République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Dr Yahia Farès de Médéa

Faculté des Sciences et de la Technologie

Département du Génie électrique et de l'Informatique

Mémoire de fin d'études pour l'obtention

du diplôme de Master en Génie électrique

Option : Systèmes Intelligents et Communicants

Thème :

Implémentation d'algorithmes de traitement d'images en VHDL

Encadré Par :

Dr. OULD ZMIRLI Mohamed

Présenté Par :

TAHRAOUI Mokhtar

Promotion : 2011 – 2012

Remerciement

Nous tenons à remercier notre encadreur *Dr. OUELD-ZMIRLI Mohamed* pour ses encouragements, ses conseils et sa disponibilité, qui nous ont été utile au cours de l'élaboration de notre projet de fin étude.
A mes parents
A mes sœurs, et à mes frères
A tous ceux qui nous ont aidés et encouragés.

ملخص:

الجزء الأول من هذا العمل يخص دراسة و محاكاة بعض خوارزميات معالجة الصور باستخدام لغة Matlab. ويخصص القسم التالي لتنفيذ خوارزميات ترشيح بلغة VHDL، بدءا من تصميم البنية المادية، واستعراض نتائج المحاكاة والتركيب وثم التنفيذ على FPGA.

Résumé :

Ce travail consiste en première partie à l'étude et à la simulation de quelques algorithmes de traitement d'images en utilisant le langage Matlab. La partie suivante est consacrée à l'implémentation en VHDL d'algorithmes de filtrage, en commençant par la conception de l'architecture matérielle, aux résultats de la simulation et ensuite la synthèse et l'implémentation sur un circuit FPGA.

Abstract:

This work consists in the first section to study and simulate some image processing algorithms using Matlab. The next section is devoted to VHDL implementation of filtering algorithms, starting with the design of the hardware architecture, the results of the simulation and then the synthesis and implementation on an FPGA.

SOMMAIRE :

Introduction générale..……..……01
Chapitre 01: Généralités des technologies des circuits logiques...……..……..02
I.1 Introduction ……....03
I.2 Les grands types des circuits logiques..…..……..…03
I.3 La classification des technologies des circuits logiques...……..….04
I.3.1 Les circuits ASIC..……..……..05
I.3.2 Les DSPs..……..……...06
I.3.3 Les circuits FPGAs..……..……...06
I.4 Historique sur FPGA..……..……..06
I.5 L'architecture des circuits FPGA (exemple XILI97)..……..……..08
I.6 Configuration des FPGA..……..……..09
I.7 Quelques fabricants des circuits FPGA..……..……...10
I.8 Conclusion..……..……...11
Chapitre 02:Techniques de traitement d'image avec simulation sous Matlab……..……..……12
II.1 Introduction..……...……....13
II.2 Acquisition d'image..……..……...13
II.2.1 Echantillonnage..……..……...14
II.2.2 Quantification..……..……..14
II.3 Prétraitement d'image..……..……14
II.3.1 Les opérateurs morphologiques :…….....15
II.3.1.1 Dilatation.…….....15
II.3.1.2 Erosion.…….....16
II.3.1.3 Ouverture.…….....16
II.3.1.4 Fermeture.…….....16
II.3.2 La convolution d'une image.…….…17
II.3.3 Le filtrage de l'image…….…18
II.3.3.1 Le filtre médian :…….…18
II.3.3.2 Le filtre moyenneur…….……19
II.3.3.3 Le filtre seuil…….……...……20
II.3.3.4 Les filtres maximal et minimal…….……20
II.3.4 Prétraitement par histogramme…….……..20
II.3.4.1 Contraste d'une image…….……...……..20
II.3.4.2 Modification linéaire du contraste…….....21
II.3.4.3 L'histogramme d'une image……...……...……21

II.3.4.4 Egalisation d'histogramme……..……..21
II.4 Quelques techniques de traitement d'image……..…….22
II.4.1 Détection des contours…….....…….…….22
II.4.1.1 Filtres dérivées premières…….……23
II.4.1.2 Filtre de Prewitt……...…..……...23
II.4.1.3 Filtre de Sobel……..…….23
II.4.1.4 Filtre de Canny……...……..……24
II.4.2 Segmentation d'image……...……..…...…24
II.4.2.1 Manuelle par le seuillage…….....24
II.4.2.2 Automatique……...……...…24
II.5 Affichage et/ou stockage d'image……...……...…25
II.6 L'implémentation sous matlab……...26
II.1 Description de l'interface réalisé sous Matlab…..……..26
II.7 Conclusion……..30
Chapitre 03 : Implémentation des algorithmes de traitement d'image en langage VHDL ..……..…..…31
III.1 Introduction…….32
III.2 Architecture de la fenêtre 3x3…….33
III.3 Le modèle VHDL de test Bench…….34
III.4 Les programmes MATLAB des fichiers d'entrée/sortie…….35
III.5 Données utilisées…….36
III.6 Mise en œuvre en VHDL de l'algorithme de filtre d'ordre par tri de données……...36
III.7 Algorithme de filtre d'ordre par tri de données mis en œuvre en VHDL……...40
III.7.1 Résultats de la simulation fonctionnelle de l'algorithme de filtrage……40
III.7.2 Résultats de la simulation fonctionnelle des opérations morphologiques…….……...…41
III.8 Résultats de la synthèse et de l'implémentation……..45
III.9 Conclusion ……...……...48
Conclusion générale……49
Annexe..……....50
Annexe A..……....51
Annexe B…..……....56

Liste des figures :

Chapitre I :……...

Fig(I.1) : classification des circuits intégrés logiques……..5

Fig(I.2) : les composants unitaires d'un circuit FPGA……....7

Fig (I.3) : Architecture interne de FPGA……....09

Fig(I.4) :les blocs à configurer dans le circuit FPGA……09

Fig (I.5) : les modes de configuration de FPGA……10

Chapitre II :…….....

Fig (II.1): le schéma du parcours d'image…….....13

Fig (II.2) :la formule mathematique de convolution matricielle……18

Fig (II.3): les étapes du parcours d'un filtre médian……..19

Fig (II.4):exemple du filtre moyenneur 3x3……...19

Fig (II.5) : graphe de modification du contraste……21

Fig (II.6) : exemple d'égalisation d'un histogramme……22

Fig (II.9) :l'interface graphique de traitement d'image matlab…….....26

Fig (II.10): Chargement de l'image à traiter……...27

Fig (II.11) : sélection du traitement désiré…….....28

Fig (II.12) : résultats de simulations utilisant Matlab……...29/30

Chapitre III :…….

Fig (III.1): Mapping des composants VHDL……34

Fig (III.2): Architecture du générateur de la fenêtre…….....35

Fig (III.3) : principe de comparaison utilisé……..37

Fig (III.4) : l'organigramme de l'algorithme de tri……...39

Fig (III.5): structure de la conception VHDL de l'algorithme de filtrage……40

Fig(III.6) : image filtrée……..41

Fig(III.7) : images érodées et dilatées …….42/43

Fig(III.8) : images ouvrées et fermées…….44/45

Fig(III.9) : résultat de synthèse de l'algorithme de filtrage……...46

Fig(III.10) : l'environnement intégré du logiciel Quartus II v 9.0 chargé du projet…..……...47

Fig(III.11) : Schémas du niveau RTL du système de filtrage……...48

Fig(III.12) : plan des étages du système implémenté……49

Liste des tables :

Table (I.1) : types des circuits logiques……..03

Introduction générale

Actuellement, le circuit FPGA (Field Programmable Gate Array) est devenu une cible très intéressante pour la mise en œuvre d'algorithmes adaptés aux applications de traitement d'images et de vidéo. L'architecture de ce circuit a permis à la technologie d'être utilisée dans de nombreuses applications couvrant tous les aspects de ces traitements. L'objectif de ce mémoire est de développer une implémentation de quelques algorithmes du traitement d'image sur un circuit FPGA.

La solution logicielle (soft) basé sur les processeurs est devenu moins utile dans le traitement vidéo en temps réel à cause de la croissance de tailles d'images traitées et de leur résolution. Les systèmes en temps réel tels que ceux qui sont ciblés par ce projet sont nécessaires pour les vitesses élevées requises dans le traitement vidéo.

En outre, un problème commun est exigé vu la grande quantité de données capturées, par exemple, à l'aide des satellites et des systèmes de détection au sol. Les systèmes de traitement numérique du signal ou les systèmes DSP (Digital signal processing) sont employées pour réduire la quantité de données à traiter, en faisant passer seulement les données pertinentes à l'analyse humaine. Finalement, on s'attend à ce que le traitement de la plupart des vidéos peut prendre place dans ces systèmes DSP, avec peu d'interaction humaine. Ceci est évidemment avantageux, car l'analyse humain de données est coûteuse et peut-être pas tout à fait exact.

Ce mémoire s'organise en trois chapitres comme suit:

Le premier chapitre donne un aperçu sur les circuits logiques et en particulier le circuit FPGA.

Le deuxième chapitre présente quelques algorithmes de traitement d'image avec des résultats de simulation utilisant le langage Matlab.

Dans le troisième chapitre on passe vers l'implémentation matérielle en VHDL d'une algorithme de filtrage, en passant par la présentation de la conception, des résultats d'une simulation fonctionnelle et ensuite la synthèse et l'implémentation sur un circuit FPGA.

Finalement, une conclusion générale et quelques perspectives possibles seront données.

Chapitre I :

Généralités des technologies des circuits logiques

I.1 Introduction :

Dans quelques années ; le monde prend une grande diversité dans le domaine technologique. Ce- ci implique une spécification pour certain développement. C'est-à-dire le sujet à rechercher et développé exige un environnement spécial par ces deux types : matériel et\ou logiciel, par exemple : les caractéristiques d'ordinateur, langage utilisé à la programmation, la version… etc.

Le techniques de traitement d'image en temps réel peuvent être implémentés soit en utilisant un circuit DSP, un circuit ASIC ou un circuit reconfigurable FPGA en utilisant le langage VHDL ou verilog.

Au traitement d'image ; lorsque la quantité de données à traiter est concerné, ces données peuvent être un vidéo qui prend une grande taille de mémoire, surtout pour les DSPs, qu'il sera couteux aux ressources, au temps,… ça nous guide vers l'utilisation d'un circuit ASIC ou FPGA.

I.2 Les grandes types des circuits logiques :

Par ordre décroissant de complexité de conception, on trouve les types de circuits intégrés suivants:

Type	Description
Full custom	Des circuits conçus à partir d'une puce de silicium, tout est modifiable : transistors (type, caractéristiques), connexions.
Standard cell	les éléments logiques sont choisis dans une bibliothèque de portes, les connexions sont libres
Gate Arrays (prédiffusés)	les éléments logiques existent déjà physiquement sur le circuit, seules les connexions peuvent être définies

Table (I.1) : types des circuits logiques

Il y a un compromis entre la complexité de conception et les performances.

I.3 La classification des technologies des circuits logiques:

Cette classification est devenue selon la technologie supportée par le circuit par rapport à sa construction, son mode de configuration, et de sa programmation.

a) *Les microprocesseurs et les mémoires RAM et ROM :* les microprocesseurs et les circuits mémoires sont attrayants pour les fabricants. Composants de base pour les systèmes informatiques, ils sont produits en très grandes séries.

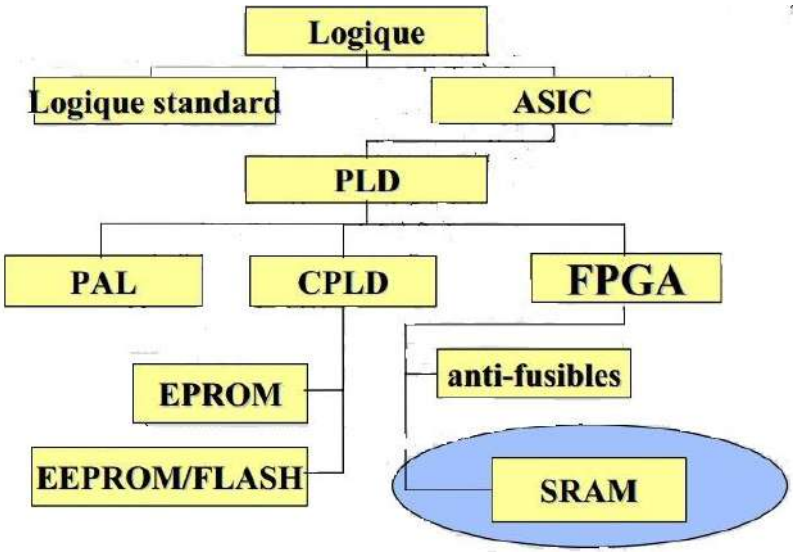
b) *Les circuits programmables sur site :* n'importe quelle fonction logique, combinatoire ou séquentielle, avec un nombre fixe d'entrées et de sorties, peut être implantée dans ces circuits. A partir de cette simple idée, plusieurs variantes d'architecture ont été développées (PAL, EPLD, FPGA,…).

c) *Les ASIC programmés chez le fondeur :* le circuit est conçu d'un point de vue logiciel par l'utilisateur, puis il est réalisé par le fondeur.

A l'heure actuelle, la majorité des circuits numériques est issue de ces trois familles. Plus simplement, on peut distinguer deux catégories de circuits intégrés : les circuits standards et les circuits spécifiques à une application :

- Les circuits standards se justifient pour de grandes quantités : microprocesseurs, contrôleurs, mémoires, …
- Les circuits spécifiques sont destinés à réaliser une ou un ensemble de fonctions dans un système bien particulier [4].

La figure suivante représente une classification des circuits intégrés numériques.



Fig(I.1) : classification des circuits intégrés logiques

Les circuits FPGA SRAM deviennent des solutions de remplacement avantageuses pour les systèmes numériques à haute intégration.

I.3.1 Les circuits ASIC :

Les circuits ASIC (Application-Specific Integrated Circuits) sont des circuits qui représentent une technologie dans laquelle les ingénieurs peuvent créer une conception matérielle fixe en utilisant une variété d'outils.[1]

on trouve aussi autres types de circuits reconfigurables comme : ASSP (Application-Specific Standard Parts), PLD (Programmable Logic Devices), soit SPLD(Simple PLD), ou CPLD (Complex PLD), FPGA(Field Programmable Gate Array). L'inconvénient des circuits ASIC est trouvé ou une erreur était dans la configuration de circuit, une modification de circuit sera couteux pour l'utilisateur.

