Titre: Title:	Conception d'un système de post-traitement vidéo pour un corrélateur optique
Auteur: Author:	Pascal Poiré
Date:	1998
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Poiré, P. (1998). Conception d'un système de post-traitement vidéo pour un corrélateur optique [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8552/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8552/
Directeurs de recherche: Advisors:	Yvon Savaria
Programme: Program:	Unspecified

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UN SYSTÈME DE POST-TRAITEMENT VIDÉO POUR UN CORRÉLATEUR OPTIQUE

PASCAL POIRÉ

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M.SC.A)

(GÉNIE ÉLECTRIQUE)

OCTOBRE 1998

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION D'UN SYSTÈME DE POST-TRAITEMENT VIDÉO POUR UN CORRÉLATEUR OPTIQUE

présenté par: POIRÉ Pascal

en vue de l'obtention du diplôme de: Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de:

M. BOIS Guy, Ph.D., président

M. SAVARIA Yvon, Ph.D., membre et directeur de recherche

M. BERGERON Alain, Ph.D., membre et codirecteur de recherche

M. SAWAN Mohamad, Ph.D., membre

REMERCIEMENTS

J'aimerais remercier mon directeur de recherche, M. Yvon Savaria, pour m'avoir guidé tout au long de mes travaux de recherche et d'avoir pris le temps de me conseiller lors de nos rencontres. Cette synergie a conduit ma maîtrise à travers les méandres du monde de la recherche et c'est en observant les résultats de mes travaux que je réalise maintenant que je suis à bon port.

Je remercie également l'Institut national d'optique de Québec et plus particulièrement mon codirecteur de recherche M. Alain Bergeron de m'avoir incité à entreprendre des études supérieures en me proposant le sujet de recherche de maîtrise du présent mémoire. Il a su répondre à mes nombreuses questions sur le complexe domaine des corrélateurs optiques.

Je remercie aussi le fonds FCAR du gouvernement du Québec pour son important soutien financier sans lequel je n'aurais jamais pu entreprendre ces études. C'est avec fierté que je remercie mes parents de m'avoir constamment épaulé lors de mes études et je tiens a souligner, de plus, leur générosité sans limites. Je remercie aussi ma conjointe Mylène pour sa patience, sa compréhension et son aide précieuse dans la correction de mes textes. Finalement, je remercie tous mes amis du groupe de recherche en microélectronique avec qui j'ai eu beaucoup de plaisir durant ces années.

RÉSUMÉ

Un corrélateur optique produit énormément d'informations en temps réel. La principale difficulté vient de l'analyse du résultat de la corrélation, soit de sortir les données du corrélateur optique. Les plans de corrélation (images), principalement composés de pics lumineux, doivent être analysés à la même vitesse que le traitement du corrélateur afin d'éviter une accumulation des données du post-traitement. Le traitement doit se faire à un débit de 30 images par seconde. De plus, le système doit être capable de décider de la présence ou de l'absence de l'objet recherché dans l'image d'entrée en temps réel.

Ce mémoire propose un algorithme de post-traitement des informations provenant d'un corrélateur optique basé sur la comparaison de mesures de pics de corrélation afin que la prise de décision soit la plus efficace possible. De plus, trois mises en oeuvre sont présentées : une en langage C, une en assembleur et une logicielle/matérielle. Une analyse des performances de ces mises en oeuvre nous a permis de conclure que même un processeur DSP, le TMS320C40, programmé en assembleur est incapable à lui seul d'effectuer le post-traitement d'un corrélateur optique. Une partition de l'algorithme nous a permis de concevoir une mise en oeuvre logicielle/matérielle dont la partie logicielle est exécutée sur le TMS320C40 et la partie matérielle réalisée sur deux FPGA. Cette version codesign, qui économise temps et argent, exécute l'algorithme deux fois plus vite que le corrélateur peut produire des données.

ABSTRACT

Optical correlators produce a large amount of informations in real time. The main problem is caused by the post-processing of the correlation's result. The correlation planes (images), mainly composed of light spots must be processed at the same speed the optical correlator produces new frames to avoid the accumulation of post-processing data. The post-processing must be designed to support a frame rate of 30 per seconds. Furthermore, this system must take the decisions in real time regarding the presence or absence of objects of interest in the image.

This thesis proposes an algorithm for the post-processing of an optical correlator based on the comparison of correlation peak's measure to fixed parameters in order to reach the best effective decision. Three implementations are exposed : one in C language, one in assembly and one hardware/software. A performance analysis of those three realizations allowed us to conclude that even using assembly language, a DSP processor, the TMS320C40 alone cannot accomplish in real time the post-processing of an optical correlator output. The partition of this algorithm enables us to realize an hardware/software implementation where the software part runs on a TMS320C40 and the hardware part was synthesized on two FPGAs. This codesign version did not require very long development efforts and it executes our algorithm two times faster than the optical correlator can produce new data.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION

De tous les systèmes de vision existants, c'est celui des êtres vivants qui est sans aucun doute le plus développé. Le but fondamental de la vision est de produire, à partir d'images, une description de la forme et de la position des objets d'une scène. Pour cela, notre système se base sur plusieurs indices visuels comme les arrêtes, les textures, les ombrages, la stéréoscopie, le mouvement,... C'est la performance de la paire cerveau-oeil humain que les chercheurs ont tant essayé d'approcher avec des sytèmes synthétiques afin de reproduire la vision. De toutes les méthodes que l'on peut retrouver dans la littérature, la plus utilisée est celle de la reconnaissance de formes à partir d'arêtes (Hildreth, 1985; Canny, 1986; Law et al., 1996). Une arrête est un changement brusque d'intensité dans une image, qui peut être causée par la limite entre deux objets différents d'une scène. Cette technique demande énormément de temps de calcul. Dans certains cas, la détection d'arêtes doit être précédée de plusieurs convolutions (filtrage du bruit, amélioration de contraste,...). La vision artificielle exige ensuite d'assembler les arêtes pour finalement tenter une reconnaissance de la structure. Il est évident que ce type d'algorithme peut essoufler la plupart des processeurs que l'on peut retrouver sur le marché.

La corrélation est une technique qui permet de mesurer le taux de similitude existant entre deux signaux. C'est grâce aux récents développements de plusieurs nouvelles technologies que les corrélateurs optiques peuvent ainsi détecter la présence d'un objet à rechercher dans une image d'entrée et résoudre ce problème de façon presque instantannée. La

reconnaissance d'objets par les corrélateurs optiques repose sur un jeu de transformée de Fourier optique en deux dimensions combinée à l'introduction d'un filtre dans le domaine des fréquences spatiales qui représente l'objet à reconnaître. À la sortie d'un tel traitement, le plan de corrélation comporte un pic intense et étroit à l'emplacement de l'objet lorsqu'il est présent en entrée.

Un corrélateur optique produit énormément d'informations en temps réel. La principale difficulté vient de l'analyse du résultat de la corrélation afin de produire le résultat désiré. Les plans de corrélation (image), principalement composés de pics lumineux, doivent être analysés à la même vitesse que le traitement du corrélateur, afin d'éviter une accumulation des résultats intermédiaires au post-traitement. Dans le cas qui nous préoccupe, le traitement doit se faire à un débit de 30 images par seconde. De plus, le système doit être capable de décider de la présence ou de l'absence de l'objet recherché dans l'image d'entrée. La proposition d'un algorithme basé sur la comparaison de mesures à des paramètres devra être réalisée afin que la prise de décision soit la plus efficace possible.

Le développement récent des circuits intégrés reprogrammables (FPGA) a ouvert de nouveaux horizons pour l'intégration d'applications. L'augmentation de leur complexité et de leur vitesse, couplée à une réduction des coûts, rend possible leur utilisation dans des systèmes mixtes où les circuits reprogrammables sont des coprocesseurs dédiés à un microprocesseur. Ces plates-formes reconfigurables permettent une partition logicielle/ matérielle d'une application. Ceci permet une utillisation du parallélisme qui résulte en une

accélération de l'application. Des chercheurs (Albaharna et al., 1994) ont investigué les gains en accélération d'applications exécutées sur des plates-formes reconfigurables. Avec les plus récents FPGA et microprocesseurs, la plupart des facteurs d'accélération sont entre 10 et 100. De plus, les auteurs ont mis en évidence l'avantage d'utiliser du matériel dédié réalisé à l'aide d'un FPGA plutôt que d'ajouter un microprocesseur additionnel : l'ajout d'un microprocesseur additionnel ne pourra qu'offrir une accélération maximale de 2. Par la création de fonctions spécifiques, la partie matérielle dédiée peut offrir un éventail de facteurs d'accélération en fonction du pourcentage de l'application ainsi accélérée et de l'accélération réalisée sur cette partie.

Un circuit intégré dédié supportant la partie matérielle d'une application partitionnée offre une plus grande accélération qu'un circuit reprogrammable. Par contre, lorsque le volume n'y est pas, chaque circuit intégré peut coûter très cher. Les frais non-récurents associés à la conception et à la production d'un circuit intégré doivent être réparti sur le nombre d'exemplaires utilisés. De plus, un circuit intégré offre une solution figée. Quant à lui, un circuit reprogrammable offre une très grande flexibilité par la reconfiguration possible de ses blocs logiques. Ceci est un avantage majeur lorsqu'un algorithme est sujet à évoluer dans le temps et que l'utilisateur peut conserver la même plate-forme et le même environnement, réduisant le temps d'apprentissage. Ainsi, un changement dans l'algorithme n'entraînera pas la création d'un nouveau circuit imprimé pour un nouveau circuit intégré. De plus, l'écart entre la performance des circuits reprogrammables d'aujourd'hui et les circuits intégrés est de plus en plus petit dans certaines classes

d'applications. La famille XC4000XL-08 du manufacturier Xilinx offre des FPGA possédant une complexité allant jusqu'à 180 000 portes logiques et ils peuvent supporter une horloge système de 100 MHz.

Lorsque l'on considère d'intégrer une solution matérielle pour répondre aux besoins d'une application, une des premières questions à se poser est : est-ce qu'un microprocesseur général ou un processeur de traitement des signaux (DSP) peut répondre à la demande? Cette avenue est la moins coûteuse en ce qui a trait aux coûts de développement et elle est la plus flexible de toutes. L'exploration de cette avenue conduit à son tour à d'autres options. En premier lieu, l'application peut être codée dans un langage de haut niveau comme par exemple le langage C, qui sera compilé dans le langage machine du processeur choisi. Une autre option, plus longue à réaliser, consiste à coder directement dans le langage assembleur du processeur. Ce temps additionnel de conception résulte parfois en un gain de performance considérable par rapport au compilateur de langage haut niveau. Lorsqu'une solution entièrement logicielle ne peut offrir les performances souhaitées, il est de mise d'envisager une solution mixte logicielle/matérielle (codesign) avant de se lancer dans la conception d'un circuit intégré dédié.

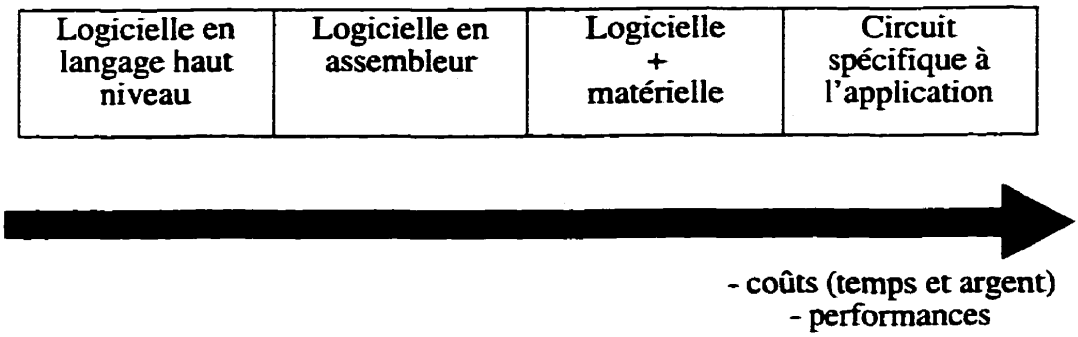

Figure I.1 Mise en oeuvre d'une application

Le premier chapitre de ce mémoire consiste en une revue de littérature. Cette revue de littérature à pour but de donner au lecteur toutes les connaissances générales nécessaires à la compréhension du mémoire. Il débute par la théorie mathématique sur la transformée de Fourier et sur la corrélation. Ensuite, il décrit le fonctionnement du corrélateur optique de Vander Lugt et de certaines composantes de ce dernier. De plus, nous traiterons de quelques systèmes hybrides (optique et digital) que l'on peut retrouver dans la littérature. Le chapitre se termine par une comparaison entre les filtres optiques adaptés et de phase et par une discussion de certaines mesures de performance des corrélateurs optiques.

Le deuxième chapitre montre les résultats obtenus par l'application de différentes mesures d'évaluation de pic de corrélation sur une banque d'image. Ces mesures d'évaluation servent à discerner les vrais pics de corrélation des faux. Cette banque représente un ensemble de 16 plans de corrélation obtenus du corrélateur de l'Institut National d'Optique

(INO) à Québec, avec qui nous collaborons dans la réalisation de ces travaux. Finalement, ce chapitre présente l'algorithme de post-traitement retenu à la lumière de ces résultats.

Le troisième chapitre présente trois différentes mises en oeuvre de l'algorithme de post-traitement. Une plate-forme composée d'un processeur DSP et de FPGA sert de support à ces implantations. Deux versions logicielles de l'algorithme, une en langage C et une en assembleur, furent réalisées et implantées sur le DSP. Une dernière mise en oeuvre logicielle/matérielle qui utilise le DSP et les FPGA en parallèle y est présentée en détail.

CHAPITRE 1

REVUE DE LITTÉRATURE

Ce chapitre regroupe les éléments essentiels pour la compréhension générale des corrélateurs optiques. Les premières sections donnent au lecteur des notions théoriques sur la transformée de Fourier 2D, la corrélation de signaux et les lentilles sphériques convergentes. Les dernières sections, plus appliquées, décrivent les corrélateurs optiques, comparent certains filtres utilisés, expliquent le fonctionnement des modulateurs spatiaux de lumière et décrient certaines mesures de performance des corrélateurs optiques.

1.1 La transformée de Fourier en 2 dimensions

La transformée de Fourier (T.F.) est un outil essentiel dans le domaine du traitement des signaux. Elle permet d'analyser le contenu fréquentiel de fonctions évoluant dans le temps. La T.F. pour des fonctions à une dimension est obtenue avec cette relation (1.1), où F(ω) est la transformée de Fourier de f(t).

$$F(\ ) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(\ )e^{-j\omega t}dt \qquad (1.1)$$

La transformée de Fourier peut s'appliquer aussi sur des fonctions à deux dimensions, par exemple des images. Dans ce cas, la T.F. ne donnera pas des fréquences temporelles mais

des fréquences spatiales. La transformée de Fourier S(u,v) d'un objet s(x,y) est donnée par cette équation.

$$S(\quad) = \frac{1}{(\quad)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} s(\quad)e^{j(\quad)}d\ dy \tag{1.2}$$

Une propriété intéressante de la T.F. est son invariance en amplitude pour des translations de l'objet à l'entrée. Les translations de l'objet à l'entrée se répercutent en changements de phase sur la T.F. de l'objet. Cette variation de phase s'exprime ainsi :

$$s(\quad) \leftrightarrow S(\quad)e^{-j2\pi(\quad)}. \tag{1.3}$$

La propriété de mise à l'échelle peut être utile afin de comprendre les conséquences d'un changement de l'échelle de l'image d'entrée sur la T.F. de celle-ci. Un agrandissement de l'image d'entrée donnera un patron de T.F. plus petit. À l'inverse, une version plus petite de l'image donnera un plus grand patron de la T.F.. Cette propriété se formule ainsi :

$$s(\quad) \leftrightarrow \frac{1}{|a_x a_y|}S\left(\quad\right). \tag{1.4}$$

Une dernière propriété de la T.F. qu'il serait bon de mentionner est celle dite de rotation. Elle stipule que si l'image d'entrée subit une rotation de θ, alors la T.F. de cette image sera tournée de θ.

Les propriétés de la T.F. 2-D n'ont pas toutes été énumérées (Braham et Horner, 1994), mais ces dernières peuvent donner au lecteur une idée très générale de la relation qui existe entre une fonction 2-D et sa T.F..

1.2 La corrélation

La corrélation est une technique fortement utilisée en reconnaissance de forme, car elle permet de mesurer le degré de similitude existant entre deux signaux. Par exemple, pour les signaux s(x,y) qui représentent l'image d'entrée si h(x,y) est une image de référence qui ne contient que l'objet à retrouver dans s(x,y), la corrélation se définit ainsi :

$$c(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} s(x', y')h(x' - x, y' - y)dx'dy' . \qquad (1.5)$$

Le point possédant la valeur maximale (pic de corrélation) dans le résultat de la corrélation c(x,y) correspond normalement à la coordonnée de l'objet qui était à retrouver dans l'image d'entrée s(x,y). Supposons que S(u,v), H(u,v) et C(u,v) sont respectivement les T.F. de s(x,y),h(x,y) et c(x,y), le théorème de la corrélation (1.6) démontre la relation qui existe entre S(u,v), H(u,v) et C(u,v) (Braham et Horner, 1994).

$$C(u, v) = H^*(u, v)S(u, v) \qquad (1.6)$$

Pour obtenir c(x,y), il suffit d'effectuer la transformée de Fourier inverse ($T.F.^{-1}$) de C(u,v).

1.3 Les lentilles sphériques convergentes

Les lentilles sphériques convergentes produisent à leur foyer arrière la T.F. d'une image située au foyer avant de la lentille (Mauldin, 1988). L'image d'entrée est présentée sous la forme d'un transparent illuminé par un faisceau uniforme de lumière cohérente. Cet arrangement peut être visualisé à la figure 1.1. Ce calcul optique parallèle (T.F.) est réalisé à la vitesse de la lumière, soit à une vitesse de $3 \cdot 10^8$ m/s. Pour une image de 256 par 256 pixels, ce traitement dans un système optique d'environ 30 cm de longueur représenterait un débit d'environ $4 \cdot 10^{15}$ opérations/s pour un ordinateur. Cet estimé (Flannery et Horner, 1989) est basé sur le nombre de multiplications et d'additions nécessaires pour cette transformée en utilisant l'algorithme de la tranformée de Fourier rapide (FFT).

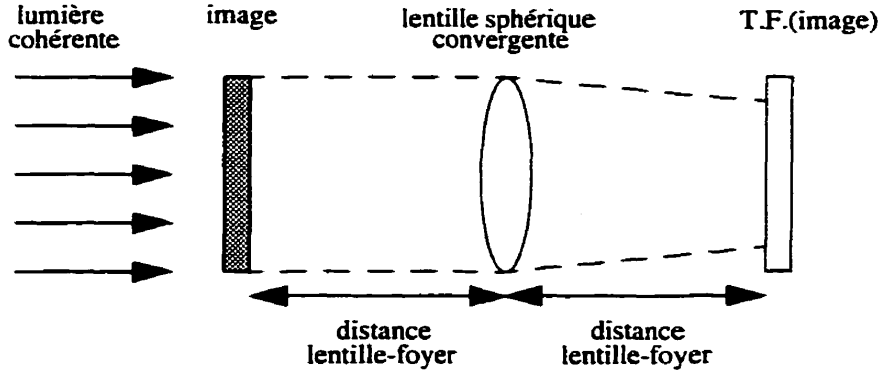

Figure 1.1 La T.F. d'une image par une lentille sphérique convergente

1.4 Description du corrélateur optique

Le traitement des signaux optique dans le plan de Fourier est certainement une des plus anciennes technique de traitement de signaux optiques. Ses racines remontent aux travaux (1893) d'un industriel allemand du nom de Ernst Abbe qui perfectionna les lentilles optiques et le microscope. Le traitement des signaux optiques dans le plan de Fourier est une modification intentionnelle du spectre spatial d'une image dans le but de modifier cette dernière (restauration d'images) ou d'y retrouver un objet que l'on recherche (Flannery et Horner, 1989). Un corrélateur optique effectue de la reconnaissance de formes. La reconnaissance de formes n'est pas un problème entièrement résolu, tant par un traitement optique que par un traitement digital. Mais avec la venue de nouveaux algorithmes et avec la constante amélioration de certains constituants, tout porte à croire qu'un système hybride optique/digital serait une solution qui profiterait du meilleur des deux mondes, afin d'obtenir un progrès significatif dans la résolution de ce problème.

Un système de reconnaissance d'objets incluant un corrélateur optique et une partie de traitement digital a été réalisé pour tirer profit du traitement parallèle et instantanné des corrélateurs optiques et des techniques digitales pour l'affichage et l'enregistrement des images et des filtres (Scholl, 1995). Ce système effectue la reconnaissance en 3 grandes étapes : la détection, la classification et l'identification de l'objet. La détection informe de la présence d'un objet dans la scène, la classification associe l'objet à une classe d'objets similaires et l'identification précise exactement le type de l'objet. Premièrement, la

caractérisation et la suppresion de l'arrière plan dans l'image d'entrée est réalisée par des filtres de convolution. Cette étape améliore les performances du corrélateur optique. Ensuite, l'orientation et la mise à l'échelle de l'objet sont estimées. Cette étape est importante car le corrélateur optique ne tolère que des différences d'environ 5% sur l'orientation et la grosseur de l'objet. Un post-traitement sur le plan de corrélation est appliqué afin de rehausser les pics de corrélation par un filtre de convolution. De plus, cette étape supprime le bruit en arrière plan. Une architecture très semblable à cette dernière fut proposée pour un système robotique d'exploration extra-terrestre (Scholl et Eberlein, 1993). Ce système analyse de l'imagerie visible et multibande, afin d'extraire la composition minérale et l'information sur la texture pour la caractérisation de sites géologiques.

En 1964, A. Vander Lugt a proposé un corrélateur utilisant un masque dans le plan de Fourier (Vander Lugt, 1964). Ce masque (filtre) représente la fonction complexe, amplitude et phase, d'un objet à retrouver. La percée réalisée par Vander Lugt fut de proposer une méthode pour la réalisation d'une fonction complexe sur un support physique. Il utilisa l'holographie pour enregistrer ce filtre. Le corrélateur optique tel que développé par Vander Lugt est celui que l'on peut voir à la figure 1.2. La puissance du corrélateur optique repose sur la propriété des lentilles sphériques convergentes à effectuer la T.F. d'une image en parallèle et ce à la vitesse de la lumière. C'est cette immence puissance de calcul qui a stimulé bon nombre de chercheurs dans ce domaine.

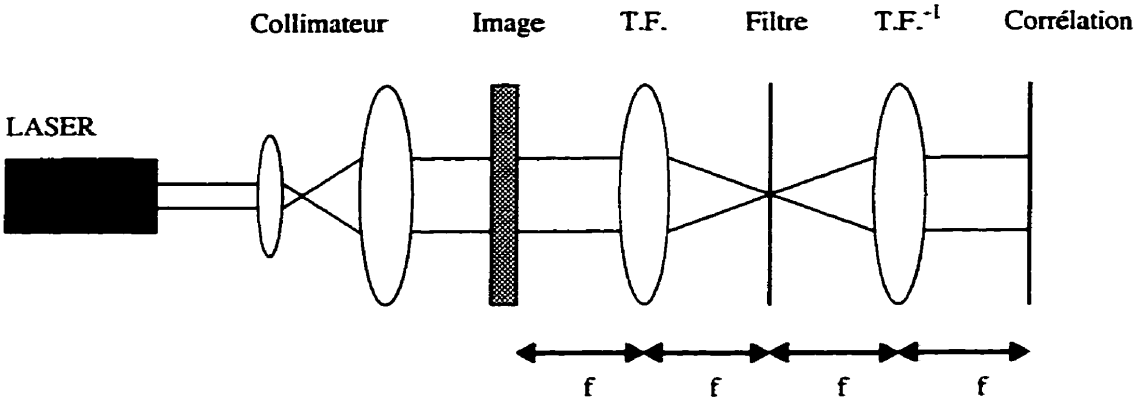

Figure 1.2 Le corrélateur optique Vander Lugt

Premièrement, la lumière cohérente issue du laser passe par un collimateur afin d'obtenir une onde plane. Le faisceau est par la suite modulé spatialement par l'image d'entrée. La première lentille réalise la T.F. de l'image d'entrée à son foyer. Ce plan où le filtrage est effectué est nommé dans la littérature *plan de Fourier*. En se référant à (1.6), le filtre H*(u,v) est inséré dans le plan de Fourier et est multiplié avec la T.F. de l'image d'entrée S(u,v). Le filtre H*(u,v) est simplement réalisé avec le complexe conjugué de la T.F. de l'objet à reconnaître. Si l'on omettait de prendre le complexe conjugué de H(u,v) pour la réalisation du filtre, le système effectuerait une convolution (Shulman, 1970). Le résultat de la multiplication est transmis à la seconde lentille qui effectue la $T.F.^{-1}$ et donne à son foyer le résultat de la corrélation. Les lentilles utilisées causent une inversion des axes de référence dans le plan de sortie (Casasent et Abramson, 1978). La figure 1.3 montre le plan de sortie du corrélateur où le pic de corrélation représente les coordonnées de l'objet reconnu par le système.

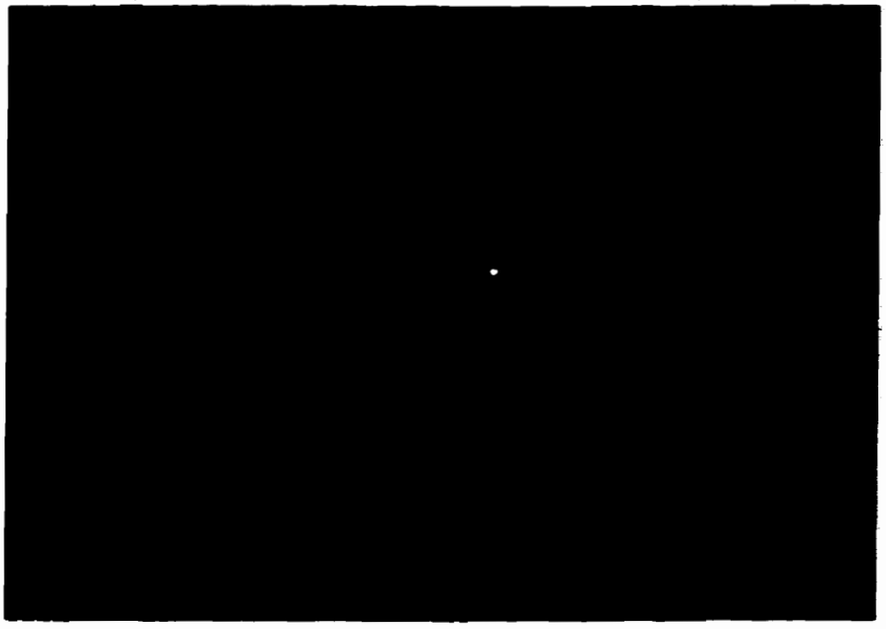

Figure 1.3 Un pic de corrélation

1.5 Filtres spatiaux

La sélection d'un filtre spatial est principalement influencée par l'application que l'on fera du corrélateur. Deux types de filtres seront traités: le filtre adapté (matched filter) et le filtre de phase (phase only filter).

Le filtre adapté est simplement réalisé en effectuant la T.F. de l'objet à identifer. Tout le contenu de la T.F. de l'objet est conservé, soit l'amplitude et la phase:

$$H(u, v) = T.F.(h(x, y)) = A_h(u, v)e^{j\Phi_h(u, v)}. \qquad (1.7)$$

Une seule opération reste à effectuer afin de réaliser une corrélation et c'est de prendre le complexe conjugué de la T.F. de l'objet à reconnaître:

$$H_a^*(\quad) = A_h(\quad)e^{-j\phi_h(\quad)}. \tag{1.8}$$

Le filtre de phase, comme son nom l'indique, ne conserve que la phase de la T.F. de l'objet à identifier, l'amplitude dans le plan de Fourier reste intouchée. Afin de réaliser une corrélation le filtre possède la phase conjuguée de la T.F. de l'objet à reconnaître:

$$H^*_p(\quad) = e^{-j\phi_h(\quad)}. \tag{1.9}$$

Ces deux types de filtres ont été comparés dans la littérature (Flannery et Horner, 1989) où certaines caractéristiques sont mises en évidence. L'immunité au bruit est généralement plus élevée avec le filtre adapté qu'avec le filtre de phase: dans une scène d'entrée hautement bruitée, le filtre adapté est meilleur que le filtre de phase afin d'identifier un objet.

Le rapport signal à bruit dans le plan de corrélation est plus petit pour le filtre de phase que pour le filtre adapté lorsqu'ils sont soumis à la même entrée. Cette preuve mathématique (Yu et Gregory, 1996) a été réalisée en additionnant à l'image d'entrée du bruit gaussien de moyenne nulle.

Par contre, les pics de corrélation obtenus à la sortie d'un même système sont plus étroits et plus intenses avec le filtre de phase qu'avec le filtre adapté (Flannery et Horner, 1989; Cohn et Horner, 1994). Une simulation par ordinateur (Flannery et Horner, 1989) a démontré que le filtre de phase donnait un pic de corrélation environ 300 fois plus élevé qu'avec le filtre adapté pour des images d'entrée avec un fond constant. A première vue, cette performance peut s'expliquer simplement. Le filtre de phase possède une haute transmission énergétique comparativement au filtre adapté, car le filtre de phase laisse passer les fréquences spatiales sans aucune atténuation. Ainsi, le filtre de phase démontre une plus grande efficacité énergétique pour l'image d'entrée acheminant un maximum d'énergie au pic de corrélation: simple conservation d'énergie.

De plus, le filtre de phase offre une plus grande discrimination contre les autres formes. Ce qui peut être très pratique lorsque l'objet à reconnaître possède des similitudes par rapport à un autre objet tout en comportant des différences. Cet avantage fait ressortir un compromis entre la selectivité et la tolérance du corrélateur optique auquel l'utilisateur est confronté. Plus un filtre est sélectif, plus il est sensible aux légères différences de l'objet, donc plus il est sensible au bruit.

Un autre avantage des filtres de phase est la réduction de l'espace mémoire nécessaire pour la conservation des filtres. Pour un même canal de communication, il est clair qu'un filtre de phase peut être chargé plus rapidement dans un corrélateur optique, ce qui est un avantage indéniable pour des applications en temps réel.

Des travaux antérieurs (Horner et Gianino, 1984) en traitement d'images ont démontré que l'information véhiculée par la phase, dans le plan de Fourier, est beaucoup plus importante que l'information contenue dans l'amplitude, dans la reconstruction d'images.

