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

DEPARTEMENT D'ELECTRONIQUE

Mémoire 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. Application à la séparation des Chromosomes

Proposé et dirigé par :

Dr. L.HAMAMI (MC)

Réalisé par:

Melle ROULA Nadjah
Melle TIRES Sabrina

- Promotion : Juin 2006 -

ملخص

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، قواعد التقسيم باعتبار الحواف و المناطق و خورزميات الهيكلة مع تنفيد فصل صور الكروموزومات من نوع FISH كتطبيق لذلك . لتنفيذ ما سبق، دي إس كيي TMS320C6211، البرنامج Code Composer Studio و MATLAB يكوّنان 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 ال script لإنشاء اتّصال بين CCS وMATLAB.

الكلمات الرّئيسيّة:الصبغيات، FISH ، معالجة الصّور، دي إس بي، 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, les algorithmes de segmentation par approche contour et région et les algorithmes de squelettisation, avec comme application l'implémentation d'un algorithme de séparation des chromosomes humains sur des images de métaphase FISH. 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 le script pour établir la liaison logique entre MATLAB et CCS.

Mots clés : chromosome, FISH, traitement d'images, séparation des chromosomes, DSP, 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 filters, the algorithms of segmentation by edge and area approaches, and the skeleton algorithms. As application, an implementation of separation algorithm of human chromosomes on metaphase FISH images. 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 script to establish a link between MATLAB and CCS.

Key words: chromosome, FISH, image processing, DSP, separation of chromosomes, DSP, Code Composer Studio, RTDX, Link for Code Composer Studio.

REMERCIEMENTS

Nos profonds remerciements vont à notre promotrice Dr L. HAMAMI qui nous a donné l'occasion de travailler sur un sujet passionnant, et pour sa confiance, ses conseils judicieux et sa collaboration.

Nous remercions Mr. D.BERKANI, professeur à l'école nationale polytechnique, de nous avoir fait l'honneur de présider le jury et pour ses encouragements de valeur.

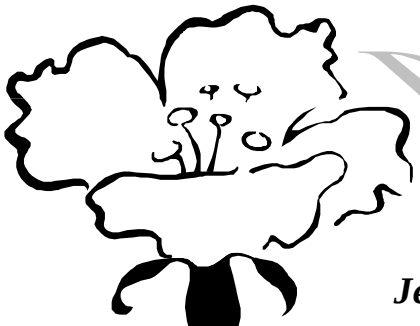
Nos remerciements vont aussi à Mr. B.BOUSSEKSOU pour avoir accepter d'examiner ce modeste travail.

Notre profonde gratitude à toutes les personnes ayant contribué à notre formation.

Nous remercions Mr Abdelkrim RAHMANIA et Salim AHMED ZAID pour leur aide précieuse.

Nous tenons à remercier Melle. Radia HAROUNE et Mme KASRI.

Dédicace



Je dédie ce travail modeste :

A ma très chère mère :
YAMINA
Symbole grandiose de patience et de beaucoup d'amour

A mon très cher père :
AHMED
Pour sa compréhension et pour ce qu'il a enduré pour m'offrir le bonheur et la confiance.

A mes très chères frères :
SOFIANE, TOUFIK et OKACHA

A ma binôme :
NADJAH, ***sans qui ce travail n'aurait jamais vu jour.***

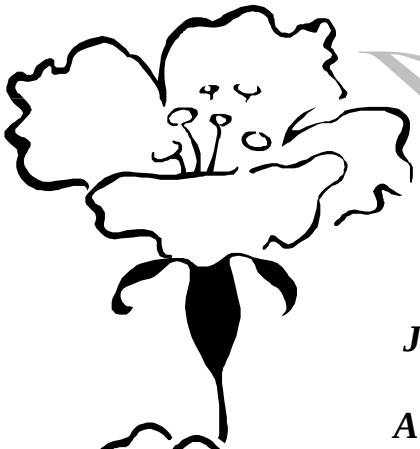
A mes amis :
RAHIM, YACINE, AHMED, AMINE, RABIE, RIAD, NABIL, BOUALEM…..

A mes amies :
NARIMENE, KAMELIA, IMANE…..

A tous mes camarades de **l'ENP**

A la mémoire de ma tante et de mes oncles.

Dédicace



Je dédie ce modeste travail à :

A ma très chère mère :
LATIFA

A mon très cher père :
MOKHTAR

A mon très cher frère :
AMMAR

A mon cousin RIAD.

A mon adorable POISSON, à qui je dois beaucoup.

A tous mes CAMARADES et amis :
HAMID, SAADEDINE, MAHIOUT, MOHAND ARAB et KARIMA.

A ma binôme :

SABRINA, sans qui ce travail n'aurait jamais vu le jour.

A tous mes camarades de l'ENP et ceux de l'université de JIJEL.

A la mémoire de AZIZ BELGACEM

TABLE DES MATIERES

CHAPITRE III

Etude de la famille des DSP C6000 et outils de développement

CHAPITRE IV

Application et Présentation des résultats

LISTE DES ILLUSTRATIONS

LISTE DES TABLEAUX

INTRODUCTION GENERALE

Le traitement d'images est une science basée sur plusieurs disciplines : les mathématiques, les probabilités, l'informatique et les architectures avancées. Les algorithmes de traitement d'images impliquent des volumes de calcul assez importants. L'essor de l'implémentation de ces algorithmes est fortement lié à l'avènement sur le marché des processeurs de traitement du signal (DSP, acronyme de Digital Signal Processor).

La réalisation d'un système automatique de séparation des chromosomes s'inscrit dans la continuité des travaux, de l'équipe de traitement d'images au laboratoire signal et communications de l'école nationale polytechnique, qui ont pour objectif la création d'un système automatique de classification des chromosomes et la détection des éventuelles anomalies chromosomiques.

Ce travail s'inscrit dans le cadre des travaux de recherche dirigés par Dr. L. HAMAMI. Il a pour but de tester l'implémentabilité d'un algorithme de séparation des chromosomes sur le processeur C6211 de la famille C62x de Texas Instruments en utilisant plusieurs approches de transfert des données. Et cela en vue de réaliser un système de traitement d'images autonome ou embarqué.

Nous tentons d'expliquer le travail réalisé à travers quatre chapitres :

Dans le premier chapitre, nous donnons quelques notions générales de cytogénétique ainsi que des techniques de préparation des images.

Le deuxième chapitre est consacré aux concepts de base en traitement d'images, il exposera les méthodes de traitement susceptibles d'être utilisées dans notre application.

Dans le troisième chapitre, une étude de la famille C6000, en particulier le TMS320C6211, a été faite ; dans lequel nous exposons également les outils de développement logiciels et matériels nécessaires à la création de l'application.

Le dernier chapitre est consacré à l'implémentation de l'algorithme de séparation des chromosomes ainsi qu'à la présentation des résultats et aux remarques générales relevées.

Nous terminons bien évidemment par une succincte conclusion.

Chapitre I

Introduction à la cytogénitique & présentation de la technique FISH

Dans ce chapitre, nous donnerons un rappel sur l'environnement chromosomique, puis nous passerons aux principales techniques utilisées pour la préparation des images notamment la technique FISH utilisée pour la préparation de notre base de données.

1.1. INTRODUCTION

La cytogénétique humaine (ou génétique chromosomique) est une discipline récente dont l'essor date de 1956 avec la détermination du nombre exact de chromosomes chez l'homme. En 1970, l'introduction des techniques de marquage en bandes des chromosomes a grandement amélioré la résolution et la sensibilité de l'analyse cytogénétique.

Enfin, l'apparition de l' hybridation in situ fluorescente en 1986 et son développement rapide offre aujourd'hui toute une panoplie d'outils permettant une étude de plus en plus fine et précise des chromosomes et de leur structure. Alors que les techniques classiques pour la réalisation d'un caryotype offrent une vue d'ensemble du génome mais avec une faible résolution (une sous bande chromosomique correspond au mieux à environ 2 Mb sur un caryotype en haute résolution), l'hybridation in situ apporte une résolution de l'ordre de quelques Kb (ou même moins si l'hybridation est réalisée sur une fibre d' ADN étiré), ce qui comble le fossé analytique entre l'étude cytogénétique et moléculaire.

1.2. ETUDE CHROMOSOMIQUE

1.2.1. Le chromosome

Le chromosome est une structure microscopique représentant le support physique des gènes. Il est constitué essentiellement d'ADN et de protéines, le tout constituant la chromatine.

On a observé les chromosomes pour la première fois à la fin du XIXème siècle. Ils ont la forme d'un bâtonnet chez la plupart des espèces animales et végétales. L'ADN des chromosomes constitue le matériel héréditaire de la cellule et est transmis de génération en génération de cellule, il est donc le support de l'information génétique. Plus l'espèce est

évoluée, plus son ADN est compacté et empaqueté par des protéines structurales. Les chromosomes se trouvent dans les cellules de tous les êtres vivants, en nombre variable et spécifique à chaque espèce. **[1]**

L'espèce humaine en compte 46 dont :

- Les chromosomes autosomiques ou somatiques, qui sont au nombre de 22 paires
- Les chromosomes sexuels qui sont au nombre d'une paire (XX chez la femme et, XY chez l'homme)

1.2.2 La structure du chromosome

Le chromosome est constitué de deux bâtonnets parallèles (figure 1.1), appelés **chromatides**, reliés entre eux par le **centromère** au niveau d'une constriction. La position du centromère sur un chromosome étant fixe, on considère qu'une séquence nucléotidique précise définit le centromère. Les centromères permettent les migrations chromosomiques au cours de la mitose. Ils assurent donc le bon déroulement de la mitose et une répartition en lots égaux des chromosomes à chaque pôle de la cellule.

L'extrémité de chaque chromatide porte le nom de **télomère**, c'est une structure spécialisée qui confère au chromosome sa stabilité. En effet, lorsqu'un télomère est perdu, l'extrémité du chromosome devient très instable, et tend à fusionner avec les extrémités d'autres chromosomes cassés, à subir des recombinaisons, ou tout simplement à être dégradée.**[3] [4]**

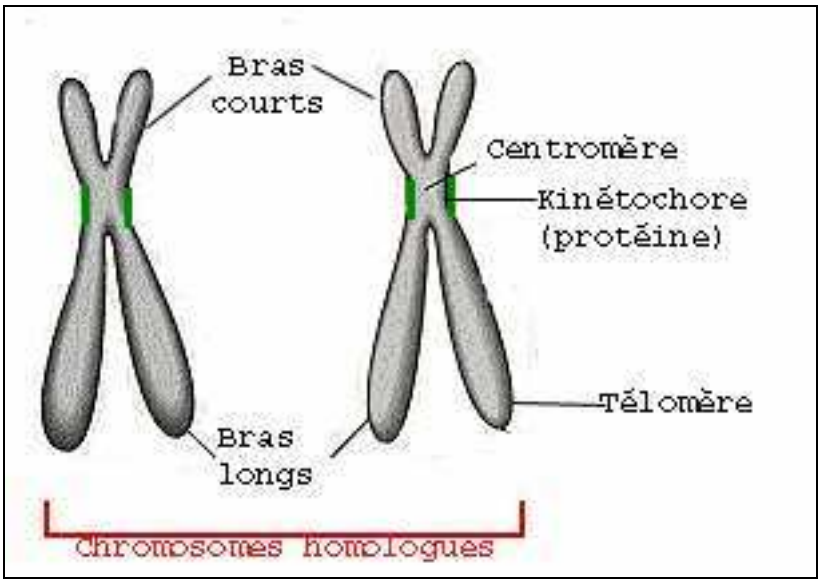


Figure 1.1 : Structure du chromosome

1.3. LA METAPHASE

Le cycle cellulaire est divisé en plusieurs phases :

L'interphase :

- la phase G1, première phase de croissance,
- la phase S durant laquelle le matériel génétique est répliqué,
- la phase G2, qui est la seconde phase de croissance cellulaire et, finalement,

La mitose :

- La phase M, cette phase est elle-même constituée de plusieurs étapes dont la métaphase (figure 1.2) deuxième phase après la prophase, qui est le rassemblement des chromosomes condensés à l'équateur de la cellule pour former la plaque équatoriale. Les tensions subies par chacun des kinétochores d'un chromosome s'équilibrent progressivement et ceux-ci s'alignent dans un plan situé à mi-chemin des deux pôles. On observe que les chromosomes sont alignés selon leur centromère. **[2]**

Les chromosomes métaphasiques sont constitués d'un bras court (noté p) et d'un bras long (noté q), reliés entre eux par le centromère qui correspond à un étranglement situé à un niveau variable du chromosome et qui sert de point d'attache au fuseau de division pendant la division cellulaire. **[4]**

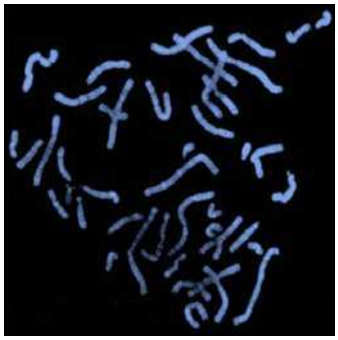


Figure 1.2 : Métaphase traitée par le DAPI

1.4. LE CARYOTYPE

Le caryotype est un arrangement des chromosomes d'une cellule, établi selon leur forme, leur taille et la position de leurs centromères **[2].** Les chromosomes sont photographiés et disposés selon un format standard, le terme caryogramme est parfois employé comme synonyme de caryotype. On réalise des caryotypes lorsqu'on soupçonne qu'une maladie ou

qu'un problème de développement soit lié à une anomalie chromosomique (congénitale ou de croissance), un retard de puberté, une stérilité ou certaines maladies héréditaires.

Le caryotype normal d'une femme contient :

- 22 paires de chromosomes autosomiques.
- une paire de chromosomes sexuels de type XX.

Le caryotype normal d'un homme contient :

- 22 paires de chromosomes autosomiques.
- une paire de chromosomes sexuels de type XY.

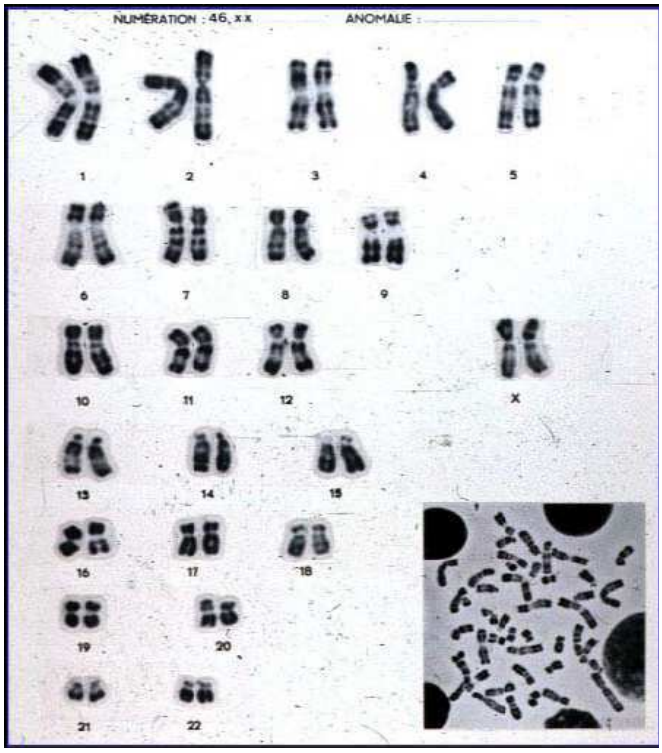


Figure 1.3 : Caryotype d'être humain de sexe féminin

Plusieurs critères vont permettre de reconnaître et de classer les chromosomes : **[4]**

- **La taille :** par convention, les chromosomes sont classés du plus grand au plus petit.
- **L'indice centromérique :** c'est-à-dire le rapport entre la taille du bras court et la taille totale du chromosome :

$$IC=\frac{p}{p+q} \qquad \textbf{(1.1)}$$

Cet indice permet de connaître trois familles de chromosomes :

- Les chromosomes **métacentriques** dont les deux bras ont une taille à peu près équivalente.
- Les chromosomes **submétacentriques** qui ont un bras court franchement plus petit que le bras long.
- Les chromosomes **acrocentriques** dont le bras court est quasi inexistant (on ne trouve sur ces bras courts que les gènes codant pour les ribosomes ; ces gènes étant présents à plusieurs centaines d'exemplaires dans le génome, la perte du bras court d'un chromosome acrocentrique n'a pas de conséquence clinique)

- **Les bandes chromosomiques** : qui sont caractéristiques de chacune des paires. Le nombre de bandes visibles est variable d'une mitose à l'autre et dépend du niveau de condensation du chromosome. Plus les chromosomes sont condensés, moins on peut observer de bandes et moins l'analyse permet de dépister des anomalies de petite taille. Le nombre de bandes par lot haploïde (c'est-à-dire pour 23 chromosomes) permet de définir la résolution de l'analyse cytogénétique ; un caryotype standard a une résolution de 300 à 550 bandes ; certaines techniques dites de haute résolution permettent d'augmenter le nombre de bandes visualisées en bloquant les chromosomes au tout début de leur condensation : on peut ainsi obtenir 800 ou même 1000 bandes par lot haploïde. Ces techniques de haute résolution sont de réalisation et d'interprétation plus délicates que le caryotype standard, mais permettent la mise en évidence d'anomalies de taille beaucoup plus réduite.

1.5. LES ANOMALIES CHROMOSOMIQUES

On appelle anomalie chromosomique tout remaniement du nombre ou de la structure des chromosomes. Ces remaniements peuvent s'observer de manière constitutionnelle (ils sont alors présents dès la naissance) soit de manière acquise au cours de processus (ils ne sont observés alors qu'au niveau des cellules tumorales). Ils résultent d'un accident survenant lors de la division cellulaire. Ils peuvent impliquer un ou plusieurs chromosomes.

On reconnaît par ailleurs les anomalies dites homogènes (quand toutes les cellules examinées portent l'anomalie) et les anomalies en mosaïque quand une fraction seulement des cellules est anormale). Il existe deux sortes d'anomalies : les anomalies de nombre (numériques) et de structure (structurales). **[3] [4]**

1.5.1. Les anomalies de nombre

Elles résultent d'une mauvaise ségrégation des chromosomes au cours de la division cellulaire, les deux chromosomes d'une même paire migrant tous les deux vers la même cellule fille. On obtient une cellule fille avec trois copies du même chromosome (soit 47 chromosomes) ou une deuxième cellule fille avec une seule copie (soit 45 chromosomes). Ces mauvaises ségrégations peuvent s'observer aussi bien au cours de la mitose que de l'une des deux divisions de méiose. Parmi les conséquences de ce type d'anomalie, on peut citer :

- **La trisomie 21, 13, 18 :** La division cellulaire donne 47 chromosomes au lieu de 46.
- **La monosomie :** L'opération donne alors 45 chromosomes au lieu de 46.

1.5.2. Les anomalies de structure

Elles sont la conséquence d'un réarrangement du matériel chromosomique. Si le réarrangement ne s'accompagne ni de perte ni de gain de matériel génétique, il est dit équilibré et n'a habituellement pas de traduction clinique ; dans le cas contraire, on parle d'anomalie déséquilibrée, qui s'accompagne le plus souvent de manifestations cliniques d'autant plus marquées que le déséquilibre est important. Ces réarrangements peuvent concerner un seul chromosome ou plusieurs. On peut avoir pour ce cas plusieurs possibilités :

- **Réarrangements touchant un seul chromosome.**
- **Inversion péricentrique :** deux cassures sur le chromosome, une de chaque côté du centromère. Recollement après inversion du fragment centromérique. Conséquence : modification de l'indice centromérique du chromosome le plus souvent.
- **Inversion paracentrique :** deux cassures sur le même bras chromosomique et recollement après inversion du fragment. Pas de modification de l'indice centromérique.
- **Délétion :** perte d'un fragment de chromosome. Il s'agit toujours d'une anomalie déséquilibrée. La délétion est dite interstitielle quand il y a perte d'un fragment intermédiaire (deux points de cassure comme dans l'inversion), terminale quand l'extrémité d'un bras chromosomique est concernée (un seul point de cassure).

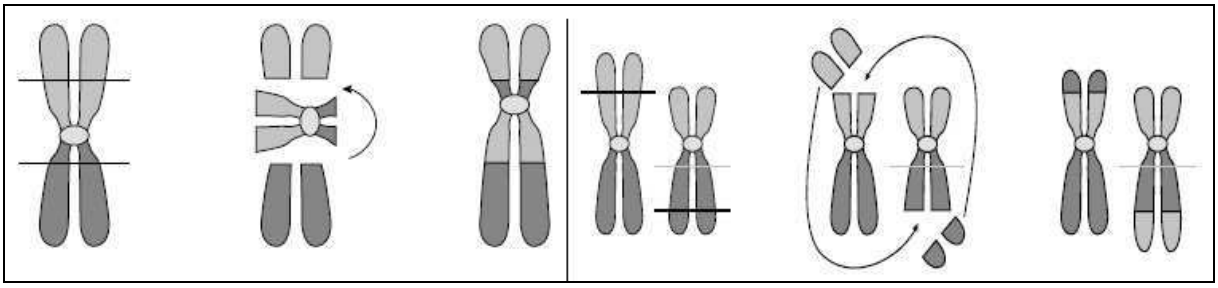


Figure 1.4 : formation d'une inversion péricentrique (à gauche) et d'une translocation réciproque (à droite)

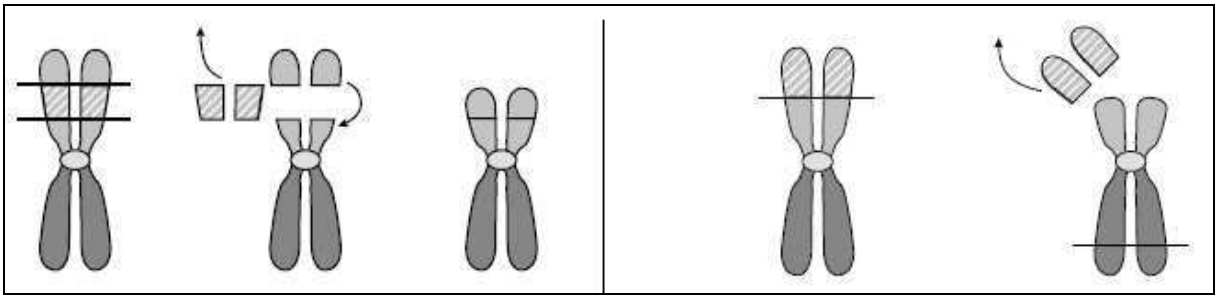


Figure 1.5 : Formation d'une délétion interstitielle (à gauche) et terminale (à droite)

1.6. TECHNIQUE DE HAUTE RESOLUTION

1.6.1. Technique FISH

Fluorescence In Situ Hybridization (FISH) utilise des molécules fluorescentes pour colorer précisément des gènes ou des chromosomes. Cette technique est particulièrement utile pour établir les cartes génétiques et pour identifier les anomalies chromosomiques.

Des séquences d'ADN complémentaires de la séquence d'un gène ou d'une partie du génome sont préparées in vitro et couplés à des fluorochromes. Cette sonde ainsi que les chromosomes sont chauffés, ce qui a pour effet de dissocier les brins complémentaires. Ils sont alors mis en présence et on les laisse refroidir. De cette manière, la sonde fluorescente peut s'apparier avec sa séquence homologue sur le chromosome. On obtient donc un ADN hybride. **[2]**

1.6.2. Technique QFISH

C'est une version simplifiée et rapide de la technique FISH. La technique de FISH modifiée peut détecter des chromosomes humains sans sacrifier la sensibilité ou la qualité du signal. Cette technique nouvelle offre l'avantage d'être plus simple à exécuter et plus rapide que les techniques conventionnelles.

1.6.3. Technique MFISH

La technique FISH multicolore utilise cinq fluorochromes différents, elle offre la possibilité d'identifier chaque chromosome par sa couleur**.** Cela va permettre d'obtenir un caryotype coloré. **[10]**

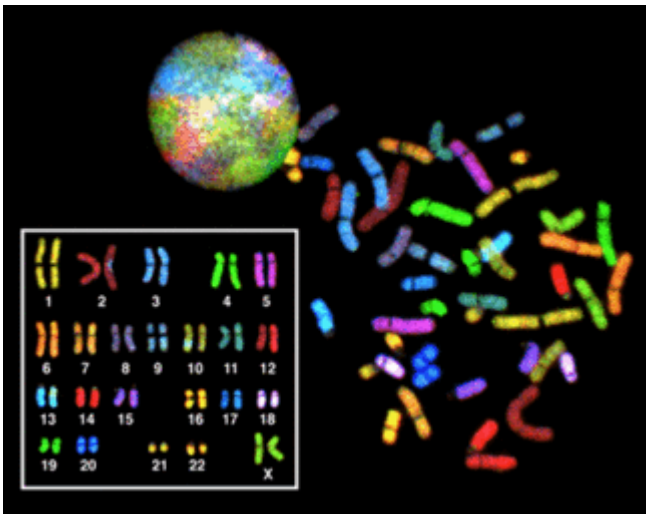


Figure 1.6 : Caryotype d'être humain de sexe féminin

1.6.4. Technique CGH

La Comparative Genomic Hybridation peut étudier le génome entier et détecter facilement des anomalies chromosomiques entraînant un changement du nombre de copies d'ADN. Dans cette approche, l'ADN test d'un malade et un ADN normal de référence sont hybridés simultanément sur une préparation chromosomique normale. Les marquages et hybridation des ADN sont réalisés tels que l'ADN test soit détecté par un fluorochrome (ex : FITC) et l'ADN normal soit détecté à une autre longueur d'onde (ex : Rhodamine). Les ratios

d'intensités de fluorescence vert/rouge obtenu pour chaque chromosome sont le reflet de la compétition des deux ADN testés et indique le type d'anomalie présente (exp: ratio=1.5 est une Trisomie). **[3] [9]**

1.7. CONCLUSION

La cytogénétique a vue une nouvelle ère avec les techniques de hautes résolutions, on est passé de la cytogénétique classique qui travaillait à l'échelle de la structure externe du chromosome à la cytogénétique moléculaire qui s'intéresse au séquençage des gènes.

