UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAîTRISE EN GÉNIE ÉLECTRIQUE

PAR
SÉBASTIEN LESUEUR

IMPLANTATION EN TECHNOLOGIE ITGE (VLSI) D'UNE LOI DE COMMANDE POUR UNE ARTICULATION FLEXIBLE

SEPTEMBRE 1999

Remerciements

En premier lieu, j'exprime toute ma reconnaissance à :

Mon directeur, le professeur Daniel Massicotte, qui a supervisé mes travaux de recherche, pour ses précieux conseils concernant particulièrement les parties algorithmiques et architecturales, pour sa disponibilité et ses encouragements.

Mon codirecteur, le professeur Pierre Sicard, pour la qualité de ses conseils principalement concernant le développement de la partie commande de mes travaux.

Je leur exprime toute ma gratitude pour le soutien financier qu'il m'ont apporté et démontrant une confiance très appréciée.

Je tiens à remercier également les membres du Laboratoire de Signaux et Systèmes Intégrés, particulièrement Frédéric Morin, Aurélien Mozipo et Martin Vidal. Je les remercie pour l'aide qu'ils m'ont apporté dans ma familiarisation avec les outils de conception en microélectronique et aussi pour l'excellente ambiance de travail qu'ils ont su maintenir dans le laboratoire.

J'adresse des remerciements particuliers à ma compagne Aline Lemoine, pour son soutien, sa patience, ses conseils et ses encouragements qui ont été une grande source de motivation et d'inspiration tout au long de mes travaux.

Enfin, je remercie très chaleureusement mes parents qui m'ont encouragé à poursuivre mes études au Québec et qui m'ont soutenu tant sur le plan moral que financier.

Résumé

L'objectif principal du projet concerne l'implantation en technologie d'intégration à très grande échelle (ITGE – VLSI – *Very Large Scaled Integrated*) d'une loi de commande pour une articulation flexible de manipulateur robotique.

La variété des applications d'une méthode performante de commande de systèmes dynamiques non linéaires montre la pertinence d'une recherche dans ce domaine. Les algorithmes de commande pour les applications requérant de hautes performances nécessitent généralement des traitements fort complexes, impliquant des quantités de calculs très élevées à réaliser en temps réel. L'efficacité de ces algorithmes dans des cas d'applications pratiques résulte alors d'un bon équilibre entre les exigences de la théorie et les possibilités d'implantation.

Face à ce défi, l'utilisation de processeurs commerciaux tels que les processeurs de traitement numérique de signaux (DSP – *Digital Signal Processor*) s'avère inefficace. La conception de processeurs spécialisés dédiés à la commande permet alors, en pensant conjointement les algorithmes et les architectures, de tirer profit du parallélisme inhérent

des algorithmes mis en œuvre. Leur intégration à très grande échelle sur silicium permettra de réaliser des sauts tant quantitatifs que qualitatifs en termes de performances, de fiabilité, de miniaturisation, etc.

Dans ce mémoire, une loi de commande est développée pour le problème de positionnement d'une articulation flexible. Les algorithmes mis en œuvre sont formulés de façon à faciliter le développement d'architectures en vue de leur implantation en ITGE. Ces architectures sont modélisées en langage de haut niveau, simulées afin d'évaluer leurs performances et synthétisées dans une technologie CMOS 0.5µm. Les résultats en surface et vitesse de l'architecture synthétisée sont présentés. Le calcul de la loi de commande par notre architecture nécessite 13 cycles, c'est-à-dire environ 300 fois moins qu'avec un DSP. L'architecture synthétisée comprend environ 400 000 transistors et a une fréquence d'horloge maximale de 4 MHz.

La contribution scientifique du projet comprend :

- le développement d'une loi de commande pour une articulation flexible tenant compte des variations de paramètres, des frottements statiques et de Coulomb et de la flexibilité.
- le développement d'un algorithme performant et robuste aux effets de quantification pour l'estimation des paramètres incertains d'une articulation flexible et basé sur le filtre de Kalman à racine carrée de covariance,
- la proposition d'architectures pour l'implantation des algorithmes mis en œuvre et finalement du contrôleur complet de l'articulation.

Tables des matières

Liste des figures

Liste des tableaux

Liste des sigles et acronymes

ASIC	Application Specific Integrated Circuit
DSP	Digital Signal Processor
FGPA	Field Programmable Gate Array
ITGE	Intégration à Très Grande Échelle
LMS	Least Mean Squares
P	Proportionnel
PD	Proportionnel Dérivatif
PE	Processeur Élementaire
PI	Proportionnel Intégral
PID	Proportionnel Intégral Dérivatif
QRD-RLS	QR Decomposition Recurive Least Squares
RLS	Recursive Least Squares
VHDL	Very High Speed Integrated Circuits Hardware Description Language
VLSI	Very Large Scale Integration

Liste des symboles

$\boldsymbol{C}$	Matrice des forces/couples centrifuges et de Coriolis
e	Erreur d'estimation
f	Gain pour la mise à jour des paramètres
$\boldsymbol{F}$	Matrice des forces/couples de frottements
F_c	Coefficient de frottement de Coulomb
F_{Comp}	Force de compensation de friction
F_F	Force de frottement totale
F_s	Coefficient de frottement statique
F_v	Coefficient de frottement visqueux
$\boldsymbol{g}$	Vecteur de gains pour la mise à jour des paramètres
$\boldsymbol{g}$	Matrices des forces/couples de gravité
I	Innovation
$\boldsymbol{I}$	Matrice identité
J	Inertie
$\boldsymbol{J}$	Matrice Jacobienne
k	Constante de torsion de l'élément de transmission

k ou n Instant d'échantillonnage

K Constante pour le changement de variable sur l'inertie

K_p Gain proportionnel du correcteur PD

$\boldsymbol{K}_s$ Matrice des coefficients de flexibilité

K_v Gain dérivatif du correcteur PD

M Dimension du vecteur de paramètre à estimer

$\boldsymbol{M}$ Matrice des masses

N Rapport de transmission

$\boldsymbol{P}$ Matrice de covariance

p Pôle en boucle fermée

$\boldsymbol{q}$ Vecteur des variables articulaires

$\boldsymbol{S}$ Matrice racine carrée de covariance

s Opérateur de Laplace

s,c Sinus et cosinus de la rotation de Givens

T Période d'échantillonnage

$\boldsymbol{T}$ Transformation de Givens

$\boldsymbol{T}_f$ Matrice des perturbations et des termes dynamiques non modélisés

t_L Constante de temps de la charge

t_M Constante de temps du moteur

u Commande d'un procédé

v Vitesse linéaire

V Critère de la méthode des moindres carrés

v_s Vitesse caractéristique de frottement statique

$\boldsymbol{W}$	Vecteur de paramètre à identifier
y	Sortie mesurable d'un système
α	Accélération angulaire
β	Rapport des variances des bruit sur la mesure et sur l'état du modèle d'estimation
δ	Coefficient d'initialisation des matrices $\boldsymbol{S}$ ou $\boldsymbol{P}$
ξ	Facteur d'amortissement
ε_ω	Limite inférieure de vitesse pour le calcul du couple de frottements
$\boldsymbol{\Phi}$	Vecteur de régression ou régresseur
η	Efficacité du moteur
η_α	Bruit sur l'accélération
η_W	Bruit sur l'état du modèle d'estimation
λ	Facteur d'oubli
θ	Position angulaire
τ	Couple de commande
τ_a	Couple total appliqué excepté les couples de frottements
τ_e	Couple extérieur
τ_F	Couple de frottement total
τ_T	Couple transmis
ω	Vitesse angulaire
ω_n	Fréquence naturelle de résonance
$\bullet_T$	Grandeur reliée à l'élément de transmission
$\bullet_L$	Grandeur reliée à la charge

$\bullet_M$ Grandeur reliée au moteur

$\tilde{\bullet}$ Signal bruité

$\bullet^T$ Transposée d'une matrice ou d'un vecteur

$\bullet^*$ Valeur désirée

$\hat{\bullet}$ Valeur estimée

$\approx$ Environ égal à

Chapitre 1
Introduction

1.1. Problématique

Les performances des systèmes d'entraînement à vitesse variable sont limitées par les propriétés des éléments mécaniques les constituant.

Les manipulateurs de type bras de robots avec articulations flexibles sont soumis aux principales contraintes suivantes [LEV96] [ARM91] [SWE84]:

- flexibilité due aux éléments de transmission,
- phénomènes de frottements,
- limites sur les valeurs des variables articulaires et sur les couples de commande et
- retours de dents au niveau des engrenages de transmission.

Les modèles les plus précis de ces perturbations comprennent des termes fortement non linéaires. D'autre part, certains paramètres mis en jeu dans les modèles d'articulations flexibles sont *a priori* incertains et variables dans le temps. Ces critères nous amènent naturellement à utiliser une méthode de commande adaptative pour concevoir une loi capable de s'adapter en temps réel au comportement de l'articulation.

Ce type de commande requiert généralement l'emploi d'algorithmes de traitement numérique fort complexes. Pour envisager l'emploi d'une telle commande dans un cas pratique, il faut être capable d'évaluer la commande en temps réel. Étant donné la complexité du traitement numérique requis, il en résulte une quantité très importante de calculs à effectuer dans des délais d'autant plus courts que le système sous contrôle est rapide.

Dans le cas d'applications requérant de hautes performances, ces délais deviennent trop faibles pour que l'utilisation de processeurs commerciaux habituels s'avère efficace. Citons par exemple : les robots employés pour la manipulation de gaufres de circuits intégrés, les systèmes de positionnement de précision, la commande de micromanipulateurs, etc. [GRA96] [DAN96]

1.2. Objectifs

L'objectif général du projet de recherche concerne l'implantation en technologie ITGE d'une loi de commande pour une articulation flexible de robot. Cet objectif comprend d'une part le développement d'une loi de commande adaptative pour une articulation

flexible permettant de satisfaire aux objectifs de commande, et d'autre part l'étude des possibilités d'implantation de la loi de commande pour aboutir finalement à l'implantation dans une structure justifiée.

Cet objectif peut être décomposé en sous objectifs :

- modélisation d'une articulation flexible et de ses contraintes,
- développement d'une loi de commande adaptative performante permettant de satisfaire aux objectifs de commande pour l'articulation modélisée,
- développement d'un algorithme permettant l'estimation en ligne des paramètres incertains de l'articulation,
- formulation de la méthode d'estimation permettant d'augmenter son parallélisme inhérent et de limiter les effets de quantification en vue de son implantation en technologie ITGE,
- proposition d'architectures ITGE capables de recevoir d'une part l'algorithme d'estimation et d'autre part le contrôleur complet,
- modélisation et simulation des architectures proposées en langage VHDL,
- synthèse des modèles VHDL dans une technologie CMOS 0.5µm,
- évaluation des performances de l'architecture dédiée à la loi de commande.

1.3. Méthodologie

L'objectif du projet sera atteint en travaillant dans un premier temps sur la commande de l'articulation considérée. Cette étape de recherche permettra d'établir les modèles dynamiques d'une articulation flexible et des phénomènes de friction associés en fonction des connaissances actuelles de la recherche dans le domaine. La connaissance de ces

modèles nous permettra de concevoir une loi de commande performante pour le contrôle de position de l'articulation en supposant tous les paramètres de l'articulation connus.

Ensuite, un algorithme d'estimation des paramètres incertains de l'articulation devra être développé. Cet algorithme devra fournir des valeurs estimées des paramètres de suffisamment bonne qualité pour maintenir les performances de la loi de commande développée lors de la première étape. Des simulations permettront d'évaluer les performances de la loi de commande et de valider les résultats avant d'envisager l'implantation.

Une étude approfondie des possibilités d'implantation de la loi de commande conçue sera ensuite menée. Cette étude portera principalement sur l'algorithme d'estimation qui devra être formulé de façon à augmenter son degré de parallélisme et à le rendre moins sensible aux effets de quantification. Cette étude devra permettre de dégager le degré de parallélisme inhérent de la loi de commande et d'évaluer les performances nécessaires à sa mise en œuvre dans un cas d'application pratique. L'étude comprendra les effets de quantification, les ressources nécessaires à l'implantation, le nombre minimum de cycles d'horloge pour l'exécution du calcul de la commande, les contraintes technologiques, etc.

Ceci permettra finalement de justifier le choix d'une technologie d'implantation. La loi de commande sera implantée dans la structure ITGE retenue et simulée pour confirmer la fonctionnalité et les performances de l'ensemble.

1.4. Structure du mémoire

Après l'introduction, le deuxième chapitre comprendra une synthèse de l'état d'avancement des connaissances dans les domaines considérés dans ce mémoire, c'est-à-dire la modélisation d'articulations de robots et des phénomènes de frottements auxquels elles sont soumises, les méthodes de commande utilisées en robotique, les algorithmes d'estimation de paramètres de systèmes dynamiques et enfin les architectures développées permettant l'implantation de ces algorithmes en technologie ITGE. Cette étude préalable permettra de constituer une base pour les différentes étapes du projet de recherche.

Dans le troisième chapitre, le modèle de l'articulation flexible et ses propriétés seront décrits. Une stratégie de commande sera développée pour le problème de positionnement de l'articulation et ses performances évaluées en supposant tous les paramètres de l'articulation connus.

Dans le quatrième chapitre, la méthode d'estimation des paramètres de l'articulation sera présentée. Le choix de cette méthode résultera d'une part d'une formulation du problème permettant l'utilisation de méthodes d'estimation connues et d'autre part de la prise en compte des contraintes d'implantation. Les performances de l'algorithme d'estimation retenu seront discutées dans ce chapitre.

Le cinquième chapitre concernera l'implantation de la loi de commande développée en technologie ITGE. Dans ce chapitre, les architectures proposées pour l'implantation de

l'algorithme d'estimation et du contrôleur de l'articulation seront présentées. Les résultats de simulations des modèles VHDL de ces architectures seront également présentés dans ce chapitre pour aboutir finalement à une étude de performances.

Finalement, la conclusion permettra de rappeler les principaux résultats obtenus et de dégager l'originalité et la contribution du projet. Cette conclusion s'achèvera par une discussion de l'application des résultats et des développements futurs du projet.

Chapitre 2

Applications de la microélectronique à la commande des robots

2.1. Modélisation des articulations de robots

Le développement d'un modèle pour un système robotique est en général indispensable pour la mise au point des stratégies de commande, la conception des lois de commande, leur réglage ou encore la conception de capteurs ou d'estimateurs d'états du système [FLA94]. Nous distinguons les modèles cinématiques qui décrivent un état permanent du système des modèles dynamiques qui permettent de représenter l'évolution du système dans le temps.

Les robots sont en général définis comme des systèmes mécaniques articulés dotés d'un organe terminal de préhension ou en forme d'outil leur permettant de réaliser des tâches.

Un robot est donc constitué de plusieurs membres, à l'image du bras humain, et les articulations sont les jonctions entre les différents membres.

La structure du modèle d'une articulation peut dépendre des caractéristiques particulières ou des conditions particulières d'utilisation de celle-ci. En conséquence, les chercheurs ont très souvent développé leurs propres modèles, satisfaisant à leurs conditions et décrivant avec le plus d'exactitude possible le comportement de leurs systèmes [SIC93]. De plus, il peut s'avérer plus intéressant de développer un modèle possédant des propriétés particulières facilitant l'emploi de méthodes de commande connues.

2.1.1. Robots rigides

La formulation de Lagrange-Euler peut être perçue comme une formulation générale de tous les modèles actuellement utilisés et comprenant tous les effets rencontrés dans le comportement des robots rigides [CRA89]. D'après cette formulation, les équations dynamiques d'un manipulateur sériel à n articulations peuvent s'écrire sous la forme matricielle suivante :

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{h} \qquad (2.1)$$

Où $\boldsymbol{M}$ est la matrice des masses, $\boldsymbol{C}$ est la matrice des termes centrifuges et de Coriolis, $\boldsymbol{F}$ est la matrice des forces de frottements, $\boldsymbol{g}$ représente l'action de la gravité, $\boldsymbol{\tau}$ est la matrice des couples appliqués par les actionneurs, le terme $\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{h}$ représente l'action de l'effecteur sur son environnement ou de l'environnement sur l'effecteur, et $\boldsymbol{q}$ désigne le vecteur des variables articulaires.

Tous les modèles de manipulateurs robotiques rigides peuvent être déduits de cette formulation en considérant les termes appropriés et en adaptant l'écriture des matrices au système étudié.

2.1.2. Robots avec joints flexibles

La conception de contrôleurs pour les robots avec joints flexibles a reçu beaucoup d'attention des chercheurs ces dernières années. La raison principale est que la flexibilité doit être prise en compte dans la modélisation et la conception de contrôleurs pour les robots dans les applications requérant de hautes performances. Ceci a par exemple été démontré expérimentalement dans [SWE84]. Il existe deux sources essentielles de flexibilité :

- L'élasticité des éléments de transmission (engrenages),
- Les mouvements brusques de charge.

Dans [ZHI96], un modèle est décrit pour un robot avec joints flexibles. Ce modèle est très semblable au modèle décrit au paragraphe 2.2.1, la flexibilité étant modélisée comme une fonction des déformations angulaires entre les actionneurs et les charges des différentes articulations. Ainsi, les équations dynamiques d'un robot avec joints flexibles peuvent s'écrire :

$$\boldsymbol{0} = \boldsymbol{M}(\boldsymbol{q}_1)\ddot{\boldsymbol{q}}_1 + \boldsymbol{N}(\boldsymbol{q}_1, \dot{\boldsymbol{q}}_1) + \boldsymbol{K}_s(\boldsymbol{q}_1 - \boldsymbol{q}_2) \tag{2.2}$$

$$\boldsymbol{J}\ddot{\boldsymbol{q}}_2 = \boldsymbol{K}_s(\boldsymbol{q}_1 - \boldsymbol{q}_2) - \boldsymbol{D}\dot{\boldsymbol{q}}_2 + \boldsymbol{\tau} + \boldsymbol{T}_f(\boldsymbol{q}_1, \dot{\boldsymbol{q}}_1) \tag{2.3}$$

Où $\boldsymbol{q}_1$ et $\boldsymbol{q}_2$ désignent respectivement les vecteurs des variables articulaires des membres et des actionneurs, $\boldsymbol{N}$ est la matrice regroupant les termes de Coriolis, de gravité et de

frottements, K_s est une matrice diagonale contenant les coefficients de flexibilité des différentes articulations, T_f représente les perturbations et les termes dynamiques non modélisés, les autres termes ont les mêmes significations que dans (2.1).

Nous verrons au troisième chapitre un modèle possible pour articulation flexible [SIC93], dans lequel la charge et le moteur sont modélisés par une structure à deux intégrateurs très largement utilisée en commande. La boîte de transmission est modélisée par un ressort de torsion équivalent donnant le couple transmis à la charge proportionnel à la déformation angulaire entre les arbres du moteur et de la charge.

2.2. Modélisation et compensation des phénomènes de frottements

2.2.1. Modélisation des phénomènes de frottements

Les phénomènes de frottements (ou friction) constituent également des contraintes non négligeables sur les performances des robots. Les modèles de frottements utilisés dans la conception des contrôleurs ont longtemps été limités à des modèles très simples, représentant les effets facilement modélisables : frottements visqueux, de Coulomb et éventuellement une modélisation simpliste des frottements statiques comme le montre la figure 2-1 [ARM91]. Ces modèles présentent généralement l'intérêt d'utiliser des fonctions simples facilitant la conception des contrôleurs associés.

Des études expérimentales ont démontré la présence d'autres phénomènes de frottements non modélisés principalement à basse vitesse. Nous retrouvons dans la littérature les

termes de friction de Stribeck, de mémoire de friction, etc. [ARM95] Ceci a amené certains chercheurs à développer de nouveaux modèles représentant de façon plus précise les phénomènes réellement rencontrés en pratique.

Seidl utilise par exemple un modèle comprenant une bande morte autour de la vitesse nulle pour traduire l'effet de frottement statique [SEI92].

Canudas de Wit propose un nouveau modèle en 1995 qui est une extension du modèle de Dahl et qui donne l'expression du modèle dynamique de friction sous forme d'une équation différentielle dont la variable est la déflexion entre les surfaces de contact [CAN95].

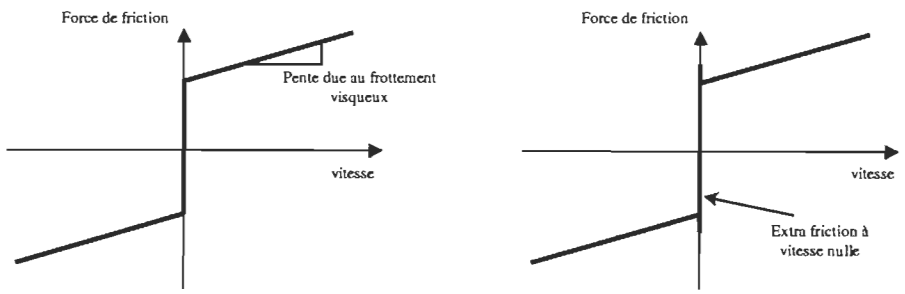


Figure 2-1 : Modèles simples de friction

Finalement, le modèle qui paraît à l'heure actuelle proposer le meilleur compromis entre l'exactitude du modèle et les possibilités de compensation donne la force de friction instantanée au point de contact de deux surfaces en fonction de la vitesse instantanée de glissement entre ces deux surfaces. Ce modèle comprend trois termes décrivant les phénomènes de frottements de Coulomb, statique et visqueux [ARM96].

D'après ce modèle, cette force peut s'écrire :

$$F_{F(t)} = F_c sign(v_{(t)}) + F_v v_{(t)} + F_s sign(v_{(t)}) \exp\left(-(v_{(t)} / v_s)^2\right) \qquad (2.4)$$

Où F_c, F_v, F_s sont respectivement les coefficients de frottements de Coulomb, visqueux et statique, v_s est la vitesse caractéristique de frottement statique et $v_{(t)}$ est la vitesse relative instantanée de déplacement. La fonction exponentielle qui modélise les frottements statiques est aussi appelée « courbe de Stribeck ». La figure 2-2 donne l'allure de la fonction décrite par l'équation (2.4).

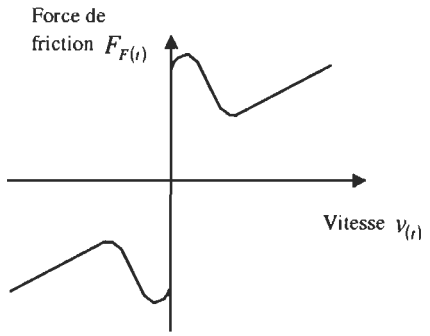


Figure 2-2 : Allure de la fonction de friction en fonction de la vitesse

Ce type de modèle n'est pas toujours utilisé dans la compensation de friction à cause de la présence de termes non linéaires plus difficiles à évaluer tels que le terme dit « de Stribeck ». Toutefois, dans le domaine du contrôle de systèmes de haute précision, il s'avère indispensable de compenser le plus efficacement possible tous les effets de frottements, et dans ce cas les modèles plus complexes sont considérés et utilisés dans la compensation.

2.2.2. *Compensation des phénomènes de frottements*

Avec le souci croissant de compenser efficacement les phénomènes de frottements dans les systèmes robotiques, de nombreux travaux de recherche ont été réalisés dans ce domaine [ARM91]. Ces phénomènes peuvent être à l'origine de pertes de performances allant jusqu'à 15%.

La solution peut se situer à deux niveaux : soit tenter de minimiser les effets de frictions par des modifications matérielles (meilleure lubrification, utilisation de matériaux plus performants, etc.) soit évaluer au mieux ces phénomènes et utiliser des méthodes de compensation [ARM96].

Les forces/couples de frictions agissent en contrainte sur le mouvement des articulations. Elles sont généralement représentées dans la modélisation d'articulations par une rétroaction négative sur la commande de celles-ci. La plupart des méthodes de compensation sont alors basées sur l'ajout à la commande d'un couple opposé au couple de friction ayant pour effet d'annuler celui-ci comme illustré à la figure 2-3.

Pour calculer ce couple total de friction, les chercheurs ont d'abord considéré les modèles de friction simples et supposé leurs paramètres connus. Ainsi, d'après les mesures de vitesse, éventuellement de position lorsque le modèle l'exige, le couple de compensation peut être calculé simplement à partir du modèle et injecté dans la commande. Ce type de compensation nécessite des expérimentations pour l'identification des coefficients de friction et peut être implanté de façon simple tant dans des structures analogiques que numériques.

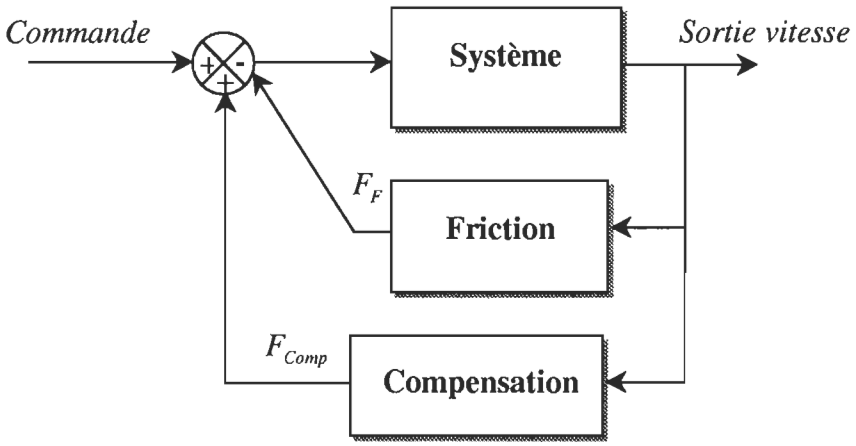


Figure 2-3 : Schéma général de compensation

Idéalement, nous souhaitons obtenir :

$$F_{Comp} = F_F \tag{2.5}$$

Cependant, il n'est pas forcément possible de déterminer les paramètres des modèles. Certaines expérimentations sur des machines ont permis d'obtenir des résultats, mais lorsqu'on considère les modèles les plus complets, tous les paramètres ne peuvent être déterminés expérimentalement. Le coefficient de frottement de Coulomb dans un moteur est parfois considéré constant et déterminé avec une bonne précision de façon expérimentale mais lorsqu'il s'agit de frottement statique, l'identification est une tâche beaucoup plus difficile. De plus, les paramètres de friction peuvent être variables à cause des changements de conditions d'opération, de l'usure des matériaux, etc. C'est pour cela que les recherches s'orientent vers l'utilisation de méthodes adaptatives permettant d'identifier en ligne ces paramètres pour une compensation plus efficace et plus efficace. La complexité de l'identification dépend alors du modèle utilisé et des variations des paramètres considérés.

2.3. Commande adaptative des systèmes robotiques

Une fois qu'un robot est conçu et construit, la conception de contrôleurs pour ce robot devient déterminante pour son utilisation dans des applications pratiques. Diverses méthodes de conception de contrôleurs telles que la conception dans le domaine des fréquences, la commande robuste, la commande adaptative, la linéarisation par retour d'état ou la commande à structure variable ont fait l'objet de nombreuses publications ces dernières années [ZHI96] [VID89]. Anderson et Spong utilisent une commande hybride, combinaison de plusieurs de ces types de commande [AND88].

Les paramètres ou les propriétés des signaux des modèles utilisés dans la commande de systèmes robotiques peuvent être inconnus ou variables dans le temps. C'est la raison pour laquelle, à l'heure actuelle, un grand nombre de travaux de recherche s'orientent vers la commande adaptative.

Par définition, un contrôleur adaptatif d'un procédé est un système capable d'adapter son comportement aux propriétés du procédé qu'il contrôle et de ses signaux. Il s'applique donc aux systèmes dont les paramètres sont inconnus ou variants dans le temps [AST87]. Les algorithmes utilisés dans les contrôleurs adaptatifs sont généralement non linéaires. Ce type de commande requiert une quantité importante de calculs, sa robustesse et sa stabilité ne sont pas toujours garanties.

Les stratégies de commande adaptative peuvent être classées en trois catégories : la programmation des gains (« gain scheduling »), la commande adaptative avec modèle de

référence (« model reference adaptive control ») et les régulateurs auto-syntonisants (« self-tuning regulators »). Toutes les stratégies de contrôle adaptatif peuvent être classifiées dans une de ces trois catégories.

- Programmation des gains [AST87] :

Il est parfois possible de trouver des variables auxiliaires du procédé qui possèdent un bon degré de corrélation avec les variations de la dynamique du procédé. Il est alors possible d'éliminer l'influence des variations des paramètres du procédé en changeant les paramètres du correcteur en fonction des variables auxiliaires. Cette approche est appelée programmation des gains.

L'avantage principal de cette stratégie est que les paramètres du correcteur peuvent être modifiés très rapidement en réponse aux fluctuations du procédé. Les facteurs limitatifs dépendent de la rapidité avec laquelle les mesures auxiliaires répondent aux fluctuations du procédé. Cependant, la conception demande beaucoup de temps à cause du grand nombre de répétitions du processus de conception et de simulation.

Ainsi, cette méthode est simple à implanter et son action rapide. C'est une méthode très employée mais qui possède des limitations très importantes : le temps de conception est long, la correction des gains est de type boucle ouverte, le nombre de gains à programmer peut être important pour obtenir les performances désirées et son utilisation avec des systèmes non linéaires peut donner des résultats peu satisfaisants. Ce type de commande est souvent employé dans l'industrie pour les systèmes linéaires.

- Commande adaptative avec modèle de référence [AST87] [DOT88] :

Il s'agit d'une autre méthode permettant d'adapter les paramètres du correcteur selon des spécifications définies par un modèle de référence qui décrit la réponse idéale de la sortie du procédé à un signal de commande.

Le contrôleur est constitué de deux boucles, une boucle interne de commande classique comprenant le procédé et le régulateur, et une boucle externe qui adapte les paramètres du régulateur de façon à minimiser l'erreur entre la sortie du procédé et la sortie du modèle de référence.

Ce type de commande peut être très efficace mais présente des problèmes de stabilité et une complexité importante des algorithmes d'ajustement des paramètres rendant difficile l'implantation dans les calculateurs numériques.

- Régulateurs auto-syntonisants [AST87] [DOT88] :

La conception d'un régulateur auto-syntonisant utilise la stratégie suivante : a) déterminer un modèle approprié pour le système; b) estimer les paramètres du système avec un estimateur récursif; c) utiliser les paramètres estimés pour déterminer la loi de commande. Cette fois encore nous retrouvons la structure à deux boucles, la boucle interne qui comprend le procédé et un correcteur avec rétroaction et la boucle externe qui ajuste les paramètres du correcteur. L'algorithme d'identification des paramètres peut être basé sur les critères de marge de gain ou de phase, sur le positionnement de pôles, sur la minimisation de la variance ou divers autres critères.

Une structure de commande également fréquemment utilisée est la structure avec anticipation et rétroaction. Il s'agit de calculer la commande du procédé comme la somme de deux termes : le premier, appelé signal d'anticipation, calculé comme la réponse du modèle inverse du procédé au signal de consigne et le deuxième, appelé signal de rétroaction, calculé en fonction des perturbations autour du point d'opération désiré et ayant pour rôle de compenser ces perturbations [KFU87].

2.4. Identification des paramètres d'un système

L'identification en ligne des paramètres d'un procédé est une clé essentielle pour le contrôle adaptatif [ISE82]. Beaucoup de recherches ont été menées dans le passé sur la commande adaptative, mais souvent limitées dans leurs applications par les possibilités matérielles. Le développement de calculateurs numériques performants et à faible coût a grandement réactivé le champ de développement du contrôle adaptatif.

Les méthodes de contrôle adaptatif des robots ont été vues au paragraphe 2.3. Il apparaît souvent nécessaire dans ce type de commande d'estimer certains paramètres du procédé ou de son modèle.

Si ces paramètres sont constants, ils peuvent éventuellement être identifiés hors ligne et être ensuite utilisés directement dans le calcul de la commande. Lorsque ces paramètres sont variables et de façon incertaine, il faut les identifier en ligne. Il existe plusieurs méthodes permettant d'identifier les paramètres d'un procédé.

L'identification peut être définie comme l'opération de détermination des caractéristiques dynamiques d'un procédé (système) dont la connaissance est nécessaire pour la conception et la mise en œuvre d'un système performant de régulation [LAN93].

L'identification en ligne de paramètres variables nécessite une capacité d'apprentissage de la méthode. En conséquence, la plupart des méthodes sont basées sur des algorithmes récursifs. Parmi les méthodes utilisées, nous trouvons :

- Méthodes graphiques :

La réponse du système à un échelon ou à une impulsion sur la variable manipulée peut contenir des renseignements intéressants sur sa dynamique [FLA94].

- Méthodes de décorrélation du vecteur d'observation et de l'erreur de prédiction :

L'idée est d'obtenir un vecteur d'observations fortement corrélé et donc représentatif des variables non bruitées du système mais non corrélé avec le bruit [LAN93].

- Méthode des moindres carrés :

Il s'agit probablement de la méthode la plus fréquemment utilisée. Elle s'applique lorsqu'un signal connu (mesurable ou calculable) peut être écrit sous une forme linéaire en fonction des paramètres à identifier.

Soit y ce signal, le modèle linaire est donné par l'équation (2.6) :

$$y = \boldsymbol{W}^T \boldsymbol{\Phi} \tag{2.6}$$

Où $\boldsymbol{W}$ est le vecteur des paramètres à identifier et $\boldsymbol{\Phi}$ est un vecteur (ou une matrice) de signaux mesurés souvent appelé régresseur.

La méthode permet alors d'estimer les paramètres en minimisant le critère d'erreur quadratique :

$$\varepsilon_{(k)} = \sum_{k=1}^{N} \left(y_{(k)} - \hat{\boldsymbol{W}}_{(k-1)}^T \boldsymbol{\Phi}_{(k)} \right)^2 \tag{2.7}$$

k est l'instant d'échantillonnage.

Plusieurs formulations de la méthode des moindres carrés existent, leur différence se situe généralement au niveau de la mise à jour des paramètres estimés en fonction de l'erreur d'estimation [AST95].

Les deux méthodes les plus répandues sont les méthodes LMS (Least Mean Squares) et RLS (Recursive Least Squares). La mise à jour des paramètres s'écrit :

$$\hat{\boldsymbol{W}}_{(k)} = \hat{\boldsymbol{W}}_{(k-1)} + \lambda \boldsymbol{\Phi}_{(k)} \left(y_{(k)} - \hat{\boldsymbol{W}}_{(k-1)}^{T} \boldsymbol{\Phi}_{(k)} \right) \tag{2.8}$$

pour la méthode LMS,

$$\begin{cases} \boldsymbol{P}_{(k)} = \boldsymbol{P}_{(k-1)} - \boldsymbol{P}_{(k-1)} \boldsymbol{\Phi}_{(k)} \boldsymbol{\Phi}_{(k)}^{T} \boldsymbol{P}_{(k-1)} \\ \hat{\boldsymbol{W}}_{(k)} = \hat{\boldsymbol{W}}_{(k-1)} + \boldsymbol{P}_{(k)} \boldsymbol{\Phi}_{(k)} \left(y_{(k)} - \hat{\boldsymbol{W}}_{(k-1)}^{T} \boldsymbol{\Phi}_{(k)} \right) \end{cases} \tag{2.9}$$

pour la méthode RLS.

Dans ces équations, λ est le gain de la méthode LMS et $\boldsymbol{P}$ est la matrice de covariance pour la méthode RLS.

Des variantes ont été apportées à ces méthodes de base pour les rendre fonctionnelles dans certains cas particuliers, nous retrouvons par exemple les méthodes RELS (Recursive Extended Least Squares), EFRLS (Exponentially Forgetting Recursive Least Squares), etc. [ISE82] [HAY98]

- Filtre de Kalman :

Le filtre de Kalman est une alternative du filtre de Wiener permettant de résoudre les problèmes d'estimation ou de prédiction de l'état et des paramètres de systèmes linéaires

dans un environnement bruité. Il permet également de minimiser le critère des moindres carrés.

Cependant, l'utilisation de ce type d'algorithme exige la disponibilité d'un modèle du comportement dynamique du procédé. Dans certaines applications, les procédés ne sont pas modélisables ou de façon peu précise, dans ce cas nous utilisons des méthodes d'identification qui ne requièrent pas nécessairement la connaissance précise du modèle, parmi ces méthodes nous retrouvons le plus fréquemment les réseaux de neurones et la logique floue [ROV95] [MOU95].

2.5. Implantation d'algorithmes de commande en technologie ITGE

Les algorithmes mis en œuvre dans les méthodes de contrôle adaptatives exigent généralement des traitements numériques très complexes, ce qui signifie de grands nombres d'opérations. Pour utiliser ces méthodes dans des applications réelles, ces opérations doivent être exécutées en temps réel, c'est-à-dire dans des temps se rapportant à la rapidité d'évolution des systèmes sous contrôle. Les algorithmes de commande sont le plus souvent implantés dans des processeurs commerciaux de traitement numérique de signaux (DSP, microcontrôleurs, etc.) [DES90] [TEO91]. Cependant, les architectures de ces processeurs sont conçues pour être versatiles et exécuter les instructions de façon séquentielle au prix de performances limitées.

Le développement de la technologie d'intégration à très grande échelle a donné naissance à de nouvelles générations de circuits tels que les ASIC, FPGA, etc. Ces circuits comprennent un très grand nombre de transistors et peuvent être configurés

spécifiquement pour réaliser des tâches précises, on parle alors de processeurs dédiés. Il devient alors possible de développer des architectures permettant la mise en parallèle des calculs et des communications de données d'un algorithme et d'implanter ces architectures directement sur silicium. Cependant, l'implantation d'algorithmes de commande en technologie ITGE en robotique est encore relativement peu répandue [BOS92].

Nous nous intéresserons plus particulièrement aux possibilités d'implantation de la méthode des moindres carrés qui sera utilisée dans notre contrôleur. Les algorithmes de base tels que LMS ou RLS ne sont pas intrinsèquement parallèles et s'avèrent le plus souvent sensibles aux effets de quantification. De grands progrès ont été réalisés ces dernières années et ont permis l'avènement d'une nouvelle classe d'algorithmes, dont la stabilité numérique, la robustesse aux effets de quantification et le degré de parallélisme ont été grandement améliorés.

En 1983, McWhirter démontre que l'algorithme QRD-RLS basé sur la décomposition QR de la méthode RLS, peut être implanté en technologie ITGE dans une architecture systolique [WHI83]. La décomposition QR qui est encore la plus utilisée, peut être obtenue en appliquant les rotations de Givens, la méthode de Gram-Schmidt ou encore la transformation de Householder [HAY98].

En 1991, Jabbari démontre que la transformation de la méthode QR avec l'opérateur δ Levinson-Schur permet de résoudre le problème de mauvais conditionnement des matrices dû à l'échantillonnage hors bande passante des signaux [JAB91].

Ces méthodes présentent néanmoins le gros inconvénient d'utiliser des fonctions de racine carrée et de division coûteuses à l'implantation. D'autres méthodes comprenant moins de divisions et de racines carrées sont apparues dès 1994, ces méthodes sont basées sur les rotations μV ou kV [FRA94] ou encore sur les rotations par tangentes [RAG96].

Le filtre de Kalman a aussi fait l'objet de nombreuses études en vue de son implantation ITGE. L'analogie des équations d'estimation avec la méthode RLS décrite dans [SAY94] et [HAY98] permet l'utilisation des mêmes types d'architectures pour l'implantation du filtre de Kalman [KUN91].

Enfin, les algorithmes à racine carrée de covariance (Square Root Covariance Algorithms) ont apporté une solution efficace au problème d'instabilité de l'équation de mise à jour de la matrice de covariance tant pour la méthode RLS que pour le filtre de Kalman [HAY98].

Chapitre 3

Modélisation et commande d'une articulation flexible

3.1. Introduction

Pour concevoir la loi de commande, nous devons d'abord définir un modèle dynamique de l'articulation flexible. Il est très important de choisir un modèle représentant le comportement du système étudié avec un degré de précision suffisamment important pour pouvoir accorder une grande confiance aux résultats obtenus. De plus, il est important de choisir une structure et des propriétés du modèle facilitant la conception de la loi de commande. Cette loi de commande devra permettre de satisfaire aux critères de contrôle, principalement le suivi de trajectoire et la stabilité, en respectant les contraintes naturelles de l'articulation (limites sur les valeurs des variables articulaires, limites sur les couples de commande, bande passante, etc.)

3.2. Modélisation d'une articulation flexible

3.2.1. Structure du modèle

Nous avons déjà vu que les équations dynamiques du mouvement d'un manipulateur rigide à *n* articulations peuvent s'écrire sous la forme des équations de Lagrange-Euler :

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}(\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} - \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{h} \qquad (3.1)$$

Où $\boldsymbol{M}$ est la matrice des masses, $\boldsymbol{C}$ est la matrice des termes centripètes, centrifuges et de Coriolis, $\boldsymbol{F}$ est la matrice des forces de frottements, $\boldsymbol{g}$ représente l'action de la gravité, $\boldsymbol{\tau}$ est la matrice des couples appliqués par les actionneurs, le terme $\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{h}$ représente l'action de l'effecteur sur son environnement et $\boldsymbol{q}$ désigne le vecteur des variables articulaires.

Le système d'équations différentielles, ou aux différences dans le cas discret, décrit par (3.1) comporte autant de lignes que le manipulateur d'articulations.

Tous les modèles de manipulateurs rigides peuvent être déduits de cette équation en considérant les termes appropriés et en adaptant l'écriture des matrices au système étudié.

Pour adapter l'écriture des équations au cas de manipulateurs flexibles, nous adoptons le modèle vu au paragraphe 2.1.2 et décrit dans [CRA89].

Dans cette étude, nous considérons le problème de contrôle de position d'une articulation flexible seule, composée d'un moteur à courant continu pilotant une charge par le biais d'une boîte de transmission équivalente à un ressort de torsion de caractéristique linéaire [SID98].

Dans la suite, les indices *M, L,* et *T* désigneront respectivement les valeurs reliées au moteur, à la charge et à la transmission. L'indice *F* désigne les termes relatifs à la friction. La figure 3-1 donne une représentation du modèle de l'articulation comprenant le moteur, la charge et la boîte de transmission.

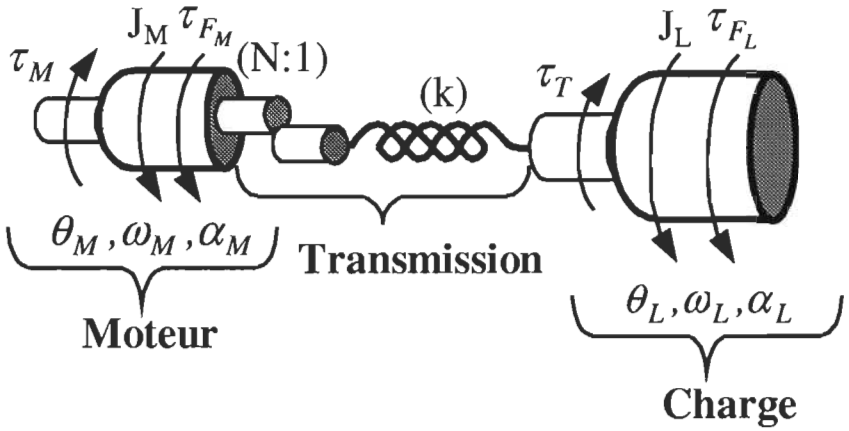


Figure 3-1 : Structure de l'articulation flexible

Les équations mathématiques du modèle sont :

$$\frac{d\omega_M}{dt} = J_M^{-1}\left(\tau_M - \tau_{F_M} - \eta\tau_T\right) \tag{3.2}$$

$$\frac{d\theta_M}{dt} = \omega_M \tag{3.3}$$

$$\frac{d\omega_L}{dt} = J_L^{-1}\left(N\tau_T - \tau_{F_L}\right) \tag{3.4}$$

$$\frac{d\theta_L}{dt} = \omega_L \tag{3.5}$$

Les variables dépendantes du temps sont les positions, vitesses et accélérations angulaires respectivement notées θ, ω et α, et les couples τ. J représente l'inertie du moteur ou de la charge et sera considérée comme un paramètre incertain. η est l'efficacité de la

boîte de transmission et dépend du mode de fonctionnement du moteur, k la raideur du ressort de torsion équivalent à la boîte de transmission et N le rapport de transmission.

Les variables η, N et k seront supposées constantes et connues dans ce travail, mais ce n'est pas toujours le cas notamment pour la constante de torsion.

Dans certains travaux de recherche, des modèles de flexibilité plus complexes ont été développés, mais le modèle retenu dans cette étude reste tout de même d'usage courant.

3.2.2. Modèle d'une articulation flexible

3.2.2.1. Modèles du moteur et de la charge

La structure générale de l'articulation flexible est divisée en trois éléments : le moteur, la charge et l'élément de transmission.

Les comportements du moteur et de la charge peuvent être décrits par des modèles similaires classiquement utilisés en commande. Ces modèles relient la position angulaire au couple appliqué, nous y retrouvons un double intégrateur et le modèle de friction comme illustré par la figure 3-2.

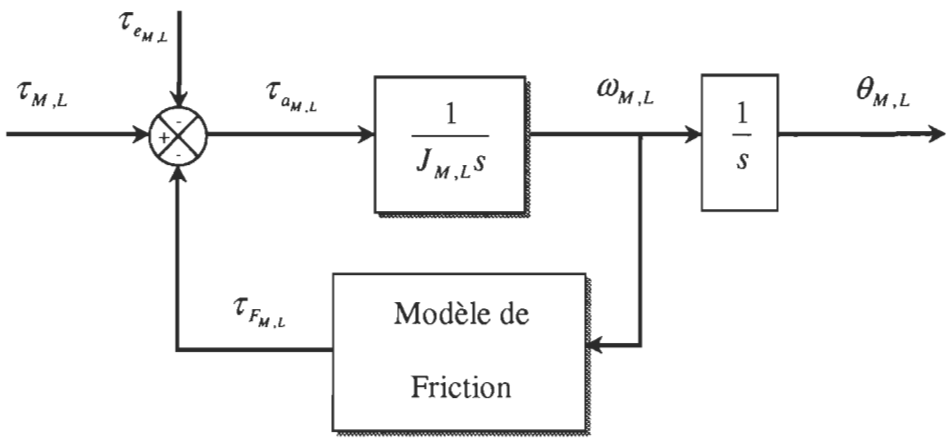


Figure 3-2 : Modèle du moteur et de la charge

τ désigne le couple appliqué, couple électromagnétique de commande pour le moteur ou couple mécanique transmis pour la charge, τ_F désigne le couple total de frottements, τ_e comprend la somme des couples extérieurs, c'est-à-dire le couple réfléchi par la charge pour le moteur et l'action des autres membres ou de l'environnement pour la charge.

3.2.2.2. Modèle des phénomènes de friction

Les phénomènes de frottements jouent un rôle important dans le contrôle des systèmes robotiques. Ces phénomènes peuvent amener des erreurs de suivi de trajectoire, des cycles limites et de nombreux autres effets indésirables [CAN95]. Les stratégies de contrôle qui tentent de compenser efficacement ces phénomènes requièrent un modèle suffisamment exact.

Généralement, la force de friction totale est exprimée en fonction de la vitesse de glissement et éventuellement de la position relative des deux surfaces.

Le modèle qui paraît à l'heure actuelle proposer le meilleur compromis entre l'exactitude du modèle et les possibilités de compensation donne la force de friction instantanée au point de contact de deux surfaces en fonction de la vitesse instantanée de glissement entre ces deux surfaces. Ce modèle comprend trois termes décrivant les phénomènes de frottement de Coulomb, statique et visqueux comme nous l'avons vu au deuxième chapitre [CAN96]. Les effets de friction peuvent être modélisés comme un couple de contrainte s'il s'agit d'une liaison rotative ou comme une force de contrainte dans le cas d'une liaison prismatique. Dans la suite, nous adopterons arbitrairement la notation de couple dans les deux cas.

Ainsi, le couple total de friction τ_F au point de contact de deux surfaces de vitesse relative ω s'écrit :

$$\tau_{F(t)} = F_c sign(\omega_{(t)}) + F_v \omega_{(t)} + F_s sign(\omega_{(t)}) \exp\left(-(\omega_{(t)} / \omega_s)^2\right) \qquad (3.6)$$

Où F_c, F_v, F_s sont respectivement les coefficients de frottements de Coulomb, visqueux et statique, ω_s est la vitesse caractéristique de frottement statique.

Nous avons vu dans le paragraphe 2.2.1 une représentation de la courbe de friction pour des valeurs numériques données des coefficients du modèle. Ce modèle est à l'heure actuelle celui qui, parmi les modèles statiques, représente le plus fidèlement les phénomènes de frictions observés expérimentalement. Cependant, la complexité des fonctions mises en œuvre amène souvent les chercheurs à utiliser des modèles plus simples au prix de performances réduites. La non linéarité du modèle autour de la vitesse nulle peut causer des oscillations numériques importantes lors des simulations. De plus, le couple de frottements ne doit pas entraîner la charge ou le moteur à vitesse faible ce qui pourrait être le cas avec le modèle de l'équation (3.6) et ne représenterait pas un comportement réel. Pour palier à cela, le couple de frottements à très basse vitesse sera calculé différemment. Nous introduisons une limite inférieure de vitesse ε_ω en dessous de laquelle le couple de frottements arrêtera complètement le mouvement si le couple appliqué (couple total excepté les termes de frottements) au moteur ou à la charge est inférieur en valeur absolue au couple de frottements. En effet, si le couple appliqué est inférieur en valeur absolue au couple de frottements, le calcul du couple total appliqué à la charge ou au moteur devient négatif comme si le couple de frottements engendrait un mouvement.

L'algorithme qui suit permet de prendre cela en considération et de faire en sorte que les phénomènes de frictions demeurent une contrainte sur le mouvement et non un phénomène générateur de mouvement. L'organigramme de cet algorithme est donné à la figure 3-3.

Les variables utilisées sont celles du moteur ou de la charge, τ_F désigne le couple de frottement, τ le couple de commande, ω la vitesse et ε_ω est la limite de vitesse inférieure qui sera fixée arbitrairement la plus faible possible et de façon à éviter les oscillations numériques ou autres problèmes dus à la discontinuité des fonctions en 0.

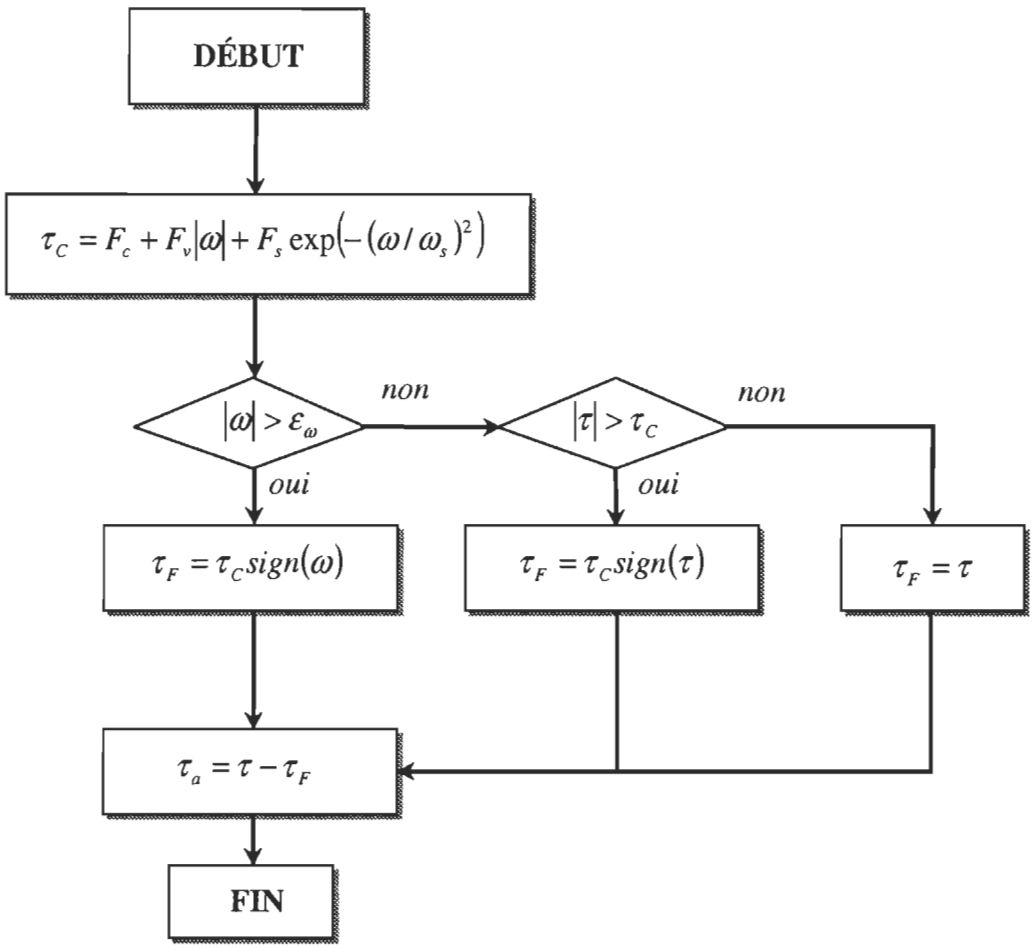


Figure 3-3 : Algorithme de calcul des couples de frottements

Cet algorithme sera utilisé pour calculer les couples de frictions aussi bien pour la charge que pour le moteur.

3.2.2.3. Modèle de la boîte de transmission

Dans cette étude, la boîte de transmission de l'articulation est supposée avoir une caractéristique linéaire exprimant le couple transmis à la charge comme le produit d'une constante par la déviation entre les positions de la charge et du moteur.

$$\tau_T = k(\theta_M - N\theta_L) \tag{3.7}$$

Ce modèle est analogue à celui exprimant la force exercée par un ressort de torsion sur une masse fixée à son extrémité.

3.2.2.4. Modèle de notre articulation flexible

En combinant les modèles du moteur, de la charge et de la boîte de transmission, le modèle de l'articulation peut être représenté par la figure 3-4. L'actionneur (moteur) et la charge sont affectés par des frottements de Coulomb, visqueux et statiques modélisés par l'équation (3.6). Dans les simulations, aucune action extérieure sur la charge ne sera envisagée. Cependant, la loi de commande sera conçue pour fonctionner même en présence d'un couple τ_e non nul, à condition d'être capable de quantifier cette grandeur.

3.2.3. Propriétés du modèle

Les valeurs numériques de *N*, η, *k*, ω_s et ε_ω sont supposées connues et constantes. La valeur de ε_ω doit être choisie la plus faible possible tout en permettant d'éviter que les forces de frottements ne génèrent de mouvement et sans créer d'oscillations à très basse vitesse.

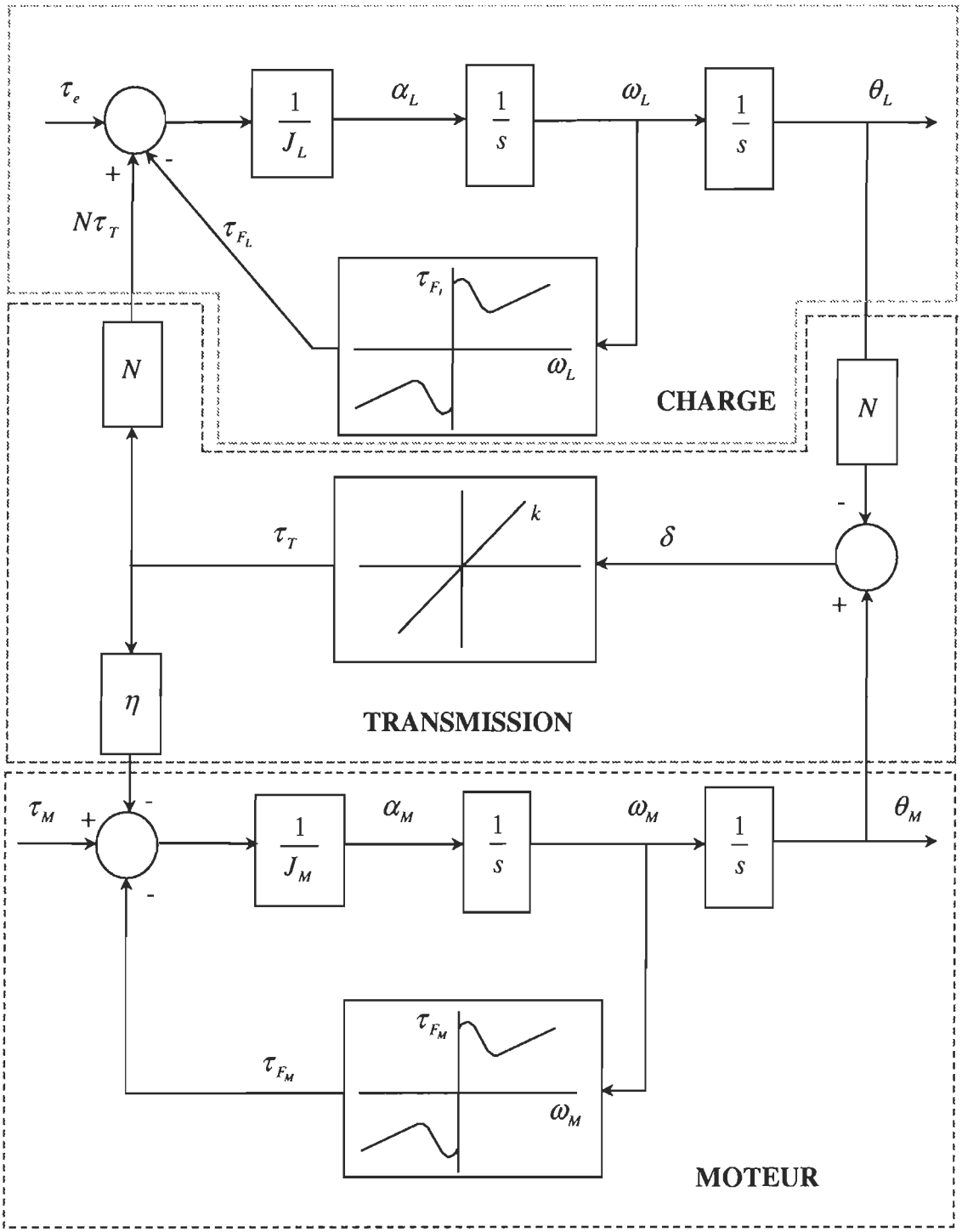


Figure 3-4 : Modèle de l'articulation flexible

Les inerties et les coefficients de friction sont les paramètres incertains du modèle de l'articulation. Les inerties peuvent varier de façon brusque, par exemple si la masse de la

charge fluctue brutalement ou encore si le manipulateur saisit un objet de son environnement. Les coefficients de frottements sont généralement considérés constants, nous choisirons dans cette étude des coefficients ayant des variations d'amplitudes peu importantes autour de leurs valeurs nominales. Les variations de tous ces paramètres sont supposées bornées, les tableaux 3-1 et 3-2 donnent toutes les valeurs numériques des paramètres de l'articulation. Nous supposerons les mêmes valeurs numériques pour les coefficients de frottements du moteur et de la charge mais ceci n'est nullement restrictif.

Tableau 3-1 : Valeurs numériques des paramètres N, k, η, ω_s, ε_ω

Paramètre	Valeur numérique
N	1.0
η	1.0
k [N.m/rad]	0.5
ω_s [rad/s^2]	9.5×10^{-2}
ε_ω [rad/s^2]	1.0×10^{-4}

Tableau 3-2 : Bornes des paramètres incertains de l'articulation

Paramètre	Borne inférieure	Borne supérieure
J_M [N.m/(rad/s^2)]	2.0×10^{-5}	1.0×10^{-3}
J_L [N.m/(rad/s^2)]	1.0×10^{-5}	2.0×10^{-3}
$F_{c_{M,L}}$ [N.m]	2.3×10^{-2}	3.4×10^{-2}
$F_{v_{M,L}}$ [N.m/(rad/s)]	1.5×10^{-2}	2.3×10^{-2}
$F_{s_{M,L}}$ [N.m]	2.0×10^{-2}	3.0×10^{-2}

3.3. Commande d'une articulation flexible

3.3.1. Décomposition du problème

Nous rappelons que le but recherché est de pouvoir faire suivre à la charge la trajectoire désirée en appliquant au moteur le couple de commande adéquat.

Parmi les méthodes de contrôle existantes, nous retrouvons très fréquemment les contrôleurs linéaires tels que les contrôleurs proportionnels (P), proportionnels dérivatifs (PD), proportionnels intégraux (PI), ou proportionnels intégraux dérivatifs (PID) qui sont les plus souvent utilisés. Le principe de ces contrôleurs est d'appliquer des gains sur les erreurs et leurs dérivées entre les sorties mesurées et les sorties de consigne. Ces méthodes sont couramment utilisées et fournissent de bons résultats pour les problèmes de contrôle de systèmes linéaires.

Cependant, lorsqu'il s'agit de contrôler des systèmes comprenant des non linéarités fortes, ces méthodes ne sont plus suffisantes. Par exemple, nous avons pu remarquer au deuxième chapitre que le modèle de friction comprend des termes fortement non linéaires. C'est pourquoi la compensation de friction ne peut en aucun cas être assurée par un de ces contrôleurs linéaires.