1.6 Les modulateurs spatiaux de lumière (MSL)

C'est sur un MSL que le filtre spatial d'un corrélateur optique est enregistré. Un MSL a la faculté de tranformer l'amplitude, la polarisation ou la phase de rayons lumineux cohérents. Plusieurs catégories de MSL existent. Ils sont basés sur différents matériaux et utilise divers phénomènes physiques comme: l'électro-optique, l'acousto-optique, la magnéto-optique,... À titre d'exemple, un MSL de la catégorie électro-optique composé de cristaux liquides (C.L.) est traité dans cette section.

Un MSL à C.L. est composé de plusieurs cellules ou pixels. Chaque cellule est constituée de C.L. allignés qui sont emprisonnés entre deux plaques de verre. Un champ électrique appliqué à travers une cellule modifie l'orientation des molécules de C.L. et de ce fait change les propriétés du matériau. Comparés aux autres matériau, ce sont les C.L. qui produisent le plus grand changement électro-optique pour un même voltage (Braham et Horner, 1994). L'asymétrie de la périodicité cristalline des C.L. cause une anisotropie de l'indice de réfraction de la lumière. Ainsi, la modification de l'alignement des C.L. permet d'obtenir une cellule à biréfringence variable en fonction du voltage appliqué. La biréfringence est le nom que l'on donne à la propriété des matériaux de posséder des indices

de réfraction différents selon l'axe (Hecht, 1988): un indice de réfraction ordinaire (n_o) et un indice de réfraction extraordinaire (n_e). Ainsi, il existe un axe de polarisation de la lumière incidente sur les C.L. pour lequel la variation de phase est maximale et la variation de polarisation est minimale pour une plage de tension donnée. Inversement, il existe un axe pour lequel la variation de polarisation est maximale et la variation de phase est minimale pour une plage de tension différente.

La modulation d'amplitude atténue ou laisse passer des rayons lumineux sans changer leur phase. Il est clair que le MSL ne peut pas amplifier la lumière. Un montage illustrant la modulation d'amplitude est présentée à la figure 1.4. La lumière cohérente en provenance du laser n'est pas polarisée. Dans cet exemple, le premier polariseur ne laisse passer que la lumière polarisée verticalement. Les C.L., selon la tension appliquée, changent la polarisation de la lumière incidente en de la lumière polarisée elliptiquement. Le dernier polariseur ne laisse passer que les composantes de la lumière qui ont été engendrées par les C.L., soit de la lumière polarisée horizontalement. Ainsi, le MSL peut moduler en amplitude sans moduler la phase car $n \equiv 1$ selon l'axe choisi.

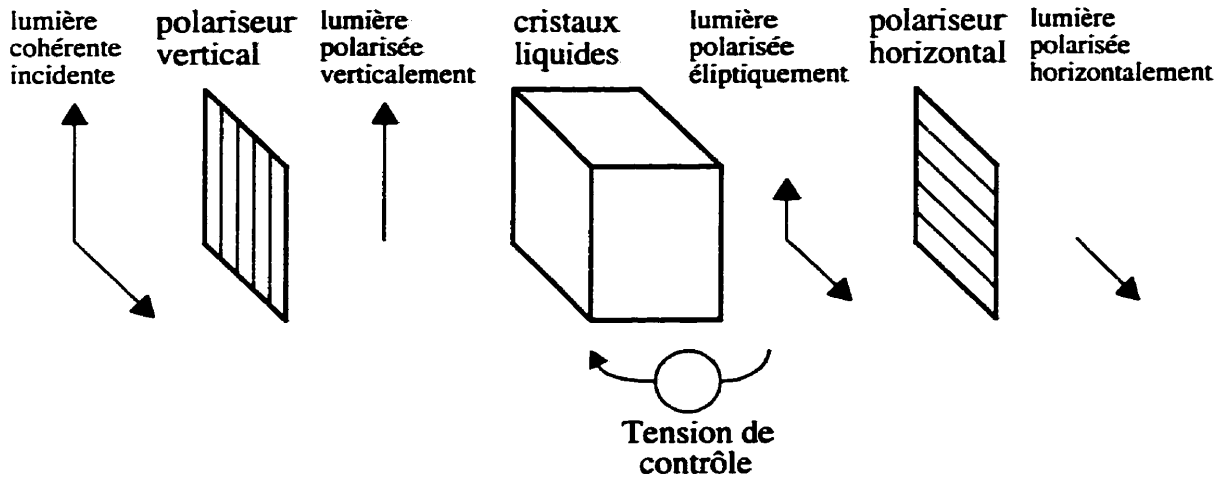

Figure 1.4 Modulation d'amplitude par un M.S.L.

La modulation de phase est possible grâce à des changements de l'indice de réfraction (n) des C.L.. Une modification de n provoque un changement de vitesse de propagation des ondes dans les C.L. selon (1.10), où v est la vitesse de l'onde dans le milieu, c est la vitesse de la lumière et n est l'indice de réfraction du milieu (Hecht, 1988). Le montage nécessaire pour la modulation de phase est identique à celui de la figure 1.4, mais une autre configuration des polariseurs sera utilisée: les polariseurs sont parallèles et permettent la propagation d'un axe de polarisation pour lequel on obtient une variation maximale de n et une variation négligeable de la polarisation.

$$v = \frac{c}{n} \tag{1.10}$$

Avec la technologie actuelle, les MSL représentent le goulot d'étranglement pour les performances d'un corrélateur optique. La principale mesure de performance des MSL est

le contraste R_c (1.11) exprimée comme le rapport de la transmission maximale de la lumière T_{max} à la transmission minimale T_{min} (Turner et al., 1993).

$$R_c = \frac{T}{t_{min}} \tag{1.11}$$

Un R_c d'environ 80 est typique des MSL réalisés avec des C.L. (Bergeron et al., 1995). Des simulations ont démontré (Gianino et al., 1995) qu'un abaissement de R_c pour le MSL donnait des pics de corrélation moins intenses, une augmentation du bruit en arrière-plan (Nekrasov et al., 1992) et une baisse dans les mesures de performance. De ce fait, un corrélateur de qualité possède un MSL dont le R_c est le plus élevé possible.

Une autre unité de comparaison entre différents MSL est le produit du taux de rafraîchissement des images par la résolution du MSL. Un débit de 10^{12} pixels/sec semble être présentement la norme (Braham et Horner, 1994), mais ce chiffre évolue constament.

1.7 Mesures de performances des corrélateurs optiques

Considérons le signal r(x) composé du signal s(x) que nous désirons détecter et du bruit n(x) de moyenne nulle et distribué selon une gaussienne.

$$r(\ ) = s(\ ) + n(\ ) \tag{1.12}$$

Le corrélateur aura en entrée le signal r(x). Le but premier de la détection est de décider si

le signal s(x) est présent dans le signal r(x), de sorte qu'il est possible de fixer un seuil pour le pic de corrélation selon lequel le signal recherché est inclus dans le signal d'entrée. Pour cela, il faut faire un bon choix de filtre à inclure dans le corrélateur. Seuls les critères de performance les plus utilisés permettant la comparaison de filtres seront énumérés.

La nature aléatoire du bruit peut provoquer une non-détection du signal à détecter lorsqu'il est présent, causée par un abaissement aléatoire du pic. L'inverse est aussi possible, soit une forte augmentation aléatoire d'un pixel causant un faux pic et par conséquent une fausse détection. Un bon critère pour caractériser cette possibilité d'erreur avec les différents filtres disponibles est le rapport du signal à bruit (SNR) (1.13). Un filtre qui minimisera les erreurs causées par le bruit aura un SNR élevé.

$$SNR = \frac{|E[\quad]|^2}{var\{\quad\}} \tag{1.13}$$

Dans cette équation y(0) est la valeur du pic de corrélation, E{...} signifie la moyenne de l'ensemble et var{...} représente la variance du même ensemble. Ce critère demande plusieurs événements statistiquement indépendants pour être appliqué : typiquement de 100 à 500 corrélations (Horner, 1992).

L'efficacité d'Horner (η_H) est une mesure directe de la fraction de l'énergie lumineuse mise à l'entrée du corrélateur qui participe réellement au traitement (Horner, 1992; Caulfield, 1982).

$$\eta_H = \frac{y(0)^2}{\sum_i s(x_i)^2} \tag{1.14}$$

L'équation (1.14) décrit l'efficacité d'Horner comme étant le rapport de l'énergie du pic de corrélation à l'énergie dans l'image d'entrée. Le numérateur est obtenu en effectuant le carré de la valeur du pic de corrélation (y(0)) et le dénominateur représente la somme du carré de chaque pixel dans l'image d'entrée (s(x)). Si l'on se rapporte à la figure 1.2, l'image d'entrée serait le plan à la droite du collimateur, soit "Image". De plus, ce paramètre donne un indice sur l'étroitesse du pic de corrélation : un pic étroit est toujours préférable car sa localisation sera plus précise. Des simulations par ordinateur comparant un filtre de phase à un filtre adapté, ont donné un η_H 83 fois plus grand pour le filtre de phase (Horner et Gianino, 1984).

L'étroitesse d'un pic peut aussi être quantifiée selon un autre critère : le PCE (Peak to Correlation Energy) (Vijaya Kumar et Hassebrook, 1990; Horner, 1992). Le PCE est décrit comme étant le rapport de l'énergie du pic de corrélation à l'énergie du plan de corrélation. Le PCE avoisine 0 pour des pics très larges. Pour des pics étroits le PCE peut valoir jusqu'à 1 au maximum. De plus, le PCE est un résultat facile à obtenir, car il nécessite seulement le plan de corrélation pour être calculé. Si l'on se rapporte encore à la figure 1.2, seul le plan "Corrélation" serait nécessaire. Cette mesure peut être appliquée à deux différents corrélateurs soumis à la même entrée afin de comparer leurs performances quant à la production d'un pic étroit. Plus de détails seront donnés sur le PCE dans le chapitre traitant

de l'algorithme de post-traitement du corrélateur optique.

Un autre critère pour mesurer l'étroitesse d'un pic est le ratio de la valeur du pic de corrélation à la valeur du second pic. Ce second pic ou pic secondaire est défini comme étant la plus grande valeur à l'extérieur d'une région autour du pic principal. Cette région s'étend jusqu'à ce que la valeur tombe, pour la première fois, en dessous de la moitié de la valeur du pic de corrélation. Deux problèmes se posent avec cette mesure (Vijaya Kumar et Hassebrook, 1990). Le premier est l'identification de la région autour du pic principal, afin de trouver le pic secondaire, ce qui est difficilement traduisible en une expression analytique. Le deuxième est que le pic secondaire peut être l'unique résultat du bruit qui ne possède aucun lien avec l'étroitesse du pic de corrélation. Des chercheurs ont trouvé une méthode pour synthétiser un filtre de phase qui ne produit pas de pic secondaire et qui donne un pic de corrélation plus étroit (Roberge et Sheng, 1996).

CHAPITRE 2

L'ALGORITHME DE POST-TRAITEMENT DU CORRÉLATEUR OPTIQUE

Ce chapitre traite de l'investigation effectuée afin de proposer un algorithme basé sur la comparaison de mesures d'évaluation de pics de corrélation. Le but premier de cet algorithme est de départager les pics de corrélation des corrélations croisées. Tout au long de ce chapitre, nous appelons un pic de corrélation, un intense et étroit pic lumineux dans le plan de corrélation qui est la conséquence de la détection de l'objet recherché dans la scène. De la même façon, une corrélation croisée est un amas évasé de pixels de faible ou de moyenne intensité dans le plan de corrélation qui est la conséquence de la présence d'un objet ressemblant à l'objet recherché. L'algorithme de post-traitement du corrélateur optique pourra signaler la présence ou l'absence de l'objet à détecter grâce à cette discrimination faite sur chaque pic de plan de corrélation. L'algorithme s'applique seulement sur le pic le plus intense dans l'éventualité où plusieurs pics seraient présents dans le plan de corrélation. Nous donnerons les résultats de cinq différentes mesures qui ont été appliquées sur la banque originale, sur la banque filtrée avec un filtre de moyenne 3 par 3, avec un filtre médian 3 par 3 et avec un filtre de moyenne 5 par 5. Ces mesures sont la valeur du pic de corrélation, le PCE, le PRR, la pente et le moment du pic de corrélation. Tous ces résultats ont été obtenus avec Matlab suite à une traduction par un script de chaque image en matrice.

2.1 La banque des plans de corrélation

Une banque de 16 images de plan de corrélation nous a été fournie par l'Institut National d'Optique pour l'application de mesures d'évaluation. Cette banque, très représentative de la réalité, comporte des pics de corrélation et des corrélations croisées de diverses qualités: une corrélation croisée intense, un faible pic de corrélation... Les premières images représentent des plans de corrélation lorsque le corrélateur optique n'est soumis qu'à un seul objet en entrée: une image comportant un pic de corrélation de reconnaissance de qualité (image1a) et six images de corrélation croisée (image2a-7a). Les dernières images (image8a-16a) illustrent un plan de corrélation avec plusieurs pics de reconnaissance. Toutes ces images sont en noir et blanc et ont une résolution de 640 pixels par 480 lignes. Les pixels sont quantifiés sur 8 bits. En tout, l'ensemble des plans de corrélation est constitué de 10 images comportant des pics de corrélation et 6 images ne possédant que des corrélations croisées. La figure 2.1 montre un aperçu tri-dimensionnel et la figure 2.2 une coupe en deux dimensions de la région d'un pic de corrélation de qualité. La figure 2.3 et la figure 2.4 illustre la même chose mais pour une corrélation croisée.

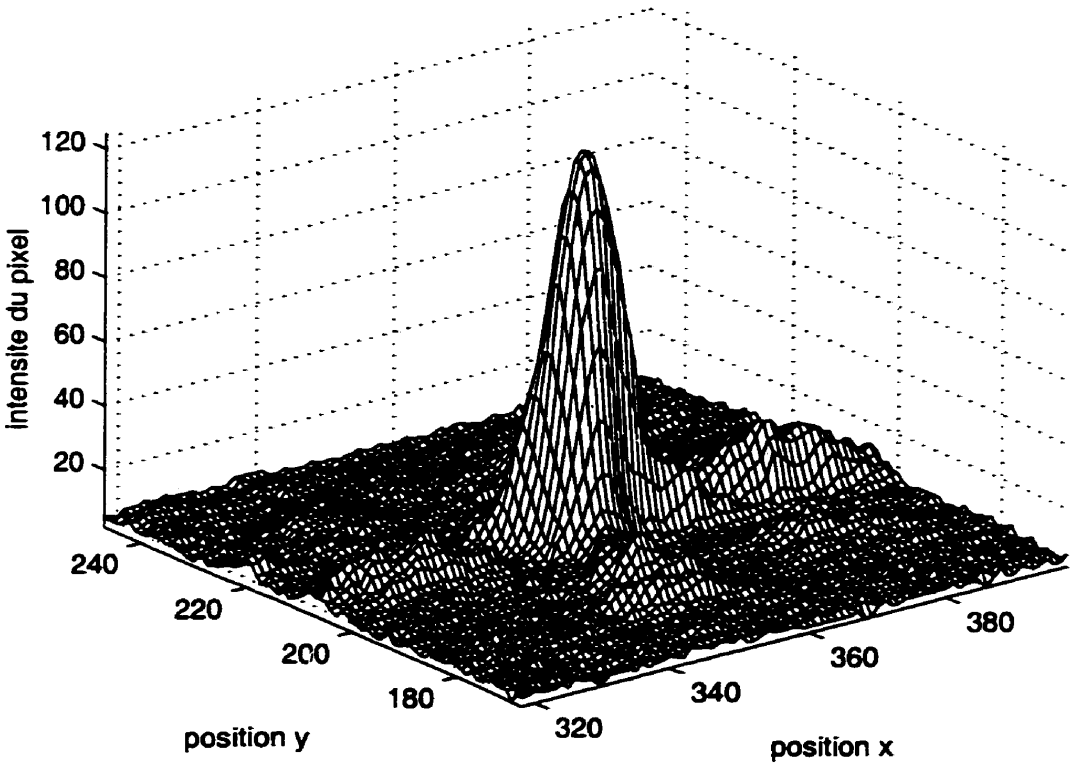

Figure 2.1 Graphique tri-dimensionnel de la région d'un pic de corrélation (image1a)

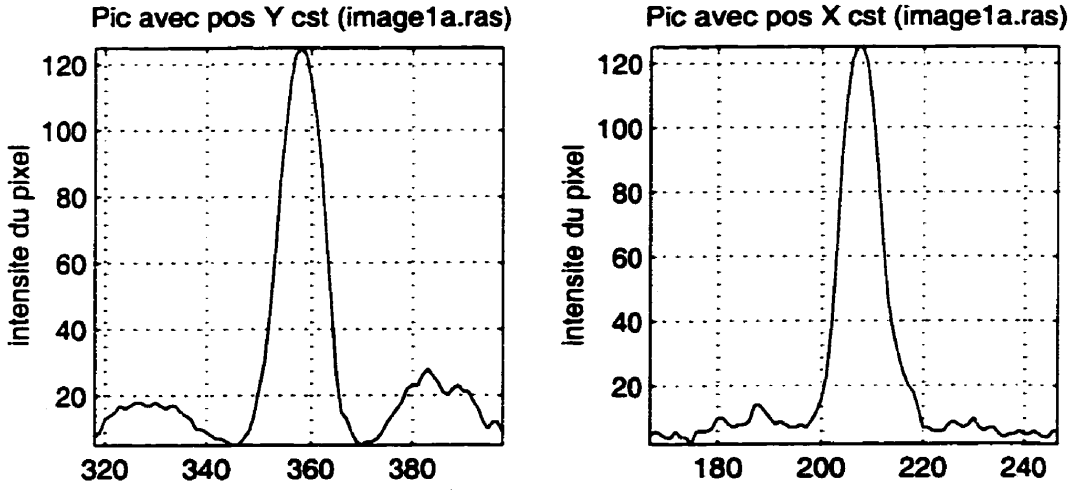

Figure 2.2 Coupe en 2 dimensions de la région d'un pic de corrélation (image1a)

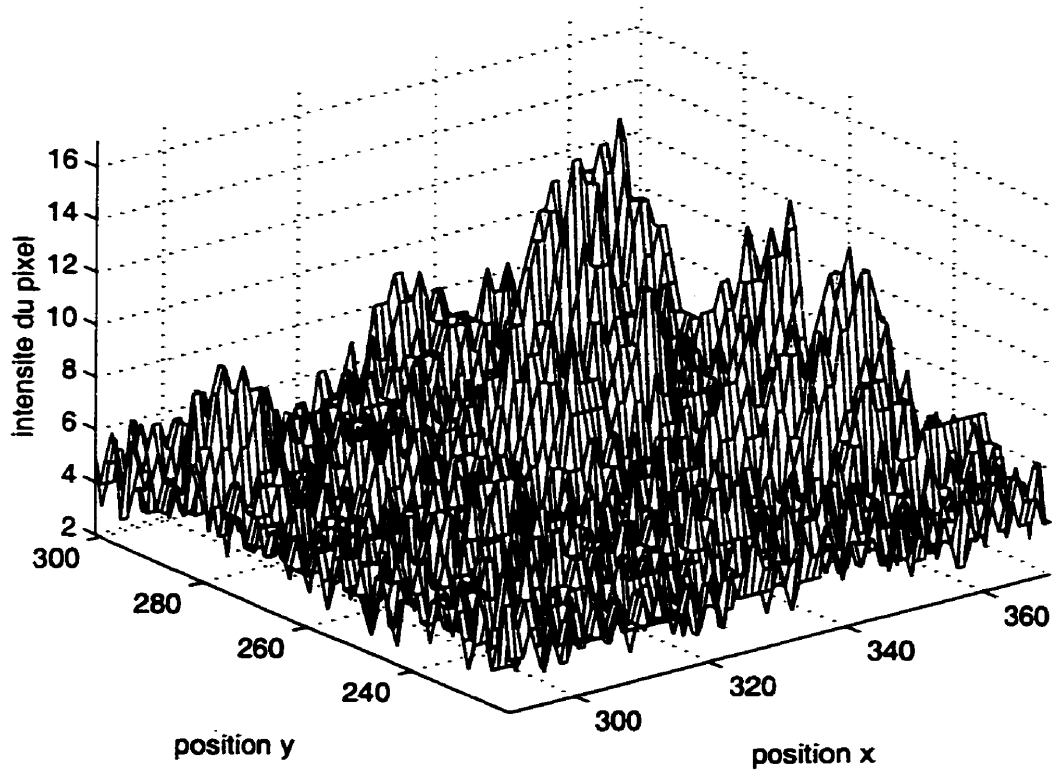

Figure 2.3 Graphique tri-dimensionnel de la région d'une corrélation croisée (image7a)

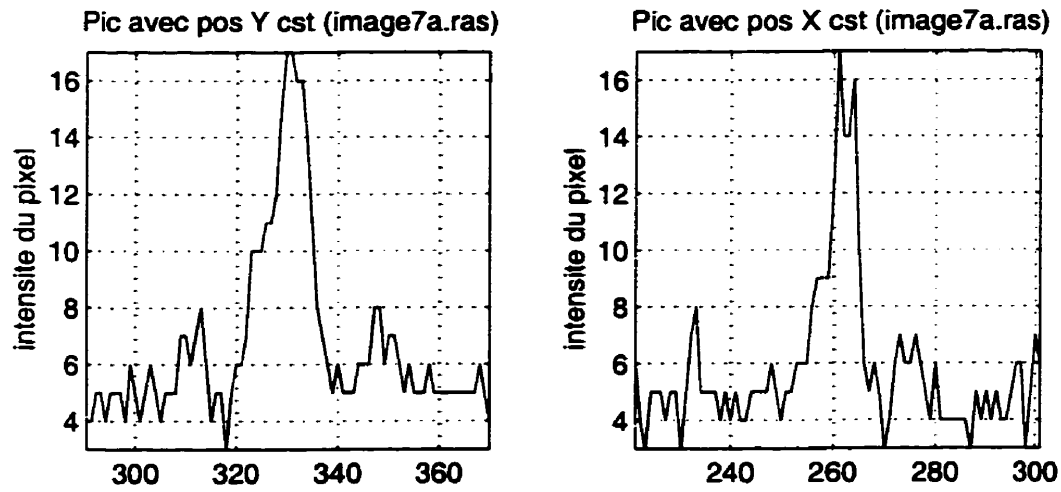

Figure 2.4 Coupe en 2 dimensions de la région d'une corrélation croisée (image7a)

2.2 Le bruit dans les plans de corrélation

Le bruit présent dans les plans de corrélation peut être la conséquence de plusieurs causes: de faibles irrégularités des lentilles sphériques convergentes, le contraste du modulateur spatial de lumière qui n'approche pas l'infini, les capteurs de la caméra CCD de sortie, l'écran à cristaux liquides qui affiche la scène observée, ... Ce bruit de faible intensité à une moyenne d'environ 5 sur une échelle de 256. Comme le montre les dernières figures, ce bruit est peu apparant dans la région d'un pic de corrélation. La forte intensité du pic de corrélation masque le bruit. À l'inverse, les pics du bruit peuvent représenter de 30% à 50% de la valeur du pixel le plus intense d'une corrélation croisée. L'objectif de l'algorithme est d'identifier un changement significatif d'intensité dans le plan de corrélation correspondant à une reconnaissance. Atténuer ce bruit pourrait correspondre à affaiblir les corrélations croisées et de ce fait réduire les chances d'une fausse détection du système. L'énergie d'un pic de corrélation est située principalement dans une plage fréquentielle plus basse que l'énergie du bruit. L'application d'un filtre spatial passe-bas sur les plans de corrélation aurait pour effet de réduire le bruit composé de hautes fréquences et d'avoir peu d'impact sur un pic de reconnaissance. Le choix d'un tel filtre doit avoir un support spatial suffisamment grand pour éliminer le bruit, mais il doit être assez petit pour ne pas détériorer le pic de corrélation et pour ne pas changer sa localisation.

Filtrer une image revient à effectuer une convolution en deux dimensions de cette dernière avec une fenêtre décrivant le filtre. Un exemple de convolution en deux dimensions entre

une fenêtre de 3 par 3 pixels et une image est donné à la figure 2.5. La fenêtre de convolution courante de l'image originale est en gris et le pixel produit par cette même fenêtre est en noir dans l'image filtrée. Les pixels hachurés de l'image originale n'auront plus à servir pour cette convolution et les pixels en gris représentent les pixels filtrés. Ainsi, chaque pixel filtré est une fonction du même pixel dans l'image originale et ses 8 pixels avoisinants. Les pixels situés sur la limite de l'image filtrée sont discartés. Ainsi, l'image filtrée est plus petite d'une lisière d'un pixel sur sa périphérie par rapport à l'image originale. Cette lisière est représentée par les pixels hachurés en double.

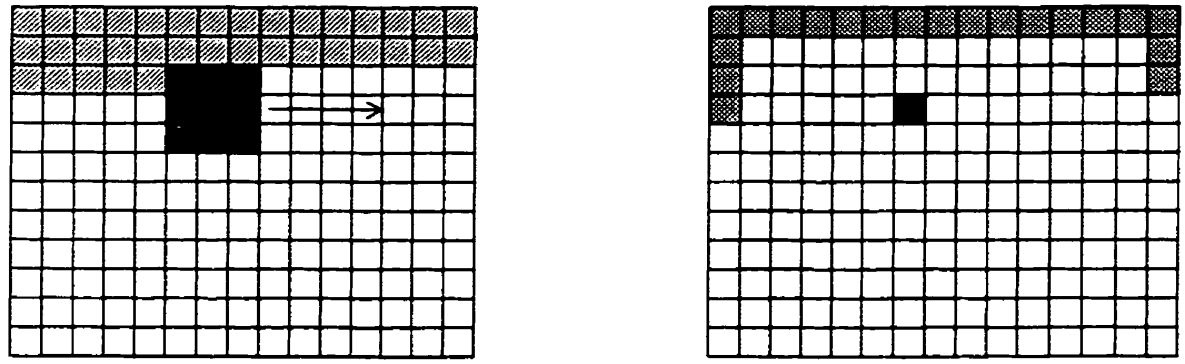

Image Image filtrée

Figure 2.5 Convolution en deux dimensions

Deux types de filtre passe-bas ont été appliqués sur la banque de plan de corrélation : le filtre de moyenne et le filtre médian. Dans le cas du filtre de moyenne 3 par 3, le calcul qui est appliqué pour chaque fenêtre est le suivant soit une moyenne du pixel courant avec ses 8 plus proches voisins.

$$\text{image filtrée}(i,j) = \frac{\sum_{m=-1}^{m=1}\left(\sum_{n=-1}^{n=1} \text{image}(i+m, j+n)\right)}{9}$$

Comme le résultat peut être fractionnaire, un arrondissement est effectué afin d'obtenir une valeur entière. Le filtre de moyenne 5 par 5 a aussi été appliqué.

Une autre manière de filtrer consiste à prendre la médiane d'une fenêtre de convolution. Plus spécifiquement, le filtre médian 3 par 3 classe en ordre croissant les pixels faisant partie de la fenêtre de convolution et choisit pour pixel résultant le $5^{ième}$ pixel des 9, soit le pixel médian. Ces deux filtres passe-bas ont le rôle d'atténuer les courtes variations d'intensité dans l'image.

Le corrélateur optique de l'Institut National d'Optique possède un filtre de phase. Tel que discuté au chapitre précédent, ce filtre produit un niveau de sortie plus grand mais il produit aussi un niveau de bruit plus grand dans le plan de corrélation que lorsque le corrélateur utilise un filtre adapté. Ainsi, l'application de ces filtres passe-bas sur les plans de corrélation contribue à nous débarasser de ce désavantage des filtres de phase, tout en conservant la plus grande discrimination offerte par ce type de filtre.

2.3 La discrimination

Parce que chaque mesure d'évaluation de pic de corrélation donne des plages de valeurs différentes, une métrique est nécessaire afin de pouvoir comparer les performances des différentes mesures d'évaluation appliquées sur les pics. Nous définissons la discrimination comme étant le rapport entre une mesure appliquée sur un pic de reconnaissance à la même

mesure appliquée à une corrélation croisée. Ainsi, la meilleure discrimination d'une mesure pour une banque d'images est le rapport entre la plus grande valeur obtenue pour un pic de corrélation et la plus faible valeur produite pour une corrélation croisée. À l'inverse, la pire discrimination d'une mesure appliquée à un ensemble de plans de corrélation est le rapport entre la plus faible valeur trouvée pour un pic de corrélation et la plus grande valeur obtenue pour une corrélation croisée. La discrimination moyenne d'une mesure est le rapport entre la moyenne des valeurs obtenues pour les pics de corrélation et la moyenne des valeurs produites pour les corrélations croisées. De cette façon, la pire discrimination représente l'extrémité inférieure de la performance d'une mesure et la meilleure discrimination sa borne supérieure. La discrimination moyenne nous renseigne sur la performance typique d'une mesure. Ces concepts de meilleure discrimination, pire discrimination et discrimination moyenne sont appliqués de façon identique pour toutes les mesures.

2.4 La valeur du pic de corrélation

La valeur du pic de corrélation est la mesure la plus importante et la plus facile à obtenir. Elle donne une mesure directe de l'intensité d'un possible pic de corrélation. L'algorithme permettant d'obtenir cette valeur compare les pixels un à un et conserve, pour chaque image, le plus intense avec ses coordonnées en x et y. Ce pixel représente le centre d'un pic de corrélation qui est toujours constitué de plusieurs pixels. Ses coordonnées et sa valeur servent de référence pour d'autres mesures. Le tableau 2.1 montre les résultats de la valeur du pic de corrélation trouvés pour les quatre banques d'images. Les trois banques d'images

filtrées ont été obtenues à partir de la banque originale. Les noms d'images en gras représentent celles comportant un pic de corrélation de reconnaissance.

Tableau 2.1 Valeurs et positions du pic pour les images de chaque banque

	position x	position y	Valeur du pic			
			originale	filtre de moyenne 3x3	filtre médian 3x3	filtre de moyenne 5x5
image1a	358	207	125	122	123	114
image2a	318	222	16	14	13	12
image3a	391	181	11	10	10	9
image4a	395	217	13	11	11	10
image5a	337	210	11	10	10	9
image6a	213	235	11	10	10	9
image7a	330	261	17	15	16	14
image8a	358	209	108	101	101	89
image9a	369	187	102	97	99	85
image10a	247	288	109	102	103	90
image11a	244	264	105	99	100	87
image12a	242	236	89	86	86	78
image13a	470	105	90	85	85	72
image14a	193	106	66	63	63	57
image15a	287	298	105	101	102	89
image16a	431	233	51	48	48	40

Le tableau 2.2 montre les différentes discriminations obtenues avec les quatre banques. Les performances sont très semblables pour chaque banque, le filtrage a eu peu d'influence.