Cela a ouvert la porte à plusieurs domaines de recherches, particulièrement en médecine, à la fois vastes et complexes. L'exploitation de ces recherches demande encore de nouvelles générations de techniques.

Chapitre II:
Nontions fondamentales sur le traitement d'images

Ce deuxième chapitre est destiné à présenter des notions générales sur l'image numérique, son acquisition et son traitement. Nous commencerons par donner les méthodes générales de prétraitement puis les différentes approches en segmentation d'images. La dernière partie du chapitre présente les différents algorithmes de squelettisation.

2.1. INTRODUCTION

La technologie actuelle a profité du flot d'informations que peut véhiculer une simple image pour créer ce qu'on appelle : la vision artificielle où l'image sera traitée numériquement.

Une image numérique a été longtemps vue comme une suite ordonnée de points dont chacun possède un attribut comme le niveau de gris ou la couleur qui le caractérise.

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.

2.2. L'ACQUISITION D'UNE IMAGE

Différents types de capteurs sont disponibles pour générer des images. Ils se distinguent par leur principe d'acquisition, leur vitesse d'acquisition, leur résolution spatiale, leur gamme spectrale ou encore leur dynamique. Néanmoins, les capteurs utilisés le plus fréquemment dans les systèmes de vision industriels sont les circuits à transfert de charges CCD. Ils résultent de l'association d'une cellule photosensible et d'un dispositif de transfert de charge. Notons que ces capteurs réalisent l'échantillonnage de l'image, mais pas sa quantification. **[9]**

La technologie CCD (Charged Couple Device) :

Un élément CCD est un semi-conducteur métal oxydes formé par une ligne de minuscules cellules photosensibles. Le nombre des cellules dans une ligne détermine la résolution dans le sens horizontal. Si une ligne contient, par exemple, 5000 cellules, un scanner A4 génère 5000 points sur les 21 centimètres de largeur. Si un décalage existe, même d'une seule cellule, des rayures apparaîtront certainement sur le scanner.

Nous distinguons deux familles de caméras CCD :

- **Les caméras matricielles :**

Les photodiodes sont assemblées en une matrice de n lignes et p colonnes.

- **Les caméras uni lignes :**

Comme leur nom l'indique, les photodiodes y sont agencées suivant une seule ligne. Elles ont l'avantage de posséder une résolution supérieure. Ce type de caméra est bien adapté à l'acquisition de documents, on déplace la feuille à vitesse constante perpendiculairement à la caméra et on fait une acquisition d'image suivant une période T. **[10]**

2.3. LA REPRESENTATION D'UNE IMAGE

L'image fait partie des signaux informationnels telle que la parole, considérée comme une projection d'objet tridimensionnelle, elle est représentée par une fonction à deux dimensions f(x, y). L'amplitude de f à toute paire de coordonnées (x, y) est appelé l'intensité ou niveau du gris de l'image à ce point.

La notion d'image qui est utilisée dans la suite est donc de nature bidimensionnelle discrète. A chaque élément (pixel) de l'image, correspond un niveau d'intensité lumineuse appartenant à {0,1,……, N-1} où 0 correspond au manque total d'illumination (couleur noire), et N-1 est la couleur blanche et les autres valeurs sont des niveaux de gris entre le noir et le blanc. **[7] [8]**

Pour des raison de commodité de représentation pour l'affichage et l'adressage, les données images sont rangées sous formes de matrice de n lignes et p colonnes : **[5]**

$$I = \begin{pmatrix} f(0,0) & f(0,1) & .... & ... & .f(0,P-1) \\ f(1,0) & f(1,1) & & & f(1,P-1) \\ ... & ... & & & ... \\ ... & ... & & & ... \\ f(N-1,0) & ... & & & f(N-1,P-1) \end{pmatrix}$$

2.3.1. Le pixel

Contraction de l'expression anglaise " picture éléments ": éléments d'image, le pixel est le plus petit point de l'image, c'est une entité calculable qui peut recevoir une structure et une quantification. Si le bit est la plus petite unité d'information que peut traiter un ordinateur, le pixel est le plus petit élément que peuvent manipuler les matériels et logiciels d'affichage ou d'impression. La lettre A, par exemple, peut être affichée comme un groupe de pixels dans la figure ci-dessous :

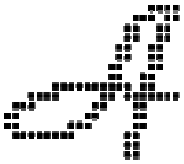


Figure 2.1 : image en pixels

La quantité d'information que véhicule chaque pixel donne des nuances entre images monochromes et images couleurs. Dans le cas d'une image monochrome, chaque pixel est codé sur un octet, et la taille mémoire nécessaire pour afficher une telle image est directement liée à la taille de l'image.

Dans une image couleur (R.V.B.), un pixel peut être représenté sur trois octets : un octet pour chacune des couleurs : rouge (R), vert (V) et bleu (B) **[5]**

2.3.2. Voisinage d'un pixel

L'image discrète est représentée par un maillage carré. Les métriques couramment utilisées en maillage carré sont désignées par d_4 et d_8.

La métrique d_4 est définie par :

$d_4(p,q)=|i_p-i_q| +|j_p-j_q|$ **(2.1)**

Cette métrique permet d'associer à un pixel P un ensemble de points appelés voisinage et défini par :

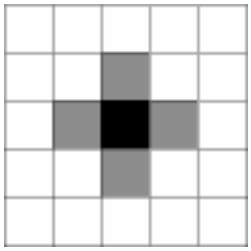
$$v_4 = \{q \in I, d_4(p,q) \leq 1 \}$$ **(2.2)**

Ce voisinage est représenté dans la figure (2.2.a.). La métrique d_8 est définie par :

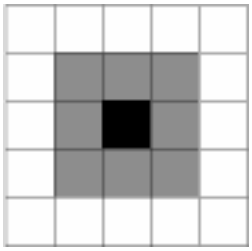
$d_8(p,q)=\text{Max}\ (|i_p-i_q| ,|j_p-j_q|)$ **(2.3)**

Cette métrique permet d'associer à un pixel P un voisinage v_8 défini par : **[6]**

$$v_8 = \{q \in I, d_8(p,q) \leq 1 \}$$ **(2.4)**



(a) 4 Voisinage d'un pixel



(b) 8 Voisinage d'un pixel

Figure 2.2 : Voisinage d'un pixel

2.3.3. Connexité

La connexité entre les pixels d'une image est un concept très important utilisé surtout dans l'établissement des frontières des objets et l'identification des composants d'une région dans une image.

Pour établir si deux pixels sont connectés, nous devons déterminer s'ils sont adjacents quelques parts (par exemple, s'ils appartiennent à un 4-voisinage) et si leurs niveaux de gris respectent un certain critère de similarité. [**12]**

2.4. 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. **[6]**

Il peut être utilisé pour améliorer la qualité d'une image (Rehaussement d'image) en introduisant quelques modifications, pour pouvoir extraire les informations utiles de celle-ci.

L'algorithme général :

Début

Pour chaque niveau de gris **faire**

Balayage de l'image

Incrémentation du nombre de pixels

Fin

Exemple :

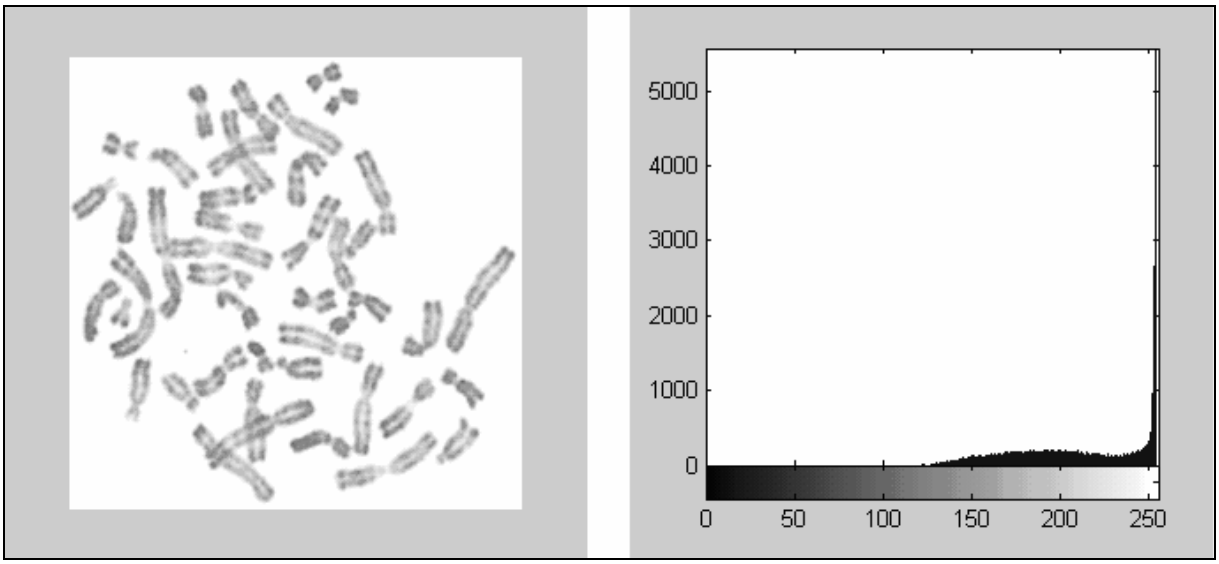


Figure 2.3 : Histogramme d'une image FISH en niveaux de gris

2.5. L'IMAGE BINAIRE

La binarisation consiste en l'obtention à partir d'une image à plusieurs niveaux de gris, d'une image à deux niveaux de gris (0 pour le noir et 255 pour le blanc). Ceci se fait en choisissant un seuil S1 pour lequel :

$$I(L(i,j)) = \begin{cases} \text{Noir si } L(i,j) > S1 \\ \text{Blanc si } L(i,j) <= S1 \end{cases} \quad \textbf{(2.5)}$$

L (i,j) est le niveau de gris du pixel (i,j)

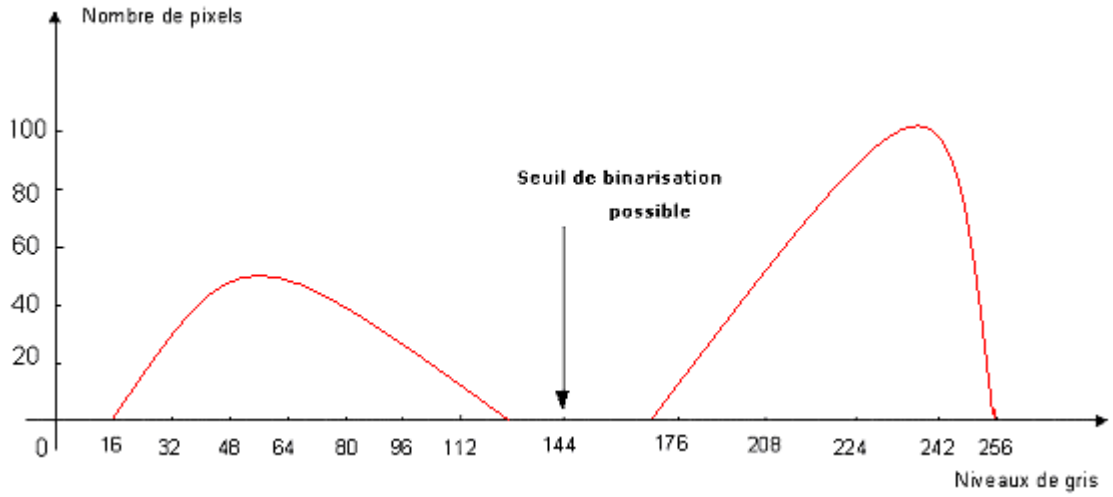


Figure 2.4 : Calcul du seuil de binarisation

L'algorithme général :

Début

Pour chaque pixel **faire**

Si niveau de gris de p<=seuil **faire** p=0

Sinon p=255

Fin

Exemple :

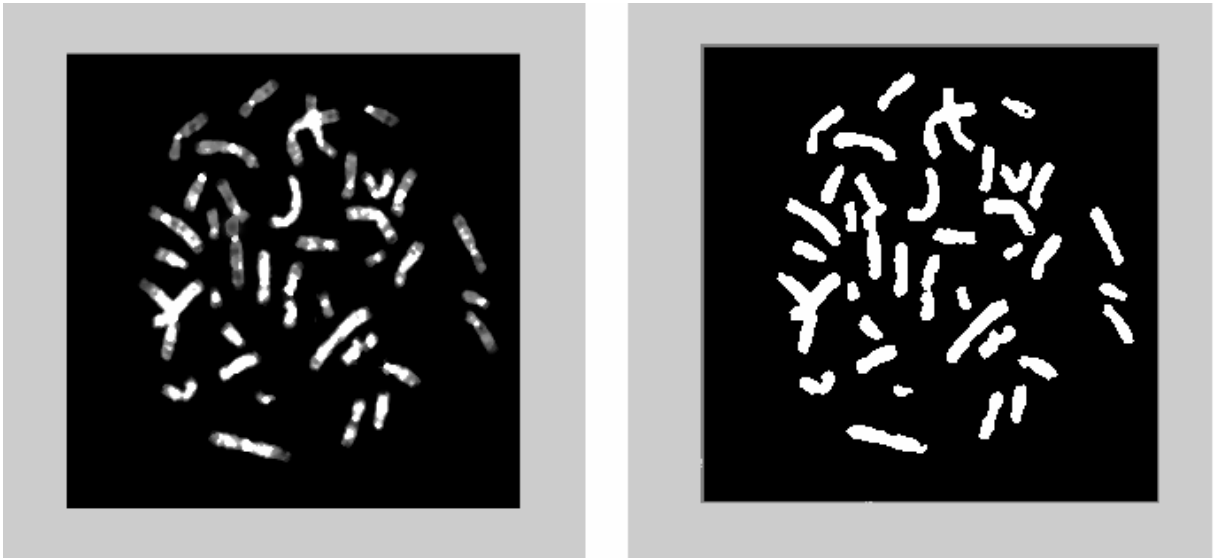


Figure2.5 : Binarisation d'une image FISH en niveaux de gris

2.6. LA RESOLUTION

La résolution d'une image est définie par un nombre de pixels par unité de longueur de la structure à numériser (classiquement en dpi (dots per inches) ou ppp (points par pouce)). Ce paramètre est défini lors de la numérisation et dépend principalement des caractéristiques du matériel utilisé lors de processus de numérisation. Plus le nombre de pixels est élevé par unité de longueur de la structure à numériser, plus la quantité d'information qui décrit cette structure est importante et plus la résolution est élevée.

2.7. LE CONTRASTE

C'est l'opposition marquée entre deux régions d'une image, plus précisément entre les régions sombres et les régions claires de cette image. Le contraste est défini en fonction des luminances de deux zones d'images.

Si L1 et L2 sont les degrés de luminosité respectivement de deux zones voisines A1 et A2 d'une image, le contraste C est défini par le rapport :

$$C = \frac{L1 - L2}{L1 + L2} \qquad \textbf{(2.6)}$$

2.8. LE BRUIT

Un bruit dans une image est considéré comme un phénomène de brusque variation de l'intensité d'un pixel par rapport à ses voisins, il provient de l'éclairage des dispositifs optiques et électroniques du capteur. **[5]**

2.9. FILTRAGE D'IMAGE

Dans la plupart des cas, le bruit d'image est considéré comme étant aléatoire, centré et additif. L'élimination du bruit se ramène alors à un problème de traitement du signal, l'opération aura comme finalité de retrouver par filtrage les niveaux d'intensité nominaux de chacune des régions. L'objectif est donc de réduire l'amplitude des variations d'intensité dans chaque région, tout en conservant les transitions entre régions adjacentes**. [6]**

Nous avons défini le bruit comme un phénomène de brusque variation d'un pixel isolé par rapport à ses voisins. Or une image possède une certaine redondance spatiale : les pixels voisins ont généralement les mêmes caractéristiques. Il est donc nécessaire d'opérer des transformations qui pour chaque pixel tiennent compte de son voisinage.

Les méthodes parmi les plus utilisées sont les techniques dites de filtrage de l'image.

Nous distinguons :

- Le filtrage linéaire
- Le filtrage non linéaire
- Le filtrage Homomorphique.
- Le filtrage Morphologique

2.9.1. Les filtres linéaires

Ces filtres opèrent sur le pixel en lui attribuant une combinaison linéaire des niveaux de gris de son voisinage. Parmi ces filtres nous citons le filtre moyenneur.

- **Filtre moyenneur**

C'est un filtre linéaire dont le principe est de considérer chaque point de l'image et d'en faire la moyenne avec les huit pixels qui lui sont voisins. Ce filtre peut être mis sous la forme d'un masque *H*, qu'on va déplacer sur toute l'image. Il faut noter que le pixel concerné par la transformation est le pixel central.

$$H = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Si I_I représente l'image originale et I_F représente l'image finale après application du filtrage, alors : $I_F = H \otimes I_I$

Où $\otimes$ représente le produit de convolution discrète du masque H avec l'image I_I.

Effectuer le produit de convolution revient à calculer la fonction (2.*), pour chaque pixel de l'image **[13]** :

$$I_F(x,y) = \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} H(i+1, j+1) I_I(x+i, y+j) \qquad \textbf{(2.7)}$$

- **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 smooth 7×7 (3 itérations). 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 utilisés 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. **[6] [11]**

2.9.2. Filtre non linéaires

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. **[6]**

Le filtre non linéaire s'oppose au précédent dans sa dénomination car il n'est pas le résultat d'une combinaison linéaire de pixels, les pixels voisins interviennent suivant une loi non linéaire. Parmi ces filtres, on peut citer le filtre médian.

- **Filtre médian**

C'est un filtre non linéaire dont le principe est de calculer en un point non pas une combinaison linéaire des niveaux de gris de ses voisins, mais une valeur médiane dépendant d'un tri des niveaux de gris des points voisins.

On peut alors suivre les étapes suivantes (Figure 2.5) :

• On classe les pixels voisins du pixel concerné par la transformation par valeurs croissantes.

• On prend la valeur médiane des pixels classés, et on l'affecte au pixel concerné, (comme le montre l'exemple si dessous)

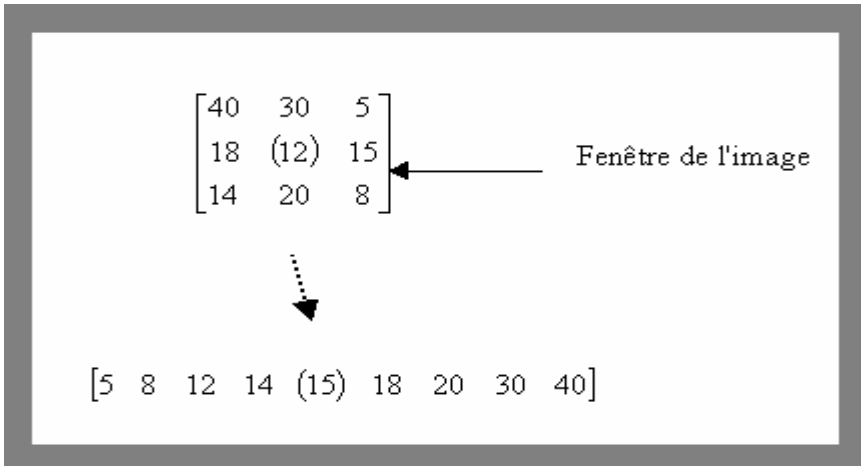


Figure 2.6 : Principe du filtre médian.

Dans cet exemple 12 est l'ancienne valeur du pixel central et 15 sa nouvelle valeur. Alors que le filtre moyen introduit du flou sur le bord des objets, le filtre médian permet l'élimination de parasites isolés dans l'image sans affecter les contours. **[13]**

Il faut souligner l'intérêt du filtre Médian qui est :

- Un pixel non représentatif dans le voisinage affectera peu la valeur médiane.
- La valeur médiane choisie étant le niveau de gris d'un des pixels considérés, nous ne créons pas alors de nouveaux niveaux de gris dans l'image. Ainsi, lorsque le filtre passe sur un contour très marqué il le préservera mieux.

Le filtre médian garde donc 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. **[24]**

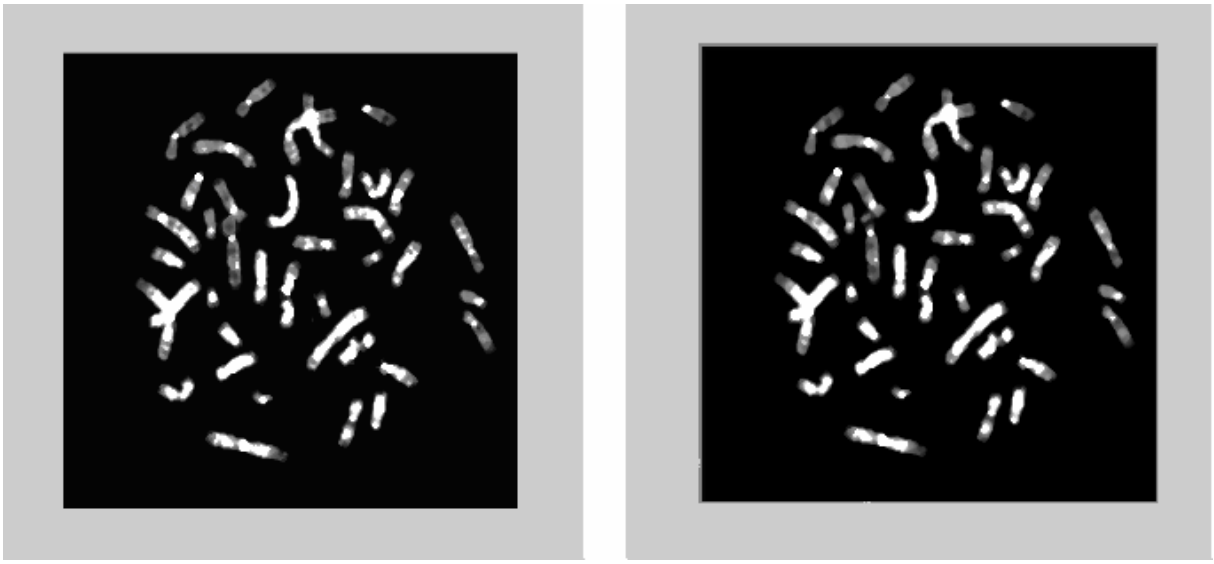
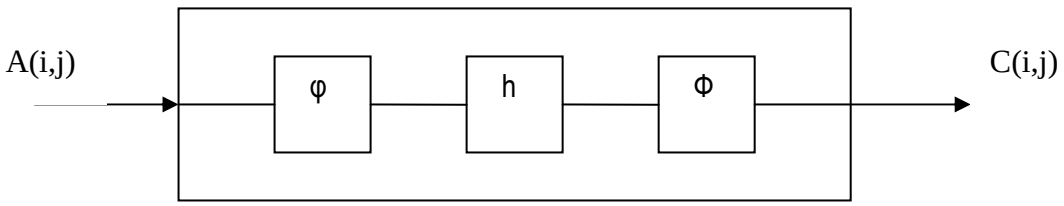


Figure2.7 : Le filtre Médian appliqué à une image FISH en niveau de gris

2.9.3. Filtre homomorphique

Le filtrage homomorphique est réalisé par une combinaison de traitements linéaire. Il permet souvent de se ramener à un problème de lissage de bruit additif. **[6]**

Le schéma de principe d'un filtre homomorphique est:



Où φ et Φ sont des opérateurs non linéaires réciproques. H est la réponse impulsionnelle d'un filtre linéaire généralement de type passe bas.

2.9.4 Filtrage morphologique

Ce type de filtrage est utilisé pour éliminer des pixels isolés dans des images binaires (2 niveaux de gris : noir (0) et blanc (1)) qui sont considérés comme du bruit. Il met en correspondance chaque pixel et son voisin par une fonction logique (ET, OU, XOR). Parmi les opérateurs morphologiques, nous citons : **[19] [20] [24]**

- ## La Dilatation

La dilatation permet d'éliminer les pixels blancs isolés. On effectue le **ET** logique des huit voisins du pixel considéré.

Elle 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.

On suivra donc les étapes suivantes : **[24]**

- On classe les pixels voisins du pixel concerné par la transformation par valeurs croissantes.
- On prend la valeur maximale des pixels classés, et on l'affecte au pixel concerné, (dans l'exemple si dessous, le pixel en question garde la même valeur vu qu'elle est la plus grande)

178	182	230
148	230	230
138	146	149

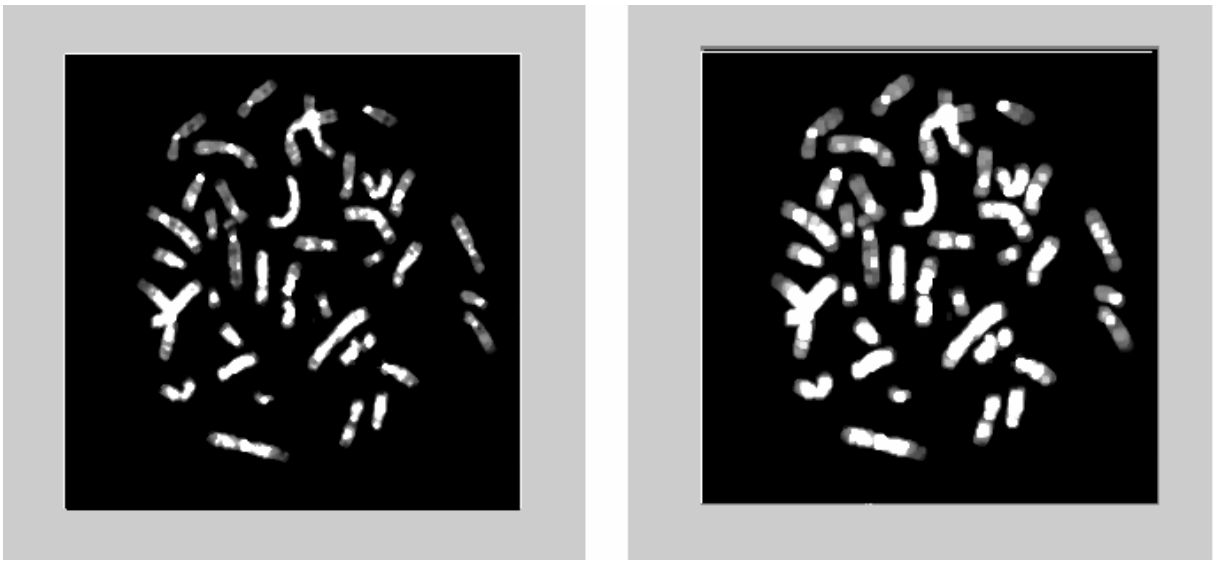


Figure2.8 : Dilatation d'une image FISH en niveau de gris

- ## L'Erosion

L'erosion permet d'éliminer les pixels noirs isolés au milieu des parties blanches de l'image, on effectue le **OU** logique des huit voisins du pixel considéré.