I.3.2 Les DSPs :

Les DSPs (digital signal processor) se situent quelque part entre un ASIC et un PC en termes de performance et de complexité de conception. Ils peuvent être programmés avec le code assembleur ou le langage C. Les performances de l'algorithme au niveau du DSP sont certainement plus élevées que sur un PC, mais dans certains cas, les systèmes ASIC ou FPGA sont le seul choix pour une conception. Cependant, les DSP sont une solution très courante et efficace de traitement des données en temps réel.

Les progrès récents dans la technologie DSP ont entraîné les mêmes implémentations d'algorithmes à haute vitesse. Bien que les avantages des ASIC et FPGA sont toujours en vigueur.

Les systèmes électroniques modernes sont de plus en plus complexes. Les contraintes de taille, de puissance dissipée et de performances sont de plus en plus sévères (téléphonie mobile, ordinateurs, traitement du signal, de l'image, etc...), et l'accroissement spectaculaire des densités.

I.3.3 Les circuits FPGAs :

Les FPGAs (Field Programmable Gate Arrays) sont des circuits intégrés qui contiennent des blocs logique configurables (programmables), ainsi que des interconnexions configurables entre ces blocs. La configuration du circuit, c'est-à-dire la programmation de sa fonction, se fait sur place (Field Programmable), sans envoi chez un fabricant.

Afin de pouvoir finaliser un FPGA, il est nécessaire d'utiliser un langage de description matériel ou bien un outil de saisie graphique. Après compilation de cette description, on obtient un fichier de configuration pour le FPGA choisi. VHDL et Verilog sont les deux langages de description les plus répandus.

I.4 Historique sur FPGA :

En 1982, trois anciens de Fairchild Semi-conductor, Bob Hartmann, Paul Newhagen et Michael Magranet terminent leur livre sur l'industrie des circuits prédiffusés (Gate Arrays : Implementing LSI Technology) par : « The probabilities are high that someone will produce an electrically alterable logic array ».

Avec Jim Sansburry (techno. HP) et Jim Hazle (finances), ils fondent la societe Altera le 3 juin 1983. C'était le début de l'industrie des PLD (programmable logic device).

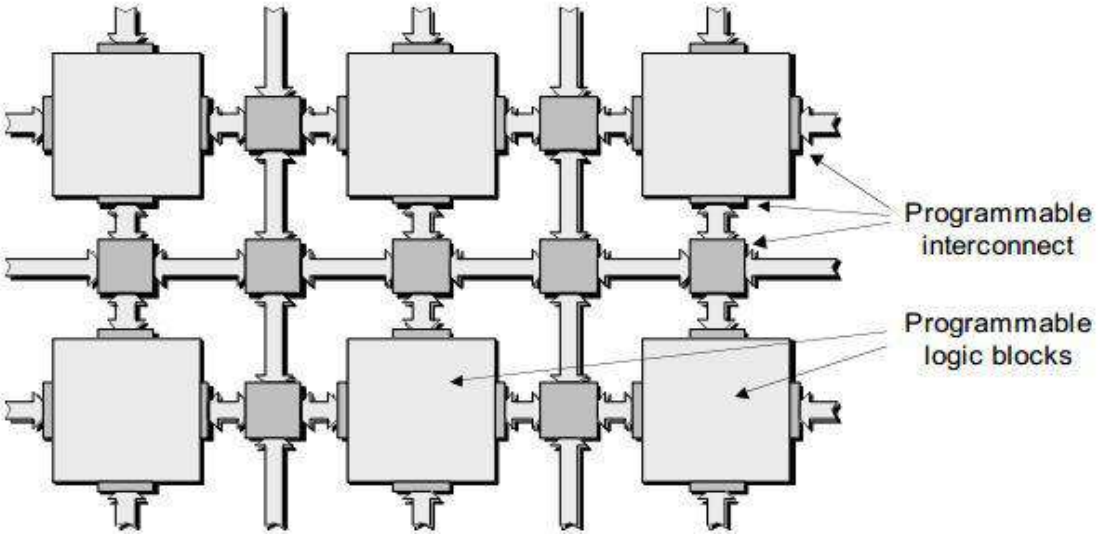
Le développement été guidé vers des circuits programmables, Il était crée des sortes de « gate arrays » dont les caractéristiques logiques sont programmables électriquement (plutôt que gravées physiquement) [3].

Les FPGA se placent à demi-chemin entre les PLD et les ASIC. Les avantages par rapport aux ASIC sont:

• coût de développement plus bas: les dépenses en NRE (nonrecurring engineering) sont moins importantes.

• modifications plus simples à réaliser.

• time-to-market plus court.



Fig(I.2) : les composants unitaires d'un circuit FPGA.

La structure du bloc logique change selon le fabricant, la famille, etc.

Aujourd'hui, la structure la plus utilisée est basée sur une look-up table (RAM) pour implémenter une fonction combinatoire plus une bascule D[1].

Les technologies de configuration des interconnexions sont: fusible(1), anti-fusible, EPROM, E2PROM/Flash, et SRAM.

Les pins d'entrée/sortie des FPGA possèdent également des caractéristiques configurables: type d'interface, impédance, voltage. Le voltage d'alimentation est en baisse constante, blocs pour la transmission de données en série à grande vitesse (transceivers)[2].

Les contraintes physiques, technologiques et commerciales imposent aujourd'hui la conception de systèmes de plus en plus : compacts, économes en énergie, rapides à concevoir, et fiables et peu coûteux.

I.5 L'architecture des circuits FPGA (exemple XILI97) :

Elle est similaire à celle des circuits prédiffusés classiques. Bien qu'il existe actuellement plusieurs fabricants de circuits FPGA dont Xilinx et Altera sont les plus connus, et plusieurs technologies et principes organisationnels, nous illustrons ce concept à partir des circuits Xilinx que nous avons employés. L'architecture, retenue par Xilinx, se présente sous forme de deux couches :

Une couche appelée circuit configurable,

Une couche réseau mémoire SRAM.

La première couche est constituée d'une matrice de blocs logiques configurables CLB permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout auteur de ces blocs logique configurable, nous trouvons des blocs entrées/sorties IOB dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs (figure I.4).

La programmation du circuit FPGA appelé aussi LCA (Logic Cells Arrays) consistera par le biais de l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ à interconnecter les éléments des CLB et des IOB afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM [13].

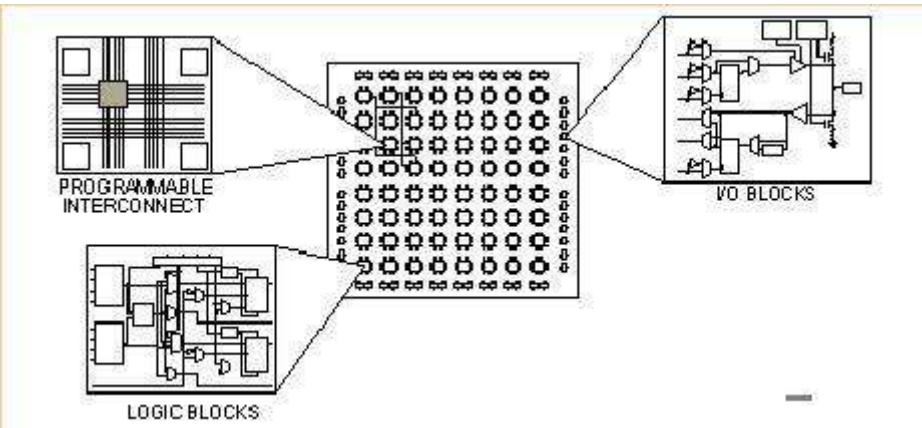
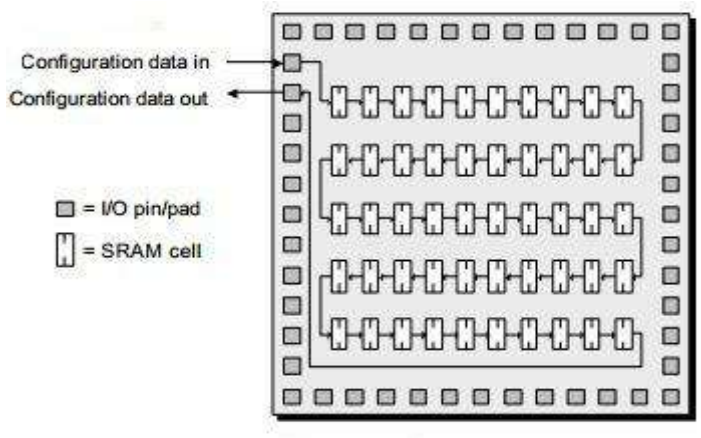


Fig (I.3) : Architecture interne de FPGA.

I.6 Configuration des FPGA:

La chaîne de bits de configuration (bit Stream) est envoyée depuis l'extérieur et distribuée en série à l'intérieur du FPGA. On peut voir les SRAM de configuration comme un énorme registre à décalage (dans la réalité, c'est divisé en frames de 1024 bits et on utilise des latches). Le temps de configuration peut être très long (on a aujourd'hui des biststreams de 25Mb).



Fig(I.4) :les blocs à configurer dans le circuit FPGA.

Il existe en général 4 modes de configuration: série avec le FPGA en maître, série avec le FPGA en esclave, parallèle avec le FPGA en maître, et parallèle avec le FPGA en esclave.

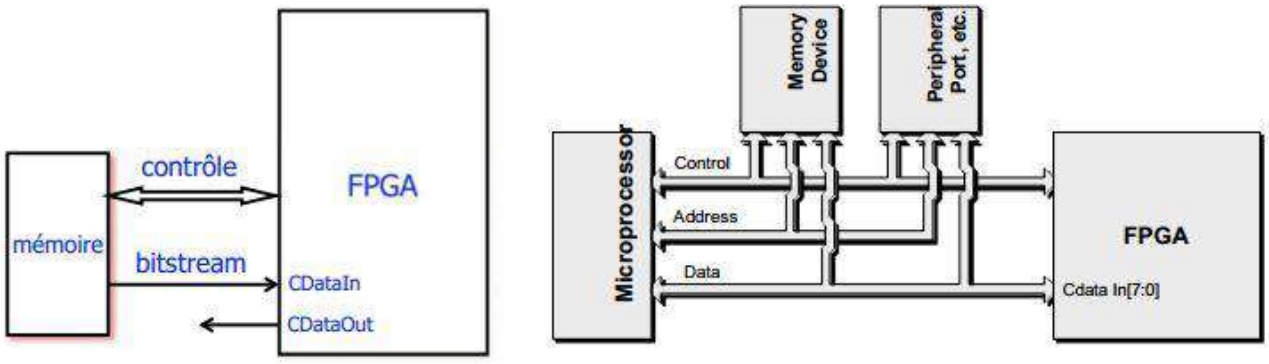


Fig (I.5) : les modes de configuration de FPGA.

Une autre façon de configurer le FPGA est l'utilisation du port JTAG (Joint Test Action Group). Normalement, ce port est utilisé pour tester le circuit intégré, grâce à la technique de boundary scan (pour debug). Un pin est utilisé pour entrer les données JTAG et un autre pour les sortir. Tous les autres pins d'entrée/sortie disposent d'un flip-flop JTAG; tous ces flip-flops sont connectés en série. En plus des commandes propres au test, le port JTAG des FPGA admet aussi des commandes de configuration, permettant le chaînage du registre de configuration au registre JTAG.

I.7 Quelques fabricants des circuits FPGA :

Les circuits FPGA sont fabriquées par plusieurs société, parmi elles on cite : Altera, Actel, ATMEL, Cypress, Lattice, Lucent, Quick Logic, et Xilinx.

Les logiciels utilisés pour implémenter un circuit FPGA :

Nous avons les outils (langages et logiciels), qu'ils sont classé sur trois niveaux :

- Outil bas niveau (synthèse physique)
  - Xilinx ISE.
  - Altera.
- Synthèse :

- Xilinx XST.
- Altera.
- Leonardo.
- Synplify.

- Simulation :
  - ModelSim.

I.8 Conclusion :

Dans ce chapitre on a présenté les différents types des circuits logiques, et en particulier le circuit FPGA. Au chapitre suivant on va présenter quelques techniques de traitement d'image, puis la simulation en utilisant Matlab comme une première étape.

Chapitre II :

Techniques de traitement d'image avec simulation sous M*atlab*

II.1 Introduction :

Après la définition des notions de base on décrit dans ce chapitre le parcours d'une image depuis son état réel jusqu'au son stockage, avec les traitements connues dans ce domaine. Et ensuite on présente l'interface réalisé utilisant le langage Matlab qui appelle les fonctions implémentant ces traitements ainsi que les résultats de simulation.

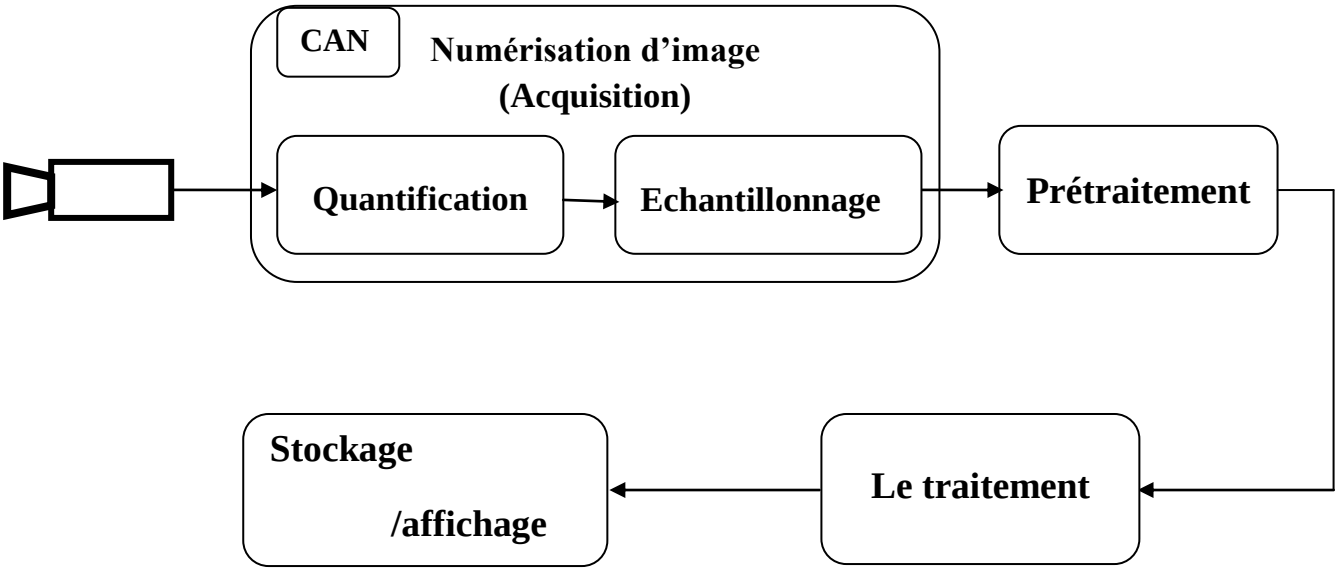


Fig (II.1): le schéma du parcours d'image.

