Validation temps réel d'algorithmes de traitement d'images sur une architecture reconfigurable : application au codage d'images et à la détection de mouvements

Messaoud Mostefai, Brahim Zitouni,
Institut d'Informatique
Université Ferhat ABBAS - Setif - Algérie –
Pr Michel Roussel Université de Reims (France)
Email : espace@mail.wissal.dz

1. Introduction

L'évolution des applications de traitement d'images durant ces dernières années fut marquée par la nécessité d'utiliser de plus en plus de systèmes répondants à des exigences accrues en termes de temps d'exécution et de ébit [1] [2] [3]. La validation de ces systèmes comporte en général les phases classiques d'élaboration, d'expérimentation logicielle voir même d'implantation matérielle. Cette dernière phase nécessite souvent l'utilisation de circuits spécialisés ASIC qui en général dépassent de loin les performances des microprocesseurs tout en restant à des coûts abordables [4].

Bien que ce type de circuits soit avantageux pour une certaine classe de traitement, il reste difficilement généralisable sur l'ensemble des applications de traitement d'images, car en général, chaque application nécessite un parallélisme, un pipelinage et des opérations arithmétiques différents, d'où la nécessité de définir une nouvelle architecture lorsque l'on change d'application [5] [6].

La disponibilité des circuits reconfigurables permet de définir une architecture flexible, capable de valider un grand nombre d'applications sans avoir besoin d'effectuer d'importantes modifications pour passer d'une application à une autre [7] [8].

Nous présentons dans ce travail une expérience de prototypage d'un codeur d'images temps réel sur une plate-forme flexible basée sur des circuits FPGA. Le codeur met en œuvre deux algorithmes : l'un pour la compression d'images (en vue de l'archivage) ; l'autre pour l'extraction de la silhouette des objets mobiles.

2. Présentation du codeur d'images temps réel

Le développement des systèmes de télésurveillance présente un intérêt considérable dans le domaine de la sécurité. Ces derniers sont construits en fonction des contraintes qui leurs sont imposées par l'application ou le site à surveiller. Mais dans la plupart des cas, la compression d'images est utilisée pour diminuer la taille physique de l'information à transmettre ou à stocker. Dans le cas d'une transmission, les contraintes essentielles sont le temps réel et le débit du canal de transmission, ce qui impose souvent, pour la réalisation matérielle, une limitation au niveau de la complexité de la méthode de compression utilisée. Par contre pour ce qui est des images stockées leur transmission peut se faire en temps différé, ce qui permet d'utiliser des techniques de compression plus complexes et plus performantes. Les systèmes de surveillance autonomes sont généralement composés d'un ensemble de caméras et de capteurs placés dans des lieux à surveiller et qui déclenchent dans le cas d'une détection suspecte une alarme (souvent transmise aux lieux de contrôle et d'intervention par voie téléphonique) ainsi qu'une opération d'archivage (en utilisant une technique de compression). Bien qu'ils soient rapides pour la prévention, les capteurs utilisés ne peuvent pas donner une information sur la nature de l'intrus qui a déclenché l'alarme. En plus la transmission des images comprimées est souvent ralentie par le faible débit des canaux de transmission.

Il est donc intéressant de disposer d'une information visuelle qui puisse nous renseigner de façon rapide sur la nature de l'intrus. En d'autres termes, utiliser une méthode qui permet d'atteindre des taux de compression importants qu'on ne peut atteindre avec les méthodes de compression classiques (en utilisant le codage par bloc, par différenciation, par transformation, etc.) et garde l'information utile à l'identification de la nature de l'intrus. Dans ces cas l'approche contour est la technique la plus utilisée. Elle consiste à transformer l'image sous forme de lignes de contours qui ne représentent qu'un faible pourcentage de la totalité de l'image, d'ou l'intérêt de les extraire pour modéliser les images.

Le codeur proposé met en œuvre les deux algorithmes dont on a parlé : un algorithme pour l'extraction de la silhouette des objets mobiles (à transmettre par ligne téléphonique) ; et un algorithme pour la compression d'images (en vue de l'archivage). La reconfigurabilité sur site de cette technologie permet d'effectuer le téléchargement de l'un des deux algorithmes dans le circuit FPGA.

3. Présentation de la plate- forme de développement

La plate-forme de développement (figure.1) est composée d'un ensemble de cartes électroniques travaillants sur un bus VME pour l'acquisition (ADI), la mémorisation (FB) et le traitement des images en temps réel (à la cadence vidéo). La nature asynchrone du bus permet de faire cohabiter des cartes processeurs (CPU), des mémoires et des périphériques présentant des caractéristiques de vitesses différentes [9].

