N° d'ordre : 1905 Année 2002

THÈSE

présentée
pour obtenir le titre de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE
Spécialité : Génie Électrique

par

Régis RUELLAND

Ingénieur de l'École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Informatique et des Télécommunications

DEA Génie Électrique de l'INPT

Apport de la co-simulation dans la conception de l'architecture des dispositifs de commande numérique pour les systèmes électriques

soutenue le 27 septembre 2002 devant le jury composé de :

M.	Jean-Paul	Hautier	Président et Rapporteur
M.	Jean-Paul	Calvez	Rapporteur
M.	Alain	Lacarnoy	Examinateur
M.	Thierry	Meynard	Examinateur
M.	Guillaume	Gateau	Codirecteur
M.	Jean-Claude	Hapiot	Directeur de thèse

Thèse préparée au Laboratoire d'Électrotechnique et d'Électronique Industrielle de l'ENSEEIHT
UMR CNRS N° 5828

À Mes Parents

Apport de la co-simulation dans la conception de l'architecture
des dispositifs de commande numérique
pour les systèmes électriques

Mots clefs

- Co-simulation
- Systèmes électriques
- Architecture de commande numérique
- Intégration numérique d'algorithme de contrôle
- Électronique de puissance
- FPGA

Résumé

Ce travail est issu d'un constat concernant l'*Adéquation* entre les contraintes associées aux *Algorithmes* de commande et l'*Architecture* d'un dispositif de commande numérique (A^3). En effet, les contraintes temporelles et fonctionnelles de ces algorithmes se répercutent sur les choix architecturaux du dispositif. La répartition des tâches entre les parties "*câblées*" (composants logiques programmables) et les parties "*programmées*" (utilisation de microprocesseurs) ne peut plus se faire à priori étant donné les évolutions technologiques dans le domaine des composants numériques.

La première partie de ce mémoire est consacrée à la présentation et à l'analyse des tâches potentielles d'un dispositif de commande dédié aux systèmes électriques avec leurs contraintes fonctionnelles et temporelles. Dans un second temps, une synthèse sur le co-design, thème de recherche dans le domaine de l'électronique numérique, est développée afin de présenter différentes voies de recherche sur les outils d'aide à la conception des dispositifs de commande.

Dans la deuxième partie, un outil potentiel, nommé "*co-simulation*", est étudié afin d'en déterminer l'intérêt et les limites. Les différentes composantes nécessaires à la mise en place d'un environnement de co-simulation sont présentées et développées au travers d'exemples d'applications.

Une partie de l'environnement de co-simulation est ensuite exploitée dans la troisième section afin d'étudier différentes solutions pour l'implantation d'observateurs dédiés aux convertisseurs multicellulaires séries. Des résultats expérimentaux concernant l'implantation d'un émulateur temps réel du convertisseur viennent enfin confirmer la validité de l'environnement de co-simulation que nous avons réalisé. La dernière partie est réservée à la synthèse du travail ainsi qu'aux conclusions et perspectives

The co-simulation contribution in the architecture design of the digital control devices dedicated to electrical systems

Keywords

- Co-simulation
- Electrical systems
- Architecture of digital control
- Power electronics
- digital design of control laws
- FPGA

Abstract

This work results from a report about the lack of adequacy between the constraints of the control algorithms and the architecture of a control device. The timing and functional constraints of an algorithm has some consequences in the architectural choices of the control device. According to the technological progress in the electronic components, we cannot make the tasks distribution between the hardware part and the software part any more whithout particular analysis.

The first part is dedicated to the introduction and the analysis of the main potential tasks which can be implanted on a control device for the electrical systems with their timing and functional constraints. Then, a synthesis about co-design which is a subject of research in the electronic domain is made. This synthesis introduces various ways of research on tools of aided to the design of control devices.

In a second part, one of a possible tool is studied in order to show the interest and its limits. The different components used to build a co-simulation environment are presented and developped with some examples.

In a third part, co-simulation elements are used to study various solutions on the implantation of particular observers. These observers are dedicated to a four level multicell converter. Some experimental results about an emulator of the multicell converter (which is a part of the observers) are shown and allow to validate the co-simulation environment built. The last part is dedicated to the synthesis of the work and to the conclusions and perspectives.

Avant Propos

Le travail présenté dans ce mémoire a été effectué au Laboratoire d'Électrotechnique et d'Électronique Industrielle (L.E.E.I.), Unité Mixte de Recherche n°5828, au sein de l'École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications (E.N.S.E.E.I.H.T.).

Au terme de ces trois années et demi de recherche enrichissantes, je tiens à remercier toutes celles et tous ceux qui ont contribué à l'aboutissement de ce travail et en particulier :

- Monsieur J.P. Hautier, Directeur du Laboratoire d'Électrotechnique et d'Électronique de Puissance de Lille et Professeur au centre ENSAM de Lille, pour m'avoir fait l'honneur d'accepter la présidence de mon jury de thèse, mais aussi pour avoir accepté d'être rapporteur de ce mémoire et pour l'intérêt qu'il a porté sur ce travail.

- Monsieur J.P. Calvez, Professeur à l'École polytechnique de Nantes, et responsable du groupe de recherche MCSE, pour avoir accepté d'être rapporteur de ce mémoire, pour l'intérêt qu'il a porté sur ce travail, et pour son regard et ses remarques pertinentes en tant que spécialiste du Co-design.

- Monsieur Alain Lacarnoy, Ingénieur de Recherche au sein de la société Schneider Electric à Grenoble, pour avoir accepté de faire partie de notre jury de thèse, et pour le regard industriel qu'il a porté sur ce travail.

- Monsieur Maurice Fadel, responsable du groupe Commande et Diagnostic des Systèmes Électriques, pour m'avoir acceuilli au sein de son équipe de recherche.

- Monsieur Yvon Chéron, Directeur du L.E.E.I., pour m'avoir accueilli au sein de son laboratoire.

- Monsieur Thierry Meynard, Directeur de recherches au CNRS, pour avoir accepté de participer au jury de thèse, mais aussi pour nous avoir permis d'utiliser les outils de co-simulation sur une application aussi intéressante que l'observation des multicellulaires. En espérant que ces travaux trouveront une continuité et porteront leurs fruits... Pour son aide, sa gentillesse et m'avoir fait découvrir les joies de la moto.

- Monsieur J.C. Hapiot, Professeur à l'E.N.S.E.E.I.H.T., pour avoir accepté d'être directeur de cette thèse. Pour son soutien perpétuel et sa confiance malgré les moments de doute, pour ses conseils et la liberté d'action qu'il m'a offert. Je le remercie aussi pour sa disponibilité et son aide précieuse, notamment lors de la rédaction de ce manuscrit.

- Monsieur G. Gateau, Maître de Conférence à l'E.N.S.E.E.I.H.T., pour avoir accepté la difficile tâche d'être codirecteur de thèse, pour ses encouragements et son positivisme. Qu'il trouve dans ce mémoire ma profonde gratitude pour son aide inestimable et ses conseils éclairés. Qu'il sache que j'ai énormément apprécié travailler à ses côtés pour faire avancé le "Schmilblik"...

J'aimerais aussi remercier les gens avec qui j'ai eu le plaisir de collaborer durant ces trois années au laboratoire :

- Monsieur Bruno DAGUES, Maître de Conférence à l'E.N.S.E.E.I.H.T., de m'avoir accueilli au sein de l'équipe d'enseignants travaillant sur la plate-forme des Travaux Pratiques d'Automatique, mais aussi pour m'avoir encouragé à la fin de mon D.E.A.

- Messieurs Stéphane CAUX et Bruno SARENI, tous deux Maîtres de Conférence à l'E.N.S.E.E.I.H.T., mes deux compères en salle de Travaux Pratiques, pour leur grande disponibilité, leurs conseils sur l'automatique.

Je tiens aussi à remercier tout le personnel du L.E.E.I. et plus particulièrement mesdames Escaig, Bodden, Schawrz et Pionnie pour leur aide administrative et leur efficacité. Je remercie aussi Fatima Mebrek et Christelle Charron pour leurs disponibilité et leurs gentillesse.

Merci au "ciel" nommé plus communément Jean Hector, Jacques Benaïoun et Phillipe Azema, administrateurs des réseaux UNIX et NT pour avoir répondu présent en intervenant lorsqu'un problème informatique nous barrait le chemin.

J'adresse également mes remerciements à tous les thésards et stagiaires du LEEI, qui, à leur manière, ont contribué à l'aboutissement de ce travail. Je citerais particulièrement :

- Le radeau de Garona 99, The Team :
  - Cyrille Bas (pour les barbecues sur ton balcon), Yann Fefermann (toujours là quand il faut), Elie Lefeuvre (c'est Mac Gyver), Christophe Turpin (il ne dort jamais...), Thanate Khaorapapong (notre sculpteur sur polystyrène), Christophe, Sid-Ali Randi (la physique, la physique,...).
  - Vincent Devanneaux pour son amitié, nos nombreuses discussions constructives, et son sens du détail.
- Laurent Gasc, pour sa constante mauvaise humeur qui met de l'ambiance, et pour notre traversée de la rue Sainte-Catherine.
- Redha Bensaïd, l'automaticien ex-futur pizzaiolo, pour nos discussions techniques et sa gentillesse.
- Dominique Pinon pour ses encouragements et sa vision des choses
- Mes compagnons de Kebab Jérémi Régnier (arrête de chasser le rogers le midi) et Dominique Alejo, et à leur amitié.

Qu'ils trouvent dans ces quelques mots mes remerciements pour leurs gentillesses, les moments agréables que j'ai pu passer avec eux.

Enfin, Je tiens à remercier avec toute mon affection mes parents et mon frère. Que ce travail vienne récompenser toutes les années de soutien que vous avez pu m'apporter.

Mais je voudrais, remercier aussi celle qui a vécu et subit mes moments de doute et mes moments de joies, celle qui m'a soutenu tout au long de ce parcours. Son regard m'a aussi beaucoup apporté durant la rédaction de ce mémoire. À celle que j'aime,
À Danielle.

Régis Ruelland
le 25 février 2004

Table des matières

Table des figures

Liste des tableaux

Notations utilisées

Convertisseur multicellulaire

- R_{ch} : Résistance de la charge.
- L_{ch} : Inductance de la charge.
- R_f : Résistance du filtre de rééquilibrage.
- L_f : Inductance du filtre de rééquilibrage.
- C_f : Capacité du filtre de rééquilibrage.
- I_{ch} : Courant absorbé par la charge.
- I_f : Courant absorbé par le filtre de rééquilibrage.
- I_t : Courant total absorbé par le filtre et la charge.
- V_s : Tension de sortie du convertisseur.
- $T_d,\ f_d$: Période et fréquence de découpage.
- P_i : ordre de commande de l'interrupteur de la cellule i.
- $\overline{P}_i$: ordre de commande complémentaire à l'ordre P_i.
- u_i : Valeur moyenne de l'ordre de commande P_i sur une période de découpage T_d.
- r_e : Résistance du filtre d'entrée du convertisseur.
- L_e : Inductance du filtre d'entrée du convertisseur.
- C_e : Capacité du filtre d'entrée du convertisseur.
- δ_i : Différence entre les ordres de commande de deux cellules successives $\delta_1 = P_1 - P_2$.
- C_i : Capacité de la cellule i.
- Vc_i : Tension aux bornes du condensateur flottant de capacité. C_i
- Uc : Tension d'entrée du convertisseur avant filtrage.
- ϕ_i : Phase de l'ordre de commande de la cellule i
- $Vcell_i$: Tension aux bornes de la cellule de commutation i.

Machine à courant continu

- L_m : Inductance d'induit .
- R_m : Résistance d'induit .
- Kem : Constante de couple.
- Cem : Couple électromagnétique.
- Cr : Couple de charge.
- V_m : Tension aux bornes de l'induit.
- I_g : Courant d'induit circulant dans la génératice à courant continu servant de charge.
- Z : Résistance variable associée au plan de charge connecté à la génératrice à courant continu.
- Ω : Vitesse de rotation.
- $f_0.\Omega$: Force de frottement visqueux de l'ensemble (moteur, génératrice).
- $f_1.\Omega$: Force de frottement visqueux modélisant le couple résistant généré par la génératrice à courant continu.
- J : Moment d'inertie de l'ensemble (moteur, génératrice).
- α : rapport cyclique imposé au hacheur quatre quadrants.
- β : pseudo rapport cyclique $\beta = \alpha - 0.5$.
- τ_m : constante de temps de la partie mécanique $\tau_m = J/(f_0 + f_1)$.
- τ_e : constante de temps électrique $\tau_e = L_m/R_m$.

Algorithme de commande .

- T_e : période d'échantillonnage de la régulation de courant.
- ε : erreur entre le courant d'induit I_m et le courant de référence I_{ref}.
- $\int_0^k \varepsilon$: Intégrale de l'erreur ε entre 0 et $t = k.T_e$ secondes.
- $1/T_i$: Constante intégrale du régulateur PI.
- K_p : Constante proportionnelle du PI.

Cas du système multitâche .

- Te_1 : Période d'échantillonnage de la régulation de vitesse.
- Te_2 : Période d'échantillonnage de la régulation de courant.

$\widehat{(.)}$	:	Estimation de la grandeur (.).
$\mathbb{V}c_i$	:	Réprésentation numérique de la tension Vc_i.
$\hat{\mathbb{I}}_{\{t,f,harmo\}}$	:	Réprésentation numérique du courant estimé $\hat{I}_{\{t,f,harmo\}}$.
$K_i,\ K_v$	:	Facteurs d'échelle respectivement associés au courant et à la tension.
T_e	:	Période d'échantillonnage de l'emulateur et de l'observateur.
Z_t	:	Impédance totale en sortie du convertisseur.
Z_{ch}	:	Impédance de la charge.
Z_f	:	Impédance du filtre de rééquilibrage.
q^{-1}	:	Opérateur de retard d'une période d'échantillonnage.
p	:	Opérateur de Laplace.
2^l	:	coéfficient multiplicateur associé aux équations de l'émulateur.
2^{bf}	:	coéfficient multiplicateur associé aux équations du filtre permettant d'extraitre $\hat{I}_{harmo}$.

Introduction Générale

Les progrès dans l'électronique de puissance et dans la conception des machines électriques donnent naissance à des systèmes électriques de plus en plus complexes ayant parfois des propriétés particulières. Afin de contrôler de tels systèmes, de nouvelles lois de commande sont élaborées et leur implantation entraîne de nouvelles contraintes dans l'élaboration des dispositifs de commande.

Si la commande d'un système électrique requiert en premier lieu un module de régulation dont la synthèse est basée sur les outils de l'automatique, les progrès effectués dans ce domaine et dans celui du traitement du signal et de l'information permettent d'envisager la mise en place d'autres modules. En effet, on voit de plus en plus apparaître des fonctions de surveillance et de diagnostic permettant d'améliorer la disponibilité du matériel. L'estimation et l'observation sont aussi de plus en plus utilisées afin de reproduire des grandeurs inaccessibles ou de diminuer, pour des raisons de coût ou de fiabilité, le nombre de capteurs nécessaires à la commande de tels systèmes.

La conception d'un dispositif de commande doit donc tenir compte de ces différentes contraintes associées à chaque module mis en place. Ces modules peuvent comporter des calculs complexes, mais aussi des contraintes temporelles et/ou des contraintes de précision importantes. Ces constatations montrent que la conception des dispositifs de commande ne peut plus suivre une règle unique, mais doit être adaptée à chaque système.

L'utilisation de l'électronique numérique semble alors inévitable. Elle n'est d'ailleurs plus à remettre en cause, et elle correspond au premier outil du concepteur pour l'élaboration de ces dispositifs.

On peut diviser les composants de l'électronique numérique en deux grandes classes. La première correspond au microprocesseurs et à ses variantes dédiées à des applications spécifiques (comme les DSP pour le traitement du signal). La deuxième classe de composants correspond quant à elle aux ASIC's et aux Composants Logiques Programmables (CLP).

Il y a encore peu de temps, le choix d'une architecture de commande était imposé par des limitations associées aux composants utilisés. En effet, le microprocesseur était considéré comme relativement lent mais pouvant effectuer des calculs complexes et les CLP, comme extrêmement rapides, mais ne pouvant exécuter que des opérations simples. Or, la limite entre l'utilisation d'un microprocesseur ou d'un CLP s'avère, avec les progrès de l'électronique numérique, de plus en plus ténue. En effet, les microprocesseurs bénéficient de progrès toujours plus importants concernant l'intégration, ce qui permet d'augmenter leurs fréquences de fonctionnement et les rend de plus en plus rapides. Les CLP sont aussi en constante évolution et permettent aujourd'hui de réaliser des opérations de plus en plus complexes.

Déterminer une architecture optimale, c'est-à-dire répondant aux différentes contraintes temporelles et fonctionnelles du système, tout en restant la plus simple et la moins chère possible, est devenue extrêmement difficile.

De plus, si les composants sont en constante évolution, les outils et méthodes de conception associés ne suivent pas la même progression, et des besoins nouveaux appa-

raissent dans l'aide à la conception des systèmes numériques.

Ces besoins existent essentiellement lorsque l'on veut construire un système mixte, composé de microprocesseurs et de composants programmables de d'ASIC, dit aussi "système mixte logiciel/matériel". Des travaux de recherche sont effectués afin de développer des méthodes de conception adaptées à de tels systèmes. Ces méthodes relèvent de ce que l'on nomme le "Co-design". L'objectif du Co-design est de définir des outils et des méthodes permettant de trouver la meilleure adéquation entre la réalisation logicielle et la réalisation matérielle, et d'accélérer le développement en réalisant la conception du logiciel et du matériel en parallèle (Concurrent Design).

Étant donné la complexité des systèmes commandes, il paraît intéressant d'étudier dans quelles mesures le co-design peut répondre aux problèmes spécifiques de la commande dédiée aux systèmes électriques. Les travaux présentés dans ce mémoire constituent une première étape dans cette étude, et se veulent prospectifs. Nous avons cherché à développer des outils typiques du co-design, puis à les éprouver en les utilisant sur des applications particulières. Ces travaux sont présentés en trois parties.

La première partie du mémoire présente les différentes composantes fonctionnelles pouvant être intégrées dans un dispositif de commande, afin de montrer les contraintes qui peuvent apparaître lors de sa conception. Nous présentons ensuite les différentes solutions technologiques qui peuvent être exploitées dans un système de commande

Dans un deuxième chapitre, les grands principes associés à une méthodologie de co-design, ainsi que les outils qui peuvent être développés pour appliquer une telle méthodologie, sont présentés. Parmi ces outils, nous mettons en avant la co-simulation et les avantages qu'elle peut apporter dans la conception des dispositifs de commande dédiés aux systèmes électriques et également son intérêt dans l'intégration de lois de commande.

Dans la deuxième partie, nous cherchons à mettre en place les différents éléments permettant de créer un environnement de co-simulation. Le premier chapitre de cette partie est essentiellement basé sur la recherche de modèles de simulation pour les microprocesseurs. Il présente, notamment, un exemple d'application utilisant un simulateur de jeu d'instructions. Dans un second temps, nous tentons de montrer l'intérêt d'une technique particulière nommée simulation compilée. Le deuxième chapitre introduit la simulation des composants logiques programmables à l'aide du langage VHDL. Il détaille le cycle de conception utilisant le VHDL et présente un outil permettant le couplage d'un simulateur VHDL avec un logiciel simulant le processus à commander (SABER).

Un exemple, portant sur la commande d'un convertisseur multicellulaire série en mode hacheur, est alors utilisé pour valider cet environnement. Ce type de convertisseur est présenté plus en détail au début de la troisième partie.

La troisième partie porte sur l'étude d'architectures matérielles pour l'implantation d'observateurs dédiés aux convertisseurs multicellulaires séries. Elle présente, dans un premier temps, des solutions d'implantation pour un observateur original appelé reconstructeur d'état permettant d'estimer les tensions flottantes avec un nombre limité de capteurs. Une de ces solutions est testée et validée en co-simulation. Dans un second temps nous montrons l'intérêt de la co-simulation dans la conception d'un émulateur du convertisseur multicellulaire, émulateur qui peut ensuite être utilisé pour de l'observation et du diagnostic.

Enfin, nous présentons les conclusions concernant cette étude prospective et nous donnons les directions de recherche possibles dans l'étude du co-design dédié à la conception des dispositifs de commande pour les systèmes électriques.

Première partie

La commande numérique et le Co-design

Chapitre 1

Architecture de dispositifs de commande numérique pour les systèmes électriques

1.1 Introduction.

L'utilisation des technologies liées à l'électronique numérique dans la commande des systèmes électriques est devenue, aujourd'hui, incontournable. Elles apportent, en particulier, une très grande flexibilité grâce à des composants programmables qui bénéficient chaque jour des progrès de la micro-électronique. Ces derniers permettent de réaliser des fonctions complexes (commande non-linéaire, commutation d'algorithme, capteurs indirects,...) avec une facilité accrue, comparée aux possibilités offertes par l'électronique analogique. De plus, dans le cas de contraintes temporelles sévères, l'utilisation de l'électronique analogique n'est plus, a priori, la solution majeure. En effet, l'association des composants actuels, dont les performances fréquentielles sont en constante augmentation, et des techniques de l'automatique permet de pallier à cette contrainte.

Nous traiterons, dans ce mémoire, de la commande numérique située au plus proche du processus à contrôler. La fonction principale d'une telle commande correspond à la régulation et à l'asservissement de grandeurs associées au processus, comme par exemple : la vitesse d'une machine électrique, ou bien la tension de sortie d'un onduleur. L'utilisateur n'ayant pas d'action directe sur le contrôle du processus, le dispositif de commande doit avoir une capacité à réagir par rapport à des perturbations et des défaillances du processus. Le dispositif de commande est un système qui doit donc faire preuve d'une certaine autonomie. Ces contraintes sont du même ordre que celles d'un système embarqué.

Un dispositif de commande est défini par une structure fonctionnelle se basant sur les outils de l'automatique et du traitement du signal. Cette structure est ensuite implantée en utilisant les ressources disponibles dans le domaine de l'électronique numérique.

Afin d'effectuer une synthèse de ce qui doit être contenu dans un tel dispositif, nous allons diviser ce chapitre en deux parties distinctes. Nous présenterons d'abord l'architecture comportementale d'un dispositif de commande afin de déterminer quelles fonctionnalités doit contenir le dispositif numérique (fonctions de calculs arithmétiques, de mémorisation, ...). Puis nous verrons les différents composants qui peuvent constituer l'architecture matérielle d'un tel dispositif.

1.2 L'architecture fonctionnelle de la commande et ses contraintes.

1.2.1 La commande et son environnement.

Un système qui requiert un traitement de l'énergie électrique utilise un ou plusieurs convertisseurs statiques auxquels sont associés une commande. C'est par l'intermédiaire de cette commande qu'un convertisseur va pouvoir réaliser au mieux sa fonction. Le dispositif de commande doit alors pouvoir communiquer avec son environnement pour connaître l'état du processus et modifier cet état à l'aide du convertisseur.

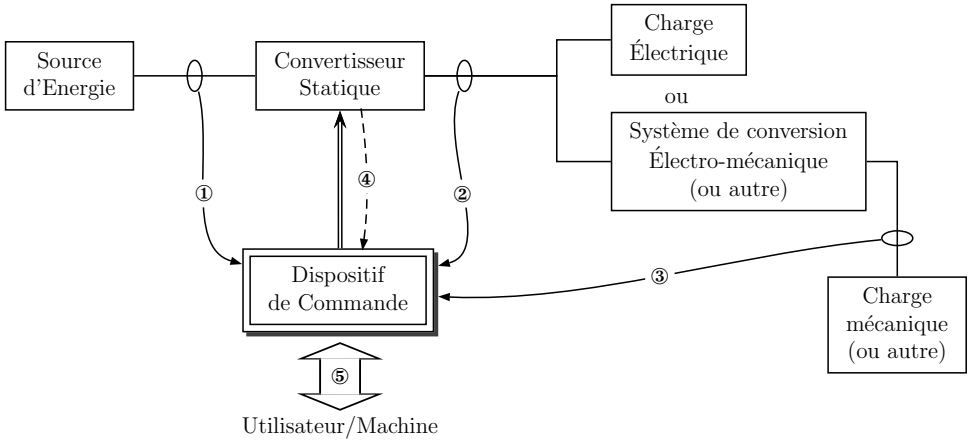


Figure I.1.1: *La commande et son environnement.*

La figure I.1.1 représente la place du dispositif de commande dans un système électrique. Elle fait notamment apparaître les différents signaux qui peuvent interagir avec la commande. On peut distinguer les signaux qui sont reçus par la commande et ceux qui sont émis.

Les signaux reçus sont utilisés pour la mesure de grandeurs nécessaires au contrôle du processus. Parmi ces signaux, des grandeurs électriques en entrée et en sortie du convertisseur sont souvent mesurées (liens ① et ②). Le lien ③ représente la mesure de grandeurs physiques de nature différente comme une température ou une vitesse mécanique.

Le lien ④ représente la mesure de grandeurs internes au convertisseur. Ces mesures peuvent s'avérer nécessaires pour le bon fonctionnement du convertisseur et indirectement pour le contrôle de la charge. C'est le cas, par exemple, des convertisseurs multicellulaires (mesures de tensions des capacités flottantes) et des convertisseurs à résonance (mesure des grandeurs du filtre résonant).

Les signaux émis correspondent aux ordres de commande d'ouverture et de fermeture des interrupteurs constituants le convertisseur.

Le lien ⑤ représente le canal de communication entre le dispositif de commande et un utilisateur ou une machine. Ce lien permet de transmettre au dispositif de commande les consignes de régulation mais aussi de récupérer différentes informations sur le système.

En considérant ainsi l'environnement, on peut noter des caractéristiques communes à tout dispositif de commande numérique.

Les signaux émis et reçus par le système de commande sont de plusieurs types. Le dispositif de commande doit pouvoir mesurer des grandeurs analogiques et/ou numériques issues des capteurs. Il doit avoir une liaison numérique pour la communication

machine/machine ou homme/machine, et doit pouvoir générer et mesurer des grandeurs logiques pour la génération des ordres de commandes et la mesure de grandeurs événementielles.

De plus, un dispositif numérique est de part sa nature un système discontinu. Il ne réagit avec son environnement extérieur (processus à commander et utilisateur) qu'à des instants discrets. Ces instants sont soumis à des contraintes temporelles qui peuvent varier de la seconde à la microseconde selon la dynamique des grandeurs à réguler.

L'observation des échanges entre le système de commande et son environnement permet d'établir les premières caractéristiques d'un dispositif de commande numérique. Les autres contraintes sont liées aux différentes tâches que celui-ci est amené à réaliser.

1.2.2 Structure fonctionnelle d'un système de commande.

Dans le cas général, on peut découper, du point de vue fonctionnel, un système de commande en sept fonctions principales (cf. figure I.1.2).

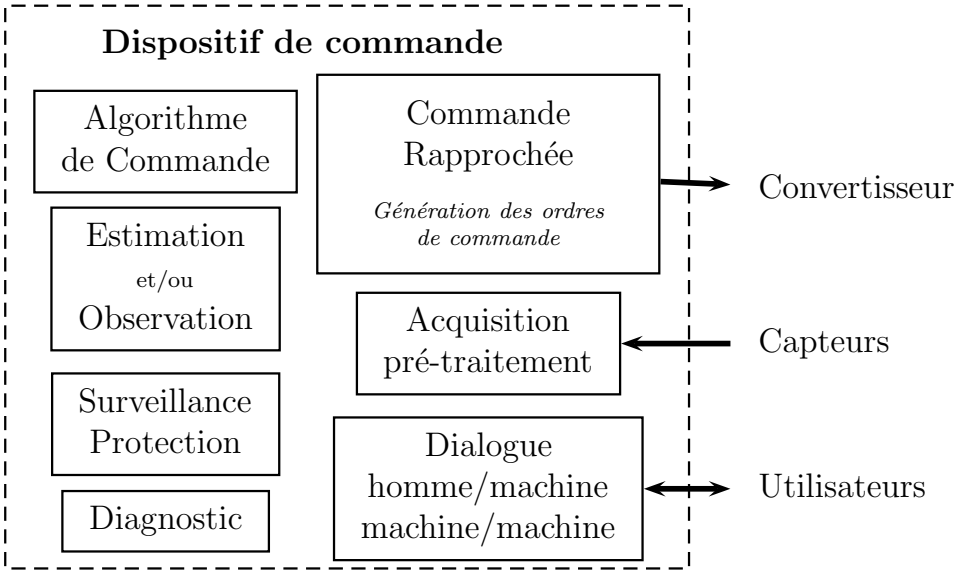


Figure I.1.2: *Les groupes fonctionnels embarqués dans un dispositif de commande.*

1.2.2.1 Algorithme de commande.

L'algorithme de commande est le centre "névralgique" du dispositif de commande. C'est le module où se calculent les différentes lois de commande en fonction des grandeurs de consigne et des grandeurs mesurées ou estimées/observées.

Différents outils mathématiques peuvent être utilisés pour réaliser le contrôle. On peut citer l'automatique linéaire, les réseaux de neurones, la logique floue, l'optimisation, ...

Toutes ces méthodes peuvent faire appel à des fonctions arithmétiques (comme l'addition, la multiplication, etc), mais aussi à des fonctions transcendantales, des fonctions d'intégration et de dérivation.

Les différentes régulations qui peuvent être utilisées sont divisibles en deux catégories du point de vue temporel :

le contrôle à échantillonnage fixe : Il s'agit de régulations qui renvoient des grandeurs de commande de manière régulière dans le temps. Comme nous l'avons

vu dans la section précédente, les grandeurs à réguler peuvent avoir des dynamiques très différentes et les régulations qui en découlent ont alors des périodes d'échantillonnages adaptées.

le contrôle réactif : Il s'agit d'une commande qui réagit selon des évènements particuliers comme par exemple le dépassement d'une grandeur par rapport à un seuil (cas de la fourchette de courant). Le cas du contrôle réactif ne permettant pas de séparer la commande rapprochée de la commande algorithmique, il en résulte des contraintes temporelles extrêmement fortes.

À l'heure actuelle, dans le cadre de la commande numérique, les régulations à échantillonnage fixe sont le plus souvent utilisées et les algorithmes de contrôle réactif nécessitent une adaptation afin d'être échantillonnés.

Les ordres de commandes obtenus par le contrôle algorithmique sont ensuite envoyés au module de commande rapprochée.

1.2.2.2 Estimation et observation.

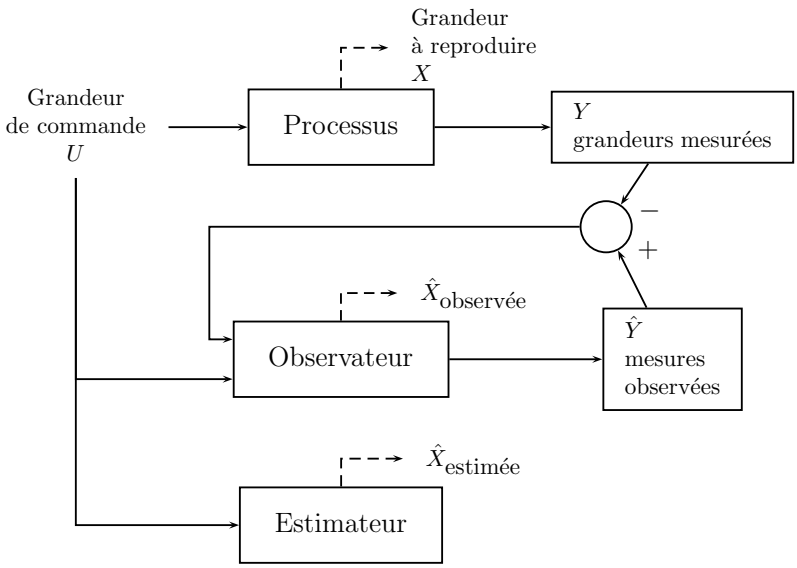


Figure I.1.3: *Schéma de principe représentant la différence entre l'observateur et l'estimateur.*

Pour des raisons de coût ou des raisons technologiques, il est parfois trop contraignant de mesurer certaines grandeurs du système. Cependant ces grandeurs peuvent représenter une information capitale pour la commande ou la surveillance. Il est alors nécessaire de reconstruire l'évolution de ces variables qui ne sont pas issues directement des capteurs. Il faut donc réaliser un *capteur indirect*. Pour cela, on utilise des estimateurs ou, selon le cas, des observateurs.

Un estimateur permet de reconstruire la grandeur recherchée en calculant en temps réel l'évolution d'un modèle du processus commandé.

Dans le cas de l'observateur, on compare l'évolution du modèle et du système réel en mesurant l'erreur sur des grandeurs que l'on peut directement capter. Cette erreur est alors utilisée pour faire converger le modèle vers le système réel. Il s'agit un système bouclé au contraire de l'estimateur (cf. figure I.1.3).

L'estimation/observation, qui est un module essentiel, demande souvent des calculs assez complexes avec des contraintes temporelles identiques à celles de la régulation.

1.2.2.3 Commande rapprochée (Génération des ordres de commande).

La commande rapprochée est en liaison directe avec le convertisseur. Elle est, du point de vue temporel, un des modules les plus critiques. La précision nécessaire doit être de l'ordre de la centaine de nanosecondes. Une erreur dans la génération des ordres de commande peut perturber le fonctionnement du système, voir endommager le convertisseur ou sa charge.

Les fonctions utilisées dans une commande rapprochée sont en général une association des fonctions de base de la logique séquentielle et combinatoire. Un exemple classique de commande rapprochée est la génération d'une Modulation de Largeur d'Impulsion (MLI).

1.2.2.4 Acquisition et pré-traitement.

La mesure issue d'un capteur peut être une grandeur continue, numérique ou binaire. La mesure n'est quasiment jamais exploitable par la commande algorithmique de manière directe. Il faut alors effectuer un pré-traitement sur ces grandeurs. Ce pré-traitement peut correspondre à un filtrage, une mise à l'échelle, ou parfois un calcul plus complexe comme par exemple, la détermination d'une fréquence en utilisant le signal d'un codeur incrémental (mesure de vitesse).

Ces grandeurs sont échantillonnées selon des cadences fixées par les contraintes temporelles des modules utilisant ces mesures. Cette cadence peut varier de quelques millisecondes à la centaine de nanosecondes.

1.2.2.5 Surveillance et diagnostic.

Le dispositif de commande étant un système embarqué, il se doit de pouvoir réagir de manière autonome en cas de dysfonctionnement du processus afin de ne pas induire de dégâts irrémédiables sur celui-ci.

La détection des défauts et l'adaptation des algorithmes de commande à ces défauts correspond aux fonctions principales de la surveillance et du diagnostic. Ce sont des tâches qui tendent à prendre de plus en plus d'importance et qu'il devient nécessaire d'intégrer dans les dispositifs de commande.

La surveillance

Il s'agit d'un module dont le temps de réaction est du même ordre de grandeur que l'algorithme de commande, ou la commande rapprochée. Il doit surveiller le bon comportement du processus commandé et détecter une anomalie "brutale". Son but est alors de déclencher, en fonction de la gravité du problème, une commutation de l'algorithme de commande pour une marche dégradée ou bien l'arrêt du système. La machine d'état est un exemple classique de module de surveillance, les transitions sont alors associées à des détections d'anomalies comme par exemple un dépassement de seuil pour une grandeur spécifique. Cependant lorsque la grandeur surveillée n'est pas directement accessible par la mesure, l'utilisation de l'estimation et de l'observation s'avère nécessaire.

Le Diagnostic

Il correspond généralement à l'analyse de données qui permet de détecter une possible dégradation du processus. Certains diagnostics peuvent être faits en temps réel

afin que l'algorithme de commande puisse s'adapter et diminuer son impact sur la dégradation.

1.2.2.6 Dialogue homme/machine ou machine/machine.

Ce module permet de faire le lien entre le dispositif de commande et l'extérieur. Ce lien peut permettre de configurer les modules que nous venons de voir dans le cas où ceux-ci sont paramétrés. Il permet notamment de fixer les grandeurs de consigne pour les régulations. Il peut aussi être utilisé afin de récupérer un certain nombre d'informations utiles pour le diagnostic hors-ligne ou bien pour une commande de plus haut niveau (commande système).

Il s'agit donc d'un module qui doit gérer les flux d'informations entrant et sortant du dispositif de commande. Selon l'environnement du dispositif, celui-ci peut être relié à un réseau ou une machine de type PC. Il peut donc y avoir différents types de protocole de communication (TCP/IP, bus de terrain, ...).

1.2.3 Contraintes fonctionnelles résultantes.

Les fonctions définies précédemment, de part leurs actions, ont des priorités respectives et des contraintes temporelles différentes. La figure I.1.4 représente les contraintes temporelles associées à chaque module que l'on retrouve le plus souvent dans un dispositif de commande.

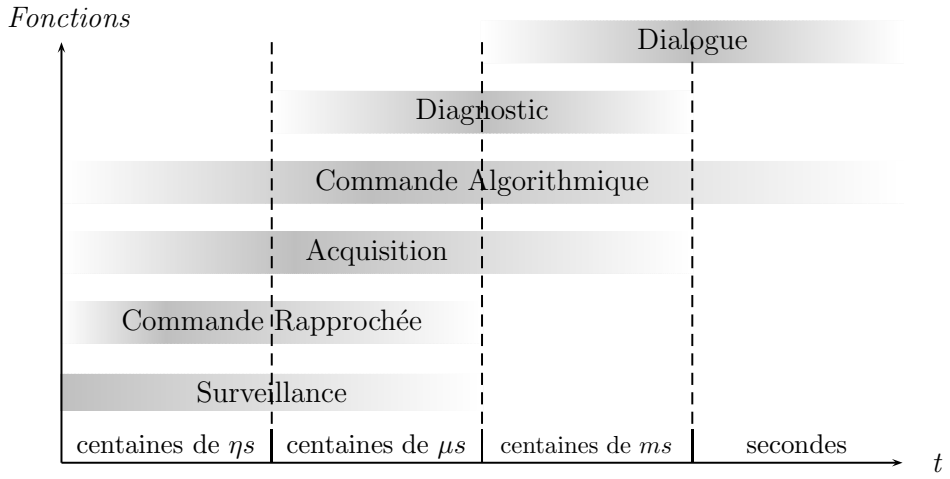


Figure I.1.4: *Contraintes temporelles associées aux fonctions du système de commande.*

La réalisation de ces différentes fonctions nécessite la mise en place d'un véritable dispositif de commande multi-tâches qui doit répondre à un certains nombre de propriétés que l'on peut résumer ainsi :

une capacité d'ordonnancement : le dispositif ayant plusieurs tâches à effectuer, celui-ci doit alors pouvoir les gérer en fonction de la priorité des différentes actions qu'elles réalisent.

une capacité de calculs arithmétiques rapides : que ce soit la commande, l'estimation/observation ou encore le diagnostic, toutes ces tâches nécessitent des calculs

arithmétiques importants avec des précisions relatives et ce, dans des intervalles de temps parfois très courts.

une capacité de traitement en logique combinatoire et séquentielle : celle-ci est utilisée pour la génération des ordres de commande et par le module de surveillance notamment pour détecter les dépassements de seuils (comparateurs). Le calcul logique permet aussi de générer des signaux périodiques pour activer les acquisitions et certaines tâches échantillonnées.

une capacité de mémorisation : cette mémorisation est nécessaire dans le cadre du diagnostic, mais aussi dans certaines commandes utilisant des fonctions temporelles comme par exemple le calcul de spectres fréquentiels.

1.3 Architecture matérielle d'un dispositif de commande numérique.

1.3.1 Fonctions numériques de l'architecture.

Nous avons vu précédemment les différentes actions qu'un système de commande est susceptible d'accomplir et les propriétés que doit détenir le dispositif dans ces conditions. Ces propriétés peuvent être réunies dans un système tel que celui présenté sur la figure I.1.5.

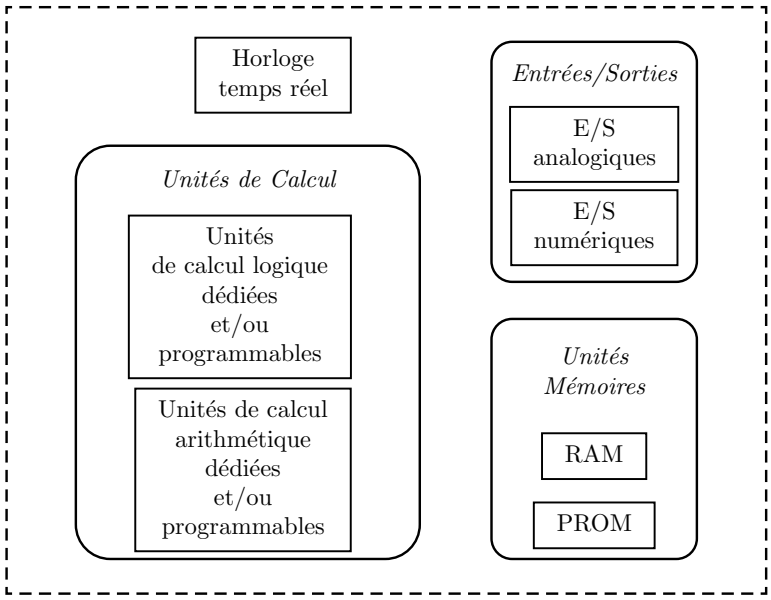


Figure I.1.5: *Composants nécessaire dans un dispositif de commande numérique.*

Le module d'Entrées/Sorties (E/S) numériques est difficilement définissable a priori car il dépend essentiellement de la nature des informations échangées et des protocoles de communication utilisés.

Le module d'Entrées/Sorties analogiques reçoit les signaux issus des capteurs et effectue une adaptation, voire un pré-traitement de ces signaux afin de les transmettre aux convertisseurs analogiques numériques (CAN). Le nombre de convertisseurs ainsi que leurs caractéristiques sont complètement dépendants des traitements numériques que vont subir les mesures.

La mémoire PROM permet la mémorisation de programmes et de données nécessaires aux unités de calculs programmables. La mémoire RAM permet quant à elle de mémoriser les données variables dans le temps utilisées lors du fonctionnement du système.

Les paramètres principaux utilisés pour le choix des unités mémoires sont les temps d'accès et la capacité de stockage.

Les unités de calcul peuvent être différenciées en deux catégories : les unités programmables et les unités dédiées. Les unités de calcul programmables apportent une très grande flexibilité et prennent peu de place en terme de surface de silicium. Elles sont par contre assez lentes car elles nécessitent des échanges permanents avec l'unité mémoire. Les unités de calcul dédiées sont, elles, beaucoup plus rapides, car les actions qu'elles doivent réaliser définissent leurs structures. Elles sont cependant moins flexibles et sont gourmandes en silicium.

Une horloge temps réel associée à une unité de logique séquentielle dédiée permet de générer des horloges avec des phases et des fréquences sous-multiples. Elles sont utilisées pour activer les différentes actions du dispositif de commande comme les acquisitions, les régulations, etc.

1.3.2 Réalisation : différentes solutions technologiques.

Pour réaliser les différentes fonctions numériques que nous venons de décrire, il existe trois grandes technologies possibles (cf. I.1.6).

- Les Composants standards (microprocesseurs et logique TTL ou CMOS),
- Les Composants Logiques Programmables,
- Les ASIC's,

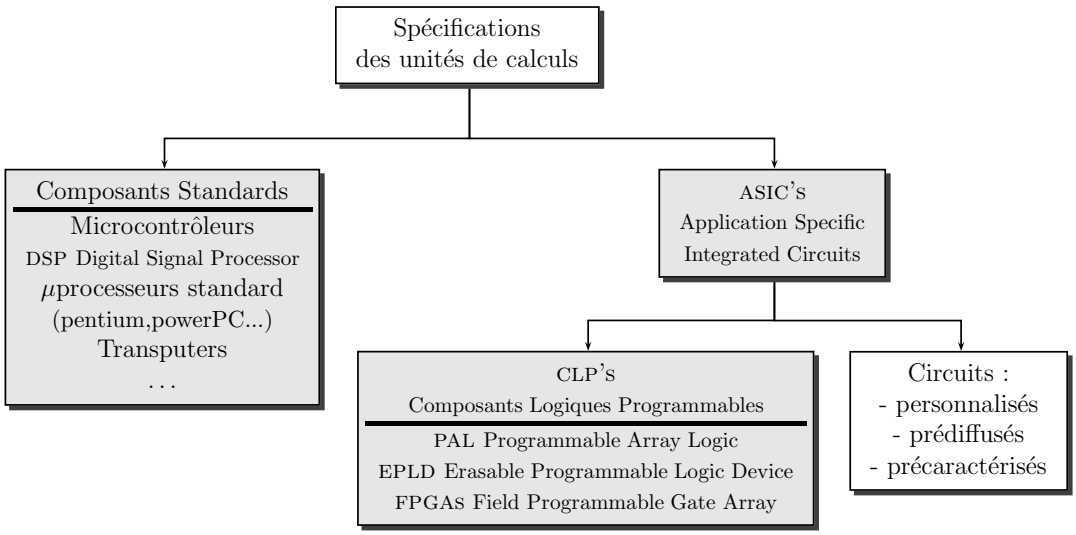


Figure I.1.6: *Technologies exploitables dans une unité de calcul.*

Les ASIC's qui ne font pas partie des CLP (cf. figure I.1.6) sont des circuits intégrés dédiés à une application particulière. On peut les répartir en plusieurs sous-ensembles parmi lesquels on a : les circuits personnalisés, les circuits pré-caractérisés, et les circuits pré-diffusés (mer de portes). Ils ont un niveau d'intégration très élevé et de très bonnes performances. Cependant, leur conception, de par les technologies employées, est coûteuse et nécessite un temps de développement relativement long. De plus, un tel composant est "figé". Il peut difficilement prendre en compte une évolution ou une mise à jour de l'algorithme. Enfin, leur utilisation est industriellement viable seulement si le volume de production est très important.

Les différents inconvénients cités ci-dessus, nous conduisent à considérer ce type de technologie comme inadéquat dans le cadre de la recherche d'une architecture optimale.

Les composants standards de type microprocesseur et les Composants Logiques Programmable semblent eux beaucoup plus avantageux. Afin de déterminer les critères de performance associés à ces technologies, nous allons décrire les principes de base qu'ils utilisent.

1.3.3 Propriétés des Microprocesseurs.

Le microprocesseur répond très souvent à un grand nombre de propriétés que nous recherchons dans un dispositif numérique. Il en est donc, en général, le composant central.

1.3.3.1 Structure générale d'un microprocesseur.

La fonction principale d'un microprocesseur est d'exécuter une suite d'instructions constituant un programme.

L'instruction contient essentiellement deux informations. Elle indique quelle opération doit être effectuée et sur quelles données (ou opérandes).

La base du fonctionnement d'un microprocesseur repose alors sur trois unités fonctionnelles.

la mémoire : elle permet de stocker le programme et les données à traiter sous forme numérique.

l'unité de traitement : elle exécute les instructions. Ces instructions correspondent à des calculs arithmétiques et logiques. Cette unité travaille sur des entités mémoire intégrées qui lui sont propres : *les registres.*

l'unité de commande : son rôle est de rechercher les instructions situées en mémoire puis de les décoder afin d'indiquer à l'unité de traitement la nature de l'opération à effectuer. Si l'opération nécessite des opérandes particuliers, l'unité de commande doit charger ceux-ci dans les registres associés à l'unité de traitement.

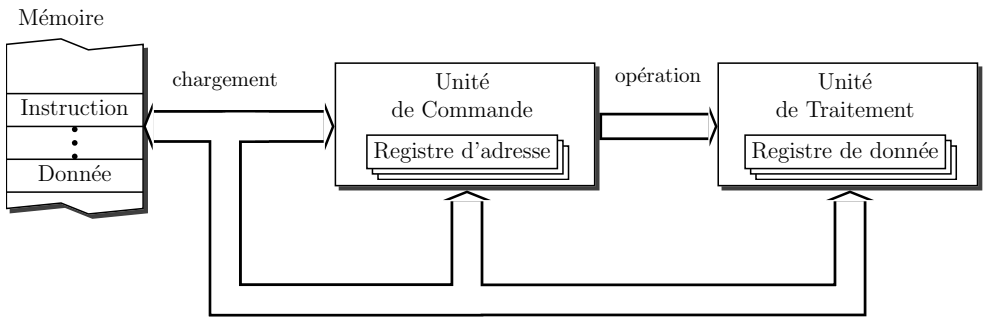


Figure I.1.7: *Les unités de base constituant un coeur de microprocesseur.*

L'exécution d'un programme par un microprocesseur est donc constituée de nombreuses étapes. Il en découle un temps de calcul rarement négligeable notamment dans le cadre de la commande de processus électriques. C'est d'ailleurs le critère le plus important dans le choix d'un microprocesseur.

Ce temps de calcul dépend d'un nombre de facteurs importants qui peuvent être résumés dans l'équation suivante [17] :

$$\frac{temps}{tâches} = \frac{instructions}{tâches} \times \frac{cycles}{instructions} \times \frac{temps}{cycle} \qquad \text{(I.1.1)}$$

Chacun des termes du produit a une valeur qui dépend de la qualité et de l'adéquation des différents composants rentrant en jeu pour l'exécution d'un programme sur un microprocesseur : le programme, le compilateur, et bien évidemment l'architecture du microprocesseur et les technologies utilisées.

- **Le facteur** $\frac{instructions}{tâches}$

  Ce rapport dépend du programme décrivant la tâche à exécuter c.-à-d. qu'il dépend de l'algorithme. En informatique, on peut obtenir un même résultat par le biais d'algorithmes différents. Cependant, ceux-ci ne seront pas équivalents en terme de nombres d'instructions utilisées. La qualité du compilateur rentre aussi en jeu puisqu'il doit traduire le programme, écrit généralement dans un langage haut niveau, en un flot d'instructions machines. Cette traduction peut s'effectuer selon différents critères comme la place mémoire, ou le temps d'exécution.

- **Le facteur** $\frac{cycles}{instructions}$

  Il est lié à deux éléments que sont le compilateur et l'architecture.

  Le compilateur doit être en adéquation avec l'architecture. Il doit pouvoir adapter le séquencement des instructions de manière à diminuer le nombre de cycles utilisés par instruction.

  L'architecture du processeur elle-même joue un rôle. On peut notamment prendre le cas des architectures CISC et RISC. Une architecture CISC est un processeur qui contient un grand nombre d'instructions dans son langage machine. Cependant chacune des ces instructions est complexe et demande un nombre de cycles important. Une architecture RISC contient par contre un nombre d'instructions faible mais très rapide.

- **Le facteur** $\frac{temps}{cycles}$

  Ce facteur est associé à l'architecture du processeur et aux éléments technologiques qui le composent. Il s'agit en particulier de l'horloge de base du microprocesseur.

L'architecture interne d'un microprocesseur a largement évoluée et différentes techniques matérielles ont vu le jour pour diminuer les facteurs de l'équation I.1.1. Ces techniques ont donné naissance à des architectures de microprocesseurs adaptées à certains types de programmes. Nous allons voir quelles sont les techniques les plus couramment utilisées. Cette connaissance peut permettre de diriger le choix sur un microprocesseur plutôt que sur un autre au regard des tâches à exécuter.

1.3.3.2 Techniques matérielles d'accélérations du traitement d'un programme.

Les co-processeurs

Pour illustrer ceci, nous pouvons citer les Unités de Calcul flottant. En vue d'accélérer les calculs de données formatées en flottant, on associe souvent une unité de calcul adaptée. L'unité permet d'accélérer le traitement de ces données. En son absence, le calcul se fait par une émulation logicielle qui consomme énormément de temps.

Le jeu d'instructions du microprocesseur est alors augmenté des opérations que peut effectuer le coprocesseur. Cette technique est intéressante lorsque le code qui doit être exécuté contient des opérations complexes récurrentes.

Architecture Harvard

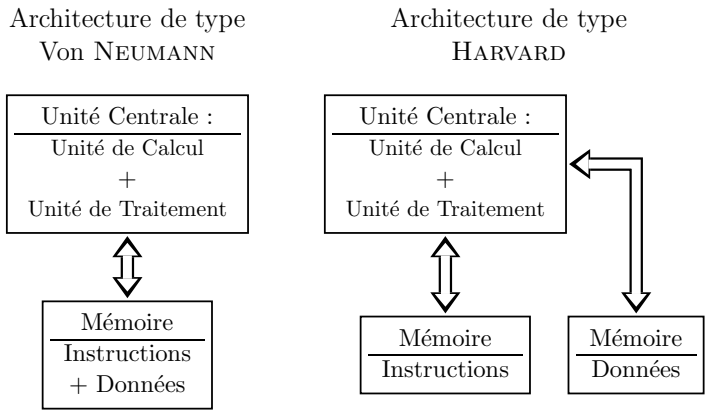


Figure I.1.8: *Comparaison entre l'architecture de Von* Neumann *et* Harvard.

Dans les premières architectures qui sont apparues, il n'y avait qu'un seul bus reliant le processeur à la mémoire. Il s'agit des architectures de type Von Neumann. Le microprocesseur doit lire l'instruction, charger les opérandes, exécuter le calcul, puis écrire le résultat en mémoire de manière totalement séquentielle. Depuis, sont apparues les architectures Harvard où deux mémoires différentes interviennent : une mémoire qui stocke les données, et une mémoire qui stocke les programmes (ensemble des instructions). Ainsi deux bus différents relient ces mémoires à l'unité centrale. Deux actions différentes peuvent être exécutées en parallèle comme par exemple : charger une instruction de la mémoire de programmes dans l'unité de commande et écrire en même temps le résultat de l'instruction précédente dans la mémoire de données.

Ce type d'architecture permet donc d'accélérer le traitement d'un programme. Cette architecture est souvent exploitée en utilisant un *pipeline*.

Principe du pipeline

Le principe du pipeline est présenté dans le schéma I.1.9. Les différentes étapes qui constituent l'exécution d'une instruction sont séquentielles et peuvent être réalisées par des unités indépendantes. On peut généralement les découper en quatre ou cinq phases :

1. le chargement de l'instruction dans l'unité de commande,
2. le décodage de l'instruction,
3. le chargement des opérandes,
4. l'exécution de l'instruction par l'unité de calcul,
5. l'écriture du résultat en mémoire.

Ainsi, comme il est indiqué sur la figure I.1.9, durant le chargement de l'instruction 3, l'instruction 2 est décodée et les opérandes de l'instruction 1 sont chargées. lorsque le pipeline est "plein", es instructions sont exécutées en 1 cycle au lieu de 4 ou 5. Cependant certaines instructions peuvent rompre la chaîne de traitement. Ce cas est généralement dû à une dépendance des données entre les instructions qui se suivent ou à des branchements conditionnels. Certains algorithmes profitent très peu de ce type d'architecture.

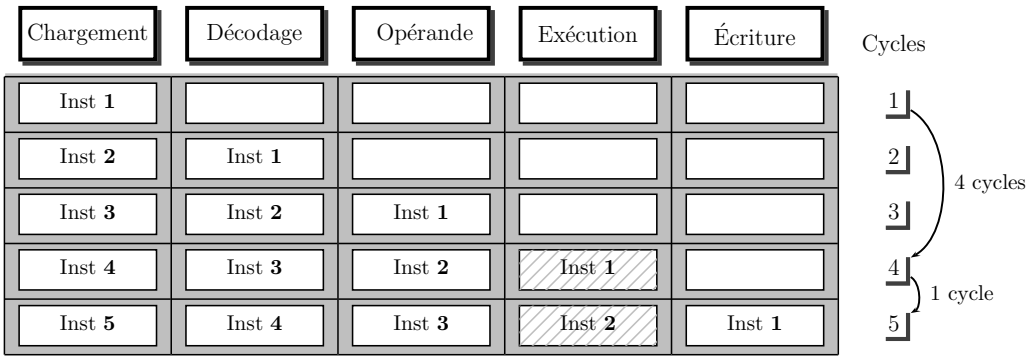


Figure I.1.9: *Principe du pipeline.*

Architecture VLIW[1]

Cette architecture est utilisée pour des instructions dont le format est supérieur au format habituel de 16 ou 32 bits (voire 64). Il s'agit d'instructions codées sur 128 ou 256 bits. Ce format d'instruction permet de charger en un même cycle de lecture plusieurs instructions codées au format de 32 bits. Sous la condition où il n'y ait pas de dépendance entre les instructions qui forment le mot de 128 ou 256 bits, on peut traiter plusieurs calculs en parallèle. Ceci implique alors une redondance des unités de calcul. Une telle architecture implique aussi que le compilateur ait pu analyser de manière précise la dépendance des données, afin de constituer les instructions longues.

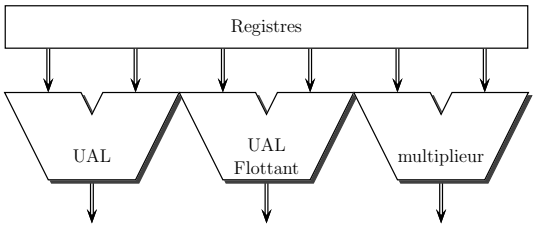


Figure I.1.10: *Représentation d'une architecture VLIW.*

1.3.3.3 Cas particulier des processeurs spécifiques de type DSP

Les DSP (Digital Signal Processing) sont des microprocesseurs dédiés au traitement du signal et destinés aux applications nécessitant de nombreux calculs (transformée de Fourier, produits de convolutions, ...). La particularité de ces puces est de regrouper un certain nombre d'éléments architecturaux qui les rend adaptées aux calculs intensifs. Ces éléments sont en général les suivants :

- une architecture Harvard, séparant bus programme et bus données,
- des instructions adaptées aux opérations classiques de traitement de signaux : multiplication & accumulation en un seul cycle d'instruction (MAC), transferts de données par DMA (Direct Access Memory) pour la plupart,
- un adressage circulaire permettant de ne pas faire des tests de modulo sur les pointeurs de tampons,
- un mode d'adressage par inversion de bits servant à réorganiser les échantillons de sortie de transformées de Fourier Rapides (FFT),

[1] Very Large Instruction Word

- des banques de mémoire, permettant de scinder par exemple partie réelle et partie imaginaire,
- une architecture à pipeline permettant de paralléliser le maximum d'opérations élémentaires (préchargement d'instruction, chargement des données, exécution d'opérations arithmétiques/flottantes, stockage des résultats)

Ces différents atouts ne se retrouvent pas sur tous les DSP. Cependant, ces exemples caractérisent les spécificités architecturales qui peuvent être faites sur ce type de microprocesseur.

Nous présentons ces microprocesseurs car ce sont actuellement les architectures les plus prisées pour l'implantation de commandes de systèmes électriques. Cependant, ces processeurs sont fortement adapté à des calculs liés aux traitement du signal. Ils sont donc conçus pour traiter des flots de calculs, et les tests de conditions et les branchements sont trés pénalisants puisqu'ils tendent à vider le pipeline ; ce qui ralentit le fonctionnement et peut faire perdre l'intérêt de l'utilisation de tels processeurs.

On voit cependant des DSP avec des architectures de plus en plus complexes qui font apparaître une forte potentialité dans le calcul parallèle puisqu'ils intègrent des instructions longues (architecture VLIW).

Lorsque l'on fait le bilan des différentes architectures de microprocesseurs, il est difficile de dire a priori celle qui sera la plus adaptée pour un dispositif de commande. En effet les performances ne dépendent pas que du microprocesseur mais aussi du compilateur et de leur adéquation avec l'algorithme. De plus, comme nous venons de le voir, les techniques d'accélération ne sont efficaces que sous certaines conditions.

Une architecture de microprocesseur peut alors être plus adaptée pour une tâche que pour une autre...

En conséquence les microprocesseurs ne peuvent pas, sous des contraintes temporelles trop fortes, répondre à tous les besoins fonctionnels.

Le cas de la MLI est significatif : cette fonction ne peut être exécutée de manière logicielle étant donnée la précision temporelle recherchée (la centaine de nanoseconde). Il apparaît alors nécessaire de constituer, dans certains cas, des unités de calculs logiques, et même parfois arithmétiques, dédiées à une tâche bien spécifique. Ceci est rendu possible par l'utilisation des composants logiques programmables. Il en existe un grand nombre de variétés. Nous allons donc dans un premier temps décrire les différentes technologies et architectures des composants logiques programmables existantes afin d'analyser leur potentiel.

1.3.4 Propriétés des composants logiques programmable.

Un composant logique programmable désigne un circuit intégré qui, en sortie d'usine, n'a pas de fonctionnalités figées. Pour lui en donner, il faut le programmer. Cette programmation peut se faire à l'aide d'une machine spécialement conçue à cet effet, ou bien à partir d'un PC. Certains composants ne peuvent se programmer qu'avant leur utilisation, d'autre peuvent se programmer ou se reprogrammer "sur site".

Parmi les composants logiques programmables, on peut différencier deux grandes classes :

- les mémoires mortes programmables,
- les composants considérés comme *"calculateurs logiques programmables"*

Dans les composants considérés comme calculateurs logiques, on distingue en terme de capacités fonctionnelles trois catégories : les PAL, les EPLD et les FPGA.

Afin de situer ces différents composants les uns par rapport aux autres en terme de capacités fonctionnelles, nous allons en présenter les caractéristiques essentielles.

1.3.4.1 Mémoires mortes programmables.

Les PROM sont construites sur un réseau de ET (circuits de décodage d'adresse) fixe et de OU (données placée en mémoire) programmable.

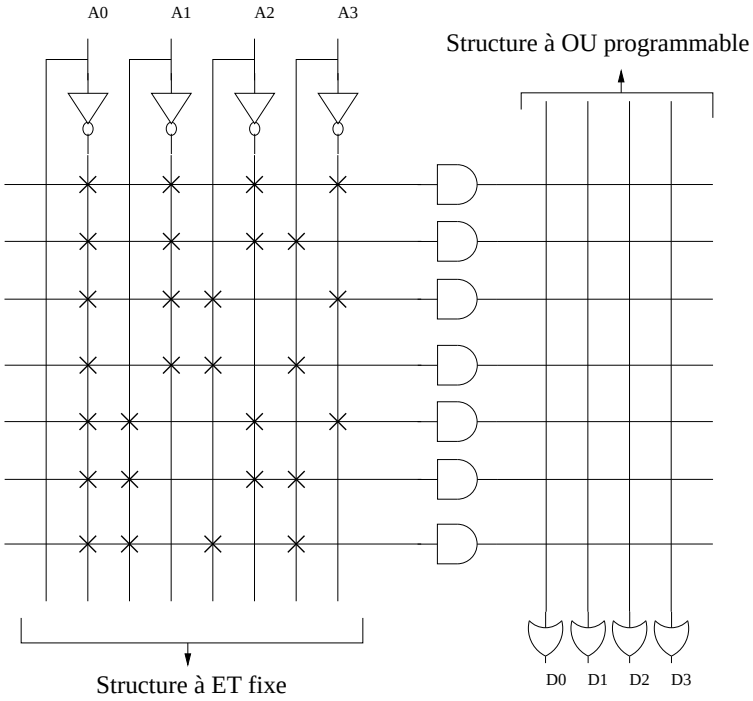


Figure I.1.11: *Structure d'une* PROM.

Les différences existant entre les types de PROM résident dans les technologies utilisées, qui permettent des taux d'intégration plus ou moins importants, et dans le mode de programmation possible. On distingue les mémoires mortes programmables une seule fois (PROM à fusible), et les mémoires mortes reprogrammables. Dans cette dernière catégorie, il existe deux techniques couramment utilisées : effacement par ultra violet ou effacement électrique. Elles se nomment alors respectivement UVPROM et EEPROM[2]. Parmi les EEPROM, les FLASH EPROM sont particulièrement performantes et reprogrammables "sur site".

Les mémoire mortes sont essentiellement utilisées pour stocker des programmes utilisés par les microprocesseurs ou des données fixes comme dans le cas de la tabulation de fonctions complexes.

Mais, elles peuvent aussi être utilisées pour remplir des fonctions logiques simples comme un décodage d'adresse ou bien, comme le montre la figure I.1.12, un automate programmable simple. Ce dernier cas est d'ailleurs couramment utilisé dans la commande des convertisseurs statiques pour générer une MLI[26].

1.3.4.2 Les PALs.

Les PALs sont historiquement les premières générations de composants dédiés au calcul logique programmable. Le principe existe depuis plus de 20 ans. Il correspond en fait à un réseau de ET *et* de OU programmable. Les sorties peuvent être rebouclées sur les entrées et la présence de bascules permet d'effectuer des fonctions combinatoires et séquentielles plus complexes. On peut ainsi réaliser avec ces composants toutes sortes de fonctions logiques de base comme des compteurs, des décodeurs, des automates, etc.

[2]Electrically Erasable Programmable Read Only Memory

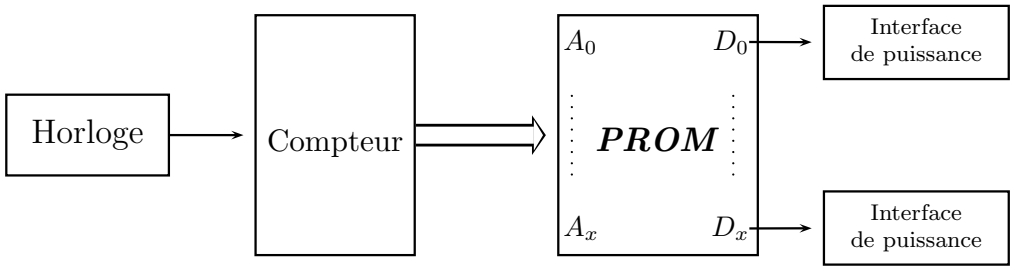


Figure I.1.12: *Structure d'un automate de commande rapprochée utilisant une* PROM.

