REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
– MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE –
ECOLE NATIONALE POLYTECHNIQUE.

DEPARTEMENT D'ELECTRONIQUE

Projet de fin d'études

En vue de l'obtention du
Diplôme d'ingénieur d'état en Electronique

Thème:

Implémentation d'algorithmes de traitement d'images sur DSP C6000

Proposé et dirigé par :

Dr. L.HAMAMI (MC)

Réalisé par:

Mr. AHMED ZAID Salim

Promotion : Juin 2005

A mes parents

REMERCIEMENTS

Je tiens à exprimer ma reconnaissance au Dr L.HAMAMI, qui m'a donné l'occasion de travailler sur un sujet passionnant, pour la confiance qu'elle m'a accordée, ses conseils judicieux et son aide précieuse.

Je remercie profondément Dr M.GUERTI, Maître de conférences à l'Ecole Nationale Polytechnique, de m'avoir fait l'honneur de présider le jury ainsi que pour ses encouragements de valeur.

Mes remerciements vont aussi au Pr D.BERKANI pour avoir accepté d'examiner ce modeste travail jugé, tout à mon honneur, par un spécialiste du domaine que j'ai traité.

Ma profonde gratitude va également à toutes les personnes ayant contribué à ma formation.

Je remercie D.DOUACHE et S.BENDI pour leur chaleureux accueil et l'intérêt apporté à mon travail.

Merci à Mohamed Amine ISMAIL « Smisha » pour son soutien moral pendant les moments difficiles. Ainsi que Aziz, Rabie, Mourad, Sofiane, Moumoh, Ali, Ramzi…pour leur amitié.

مختصر

يكمن هذا العمل في تنفيذ الرّوتين المتعلقة بمعالجة الصّور على دي إس بي C6211 من فئة C6000 للمعالجات الرقمية للإشارة. قواعد معالجة الصّور المختارة تتمثل في الفلاتر التقليديّة و التكيّفيّة، بالإضافة إلى قواعد التقسيم باعتبار الحواف والمناطق. لتنفيذ ذلك، دي إس كيي TMS320C6211، البرنامج Code Composer Studio و MATLAB يكوّنون أدوات التّطوير التي تتيح تمثيل الأجزاء "الإكتساب، المعالجة والتمثيل". الرابطة RTDX مستعملة لإنشاء اتّصال بين CCS وDSK. Link for Code Composer Studio للبرنامج MATLAB مستعمل من طرف واجهة المستخدم لإنشاء اتّصال بين CCS وMATLAB.

الكلمات المفتاحية: خوارزميات معالجة الصّور، دي إس بي، Code Composer Studio، RTDX، Link for Code Composer Studio

Abstract

This work consists of an implementation of routines of image processing on the DSP C6211 of C6000 family. The image processing algorithms selected are those of conventional and adaptive filters, and the algorithms of segmentation by edge and area approaches. To be done, DSK TMS320C6211, the software Code Composer Studio and MATLAB constitute the development tools allowing the representation of the parts "acquisition, processing and displaying". RTDX is a logic link used to establish a communication between the CCS and DSK. Link for Code Composer Studio of MATLAB is used by the graphic user interface (GUI) to establish a link between MATLAB and CCS.

Key words: image processing algorithms, DSP, Code Composer Studio, RTDX, Link for Code Composer Studio.

Résumé

Ce travail consiste en une implémentation de routines de traitement d'images sur le DSP C6211 de la famille C6000. Les algorithmes de traitements d'images choisis sont les algorithmes de filtrage classiques et adaptatifs, et les algorithmes de segmentation par approche contour et région. Pour ce faire, la carte DSK TMS320C6211, les logiciels Code Composer Studio et MATLAB constituent les outils de développement permettant la représentation des parties « acquisition, traitement et affichage ». La liaison RTDX est utilisée pour établir une communication entre le CCS et la carte DSK. Le Link for Code Composer Studio de MATLAB est utilisé par l'interface graphique (GUI) pour établir la liaison logique entre MATLAB et CCS.

Mots clés : algorithmes de traitement d'images, DSP, Code Composer Studio, RTDX, Link for Code Composer Studio.

TABLE DES MATIERE

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

L'essor du traitement du signal, en général, dans le domaine applicatif est fortement lié à l'avènement sur le marché des processeurs de traitement du signal (DSP, acronyme de Digital Signal Processor) il y a de cela une quinzaine d'années. Depuis, ces processeurs sont utilisés dans une multitude de domaines dont le traitement d'images. Le traitement d'images est une science basée sur plusieurs disciplines : les mathématiques, les probabilités, l'informatique et les processeurs (ou architectures avancées).

Ce projet de fin d'études entre dans le cadre des travaux de recherche du Dr.L.HAMAMI, il permet dans ses limites de voir la faisabilité des algorithmes de traitement d'images sur le processeur C6211 de la famille C62x de Texas Instruments, pour réaliser dans l'avenir un système de traitement d'images autonome ou embarqué. La partie théorique sera conclue par une application et création d'une librairie de fonctions pouvant être utilisée sous CCS.

Ce rapport comporte les chapitres suivants :

Le chapitre I donne quelques notions sur le traitement d'images en général, et les algorithmes de prétraitement nécessaires pour améliorer la qualité de l'image.

Le chapitre II est consacré aux algorithmes de détection de contours adoptant les approches dérivatives de premier ordre et du deuxième ordre ; ainsi qu'un algorithme d'extraction des régions homogènes par approche multifractale.

Le chapitre III, est une étude de la famille C6000 en particulier le processeur C6211 qui est le cœur de la carte DSK (support de développement). Nous présentons une description fonctionnelle et logicielle de ce composant.

Le chapitre IV présente les outils de développement logiciel et matériel employés dans la réalisation de l'application. Le Code Composer Studio de Texas Instruments avec la technologie RTDX, le Link for Code Composer Studio sous MATLAB de MathWorks, et la carte DSP Starter Kit de Texas Instruments.

Dans le chapitre V, les algorithmes présentés aux chapitres précédents feront l'objet de fonctions sous CCS, ainsi qu'une création d'une interface graphique pouvant communiquer avec la cible moyennant l'RTDX et le Link for Code Composer. Nous présentons également les résultats des traitements.

Chapitre 1

Concepts de base en traitement d'images et prétraitement

CHAPITRE I

TRAITEMENT D'IMAGES

1.1 INTRODUCTION

Théoriquement, le traitement d'images est un ensemble d'approches, de méthodes, de techniques et d'outils offerts par les mathématiques et le traitement du signal. Cette panoplie a la volonté de résoudre la majorité des problèmes qui peuvent se présenter lorsqu'il est nécessaire d'extraire et de comprendre de façon automatique les informations présentes dans une image. Ces informations peuvent être significatives pour le système visuel ou pour un domaine d'application particulier.

En pratique, le traitement d'images (ou son analyse) est une suite de phases qui doivent être exécutées, depuis la formation de l'image jusqu'à la prise de décision fondée sur son contenu. Certaines de ces phases successives sont souvent étroitement liées et souvent indissociables. Les unes sont obligatoires alors que l'exécution de certaines autres s'avère parfois facultative.

L'analyse d'image automatisée voit son essor grâce au progrès de l'informatique et à l'apparition des images en niveaux de gris héritées de la recherche spatiale. A l'heure actuelle, l'analyse d'image est une discipline dont les fondements théoriques reposent sur les mathématiques et les fondements pratiques sur l'informatique et l'électronique.

Dans ce chapitre, nous parlons brièvement des concepts de base liés à l'analyse numérique d'image de façon générale et son filtrage. Nous nous intéresserons aux images en niveau de gris.

1.2 DEFINITION D'UNE IMAGE

L'image est définie, physiquement, comme étant la projection d'objets tridimensionnels sur un plan. Elle fait parti des signaux informationnels telle que la parole, et se présente soit sous son aspect continu ou discret (formée de points élémentaires). Son traitement nécessite la connaissance de certains paramètres, caractéristiques, et entités.

1.2.1 Image numérique

Il s'agit d'une image dont la surface est divisée en pixels « picture elements », figure 1.1, considérés comme la plus petite entité calculable et qui peuvent être manipulés par les matériels et logiciels d'affichage et d'impression.

FIGURE 1.1 Image en pixels

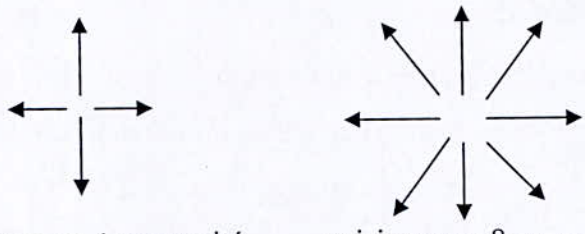
Ce format (une matrice *f (i , j)*) est issu de la numérisation d'une image continue représentée par un signal *f(x,y)* (i.e distribution continue d'intensité dans un plan). Dans le cas d'une image en niveau de gris, chaque pixel est codé sur un byte c'est à dire une valeur possible entre [0 255], et ayant ainsi une couleur allant du noir (i.e valeur 0) au blanc (i.e valeur 255). Pour une image en couleur, le pixel est codé sur trois bytes qui correspondent au rouge, vert et bleu offrant ainsi un spectre riche de couleurs (2^{24} couleurs). [2]

1.2.2 Voisinage d'un pixel

Le voisinage d'un pixel est composé de tous les pixels qui l'entourent immédiatement. Si p est un pixel d'une image D, alors le voisinage de p est le plus petit sous-ensemble de D qui contient p.

Dans une image numérique, on distingue deux types de connexités relatives au voisinage utilisé : la 4-connexité et la 8-connexité (figure 1.2).

Le voisinage d'un pixel (i,j) est dit 4-connexe s'il est formé des quatre pixels de coordonnées spatiales (i+1,j),(i,j+1),(i-1,j),(i,j-1), et il est dit 8-connexe s'il est formé des pixels de coordonnées spatiales (i+1,j),(i,j+1),(i-1,j),(i,j-1),(i+1,j-1),(i+1,j+1),(i-1,j+1),(i-1,j-1). [1] [2]



Voisinage en 4-connexité voisinage en 8-connexité

FIGURE 1.2 Image en pixels

1.2.3 Résolution d'une image

La résolution traduit la clarté et la finesse des détails. Elle est exprimée en nombre de pixels par unité de longueur. Il s'agit d'un paramètre qui est défini lors du processus de numérisation ou de reproduction de l'image.

1.2.4 Contraste

C'est l'opposition marquée entre deux régions juxtaposées d'une image, plus précisément les régions sombres et les régions claires de cette image [2]. Il est défini en fonction des luminances L_i et L_{i+1} de deux zones de l'image par l'équation (1.1) :

$$c = \frac{L_i - L_{i+1}}{L_i + L_{i+1}} \tag{1.1}$$

1.2.5 Histogramme

L'histogramme est une fonction qui donne la fréquence d'apparition de tous les niveaux de gris dans une image donnée et qui ne tient pas compte de leurs distributions spatiales. Le calcul d'histogramme peut faire l'objet d'une diminution de l'erreur de quantification, une comparaison de deux images obtenues sous éclairages différents et une amélioration de certaines proportions afin d'extraire les informations utiles (figure 1.3). [1]

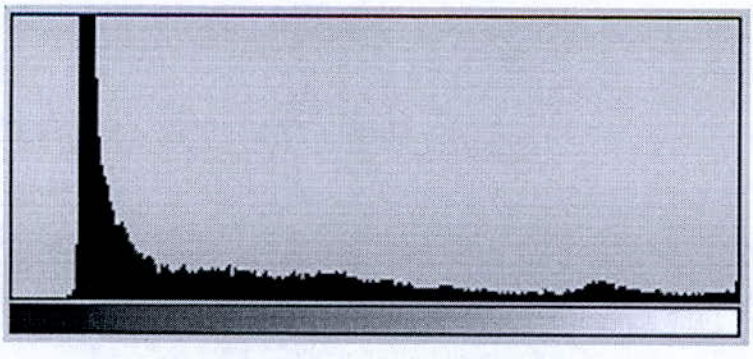


FIGURE 1.3 Histogramme des niveaux de gris

1.2.6 Bruit d'image

Le bruit se présente sous forme de fluctuations autour des valeurs moyennes d'intensité des régions formant l'image. Il peut provenir soit du dispositif d'acquisition (scanner, caméra, amplificateur,...) soit de la scène elle-même, ce qui entraîne une dégradation de la qualité de l'image. On distingue trois types de bruit : le bruit additif qui est le plus fréquent, le bruit multiplicatif (exemple, bruit de speckle dans l'image radar ou grains sur les films radiographiques) et le bruit convolutif. [1]

1.3 PROCESSUS D'ANALYSE D'IMAGES

Le processus d'analyse d'images, qui a pour but de fournir une description ou une interprétation d'une scène à partir de l'information extraite de l'image, peut être décomposé en plusieurs étapes, comme le montre la figure 1.4 :

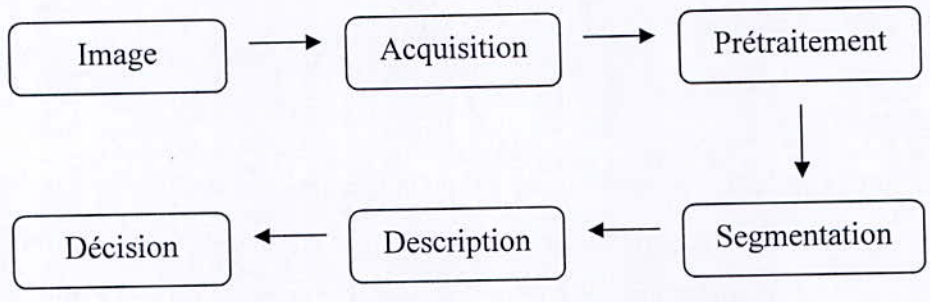


FIGURE 1.4 Etapes du processus d'analyse d'images

Au début, on doit faire l'acquisition d'une image, en discrétisant l'image réelle continue. Normalement la quantité d'information brute initiale, après la discrétisation, est très volumineuse et difficile à manipuler. De plus, cette discrétisation entraîne une perte d'information, et peut poser des problèmes d'ordre technique, comme l'illumination et la texture, entre autres.

Étant bruitées ou floues, ces images doivent passer par un prétraitement avant toute opération de détection, où l'information dégradée est restaurée. Dans cette étape, des détails de l'image peuvent être rehaussés par plusieurs techniques.

Ensuite vient la phase la plus importante et la plus difficile du processus d'analyse d'images : la segmentation. Segmenter une image correspond à trouver les régions qui ont un sens. Cette étape doit permettre d'interpréter l'image aussi bien que le ferait l'observateur.

A partir de là, viennent les traitements de haut niveau, tels que la description de l'image, la reconnaissance des formes et les décisions qui pourront être prises à partir des résultats fournis par la segmentation.[3]

1.4 PRETRAITEMENT

Le prétraitement a pour but de faciliter la segmentation en renforçant la ressemblance entre pixels appartenant à une même région, ou en accentuant la dissemblance entre pixels appartenant à des régions différentes. Il s'agit d'effectuer des traitements concernant la modification d'histogramme et la réduction de bruit.[1]

1.4.1 Modification d'histogramme

On cherche à améliorer l'image en lui appliquant des transformations ponctuelles d'intensité (figure 1.5). C'est-à-dire, qu'à tout pixel d'intensité a_s on associe une intensité $a_s'=T(a_s)$.

Du fait de leur caractère ponctuel, les méthodes de transformation d'histogramme n'affectent pas la forme des régions. Elles en modifient uniquement l'apparence visuelle. [4]

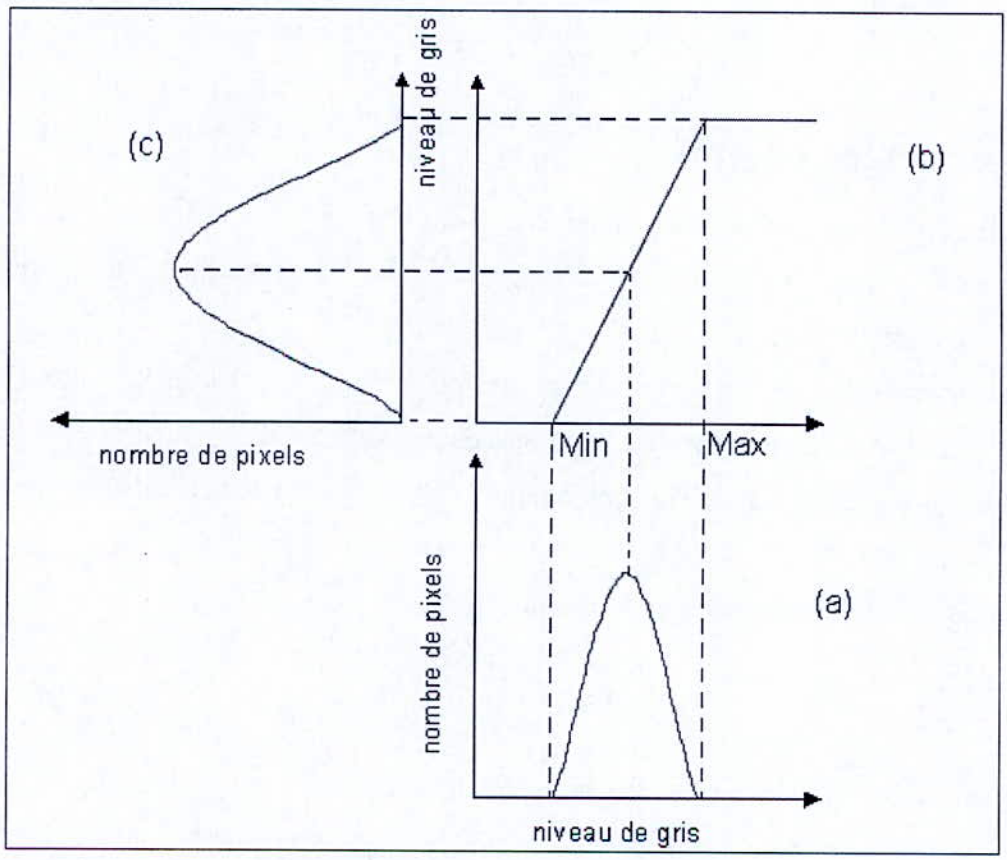


FIGURE 1.5 Modification d'histogramme

1.4.2 Binarisation par seuillage

Le but de cet algorithme est la binarisation d'images à niveaux de gris. Ceci revient à séparer les pixels de l'image en deux classes, la première ayant un niveau maximal (typiquement255) et la seconde un niveau minimal (0). La méthode consiste à précalculer un seuil d'après l'histogramme de l'image qui approche le mieux l'image binarisée de l'originale (figure 1.5). [4]

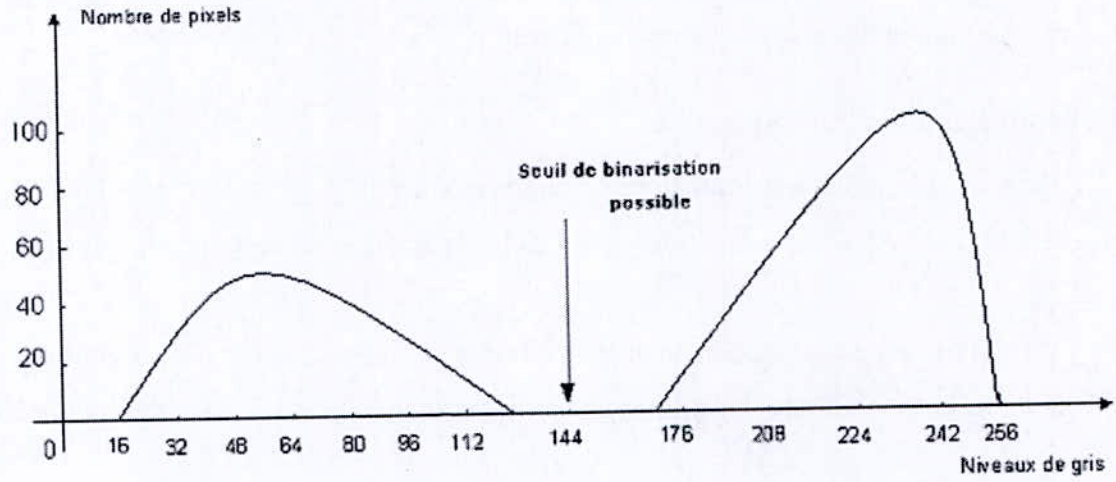


FIGURE 1.5 Choix du seuil sur l'histogramme

1.5 REDUCTION DU BRUIT

Les procédés d'acquisition (caméra, amplificateur, quantificateur, ...) d'images induisent des perturbations qui peuvent être gênantes pour la compréhension et le traitement de la scène, comme le bruit d'image.

Le bruit d'image est en toute rigueur considéré comme un champ aléatoire. Du fait de sa haute fréquence, on ne conserve pour le caractériser que le premier ordre (densité de probabilité f() sur un pixel de l'image(équation 1.2) et parfois le second (corrélation entre pixels).

On ne conservera ici que le premier ordre, modélisé par :

$$f(a)=Cexp(-K|a|^n) \tag{1.2}$$

Cette modélisation permet de retrouver la nature plus ou moins impulsionnelle du bruit, c'est-à-dire sa tendance a s'écarter de l'espérance mathématique de la région considérée. Pour n=1, on trouve le bruit exponentiel ou impulsionnel.

Pour n=2, on a le bruit gaussien

L'objectif avoué du filtrage est de réduire les variations d'intensité au sein de chaque région de l'image tout en respectant l'intégrité des scènes : les transitions entre régions homogènes, les éléments significatifs de l'image doivent être préservés au mieux. Différentes méthodes de filtrage ont été développées suivant le type et l'intensité du bruit, ou les applications auxquelles on destine l'image. Les premières et les plus simples de ces méthodes sont basées sur le filtrage linéaire stationnaire (invariant par translations), mais les limitations de ces techniques (en particulier leur mauvaise conservation des transitions) a conduit au développement des filtres "non-linéaires".