L'utilisateur a à sa disposition des connecteurs libres lui permettant d'introduire des cartes développées (CAS : **C**arte d'**A**pplication **S**pécifiques) pour effectuer des traitements matériels. Dans notre travail on utilise le connecteur qui contient les données vidéo, les données mémoire, les signaux horloge, les signaux de synchronisation ainsi que les alimentations. Les FPGAs qui seront câblés sur notre carte d'applications spécifiques utilisent différents signaux d'initialisation et de cadrage et ce afin d'opérer dans le domaine de validité du signal vidéo. La carte d'applications spécifiques (CAS) construite autour d'FPGAs utilise des mémoires RAM pour le désentrelacement des images avant traitement, et leur réentrelacement après traitement (pour affichage); les mémoires FIFO sont utilisées pour effectuer des retards de lignes et ceci dans le cas de traitements qui opèrent simultanément sur plusieurs lignes de l'image. En plus des mémoires de désentrelacement/ réentrelacement et des FIFO les FPGAs ont accès aux plans mémoire de la carte FB, ce qui permet d'effectuer des traitement sur des images consécutives.

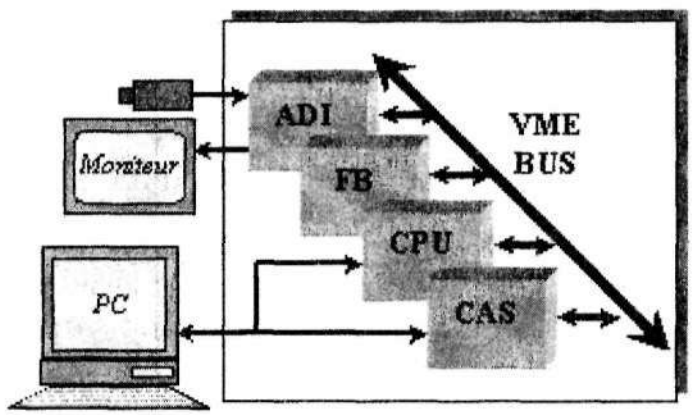

Fig.1 *Plate-forme de développement*

3. Présentation de la plate- forme de développement

La plate-forme de développement (figure.1) est composée d'un ensemble de cartes électroniques travaillants sur un bus VME pour l'acquisition (ADI), la mémorisation (FB) et le traitement des images en temps réel (à la cadence vidéo). La nature asynchrone du bus permet de faire cohabiter des cartes processeurs (CPU), des mémoires et des périphériques présentant des caractéristiques de vitesses différentes [9].

L'utilisateur a à sa disposition des connecteurs libres lui permettant d'introduire des cartes développées (CAS : **C**arte d'**A**pplication **S**pécifiques) pour effectuer des traitements matériels. Dans notre travail on utilise le connecteur qui contient les données vidéo, les données mémoire, les signaux horloge, les signaux de synchronisation ainsi que les alimentations. Les FPGAs qui seront câblés sur notre carte d'applications spécifiques utilisent différents signaux d'initialisation et de cadrage et ce afin d'opérer dans le domaine de validité du signal vidéo. La carte d'applications spécifiques (CAS) construite autour d'FPGAs utilise des mémoires RAM pour le désentrelacement des images avant traitement, et leur réentrelacement après traitement (pour affichage); les mémoires FIFO sont utilisées pour effectuer des retards de lignes et ceci dans le cas de traitements qui opèrent simultanément sur plusieurs lignes de l'image. En plus des mémoires de désentrelacement/ réentrelacement et des FIFO les FPGAs ont accès aux plans mémoire de la carte FB, ce qui permet d'effectuer des traitement sur des images consécutives.

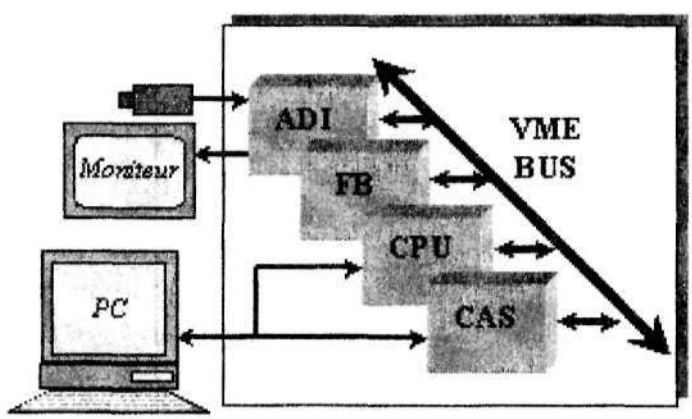

Fig.1 *Plate-forme de développement*

4. Les FPGAs

Un FPGA est constituée d'une matrice de blocs reconfigurables, entourés par des blocs d'entrée sortie (Fig.2). La régularité de répartition des blocs reconfigurables, qui en plus sont identiques, facilite la tâche de partitionnement, de placement et routage des applications développées.