Toutefois, il est remarquable de constater à quel point un simple contrôleur proportionnel dérivatif peut assurer une stabilité asymptotique globale en l'absence de gravité et de friction [LEV96]. Lorsque nous considérons les non linéarités dues aux phénomènes de friction ou aux variations d'inertie, les contrôleurs linéaires ne peuvent garantir qu'une

stabilité locale [ZHI96]. Une loi de commande non linéaire offre alors l'avantage d'assurer une stabilité globale et de meilleures performances en utilisant moins d'énergie.

Deux contrôleurs seront conçus, un pour le moteur et un pour la charge, mais la similarité des modèles de la charge et du moteur nous amène finalement à concevoir une structure de contrôleur identique pour les deux en adaptant l'écriture des termes utilisés suivant si on considère la charge ou le moteur.

Cette structure de contrôleur comprendra donc deux boucles. Une boucle interne comprenant l'articulation et un correcteur linéaire de type proportionnel dérivatif, permettant de satisfaire les critères de stabilité locale et des caractéristiques désirées en boucle fermée telles que le dépassement nul, le temps de stabilisation à 5%, etc. Cette partie est couramment appelée *compensation standard.* La boucle externe comprendra un estimateur permettant de compenser les effets de friction et les variations de paramètres tels que les inerties et les coefficients de frottements.

Nous supposons que si ce contrôleur est capable de compenser efficacement les perturbations non linéaires, le correcteur linéaire proportionnel dérivatif peut assurer la stabilité globale. Cette structure de contrôleur à deux boucles est très fréquemment utilisée et correspond à la structure d'un régulateur auto-syntonisant vue au paragraphe 2.3.

3.3.2. Stratégie de commande

Nous supposons connue initialement la trajectoire désirée de la charge, i.e. les positions, vitesses et accélérations angulaires de consigne en fonction du temps [SEI92].

La stratégie de commande consiste alors, à partir de la trajectoire désirée pour la charge, à calculer le couple à appliquer au moteur pour obtenir cette trajectoire. La stratégie peut alors se décomposer en trois étapes, en parcourant le modèle de l'articulation de la charge vers le moteur (procédure dite « back stepping ») tel qu'illustré à la figure 3-5. Les trois étapes sont :

- Calcul du couple à transmettre à la charge pour lui permettre de suivre la trajectoire désirée,
- Calcul de la trajectoire désirée pour le moteur permettant d'appliquer à la charge le couple calculé à l'étape précédente,
- Calcul du couple à appliquer au moteur pour lui permettre de suivre la trajectoire désirée générée à l'étape précédente.

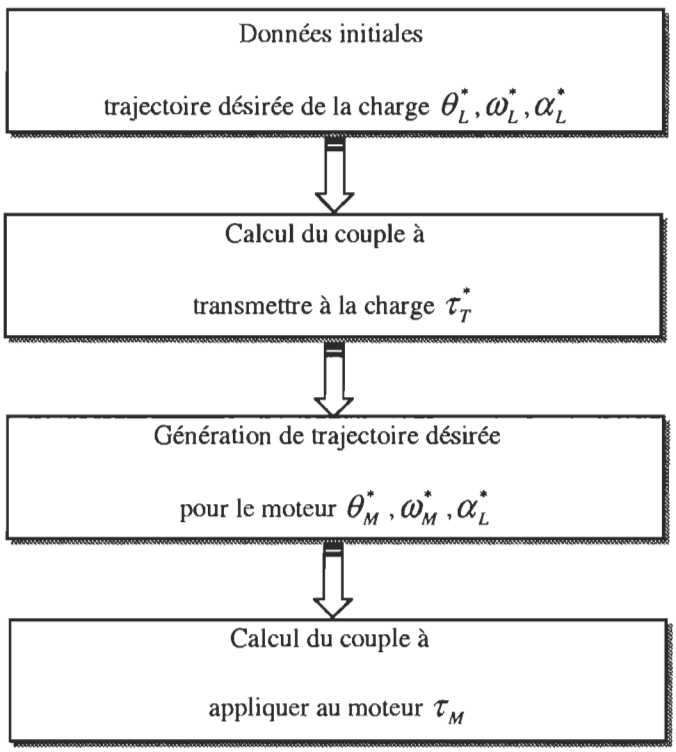


Figure 3-5 : Stratégie de commande utilisée

À partir de la trajectoire désirée de la charge, nous pouvons déduire le couple à transmettre à la charge d'après l'équations (3.4) du modèle et en ajoutant les termes de correction linéaire. Ainsi le couple transmis à la charge τ_T^* sera calculé par :

$$\tau_T^* = \frac{1}{N}\left[\hat{J}_L \alpha_L^* + \hat{\tau}_{F_L} + K_{V_L}\left(\omega_L^* - \omega_L\right) + K_{P_L}\left(\theta_L^* - \theta_L\right)\right] \tag{3.8}$$

θ_L^*, ω_L^* et α_L^* désignent respectivement les position, vitesse et accélération angulaires désirées de la charge. En pratique, l'accélération angulaire n'est pas mesurable ou très difficilement. Dans les simulations, nous supposerons que nous pouvons disposer d'une mesure bruitée de ce signal. Pour envisager l'emploi de notre loi de commande dans un cas pratique, il faudra développer une méthode performante d'approximation des accélérations de la charge et du moteur. Ces méthodes devront s'appuyer sur les données connues, principalement les vitesses et positions angulaires.

En supposant que l'équation (3.7) reste valide pour les valeurs désirées, l'équation (3.9) donne une autre expression du couple transmis désiré :

$$\tau_T^* = k\left(\theta_M^* - N\theta_L^*\right) \tag{3.9}$$

Cette équation nous permet maintenant de déduire la position désirée pour le moteur en fonction de la position désirée de la charge et du couple transmis désiré qui sont désormais supposés connus.

$$\theta_M^* = \frac{\tau_T^*}{k} + N\theta_L^* \tag{3.10}$$

Puisque nous désirons connaître la trajectoire complète du moteur, i.e. les positions, vitesses et accélérations angulaires, l'équation précédente peut être utilisée pour obtenir la vitesse et l'accélération désirée pour le moteur par dérivations successives :

$$\omega_M^* = \frac{d\theta_M^*}{dt} \tag{3.11}$$

$$\alpha_M^* = \frac{d\omega_M^*}{dt} \tag{3.12}$$

En injectant (3.10) dans (3.11) nous avons :

$$\omega_M^* = \frac{d}{dt}\left(\frac{\tau_T^*}{k} + N\theta_L^*\right) \tag{3.13}$$

L'expression de la vitesse angulaire désirée comprend donc deux termes. Le deuxième terme est facile à calculer puisqu'il s'agit, à une constante près, de la dérivée de la position désirée de la charge et donc de la vitesse désirée de la charge.

Le premier terme est plus complexe à évaluer car la dérivée du couple transmis τ_T^*, calculable à partir de (3.8), comprend les dérivées de plusieurs termes difficiles à évaluer telles que la dérivée du couple de frottements estimé ou encore la dérivée de l'accélération de la charge. En négligeant les termes d'ordres élevés dans le calcul des vitesses et accélérations du moteur [SIC93], il ne reste pour le calcul de la vitesse désirée du moteur que des termes simples à évaluer comme le montre l'équation (3.14).

$$\omega_M^* \cong \frac{d}{dt}\left(\frac{K_{V_L}\left(\omega_L^* - \omega_L\right) + K_{P_L}\left(\theta_L^* - \theta_L\right)}{k}\right) + \frac{d}{dt}\left(N\theta_L^*\right) \tag{3.14}$$

Finalement, l'expression de la vitesse désirée du moteur est :

$$\omega_M^* = N\omega_L^* + \frac{1}{k} K_{P_L}\left(\omega_L^* - \omega_L\right) + \frac{1}{k} K_{V_L}\left(\alpha_L^* - \alpha_L\right) \tag{3.15}$$

En procédant par la même méthode et en négligeant encore les termes appropriés, nous obtenons l'accélération désirée pour le moteur :

$$\alpha_M^* = N\alpha_L^* + \frac{1}{k} K_{P_L}\left(\alpha_L^* - \alpha_L\right) \tag{3.16}$$

Ainsi, la trajectoire du moteur est complète, il ne reste plus qu'à calculer le couple de commande permettant au moteur de suivre cette trajectoire. Le couple moteur est calculé sur la base de l'équation (3.2) en ajoutant les termes proportionnels et dérivatifs du correcteur linéaire de la même façon que pour la charge.

Ce couple est calculé par une expression analogue à celle du couple de charge :

$$\tau_M = \hat{J}_M \alpha_M^* + \hat{\tau}_{F_M} + \eta\tau_T^* + K_{V_M}\left(\omega_M^* - \omega_M\right) + K_{P_M}\left(\theta_M^* - \theta_M\right) \tag{3.17}$$

En appliquant ce couple au moteur, celui-ci suivra une trajectoire proche de sa trajectoire de consigne. En conséquence nous devrions obtenir le bon couple transmis à la charge pour que celle-ci suive également sa trajectoire de consigne. Ceci résume la stratégie globale de contrôle.

Remarques :

Le modèle de la boîte de transmission pourrait comprendre certains effets connus ayant fait l'objet de nombreux travaux de recherche tels que le retour de dents (backlash), l'hystérésis, etc. Dans ce travail, nous limitons le modèle de transmission à une constante de torsion, ce qui est toutefois une représentation courante.

Les positions et vitesses de la charge et du moteur sont des données supposées mesurables. L'accélération n'étant pas toujours facilement mesurable, nous pouvons soit considérer qu'elle est tout de même mesurable, soit utiliser une approximation calculée à partir de la vitesse. Par exemple, l'approximation d'Euler donne une valeur approchée de l'accélération instantanée en fonction des vitesses instantanées du même instant et de l'instant précédent.

$$\alpha_{(kT)} = \frac{\omega_{(kT)} - \omega_{((k-1)T)}}{T} \tag{3.18}$$

Où T est la période d'échantillonnage et k est un entier. Cette méthode peut donner des résultats peu satisfaisants si nous considérons que la vitesse est un signal bruité, ce qui est généralement le cas à moins d'utiliser la vitesse filtrée pour l'évaluation de (3.18).

La qualité de contrôle de l'articulation dépend de plusieurs conditions :

- Le modèle retenu pour l'articulation doit bien représenter son comportement dynamique,
- Les estimés des coefficients de friction et des inerties doivent être de bonne qualité,
- Les gains des correcteurs proportionnels dérivatifs doivent assurer un bon placement de pôles pour le moteur et la charge, et ainsi permettre de satisfaire les contraintes de suivi de trajectoire en terme de rapidité, d'erreur stationnaire, etc.

3.3.3. Génération de trajectoire pour la charge

Nous avons vu que l'objectif de commande de l'articulation flexible est de faire suivre à la charge une trajectoire désirée. Plus généralement, les robots sont utilisés pour réaliser des tâches telles que le déplacement d'objets, l'inspection de surfaces, etc. Pour réaliser une tâche, le robot doit effectuer un certain mouvement dans son espace de travail permettant à son effecteur de réaliser la tâche qui lui est assignée.

Pour un manipulateur à plusieurs articulations, nous définissons par trajectoire les variations dans le temps des positions, vitesses et accélérations de toutes les variables articulaires [CRA89]. Il arrive aussi que la trajectoire soit limitée aux positions et vitesses et même parfois aux positions seules.

Dans cette étude, nous ne considérons qu'une des articulations d'un manipulateur, la trajectoire de consigne sera donc complètement définie par les positions, vitesses et accélérations de la variable articulaire de cette articulation. Cette variable sera de type angulaire si l'articulation est rotative et linéaire si l'articulation est prismatique.

Chaque mouvement d'une articulation est toujours caractérisé par une position initiale et une position finale, le mouvement consiste alors à se déplacer du point initial au point final en un temps déterminé et avec une trajectoire dans l'espace déterminée. Il existe de nombreuses possibilités pour se déplacer d'une position à une autre, les mouvements rencontrés en robotique sont souvent des mouvements en ligne droite, circulaires, paraboliques , etc.

Les positions initiales et finales constituent les premières données pour la génération de la trajectoire. À partir de ces points, de nombreuses méthodes existent pour générer une trajectoire complète.

Les consignes de position de type échelon sont parfois utilisées, c'est-à-dire que l'on suppose que l'articulation se rend instantanément du point initial au point final. Ce type de consigne présente néanmoins le gros inconvénient de ne pas être réaliste puisque le caractère discontinu de la position implique des vitesses et accélérations articulaires très grandes et donc des couples de commande très élevés risquant d'endommager le manipulateur ou de créer des oscillations importantes. D'une manière générale les consignes avec des variations trop brusques sont à éviter, elles peuvent parfois être utiles en simulation pour observer le comportement d'une loi de commande lorsque les modes à hautes fréquences du système sont excités. Les trajectoires de consigne doivent donc de préférence être souples. Les méthodes de génération de trajectoire consistent le plus souvent à partir d'un profil pour l'accélération et à en déduire les profils de vitesse et position par intégration analytique.

Un type de trajectoire fréquemment utilisé est celui pour lequel la consigne de position est dite « en S » [CRA89]. C'est ce type de trajectoire qui sera utilisé dans nos simulations.

Nous supposerons que le manipulateur considéré doit réaliser un mouvement répétitif dans lequel il se déplace d'une position initiale nulle à une position finale unitaire. Cette

position finale est maintenue pendant une durée déterminée afin que l'effecteur puisse réaliser sa tâche et aussi pour amortir les oscillations éventuelles causées par l'arrêt du manipulateur. Enfin, le manipulateur revient par le même chemin jusqu'à sa position initiale.

Nous savons déjà que pendant la phase à position constante, les vitesses et accélérations sont nulles. Il reste à spécifier le type de trajectoire pendant les phases de mouvements.

La position pendant les phases de mouvements est de type « en S », c'est-à-dire que la courbe de la position articulaire désirée est constituée de morceaux paraboliques basés sur l'équation d'une parabole centrée sur l'origine de la forme :

$$\theta^*_{(t)} = at^2 \tag{3.19}$$

Dans ce cas, nous obtenons les vitesses et accélérations en dérivant simplement l'équation (3.19) :

$$\omega^*_{(t)} = \frac{d\theta^*_{(t)}}{dt} = 2at \tag{3.20}$$

$$\alpha^*_{(t)} = \frac{d^2\theta^*_{(t)}}{dt^2} = 2a \tag{3.21}$$

Les figures 3-6 à 3-8 donnent une représentation de la trajectoire complète de l'articulation pour une répétition du mouvement. Dans ce type de trajectoire, la position est définie par des morceaux de paraboles pendant les phases de mouvement. Les équations de ces morceaux de paraboles seront déduits de (3.20) et (3.21) par changement de repère. La forme de la trajectoire permet d'assurer d'une part un démarrage doux pour l'articulation et d'autre part une arrivée en douceur au point final.

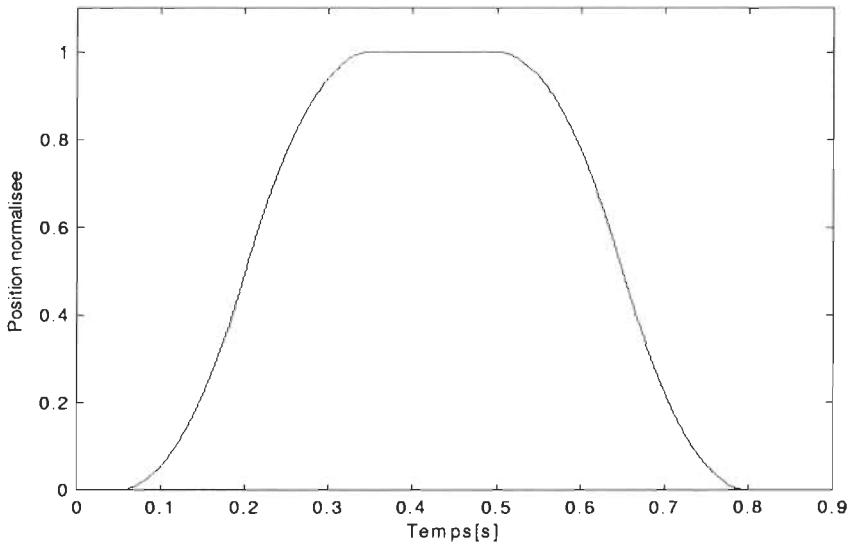


Figure 3-6 : Position articulaire désirée en fonction du temps, description du mouvement de base

Nous pourrons remarquer que ce profil de position entraîne une accélération de consigne discontinue. Dans certains cas il est préférable et plus réaliste d'imposer la contrainte de continuité sur l'accélération aussi.

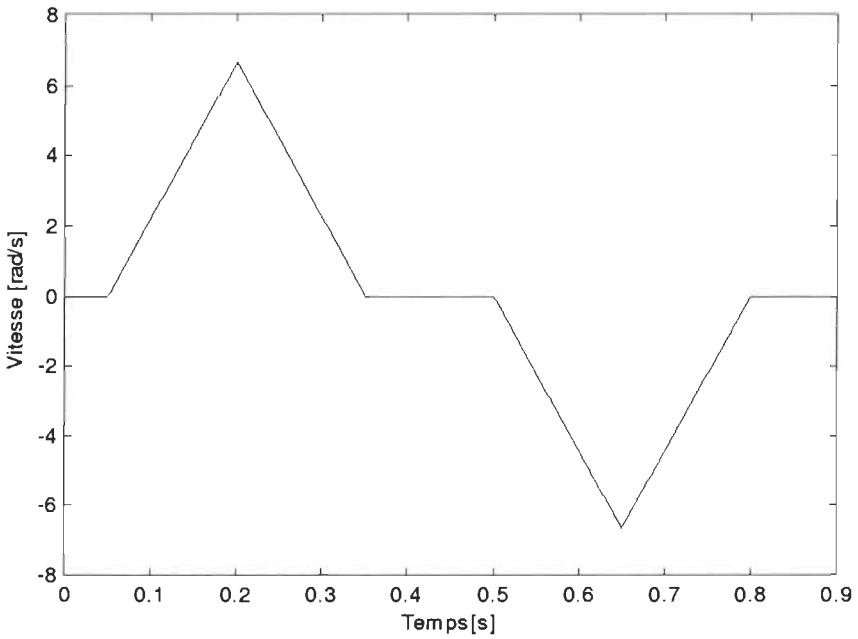


Figure 3-7 : Vitesse articulaire désirée en fonction du temps

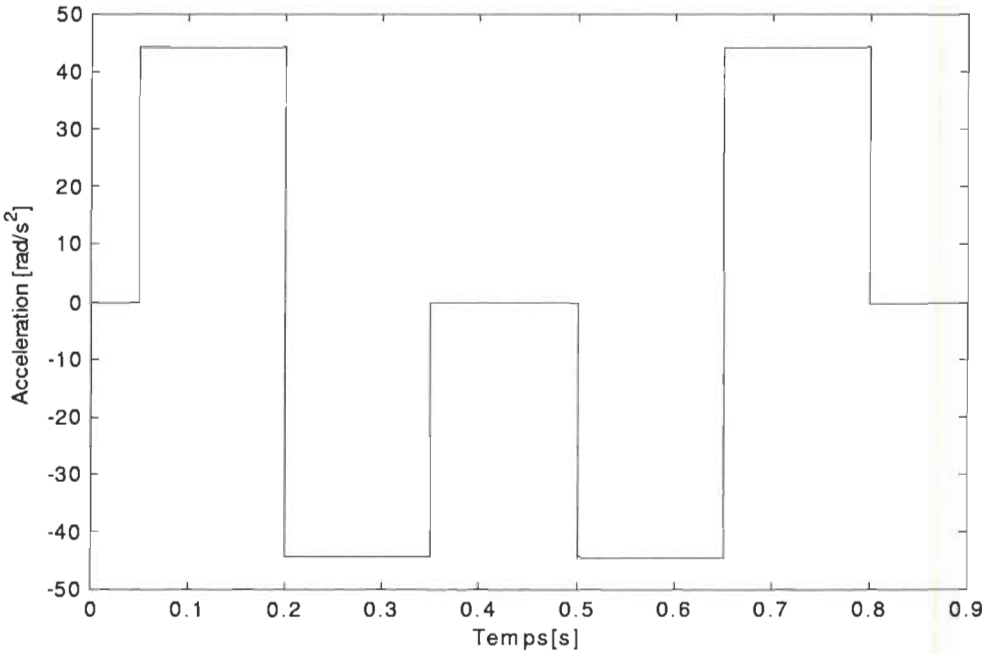


Figure 3-8 : Accélération articulaire désirée en fonction du temps

Nous imposons tout d'abord une phase de courte durée pendant laquelle le manipulateur est au repos. Ensuite, il y a une phase d'accélération constante positive de sorte que la position décrive un morceau de parabole, cette phase est suivie d'une décélération négative constante de valeur opposée à l'accélération précédente.

Cette première portion de la trajectoire permet à l'articulation d'atteindre sa position finale avec une vitesse et une accélération nulles, cette position sera maintenue pendant 0.05 seconde. Les mouvements contraires seront utilisés pour le retour à la position initiale.

Phase 1 : Repos, $t \in [0;0.05]$:

Cette phase est caractérisée par le repos du manipulateur. Durant cette phase les positions, vitesses et accélérations articulaires sont nulles.

$$\theta^*_{(t)} = \omega^*_{(t)} = \alpha^*_{(t)} = 0 \qquad (3.22)$$

Phase 2 : Mouvement du point initial jusqu'au point de mi parcours, t∈ [0.05;0.20] :

Ce mouvement est effectué à accélération positive constante et permet d'amener la position articulaire à mi parcours c'est-à-dire à la valeur normalisée 0.5. L'expression de la position sur cet intervalle est déduite des équations (3.19) à (3.21) par changement de repère puisque la base de la parabole n'est pas l'origine (0,0) mais le point (0.05,0).

Les équations de changement de repère sont :

$$\begin{cases} \theta^{*'} = \theta^{*} - \theta_0^{*} \\ t' = t - t_0 \end{cases} \tag{3.23}$$

Avec ce changement de repère la position peut être écrite sous la même forme que dans (3.19) mais avec les variables θ' et t' :

$$\theta^{*'} = a(t')^2 \tag{3.24}$$

Pour obtenir l'équation de la position θ^* en fonction du temps t nous utilisons les équations de (3.23) :

$$\theta^{*} - \theta_0^{*} = a(t - t_0)^2 \tag{3.25}$$

Cette équation nous permettra de trouver les expressions de tous les morceaux de paraboles de la trajectoire en connaissant simplement les coordonnées de la base de la parabole et en appliquant les formules de symétrie convenablement choisies.

Pour le premier intervalle, la base de la parabole est le point (0.05,0). Nous savons aussi que la valeur de la position en t = 0.20 est 0.5, en remplaçant dans (3.25) nous obtenons une équation du premier degré de la variable a d'où on déduit $a = (0.5-0)/(0.20-0.05)^2$. L'équation complète de la position articulaire sur cet intervalle de temps est donc finalement :

$$\theta^*_{(t)} = \frac{0.5}{(0.20-0.05)^2}(t-0.05)^2 \tag{3.26}$$

Nous obtenons ensuite les vitesses et accélérations de consigne en dérivant l'équation (3.26).

$$\omega^*_{(t)} = 2*\frac{0.5}{(0.20-0.05)^2}(t-0.05) \tag{3.27}$$

$$\alpha^*_{(t)} = 2*\frac{0.5}{(0.20-0.05)^2} \tag{3.28}$$

Phase 3 : Mouvement du point de mi parcours au point final, t∈ [0.20;0.35] :

Cette phase est caractérisée par une accélération négative constante permettant à l'articulation d'atteindre le point final.

Cette fois le point de base de la parabole est le point (0.35,1). Le coefficient de la parabole est égal en valeur absolue et opposé en signe à celui du tronçon précédent en raison de la symétrie de ces deux morceaux de trajectoires par rapport au point de mi parcours.

L'équation de la position articulaire sur ce tronçon est donc :

$$\theta^*_{(t)} = -\frac{0.5}{(0.35-0.20)^2}(t-0.35)^2 + 1 \tag{3.29}$$

Nous dérivons ensuite de nouveau pour obtenir les expressions des vitesses et accélérations.

$$\omega^*_{(t)} = -2 * \frac{0.5}{(0.35-0.20)^2}(t-0.35) \tag{3.30}$$

$$\alpha^*_{(t)} = -2 * \frac{0.5}{(0.35-0.20)^2} \tag{3.31}$$

Phase 4 : Stabilisation au point d'arrivée : t∈[0.35;0.50] :

Dans cette phase le manipulateur est au repos au point d'arrivée. Nous supposons que cette phase permet au manipulateur de se stabiliser au point d'arrivée et de réaliser une tâche. Pendant cette phase il n'y a donc pas de mouvement, la position de consigne demeure constante égale à 1 et les vitesses et accélérations sont nulles.

$$\theta^*_{(t)} = 1 \tag{3.32}$$

$$\omega^*_{(t)} = \alpha^*_{(t)} = 0 \tag{3.33}$$

Phase 5 : Retour au point de mi parcours, t∈[0.50;0.65] :

Durant cette phase, la position articulaire est ramenée jusqu'au point de mi parcours avec une accélération négative constante. Le point de base de la parabole de support de la

trajectoire est (0.50,1), nous en déduisons par la méthode précédente les équations de la trajectoire sur ce tronçon :

$$\theta^*_{(t)} = -\frac{0.5}{(0.65-0.50)^2}(t-0.50)^2 + 1 \tag{3.34}$$

$$\omega^*_{(t)} = -2*\frac{0.5}{(0.65-0.50)^2}(t-0.50) \tag{3.35}$$

$$\alpha^*_{(t)} = -2*\frac{0.5}{(0.65-0.50)^2} \tag{3.36}$$

Phase 6 : Déplacement du point de mi parcours jusqu'au point initial, t∈ [0.65;0.80] :

Cette dernière étape permet de ramener l'articulation à sa position initiale, ce mouvement est réalisé avec une accélération positive constante et les équations de la trajectoire sur cet intervalle sont :

$$\theta^*_{(t)} = \frac{0.5}{(0.80-0.65)^2}(t-0.80)^2 \tag{3.37}$$

$$\omega^*_{(t)} = 2*\frac{0.5}{(0.80-0.65)^2}(t-0.80) \tag{3.38}$$

$$\alpha^*_{(t)} = 2*\frac{0.5}{(0.80-0.65)^2} \tag{3.39}$$

Les six phases précédemment décrites caractérisent complètement la trajectoire désirée pour la charge de l'articulation et définissent un mouvement élémentaire qui sera répétitif.

3.3.4. Conception des contrôleurs

3.3.4.1. Conception des correcteurs linéaires

Comme nous l'avons vu dans les équations (3.8) et (3.17), les expressions des couples du moteur et de la charge comprennent des termes proportionnels sur les erreurs de positions et de vitesses angulaires. Les gains sont notés d'une façon générale K_p pour le gain sur l'erreur de position et K_v pour le gain sur l'erreur de vitesse, avec les indices M et L désignant respectivement les gains associés au moteur et la charge.

Pour trouver les gains de ce type de contrôleur, nous écrivons généralement une équation différentielle sur une erreur qui peut être par exemple l'erreur de position, et nous calculons alors les gains du contrôleur pour que cette équation admette une solution stable.

Pour cela, nous partons des équations (3.2) et (3.4) du modèle de l'articulation vu au paragraphe 3.2.1 :

$$J_M \alpha_M = \tau_M - \tau_{F_M} - \eta \tau_T \tag{3.40}$$

$$J_L \alpha_L = N \tau_T - \tau_{F_L} \tag{3.41}$$

Nous pouvons remplacer dans la première équation de ce système l'expression du couple moteur par celle de (3.17), il vient alors :

$$J_M \alpha_M = \hat{J}_M \alpha_M^* + \hat{\tau}_{F_M} + \eta \tau_T^* + K_{V_M}\left(\omega_M^* - \omega_M\right) + K_{P_M}\left(\theta_M^* - \theta_M\right) - \tau_{F_M} - \eta \tau_T \tag{3.42}$$

En supposant que les valeurs estimées des inerties et des coefficients de frottements sont de bonne qualité, nous pouvons poser :

$$\begin{cases} \hat{J}_M \approx J_M \\ \hat{\tau}_{F_M} \approx \tau_{F_M} \end{cases} \tag{3.43}$$

Ceci nous permet déjà de simplifier l'équation (3.42) sous la forme de l'équation (3.44) après passage à droite du terme $J_M \alpha_M$ et simplification.

$$0 = \hat{J}_M(\alpha_M^* - \alpha_M) + K_{V_M}(\omega_M^* - \omega_M) + K_{P_M}(\theta_M^* - \theta_M) + \eta(\tau_T^* - \tau_T) \tag{3.44}$$

Nous savons aussi que le couple transmis à la charge par la boîte de transmission peut être calculé en fonction des positions de la charge et du moteur comme nous l'avons vu dans l'équation (3.7) :

$$\tau = k(\theta_M - N\theta_L)$$

Nous supposerons que cette équation reste valide pour les valeurs désirées :

$$\tau_T^* = k(\theta_M^* - N\theta_L^*) \tag{3.45}$$

Les deux équations précédentes nous permettent de calculer la différence entre les couples transmis désiré et réel :

$$\tau_T^* - \tau_T = k(\theta_M^* - \theta_M) - Nk(\theta_L^* - \theta_L) \tag{3.46}$$

En injectant maintenant ce résultat dans l'équation (3.44) et en factorisant les termes appropriés nous obtenons presque l'équation d'erreur recherchée, il subsiste néanmoins un terme dépendant de l'erreur de position côté charge dû à la flexibilité :

$$0 = \hat{J}_M\left(\alpha_M^* - \alpha_M\right) + K_{V_M}\left(\omega_M^* - \omega_M\right) + \left(K_{P_M} + \eta k\right)\left(\theta_M^* - \theta_M\right) - \eta N k\left(\theta_L^* - \theta_L\right) \qquad (3.47)$$

Pour avoir directement une équation différentielle d'une erreur et calculer ses solutions, il faudrait que cette équation ne comporte que des termes dérivés d'une même erreur ce qui n'est pas le cas ici puisque l'erreur de position de la charge apparaît dans un des termes. Cela montre bien le couplage qui existe entre le moteur et la charge à cause de la flexibilité.

La même approche peut être employée pour tenter de déterminer une équation d'erreur du côté de la charge. L'expression du couple réel transmis à la charge apparaît dans l'équation (3.48) :

$$J_L\alpha_L = N\tau_T - \tau_{F_L} \qquad (3.48)$$

Le couple idéalement transmis à la charge devrait être tel que :

$$N\tau_T^* = \hat{J}_L\alpha_L^* + \hat{\tau}_{F_L} + K_{V_L}\left(\omega_L^* - \omega_L\right) + K_{P_L}\left(\theta_L^* - \theta_L\right) \qquad (3.49)$$

Que nous pouvons encore écrire :

$$\hat{J}_L\alpha_L^* = N\tau_T^* - \hat{\tau}_{F_L} - K_{V_L}\left(\omega_L^* - \omega_L\right) - K_{P_L}\left(\theta_L^* - \theta_L\right) \qquad (3.50)$$

La différence entre les équations (3.48) et (3.50) nous donne :

$$\hat{J}_L\alpha_L^* - J_L\alpha_L = N\left(\tau_T^* - \tau_T\right) + \tau_{F_L} - \hat{\tau}_{F_L} - K_{V_L}\left(\omega_L^* - \omega_L\right) - K_{P_L}\left(\theta_L^* - \theta_L\right) \qquad (3.51)$$

Ici encore nous supposons que les estimés des inerties et des coefficients de frottements de la charge sont de suffisamment bonne qualité pour permettre d'écrire :

$$\begin{cases} \hat{J}_L \approx J_L \\ \hat{\tau}_{F_L} \approx \tau_{F_L} \end{cases} \qquad (3.52)$$

Ceci permet de simplifier l'équation (3.51) en utilisant de plus l'équation (3.46) pour remplacer la différence entre le couple transmis idéal et le couple transmis réel. Finalement nous obtenons une équation de forme semblable à celle obtenue côté moteur :

$$\hat{J}_L\left(\alpha_L^* - \alpha_L\right) + K_{V_L}\left(\omega_L^* - \omega_L\right) + \left(K_{P_L} + N^2k\right)\left(\theta_L^* - \theta_L\right) - Nk\left(\theta_M^* - \theta_M\right) = 0 \qquad (3.53)$$

Cette fois encore, nous retrouvons le couplage entre le moteur et la charge dans l'équation d'erreur côté charge.

Nous pouvons, à partir des équations (3.47) et (3.53), reformuler notre problème sous la forme d'une équation d'état comme le montre l'équation (3.54). Pour assurer la stabilité de la charge et du moteur, il faut trouver les valeurs des gains $K_{P_M}, K_{V_M}, K_{P_L}, K_{V_L}$ permettant d'avoir des solutions stables pour ce système d'équations.

$$\begin{bmatrix} \alpha_L^* - \alpha_L \\ \alpha_M^* - \alpha_M \\ \omega_L^* - \omega_L \\ \omega_M^* - \omega_M \end{bmatrix} = \begin{bmatrix} -\frac{K_{V_L}}{\hat{J}_L} & 0 & -\frac{\left(K_{P_L} + N^2 k\right)}{\hat{J}_L} & \frac{Nk}{\hat{J}_L} \\ 0 & -\frac{K_{V_M}}{\hat{J}_M} & \frac{\eta N k}{\hat{J}_M} & -\frac{\left(K_{P_M} + \eta k\right)}{\hat{J}_M} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \omega_L^* - \omega_L \\ \omega_M^* - \omega_M \\ \theta_L^* - \theta_L \\ \theta_M^* - \theta_M \end{bmatrix} \quad (3.54)$$

Nous désirons que le moteur et la charge aient des réponses en boucle fermée satisfaisant à des critères particuliers tels que :

- Dépassement nul,
- Temps de réponse,
- ...

Pour cela nous devons d'abord définir quantitativement ces critères pour le moteur et pour la charge. Nous savons déjà que la réponse du moteur devra être plus rapide que celle de la charge pour que celui-ci réagisse assez rapidement et puisse positionner la charge dans les temps désirés. La flexibilité introduit ici une grande complexité puisque le système est du quatrième ordre. Pour placer les quatre racines de l'équation caractéristique du système (3.54) à des valeurs désirées, il faudrait être capable d'exprimer ces racines de façon analytique en fonction des différents gains recherchés, puis en déduire les valeurs des gains appropriées.

Cependant, il n'existe pas de méthode permettant d'exprimer de façon analytique les solutions d'une équation polynomiale de degré quatre, alors le problème est très complexe.

Dans ce projet, l'objectif n'est pas de mettre au point une méthode pour résoudre ce problème.

Pour le contourner, nous pouvons réduire sa complexité en considérant séparément la charge et le moteur. Nous rappelons que les contrôleurs proportionnels dérivatifs sont conçus pour assurer la stabilité de la charge et du moteur en supposant que les termes de frottements sont déjà compensés. Nous ne considérerons donc aucun frottement dans le calcul des gains.

Nous pouvons réécrire les équations d'erreurs (3.47) et (3.53) en supposant que :

- le moteur est plus rapide que la charge pour pouvoir réagir suffisamment rapidement aux mouvements de celle-ci. Ainsi, vu du côté moteur nous pouvons supposer que $\theta_L^* - \theta_L$ est constant.
- le moteur étant rapide, sa trajectoire converge très rapidement vers la trajectoire désirée et donc $\theta_M^* - \theta_M \approx 0$.

Ainsi, nous obtenons un nouveau système d'équations plus simples à résoudre :

$$\begin{cases} \hat{J}_M(\alpha_M^* - \alpha_M) + K_{V_M}(\omega_M^* - \omega_M) + (K_{P_M} + \eta k)(\theta_M^* - \theta_M) = c_1 \\ \hat{J}_L(\alpha_L^* - \alpha_L) + K_{V_L}(\omega_L^* - \omega_L) + (K_{P_L} + N^2 k)(\theta_L^* - \theta_L) = 0 \end{cases} \tag{3.55}$$

où c_1 est une constante.

Étant donné la similarité entre les équations d'erreurs côté charge et côté moteur, nous pouvons considérer une seule équation applicable pour les deux cas en supprimant les indices M et L :

$$\hat{J}(\alpha^* - \alpha) + K_V(\omega^* - \omega) + (K_P + a)(\theta^* - \theta) = c_2 \tag{3.56}$$

Où c_2 est une constante.

Or, en supposant négligeables les approximations commises dans le calcul de ω_M^* et α_M^*, les erreurs de vitesses et d'accélérations sont respectivement les dérivées premières et deuxièmes des erreurs de positions. Notons *e* l'erreur de position, nous avons :

$$e = \theta^* - \theta \tag{3.57}$$

L'équation (3.56) peut donc s'écrire aussi comme équation différentielle de la variable e :

$$\hat{J}\ddot{e} + K_V\dot{e} + (K_P + a)e = c_3 \tag{3.58}$$

Où c_3 est une constante.

Cette équation différentielle du deuxième ordre est plus simple à résoudre.

L'équation (3.58) peut être réécrite dans le domaine de Laplace :

$$\left(\hat{J}s^2 + K_V s + (K_P + a)\right)e = c_3 \tag{3.59}$$

Nous retrouvons ici une forme classique rencontrée en commande souvent écrite de la façon suivante :

$$\left(s^2 + 2\xi\omega_n s + \omega_n^2\right)e = c_3 \tag{3.60}$$

ξ est appelé coefficient d'amortissement, ω_n est appelée fréquence naturelle de résonance.

Dans un premier temps, nous considérons un amortissement critique, c'est-à-dire que le coefficient d'amortissement est égal à 1, dans ce cas l'équation (3.60) admet une solution double.

Les solutions de l'équation (3.60) sont les pôles du système (moteur ou charge) en boucle fermée. Ces pôles sont imposés au départ par les performances que l'on désire pour le système. Ces pôles seront doubles si nous choisissons l'amortissement critique.

Le moteur et la charge sont caractérisés principalement par leur rapidité de réponse qui est en général exprimée en terme de constante de temps. Si la constante de temps du moteur est t_M et si celle de la charge est t_L alors les pôles sont calculés directement comme étant les inverses opposés de ces constantes de temps. Soit p_M le pôle double pour le moteur et p_L celui de la charge, nous avons alors :

$$\begin{cases} p_M = -1/t_M \\ \\ p_L = -1/t_L \end{cases} \tag{3.61}$$

Nous pouvons alors écrire le polynôme caractéristique de l'équation d'erreur soit en utilisant la valeur des pôles, soit en utilisant la forme de (3.58). En égalisant les formes factorisées et développées du polynôme caractéristique de l'équation d'erreur côté moteur nous obtenons :

$$(s - p_M)^2 = s^2 + \frac{K_{V_M}}{\hat{J}_M} s + \frac{K_{P_M} + a}{\hat{J}_M} \tag{3.62}$$

L'égalisation des termes de même degré de ces polynômes permet de calculer les gains recherchés.

En remplaçant a par sa valeur suivant si nous considérons la charge ou le moteur, nous obtenons pour le calcul des gains côté moteur.

$$\begin{cases} K_{V_M} = -2 * \hat{J}_M p_M \\ K_{P_M} = p_M^2 \hat{J}_M - \eta k \end{cases} \tag{3.63}$$

De même nous obtenons les gains côté charge :

$$\begin{cases} K_{V_L} = -2 * \hat{J}_L p_L \\ K_{P_L} = p_L^2 \hat{J}_L - N^2 k \end{cases} \tag{3.64}$$

Si nous désirons une constante de temps de 10 ms pour le moteur alors les pôles pour le moteur sont :

$$p_M = -1/\left(10 * 10^{-3}\right) = -100$$

Le calcul des gains K_{P_M} et K_{V_M} dépend alors uniquement de la valeur estimée de l'inertie du moteur. Le contrôleur proportionnel dérivatif est donc en quelque sorte adaptatif puisque les gains sont adaptés à chaque changement de la valeur de l'inertie estimée. La charge quant à elle doit être plus lente en principe que le moteur, c'est-à-dire que les pôles devront être inférieurs en valeur absolue à ceux du moteur.

À partir de ces valeurs, il ne reste qu'à faire des simulations du système complet avec les gains calculés pour les contrôleurs proportionnels dérivatifs et sans friction pour voir quels sont les effets de la flexibilité sur le comportement du système.

D'après les résultats de ces simulations, nous pourrons déduire s'il faut augmenter la rapidité du moteur ou bien ajouter de l'amortissement sur la charge, etc. Nous

modifierons les valeurs des pôles en conséquence jusqu'à pouvoir obtenir de bonnes performances et limiter les effets de la flexibilité.

D'après les résultats de simulations obtenus, les pôles ont été fixés à :

$$p_M = -135$$
$$p_L = -50$$

Une fois que nous avons trouvé les valeurs des pôles permettant d'obtenir des résultats satisfaisants, nous fixons alors le calcul des gains des contrôleurs PD de la charge et du moteur par les équations (3.63) et (3.64).

Bien que cette méthode permette, pour la trajectoire de consigne choisie dans cette étude, de calculer des pôles assurant un bon comportement général de la loi de commande, elle n'en demeure pas moins très limitative. En effet, la trajectoire de consigne pourrait avoir une forme tout à fait quelconque et avoir un spectre fréquentiel plus étendu et la méthode que nous venons de voir ne permettrait plus alors de trouver des pôles assurant un bon suivi de trajectoire en absence de friction. Dans ce cas, une méthode plus générale tenant compte de la flexibilité devrait être établie. Nous devrons alors concevoir une méthode systématique permettant d'obtenir en ligne les valeurs des gains proportionnels et dérivatifs

3.3.4.2. Structure globale du contrôleur de l'articulation

Dans le paragraphe 3.3.2, nous avons vu que le calcul de la loi de commande pour le moteur nécessitait l'estimation des inerties et des couples de frottements de la charge et du moteur.

Le couple de charge peut être décomposé comme la somme de termes linéaires et non linéaires :

$$\tau_T^* = \tau_{T_l}^* + \tau_{T_{nl}}^* \tag{3.65}$$

Où :

$$\begin{cases} \tau_{T_l}^* = K_{P_L}\left(\theta_L^* - \theta_L\right) + K_{V_L}\left(\omega_L^* - \omega_L\right) \\ \tau_{T_{nl}}^* = \hat{J}_L \alpha_L^* + \hat{\tau}_{F_L} \end{cases} \tag{3.66}$$

De même, le couple de commande du moteur peut être décomposé comme suit :

$$\tau_M = \tau_{M_l} + \tau_{M_{nl}} \tag{3.67}$$

Où :

$$\begin{cases} \tau_{M_l} = K_{P_M}\left(\theta_M^* - \theta_M\right) + K_{V_M}\left(\omega_M^* - \omega_M\right) \\ \tau_{M_{nl}} = \hat{J}_M \alpha_M^* + \hat{\tau}_{F_M} + \eta \tau_T^* \end{cases} \tag{3.68}$$

Ainsi, la structure du contrôleur de l'articulation peut être décomposée comme illustré par la figure 3-9. Cette structure permet de mettre en évidence la stratégie de commande, c'est-à-dire dans un premier temps le calcul du couple de charge idéal, dans un deuxième temps le calcul de la trajectoire de consigne pour le moteur, et enfin le calcul du couple de commande du moteur à partir de cette trajectoire. Nous pouvons déjà facilement remarquer que cette structure peut se décomposer en deux parties dont les formes sont très similaires ce qui facilitera l'implantation. La différence entre le deux blocs se situera au niveau de l'entrée de la trajectoire désirée. Côté charge, la trajectoire désirée est une entrée externe connue, généralement imposée par l'opérateur en fonction de la tâche à réaliser, alors que pour le moteur, la trajectoire de consigne est calculée à l'intérieur de la loi de commande. Nous devrons donc ajouter dans une des deux parties de l'architecture, les éléments nécessaires à la génération de trajectoire pour le moteur.

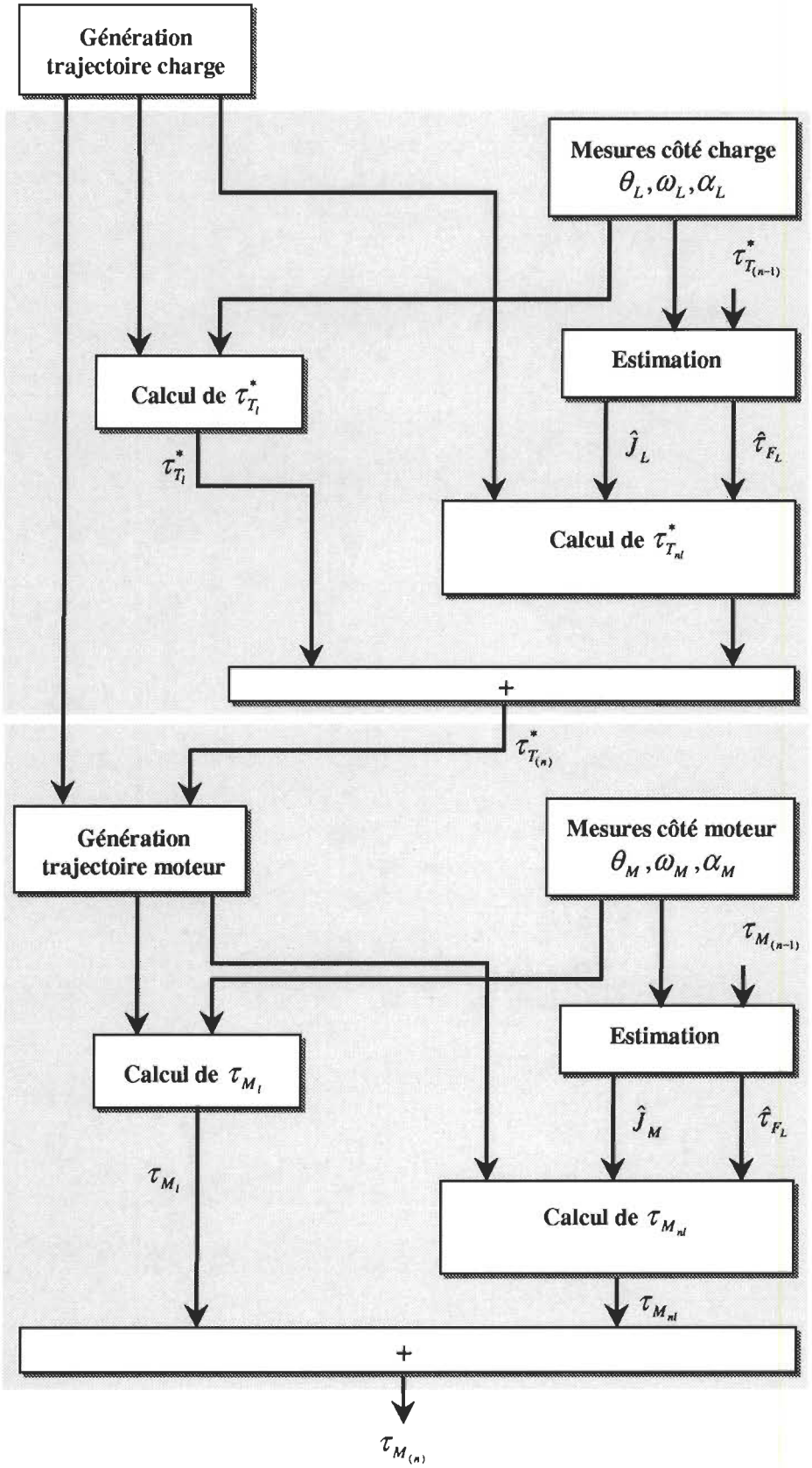


Figure 3-9 : Structure du contrôleur de l'articulation

3.3.5. Simulations – Résultats

3.3.5.1. Cas idéal

Dans un premier temps, les simulations ont été réalisées en supposant tous les paramètres de l'articulation connus afin d'évaluer les performances de la loi de commande développée. Les figures 3-12 et 3-14 représentent les erreurs de positions et vitesses réelles obtenues pour la charge par rapport aux valeurs désirées illustrées par les figures 3-11 et 3-13.

Dans ces simulations, nous avons considéré une inertie constante pour le moteur et égale à 3.6×10^{-4}. L'inertie de la charge varie comme illustré à la figure 3-10. L'inertie du moteur est considérée constante et égale à 3.6×10^{-4}. Nous conserverons ces caractéristiques pour les inerties dans toutes les simulations. Les valeurs numériques des autres paramètres ont été données dans les tableaux 3-1 et 3-2. Dans la suite, les erreurs représentées sont les différences entre les valeurs désirées et réelles ou entre les valeurs réelles et estimées dans le cas d'une erreur d'estimation

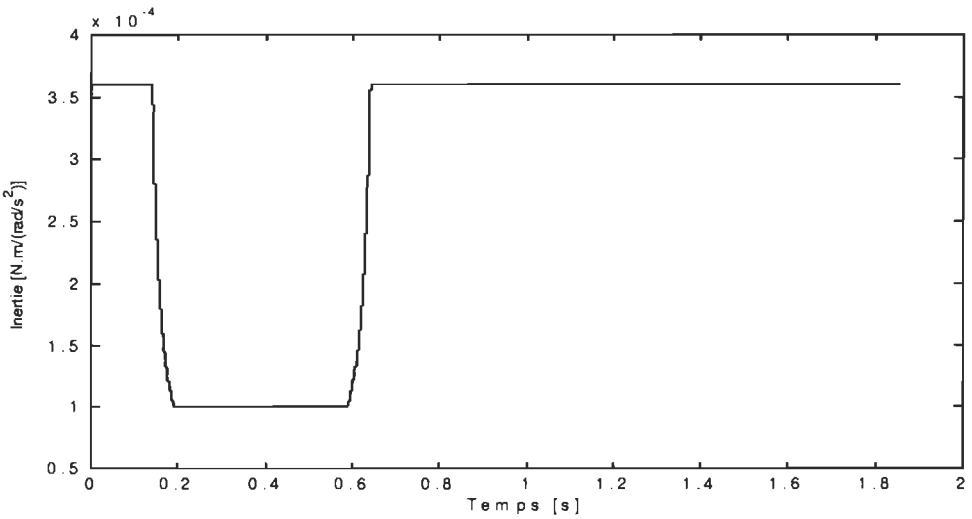


Figure 3-10 : Variations d'inertie de la charge

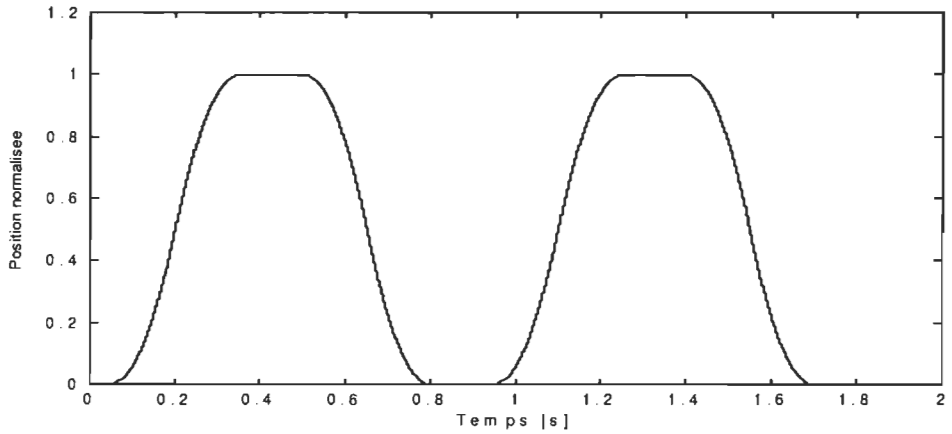


Figure 3-11 : Position désirée de la charge

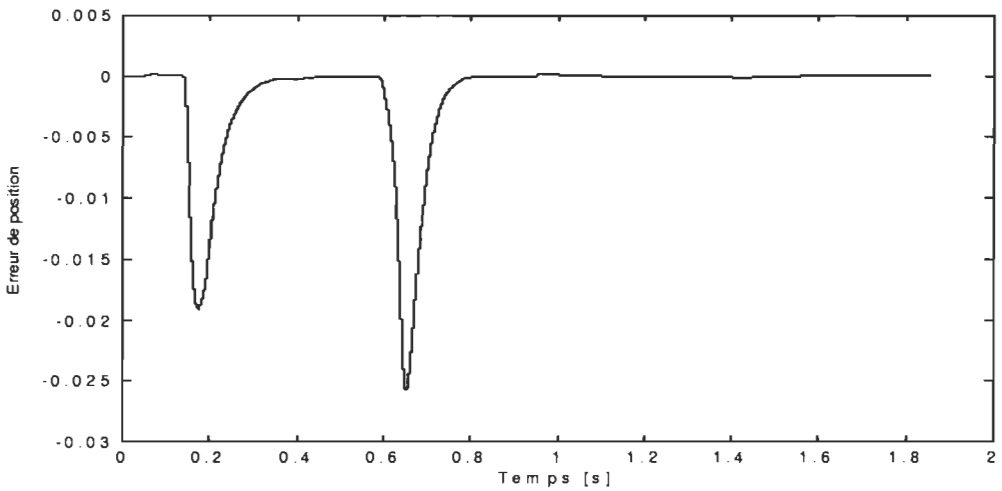


Figure 3-12 : Erreur de position de la charge

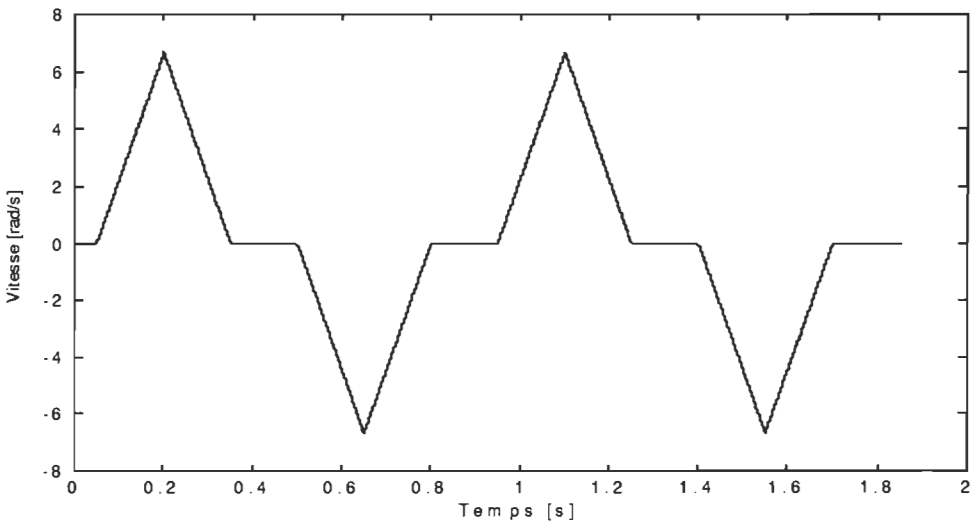


Figure 3-13 : Vitesse désirée de la charge

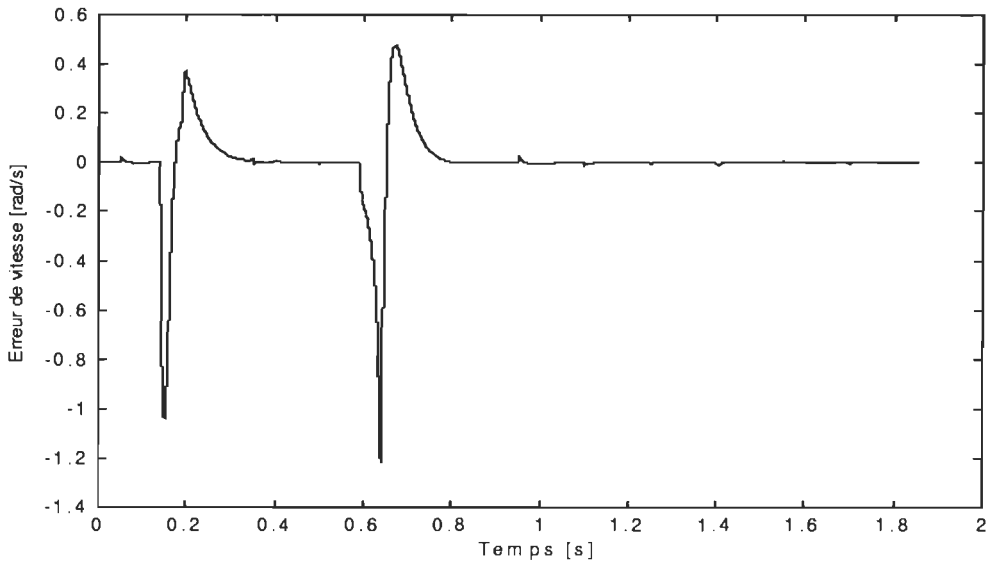


Figure 3-14 : Erreur de vitesse de la charge

Nous pouvons constater que les erreurs de positions et de vitesses augmentent lorsque l'inertie de la charge varie. Ces erreurs se stabilisent lorsque l'inertie de la charge redevient constante. Plusieurs causes peuvent être à l'origine de ce phénomène :

- le couple ne peut être transmis instantanément à la charge à cause de la dynamique de transmission de l'articulation,
- l'approximation faite pour le placement de pôles donne des pôles légèrement erronés dans les phases de variations d'inertie

3.3.5.2. Effets des couples de frottements

Pour évaluer les effets des couples de frottements, des simulations ont été réalisées en n'appliquant aucune compensation de frottements. Les courbes de position et vitesse de la

charge sont données dans les figures 3-15 et 3-16. La courbe en pointillés représente la courbe désirée comme ce sera le cas pour toute la suite du mémoire.

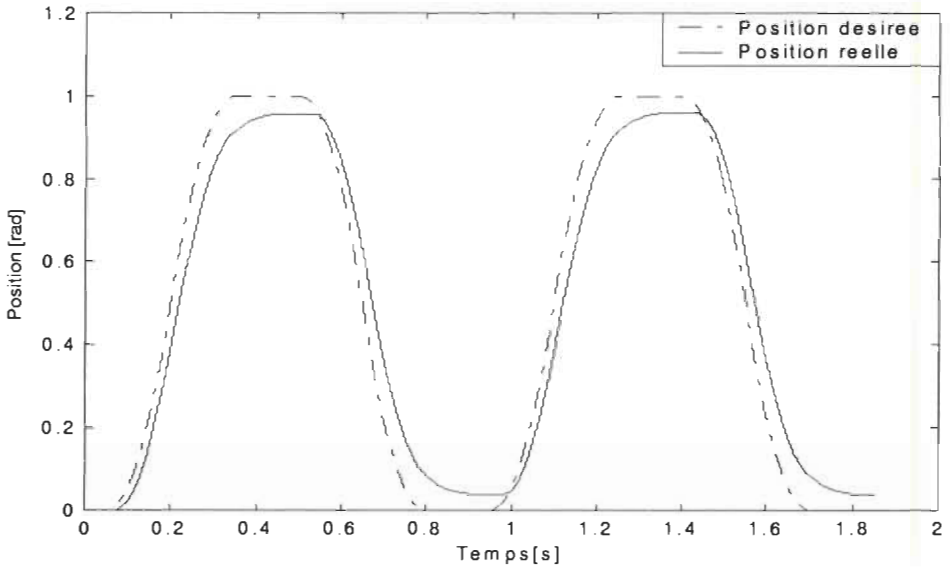


Figure 3-15 : Effets des frottements sur la réponse en position

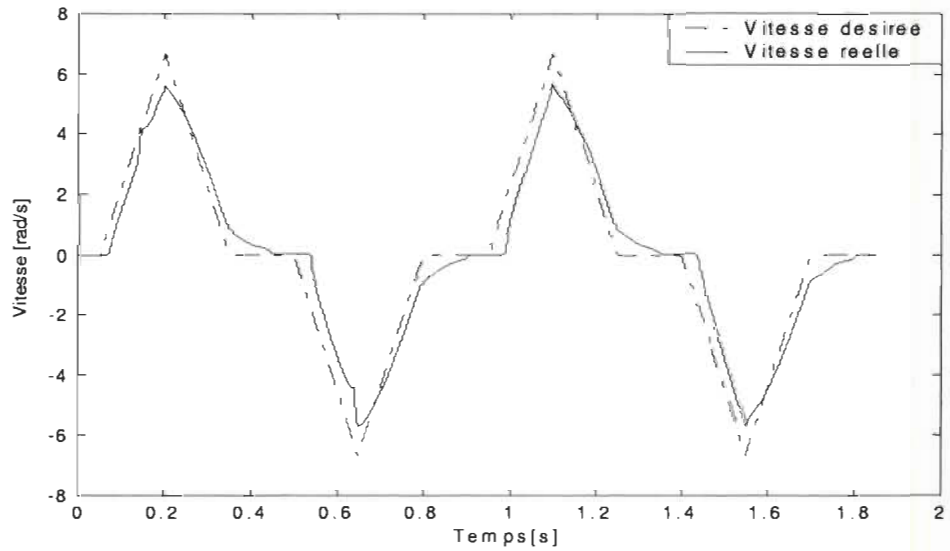


Figure 3-16 : Effets des frottements sur la réponse en vitesse

Les figures 3-17 et 3-18 présentent l'évolution des erreurs absolues de position et de vitesse avec et sans frottements.