Dans la plupart des cas présentés ici, le filtrage consiste à balayer l'image par une fenêtre d'analyse de taille finie (masque ou kernel) : le calcul du nouveau niveau de gris du pixel considéré ne prend en compte que les plus proches voisins de celui-ci. [1] [6]

1.6 FILTRES LINEAIRES

Ces opérateurs sont caractérisés par leur réponse impulsionnelle *h(x,y)*. La relation entrée-sortie est décrite par l'équation de convolution :

$$C(x,y) = A * h(x,y) = \iint A(\alpha,\beta)h(x-\alpha, y-\beta)d\alpha d\beta \qquad (1.3)$$

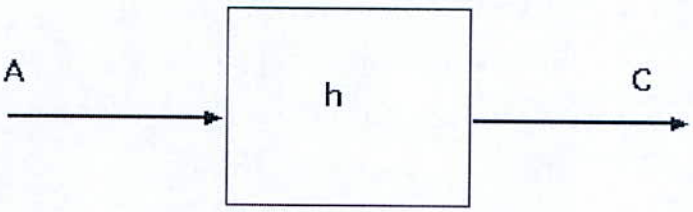


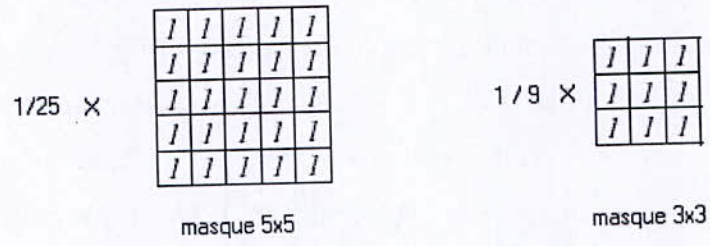
FIGURE 2.6 Filtrage linéaire

Dans le cas d'une image numérique, l'équation (1.3) est remplacée par la relation de convolution discrète (1.4) : [1] [4]

$$D[i,j] = \sum_m \sum_n h[m,n]A[i-m, j-n] = \sum_m \sum_n A[m,n]h[i-m, j-n] \qquad (1.4)$$

1.6.1 Filtre moyenneur

Le niveau de gris du pixel central est remplacé par la moyenne des niveaux de gris des pixels environnants. La taille du masque dépend de l'intensité du bruit et de la taille des détails significatifs de l'image traitée.



Les effets du filtre moyenneur varient avec la taille du noyau : plus les dimensions du noyau seront importantes, plus le bruit sera éliminé ; mais en contrepartie, les détails fins seront eux aussi effacés et les contours étalés. [1] [5]

1.6.2 Filtre smooth

Les coefficients du masque pour un filtre 3×3 sont :

1	2	1
2	4	2
1	2	1

Les coefficients sont calculés en utilisant des pondérations gaussiennes. Des itérations successives permettent d'obtenir le smooth 5×5 (2 itérations) et le smoot 7×7 (3 itérations).

Dans ce cas aussi, l'effet du filtre augmente avec la taille de son masque. Les contours et les détails fins sont cependant mieux conservés qu'avec le moyenneur.

En effet, en utilisant une pondération gaussienne, le filtre smooth prend mieux en compte les corrélations entre pixels, notamment pour une texture (la fonction de corrélation des niveaux de gris pour une texture est fréquemment modélisée par une gaussienne).

Le filtre smooth est un bon exemple des performances qu'on peut obtenir avec un filtre linéaire à réponse impulsionnelle finie. Le gros avantage de ces filtres est leur facilité de conception et d'implémentation, mais ils ne peuvent être utilisé pour des travaux trop fins (la détérioration des contours qu'ils induisent par exemple, empêchera une segmentation fine des images). Ces limitations ont donc conduit à la conception de filtres non-linéaires. [1] [6]

1.7 FILTRES NON-LINEAIRES

Ces opérateurs ont été développés pour pallier aux insuffisances des filtres linéaires : principalement la mauvaise conservation des contours. Ils ont le défaut d'infliger des déformations irréversibles à l'image. [1]

1.7.1 Filtre médian

Le niveau de gris du pixel central est remplacé par la valeur médiane de tous les pixels de la fenêtre d'analyse centrée sur le pixel. La taille du masque dépend de la variance du bruit et de la taille des détails significatifs de l'image traitée.

La figure 1.7 illustre le fonctionnement d'un filtre médian de fenêtre d'analyse 3×3 sur un exemple :

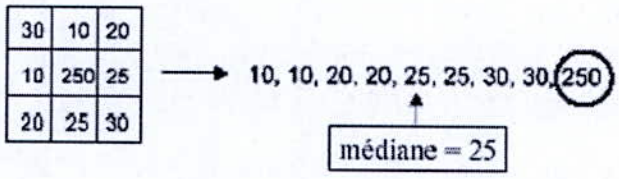


FIGURE 1.7 Filtrage médian

Le médian garde la netteté de l'image pour les éléments de dimensions importantes par rapport au noyau du filtre, mais élimine les détails fins de manière irrémédiable. [5] [6]

1.7.2 Filtre de Nagao

Le principe de ce filtrage est de remplacer chaque pixel de l'image par la valeur moyenne des pixels contenus dans une fenêtre particulière. Il s'agit de choisir la fenêtre la mieux adaptée parmi un certain nombre de fenêtres prédéfinies. Neuf fenêtres sont ici définies, chacune contenant neuf pixels, dont le pixel à remplacer. Elles s'inscrivent dans une fenêtre de taille 5x5 centrée sur le pixel à modifier.

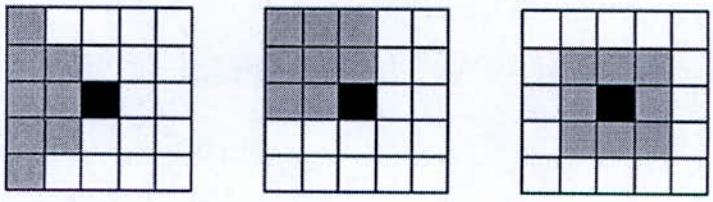


FIGURE 1.8 Masques de Nagao

Sur la figure 1.8 sont représentées trois fenêtres, contenant les pixels gris et le pixel noir (pixel central à remplacer). Les fenêtres de filtrages en sont déduites de la façon suivante : quatre fenêtres sont issues de celle de gauche (par rotations d'angles 0, 90, 180 et 270 degrés) ; quatre sont issues de celle du milieu de la même façon ; une est celle de droite.

La fenêtre qui sera sélectionnée est celle dont la variance est la plus faible. La valeur moyenne de cette fenêtre remplacera celle du pixel central.

Ce filtrage permet de lisser l'image. Toutefois, le fait de choisir la fenêtre de plus petite variance implique que cette moyenne est faite sur une zone relativement homogène. Ainsi, les contours sont conservés : un pixel proche d'un contour sera remplacé par une valeur homogène à la zone à laquelle il appartient, et non pas par une valeur moyenne entre la zone à laquelle il appartient et celle de l'autre côté du contour. [1] [6]

1.8 FILTRES MORPHOLOGIQUES

Le filtre morphologique est utilisé pour éliminer des pixels isolés dans une image binaire, qui sont considérés comme un bruit. Il met en correspondance chaque pixel avec son voisinage par une fonction logique (OR, XOR, AND).Parmi, les opérateurs morphologiques, nous citons : [1]

1.8.1 Dilatation

La dilatation consiste en le choix du plus grand des éléments du masque. Elle élimine les tâches blanches dans des zones noires mais ajoute des pixels noirs au contour des objets présents dans l'image. La dilatation élimine les tâches blanches dans des zones noires mais ajoute des pixels noirs au contour des objets présents dans l'image. [5] [7]

1.8.2 Erosion

L'érosion consiste en le choix du plus petit des éléments du masque. Elle permet d'éliminer les pixels noirs isolés au milieu des parties blanches de l'image. Autres effets habituels de l'érosion sont : la séparation des objets à l'endroit des étranglements, le rétrécissement des objets de grande taille et la disparition des petites composantes. [5] [7]

1.8.3 Ouverture

L'ouverture est constituée par une opération d'érosion suivie d'une dilatation. En général, l'image traitée diffère de l'image de départ : l'ensemble ouvert est plus régulier et

moins riche en détails que l'ensemble initial. La transformation par ouverture adoucit donc les contours. [5] [7]

1.8.4 Fermeture

La fermeture est l'opération inverse de l'ouverture, qui consiste à faire subir à l'image une dilatation suivie d'une érosion. Elle permet aussi de retrouver la taille normale des objets de l'image. [5] [7]

1.9 FILTRES ADAPTATIFS

Lorsque l'on veut filtrer une image, on peut rencontrer deux types de situations : soit la fenêtre d'analyse est située à l'intérieur d'une région, et on a affaire à un signal stationnaire éventuellement perturbé par du bruit à caractère impulsionnel ou continu ; soit elle inclut une transition entre deux régions (contour). L'intérêt des filtres adaptatifs est la possibilité d'ajuster la structure ou les coefficients d'un opérateur à chacune de ces situations. Il s'agit donc de filtres comportant un étage de décision et un étage de filtrage de données. [1]

1.9.1 Moyenne adaptative

Le filtre est décrit par l'équation 1.5 suivante :

$$C[i,j] = \frac{\sum_{k=1}^{L} w(a_k - A[i,j]) \cdot a_k}{\sum_{k=1}^{L} w(a_k - A[i,j])} \quad ; \tag{1.5}$$

avec L : taille du masque; $w(x) = \begin{cases} 1 & \text{si } |x| \le t \ (t \text{ seuil préfixé}) \\ 0 & \text{sinon} \end{cases}$

Lorsque l'on est en présence d'un contour, ce filtre évite l'effet d'élargissement de la zone de transition qui peut entraîner une perte d'information sur la géométrie de la région. Pour une des régions homogènes ce filtre se comporte comme un filtre moyenneur.

Il est nécessaire, pour un bon fonctionnement, de déterminer le seuil t à partir de l'écart type du bruit. Si cette donnée n'est pas disponible, on peut choisir le seuil t comme étant la médiane des écarts par rapport à la médiane (voir l'équation 1.6) :

$$t = median\left\{\left|a_1 - a_{\left(\frac{L+1}{2}\right)}\right|,\ldots,\left|a_k - a_{\left(\frac{L+1}{2}\right)}\right|,\ldots,\left|a_L - a_{\left(\frac{L+1}{2}\right)}\right|\right\} \quad (1.6)$$

Le filtre ainsi réalisé, appelé M-MAD-TM, présente un bon compromis entre performance et complexité. [1]

1.9.2 Filtre de rang adaptatif

Les filtres d'ordre ou filtres de rang sont généralement vus comme des combinaisons des statistiques d'ordre d'un ensemble d'observations. [1] [8]

Soient, selon la figure 1.8 :

- a_k (resp. b_k) l'échantillon à la position k du signal d'entrée (resp. sortie)
- $A_k = \{a_1 .. a_L\}$ l'ensemble des échantillons disponibles dans une fenêtre d'observation centrée à la position A[i,j]

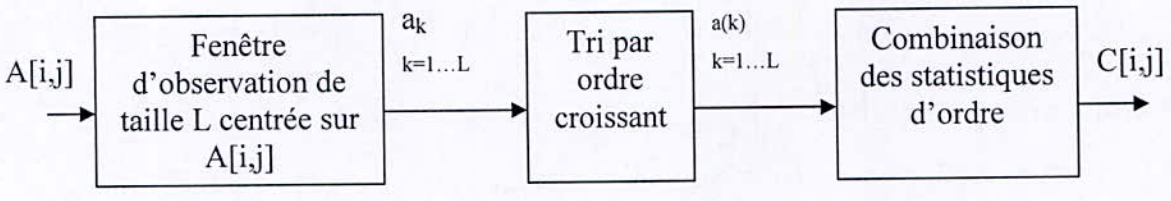


FIGURE 1.8 Filtre d'ordre

Comme toujours, lors du filtrage d'une image, soit la fenêtre d'analyse est située à l'intérieur d'une région : on a alors affaire à un signal stationnaire éventuellement bruité, soit la fenêtre comporte une transition entre deux régions. L'intérêt des filtres de rang est de bien se comporter dans les deux cas, contrairement aux filtres linéaires qui détériorent les transitions par moyennage.

Cependant, l'image peut changer de nature selon les régions, il est donc préférable d'utiliser des filtres de rang " adaptatifs ", c'est à dire des filtres dont la fonction f dépend des valeurs des pixels de la fenêtre d'analyse. En d'autres termes, la combinaison des statistiques d'ordre est variable d'une fenêtre d'observation à une autre. Un filtre de rang adaptatif aura alors les mêmes bonnes propriétés pour les contours tout en ayant de meilleures performances dans les régions homogènes bruitées.

On définit un indice ρ compris entre 0 et 1, et k désigne le rang ; le filtre de rang adaptatif est donné par la formule (1.7) :

le rang : r=Ent(1+ρ(L-1)) (1.7)

$$\text{avec } \rho=\frac{1}{2}+\frac{\sum_{k=1}^{L} a_{(k)}-a_{(l)}-\left(a_{(L)}-a_{(l)}\right)\frac{k-1}{L-1}}{(L-1)\left(a_{(L)}-a_{(l)}\right)}$$

L'étage de décision utilise l'information sur la symétrie (ou la dissymétrie) de la répartition des statistiques d'ordre $a_{(k)}$. Dans le cas d'une répartition symétrique (région homogène non bruitée) la valeur de ρ est proche de ½ et le filtre adaptatif a le comportement d'un filtre médian. En cas de présence d'une impulsion parasite positive, cette répartition est décalée du côté des statistiques de rang faible, le filtre sélectionne une statistique de rang faible, ce qui élimine le bruit. Et inversement en présence d'un bruit impulsionnel négatif.

1.10 CONCLUSION

Dans ce chapitre nous avons présenté quelques définitions liées au traitement d'images, ainsi qu'une phase de son processus à savoir le prétraitement fondé sur les algorithmes de filtrage classiques et adaptatifs. Ces derniers, basés sur des attributs stochastiques et des transformations ponctuelles, présentent davantage d'améliorations dans la réduction du bruit introduit par l'acquisition.

Chapitre 2

Segmentation d'images

CHAPITRE II

SEGMENTATION D'IMAGES

2.1 INTRODUCTION

La segmentation est l'étape qui suit le prétraitement de l'image, elle consiste à cerner les formes des objets, qui constituent cette image ; le partitionnement en objets disjoints et leur union couvre toute l'image. Son but est d'extraire les entités d'une image, afin d'y appliquer un traitement spécifique pour une éventuelle interprétation de l'image. Il existe deux approches en segmentation : l'approche frontière (détection de contours) et l'approche région.

2.2 DETECTION DE CONTOURS

Les contours sont des zones de transition séparant les éléments de l'image et leur arrière plan. Ces transitions sont considérées comme des variations brusques d'intensité, délimitant ainsi les objets. Plusieurs méthodes ont été adoptées dont les méthodes dérivatives du premier ordre et du deuxième ordre et celle basée sur la morphologie mathématique. [1]

2.2.1 Opérateurs dérivatifs du premier ordre

2.2.1.1 Opérateurs gradient

Cet opérateur permet de caractériser et de repérer les zones de transition. La détection de contours revient à déterminer les extrema locaux dans la direction du gradient. [1]

Le gradient d'une image est obtenu en appliquant le vecteur suivant, équation (2.1) :

$$\nabla I(x,y) = \left(\frac{\partial I(x,y)}{\partial x}, \frac{\partial I(x,y)}{\partial y} \right)^t \tag{2.1}$$

Ce vecteur peut être représenté par son module m (relation 2.2) et sa phase ϕ (relation 2.3) qui donne la direction :

$$m = \sqrt{\left(\frac{\partial I(x,y)}{\partial x} \right)^2 + \left(\frac{\partial I(x,y)}{\partial y} \right)^2} \tag{2.2}$$

$$\phi = \arctan\left(\frac{\partial I(x,y)}{\partial y} / \frac{\partial I(x,y)}{\partial x}\right) \tag{2.3}$$

D'après la relation (2.1) il s'agit, dans le cas discret, de calculer les différences des luminances suivant x et y. Pour des commodités de calculs, on prend :

$$m = \left|\frac{\partial I(x,y)}{\partial x}\right| + \left|\frac{\partial I(x,y)}{\partial y}\right| \tag{2.4}$$

On peut donc définir les relations suivantes (2.5) appliquées dans le cas discret :

$$\begin{cases} G_x = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ et } G_y = \begin{bmatrix} 1 & -1 \end{bmatrix} \\ I_x(i,j) = \begin{bmatrix} I(i,j) & I(i,j+1) \end{bmatrix} G_x \\ I_y(i,j) = G_y \begin{bmatrix} I(i,j) & I(i+1,j) \end{bmatrix}^t \\ I'(i,j) = \left|I_x(i,j)\right| + \left|I_y(i,j)\right| \end{cases} \tag{2.5}$$

2.2.1.2 Opérateurs de Roberts

C'est un détecteur de contours qui calcule les dérivées suivant les transitions diagonales, les masques de Roberts sont définis par les matrices suivantes :

$$h1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad h2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Le calcul s'effectue de la même façon que celui de l'opérateur gradient, mais qui doit être précédé par un filtrage. [5]

2.2.1.3 Opérateurs de Prewitt

L'image est convoluée avec le masque de Prewitt ci-après :

$$h1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad h2 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Chaque masque intègre un filtre de lissage et un détecteur de contour du premier ordre [1]. Le calcul se fait ainsi : $C[i,j] = |A[i,j] * h_1| + |A[i,j] * h_2|$

2.2.1.3 Opérateurs de Sobel

Les masques de Sobel sont :

$$h1=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} \qquad h2=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}$$

Ces opérateurs effectuent une moyenne locale sur trois pixels, et détermine la dérivée du premier ordre. C'est une opération moins sensible au bruit. Les opérateurs de Sobel sont fréquemment utilisés dans la détection de contours.[1]

2.2.1.4 Opérateurs de gradients directionnels de Kirsh

Il s'agit d'un opérateur à huit masques obtenus par rotation de $\pi/4$ de l'opérateur de base h_0.

$$h_0=\begin{bmatrix}5 & 5 & 5\\ -3 & 0 & -3\\ -3 & -3 & -3\end{bmatrix} \quad h_1=\begin{bmatrix}5 & 5 & -3\\ 5 & 0 & -3\\ -3 & -3 & -3\end{bmatrix}\ldots$$

Comme réultat le gradient sera celui correspondant à la valeur maximale donnée par : $Max\ i=0..7|\ hi * A\ |$, ou hi est le $i^{ème}$ masque, et A est une fenêtre donnée de l'image.[1]

2.2.2 Opérateurs dérivatifs du second ordre

Cette méthode consiste à calculer le Laplacien de l'image, et à repérer les points des contours par son passage par zéro [1]. Le Laplacien d'un signal continu est donné par la relation (2.6) :

$$\Delta I(x,y)=\frac{\partial^2 I(x,y)}{\partial^2 x}+\frac{\partial^2 I(x,y)}{\partial^2 y} \tag{2.6}$$

Dans le cas discret des approximations du Laplacien, calculé sur un voisinage 3×3, correspondent aux masques suivants :

$$H_1=\begin{pmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{pmatrix} \qquad H_2=\begin{pmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{pmatrix} \qquad H_3=\begin{pmatrix}1 & -2 & 1\\ -2 & 4 & -2\\ 1 & -2 & 1\end{pmatrix}$$

2.2.3 Gradient morphologique

On peut obtenir un contour en calculant la différence entre la dilatation d'une image donnée et son érosion [1], relation (2.7) :

$$C(i,j) = Dilatation(A(i,j)) - Erosion(A(i,j)) \tag{2.7}$$

Le calcul de la norme du gradient est très sensible au bruit. Cette approche ne peut s'appliquer que sur une image lissée.

2.3 SEGMENTATION PAR APPROCHE MULTIFRACTALE

L'analyse multifractale des images consiste à définir des mesures à partir des niveaux de gris, à en calculer les spectres, et à traiter les points sur la base des informations à la fois locales et globales qui en résultent. Contrairement à d'autres approches, aucun filtrage n'est effectué. On peut ainsi effectuer des segmentations, du débruitage, ou de la détection de changements (contours) [9]. Avant d'introduire le formalisme, nous allons donner certaines notions et définir les attributs fractals liés à notre analyse.

2.3.1 Fractal

C'est une figure géométrique de structure complexe dont la création ou la forme met en jeu des règles utilisant le fractionnement. Cette construction est liée à l'invariance d'échelle où on répète la même transformation à des échelles de plus en plus petites, un exemple d'objet fractal : le tapis de ***Sierpinski*** (figure 2.1). Une des caractéristiques principales de tout objet fractal est sa dimension fractale, qui mesure son degré d'irrégularité et de brisure.[10]

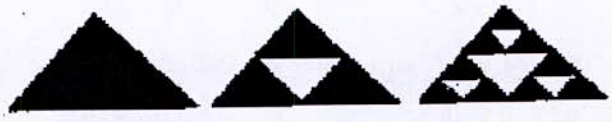


FIGURE 2.1 Tapis de Sierpinsky

2.3.2 Formalisme multifractal pour l'analyse d'image

La géométrie fractale a été introduite pour décrire les relations d'échelle entre des structures géométriques et l'échelle d'analyse de ces structures : la "taille" d'un ensemble fractal varie comme l'échelle à laquelle il est examiné, donné par la dimension fractale. La généralisation à la notion de multifractale revient essentiellement à considérer les ensembles multifractals comme une hiérarchie d'ensembles dont chacun a sa propre dimension fractale. Ainsi l'analyse multifractale fournit une relation d'échelle qui requiert une famille (éventuellement infinie) de dimensions, plutôt qu'une seule dimension comme dans le cas de la géométrie fractale.