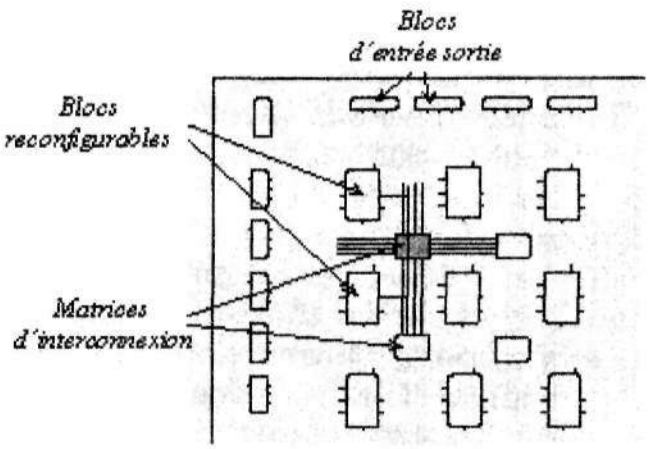

Fig.2 *Structure interne d'une partie d'un FPGA*

Les outils de programmation fournis avec les FPGAs permettent de définir une application sous forme de schéma ou d'équations. A partir de ces définitions, le logiciel calcule la configuration des blocs et leurs interconnexions; il assure enfin le transfert de ces informations dans le FPGA [10][11].

5. Recherche Algorithmique

Dans le cas de séquences d'images la quantité d'information traitée est considérable. D'importantes réductions peuvent être obtenues en exploitant aussi bien la corrélation spatiale entre pixels de l'image que la corrélation temporelle entre images successives [12].

Une étude détaillée des différentes techniques de réduction de la quantité d'informations dans les séquences d'images nous a permis de choisir la technique de compression par transformation pour l'archivage et l'approche contour pour l'extraction de la silhouette des objets mobile en vue d'une transmission rapide par ligne téléphonique.

6. Implantation de l'algorithme de compression par transformation

Dans les techniques de compression par transformation les images sont découpées en blocs (généralement de taille 4x4, 8x8 ou 16x16) pour exploiter la corrélation qui existe entre pixels voisins. Les transformations usuelles (la transformée en cosinus discrète notée *TCD*, les transformations de *Hadamard* et de *Haar*) produisent un ensemble de coefficients transformés *relativement décorrelés*, pouvant être efficacement quantifiés et codés [13].

Il est possible d'avoir des coefficients complètement décorrélés et ce en utilisant la transformée de *Karhunen-Loève*. Mais à cause de sa complexité cette dernière est rarement utilisée.

Le formalisme général d'une transformation linéaire de signaux bidimensionnels est le suivant :

$$X(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a(m,n,k,l)\, x(k,l) \quad (1)$$

avec $m,n = 0,\ldots\ldots\ldots\ldots N-1$

$a(m,n,k,l)$ est l'élément général d'un tableau $A(m,n,k,l)$ appelé *noyau*. Presque la totalité des transformations possèdent des noyaux séparables et symétriques [14].

$$A[m,n,k,l] = A'[m,k]\, A'[n,l] \quad (2)$$

Le principe d'une compression/décompression par transformation est présenté en figure.3. Le partitionnement de l'image est suivi d'un changement de représentation, d'une réduction irréversible du débit (Quantification) et d'une mise en forme par codage entropique réversible. L'opération de quantification consiste à transmettre de façon plus ou moins précise les amplitudes des coefficients.

Pour obtenir un débit spécifié, une mémoire tampon est intercalée avant la sortie et une régulation vient contrôler le fonctionnement de la partie qui réduit le débit en faisant varier les paramètres de quantification. Le récepteur effectue les opérations en sens inverse: décodage entropique, changement de représentation inverse, et enfin reconstruction de l'image.

Dans l'optique d'une adaptation de l'algorithme à l'architecture dont nous disposons, la transformée de Hadamard sera retenue ; elle ne fait appel qu'à des additions et soustractions beaucoup plus faciles à implanter que des multiplications. Par ailleurs elle donne des résultats peu inférieurs à la classique TCD.

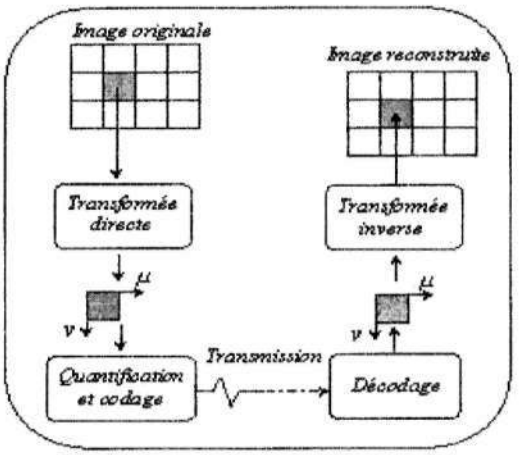

Fig.3 *Chaîne de compression/décompression*

Pour son implantation, la transformée de Hadamard 2-D est décomposée en trois parties (figure.4). La première partie réalise le Hadamard 1-D, la partie du milieu est une mémoire de transposition. La dernière partie réalise un deuxième Hadamard 1-D. Le rôle de la mémoire de transposition est de stocker les données en provenance du premier Hadamard 1-D et les reséquencer pour le deuxième Hadamard 1-D dans l'ordre correct [12].