Tableau 2.2 Comparaison des discriminations pour la valeur du pic de corrélation

Banque	meilleure discrimination	pire discrimination	discrimination moyenne
Originale	11,364	3,353	7,215
Filtre de moyenne 3x3	12,200	3,200	7,748
Filtre médian 3x3	12,300	3,000	7,800
Filtre de moyenne 5x5	12,667	2,857	7,629

Il serait risqué de se baser uniquement sur cette mesure de post-traitement de corrélateur optique afin de tirer des conclusions quant à la présence ou à l'absence de l'objet recherché dans une scène. Cette mesure est fortement dépendante du niveau de biais dans l'image donc de l'énergie à l'entrée du système. Par exemple, selon cette mesure des résultats semblables pourraient être obtenus pour un pic de corrélation lorsque le système a peu d'énergie en entrée et pour une corrélation croisée lorsque le corrélateur est soumis à plus d'énergie.

2.5 Le PCE

Le PCE (Peak to Correlation Energy) est le rapport du pixel le plus intense du pic sur la somme de tous les pixels du plan de corrélation. L'algorithme pour le PCE est identique à celui de la valeur du pic de corrélation, sauf que la somme de tous les pixels est exécutée

en plus. Les résultats obtenus avec les quatre banques peuvent être visualisés au tableau 2.3.

Tableau 2.3 Valeur du PCE pour les images de chaque banque

	PCE (10^{-6})			
	originale	filtre de moyenne 3x3	filtre médian 3x3	filtre de moyenne 5x5
image1a	90,522	88,007	89,081	81,924
image2a	12,480	10,948	10,155	9,401
image3a	8,745	7,985	7,974	7,206
image4a	10,437	8,869	8,860	8,090
image5a	8,694	7,924	7,911	7,145
image6a	8,757	7,986	7,981	7,201
image7a	13,221	11,694	12,472	10,927
image8a	85,931	80,611	80,590	71,196
image9a	81,359	77,631	79,163	68,181
image10a	86,471	81,198	82,006	71,797
image11a	83,462	78,901	79,711	69,492
image12a	70,532	68,432	68,400	62,229
image13a	71,922	68,171	68,151	57,906
image14a	52,740	50,527	50,445	45,831
image15a	106,640	102,880	103,860	90,796
image16a	39,750	37,523	37,497	31,324

Le tableau 2.4 compare les différentes discriminations obtenues avec le PCE. Ces résultats sont presque identiques aux discriminations obtenues avec la mesure du pic de corrélation. Ceci s'explique par la grande constance retrouvée avec la somme des pixels des plans de

corrélation pour toutes les images. La somme des pixels pour une image est en moyenne égale à 1 250 000, peu importe la présence d'un pic de corrélation.

Tableau 2.4 Comparaison des discriminations pour le PCE

Banque	meilleure discrimination	pire discrimination	discrimination moyenne
Originale	12,266	3,007	7,405
Filtre de moyenne 3x3	12,983	3,209	8,074
Filtre médian 3x3	13,129	3,006	8,009
Filtre de moyenne 5x5	12,708	2,867	7,813

2.6 Le PRR

Le PRR (Peak to Radius Ratio) est le rapport de la valeur du pic de corrélation sur la moyenne des longueurs des rayons du pic. Un rayon est constitué des pixels situés entre celui qui est le plus intense et le premier rencontré dont la valeur est inférieure à un certain pourcentage du plus intense. En théorie, cette mesure devrait être faite en coordonnée polaire, mais les images étant constituées de pixels, il est beaucoup plus pratique de travailler en coordonnées cartésiennes. De plus, un compromis est proposé ici tenant compte de coordonnées cartésiennes et d'un système temps réel qui n'aurait pas à conserver toute l'image. L'algorithme pour le PRR trouve le pic de corrélation, il calcule la valeur limite selon un pourcentage du pixel le plus intense et finalement il trouve les rayons en x et en y avec seulement le quart inférieur droit du pic de corrélation, voir figure 2.6. Le dénominateur servant au calcul du PRR est la moyenne des rayons en x et en y. Cette

approximation de la caractérisation du pic de corrélation avec seulement le quart inférieur droit est rendue possible grâce à sa grande symétrie (figure 2.1 et figure 2.2).

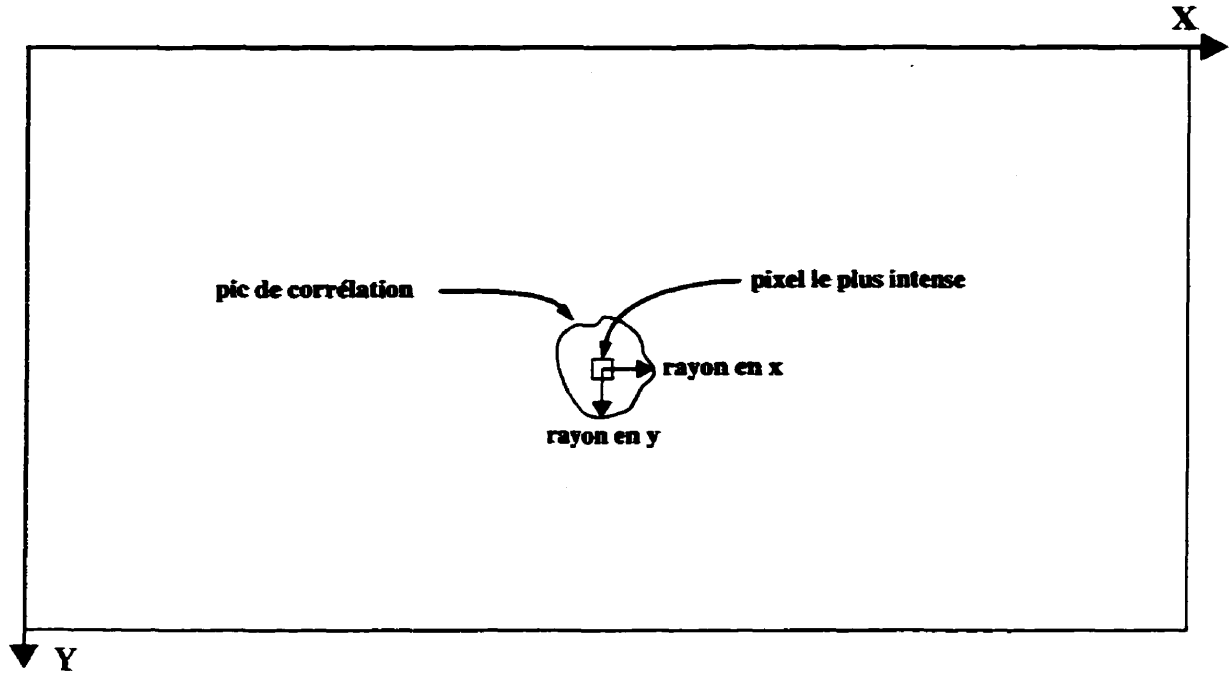

Figure 2.6 Les rayons d'un pic

Des mesures du PRR furent prises à 50, 30 et 20% du pixel le plus intense. Le tableau 2.5 montre les valeurs du PRR à 20%. La valeur moyenne des rayons est environ égale à 8 pour les pics de corrélation et 275 pour les corrélations croisées. À maintes reprises, le rayon d'une corrélation croisée atteint les limites de l'image.

Tableau 2.5 Valeur du PRR à 20% pour les images de chaque banque

	PRR à 20%			
	originale	filtre de moyenne 3x3	filtre médian 3x3	filtre de moyenne 5x5
image1a	14,706	13,556	14,471	12,000
image2a	0,252	0,048	0,045	0,041
image3a	0,040	0,037	0,037	0,033
image4a	0,051	0,043	0,043	0,039
image5a	0,038	0,038	0,038	0,034
image6a	0,044	0,030	0,030	0,027
image7a	0,642	0,057	0,061	0,053
image8a	14,400	12,625	13,467	10,471
image9a	13,600	12,125	13,200	10,000
image10a	14,533	12,750	13,733	11,250
image11a	15,000	13,200	13,333	10,235
image12a	11,867	10,750	11,467	9,177
image13a	12,857	11,333	11,333	8,000
image14a	8,250	7,412	7,875	6,333
image15a	14,000	13,467	13,600	11,125
image16a	8,500	8,000	8,000	5,333

Le tableau 2.6 compare les discriminations du PRR trouvées sur les quatre banques. C'est le PRR à 20% qui donne les meilleures performances avec les banques filtrées par les filtres 3x3 de moyenne et médian. De plus, le filtrage des plans de corrélation augmente

considérablement la discrimination : tous les rayons des corrélations croisées calculés pour le PRR à 20% atteignent les limites de l'image. Lorsqu'appliqué, le filtre de moyenne 3x3 offre une pire discrimination 10.1 fois plus élevée qu'avec la banque originale et une discrimination moyenne 3.8 fois supérieure. Le filtre médian 3x3 a eu pour impact d'améliorer de 10.1 fois la pire discrimination et 4.0 fois la discrimination moyenne. Ces deux filtres offrent des performances presque semblables.

Ces grandes améliorations des pires discriminations sur les images non filtrées s'expliquent facilement. Pour les pires discriminations, l'amélioration est causée par le débordement de tous les rayons des corrélations croisées sans exceptions. Cette tendance générale au débordement des rayons d'une corrélation croisée est causée par l'affaiblissement du pixel le plus intense et par le rehaussement des pixels de faible intensité causés par l'application de ces filtres. Avec un pixel le plus intense plus petit après filtrage, il en résulte une valeur limite plus petite ce qui cause une condition d'arrêt plus sévère. Cette condition d'arrêt est plus difficile à rencontrer à cause de l'augmentation de la valeur des pixels les plus faibles. Ainsi, avec des rayons plus grands, le PRR s'en retrouve plus petit. Le filtre de moyenne 5x5 donne de moins bons résultats que le 3x3. Ceci s'explique par le fait que le 5x5 atténue plus fortement le pixel le plus intense d'un pic de corrélation, ce qui lui donne un PRR plus petit. Si on analyse les données du tableau 2.5, on peut voir qu'il n'y a presque aucun changement, entre les filtres de moyenne 3x3 et 5x5, pour le PRR à 20% des corrélations croisées. Mais pour celui des pics de corrélation, il subit une baisse avec le 5x5.

Tableau 2.6 Comparaison des discriminations pour le PRR

Banque	meilleure discrimination	pire discrimination	discrimination moyenne
PRR à 50%			
Originale	19,636	3,187	9,904
Filtre de moyenne 3x3	21,805	3,912	11,001
Filtre médian 3x3	19,898	3,000	10,166
Filtre de moyenne 5x5	37,778	4,127	14,286
PRR à 30%			
Originale	64,935	3,882	14,770
Filtre de moyenne 3x3	439,649	21,818	79,689
Filtre médian 3x3	447,950	24,150	111,005
Filtre de moyenne 5x5	565,778	29,524	175,721
PRR à 20%			
Originale	394,737	12,850	71,747
Filtre de moyenne 3x3	451,867	130,035	273,033
Filtre médian 3x3	486,212	129,691	285,680
Filtre de moyenne 5x5	446,667	100,762	247,857

2.7 La pente du pic de corrélation

La pente du pic de corrélation est le rapport entre la différence de deux pixels et la distance qui les sépare. Le même compromis appliqué au PRR est fait ici. La pente du pic est la moyenne de la pente mesurée en x et en y pour le quart inférieur droit du pic. La pente n'est

pas évaluée à partir d'un pourcentage de la valeur du pic, elle est mesurée avec des fenêtres fixes en référence au pic de corrélation. Des fenêtres de 3 et 4 pixels ont été choisies lors des mesures. Une fenêtre de 2 pixels est jugée trop petite car sa sensibilité au bruit est plus grande. Une fenêtre de 5 pixels et plus est jugée trop grande, car elle excèderait la région du pic de corrélation. Les indices de pixels utilisés dans ces tableaux sont définis en référence au pic de corrélation: le pixel 0 est le pic de corrélation lui même. Les fenêtres commencent à partir du pixel 2 car la valeur de la pente autour du pic est faible. Le tableau 2.7 donne les discriminations obtenues avec cette mesure pour les meilleures fenêtres. Les fenêtres 4 à 7, 5 à 8, 4 à 8 et 5 à 9 ont aussi été mesurées, mais elles ont données de moins bons résultats : l'absence de ces résultats allège la lecture de cette section. Les nombreux tableaux donnant les valeurs de pente ont été omis pour la même raison. Pour la pente du pic de corrélation, le filtrage des plans de corrélation a eu peu d'influence et même dans certain cas il a abaissé les performances de la mesure. Aucune fenêtre ne s'est démarquée des autres.

Tableau 2.7 Comparaison des discriminations pour la pente du pic

Banque	meilleure discrimination	pire discrimination	discrimination moyenne
pixel 2 à 5			
Originale	124,754	4,538	15,152
Filtre de moyenne 3x3	26,736	3,714	11,800
Filtre médian 3x3	40,667	4,231	13,730
Filtre de moyenne 5x5	21,000	4,875	15,107
pixel 3 à 6			
Originale	38,334	2,474	9,636
Filtre de moyenne 3x3	53,054	3,728	12,278
Filtre médian 3x3	36,667	3,889	12,583
Filtre de moyenne 5x5	30,333	3,778	13,060
pixel 2 à 6			
Originale	46,667	4,000	13,908
Filtre de moyenne 3x3	32,750	3,688	12,188
Filtre médian 3x3	27,600	4,286	13,620
Filtre de moyenne 5x5	28,500	4,364	13,958
pixel 3 à 7			
Originale	49,000	2,526	11,474
Filtre de moyenne 3x3	33,500	3,833	12,646
Filtre médian 3x3	35,500	4,222	13,705
Filtre de moyenne 5x5	29,000	3,167	12,551

2.8 Le moment du pic de corrélation

Le moment du pic de corrélation est le rapport de la somme des pixels appartenant aux rayons multiplié par leur distance au pixel le plus intense sur la somme de ces pixels. Le pixel le plus intense est considéré comme le centre du pic de corrélation pour le calcul du moment.

$$moment = \frac{\sum(pixel_i \cdot distance_i)}{\sum(pixel_i)}$$

En théorie, le calcul du moment se fait facilement en coordonnées polaires, mais en pratique, mesurer le moment sur une matrice représentant une image se fait naturellement en coordonnées cartésiennes. Le même compromis appliqué au calcul du PRR et à la pente du pic est réalisé pour le calcul du moment. Le calcul du moment du pic est seulement effectué avec les pixels situés sur le rayon en x et le rayon en y, voir figure 2.6.

Les limites mises à l'essai pour le calcul du moment sont de 50, 30 et 20% de la valeur du pixel le plus intense. Le moment d'une corrélation croisée étant plus élevé que le moment d'une corrélation, l'inverse des moments est utilisé pour les calculs de discrimination. Les résultats de discrimination obtenus pour la mesure du moment sont disponibles au tableau 2.8. La limite à 50% offre une performance médiocre, avec dans le pire cas une discrimination à 0,780. Une discrimination inférieure à 1 signifie que la mesure appliquée a une corrélation croisée a donnée une plus grande valeur que pour un pic de corrélation. Ce qui signifie que la mesure ainsi appliquée ne peut départager un vrai pic de corrélation

d'un faux. La limite à 20% est celle qui a donné les meilleurs résultats pour le calcul du moment. Le filtrage des images a eu un impact positif sur les résultats. Les filtres de moyenne et médian donnent les mêmes performances.

Tableau 2.8 Comparaison des discriminations pour le moment

Banque	meilleure discrimination	pire discrimination	discrimination moyenne
moment à 50%			
Originale	2,332	0,780	1,533
Filtre de moyenne 3x3	2,359	0,906	1,503
Filtre médian 3x3	2,134	0,688	1,458
Filtre de moyenne 5x5	4,199	1,110	2,348
moment à 30%			
Originale	10,906	1,061	4,544
Filtre de moyenne 3x3	55,271	7,647	28,851
Filtre médian 3x3	68,793	7,946	31,850
Filtre de moyenne 5x5	69,088	13,363	43,102
moment à 20%			
Originale	55,306	4,513	31,632
Filtre de moyenne 3x3	71,448	35,099	44,357
Filtre médian 3x3	77,479	36,898	46,319
Filtre de moyenne 5x5	61,322	33,806	41,529

2.9 Analyse des performances des mesures d'évaluation

Des discriminations presques identiques ont été obtenues avec la valeur du pic de corrélation et le PCE. Cette ressemblance est causée par la constance retrouvée avec la somme des pixels de chaque image. Le filtrage des images a très peu changé la valeur de ces deux mesures. Même avec le filtrage des plans de corrélation, ces deux mesures demeurent peu discriminantes. Par contre, la mesure du PCE est indépendante de la quantité d'énergie dans le plan de corrélation donc de l'illumination à l'entrée du corrélateur optique. Supposons l'image I_1 et l'image I_2 où I_2 a été obtenue en multipliant tous les pixels de I_1 par 2. Dans cette équation, pix_max représente le pixel le plus intense et somme_pix la somme de tous les pixels de cette image.

$$\text{PCE}\, I_1 = \text{PCE}\, I_2 = \frac{\text{pix\_max}}{\text{somme\_pix}} = \frac{2 \cdot \text{pix\_max}}{2 \cdot \text{somme\_pix}}$$

De plus, le PCE mesure la proportion de l'énergie d'entrée qui se retrouve dans le pic de corrélation. Ceci peut donner un indice de la distorsion d'un objet lors de sa reconnaissance. Une opération de contrôle de la qualité lorsque de petits défauts doivent être détectés peut tirer avantage de cette mesure. Ainsi, le PCE peut être conservé afin de nous informer sur le niveau de distorsion d'un objet lorsque d'autres mesures s'occupent de la détection de ce dernier.

Les dernières sections nous indiquent clairement que la mesure du PRR à 20% est celle qui offre la plus grande discrimination sur tous les points, dans le pire cas, la mesure donnait

une valeur 130 fois plus grande pour un pic de corrélation que pour une corrélation croisée avec l'application des filtres 3x3. Les filtres 3x3 de moyenne et médian donnent les mêmes résultats. La discrimination moyenne du PRR à 20% est 9% plus petite avec l'application du filtre de moyenne 5x5 qu'avec le 3x3.

La mesure de la pente du pic a été dans certain cas peu améliorée et dans d'autres empirée avec l'application des différents filtres. La mesure de la pente du pic demeure peu discriminante même avec l'application des filtres.

Le filtrage des images a eu un effet positif sur la discrimination du moment à 20%. Le filtrage avec les filtres 3x3 affecte très peu les résultats sur les pics de corrélation mais a eu plus d'impact sur les corrélations croisées. Le bruit dans les images a plus d'influence sur les mesures appliquées aux corrélations croisées, ce qui est causé par les plus faibles niveaux d'intensité rencontrés. Le filtrage a eu pour impact de réduire ce bruit et par conséquent d'allonger les rayons résultant des corrélations croisées. Le filtrage a aussi eu pour effet d'homogénéiser les valeurs obtenues dans chaque groupe : pic de corrélation et corrélation croisée. Le filtrage a eu des conséquences très positives sur cette mesure en augmentant de 7,8 fois la pire discrimination et de 1,4 fois la discrimination moyenne avec le filtre de moyenne 3x3. Le filtre médian a amélioré de 8,2 fois la pire discrimination et de 1,5 fois la discrimination moyenne. Les deux filtres 3x3 offrent des performances très semblables. Le filtre de moyenne 5x5 donne de moins bonnes performances que le 3x3.

2.10 L'algorithme de post-traitement

À la lumière de ces résultats, il est maintenant possible de choisir judicieusement les mesures qui répondent à l'objectif visé, soit de discriminer les vrais pics de corrélation des faux. Les gains en performance de certaines mesures rencontrés dans les dernières sections justifient l'utilisation d'un filtre sur les plans de corrélation. Les meilleures mesures sont le PRR et le moment à 20% avec l'application d'un filtre 3x3. Même si le PCE offre une faible discrimination, il est conservé afin de donner un indice sur la distorsion d'un objet. Comme le filtre de moyenne 3x3 offre les mêmes performances que le médian, le filtre de moyenne est choisi à cause de sa plus faible complexité d'implantation. Le filtre médian serait composé de plusieurs paires de comparateurs-multiplexeurs, le tout agencé en un arbre de tri afin d'obtenir un classement croissant ou décroissant des pixels de la fenêtre de convolution. Le filtre de moyenne n'est constitué que d'additionneurs formant un arbre de sommation.

La figure 2.7 montre l'algorithme choisi pour le post-traitement du corrélateur optique. Une caméra achemine les plans de corrélation digitalisés au système qui les filtre avec le filtre de moyenne 3x3. Ensuite, la prise de mesures s'effectue sur chaque image. Dans une première approche, ce ne sont que les valeurs entières des mesures qui sont retenues. Ceci diminu la complexité du traitement en temps réel en ne calculant pas les résultats en virgule flottante pendant l'analyse de l'image. Ces valeurs entières sont celles permettant le calcul des mesures du PRR, du moment et du PCE. Un compromis nous permet de diminuer la

complexité du traitement. Il s'agit de calculer le PRR et le moment à 25% du pixel le plus intense. Ceci est obtenu en effectuant un décalage de 2 bits vers la droite ce qui revient à faire une division entière par 4. De plus, la position du pixel le plus intense de l'image est conservée. Lorsque toute l'image est reçue, le calcul des mesures est finalisé avec une précision virgule flottante.

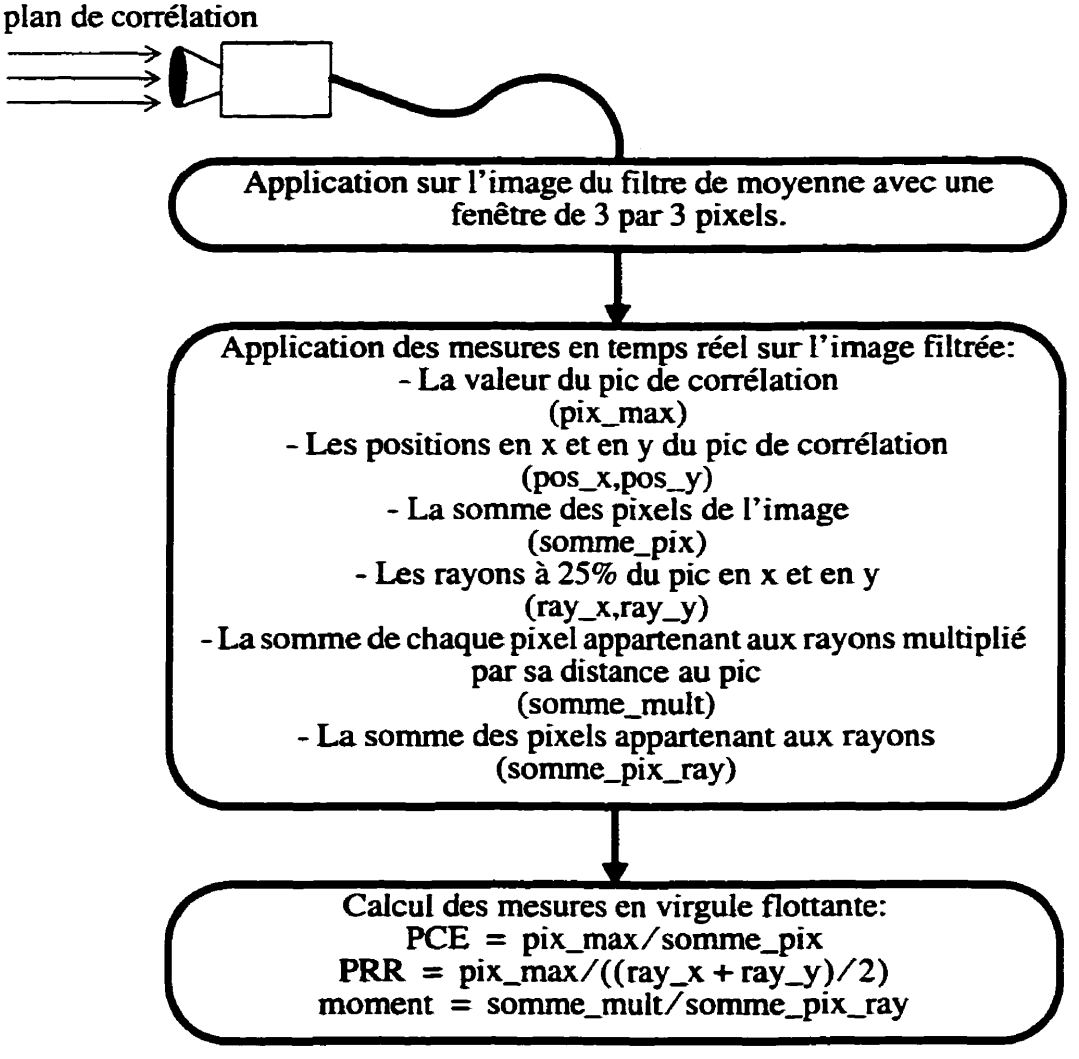

Figure 2.7 L'algorithme de post-traitement du corrélateur optique

CHAPITRE 3

LES MISES EN OEUVRE

Ce chapitre présente en détail les différentes mises en oeuvre de l'algorithme de post-traitement du corrélateur optique défini dans le dernier chapitre. Trois implantations effectuant exactement le même traitement ont été réalisées. La première a été écrite en langage C, la deuxième dans le langage assembleur du TMS320C40 et la troisième est une version logicielle/matérielle réalisée avec un mélange de langages C et VHDL. Ces trois mises en oeuvre constituent une étude qui nous permettra de faire des relations entre les performances, les coûts et le temps de réalisation des différentes implantations.

3.1 La plate-forme reconfigurable

La plate-forme reconfigurable est un support qui permet la réalisation d'applications partitionnées en une partie logicielle et en une partie matérielle dédiée. Cette plate-forme est reliée à un PC, système hôte, par une carte mère de la compagnie Spectrum, la carte QPC40. Cette carte mère possède 4 sites TIM (Texas Instrument Module), sur lesquels deux modules TIM viennent s'attacher : une carte DSP pour l'exécution de la partie logicielle et une carte coprocesseur DSP reconfigurable pour l'implantation de la partie matérielle dédiée. Ces 2 modules TIM sont reliés entre eux par 3 ports asynchrones.

3.1.1 La carte DSP

La carte DSP (MDC40S) de la compagnie Spectrum est composée d'un processeur DSP virgule flottante, le TMS320C40, et de 3 banques de mémoire vive de 128K mots de 32 bits sans aucun état d'attente. Le TMS320C40 est un processeur parallèle qui à la base a une fréquence d'opération de 25 MHz et qui est capable d'effectuer une multiplication virgule flottante ou entière en un cycle. Il possède 6 ports de communication bidirectionnels pouvant faire des échanges de données à un rythme de 20Moctets/sec et 6 coprocesseurs DMA pour effectuer des transferts de données sans la supervision du CPU.

3.1.2 La carte coprocesseur DSP reconfigurable

La carte X-CIM de la compagnie MiroTech permet l'implantation de la partie matérielle dédiée d'un algorithme partitionné. L'utilisateur dispose de deux processeurs virtuels, VPE1 et VPE2, pour réaliser la fonction qu'il désire. Ces deux VPE sont des FPGA XC4013E-3 du manufacturier Xilinx, ce qui permet la réalisation de circuits ayant une complexité maximale de 26 000 portes. Chaque VPE est connecté à une paire de FIFO d'entrée et de sortie afin de réaliser des communications asynchrones entre le DSP et les processeurs virtuels. Ces FIFOs sont situés dans le module de communication (CPM), voir figure 3.1. Les deux VPE peuvent accéder entièrement aux 4 banques de mémoire vive de la carte, soit 128K mots de 64 bits. De plus, ces banques sont divisées en deux espaces d'adressage indépendants, banque 0 et 1 et banque 2 et 3. Le module du filtre de moyenne de l'algorithme de post-traitement du corrélateur optique a utilisé avec avantage cette

architecture mémoire. Le module de configuration (CSU) est principalement responsable de fournir deux horloges configurables aux processeurs virtuels.

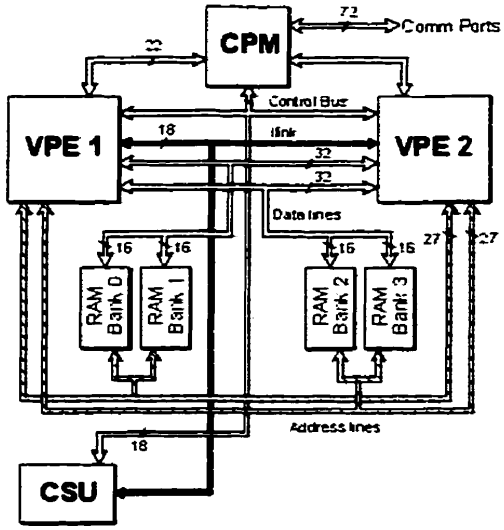

Figure 3.1 L'architecture du coprocesseur DSP reconfigurable
(reproduit avec la permission de MiroTech)

3.2 La mise en oeuvre C

Cette mise en oeuvre traduit directement en langage C l'algorithme de post-traitement du corrélateur optique et fut implantée sur la carte DSP.