L'idée de départ est d'analyser la régularité locale du signal, sur laquelle on n'a fait aucune hypothèse. La notion de singularité est ainsi introduite. Il apparaît que l'information de régularité ponctuelle est très utile sous cette forme, mais qu'il faut lui adjoindre des informations structurelles. Précisément, l'analyse locale est complétée par une description globale, de plus haut niveau. Cette description consiste le plus souvent à mesurer la taille des ensembles de points de même régularité : c'est la notion de dimension qui est définie.[13]

2.3.3 Exposant de singularité

Soit μ une mesure quelconque et $supp\mu$ son support. Une mesure permet d'associer des poids relatifs aux différentes parties d'un ensemble, son support est l'ensemble où la mesure est définie.

On appelle exposant de singularité au point $x_0 \in supp\mu$, la limite (2.7) :

$$\alpha(x_0) = \lim_{\varepsilon \to 0^+} \frac{\ln \mu\left(B_{x_0}(\varepsilon)\right)}{\ln \varepsilon} \tag{2.7}$$

où $B_{x0}(\varepsilon)$ désigne une boule centrée en x_0 et de taille ε.

L'exposant de singularité en un point rend compte du degré local de régularité de la mesure considérée.[11] [12]

2.3.4 Spectre multifractal

Le spectre $f(\alpha)$ des singularités associées à la mesure μ est la dimension fractale de l'ensemble des points x_0 tel que $\alpha(x_0) = \alpha$:

$$f(\alpha) = d_F\left(\left\{x_0 \in Supp\mu / \alpha(x_0) = \alpha\right\}\right) \quad (2.8)$$

Le spectre $(\alpha, f(\alpha))$ contient un grand nombre d'informations. Les α donnent une mesure locale des singularités, $f(\alpha)$ mesure le comportement fractal des points de même α.

- si $f(\alpha) \simeq 2$, on classe le point comme appartenant à une région homogène,
- si $f(\alpha) \simeq 1$, on classe le point comme appartenant à un contour régulier,
- si $f(\alpha)$ est entre 1 et 2, on classe le point comme appartenant à un contour irrégulier.

On peut effectuer de même, sur la base des informations fournies par le spectre multifractal, du débruitage et de la détection de changement dans des séquences d'images. [9]

2.3.5 Algorithme de segmentation multifractale

Il s'agit de voir le comportement des exposants de singularité locaux, c'est-à-dire calculés en chaque point de l'image, nous obtenons alors une image des exposants de singularité. [10] [11] [12]

Rappelons qu'un exposant de singularité mesure le degré de régularité et d'homogénéité d'une mesure, c'est cette caractéristique qui est exploitée dans cet algorithme. Donc, la segmentation procède par la recherche et le groupement des points ayant des exposants de singularité proches, l'algorithme de segmentation est le suivant :

- calculer les exposants de singularité en chaque point de l'image, et trouver la singularité minimale α_{min} et la singularité maximale α_{max} ;
- diviser le spectre de singularité $[\alpha_{min}, \alpha_{max}]$ en N parties égales I_k ;
- pour chaque sous intervalle I_k, trouver l'ensemble des points de l'image ayant l'exposant de singularité appartenant à l'intervalle I_k, ce qui nous permet d'avoir N sous images, chacune correspondant à un intervalle iso-alpha. Notons que pour un intervalle $I_k = [\alpha_1, \alpha_2]$ donné, les pixels retenus seront ceux dont le niveau de gris sera compris dans l'intervalle $\left[p\left(\frac{1}{L}\right)^{\alpha_2}, p\left(\frac{1}{L}\right)^{\alpha_1}\right]$.

Le calcul de l'exposant de singularité s'effectue comme suit, relation (2.9) :

$$m(I(x,y)) \approx (1/r)^{-\alpha(x,y)} \text{ quand } r \to 0 \quad (2.9)$$

avec :

$m(I(x,y))$: est la mesure associée au point (x,y) de niveau de gris $I(x,y)$.

r : est la largeur de boite entourant le point (x,y).

$\alpha(x,y)$: est l'*exposant de singularité ou de* ***Hölder*** au point (x,y).

Ce qui nous permet d'écrire l'équation suivante (2.10) :

$$\frac{p(x,y)}{p} = \left(\frac{r}{L}\right)^{\alpha(x,y)} \tag{2.10}$$

soit à l'échelle du pixel (2.11) :

$$p(x,y) = p\left(\frac{1}{L}\right)^{\alpha} \tag{2.11}$$

avec :

p :la somme des niveau de gris de toute l'image

L : la taille de l'image sur une dimension

$p(x,y)$: la somme des niveaux de gris des pixels au voisinage (x, y).

2.4 CONCLUSION

La segmentation est une phase incontournable dans l'analyse d'image. Elle permet d'extraire les informations nécessaires à la localisation et la délimitation des structures d'intérêt dans l'image. Nous avons exposé dans ce chapitre les différentes méthodes de segmentation par approche frontière, et par approche région en utilisant le formalisme multifractal qui est fréquemment adopté dans l'imagerie médicale. Ces méthodes feront aussi l'objet de notre application.

Chapitre 3

Etude de la famille C6000

CHAPITRE III

ETUDE DE LA FAMILLE C6000 (C6211)

Dans ce qui suit, nous présentons une étude de la famille C6000, en particulier le C6211, et tout ce qui est en relation avec sa programmation et notre application, sans aborder les généralités sur les DSP faisant maintenant partie du cours MSR.

3.1 INTRODUCTION

Le DSP (Digital Signal Processor) fait partie d'un type particulier de microprocesseurs. Ses fonctions spéciales ainsi que son architecture le rendent performant dans le domaine du traitement du signal et d'automatisme. Chose qui a augmenté son champ d'applications (communications, imagerie, applications militaires…). Il existe plusieurs fabriquants des DSP comme Analog Devices, Motorolla et le leader du marché Texas Instruments.

Texas Instruments a introduit la première génération TMS320C10 digital signal processor en 1982, le TMS320C25 en 1986, et le TMS320C50 en 1991. Plusieurs versions de chacun de ces processeurs C1x, C2x et C5x existent avec différentes caractéristiques, comme la vitesse d'exécution. Ces processeurs 16 bits sont tous des processeurs à virgule fixe et ont un code compatible.

Le processeur TMS320C30 à virgule flottante est introduit à la fin des années quatre-vingt. Le C31, C32 et le plus récent C33 sont tous des membres de la famille C3x des processeurs à virgule flottante. Les C4x le sont aussi, introduits ultérieurement, et ont un code compatible avec les processeurs C3x qui sont tous basés sur l'architecture de Harvard modifiée.

Il est important de faire rappeler l'avantage d'une telle architecture, qui donne un bon compromis entre l'architecture de VonNeumann pénalisante par ses faibles performances en matière de cycles d'exécution des instructions, et l'architecture de Harvard par son prix élevé.

Le TMS320C6201 (C62x), présenté en 1997, est le premier membre de la famille C6x des DSP à virgule fixe. Contrairement à ses prédécesseurs (C1x, C2x et C5x), le C62x est basé sur l'architecture very-long-instruction-word (VLIW) qui prend toujours en considération l'architecture de Harvard modifiée. L'architecture VLIW possède un jeu

d'instructions simple, qui s'avèrent plus indispensables pour les taches que dans le cas d'un DSP conventionnel.

Le code du C62x n'est pas compatible avec les générations précédentes des processeurs virgule fixe. Ultérieurement, le processeur à virgule flottante TMS320C6701 (C67x) est introduit comme un autre membre de la famille C6x des DSP TI. Le jeu d'instructions du C62x est la base du jeu d'instructions C67x. Une addition récente aux processeurs de la famille C6x est le DSP virgule fixe C64x.

Le DSP virgule fixe est privilégié dans le cas d'applications (ou périphériques) utilisant des batteries, comme les téléphones cellulaires, car il consomme moins d'énergie qu'un équivalent en virgule flottante.

Contrairement aux processeurs 16 bits à virgule fixe C1x, C2x et C5x, le DSP C6x virgule fixe (32 bits) présente une amélioration de la dynamique et de la précision. A noter qu'avec ce type de processeur, il faut mettre les données à l'échelle et prendre en considération le débordement.

Le processeur virgule flottante est en général très coûteux par son nombre de broches, le nombre de bus qu'il intègre ainsi que les circuits additionnels nécessaires pour tenir compte de l'arithmétique en virgule fixe.

Plusieurs facteurs comme le prix, la puissance consommée et la vitesse de traitement interviennent dans le choix du DSP. [14] [15]

3.2 ARCHITECTURE DU TMS320C6x

Le TMS320C6211 mis en œuvre sur la carte DSK, qui servira d'un support de développement d'applications par la suite, est un processeur à virgule fixe basé sur l'architecture VLIW. Sa mémoire interne inclut deux niveaux de mémoire cache avec 4kB pour le niveau 1 concernant la cache programme (L1P), 4kB pour le niveau concernant la cache données (L1D) et 64kB de RAM en niveau 2 pour les allocations données/programme (L2). L'environnement comprend des mémoires synchrones (SDRAM) et des mémoires asynchrones (SRAM et EPROM). Les périphérique du DSP en question sont : deux ports séries multi-canaux avec buffer (McBSPs), deux timers, un port d'interfaçage hôte de 16 bits (HPI) et une interface mémoire externe de 32 bits (EMIF). Il requière 3.3 V pour I/O et 1.8 V pour le cœur.

Le chemin de données (bus internes) est formé d'un bus d'adresse programme de 32bits, d'un bus programme de 256bits pour faire loger huit instructions de 32bits, deux bus d'adresse de données 32bits, deux bus de données 32bits et deux bus de données store de 64bits. Avec un bus de 32bits pour les adresses, l'espace mémoire total adressable est de 2^{32} = 4GB, incluant quatre espaces mémoires externes : CE0, CE1, CE2, et CE3. La figure 3.1 montre le diagramme en blocs des parties fonctionnelles du C6211.

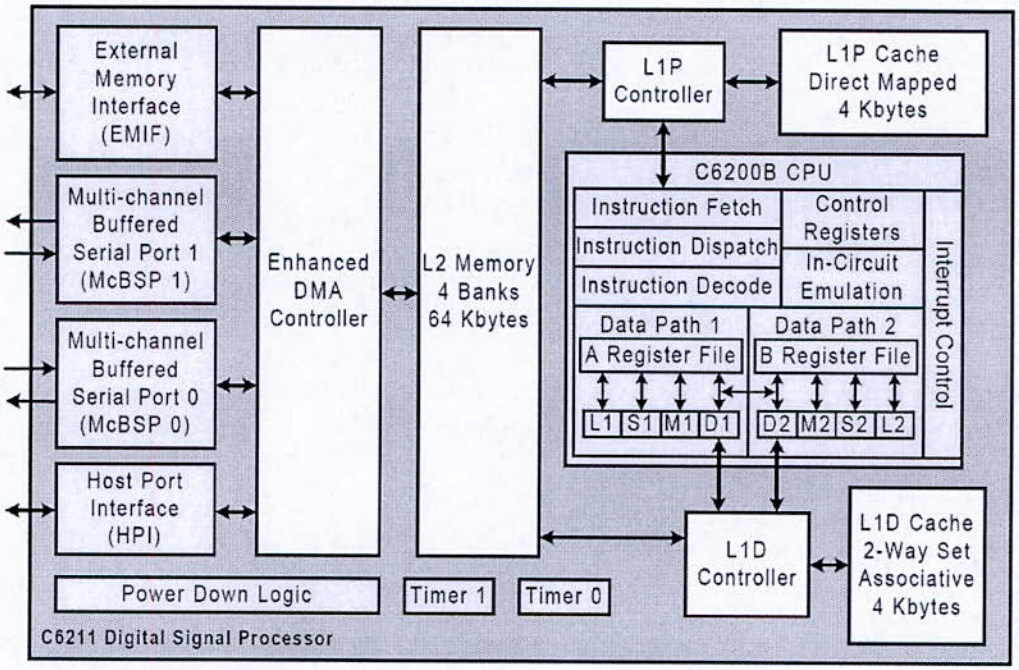


FIGURE 3.1 Block Diagram du TMS320C6211

L'indépendance des bancs mémoires des processeurs C6x permet un double accès en mémoire en un seul cycle d'instruction, et donc deux instructions loads ou stores peuvent être exécutées en parallèle tant que les données à y accéder résident dans différents blocs.

Le C6211 du DSK inclut 72kB de mémoire interne adressée à partir de l'adresse 0x00000000, et 16MB de SDRAM adressée via CE0 à partir de l'adresse 0x80000000. Le DSK inclut aussi une mémoire flash de 128kB à partir de 0x90000000. Le tableau 3.1 montre la configuration mémoire du DSP C6211. [14] [16] [17]

Tableau 3.1 Résumé, Configuration Mémoire

Address Range (Hex)	Size (Bytes)	Description of Memory Block
0000 0000—0000 FFFF	64K	Internal RAM (L2)
0001 0000—017F FFFF	24M–64K	Reserved
0180 0000—0183 FFFF	256K	Internal configuration bus EMIF registers
0184 0000—0187 FFFF	256K	Internal configuration bus L2 control registers
0188 0000—018B FFFF	256K	Internal configuration bus HPI register
018C 0000—018F FFFF	256K	Internal configuration bus McBSP 0 registers
0190 0000—0193 FFFF	256K	Internal configuration bus McBSP 1 registers
0194 0000—0197 FFFF	256K	Internal configuration bus timer 0 registers
0198 0000—019B FFFF	256K	Internal configuration bus timer 1 registers
019C 0000—019F FFFF	256K	Internal configuration bus interrupt selector registers
01A0 0000—01A3 FFFF	256K	Internal configuration bus EDMA RAM and registers
01A4 0000—01FF FFFF	6M–256K	Reserved
0200 0000—0200 0033	52	QDMA registers
0200 0034—2FFF FFFF	736M–52	Reserved
3000 0000—3FFF FFFF	256M	McBSP 0/1 data
4000 0000—7FFF FFFF	1G	Reserved
8000 0000—8FFF FFFF	256M	External memory interface CE0
9000 0000—9FFF FFFF	256M	External memory interface CE1
A000 0000—AFFF FFFF	256M	External memory interface CE2
B000 000—BFFF FFFF	256M	External memory interface CE3
C000 0000—FFFF FFFF	1G	Reserved

3.3 UNITES FONCTIONNELLES

Le CPU consiste en huit unités fonctionnelles indépendantes divisées en deux chemins de données A et B telle que montrés en figure 3.1. Chaque chemin comporte une unité pour la multiplication (.M), pour les opérations logiques et arithmétiques (.L), pour les branchements, la manipulation des bits et opérations arithmétiques (.S), et pour le loading/storing et opérations arithmétiques (.D). Les unités .S et .L sont utilisées par les instructions arithmétiques, logiques, et de branchements. Tous les transferts de données se font via les unités .D. [21]

Les opérations arithmétiques telles que celles d'addition ou de soustraction (ADD ou SUB), peuvent être exécutées par toutes les unités exceptées les unités .M. Les huit unités fonctionnelles consistent en quatre unités arithmétiques et logiques ALUs travaillant en virgule fixe (deux .L et deux .S), deux autres unités arithmétiques virgule fixe (.D), et deux multiplieurs virgule fixe (.M). Chaque unité fonctionnelle peut lire directement du ou écrire directement dans un registre du fichier à travers son propre chemin. Chaque chemin inclut un ensemble de seize registres de 32bits, de A0 à A15 et de B0 à B15. Donc il y a 32 registres à

usage général; certains sont réservés pour les modes d'adressage spéciaux ou utilisés par les instructions conditionnelles.

Deux chemins croisés (cross-paths 1x et 2x) permettant aux U.F à partir d'un chemin de données l'accès à un opérande de 32bits dans un fichier du registre du banc opposé. Un maximum de deux accès par cycle utilisant le cross-path est permis (figure3.2)

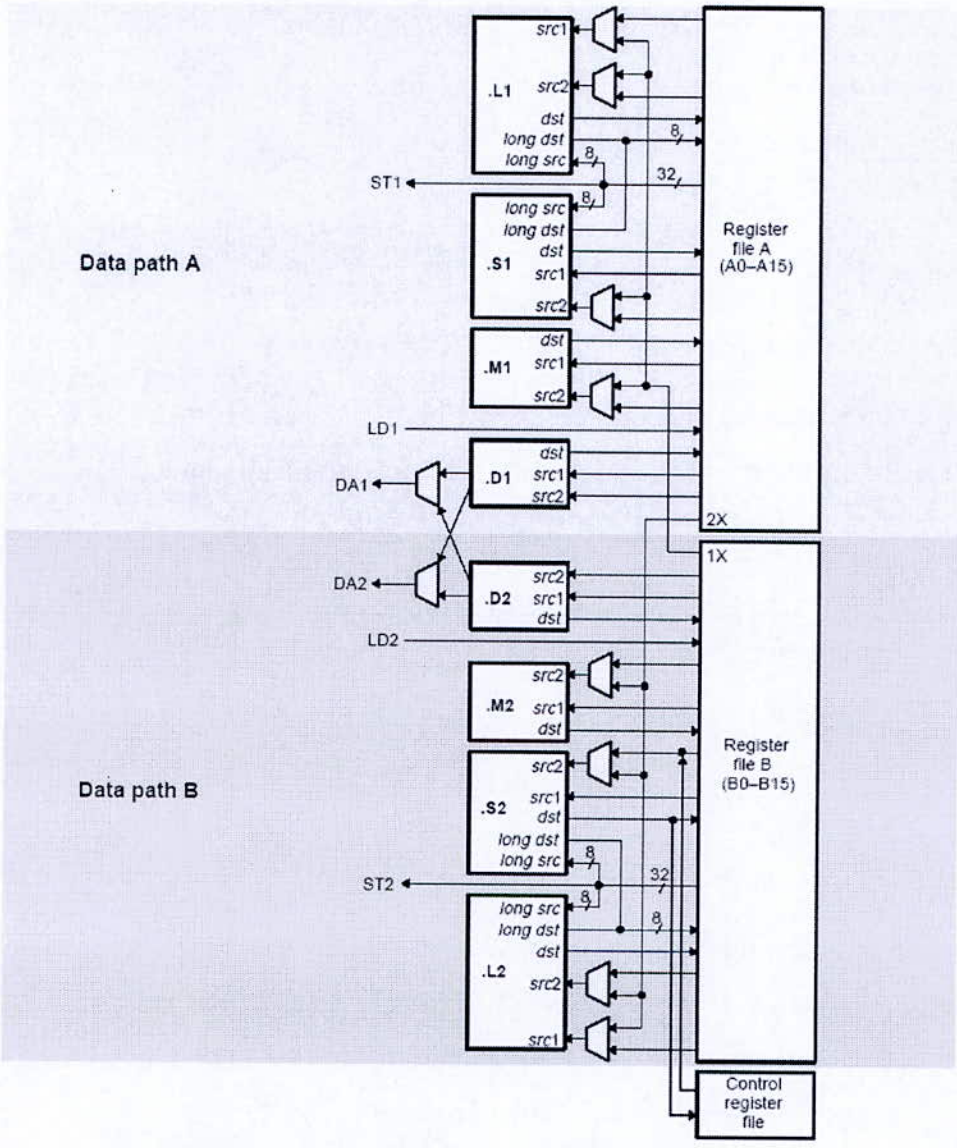


FIGURE 3.2 Chemins de données TMS320C62x

3.4 FETCH ET EXECUTE PACKETS

L'architecture VELOCITI, introduite par TI, est une dérivée de l'architecture VLIW. L'execute packet (EP) consiste en un groupe d'instructions exécutées en parallèle en un seul cycle d'horloge. Le nombre d'EP inclus dans un fetch packet (FP) peut varier de un (avec huit

instructions parallèles) à huit (sans instructions parallèles). L'architecture VLIW est modifiée pour permettre plus d'un EP. [14] [18]

Le bit le moins significatif de chaque instruction de 32bits sert à déterminer si l'instruction suivante appartient au même EP (si 1) ou fait partie de l'EP suivant (si 0).

Pour illustrer, considérons un FP avec trois EP tel que : EP1 avec deux instructions parallèles, EP2 et EP3 chacun avec trois instructions parallèles :

```
        Instruction A       |
||      Instruction B       |  EP1

        Instruction C       |
||      Instruction D       |  EP2
||      Instruction E       |

        Instruction F       |
||      Instruction G       |  EP3
||      Instruction H       |
```

Donc FP est tel qu'il apparaît sur la figure 3.3 Le bit 0 de chaque instruction de 32bits est le bit 'p', qui signale la présence ou non d'un parallélisme avec l'instruction suivante.

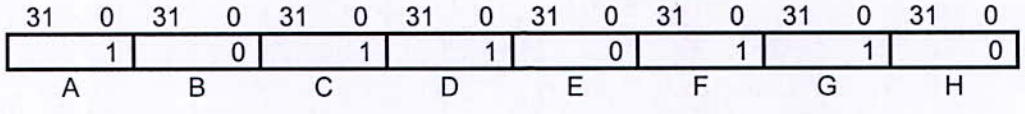


FIGURE 3.3 Un FP avec trois EPs, montrant le bit 'p' de chaque instruction

3.5 PIPELINE

Le pipeline représente la clé du parallélisme d'instructions qui nécessitent un bon timing. Il existe trois étages de pipeline : program fetch, decode, et execute.