1.3.4.3 Les EPLDs.

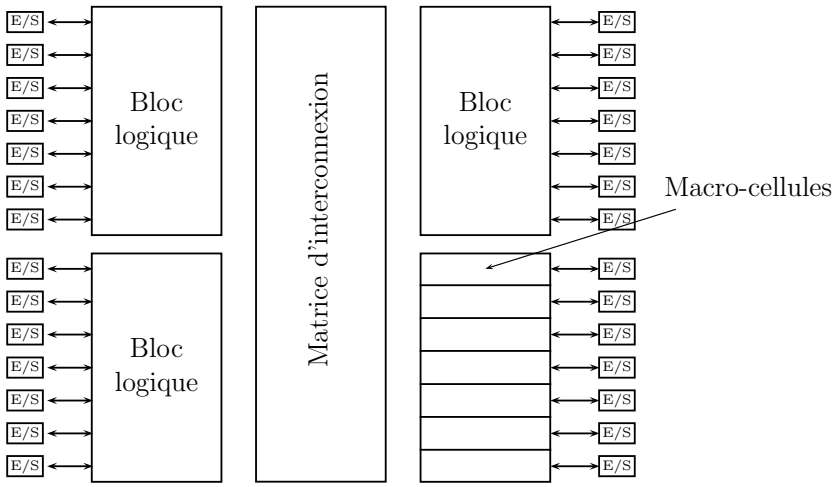


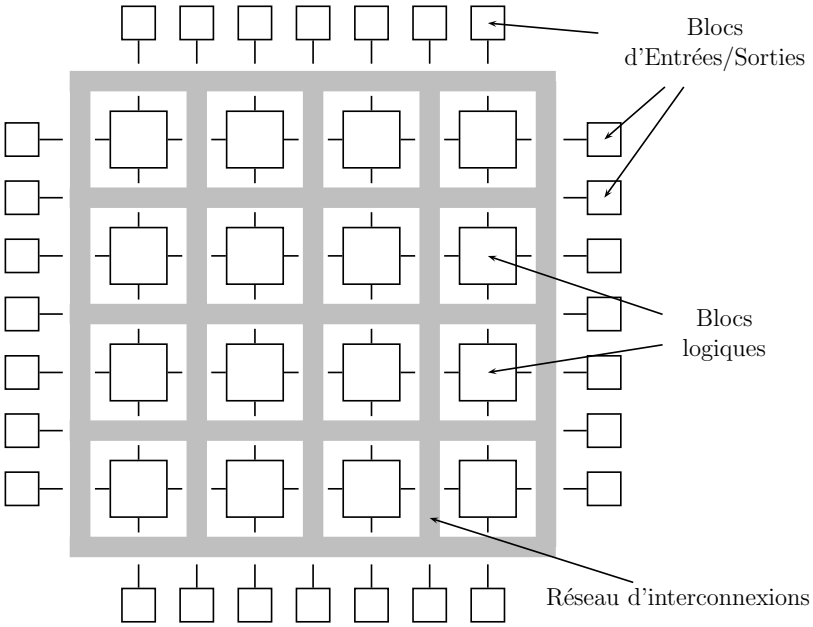
Figure I.1.13: *Structure générale d'un EPLD.*

Le terme EPLD est issu de la société Altera qui a mis sur le marché les premiers composants de ce type. Ces composants ont un niveau d'intégration bien plus important que les PALs. Une vue simplifiée est de considérer les EPLD comme un réseau de PAL. Ils peuvent en effet remplacer des réalisations utilisant 20 à 25 PAL. La figure I.1.13 montre la structure d'un EPLD. Les macro-cellules ont une structure ressemblant à celle des PAL mais la bascule peut être configurée en bascule D, RS,... Ces macro-cellules sont regroupées et reliées entre elles pour former un bloc logique. Les blocs logiques sont reliés à une matrice d'interconnexion programmable.

Seules les récentes versions d'EPLD sont programmables sur site. Cependant, les EPLDs comportent à l'intérieur des macro-cellules une structure d'interconnexion fixe, ce qui induit un grand nombre de portes inutilisables par le reste du circuit. De plus, s'ils disposent d'un nombre important d'entrées, il n'y a que peu de sorties (sorties des macro-cellules).

1.3.4.4 Les FPGAs.

Afin de remédier aux différentes limitations des EPLDs, la société Xilinx a inventé au début des années 80 des composants dont les interconnexions étaient beaucoup plus configurables.



*

Figure I.1.14: *Architecture conceptuelle d'un FPGA.*

Ces composants, nommés LCA[3], sont basés sur le principe des réseaux de portes logiques programmables par masque (technique utilisé dans la création des ASICs prédiffusés). Cependant, les portes élémentaires sont remplacées par des blocs logiques configurables capables d'effectuer différentes fonctions combinatoires et séquentielles (CLB[4]). Ils correspondent aux macro-cellules des EPLDs et le réseau d'interconnexion est totalement reprogrammable puisque l'ensemble est contrôlé par une mémoire RAM.

La société Altera a développé le même type de composants nommés "*Flex*".

L'utilisation de ces principes et de ces technologies apporte un plus grand niveau d'intégration et une flexibilité beaucoup plus importante qui permet de mieux exploiter les ressources internes de ces composants.

De plus, la programmation étant basée sur de la mémoire RAM, ils sont programmables "sur site" (en quelques millisecondes) et sont re-configurables dynamiquement.

Ces composants permettent de réaliser des fonctions logiques et des fonctions de mémorisation d'un niveau très élevé (RAM, FIFO[5]s) mais aussi de véritables UAL[6]. Ces composants sont utilisés pour effectuer du calcul dédié. En effet, en réalisant certains traitements par un "câblage de portes logiques", on peut fortement accélérer le calcul d'une tâche. Par exemple, si une opération correspondant à une instruction de microprocesseur est câblée dans un FPGA, les actions de chargement et de décodage de l'instruction deviennent inexistantes, et plusieurs opérations peuvent être traitées en parallèle.

[3]Logic Cell Array
[4]Configurable Logic Bloc
[5]First-In First-Out
[6]Unité Arithmétique et Logique

Comme nous venons de le voir, la gamme de composants logiques programmables est assez large. Elle permet de réaliser un très grand nombre de fonctions. On peut effectuer une simple mémorisation en utilisant une PROM ou bien, au coût d'une certaine complexité, on peut concevoir une commande de machine en effectuant une interconnexion de FPGA [53].

Nous avons vu aussi que chaque composant avait des spécificités particulières qui lui donnent un certain champ d'application. Cependant, ces champs d'applications se recouvrent en partie. Il est donc parfois difficile de déterminer a priori quel composant est le plus adapté pour réaliser une fonction particulière.

Notons aussi que la limitation première de ces composants est liée au nombre de portes disponibles, même si cette contrainte est de plus en plus repoussée vu le niveau d'intégration utilisé.

1.3.5 Les Entrées Analogiques.

Nous venons de voir les différentes technologies qui pouvaient répondre à la réalisation des ensembles {Unités de calculs, Unités Mémoires}. Ce sont des ensembles qui sont primordiaux pour réaliser un système de commande numérique. Mais ceux-ci ne pourront fonctionner correctement que s'ils reçoivent une information adaptée. C'est le rôle des CAN[7] que de transmettre les informations analogiques au "monde" du numérique. Leur choix est donc tout aussi important.

Ces convertisseurs ont un double effet sur le signal analogique initial :
- l'échantillonnage : on passe d'une valeur continue dans le temps à une valeur discrète.
- Le deuxième impact, correspond à la quantification. Une grandeur continue peut prendre une infinité de valeurs alors que, dans un système numérique, le nombre de valeurs que peut prendre une grandeur est fini. Les CAN's génèrent alors un bruit dit "bruit de quantification" par rapport à la grandeur analogique.

De plus, les calculs qu'effectuent les convertisseurs, pour transformer une information analogique en une information numérique, demandent un temps non négligeable par rapport aux contraintes de calculs liées à la commande des systèmes électrique.

Le bruit de quantification ainsi que le temps de conversion sont alors les paramètres les plus importants à prendre en compte lors du choix des CAN's pour leur utilisation dans un dispositif de commande.

1.4 Conclusion.

Nous avons vu dans ce chapitre qu'un dispositif de commande pouvait être chargé d'effectuer des tâches de différents types (commande, observation, surveillance, ...). Le spectre des fonctionnalités requis par un dispositif de commande peut donc être très large.

Nous avons vu ensuite que la réalisation des différentes fonctionnalités pouvait se faire en utilisant l'association de différentes technologies numériques. Cependant, lorsque l'objectif est de concevoir une architecture en totale adéquation avec les algorithmes de commande, afin d'obtenir les meilleurs performances pour un coût, une complexité et un temps de développement les plus faibles possibles, le choix des différents composants et de leur association est extrêmement difficile. En effet, le nombre de paramètres à prendre en compte et leurs interactions rend délicat le choix d'une architecture.

[7] Convertisseurs Analogique Numérique

Actuellement on sait que l'association des technologies numériques permet de répondre aux différentes fonctionnalités demandées. Cependant une sous-exploitation de ces technologies et de leurs associations est palliée par l'utilisation des composants sur-performants. Ceci implique un coût et un temps de développement non négligeable.

La recherche d'une méthodologie pour la conception d'une architecture optimisée paraît donc intéressante surtout lorsque le dispositif doit ensuite être produit de manière industrielle. Ce domaine de recherche concernant les systèmes purement électroniques est actuellement en plein effervescence, notamment pour les systèmes mixtes logiciels et matériels. Les méthodologies de conception des systèmes mixtes sont nommés méthodologies de CO-DESIGN. Dans le chapitre suivant, nous allons décrire à quoi correspond une telle méthodologie et quels sont les outils qu'elle utilise. Ainsi nous pourrons voir s'il est possible d'exploiter de tels outils dans la recherche d'une architecture optimale pour la commande des systèmes électriques.

Chapitre 2

Les éléments du Co-Design

2.1 Introduction.

L'évolution des technologies dans le domaine de l'électronique numérique a ces dernières années été d'une importance considérable. Cette progression s'est faite tant au niveau de l'intégration des circuits spécifiques qu'au niveau des architectures de microprocesseurs et de leur fréquence de fonctionnement. Cette évolution engendre des potentialités très importantes. Cependant, ces potentialités sont actuellement sous-exploitées car l'évolution des méthodes de conception associées n'a pas suivie la même croissance. Il existe notamment des lacunes dans les méthodologies de conception lorsqu'il s'agit de définir une architecture combinant des microprocesseurs standards et des composants spécialisés.

Le terme de Co-design est apparu pour définir une méthodologie de conception des systèmes mixtes : matériels (ASIC's incluant les composants logiques programmables) et logiciels (algorithmes programmés sur microprocesseurs).

La mise en place de systèmes complexes, rendue possible par les progrès technologiques, ne permet plus d'appréhender aisément les problèmes de conception de bas niveau. Il faut alors augmenter le niveau d'abstraction ce qui implique la construction d'outils pour l'interprétation, l'analyse et la retranscription vers des fonctions de bas niveau. Les méthodologies de co-design se proposent d'intégrer ce type d'outils dans le cadre des systèmes mixtes matériel/logiciel. Ces méthodologies doivent notamment prendre en compte le caractère hétérogène de tels systèmes. Le co-design cherche à répondre à deux exigences principales qui sont :

- déterminer le meilleur compromis entre la réalisation logicielle et la réalisation matérielle,
- augmenter la productivité en effectuant la conception du logiciel et du matériel de manière simultanée (***co***ncurrent ***design***).

Nous allons essayer de définir au mieux les différentes activités que couvre une méthodologie de co-design, afin de voir dans quelle mesure une telle méthodologie peut répondre aux problèmes de la conception d'un dispositif de commande numérique dédié aux systèmes électriques.

2.2 La méthodologie de Co-design

2.2.1 Cycle de conception actuel des systèmes électroniques

Le cycle de conception que nous allons aborder concerne les systèmes de l'électronique numérique. Ils sont essentiellement composés des éléments énoncés dans la section 1.3.2 du chapitre 1. Ils comportent donc des composants dit standards du type microprocesseurs pouvant embarquer du logiciel (nous les nommerons processeurs logiciels), et des composants dédiés que sont les ASIC's et les composants logiques programmables (cette deuxième catégorie sera nommée processeur matériel).

Le cycle de conception d'un système électronique composé des deux types de processeurs suit le plus souvent le diagramme proposé dans la figure I.2.1.

Le cahier des charges. Il est le plus souvent rédigé en langage naturel. Il sert de point de départ et de support pour déterminer les spécifications comportementales du système.

Spécifications systèmes. Ces spécifications ont pour but de décrire les différentes fonctions qui devront être développées ainsi que leurs interactions. L'assemblage des différentes fonctions définit alors le système tel qu'il doit être pour remplir les conditions décrites dans le cahier des charges. Ces spécifications sont très souvent, elles aussi, rédigées en langage naturel. Cela comporte plusieurs points négatifs :

- en utilisant un tel mode de spécification, il peut apparaître des ambiguïtés dû au vocabulaire utilisé.
- pour des raisons de lisibilité une description peut parfois manquer de précision.
- en utilisant le langage naturel, la description correspond souvent à une liste de scénarios possibles, mais une telle énumération peut ne pas être complète.

Ces différents points sont alors sources d'incompréhension et d'erreurs. Afin d'annuler ces sources d'erreurs, il est de plus en plus courant d'utiliser des langages plus formels pour spécifier le système. L'utilisation de tels langages permet de développer des outils aidant à valider les spécifications. Ce type d'outil est d'ailleurs utilisé dans la description des systèmes logiciels et des télécommunications.

Le partitionnement. Il correspond aux choix de l'architecture et à la répartition des différentes tâches sur les composants choisis. Ces choix sont faits a priori. Ils sont souvent dirigés par l'expérience des décideurs ou bien par la réutilisation de produits déjà existants. L'exploration des différentes possibilités semble en effet souvent trop longue et trop coûteuse

Les spécifications logicielles et matérielles. Une fois le partitionnement déterminé, le travail est souvent attribué à deux équipes distinctes.

Or, de l'expérience d'un corps de métier naît un système de spécifications adapté aux problèmes qu'il est amené à rencontrer. Les spécifications qui étaient décrites au niveau système sont donc souvent réécrites sous une forme adaptée et compréhensible pour l'équipe considérée (logicielle ou matérielle).

Cette phase de "reprise en main" met souvent en exergue des points non pris en compte dans la spécification système, ce qui amène alors un premier rebouclage afin d'améliorer ces spécifications.

Synthèse du logiciel et du matériel. Elles sont souvent effectuées de manière trop séparées, sans prendre en compte les caractéristiques de la partie complémentaire.

L'intégration. Ce n'est qu'à partir de cette étape que le logiciel peut être réellement testé sur le prototype matériel. Mais, la détection d'erreurs à ce stade est souvent coûteuse car leurs corrections nécessitent des retours sur l'étape de synthèse.

L'évaluation des performances. Lorsque le prototype fonctionne, on peut alors mesurer les performances précises du système. Si celles-ci ne sont pas suffisantes, on

peut être amené à réécrire une partie du logiciel, ou bien remonter dans la chaîne de conception afin de réévaluer le partitionnement. Dans le pire des cas, un autre choix doit être fait pour l'architecture en terme de composants.

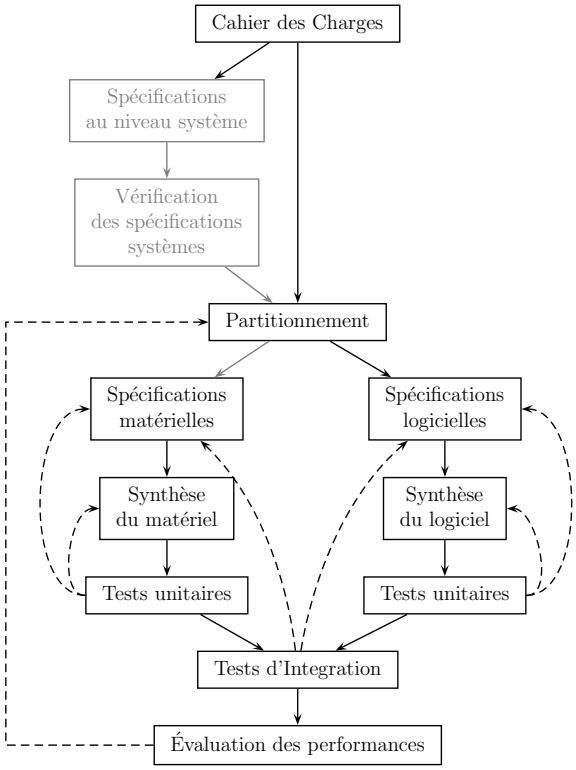


Figure I.2.1: *Cycle de conception actuel*

Un tel cycle de conception peut amener à des problèmes qui sont résolus par de nombreux retours en arrière. Ces retours sont coûteux en temps de développement et peuvent aussi remettre en cause le choix des composants matériels.

Une méthodologie de co-design tente de résoudre les différents problèmes qui viennent d'être soulevés par l'utilisation de méthodes et d'outils tout au long de la chaîne de conception.

2.2.2 La méthodologie de co-design

Le but d'une méthodologie de co-design est d'aider le concepteur à obtenir un produit fini en respectant les contraintes fixées par le cahier des charges pour un temps de développement et un coût les plus faibles possibles.

De manière générale, une méthodologie organise et coordonne l'utilisation de différentes techniques et de différents outils qui permettent d'aboutir au produit final. Les études sur le co-design tendent à définir des méthodes et des outils cohérents qui peuvent être appliqués tout au long de la chaîne de conception du produit. Le co-design englobe par conséquent les différents domaines abordés lors de la conception d'un système mixte. Ces domaines sont : la spécification au niveau système, l'analyse des spécifications et la répartition logiciel/matériel, la synthèse logicielle et matérielle et l'estimation de performances.

Le co-design va donc rajouter, dans les différentes étapes de conception, des outils permettant de valider et de tester les choix effectués. L'ajout de ces outils et des for-

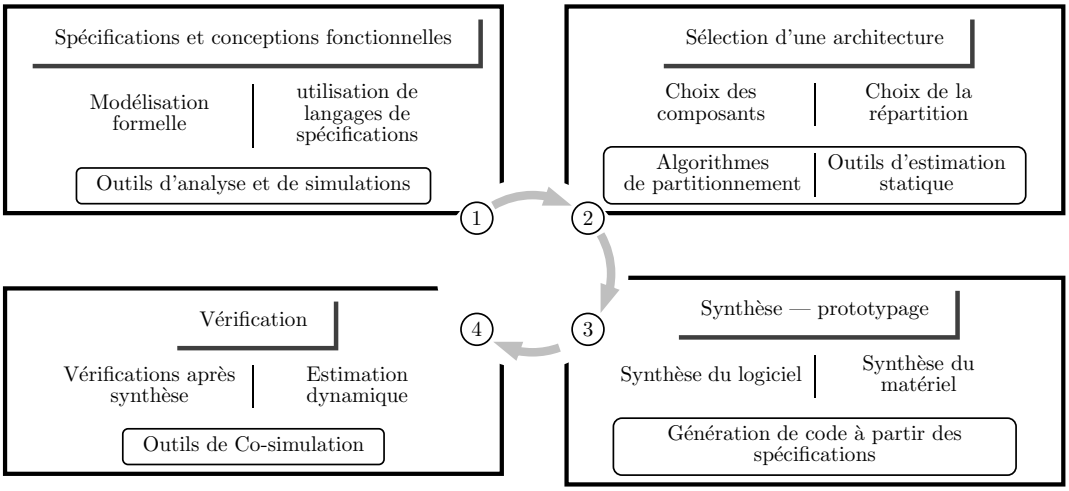


Figure I.2.2: *Les domaines de recherche pour le co-design*

malismes associés semble rendre le cycle de développement plus lent mais il permet de détecter certaines erreurs très tôt dans le cycle de conception et diminue donc leurs propagations jusqu'à l'étape d'intégration. Cette diminution permet de réduire le nombre de "rebouclage" et accélère donc le processus de conception complet.

Nous allons aborder chacun de ces domaines du point de vue du co-design afin de montrer les différents outils qui peuvent être élaborés dans chacun des ces domaines ainsi que leurs liaisons.

2.3 Les spécifications fonctionnelles

Les spécifications au niveau système permettent de décrire l'ensemble des fonctions nécessaires ainsi que leurs différents liens sans prendre en compte les problèmes d'implantation.

La définition de modèles formels pour la description fonctionnelle d'un système est un des thèmes de recherche du co-design. En utilisant de tels modèles, il est possible d'éliminer la plupart des erreurs que nous avons mentionnées dans le cadre de la spécification à l'aide d'un langage naturel.

L'utilisation de spécifications formelles doit donc permettre de rendre les descriptions non-ambigües et complètes pour décrire la totalité des comportements. Ces spécifications doivent pouvoir être réutilisables afin d'accélérer les développements futurs.

Se pose alors le problème du choix ou de la création de langages permettant d'élaborer les spécifications et de représenter aisément les modèles formels utilisés. Des outils basés sur les langages choisis peuvent être construits pour analyser, vérifier et simuler les spécifications.

Pour présenter le type de langages et de modèles exécutables qui sont utilisés dans le cadre du co-design, nous allons tout d'abord rappeler les concepts qui doivent être pris en compte pour la spécifications des systèmes temps-réel.

2.3.1 Les systèmes temps réel

Un système temps réel doit répondre à un besoin dans un temps déterminé. Ce besoin est signalé par des évènements extérieurs au système. Le calcul d'une réponse doit alors se faire sous une contrainte de temps tout en restant à l'écoute d'autres évènements extérieurs.

Les systèmes temps réel sont alors souvent composés de deux classes de systèmes :

(a) Système transformationnel (b) Système réactif

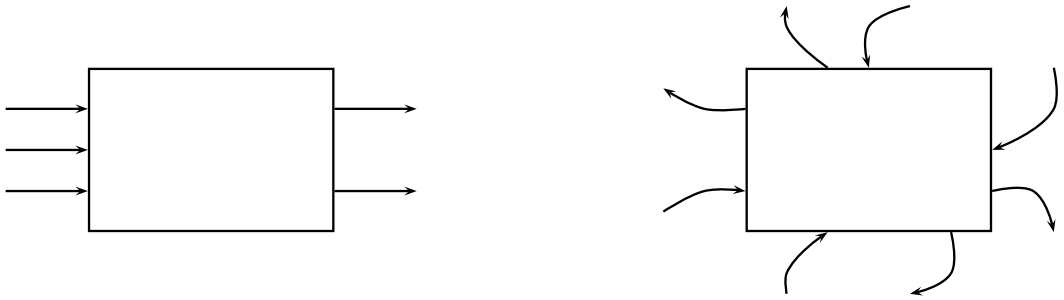


Figure I.2.3: *Les systèmes temps réels composés de deux classes*

- les systèmes transformationnels : ce sont des systèmes qui reçoivent des entrées lors de leurs activations et qui calculent des sorties dépendantes de ces entrées.
- les systèmes réactifs : ce sont des systèmes qui surveillent de manière continu l'environnement extérieur et émettent leurs grandeurs de sorties lors d'évènements associés aux signaux extérieurs. (ex : les interfaces homme/machine comme la gestion du clavier).

Pour spécifier l'association et le fonctionnement des tels systèmes, on doit pouvoir spécifier en plus des actions de calculs, 4 concepts de base. Ces concepts sont :

la concurrence : elle permet d'indiquer les actions qui peuvent être menées en parallèle. Cela permet de simplifier la représentation du système. La concurrence doit pouvoir être représentée à plusieurs niveaux (tâche, instruction, opération).

la hiérarchie : pouvoir hiérarchiser un système permet de contrôler sa complexité. Dans le cas d'un système complexe, il est extrêmement difficile de décrire le système directement. On préférera faire une décomposition en sous-systèmes plus faciles à traiter.

la communication : décrire la communication est essentiel pour représenter les liens entre les modules concurrents. Il existe principalement deux types de communication : la communication par mémoire partagée, où les différents modules écrivent et lisent dans une mémoire commune, et la communication par échange de message, où les modules communiquent selon un protocole particulier.

la synchronisation : elle permet de coordonner l'exécution des différents modules concurrents ainsi que leurs communications.

Nous allons voir que peu de modèles au niveau système peuvent représenter ces quatre concepts de manière équivalente. Notons aussi que les modèles doivent permettre la spécification de calculs arithmétiques et logiques ainsi que les expressions algorithmiques usuelles telles que l'itération et le branchement.

2.3.2 Les représentations formelles fondamentales

Nous allons énumérer un certain nombre de modèles de bases utilisés pour représenter les spécifications d'un système sous une forme analysable et exécutable.

Ces modèles diffèrent par le fait qu'ils représentent un système sous un axe particulier. On peut d'ailleurs classer ces représentations en différentes catégories : les modèles orientés état et les modèles orientés activités.

2.3.2.1 Les modèles orientés états

Cette modélisation est adaptée pour représenter les systèmes dits de contrôle ou réactifs. Ils permettent de représenter le comportement temporel d'un système par rapport aux différents évènements qui peuvent survenir. Ce sont des modèles qui permettent de recenser les états que peut prendre le système et qui expriment les conditions de passage d'un état à l'autre.

Automate à états finis

Le principe des automates est de représenter la solution d'un problème par une succession d'étapes définies, séparées les unes des autres. Chaque étape est représentée par une valeur faisant partie d'un ensemble fini que l'on appelle l'état (interne) du système. Cet état est physiquement matérialisé par un nombre contenu dans une mémoire.

La machine à états finis est un modèle formel et déterministe (un seul état possible) qui est utilisé dans la synthèse des fonctions logiques séquentielles.

La représentation formelle d'un automate à états finis correspond à un Quintuplet :

$$< E, S, Q, w \ : E \times Q \rightarrow S, \delta \ : E \times Q \rightarrow Q >$$

- $S = \{s_1, s_2, \ldots\}$ correspond à l'ensemble des combinaisons des sorties,
- $E = \{e_1, e_2, \ldots\}$ correspond à l'ensemble des combinaisons des entrées,
- $Q = \{q_1, q_2, \ldots\}$ correspond à l'ensemble des états possibles du système,
- w est la fonction de calcul du vecteur de sortie,
- δ est la fonction de calcul de l'état suivant.

Une machine à états finis peut se représenter graphiquement par un diagramme d'états et de transitions (figure I.2.4).

Exemple de représentation : contrôle d'un ascenseur. Le contrôle d'un ascenseur peut être modélisé par une machine à états finis comme suit :

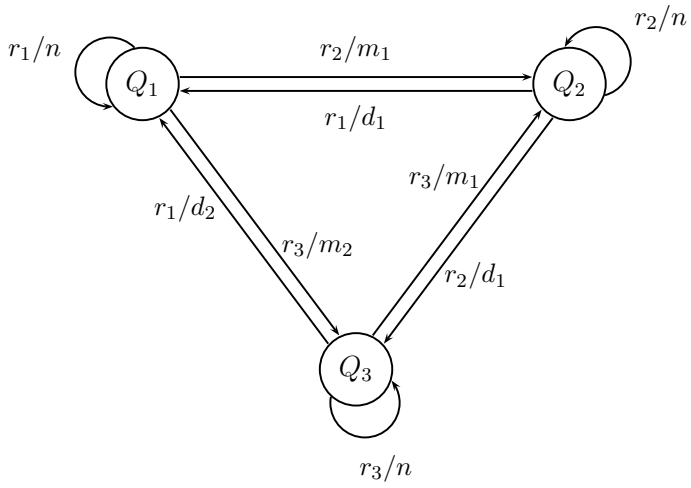


Figure I.2.4: *Exemple de modélisation par machine à états finis*

$Q = \{q_1, q_2, q_3\}$ les 3 étages représentent les trois états possibles.

$E = \{r_1, r_2, r_3\}$ les requêtes d'accès à un des trois étages représentent l'ensemble des entrées.

$S = \{d_1, d_2, n, m_1, m_2\}$ l'ensemble des sorties est représenté par la direction et le nombre d'étages à franchir (d_1 représente ainsi la demande de descendre d'un étage).

Cependant, une représentation par machine à états finis deviendra très vite illisible pour des applications importantes, qui nécessitent d'utiliser un très grand nombre d'états. Cela peut être évité en étendant le formalisme par l'utilisation d'instructions d'assignation de variables internes. Ces variables réduisent la complexité apparente des automates en cachant des informations de moindre importance. Par exemple, si on utilise une variable pour exprimer la valeur d'un octet, on n'aura pas besoin d'exprimer les différentes valeurs de cet octet en 256 états différents. On parle alors de machine à état finis étendue ou bien de machine à états finis avec chemin de données. Ces modèles ne permettent pas cependant de représenter la concurrence et la hiérarchie. Il ne rend pas non plus compte de la complexité d'une activité en terme de calcul pour le traitement des données.

Réseau de Pétri

Le réseau de Pétri est représenté par un quadruplet $R =< P, T, Pre, Post >$.
où :
- P : est un ensemble fini de places,
- T : est un ensemble fini de transitions,
- Pre : $P \times T \longrightarrow N$ est l'application places précédentes,
- Post : $P \times T \longrightarrow N$ est l'application places suivantes.

Les applications peuvent être représentées de manière algébrique en considérant le réseau de Pétri comme un graphe contenant deux types de noeuds : les places et les transitions. L'application *Pre* représente alors les arcs qui relient une place à une transition et l'application *Post* représente les arcs qui relient les transitions aux places.

Un réseau de pétri permet de représenter les systèmes ayant des états concurrents. Il est possible de représenter un grand nombre de caractéristiques d'un système temps réel comme la mise en séquence (a), le branchement (b), la synchronisation (c), le partage de ressources (e) et la concurrence (d) (figure I.2.5).

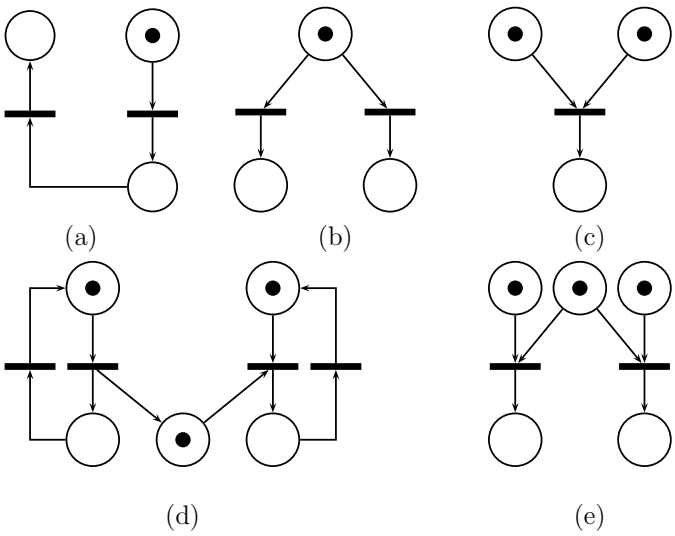


Figure I.2.5: *Représentations par les réseaux de Pétri de changements d'états d'un système temps réel*

Ils ont été utilisés pour la mise en place d'une méthodologie de conception pour la commande de systèmes électriques axée sur la sûreté de fonctionnement[5].

La notion de hiérarchie dans les réseaux de pétri n'est cependant pas présente.

2.3.2.2 Les modèles orientés activités

Ces modèles sont utilisés pour représenter les systèmes dont la tâche principale correspond à un traitement de données (systèmes transformationnels).

Diagramme de flots de données

Le diagramme de flots de données représente un calcul où les arcs représentent le transfert d'une donnée et les noeuds représentent une activité comme une instruction, une opération arithmétique, une fonction, etc. On distingue aussi deux autres types de noeuds, les noeuds d'entrées et de sorties et les noeuds de stockage (mémoire). Ce modèle supporte la hiérarchie : l'activité d'un noeud peut être un autre graphe de flots de données.

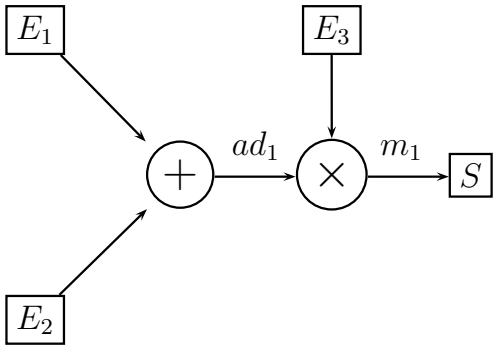


Figure I.2.6: *Exemple de diagramme de flots de données*

Cette modélisation est souvent utilisée dans le domaine du traitement du signal pour représenter différents calculs comme les filtres numériques, ou bien les transformées de Fourier rapide (FFT[1]). Une propriété importante de ce type de diagramme est qu'il représente naturellement la dépendance des opérations par rapport aux données. On peut déterminer aisément l'ordonnancement des tâches ainsi que le parallélisme potentiel.

Diagramme de flots de contrôle

Dans ces diagrammes, les arcs, qui indiquent la mise en séquence des opérations, représentent un flot de contrôle. Ils sont adaptés pour représenter des algorithmes comportant des branchements et des itérations. Ce type de graphe permet de spécifier l'ordonnancement et les activités qui ne dépendent pas d'évènements externes.

2.3.2.3 Les modélisations hétérogènes

Les différents modèles que nous venons de voir sont très fortement orientés et ne permettent de représenter que des systèmes bien particuliers. Aussi, on associe parfois les différents modèles afin de pouvoir représenter aisément une plus grande gamme de systèmes. On peut citer comme exemple caractéristique le modèle de graphe de flots de données et de contrôle qui associe, comme son nom l'indique, les deux diagrammes d'activités vu précédemment. Le couplage de deux diagrammes se fait par la hiérarchie : une activité est soit un diagramme de flots de contrôle, soit un diagramme de flots de données, soit une activité de base.

Parmi les projets de co-design, le projet PTOLEMY [63] utilise la modélisation hétérogène. Dans ce projet, un grand nombre de modèles différents ont été implantés à

[1]Fast Fourier Transformed

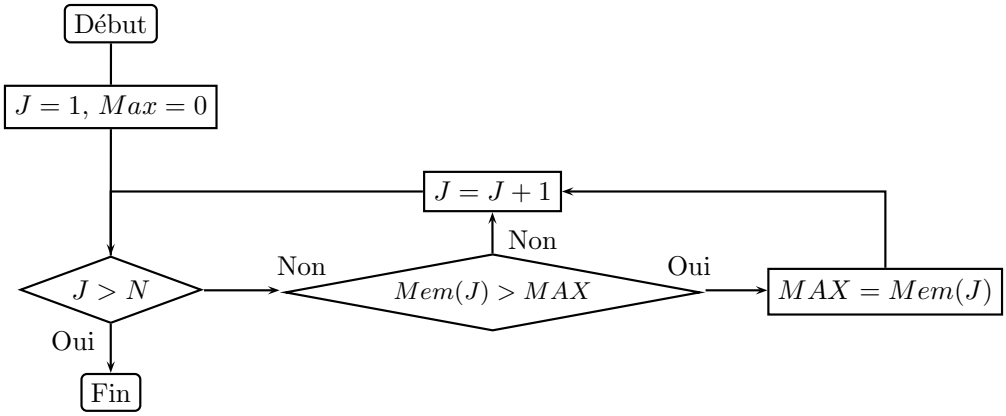


Figure I.2.7: *Exemple de diagramme de flot de contrôle*

partir d'une plateforme ouverte orientée objet et peuvent être combinés de la même manière que les CDFG[2].

2.3.3 Les langages de spécification et les projets de Co-design

Nous venons de voir les différents modèles permettant de représenter un système, le simuler et l'analyser. La construction de ces modèles analysables et/ou exécutables se fait alors par l'intermédiaire d'un ou plusieurs langages qui permettent de capturer les spécifications du système.

Dans le domaine du co-design, différents projets ont vu le jour et ces projets ont choisi un ou des langages de spécifications différents selon le type de système qu'ils doivent étudier et la méthodologie qu'ils veulent établir. Nous récapitulons dans cette section les langages les plus utilisés pour la spécification de systèmes mixtes, ainsi que certains projets de co-design les exploitant.

2.3.3.1 Les langages de description matériel

VHDL

Le langage VHDL est un langage de description matériel des systèmes numériques. C'est un langage standard IEEE.

Le VHDL est basé sur l'assemblage de composants nommés entités. Ces entités peuvent être définies par le langage sous la forme d'instructions séquentielles et/ou parallèles. Ces instructions peuvent soit correspondre à l'assemblage d'autres entités, soit correspondre à des instructions de base du langage.

Certaines variables peuvent être évaluées dans le temps.

La communication entre les entités se fait par l'intermédiaire de mémoires partagées représentées par des signaux.

C'est un langage qui permet de représenter la plupart des concepts nécessaires à la description des systèmes temps réel. Il n'a cependant pas d'instructions spécifiques aux transitions d'états et permettant de définir des exceptions.

Il est utilisé par plusieurs projet de co-design notamment le projet Lycos (LYngby COSynthesis)[48]. Ce projet, de l'université technique du Danemark, traduit les spécifications VHDL en un graphe de flots de données et de contrôle sous un format intermé-

[2]Control/Data Flow Graph

diaire qui peut ensuite servir à différents outils et notamment un outil de simulation de ces spécifications et un outil de partitionnement.

HardwareC

Le langage HardwareC est un langage de description matériel issu du langage C et étendu par l'adjonction d'instructions et de constructions utiles à la description matérielle des systèmes numériques. Il n'est cependant que très peu utilisé par les outils commerciaux de synthèse numérique.

Le projet Vulcan dirigé par R.K.GUPTA utilise ce langage de spécification pour le traduire en un graphe de flots de données et de contrôle qui doit ensuite être implanté sur une architecture fixe comprenant un ASIC et un microprocesseur.

SystemC

Le langage SystemC est un ensemble de classes C++ dédié à la conception des systèmes électroniques. En fait, SystemC, tout en restant compatible avec le C Ansi, introduit de nouveaux concepts lui permettant de faire de la conception au niveau système d'un circuit :

- gestion des aspects temporels (description des horloges) ;
- bibliothèque de classes C++ pour la description de ports, d'interfaces, etc. ;
- gestion du parallélisme ;
- spécifications pour la vérification.

Les différents types de base sont : le bit, le vecteur de bit, l'entier, le flottant, le signal 4 états (0,1,X,Z), et même le nombre à virgule fixe.

Un consortium : The Open SystemC Initiative [1] essentiellement dirigé par Synopsys, a mis en place une plateforme d'échange qui contient notamment un compilateur et l'ensemble des librairies nécessaires à la simulation du langage SystemC. Ce langage a pour objectif de supplanter les langages HDL en introduisant la spécification des parties logicielles.

2.3.3.2 Langages de spécifications dédiés.

SDL

Le langage SDL (Specification and Description Language) a été défini pour décrire les systèmes temps réels distribués et dédiés à la télécommunication. Il est standardisé par l'ITU[3]. Un système décrit en SDL est composé d'un ensemble de processus communiquant au travers de signaux[49]. Ces processus sont ensuite définis comme des automates à états finis. La communication (implicite au langage) est totalement asynchrone.

Ce langage est utilisé dans le projet COSMOS [23] qui traduit les spécifications en un format intermédiaire qui représente un ensemble de machines à états finis étendues (acceptant la hiérarchie) et communiquant par des appels de procédure à distance RPC (Remote Procedure Call).

StateCharts

Les STATECHARTS, définis par David HAREL, correspondent à un langage graphique permettant de représenter des machines à états finis avec en plus une description de concurrence et de communication. Ce langage est utilisé pour la spécification des logiciels de systèmes temps réels embarqués

Il existe un langage très proche des StateCharts défini dans le cadre du co-design comme une extension, le langage SpecChart de D. GAJSKI [24].

[3]International Telecommunication Union

2.3.3.3 Les langages Synchrones

Les langages synchrones ont été définis pour décrire le fonctionnement des systèmes temps réel. Pour cela, une sémantique temporelle a été ajoutée par rapport aux langages asynchrones tels que AdA, C, ... Ceci permet d'indiquer à quel instant intervient une opération. Un instant donné n'est pas indiqué par rapport au temps physique mais au travers du comptage d'évènements[28]. Afin de rester indépendant de toutes implémentations, ces langages font deux hypothèses fortes : les temps de calcul et les temps de communication sont considérés comme nuls. Parmi les langages synchrones, deux langages ont été utilisés pour des projets de Co-design.

Esterel

Le langage Esterel est un langage synchrone orienté contrôle. Il est utilisé dans le projet de co-design POLIS [61] qui retranscrit les spécifications en un ensemble de machines d'états finis supportant la concurrence et la communication Il existe aussi des compilateurs du langage Esterel en un ensemble d'équations booléennes utilisées dans la conception de processeur matériel.

Signal

Le langage Signal [38] est un langage synchrone orienté activité. Il est utilisé dans le projet Syndex pour construire un graphe de flots de données utilisé ensuite pour répartir les tâches sur une architecture multi-processeurs.

On constate qu'un grand nombre de langages a été utilisé pour saisir les spécifications dans le co-design. On constate que le choix d'un langage dépend des paramètres sur lesquels on veut agir pour tester différentes conceptions et donc du type d'outils que l'on veut mettre en place pour aider le concepteur mais aussi du type de système à concevoir.

Si l'application est essentiellement réactive et contient peu de calcul, on utilisera plutôt un langage permettant de générer facilement un ensemble de machine à états finis. Si l'application comporte beaucoup de sous-systèmes du type transformationnel, on choisira alors un langage permettant de générer un graphe de flots de contrôle et de données.

Dans le cas d'un système mixte, où l'on considère que la réactivité et la transformation ont autant d'importance, on utilisera une spécification hétérogène.

La granularité est aussi un paramètre important qui influe sur le choix du langage de spécification à utiliser. Le choix de ce paramètre dépend du type d'outil utilisé en aval des spécifications, lorsque le concepteur fait faire le partitionnement et la synthèse.

Si l'on considère par exemple une granularité au niveau de l'instruction, voir au niveau du bit, le choix d'un langage de spécification tel que SDL ou bien StateCharts ne semble pas réellement adapté, car le formalisme ne permet pas de décrire les fonctionnalités de manière suffisamment simple.

2.4 L'exploration d'architectures.

2.4.1 Les choix dans une méthodologie d'exploration d'architectures

Une fois que les spécifications ont été validées, il faut déterminer l'architecture sur laquelle seront développées les fonctions. Cette architecture doit satisfaire un ensemble de contraintes et/ou minimiser un certain nombre de paramètres comme le temps de calcul, la consommation, le poids, la fiabilité, le coût, etc.

Lors du choix de l'architecture, le concepteur est amené à choisir les fonctionnalités qui seront réalisées en logiciel, c.-à-d. implantées sur un ou plusieurs microprocesseurs, que nous nommerons processeurs logiciels, et celles qui seront réalisées par l'association de composants spécifiques, que nous nommerons processeurs matériels. Mais pour cela, le concepteur doit aussi choisir le type et le nombre de composants qui constitueront l'architecture. Il y a donc deux problématiques dans l'exploration d'architectures qui sont : la sélection des composants et le partitionnement des fonctionnalités sur ces composants.

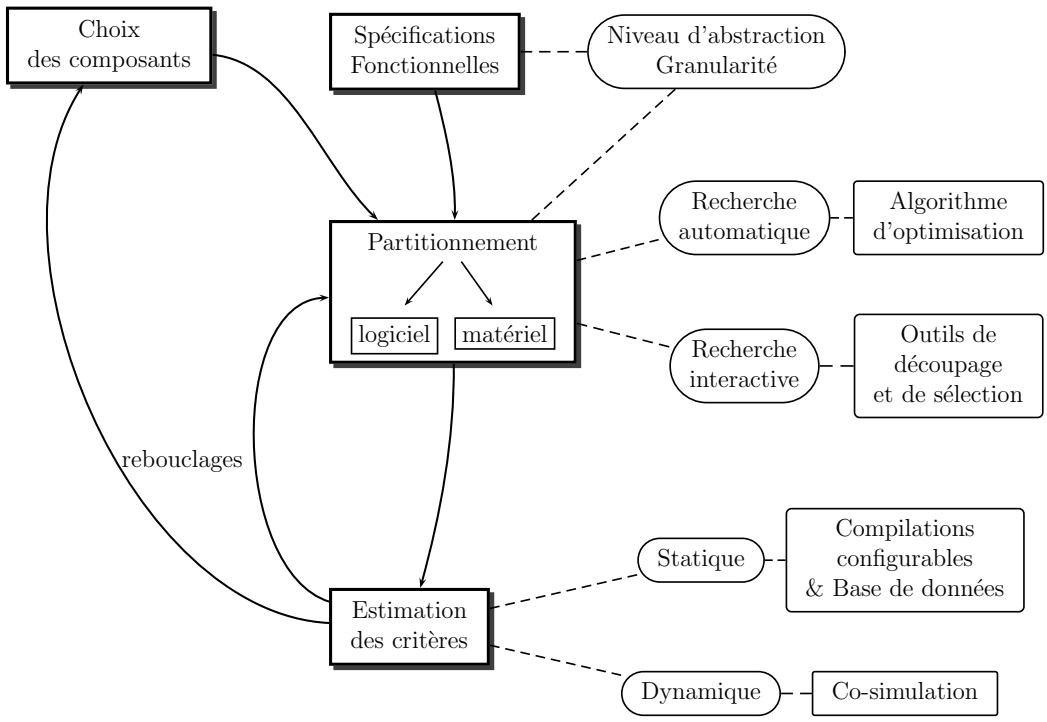


Figure I.2.8: *Les principaux problèmes associés à l'exploration d'architecture*

Un des buts principaux de la recherche en co-design est d'élaborer les outils permettant d'aider à choisir cette architecture. La figure I.2.8 représente le cycle de recherche d'une architecture optimale et en montre les différentes étapes.

Il apparaît alors deux types d'outils nécessaires à l'élaboration de ce cycle. Le premier correspond aux outils de partitionnement et le second comprend les outils d'estimation de performances associés à une architecture. Ces outils peuvent être de nature différente selon les choix méthodologiques qui sont :

- la granularité,
- le type de partitionnement : automatique ou interactif,
- le type d'estimation : statique ou dynamique.

2.4.2 La granularité

Les spécifications fonctionnelles peuvent avoir différents niveaux de granularité. La granularité correspond aux éléments qui seront considérés comme indivisibles pour le partitionnement. Ces atomes peuvent être la tâche, la fonction ou procédure, ou bien l'instruction.

Le choix de la granularité influe fortement sur le type de partitionnement et les performances globales du système [32]. Si l'on considère par exemple un système constitué

de deux tâches macroscopiques T1 et T2 et une architecture constituée d'un processeur logiciel (SW) et d'un processeur matériel (HW), il y a alors quatre possibilités de partitionnement :

1. $T1, T2 \longrightarrow SW$
2. $T1 \longrightarrow SW;\ T2 \longrightarrow HW$
3. $T2 \longrightarrow SW;\ T1 \longrightarrow HW$
4. $T1, T2 \longrightarrow HW$

Les cas 1 et 4 sont triviaux. Les systèmes logiciels/matériels correspondent aux cas 2 et 3. Cependant, il se peut que les contraintes du système soient satisfaites en n'implantant dans le processeur matériel qu'une boucle de calcul intensive contenue dans une des deux tâches. Mais, étant donnée la granularité considérée, toute la tâche doit être implantée sur le processeur matériel ce qui est plus coûteux que nécessaire. Cette exemple montre l'impact d'une granularité trop "grossière".

Dans le cas contraire, si la granularité est trop fine, le champ d'exploration peut devenir trop important.

La granularité devra donc être choisie en fonction du niveau de complexité des systèmes étudiés et des outils de spécification et de partitionnement que l'on veut mettre en place.

2.4.3 Le partitionnement

L'exploration de toutes les architectures possibles pour un système donné est impraticable en réalité, car pour un nombre de composants C et une granularité composés de A atomes, l'ensemble des solutions possibles est exponentiel : C^A. Il existe alors deux approches différentes pour le partitionnement.

2.4.3.1 La recherche interactive

Cette recherche se base sur l'expérience et l'esprit d'analyse du concepteur qui décide des partitionnements à estimer afin de trouver l'architecture qui permet de respecter les contraintes. Dans ce cas, l'environnement de co-design doit fournir à l'utilisateur une boîte à outils lui permettant de sélectionner un ensemble d'atomes et de les affecter à un composant particulier. Ce type d'outil a été implanté dans le projet de co-design Cosmos [49]. Le projet Ptolemy utilise des mécanismes identiques permettant de découper les spécifications en plusieurs tâches et d'associer un langage particulier pour la synthèse de chaque tâche. Le découpage doit en plus tenir compte des interfaces de communications entre les tâches qui ne sont pas sur le même composant.

2.4.3.2 La recherche automatique.

La recherche automatique est basée sur un algorithme dit "heuristique" qui ne parcourt qu'une partie de l'arbre des solutions. Cette approche a été utilisée dans plusieurs projets. Cependant la complexité du problème étant importante, les projets fixent le choix des composants. Ce choix correspond à l'association d'un processeur logiciel et d'un processeur matériel. L'algorithme est alors un algorithme dit de partitionnement.

Le problème du partitionnement se ramène alors à un problème de répartition des tâches entre le processeur logiciel et matériel.

Pour la résolution automatique du partitionnement, un graphe de type flots de données et de contrôle est généralement utilisé. Ce type de graphe permet en effet de résoudre une partie du problème de l'ordonnancement des atomes. En effet, l'algorithme de partitionnement doit, pour répartir les différents atomes sur le processeur logiciel et le processeur matériel, déterminer l'ordre d'exécution de ceux-ci.

La répartition se fait généralement en considérant trois critères :

1. le temps de calcul,
2. la taille du système : elle correspond, dans le cas du processeur matériel, au nombre de portes logiques et de bascules utilisées et, dans le cas du processeur logiciel, au nombre d'octets nécessaires à la sauvegarde du programme et des données associées,
3. la communication entre le processeur matériel et le processeur logiciel (quantité de données transférées).

Ces trois critères dans certains algorithmes sont normalisés et composés pour construire une fonction coût :

$$\text{fonction coût} = k1 * temps\_calcul + k2 * taille + k3 * communication$$

Les méthodes de partitionnement sont alors construites de la manière suivante.

Une première partition est élaborée manuellement ou automatiquement. Certains projets partent d'une partition qui est soit entièrement logicielle (le projet COSYMA [57]) et dans ce cas elle ne répond pas, en général, au critère de temps de calcul, soit entièrement matérielle (projet VULCAN [29]) et elle a un coût important en terme de taille.

À partir de la partition initiale, l'algorithme de partitionnement va déplacer des atomes entre le processeur logiciel et le processeur matériel afin de diminuer la fonction coût et de respecter les contraintes. Le déplacement des atomes n'est cependant pas aléatoire mais se base sur des critères de proximité afin de limiter le nombre de communications entre le logiciel et le matériel.

Après chaque déplacement la fonction coût est réévaluée en estimant les différents critères et un partitionnement est de nouveau effectué jusqu'à obtenir un minimum de la fonction coût. Ce minimum est cependant très souvent un minimum local étant donné les algorithmes utilisés. En effet, on utilise des heuristiques comme les algorithmes gloutons [29], le recuit simulé [57] ou bien la programmation linéaire entière[55].

Les deux approches se basent sur une estimation des performances du système. Ces performances se composent essentiellement du temps de calcul et de la taille de chaque répartition. Notons cependant que d'autres critères peuvent aussi s'ajouter comme la consommation et le coût du système. L'estimation des performances est donc une étape très importante dans l'exploration des architectures qui ne peut se dissocier, comme nous allons le voir, de la synthèse des parties matérielles et logicielles.

2.5 Synthèse et estimation statique d'un système mixte logiciel/matériel

Pour la recherche d'une configuration optimale, il est intéressant de pouvoir estimer rapidement les différentes métriques d'une partition donnée. Ce facteur (vitesse d'estimation) permet d'augmenter, dans un temps donné, le nombre de solutions explorées dans le cadre de l'approche interactive, et d'accélérer la convergence des algorithmes, dans le cadre d'une recherche automatique. Cependant la rapidité et la précision d'une estimation sont des critères antagonistes. En effet, plus un modèle d'estimation est précis, plus le temps de calcul associé est grand. Ce rapport rapidité/précision ne doit pas être trop important car il peut influer sur le résultat final.

On peut classer les modèles d'estimation du logiciel comme du matériel en deux grandes catégories : les modèles statiques et les modèles dynamiques. Le modèle dynamique nécessite une simulation du système avec excitation des entrées. Ce type d'estimation est rarement utilisé dans le cadre d'un rebouclage pour le partitionnement automatique. Les modèles statiques sont eux, par contre, souvent utilisés. Dans cette

section, nous allons voir les différente étapes de la synthèse afin de situer les différents modèles d'estimations statiques possibles et leurs niveaux de précision.

2.5.1 Intérêt de la simulation fonctionnelle dans l'estimation

La simulation fonctionnelle avant partitionnement peut apporter différentes informations aidant à l'estimation des performances et donc au partitionnement. En effet, lorsque les spécifications fonctionnelles sont exécutables, il est possible d'ajouter des annotations permettant de connaître la fréquence d'exécution des instructions, des fonctions, ou des tâches du système. Ce sont des techniques couramment utilisées dans le domaine du logiciel (utilisation de "profiler"). Elles permettent, dans ce domaine, de repérer les fonctions qui sont le plus utilisées ou qui sont les plus "gourmandes en temps de calcul". Ce repérage permet de déterminer les fonctions qui doivent être optimisées [49].

Il est aussi possible de relever la fréquence de lecture et d'écriture concernant les variables, et de déterminer le nombre d'itérations des boucles non-déterministes. L'ensemble de ces informations peut être utilisé pour modéliser le fonctionnement du système en valeurs moyennes. Il est aussi possible d'utiliser les valeurs maximales obtenues par simulation pour déterminer les *"chemins critiques"* de l'algorithme.

2.5.2 Synthèse et Estimation du processeur logiciel

2.5.2.1 Les étapes de la synthèse du logiciel

Afin de discuter des différents modèles d'estimation possibles, nous allons tout d'abord décrire le flot de conception d'une application logicielle.

La conception du système logiciel, après les spécifications, correspond à la synthèse.

Cette synthèse débute par un raffinement des spécifications fonctionnelles qui fixent notamment l'ordonnancement et la priorité des différentes tâches sur le microprocesseur. En effet, cet ordonnancement n'est pas forcément pris en compte dans les spécifications : si, par exemple, deux tâches qui ont été spécifiées comme concurrentes doivent être réalisées en logiciel, le microprocesseur qui constitue une ressource critique doit être partagé entre ces deux tâches. Selon le type d'application, différents ordonnancements peuvent être utilisés (statique ou dynamique, préemptif ou non-préemptif). Un système d'exploitation temps réel peut être utilisé, afin de gérer les différentes tâches que le microprocesseur doit exécuter. Tous ces choix constituent une partie de la synthèse du logiciel. Cependant, les contraintes de temps et de sûreté de fonctionnement des applications temps-réel limitent fortement la complexité et le type d'ordonnancement possible.

La deuxième partie de la synthèse correspond à l'écriture de l'algorithme de chaque tâche à l'aide d'un langage de haut niveau comme le C, l'Ada,... Cette écriture, dans le cas où les spécifications sont suffisamment précises, peut être en partie automatisée (génération de code). Le code écrit est alors compilé. La compilation passe par plusieurs étapes parmi lesquelles apparaît la génération du code en langage d'assemblage totalement dédié à un microprocesseur particulier. Enfin, le programme décrit en langage d'assemblage est compilé pour donner le code objet chargé en mémoire.

L'estimation statique porte essentiellement sur la deuxième partie de la synthèse.

Remarque 2.5.1 (Sur la Compilation). *Pour obtenir le code assembleur, le compilateur passe par des étapes intermédiaires bien spécifiques [2]. Durant ces étapes, la plupart des optimisations du code sont effectuées. Il existe des optimisations à différents niveaux. Certaines peuvent être indépendantes du microprocesseur cible et fortement dépendantes du code source, comme par exemple l'élimination de sous-expression commune et la propagation des constantes. D'autres optimisations peuvent être appliquées*

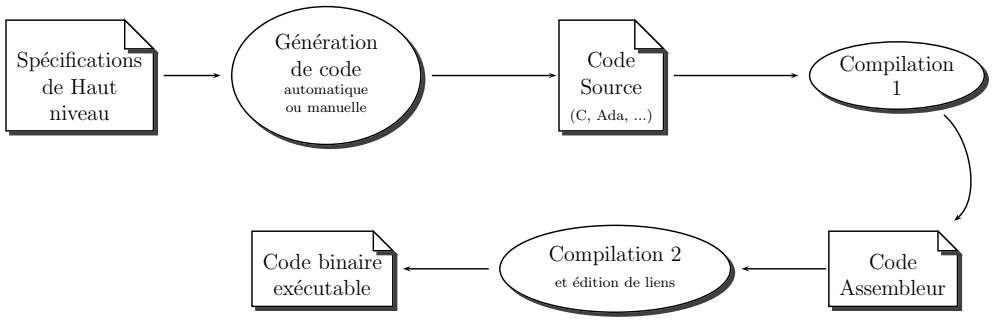


Figure I.2.9: *Flot de synthèse du logiciel*

par rapport à l'architecture notamment à l'utilisation des différentes ressources comme les registres spécifiques (flottants, entiers), l'utilisation du pipeline, ou bien l'utilisation d'une architecture de type VLIW.

L'estimation des différentes mesures peut s'effectuer à plusieurs niveaux de compilation. Cependant nous allons voir que plus l'estimation intervient tardivement dans la synthèse, plus celle-ci sera précise.

2.5.2.2 Les différents niveaux d'estimation possibles.

Estimation au niveau du code source

La première estimation qui peut être faite correspond au niveau fonctionnel. À ce niveau, aucun microprocesseur n'a encore été choisi. Si l'outil de spécification utilisé initialement dans le cycle de conception est suffisamment précis (ou les spécifications peuvent être aisément affinées), on peut utiliser ces spécifications en associant un temps de calcul et une taille mémoire à chaque atome du système. Cette méthode requiert alors la génération d'une base de données pour les différents atomes possibles. Cependant l'addition du temps de calcul de chaque atome n'est que très rarement égal au temps de calcul de l'association des atomes. Ce problème est induit par les différentes transformations que subit le code durant les étapes de compilation.

Une possibilité du même ordre de précision consiste à générer, à partir des spécifications, le code de haut niveau puis d'analyser le programme en associant, comme précédemment, un temps de calcul particulier à chaque instruction. Ce type d'estimation a d'ailleurs été réalisé dans le cadre du projet POLIS. Les résultats obtenus indiquent des erreurs sur les estimations allant jusqu'à 20% pour des codes dont la structure est assez régulière.

Estimation au niveau du code assembleur

Une autre possibilité pour l'estimation du logiciel est de compiler le langage de haut niveau en un code assembleur générique. Ce dernier n'est associé à aucun microprocesseur spécifique, mais il reproduit les instructions communément utilisées dans les microprocesseurs comme l'addition, la soustraction, la multiplication, le chargement d'une instruction vers un registre, etc. Ce compilateur doit pouvoir réaliser certaines optimisations, toujours indépendamment d'un microprocesseur donné.

Le code assembleur générique est ensuite analysé par un outil auquel on spécifie un fichier dit de "technologie". Dans ce fichier, chaque instruction générique est étiquetée d'une information de temps de calcul et de taille mémoire, ces informations étant liées à un microprocesseur particulier.

Cet outil comporte des intérêts et des inconvénients. L'intérêt principal est qu'il est possible de faire l'analyse pour un nouveau microprocesseur de manière rapide et simple. Il suffit en effet de créer le fichier de "technologie" contenant les informations sur la taille mémoire et de temps de calcul de chaque instruction. L'inconvénient principal est lié au fait que le compilateur est totalement dissocié du processeur cible. Or il existe une étroite relation entre le jeu d'instruction du microprocesseur et le compilateur dédié qui permet de faire des optimisations particulières. De plus le code assembleur générique ne peut contenir que les instructions communes aux différents microprocesseurs, ce qui exclut les instructions qui font la particularité d'un microprocesseur. Cette technique sera par exemple peu efficace dans le cas de microprocesseurs CISC [27].

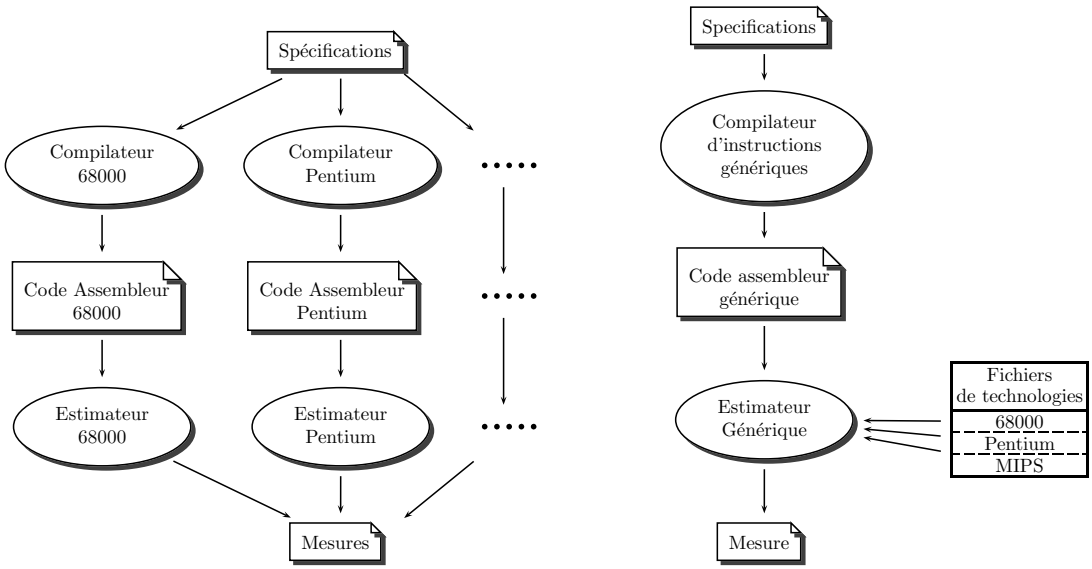


Figure I.2.10: *Deux types d'estimateurs possibles au niveau assembleur*

La deuxième possibilité est d'utiliser le compilateur dédié à chaque microprocesseur afin d'optimiser le code assembleur. Puis un estimateur associé au langage assembleur du processeur cible est utilisé pour déterminer les mesures de temps de calcul et de taille mémoire. L'inconvénient principal est bien sur l'accès aux différents compilateurs et la construction d'un estimateur dédié.

Nous pouvons constater qu'un outil d'estimation peut être utilisé à chaque étape de la synthèse. Cependant, plus l'estimation est fait tardivement dans la synthèse du logiciel, plus les outils sont complexes et difficiles à mettre en œuvre. En contre partie, la précision des estimations augmente avec la prise en compte de plus en plus importante des technologies utilisées.

Le choix de l'estimateur utilisé devra donc être adapté à la précision recherchée et à l'exploitation des mesures. Il est notamment préférable d'augmenter la précision au fur et à mesure que l'on avance dans le cycle de conception afin de valider au mieux les choix antécédents.

2.5.3 Synthèse et Estimation des performances du processeur matériel

La synthèse d'un système matériel peut suivre différentes méthodes. La méthode dite ascendante (*bottom-up*) se base sur un ensemble de modules caractérisés par leurs fonctionnalités et leurs performances. La conception se fait alors par la construction

d'une architecture constituée de ces modules et respectant les contraintes imposées par le cahier des charges.

La conception peut se faire aussi selon une méthode descendante (*top-down*), qui est basée sur le raffinement progressif des spécifications identifiées à partir du cahier des charges. Le co-design se base sur une conception descendante en utilisant les spécifications fonctionnelles décrites au niveau système comme point de départ.

Dans le cadre des processeurs matériels, la synthèse est composée de deux étapes. La première correspond à la synthèse architecturale, la seconde est nommée synthèse logique.

2.5.3.1 La synthèse architecturale

La synthèse architecturale est aussi nommée synthèse comportementale ou synthèse de haut niveau. Le but de cette synthèse consiste à transformer la description fonctionnelle du système en une architecture numérique appelée communément : description au niveau registre (RTL[4]). La description RTL représente le fonctionnement du système en utilisant les structures numériques fondamentales que sont les ALU, les multiplexeurs, les décodeurs, les registres, etc.

Une méthodologie de co-design cherche à inclure des outils de synthèse architecturale automatique ou semi-automatique. Ce type d'outil retranscrit une spécification fonctionnelle en l'association de deux type d'entités : le chemin de données et l'unité de contrôle. Par analogie avec les microprocesseurs, le chemin de données correspond à l'ensemble {mémoire de données + registres de données + ALU} et l'unité de contrôle a le même rôle que l'ensemble {unité de commande + registres d'adresses + mémoire programme}.

Le chemins de données est donc composé de trois types de composants :

1. les unités de stockage,
2. les unités fonctionnelles comme les ALU, les comparateurs, multiplieurs, etc,
3. les unités d'interconnexions.

L'unité de contrôle est une machine à états finis qui séquence l'exécution des opérations dans le chemin de données.

La synthèse de l'unité de contrôle et du chemin de données s'effectue en deux étapes : l'ordonnancement et l'allocation des ressources [74].

- L'ordonnancement permet de déterminer le séquencement des calculs. Une séquence pouvant comporter plusieurs opérations en parallèle.
- L'allocation des ressources sélectionne le type et la quantité des modules matériels issus d'une bibliothèque, et les met en correspondance avec chaque opération. Elle accomplit également les allocations de registres et des connexions.

La synthèse se base sur l'optimisation d'une fonction coût qui est une combinaison du coût des ressources matérielles, du cycle d'exécution, et du nombre d'étapes de contrôle.