II.2 Acquisition d'image :

L'acquisition d'images est loin de se restreindre au spectre visible, mais parcourt l'ensemble du spectre électromagnétique, des rayons gamma jusqu'aux ondes radio. L'utilisation d'une bande de fréquence dépend essentiellement de l'application. Ainsi, les images acquises par rayons X ou par rayons gamma sont surtout utilisées en imagerie médicale et en astronomie.

Il est également possible de produire des images à partir de radar. La différence avec un capteur classique, qui se contente de recevoir un signal électromagnétique, est qu'il produit sa propre illumination par génération d'ondes radio grâce à une antenne, l'image est alors produite à partir de la réflexion du signal sur la cible. Il est également possible de générer une image en dehors du

spectre électromagnétique, par exemple à partir d'une source sonore (échographie, prospection pétrolière...).

L'acquisition d'une image est l'opération du passage entre l'image réelle et l'image numérique. L'acquisition est donc la première étape utilisée pour obtenir une image numérisée facile à manipuler et traiter. Cette étape se passe par deux opérations essentielles qui sont : l'échantillonnage et la quantification. L'acquisition est définis aussi par la conversion analogique-numérique d'image. Car le signal contenant une infinité des points de données, ce ci implique la minimisation de la quantité de donnée par ces opérations.

II.2.1 Echantillonnage : c'est l'opération de segmentation du signal d'image au niveau d'axe des x pour obtenir des échantillons suffisants à définir l'image réelle par celle numérique.

II.2.2 Quantification : c'est l'opération de segmentation du signal d'image au niveau d'axe des Y. Cette opération peut effectuer par plusieurs types de quantifieurs existants, on citons par exemple : le quantifieur bloqueur (bloquer le signal au long d'échantillon à son première valeur.), moyenneur (qui prend la moyenne des valeurs d'échantillon comme valeur quantifiée au cet échantillon.)….

II.3 Prétraitement d'image :

L'étape de prétraitement signifie la préparation de l'image au traitement donné. Cette phase a lieu juste après l'acquisition des images et a pour objectif d'améliorer la qualité de l'image. Cette étape a plusieurs opérations chacune selon le traitement désigné, par exemple : la modification d'histogramme, la réduction des bruits par le filtrage, la convolution, et aux plus autres quatre méthodes, qu'elles sont regroupées sous la nomenclature « les opérateurs morphologiques ».

II.3.1 Les opérateurs morphologiques :

La morphologie mathématique offre des opérateurs non linéaires particulièrement utiles pour filtrer, segmenter et quantifier des images. Initialement destinée au traitement des images binaires, elle a très vite été généralisée aux images à niveaux de gris, puis aux images en couleurs et multi-spectrales. Ces opérations basées sur un type de traitement nommé « l'élément structurant ».

La nature des opérateurs morphologiques fait qu'ils se prêtent bien au développement de circuits électroniques spécialisés (ou bien à l'utilisation de FPGA).

L'élément structurant : c'est l'unité utilisée pour effectuer une opération morphologique, qui signifie le domaine d'application d'opération. Par exemple en traitement d'image c'est-à-dire traitement des matrices, on choisit un domaine pour partitionner notre traitement. Parmi les éléments structurants les plus utilisés on trouve : type carré, hexagonal, crois …etc. A notre traitement on choisit le type carré.

II.3.1.1 Dilatation :

Soit une image binaire X et un élément structurant M. La dilatation de X par M est une image binaire définie par :

$$DilM(X) = \{(x,y) \mid M(x,y) \cap X \neq \emptyset\}[5]$$

Dans notre application l'élément structurant (M) est un carré entour le pixel désigné, avec la dilatation est la recherche du maximum dans cet élément.

Algorithme :

Pour chaque pixel de l'image x(i,j) faire
Fen=(x(i-1 :i+1,j-1 :j+1)) ;
DilM(i,j)=max(fen) ; prendre le maximum;
End pour

II.3.1.2 Erosion :

Soit une image binaire X et un élément structurant M. L'érosion de X par M est une image binaire définie par :

$$EroM(X) = \{(x,y) \mid M(x,y) \subset X\}[5]$$

Avec l'érosion est la recherche du minimum dans l'élément structurant.

Algorithme :

Pour chaque pixel de l'image x(i,j) faire
Fen=(x(i-1 :i+1,j-1 :j+1)) ;
DilM(i,j)=min(fen) ; prendre le minimum;

End pour

II.3.1.3 Ouverture :

Soit une image binaire X et un élément structurant M. L'ouverture de X par M est une image binaire définie par :

OuvM(X) = DilM-(EroM(X)). [5]

L'ouverture c'est l'opération de dilatation d'une image déjà érodée.

II.3.1.4 Fermeture :

Soit une image binaire X et un élément structurant M. La fermeture de X par M est une image binaire définie par :

FerM(X) = EroM-(DilM(X)). [5]

La fermeture c'est l'opération d'érosion d'une image déjà dilatée.

Les opérateurs morphologiques peuvent être utilisés dans plusieurs applications du traitement d'image: l'élimination du bruit, l'extraction de contours, le remplissage de régions et amincissement et épaississement [5].

II.3.2 La convolution d'une image :

En pratique, la convolution numérique d'une image se fera par une sommation de multiplications, Un filtre de convolution est une matrice (image), généralement *(mais pas toujours)* de taille impaire et Symétrique 3x3, 5x5, 7x7, …

La convolution consiste à recalculer la valeur d'un pixel donné d'une image de départ (image source) en se basant sur la valeur du pixel lui-même et sur la valeur des pixels environnants le pixel à recalculer.

Chaque pixel environnant est affecté d'un coefficient de calcul, l'ensemble des coefficients utilisés sont regroupés dans un tableau de valeurs (appelé aussi "matrice"). Ce tableau est appelé le masque.

La valeur finale du pixel dans l'image finale (image destination) correspond à un calcul prenant en compte tous les pixels environnants du pixel de départ et le pixel de départ, combinés un à un aux coefficients du masque [12].

Convolution d'une image par un filtre 2D :

$$I'(i,j) = I(i,j) * filtre(i,j)$$

$$I'(i,j) = \sum_{u} \sum_{v} I(i-u, j-v) \cdot filtre(u,v)$$

Fig (II.2) :la formule mathematique de convolution matricielle.

Le masque de convolution représente un filtre linéaire permettant de modifier la forme de l'image [10].

II.3.3 Le filtrage de l'image.

Le filtrage c'est l'élimination de bruit intégré dans l'image à cause d'une mauvaise acquisition à partir de l'appareil ou l'environnement d'image. Parmi ces filtres existants on a le filtre moyenneur, bloqueur et médian.

II.3.3.1 Le filtre médian :

L'objectif du filtre médian est la réduction du bruit impulsionnel (pics aberrants dans l'image). Les rayons cosmiques sont l'une des causes d'un tel bruit : une particule très énergétique interagit avec le capteur. Cela produit une impulsion aberrante dans l'image, très localisée et repérable car elle n'a pas le profil gaussien d'une étoile. On remplace dans l'image initiale chaque pixel par la médiane des intensités des 9 pixels avoisinants. [6]

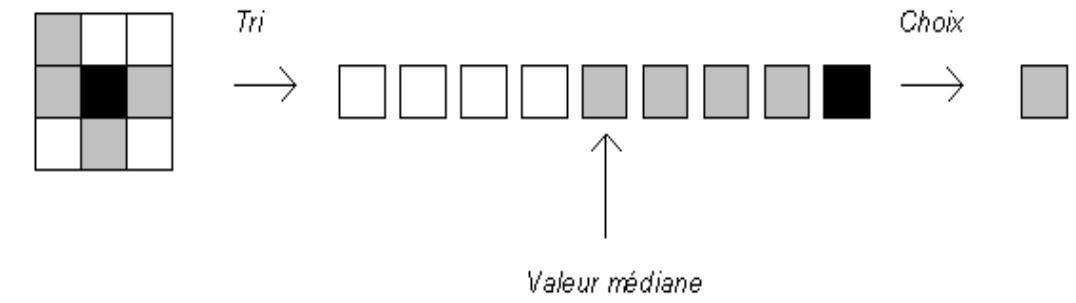


Fig (II.3): les étapes du parcours d'un filtre médian.

Algorithme :

Pour chaque pixel de l'image X(i,j) faire

Fen=X(i-1 :i+1, j-1 :j+1) ;

Fen_triée=tri(fen) ; % tri de fenêtre(matrice) dans un vecteur.

Image_filtrée(i,j)=fen_triée(5) ; % median de (09) éléments.

Fin pour ;

II.3.3.2 Le filtre moyenneur :

Ce filtre à comme masque une fenêtre de taille NxN tel que n soit impair, chaque valeur du masque est multipliée par (1/N).

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

ou

1/9

1	1	1
1	1	1
1	1	1

3x3

Fig (II.4):exemple du filtre moyenneur 3x3.

II.3.3.3 Le filtre seuil :

Le filtre seuil part du même principe que le filtre médian mais permet de produire des images logiques (tous les pixels ont pour intensité 0 ou 1).

On se fixe un seuil donné *S*. La nouvelle intensité du pixel considéré est 1 si la somme des intensités des 9 pixels avoisinants est supérieure à *S*, 0 sinon.

II.3.3.4 Les filtres maximal et minimal :

Le filtre maximal remplace chaque pixel par la plus grande intensité des 9 pixels avoisinants. Cela correspond à une dilatation de l'image.

Le filtre minimal remplace chaque pixel par la plus petite intensité des 9 pixels avoisinants. Cela correspond à une érosion de l'image.

Si un bruit faible existe dans l'image, il sera éliminé par érosion. Une dilatation rendra ensuite à l'image une dynamique comparable à celle de l'image initiale. [6].

II.3.4 Prétraitement par histogramme :

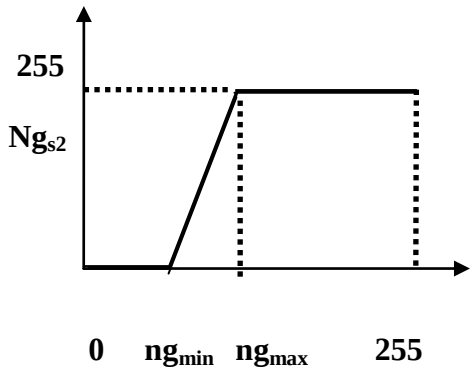
II.3.4.1 Contraste d'une image :

Il est défini par le pouvoir de séparabilité visuelle des niveaux de gris adjacents. Pour certaines images cette séparabilité nécessite un étalement de l'intervalle des niveaux de gris ou de façon générale une modification de contraste. Il existe différentes formes de modification de contraste de la dynamique d'une image qui peut être linaire, logarithmique, uniforme. Le principe de modification de contraste qui entraine une modification d'histogramme à partir d'une fonction appelée fonction de transfert.

II.3.4.2 Modification linéaire du contraste :

C'est l'opération la plus usuelle. Elle consiste à définir et de trouver une fonction F tel que Is=F(Ie), avec Is: image de sortie modifiée et Ie: l'image d'entrée.

Cette fonction est de l'intervalle des ng (niveaux de gris) en entrée et l'intervalle désiré en sortie.



$$\begin{cases} Is = ngs1 \quad si \quad Ie < ngmin. \\ Is = * ngs2 \,/(ngmax - ngmin) \quad si\, ngmin < Ie < ngmax. \\ Is = ngs2 \quad si\, Ie > ngmax. \end{cases}$$

Fig (II.5) : graphe de modification du contraste.

II.3.4.3 L'histogramme d'une image :

Il définit la distribution des niveaux de gris d'une image. C'est-à-dire, calcul du nombre d'occurrence de chaque niveau de gris et le représenté dans un seul graphe.

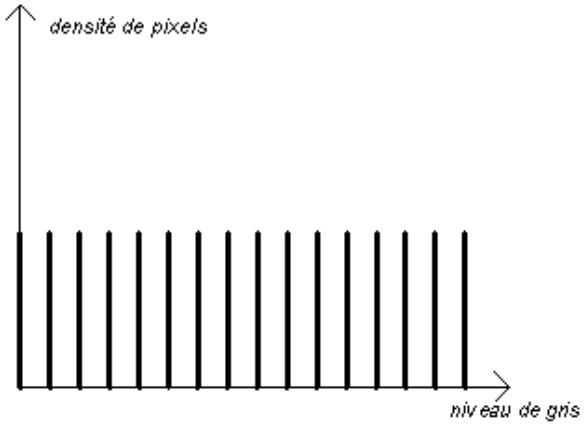
II.3.4.4 Egalisation d'histogramme :

Consiste à rendre le plus plat possible, l'histogramme des niveaux de gris de l'image. L'image est associée à une modélisation probabiliste dont l'histogramme représente la densité de probabilité de la variable niveau de gris (ng). Il permet de renforcer le contraste sur des détails qui sont masqués par des variations d'intensité de plus grande amplitude et à plus grande échelle [9].