3.2.1 L'algorithme de post-traitement du corrélateur optique

Cette section présente plus en détail l'algorithme de post-traitement du corrélateur optique tel qu'il a été programmé dans les deux mises en oeuvre logicielles. La variable PIX_MAX représente le pixel de plus grande valeur, VAL_LIM vaut 25% de la valeur de PIX_MAX et représente la condition d'arrêt de traitement en x et en y. POS_X et POS_Y sont

respectivement la position en x et la position en y du pixel le plus intense. SOMME_PIX représente la somme totale de tous les pixels de l'image, TRAIT_X et TRAIT_Y sont des drapeaux pour signaler s'il y a un traitement actif en x ou en y pour les mesures s'appliquant aux rayons du pic de corrélation comme le PRR et le moment, RAY_X et RAY_Y représentent les rayons en x et en y à 25% de la valeur du pic de corrélation (VAL_LIM). SOMME_MULT est la somme des pixels appartenant aux rayons multipliés par leur distance au pic détecté et SOMME_PIX_RAY est la somme de ces pixels. Les calculs sont effectués pour les pixels appartenant aux rayons à la droite et au dessous du pic détecté. Le rapport de ces deux dernières mesures donne la valeur du moment du pic de corrélation. IMAGE est un tableau contenant le plan de corrélation dans lequel les pixels sont groupés selon les lignes en paquets de 4 sur des mots de 32 bits, TAMPON est un tableau contenant les pixels séparés des 3 lignes actives servant à la formation des pixels filtrés d'une ligne, PIX_COURANT conserve la valeur du pixel filtré courant, PIXEL et LIGNE indiquent respectivement la position en x et en y du pixel filtré courant. Sachant que le C40 ne peut effectuer directement une division en virgule flottante la stratégie proposée par Texas Instrument fut appliquée. Elle consiste à calculer l'inverse du diviseur en virgule flottante et de multiplier cet inverse au numérateur de la division. La sous-routine d'inversion proposée par T.I. prend 7 cycles et la multiplication un seul cycle. Cette routine d'inversion fait appel à une instruction, RCPF, qui calcule directement une approximation sur 16 bits de la mantisse du nombre en virgule flottante à inverser. Des divisions avec une précison en virgule flottante sont nécessaires pour le calcul du PRR, du moment et du PCE.

```
- PIX_MAX=POS_X=POS_Y=SOMME_PIX=TRAIT_X=TRAIT_Y=0
- sépare paquets de 4 pix de 1ère ligne de IMAGE et met dans 1ère ligne de TAMPON[][]
- sépare paquets de 4 pix de 2ème ligne de IMAGE et met dans 2ème ligne de TAMPON[][]
- Pour LIGNE=3 à 480
    |- sépare paquets de 4 pix de LIGNE de IMAGE et met dans (LIGNE % 3) de TAMPON[][]
    |- Pour PIXEL=2 à 639
    |-   |- PIX_COURANT=0
    |-   |- Pour m=1 à 3
    |-   |-   |- Pour n=-1 à 1
    |-   |-   |-   |- PIX_COURANT=PIX_COURANT + TAMPON[m][PIXEL+n]
    |-   |-   |- Fin Pour
    |-   |- Fin Pour
    |-   |- PIX_COURANT = PIX_COURANT / 8
    |-   |- SOMME_PIX = SOMME_PIX + PIX_COURANT
    |-   |- Si PIX_COURANT > PIX_MAX alors
    |-   |-   |- PIX_MAX = PIX_COURANT
    |-   |-   |- POS_X = PIXEL
    |-   |-   |- POS_Y = LIGNE-1
    |-   |-   |- RAY_X = RAY_Y = -1
    |-   |-   |- VAL_LIM = PIX_MAX / 4
    |-   |-   |- SOMME_PIX_RAY = SOMME_MULT = 0
    |-   |-   |- TRAIT_X = TRAIT_Y = 1
    |-   |- Fin Si
    |-   |- Si TRAIT_X=1 et que POS_Y=LIGNE-1 alors
    |-   |-   |- Si PIX_COURANT > VAL_LIM alors
    |-   |-   |-   |- RAY_X = RAY_X +1
    |-   |-   |-   |- Si RAY_X != 0 alors
    |-   |-   |-   |-   |- SOMME_PIX_RAY = SOMME_PIX_RAY + PIX_COURANT
    |-   |-   |-   |-   |- SOMME_MULT = SOMME_MULT + (RAY_X * PIX_COURANT)
    |-   |-   |-   |- Fin Si
    |-   |-   |- Autre
    |-   |-   |-   |- TRAIT_X = 0
    |-   |-   |- Fin Si
    |-   |- Fin Si
    |-   |- Si TRAIT_Y=1 et que POS_X=PIXEL alors
    |-   |-   |- Si PIX_COURANT > VAL_LIM alors
    |-   |-   |-   |- RAY_Y = RAY_Y +1
    |-   |-   |-   |- Si RAY_Y != 0 alors
    |-   |-   |-   |-   |- SOMME_PIX_RAY = SOMME_PIX_RAY + PIX_COURANT
    |-   |-   |-   |-   |- SOMME_MULT = SOMME_MULT + (RAY_Y * PIX_COURANT)
    |-   |-   |-   |- Fin Si
    |-   |-   |- Autre
    |-   |-   |-   |- TRAIT_Y = 0
    |-   |-   |- Fin Si
    |-   |- Fin Si
    |- Fin Pour
- Fin Pour
- RAYON_INV = 1 / ((RAY_X + RAY_Y) * 0.5)
- PRR = PIX_MAX * RAYON_INV
- MOMENT = SOMME_MULT * (1 / SOMME_PIX_RAY)
- PCE = PIX_MAX * (1 / SOMME_PIX)
```

L'image filtrée est de taille réduite par rapport à l'image originale (voir section 2.2). La valeur du pixel filtré (PIX_COURANT) est limitée à la valeur maximale d'un pixel, soit 255. Cette limitation est nécessaire car il serait possible d'obtenir une valeur plus grande que 255 à cause de la division par huit.

3.3 La mise en oeuvre assembleur

L'assembleur est le langage machine d'un processeur. Par ses mnémoniques, il fait appel directement aux différentes instructions implantées sur un processeur : le chargement d'un registre, une addition, une écriture mémoire, un branchement, ... Ainsi, l'assembleur étant un language de plus bas niveau que le C, il est possible de réaliser une mise en oeuvre plus performante d'un algorithme au prix d'un temps de réalisation plus long. L'algorithme de post-traitement du corrélateur optique décrit à la section précédente a été réalisé avec l'assembleur du TMS320C40 et cette réalisation a été exécutée sur la carte DSP.

3.4 La mise en oeuvre logicielle/matérielle

Le but de cette mise en oeuvre est de caractériser les avantages d'une implantation mixte logicielle/matérielle. Cette réalisation mixte effectue exactement le même traitement que les deux mises en oeuvre logicielles. Ceci a été rendu possible par le fait que nous avons défini l'algorithme en prévision d'une implantation mixte en évitant d'y introduire des traitements irréalisables de façon causale ou trop coûteux.

3.4.1 Partition de l'algorithme de post-traitement

La séparation des opérations logiques entre les parties logicielle et matérielle fut accomplie avec l'objectif de minimiser les communications entre elles, tout en équilibrant leurs temps d'exécution respectifs. La partie matérielle dédiée tire sa performance de l'exécution en parallèle d'opérations faites sur des unités de traitement réalisées sur mesure. Ainsi, les modules de l'algorithme permettant une telle exploitation du parallélisme se retrouvent, lorsque les ressources le permettent, sur la carte coprocesseur DSP reconfigurable. De plus, le système de post-traitement logiciel/matériel peut tirer avantage de la flexibilité et du faible coût d'une implantation logicielle pour les opérations moins couramment utilisées. Par ailleurs, toutes les opérations avec une précision en virgule flottante comme la multiplication sont avantageusement déléguées au DSP car il possède des unités de traitement dédiées à cette tâche.

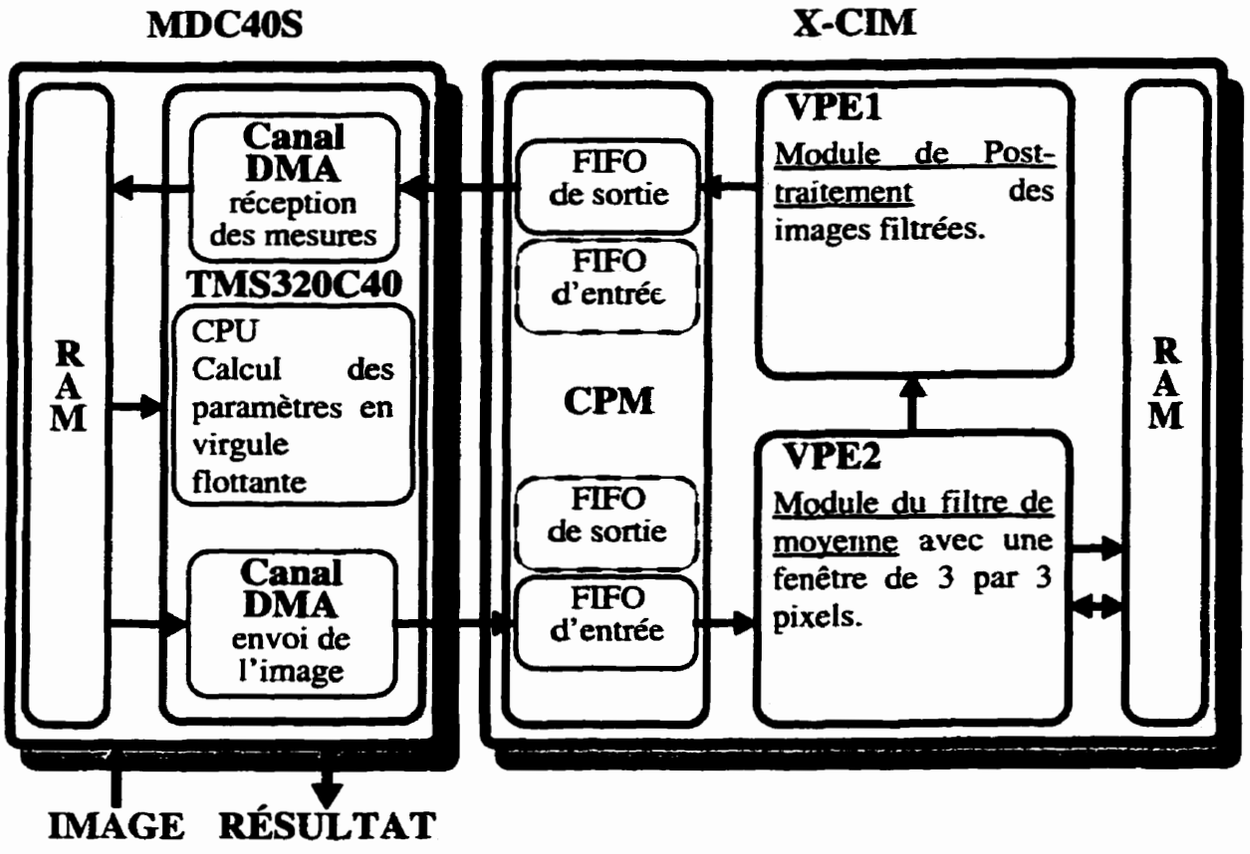

Figure 3.2 Partition de l'algorithme de post-traitement sur la plate-forme reconfigurable.

La partition de l'algorithme de post-traitement du corrélateur optique est illustrée à la figure 3.2. L'image du plan de corrélation est acheminée de la mémoire du C40 au FIFO d'entrée du VPE2 contenant le module du filtre de moyenne par un canal DMA. L'envoi de l'image est non-entrelacé par paquets de 4 pixels : les pixels ont une résolution de 8 bits et les communications entre les parties logicielle et matérielle se font sur 32 bits. Le VPE2 filtre l'image reçue et envoie le premier pixel filtré au VPE1 quelques cycles après qu'il ait obtenu du C40 le premier paquet de pixels de la troisième ligne de l'image. Cette dernière condition permet la formation de la première fenêtre de 3 par 3 pixels de l'image. Ensuite,

le VPE1 applique ses mesures au fur et à mesure que les pixels filtrés lui sont acheminés. L'échange des pixels filtrés du VPE2 au VPE1 se fait sur 8 bits avec en plus, un bit servant de drapeau afin d'indiquer la validité de la donnée. Cette méthode de communication inter-VPE fut choisi afin de réduire la complexité de la logique étant donnée la nature asynchrone des communications avec le C40.

Le VPE1 envoie au C40 les mesures appliquées sur l'image filtrée tout de suite après avoir reçu le dernier pixel filtré de l'image courante. Finalement, le C40 calcule le PCE, le PRR et le moment du plan de corrélation filtré avec une précision virgule flottante à partir des mesures obtenues du VPE1.

Afin de vérifier la faisabilité de cette partition, une étude a dû être faite quant au coût des communications lors de l'envoi de l'image au VPE2. Les différentes images de la banque des plans de corrélation ont une résolution de 640 par 480 pixels. Le système doit traiter 30 images par seconde, ce qui fixe un taux d'échange minimal de :

$$(\qquad)\text{pixels/image} \cdot 30\text{images/sec} = 9\,216\,000\text{pixels/sec}$$

Les ports du C40 peuvent effectuer des transferts jusqu'à 20M Octets/sec. Les pixels étant codés sur 8 bits, l'envoi des images n'introduira donc aucun retard. Le port de communication permettrait même des images comportant 2 fois plus de pixels ou un traitement au double du débit actuel.

3.4.2 La méthodologie de conception de la partition matérielle

Les deux modules, VPE1 et VPE2, de l'algorithme de post-traitement ont été conçus selon une méthodologie de conception haut niveau (Figure 3.3). Ces deux FPGA ont été programmés à partir d'une description VHDL comportementale. Une simulation fonctionnelle du VHDL a été exécutée avec CoreSpex. CoreSpex est un modèle VHDL qui contient le comportement temporel de la carte X-CIM. La synthèse a été effectuée avec Synopsys et le placement/routage avec Xilinx M1.3. Une description VHDL structurelle décrivant les modules fut créée par les outils de Xilinx afin de faire une simulation après placement/routage avec CoreSpex pour la validation finale des modules.

Le principal problème fut le manque de documentation de la compagnie MiroTech sur la démarche complète et en détail pour le fonctionnement de la carte. De plus, l'environnement CoreSpex n'était pas adapté au simulateur de Synopsys lorsque nous avons commencé le projet. Beaucoup de déverminage a du être fait en ce sens lors de plusieurs essais infructueux. Il nous aurait été impossible de réaliser ce projet avec seulement la documentation fournie. C'est un lien direct avec cette compagnie qui nous a permis d'obtenir le fonctionnement de la carte X-CIM.

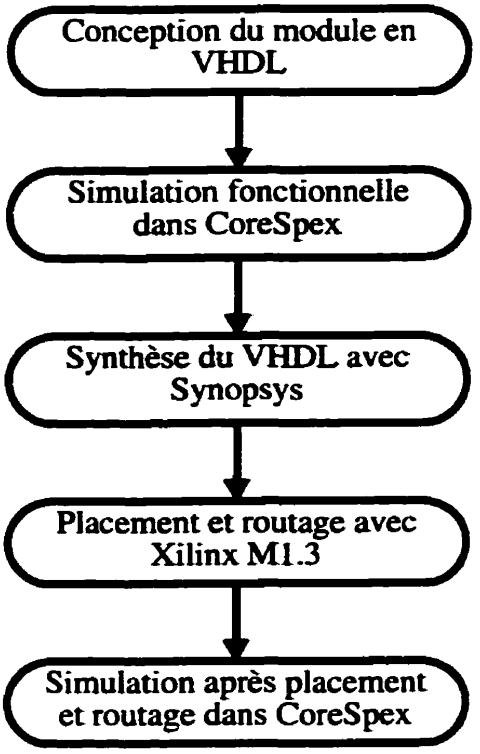

Figure 3.3 Méthodologie de conception de la partition matérielle

3.4.3 Le module du filtre de moyenne

Ce module est responsable de l'application d'un filtre de moyenne avec une fenêtre de 3 par 3 pixels sur les plans de corrélation obtenus à la sortie du corrélateur optique afin d'augmenter la discrimination des mesures entre les vrais et les faux pics de corrélation. Ce module, illustré à la figure 3.4, est composé de plusieurs sous-modules. Une machine à états gère les communications avec le FIFO d'entrée et une autre avec celles de la mémoire vive (RAM), une dernière machine à états, le contrôleur, régit l'interaction des différents sous-modules lors du traitement d'une image complète. Le compteur de lignes assure le déclenchement du processus de fin de traitement d'une image et le compteur d'adresses

produit la bonne séquence d'adresses mémoire pour la formation des fenêtres du filtre. Le registre à décalage forme les fenêtres de 9 pixels à partir du chargement des paquets de pixels courants et les achemine à l'unité de sommation qui envoie le résultat à une unité qui effectue une division entière par 8. La sortie de ce dernier module donne un pixel filtré.

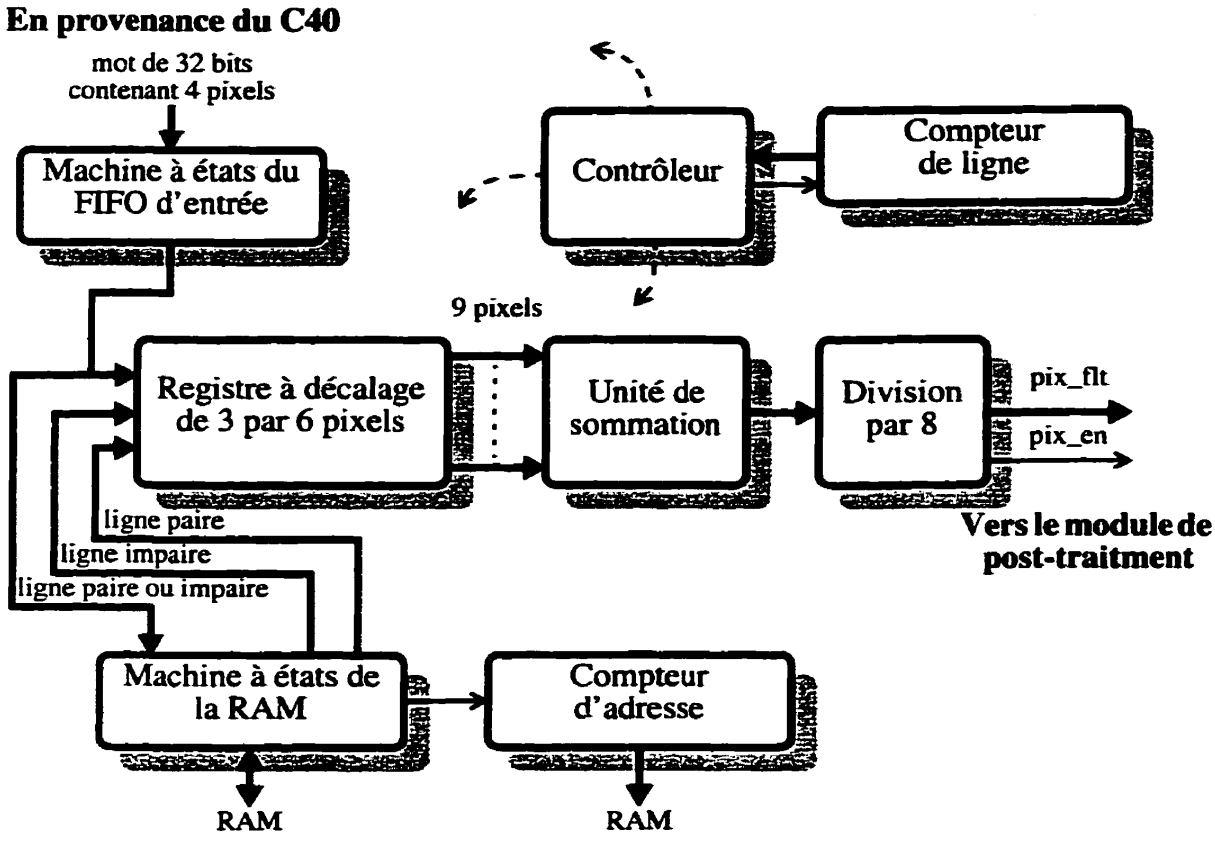

Figure 3.4 Architecture du filtre de moyenne avec une fenêtre de 3 par 3 pixels.

3.4.3.1 Le contrôleur du filtre de moyenne

La séquence de contrôle présentée à la figure 3.5 demeure peu complexe. Tout d'abord, il s'agit d'écrire en mémoire les deux premières lignes de l'image dans leurs espaces d'adressage respectifs : ligne paire et ligne impaire. Ces deux lignes sont essentielles à la formation des premières fenêtres du filtre. La réception du premier paquet de 4 pixels de la troisième ligne vidéo fait basculer le contrôleur dans une boucle de 2 états qui prendra fin lors de la création du dernier pixel filtré de l'image. Le premier état de cette boucle attend l'arrivée d'un nouveau paquet de 4 pixels en provenance du FIFO et lorsque ces données sont disponibles, il les charge avec les 2 paquets (lignes paire et impaire) venant de la mémoire vive de la carte. Ensuite, le dernier état effectue 4 décalages du registre, ce qui produit 4 pixels filtrés. Le chargement du premier mot d'une ligne produit seulement 2 pixels filtrés. Finalement, lorsque toute l'image a été filtrée, tout le module se remet en état d'attente de la réception d'une nouvelle image.

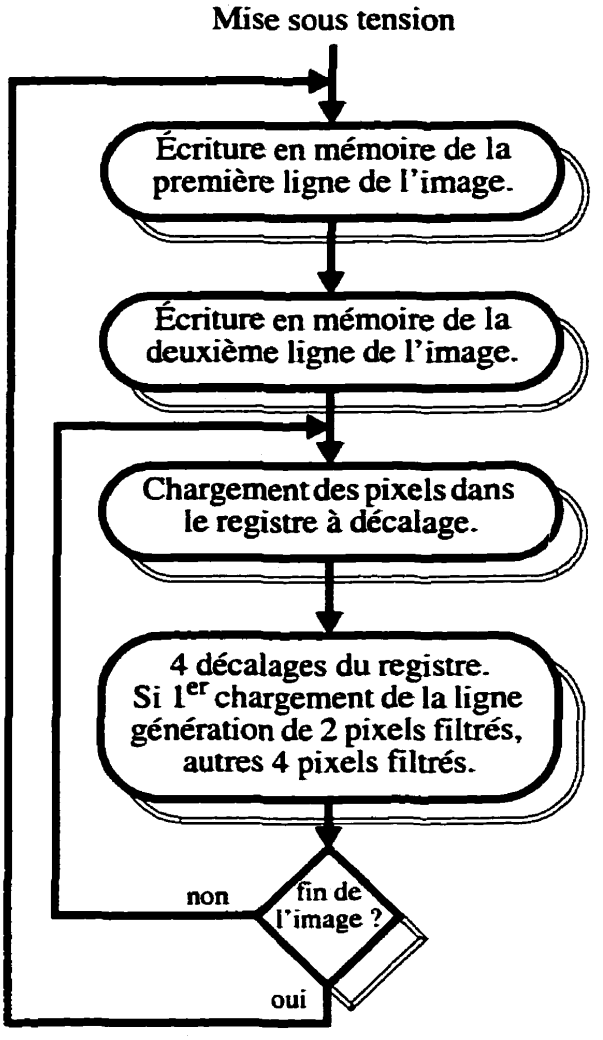

Figure 3.5 Diagramme d'états du contrôleur du filtre de moyenne.

3.4.3.2 Le registre à décalage

L'existence et l'architecture du registre à décalage sont une conséquence de deux facteurs: le recouvrement entre les paquets adjacents de 4 pixels pour la formation des fenêtres lors du filtrage et le nombre de pixels par paquet. Dans l'équation:

$$\text{profondeur}_{min} = \text{largeur}_{fen} + (\quad\quad)$$

profondeur_{min} représente la profondeur minimale du registre, largeur_{fen} la largeur de la fenêtre du filtre, pixels_{pqt} le nombre de pixels par paquet lors du chargement et le -1 prend en considération le recouvrement minimal entre la fenêtre du filtre et le nouveau paquet de pixels. Dans ce projet, la profondeur minimale du registre à décalage est de $3 + (4 - 1) = 6$.

La figure 3.6 présente l'architecture du registre à décalage avec profondeur minimale telle que réalisée pour la partition matérielle. L'utilisation de multiplexeurs est nécessaire afin de faire circuler les données précédemment chargées ou de charger le nouveau paquet de pixels en provenance du FIFO d'entrée et ceux de la mémoire vive. La sélection de ces multiplexeurs est opérée par le contrôleur. La sortie du registre à décalage est acheminée à l'unité de sommation.

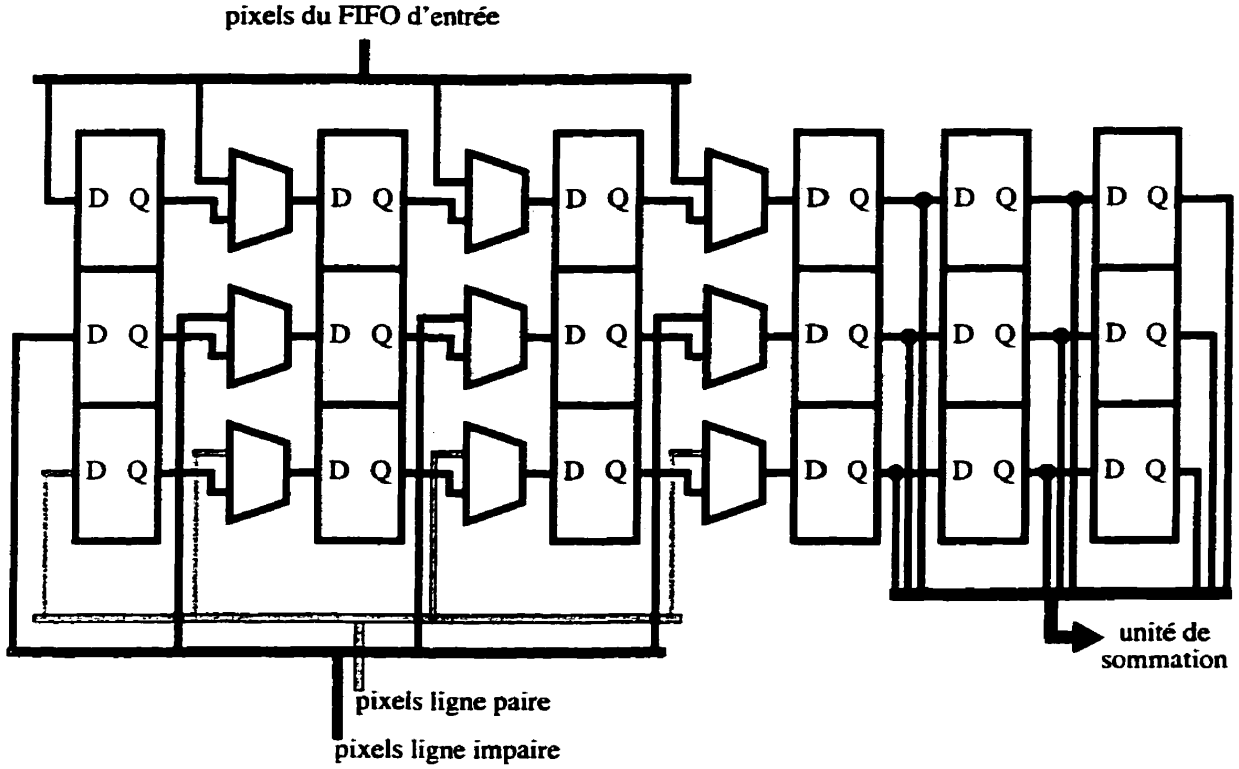

Figure 3.6 Architecture du registre à décalage.

3.4.3.3 L'interface du filtre de moyenne avec la RAM

La figure 3.7 illustre l'interface du module du filtre de moyenne avec la mémoire vive de la carte. Cette mémoire possède, tout au long du filtrage, l'équivalent de 2 lignes vidéo qui précèdent le nouveau paquet de 4 pixels en provenance du FIFO d'entrée. Cette quantité de pixels emmagasinés correspond au minimum requis pour la création des fenêtres de 3 par 3 pixels. Une partition de la mémoire a été effectuée afin d'accéder indépendamment en écriture soit à une ligne paire ou à une ligne impaire. Il y a 2 types de fenêtres lors du filtrage: une configuration lignes paire-impaire-paire et impaire-paire-impaire. Ainsi peu

importe la catégorie, ligne paire ou impaire, des nouveaux pixels, la configuration de la fenêtre nécessite toujours d'ajouter deux paquets venant d'une ligne paire et d'une ligne impaire. En plus d'être chargé immédiatement dans le registre à décalage, chaque nouveau paquet de 4 pixels en provenance du FIFO d'entrée est écrit dans l'espace mémoire de sa catégorie. Après avoir commandé l'écriture en mémoire des 2 premières lignes de l'image, le contrôleur permet à la machine à états de la RAM de parcourir 4 états tout au long du filtrage. Le premier lit et charge dans des registres intermédiaires les deux paquets de 4 pixels, lignes paire et impaire, qui coïncideront avec le prochain paquet venant du FIFO d'entrée. Deuxièmement, la machine à états attend ces 4 nouveaux pixels pour les charger dans le registre d'écriture de la ligne paire ou impaire. La figure 3.7 montre un cas où les 4 pixels en provenance du C40 appartiennent à une ligne impaire. À ce moment, le contrôleur charge le registre à décalage. Le troisième état permet l'écriture du nouveau paquet dans un des deux espaces d'adressage. Le dernier état incrémente le compteur circulaire d'adresse et à la fin de chaque cycle, 0 à 159, l'espace mémoire d'écriture est interverti.

La figure 3.8 illustre le fonctionnement des trois machines à états du module du filtre de moyenne.

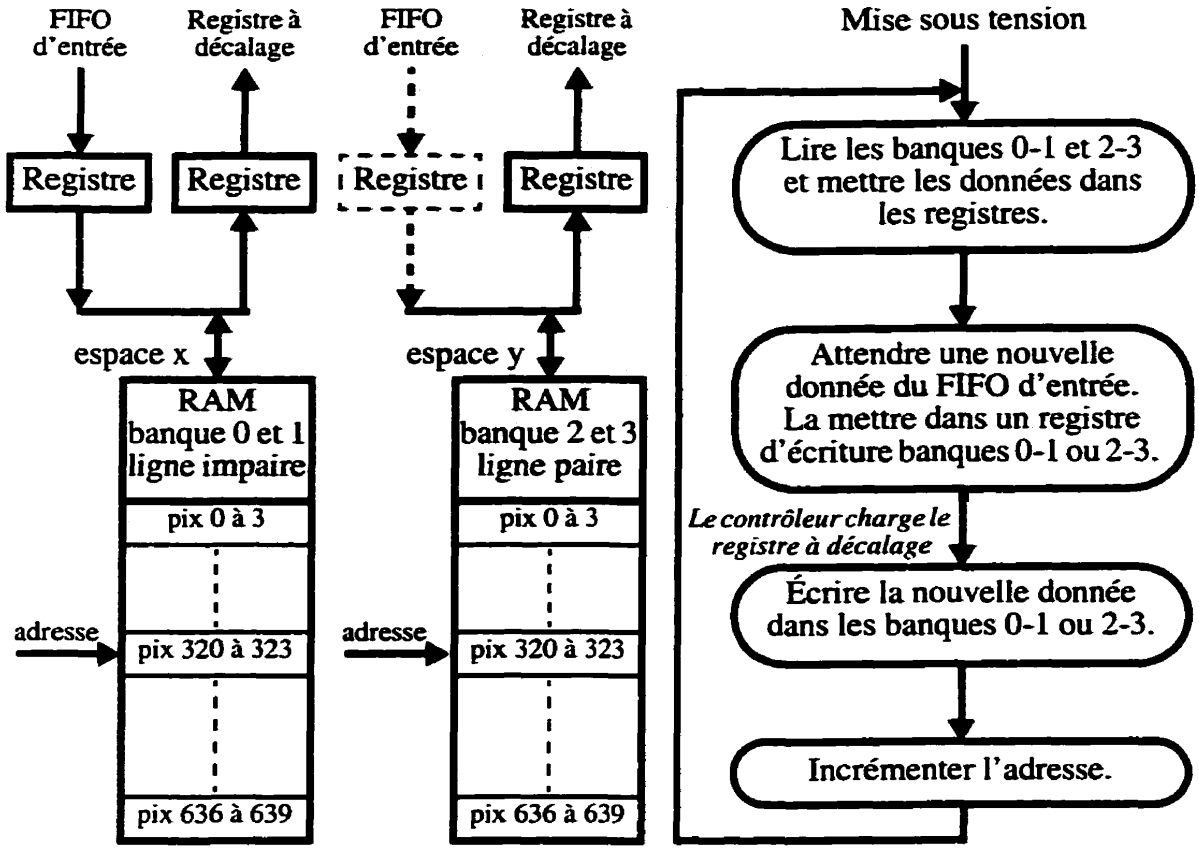

Figure 3.7 Fonctionnement de l'interface du filtre de moyenne avec la RAM.