2.5.3.2 La synthèse logique

La synthèse logique consiste à transformer une description RTL en un ensemble de portes logiques tout en déterminant une interconnexion optimale. Cette synthèse est totalement dépendante du composant sur lequel seront implantées les fonctions que l'on veut réaliser.

Une phase d'optimisation logique cherche à réécrire les équations booléennes (pour un circuit combinatoire) ou à minimiser le nombre d'états (pour un circuit séquentiel) d'un bloc décrit au niveau RTL. Une phase d'allocation technologique détermine la

[4]Register Transfer Level

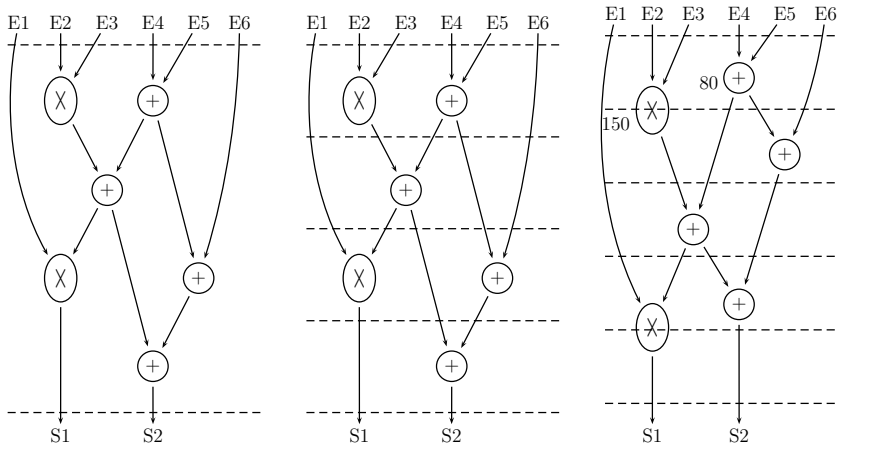


Figure I.2.11: *Influence des paramètres de la synthèse architecturale dans les métriques du système*

meilleure structure à base de cellules standards dans une technologie donnée ou la meilleure programmation d'un circuit FPGA.

Cette synthèse s'effectue en essayant d'optimiser la surface de silicium utilisée et les délais ou temps de propagation des signaux entre les différentes portes logiques.

2.5.3.3 L'estimation des performances

Après avoir défini les différentes étapes de la synthèse du processeur matériel, on peut définir les différents niveaux d'estimations possibles.

Le premier niveau correspond au niveau des spécifications fonctionnelles, après avoir construit le graphe de flots de contrôle et de données. Puisqu'aucune synthèse n'a encore été exécutée, il n'est pas possible d'estimer correctement le nombre de portes logiques qui seront utilisées. Cela ne peut être fait qu'en limitant l'exploration de la synthèse architecturale (parallélisation maximale des calculs ou minimisation des ressources). Dans ce cadre plus restreint, une analyse du CDFG peut permettre d'estimer le nombre de ressources utilisées et donc d'estimer approximativement la taille résultante.

L'estimation après la synthèse architecturale semble, quant à elle, relativement précise, en considérant une bibliothèque de composants fondamentaux caractérisés par le nombre de portes qu'ils utilisent et leurs temps de calculs. Il manque cependant l'ensemble des temps de propagation associés au routage.

L'estimation après synthèse logique correspond à l'estimation la plus précise et donne alors pour le temps de calcul, le temps de propagation du chemin critique.

Actuellement, l'estimation des performances après synthèse architecturale ou synthèse logique est couramment utilisée dans les outils commerciaux et ne pose pas de problèmes particuliers. La synthèse architecturale est, par contre, loin d'être un domaine maîtrisé.

2.6 La co-simulation

Après le partitionnement, nous avons vu que les synthèses du logiciel et du matériel sont constituées de plusieurs étapes. Le résultat de ces étapes correspond soit à des

spécifications plus raffinées sur le fonctionnement du système, soit à un code dans un langage intégrant des informations technologiques de plus en plus précises. Il est d'ailleurs préférable de valider chaque étape de la synthèse car, dans le cas contraire, des erreurs peuvent s'introduire très tôt et se propager durant les autres étapes de la synthèse. La détection tardive demande alors des itérations et des investigations plus longues pour déterminer la source de l'erreur.

Pour valider simultanément les étapes de la synthèse du logiciel et de la synthèse du matériel, on utilise des techniques dites de co-simulation qui consistent à simuler conjointement la partie matérielle et la partie logicielle du système.

Un système mixte logiciel/matériel étant composé de microprocesseurs et d'ASIC's, la co-simulation consiste à simuler le logiciel sur un modèle plus ou moins précis du microprocesseur et à coordonner cette simulation avec la simulation d'un ASIC (le plus souvent modélisé en HDL[5]). Selon les modèles de simulation utilisés lors d'une co-simulation, on peut valider le fonctionnement mais aussi faire des estimations dynamiques sur les performances du système comme le temps de calcul et l'utilisation dynamique de la mémoire.

2.6.1 La co-simulation dans le cycle de conception

La co-simulation peut être utilisée pour vérifier différentes étapes de la synthèse (cf. figure I.2.12).

Le premier niveau correspond à la simulation conjointe des spécifications du processeur matériel décrites en HDL et les spécifications du logiciel écrites en langage de haut niveau. Cette co-simulation permet de faire une vérification fonctionnelle des deux parties et de leurs communications. À ce stade, il n'est pas encore possible de déterminer les temps de calcul des différentes parties.

Une co-simulation de deuxième niveau peut ensuite être exécutée pour valider le bon fonctionnement du processeur matériel après la synthèse architecturale. La simulation s'effectue alors avec une précision temporelle correspondant au cycle d'horloge du processeur matériel.

À partir de ce niveau, le logiciel peut soit être adapté, soit être utilisé avec un modèle de microprocesseur permettant au minimum de reproduire son interface externe (les bus de données, d'adresse, les interruptions,...). Différents modèles de microprocesseurs peuvent répondre à ce premier critère et donner d'autres informations avec des niveaux de granularité différents (instruction assembleur, cycle d'horloge, nanoseconde,...).

Le dernier niveau de co-simulation utilise une description du processeur matériel par des portes logiques, en considérant les propriétés associées à une technologie d'intégration particulière (type de FPGA, d'EPLD, etc). Ce niveau permet alors de vérifier le bon comportement du système en tenant compte de certains impératifs technologiques comme les temps de propagation.

Un environnement de co-simulation nécessite aussi l'utilisation de mécanisme de communication entre les différentes applications et l'utilisation de modèles de microprocesseurs avec différents niveaux de précision. Or ces deux aspects sont actuellement très peu développés et sont donc très loin d'être génériques.

2.6.2 Intérêt de la co-simulation dans la commande des systèmes électriques

Actuellement, la conception d'une architecture de commande pour les systèmes électriques nécessite généralement l'utilisation conjointe de composant logiques programmables et de microprocesseurs. Le CLP est alors utilisé pour effectuer les tâches ayant une contrainte temporelle dure mais dont les calculs sont relativement simples et le

[5]Hardware Description Language

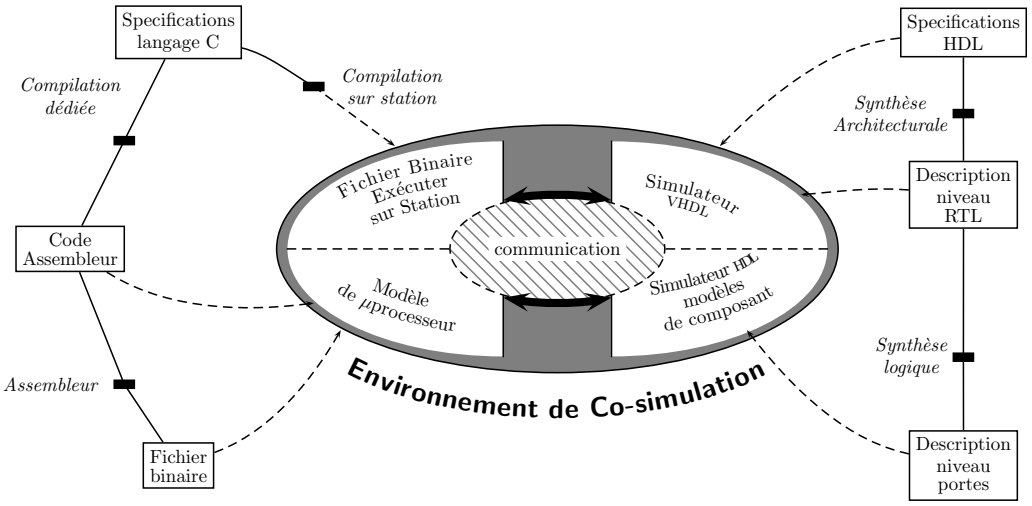


Figure I.2.12: *Les différentes co-simulation possibles lors de la synthèse du système*

microprocesseur est utilisé pour traiter des algorithmes et des opérations arithmétiques plus complexes.

Cependant, comme l'indique la figure I.2.13, grâce à l'évolution des technologies, les composants de type FPGA permettent d'intégrer des calculs de plus en plus complexes et les microprocesseurs ont des cycles d'horloge de plus en plus faibles. La répartition des tâches et le choix des composants constituant la commande d'un système électrique sont donc des problèmes typiques auxquels peut répondre une méthodologie de co-design.

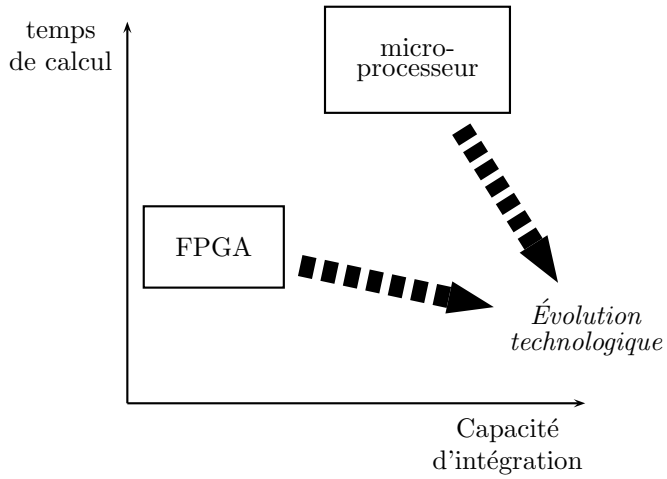


Figure I.2.13: *Évolution technologique des composants électroniques*

Dans le but d'établir un environnement de co-design dédié à la conception de commande pour les systèmes électriques, nous avons choisi de commencer par explorer les problèmes associés à la co-simulation. Cette première exploration permettra alors de déterminer les besoins en développement et l'intérêt de ce domaine, mais aussi quels outils pourront être mis en place en amont de la co-simulation.

La co-simulation semble en effet apporter une aide dans la conception sous deux formes différentes.

Premièrement, la co-simulation permet de valider la synthèse conjointe du logiciel et du matériel. Une telle simulation permet l'accès à des grandeurs qu'il est difficile d'acquérir sur un prototype réel notamment sur les CLP. La co-simulation peut donc accélérer le développement d'une application.

Durant la synthèse, des choix sur le type et la taille des variables doivent être faits (8, 16, 32 bits, entier, flottants). Dans le cadre de la commande, ces choix peuvent influer sur les performances. La co-simulation peut permettre d'évaluer rapidement l'impact de ces choix.

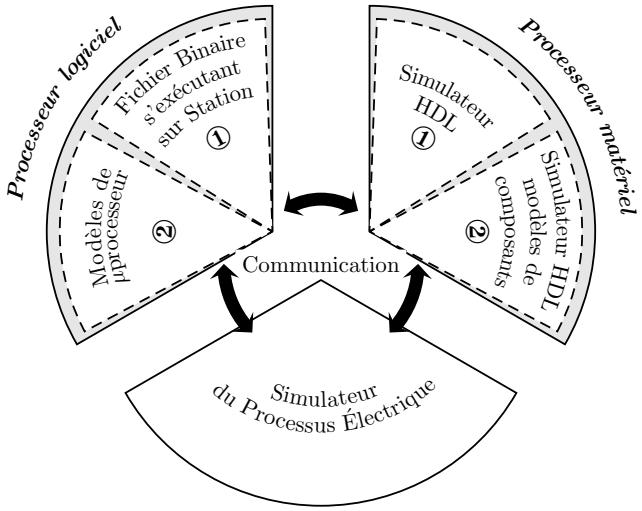


Figure I.2.14: *Environnement de co-simulation adapté à la commande des systèmes électriques*

Deuxièmement, la co-simulation peut permettre d'estimer les temps de calculs et l'utilisation dynamique des mémoires. Ces résultats peuvent alors confirmer ou réfuter le choix des composants avant la réalisation du prototype réel et aider à valider un partitionnement particulier.

Pour cela, la co-simulation doit inclure un troisième domaine qui correspond à la simulation du processus à commander. La figure (I.2.14) résume les éléments nécessaire pour effectuer la co-simulation d'un dispositif de commande d'un système électrique. Dans un tel environnement, on peut en effet effectuer une co-simulation de premier niveau à l'aide d'un simulateur de langage HDL, d'un compilateur de programme C, et du simulateur du processus électrique commandé. Si l'ensemble des systèmes de communications sont mis en place, on peut aussi envisager d'utiliser deuxième niveau de co-simulation qui utilise des modèles de microprocesseurs et un simulateur HDL avec des modèles de composants numériques. Ce deuxième niveau permet d'intégrer les contraintes technologiques et de voir l'impact de ceux-ci sur la commande du système.

2.7 Conclusion

En parcourant les différentes étapes de la conception d'un système mixte logiciel/-matériel, nous avons présenté les différents outils qui pouvaient aider le concepteur dans sa tâche. Nombre de ces outils semblent avoir un intérêt pour permettre d'intégrer au mieux un système de commande pour les dispositifs électriques. Cependant, plusieurs domaines ne sont pas encore au point ou sont très limités comme la recherche de partitionnement automatique.

De plus, il n'existe pas d'environnement permettant de faire l'ensemble du cycle de conception et qui soit adapté à la commande des systèmes électriques. Nous nous sommes attardés dans ce chapitre sur deux étapes particulières que sont la spécification d'une part et l'estimation des performances d'autre part. En effet, il semble que le noeud du problème pour établir une méthodologie du co-design soit de relier ces deux extrêmes. Il faut donc rechercher les modèles de spécification les plus adaptés aux systèmes à concevoir tout en ayant une forme la plus simple possible, afin de pouvoir construire des outils d'analyse puissants permettant d'estimer rapidement les performances. Cette partie représente la partie haute du cycle de conception, lorsque les différents choix ont été réalisés, il faut aider le concepteur à effectuer la synthèse de ses spécifications. Dans ce domaine, la co-simulation semble un outil intéressant puisqu'elle peut lui permettre de détecter des erreurs bien avant la validation expérimentale. Elle peut aussi lui donner des informations précises concernant les performances, ce qui peut confirmer ses choix ou bien lui permettre de revenir en arrière dans le cycle de conception avant d'avoir réaliser un quelconque prototype du système.

Nous allons tenter de vérifier que cela est applicable dans le cas de la commande des systèmes électriques, en adaptant la co-simulation à nos besoins propres. Il nous faut pour cela développer les outils de co-simulation en prenant en compte le système électrique à commander. C'est d'ailleurs dans la deuxième partie de ce mémoire que nous présentons les différentes approches effectuées sur la co-simulation pour l'intégration des lois de commande dédiées aux systèmes électriques.

Deuxième partie

La co-simulation dans le cadre de la commande des systèmes électriques

Chapitre 3

La co-simulation : Cas des microprocesseurs

3.1 Introduction.

À l'heure actuelle, dans un dispositif de commande numérique, le composant clé correspond au microprocesseur qui peut avoir en charge plusieurs tâches, qui peuvent aller de la régulation à l'interface homme/machine. De plus, dans le cadre des systèmes électriques, les modules de régulation possèdent des constantes de temps extrêmement faibles et très peu modifiables. Ce qui nous conduit donc à des systèmes temps réel ayant des contraintes temporelles sévères.

Le choix du microprocesseur est alors déterminant, et il est souvent difficile d'évaluer ses performances réelles avant les essais expérimentaux, notamment, les temps de calculs associés aux différentes tâches.

Ce constat nous a amené, dans un premier temps, à rechercher dans la co-simulation une solution pour estimer les performances temporelles.

3.2 Différents modèles de microprocesseur.

Actuellement, il existe plusieurs techniques permettant de simuler le comportement du microprocesseur. La variété de ces techniques est due au fait que chacune répond à des besoins différents. Nous allons présenter les principales techniques existantes [66, 15], afin d'identifier les modèles les plus appropriés aux différents niveaux de co-simulation désirés pour nos applications.

3.2.1 Simulation matérielle.

Le modèle *"matériel"* correspond à la modélisation la plus fine du microprocesseur. Elle consiste à reproduire le fonctionnement logique des différents modules composant le microprocesseur comme l'unité de commande, l'unité arithmétique et logique (UAL), etc. C'est un modèle qui est généralement décrit en utilisant un langage de description matériel (comme le VHDL). Ces modèles sont souvent développés par les concepteurs du microprocesseur pour leurs besoins propres. Ils sont cependant rarement accessibles au grand public pour des raisons de propriété intellectuelle, les seuls modèles accessibles sont souvent associés à des microprocesseurs relativement vieux. Étant très détaillé, ces modèles sont très gourmands en temps de calcul et rendent compte de détails qui s'avèrent inutiles par rapport à nos objectifs.

3.2.2 Simulation du jeu d'instructions.

Le modèle de simulation matériel vu au paragraphe 3.2.1 possède une précision de l'ordre de la nanoseconde, voire de la période d'horloge du microprocesseur. Cependant, ce dernier peut aussi être modélisé avec une précision correspondant à l'évolution du compteur programme, ce qui correspond à l'exécution d'un cycle mémoire ou d'une instruction. Nous avons référencé deux types de simulateur à ce niveau de précision.

3.2.2.1 Simulation interprétée.

Le simulateur du jeu d'instructions utilisant la technique dite de simulation interprétée est le plus couramment utilisé. En général, ce type de modèle fait partie des outils de développement distribués également avec le compilateur, l'assembleur, l'éditeur de liens et le débogueur, fournis par le constructeur.

Ce modèle interprète chaque instruction du microprocesseur et reproduit fonctionnellement l'opération à l'aide de registres virtuels. Il doit être capable, entre autres, de charger le code exécutable, censé être chargé sur la cible, dans une mémoire virtuelle puis réaliser les phases de décodage et d'exécution.

Ce type de modèle apparaît, actuellement, comme un outil essentiel dans le cadre du co-design et notamment dans l'étape de synthèse puisqu'il permet de valider le fonctionnement du code [44]. Cependant, il existe très peu de modèles "ouverts" permettant une connexion avec d'autres logiciels afin de pouvoir les exploiter en co-simulation. De plus, le temps d'interprétation de chaque instruction n'est pas négligeable, ce qui entraîne un temps de simulation relativement important.

3.2.2.2 Simulation compilée.

La simulation compilée est une technique beaucoup moins répandue mais plus efficace en terme de temps de simulation [75, 39]. Le principe de cette technique consiste à retranscrire une instruction du microprocesseur cible en une instruction identique pour la station de travail ou, si elle n'existe pas, une série d'instructions reproduisant la même fonctionnalité. Ainsi, les opérations de chargement et de décodage d'instructions réalisées dans la simulation interprétée ne sont plus nécessaires car elles sont déportées dans la fonction de retranscription. La simulation du microprocesseur est donc fortement accélérée.

3.2.3 Annotation du code source.

L'annotation du code source est une méthode qui ne tient pas compte du fonctionnement interne du microprocesseur [69]. Seul le temps de calcul est estimé et le programme source est exécuté après une simple compilation sur une station de travail. Il s'agit de la technique énoncée dans la section 2.5.2.2 page 38 du chapitre 2 qui consiste à exécuter le code source en rajoutant des instructions et des variables permettant de calculer la durée d'exécution du programme. Ce type de simulation est peu précis et ne tient pas, ou très peu, compte des optimisations produites par le compilateur en fonction de l'architecture de la cible, comme par exemple le pipeline.

3.2.4 Modélisation au niveau bus.

Ce modèle représente le fonctionnement du microprocesseur au niveau de ses bus externes comme le bus de données ou le bus d'adresse. Il est utilisé pour réaliser des bancs de test sur les interfaces matérielles avec le microprocesseur. Il ne permet pas de faire des estimations sur les performances du microprocesseur lui-même, mais permet

une validation de l'environnement proche du microprocesseur comme les accès mémoire ou les CAN et CNA [33].

Parmi les différentes méthodes énumérées, les simulateurs de jeux d'instructions semblent présenter le meilleur compromis entre précision et durée de simulation pour être intégré dans un outil de co-simulation (utilisé pour la synthèse et la vérification). En effet, la simulation du jeu d'instructions permet d'estimer le temps de calcul de manière relativement précise (si l'on connaît le temps d'exécution de chaque instruction), et le niveau de représentation permet de prendre en compte les problèmes d'optimisation associés à l'architecture même du microprocesseur (niveau assembleur).

Après avoir déterminé le type de simulateur de microprocesseur qui semble le plus approprié, il nous faut maintenant évaluer l'intérêt d'un tel outil dans le cadre de la commande des systèmes électriques. Cette évaluation nécessite alors de mettre en place un environnement minimal permettant d'effectuer des tests de co-simulation.

3.3 Simulation interprétée du jeu d'instructions. Étude de cas : le 68332.

Les choix pour mettre en place un environnement minimal de co-simulation se sont portés, dans un premier temps, sur l'utilisation d'un *simulateur du jeu d'instructions du micro-contrôleur 68332* de Motorola associé à un simulateur de type système nommé SABER très utilisé dans la communauté du génie électrique. L'utilisation de ces deux logiciels va permettre de réaliser la co-simulation d'une commande numérique de machine à courant continu. Étant donné qu'un prototype matériel d'une telle application est disponible, la co-simulation pourra être confrontée à des résultats expérimentaux.

À l'aide de la co-simulation, nous chercherons alors à estimer les temps de calcul de la commande et à comparer les résultats selon le type de codage utilisé, permettant ainsi d'optimiser le format de représentation des variables utilisées dans l'algorithme de contrôle.

3.3.1 Système co-simulé.

3.3.1.1 Présentation de la maquette expérimentale.

Le schéma de principe du dispositif est donné par la figure II.3.1. L'étage de puissance est composé d'une source de tension continue, d'un hacheur quatre quadrants, d'un moteur et d'une génératrice débitant sur un plan de charge. Cet étage est équipé de capteurs et d'actionneurs reliés aux différents périphériques du calculateur.

Le dispositif de commande est construit autour du micro-contrôleur 68332. Il comprend différentes interfaces permettant l'échange d'informations avec le procédé et l'utilisateur.

Les périphériques du dispositif de commande

Parmi les principaux périphériques du dispositif de commande représentés sur la figure II.3.1, on distingue :

- Un temporisateur programmable (PTM[1]) comportant 3 ports, utilisé pour générer les ordres de commande dédiés au hacheur et les horloges temps réel.
- Une interface parallèle (PIA[2]) 8 bits pour acquérir la vitesse du moteur calculée par un fréquencemètre.

[1] Programmable Timer Module
[2] Peripheral Interface Adaptator

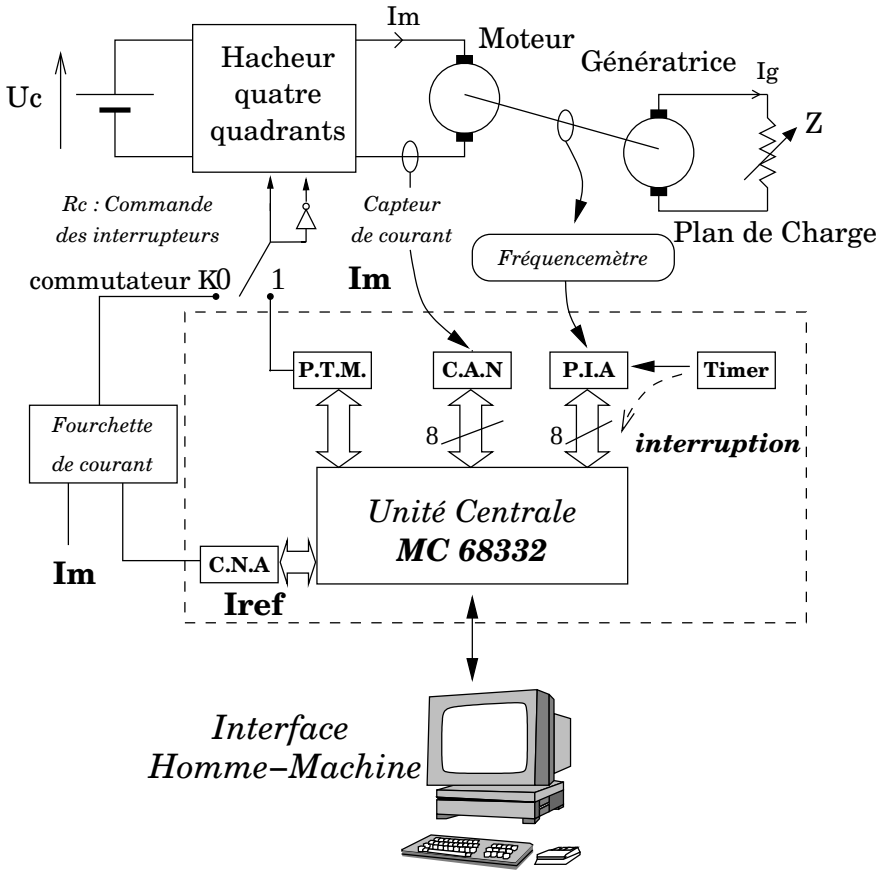


Figure II.3.1: *Schéma de principe de la maquette expérimentale.*

- Un convertisseur analogique-numérique (CAN) 8 bits, signé ayant un temps de conversion de 2.5 μs, et utilisé pour acquérir la valeur moyenne du courant machine I_m obtenue après filtrage.
- Un convertisseur numérique-analogique pouvant être utilisé pour générer un courant de consigne I_{ref}, dans le cas d'un contrôle par fourchette.

Le micro-contrôleur 68332

Il s'agit d'un micro-contrôleur 32 bits, basé sur l'architecture du 68000 (16 bits de données et 24 bits d'adresses). Les échanges avec les circuits périphériques peuvent s'effectuer de manière synchrone ou asynchrone. Il comprend en interne de la mémoire vive (256k), plusieurs modules spécifiques et quelques interfaces d'entrée/sortie. Les caractéristiques essentielles de ce micro-contrôleur sont les suivantes :

- Horloge programmable de 131 kHz à 20,972 MHz,
- Module Central Processing Unit (CPU) - l'unité centrale est basée sur une architecture 32 bits et présente les mêmes ressources matérielles que le 68000. Le code de programmation est compatible avec celui du 68010,
- Module System Integration Module (SIM) - il permet de configurer différents paramètres, tel que la fréquence d'horloge, les chips selects, les interruptions, etc...
- Module Time Processor Unit (TPU) - 16 temporisateurs programmables 16 bits pouvant servir de générateurs de signaux carrés programmables et fonctionner en fréquencemètre/périodemètre. On utilisera ce module pour mesurer le temps d'exécution des programmes.
- Module Queue Serial Module (QSM) - 2 entrées/sorties séries synchrones et asynchrones, exploitées en particulier pour le téléchargement des programmes via un

ordinateur hôte de type PC.

Sur le dispositif expérimental, ce micro-contrôleur est utilisé comme un microprocesseur, à savoir : la partie CPU et le module SIM pour la configuration du système. Les périphériques externes sont connectés de manière classique par l'intermédiaire des bus de données, d'adresses et de contrôle.

À partir de cette maquette, on peut réaliser les configurations suivantes :

- la régulation numérique du courant dans la machine,
- la régulation numérique de vitesse et la régulation du courant peut être relayée à une régulation de nature analogique, en positionnant le commutateur K à 0 (cf. figure II.3.1),
- la régulation de vitesse et la régulation de courant peuvent être toutes les deux numériques (commutateur K à 1).

3.3.1.2 Principe de la commande.

Modélisation

Afin d'expliquer les principes de cette commande, nous présentons brièvement la modélisation de l'étage de puissance. Cette étage de puissance est constitué du hacheur 4 quadrants, de la machine à courant continu, et de la charge composée d'une génératrice à courant continu débitant sur un plan de charge (résistance Z sur la figure II.3.1).

Le modèle est basé sur les hypothèse suivantes :

- les machines à courant continu ont un flux constant sur toute la plage de fonctionnement,
- la fréquence de fonctionnement du hacheur est suffisante pour considérer comme négligeable l'influence de l'ondulation de courant ΔI.

Selon ces hypothèses et en notant V_m, I_m les valeurs moyennes en tension et en courant aux bornes de l'induit du moteur, on obtient les équations (II.3.1) modélisant le système.

Pour le moteur :

$$V_m - E = L.\frac{dI_m}{dt} + R.I_m \qquad avec\ E = K.\Omega \tag{II.3.1a}$$

$$Cem - Cr = J.\frac{d\Omega}{dt} + f_0.\Omega \qquad avec\ Cem = Kem.I_m \tag{II.3.1b}$$

Pour la génératrice :

$$Cr = Kem.Ig = Kem.\frac{K.\Omega}{R+Z} = f_1.\Omega \tag{II.3.1c}$$

Pour le hacheur :

$$V_m = (2.\alpha - 1).U_c = 2.\beta.U_c \tag{II.3.1d}$$

$$-0.5 \leq \beta \leq 0.5\ et\ 0 \leq \alpha \leq 1 \tag{II.3.1e}$$

α représente le rapport cyclique du hacheur. Ce rapport cyclique correspond à la valeur moyenne du signal de commande envoyé aux interrupteurs du hacheur, soit $\alpha = \frac{t_{on}}{T_d}$. Comme il s'agit d'un hacheur 4 quadrants, lorsque R_c vaut 1, on applique aux

bornes de la machine la tension U_c et lorsque R_c vaut 0, on lui applique la tension $-U_c$. β est défini comme suit :

$$\beta = \alpha - \frac{1}{2} \tag{II.3.2}$$

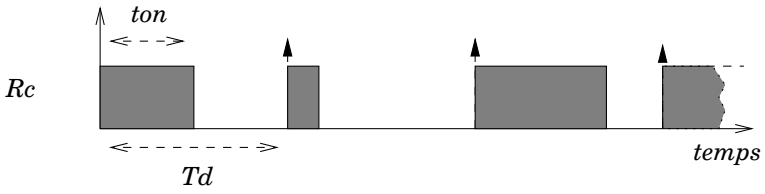


Figure II.3.2: *Signal de commande des interrupteurs du hacheur.*

Stratégie de commande

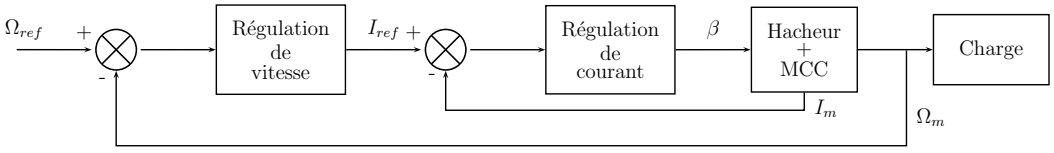
Les équation précédentes montrent que pour contrôler la vitesse, il faut agir sur le couple (Cem) et donc contrôler le courant moyen I_m. Les équations étant couplées, la commande de la machine à courant continu utilise l'hypothèse de la séparation des modes :

On suppose que la constante de temps mécanique du moteur $\tau_m = \frac{J}{f0+f1}$ (eq. II.3.1b) est très grande devant la constante de temps électrique $\tau_e = \frac{L}{R}$ (eq. II.3.1a)

$$\boxed{\tau_e << \tau_m}$$

1. ainsi, la vitesse est considérée comme constante lors de la synthèse du régulateur de courant,
2. le couple électromagnétique Cem est considéré comme égal à sa référence lors de la synthèse du régulateur de vitesse.

Ces hypothèses permettent d'utiliser une structure de boucles imbriquées (fig. II.3.3).



$\beta = \alpha - \frac{1}{2}$ et α : Rapport cyclique

Figure II.3.3: *schéma bloc de la régulation de vitesse par séparation des modes.*

Dans un premier temps, seule la boucle de régulation numérique de courant est effectuée sur le microprocesseur, ce qui correspond à une commande en couple du moteur.

Le cahier des charges de la régulation de courant étant le suivant :

- erreur statique nulle en régime permanent
- bonne dynamique en régime transitoire
- pas de dépassement lors d'un échelon de référence pour éviter le surdimensionnement du hacheur en courant

L'implantation numérique du PI a été réalisée en utilisant la théorie des systèmes continus puis en approximant par la méthode des rectangles la variable de Laplace $p \rightarrow \frac{1-z^{-1}}{T_e}$. La valeur de β étant par définition limitée entre -0.5 et 0.5, cette limitation

est prise en compte dans le régulation de manière dynamique, ce qui nécessite des calculs supplémentaires lorsque la valeur de β calculée de manière classique dépasse les limitations.

En dehors des cas de saturation, on a le calcul suivant :

$$\int_0^{k.T_e} \varepsilon(t).dt = \int_0^{(k-1).T_e} \varepsilon(t).dt + \frac{T_e}{T_i}.\varepsilon(k.T_e) \tag{II.3.3}$$

$$\beta = K_p.\varepsilon(k.T_e) + \int_0^{k.T_e} \varepsilon(t).dt \tag{II.3.4}$$

Lorsqu'il y a saturation on a alors :

$$si\ \beta > \beta_{max}$$

$$\beta = \beta_{max} \tag{II.3.5}$$

$$\varepsilon(k.T_e) = \frac{\beta_{max} - \int_0^{(k-1).T_e} \varepsilon(t).dt}{\frac{T_e}{T_i} + K_p} \tag{II.3.6}$$

$$\int_0^{k.T_e} \varepsilon(t).dt = \int_0^{(k-1).T_e} \varepsilon(t).dt + \frac{T_e}{T_i}.\varepsilon(k.T_e) \tag{II.3.7}$$

Ce calcul, qui consiste à réajuster le terme intégral, permet de garder la linéarité du correcteur lors de la saturation de β [64].

Le programme de régulation est activé en temps réel avec une période égale à T_e, les fonctions assurées par le micro-contrôleur sont données sur la figure II.3.4. Le port 3 du PTM génère le signal de commande R_c en fonction de la valeur de β.

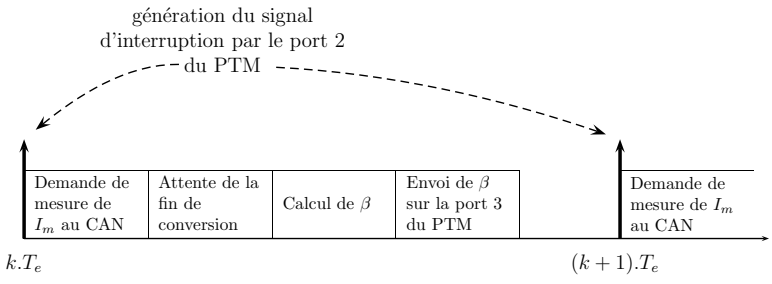


Figure II.3.4: *les différentes étapes dans la régulation de courant effectuée par le 68332.*

3.3.2 Un premier environnement de co-simulation.

Afin de valider notre régulation de courant, nous souhaitons utiliser un environnement de co-simulation intégrant un simulateur du microprocesseur 68332 et un simulateur de type système tel que SABER afin de simuler l'étage de puissance. Les parties à simuler par chaque simulateur sont représentées sur la figure II.3.5.

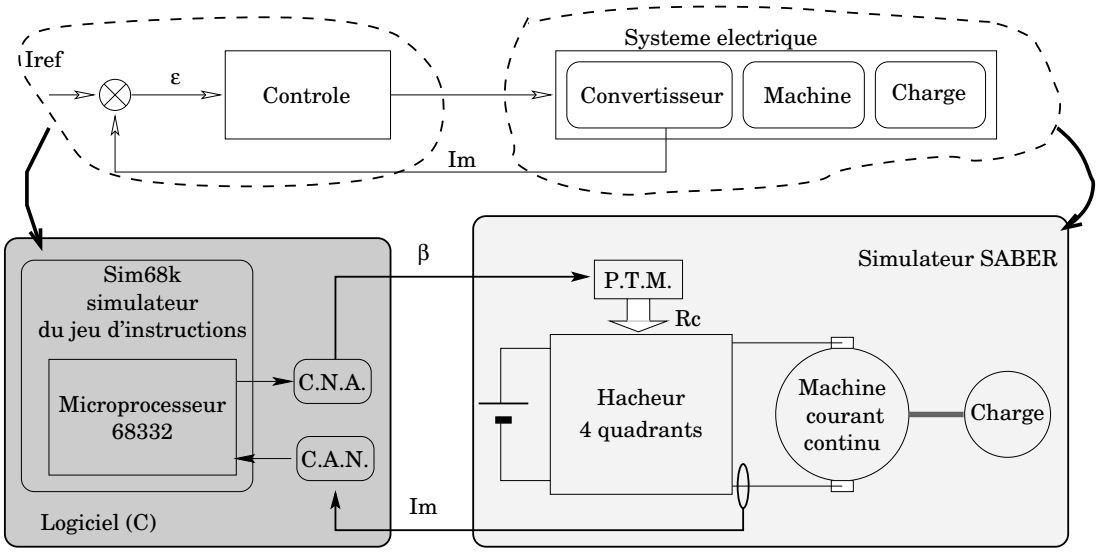
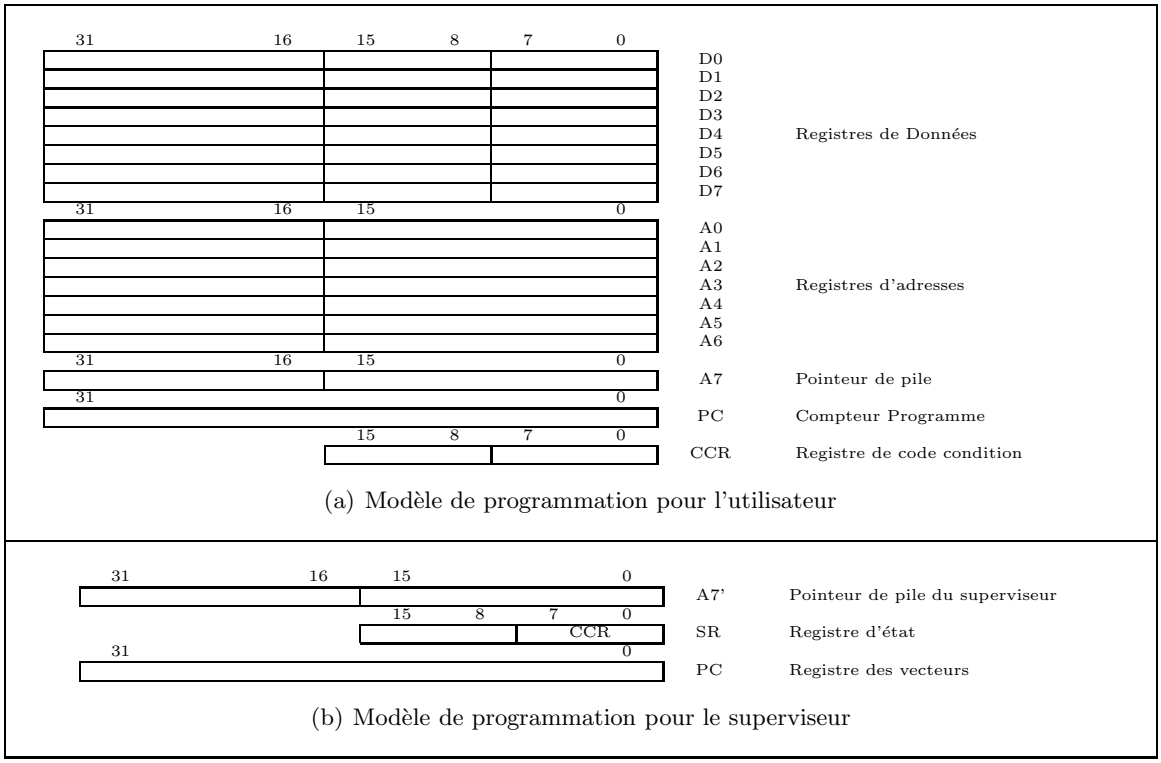


Figure II.3.5: *Un premier environnement de co-simulation recherché.*



(a) Modèle de programmation pour l'utilisateur

(b) Modèle de programmation pour le superviseur

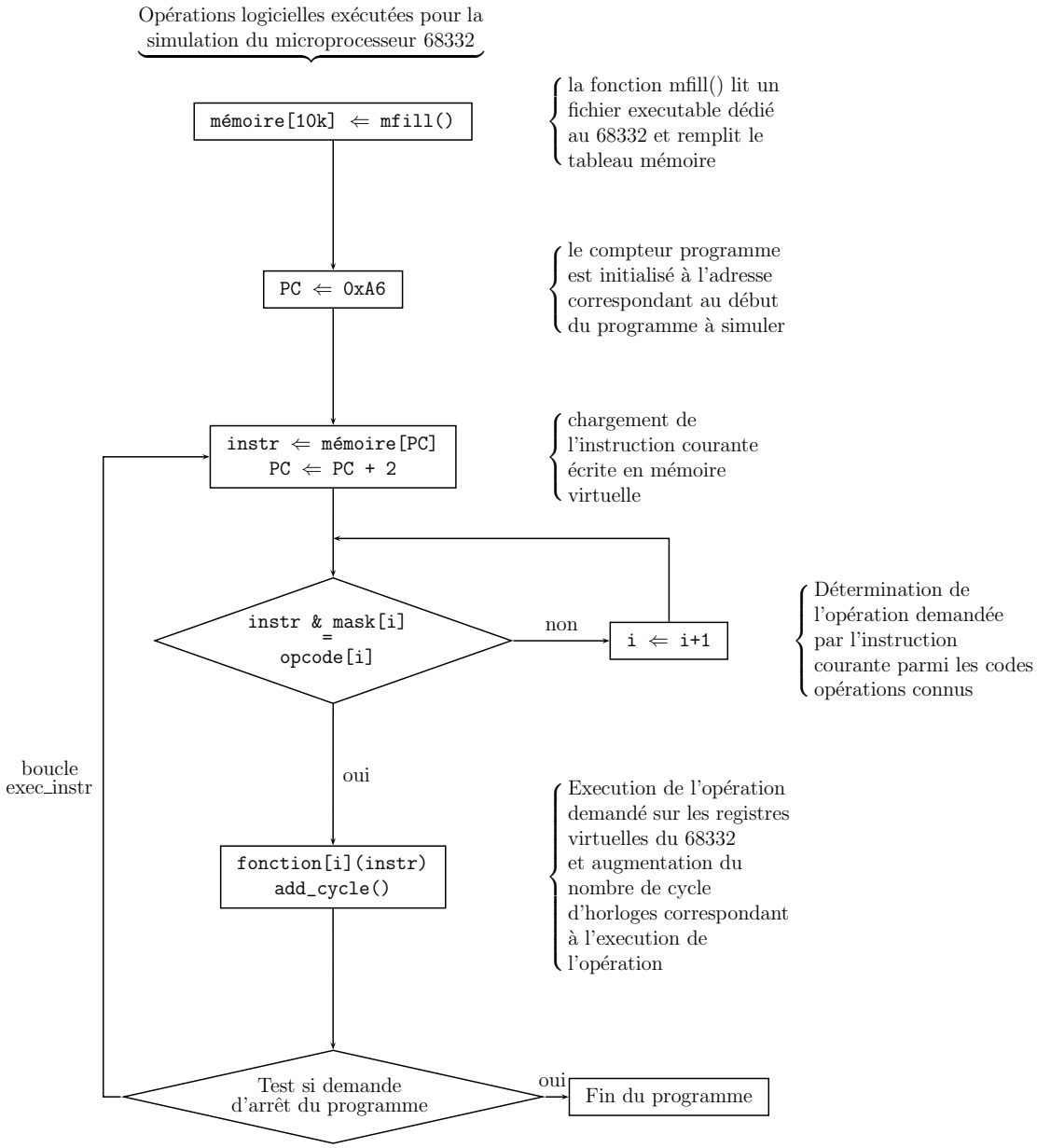
Tableau II.3.1: *Les modèles de programmation du 68332.*

3.3.2.1 Le modèle de simulation du 68332–sim68k.

La simulation interprétée du 68332 est essentiellement basée sur le modèle de programmation du microprocesseur. Ce modèle est issu de la description de l'architecture au niveau des registres (cf. tableau II.3.1) :

- 8 registres de données 32 bits : D0, D1, . . . , D7
- 8 registres d'adresses de 32 bits : A0, A1, . . . , A7
- 1 compteur de programme de 32 bits : PC
- 1 registre d'état (*status register*) de 16 bits : SR

La modélisation se limite à l'unité centrale et à une partie de la mémoire vive, qui sont les ressources utilisées pour le calcul des algorithmes de régulation que nous allons étudier.



Fonctions et variables principales du sim68k	
`opcode[i]`	tableau des codes opérations connus
`mask[i]`	tableau des masques de bits associés aux codes opérations i
`fonction[i]`	fonction réalisant l'opération i

Figure II.3.6: *Fonctionnement logiciel de sim68k.*

Sim68k est programmée en utilisant les principes suivants : à chaque registre du modèle de programmation est associé une variable globale et la mémoire virtuelle du 68332 correspond à un tableau d'octets (`char mémoire[]`). Dans les simulations que nous avons faites, le code de la régulation que nous avons exécuté n'a pas dépassé la taille de 10k octets. La simulation de l'exécution d'un programme se passe alors selon les étapes décrites en figure II.3.6.

3.3.2.2 Le simulateur de type système SABER.

Le logiciel de simulation utilisé pour représenter le fonctionnement du processus (étage de puissance) correspond au logiciel SABER développé par Analogy Incorporation et actuellement maintenu par Avant ! Le choix de ce logiciel s'est basé sur plusieurs critères que nous allons exposer par la suite.

Un système sous SABER est scindé en deux parties : les composants analogiques et les composant numériques. Un moteur de simulation est associé à chacune de ces parties. Nous en présentons les grands principes sachant qu'une description plus précise est disponible dans [45].

Le moteur de simulation analogique de SABER

Sa structure est basée sur la représentation nodale d'un système physique, c'est-à-dire qu'un composant analogique peut-être défini comme un module comportant des points de connexions.

Sur chacun des points, deux grandeurs sont explicitement déclarées : une grandeur de type flux et une grandeur de type potentiel. Une relation entre les deux types de grandeurs modélise le comportement du composant (en général, un jeu d'équations différentielles linéaires ou non).

Par exemple, dans le cas d'une résistance, deux noeuds sont définis aux bornes de celle-ci. Chacun de ces noeuds possède un potentiel électrique (V_1 et V_2) (grandeur de type potentiel) et est traversé par un courant I (grandeur de type flux). La relation habituelle $I = \frac{V_2 - V_1}{R}$ modélise alors le comportement de la résistance.

Un grand nombre de composants physiques peuvent être représentés en utilisant ce principe, qu'ils soient électrique ou mécanique. En effet, dans ces deux domaines de la physique on peut identifier des variables de flux et des potentiels. On retrouve les équivalences suivantes :

- le courant, la force, et le couple sont des grandeurs de type flux,
- la tension, et la vitesse sont de type potentiel.

Les lois de KIRCHOFF peuvent alors être appliquées pour représenter l'interaction entre les composants et une analyse transitoire du système est effectuée en intégrant numériquement le système d'équations différentielles non-linéaires qui en résulte.

Le moteur de simulation numérique

Il correspond à un simulateur d'évènement. Ce type de simulateur est adapté à la simulation des grandeurs qui évoluent de manière discrète dans le temps.

Par définition, tous les changements associés à une grandeur *discrète* sur un horizon donné, peuvent être énumérés par un nombre fini de dates.

L'état d'un système discret est alors est représenté par un ensemble de variables discrètes qui sont définies par un couple *(valeur,date)*, la *date* étant celle à partir de laquelle la *valeur* est effective.

La simulation du système discret consiste à mettre à jour l'état du système à chaque fois qu'une variable change de valeur (évènement). Cette mise à jour génère alors d'autres évènements à la même date ou à une date future.

L'ensemble des évènements en attente d'émergence est ordonné et mis à jour par ce qui est appelé *l'échéancier*.

L'ensemble des composants numériques et analogiques sont décrits selon un langage propre au logiciel SABER dénommé MAST.

Principe de communication entre la simulation analogique et la simulation numérique

L'échange des données du monde numérique vers le monde analogique est réalisé par l'algorithme de Calaveras, qui comprend 3 étapes :

1. Calcul de l'état des grandeurs analogiques en $n+1$: X^1_{n+1}
2. Détecter si entre t_n et t_{n+1}, il s'est produit un évènement numérique de date t_{event}
3. Dans l'affirmative, et si le composant numérique contient l'instruction MAST `schedule_next_time(`t_{event}`)`, le système analogique est réévalué à cet instant par une interpolation du second degré du type :

$$X^2_{n+1} = a \times t^2_{event} + b \times t_{event} + c \tag{II.3.8}$$

à partir des données X_{n-1}, X_n et X^1_{n+1}.
Le point X^1_{n+1} est ensuite rejeté et remplacé par le point X^2_{n+1}. L'horloge analogique est alors recalée à t_{event}.

L'échange des données du monde analogique vers le monde numérique utilise une instruction `threshold()` qui permet de déclencher un évènement discret à partir d'une grandeur analogique lors du franchissement d'un seuil. Le principe d'échange est le suivant : le simulateur analogique évalue X^1_{n+1}. Si celui-ci dépasse le seuil indiqué avec la commande `threshold(`X_0`)`, le vecteur X_0 est calculé en effectuant une interpolation entre le point X_n et le point X_{n+1}.

Le monde analogique et le monde discret échangent donc des informations suivant les deux instructions décrites précédemment :

- `threshold()` analogique → discret
- `schedule_next_time()` discret → analogique

Le choix de SABER permet donc de connecter aisément tout type de système numérique par l'intermédiaire du moteur de simulation à évènement discret et son interface avec le monde analogique.

De plus, les composants analogiques peuvent être décrits selon des modèles mathématiques bien connus (relations algébro-différentielles), et son aspect modulaire permet de définir des modèles plus ou moins fins du processus et de les inter-changer aisément.

3.3.2.3 L'interface SABER ⇔ SIM68K.

Le logiciel SABER permet d'appeler des fonctions écrites en "C". L'interfaçage entre les deux logiciels va s'effectuer grâce à cette propriété.

En effet, une fonction décrite en langage "C" nommée `inter_sab_68k()` et située dans un composant numérique (SABER), est appelée à chaque front montant d'une horloge de période T_e modélisant le port 2 du PTM (qui joue le rôle d'une horloge temps réelle sur la maquette expérimentale). Cette fonction communique alors avec sim68k par deux intermédiaires. Tout d'abord, elle a accès à la mémoire virtuelle du microprocesseur. Ensuite elle contrôle l'exécution du programme car c'est elle qui contrôle le déroulement de la boucle `exec_instr` définie sur la figure II.3.6 page 57. Cette fonction peut donc écrire la donnée numérique correspondant au courant I_m venant du CAN et lire la donnée que le microprocesseur veut attribuer au port 3 du PTM, correspondant à la valeur de β, pour commander le hacheur. La mise à jour de β se fait en tenant compte du temps de calcul associé à la régulation.

En effet, la procédure `inter_sab_68k()` a accès à la variable indiquant le nombre de cycles qu'exécute le microprocesseur. Ainsi la valeur de β peut être prise en compte par l'échéancier pour une mise à jour à la date *tps_calc*. Ces principes sont représentés sur la figure II.3.7.

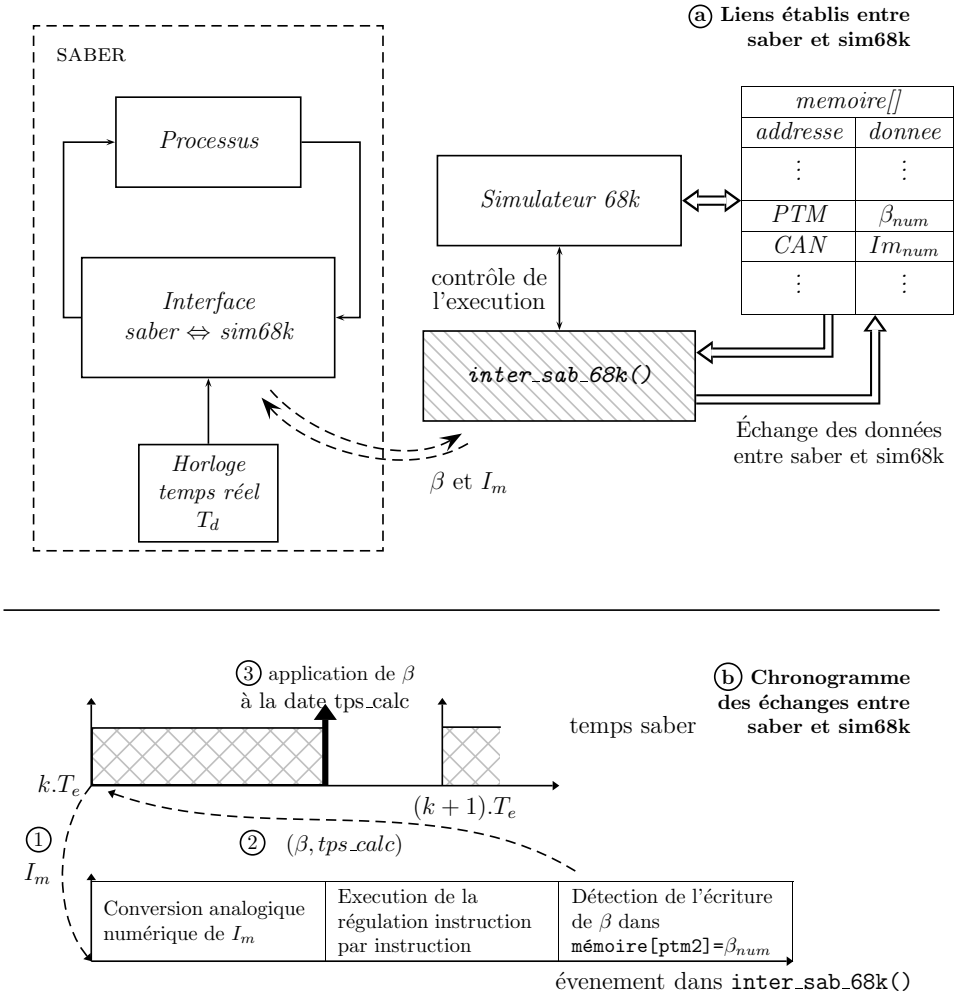


Figure II.3.7: *Principe de la connexion saber sim68k.*

Afin d'évaluer au mieux le temps de calcul, le simulateur a été modifié pour prendre en compte le nombre de cycles d'horloges lors d'une lecture ou d'une écriture correspondant à des adresses de périphériques. En effet, les différents périphériques de la maquette étant des modules synchrones, leurs accès sont cadencés sur une horloge particulière qui implique des temps plus longs.

La co-simulation dans la cadre de la régulation de courant nous a permis alors d'estimer le temps de calcul mais aussi de vérifier la validité du code de commande écrit.

Ainsi, dans ce premier environnement de co-simulation, nous avons réalisé un module nommé `inter_sab_68k()` qui permet d'interfacer le logiciel SABER à un simulateur de microprocesseur en tenant compte des périphériques d'entrées/sorties. Cet environnement a été réalisé de façon à simplifier l'utilisation.

Ainsi, l'étage de puissance correspondant au processus est modélisé sous SABER et l'utilisation du simulateur sim68k se présente simplement comme un bloc échantillonné effectuant le calcul de la régulation avec, comme entrée, le courant I_m et, comme sortie, β. Ce bloc s'utilise simplement en connectant ses entrées et sorties au reste du processus et en indiquant le fichier binaire que doit charger sim68k dans sa mémoire virtuelle (cf. figure II.3.8).

Le schéma général du système sous SABER est d'ailleurs décrit sur la figure II.3.9.

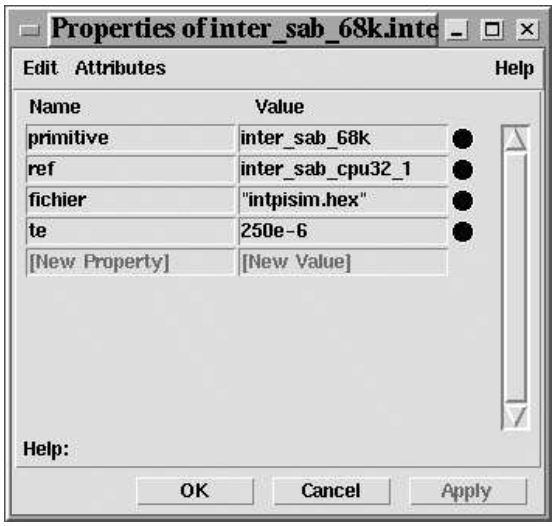


Figure II.3.8: *Propriétés vue par saber du simulateur sim68k.*

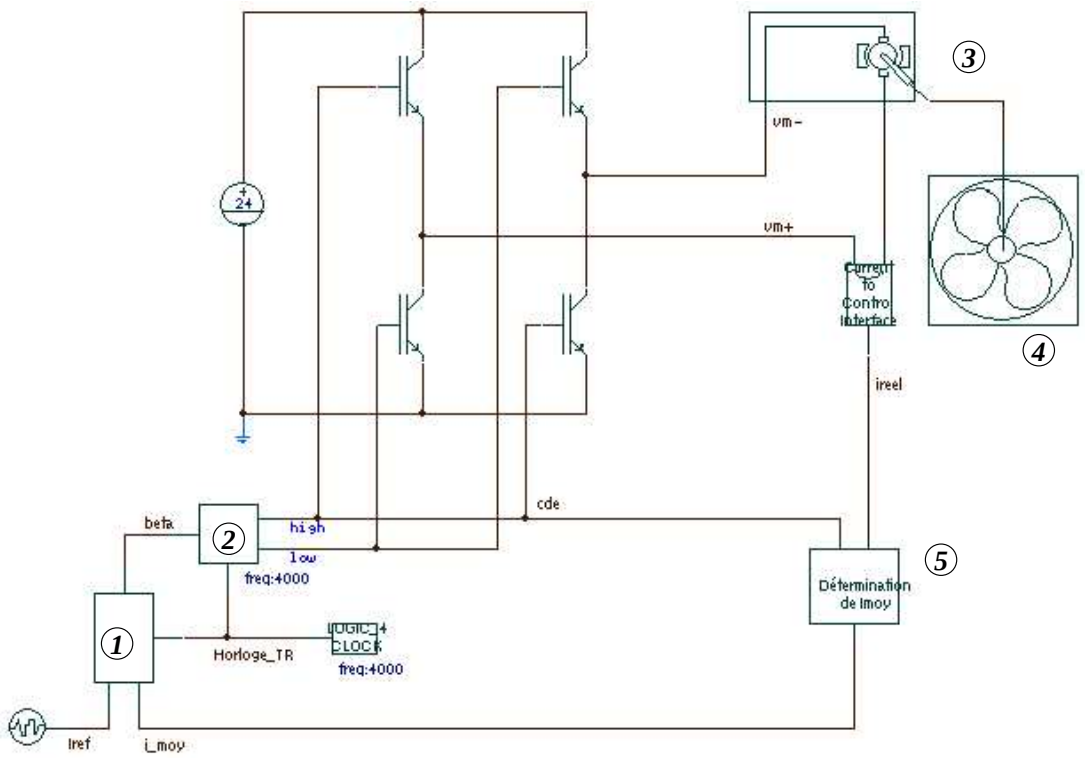


Figure II.3.9: *Schéma général décrit sous* SABER.

On peut y voir les éléments suivants :

① le module représentant le microprocesseur 68332,

② le module calculant les ordres de commande et correspondant au port 3 du P.T.M sur la maquette expérimentale,

③ le modèle SABER de la machine à courant continu,

④ la charge représentée par un frottement visqueux du type $Cr = f_1.\Omega$,

⑤ le filtre associé à la mesure du courant I_m.

3.3.2.4 Résultats de co-simulation de la régulation de courant.

Les résultats présentés ici n'ont pas pour but d'étudier la structure de régulation mais de confronter les résultats obtenus par co-simulation aux résultats expérimentaux.

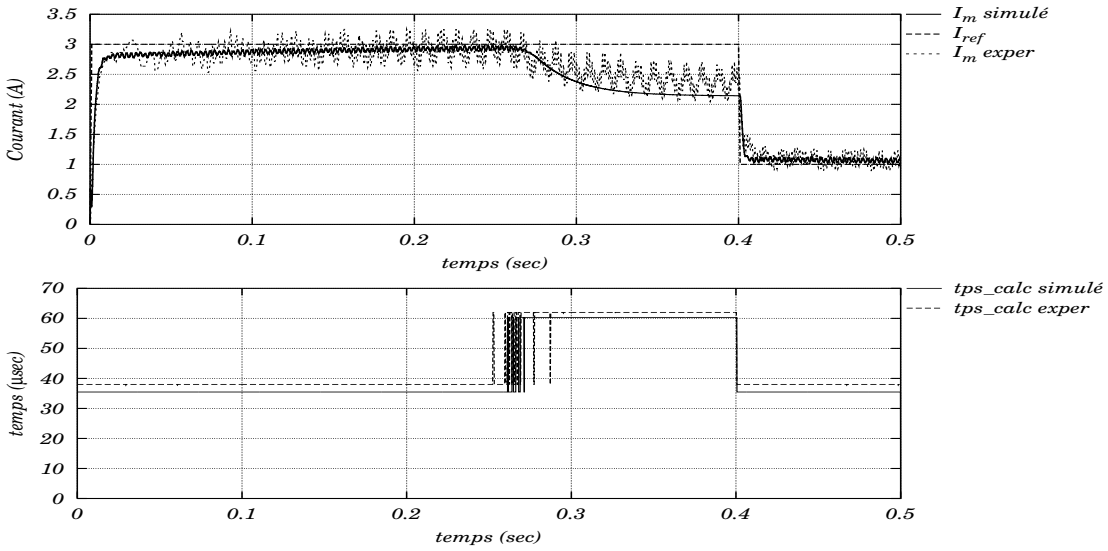


Figure II.3.10: *Régulation de courant de type PI sans compensation de f.e.m.*

Nous présentons ici, le résultat de simulation dans le cas où l'on effectue une régulation de courant à l'aide d'un correcteur PI (Proportionnel Intégral). Cette régulation ne prend pas en compte l'évolution de la f.e.m, ce qui engendre de faibles performances lorsque l'erreur, entre le courant de référence I_{ref} et le courant d'induit I_m, est faible.

Le profil de simulation présenté correspond au démarrage de la machine à demi-charge, avec un courant de référence égal à sa valeur nominale. On observe, figure II.3.10, le phénomène classique du décrochage du courant. En effet, étant en présence d'une commande en couple, la vitesse de rotation à tendance à augmenter. Cependant, la force contre-électromotrice $E = Kem.\Omega$ ne peut plus être compensée par la tension du hacheur $V_m = 2.\beta.U_c$ car la valeur de β entre en saturation.

Le courant expérimental étant fortement bruité, les courbes suivantes sont présentées en lissant celui-ci, afin d'observer l'évolution de la valeur moyenne.

Les temps de calcul affichés figure II.3.10 correspondent aux temps de calcul de la régulation en tenant compte des accès aux périphériques extérieurs. Les différences que l'on constate entre la co-simulation et les résultats expérimentaux peuvent provenir de différentes sources potentielles. En effet, il existe des erreurs de mesures car celles-ci sont effectuées à l'aide d'un temporisateur programmable (module TPU) utilisant une horloge quatre fois plus lente que l'horloge système. De plus, la prise en compte des temps d'accès aux périphériques extérieur dans le temps de calcul estimé par sim68k reste encore approximatif. Compte tenu des ces différentes sources d'erreurs, les écarts obtenus semblent acceptables ($\approx 8\%$).

Nous avons ensuite évalué la régulation en considérant deux implantations numériques différentes. La régulation a été codée en utilisant des variables entières de taille 32 bits (type `integer`) puis de taille 16 bits (type `short`).

On constate dans ce cas que le temps de calcul entre les deux routines de régulation est très peu différent (figure II.3.11) et que la régulation en elle-même subit peu l'influence du format de codage (figure II.3.12).