En appliquant une érosion, ces tâches noires peuvent être éliminées mais la taille des objets présents dans l'image diminue car l'érosion enlève des pixels du contour, entraînant une déformation de certains objets.

L'érosion consiste en le choix du plus petit des éléments du masque. Elle entraîne des effets habituels tels que : la séparation des objets à l'endroit des étranglements, le rétrécissement des objets de grande taille et la disparition des petites composantes. Dans l'exemple si dessous, le pixel en question prend la valeur 138 vu qu'elle est la plus petite **[24].**

178	182	230
148	230	230
138	146	149

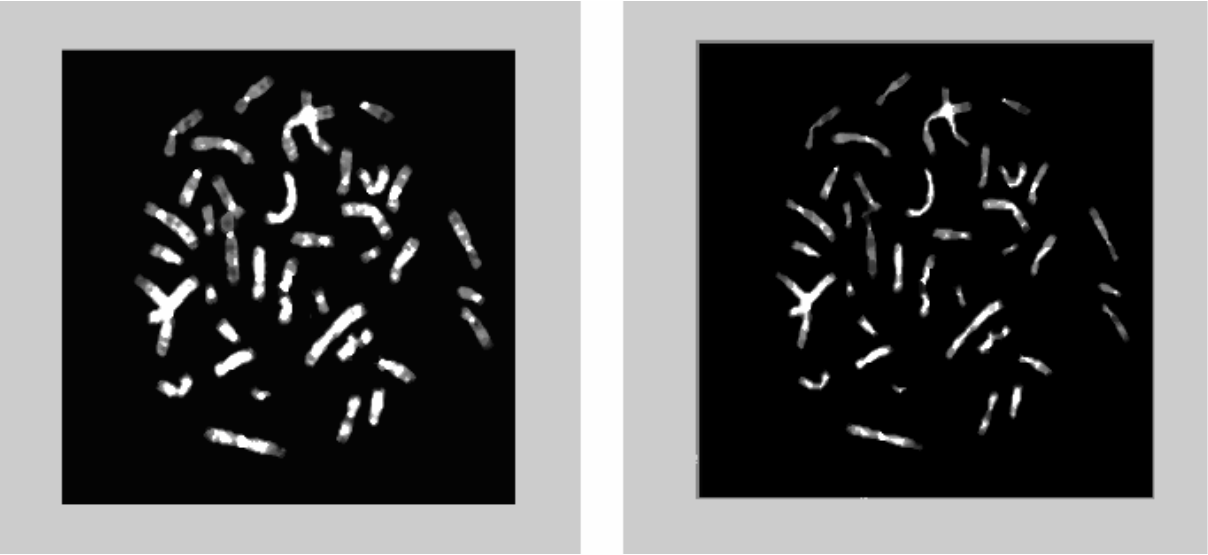


Figure2.9 : Erosion d'une image FISH en niveau de gris

- **L'Ouverture**

L'ouverture est constituée par une opération d'érosion suivie d'une dilatation. Elle permet de retrouver la taille normale des objets de l'image.

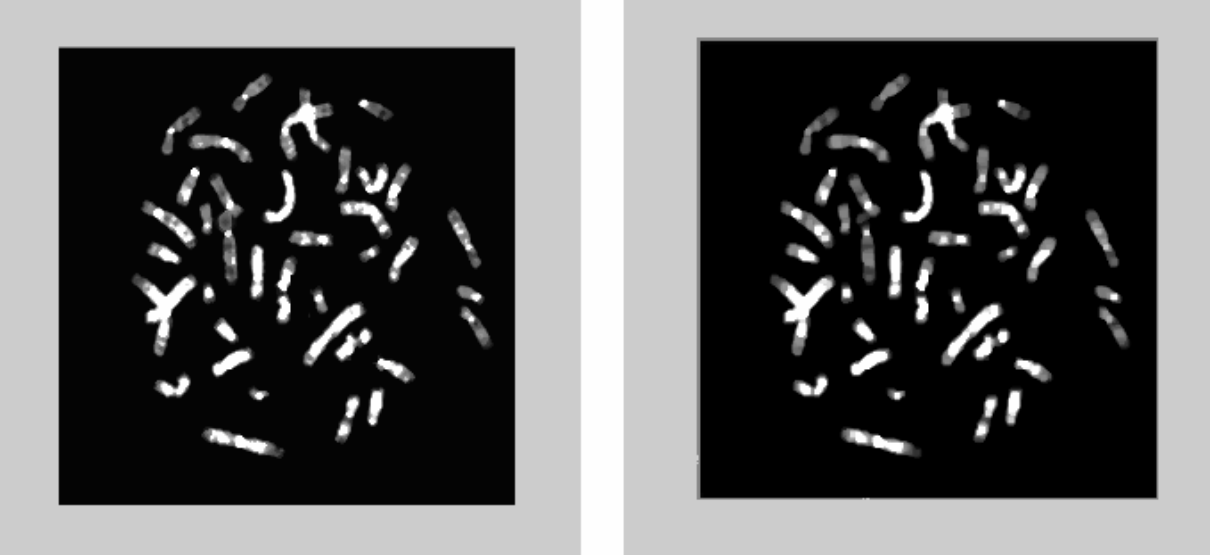


Figure2.10 : Ouverture d'une image FISH en niveau de gris

- **La Fermeture**

La fermeture est une opération morphologique qui consiste à faire subir à l'image une opération de dilatation suivie d'une érosion. Elle permet de retrouver la taille normale des objets de l'image.

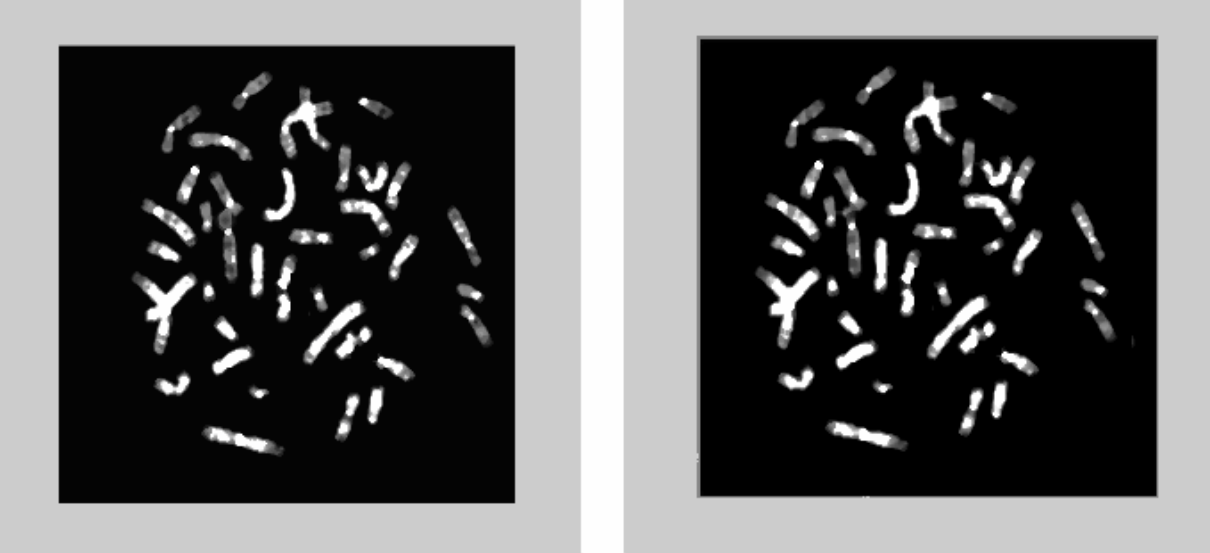


Figure2.11: Fermeture d'une image FISH en niveaux de gris

2.10. SEGMENTATION D'IMAGES

La segmentation vise la décomposition spatiale de l'image en zones d'intérêt homogènes (au vu de certaines caractéristiques) et à l'extraction d'entités significatives.
Elle joue un rôle prépondérant dans le traitement et l'analyse d'image et la vision par ordinateur.

Elle constitue une étape fondamentale, qui se situe entre, d'une part, l'acquisition de l'image et son amélioration et, d'autre part, sa description et la prise de décision finale. **[14]**
Une région est un ensemble connexe de points image (pixels) ayant des propriétés communes (intensité, texture…) qui les différencient des pixels des régions voisines. Les connaissances utilisées sont le plus souvent du domaine de l'image numérique et du traitement du signal **[6]**.

La segmentation d'image étant la pierre de base de tout système de vision. Une image constitue une représentation d'un univers composé d'entités (objets). Le but de toute méthode de segmentation est l'extraction d'attributs caractérisant ces objets. Ces attributs correspondent à des points d'intérêt (contours) ou des zones caractéristiques de l'image (régions). D'où la naissance de deux approches : Frontière et région. **[15]**

2.10.1. Détection de contours

La détection de contours dans les images a débuté de façon extrêmement empirique par des opérateurs locaux qui, soit estimaient un gradient, soit convoluaient l'image par des masques caractéristiques des contours. Dans les années 80, des approches plus systématiques ont été mises en place par Marr, puis Canny en 1986, pour obtenir des contours plus significatifs. Ces travaux ont abouti à une bonne compréhension de ce qu'il faut faire pour détecter les contours, mais la définition même des contours demeure très vague, ce qui rend ces techniques encore peu efficaces sur un problème concret. Même si de très gros progrès ont été accomplis dans ce domaine, les techniques empiriques d'estimation du gradient proposées dans les années 70-80 restent souvent encore employées en concurrence de techniques plus modernes. **[19] [20]**

Dans une image, les variations d'intensité représentent des changements de propriétés physiques ou géométriques, de la scène ou de l'objet observé correspondant par exemple à :

- Des variations d'illumination, des ombres.
- Des changements d'orientation ou de distance à l'observateur.
- Des changements de réflectance de surface.
- Des variations d'absorption des rayons (lumineux, X, etc.).

Dans un grand nombre de cas, ces variations d'intensité sont des informations importantes pour les opérations situées en aval de la segmentation**.**

Il existe deux approches selon lesquelles se fait la détection de contour :

- **Approche gradient :** détermination des extrémas locaux dans la direction du gradient.
- **Approche laplacien :** détermination des passages par zéro du laplacien.

Ces approches reposent sur le fait que les contours correspondent des discontinuités d'ordre 0 de la fonction d'intensité.

Le calcul de dérivée nécessite un pré filtrage des images. Filtrage linéaire pour les bruits de moyenne nulle (par exemple bruit blanc Gaussien, filtre Gaussien). Filtrage non-linéaire pour les bruits impulsionnels (filtre médian par exemple). **[6] [16]**

Les différentes approches existantes se classent ensuite suivant la manière d'estimer les dérivées de la fonction d'intensité, il existe alors :

- Les méthodes dérivatives.
- Les méthodes morphologiques.

2.10.1.1 Opérateur dérivatif du premier ordre

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. **[6]**

Le gradient d'une image est obtenu en appliquant le vecteur suivant, équation (2.8) :

$$\nabla I(x,y) = \left(\frac{\partial I(x,y)}{\partial x}, \frac{\partial I(x,y)}{\partial y} \right)^t \quad \textbf{(2.8)}$$

Ce vecteur peut être représenté par son module *m* (relation 2.9) et sa phase ϕ (relation 2.10) qui donne la direction :

$$m = \sqrt{\left(\frac{\partial I(x,y)}{\partial x} \right)^2 + \left(\frac{\partial I(x,y)}{\partial y} \right)^2} \quad \textbf{(2.9)}$$

$$\phi = \arctan\left(\frac{\partial I(x,y)}{\partial y} / \frac{\partial I(x,y)}{\partial x} \right) \quad \textbf{(2.10)}$$

D'après la relation (2.8) il s'agit, dans le cas discret, de calculer les différences des luminances suivant *x* et *y*. Pour des commodités de calculs, on prend :

$$m = \left|\frac{\partial I(x,y)}{\partial x}\right| + \left|\frac{\partial I(x,y)}{\partial y}\right| \quad \textbf{(2.11)}$$

On peut donc définir les relations suivantes (2.12) appliquées dans le cas discret :

$$\begin{cases} G_x = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ et } G_y = \begin{bmatrix} 1 & -1 \end{bmatrix} \\ I_x(i,j) = \begin{bmatrix} I(i,j) & I(i,j+1) \end{bmatrix} G_x \\ I_y(i,j) = G_y \begin{bmatrix} I(i,j) & I(i+1,j) \end{bmatrix}^t \\ I'(i,j) = \left|I_x(i,j)\right| + \left|I_y(i,j)\right| \end{cases} \quad \textbf{(2.12)}$$

- **Opérateurs de Prewitt et de Sobel**

Pour ces opérateurs, les dérivées directionnelles horizontale et verticale s'expriment sous la forme :

$$A_j[i,j] = h_j * A[i,j] \quad \text{et} \quad A_i[i,j] = h_i * A[i,j]$$

$$\text{Avec: } h_j = \begin{pmatrix} 1 & 0 & -1 \\ c & 0 & -c \\ 1 & 0 & -1 \end{pmatrix} \text{ et } hi = \begin{pmatrix} 1 & c & 1 \\ 0 & 0 & 0 \\ -1 & -c & -1 \end{pmatrix}$$

Les matrices h_i et h_j, appelées aussi masques, sont les noyaux de convolution de filtres à réponse impulsionnelle finie.

Les masques de Prewitt sont définis par c=1 et les masques de Sobel par c=2.

Rappelons que la relation entre convolution et corrélation peut être traduite par l'expression :

$$h*A[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} h(m,n).A[i-m,j-n]$$

$$= \sum_{m=-M}^{M} \sum_{n=+N}^{N} h(-m,-n).A[i+m,j+n] \quad \textbf{(2.13)}$$

Où h est le noyau de convolution de taille (2M+1)*(2N+1).

Les calculs précédents des deux dérivées directionnelles peuvent donc être considérées comme des corrélations avec les « gabarits » $h_j[-j]$ et $h_i[-i]$.**[17]**

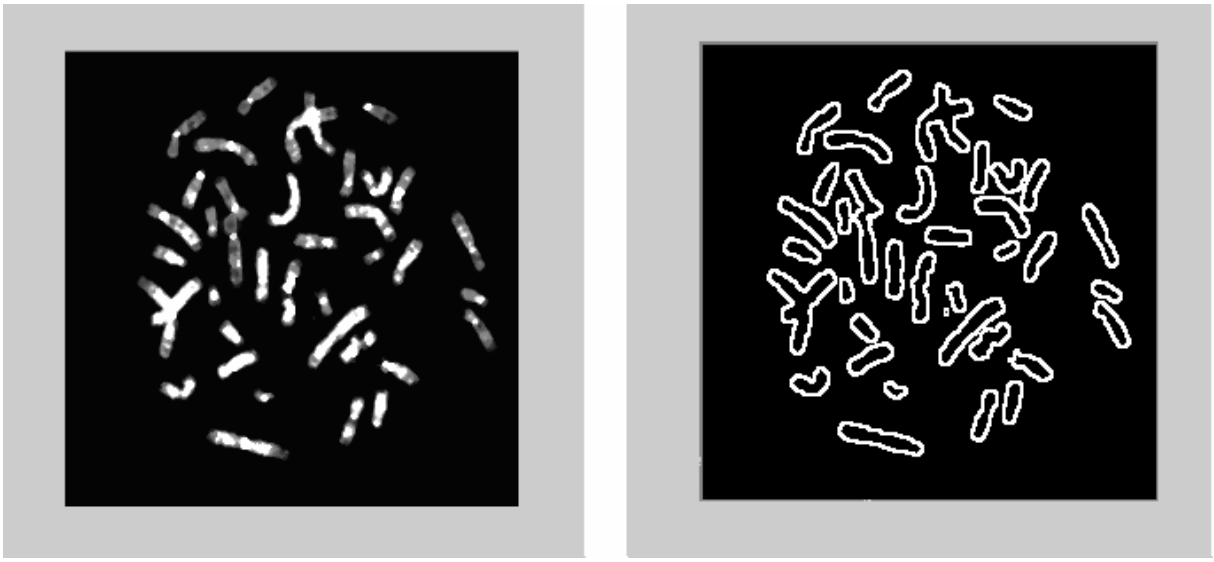


Figure2.12 : Opérateur de Prewitt appliqué sur une image FISH après binarisation

- **Opérateur 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 :

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

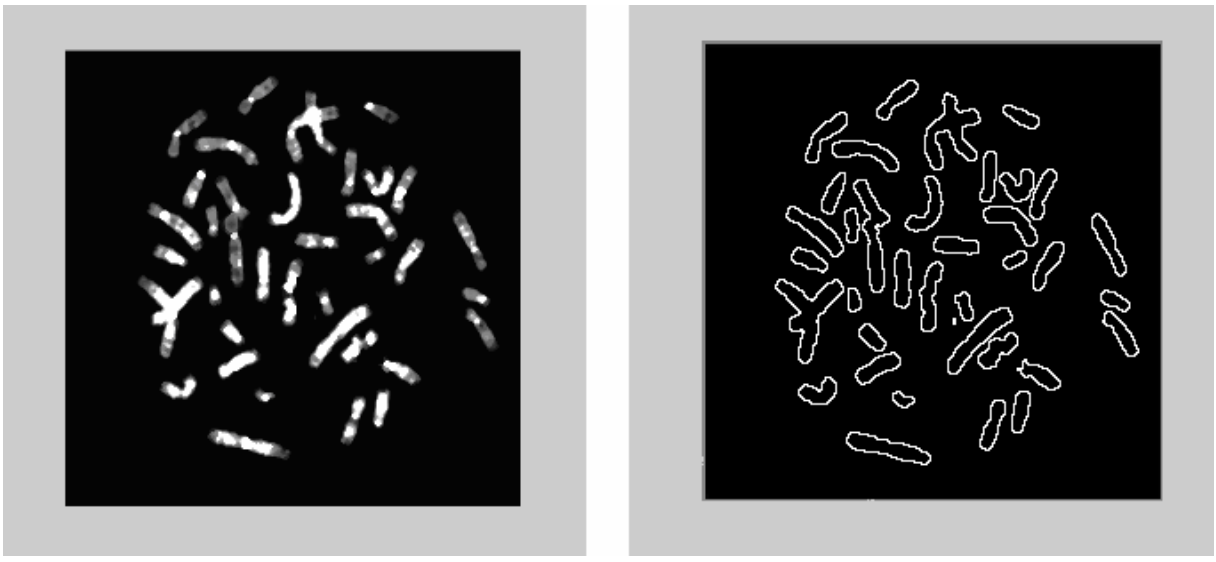


Figure2.13 : Opérateur de Roberts appliqué sur une image FISH après binarisation

2.10.1.2 Opérateur dérivatif du deuxième ordre

Cette méthode consiste à calculer le Laplacien de l'image, et à repérer les points de contours par son passage par zéro **[6].** Le Laplacien d'un signal continu est donné par la relation suivante:

$$\Delta I(x,y) = \frac{\partial^2 I(x,y)}{\partial^2 x} + \frac{\partial^2 I(x,y)}{\partial^2 y} \qquad \textbf{(2.14)}$$

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

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

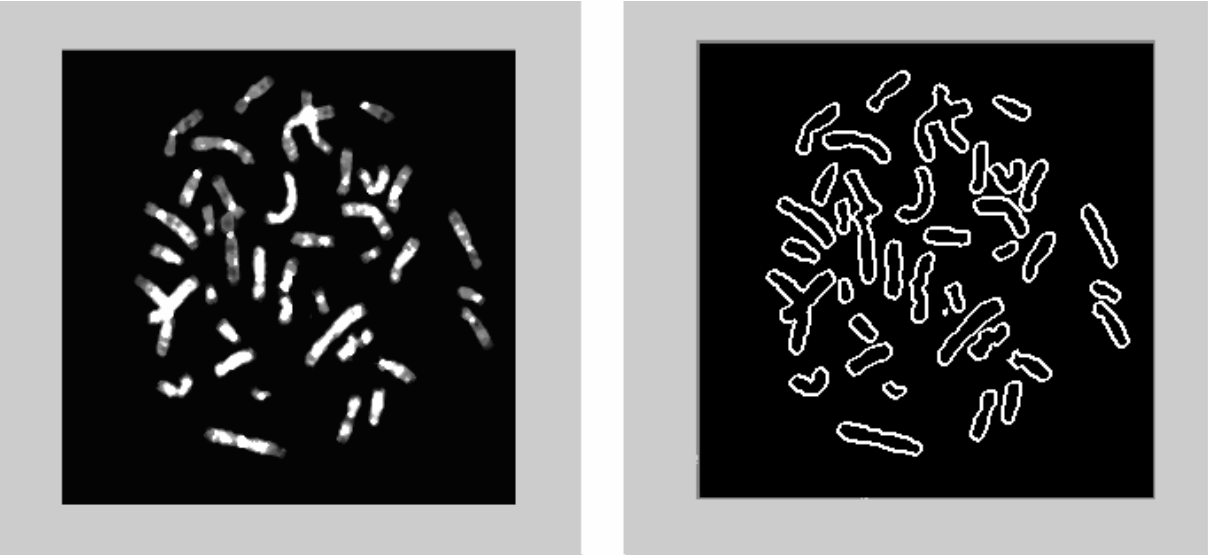


Figure2.14 : Laplacien 1 appliqué sur une image FISH après binarisation

2.10.1.3 Les méthodes morphologiques dans la détection de contours

Le filtrage morphologique est utilisé en traitement d'image pour partitionner une image en zones homogènes selon certains critères et également pour isoler directement certains objets à étudier.

La détection de contours se fait par le calcul de la norme du gradient morphologique. Ce calcul du gradient est très sensible au bruit, il ne peut s'appliquer que sur une image lissée. De plus les contours obtenus après un simple seuillage sont épais, il faut donc les amincir.

Donc, l'algorithme de détection de contours par la morphologie mathématique peut être résumé comme suit :

- Lissage de l'image par un filtre alterné séquentiel comportant une fermeture suivie d'une ouverture (avec le disque unité). (voir chapitre précédent)
- Calcul du gradient morphologique symétrique en tout point.
- Amincissement de l'image avec l'élément structurant *L* dans toutes les directions jusqu'à stabilisation.
- Seuillage.

Notons que nous pouvons aussi utiliser le gradient par érosion ou le gradient par dilatation pour effectuer la détection de contours. **[17][18]**

2.10.2. Approche région

L'approche duale de la détection des contours pour la décomposition d'une image en ses formes élémentaires est l'approche par régions. Elle repose sur la recherche de zones possédant des attributs communs, soit de luminosité, soit, plus rarement, de textures.

La notion de « région » fait référence à des groupements de points ayant des propriétés communes. Les méthodes de l'approche région aboutissent directement à une partition de l'image, chaque pixel étant affecté à une région unique.

Contrairement aux techniques d'extraction de contours, la segmentation en régions homogènes est basée sur les propriétés intrinsèques des régions. Le choix de ces propriétés détermine ce qu'on appelle « le critère de segmentation »**. [20]**

Dans cette approche, on peut regrouper :

- Les méthodes de classification.
- Les méthodes Markoviennes.
- Les méthodes structurales.
- Les méthodes utilisant les critères d'homogénéité.

2.10.2.1 Segmentation par utilisation de critères d'homogénéité

La construction des régions de l'image se fait en utilisant un ou plusieurs critères d'homogénéité. Il existe plusieurs méthodes qui utilisent les critères d'homogénéité.

Les principaux critères d'homogénéité d'une région utilisés sont :

- Niveau de gris.
- La couleur pour les images en couleurs.
- Le mouvement.

Plusieurs méthodes sont utilisées :

Méthodes par séparation (méthodes globales)

- A partir d'une image initiale, on teste si le critère d'homogénéité est vérifié.
- Si c'est le cas l'algorithme s'arrête.
- Dans le cas contraire, on décompose l'image en zones plus petites, on réapplique la méthode pour chaque zone, jusqu'à l'obtention d'une zone qui valide le critère.

Méthodes par fusion

- A partir d'une partition assez fine de l'image (parfois composée d'un seul pixel).
- On fusionne les régions adjacentes si elles vérifient le critère d'homogénéité.
- On s'arrête quand le critère de fusion n'est plus vérifié.

Méthodes mixtes (split and merge)

Ces méthodes combinent les méthodes précédentes. A partir d'une partition arbitraire de l'image en régions, à chaque étape de l'algorithme les régions sont soit divisées si elles ne sont pas homogènes, soit fusionnées si elles sont similaires.