1. Etage de recherche d'instruction (program fetch) est composé de quatre phases :
   a) PG : génération de l'adresse de l'instruction (dans le CPU).
   b) PS : envoi d'adresse d'instruction (à la mémoire).
   c) PW : attente de la disponibilité de la donnée au niveau du buffeur (lectrure en mémoire).
   d) PR : réception du paquet d'instructions au niveau du CPU.

2. Etage de décodage d'instructions, composé de deux étages :
   a) DP : dispatcher toutes les instructions contenues dans un FP sur les unités

fonctionnelles appropriées.

b) DC : décodage d'instructions.

3. Etage d'exécution : comprend six phases en virgule fixe et jusqu'à dix phases en virgule flottante, à cause des délais (latences) associés aux instructions suivantes :
   a) Instruction de Multiplication, qui consiste en deux phases à cause d'un délai.
   b) Instruction de chargement (load), qui consiste en cinq phases à cause de quatre délais.
   c) Instructions de branchement, qui consiste en six phases à cause de cinq délais.

Le tableau 3.2 représente les phases de pipeline, et le tableau 3.3 montre l'effet du pipeline. La première rangée (Tableau 3.3) représente les cycles 1,2,...,12. Chaque rangée suivante représente un FP. Les rangées représentées par PG, PS,..., illustrent la phase associée à chaque FP. Le PG du premier FP commence au premier cycle, et le PG du second FP commence au deuxième cycle, et ainsi de suite. On suppose qu'un FP contient un EP.

Tableau 3.2 Phases de Pipeline

Recherche d'instruction				Décodage		Exécution
PG	PS	PW	PR	DP	DC	E1-E6

Par exemple, au septième cycle, lorsque les instructions du premier FP sont dans la première phase d'exécution E1, les instructions du second FP sont en phase de décodage, et les instructions du troisième FP sont en phase de dispatching, et ainsi de suite. Toutes les sept instructions sont en avance à travers les différentes phases. Ainsi, au septième cycle, le pipeline est plein 'the pipeline is full'.

Tableau 3.3 Effets du Pipeline

Cycle d'horloge											
1	2	3	4	5	6	7	8	9	10	11	12
PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6
	PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5
		PG	PS	PW	PR	DP	DC	E1	E2	E3	E4
			PG	PS	PW	PR	DP	DC	E1	E2	E3
				PG	PS	PW	PR	DP	DC	E1	E2
					PG	PS	PW	PR	DP	DC	E1
						PG	PS	PW	PR	DP	DC

La plupart des instructions prennent une seule phase d'exécution. Les instructions telles que la multiplication (MPY), chargement (LDH/LDW), et branchement (B) prennent deux,

cinq, et six phases respectivement. [14] [18]

3.6 LES REGISTRES

Deux fichiers contenant chacun un enemble de registres (A0-A16 et B0-B16). Les registres A0, A1, B0, B1 et B2 sont utilisés comme étant des registres conditionnels. Les registres de A4 à A7 et de B4 à B7 utilisés par l'adressage circulaire. Les registres de A0 à A9 et de B0 à B9 (excepté le B3) sont temporairement des registres à usage général. Les registres de A0 à A15 et de B10 à B15 utilisés sont enregistrés et après restaurés avant le retour d'une subroutine.

Une donnée sur 40bits peut être rangée dans une paire de registres. Les 32bits de poid faible sont stockés dans les registres à indice pair (e.g, A2) et les 8bits qui restent sont stockés dans les 8LSBs du registre impaire suivant (e.g, A3).

Ces 32 registres sont considérés des comme registres à usage général. Plusieurs registres spéciaux existent aussi pour le contrôle et les interruptions; par exemple : le registre de mode d'adressage (AMR) utilisé pour l'adressage circulaire, et les registres du contrôle d'interruptions. [14] [21]

3.7 LES MODES D'ADRESSAGE LINEAIRES ET CIRCULAIRES

Les modes d'adressages déterminent la façon d'accéder à la mémoire. Ils spécifient aussi la manière avec laquelle on manipule les données, comme la recherche d'un opérande indirectement à partir d'une position mémoire. Sur le C6x les deux modes d'adressage, linéaire et circulaire, sont supportés. Le plus communément utilisé est le mode d'adressage indirect de la mémoire.[18]

3.7.1 Adressage indirect

L'adressage indirect peut être utilisé avec ou sans déplacement. Le registre R représente un des 32 registres de A et B, qui peut se pointer sur les adresses mémoires. A proprement parler, ces registres sont des pointeurs. Le mode d'adressage indirect utilise " * " en conjonction avec l'un des 32 registres. Pour illustrer, considérons le registre d'adresse R.

1. *R. Le registre R contient l'adresse de la position mémoire où la valeur de la donnée est stockée.

2. *R++(d). Le registre R contient une adresse. Après son utilisation, le registre R est post-incrémenté telle que la nouvelle adresse est l'adresse actuelle avec un déplacement d. Le même raisonnement est valable dans le cas (--).
3. *++R(d). L'adresse est préincrémentée d'un déplacement d, telle que l'adresse courante est R+d. Un double ' - ' introduit une prédécrémentation de l'adresse mémoire telle que la courante sera R-d.
4. *+R(d). L'adresse est préincrémentée par d telle que l'adresse courante est R+d. Cependant, dans ce cas, R est préincrémenté sans modification. Contrairement au cas précédent, R n'est pas mis à jour ou modifié.

3.7.2 Adressage circulaire

L'adressage circulaire est utilisé pour créer un buffer circulaire au niveau matériel, qui est fréquemment utilisé par plusieurs algorithmes du traitement du signal. Exemples, les filtres numériques et corrélations.

Le C6x a une architecture dédiée mettant en oeuvre l'adressage circulaire, qui peut être utilisé en conjonction avec un buffer circulaire afin de mettre à jour les échantillons en remplaçant l'ancien échantillon par le nouveau et éviter le décalage direct de données. Lorsque le pointeur atteint la position fin du buffer circulaire qui contient le dernier élément (échantillon) du buffer, et après son incrémentation, le pointeur automatiquement pointe sur la position début du buffer circulaire qui contient le premier élément.

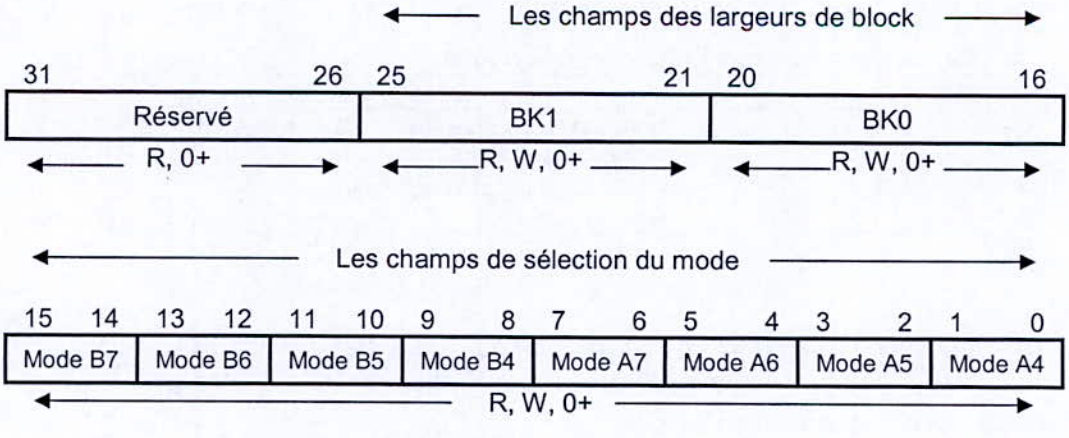


FIGURE 3.4 Registre des modes d'adressage

Deux buffers circulaires indépendants sont disponibles en utilisant BK0 et BK1 avec le registre mode d'adressage (AMR). Les huit registres de A4 à A7 et de B4 à B7, en conjonction avec les unités .D, peuvent être utilisés comme pointeurs (NB : tous les registres peuvent être utilisés en adressage linéaire). Le segment de code suivant illustre l'utilisation du buffer circulaire en utilisant le registre B2 pour configurer le AMR (seul la partie B intervient dans la configuration des registres spéciaux) :

```
MVK    .S2    0x0004,B2   ;les 16bits faibles de B2 sont affectés. Le A5 est sélectionné
                           comme pointeur
MVKLH .S2     0x0005,B2   ;les 16bits forts de B2 sont affectés. Sélection de B0, fixer N=5
MVC    .S2    B2,AMR      ;transfert des 32bits de B2 vers le AMR
```

Les deux instructions MVK et MVKLH, utilisant l'unité .S, font transférer la valeur 0x0004 dans les 16LSBs du registre B2 et la valeur 0x0005 dans les 16MSBs de B2. L'instruction MVC (transfert de constante) est la seule instruction qui peut avoir l'accès au registre AMR et autres registres de contrôle, comme elle est exécutée exclusivement sur la partie B en conjonction avec les registres et unités fonctionnelles de celle-ci. En résumé, une valeur de 32bits créee dans B2, est transférée par la suite au registre AMR via l'instruction MVC.

La valeur 0x0004 = $(0100)_B$ positionne le bit 2 du AMR à 1 (les autres à zéro). Ce qui se traduit par le choix du mode 01 et prendre A5 comme pointeur du buffer circulaire utilisant le block BK0. La table 3.4 présente les modes associés avec les registres pointeurs.

La valeur 0x0005 = $(0101)_B$ positionne le 16ème et 18ème bit du AMR à 1. Cela correspond à la valeur N utilisée pour fixer la taille du buffer i.e $2^{N+1} = 64$ en utilisant le BK0.

Tableau 3.4 Description des Modes du AMR

Mode		Description
0	0	Adressage linéaire (par défaut, reset)
0	1	Adressage circulaire utilisant BK0
1	0	Adressage circulaire utilisant BK1
1	1	Réservé

3.8 LE JEU D'INSTRUCTIONS DU TMS320C6x

3.8.1 Format du code assembleur

Un code en assembleur C6x est représenté par le champ suivant [16]:

```
Label      ||    [ ]   Instruction      Unit  Operands    ;comments
```

Un label (i.e étiquette), s'il est présent, indique l'adresse ou la position mémoire qui contient l'instruction ou la donnée. L'étiquette doit être sur la première colonne. Les deux barres parallèles (| |) sont ajoutées si l'instruction s'exécute en parallèle avec celle qui la précède. Le champ qui suit est optionnel pour rendre l'instruction associée conditionnelle. Pour ce faire, cinq registres sont disponibles à savoir A1, A2, B0, B1 et B2. Par exemple, [A1] indique que l'instruction associée est exécutée si A1 est différent de zéro ; alors qu'avec [!A1] l'instruction associée est exécutée si A1 est égale à zéro. On peut dire que toutes les instructions du C6x sont conditionnelles ; c'est l'une des caractéristiques de l'architecture VLIW. Le champ '`Instruction`' peut être soit une directive (i.e une commande pour l'assembleur) ou bien un mnémonique. Exemple :

```
.word       value
```

Cette directive réserve 32 bits en mémoire pour '*value*'. Le champ '`Unit`', qui peut être l'une des huit unités fonctionnelles, est optionnel. Ce champ permet de désigner l'unité qui sera responsable de l'exécution de l'instruction. Les commentaires mis sur la première colonne sont soit précédés par un astérisque ou un point virgule, autrement par un point virgule.

3.8.2 Directives 'assembleur'

Une directive 'assembleur' n'est pas une instruction, c'est un message pour l'assembleur (pas le compilateur). Prise en compte lors du processus d'assemblage, la directive n'occupe pas d'espace mémoire comme l'instruction, et ne produit pas un code exécutable.

Les adresses des différentes sections peuvent être spécifiées par des directives. Par exemple,

```
.sect       'my_section'
```

L'assembleur crée diverses sections grâce aux directives, qu'on verra ultérieurement, telles que .text pour le code exécutable, .data pour les données et .bss pour la déclaration des variables.[18]

Autres directives fréquemment utilisées :

`.short` : initialiser un entier de 16bits.

`.int` : initialiser un entier de 32bits (aussi `.word` ou `.long`).

`.float` : initialiser une constante selon le standard virgule flottante IEEE-32bits simple précision.

`.double` : initialiser une constante selon le standard virgule flottante IEEE-64bits double précision.

3.9 ASSEMBLAGE LINEAIRE

Une alternative au C, ou au code assembleur est l'assemblage linéaire 'linear assembly'. Il s'agit d'écrire un code en assembleur sans tenir compte du parallélisme et de l'affectation des opérations aux unités correspondantes. C'est une solution qui présente un compromis entre la complexité du code assembleur et son efficacité, et cela grâce à l'optimisateur du code assembleur (Assembler optimizer). Comme procédure, une formulation du programme source en linear assembly (fichier .sa) génére un programme source en assembleur (.asm) ; de la même façon le compilateur C (avec options d'optimisations) est utilisé en conjonction avec le code C du programme source. [14]

3.10 TIMERS

Grâce à deux timers de 32bits, on peut cadencer les périphériques, compter les évènements, générer des impulsions de commande, interrompre le CPU, et déclencher le transfert à travers le DMA. [21]

3.11 LES INTRERRUPTIONS

L'interruption, qui est une rupture de séquence asynchrone, est générée suite à un événement interne ou externe (le plus probable). Lors d'une interruption, le traitement courant du processeur est arrêté de manière à exécuter la tache associée ; il s'agit d'une routine d'interruption surveillée par le service des routines d'interruption (ISR) ; et cela tout en sauvegardant les conditions et les registres du programme pour être restaurés lors du retour. On rencontre seize sources d'interruption incluant deux des timers, quatre interruptions externes, quatre interruptions McBSP, et quatre interruptions DMA. Douze des interruptions sont des interruptions CPU. [14] [21]

3.11.1 Registre de contrôle d'interruptions

Les registres de contrôle d'interruptions sont les suivants [16]:

1. CSR (control status register) : contient le bit (GIE : global interrupt enable) et autres bits de contrôle et d'état, figure 3.5

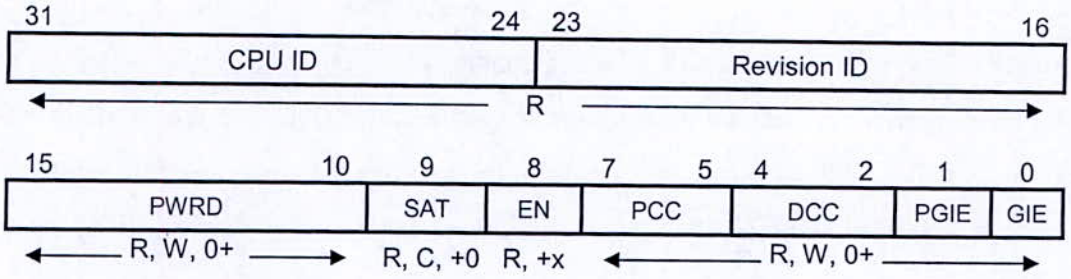


FIGURE 3.5 Registre de contrôle et d'état

2. IER (interrupt enable register) : pour activer/désactiver des interruptions, figure 3.6

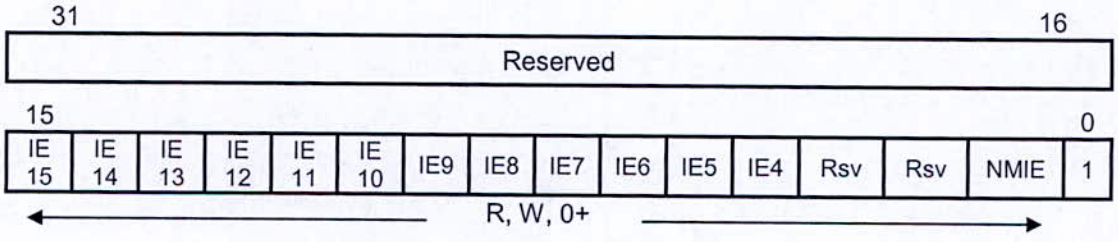


FIGURE 3.6 Registre d'activation d'interruptions

3. IFR (interrupt flag register) : montre l'état des interruptions
4. ISR (interrupt set register) : permet de positionner un bit dans l'IFR manuellement
5. ICR (interrupt clear register) : permet d'effacer un bit dans l'IFR manuellement
6. ISTP (interrupt service table pointer) : pour localiser l'IST, il se pointe au début du tableau service d'interruption, figure 3.7

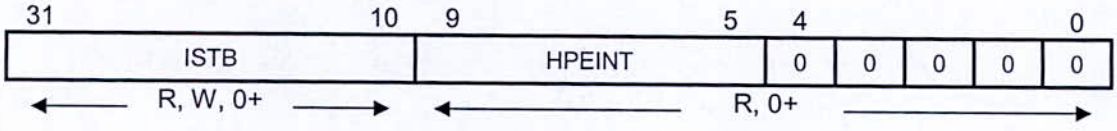


FIGURE 3.7 Pointeur du tableau de service d'interruptions

7. IRP (interrupt return pointer) : porte l'adresse de retour d'une interruption masquable

8. NRP (nonmaskable interrupt return pointer) : porte l'adresse de retour d'une interruption non masquable

Les interruptions non masquables sont les plus prioritaires, avec le reset ayant la haute priorité suivie de la NMI. Le registre IER est utilisé pour positionner une interruption, vérifier et reconnaître l'interruption survenue à partir du registre indicateur d'interruptions (IFR).

La NMI peut être désactivée en mettant à zéro le bit NMIE du CSR. Si NMIE est mis à zéro, toutes les interruptions de INT4 à INT15 sont désactivées.

Le reset (activation bas) est utilisé pour arrêter le CPU, alors que le signal NMI alerte le CPU en cas d'éventuels problèmes matériels. L'ordre de priorité des interruptions masquables et non masquables est présenté suivant le tableau des services d'interruptions (tableau 3.5).

Pour traiter une interruption masquable, il faut vérifier que les bits suivants sont positionnés :

1. le bit GIE de CSR
2. le bit NMIE de IER
3. le bit IE correspond de IER
4. le bit du IFR correspondant

Pour qu'une interruption se produise, l'unité centrale de traitement ne doit pas être entrain d'exécuter une instruction de branchement.

Tableau 3.5 Service d'interruption

Interrupt	Offset
RESET	000h
NMI	020h
Reserved	040h
Reserved	060h
INT4	080h
INT5	0A0h
INT6	0C0h
INT7	0E0h
INT8	100h
INT9	120h
INT10	140h
INT11	160h
INT12	180h
INT13	1A0h
INT14	1C0h
INT15	1E0h

Le tableau service d'interruption (IST, tableau 3.5) est utilisé lorsqu'une interruption commence. A chaque interruption est associé un FP. La table contient 16 FPs, chacun avec huit instructions. Les adresses, à droite, correspondent à un offset associé avec chaque interruption spécifique. Par exemple, le FP pour INT11 est à l'adresse de base plus l'offset de 160h. Etant donné que chaque FP contient huit instructions de 32bits, chaque offset dans le tableau est incrémenté de 20h soit 32.

Le FP du reset doit être à l'adresse 0. Cependant, les FPs associés aux autres interruptions peuvent être déplacés en mentionnant l'adresse de déplacement au niveau du ISTP (la partie ISTB : interrupt service table pointer).

Le tableau 3.6 montre les valeurs qui peuvent être sélectionnées pour choisir un type spécifique d'interruption.

Tableau 3.6 Sélection d'interruption

Interrupt Selector	Type	Description
00000	DSPINT	Host port to DSP interrupt
00001	TINT0	Timer 0 interrupt
00010	TINT1	Timer 1 interrupt
00011	SD_INT	EMIF SDRAM timer interrupt
00100	EXT_INT4	External interrupt pin 4
00101	EXT_INT5	External interrupt pin 5
00110	EXT_INT6	External interrupt pin 6
00111	EXT_INT7	External interrupt pin 7
01000	DMA_INT0	DMA channel 0 interrupt
01001	DMA_INT1	DMA channel 1 interrupt
01010	DMA_INT2	DMA channel 2 interrupt
01011	DMA_INT3	DMA channel 3 interrupt
01100	XINT0	McBSP0 transmit interrupt
01101	RINT0	McBSP0 receive interrupt
01110	XINT1	McBSP1 transmit interrupt
01111	RINT1	McBSP1 receive interrupt

3.11.2 Reconnaissance d'interrupption

Les signaux IACK et INUMx (INUM0 à INUM3) sont représentés par des broches sur le C6x qui reconnaissent l'interruption survenue et en cours d'exécution. Les quatre signaux INUMx indiquent le numéro d'interruption en cours d'exécution. Exemple :

```
INUM3 = 1 (MSB), INUM2 = 0, INUM1 = 1, INUM0 = 1 (LSB)
```

Ce qui correspond à $(1011)_b = 11$, cela signifie que INT11 est en traitement.

Le bit IE11 est mis à 1 pour activer INT11. On peut vérifier que le bit IF11 est mis à 1 aussi.

Toutes les interruptions se mettent en attente lorsque une instruction de branchement n'est pas achevée. Vu que l'instruction de branchement prend cinq cycles, une boucle ayant un retard inférieur à six cycles ne peut être interrompue.

3.12 LES McBSP

Deux multichannels buffered serial ports (McBSPs) sont disponibles qui servent d'interfaces pour les périphériques standard. Comme ils présentent des avantages tels que une communication en mode full-duplex, une synchronisation et datagrammes indépendants pour l'émission et la réception, ainsi qu'une interface directe avec le AC97 et dispositifs conformes d'IIS. Ils permettent diverses tailles de données entre 8 et 32 bits.[14] [21]

La communication de données externe peut se produire pendant que des données sont déplacées intérieurement. La figure 3.8 montre un schéma fonctionnel interne d'un McBSP. Les broches transmission de données (DX) et réception de données (DR) sont employées pour la communication de données. Le contrôle des paramètres (horloge et synchronisation des trames) s'effectue via CLKX, CLKR, FSX, et FSR. Le CPU ou le contrôleur DMA lit les données à partir du registre de réception de donnée (DRR) et écrit les données à transmettre dans le registre de transmission de données (DXR). Le registre à décalage de transmission (XSR) envoie les données à travers DX, alors que le registre à décalage de réception (RSR) copie les données reçues sur le DR dans le buffer du registre de réception (RBR). Les données dans RBR sont transférées par la suite à DRR et lues par l'unité centrale de traitement ou le contrôleur DMA.