L'architecture d'un bloc Hadamard 1-D pour des sous images de taille 8x8 est présentée en figure.5. Il est constitué de huit ALU qui travaillent en parallèle sur les mêmes données en entrée. Chaque ALU est constituée d'un A/S (Additionneur/Soustracteur) et de deux étages de mémorisation. La taille de chaque (A/S) est choisie de façon à pouvoir effectuer la totalité des opérations d'une ligne ou colonne de la matrice de Hadamard. Les résultats intermédiaires sont stockés dans le premier étage de mémorisation puis réinjectés à l'entrée des (A/S) afin d'effectuer les opérations avec les données suivantes. Une fois les huit données acquises, on stocke les résultats finaux présents à la sortie du

premier étage dans un deuxième étage puis on refait l'opération sur les huit données suivantes; entre temps on évacue les résultats du deuxième étage.

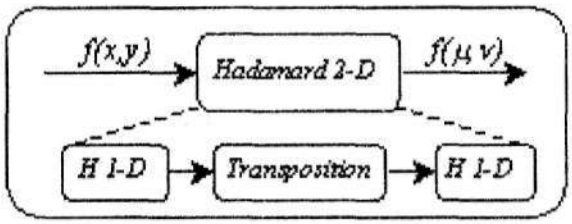

Fig.4 *Transformée de Hadamard 2-D*

Le rôle de la mémoire de transposition est de stocker les données en provenance du premier Hadamard 1-D et les reséquencer pour le deuxième Hadamard 1-D dans l'ordre correct. Les circuits FPGA de la famille XC40XX offrent la possibilité de configurer les LUT des CLB en mémoires SRAM (de 32 bits chacun) [6], et ont l'avantage d'être plus rapides que les mémoires classiques permettant ainsi d'accroître de façon considérable les performances des applications réalisées sur des FPGA et nécessitant de la mémoire pour effectuer leurs traitements. Le coût d'implantation de la transformée de Hadamard directe sur FPGA 40XX est de 283 CLBs.

6.1 Vers un compromis surface/performance

Ayant restreint le champ de travail de notre codeur à des applications de télésurveillance qui en général n'exigent pas des images reconstruites de très bonne qualité, nous avons introduit des modifications dans l'algorithme de transformation de Hadamard de façon à réduire le coût en surface d'implantation tout en offrant des résultats satisfaisant.

Une étude de la répartition des coefficients sur les 64 emplacements d'une matrice transformée, montre que l'on conserve l'information essentielle de l'image en ne tenant compte que des coefficients situés dans le coin supérieur gauche de chaque bloc transformé (figure.6). Ces valeurs représentent les fréquences basses possédant une forte amplitude. En revanche, les autres coefficients présentent moins d'intérêt, d'une part parce qu'ils sont majoritairement faibles, d'autre part les rares coefficients d'amplitude importante correspondent aux

hautes fréquences de l'image, c'est à dire aux détails. Ne pas tenir compte de ces coefficients au moment de la compression permet de restituer lors de la décompression l'information principale de l'image [13].

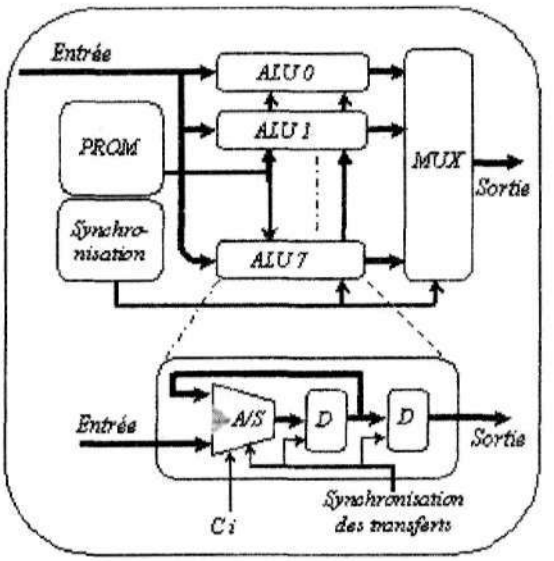

Fig.5 Bloc Hadamard 1-D

La technique de sélection zonale intervient en aval de la transformation ; elle permet certes de réduire la taille du code à transmettre mais n'a aucun impacte sur la structure de calcul des coefficients. Dans ce qui suit on introduira la transformation de Hadamard réduite (THR) qui est une transformation à présélection zonale. Cette dernière permet un réduction considérable dans la structure de calcul matérielle tout en offrant des résultats satisfaisants du point de vue taux de compression et qualité des images reconstruites.