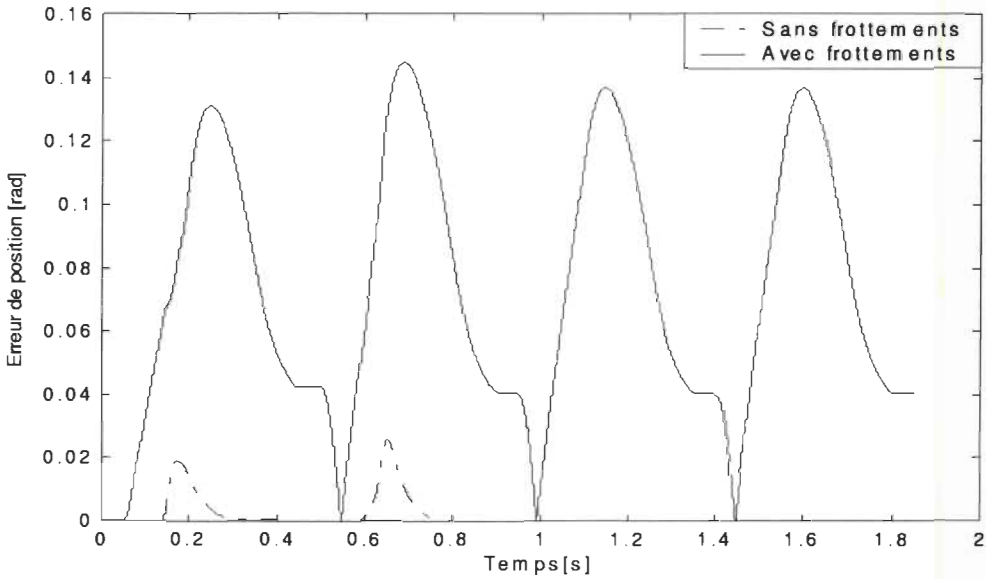


Figure 3-17 : Effets des frottements sur l'erreur absolue de position.

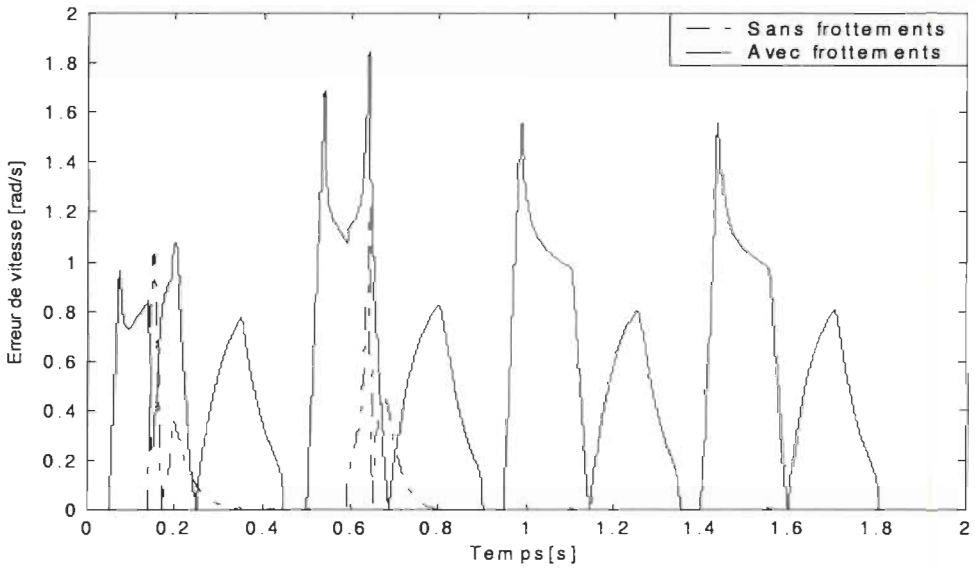


Figure 3-18 : Effets des frottements sur l'erreur absolue de vitesse

Ces courbes nous permettent de constater l'importance des forces de frottements qui peuvent engendrer une erreur absolue de position allant jusqu'à 15%.

Le tableau 3-3 donne les erreurs absolues moyennes obtenues sur les positions et vitesses avec et sans compensation de friction. La compensation de friction ici est idéale, c'est-à-dire que nous avons injecté un couple de compensation toujours rigoureusement égal au couple de friction. La compensation de friction réduit par un facteur de l'ordre de 10 l'erreur absolue moyenne sur les positions et vitesses.

Tableau 3-3 : Erreurs absolues moyennes de position et vitesse avec et sans compensation de friction

Signal	Erreur absolue moyenne
Position sans compensation de friction	7.8×10^{-2}
Position avec compensation de friction idéale	1.6×10^{-3}
Vitesse sans compensation de friction	5.7×10^{-1}
Vitesse avec compensation de friction idéale	4.9×10^{-2}

3.4. Conclusion

Les simulations du modèle complet de l'articulation flexible avec son contrôleur ont été réalisées dans Simulink™, les schémas et programmes sont présentés en annexe C.

Les simulations réalisées permettent tout d'abord de constater que la loi de commande développée est performante et nous permet de satisfaire à nos objectifs de commande visés. Nous pouvons constater par les résultats obtenus que le suivi de trajectoire est d'excellente qualité. La position réelle de l'articulation suit la position désirée et il n'y a ni dépassement ni oscillations tant durant les phases de mouvements que dans les phases de repos.

Sur les figures 3-11 à 3-14, nous pouvons remarquer les effets de la flexibilité principalement lors des variations de l'inertie de la charge. De plus, les simulations sans compensation de friction ont montré quelles pouvaient être les conséquences de ces phénomènes sur les performances de commande, ce qui justifie l'utilisation d'une méthode de compensation. En effet, nous voyons sur les figures 3-15 et 3-16 que les frottements ralentissent le mouvement de la charge et que celle-ci n'atteint jamais sa position finale de consigne ce qui serait très gênant en pratique.

Nous pouvons déjà conclure à cette étape que si nous pouvons connaître avec suffisamment de précision les paramètres de l'articulation, alors les performances de la loi de commande seront préservées.

Chapitre 4

Estimation des paramètres de l'articulation

4.1. Introduction

L'identification des paramètres d'un système est un problème important dans la commande adaptative [ROL82]. Beaucoup de recherches ont été menées dans le passé sur la commande adaptative, mais souvent limitées dans leurs applications par les possibilités des calculateurs numériques. Le développement de calculateurs numériques performants et à faible coût a grandement réactivé le champ de d'application de la commande adaptative.

Par définition, un contrôleur adaptatif d'un procédé est un système capable d'adapter son comportement aux propriétés du procédé qu'il contrôle et de ses signaux.

Il s'applique donc aux systèmes dont les paramètres sont inconnus ou variants dans le temps [AST87].

Les contrôleurs adaptatifs peuvent être classifiés en deux groupes : contrôleurs adaptatifs directs et indirects.

Dans un contrôleur adaptatif indirect, les paramètres du procédé sous contrôle sont explicitement identifiés et utilisés dans la conception du contrôleur.

Dans un contrôleur adaptatif direct, il n'y a pas d'identification explicite des paramètres du procédé, mais les gains du contrôleur sont directement adaptés en fonction de la réponse du procédé.

Très souvent, ces contrôleurs sont implantés dans des calculateurs numériques car ceux-ci permettent une facilité de traitement et des performances bien supérieures aux implantations analogiques réalisées autrefois. Le choix de la fréquence d'échantillonnage est une première étape importante, celle-ci ne devra pas être choisie trop faible au risque de pertes d'informations et pas trop élevée pour ne pas travailler en dehors de la bande passante du système contrôlé en boucle fermée.

Dans ce projet, un contrôleur adaptatif indirect est conçu pour le contrôle d'une articulation flexible. Le contrôleur est adapté en fonction des résultats d'identification des paramètres du procédé et des erreurs entre les signaux de consigne et les mêmes signaux mesurés. La figure suivante donne la représentation générale d'un tel schéma de contrôle.

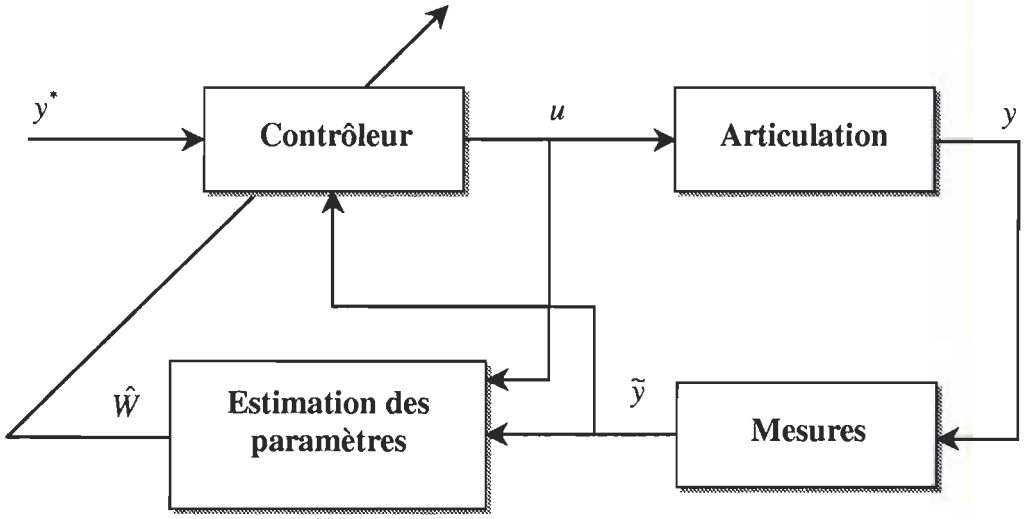


Figure 4-1 : Schéma général du contrôleur adaptatif indirect

Avec :

- y^* le vecteur des signaux de consigne,
- y le vecteur des sorties mesurables du système,
- $\tilde{y}$ le vecteur des résultats de mesures,
- u la commande du système,
- $\hat{W}$ le vecteur des grandeurs estimées du système.

4.2. Définition du problème d'estimation

4.2.1. Paramètres à estimer

Le contrôle de précision de systèmes mécaniques en présence de friction est une tâche difficile. Les différents coefficients mis en jeu dans les modèles de friction sont en général difficilement mesurables expérimentalement et peuvent de plus varier dans le temps. C'est pourquoi l'emploi de méthodes adaptatives pour la compensation des phénomènes de friction est justifiée [CAN96].

D'autre part, l'inertie du moteur et de la charge dans une articulation flexible sont des paramètres qui sont, à priori, inconnus et qui peuvent varier dans le temps, ce qui justifie cette fois encore l'utilisation de méthodes adaptatives pour l'identification de ces paramètres.

Lors de la conception de la loi de commande au deuxième chapitre les équations des couples désirés du moteur et de la charge ont été données :

$$\tau_T^* = \frac{1}{N}\left[\hat{J}_L \alpha_L^* + \hat{\tau}_{F_L} + K_{V_L}\left(\omega_L^* - \omega_L\right) + K_{P_L}\left(\theta_L^* - \theta_L\right)\right] \tag{4.1}$$

$$\tau_M = \hat{J}_M \alpha_M^* + \hat{\tau}_{F_M} + \eta \tau_T^* + K_{V_M}\left(\omega_M^* - \omega_M\right) + K_{P_M}\left(\theta_M^* - \theta_M\right) \tag{4.2}$$

Dans ces équations, nous pouvons remarquer que la connaissance des grandeurs estimées $\hat{\tau}_{F_L}$, $\hat{J}_L$, $\hat{\tau}_{F_M}$ et $\hat{J}_M$ est requise. $\hat{\tau}_{F_L}$ et $\hat{\tau}_{F_M}$ désignent respectivement les couples de friction totaux estimés côté charge et côté moteur. D'après le modèle retenu pour ces grandeurs, la connaissance du couple de friction côté charge ou côté moteur revient à la connaissance des trois paramètres de friction du modèle et de la vitesse angulaire instantanée.

Ainsi, côté moteur les coefficients de frottement statique, visqueux et de Coulomb devront être estimés, nous les notons respectivement $\hat{F}_{s_M}$, $\hat{F}_{v_M}$ et $\hat{F}_{c_M}$, et il en sera de même côté charge avec les paramètres $\hat{F}_{s_L}$, $\hat{F}_{v_L}$ et $\hat{F}_{c_L}$.

En ajoutant à ces paramètres les inerties du moteur et de la charge qui sont aussi à estimer, cela nous donne huit paramètres à estimer, dont quatre côté moteur et quatre coté charge.

4.2.2. Propriétés des paramètres à estimer

Les différents paramètres à estimer sont à priori incertains, c'est-à-dire qu'on ne peut déterminer à l'avance leur valeur. Au niveau des algorithmes permettant l'identification de paramètres on retrouve très souvent des algorithmes récursifs [AST95] [ROL82] [SAL88]. La forme récursive de ces algorithmes nécessite de donner une valeur initiale à chacun des paramètres.

Cette valeur initiale peut être choisie de manière aléatoire si aucune information sur les valeurs réelles des paramètres n'est disponible, mais dans certains cas nous pouvons avoir une idée approximative de ces valeurs initiales réelles ou au moins en connaître un ordre de grandeur.

Dans le cas des coefficients de frottements par exemple, ceux-ci seront supposés varier constamment dans le temps, mais de façon lente et avec une amplitude de variations faible autour d'une valeur nominale. Pour initialiser les coefficients estimés de friction, nous supposons alors disposer d'une idée sur l'ordre de grandeur des coefficients réels qui nous permettra de fixer des valeurs initiales. Cependant, nous montrerons que les algorithmes d'estimation utilisés fonctionnent très bien en choisissant des valeurs nulles pour valeurs initiales.

L'inertie quant à elle est un paramètre qui sera supposé avoir des variations peu fréquentes dans le temps mais brusques et d'amplitudes importantes.

Par exemple, lorsque le manipulateur va accrocher une charge dans son mouvement, l'inertie de la charge sera grandement augmentée. Le phénomène inverse se produit lorsque le manipulateur lâche la charge et qu'il se retrouve à vide. L'inertie du moteur peut être considérée constante.

Initialement nous supposons que le robot est à vide et que l'on peut connaître cette fois encore un ordre de grandeur pour la valeur nominale des inerties côté charge et côté moteur nous permettant de fixer des valeurs initiales pour ces paramètres.

Ces propriétés sur les paramètres à estimer seront importantes dans le choix de l'algorithme d'identification et des valeurs des paramètres de réglage de celui-ci [AST95].

4.3. Choix d'une méthode d'estimation

Puisque l'identification des paramètres est réalisée pour un système adaptatif, il est important d'avoir une bonne compréhension de tous les aspects du problème [AST95].

Le choix du modèle et des paramètres est crucial. Le problème d'identification est grandement simplifié lorsqu'il existe un modèle d'un signal connu linéaire vis-à-vis des paramètres recherchés, ce qui sera le cas dans cette étude.

Les algorithmes d'identification de paramètres sont nombreux, chacun adapté à des conditions d'utilisation particulières et des niveaux de performances particuliers. Parmi ceux-ci, certains sont récursifs et sont, de par leur nature récursive, bien adaptés aux problèmes d'identification en ligne en temps réel et également au développement d'architectures pour l'implantation.

Les deux points précédemment évoqués, i.e. la linéarité du modèle vis-à-vis des paramètres et la préférence pour un algorithme récursif ont amené le choix vers un algorithme basé sur la méthode des moindres carrés.

Plus particulièrement nous nous intéresserons à l'algorithme des moindres carrés récursif (RLS) dont les études comparatives montrent qu'il offre une meilleure rapidité de convergence que la plupart des autres méthodes au coût toutefois d'avoir une complexité de calcul plus élevée [MIC92] [HAY91].

Karl Friedrich Gauss formula le principe des moindres carrés à la fin du dix-huitième siècle pour déterminer les orbites des planètes. Il établit que les paramètres inconnus à identifier devraient être choisis en considérant que :

« *La somme des carrés des différences entre les valeurs actuellement observées et les valeurs calculées multipliées par des nombres représentant le degré de précision est un minimum* » [AST95].

Soit une sortie mesurable y d'un procédé et reliée aux paramètres à déterminer par une équation linéaire vis-à-vis de ces paramètres telle que :

$$y_{(n)} = \boldsymbol{\Phi}^{T}_{(n)}\boldsymbol{\theta}_{(n-1)} \tag{4.3}$$

Où $\boldsymbol{\Phi}$ désigne le vecteur de régression, $\boldsymbol{\theta}$ le vecteur des paramètres à identifier et n l'instant d'échantillonnage. Nous verrons plus tard qu'il faudra ajouter à ce modèle le bruit de mesure et nous verrons comment le prendre en compte dans les équations

d'estimation. La méthode des moindres carrés vise alors à estimer les paramètres de façon à minimiser le critère des moindres carrés :

$$V(\boldsymbol{\theta},n)=\frac{1}{2}\sum_{i=1}^{n}\left(y(i)-\boldsymbol{\Phi}_{(i)}^{T}\boldsymbol{\theta}_{(i-1)}\right)^{2} \tag{4.4}$$

Dans la méthode des moindres carrés récursive, il y deux équations de mise à jour, la première pour les paramètres estimés, la seconde pour la matrice de covariance.
De nombreuses versions de la méthode RLS ont été élaborées afin de lui permettre d'être fonctionnelle face à certaines contraintes [SAL82].

La méthode RLS classique est reconnue pour avoir les propriétés optimales lorsque les paramètres à estimer sont invariants. Cependant, cette méthode ne s'applique plus aux problèmes d'identification de paramètres variants dans le temps puisque le gain de l'algorithme converge vers 0 après un certain nombre d'itérations et ne lui permet plus par la suite de trouver de nouvelles valeurs pour les paramètres estimés.

La version modifiée la plus connue de cette méthode est basée sur l'utilisation de poids exponentiels sur les erreurs successives pour calculer le critère à minimiser. Ces poids sont calculés comme puissances entières croissantes d'un coefficient appelé facteur d'oubli, qui permet à l'algorithme « d'oublier » le minimum qu'il a trouvé pour rechercher de nouvelles valeurs de paramètres [SAL88].
Le nouveau critère minimisé par la méthode s'écrit alors :

$$V(\boldsymbol{\theta},n)=\frac{1}{2}\sum_{i=1}^{n}\lambda^{n-i}\left(y(i)-\boldsymbol{\Phi}_{(i)}^{T}\boldsymbol{\theta}_{(i-1)}\right) \tag{4.5}$$

Cependant, l'utilisation de cette méthode peut donner lieu à des divergences importantes des coefficients de la matrice de covariance quand l'information contenue dans le régresseur n'est pas assez riche, d'autres modifications supplémentaires peuvent alors être utilisées pour éviter ces phénomènes [SAL88]. L'utilisation du facteur d'oubli donne de bonnes performances lorsque les paramètres ont des variations temporelles continues mais lentes [HAY95].

Pour le cas de variations peu fréquentes mais brusques comme c'est le cas pour le paramètre d'inertie, une solution peut être de réinitialiser la matrice de covariance périodiquement à une valeur de αI où α est une constante positive et I la matrice identité. Cette méthode est connue dans la littérature sous le nom de « Resetting » [ADA95] et n'est efficace que si nous savons à l'avance à quels moments les paramètres varient afin de réinitialiser la matrice de covariance à ces instants.

4.4. Utilisation de la méthode RLS pour le cas étudié

4.4.1. Formulation

Nous devons maintenant trouver comment appliquer la méthode RLS d'estimation des paramètres au problème posé dans cette recherche.

Étant donné la similarité des structures côté charge et côté moteur, l'identification sera effectuée par la même méthode. Pour éviter les répétitions, les équations seront écrites sous une forme générale permettant de les associer soit au moteur soit à la charge, en précisant simplement les indices adéquats.

Les quatre paramètres recherchés sont :

- L'inertie J,
- Le coefficient de frottement de Coulomb F_c,
- Le coefficient de frottement visqueux F_v,
- Le coefficient de frottement statique F_s.

Seidl formula la méthode d'identification des paramètres d'inertie et de friction en utilisant comme signal la position mesurée [SEI92]. Cependant, ce signal ne traduit pas suffisamment bien le comportement dynamique du système pour permettre une bonne qualité d'identification. De plus, dans sa formulation, le vecteur des paramètres à estimer contient plusieurs constantes ce qui, d'après les résultats de simulations, peut altérer la convergence de l'algorithme.

Pour résoudre notre problème d'estimation, nous adopterons la même approche, mais c'est l'accélération qui sera le signal utilisé pour l'identification. Ce signal n'est en général pas mesurable mais nous supposerons connaître une bonne approximation de cette grandeur.

Nous rappelons les équations du modèle de l'articulation :

$$\begin{cases} J_M \alpha_M = \tau_M - \eta \tau_T - \tau_{F_M} \\ \\ J_L \alpha_L = N \tau_T - \tau_{F_L} \end{cases} \tag{4.6}$$

Avec les notations générales, nous utilisons J pour l'inertie, α pour l'accélération, τ pour le couple appliqué excepté les couples de frottements et τ_F le couple total de frottements.

Ceci nous permet de n'écrire désormais qu'une équation pouvant représenter aussi bien le moteur que la charge. Dans la suite, nous ne considérerons plus qu'une seule des deux parties, notre objectif étant maintenant de mettre au point un algorithme d'estimation performant.

Pour le moteur il suffira de prendre :

$$\begin{cases} \tau = \tau_M - \eta\tau_T \\ \alpha = \alpha_M \\ J = J_M \\ \tau_F = \tau_{F_M} \end{cases} \tag{4.7}$$

Et pour la charge :

$$\begin{cases} \tau = N\tau_T \\ \alpha = \alpha_L \\ J = J_L \\ \tau_F = \tau_{F_L} \end{cases} \tag{4.8}$$

Maintenant, l'équation générale pour le moteur ou la charge s'écrit :

$$J\alpha = \tau - \tau_F \tag{4.9}$$

Le couple de frottements a été modélisé dans le troisième chapitre et décomposé en trois termes, chacun de ces termes comprenant un des coefficients à identifier.

Là encore nous écrivons de façon générale l'expression du modèle de frottements :

$$\tau_F = F_c sign(\omega) + F_v\omega + F_s sign(\omega)\exp\left(-(\omega/\omega_s)^2\right) \tag{4.10}$$

Il suffit d'ajouter les indices M et L pour obtenir les équations du moteur et de la charge respectivement.

En injectant (4.10) dans (4.9) nous obtenons :

$$J\alpha = \tau - F_c sign(\omega) - F_v \omega - F_s sign(\omega) \exp\left(-(\omega/\omega_s)^2\right) \tag{4.11}$$

Dans le but d'utiliser l'accélération comme signal d'entrée de l'algorithme d'identification par la méthode des moindres carrés, une expression linéaire de cette grandeur vis-à-vis des paramètres inconnus est recherchée.

Cette expression linéaire apparaît directement dans l'équation (4.12), obtenue à partir de (4.11) par division par l'inertie :

$$\alpha = \frac{1}{J}\tau - \frac{F_c}{J} sign(\omega) - \frac{F_v}{J}\omega - \frac{F_s}{J} sign(\omega) \exp\left(-(\omega/\omega_s)^2\right) \tag{4.12}$$

La forme linéaire recherchée est ainsi obtenue :

$$\alpha = \boldsymbol{\Phi}^T \boldsymbol{W} \tag{4.13}$$

Avec :

$$\boldsymbol{\Phi}^T = \left[\tau, -sign(\omega), -\omega, -sign(\omega) exp\left(-(\omega/\omega_s)^2\right)\right] \tag{4.14}$$

$$\boldsymbol{W} = \left[\frac{1}{J}, \frac{F_c}{J}, \frac{F_v}{J}, \frac{F_s}{J}\right] \tag{4.15}$$

$\boldsymbol{\Phi}$ est appelé régresseur ou vecteur de regression et $\boldsymbol{W}$ est appelé vecteur des paramètres.

Remarques : La notation $\boldsymbol{W}$ a été préférée pour le vecteur des paramètres puisque la lettre θ généralement employée représente déjà la position angulaire dans notre système.
Les paramètres recherchés pour le contrôle sont en fait les inerties J et les coefficients de friction F_c, F_v *et* F_s.

Notre formulation du problème d'identification des paramètres de l'articulation nous permet d'obtenir des paramètres qui ne sont pas directement ceux recherchés. Le premier paramètre permet d'obtenir une estimation du paramètre d'inertie par inversion, et les autres sont couplés au premier par multiplication. Un inconvénient de cette formulation est que les paramètres sont directement corrélés par la présence du terme $1/J$ dans chacun d'entre eux. En conséquence, une erreur importante dans l'estimation du premier paramètre entraînera directement une erreur dans l'estimation des autres paramètres.

Ainsi formulée, la méthode récursive des moindres carrés est appliquée pour l'estimation des paramètres de $\boldsymbol{W}$, cet algorithme est composé principalement de trois équations :

- Calcul de l'innovation e,
- Mise à jour du vecteur des paramètres estimés,
- Mise à jour de la matrice de covariance.

En termes d'équations, nous écrivons :

$$e_{(n)} = \alpha_{(n)} - \boldsymbol{\Phi}_{(n)}^T \hat{\boldsymbol{W}}_{(n-1)} \tag{4.16}$$

$$\hat{\boldsymbol{W}}_{(n)} = \hat{\boldsymbol{W}}_{(n-1)} + \left(\boldsymbol{P}_{(n-1)}\boldsymbol{\Phi}_{(n)}\right)/\left(\lambda + \boldsymbol{\Phi}_{(n)}^T \boldsymbol{P}_{(n-1)}\boldsymbol{\Phi}_{(n)}\right) e_{(n)} \tag{4.17}$$

$$\boldsymbol{P}_{(n)} = (1/\lambda)\boldsymbol{P}_{(n-1)} - (1/\lambda)\left(\boldsymbol{P}_{(n-1)}\boldsymbol{\Phi}_{(n)}\boldsymbol{\Phi}_{(n)}^T \boldsymbol{P}_{(n-1)}\right)/\left(\lambda + \boldsymbol{\Phi}_{(n)}^T \boldsymbol{P}_{(n-1)}\boldsymbol{\Phi}_{(n)}\right) \tag{4.18}$$

L'initialisation de la méthode RLS peut être réalisée par :

$$\begin{cases} \boldsymbol{P}_0 = \delta \boldsymbol{I} \\ \hat{\boldsymbol{W}}_0 = 0 \end{cases} \tag{4.19}$$

Où δ est une constante positive qui sera choisie d'autant plus grande que la confiance accordée aux valeurs initiales des paramètres estimés sera faible.

Ces équations constituent le noyau essentiel de l'identification des paramètres de l'articulation.

Le couple de friction estimé qui est utilisé dans la loi de commande est calculé d'après les paramètres estimés :

$$\hat{\tau}_F = \hat{F}_c sign(\omega) + \hat{F}_v \omega + \hat{F}_s sign(\omega) \exp\left(-(\omega/\omega_s)^2\right) \tag{4.20}$$

L'utilisation du facteur d'oubli λ peut amener la matrice de covariance à prendre des valeurs très élevées en absence d'excitation. Lorsque λ est choisi égal à 1, les valeurs de la matrice de covariance tombent rapidement à 0, et l'algorithme n'est plus capable alors de détecter les variations de paramètres. Ce choix est donc adapté aux cas où les paramètres à identifier sont invariants. Lorsque nous fixons le facteur d'oubli à une valeur inférieure à 1, l'équation (4.18) de mise à jour de la matrice de covariance nous montre que lorsque les valeurs du vecteur $\boldsymbol{\Phi}$ sont très faibles, nous avons :

$$\boldsymbol{P}_n \approx \frac{1}{\lambda} \boldsymbol{P}_{n-1} \tag{4.21}$$

Il apparaît clairement que les valeurs de la matrice $\boldsymbol{P}$ vont alors diverger jusqu'à ce que $\boldsymbol{\Phi}$ reprenne des valeurs non négligeables devant $\boldsymbol{P}$. Des simulations ont permis de montrer que les coefficients de la matrice de covariance pouvaient prendre des valeurs de 10^{100} ce qui, en terme d'implantation, nous obligerait à implanter l'algorithme avec une représentation à virgule flottante.

Dans la méthode utilisée ici, ce problème a été évité en choisissant le facteur d'oubli de façon adaptative. Nous pouvons considérer que lorsque l'erreur d'estimation est faible, nous n'avons pas besoin de modifier les valeurs estimées des paramètres. Lorsque l'erreur d'estimation devient importante, cela signifie que les valeurs estimées des paramètres doivent être modifiées. Ainsi, le facteur d'oubli sera choisi différent de 1 seulement lorsqu'un niveau d'erreur d'estimation suffisamment important sera détecté. Nous pouvons par exemple définir un seuil d'erreur ε_e tel que :

$$\text{Si } |e_n| > \varepsilon_e \quad \text{alors } \lambda = \lambda_{inf}$$

$$\text{Si } |e_n| \leq \varepsilon_e \quad \text{alors } \lambda = 1$$

Où $\lambda_{inf} \in]0;1[$.

Il est alors possible de généraliser cette méthode en définissant plusieurs seuils d'erreurs d'estimation et plusieurs valeurs du facteur d'oubli pour chacun des intervalles. Dans notre cas, l'utilisation d'un seuil a permis d'obtenir de bonnes performances de l'algorithme tout en limitant l'amplitude des valeurs de la matrice de covariance.

Les paramètres estimés seront en sortie passés dans une fonction de saturation définie suivant les bornes inférieures et supérieures supposées de ces paramètres afin d'éviter d'utiliser des valeurs incohérentes.

4.4.2. Effets de quantification

Les algorithmes de traitement des signaux sont destinés à être implantés dans des circuits intégrés numériques qui ne peuvent manipuler que des signaux numériques. L'amplitude d'un signal numérique est limitée à un nombre fini de valeurs, c'est ce qu'on appelle la quantification.

Les effets de la quantification des signaux sur un algorithme peuvent se situer à plusieurs niveaux : sur les entrées, sur les calculs internes et sur les sorties [KUN91].

Afin de déterminer les effets de quantification sur notre algorithme d'estimation, il faut remplacer toutes les variables par leurs valeurs quantifiées.

En pratique, nous rencontrons deux méthodes de représentation numérique des nombres réels : la représentation à virgule fixe et la représentation à virgule flottante. La représentation en virgule flottante offre une plus grande précision dans la représentation des nombres mais elle est plus coûteuse en temps de calcul et en surface d'intégration. C'est pourquoi nous nous intéresserons ici à l'implantation à virgule fixe.

La quantification des signaux en virgule fixe est caractérisée par :

- Le nombre de bits pour la représentation,
- Le type de représentation des nombres négatifs,
- La loi de quantification à l'intérieur de la dynamique,
- La loi de quantification à l'extérieur de la dynamique.

Pour notre étude d'effets de quantification, nous allons fixer le type de représentation des nombres négatifs, la loi de quantification et la loi de dépassement, puis nous ferons varier

le nombre de bits de représentation pour évaluer le nombre minimum de bits nécessaires au maintien des performances de l'algorithme d'estimation.

Ainsi, les caractéristiques suivantes ont été retenues :

- Représentation des nombres négatifs en complément à 2,
- Quantification à l'intérieur de la dynamique par arrondi,
- Quantification à l'extérieur de la dynamique par saturation.

Un bit de signe est destiné à distinguer les nombres négatifs des nombres positifs, ce bit sera 0 si le nombre est positif et 1 si le nombre est négatif.

Pour la quantification à l'intérieur de la dynamique, nous choisissons la méthode de l'arrondi qui consiste à choisir la valeur représentable la plus proche de la valeur à représenter. Pour la quantification à l'extérieur de la dynamique, nous choisissons la méthode de saturation qui consiste à saturer les valeurs qui dépassent le domaine de la dynamique.

4.4.3. Résultats et discussion

Étant donné que la méthode d'estimation est rigoureusement analogue côté moteur et côté charge, les simulations ont été effectuées seulement pour la charge et suffiront à valider la méthode. Ces simulations ont permis de montrer qu'un minimum de 28 bits était nécessaire pour maintenir de bonnes performances avec la méthode RLS.

Les résultats de simulation pour une quantification sur 28 bits sont présentés dans les figures 4-2 à 4-5 où nous retrouvons :

- les inerties réelles et estimées,
- le couple de friction réel et l'erreur d'estimation sur ce couple,
- l'erreur de position.

Pour les simulations, nous supposons disponible une mesure de l'accélération avec un bruit additif de 20dB. Les constantes de simulation sont :

$$\lambda_{inf} = 0.97,$$

$$\delta = 10,$$

$$T = 0.0001s$$

Les mêmes valeurs numériques seront utilisées pour les simulations du filtre de Kalman. Les résultats obtenus pour une représentation à 24 et 20 bits sont présentés en annexe A. Nous pouvons remarquer que la friction totale est estimée avec une bonne précision tant en régime statique que dynamique. Sur la figure 4-4, nous constatons que l'algorithme peut parfois mettre un certain temps avant de trouver une bonne valeur estimée de l'inertie mais ceci s'explique par le fait que dans les portions où l'accélération est nulle, l'estimateur est désactivé. Nous remarquons alors que dès que l'accélération redevient non nulle, l'estimation reprend et l'algorithme retrouve très rapidement de bonnes valeurs estimées.

Le tableau 4-1 présente les erreurs absolues moyennes obtenues avec cette méthode sur l'inertie, le couple total de frottements, la vitesse et la position de la charge. Ces résultats ont été obtenus avec une représentation à 28 bits des données.

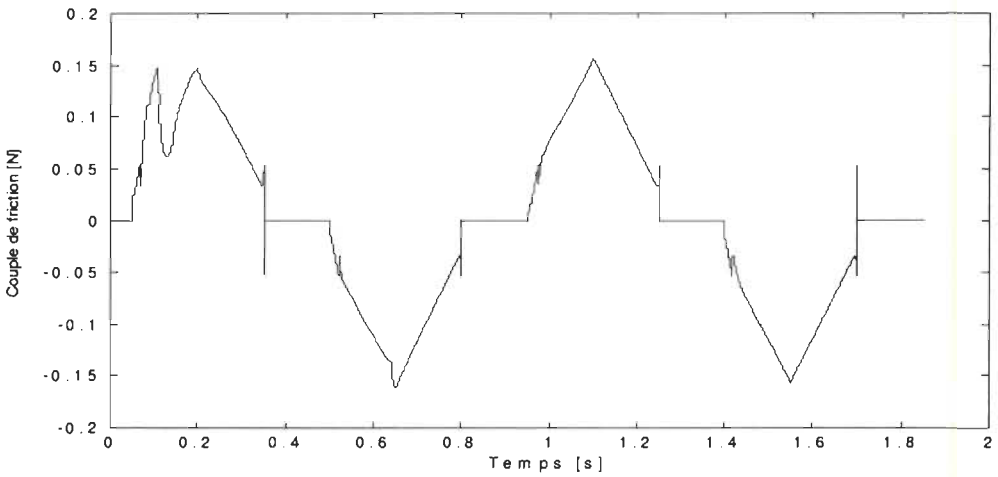


Figure 4-2 : Couples de friction réel avec la méthode RLS sur 28 bits

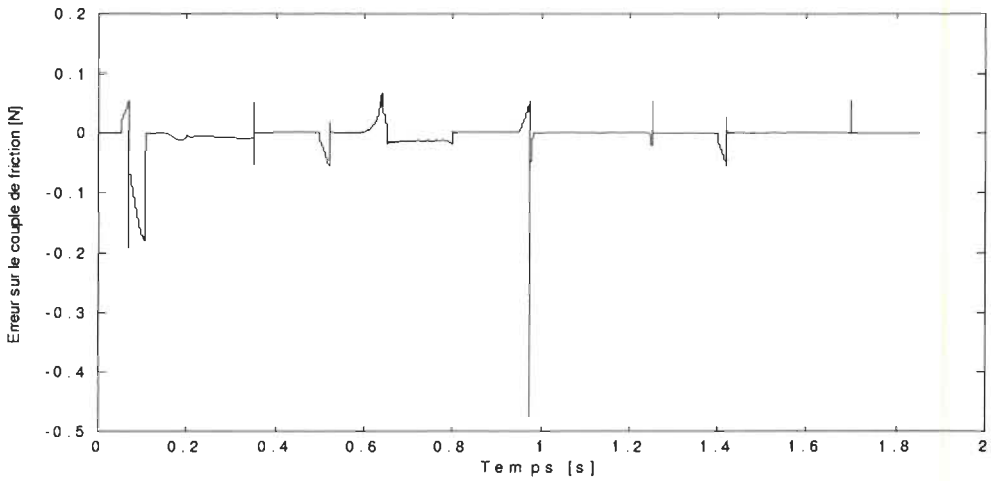


Figure 4-3 : Erreur sur l'estimation du couple de friction, méthode RLS sur 28 bits

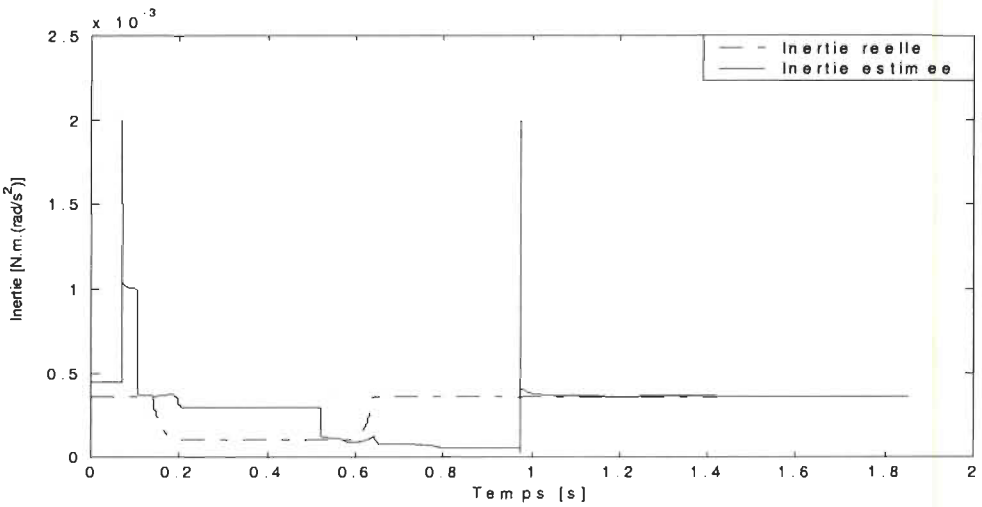


Figure 4-4 : Inerties réelles et estimées, méthode RLS sur 28 bits

L'étude d'effets de quantification sur la méthode RLS a montré qu'un minimum de 28 bits seraient nécessaires à son implantation. Ce nombre de bits nous indique déjà que l'implantation de cette méthode en technologie ITGE serait très coûteuse en surface de silicium. De plus, cette méthode ne possède pas un degré de parallélisme naturel très important et est très exigeante en quantités de calculs. Ces trois points la rendent peu attrayante pour l'implantation.

Nous devons donc penser à reformuler la méthode d'estimation afin de diminuer les effets de quantification et augmenter si possible le parallélisme naturel de l'algorithme d'estimation.

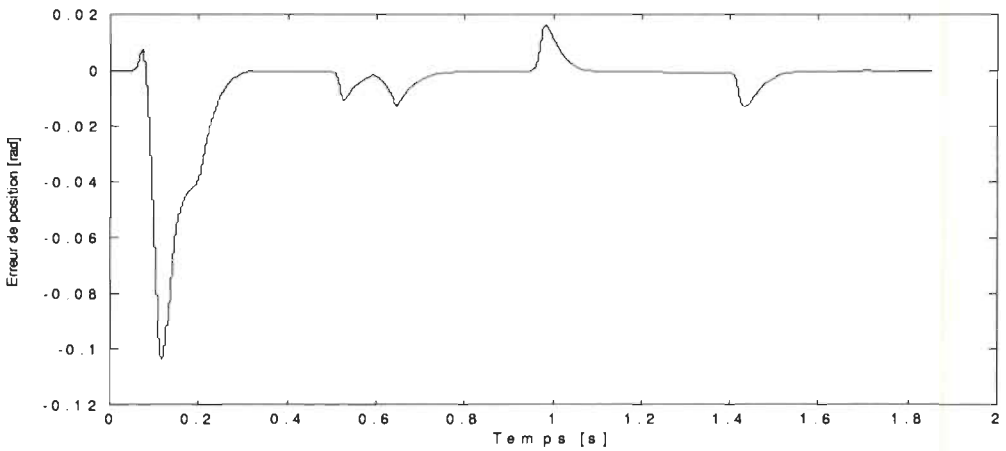


Figure 4-5 : Erreur de positions pour la charge, méthode RLS sur 28 bits

Tableau 4-1 : Erreurs absolues moyennes de position, vitesse, inertie estimée et couple de friction estimé avec la méthode RLS sur 28 bits

Signal	Erreur absolue moyenne
Position	6.0×10^{-3}
Vitesse	8.9×10^{-2}
Inertie estimée	1.7×10^{-5}
Couple de friction total estimé	6.8×10^{-3}

4.5. Formulation de la méthode en vue d'une intégration

4.5.1. Filtre de Kalman à racine carrée de covariance

La sensibilité de la méthode RLS aux effets de quantification provient essentiellement de la propagation de la matrice de covariance $\boldsymbol{P}$ dont nous avons vu que les coefficients pouvaient prendre des valeurs très élevées notamment avec l'utilisation du facteur d'oubli. L'idée principale des algorithmes à racine carrée de covariance est de propager la racine carrée de $\boldsymbol{P}$ afin de réduire la dynamique et augmenter la robustesse face aux effets de quantification [HAY98].

Si $\boldsymbol{S}$ est la matrice racine carrée de $\boldsymbol{P}$ alors nous avons :

$$\boldsymbol{P} = \boldsymbol{S}\boldsymbol{S}^{T} \tag{4.22}$$

D'après le lemme de factorisation de matrices, si nous considérons deux matrices $\boldsymbol{A}$ et $\boldsymbol{B}$ de dimension $N*M$, alors :

$$\boldsymbol{A}\boldsymbol{A}^{H} = \boldsymbol{B}\boldsymbol{B}^{H} \tag{4.23}$$

Si et seulement s'il existe une matrice unitaire $\boldsymbol{\Theta}$ telle que

$$\boldsymbol{B} = \boldsymbol{A}\boldsymbol{\Theta} \tag{4.24}$$

En utilisant ce lemme de factorisation de matrices dans l'équation de propagation de la matrice de covariance, la matrice $\boldsymbol{S}$ peut être calculée de façon récursive. Les étapes de calcul permettant d'aboutir a cette équation récursive sont présentées dans [HAY96]. La matrice $\boldsymbol{S}$ a la propriété d'être triangulaire supérieure.

La matrice $\boldsymbol{S}$ sera évaluée par une méthode de triangularisation de matrices. Parmi les méthodes de triangularisation de matrices, les plus performantes utilisées ont été développées par Gram-Schmidt, Householder et Givens (cf. deuxième chapitre).

Nous utiliserons la méthode basée sur les rotations de Givens qui est reconnue pour sa stabilité numérique et sa robustesse aux effets de quantification. Cette méthode peut être appliquée sur l'algorithme RLS (méthode QRD-RLS) et sur le filtre de Kalman. Nous emploierons ici une adaptation du filtre de Kalman qui s'est avérée plus performante et moins sensible aux effets de quantification. Notre modèle d'estimation est décrit par les équations (4.25) et (4.26) :

$$\boldsymbol{W}_{(k)} = \boldsymbol{W}_{(k-1)} + \boldsymbol{\eta}_{W(k)} \tag{4.25}$$

$$\tilde{\alpha}_{(k)} = \boldsymbol{W}_{(k)}^{T}\boldsymbol{\Phi}_{(k)} + \eta_{\alpha(k)} \tag{4.26}$$

η_α est le bruit sur la mesure de l'accélération α et $\boldsymbol{\eta}_W$ le bruit sur les paramètres estimés. Pour utiliser le filtre de Kalman, nous supposons que les bruits $\boldsymbol{\eta}_W$ et η_α sont des bruits blancs non corrélés.

Les équations (4.27) à (4.29) décrivent alors l'algorithme d'estimation utilisé :

$$\boldsymbol{S}_{(k+1/k)}^{T} = \frac{1}{\lambda_{(k)}}\boldsymbol{S}_{(k/k)}^{T} \tag{4.27}$$

$$\begin{bmatrix} f_{(k+1)} & \boldsymbol{g}_{(k+1)}^{T} \\ \boldsymbol{0} & \boldsymbol{S}_{(k+1/k+1)}^{T} \end{bmatrix} = \boldsymbol{T}\begin{bmatrix} \beta & \boldsymbol{0}^{T} \\ \boldsymbol{S}_{(k+1/k)}^{T}\boldsymbol{\Phi}_{(k+1)} & \boldsymbol{S}_{(k+1/k)}^{T} \end{bmatrix} \tag{4.28}$$

$$\hat{\boldsymbol{W}}_{(k+1/k+1)} = \hat{\boldsymbol{W}}_{(k+1/k)} + \frac{\boldsymbol{g}_{(k+1)}}{f_{(k+1)}}\left(\tilde{\alpha}_{(k+1)} - \hat{\boldsymbol{W}}_{(k+1/k)}^{T}\boldsymbol{\Phi}_{(k+1)}\right) \tag{4.29}$$

Le paramètre λ joue ici le même rôle que le facteur d'oubli dans la méthode RLS et sera calculé de façon adaptative comme nous l'avons vu au paragraphe 4.4.1.

L'initialisation de l'algorithme est réalisée de la façon suivante :

$$\begin{aligned} \hat{\boldsymbol{W}}_{(0/0)} &= \boldsymbol{0} \\ \boldsymbol{S}^{T}_{(0/0)} &= \delta \boldsymbol{I} \end{aligned} \tag{4.30}$$

Où δ est une constante positive et $\boldsymbol{I}$ désigne la matrice identité. $\boldsymbol{T}$ est la matrice de transformation unitaire basée sur les rotations de Givens, β est le rapport des variances des bruits sur la sortie mesurée et sur les paramètres estimés.

$$\beta = \frac{\sigma_v^2}{\sigma_w^2} \tag{4.31}$$

4.5.2. Réduction des effets de quantification

La cause principale des effets de quantification lorsque nous utilisons la représentation à virgule fixe est la dynamique des variables de l'algorithme.

Lors de l'étude d'effets de quantification sur la méthode RLS, il est apparu que la valeur maximale à représenter lorsque nous pouvions maîtriser les fluctuations des coefficients de la matrice de covariance est la première valeur du vecteur des paramètres estimés, c'est-à-dire l'inverse de l'inertie. De plus, la valeur de l'inertie est utilisée dans le calcul du couple de commande, nous devons donc dans la même dynamique représenter l'inertie

et son inverse. Ce paramètre ayant des valeurs de l'ordre de 10^{-4}, alors son inverse est de l'ordre de 10^4 ce qui implique une grande dynamique pour la quantification et donc une perte de précision avec la représentation à virgule fixe.

Avec la méthode basée sur le filtre de Kalman à racine carrée de covariance, nous avons toujours ce problème. Cependant, le problème peut être contourné en procédant au changement de variable suivant :

$$J' = KJ \tag{4.32}$$

Où K doit être choisi de façon à diminuer l'écart entre les valeurs de J' et $1/J'$. Prenons par exemple :

$$K = 100$$

Alors les valeurs de J' et $1/J'$ sont de l'ordre de 10^{-2} et 10^2 respectivement. La dynamique pour ces deux variables a été considérablement réduite. Il faut maintenant remplacer la variable J dans les équations de l'algorithme et voir les conséquences de ce changement de variable sur les autres grandeurs.

Nous avons vu au paragraphe 4.4.1 que l'inertie intervenait dans l'équation (4.9) :

$$J\alpha = \tau - \tau_F$$

En effectuant le changement de variable et en procédant de même qu'au paragraphe 4.4.1, nous obtenons une nouvelle formulation de l'équation (4.12) :

$$\alpha' = \frac{1}{J'}\tau - \frac{F_c}{J'}sign(\omega) - \frac{F_v}{J'}\omega - \frac{F_s}{J'}sign(\omega)\exp\left(-(\omega/\omega_s)^2\right) \tag{4.33}$$

α' est obtenue par le même changement de variable à partir de α :

$$\alpha' = \alpha / K \tag{4.34}$$

Pour utiliser ce changement de variable il faudra également revoir les équations 3.8 et 3.18 de calcul des couples de charge et de commande du moteur puisque ceux-ci dépendent des inerties. Ces équations deviennent :

$$\tau_T^* = \frac{1}{N}\left[\hat{J}'_L \alpha_L'^* + \hat{\tau}_{F_L} + K_{V_L}\left(\omega_L^* - \omega_L\right) + K_{P_L}\left(\theta_L^* - \theta_L\right)\right] \tag{4.35}$$

$$\tau_M = \hat{J}'_M \alpha_M'^* + \hat{\tau}_{F_M} + \eta\tau_T^* + K_{V_M}\left(\omega_M^* - \omega_M\right) + K_{P_M}\left(\theta_M^* - \theta_M\right) \tag{4.36}$$

Où :

$$\alpha'^* = \alpha^* / K \tag{4.37}$$

Pour réduire encore les effets de quantification sur l'algorithme d'estimation des paramètres et sur la loi de commande, nous pouvons étudier plus en détail la dynamique de chacune des variables utilisées. Les simulations ont montré que la dynamique des variables demeurait plus grande à l'intérieur de l'algorithme d'estimation que pour le calcul de la loi de commande par les équations (4.35) et (4.36). Nous pouvons alors diviser la calcul des couples de commande en deux blocs et normaliser les entrées suivant la dynamique de leurs variables. La figure 4-2 donne une représentation de cette décomposition. Les variables externes au bloc d'estimation seront normalisées par un maximum moins élevé que les variables internes à ce bloc. D'une manière générale, toutes les variables seront normalisées par un premier maximum et les variables du bloc

d'estimation seront normalisées une seconde fois par le maximum de la dynamique des variables de ce bloc.

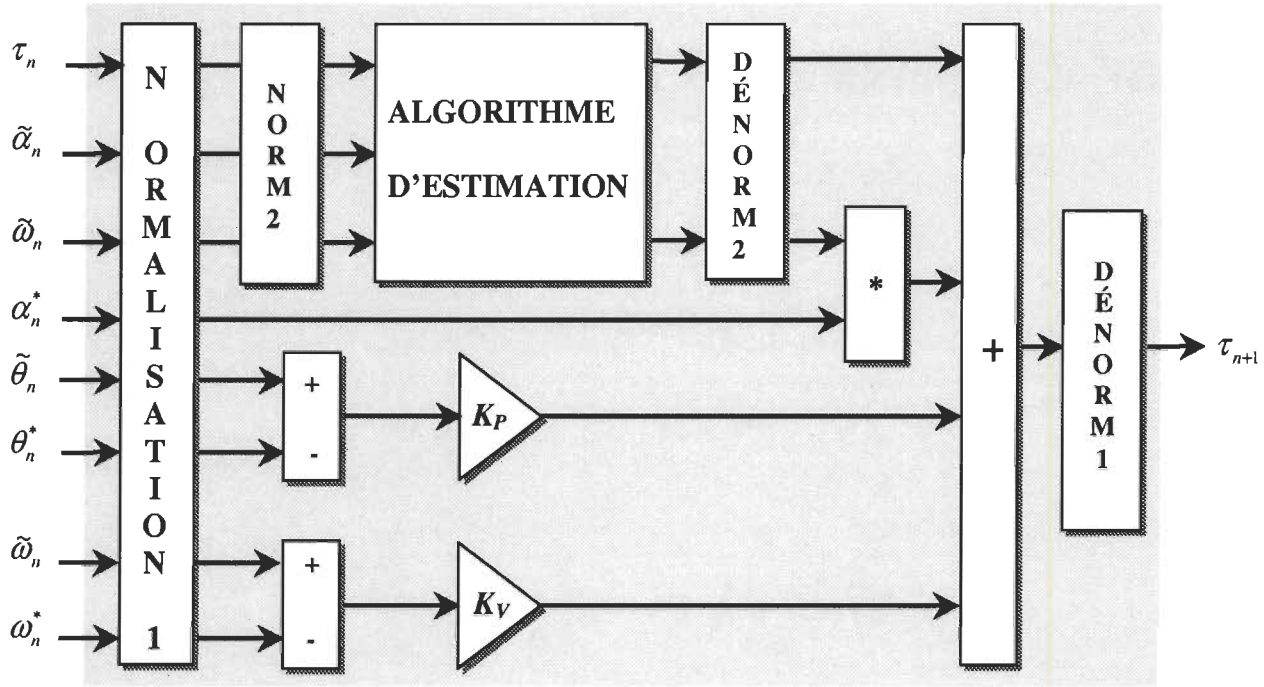


Figure 4-6 : Structure de normalisation des données

La normalisation est habituellement une division par la plus grande des valeurs à représenter. Nous choisirons ici pour chacun des deux blocs une valeur maximale supérieure à la plus grande valeur à représenter pour éviter les problèmes de saturation et nous choisirons de plus cette valeur comme étant une puissance de deux ce qui nous permettra d'effectuer la normalisation et la dénormalisation par décalages à droite et à gauche des variables numériques d'entrée. Pour la deuxième normalisation, nous choisirons le maximum de la même façon mais en le divisant par le maximum de la première normalisation puisque les données auront déjà été normalisées une première fois. D'une manière générale, toutes les données sont représentées sur 16 bits et normalisées par rapport au maximum de toutes les variables non internes à l'estimateur,

et les variables de l'estimateur sont normalisées une seconde fois pour permettre à l'estimateur d'opérer dans une dynamique plus grande.

4.5.3. Résultats et discussion

Nous avons réalisé les mêmes simulations qu'avec la méthode RLS et cette fois encore avec la charge seule. Les résultats de simulation avec quantification à 16 bits sont présentés dans les figures 4-7 à 4-10. Ces résultats nous permettent de constater la diminution des effets de quantification en utilisant l'algorithme basé sur le filtre de Kalman à racine carrée de covariance avec le changement de variable de (4.32). Nous pouvons constater qu'à partir d'une représentation sur 16 bits, les résultats s'avèrent très satisfaisants. L'utilisation de cette méthode a donc permis de réduire le nombre de bits nécessaires de 28 à 16 ce qui est considérable en terme de coût d'implantation. Ces résultats confirment la robustesse des méthodes à racine carrée de covariance aux effets de quantification et l'intérêt qu'elles présentent par rapport aux méthodes classiques telles que la méthode RLS standard.

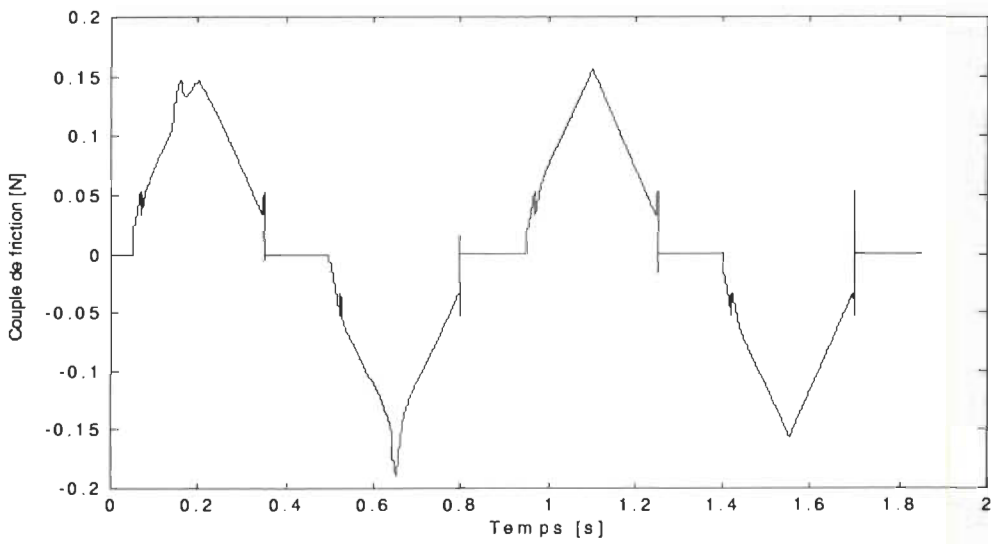


Figure 4-7 : Couple de friction réel de la charge

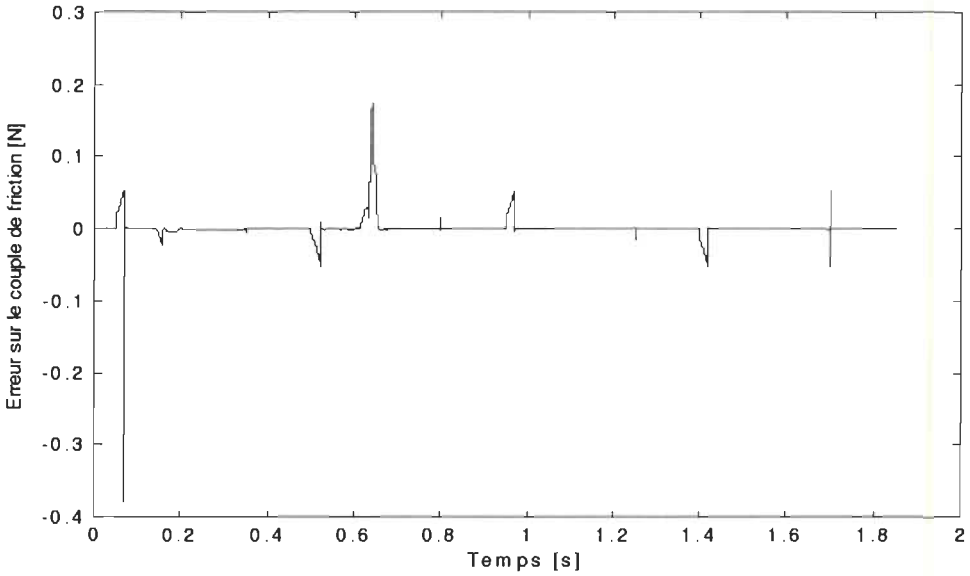


Figure 4-8 : Erreur d'estimation sur le couple de friction, filtre de Kalman sur 16 bits

Les résultats de simulation pour une représentation à 20 et 24 bits sont présentés en annexe B. Le paramètre β a été choisi égal à 0.9 pour les simulations.

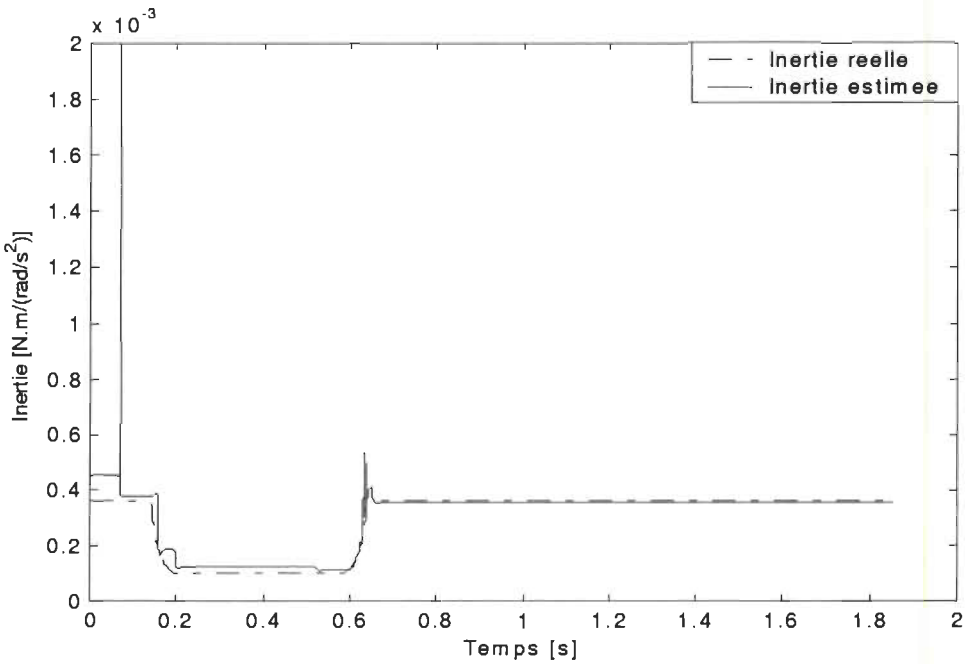


Figure 4-9 : Inerties réelles et estimées de la charge, filtre de Kalman sur 16 bits

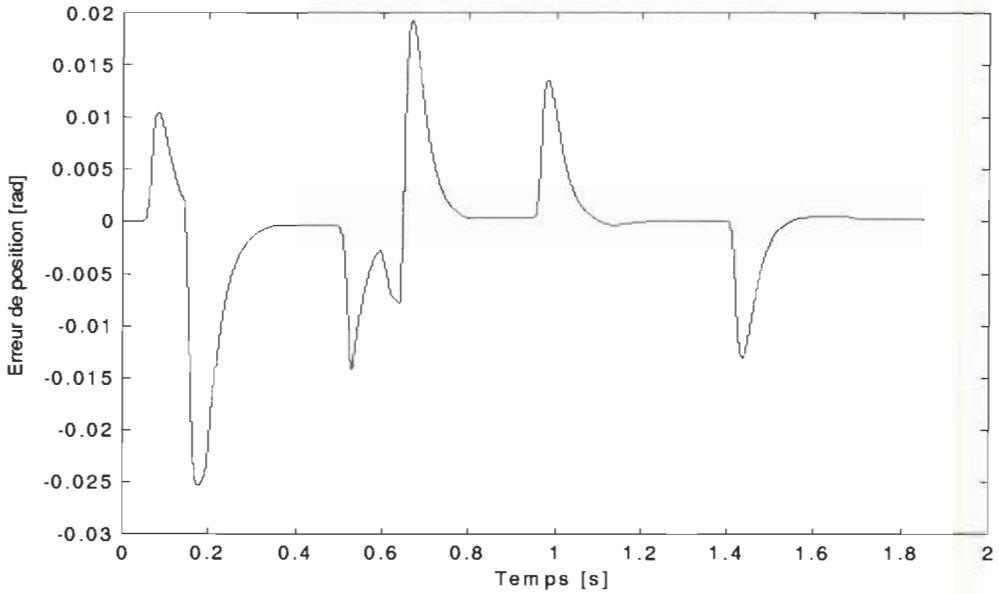


Figure 4-10 : Erreur de position de la charge, filtre de Kalman sur 16 bits

Nous pouvons constater que l'algorithme est stable quand la quantité à estimer est invariante, et qu'il est capable de suivre les variations de paramètres. Ceci est grandement dû à l'utilisation particulière du facteur λ qui permet à la fois d'éviter l'instabilité en prenant la valeur 1 lorsque l'excitation de l'algorithme est faible et de suivre les variations de paramètres lorsque sa valeur est différente de 1 en augmentant la trace de la matrice de covariance par l'équation (4.27).