Elles nécessitent l'utilisation de structures de données compliquées (arbre). Par conséquent, le temps de calcul (parcours de l'arbre) et l'espace mémoire utilisé seront considérables. **[13][21]**

Segmentation par étiquetage en composantes connexes

Cette méthode est récursive et simple à implémenter, nous l'exposerons d'une manière plus exhaustive, pour les besoins de notre application, dans ce qui suit.

2.10.3 SEGMENTATION PAR ETIQUETAGE EN COMPOSANTES CONNEXES

Partant d'une image binaire, on souhaite souvent faire des mesures sur les ensembles de points connexes qui la composent. Ces éléments sont appelés les composantes connexes de l'image.

L'étiquetage des composantes connexes d'une image binaire consiste à attribuer un label, étiquette ou numéro différent pour chaque composante connexe et identique pour tous les pixels d'une même composante. Il existe de nombreux algorithmes réalisant cette fonction. Ils dépendent de la connexité considérée et se différencient aussi par leur approche séquentielle ou parallèle.

Une composante connexe peut alors être extraite par l'intermédiaire de son étiquette pour faire un traitement spécifique. L'image des étiquettes est une formulation implicite des composantes connexes; l'extraction aura pour but la transformation de cette formulation implicite en une formulation explicite (liste des composantes connexes avec des attributs par exemple).

2.10.3.1 Prédécesseurs d'un pixel

Il est nécessaire de définir cette notion dans le cadre du traitement des images binaires. Soit P un pixel d'une image. Dans le schéma suivant, les pixels indiqués ***a*** sont les prédécesseurs du pixel *P* dans un balayage avant, les pixels indiqués ***b*** sont les prédécesseurs de *P* dans un balayage arrière.

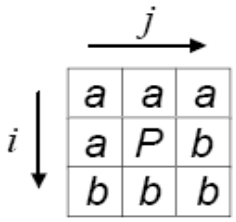


Figure 2.15: Désignation des prédécesseurs d'un pixel
(En fonction du sens de balayage)

2.10.3.2 Etiquetage séquentiel avec correspondance entre points

L'idée de l'algorithme est d'exploiter l'étiquetage effectué lors du premier parcours pour une affectation finale des étiquettes ne demandant ainsi qu'un seul parcours supplémentaire. Une table de correspondance T est créée et initialisée par T(i) = i. Le premier balayage séquentiel de l'image est défini par le traitement suivant, appliqué en tout point P.

- **Algorithme général**

Pour chaque pixel de l'image **faire**
Si Tous les prédécesseurs de P appartiennent au fond
Alors Affecter une nouvelle étiquette à P
Sinon
    Si Tous les prédécesseurs de P qui sont objet ont une même étiquette
    Alors Attribuer cette étiquette à P
    Sinon
        Rechercher la plus petite étiquette **non nulle** de ces prédécesseurs
        Affecter T(e) à P
        Mettre à jour la table T par
        Pour chaque prédécesseur d'étiquette **a** telle que T(e) != T(a) **Faire**
            Tant que T(a) != a **Faire**
            k = T(a)
            T(a) = T(e)
            a = k
            Fin Faire
        Fin Pour
    Fin Si
Fin Si

A la fin de ce balayage de l'image, on actualise la table *T* de manière à ce qu'à tout indice corresponde l'étiquette définitive de l'objet.

Pour i = 1 à nombre d'étiquettes utilisées

Faire
    j = i
    Tant que T(j) != j Faire j = T(j)
    T(i) = j
    Fin pour

Au terme de ce traitement, les points d'un même objet peuvent avoir des valeurs différentes mais la table *T* permet de les faire toutes correspondre à une même étiquette.

Le second balayage sert à l'attribution définitive sur l'image d'une même étiquette pour tous les points d'un même objet. Cet algorithme nécessite donc deux balayages et l'emploi d'une table de correspondance. Il induit également la génération de nombreuses étiquettes temporaires dans le cas d'un objet de forme complexe, ce qui implique la définition d'une table *T* de grande taille. Ceci est dû à la vue uniquement locale de l'algorithme lors de l'attribution des étiquettes.

2.11 SQUELETTISATION DE L'IMAGE

Le squelette est une représentation d'une forme très utilisée car il conserve les propriétés topologiques de la forme qu'il représente. La notion de squelette est apparue pour l'étude des objets minces. En effet, pour de telles figures, il est certain que c'est l'allure d'une représentation filiforme qui est importante. La reconnaissance des caractères en constitue un exemple typique.

La squelettisation est une étape essentielle de la reconnaissance de formes. Elle a pour but de décrire chaque objet par un ensemble de lignes infiniment fines (analogues à une ossature de l'objet), réduisant sensiblement le volume d'information à manipuler. Le squelette est généralement défini comme étant l'ensemble des lignes médianes, c'est-à-dire l'ensemble des points équidistants de deux points de la frontière. L'obtention du squelette des images mosaïques binaires peut conduire à des erreurs de connexité.

L'idée de squelettisation consiste à centrer dans la forme un squelette qui soit significatif de l'élongation et des déformations de celle-ci. Typiquement, le squelette d'un cercle va alors être son centre, celui d'une ellipse son grand axe, ... Malheureusement, le squelette, pour être fidèle à la forme initiale, est très sensible au bruit (petite déformation du contour, présence d'un trou, ...).

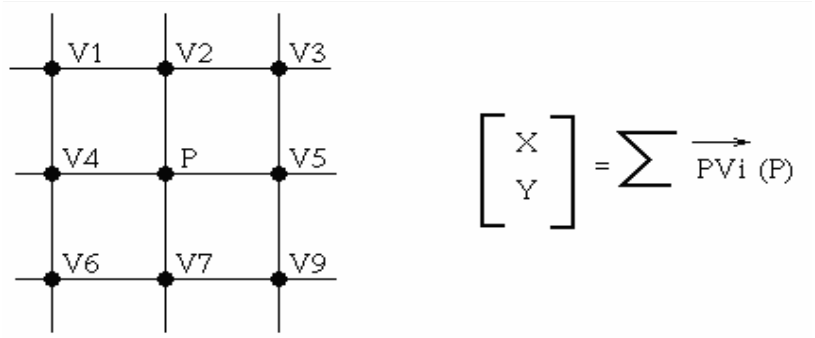
2.11.1 Algorithme de Marthon

Cet algorithme est également un algorithme de suppression de points:

Soit le point M considéré de coordonnées (x, y). Soit l'ensemble Mi (xi, yi) de ses points voisins en n-connexité, la conservation ou non du point M lors de la squelettisation dépend des deux valeurs X et Y définies comme suit :

$$X=(x1-x)+(x2-x)+...+(xn-x) \quad \textbf{(2.15)}$$

Y=(y1-y)+(y2-y)+...+(yn-y) **(2.16)**



Si un point est intérieur à l'objet alors |X|+|Y| est petit. Si le point est au bord de l'objet alors |X|+|Y| est grand. En conséquence,

- Si |X|+|Y|=4, alors le point M est supprimé
- Si |X|+|Y|<=2, alors le point M est conservé
- Si |X|+|Y|=3, alors le point M est conservé ou supprimé suivant le nombre de ses voisins. **[22]**

2.11.2 Algorithme de Stentiford

Cet algorithme utilise un ensemble de matrices 3 × 3 pour balayer l'image.

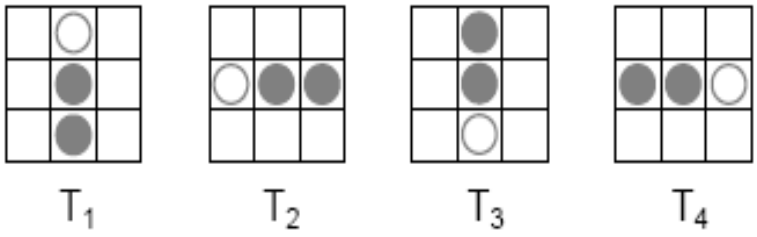


Figure 2.16 : Modèles pour identifier les pixels à éroder selon l'algorithme de Stentiford. Les cases vides correspondent aux pixels dispensés de vérification.

L'algorithme de Stentiford peut être exposé comme suit **: [23]**

- Localiser un pixel de coordonnées (i, j) qui correspond au modèle T_1. Avec ce modèle, tous les pixels du haut de l'image seront retranchés, de gauche à droite et de haut en bas.
- Si le pixel central n'est pas un *pointfinal*, et s'il a un nombre de connectivité égal à 1, alors le marquer pour suppression.

Pixel pointfinal : un pixel est considéré comme *pointfinal* s'il n'est connecté qu'à un seul pixel. Cela dit, si un pixel noir n'a qu'un seul voisin noir sur ses huit possibles voisins.

Nombre de connectivité : c'est une mesure du nombre d'objets reliés à un pixel particulier.

$$C_n = \sum_{k \in S} N_k - \left(N_k \times N_{k+1} \times N_{k+2}\right) \quad \textbf{(2.17)}$$

Où : N_k est la couleur des huit voisins du pixel analysé. N0 est le pixel central. N1 est la valeur de la couleur du pixel à la droite du central, et le reste est numéroté dans le sens des aiguilles d'une montre autour du centre.

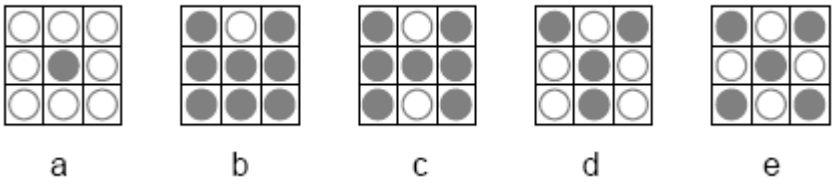


Figure 2.17 Nombre de connexités

a) Nombre de connectivité = 1.
b) Nombre de connectivité = 1, le pixel central peut être ôté sans affecter la connectivité entre gauche et droite.
c) Nombre de connectivité = 2, la suppression du pixel central déconnectera les deux côtés.
d) Nombre de connectivité = 3.
e) Nombre de connectivité = 4.

- Répéter les étapes 1 et 2 pour tous les pixels correspondant à T1.
- Répéter les étapes 1 à 3 pour le reste des modèles : T2, T3 et T4.

T2 correspondra aux pixels du côté gauche de l'objet, mouvant de bas en haut et de gauche à droite. T3 sélectionnera les pixels du bas de l'image meut de droite à gauche et de bas en haut. T4 localise les pixels du côté droit de l'objet, mouvant de haut en bas et de droite à gauche.

- changer en blanc la couleur des pixels marqués pour suppression.

2.11.3 Algorithme de Zang et Suen

Cet Algorithme est rapide et simple à implémenter. Il est constitué de deux sub-itérations. Dans la première, un pixel p (i, j-1) est supprimé si les conditions suivantes sont satisfaites : **[23]**

- Son nombre de connectivité est un.
- Il a au moins deux voisins noirs et pas plus de six.
- Au moins un des p (i, j+1), p (i-1, j) et p (i, j-1) sont blancs.
- Au moins un des p (i-1, j), p (i+1, j) et p (i, j-1) sont blancs.

Dans la seconde sub-itération, les conditions 3 et 4 changent :

- Son nombre de connectivité est un.
- Il a au moins deux voisins noirs et pas plus de six.
- Au moins un des p (i-1, j), p (i, j+1) et p (i+1, j) sont blancs.
- Au moins un des p (i, j+1), p (i+1, j) et p (i, j-1) sont blancs.

A la fin, les pixels qui satisfont les conditions seront ôtés. Si à la fin de chaque sub-itération il n'y a point de pixel à supprimer, alors l'algorithme s'arrête.

2.11.4 Algorithme d'épluchage –nettoyage

Cet algorithme est basé sur deux phases de traitement dans l'une, se fait la suppression des points et dans l'autre, se fait le nettoyage du squelette.

1ère phase : **Epluchage**

On considère que le pixel du fond est Blanc et le pixel du motif Noir. On balaye l'image avec les 16 masques 3x3 issus de la version "maillage rectangulaire" de l'alphabet de Golay. (Figure 2.7).Si l'une des configurations est vérifiée, le pixel central est remplacé par Noir. L'opération est réitérée jusqu'à stabilité.

2ème phase : **Nettoyage**

Si les frontières de l'objet sont bruitées, le squelette peut comporter beaucoup de barbules. Il convient alors d'appliquer alors des post-traitements, également sous forme de masques, représentant des transformations de voisinage. Ces masquages (au nombre de 8) suivent le même principe que l'épluchage.

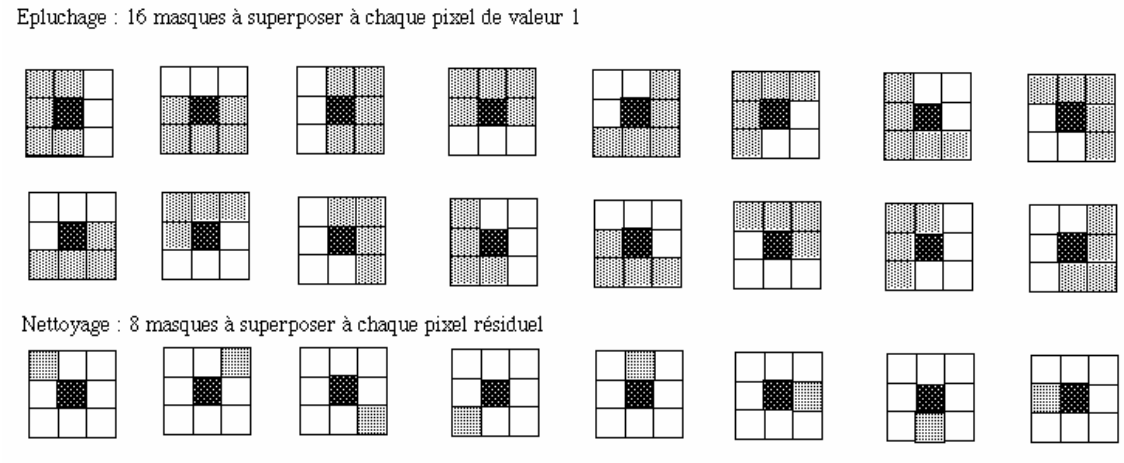


Figure2.18 : Les masques d'épluchage et de nettoyage

2.12. CONCLUSION

Ce chapitre a fait l'objet d'une étude générale, pas très exhaustive, des techniques de traitement d'images ; en commençant par les méthodes générales de prétraitement et en terminant par les approches de segmentation et les algorithmes de squelettisation en guise d'introduction à un travail élaboré en vue d'une application des théories présentées.

Chapitre III
Etude de la famille des DSP C6000 et outils de développement

Dans la première partie de ce chapitre, nous allons faire une étude des DSP de la famille C6000, en particulier le TMS320C6211 ainsi que tout ce qui est en relation avec sa programmation dans le cas de notre application. Dans la seconde, nous exposerons les outils nécessaires pour le traitement de signal avec le DSP. Voir le Code Composer Studio (CCS) qui fournit un environnement de développement intégré (IDE), le DSP Starter Kit (DSK) avec le processeur TMS320C211 à virgule fixe et le Link for Code Composer Studio de MATLAB.

3.1 INTRODUCTION

De nos jours, Le DSP (Digital Signal Processor) présente une puissance importante de calcul et d'implémentation des techniques numériques de traitement du signal, cette puissance est assurée par sa grande vitesse d'exécution, ses fonctions spéciales et son jeu d'instructions optimisé pour le traitement numérique du signal. En effet, tout système fondé autour d'un DSP bénéficie des avantages dérivant de ses particularités architecturales et de programmation; dont on peut citer : Le temps réel, la flexibilité, la fiabilité et la réduction des coûts.

3.2 ETUDE DE LA FAMILLE C6000

Il existe plusieurs fabricants de DSP comme Analog Devices, Motorola et le leader du marché Texas Instruments. Ce dernier a introduit en 1982, le TMS32010 premier membre de ce qui allait devenir la famille de DSP la plus répandue, 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.

A la fin des années quatre vingt, on a introduit le TMS320C30. 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.

Le TMS320C6201 (C62x), présenté en 1997, est le premier membre de la famille C6x des processeurs à virgule fixe (32 bits). Contrairement à ses prédécesseurs de DSP à virgule fixe (16 bits): C1x, C2x et C5x, le C6x est basé sur l'architecture VLIW (Very-Long-Instruction-Word) qui est caractérisée par la séparation entre mémoire données et mémoire programme comme l'architecture Harvard, il n'a pas un code compatible aux leurs et il présente une amélioration de la dynamique et de la précision.

Par la suite, TI a introduit le TMS320C6701 (C67x), un processeur à virgule flottante qui a un jeu d'instructions basé sur celui du C62x. 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.

En outre, le DSP virgule fixe consomme moins d'énergie que son équivalent en virgule flottante, du coup il est conseillé pour les applications qui utilisent des batteries. **[36]**

3.2.1 ARCHITECTURE DU TMS320C6x

Le TMS320C6211 mis en œuvre sur la carte DSK, notre support de développement d'applications par la suite, est un processeur à virgule fixe basé sur l'architecture VLIW qui le rend un excellent choix pour les applications multicanaux et multitâches.

Le tableau 3.1 nous donne un résumé des différentes caractéristiques du TMS320C6211. **[35]**

Tableau 3.1 : Caractéristique du TMS320C6211

HARDWARE FEATURES		C6211 (FIXED-POINT DSP)
Peripherals	EMIF (Clock source = ECLKIN)	1
	EDMA (Internal clock source = CPU clock frequency)	1
	HPI	1
	McBSPs (Internal clock source = CPU/2 clock frequency)	2
	32-Bit Timers (Internal clock source = CPU/4 clock frequency)	2
On-Chip Memory	Size (Bytes)	72K
	Organization	4K-Byte (4KB) L1 Program (L1P) Cache 4KB L1 Data (L1D) Cache 64KB Unified Mapped RAM/Cache (L2)
CPU ID+ CPU Rev ID	Control Status Register (CSR.[31:16])	0x0002
Frequency	MHz	167, 150
Cycle Time	ns	6 ns (C6211-167) 6.7 ns (C6211-150)
Voltage	Core (V)	1.8
	I/O (V)	3.3
PLL Options	CLKIN frequency multiplier	Bypass (x1), x4
BGA Packages	27 x 27 mm	256-Pin BGA (GFN)
Process Technology	µm	0.18 µm
Product Status Product Preview (PP) Advance Information (AI) Production Data (PD)		PD

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. **[35]**

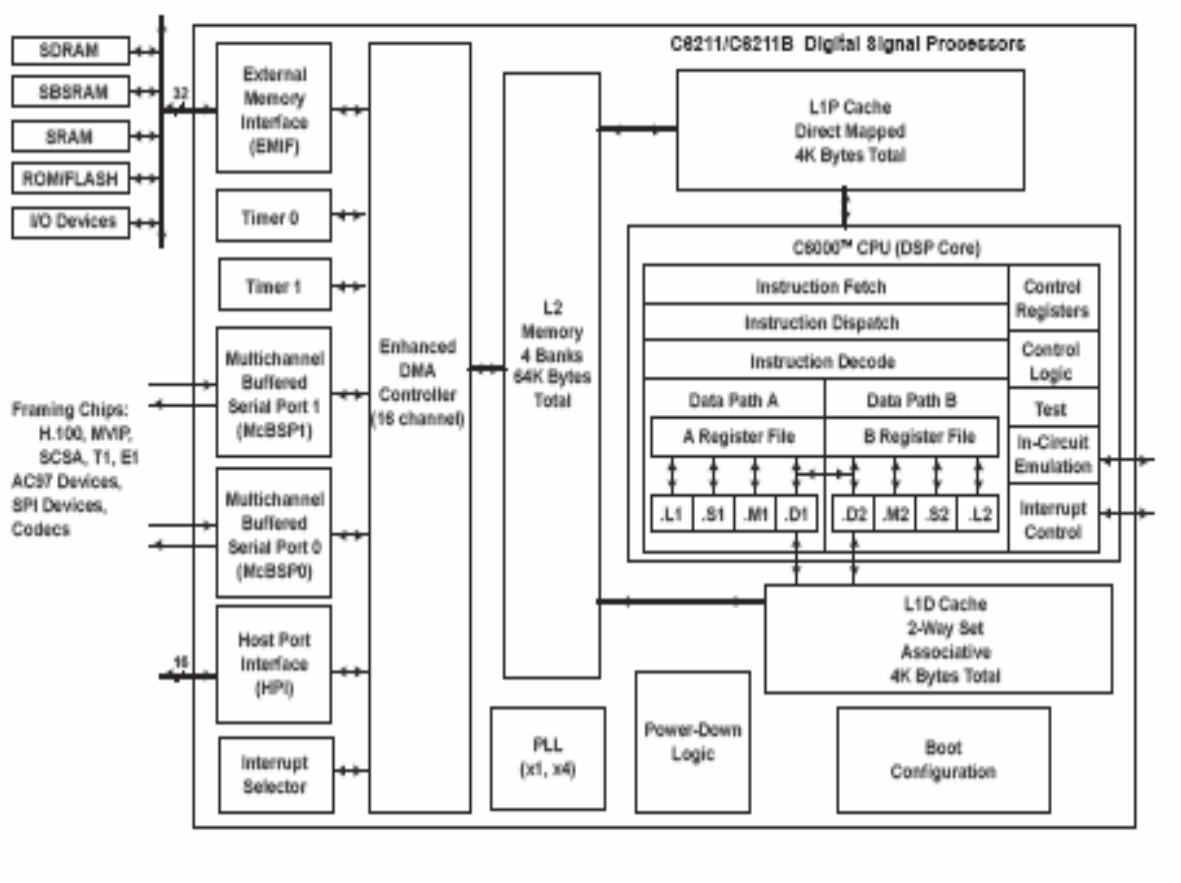


Figure 3.1 : Block Diagram du TMS320C6211

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

Le C6211 du DSK inclut 72kB de mémoire interne adressable à partir de l'adresse 0x00000000, et 16MB de SDRAM adressable via CE0 à partir de l'adresse 0x80000000. Le DSK inclut aussi une mémoire flash de 128kB à partir de 0x90000000. Le tableau 3.2 montre la configuration mémoire du DSP C6211. **[25] [27] [28][36]**

Tableau 3.2 : Résumé de la Configuration Mémoire du TMS320C6211

MEMORY BLOCK DESCRIPTION	BLOCK SIZE (BYTES)	HEX ADDRESS RANGE
Internal RAM (L2)	64K	0000 0000 – 0000 FFFF
Reserved	24M – 64K	0001 0000 – 017F FFFF
External Memory Interface (EMIF) Registers	256K	0180 0000 – 0183 FFFF
L2 Registers	256K	0184 0000 – 0187 FFFF
HPI Registers	256K	0188 0000 – 018B FFFF
McBSP 0 Registers	256K	018C 0000 – 018F FFFF
McBSP 1 Registers	256K	0190 0000 – 0193 FFFF
Timer 0 Registers	256K	0194 0000 – 0197 FFFF
Timer 1 Registers	256K	0198 0000 – 019B FFFF
Interrupt Selector Registers	256K	019C 0000 – 019F FFFF
EDMA RAM and EDMA Registers	256K	01A0 0000 – 01A3 FFFF
Reserved	6M – 256K	01A4 0000 – 01FF FFFF
QDMA Registers	52	0200 0000 – 0200 0033
Reserved	736M – 52	0200 0034 – 2FFF FFFF
McBSP 0/1 Data	256M	3000 0000 – 3FFF FFFF
Reserved	1G	4000 0000 – 7FFF FFFF
EMIF CE0†	256M	8000 0000 – 8FFF FFFF
EMIF CE1†	256M	9000 0000 – 9FFF FFFF
EMIF CE2†	256M	A000 0000 – AFFF FFFF
EMIF CE3†	256M	B000 0000 – BFFF FFFF
Reserved	1G	C000 0000 – FFFF FFFF

3.2.2 UNITES FONCTIONNELLES :

Le CPU consiste en deux chemins de données A et B, chaque chemin comporte quatre unités fonctionnelles indépendantes : (voir figure 3.1)

- Une unité pour la multiplication (.M).
- Une unité pour les opérations logiques et arithmétiques (.L).
- Une unité pour les branchements, la manipulation des bits et opérations arithmétiques (.S).
- Une unité 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. **[31] [36]**

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)

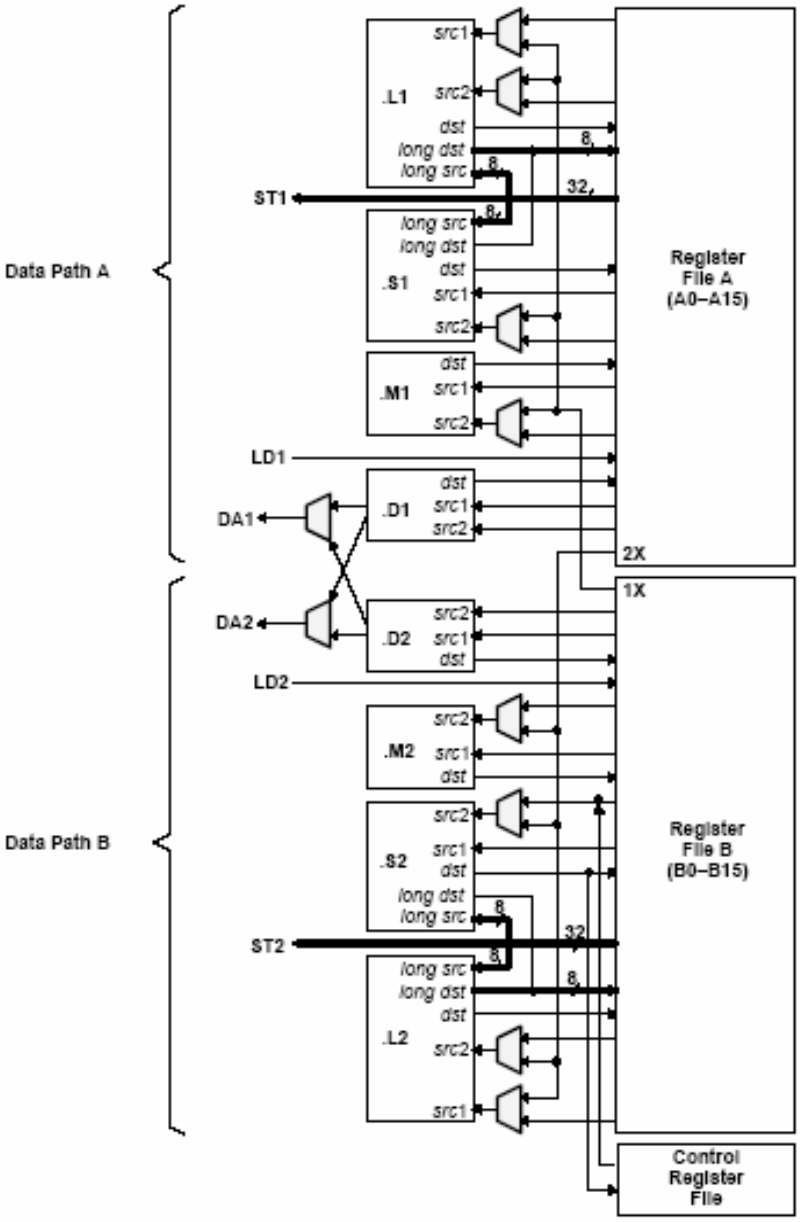


Figure 3.2 : Chemins de données TMS320C62x

3.2.3 LES REGISTRES

Deux fichiers contenant chacun un ensemble 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'un sous programme.