On trouve d'autres registres comme le registre de contrôle du port série (SPCR), le registre de contrôle d'émission/émission (XCR/RCR), le registre d'activation des canaux d'émission/réception (XCER/RCER), et le registre de contrôle de broche (PCR).

Les deux McBSP sont utilisés pour l'entrée et la sortie à travers le codec du DSK. McBSP0 est utilisé pour le contrôle et McBSP1 pour l'émission et la réception des données.

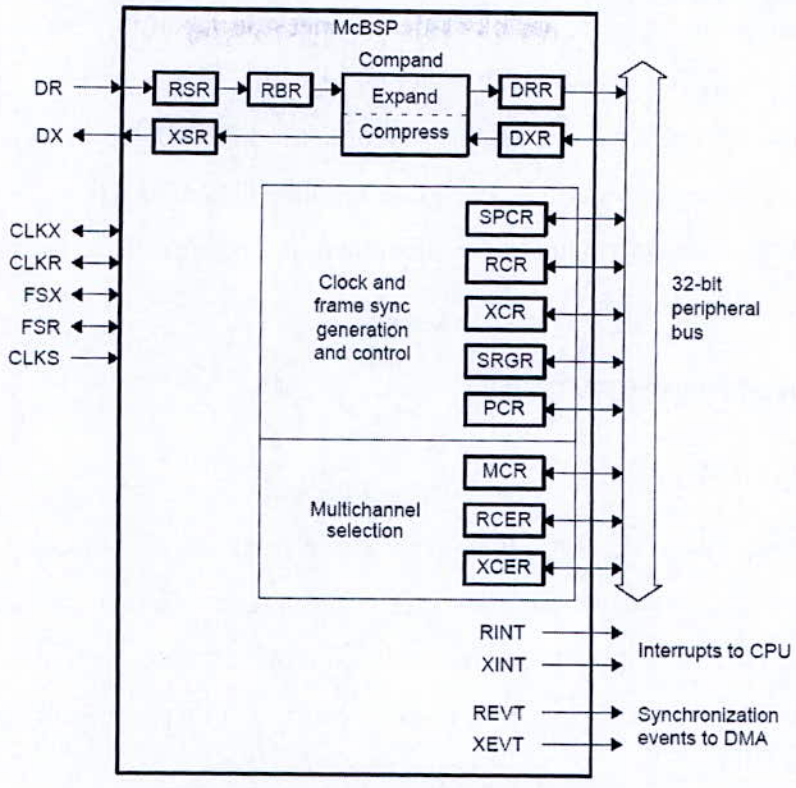


FIGURE 3.8 Le schéma interne du McBSP

3.13 L'ACCES MEMOIRE DIRECT (DMA)

L'accès mémoire direct (DMA) tient compte du transfert des données vers et de la mémoire interne ou des périphériques externes sans interposition du CPU. Quatre canaux d'accès direct à la mémoire peuvent être configurés indépendamment pour le transfert de données. Un canal (auxiliaire) additionnel est disponible pour le DMA avec l'interface du port hôte (HPI). Le DMA peut accéder à la mémoire du processeur et à l'interface mémoire externe (EMIF). Des données de différentes tailles peuvent être transférées : octets, mot de 16 bits, et mots de 32 bits.

Un certain nombre de registres de DMA sont utilisés pour le configurer : adresse (source et destination), index, recharge de compteur, données globales de DMA et registres de contrôle. La source et les adresses de destination peuvent être une mémoire interne de programme, une mémoire interne de données, une interface externe de mémoire, et bus de périphériques internes. Les transferts de DMA peuvent être déclenchés aussi bien par des interruptions à partir de périphériques internes qu'à partir des broches externes.

Pour chaque ressource, chaque canal d'accès direct à la mémoire peut être programmé pour des priorités avec le CPU. Entre les quatre canaux du DMA, le canal 0 a la priorité la plus élevée et le canal 3 a la plus basse priorité. Chaque canal d'accès direct à la mémoire peut

être utilisé pour lancer le transfert par blocs des données de façon indépendante. Un bloc de données contient des trames de même ou différent contexte, et chaque trame est formée d'un nombre d'éléments. Le registre recharge des compteurs du DMA contient une valeur pour indiquer le nombre de trames (16 MSBs) et le nombre d'éléments (16 LSBs). EDMA est également disponible avec 16 canaux indépendamment programmables. [14] [21]

3.14 CONSIDERATIONS MEMOIRE

3.14.1 Allocation de données

Des blocs de code et de données peuvent être répartis dans la mémoire sous forme de sections définies dans le fichier de commande (.cmd). Ces sections peuvent être initialisées ou non initialisées. Les sections initialisées ou non initialisées, excepté .text, ne peuvent pas être définies dans la mémoire interne de programme [18]. Les sections initialisées sont :

1. `.cinit` : pour les variables globales et locales
2. `.const` : pour les constantes globales et locales
3. `.switch` : contient les vecteurs des sauts
4. `.text` : pour le code exécutable et les constantes

Et les sections non initialisées :

1. `.bss` : pour les variables globales et locales
2. `.far` : pour les variable globales et locales déclarées comme « far »
3. `.stack` : réserve de la mémoire pour la pile
4. `.sysmem` : réserve l'espace pour les allocations de mémoire utilisées par les fonctions telles que malloc, calloc et realloc.

Le linker peut être utilisé pour placer certaines sections, telle que .text, dans la mémoire interne pour plus d'efficacité des opérations.

3.14.2 Alignement de données

Le C6x accède toujours aux données alignées qui lui permettent d'adresser des bytes, des mots de 16 bits, et des mots de 32 bits. Le format de données se compose soit de quatre limites de byte, soit de deux limites de mot 16bits, ou d'une limite de mot 32bits. Par exemple, pour assigner un chargement de 32 bits avec LDW, l'adresse doit être alignée avec une limite

de mot 32bits de sorte que les 2 bits inférieurs de l'adresse soient zéro. Autrement, des données incorrectes peuvent être chargées.

3.14.3 Directives pragma

Les directives pragma obligent le compilateur à prendre certaines fonctions en considération. Les pragma incluent `DATA_ALIGN`, `DATA_SECTION`, et autres. Exemple, la pragma `DATA_SECTION` a la syntaxe :

```
#pragma DATA_SECTION (symbol, "my_section");
```

Cette directive permet de réserver un espace pour *symbol* dans la section my_section.

Une autre directive utile,

```
#pragma MUST_ITERATE (20,20);
```

Informe le compilateur sur le nombre de fois d'exécution de la boucle. [23]

3.14.4 Modèles de mémoire

Le compilateur génère un petit modèle de mémoire (small model memory). Chaque donnée est manipulée comme si déclarée *near* à moins qu'on la déclare spécifiquement *far*. Si la pragma DATA_SECTION est employée, la donnée est indiquée en tant que variable *far*.

Un modèle de mémoire large peut être produit avec les options de l'éditeur de liens mlx (x = 0 à 4). Si aucun niveau n'est indiqué, les données et les appels de fonctions sont déclarés *near*. Ces modèles peuvent être employés en cas d'appels de fonction plus loin que 1M mots. [14] [23]

3.15 TYPES DE DONNEES

Les types de données supportées par la famille C6000 en général sont :

1. `short` : de taille 16 bits représenté en complément à 2, avec une plage de variation de -2^{15} à $(2^{15} - 1)$
2. `int` or `signed int` : de taille 32 bits représenté en complément à 2, avec une dynamique entre -2^{32} à $(2^{15} - 1)$
3. `float` : de taille de 32 bits représentée selon la norme IEEE 32-bit avec une dynamique de $2^{-126} = 1.175494 \times 10^{-38}$ jusqu'à $2^{+128} = 3.40282346 \times 10^{38}$.
4. `double` : de taille de 64 bits représentée selon la norme IEEE 64-bit avec une dynamique de $2^{-1022} = 2.22507385 \times 10^{-308}$ jusqu'à $2^{+1024} = 1.79769313 \times 10^{+308}$.

Le processeur C6211, qui est un DSP virgule fixe, ne supporte pas ces deux derniers types.

Pour la multiplication virgule fixe, l'utilisation des données de type *short* par exemple est plus efficace (moins de cycles) que d'utiliser le type *int*. L'utilisation des constantes peut également augmenter les performances du code. En ce qui concerne la division, elle est réalisée par l'algorithme de Newton-Raphson. [18]

3.16 OPTIMISATION DU CODE

Des techniques d'optimisation, telle que l'utilisation des options du compilateur et les 'intrinsics' en addition avec les directives pragma déjà traitées, présentent la deuxième étape après validation du fonctionnement du code C et avant le passage à une optimisation au niveau assembleur. [19] [14]

3.16.1 Options du compilateur

Quand l'optimisateur est appelé, les étapes suivantes sont exécutées. Un programme en C d'abord passé par un analyseur syntaxique qui exécute des fonctions de prétraitement et génère un fichier intermédiaire (.if) qui devient le fichier d'entrée de l'optimisateur. L'optimisateur génère par la suite un fichier (.opt) qui passe par le générateur de code pour d'autres optimisations et obtenir un fichier ASM (figure 3.9).

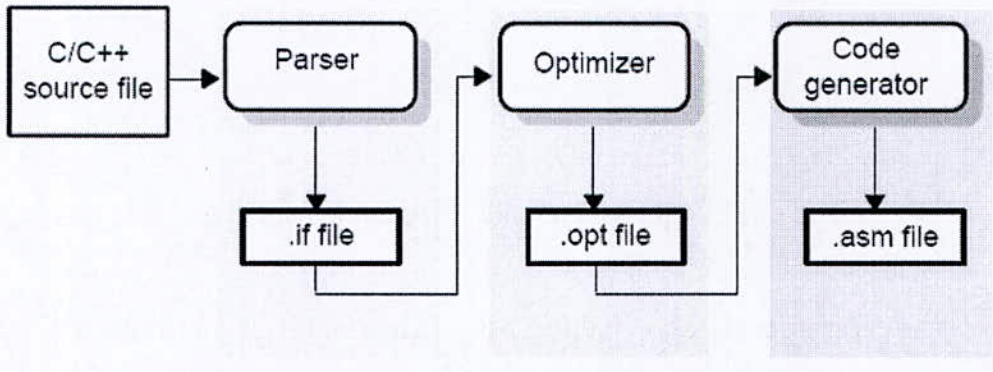


FIGURE 3.9 Compilation d'un programme C/C++ avec optimisations

Les options sont, en résumé (détails, voir [spru187k.pdf]) :

1. –o0 : optimise l'utilisation des registres.
2. –o1 : optimisation locale des fonctions en addition à l'optimisation précédente.

3. –o2 : optimisation globale en addition des optimisations –o0 et –o1.
4. –o3 : optimisation du fichier plus les optimisations précédentes.

3.16.2 Fonction C « intrinsics »

Des fonctions en C, appelées intrinsics functions, sont disponibles pour augmenter l'efficacité du code exécutable :

1. `int _mpy()` : est l'équivalent de l'instruction MPY en ASM, qui multiplie les 16 LSBs d'un nombre par les 16 LSBs d'un autre nombre.
2. `int _mpy()` : est l'équivalent de l'instruction MPYH en ASM, qui multiplie les 16 MSBs d'un nombre par les 16 MSBs d'un autre nombre.
3. `int _mpylh()` : est l'équivalent de l'instruction MPYLH en ASM, qui multiplie les 16 LSBs d'un nombre par les 16 MSBs d'un autre nombre.
4. `int _mpyhl()` : est l'équivalent de l'instruction MPYHL en ASM, qui multiplie les 16 MSBs d'un nombre par les 16 LSBs d'un autre nombre.
5. `int _sadd()` : est l'équivalent de l'instruction SADD en ASM, faisant la somme de deux nombres entiers et sature le résultat.
6. `int _ssub()` : est l'équivalent de l'instruction SSUB en ASM, faisant la soustraction entre deux nombres entiers et sature le résultat.

Il est clair qu'il s'agit de la version C des fonctions fréquemment utilisées qui permettent un accès direct aux équivalentes en ASM. [22]

3.17 CONCLUSION

A travers cette étude, nous avons pu voir certaines améliorations apportées aux DSP C6000 par rapport à ses prédécesseurs, sur le plan architectural :

- ✓ Architecture VILOCITI
- ✓ Interfaces de communication

et de programmation :

- ✓ performances du compilateur C
- ✓ options d'environnement

C'est une cible qui convient très bien lorsqu'il s'agit d'un traitement d'image en temps réel.

Chapitre 4

Outils de développement

CHAPITRE IV

OUTILS DE DEVELOPPEMENT

4.1 INTRODUCTION

Ce chapitre présente plusieurs outils disponibles pour le traitement numérique du signal avec le DSP. Ces outils incluent le code composer studio (CCS) qui fournit un environnement de développement intégré (IDE) ; le DSP Starter kit (DSK) avec le processeur TMS320C6211 à virgule fixe et un support d'entrée sortie et MATLAB.

4.2 CODE COMPOSER STUDIO

Code Composer Studio (CCS) est un environnement intégré de développement de code pour les DSP de Texas Instrument. Il est fourni en standard avec la carte de développement pour le DSP.

CCS fournit plusieurs outils pour faciliter la construction et la mise au point des programmes de DSP. Il comprend un éditeur de code source, un compilateur de langage C/C++, un assembleur de code source, un éditeur de liens et un environnement d'exécution qui permet de télécharger un programme exécutable sur une carte cible, de l'exécuter et de le déboguer au besoin. CCS comprend aussi des outils qui permettent l'analyse en temps réel d'un programme en cours d'exécution et des résultats produits. Finalement, il fournit un environnement de gestion de fichiers qui facilite la construction et la mise au point des programmes.

4.2.1 Outils de génération de code

Le code composer (CCStudio IDE) fournit une interface graphique adéquate pour l'utilisation des outils de génération de code (code generation tools). Il garde en permanence les informations utilisées pour construire et compiler un programme ou une librairie, la figure 4.1 montre le processus de développement du code exécutable. [22] [23]

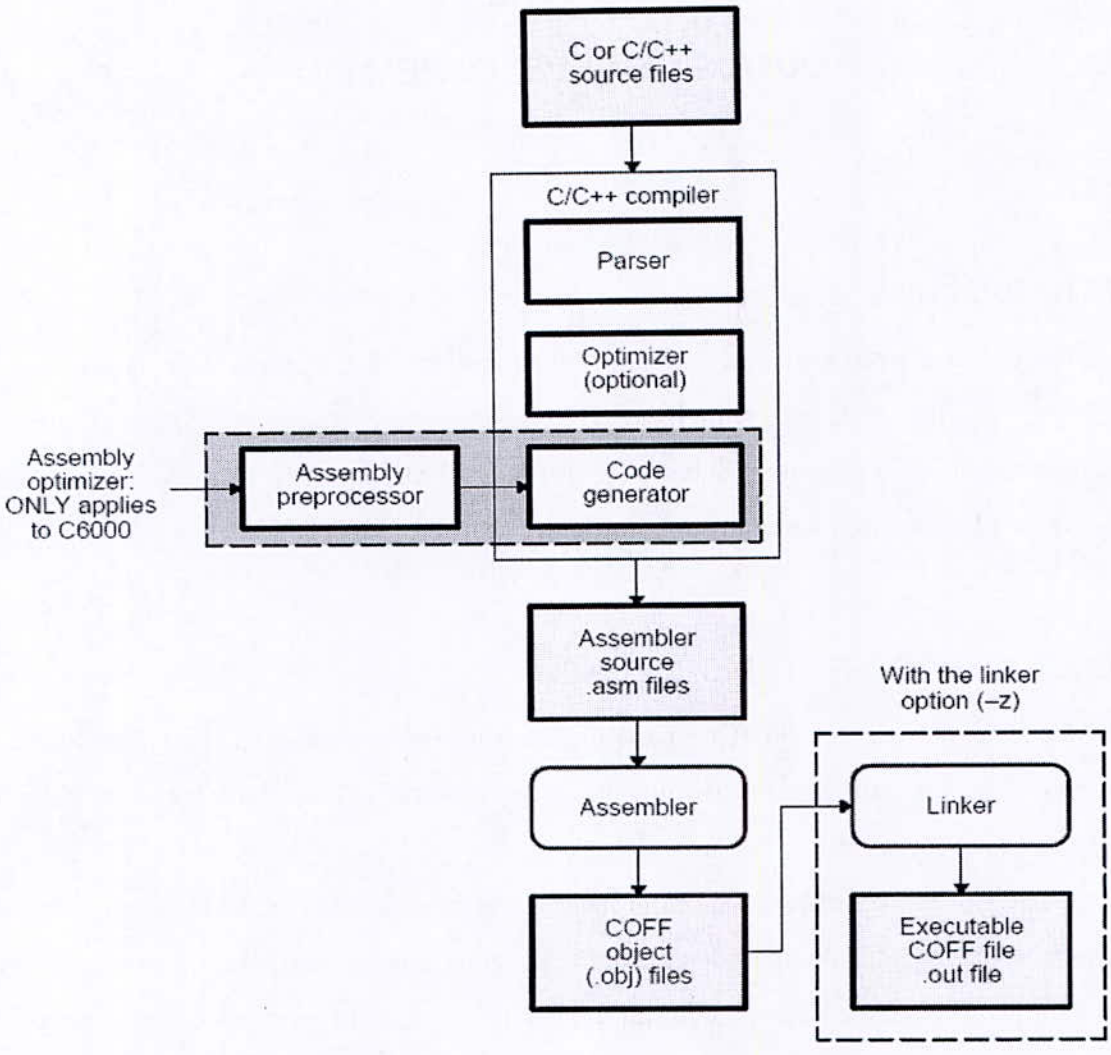


FIGURE 4.1 Processus de développement du code exécutable

Un fichier projet, dont nous allons voir sa création plus loin, enregistre :

- les noms de fichiers du code source et la librairie des projets ;
- des options de compilation, d'assemblage et de lien ;
- il inclut aussi les dépendances des fichiers.

Quand on construit un programme avec le code composer, l'outil de génération du code le mieux approprié est utilisé pour compiler, assembler et/ou faire intervenir l'éditeur de lien du programme. Les options de compilation, d'assemblage et du linker peuvent être spécifiées et configurées via la boite de dialogue « Build options », figure 4.2. On peut avoir l'accès à presque toutes les options. Celles qui ne sont pas représentées peuvent être introduites aisément à travers la ligne de commande.

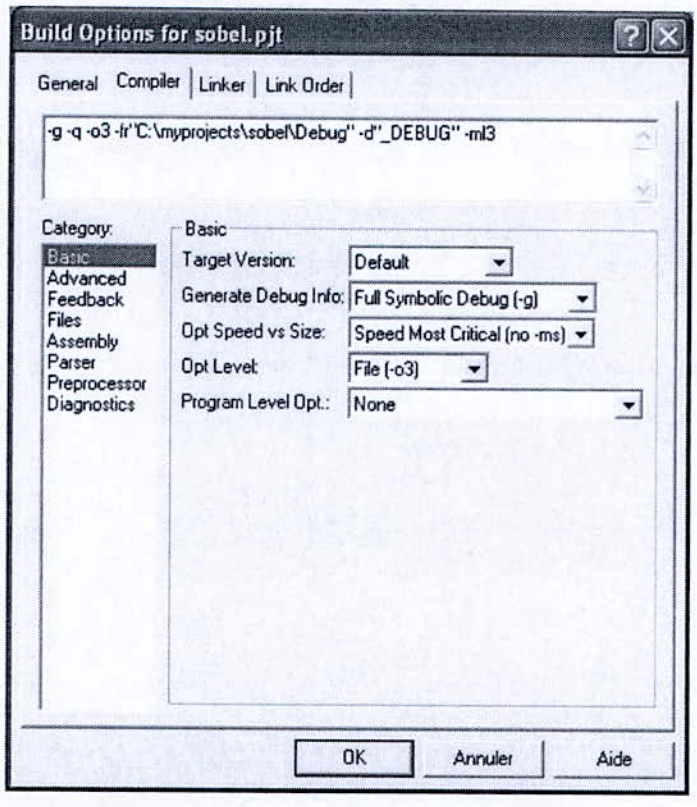


FIGURE 4.2 Boite de dialogue Build Options

4.2.1.1 Outils de développement pour l'assembleur

Les outils de développement de langage d'assemblage sont les suivants [23]:

Assembleur : l'assembleur traduit les fichiers source, écrits en langage assembleur, en fichiers objet de langage machine. Le langage machine est basé sur le format de fichier « common object file format » (COFF).

Archiver : l'archiver permet de rassembler un groupe de fichiers dans un seul fichier d'archives simple appelé *une bibliothèque*. De plus, il permet de modifier une bibliothèque en supprimant, remplaçant, extrayant, ou ajoutant des membres (routines). Une des applications les plus utiles de l'archiver est de construire une bibliothèque des modules d'objet.

Éditeur de liens (Linker) : l'éditeur de liens combine les fichiers objet dans un fichier objet exécutable par la cible. Pendant la création du fichier exécutable, il exécute la mise en mémoire adressable disponible, selon la cible, des fonctions définies par le fichier de commande et associe les références externes.