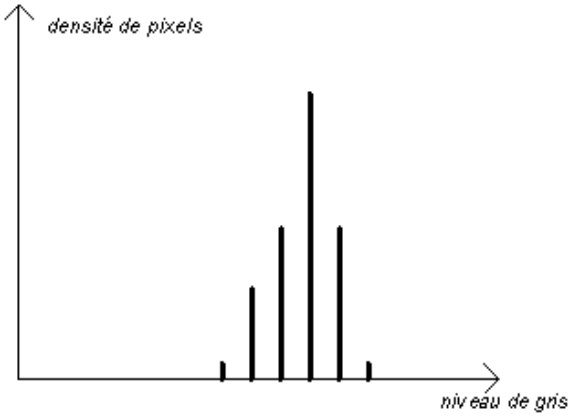
Exemples d'histogrammes [6] :

a) Histogramme constant

b) Faible contraste

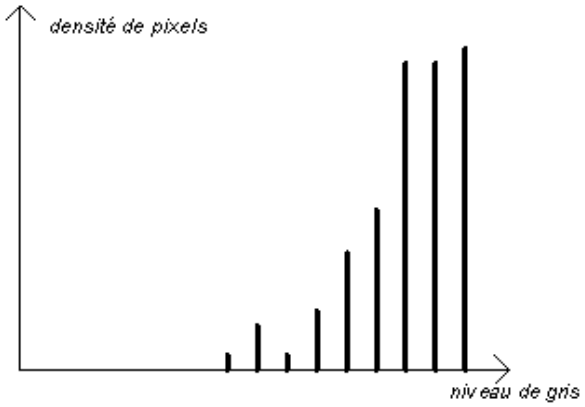


C'est l'histogramme idéal. L'image est contrastée et tous les niveaux de gris sont également utilisés.



la dynamique de l'image est faible : celle-ci est peu contrastée. Quelques niveaux de gris sont exploités.

c) Saturation



Il y a une saturation du capteur : les niveaux de l'image et du numérisateur sont décalés.

Fig (II.6) : exemple d'égalisation d'un histogramme.

II.4 Quelques techniques de traitement d'image :

Dans cette étape on commence réellement au traitement numérique d'une image, à travers la matrice des valeurs de pixels résultante au prétraitement. Le traitement ici correspond aux besoins d'utilisateurs. Parmi ces besoins on a la détection des conteurs, la segmentation …. etc.

II.4.1 Détection des contours:

Ces filtres transforment l'image d'entrée en une image noire sauf aux points où un contour est détecté qui est marqué en blanc. Les valeurs absolues importent peu, il est sans intérêt de changer d'échelle comme pour un lissage.

La détection est basée sur la dérivation selon les deux coordonnées. Si on considère classiquement les signaux comme des sommes de sinusoïdes, la dérivation apparaît comme un filtre

passe-haut qui introduit donc du bruit à l'origine de faux contours. Des méthodes plus élaborées ont été systématisées pour les professionnels.

II.4.1.1 Filtres dérivées premières : Le filtre le plus simple consiste à calculer les différences entre pixels voisins sur les horizontales puis sur les verticales. Chaque extremum correspond à un point d'un contour.

II.4.1.2 Filtre de Prewitt : le filtre de Prewitt introduit un flou, chacune des deux matrices étant le produit du filtre dérivation dans la direction considérée par un filtre de flou rectangulaire selon l'autre direction.

Filtres de Prewitt :

-1 -1 -1 -1 0 1

0 0 0 -1 0 1

1 1 1 -1 0 1

II.4.1.3 Filtre de Sobel :

La technique précédente est améliorée en remplaçant le filtre rectangulaire par un filtre triangulaire.

Filtres de Sobel (module du gradient) :

-1 -2 -1 1 0 1

0 0 0 -2 0 2

1 2 1 -1 0 1

II.4.1.4 Filtre de Canny :

C'est un filtre de Sobel précédé par un lissage gaussien et suivi par un seuillage. Ce filtre est conçu pour être optimal, au sens de trois critères [8].

II.4.2 Segmentation d'image:

La segmentation est une des étapes critiques de l'analyse d'images qui conditionne la qualité des mesures effectuées ultérieurement. Elle permet d'isoler dans l'image les objets sur lesquels doit porter l'analyse, de séparer, dans une image, les régions d'intérêt du fond. Il existe plusieurs techniques, la plus simple étant le seuillage des valeurs de niveaux de gris des images.[11]

La segmentation peut être manuelle ou automatique.

II.4.2.1 Manuelle par le seuillage: il consiste de bien déterminer une valeur S appelée « seuil » qu'elle permet de sélectionner les objets a voulu, c'est-à-dire : la valeur séparative permettant de distinguer l'objet.

L'image résultante soit binaire (0/1) ou (0/255) par un seuillage simple S, sinon par deux seuils définies (S1, S2), et dans ce cas 1 si la valeur est supérieure à S2 ou 0 si inferieure à S1.

II.4.2.2 Automatique:

C'est une méthode qui utilise un seuillage automatique ou un filtre passe haut. La première se base sur l'analyse de distribution des niveaux de gris selon l'histogramme cumulé ou par la maximisation d'entropie, méthode k-means…etc. La deuxième a comme objectif de ne pas éliminer les bruits de fond d'image et l'augmentation du contraste qu'il aide à détecter les régions des objets [11].

II.5 Affichage et/ou stockage d'image :

Pour stocker la matrice résultante du traitement soit :

- On utilise des commandes réservées du logiciel. En matlab, on a la commande suivante :
  - *Imwrite (M,"nom_fichier",format);*
- On programme l'opération de stockage dans un fichier comme suit :
  - Définir un chemin pour le fichier à stocker et son nom (pour créer un emplacement au fichier).
  - Connaitre l'entête du format a utilisé, et gère elle comme exceptions.

- Charger les valeurs de matrice image dans le fichier par l'utilisation des boucles (*loop*).

Et pour l'affichage d'une image on a aussi soit :

- Par des commandes comme avec matlab :
  - *Imshow(nom_image) ;*
  - *Image(nm_image) ;*
  - Ou *imagesc(nom_image) ;*
- Ou bien l'afficher à l'ouverture d'image après le stockage dans un fichier en utilisant l'un des programmes de *photo viewer.*

Dans notre travail, la partie de simulation utilise une interface graphique, c'est-à-dire ces fonctions sont utilisées dans le code source de chaque opération.

II.6 L'implémentation sous matlab :

Description de l'interface réalisé sous Matlab :

La majorité de techniques citées auparavant sont implémentées en MATLAB, par des fonctions selon l'algorithme de chacune. On les regroupe dans un seul programme sous forme d'une interface graphique.

Premièrement on décrit l'interface d'une façon générale illustré par la figure (II.9).

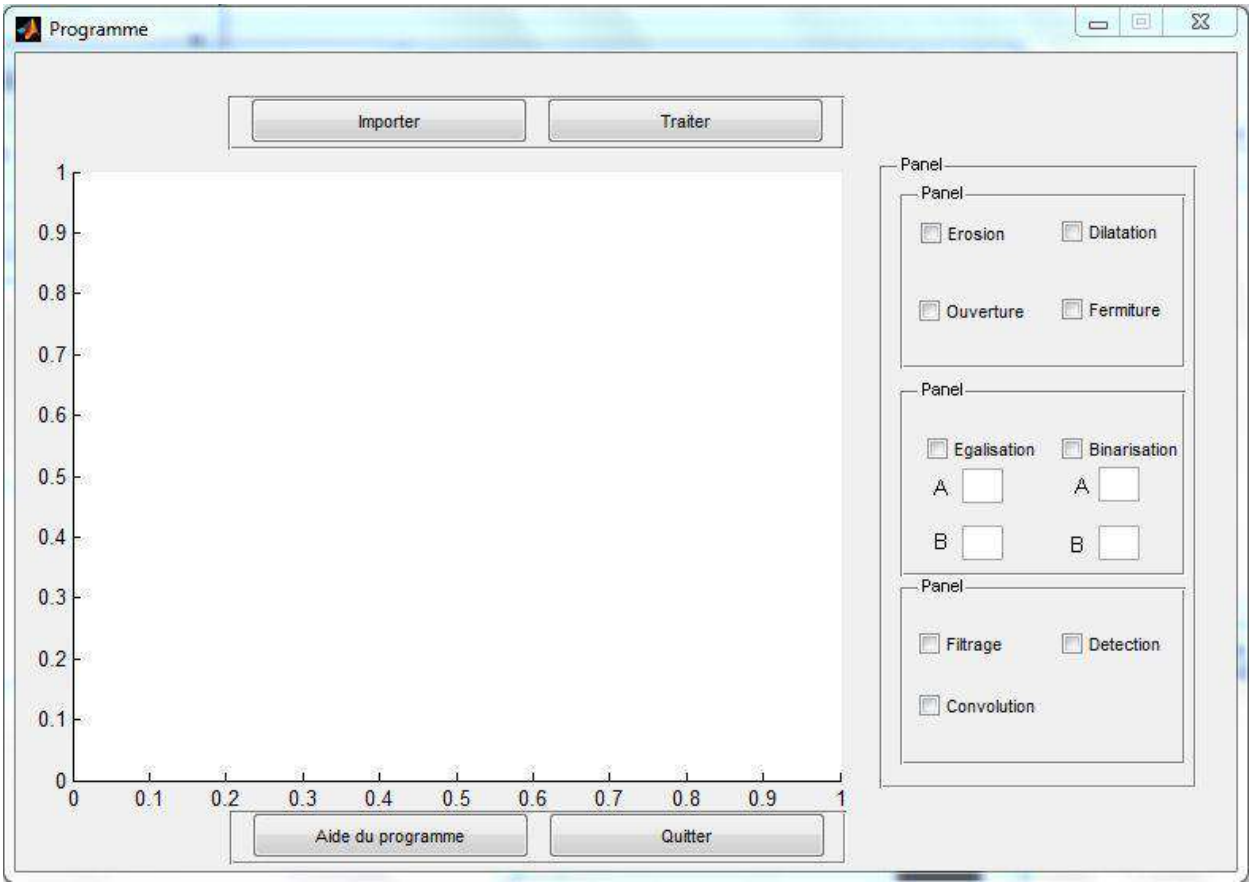


Fig (II.9) :l'interface graphique de traitement d'image matlab.

L'interface contient quatre boutons: importer, traiter, aide et quitter. A gauche une zone pour afficher l'image à sélectionner (importer), et un panneau de sélection d'algorithmes ou méthodes de traitement voulu par l'utilisateur.

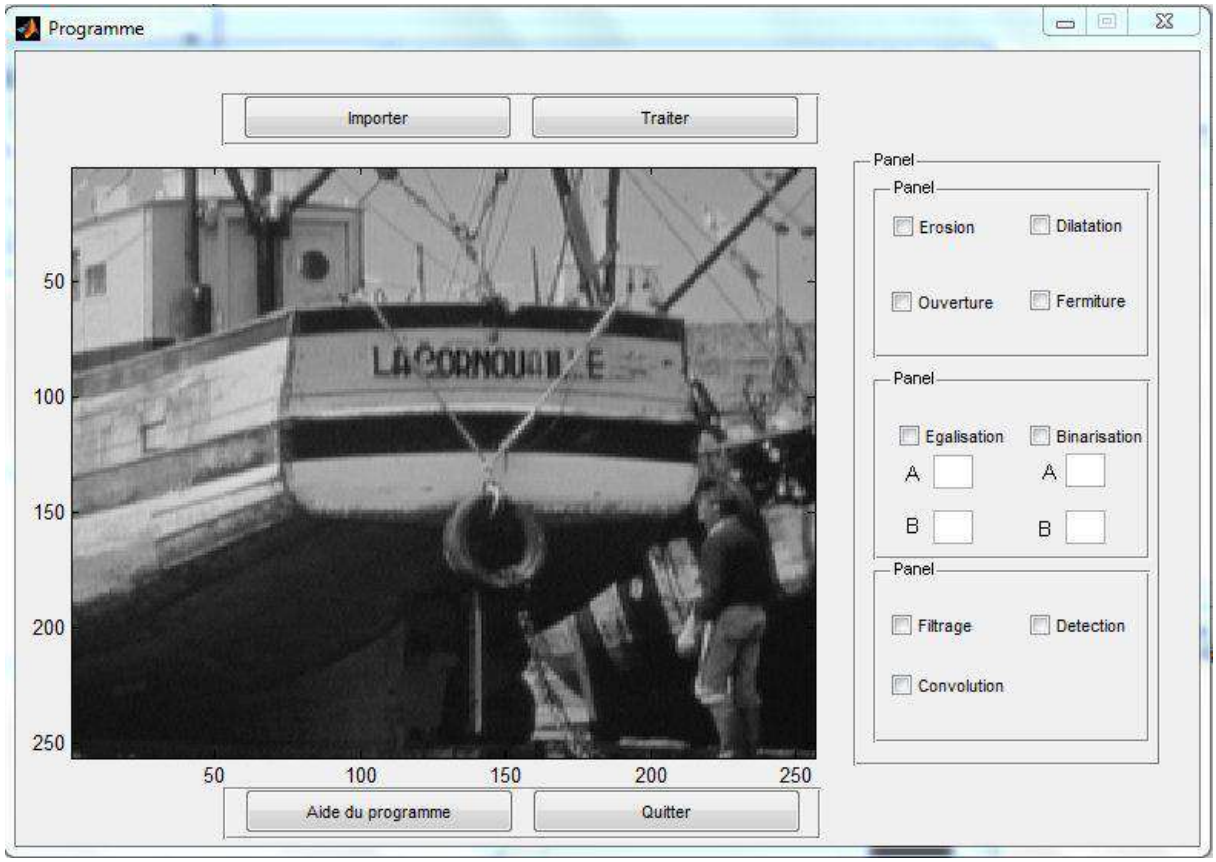


Fig (II.10): Chargement de l'image à traiter.

L'utilisation du bouton (importer) importe l'image qu'elle sera afficher sur l'interface. Après, on sélectionne la méthode et\ou l'opération pour le traitement désiré sur l'image initiale. Par exemple, on peut choisir une méthode parmi les suivantes: (érosion, fermeture, filtrage et détection des contours). Fig(II.11).