Cependant ce genre d'information peut être utile dans le cas où l'on cherche à diminuer la taille du code au maximum, ou bien lorsque la taille des variables permet

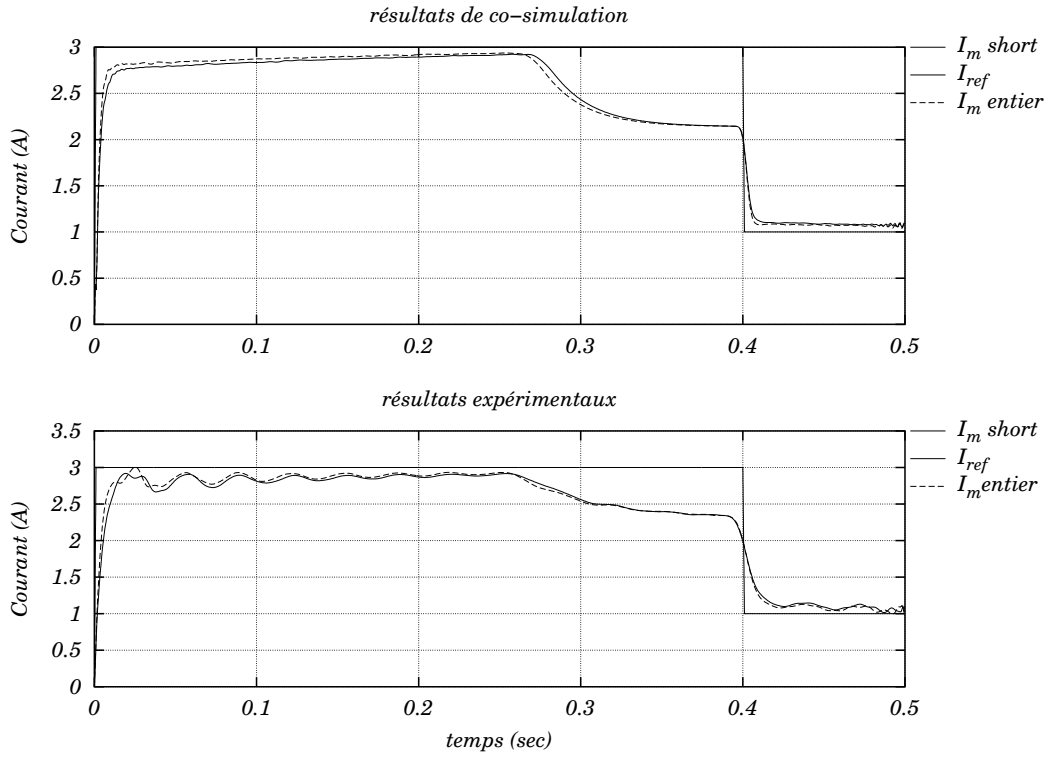


Figure II.3.11: *Comparaisons de la régulation selon le format de codage 32 bits (int) ou 16 bits (short).*

d'augmenter le débit des données vers un autre équipement.

3.3.3 Cas d'un système multitâche.

Dans l'étude précédente, le microprocesseur n'avait qu'une seule tâche. Par ailleurs, le mécanisme d'interruption n'a pas été pris en compte. Dans le cas de plusieurs tâches, il devient nécessaire d'intégrer ce mécanisme en particulier pour la gestion des priorités.

sim68k ne simulant pas la gestion des interruptions matérielles dans sa version initiale, nous avons rajouté un module permettant de les prendre en compte. Nous l'avons ensuite validé dans le cas d'une double régulation numérique : régulation de vitesse et régulation de courant (configuration décrite en 3.3.1.1 page 53).

3.3.3.1 La gestion des interruptions.

Une interruption est un signal matériel provoquant une rupture de séquence dans le déroulement du programme en cours. Pour réaliser la rupture de séquence, le microprocesseur recherche l'adresse du nouveau programme à exécuter dans la table des vecteurs. Il utilise pour cela le numéro de vecteur dont le rôle est de pointer sur cette adresse.

Dans le cas que nous allons étudier, le microprocesseur calcule automatiquement le numéro de vecteur à utiliser. Un masque contenu dans le registre d'état indique le niveau de priorité en cours. Les interruptions de niveau inférieur ou égal au niveau de priorité courante sont inhibées. Le 68000 comprend 7 niveaux de priorité.

Lorsqu'une interruption est prise en compte, la séquence de traitement est la suivante :

1. Copie du registre d'état dans un registre interne au 68000,

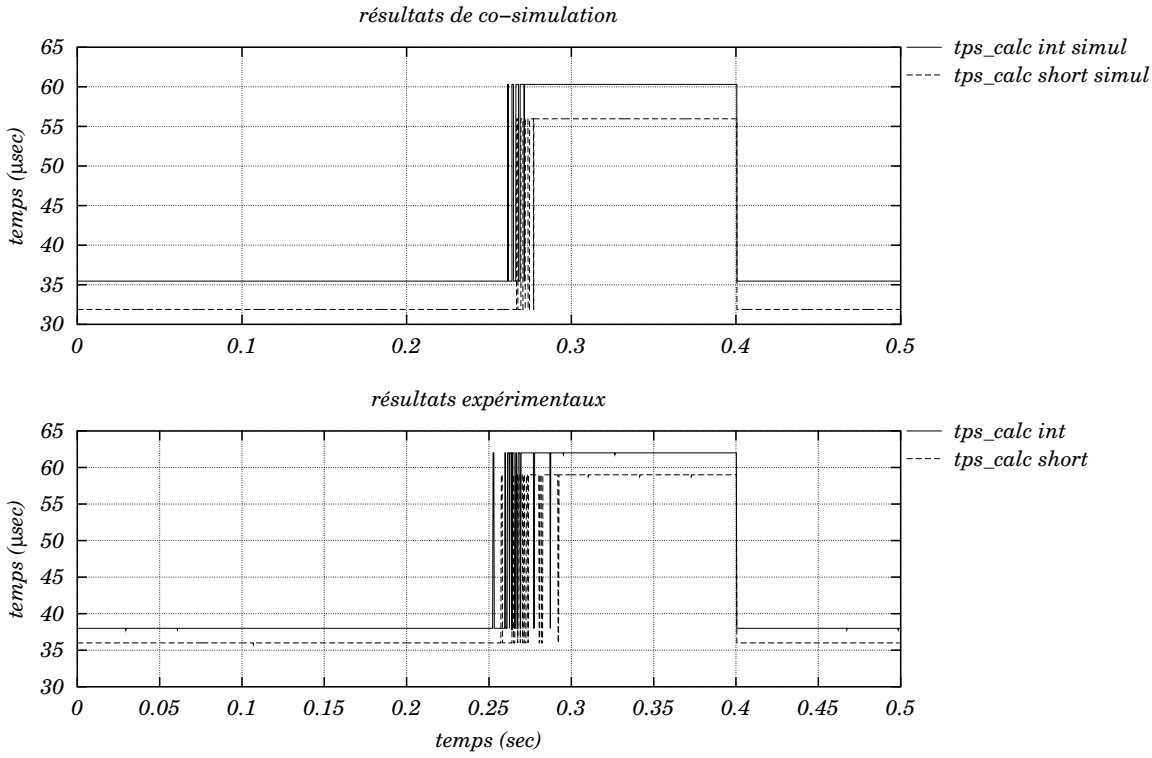


Figure II.3.12: *Comparaison des résultats suivant le format de codage 32 bits (int) ou 16 bits (short).*

2. Passage en mode superviseur et inhibition du mode trace,
3. *recherche du n° de vecteur*, cette recherche se fait de manière automatique par le microprocesseur et correspond au calcul suivant : $(I2\ I1\ I0)_2 + (24)_{10}$
4. sauvegarde du compteur programme et du registre d'état dans la pile superviseur pointer par $A'7$ (cf. tableau 3.1(b) page 56),
5. recopie du niveau de priorité (I0,I1,I2) dans le registre d'état SR,
6. recherche de l'adresse du programme à exécuter dans la table des vecteurs,
7. mise de cette adresse dans le compteur programme,
8. exécution de la routine,
9. fin de la routine par l'instruction RTE qui recharge l'ancien contexte, c'est-à-dire le registre d'état et le compteur programme qui étaient sauvegardés dans la pile superviseur.

Comme la rupture de séquence ne s'effectue qu'à la fin de l'exécution d'une instruction, nous avons ajouté le module reproduisant une partie de ce mécanisme dans la boucle `exec_instr` (figure II.3.6 page 57).

3.3.3.2 Mise à jour de l'interface saber sim68k.

Afin de prendre en compte la gestion des interruptions matérielles et d'effectuer le co-simulation de la double régulation numérique, l'interface entre SABER et sim68k a été légèrement modifiée.

On considère que les dates d'interruption sont connues, puisque nous effectuons des régulations à échantillonnage régulier. SABER est toujours considéré comme maître et sim68k comme esclave. Cependant l'arrêt de sim68k peut maintenant s'effectuer dans deux cas :

1. la routine de courant se termine et renvoie la valeur de β,
2. une nouvelle interruption est demandée, on doit alors mettre à jour les grandeurs d'entrée (vitesse et/ou courant d'induit).

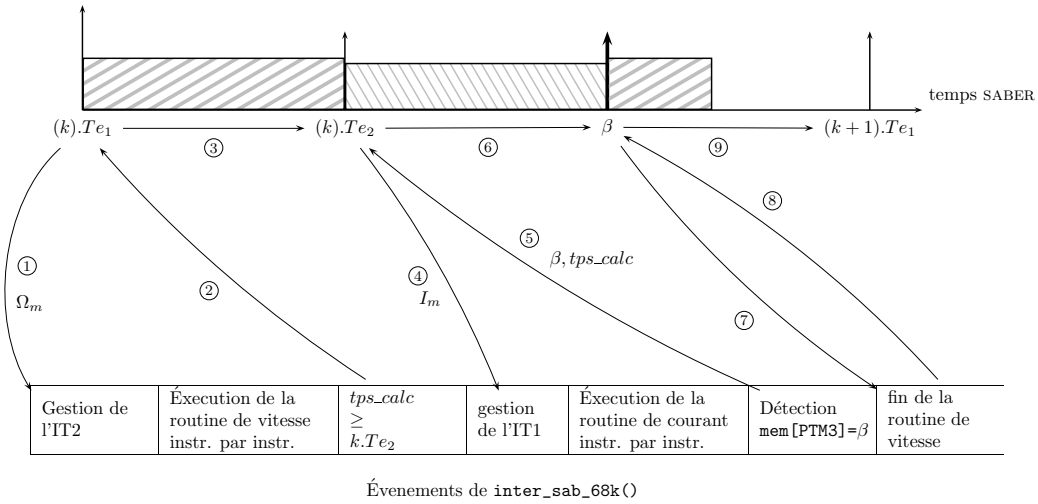


Figure II.3.13: *Communication* SABER ⇔ *sim68k : cas de la double régulation.*

Le schéma II.3.13 représente la communication entre SABER et sim68k dans le cas de la double régulation et lorsque la routine d'interruption utilisée pour calculer la valeur de β interrompt la routine de vitesse. Nous avons représenté les échanges en 9 étapes. Nous énumérons ces 9 étapes :

① envoi d'une demande d'interruption à sim68k de niveau 2 (routine de vitesse),

② la routine s'exécute jusqu'à ce que le nombre de cycles atteigne une valeur telle que la routine de courant doive être appelée,

③ simulation du système sous SABER jusqu'à la demande d'interruption pour l'exécution de la routine de courant,

④ demande d'interruption de niveau 1 (régulation de courant) prise en compte sous sim68k et mise à jour de la mesure du courant d'induit I_m,

⑤ simulation de la routine de courant jusqu'à l'écriture de β sur le port 3 du PTM, puis envoie de la valeur de β et de sa date de mise à jour,

⑥ simulation sous saber jusqu'à la mise à jour de β sur le PTM 3,

⑦ reprise du calcul de la routine de vitesse,

⑧ fin de la routine de vitesse,

⑨ simulation jusqu'à la prochaine demande d'interruption.

3.3.3.3 Résultats.

Nous présentons ici les résultats de simulation dans le cas où la vitesse et le courant sont régulés numériquement par le microprocesseur. Les régulateurs utilisés sont de type PI, et synthétisés selon l'hypothèse de la séparation des modes (cf. section 3.3.1.2 page 54).

La figure II.3.14 représente l'évolution de la vitesse, celle-ci évolue jusqu'à atteindre un maximum en 4 ms. En fait, la référence de vitesse qui a été choisie est volontairement trop importante, les régulations cherchant à atteindre cette référence vont entrer en

saturation, ce qui nous permet de tester différents cas possibles dans les algorithmes de régulation, c'est-à-dire lorsque $I_{ref} = I_{max}$ et $\beta = \beta_{max}$. En effet, la valeur de β est, comme nous l'avons vu précédemment, limitée par l'utilisation du hacheur, mais le courant de référence imposé par la régulation de vitesse est aussi limité à I_{max}, afin de ne pas détériorer le système.

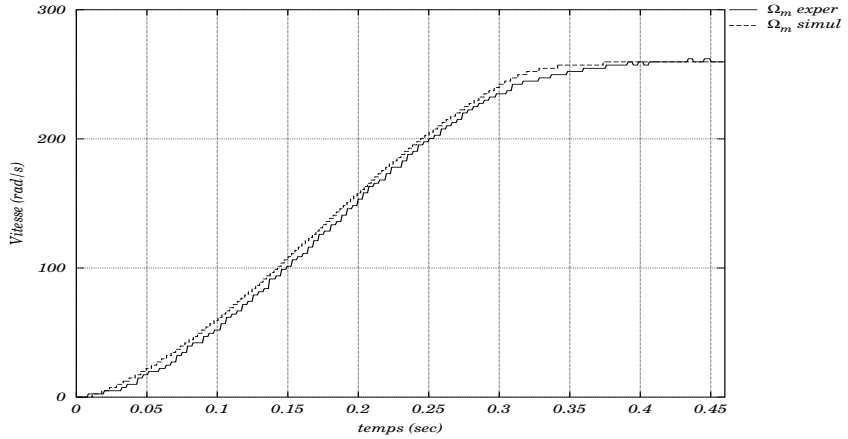


Figure II.3.14: *Comparaison entre la co-simulation et l'expérience pour la régulation de vitesse.*

La figure II.3.15 présente l'évolution du temps de calcul de la régulation de vitesse. On constate la présence de "pics" dans les résultats expérimentaux et de co-simulations. Ceci est en partie dû au fait que les horloges temps réel pour les interruptions ne sont pas multiples. Étants non multiples, il apparaît un phénomène de glissement entre les deux fronts montants d'horloge (associée à l'échantillonnage des régulations de vitesse et de courant).

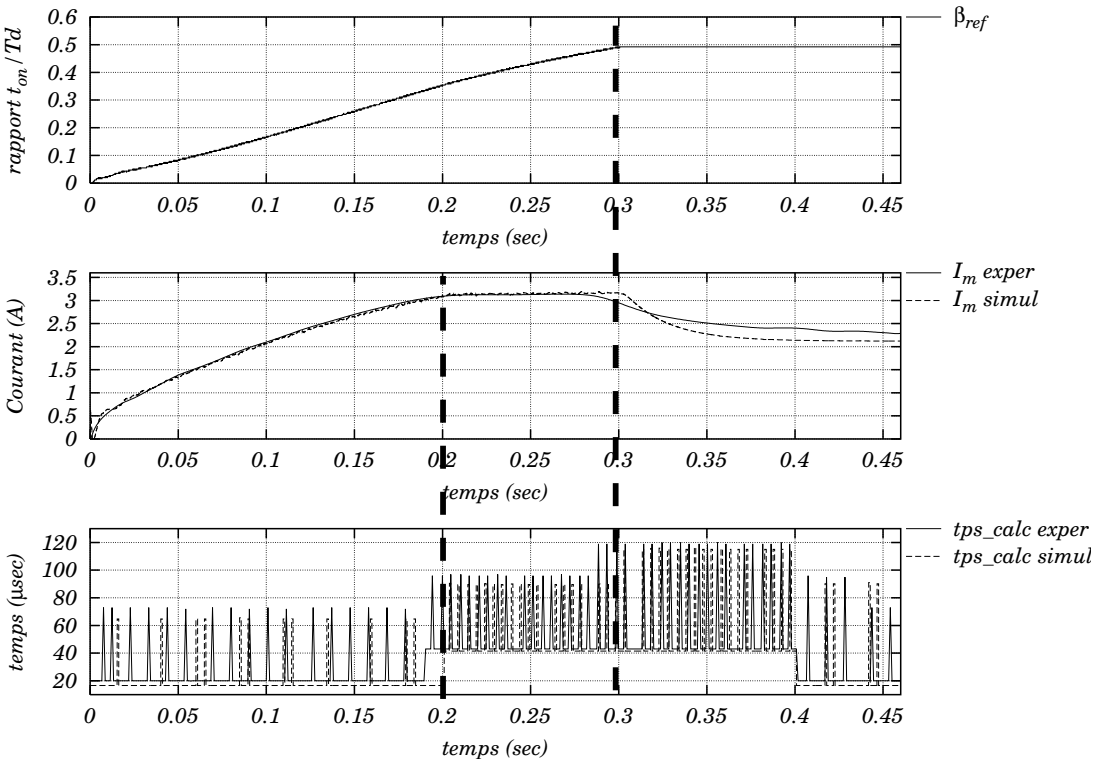


Figure II.3.15: *Comparaison des temps de calcul estimés en co-simulation et des temps de calcul mesurés expérimentalement pour la régulation de vitesse.*

La conséquence de ce glissement est l'interruption irrégulière de la routine de vitesse par la routine de courant qui introduit, ou non, un temps de calcul supplémentaire pour la régulation de vitesse. Le premier échelon observé à 200 ms est dû à la saturation du courant de référence ($I_{ref} = I_{max}$ *et* $\beta < \beta_{max}$). Le temps de calcul de la régulation de vitesse étant plus grand en raison de cette saturation, on voit nécessairement apparaître un nombre de pics de plus en plus important. Le deuxième échelon qui apparaît à $300ms$ environ est associé à la saturation de β ($I_{ref} = I_{max}$ *et* $\beta = \beta_{max}$).

La co-simulation permet de visualiser des données, comme l'évolution du temps de calcul, qu'il est difficile d'acquérir sur un système expérimental. Ces données peuvent alors rendre compte de problèmes particuliers et aider à la mise au point des régulations.

3.4 Un Essai de simulation compilée.

Lors de la présentation des différents simulateurs de microprocesseurs, nous avons indiqué que la simulation compilée semblait être une alternative. Nous avons alors évalué cette technique en réalisant un essai relativement simple. Cet essai porte sur l'estimation du temps de calcul pour une régulation de type PI exécutée par un DSP TMS320 de type C3x de Texas Instruments. Dans ce cas, le régulateur correspondra à la régulation de vitesse implantée dans le cas du 68332.

3.4.1 Principe de la simulation compilée.

Comme nous l'avons indiqué précédemment, la simulation compilée consiste à retranscrire le fichier binaire dédié à la cible en un fichier binaire exécutable sur station, tout en ajoutant des informations telles que le temps de calcul.

Le procédé de translation d'un fichier binaire en un autre fichier binaire peut être réalisé de deux manières. L'approche directe consiste à transformer directement l'exécutable de la machine cible en un exécutable pour la station de travail.

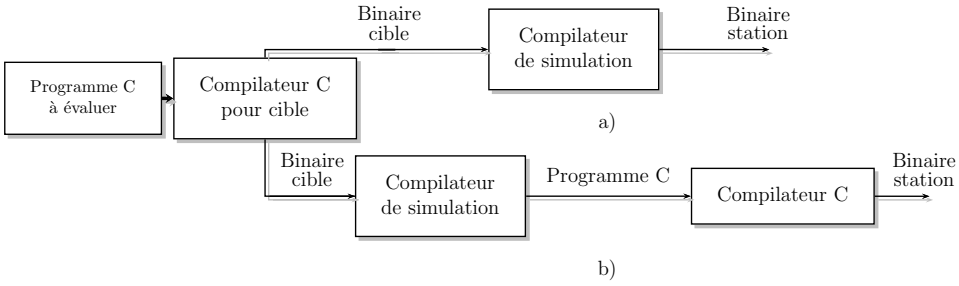


Figure II.3.16: *Deux approches différentes pour la translation de fichier binaires.*

Ce schéma implique la création d'un compilateur complexe (figure II.3.16.a) devant retranscrire un langage binaire en un autre langage binaire ce qui, de plus, le rend peu portable (le résultat binaire ne pourra s'exécuter que sur un type de machine). La seconde méthode (figure II.3.16.b) consiste à séparer la transcription en deux étapes [75].

La première étape repose sur la compilation de l'exécutable initial en un programme écrit avec un langage de haut niveau comme le C. Ce programme C est ensuite compilé pour une station de travail particulière. De cette manière, seule la première étape nécessite le développement d'un compilateur car la deuxième étape utilise les outils de compilation existants. La portabilité en est accrue.

Nous avons retenu cette deuxième approche pour reproduire le fonctionnement du

DSP TMS320C3X de Texas Instrument lors de l'exécution de la régulation de vitesse. Nous avons utilisé le fichier assembleur comme point de départ. Ce fichier assembleur résulte de la compilation par les outils propriétaires du programme C décrivant l'algorithme de régulation. Nous avons ensuite développé un compilateur spécifique permettant de retranscrire les instructions assembleur du C3x en instructions équivalentes en langage C.

3.4.2 Technique de compilation utilisée.

Un compilateur est généralement composé de plusieurs phases [2], comme représenté sur la figure II.3.17.

L'analyse lexicale consiste à lire un flot de caractères venant du programme source et à regrouper les caractères qui forment un mot ayant une signification particulière appelé unité lexicale. Par exemple, dans la figure II.3.17, `position` est une unité lexicale de type "identificateur" (id) et `:=` est une autre unité lexicale. Le flot d'unités lexicales est alors lu par l'analyseur syntaxique qui cherche à regrouper les unités lexicales en structures grammaticales représentées sous la forme d'arbre syntaxique. L'identification de la structure grammaticale est, après une analyse sémantique, utilisée par le générateur de code intermédiaire. L'analyse sémantique, quant à elle, contrôle si le programme source contient des erreurs sémantiques (comme le type des opérandes) et collecte des informations nécessaires à la production du code intermédiaire.

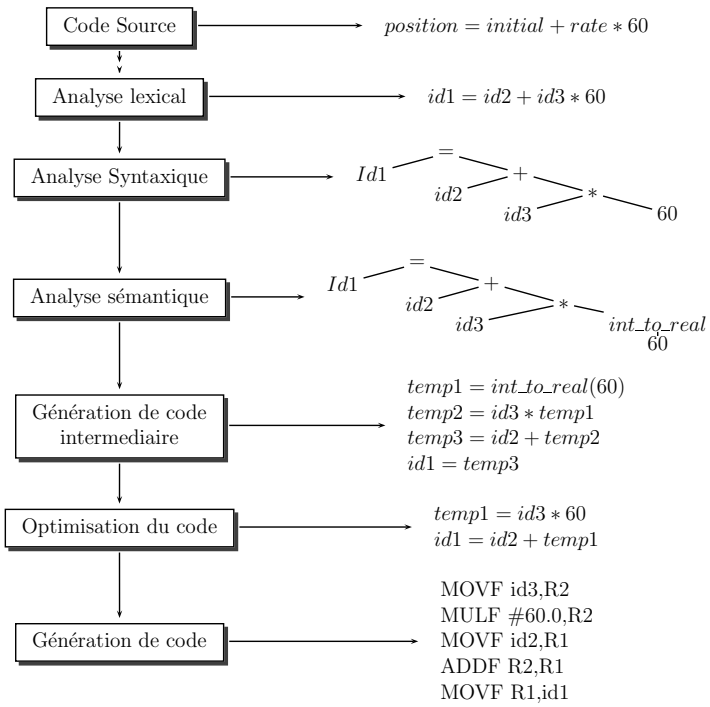


Figure II.3.17: *Les différentes phases habituelles d'un compilateur.*

Étant donné que nous désirons simplement effectuer une correspondance entre une instruction assembleur et une instruction (ou une série d'instructions) en C, nous n'avons pas eu besoin de nous intéresser aux étapes correspondant aux phases d'optimisations et de générations de code. En effet, il est possible de construire un compilateur simple, en une seule passe, s'appuyant essentiellement sur l'analyse lexicale et l'analyse syntaxique appelé : traduction dirigée par la syntaxe.

Cette technique associe des actions dites sémantiques aux règles grammaticales.

Les règles grammaticales sont décrites en utilisant une notation appelée grammaire non contextuelle. Cette grammaire correspond à une suite de règles que l'on nomme production. Une production se présente sous la forme suivante :

instruction	$\longrightarrow$	ADDI src dest | LDI src dest
src	$\longrightarrow$	registre | adresse
dest	$\longrightarrow$	registre | adresse

La barre verticale correspond à une opération qui peut se traduire par "ou". Dans cet exemple, on définit une instruction comme étant l'enchaînement des unités lexicales "ADDI src dest" ou bien "LDI src dest".

Une traduction dirigée par la syntaxe associe alors à chaque règle grammaticale des actions. Par exemple :

instruction	$\longleftarrow$	ADDI src dest	{imprimer("c'est une addition")}
	|	LDI src dest	{imprimer("c'est un chargement d'entier")}

Le programme de translation assembleur C3X vers le langage C a été développé selon cette technique, en utilisant les outils classiques de constructeurs d'analyse lexicale et d'analyse syntaxique que sont lex et yacc.

3.4.3 Évaluation du temps de calcul et précision.

Afin d'évaluer le temps de calcul d'un programme, une variable globale est incrémentée d'un nombre de cycles correspondant au temps de calcul de chaque instruction assembleur analysée. Cependant, le temps de cycle de certaines instructions varie selon les données traitées. Ceci est essentiellement dû aux conflits de pipeline ou aux conflits d'accès mémoire qui introduisent des temps d'attente. Cependant, la plupart de ces variations peuvent se prévoir durant l'analyse du code assembleur et peuvent être pris en compte.

La précision, dans ce cas particulier, n'a pas posé de problème, étant donné que le DSP C3x effectue des calculs très proches des formats standards. Cependant, si cela n'avait pas été le cas, il aurait fallu écrire le code en C++ afin de surcharger et de redéfinir les opérateurs arithmétiques usuels.

3.4.4 Résultats de la simulation compilée.

Le tableau II.3.2, présente un résultat de retranscription d'un fichier assembleur.

Dans cet exemple, la ligne 4 du fichier assembleur, c'est à dire l'instruction :

```
LDI *+FP(9),AR0.
```

correspond au chargement d'une valeur entière, contenue dans une mémoire située à l'adresse FP+9, dans le registre d'adresse AR0, et est transformée par l'instruction C écrite en ligne 2 :

```
AR0=(int)*(FP+9).
```

On constate que cette instruction C reproduit la même action en utilisant des variable notées AR0 et FP qui ont la même fonction que dans le code assembleur.

En effet, les différents registres sont, comme dans le cas du sim68k, représentés par des variables. Ainsi, en utilisant un *"debogueur"* classique, on peut visualiser ces variables et effectuer le deboguage du code s'exécutant sur le C3x.

code assembleur	code C résultant

```
debut
L1:
 .line 30
 LDI *+FP(9),AR0
 LDI *+FP(10),AR1
 SUBF *AR1,*AR0,R0
 STF R0,*+FP(8)
 .line 31
 MPYF R0,*+FP(1),R1
 ADDF *+FP(6),R1
 STF R1,*+FP(5)
 .line 32
 MPYF *+FP(2),R0
 ADDF R1,R0
 STF R0,*+FP(11)
 .line 29
 LDI @STATIC_1,R2
 ADDI 1,R2
 STI R2,@STATIC_1
 CMPI 7,R2
 BLE L1
 .line 35
 CMPF *+FP(13),R0
 BGT L3
 .line 36
 LDF *+FP(13),R0
 STF R0,*+FP(11)
fin_asm
```

```
L1:
AR0 = (int) *(FP+9) ;++Tecalc
AR1 = (int) *(FP+10) ;++Tecalc
R0 = *(AR0) - *(AR1) ;++Tecalc
*(FP+8) = R0 ;++Tecalc
R1= *(FP+1) * R0 ;++Tecalc
R1 = R1 + *(FP+6) ;++Tecalc
*(FP+5) = R1 ;++Tecalc
R0 = R0 * *(FP+2) ;++Tecalc
R0 = R0 + R1 ;++Tecalc
*(FP+11) = R0 ;++Tecalc
++Tecalc;
R2 = (int) *(STATIC_1) ;++Tecalc
R2 = R2 + 1 ;++Tecalc
++Tecalc;
*(STATIC_1) = R2 ;++Tecalc
if ((R2-7)==0) Z=1; else
    {
     Z=0;
    if ((R2-7)<0) N=1; else N=0;
    }
++Tecalc
/******LE******/
if ((N==1)||(Z==1)){
    Tecalc=Tecalc+4;goto L1;
    }
if ((R0-*(FP+13))==0) Z=1; else
    {
     Z=0;
    if ((R0-*(FP+13))<0) N=1; else N=0;
    }
++Tecalc
/******GT******/
if ((Z==0) && (N==0)){
    Tecalc=Tecalc+4;goto L3;
    }
R0 = *(FP+13) ;++Tecalc
*(FP+11) = R0 ;++Tecalc
```

Tableau II.3.2: *Exemple de retranscription assembleur → C incluant l'évaluation du temps de calcul "Tecalc".*

Afin de tester la performance de notre compilateur, nous avons comparé les temps de calcul du programme résultant de la simulation compilée à ceux obtenus à l'aide du simulateur fourni par le constructeur. Les tests se sont effectués en fixant la valeur de la vitesse de consigne et la vitesse mesurée. Une première série d'essais s'est effectuée avec $V_{mes} = 100$ permettant de contraindre le régulateur à rentrer en saturation. La seconde utilise $V_{mes} = 295$, laissant le calcul du régulateur dans sa partie linéaire. Deux branches de l'algorithme ont ainsi été testées et ceci selon différentes options de compilation. Nous avons obtenu une erreur maximale de 7%, comme le montre le tableau II.3.3.

		DSPC3X	Simul. Compilée	erreur relative
option de compilation pour la routine écrite en C		⇒	sans option	
Vref= 300 Vmes= 100	Iref Tps calc	5 306	5 302	- 1.3 %
Vref= 300 Vmes= 295	Iref Tps calc	3.341... 252	3.341... 256	- 1.6 %
option de compilation pour la routine écrite en C		⇒	-o0 (option de premier niveau)	
Vref= 300 Vmes= 100	Iref Tps calc	5 179	5 172	- 4 %
Vref= 300 Vmes= 295	Iref Tps calc	3.341... 172	3.341... 175	- 2%
option de compilation pour la routine écrite en C		⇒	-o2 (option de troisième niveau)	
Vref=300 Vmes=100	Iref Tps calc	5 106	5 99	- 7%
Vref= 300 Vmes= 295	Iref Tps calc	3.341... 101	3.341... 98	- 3%

Tableau II.3.3: *Comparatif entre la simulation compilée et le simulateur de Texas Instrument.*

3.5 Conclusion.

Dans ce chapitre, nous avons présenté les différents outils permettant de simuler le fonctionnement du microprocesseur. Cependant, vu le cadre de notre application, nous nous sommes limités à la simulation interprétée et à la simulation compilée.

La première méthode a été appliquée au microprocesseur 68332 étant donné que nous disposions de ce simulateur. Grâce à lui, nous avons développé un environnement de co-simulation en lui associant le logiciel SABER. Cet environnement a été traité dans le cas d'une commande de vitesse pour une machine à courant continu.

Par le biais de cette application, nous avons montré l'intérêt de la co-simulation. Cet outil permet de valider le code de commande devant être implanté sur la cible et de réaliser des tests de performance (temps de calculs, format de codage). La précision obtenue après comparaison avec le dispositif expérimental confirme l'intérêt de cette méthode dans le développement des algorithmes de commande. Cependant, cette méthode est conditionnée par l'existence d'un simulateur du microprocesseur utilisé.

Pour pallier cet inconvénient, nous avons utilisé la méthode de simulation compilée. La transcription du code assembleur en langage C nécessite cependant la réalisation d'un compilateur spécifique et une bonne connaissance du microprocesseur à simuler. Malgré cela, il nous semble que cette méthode présente un potentiel important que nous n'avons pas pu exploiter par manque d'expérience sur les techniques de compilation. Il nous semble en effet possible de retranscrire le programme en partant du code source et non du code assembleur comme nous l'avons fait [39]. De plus, il nous semble possible d'évaluer l'impact de l'architecture interne du microprocesseur sur le temps de calcul en autorisant ou interdisant l'utilisation de certaines ressources comme le MAC[3], le pipeline, ... Ce qui pourrait permettre l'identification de l'architecture (ou du type de microprocesseur) la plus adaptée à l'algorithme qui doit être exécuté.

[3]Multiply And Accumulate

Chapitre 4

Intégration du VHDL dans un environnement de co-simulation

4.1 Introduction.

Les composants logiques programmables ont des niveaux d'intégration de plus en plus élevés et aujourd'hui, la conception de systèmes numériques complexes utilisant une représentation au niveau des portes logiques n'est plus envisageable. Des outils de conception, dit de haut niveau, sont apparus afin d'améliorer les méthodes d'intégration de la logique sur un CLP. Si on fait une analogie avec les outils du logiciel, on peut situer la représentation à l'aide de portes logiques au niveau du langage d'assemblage, les outils de conception de haut niveau étant comparables aux langages structurés de type "C,Ada,...".

De plus, comme l'écart de complexité entre les ASIC's et les CLP est de plus en plus faible, les outils d'aide à la conception se sont unifiés, et quelques langages standards, tels que VHDL ou Verilog, peuvent être employés pour définir la logique, où plus généralement les algorithmes, à intégrer quelque soit le composant. Ainsi, un algorithme décrit dans un de ces langages peut être évaluée sur un CLP avant d'être intégrée sur un ASIC et ce, avec un minimum de modifications.

L'utilisation du VHDL, et plus généralement des langages de description matériels, dans une optique de réutilisation, semble donc incontournable dans la conception des systèmes numériques complexes.

Ce chapitre présente les aspects essentiels du langage VHDL : son utilisation dans le flot de conception de la logique à intégrer sur un composant et l'organisation typique d'une spécification VHDL. Nous présentons ensuite la création des liens nécessaires à son utilisation en co-simulation. L'interface de co-simulation générée est ensuite validée par l'intermédiaire d'un exemple d'application.

4.2 De l'utilisation du VHDL.

4.2.1 Flot de conception basé sur le VHDL.

La figure II.4.1 illustre les différentes étapes du flot de conception, et montre les différents fichiers et outils nécessaires. Les étapes nécessitant un outil particulier sont indiquées en gris et les données ne pouvant être représentées en VHDL sont indiquées en italique.

Le flot de conception actuel basé sur le VHDL part d'une description au niveau RTL. Cette description peut être simulée afin de valider l'aspect fonctionnel (simulation de niveau 1). Cette simulation nécessite alors la création d'un environnement de test

(testbench) qui déclare une instance du composant à tester et un ensemble de ressources permettant de générer des stimuli appropriés aux bornes de celui-ci.

Après cette validation, le modèle RTL est synthétisé par un outil de synthèse logique en fonction de contraintes définies par l'utilisateur, sur la surface et/ou le délai que doit satisfaire le circuit. La définition de ces contraintes se fait généralement en utilisant un langage propre à l'outil de synthèse. La synthèse logique se base sur une bibliothèque de fonctions logiques disponibles pour le composant cible.

Le résultat de la synthèse est une description sous la forme d'une liste de connexions (Netlist) de portes logiques, de registres et de macro composants particuliers au CLP cible. D'autre part, cette description peut s'obtenir en VHDL en utilisant une bibliothèque des différentes portes utilisées et fournies par le fondeur ou fabricant de FPGA. Une simulation peut alors être effectuée dans ce niveau intermédiaire, en intégrant le VHDL synthétisé, afin de valider l'étape de synthèse logique (simulation de niveau 2).

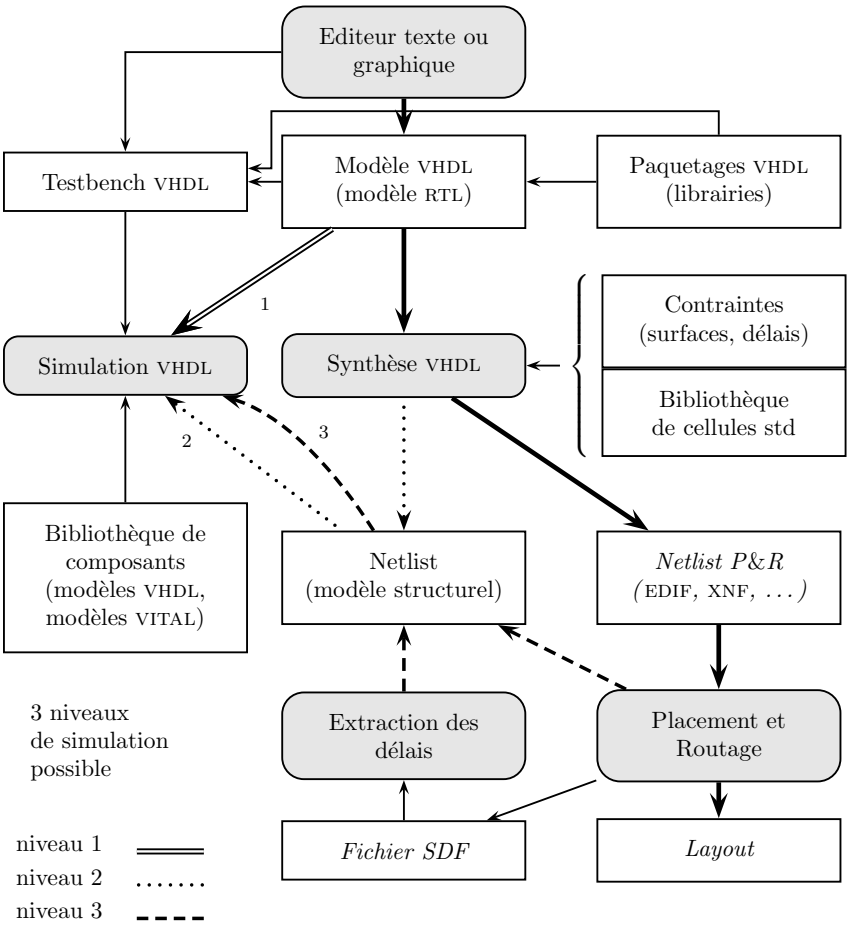


Figure II.4.1: *Flot de conception utilisant le* VHDL.

L'association de portes définie par la synthèse logique est décrite dans un fichier selon une syntaxe particulière correspondant le plus souvent à la norme EDIF (Electronic Design Interchange Format). Ce résultat est alors analysé pour déterminer le placement des différentes portes sur le composant "réel", puis les différentes liaisons sont calculées en utilisant les pistes d'interconnexions du circuit cible. Le résultat du placement et routage (P&R) correspond au fichier qui servira à la configuration du CLP. Durant cette étape, un fichier VHDL décrivant la structure interne du CLP selon des modèles particuliers peut être généré. Ces modèles sont usuellement définis selon

la norme VITAL[1]. Cette norme permet d'annoter le code VHDL avec les délais dûs aux interconnexions, et calculés durant la phase de placement et routage.

Ces délais sont écrits toujours selon la norme VITAL dans un fichier SDF (Standard Delay Format). Une simulation de troisième niveau peut alors être réalisée afin de valider le bon fonctionnement du système tout en tenant compte des temps de propagations. Dans notre cas, l'ensemble des simulations s'effectuera en utilisant le logiciel *Modelsim* de la société ModelTechnology.

4.2.2 Organisation d'un modèle VHDL

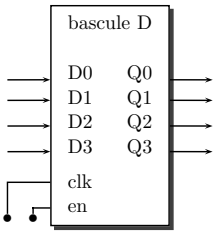
Cette section n'a pas pour but de décrire le langage VHDL. Elle donne simplement les principes de base à partir desquels il est possible de spécifier le fonctionnement de la logique à intégrer sur un CLP.

Afin de pouvoir décrire au mieux le comportement d'un composant numérique, la représentation spatiale (comme les schémas) et l'analyse séquentielle (algorithmes) sont exploitées par le langage VHDL. En effet, le programmeur peut définir un système comme l'association de composants. Il peut adopter une description structurelle et hiérarchique ou décrire son comportement sous la forme d'un algorithme.

Pour cela, le VHDL se base sur une unité de conception élémentaire, *l'entité*. Une entité est déclarée comme une boite noire, c'est à dire que sa déclaration ne comprend que les entrées, les sorties, et les paramètres (si elle en comporte). Nous allons illustrer nos propos en utilisant l'exemple d'un registre de 4 bits.

code 4.1: *Déclaration d'une entité*

```
-- déclaration de l'entite bascule 4 bits
```




```
--
entity reg4 is
 port(
  d0, d1, d2, d3 : in bit;   -- données d'entrées
  clk, en : in bit;          -- horloge et signal de controle
  q0, q1, q2, q3 : bit out);-- données de sorties
 );
end reg4;
```

Le registre est déclaré dans le code 4.1 et aucun élément n'indique encore quelle est sa structure interne. La description de son fonctionnement s'effectue en définissant son *architecture*. Celle-ci peut se faire de différentes manières au choix de l'utilisateur :

- la description structurelle (code 4.2) : dans ce cas, l'architecture est décrite comme l'association de bascules D (*dlatch*) qui sont elles mêmes définies comme des entités indépendantes,

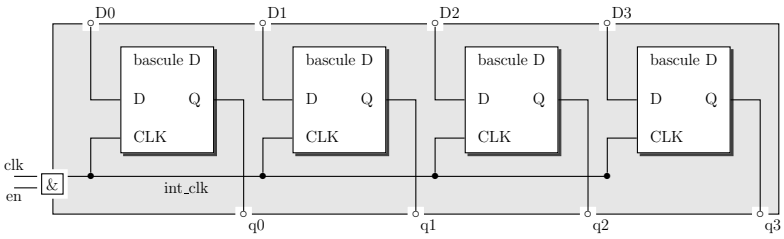
[1] VHDL Initiative Towards VHDL Libraries

code 4.2: *Définition d'une architecture structurelle.*

```
architecture struct of reg4 is
--déclaration du composant instancié
component dlatch is
 port(d,clk: in bit;q:out bit);
end component;
-- signaux internes
signal clk_int:bit;
begin
-- instanciation des bascules
bit0: dlatch port map (d=>d0,clk=>int_clk,q=>q0);
bit1: dlatch port map (d=>d1,clk=>int_clk,q=>q1);
bit2: dlatch port map (d=>d2,clk=>int_clk,q=>q2);
bit3: dlatch port map (d=>d3,clk=>int_clk,q=>q3);

-- génération de l'horloge interne
clk_int <= clk and en;
end struct;
```

schéma équivalent de l'architecture




```
--
```

– la description comportementale (code 4.3) : il s'agit de décrire le fonctionnement du registre sous la forme d'un algorithme. La vision fonctionnelle apporte ainsi la puissance des langages de programmation.

code 4.3: *Définition d'une architecture comportementale.*

```
architecture behaviour of reg4 is
begin
 clk_int <= clk and en;
 process
 begin
  if (clk_int'event and clk_int == '1') then
   q0 <= d0;
   q1 <= d1;
   q2 <= d2;
   q3 <= d3;
  end if;
 end process;
end behaviour;
```

Le plus souvent, on utilisera une description structurelle au niveau supérieur d'un projet, et les modules instanciés seront décrits suivant le type de fonctionnalité à spécifier. Les programmes VHDL générés par les outils de placement et routage, à des fins de vérification temporelle du bon fonctionnement, sont essentiellement structurels : ils reproduisent le câblage réellement effectué dans le circuit.

Nous avons présenté le langage VHDL comme un outil de spécification pour la logique câblée. Cependant, le langage VHDL est aussi un véritable langage de simulation. Cet aspect est d'ailleurs utilisé pour créer l'environnement de test d'un composant. Dans le

cas de tests unitaires, cet environnement est relativement simple. Cependant, lorsqu'il s'agit de tester le comportement global du composant au sein du système, il nécessite d'être interactif. Il est en effet difficile d'être exhaustif dans la génération des vecteurs de test.

La création de d'un tel environnement s'avère alors aussi complexe que le composant lui même. De plus le langage VHDL n'est pas toujours le langage le plus adapté pour spécifier le modèle du système entourant le composant. Cette première analyse montre l'intérêt d'intégrer le VHDL dans un environnement de co-simulation afin d'obtenir une validation dynamique des composants écrits en VHDL

4.3 Le VHDL et la co-simulation

Lorsque la structure logique qui doit être implantée sur un CLP a été spécifiée, il faut pouvoir évaluer son bon fonctionnement. L'ensemble de cette structure VHDL peut être vue comme l'*architecture* d'une *entité*. Cette entité représente alors le composant logique programmable avec ses entrées et ses sorties.

Dans le cadre de la co-simulation, on doit pouvoir connecter cette entité avec d'autres simulateurs permettant de reproduire l'environnement externe du composant et ce, quelque soit le niveau d'abstraction fait sur les entrées et sorties.

Intégrer le VHDL dans un environnement de co-simulation consiste alors à mettre en place ses interfaces.

En effet, la co-simulation est basée sur l'exécution conjointe de simulateurs, chaque simulateur représentant une partie spécifique du circuit à concevoir ou de son environnement. Une interface de co-simulation doit permettre l'échange de données entre les simulateurs tout en respectant les contraintes de types et de taille mais surtout en respectant la synchronisation temporelle des deux simulateurs. Le concepteur doit pouvoir tester son circuit, en reliant celui-ci avec d'autres simulateurs sans que cette interface n'intervienne dans les spécifications de son système.

Initialement, la co-simulation qui correspond à la simulation conjointe du logiciel et du matériel ne fait intervenir que des composants numériques. La connexion entre les simulateurs se fait de façon particulière en introduisant la notion de "*bus de co-simulation*".

Cependant, dans le cas des dispositifs de commande pour les systèmes électriques, l'ensemble des composants avec lesquels va interagir un CLP peut être divisé en deux parties : les composants numériques (tels que les mémoires et le microprocesseur) et les interfaces avec le monde analogique (tels que des CAN, des CNA's, des comparateurs, ...) qui permettent de dialoguer avec le procédé ou une partie du procédé à commander.

Dans le but d'utiliser la co-simulation pour la conception des dispositifs de commande ou simplement pour l'intégration des lois de commande, il était donc nécessaire de construire une interface spécifique entre un simulateur VHDL et un logiciel permettant de simuler le procédé à commander qui, dans notre cas, correspond au logiciel SABER.

Afin de situer l'interface que nous avons mis en place par rapport aux connexions habituellement développées en co-simulation, nous présentons dans une première partie une définition du "bus de co-simulation". Ensuite, nous détaillons le fonctionnement de l'interface qui à été développée, en présentant dans un premier temps les principales techniques informatiques permettant d'échanger des données entre processus.

4.3.1 Définition d'un bus de co-simulation

La communication entre deux simulateurs peut se faire selon deux modes différents [72][42]

– Le premier consiste à utiliser une relation de type maître-esclave entre les deux simulateurs (ils s'exécutent alors au sein d'un même processus au sens informatique du terme). Dans ce cas, l'un des simulateurs (par convention définit comme maître) appelle l'autre (alors défini comme esclave) par le biais d'une procédure (figure II.4.2.a). Ce type de communication peut s'avérer complexe car il est nécessaire que le simulateur esclave puisse sauver l'ensemble du contexte à la fin de son appel pour pouvoir repartir du même état (en rechargeant le contexte) au prochain appel (figure II.4.2.b). Le lancement du simulateur esclave, la sauvegarde et la restitution peuvent s'avérer très coûteux en terme de ressources et temps de simulation.

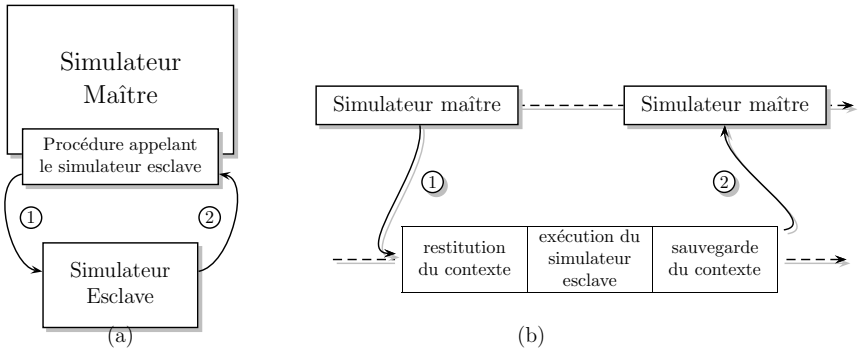


Figure II.4.2: *Représentation du mode de fonctionnement maître/esclave pour l'échange de données entre deux simulateurs*

– Le second mode consiste à utiliser un système de simulation distribuée. Les deux simulateurs fonctionnent en parallèle et s'échangent des informations via un canal de communication (figure II.4.3).

C'est ce deuxième mode de communication qui est généralement utilisé en co-simulation.

Dans le cadre de la simulation conjointe du logiciel et du matériel, on cherche à faire communiquer deux simulateurs reproduisant le fonctionnement de composants numériques qui peuvent avoir des niveaux d'abstractions plus ou moins élevés. Les échanges entre les simulateurs sont déclenchés par les composants simulés effectuant des demandes de lecture et d'écriture. Les procédures d'accès aux entrées/sorties utilisées dans les composants numériques simulés sont redéfinies, non plus pour communiquer avec un bus matériel mais, avec un canal de communication entre les simulateurs qui fait transiter les données et les informations permettant de savoir à quelles variables sont associées les valeurs. Autrement dit, les informations qui transitent par le canal de communication sont du même "type" que dans le cas d'un bus matériel mais avec des niveaux d'abstractions différents. Ce canal est alors nommé *bus de co-simulation*. Dans ce canal, il transite aussi des informations nécessaires à la synchronisation des différents simulateurs.

Dans notre cas nous faisons intervenir des composants analogiques, puisque nous voulons introduire un simulateur qui reproduit le fonctionnement du système sous contrôle (simulateur de type circuit). Il nous faudra alors utiliser des procédures d'échanges via le canal de communications qui ne sont plus basées uniquement sur les demandes de lecture et d'écriture mais aussi sur l'évolution des signaux analogiques.

En co-simulation, les simulateurs fonctionnent en parallèles. Ils sont considérés par un système d'exploitations comme des processus indépendants. Il faut donc pour établir une communication entre les différents simulateurs utiliser des techniques informatiques permettant de faire communiquer deux processus situés dans notre cas sur la machine hôte.

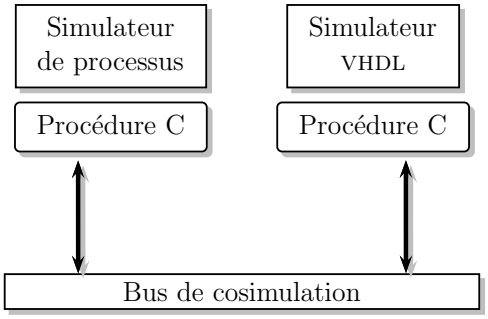


Figure II.4.3: *Utilisation d'un bus de co-simulation.*

4.3.2 Les différents types de communication inter-processus

La mise en place d'un canal de communication peut être faite en utilisant différentes technologies informatiques : on peut répertorier les mécanismes de communication inter-processus du système Unix, ainsi que le mécanisme des sockets et des RPC[2].

Les sockets peuvent être utilisées sur une station de travail unique mais il est aussi possible de mettre en place des communications réseaux (ethernet) utilisant des protocoles tels que UDP[3] ou TCP[4].

Les appels de procédures à distance (RPC) sont utilisées pour le réseau et permettent dans bien des cas de s'affranchir de la programmation bas niveau des couches réseau.

L'utilisation des protocoles réseaux a l'avantage d'être portable. Les sockets et les RPC existent sur les principaux systèmes d'exploitation et il est possible de réaliser des co-simulations en utilisant plusieurs stations. Cependant cela nécessite de mettre en place une structure logicielle plus complexe et les temps de communication sont plus long que les communications inter-processus du système Unix. Nous avons choisi d'utiliser ces dernières qui n'utilisent alors qu'une seule machine.

Parmi ces mécanismes, on répertorie les FIFO's, dit aussi tubes de communication, et les IPC[5] System V qui regroupent les sémaphores, les segments de mémoire partagée, et les files de messages.

- Les sémaphores sont des objets d'IPC utilisés pour synchroniser des processus entre eux. Ils constituent une solution pour résoudre le problème d'exclusion mutuelle et permettent en particulier de régler les conflits d'accès concurrents de processus distincts à une même ressource.
- La communication par messages s'effectue par échange de données stockées dans une boite aux lettres sous forme de files. Chaque processus peut émettre des messages et en recevoir.
- Le troisième type d'IPC correspond au partage de mémoires entre deux ou plusieurs processus. Il constitue le moyen le plus rapide d'échange de données. La zone de mémoire partagée est utilisée par chacun des processus comme si elle faisait partie de chaque programme. Le partage permet aux processus d'accéder à un espace d'adressage commun.

C'est cette dernière méthode que nous avons retenue et implantée pour faire communiquer le logiciel SABER et le simulateur VHDL (MODELSIM)

[2]Remote Procedure Call
[3]User Datagram Protocol
[4]Transmission Control Protocol
[5]Inter Process Communication

Principe de la mémoire partagée

Un processus commence par créer un segment de mémoire partagée et ses structures de contrôle. Lors de cette création, ce processus définit les droits d'opérations globaux pour le segment de mémoire partagée, définit la taille en octets, et a la possibilité de spécifier le mode d'accès d'un processus attaché à ce segment (lecture seule, écriture seule, ...).

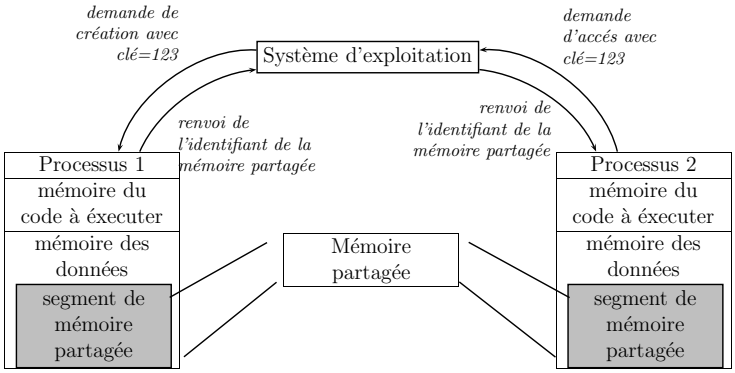


Figure II.4.4: *Principe de la mémoire partagée*

Si un autre processus veut s'attacher à ce segment mémoire (pour pouvoir lire et écrire), il doit avoir un identificateur de ce segment. Cet identificateur est fourni par le système d'exploitation, lorsque le processus lui demande, en utilisant une clé qui correspond à un nom numérique. Ce nom numérique est choisi arbitrairement, et est utilisé en premier lieu par le processus qui crée la mémoire partagée.

Une fois le processus attaché à un segment de mémoire, il reçoit un pointeur sur ce segment et peut l'utiliser comme s'il s'agissait d'un pointeur normal.

4.3.3 Mise en place de l'interface SABER ⇔ VHDL

Nous avons indiqué que nous voulions mettre en place un canal de communication entre le logiciel SABER et un simulateur VHDL afin de pouvoir utiliser la co-simulation dans la conception de dispositifs de commande.

Afin de rendre transparent la connexion entre SABER et le simulateur VHDL, nous avons cherché à faire apparaître l'entité VHDL sous la forme d'un composant SABER. En effet, l'environnement de simulation SABER permet de modéliser un système sous la forme d'un schéma. Celui-ci se dessine en connectant différents composants choisis au sein d'une bibliothèque ou défini par l'utilisateur. Pour définir le composant SABER représentant l'entité VHDL à connecter, nous avons créé un outil permettant de générer automatiquement le schéma et les connexions avec le simulateur MODELSIM. Nous détaillons, dans la suite les échanges de données nécessaires à la synchronisation des deux simulateurs et nous présentons par l'intermédiaire d'un exemple le fonctionnement de l'outil de génération automatique.

4.3.3.1 Principe d'échange des données entre SABER et VHDL

Durant une session de simulation SABER, le composant VHDL est vu comme un élément numérique écrit en MAST (figure II.4.5). Ce modèle numérique SABER est en fait une simple *"interface"* utilisant une procédure C_{saber} pour se connecter à un segment

de mémoire partagée. Cette interface récupère les signaux d'entrées pour les transmettre au simulateur VHDL d'une part, et met à jour les signaux de sorties venant de l'entité VHDL d'autre part.

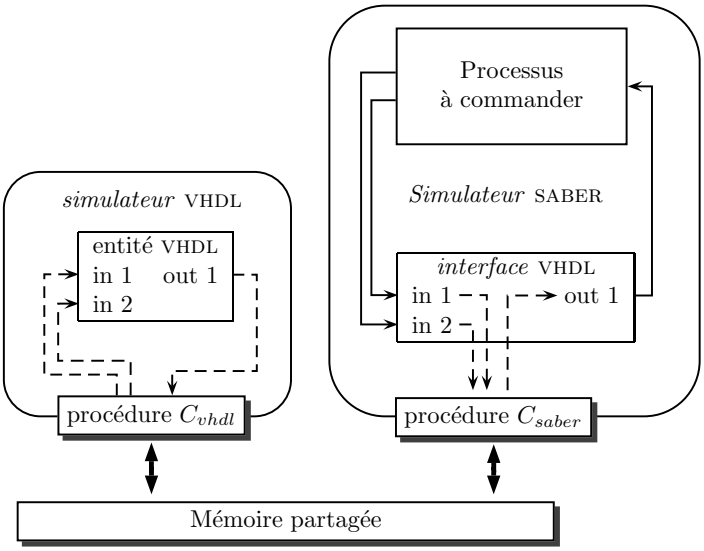


Figure II.4.5: *Principe de l'interface saber vhdl.*

Ceci est possible grâce au fait que le VHDL, dans la norme IEEE-1076'93, autorise l'appel de fonctions écrites dans un langage étranger (foreign function). Ainsi une procédure C_{vhdl} est connectée à l'entité VHDL que nous voulons simuler et permet de communiquer avec la mémoire partagée.

Étant donné que les signaux venant de SABER sont aussi analogiques, nous devons tenir compte de ce type de grandeurs qui évoluent de manière continue dans le temps. Celles-ci sont calculées par des algorithmes à pas variable dans le cas du logiciel SABER.

4.3.3.2 Synchronisation

Dans le cas de co-simulations ne faisant intervenir que des composants numériques, l'échange des données entre la partie logicielle, généralement réalisée en C, et la partie matérielle, réalisée en VHDL, se fait via un *bus de co-simulation*. Dans [72], il est présenté plusieurs techniques de synchronisation utilisant un tel bus de co-simulation et permettant d'exploiter l'exécution parallèle des simulateurs. La figure II.4.6.a présente un mode de communication où l'échange des données ne s'effectue que lors de l'appel de procédures d'entrées/sorties. Un simulateur envoie des données lorsque cela est nécessaire et se met en attente pour recevoir un accusé de réception de la part de l'autre simulateur.

La figure II.4.6.b présente un autre mode de communication où les différentes requêtes sont retenue dans des files de messages (FIFO) diminuant ainsi les instants d'attentes possibles.

On constate que ces modes de communication ne peuvent être mis en place que lorsqu'il y a demande de lecture et demande d'écriture de la part des composants.

Dans notre cas, la synchronisation des simulateurs ne peut pas exploiter les échanges de données mis en place dans le cadre de bus de co-simulation.

La simulation des grandeurs analogiques à une date t ne peut se faire correctement qu'en tenant compte des évènements discrets venant du VHDL à cette date. De même, le VHDL doit tenir compte des variations des entrées venant de SABER. Les simulateurs

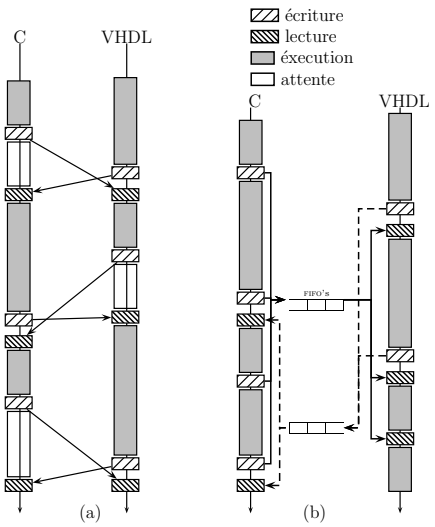


Figure II.4.6: *Différentes synchronisation possibles en utilisant un bus de co-simulation classique*

ne peuvent avancer dans leur calculs respectifs que si ils "connaissent" à quel date aura lieu le prochain événement associé à l'autre simulateur.

Dans ces conditions, il semble que la synchronisation entre les deux simulateurs ne puisse se faire que selon deux possibilités. La première consiste à imposer des points de rendez vous discrets (à pas fixe) pour mettre à jour les signaux échangés par les deux simulateurs. Cette date doit alors être adaptée à la fréquence de variation des entrées/sorties de l'entité VHDL. Cette première méthode nécessite, soit l'intervention de l'utilisateur qui doit déterminer la valeur de ce pas de discrétisation, et dans ce cas une erreur dans le choix de ce pas pourrait rendre le résultat de la co-simulation caduc, soit le pas de discrétisation est fixé sur le pas de résolution de la simulation VHDL et dans ce cas on peut avoir des échanges inutiles entre les deux simulateurs.

La seconde possibilité, que nous avons adoptée, est d'utiliser le fait que le simulateur VHDL est un simulateur à évènements discrets. Il a donc un échéancier contenant la date des différents évènements à venir. Il est alors possible de le scruter afin de connaître la date du prochain évènement VHDL en attente, s'il en existe.

Cette synchronisation se fait alors en échangeant un certain nombre d'informations par le biais de la mémoire partagée. Celle-ci est considérée comme un tableau contenant les variables suivantes :

La figure II.4.7 montre alors le protocole de synchronisation mis en place.

Cette exemple montre le principe en cours de fonctionnement

Saber indique au simulateur VHDL (MODELSIM), par l'intermédiaire de la variable *temps_saber*, jusqu'à quelle date (t_1) il a effectué la simulation (étapes ② → ③) et demande au simulateur VHDL d'avancer son temps de simulation jusqu'à cette date (étapes ③ → ④). Cette date t_1 est soit :

1. associée à un événement sur une des entrées de l'entité,
2. associée à un événement VHDL qui était contenu dans l'échéancier de MODELSIM et qui a été indiqué à SABER. (comme dans les étapes ⑤ → ⑥)

Ce mode de fonctionnement permet alors de rendre la communication totalement indépendante du système à simuler. Cependant, il s'agit d'un mode de communication qui ne profite pas totalement du fait que chaque simulateur est un processus à part entière puisque lorsqu'un simulateur effectue ses calculs, l'autre est en attente.

nom de variable	rôle de la variable
in_k	k signaux d'entrées de l'entité VHDL
out_j	j signaux de sorties de l'entité VHDL
temps_saber	date à laquelle les signaux d'entrées de l'entité VHDL (venant de saber) peuvent être pris en compte par le VHDL
flag	variable jouant le rôle de sémaphore pour l'accès au segment de mémoire
proch_evt_vhdl	date du prochain événement VHDL utilisé par SABER
status	SABER indique si on est en mode initialisation, calcul ou fin de simulation

Tableau II.4.1: *Les différentes variables en mémoire partagée.*

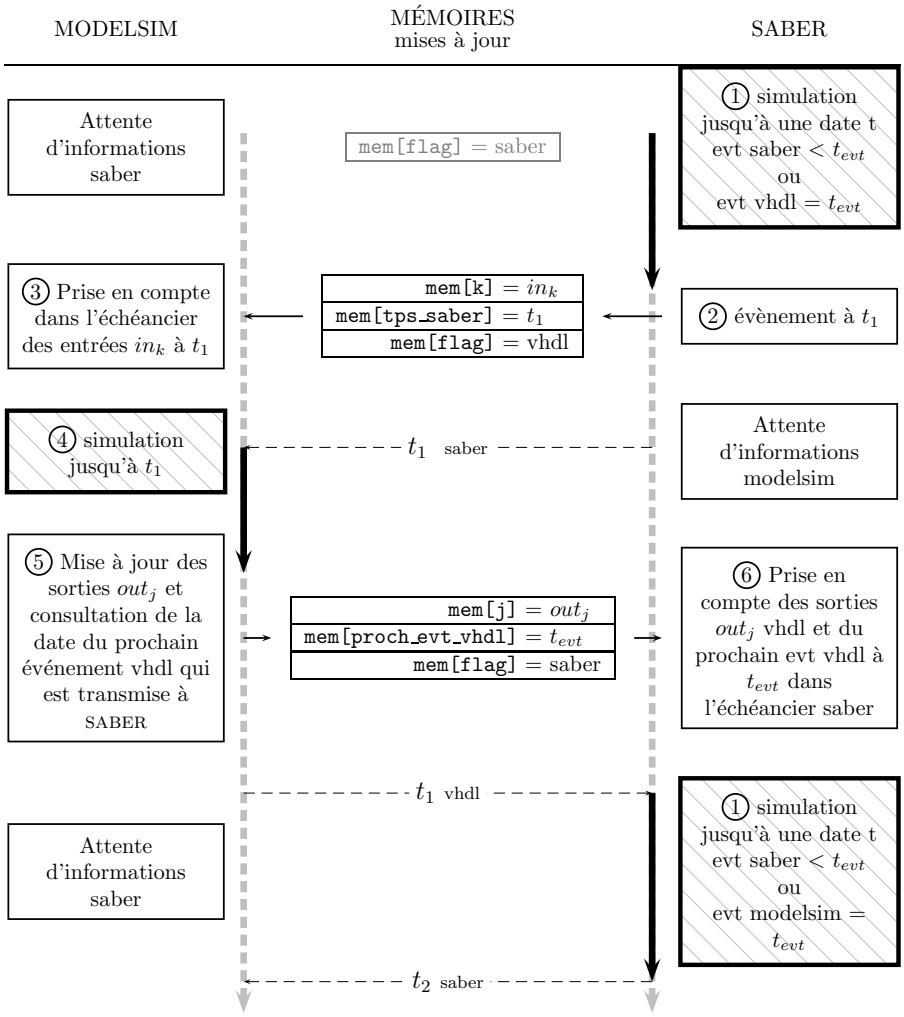


Figure II.4.7: *Synchronisation des échanges de données entre saber et vhdl.*

4.3.3.3 Génération automatique de l'interface

Afin d'intégrer et de rendre transparent la connexion entre SABER et MODELSIM, nous avons voulu intégrer l'utilisation de l'entité VHDL à tester dans un schéma SABER. Pour réaliser cette intégration, la vue externe de l'entité VHDL définie par ses entrées et

ses sorties, doit être déclarée comme un composant SABER. Pour définir un tel composant, il nous faut créer un premier fichier, que l'on nommera "template", écrit en MAST et décrivant son fonctionnement, puis un deuxième fichier contenant les informations nécessaires à sa représentation graphique. Dans notre cas, le "template" ne va pas être utilisé pour décrire le fonctionnement du composant mais pour synchroniser les accès aux entrées/sorties du composant VHDL par l'intermédiaire de la procédure C_{saber}, celle-ci étant attachée au segment de mémoire partagée (figure II.4.5).