Une donnée sur 40bits peut être rangée dans une paire de registres. Les 32bits de poids faible sont stockés dans les registres à indice pair (c.à.d A2) et les 8bits qui restent sont stockés dans les 8LSBs du registre impair suivant (c.à.d A3).

Ces 32 registres sont considérés comme des 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. **[25] [31] [36]**

3.2.4 LES 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. **[31]**

3.2.5 LES McBSP

Deux multichannels buffered serial ports (McBSP) sont disponibles servant d'interfaces pour les périphériques standard. 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. **[25] [31]**

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. **[37]**

On trouve d'autres registres comme le registre de contrôle du port série (SPCR), le registre de contrôle d'émission/réception (XCR/RCR), le registre d'activation des canaux d'émission/réception (XCER/RCER) et le registre de contrôle de broche (PCR). **[36]**

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.

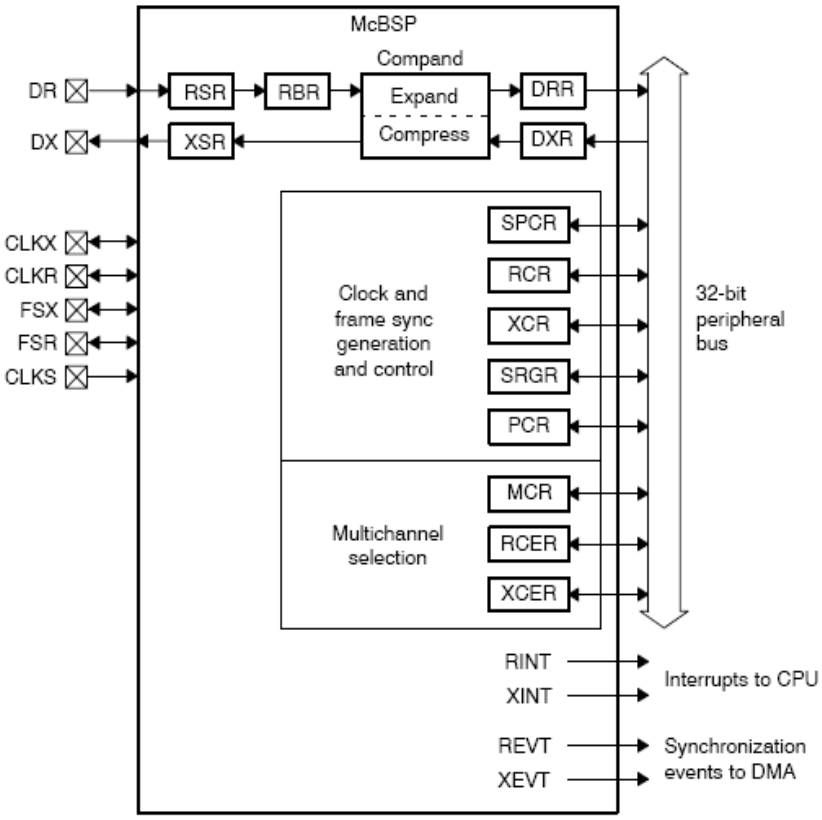


Figure 3.3 : Schéma interne du McBSP

3.2.6 L'ACCES MEMOIRE DIRECT:

L'accès direct mémoire ou DMA (acronyme anglais de Direct Memory Access) est un transfert de données vers et de la mémoire interne ou des périphériques externes sans intervention du CPU si ce n'est pour initier et conclure le transfert. La conclusion du transfert ou la disponibilité du périphérique peuvent être signalés par interruption.

Le DMA est donc nécessaire pour soulager le coeur du DSP et conserver la fluidité d'utilisation d'un système multi-tâches lors de l'accès à des périphériques rapides, comme les disques durs. En effet, en l'absence de DMA, le système est presque bloqué pendant les transferts de données. Par ailleurs, pour des périphériques rapides, il est impossible de transférer chaque donnée par une interruption. Une alternative est que le périphérique ait une mémoire tampon partagée avec le système, dont le remplissage est signalé par une interruption.

En effet 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. **[36]**

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. **[25] [31]**

3.2.7 LES INTERRUPTIONS

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 tâche 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. **[25] [31]**

3.2.7. 1 Les registres d'interruptions

Les registres de contrôle d'interruptions sont les suivants **[26]:**

- CSR (control status register) : contient le bit (GIE : global interrupt enable) et autres bits de contrôle et d'état, figure 3.4

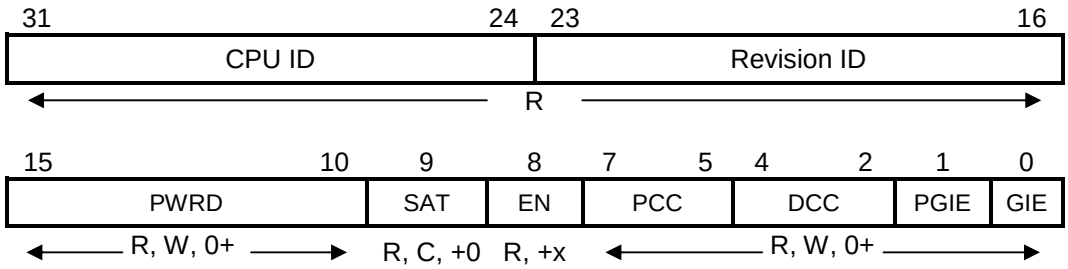


Figure 3.4 : Registre de contrôle et d'état

- IER (interrupt enable register) : pour activer/désactiver des interruptions, figure 3.5

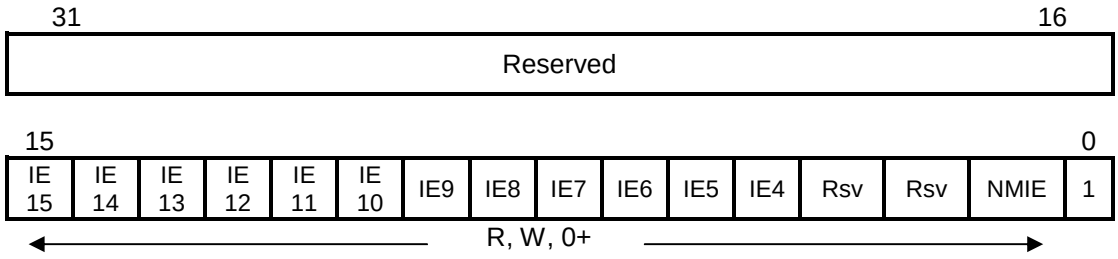


Figure 3.5 : Registre d'activation d'interruptions

- IFR (interrupt flag register) : montre l'état des interruptions.
- ISR (interrupt set register) : permet de positionner un bit dans l'IFR manuellement.
- ICR (interrupt clear register) : permet d'effacer un bit dans l'IFR manuellement.
- ISTP (interrupt service table pointer) : pour localiser l'IST, il se pointe au début du tableau service d'interruption (figure 3.6).

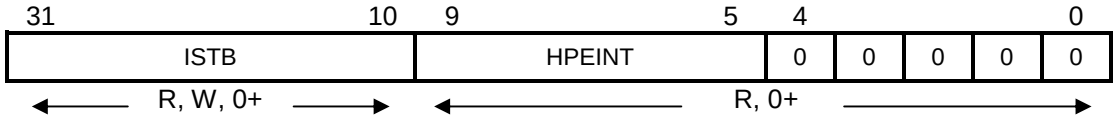


Figure 3.6 : Pointeur du tableau de service d'interruptions

- IRP (interrupt return pointer) : porte l'adresse de retour d'une interruption masquable.
- NRP (no maskable 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 3.3).

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 3.3 : 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 3.3) 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 3.4 montre les valeurs qui peuvent être sélectionnées pour choisir un type spécifique d'interruption.

Tableau 3.4 : 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

3.2.7.2. Reconnaissance d'interruption

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 cours de 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.

3.2.8 EXECUTE et FETCH 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. **[25] [29]**

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 3.7. 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.

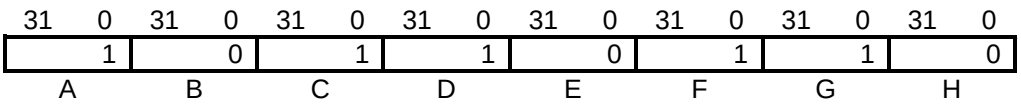


Figure 3.7 : Un FP avec trois EPs, montrant le bit 'p' de chaque instruction

3.2.9 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 buffer (lecture 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 3.5 représente les phases de pipeline, et le tableau 3.6 montre l'effet du pipeline. La première rangée (Tableau 3.6) 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 3.5 : 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 3.6 : 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. **[25] [29 [36]**

]

3.2.10 LES MODES D’ADRESSAGE

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. **[29]**

3.2.10.1 L’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 pointer les adresses mémoires. A proprement parler, ces registres sont des pointeurs. Le mode d'adressage indirect utilise le symbole " * " 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 pré incrémentée d'un déplacement d, telle que l'adresse courante est R+d. Un double ' - ' introduit une pré décrémentation de l'adresse

mémoire telle que la courante sera R-d.

4. *+R(d). L'adresse est pré incrémentée par d telle que l'adresse courante est R+d. Cependant, dans ce cas, R est pré incrémenté sans modification. Contrairement au cas précédent, R n'est pas mis à jour ou modifié. **[36]**

3.2.10.2 L'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. Citons comme Exemples, les filtres numériques et les 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.

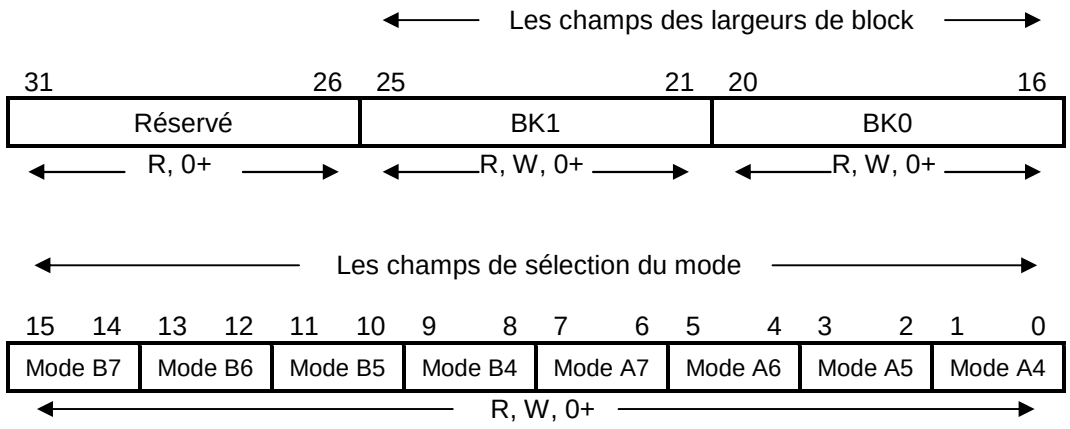


Figure 3.8 : Registre des modes d'adressage

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 créée 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 prend 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 3.7 : 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

3.2.11 LE JEU D'INSTRUCTIONS DU TMS320C6x

3.2.11.1 Format du code assembleur

Un code en assembleur C6x est représenté par le champ suivant : **[27] [36]**

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

Un label (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 égal 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.

3.2.11.2 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. **[29] [36]**

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.

3.2.12 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) génère 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. **[25] [36]**

3.2.13 CONSIDERATIONS MEMOIRE

3.2.13.1 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 **[29]**. 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 variables 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.

3.2.13.2 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.

3.2.13.3 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. **[34]**

3.2.13.4 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. **[25] [34]**

3.2.14 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. **[29] [36]**

3.2.15 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. **[25] [30]**

3.2.15.1 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).

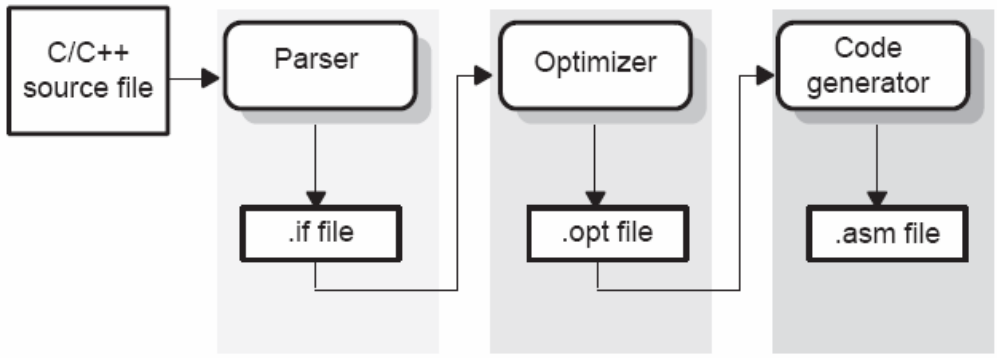


Figure 3.9 Compilation d'un programme C/C++ avec optimisations

Les options sont, en résumé :

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.

3.2.15.2 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 _mpyh()` : 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. [33]

3.3 OUTILS DE DEVELOPPEMENT

TI offre une ligne étendue des outils de développement pour le TMS320C6000, y compris des outils pour évaluer l'exécution des processeurs, la génération de code et le développement d'algorithmes d'implémentation, ainsi que la gestion du software et du hardware.

Les produits suivants soutiennent le développement d'application sur le TMS320C6000 : **[35]**

- Le développement Software :

Dans cette catégorie, on a le Code Composer Studio, le compilateur C/C++/ et l'Assembleur, ainsi que le Real Time Data eXchange (RTDX) et DSP/BIOS pour le transfert de données entre un ordinateur hôte et les périphériques du DSP.

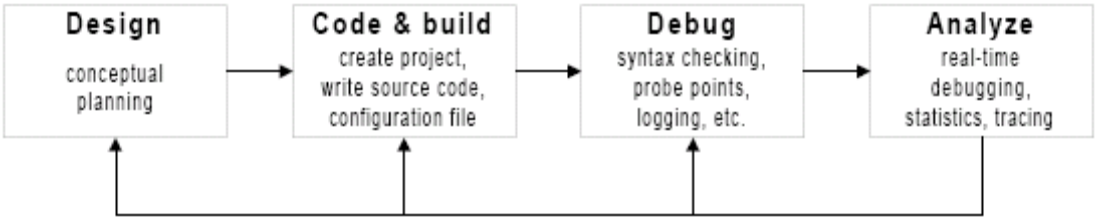
- Le développement Hardware :

On trouvera dans cette catégorie, l'émulateur XDS (Extended Development System) et EVM (Evaluation Module)

3.3.1 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.



3.3.1.1Outils de génération du code

Le Code Composer Studio offre une interface graphique adéquate pour l'utilisation des outils de génération du code. Il garde en permanence toutes les informations ou les fichiers utilisés pour la création ou la compilation d'un programme ou d'une librairie. La figure suivante montre le processus du développement software.

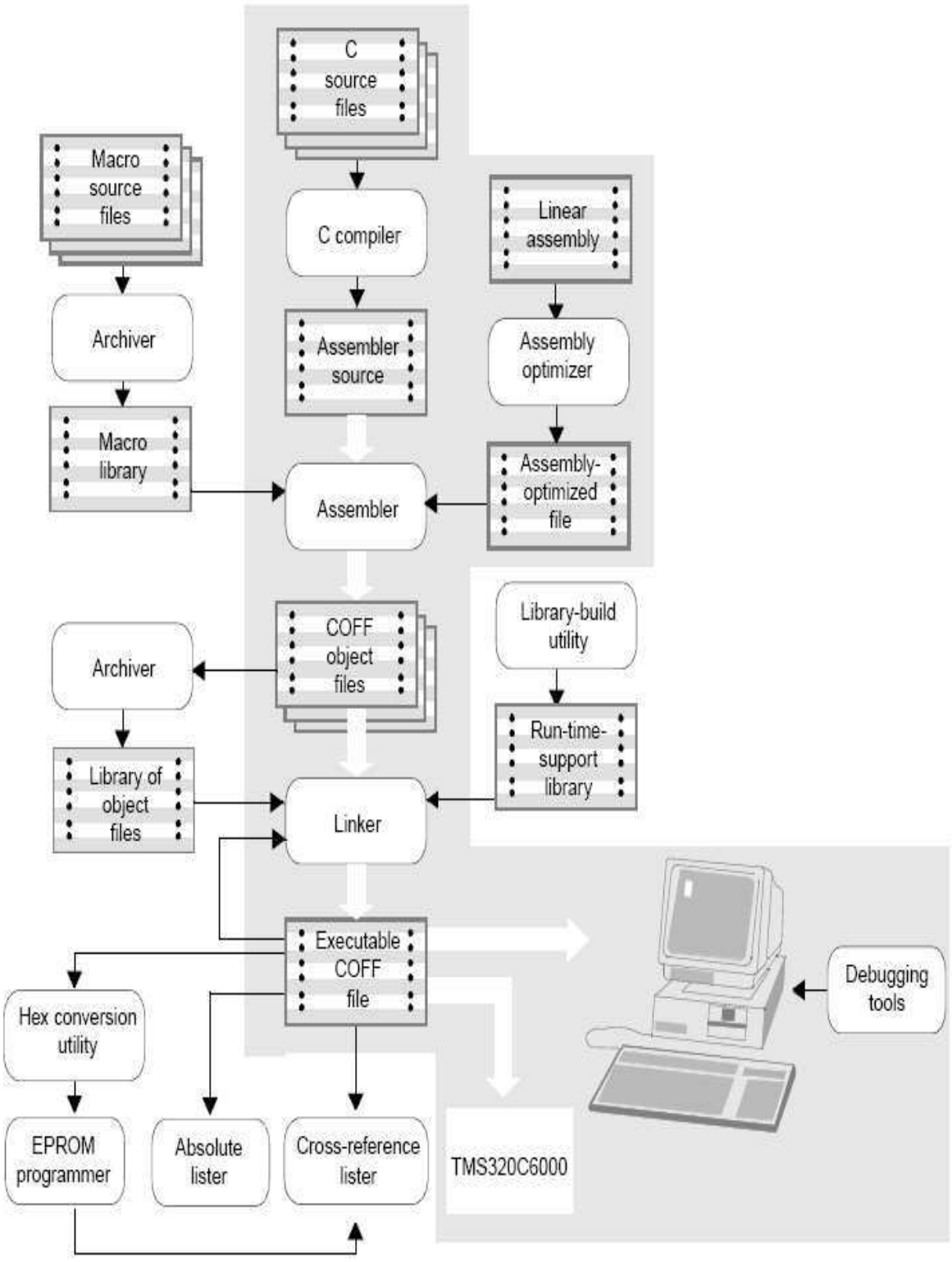


Figure 3.10: Processus du développement du code exécutable

Le CCS nous permet de modifier les options de compilation du programme, de l'assemblage et du linker à travers une boite de dialogue « Build Options ». (voir figure 3.11)

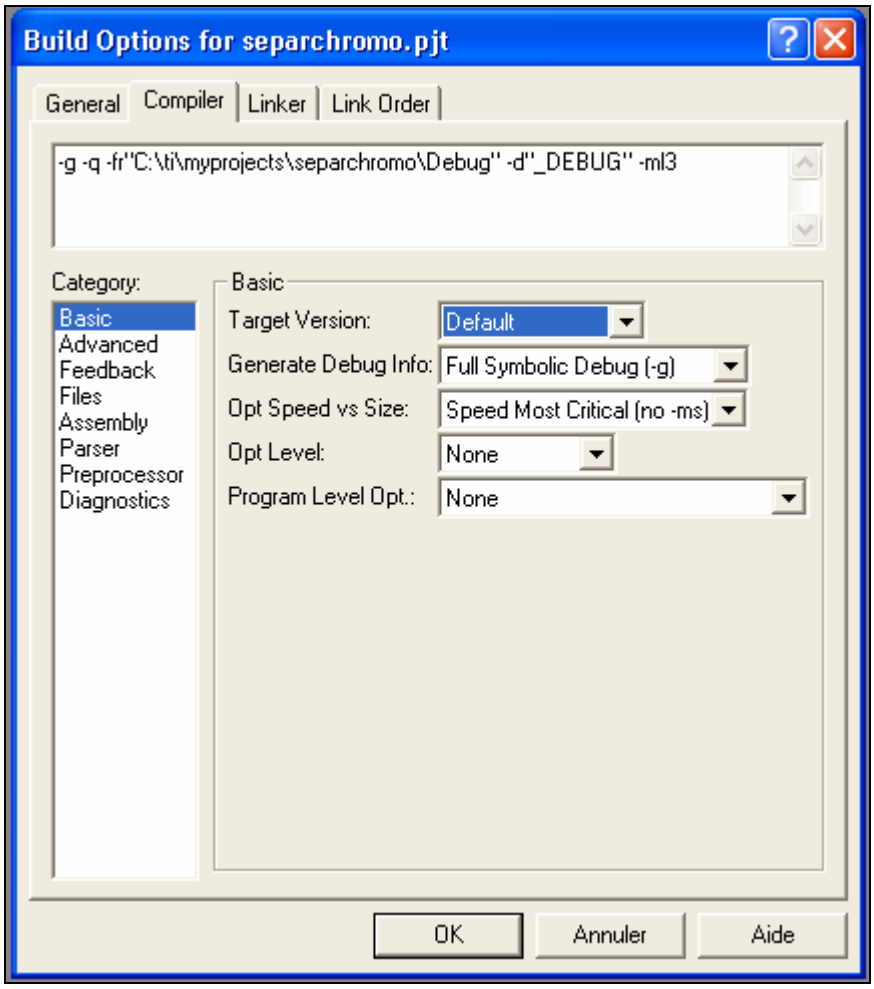


Figure 3.11 : Boite de dialogue « Build Options »

- ## Outils de développement pour l'assembleur

Les outils de développement de langage d'assemblage sont les suivants : **[34]**

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.

- ## Outils de développement pour C/C++

Les outils de développement pour C/C++ sont : **[34]**

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.

3.3.1.2 Graphe d'optimisation

En suivant le graphe de la figure 3.13, on peut atteindre les meilleures performances en matière de parallélisme d'exécution, et optimisation de la taille du code. **[32]**

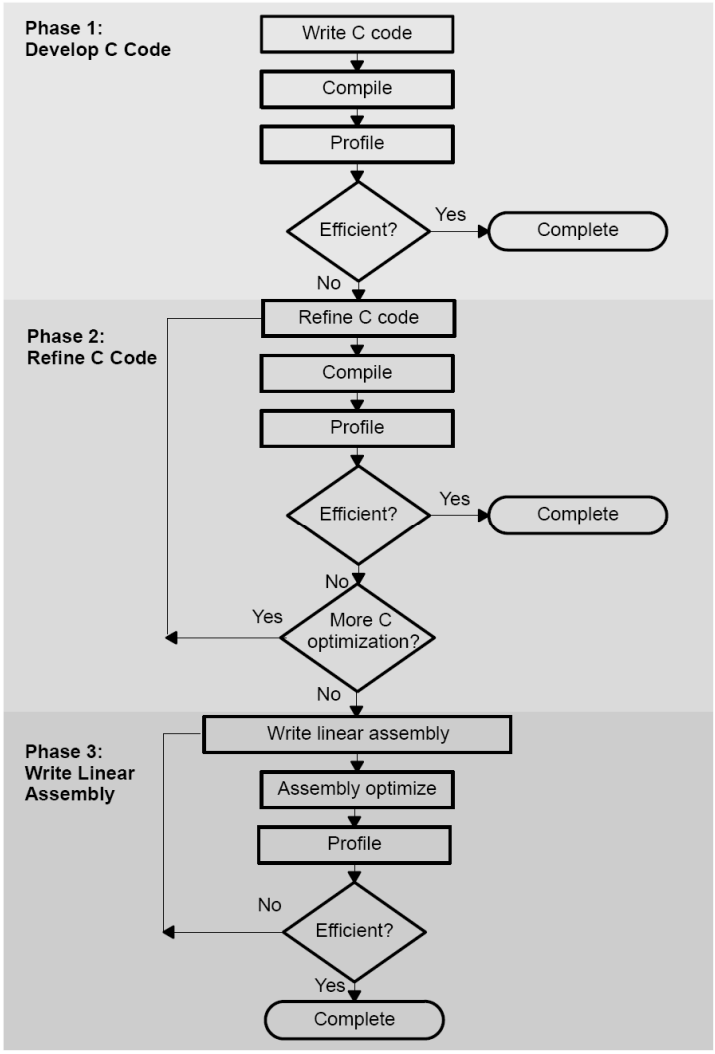


Figure 3.12 Graphe d'optimisation

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, 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.