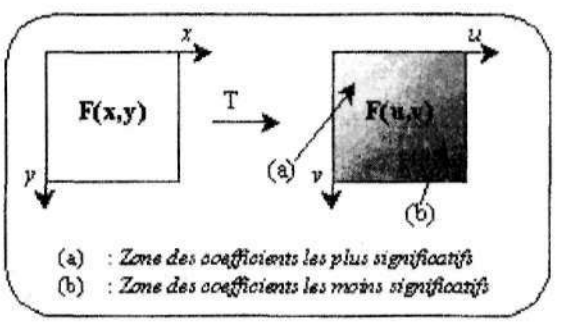

Fig.6 *Répartition des coefficients*

6.2 Transformée de Hadamard réduite (THR)

Afin d'effectuer une préselection des coefficients à coder, on procédera par l'élimination des éléments de la matrice de Hadamard qui induisent au calcul des coefficients non importants Il est donc possible d'avoir plusieurs niveaux de réduction dans la matrice de Hadamard et ce en fonction de la qualité de l'image exigée après reconstruction. Le niveau 1 indique que les matrices de Hadamard utilisées comportent une ligne ou une colonne nulle. Le niveau 2 indique que les matrices de Hadamard utilisées comportent deux lignes ou deux colonnes nulles (Fig.7) et ainsi de suite. La zone des coefficients sélectionnés est donc une zone carrée. Pour effectuer la transformation inverse on utilisera les mêmes matrices que celles utilisées pour la transformation directe mais dans un ordre transposé.

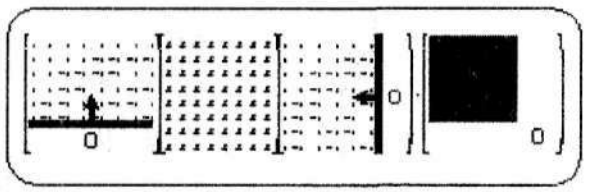

Fig.7 *THR 2-D (Niveau 2)*

Afin de varier le taux de compression, on adjondra une fonction de seuillage des coefficients en aval de la transformation. Ainsi tout coefficient dont la valeur est inférieure au seuil choisi sera remplacé par zéro.

Le gain obtenu avec la THR en temps de calcul pour un traitement logiciel et en surface d'implantation pour un traitement matériel est considérable tout simplement parcequ'on ne calcule que les coeficients qui nous intéressent. Mais il est important de noter que l'utilisation de la THR doit être réstreint à des applications qui n'exigent pas un haute qualité de reconstruction, car l'élimination automatique des coefficients haute fréquence peut induire des erreurs de reconstruction dans la zone à fort détail.

L'algorithme a été validé par logiciel en l'appliquant à une série d'images de la banque du *GDR 134*. Un exemple d'échantillon de test, l'image 'femme' à la quelle est appliquée la THR4 avec différents seuils est présenté en figure 8.

Il est important de noter que cette modification apportée à la transformée de Hadamard est facilement applicable à d'autres transformations telles la TCD (*Travail en cours*).

6.3 Codage des coefficients

Le débit binaire exact est calculé suivant la technique de codage qui doit tenir compte de la concentration des coefficients autour de la composante continue F(0,0) du bloc transformé. Dans le cas de la THR 4, la zone sélectionnée a une taille de 4X4. Notre méthode de codage consiste à utiliser un mot d'état de deux octets qui indiquera la position des coefficients non nuls à transmettre (Fig.9).

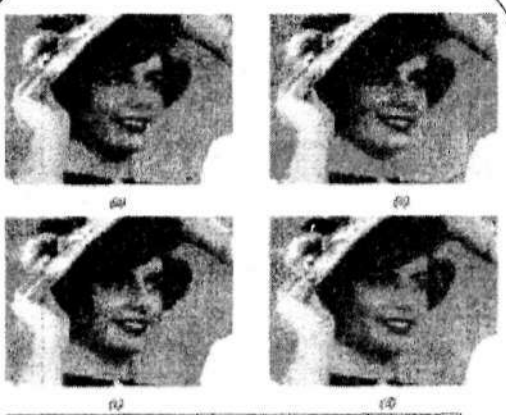

Image /seuil	Débit binaire	PSNR
(b) / 2	0.86	30.2
(c) / 3	0.75	29.3
(d) / 4	0.55	27.8

(a) image originale
(b), (c), (d) : images résultat avec des seuils différents 2, 3, 4

Fig.8 *Résultats de simulation*

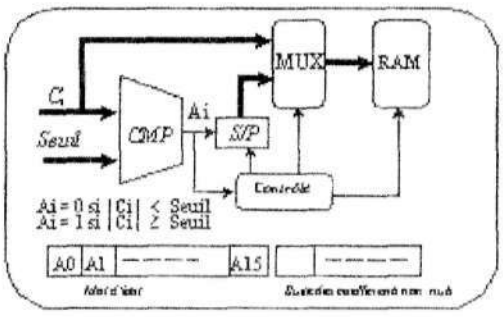

Fig.9 *Codage des coefficients*

L'algorithme de la THR4 opère en temps réel sur des images 512x512 avec une fréquence de travail de 10 MHZ, et un coût d'implantation de 188 CLBs.