Pour échanger correctement l'ensemble des informations, les deux procédures C_{saber} et C_{vhdl} doivent connaître, avant le début de la co-simulation, le nom des différents signaux d'entrées et de sorties de l'entité VHDL et leurs places dans la mémoire partagée. Pour cela nous avons recours à un fichier nommé "mem.desc" qui énumère les entrées et les sorties de l'entité. À l'initialisation de la co-simulation, du côté VHDL, la procédure C_{vhdl} lit ce fichier mem.desc, et crée les liaisons adéquates entre l'entité VHDL et le segment de mémoire partagée.

Nous pouvons donc constater que pour intégrer un composant VHDL dans une simulation SABER, il nous faut générer un certain nombre de fichiers qui dépendent de la structure de l'entité en terme d'entrées/sorties. Ces fichiers sont :

- un template permettant de faire le lien entre le schéma SABER et la procédure C_{saber} ;
- Un fichier contenant la description graphique du composant sous SABER
- un fichier mem.desc permettant à la procédure C_{vhdl} de connaître la place des signaux de l'entité VHDL dans la mémoire partagée.

Afin de ne pas avoir à créer manuellement l'ensemble de ces fichiers à chaque fois, nous avons mis en place un outil de génération automatique. Cet outil analyse l'entité VHDL à connecter pour déterminer quelles sont les entrées et les sorties. Une fois l'analyse de l'entité effectuée, l'outil génère automatiquement les différents fichiers cités précédemment. Nous allons illustrer l'utilisation de l'outil d'analyse par l'intermédiaire d'un exemple : une simple porte "and", dont l'entité VHDL est spécifiée comme suit,

code 4.4: *Entité spécifiant une porte and*

```
library IEEE;
use IEEE.std_logic_1164.all;
-- déclaration de l'entite
entity and_gate is
  port(in1, in2 : in std_logic; out1 : out std_logic);
end;
```

Cette entité comprend donc deux entrées de type `std_logic` et une sortie de même type.

L'outil d'analyse crée les fichiers suivants :

- and_gate.sin : template MAST, représentant le fonctionnement de la "coquille" sous SABER et définissant les appels à la fonction C pour synchroniser les entrées et sorties avec le VHDL.
- and_gate.c : procédure C_{saber} qui réalise le lien entre SABER et la mémoire partagée.
- mem.desc : fichier définissant l'ordre des signaux de l'entité VHDL dans la mémoire partagée.

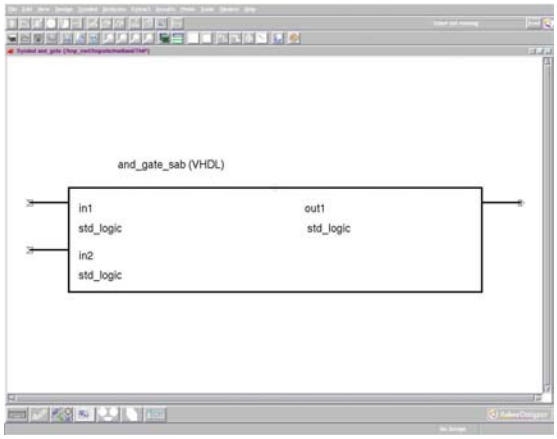


Figure II.4.8: *Représentation de la porte and sous SABER (fichier* and_gate.ai_sym*).*

```
# Do not Modify this file#
-REGION -------------------------
/and_gate_saber/instance_of_and_gate;
-INPUT --------------------------
0=in1;
1=in2;
-OUTPUT-------------------------
2=out1;
-INPUT-OUTPUT-------------------------
-CONTROL-----------------------
3=[time];
4=[time_for_output];
5=[flag];
6=[proch_evt_modelsim];
7=[evt saber / modelsim + fin + init];
# End on Mon Jul  8 10:29:03 2002 #
```

– and_gate_sab.vhd : fichier VHDL attachant la procédure C_{vhdl} à l'entité `and_gate`.
– and_gate.ai_sym : fichier représentant le schéma SABER (figure II.4.8) de la porte and.

Le composant VHDL est alors directement utilisable sous SABER comme un de ses propres composants et son utilisation ne nécessite aucune manipulation particulière.

Cette interface peut être utilisée pour valider les spécifications VHDL en effectuant une simulation de premier niveau. Mais il est aussi possible d'effectuer des simulations de second niveau, c'est à dire après synthèse logique, et de troisième niveau (après placement et routage), afin de valider chaque étape dans la conception de la logique câblée. Nous allons tout d'abord présenter plusieurs essais en utilisant des simulations de premier niveau. Mais nous verrons dans le chapitre 6 que nous avons aussi réalisé des co-simulations en simulant la partie VHDL après placement et routage.

4.4 Application au contrôle d'un convertisseur multicellulaire

Pour illustrer le principe de la co-simulation, nous avons choisi l'implantation d'une loi de commande non-linéaire appliquée à un convertisseur multicellulaire [50]. Ce convertisseur est représenté sur la figure II.4.9 et correspond à un hacheur série à 4

niveaux de tensions. Cet exemple n'ayant pas pour but d'étudier précisément la commande d'un convertisseur multicellulaire, nous renvoyons le lecteur à [26] dans lequel le fonctionnement d'un tel convertisseur est présenté (Nous présenterons cependant plus de détails sur certains aspects des multicellulaire dans le chapitre 5 et 6). Cet exemple a pour but de valider les interfaces mises en place mais aussi de montrer le type de co-simulation qui peut-être effectué.

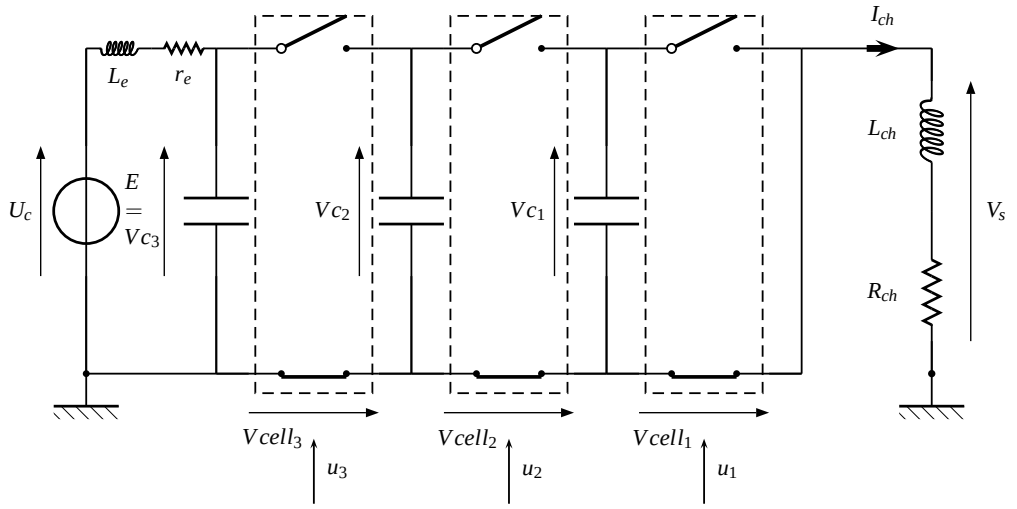


Figure II.4.9: *Hacheur à 3 cellules de commutations.*

La commande d'un tel convertisseur a en charge de réguler la tension moyenne aux bornes des capacités flottantes Vc_1 et Vc_2 afin que celles-ci soient toujours respectivement égales à $E/3$ et $2.E/3$. Dans le cas contraire, il apparaît des surtensions aux bornes des interrupteurs de puissance qui peuvent être fatales à leur fonctionnement. La commande devra aussi réguler le courant moyen I_{ch}. Pour présenter la loi de commande que nous allons appliquer, nous rappelons succinctement la modélisation d'un tel convertisseur.

4.4.1 Modélisation du système

La figure II.4.10, représentant deux cellules de commutation connectées à une capacité flottante, nous permet d'introduire les notations que nous allons utiliser.

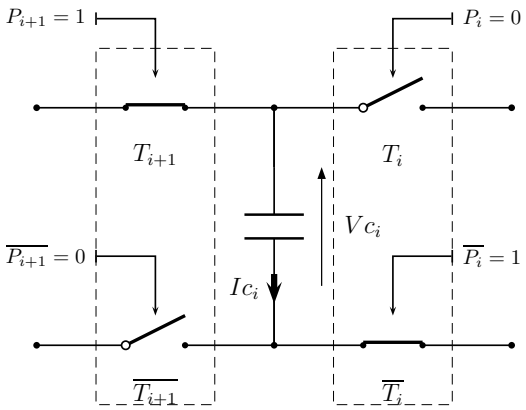


Figure II.4.10: *Capacité flottante et cellules de commutations.*

P_i représente la commande de l'interrupteur du haut de la cellule de commutation i ($\overline{P_i}$ représentant la commande complémentaire envoyée à l'interrupteur du bas). Elle vaut 1 lorsque l'interrupteur est fermé et 0 lorsqu'il est ouvert.

Par l'intermédiaire de ces notations, sachant que dans une cellule de commutation il y a toujours un et un seul interrupteur fermé, la tension aux bornes des capacités flottantes peut s'exprimer en fonction du courant de charge et de l'état des cellules de commutation qui lui sont connectées.

$$Ic_i = (P_{i+1} - P_i).I_{ch} \tag{II.4.1}$$

Ainsi l'évolution du courant dans les capacités flottantes s'exprime suivant la relation II.4.3

$$Ic_i = C_i.\frac{dVc_i}{dt} \tag{II.4.2}$$

$$\dot{V}c_i = \frac{1}{C_i}.(P_{i+1} - P_i).I_{ch} \tag{II.4.3}$$

Si les commandes u_1, u_2, u_3 (nommées rapport cycliques) représentent les valeurs moyennes des commandes respectives P_1, P_2, P_3 sur une période T_d appelée période de découpage,

$$u_i = \frac{1}{T_d}\int_0^{Td} u_i.dt \tag{II.4.4}$$

alors, le comportement aux valeurs moyennes d'un hacheur 3 cellules est décrit par le système d'équations suivant :

$$u_{\{1,2,3\}} \in [0\dots 1] \begin{cases} \dot{v}c_1 = \dot{x}_1 = -a_1.x_3.(u_1 - u_2) \\ \dot{v}c_2 = \dot{x}_2 = -a_2.x_3.(u_2 - u_3) \\ i_{charge} = \dot{x}_3 = -b_0.x_3 + b_1.V_s \end{cases} \tag{II.4.5}$$

où les variables sont : $\begin{pmatrix} x_1 = vc_1 \\ x_2 = vc_2 \\ x_3 = i_{load} \end{pmatrix} = X$ $\quad a_1 = \frac{1}{C_1} \quad a_2 = \frac{1}{C_2}$ $\quad b_0 = \frac{R}{L} \quad b_1 = \frac{1}{L}$

et $V_s = Vcell_1 + Vcell_2 + Vcell_3 = u_3.(E - Vc_2) + u_2.(Vc_2 - Vc_1) + u_1.Vc_1$

Le système peut être représenté sous la forme d'un modèle affine non-linéaire donné dans l'équation (II.4.6) et basé sur les grandeurs moyennes du système.

$$\dot{X} = f(\mathrm{X}) + \sum_{j=1}^{3} g(\mathrm{X})u_j \tag{II.4.6}$$

avec :

$$f(\mathrm{X}) = \begin{bmatrix} 0, & 0, & -b_0 x_3 \end{bmatrix}^t \tag{II.4.7}$$

$$g_{h3}(\mathrm{X}) = \begin{bmatrix} -x_3\, a_1 & x_3\, a_1 & 0 \\ 0 & -x_3\, a_2 & x_3\, a_2 \\ b_1 x_1 & b_1(x_2 - x_1) & b_1(e_c - x_2) \end{bmatrix} \tag{II.4.8}$$

La matrice $g_{h3}(X)$ se décompose en 3 colonnes. Chacune de ces colonnes représente une fonction $g_{h3_j}(X)$ pour $j \in \{1,2,3\}$. Ainsi la fonction $g_{h3}(X)$, s'écrit :

$$g_{h3} = [g_{h3_1}(X), g_{h3_2}(X), g_{h3_3}(X)] \tag{II.4.9}$$

Cette représentation montre que les non-linéarités qui interviennent sont simplement dues au couplage des commandes.

4.4.2 Loi de commande appliquée

À la représentation affine non-linéaire, il peut être appliqué une méthode de linéarisation exacte permettant de découpler les variables d'état du système. Cette commande a été étudiée dans [26].

Il apparaît un nouveau vecteur d'entrée V relié au vecteur u, composé des trois rapport cycliques, par la relation II.4.10.

$$u(X) = \alpha(X) + \beta(X).V \tag{II.4.10}$$

avec :

$$\alpha(\mathrm{X}) = \begin{pmatrix} \frac{-b_0 x_3}{b_1 e_c} \\ \frac{-b_0 x_3}{b_1 e_c} \\ \frac{-b_0 x_3}{b_1 e_c} \end{pmatrix} \quad \text{et} \quad \beta(\mathrm{X}) = \begin{pmatrix} \frac{x_1 - e_c}{a_1 x_3 e_c} & \frac{x_2 - e_c}{a_2 x_3 e_c} & \frac{1}{b_1 e_c} \\ \frac{x_1}{a_1 x_3 e_c} & \frac{x_2 - e_c}{a_2 x_3 e_c} & \frac{1}{b_1 e_c} \\ \frac{x_1}{a_1 x_3 e_c} & \frac{x_2}{a_2 x_3 e_c} & \frac{1}{b_1 e_c} \end{pmatrix}$$

Le découplage permet alors d'avoir une grandeur de commande V_i associée à chaque grandeur à réguler Y_i (cf. Figure II.4.11).

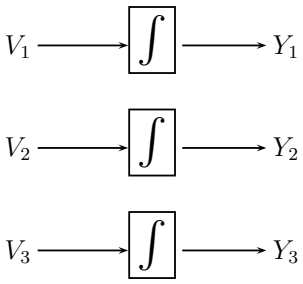


Figure II.4.11: *Représentation du système découplé.*

Ce découplage rend possible l'application sur celui-ci d'un second bouclage linéaire qui permet d'imposer la dynamique voulue sur les grandeurs à réguler que sont le courant de charge et les tensions des capacités flottantes.

Les tâches que doit effectuer un dispositif de commande pour réaliser une telle commande sont donc les suivantes :

1. Acquérir les tensions flottantes Ec, Vc_1, Vc_2, ainsi que le courant de charge I_{ch}
2. Calculer le vecteur de référence X_{ref} (puisque $Vc_{ref_i} = i.E_c/3$)
3. Calculer le vecteur V en utilisant un régulateur proportionnel,
4. Calculer le vecteur des rapports cycliques en effectuant le calcul décrit dans l'équation II.4.10.
5. Calculer les ordres de commande associés aux trois cellules par Modulation de Largeur d'Impulsion.

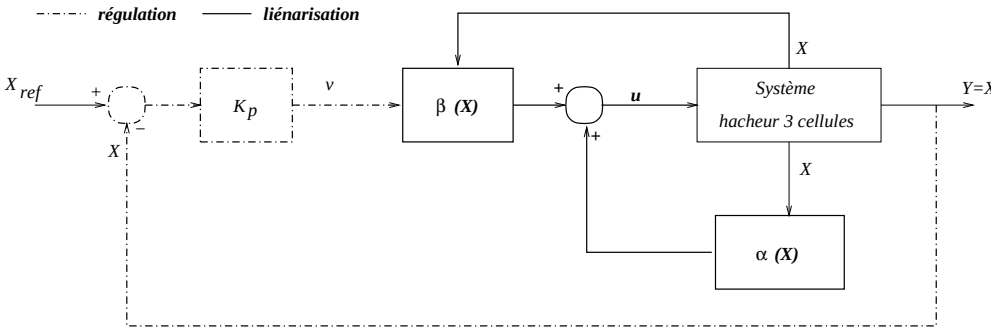


Figure II.4.12: *Boucle de linéarisation et de régulation de la commande.*

La période d'échantillonnage T_e de la commande étant calée sur la période de découpage T_d, l'ensemble des tâches allant de 1 à 4 doivent être réalisées en un temps inférieur à cette période.

4.4.3 Implantation de la loi de commande en co-simulation

Nous allons évaluer, par co-simulation, plusieurs paramètres associés à l'implantation d'une telle loi de commande. L'architecture sur laquelle sera implantée la loi de commande est représenté sur la figure II.4.13. Ce dispositif est constitué d'un microprocesseur et d'un FPGA. Ce dernier assure le dialogue avec les différents CANs et génère les ordres de commande des interrupteurs de puissance.

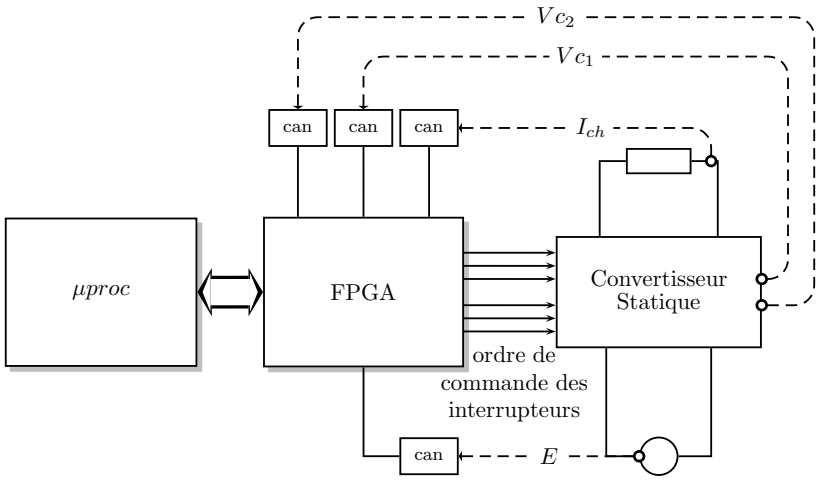


Figure II.4.13: *Architecture du dispositif de commande.*

Pour évaluer une telle architecture, nous devons spécifier en C les différentes tâches qui devront être calculées par le microprocesseur et en VHDL, celles que devra exécuter le FPGA. Le test des différents programmes, doit alors utiliser un modèle de microprocesseur pour les tâches décrites en C et l'interface SABER/MODELSIM pour les tâches décrites en VHDL.

Cependant, le but de cet exemple est essentiellement de valider l'outil de co-simulation entre SABER et MODELSIM, et les modèles de microprocesseur détaillés sont actuellement peu adaptés pour être intégrés dans un environnement de co-simulation. Nous avons donc choisi un modèle de microprocesseur au niveau bus tel que nous l'avons présenté au chapitre précèdent au paragraphe 3.2.4 page 50.

L'utilisation d'un modèle de microprocesseur, au niveau bus, ne prend pas en compte

l'architecture interne d'un microprocesseur, mais permet de simuler les cycles de lecture et d'écriture. Ainsi, le code de commande 'C' utilisé en co-simulation peut être écrit comme s'il allait s'exécuter sur le microprocesseur cible puisque les procédures d'entrées/sorties sont aussi simulées. De plus, les signaux de bus issus d'un tel modèle peuvent permettre de valider la partie VHDL en lien avec le microprocesseur qui est généralement difficile à mettre au point sur le dispositif expérimental.

Une tel système de co-simulation : "*modèle de microprocesseur au niveau bus + simulateur* VHDL *+ simulateur système (*SABER*)*", peut dans un premier temps nous permettre de valider la synthèse du code VHDL et du code C, et de tester des répartitions différentes de l'ensemble des tâches entre le microprocesseur et le FPGA.

4.4.3.1 Modélisation du microprocesseur au niveau bus

Suite à l'expérience acquise sur les communications inter-processus lors de l'élaboration de l'interface SABER/MODELSIM, nous avons mis en place un véritable bus de co-simulation entre SABER et un programme C censé exécuter une routine d'interruption.

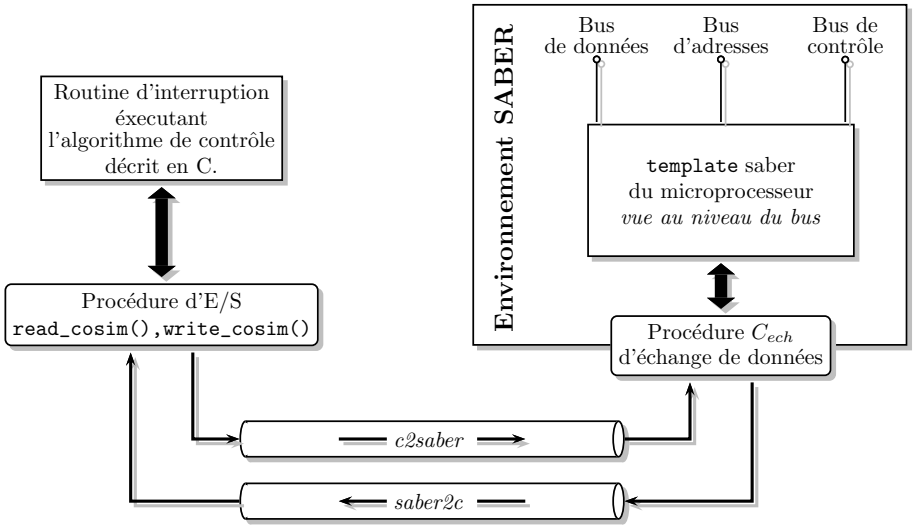


Figure II.4.14: *Principe du modèle de microprocesseur au niveau bus.*

Ainsi, même si nous n'avons pas de simulateur de microprocesseur permettant de prendre en compte son architecture et d'estimer le temps de calcul, nous pouvons implanter la routine de régulation qui devra s'exécuter sur le futur dispositif de commande et valider l'écriture du code. Ce dernier pourra d'ailleurs être directement implanté sur le dispositif expérimental puisqu'à ce niveau, le simulation est entièrement transparente.

Structure de l'interface

L'objectif de la co-simulation étant de mettre au point la commande du procédé, nous n'avons considéré que les tâches de régulation qui s'exécutent en temps réel et qui sont les plus difficiles à mettre au point expérimentalement. Pour cela, nous avons considéré que le microprocesseur n'exécutait qu'une routine active par interruption.

Le bus de co-simulation (figure II.4.14) a été réalisé en utilisant deux tubes de communications unidirectionnel de type FIFO que l'on nomme *c2saber* et *saber2c*.

Par l'intermédiaire de ces tubes, une procédure C_{ech} reliée à SABER communique avec les procédures d'entrées/sorties utilisées par la routine d'interruption. Cette procédure C_{ech} est elle même reliée à un "template" SABER. On retrouve, en fait, du côté SABER, une structure semblable à l'interface utilisée avec le VHDL.

Cependant, le template est relativement générique puisqu'il représente les trois bus matériels habituellement présents sur un microprocesseur : le bus de données, le bus d'adresse et le bus de contrôle.

Échanges des données

Comme sur le microprocesseur cible, la routine d'interruption écrit ou lit une donnée associée à un périphérique par l'intermédiaire des procédures d'entrées/sorties.

En effet, dans le cas de la co-simulation, tous les accès aux périphériques effectués par la routine utilisent une fonction `read_cosim(adresse,donnée)` pour les lectures et `write_cosim(adresse,donnée)` pour les écritures. Ces requêtes sont, alors, envoyées sur le bus de co-simulation qui les transmet au logiciel SABER. L'interface au sein de SABER recevant les requêtes de lecture et d'écriture les interprète et génère en conséquence les signaux sur le bus de contrôle, d'adresse et de données.

Un exemple de dialogue entre les procédures d'entrées/sorties et la procédure C_{ech} est représenté sur la figure II.4.15.

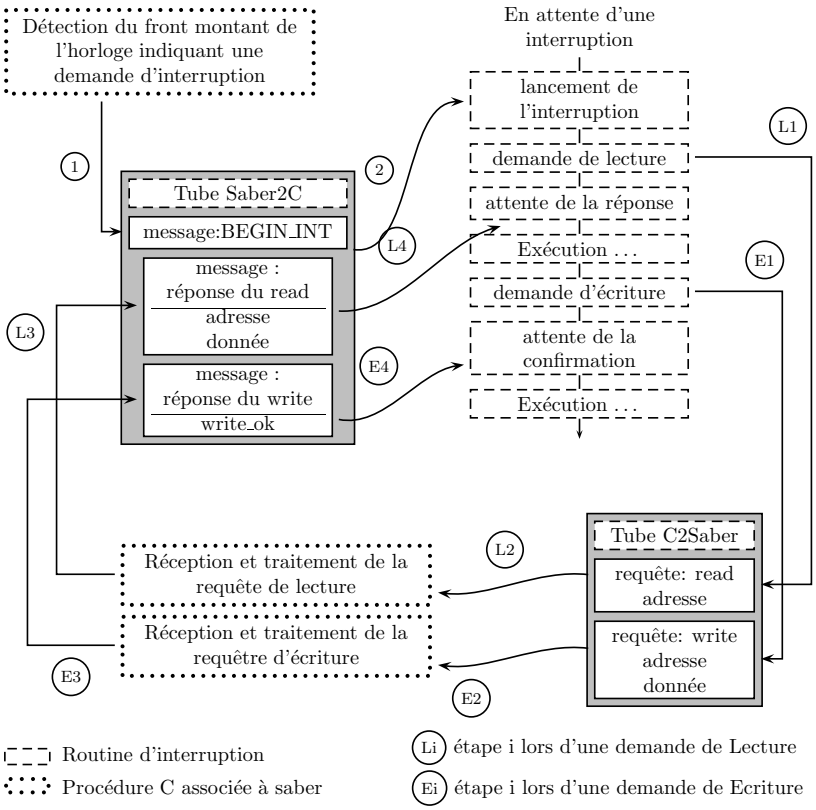


Figure II.4.15: *Dialogue effectué entre la routine d'interruption et la procédure C sous SABER*

Cette figure présente le déclenchement de la routine d'interruption qui fait alors une requête de lecture, puis une seconde requête qui correspond à une écriture. Par ces exemples, la figure permet de visualiser le protocole de communication que nous avons mis en place entre les procédures d'entrées/sorties qui envoient des requêtes dans le

tube *c2saber* et la procédure C_{ech} qui répond à ces requêtes par des messages envoyés dans le tube *saber2c*.

La figure II.4.16 situe les évènements de manière temporelle, entre les requêtes effectuées par la routine d'interruption et les signaux générés par le composant SABER représentant le microprocesseur au niveau bus.

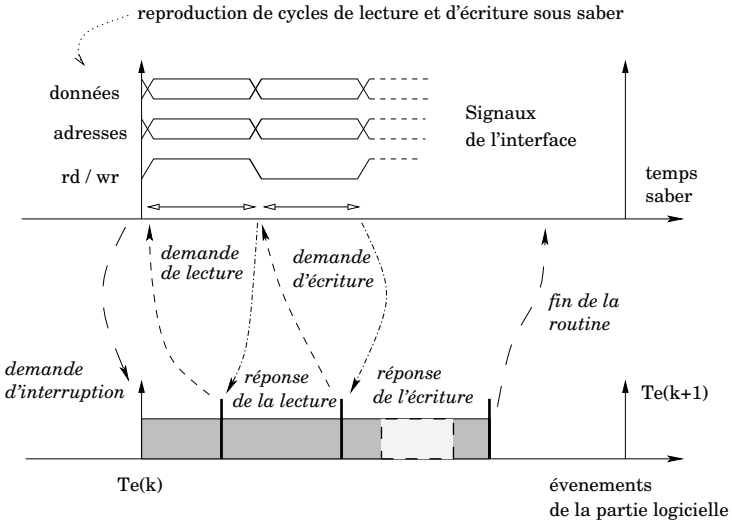


Figure II.4.16: *Visualisation des évènements dans le temps entre la routine d'interruption et les signaux de bus de l'interface saber*

Ce diagramme montre que lorsqu'une lecture est demandée, le signal de contrôle $rd/\overline{wr}$ reste en position haute, et l'adresse de la donnée demandée est écrite sur le bus d'adresse. Lorsqu'une écriture est demandée, le signal $rd/\overline{wr}$ passe en position basse et la donnée ainsi que son adresse sont positionnées sur les bus correspondants.

Cet exemple représente simplement le principe. Les signaux générés par les bus sont en effet, complètement configurables. Ainsi ce modèle de microprocesseur peut être paramétré pour représenter le protocole de lecture et d'écriture associé à un microprocesseur particulier et peut ainsi permettre la mise au point des dialogues avec celui-ci.

Dans l'environnement SABER, il serait aussi possible de connecter les signaux de bus à des composants numériques tels que des CAN, CNA, ... Pour notre exemple, ces signaux seront reliés au composant SABER représentant le FPGA.

Comme nous l'avons dit précédemment, la routine d'interruption codée en C est directement implantable sur le dispositif de commande expérimental, si ce n'est qu'il faut redéfinir les deux fonctions `read_cosim()` et `write_cosim()`.

4.4.3.2 Mise en place de l'architecture dans l'environnement de co-simulation

Le modèle de microprocesseur que nous venons de voir est utilisé pour évaluer l'architecture présenté en figure II.4.13. Un premier partitionnement des tâches énumérées en 4.4.2 a été mis en place. Ce partitionnement est représenté en figure II.4.17.

La routine d'interruption implantée sur le microprocesseur calcule la régulation proportionnelle et le découplage non linéaire.

Le FPGA transmet la valeurs des CAN lorsque le microprocesseur le lui demande, et calcule les ordres de commande à imposer aux interrupteurs de puissance en fonction des rapport cycliques (u_i) transmis par la routine d'interruption.

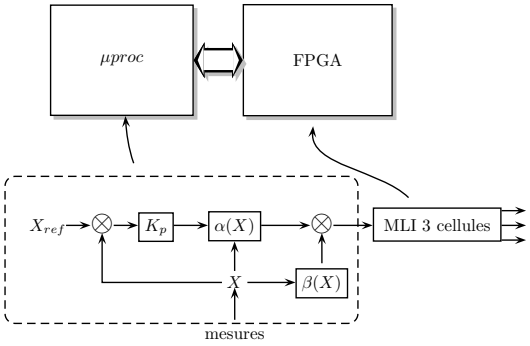


Figure II.4.17: *Premier "partitionnement".*

Il utilise pour cela une méthode de Modulation de Largeur d'Impulsion (MLI) présenté plus loin.

La figure II.4.18 présente la structure écrite en VHDL permettant de simuler les CAN et le CLP. Elle montre aussi la liaison avec les différents bus du microprocesseur (*rd/wr,adresse,data*). Les autres entrées de l'entité VHDL sur le schéma sont reliées au procédé à commander par l'intermédiaire de capteurs, et les sorties correspondent aux commandes à imposer aux interrupteurs de puissance du convertisseur.

La logique introduite dans le CLP est divisée en deux modules. Un premier module dialogue avec les CAN et le microprocesseur (module *décodeur d'adresse* sur la figure II.4.18). Un second module calcule les ordres de commande par MLI en utilisant les valeurs des rapports cycliques que le premier module lui transmet.

Le fonctionnement global de la commande est le suivant : lorsque la routine d'interruption est appelée, celle-ci fait une demande de conversion au FPGA en écrivant dans un registre particulier de celui-ci. Le FPGA transmet alors cette demande aux différents convertisseurs analogiques numériques. Dans les simulations effectuées, nous n'avons pas considéré de temps de conversion pour les CAN. Ainsi, une fois que la routine a demandé la conversion, les données sont immédiatement accessibles. La routine effectue donc 4 lectures correspondant aux 4 registres contenant les valeurs de E, Vc_1, Vc_2, I_{ch} au format numérique de 8 bits. Comme nous ne prenons pas en compte les temps de calcul, les valeurs des rapports cycliques calculés sont directement renvoyées aux FPGA. Un exemple de ces échanges est visible sur la figure II.4.19

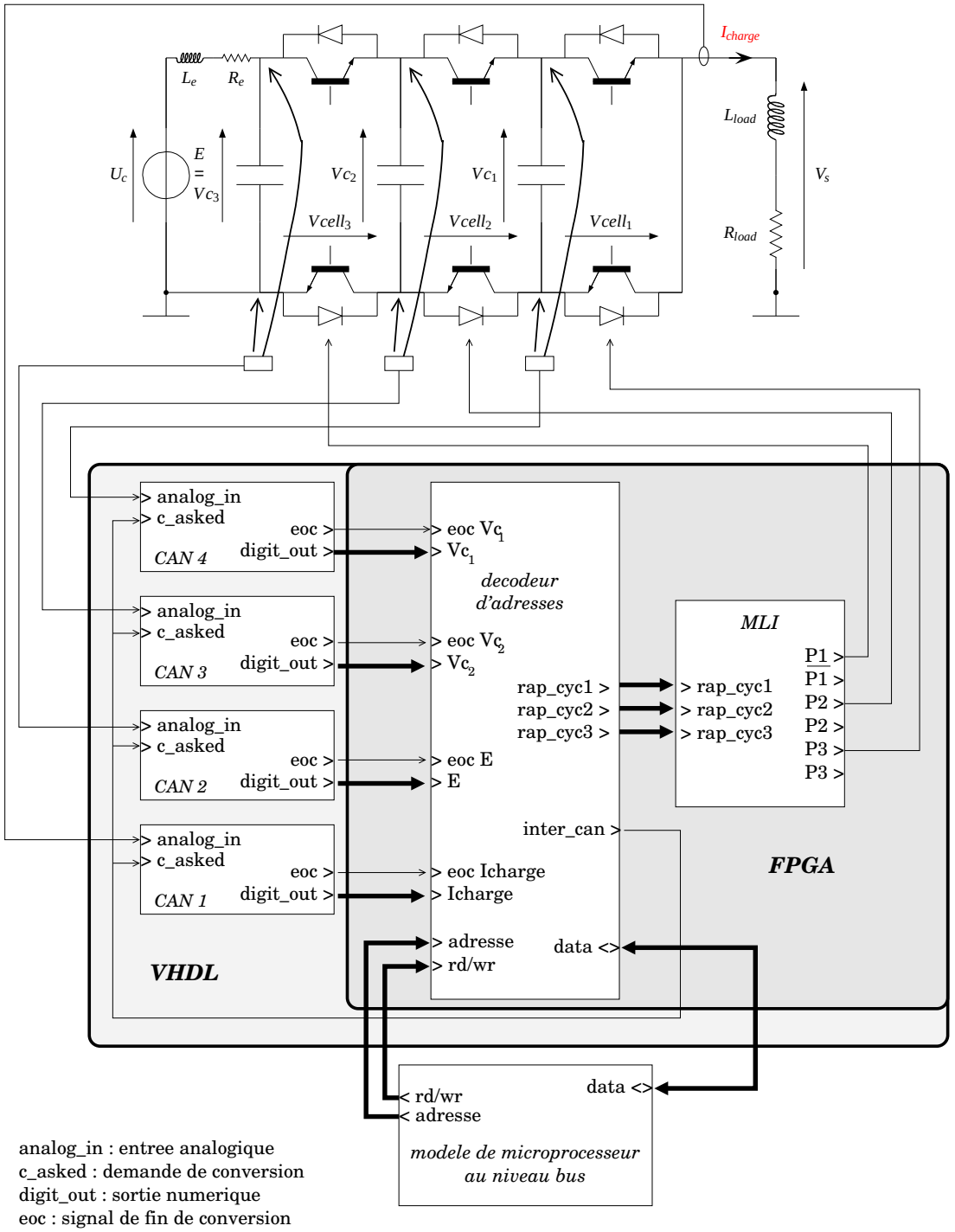


Figure II.4.18: *Architecture co-simulée.*

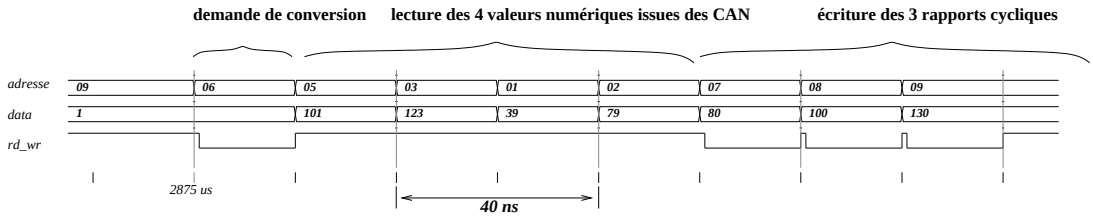


Figure II.4.19: *Cycles d'échanges de données entre la routine d'interruption et le FPGA*

4.4.3.3 Influence de la résolution de la MLI

On a donc, avec cette répartition des tâches, principalement deux modules à implanter dans le FPGA : le décodeur d'adresse et le générateur de signaux de commande par MLI.

Le générateur de signaux doit calculer les instants auxquels il faut fermer et ouvrir un interrupteur de puissance. Ce calcul se fait en fonction du rapport cyclique (u_i) qui correspond à la valeur moyenne du temps de fermeture de l'interrupteur de puissance sur une période de découpage (T_d).

$$u_i = \frac{1}{T_d}\int_0^{T_d} P_i.dt = \frac{t_{on}}{T_d} \tag{II.4.11}$$

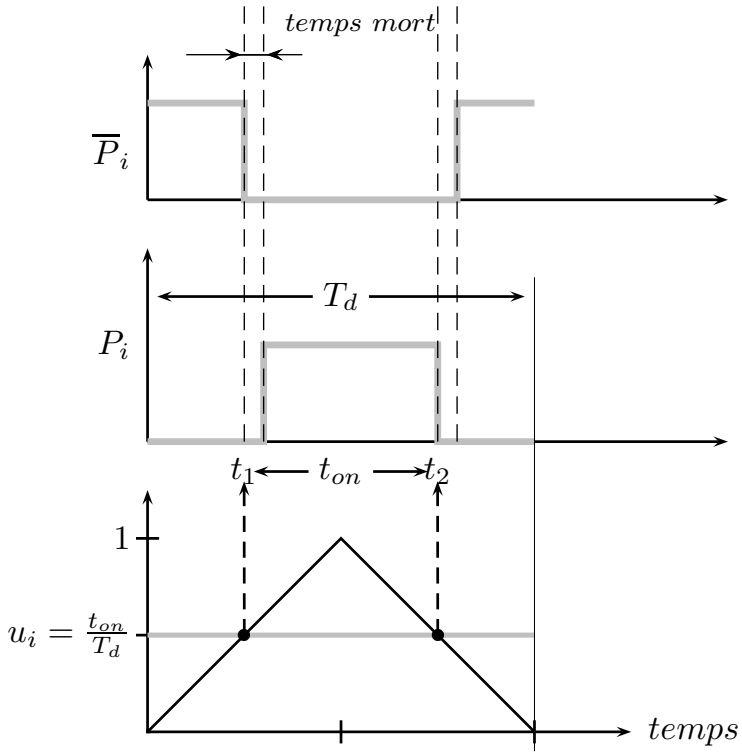


Figure II.4.20: *Calcul des instant de commutations par MLI régulière symétrique.*

Le calcul réalisé habituellement pour déterminer les instants de commutation est montré en figure II.4.20. Il consiste à générer une dent de scie (porteuse) périodique variant entre 0 et 1 et de période T_d, et de comparer cette dent de scie au rapport cyclique. L'intersection des deux courbes indique alors un instant de commutation.

Cependant, il faut insérer des temps mort afin de ne pas créer de court-circuit.

La réalisation numérique d'une commande par MLI consiste donc à générer une dent de scie à l'aide d'un compteur puis à comparer la sortie de ce compteur au rapport cyclique.

La numérisation discrétise la valeur du rapport cyclique et par conséquent la dent de scie. Les instants de commutations sont alors quantifiés sur la période de découpage. La précision de la commande dépend donc du nombre de bits utilisé pour coder le rapport cyclique.

En effet, si on utilise seulement 4 bits pour coder le rapport cyclique et que la période de découpage est de $50\,\mu s$, le rapport cyclique ne pouvant prendre que 15 valeurs différentes, on ne pourra définir des instants de commutation qu'avec une résolution de $\frac{50\mu s}{2\times(2^4-1)} = 1.666\mu s$

Dans le cas d'un convertisseur multicellulaire, il est nécessaire de déphaser les ordres de commande des interrupteurs des 3 cellules de $\phi = \frac{2.\pi}{3}$. Ce déphasage est réalisé en décalant les trois porteuses (figure II.4.21) avec une précision qui dépend encore du format de codage du rapport cyclique.

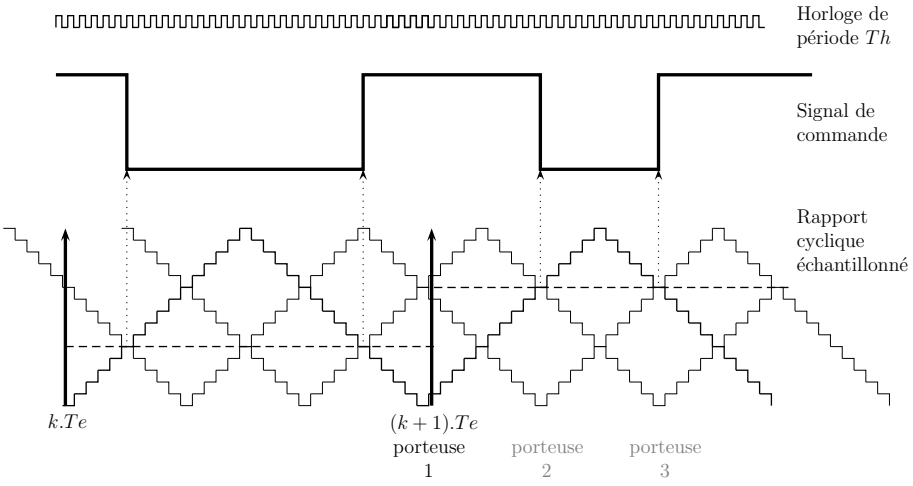


Figure II.4.21: *Principe de la mli numérique sur 4 bits.*

Nous avons cherché à étudier l'influence de la résolution de la MLI sur le contrôle du convertisseur. Cette étude est associée à la recherche d'un gain de surface utilisée sur le CLP. En effet, diminuer la résolution revient à diminuer la taille des compteurs, registres et comparateurs utilisés. Or, si l'on cherche à déporter depuis le microprocesseur une partie de la commande sur le FPGA, il peut être nécessaire d'optimiser les différents modules à implanter, suivant un critère "surface de silicium/précision".

Les simulations ont été effectuées avec un convertisseur multicellulaire série ayant les paramètres suivant :

$$\begin{array}{lll} R & = 10 & \Omega \\ L & = 1.5 & mH \\ Kp & = 5000 & \\ T_e & = 62.5 & \mu s \\ C_1 = C_2 & = 40 & \mu F \\ r_e & = 0.6 & \Omega \\ L_e & = 1 & mH \\ C_e & = 500 & \mu F \\ T_d & = 62.5 & \mu s \\ temps\ morts & \approx 1 & \mu s \end{array} \qquad \text{(II.4.12)}$$

La résolution a été évaluée pour des valeurs allant de 8 à 4 bits. Afin de rester à une période de découpage de $62.5\,\mu s$ (soit une fréquence de $16\,kHz$), l'horloge de base T_h représentée sur la figure II.4.21 doit être adaptée. En effet, la relation suivante entre le période de découpage T_d, le nombre de bits utilisés n et l'horloge T_h qui cadence les compteurs de la MLI est la suivante :

$$T_d = 2 \times (2^n - 1) \times T_h \qquad \text{(II.4.13)}$$

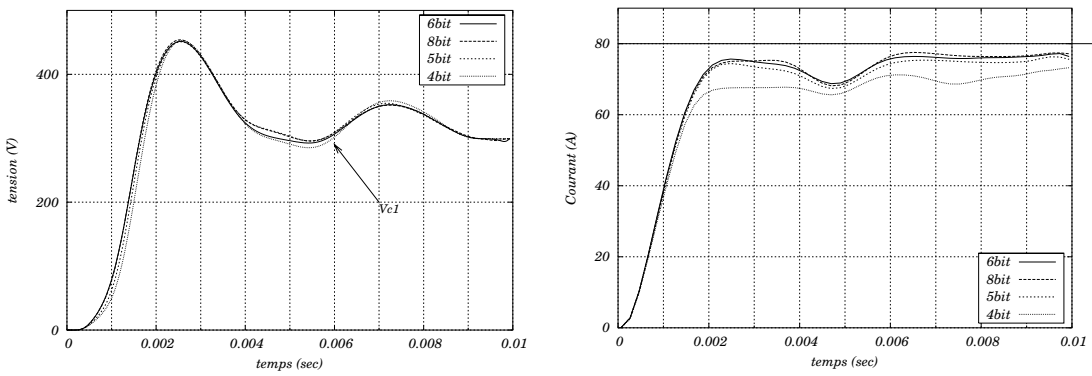
Le tableau II.4.2 récapitule l'évolution de cette horloge mais aussi la valeur du temps mort en fonction du nombre de bits.

résolution	4	5	6	8
horloge T_h	$2.23\mu s$	$1.077\mu s$	$.504\mu s$	$.123\mu s$
temps morts	T_h	T_h	$2.T_h$	$9.T_h$

Tableau II.4.2: *Tableau des différentes horloges.*

La figure II.4.22 montre l'allure des tensions flottantes (V_{c1} et V_{c2}) et la tension d'entrée (E_c) lors d'un démarrage du convertisseur avec un courant de référence de $I_{ref} = 80$ Ampères.

On constate qu'elle suivent relativement bien leurs valeurs de référence correspondant à $E_c/3$ et $2.E_c/3$, à part dans le cas d'une résolution de 4 bits ou l'on observe une dégradation de la tension aux bornes de la capacité C_1. Le courant a un comportement très proche de celui issu d'une commande analogique lorsqu'on utilise un codage sur 8 bits. Il se dégrade toutefois lorsque la résolution diminue et devient inacceptable dans le cas ou l'on utilise seulement 4 bits.



a- Tensions flottantes

b- Courant de charge moyen I_{ch}

Figure II.4.22: *Influence du format de codage de la MLI sur la commande du convertisseur.*

Nous avons effectué une synthèse logique du code VHDL utilisé en co-simulation de niveau 1 afin d'obtenir un ordre de grandeur sur le nombre de portes utilisées par le calcul de la MLI. Cette synthèse a été réalisée pour un composant particulier : un Flex10k d'Altera. Elle donne un ordre d'idée sur l'évolution du nombre de portes lorsque la résolution de la MLI est augmentée. Cependant, les codes VHDL utilisés n'ont pas été validés après synthèse et nécessitent un raffinement (et une optimisation) afin de pouvoir être intégré sur un composant.

La synthèse n'a été effectué que sur le module MLI.

4.4.3.4 Évaluation d'un autre partitionnement des tâches

Comme nous l'avons dit précédemment, l'aspect temporel n'a pas été pris en compte dans cette partie. Étudier des répartitions de tâches différentes présente comme intérêt principal d'évaluer la possibilité d'implanter d'autres fonctions sur le FPGA et d'étudier en particulier l'influence du format de codage.

Ce deuxième partitionnement consiste à intégrer, sur le FPGA, le régulateur proportionnel et le calcul des références de tension (Vc_1ref et Vc_2ref). L'étude porte sur le format de codage de K_p. Le format des grandeurs de référence, quant à lui, est identique à celui des grandeurs mesurées, qui dépend de la résolution des CAN (8 bits dans notre

Résolution de la MLI	Cellules Logiques utilisées
4	117/576 (20%)
6	168/576 (29%)
8	201/576 (34%)
10	237/576 (41%)

Tableau II.4.3: *Nombre de bloc logiques utilisés dans un composant de type Flex10k pour le module* MLI

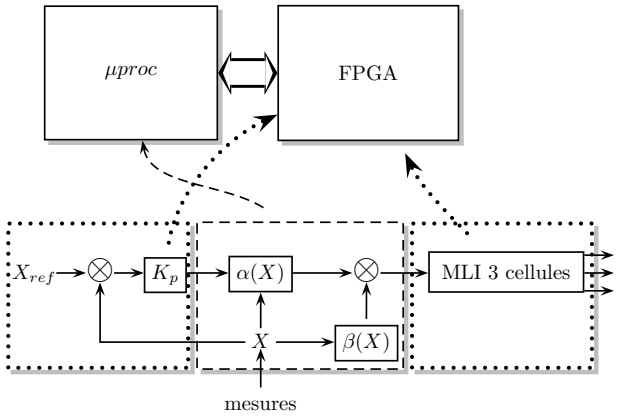
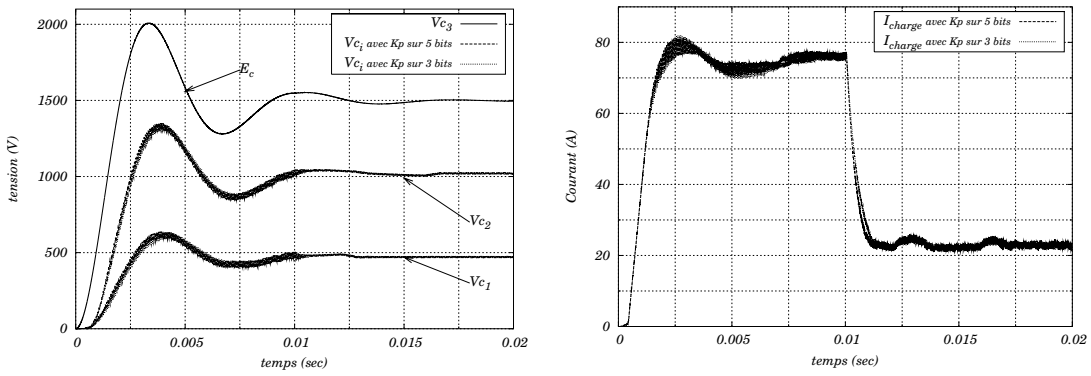


Figure II.4.23: *Deuxième partitionnement testé.*

cas). Les résultats obtenus sont visibles sur la figure II.4.24.

La simulation effectuée correspond à un démarrage du convertisseur avec un courant de référence de 80 Ampères. Un échelon sur le courant de référence est ensuite appliqué pour atteindre 20 Ampère à 20 ms.

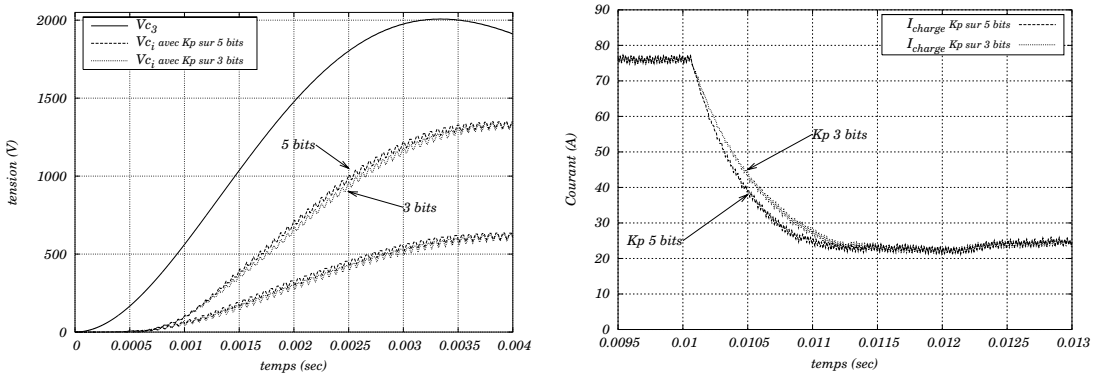


a- Tensions flottantes

b- Courant de charge I_{ch}

Figure II.4.24: *Test de l'influence du codage de K_p sur la commande*

La plate-forme de simulation ainsi mise en place peut aussi être utilisée pour évaluer d'autres éléments tels que la résolution des convertisseurs analogiques numériques. De plus le comportement des différents bus du microprocesseur peut être programmé afin



a- Tensions flottantes.

b- Courant de charge.

Figure II.4.25: *Zoom sur les tensions au démarrage, et le régime transitoire du courant.*

de reproduire celui d'un microprocesseur particulier. Cet élément permet en particulier d'analyser l'influence du format du bus des données.

4.5 Conclusion

Dans ce chapitre nous avons mis en place les éléments nécessaires à un premier environnement de co-simulation dédié aux dispositifs de commande pour les systèmes électriques.

Pour réaliser cet environnement nous avons dû mettre en place, par l'intermédiaire de communications inter-processus, des protocoles de communication entre les différents simulateurs. Actuellement, ces interfaces fonctionnent et ne posent pas de problèmes particuliers de temps de calcul lorsque nous effectuons des simulation du VHDL au premier niveau. Cependant l'utilisation de ces protocoles pour une simulation VHDL de troisième niveau est beaucoup plus coûteuse en temps. Ceci pourrait s'améliorer en utilisant un mode de communication par rendez-vous.

Cet environnement a été développé en utilisant deux simulateurs commerciaux que sont SABER et MODELSIM. Néanmoins, les principes de connexions que nous avons implantés par l'intermédiaire des outils Unix, sont applicables à d'autres simulateurs, si ceux-ci peuvent appeler des procédures C (par exemple MATLAB ou SCILAB). De plus nous nous sommes limités à l'utilisation des communications inter-processus du système Unix. Mais il est possible d'étendre les connexions en utilisant un réseau local ce qui rendrait la mise en place des connexions portable sur d'autres systèmes d'exploitations.

La réalisation d'une interface entre le VHDL et le simulateur SABER a constitué une étape importante dans notre travail. Les résultats que nous avons présentés ont permis de valider en grande partie cet environnement de co-simulation même si l'aspect temporel n'apparaît pas. Cet aspect ne pose pas de problème en soi, puisque les simulations VHDL de niveau 3 tiennent compte des temps de propagation. Il n'en est pas de même pour le microprocesseur puisqu'il faut pouvoir disposer d'un simulateur adaptable. La solution de la simulation compilée reste une bonne alternative à ce problème, mais nécessite un travail important.

Quoiqu'il en soit, nous avons exploité cet outil dans le développement d'observateurs pour la commande du convertisseur multicellulaire que nous présentons dans la troisième partie de ce mémoire.

Troisième partie

Application de la co-simulation dans l'étude d'observateurs dédiés aux convertisseurs multicellulaires

Chapitre 5

Étude d'architectures pour l'implantation d'un pseudo-observateur dédié aux multicellulaires

5.1 Introduction

L'apparition des convertisseurs multicellulaires séries date du début des années 90. Elle se trouve directement directement liée aux besoins grandissants de l'industrie en puissances commutées de plus en plus élevées. En effet, l'amélioration intrinsèque des semi-conducteurs ne permettant pas de satisfaire ces besoins, il a fallu, afin d'augmenter la puissance traitée, développer une nouvelle structure de conversion d'énergie. Son principe repose sur la mise en série de cellules de commutation afin de répartir la tension totale au niveau de chaque cellule et ainsi faire partager les contraintes en tension sur plusieurs composants semi-conducteurs. On reste ainsi à des amplitudes de tension supportables par les interrupteurs de puissance.

Toutefois, la structure multicellulaire nécessite l'utilisation de capacités flottantes dont les tensions aux bornes doivent être maîtrisées et maintenues à des niveaux bien définis pour assurer le bon fonctionnement de l'ensemble du système. Il est possible de réguler le niveau de ces tensions par l'intermédiaire de commande en boucle fermée. Ceci nécessite la mesure des différentes grandeurs à réguler et en particulier, l'utilisation de capteurs de tensions flottantes, qui pose de gros problème de réalisation pour un résultat souvent décevant. Par ailleurs, le nombre de capteurs nécessaire augmente avec le nombre de cellules. Il est donc particulièrement intéressant de pouvoir capter ces grandeurs de manière indirecte, en utilisant un estimateur ou un observateur d'état.

Ce chapitre présente uniquement une structure d'estimation de ces tensions flottantes et étudie différentes méthodes permettant son implantation numérique.

Cette présentation ne peut se faire sans introduire les principes de fonctionnement du convertisseur multicellulaire. C'est ce que nous proposons de faire en première partie en nous basant sur les travaux concernant les propriétés du convertisseur [14, 22], sa commande [26, 70]et quelques méthodes d'observation [8]. Dans un deuxième temps, nous détaillerons le fonctionnement de l'estimateur et nous étudierons différentes variantes permettant de l'implanter sur un composant logique programmable de type FPGA.

5.2 Présentation du convertisseur multicellulaire série à 3 cellules de commutation

Les études ont été menées sur une structure de convertisseur multiniveaux particulière : le convertisseur série à 3 cellules. La présentation de ces convertisseurs est donc volontairement axée sur cette topologie mais le lecteur pourra se reporter aux ouvrages [14, 62] qui présentent de manière plus générale le principe du multicellulaire.

La présentation et la modélisation de ce convertisseur sont effectuées en utilisant certaines hypothèses simplificatrices qui sont :

- les interrupteurs de puissance seront supposés parfaits (chute de tension à l'état passant, courant de fuite et temps de commutation nuls),
- les sources de tension et de courant seront supposées parfaites.

Les convertisseurs multicellulaires sont construits à partir des éléments de base que sont les cellules de commutations. Celles-ci sont interconnectées entre elles par l'intermédiaire de sources de tension flottantes. La figure III.5.1 représente un convertisseur 3 cellules.

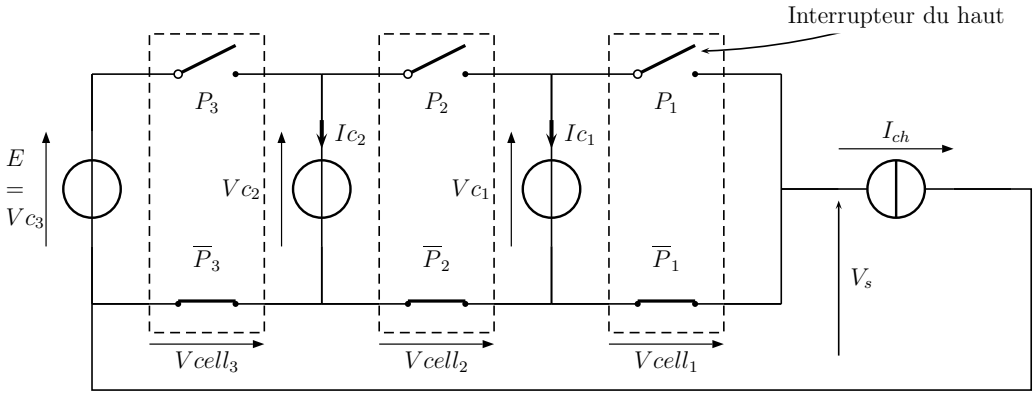


Figure III.5.1: *Convertisseur à 3 cellules de commutation*

Les ordres de commande de chacun des deux interrupteurs dans la cellule de commutation i sont notés P_i et $\bar{P}_i$ (i variant de 1 à 3). Un ordre de commande ne peut prendre que deux valeurs : il vaut 1 pour fermer l'interrupteur et 0 pour l'ouvrir. D'après les règles de commande associées à une cellule de commutation, P_i et $\overline{P}_i$ doivent toujours être complémentaires.

De plus, la commande d'une cellule de commutation se fait indépendamment de l'état des interrupteurs de la cellule voisine. On peut ainsi déphaser la commande des cellules les unes par rapport aux autres sans que cela ne soit dangereux pour la survie des composants.

Étant donné ces principes de commande, il existe, à tout instant, trois interrupteurs à l'état passant et trois interrupteurs à l'état bloqué. Afin de répartir la tension d'entrée E sur les trois interrupteurs de puissance à l'état bloqué, les tensions flottantes Vc_1 et Vc_2 doivent être respectivement égales à $\frac{1}{3}E$ et $\frac{2}{3}E$. Ainsi, chacune des trois cellules voit une tension égale à $\frac{1}{3}E$, et dans ces conditions, le convertisseur est dit équilibré.

Le courant circulant dans une source de tension flottante peut s'exprimer en fonction des ordres de commande et du courant de charge suivant l'équation III.5.1

$$Ic_i = I_{ch}.(P_{i+1} - P_i) \tag{III.5.1}$$

La tension aux bornes de l'interrupteur du bas d'une cellule de commutation peut s'écrire en fonction des ordres de commande et de la valeur des tensions flottantes et de la tension d'entrée $Vc_3 = E$ $(Vc_0 = 0)$

$$Vcell_i = P_i.(Vc_i - Vc_{i-1}) \tag{III.5.2}$$

La tension en sortie du convertisseur V_s correspond à la somme de ces tensions.

$$V_s = \sum Vcell_i \tag{III.5.3}$$

Il a été montré que lorsque le convertisseur est commandé avec des rapports cycliques égaux sur toutes les cellules de commutation et déphasés de $\frac{2\pi}{3}$, les sources de tension flottantes pouvaient être remplacées par des condensateurs. Nous considérerons à présent les sources de tensions flottantes comme des capacités dont le potentiel est flottant.

Le convertisseur qui a été étudié est présenté sur la figure III.5.2. Nous considérerons la tension aux bornes de la capacité Vc_3 comme connue et comme étant la source d'alimentation du convertisseur. La charge est constituée d'une résistance et d'une inductance.

Dans ces conditions, l'état du convertisseur est totalement déterminé par les ordres de commande et les variables d'état $Vc_1, Vc_2,$ *et* I_{ch}.

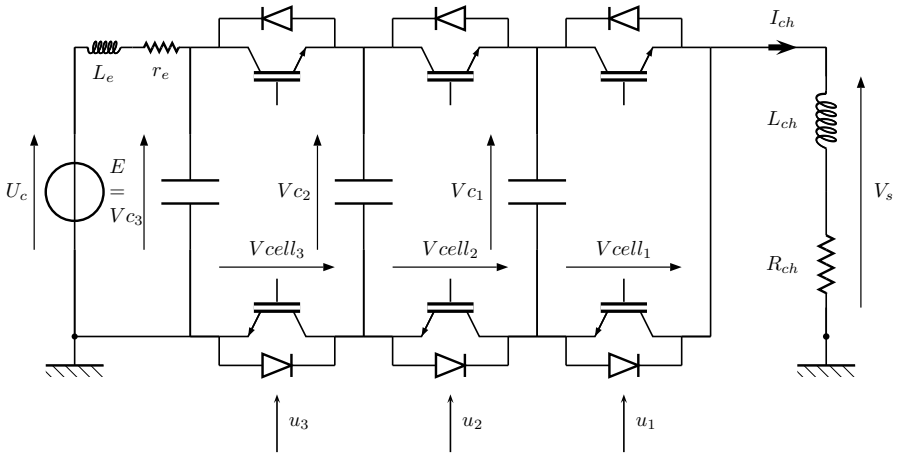


Figure III.5.2: *Convertisseur série 3 cellules en mode hacheur sur une charge RL*

À partir des équations présentées précédemment sur les cellules de commutation, on peut calculer l'évolution des différentes variables d'états suivant le système d'équations III.5.4.

$$\begin{cases} \dfrac{dVc_1}{dt} = \dfrac{1}{C_1}.I_{ch}.(P_2 - P_1) \\ \dfrac{dVc_2}{dt} = \dfrac{1}{C_2}.I_{ch}.(P_3 - P_2) \\ \dfrac{dI_{ch}}{dt} = \dfrac{1}{L}\cdot(V_s - R.I_{ch}) = \dfrac{1}{L}(P_1.(Vc_1) + P_2.(Vc_2 - Vc_1) + P_3.(E - Vc_2) - R.I_{ch}) \end{cases} \tag{III.5.4}$$

Ce système d'équation représente le modèle instantané du convertisseur et peut se représenter sous la forme d'une équation d'état :

$$\dot{X} = A(P).X + B(P).E \tag{III.5.5}$$

où $X = \begin{bmatrix} Vc_1 & Vc_2 & I_{ch} \end{bmatrix}$. Ce système d'état est un système non-linéaire car les matrices A et B dépendent des ordres de commande $P = \begin{bmatrix} P_1 & P_2 & P_3 \end{bmatrix}$.

On en déduit le modèle moyen en remplaçant les ordres de commande instantanés par leurs valeurs moyennes sur une période. On considérera que cette période est constante et identique pour les 3 cellules. Elle est appelée période de découpage : T_d.

$$u_i = \frac{1}{T_d}\int_0^{Td} P_i.dt \tag{III.5.6}$$

Il a été montré qu'un tel convertisseur fonctionnait de manière optimum lorsque les ordres de commande P_1, P_2, P_3 avaient la même valeur moyenne sur une période de découpage (rapports cycliques identiques) et étaient déphasés de $2.\pi/p$ (p étant le nombre de cellules constituant le convertisseur).