3.3.1.3 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 [32]:

Les points d'arrêt (Break points) : 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 break points 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 débuguant 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++.

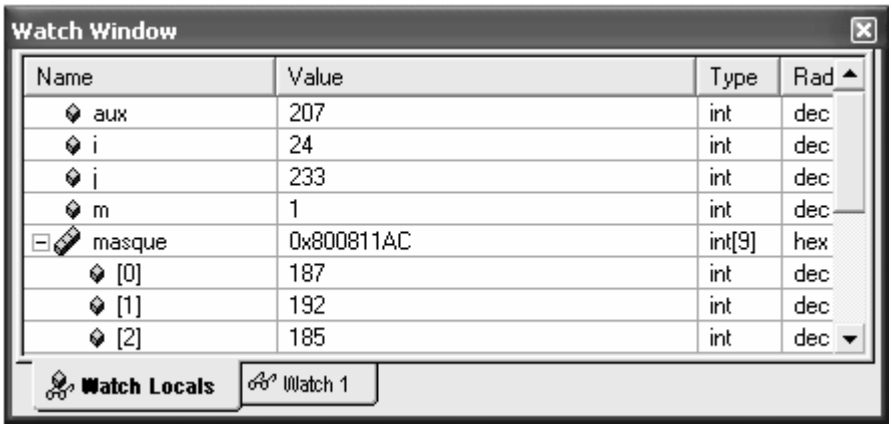


Figure 3.13 Watch windows

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 du break points par le fait qu'ils n'introduisent pas un arrêt permanant (lancement manuel après arrêt) de l'exécution du programme mais permettent 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 probes 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 extension 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 aux besoins du concepteur. (figure 3.15)

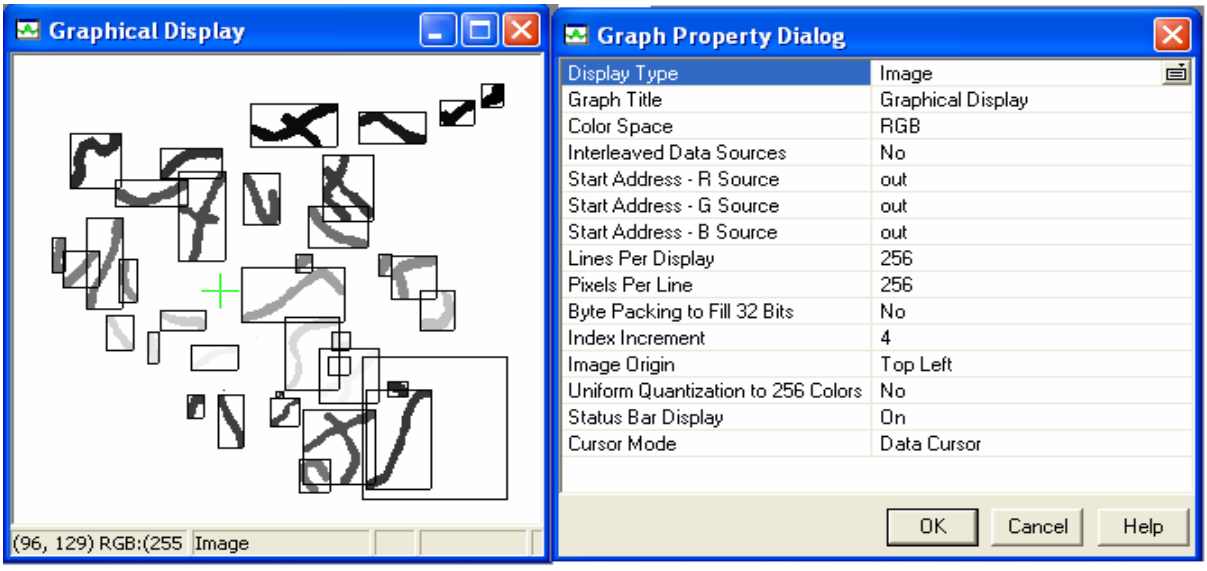


Figure 3.14 Exemple d'affichage graphique

3.3.1.4 IMGLIB

La C62x IMGLIB est une bibliothèque optimisée de fonctions de traitement d'Image/Vidéo pour des programmeurs en C à l'aide des dispositifs de TMS320C62x. Elle inclut beaucoup de routines de traitement d'image optimisée.

Ces routines sont typiquement utilisées dans des applications en temps réel. En employant ces routines, on peut réaliser un code équivalent avec une vitesse d'exécution considérablement plus rapidement.

3.3.1.5 Gestion de projet

Avant de commencer tout développement, il est nécessaire de créer un nouveau projet. Ce dernier peut être une librairie ou une application exécutable.

Il faut tout d'abord créer le fichier de gestion du projet (extension .pjt). Pour ce faire, on choisit **New** dans le menu **Project**, la fenêtre suivante va apparaître **(figure 3.16).**

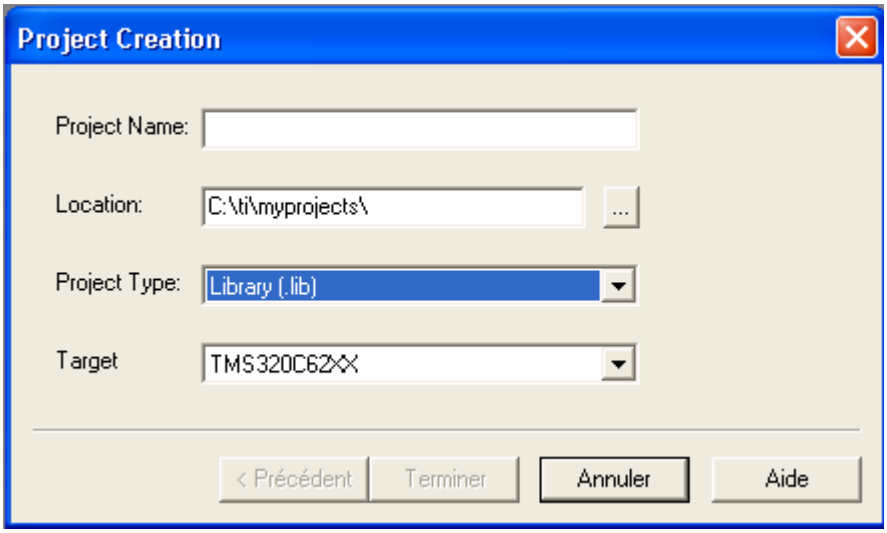


Figure 3.15: Création de projet

Dans le champ « Project Type » on fait rentrer le type de notre projet, Library (.lib) pour la création d'une bibliothèque et Executable (.out) pour la création d'une application exécutable.

Dans le champ « Target » on fait le choix de la famille utilisée pour notre application. Et les champs « Project Name » et « Location » servent à nommer et à localiser notre projet.

- **Création d'une librairie :**

Une fois le fichier de gestion de projet créé, on ajoute les fichiers sources (*.c) et la librairie utilisée (*.lib) en utilisant la commande « Add Files to Project » dans le menu « Project » à chaque fois. Ensuite on utiliser la commande « Rebuild all » pour construire le projet (extension .lib).

- **Création d'une application exécutable :**

Dans ce type de projet, on peut ajouter les fichiers sources (*.c), les librairies (.lib) qui peuvent être aussi d'autres projets de type librairie ainsi que le fichier commande (*.cmd)

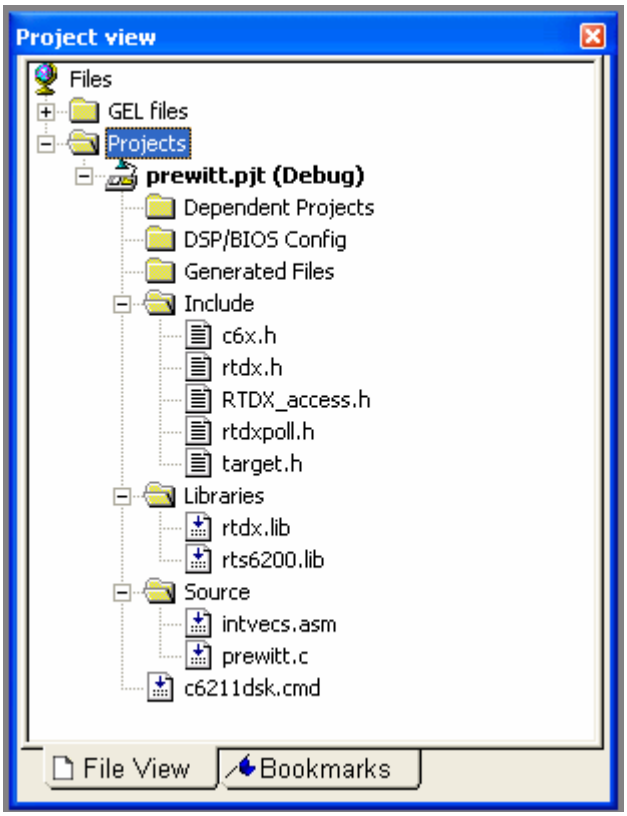


Figure 3.16 : projet créé

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 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'erreurs, 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 commande **halt**.

3.3.2 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.

3.3.3 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 3.18). **[34] [36] [37]**

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 du MATLAB pour réaliser un script et la visualisation des données a été faite directement sous MATLAB.

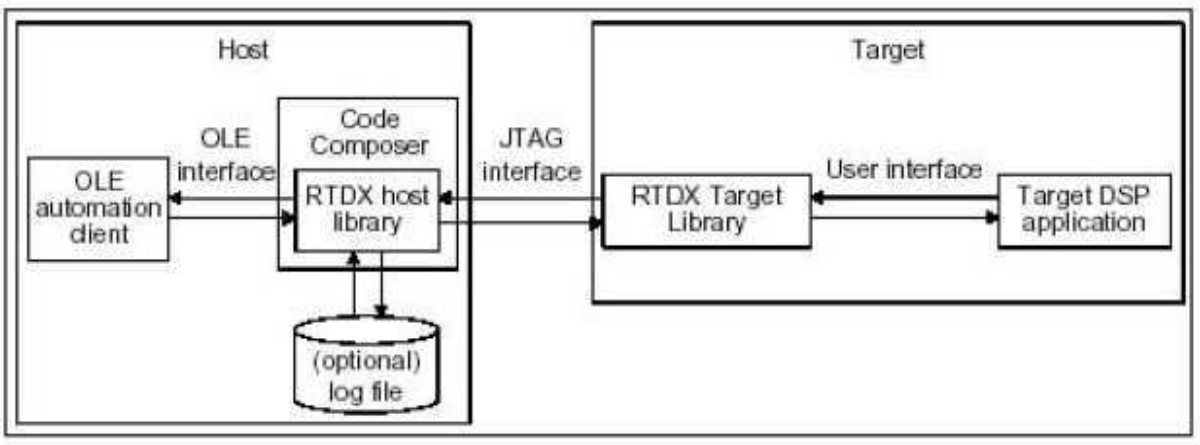


Figure 3.17 Connexion entre hôte et cible via JTAG

3.3.3.1 Communication PC-Cible

Dans la communication PC-Cible, le canal d'entrée doit être configuré. La cible envoie une demande de réception de données en utilisant les routines définies lors de la programmation. 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 (Figure 3.19).**[35] [36]**

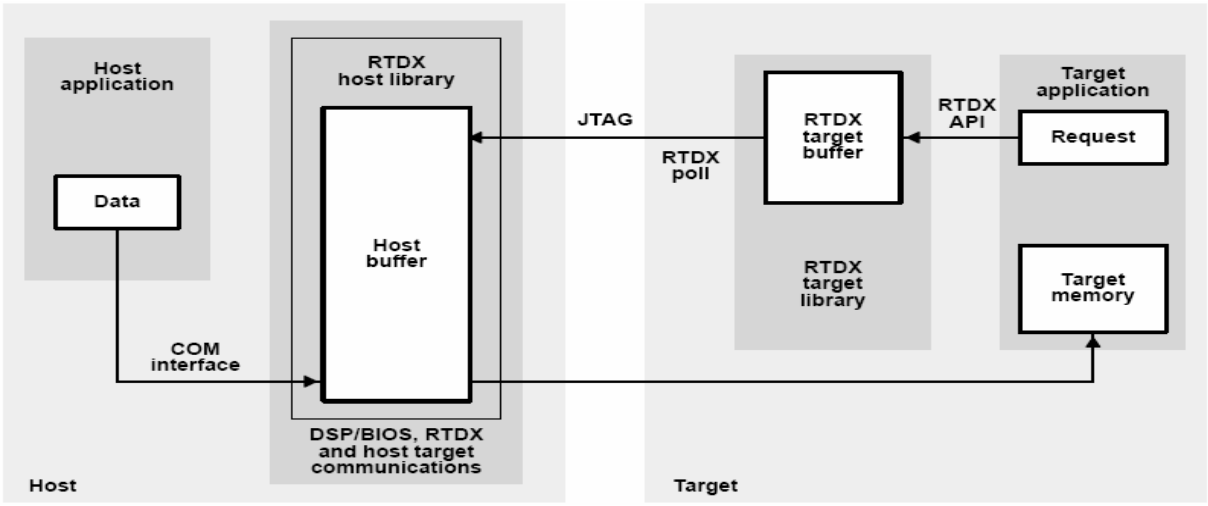


Figure 3.18 Etablissement de communication PC-Cible

3.3.3.2 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 3.20).**[35] [36]**

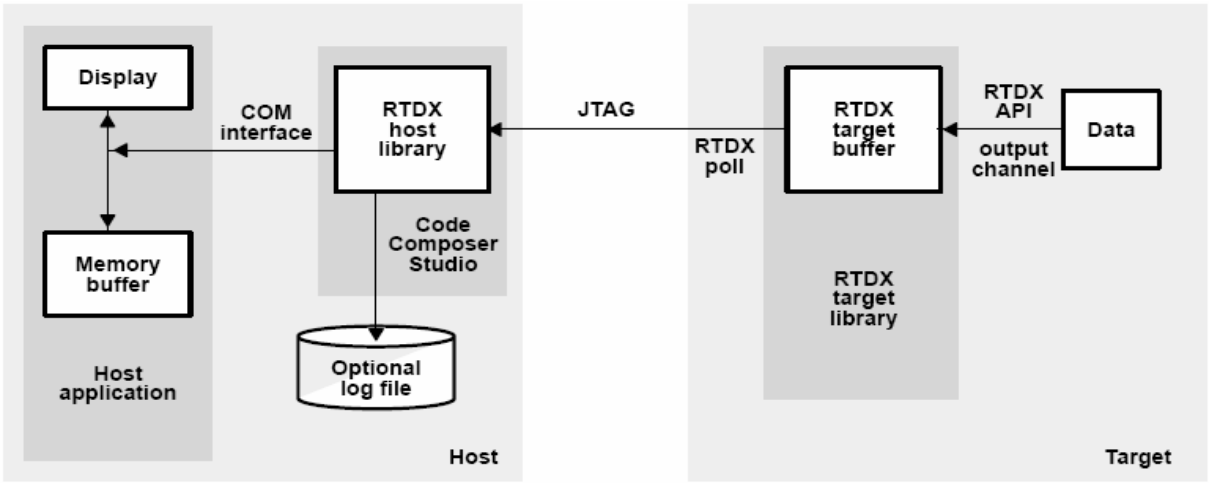


Figure 3.19 Etablissement de communication Cible-PC

3.3.3.3 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 fenêtres suivantes :

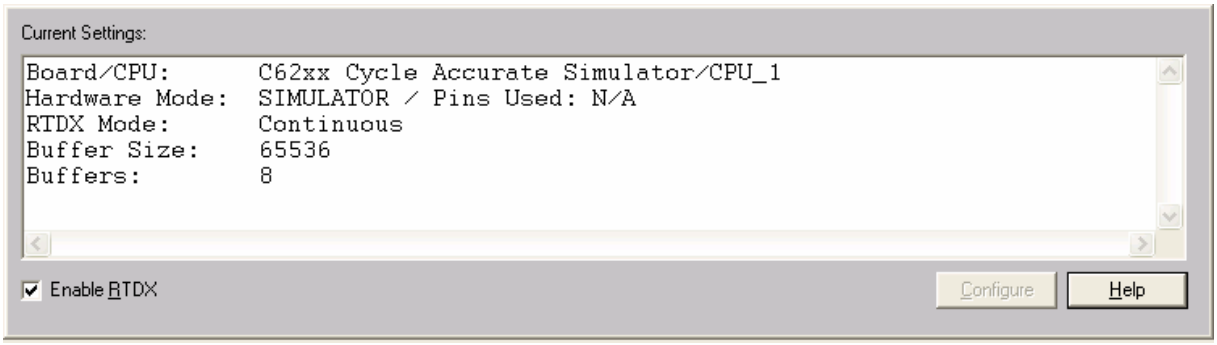


Figure 3.20 : Diagnostics Control

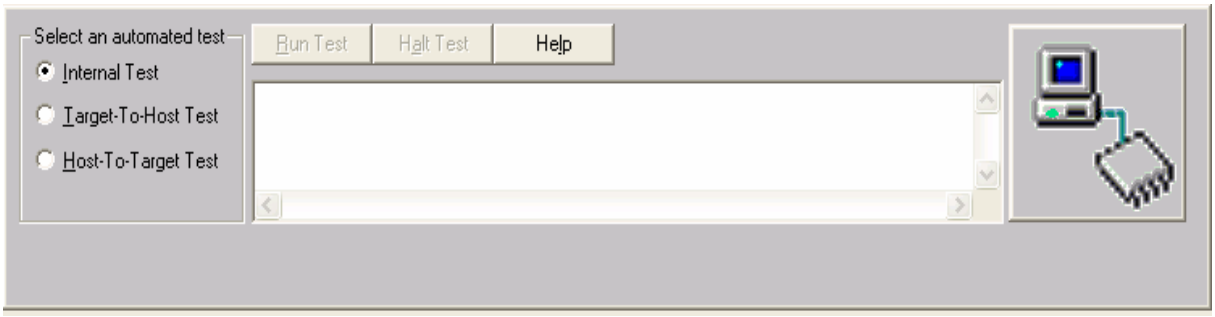


Figure 3.21 : Configuration Control

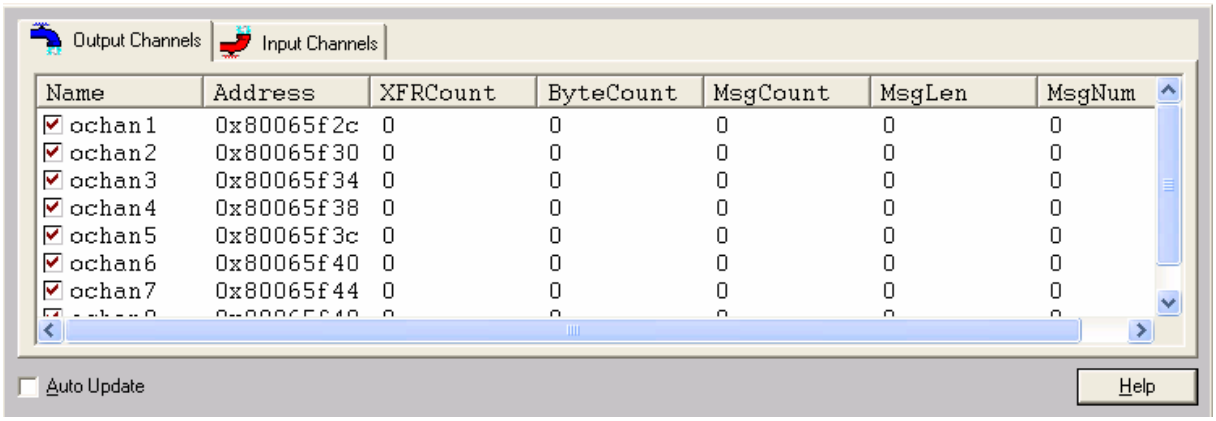


Figure 3.22 : Channel Viewer Control

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'informations, se référer au Help du code composer.

3.3.4 LINK FOR CODE COMPOSER

Pour communiquer avec le Code Composer Studio, nous avons choisi le **MATLAB SP2** pour créer un script au lieu de **Visual C++**,**Visual Basic** ou le **Lab View** qui sont sans doute plus performants, cela est dû au temps de développement nécessaire qui n'était pas à notre portée. Ce script est considéré comme application client pour l'ensemble des entités formant le système de traitement d'images que nous avons élaboré.

Pour ce faire, nous avons utilisé au sein du logiciel **MATLAB** une de 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 registres de la cible.

Le **Link for Code Composer Studio** nous donne la possibilité de choisir la cible sur laquelle on veut travailler, de créer un projet sous **CCS**, d'ajouter les fichier nécessaires à sa compilation et de l'exécuter. (Voir le help du MATLAB pour plus de détails)

Comme le **RTDX** fait partie du **CCS**, le **Link for Code Composer Studio** nous permet d'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 et enfin visualiser les résultats.

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'invité de commande est :

```
CCSDSP Object:
    API version       : 1.2
    Processor type    : TMS320C6711
    Processor name    : CPU_1
    Running?          : No
    Board number      : 1
    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 les fonctions nécessaires utilisées dans ce cadre.

3.4 Conclusion

Dans ce chapitre, nous avons fait une brève étude sur la famille **C6000** et 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 le script assisté par le **Link for Code Composer Studio** afin d'établir la communication **RTDX** avec la cible **DSK** et visualiser les images traitées.

Chapitre IV
Application et Présentation des résultats

Ce chapitre est destiné à présenter les différents algorithmes que nous avons implémentés en vue de la réalisation de notre application qui a pour but l'implémentation d'un algorithme de séparation des chromosomes qui se touchent ou qui se chevauchent à partir d'une image de mitose. Nous exposerons aussi dans ce chapitre les quatre solutions RTDX utilisées afin d'établir une communication avec la carte DSK.

4.1 INTRODUCTION

Dans ce chapitre, nous utiliserons toutes les notions déjà vues dans les trois chapitres pour effectuer des liaisons entre les différentes composantes régissant les exigences de notre application.

Nous présenterons en détails les différentes étapes du développement, ainsi que les problèmes rencontrés avec les solutions apportées d'une part, d'autre part nous expliciterons l'algorithme de séparation des chromosomes se touchant ou qui se chevauchant.

4.2 PRISE EN CHARGE DE LA CARTE

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.
Il est nécessaire, pour l'utilisation de la carte, de configurer le CCS pour la prendre en charge, et cela moyennant le **Setup**, (voir figure 4.1).

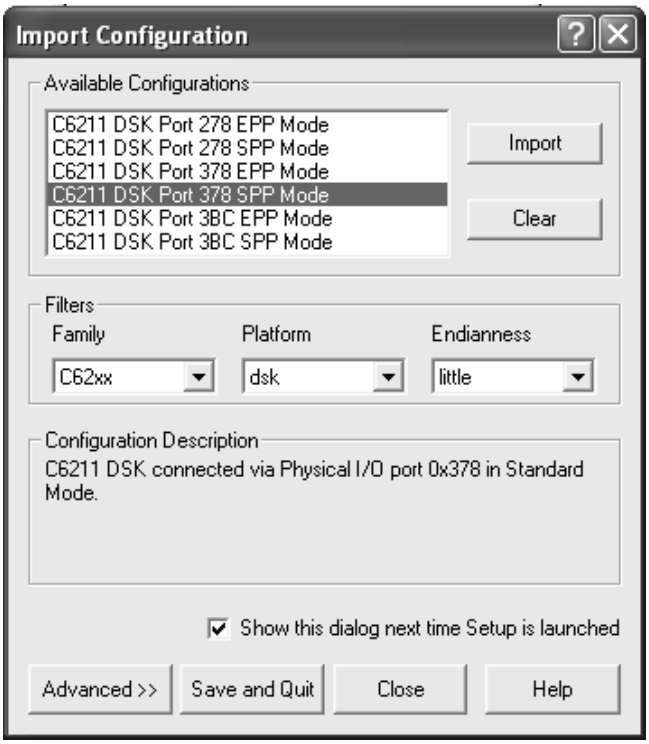


Figure 4.1 Fenêtre de configuration

4.3 SOUS CODE COMPOSER

Nous procédons tout d'abord à la création des projets qui vont servir comme exemples pour la démonstration des solutions RTDX proposées.

Soit le projet désigné par **marthon_vec1.pjt** tel que présenté dans le chapitre 3. La cible choisie sera la famille C62xx. Le fichier généré à la fin est exécutable (*.out).

Les fichiers nécessaires à la création de notre projet sont :

`marthon.cpp` : code source d'un algorithme de squelettisation.

`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.

`rtdxsim.lib` : librairie contenant des fonctions permettant d'établir la communication et la transmission des données, utilisée lors de la simulation

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

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

#define SIZE 65536
#define SZ 8192
#define LINE 256
#define pxLINE 256
#define seuil 205
#define CONNECTIVITY ((b1-(b1&b3))+(b3-(b3&b5))+(b5-(b5&b7))+(b7-(b7&b1))-(b2&

short buf_in[SIZE],m_pDmy[SIZE],m_pBfr[SIZE],bDone;
short buf_in1[SZ],buf_in2[SZ],buf_in3[SZ],buf_in4[SZ];
short buf_in5[SZ],buf_in6[SZ],buf_in7[SZ],buf_in8[SZ];

static void GetNeighbords(int,int);
static void SkeletCleaner(int,int);

RTDX_CreateInputChannel(ichan); //data transfer PC-->DSK
/* declare 8 global output channels */
RTDX_CreateOutputChannel( ochan1 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan2 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan3 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan4 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan5 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan6 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan7 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan8 );//data transfer DSK-->PC
```

Cette première solution consiste à créer un canal d'entrée qui permet le transfert des données de l'application client vers la carte DSK, et huit canaux de sortie qui servent au transfert des données dans le chemin inverse.