La figure 4-10 peut être comparée directement avec la figure 4-5. Nous remarquons alors que l'erreur de position angulaire de la charge demeure inférieure à 2% alors qu'avec la méthode RLS cette erreur atteint un maximum supérieur à 10%.

Les résultats sont évalués quantitativement par le calcul de l'erreur absolue moyenne sur la position, la vitesse, l'inertie estimée et le couple de friction estimé. Les résultats sont regroupés dans le tableau 4-2.

Tableau 4-2 : Erreurs absolues moyennes de position, vitesse, inertie estimée et couple total de friction estimé pour une représentation à 16 bits

Signal	Erreur absolue moyenne
Position	3.3×10^{-2}
Vitesse	4.7×10^{-1}
Inertie estimée	1.1×10^{-5}
Couple de friction total estimé	3.1×10^{-3}

Nous constatons par comparaison avec le tableau 4-1 que cette méthode offre des résultats nettement supérieurs à ceux obtenus avec la méthode RLS standard. Les erreurs absolues moyennes sur les quantités estimées ont été réduites d'un facteur de l'ordre de 2. Le tableau 4-3 permet de comparer les erreurs absolues moyennes obtenues par les deux méthodes présentées avec une représentation à 20 et 24 bits.

Tableau 4-3 : Comparaison des erreurs absolues moyennes des deux méthodes d'estimation avec une quantification à 20 et 24 bits

Signal	RLS		Filtre de Kalman	
	20 bits	24 bits	20 bits	24 bits
Position de la charge	7.20×10^{-1}	2.15×10^{-2}	3.3×10^{-3}	3.2×10^{-3}
Friction estimée de la charge	6.71×10^{-1}	2.01×10^{-2}	3.10×10^{-3}	3.05×10^{-3}
Inertie estimée de la charge	5.94×10^{-4}	1.91×10^{-4}	1.6×10^{-5}	1.55×10^{-5}

4.6. Évaluation de performances de la loi de commande dans un DSP

Le nombre de cycles nécessaires à l'implantation de la loi de commande dans un processeur de type DSP composé d'une unité de traitement a été estimé à plus de 4000 cycles. Considérant un DSP avec une fréquence d'horloge de 50 MHz, le calcul de la loi de commande ne peut donc être effectué en moins de 80 µs. Pour réduire ce temps de calcul, nous devons exploiter le parallélisme inhérent aux algorithmes afin de rendre parallèles les calculs et les communications de données dans une architecture ITGE dédiée composée de plusieurs processeurs élémentaires.

4.7. Conclusion

Les programmes de simulation des modèles et algorithmes développés sont présentés en annexe D.

Les différents résultats obtenus ont montré l'importance de l'étude algorithmique puisque celle-ci nous a permit de remplacer une méthode d'estimation nécessitant un minimum de 28 bits pour la représentation des données par une autre méthode n'en nécessitant que 16. La méthode basée sur le filtre de Kalman à racine carrée de covariance permet d'obtenir de meilleurs résultats que la méthode RLS même avec un nombre de bits moins important pour la représentation des données. Ces résultats démontrent également qu'il est très important de réaliser l'étude d'effets de quantification avant d'envisager l'étape d'implantation. La méthode RLS offrait une bonne qualité d'estimation à première vue, mais s'est avérée inutilisable à cause de sa sensibilité aux effets de quantification.

Par sécurité, nous utiliserons une représentation sur 20 bits pour toutes les variables du contrôleur, cela signifie que les mesures des positions et vitesses devront être converties sur 20 bits. L'utilisation de capteurs avec convertisseurs analogique/numérique ne permettrait pas de maintenir une bonne précision au delà de 12 bits, mais en utilisant des capteurs à impulsions, le résultat se trouve directement sous une forme numérique et permet d'éviter les problèmes reliés à la conversion.

Nous devons donc maintenant développer des architectures permettant l'implantation des équations d'estimation des paramètres ainsi que des calculs de couples pour la charge et le moteur.

Chapitre 5

Implantation en technologie ITGE du contrôleur proposé

5.1. Introduction

Dans le quatrième chapitre, nous avons montré que le filtre de Kalman à racine carrée de covariance est une solution performante pour notre problème d'estimation et présente le double avantage d'avoir une bonne robustesse aux effets de quantification et une structure intrinsèquement parallèle. Ces caractéristiques nous ont permit de conclure que cette formulation était la plus favorable à l'implantation en technologie ITGE [MOZ99].

Après l'étude d'effets de quantification, la première étape vers l'implantation est le développement d'architectures capables de recevoir les algorithmes développés. Ces architectures doivent tirer un maximum de profit des possibilités de mise en parallèle des calculs et des communications de données des algorithmes.

Même si l'étude architecturale paraît, dans ce mémoire, séparée de l'étude algorithmique, dans la réalité les développements d'algorithmes et d'architectures se font de manière conjointe si nous désirons aboutir à une implantation efficace. Ainsi, le choix du filtre de Kalman à racine carrée de covariance était déjà motivé au départ par sa structure facilitant le développement d'une architecture parallèle associée.

Les troisième et quatrième chapitres nous ont permit de constater que le contrôleur conçu pour une articulation flexible est divisé en deux blocs dont les structures sont très similaires. Ceci est intéressant pour le développement de l'architecture du contrôleur. En effet, nous pourrons nous inspirer grandement du développement architectural du contrôleur de la charge pour concevoir l'architecture du contrôleur du moteur (Figure 3-9).

Le développement des architectures sera décomposé en trois étapes :

- développement d'une architecture pour le contrôleur de la charge (section 5.2),
- développement d'une architecture pour le contrôleur du moteur (section 5.3),
- développement d'une architecture pour le contrôleur complet de l'articulation comprenant les deux architectures précédentes (section 5.4).

5.2. Architecture du contrôleur de la charge

5.2.1. Structure générale

Dans le troisième chapitre, nous avons pu remarquer la structure à deux boucles des contrôleurs de la charge et du moteur. Cette structure correspond à celle d'un régulateur

auto-syntonisant. Dans l'architecture du contrôleur de la charge, nous retrouvons cette décomposition et nous ajoutons les éléments nécessaires à la génération de trajectoire désirée pour le moteur. La figure 5-1 donne une représentation schématique des données d'entrée/sortie du contrôleur de la charge pour un premier niveau de décomposition.

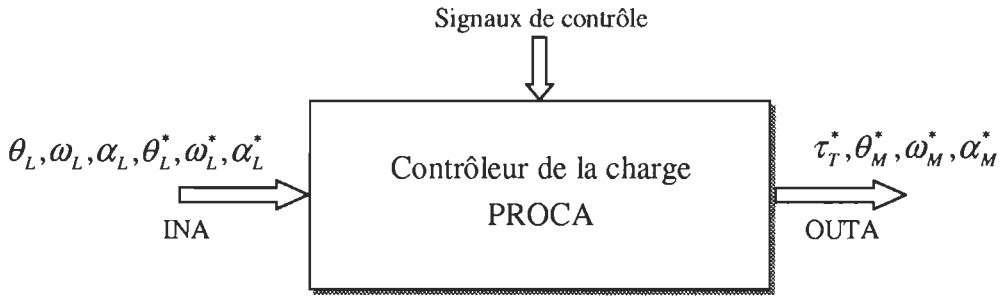


Figure 5-1 : Structure globale du contrôleur de la charge

Le contrôleur utilise les trajectoires réelles et désirées de la charge pour calculer d'une part le couple idéal à transmettre à la charge et d'autre part générer la trajectoire désirée du moteur. Ces calculs sont effectués suivant les équations présentées aux troisième et quatrième chapitres. Dans la suite, les bus représentés sur les schémas seront, sauf spécification contraire, des bus de 20 bits. Les suffixes « A » et « B » désigneront respectivement les blocs reliés aux contrôleurs de la charge et du moteur.

La figure 5-2 représente le deuxième niveau de décomposition du contrôleur de la charge, où nous retrouvons les blocs suivants :

- un réseau systolique qui effectue les calculs,
- un bloc de multiplexage qui distribue les données dans l'architecture,
- un bloc de contrôle fournissant les signaux de contrôle de l'ensemble.

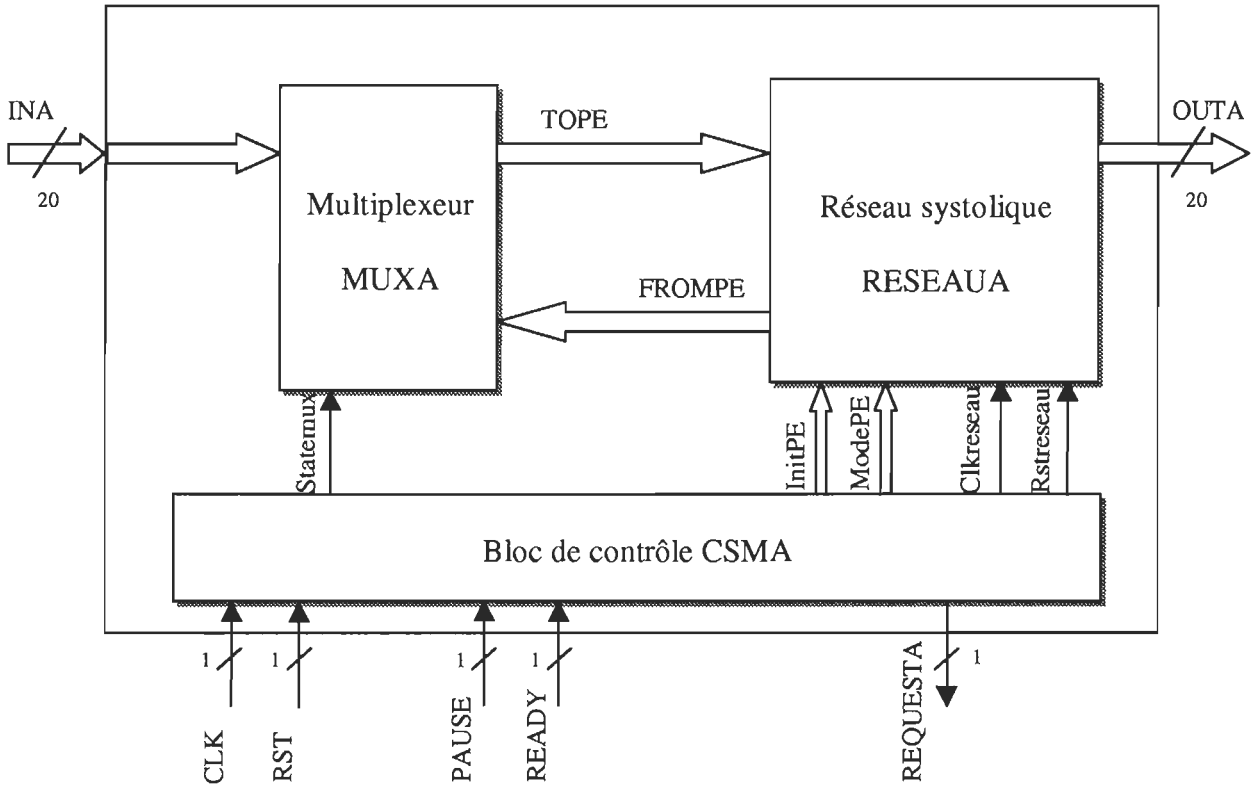


Figure 5-2 : Contrôleur de la charge PROCA, deuxième niveau de décomposition

Les signaux externes mis en jeu sont :

- INA : bus d'entrée du contrôleur PROCA;
- OUTA : bus de sortie du contrôleur PROCA;
- CLK : horloge externe;
- RST : reset externe;
- PAUSE : signal de pause du contrôleur PROCA;
- READY : signal permettant au contrôleur de savoir si les données requises sont prêtes à être lues;
- REQUESTA : signal de requête des entrées.

Les signaux internes sont :

- TOPE : données envoyées du multiplexeur au réseau systolique (6 bus de 20 bits);

- FROMPE : données envoyées du réseau systolique au multiplexeur (6 bus de 20 bits);
- Statemux : état du multiplexeur lui permettant de savoir comment positionner ses sorties en fonction de ses entrées;
- INITPE : signaux d'initialisation des processeurs élémentaires du réseau (5×1 bit);
- MODEPE : modes de fonctionnement des processeurs élémentaires;
- Clkreseau : horloge du réseau;
- Rstreseau : reset du réseau;

Le réseau systolique est synchronisé par le signal d'horloge qui lui est fourni par le bloc de contrôle. Le multiplexeur est de type combinatoire, les sorties sont modifiées dès qu'une ou plusieurs entrées sont modifiées. Les différentes architectures proposées sont conçues pour limiter au maximum le stockage des données et donc limiter le nombre de mémoires utilisées.

5.2.2. Le réseau systolique

Nous ajoutons un troisième niveau de décomposition pour le réseau systolique. Les réseaux systoliques sont des réseaux réguliers de processeurs relativement simples appelés processeurs élémentaires (PE) réalisés sous forme de circuits intégrés [KUN82] [QUI89]. Les principaux avantages des architectures systolique résident dans leur régularité, leur simplicité, leur parallélisme et la localité des communications de données. Le réseau systolique que nous utilisons dans le contrôleur a trois fonctions de calcul :

- estimation des paramètres de la charge selon les équations (4.27) à (4.29),
- calcul du couple de charge idéal selon l'équation (3.8),
- génération de trajectoire désirée pour le moteur selon les équations (3.10), (3.15) et (3.16).

Ce réseau systolique est divisé en deux parties. Un réseau triangulaire réalisant l'estimation des paramètres, et un processeur élémentaire supplémentaire pour le calcul du couple de charge et la génération de trajectoire du moteur.

5.2.2.1. Estimation des paramètres

Nous rappelons que l'algorithme d'estimation des paramètres est basé sur le filtre de Kalman à racine carrée de covariance et a été décrit dans la section 4.5 du quatrième chapitre. Nous pouvons constater que les équations de l'algorithme proposé sont principalement composées d'une triangularisation de matrice (4.28) basée sur les rotations de Givens.

Dans notre architecture nous utiliserons un réseau systolique réalisant la triangularisation de matrices et inspiré d'un réseau développé par Gentleman et Kung [QUI89]. Les autres calculs pour l'estimation de paramètres seront également réalisés par ce réseau.

Le réseau est constitué, pour un problème de dimension M, de (M+1)(M+2)/2 processeurs connectés selon une topologie triangulaire comme le montre la figure 5-3 pour le cas où M=4, ce qui sera notre cas. Nous nommons les processeurs élémentaires en utilisant leurs numéros de ligne et de colonne dans le réseau tel qu'illustré par la figure 5-3.

Nous distinguons dans ce réseau deux types de cellules, les cellules rondes et les cellules carrées. Les cellules rondes génèrent les rotations de Givens et les cellules carrées les appliquent sur les coefficients de la matrice à triangulariser. Plus précisément, les cellules rondes génèrent les cosinus et sinus utilisés pour la rotation. Les figures 5-4 et 5-5 décrivent la structure et le comportement des deux types de cellules utilisées.

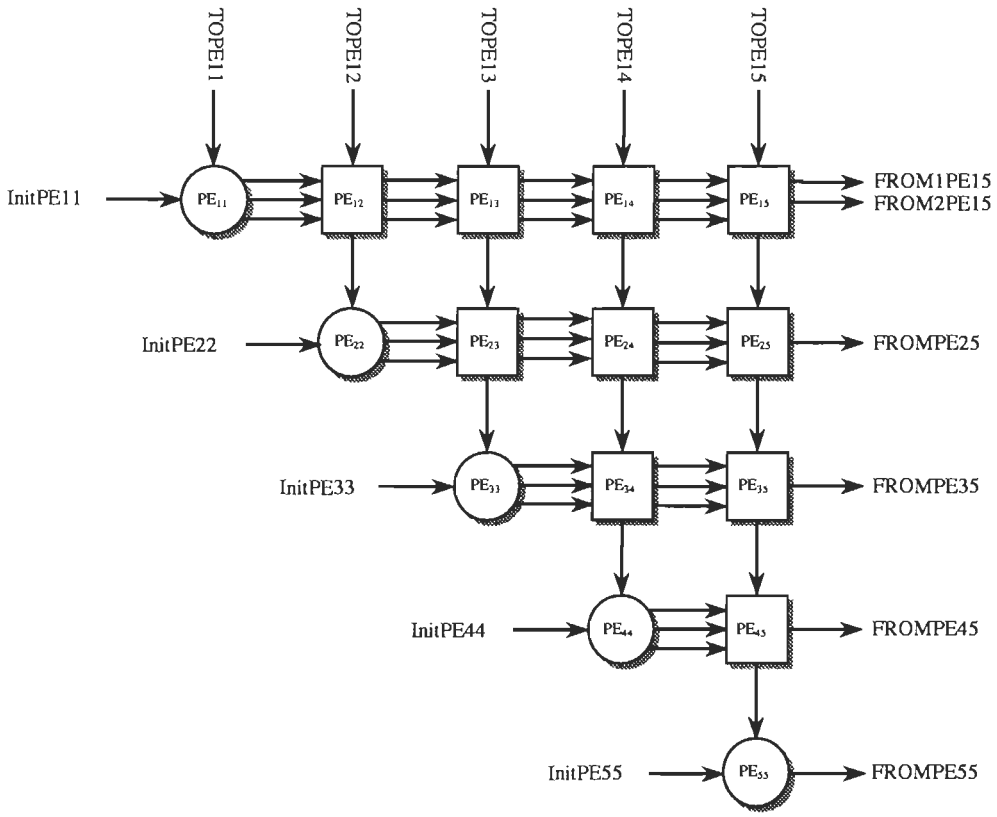
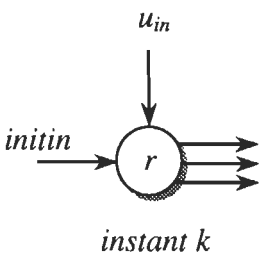


Figure 5-3 : Réseau systolique pour l'estimation de paramètres (M=4)



Si initin=1

$r = u_{in}$

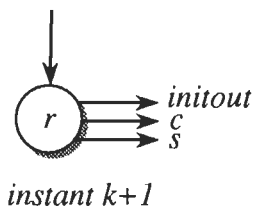
Sinon

Si $u_{in}=0$ $\quad \begin{cases} c = 1 \\ s = 0 \end{cases}$

Sinon

Si $|u_{in}| \geq |r|$ $\quad \begin{cases} t = r / u_{in} \\ s = 1/\sqrt{1+t^2} \\ c = st \end{cases}$

Sinon $\quad \begin{cases} t = u_{in} / r \\ c = 1/\sqrt{1+t^2} \\ s = ct \end{cases}$



initout = initin

Figure 5-4 : Fonctionnement des cellules rondes pour la triangularisation

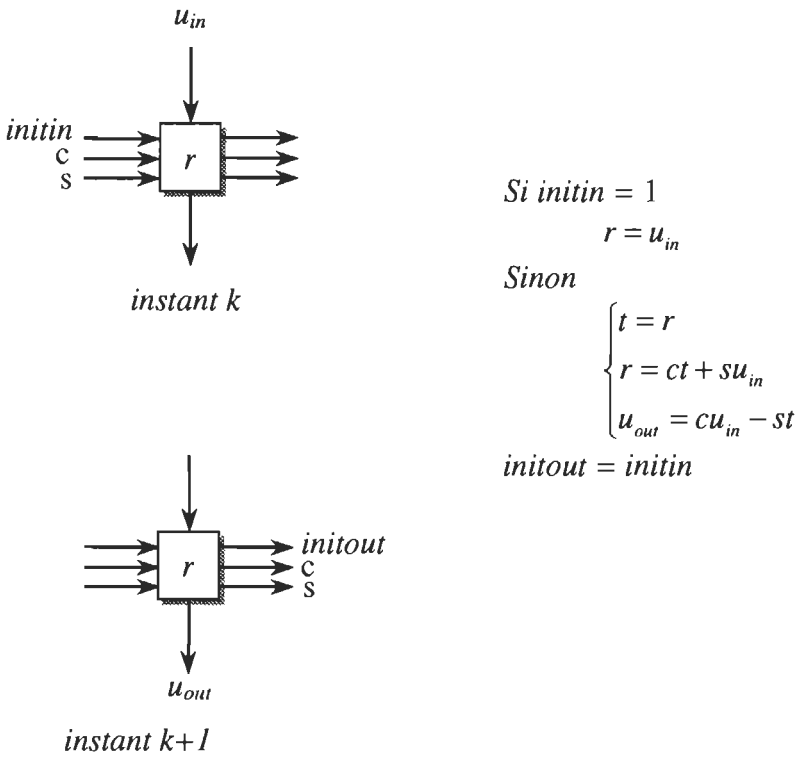


Figure 5-5 : Fonctionnement des cellules carrées pour la triangularisation

Les équations données dans les figures 5-4 et 5-5 décrivent le comportement des processeurs élémentaires dans le cas où ceux-ci réalisent la factorisation de Givens. Pour réaliser les autres opérations de l'algorithme, les mêmes cellules seront utilisées mais avec des modes de fonctionnement différents. La différenciation des modes de fonctionnement sera dictée par les signaux provenant du bloc de contrôle suivant le type d'opération à réaliser (signal ModePE). Ceci permet de n'utiliser qu'un seul réseau pour effectuer tous les calculs. Le réseau triangulaire sera utilisé à la fois pour réaliser la triangularisation de l'équation (4.28) et pour les calculs de mise à jour des équations (4.27) et (4.29). Le séquencement des opérations dans le réseau pour le filtre de Kalman à racine de covariance a été inspiré de [MOZ99]. Pour simplifier l'écriture nous utilisons les notations $\boldsymbol{S}_{(k)}^{+}$, $\boldsymbol{A}$, I et $\boldsymbol{\Phi}_{r(k)}$ décrites par les équations (5.1) à (5.4).

$$S^{+}_{(k)} = \left(1/\lambda_{(k)}\right) S^{T}_{(k/k)} \tag{5.1}$$

$$A_{(k)} = \begin{bmatrix} \beta & 0^{T} \\ S^{T}_{(k/k-1)} \Phi_{(k+1)} & S^{T}_{(k/k-1)} \end{bmatrix} \tag{5.2}$$

$$I_{(k)} = \tilde{\alpha}_{k} - \hat{W}^{T}_{(k-1)} \Phi_{(k)} \tag{5.3}$$

$$\Phi_{r(k)} = \left[\tau^{*}_{T}; \omega_{L(k)}\right] \tag{5.4}$$

La figure 5-6 donne la représentation de l'architecture utilisée et du flot de données pour le calcul des équations du filtre de Kalman à racine carrée de covariance.

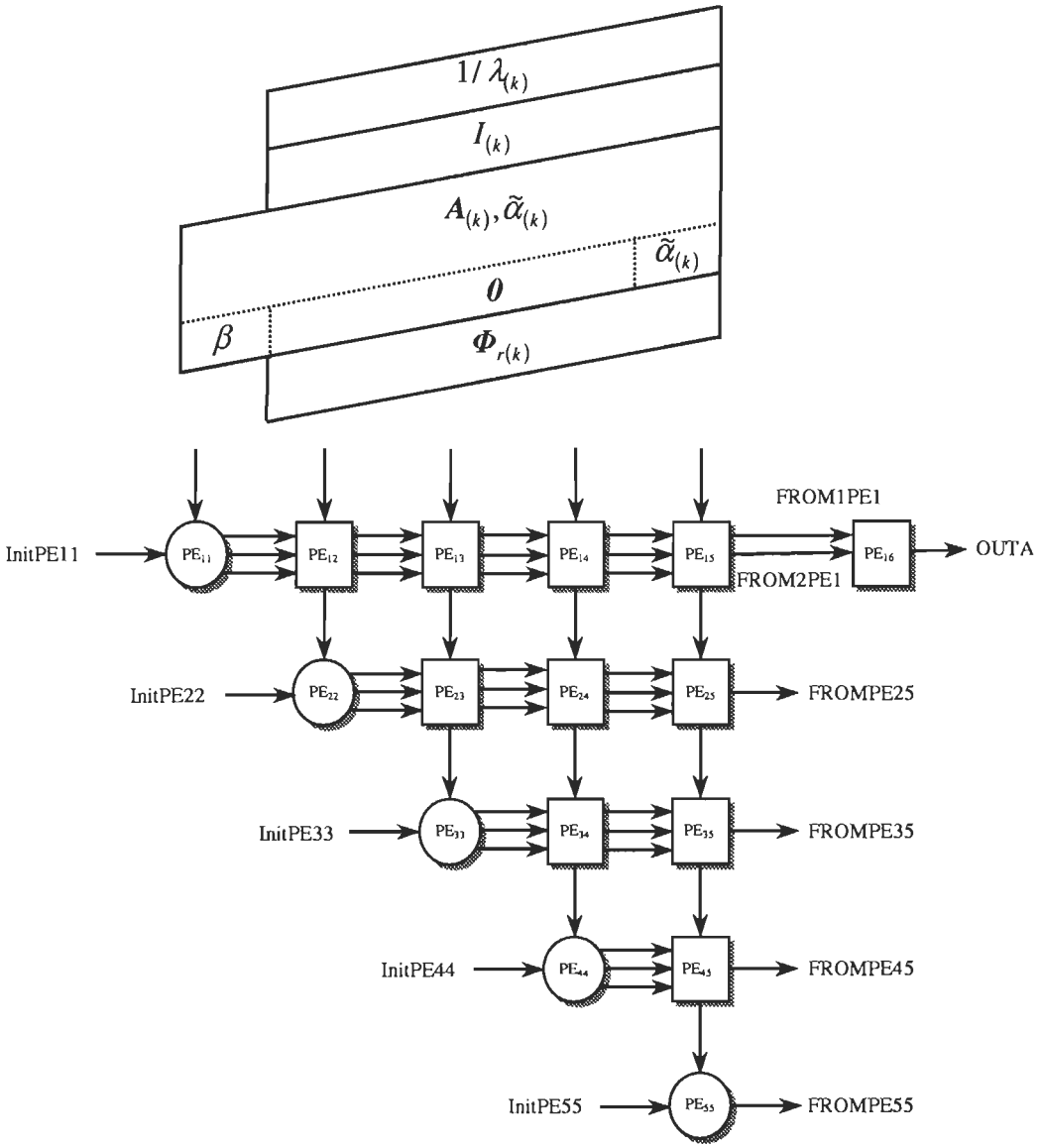


Figure 5-6 : Architecture et flot de données de l'estimateur de paramètres

Nous utilisons le vecteur $\boldsymbol{\Phi}_{r(k)}$ puisque les entrées disponibles pour calculer le vecteur $\boldsymbol{\Phi}$ sont le couple et la vitesse. Les quatre coefficients du vecteur $\boldsymbol{\Phi}$ seront calculés par les processeurs élémentaires à partir de ces deux entrées. Le calcul des paramètres estimés nécessite M+1 cycles soit 5 cycles d'horloge pour l'initialisation du réseau et 3M+1 cycles soit 13 cycles pour chaque nouvelles série de paramètres.

Les différentes étapes pour le calcul du vecteur $\hat{W}_{(k)}$ sont les suivantes :

- **Étape 1**, cycles 1 à 4 : $\boldsymbol{\Phi}_{r(k)}$ entre dans les processeurs PE_{12} et PE_{13}. Les paramètres estimés de l'instant précédent sont stockés dans les quatre derniers processeurs de la première ligne. PE_{12} calcule $\Phi_{(k)}(1)\hat{W}_{(k-1)}(1)$ et le transmet à droite à PE_{13}. PE_{13} reçoit $\omega_{L(k)}$, évalue $\Phi_{(k)}(2)$, calcule $\Phi_{(k)}(1)\hat{W}_{(k-1)}(1)+\Phi_{(k)}(2)\hat{W}_{(k-1)}(2)$ et le transmet à droite ainsi que $\omega_{L(k)}$ vers PE_{14}. PE_{14} et PE_{15} réalisent les mêmes opérations en générant $\Phi_{(k)}(3)$ et $\Phi_{(k)}(4)$ à partir de $\omega_{L(k)}$. Le calcul de $\hat{W}^T_{(k-1)}\Phi_{(k)}$ est achevé par PE_{15} et stocké dans ce processeur au cycle 4.

- **Étape 2**, cycles 2 à 9 : Les éléments du vecteur $\boldsymbol{\Phi}_{(k)}$ sont transmis successivement dans le réseau triangulaire inférieur composé des quatre dernières lignes et colonnes. La matrice $\boldsymbol{S}^T_{(k/k-1)}$est stockée dans cette partie du réseau. Au fur et à mesure qu'ils sont transmis dans le réseau, les éléments de $\boldsymbol{\Phi}_{(k)}$ sont multipliés et accumulés avec les éléments de $\mathbf{S}^T_{(k/k-1)}$. Les quatre éléments de $\boldsymbol{S}^T_{(k/k-1)}\boldsymbol{\Phi}_{(k)}$ sont finalement sortis par les processeurs de la dernière colonne. Ces résultats sont réintroduits dans le réseau pour former la matrice $\boldsymbol{A}$.

- **Étape 3**, cycles 4 à 16 : la matrice $\boldsymbol{A}$ entre par la haut dans les 5 dernières colonnes, et est triangularisée. Le fonctionnement des processeurs pour cette étape ont été donnés dans les figures 5-4 et 5-5. Le résultat de triangularisation est stocké dans le réseau. Ainsi, PE_{11} contient $f_{(k)}$, les autres processeurs de la première ligne contiennent les éléments de $\boldsymbol{g}_{(k)}^T$ et les processeurs du réseau triangulaire inférieur contiennent les éléments de la nouvelle matrice $\boldsymbol{S}_{(k/k)}^T$.

- **Étape 4**, cycle 6 : $\tilde{\alpha}_{(k)}$ entre dans PE_{15} qui contient $\hat{\boldsymbol{W}}_{(k-1)}^T \boldsymbol{\Phi}_{(k)}$ et calcule l'innovation $I_{(k)}$ d'après l'équation (5.3).

- **Étape 5**, cycles 10 à 14 : PE_{11} calcule $f_{(k)}^{-1}$ qui sera transmis par la droite à PE_{12}. PE_{12} reçoit $f_{(k)}^{-1}$ et l'innovation $I_{(k)}$ pour calculer la nouvelle valeur de $\hat{W}_{(k)}(1)$ conformément à l'équation (4.29) : $\hat{W}_{(k)}(1) = f_{(k)}^{-1} g_{(k)}^T(1) I_{(k)} + \hat{W}_{(k-1)}(1)$. PE_{12} transmet ensuite $f_{(k)}^{-1}$ à PE_{13} qui calcule $\hat{W}_{(k)}(2)$ de façon analogue. PE_{14} et PE_{15} effectuent ensuite la même opération et le calcul des quatre nouveaux paramètres estimés est terminé au cycle 14.

- **Étape 6**, cycles 10 à 19 : PE_{15} calcule $1/\lambda_{(k)}$ en fonction de l'innovation. Le coefficient $1/\lambda_{(k)}$ est transmis par le haut aux quatre dernières colonnes du réseau et traverse la première ligne. Dans le réseau triangulaire inférieur il rencontre et multiplie chacun des coefficients de la matrice $\boldsymbol{S}_{(k/k)}^T$ pour donner la nouvelle matrice $\boldsymbol{S}_{(k+1)}^+$ conformément à l'équation (5.1).

Nous avons vu au quatrième chapitre que les paramètres obtenus ne sont pas directement les paramètres recherchés (inerties et coefficients de friction). Nous devons donc

effectuer quelques opérations supplémentaires pour finalement obtenir les paramètres recherchés. Le premier paramètre obtenu et stocké dans PE_{12} est l'inverse de l'inertie. Au cycle 1, PE_{12} calcule $\hat{J}_{L(k)} = 1/\hat{W}_{(k)}(1)$ et le transmet à PE_{13}. Le deuxième paramètre stocké dans PE_{13} est le coefficient de frottement de Coulomb estimé divisé par l'inertie. PE_{13} multiplie donc ce paramètre par l'inertie estimée pour obtenir la valeur de $\hat{F}_{c(k)}$ au cycle 2. PE_{13} transmet $\hat{J}_{L(k)}$ à PE_{14} qui effectue la même opération, de même pour PE_{15}. Finalement, les paramètres recherchés sont sortis du réseau aux cycles 4, 5, 6 et 7.

5.2.2.2. Calcul du couple de charge et génération de trajectoire pour le moteur

Lorsque les paramètres estimés sont obtenus, le couple idéal de charge et la trajectoire désirée pour le moteur peuvent être calculés d'après les équations vues au troisième chapitre. Nous rappelons ces équations :

$$\tau_T^* = \frac{1}{N}\left[\hat{J}_L \alpha_L^* + \hat{\tau}_{F_L} + K_{V_L}\left(\omega_L^* - \omega_L\right) + K_{P_L}\left(\theta_L^* - \theta_L\right)\right]$$

$$\theta_M^* = \frac{\tau_T^*}{k} + N\theta_L^*$$

$$\omega_M^* = N\omega_L^* + \frac{1}{k}K_{P_L}\left(\omega_L^* - \omega_L\right) + \frac{1}{k}K_{V_L}\left(\alpha_L^* - \alpha_L\right)$$

$$\alpha_M^* = N\alpha_L^* + \frac{1}{k}K_{P_L}\left(\alpha_L^* - \alpha_L\right)$$

Comme nous l'avons vu au paragraphe 3.3.4.1, les gains du correcteur PD de la charge sont calculés de façon adaptative pour assurer un bon placement de pôles.

Les expressions obtenues pour les gains ont été donné dans l'équation (3.64) :

$$\begin{cases} K_{V_L} = -2\hat{J}_L p_L \\ K_{P_L} = p_L^2 \hat{J}_L \end{cases}$$

Nous pouvons constater que le calcul des gains proportionnels et dérivatifs dépend de l'inertie estimée de la charge.

Posons :

$$K_{V_L} = -2p_L \tag{5.5}$$

$$K_{P_L} = p_L^2 \tag{5.6}$$

K_{V_L} et K_{P_L} sont des constantes définies lors de la conception de la loi de commande. Nous pouvons alors factoriser l'inertie estimée dans les équations de calcul du couple de charge comme le montre l'équation (5.7).

$$\tau_T^* = \frac{1}{N}\left[\hat{J}_L\left(\alpha_L^* + K_{V_L}\left(\omega_L^* - \omega_L\right) + K_{P_L}\left(\theta_L^* - \theta_L\right)\right) + \hat{\tau}_{F_L}\right] \tag{5.7}$$

Nous limiterons ainsi à une multiplication par l'inertie estimée. Nous procédons à la même factorisation pour l'évaluation des vitesses et accélérations désirées du moteur :

$$\omega_M^* = N\omega_L^* + \frac{1}{k}\hat{J}_L\left(K_{P_L}\left(\omega_L^* - \omega_L\right) + \frac{1}{k}K_{V_L}\left(\alpha_L^* - \alpha_L\right)\right) \tag{5.8}$$

$$\alpha_M^* = N\alpha_L^* + \frac{1}{k}\hat{J}_L K_{P_L}\left(\alpha_L^* - \alpha_L\right) \tag{5.9}$$

Pour effectuer ces calculs, nous avons ajouté au réseau systolique un processeur élémentaire nommé PE_{16}.

La figure 5-7 donne une représentation de ce processeur élémentaire avec ses entrées et sorties.

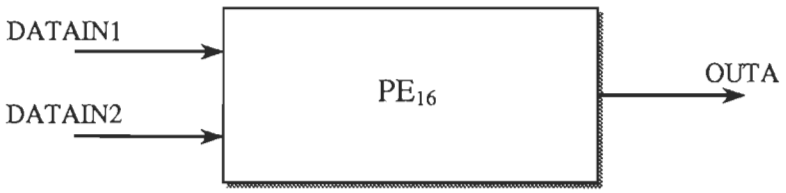


Figure 5-7 : Structure du processeur PE_{16}

Nous devons répartir les calculs à effectuer par ce processeur en synchronisant les calculs avec la sortie des paramètres estimés du réseau triangulaire.

Le processeur PE_{16} contient cinq registres dans lesquels seront stockés les résultats intermédiaires :

- $reg\tau_T$: pour le calcul de τ_T^*,
- $reg\tau_{F_L}$: pour le calcul du couple de frottements estimé $\hat{\tau}_{F_L}$,
- $reg\theta_M^*$: pour la position désirée du moteur,
- $reg\omega_M^*$: pour la vitesse désirée du moteur,
- $reg\alpha_M^*$: pour l'accélération désirée du moteur.

Les calculs seront effectués à l'intérieur des 13 cycles de la façon suivante :

- Cycle 1 : lecture de ω_L.
- Cycle 2 : lecture de ω_L^*, calcul de $K_{V_L}(\omega_L^* - \omega_L)$ stocké dans $reg\tau_T$ et calcul de $(1/k)K_{P_L}(\omega_L^* - \omega_L)$ stocké dans $reg\omega_M^*$.
- Cycle 3 : lecture de θ_L.

- Cycle 4 : lecture de θ_L^*, calcul de $K_{P_L}(\theta_L^* - \theta_L)$ accumulé dans $reg\tau_T$ et calcul de $N\theta_L^*$ stocké dans $reg\theta_M^*$.
- Cycle 5 : lecture de α_L^* et $\hat{J}_L$, calcul de $\hat{J}_L(\alpha_L^* + reg\tau_T)$ stocké dans $reg\tau_T$, calcul de $N\alpha_L^*$ stocké dans $reg\alpha_M^*$.
- Cycle 6 : lecture de $\hat{F}_{c_L}$ et de α_L, calcul de $\hat{F}_{C_L} sign(\omega_L)$ accumulé dans $reg\tau_T$, calcul de $N\omega_L^* + \hat{J}_L((1/k)K_{V_L}(\alpha_L^* - \alpha_L) + reg\omega_M^*)$ stocké dans $reg\omega_M^*$ et calcul de $(1/k)\hat{J}_L K_{P_L}(\alpha_L^* - \alpha_L) + reg\alpha_M^*$.
- Cycle 7 : lecture de $\hat{F}_{V_L}$ et calcul de $\hat{F}_{V_L}\omega_L$ accumulé dans $reg\tau_T$.
- Cycle 8 : lecture de $\hat{F}_{S_L}$, calcul de $\hat{F}_{S_L} sign(\omega_L)\exp(-(\omega_L/\omega_s)^2)$ accumulé dans $reg\tau_T$ et calcul de $reg\tau_T/k$ accumulé dans $reg\theta_M^*$.

Les variables ω_M^* et α_M^* sont disponible dès le cycle 3 tandis que τ_T^* et θ_M^* sont disponibles dès le cycle 5. Ces données seront envoyées directement au processeur PE_{16} du contrôleur du moteur qui calculera le couple moteur de façon analogue.

5.3. Architecture du contrôleur du moteur

5.3.1. Structure générale

Le contrôleur du moteur a une structure très similaire à celle du contrôleur de la charge. La figure 5-8 donne une représentation schématique des données d'entrée/sortie du contrôleur de la charge pour un premier niveau de décomposition.

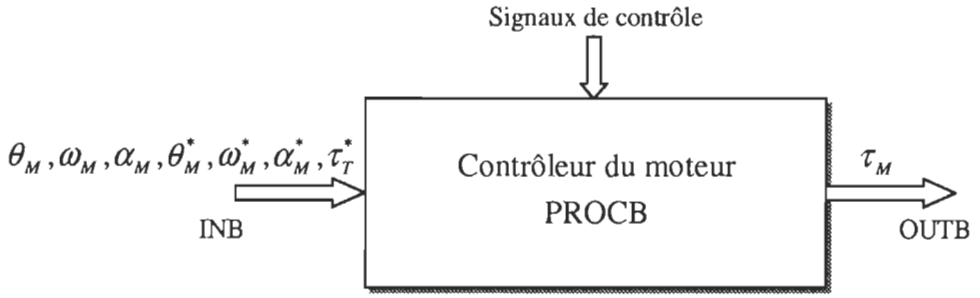


Figure 5-8 : Structure globale du contrôleur du moteur

Le contrôleur utilise les trajectoires réelles et désirées du moteur ainsi que le couple idéal de charge pour calculer le couple à appliquer au moteur selon l'équation (3.17). La figure 5-9 représente le deuxième niveau de décomposition du contrôleur du moteur, où nous retrouvons les mêmes blocs que pour la charge.

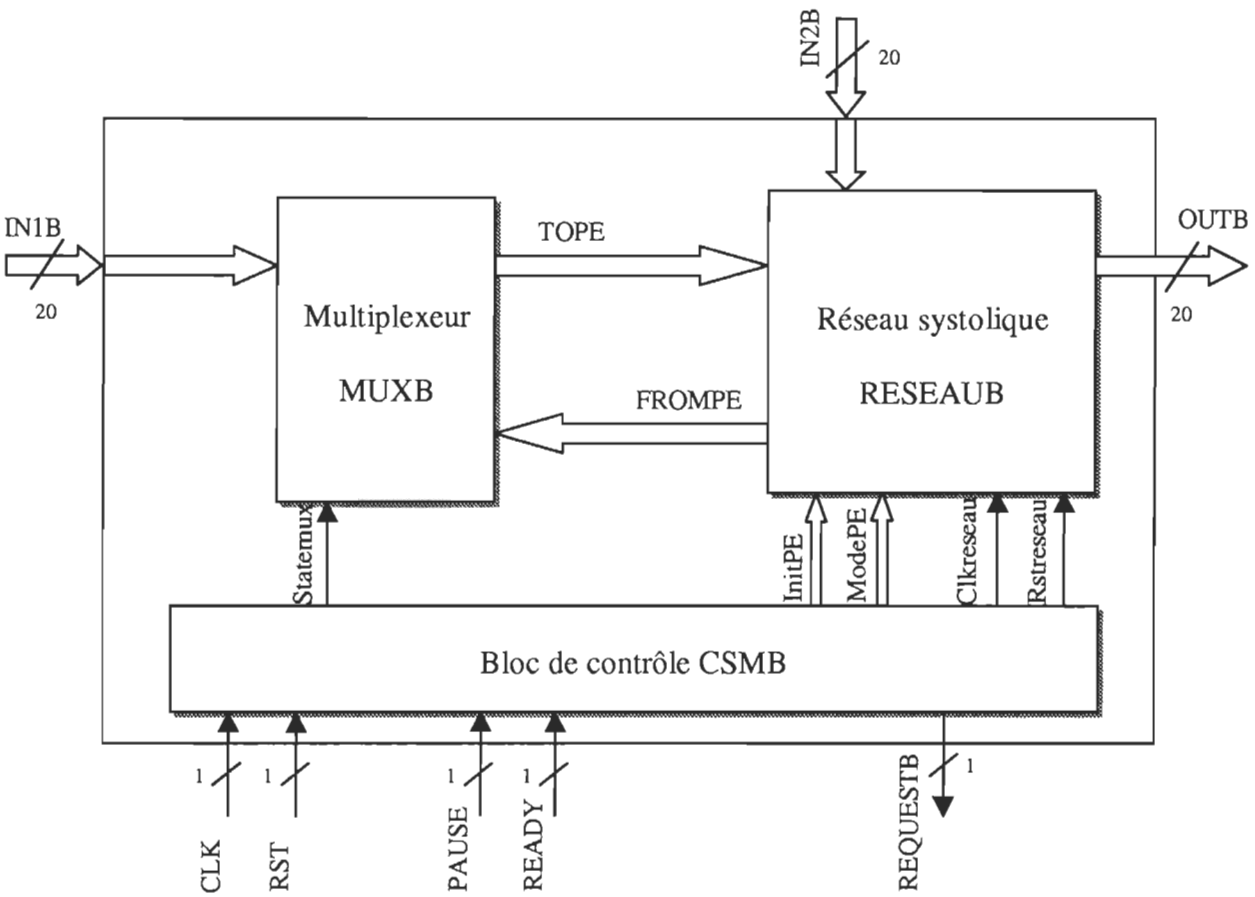


Figure 5-9 : Contrôleur du moteur PROCB, deuxième niveau de décomposition

Nous retrouvons les mêmes blocs et les mêmes signaux que ceux du contrôleur de la charge PROCA décrit à la figure 5-2, mais le contrôleur du moteur a deux bus d'entrées :

- IN1B : bus des entrées externes,
- IN2B : bus des entrées provenant du processeur PE_{16} du contrôleur de la charge.

5.3.2. Le réseau systolique

5.3.2.1. Estimation des paramètres

L'estimation des paramètres du moteur est effectuée par un réseau systolique identique à celui décrit pour le contrôleur de la charge avec les entrées correspondant au moteur.
Le déroulement des opérations s'effectue à l'intérieur des 13 cycles et rigoureusement en même temps que l'estimation des paramètres de la charge. Le séquencement des opérations suivant les cycles est donc exactement le même et a été décrit au paragraphe 5.2.2.1.

5.3.2.2. Calcul du couple moteur

Comme pour le contrôleur de la charge, nous avons ajouté au réseau systolique un processeur élémentaire nommé PE16 qui effectue le calcul du couple moteur τ_M.
Nous rappelons l'expression du couple moteur donnée au chapitre 3 :

$$\tau_M = \hat{J}_M \alpha_M^* + \hat{\tau}_{F_M} + \eta \tau_T^* + K_{V_M}\left(\omega_M^* - \omega_M\right) + K_{P_M}\left(\theta_M^* - \theta_M\right)$$

Nous procédons à la factorisation de l'inertie dans l'expression du couple moteur :

$$\tau_M = \hat{J}_M\left(\alpha_M^* + K_{V_{Mr}}\left(\omega_M^* - \omega_M\right) + K_{P_{Mr}}\left(\theta_M^* - \theta_M\right)\right) + \hat{\tau}_{F_M} + \eta\tau_T^* \tag{5.10}$$

Avec :

$$K_{V_{Mr}} = -2p_M \tag{5.11}$$

$$K_{P_{Mr}} = p_M^2 \tag{5.12}$$

Le processeur PE16 contient cinq registres dans lesquels seront stockés les résultats intermédiaires :

- $reg\tau_M$: pour le calcul de τ_M,

- $reg\tau_{F_M}$: pour le calcul du couple de frottements $\hat{\tau}_{F_M}$.

Les calculs seront effectués à l'intérieur des 13 cycles de la façon suivante :

- Cycle 5 : lecture de $\hat{J}_M$.
- Cycle 6 : lecture de ω_M et de $\hat{F}_{c_M}$, calcul de $\hat{F}_{c_M} sign(\omega_M)$ stocké dans $reg\tau_{F_M}$.
- Cycle 7 : lecture de ω_M^* et de $\hat{F}_{v_M}$, calcul de $\hat{F}_{v_M}\omega_M$ accumulé dans $reg\tau_{F_M}$ et calcul de $K_{V_{Mr}}\left(\omega_M^* - \omega_M\right)$ stocké dans $reg\tau_M$.
- Cycle 8 : lecture de $\hat{F}_{s_M}$ et calcul de $\hat{F}_{s_M} sign(\omega_M)\exp\left(-(\omega_M / \omega_s)^2\right)$ accumulé dans $reg\tau_{F_M}$.
- Cycle 9 : lecture de θ_M et θ_M^*, calcul de $K_{P_{Mr}}\left(\theta_M^* - \theta_M\right)$ accumulé dans $reg\tau_M$.
- Cycle 10 : lecture de α_M^* et calcul de $\hat{J}_M\left(\alpha_M^* + reg\tau_M\right) + reg\tau_{F_M}$ stocké dans $reg\tau_M$.
- Cycle 11 : lecture de τ_T^* et calcul final de $\tau_M = reg\tau_M + \eta\tau_T^*$.

5.4. Architecture du contrôleur complet

L'architecture du contrôleur complet de l'articulation est composée des deux architectures présentées aux paragraphes 5.2 et 5.3. La figure 5-10 donne une représentation schématique de cette architecture.

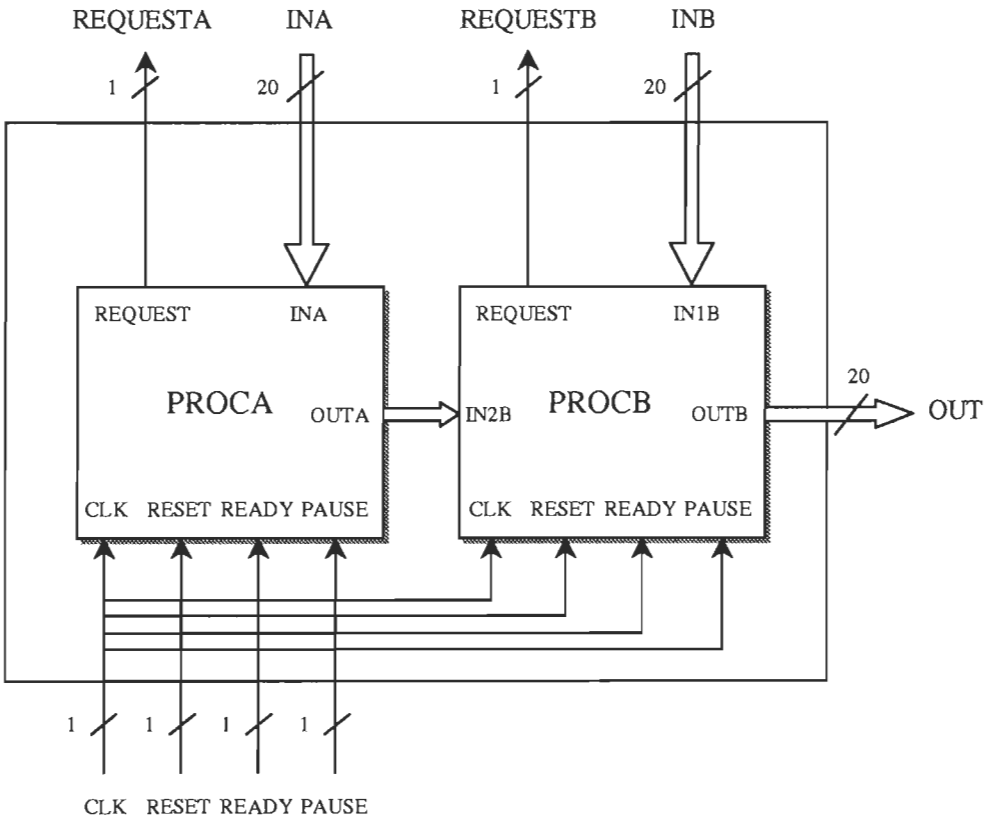


Figure 5-10 : Schéma structurel du contrôleur complet de l'articulation

Nous avons donc pour notre processeur un total de 66 broches auquel nous devrons ajouter les broches d'alimentation et de test. La latence de l'architecture est de $(3M+1)/f$ où f est la fréquence d'horloge interne. Le débit est égal à l'inverse de la latence.

5.5. Résultats de simulations des modèles VHDL

Les différentes architectures présentées ont été modélisées et simulées en langage de haut niveau VHDL dans le logiciel Mentor Graphics. Les blocs de contrôle ont été modélisés par des machines à états et réalisées dans System Architect.

Pour pouvoir évaluer la fonctionnalité des architectures développées, nous avons simulé l'architecture du contrôleur complet en boucle ouverte. Pour cela, nous avons utilisés les données obtenues lors des simulations du système en boucle fermée dans Matlab® et nous avons injecté ces entrées dans notre architecture.

Les figures 5-11 à 5-14 présentent les résultats d'estimation obtenus pour les inerties et couples de friction de la charge et du moteur.

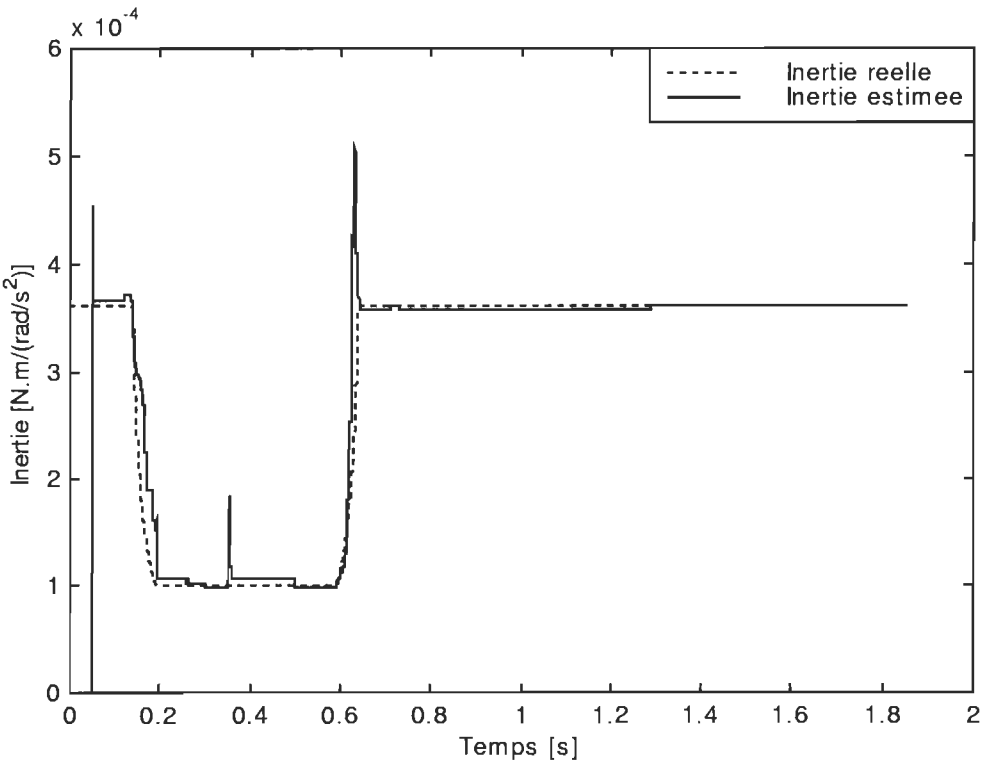


Figure 5-11 : Inerties réelles et estimées de la charge

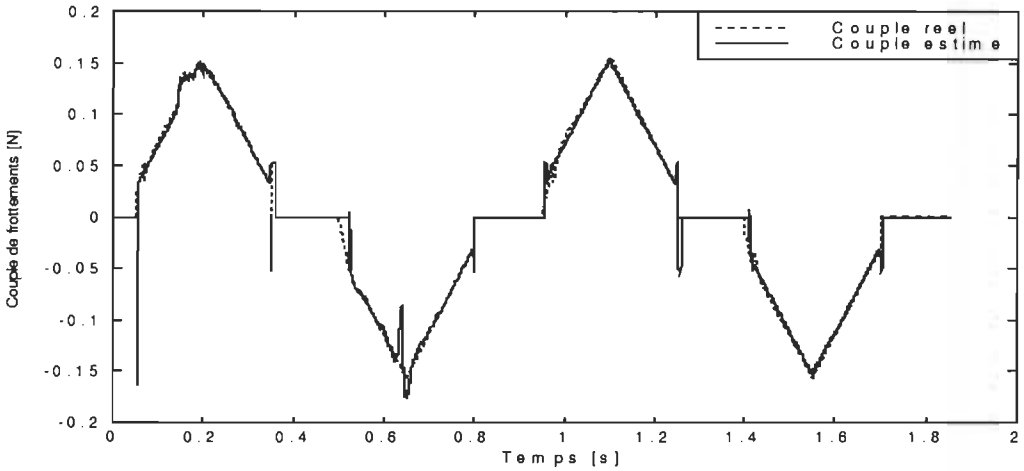


Figure 5-12 : Couples de frottements réels et estimés de la charge

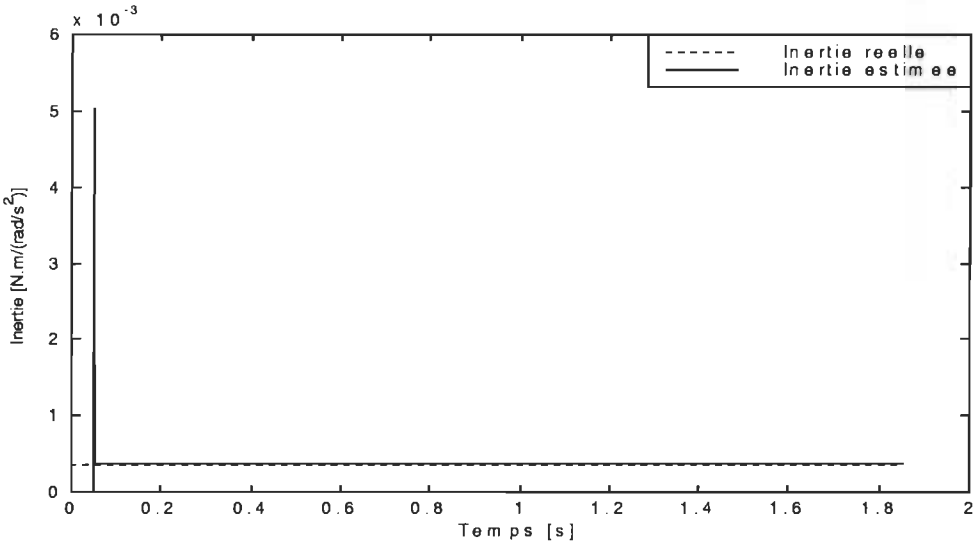


Figure 5-13 : Inerties réelles et estimées du moteur

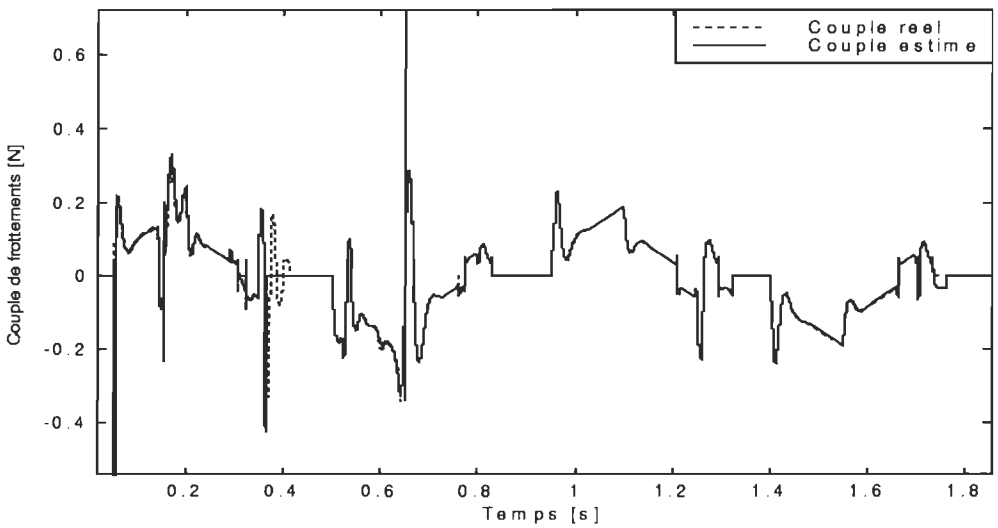


Figure 5-14 : Couples de frottements réels et estimés du moteur

Les résultats obtenus permettent de confirmer la fonctionnalité des architectures développées pour l'estimation des paramètres de la charge et du moteur. Les figures 5-15 à 5-19 présentent les résultats obtenus pour la génération de trajectoire du moteur (position et vitesse) et pour les couples calculés (moteur et charge).

Pour ces courbes, nous prenons comme références les résultats obtenus lors des simulations du système complet en virgule flottante dans Matlab®.

