers using Matlab/Simulink ». In *13th IEEE International Conference on Emerging Technologies and Factory Automation, 15-18 Sept. 2008*. p. 1055-8. Coll. « 13th IEEE International Conference on Emerging Technologies and Factory Automation ». Piscataway, NJ, USA: IEEE. <<http://dx.doi.org/10.1109/ETFA.2008.4638523>>.
- Bensoussan, David. 2008. *Commande Moderne, Approche par modèles continus et discrets*. Presses Internationales Polytechnique.
- Besançon-Voda, A., et S. Gentil. 1999. *Régulateurs PID analogiques et numériques*. 25. (Les Techniques de l'ingénieur).
- Brettle, J., et A. Kruger. 2009. « Higher-level development and COTS hardware expand FPGA boundaries ». In *AUTOTESTCON, 2009 IEEE* (14-17 Sept. 2009). p. 121-124.
- Brown, S. 1996. « FPGA architectural research: a survey ». *Design & Test of Computers, IEEE*, vol. 13, n° 4, p. 9-15.
- Burghartz, J. N., M. Irmscher, F. Letzkus, J. Kretz et D. Resnick. 2006. « Lithography for the 32-nm Node and Beyond ». In *Bipolar/BiCMOS Circuits and Technology Meeting, 2006* (8-10 Oct. 2006). p. 1-5.

- Chan, Y. F., M. Moallem et W. Wang. 2004. « Efficient implementation of PID control algorithm using FPGA technology ». In *Decision and Control, 2004. CDC. 43rd IEEE Conference on* (14-17 Dec. 2004). Vol. 5, p. 4885-4890 Vol.5.
- Chan, Y. F., M. Moallem et W. Wang. 2007. « Design and Implementation of Modular FPGA-Based PID Controllers ». *IEEE Transactions on industrial electronics*, vol. 54, n° 4 (Aout), p. 1898-1906.
- Chander, S., P. Agarwal et I. Gupta. 2010. « FPGA-based PID controller for DC-DC converter ». In *Power Electronics, Drives and Energy Systems (PEDES) & 2010 Power India, 2010 Joint International Conference on* (20-23 Dec. 2010). p. 1-6.
- Charaabi, L., E. Monmasson et I. Slama-Belkhodja. 2002. « Presentation of an efficient design methodology for FPGA implementation of control systems. Application to the design of an antiwindup PI controller ». In *IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the]* (5-8 Nov. 2002). Vol. 3, p. 1942-1947 vol.3.
- Cong, J., Liu Bin, S. Neuendorffer, J. Noguera, K. Vissers et Zhang Zhiru. 2011. « High-Level Synthesis for FPGAs: From Prototyping to Deployment ». *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, n° 4, p. 473-491.
- Derrien, Steven. 2002. « Étude quantitative des techniques de partitionnement de réseaux de processeurs pour l'implantation sur circuits FPGA ». Thèse de doctorat en ligne, Rennes, Université Rennes 1, 205 p. <<http://www.irisa.fr/cosi/Derrien/publi/these.pdf>>.
- Gagnon, Tommy. 2011. « Modélisation causale et acausale d'un système électro-hydraulique ». Montréal, Ecole de technologie supérieure.
- Garcia, O., P. Zumel, A. de Castro, J. A. Cobos et J. Uceda. 2004. « An automotive 16 phases DC-DC converter ». In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual* (20-25 June 2004). Vol. 1, p. 350-355 Vol.1.
- Gauthier, G. 2010. *Cours de SYS823 : Modélisation et Automatisation des procédés industriels* Coll. « Notes de cours ». Montréal.
- Gupta, Vikas, Kavita Khare et R.P Singh. 2009. *Efficient FPGA Design and Implementation of Digital PID Controllers in Simulink*.
- Han, Y., R. Tzoneva et S. Behardien. 2007. « MATLAB, LabVIEW and FPGA linear control of an inverted pendulum ». In *AFRICON 2007* (26-28 Sept. 2007). p. 1-7.