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. 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, deux vecteurs m_pDmy[SIZE],m_pBfr[SIZE] pour des traitements intermédiaires et les vecteurs `buf_inx[SZ]` pour les transferts DSK→Host, sont utilisés.

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

La fonction principale du programme est la suivante :

```
int main(void)
{
    int i,j;
    TARGET_INITIALIZE(); //init for interrupt
    while(!RTDX_isInputEnabled(&ichan)) //for MATLAB to enable RTDX
    puts("\n\n Waiting to read "); //while waiting
    RTDX_read(&ichan,buf_in,sizeof(buf_in));//read data by DSK
    puts("\n\n Read Completed");

    #pragma MUST_ITERATE(SIZE,SIZE)
    for (i=0;i<SIZE;i++)
    {
    if (buf_in[i]<=seuil) {buf_in[i]=0;m_pBfr[i]=1;}
    else {buf_in[i]=255;m_pBfr[i]=0;}
    }

    bDone=false;
    while(!bDone)
    {
    #pragma MUST_ITERATE(SIZE,SIZE)
    for (i=0;i<(SIZE-1);i++) {m_pDmy[i]=m_pBfr[i];}
    bDone=true;
    #pragma MUST_ITERATE(LINE,LINE)
    for(i=0;i<(LINE-1);i++)
    #pragma MUST_ITERATE(pxLINE,pxLINE)
    for(j=0;j<(pxLINE-1);j++)
    {
    if ((i!=0) && ((i+1)!=LINE) && (j!=0) && ((j+1)!=pxLINE))
    {
    if ( m_pBfr[i*LINE+j]==1) GetNeighbords(i,j);
```

Dans cette partie du programme, la déclaration des variables locales 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 le script créé 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.

Les directives pragma abordées au chapitre III sont destinées au compilateur, elles aident 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. Ces instructions n'ont pas été utilisées dans ce cas.

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 cette tâche :

```
#pragma MUST_ITERATE(SZ,SZ)
for (i=0;i<SZ;i++) buf_in1[i]=buf_in[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ;i<SZ*2;i++) buf_in2[i-SZ]=buf_in[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*2;i<SZ*3;i++) buf_in3[i-SZ*2]=buf_in[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*3;i<SZ*4;i++) buf_in4[i-SZ*3]=buf_in[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*4;i<SZ*5;i++) buf_in5[i-SZ*4]=buf_in[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*5;i<SZ*6;i++) buf_in6[i-SZ*5]=buf_in[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*6;i<SZ*7;i++) buf_in7[i-SZ*6]=buf_in[i];
#pragma MUST_ITERATE(SZ,SZ)
for (i=SZ*7;i<SZ*8;i++) buf_in8[i-SZ*7]=buf_in[i];

while(!RTDX_isOutputEnabled(&ochan1)) //for MATLAB to enable RTDX
puts("\n\n Waiting to ochan1"); //while waiting
RTDX_write(&ochan1,buf_in1,sizeof(buf_in1));//send data from DSK to PC
puts("\n\n Write 1 Completed");

while(!RTDX_isOutputEnabled(&ochan2)) //for MATLAB to enable RTDX
puts("\n\n Waiting to ochan2"); //while waiting
RTDX_write(&ochan2,buf_in2,sizeof(buf_in2));//send data from DSK to PC
puts("\n\n Write 2 Completed");

while(!RTDX_isOutputEnabled(&ochan3)) //for MATLAB to enable RTDX
```

Dans cette partie du programme, nous avons partagé notre image de sortie en huit vecteurs de taille SZ tout en utilisant les directives pragma. La taille SZ dans cette solution ne peut pas être dépassée pour un transfert des vecteurs vers MATLAB. De la même façon que 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.

La deuxième solution proposée, dans le cadre de l'établissement d'une connexion entre l'application client et la carte DSK, consiste à créer un seul vecteur de sortie au lieu de huit. Pour cela un deuxième projet a été créé et nommé **marthon_vec2.pjt.**

Le fichier source devient alors :

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

#define SIZE 65536
#define SZ 8192
#define LINE 256
#define pxLINE 256
#define seuil 205
#define CONNECTIVITY ((b1-(b1&b3))+(b3-(b3&b5))+(b5-(b5&b7))+(b7-(b7&b1)

short buf_in[SIZE],m_pDmy[SIZE],m_pBfr[SIZE],bDone;
short buf_in1[SZ],buf_in2[SZ],buf_in3[SZ],buf_in4[SZ];
short buf_in5[SZ],buf_in6[SZ],buf_in7[SZ],buf_in8[SZ];

static void GetNeighbords(int,int);
static void SkeletCleaner(int,int);

RTDX_CreateInputChannel(ichan); //data transfer PC-->DSK
RTDX_CreateOutputChannel(ochan);//data transfer DSK-->PC
```

Dans cette solution le vecteur de sortie sera activé et désactivé à chaque transfert. Il était indispensable d'effectuer huit transferts de taille SZ chacun. Pour remédier à ce problème nous avons créé une autre solution qui consiste à recevoir des matrices de tailles beaucoup plus importantes que celles utilisées dans la première solution.

Pour cela un troisième projet a été créé et nommé **marthon_mat1.pjt**, dans cette approche nous avons utilisé un canal d'entrée et uniquement quatre canaux de sortie. Ce qui signifie que notre image a été partagé en quatre paquets de taille 16384.

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

#define SIZE 65536
#define SZ 16384
#define LINE 256
#define pxLINE 256
#define seuil 205
#define CONNECTIVITY ((b1-(b1&b3))+(b3-(b3&b5))+(b5-(b5&b7))+(b7-(b7&b

short buf_in[SIZE],m_pDmy[SIZE],m_pBfr[SIZE],bDone;
short buf_in1[SZ],buf_in2[SZ],buf_in3[SZ],buf_in4[SZ];

static void GetNeighbords(int,int);
static void SkeletCleaner(int,int);

RTDX_CreateInputChannel(ichan); //data transfer PC-->DSK
RTDX_CreateOutputChannel(ochan1);//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan2 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan3 );//data transfer DSK-->PC
RTDX_CreateOutputChannel( ochan4 );//data transfer DSK-->PC
```

Nous avons constaté qu'il était possible de garder cette même solution avec un seul vecteur de sortie. Ce canal est activé et désactivé à chaque transfert. Un quatrième projet nommé **marthon_mat2.pjt** a été créé**.**

Le fichier source devient alors :

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

#define SIZE 65536
#define SZ 16384
#define LINE 256
#define pxLINE 256
#define seuil 205
#define CONNECTIVITY ((b1-(b1&b3))+(b3-(b3&b5))+(b5-(b5&b7))+(b7-(b7&

short buf_in[SIZE],m_pDmy[SIZE],m_pBfr[SIZE],bDone;
short buf_in1[SZ],buf_in2[SZ],buf_in3[SZ],buf_in4[SZ];

static void GetNeighbords(int,int);
static void SkeletCleaner(int,int);

RTDX_CreateInputChannel(ichan); //data transfer PC-->DSK
RTDX_CreateOutputChannel(ochan);//data transfer DSK-->PC
```

4.4 SOUS MATLAB

Nous allons maintenant exposer les fonctions, responsables de la communication avec le code composer.

Quatre fonctions (ccsvec1, ccsvec2, ccsmat1 et ccsmat2) ont 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 dur.

```
indata=imread(ifile);
indata=rgb2gray(indata);
indata=indata(1:65536);
ccsboardinfo %board info
cc = ccsdsp('boardnum',0); %set up CCS object
reset(cc) %reset board
visible(cc,1); %for CCS window
```

Cette première partie est commune aux quatre fonctions. Les trois premières lignes sont utilisées pour lire l'image FISH, la transformer en niveau de gris et enfin la mettre sous forme de vecteur. L'image peut être ainsi véhiculée à travers les liaisons logiques et physiques.

Nous introduisons maintenant les instructions du Link for Code Composer 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.

La deuxième partie de notre fonction destinée pour la réception des huit vecteurs de taille identique est la suivante :

```
configure(cc.rtdx,65536,8);
enable(cc.rtdx); %enable RTDX
if ~isenabled(cc.rtdx)
error('RTDX is not enabled')
end
cc.rtdx.set('timeout', 10); %set time out for RTDX
open(cc,x); %open project
load(cc,y); %load executable file
run(cc); %run
```

Dans le cas où nous recevons des matrices de taille 256*64, la commande de configuration des canaux est :

```
configure(cc.rtdx,65536,4);
```

La section suivante va nous permettre d'ouvrir les canaux de sortie nécessaires pour chaque cas, celle de huit canaux de sortie sera comme suit:

```
 %configure four RTDX channels
open(cc.rtdx,'ichan','w'); %open input channel
open(cc.rtdx,'ochan1','r'); %open output channel1
open(cc.rtdx,'ochan2','r'); %open output channel2
open(cc.rtdx,'ochan3','r'); %open output channel3
open(cc.rtdx,'ochan4','r'); %open output channel4
open(cc.rtdx,'ochan5','r'); %open output channel5
open(cc.rtdx,'ochan6','r'); %open output channel6
open(cc.rtdx,'ochan7','r'); %open output channel7
open(cc.rtdx,'ochan8','r'); %open output channel8
%open(cc.rtdx,'ichan','w');
pause(1) %wait for RTDX channel to open
```

Pour le cas de quatre canaux de sortie nous aurons :

```
 %configure four RTDX channels
open(cc.rtdx,'ichan','w'); %open input channel
open(cc.rtdx,'ochan1','r'); %open output channel1
open(cc.rtdx,'ochan2','r'); %open output channel2
open(cc.rtdx,'ochan3','r'); %open output channel3
open(cc.rtdx,'ochan4','r'); %open output channel4
pause(1) %wait for RTDX channel to open
```

Pour les deux cas où nous avons uniquement un seul canal d'entrée et un seul canal de sortie nous aurons:

```
 %configure two RTDX channels
open(cc.rtdx,'ichan','w'); %open input channel
open(cc.rtdx,'ochan','r'); %open output channel
```

Après avoir chargé le fichier.out, l'instruction run(cc) permet d'exécuter le programme présent sur la cible. Nous procédons maintenant au transfert de données ; pour cela un canal d'entrée est activé :

```
%open(cc.rtdx,'ochan1','r');
pause(0.1) %wait for RTDX channel to open
enable(cc.rtdx,'ichan'); %enable channel TO DSK
if isenabled(cc.rtdx,'ichan')
writemsg(cc.rtdx,'ichan', int16(indata)) %send 16-bit data to DSK
pause(1)
else
error('Channel ''ichan'' is not enabled')
end
```

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

```
%open(cc.rtdx,'ochan','r');
enable(cc.rtdx,'ochan1'); %enable channel FROM DSK
if isenabled(cc.rtdx,'ochan1')
outdata1=readmsg(cc.rtdx,'ochan1','int16'); %read 16-bit data from DSK
pause(0.1)
else
error('Channel ''ochan1'' is not enabled')
end
```

Cette section de la fonction sera répétée pour chaque canal de sortie, jusqu'à la réception des huit vecteurs de taille 8192.

Dans le cas des matrices de tailles 256*64, notre fonction sera comme suit :

```
%open(cc.rtdx,'ochan','r');
enable(cc.rtdx,'ochan1'); %enable channel FROM DSK
if isenabled(cc.rtdx,'ochan1')
outdata1=readmat(cc.rtdx,'ochan1','int16',[256 64]); %read 16-bit data from DSK
pause(0.1)
else
error('Channel ''ochan1'' is not enabled')
end
```

Pour les deux cas où un seul vecteur de sortie est créé, nous devons activer puis désactiver le canal de sortie ochan à chaque transfert de données.

Une fois la réception terminée, nous désactivons RTDX, nous fermons les canaux d'entrée et de sortie utilisés lors du transfert de données vers ou de la carte, nous fermons le projet puis nous affichons le résultat du traitement à l'aide de la section suivante :

```
im=[outdata1 outdata2 outdata3 outdata4];
im=double(im);
im=mat2gray(im);
imshow(im);
```

Concernant la solution des huit vecteurs, nous aurons :