Dans ces conditions :

- les condensateurs restent chargés à leurs valeurs optimales qui sont $\frac{1}{3}E$ et $\frac{2}{3}E$,
- la tension de sortie présente une fréquence de découpage apparente f_a égale à trois fois la fréquence de découpage de chaque cellule $f_a = 3.f_{dec}$,
- on peut obtenir 4 niveaux de tension différents en sortie (V_s) selon la valeur du rapport cyclique :

$$\begin{cases} u_i \in [0, \frac{1}{3}] & \Rightarrow V_s \in \{0, \frac{E}{3}\}, \\ u_i \in [\frac{1}{3}, \frac{2}{3}] & \Rightarrow V_s \in \{\frac{E}{3}, \frac{2.E}{3}\} \\ u_i \in [\frac{2}{3}, 1] & \Rightarrow V_s \in \{\frac{2.E}{3}, E\} \end{cases}$$

5.3 Commande des convertisseurs multicellulaires

5.3.1 Méthodes et stratégies utilisées

Le fonctionnement du convertisseur est optimum lorsque celui-ci est équilibré, c'est-à-dire que les tensions flottantes valent respectivement $Vc_1 = \frac{1}{3}E$ et $Vc_2 = \frac{2}{3}E$. Cependant, si la tension d'entrée E varie brusquement, les tensions aux bornes des capacités flottantes ne seront plus égales à $\frac{1}{3}E$ et $^2/_3E$ immédiatement après la variation de E. On aura alors l'apparition de surtensions aux bornes de certains interrupteurs, ce qui peut mettre en danger le fonctionnement du convertisseur.

D'autre part, l'équilibre des tensions n'est respecté que si la valeur des rapports cycliques aux bornes de toutes les cellules est rigoureusement identique. La réalisation pratique des ordres de commande (quantification, temps mort) peut amener des imperfections et donc de légères différences entre ces rapports cycliques, ce qui peut générer des déséquilibres aux bornes des capacités flottantes.

D'après l'équation III.5.1, la valeur des ordres de commandes influe sur le signe du courant qui traverse une capacité. En agissant sur ce signe et sur le temps, on peut alors contrôler la tension aux bornes de la capacité. Dans le cas d'un hacheur classique à deux cellules, la commande ne fait que contrôler le courant de charge en agissant sur les ordres de commande des interrupteurs. Dans le cas du multicellulaire, la commande du système aura un double objectif : le premier consistera à réguler les niveaux de tension aux bornes des capacités flottantes et le second sera de contrôler le courant de sortie I_{ch}.

Une commande de ce type a d'ailleurs été présentée dans le chapitre précédent. Il s'agissait d'une commande par découplage entrées/sorties basée sur une linéarisation

exacte. Elle est calculée à partir du modèle moyen et fait partie des commandes dites "en durée".

Dans une commande en durée, la conversion d'énergie s'effectue par modulation du temps de conduction des interrupteurs i.e. par action sur la valeur moyenne des commandes c'est-à-dire des rapports cycliques.

D'autres techniques de commande existent comme la commande découplante linéaire. Dans ce cas, on linéarise le système autour d'un point de fonctionnement avant d'appliquer un découplage puis une régulation classique de type PI (régulateurs Proportionnel Intégral) [70, 26].

Des commandes en amplitude ont aussi été étudiées[60]. La commande en amplitude consiste à déterminer de manière instantanée l'état des cellules de commutation en fonction de la mesure des variables d'états du système. L'application la plus classique d'une telle commande correspond à la fourchette de courant. Dans le cas des convertisseurs multicellulaires, ce type de commande a montré des performances supérieures aux commandes précédemment citées et une bonne robustesse par rapport aux variations paramétriques de la charge.

Cependant toutes ces commandes ne peuvent être exploitées qu'en mesurant la tension d'entrée du convertisseur, la tension aux bornes des capacités flottantes et le courant de charge.

Les capteurs de tensions flottantes et leurs chaînes de traitement sont délicats à mettre en oeuvre sur les systèmes haute tensions. En effet, on doit d'abord mesurer la différence de potentiel aux bornes des condensateurs flottants à l'aide d'une sonde différentielle. La sortie de cet étage est ensuite mise en forme puis numérisée à l'aide d'un convertisseur analogique/numérique.

Le coût important de l'ensemble de la chaîne de conversion est, de plus, multiplié par le nombre de cellules utilisées dans le convertisseur. Cette chaîne pose donc des problèmes de coût mais aussi d'encombrement et de fiabilité. De plus, vu de la commande, l'ajout des différents éléments nécessaire à la conversion introduit des constantes de temps et des non-linéarités qui peuvent s'avérer perturbantes pour le système.

Aussi, il apparaît intéressant de pouvoir reconstituer la valeur des différentes tensions flottantes avec des capteurs plus classiques, et si possible, en nombre réduit.

5.3.2 Observation du convertisseur multicellulaire série

Nous utiliserons la notion d'estimateur et d'observateur qui a été introduite en section 1.2.2.2 page 8. L'observation du système consiste à reproduire en temps réel une image des grandeurs d'état du système à partir des ordres de commande qui lui sont appliqués et des mesures qu'il est possible de réaliser.

Lorsque l'on cherche à reproduire les grandeurs d'état du système en utilisant un modèle n'ayant comme entrées que les ordres de commande qui sont appliqués au système réel, on réalise un estimateur. Dans le cas d'un estimateur, la notion de boucle fermée et en particulier de dynamique d'observation disparaît.

Cependant, l'utilisation d'un estimateur n'est pas suffisante dans plusieurs cas :
- lorsque l'on ne connaît pas la valeur initiale des variables d'état du système,
- lorsque le modèle n'est pas suffisamment précis et que des erreurs peuvent s'introduire dans le temps,
- lorsque le système réel subit des perturbations qui ne sont pas issues de l'organe de commande et qui sont donc invisibles pour l'estimateur.

L'utilisation d'un observateur permet, en théorie, de pallier à ces limitations.

L'observateur est souvent composé de deux modules :
- le premier module permet "d'estimer" les grandeurs d'états en fonction des grandeurs de commande et utilise un modèle du système.

– le deuxième module est nommé sur le schéma III.5.3 "gain d'observation". Ce module mesure l'erreur entre les grandeurs réelles du systèmes et celles reproduites par le modèle. Cette erreur est alors utilisée par le gain d'observation pour injecter dans le premier module l'information nécessaire à la convergence du modèle vers le système réel.

Dans le cas du multicellulaire série, la grandeur de mesure utilisée correspond au courant de charge.

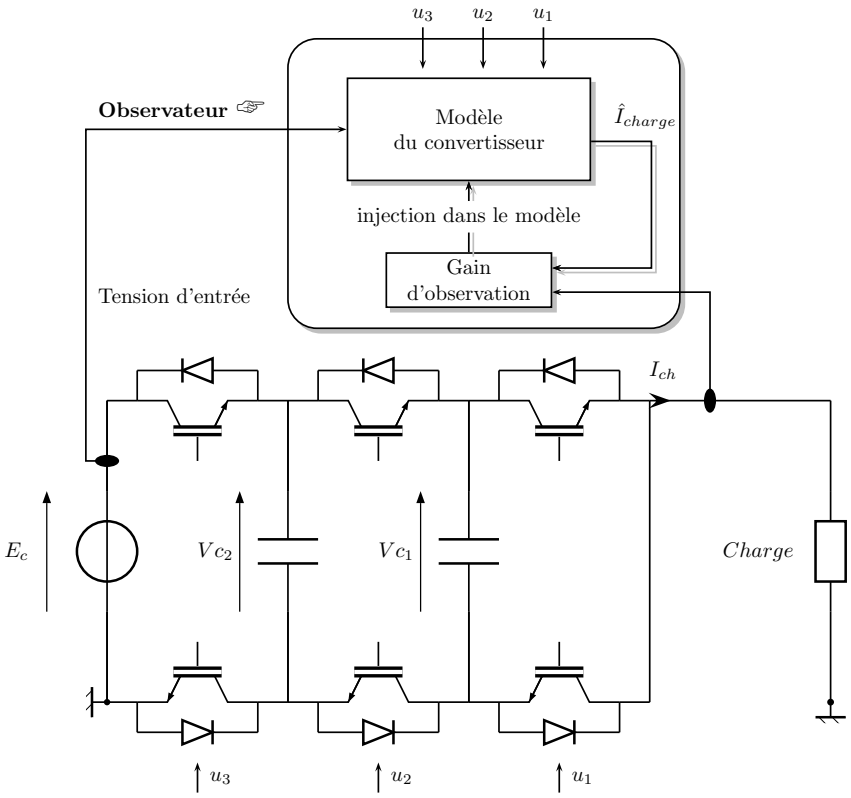


Figure III.5.3: *Schéma de principe des observateurs dans le cas du convertisseur multicellulaire*

Des travaux sur l'observation des tensions aux bornes des capacités flottantes à partir du courant de charge et de la tension d'entrée ont été menés dans [8]. Nous résumons ici brièvement les principaux résultats qui en ressortent.

La différence provient surtout du modèle du convertisseur utilisé à des fins d'observation.

Cas du modèle moyen

Nous avons vu au chapitre précédent que le modèle moyen du convertisseur multicellulaire permet d'avoir un modèle relativement simple qui permet d'élaborer des lois de commandes découplantes pour la régulation des tensions aux bornes des capacités flottantes et du courant de charge. Cependant, en régime permanent (i.e. lorsque les rapports cycliques sont égaux), il n'est pas possible d'observer les tensions flottantes à partir du courant de charge. En effet les équations du système III.5.4 nous indiquent que lorsque $u_1 = u_2 = u_3$, le courant de sortie est le même quelle que soit la valeur des tensions flottantes.

Cas des modèles échantillonnés

Deux modèles échantillonnés ont été utilisés. Le premier correspond au modèle échantillonné exacte sur une période de découpage. Il consiste à intégrer les variables d'état du modèle instantané sur une période de découpage en fonction des ordres de commande envoyés durant cette période.

Ce modèle permet de calculer l'état du système à la fin de la période de découpage, et ce, en fonction de l'état du système en début de période.

À partir de ce modèle, un observateur de Luenberger a été développé. L'auteur conclut sur le fait que cet observateur est peu robuste par rapport aux variations paramétriques et aux bruits de mesure. De plus il comporte un volume de calcul très important, ce qui rend son implantation difficilement envisageable.

L'autre modèle échantillonné correspond à l'intégration d'un modèle moyen calculé sur un tiers de la période de découpage. À partir de ce modèle, un filtre de Kalman récursif a été évalué. Il en résulte un observateur relativement robuste par rapport aux variations paramétriques lorsque celui-ci est correctement réglé. Ce réglage est cependant difficile à réaliser et l'utilisation d'un tel observateur nécessite la mise en place d'un dispositif de commande performant afin de réaliser l'ensemble des calculs en un temps inférieur au tiers de la période de découpage.

Cas du modèle instantané

En utilisant le modèle instantané, il est possible d'implanter un observateur à mode glissant qui s'avère avoir de bonnes performances et une certaine robustesse face aux variations paramétriques. Cependant, comme précédemment, cet observateur présente un volume de calculs important.

Les solutions que nous venons de présenter ont en commun deux inconvénients majeurs :

- elles nécessitent un volume de calculs important,
- elles sont sensibles aux variations de la charge.

Nous avons donc choisi une dernière méthode consistant à "reconstruire" l'état du convertisseur à chaque instant [26].

5.4 Reconstitution des grandeurs d'états à l'aide des tensions

La méthode utilisée est basée sur le modèle instantané du convertisseur. Pour s'affranchir de la charge, les tensions flottantes sont reconstituées à partir de la tension de sortie V_s.

La figure III.5.4 présente le schéma de principe d'une commande utilisant ce reconstructeur d'état. Les entrées de ce reconstructeur d'état sont la tension de sortie du convertisseur V_s, les ordres de commande associés à chaque cellule de commutation P_1, P_2, P_3. La commande quant à elle nécessite toujours la mesure du courant de charge.

À tout instant, nous supposons que l'état des interrupteurs est imposé par les ordres de commande associés (interrupteurs parfaits). Le raisonnement utilise un modèle instantané du convertisseur. Soit $P^* = [P_1 \quad P_2 \quad P_3]$ l'état des cellules à chaque instant. Les deux états que peut prendre chaque cellule sont '1' et '0', et correspondent respectivement à la conduction ou au blocage de l'interrupteur du "haut". Le vecteur P^* connaît alors 2^3 valeurs possibles. On notera que la tension Vs est une composition

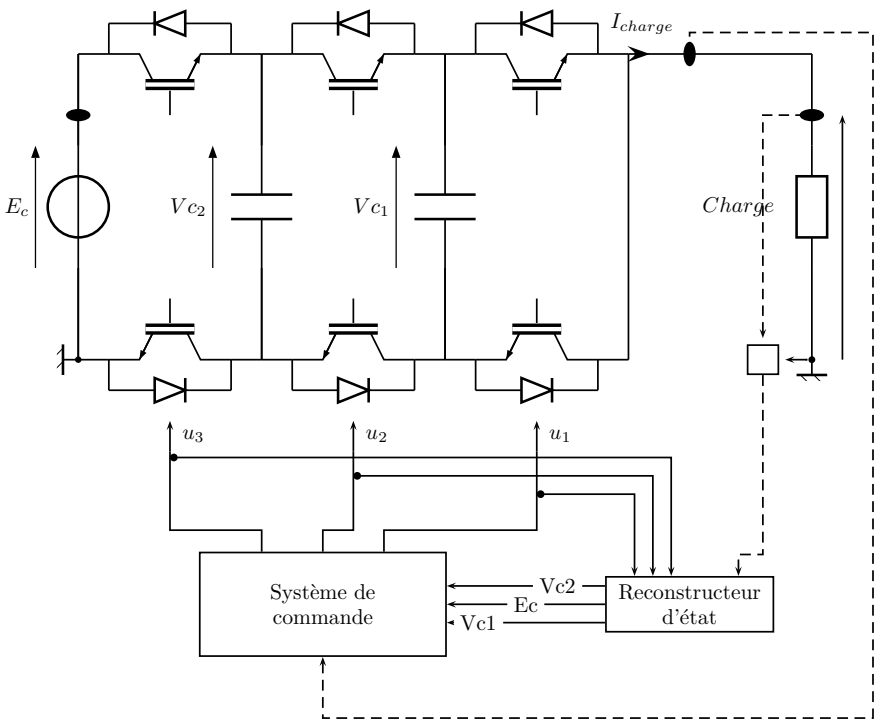


Figure III.5.4: *Commande avec reconstructeur d'état*

valeur	P_1	P_2	P_3	Vs	à l'équilibre
0	0	0	0	0	0
1	0	0	1	Vc_1	$\frac{E}{3}$
2	0	1	0	$Vc_2 - Vc_1$	$\frac{E}{3}$
3	0	1	1	Vc_2	$\frac{2.E}{3}$
4	1	0	0	$Ec - Vc_2$	$\frac{E}{3}$
5	1	0	1	$Ec - Vc_2 + Vc_1$	$\frac{2.E}{3}$
6	1	1	0	$Ec - Vc_1$	$\frac{2.E}{3}$
7	1	1	1	Ec	$\frac{E}{1}$

Tableau III.5.1: *Tension en sortie du convertisseur en fonction de la commande des interrupteurs*

des tensions E_c, Vc_1, Vc_2 qui dépend du vecteur P^*. Le tableau III.5.1 récapitule les différentes valeurs possibles du vecteur P^* ainsi que les valeurs de Vs correspondantes.

On remarquera que pour les états 1,3,7, la mesure de Vs nous donne directement la mesure de l'une des tensions recherchées (ce cas sera nommé *mesure directe*). Cependant, rien ne garantit que ces combinaisons vont apparaître de manière régulière et il est nécessaire de prendre en compte tous les états.

Par la suite on considérera que la valeur moyenne des tensions varient peu par rapport à la fréquence de découpage.

La mesure de V_s pouvant se faire de manière échantillonnée, on a pour le i^{eme}

échantillon de V_s la relation suivante :

$$V_{s_i} = \begin{bmatrix} P_{1_i} - P_{2_i} & P_{2_i} - P_{3_i} & P_{3_i} \end{bmatrix} \begin{bmatrix} Vc_1 \\ Vc_2 \\ E_c \end{bmatrix} \tag{III.5.7}$$

Ayant trois grandeurs à reconstituer, trois mesures successives de la tension de sortie dans un intervalle de temps de l'ordre de la période de découpage, nous donne alors la relation matricielle suivante :

$$\begin{bmatrix} V_{s_3} \\ V_{s_2} \\ V_{s_1} \end{bmatrix} = \begin{bmatrix} P_{m_3} \\ P_{m_2} \\ P_{m_1} \end{bmatrix} . \begin{bmatrix} Vc_1 \\ Vc_2 \\ E_c \end{bmatrix} = \begin{bmatrix} P_{1_3} - P_{2_3} & P_{2_3} - P_{3_3} & P_{3_3} \\ P_{1_2} - P_{2_2} & P_{2_2} - P_{3_2} & P_{3_2} \\ P_{1_1} - P_{2_1} & P_{2_1} - P_{3_1} & P_{3_1} \end{bmatrix} \begin{bmatrix} Vc_1 \\ Vc_2 \\ E \end{bmatrix} \tag{III.5.8}$$

Si la matrice est inversible on peut reconstituer les trois grandeurs recherchées.

$$\begin{bmatrix} Vc_1 \\ Vc_2 \\ Ec \end{bmatrix} = \begin{bmatrix} P_m^{*\,-1} \end{bmatrix} . \begin{bmatrix} V_{s1} \\ V_{s2} \\ V_{s3} \end{bmatrix} \tag{III.5.9}$$

5.4.1 Contraintes sur la mesure

Comme nous l'avons vu précédemment, il se peut que l'acquisition de trois grandeurs successives ne nous permette pas d'accéder à une matrice inversible. Il faut alors déterminer une stratégie qui permette de retenir 3 mesures rendant la matrice inversible. Pour cela, nous avons utilisé la méthode présentée sur la figure III.5.5.

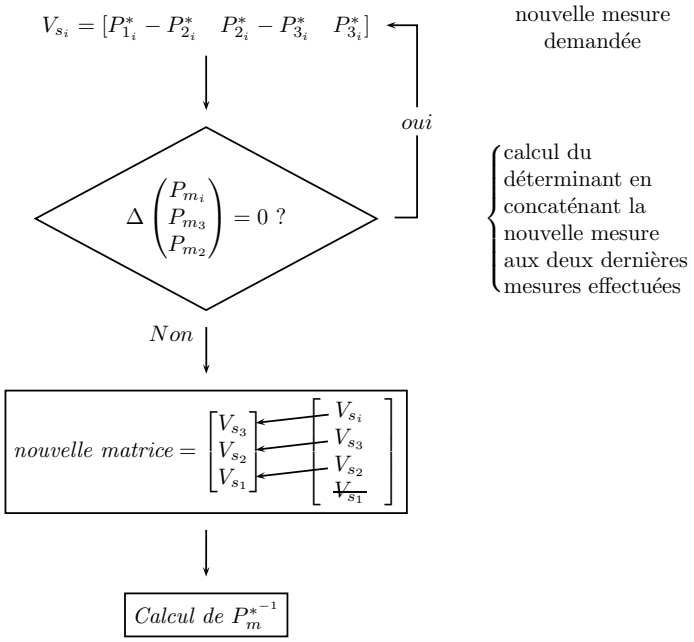


Figure III.5.5: *Choix des vecteurs composant la matrice à inverser*

A l'état initial on effectue trois mesures différentes de V_s puis on laisse l'algorithme se dérouler.

Ce reconstructeur d'état a été étudié en simulation dans [26] en considérant une mesure de V_s à cadence fixe. Les résultats de simulation ont permis de valider la méthode pour différents rapports cycliques.

5.5 Implantation numérique du reconstructeur d'états

Bien que validée en simulation, cette méthode nécessite quelques modifications pour tenir compte :

- des imperfections des interrupteurs de puissance,
- des contraintes temps réel ($T_d \approx qq\mu s$),
- des calculs à effectuer.

Nous présentons ici les différentes solutions qui ont permis de prendre ne compte ces problèmes.

5.5.1 Échantillonnage de la tension de sortie V_s

L'étude précédente est basée sur une mesure régulière de V_s dans le temps. Cependant, si la période d'acquisition est très faible, le système va mesurer plusieurs valeurs de V_s pour des ordres de commande identiques. Or deux mesures identiques ne permettent pas d'inverser la matrice P_m^*. Un certain nombre de mesures sont donc inutiles. Par contre, si la période d'acquisition est trop grande, on peut "louper" une mesure de Vs qui aurait permis d'inverser la matrice P_m^*.

En fait, l'acquisition d'une mesure de V_s n'apparaît intéressante que si celle-ci correspond à une nouvelle composition des tensions recherchées. Ainsi cet estimateur n'a pas de contraintes de régularité au niveau de la mesure. Nous n'allons donc pas utiliser un échantillonnage régulier dans le temps. Par contre, une nouvelle mesure de tension sera déclenchée après un changement d'état du vecteur P^* .

Cependant, il faut prendre en compte plusieurs phénomènes dus aux imperfections du convertisseur et des mesures. En effet, les interrupteurs de puissance mettent un certain temps avant de conduire ou de se bloquer. Le capteur de tension n'a pas une bande passante infinie ce qui introduit un retard dans la mesure.

Toutes ces imperfections nous obligent à imposer un retard entre le changement des ordres de commande P^* et le déclenchement d'une nouvelle mesure. Sans ce retard, la mesure de V_s risquerait d'être incohérente avec l'état des interrupteurs supposé représenté par le vecteur P^* . On notera aussi que faire une mesure à des instants réguliers sans prendre en compte le changement d'état du vecteur P^* pourrait amener aux mêmes incohérences. La figure III.5.6 présente un exemple du fonctionnement de demande d'acquisition retardé d'un temps égal à $6.5\mu s$ à priori.

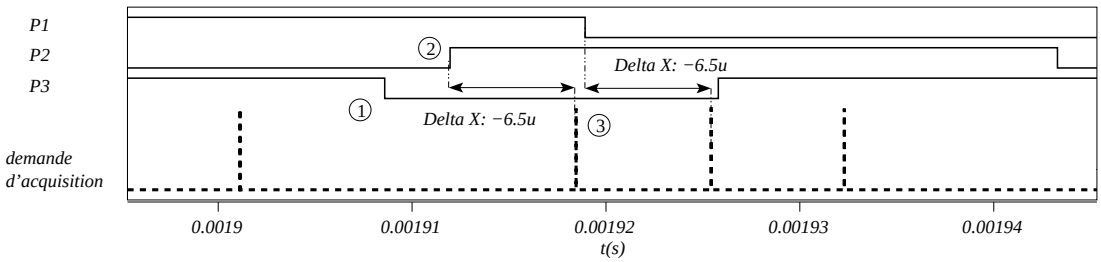


Figure III.5.6: *Exemple d'acquisitions retardées de 6.5µs par rapport aux ordres de commande.*

Sur cette figure on peut voir l'évolution des trois ordres de commande P_1, P_2, P_3, et le signal de demande d'acquisition d'une nouvelle mesure. On peut constater en ① un changement de P_3. Cependant il n'y a pas de demande d'acquisition associée. En effet, comme un changement dans les ordres de commande s'est effectué avant le délai de $6.5\,\mu s$ (changement de P_2 ②), la demande d'acquisition qui était associée au changement de P_3 a été annulée.

5.5.2 Réalisation des calculs à l'aide de matrices pré-calculées

Nous venons de voir sur l'algorithme de la figure III.5.5 qu'il est nécessaire d'effectuer des calculs après chaque mesure de V_s et ce sur un intervalle de temps proche de la période de découpage.

Cependant, cet estimateur doit effectuer des mesures de V_s parfois très rapprochées, et dés qu'une mesure a été acquise, il faut inhiber les demandes d'acquisition durant les calculs nécessaires à l'estimation.

Afin d'inhiber un minimum de demandes d'acquisition, il faut effectuer les calculs au plus vite.

Le calcul d'un déterminant ou une inversion de matrice correspond à une phase calculatoire assez lourde. Il serait donc intéressant de trouver une méthode permettant d'implanter ce reconstructeur d'état dans un composant logique programmable. L'utilisation d'un tel composant permettrait de diminuer le temps de calcul et donc de diminuer l'intervalle minimal entre deux mesures de V_s.

Nous avons alors testé deux méthodes d'implantation possibles. Nous présentons tout d'abord une méthode d'inversion de matrice pré-calculée.

5.5.2.1 Principe

Dans l'optique de ne pas effectuer de calculs trop lourds, nous avons choisi d'utiliser une table de vérité nous indiquant si une matrice est inversible ou non. Cette table de vérité (III.5.2) a comme entrées les trois vecteurs $P*$ associés aux trois mesures de V_s (cf tableau III.5.1). La sortie de cette table vaut '1' si le déterminant est non nul, sinon '0'. Chaque ordre de commande P_i vaut 0 ou 1, Il s'agit alors d'une table comprenant $(2^3)^3 = 512$ entrées.

indice du tableau									déterminant
P_{1_1}	P_{2_1}	P_{3_1}	P_{1_2}	P_{2_2}	P_{3_2}	P_{1_3}	P_{2_3}	P_{3_3}	Δ

Tableau III.5.2: *entête du tableau contenant les valeurs du déterminant*

Pour chaque valeur en entrée où le déterminant n'est pas nul, une inversion de matrice est à effectuer. Là encore, nous pouvons pré-calculer les solutions en mettant la valeur des coefficients λ_i du système résolu (équation III.5.10) dans un tableau. L'entrée de ce tableau correspond alors à l'entrée de la table de vérité concaténée avec l'indice i allant de 1 à 9.

$$\begin{bmatrix} Vc_1 \\ Vc_2 \\ E_c \end{bmatrix} = \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 \\ \lambda_4 & \lambda_5 & \lambda_6 \\ \lambda_7 & \lambda_8 & \lambda_9 \end{bmatrix} \begin{bmatrix} V_{s1} \\ V_{s2} \\ V_{s3} \end{bmatrix} \tag{III.5.10}$$

Dans le cas d'un convertisseur comportant 3 cellules, les coefficients appartiennent à l'ensemble : $\lambda_i \in \{\pm 2, \pm 1, \pm 0.5, 0\}$, ce qui numériquement peut s'implanter facilement. En effet dans le cas où les différentes grandeurs sont codées en entier, la multiplication par 2 correspond à un décalage à gauche et la multiplication par 0.5 correspond à un décalage à droite.

5.5.2.2 Influence du délai d'acquisition

Nous avons effectué des simulations en évaluant l'influence du retard entre la demande d'acquisition de la tension V_s et un changement d'état d'une cellule.

Deux types de simulations ont été effectuées. La première que l'on nomme "boucle ouverte", correspondant au cas où le convertisseur est commandé à partir des grandeurs

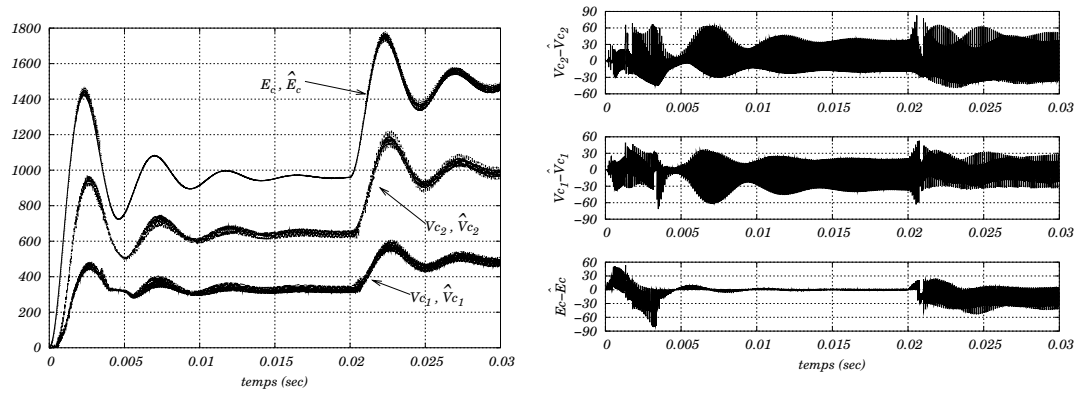
réelles des tensions Vc_1, Vc_2 et Ec. Dans le cas de la "boucle fermée", ces tensions proviennent du reconstructeur d'état.

L'ensemble des simulations est effectué avec les paramètres suivants :

$$\begin{array}{lll} R_{ch} & = 10 & \Omega \\ L_{ch} & = 1.5 & mH \\ Kp & = 5000 & \\ T_e & = 62.5 & \mu s \\ C_1 = C_2 & = 40 & \mu F \\ r_e & = 0.6 & \Omega \\ L_e & = 1 & mH \\ C_e & = 500 & \mu F \end{array} \qquad \text{(III.5.11)}$$

Nous avons utilisé le profil suivant : au démarrage du convertisseur, les tensions aux bornes des capacités flottantes sont nulles, et le courant de référence est fixé à 80 ampères. La tension d'entrée subit ensuite un échelon d'amplitude à 0.02 secondes.

Évaluation en boucle ouverte



a- Estimation des tensions flottantes

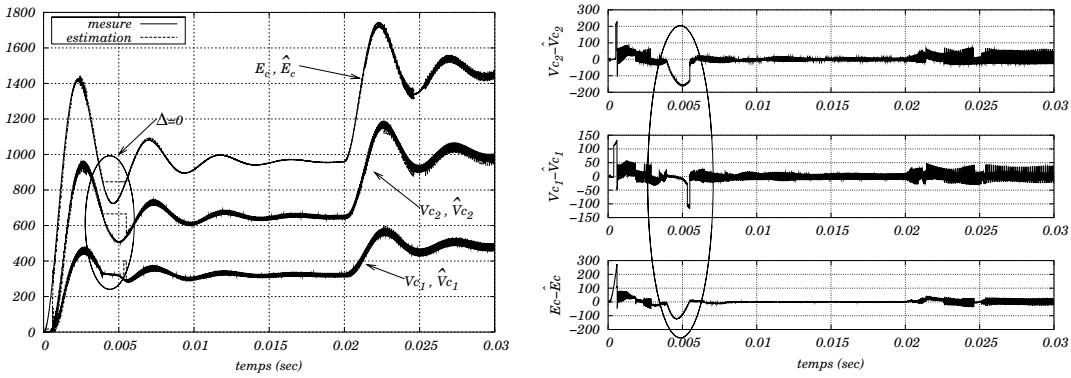
b- Erreur entre les grandeurs estimées et les grandeurs mesurées

Figure III.5.7: *Estimation en boucle ouverte avec un délai de* $1.5\mu s$

La figure III.5.7 correspond à la simulation de l'estimateur non-bouclé avec un retard avant l'acquisition de $1.5\mu s$, ce qui correspond au temps de conversion d'un convertisseur A/N moyen. On constate alors le bon fonctionnement de celui-ci. La figure III.5.7 montre d'ailleurs l'évolution de l'erreur entre chaque grandeur mesurée et estimée. On peut voir que l'erreur dépasse rarement les 10%. Nous avons alors testé le fonctionnement en augmentant le délai d'acquisition jusqu'à $6.5\mu s$ (figure III.5.8). On constate alors que ce délai est trop important puisqu'il existe des configurations où le déterminant reste nul pendant plusieurs périodes d'échantillonnage, notamment à 5 ms, ce qui provoque un blocage de l'évolution des valeurs estimées jusqu'à ce que les rapports cycliques évoluent de manière à obtenir de nouveau une matrice inversible.

Évaluation en boucle fermée

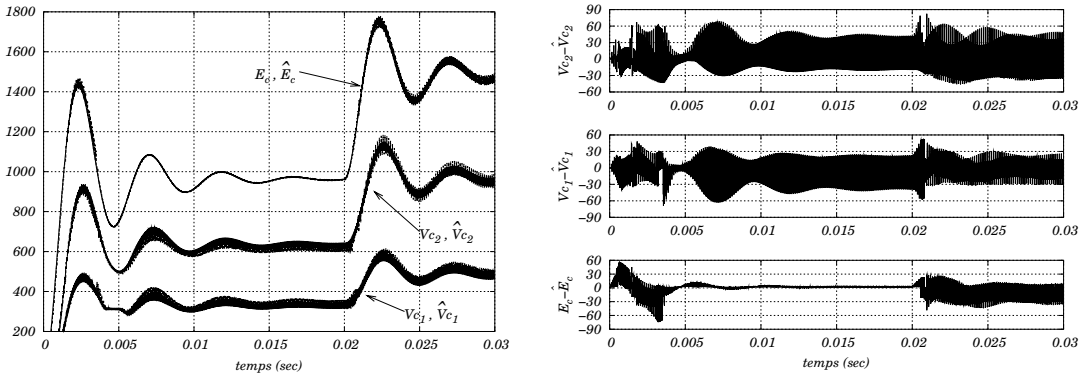
Nous avons testé le comportement de l'estimateur en boucle fermée, i.e. que la commande du convertisseur utilise non plus les grandeurs mesurées mais les grandeurs estimées par le reconstructeur d'état.



a- Estimation des tensions flottantes

b- Erreur entre les grandeurs estimées et mesurées

Figure III.5.8: *Estimation en boucle ouverte avec un délai de* $6.5\mu s$



a- Estimation des tensions flottantes

b- Zoom sur un régime transitoire

Figure III.5.9: *Estimation en boucle fermée avec un délai de* $1.5\mu s$

Dans le cas d'un délai de $1.5\mu s$, figure III.5.9-b-, on obtient des erreurs du même ordre de grandeur qu'en boucle ouverte. Par contre pour un délai de $6.5\mu s$, on constate sur la figure III.5.10 que le courant n'est plus contrôlé. En effet, celui-ci s'écarte de la courbe correspondant au cas ou la commande utilise les mesures des tensions flottantes (cas idéal vis-à-vis de l'estimation). Cette perte de contrôle est en fait du à une mauvaise estimation des tensions flottantes que l'on peut voir en figure III.5.11. En effet, la matrice P_m^* n'étant plus inversible, les tensions estimées restent constantes et égales aux dernières valeurs calculées (ou le déterminant n'était pas nul). Dans ce cas, la commande utilise des valeurs de tensions constantes qui ne correspondent pas aux évolutions réelles et cela provoque une saturation des rapport cycliques qui engendre un courant maximal dans la charge.

On peut constater que le délai d'acquisition a une influence particulière sur le fonctionnement de l'estimateur et qu'il apparaît certains cas ou ce délai ne permet plus d'estimer correctement les tensions.

L'ensemble des fonctions utilisées pour l'estimation pouvant être pré-calculées, il suffit maintenant de mémoriser l'ensemble des résultats. Dans le cas d'un convertisseur 3 cellules, il nous faut mémoriser le test d'inversibilité qui requiert 3 octets par entrée (chaque entrée est codée sur 10 bits) et qui contient 512 entrées. Ensuite, l'ensemble

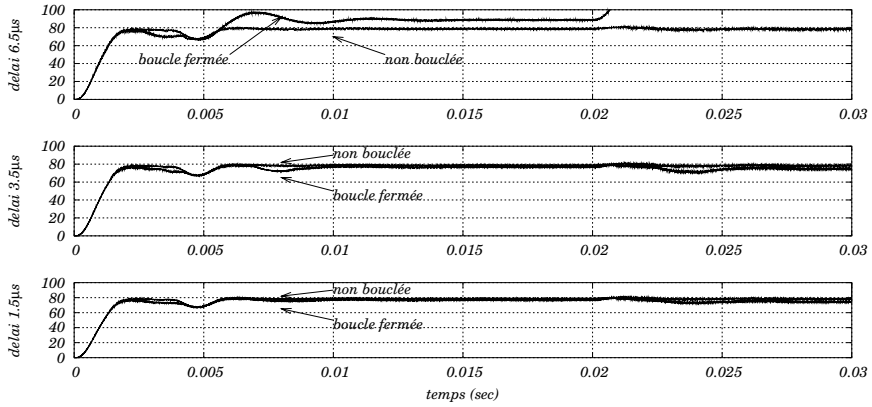
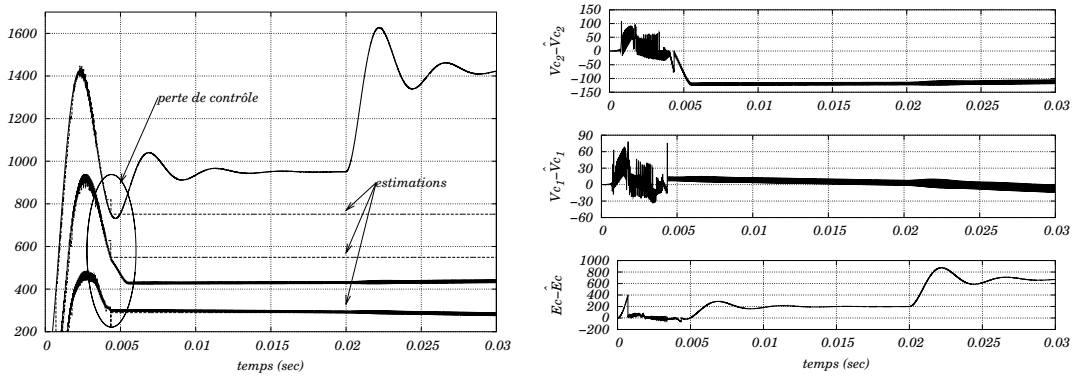


Figure III.5.10: *comparaison des courants régulés en utilisant les grandeurs estimées et les grandeurs mesurées*



a- Estimation des tensions flottantes

b- Zoom sur un régime transitoire

Figure III.5.11: *Estimation en boucle fermée avec un délai de* $6.5\mu s$

des coefficients λ_i doit aussi être mémorisé, chaque coefficient peut être codé sur 3 bits et nous avons 9 coefficients par entrée. Cela demande alors $4\,octets \times 512$. L'ensemble des calculs nécessite alors une mémoire de 3 584octets. Ce calcul montre qu'en utilisant un composant logique programmable de taille moyenne, il est tout à fait possible d'implanter un tel estimateur.

Nous avons cependant étudié un estimateur dérivé du principe originel qui semble plus adapté pour les composant logiques programmables.

5.5.3 Évaluation du reconstructeur d'état calculé à partir d'une machine à états finis

Nous avons essayé d'utiliser une variante possible du reconstructeur vu précédemment, afin de ne pas avoir de déterminant à calculer et d'aboutir à une forme plus adaptée à une implantation en logique câblée.

Dans cette variante, nous n'attendons pas qu'une matrice soit inversible pour faire une nouvelle estimation. Nous ne rejetons, en fait, aucune mesure. À chaque nouvelle acquisition de V_s, nous effectuons l'estimation d'une des trois grandeurs (E, Vc_1, Vc_2). Cette estimation est calculée en résolvant l'équation (III.5.7) page 111 ayant pour inconnue une des trois grandeurs recherchées. Le choix de la nouvelle grandeur à estimer

s'effectue en fonction d'un ordre de priorité. Nous donnons une priorité maximale pour la grandeur estimée la plus ancienne et respectivement la priorité la plus faible pour la dernière grandeur qui a été estimée.

Exemple 5.5.1 Imaginons que la dernière tension qui ait été estimée soit E et la plus ancienne tension estimée soit Vc_1. On a alors un vecteur des tensions estimées écrit dans l'ordre suivant :

tension à estimer en priorité	1	Vc_1
	2	Vc_2
	3	Ec

Si un changement se fait sur les ordres de commande, et que l'on mesure Vs tel que $V_s = Vc_2 - Vc_1$, alors on utilisera la mesure de V_s de la manière suivante :

$$\hat{V}c_1 = V_s + \hat{V}c_2 \tag{III.5.12}$$

On rafraîchit ainsi l'estimation de Vc_1 et on obtient un nouvel ordre de priorité pour les estimations à effectuer :

tension à estimer en priorité	1	Vc_2
	2	Ec
	3	Vc_1

Par contre si la mesure de V_s correspond à la composition, $V_s = Ec - Vc_2$, alors on rafraîchira l'estimation de la tension Vc_2 de la manière suivante :

$$\hat{V}c_2 = \hat{E}c - V_s \tag{III.5.13}$$

Et dans ce cas, le tableau des priorités est le suivant :

tension à estimer en priorité	1	Vc_1
	2	Ec
	3	Vc_2

Le mécanisme de détermination du calcul à effectuer, en fonction de la priorité et de la mesure de Vs, peut se représenter à l'aide du formalisme des machines à états finis. Ce qui en fait un système relativement simple à implanter sur un CLP. Nous détaillons la construction de cette machine à états.

5.5.4 Choix des états

Un état est fixé pour chaque ordre de priorité possible. Le choix des états est détaillé dans le tableau III.5.3.

Connaissant les grandeurs prioritaires à estimer, on détermine, pour une nouvelle mesure de V_s, le calcul qui peut être fait. Cette détermination correspond donc au calcul de la transition.

Les ordres de priorités possibles $\oplus \longrightarrow \ominus$	Etat associé
$E_c \quad Vc_2 \quad Vc_1$	S_0
$Vc_2 \quad E_c \quad Vc_1$	S_1
$E_c \quad Vc_1 \quad Vc_2$	S_2
$Vc_1 \quad E_c \quad Vc_2$	S_3
$Vc_2 \quad Vc_1 \quad E_c$	S_4
$Vc_1 \quad Vc_2 \quad E_c$	S_5

Tableau III.5.3: *Association d'un état pour chaque ordre de priorité*

5.5.5 Calcul des transitions

Nous devons identifier quelles sont les tensions qui composent la tension de sortie V_s. Le vecteur indiquant l'état des cellules P^* nous donne cette information.

En effet, nous avons vu qu'à une mesure de Vs correspond une combinaison des tensions E_c, Vc_1, Vc_2. Cette **combinaison** est calculée à l'aide du vecteur C_1^* (III.5.14).

Si l'on veut déterminer quelles sont, parmi les trois tensions, celles qui apparaissent effectivement dans la **composition** de la tension V_s, il nous faut appliquer la relation (III.5.15) :

$$P^* \longrightarrow C_1^* = \begin{bmatrix} P_{1_i} - P_{2_i} & P_{2_i} - P_{3_i} & P_{3_i} \end{bmatrix} \tag{III.5.14}$$

$$P^* \longrightarrow C_2^* = \begin{bmatrix} P_{1_i} \oplus P_{2_i} & P_{2_i} \oplus P_{3_i} & P_{3_i} \end{bmatrix} \tag{III.5.15}$$

Les coefficients du vecteur C_2^* indiquent par un '1' la présence ou non des tensions respectives E_c, Vc_2, Vc_1 . On notera que les relations (III.5.14) et (III.5.15) sont toutes les deux bijectives par rapport au vecteur P^*. Ainsi pour une seule valeur de ce vecteur, il existe une seule et unique **composition** des tensions, ainsi qu'une seule et unique **combinaison**.

Nous avons choisi d'utiliser le vecteur C_2^* pour représenter la transition . Le tableau III.5.4 référence les différentes transitions possibles.

Transition	$C_2^*(1)$ E_c	$C_2^*(2)$ Vc_2	$C_2^*(3)$ Vc_1	V_s
t_1	0	0	1	Vc_1
t_2	0	1	0	Vc_2
t_3	0	1	1	$Vc_2 - Vc_1$
t_4	1	0	0	E_c
t_5	1	0	1	$E_c - Vc_1$
t_6	1	1	0	$E_c - Vc_2$
t_7	1	1	1	$E_c - Vc_2 + Vc_1$

Tableau III.5.4: *Ensemble des transitions possibles*

5.5.6 Représentation de l'estimateur

En ayant choisi les états et les transitions précédents, nous pouvons alors représenter l'estimateur comme un diagramme de transitions (figure III.5.12 page suivante).

Nous allons implanter ce diagramme de transitions sous la forme d'une machine de Mealy(figure III.5.13 page ci-contre). Ce type de machine à états finis est essentiellement

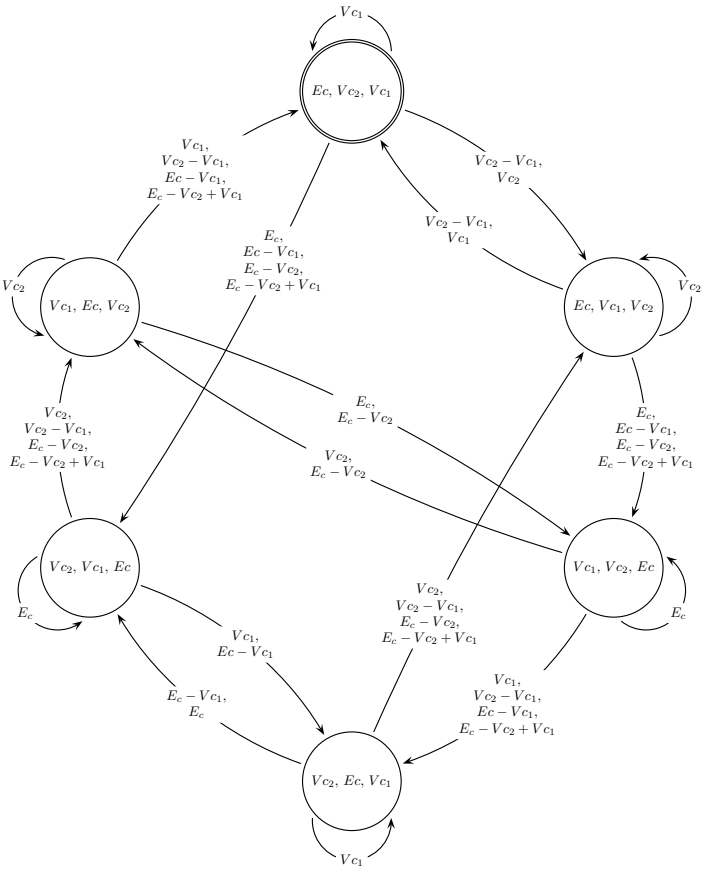


Figure III.5.12: *Représentation de la Machine d'état*

composée de deux *étages*. Le premier correspond au calcul d'un nouvel état en fonction de l'état précédent et de la transition. Le deuxième étage calcule les grandeurs de sortie en fonction de l'état présent et de la valeur de transition.

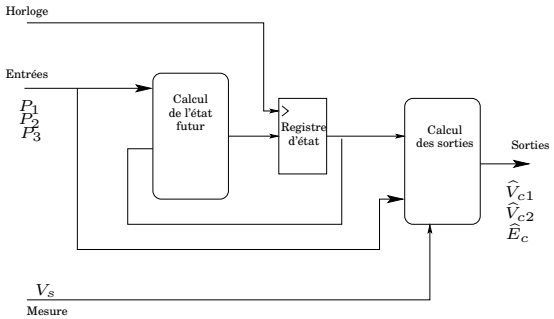


Figure III.5.13: *Estimateur représenté par une machine de mealy*

Cette machine sera synchronisée sur la demande d'acquisition retardée par rapport aux changements sur les ordres de commande. Ce retard est réalisé par un simple "décompteur".

Ce type de représentation est bien adaptée à une implantation numérique.

Nous avons effectué la simulation de cet estimateur, dont on peut voir le résultat en figure III.5.14 page suivante. Cette simulation a été effectuée en utilisant un délai d'attente, avant acquisition, de $3\mu s$. On constate qu'il existe des configurations où

l'estimateur ne donne pas de résultats correctes et ces erreurs ne sont pas négligeables.

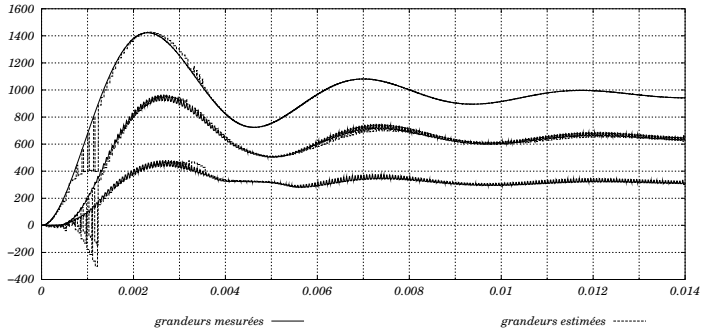


Figure III.5.14: *Utilisation d'une machine à états finis pour l'estimation des trois tensions E, Vc_1, Vc_2 avec un délai avant acquisition de $3\mu s$*

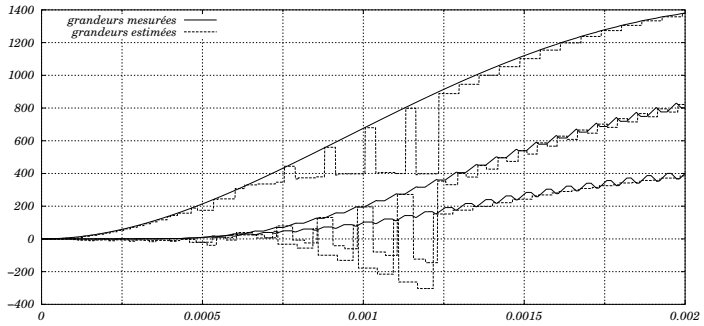


Figure III.5.15: *Utilisation d'une machine à états finis pour l'estimation des trois tensions E, Vc_1, Vc_2 avec un délai avant acquisition de $3\mu s$*

Durant la période comprise entre 0.5 et 1.2 ms (cf. Figure III.5.15), les différentes transitions que voit la machine d'état, sont $t_{3,5,6,7}$. On constate en fait qu'il n'y a, durant cette intervalle, aucune mesure directe d'une des trois tensions E, Vc_1, Vc_2. L'estimation des grandeurs, durant ce laps de temps, est donc basée sur une ancienne mesure directe, ce qui explique l'augmentation de l'erreur au fur et à mesure de l'évolution de la tension E.

Cette simulation montre que l'utilisation d'une telle machine à états n'apparaît pas robuste vis-à-vis des variations de la tension du bus continu.

Utilisation de la mesure de la tension d'entrée du convertisseur

La plupart du temps, sur les applications industrielles, la tension d'entrée du convertisseur est mesurée principalement pour des raisons de sécurité et il ne paraît pas envisageable d'utiliser la mesure venant d'un estimateur ou d'un observateur, sa reproduction n'étant jamais fiable à 100%.

L'existence d'un capteur pour la tension du bus continu, qui est apparemment nécessaire, peut alors rendre l'estimateur des tensions, aux bornes des capacités flottantes, très fiable et, comme nous l'avons vu, aisément implantable sur un composant numérique de type FPGA.

Ce dernier cas à d'ailleurs été testé en co-simulation en utilisant l'architecture présentée figure III.5.16

Deux simulations utilisant l'architecture de la figure III.5.16 sont présentée. Entre les deux simulations (figure III.5.17 et figure III.5.18), on a fait évoluer la résolution

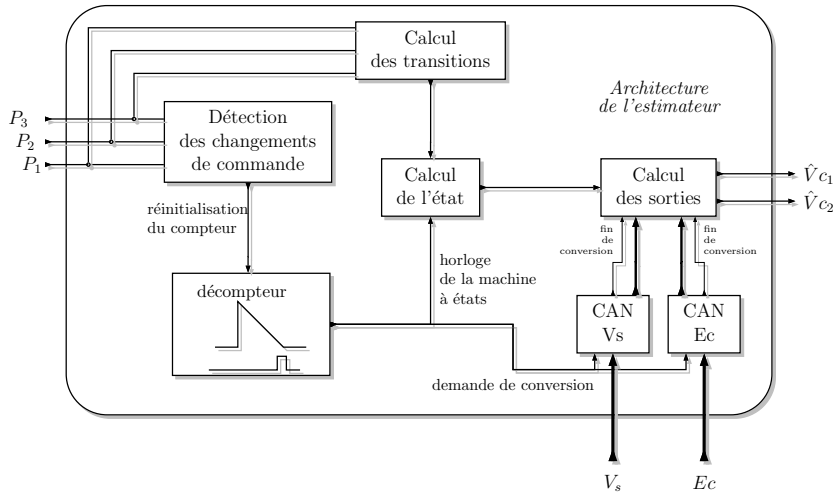


Figure III.5.16: *Architecture du reconstructeur d'état testée en co-simulation*

des CAN afin de voir l'influence de ceux-ci.

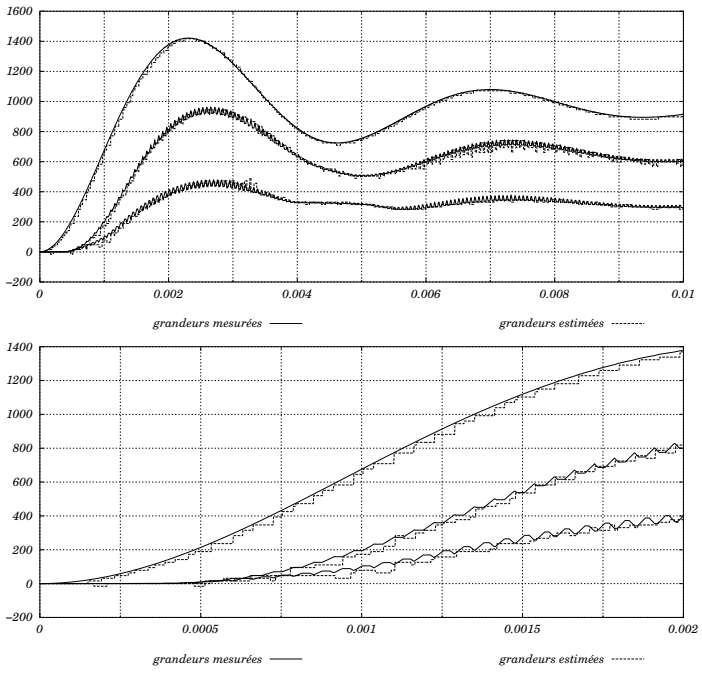


Figure III.5.17: *Machine d'état utilisant V_s et E pour estimée les tensions flottantes, et des CAN d'une résolution de 8 bits*

Cependant ces résultats de simulations correspondent à des simulations en boucle ouverte (l'estimation n'est pas utilisée pour le contrôle des tensions). Une telle co-simulation apparaîtrait plus intéressante en étudiant l'influence de la résolution en boucle fermée.

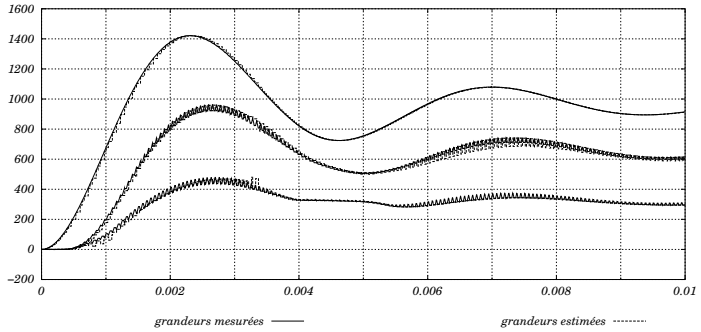


Figure III.5.18: *Machine d'état utilisant V_s et E pour estimée les tensions flottantes, et des CAN d'une résolution de 11 bits*

Les résultats sont tout de même très encourageant, étant donné la simplicité des calculs à effectuer pour mettre en place un tel estimateur.

5.6 Conclusion

Ce chapitre a présenté les convertisseurs multicellulaires séries et la problématique liée à la mesure des tensions aux bornes des capacités flottantes. Nous avons alors étudié le principe du reconstructeur d'état développé en premier lieu dans [26] et [14] afin de rechercher des solutions architecturales pour son implantation dans un composant logique programmable.

Une modification dans l'acquisition des mesures a été effectuée, afin de les rendre plus sûres. Puis dans un deuxième temps, nous avons évalué une variante du principe rendant l'estimateur beaucoup compact dans le cadre d'une implantation sur un composant logique programmable. Cette variante ne s'est avérée concluante que lorsqu'il est possible de mesurer la tension en entrée du convertisseur en plus de la tension de sortie.

Les observateurs utilisant le seul courant de charge sont sensibles aux variations de charges or cet estimateur est totalement indépendant de la charge et présente donc un grand intérêt de ce point de vue. Cependant, afin de pouvoir acquérir dans des intervalles de temps faibles, des mesures de la tension de sortie, cet estimateur nécessite un capteur de tension avec un bande passante importante. De plus, nous n'avons pas testé l'influence du bruit de mesure. Il serait intéressant de réaliser des tests expérimentaux afin d'étudier le comportement de l'estimateur dans un environnement bruité.

Chapitre 6

Conception d'un émulateur/observateur pour convertisseur multicellulaire

6.1 Introduction

Dans le chapitre précédent nous avons étudié l'implantation d'un estimateur pour les tensions aux bornes des capacités flottantes. Cet estimateur est assez performant, mais il ne permet pas de reproduire l'évolution des tensions flottantes à l'intérieur de la période de découpage. Or, les différentes commande en amplitude dont nous avons brièvement énuméré les performances au chapitre précédent (page 107), nécessitent cette information. Il peut donc être intéressant d'élaborer un observateur permettant de reproduire l'ondulation de tension aux bornes des capacités flottantes.

D'autre part, le contrôle des tensions aux bornes des capacités flottantes nécessite un dispositif complexe (capteurs et électronique associée) dont le coût ne plaide pas en sa faveur.

Par conséquent, à l'heure actuel, l'équilibrage des tensions n'est pas contrôlé activement mais simplement accéléré par le biais d'un circuit auxiliaire particulier nommé filtre de rééquilibrage. Cependant l'accélération apportée par ces filtres ne sécurise pas complètement le fonctionnement du convertisseur. Un observateur du système global utilisant un composant de faible coût en vue de la surveillance serait alors non négligeable.

En vue de cet objectif, nous présenterons, dans un premier temps, la conception d'un émulateur temps réel permettant de reproduire ces ondulations de tension. Un tel émulateur pourra d'ailleurs être utilisé, indépendamment de l'observation, pour le test de dispositifs de commande. Puis, on étudiera, dans un second temps, l'association de différent gain d'observation possible à l'émulateur, afin de construire le système d'observation.

6.2 Rôle du filtre de rééquilibrage

Une étude a montré qu'une des propriétés des convertisseurs multicellulaires est l'équilibrage naturel des tensions aux bornes des cellules de commutation[14]. En effet, en générant des rapports cyclique égaux avec un déphasage de $2\pi/3$, les tensions flottantes évoluent naturellement vers leurs valeurs optimales $Vc_1 = \frac{1}{3}E_c$ et $Vc_2 = \frac{2}{3}E_c$. Cet équilibrage naturel est dû aux harmoniques de courant à la fréquence de découpage, et qui sont présents quand les tensions sont déséquilibrées.

La figure III.6.1 indique la présence d'un harmonique à f_{dec} (20kHz) lorsque les ten-

sions flottantes sont déséquilibrées (état instable), lequel disparaît lorsque les tensions flottantes sont équilibrées (état stable).

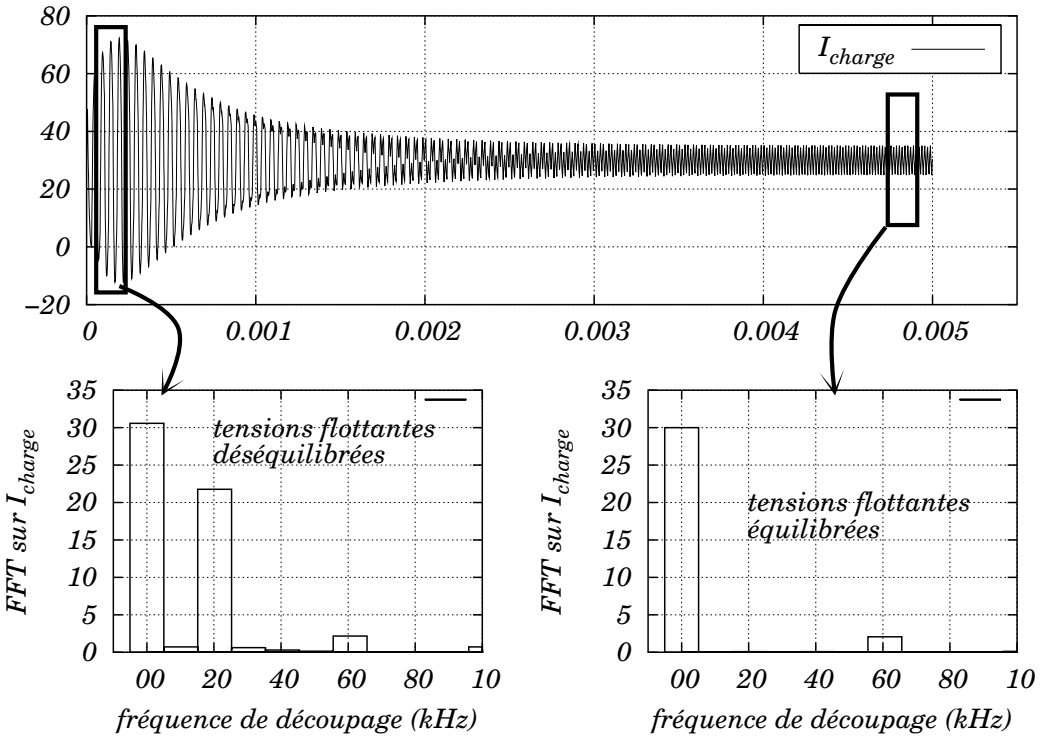


Figure III.6.1: *Spectre fréquentiel des harmoniques de courant dans le cas de tensions déséquilibrées et équilibrées*

Cette dynamique d'équilibrage naturel est par contre lente et peut être fatale à la survie des interrupteurs lors de variations brutales de la tension d'alimentation E_c.
On se propose donc d'accélérer cet équilibrage en ajoutant, en parallèle à la charge, un filtre auxiliaire R_f, L_f, C_f (cf. Figure III.6.2) [14].

6.2.1 Caractéristiques du filtre

Le filtre de rééquilibrage doit avoir une faible impédance dans la gamme de fréquence proche de la fréquence de découpage f_{dec} de façon à amplifier les harmoniques de courant à cette fréquence. Son impédance doit être plus élevée pour des fréquences multiples de $p.f_{dec}$ (ou p correspond au nombre de cellules de commutations) afin de ne pas consommer d'énergie lorsque les tensions sont équilibrées [14].

Pour un convertisseur à trois cellules de commutations, un circuit résonnant accordé à la fréquence de découpage f_d est suffisant.

La fonction de transfert du filtre auxiliaire de rééquilibrage est donnée par l'équation III.6.1.

$$\frac{I(p)}{V_{dec}(p)} = \frac{C_f p}{L_f C_f p^2 + R_f C_f p + 1} \tag{III.6.1}$$

où p représente ici l'opérateur de Laplace.

La figure III.6.3 donne le comportement du filtre dans un diagramme de Bode. Ce tracé correspond aux valeurs de résistance, de capacité et d'inductance qui ont été établies pour obtenir une fréquence de résonance égale à f_{dec}, pour un amortissement et un facteur de qualité donnés.

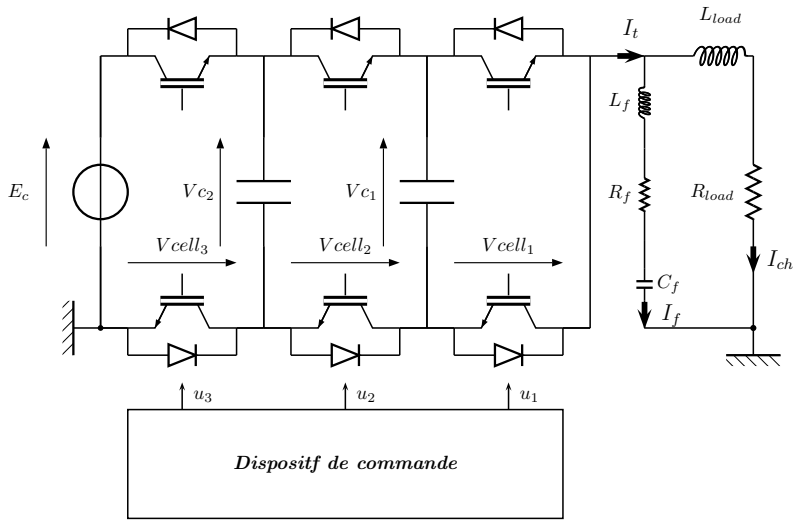


Figure III.6.2: *Convertisseur à 3 cellules de commutation avec filtre de rééquilibrage*

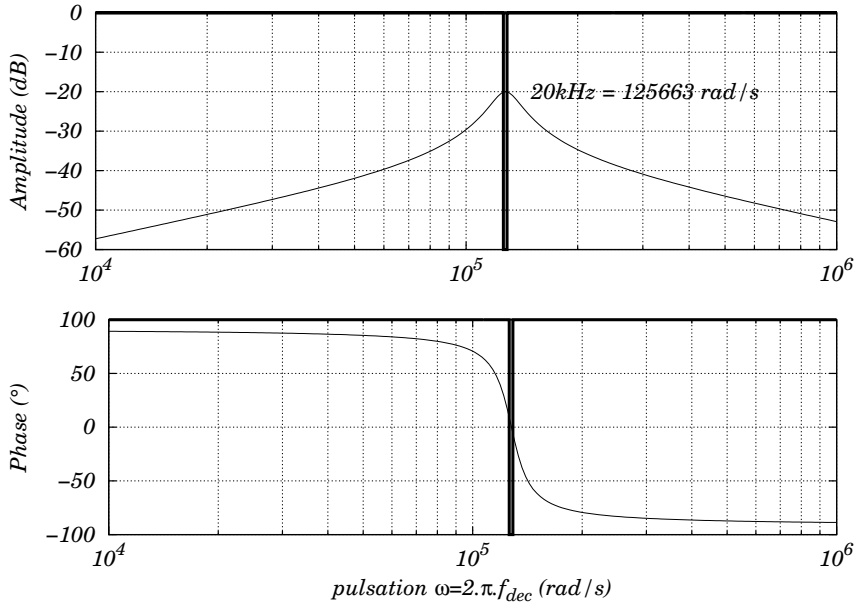


Figure III.6.3: *Diagramme de Bode du filtre de rééquilibrage*

$$\begin{cases} C_f = 136\ \eta F \\ L_f = 450\ \mu H \\ R_f = 10\ \Omega \end{cases}$$

6.3 Conception de l'émulateur temps réel

6.3.1 Approche analogique

Des travaux antérieurs ont été réalisés pour implémenter un modèle échantillonné dont la fréquence est trois fois plus importante que la période de découpage, afin de prendre en compte un minimum d'information fréquentielle sur la variation du courant de charge durant cette période de découpage. Cette information, comme nous venons de le voir, est essentielle pour déterminer l'état d'équilibre des tensions flottantes et donc,

pour être utilisée au sein d'un observateur. Les volumes de calculs demandés par la résolution de ce modèle, étant donné les contraintes de temps, sont encore relativement importants et nécessitent l'utilisation de DSP performants[8].