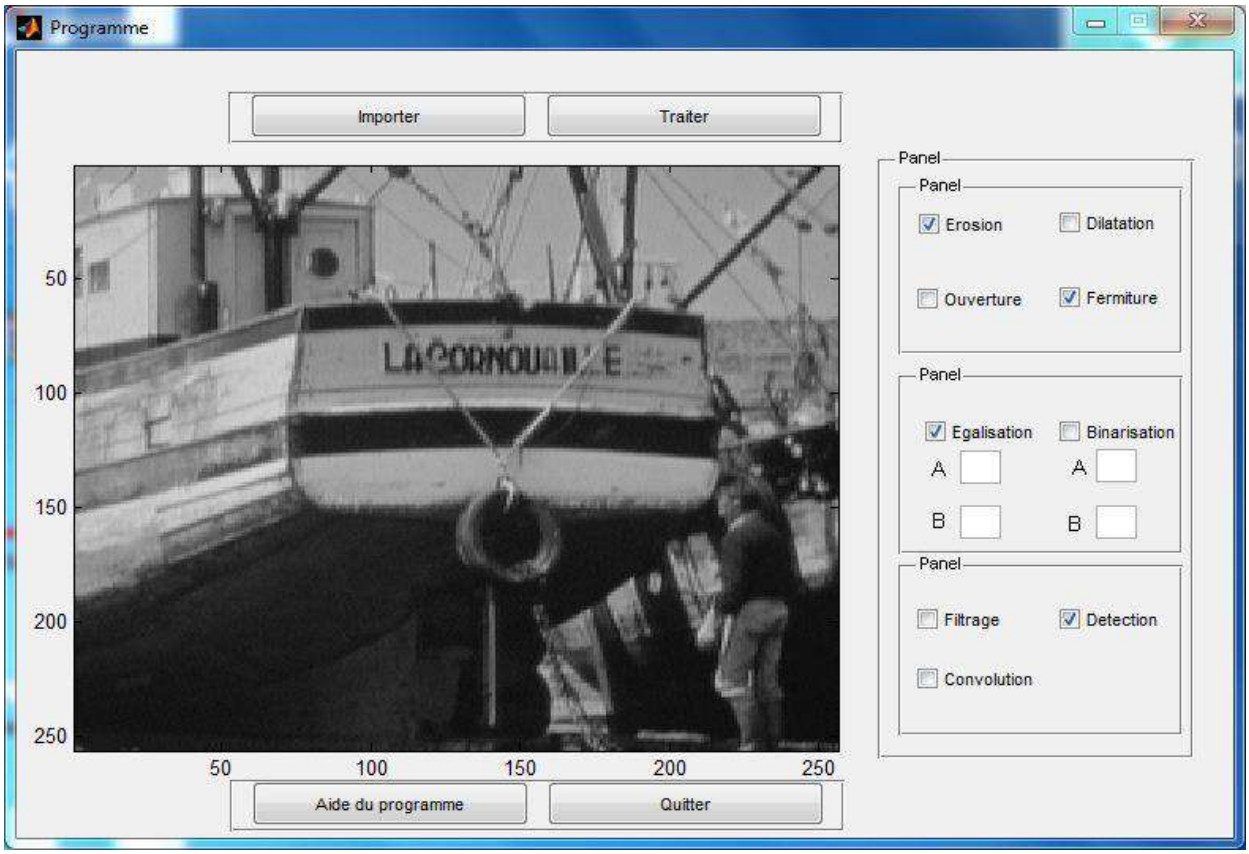


Fig (II.11) : sélection du traitement désiré.

L'étape suivante et de cliquer sur le bouton traiter, les résultats sont fournées. Chacune des images traitées s'affichent dans des figures séparées.

On a choisi une image du bateau de taille 256x256. La figure II.12 montre les résultats des images correspondantes aux traitements disponibles dans cette interface réalisée.



Fig (II.12-a) : image initiale.



Fig(II.12-b) : image érodée.



Fig (II.12-c) : image dilatée.



Fig (II.12-d) : image ouvrée.



Fig (II.12-e) : image fermée.



Fig (II.12-f) : image égalisée.

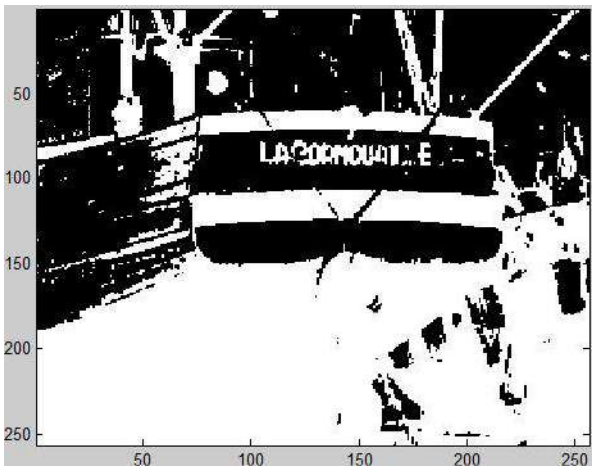


Fig (II.12-g) : image binarisée (100-200).



Fig (II.12- h) : image filtrée.

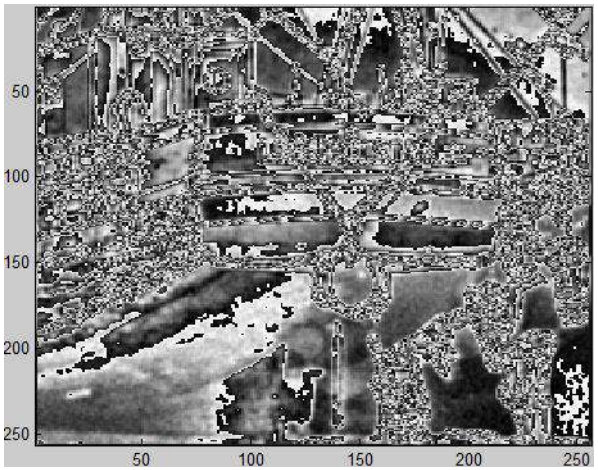
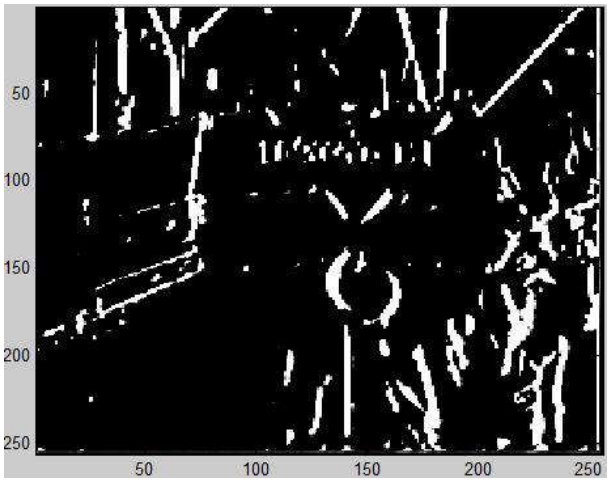


Fig (II.12-i) : image convoluée.



Fig(II.12-j) : image détectée(Sobel).

Fig (II.12) : résultats de simulations utilisant Matlab.

II.7 Conclusion :

Dans ce chapitre on a décrit quelques techniques de prétraitement et traitement d'image. Ensuite on a présenté l'interface qui regroupe quelques techniques implémentées sous Matlab ainsi que ses résultats de simulations.

Chapitre III :

Implémentation des algorithmes de traitement d'image en langage VHDL

III.1 Introduction :

Dans ce chapitre on va présenter la conception, la simulation et l'implémentation sur un circuit FPGA de quelques algorithmes de traitement d'image discutés au chapitre précédent.

Et avant de commencer, on a besoin de quelques algorithmes complémentaires.

C'est un travail qui suit un concept globale hiérarchique composé de :

- une fenêtre mobile3X3 des données de l'image en temps réel en langage VHDL
- un banc d'essai de test (simulation) en langage VHDL
- deux interfaces MATLAB pour les fichiers d'entrée/sortie

La plupart des concepteurs des systèmes numériques trouve plus pratique de développer d'une manière hiérarchique en langage VHDL Ceci est fait pour faciliter la réutilisation du code.

En plus, l'un des concepts principaux de ce projet a été l'élaboration d'un code VHDL qui est en grande partie indépendante du circuit, ce qui signifie que la plupart du code peut être compilé pour n'importe architecture FPGA.

Donc, l'utilisation des ressources matérielles et en particulier les unités de mémoires est limité pour atteindre l'interchangeabilité du code vis-à-vis le type du circuit ciblé.

Pour ces motifs, la seule spécifique du code VHDL relié au matériel se trouve dans les unités de mémoire FIFO trouvés dans le générateur de fenêtre 3x3 décrit dans le prochain paragraphe.

En plus, on a utilisé des ports maps en VHDL qui permettent le câblage des signaux VHDL à partir du niveau actuel de la hiérarchie à l'autre. Cela signifie qu'un algorithme peut référencer différents algorithmes VHDL, ce qui permet la réutilisation du code. La figure *(III.1)* montre un exemple de la façon dont cela est fait en VHDL.

```
entity AlgorithmA is
   port (...);
end AlgorithmA;

architecture AlgorithmA_arch of AlgorithmA is

   component SubAlgorithmA
   port (...);
   end component SubAlgorithmA;

   component WindowGenerator
   port (...);
   end component WindowGenerator;

   component RowColumnCounter
   port (...);
   end component RowColumnCounter;

begin

   SubAlgorithmAMap: SubAlgorithmA
   port map (...);

   WindowGeneratorMap: WindowGenerator
   port map (...);

   RowColumnCounterMap: RowColumnCounter
   port map (...);

   -- algorithm process

end AlgorithmA_arch;
```

Fig (III.1): Mapping des composants VHDL.

III.2 Architecture de la fenêtre 3x3:

Afin de mettre en œuvre un système de fenêtre glissante en VHDL, on a conçu un modèle qui profite de certaines caractéristiques des circuits FPGA. Un FPGA généralement gère facilement des bascules, mais l'instanciation de la mémoire sur ce circuit est plus difficile.

On a choisi que la sortie de l'architecture de la fenêtre doit être un vecteur de pixels, et contient aussi un signal de validation de données, qui est utilisé pour informer l'algorithme utilisant cette fenêtre de l'instant où les données sont prêtes à être traitées.

Comme il a été jugé nécessaire pour atteindre une performance maximale et une surface du circuit relativement restreint, des unités FIFO spécifiques du circuit FPGA ont été utilisés.

Une taille d'une fenêtre 3x3 a été choisie parce qu'elle était assez petite pour être facilement implémenter sur la cible FPGA, et elle est considérée comme assez importante pour être efficace pour les tailles d'image les plus couramment utilisés. Avec des tailles de fenêtres plus grandes, plus de FIFO et de bascules doivent être utilisés, ce qui augmente les ressources FPGA utilisées de manière significative. La figure *(III.2)* montre une représentation graphique de l'architecture de la mémoire FIFO et les bascules utilisés pour cette conception.

L'annexe présente le code source VHDL pour l'unité de génération de la fenêtre. Les codes pour chaque entité FIFO ne sont pas inclus, qui ont été générés par les outils du constructeur.

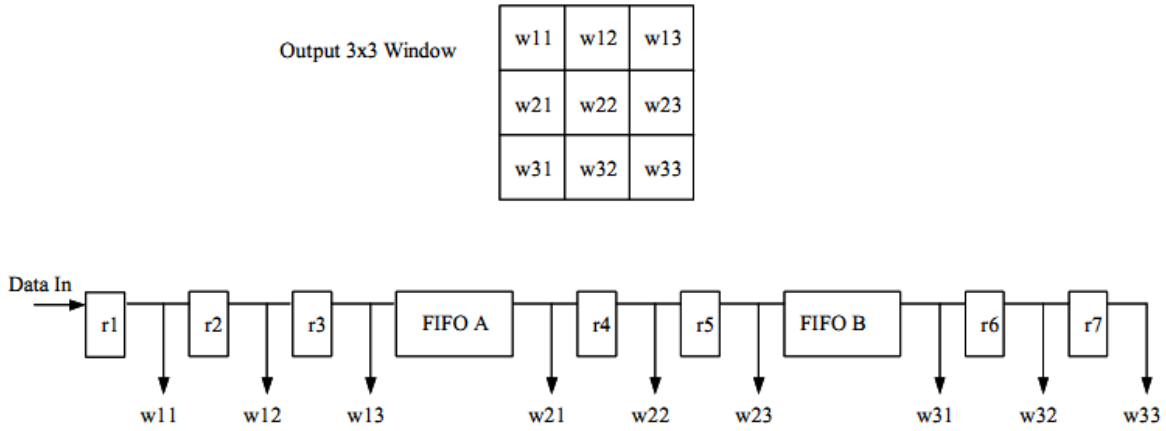


Fig (III.2): Architecture du générateur de la fenêtre.

III.3Le modèle VHDL de test Bench:

Afin d'examiner le fonctionnement correct du code VHDL on a utilisé l'outil logiciel de simulation « Modelsim ». Le simulateur prend le code VHDL et le simule comme il fonctionnerait dans le matériel. Pour ce faire, le concepteur doit fournir au simulateur des entrées valides pour produire les résultats attendus.

Une méthode efficace et commune de simulation du code VHDL est l'utilisation d'un type particulier de code VHDL appelé un banc d'essai (test bench). Ce banc d'essai est efficace à entourer le code VHDL dont le concepteur souhaite simuler et aussi de stimuler l'entité testé.

Un banc d'essai pour un algorithme est responsable de la stimulation des signaux essentiels d'entrée à cet algorithme. Les modèles utilisés dans cette approche sont toutes synchrones, un signal d'horloge doit être

stimulé. En outre, tous les algorithmes conçus fourni une fonctionnalité de réinitialisation (signal de reset), ce qui permet aux algorithmes à effacer leur contenu à tout moment. En plus, étant donné que tous les algorithmes de ce projet lisent des images d'entrée et écrivent des images en sortie, une méthode d'entrée de données et de sortie doit être fournie pour la simulation fonctionnelle.