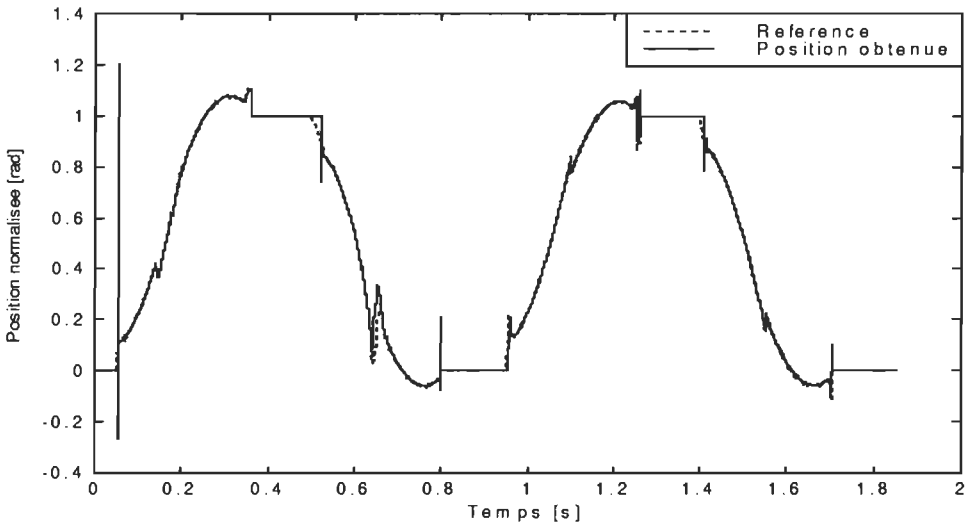


Figure 5-15 : Position désirée générée pour le moteur

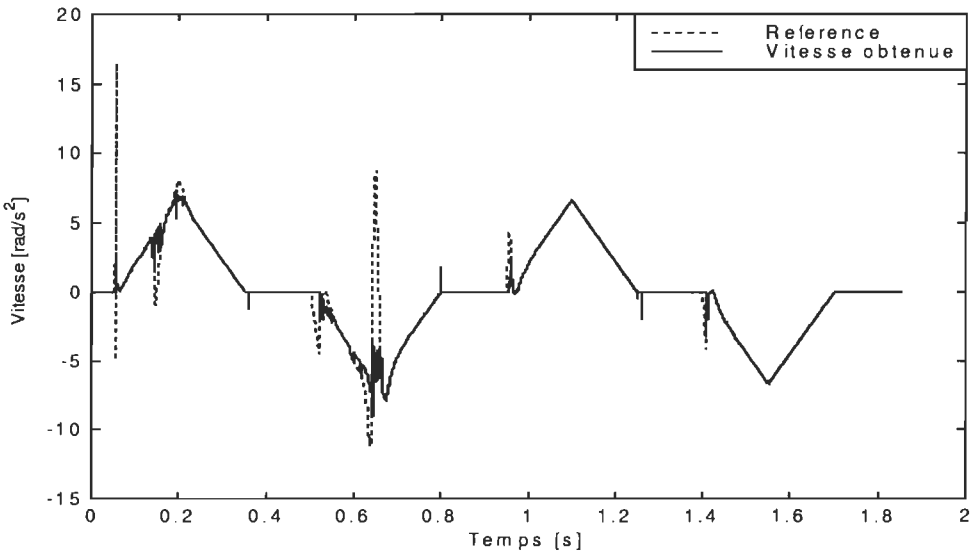


Figure 5-16 : Vitesse désirée générée pour le moteur

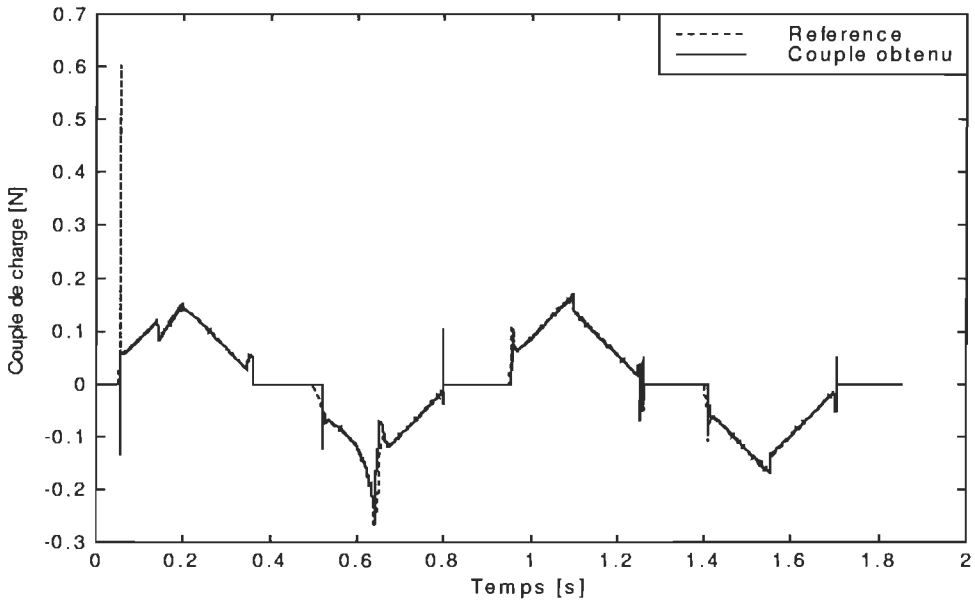


Figure 5-17 : Couple de charge

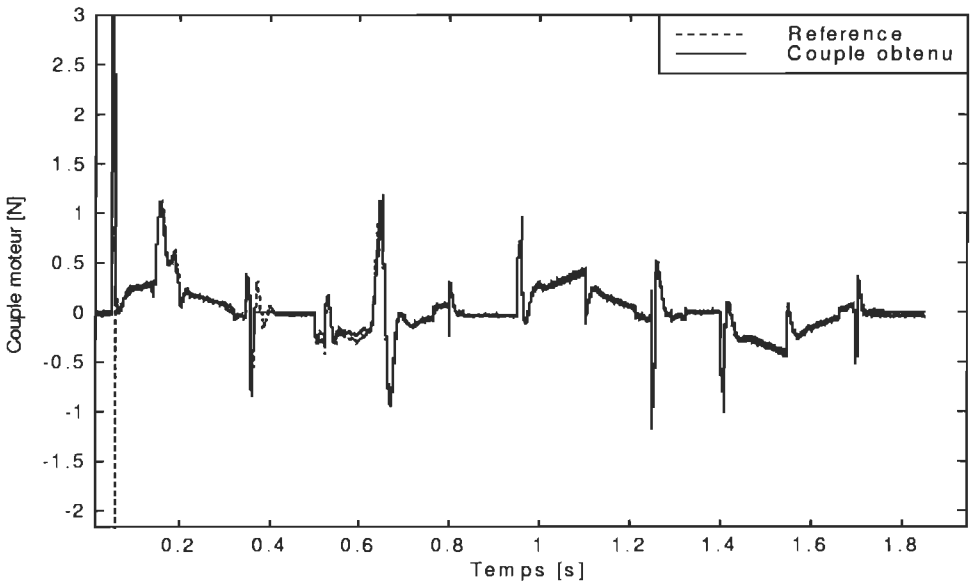


Figure 5-18 : Couple moteur

Pour tous ces signaux, les résultats obtenus par la simulation des modèles VHDL de l'architecture sont très proches des résultats que nous avions obtenus dans Matlab®. Le tableau 5-1 présente les erreurs absolues moyennes calculées entre les signaux obtenus et les signaux de référence.

Tableau 5-1 : Erreurs absolues moyennes obtenues pour la simulation des modèles VHDL de l'architecture du contrôleur

Signal	Erreur absolue moyenne
Inertie estimée de la charge	1.59×10^{-5}
Couple de friction estimé de la charge	2.30×10^{-3}
Inertie estimée du moteur	1.11×10^{-5}
Couple de friction estimé du moteur	9.51×10^{-3}
Position désirée du moteur	$5.50.\times10^{-3}$
Vitesse désirée du moteur	2.49×10^{-2}
Couple de charge	2.70×10^{-3}
Couple moteur	3.20×10^{-3}

Le couple de commande du moteur obtenu par simulation de l'architecture du contrôleur complet est proche du couple de référence ce qui nous assure d'obtenir de bons résultats pour le suivi de trajectoire de la charge. Les différents programmes de modélisation en langage VHDL des architectures développées sont fournis en annexe E.

5.6. Résultats de synthèse dans une technologie 0.5µm

L'architecture du processeur complet a été synthétisée avec Synopsys suivant un taux d'optimisation faible dans une technologie CMOS Hewlet-Packard de 0.5µm disponible de MOSIS (*MOS Integrated services*) et obtenu par la Société Canadienne de Microélectronique (SCM - *CMC - Canadian Microelectronic Corporation*).

Le tableau 5-2 donne la répartition du nombre de transistors pour l'intégration.

Tableau 5-2 : Répartition du nombre de transistor dans l'architecture

Élément	Nombre de transistors
Bloc de contrôle	20 000
Multiplexeur	10 000
PE ronds	10 000
PE carrés : PE_{15} et PE_{16}	16 000
PE carrés : PE_{12}, PE_{13} et PE_{14}	14 400
Autres PE carrés	5 000

L'architecture synthétisée comprend environ 400 000 transistors et a une fréquence maximale de fonctionnement de 4MHz. À cette fréquence, le temps de calcul de la commande est de 3µs ce qui est environ 27 fois plus faible que le temps de calcul estimé à la section 4.6 pour un DSP avec une fréquence d'horloge 12 fois plus élevée.

5.7. Conclusion

L'étude algorithmique a permis de mettre au point un algorithme d'estimation des paramètres de l'articulation dont la formulation a facilité le développement d'une architecture pour son implantation. Les résultats de simulations démontrent que les performances de la loi de commande demeurent satisfaisantes une fois que celle-ci est implantée dans notre architecture.

La latence, pour un problème de dimension M est de $(3M+1)/f$ où f est la fréquence d'horloge de notre processeur. Le débit de l'architecture est égal à l'inverse de la latence, nous pouvons donc commander l'articulation à la fréquence maximale $f/(3M+1)$.

Chapitre 6
Conclusion

L'étude des modèles dynamiques de l'articulation flexible nous a permis de mettre au point une stratégie de commande pour le problème de positionnement de précision de la charge. Nous avons ensuite développé un algorithme d'estimation des paramètres incertains de l'articulation nous permettant d'obtenir une qualité d'estimation suffisante au maintien des performances de la loi de commande. Enfin, nous avons tiré un maximum de profit des possibilités de mise en parallèle des calculs et des communications de données des algorithmes mis en œuvre dans notre loi de commande. Ceci nous a permit de développer des architectures hautement parallèles pour aboutir à une implantation efficace du contrôleur de l'articulation. L'architecture du contrôleur a été synthétisée en technologie CMOS – 0.5µm et comprend environ 400 000 transistors. Le calcul de la loi de commande est réalisé en 13 cycles d'horloge à une fréquence maximale de 4 MHz. L'implantation dans un processeur à architecture séquentielle de

type DSP nécessiterait 300 fois plus de cycles et serait 27 fois moins rapide même avec une fréquence d'horloge de 50 MHz.

L'originalité de ce projet a été d'apporter une contribution technologique à la robotique par le développement de processeurs spécialisés dédiés aux applications de commande de manipulateurs flexibles de hautes performances. La contribution se divise en trois domaines :

- une contribution au domaine de la commande par le développement d'une loi de commande pour un système comprenant des linéarités dures telles que la flexibilité, les frottements, les variations d'inertie, etc.
- une contribution algorithmique par le développement d'un algorithme d'estimation de paramètres variants basé sur le filtre de Kalman à racine carrée de covariance. L'algorithme présenté a une mémoire adaptative par l'utilisation particulière d'un facteur d'oubli,
- une contribution au domaine de la microélectronique par le développement d'une architecture dédiée à la commande d'une articulation flexible de manipulateur robotique.

Les développements futurs de ce projet devront permettre d'apporter les améliorations suivantes :

- estimation de la constante de flexibilité,
- estimation de la vitesse caractéristique de frottement statique,

- développement d'une méthode robuste permettant d'obtenir une bonne approximation des accélérations angulaires du moteur et de la charge,
- développement d'une méthode systématique de placement de pôles pour le moteur et la charge et tenant compte directement de la flexibilité.

Références

[AND88] R. J. Anderson, M. W. Spong, « Hybrid impedance control of robotic manipulators », IEEE Journal of Robotics and Automation, Vol. 4, pp. 549-56, 1988.

[ARM91] B. Armstrong-Helouvry, « Control of machines with friction », Kluwer Academics Publisher, Dordrecht, Netherlands, 1991.

[ARM93] B. Armstrong-Helouvry, P. Dupont, « Compensation techniques for servos with friction », Proceedings of the 1993 American Control Conference, Vol. 2, p. 1905-9, San Francisco, USA, Juin 1993.

[ARM95] B. Armstrong et C. Canudas de Wit, « Friction modeling and compensation» dans The control handbook, Chap. 77, W.S. Levine, Éditeur, IEEE Press., 1996.

[AST95] K.J. Astrom, « Adaptive control 2nd edition », Addison-Wesley edition, 1995.

[BOS92] B. K. Bose, « Recent advances in Power Electronics », IEEE Trans. Power Electronics, Vol. 7, No. 1, p. 2-16, 1992.

[CAN95] C.C. de Wit, Associate, « A new model for control of systems with friction », IEEE Transaction on Automatic Control, Vol. 40, No. 3, Mars 1995.

[CHE97] Y. Chen, R. O'Connell, « Active power line conditioner with a neural network control », IEEE Trans. Industry Applications, Vol. 33, No. 4, p. 1131, 1997.

[CRA88] J.J. Craig, « Adaptive control of mechanical manipulators », Addison-Wesley Publishing Company, 1988.

[CRA89] J.J. Craig, « Introduction to robotics second edition », Addison-Wesley Publishing Company, 1989.

[CRI86] A.J. Critchlow, « Introduction to robotics », Macmillan Publishing Company, New-York, 1986.

[DAN96] M.J. Daneman, N.C. Tien, O. Solgaard, « Linear microvibromotor for positioning optical components », Journal of Microelectromechanical Systems, Vol. 5, No. 3, pp. 159-65, Septembre 1996.

[DES90] L.A. Dessaint, B.J. Hebert, H. LeHuy, G. Cavuoti, « A DSP-based adaptive controller for a smooth positionning system », IEEE Trans. Industrial Electronics, Vol.37, No. 5, p. 372-7, Oct. 1990.

[DOT88] Y. Dote, « Application of modern control techniques to motor control », Proceedings of the IEEE, Vol. 76, No. 4, p. 438-454, Avril 1988.

[JAB91] F. Jabbari, « Lattice filters for RLS estimation of a delta operator-based model », IEEE Trans. Automatic Control, Vol. 36, No. 7, pp. 869-75, Juillet 1991.

[FEE97] M. Feemster, P. Vedagarbha, D.M, Dawson et D. Haste, « Adaptive control techniques for friction compensation », American Control Conference, 16 Oct. 1997.

[FLA94] J.M. Flaus, « La régulation industrielle – régulateurs PID, prédictifs et flous », Hermès, Paris, 1994.

[FRA94] E. N. Frantzeskakis, K. J. R. Liu, « A class of square root and division free algorithms and architectures for QRD-based adaptive signal processing », IEEE Trans. on Signal Processing, pp. 2455-2469, Septembre 1994.

[GHO89] F. Ghorbel, J. Y. Hung, M. W. Spong, «Adaptive control of flexible-joint manipulators », Proceedings of the 1989 IEEE Int. Conf. on Robotics and Automation, pp. 15-19, 1989.

[GRA96] J.O. Gray, D.G. Caldwell, « Advanced robotics & intelligent machines », Institution of Electrical Engineers, Londres, 1996.

[HAY91] S. Haykin, « Adaptative Filter Theory », Seconde édition, Prentice Hall, 1991.

[HAY95] S. Haykin, A.H. Sayed, J. Zeidler, P. Yee, P. Wei, « Tracking of linear time-varying systems », MILCOM 95 – Universal Communications Conference Record, Vol. 2, p. 602-6, San Diego, USA, Nov. 1995.

[HAY96] S. Haykin, « Adaptive filter theory – Third edition », Prentice Hall, 1996.

[ISE82] R. Isermann, « Parameter adaptive control algorithms - a tutorial », International Federation of Automatic Control, 1982.

[ISE90] R. Isermann, « Estimation of physical parameters for dynamical processes with application to an industrial robot », Proceedings of the 1990 American Control Conference, Vol. 2, p. 1396-401, San Diego, USA, Mai 1990.

[KAV98] A. Kaviani, S. Brown, « Efficient implementation of array multipliers in FPGAs », Le 5^{e} colloque Canadien sur les circuits programmables, FPD'98, p. 155-160, Montréal, 7-10 Juin 1998.

[KFU87] K. S. Fu, R. C. Gonzalez, C. S. G. Lee, « Robotics – Control, sensing, vision and intelligence », McGraw-Hill, 1987.

[KUN82] H. T. Kung, « Why systolic architectures », IEEE Computer, Vol. 15, pp. 37-46, 1982.

[KUN89] S. Y. Kung, J. N. Hwang, « Neural network architectures for robotic applications », IEEE Trans. Robotics and Automation, Vol. 5, No. 5, p. 641-657, 1989.

[KUN91] S.Y. Kung, J.N. Hwang, « Systolic array design for Kalman filtering », IEEE Trans. On Signal Processing, Vol. 39, No 1, pp. 171-82, Jan. 1991.

[MKU91] M. Kunt, et al., « Techniques modernes de traitement numérique des signaux », Presses polytechniques et universitaires romandes, Vol. 1, 1991.

[LAN93] I. D. Landau, « Identification et commande des systèmes », 2ème édition, Hermès, Paris, 1993.

[LEV96] W.S. Levine, " The control handbook", CRC Press., 1996.

[LIN86] F. Ling. D. Manolakis, J. G. Proakis, « A recursive modified Gram-Schmidt algorithm fpr least squares estimation », IEEE Trans. on Acoust., Speech, and Signal Processing, Vol. 34, No. 8, pp. 829-836, Août 1986.

[LOZ92] R. Lozano, B. Brogliato, « Adaptive control of robot manipulators with flexible joints », IEEE Trans. on Automatic Control, Vol. 37, pp. 174-81, 1992.

[MAS95] D. Massicotte, R.Z. Morawski, A. Barwicz, « Incorporation of a positivity constraint into a Kalman-filter-based algorithm for correction of spectrometric data », IEEE Trans. Instrumentation and Measurement, Vol. 44, No 1, pp. 2-7, Février 1995.

[MAS98] D. Massicotte, « A systolic VLSI implementation of Kalman-filter-based algorithms for signal reconstruction », IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vol. 8, pp. 3029-032, Seattle, USA, Mai 1998.

[MIC92] F. Michault, « Méthodes adaptatives pour le signal : outils mathématiques de mise en œuvre des algorithmes », Hermes, Paris 1992.

[MOU95] V.G. Mougdal, W.A. Kwong, K.M. Passino, S. Yurkovich, « Fuzzy learning control of a flexible-link robot », IEEE Trans. on Fuzzy Systems, Vol. 3, No. 2, p. 199-210, Mai 1995.

[MOZ98] A. L. T. Mozipo, D. Massicotte, P. Quinton et T. Risset, «Automatic synthesis of a parallel architecture for Kalman filtering using MMAlpha », 1998 Int. Conf. on Parallel Computing in Electrical Engineering (PARELEC'98), Bialystok, Pologne, pp. 201-6, 2-5 Sept. 1998.

[MOZ99] A. L. T. Mozipo, D. Massicotte, P. Quinton et T. Risset, « A parallel architecture for adaptive channel equalization based on Kalman filter using MMAlpha », 1999' Canadian Conference on Electrical and Computer Engineering, Edmonton, Canada, Mai 1999.

[NAR90] K. S. Narendra, K. Parthasarathy, « Identification and control of dynamical systems using neural networks », IEEE Trans. on Neural Networks, Vol. 1, No. 1, pp. 4-26, Mars 1990.

[QUI89] P. Quinton, Y, Robert, « Algorithmes et architectures systoliques », Masson, Paris, 1989.

[RAG96] K.J. Raghunath, K.K. Parhi, « Pipelined RLS adaptive filtering using scaled tangent rotations (STAR) », IEEE Trans. on Signal Processing, Vol. 44, No. 10, p. 2591-604, Oct. 1996.

[ROS94] M.E. Rosheim, « Robot evolution the development of anthrobotics », John Miley and Sons, Inc., 1994.

[ROV95] G.A. Rovithakis, M.A. Christodoulou, « Direct adaptive regulation of unknown nonlinear dynamical systems via dynamic neural networks », IEEE Trans. on Systems Mans and Cybernetics, Vol. 25, No. 12, p. 1578-94, Déc. 1995.

[SAL88] M.E. Salgado, G.C. Goodwin and R.H. Middleton, « Modified least squares algorithm incorporating exponential resetting and forgetting », Int. J. Control, Vol. 47, No.2, 477-491, 1988.

[SAY94] A. H. Sayed, T. Kailath, « A state space approach to adaptive RLS filtering », IEEE Trans. on Signal Processing, Vol. 11, No. 3, pp. 18-60, Juillet 1994.

[SCI96] L. Sciavicco, B. Siciliano, « Modeling and control of robot manipulators », McGraw-Hill, 1996.

[SEI92] D.R. Seidl, T.L. Reinking et R.D. Lorentz, « Use of neural networks to identify and compensate for friction in precision, position controlled mechanisms », IEEE Transactions on Industry Applications, Vol. 2, pp. 1937-1944, Octobre 1992.

[SEI95] D.R. Seidl, S.L. Lam, J.A. Putman, R.D. Lorenz, « Neural network compensation of gear backlash hysteresis in position-controlled mechanisms », IEEE Trans. on Industry Applications, Vol. 31, No. 6, pp. 1475-83, Nov. 1995.

[SIC93] P. Sicard, « Trajectory tracking of flexible joint manipulators with passivity based controller », thèse doctorale, Rensselaer Polytechnic Institute, Juin 1993.

[SID98] E. Y. O. Sidi, P. Sicard, D. Massicotte, S. Lesueur, « Adaptative high precision position control for a flexible joint with friction and parameter uncertainties using neural networks », Congrès Canadien en Génie Électrique et Informatique, Mai 1998, Waterlo ON.

[SPO89] M. W. Spong, « Adaptive control of flexible-joint manipulators », Systems & Control Letters, Vol. 13, pp. 15-21, 1989.

[SWE84] L. M. Sweet, M. C. Good, « Redefinition of the robot motion control problem : effects of plant dynamics, drive system constraints, and user

requirement », Proceedings of the 23rd IEEE Conference on Decision and Control, pp. 724-31, 1984.

[TEO91] E.K. Teoh, Y.E. Yee, « A DSP-based adaptive controller for robotic control application », IEEE Int. Conf. on Systems Mans and Cybernetics, Vol. 2, p. 961-5, Charlottesville, USA, Oct. 1991.

[TOM94] P. Tomei, « Tracking control of flexible joint robots with uncertain parameters and disturbances », IEEE Trans. On Automatic Control, Vol. 39, No. 5, p. 1067-72, Mai 1994.

[VID89] M. Vidyasagar, M. W. Spong, « Robot dynamics and control », John Wiley & Sons, New-York, 1989.

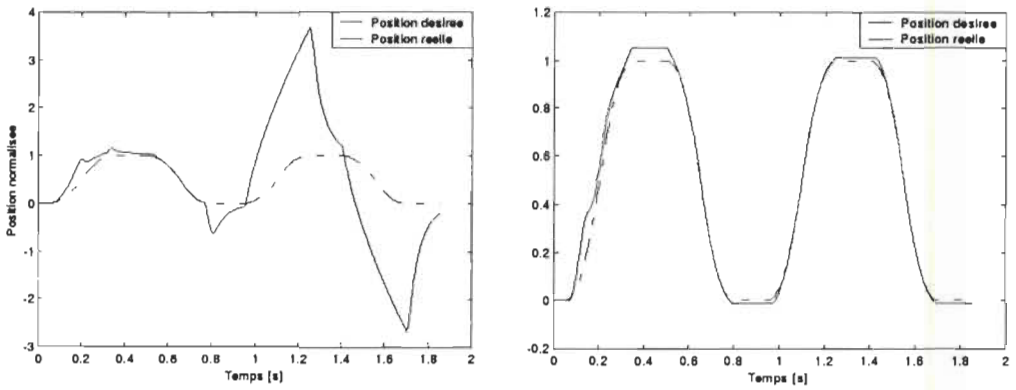
[VLS98] « High performance VLSI signal processing innovative architectures and algorithms », Vol. 2, Systems design and applications, IEEE Press., 1998.

[WHI83] J. G. McWhirter, « Recursive least-square minimization using a systolic array », Proc. SPIE, Vol. 431, pp. 415-431, 1983.

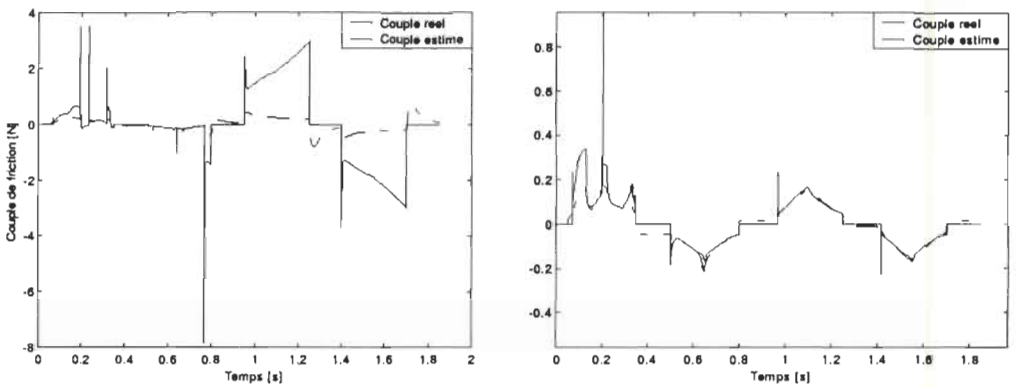
[ZHI96] Q. Zhihua, D. M. Dawson, « Robust tracking control of robots manipulators », IEEE Press., New-York, 1996.

Documents annexes

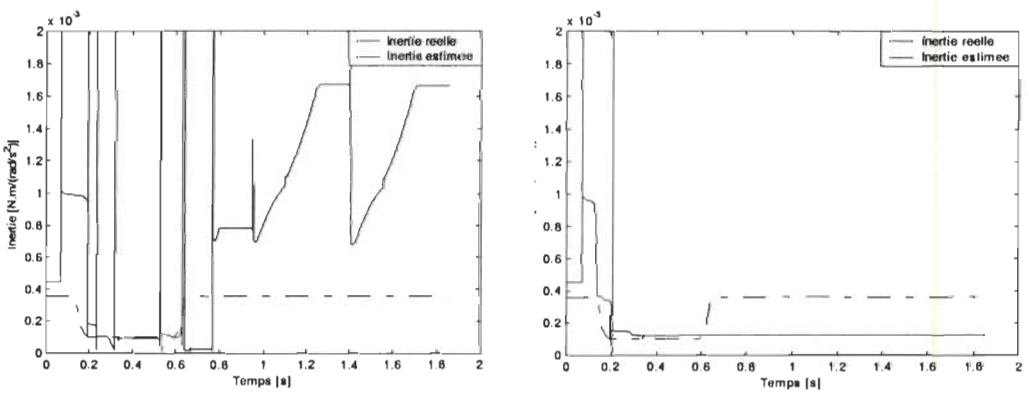
Annexe A : Résultats de simulations avec la méthode RLS sur 20 bits (à gauche) et 24 bits (à droite)



Position désirées et réelles pour une représentation à 20 et 24 bits

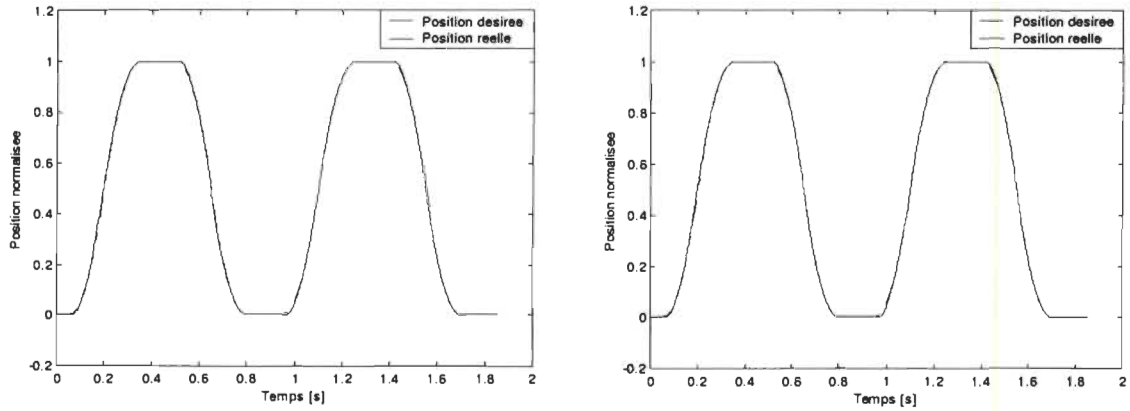


Couples de friction réels et estimés pour une représentation à 20 et 24 bits

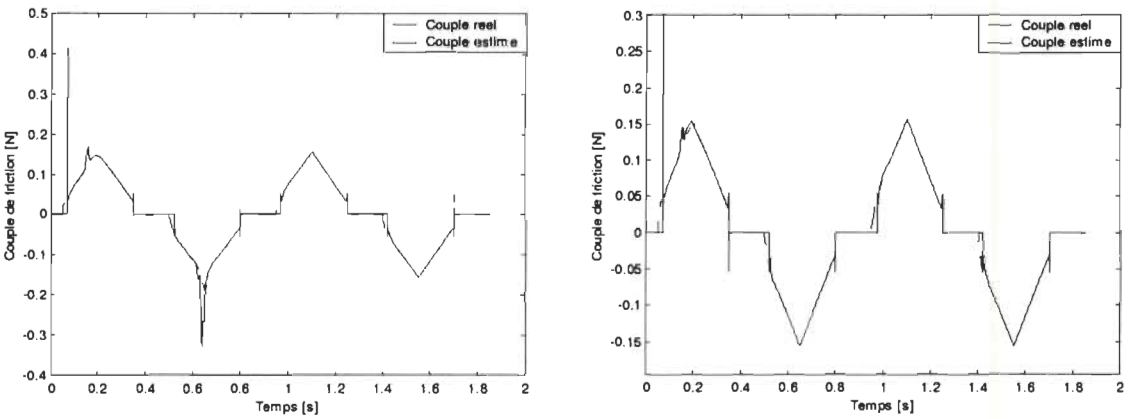


Inerties réelles et estimées de la charge pour une représentation à 20 et 24 bits

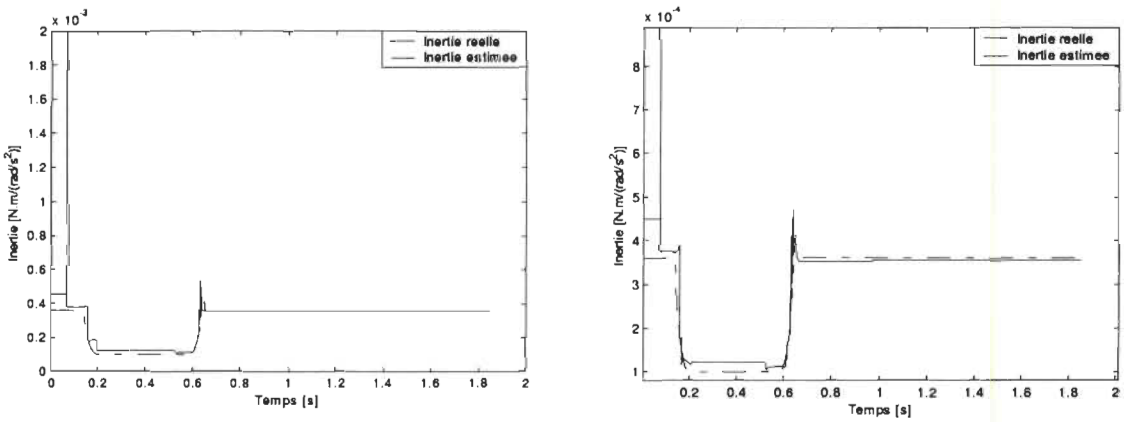
Annexe B : Résultats de simulations avec le filtre de Kalman sur 20 bits (à gauche) et 24 bits (à droite)



Position désirées et réelles pour une représentation à 20 et 24 bits

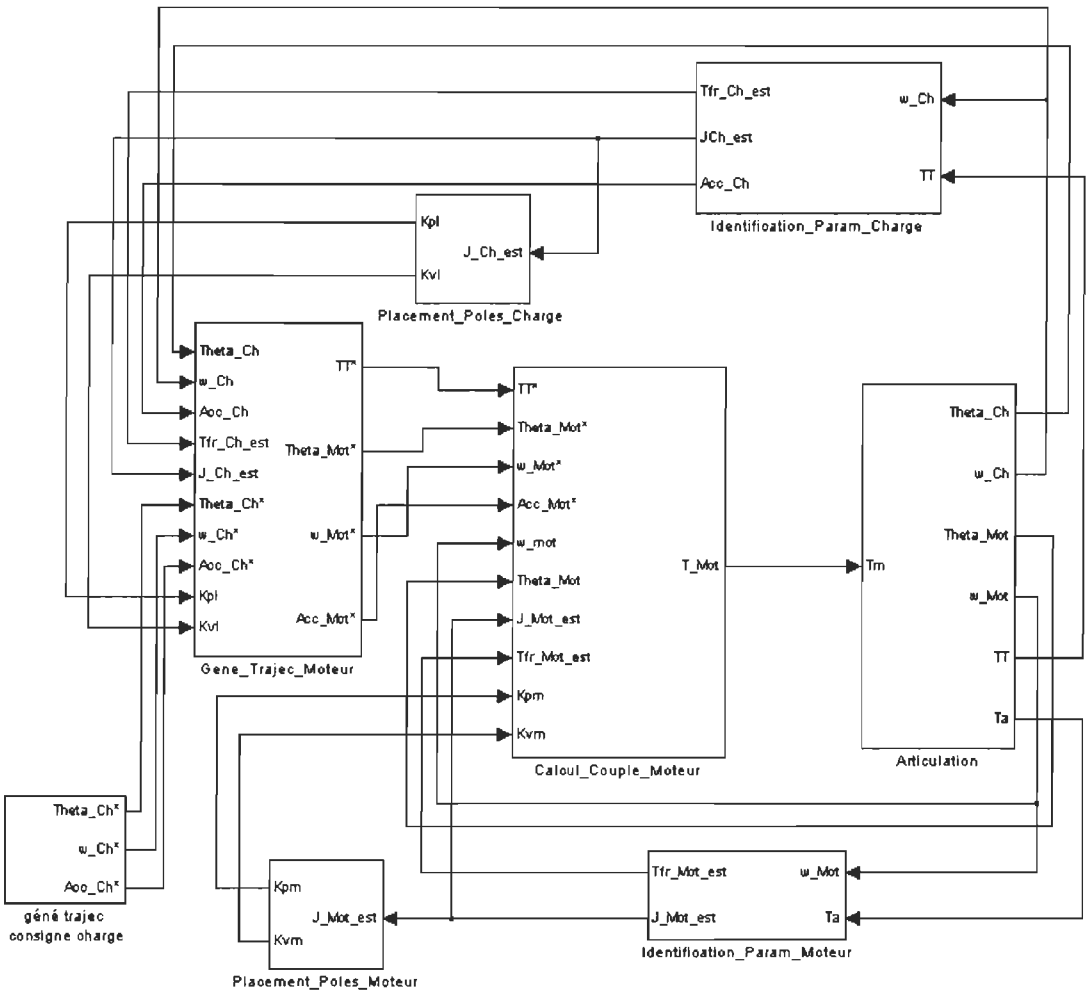


Couples de friction réels et estimés pour une représentation à 20 et 24 bits

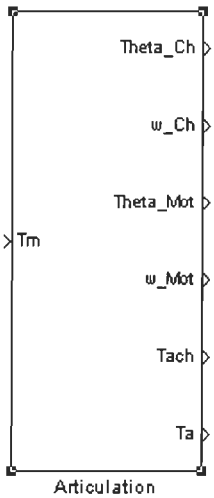


Couples de friction réels et estimés pour une représentation à 20 et 24 bits

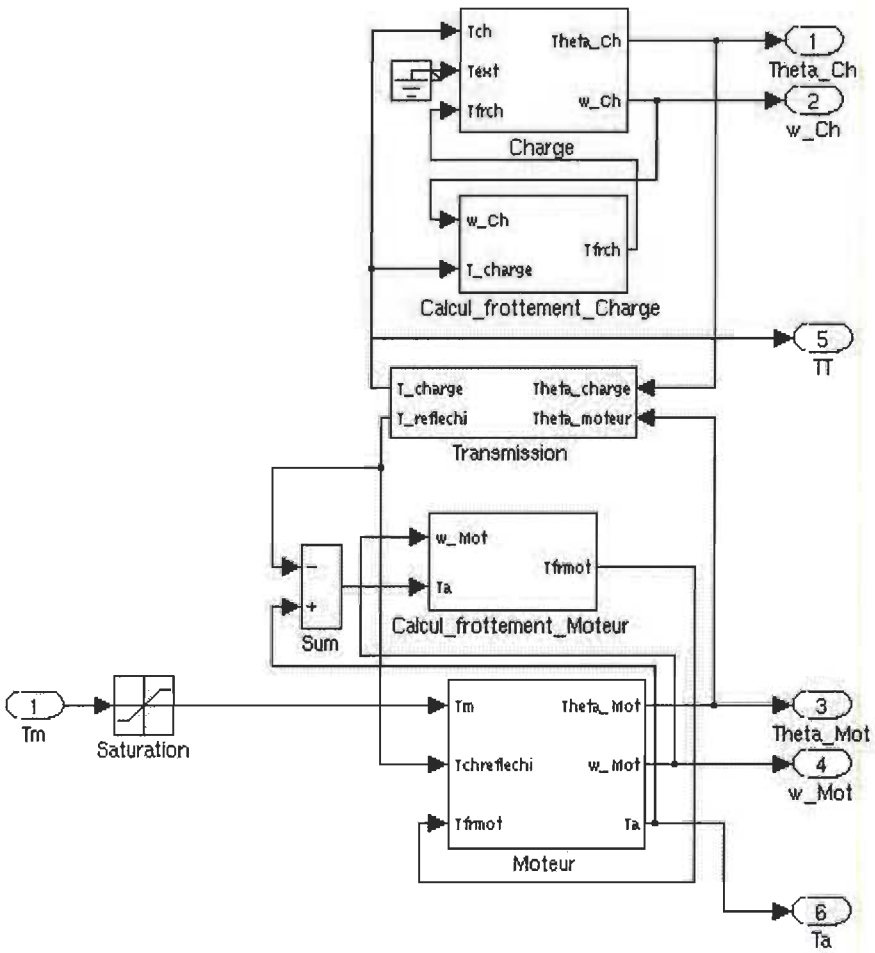
Annexe C : Schémas et programmes de simulation de l'articulation flexible dans Simulink^{TM} avec la méthode RLS



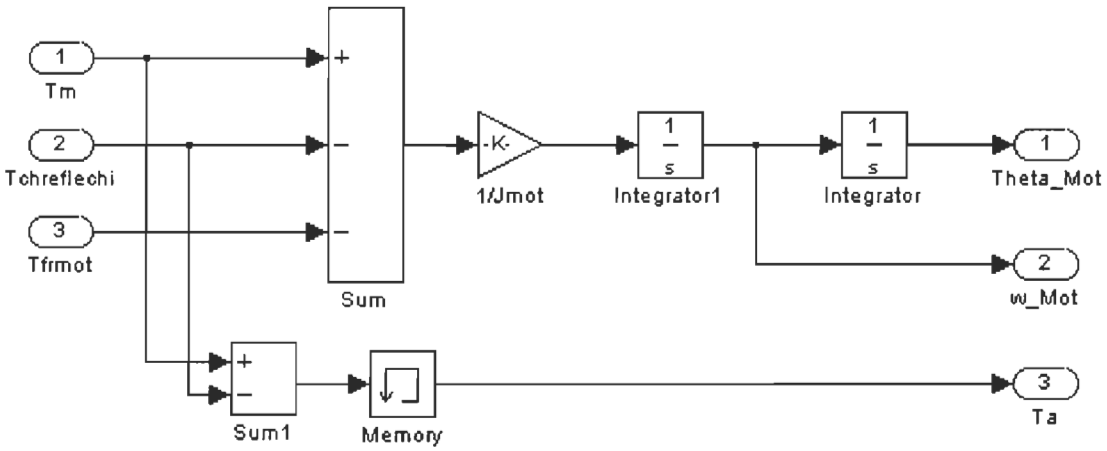
Système complet : articulation flexible avec son contrôleur



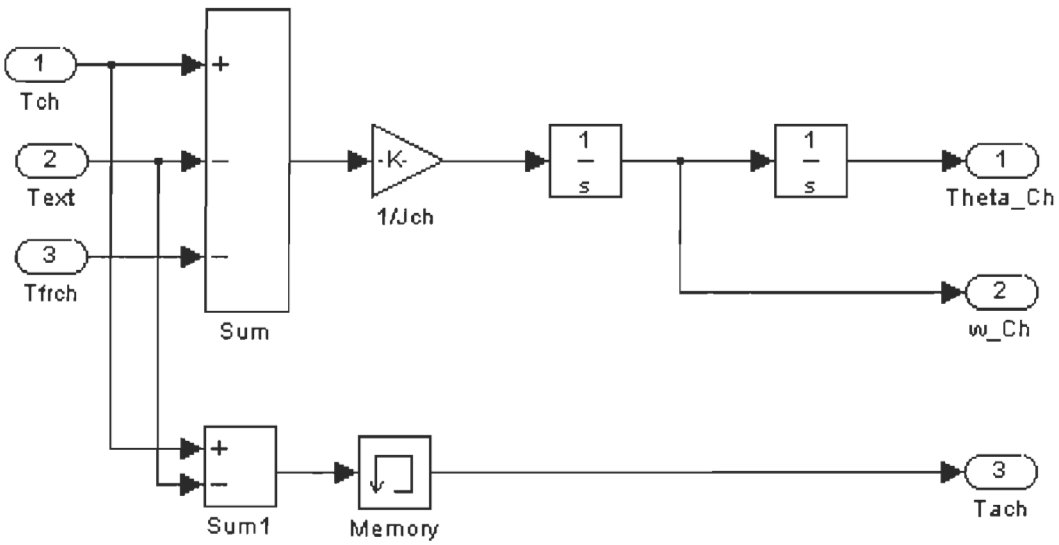
Modèle de l'articulation flexible, premier niveau



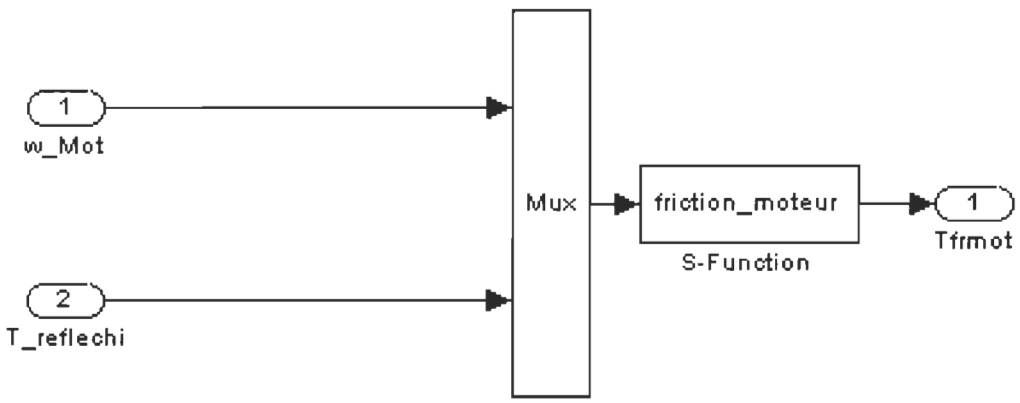
Modèle de l'articulation flexible, deuxième niveau