La conception d'un estimateur, aussi précis que possible, qui pourrait être implanté sur un composant peu coûteux, serait alors d'un intérêt non négligeable.

L'étude d'un convertisseur trois cellules montre que si les règles de commande associées aux cellules de commutation sont respectées[1], on peut en déduire les lois suivantes :

- la tension appliquée à un interrupteur ouvert est donnée par :
  - $V_{ouvert} = Vc_1$ pour un interrupteur de la cellule 1
  - $V_{ouvert} = Vc_2 - Vc_1$ pour un interrupteur de la cellule 2
  - $V_{ouvert} = E - Vc_2$ pour un interrupteur de la cellule 3
- le courant traversant un interrupteur fermé est donné par :
  - $I = I_t$ pour tous les interrupteurs, I_t représentant le courant total, c'est à dire la somme des courants traversant la charge et le filtre.

Ces lois conduisent à un schéma simple reproduisant le fonctionnement du convertisseur trois cellules (figure III.6.4) qui retranscrit les équations du modèle instantané données en (III.6.2).

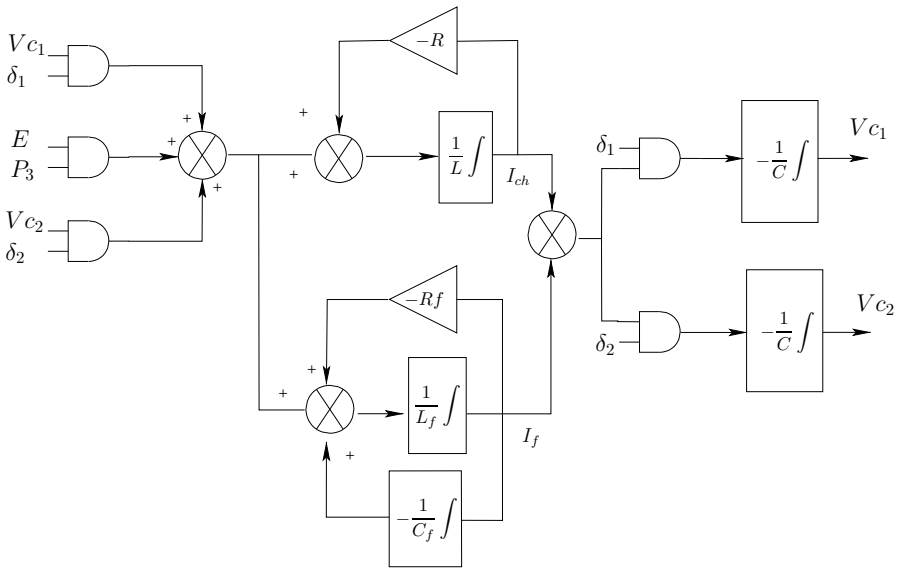


Figure III.6.4: *Schéma Bloc du modèle du convertisseur*

$$\dot{X} = A(P^*).X + B(P^*).E_c \tag{III.6.2}$$

avec :

$$A(P^*) = \begin{bmatrix} -\frac{R}{L} & \frac{\delta_1}{L} & \frac{\delta_2}{L} & 0 & 0 \\ -\frac{\delta_1}{C} & 0 & 0 & -\frac{\delta_1}{C} & 0 \\ -\frac{\delta_2}{C} & 0 & 0 & -\frac{\delta_2}{C} & 0 \\ 0 & \frac{\delta_1}{L_f} & \frac{\delta_2}{L_f} & -\frac{R_f}{L_f} & -\frac{1}{L_f} \\ 0 & 0 & 0 & \frac{1}{C_f} & 0 \end{bmatrix}, \quad \begin{array}{rcl} B(P^*) & = & \begin{bmatrix} \frac{P_3}{L} & 0 & 0 & \frac{P_3}{L_f} & 0 \end{bmatrix}^t \\ P^* & = & \begin{bmatrix} P_1 & P_2 & P_3 \end{bmatrix}^t \\ X & = & \begin{bmatrix} I_{ch} & Vc_1 & Vc_2 & I_f & Vc_f \end{bmatrix}^t \\ \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} & = & \begin{bmatrix} P_1 - P_2 \\ P_2 - P_3 \end{bmatrix} \end{array}$$

On peut voir que ce schéma fait appel à des intégrateurs (capacités et inductances) et des portes logiques représentant les interrupteurs. Ces portes sont en fait des interrupteurs analogiques pilotés par les signaux δ_1 et δ_2. Un tel modèle a été réalisé avec des composants analogiques[76]. Bien qu'il soit difficile de garantir la précision obtenue

[1] seul un interrupteur de chaque cellule est conducteur à la fois

par ce circuit, nous pensons que cette solution est intéressante. Cependant, les composants appropriés (FPAA)[2] ne semblent pas encore assez fiables pour une réalisation industrielle .

Nous allons donc réaliser l'implantation de ce modèle sur un FPGA et tout d'abord, procéder à la discrétisation des équations.

6.3.2 Échantillonnage du modèle.

La discrétisation du modèle analogique nécessite l'introduction d'un échantillonnage. La période d'échantillonnage doit donc être plus petite que les constantes de temps électriques du système.

Nous supposerons dans un premier temps, que ce ratio est suffisamment important pour permettre d'utiliser une méthode de résolution simple telle que la méthode d'Euler. Le système échantillonné est alors décrit par l'équation (III.6.3) :

$$X(j+1) = X(j) + T_e.\dot{X}(j) \tag{III.6.3}$$

Nous avons réalisé une première co-simulation utilisant cette méthode. Les résultats sont acceptables pour des périodes d'échantillonnage inférieures à $100\,\eta s$, alors que des erreurs importantes apparaissent pour des périodes d'échantillonnage plus grandes. La figure III.6.5 illustre cette limitation en présentant la forme d'onde du courant dans le filtre de rééquilibrage au démarrage du convertisseur. Nous signalons ici que nous nous sommes uniquement intéressé à ce courant étant donné qu'il représente, avec la tension aux bornes de la capacité du filtre C_f, la grandeur ayant la dynamique la plus importante dans ce système.

Comme on peut le constater sur la figure III.6.5, le courant traversant le filtre de rééquilibrage est une bien une image du déséquilibre des tensions flottantes.

Sur les agrandissements de la figure III.6.5, la réponse du modèle continu est comparée aux réponses du modèle échantillonné pour une période d'échantillonnage de $100\,\eta s$ (courbe de gauche) et $480\,\eta s$ (courbe de droite). On peut voir qu'à $480\,\eta s$ l'intégration des équations d'état n'est pas compatible avec la méthode d'Euler, du moins pour l'émulation de filtre de rééquilibrage. En effet, le problème ne se pose pas pour les autres variables du système qui évoluent plus lentement. Or, en vue de l'implantation sur un FPGA, nous cherchons, a priori, à augmenter la période d'échantillonnage afin de ne pas rencontrer de problèmes dus aux temps de propagation lors de l'intégration.

Une analyse du système montre que celui-ci est composé de deux parties aux caractéristiques très différentes. D'une part, une partie composée d'interrupteurs et de capacités flottantes, d'autre part, une partie correspondant au filtre de rééquilibrage. La première partie est un système à topologie variable, c'est-à-dire que la structure du circuit électrique varie en fonction de l'état des interrupteurs. Ceci se retrouve dans les équations du modèle instantané (III.6.2) par le fait qu'une partie de la matrice A dépend de l'état des interrupteurs. A contrario, l'équation d'état du filtre ne dépend pas des signaux de commande, mais est caractérisée par une constante de temps plus petite. Pour obtenir de meilleurs résultats avec une fréquence d'échantillonnage plus petite, seule la résolution des équations du filtre de rééquilibrage nécessite une amélioration. Étant donné la topologie fixe de cette partie, une méthode plus élaborée peut être implémentée. Les équations différentielles du système initial sont alors scindées pour former deux sous-systèmes (III.6.4).

$$\dot{X}_c = A_c(P^*).X_c + B_c(P^*).U_c \tag{III.6.4a}$$

$$\dot{X}_f = A_f.X_f + B_f(P^*).U_f \tag{III.6.4b}$$

[2]Field Programmable Analog Array

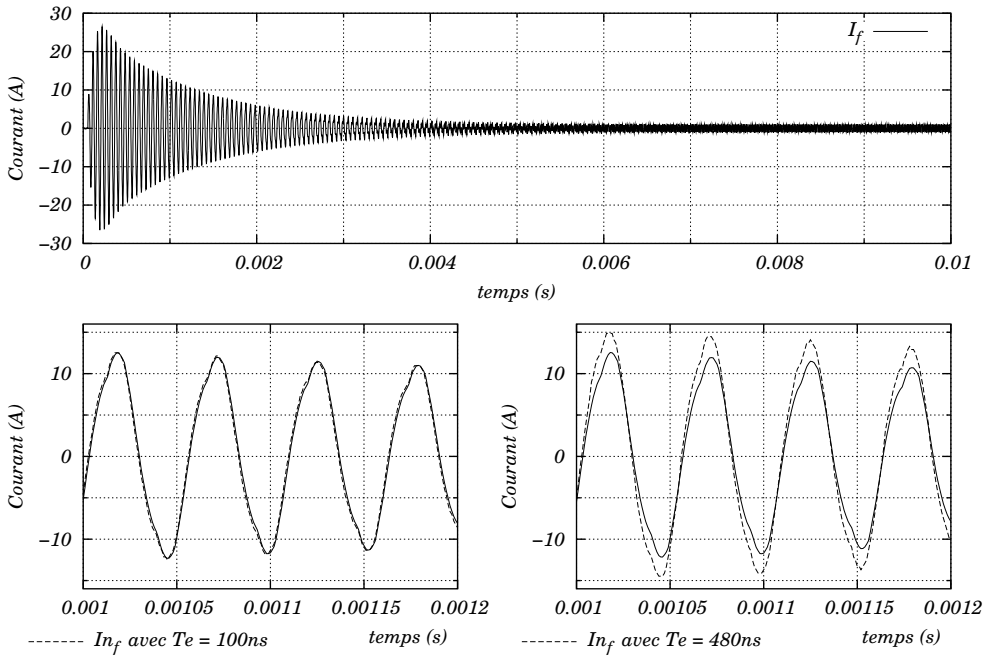


Figure III.6.5: *Simulation du modèle du convertisseur utilisant la méthode d'Euler*

avec :

$$X_f = \begin{bmatrix} I_f & Vc_f \end{bmatrix}^T \qquad X_c = \begin{bmatrix} I_{ch} & Vc_1 & Vc_2 \end{bmatrix}^T$$

$$A_c(P^*) = \begin{bmatrix} -\frac{R}{L} & \frac{\delta_1}{L} & \frac{\delta_2}{L} \\ -\frac{\delta_1}{C} & 0 & 0 \\ -\frac{\delta_2}{C} & 0 & 0 \end{bmatrix}, \qquad A_f = \begin{bmatrix} -\frac{R_f}{L_f} & -\frac{1}{L_f} \\ \frac{1}{C_f} & 0 \end{bmatrix},$$

$$B_c(P^*) = \begin{bmatrix} \frac{P_3}{L} & 0 \\ 0 & -\frac{\delta_1}{C} \\ 0 & -\frac{\delta_2}{C} \end{bmatrix}, \qquad B_f(P^*) = \begin{bmatrix} \frac{P_3}{L_f} & \frac{\delta 1}{L_f} & \frac{\delta_2}{L_f} \\ 0 & 0 & 0 \end{bmatrix},$$

$$U_c = \begin{bmatrix} E \\ I_f \end{bmatrix}, \qquad U_f = \begin{bmatrix} E & Vc_1 & Vc_2 \end{bmatrix}^T$$

L'équation (III.6.4b) fait intervenir une matrice constante A_f, caractéristique de la topologie fixe de ce sous-système. Il est alors possible de déterminer l'état X_f à l'instant $(j+1).T_e$ à partir de l'état à l'instant $j.T_e$ en utilisant l'équation (III.6.5).

$$\dot{X}_f(j+1) = X_f(j) + (F - I).X_f(j) + G(P^*).U_f(j) \tag{III.6.5}$$

avec :

$$\begin{cases} F &= e^{A_f.Te}, \\ G &= \int_{Te.j}^{Te.(j+1)} e^{A_f.(Te.j-\tau)} d\tau.B_f(P^*) \end{cases}$$

Cette formulation est théoriquement exacte quelque soit la période d'échantillonnage. Cependant, le filtre de rééquilibrage n'est qu'un sous-système et la période d'échantillonnage doit rester compatible avec la dynamique des grandeurs échangées entre les

deux sous-systèmes. Cette exigence permet néanmoins de plus amples périodes d'échantillonnage que celles nécessitées par la méthode d'Euler.

6.3.3 Résolution Numérique

La résolution numérique utilisant un FPGA ne nous permet pas d'utiliser des variables écrites en virgule flottante. En effet, les calculs exploitant ce type sont plus coûteux en nombre de portes et ne sont pas encore couramment utilisés. En fait, les calculs sont usuellement effectués en "complément à deux".

De manière arbitraire, nous avons choisi d'utiliser le même facteur d'échelle pour les grandeurs de même nature (tension, courant). Les facteurs d'échelle sont donnés par :

$$K_i = \frac{I_{max}}{2^{N-1}-1} \tag{III.6.6}$$

$$K_v = \frac{V_{max}}{2^{N-1}-1} \tag{III.6.7}$$

dans lequel :
- V_{max} est considérée comme la valeur maximale de tension,
- I_{max} est considérée comme la valeur maximale de courant,
- N est le nombre de bits utilisé pour coder ces grandeurs.

Dans le but d'expliquer la numérisation des équations (III.6.4) et pour ne pas ajouter de notations complexes, nous avons décidé de décrire une seule équation. Mais on notera que le principe peut être appliqué aux autres équations.

L'équation caractérisant l'évolution de Vc_1 est utilisée comme exemple :

$$Vc_1(j+1) = Vc_1(j) - T_e.\frac{\delta_1}{C}.[I_{ch}(j) + I_f(j)] \tag{III.6.8}$$

Après application des facteurs d'échelle, toutes les entrées (courants et tensions) sont codées sur N bits en complément à deux et l'équation devient :

$$\mathbb{V}c_1(j+1) = \mathbb{V}c_1(j) - \delta_1.\alpha.[\mathbb{I}_{ch}(j) + \mathbb{I}_f(j)]$$

avec $\alpha = Te.\frac{K_i}{K_v}.\frac{1}{C}$, $\mathbb{V}c_1 = \frac{1}{K_v}.Vc_1$

Le coefficient résultant α est plus petit que l'unité, et donc l'erreur d'arrondi le rendrait nul. L'équation doit être multipliée par un coefficient (2^l) dans le but de rendre l'erreur d'arrondi acceptable. L'équation numérique à implanter est alors la suivante :

$$\mathbb{V}lc_1(j+1) = \mathbb{V}lc_1(j) - \delta_1.\alpha_l.[\mathbb{I}_{ch}(j) + \mathbb{I}_f(j)] \tag{III.6.9}$$

avec : $\alpha_l = 2^l.Te.\frac{K_i}{K_v}.\frac{1}{C}$ et $\mathbb{V}lc_1 = 2^l.\mathbb{V}c_1$

La performance de l'émulateur dépend alors de plusieurs paramètres. En supposant que les coefficients du modèle sont constants, les différents paramètres qui peuvent affecter la résolution sont :
- la période d'échantillonnage **Te**,
- le nombre de bits **N** utilisé pour coder les grandeurs de tension et de courant,
- la précision des coefficients après multiplication des matrices par $\mathbf{2^l}$.

Notons que la place allouée sur le FPGA dépend aussi de ces paramètres. Leur choix doit donc être fait avec attention.

Nous allons pour cela nous aider de la co-simulation.

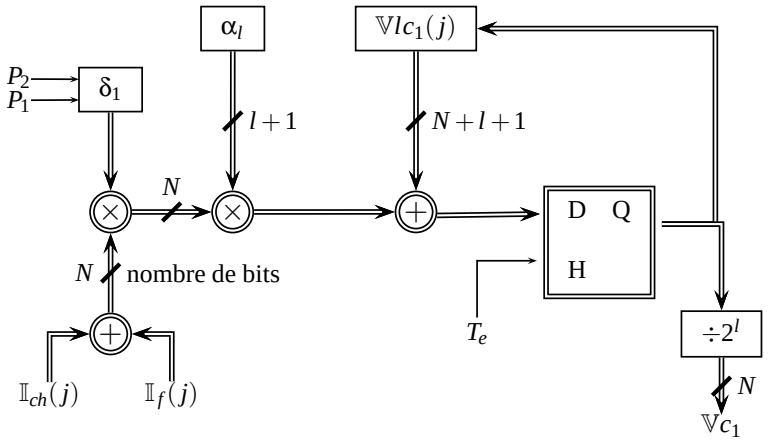


Figure III.6.6: *Calcul numérique de* Vc_1

6.3.4 Co-simulation au niveau fonctionnel

L'émulateur étant spécifié en VHDL, nous pouvons effectuer une co-simulation de *premier niveau* et observer l'impact des paramètres $\{Te, N, l\}$. Les figures III.6.7-a- et III.6.7-b- montrent, pour 2 valeurs différentes de N, les résultats obtenus au démarrage du convertisseur. Ces résultats sont comparés à ceux obtenus à l'aide du modèle continu.

Nous pouvons voir sur la figure III.6.7-a- que les tensions aux bornes des capacités évoluent vers les bonnes valeurs moyennes, même en diminuant le nombre de bits à $N = 8$. Cependant, la précision des coefficients liées au paramètre 2^l a un impact plus significatif.

Après plusieurs co-simulations, cela nous a conduit à choisir un paramètre l égal à 11, ce qui donne une erreur maximale de 4.16% sur les coefficients. Notons aussi que la période d'échantillonnage est de $480\,\eta s$, montrant ainsi l'intérêt de la décomposition en deux sous systèmes.

Différentes co-simulations nous ont montré que la troncature des entrées liée au paramètre N n'a pas une grande influence sur le comportement de l'estimateur.

Le choix de N dépend en fait de l'utilisation de l'émulateur. Si le but est d'obtenir la valeur moyenne des tensions aux bornes des capacités flottantes, on pourra utiliser une valeur faible de N. Si, par contre, nous voulons restituer les ondulations de tensions, il nous faudra augmenter la valeur de ce paramètre.

La spécification VHDL utilisée dès le début du cycle de conception va nous permettre d'accélérer celui-ci. En effet, une fois que les paramètres sont déterminés par une co-simulation de *premier niveau*, le code VHDL ne nécessite plus qu'un raffinement afin de le rendre entièrement synthétisable.

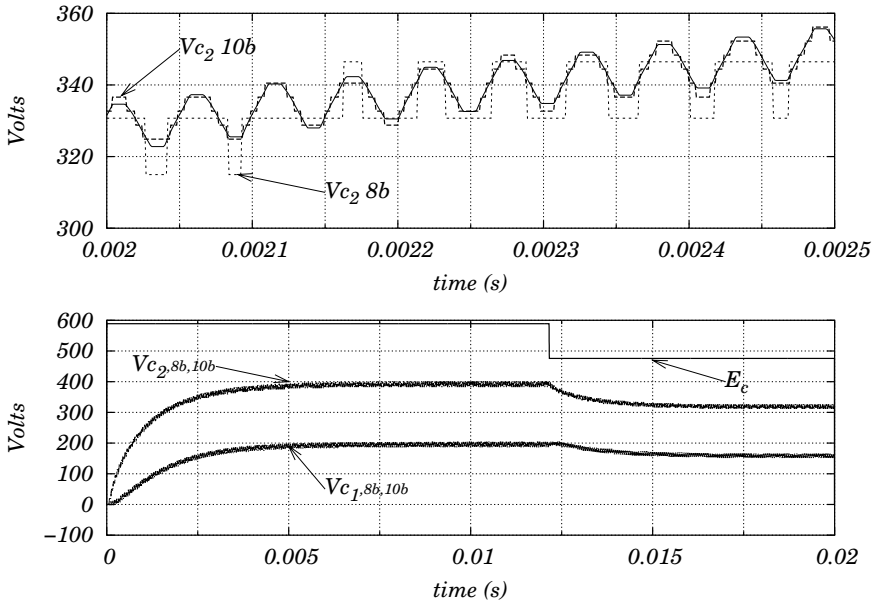
6.3.5 Implantation de l'émulateur sur FPGA

Comme nous l'avons vu au chapitre 4 (page 73), l'implantation d'un code VHDL est essentiellement composé de deux étapes :

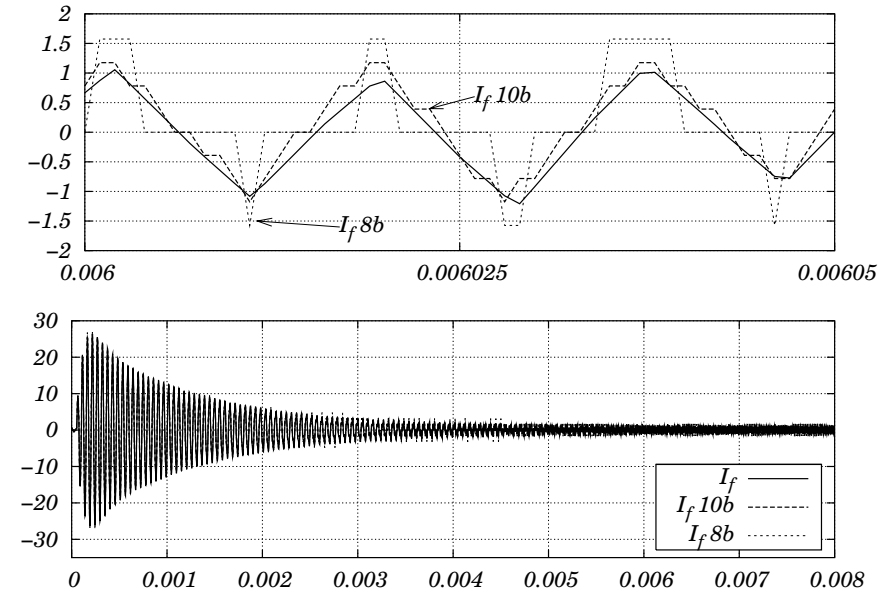
1. Synthèse,
2. Placement et Routage.

Ces deux étapes ont été validées par des co-simulations de *niveau 2 et niveau 3* (cf. § 4.2.1 page 73).

Dans le but de valider entièrement la conception de l'émulateur, nous avons réalisé son implémentation sur un composant Altera Acex 1K50.



a- Tensions aux bornes des capacités flottantes au démarrage et après un échelon de la tension d'entrée.



b- Courant du filtre de rééquilibrage au démarrage.

Figure III.6.7: *Co-simulations fonctionnelles de l'émulateur avec* $Te = 480\,\eta s$, $l = 11$, $N = \mathbf{8}$ *ou* $N = \mathbf{10}$

Pour valider son fonctionnement sur le composant, il nous a fallu générer les ordres de commandes dédiés aux interrupteurs de puissance et la tension d'entrée du convertisseur E_c. Nous avons donc implanté une MLI sur le même FPGA pour générer les ordres de commande. L'image de la tension d'entrée E_c est, quant à elle, transmise dans un registre du FPGA par un microprocesseur directement relié au composant logique programmable.

Cette connexion nous a ensuite permis de lire et de mémoriser à tout instant les différentes variables d'état de l'émulateur qui sont situées dans des registres particuliers au sein du FPGA.

L'évaluation de l'émulateur a été réalisée avec les paramètres suivants :
$Te = 480\,\eta s,\ N = 10,\ f_d = 18.67\,kHz,\ E = 590\,V.$
Le composant ACEX 1K50 est alors rempli approximativement à 70%.

Le résultat présenté en figure III.6.8 montre les tensions aux bornes des capacités flottantes au démarrage du convertisseur avec des tensions nulles au départ et des rapports cycliques constants et égaux à 0.5. Elles sont comparées aux courbes du modèle continu.

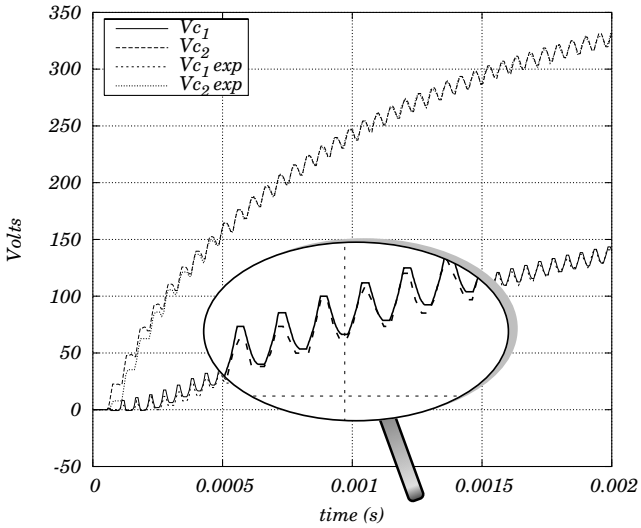


Figure III.6.8: *Comparaison entre la simulation du modèle de type circuit et l'émulateur implanté sur le* FPGA

6.4 Premiers tests sur la conception de l'observateur

Comme nous l'avons dit précédemment, cet estimateur temps réel est destiné à restituer les tensions flottantes aux bornes des capacités. Vu les résultats obtenus, il peut être utilisé tel quel comme estimateur.

Cependant, il ne peut être garanti que les conditions initiales du convertisseur et de l'estimateur soient identiques. De plus, la discrétisation des variables peut introduire des erreurs cumulatives qui peuvent devenir significatives après un grand nombre de périodes d'échantillonnage. C'est pourquoi il est nécessaire d'agir sur certains paramètres de l'émulateur à l'aide d'un rebouclage adapté. On parlera alors d'observateur.

6.4.1 Principe de l'observateur dédié au convertisseur multicellulaire

Une étude précédente [76] proposait un rebouclage permettant à l'estimateur de converger vers les tensions du convertisseur. Ce rebouclage est composé de deux capteurs pour les raisons suivantes.

Dans la charge R, L, la résistance peut varier entre $[0, +\infty]$. Ainsi l'équation du modèle instantané reproduisant le courant dans la charge ne peut plus être calculée par l'émulateur. On utilise alors un premier capteur permettant de récupérer le courant de charge. Ce capteur n'engendre, cependant, pas de sur-coût étant donné qu'il est nécessaire à la régulation du courant.

Lorsque les tensions émulées et les tensions réelles ont la même valeur initiale, le comportement de l'émulateur reproduit fidèlement le fonctionnement du convertisseur réel. Cependant, si leurs conditions initiales sont différentes ou si des perturbations interviennent sur le système réel, cela n'est plus vérifié.

C'est pourquoi, il est nécessaire d'obtenir une information sur l'équilibrage des tensions flottantes. Or, nous avons vu que le courant circulant dans le filtre R_f, L_f, C_f véhiculait une image du déséquilibre puisqu'il permet un rééquilibrage plus rapide. Ce courant filtre est donc capté, puis comparé au courant estimé. l'erreur obtenue est ensuite réinjectée dans l'émulateur.

La structure de cet observateur est schématisée sur la figure III.6.9.

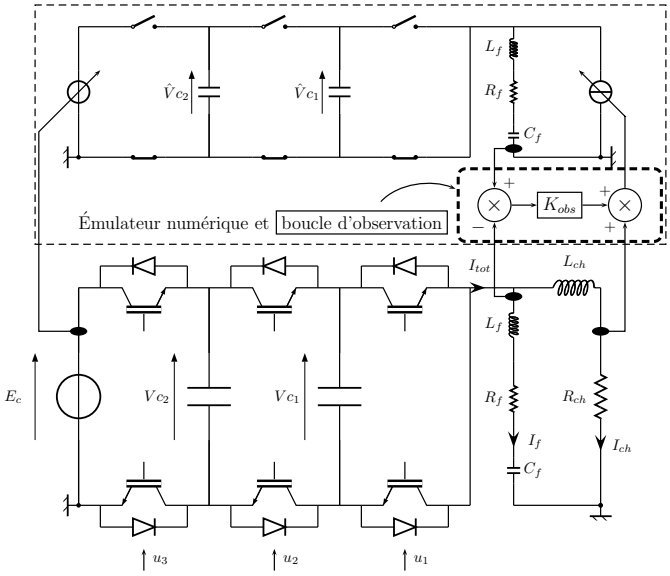


Figure III.6.9: *Schéma bloc de la boucle d'observation utilisant deux capteurs*

Un résultat de la simulation fonctionnelle sans les contraintes architecturales est présenté sur la figure III.6.10.

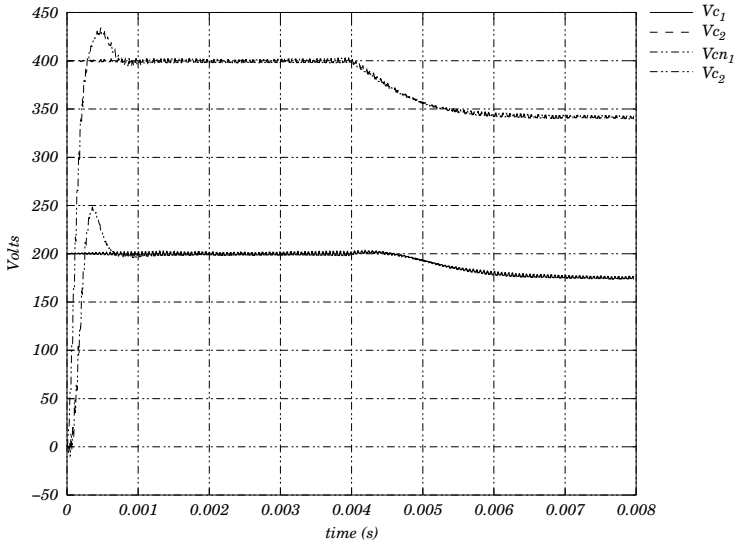


Figure III.6.10: *Simulation fonctionnelle de la boucle d'observation utilisant 2 capteurs de courant*

Cette simulation montre les tensions flottantes avec des conditions initiales correspondant à l'équilibre pour le convertisseur, et nulles pour celles de l'émulateur. Les

courbes montrent la convergence rapide des tensions de l'émulateur vers celles sur système réel et le suivi lorsqu'il y a une variation sur la tension d'entrée.

Néanmoins, cette structure nécessite un capteur supplémentaire. Nous avons donc cherché à diminuer le nombre de ces capteurs, en étudiant d'autres structures, dérivées de ce principe.

6.4.2 Étude des structures de rebouclage

Le but, si nous nous intéressons à réduire le montage à un seul capteur, est d'extraire par filtrage, les harmoniques qui nous intéressent à partir d'une seule mesure du courant total, comme le montre la figure III.6.11.

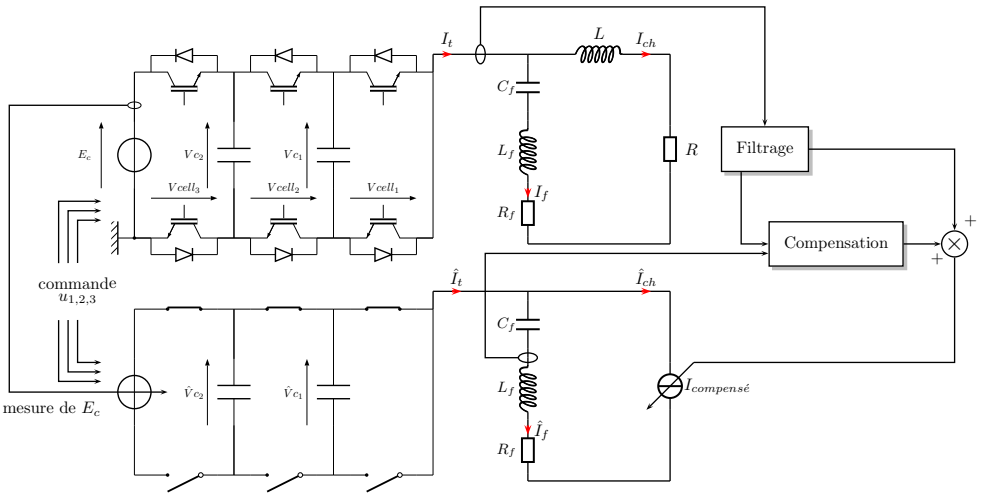


Figure III.6.11: *Solution à un capteur*

Le module de filtrage doit permettre de sélectionner les harmoniques de courant qui conditionnent la convergence des tensions flottantes à leurs valeurs de stabilité,

Le module de compensation réalise la différence entre les états réels et émulés, à laquelle il rajoute un gain [3].

La valeur réinjectée dans l'émulateur correspond alors au courant de charge reconstruit[4].

Tout le problème consiste donc à déterminer les paramètres du filtre qui permettront d'obtenir les bonnes harmoniques et à fixer un mode de compensation qui permette de faire converger rapidement l'émulateur vers les valeurs du convertisseur. On a donc développé différentes structures que l'on décrit dans les rubriques ci-dessous [5]

6.4.3 Structure n°1

Cette première structure utilisera 2 filtres développés à partir des fonctions de transfert exactes $\frac{I_f}{I_t}$ et $\frac{I_{ch}}{I_t}$. Ces fonctions sont notées $it2ich$ pour l'extraction du courant de charge et $it2if$ pour le courant dans le filtre de rééquilibrage.

On a recherché à travers la structure de la figure III.6.12 à reconstruire fidèlement les courants filtre et les courants de charge nécessaires à la convergence de l'émulateur.

[3] appelé aussi gain de l'observateur

[4] c'est à dire le courant de charge + l'erreur de compensation

[5] Ces structures n'ont été testées qu'en utilisant des simulations comportementales à l'aide l'outil de simulation MATLAB/SIMULINK

Nous avons pour cela considéré une charge nominale et utilisé les valeurs de cette charge[6] pour exprimer le courant de charge et le courant de filtre en fonction du courant total.

$$\frac{I_f}{I_t}(p) = \frac{\frac{V_s(p)}{Z_f(p)}}{\frac{V_s(p)}{Z_t(p)}} = \frac{Z_t(p)}{Z_f(p)} \tag{III.6.10}$$

avec les impédances de charge et de filtre Z_f et Z_{ch} :

$$Z_f(p) = R + L.p \tag{III.6.11}$$

$$Z_{ch}(p) = R_f + L_f.p + \frac{1}{C_f.p} \tag{III.6.12}$$

et l'impédance totale :

$$Z_t(p) = \frac{Z_f(p) * Z_{ch}(p)}{Z_f(p) + Z_{ch}(p)} \tag{III.6.13}$$

Donc,

$$it2if(p) = \frac{I_f}{I_t}(p) = \frac{Z_{ch}(p)}{Z_f(p) + Z_{ch}(p)} \tag{III.6.14}$$

$$it2ich(p) = \frac{I_{ch}}{I_t}(p) = 1 - it2if(p) \tag{III.6.15}$$

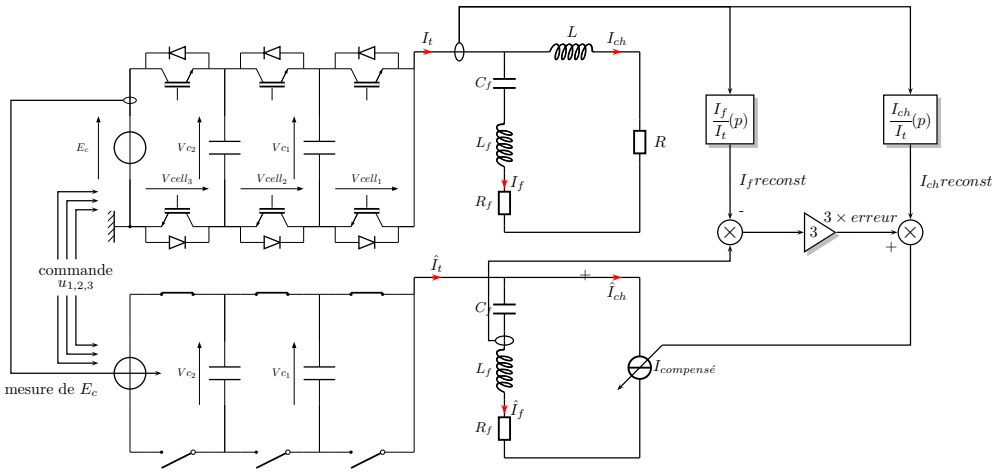


Figure III.6.12: *Observateur basé sur les fonctions de transferts I_f/I_t et I_{ch}/I_t*

On implante donc les fonctions de transfert ci-dessus dans la structure de la figure III.6.12, en considérant les valeurs suivantes pour les différents paramètres de la charge et du filtre :

$$\begin{cases} C_f = 136\ \eta F \\ L_f = 450\ \mu H \\ R_f = 10\ \Omega \end{cases} \quad \begin{cases} L = 100\ \mu H \\ R = 10\ \Omega \end{cases}$$

[6] valeurs issues du système réel

On obtient alors des résultats très satisfaisants lorsque la charge du convertisseur correspond à la charge utilisée dans le calcul des filtres, comme le montre la figure III.6.13.

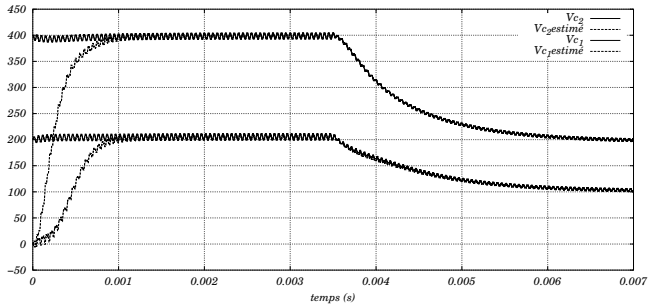


Figure III.6.13: *Structure n° 1 : tracé des tensions flottantes*

Temps de convergence à 3% : 0.99 ms

On obtient une bonne dynamique de convergence de l'observateur. Cependant l'utilisation de ce rebouclage le rend peu robuste par rapport aux variations paramétriques. En effet, les fonctions de transfert sont calculées en fonction des impédances du circuit et sont donc totalement dépendantes de ces valeurs. Cet aspect est loin d'être négligeable vu que la valeur de résistance de charge peut être variable. Il faudrait systématiquement modifier en conséquence les équations de transfert.

6.4.4 Structure n°2

Le courant dans le filtre de rééquilibrage ayant un rôle important dans le comportement du convertisseur nous avons déplacé le capteur de courant pour aboutir à la structure donnée en figure III.6.14

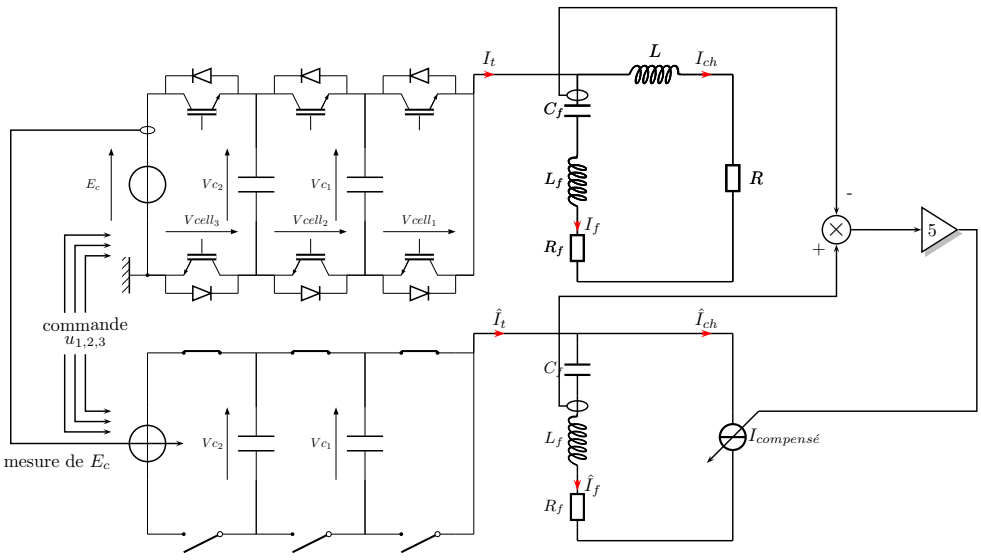
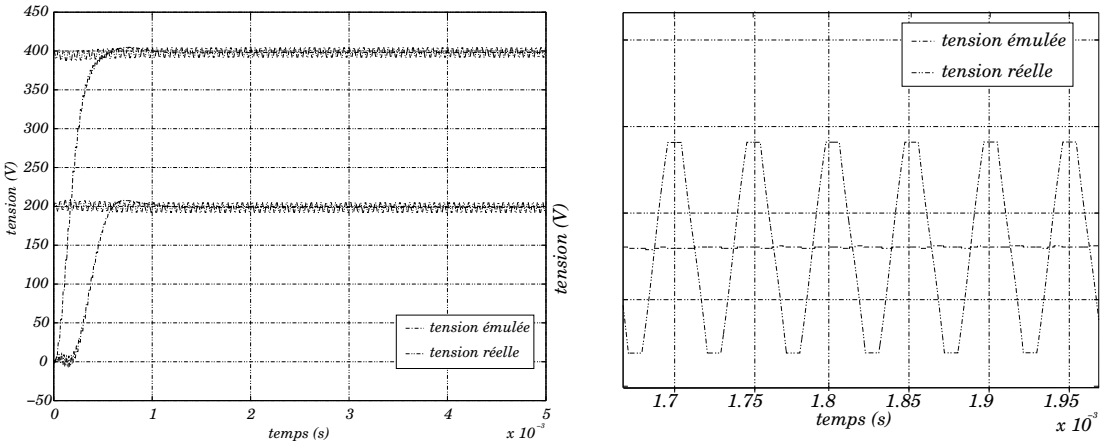


Figure III.6.14: *Observateur basé sur la comparaison des courants filtre*

L'inconvénient de cette structure est que nous n'avons plus accès au courant de charge. Ce qui se retrouve sur le tracé de la figure III.6.15, où même si les tensions

possèdent une bonne dynamique de convergence, elles ne retranscrivent plus l'amplitude des tensions flottantes.

L'étude de cette structure permet toutefois de conclure sur le fait que ce sont bien les harmoniques du courant du filtre de rééquilibrage qui véhiculent l'information nécessaire pour assurer la convergence de l'observateur.



a-

b- zoom sur une des tension flottantes (Vc_2)

Figure III.6.15: *Structure n° 2 : tracé des tensions flottantes*

Temps de convergence à 3% : 0.57 ms

6.4.5 Structure n°3

L'étude des structures précédentes nous a montré que le courant circulant dans le filtre de rééquilibrage avaient un rôle primordial, mais que son extraction à partir du courant total ne pouvait pas se faire de manière exacte.

Comme nous n'allons pas adapter les équations du filtre $\frac{I_f}{I_t}(p)$ en fonction des valeurs prises par la charge (résistance R), on propose d'utiliser une troisième structure utilisant un filtre moins sélectif.

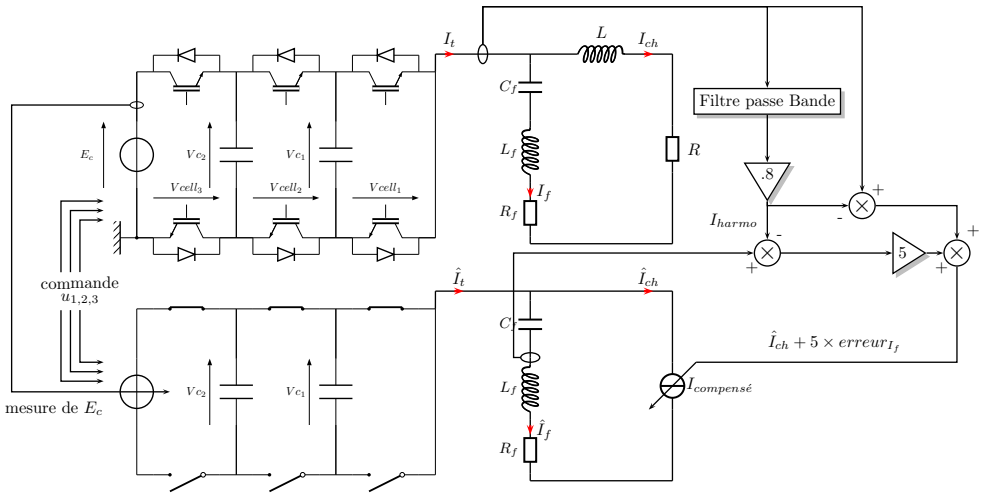


Figure III.6.16: *Structure retenue pour l'observateur*

Pour cela on utilise un filtre du second ordre dont on fera varier la sensibilité, la pulsation propre et la bande passante.

$$\begin{cases} \text{Q : facteur de qualité} \\ \text{K : gain statique} \\ \omega_0 \text{ : pulsation propre} \end{cases} \tag{III.6.16}$$

On utilise alors une structure où le filtre $it2if$ est assimilé à un passe bande $H(p)$ du second ordre défini par l'équation III.6.17

$$H(p) = \frac{K.p}{\frac{Q}{\omega_0}.p^2 + p + Q.\omega_0} \tag{III.6.17}$$

On se propose donc de modifier la bande passante du filtre passe-bande par rapport à la fonction de transfert initiale $it2if(p)$, en la décalant dans la zone haute fréquence, de manière à augmenter l'influence de l'harmonique à $3f_{dec}$. Ceci se traduit par le choix des paramètres suivants pour le filtre passe-bande :

$$\begin{cases} Q = 1 \\ K = 1 \\ \omega_0 : 2\pi \times 25000 \end{cases}$$

La simulation visible à la figure III.6.15 indique que la convergence est plus rapide et valide donc le fait que l'harmonique de courant à $3f_{dec}$ contribue bien à la convergence du système.

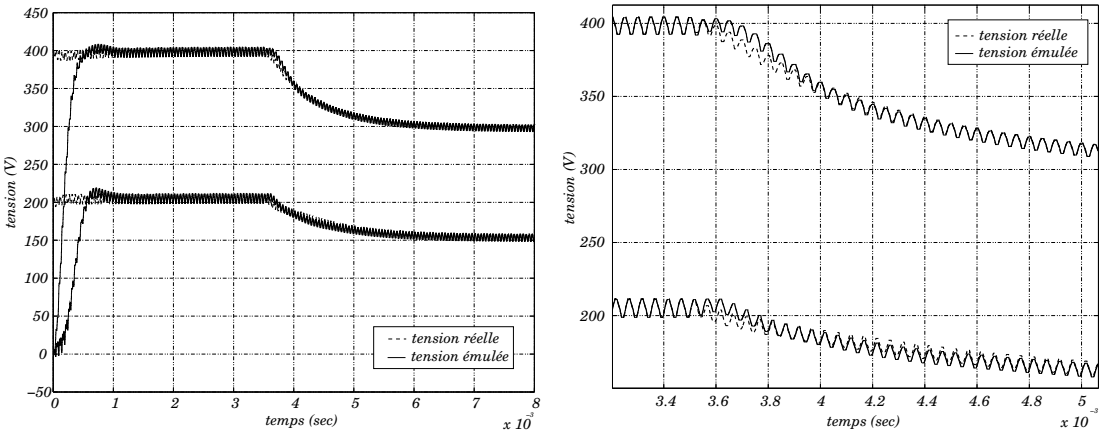


Figure III.6.17: *Structure n°3 : tracé des tensions flottantes*

Temps de convergence à 3% : 0.72 ms

La figure III.6.17 révèle, par contre, des problèmes de convergence de l'émulateur face à des variations de la tension d'entrée, qui sont minimes lorsque le paramètre de charge est nominal mais qui augmentent, lorsque la résistance de charge varie.

Il est aussi important de remarquer qu'en terme de robustesse, les tracés de la figure III.6.18 révèlent que les paramètres du filtre sont moins dépendants du paramètre de charge (résistance R) que dans le cas ou l'on utilise la fonction de transfert exacte. Les résultats de la simulation traduisent le fait que pour une valeur de résistance de charge dix fois supérieure, l'observateur se comporte encore convenablement (cf. Figure III.6.18).

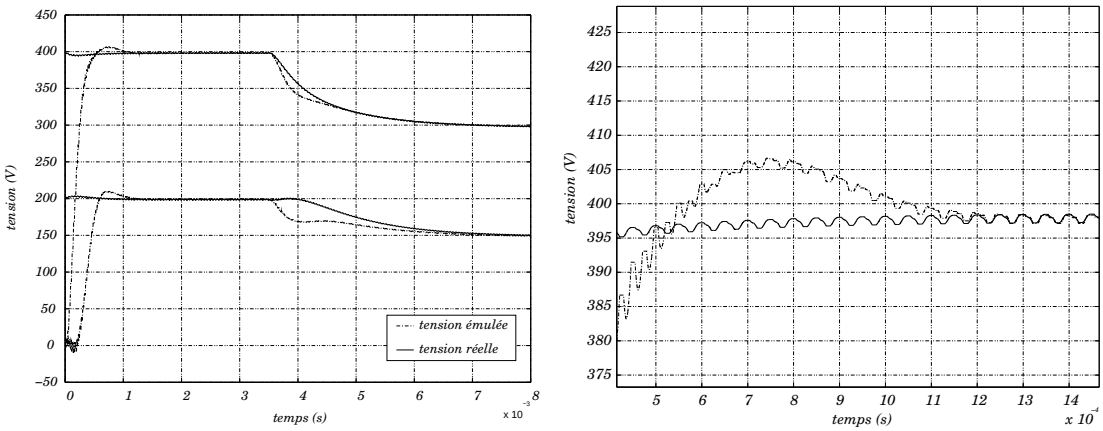


Figure III.6.18: *Robustesse de l'observateur face aux variations de la résistance de charge* $R = 10 \times R_{nominal}$

6.4.6 Conclusion sur les structures étudiées

Au travers des différentes structures étudiées, on relèvera l'importance de baser le module de compensation sur la comparaison des courants du filtre de rééquilibrage réel et estimé, ainsi que de reconstituer le courant de charge par la différence entre le courant total et la sortie du filtre passe-bande.

En ce qui concerne le paramétrage du filtre passe-bande, on a mis en évidence l'importance de chacun des harmoniques présents dans le courant filtre et notamment des harmoniques à f_{dec} et $3 \cdot f_{dec}$ qui sont en grande partie responsables de la convergence des valeurs de l'émulateur vers les valeurs réelles.

Après avoir déterminé les paramètres et la structure de la boucle d'observation à utiliser, nous avons étudié sa conception numérique, à l'aide de la co-simulation.

6.5 Conception numérique de la boucle d'observation

6.5.1 Modèle numérique de la boucle d'observation

L'échantillonnage de l'observateur consiste à utiliser une formulation discrète pour le filtre passe-bande dont nous rappelons l'équation :

$$H(p) = \frac{K.p}{\frac{Q}{\omega_0}.p^2 + s + Q.\omega_0} \tag{III.6.18}$$

avec :

$$\begin{cases} Q = 1 \\ K = 1 \\ \omega_0 : 2\pi * 25000 \end{cases}$$

Il s'écrira comme suit en discret :

$$H(z) = \frac{n_1 z + n_2}{z^2 + d_1 z + d_2} \tag{III.6.19}$$

avec pour une période d'échantillonnage de $480\eta s$:

$$\begin{array}{ll} n_1 = 0.07546 & d_2 = 0.9245 \\ n_2 = -0.07546 & d_1 = -1.919 \end{array}$$

En vue de l'implantation de ce filtre, on préférera l'exprimer en utilisant l'opérateur z^{-1} qui permet de déterminer l'équation de récurrence comme suit :

$$H(z^{-1}) = \frac{n_1.z^{-1} + n_2.z^{-2}}{1 + d_1.z^{-1} + d_2.z^{-2}} \tag{III.6.20}$$

ce qui donne :

$$\hat{I}_{harmo}(k) = n_1.I_t(k-1) + n_2.I_t(k-2) - d_1.\hat{I}_{harmo}(k-1) - d_2.\hat{I}_{harmo}(k-2) \tag{III.6.21}$$

Une fois l'échantillonnage temporel de l'équation calculé, il faut mettre toutes les grandeurs décimales du filtre sous forme d'entiers, tout en choisissant judicieusement les coefficients de mise en forme pour éviter d'une part, de coder inutilement sur un nombre de bits trop important et, d'autre part, de perdre en précision.

L'équation de ce filtre utilise comme entrée le courant total que l'on acquiert à l'aide d'un capteur suivi d'un convertisseur analogique numérique. La quantification des équations du filtre, utilise le facteur d'échelle K_i donné en III.6.6. Certains coefficients ne peuvent comme dans le cas de l'émulateur, pas être codés en entier sans d'abord être multipliés par un coefficient. Le coefficient utilisé 2^{bf} est une puissance de deux afin de simplifier la division.

Ce qui nous donne l'équation suivante :

$$2^{bf}.K_i\hat{\mathbb{I}}_{harmo} = K_i.\{Nb_1.\mathbb{I}_t(k-1) + Nb_2.\mathbb{I}_t(k-2) - Db_1.\hat{\mathbb{I}}_{harmo}(k-1) - Db_2.\hat{\mathbb{I}}_{harmo}(k-2)\} \tag{III.6.22}$$

avec $Nb_i = 2^{bf}.n_i$ et $Db_i = 2^{bf}.d_i$

Le calcul de cette équation est représenté schématiquement sur la figure III.6.19.

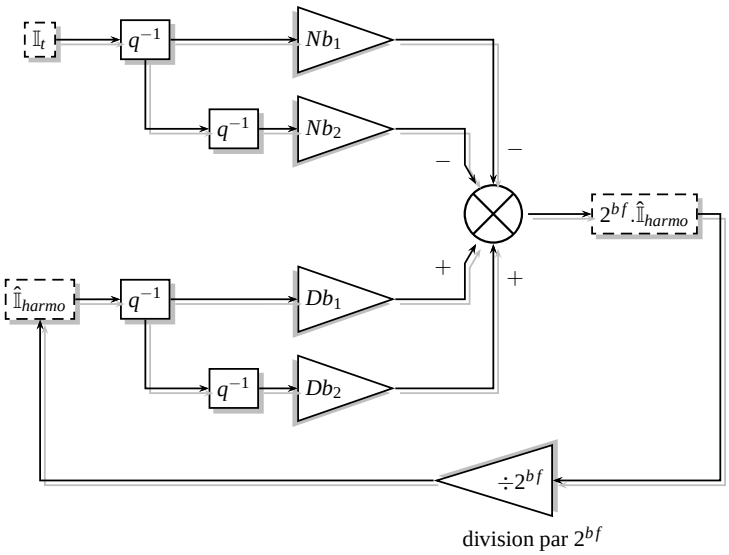


Figure III.6.19: *Modèle numérique du filtre.*

C'est cette structure que nous voulons implanter en parallèle avec l'émulateur.

6.6 Premières évaluations en co-simulation

La figure III.6.20 représente l'organisation des spécifications VHDL de l'observateur. Elles sont divisées en deux entités : la première correspond à l'émulateur modifié pour

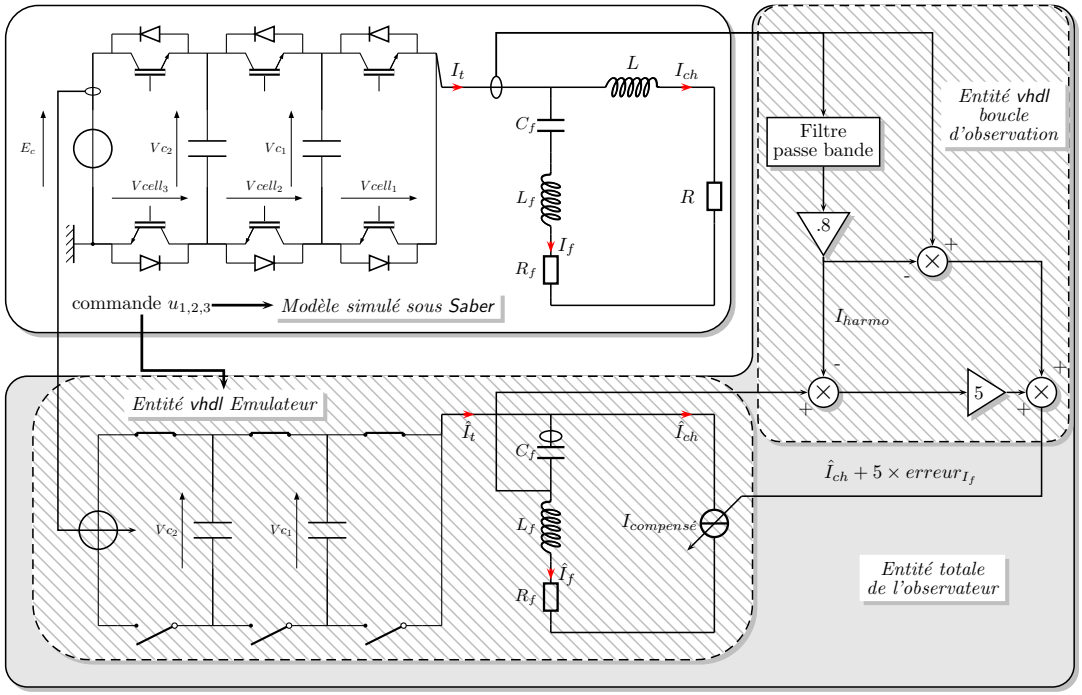


Figure III.6.20: *Organisation des entités et connexion avec Saber*

recevoir le courant de charge additionné de l'erreur d'observation $K.(\hat{I}f - \hat{I}_{harmo})$; la seconde entité décrit le fonctionnement du rebouclage incluant le filtre calculé précédemment. L'entité "vue" sous saber correspond à l'association des ces deux sous-entités.

Dans la gamme des tests réalisables dans l'environnement de co-simulation, on a choisit de faire varier la période d'échantillonnage de l'observateur, afin de pouvoir dimensionner la fréquence d'acquisition du courant nécessaire à la convergence de l'observateur.

6.6.1 Échantillonnage à $480\eta s$

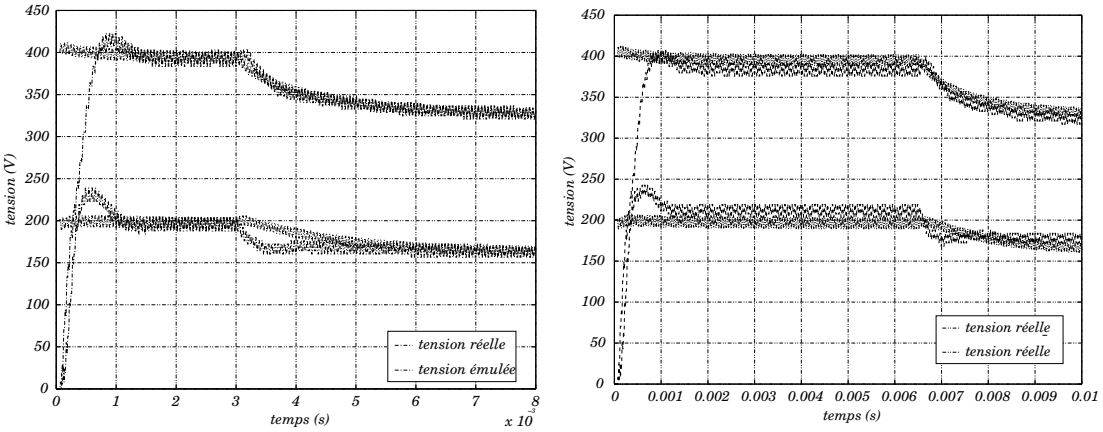
On se place dans un premier temps à une période ou l'émulateur fonctionne sans problème particulier c'est à dire à $T_e = 480\eta s$. Les résultats présentés à la figure III.6.21-a- sont à comparer avec les tracés obtenus sur la figure III.6.17 de la page 138.

On peut remarquer que les dynamiques de convergence sont à peu de choses près identiques. La seule différence se situe au niveau du comportement des tensions émulées face à des variations de la tension d'entrée. Cette simulation permet toutefois de valider l'implantation de l'observateur dans l'environnement de co-simulation.

6.6.2 Échantillonnage à $960\eta s$

Il est intéressant d'apprécier l'influence de la fréquence d'échantillonnage de l'observateur sur l'estimation des tensions flottantes du convertisseur. Les résultats de simulation ci-dessous (figure III.6.21-b-) présentent l'observateur cadencé au double de la période d'échantillonnage testée précédemment, c'est à dire $Te = 960\eta s$

On constate que la dynamique de l'observateur est bonne, même si les tensions flottantes émulées n'arrivent pas à se stabiliser exactement aux valeurs optimales des tensions flottantes réelles. Ceci est vraisemblablement dû aux erreurs issues de l'interpolation entre deux acquisitions de l'observateur. L'examen des figures III.6.22-a- et III.6.22-b- permet en effet de constater que les courants filtre émulés et réels sont strictement semblables et qui plus est en phase. Ce qui signifie que l'estimation du courant

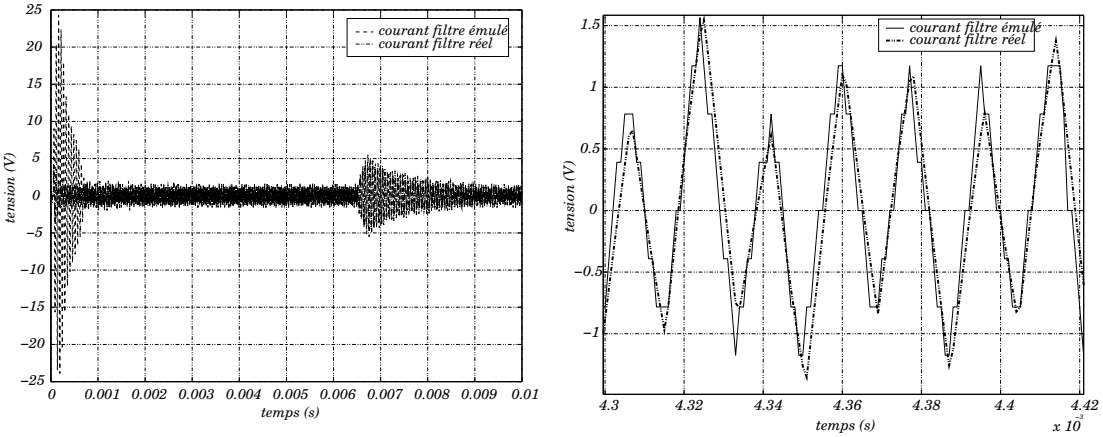


a- Échantillonnage à $480\eta s$

b- Échantillonnage à $960\,\eta s$

Figure III.6.21: *Évolution des tensions flottantes de l'observateur numérique*

filtre à partir du courant total dans la boucle d'observation est correcte et de manière générale que le passage à $960\eta s$ ne perturbe pas le fonctionnement de la boucle d'observation. Une amélioration serait donc à apporter au niveau de la réalisation numérique de l'émulateur pour de telle période d'échantillonnage ou encore d'étudier d'autres boucles d'observations qui puissent compenser ce phénomène.



a- Démarrage et perturbation de la tension d'entrée

b- Zoom en régime équilibré

Figure III.6.22: *Évolution du courant filtre observé lors d'un échantillonnage à* $960\eta s$

6.7 Conclusion

Ce chapitre montre la réalisation d'un émulateur numérique temps réel du convertisseur multicellulaire série à 3 cellules de commutation. Sa conception s'est vue grandement accélérée par l'utilisation de la co-simulation notamment en développant et en validant au plutôt son code VHDL. Celui-ci a d'ailleurs été implanté sur un composant Altera et a donc été pleinement validé. Dans la deuxième partie, des structures de rebouclages sont étudiées pour utiliser cet émulateur en observation. La structure qui semble la plus performante à l'heure actuelle a été numérisée et validée en co-simulation pour

de faibles période d'échantillonnages.

Nous pouvons envisager deux directions pour la suite de ce travail, la première serait d'utiliser la co-simulation afin d'optimiser la résolution des grandeurs et des coefficients en vue de diminuer la place prise par un tel observateur dans un CLP. Il faudrait aussi réaliser des tests expérimentaux permettant de confirmer le fonctionnement d'un tel observateur même dans un environnement bruité.

La seconde orientation possible, serait d'étudier d'autres boucles d'observations basée sur une extraction fréquentielle, utilisant les séries de Fourier, pour l'harmonique du courant de charge à la fréquence de découpage.

Conclusion Générale

Le travail présenté dans ce mémoire constitue une première étude sur l'intérêt et l'adaptation des outils du co-design pour l'intégration numérique des lois de commandes dédiées aux systèmes électriques. Cette première étude s'est portée sur un outil particulier qu'est la co-simulation.

La première partie du mémoire présente la problématique en décrivant les différentes fonctionnalités que peut être amené à pourvoir un dispositif de commande, que ce soit dans le domaine de l'observation, de la commande ou du diagnostic. On montre ainsi la complexité des besoins que doit remplir le dispositif de commande et les contraintes associées comme les volumes de calcul, les ordres de commande d'un convertisseur et les périodes d'échantillonnages faibles, etc.