Lorsque l'on souhaite traiter des images avec un algorithme VHDL, on doit d'abord créer un banc d'essai qui peut lire le fichier contenant ces données. Si l'on veut voir l'image traitée, une autre caractéristique doit être incluse dans le banc d'essai pour permettre l'écriture du fichier de sortie. Ces caractéristiques sont essentielles pour la facilité d'utilisation des bancs d'essais utilisés dans le projet, et ils sont très utiles lorsqu'ils sont associés à un programme tel que MATLAB, ce qui assure une représentation efficace des images et des capacités de visualisation. Cette fonctionnalité est décrite dans la prochaine section de ce chapitre. Un banc d'essai pour le fichier de filtrage « ro_filt_3x3.vhd », nommé « ro_filt_3x3_TB.vhd », se trouve en annexe.

III.4 Les programmes MATLAB des fichiers d'entrée/sortie:

Afin de traiter les données provenant d'images réelles dans les simulations VHDL, il est nécessaire de créer une méthode de transfert d'images dans un format standard, qu'on peut le lire par un processus écrit en langage VHDL. Les fonctions de lecture/écriture du langage VHDL fonctionnent facilement avec des fichiers constitués de données sur chaque ligne (vecteur colonne), cette méthode a été choisie. MATLAB est utilisé pour implémenter cette fonctionnalité, car il est très efficace dans la manipulation des données matricielles, telles que les images.

Un fichier MATLAB (m-file) appelé « m2vhdl.m » a été créé pour lire un fichier d'entrée à n'importe quel format d'image reconnu par Matlab, et le convertir en un fichier constitué de données sur chaque ligne.

Les données de ce format peuvent être lues dans le banc d'essai VHDL à l'aide de standards fonctions textes d'entrée/sortie VHDL. Après que le simulateur traite ces données (effectivement, traitée par l'algorithme en VHDL), les données de sortie de l'algorithme est écrit par le banc d'essai dans un fichier de sortie de même format que le fichier d'entrée.

Ensuite, un autre fichier MATLAB a été écrit pour lire ces données et la reconvertir en une matrice dans MATLAB. Cette routine, appelé « vhdl2m.m », permet l'analyse et la comparaison des images traitées par le programme VHDL avec le programme de traitement écrit en langage MATLAB du chapitre précédent. Cette analyse est essentielle pour déterminer la fonctionnalité correcte.

Le code source de ces deux fichiers Matlab se trouve en annexe.

III.5 Données utilisées :

Dans ce projet on a utilisé des images de taille 128 par 128 avec 8 bits de niveaux de gris. Pour tout système de traitement d'image en temps réel, la taille des données utilisées affecte beaucoup les performances du système réalisé (temps de traitement), ainsi que le nombre de ressources utilisées sur le circuit FPGA.

Heureusement, le code VHDL pour ce projet est rédigé de telle manière à faciliter l'utilisation de différentes largeurs de bits. Cela peut être fait en remplaçant les éléments FIFO dans la conception « window_3x3.vhd » et en changeant le paramètre générique « vwidth » dans tous les modèles VHDL.

III.6 Mise en œuvre en VHDL de l'algorithme de filtre d'ordre par tri de données:

L'objectif de ce projet est la mise en œuvre d'un algorithme sur un circuit FPGA. Cela a été accompli en modélisant cet algorithme dans le langage VHDL et après on fait la simulation, la synthèse et l'implémentation de ce dernier.

Le filtre d'ordre par tri utilise le générateur de fenêtre des pixels 3x3. Son fonctionnement est assez simple. Comme indiqué au chapitre précédent, le filtre doit d'abord trier les valeurs des pixels dans une fenêtre dans l'ordre croissant. La méthode la plus efficace du point de vue matériel est l'utilisation des unités de comparaison et de tri, qui permettent de trier une fenêtre de neuf pixels dans une liste ordonnée pour une utilisation dans le filtre.

On a mis en œuvre la structure représenté par la figure *(III.3)*. Elle donne en sortie une liste triée après un délai de 14 cycles d'horloge. Comme cette structure fonctionne en pipeline, après le délai initial, le système produit une liste triée valide à chaque cycle d'horloge.

L'algorithme VHDL qui met en œuvre cette conception « sort_3x3.vhd », est composé d'une série de registres et de comparateurs, comme le montre la figure *Fig (III.4)*. Le code source du modèle « sort_3x3.vhd » se trouve en annexe.

Chaque bloc r_{XX} est un registre et chaque bloc c_{XX} est une unité de comparaison, qui consiste en une simple décision. Cette conception peut être facilement écrite en VHDL en utilisant l'instruction if / else comme suit :

```
If (Wx1 < Wx2) then
    Cx1_L <= Wx1;
```

```
        Cx1_H <= Wx2;
Else
        Cx1_L <= Wx2;
        Cx1_H <= Wx1;
End if;
```

Fig(III.3) : principe de comparaison utilisé.

L'organigramme qui correspond à l'opération de tri croissant comporte douze étapes de comparaison, l'organigramme représenté par la figure (*III.3)* illustre seulement les neuf premiers étapes de l'algorithme.

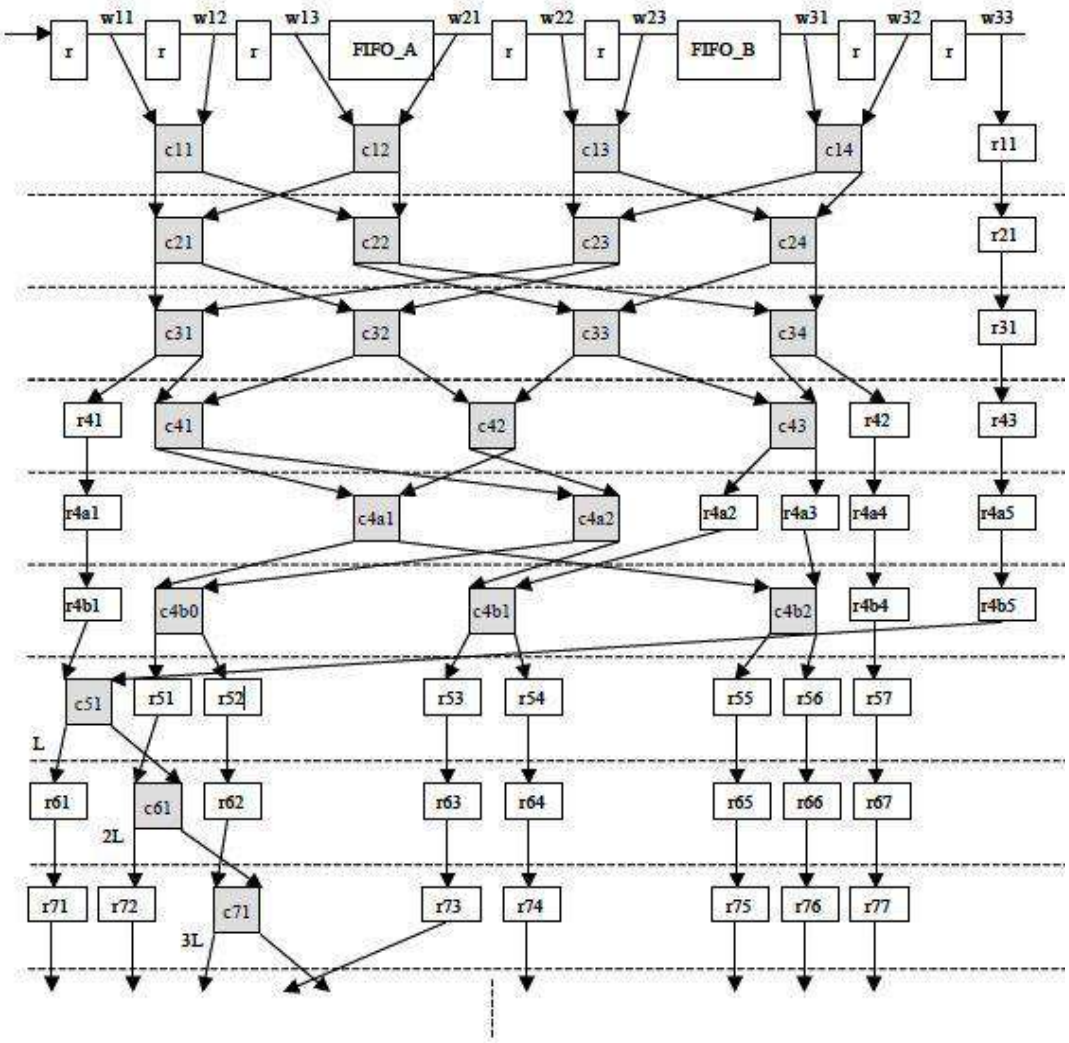


Fig (III.4) : l'organigramme de l'algorithme de tri.

Après que la liste triée est généré avec l'entité VHDL « sort_3x3 », l'algorithme décrivant la fonctionnalité du filtre « ro_filt_3x3.vhd », peut fonctionner sur cette liste pour produire en sortie les données de l'image filtrée. Comme il est mentionné auparavant, le filtre par tri donne une valeur du pixel dans l'emplacement d'origine comme spécifié par le rang du filtre.

Pour faire ce balayage de l'image correctement, un compteur doit être utilisé pour indiquer quand il faut changer l'état du signal de sortie de validation de données. Étant donné qu'il est souhaitable que l'image de sortie aura la même taille que l'image d'entrée, et l'utilisation du générateur de fenêtre permet de réduire efficacement la quantité de données valides de sortie, des pixels de valeurs zéro doivent être placés autour des frontières de l'image.

Afin de faire cela correctement, des compteurs sont utilisés pour indiquer à l'algorithme quand les frontières commencent. Un compteur VHDL a été écrit afin de compter le mouvement des pixels. Puisque les images sont des données en deux dimensions, deux compteurs sont nécessaires: l'un pour compter les lignes et l'autre pour compter les colonnes dans l'image. L'entité VHDL qui implémente cette fonctionnalité est appelée « rc_counter.vhd » et se trouve en annexe.

Pour que le filtre par tri fonctionne correctement, les trois entités VHDL doit être instancié dans l'algorithme lui-même. Cela se fait avec les instructions standards de composants VHDL et les ports maps. La figure *(III.5)* montre la structure de la conception VHDL utilisée pour cet algorithme de filtrage.

« ro_filt_3x3 » interprète et contrôle les signaux sur les trois entités pour réaliser une conception cohérente, dont le résultat est un filtre d'ordre par tri. L'ordre est spécifié avec un générique VHDL, donc il est uniquement modifiable avant synthèse.

```
entity ro_filt_3x3 is
      port (...);
end ro_filt_3x3;


architecture ro_filt_3x3_arch of ro_filt_3x3 is
      component sort_3x3
      port (...);
      end component sort_3x3;


      component window_3x3
      port (...);
      End component window_3x3


      component rc_counter
      port (...);
      end component rc_counter;
```

Fig (III.5): structure de la conception VHDL de l'algorithme de filtrage.

III.7 Résultats de la simulation fonctionnelle :

III.7.1 Résultats de l'algorithme du filtre médian :

On a simulé notre conception en VHDL à l'aide du logiciel Modelsim. Ensuite on a comparé le résultat de la structure matérielle (figure III-6a) avec celle obtenu avec Matlab (Figure III-6b). Dans ce test, l'algorithme implémenté est celle du filtrage par tri d'ordre 5 (filtre médian).



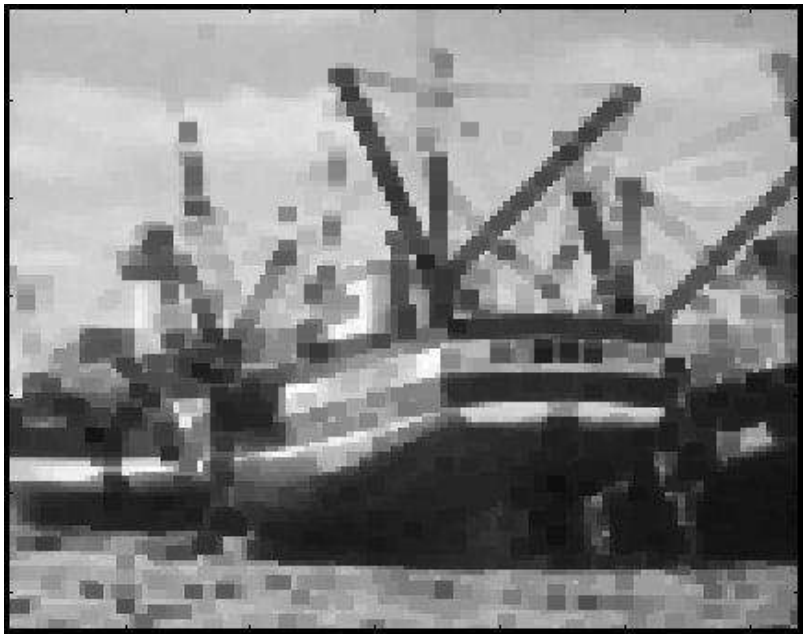
Fig(III.6-a) : image filtrée en utilisant matlab.



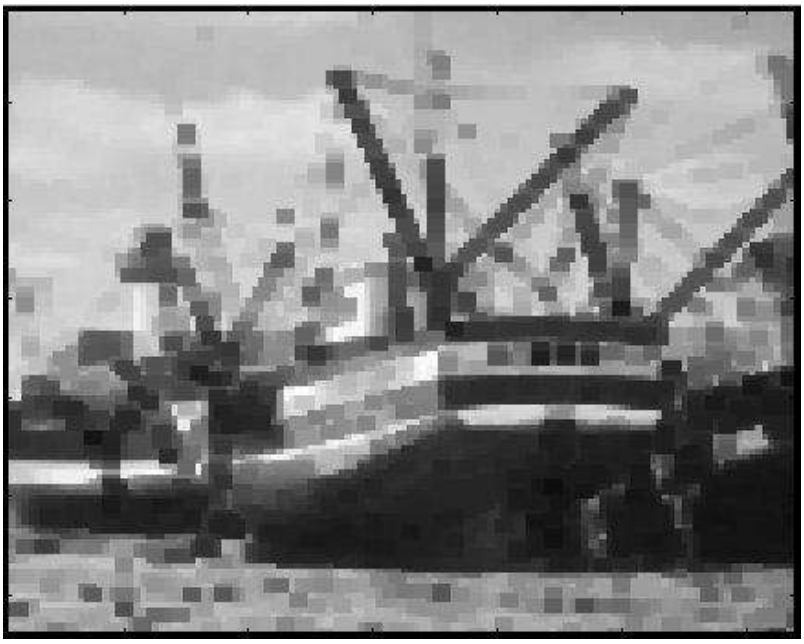
Fig(III.6-b) : image filtrée en utilisant le VHDL.