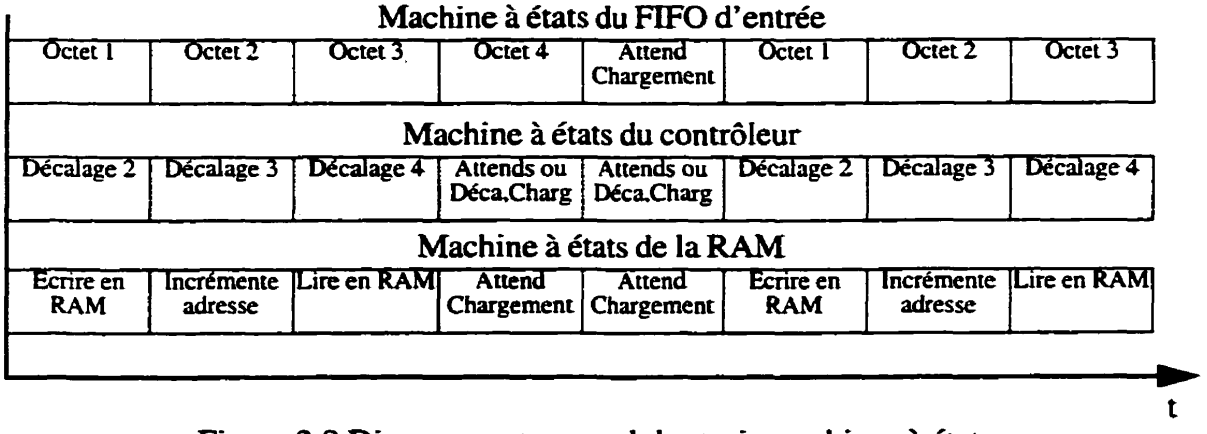

Figure 3.8 Diagramme temporel des trois machines à états.

3.4.4 Le module de post-traitement des images filtrées

Ce module est responsable de la prise de mesures d'évaluation de pic de corrélation sur les plans de corrélation précédemment filtrés. Plusieurs petits sous-modules et deux machines à états composent ce processeur virtuel : une machine à états pour le FIFO de sortie et une autre pour le contrôleur. La figure 3.9 présente un schéma complet de l'architecture de ce module qui a pour entrée pix_flt, le pixel filtré et pix_en le signal de contrôle. À la fin du plan de corrélation filtré, ce module envoie au C40 les mesures obtenues sur l'image. Tous les noms des signaux de la figure 3.9 sont les mêmes que ceux des variables de l'implantation en C (section 3.2.1), sauf pour les nouveaux signaux.

Deux regroupements de 3 sous-modules identiques servent aux positions en x et en y. Chacun est composé d'un compteur, d'un registre et d'un comparateur d'égalité. Les deux compteurs possèdent les positions du pixel filtré courant, pix_flt, et indiquent au contrôleur la fin de l'image. Les registres, position x et y du pic, capturent la localisation du pixel filtré le plus intense : pix_max. Cette capture est commandée par le signal nouv_max. Le comparateur d'égalité pour la position en x signale au module si le pixel filtré courant est situé sur la même colonne vidéo que le pixel le plus intense par le signal colonne_pic. À son tour, le comparateur d'égalité pour la position en y signale si pix_flt est sur la même ligne vidéo que pix_max par le signal ligne_pic.

Les valeurs des rayons sont obtenues par des compteurs. Par exemple, le compteur du rayon en x du pic de corrélation peut s'incrémenter s'il y a eu un nouveau pixel filtré maximal,

nouv_max, si le pixel filtré courant, pix_flt, est positionné sur la ligne de pix_max et si aucun pix_flt de la ligne n'a eu une valeur inférieure à pix_max divisé par 4, petit_pix. Le même processus est appliqué pour le rayon en y.

Un regroupement de 2 sous-modules s'occupe de la capture de pix_max : un registre qui conserve cette valeur et un comparateur "plus grand que" qui génère le signal nouv_max. Le signal nouv_max est actif lorsque pix_flt est plus grand que pix_max et il contrôle la capture des nouvelles valeurs de pix_max, pos_x et pos_y. De plus, nouv_max réinitialise ray_x, ray_y, somme_mult et somme_pix_ray. Ce signal est celui qui a le plus de répercussions sur le fonctionnement de ce circuit car il indique aux différentes unités qui calculent les mesures qu'un nouveau pixel filtré maximal a été rencontré dans le plan de corrélation filtré.

Deux sous-modules, un diviseur par 4 et un comparateur "plus grand que" génèrent le signal petit_pix. Ce signal est actif lorsque le présent pix_flt est plus petit que pix_max divisé par 4. Ceci représente une condition d'arrêt pour les mesures qui s'appliquent aux rayons du pic de corrélation.

Un regroupement de 3 sous-modules s'occupe du calcul de somme_mult. Ce regroupement, le plus complexe du module, est composé d'un multiplexeur, d'une unité de multiplication et d'un accumulateur. Le multiplexeur permet la sélection entre les deux rayons, ray_x et ray_y, et il est contrôlé par les signaux ligne_pic et colonne_pic. Cette

sélection, rayon, est multipliée par pix_flt dans l'unité de multiplication. L'unité de multiplication est pipelinée et peut effectuer toutes les multiplications entières non-signées de 8 bits par 9 bits. Finalement, l'accumulateur conserve et additionne les différentes multiplications des pixels des rayons du pic afin d'obtenir le résultat somme_mult. Un simple accumulateur contrôlé par ligne_pic et colonne_pic, est nécessaire pour la génération de la somme des pixels filtrés appartenant aux rayons : somme_pix_ray. Un dernier accumulateur effectue la somme de tous les pixels de l'image, somme_pix..

3.4.4.1 Le contrôleur du module de post-traitement

Tout ce qui a été décrit dans la section 3.4.4 représente le premier état du contrôleur du module de post-traitement dont le diagramme de transition d'états est illustré à la figure 3.10. Lorsque le dernier pixel filtré de l'image a été reçu, le contrôleur passe à son deuxième état. Cet état permet à l'unité de multiplication, qui est pipelinée, de terminer son traitement sur les dernières multiplications, afin d'obtenir la valeur finale de la somme de chaque pixel appartenant aux rayons multipliés par sa distance au pixel maximal : somme_mult. Au troisième état, les mesures sont prêtes à être envoyées au C40. Cinq mots de 32 bits sont nécessaires pour communiquer les mesures résultantes. La bulle du diagramme d'états montre l'organisation spatiale et temporelle de ces mesures. C'est le contrôleur qui dirige le multiplexeur qui est responsable de présenter au FIFO de sortie les résultats obtenus. Finalement, le dernier état réinitialise tout le module afin qu'il soit prêt à recevoir un nouveau plan de corrélation filtré.

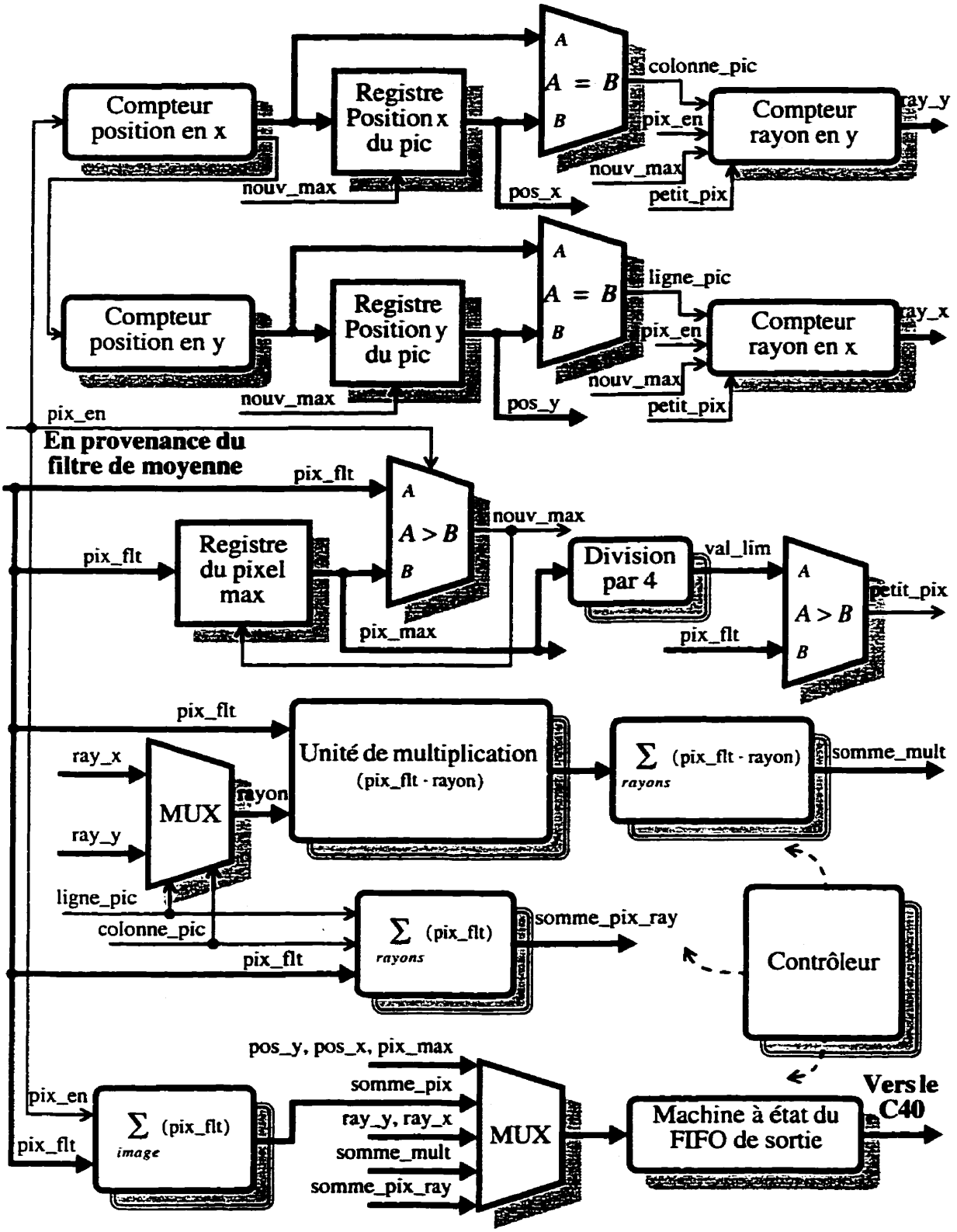

Figure 3.9 Architecture du module de post-traitement.

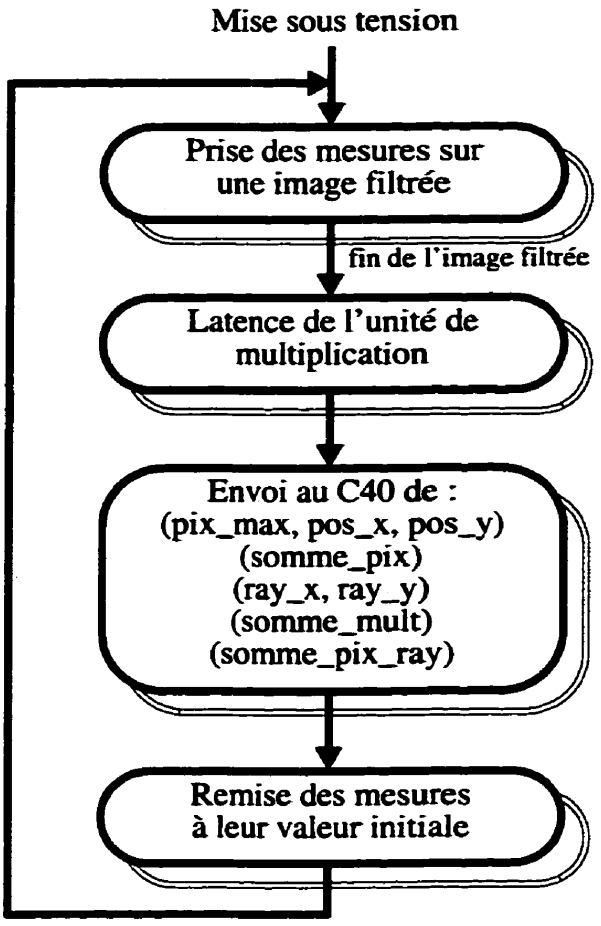

Figure 3.10 Diagramme d'états du contrôleur du module de post-traitement.

3.4.4.2 L'unité de multiplication

Une unité de multiplication de 8 par 9 bits est nécessaire pour le calcul du moment dans le module de post-traitement. Les rayons d'un pic de corrélation sont sur 9 bits chacun et le pixel filtré est sur 8 bits. Le traitement du présent processeur virtuel étant en temps réel, le multiplicateur doit être capable de prendre une nouvelle multiplication à chaque cycle. L'opération à effectuer, une multiplication entière non-signée, est schématisée dans l'exemple qui suit.

										A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1		
								×											
									B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1		
M1=										A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1		si B_1=1 sinon A=0
								+											
M2=									A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0		si B_2=1 sinon A=0
							+												
M3=								A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0	0		si B_3=1 sinon A=0
						+													
M4=							A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0	0	0		si B_4=1 sinon A=0
					+														
M5=						A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0	0	0	0		si B_5=1 sinon A=0
				+															
M6=					A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0	0	0	0	0		si B_6=1 sinon A=0
			+																
M7=				A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0	0	0	0	0	0		si B_7=1 sinon A=0
		+																	
M8=			A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0	0	0	0	0	0	0		si B_8=1 sinon A=0
	+																		
M9=		A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	0	0	0	0	0	0	0	0		si B_9=1 sinon A=0
	R_{17}	R_{16}	R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1		

Le résultat de la multiplication, R, est obtenu en additionnant M1 à M9. Chaque M*x* est généré par un multiplexeur contrôlé par B*x*. Par exemple, si $B_1 = 1$ alors M1 $=A_8A_7A_6A_5A_4A_3A_2A_1$ sinon M1 = 00000000. Ces multiplexeurs ne sont pas illustrés dans la figure 3.11. L'architecture du multiplieur tire profit des additionneurs à propagation rapide de la retenue que l'on peut programmer sur les FPGA de la famille 4000 de Xilinx.

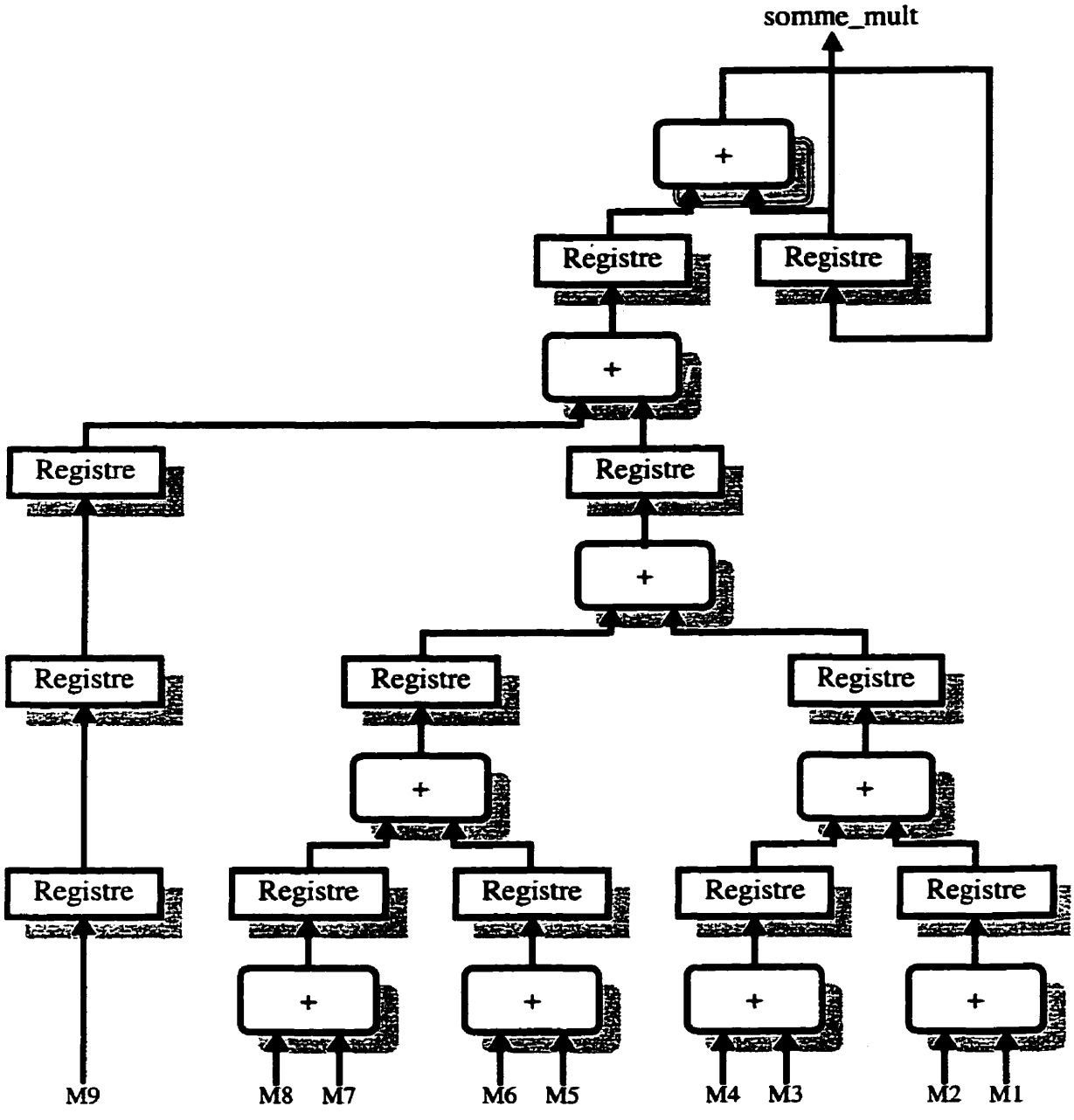

Figure 3.11 Architecture de l'unité de multiplication

Ainsi, en séparant les étages d'additionneurs rapides avec des registres, le multiplicateur ne deviendra pas un chemin critique pour l'obtention d'une fréquence d'opération élevée. De plus, les zéros à la droite des Mx étant le résultat de la multiplication ne sont pas traduit en noeuds dans le circuit. L'architecture tient compte de l'effet d'imbrication des additionneurs. Ainsi, lorsque dans le premier étage M8 et M9 sont additionnés ensemble c'est un additionneur de 8 bits qui est synthétisé et non un additionneur de 16 bits : en effet il est inutile d'additionner les zéros.

3.5 Résultats de synthèse

Le tableau 3.1 présente les résultats de synthèse obtenus avec Synopsys pour les principales unités des deux processeurs virtuels. Les FPGA , des XC4013E-3 du manufacturier Xilinx, comportent 576 CLB chacuns. Les deux modules conçus pouvant être respectivement contenus dans ce type de FPGA, aucun effort n'a du être déployé afin de réduire l'utilisation des CLB : le VPE1 est utilisé à 65% et le VPE2 à 56%.

Les deux FPGA ont une fréquence d'opération maximale de 25 MHz. Il ne servirait à rien de pousser davantage cette limite car le port de communication reliant le C40 à la carte X-CIM échange les données à une fréquence maximale de 20 MHz. Augmenter la fréquence des VPE les ferait consommer plus vite des données qui ne peuvent arriver plus rapidement.

Tableau 3.1 Complexité des différentes unités

Unité	CLB
Machine à états du FIFO d'entrée	23
Unité de sommation	103
Compteur d'adresse	11
Contrôleur + autres logiques	52
Machine à états de la RAM	5
Registre à décalage	108
Compteur de ligne	5
total VPE2	**324**
Multiplicateur	104
Machine à états du FIFO de sortie	24
Contrôleur + autres logiques et arithmétiques	245
total VPE1	**373**

3.6 Comparaison des performances

Pour chacune des trois mises en oeuvre, les temps de traitement pour un plan de corrélation ont été mesurés sur la plate-forme reconfigurable. La mesure des temps d'exécution fut effectuée de la même façon pour les trois implantations. Deux points d'arrêt furent insérés dans la partie logicielle et grâce à l'émulateur du C40 il était possible d'obtenir le nombre

de cycle du C40 entre ces deux points. Le tableau 3.2 présente ces résultats. Le gain de performance obtenu en codant en assembleur l'algorithme de post-traitement est de 3,5 par rapport à l'implantation logicielle en langage C. La mise en oeuvre logicielle/matérielle traite un plan de corrélation 35 fois plus vite que celle en assembleur et 121 fois plus vite que celle en langage C. Les deux implantations logicielles ne peuvent pas traiter un débit d'images en temps réel de 30 images par seconde. La mise en oeuvre logicielle/matérielle traite les images à une cadence deux fois plus rapide que nécessaire pour un flot d'images en temps réel. De plus, la performance de la version codesign est limitée par la vitesse à laquelle le C40 peut fournir les pixels à la partie matérielle, car les FPGA consomment les données dès qu'elles ont été reçues. Une limitation d'outils ne nous a pas permis de trouver la source d'une erreur. La valeur du pixel maximal et la somme des pixels dans l'image diffèrent de quelques unités. Malheureusement, Synopsys ne permettait pas d'inclure le fichier contenant les délais des noeuds du circuit.

Tableau 3.2 Les temps d'exécution des 3 mises en oeuvre

Mise en oeuvre	Temps d'exécution
langage C	1,8690 sec/image
assembleur C40	0,5387 sec/image
logicielle/matérielle	0,0154 sec/image

CONCLUSION

Dans ce mémoire, nous avons introduit les connaissances nécessaires pour une compréhension générale des corrélateurs optiques. Nous avons exploré diverses mesures de pic de corrélation afin d'obtenir un algorithme de post-traitement pour un corrélateur optique. L'objectif de cet algorithme est de départager les pics de corrélation des corrélations croisées d'un flot d'images en temps réel. La réalisation de trois différentes mises en oeuvre du même algorithme de post-traitement nous a permis de faire une étude comparative quant aux performances obtenues par ces dernières.

Le premier chapitre débutait en traitant de la transformée de Fourier, de la corrélation de signaux et de la propriété des lentilles sphériques convergentes à effectuer une transformée de Fourier en deux dimensions. Toutes ces notions permettaient au lecteur de comprendre la description du fonctionnement d'un corrélateur optique Vander Lugt et certains types de filtres optiques utilisés par ces derniers. De plus, les modulateurs spatiaux de lumière ont été décrits dans leur rôle de support physique des filtres optiques. Finalement, une description de certaines mesures de performance des corrélateurs optiques, retrouvées dans la littérature, a été réalisée.

Le deuxième chapitre décrivait l'investigation réalisée dans l'objectif de proposer un algorithme pour départager les pics de corrélation des corrélations croisées. Il est basé sur la comparaison de mesures d'évaluation de pics de corrélation. Premièrement, la banque

des plans de corrélation fournie par l'Institut national d'optique a été décrite et nous avons analysé l'impact du bruit que l'on y retrouve sur les pics de corrélation et les corrélations croisées. Nous avons prouvé que l'application de différents filtres passe-bas améliorait considérablement la discrimination de certaines mesures d'évaluation de pic de corrélation grâce à l'atténuation de ce bruit. Les filtres de moyenne et médian ont été comparés selon les différentes performances obtenues avec les mesures d'évaluation de pic de corrélation. Les mesures qui firent l'objet de l'étude comparative furent : la valeur du pic de corrélation, le PCE (Peak to Correlation Energy), le PRR (Peak to Radius Ratio), la pente du pic de corrélation et le moment. Selon les résultats de cette étude, nous avons été en mesure de concevoir un algorithme de post-traitement d'un corrélateur optique performant. En premier lieu, cet algorithme traite l'image avec un filtre de moyenne de 3 par 3 pixels et par la suite il trouve sur cette image filtrée le pixel le plus intense, le PCE, le PRR et le moment à 25%.

Le troisième chapitre présentait les trois mises en oeuvre de l'algorithme de post-traitement qui ont été réalisées sur la plate-forme reconfigurable. Cette plate-forme est principalement composée d'une carte avec un processeur DSP et d'une carte avec deux FPGA. Elle permet la partition d'une application en une partie logicielle et une partie matérielle. Des mises en oeuvre en langage C, en assembleur du C40 et une logicielle/matérielle ont été réalisées sur la plate-forme reconfigurable. Les deux mises en oeuvre logicielles ont été décrites. Par la suite, la partition, la méthodologie de conception et les différents modules de la mise en oeuvre logicielle/matérielle furent exposés. La synthèse de la partie matérielle a donné

d'excellents résultats : une horloge système maximale de 25 MHz et une occupation des CLB du premier FPGA à 65% et du deuxième à 56%. Ainsi, il serait inutile de synthétiser une horloge plus rapide pour les deux FPGA, car à 25 MHz le goulot d'étranglement pour le temps d'exécution selon cette partition est causé par le taux d'échange des données à travers les ports du C40. Ce chapitre concluait sur les performances respectives des différentes mises en oeuvre. Des trois implantations réalisées, c'est seulement l'implantation logicielle/matérielle qui a su résoudre le problème lié au post-traitement d'un corrélateur optique. La mise en oeuvre en langage C est 56 fois trop lente et la version en assembleur est 16 fois trop lente. La version codesign est si performante qu'elle pourrait même traiter à elle seule deux corrélateurs optiques en temps réel ou un flot de 60 images par seconde pour un corrélateur optique.

L'algorithme de post-traitement d'un corrélateur optique du présent mémoire est limité à la détection d'un seul pic de corrélation par image. Une amélioration quant à cette limitation serait réalisable. Cet algorithme serait composé de N domaines où chaque domaine dans l'image serait propre à un pic de corrélation potentiel. La dimension d'une telle région serait dictée par l'espace occupé par un pic de corrélation. Une région trop grande pourrait à la limite inclure deux pics de corrélation et une région trop petite pourrait associer plusieurs domaines à un même pic de corrélation. Ainsi, cet algorithme serait composé de deux niveaux de compétition.

À plus haut niveau, une compétition se ferait entre pics de corrélation potentiels dans l'image pour l'obtention d'un domaine. Chaque domaine en compétition serait représenté par son pixel le plus intense. Si un pixel rencontré dans l'image ne fait partie d'aucun domaine et qu'il est plus intense que celui du plus faible des domaines, il se verrait attribuer ce domaine. Ce qui implique que l'algorithme doit connaître en temps réel lequel de ses domaines est le plus faible.

À plus bas niveau, la compétition se ferait à l'intérieur d'un même domaine. Le pixel courant le plus intense d'un domaine perdrait sa position si un pixel plus intense était rencontré à l'intérieur de ce domaine. Pour déterminer si le pixel courant fait partie du domaine, le système pourrait calculer sa distance au pixel le plus intense. Si cette distance est inférieure à une certaine limite représentant la dimension d'un domaine, le pixel courant serait identifié comme faisant partie de ce domaine.

Ce qui différencie les deux niveaux de compétition, c'est l'appartenance, ou la non-appartenance, du pixel courant à un domaine existant. De plus, cet algorithme pourrait s'exécuter en temps réel, ce qui éviterait de temporiser l'image en mémoire.

En conclusion, ce mémoire a démontré que la partition d'un algorithme en une partie logicielle et en une partie matérielle peut accélérer énormément une application qui demande un débit de calcul très élevé comme l'algorithme décrit dans ce mémoire. Cette partition nous permet de profiter à la fois de la flexibité et du faible coût des processeurs

pour la partie logicielle et de la rapidité d'exécution de circuits dédiés pour la partie matérielle. De plus, cette partie matérielle a été implantée sur des FPGA, ce qui résulte en une économie importante de coût et en un temps de développement plus rapide lorsque comparé à la conception d'un circuit intégré. Par cette mise en oeuvre logicielle/matérielle, nous avons dépassé les attentes par un facteur de deux en ce qui concerne la performance requise.

RÉFÉRENCES

ALBAHARNA, O., CHEUNG, P., et CLARKE, T. (mai 1994). Virtual hardware and the limits of computational speed-up. Proceedings of the IEEE International Symposium on Circuit and Systems, 159-162.

BERGERON, A., GAUVIN, J., GAGNON, F., GINGRAS, D., ARSENAULT, H.H., et DOUCET, M. (août 1995). Phase calibration and applications of a liquid-crystal spatial light modulator. Applied Optics, 34, 5133-5139.

BRAHAM, J., et HORNER, J.L. (1994). Real-time optical information processing, Academic Press, 531p.

CANNY, J. (novembre 1986). A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8, 679-697.

CASASENT, D.P., et ABRAMSON, N.H. (1978). Optical data processing : applications, Springer-Verlag, 286p.

CAULFIELD, H.J. (décembre 1982). Role of the Horner efficiency in the optimization of the spatial filters for optical pattern recognition. Applied Optics, 21, 4391-4392.

COHN, R.W., et HORNER, J.L. (août 1994). Effects of systematic phase errors on phase-only correlation. Applied Optics, 33, 5432-5439.

FLANNERY, D.L., et HORNER, J.L. (octobre 1989). Fourier optical signal processors. Proceedings of the IEEE , 77 , 1511-1527.

GIANINO, P.D., WOODS, C.L., et HORNER, J.L. (octobre 1995). Analysis of spatial light modulator contrast ratios and optical correlation. Applied Optics, 34, 6682-6694.

HECHT, E. (1988). Optics - Second Edition, Addison-Wesley Publishing company, inc., 676 p.

HILDRETH, E.C. (septembre 1985). Edge detection. Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1-22.

HORNER, J.L., et GIANINO, P.D. (mars 1984). Phase-only matched filtering. Applied Optics, 23, 812-816.

HORNER, J.L. (janvier 1992). Metrics for assessing pattern-recognition performance. Applied Optics, 31, 165-166.

HORNER, J.L. (juillet 1992). Clarification of Horner efficiency. Applied Optics, 31, 4629.

LAW, T., ITOH, H., et SEKI, H. (mai 1996). Image filtering, edge detection, and edge tracing using fuzzy reasoning. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18, 481-491.

MAULDIN, J.H. (1988). Light, lasers and optics - First Edition, Blue Ridge Summit, 243p.

NEKRASOV, V.V., ZBOROVSKY, A.A., IVANOV, B.B., TZUKERMAN, E.V., et SHIYAK, F.D. (avril 1992). Real-time coherent optical correlator for machine vision systems. Optical Engineering, 31, 789-793.

ROBERGE, D., et SHENG, Y. (septembre 1996). Optical real-time correlator for implementation of phase-only composite filters. Optical Engineering, 35, 2541-2547.