7. Implantation de l'algorithme d'extraction de la silhouette d'objets mobiles

La localisation d'objets mobiles est une technique de réduction de la quantité d'information présente dans une séquence d'images. Elle permet d'atteindre des taux de compression élevés du fait qu'on ne transmet que les contours des objets qui ont bougé. Ceci la rend fort utile dans le cas ou on veut effectuer de la surveillance à distance et en utilisant des canaux de transmission à bas débit tel que le réseau téléphonique. L'abondance de la littérature concernant ce problème est significative de son intérêt. Le lecteur trouvera dans [15] [16] une revue des principales méthodes proposées. L'opérateur choisi pour une implantation sur FPGA est celui d'Orkisz; simple à implanter, rapide, et ne présentant pas de contraintes quant au caractère du mouvement et à la connaissance préalable du mouvement.

7.1. Présentation de l'opérateur d'Orkisz

L'opérateur d'Orkisz *(3)* permet de localiser les frontières mobiles dans l'image courante par comparaison des frontières de l'image courante à celles des images précédente et suivante et ceci afin de distinguer les véritables éléments mobiles des éléments statiques désoccultés.

$$\forall x,y \quad M_n(x,y) = max(G_n(x,y), G_{n-1}(x,y), G_{n+1}(x,y)) - max(G_{n-1}(x,y), G_{n+1}(x,y)) \quad (3)$$

avec G_n , G_{n-1} , G_{n+1} *les gradients réspectifs des images* I_n , I_{n-1} , I_{n+1}

7.2. Structure matérielle de l'opérateur

Pour l'implantation de cet opérateur sur notre station on aura besoin de mettre en œuvre un opérateur de détection de contours. Parmi différents opérateurs d'extraction de contours possibles, nous avons choisi l'opérateur de Sobel qui se prête particulièrement bien à une implantation sur FPGA [17].

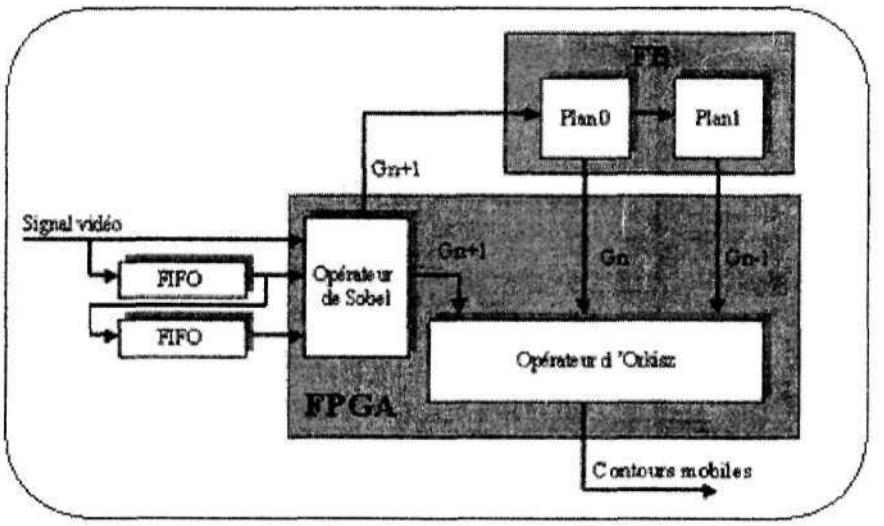

Fig.9. Structure générale de l'extracteur de contours mobiles

Deux plans mémoire mis en cascade permettront de mémoriser les gradients G_n et G_{n-1} déjà calculés auparavant et qui seront utilisés avec le gradient en cours de calcul de l'image (G_{n+1}) dans l'opérateur d'Orkisz (figure.9).

L'opérateur de Sobel est présenté en figure.10a. Ses masques possèdent la propriété de séparabilité et permettent ainsi de remplacer le filtre bidimensionnel par une succession de filtres monodimentionnels (figure.10b). L'opération d'extraction de contours est donc réduite à l'application successive des masque [1,0,-1] et [1,2,1] aux lignes et colonnes de la matrice image (3x3) traitée.

La structure de l'opérateur de Sobel est présentée en figure.11. Les ALU A1 et A2 effectuent respectivement les opérations (x1 + 2.x2 + x3) et (x1-x2). Le maximum du gradient est calculé en prenant le maximum des deux valeurs absolues. Ainsi $G = \max(|G_h|, |G_v|)$.