III.7.2 Résultats de la simulation fonctionnelle des opérations morphologiques :

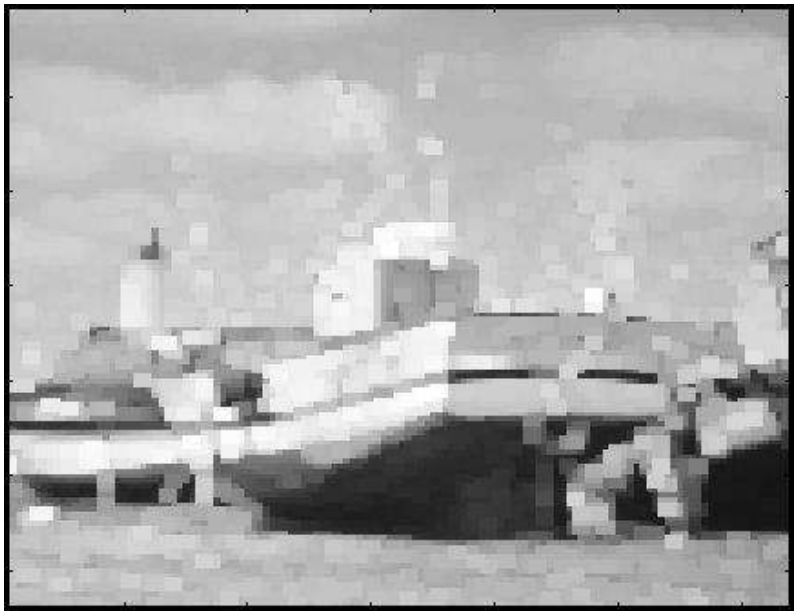
Ces opérations se basent sur un critère principal qu'est le choix d'élément structurant, dans ce travail on a utilisé l'élément structurant carrée d'une fenêtre de 3x3. Par rapport aux deux opérations d'érosion et dilatation, elles sont représentées comme nous avons vu précédemment (chapitre II) par des filtres d'ordre 1 et d'ordre 9 respectivement, en outre l'ouverture est une dilatation d'une image érodée, et la fermeture est une érosion d'une image dilatée.



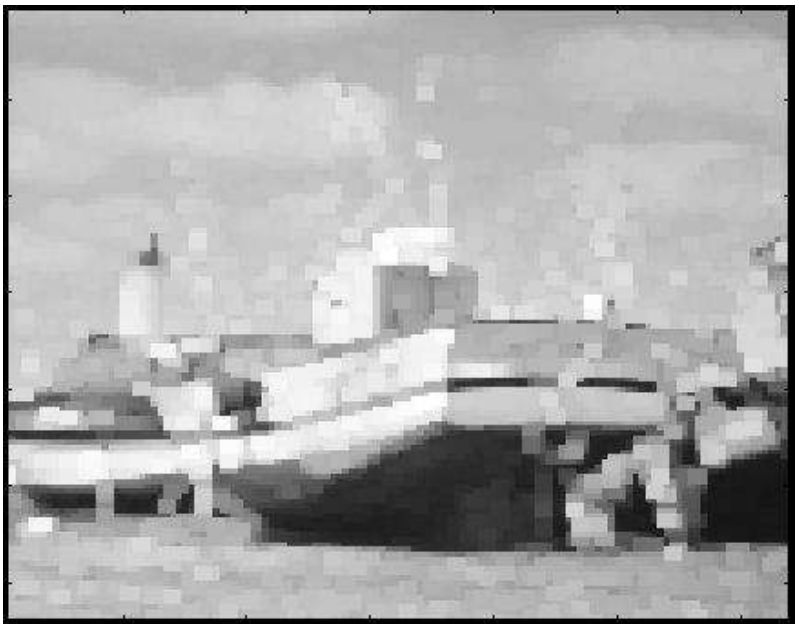
Fig(III.7-a) : image érodée en matlab.



Fig(III.7-b) : image érodée en VHDL.

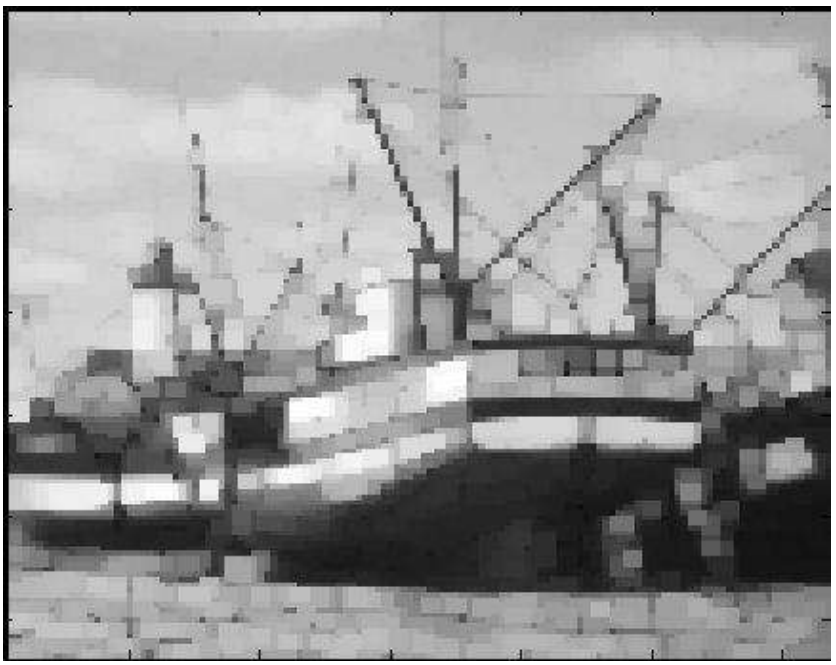


Fig(III.7-c) : image dilatée en matlab.

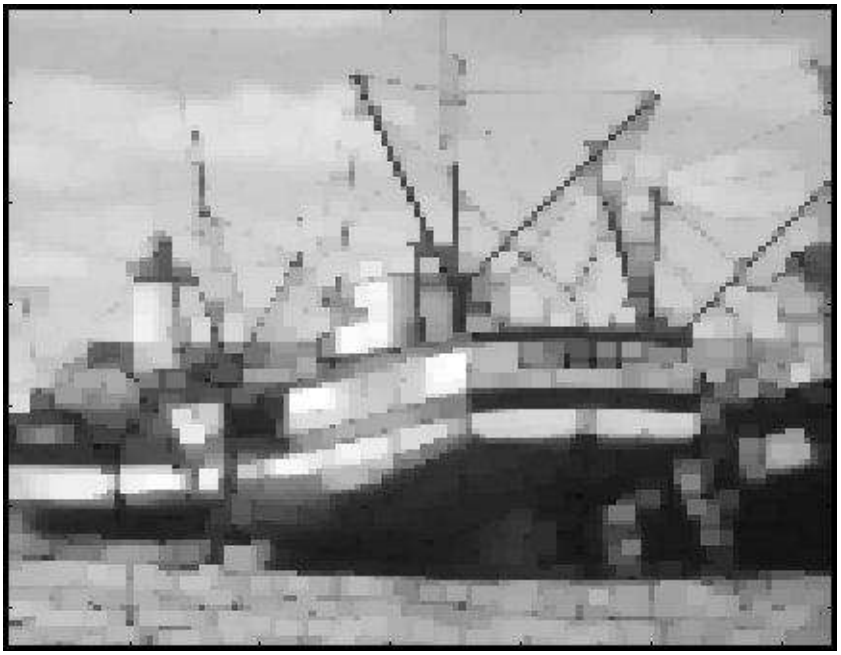


Fig(III.7-d) : image dilatée en VHDL.

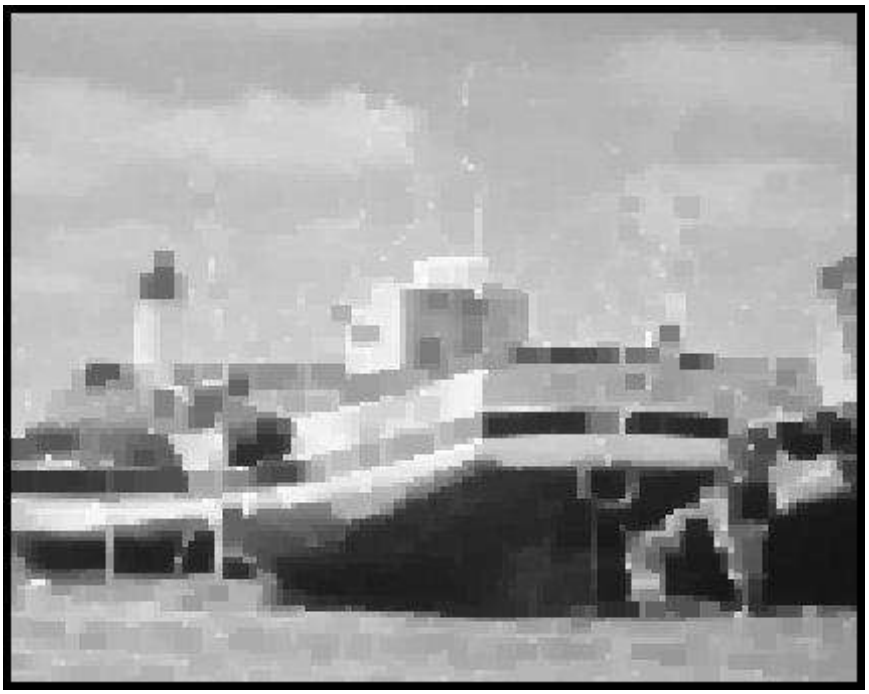
En VHDL pour l'ouverture et la fermeture on fait pour chaque opération une double simulation telle que la première prend l'image initiale (originale), et la deuxième prend comme entrée l'image résultante de la première simulation. Les résultats sont représentés dans les figures *Fig(III.8-a -III.8-d)*.



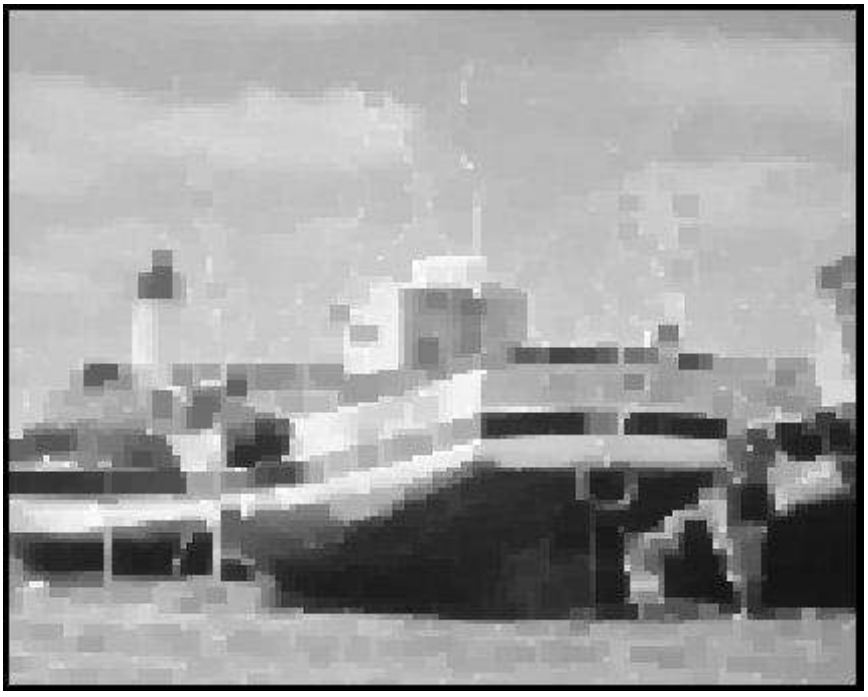
Fig(III.8-a) : image ouvrée en matlab.



Fig(III.8-b) : image ouvrée en VHDL.



Fig(III.8-c) : image fermée en matlab.



Fig(III.8-d) : image fermée en VHDL.