SCHOLL, M.S. (mars 1995). Architecture for object identification : incorporating an optical correlator and digital processing for display and recording of optical data. Optical Engineering, 34, 887-895.

SCHOLL, M.S., et EBERLEIN, S. (avril 1993). Automated site characterization for robotic sample acquisition systems. Optical Engineering, 32, 840-846.

SHULMAN, A.R. (1970). Optical data processing, John Wiley & Sons, inc, 710p.

TURNER, R.M., JARED, D.A., SHARP, G.D., et JOHNSON, K.M. (juin 1993). Optical correlator using very-large-scale integrated circuit/ferroelectric-liquid-crystal electrically addressed spatial light modulators. Applied Optics, 32, 3094-3101.

VANDER LUGT, A. (Avril 1964). Signal detection by complex spatial filtering. IEEE Transaction on Information Theory, 139-145.

VIJAYA KUMAR, B.V.K., et HASSEBROOK, L. (juillet 1990). Performance measures for correlation filters. Applied Optics, 29, 2997-3006.

YU, F.T.S., et GREGORY, D.A. (mai 1996). Optical pattern recognition : architectures and techniques. Proceedings of the IEEE, 84, 733-751.

ANNEXE A

Les scripts Matlab

```
function [img,map,R,G,B] = ReadRaster(name)

%
%  Cette fonction créé une matrice "img" dans Matlab à partir d'une image
%  de type Sun raster. "map" est la carte de couleur de l'image et "R", "G" et "B"
%  sont les composantes rouge,verte et bleu si l'image est sur 24 bits.
%

magic = '59a66a95';
RAS_MAGIC = hex2dec(magic); % nombre magique
RMT_NONE = 0;
RMT_EQUAL_RGB = 1;% red[ras_maplength/3],green[],blue[]

error(nargchk(1,1,nargin));
if nargout < 1
    error('Output arguments are missing');
elseif nargout > 5
    error('Too many output argument');
end

if(nargin < 1)
    error('Not enough input arguments.');end

fid = fopen(name,'rb');
if fid == -1
    error('Cannot open the file.'); end

if (fread(fid,1,'int32')== RAS_MAGIC)
    ras_width = fread(fid,1,'int32');
    ras_height = fread(fid,1,'int32');
    ras_depth = fread(fid,1,'int32');
    ras_length = fread(fid,1,'int32');
    ras_type = fread(fid,1,'int32');
    ras_maptype = fread(fid,1,'int32');
    ras_maplength = fread(fid,1,'int32');

    if ras_maptype == RMT_EQUAL_RGB
        red = zeros(1,ras_maplength/3);
        green=zeros(1,ras_maplength/3);
        blue = zeros(1,ras_maplength/3);
        red = fread(fid,[1,ras_maplength/3],'uchar');
        green = fread(fid,[1,ras_maplength/3],'uchar');
        blue = fread(fid,[1,ras_maplength/3],'uchar');
```

```
    map = [red'./256,green'./256,blue'./256];

    A = zeros(1,ras_height*ras_width);
    eval(['A = fread(fid,inf,"uint',int2str(ras_depth),'");']);
    A = A';
    if (( rem(ras_height,2) == 0) & (rem(ras_width,2) == 0))
        img = reshape(A,ras_width,ras_height);
        img = img';
    elseif (rem(ras_width,2) ~= 0)
        img = reshape(A,ras_width+1,ras_height);
        img = img';
        img = img(:,1:ras_width); % Remove excess column
    else
        img = reshape(A,ras_width,ras_height);
        img = img';
    end

elseif (ras_maptype == RMT_NONE)  % noir et blanc ou image 24 bits.

    A = zeros(ras_height,ras_width);
    eval(['A = fread(fid,inf,"uint',int2str(ras_depth),'");']);
    img = zeros(ras_height,ras_width);
    if (ras_depth == 24)
        fseek(fid,32,'bof');
        tmp = fread(fid,inf,'uchar');
        R = tmp(1:3:length(tmp));
        G = tmp(2:3:length(tmp));
        B = tmp(3:3:length(tmp));
    end
    if (( rem(ras_height,2) == 0) & (rem(ras_width,2) == 0))
        img = reshape(A,ras_width,ras_height);
        img = img';
        if (ras_depth == 24)
            R = reshape(R,ras_width,ras_height);
            G = reshape(G,ras_width,ras_height);
            B = reshape(B,ras_width,ras_height);
            R = R';G = G';B = B';
        end
    elseif (rem(ras_width,2) ~= 0)
        img = reshape(A,ras_width+1,ras_height);
        img = img';
        img = img(:,1:ras_width); % Enlève colonne en excès
        if (ras_depth == 24)
```

```
                R = reshape(R,ras_width+1,ras_height);
                G = reshape(G,ras_width+1,ras_height);
                B = reshape(B,ras_width+1,ras_height);
                R = R';G = G';B = B';
                R = R(:,1:ras_width);G = G(:,1:ras_width);B = B(:,1:ras_width);
            end
        else
            img = reshape(A,ras_width,ras_height);
            img = img';
            if (ras_depth == 24)
                R = reshape(R,ras_width,ras_height);
                G = reshape(G,ras_width,ras_height);
                B = reshape(B,ras_width,ras_height);
                R = R';G = G';B = B';
            end
        end
        if (ras_depth == 1)
            img = img.*255;
            colormap(gray(2));
            map = colormap;
        elseif (ras_depth <= 8)
            eval(['map = colormap(gray(2^',int2str(ras_depth),'-1));']);
            map = colormap;
        else
            map = gray(256);
        end
    end

else
    error('Not a raster file.');
end
fclose(fid);
```

```
function WriteRaster(img,name,map,Nbits,R,G,B)
%
%  Cette fonction enregistre dans un fichier une matrice decrivant une image
%  Sun raster. "img" représente la matrice, "name" est le nom du fichier créé.
%  Si "map" n'est pas fourni le script assume que l'image est sur 256 tons de gris.
%  This function saves the input image img in a SunRaster file format.
%

magic = '59a66a95';
RAS_MAGIC = hex2dec(magic); % Nombre magique
RMT_NONE = 0;
RMT_EQUAL_RGB = 1;

if nargin < 2
    error('At least two input arguments.');
elseif nargin > 7
    error(' Too many inputs');
elseif nargin < 3
    map = gray(256);
    Nbits = 8;
elseif nargin < 4
    Nbits = 8;
end

if (rem(Nbits,8) ~= 0)
    message = [num2str(Nbits),' bits is not supported'];
    error(message); end

fid = fopen(name,'wb');
if fid == -1
    error('Cannot create the file.');
end

ras_depth = Nbits;
ras_type = 1;


if (Nbits == 24)
    [N,M] = size(R);
    if ( size(R) ~= size(G) | size(G) ~= size(B) | size(R) ~= size(B))
        error('R, G, B images must have the same dimensions!'); end
    ras_length = N*M*3;
    ras_maptype = RMT_NONE;
```

```
    ras_maplength = 0;
else
    [N,M] = size(img);
    [Nmap,Mmap] = size(map);
    ras_length = N*M;
    ras_maptype = RMT_EQUAL_RGB;
    ras_maplength = Nmap*Mmap;
    if (max(max(map)) <=1)
        map = map*(2^Nbits-1);
    end
end


fwrite(fid,RAS_MAGIC,'int32');
fwrite(fid,M,'int32');
fwrite(fid,N,'int32');
fwrite(fid,ras_depth,'int32');
fwrite(fid,ras_length,'int32');
fwrite(fid,ras_type,'int32');
fwrite(fid,ras_maptype,'int32');
fwrite(fid,ras_maplength,'int32');

if (ras_maptype == RMT_EQUAL_RGB)
    red   = map(:,1); green = map(:,2); blue  = map(:,3);
    fwrite(fid,red,'uchar'); fwrite(fid,green,'uchar'); fwrite(fid,blue,'uchar');
    eval(['fwrite(fid,img","uint',int2str(ras_depth),'");']);
else
    if (length(img) ~= 0)
        eval(['fwrite(fid,img","uint',int2str(Nbits),'");']);
    else
    % Only the RGB have been provided
    error('Must generate an image from RGB: Not yet implemented');
    end
end
fclose(fid);
%end;
```

```
function filtre_moy3x3_bitshift3(name)

%
%  Cette fonction filtre une image avec un filtre de moyenne
%  de 3 par 3 pixels. L'image filtrée est la matrice "img_f".
%  name = nom du fichier de l'image raster a filtrer

[img,map] = ReadRaster(name);

M = size(img,1);   % M = nombre de range
N = size(img,2);   % N = nombre de colonne
img_f= img;

%%%%%%%%%%%%%%%%%%%%%%%%%%
% boucle pour le filtrage
%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=2:(M-1),
    for i=2:(N-1),
        temp = floor((img(j-1,i-1)+img(j-1,i)+img(j-1,i+1)+img(j,i-
1)+img(j,i)+img(j,i+1)+img(j+1,i-1)+img(j+1,i)+img(j+1,i+1))/8);
        if(temp>255)
            temp =255
        end
        img_f(j,i) = temp;
    end
end
```

```
function filtre_moy5x5(name,name_f)

%
%  Cette fonction filtre une image avec un filtre de moyenne
%  de 5 par 5pixels. L'image filtrée est la matrice "img_f".
%  name = nom du fichier de l'image raster a filtrer
%

[img,map] = ReadRaster(name);

M = size(img,1);% M = nombre de range
N = size(img,2);% N = nombre de colonne
img_f= img;

        % boucle pour le filtrage
for j=3:(M-2),
    for i=3:(N-2),
        img_f(j,i) = round((img(j-2,i-2)+img(j-2,i-1)+img(j-2,i)+img(j-2,i+1) ...
                    +img(j-2,i+2) ...
                    +img(j-1,i-2)+img(j-1,i-1)+img(j-1,i)+img(j-1,i+1)+img(j-1,i+2) ...
                    +img(j,i-2)+img(j,i-1)+img(j,i)+img(j,i+1)+img(j,i+2) ...
                    +img(j+1,i-2)+img(j+1,i-1)+img(j+1,i)+img(j+1,i+1)+img(j+1,i+2) ...
                    +img(j+2,i-2)+img(j+2,i-1)+img(j+2,i)+img(j+2,i+1)+img(j+2,i+2))...
                    /25);
    end
end
```

```
function filtre_med(name,name_f)

%
%  Cette fonction filtre une image avec un filtre médian de 3 par 3 pixels.
%  Le pixel resultant est le 5 ieme pixel lorsque les 9 sont classes.
%

[img,map] = ReadRaster(name);

M = size(img,1);% M = nombre de range
N = size(img,2);% N = nombre de colonne
img_f= img;
temp = [1 2 3 4 5 6 7 8 9];

        % boucle pour le filtrage
for j=2:(M-1),
    for i=2:(N-1),
        temp(1)=img(j-1,i-1);
        temp(2)=img(j-1,i);
        temp(3)=img(j-1,i+1);
        temp(4)=img(j,i-1);
        temp(5)=img(j,i);
        temp(6)=img(j,i+1);
        temp(7)=img(j+1,i-1);
        temp(8)=img(j+1,i);
        temp(9)=img(j+1,i+1);
        img_f(j,i) = median(temp);
    end
end
```

```
function [pos_x,pos_y,val_pic,somme] = find_pic(img)

%
% Cette fonction donne la position et la valeur du pixel le plus
% intense d'une image
%

M = size(img,1);% M = nombre de range
N = size(img,2);% N = nombre de colonne

val_pic = 0;
pos_x= 1;
pos_y= 1;
somme = 0;

% boucle pour trouver la valeur max

for j=3:(M-2),
    for i=3:(N-2),
        if img(j,i)>val_pic
            val_pic= img(j,i);
            pos_x= i;
            pos_y= j;
        end
    end
end

val_pic
pos_x
pos_y
```

```
function [prr]=prr(name,pct)

%
% Cette fonction prend une image sun raster "name" et y applique la
% mesure du prr selon le pourcentage fourni "pct".
%

img = ReadRaster(name);

[pos_x,pos_y,val_pic] = find_pic(img);

% les constantes
M       = size(img,1);% M = nombre de range
N       = size(img,2);% N = nombre de colonne
val_lim= pct*val_pic

% pour trouver le rayon x correspondant a pct*val_pic
i = pos_x;

while ((img(pos_y,i) > val_lim) & (i~=N))
    i = i + 1;
end

ray_x = i - pos_x

% pour trouver le rayon y correspondant a pct*val_pic
j = pos_y;

while ((img(j,pos_x) > val_lim) & (j~=M))
    j = j + 1;
end

ray_y = j - pos_y

prr = val_pic/((ray_x + ray_y)/2)
```

```
function [pente_moyenne] = pente_pic(name,r1,r2)

%
% Cette fonction donne la valeur de la pente du pic d'une image selon
% les parametres.
% name = 'nomdufichier.ras' (sun raster file)
% r1   = distance en pixel a partir du pic pour la premiere valeur pour pente
% r2   = distance en pixel a partir du pic pour la deuxieme valeur pour pente
% Ce fichier donne la pente resultante en x et en y
%

img = ReadRaster(name);

[pos_x,pos_y,val_pic] = find_pic(img);

x1 = pos_x + r1;
y1 = pos_y + r1;

x2 = pos_x + r2;
y2 = pos_y + r2;

delta = r2 - r1;

pente_x = (img(pos_y,x1) - img(pos_y,x2))/delta
pente_y = (img(y1,pos_x) - img(y2,pos_x))/delta
pente_moyenne = (pente_x + pente_y)/2
```

```
function [moment1] = moment1(name,pct)

%
% name = 'nomdufichier.ras'
% pct  = fraction du pic pour la valeur des rayons x et y
% Ce fichier prend une image .ras en entree et une fraction
% et donne la valeur du pic , sa position xy, et les rayons

img = ReadRaster(name);

[pos_x,pos_y,val_pic] = find_pic(img);

% les constantes
M       = size(img,1);% M = nombre de range
N       = size(img,2);% N = nombre de colonne

val_lim= pct*val_pic;
somme_pix = 0;
somme_mult = 0;

% pour traiter les valeurs sur le rayon x jusqu'a val_lim
i = pos_x + 1;

while ((img(pos_y,i) > val_lim) & (i~=N))
    somme_pix = somme_pix + img(pos_y,i);
    somme_mult = somme_mult + ((i - pos_x) * img(pos_y,i));
    i = i + 1;
end

ray_x = i - pos_x;

% pour traiter les valeurs sur le rayon y jusqu'a val_lim
j = pos_y + 1;

while ((img(j,pos_x) > val_lim) & (j~=M))
    somme_pix = somme_pix + img(j,pos_x);
    somme_mult = somme_mult + ((j - pos_y) * img(j,pos_x));
    j = j + 1;
end

ray_y = j - pos_y;
moment1 = somme_mult / somme_pix
```

```
function mesh_zoom(name,fen)

% Cette fonction prend une image .ras en entree et fait le graphique 3D
% avec une fenetre de zoom modifiable autour du pic de l'image
% parametres :
% name = 'fichier.ras'
% fen= moitie de la fenetre mesuree en indice

img = ReadRaster(name);

[pos_x,pos_y,val_pic] = find_pic(img)

fen_x= [(pos_x - fen):(pos_x + fen)];
fen_y= [(pos_y - fen):(pos_y + fen)];
img_zoom= img(fen_y,fen_x);

%colormap(gray)
%caxis([10 0])

mesh(fen_x,fen_y,img_zoom)
axis tight
%title(['Region du pixel le plus intense (',name,')'])
title(['Region du pixel le plus intense (image7a-f.ras)'])
xlabel('position x')
ylabel('position y')
zlabel('intensite du pixel')
rotate3d
```

```
function plotzoomxypic(fen)

%
% Cette fonction fait afficher deux graphiques chacun représentant une
% coupe en deux dimensions du pic de corrélation selon la fenêtre (fen).
%

name='image7a.ras'
img = ReadRaster(name);

[pos_x,pos_y,val_pic] = find_pic(img);

% pos_x constant et variation selon y avec fenetre
fen_y= [pos_y:(pos_y + fen)];
%fen_y= [(pos_y - fen):(pos_y + fen)];
Y       = img(fen_y,pos_x);

% pos_y constant et variation selon x avec fenetre
fen_x= [pos_x:(pos_x + fen)];
%fen_x= [(pos_x - fen):(pos_x + fen)];
X       = img(pos_y,fen_x);

subplot(211)
plot(fen_x,X)
axis([330 370 3 18])
%axis tight
title(['Pic avec pos Y cst (pas de filtrage)'])
xlabel('position en X')
ylabel('intensite du pixel')
grid

subplot(212)
plot(fen_y,Y)
axis([261 301 3 18])
%axis tight
title(['Pic avec pos X cst (pas de filtrage)'])
xlabel('position en Y')
ylabel('intensite du pixel')
grid
```

```
function CreateTab(img,name)

%
% Cette fonction créé un fichier en langage C qui taduit une matrice matlab
% "img" en un tableau dans le fichier C.
%

fid = fopen(name,'wb');
if fid == -1
    error('Cannot create the file.');
end


fprintf(fid,'{');

for j=1:480,

    fprintf(fid,'{');

    for i=1:4:637,
        if (mod(i,101)==0)
            fprintf(fid,'\n');
        end
        som = img(j,i) + (256*img(j,i+1)) + (65536*img(j,i+2)) + (16777216*img(j,i+3));
        fprintf(fid,'%i,',som);
    end

    fprintf(fid,'Q},\n');
end

fprintf(fid,'}');

fclose(fid);
```

```
function CreateCOMP(file_in,file_out)

%
% Cette fonction créé un fichier COMP pour l'environnement de
% simulation CoreSpex. Le fichier COMP contient l'image a traiter.
%

img = ReadRaster(file_in);

fid = fopen(file_out,'wb');
if fid == -1
    error('Cannot create the file.');
end

for j=1:480,
    for i=1:4:637,
        fprintf(fid,'\n');
        fprintf(fid,'0.0 ns  0 c  ');

        if img(j,i+3) == 0
            fprintf(fid,'00');
        elseif img(j,i+3) < 16
            fprintf(fid,'0%X',img(j,i+3));
        else
            fprintf(fid,'%X',img(j,i+3));
        end

        if img(j,i+2) == 0
            fprintf(fid,'00');
        elseif img(j,i+2) < 16
            fprintf(fid,'0%X',img(j,i+2));
        else
            fprintf(fid,'%X',img(j,i+2));
        end

        if img(j,i+1) == 0
            fprintf(fid,'00');
        elseif img(j,i+1) < 16
            fprintf(fid,'0%X',img(j,i+1));
        else
            fprintf(fid,'%X',img(j,i+1));
        end
```

```
        if img(j,i) == 0
            fprintf(fid,'00');
        elseif img(j,i) < 16
            fprintf(fid,'0%X',img(j,i));
        else
            fprintf(fid,'%X',img(j,i));
        end

    end
end

fclose(fid);
```

ANNEXE B

Le programme en langage C de l'algorithme de post-traitement du corrélateur optique

```
/*****************************************************************/
/* Algorithme du post-traitement d'un correlateur optique        */
/*                                                               */
/*  L'image a traiter (image[][]) est constituee de :            */
/*  480 lignes et de 640 pixels par ligne.                       */
/*  Les pixels (8 bits chacun) sont regroupes en paquet de 4     */
/*  car le C40 traite des mots de 32 bits.                       */
/*****************************************************************/


/*********************************/
/* Déclaration des fonctions     */

void rempl_ligne(unsigned int *src, unsigned int *dst);

void main(void)
{

/*********************************/
/* Déclaration des variables */

    unsigned int i,j,m,n;
  unsigned int pix_max;
  unsigned int pos_x;
  unsigned int pos_y;
  unsigned int pix_courant;
  unsigned int somme_pix;
  unsigned int val_lim;
  unsigned int trait_x;
  unsigned int trait_y;
  unsigned int somme_pix_ray;
  unsigned int somme_mult;
  int ray_x;
  int ray_y;
  float moment;
  float PRR;
  float PCE;
    unsigned int tampon[3][640];
  extern unsigned int image[480][160];

/*************************************************/
/* Make sure 'C4x cache and interrupts are enabled */
  asm( "  LDHI 0002Fh,R0  " );
```

```
  asm( " OR  0F800h,R0 " );
  asm( " LDPE R0,IVTP  " );
  asm( " LDPE R0,TVTP  " );
  asm( " AND 0E3FFh,ST " );
  asm( " OR  03800h,ST " );

/*********************************/
/* initialisation des variables  */

    pix_max=0;
    pos_x=0;
    pos_y=0;
  somme_pix=0;
  PCE=0;
  PRR=0;
  trait_x=0;
  trait_y=0;

/*****************************************************/
/* remplir les 2 premieres lignes du tableau tampon  */
/* j:lignei:pixel                                    */

    rempl_ligne(image[0], tampon[0]);
  rempl_ligne(image[1], tampon[1]);

/*****************************************************/
/* on applique la fenetre 3x3 qui fait un filtre de moyenne */

    for(j=2;j<480;j++)
    {
    m = j % 3;
    rempl_ligne(image[j], tampon[m]);

    for(i=1;i<639;i++)
    {
    pix_courant = 0;
    for(m=0;m<3;m++)
      {
      for(n=(i-1);n<(i+2);n++)
        {
            pix_courant += tampon[m][n];
        }
      }
```

```
pix_courant = pix_courant >> 3;   /* approx division par 9 sans reste*/
somme_pix += pix_courant;

if(pix_courant > pix_max)
{
trait_x = 1;   /* flag */
  trait_y = 1;
pix_max = pix_courant;
  pos_x = i;
  pos_y = j;
  ray_x = -1;
  ray_y = -1;
  val_lim = pix_courant >> 2;/* division par 4 sans reste (25% pix_max)*/
  moment = 0;
  somme_pix_ray = 0;
  somme_mult = 0;
}

if(trait_x && (pos_y == j))
{
if(pix_courant > val_lim)
{
  ++ray_x;
    if(ray_x != 0)
    {
    somme_pix_ray += pix_courant;
      somme_mult += (ray_x * pix_courant);
    }
}
  else
  {
  trait_x = 0;
  }
}

if(trait_y && (pos_x == i))
{
if(pix_courant > val_lim)
{
  ++ray_y;
    if(ray_y != 0)
    {
    somme_pix_ray += pix_courant;
```

```
            somme_mult += (ray_y * pix_courant);
          }
        }
        else
        {
        trait_y = 0;
        }
      }

    }
    }

/* pour avoir l'indice pos_x comme dans matlab */
/* pos_y l'est deja                            */
  ++pos_x;

  PRR = (float)pix_max / (((float)(ray_x + ray_y)) / 2.0);
  moment = ((float)somme_mult) / ((float)somme_pix_ray);
  PCE = ((float)pix_max) / ((float)somme_pix);

}

void rempl_ligne(unsigned int *src, unsigned int *dst)
{
    int i;

  for(i=0;i<160;i++)
  {
    *(dst++) = (0x000000FF & (*src));
    *(dst++) = (0x0000FF00 & (*src))>>8;
    *(dst++) = (0x00FF0000 & (*src))>>16;
    *(dst++) = (0xFF000000 & (*src))>>24;
    ++src;
  }
}
```

ANNEXE C

Le programme en assembleur du TMS320C40 de l'algorithme de post-traitement du corrélateur optique

```
*********************************************************
*
*   Algorithme de post-traitement d'un corrélateur
*
*********************************************************

.version40
FP .set     AR3
.globlREMPL_LIGNE
.globlmain

*********************************************************
* FONCTION PRINCIPALE : main()
*********************************************************
main:
PUSHFP          ;*** pousse AR3 sur le stack
                ;*** met dans AR3 la valeur de SP
LDISP,FP        ;*** FP (AR3) pointe sur adresse de base de tampon[][]
ADDI1920,SP     ;*** incremente SP de 1921
.globl_image    ;*** image[][] originale variable externe

***********************************************
*** R0non-reserve (calcul temporaire)
*** R1non-reserve (calcul temporaire)
*** R2assigned to variable  somme_pix
*** R3assigned to variable  ray_x
*** R4assigned to variable  j
*** R5assigned to variable  pix_max
*** R6assigned to variable  ray_y
*** R7utilise par REMPLACE_LIGNE non-protege
*** R8assigned to variable  val_lim
*** R9assigned to variable  i
*** R10assigned to variable  somme_pix_ray
*** R11assigned to variable  somme_mult
*** AR0assigned to pointeur element de IMAGE
*** AR1assigned to pointeur element de tampon
*** AR4assigned to variable  trait_y
*** AR5assigned to variable  trait_x
*** AR6assigned to variable  pos_y
*** AR7assigned to variable  pos_x
*** BKassigned to variable  pix_courant
*** IR1assigned to 640
***********************************************
```

```
LDHI 0002Fh,R0          ;*** R0 = 002F 0000h
OR  0F800h,R0           ;*** R0 = 002F F800h
LDPE R0,IVTP            ;*** IVTP pointe a 002F F800h
LDPE R0,TVTP            ;*** TVTP pointe a 002F F800h
                        ;*** force CF,CE et CC a 0.

LDHI00010h,AR0          ;*** ARO = 0010 0000h
LDHI0FFFFh,R1           ;***
ADDI0C00Fh,R1           ;*** R1 = FFFF FF0Fh
AND3R1,*AR0,R0          ;***
STIR0,*AR0              ;***

ADDI00004h,AR0          ;*** AR0 = 0010 0004h
AND3R1,*AR0,R0          ;***
STIR0,*AR0              ;***

AND  0E3FFh,ST          ;*** cache not enabled
                        ;*** force CE,CC et GIE a 1.
                        ;*** (CE,CC) cache enabled but not frozen (can write)
OR  03800h,ST           ;*** (GIE) CPU respond to an enabled interrupt

LDI640,IR1              ;*** IR1 reserve pour index lors somme fenetre

LDHI030h,AR1
ADDIIMAGE+3,AR1

ADDI3FP,1,AR0           ;*** (AR0 = FP + 1)

CALLREMPL_LIGNE         ;*** appel fonction mettre nouvelle ligne dans tampon

CALLREMPL_LIGNE         ;*** appel fonction mettre nouvelle ligne dans tampon

*******************************
*   AR0 = tampon + (2*640)  pointe sur 3e ligne
*   AR1 = IMAGE + 320
*******************************

                        ;*** Initialisation de variables
LDI0,R2                 ;*** somme_pix
LDI0,R5
LDA0,AR7
LDA0,AR6
LDA0,AR5
```

```
LDA0,AR4
LDI2,R4
ADDI3AR0,IR1,IR0      ;*** IR0 possede l'adresse fin tampon (tampon+(3*640))
L2:

CALLREMPL_LIGNE

LDI1,R9               ;*** R9 indice du pixel sur ligne "i"

*********************************************************
* A ce moment tampon[][] est pret pour filtrage et calcul mesures *
*********************************************************

*******************************************
*  Additionne dans BK les 9 pixel de la fenetre  *
*   A B C                                 *
*   D E F                                 *
*   G H I                                 *
* BK est un accumulateur                  *
*******************************************
L4:                   ;*** INITIALISATION
ADDI3FP,R9,AR2        ;***
LDA0,BK               ;***

ADDI3*AR2,*+AR2(1),BK ;*** BK =  A + B
ADDI*+AR2(2),BK       ;*** BK = BK + C
ADDI*++AR2(IR1),BK    ;*** BK = BK + D
ADDI*+AR2(1),BK       ;*** BK = BK + E
ADDI*+AR2(2),BK       ;*** BK = BK + F
ADDI*++AR2(IR1),BK    ;*** BK = BK + G
ADDI*++AR2,BK         ;*** BK = BK + H
ADDI*++AR2,BK         ;*** BK = BK + I

LSH-3,BK              ;*** La somme des 9 pixels est /8 sans reste (BK=BK>>3)
ADDIBK,R2             ;*** somme_pix = somme_pix + pix_courant
CMPIR5,BK             ;*** pix_courant - pix_max(R5 = pix_max)
BHIL13                ;*** branche si (pix_courant > pix_max)

CMPI0,AR5             ;***
BZ L15                ;*** branche si trait_x == 0

CMPIR4,AR6            ;***
BNZL15                ;*** branche si j != pos_y(AR6 - R4) = 0
```

```
CMPIR8,BK          ;*** (pix_courant - val_limite)
LDILS0,AR5         ;*** Si (pix_courant - val_limite)<=0 alors trait_x = 0
BLSLL4             ;***

ADDI1,R3           ;*** ray_x = ray_x + 1
ADDIBK,R10         ;*** somme_pix_ray = somme_pix_ray + pix_courant
MPYI3R3,BK,R0      ;*** R0 = ray_x * pix_courant
ADDIR0,R11         ;*** somme_mult = somme_mult + R0

B   LL4            ;*** branche toujours a LL4

*************************************
* Entre dans L13 si pix_courant > pix_max*
*************************************
L13:
LDIBK,R5           ;*** pix_max = pix_courant
LDAR9,AR7          ;*** pos_x = i
LDAR4,AR6          ;*** pos_y = j
LSH3-2,R5,R8       ;*** val_limite = pix_courant>>3 ou 25%pix_courant
LDA1,AR5           ;*** trait_x = 1
LDA1,AR4           ;*** trait_y = 1
LDI0,R3            ;*** ray_x = 0
LDI0,R6            ;*** ray_y = 0
LDI0,R10           ;*** somme_pix_ray = 0
LDI0,R11           ;*** somme_mult = 0
B   LL4            ;***

L15:
CMPI0,AR4
BZ LL4             ;*** branche si (trait_y == 0)

CMPIR9,AR7

BNZLL4             ;*** branche si (i != pos_x)

CMPIR8,BK          ;*** pix_courant - val_limite
LDILS0,AR4         ;*** Si (pix_courant - val_limite)<=0 alors trait_y = 0
BLSLL4             ;***

ADDI1,R6           ;*** ray_y = ray_y + 1
ADDIBK,R10         ;*** somme_pix_ray = somme_pix_ray + pix_courant
MPYI3R6,BK,R0      ;*** R0 = ray_y * pix_courant
```

```
ADDIR0,R11          ;*** somme_mult = somme_mult + R0

LL4:
ADDI1,R9            ;*** ++i ou ++R9
CMPI639,R9          ;***
BLOL4               ;*** branche si (i - 639) < 0

CMPIIR0,AR0         ;***
BNZMODULO           ;*** Si (AR0 == tampon + (3*640)) alors
ADDI3FP,1,AR0       ;*** AR0 = tampon

MODULO:
ADDIR4,1,R4
CMPI480,R4
BLOL2               ;*** branche si (R4 - 480) < 0

*****************************
* Rendu ici le traitement est fini *
*****************************

ADDI3R3,R6,R0       ;*** R0 = ray_x + ray_y
FLOATR0,R1          ;*** R1 = float(R0)
LDF0.5,R0           ;*** R0 = 0.5
MPYFR1,R0           ;*** R0 = (ray_x + ray_y) / 2

*********************************
* ici c'est pour avoir l'inverse de R0
*********************************
RCPFR0,R1           ;***
MPYF3R1,R0,R7       ;*** 1ere iteration
SUBRF2.0,R7         ;*** (precision de 16 bits)
MPYFR7,R1           ;***
MPYF3R1,R0,R7       ;*** 2eme iteration
SUBRF2.0,R7         ;*** (precision de 32 bits)
MPYFR7,R1           ;*** R1 = 1 / R0

FLOATR5,R0          ;*** R0 = float(pix_max)
MPYF3R0,R1,R3       ;*** R3 = float(PRR de l'image)

FLOATR2,R0          ;*** R0 = float(somme_pix)
RCPFR0,R1           ;***
MPYF3R1,R0,R7       ;*** 1ere iteration
SUBRF2.0,R7         ;*** (precision de 16 bits)
```

```
MPYFR7,R1          ;***
MPYF3R1,R0,R7      ;*** 2eme iteration
SUBRF2.0,R7        ;*** (precision de 32 bits)
MPYFR7,R1          ;*** R1 = 1 / R0

FLOATR5,R0         ;*** R0 = float(pix_max)
MPYF3 R0,R1,R8     ;*** R8 = float(PCE)

FLOAT R10,R0       ;*** R0 = float(somme_pix_ray)
RCPFR0,R1          ;***
MPYF3R1,R0,R7      ;*** 1ere iteration
SUBRF2.0,R7        ;*** (precision de 16 bits)
MPYFR7,R1          ;***
MPYF3R1,R0,R7      ;*** 2eme iteration
SUBRF2.0,R7        ;*** (precision de 32 bits)
MPYFR7,R1          ;*** R1 = 1 / R0

FLOAT R11,R0       ;*** R0 = float(somme_mult)
MPYF3R0,R1,R10     ;*** R10 = MOMENT

LDI*-FP(1),R1
LDI*FP,FP

SUBI1923,SP
B   R1

.globlREMPL_LIGNE

*****************************************************
* FONCTION : REMPL_LIGNE
* le but de cette fonction est de sortir les pixels
* de IMAGE et de les mettre dans tampon.
* dans IMAGE les pixels (8bits) sont en paquet de 4.
* dans tampon les pixels sont seul sur 32 bits dans LSB.
*
* La fonction recoit deux valeur par des registres :
*   AR0 == dst  (ligne courante de tampon)
*   AR1 == src  (ligne courante de IMAGE)
*   R7 EST AFFECTE
*****************************************************
REMPL_LIGNE:
LDI159,R7              ;*** load 159 dans R7
LDI000000FFh,R0        ;*** (0000 00FFh) -> R0
```

```
FRAGMENTE:

ANDR0,*AR1,R1     ;*** R1 = (0000 00FFh AND image[j][i])
***************PREMIER pixel
STIR1,*AR0++      ;*** R1 -> tampon++
LSH-8,*AR1,R1     ;*** R1 = image[i][j] avec 8 bit shift vers la droite
ANDR0,R1          ;*** R1 = R1 AND (0000 00FFh)
***************DEUXIEME pixel
STIR1,*AR0++      ;*** R1 -> tampon++
LSH-16,*AR1,R1    ;*** R1 = image[i][j] avec 16 bit shift vers la droite
ANDR0,R1          ;*** R1 = R1 AND (0000 00FF)
***************TROISIEME pixel
STIR1,*AR0++      ;*** R1 -> tampon++
LSH3-24,*AR1,R1   ;*** R1 = image[i][j] avec 24 bit shift vers la droite
***************QUATRIEME pixel
STIR1,*AR0++      ;*** R1 -> tampon++
ADDI1,AR1         ;*** prochain block de 4 pixels
SUBI1,R7          ;*** R7 = R7 -1
BGEFRAGMENTE      ;*** tant que RC >= 0 branche a FRAGMENTE

POPR1             ;*** Remet l'ancienne valeur dans AR4
B   R1            ;*** retourne au main

*******************************************
* DEFINE CONSTANTS                        *
*******************************************
.sect".const"
IMAGE:
.word _image      ;

.end
```