Absolute Lister : il accepte les fichiers exécutables comme entrée et crée des fichiers .abs en sortie. En assemblant ce fichier on obtient une liste du code contenant les adresses absolues des instructions du code exécutable.

Cross-reference Lister : le générateur de liste de correspondance emploie les fichiers exécutables pour produire une liste de correspondance montrant des symboles, leurs définitions et leurs références dans les fichiers source liés. C'est donc un rapport de l'opération exécutée par le linker.

Utilitaire de conversion hexadécimal : son utilité réside dans la conversion de fichier exécutable de format COFF en format adapté aux EPROM Ti-Étiqueté, d'ASCII-hex, d'Intel, de Motorola-S, ou de Tektronix. L'EPROM en question fait l'objet d'une mémoire de stockage du code exécutable via laquelle le DSP est amorcé.

4.2.1.1 Outils de développement pour C/C++

Les outils de développement pour C/C++ sont [23]:

Compilateur C/C++ : le compilateur de C/C++ accepte le code source C/C++ et produit le code source en langage assembleur. Donc il s'agit de compiler et faire l'assemblage des fichiers ; si une optimisation est invoquée l'optimisateur de code C intervient. Ces fonctions d'optimisation, déjà évoquées dans le chapitre III, sont définies dans la boite de dialogue « Build Options ».

Optimisateur du code Assembleur : il permet de prendre en considération le code écrit en assembleur linéaire .sa (Linear assembler) c'est-à-dire qu'il est écrit sans tenir compte du parallélisme ou une optimisation d'utilisation des registre. Son rôle est d'optimiser cette structure en employant les registres appropriés, ainsi qu'un autre apport d'optimisation pour les boucles, et cela pour obtenir un code fortement parallélisé.

Utilitaire de construction de librairies : utilisé pour réaliser un archivage de routines fréquemment utilisées sous forme de librairies, cet utilitaire génère un fichier d'extension .lib. Des librairies sont disponibles sont fournies par le concepteur du logiciel CCS (TI), telle que la librairie rts6200.lib utilisée par le linker afin de construire le fichier exécutable adéquat à une cible de la famille C62x.

4.2.2 Graphe d'optimisation

En suivant le graphe de la figure 4.3, on peut atteindre les meilleures performances en matière de parallélisme d'exécution, et optimisation de la taille du code.[23]

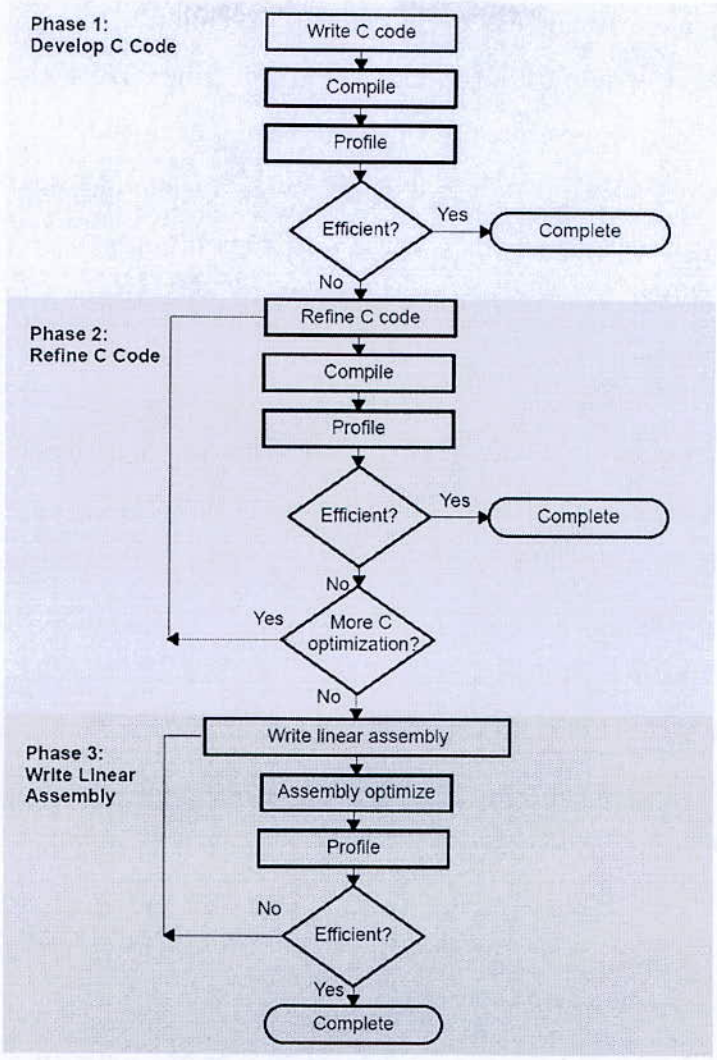


FIGURE 4.3 Graphe d'optimisation

La phase 1 : après écriture du code C et sa compilation dans le cadre d'un projet, l'utilisation de l'outil « Profiler » est indispensable pour voir l'efficacité du code C pour l'application. Cet outil, utilisé sous CCS après chargement du code exécutable en mémoire, permet de compter le nombre d'instructions chaque fonction sélection dans la boite de dialogue de l'outil en question, de cette façon on peut déterminer la partie du code C nécessitant une optimisation. Si le code s'avère convenable aux fins de l'application la phase 1 est suffisante.

La phase 2 : dans cette phase, on procède à une révision de la partie du code C nécessitant l'optimisation selon les résultats du « profiler », il s'agit donc de trouver un code C plus performant, et utiliser les instructions « intrinsics », dont nous avons parlé au Chapitre III, ainsi que les options de compilation. Cette partie apporte le mieux dans l'optimisation rapide du code exécutable à travers la programmation en C.

La phase 3 : dans certains cas où le temps d'exécution est un paramètre primordial dans le développement d'applications (application en temps réel) la partie du code C pénalisante doit être reconstruite en se basant sur l'assemblage linéaire.

Nous n'avons pas atteint cette phase dans notre application bien que certaines boucles demandent beaucoup de calcul, et cela à cause du temps d'une part et d'autre part ça non nécessité dans notre cas où nous travaillons en différé.

4.2.3 Outils de débugage

Les outils de débugage sont d'importance cruciale dans le développement d'applications, car ils permettent un gain en temps important ; en leur absence on est amené à réaliser des tests par partie. Cela peut être constaté lorsqu'on rencontre des problèmes liés à la programmation. Dans ce qui suit, nous allons citer en bref ces outils [23]:

Les points d'arrêt (Breakpoints) : les points d'arrêt sont considérés comme éléments essentiels dans n'importe quelle session de débugage. Ils ont pour rôle d'arrêter l'exécution du programme. Tandis que le programme est à l'arrêt, on peut vérifier l'état du programme, examiner ou modifier des variables, vérifier la pile d'appel,...etc. Les breakpoints peuvent être placés sur une ligne de code source dans l'éditeur de texte ou bien au niveau d'une instruction désassemblée sur la fenêtre de désassemblage.

Les fenêtre Watch windows : en débugant un programme, il est souvent utile de comprendre comment la valeur d'une variable change durant l'exécution du programme. La fenêtre « Watch window », figure 4.4, permet de surveiller les valeurs des variables locales et globales et des expressions de C/C++.

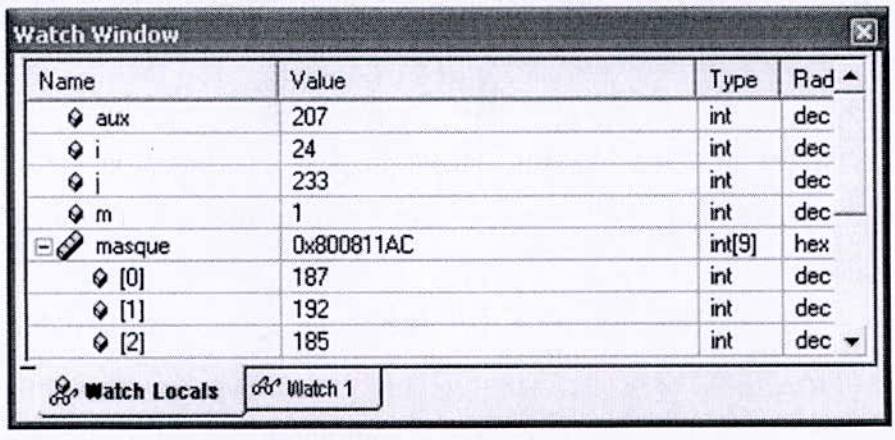


FIGURE 4.4 Watch windows

Points sondes (Probe Points) : les probe points sont utiles pour le développement d'algorithmes. On peut les employer afin : de transférer les données d'entrée à partir d'un fichier sur le PC hôte au buffer de la cible utilisée par l'application, et ceux de sortie à partir du buffer de la cible à un fichier dans le PC hôte pour d'éventuels analyses.

Ils sont différents des breakpoints par le fait qu'ils n'introduisent pas un arrêt permanant (lancement manuel après arrêt) de l'exécution du programme mais permette de relancer l'exécution de façon automatique. Entre temps on peut mettre à jours les paramètres du système ainsi que l'introduction des données. Donc pour tester une application, on peut introduire les probe points pour faire animer le travail sur des petits échantillons de l'entité à traiter.

Simulator Analysis : c'est un outil qui permet de surveiller et d'estimer les performances de contrôle de chaque évènement, on peut déterminer par exemple le nombre de cycles d'horloge écoulés dans l'exécution soit d'une fonction ou interruption ou bien encore ceux concernant la lecture et l'écriture.

General extention language (GEL) : le GEL est un langage descriptif, qui ressemble au C. Il permet de créer des fonctions pour augmenter l'accès aux ressources de l'environnement. Ces fonctions sont déjà définies, il suffit selon le besoin de faire appel à ces fonctions et d'introduire les paramètres nécessaires. De cette façon on construit des raccourcis vers les fonctionnalités de l'environnement.

Affichage des graphes : l'exécution d'un programme en utilisant seulement les points d'arrêt et les probe points ne donne pas une bonne interprétation des résultats, dans ce cas un outil d'affichage graphique est disponible et adapté au besoins du concepteur, figure 4.5.

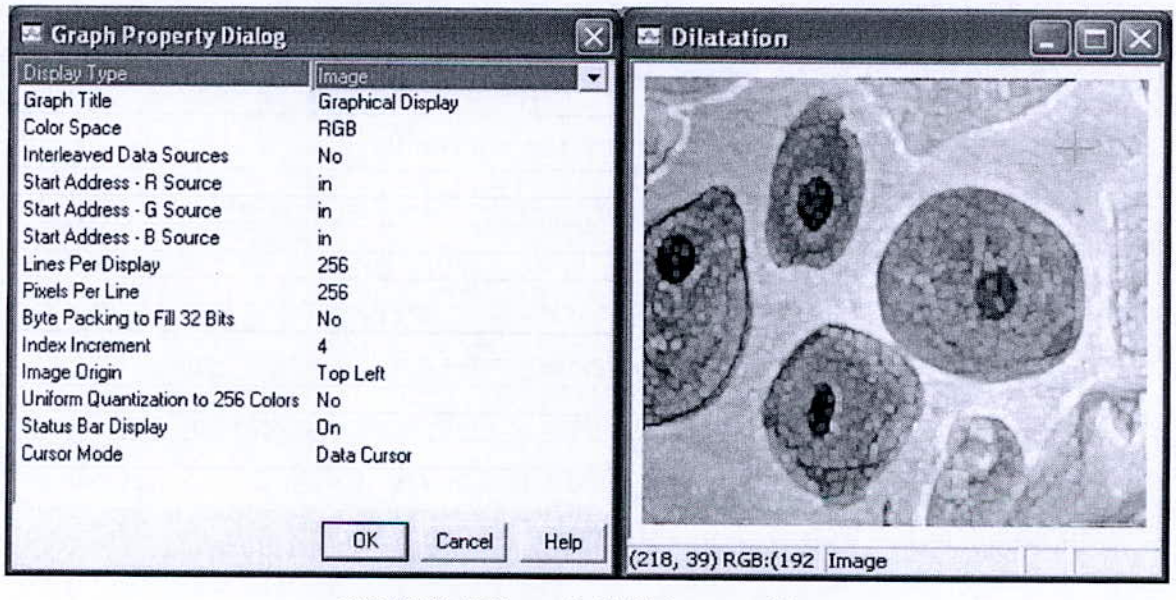


FIGURE 4.5 Exemple d'affichage graphique

4.2.4 Outils d'optimisation

L'optimisation du code que ce soit sa taille ou bien son temps d'exécution est une étape incontournable dans la programmation des DSP amenés à travailler en temps réel. Nous allons, dans cette partie, donner un aperçu sur deux outils : le profiler et le merveilleux PBC.

Le profiler : cet outil permet de réduire considérablement le temps d'identification des zones d'étranglement dans un code et de les éliminer, car il offre la possibilité d'analyser l'exécution d'un programme par la détermination des sections où le processeur prend beaucoup de temps. Comme exemple, on peut voir en combien de cycles une fonction peut être exécutée ainsi que le nombre d'appels de celle-ci, figure 4.6.

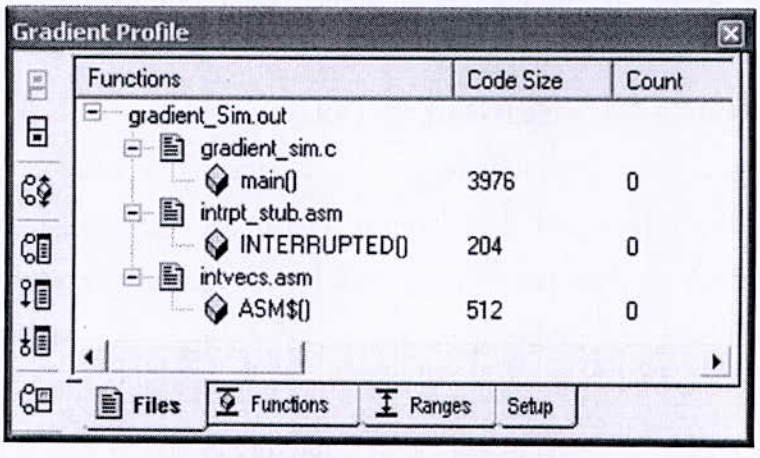


FIGURE 4.6 Fenêtre du profiler

Le PBC (Profile Based Compiler) : le PBC est un outil qui permet d'augmenter au mieux et facilement le compromis entre nombre de cycles d'horloge et la taille du code. En utilisant les options du PCB (par défaut, le PBC a cinq configurations différentes définies), on peut ainsi choisir le mode d'optimisation qui nous convient ; de cette façon le PCB collecte toutes les informations sur les fonctions et les données pour déterminer les bonnes combinaisons possibles des options de compilation. Le PCB va générer après un graphe montrant ces combinaisons, et il appartient à l'utilisateur de choisir celle la plus adéquate à l'application, figure 4.7.[23]

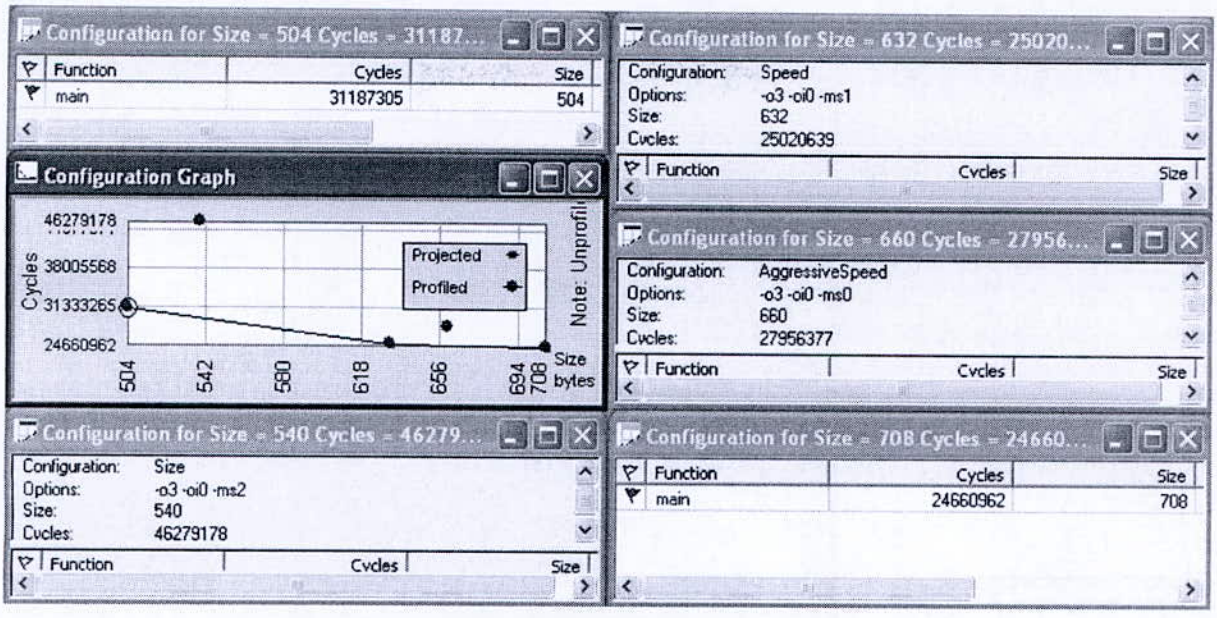


FIGURE 4.7 Fenêtre du profiler (PBC)

4.2.5 Notion de projet

Avant de commencer tout développement, il est nécessaire de créer un nouveau projet. Pour ce faire, la démarche à suivre est la suivante :

Il faut tout d'abord créer le fichier de gestion du projet (extension .pjt). Pour cela, on va dans le fichier Project et on clique sur new, la fenêtre suivante va apparaître (figure 4.8):

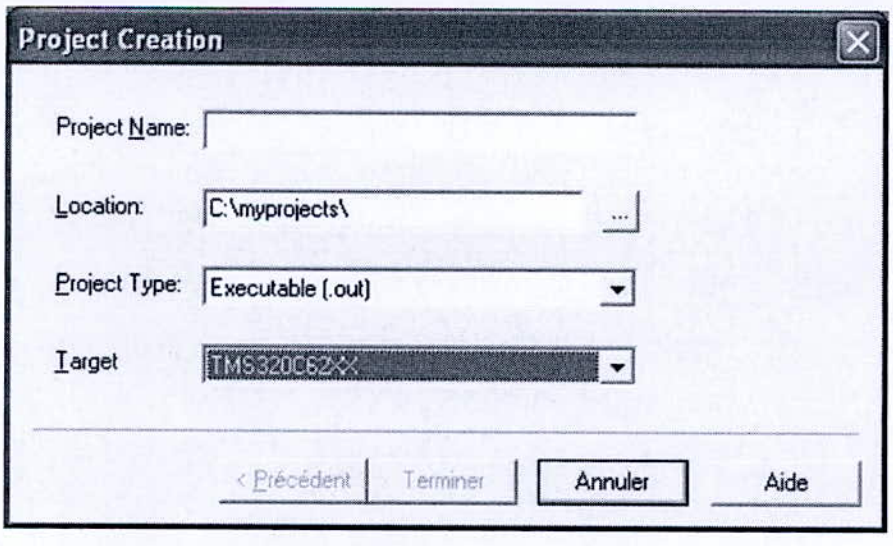


FIGURE 4.8 Création d'un projet

Une fois ce fichier créé, on peut ajouter des fichiers source à ce projet, comme par exemple des fichiers C, fichiers d'entête (math.h, stdio.h, etc.), le fichier donnant la tables des vecteurs d'interruption, et les librairies (.lib) ainsi que le fichier commande (.cmd), figure 4.9.

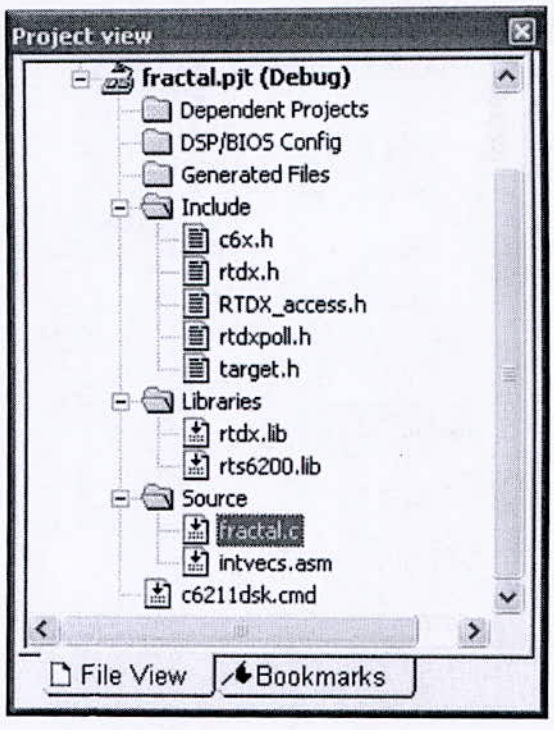


FIGURE 4.9 Projet créé

Le fichier .cmd est un fichier de commande dans lequel est définie la répartition en sections, dans la mémoire, des différents éléments d'une application, telles que les sections pour les variables locales et globales, et celle concernant le code.

Une fois que le fichier code source principal a été écrit, et que tous les fichiers annexes sont présents dans le projet, il faut " construire " le projet, c'est à dire générer un fichier exécutable compréhensible par le DSP. Pour cela, Il faut utiliser l'instruction " Rebuild all ", dans le menu " project ". Cette fonction réalise toutes les opérations nécessaires à la génération du fichier compréhensible par le DSP, c'est à dire sauvegarde des fichiers, assemblage des fonctions écrites en langage assembleur, élaboration des liens entre les différents fichiers, et compilation du source C principal. Une fois le fichier généré, s'il n'y a pas d'erreur, on obtient un message signalant l'absence d'erreurs. On peut alors lancer le programme sur le DSP, à l'aide de la commande " run " du menu " debug ". On peut stopper l'exécution du programme à l'aide de la fonction " halt ", et effacer le programme de la mémoire du DSP, à l'aide de la commande " reset DSP ".