Les différentes technologies numériques qui peuvent être utilisées pour répondre à ces besoins sont présentées afin de montrer les potentialités existantes notamment dans le domaine des composants logiques programmables. La problématique est associée au fait que les progrès technologiques laissent percevoir de fortes potentialités, difficiles à évaluer a priori et impossible à évaluer de manière expérimental. De plus nous souhaitons proposer des outils qui permettent de s'affranchir des évolutions souvent trop rapide afin d'avoir des outils stables.

Ceci rejoint un constat plus global fait dans le domaine de la conception des systèmes électroniques et notamment dans la conception des systèmes mixtes logiciels et matériels faisant appel à la fois à des parties câblées et programmées. Les études dans le domaine de la conception mixte, connus sous le nom de Co-design, tentent de mettre en place un certain nombre d'outils pour aider le concepteur. Celui-ci cherche, à partir des spécifications et des contraintes fonctionnelles, une architecture mixte la plus appropriée. C'est ce domaine de recherche que nous présentons de manière globale dans le deuxième chapitre. Ce dernier tente notamment de faire une synthèse des différents outils qu'il est possible de mettre en place dans le cycle de conception d'un système mixte. Deux points sont essentiellement développés : les différents moyens de spécifications d'une part et les techniques d'estimation de performances d'autre part. Ce sont, en effet, les deux points clés nécessaires à la mise en place d'une méthodologie de conception. Puisque sans spécifications, il ne peut pas être construit d'outils d'analyse et de transformation aidant le concepteur dans sa tâche ; et la majeur partie de l'analyse se base sur l'estimation des performances.

Parmi les outils d'estimation et de vérification, nous mettons en avant la co-simulation. L'utilisation de ce principe peut en effet répondre à certains besoins propres à l'intégration numérique des lois de commande sur des composants programmés ou câblés. L'ajout de certaines particularités, notamment l'apport de la simulation des procédés à commander, peut permettre de tester des solutions architecturales et d'évaluer ces effets sur la commande du système. La deuxième partie du mémoire cherche donc à mettre en place des outils et techniques nécessaires pour la co-simulation des dispositifs de commande dédiés aux systèmes électriques.

Une des premières questions que se pose généralement le concepteur lors de l'intégra-

tion des lois de commande est de savoir si le microprocesseur choisit sera suffisamment performant.

Nous montrons, par l'intermédiaire d'une application, que la co-simulation, en utilisant des modèles de microprocesseurs adaptés, peut répondre à ce type de question et permettre de récupérer d'autres informations utiles pour la mise en place d'une architecture de commande comme par exemple la taille mémoire à prévoir pour le système informatique. Cependant les simulateurs de microprocesseurs sont pour la plupart, mis en place par les constructeurs qui ne laissent pas la possibilité de les intégrer dans un environnement extérieur de co-simulation. La simulation compilée dont nous présentons le principe, pourrait pallier ce problème en permettant d'étudier l'influence des différents paramètres associés à une architecture de microprocesseur. Une telle étude permettrait au concepteur de définir les caractéristiques du microprocesseur à utiliser (architecture vliw, utilisation de mac, intérêt du pipeline,...).

Dans le cadre de la commande des systèmes électrique, nous avons vu que la co-simulation pouvait répondre à des besoins concernant les méthodes d'intégration des CLP. La mise en place d'une interface entre un simulateur VHDL et le logiciel SABER montre qu'il est possible d'évaluer rapidement l'impact certains choix dans la conception de la partie logique de la commande.

Cette interface a été implantée en utilisant les techniques de communication interprocessus du système Unix. La construction de cette interface permet à l'utilisateur d'intégrer rapidement une entité VHDL au sein d'un modèle du système à commander, ce qui permet d'évaluer son fonctionnement et d'étudier l'influence des paramètres architecturaux tout en ayant le dialogue et les échanges entre le système de commande et le système simulé.

La troisième partie utilise l'expérience acquise et l'outil mis en place pour étudier différentes structures numériques pour l'observation et l'émulation du convertisseur multicellulaire et son implantation sur FPGA.

Nous montrons en particulier dans le sixième chapitre, l'intégration numérique d'un émulateur du convertisseur contenant 5 variables d'états. Cette intégration a été réalisée en utilisant la co-simulation. Cette dernière nous à permis de valider le développement mais aussi de détecter rapidement les problèmes intervenant lors de la numérisation d'un tel émulateur.

Par la suite, la co-simulation nous a permis de réaliser les premiers tests dans l'utilisation de cet émulateur pour l'observation des tensions flottantes.

L'utilisation de la co-simulation dans l'intégration des lois de commande est devenue aujourd'hui incontournable. Elle permet de tester puis de valider des choix architecturaux, de mettre au point les codes de commande ainsi que les fonctions logiques, de vérifier les compatibilités temporelles et d'estimer les performances du système. Ceci en amont de la phase de réalisation du dispositif de commande. La cosimulation apporte au concepteur une première expertise lui permettant de gagner un temps précieux lors de l'implantation puis de la mise au point des lois de commande.

Nous avons montré qu'il était possible d'intégrer de plus en plus de calcul sur un FPGA de taille moyenne, en particulier lors de l'intégration de l'émulateur temps réel.

L'évolution des technologies va rendre ce type d'action de plus en plus courante, et l'on voit d'ailleurs apparaître des composants logiques programmables intégrant déjà un un cœur de microprocesseur. La rapidité des calculs effectués par les CLP rend ces derniers avantageux, et leur niveau d'intégration ne cesse d'augmenter. Ces deux atouts permettent aujourd'hui l'intégration de lois de commande complexes.

Cette remarque montre une fois de plus l'intérêt de la co-simulation (caractéristique du microprocesseur, partitionnement des tâches, format de codage, etc ...). Cependant, l'utilisation d'un tel outil demande des connaissances approfondies dans le domaine des

techniques numériques. Il faut en effet définir une première architecture, et ceci ne peut se faire sans une expérience et une connaissance importante du domaine.

Il serait alors intéressant de mettre en place des outils permettant d'aider le concepteur à effectuer la synthèse architecturale (présenté au chapitre 2) ou rendant cette synthèse quasi automatique en fonction de contraintes de place et/ou de temps, afin d'optimiser les ressources accessibles d'un CLP.

Cependant, l'ensemble des problèmes posés par la création d'outils d'aide à la conception reste assujetti à la forme des spécifications qu'il peut y avoir en entrée. Cette recherche de spécifications adaptées à la conception des dispositifs de commande pour les systèmes électriques semble être une tâche relativement complexe car elle doit réaliser la synthèse des systèmes existants afin d'en faire ressortir les méthodes les plus adaptées. Une telle recherche pourrait d'ailleurs tenter de relier les spécifications du co-design avec le formalisme basé sur les GIC[7]. afin de garder une continuité entre la synthèse d'une loi de commande et son intégration sur un dispositif de commande.

[7]Graphe Informationnel de Causalité

Bibliographie

[1] « The open systemc initiative » – url, www.systemc.org. 32

[2] A. Aho, R. Sethi et J. Ullman – *Compilateurs : Principes, techniques et outils*, InterEditions, 1991. 37, 68, 157

[3] H. F. et al. – « Commutation dans les convertisseurs statiques », *Techniques de l'Ingénieur* **D3153**, p. 1–18.

[4] — , « Elements constitutifs et synthèse des convertisseurs statiques », *Techniques de l'Ingénieur* **D3152**, p. 1–17.

[5] J. F. Aubry – « Conception des systèmes de commande numériques des convertisseurs électromécaniques : vers une méthodologie intégrant la sûreté de fonctionnement », Thèse d'état, Institut National Polytechnique de Lorraine, Mai 1987. 29

[6] P. Barre, J. Caron, J. Hautier et M. Legrand – *Analyse et modèles*, Editions Ellipse, 1995.

[7] S. Ben Saoud – « Emulateur temps réel d'associations convertisseurs statiques / machines électriques / capteurs, etude, conception et réalisation, nouvelle approche de validation des dispositifs de commande numérique », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1996.

[8] R. Bensaid – « Observateurs des tensions aux bornes des capacités flottantes pour les convertisseurs multicellulaires séries », Thèse de doctorat, Institut National Polytechnique de Toulouse, 2001. 103, 108, 126

[9] G. Blanchet et P. Devriendt – « Processeurs de traitement numérique du signal (dsp) », *Techniques de l'ingénieur* **E 3**, no. 565.

[10] M. Boyer – « étude et réalisation d'un asic dédié à la commande des convertisseurs à résonnance série non réversibles, commande par trajectoire optimale », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1996.

[11] H. Bühler – *Réglages échantillonnés*, vol. 2, Presses polytechniques romandes, 1983.

[12] J. Calvez – « Modélisation et conception des systèmes électroniques », http ://mcse.ireste.fr/.

[13] J. Caron et J. Hautier – *Commande de processus*, Editions Ellipse, 1997.

[14] P. Carrere – « Etude et réalisation des convertisseurs multicellulaires séries à igbt : Equilibrage des tensions flottantes », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1996. 103, 104, 122, 123, 124

[15] W.-T. Chang, A. Kalavade et E. Lee – « Effective heterogenous design and co-simulation », *NATO Advanced Study Institute Workshop on Hardware/Software Codesing* (1995). 49

[16] R. J. Chevance – « Architectures et performances », *Techniques de l'ingénieur* **HA** (1998), no. H1160.

[17] — , « Classification des architectures », *Techniques de l'ingénieur* **HA** (1998), no. H1159. 13

[18] — , « Microprocesseurs introduction », *Techniques de l'ingénieur* **HA** (1998), no. H1158.

[19] F. Clouté, J.-N. Contensou, D. Esteve, P. Pampagnin, P. Pons et Y. Favard – « Hardware/software co-design of an avionics communication protocol interface system : an industrial case study », *CODES*, 1999.

[20] C. COMETE – *Codesign, conception conjointe logiciel-matériel*, Eyrolles, 1998.

[21] A. Dancel – *Le langage C*, Mars 1997, École Nationale de l'Aviation Civile.

[22] P. Davancens et T. A. Meynard – « Etude des convertisseurs multicellulaires parallèles, deuxième partie : Analyse du modèle », *J. Phys III* (1997), p. 161–177. 103

[23] J. Daveau, G. Marchioro, I. T. Ben et A. A. Jerraya – « COSMOS :an SDL based hardware/software codesign environment », *Current Issues in Electronic Modelling, Co-design and co-verification*, vol. 8, Déc 1996. 32

[24] D. D. Gajski, G. Aggarval, E.-S. Chang, R. Dömer, T. Ishii, J. Kleinsmith et J. Zhu – « Methodology for design of embedded systems », Technical Report UCI-ICS-98-07, Dept. if information and Computer Science, university of California, Irvine, Mars 1998. 32

[25] D. D. Gajski, F. Vahid, S. Narayan et G. Jie – *Specification and design of embedded systems*, P T R Prentice Hall, 1994.

[26] G. Gateau – « Contribution à la commande des convertisseurs statiques multicellulaires série, commande non linéaire et commande floue », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1997. 18, 86, 88, 103, 107, 109, 111, 122

[27] J. Gong, D. D. Gajski et S. Narayan – « Software estimation form executable specifications », Technical Report ICS-93-5, Dept. of information and Computer Science, university of California, Irvine, Mars 1993. 39

[28] T. Grandpierre, C. Lavarenne et Y. Sorel – « Modèle d'exécutif distribué temps réel pour SynDEx », Rapport de recherche 3476, INRIA, Août 1998. 33

[29] R. K. Gupta et G. De Micheli – « Hardware-software co-synthesis of digital systems », *IEEE Design and Test of Computers*, vol. 3, Sept 1993, p. 29–41. 36

[30] J. Hautier et J. Faucher – « Le Graphe Informationnel Causal », *Bulletin de l'union des physiciens*, vol. 90, Juin 1996.

[31] D. Heller – « Evaluation des performances dans le cadre du codesign. », Thèse de doctorat, Institut de Recherche et d'Enseignement Supérieur aux Techniques de l'Electronique, Universités de Nantes, 1998.

[32] J. Henkel et R. Ernst – « A hardware/software partitioner using a dynamically determined granularity », *Design Automation Conference* (Anaheim, CA, USA), 1997. 34

[33] K. Hines et G. Borriello – « Pia : A framework for embedded system cosimulation with dynamic communication support », Technical report, Chinook Project, Oct. 1996. 51

[34] A. I. Jazouli, N.-E. Radi et T. Zghal – *Programmation réseau sur tcp/ip, l'interface des sockets*, ENSIMAG, Année Spéciale Informatique, 1993/1994, Responsable : Serge Rouveyrol.

[35] A. Kalavade et E. A. Lee – « A hardware / software codesign methodology for dsp applications », *IEEE Design and Test of Computers*, IEEE, Sept 1993, p. 16–28.

[36] — , « The extended partionning problem : », *International Workshop on Rapid Systems Prototyping*, Juin 1995.

[37] Y. Kim, K. Kim, Y. Shin, T. Ahn, W. Sung, K. Choi et S. Ha – « An integrated hardware-software cosimulation environment for heterogeous sytems prototyping », *Proceedings of 7th IEEE INternational Workshop on Rapid Systems Prototyping*, Juin 1996, p. 66–77.

[38] A. Kountouris et C. Wolinski – « A real-time hw/sw co-design approach based on the signal language and its environment », Rapport de recherche 3046, INRIA, Octobre 1996. 33

[39] M. Lajolo, M. Lazarescu et A. Sangiovanni-Vincentelli – « A compiled-based software estimation scheme for hardware/software co-simulation », *Internation Workshop on Hardware/Software Codesign*, CODES/CASHE, Mar 1999. 50, 72

[40] H. Le-Huy – « Microprocessors and digital ic's for motion control », *IEEE Proceedings* **82** (1994), no. 8.

[41] P. Le Marrec, C. Valderrama, F. Hessel et A. Jerraya – « Hardware, software mechanical cosimulation for automotive applications », *IEEE International Workshop on Rapid Systems Prototyping*, Juin 1998.

[42] P. Le Marrec – « Cosimulation multiniveaux dans un flot de conception multilangage », Thèse de doctorat, Institut National Polytechnique de Grenoble, 2000, Spécialité Microélectronique. 77

[43] « Lisa : Processor description language for architecture exploration, implementation and system level simulation » – Institute for Integrated Signal Processing System, http ://www.iss.rwth-aachen.de/Projekte/Tools/LISA/.

[44] J. Liu, M. Lajolo et A. Sangiovanni-Vincentelli – « Software timing analysis using hw/sw cosimulation and instruction set simulator », *Workshop on Hardware/Software Co-design*, CODES/CASHE, Mars 1998. 50

[45] C. Lochot – « Modélisation et caractérisation des phénomènes couplés dans une chaine de traction ferroviaire asynchrone », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1999. 58

[46] J.-P. Louis et C. Bergmann – « Commande numérique des ensembles convertisseurs-machines », *RGE* **92** (1992), no. 5.

[47] « Lycos project » – Technical University of Lyngby, Denmark, http ://www.it.dtu.dk/ lycos/.

[48] J. Madsen, J. Grode, P. Knudsen, M. Petersen et A. Haxthausen – « LYCOS : then lyngby co-synthesis system », *Design Automation for Embedded Systems*, vol. 2, Depatment of Information Technology, Technical University of Denmark, 1997. 31

[49] G. F. Marchioro – « Découpage tranformationnel pour la conception de systèmes mixtes logiciel/matériel », Thèse de doctorat, Institut National Polytechnique de Grenoble, 1998. 32, 35, 37

[50] T. Meynard et H. Foch – « Multilevel conversion : high voltage choppers and voltage-source inverter », *PESC* (1992), p. 397–405. 85

[51] T. Meynard et H. Foch – « Brevet français N° 91,09582 du 25 juillet 91, dépot international PCT (Europe, Japon, U.S.A, Canada) N° 92,00652 du 8 juillet 92 ».

[52] « Model technology » – Mentor Graphics Company, http ://www.model.com/.

[53] E. Monmasson – « Architecture de dispositifs de commande numérique, application à la variation de vitesse, réalisation à l'aide de prédiffusés reprogrammables », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1993. 21

[54] F. Naçabal – « Outils pour l'exploration d'architectures programmables embarquées dans le cadre d'applications industrielles », Thèse de doctorat, Institut National Polytechnique de Grenoble, 1998.

[55] R. Niemann et P. Marwedel – « An algorithm for hardware / software partitioning using mixed integer linear programming », *Design Automation for Embedded Systems*, 1997. 36

[56] B. Olivier, B. Bolz et T. Elsensohn – *Vous avez dit : IPC sous Unix system V ?*, ENSIMAG, 1991.

[57] A. Österling, T. Benner, R. Ernst, D. Herrmann, T. Scholz et W. Ye – « Cosyma :cosynthesis for embedded architectures », http ://www.ida.ing.tu-bs.de/projects/cosyma/. 36

[58] O. Pasquier et J. Calvez – « An object executable model for simulation of real-time hw/sw systems », *Workshop on Hardware/Software Co-design*, CODES/-CASHE, 1999.

[59] D. Pinon, M. Fadel et T. Meynard – « Commande par mode glissant d'un hacheur à deux cellules : étude de l'installation des cycles limites », *Revue Internationale de Génie Electrique* **1** (1998), no. 3, p. 393–415.

[60] D. Pinon – « Commandes des convertisseurs multicellulaires par mode de glissement », Thèse de doctorat, Institut National Polytechnique de Toulouse, Juin 2000. 107

[61] « A framework for hardware-software co-design of embedded systems » – http ://www-cad.eecs.berkeley.edu/ polis/. 33

[62] L. Prissé – « Etude, conception et mise en œuvre de convertisseurs multicellulaires séries à igbt », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1995. 104

[63] « Ptolemy project » – UC Berkeley, Department of EECS, http ://ptolemy.eecs.berkeley.edu/. 30

[64] X. Roboam et H. Kabbaj – « Micro-projet de commande numérique temps réel, la boucle de vitesse d'une mcc », Note interne, ENSEEIHT-INPT, Janvier 1996. 55

[65] O. Roux – « Langages réactifs synchrones et asynchrones », *Techniques de l'ingénieur* **S8060** (2000), no. S.

[66] J. Rowson – « Hardware/software co-simulation », *Proceedings of 31th ACM/IEEE Design Automation Conference*, Juin 1994, p. 439–440. 49

[67] « Saber mixed-signal/mixed-technology simulator » – Synopsys, http ://www.analogy.com/products/.

[68] J. Staunstrup et W. Wolf (éds.) – *Hardware/software co-design : Principles and practice*, Kluwer Academic, 1997.

[69] K. Suzuku et A. Sangiovanni-Vincentelli – « Efficient software performance estimation methods for hardware/software codesign », *Design Automation Conference*, 1996. 50

[70] O. Tachon – « Commande découplante linéaire des convertisseurs multicellulaires série », Thèse de doctorat, Institut National Polytechnique de Toulouse, 1998. 103, 107

[71] Texas Instrument – *Tms320c3x user's guide*, july 1997. 164

[72] C. A. Valderrama – « Prototype virtuel pour la génération des architectures mixtes logicielles/matérielles », Thèse de doctorat, Institut National Polytechnique de Grenoble, 1998. 77, 81

[73] C. Vieillefond – *Mise en œuvre du 68000*, SYBEX, 1984.

[74] V. P. Vijayaraghavan – « Exploration des liens entre la synthèse de haut niveau (hls) et la synthèse au niveau transferts de registres (rtl) », thèse de doctorat, l'Institut National Polytechnique de Grenoble, Mars 1992. 40

[75] V. Živojnović, S. Pees, C. Schläger, R. Weber et H. Meyr – « Supersim - a new technique for simulation of programmable dsp architectures », *Proceddings of ICSPAT*, 1995, p. 1748–1763. 50, 67

[76] M. Zécri – « étude des potentialités du traitement analogique appliqué au domaine du génie électriuqe », Thèse de doctorat, Institut National Polytechnique de Toulouse, 2000. 126, 132

[77] M. Zécri, H. Schneider et L. Etcheverry – « Analogue real time emulation for power converters », *International Power Electronics and Motion Control Conference*, Août 2000.

Annexes

Annexe A

Simulation Compilée

Durant nos travaux, nous avons tenté de réaliser un petit exemple de simulation compilée. Pour cela il nous a fallut comprendre les principes de base d'un compilateur et prendre en main les outils nécessaires à la création d'un compilateur et notamment la partie frontale (analyseur lexical et syntaxique).

L'exemple que nous avons bâti sur la traduction du langage assembleur des TMS320-C3X n'est pas exhaustif et comprend probablement des erreurs dues à notre inexpérience dans la mise en place d'un compilateur. Cependant, cette annexe est mise à disposition afin de montrer les potentialités des outils permettant la création de traducteurs automatiques et afin d'introduire des notions de bases sur ce sujet pour les lecteurs intéressés.

Les explications fournies dans cette annexe sont trés fortement inspirées du document [2] qui fait référence dans le domaine des compilateurs.

Nous allons spécifier en langage Lex et en langage Yacc une grammaire permettant de retranscrire la vingtaine d'instructions qui constitue le fichier assembleur que nous voulons analyser. Le résultat de cette traduction est une suite d'instructions C que nous pourrons alors exécuter et qui représente les instructions assembleur d'un fichier résultant d'une compilation C pour le TMS320C3X.

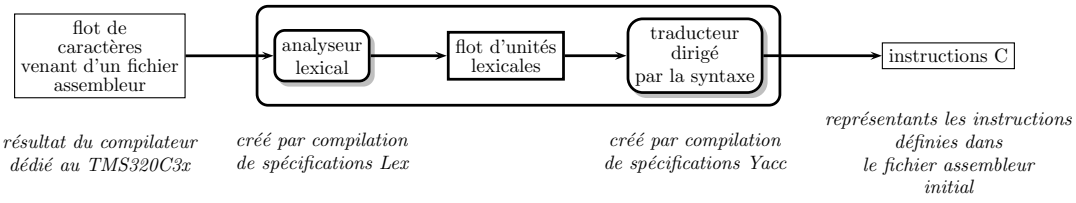


Figure III.A.1: *Structure et utilisation du compilateur généré en utilisant Lex et Yacc.*

Nous allons décrire les différentes étapes nécessaires à la création du compilateur en introduisant quelques notions essentielles à la compréhension de la syntaxe des spécifications Lex et Yacc.

A.1 L'analyse lexical et l'utilisation de Lex

Lex permet de décrire un analyseur lexical à partir de notations spécifiques (le langage Lex), essentiellement basées sur les expressions régulières (que nous présenterons dans la section suivante). On peut associer à la reconnaissance d'une unité lexicale une action particulière qui est décrite par un morceau de code C. Cette action particulière permet notamment de transmettre des informations utiles à l'analyseur syntaxique, informations telles que les lexèmes (unités lexicales) reconnues.

Un programme écrit en langage Lex est décrit en 3 parties :

1 déclarations
%%
2 règles de traduction
%%
3 procédures auxiliaires

1. La section des déclarations comprend des déclarations de variables, des constantes littérales, des définitions régulières.
2. Les règles de traduction sont codées de la manière suivante :

   m_1 { $action_1$ }
   m_2 { $action_2$ }
   .....

   où chaque m_i est une expression régulière et chaque $action_i$ est décrite en C et se réalise à chaque fois que l'unité lexicale m_i est reconnue.
3. La troisième section contient la définition de procédures auxiliaires qui pourraient être utilisées lors des actions $action_i$.

En général l'analyseur lexical ainsi créé est "connecté" à un analyseur syntaxique. L'analyseur lexical renvoie le lexème (unité lexicale) reconnu après avoir effectué l'action sémantique. Il peut aussi passer une valeur associée au lexème par l'intermédiaire d'une variable globale (`yylval`). Lex a été conçu pour produire des analyseurs lexicaux qui puissent être utilisés avec Yacc.

Lex est utilisé de la manière décrite à la figure III.A.2. Dans un premier temps on écrit les spécifications d'un analyseur lexical en langage Lex "`lexical.l`". Ensuite, `lexical.l` est soumis au compilateur Lex pour produire un programme C `lex.yy.c`. Le programme `lex.yy.c` est un analyseur lexical spécifié en C qui doit être compilé pour donner l'exécutable qui transforme un flot d'entrée en une suite d'unités lexicales.

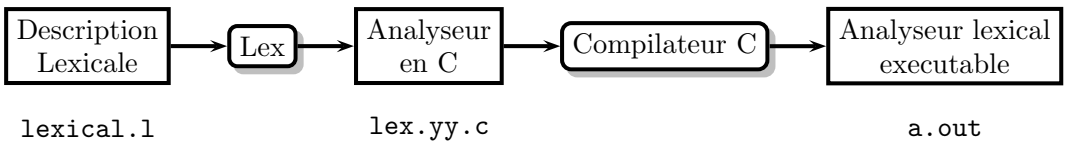


Figure III.A.2: *Création d'un analyseur lexical avec Lex.*

A.1.1 Les expressions régulières

Le compilateur Lex est un outil qui permet de spécifier le fonctionnement d'analyseurs lexicaux essentiellement à partir d'expressions régulières. Afin d'expliquer quelques principes associés aux expressions régulières, il nous faut définir les opérations de bases qui peuvent s'appliquer aux langages.

A.1.1.1 Opérations sur les langages

Dans la définition des langages on utilise l'exponentiation ; ainsi s^3 représente sss, et s^0 représente l'ensemble vide $\in$. soient L et M deux expressions rationnelles, on peut alors appliquer les opérations suivantes pour définir une autre expression rationnelle :

Opération	notations	définition
union de L et M	$L\|M$,$L \cup M$	$L\|M = \{s \mid s \in M \text{ ou } s \in L\}$
concaténation de L et M	LM	$LM = \{st \mid s \in L \text{ et } t \in M\}$
fermeture de Kleene de L	L^*	$L^* = \cup_{i=0}^{\infty} L^i$
fermeture positive de Kleene	L^+	$L^* = \cup_{i=1}^{\infty} L^i$

Tableau III.A.1: *Définitions des opérations sur langages.*

A.1.1.2 Syntaxe des expressions rationnelles

Les expressions régulières utilisent les opérations vue précédemment avec les propriétés suivantes :

1. l'opérateur unaire * a la plus haute priorité et est associatif à gauche,
2. la concaténation possède la deuxième plus haute priorité et est associative à gauche,
3. | possède la plus faible priorité et est associatif à gauche.

En utilisant ces conventions, voici quelques exemples d'expressions régulières :

Exemple A.1.1

1. L'expression régulière $a|b$ dénote l'ensemble {a,b}.
2. L'expression régulière $(a|b)(a|b)$ dénote aa, ab, ba, bb, l'ensemble de toutes les chaînes de a et b de longueur 2. Une autre expression régulière désignant le même ensemble est $aa|ab|ba|bb$.
3. L'expression régulière a^* dénote l'ensemble de toutes les chaînes formées d'un nombre quelconque (voire nul) de a.
4. L'expression régulière $a|a^*b$ dénote l'ensemble contenant la chaîne a et toutes les chaînes constituées d'un nombre quelconque (voire nulle) de a subi d'un b.

À partir des expressions régulières on peut créer des définitions régulières en donnant un nom à des sous expressions régulières, afin de simplifier la lecture.

Exemple A.1.2 On peut par exemple définir un ensemble d'identificateurs correspondant à l'ensemble des lettres et chiffres commençant par une lettre.

lettre → A|B|...|Z|a|b|...|z
chiffre → 0|1|...|9
id → **lettre(lettre|chiffre)***

Certaines expressions apparaissent si fréquemment que des notations abrégées ont été introduites pour des raisons de commodités :

- l'opérateur unaire ? signifie "zéro ou une instance". La notation r? est une abréviation de $r| \in$.
- La notation `[abc]` correspond à une classe de caractères, où `a,b` et `c` sont des symboles d'alphabets, dénote l'expression régulière `a|b|c`. Une classe de caractères comme `[a-z]` dénote l'expression régulière `a|b|...|z`.

A.1.2 Programme Lex de la simulation compilée

Les expressions régulières, dont nous avons expliqué les principes de base, permettent de décrire le fonctionnement de l'analyseur lexical que nous voulons utiliser pour la simulation compilée.

Cet analyseur, qui n'est pas complet, nous permet de faire les premiers essais sur la simulation compilée.

code A.1: *Fichier lex :* `lexical.l`

```
/*
* déclaration des procédures définies
* dans la troisième partie
*/
%{
void testcond();
%}
/* définitions des expressions regulières */

/* définition des délimiteurs entre les lexèmes */
delim    [ \t]
bl       {delim}+

/* définition des differents type de constantes */
chiffre      [0-9]
decimal      (-)?{chiffre}+
binaire      [01]*[bB]
octal        [0-7]*[qQ]
hexa         [0-9a-fA-F]*[hH]
flottant     ([+\-]?{chiffre}+)?\.({chiffre}+)?([Ee][+\-]?{chiffre}+)?
nombre       {decimal}|{octal}|{binaire}|{hexa}|{flottant}


lettre       [a-zA-Z"_"]
identif      {lettre}({lettre}|{chiffre})*


/* définition des différents type d'adressage */
registre     R[0-7]
auxi         AR[0-7]
pile         SP
cond         [\tPZNGL][EZTN]?
adr_direct       "@"
adr_indirect     "*"
directive    "."*
label        {lettre}({lettre}|{chiffre})*":"

/* fin de la partie déclaration */
/* Les REGLES de TRADUCTION */
/*
*  on rajoute deux mot cles : "debut" et "fin_asm" dans le code
    assembleur
*  pour délimiter la portion de code que l'on veut retranscrire
*/
%%
debut    {
             yylval=MALLOC(char,yyleng);
         }
fin_asm  {
             free(yylval);
         }
{bl}         {   }
{decimal}    {
                 yylval=MALLOC( char, yyleng);
```

```
                    strcpy(yylval,yytext);return(DECIMAL);
55              }
    {binaire}   {
                    yylval=MALLOC( char, yyleng);
                    strcpy(yylval,yytext); return(BINAIRE);
                }
60  {octal} {
                    yylval=MALLOC( char, yyleng);
                    strcpy(yylval,yytext);
                    return(OCTAL);
            }
65  {hexa}  {
                yylval=MALLOC( char, yyleng);
                strcpy(yylval,yytext);
                return(HEXA);
            }
70  {flottant}  {
                    yylval=MALLOC( char, yyleng);
                    strcpy(yylval,yytext);
                    return(FLOTTANT);
                }
75  {registre}  {
                    yylval=MALLOC( char, yyleng);
                    strcpy(yylval,yytext);
                    return(REGISTER);
                }
80  {auxi}  {
                yylval=MALLOC( char, yyleng);
                strcpy(yylval,yytext);
                return(AUXI);
            }
85  {pile}  {
                yylval=MALLOC( char, yyleng);
                strcpy(yylval,yytext);
                return(POINTEUR);
            }
90  {label} {
                yylval=MALLOC( char, yyleng);
                strcpy(yylval,yytext);
                return(LABEL);
            }
95
    {adr_direct}    {return(ADR_DIRECT);}
    {adr_indirect}    {return(ADR_INDIRECT);}
    {directive}     {return(DIRECTIVE);}

100 ADDI        { return(ADDI);}
    LDI         { return(LDI);}
    STF         { return(STF);}
    LDF         { return(LDF);}
    PUSHF       { return(PUSHF);}
105 PUSH        { return(PUSH);}
    CALL        { return(CALL);}
    SUBF        { return(SUBF);}
    SUBRF       { return(SUBRF);}
    MPYF        { return(MPYF);}
110 ADDF        { return(ADDF);}
    STI         { return(STI);}
    CMPF        { return(CMPF);}
    CMPI        { return(CMPI);}
    RND         { return(RND);}
115 B/{cond}    { return(BRANCH);}
    {cond}      {
                    yylval=MALLOC( char, yyleng);
                    testcond(yytext,yyleng,yylval);
```

```
                    return(COND);
120             }
    \n          { return (RETOUR);}
    {identif}   {
                    yylval=MALLOC( char, yyleng);
                    strcpy(yylval,yytext);
125                 return(TEXTE);
                }
    .           { return yytext[0];}
    %%
    /* le point indique n'importe quel caractère */
130 /* déclaration de fonctions auxiliaires utilisées dans les actions */
    void testcond( condition, longueur, result)
    char condition[];
    int longueur;
    char *result;
135 {
    int flag=0;
    if((strncmp("Z",condition,1)==0) && (flag==0))
        {
        printf("/******Z******/\n");
140     strcpy(result,"5");flag=1;
        }
    if((strncmp("LE",condition,1)==0) && (flag==0))
        {
        printf("/******LE******/\n");
145     strcpy(result,"8");flag=1;
        }
    if((strncmp("GT",condition,2)==0) && (flag==0))
        {
        printf("/******GT******/\n");
150     strcpy(result,"9");flag=1;
        }
    if((strncmp("NZ",condition,2)==0) && (flag==0))
        {
        printf("/******NZ******/\n");
155     strcpy(result,"6");flag=1;
        }
    if((strncmp("LT",condition,2)==0) && (flag==0))
        {
        printf("/******LT******/\n");
160     strcpy(result,"7");flag=1;
        }
    }
```

Cet analyseur lexical seul n'a pas grande utilité. Il a été conçu afin d'être relié à un analyseur syntaxique qui va interpréter l'association des unités lexicales reconnues.

A.2 l'analyseur syntaxique et utilisation de Yacc

La construction d'un analyseur syntaxique avec Yacc se fait en décrivant les spécifications sous le format d'une grammaire non contextuelle.

Une grammaire non contextuelle comprend quatre composants :

1. un ensemble d'unités lexicales appelées symboles terminaux (venant du programme défini précédemment dans notre cas) ;
2. un ensemble de non-terminaux ;
3. un ensemble de production où chaque production est constituée d'un non-terminal, appelé partie gauche de la production, d'une flèche, et d'une suite d'unités lexicales et de non-terminaux appelés partie droite de la production.
4. la désignation d'un des non-terminaux en tant que symbole de départ ou axiome.

Exemple A.2.1 les expressions composées de chiffres et de signes plus et moins, par exemple `9-5+2, 3-1` peuvent être spécifiées par la grammaire non contextuelle suivante :

$$\begin{aligned} \textit{liste} &\rightarrow \textit{liste}\texttt{+}\textit{chiffre} \\ \textit{liste} &\rightarrow \textit{liste}\texttt{-}\textit{chiffre} \\ \textit{liste} &\rightarrow \textit{chiffre} \\ \textit{chiffre} &\rightarrow \texttt{0|1|2|3|4|5|6|7|8|9} \end{aligned} \tag{III.A.1}$$

Les unités lexicales dans cet exemple sont les symboles : `+ - 0 1 2 3 4 5 6 7 8 9`, les non-terminaux sont *liste* et *chiffre* et *liste* correspond à l'axiome de départ, car ses productions sont données en tête.

Sous Yacc, le symbole " `:` " est équivalent à la flèche "→".

A.2.1 Construction d'un analyseur syntaxique avec Yacc

Le squelette d'un programme écrit dans le langage Yacc est de la forme :

1.déclarations
%%
2.règles de traduction %%
3.routines en C optionnelles

1. Dans la section déclaration, on déclare les fonctions, procédures, et variables utilisées. On déclare aussi les unités lexicales de la grammaire que l'analyseur syntaxique peut reconnaître.

   Exemple A.2.2 : `%token REGISTER`
   "`REGISTER`" est déclaré comme une unité lexicale.

2. Dans la deuxième section, une règle grammaticale est décrite par une production suivie d'une action sémantique :

```
<partie gauche>  :  <suite d'unités lexicales ou non terminal 1>
                        {action sémantique 1}
                 |  <suite d'unités lexicales ou non terminal 2>
                        {action sémantique 2}
                 ;
```

   Exemple A.2.3

```
INSTR:      ADDI src',dest
     {
        printf("%s = %s + %s ;Tecalc=Tecalc+1;",$4,$4,$2) ;
     }
```

   L'action sémantique est décrite en C, et le symbole `$i` référence la valeur du i^{eme} symbole de la grammaire en partie droite. Dans l'exemple, `$4` représente la valeur du lexème `dest`.

3. La troisième section, comme dans le cas de Lex, contient la définition de procédures auxiliaires qui pourraient être utilisées lors des actions sémantiques. De plus, un analyseur lexical `yylex()` doit être fourni dans cette partie. Lorsque l'on utilise Lex pour l'analyse lexical, il suffit d'ajouter dans cette partie l'instruction suivante :

```
#include "lex.yy.c"
```

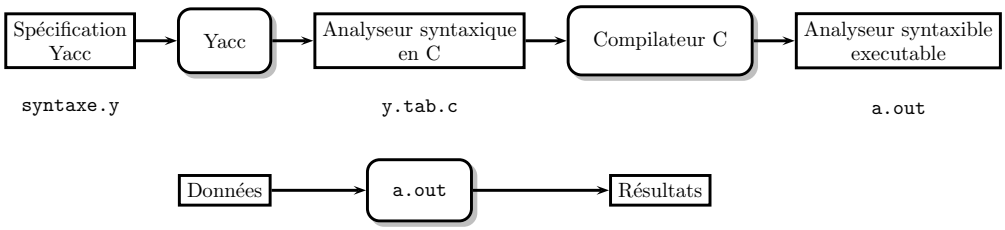


Figure III.A.3: *Création, avec Yacc, d'un traducteur de Données vers Résultats.*

Finalement, la figure III.A.3 rappel les étapes nécessaires à la création de l'analyseur syntaxique lorsque l'on utilise des spécifications décrites en langage Yacc.

Pour créer notre compilateur, il faut alors effectuer les étapes suivantes :

1. "pré"compiler l'analyseur lexical :

   `lex lexical.l`

2. "pré"compiler l'analyseur syntaxique :

   `yacc syntaxe.y`

3. compiler le traducteur utilisant les fichiers `lex.yy.c` et `y.tab.C`

   `cc y.tab.c -ly -ll`

A.2.2 Application à la simulation d'un DSP TMS320C3x

Dans notre cas, la grande difficulté dans l'utilisation d'une telle technique correspond au fait qu'il faut retranscrire une instruction assembleur avec un langage de plus haut niveau comme le C. Nous allons d'ailleurs exposer quelques un de ces problèmes.

Gestion du temps de calcul :

Après chaque instruction C correspondant à une instruction assembleur, une variable appelée `Tps_calc` est incrémentée d'un nombre de cycles dépendant de l'instruction représentée.

Remarque A.2.1. *Cette donnée est accessible dans le manuel utilisateur du microprocesseur C3x [71].*

Le temps de cycles de certaines instructions peut varier selon les instructions qui précèdent. Ce phénomène est essentiellement dû aux conflits de pipeline ou bien aux accès mémoire, mais ceux-ci peuvent tout à fait se prévoir a priori et sont donc pris en compte durant l'analyse du texte assembleur.

Gestion de l'appel de sous-routine :

Un appel de sous-routine en assembleur est retranscrit en C grâce à une étiquette (label) et à l'instruction "goto" mais il faut aussi pouvoir revenir au programme principal une fois la sous-routine terminée (fonction qui est intrinsèque en assembleur). Ainsi, nous insérons un label du type `FIN_<nom_de_la_routine>` après l'appel de la sous-routine et à la fin de celle-ci on insère l'instruction "`goto FIN_<nom_de_la_routine>`". Une fois la sous-routine exécutée, on revient donc au programme principal. Le fichier assembleur que nous utilisons appelle parfois des routines décrites dans une bibliothèque de fonctions. Dans ce cas, la fonction n'est pas directement décrite en assembleur mais est déjà compilée et correspond à un fichier objet. Pour résoudre ce problème, deux solutions sont possibles : soit on retranscrit la fonctionnalité directement en C, soit on recompile la fonction, si on a le fichier source, et on obtient un fichier assembleur que l'on peut retranscrire en C comme pour la routine principale.

Précision :

Le code généré en C représente des instructions assembleur et dans le cas du TMS320C3X, le problème de précision ne se pose pas car il s'agit d'un microprocesseur utilisant un format de 32 bits. Le format "float" du code C utilisé sur une station est donc le même que celui utilisé par le DSP. Notons que c'est un problème qui interviendra lors de la simulation de microprocesseurs à virgule fixe. Dans ce cas, il faudra alors créer les fonctions arithmétiques et logiques prenant en compte le format des registres.

code A.2: *fichier yacc : **syntaxe.y***

```
%{
#define YYSTYPE char *
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MALLOC(t,n) (t *) malloc(sizeof(t)*n)
#define REALLOC(t,pt,n) (t *) realloc(pt,sizeof(t)*n)
char *texte;
int Tecalc=0,v;
int test;
%}
/* définition des lexèmes */
%token   DECIMAL BINAIRE OCTAL HEXA FLOTTANT TEXTE
%token   RETOUR FIN
%token   REGISTER ADR_DIRECT ADR_INDIRECT DIRECTIVE POINTEUR AUXI
%token   LABEL
%token   ADDI LDI STF LDF PUSHF PUSH CALL SUBF MPYF ADDF STI BRANCH
    COND CMPF CMPI RND SUBRF
/* défintion de l'axiome */
%start ligne
/* declaration des regles de traductions */
%%
ligne   :   ligne expr RETOUR
        |   ligne RETOUR
        |   /*vide*/
        |   error '\n'          {yyerror("probleme");yyerrok;}
        ;
expr    :   LABEL               {printf("%s\n",$1);}
        |   INSTR               {printf("\n");}
        |   DIRECTIVE           {printf("\n");}
        ;
INSTR   :   ADDI src','dest
     {
        printf("%s = %s + %s;Tecalc=Tecalc+1;",$4,$4,$2) ;
     }
        |   LDI src','dest
            {
                printf("%s = (int) %s;Tecalc=Tecalc+1;",$4,$2);
            }
        |   STF src','dest
            {
                printf("%s = %s;Tecalc=Tecalc+1;",$4,$2);
            }
        |   LDF src','dest
            {
                printf("%s = %s;Tecalc=Tecalc+1;",$4,$2);
            }
        |   PUSHF src
            {
                printf("SP=SP+1 ; %s=%s>>4 ; %s=%s<<4 ; *SP=(double)%s;
                    Tecalc=Tecalc+1;",$2,$2,$2,$2,$2);
            }
```

```
        |   PUSH src
            {
                printf("SP=SP+1␣;␣*SP=(int)%s␣;Tecalc=Tecalc+1;",$2);
            }
        |   SUBF src','src','dest
            {
                printf("%s␣=␣%s␣-␣%s␣;Tecalc=Tecalc+1;",$6,$4,$2);}

        |   SUBF src','dest
            {
                printf("%s␣=␣%s␣-␣%s␣;Tecalc=Tecalc+1;",$4,$4,$2);
            }
        |   SUBRF src','dest
            {
                printf("%s␣=␣%s␣-␣%s␣;Tecalc=Tecalc+1;",$4,$2,$4);
            }
        |   MPYF src','src','dest
            {
                printf("%s=␣%s␣*␣%s␣;Tecalc=Tecalc+1;",$6,$4,$2);}

        |   MPYF src','dest
            {
                printf("%s␣=␣%s␣*␣%s␣;Tecalc=Tecalc+1;",$4,$4,$2);}

        |   ADDF src','src','dest
            {
                printf("%s=␣%s␣+␣%s␣;Tecalc=Tecalc+1;",$6,$4,$2);
            }
        |   ADDF src','dest
            {
                printf("%s␣=␣%s␣+␣%s␣;Tecalc=Tecalc+1;",$4,$4,$2);
            }
        |   STI src','dest
            {
                printf("%s␣=␣%s␣;Tecalc=Tecalc+1;",$4,$2);
            }
        |   CMPF src','dest
            {
                printf("if␣((%s-%s)==0)␣Z=1;␣else\n\t{\n\tZ=0;\n\tif␣((%s
                    -%s)<0)␣N=1;␣else␣\
␣␣␣␣␣␣␣␣␣␣␣␣N=0;\n\t}\nTecalc=Tecalc+1;",$4,$2,$4,$2);
            }
        |   CMPI src','dest
            {
                printf("if␣((%s-%s)==0)␣Z=1;␣else\n\t{\n\t␣Z=0;\n\tif␣((%
                    s-%s)<0)␣N=1;␣else␣\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣N=0;\n\t}\nTecalc=Tecalc+1;",$4,$2,$4,$2);
            }
        |   RND src
            {
                printf("/*arrondi␣de␣%s*/␣;␣Tecalc␣=␣Tecalc␣+␣1;",$2);
            }
        |   RND src','dest
            {
                printf("%s=%s;␣Tecalc␣=␣Tecalc␣+␣1;␣/*arrondi*/",$4,$2);
            }
        |   BRANCH src
            {
                printf("␣Tecalc␣=␣Tecalc␣+␣4;goto␣%s;",$2);
            }
        |   BRANCH COND src
            {
                test=atoi($2);
                if( strncmp($3,"R",1) == 0 ) {}
                else
```

```
                {
115             switch(test){
                      case 5:printf("if␣(Z==1){\n\tTecalc=Tecalc+4;goto␣%
                          s;\n\t}\n",$3);break;

                      case 6:printf("if␣(Z==0){\n\tTecalc=Tecalc+4;goto␣%
                          s;\n\t}\n",$3);break;

120                   case 7:printf("if␣(N==1){\n\tTecalc=Tecalc+4;goto␣%
                          s;\n\t}\n",$3);break;

                      case 8:printf("if␣((N==1)||(Z==1)){\n\tTecalc=
                          Tecalc+4;goto␣%s;\n\t}\n",$3);break;

                      case 9:printf("if␣((Z==0)␣&&␣(N==0)){\n\tTecalc=
                          Tecalc+4;goto␣%s;\n\t}\n",$3);break;
125
                      case 21:printf("Tecalc␣=␣Tecalc␣+␣4;goto␣%s;␣",$3);
                          break;
                      }
                }
            }
130     |   CALL src
            {
                printf("Tecalc=Tecalc+␣4␣+␣Te_%s;flag=1;goto␣%s␣;\nFIN_%s
                    :\nflag=0;",$2,$2,$2);
            }
        ;
135 src :   REGISTER
        |   ADR_DIRECT terme'+'terme
            {
                $$=MALLOC(char,strlen($2)+strlen($4)+4);sprintf($$,"*(%s
                    +%s)",$2,$4);
                if (strcmp($2,"CONST")==0)
140                 {
                    printf("++Tecalc;");
                    }
                else if(strncmp($2,"STATIC",4)==0)
                    {
145                 printf("++Tecalc;");
                    }
            }
        |   ADR_DIRECT terme
            {
150             $$=MALLOC(char,strlen($2)+3);sprintf($$,"*(%s)",$2);
                if (strcmp($2,"CONST")==0)
                    {
                    printf("++Tecalc;");
                    }
155             else if(strncmp($2,"STATIC",4)==0)
                    {
                    printf("++Tecalc;");
                    }
            }
160     |   ADR_INDIRECT'+'terme'('terme')'
            {
                $$=MALLOC(char,strlen($3)+strlen($5)+4);sprintf($$,"*(%s
                    +%s)",$3,$5);
            }
        |   ADR_INDIRECT'-'terme'('terme')'
165         {
                $$=MALLOC(char,strlen($3)+strlen($5)+4);sprintf($$,"*(%s
                    -%s)",$3,$5);
            }
        |   ADR_INDIRECT'+'terme
```

```
            {
170             $$=MALLOC(char,strlen($3)+5);sprintf($$,"*(%s+1)",$3);
            }
        |   ADR_INDIRECT terme
            {
                $$=MALLOC(char,strlen($2)+3);sprintf($$,"*(%s)",$2);
175         }
        |   terme
        ;
    dest:   REGISTER
        |   ADR_DIRECT terme'+'terme
180         {
                $$=MALLOC(char,strlen($2)+strlen($4)+4);sprintf($$,"*(%s
                    +%s)",$2,$4);
                if (strcmp($2,"CONST")==0)
                    {
                    printf("++Tecalc;");
185                 }
                else if(strncmp($2,"STATIC",4)==0)
                    {
                    printf("++Tecalc;");
                    }
190         }
        |   ADR_DIRECT terme
            {
                $$=MALLOC(char,strlen($2)+3);sprintf($$,"*(%s)",$2);
                if (strcmp($2,"CONST")==0)
195                 {
                    printf("++Tecalc;");
                    }
                else if(strncmp($2,"STATIC",4)==0)
                    {
200                 printf("++Tecalc;");
                    }
            }
        |   ADR_INDIRECT'+'terme'('terme')'
            {
205             $$=MALLOC(char,strlen($3)+strlen($5)+4);sprintf($$,"*(%s
                    +%s)",$3,$5);
            }
        |   ADR_INDIRECT'-'terme'('terme')'
            {
                $$=MALLOC(char,strlen($3)+strlen($5)+4);sprintf($$,"*(%s
                    -%s)",$3,$5);
210         }
        |   ADR_INDIRECT'+'terme
            {
                $$=MALLOC(char,strlen($2)+5);sprintf($$,"*(%s+1)",$3);
            }
215     |   ADR_INDIRECT terme
            {
                $$=MALLOC(char,strlen($2)+3);sprintf($$,"*(%s)",$2);
            }
        |   terme
220     ;
    terme   :   TEXTE       {}
        |   DECIMAL     {}
        |   BINAIRE     {}
        |   OCTAL       {}
225     |   HEXA        {}
        |   FLOTTANT    {}
        |   AUXI        {}
        |   POINTEUR    {}
        ;
230 %%
```

```
#include "lex.yy.c"
```

Annexe B

Synthèse des régulations de vitesse et de courant pour la machine à courant continu

B.1 Régulation de courant

le calcul de la régulation de courant s'effectue en utilisant une méthode de compensation des pôles.

Étant donné que la fonction de transfert du système électrique (sans tenir compte de la *f.e.m*) est :

$$H(p) = \frac{I_m}{V_m}(p) = \frac{1}{R} \cdot \frac{1}{\frac{L}{R}.p + 1} = \frac{K_e}{1 + \tau_e.p} \tag{III.B.1}$$

avec $K_e = \frac{1}{R}$ et $\tau_e = \frac{L}{R}$ et que le régulateur proportionnel intégral s'écrit de la manière suivante :

$$Reg(p) = \frac{V_{ref}}{\varepsilon}(p) = K_p. + \frac{1}{T_i \cdot p} = \frac{K_p.p + 1}{T_i.p} \tag{III.B.2}$$

En boucle ouverte on le système suivant :

$$G_{bo}(p) = \frac{I}{\varepsilon}(p) = \frac{K_p.p + 1}{T_i.p} \cdot \frac{K_e}{1 + \tau_e.p} \tag{III.B.3}$$

Après compensation du pôle électrique $K_p = \tau_e$, le système en boucle ouverte devient :

$$G_{bo}(p) = \frac{K_e}{T_i.p} \tag{III.B.4}$$

Soit en boucle fermée :

$$G_{bf}(p) = \frac{1}{1 + \frac{T_i}{K_e}.p} \tag{III.B.5}$$

Les paramètres du régulateurs sont donc :

$$K_p = \frac{L}{R} \tag{III.B.6}$$

$$T_i = \frac{\tau_{voulu}}{R} \tag{III.B.7}$$

B.2 Régulation de vitesse

Le cahier des charges qui a été fixé pour la régulation de vitesse est le suivant :
- erreur statique nulle,
- Dépassement de l'ordre 20% sur une réponse indicielle.

Pour la synthèse du régulateur de vitesse, étant donné que l'hypothèse de la séparation des modes a été posée, on considère que Iref=Im (Seule la fonction de transfert de la partie mécanique est considérée). Le calcul des coefficients du régulateur se fait en utilisant la théorie des systèmes continus.

Le calcul des coefficient du régulateur se fait par la méthode de placement de pôles.

Étant donné le cahier des charges, le dénominateur de la fonction de transfert en boucle fermée doit être de la forme :

$$D(p) = p^2 + 2.\xi.\omega_n.p + \omega_n^2 \tag{III.B.8}$$

Et le Dépassement peut s'exprimer de la manière suivante :

$$D = e^{-(\frac{\pi.\xi}{\sqrt{1-\xi^2}})} \tag{III.B.9}$$

On en déduit $\xi = 0.45$ et $\omega_n = 2.\pi\ rad.s^{-1}$

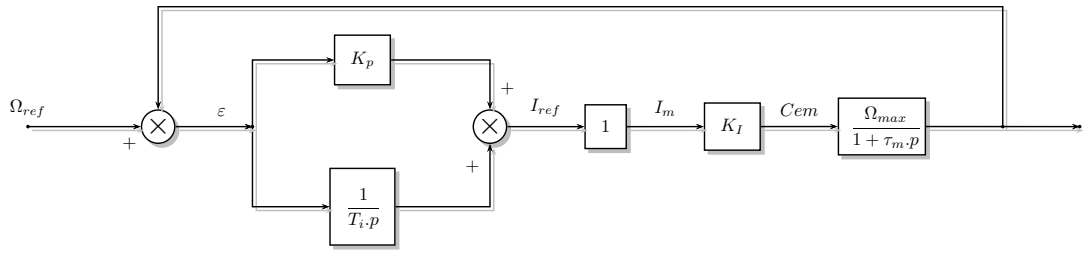


Figure III.B.1: *Structure d'un régulateur Proportionnel Intégrale linéaire (sans saturation)*

La structure du régulateur utilisé est un PI (cf. fig. III.B.1)(proportionnel intégral). la fonction de transfert en boucle fermée est alors :

$$T_{bf}(p) = \frac{(p.K_p + \frac{1}{T_i}).\frac{K_I\Omega_{max}}{\tau_m}}{p^2 + p.\frac{1+K_p.K_I.\Omega_{max}}{\tau_m} + \frac{K_I\Omega_{max}}{T_i.\tau_m}}$$

Par identification, on obtient :

$T_i = \frac{K_I.\Omega_{max}}{\tau_m.\omega_n^2}$

$K_p = \frac{2.\xi}{T_i.\omega_n} - \frac{1}{K_I\Omega_{max}}$

On peut noter qu'une telle structure apporte un zero supplémentaire dans la fonction de transfert en boucle fermée. Mais celui-ci a une valeur telle qu'il ne perturbe pas la régulation.

Annexe C

Spécifications VHDL de l'observateur

spécifications VHDL de l'émulateur adaptés pour l'observation avec $T_e = 480\eta s$

```
--- librairies utilisees ----------------------------------------
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
--- entite emulateur ---------------------------------------------
ENTITY emula_rl_obs IS
 PORT (
        clk : IN std_logic;
        P1  : IN std_logic;
        P2  : IN std_logic;
        P3  : IN std_logic;
        E   : IN integer;
        NIL_integ8b : IN integer;

  NVc1 : OUT integer;
  NVc2 : OUT integer;
  NIl :  OUT integer;
  NIf :  OUT integer;
  NVcf : OUT integer;
  Vdec : OUT integer
    );
END EMULA_RL_obs;

ARCHITECTURE arch_emula_obs OF emula_rl_obs IS
constant coeff_11: signed (11 downto 0):=   "000011000101";
constant coeff_12 : signed (11 downto 0):=  "000011000101";
constant coeff_13 : signed (11 downto 0):=  "000011000101";

constant coeff_21 : signed (11 downto 0):=  "000000000101";
constant coeff_24 : signed (11 downto 0):=  "000000000101";

constant coeff_31 : signed (11 downto 0):=  "000000000101";
constant coeff_34 : signed (11 downto 0):=  "000000000101";

constant coeff_42 : signed (11 downto 0):=  "000000101100";
constant coeff_43 : signed (11 downto 0):=  "000000101100";
constant coeff_44 : signed (11 downto 0):=  "000000110011";
constant coeff_45 : signed (11 downto 0):=  "000000101011";

constant coeff_54 : signed (11 downto 0):=  "010110011101";
constant coeff_55 : signed (11 downto 0):=  "000000001000";


constant coeffB_11: signed (11 downto 0):=  "000011000101";
constant coeffB_41: signed (11 downto 0):=  "000000101100";
constant coeffB_51: signed (11 downto 0):=  "000000000000";
```

```
    --SIGNAL NIL_integ : signed (21 downto 0):= (others => '0');
50  SIGNAL NIf_integ : signed (21 downto 0):= (others => '0');
    SIGNAL NVcf_integ : signed (21 downto 0):= (others => '0');
    SIGNAL NdVcf_integ : signed (21 downto 0):= (others => '0');
    SIGNAL NVcs_integ : signed (21 downto 0):=(others => '0');
    SIGNAL NVc2_integ : signed (21 downto 0):=(others => '0');
55  SIGNAL NVc1_integ : signed (21 downto 0):=(others => '0');

    --SIGNAL NIL_integ8b : signed (9 downto 0);
    SIGNAL NIf_integ8b : signed (9 downto 0);
    SIGNAL NVcf_integ8b : signed (9 downto 0);
60  SIGNAL NdVcf_integ8b : signed (9 downto 0);
    SIGNAL NVcs_integ8b : signed (9 downto 0);
    SIGNAL NVc2_integ8b : signed (9 downto 0);
    SIGNAL NVc1_integ8b : signed (9 downto 0);
    SIGNAL Vdec_tmp : signed (9 downto 0);
65

    function test_sign(d1,d2:std_logic;nombre : signed) return signed is
    variable sel : std_logic_vector(0 to 1);
    begin
70  sel := d1&d2;
    CASE sel IS
      WHEN "01"  => RETURN ("-"(nombre));
      WHEN "10"  => RETURN (nombre);
      WHEN OTHERS => RETURN to_signed(0,(nombre'length));
75   END CASE;

    end test_sign;

    BEGIN
80
    PROCESS(clk)

    BEGIN
    IF (clk'event and clk ='1') THEN
85
    Vdec_tmp <=  test_sign(P1,P2, NVc1_integ8b)
       + test_sign(P2,P3,NVc2_integ8b)
       + test_sign(P3,'0',to_signed(E,10));

90  ------------------------------------------------------------------
    ------------------------ Calcul des composantes du vecteur d'état---
    ------------------------------------------------------------------

    --- Courant de charge ----------------------------------------------
95  --- equation utilisee dans le cas de l'emulateur sans rebouclage ---
    --NIL_integ <= NIL_integ -coeff_11*NIL_integ8b
    --    + coeff_12 *( test_sign(P1,P2,NVc1_integ8b)
    --              + test_sign(P2,P3,NVc2_integ8b))
    --    + coeffB_11*test_sign(P3,'0',to_signed(E,10));
100 --
    -------- Tension capa 1 -------------------------------------------
    ------------------------------------------------------------------
    IF (NVc1_integ >0) THEN
        NVc1_integ <= NVc1_integ
105                 + test_sign
                    (
                        P2, P1,
                        coeff_21*(to_signed(NIL_integ8b,10)+NIf_integ8b)
                    );
110 ELSE
        NVc1_integ <= test_sign
```

```
                    (
                       P2,P1,
                       coeff_21*(to_signed(NIL_integ8b,10)+NIf_integ8b)
                    );
END IF;
-------- Tension capa 2 ------------------------------------------
------------------------------------------------------------------
IF (NVc2_integ >0) THEN
    NVc2_integ <= NVc2_integ
                + test_sign
                (
                    P3,P2,
                    coeff_21*(to_signed(NIL_integ8b,10)+NIf_integ8b)
                );
ELSE
    NVc2_integ <= test_sign
                    (
                        P3,P2,
                        coeff_21*(to_signed(NIL_integ8b,10)+NIf_integ8b
                            )
                    );
END IF;
--------- Courant filtre -----------------------------------------
------------------------------------------------------------------
NIf_integ <= NIf_integ
            + coeff_42 * (
                            test_sign(P1,P2, NVc1_integ8b)
                            + test_sign(P2,P3,NVc2_integ8b)
                        )
   - coeff_44*NIf_integ8b
   - coeff_45*NVcf_integ8b
   + coeffB_41*test_sign(P3,'0',to_signed(E,10));

-------- Tension capa filtre ------------------------------------
------------------------------------------------------------------
NVcf_integ <= NVcf_integ + coeff_54*NIf_integ8b
   - coeff_55*NVcf_integ8b
   + coeffB_51*test_sign(P3,'0',to_signed(E,10));
END IF;
END PROCESS;
------------------------------------------------------------------
------------------------------------------------------------------

-- Troncature et rebouclage des grandeurs d'etat
-- NIL_integ8b <= NIL_integ(21 downto 12);
NIf_integ8b <= NIf_integ(21 downto 12);
NVcf_integ8b <= NVcf_integ(21 downto 12);

NVc1_integ8b <= NVc1_integ(21 downto 12)
                when (NVc1_integ > 0) else (others =>'0');
NVc2_integ8b <= NVc2_integ(21 downto 12)
                when (NVc2_integ > 0) else (others =>'0');


-- AFFECTATION DES PORTS de sortie
NVc1 <= to_integer(NVc1_integ8b);
NVc2 <=to_integer(NVc2_integ8b);
NIl <= to_integer(to_signed(NIL_integ8b,10));
NIf <= to_integer(NIf_integ8b);
Vdec <= to_integer(Vdec_tmp);
NVcf <= to_integer(NVcf_integ8b);
END arch_emula_obs;
```

spécifications VHDL du rebouclage de l'observateur pour $T_e = 480\eta s$

```
--librairies utilisees ---------------------------------------------
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

--entite gain de l'observateur -------------------------------------
ENTITY em_obs IS
 PORT (
 clk : IN std_logic;
 it  : IN integer;
 IfE : IN integer;

 Niload_est  : OUT integer;
 Nif_est     : OUT integer;
 Nerreur_int : OUT integer
 );
END em_obs;

-----------------------------------------------------------------------
ARCHITECTURE arch_obs OF em_obs IS
constant coef1 : signed (9 downto 0):= "0000010011";
constant coef2 : signed (9 downto 0):= "0000010011";
constant coef3 : signed (9 downto 0):= "0111101100";
constant coef4 : signed (9 downto 0):= "0011101101";

constant gainf : signed (9 downto 0):= "0001000000";
constant gaino : signed (9 downto 0):= "0001010000";
constant gaini : signed (9 downto 0):= "0000001101";
constant gainb : signed (9 downto 0):= "0000010000";

-- memoires du filtre
signal vart0 : signed (15 downto 0):="0000000000000000";
signal vart1 : signed (15 downto 0):="0000000000000000";
signal vart2 : signed (15 downto 0):="0000000000000000";
signal varf1 : signed (15 downto 0):="0000000000000000";
signal varf2 : signed (15 downto 0):="0000000000000000";

signal if_est_int : signed (25 downto 0):="00000000000000000000000000
   ";
signal iload_est_int : signed (25 downto 0):="
   00000000000000000000000000";
signal if_est10b : signed (15 downto 0);
signal iload_est10b : signed (15 downto 0);
signal Ifest : integer;
signal erreur : signed (25 downto 0):="00000000000000000000000000";
signal erreur16b : signed (15 downto 0);

BEGIN
PROCESS(clk)
BEGIN
IF (clk'event and clk='1') THEN
---------------------------------
---
---  in ----->z-1----->vart1
---           |
---           z-1----->vart2
---
---------------------------------
vart0<=to_signed(it,16);
varf1<=if_est10b;

vart1<=vart0;
vart2<=vart1;
```

```
varf2<=varf1;

if_est_int <= coef1*vart0
            - coef2*vart1
            + coef3*if_est10b
            - coef4*varf1;

iload_est_int <= (to_signed(Ifest,16)*gaino-if_est10b*gainf)
                + (to_signed(it,16)*gainb-if_est10b*gaini);

erreur <= to_signed(Ifest,16)*gaino-if_est10b*gainf;
END IF;

END PROCESS;
--TRONCATURE-------------------------------------------------------
if_est10b <= resize(shift_right(if_est_int,8),16) ;
iload_est10b <= resize(shift_right(iload_est_int,4),16);
Ifest <= IfE*64;
erreur16b <= resize(shift_right(erreur,4),16);
--AFFECTATION DES PORTS DE SORTIE----------------------------------
Nif_est <= to_integer(if_est10b);
Niload_est <= to_integer(iload_est10b)/64;
Nerreur_int <= to_integer(erreur16b);
END arch_obs;
```

structure VHDL totale associant l'émulateur et le rebouclage

```
--squelette pour la simulation d'un observateur numerique
--librairies untilisees-------------------------------------
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

-- entite ----------------------------------------
ENTITY emobs IS
 PORT (
        clk : IN std_logic;
        it  : IN integer;
        P1  : IN std_logic;
        P2  : IN std_logic;
        P3  : IN std_logic;
        E   : IN integer;

  NVc1    : OUT integer;
  NVc2    : OUT integer;
  NIl     :  OUT integer;
  NIf     :  OUT integer;
  NVcf    : OUT integer;
  Vdec    : OUT integer;
  Niffiltre: OUT integer;
  Nerreur : OUT integer
 );
END emobs;


-----------------------------------------------------------------
ARCHITECTURE arch_emobs OF emobs IS
-- composant rebouclage --
component em_obs
 PORT (
 clk : IN std_logic;
 it  : IN integer;
 IfE : IN integer;
```

```
    Niload_est  : OUT integer;
    Nif_est     : OUT integer;
    Nerreur_int : OUT integer
    );
  end component;
  -- composant emulateur --
  component emula_rl_obs IS
    PORT (
    clk : IN std_logic;
    P1  : IN std_logic;
    P2  : IN std_logic;
    P3  : IN std_logic;
    E   : IN integer;
    NIL_integ8b : IN integer;
    NVc1    : OUT integer;
    NVc2    : OUT integer;
    NIl     : OUT integer;
    NIf     : OUT integer;
    NVcf    : OUT integer;
    Vdec    : OUT integer
       );
  end component;
  -- lien entre les deux entites --
  signal Nif_link : integer;
  signal Niload_link : integer;

  BEGIN
  instance_emobs : em_obs
      port map (clk ,it,Nif_link,Niload_link,Niffiltre,Nerreur);
  instance_emularlobs: emula_rl_obs
      port map (clk ,P1,P2,P3,E,Niload_link,NVc1,NVc2,NIl,Nif_link,NVcf
          ,Vdec);
  Nif<=Nif_link;
  END arch_emobs;
```