ANNEXE D

Le VHDL du filtre de moyenne

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity filtre_moy is
 port ( Reset   : in  std_logic;   -- Reset input
        GenClk1 : in  std_logic;   -- Clock1 input

        ERCOSC  : out std_logic;
        ECmdReq : out std_logic;
        CPMSync : in  std_logic;

   -- Input FIFO
        FID   : in  std_logic_vector(7 downto 0);
        FIRdy : in  std_logic;
        FIEn  : out std_logic;

   -- Memory banks 0 and 1
        XGate : in   std_logic;
        XA    : out  std_logic_vector(18 downto 8);
        X0A   : out  std_logic_vector(7 downto 0);
        X1A   : out  std_logic_vector(7 downto 0);
        XD    : inout std_logic_vector(31 downto 0);
        X0OE  : out  std_logic;
        X0WE  : out  std_logic;
        X1OE  : out  std_logic;
        X1WE  : out  std_logic;

   -- Memory banks 2 and 3
        YGate : in   std_logic;
        YA    : out  std_logic_vector(18 downto 8);
        Y2A   : out  std_logic_vector(7 downto 0);
        Y3A   : out  std_logic_vector(7 downto 0);
        YD    : inout std_logic_vector(31 downto 0);
        Y2OE  : out  std_logic;
        Y2WE  : out  std_logic;
        Y3OE  : out  std_logic;
        Y3WE  : out  std_logic;

   -- Inter-VPE link
        ILINK : out std_logic_vector(23 downto 0)
      );
end filtre_moy;
```

```
architecture behave of filtre_moy is


   ------------------------------------------------------------------
   -- section des composants
   ------------------------------------------------------------------
component compt8re
port (
clk      : in  std_logic;
     rst          : in  std_logic;
         clk_en       : in  std_logic;
         q            : out std_logic_vector(7 downto 0);
         parite_out: out std_logic;
         zero         : out std_logic
    );
end component;

component fifo_in_vpe
    port (
         Engclk      : in std_logic;
         Frame_out: out std_logic_vector(31 downto 0);
         FrameAvail: out std_logic;
         FrameTaken: in std_logic;
         RESET_in: in std_logic;

         FifoEn      : out std_logic;
         FifoRdy     : in std_logic;
         FifoData: in std_logic_vector(7 downto 0));
end component;

component controleur
port (
            clk: in std_logic;
         rst          : in std_logic;
         parite_in: in std_logic;
         zero         : in std_logic;  -- ca vient de compt8re
         FrameAvail: in std_logic;  -- ca vient de fifo_in
         RamWordAvail: in std_logic;  -- ca vient de ram_fsm

         LoadReg   : out std_logic;
         ShiftReg_en: out std_logic;
         WindowOK: out std_logic  -- signal une fen3x3 valide
    );
```

```
end component;

component ram_fsm
    port (
        clk         : in std_logic;
        rst         : in std_logic;
        LoadReg     : in std_logic;

        Add_clk_en: out std_logic;
        RamWordAvail: out std_logic;
        RAM_WE: out std_logic;
        RAM_OE : out std_logic;
        DATA_en : out std_logic
    );
end component;

component registres
port(
          pix_fifo: in std_logic_vector(31 downto 0);
        pix_lign_impr: in std_logic_vector(31 downto 0);
        pix_lign_pair: in std_logic_vector(31 downto 0);

        clk         : in std_logic;
        rst         : in std_logic;
        clk_en      : in std_logic;
        sel         : in std_logic;-- select pour les multiplexeurs

        p1          : out std_logic_vector(7 downto 0);
        p2          : out std_logic_vector(7 downto 0);
        p3          : out std_logic_vector(7 downto 0);
        p4          : out std_logic_vector(7 downto 0);
        p5          : out std_logic_vector(7 downto 0);
        p6          : out std_logic_vector(7 downto 0);
        p7          : out std_logic_vector(7 downto 0);
        p8          : out std_logic_vector(7 downto 0);
        p9          : out std_logic_vector(7 downto 0)
    );
end component;

component additionneur
port(
          clk: in std_logic;
        rst         : in std_logic;
```

```
        WindowOK: in std_logic;
        p1        : in std_logic_vector(7 downto 0);
        p2        : in std_logic_vector(7 downto 0);
        p3        : in std_logic_vector(7 downto 0);
        p4        : in std_logic_vector(7 downto 0);
        p5        : in std_logic_vector(7 downto 0);
        p6        : in std_logic_vector(7 downto 0);
        p7        : in std_logic_vector(7 downto 0);
        p8        : in std_logic_vector(7 downto 0);
        p9        : in std_logic_vector(7 downto 0);

        Data_Ilink_en: out std_logic;
        pix_flt   : out std_logic_vector(7 downto 0)
    );
end component;

signal Add_clk_en : std_logic;
signal Adresse : std_logic_vector(7 downto 0);
signal parite : std_logic;
signal zero : std_logic;
signal Frame_out : std_logic_vector(31 downto 0);
signal FrameAvail : std_logic;
signal LoadReg : std_logic;
signal RamWordAvail : std_logic;
signal ShiftReg_en : std_logic;
signal WindowOK : std_logic;
signal RAM_WE : std_logic;
signal RAM_OE : std_logic;
signal pix_lign_impr : std_logic_vector(31 downto 0);
signal pix_lign_pair : std_logic_vector(31 downto 0);
signal p1 : std_logic_vector(7 downto 0);
signal p2 : std_logic_vector(7 downto 0);
signal p3 : std_logic_vector(7 downto 0);
signal p4 : std_logic_vector(7 downto 0);
signal p5 : std_logic_vector(7 downto 0);
signal p6 : std_logic_vector(7 downto 0);
signal p7 : std_logic_vector(7 downto 0);
signal p8 : std_logic_vector(7 downto 0);
signal p9 : std_logic_vector(7 downto 0);
signal dirXD : std_logic;
signal dirYD : std_logic;
signal Word_out : std_logic_vector(31 downto 0);
signal pix_flt : std_logic_vector(7 downto 0);
```

```
signal Data_Ilink_en : std_logic;
signal DATA_en : std_logic;


begin

 ERCOSC <= 'Z';
 ECmdReq <= CPMSync;
 ILINK <= "ZZZZZZZZZZZZZZZZ" &  Data_Ilink_en & pix_flt;


 Cpt_add : compt8report map (GenClk1,Reset,Add_clk_en,Adresse,parite,zero);

 FifoIn :  fifo_in_vpeport map
(GenClk1,Frame_out,FrameAvail,LoadReg,Reset,FIEn,FIRdy,FID);

 Cont :    controleurport map
(GenClk1,Reset,parite,zero,FrameAvail,RamWordAvail,LoadReg,ShiftReg_en,Window
OK);

 RAM :     ram_fsmport map
(GenClk1,Reset,LoadReg,Add_clk_en,RamWordAvail,RAM_WE,RAM_OE,DATA_en)
;

 REGS :    registresport map
(Frame_out,pix_lign_impr,pix_lign_pair,GenClk1,Reset,ShiftReg_en,LoadReg,p1,p2,p3,
p4,p5,p6,p7,p8,p9);

 ADD :     additionneur port map
(GenClk1,Reset,WindowOK,p1,p2,p3,p4,p5,p6,p7,p8,p9,Data_Ilink_en,pix_flt);

 READ_RAM_REGS : process(GenClk1,Reset,RAM_OE,XD,YD)
 begin
    if (Reset='1') then
      pix_lign_impr <= (others=>'0');
      pix_lign_pair <= (others=>'0');
    elsif (RAM_OE ='0') then
      if (GenClk1='1' and GenClk1'event) then
         pix_lign_impr <= XD;
         pix_lign_pair <= YD;
      end if;
    end if;
 end process;
```

```
WRITE_RAM_REGS : process(GenClk1,Reset,LoadReg,Frame_out)
begin
   if (Reset='1') then
     Word_out <= (others=>'0');
   elsif (LoadReg='1') then
     if (GenClk1='1' and GenClk1'event) then
        Word_out <= Frame_out;
     end if;
   end if;
end process;

-------------------------------------------------------------------------
-- LES SIGNAUX POUR LA RAM
-------------------------------------------------------------------------
XD<= Word_out after 7 ns when dirXD='0' else (others=>'Z') after 7 ns;
YD<= Word_out after 7 ns when dirYD='0' else (others=>'Z') after 7 ns;
dirXD<= '0' when (DATA_en='1' and XGate='1' and parite='0') else '1';
dirYD<= '0' when (DATA_en='1' and YGate='1' and parite='1') else '1';

-- dirXD<= '0' when (RAM_WE='0' and XGate='1' and parite='0') else '1';
-- dirYD<= '0' when (RAM_WE='0' and YGate='1' and parite='1') else '1';

X0A<= Adressewhen Xgate='1' else (others=>'Z');
X1A<= Adressewhen Xgate='1' else (others=>'Z');
XA<= (others=>'0')when Xgate='1' else (others=>'Z');
Y2A<= Adressewhen Ygate='1' else (others=>'Z');
Y3A<= Adressewhen Ygate='1' else (others=>'Z');
YA<= (others=>'0')when Ygate='1' else (others=>'Z');
X0OE <= RAM_OE when Xgate='1' else 'Z';
X1OE <= RAM_OE when Xgate='1' else 'Z';
Y2OE <= RAM_OE when Ygate='1' else 'Z';
Y3OE <= RAM_OE when Ygate='1' else 'Z';
X0WE <= RAM_WE when (Xgate='1' and parite='0') else 'Z';
X1WE <= RAM_WE when (Xgate='1' and parite='0') else 'Z';
Y2WE <= RAM_WE when (Ygate='1' and parite='1') else 'Z';
Y3WE <= RAM_WE when (Ygate='1' and parite='1') else 'Z';

end behave;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity fifo_in_vpe is
    port (
        --=-- Module Interface --=--

        Engclk     : in std_logic;
        Frame_out: out std_logic_vector(31 downto 0);
        FrameAvail: out std_logic;
        FrameTaken: in std_logic;
        RESET_in: in std_logic; -- ** Dummy RESET for Synthesis

        --=-- CPM Interface --=--

        FifoEn     : out std_logic;
        FifoRdy    : in std_logic;
        FifoData: in std_logic_vector(7 downto 0));
end fifo_in_vpe;

architecture behavioral of fifo_in_vpe is

    -- -=-=-=-=-=-=-=-=-=-=-=- --
    -- States definitions --
    -- -=-=-=-=-=-=-=-=-=-=-=- --

    type input_states_def is (PowerUp, WaitByte1, LoadByte2, LoadByte3, LoadByte4,
WaitForTake);

    attribute enum_encoding : string;
    attribute enum_encoding of input_states_def: type is "000001 000010 000100
001000 010000 100000";
    signal present_in_state, next_in_state: input_states_def;

    -- -=-=-=-=-=-=-=-=-=-=-=- --
    -- Signals definitions --
    -- -=-=-=-=-=-=-=-=-=-=-=- --

    signal FrameBuf       : std_logic_vector(31 downto 0);
    signal byte_en        : std_logic_vector(3 downto 0);
    signal vdd                 : std_logic;

    -- -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- --
```

```
    -- Components definitions --
    -- -=-=-=-=-=-=-=-=-=-=-=-= --

    component Framer
    port (
            clk         : in std_logic;
            rst         : in std_logic;
            data_in: in std_logic_vector(7 downto 0);
            store_en: in std_logic;
            byte_en: in std_logic_vector(3 downto 0);
            frame_out: out std_logic_vector(31 downto 0)
    );
    end component;


begin


    -- ---------------------------- --
    -- ++ -- Basic Assignments -- ++ --

    Frame_out <= FrameBuf;
    vdd       <= '1';

    -- ---------------------------------- --
    -- ++ -- State Machine definitions -- ++ --

    FSMdef: process(Engclk, RESET_in)
    begin
        if (RESET_in = '1') then
            present_in_state <= PowerUp;
        elsif (Engclk='1' and Engclk'event) then
            present_in_state <= next_in_state;
        end if;
    end process;


    -- ------------------------------- --
    -- ++ -- State machine behavior -- ++ --

    FSM1: process(present_in_state, FifoRdy, FrameTaken)
    begin
        case present_in_state is
```

```
            when PowerUp =>
                next_in_state <= WaitByte1;

            when WaitByte1 =>
                if (FifoRdy = '1') then
                    next_in_state <= LoadByte2;
                else
                    next_in_state <= WaitByte1;
                end if;

            when LoadByte2 =>
                next_in_state <= LoadByte3;

            when LoadByte3 =>
                next_in_state <= LoadByte4;

            when LoadByte4 =>
                next_in_state <= WaitForTake;

            when WaitForTake =>
                if (FrameTaken = '1') then
                    next_in_state <= WaitByte1;
                else
                    next_in_state <= WaitForTake;
                end if;

        end case;
    end process;


    -- ------------------------------------ --
    -- ++ -- State Machine Dependant Signals -- ++ --

    byte_en(0) <= '1' when (present_in_state = WaitByte1) else '0';

    byte_en(1) <= '1' when (present_in_state = LoadByte2) else '0';

    byte_en(2) <= '1' when (present_in_state = LoadByte3) else '0';

    byte_en(3) <= '1' when (present_in_state = LoadByte4) else '0';

    FifoEn <= '0' when ((present_in_state = WaitByte1) or
                (present_in_state = LoadByte2) or
```

```
                (present_in_state = LoadByte3) or
                (present_in_state = LoadByte4)) else '1';

    FrameAvail <= '1' when (present_in_state = WaitForTake) else '0';


    -- ------------------------ --
    -- ++ -- ReFraming Buffer -- ++ --

    Frm : FRAMER port map
        (
            Engclk,
            RESET_in,
            fifoData,
            vdd,
            byte_en,
            FrameBuf
        );


end behavioral;
```

```
library ieee;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;

entity controleur is
port (clk    : in std_logic;
      rst    : in std_logic;
      parite_in: in std_logic;
      zero   : in std_logic;  -- ca vient de compt8re
      FrameAvail: in std_logic;  -- ca vient de fifo_in
      RamWordAvail: in std_logic;  -- ca vient de ram_fsm

      LoadReg: out std_logic;
      ShiftReg_en: out std_logic;
      WindowOK: out std_logic  -- signal une fen3x3 valide
     );
end controleur;

architecture behave of controleur is

   type ctr_state is (PowerUp,WriteLine1,WriteLine2,LoadWords,ShiftReg);
   attribute enum_encoding : string;
   attribute enum_encoding of ctr_state : type is "00001 00010 00100 01000 10000";
   signal present_ctr_state, next_ctr_state : ctr_state;

   signal Qout: std_logic_vector(1 downto 0);
   signal FirstPixLine : std_logic;
   signal compt_y_en: std_logic;
   signal pos_y: std_logic_vector(8 downto 0);


begin

   FSMdef : process(clk,rst)
   begin
    if (rst = '1') then
       present_ctr_state <= PowerUp;
    elsif (clk='1' and clk'event) then
       present_ctr_state <= next_ctr_state;
    end if;
   end process;
```

```
  FSM1:
process(present_ctr_state,parite_in,zero,FrameAvail,RamWordAvail,Qout,pos_y)
  begin
   case present_ctr_state is
      when PowerUp =>
       next_ctr_state <= WriteLine1;

      when WriteLine1 =>
       if (parite_in = '1') then
          next_ctr_state <= WriteLine2;
       else
          next_ctr_state <= WriteLine1;
       end if;

      when WriteLine2 =>
       if (parite_in = '0') then
          next_ctr_state <= LoadWords;
       else
          next_ctr_state <= WriteLine2;
       end if;

      when LoadWords =>
       if (FrameAvail='1' and RamWordAvail='1') then
          next_ctr_state <= ShiftReg;
       else
          next_ctr_state <= LoadWords;
       end if;

      when ShiftReg =>
       if (Qout=3) then
          if (pos_y=479 and zero='1') then
           next_ctr_state <= WriteLine1;
          else
           next_ctr_state <= LoadWords;
          end if;
       else
          next_ctr_state <= ShiftReg;
       end if;


  end case;
 end process;
```

```
compt2e : process(clk,rst,present_ctr_state)
begin
 if (rst='1') then
    Qout <= (others=>'0');
 elsif (clk='1' and clk'event) then
    if (present_ctr_state = ShiftReg) then
     Qout <= Qout + 1;
    end if;
 end if;
end process compt2e;


shift2 : process(clk,rst,Qout,zero)
begin
 if (rst='1') then
    FirstPixLine <= '1';
 elsif (Qout=0 and zero='1') then
    FirstPixLine <= '1';
 elsif (Qout=2) then
    FirstPixLine <= '0';
 end if;
end process shift2;

compt_pos_y : process (clk,rst,compt_y_en,pos_y)
begin
 if (rst='1') then
    pos_y <= "000000010";--"111011110";
 elsif (clk'event and clk='1') then
    if (compt_y_en='1') then
     pos_y <= pos_y + 1;
     if (pos_y=479) then
        pos_y <= "000000010";
     end if;
    end if;
 end if;
end process compt_pos_y;

 LoadReg<= '1' when ((FrameAvail='1' and RamWordAvail='1' and
present_ctr_state=LoadWords ) or
            (FrameAvail='1' and RamWordAvail='1' and
present_ctr_state=WriteLine1 ) or
            (FrameAvail='1' and RamWordAvail='1' and
```

```
present_ctr_state=WriteLine2)) else '0';

   ShiftReg_en <= '1' when (next_ctr_state=ShiftReg) else '0';

   WindowOK<= '1' when (present_ctr_state = ShiftReg and FirstPixLine = '0') else '0';

   compt_y_en<= '1' when ((present_ctr_state = ShiftReg and next_ctr_state =
LoadWords and zero = '1') or
                (present_ctr_state = ShiftReg and next_ctr_state = WriteLine1)) else '0';

end behave;
```

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;

entity additionneur is
port(
      clk: in std_logic;
     rst      : in std_logic;
     WindowOK: in std_logic;
      p1: in std_logic_vector(7 downto 0);
     p2       : in std_logic_vector(7 downto 0);
     p3       : in std_logic_vector(7 downto 0);
     p4       : in std_logic_vector(7 downto 0);
     p5       : in std_logic_vector(7 downto 0);
     p6       : in std_logic_vector(7 downto 0);
     p7       : in std_logic_vector(7 downto 0);
     p8       : in std_logic_vector(7 downto 0);
     p9       : in std_logic_vector(7 downto 0);

     Data_Ilink_en: out std_logic;
     pix_flt : out std_logic_vector(7 downto 0)
     );
end additionneur;

architecture behave of additionneur is

   signal s1 : std_logic_vector(8 downto 0);
   signal s2 : std_logic_vector(8 downto 0);
   signal s3 : std_logic_vector(8 downto 0);
   signal s4 : std_logic_vector(8 downto 0);
   signal s1_c : std_logic_vector(8 downto 0);
   signal s2_c : std_logic_vector(8 downto 0);
   signal s3_c : std_logic_vector(8 downto 0);
   signal s4_c : std_logic_vector(8 downto 0);
   signal p9_c1 : std_logic_vector(7 downto 0);
   signal p9_c2 : std_logic_vector(7 downto 0);
   signal p9_c3 : std_logic_vector(7 downto 0);
   signal s5 : std_logic_vector(9 downto 0);
   signal s6 : std_logic_vector(9 downto 0);
   signal s5_c : std_logic_vector(9 downto 0);
   signal s6_c : std_logic_vector(9 downto 0);
   signal s7 : std_logic_vector(10 downto 0);
```

```
    signal s7_c : std_logic_vector(10 downto 0);
    signal temp : std_logic_vector(11 downto 0);
    signal WOK_c1 : std_logic;
    signal WOK_c2 : std_logic;
    signal WOK_c3 : std_logic;

begin

    s1    <= ('0' & p1) + ('0' & p2);
    s2    <= ('0' & p3) + ('0' & p4);
    s3    <= ('0' & p5) + ('0' & p6);
    s4    <= ('0' & p7) + ('0' & p8);
    s5    <= ('0' & s1_c) + ('0' & s2_c);
    s6    <= ('0' & s3_c) + ('0' & s4_c);
    s7    <= ('0' & s5_c) + ('0' & s6_c);
   temp<= ('0' & s7_c) + ("0000" & p9_c3);


    arbre_somme : process (clk,rst)
    begin
     if  (rst='1') then
        s1_c<= (others=>'0');
        s2_c<= (others=>'0');
        s3_c<= (others=>'0');
        s4_c<= (others=>'0');
        s5_c<= (others=>'0');
        s6_c<= (others=>'0');
        s7_c<= (others=>'0');
        p9_c1<= (others=>'0');
        p9_c2<= (others=>'0');
        p9_c3<= (others=>'0');
        WOK_c1<= '1';
        WOK_c2<= '1';
        WOK_c3<= '1';
        Data_Ilink_en <= '1';
     elsif (clk='1' and clk'event) then
        s1_c<= s1;
        s2_c<= s2;
        s3_c<= s3;
        s4_c<= s4;
        WOK_c1<= not(WindowOK);
        s5_c<= s5;
        s6_c<= s6;
```

```
        WOK_c2<= WOK_c1;
        s7_c<= s7;
        WOK_c3<= WOK_c2;
        Data_Ilink_en <= WOK_c3;
        p9_c1<= p9;
        p9_c2<= p9_c1;
        p9_c3<= p9_c2;
      end if;
    end process arbre_somme;

    divi8 : process (clk,rst,temp)
    begin
      if (rst='1') then
        pix_flt <= (others=>'0');
      elsif (clk'event and clk = '1') then
        if (temp(11)='1') then
          pix_flt <= (others=>'1');
        else
          pix_flt <= temp(10 downto 3);
        end if;
      end if;
    end process divi8;

end behave;

library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;
```

```
entity compt8re is
port (clk    : in  std_logic;
      rst    : in  std_logic;
      clk_en : in  std_logic;
      q      : out std_logic_vector(7 downto 0);
      parite_out: out std_logic;
      zero   : out std_logic
      );
end compt8re;


architecture behave of compt8re is

signal qin :std_logic_vector(7 downto 0);
signal parite_s : std_logic;

begin

compteur : process (clk,rst,clk_en,qin)
begin
   if (rst='1') then
     qin<=(others=>'0');
     parite_s<='0';
   elsif (clk='1' and clk'event) then
     if (clk_en='1') then
        qin <= qin + 1;
        if (qin=159) then
          qin<=(others=>'0');
          parite_s<=not(parite_s);
        end if;
     end if;
   end if;
end process compteur;

q   <= qin;
parite_out<= parite_s;
zero<= '1' when (qin = 0) else '0';

end behave;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity framer is
    port (
            --=--- Module Interface --=--

            clk         : in std_logic;
            rst         : in std_logic;
            data_in: in std_logic_vector(7 downto 0);
            store_en: in std_logic;
            byte_en: in std_logic_vector(3 downto 0);

            --=-- Frame out --=--

            frame_out: out std_logic_vector(31 downto 0));
end framer;

architecture behavioral of framer is

    -- -=-=-=-=-=-=-=-=-=- --
    -- Signal Declarations --
    -- -=-=-=-=-=-=-=-=-=- --

    signal frame_buf   : std_logic_vector (31 downto 0);


begin

    -- ------------------------- --
    -- ++ -- Basic Assignments -- ++ --

    frame_out <= frame_buf;


    -- ------------------------- --
    -- ++ -- Fifos Description -- ++ --

    FrameBuff : process(clk,rst)
    begin
        if (rst='1') then
                frame_buf <= (others=>'0');
        elsif (clk='1' and clk'event) then
```

```
            if (store_en = '1') then
                if (byte_en(0) = '1') then
                    frame_buf(7 downto 0) <= Data_in;
                end if;

                if (byte_en(1) = '1') then
                    frame_buf(15 downto 8) <= Data_in;
                end if;

                if (byte_en(2) = '1') then
                    frame_buf(23 downto 16) <= Data_in;
                end if;

                if (byte_en(3) = '1') then
                    frame_buf(31 downto 24) <= Data_in;
                end if;
            end if;
        end if;
    end process;
end behavioral;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity ram_fsm is
    port (
        clk     : in std_logic;
        rst     : in std_logic;
        LoadReg: in std_logic;

        Add_clk_en: out std_logic;
        RamWordAvail: out std_logic;
        RAM_WE: out std_logic;
        RAM_OE: out std_logic;
        DATA_en: out std_logic
    );
end ram_fsm;

architecture behave of ram_fsm is

  type ram_state is (PowerUp, Read, WaitLoad, Write, IncAdd);
  attribute enum_encoding : string;
  attribute enum_encoding of ram_state : type is "00001 00010 00100 01000 10000";
  signal present_ram_state, next_ram_state : ram_state := PowerUp;

  signal Frame_out : std_logic_vector(31 downto 0);

begin

  FSMdef: process(clk,rst)
  begin
   if (rst = '1') then
      present_ram_state <= PowerUp;
   elsif (clk'event and clk = '1') then
      present_ram_state <= next_ram_state;
   end if;
  end process;


  FSM1: process(present_ram_state,LoadReg )
  begin
   case present_ram_state is
      when PowerUp =>
       next_ram_state <= Read;
```

```
        when Read =>
          next_ram_state <= WaitLoad;

        when WaitLoad =>
          if (LoadReg='1') then
             next_ram_state <= Write;
          else
             next_ram_state <= WaitLoad;
          end if;

        when Write =>
          next_ram_state <= IncAdd;

        when IncAdd =>
          next_ram_state <= Read;

      end case;
    end process;


    RamWordAvail<= '1' when (present_ram_state = WaitLoad) else '0';
    Add_clk_en<= '1' when (present_ram_state = IncAdd) else '0';

    ------------------------
    -- ECRITURE RAM
    ------------------------
    RAM_WE<= '0' when (present_ram_state = Write) else '1';
    RAM_OE<= '0' when (present_ram_state = Read) else '1';
    DATA_en<= '1' when (present_ram_state = Write or present_ram_state = IncAdd)
else '0';


end behave;
```

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;

entity registres is
port( pix_fifo: in std_logic_vector(31 downto 0);
      pix_lign_impr : in std_logic_vector(31 downto 0);
      pix_lign_pair : in std_logic_vector(31 downto 0);

      clk    : in std_logic;
      rst    : in std_logic;
      clk_en: in std_logic;
      sel    : in std_logic;-- select pour les multiplexeurs

      p1     : out std_logic_vector(7 downto 0);
      p2     : out std_logic_vector(7 downto 0);
      p3     : out std_logic_vector(7 downto 0);
      p4     : out std_logic_vector(7 downto 0);
      p5     : out std_logic_vector(7 downto 0);
      p6     : out std_logic_vector(7 downto 0);
      p7     : out std_logic_vector(7 downto 0);
      p8     : out std_logic_vector(7 downto 0);
      p9     : out std_logic_vector(7 downto 0)
    );
end registres;
architecture behave of registres is
   signal da1 : std_logic_vector(7 downto 0);
   signal da2 : std_logic_vector(7 downto 0);
   signal da3 : std_logic_vector(7 downto 0);
   signal db1 : std_logic_vector(7 downto 0);
   signal db2 : std_logic_vector(7 downto 0);
   signal db3 : std_logic_vector(7 downto 0);
   signal dc1 : std_logic_vector(7 downto 0);
   signal dc2 : std_logic_vector(7 downto 0);
   signal dc3 : std_logic_vector(7 downto 0);
   signal qa1 : std_logic_vector(7 downto 0);
   signal qa2 : std_logic_vector(7 downto 0);
   signal qa3 : std_logic_vector(7 downto 0);
   signal qb1 : std_logic_vector(7 downto 0);
   signal qb2 : std_logic_vector(7 downto 0);
   signal qb3 : std_logic_vector(7 downto 0);
   signal qc1 : std_logic_vector(7 downto 0);
```