```
disable(cc.rtdx); %disable RTDX
close(cc.rtdx,'ichan'); %close input channel
close(cc.rtdx,'ochan1'); %close output channel1
close(cc.rtdx,'ochan2'); %close output channel2
close(cc.rtdx,'ochan3'); %close output channel3
close(cc.rtdx,'ochan4'); %close output channel4
close(cc.rtdx,'ochan5'); %close output channel5
close(cc.rtdx,'ochan6'); %close output channel6
close(cc.rtdx,'ochan7'); %close output channel7
close(cc.rtdx,'ochan8'); %close output channel8
close(cc,x,'project');
im=[outdata1 outdata2 outdata3 outdata4 outdata5 outdata6 outdata7 outdata8];
im=vec2mat(im,256);
im=double(im);
im=im';
im=mat2gray(im);
imshow(im);
```

4.5 RESULTATS DES ALGORITHMES IMPLEMENTES

4.5.1 Algorithmes de prétraitement

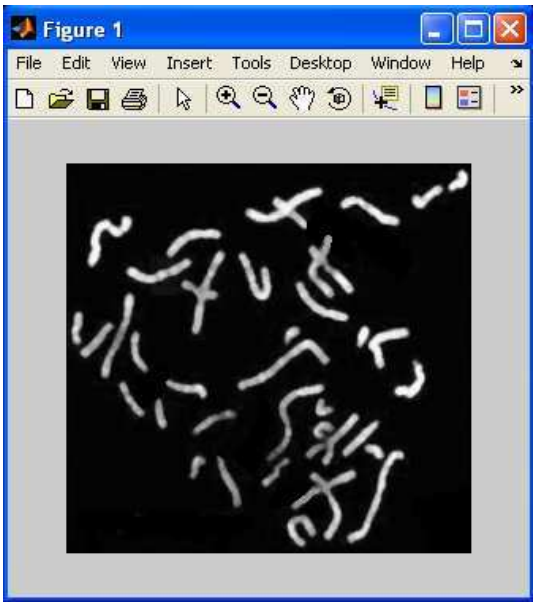


Figure 4.2: Image traitée, filtre Médian

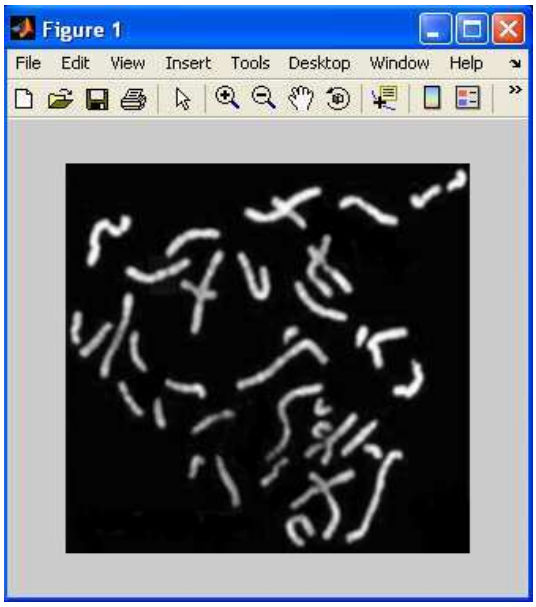


Figure 4.3 : Image traitée, filtre Smooth

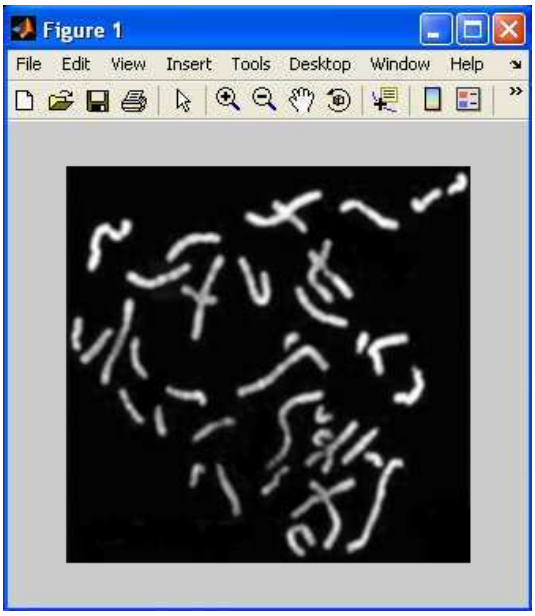


Figure 4.4 : Image traitée, filtre Moyenneur

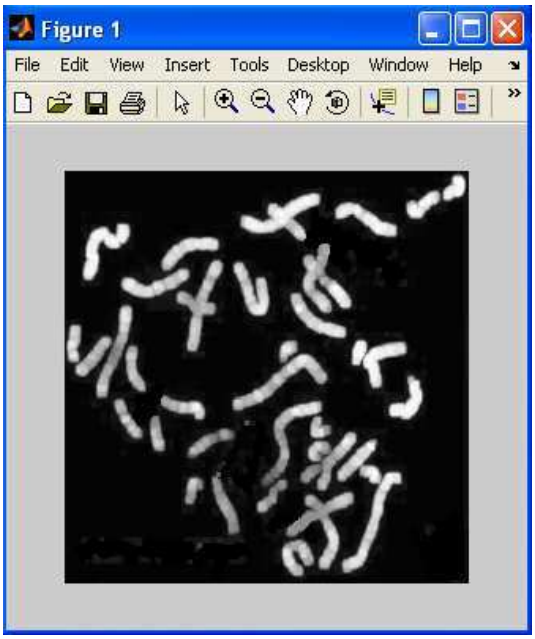


Figure 4.5: Image traitée, filtre morphologique (dilatation)

Figure 4.6 : Image traitée, filtre morphologique (érosion)

Figure 4.7: Image traitée, filtre morphologique (ouverture)

Figure 4.8 : Image traitée, filtre morphologique (fermeture)

4.5.2 Algorithmes de détecteurs de contours

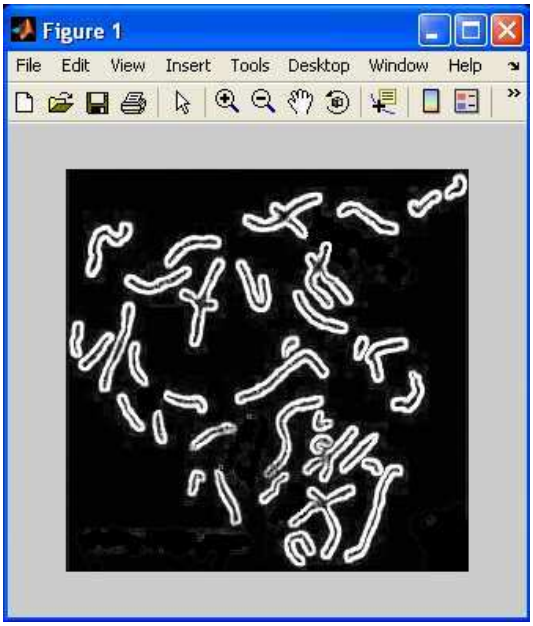


Figure 4.9: Image traitée, filtre Prewitt

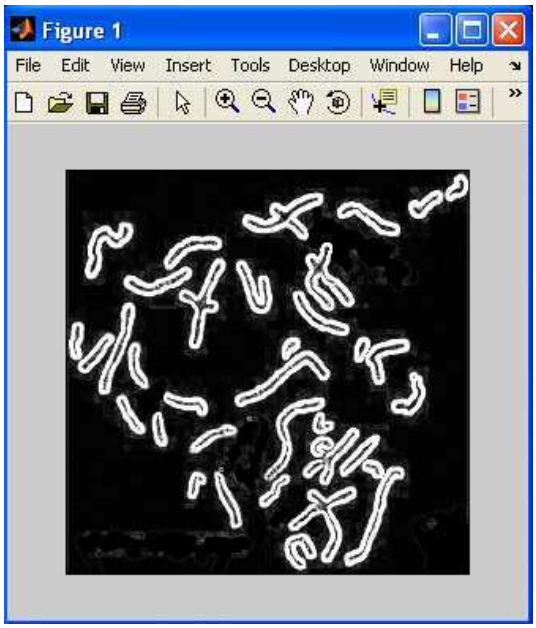


Figure 4.10 : Image traitée, filtre Sobel

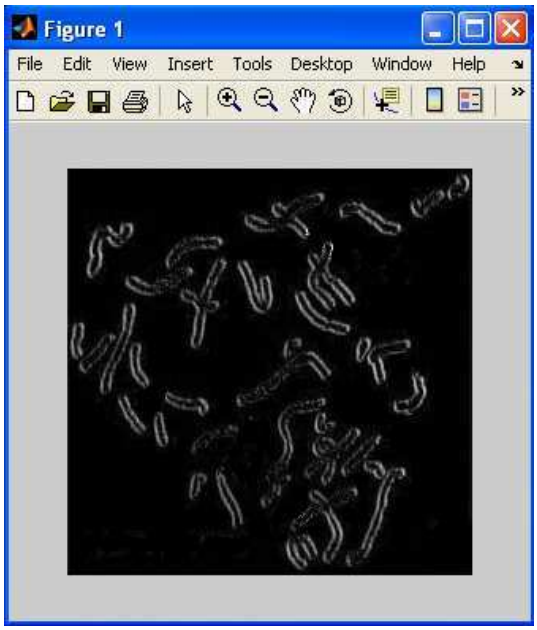


Figure 4.11: Image traitée, filtre Roberts

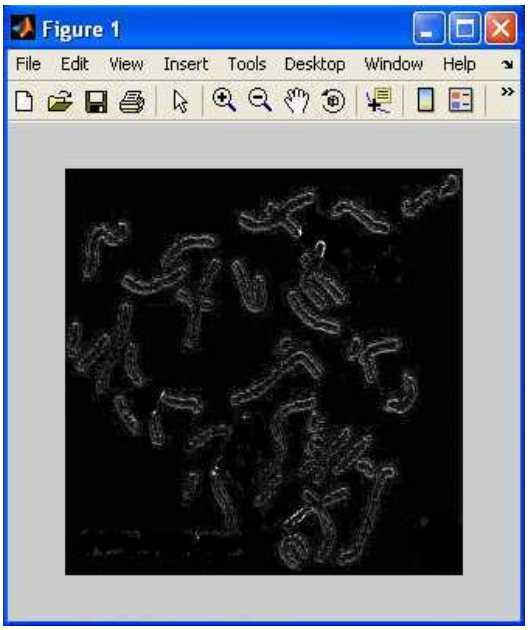


Figure 4.12: Image traitée, filtre Laplacien

4.5.3 Algorithme d'étiquetage

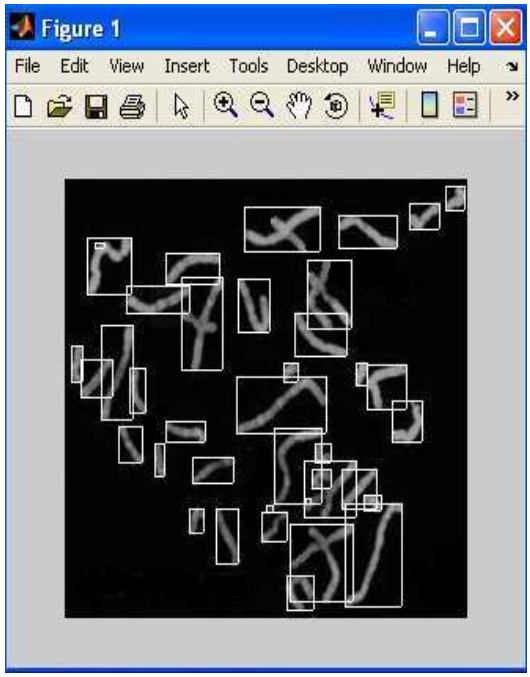


Figure 4.13: Image étiquetée

4.5.4 Algorithmes de squelettisation

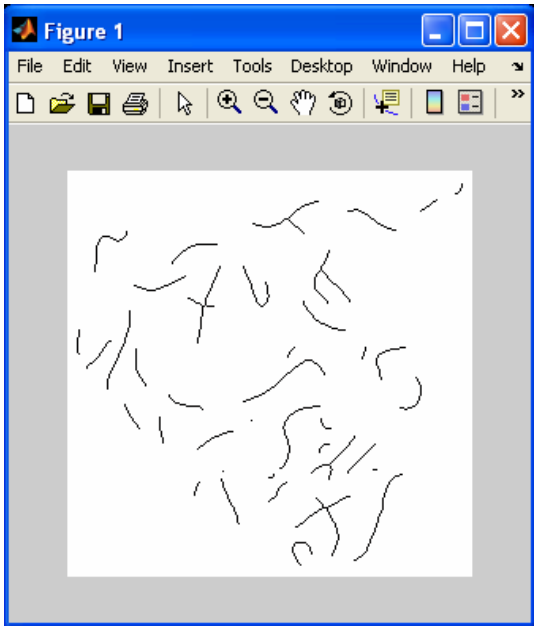


Figure 4.14: Algorithme de Marthon

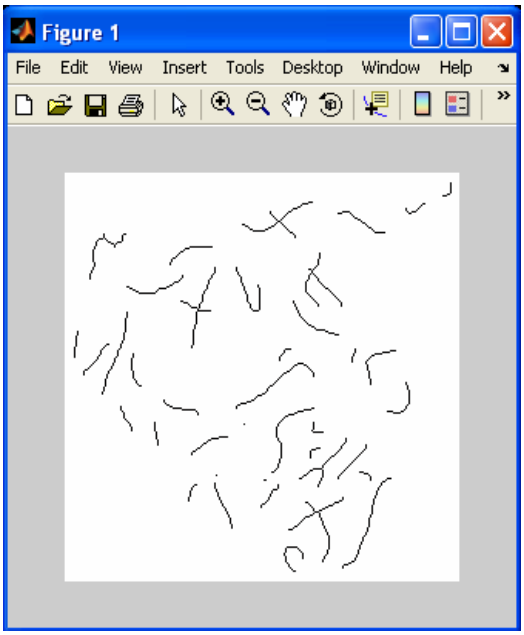


Figure 4.15: Algorithme de Stentiford

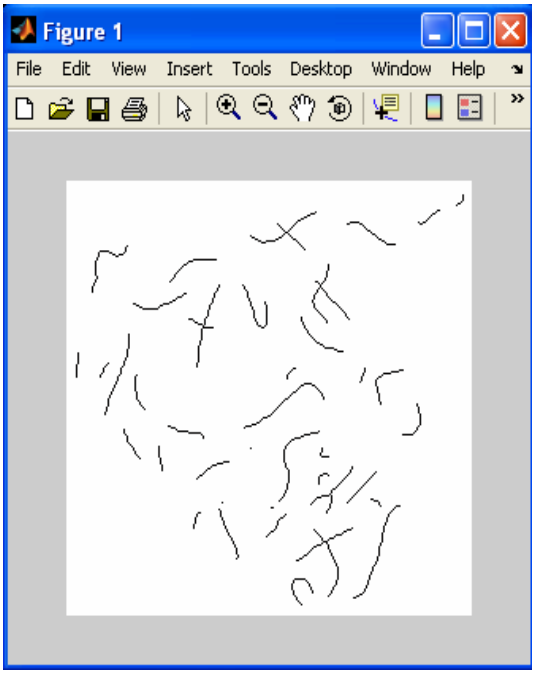


Figure 4.16: Algorithme de Zang&Suen

4.6 ALGORITHME DE SEPARATION DES CHROMOSOMES

4.6.1 Introduction

Les systèmes de classification des chromosomes peuvent souvent se heurter à des problèmes dus à la base de données. En effet les images à traiter présentent fréquemment des cas de contacts (chromosomes qui se touchent) et/ou qui se chevauchent. La présence de tels cas peut induire totalement en erreur le système. Un système automatique de détection et de séparation s'avère donc nécessaire. Néanmoins, la difficulté de concevoir un tel système reste importante, et cela est du à la nature variable des chromosomes (volume, orientation dans l'image) ainsi qu'au nombre important de chromosomes et d'objets chromosomiques. Une technique de séparation, la plus générale possible, est indispensable pour traiter la majeure partie de ces cas.

Le schéma général du système de traitement automatique des images de chromosomes est présenté dans la figure 4.13

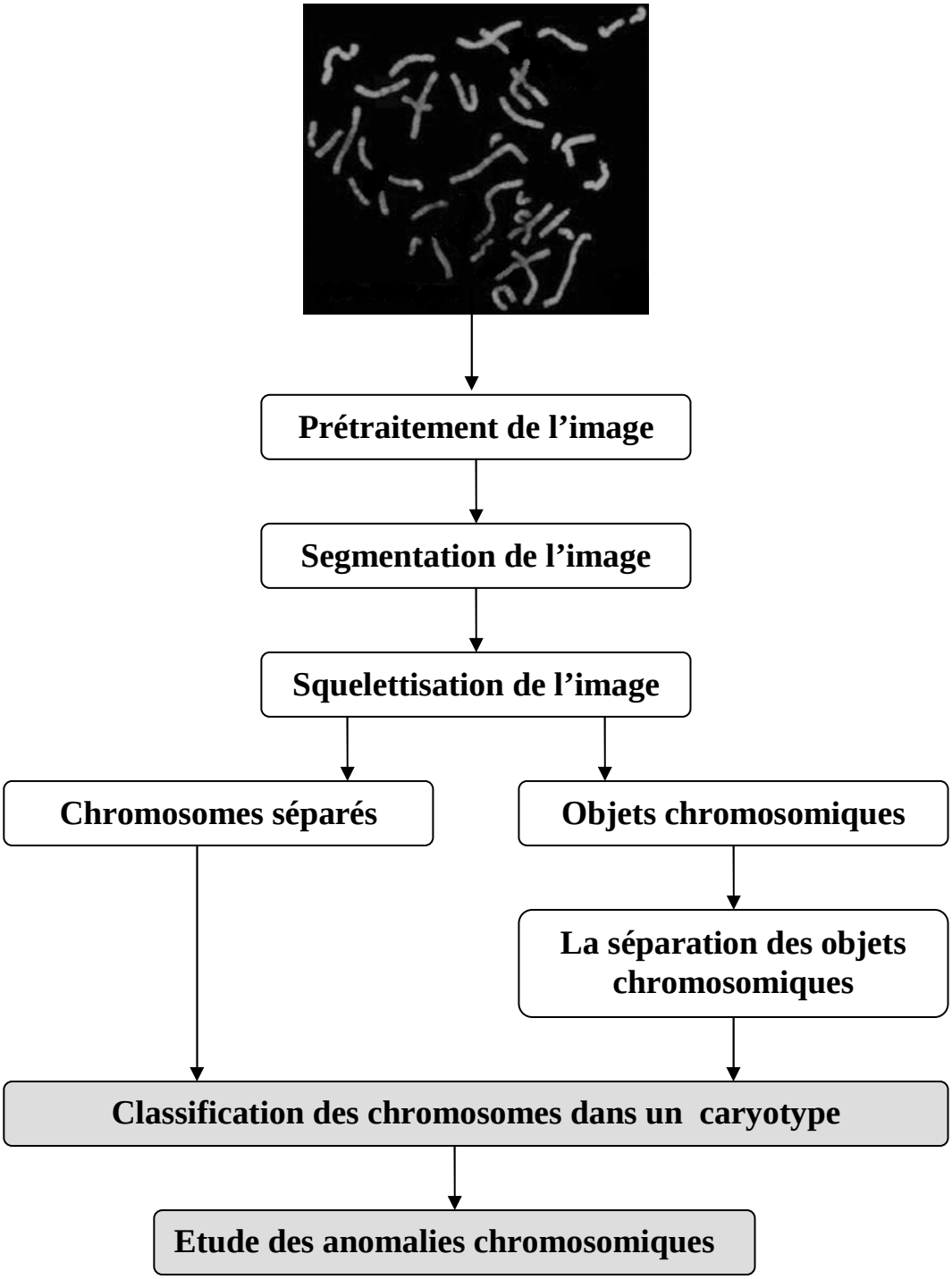


Figure 4.17 : Système de traitement automatique des images de chromosomes

Dans notre projet, nous avons implémenté la première partie du système présenté dans la figure 4.13. Les deux autres parties feront l'objet d'autres études.

4.6.2 Description de l'algorithme de séparation

Nous allons à présent détailler les étapes de séparation comme présenté dans l'organigramme (Figure 4.13),

- **Binarisation de l'image**

Cette étape consiste, comme nous l'avons expliqué dans le deuxième chapitre, à transformer l'image en une image bichrome. Il faut noter que pour les besoins de l'algorithme de l'étiquetage et celui de la squelettisation, nous avons inversé la couleur du fond et de l'objet. L'algorithme de binarisation devient alors :

$$I(L(i,j)) = \begin{cases} \text{Blanc si } L(i,j) < S1 \\ \text{Noir si } L(i,j) > S1 \end{cases}$$

L (i,j) est le niveau de gris du pixel (i,j)

- **Etiquetage de l'image**

Cette étape permet de délimiter les régions à traiter. En effet, chaque objet doit être considéré à part. La délimitation de l'objet se fait grâce à un cadrage, comme nous le voyons dans la Figure (4.9). Après étiquetage selon l'algorithme général présenté dans le chapitre II. Le cadrage se fait comme suit :

Début
Pour chaque pixel de l'image
Pendant que pixel est étiqueté **ET** Limite supérieure de l'étiquette (e) n'est pas trouvée
Limite supérieure de l'étiquette (e) =i ;
Fin pendant que.

Le même algorithme est reconduit pour le calcul des autres limites du cadre.

- **Squelettisation de l'image**

L'algorithme adopté pour la squelettisation est celui de Zang & Suen. Ce choix est motivé par les raisons suivantes :

- Cet algorithme contrairement à celui de Marthon nous permet de garder toutes les branches d'un objet chromosomique composé de chromosomes chevauchés ou qui se touchent.

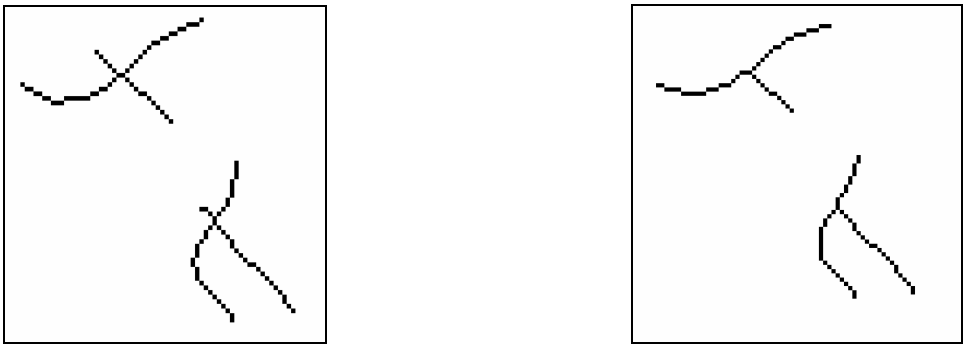


Figure 4.18 : Résultats de la squelettisation donnée par Marthon (à droite) et Zang&Suen (à gauche)

- Il nous donne le nombre d'extrémités nécessaires pour les objets chromosomiques. Ce n'est pas le cas pour l'algorithme de Stentiford qui, en ajoutant quelques branches tout au long du squelette fausse le nombre d'extrémités existantes.

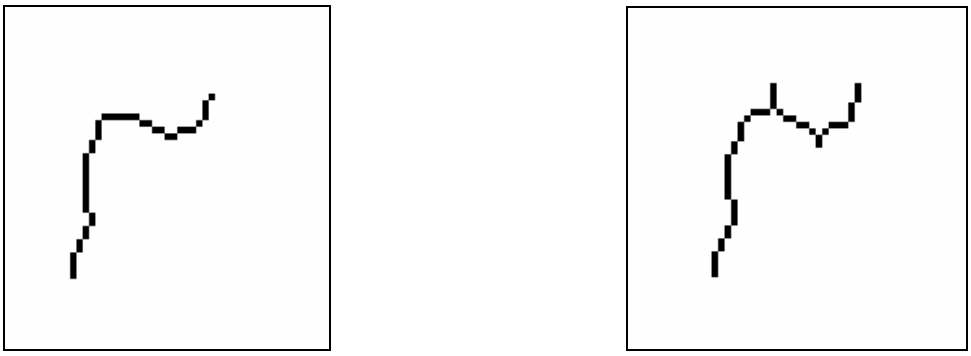


Figure 4.19 : Résultats de la squelettisation donnée par Stentiford (à droite) et Zang&Suen (à gauche)

- L'algorithme donne une complète continuité du squelette. Ce qui n'est pas le cas pour l'algorithme d'épluchage et nettoyage.

Ces motivations sont illustrées par la figure suivante

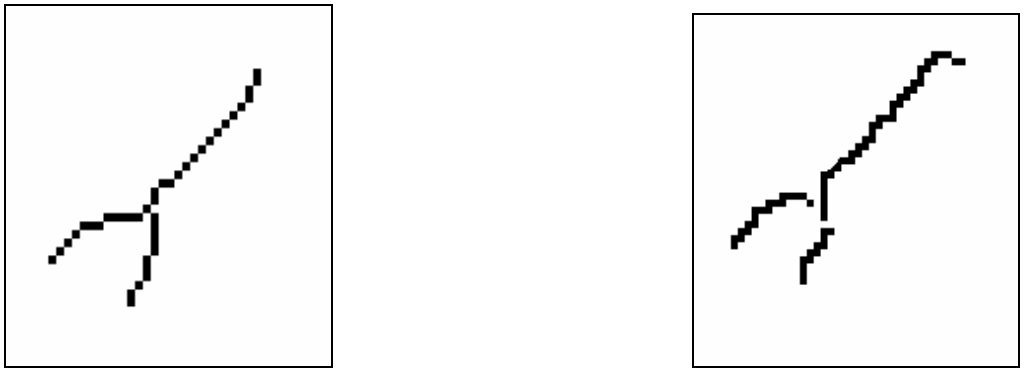


Figure 4.20 : Résultats de la squelettisation donnée par l'épluchage (à droite) et Zang&Suen (à gauche)

Chaque squelette résultant portera l'étiquette de l'objet en cours. Cette opération servira, comme nous allons le voir plus loin à réduire le nombre de balayages dans les prochaines opérations.

A noter que les différents traitements par les algorithmes de squelettisation (Zang & Suen , Marthon et Stentiford) ont subi une étape supplémentaire qui est le nettoyage du squelette, qui a permis d'avoir une meilleure continuité du squelette.

- **La sélection des objets chromosomiques**

Cette étape est composée de plusieurs sous étapes qui sont :

- Sélection des objets intéressants : cela se fait par la détection des objets ayant un pixel du squelette à plus de deux voisins objets (point d'intersection des squelettes).
- Nettoyage des cadres : cela signifie la suppression des objets appartenant à un cadre différent et qui ne sont pas des objets intéressants. Cette opération est nécessaire dans la mesure où la présence de ces objets poserait problème dans les étapes futures.
- Affectation de nouvelles étiquettes : cette étape permet de réduire le nombre de balayages de l'image. A partir des anciennes étiquettes précédemment affectées aux squelettes, nous affectons de nouvelles étiquettes plus petites et uniquement aux objets chromosomiques à traiter.
- Dissociation entre les cas de chevauchement et les cas de contact : cela se fait par le comptage du nombre d'extrémités par objet ; un cas de chevauchement présente quatre extrémités tandis qu'un cas de contact en présente trois.

- **Restauration des objets chromosomiques**

Cette étape consiste à restaurer, à partir de l'image étiquetée, les objets sélectionnés. Puis l'affectation des nouvelles étiquettes à ceux là.

Un deuxième nettoyage s'avère nécessaire pour supprimer les objets non intéressants.

- **Détection de contours**

Dans cette étape, nous avons opté pour détecteur de Roberts. Ce choix est justifié par la finesse des contours que ce filtre donne, et l'absence de discontinuités de ceux-ci.

- **Affichage du contour sur squelette**

Cette étape permet de détecter les points de coupe. La détection de ces points se fait comme suit :

A partir du point d'intersection précédemment détecté. Nous calculons les distances entre celui-ci et le contour de l'objet selon les huit directions de Freeman. L'enveloppe sera dessinée à partir des quatre points du contour les plus proches du point d'intersection.

- **Séparation des objets chromosomiques**

Après délimitation par l'enveloppe des pixels objets communs entre deux chromosomes chevauchés. Nous avons opéré à la séparation de ceux-ci.

La séparation des chromosomes qui se touchent s'est faite par la détection du point de touche.

Tout les objets traités son restaurés à partir de l'image d'origine dans une nouvelle image.

- **Squelettisation des chromosomes obtenus après séparation**

Cette étape ne fait pas partie de l'algorithme de séparation, elle sert uniquement à tester les résultats obtenus. En effet l'importance de l'étape de séparation réside dans la préservation des attributs des chromosomes. Ceci permettra de procéder à une bonne extraction des paramètres en vue d'une classification.

Figure 4.21 : Test des résultats obtenus

4.7 INTERPRETATION DES RESULTATS

Après avoir testé les quatre approches d'implémentation sur notre algorithme de séparation des chromosomes. Nous avons opté pour la quatrième solution (ccsmat2) qui consistait en la réception de matrices de taille 256*64, tout en utilisant un seul canal de sortie. Ce choix est justifié par l'importance de la taille des matrices à la réception et par sa rapidité.

L'algorithme de séparation a donnée de bons résultats.

4.8 CONCLUSION

La communication avec la carte DSK via le port parallèle, n'a pas été performante (très lente) chose qui pose une perte de message au niveau de la liaison CCS----MATLAB, par ailleurs une simulation de la carte sous CCS est possible, cela nous a permis de tester les quatre solutions que nous avons proposées.

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.

L'algorithme de séparation des chromosomes a été implémenté, les résultats présentés dans ce chapitre permettent d'évaluer l'efficacité de notre méthode. Nous avons traité une image comportant quatre chevauchements, et un contact de trois extrémités. Un test utilisant la squelettisation a été effectué, prouvant la validité des résultats attendus.

.

CONCLUSION GENERALE

A travers notre travail, nous avons pu nous initier au domaine très intéressant de la cytogénétique ainsi qu'aux techniques de préparation des images.

Une étude générale des techniques de traitement d'images s'est avérée nécessaire sinon primordiale. En commençant par les algorithmes de prétraitement, de segmentation par les deux approches (contour et région) et en terminant par les algorithmes de squelettisation. Une implémentation préalable de tous ces algorithmes nous a permis de faire le choix des techniques nécessaires pour la séparation des chromosomes se touchant ou se chevauchant dans une image de métaphase.

Notre but étant l'implémentation de cet algorithme sur un DSP de la famille C6000 de TI, à savoir la cible C6211 mise en œuvre sur une carte DSK. Nous avons utilisé 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. Malheureusement la communication avec la carte en utilisant un câble parallèle s'est avérée lente entraînant de ce fait une perte de données. Ainsi, une simulation a été nécessaire pour pouvoir tester la faisabilité de la connexion.

Nous avons réalisé une librairie de routines de traitement d'images pouvant être utilisée sous Code Composer Studio.

Un test a été effectué concernant la fiabilité de notre algorithme de séparation vis-à-vis des paramètres nécessaires pour une éventuelle classification des chromosomes et détection des anomalies.

Notre souhait serait de pouvoir réaliser d'une application en temps réel ; malheureusement 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 ceci 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 au laboratoire signal et communications.

Grâce à ce projet, nous avons pu acquérir des connaissances précieuses dans une branche très intéressante de l'électronique que nous n'avions pas eu l'occasion d'étudier, ainsi qu'un bon contact avec la pratique.

PERSPECTIVES

- Continuer dans la même lancée à réaliser un système automatique de classification des chromosomes, qui servira à mettre au point un système de traitement d'images totalement autonome ou embarqué.
- La création de bibliothèque de traitement d'images, afin d'alléger nos programmes et d'optimiser le temps d'implémentation.
- 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 deTexas Instruments concernant l'RTDX et les périphériques, au lieu de MATLAB.

BIBLIOGRAPHIE

[1] http://www.ac-versailles.fr/etabliss/herblay/genetiqu/fiches/gene.htm

[2] http://fr.wikipedia.org/wiki/Chromosome

[3] H. AIT ABDESSLAM, ” Réalisation d’un système d’aide à la détection des aberrations chromosomiques”,Thèse de magister présentée à l’ENP spécialité électronique, février 2005.

[4] http://cvirtuel.cochin.univ-paris5.fr/cytogen/1-4.htm

[5] http://iquebec.ifrance/kadchakib/chap1/chap1.htm

[6] J. COCQUEREZ, S. PHILIPP, ” Analyse d’images : filtrage et segmentation”, édition MASSON, 1995.

[7] C. Gonzalez, R. E. Woods, ”Digital Image Processin (eBook) ”,Prentice Hall, 2nd Edition 2002

[8] G. BRAVIANO, ”Logique floue en segmentation d’images : seuillage par entropie et structures pyramidales irrégulières”,Thèse de doctorat UJF Grenoble 1

[9] http://www710.univ-lyon1.fr/~fdenis/club_EEA/cours

[10] H.KRAUS, ” Scan & retouche d’images”,CAMPUS Press , 1999 .

[11] J. FAUQUEUR, ”Traitement et Manipulation d'image”, INRIA, 2000-2001.

[12] A.MASTER, R.KHELIFI, ” Segmentation des images coronarographiques : Application a la détection des cardiopathies”, Diplôme d’ingénieur d’état en électronique ENP, promotion 2005.

[13] J.J.TOUMAZET, ”Traitement de l’image sur micro-ordinateur”, SYBEX, 1987

[14] P. CANHAM VAN DIJKEN, ”L’image numérique.Institut de Pathologie de LAUSANNE”.

[15] O.MONGAT, ”Vision par ordinateur”, INRIA 1990.

[16] I. BLOCH, Y. GOUSSEAU, D. MATIGNON, B. PESQUET-POPESCU, F. SCHMITT, M. SIGELLE, F. TUPIN, ”Polycopié du cours ANIM”, Département TSI - Télécom- Paris Version 5.0, Septembre 2005.

[17] A. COUTANT, ”La méthode des contours actifs en traitement des images”,Mémoire pour l’examen probatoire en Calcul Scientifique”, Conservatoire National des Arts et Métiers Paris, février 2005.

[18] www.deverney.fr

[19] N. ROULA, ”Filtrage et Segmentation d’images comparés de diverses approches ;Approche Variationnelle & Approche Neurale,Séminaire de radiotechnique”, dirigé par Mr. A.ZERGUERRAS, Février 2006.

[20] S.TIRES, ” Filtrage et Segmentation d’images comparés de diverses approches Approche Markovienne & Approche Neurale”, Séminaire de radiotechnique, dirigé par Mr A.ZERGUERRAS, Février 2006.

[21] H.BENHIDOUR, ”Système de classification des chromosomes par réseau de neurones” ,Diplôme d’ingénieur d’état en Informatique, INI, promotion 2001.

[22] A. CHABANE, M. MICHOT, ”Méthodes de Traitement des Images”,ENST, 1997.

[23] A. RAHMANIA, ”Système de détection d’aberrations chromosomiques structurales ” Diplôme d’ingénieur d’état en Informatique, USTHB, promotion 2002.

[24] S.TIRES,N. ROULA, ”Simulation d’algorithme de réduction de bruit & de détection de contours sur DSP C6000”,Mini projet de microprocesseurs spécialisés dirigé par Dr.L. HAMAMI, ENP 2006.

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

[26] TMS320C6000 CPU and Instruction Set, SPRU189f.pdf, Texas Instruments, Data sheet.

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

[28] TMS320C6000 Programmer’s Guide, SPRU198d.pdf, Texas Instruments.

[29] TMS320C6000 Optimizing Compiler User’s Guide, SPRU187g.pdf, Texas Instruments.

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

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

[32] Code Composer Studio Help, Texas Instruments.

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

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

[35] TMS320C6211, TMS320C6211B Fixed-Point Digital Signal Processor

[36] S. AHMED ZAID, ”Implémentation sur DSP C6000 d’algorithmes de traitement d’image”, Diplôme d’ingénieur d’état en Electronique, ENP ,promotion 2006.

[37] DSP II: ELEC 4523, Real-Time Data Exchange.

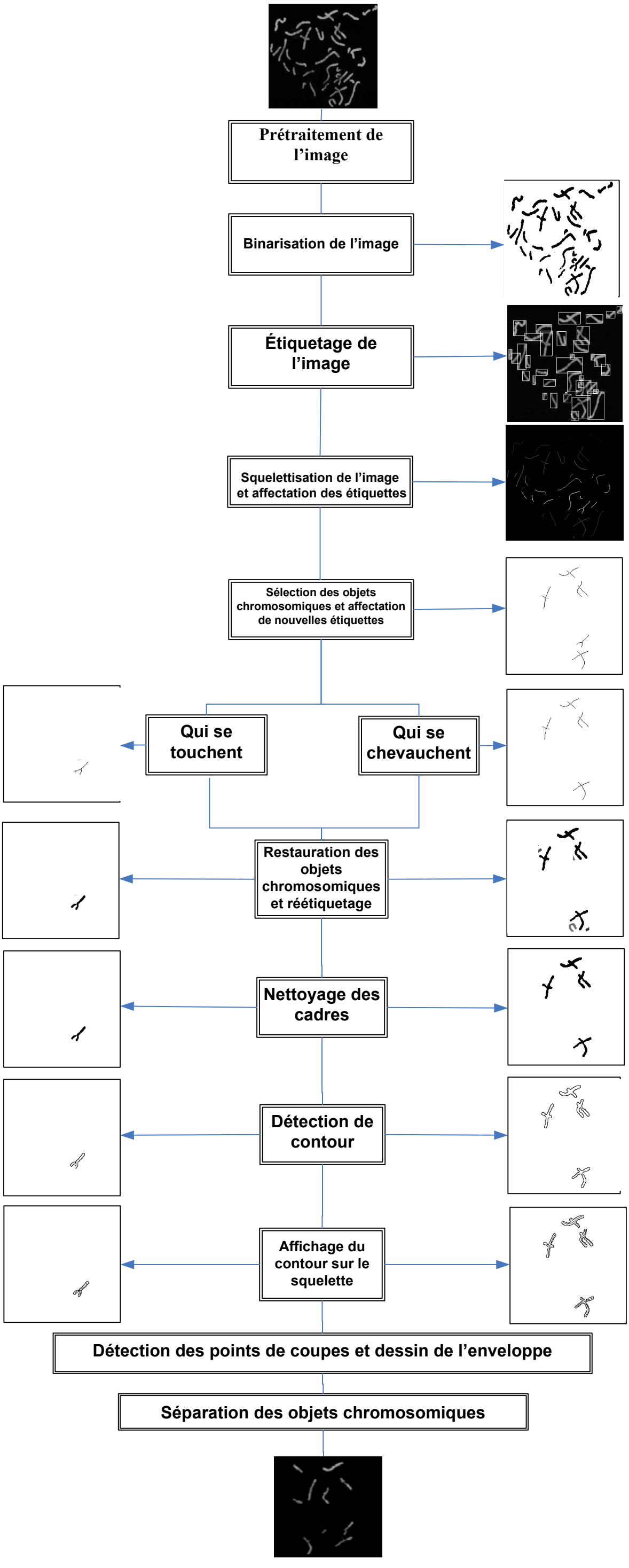


Figure 4.22:ORGANIGRAMME DE L'ALGORITHME DE SÉPARATION DES CHROMOSOMES