- Hennessy, John, et David Patterson. 1995. *Architecture des ordinateurs : une approche quantitative*, Traduction de l'édition américaine : Computer Architecture. A quantitative Approach. Paris: Ediscience, 751 p.
- IEEE. 1988. « IEEE Standard VHDL Language Reference Manual ». *IEEE Std 1076-1987*, p. 1-201.
- IEEE. 1994. « IEEE Standard VHDL Language Reference Manual ». *ANSI/IEEE Std 1076-1993*, p. 1-249.
- IEEE. 1996. « IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language ». *IEEE Std 1364-1995*, p. 1-666.
- IEEE. 1999. « IEEE Standard VHDL Analog and Mixed-Signal Extensions ». *IEEE Std 1076.1-1999*, p. 1-314.
- IEEE. 2005. « IEEE Standard for System Verilog- Unified Hardware Design, Specification, and Verification Language ». *IEEE Std 1800-2005*, p. 0_1-648.
- IEEE. 2007. « Approved Draft Standard VHDL Analog and Mixed-Signal Extensions (Revision of IEEE Std 1076.1-1999) ». *IEEE Approved Draft Std P1076.1/D3.3, Feb 6, 2007*.
- Intel. 2012. *Processors Guide Quick Reference*. En ligne. <<http://www.intel.com/pressroom/kits/quickreffam.htm>>. Consulté le 20 octobre 2011.
- Jullien, G. A., et M. A. Bayoumi. 1991. « A review of VLSI technologies in digital signal processing ». In *Circuits and Systems, 1991., IEEE International Symposium on* (11-14 Jun 1991). p. 2347-2350 vol.4.
- Kelly, Jamie, Vittal Rao, Hardy Pottinger et Clifford Bowman. 1997. « Design and implementation of digital controllers for smart structures using field programmable gate arrays ». *Smart Materials and Structures*, vol. 6, n° 5 (Octobre 1997), p. 559-572.
- Kirby, B., et H. Kang. 2008. « Model Based Design for Power Systems Protection Relays, Using Matlab & Simulink ». In *Developments in Power System Protection, 2008. DPSP 2008. IET 9th International Conference on* (17-20 March 2008). p. 654-657.
- Lazaro, J., A. Astarloa, J. Arias, U. Bidarte et A. Zuloaga. 2006. « Simulink/Modelsim Simulabel VHDL PID Core for Industrial SoPC Multiaxis Controllers ». In *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on* (6-10 Nov. 2006). p. 3007-3011.

- Li, Tianjian; Fujimoto, Yasutaka 2008. « FPGA-based Current Controller for High-speed Communication and Real-time Control System ». *IEEE*.
- Líčko, Miroslav; Schier, Jan; Tichý, Milan; Kühl Markus. 2003. « MATLAB/Simulink Based Methodology for Rapid-FPGA-Prototyping ».
- Mathworks. 2007. *Matlab/Simulink*. Mathworks.
- Matlab/Simulink. 2007. *Simulink HDL Coder*. The Mathworks.
- Maxfield, Clive. 2004. *The Design Warrior's Guide to FPGAs*. Burlington, MA (USA): Elsevier, 542 p.
- Mingyao, Ma, Li Wuhua et He Xiangning. 2010. « An FPGA-based mixed-signal controller for switching mode converters ». In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society* (7-10 Nov. 2010). p. 1703-1708.
- Monmasson, Eric; Cirstea, Marcian;. 2007. « FPGA Design Methodology for Industrial Control Systems - A Review ». *IEEE Transactions on Industrial Electronics*, vol. 54, n° 4 (August), p. 1824-1842.
- Moore, B. 1981. « Principal component analysis in linear systems: Controllability, observability, and model reduction ». *Automatic Control, IEEE Transactions on*, vol. 26, n° 1, p. 17-32.
- Moore, G. E. 1965. « Cramming More Components onto Integrated Circuits ». *Electronics*, vol. 38, n° 8, p. 114-117.
- Mosterman, P. 2007. « Model-Based Design of Embedded Systems ». In *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on* (3-4 June 2007). p. 3-3.
- Nicolescu, G., et P.J. Mosterman. 2009. *Model-Based Design for Embedded Systems*. CRC Press. <<http://books.google.ca/books?id=8Cjg2mM-m1MC>>.
- Nise, Norman. 2011. *Control Systems Engineering*, 6ème édition. John Wiley & Sons, 926 p.
- Odet, Christophe. 2009. *Traitement du Signal*. Coll. « Notes de cours en traitement du signal ». Lyon: INSA Lyon Génie Electrique.
- Ownby, M., et W. H. Mahmoud. 2003. « A design methodology for implementing DSP with Xilinx System Generator for Matlab ». In *System Theory, 2003. Proceedings of the 35th Southeastern Symposium on* (16-18 March 2003). p. 404-408.