4.3 CARTE DSK

La carte utilisée pour le développement est formée des composants suivants :

- DSP TMS320C6211 à 150MHz
- 16 M Bytes de mémoire externe SDRAM 100MHz
- 128 K Bytes de mémoire FLASH programmable et effaçable
- Port parallèle d'interfaçage (SPP) entre la carte DSK et le PC hôte
- Port d'interfaçage hôte permet l'accès à toutes les ressources mémoire du DSP à travers le port parallèle (moyen d'implémentation du code)
- Outil embarqué JTAG (Joint Test Action Group), accès en direct par le support XDS510 ou bien par émulation via le port parallèle
- 16-bit audio codec
- Quatre LED d'indication utilisées lors de l'amorçage
- Deux supports d'extension de mémoire ou périphérique

4.4 TECHNOLOGIE RTDX

Le Real Time Data eXchange (RTDX) permet aux développeurs de systèmes de transférer des données entre un ordinateur hôte et les périphériques de la cible. Il s'agit d'une communication bidirectionnelle qui est en mode half-duplex lorsqu'on utilise le port parallèle. Les données reçues de la cible peuvent être analysées et visualisées sur l'ordinateur hôte, ainsi qu'une possibilité d'ajuster les paramètres de l'application en utilisant les outils disponibles sous CCS.

Le RTDX se compose des deux composants de cible et du PC hôte. Une petite bibliothèque de RTDX fonctionne sur l'application de la cible. L'application de cible fait des appels de fonction à l'api (interface de programmation d'application) de cette bibliothèque afin de transférer des données vers ou à partir de la cible, par l'intermédiaire de l'interface JTAG (figure 4.10).

Sur la plateforme hôte, une bibliothèque de RTDX est disponible fonctionnant en conjonction avec CCS IDE. Pour visualiser les données on peut utiliser une interface graphique basée sur les API RTDX et programmée sous Visual C++ ou Visual Basic ou bien le LabView de National Instruments. Dans notre cas, nous avons utilisé le Linker for Code Composer Studio de MATLAB pour réaliser l'interface. [23]

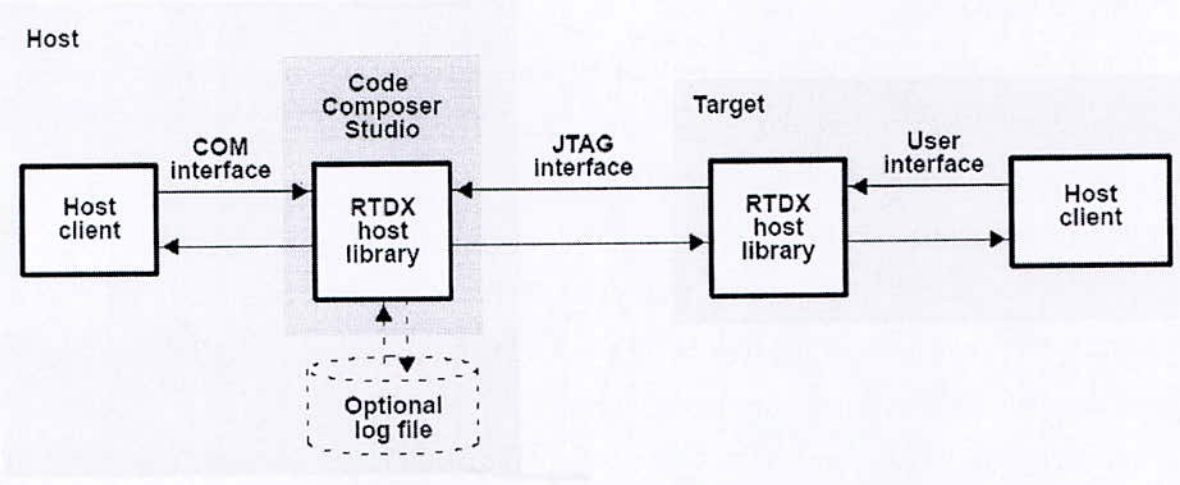


FIGURE 4.10 Connexion entre hôte et cible via JTAG

4.4.1 Communication Cible-PC

Dans la communication Cible-PC hôte, le canal de sortie doit être configuré. Les données sont écrites dans le canal de sortie moyennant les routines définies dans l'environnement de développement. Ces données sont immédiatement enregistrées dans un buffer au niveau de la cible définie dans une librairie (créée par compilation du projet). Les données présentes dans le buffer sont alors envoyées au PC hôte via l'interface JTAG (ou bien le port parallèle). La librairie RTDX générée pour le PC hôte reçoit ces données de l'interface JTAG et les enregistre dans un buffer mémoire ou un fichier (figure 4.11).[24]

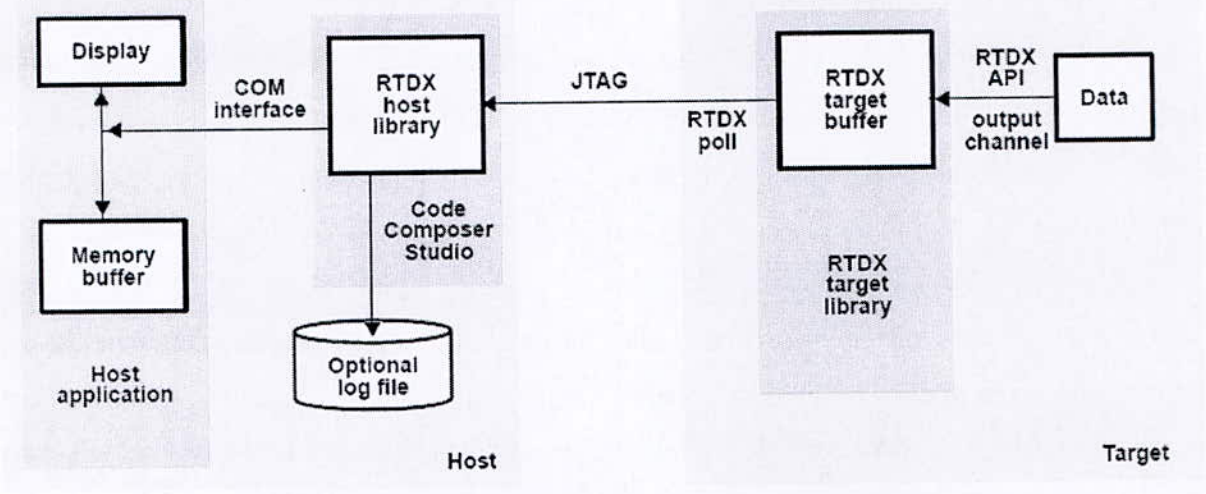


FIGURE 4.11 Etablissement de communication Cible-PC

4.4.2 Communication PC-Cible

Pour recevoir les données du côté cible à partir du PC hôte, on doit déclarer (configurer) un canal d'entrée pour la cible. La cible envoie une demande de réception de données en utilisant les routines définies lors de la programmation (voir l'annexe). Cette demande est enregistrée dans un buffer de la cible et envoyée par la suite au PC via l'interface JTAG (ou port parallèle). Une fois la demande reçue, le PC hôte écrit les données dans le buffer défini par la librairie du PC via l'interface COM (liaison logique entre l'application client et la librairie RTDX). Le PC ou mieux encore l'environnement de développement peut procéder au transfert des données via l'interface JTAG à l'adresse mémoire désirée au préalable. En fin une notification est générée par la cible signalant la fin de l'opération (figure 4.12).[24]

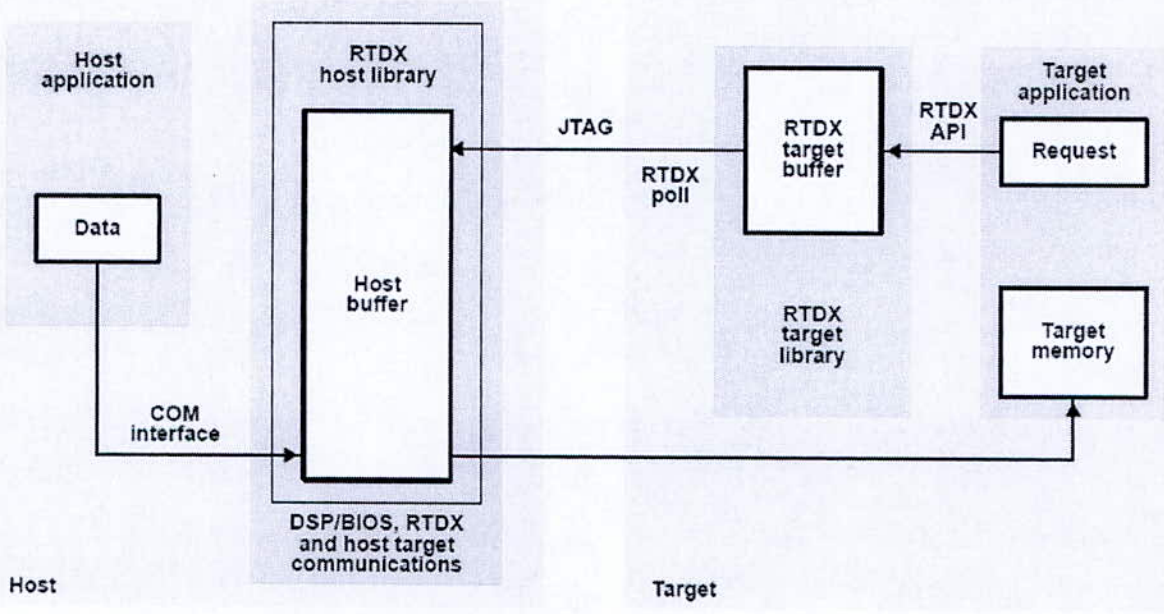


FIGURE 4.11 Etablissement de communication PC-Cible

4.4.3 Configuration de l'RTDX

Des outils sont disponibles sous CCS permettant la configuration des échanges de données en spécifiant plusieurs paramètres, ainsi que la possibilité d'effectuer un diagnostique de la communication RTDX de la carte DSK.

A partir du menu « Tools » du CCS, sous menu « RTDX », on peut voir les options suivantes :

Diagnostics Control
Configuration Control
Channel Viewer Control

La configuration des options de contrôle inclut le nombre de buffers utilisés par le code composer, la taille des buffers, et le mode de communication (continue ou non).

Le Channel Viewer permet de voir les canaux en cours d'utilisation, leur adresse et le nombre de messages envoyés. Pour plus d'information, se référer à l'aide du code composer.

4.5 LINK FOR CODE COMPOSER STUDIO DE MATLAB

Pour donner une bonne présentation des résultats, nous avons choisi le « MATLAB 7 SP2 » pour créer l'interface graphique au lieu de Visual C++ ou Visual Basic qui sont sans doute plus performants, cela est dû au temps de développement nécessaire qui n'était pas à notre portée. Cette interface graphique est considérée comme application client pour l'ensemble des entités formant le système de traitement d'images.

Pour ce faire, nous avons utilisé au sein du logiciel MATLAB une des ses fonctionnalités les plus récentes à savoir le « Link for Code Composer Studio ». Il s'agit d'un outil de développement qui permet d'utiliser des fonctions pour communiquer avec le code composer studio et avec les informations stockées dans la mémoire et les registre de la cible. Comme le RTDX fait partie du CCS, on peut aussi utiliser ce lien pour créer des canaux d'entrée et de sortie, et les activer ou les désactiver, et bien sûr établir des communications RTDX.

Pour établir le lien avec le CCS on utilise la commande ccsdsp permettant de créer un objet de lien :

```
cc = ccsdsp
```

la réponse de l'invite de commande sera :

```
CCSDSP Object:
  API version       : 1.0
  Processor type    : C67
  Processor name    : CPU
  Running?          : No
  Board number      : 0
  Processor number  : 0
  Default timeout   : 10.00 secs

  RTDX channels     : 0
```

Ces valeurs sont générées par défaut. Nous donnerons, dans le chapitre suivant, les valeurs adaptées à notre application, ainsi que d'autres fonctions utilisées dans ce cadre.

4.6 CONCULSION

Dans ce chapitre, nous avons présenté les outils de développement de notre application. L'environnement de développement intégré « Code Composer Studio » de TI qui sera un moyen de simulation et d'implémentation, et MATLAB pour réaliser une interface graphique (GUI) assistée par le « Link for Code Composer Studio » afin d'établir la communication (RTDX) avec la cible (DSK) et visualiser les images traitées.

Chapitre 5

Application

CHAPITRE V

APPLICATION

5.1 INTRODUCTION

Dans ce qui suit, nous allons introduire toutes les notions déjà vues dans les quatre chapitres pour effectuer des liaisons entre les différentes composantes remplissant les exigences de notre application.

Ce chapitre présente en détails les différentes étapes du développement, ainsi que les problèmes rencontrés avec les solutions apportées, et cela en se basant sur un exemple (un des algorithmes que nous avons implémenté).

5.2 PRISE EN CHARGE DE LA CARTE

Il est nécessaire, pour l'utilisation de la carte, de configurer le CCS pour la prendre en charge, et cela moyennant le « Setup », figure 5.1 :

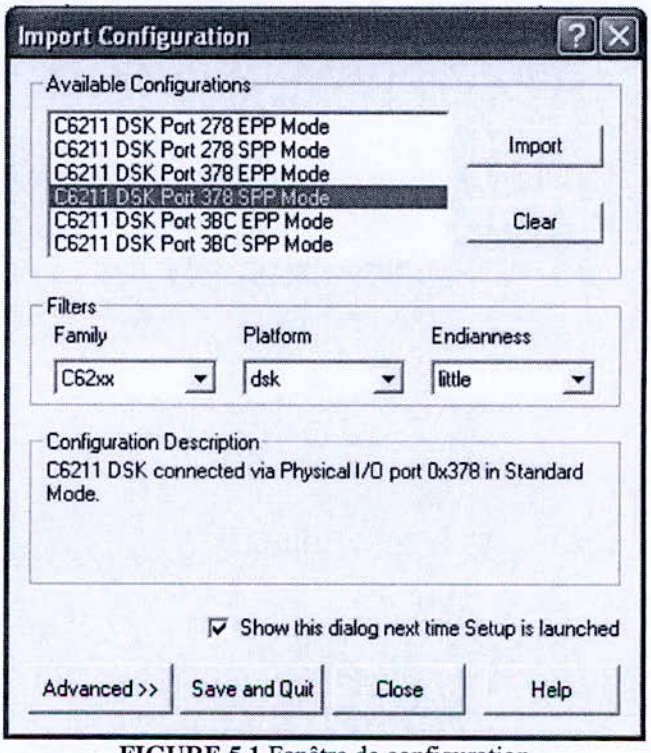


FIGURE 5.1 Fenêtre de configuration

La carte utilisée est la DSK TMS320C6211, connectée au PC via un port parallèle mode SPP, et par l'utilisation du mode mémoire « little endian ».

5.3 SOUS CODE COMPOSER

Considérons l'algorithme du détecteur de contours « gradient ». Nous procèdons, avant tout, à la création du projet que nous désignons par « gradient.pjt » tel que c'est présenté au chapitre IV. La cible choisie sera la famille C62xx. Le fichier générée à la fin est exécutable (*.out).

Après avoir créée le projet, nous ajoutons les fichiers suivants :

`gradient.c` : code source de l'application.

`rts6200.lib` : librairie standard pour prendre en charge les fonctionnalités du processeur.

`rtdx.lib` : librairie contenant des fonctions permettant d'établir la communication et la transmission des données.

`intvecs.asm` : table de définition des vecteurs d'interruption masquable et non masquable

`c6211dsk.cmd`: fichier de commande incluant la dimension de la pile (stack) et celle d'allocation dynamique (heap). Il définit aussi les différentes sections du code et des variables locales et globales.

Le fichier source est composé de trois catégories d'instructions : les instructions de déclaration des variables locales/globales et définition des fichiers d'entête, les instructions du traitement envisagé, et celles du contrôle des transferts des données.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <rtdx.h>
#include "target.h"

#define SZ 8192
#define SIZE 65536
#define LINE 256

short buf_in[SIZE],buf_out[SIZE];
short buf_out1[SZ],buf_out2[SZ],buf_out3[SZ],buf_out4[SZ],buf_out5[SZ];
short buf_out6[SZ],buf_out7[SZ],buf_out8[SZ];

RTDX_CreateInputChannel(ichan); //transfert de données PC-->DSK
RTDX_CreateOutputChannel(ochan);//transfert de données DSK-->PC
```

La section de programme précédente montre l'association de deux fichiers d'entête à savoir `rtdx.h` et `target.h`. Le fichier `rtdx.h` (fourni par TI) contient une table de définition des procédures utilisées par la librairie `rtdx.lib` et facilite ainsi l'accès aux fonctions de celle-ci. Le fichier `target.h` contient une fonction permettant d'initialiser le processeur qui s'appelle `TARGET_INITIALIZE()`.

```
/*****************************************************************************
* $RCSfile: target.h,v $
* $Revision: 1.1 $
* $Date: 2000/09/21 15:50:43 $
* Copyright (c) 2000 Texas Instruments Incorporated
*
* C6x specific initialization details
*****************************************************************************/
#ifndef __TARGET_H
#define __TARGET_H
#include <c6x.h>                /* IER,ISR,CSR registers                */

/*  RTDX is interrupt driven on the C6x.
    So enable the interrupts now, or it won't work.
*/

#define IER_NMIE    0x00000002
#define CSR_GIE     0x00000001
#define TARGET_INITIALIZE() \
    IER |= 0x00000001 | IER_NMIE; \
    CSR |= CSR_GIE;

#endif /* __TARGET_H    */
```

Cette initialisation concerne le registre de contrôle d'état CSR et le registre d'activation d'interruption IER. Il s'agit de positionner le bit 0 du registre CSR pour une activation globale d'interruption, et de positionner le bit 1 du registre IER pour activer l'interruption non masquable.

Nous considérons des images de taille standard (256×256), pour cela nous définissons le nombre de pixels égal à 65536 rangés dans un vecteur de cette taille. Et soit le nombre de pixel par ligne est pris égal à 256. La constante SZ représente la taille du buffer de sortie. Cette valeur de SZ est choisie après des tests de différentes tailles de radix 2 pour trouver la valeur maximale d'éléments que nous pouvons transmettre à MATLAB.

Pour acquérir le vecteur représentatif des intensités de toute l'image traitée, nous avons pris des segments égaux de taille SZ.

En résumé, un vecteur `buf_in[SIZE]` pour l'entrée, un vecteur `buf_out[SIZE]` pour la sortie pour une éventuelle visualisation sous CCS et les vecteurs `buf_outx[SZ]` pour les transferts DSK→Host.

Les deux lignes qui suivent la déclaration des constantes sont nécessaires pour créer les canaux d'entrée/sortie RTDX du côté processeur (ichan et ochan).

La fonction principale du programme est la suivante :

```
void main(void)
{

    int i;
    short outx,outy,s;

    TARGET_INITIALIZE(); //Initialisation des interruptions
    while(!RTDX_isInputEnabled(&ichan))//Il appartient à MATLAB
    puts("\n\n Waiting to read ");      //d'activer ichan; Attente
    RTDX_read(&ichan,buf_in,sizeof(buf_in));//Lecture par DSK
    puts("\n\n Read Completed");

    #pragma MUST_ITERATE(SIZE,SIZE)
    for (i=0;i<SIZE;i++)
    {
        if ((i+1)%(LINE)!=0) outx=_ssub(buf_in[i+1],buf_in[i]);
        else outx=buf_in[i];
        if (i<SIZE-LINE) outy=_ssub(buf_in[i+LINE],buf_in[i]);
        else outy=buf_in[i];
        s=_sadd(_abs(outx),_abs(outy));
        if (s>255) buf_out[i]=255; else buf_out[i]=s;
    }
```

Dans cette partie de programme, nous avons déclaré les variables locales pour effectuer des calculs intermédiaires. Cette déclaration est suivie d'une initialisation des interruptions telles qu'elles sont définies dans `target.h`. Le transfert de données du PC vers la DSK est contrôlé par l'interface créée sous MATLAB, la boucle while est utilisée pour mettre le processeur en attente jusqu'à l'activation du canal « ichan » par MATLAB afin de synchroniser les évènements entre la DSK et MATLAB. Après activation du canal d'entrée, le processeur peut procéder à la lecture.

La partie traitement d'images est une boucle dont le nombre d'itérations est utilisé dans une des directives pragma, abordées au chapitre III. Cette directive, destinée au compilateur, aide à optimiser le pipeline et le parallélisme lors de la compilation. D'autres optimisations résident dans l'utilisation des instructions « intrinsics » qui sont directement remplacées par leur équivalent en assembleur.

Une fois le traitement achevé sur le vecteur des intensités, nous procédons à la récupération de l'image traitée sous son format vectoriel. Voici la partie du programme qui effectue la tache :

```
#pragma MUST_ITERATE(SZ,SZ)
for (i=0;i<SZ;i++) buf_out1[i]=buf_out[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ;i<SZ*2;i++) buf_out2[i-SZ]=buf_out[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*2;i<SZ*3;i++) buf_out3[i-SZ*2]=buf_out[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*3;i<SZ*4;i++) buf_out4[i-SZ*3]=buf_out[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*4;i<SZ*5;i++) buf_out5[i-SZ*4]=buf_out[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*5;i<SZ*6;i++) buf_out6[i-SZ*5]=buf_out[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*6;i<SZ*7;i++) buf_out7[i-SZ*6]=buf_out[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*7;i<SZ*8;i++) buf_out8[i-SZ*7]=buf_out[i];

while(!RTDX_isOutputEnabled(&ochan)) //Attente d'activation de ochan
puts("\n\n Waiting to write 1"); // Attente
RTDX_write(&ochan,buf_out1,sizeof(buf_out1));//Envoi de données
puts("\n\n Write 1 Completed");
while(RTDX_isOutputEnabled(&ochan)) // attente de la fermeture
puts("\n\n Waiting to write 2"); //En attente
```

Tout en utilisant les directives pragma, on forme huit vecteurs de taille SZ à partir du vecteur de sortie, cela à cause de la non possibilité d'effectuer des transferts de vecteur à MATLAB dont la taille est supérieure à SZ selon les testes que nous avons effectués, mais ce point reste à vérifier en faisant varier les différents paramètres sous MATLAB.

De la même façon que la précédemment, nous transférons les données en contrôlant l'activation et la désactivation du canal de sortie.

Remarque : pour des raisons d'optimisation, nous choisissons l'option –o3 du compilateur. De plus, nous choisissons le modèle mémoire Far Calls & Data pour remplacer les instructions de transferts sur 16bits par celles travaillant sur des adresses de 24bits.

5.3 SOUS MATLAB

Nous allons maintenant expliquer la fonction, responsable de la communication avec le code composer, utilisée par l'interface graphique.

La fonction a pour entrées :

`x` : le chemin avec le fichier projet de l'algorithme sélectionné

`y` : le chemin avec le fichier *.out à implémenter

`ifile` : le nom de fichier image importé du disque dure.