```
    signal qc2 : std_logic_vector(7 downto 0);
    signal qc3 : std_logic_vector(7 downto 0);
    signal p1s : std_logic_vector(7 downto 0);
    signal p2s : std_logic_vector(7 downto 0);
    signal p4s : std_logic_vector(7 downto 0);
    signal p5s : std_logic_vector(7 downto 0);
    signal p7s : std_logic_vector(7 downto 0);
    signal p8s : std_logic_vector(7 downto 0);
begin
    reg : process (clk,rst,clk_en,sel)
    begin
     if rst='1' then
        p1s <= (others=>'0');
        p2s <= (others=>'0');
        p3 <= (others=>'0');
        p4s <= (others=>'0');
        p5s <= (others=>'0');
        p6 <= (others=>'0');
        p7s <= (others=>'0');
        p8s <= (others=>'0');
        p9 <= (others=>'0');
        qa1 <= (others=>'0');
        qa2 <= (others=>'0');
        qa3 <= (others=>'0');
        qb1 <= (others=>'0');
        qb2 <= (others=>'0');
        qb3 <= (others=>'0');
        qc1 <= (others=>'0');
        qc2 <= (others=>'0');
        qc3 <= (others=>'0');
     elsif (clk_en='1') then
        if (clk='1' and clk'event) then
         p3 <= p2s;
         p2s <= p1s;
         p1s <= da1;
         qa1 <= db1;
         qb1 <= dc1;
         qc1 <= pix_fifo(31 downto 24);
         p6 <= p5s;
         p5s <= p4s;
         p4s <= da2;
         qa2 <= db2;
         qb2 <= dc2;
```

```
            qc2 <= pix_lign_impr(31 downto 24);
            p9 <= p8s;
            p8s <= p7s;
            p7s <= da3;
            qa3 <= db3;
            qb3 <= dc3;
            qc3 <= pix_lign_pair(31 downto 24);
          else
            null;
          end if;
        else
          null;
        end if;
      end process;
      -- si sel=0 alors le registre circule les pixels
      -- les mux sont ainsi fait pour utiliser les 3 states buffer du FPGA
      da1 <= qa1 when (sel='0') else (others=>'Z');
      da2 <= qa2 when (sel='0') else (others=>'Z');
      da3 <= qa3 when (sel='0') else (others=>'Z');
      db1 <= qb1 when (sel='0') else (others=>'Z');
      db2 <= qb2 when (sel='0') else (others=>'Z');
      db3 <= qb3 when (sel='0') else (others=>'Z');
      dc1 <= qc1 when (sel='0') else (others=>'Z');
      dc2 <= qc2 when (sel='0') else (others=>'Z');
      dc3 <= qc3 when (sel='0') else (others=>'Z');
      -- si sel=1 alors le registre load les nouveaux pixels
      da1 <= pix_fifo(7 downto 0) when (sel='1') else (others=>'Z');
      db1 <= pix_fifo(15 downto 8) when (sel='1') else (others=>'Z');
      dc1 <= pix_fifo(23 downto 16) when (sel='1') else (others=>'Z');
      da2 <= pix_lign_impr(7 downto 0) when (sel='1') else (others=>'Z');
      db2 <= pix_lign_impr(15 downto 8) when (sel='1') else (others=>'Z');
      dc2 <= pix_lign_impr(23 downto 16) when (sel='1') else (others=>'Z');
      da3 <= pix_lign_pair(7 downto 0) when (sel='1') else (others=>'Z');
      db3 <= pix_lign_pair(15 downto 8) when (sel='1') else (others=>'Z');
      dc3 <= pix_lign_pair(23 downto 16) when (sel='1') else (others=>'Z');
      p1 <= p1s;
      p2 <= p2s;
      p4 <= p4s;
      p5 <= p5s;
      p7 <= p7s;
      p8 <= p8s;
    end behave;
```

ANNEXE E

Le VHDL du module de post-traitement

```
LIBRARY ieee;
 USE ieee.std_logic_1164.ALL;
 use IEEE.std_logic_arith.all;
 use IEEE.std_logic_unsigned.all;

entity post_trait is
 port ( Reset   : in  std_logic;
        GenClk1 : in  std_logic;
        ECmdReq : out std_logic;
        CPMSync : in  std_logic;

        FOD     : out std_logic_vector(7 downto 0);
        FORdy   : in  std_logic;
        FOEn    : out std_logic;

        ILINK   : in std_logic_vector(23 downto 0)
      );
end post_trait;


architecture behave of post_trait is

component pixelMAX
port (clk    : in  std_logic;
     rst     : in  std_logic;
     clk_en  : in  std_logic;
     pix_flt : in  std_logic_vector(7 downto 0);
     colon_pos_x: in  std_logic;
     ligne_pos_y: in  std_logic;
     fin_data: in  std_logic;

     pix_max: out std_logic_vector(7 downto 0);
     NewMax: out std_logic;
     somme_pix: out std_logic_vector(26 downto 0);
     rayon   : out std_logic_vector(8 downto 0);
     ray_x   : out std_logic_vector(8 downto 0);
     ray_y   : out std_logic_vector(8 downto 0);
     somme_pix_ray: out std_logic_vector(17 downto 0);
     trait_x : out std_logic;
     trait_y : out std_logic;
     small_pix: out std_logic
     );
end component;
```

```
component compteurXY
port (clk    : in  std_logic;
      rst    : in  std_logic;
      clk_en : in  std_logic;
      NewMax: in  std_logic;

      pos_x  : out std_logic_vector(9 downto 0);
      pos_y  : out std_logic_vector(8 downto 0);
      colon_pos_x: out std_logic;
      ligne_pos_y: out std_logic;
      fin_frame: out std_logic
      );
end component;

component multiplicateur
port(
       clk: in std_logic;
      rst     : in std_logic;
      NewMax: in std_logic;
      clk_en : in std_logic;
       pix_pic: in std_logic_vector(7 downto 0);
      rayon   : in std_logic_vector(8 downto 0);

      somme_mult: out std_logic_vector(24 downto 0)
      );
end component;

component fifo_out
port(

      Engclk     : in std_logic;
      Frame_in: in std_logic_vector(31 downto 0);
      FrameAvail: in std_logic;
      FrameTaken: out std_logic;
      RESET_in: in std_logic;

      FifoEn     : out std_logic;
      FifoRdy    : in std_logic;
      FifoData: out std_logic_vector(7 downto 0)
      );
end component;
```

```
signal pix_flt : std_logic_vector(7 downto 0);
signal Data_Ilink_en : std_logic;
signal colon_pos_x : std_logic;
signal ligne_pos_y : std_logic;
signal fin_frame : std_logic;
signal pix_max : std_logic_vector(7 downto 0);
signal NewMax : std_logic;
signal somme_pix : std_logic_vector(26 downto 0);
signal rayon : std_logic_vector(8 downto 0);
signal ray_x : std_logic_vector(8 downto 0);
signal ray_y : std_logic_vector(8 downto 0);
signal mult_regs_en : std_logic;
signal somme_pix_ray : std_logic_vector(17 downto 0);
signal pos_x : std_logic_vector(9 downto 0);
signal pos_y : std_logic_vector(8 downto 0);
signal somme_mult : std_logic_vector(24 downto 0);
signal Qout : std_logic_vector(1 downto 0);
signal fin_data : std_logic;
signal Frame_in : std_logic_vector(31 downto 0);
signal FrameAvail : std_logic;
signal FrameTaken : std_logic;
signal trait_x : std_logic;
signal trait_y : std_logic;
signal small_pix : std_logic;

type post_state is
(PowerUp,Traitement,LatenceMult,SendWord1,SendWord2,SendWord3,SendWord4,Sen
dWord5,FinFrame);
attribute enum_encoding : string;
attribute enum_encoding of post_state : type is "0001 0010 0100 1000";
signal present_post_state, next_post_state : post_state;


begin


  ECmdReq <= CPMSync;

  entree : process(GenClk1,Reset,ILINK)
  begin
   if (Reset='1') then
      Data_Ilink_en <= '0';
```

```
    pix_flt <= (others=>'0');
  elsif (GenClk1='1' and GenClk1'event) then
    pix_flt <= ILINK(7 downto 0);
    Data_Ilink_en <= not(ILINK(8));
  end if;
end process entree;


FSMdef : process(GenClk1,Reset,next_post_state)
begin
  if (Reset = '1') then
    present_post_state <= PowerUp;
  elsif (GenClk1='1' and GenClk1'event) then
    present_post_state <= next_post_state;
  end if;
end process;


FSM_post: process(present_post_state,fin_frame,Qout,FrameTaken)
begin
  case present_post_state is
    when PowerUp =>
      next_post_state <= Traitement;

    when Traitement =>
      if (fin_frame='1') then
        next_post_state <= LatenceMult;
      else
        next_post_state <= Traitement;
      end if;

  -- LatenceMult pour vider le pipeline du multiplicateur 3 coups d'horloge
    when LatenceMult =>
      if (Qout=3) then
        next_post_state <= SendWord1;
      else
        next_post_state <= LatenceMult;
      end if;

    when SendWord1 =>
      if (FrameTaken='1') then
        next_post_state <= SendWord2;
      else
```

```
            next_post_state <= SendWord1;
          end if;

        when SendWord2 =>
          if (FrameTaken='1') then
            next_post_state <= SendWord3;
          else
            next_post_state <= SendWord2;
          end if;

        when SendWord3 =>
          if (FrameTaken='1') then
            next_post_state <= SendWord4;
          else
            next_post_state <= SendWord3;
          end if;

        when SendWord4 =>
          if (FrameTaken='1') then
            next_post_state <= SendWord5;
          else
            next_post_state <= SendWord4;
          end if;

        when SendWord5 =>
          if (FrameTaken='1') then
            next_post_state <= FinFrame;
          else
            next_post_state <= SendWord5;
          end if;

        when FinFrame =>
          next_post_state <= Traitement;

      end case;
    end process FSM_post;

muxxx :
process(present_post_state,pix_max,pos_x,pos_y,somme_pix,ray_x,ray_y,somme_mult,s
omme_pix_ray)
    begin
      case present_post_state is
```

```
        when SendWord1 =>
          Frame_in(26 downto 0) <= pos_y & pos_x & pix_max;

        when SendWord2 =>
          Frame_in(26 downto 0) <= somme_pix;

        when SendWord3 =>
          Frame_in(17 downto 0) <= ray_y & ray_x;

        when SendWord4 =>
          Frame_in(24 downto 0) <= somme_mult;

        when SendWord5 =>
          Frame_in(17 downto 0) <= somme_pix_ray;

        when others =>
          Frame_in <= (others=>'X');
      end case;
    end process muxxx;


    mult_regs_en <= '1' when ((trait_x='1' and ligne_pos_y='1' and small_pix='0' and
Data_Ilink_en='1') or
                    (trait_y='1' and colon_pos_x='1' and small_pix='0' and Data_Ilink_en='1')
or
                    present_post_state=LatenceMult) else '0';

    fin_data<= '1' when (present_post_state=FinFrame) else '0';

    FrameAvail<= '1' when (present_post_state=SendWord1 or
                    present_post_state=SendWord2 or
                    present_post_state=SendWord3 or
                    present_post_state=SendWord4 or
                    present_post_state=SendWord5) else '0';


    cmpt_latence_mult : process (GenClk1,Reset,present_post_state,Qout)
    begin
      if (Reset='1') then
        Qout <= (others =>'0');
      elsif (present_post_state=LatenceMult) then
        if (GenClk1='1' and GenClk1'event) then
          Qout <= Qout + 1;
```

```
      end if;
    end if;
  end processcmpt_latence_mult;


  U_pixelMAX : pixelMAX port map
(GenClk1,Reset,Data_Ilink_en,pix_flt,colon_pos_x,ligne_pos_y,fin_data,pix_max,New
Max,

somme_pix,rayon,ray_x,ray_y,somme_pix_ray,trait_x,trait_y,small_pix);

  U_compteurXY : compteurXY port map
(GenClk1,Reset,Data_Ilink_en,NewMax,pos_x,pos_y,colon_pos_x,ligne_pos_y,fin_fram
e);

  U_multiplicateur : multiplicateur port map
(GenClk1,Reset,NewMax,mult_regs_en,pix_flt,rayon,somme_mult);

  U_FifoOut : fifo_out port map
(GenClk1,Frame_in,FrameAvail,FrameTaken,Reset,FOEn,FORdy,FOD);


end behave;
```

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;

entity compteurXY is
port (clk    : in  std_logic;
      rst    : in  std_logic;
      clk_en : in  std_logic;
      NewMax: in  std_logic;

      pos_x  : out std_logic_vector(9 downto 0);
      pos_y  : out std_logic_vector(8 downto 0);
      colon_pos_x: out std_logic;
      ligne_pos_y: out std_logic;
      fin_frame: out std_logic
      );
end compteurXY;


architecture behave of compteurXY is

signal cmpt_en_y : std_logic;
signal cmpt_x: std_logic_vector(9 downto 0);
signal cmpt_y: std_logic_vector(8 downto 0);
signal pos_x_s: std_logic_vector(9 downto 0);
signal pos_y_s: std_logic_vector(8 downto 0);

begin

position_x : process (clk,rst,clk_en,cmpt_x)
begin
   if (rst='1') then
     cmpt_x <= "0000000010";
   elsif (clk_en='1') then
     if (clk='1' and clk'event) then
        if (cmpt_x=639) then
          cmpt_x <= "0000000010";
        else
          cmpt_x <= cmpt_x + 1;
        end if;
     end if;
   end if;
```

```
end process position_x;

position_y : process (clk,rst,cmpt_en_y,cmpt_y)
begin
   if (rst='1') then
    cmpt_y <= "000000010";
   elsif (cmpt_en_y='1') then
    if (clk='1' and clk'event) then
       if (cmpt_y=479) then
        cmpt_y <= "000000010";
       else
        cmpt_y <= cmpt_y + 1;
       end if;
    end if;
   end if;
end process position_y;

cmpt_en_y<= '1' when (cmpt_x=639 and clk_en='1') else '0';
fin_frame<= '1' when (cmpt_y=479 and cmpt_en_y='1') else '0';

nouveau_max : process (clk,rst,clk_en,NewMax,cmpt_x,cmpt_y)
begin
   if (rst='1') then
    pos_x_s <= (others =>'0');
    pos_y_s <= (others =>'0');
   elsif (NewMax='1' and clk_en='1' ) then
    if (clk='1' and clk'event) then
       pos_x_s <= cmpt_x;
       pos_y_s <= cmpt_y;
    end if;
   end if;
end process nouveau_max;

pos_x <= pos_x_s;
pos_y <= pos_y_s;
colon_pos_x <= '1' when (cmpt_x = pos_x_s) else '0';
ligne_pos_y <= '1' when (cmpt_y = pos_y_s) else '0';

end behave;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity fifo_out is
    port (

        Engclk     : in std_logic;
        Frame_in: in std_logic_vector(31 downto 0);
        FrameAvail: in std_logic;
        FrameTaken: out std_logic;
        RESET_in: in std_logic;

        FifoEn     : out std_logic;
        FifoRdy    : in std_logic;
        FifoData: out std_logic_vector(7 downto 0));
end fifo_out;

architecture behavioral of fifo_out is

    type output_states_def is (PowerUp, DummyWait, WaitForAvail, WaitForRdy,
SendByte2, SendByte3, SendByte4 );
    attribute enum_encoding : string;
    attribute enum_encoding of output_states_def: type is "0000001 0000010 0000100
0001000 0010000 0100000 1000000";
    signal present_out_state, next_out_state: output_states_def;

    signal load_sig, shift_sig: std_logic;


    component Shifter
    port (
            clk     : in std_logic;
            rst     : in std_logic;
            frame_in: in std_logic_vector(31 downto 0);
            load_en: in std_logic;
            shift_en: in std_logic;
            data_out: out std_logic_vector(7 downto 0)
    );
    end component;


begin
```

```
FSMdef: process(engclk, RESET_in)
begin
    if (RESET_in = '1') then
        present_out_state <= PowerUp;
    elsif (engclk='1' AND engclk 'EVENT) then
        present_out_state <= next_out_state;
    end if;
end process;


FSM1: process(present_out_state, FifoRdy, FrameAvail)
begin
    case present_out_state is
        when PowerUp =>
            next_out_state <= DummyWait;

        when DummyWait =>
          if (FrameAvail='1') then
            next_out_state <= WaitForAvail;
          else
            next_out_state <= DummyWait;

          end if;

        when WaitForAvail =>
            if (FrameAvail = '1') then
                next_out_state <= WaitForRdy;
            else
                next_out_state <= WaitForAvail;
            end if;

        when WaitForRdy =>
            if (FifoRdy = '1') then
                next_out_state <= SendByte2;
            else
                next_out_state <= WaitForRdy;
            end if;

        when SendByte2 =>
            next_out_state <= SendByte3;

        when SendByte3 =>
            next_out_state <= SendByte4;
```

```
            when SendByte4 =>
                next_out_state <= DummyWait;

        end case;
    end process;


    FifoEn <= '0' when ((present_out_state = WaitForRdy) or
                (present_out_state = SendByte2) or
                (present_out_state = SendByte3) or
                (present_out_state = SendByte4)) else '1';


    -- avant c'etait a present_out_state= WaitForAvail
    FrameTaken <= '1' when ( (FrameAvail = '1') and
                        (present_out_state = WaitForRdy)) else '0';

    load_sig <= '1' when ( (FrameAvail = '1') and
                        (present_out_state = WaitForAvail) ) else '0';

    shift_sig <= '1' when ( (present_out_state = WaitForRdy and FifoRdy = '1') or
                        (present_out_state = SendByte2) or
                        (present_out_state = SendByte3) or
                        (present_out_state = SendByte4) ) else '0';


    Shft : Shifter port map
        (
            engclk,
            RESET_in,
            Frame_in,
            load_sig,
            shift_sig,
            fifoData
        );


end behavioral;
```

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;

entity multiplicateur is
port(
      clk: in std_logic;
     rst       : in std_logic;
     NewMax: in std_logic;
     clk_en : in std_logic;
      pix_pic: in std_logic_vector(7 downto 0);
     rayon   : in std_logic_vector(8 downto 0);

     somme_mult: out std_logic_vector(24 downto 0)
     );
end multiplicateur;

architecture behave of multiplicateur is

signal s1 : std_logic_vector(9 downto 0);
signal s2 : std_logic_vector(9 downto 0);
signal s3 : std_logic_vector(9 downto 0);
signal s4 : std_logic_vector(9 downto 0);
signal s5 : std_logic_vector(11 downto 0);
signal s6 : std_logic_vector(11 downto 0);
signal s7 : std_logic_vector(15 downto 0);
signal s8 : std_logic_vector(24 downto 0);
signal somme_mult_s : std_logic_vector(24 downto 0);

signal s1_c : std_logic_vector(9 downto 0);
signal s2_c : std_logic_vector(9 downto 0);
signal s3_c : std_logic_vector(9 downto 0);
signal s4_c : std_logic_vector(9 downto 0);
signal s5_c : std_logic_vector(11 downto 0);
signal s6_c : std_logic_vector(11 downto 0);
signal s7_c : std_logic_vector(15 downto 0);

signal m1 : std_logic_vector(7 downto 0);
signal m2 : std_logic_vector(7 downto 0);
signal m3 : std_logic_vector(7 downto 0);
signal m4 : std_logic_vector(7 downto 0);
signal m5 : std_logic_vector(7 downto 0);
```

```
signal m6 : std_logic_vector(7 downto 0);
signal m7 : std_logic_vector(7 downto 0);
signal m8 : std_logic_vector(7 downto 0);
signal m9 : std_logic_vector(7 downto 0);
signal m9_c1 : std_logic_vector(7 downto 0);
signal m9_c2 : std_logic_vector(7 downto 0);
signal m9_c3 : std_logic_vector(7 downto 0);
signal ray_pix : std_logic_vector(16 downto 0);
signal ray_pix_c : std_logic_vector(16 downto 0);

begin

   m1<= pix_pic when rayon(0)='1' else (others=>'0');
   m2<= pix_pic when rayon(1)='1' else (others=>'0');
   m3<= pix_pic when rayon(2)='1' else (others=>'0');
   m4<= pix_pic when rayon(3)='1' else (others=>'0');
   m5<= pix_pic when rayon(4)='1' else (others=>'0');
   m6<= pix_pic when rayon(5)='1' else (others=>'0');
   m7<= pix_pic when rayon(6)='1' else (others=>'0');
   m8<= pix_pic when rayon(7)='1' else (others=>'0');
   m9<= pix_pic when rayon(8)='1' else (others=>'0');

   s1<=(("00" & m1(7 downto 1)) + ('0' &  m2)) & m1(0);
   s2<=(("00" & m3(7 downto 1)) + ('0' &  m4)) & m3(0);
   s3<=(("00" & m5(7 downto 1)) + ('0' &  m6)) & m5(0);
   s4<=(("00" & m7(7 downto 1)) + ('0' &  m8)) & m7(0);

   s5<=(("00" & s1_c(9 downto 2)) + s2_c) & s1_c(1 downto 0);
   s6<=(("00" & s3_c(9 downto 2)) + s4_c) & s3_c(1 downto 0);

   s7<=(("0000" & s5_c(11 downto 4)) + s6_c) & s5_c(3 downto 0);

   ray_pix <=(('0' & s7_c(15 downto 8)) + ('0' & m9_c3)) & s7_c(7 downto 0);

   s8<=("00000000" & ray_pix_c) + somme_mult_s;

   somme_mult <= somme_mult_s;

   arbre_mult : process
(clk,rst,clk_en,NewMax,s1,s2,s3,s4,s5,s6,s7,s8,m9,m9_c1,m9_c2,ray_pix)
   begin
    if  (rst='1') then
       s1_c<= (others=>'0');
```

```
        s2_c<= (others=>'0');
        s3_c<= (others=>'0');
        s4_c<= (others=>'0');
        s5_c<= (others=>'0');
        s6_c<= (others=>'0');
        s7_c<= (others=>'0');
        m9_c1<= (others=>'0');
        m9_c2<= (others=>'0');
        m9_c3<= (others=>'0');
        ray_pix_c<= (others=>'0');
        somme_mult_s <= (others=>'0');
      elsif (clk='1' and clk'event) then
        if (NewMax='1') then
          s1_c<= (others=>'0');
          s2_c<= (others=>'0');
          s3_c<= (others=>'0');
          s4_c<= (others=>'0');
          s5_c<= (others=>'0');
          s6_c<= (others=>'0');
          s7_c<= (others=>'0');
          m9_c1<= (others=>'0');
          m9_c2<= (others=>'0');
          m9_c3<= (others=>'0');
          ray_pix_c    <= (others=>'0');
          somme_mult_s <= (others=>'0');
        elsif (clk_en='1') then
          s1_c<= s1;
          s2_c<= s2;
          s3_c<= s3;
          s4_c<= s4;
          s5_c<= s5;
          s6_c<= s6;
          s7_c<= s7;
          m9_c1<= m9;
          m9_c2<= m9_c1;
          m9_c3<= m9_c2;
          ray_pix_c <= ray_pix;
          somme_mult_s <= s8;
        end if;
      end if;
    end process arbre_mult;

end behave;
```

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;

entity pixelMAX is
port (clk    : in  std_logic;
      rst    : in  std_logic;
      clk_en : in  std_logic;
      pix_flt : in  std_logic_vector(7 downto 0);
      colon_pos_x: in  std_logic;
      ligne_pos_y: in  std_logic;
      fin_data: in  std_logic;

      pix_max: out std_logic_vector(7 downto 0);
      NewMax: out std_logic;
      somme_pix: out std_logic_vector(26 downto 0);
      rayon  : out std_logic_vector(8 downto 0);
      ray_x  : out std_logic_vector(8 downto 0);
      ray_y  : out std_logic_vector(8 downto 0);
      somme_pix_ray: out std_logic_vector(17 downto 0);
      trait_x : out std_logic;
      trait_y : out std_logic;
      small_pix: out std_logic
      );
end pixelMAX;

architecture behave of pixelMAX is

   signal NewMax_s : std_logic;
   signal small_pix_s : std_logic;
   signal ray_x_s : std_logic_vector(8 downto 0);
   signal ray_y_s : std_logic_vector(8 downto 0);
   signal inc_ray_x : std_logic;
   signal inc_ray_y : std_logic;
   signal trait_x_s : std_logic;
   signal trait_y_s : std_logic;
   signal val_lim : std_logic_vector(5 downto 0);
   signal pix_max_s : std_logic_vector(7 downto 0);
   signal somme_pix_ray_s : std_logic_vector(17 downto 0);
   signal somme_pix_ray_en : std_logic;
   signal somme_pix_s : std_logic_vector(26 downto 0);
```

```
begin

   nouveau_max : process (clk,rst,fin_data,clk_en,NewMax_s,pix_flt)
   begin
    if (rst='1') then
       pix_max_s <= (others =>'0');
    elsif (clk='1' and clk'event) then
       if (fin_data='1') then
        pix_max_s <= (others =>'0');
       elsif (clk_en='1' and NewMax_s='1') then
        pix_max_s <= pix_flt;
       end if;
    end if;
   end process nouveau_max;


   traitements : process
(clk,rst,fin_data,clk_en,NewMax_s,small_pix_s,colon_pos_x,ligne_pos_y,ray_x_s)
   begin
    if (rst='1') then
       trait_x_s <= '0';
       trait_y_s <= '0';
    elsif (clk='1' and clk'event) then
       if (fin_data='1') then
        trait_x_s <= '0';
        trait_y_s <= '0';
       elsif (clk_en='1') then
        if (NewMax_s='1') then
           trait_x_s <= '1';
           trait_y_s <= '1';
        elsif ((small_pix_s='1' and ligne_pos_y='1') or ray_x_s=511) then
           trait_x_s <= '0';
        elsif (small_pix_s='1' and colon_pos_x='1') then
           trait_y_s <= '0';
        end if;
       end if;
    end if;
   end process traitements;

   val_lim<= pix_max_s(7 downto 2);
   NewMax_s<= '1' when (pix_flt > pix_max_s) else '0';
   NewMax<= NewMax_s;
   small_pix_s<= '1' when (("00" & val_lim) >= pix_flt) else '0';
```

```
    small_pix<= small_pix_s;
    pix_max<= pix_max_s;
    trait_x<= trait_x_s;
    trait_y<= trait_y_s;

    rayons : process (clk,rst,NewMax_s,clk_en,inc_ray_x,inc_ray_y,ray_x_s,ray_y_s)
    begin
     if (rst='1') then
        ray_x_s <= "000000001";
        ray_y_s <= "000000001";
     elsif (clk_en='1') then
        if (clk='1' and clk'event) then
         if (NewMax_s='1') then
            ray_x_s <= "000000001";
            ray_y_s <= "000000001";
         elsif (inc_ray_x='1') then
            ray_x_s <= ray_x_s + 1;
         elsif (inc_ray_y='1') then
            ray_y_s <= ray_y_s + 1;
         end if;
        end if;
     end if;
    end process rayons;

    somme_pix_ray_en<= '1' when ((trait_x_s='1' and ligne_pos_y='1' and
small_pix_s='0') or
                    (trait_y_s='1' and colon_pos_x='1' and small_pix_s='0')) else '0';

    inc_ray_x<= '1' when (trait_x_s='1' and ligne_pos_y='1' and small_pix_s='0' and
ray_x_s/=511) else '0';
    inc_ray_y<= '1' when (trait_y_s='1' and colon_pos_x='1' and small_pix_s='0') else '0';
    rayon<= ray_x_s when (ligne_pos_y='1') else ray_y_s;
    ray_x<= ray_x_s;
    ray_y<= ray_y_s;

    sommateur_pix_ray :
process(clk,rst,clk_en,NewMax_s,somme_pix_ray_en,somme_pix_ray_s,pix_flt)
    begin
     if (rst='1') then
        somme_pix_ray_s <= (others =>'0');
     elsif (clk_en='1') then
        if (clk='1' and clk'event) then
         if (NewMax_s='1') then
```

```
            somme_pix_ray_s <= (others =>'0');
          elsif (somme_pix_ray_en='1') then
            somme_pix_ray_s <= somme_pix_ray_s + pix_flt;
          end if;
        end if;
      end if;
    end process sommateur_pix_ray;

    somme_pix_ray <= soname_pix_ray_s;


    sommateur_pixels : process(clk,rst,clk_en,fin_data,somme_pix_s,pix_flt)
    begin
      if (rst='1') then
        somme_pix_s <= (others =>'0');
      elsif (clk_en='1') then
        if (clk='1' and clk'event) then
          if (fin_data='1') then
            somme_pix_s <= (others =>'0');
          else
            somme_pix_s <= somme_pix_s + pix_flt;
          end if;
        end if;
      end if;
    end process sommateur_pixels;

    somme_pix <= somme_pix_s;

end behave;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity shifter is
    port (
        --=-- Module Interface --=--
        clk      : in std_logic;
        rst      : in std_logic;
        frame_in: in std_logic_vector(31 downto 0);
        load_en: in std_logic;
        shift_en: in std_logic;
        --=-- Data out --=--
        data_out: out std_logic_vector(7 downto 0)
    );
end shifter;
architecture behavioral of shifter is
    -- -=-=-=-=-=-=-=-=-=-=- --
    -- Signal Declarations --
    -- -=-=-=-=-=-=-=-=-=-=- --
    signal frame_buf   : std_logic_vector (31 downto 0);
begin
    -- ------------------------- --
    -- ++ -- Basic Assignments -- ++ --
    data_out <= frame_buf(7 downto 0);
    -- ------------------------- --
    -- ++ -- Fifos Description -- ++ --
    FrameBuff : process(clk,rst)
    begin
        IF (rst = '1') THEN
            frame_buf<="00000000000000000000000000000000";
        elsif (clk='1' AND clk 'EVENT) then
            if (load_en = '1') then
                frame_buf <= frame_in;
            elsif (shift_en = '1') then
                frame_buf(23 downto 0) <= frame_buf(31 downto 8);
            end if;
        end if;
    end process;
end behavioral;
```