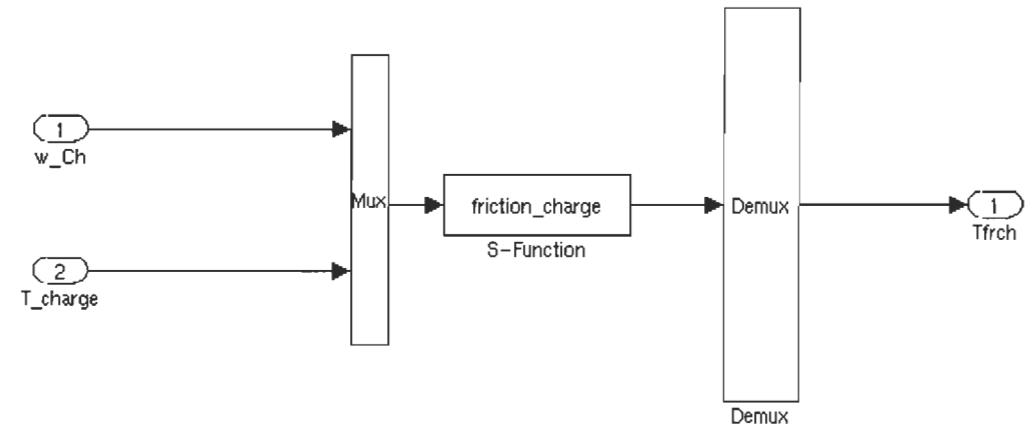
Modèle dynamique du moteur



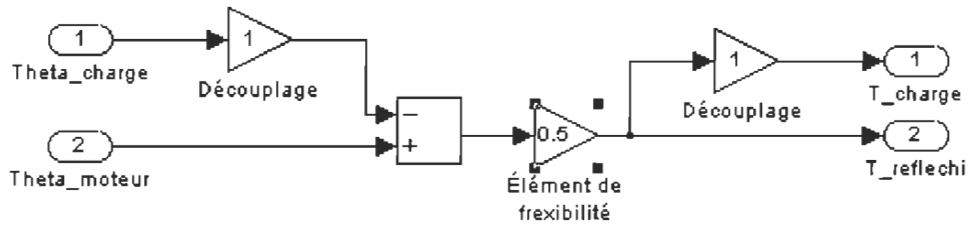
Modèle dynamique du moteur



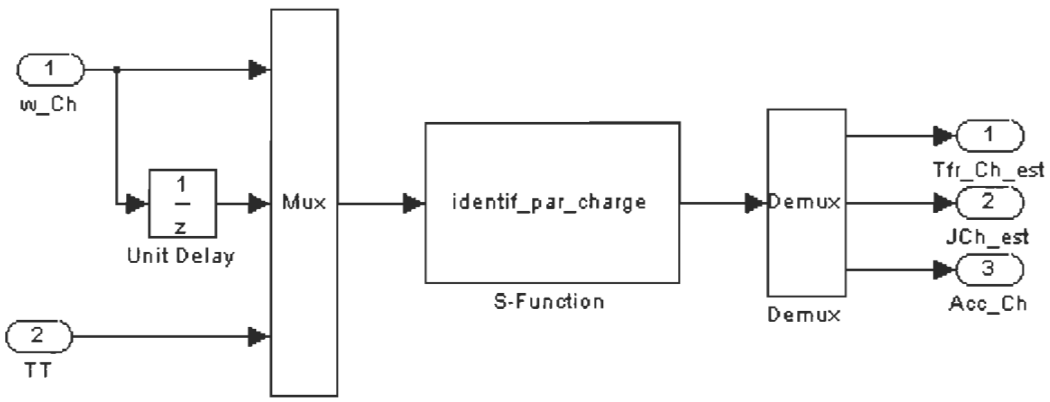
Modèle de friction du moteur



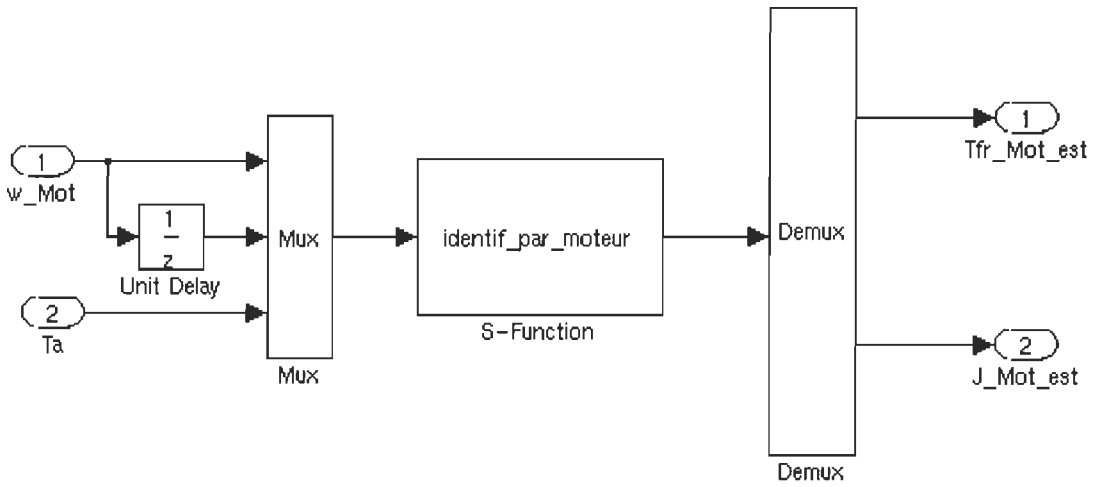
Modèle de friction de la charge



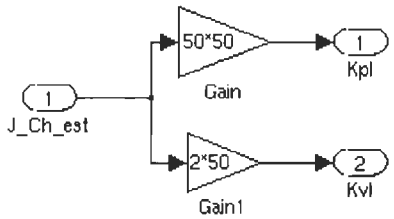
Modèle de la boîte de transmission



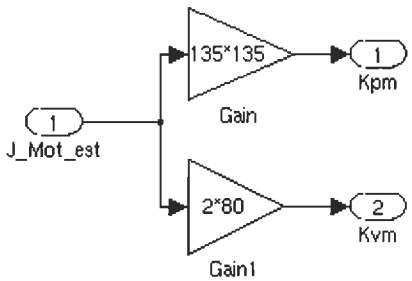
Estimateur des paramètres de la charge



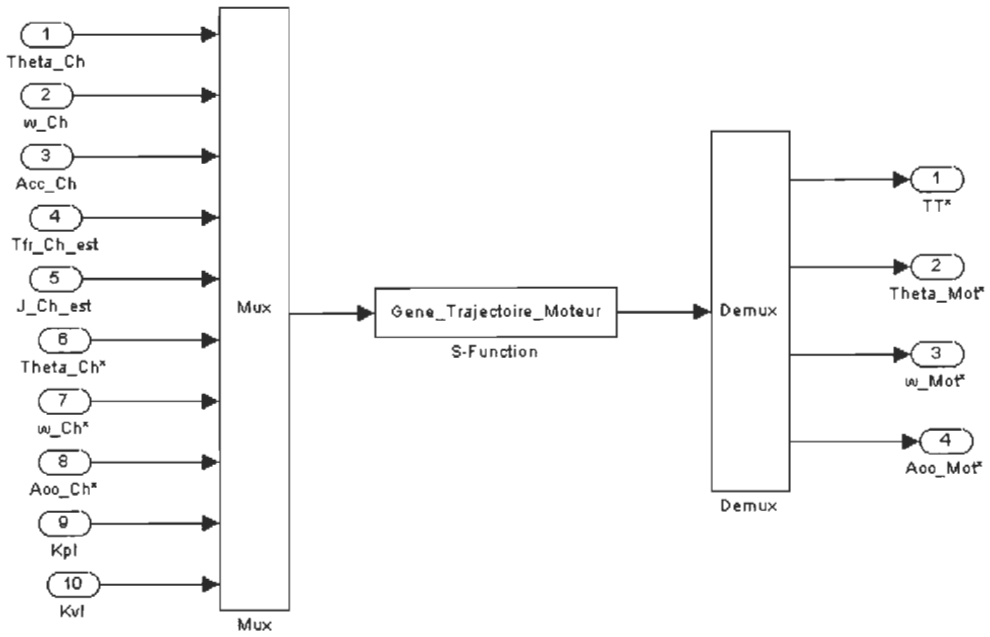
Estimateur des paramètres du moteur



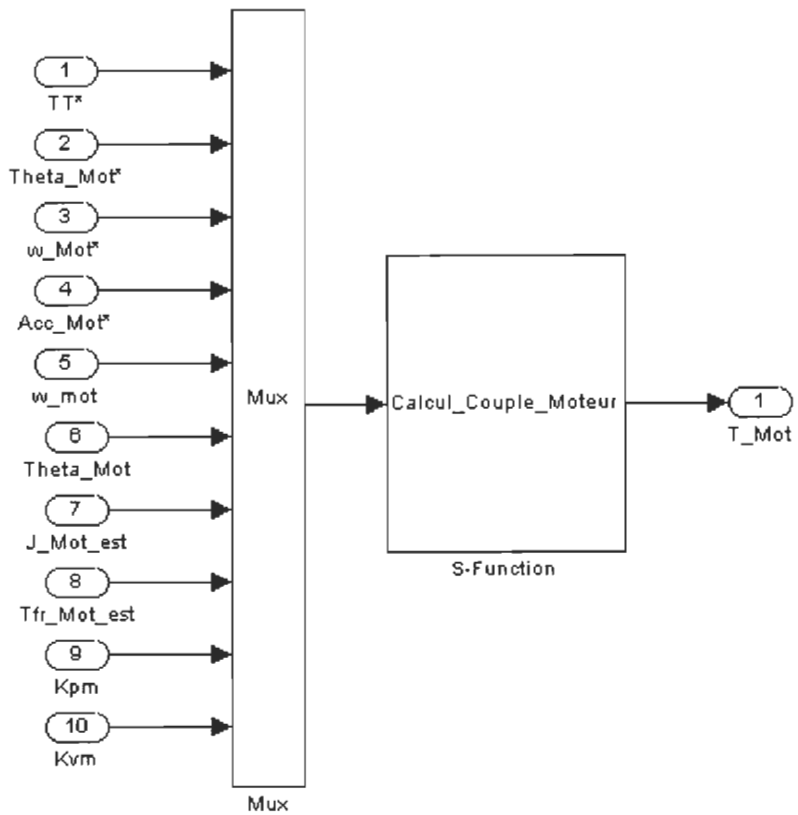
Placement de pôles de la charge



Placement de pôles du moteur



Générateur de trajectoire pour le moteur



Calcul du couple de commande du moteur

```
/*
 *    Calcul du couple de frottement moteur
 *    Modele = frottement visqueux + Coulomb + statique (Stribeck)
 *
 *    Entree : Vitesse et couple charge/moteur
 *    Sortie : Couple de friction
 *    Designation : u[0]=vitesse; u[1]=couple; y[0]=couple de friction
 */

#define S_FUNCTION_NAME friction_moteur

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(    S, 0);   /* number of continuous states */
    ssSetNumDiscStates(    S, 0);   /* number of discrete states   */
    ssSetNumInputs(        S, 2);   /* number of inputs            */
    ssSetNumOutputs(       S, 1);   /* number of outputs           */
    ssSetDirectFeedThrough(S, 1);   /* direct feedthrough flag     */
    ssSetNumSampleTimes(   S, 1);   /* number of sample times      */
    ssSetNumSFcnParams(    S, 0);   /* number of input arguments   */
    ssSetNumRWork(S, 0);            /* number of real work vector
                                    elements                       */
    ssSetNumIWork(S, 0);            /* number of integer work vector
                                    elements                       */
    ssSetNumPWork(S, 0);            /* number of pointer work vector
                                    elements                       */
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */
```

```
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

   double Fc, Fs, Fv, vs, epsilo, Tc, temp1, signev, signet;

   Fc=0.0288;
   Fv=0.0191;
   Fs=0.025;
   vs=0.095;
   epsilo=1e-4;

   if (u[0]>=0.0)
      signev=1.0;
   else
      signev=-1.0;

   if (u[1]>=0.0)
      signet=1.0;
   else
      signet=-1.0;

  temp1=u[0]/vs;
  Tc=Fc+Fv*fabs(u[0])+Fs*exp(-(temp1*temp1));
  if (fabs(u[0])>=epsilo)
      {
      y[0]=Tc*signev;
      }
  else
      {
      if (fabs(u[1])>Tc)
            y[0]=Tc*signev;
      else
            y[0]=u[1];
      }
}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
```

```
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif


/*
 *    Calcul du terme de frottement de la charge
 *    Modèle = frottement visqueux + Coulomb + statique (Stribeck)
 *
 *    Entrée : Vitesse et couple charge/moteur
 *    Sortie : Couple de friction
 *    Désignation : u[0]=vitesse; u[1]=couple; y[0]=couple de friction
 */

#define S_FUNCTION_NAME friction_charge


#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(S,0);  /* number of continuous states      */
    ssSetNumDiscStates(S,0);  /* number of discrete states        */
    ssSetNumInputs(S,2);      /* number of inputs                 */
    ssSetNumOutputs(S,1);     /* number of outputs                */
    ssSetDirectFeedThrough(S,1);   /* direct feedthrough flag     */
    ssSetNumSampleTimes(S,1); /* number of sample times           */
    ssSetNumSFcnParams(S,0);  /* number of input arguments        */
    ssSetNumRWork(S,0);       /* number of real work vector elements  */
    ssSetNumIWork(S,0);       /* number of integer work vector element*/
    ssSetNumPWork(S,0);       /* number of pointer work vector element*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
```

```
{
}
/*
 * mdlOutputs - compute the outputs
 */
static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{
   double Fc, Fs, Fv, vs, epsilo, Tc, templ, signev, signet;

   Fc=0.0288;
   Fv=0.0191;
   Fs=0.025;
   vs=0.095;
   epsilo=le-3;

   if (u[0]>=0.0)
      signev=1.0;
   else
      signev=-1.0;
   if (u[1]>=0.0)
      signet=1.0;
   else
      signet=-1.0;

  templ=u[0]/vs;
  Tc=Fc+Fv*fabs(u[0])+Fs*exp(-(templ*templ));
  if (fabs(u[0])>=epsilo)
            {
            y[0]=Tc*signev;
            }
  else
      {
            if (fabs(u[1])>Tc)
                  {
                  y[0]=Tc*signev;
                  }
            else
                  {
                  y[0]=u[1];
                  }
            }
 }

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
```

```
{
}

/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif



/*
 * Calcul de la trajectoire désirée pour le moteur
 *
 * Entrees : Position charge ThCh
 *           Vitesse charge wCh
 *           Accélération charge AccCh
 *           Couple de frottement charge estimé TfrChest
 *           Inertie charge estimée JChest
 *           Position charge désirée ThCh*
 *           Vitesse charge désirée wCh*
 *           Accélération charge désirée AccCh*
 *          Gain proportionnel sur l'erreur de position de la charge Kpl
 *           Gain proportionnel sur l'erreur de vitesse de la charge Kvl
 *
 * Sorties : Couple de charge désiré TT*
 *           Position moteur désirée ThMot*
 *           Vitesse moteur désirée wMot*
 *           Accélération moteur désirée AccMot*
 *
 * Designation : u[0]=ThCH; u[1]=wCh; u[2]=AccCh; u[3]=TfrChest;
 *           u[4]=JChest; u[5]=ThCh*; u[6]=wCh*; u[7]=AccCh* ; u[8]=kpl;
 *                      u[9]=kvl
 *           y[0]=TT*; y[1]=ThMot*; y[2]=wMot*; y[3]=Accmot*;
 */

#define S_FUNCTION_NAME Gene_Trajectoire_Moteur

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(S,0);  /* number of continuous states     */
    ssSetNumDiscStates(S,0);  /* number of discrete states       */
```

```
    ssSetNumInputs(S,10);     /* number of inputs                 */
    ssSetNumOutputs(S,4);     /* number of outputs                */
    ssSetDirectFeedThrough(S,1);   /* direct feedthrough flag     */
    ssSetNumSampleTimes(S,1); /* number of sample times           */
    ssSetNumSFcnParams(S,0);  /* number of input arguments        */
    ssSetNumRWork(S,0);       /* number of real work vector elements  */
    ssSetNumIWork(S,0);       /* number of integer work vector element*/
    ssSetNumPWork(S,0);       /* number of pointer work vector element*/
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

   double N, k;

   N=1.0;
   k=0.50;

   y[0]=(1/N)*(u[4]*u[7]+u[3]+u[9]*(u[6]-u[1])+u[8]*(u[5]-u[0]));
   y[1]=(y[0]/k)+N*u[5];
   y[2]=N*u[6]+(1/k)*(u[9]*(u[7]-u[2])+u[8]*(u[6]-u[1]));
   y[3]=N*u[7]+(u[8]/k)*(u[7]-u[2]);

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
```

```
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}


/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif



/*
 * Calcul du couple de commande du moteur
 *
 * Entrees : Couple de charge desire TT*
 *           Position moteur désirée ThMot*
 *           Vitesse moteur désirée wMot*
 *           Accélération moteur désirée AccMot*
 *           Vitesse moteur mesurée wMot
 *           Position moteur mesurée Thmot
 *           Inertie moteur estimée JMotest
 *           Couple de frottement moteur estimé TfrMotest
 *           Gain proportionnel sur l'erreur de position moteur Kpm
 *           Gain proportionnel sur l'erreur de vitesse moteur Kvm

 * Sortie : Couple de commande du moteur Tm
 *
 * Designation : u[0]=TT*; u[1]=ThMot*; u[2]=wMot*; u[3]=AccMot*;
 *           u[4]=wMot; u[5]=ThMot; u[6]=JMotest; u[7]=TfrMotest;
 *           u[8]=Kpm; u[9]=Kvm; y[0]=Tm
 */

#define S_FUNCTION_NAME Calcul_Couple_Moteur

#include "simstruc.h"
#include "math.h"

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumContStates(S,0);  /* number of continuous states      */
    ssSetNumDiscStates(S,0);  /* number of discrete states        */
    ssSetNumInputs(S,10);     /* number of inputs                 */
    ssSetNumOutputs(S,1);     /* number of outputs                */
```

```
    ssSetDirectFeedThrough(S,1);    /* direct feedthrough flag    */
    ssSetNumSampleTimes(S,1);       /* number of sample times     */
    ssSetNumSFcnParams(S,0);        /* number of input arguments  */
    ssSetNumRWork(S,0); /* number of real work vector elements    */
    ssSetNumIWork(S,0); /* number of integer work vector elements*/
    ssSetNumPWork(S,0); /* number of pointer work vector elements*/
}


/*
 * mdlInitializeSampleTimes - initialize the sample times array
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 */
static void mdlInitializeConditions(double *x0, SimStruct *S)
{
}

/*
 * mdlOutputs - compute the outputs
 */

static void mdlOutputs(double *y, double *x, double *u, SimStruct *S,
int tid)
{

   double n;

   n=1.00;

   y[0]=u[6]*u[3]+u[7]+n*u[0]+u[9]*(u[2]-u[4])+u[8]*(u[1]-u[5]);

}

/*
 * mdlUpdate - perform action at major integration time step
 */

static void mdlUpdate(double *x, double *u, SimStruct *S, int tid)
{
}

/*
 * mdlDerivatives - compute the derivatives
 */
static void mdlDerivatives(double *dx, double *x, double *u, SimStruct
*S, int tid)
{
}
```

```
/*
 * mdlTerminate - called when the simulation is terminated.
 */
static void mdlTerminate(SimStruct *S)
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif


%     IDENTIFICATION DES PARAMETRES DU MOTEUR              %
%                                                          %
%      Entrees :  Vitesse du moteur wmot                   %
%                              Couple applique au moteur Ta %
%                                                          %
%      Sorties :  Couple de frottement estime TFM          %
%                 Inertie moteur estimee JM                %
%                                                          %
%   Sebastien Lesueur, Decembre 1998                       %

function [sys,x0,str,ts] = identif_par_moteur(t,x,u,flag)

% Initialisation des constantes de simulation

lambda=1.00;
vs=0.095;
epsilo=1e-4;
T=1e-4;
Jchi=3.6e-4;
Fc=0.0288;
Fv=0.0191;
Fs=0.025;

switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
[sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);
```

```
  case {1,4,9},
    sys=[];

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl

%
%=============================================================================
======
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
function.
%=============================================================================
======
%
function
[sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)


%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.
%
sizes = simsizes;

sizes.NumContStates  = 0;
sizes.NumDiscStates  = 20;
sizes.NumOutputs     = 5;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);


%
% initialize the initial conditions
%
x0  = [0.5/Jchi
       0.5*Fc/Jchi
       0.5*Fv/Jchi
       0.5*Fs/Jchi
       1200
       0
       0
            0
       0
       1200
       0
       0
```

```
        0
        0
        1200
        0
        0
        0
        0
        1200];
%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [T 0];

% end mdlInitializeSizes

%
%=============================================================================
======
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
======
%
function sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)
accest=(u(1)-u(2))/T;
if (abs(accest)>0.01)&(abs(u(1))>0.005)
      P=[x(5:8,1)'
      x(9:12,1)'
      x(13:16,1)'
      x(17:20,1)'];

      phi=[u(3)
        -sign(u(1))
              -u(1)
        -sign(u(1))*exp(-((u(1)/vs)^2))];

      P=(1/lambda)*P-(1/lambda)*(P*phi)/(lambda+phi'*P*phi)*phi'*P;
      accest=(u(1)-u(2))/T;
      alpha=accest-phi'*x(1:4,1);
      sys=[x(1:4,1)+P*phi/(lambda+phi'*P*phi)*alpha
        P(1,1:4)'
       P(2,1:4)'
       P(3,1:4)'
       P(4,1:4)'];
 else
   sys=x;
 end

% end mdlUpdate
%
```

```
%=========================================================================
======
% mdlOutputs
% Return the block outputs.
%=========================================================================
======
%
function sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

if (abs(u(1))>epsilo)
      sys=[x(2:4,1)'/x(1,1)*[sign(u(1));u(1);sign(u(1))*…
      exp(-((u(1)/vs)^2))]
      1/x(1,1)
      max(0,x(2,1)/x(1,1))
      max(0,x(3,1)/x(1,1))
      max(0,x(4,1)/x(1,1))];


else
      Tc=x(2,1)/x(1,1)+x(3,1)/x(1,1)*abs(u(1))+x(4,1)/x(1,1)*exp(-
      (u(1)/vs)^2);

      if (abs(u(3))>Tc)
            sys=[Tc*sign(u(3))
                    1/x(1,1)
              max(0,x(2,1)/x(1,1))
              max(0,x(3,1)/x(1,1))
              max(0,x(4,1)/x(1,1))];
      else
         sys=[u(3)
              1/x(1,1)
              max(0,x(2,1)/x(1,1))
              max(0,x(3,1)/x(1,1))
              max(0,x(4,1)/x(1,1))];
      end
  end

% end mdlOutputs



%     IDENTIFICATION DES PARAMETRES DE LA CHARGE                    %
%                                                                   %
%      Entrees :  Vitesse de la charge wch                          %
%                 Couple applique a la charge TT                    %
%                                                                   %
%      Sorties :  Couple de frottement estime TFL                   %
%                              Inertie charge estimee JL            %
%                                                                   %
% Sebastien Lesueur, Decembre 1998                                  %


function [sys,x0,str,ts] = identif_par_charge(t,x,u,flag)

% Initialisation des constantes de simulation

lambda=1.00;
```

```
vs=0.095;
epsilo=1e-3;
T=1e-4;
Jchi=5.2e-4;
Fc=0.0288;
Fv=0.0191;
Fs=0.025;

switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,

[sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs);
  case {1,4,9},
    sys=[];

  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% end sfuntmpl
%
%=============================================================================
======
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-
function.
%=============================================================================
======
%
function
[sys,x0,str,ts]=mdlInitializeSizes(lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

%
% call simsizes for a sizes structure, fill it in and convert it to a
% sizes array.
%
% Note that in this example, the values are hard coded.  This is not a
% recommended practice as the characteristics of the block are typically
% defined by the S-function parameters.
%
```

```
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 20;
sizes.NumOutputs     = 6;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed

sys = simsizes(sizes);
%
% initialize the initial conditions
%
x0  = [0.5*1/Jchi
       0.5*Fc/Jchi
       0.5*Fv/Jchi
       0.5*Fs/Jchi
       1200
       0
       0
            0
       0
       1200
       0
       0
       0
       0
       1200
       0
       0
       0
       0
       1200];

%
% str is always an empty matrix
%
str = [];

%
% initialize the array of sample times
%
ts  = [T 0];

% end mdlInitializeSizes

%
%=============================================================================
======
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
======
%
function sys=mdlUpdate(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

acccalc=(u(1)-u(2))/T;
```

```
if (abs(acccalc)>0.0001)&(abs(u(1))>0.005)
            P=[x(5:8,1)'
            x(9:12,1)'
            x(13:16,1)'
            x(17:20,1)'];

            phi=[u(3)
              -sign(u(1))
                    -u(1)
                -sign(u(1))*exp(-((u(1)/vs)^2))];

            P=(1/lambda)*P-
(1/lambda)*(P*phi)/(lambda+phi'*P*phi)*phi'*P;
            acccalc=(u(1)-u(2))/T;
            alpha=acccalc-phi'*x(1:4,1);
            sys=[x(1:4,1)+P*phi/(lambda+phi'*P*phi)*alpha
              P(1,1:4)'
              P(2,1:4)'
           P(3,1:4)'
              P(4,1:4)'];
      else
            sys=x;
end

% end mdlUpdate

%
%==========================================================================
======
% mdlOutputs
% Return the block outputs.
%==========================================================================
======
%
function sys=mdlOutputs(t,x,u,lambda,vs,epsilo,T,Jchi,Fc,Fv,Fs)

if (abs(u(1))>epsilo)
      sys=[x(2:4,1)'/x(1,1)*[sign(u(1));u(1);sign(u(1))*exp(-
((u(1)/vs)^2))]
        1/x(1,1)
        (u(1)-u(2))/T
        max(0,x(2,1)/x(1,1))
        max(0,x(3,1)/x(1,1))
        max(0,x(4,1)/x(1,1))];

else
     Tc=x(2,1)/x(1,1)+x(3,1)/x(1,1)*abs(u(1))+x(4,1)/x(1,1)*exp(-
     (u(1)/vs)^2);

      if (abs(u(3))>Tc)
            sys=[Tc*sign(u(3))
              1/x(1,1)
              (u(1)-u(2))/T
              max(0,x(2,1)/x(1,1))
              max(0,x(3,1)/x(1,1))
              max(0,x(4,1)/x(1,1))];
```

```
      else
         sys=[u(3)
              1/x(1,1)
              (u(1)-u(2))/T
              max(0,x(2,1)/x(1,1))
              max(0,x(3,1)/x(1,1))
              max(0,x(4,1)/x(1,1))];
     end

end

% end mdlOutputs
```

Annexe D : Programmes de simulation des algorithmes dans Matlab®

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                           %
% Fonction d'initialisation des constantes de simulation    %
%                                                           %
% Sebastien Lesueur                                         %
% Derniere modification : 25/03/99                          %
%                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Jmnom,Kp,Kd,Fcnom,Fsnom,Fvnom,vs,t0,tf,T,epsilo,x0]=init;

Jmnom=3.6e-4;       % Inertie du moteur
Kp=2500*Jmnom;      % Gain proportionnel du correcteur
Kd=100*Jmnom;       % Gain derivatif pour facteur d'amortissement = 1
Fcnom=0.0288;       % Coefficient de frottement de Coulomb
Fsnom=0.025;        % Coefficient de frottement de statique
Fvnom=0.0191;       % Coefficient de frottement de visqueux
vs=0.095;           % Constante de vitesse du modele de frottement

t0=0;               % Temps initial
tf=1.85;            % Temps final

T=1e-4;             % Periode d'echantillonnage
epsilo=0.0001;      % Seuil de vitesse pour l'evaluation du frottement
x0=[0 0]';          % Initialisation du vecteur d'etat

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                  %
%  FONCTION D'INTEGRATION NUMERIQUE POUR           %
%  SIMULER LE COMPORTEMENT REEL DU MOTEUR          %
%                                                  %
% Entrees : instant t, x le vecteur d'etat initial %
% Sortie : Nouveau vecteur d'etat                  %
% Sebastien Lesueur                                %
% Derniere modification : 26/02/99                 %
%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xdot=moteur(t,x)

global Fcnom Fvnom Fsnom vs epsilo signalcom u Ff v

% Calcul des parametres reels de friction

Fc=Fcnom+Fcnom*2/100*sin(2*pi*2*t);
Fv=Fvnom+Fvnom*2/100*sin(2*pi*2*t);
Fs=Fsnom+Fsnom*2/100*sin(2*pi*2*t);

% Variations de l'inertie
p=(1/(1e-4)-(1/(3.6e-4)))/(0.19-0.14);
if (t<0.14) Jm=3.6e-4;
elseif (t>=0.14)&(t<0.19) Jm=1/((1/(3.6e-4))+p*(t-0.14));
elseif (t>=0.19)&(t<0.59) Jm=1e-4;
elseif (t>=0.59)&(t<0.64) Jm=1/((1/(1e-4))-p*(t-0.59));
else Jm=3.6e-4;
end
```

```
A=(Fc+Fv*abs(v)+Fs*exp(-(v/vs)^2));

% Condition pour ne pas que la friction cree de mouvement quand le
% couple de commande du moteur est faible

if abs(v)>epsilo
   Ff=A*sign(v);
elseif abs(u)>A
   Ff=A*sign(u);
else  Ff=u;
end

% integrateur 1/Jms sur la force:sortie vitesse

xdot(1,1)=u-Ff;

% action integrale sur la vitesse:sortie position
v=(1/Jm)*x(1,1);
xdot(2,1)=v;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                           %
% SIMULATION DE LA CHARGE AVEC SON CONTROLEUR               %
%                                                           %
%  Choix de consigne de trajectoire                         %
%     - type acceleration carree                            %
%                                                           %
%  Choix de l'algorithme de compensation:                   %
%     - RLS avec facteur d'oubli adaptatif                  %
%  Modele de friction : visqueux+Coulomb+statique           %
%                                       (Stribeck)          %
%                                                           %
% Sortie : Reponse en position De la charge                 %
%                                                           %
% Sebastien Lesueur                                         %
% Derniere modification : 10/03/99                          %
%                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
% CHARGEMENT DE LA TRAJECTOIRE DE CONSIGNE
% ----------------------------------------
load trajec3.mat

% DECLARATION DES VARIABLES GLOBALES
%-----------------------------------

global Jmnom Fcnom Fvnom Fsnom vs u epsilo Ff signalcom

% CONSTANTES DE SIMULATION
% ------------------------

[Jmnom Kp Kd Fcnom Fsnom Fvnom vs t0 tf T epsilo x0]=init;
[vect_borne_inf vect_borne_sup]=bornes_param(Fcnom,Fvnom,Fsnom);
```

```
temps=[];sauve1=[];sauve2=[];sauve3=[];sauve4=[];sauve5=[];sauve6=[];sau
ve7=[];sauve7=[];sauve8=[];
Ff=0;pos=0;vit=0;acc=0;vitant=0;kk=[0;0;0;0];u=0;phik=[0;0;0;0];
ekant=0;
Pint=zeros(4);M1=zeros(4);l=1;Mr=zeros(5);Tfest=0;

% INITIALISATION DES VARIABLES DE L'ALGORITHME D'IDENTIFICATION
% ----------------------------------------------------

coeffbeta=1000;      % Coefficient d'initialisation de la matrice P
betta=0.9;           % Variance du bruit de mesure
zer=[0 0 0 0]';
Pk=coeffbeta*eye(4); % Matrice de covariance initiale
l=1.0;

% Valeurs initiales des parametres estimes

xs=0.8*[1/Jmnom;Fcnom/Jmnom;Fvnom/Jmnom;Fsnom/Jmnom];xk=[0;0;0;0];
Jmest=1/xs(1);Fcest=xs(2)/xs(1);Fvest=xs(3)/xs(1);Fsest=xs(4)/xs(1);
p=(1/(1e-4)-(1/(3.6e-4)))/(0.19-0.14);

% SIMULATION DU SYSTEME COMPLET A LA FREQUENCE 1/T
% -----------------------------------------------

for k=t0/T:1:tf/T-1
   i=k+1;
   tem=k*T;

   % SIMULATION DE L'INERTIE REELLE DANS LE SYSTEME
   if (tem<0.14) Jm=3.6e-4;
   elseif (tem>=0.14)&(tem<0.19) Jm=1/((1/(3.6e-4))+p*(tem-0.14));
   elseif (tem>=0.19)&(tem<0.84) Jm=1e-4;
   elseif (tem>=0.84)&(tem<0.89) Jm=1/((1/(1e-4))-p*(tem-0.84));
   else Jm=3.6e-4;
   end

   % MESURE DES SORTIES
   % Sortie vitesse a l'instant n
   vit=(1/Jm)*x0(1,1);      % Mesure de la vitesse
   % Sortie accélération a l'instant n
   acc=(u-Ff)/Jm;           % Approximation de mesure de l'acceleration
   br=8*rand(1)-4;          % Bruit sur la mesure d'acceleration
   acc=acc+br;              % Signal d'acceleration bruite

   % Sortie position a l'instant n
   pos=x0(2,1);

   % CALCUL DES ERREURS DE POSITION ET VITESSE
   % Erreurs de position et de vitesse
   ep=posd(i)-pos;
   ev=vitd(i)-vit;



   % ESTIMATION DES PARAMETRES
   % -------------------------
```

```
   if (abs(vit)>epsilo)
         phik=[u;-sign(vit);-vit;-sign(vit)*exp(-(vit/vs)^2)];
         ek=acc-phik'*xk;
         l=choixrls(ek);

         % Mise a jour des gains sur l'erreur
         kk=Pk*phik/(l+phik'*Pk*phik);

         % Mise a jour des parametres estimes
         xk=xk+kk*ek;

         % Mise a jour de la matrice de covariance
         Pk=(1/l)*Pk-(1/l)*kk*phik'*Pk;

         % Contraintes sur les bornes
         xs=satur(xk,vect_borne_inf,vect_borne_sup);

         % Sorties de l'estimateur
         Jmest=1/xs(1,1);                              % Inertie estimee
         Tfest=[xs(2,1) xs(3,1) xs(4,1)]*Jmest*
         [sign(vit);vit;sign(vit)*exp(-(vit/vs)^2)];   % Friction estimee

   else
         Tfest=0;
   end

   % CALCUL DE LA COMMANDE
   % ---------------------

   u=Jmest*accd(i)+Kp*ep+Kd*ev+Tfest;

   % SIMULATION DU MOTEUR SUR [kT,(k+1)T] POUR LE PROCHAIN ECHANTILLON
   % ------------------------------------------------------------------

   [tt,xx]=ode45('moteur',[k*T (k+1)*T],x0);
   x0=xx(length(tt),1:2)';

   % SAUVEGARDE DES RESULTATS
   % ------------------------

   temps=[temps tt(length(tt))];
   sauve1=[sauve1 Pk];
   sauve2=[sauve2 pos];
   sauve4=[sauve4 xk];
   sauve6=[sauve6 Ff];
   sauve7=[sauve7 Tfest];
   sauve5=[sauve5 kk];
   sauve3=[sauve3 Jmest];
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  SIMULATION DE LA CHARGE AVEC SON CONTROLEUR                                 %
%                                                                              %
%  Consigne de trajectoire                                                     %
%     - type acceleration carree                                               %
%                                                                              %
%  Choix de l'algorithme de compensation:                                      %
%     - Filtre de Kalman a racine carree de covariance modifie                 %
%  Modele de friction : visqueux+Coulomb+statique (Stribeck)                   %
%                                                                              %
%  Sortie : Reponse en position de la charge                                   %
%                                                                              %
% Sebastien Lesueur                                                            %
% Derniere modification : 16/03/99                                             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
% CHARGEMENT DE LA TRAJECTOIRE DE CONSIGNE
% ----------------------------------------
load trajec3.mat

% DECLARATION DES VARIABLES GLOBALES
%-----------------------------------

global Jmnom Fcnom Fvnom Fsnom vs u epsilo Ff signalcom

% CONSTANTES DE SIMULATION
% ------------------------

[Jmnom Kp Kd Fcnom Fsnom Fvnom vs t0 tf T epsilo x0]=init;
[vect_borne_inf vect_borne_sup]=bornes_param_b(Fcnom,Fvnom,Fsnom);

temps=[];sauve1=[];sauve2=[];sauve3=[];sauve4=[];sauve5=[];
Ff=0;pos=0;vit=0;acc=0;vitant=0;kk=[0;0;0;0];u=0;phik=[0;0;0;0];
Tfest=0;

% INITIALISATION DES VARIABLES DE L'ALGORITHME D'IDENTIFICATION
% -------------------------------------------------------------
K=100;                      % Coefficient pour le changement de
                            variable J'=KJ
coeffbeta=10;               % Coefficient d'initialisation de la
                            matrice P
betta=0.9;                  % Rapport des variances de bruit
zer=[0 0 0 0]';
Pk=coeffbeta*eye(4);        % Matrice de covariance initiale
l=1.0;                      % Initialisation du facteur d'oubli

% Valeurs initiales des parametres estimes

xs=0.8*[1/(K*Jmnom);Fcnom/(K*Jmnom);Fvnom/(K*Jmnom);Fsnom/(K*Jmnom)];
xk=[0;0;0;0];
Jpest=1/xs(1);Fcest=xs(2)/xs(1);Fvest=xs(3)/xs(1);Fsest=xs(4)/xs(1);

% Variable pour le calcul de l'inertie reelle de la charge
p=(1/(1e-4)-(1/(3.6e-4)))/(0.19-0.14);
```

```
% SIMULATION DU SYSTEME COMPLET A LA FREQUENCE 1/T
% ------------------------------------------------

for k=t0/T:1:tf/T-1
   i=k+1;
   tem=k*T;                                  % Valeur du temps

   % INERTIE REELLE DANS LE SYSTEME
   if (tem<0.14) Jm=3.6e-4;
   elseif (tem>=0.14)&(tem<0.19) Jm=1/((1/(3.6e-4))+p*(tem-0.14));
   elseif (tem>=0.19)&(tem<0.59) Jm=1e-4;
   elseif (tem>=0.59)&(tem<0.64) Jm=1/((1/(1e-4))-p*(tem-0.59));
   else Jm=3.6e-4;
   end

   % MESURE DES SORTIES
   % ------------------

   vit=(1/Jm)*x0(1,1);          % Mesure de la vitesse
   acc=(u-Ff)/Jm;               % Approximation de mesure de
                                l'acceleration
   br=8*rand(1)-4;              % Mesure bruitee de l'acceleration
   acc=(acc+br);                % Signal d'acceleration bruite
   accp=acc/K;                  % Acceleration apres changement de
                                variable
   accpd=accd(i)/K;             % Acceleration desiree apres changement
                                de variable

   % Sortie du moteur:position a l'instant n
   pos=x0(2,1);

   % CALCUL DES ERREURS DE POSITION ET VITESSE

   ep=posd(i)-pos;
   ev=vitd(i)-vit;

   % ESTIMATION DES PARAMETRES
   % -------------------------

   if (abs(vit)>epsilo)&(abs(accd(i))>0.1)
      % Vecteur phi des entrees (regresseur)
      phik=[u;-sign(vit);-vit;-sign(vit)*exp(-(vit/vs)^2)];
      % Erreur d'estimation
      ek=accp-phik'*xk;
      % Choix de l
      l=choix(ek);
      M1=[(1/l)*Pk];
      % Calcul de Pk+1/k
      Pint=Givens(M1);
      M=[betta zer'
         Pint*phik Pint];
      Mr=Givens(M);
      % Gain sur l'erreur d'estimation
      kk=(Mr(1,2:5)'/Mr(1,1));

      % Mise a jour des parametres estimes
      xk=xk+kk*ek;
```

```
      % Mise a jour de la matrice de covariance
      Pk=Mr(2:5,2:5);

      % Saturation des parametres
      xs=satur(xk,vect_borne_inf,vect_borne_sup);

      % Sorties de l'estimateur
      Jpest=1/xs(1,1);                              % Inertie estimee
      Tfest=[xs(2,1) xs(3,1) xs(4,1)]*Jpest*
      [sign(vit);vit;sign(vit)*exp(-(vit/vs)^2)];  % Couple de
                                                    friction estime

      else
            Tfest=0;
      end


   % CALCUL DE LA COMMANDE
   % ---------------------

   u=Jpest*accpd+Kp*ep+Kd*ev+Tfest;

   % SIMULATION DU MOTEUR SUR [kT,(k+1)T] POUR LE PROCHAIN ECHANTILLON
   % ------------------------------------------------------------------

   [tt,xx]=ode45('moteur',[k*T (k+1)*T],x0);
   x0=xx(length(tt),1:2)';

   % SAUVEGARDE DES RESULTATS
   % ------------------------
   temps=[temps tt(length(tt))];
   sauve1=[sauve1 pos];
   sauve2=[sauve2 xk];
   sauve3=[sauve3 Ff];
   sauve4=[sauve4 Tfest];
   sauve5=[sauve5 Jpest];
end

%                                                    %
% Fonction de saturation des parametres estimes      %
% Entrees : xk parametres estimes                    %
%           vect_borne_inf bornes inferieures        %
%           vect_borne_sup bornes superieures        %
% Sebastien Lesueur                                  %
% Derniere modification : 25/03/99                   %

function xs=satur(xk,vect_borne_inf,vect_borne_sup);

for i=1:4
         if xk(i,1)<=vect_borne_inf(i,1)
            xs(i,1)=vect_borne_inf(i,1);
         elseif xk(i,1)>=vect_borne_sup(i,1)
            xs(i,1)=vect_borne_sup(i,1);
         else
               xs(i,1)=xk(i,1);
         end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
% Programme de generation de trajectoire pour la charge                        %
% Entree : tem = valeur du temps                                               %
% Sorties : posd = position articulaire desiree                                %
%           vitd = vitesse articulaire desiree                                 %
%           accd = acceleration articulaire desiree                            %
% Type de trajectoire : acceleration carree                                    %
%                       profil de position en S                                %
% Sebastien Lesueur                                                            %
% Derniere modification : 25/03/99                                             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [posd,vitd,accd]=genetrajec(tem);

if (tem<=0.05)
      posd=0;
      vitd=0;
      accd=0;
elseif (tem>0.05)&(tem<=0.20)
      posd=((0.5/0.0225)*(tem-0.05)^2);
      vitd=2*(0.5/0.0225)*(tem-0.05);
      accd=(2*(0.5/0.0225));
elseif (tem>0.20)&(tem<=0.35)
      posd=(-(0.5/0.0225)*(tem-0.35)^2+1);
      vitd=-2*(0.5/0.0225)*(tem-0.35);
      accd=-(2*(0.5/0.0225));
elseif (tem>0.35)&(tem<=0.50)
      posd=1;
      vitd=0;
      accd=0;
elseif (tem>0.50)&(tem<=0.65)
      posd=-(0.5/0.0225)*(tem-0.50)^2+1;
      vitd=-2*(0.5/0.0225)*(tem-0.50);
      accd=-2*(0.5/0.0225);
elseif (tem>0.65)&(tem<=0.80)
      posd=(0.5/0.0225)*(tem-0.80)^2;
      vitd=2*(0.5/0.0225)*(tem-0.80);
      accd=2*(0.5/0.0225);
elseif (tem>0.80)&(tem<=0.95)
      posd=0;
      vitd=0;
      accd=0;
elseif (tem>0.95)&(tem<=1.10)
      posd=((0.5/0.0225)*(tem-0.95)^2);
      vitd=2*(0.5/0.0225)*(tem-0.95);
      accd=(2*(0.5/0.0225));
elseif (tem>1.10)&(tem<=1.25)
      posd=(-(0.5/0.0225)*(tem-1.25)^2+1);
      vitd=-2*(0.5/0.0225)*(tem-1.25);
      accd=-(2*(0.5/0.0225));
elseif (tem>1.25)&(tem<=1.40)
      posd=1;
      vitd=0;
      accd=0;
```

```
elseif (tem>1.40)&(tem<=1.55)
      posd=-(0.5/0.0225)*(tem-1.40)^2+1;
      vitd=-2*(0.5/0.0225)*(tem-1.40);
      accd=-2*(0.5/0.0225);
elseif (tem>1.55)&(tem<=1.70)
      posd=(0.5/0.0225)*(tem-1.70)^2;
      vitd=2*(0.5/0.0225)*(tem-1.70);
      accd=2*(0.5/0.0225);
elseif (tem>1.70)&(tem<=1.85)
      posd=0;
      vitd=0;
      accd=0;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                       %
% Fonction de choix du facteur d'oubli en               %
% fonction de l'erreur d'estimation                     %
%                                                       %
% Entree : erreur ek                                    %
% Sortie : l facteur d'oubli                            %
% Sebastien Lesueur                                     %
% Derniere modification : 25/03/99                      %
%                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function l=choix(ek);

epse=6/100;             % Seuil pour l'erreur
if abs(ek)>epse
   l=0.99;
else
   l=l;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
% Fonction de choix du facteur d'oubli                                  %
% inverse en fonction de l'erreur d'estimation                          %
% Version avec quantification                                           %
% Entree : erreur ek                                                    %
% Sortie : 1/l inverse du facteur d'oubli                               %
% Sebastien Lesueur                                                     %
% Derniere modification : 25/03/99                                      %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function li=choixq(ek);

epse=6/100;                          % Seuil pour l'erreur

if abs(ek)>epse
   li=quant(1/0.99);
else
   li=quant(1);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                     %
% Fonction d'initialisation des bornes pour les       %
% paramètres estimées                                 %
%                                                     %
% Saturation de chacun des 4 paramètres               %
% Entrees : valeurs nominales des coefficients        %
% de friction                                         %
% Sebastien Lesueur                                   %
% Derniere modification : 25/03/99                    %
%                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function
[vect_borne_inf,vect_borne_sup]=bornes_param(Fcnom,Fvnom,Fsnom);

Jinf=2e-5;
Jsup=2e-3;
Fcinf=0.8*Fcnom;
Fcsup=1.2*Fcnom;
Fvinf=0.8*Fvnom;
Fvsup=1.2*Fvnom;
Fsinf=0.8*Fsnom;
Fssup=1.2*Fsnom;

[vect_borne_inf]=[1/Jsup;Fcinf/Jsup;Fvinf/Jsup;Fsinf/Jsup];
[vect_borne_sup]=[1/Jinf;Fcsup/Jinf;Fvsup/Jinf;Fssup/Jinf];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                        %
% Triangularisation de matrices par rotation de Givens   %
% Utilisation des fonctions gen_rot.m et appli_rot.m     %
%                                                        %
% Entree : matrice A a triangulariser                    %
% Sortie : W: matrice triangularisee                     %
% Sebastien Lesueur                                      %
% Derniere modification : 25/03/99                       %
%                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function W = Givens(A);

m = length(A(:,1));
n = length(A(1,:));
mdim = min(m,n);

%  Entrees

aout_1(1,1) = 0;
c(1,1) = 0;
s(1,1) = 0;
r(1,1) = A(1,1);
aout = [];
```

```
% Initialisation des temps
for i=1:m
   for j=i:n
      if i==1
         t_init(1,j) = j;
      else
         t_init(i,j) = t_init(i-1,j)+2;
      end
   end
end

nb_step = 2*min(m,n) + max(m,n) -1;

for j=1:n
   input(j:j+m-1,j) = A(:,j);
end

for step = 1:nb_step
   for i=1:mdim
      for j=i:mdim
         active(i,j) = (t_init(i,j)<=step & step<=t_init(i,j)+m-i);
         init(i,j) = not(step>t_init(i,j));
         if active(i,j)==1
            if i==1
               ain(i,j) =  input(step,j);
            else
               ain(i,j) = aout_1(i-1,j);
            end
            if i==j
               if step == t_init(i,j)
                  r(i,j) = ain(i,j);
                  init(i,j) = 0;
               else
                  out = gen_rot(ain(i,j),r_1(i,j));
                  r(i,j) = out(3);
                  c(i,j) = out(1);
                  s(i,j) = out(2);
               end
            else
               if step == t_init(i,j)
                  r(i,j) = ain(i,j);
                  init(i,j) = 0;
               else
                  out = appli_rot(ain(i,j),r_1(i,j),
                        c_1(i,j-1),s_1(i,j-1));
                  r(i,j) = out(3);
                  c(i,j) = out(1);
                  s(i,j) = out(2);
                  aout(i,j) = out(4);
               end
            end
         end
      end
   end
      aout_1 = aout;
   c_1 = c;
   s_1 = s;
```

```
   r_1 = r;
end
W = r;                  % Matrice de sortie triangularisee

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                           %
%                                                           %
% Programme d'application des rotations de                  %
% Givens par les cellules carrees                           %
% Entrees : ain = coefficents de la matrice                 %
%               a triangulariser                            %
%               c,s cosinus et sinus de rotation            %
%           r registre interne                              %
% Sorties : aout = coefficents de la matrice                %
%               a triangulariser                            %
%               rout = nouvelles valeurs du                 %
%               c,s cosinus et sinus de rotation            %
%           registre interne                                %
% Sebastien Lesueur                                         %
% Derniere modification : 25/03/99                          %
%                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function out=appli_rot(ain,r,c,s);

rout = c*r + s*ain;
aout = -s*r + c*ain;

out = [c,s,rout,aout];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                       %
% Programme de generation des rotations de                              %
% Givens par les cellules rondes                                        %
% Entrees : ain = coefficents de la matrice                             %
%               a triangulariser                                        %
%               r registre interne                                      %
% Sorties : c,s = cosinus et sinus pour la rotation                     %
%               a triangulariser                                        %
%               rout = nouvelles valeurs du registre                    %
%           interne                                                     %
% Sebastien Lesueur                                                     %
% Derniere modification : 25/03/99                                      %
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function out=gen_rot(ain,r);

if ain == 0
   c = 1;
   s = 0;
else
   if abs(ain)>=abs(r)
      t = r/ain;
      s = 1/sqrt(1+t*t);
```

```
      c = s*t;
   else
      t = ain/r;
      c = 1/sqrt(1+t*t);
      s = c*t;
   end
end
rout = c*r + s*ain;
out = [c,s,rout];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                  %
% Programme d'application des rotations de         %
% Givens par les cellules carrees                  %
% Avec quantification                              %
% Entrees : ain = coefficents de la matrice        %
%                a triangulariser                  %
%                c,s cosinus et sinus de rotation  %
%            r registre interne                    %
% Sorties : aout = coefficents de la matrice       %
%                a triangulariser                  %
%                rout = nouvelles valeurs du       %
%                c,s cosinus et sinus de rotation  %
%            registre interne                      %
% Sebastien Lesueur                                %
% Derniere modification : 25/03/99                 %
%                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function out = appli_rotq(ain,r,c,s);

rout = quant(quant(c*r) + quant(s*ain));
aout = quant(quant(-s*r) + quant(c*ain));
out = [c,s,rout,aout];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                        %
% Programme de generation des rotations de               %
% Givens par les cellules rondes                         %
% avec quantification                                    %
% Entrees : ain = coefficents de la matrice              %
%                a triangulariser                        %
%                r registre interne                      %
% Sorties : c,s = cosinus et sinus pour la rotation      %
%                a triangulariser                        %
%                rout = nouvelles valeurs du registre    %
%            interne                                     %
% Sebastien Lesueur                                      %
% Derniere modification : 25/03/99                       %
%                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function out=gen_rotq(ain,r);
if ain == 0
   c = 1;
   s = 0;
```

```
else
   if abs(ain)>=abs(r)
      t = quant(r/ain);
      s = quant(1/sqrt(1+t*t));
      c = quant(s*t);
   else
      t = quant(ain/r);
      c = quant(1/sqrt(1+t*t));
      s = quant(c*t);
   end
end
rout = quant(c*r + s*ain);
out = [c,s,rout];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                         %
% Triangularisation de matrices par rotation de Givens    %
% Utilisation des fonctions gen_rotq.m et appli_rotq.m    %
% Avec quantification des donnees                         %
%                                                         %
% Entree : matrice A a triangulariser                     %
% Sortie : W: matrice triangularisee                      %
% Sebastien Lesueur                                       %
% Derniere modification : 25/03/99                        %
%                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function W = Givensq(A);

m = length(A(:,1));
n = length(A(1,:));

mdim = min(m,n);

% Entrees

aout_1(1,1) = 0;

c(1,1) = 0;
s(1,1) = 0;
r(1,1) = A(1,1);
aout = [];

% Initialisation des temps pour les processeurs

for i=1:m

   for j=i:n
      if i==1
         t_init(1,j) = j;
      else
         t_init(i,j) = t_init(i-1,j)+2;
      end
   end
end

nb_step = 2*min(m,n) + max(m,n) -1;
```

```
for j=1:n
   input(j:j+m-1,j) = A(:,j);
end

for step = 1:nb_step

   for i=1:mdim
      for j=i:mdim
         active(i,j) = (t_init(i,j)<=step & step<=t_init(i,j)+m-i);

         init(i,j) = not(step>t_init(i,j));

         if active(i,j)==1
            if i==1

               ain(i,j) =  input(step,j);

            else
               ain(i,j) = aout_1(i-1,j);
            end

            if i==j
               if step == t_init(i,j)
                  r(i,j) = ain(i,j);
                  init(i,j) = 0;
               else
                  out = gen_rotq(ain(i,j),r_1(i,j));

                  r(i,j) = out(3);
                  c(i,j) = out(1);
                  s(i,j) = out(2);
               end
            else
               if step == t_init(i,j)
                  r(i,j) = ain(i,j);
                  init(i,j) = 0;
               else
                  out = appli_rotq(ain(i,j),r_1(i,j),c_1(i,j-1),s_1(i,j-
1));

                  r(i,j) = out(3);
                  c(i,j) = out(1);
                  s(i,j) = out(2);
                  aout(i,j) = out(4);

               end
            end
         end
      end
   end

   aout_1 = aout;
   c_1 = c;
   s_1 = s;
   r_1 = r;
end
W = r;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  FONCTION DE QUANTIFICATION AVEC LA            %
%  NORMALISATION 1                               %
%  ENTREE : xn valeur normalisee a quantifier    %
%                                                %
%  SORTIE : xq=valeur quantifiee de xn           %
%                                                %
%  QUANTIFICATION PAR ARRONDI, VIRGULE FIXE      %
%  SATURATION POUR LE DEPASSEMENT                %
%  1 BIT DE SIGNE                                %
%                                                %
% Sebastien Lesueur                              %
% Derniere modification : 11/08/98               %
%                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xq=quant(xn);
valmax=256;              % Valeur maximale de la dynamique

xn=xn/valmax;            % Normalisation
max=1;
Nb_bit=16;               % Nombre de bits de representation des donnees

pas=1/(2^(Nb_bit-1));    % Pas de quantification

if (xn<0)
   bit_signe=1;
else
   bit_signe=0;
end

max=1-2^(-Nb_bit+1);     % Valeur maximale de la dynamique

if (abs(xn)>=max)
   module=max/pas;
else
   module=round(abs(xn)/pas);
end

xq=(-1)^(bit_signe)*module*pas*valmax;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  FONCTION DE QUANTIFICATION AVEC LA NORMAISATION 2  %
%  ENTREE : xn valeur normalisee a quantifier         %
%                                                     %
%  SORTIE : xq=valeur quantifiee de xn                %
%  QUANTIFICATION PAR ARRONDI, VIRGULE FIXE           %
%  SATURATION POUR LE DEPASSEMENT                     %
%  1 BIT DE SIGNE                                     %
%  Sebastien Lesueur                                  %
% Derniere modification : 11/08/98                    %
%                                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xq=quant2(xn);
valmax=64;                  % Valeur maximale de la dynamique
```

```
xn=xn/valmax;            % Normalisation
max=1;
Nb_bit=16;               % Nombre de bits de representation des donnees

pas=1/(2^(Nb_bit-1));    % Pas de quantification

if (xn<0)
   bit_signe=1;
else
   bit_signe=0;
end

max=1-2^(-Nb_bit+1);     % Valeur maximale de la dynamique

if (abs(xn)>=max)
   module=max/pas;
else
   module=round(abs(xn)/pas);
end

xq=(-1)^(bit_signe)*module*pas*valmax;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                   %
% Fonction de quantification d'une matrice          %
% Entree : Min matrice a quantifier                 %
% Sortie : Mout matrice quantifiee                  %
% Sebastien Lesueur                                 %
% Derniere modification : 25/03/99                  %
%                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Mout=quantmat(Min)

for b=1:length(Min)
   for c=1:length(Min)
      Mout(b,c)=quant(Min(b,c));
   end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                   %
% Fonction de quantification d'un vecteur           %
% Entree : Vin vecteur a quantifier                 %
% Sortie : Vout vecteur quantifie                   %
% Sebastien Lesueur                                 %
% Derniere modification : 25/03/99                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function vout=quantvect(vin);

for a=1:4
   vout(a,1)=quant(vin(a,1));
end
```

Annexe E : Programmes de simulation des modèles VHDL du contrôleur

```
------------------------------------------------------------
--
-- proc_global.vhd
--       Entite et architecture pour le processeur complet
--       comprenant les deux parties (charge et moteur)
--
-- Par :
-- Sebastien Lesueur
-- 20 Avril 1999
--
------------------------------------------------------------


LIBRARY IEEE;
   USE IEEE.std_logic_1164.all;

LIBRARY lib,lib3,lib3b;
   USE lib.constants.ALL;
   USE lib.type_def.ALL;
   USE lib.components.ALL;

ENTITY proc_global IS
   PORT (clk         : IN clk_type;
         reset       : IN rst_type;
         ready       : IN STD_LOGIC;
         requestA    : OUT STD_LOGIC;
         requestB    : OUT STD_LOGIC;
         datainA     : IN data_type;
         datainB     : IN data_type;
         dataout     : OUT data_type;
         addr        : OUT addr_type
         );
END proc_global;

ARCHITECTURE struct_proc_global OF proc_global IS

SIGNAL outAtoB       : data_type;

BEGIN

   processeurA : procA PORT MAP (clk => clk,
                                 reset => reset,
                                 ready => ready,
                                 request => requestA,
                                 datain => datainA,
                                 dataout => outAtoB
                                 );

   processeurB : procB PORT MAP (clk => clk,
                                 reset => reset,
                                 ready => ready,
                                 addr => addr,
                                 request => requestB,
                                 datain => datainB,
                                 data_top => outAtoB,
                                 dataout => dataout
                                 );
```

```
END struct_proc_global;


-- synopsys synthesis_off;


CONFIGURATION conf_proc_global OF proc_global IS
   FOR struct_proc_global
      FOR processeurA : procA USE CONFIGURATION lib3.conf_procA;
      END FOR;

      FOR processeurB : procB USE CONFIGURATION lib3b.conf_procB;
      END FOR;
   END FOR;

END conf_proc_global;

-- synopsys synthesis_on;


------------------------------------------------------------------------
--
-- quickhdl.ini
--       Fichier d'initialisation
--
-- Par :
-- Sebastien Lesueur
-- 20 Avril 1999
--
------------------------------------------------------------------------

[Library]

arithmetic = /p/hpi03/mentor/pkgs/qhdl_libs/arithmetic
ieee = /p/hpi03/mentor/pkgs/qhdl_libs/ieee
MGC_SA_ANIMATION = $MGC_HOME/lib/mgc_sa_animation

mgc_portable = $MGC_HOME/pkgs/qhdl_libs/mgc_portable
std_developerskit = $MGC_HOME/pkgs/qhdl_libs/std_developerskit
synopsys = $MGC_HOME/pkgs/qhdl_libs/synopsys
verilog = $MGC_HOME/pkgs/qhdl_libs/verilog
lib = /u/hping/lesueur/mentor/proc_SRCKF/lib/work
libb = /u/hping/lesueur/mentor/procB_SRCKF/lib/work
lib2= /u/hping/lesueur/mentor/proc_SRCKF/control
lib2b= /u/hping/lesueur/mentor/procB_SRCKF/control
lib3= /u/hping/lesueur/mentor/proc_SRCKF/compil
lib3b= /u/hping/lesueur/mentor/procB_SRCKF/compil
design = /u/hping/lesueur/mentor
control_control_sdslocal                                        =
/u/hping/lesueur/mentor/proc_SRCKF/control/_work_qv
controlb_control_sdslocal                                       =
/u/hping/lesueur/mentor/procB_SRCKF/control/_work_qv

lib_control = /u/hping/lesueur/mentor/proc_SRCKF
lib_controlb = /u/hping/lesueur/mentor/procB_SRCKF

DW02 = /u/hping/lesueur/mentor/dw02/src
```

```
others = $MGC_HOME/lib/quickhdl.ini

[vsim]
main_window_pos = 34 210 739 693
signal_window_pos = 53 938 315 938
wave_window_pos = 287 18 1250 704
df_window_pos = 92 720 508 903
variable_window_pos = 552 512 720 366
list_window_pos = 177 573 577 905
source_window_pos = 527 0 1268 464
structure_window_pos = 135 579 550 860
process_window_pos = 516 573 257 185
; Dynamically linked object containing user's
; Verilog PLI tasks and functions.
; Veriuser = veriuser.o

; Default run length
RunLength = 100

; Iterations that can be run without advancing simulation time
IterationLimit = 5000

; Simulator resolution
Resolution = ns

; Stop the simulator after an assertion message
; 0 = Note  1 = Warning  2 = Error  3 = Failure  4 = Fatal
BreakOnAssertion = 3

; List Translation
; Map an enumerated value to 0, 1, or Z.  Default is 'X';
List0 = '0' FALSE 'L' s0r s0s
List1 = '1' TRUE 'H' s1r s1s
ListZ = 'Z' sxz s0z s1z
listX = sxr sxz sxs sxi s0i s1i

; Force Translation
; Map 1s and 0s to the enumerated value
Force0 = '0' FALSE s0s
Force1 = '1' TRUE s1s

; Default radix for all windows and commands...
; 0 = symbolic, 2 = binary, 8 = octal, 10 = decimal, 16 = hex
DefaultRadix = 0

; This controls the number of characters of a signal name
; shown in the waveform window. The default value or a
; value of zero tells VSIM to display the full name.
; WaveSignalNameWidth = 10

; QHSIM Startup command
; Startup = do startup.do

; Save the command window contents to this file
; TranscriptFile = transcript
```

```
; Disable assertion messages
; ignoreNote = 1
; ignoreWarning = 1
; ignoreError = 1
; ignoreFailure = 1

; If zero, open files when elaborated
; else open files on first read or write
; DelayFileOpen = 0

; window positions
; Position values correspond as given below
; window_name_pos = y_position x_position width height
schematic_window_pos = 29 555 617 290

; List of object libraries to search for C interface
ForeignLibs = /usr/lib/libc.a

; Delay opening of files until first read or write. (enable=1)
DelayFileOpen = 0

[lmc]
; QuickHDL's interface to SWIFT software
libsm = $MGC_HOME/pkgs/quickhdl/.lib/libsm.sl
; Logic Modeling's SWIFT software (HP 9000 Series 700)
libswift = $LMC_HOME/lib/hp700.lib/libswift.sl

; Note: See Synopsis documentation on how to install
;       the hardware modeler's SFI software. Modify
;       the below given libsfi definition to incorporate
;       the hardware modeler's installation path.
;
;libsfi = <install_dir>/sms/lib/pa_hp90/libsfi.sl
;
; QHSIM's hardware modeler interface to SFI software
libhm = $MGC_HOME/pkgs/quickhdl/.lib/libhm.sl


------------------------------------------------------------------------
--
-- proc.vhd
--        Entite et architecture pour le processeur A
--
--
-- Par :
-- Sebastien Lesueur
-- 20 Avril 1999
--
------------------------------------------------------------------------


LIBRARY IEEE;
   USE IEEE.std_logic_1164.all;


LIBRARY lib, lib2, lib3;
   use lib.constants.all;
```

```
   use lib.type_def.all;
   use lib.components.all;

ENTITY procA IS
   PORT (clk          : IN clk_type;
         reset        : IN rst_type;
         ready        : IN STD_LOGIC;
         request      : OUT STD_LOGIC;
         datain       : IN data_type;
         dataout      : OUT data_type
         );
END procA;


ARCHITECTURE struct_procA OF procA IS
   SIGNAL   datainPE11, datainPE12,
            datainPE13, datainPE14,
            datainPE15, datain1PE16,
            datain2Pe16,dataout1PE15,
            dataout2PE15, dataoutPE16,
            dataoutPE25,dataoutPE35,
            dataoutPE45, dataoutPE55          : data_type;
   SIGNAL   stat                              : state_type;
   SIGNAL   ck                                : clk_type;
   SIGNAL   rst                               : rst_type;
   SIGNAL   initPE11, initPE22, initPE33,
            initPE44, initPE55                : init_type;
   SIGNAL   modePE11
                                              : mode_type1;
   SIGNAL   modePE12, modePE13, modePE14,
            modePE15                          : mode_type2;
   SIGNAL   modePE16, modePE22, modePE23,
            modePE24, modePE25, modePE33,
            modePE34,modePE35, modePE44,
            modePE45, modePE55                : mode_type1;

BEGIN


   reseau_proc : reseau PORT MAP(clkreseau => ck,
                                 rstreseau => rst,
                                 initPE11 => initPE11,
                                 initPE22 => initPE22,
                                 initPE33 => initPE33,
                                 initPE44 => initPE44,
                                 initPE55 => initPE55,
                                 modePE11 => modePE11,
                                 modePE12 => modePE12,
                                 modePE13 => modePE13,
                                 modePE14 => modePE14,
                                 modePE15 => modePE15,
                                 modePE16 => modePE16,
                                 modePE22 => modePE22,
                                 modePE23 => modePE23,
                            modePE24 => modePE24,
                            modePE25 => modePE25,
                            modePE33 => modePE33,
```

```
                        modePE34 => modePE34,
                        modePE35 => modePE35,
                        modePE44 => modePE44,
                        modePE45 => modePE45,
                        modePE55 => modePE55,
                        datainPE11 => datainPE11,
                        datainPE12 => datainPE12,
                        datainPE13 => datainPE13,
                        datainPE14 => datainPE14,
                        datainPE15 => datainPE15,
                        datain1PE16 => datain1PE16,
                        datain2PE16 => datain2PE16,
                        dataout1PE15 => dataout1PE15,
                        dataout2PE15 => dataout2PE15,
                        dataoutPE16 => dataout,
                        dataoutPE25 => dataoutPE25,
                        dataoutPE35 => dataoutPE35,
                        dataoutPE45 => dataoutPE45,
                        dataoutPE55 => dataoutPE55
                        );

mux : muxbloc PORT MAP( rst => reset,
                        state => stat,
                        datain => datain,
                        from1PE15 => dataout1PE15,
                        from2PE15 => dataout2PE15,
                        fromPE16 => dataoutPE16,
                        fromPE25 => dataoutPE25,
                        fromPE35 => dataoutPE35,
                        fromPE45 => dataoutPE45,
                        fromPE55 => dataoutPE55,
                        toPE11 => datainPE11,
                        toPE12 => datainPE12,
                        toPE13 => datainPE13,
                        toPE14 => datainPE14,
                        toPE15 => datainPE15,
                        to1PE16 => datain1PE16,
                        to2PE16 => datain2PE16
                        );

csm : control PORT MAP( clk => clk,
                        ready => ready,
                        reset => reset,
                        clkreseau => ck,
                        initPE11 => initPE11,
                        initPE22 => initPE22,
                        initPE33 => initPE33,
                        initPE44 => initPE44,
                        initPE55 => initPE55,
                        modePE11 => modePE11,
                        modePE12 => modePE12,
                        modePE13 => modePE13,
                        modePE14 => modePE14,
                        modePE15 => modePE15,
                        modePE16 => modePE16,
                        modePE22 => modePE22,
                        modePE23 => modePE23,
```

```
                              modePE24 => modePE24,
                              modePE25 => modePE25,
                              modePE33 => modePE33,
                              modePE34 => modePE34,
                              modePE35 => modePE35,
                              modePE44 => modePE44,
                              modePE45 => modePE45,
                              modePE55 => modePE55,
                              request => request,
                              rstreseau => rst,
                              state_mux => stat
                              );
END struct_procA;


-- synopsys synthesis_off;


CONFIGURATION conf_procA OF procA IS
   FOR struct_procA
      FOR reseau_proc : reseau USE CONFIGURATION lib3.conf_reseau;
      END FOR;

      FOR mux : muxbloc USE ENTITY lib3.muxbloc(behav_muxbloc);
      END FOR;

      FOR csm : control USE ENTITY lib2.control(data_flow);
      END FOR;
   END FOR;

END conf_procA;

-- synopsys synthesis_on;



------------------------------------------------------------------------
--
--    Reseau.vhd
--    Description de l'entite et de l'architecture du reseau systolique
--    triangulaire ReseauA
--
--    Par : Sebastien Lesueur
--    12 Avril 1999
--
------------------------------------------------------------------------


LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE lib.type_def.ALL;
   USE lib.components.ALL;


ENTITY reseau IS
   PORT( clkreseau                        : IN clk_type;
      rstreseau                           : IN rst_type;
```

```
      initPE11, initPE22, initPE33,
      initPE44, initPE55                  : IN init_type;
      modePE11                            : IN mode_type1;
      modePE12, modePE13,modePE14,
      modePE15                            : IN mode_type2;
      modePE16,modePE22,modePE23, modePE24,
      modePE25,modePE33, modePE34,
      modePE35,modePE44, modePE45,
      modePE55                            : IN mode_type1;
      datainPE11, datainPE12,datainPE13,
      datainPE14, datainPE15, datain1PE16,
      datain2PE16                         : IN data_type;
      dataout1PE15, dataout2PE15,
      dataoutPE16, dataoutPE25, dataoutPE35,
      dataoutPE45, dataoutPE55            : OUT data_type
       );
END reseau;


ARCHITECTURE struct_reseau OF reseau IS

SIGNAL s_sig1, c_sig1, data_sig1                : data_type4;
SIGNAL s_sig2, c_sig2, data_sig2                : data_type3;
SIGNAL s_sig3, c_sig3, data_sig3                : data_type2;
SIGNAL s_sig4, c_sig4, data_sig4                : data_type;

SIGNAL init_sig1                                : init_type4;
SIGNAL init_sig2                                : init_type3;
SIGNAL init_sig3                                : init_type2;
SIGNAL init_sig4                                : init_type;


BEGIN

   PE11 : rona PORT MAP(clk => clkreseau, rst => rstreseau, mode =>
   modePE11, initin => initPE11, datain => datainPE11,
   cout => c_sig1(1), sout => s_sig1(1), initout => init_sig1(1));

   PE12 : cara PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE12, initin => init_sig1(1), datain => datainPE12,
   cin => c_sig1(1), sin => s_sig1(1), sout => s_sig1(2),
   cout => c_sig1(2), initout => init_sig1(2),
   dataout_dn => data_sig1(1));

   PE13 : cara PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE13, initin => init_sig1(2), datain => datainPE13,
   cin =>  c_sig1(2), sin => s_sig1(2), sout => s_sig1(3),
   cout => c_sig1(3), initout=> init_sig1(3),
   dataout_dn => data_sig1(2));

   PE14 : cara PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE14, initin => init_sig1(3), datain => datainPE14, cin
   => c_sig1(3), sin => s_sig1(3),sout => s_sig1(4), cout => c_sig1(4),
   initout => init_sig1(4), dataout_dn => data_sig1(3));
   PE15 : carc PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE15, initin => init_sig1(4), datain => datainpe15,
   cin => c_sig1(4), sin => s_sig1(4), sout => dataout2PE15,
```

```
   cout => dataout1PE15, initout => OPEN, dataout_dn => data_sig1(4));

   PE16 : card PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE16, datain1 => datain1PE16, datain2 => datain2PE16,
   dataout_dn => dataoutPE16);

   PE22 : ronb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE22, initin => initPE22, datain => data_sig1(1),
   cout => c_sig2(1), sout => s_sig2(1), initout => init_sig2(1));

   PE23 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE23, initin => init_sig2(1), datain => data_sig1(2),
   cin => c_sig2(1), sin => s_sig2(1), cout => c_sig2(2),
   sout => s_sig2(2), initout => init_sig2(2),
   dataout_dn => data_sig2(1));

   PE24 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE24, initin => init_sig2(2), datain => data_sig1(3),
   cin => c_sig2(2), sin => s_sig2(2), cout => c_sig2(3),
   sout => s_sig2(3), initout => init_sig2(3),
   dataout_dn => data_sig2(2));

   PE25 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE25, initin => init_sig2(3), datain => data_sig1(4),
   cin => c_sig2(3), sin => s_sig2(3), cout => dataoutPE25,
   sout => OPEN, initout => OPEN, dataout_dn => data_sig2(3));

   PE33 : ronb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE33, initin => initPE33, datain => data_sig2(1),
   cout => c_sig3(1), sout => s_sig3(1), initout => init_sig3(1));

   PE34 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE34, initin => init_sig3(1), datain => data_sig2(2),
   cin => c_sig3(1), sin => s_sig3(1), cout => c_sig3(2),
   sout => s_sig3(2), initout => init_sig3(2),
   dataout_dn => data_sig3(1));

   PE35 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE35, initin => init_sig3(2), datain => data_sig2(3),
   cin => c_sig3(2), sin => s_sig3(2), cout => dataoutPE35,
   sout => OPEN, initout => OPEN, dataout_dn => data_sig3(2));

   PE44 : ronb PORT MAP(clk => clkreseau, rst => rstreseau, mode =>
   modePE44, initin => initPE44, datain => data_sig3(1),cout => c_sig4,
   sout => s_sig4, initout => init_sig4);

   PE45 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE45, initin => init_sig4, datain => data_sig3(2),
   cin => c_sig4, sin => s_sig4, cout => dataoutPE45, sout => OPEN,
   initout => OPEN, dataout_dn => data_sig4);

   PE55 : ronc PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE55, initin => initPE55, datain => data_sig4,
   cout => dataoutPE55, sout => OPEN, initout => OPEN);
END struct_reseau;
```

```
CONFIGURATION conf_reseau OF reseau IS
   FOR struct_reseau

      FOR PE11 : rona USE ENTITY lib.rona(behav_rona);
      END FOR;

      FOR PE12 : cara USE ENTITY lib.cara(behav_cara);
      END FOR;

      FOR PE13 : cara USE ENTITY lib.cara(behav_cara);
      END FOR;

      FOR PE14 : cara USE ENTITY lib.cara(behav_cara);
      END FOR;

      FOR PE15 : carc USE ENTITY lib.carc(behav_carc);
      END FOR;

      FOR PE16 : card USE ENTITY lib.card(behav_card);
      END FOR;

      FOR PE22 : ronb USE ENTITY lib.ronb(behav_ronb);
      END FOR;

      FOR PE23 : carb USE ENTITY lib.carb(behav_carb);
      END FOR;

      FOR PE24 : carb USE ENTITY lib.carb(behav_carb);
      END FOR;

      FOR PE25 : carb USE ENTITY lib.carb(behav_carb);
      END FOR;

      FOR PE33 : ronb USE ENTITY lib.ronb(behav_ronb);
      END FOR;

      FOR PE34 : carb USE ENTITY lib.carb(behav_carb);
      END FOR;

      FOR PE44 : ronb USE ENTITY lib.ronb(behav_ronb);
      END FOR;

      FOR PE45 : carb USE ENTITY lib.carb(behav_carb);
      END FOR;

      FOR PE55 : ronc USE ENTITY lib.ronc(behav_ronc);
      END FOR;

   END FOR;
END conf_reseau;



------------------------------------------------------------
--
```

```
--   muxbloc.vhd
--        Bloc de multiplexage des donnees du controleur A
--
-- Par : Sebastien Lesueur
--       4 Mai 1999
--
-------------------------------------------------------------------


LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;

   USE lib.constants.ALL;
   USE lib.type_def.ALL;

ENTITY muxbloc IS
            PORT( rst               : IN rst_type;
                  state             : IN state_type;
                  datain            : IN data_type;
                  from1PE15         : IN data_type;
                  from2PE15         : IN data_type;
                  fromPE16          : IN data_type;
                  fromPE25          : IN data_type;
                  fromPE35          : IN data_type;
                  fromPE45          : IN data_type;
                  fromPE55          : IN data_type;
                  toPE11            : OUT data_type;
                  toPE12            : OUT data_type;
                  toPE13            : OUT data_type;
                  toPE14            : OUT data_type;
                  toPE15            : OUT data_type;
                  to1PE16           : OUT data_type;
                  to2PE16           : OUT data_type
                  );
END muxbloc ;


ARCHITECTURE behav_muxbloc OF muxbloc IS

BEGIN

   main : PROCESS (rst, state, datain, from1PE15, from2PE15, fromPE16,
   fromPE25, fromPE35, fromPE45, fromPE55)

   BEGIN

      CASE state IS
         WHEN state00 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= lambd2;
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');
            to1PE16 <= (OTHERS => '0');
            to2PE16 <= (OTHERS => '0');
         WHEN state01 =>
            toPE11 <= (OTHERS => 'W');
```

```
        toPE12 <= datain;
        toPE13 <= lambd2;
        toPE14 <= (OTHERS => 'W');
        toPE15 <= (OTHERS => 'W');

    WHEN state02 =>
        toPE11 <= (OTHERS => 'W');
        toPE12 <= (OTHERS => 'W');
        toPE13 <= datain;
        toPE14 <= lambd2;
        toPE15 <= (OTHERS => 'W');

    WHEN state03 =>
        toPE11 <= (OTHERS => 'W');
        toPE12 <= (OTHERS => 'W');
        toPE13 <= (OTHERS => 'W');
        toPE15 <= lambd2;