```
function proim=CCS(x,y,ifile)
indata=imread(ifile);
indata=indata';
indata=indata(1:65536);
ccsboardinfo % info de la cible
cc = ccsdsp('boardnum',0); %enregistrement d'objet ccsdsp
reset(cc) %reset de la carte
configure(cc.rtdx,65536,8); %configuration de RTDX
enable(cc.rtdx); %activation de RTDX
if ~isenabled(cc.rtdx)
error('RTDX non avtivé')
end
cc.rtdx.set('timeout', 300); %le temps d'arrêt
open(cc,x); %chargement du projet; peut être éliminé
load(cc,y); %chargement du fichier exécutable
```

Les trois premières lignes visent à mettre en forme les données, extraites de l'image à traiter, sous forme de vecteur. Les données, sur les intensités d'image, pouvant être ainsi véhiculées à travers les liaisons logiques et physiques.

Nous introduisons maintenant les instructions du Link for Code Compose Studio de MATLAB. L'instruction ccsboardinfo donne des informations sur la cible installée pour d'éventuelles corrections du script (configuration). L'instruction qui suit sert à créer et enregistrer l'objet qui établit le lien avec CCS. Une fois le lien établi, comme toute manipulation, nous faisons appel au reset.

Pour acquérir le vecteur sur la carte il faut spécifier la taille du buffer utilisé par le PC. La configuration étant exécutée, nous activons maintenant la fonction RTDX sous CCS. En fixant le timeout de MATLAB, nous pouvons ouvrir le projet sous CCS et charger l'exécutable dans la carte.

```
run(cc); %run sous CCS
open(cc.rtdx,'ichan','w'); %ouverture de ichan sous MATLAB
open(cc.rtdx,'ochan','r'); %ouverture de ochan sous MATLAB
pause(1)
enable(cc.rtdx,'ichan'); %activation du canal ichan sous CCS
if isenabled(cc.rtdx,'ichan')
writemsg(cc.rtdx,'ichan', int16(indata)) %envoi de données 16bits
pause(1)
else
error('Cannal ''ichan'' non activé')
end
```

L'instruction run(cc) permet d'exécuter le programme présent sur la cible. Nous activons le canal d'entrée pour envoyer les données et sortir de la boucle d'attente programmée sous CCS, mais entre temps nous devons déclarer les canaux d'entrée et de sortie sous MATLAB.

Une fois les données transférées, le processeur effectue le traitement et se met à l'attente pour une activation du port de sortie à partir de MATLAB, et procéder à l'envoi des résultats à MATLAB en utilisant la section suivante :

```
enable(cc.rtdx,'ochan'); %Activation du canal de sortie
if isenabled(cc.rtdx,'ochan')
outdata1=readmsg(cc.rtdx,'ochan','int16') %Lecture des résultats
pause(0.1)
else
error('Canal ''ochan'' non activé')
end
```

Les huit vecteurs représentant l'image sont ensuite rassemblés et convertis en matrice de niveau de gris pour être affichée. La figure 5.1 représente l'interface graphique « DSPGUI » créée sous MATLAB.

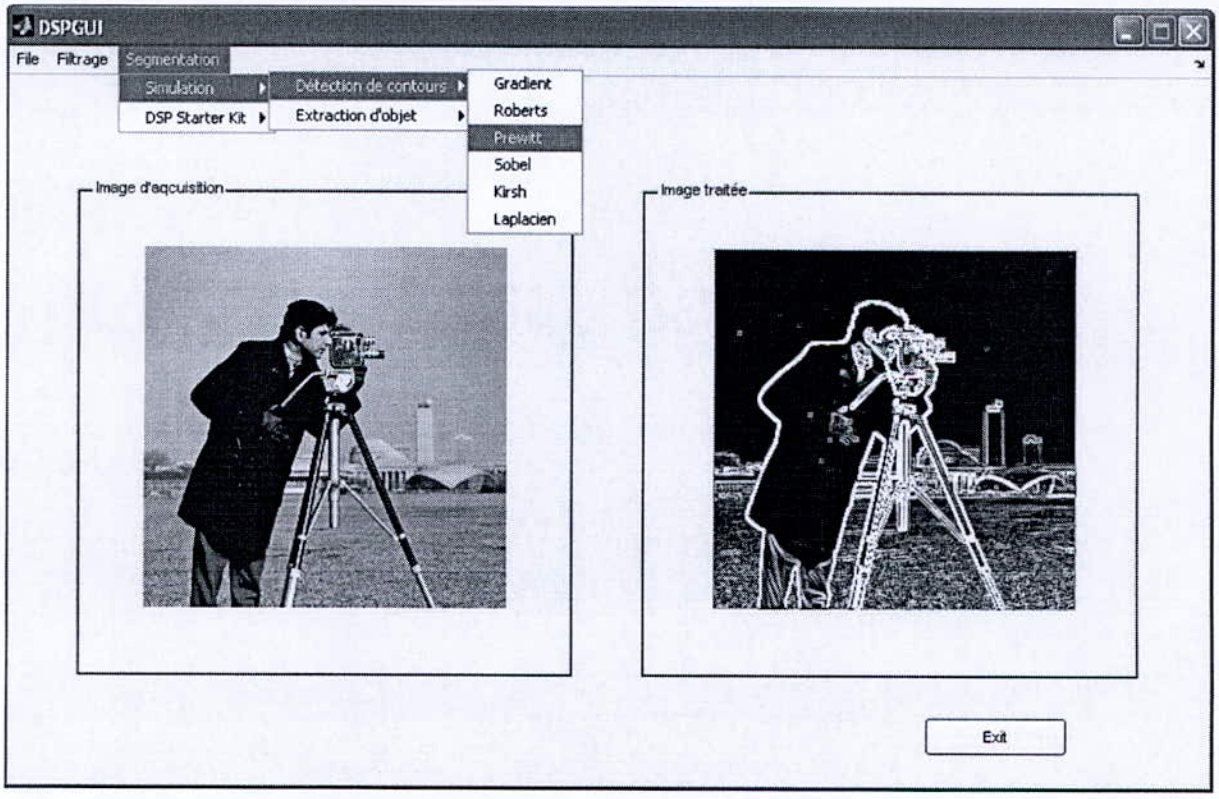


FIGURE 5.1 L'interface graphique DSPGUI

Pour ouvrir une image, il faut sélectionner le menu « File » puis « Open » (figure 5.2).

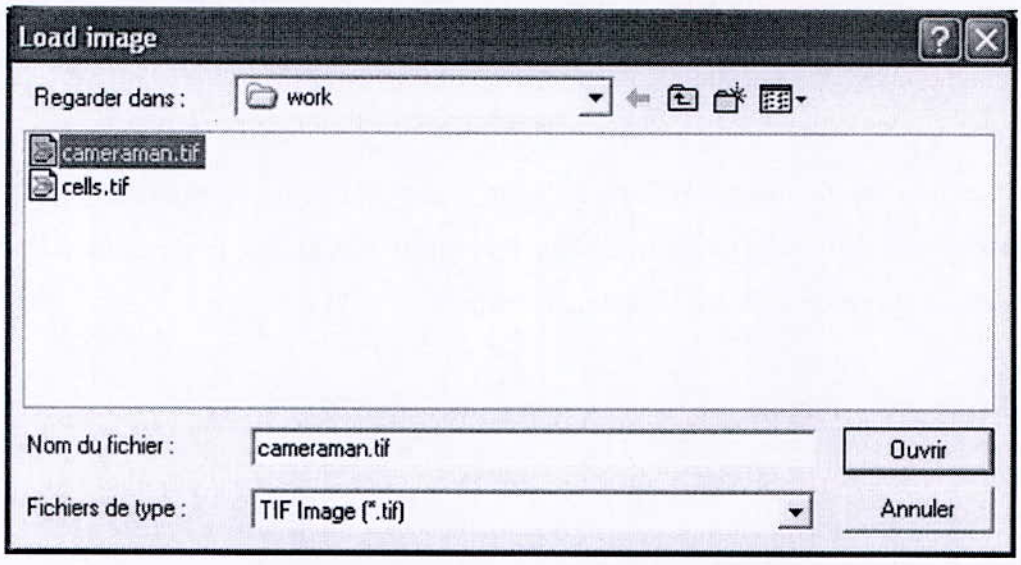


FIGURE 5.2 Fenêtre de chargement d'image

Et pour sauvegarder les résultats, nous utilisons « Save » dans le menu « File ».

5.4 RESULTATS

5.4.1 Filtrage

La figure 5.3 représente l'image originale perturbée par un bruit impulsionnel. Nous appliquons à cette image les algorithmes de filtrage étudiés, et nous obtenons :

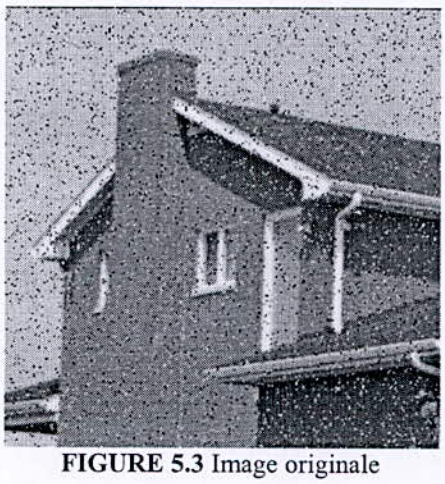


FIGURE 5.3 Image originale

FIGURE 5.4 Image traitée, filtre moyenneur

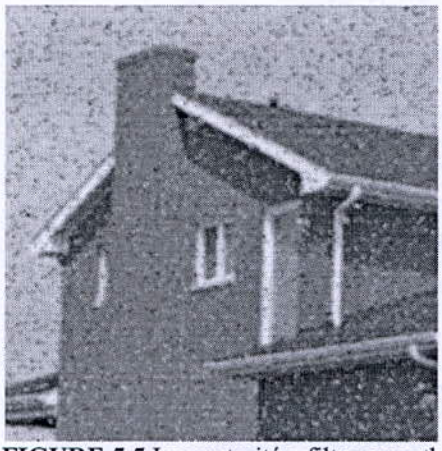


FIGURE 5.5 Image traitée, filtre smooth

FIGURE 5.6 Image traitée, filtre médian

FIGURE 5.7 Image traitée, filtre d'ordre adaptatif

FIGURE 5.8 Image originale

FIGURE 5.9 Image traitée, filtre morphologique dilatation

FIGURE 5.10 Image traitée, filtre morphologique Fermeture

5.4.2 Détecteurs de contours

La figure 5.11 représente l'image originale qui va subir différents traitements en vu d'une détection de contours :

FIGURE 5.11 Image originale

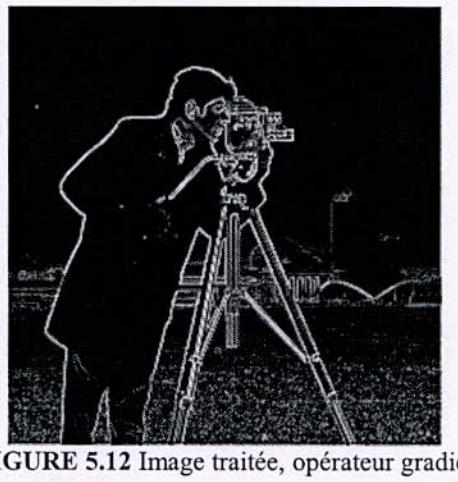


FIGURE 5.12 Image traitée, opérateur gradient

FIGURE 5.13 Image traitée, opérateur Roberts

FIGURE 5.14 Image traitée, opérateur Prewitt

FIGURE 5.15 Image traitée, opérateur Sobel

FIGURE 5.16 Image traitée, opérateur Kirsh

FIGURE 5.17 Image traitée, résultats de l'approche multifractale

5.5 CONCLUSION

La communication avec la carte DSK via le port parallèle, n'est pas performante (très lente) chose qui pose une perte de message au niveau de la liaison CCS----MATLAB, par ailleurs une visualisation sous CCS est possible. Nous avons, pour résoudre ce problème, procédé à un envoi de données élément par élément ; et cela en créant une boucle de contrôle sous CCS et une autre sous MATLAB ; au lieu d'utiliser les vecteurs (la liaison logique en question reste toujours lente mais sans perte de message).

On peut remédier à ces problèmes en utilisant le câble d'émulation XDS510 (ou XDS560) utilisant la connexion JTAG avec la carte et la connexion PCI avec le PC.

CONCLUSION GENERALE

Nous avons, dans ce projet, commencé par étudier les différents algorithmes de filtrage et de détection de contours utilisés dans le traitement des images avant tout traitement avancé. Le filtrage, quelque soit son effet, apporte une amélioration sur la qualité de l'image pour rendre l'étape de segmentation aisée donnant des résultats plus significatifs. Nous nous sommes intéressés beaucoup plus, dans l'étude des filtres, aux filtres adaptatifs qui utilisent les informations sur les statistiques d'ordre un et deux et la symétrie de ces statistiques. Dans l'étude de la segmentation, nous avons fait appel au formalisme multifractal qui peut être adopté par les deux approches de la segmentation, nous avons constaté que les valeurs supérieures du coefficient de singularité correspondent aux contours alors que les petites valeurs correspondent aux régions homogènes.

Notre but était d'implémenter ces algorithmes sur un DSP de la famille C6000 de TI, à savoir la cible C6211 mise en œuvre sur une carte DSK. Nous avons pu atteindre ce but en utilisant la liaison logique RTDX pour communiquer avec le processeur, les images traitées sont affichées sous MATLAB à l'aide du Link for Code Composer Studio. Nous avons aussi réalisé une librairie de routines de traitement d'images pouvant être utilisée sous Code Composer Studio.

Nous avons voulu aller au delà des buts de ce projet en réalisant une application en temps réel, mais la non disponibilité des ressources matériels tels que le câble d'émulation XDS560 et la carte d'extension IDK (Imaging Developer's Kit) nous a empêché de le faire. En considérant cela dans l'une de nos perspectives, nous envisageons dans un avenir proche une bonne progression dans le domaine applicatif du traitement d'images en général du laboratoire signal et communications.

Grâce à ce projet, nous avons pu avoir des connaissances précieuses dans une branche de l'électronique que nous n'avons pas étudiée auparavaant, ainsi qu'un bon contact avec la pratique.

PERSPECTIVES

- Développer des applications temps réel en utilisant l'IDK et XDS560 et introduction de la solution DSP/BIOS.
- Réalisation d'un host client en utilisant le Visual C++ de Microsoft et les API de Texas Instruments concernant l'RTDX et les périphériques, au lieu de MATLAB.

BIBLIOGRAPHIE

[1] JP.COCQEREZ et S.PHILIPP, Analyse d'images : filtrage et segmentation, Edition MASSON 1995.

[2] C.KADDOUR, Généralités sur le traitement d'images, http://iquebec.ifrance/kadchakib/chap1/chap1.htm.

[3] Gilson BRAVIANO, Logique floue en segmentation d'images : seuillage par entropie et structures pyramidales irrégulières, thèse doctorat UJF Grenoble 1.

[4] Y.BOULFANI, S.DOUMANDJI, Implémentation sur DSP C5000 de filtres optimaux appliqués aux images et introduction de réseaux neuronaux, PFE 2004 ENP.

[5] C.CHAOUCHI, Etude et implémentation de détecteurs de contours, PFE 2004 ENP.

[6] Bibliothèque d'exemples de traitement des images, ENST BETI, http://www.tsi.enst.fr/~MAITRE/BETI/index.html.

[7] Y.BRIKI, D.SMATI, Contribution à la conception d'un outil d'aide pour le choix d'une technique de traitement d'images, PFE 2004 INI.

[8] P.ZAMPERONI, Variation on the rank-order filtering theme for grey-tone and binary image enhancement, International Conference on Acoustic Speech and Signal Processing, May 1989, vol 3, pages 1401-1404.

[9] INRIA, http://www.inria.fr/rapportsactivite/RA2001/fractales/domai_image_ct.html.

[10] N.LASSOUAOUI, Segmentation des images par différentes approches est optimisation avec les algorithmes génétiques, thèse de doctorat en électronique 2004.

[11] N.LASSOUAOUI, L.HAMAMI, Genetic algorithms and multifractal segmentation of cervical cell images, Proceedings of seventh international symposium on Signal Processing and Its Application, July 2003, vol 2, pages 1-4.

[12] I.Reljin, B.Reljin, I.Pavlovic, I.Rakocevic, Multifractal analysis of gray-scale images, 10th Mediterranean Electrotechnical Conference, Sept. 2000, Vol 2, pages 490-493.

[13] J.GRAZZINI, Analyse multiéchelle et multifractale d'images météorologiques : application à la détection de zones de précipitation, thèse de doctorat MARNE-LA-VALLEE.

[14] R.CHASSAING, DSP Application Using C and the TMS320C6x DSK, Edition Wiley 2002.

[15] S.AHMED ZAID, Les DSP : étude du TMS320C31, Mini projet dirigé par Mr R.SADOUN, Séminaire ENP 2004.

[16] TMS320C6000 CPU and Instruction Set, SPRU189f.pdf, Texas Instruments.

[17] TMS320C6000 Peripherals, SPRU190d.pdf, Texas Instruments.

[18] TMS320C6000 Programmer's Guide, SPRU198d.pdf, Texas Instruments.

[19] TMS320C6000 Optimizing Compiler User's Guide, SPRU187g.pdf, Texas Instruments.

[20] TMS320C6211 Cache Analysis, SPRA472.pdf, Texas Instruments.

[21] TMS320C6000 Technical Brief, SPRU197d.pdf, Texas Instruments.

[22] Code Composer Studio Help, Texas Instruments.

[23] Code Composer Studio Getting Started Guide, SPRU509c.pdf, Texas Intruments.

[24] DSP/BIOS RTDX and Host-Target Communications, SPRA895.pdf, Texas Instruments.

مختصر

يكمن هذا العمل في تنفيذ الرّوتين المتعلقة بمعالجة الصّور على دي إس بي C6211 من فئة C6000 للمعالجات الرقمية للإشارة. قواعد معالجة الصّور المختارة تتمثل في الفلاتر التقليديّة و التكيّفيّة، بالإضافة إلى قواعد التقسيم باعتبار الحواف والمناطق. لتنفيذ ذلك، دي إس كيي TMS320C6211، البرنامج Code Composer Studio و MATLAB يكوّنون أدوات التّطوير التي تتيح تمثيل الأجزاء "الإكتساب، المعالجة والتمثيل". الرابطة RTDX مستعملة لإنشاء اتّصال بين CCS وDSK. Link for Code Composer Studio للبرنامج MATLAB مستعمل من طرف واجهة المستخدم لإنشاء اتّصال بين CCS وMATLAB.

الكلمات المفتاحية: خوارزميات معالجة الصّور، دي إس بي، Code Composer Studio، RTDX، Link for Code Composer Studio

Abstract

This work consists of an implementation of routines of image processing on the DSP C6211 of C6000 family. The image processing algorithms selected are those of conventional and adaptive filters, and the algorithms of segmentation by edge and area approaches. To be done, DSK TMS320C6211, the software Code Composer Studio and MATLAB constitute the development tools allowing the representation of the parts "acquisition, processing and displaying". RTDX is a logic link used to establish a communication between the CCS and DSK. Link for Code Composer Studio of MATLAB is used by the graphic user interface (GUI) to establish a link between MATLAB and CCS.

Key words: image processing algorithms, DSP, Code Composer Studio, RTDX, Link for Code Composer Studio.

Résumé

Ce travail consiste en une implémentation de routines de traitement d'images sur le DSP C6211 de la famille C6000. Les algorithmes de traitements d'images choisis sont les algorithmes de filtrage classiques et adaptatifs, et les algorithmes de segmentation par approche contour et région. Pour ce faire, la carte DSK TMS320C6211, les logiciels Code Composer Studio et MATLAB constituent les outils de développement permettant la représentation des parties « acquisition, traitement et affichage ». La liaison RTDX est utilisée pour établir une communication entre le CCS et la carte DSK. Le Link for Code Composer Studio de MATLAB est utilisé par l'interface graphique (GUI) pour établir la liaison logique entre MATLAB et CCS.

Mots clés : algorithmes de traitement d'images, DSP, Code Composer Studio, RTDX, Link for Code Composer Studio.