Le coût d'implantation de la structure complète de détection temps réel de contours mobiles sur FPGA 40XX est de 92 CLBs.

Il est important de noter que les mémoires externes utilisées pour mémoriser les gradients G_n et G_{n-1} font partie du système de développement et plus particulièrement de la carte mémoire. Cette solution a effectivement permis de réduire l'encombrement de la carte

de traitement mais en contre partie l'a rendue dépendante du système de développement. Dans le but d'avoir un système autonome, il serait intéressant de voir s'il est possible d'effectuer une réduction de la taille des mémoires externes qui en général sont encombrantes surtout pour les systèmes embarqués. La solution qui permettrait d'effectuer cette réduction est celle qui consiste à travailler non pas avec des images de contours gradients (non seuillées) mais plutôt avec des images de contours binaires. Si cette solution est déconseillée dans le cas ou l'on utilise des opérateurs d'extraction de contours classiques (approche gradient), elle reste à explorer si d'autres approches sont utilisées.

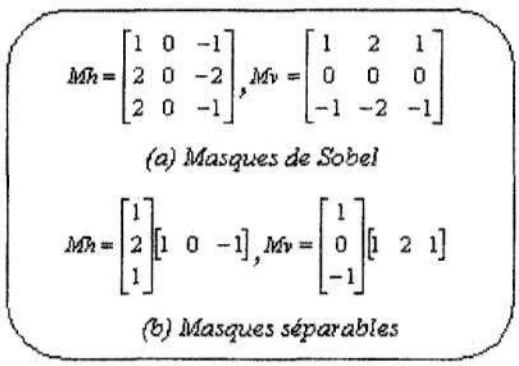

Fig.10 *Masques utilisés*

L'approche proposée utilise les concepts de la logique floue qui trouve son application lorsque les phénomènes étudiés sont vagues, imprécis et difficilement modélisables de manière algébrique. C'est le cas des contours en traitement d'images qui représentent la frontière entre deux régions souvent difficiles à discerner.

L'implantation de l'opérateur d'Orkisz en utilisant l 'approche floue sera détaillé dans une prochaine publication. Les résultats obtenus sont proches à ceux obtenus avec l'opérateur de Sobel mais avec des contours plus fins et moins discontinus. Le coût d'implantation de cet opérateur est de 153 CLBs. Bien que l'opérateur de Sobel soit moins coûteux en nombre de CLBs que l'opérateur flou, ce dernier permet de travailler directement sur des images binaires réduisant ainsi de façon considérable la taille de la mémoire externe nécessaire à l'opérateur d'extraction de contours mobiles.

Beaucoup de travail reste à faire pour réduire l'effet du bruit additif qui

en général est lié au type d'éclairage choisi. En effet une même scène conduit à des résultats différents en lumière naturelle et en lumière artificielle; les ombres des objets induisent de faux contours, notamment pour les objets en mouvement. Un éclairage uniforme de la scène (isotrope) ainsi qu'un lissage approprié des images acquises sont donc nécessaires pour l'obtention de contours moins bruités et plus représentatifs des objets en mouvement.

De nettes améliorations peuvent être apportées en utilisant des opérateurs d'extraction de contours plus robustes au bruit et par conséquent plus complexes. Cette solution n'est pas à exclure même en utilisant la technologie reconfigurable car cette dernière ne cesse de croître en capacité d'intégration tout en affrant des circuits de moins en moins chers [18].

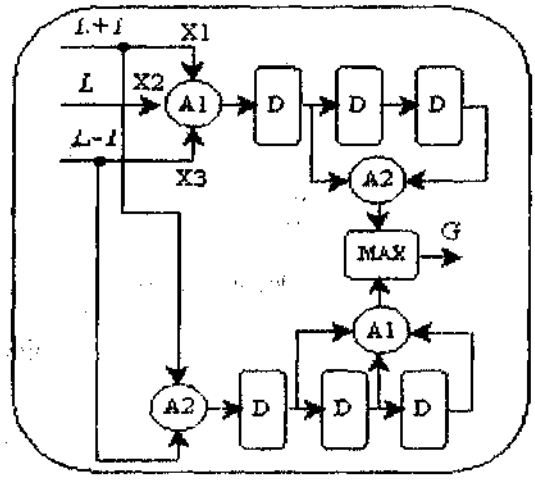

Fig.11 *Implantation de l'opérateur de Sobel*

8. Conclusion

La recherche dans le domaine de l'architecture et des circuits est difficile, car elle nécessite non seulement une parfaite maîtrise des technologies employées mais aussi une très bonne connaissance des caractéristiques des applications et de la nature de leur environnement.

La technologie reconfigurable permet la réalisation de stations d'expérimentation temps réel d'algorithmes de traitement d'images tel que la compression et le filtrage sans avoir recours à des cartes spécialisées qui coûtent excessivement cher.