    WHEN state04 =>
        toPE11 <= (OTHERS => 'W');
        toPE12 <= (OTHERS => 'W');
        toPE13 <= (OTHERS => 'W');
        toPE14 <= (OTHERS => 'W');

    WHEN state05 =>
        toPE11 <= beta;
        toPE12 <= (OTHERS => 'W');
        toPE13 <= (OTHERS => 'W');
        toPE14 <= (OTHERS => 'W');
        toPE15 <= (OTHERS => 'W');
        to1PE16 <= (OTHERS => '0');
        to2PE16 <= (OTHERS => '0');

    WHEN state1 =>
        toPE11 <= fromPE25;
        toPE12 <= fromPE55;
        toPE13 <= (OTHERS => 'W');
        toPE14 <= (OTHERS => 'W');
        toPE15 <= (OTHERS => 'W');
        to1PE16 <= datain;
        to2PE16 <= from1PE15;

    WHEN state2 =>
        toPE11 <= fromPE35;
        toPE12 <= fromPE25;
        toPE13 <= fromPE55;
        toPE14 <= (OTHERS => 'W');
        toPE15 <= datain;
        to1PE16 <= datain;
        to2PE16 <= from1PE15;

    WHEN state3 =>
        toPE11 <= fromPE45;
        toPE12 <= fromPE35;
        toPE13 <= fromPE25;
        toPE14 <= fromPE55;
        toPE15 <= (OTHERS => 'W');
```

```
      to1PE16 <= datain;
      to2PE16 <= from1PE15;

   WHEN state4 =>
      toPE11 <= fromPE55;
      toPE12 <= fromPE45;
      toPE13 <= fromPE35;
      toPE14 <= fromPE25;
      toPE15 <= fromPE45;
      to2PE16 <= from1PE15;

   WHEN state5 =>
      toPE11 <= (OTHERS => 'W');
      toPE12 <= fromPE55;
      toPE13 <= fromPE45;
      toPE14 <= fromPE35;
      toPE15 <= fromPE25;
      to2PE16 <= from1PE15;

   WHEN state6 =>
      toPE11 <= (OTHERS => 'W');
      toPE12 <= (OTHERS => 'W');
      toPE13 <= fromPE55;
      toPE14 <= fromPE45;
      toPE15 <= fromPE35;

   WHEN state7 =>
      toPE11 <= (OTHERS => 'W');
      toPE12 <= from2PE15;
      toPE13 <= (OTHERS => 'W');
      toPE14 <= fromPE55;
      toPE15 <= fromPE45;

   WHEN state8 =>
      toPE11 <= (OTHERS => 'W');
      toPE12 <= from1PE15;
      toPE13 <= from2PE15;
      toPE14 <= (OTHERS => 'W');
      toPE15 <= fromPE55;

   WHEN state9 =>
      toPE11 <= (OTHERS => 'W');
      toPE12 <= datain;
      toPE13 <= from1PE15;
      toPE14 <= from2PE15;
      toPE15 <= (OTHERS => 'W');

   WHEN state10 =>
      toPE11 <= (OTHERS => 'W');
      toPE12 <= (OTHERS => 'W');
      toPE13 <= datain;
      toPE14 <= from1PE15;
      toPE15 <= from2PE15;
      to1PE16 <= datain;

   WHEN state11 =>
      toPE11 <= (OTHERS => 'W');
```

```
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE15 <= from1PE15;
            to1PE16 <= datain;

         WHEN state12 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            to1PE16 <= datain;

         WHEN state13 =>
            toPE11 <= fromPE55;
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');
            to1PE16 <= datain;

         WHEN OTHERS =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');
      END CASE ;

   END PROCESS main;

END behav_muxbloc;


--
-- Component : control (Machine a etat du bloc de controle
-- Controleur A
-- Generated by System Architect version v8.5_3.3 by lesueur on Jun 15,
99
--
-- clock :: clk
-- reset :: reset
-- Source views :-
-- $CONTROL/control/types/types
--
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
LIBRARY control_control_sdslocal ;
USE control_control_sdslocal.types.all;

ENTITY control IS
   PORT (
      clk : IN clk_type;
      ready : IN std_logic;
      reset : IN rst_type;
      clkreseau : OUT clk_type;
      initPE11 : OUT init_type;
      initPE22 : OUT init_type;
```

```
      initPE33 : OUT init_type;
      initPE44 : OUT init_type;
      initPE55 : OUT init_type;
      modePE11 : OUT mode_type1;
      modePE12 : OUT mode_type2;
      modePE13 : OUT mode_type2;
      modePE14 : OUT mode_type2;
      modePE15 : OUT mode_type2;
      modePE16 : OUT mode_type1;
      modePE22 : OUT mode_type1;
      modePE23 : OUT mode_type1;
      modePE24 : OUT mode_type1;
      modePE25 : OUT mode_type1;
      modePE33 : OUT mode_type1;
      modePE34 : OUT mode_type1;
      modePE35 : OUT mode_type1;
      modePE44 : OUT mode_type1;
      modePE45 : OUT mode_type1;
      modePE55 : OUT mode_type1;
      request : OUT std_logic;
      rstreseau : OUT rst_type;
      state_mux : OUT state_type
   );
END control ;

--
-- Component : control
--
-- Generated by System Architect version v8.5_3.3 by lesueur on Jun 15,
99
--
-- clock :: clk rising
-- reset :: reset active_high synchronous_reset
-- animation_mode :: noanimate
-- compatible :: AutoLogic II
-- Source views :-
-- $CONTROL/control/state_machine
-- $CONTROL/control/types/types
--

ARCHITECTURE state_machine OF control IS
   TYPE control_state_type is (
      state00,
      state02,
      state03,
      state04,
      state05,
      state1,
      state2,
      state3,
      state4,
      state5,
      state6,
      state7,
      state8,
      state9,
      state10,
```

```
      state11,
      state12,
      state13,
      state01
   );


   -- SDS Defined State Signals
   SIGNAL current_state : control_state_type := state00 ;
   SIGNAL next_state : control_state_type := state00 ;
BEGIN

   ----------------------------------------------------------------------
   clocked : PROCESS (
      clk
   )
   ----------------------------------------------------------------------
   BEGIN
      IF ( clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0' ) THEN
         IF ( reset = '1' ) THEN
            current_state <= state00;
            -- Start State Actions
            request<='0';
            initPE11<='0';
            initPE22<='0';
            initPE33<='0';
            initPE44<='0';
            initPE55<='0';
            modePE11<= "000";
            modePE12<= "0000";
            modePE13<= "0000";
            modePE14<= "0000";
            modePE15<= "0000";
            modePE22<= "000";
            modePE23<= "000";
            modePE24<= "000";
            modePE25<= "000";
            modePE33<= "000";
            modePE34<= "000";
            modePE35<= "000";
            modePE44<= "000";
            modePE45<= "000";
            rstreseau <= reset;
            modePE55 <= "000";
            modePE16 <= "000";
         ELSE
            current_state <= next_state;

            -- State Actions
            CASE next_state IS
            WHEN state00 =>
               request<='0';
               initPE11<='0';
               initPE22<='0';
               initPE33<='0';
               initPE44<='0';
               initPE55<='0';
```

```
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "0000";
        modePE14<= "0000";
        modePE15<= "0000";
        modePE22<= "000";
        modePE23<= "000";
        modePE24<= "000";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        rstreseau <= reset;
        modePE55 <= "000";
        modePE16 <= "000";
    WHEN state02 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0010";
        modePE13 <= "0001";
        modePE14<= "0000";
        modePE15<= "0000";
        modePE22<= "001";
        modePE23<= "000";
        modePE24<= "000";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state03 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "1010";
        modePE14<= "0001";
        modePE15<= "0000";
        modePE22<= "010";
        modePE23<= "001";
        modePE24<= "000";
        modePE25<= "000";
```

```
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state04 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "0000";
        modePE14<= "1011";
        modePE15<= "0001";
        modePE22<= "011";
        modePE23<= "010";
        modePE24<= "001";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state05 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "0000";
        modePE14<= "0000";
        modePE15<= "0010";
        modePE22<= "000";
        modePE23<= "011";
        modePE24<= "010";
        modePE25<= "001";
        modePE33<= "010";
        modePE34<= "001";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "110";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state1 =>
```

```
        request<='0';
        initPE11<='1';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "010";
        modePE12<= "0000";
        modePE13<= "0000";
        modePE14<= "0111";
        modePE15<= "1001";
        modePE22<= "000";
        modePE23<= "101";
        modePE24<= "011";
        modePE25<= "010";
        modePE33<= "000";
        modePE34<= "010";
        modePE35<= "100";
        modePE44<= "001";
        modePE45<= "000";
        modePE55<= "010";
        rstreseau <= reset;
        modePE16 <= "011";
    WHEN state2 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "010";
        modePE12<= "0100";
        modePE13<= "0000";
        modePE14 <= "1001";
        modePE15<= "0110";
        modePE22<= "000";
        modePE23<= "000";
        modePE24 <= "011";
        modePE25<= "011";
        modePE33<= "011";
        modePE34<= "111";
        modePE35<= "010";
        modePE44<= "010";
        modePE45<= "100";
        modePE55<= "010";
        rstreseau <= reset;
        modePE16 <= "100";
    WHEN state3 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "010";
        modePE12<= "0100";
        modePE13<= "0100";
```

```
        modePE14<= "0000";
        modePE15<= "0111";
        modePE22<= "000";
        modePE23<= "000";
        modePE24<= "101";
        modePE25<= "011";
        modePE33<= "000";
        modePE34<= "011";
        modePE35<= "011";
        modePE44<= "101";
        modePE45<= "010";
        modePE55<= "011";
        rstreseau <= reset;
        modePE16 <= "101";
    WHEN state4 =>
        request<='0';
        initPE11<='0';
        initPE22<='1';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "010";
        modePE12<= "0100";
        modePE13<= "0100";
        modePE14<= "0100";
        modePE15<= "1010";
        modePE22<= "100";
        modePE23<= "000";
        modePE24<= "000";
        modePE25<= "011";
        modePE33<= "000";
        modePE34<= "101";
        modePE35<= "011";
        modePE44<= "101";
        modePE45<= "011";
        modePE55<= "100";
        rstreseau <= reset;
        modePE16 <= "110";
    WHEN state5 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "010";
        modePE12<= "0100";
        modePE13<= "0100";
        modePE14<= "0100";
        modePE15<= "0011";
        modePE22<= "100";
        modePE23<= "110";
        modePE24<= "000";
        modePE25<= "101";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "011";
```

```
        modePE44<= "011";
        modePE45<= "011";
        modePE55<= "010";
        rstreseau <= reset;
        modePE16 <= "111";
    WHEN state6 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "011";
        modePE12<= "0100";
        modePE13<= "0100";
        modePE14<= "0100";
        modePE15<= "0011";
        modePE22<= "100";
        modePE23<= "110";
        modePE24<= "110";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "101";
        modePE44<= "000";
        modePE45<= "011";
        modePE55<= "010";
        rstreseau <= reset;
        modePE16 <= "010";
    WHEN state7 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='1';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0101";
        modePE13<= "0100";
        modePE14<= "0100";
        modePE15<= "1000";
        modePE22<= "100";
        modePE23<= "110";
        modePE24<= "110";
        modePE25<= "110";
        modePE33<= "100";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "101";
        modePE55<= "010";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state8 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
```

```
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0110";
        modePE13<= "0101";
        modePE14<= "0100";
        modePE15<= "0011";
        modePE22<= "000";
        modePE23<= "110";
        modePE24<= "110";
        modePE25<= "110";
        modePE33<= "100";
        modePE34<= "110";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "101";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state9 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "001";
        modePE12<= "0001";
        modePE13<= "0110";
        modePE14<= "0101";
        modePE15<= "0011";
        modePE22<= "000";
        modePE23<= "000";
        modePE24<= "110";
        modePE25<= "110";
        modePE33<= "100";
        modePE34<= "110";
        modePE35<= "110";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state10 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='1';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0010";
        modePE13<= "0001";
        modePE14<= "0110";
        modePE15<= "0100";
        modePE22<= "001";
```

```
        modePE23<= "000";
        modePE24<= "000";
        modePE25<= "110";
        modePE33<= "000";
        modePE34<= "110";
        modePE35<= "110";
        modePE44<= "100";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state11 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0011";
        modePE13<= "1010";
        modePE14<= "0001";
        modePE15<= "0101";
        modePE22<= "010";
        modePE23<= "100";
        modePE24<= "000";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "110";
        modePE44<= "100";
        modePE45<= "110";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state12 =>
        request<='0';
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "1000";
        modePE14<= "1011";
        modePE15<= "0001";
        modePE22<= "011";
        modePE23<= "010";
        modePE24<= "100";
        modePE25<= "000";
        modePE33<= "001";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "110";
        modePE55<= "000";
```

```
            rstreseau <= reset;
            modePE16 <= "001";
         WHEN state13 =>
            request<='0';
            initPE11<='0';
            initPE22<='0';
            initPE33<='0';
            initPE44<='0';
            initPE55<='1';
            modePE11<= "000";
            modePE12<= "0000";
            modePE13<= "1001";
            modePE14<= "1000";
            modePE15<= "0010";
            modePE22<= "000";
            modePE23<= "011";
            modePE24<= "010";
            modePE25<= "100";
            modePE33<= "010";
            modePE34<= "100";
            modePE35<= "000";
            modePE44<= "000";
            modePE45<= "000";
            modePE55<= "001";
            rstreseau <= reset;
            modePE16 <= "010";
         WHEN state01 =>
            request<='0';
            initPE11<='0';
            initPE22<='0';
            initPE33<='0';
            initPE44<='0';
            initPE55<='0';
            modePE11<= "001";
            modePE12<= "0001";
            modePE13<= "0000";
            modePE14<= "0000";
            modePE15<= "0000";
            modePE22<= "000";
            modePE23<= "000";
            modePE24<= "000";
            modePE25<= "000";
            modePE33<= "000";
            modePE34<= "000";
            modePE35<= "000";
            modePE44<= "000";
            modePE45<= "000";
            modePE55<= "000";
            rstreseau <= reset;
            modePE16 <= "000";
         WHEN OTHERS =>
            NULL;
         END CASE;

      END IF;

   END IF;
```

```
END PROCESS clocked ;

-------------------------------------------------------------------
set_next_state : PROCESS (
   current_state,
   clk,
   ready,
   reset
)
-------------------------------------------------------------------
BEGIN
   next_state <= current_state;
   CASE current_state IS
   WHEN state00 =>
      IF ( TRUE ) THEN
         next_state <= state01;
      END IF;

   WHEN state02 =>
      IF ( TRUE ) THEN
         next_state <= state03;
      END IF;

   WHEN state03 =>
      IF ( TRUE ) THEN
         next_state <= state04;
      END IF;

   WHEN state04 =>
      IF ( TRUE ) THEN
         next_state <= state05;
      END IF;

   WHEN state05 =>
      IF ( TRUE ) THEN
         next_state <= state1;
      END IF;

   WHEN state1 =>
      IF ( TRUE ) THEN
         next_state <= state2;
      END IF;

   WHEN state2 =>
      IF ( TRUE ) THEN
         next_state <= state3;
      END IF;

   WHEN state3 =>
      IF ( TRUE ) THEN
         next_state <= state4;
      END IF;

   WHEN state4 =>
      IF ( TRUE ) THEN
         next_state <= state5;
```

```
        END IF;

    WHEN state5 =>
        IF ( TRUE ) THEN
            next_state <= state6;
        END IF;

    WHEN state6 =>
        IF ( TRUE ) THEN
            next_state <= state7;
        END IF;

    WHEN state7 =>
        IF ( TRUE ) THEN
            next_state <= state8;
        END IF;

    WHEN state8 =>
        IF ( TRUE ) THEN
            next_state <= state9;
        END IF;

    WHEN state9 =>
        IF ( TRUE ) THEN
            next_state <= state10;
        END IF;

    WHEN state10 =>
        IF ( TRUE ) THEN
            next_state <= state11;
        END IF;

    WHEN state11 =>
        IF ( TRUE ) THEN
            next_state <= state12;
        END IF;

    WHEN state12 =>
        IF ( TRUE ) THEN
            next_state <= state13;
        END IF;

    WHEN state13 =>
        IF ( TRUE ) THEN
            next_state <= state1;
        END IF;

    WHEN state01 =>
        IF ( TRUE ) THEN
            next_state <= state02;
        END IF;

    WHEN OTHERS =>
        NULL;
    END CASE;

END PROCESS set_next_state ;
```

```
-------------------------------------------------------------------
unclocked : PROCESS (
   current_state,
   clk,
   ready,
   reset
)
-------------------------------------------------------------------

-------------------------------------------------------------------
BEGIN

   -- State Actions
   CASE current_state IS
   WHEN state00 =>
      state_mux<= "00000";
      clkreseau <= clk;
   WHEN state02 =>
      state_mux<= "00010";
      clkreseau <= clk;
   WHEN state03 =>
      state_mux<= "00011";
      clkreseau <= clk;
   WHEN state04 =>
      state_mux<= "00100";
      clkreseau <= clk;
   WHEN state05 =>
      state_mux<= "00101";
      clkreseau <= clk;
   WHEN state1 =>
      state_mux<= "00110";
      clkreseau <= clk;
   WHEN state2 =>
      state_mux<= "00111";
      clkreseau <= clk;
   WHEN state3 =>
      state_mux<= "01000";
      clkreseau <= clk;
   WHEN state4 =>
      state_mux<= "01001";
      clkreseau <= clk;
   WHEN state5 =>
      state_mux<= "01010";
      clkreseau <= clk;
   WHEN state6 =>
      state_mux<= "01011";
      clkreseau <= clk;
   WHEN state7 =>
      state_mux<= "01100";
      clkreseau <= clk;
   WHEN state8 =>
      state_mux<= "01101";
      clkreseau <= clk;
   WHEN state9 =>
      state_mux<= "01110";
      clkreseau <= clk;
```

```
      WHEN state10 =>
         state_mux<= "01111";
         clkreseau <= clk;
      WHEN state11 =>
         state_mux<= "10000";
         clkreseau <= clk;
      WHEN state12 =>
         state_mux<= "10001";
         clkreseau <= clk;
      WHEN state13 =>
         state_mux<= "10010";
         clkreseau <= clk;
      WHEN state01 =>
         state_mux<= "00001";
         clkreseau <= clk;
      WHEN OTHERS =>
         NULL;
      END CASE;

   END PROCESS unclocked ;
END state_machine ;


------------------------------------------------------------
--
--  components.vhd
--       Paquetage de definition des composants
--
--  Par : Sebastien Lesueur
--       4 Mai 1999
--
------------------------------------------------------------

library IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE lib.constants.ALL;
   USE lib.type_def.ALL;

PACKAGE components IS

------------------------------------------------------------------------
--       DECLARATION DES PE RONDS DE TYPE RONA, RONB ET RONC          --
------------------------------------------------------------------------

COMPONENT rona
      PORT( clk            : IN clk_type;
            rst            : IN rst_type;
            mode           : IN mode_type1;
            initin         : IN init_type ;
            datain         : IN data_type ;
            cout, sout     : OUT data_type ;
            initout        : OUT init_type
            );
END COMPONENT;
COMPONENT ronb
      PORT( clk            : IN clk_type;
```

```
            rst             : IN rst_type;
            mode            : IN mode_type1;
            initin          : IN init_type ;
            datain          : IN data_type ;
            cout, sout      : OUT data_type ;
            initout         : OUT init_type
            );
END COMPONENT;

COMPONENT ronc
      PORT( clk             : IN clk_type;
            rst             : IN rst_type;
            mode            : IN mode_type1;
            initin          : IN init_type ;
            datain          : IN data_type ;
            cout, sout      : OUT data_type ;
            initout         : OUT init_type
            );
END COMPONENT;

-------------------------------------------------------------------------
--    DECLARATIONS DES PE CARRES DE TYPE CARA, CARB, CARC ET CARD     --
-------------------------------------------------------------------------


COMPONENT cara
      PORT( clk               : IN clk_type;
            rst               : IN rst_type;
            mode              : IN mode_type2;
            initin            : IN init_type ;
            datain            : IN data_type ;
            cin, sin          : IN data_type ;
            sout              : OUT data_type;
            cout              : OUT data_type;
            initout           : OUT init_type;
            dataout_dn        : OUT data_type
            );
END COMPONENT;


COMPONENT carb
      PORT( clk               : IN clk_type;
            rst               : IN rst_type;
            mode              : IN mode_type1;
            initin            : IN init_type ;
            datain            : IN data_type ;
            cin, sin          : IN data_type ;
            sout              : OUT data_type;
            cout              : OUT data_type;
            initout           : OUT init_type;
            dataout_dn        : OUT data_type
            );
END COMPONENT;


COMPONENT carc
         PORT( clk              : IN clk_type;
```

```
                rst                 : IN rst_type;
                mode                : IN mode_type2;
                initin              : IN init_type ;
                datain              : IN data_type ;
                cin, sin            : IN data_type ;
                sout                : OUT data_type;
                cout                : OUT data_type;
                initout             : OUT init_type;
                dataout_dn          : OUT data_type
                );
END COMPONENT;

COMPONENT card
          PORT( clk                 : IN clk_type;
                rst                 : IN rst_type;
                mode                : IN mode_type1;
                datain1             : IN data_type ;
                datain2             : IN data_type ;
                dataout_dn          : OUT data_type
                );
END COMPONENT;

-------------------------------------------------------------------
--              DECLARATION DU RESEAU DE PROCESSEURS             --
-------------------------------------------------------------------

COMPONENT reseau
   PORT( clkreseau                              : IN clk_type;
         rstreseau                              : IN rst_type;
         initPE11, initPE22, initPE33,
         initPE44, initPE55                     : IN init_type;
         modePE11                               : IN mode_type1;
         modePE12, modePE13, modePE14,
         modePE15                               : IN mode_type2;
         modePE16, modePE22,modePE23, modePE24,
         modePE25,modePE33, modePE34,
         modePE35,modePE44, modePE45,
         modePE55                               : IN mode_type1;
         datainPE11, datainPE12,
         datainPE13, datainPE14, datainPE15,
         datain1PE16, datain2PE16               : IN data_type;
         dataout1PE15, dataout2PE15, dataoutPE16,
         dataoutPE25, dataoutPE35,
         dataoutPE45, dataoutPE55               : OUT data_type
         );
END COMPONENT;


-------------------------------------------------------------------
--                      DECLARATION DU MULTIPLIXEUR              --
-------------------------------------------------------------------

COMPONENT muxbloc
          PORT( rst                 : IN rst_type;
                state               : IN state_type;
                datain              : IN data_type;
                from1PE15           : IN data_type;
```

```
                from2PE15          : IN data_type;
                fromPE16           : IN data_type;
                fromPE25           : IN data_type;
                fromPE35           : IN data_type;
                fromPE45           : IN data_type;
                fromPE55           : IN data_type;
                toPE11             : OUT data_type;
                toPE12             : OUT data_type;
                toPE13             : OUT data_type;
                toPE14             : OUT data_type;
                toPE15             : OUT data_type;
                to1PE16            : OUT data_type;
                to2PE16            : OUT data_type
                );
END COMPONENT ;


-------------------------------------------------------------------
--                DECLARATION DE LA MACHINE A ETAT               --
-------------------------------------------------------------------

COMPONENT sm
   PORT (clk : IN clk_type;
         rst : IN rst_type;
         stateout : OUT state_type);
END COMPONENT ;

-------------------------------------------------------------------
--                   DECLARATION DU BLOC DE CONTROLE             --
-------------------------------------------------------------------

COMPONENT control
            PORT( clk              : IN clk_type;
                  ready            : IN rst_type;
                  reset            : In rst_type;
                  clkreseau        : OUT clk_type;
                  initPE11         : OUT init_type;
                  initPE22         : OUT init_type;
                  initPE33         : OUT init_type;
                  initPE44         : OUT init_type;
                  initPE55         : OUT init_type;
                  modePE11         : OUT mode_type1;
                  modePE12         : OUT mode_type2;
                  modePE13         : OUT mode_type2;
                  modePE14         : OUT mode_type2;
                  modePE15         : OUT mode_type2;
                  modePE16         : OUT mode_type1;
                  modePE22         : OUT mode_type1;
                  modePE23         : OUT mode_type1;
                  modePE24         : OUT mode_type1;
                  modePE25         : OUT mode_type1;
                  modePE33         : OUT mode_type1;
                  modePE34         : OUT mode_type1;
                  modePE35         : OUT mode_type1;
                  modePE44         : OUT mode_type1;
                  modePE45         : OUT mode_type1;
                  modePE55         : OUT mode_type1;
```

```
                request         : OUT rst_type;
                rstreseau       : OUT rst_type;
                state_mux       : OUT state_type
            );

END COMPONENT ;

-------------------------------------------------------------------
--              DECLARATION DU PROCESSEUR A (charge)             --
-------------------------------------------------------------------

COMPONENT procA
        PORT( clk           : IN clk_type;
              reset         : IN rst_type;
              ready         : IN STD_LOGIC;
              request       : OUT STD_LOGIC;
              datain        : IN data_type;
              dataout       : OUT data_type
              );
END COMPONENT;

-------------------------------------------------------------------
--                DECLARATION DU PROCESSEUR B (moteur)           --
-------------------------------------------------------------------

COMPONENT procB
        PORT( clk           : IN clk_type;
              reset         : IN rst_type;
              ready         : IN STD_LOGIC;
              addr          : OUT addr_type;
              request       : OUT STD_LOGIC;
              datain        : IN data_type;
              data_top      : IN data_type;
              dataout       : OUT data_type
              );
END COMPONENT;


END components;
```

```
-------------------------------------------------------------------
--
```

```
-- carres.vhd
--    Description de l'entite et de l'architecture des PE carres
--    du controleur A
--
-- Par : Sebastien Lesueur
--       12 Avril 1999
--
--------------------------------------------------------------------------


LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE lib.constants.ALL;
   USE lib.type_def.ALL;
   USE lib.fonctions.ALL;

--------------------------------------------------------------------------
--                DECLARATION D'ENTITE POUR PE(1,2:4)                   --
--------------------------------------------------------------------------

ENTITY cara IS
         PORT( clk            : IN clk_type;
               rst            : IN rst_type;
               mode           : IN mode_type2;
               initin         : IN init_type ;
               datain         : IN data_type ;
               cin, sin       : IN data_type ;
               sout           : OUT data_type;
               cout           : OUT data_type;
               initout        : OUT init_type;
               dataout_dn     : OUT data_type
               );
END cara;


--------------------------------------------------------------------------
--                   ARCHITECTURE DE PE(1,2:4)                          --
--------------------------------------------------------------------------

ARCHITECTURE behav_cara OF cara IS

BEGIN

PROCESS
   VARIABLE tempdatain, temps, tempc, reg1, tempreg1, tempreg2,
      reg2, tempcout, tempsout, tempdataout_dn  : data_type;

   BEGIN


         WAIT UNTIL (clk'EVENT AND clk ='1');
         IF rst = '1' THEN
            -- Initialisation des registres internes
            reg1 := (OTHERS => '0');
```

```
      reg2 := (OTHERS => '0');

   ELSE
      -- Lecture des entrees
      tempdatain := datain;
      tempc := cin;
      temps := sin;

   CASE mode IS

         WHEN mode1b =>
            -- lambda passe au travers
            dataout_dn <= datain;

         WHEN mode2b =>
            -- Mode MAC pour PE12 pour calculer W*phi (phi(2)=Tau)
            cout <= produit(tempdatain, reg2);
            dataout_dn <= datain;

         WHEN mode3b =>
            -- Calcul et sortie a droite de J pour PE12 avec
            -- saturation
            reg1 := division(cunnor,reg2);
            IF reg1 > Jsup THEN
                  cout <= Jsup;
            ELSIF reg1 <Jinf THEN
                  cout <= Jinf;
            ELSE
                  cout <= reg1;
            END IF;

         WHEN mode4b =>
            -- Mode de triangularisation
            IF initin = '1' OR initin = 'H' THEN
               reg1 := tempdatain;
            ELSE
               tempreg1 := reg1;
               appli_rot(tempdatain, tempreg1, tempc, temps,
               tempcout, tempsout,
               reg1, tempdataout_dn);
               cout <= tempcout;
               sout <= tempsout;
               dataout_dn <= tempdataout_dn;
            END IF;
            initout <= initin;

         WHEN mode5b =>
               -- Division de G par F
               cout <= tempc;
               reg1 := produit(reg1, tempc);

         WHEN mode6b =>
               -- Mode MAC pour le calcul des nouveaux W=W+G/F*I
               reg2 := reg2 + produit(reg1, tempdatain);

         WHEN mode7b =>
               -- Donnee passe au travers de gauche a droite
```

```
                cout <= tempc;

            WHEN mode8b =>
                -- Calcul des W(i)/W(1) pour PE13 et PE14
                reg1 := produit(reg2,tempc);
                cout <= tempc;

            WHEN mode9b =>
                -- Sortie de W(2)/W(1) pour PE13
                IF reg1 > Fcsup THEN
                   reg1 := Fcsup;
                END IF;
                IF reg1 < Fcinf THEN
                   reg1 := Fcinf;
                END IF;
                cout <= reg1;

            WHEN mode10b =>
                -- Mode MAC pour PE13 pour calculer
                -- W*phi (phi(3)=-sign(w))

                IF tempdatain(19) = '1' THEN
                   cout <= tempc + reg2;
                   dataout_dn <= cunnor;
                ELSE
                   cout <= tempc - reg2;
                   dataout_dn <= -cunnor;
                END IF;
                -- w est sorti a droite sur le sinus
                sout <= datain;

            WHEN mode11b =>
                -- Mode MAC pour PE14 pour calculer
                -- W*phi (phi(3)=-w)
                cout <= tempc + produit(-temps, reg2);
                dataout_dn <= -temps;
                -- w est sorti a droite sur le sinus
                sout <= temps;

            WHEN mode12b =>
                -- Sortie de W(3)/W(1) pour PE14
                IF reg1 > Fvsup THEN
                   reg1 := Fvsup;
                END IF;
                IF reg1 < Fvinf THEN
                   reg1 := Fvinf;
                END IF;
                cout <= reg1;

            WHEN OTHERS =>
        END CASE;
      END IF;
   END PROCESS;

END behav_cara;
LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
```

```
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE lib.constants.ALL;
   USE lib.type_def.ALL;
   USE lib.fonctions.ALL;

-------------------------------------------------------------------------
--    DECLARATION D'ENTITE POUR PE(2,3:5), PE(3,4:5) ET PE(4,5)        --
-------------------------------------------------------------------------

ENTITY carb IS
         PORT( clk               : IN clk_type;
               rst               : IN rst_type;
               mode              : IN mode_type1;
               initin            : IN init_type ;
               datain            : IN data_type ;
               cin, sin          : IN data_type ;
               sout              : OUT data_type;
               cout              : OUT data_type;
               initout           : OUT init_type;
               dataout_dn        : OUT data_type
               );
END carb;

-------------------------------------------------------------------------
--             ARCHITECTURE DE PE(2,3:5), PE(3,4:5) ET PE(4,5)         --
-------------------------------------------------------------------------

ARCHITECTURE behav_carb OF carb IS
BEGIN

PROCESS
   VARIABLE tempdatain, temps, tempc, reg1, tempreg1, tempcout,
      tempsout, tempdataout_dn    : data_type;
   BEGIN


         WAIT UNTIL (clk'EVENT AND clk ='1');
         IF rst = '1' THEN
         reg1 := (OTHERS => '0');
         -- Initialisation du registre interne (valeur de S0)

         ELSE
            tempdatain := datain;
            tempc := cin;
            temps := sin;
         CASE mode IS

            WHEN mode1 =>
               -- lambda passe au travers
               dataout_dn <= datain;



            WHEN mode2 =>
               -- Mode MAC pour calculer S*phi
```

```
            cout <= tempc + produit(tempdatain, reg1);
            dataout_dn <= tempdatain;

         WHEN mode3 =>
            -- Donnee passe au travers de gauche a droite
            cout <= tempc;

         WHEN mode4 =>
            -- Multiplication de ST par lambda pour evaluer S+
            reg1 := produit(reg1,tempdatain);
            dataout_dn <= tempdatain;

         WHEN mode5 =>
            -- Sortie a droite du contenu du registre interne
            cout <= reg1;

         WHEN mode6 =>
            -- Triangularisation
            IF initin = '1' OR initin = 'H' THEN
               reg1 := tempdatain;
            ELSE
               tempreg1 := reg1;
               appli_rot(tempdatain, tempreg1, tempc, temps,
               tempcout, tempsout, reg1, tempdataout_dn);
               cout <= tempcout;
               sout <= tempsout;
               dataout_dn <= tempdataout_dn;

            END IF;
            initout <= initin;

         WHEN mode7 =>
               cout <= (OTHERS =>'0');

         WHEN OTHERS =>

      END CASE;
   END IF;

END PROCESS;
END behav_carb;


LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE lib.constants.ALL;
   USE lib.type_def.ALL;
   USE lib.fonctions.ALL;



------------------------------------------------------------
--          DECLARATION D'ENTITE POUR PE(1,5)             --
```

```
-----------------------------------------------------------------

ENTITY carc IS
        PORT( clk               : IN clk_type;
              rst               : IN rst_type;
              mode              : IN mode_type2;
              initin            : IN init_type;
              datain            : IN data_type;
              cin, sin          : IN data_type;
              sout              : OUT data_type;
              cout              : OUT data_type;
              initout           : OUT init_type;
              dataout_dn        : OUT data_type
              );
END carc;

-----------------------------------------------------------------
--                  ARCHITECTURE DE PE(1,5)                    --
-----------------------------------------------------------------


ARCHITECTURE behav_carc OF carc IS
BEGIN

PROCESS
   VARIABLE tempdatain, temps, tempc, regl, tempregl, tempreg2,
   reg2, reg3, reg4, tempcout, tempsout, tempdataout_dn  : data_type;

   BEGIN

        WAIT UNTIL (clk'EVENT AND clk ='1');
        IF rst = '1' THEN
        -- Initialisation des registres internes
        regl := (OTHERS => '0');
        reg2 := (OTHERS => '0');
        reg3 := (OTHERS => '0');
        reg4 := lambd2;

        ELSE
           tempdatain := datain;
           tempc := cin;
           temps := sin;
        CASE mode IS

              WHEN mode1b =>
                 -- Donnee passe au travers de haut en bas
                 dataout_dn <= datain;

              WHEN mode2b =>
                 -- Calcul de -sign(w)*exp(Kw^2)=phi(4)
                 -- pour PE15 et calcul de W(4)*phi(4)
                 tempdatain := shift2d(exp(shift2l(temps)));
                 IF temps(19) = '0' THEN
                    tempdatain := -tempdatain;
                 END IF;
                 reg3 := tempc + produit(tempdatain, reg2);
                 dataout_dn <= tempdatain;
```

```
WHEN mode3b =>
   -- Triangularisation
   IF initin = '1' OR initin = 'H' THEN
      reg1 := tempdatain;
   ELSE
      tempreg1 := reg1;
      appli_rot(tempdatain, tempreg1, tempc, temps,
      tempcout, tempsout, reg1, tempdataout_dn);
      sout <= reg3;
      dataout_dn <= tempdataout_dn;
   END IF;

WHEN mode4b =>
      -- Division de G par F
      reg1 := produit(reg1, tempc);

WHEN mode5b =>
      -- Mode MAC pour le calcul du nouveau W
      reg2 := reg2 + produit(reg1, tempdatain);

WHEN mode6b =>
      -- Donnee passe au travers de gauche a droite
      cout <= tempc;

WHEN mode7b =>
      -- Calcul de l'innovation
      reg3 := tempdatain+ (-reg3);
      cout <= tempc;

WHEN mode8b =>
      -- Calcul du nouveau coefficient 1/lambda
      -- et triangularisation
      IF ABS(reg3) > epse THEN
      ELSE
         reg4 := lambd2;
      END IF;

      tempreg1 := reg1;
      appli_rot(tempdatain, tempreg1, tempc, temps,
      tempcout, tempsout, reg1, tempdataout_dn);
      dataout_dn <= tempdataout_dn;
      sout <= reg3;
      cout <= reg4;

WHEN mode9b =>
      -- Calcul de W(4)/W(1)
      reg1 := produit(tempc, reg2);
      -- Sortie de 1/W(1)
      cout <= tempc;

WHEN mode10b =>
      -- Sortie de W(4)/W(1) pour PE15
      IF reg1 > Fssup THEN
         reg1 := Fssup;
      END IF;
      IF reg1 < Fsinf THEN
```

```
                        reg1 := Fsinf;
                    END IF;
                    cout <= reg1;

                WHEN OTHERS =>

            END CASE;
        END IF;
    END PROCESS;

END behav_carc;


LIBRARY IEEE, lib;
    USE IEEE.STD_LOGIC_1164.ALL;
    USE IEEE.STD_LOGIC_ARITH.ALL;
    USE IEEE.STD_LOGIC_SIGNED.ALL;

    USE lib.constants.ALL;
    USE lib.type_def.ALL;
    USE lib.fonctions.ALL;

-------------------------------------------------------------------
--                DECLARATION D'ENTITE DE PE(1,6)                --
-------------------------------------------------------------------

ENTITY card IS
          PORT( clk               : IN clk_type;
                rst               : IN rst_type;
                mode              : IN mode_type1;
                datain1           : IN data_type ;
                datain2           : IN data_type ;
                dataout_dn        : OUT data_type
                );
END card;


-------------------------------------------------------------------
--                   ARCHITECTURE DE PE(1,6)                     --
-------------------------------------------------------------------


ARCHITECTURE behav_card OF card IS
BEGIN

PROCESS
    VARIABLE tempdatain, tempc, reg, regwL, regTT, regaccMd, regwMd,
reg0Md, regwLd : data_type;
       BEGIN

          WAIT UNTIL (clk'EVENT AND clk ='1');
          IF rst = '1' THEN
          -- Initialisation des registres internes
          reg := (OTHERS => '0');
          regwL := (OTHERS => '0');
          regTT := (OTHERS => '0');
          reg0Md := (OTHERS => '0');
```

```
regwMd := (OTHERS => '0');
regaccMd := (OTHERS => '0');
dataout_dn <= (OTHERS => '0');
regwLd := (OTHERS => '0');

ELSE
   tempdatain := datain1;
   tempc := shift2l(datain2);

CASE mode IS

      WHEN mode0 =>
         regTT := (OTHERS => '0');
         -- Lecture de wL et stockage dans regwL
         regwL := shift2l(tempdatain);
         dataout_dn <= regTT;

      WHEN mode1 =>
         -- Calcul de KvLr*(wLd-wL) dans regTT
         reg := tempdatain - regwL;
         -- Calcul de (1/k)*KpLr*(wLd-wL) dans rewMd
         regTT := produit2(Kvlr, reg);
         regwMd := shift2l(produit2(Kplr,produit2(kinv,reg)));
         -- Stockage de wLd dans regwLd
         regwLd := tempdatain;

      WHEN mode2 =>
         -- Lecture de posL
         reg := tempdatain;
         dataout_dn <= regaccMd;

      WHEN mode3 =>
         -- Calcul de Kplr(posLd-posL)
         -- et accumulation dans regTT
         regTT := regTT + shift2l(produit2(Kplr,
                                  tempdatain-reg));
         -- Stockage de N*posLd dans posMd
         reg0Md := produit2(cunnor64,tempdatain);

      WHEN mode4 =>
         -- Calcul de Jm*(accL*+...)
         reg := regTT + tempdatain;
         regTT := produit2(tempc,reg);
         reg := tempc;
         regaccMd := tempdatain;

      WHEN mode5 =>
         IF regwL(19) = '1' THEN
            regTT := regTT - tempc;
         ELSE
            regTT := regTT + tempc;
         END IF;
         regwMd := produit2(cunnor64,regwLd)+
         produit2(reg,regwMd+produit2(Kvlr,
         produit2(kinv,regaccMd-shift2l(tempdatain))));
         regaccMd := produit2(cunnor64,regaccMd)+
         shift2l(produit2(Kplr,produit2(kinv,
```

```
                produit2(reg, regaccMd-shift21(tempdatain)))));
                dataout_dn <= regwMd;

            WHEN mode6 =>
                regTT := regTT + produit2(tempc, regwL);

            WHEN mode7 =>
                IF regwL(19) = '1' THEN
                    regTT := regTT - produit2(tempc, exp(regwL));
                ELSE
                    regTT := regTT + produit2(tempc, exp(regwL));
                END IF;
                reg0Md := reg0Md + produit2(kinv, regTT);
                dataout_dn <= reg0Md;

            WHEN OTHERS =>
        END CASE;
    END IF;

END PROCESS;
END behav_card;

-------------------------------------------------------------------------
--
-- ronds.vhd
--    Description de l'entite et de l'architecture des PE ronds
--    du controleur A
--
-- Par : Sebastien Lesueur
--    Avril 1999
--
-------------------------------------------------------------------------

LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE lib.constants.ALL;
   USE lib.type_def.ALL;
   USE lib.fonctions.ALL;


-------------------------------------------------------------------------
-----------------
--                          DECLARATION D'ENTITE POUR PE(1,1)
-------------------------------------------------------------------------
-----------------

ENTITY rona IS
    PORT( clk            : IN clk_type;
          rst            : IN rst_type;
          mode           : IN mode_type1;
          initin         : IN init_type ;
          datain         : IN data_type ;
          cout, sout     : OUT data_type ;
          initout        : OUT init_type
```

```
            );
END rona;

-------------------------------------------------------------------------
--                    ARCHITECTURE DE  PE(1,1)                         --
-------------------------------------------------------------------------

ARCHITECTURE behav_rona OF rona IS
BEGIN
PROCESS
   VARIABLE reg, tempreg, tempc, temps, tempdatain : data_type;

   BEGIN

         WAIT UNTIL (clk'EVENT AND clk = '1');
         IF rst = '1' THEN
               -- Initialisation du registre interne a 0
               reg := (OTHERS => '0');
               tempdatain := (OTHERS => '0');
         ELSE
         initout <= initin;
         tempdatain := datain;

         CASE mode IS

            WHEN mode1 =>
               -- 0 est sorti a droite
               cout <= (OTHERS => '0');

            WHEN mode2 =>
               -- Triangularisation
               IF (initin = '1') OR (initin = 'H') THEN
                  reg := tempdatain;
               ELSE
                  tempreg := reg;
                  gen_rot(tempdatain, tempreg, tempc, temps, reg);
                  cout <= tempc;
                  sout <= temps;
               END IF;

            WHEN mode3 =>
               -- Inversion de f
               tempreg := reg;
               reg := division(cunnor,tempreg);
               cout <= reg;

            WHEN OTHERS =>

         END CASE;
      END IF;

END PROCESS;

END behav_rona;

LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
```

```
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE lib.constants.ALL;
   USE lib.type_def.ALL;
   USE lib.fonctions.ALL;

-------------------------------------------------------------------------
--       DECLARATION D'ENTITE POUR PE(2,2), PE(3,3) ET PE(4,4)         --
-------------------------------------------------------------------------

ENTITY ronb IS
      PORT( clk            : IN clk_type;
            rst            : IN rst_type;
            mode           : IN mode_type1;
            initin         : IN init_type ;
            datain         : IN data_type ;
            cout, sout     : OUT data_type ;
            initout        : OUT init_type
            );
END ronb;

-------------------------------------------------------------------------
--             ARCHITECTURE DE PE(2,2), PE(3,3) ET PE(4,4)             --
-------------------------------------------------------------------------


ARCHITECTURE behav_ronb OF ronb IS

BEGIN

PROCESS
   VARIABLE tempreg, reg, tempc, temps, tempdatain : data_type;

   BEGIN


      WAIT UNTIL (clk'EVENT AND clk = '1');
      IF rst = '1' THEN
         reg := val_init_S;
         tempreg := (OTHERS => '0');

      ELSE
         initout <= initin;
         tempdatain := datain;

         CASE mode IS

            WHEN mode1 =>
               -- Produit de ST par 1/lambda
               tempreg := produit(reg,tempdatain);
               reg := tempreg;
               cout <= (OTHERS => '0');


            WHEN mode2 =>
               -- Produit de ST par phi
```

```
            cout <= produit(reg,tempdatain);

         WHEN mode3 =>
            --  Contenu du registre reg sorti a droite
            cout <= reg;

         WHEN mode4 =>
            -- Triangularisation
            IF (initin = '1') OR (initin = 'H') THEN
               reg := tempdatain;
            ELSE
               tempreg := reg;
               gen_rot(tempdatain, tempreg, tempc, temps, reg);
               cout <= tempc;
               sout <= temps;
            END IF;

         WHEN mode5 =>
               -- 0 est sorti a droite
               cout <= (OTHERS => '0');

         WHEN OTHERS =>

      END CASE;
   END IF;
END PROCESS;

END behav_ronb;


LIBRARY IEEE, lib;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE lib.constants.ALL;
   USE lib.type_def.ALL;
   USE lib.fonctions.ALL;

--------------------------------------------------------------------
--                  DECLARATION D'ENTITE POUR PE(5,5)             --
--------------------------------------------------------------------

ENTITY ronc IS
   PORT( clk            : IN clk_type;
         rst            : IN rst_type;
         mode           : IN mode_type1;
         initin         : IN init_type ;
         datain         : IN data_type ;
         cout, sout     : OUT data_type ;
         initout        : OUT init_type
         );
END ronc;


--------------------------------------------------------------------
--                  ARCHITECTURE DE  PE(5,5)                      --
```

```
--------------------------------------------------------------------------

ARCHITECTURE behav_ronc OF ronc IS

BEGIN

PROCESS
   VARIABLE tempreg, reg, tempc, temps, tempdatain : data_type;
   BEGIN


      WAIT UNTIL (clk'EVENT AND clk = '1');
      IF rst = '1' THEN
         reg := val_init_S;
         tempreg := (OTHERS => '0');

      ELSE
         initout <= initin;
         tempdatain := datain;
         CASE mode IS

            WHEN mode1 =>
                  -- Triangularisation
                  IF (initin = '1') OR (initin = 'H') THEN
                     reg := tempdatain;
                     cout <= beta;
                  ELSE
                     tempreg := reg;
                     gen_rot(tempdatain, tempreg, tempc, temps, reg);
                     cout <= beta;
                  END IF;

            WHEN mode2 =>
                  -- 0 est sorti a droite
                  cout <= (OTHERS => '0');

            WHEN mode3 =>
                  -- Produit de ST par lambda
                  tempreg := produit(reg,tempdatain);
                  reg := tempreg;
                  cout <= (OTHERS => '0');

            WHEN mode4 =>
                  -- Produit de ST par phi
                  cout <= produit(reg,tempdatain);

            WHEN mode5 =>
                  --  Contenu du registre reg sorti a droite
                  cout <= reg;

            WHEN mode6 =>
                     cout <= beta;

            WHEN OTHERS =>

         END CASE;
      END IF;
```

```
END PROCESS;

END behav_ronc;


-------------------------------------------------------------------------
---
--
-- proc.vhd
--        Entite et architecture pour le processeur B
--
--
-- Par : Sebastien Lesueur
--       20 Avril 1999
--
-- -----------------------------------------------------------------------


LIBRARY IEEE;
   USE IEEE.std_logic_1164.all;


LIBRARY libb, lib2b, lib3b;
   use libb.constants.all;
   use libb.type_def.all;
   use libb.components.all;

ENTITY procB IS
   PORT (clk          : IN clk_type;
         reset        : IN rst_type;
         ready        : IN STD_LOGIC;
         addr         : OUT addr_type;
         request      : OUT STD_LOGIC;
         datain       : IN data_type;
         data_top     : IN data_type;
         dataout      : OUT data_type
         );
END procB;


ARCHITECTURE struct_procB OF procB IS
   SIGNAL   datainPE11, datainPE12,
            datainPE13, datainPE14,
            datainPE15, datain1PE16,
            datain2Pe16,datain3PE16, dataout1PE15,
            dataout2PE15, dataoutPE16,
            dataoutPE25,dataoutPE35,
            dataoutPE45, dataoutPE55          : data_type;
   SIGNAL   stat                              : state_type;
   SIGNAL   ck                                : clk_type;
   SIGNAL   rst                               : rst_type;
   SIGNAL   initPE11, initPE22, initPE33,
            initPE44, initPE55                : init_type;
   SIGNAL   modePE11
                                              : mode_type1;
   SIGNAL   modePE12, modePE13, modePE14,
            modePE15                          : mode_type2;
```

```
   SIGNAL   modePE16, modePE22, modePE23,
            modePE24, modePE25, modePE33,
            modePE34,modePE35, modePE44,
            modePE45, modePE55              : mode_type1;

BEGIN

   reseau_proc : reseau PORT MAP(clkreseau => ck,
                                 rstreseau => rst,
                                 initPE11 => initPE11,
                                 initPE22 => initPE22,
                                 initPE33 => initPE33,
                                 initPE44 => initPE44,
                                 initPE55 => initPE55,
                                 modePE11 => modePE11,
                                 modePE12 => modePE12,
                                 modePE13 => modePE13,
                                 modePE14 => modePE14,
                                 modePE15 => modePE15,
                                 modePE16 => modePE16,
                                 modePE22 => modePE22,
                                 modePE23 => modePE23,
                                 modePE24 => modePE24,
                                 modePE25 => modePE25,
                                 modePE33 => modePE33,
                                 modePE34 => modePE34,
                                 modePE35 => modePE35,
                                 modePE44 => modePE44,
                                 modePE45 => modePE45,
                                 modePE55 => modePE55,
                                 datainPE11 => datainPE11,
                                 datainPE12 => datainPE12,
                                 datainPE13 => datainPE13,
                                 datainPE14 => datainPE14,
                                 datainPE15 => datainPE15,
                                 datain1PE16 => datain1PE16,
                                 datain2PE16 => data_top,
                                 datain3PE16 => datain3PE16,
                                 dataout1PE15 => dataout1PE15,
                                 dataout2PE15 => dataout2PE15,
                                 dataoutPE16 => dataoutPE16,
                                 dataoutPE25 => dataoutPE25,
                                 dataoutPE35 => dataoutPE35,
                                 dataoutPE45 => dataoutPE45,
                                 dataoutPE55 => dataoutPE55
                                 );

   mux : muxbloc PORT MAP( rst => reset,
                           state => stat,
                           datain => datain,
                           from1PE15 => dataout1PE15,
                           from2PE15 => dataout2PE15,
                           fromPE16 => dataoutPE16,
                           fromPE25 => dataoutPE25,
                           fromPE35 => dataoutPE35,
                           fromPE45 => dataoutPE45,
                           fromPE55 => dataoutPE55,
```

```
                              toPE11 => datainPE11,
                              toPE12 => datainPE12,
                              toPE13 => datainPE13,
                              toPE14 => datainPE14,
                              toPE15 => datainPE15,
                              to1PE16 => datain1PE16,
                              to2PE16 => datain3PE16,
                              dataout => dataout
                              );

   csm : control PORT MAP( clk => clk,
                              ready => ready,
                              reset => reset,
                              addr => addr,
                              clkreseau => ck,
                              initPE11 => initPE11,
                              initPE22 => initPE22,
                              initPE33 => initPE33,
                              initPE44 => initPE44,
                              initPE55 => initPE55,
                              modePE11 => modePE11,
                              modePE12 => modePE12,
                              modePE13 => modePE13,
                              modePE14 => modePE14,
                              modePE15 => modePE15,
                              modePE16 => modePE16,
                              modePE22 => modePE22,
                              modePE23 => modePE23,
                              modePE24 => modePE24,
                              modePE25 => modePE25,
                              modePE33 => modePE33,
                              modePE34 => modePE34,
                              modePE35 => modePE35,
                              modePE44 => modePE44,
                              modePE45 => modePE45,
                              modePE55 => modePE55,
                              request => request,
                              rstreseau => rst,
                              state_mux => stat
                              );

END struct_procB;

-- synopsys synthesis_off;

CONFIGURATION conf_procB OF procB IS
   FOR struct_procB
      FOR reseau_proc : reseau USE CONFIGURATION lib3b.conf_reseau;
      END FOR;

      FOR mux : muxbloc USE ENTITY lib3b.muxbloc(behav_muxbloc);
      END FOR;

      FOR csm : control USE ENTITY lib2b.control(data_flow);
      END FOR;
   END FOR;
```

```
END conf_procB;

-- synopsys synthesis_on;


--------------------------------------------------------------------------
--
-- Reseau.vhd
--    Description de l'entite et de l'architecture du reseau systolique
--    triangulaire du controleur B
--
-- Par : Sebastien Lesueur
--       12 Avril 1999
--
--------------------------------------------------------------------------


LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE libb.type_def.ALL;
   USE libb.components.ALL;


ENTITY reseau IS
   PORT( clkreseau                              : IN clk_type;
         rstreseau                              : IN rst_type;
         initPE11, initPE22, initPE33,
         initPE44, initPE55                     : IN init_type;
         modePE11                               : IN mode_type1;
         modePE12, modePE13,modePE14,
         modePE15                               : IN mode_type2;
         modePE16,modePE22,modePE23, modePE24,
         modePE25,modePE33, modePE34,
         modePE35,modePE44, modePE45,
         modePE55                               : IN mode_type1;
         datainPE11, datainPE12,datainPE13,
         datainPE14, datainPE15, datain1PE16,
         datain2PE16, datain3PE16               : IN data_type;
         dataout1PE15, dataout2PE15,
         dataoutPE16, dataoutPE25, dataoutPE35,
         dataoutPE45, dataoutPE55               : OUT data_type
       );
END reseau;

ARCHITECTURE struct_reseau OF reseau IS

SIGNAL s_sig1, c_sig1, data_sig1                : data_type4;
SIGNAL s_sig2, c_sig2, data_sig2                : data_type3;
SIGNAL s_sig3, c_sig3, data_sig3                : data_type2;
SIGNAL s_sig4, c_sig4, data_sig4                : data_type;
SIGNAL init_sig1                                : init_type4;
SIGNAL init_sig2                                : init_type3;
SIGNAL init_sig3                                : init_type2;
SIGNAL init_sig4                                : init_type;

BEGIN
```

```
PE11 : rona PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE11, initin => initPE11, datain => datainPE11,
cout => c_sig1(1), sout => s_sig1(1), initout => init_sig1(1));

PE12 : cara PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE12, initin => init_sig1(1), datain => datainPE12,
cin => c_sig1(1), sin => s_sig1(1), sout => s_sig1(2),
cout => c_sig1(2), initout => init_sig1(2),
dataout_dn => data_sig1(1));

PE13 : cara PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE13, initin => init_sig1(2), datain => datainPE13,
cin => c_sig1(2), sin => s_sig1(2), sout => s_sig1(3),
cout => c_sig1(3), initout=> init_sig1(3),
dataout_dn => data_sig1(2));

PE14 : cara PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE14, initin => init_sig1(3), datain => datainPE14,
cin => c_sig1(3), sin => s_sig1(3), sout => s_sig1(4),
cout => c_sig1(4), initout => init_sig1(4),
dataout_dn => data_sig1(3));

PE15 : carc PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE15, initin => init_sig1(4), datain => datainpe15,
cin => c_sig1(4), sin => s_sig1(4), sout => dataout2PE15,
cout => dataout1PE15, initout => OPEN, dataout_dn => data_sig1(4));

PE16 : card PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE16, datain1 => datain1PE16, datain2 => datain2PE16,
datain3 => datain3PE16, dataout => dataoutPE16);

PE22 : ronb PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE22, initin => initPE22, datain => data_sig1(1),
cout => c_sig2(1), sout => s_sig2(1), initout => init_sig2(1));


PE23 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE23, initin => init_sig2(1), datain => data_sig1(2),
cin => c_sig2(1), sin => s_sig2(1), cout => c_sig2(2),
sout => s_sig2(2), initout => init_sig2(2),
dataout_dn => data_sig2(1));

PE24 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE24, initin => init_sig2(2), datain => data_sig1(3),
cin => c_sig2(2), sin => s_sig2(2), cout => c_sig2(3),
sout => s_sig2(3), initout => init_sig2(3),
dataout_dn => data_sig2(2));

PE25 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE25, initin => init_sig2(3), datain => data_sig1(4),
cin => c_sig2(3), sin => s_sig2(3), cout => dataoutPE25,
sout => OPEN, initout => OPEN, dataout_dn => data_sig2(3));

PE33 : ronb PORT MAP(clk => clkreseau, rst => rstreseau,
mode => modePE33, initin => initPE33, datain => data_sig2(1),
cout => c_sig3(1), sout => s_sig3(1), initout => init_sig3(1));
```

```
   PE34 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE34, initin => init_sig3(1), datain => data_sig2(2),
   cin => c_sig3(1), sin => s_sig3(1), cout => c_sig3(2),
   sout => s_sig3(2), initout => init_sig3(2),
   dataout_dn => data_sig3(1));

   PE35 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE35, initin => init_sig3(2), datain => data_sig2(3),
   cin => c_sig3(2), sin => s_sig3(2), cout => dataoutPE35,
   sout => OPEN, initout => OPEN, dataout_dn => data_sig3(2));

   PE44 : ronb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE44, initin => initPE44, datain => data_sig3(1),
   cout => c_sig4, sout => s_sig4, initout => init_sig4);

   PE45 : carb PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE45, initin => init_sig4, datain => data_sig3(2),
   cin => c_sig4, sin => s_sig4, cout => dataoutPE45, sout => OPEN,
   initout => OPEN, dataout_dn => data_sig4);

   PE55 : ronc PORT MAP(clk => clkreseau, rst => rstreseau,
   mode => modePE55, initin => initPE55, datain => data_sig4,
   cout => dataoutPE55, sout => OPEN, initout => OPEN);

END struct_reseau;

CONFIGURATION conf_reseau OF reseau IS
   FOR struct_reseau

      FOR PE11 : rona USE ENTITY libb.rona(behav_rona);
      END FOR;

      FOR PE12 : cara USE ENTITY libb.cara(behav_cara);
      END FOR;

      FOR PE13 : cara USE ENTITY libb.cara(behav_cara);
      END FOR;

      FOR PE14 : cara USE ENTITY libb.cara(behav_cara);
      END FOR;

      FOR PE15 : carc USE ENTITY libb.carc(behav_carc);
      END FOR;

      FOR PE16 : card USE ENTITY libb.card(behav_card);
      END FOR;

      FOR PE22 : ronb USE ENTITY libb.ronb(behav_ronb);
      END FOR;

      FOR PE23 : carb USE ENTITY libb.carb(behav_carb);
      END FOR;

      FOR PE24 : carb USE ENTITY libb.carb(behav_carb);
      END FOR;
      FOR PE25 : carb USE ENTITY libb.carb(behav_carb);
      END FOR;
```

```
      FOR PE33 : ronb USE ENTITY libb.ronb(behav_ronb);
      END FOR;

      FOR PE34 : carb USE ENTITY libb.carb(behav_carb);
      END FOR;

      FOR PE44 : ronb USE ENTITY libb.ronb(behav_ronb);
      END FOR;

      FOR PE45 : carb USE ENTITY libb.carb(behav_carb);
      END FOR;

      FOR PE55 : ronc USE ENTITY libb.ronc(behav_ronc);
      END FOR;

   END FOR;
END conf_reseau;

-------------------------------------------------------------------------
--
--   muxbloc.vhd
--        Bloc de multiplexage des donnees du controleur B
--
-- Par :    Sebastien Lesueur
--          4 Mai 1999
--
-------------------------------------------------------------------------

LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE libb.constants.ALL;
   USE libb.type_def.ALL;

ENTITY muxbloc IS
            PORT( rst               : IN rst_type;
                  state             : IN state_type;
                  datain            : IN data_type;
                  from1PE15         : IN data_type;
                  from2PE15         : IN data_type;
                  fromPE16          : IN data_type;
                  fromPE25          : IN data_type;
                  fromPE35          : IN data_type;
                  fromPE45          : IN data_type;
                  fromPE55          : IN data_type;
                  toPE11            : OUT data_type;
                  toPE12            : OUT data_type;
                  toPE13            : OUT data_type;
                  toPE14            : OUT data_type;
                  toPE15            : OUT data_type;
                  to1PE16           : OUT data_type;
                  to2PE16           : OUT data_type;
                  dataout           : OUT data_type
                  );
END muxbloc ;
ARCHITECTURE behav_muxbloc OF muxbloc IS
```

```
BEGIN

   main : PROCESS (rst, state, datain, from1PE15, from2PE15, fromPE16,
fromPE25, fromPE35, fromPE45, fromPE55)

   BEGIN

      CASE state IS
         WHEN state00 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= lambd2;
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');
            to1PE16 <= (OTHERS => '0');
            to2PE16 <= (OTHERS => '0');
            dataout <= (OTHERS => '0');

         WHEN state01 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= datain;
            toPE13 <= lambd2;
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');

         WHEN state02 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= datain;
            toPE14 <= lambd2;
            toPE15 <= (OTHERS => 'W');

         WHEN state03 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE15 <= lambd2;

         WHEN state04 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');

         WHEN state05 =>
            toPE11 <= beta;
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');

         WHEN state1 =>
            toPE11 <= fromPE25;
            toPE12 <= fromPE55;
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');
```

```
    to2PE16 <= from1PE15;