- Paiz, Carlos, et Mario Porrman. 2007. « The Utilization of Reconfigurable Hardware to Implement Digital Controllers: a Review ». In *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on* (4-7 June 2007). p. 2380-2385.
- Patel, P., et M. Moallem. 2010. « Reconfigurable system for real-time embedded control applications ». *Control Theory & Applications, IET*, vol. 4, n° 11, p. 2506-2515.
- Peled, A., et Liu Bede. 1973. « A new approach to the realization of nonrecursive digital filters ». *Audio and Electroacoustics, IEEE Transactions on*, vol. 21, n° 6, p. 477-484.
- Perry, S. 2009. « Model Based Design needs high level synthesis - A collection of high level synthesis techniques to improve productivity and quality of results for model based electronic design ». In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.* (20-24 April 2009). p. 1202-1207.
- Rahul, Dubey, Agarwal Pramod et M. K. Vasantha. 2007. « Programmable Logic Devices for Motion Control—A Review ». *Industrial Electronics, IEEE Transactions on*, vol. 54, n° 1, p. 559-566.
- Retif, Jean-Marie. 2008. *Automatique Régulation*. Coll. « Notes de cours ».
- Riesgo, T., Y. Torroja et E. de la Torre. 1999. « Design methodologies based on hardware description languages ». *Industrial Electronics, IEEE Transactions on*, vol. 46, n° 1, p. 3-12.
- Rongqing, Li, Zhou Rong, Li Guangxin, He Weimin, Zhang Xiaoqian et T. J. Koo. 2011. « A Prototype of Model-Based Design Tool and Its Application in the Development Process of Electronic Control Unit ». In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual* (18-22 July 2011). p. 236-242.
- Ruei-Xi, Chen, Chen Liang-Gee et Chen Lilin. 2000. « System design consideration for digital wheelchair controller ». *Industrial Electronics, IEEE Transactions on*, vol. 47, n° 4, p. 898-907.
- Rugh, W. J. 1991. « Analytical framework for gain scheduling ». *Control Systems Magazine, IEEE*, vol. 11, n° 1, p. 79-84.
- Sanchez-Solano, S., A. J. Cabrera, I. Baturone, F. J. Moreno-Velo et M. Brox. 2007. « FPGA Implementation of Embedded Fuzzy Controllers for Robotic Applications ». *Industrial Electronics, IEEE Transactions on*, vol. 54, n° 4, p. 1937-1945.
- Santo, B. 2009. « 25 microchips that shook the world ». *Spectrum, IEEE*, vol. 46, n° 5, p. 34-43.

- Scrofano, R., Choi Seonil et V. K. Prasanna. 2002. « Energy efficiency of FPGAs and programmable processors for matrix multiplication ». In *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on* (16-18 Dec. 2002). p. 422-425.
- Shanblatt, Michael A.;Foulds , Brian;. 2005. « Proceedings of the 2005 IEEE International Conference on Microeletronic Systems Education (MSE'005) ». In.
- Stubkier, S., H. C. Pedersen, T. O. Andersen, A. Madsen et M. Dahl. 2011. « Analysis of and H ∞ controller design for an electro-hydraulic servo pressure regulator ». In *Fluid Power and Mechatronics (FPM), 2011 International Conference on* (17-20 Aug. 2011). p. 516-521.
- Trimberger, S. 1993. « A reprogrammable gate array and applications ». *Proceedings of the IEEE*, vol. 81, n^o 7, p. 1030-1041.
- Van Loan, C. 1978. « Computing integrals involving the matrix exponential ». *Automatic Control, IEEE Transactions on*, vol. 23, n^o 3, p. 395-404.
- White, S. A. 1989. « Applications of distributed arithmetic to digital signal processing: a tutorial review ». *ASSP Magazine, IEEE*, vol. 6, n^o 3, p. 4-19.
- Xilinx. 2006. « Spartan 3A FPGA Starter Kit User Guide ». En ligne.
- Xilinx. 2007. « Spartan-3A FPGA Starter Kit Board User Guide ».
- Xilinx. 2009. *Spartan-3 FPGA Family Data Sheet*, v2.5. En ligne. Coll. « Product Specification ». 217 p. <http://www.xilinx.com/support/documentation/spartan-3_data_sheets.htm>.
- Xilinx. 2011a. « High-Level Implementation of Bit- and Cycle-Accurate Floating-Point DSP Algorithms with Xilinx FPGAs ». *White Paper*, vol. 7 series FPGA, n^o 409 (Octobre).
- Xilinx. 2011b. « Spartan-3 Generation FPGA User Guide ». En ligne. <http://www.xilinx.com/support/documentation/spartan-3a_user_guides.htm>.
- Xilinx. 2012a. *Design Basics, What are FPGAs ?* En ligne. <<http://www.xilinx.com/company/gettingstarted/>>. Consulté le 24 janvier 2012.
- Xilinx. 2012b. *System Generator for DSP*. <<http://www.xilinx.com/tools/sysgen.htm>>. Consulté le 12 juillet 2011.
- Yu, C.C. . 1999. *Autotuning of PID Controllers*. Londres: Springer, 226 p.

- Zerbe, V., et B. Andelkovic. 2004. « Design flow for automated programming of FPGA ». In *Microelectronics, 2004. 24th International Conference on* (16-19 May 2004). Vol. 2, p. 715-718 vol.2.
- Zhang, Yuxi, Li Kang, Jun Wang, Jinping Sun et Zulin Wang. 2010. « Methods and experience of using Matlab and FPGA for teaching practice in digital signal processing ». In *Education and Management Technology (ICEMT), 2010 International Conference on* (2-4 Nov. 2010). p. 414-417.
- Zhengwei, Fang, J. E. Carletta et R. J. Veillette. 2005. « A methodology for FPGA-based control implementation ». *Control Systems Technology, IEEE Transactions on*, vol. 13, n° 6, p. 977-987.
- Zhixun, Ma, T. Friederich, Gao Jianbo et R. Kennel. 2011. « Model based design for System-On-Chip sensorless control of Synchronous Machine ». In *Sensorless Control for Electrical Drives (SLED), 2011 Symposium on* (1-2 Sept. 2011). p. 85-89.