Une telle station a été décrite ici : en plus des cartes standards d'acquisition et de mémorisation, nous avons réalisé une carte de traitement d'images temps réel (en utilisant des circuits reconfigurables FPGAs de la famille XILINX) qui permet de développer et de tester en temps réel divers algorithmes de traitement d'images), sans avoir à effectuer trop de modifications pour passer d'une application à une autre.

En fin on peut dire que la réussite d'un projet finalisé nécessite souvent d'effectuer des études globales et précises sur les architectures et les algorithmes et d'utiliser toutes les technologies nécessaires pour aboutir à une solution satisfaisante. Pour cela, il est nécessaire de travailler avec des langages de modélisation de haut niveau qui permettent une conception indépendante de la technologie et qui garantissent la portabilité des applications développées. C'est dans ce sens que nous nous orientons en choisissant d'utiliser dans les futurs travaux le langage VHDL[19], qui permet d'effectuer une description du matériel avec un niveau d'abstraction élevé libérant ainsi le concepteur des détails d'implantation et offrant un lien avec les différentes technologies et méthodologies de conception tout en restant indépendant de celles-ci.

Remerciement

Nous tenons à remercier le Professeur Michel Roussel ainsi que son équipe du LTI de l'université de Reims, pour leurs conseils critiques qui nous ont permis de mener à bien ce travail.

Références Bibliographiques

[1] P.M. Athanas, A.L. Abott, *'Real-time Image Processing on a custom Computing Platform', IEEE on Computer, Février 1995.pp 16-24.*

[2] *W.E.Blanz, 'VLSI oriented architecture for real time image processing', Parallel Architecture for Image Processing, 1990, Vol 1246, pp 57-68,.*

[3] *R. Bernard, G. Million, M. Roussel, 'Real Time Inspection of Wood Boards Thickness with minimal FPGA', International Conference on Quality Control by Artificial Vision, Le Creusot, mai 1997, pp 57-68.*

[4] *J.F. Quesne 'Vision robotique :Architecture data flow pour le traitement d'images en temps réel', Thèse de Doctorat* Paris XI Orsay, Janvier1992.

[5] M. Platzner, G. de Micheli, 'Acceleration of satisfiability Algorithms by Reconfigurable Hardware', International workshop on Field Programmable Logic and Applications, Springer-Verglas, Berlin, 1998,
pp 69 – 78.

[6] R.Iwanczuk, ' Using the XC4000 RAM capabilities', The programmable logic data book, XILINX 1994.

[7] W.H. Mangione-Smith, 'Seeking Solutions in Configurable Computing' , IEEE on Computer, Décembre 1997, pp 38-43.

[8] G. Millon, ' Contribution à l'exploitation dynamique de la reconfigurabilité des FPGAs : étude et validation d'une architecture dédiée au traitement d'image' , Thèse de Doctorat de l'Université de Reims-Champagne-Ardenne, juillet 1999.

[9] Imaging Technology Inc, Série 150/151, Video bus and cross port switch specifications, 47-H15012-00, Août 86.

[10] XACT, Viewlogic Interface User Guide, XILINIX, San Jose, CA95124 3400, 1994.

[11] J. Rose et al., 'Architecture of Field programmable Gate Arrays : The Effect of Logic Bloc Functionality on Area Efficiency' , IEEE J. Solide State Circuits, Oct 1990, pp 1217-1225.

[12] Peter A. Rouetz, Po Tong, Douglas. Bailey, Daniel A. Luthi, and Peng H. Ang , ' A High-Performance Full-Motion Video Compression Chip Set', IEEE Transaction on circuits and systems for video technology, vol 2, N°2, Juin 1992,pp 111-122

[13] Yun Q. SHI, H. Sun, 'Image and video compression for multimedia engineering-Fundamentals, Algorithms, and Standards ' , CRC Press LLC, 2000.

[14] R.C. Gonzalez & R.E. Woods, 'Digital image processing' , Addisson Wesley, 1992.

[15] B. Andrew, I.Michael , 'Active contours : the application of techniques from graphics, vision, control theory and statistics to visual tracking of shapes in motion', Springer, cop. 1998.

[16] M.Orkisz, ' Localisation d'objets mobiles dans des scènes naturelles filmées par une caméra fixe' , Revue Traitement du Signal, Vol 9, N° 4, pp 325-346, 1992.

[17] M.Alves, M.Akil, 'Circuits reconfigurables et traitement bas niveau d'images en temps réel', Colloque Gretsi, septembre 1993, pp 1011-1014.

[18] A. Dehon, 'The Density advantage of configurable computing' , IEEE on computer, April 2000, pp 41- 49.

[19] K.Skahill,'VHDL for programmable logic', Cypress Semiconductor-Addison wesley, 1996 .