WHEN state2 =>
    toPE11 <= fromPE35;
    toPE12 <= fromPE25;
    toPE13 <= fromPE55;
    toPE14 <= (OTHERS => 'W');
    toPE15 <= datain;
    to2PE16 <= from1PE15;

WHEN state3 =>
    toPE11 <= fromPE45;
    toPE12 <= fromPE35;
    toPE13 <= fromPE25;
    toPE14 <= fromPE55;
    toPE15 <= (OTHERS => 'W');
    to2PE16 <= from1PE15;

WHEN state4 =>
    toPE11 <= fromPE55;
    toPE12 <= fromPE45;
    toPE13 <= fromPE35;
    toPE14 <= fromPE25;
    toPE15 <= fromPE45;
    to2PE16 <= from1PE15;

WHEN state5 =>
    toPE11 <= (OTHERS => 'W');
    toPE12 <= fromPE55;
    toPE13 <= fromPE45;
    toPE14 <= fromPE35;
    toPE15 <= fromPE25;
    to1PE16 <= datain;

WHEN state6 =>
    toPE11 <= (OTHERS => 'W');
    toPE12 <= (OTHERS => 'W');
    toPE13 <= fromPE55;
    toPE14 <= fromPE45;
    toPE15 <= fromPE35;

WHEN state7 =>
    toPE11 <= (OTHERS => 'W');
    toPE12 <= from2PE15;
    toPE13 <= (OTHERS => 'W');
    toPE14 <= fromPE55;
    toPE15 <= fromPE45;

WHEN state8 =>
    toPE11 <= (OTHERS => 'W');
    toPE12 <= from1PE15;
    toPE13 <= from2PE15;
    toPE14 <= (OTHERS => 'W');
    toPE15 <= fromPE55;
    dataout <= fromPE16;
WHEN state9 =>
    toPE11 <= (OTHERS => 'W');
```

```
            toPE12 <= datain;
            toPE13 <= from1PE15;
            toPE14 <= from2PE15;
            toPE15 <= (OTHERS => 'W');

         WHEN state10 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= datain;
            toPE14 <= from1PE15;
            toPE15 <= from2PE15;
            to1PE16 <= datain;

         WHEN state11 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE15 <= from1PE15;

         WHEN state12 =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');

         WHEN state13 =>
            toPE11 <= fromPE55;
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');

         WHEN OTHERS =>
            toPE11 <= (OTHERS => 'W');
            toPE12 <= (OTHERS => 'W');
            toPE13 <= (OTHERS => 'W');
            toPE14 <= (OTHERS => 'W');
            toPE15 <= (OTHERS => 'W');
            dataout <= (OTHERS => 'W');
      END CASE ;

   END PROCESS main;

END behav_muxbloc;

--
-- Component : control
-- Machine a etat du bloc de contrôle pour le controleur B
-- Generated by System Architect version v8.5_3.3 by lesueur on Jun 09,
99
--
-- clock :: clk
-- reset :: reset
-- Source views :-
-- $CONTROLB/control/types/types
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
```

```
LIBRARY controlb_control_sdslocal ;
USE controlb_control_sdslocal.types.all;

ENTITY control IS
   PORT (
      clk : IN clk_type;
      ready : IN std_logic;
      reset : IN rst_type;
      addr : OUT addr_type;
      clkreseau : OUT clk_type;
      initPE11 : OUT init_type;
      initPE22 : OUT init_type;
      initPE33 : OUT init_type;
      initPE44 : OUT init_type;
      initPE55 : OUT init_type;
      modePE11 : OUT mode_type1;
      modePE12 : OUT mode_type2;
      modePE13 : OUT mode_type2;
      modePE14 : OUT mode_type2;
      modePE15 : OUT mode_type2;
      modePE16 : OUT mode_type1;
      modePE22 : OUT mode_type1;
      modePE23 : OUT mode_type1;
      modePE24 : OUT mode_type1;
      modePE25 : OUT mode_type1;
      modePE33 : OUT mode_type1;
      modePE34 : OUT mode_type1;
      modePE35 : OUT mode_type1;
      modePE44 : OUT mode_type1;
      modePE45 : OUT mode_type1;
      modePE55 : OUT mode_type1;
      request : OUT std_logic;
      rstreseau : OUT rst_type;
      state_mux : OUT state_type
   );
END control ;

--
-- Component : control
--
-- Generated by System Architect version v8.5_3.3 by lesueur on Jun 09,
99
-- clock :: clk rising
-- reset :: reset active_high synchronous_reset
-- animation_mode :: noanimate
-- compatible :: AutoLogic II
-- Source views :-
-- $CONTROLB/control/state_machine
-- $CONTROLB/control/types/types
--

ARCHITECTURE state_machine OF control IS
   TYPE control_state_type is (
      state00,
      state02,
      state03,
      state04,
```

```
      state05,
      state1,
      state2,
      state3,
      state4,
      state5,
      state6,
      state7,
      state8,
      state9,
      state10,
      state11,
      state12,
      state13,
      state01
   );


   -- SDS Defined State Signals
   SIGNAL current_state : control_state_type := state00 ;
   SIGNAL next_state : control_state_type := state00 ;
BEGIN

   ----------------------------------------------------------------------
   clocked : PROCESS (
      clk
   )
   ----------------------------------------------------------------------
   BEGIN
      IF ( clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0' ) THEN
         IF ( reset = '1' ) THEN
            current_state <= state00;
            -- Start State Actions
            request<='0';
            addr<= "000";
            initPE11<='0';
            initPE22<='0';
            initPE33<='0';
            initPE44<='0';
            initPE55<='0';
            modePE11<= "000";
            modePE12<= "0000";
            modePE13<= "0000";
            modePE14<= "0000";
            modePE15<= "0000";
            modePE22<= "000";
            modePE23<= "000";
            modePE24<= "000";
            modePE25<= "000";
            modePE33<= "000";
            modePE34<= "000";
            modePE35<= "000";
            modePE44<= "000";
            modePE45<= "000";
            rstreseau <= reset;
            modePE55 <= "000";
            modePE16 <= "000";
```

```
ELSE
   current_state <= next_state;

   -- State Actions
   CASE next_state IS
   WHEN state00 =>
      request<='0';
      addr<= "000";
      initPE11<='0';
      initPE22<='0';
      initPE33<='0';
      initPE44<='0';
      initPE55<='0';
      modePE11<= "000";
      modePE12<= "0000";
      modePE13<= "0000";
      modePE14<= "0000";
      modePE15<= "0000";
      modePE22<= "000";
      modePE23<= "000";
      modePE24<= "000";
      modePE25<= "000";
      modePE33<= "000";
      modePE34<= "000";
      modePE35<= "000";
      modePE44<= "000";
      modePE45<= "000";
      rstreseau <= reset;
      modePE55 <= "000";
      modePE16 <= "000";
   WHEN state02 =>
      request<='0';
      addr<= "000";
      initPE11<='0';
      initPE22<='0';
      initPE33<='0';
      initPE44<='0';
      initPE55<='0';
      modePE11<= "000";
      modePE12<= "0010";
      modePE13 <= "0001";
      modePE14<= "0000";
      modePE15<= "0000";
      modePE22<= "001";
      modePE23<= "000";
      modePE24<= "000";
      modePE25<= "000";
      modePE33<= "000";
      modePE34<= "000";
      modePE35<= "000";
      modePE44<= "000";
      modePE45<= "000";
      modePE55<= "000";
      rstreseau <= reset;
      modePE16 <= "000";
   WHEN state03 =>
      request<='0';
```

```
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "1010";
        modePE14<= "0001";
        modePE15<= "0000";
        modePE22<= "010";
        modePE23<= "001";
        modePE24<= "000";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state04 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "0000";
        modePE14<= "1011";
        modePE15<= "0001";
        modePE22<= "011";
        modePE23<= "010";
        modePE24<= "001";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state05 =>
        request<='0';
        addr<=(OTHERS => '0');
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
```

```
        modePE12<= "0000";
        modePE13<= "0000";
        modePE14<= "0000";
        modePE15<= "0010";
        modePE22<= "000";
        modePE23<= "011";
        modePE24<= "010";
        modePE25<= "001";
        modePE33<= "010";
        modePE34<= "001";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "110";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state1 =>
        request<='0';
        addr<= "000";
        initPE11<='1';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "010";
        modePE12<= "0000";
        modePE13<= "0000";
        modePE14<= "0111";
        modePE15<= "1001";
        modePE22<= "000";
        modePE23<= "101";
        modePE24<= "011";
        modePE25<= "010";
        modePE33<= "000";
        modePE34<= "010";
        modePE35<= "100";
        modePE44<= "001";
        modePE45<= "000";
        modePE55<= "010";
        rstreseau <= reset;
        modePE16 <= "001";
    WHEN state2 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "010";
        modePE12<= "0100";
        modePE13<= "0000";
        modePE14 <= "1001";
        modePE15<= "0110";
        modePE22<= "000";
        modePE23<= "000";
        modePE24 <= "011";
```

```
    modePE25<= "011";
    modePE33<= "011";
    modePE34<= "111";
    modePE35<= "010";
    modePE44<= "010";
    modePE45<= "100";
    modePE55<= "010";
    rstreseau <= reset;
    modePE16 <= "001";
WHEN state3 =>
    request<='0';
    addr<= "000";
    initPE11<='0';
    initPE22<='0';
    initPE33<='0';
    initPE44<='0';
    initPE55<='0';
    modePE11<= "010";
    modePE12<= "0100";
    modePE13<= "0100";
    modePE14<= "0000";
    modePE15<= "0111";
    modePE22<= "000";
    modePE23<= "000";
    modePE24<= "101";
    modePE25<= "011";
    modePE33<= "000";
    modePE34<= "011";
    modePE35<= "011";
    modePE44<= "101";
    modePE45<= "010";
    modePE55<= "011";
    rstreseau <= reset;
    modePE16 <= "010";
WHEN state4 =>
    request<='0';
    addr<= "000";
    initPE11<='0';
    initPE22<='1';
    initPE33<='0';
    initPE44<='0';
    initPE55<='0';
    modePE11<= "010";
    modePE12<= "0100";
    modePE13<= "0100";
    modePE14<= "0100";
    modePE15<= "1010";
    modePE22<= "100";
    modePE23<= "000";
    modePE24<= "000";
    modePE25<= "011";
    modePE33<= "000";
    modePE34<= "101";
    modePE35<= "011";
    modePE44<= "101";
    modePE45<= "011";
    modePE55<= "100";
```

```
    rstreseau <= reset;
    modePE16 <= "011";
WHEN state5 =>
    request<='0';
    addr<= "000";
    initPE11<='0';
    initPE22<='0';
    initPE33<='0';
    initPE44<='0';
    initPE55<='0';
    modePE11<= "010";
    modePE12<= "0100";
    modePE13<= "0100";
    modePE14<= "0100";
    modePE15<= "0011";
    modePE22<= "100";
    modePE23<= "110";
    modePE24<= "000";
    modePE25<= "101";
    modePE33<= "000";
    modePE34<= "000";
    modePE35<= "011";
    modePE44<= "011";
    modePE45<= "011";
    modePE55<= "010";
    rstreseau <= reset;
    modePE16 <= "100";
WHEN state6 =>
    request<='0';
    addr<= "000";
    initPE11<='0';
    initPE22<='0';
    initPE33<='0';
    initPE44<='0';
    initPE55<='0';
    modePE11<= "011";
    modePE12<= "0100";
    modePE13<= "0100";
    modePE14<= "0100";
    modePE15<= "0011";
    modePE22<= "100";
    modePE23<= "110";
    modePE24<= "110";
    modePE25<= "000";
    modePE33<= "000";
    modePE34<= "000";
    modePE35<= "101";
    modePE44<= "000";
    modePE45<= "011";
    modePE55<= "010";
    rstreseau <= reset;
    modePE16 <= "101";
WHEN state7 =>
    request<='0';
    addr<= "000";
    initPE11<='0';
    initPE22<='0';
```

```
        initPE33<='1';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0101";
        modePE13<= "0100";
        modePE14<= "0100";
        modePE15<= "1000";
        modePE22<= "100";
        modePE23<= "110";
        modePE24<= "110";
        modePE25<= "110";
        modePE33<= "100";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "101";
        modePE55<= "010";
        rstreseau <= reset;
        modePE16 <= "110";
    WHEN state8 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0110";
        modePE13<= "0101";
        modePE14<= "0100";
        modePE15<= "0011";
        modePE22<= "000";
        modePE23<= "110";
        modePE24<= "110";
        modePE25<= "110";
        modePE33<= "100";
        modePE34<= "110";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "101";
        rstreseau <= reset;
        modePE16 <= "111";
    WHEN state9 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "001";
        modePE12<= "0001";
        modePE13<= "0110";
        modePE14<= "0101";
```

```
        modePE15<= "0011";
        modePE22<= "000";
        modePE23<= "000";
        modePE24<= "110";
        modePE25<= "110";
        modePE33<= "100";
        modePE34<= "110";
        modePE35<= "110";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "001";
    WHEN state10 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='1';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0010";
        modePE13<= "0001";
        modePE14<= "0110";
        modePE15<= "0100";
        modePE22<= "001";
        modePE23<= "000";
        modePE24<= "000";
        modePE25<= "110";
        modePE33<= "000";
        modePE34<= "110";
        modePE35<= "110";
        modePE44<= "100";
        modePE45<= "000";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "001";
    WHEN state11 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0011";
        modePE13<= "1010";
        modePE14<= "0001";
        modePE15<= "0101";
        modePE22<= "010";
        modePE23<= "100";
        modePE24<= "000";
        modePE25<= "000";
        modePE33<= "000";
        modePE34<= "000";
```

```
        modePE35<= "110";
        modePE44<= "100";
        modePE45<= "110";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "000";
    WHEN state12 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='0';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "1000";
        modePE14<= "1011";
        modePE15<= "0001";
        modePE22<= "011";
        modePE23<= "010";
        modePE24<= "100";
        modePE25<= "000";
        modePE33<= "001";
        modePE34<= "000";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "110";
        modePE55<= "000";
        rstreseau <= reset;
        modePE16 <= "001";
    WHEN state13 =>
        request<='0';
        addr<= "000";
        initPE11<='0';
        initPE22<='0';
        initPE33<='0';
        initPE44<='0';
        initPE55<='1';
        modePE11<= "000";
        modePE12<= "0000";
        modePE13<= "1001";
        modePE14<= "1000";
        modePE15<= "0010";
        modePE22<= "000";
        modePE23<= "011";
        modePE24<= "010";
        modePE25<= "100";
        modePE33<= "010";
        modePE34<= "100";
        modePE35<= "000";
        modePE44<= "000";
        modePE45<= "000";
        modePE55<= "001";
        rstreseau <= reset;
        modePE16 <= "001";
    WHEN state01 =>
```

```
            request<='0';
            addr<= "000";
            initPE11<='0';
            initPE22<='0';
            initPE33<='0';
            initPE44<='0';
            initPE55<='0';
            modePE11<= "001";
            modePE12<= "0001";
            modePE13<= "0000";
            modePE14<= "0000";
            modePE15<= "0000";
            modePE22<= "000";
            modePE23<= "000";
            modePE24<= "000";
            modePE25<= "000";
            modePE33<= "000";
            modePE34<= "000";
            modePE35<= "000";
            modePE44<= "000";
            modePE45<= "000";
            modePE55<= "000";
            rstreseau <= reset;
            modePE16 <= "000";
         WHEN OTHERS =>
            NULL;
         END CASE;

      END IF;

   END IF;

END PROCESS clocked ;

-----------------------------------------------------------------------
set_next_state : PROCESS (
   current_state,
   clk,
   ready,
   reset
)
-----------------------------------------------------------------------
BEGIN
   next_state <= current_state;
   CASE current_state IS
   WHEN state00 =>
      IF ( TRUE ) THEN
         next_state <= state01;
      END IF;

   WHEN state02 =>
      IF ( TRUE ) THEN
         next_state <= state03;
      END IF;

   WHEN state03 =>
      IF ( TRUE ) THEN
```

```
            next_state <= state04;
        END IF;

    WHEN state04 =>
        IF ( TRUE ) THEN
            next_state <= state05;
        END IF;

    WHEN state05 =>
        IF ( TRUE ) THEN
            next_state <= state1;
        END IF;

    WHEN state1 =>
        IF ( TRUE ) THEN
            next_state <= state2;
        END IF;

    WHEN state2 =>
        IF ( TRUE ) THEN
            next_state <= state3;
        END IF;

    WHEN state3 =>
        IF ( TRUE ) THEN
            next_state <= state4;
        END IF;

    WHEN state4 =>
        IF ( TRUE ) THEN
            next_state <= state5;
        END IF;

    WHEN state5 =>
        IF ( TRUE ) THEN
            next_state <= state6;
        END IF;

    WHEN state6 =>
        IF ( TRUE ) THEN
            next_state <= state7;
        END IF;

    WHEN state7 =>
        IF ( TRUE ) THEN
            next_state <= state8;
        END IF;

    WHEN state8 =>
        IF ( TRUE ) THEN
            next_state <= state9;
        END IF;

    WHEN state9 =>
        IF ( TRUE ) THEN
            next_state <= state10;
        END IF;
```

```
    WHEN state10 =>
       IF ( TRUE ) THEN
          next_state <= state11;
       END IF;

    WHEN state11 =>
       IF ( TRUE ) THEN
          next_state <= state12;
       END IF;

    WHEN state12 =>
       IF ( TRUE ) THEN
          next_state <= state13;
       END IF;

    WHEN state13 =>
       IF ( TRUE ) THEN
          next_state <= state1;
       END IF;

    WHEN state01 =>
       IF ( TRUE ) THEN
          next_state <= state02;
       END IF;

    WHEN OTHERS =>
       NULL;
    END CASE;

END PROCESS set_next_state ;

-----------------------------------------------------------------------
unclocked : PROCESS (
   current_state,
   clk,
   ready,
   reset
)
-----------------------------------------------------------------------

-----------------------------------------------------------------------
BEGIN

   -- State Actions
   CASE current_state IS
   WHEN state00 =>
      state_mux<= "00000";
      clkreseau <= clk;
   WHEN state02 =>
      state_mux<= "00010";
      clkreseau <= clk;
   WHEN state03 =>
      state_mux<= "00011";
      clkreseau <= clk;
   WHEN state04 =>
      state_mux<= "00100";
```

```
         clkreseau <= clk;
      WHEN state05 =>
         state_mux<= "00101";
         clkreseau <= clk;
      WHEN state1 =>
         state_mux<= "00110";
         clkreseau <= clk;
      WHEN state2 =>
         state_mux<= "00111";
         clkreseau <= clk;
      WHEN state3 =>
         state_mux<= "01000";
         clkreseau <= clk;
      WHEN state4 =>
         state_mux<= "01001";
         clkreseau <= clk;
      WHEN state5 =>
         state_mux<= "01010";
         clkreseau <= clk;
      WHEN state6 =>
         state_mux<= "01011";
         clkreseau <= clk;
      WHEN state7 =>
         state_mux<= "01100";
         clkreseau <= clk;
      WHEN state8 =>
         state_mux<= "01101";
         clkreseau <= clk;
      WHEN state9 =>
         state_mux<= "01110";
         clkreseau <= clk;
      WHEN state10 =>
         state_mux<= "01111";
         clkreseau <= clk;
      WHEN state11 =>
         state_mux<= "10000";
         clkreseau <= clk;
      WHEN state12 =>
         state_mux<= "10001";
         clkreseau <= clk;
      WHEN state13 =>
         state_mux<= "10010";
         clkreseau <= clk;
      WHEN state01 =>
         state_mux<= "00001";
         clkreseau <= clk;
      WHEN OTHERS =>
         NULL;
      END CASE;

   END PROCESS unclocked ;
END state_machine ;


--------------------------------------------------------------------
--
```

```
-- carres.vhd
--    Description de l'entite et de l'architecture des PE carres
--    du controleur B
--
-- Par : Sebastien Lesueur
--       12 Avril 1999
--
--------------------------------------------------------------------


LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE libb.constants.ALL;
   USE libb.type_def.ALL;
   USE libb.fonctions.ALL;

--------------------------------------------------------------------
--                Declararion d'entite pour PE(1,2:4)             --
--------------------------------------------------------------------

ENTITY cara IS
         PORT( clk            : IN clk_type;
               rst            : IN rst_type;
               mode           : IN mode_type2;
               initin         : IN init_type ;
               datain         : IN data_type ;
               cin, sin       : IN data_type ;
               sout           : OUT data_type;
               cout           : OUT data_type;
               initout        : OUT init_type;
               dataout_dn     : OUT data_type
               );
END cara;

--------------------------------------------------------------------
--                   Architecture pour PE(1,2:4)                  --
--------------------------------------------------------------------

ARCHITECTURE behav_cara OF cara IS

BEGIN

PROCESS
   VARIABLE tempdatain, temps, tempc, reg1, tempreg1, tempreg2,
      reg2, tempcout, tempsout, tempdataout_dn  : data_type;

   BEGIN
         WAIT UNTIL (clk'EVENT AND clk ='1');
         IF rst = '1' THEN
            -- Initialisation des registres internes
            reg1 := (OTHERS => '0');
            reg2 := (OTHERS => '0');
         ELSE
            tempdatain := datain;
```

```
      tempc := cin;
      temps := sin;

   CASE mode IS

         WHEN mode1b =>
            -- lambda passe au travers
            dataout_dn <= datain;

         WHEN mode2b =>
            -- Mode MAC pour PE12 pour calculer W*phi (phi(2)=Tau)
            cout <= produit(tempdatain, reg2);
            dataout_dn <= datain;

         WHE mode3b =>
            -- Calcul et sortie a droite de J pour PE12
            -- avec saturation
            reg1 := division(cunnor,reg2);
            IF reg1 > Jsup THEN
                  cout <= Jsup;
            ELSIF reg1 <Jinf THEN
                  cout <= Jinf;
            ELSE
                  cout <= reg1;
            END IF;

         WHEN mode4b =>
            -- Triangularisation
            IF initin = '1' OR initin = 'H' THEN
               reg1 := tempdatain;
            ELSE
               tempreg1 := reg1;
               appli_rot(tempdatain, tempreg1, tempc, temps,
               tempcout, tempsout,
               reg1, tempdataout_dn);
               cout <= tempcout;
               sout <= tempsout;
               dataout_dn <= tempdataout_dn;
            END IF;
            initout <= initin;

         WHEN mode5b =>
            -- Division de G par F
            cout <= tempc;
            reg1 := produit(reg1, tempc);

         WHEN mode6b =>
            -- Mode MAC pour le calcul des nouveaux W
            reg2 := reg2 + produit(reg1, tempdatain);

         WHEN mode7b =>
            cout <= tempc;

         WHEN mode8b =>
            -- Calcul des W(i)/W(1) pour PE13 et PE14
            reg1 := produit(reg2,tempc);
            cout <= tempc;
```

```
                WHEN mode9b =>
                   -- Sortie de W(2)/W(1) pour PE13

                   IF reg1 > Fcsup THEN
                      reg1 := Fcsup;
                   END IF;
                   IF reg1 < Fcinf THEN
                      reg1 := Fcinf;
                   END IF;
                   cout <= reg1;

                WHEN mode10b =>
                   -- Mode MAC pour PE13 pour calculer
                   -- W*phi (phi(3)=-sign(w))

                   IF tempdatain(19) = '1' THEN
                      cout <= tempc + reg2;
                      dataout_dn <= cunnor;
                   ELSE
                      cout <= tempc - reg2;
                      dataout_dn <= -cunnor;
                   END IF;
                   --dataout_dn <= tempdatain;
                   -- w est sorti a droite sur le sinus
                   sout <= datain;

                WHEN mode11b =>
                   -- Mode MAC pour PE14 pour calculer W*phi (phi(3)=-w)
                   cout <= tempc + produit(-temps, reg2);
                   dataout_dn <= -temps;
                   -- w est sorti a droite sur le sinus
                   sout <= temps;

                WHEN mode12b =>
                   -- Sortie de W(3)/W(1) pour PE14
                   IF reg1 > Fvsup THEN
                      reg1 := Fvsup;
                   END IF;
                   IF reg1 < Fvinf THEN
                      reg1 := Fvinf;
                   END IF;
                   cout <= reg1;

                WHEN OTHERS =>

          END CASE;

       END IF;
    END PROCESS;

END behav_cara;



LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
```

```
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE libb.constants.ALL;
   USE libb.type_def.ALL;
   USE libb.fonctions.ALL;

-------------------------------------------------------------------------
--          Architecture pour PE(2,3:5), PE(3,4:5) et PE(4,5)          --
-------------------------------------------------------------------------

ENTITY carb IS
         PORT( clk               : IN clk_type;
               rst               : IN rst_type;
               mode              : IN mode_typel;
               initin            : IN init_type ;
               datain            : IN data_type ;
               cin, sin          : IN data_type ;
               sout              : OUT data_type;
               cout              : OUT data_type;
               initout           : OUT init_type;
               dataout_dn        : OUT data_type
               );
END carb;

ARCHITECTURE behav_carb OF carb IS
BEGIN

PROCESS
   VARIABLE tempdatain, temps, tempc, regl, tempregl, tempcout,
      tempsout, tempdataout_dn   : data_type;
   BEGIN


         WAIT UNTIL (clk'EVENT AND clk ='1');
         IF rst = '1' THEN
         -- Initialisation du registre interne (valeur de S0)
         regl := (OTHERS => '0');

         ELSE
            tempdatain := datain;
            tempc := cin;
            temps := sin;
         CASE mode IS

            WHEN model =>
               -- lambda passe au travers
               dataout_dn <= datain;

            WHEN mode2 =>
               -- Mode MAC pour calculer S*phi
               cout <= tempc + produit(tempdatain, regl);
               dataout_dn <= tempdatain;


            WHEN mode3 =>
               -- Donnee passe au travers de gauche a droite
```

```
               cout <= tempc;

            WHEN mode4 =>
               -- Multiplication de ST par lambda pour evaluer S+
               reg1 := produit(reg1,tempdatain);
               dataout_dn <= tempdatain;

            WHEN mode5 =>
               -- Sortie a droite du contenu du registre interne
               cout <= reg1;

            WHEN mode6 =>
               -- Triangularisation
               IF initin = '1' OR initin = 'H' THEN
                  reg1 := tempdatain;
               ELSE
                  tempreg1 := reg1;
                  appli_rot(tempdatain, tempreg1, tempc, temps,
                  tempcout, tempsout, reg1, tempdataout_dn);
                  cout <= tempcout;
                  sout <= tempsout;
                  dataout_dn <= tempdataout_dn;
               END IF;
               initout <= initin;

            WHEN mode7 =>
                  cout <= (OTHERS =>'0');

            WHEN OTHERS =>

         END CASE;
      END IF;

   END PROCESS;
END behav_carb;


LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;
   USE libb.constants.ALL;
   USE libb.type_def.ALL;
   USE libb.fonctions.ALL;


--------------------------------------------------------------------
--               Declararion d'entite pour les PE(1,5)            --
--------------------------------------------------------------------

ENTITY carc IS
         PORT( clk              : IN clk_type;
               rst              : IN rst_type;
               mode             : IN mode_type2;
               initin           : IN init_type ;
               datain           : IN data_type ;
               cin, sin         : IN data_type ;
```

```
                sout                : OUT data_type;
                cout                : OUT data_type;
                initout             : OUT init_type;
                dataout_dn          : OUT data_type
                );
END carc;


-------------------------------------------------------------------------
--                    Architecture pour PE(1,5)                        --
-------------------------------------------------------------------------


ARCHITECTURE behav_carc OF carc IS
BEGIN

PROCESS
   VARIABLE tempdatain, temps, tempc, reg1, tempreg1, tempreg2,
   reg2, reg3, reg4, tempcout, tempsout, tempdataout_dn  : data_type;

   BEGIN


         WAIT UNTIL (clk'EVENT AND clk ='1');
         IF rst = '1' THEN
         -- Initialisation des registres internes
         reg1 := (OTHERS => '0');
         reg2 := (OTHERS => '0');
         reg3 := (OTHERS => '0');
         reg4 := lambd2;
         cout <= (OTHERS => '0');

         ELSE
            tempdatain := datain;
            tempc := cin;
            temps := sin;
         CASE mode IS

               WHEN mode1b =>
                  -- Donnee passe au travers de haut en bas
                  dataout_dn <= datain;

               WHEN mode2b =>
                  -- Calcul de -sign(w)*exp(Kw^2)=phi(4) pour PE15
                  -- et calcul de W(4)*phi(4)
                  tempdatain := shift2d(exp(shift2l(temps)));
                  IF temps(19) = '0' THEN
                     tempdatain := -tempdatain;
                  END IF;
                  reg3 := tempc + produit(tempdatain, reg2);
                  dataout_dn <= tempdatain;

               WHEN mode3b =>
                  -- Triangularisation
                  IF initin = '1' OR initin = 'H' THEN
                     reg1 := tempdatain;
                  ELSE
```

```
                tempreg1 := reg1;
                appli_rot(tempdatain, tempreg1, tempc, temps,
                tempcout, tempsout, reg1, tempdataout_dn);
                sout <= reg3;
                dataout_dn <= tempdataout_dn;
             END IF;

          WHEN mode4b =>
             -- Division de G par F
             reg1 := produit(reg1, tempc);

          WHEN mode5b =>
             -- Mode MAC pour le calcul du nouveau W
             reg2 := reg2 + produit(reg1, tempdatain);

          WHEN mode6b =>
             -- Donnee passe au travers de gauche a droite
             cout <= tempc;

          WHEN mode7b =>
             -- Calcul de l'innovation et sortie de W
             reg3 := tempdatain+ (-reg3);
             cout <= tempc;

          WHEN mode8b =>
             -- Calcul du nouveau coefficient 1/lambda
             IF ABS(reg3) > epse THEN
                reg4 := lambd1;
             ELSE
                reg4 := lambd2;
             END IF;
             tempreg1 := reg1;
             appli_rot(tempdatain, tempreg1, tempc, temps,
             tempcout, tempsout, reg1, tempdataout_dn);
             dataout_dn <= tempdataout_dn;
             sout <= reg3;
             cout <= reg4;

          WHEN mode9b =>
             reg1 := produit(tempc, reg2);
             cout <= tempc;

          WHEN mode10b =>
             -- Sortie de W(4)/W(1) pour PE15
             IF reg1 > Fssup THEN
                reg1 := Fssup;
             END IF;
             IF reg1 < Fsinf THEN
                reg1 := Fsinf;
             END IF;
             cout <= reg1;

          WHEN OTHERS =>
       END CASE;
    END IF;
END PROCESS;
```

```
END behav_carc;


LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE libb.constants.ALL;
   USE libb.type_def.ALL;
   USE libb.fonctions.ALL;

-------------------------------------------------------------------------
--                Declararion d'entite pour les PE(1,6)                --
-------------------------------------------------------------------------

ENTITY card IS
      PORT( clk               : IN clk_type;
            rst               : IN rst_type;
            mode              : IN mode_type1;
            datain1           : IN data_type ;  -- Donnees de
                                                -- l'exterieur
            datain2           : IN data_type ;  -- Donnees de PE16
                                                -- du reseau A
            datain3           : IN data_type ;  -- Donnees du
                                                -- reseau triangulaire
            dataout           : OUT data_type
            );
END card;


-------------------------------------------------------------------------
--                    Architecture pour PE(1,6)                        --
-------------------------------------------------------------------------


ARCHITECTURE behav_card OF card IS
BEGIN

PROCESS
   VARIABLE tempdatain1, tempdatain2,
            tempc, regJM, regwM, regTM, regTFM, reg : data_type;
      BEGIN

         WAIT UNTIL (clk'EVENT AND clk ='1');
         IF rst = '1' THEN
         -- Initialisation des registres internes
         regwM := (OTHERS => '0');
         regTM := (OTHERS => '0');
         regJM := (OTHERS => '0');
         ELSE
            tempdatain1 := datain1;
            tempdatain2 := datain2;
            tempc := shift2l(datain3);

         CASE mode IS
```

```
                WHEN mode0 =>
                   -- Lecture et stockage de wM
                   regwM := shift2l(tempdatain1);

                WHEN mode1 =>
                   -- Lecture de J estimee
                   regJM := tempc;

                WHEN mode2 =>
                   IF regwM(19) = '1' THEN
                      regTFM := -tempc;
                   ELSE
                      regTFM := tempc;
                   END IF;

                WHEN mode3 =>
                   regTFM := regTFM + produit2(tempc, regwM);
                   reg := tempdatain2 - regwM;
                   regTM := produit2(Kvmr, reg);

                WHEN mode4 =>
                   IF regwM(19) = '1' THEN
                      regTFM := regTFM - produit2(tempc, exp(regwM));
                   ELSE
                      regTFM := regTFM + produit2(tempc, exp(regwM));
                   END IF;

                WHEN mode5 =>
                   reg := tempdatain2 - tempdatain1;
                   regTM := regTM + shift2l(produit2(Kpmr,reg));

                WHEN mode6 =>
                   reg := tempdatain2 + regTM;
                   regTM := regTFM + produit2(regJM, reg);

                WHEN mode7 =>
                   regTM := regTM + produit2(cunnor64, tempdatain2);
                   dataout <= regTM;
                WHEN OTHERS =>
             END CASE;
          END IF;

END PROCESS;
END behav_card;
```

```
--------------------------------------------------------------------
--
```

```
-- ronds.vhd
--    Description de l'entite et de l'architecture des PE ronds
--    du controleur B
--
-- Par : Sebastien Lesueur
--    Avril 1999
--
--------------------------------------------------------------------------

LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE libb.constants.ALL;
   USE libb.type_def.ALL;
   USE libb.fonctions.ALL;


--------------------------------------------------------------------------
--                    DECLARATION D'ENTITE POUR PE(1,1)                 --
--------------------------------------------------------------------------

ENTITY rona IS
      PORT( clk            : IN clk_type;
            rst            : IN rst_type;
            mode           : IN mode_type1;
            initin         : IN init_type ;
            datain         : IN data_type ;
            cout, sout     : OUT data_type ;
            initout        : OUT init_type
            );
END rona;

--------------------------------------------------------------------------
--                    ARCHITECTURE DE  PE(1,1)                          --
--------------------------------------------------------------------------

ARCHITECTURE behav_rona OF rona IS
BEGIN
PROCESS
   VARIABLE reg, tempreg, tempc, temps, tempdatain : data_type;

   BEGIN

         WAIT UNTIL (clk'EVENT AND clk = '1');
         IF rst = '1' THEN
               -- Initialisation du registre interne a 0
               reg := (OTHERS => '0');
               tempdatain := (OTHERS => '0');
         ELSE
         initout <= initin;
         tempdatain := datain;


         CASE mode IS
            WHEN mode1 =>
```

```
            -- 0 est sorti a droite
            cout <= (OTHERS => '0');

         WHEN mode2 =>
            -- Triangularisation
            IF (initin = '1') OR (initin = 'H') THEN
               reg := tempdatain;
            ELSE
               tempreg := reg;
               gen_rot(tempdatain, tempreg, tempc, temps, reg);
               cout <= tempc;
               sout <= temps;
            END IF;

         WHEN mode3 =>
            -- Inversion de f
            tempreg := reg;
            reg := division(cunnor,tempreg);
            cout <= reg;

         WHEN OTHERS =>

      END CASE;
   END IF;

END PROCESS;

END behav_rona;


LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE libb.constants.ALL;
   USE libb.type_def.ALL;
   USE libb.fonctions.ALL;

-------------------------------------------------------------------------
--          DECLARATION D'ENTITE POUR PE(2,2), PE(3,3) ET PE(4,4)     --
-------------------------------------------------------------------------

ENTITY ronb IS
      PORT( clk            : IN clk_type;
            rst            : IN rst_type;
            mode           : IN mode_type1;
            initin         : IN init_type ;
            datain         : IN data_type ;
            cout, sout     : OUT data_type ;
            initout        : OUT init_type
            );
END ronb;

-------------------------------------------------------------------------
--             ARCHITECTURE DE PE(2,2), PE(3,3) ET PE(4,4)             --
-------------------------------------------------------------------------
```

```
ARCHITECTURE behav_ronb OF ronb IS

BEGIN

PROCESS
   VARIABLE tempreg, reg, tempc, temps, tempdatain : data_type;

   BEGIN

      WAIT UNTIL (clk'EVENT AND clk = '1');
      IF rst = '1' THEN
         reg := val_init_S;
         tempreg := (OTHERS => '0');

      ELSE
         initout <= initin;
         tempdatain := datain;

         CASE mode IS

            WHEN mode1 =>
               -- Produit de ST par 1/lambda
               tempreg := produit(reg,tempdatain);
               reg := tempreg;
               cout <= (OTHERS => '0');

            WHEN mode2 =>
               -- Produit de ST par phi
               cout <= produit(reg,tempdatain);

            WHEN mode3 =>
               --  Contenu du registre reg sorti a droite
               cout <= reg;

            WHEN mode4 =>
               -- Triangularisation
               IF (initin = '1') OR (initin = 'H') THEN
                  reg := tempdatain;
               ELSE
                  tempreg := reg;
                  gen_rot(tempdatain, tempreg, tempc, temps, reg);
                  cout <= tempc;
                  sout <= temps;
               END IF;

            WHEN mode5 =>
               -- 0 est sorti a droite
               cout <= (OTHERS => '0');

            WHEN OTHERS =>

         END CASE;
      END IF;
END PROCESS;
```

```
END behav_ronb;


LIBRARY IEEE, libb;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_SIGNED.ALL;

   USE libb.constants.ALL;
   USE libb.type_def.ALL;
   USE libb.fonctions.ALL;

--------------------------------------------------------------------
--                  DECLARATION D'ENTITE POUR PE(5,5)              --
--------------------------------------------------------------------

ENTITY ronc IS
      PORT( clk            : IN clk_type;
            rst            : IN rst_type;
            mode           : IN mode_type1;
            initin         : IN init_type ;
            datain         : IN data_type ;
            cout, sout     : OUT data_type ;
            initout        : OUT init_type
            );
END ronc;

--------------------------------------------------------------------
--                  ARCHITECTURE DE  PE(5,5)                       --
--------------------------------------------------------------------

ARCHITECTURE behav_ronc OF ronc IS

BEGIN

PROCESS
   VARIABLE tempreg, reg, tempc, temps, tempdatain : data_type;
   BEGIN

      WAIT UNTIL (clk'EVENT AND clk = '1');
      IF rst = '1' THEN
         reg := val_init_S;
         tempreg := (OTHERS => '0');

      ELSE
         initout <= initin;
         tempdatain := datain;
         CASE mode IS

            WHEN model =>
               -- Triangularisation
               IF (initin = '1') OR (initin = 'H') THEN
                  reg := tempdatain;
                  cout <= beta;

               ELSE
                  tempreg := reg;
```

```
                gen_rot(tempdatain, tempreg, tempc, temps, reg);
                cout <= beta;
             END IF;

          WHEN mode2 =>
             -- 0 est sorti a droite
             cout <= (OTHERS => '0');

          WHEN mode3 =>
             -- Produit de ST par lambda
             tempreg := produit(reg,tempdatain);
             reg := tempreg;
             cout <= (OTHERS => '0');

          WHEN mode4 =>
             -- Produit de ST par phi
             cout <= produit(reg,tempdatain);

          WHEN mode5 =>
             --  Contenu du registre reg sorti a droite
             cout <= reg;

          WHEN mode6 =>
             cout <= beta;

          WHEN OTHERS =>

       END CASE;
    END IF;
END PROCESS;

END behav_ronc;


--------------------------------------------------------------------------
--
-- Fichier : fonc_head.vhd
-- Paquetage de declaration des fonctions et procedures
-- (Tete des fonctions)
--
-- Auteur :
--          Sebastien Lesueur
--          28 Avril 1999
--
--------------------------------------------------------------------------

LIBRARY IEEE, DW02, lib;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_signed.all;

USE DW02.DW02_components.all;
USE lib.type_def.all;


PACKAGE fonctions IS
```

```
   -- Produit de deux nombres normalises par 256
   FUNCTION produit(a,b : SIGNED) RETURN data_type;

   -- Produit de deux nombres normalises par 64
   FUNCTION produit2(a,b : SIGNED) RETURN data_type;

   -- Racine carree
   FUNCTION sq_root(a : IN UNSIGNED) RETURN data_type;

   -- Calcul de l'inverse d'un nombre (24 bits)
   FUNCTION inversion(b : IN SIGNED) RETURN data_type_ext;

   -- Division de deux nombres
   FUNCTION division(a,b : IN SIGNED) RETURN data_type;

   -- Decalage a gauche de 9 bits
   FUNCTION shift9l(a: IN UNSIGNED) RETURN UNSIGNED;

   -- Decalage a gauche de 7 bits
   FUNCTION shift7l(a: IN UNSIGNED) RETURN UNSIGNED;

   -- Decalage a gauche de 2 bits
   FUNCTION shift2l(a: IN SIGNED) RETURN SIGNED;

   -- Decalage a droite de 2 bits
   FUNCTION shift2d(a: IN SIGNED) RETURN SIGNED;

   -- Generation des rotations de Givens
   PROCEDURE gen_rot(ain, r : IN SIGNED;
                     c,s, rout : OUT SIGNED);

   -- Application des rotations de Givens
   PROCEDURE appli_rot(ain, r, cin, sin : IN SIGNED;
                     cout, sout, rout, aout : OUT SIGNED);

   -- Fonction exponentielle (exp(-(v/vs)^2))
   FUNCTION exp(v : IN SIGNED) RETURN data_type;

END fonctions;

------------------------------------------------------------------------
--
-- Fichier : fonc_body.vhd
-- Paquetage de description des fonctions et procedures
-- (Corps des fonctions)
--
-- Auteur :
--          Sebastien Lesueur
--          28 Avril 1999
--
------------------------------------------------------------------------

LIBRARY IEEE, DW02, lib;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_signed.all;
USE DW02.DW02_components.all;
```

```
USE lib.type_def.all;
USE lib.constants.all;

PACKAGE BODY fonctions IS

--------     Produit de deux nombres normalises par 256    -------

FUNCTION produit(a,b : IN SIGNED) RETURN data_type IS
VARIABLE p : long_word;
VARIABLE prod : data_type;
BEGIN
   p := a*b;
   prod(nb_var-1) := p(nb_lword-1);
   prod(nb_var-2 DOWNTO 0) := p(29 DOWNTO 11);
   RETURN prod;
END produit;

--------     Produit de deux nombres normalises par 64     -------

FUNCTION produit2(a,b : IN SIGNED) RETURN data_type IS
VARIABLE p : long_word;
VARIABLE prod : data_type;
BEGIN
   p := a*b;
   prod(nb_var-1) := p(nb_lword-1);
   prod(nb_var-2 DOWNTO 0) := p(31 DOWNTO 13);
   RETURN prod;
END produit2;

--------     Calcul de la racine carree sur [1;2]          -------


FUNCTION sq_root (a: IN UNSIGNED) RETURN data_type IS
VARIABLE p1 : unsigned(39 downto 0);
variable p2 : UNSIGNED(23 DOWNTO 0);
VARIABLE s : unsigned(19 downto 0);
variable sq : data_type;
BEGIN
   p1 := a*unsigned(cunnor);
   p2(23) := p1(nb_lword-1);
   p2(22 downto 0) := p1(22 downto 0);
   -- infer DW02_sqrt
   s(11 downto 0) := sqrt(p2);
   s(19 downto 12) := (OTHERS => '0');
   sq := SIGNED(s);
RETURN sq;
END sq_root;

-------   Inversion d'un nombre, format etendu a 24 bits   -------

FUNCTION inversion(b : IN SIGNED) RETURN data_type_ext IS
VARIABLE binv, binvint, binvint2, b24, prod, p3, p3int : data_type_ext;
VARIABLE p : SIGNED(47 DOWNTO 0);
BEGIN
   -- infer DW02_divide
   b24 := (OTHERS => '0');
   b24(19 DOWNTO 0) := ABS(b);
```

```
   binvint := cun24 / ABS(b24);
   p := binvint*b24;
   prod(23) := p(47);
   prod(22 DOWNTO 0) := p(29 DOWNTO 7);

   p3int := cun24nor - prod;
   p3 := (others => '0');
   p3(23) := p3int(23);
   p3(22 downto 8) := p3int(14 downto 0);

   if p3 > b24 then
      binvint2 := binvint + lsb;
   else
      binvint2 := binvint;
   end if;
   if b24(23) = '1' then
      binv := -binvint2;
   else
      binv := binvint2;
   end if;
RETURN binv;
END inversion;

--------      Fonction de division de deux nombres        ---------


FUNCTION division(a,b : IN SIGNED) RETURN data_type IS

VARIABLE quotint, quot : data_type;
VARIABLE p : SIGNED(47 DOWNTO 0);
VARIABLE binv24,a24 : data_type_ext;
      BEGIN
         binv24 := inversion(ABS(b));
         a24 := (OTHERS => '0');
         a24(23 DOWNTO 4) := ABS(a);
         p := a24 * binv24;
         quotint(19) := p(47);
         quotint(18 DOWNTO 0) := p(33 DOWNTO 15);
         IF a(19) = b(19) THEN
            quot := quotint;
         ELSE
            quot := -quotint;
         END IF;
      RETURN quot;
END division;

---------          Decalage a gauche de 9 bits          ---------


FUNCTION shift9l(a : IN UNSIGNED) RETURN UNSIGNED IS
   variable ashift : unsigned(19 downto 0);
   BEGIN
         ashift(18 DOWNTO 0) := (OTHERS => '0');
         ashift(19) := a(19);
         ashift(18 DOWNTO 9) := a(9 DOWNTO 0);
   RETURN ashift;
```

```
END shift9l;

---------          Decalage a gauche de 7 bits          ---------

FUNCTION shift7l(a : IN UNSIGNED) RETURN UNSIGNED IS
   variable ashift2 : UNSIGNED(19 DOWNTO 0);
   BEGIN
         ashift2(18 DOWNTO 0) := (OTHERS => '0');
         ashift2(19) := a(19);
         ashift2(18 DOWNTO 7) := a(11 DOWNTO 0);
   RETURN ashift2;
END shift7l;

---------          Decalage a gauche de 2 bits          ---------

FUNCTION shift2l(a : IN SIGNED) RETURN SIGNED IS
   variable ashift : data_type;
   BEGIN
         ashift(18 DOWNTO 0) := (OTHERS => '0');
         ashift(19) := a(19);
         ashift(18 DOWNTO 2) := a(16 DOWNTO 0);
   RETURN ashift;
END shift2l;

---------          Decalage a droite de 2 bits          ---------

FUNCTION shift2d(a : IN SIGNED) RETURN SIGNED IS
   variable ashift : data_type;
   BEGIN
         ashift(18 DOWNTO 17) := (OTHERS => a(19));
         ashift(19) := a(19);
         ashift(16 DOWNTO 0) := a(18 DOWNTO 2);
   RETURN ashift;
END shift2d;

----------   Procedure de generation des rotations de Givens   ---------

PROCEDURE gen_rot(ain, r: IN SIGNED;
                  c,s, rout : OUT SIGNED) IS
VARIABLE absa, absr, t, tempc, temps : data_type;
   BEGIN
      absa := ABS(ain);
      absr := ABS(r);

      IF ain = czero THEN
            tempc := lambd2;
            c := tempc;
            temps := czero;
            s := temps;
      ELSE
         IF absa >= absr THEN
               t := division(r,ain);
               temps :=
division(cunnor,sq_root(cunnor_uns+UNSIGNED(produit(t,t))));
               tempc := produit(temps,t);
               c := tempc;
               s := temps;
```

```
            ELSE
               t := division(ain,r);
               tempc :=
division(cunnor,sq_root(cunnor_uns+UNSIGNED(produit(t,t))));
               temps := produit(tempc,t);
               c := tempc;
               s := temps;
            END IF;
      END IF;
   rout := produit(tempc,r)+produit(temps,ain);
END gen_rot;

--------   Procedure de d'application des rotations de Givens   --------

PROCEDURE appli_rot(ain, r, cin, sin : IN SIGNED;
                    cout, sout, rout, aout : OUT SIGNED) IS
BEGIN

   aout:=produit(-sin,r)+produit(cin,ain);
   rout:=produit(cin,r)+produit(sin,ain);
   cout:=cin;
   sout:=sin;

END appli_rot;

------     Calcul de la fonction exponentielle: exp(-(v/vs)^2)   -------

FUNCTION exp(v : IN SIGNED) RETURN data_type IS
VARIABLE x, r, expv : data_type;
BEGIN
   r:=ABS(v);
         x:= produit2(r,c5);
         IF r < x1 THEN
            expv :=
produit2(b11,produit2(x,produit2(x,x)))+produit2(b31,x)+b41;
         ELSIF r>= x1 AND r< x2 THEN
            expv :=
produit2(b12,produit2(x,produit2(x,x)))+produit2(b22,produit2(x,x))+prod
uit2(b32,x)+b42;
         ELSIF r>= x2 AND r< x3 THEN
            expv :=
produit2(b13,produit2(x,produit2(x,x)))+produit2(b23,produit2(x,x))+prod
uit2(b33,x)+b43;
         ELSIF r>= x3 AND r< x4 THEN
            expv :=
produit2(b14,produit2(x,produit2(x,x)))+produit2(b24,produit2(x,x))+prod
uit2(b34,x)+b44;
         ELSIF r>= x4 AND r<= x5 THEN
            expv :=
produit2(b15,produit2(x,produit2(x,x)))+produit2(b25,produit2(x,x))+prod
uit2(b35,x)+b45;
         ELSE
            expv := czero;
         END IF;
   RETURN expv;
END exp;
```

```
END fonctions;


------------------------------------------------------------------------
--
-- Fichier : type_def.vhd
--       Paquetage de definition des types et sous-types utilises
--
-- Par :    Sebastien Lesueur
--          18 Avril 1999
--
------------------------------------------------------------------------

LIBRARY IEEE, lib;
 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;

 USE lib.constants.ALL;

PACKAGE type_def IS

   -- Type des donnees sur 16 bits
   SUBTYPE data_type IS SIGNED( nb_var-1 DOWNTO 0 );
   -- Type des donnees sur 32 bits
   SUBTYPE long_word IS SIGNED( nb_lword-1 DOWNTO 0 );
   -- Type des donnees sur 24 bits
   SUBTYPE data_type_ext IS SIGNED( nb_var_ext-1 DOWNTO 0 );
   -- Type de l'horloge
   SUBTYPE clk_type IS STD_LOGIC ;
   -- Type du Reset
   SUBTYPE rst_type IS STD_LOGIC ;
   -- Type des signaux d'initialisation des PE
   SUBTYPE init_type IS STD_LOGIC;

   -- Type de mode a 3 bits
   SUBTYPE mode_type1 IS STD_LOGIC_VECTOR (2 DOWNTO 0);
   -- Type de mope a 4 bits
   SUBTYPE mode_type2 IS STD_LOGIC_VECTOR(3 DOWNTO 0);

   -- Vecteur de 4 donnees de type data_type
   TYPE data_type4 IS ARRAY(1 TO 4) OF data_type;
   -- Vecteur de 3 donnees de type data_type
   TYPE data_type3 IS ARRAY(1 TO 3) OF data_type;
   -- Vecteur de 2 donnees de type data_type
   TYPE data_type2 IS ARRAY(1 TO 2) OF data_type;

   -- Vecteur de 4 signaux d'initialisation
   TYPE init_type4 IS ARRAY(1 TO 4) OF init_type;
   -- Vecteur de 3 signaux d'initialisation
   TYPE init_type3 IS ARRAY(1 TO 3) OF init_type;
   -- Vecteur de 2 signaux d'initialisation
   TYPE init_type2 IS ARRAY(1 TO 2) OF init_type;

   -- Type de l'etat pour la machine a etat
   SUBTYPE state_type IS STD_LOGIC_VECTOR(4 DOWNTO 0);
```

```
   -- Type pour l'adresse de sortie
   SUBTYPE addr_type IS STD_LOGIC_VECTOR(2 DOWNTO 0);

END type_def;

--------------------------------------------------------------------------
--
-- Fichier : constants.vhd
-- Paquetage de definition des constantes
--
--
-- Auteur :
--          Sebastien Lesueur
--          28 Avril 1999
--------------------------------------------------------------------------

LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;

PACKAGE constants IS

-- NOMBRES DE BITS POUR LES DIFFERENTS FORMATS DE DONNEES

   -- Type de donnee sur 20 bits
   CONSTANT nb_var : integer := 20;
   -- Type de donnee sur 24 bits
   CONSTANT nb_var_ext : integer := 24;
   -- Type de donnee sur 40 bits
   CONSTANT nb_1word : integer := 40;

-----------------------------------------------------------------------
--          CONSTANTES POUR LES CALCULS INTERNES DES PE              --
-----------------------------------------------------------------------


CONSTANT czero : SIGNED(nb_var-1 DOWNTO 0) := "00000000000000000000";
CONSTANT cundemi : UNSIGNED(nb_var-1 DOWNTO 0):= "01000000000000000000";
-- 1 pour la division
CONSTANT c1 : SIGNED(nb_var-1 DOWNTO 0) := "00111111111111111111";
CONSTANT cun : SIGNED(nb_var-1 DOWNTO 0) := "01111111111111111111";
-- 1 sur 24 bits pour division
CONSTANT cun24:SIGNED(nb_var_ext-1 DOWNTO 0):=
"001111111111111111111111";
-- 1 normalise par 256 sur 20 bits
CONSTANT cunnor : SIGNED(nb_var-1 DOWNTO 0) := "00000000100000000000";
-- 1 normalise sur 24 bits
CONSTANT cunnor64 : SIGNED(nb_var-1 DOWNTO 0) := "00000010000000000000";
CONSTANT cun24nor : SIGNED(nb_var_ext-1 DOWNTO 0):=
"000000001000000000000000";
-- Valeur initiale des coefficients diagonaux de S
CONSTANT val_init_S : SIGNED(nb_var-1 DOWNTO 0) :=
"00000101000000000000";

-- Valeur normalisee du rapport des covariances
CONSTANT beta : SIGNED(nb_var-1 DOWNTO 0) := "00000000011100110011";
```

```
-- Valeur normalisee du seuil d'innovation
CONSTANT epse : SIGNED(nb_var-1 DOWNTO 0) := "00000000000001110000";
-- Valeur 1/0.99 pour 1/lambda
CONSTANT lambd1 : SIGNED(nb_var-1 DOWNTO 0) :="00000001000001000000";
-- Valeur 1/1 pour 1/lambda
CONSTANT lambd2 : SIGNED(nb_var-1 DOWNTO 0) :="00000001000000000000";
-- 1 LSB sur 24 bits
CONSTANT lsb : SIGNED(nb_var_ext-1 DOWNTO 0) :=
"000000000000000000000001";          CONSTANT cunnor_uns :
-- 1 normalise non signe sur 20 bits
UNSIGNED(nb_var-1 DOWNTO 0) := "00000001000000000000";
-- Gain Kp1r divise par 4
CONSTANT Kp1r : SIGNED(nb_var-1 DOWNTO 0) := "01001000000000000000";
-- Gain Kv1r
CONSTANT Kv1r : SIGNED(nb_var-1 DOWNTO 0) := "00001001100110011101";
-- Gain Kp1r divise par 4
CONSTANT Kpmr : SIGNED(nb_var-1 DOWNTO 0) := "01011011001000000000";
-- Gain Kv1r
CONSTANT Kvmr : SIGNED(nb_var-1 DOWNTO 0) := "00000011001100110011";
-- 1/k (coefficient de torsion) normalise
CONSTANT kinv : SIGNED(nb_var-1 DOWNTO 0) := "00001000000000000000";

-----------------------------------------------------------------
--             MODES DES PROCESSEURS ELEMENTAIRES              --
-----------------------------------------------------------------

   CONSTANT mode0 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
   CONSTANT mode1 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "001";
   CONSTANT mode2 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "010";
   CONSTANT mode3 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011";
   CONSTANT mode4 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "100";
   CONSTANT mode5 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "101";
   CONSTANT mode6 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "110";
   CONSTANT mode7 : STD_LOGIC_VECTOR(2 DOWNTO 0) := "111";
   CONSTANT mode0b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
   CONSTANT mode1b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0001";
   CONSTANT mode2b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0010";
   CONSTANT mode3b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0011";
   CONSTANT mode4b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0100";
   CONSTANT mode5b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0101";
   CONSTANT mode6b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0110";
   CONSTANT mode7b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0111";
   CONSTANT mode8b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1000";
   CONSTANT mode9b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1001";
   CONSTANT mode10b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1010";
   CONSTANT mode11b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1011";
   CONSTANT mode12b : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1100";

-- ----------------------------------------------------------------
--             ETATS DES PROCESSEURS ELEMENTAIRES                 --
-----------------------------------------------------------------

   CONSTANT state00 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00000";
   CONSTANT state01 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00001";
   CONSTANT state02 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00010";
   CONSTANT state03 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00011";
   CONSTANT state04 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00100";
```

```
    CONSTANT state05 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00101";
    CONSTANT state1 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00110";
    CONSTANT state2 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00111";
    CONSTANT state3 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01000";
    CONSTANT state4 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01001";
    CONSTANT state5 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01010";
    CONSTANT state6 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01011";
    CONSTANT state7 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01100";
    CONSTANT state8 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01101";
    CONSTANT state9 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01110";
    CONSTANT state10 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "01111";
    CONSTANT state11 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "10000";
    CONSTANT state12 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "10001";
    CONSTANT state13 : STD_LOGIC_VECTOR(4 DOWNTO 0) := "10010";

-------------------------------------------------------------------
--       CONSTANTES POUR L'APPROXIMATION DE LA FONCTION EXP       --
-------------------------------------------------------------------
CONSTANT b11 : SIGNED(nb_var-1 DOWNTO 0) := "1110010000100010110";
CONSTANT b31 : SIGNED(nb_var-1 DOWNTO 0) := "1111111010111111010";
CONSTANT b41 : SIGNED(nb_var-1 DOWNTO 0) := "0000010000000000000";
CONSTANT b12 : SIGNED(nb_var-1 DOWNTO 0) := "0001010000100100101";
CONSTANT b22 : SIGNED(nb_var-1 DOWNTO 0) := "1110111010110101011";
CONSTANT b32 : SIGNED(nb_var-1 DOWNTO 0) := "0000001011010001101";
CONSTANT b42 : SIGNED(nb_var-1 DOWNTO 0) := "0000001111010110101";
CONSTANT b13 : SIGNED(nb_var-1 DOWNTO 0) := "0000100100010000010";
CONSTANT b23 : SIGNED(nb_var-1 DOWNTO 0) := "1111011010010010111";
CONSTANT b33 : SIGNED(nb_var-1 DOWNTO 0) := "1111111100001101001";
CONSTANT b43 : SIGNED(nb_var-1 DOWNTO 0) := "0000010000100011000";
CONSTANT b14 : SIGNED(nb_var-1 DOWNTO 0) := "0000000000011010101";
CONSTANT b24 : SIGNED(nb_var-1 DOWNTO 0) := "0000011101000001001";
CONSTANT b34 : SIGNED(nb_var-1 DOWNTO 0) := "1111100101101010010";
CONSTANT b44 : SIGNED(nb_var-1 DOWNTO 0) := "0000011000010001100";
CONSTANT b15 : SIGNED(nb_var-1 DOWNTO 0) := "1111101011000001011";
CONSTANT b25 : SIGNED(nb_var-1 DOWNTO 0) := "0000111011101001001";
CONSTANT b35 : SIGNED(nb_var-1 DOWNTO 0) := "1111000101011010111";
CONSTANT b45 : SIGNED(nb_var-1 DOWNTO 0) := "0000010011110011000";
CONSTANT c5 : SIGNED(nb_var-1 DOWNTO 0) := "0000101000000000000";
CONSTANT x1 : SIGNED(nb_var-1 DOWNTO 0) := "0000000000011000011";
CONSTANT x2 : SIGNED(nb_var-1 DOWNTO 0) := "0000000000110000101";
CONSTANT x3 : SIGNED(nb_var-1 DOWNTO 0) := "0000000001100001010";
CONSTANT x4 : SIGNED(nb_var-1 DOWNTO 0) := "0000000010010001111";
CONSTANT x5 : SIGNED(nb_var-1 DOWNTO 0) := "0000000011000010100";

-------------------------------------------------------------------
--       CONSTANTES POUR LA SATURATION DES PARAMETRES ESTIMES     --
-------------------------------------------------------------------
CONSTANT Jinf : SIGNED(nb_var-1 DOWNTO 0) := "0000000000000000100";
CONSTANT Jsup : SIGNED(nb_var-1 DOWNTO 0) := "0000000000110011010";
CONSTANT Fcinf : SIGNED(nb_var-1 DOWNTO 0) := "0000000000000101111";
CONSTANT Fcsup : SIGNED(nb_var-1 DOWNTO 0) := "0000000000001000111";
CONSTANT Fvinf : SIGNED(nb_var-1 DOWNTO 0) := "0000000000000011111";
CONSTANT Fvsup : SIGNED(nb_var-1 DOWNTO 0) := "0000000000000101111";
CONSTANT Fsinf : SIGNED(nb_var-1 DOWNTO 0) := "0000000000000101001";
CONSTANT Fssup : SIGNED(nb_var-1 DOWNTO 0) := "0000000000000111101";

END constants;
```