III.8 Résultats de la synthèse et de l'implémentation :

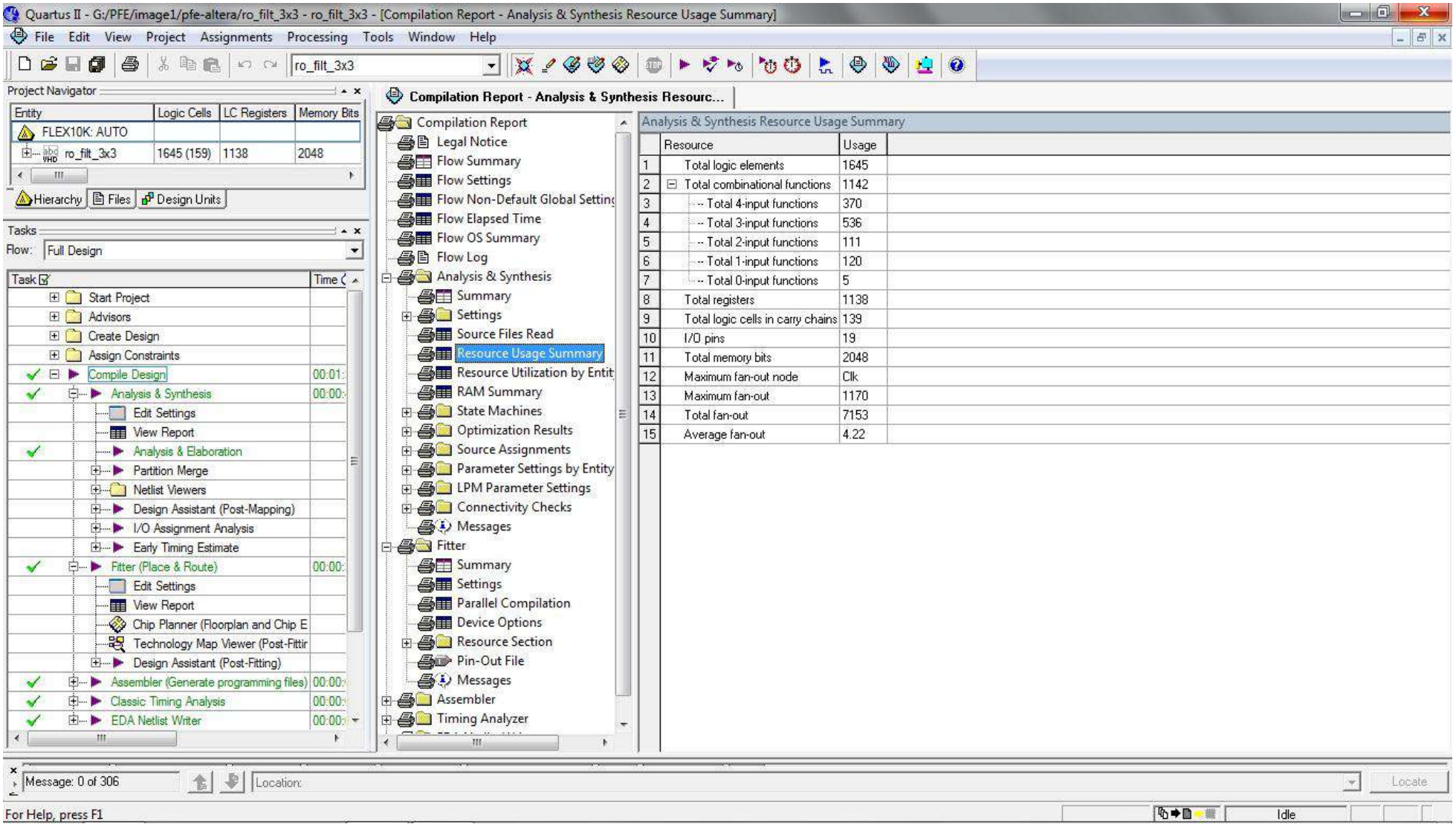
Afin de continuer les étapes de la conception, on a utilisé le logiciel "QUARTUS II v 9.0" de la société ALTERA. On a ciblé un circuit d'Altéra, de la famille FLEX10K. On a choisi ce circuit dans le but de continuer ce projet dans le futur en utilisant le KIT disponible au niveau du laboratoire de notre département qui contient le même circuit.

La figure III.8 montre le résultat de la synthèse de l'algorithme de filtrage par tri. On remarque que l'occupation des éléments logiques est de 71%, la consommation de 2048 bits de mémoire (2 FIFO : 2X128X8bits=2048), et 19 broches d'entrées/sorties (D:8bits + Dout:8 bits + clk: 1bit +reset:1bit + DV:1bit).

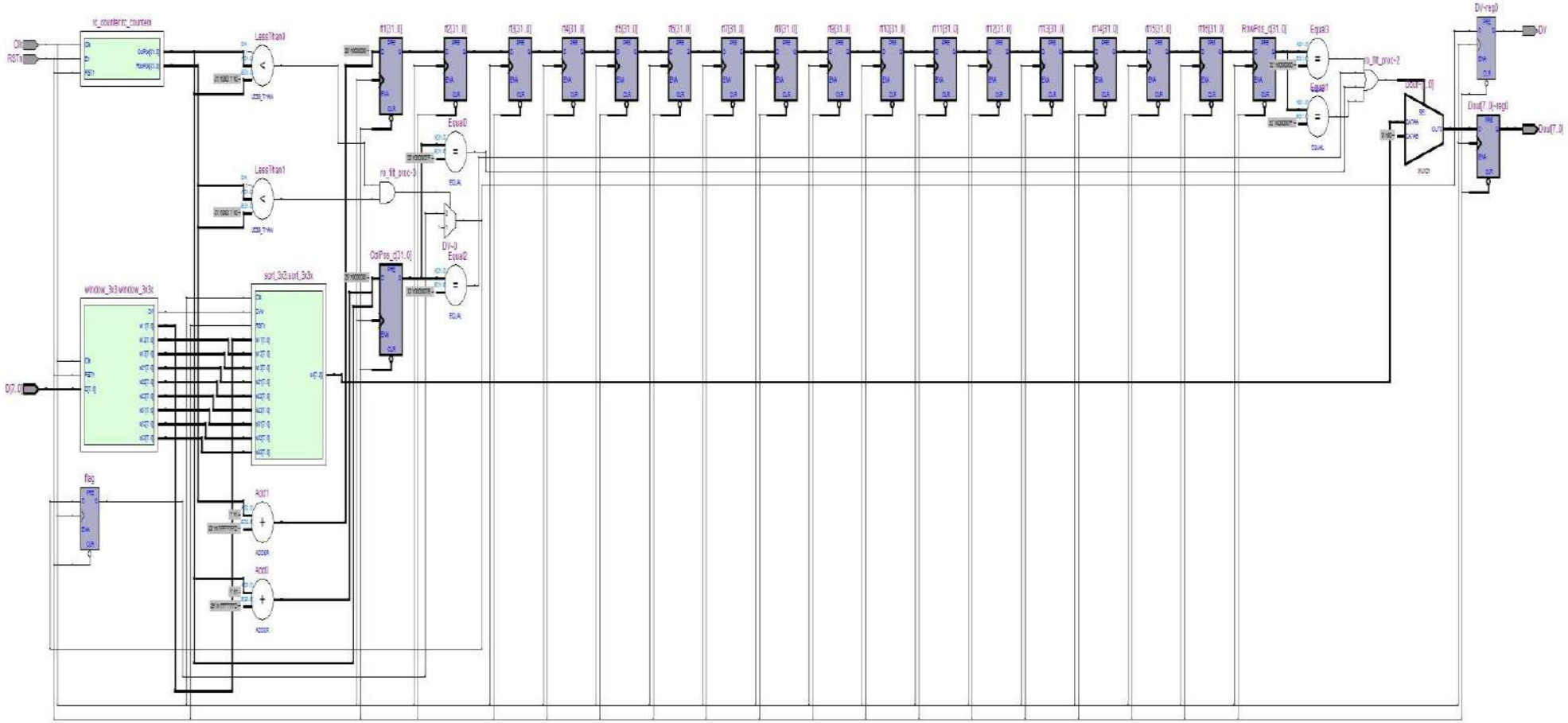
Fitter Status	Successful - Sun Sep 09 22:53:11 2012
Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	ro_filt_3x3
Top-level Entity Name	ro_filt_3x3
Family	FLEX10K
Device	EPF10K40RC208-3
Timing Models	Final
Total logic elements	1,645 / 2,304 (71 %)
Total pins	19 / 147 (13 %)
Total memory bits	2,048 / 16,384 (13 %)

Fig(III.9) : résultat de synthèse de l'algorithme de filtrage.

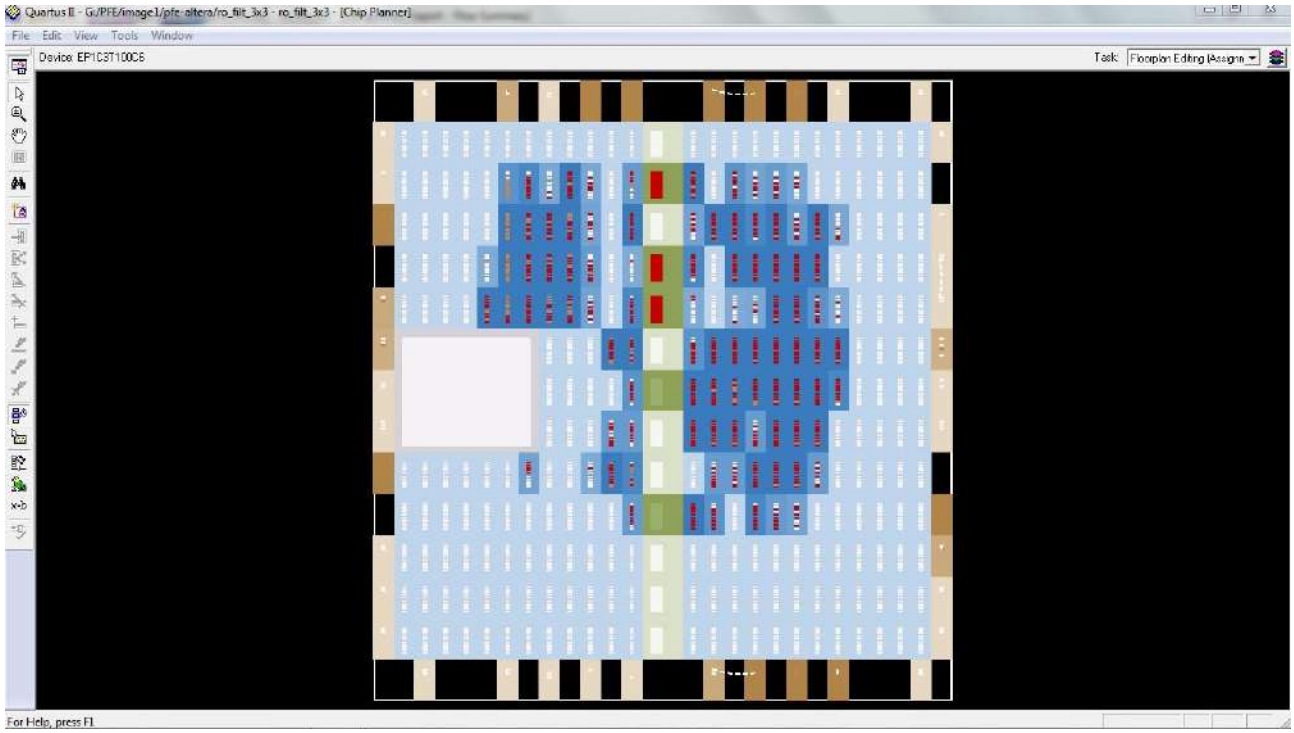
La figure III.10 montre l'environnement du logiciel avec le projet conçu. On peut visualiser le schéma du niveau RTL après synthèse avec le "RTL viewer" comme le montre la figure III.11. Tandis que la figure III.12 représente le "floorplan" du circuit FPGA après implémentation (placement et routage).



Fig(III.10) : l'environnement intégré du logiciel Quartus II v 9.0 chargé du projet.



Fig(III.11) : Schémas du niveau RTL du système de filtrage.



Fig(III.12) : floor plan du système implémenté

III.9 Conclusion :

Dans ce chapitre on a présenté les étapes de notre implémentation sur un circuit FPGA d'une algorithme de traitement d'image. On a commencé par la description de la conception matérielle. Ensuite on a vérifié son fonctionnement en comparant les résultats obtenus en utilisant les deux logiciels: Modelsim pour le modèle en VHDL et Matlab. Après, on a passé à l'étape de synthèse logique en se servant du logiciel d'ALTERA: le QUARTUS II. Et finalement on a implémenté l'algorithme sur un circuit de type FLEX10K.

Conclusion générale

Le développement d'algorithmes de traitement d'image sur un circuit FPGA est assez complexe, mais les résultats obtenus sont très satisfaisants. Dans ce mémoire on a présenté une solution matérielle rapide pour l'implémentation d'un algorithme de fenêtrage, ceci est facilité par l'utilisation de technologie FPGA.

En effet, avec l'aide du générateur de fenêtre, on peut réaliser plusieurs techniques de traitement d'images, dont beaucoup peuvent être synthétisés pour des applications nécessitant un exécution rapide comme dans les systèmes de traitement en temps réel.

Plusieurs connaissances ont été acquises durant la réalisation de ce projet. On peut dire que les FPGA sont excellentes pour un grand nombre d'applications de traitement d'images.

Il convient également de noter que les algorithmes de ce projet étaient d'excellents choix pour la mise en œuvre sur un FPGA, parce qu'ils n'utilisent pas les calculs mathématiques en virgule flottante et ils ne comportent pas de mathématiques complexes.

Et comme perspective, pour certaines applications qui nécessitent un calcul mathématique complexe on peut penser à concevoir un système basé sur un circuit DSP. En effet, le concepteur qui souhaite la meilleure combinaison de rapidité et de souplesse devrait se tourner vers un système mixte constitué des deux circuits: FPGA et DSP. Un tel système peut tirer profit des aspects positifs de chaque architecture, et peut permettre au concepteur de créer un algorithme sur un système qui lui convient le mieux.

Annexe A

Les programmes *matlab* des algoritmes de traitement d'image du deuxième chapitre :

Fichier : image_erode.m : algorithme d'opération d'érosion

```
function im_er=image_erode(im)
[m,n]=size(im);
im_er1(m+2,n+2)=zeros;
im_er2(m,n)=zeros;

im=double(im);
x(3,3)=zeros;
for i=1:m
for j=1:n
im_er1(i+2,j+2)=im(i,j);
end
end
for i=2:m+1
for j=2:n+1
k=i-1;
l=j-1;
x=im_er1((i-1):(i+1),(j-1):(j+1));
im_er2(i,j)=min(min(x));
end
end
for i=1:m
for j=1:n
im_er3(i,j)=im_er2(i+1,j+1);
end
end
im_er=uint8(im_er3);
figure
colormap(gray(256));
image(im_er)
title('erosion');
```

Fichier : image_dilate.m : algorithme d'opération de dilatation

```
function im_dil=image_dilate(im)
[m,n]=size(im);
im_dil1(m+2,n+2)=zeros;
im_dil2(m,n)=zeros;

im=double(im);
x(3,3)=zeros;
for i=1:m
    for j=1:n
        im_dil1(i+2,j+2)=im(i,j);
    end
end
    for i=2:m+1
    for j=2:n+1
         k=i-1;
         l=j-1;
            for k=i-1:i+1
            for l=j-1:j+1
```

```
                x(k-i+2,l-j+2)=im_dil1(k,l);
            end
        end
        im_dil2(i,j)=max(max(x));
        end
        end
 for i=1:m
 for j=1:n
   im_dil3(i,j)=im_dil2(i+1,j+1);
 end
 end
im_dil=uint8(im_dil3);
figure
colormap(gray(256));
image(im_dil)
title('delatation');
```

Fichier : image_egal.m : algorithme d'égalisation

```
function im_egal=image_egal(im,a,b)
[x,y]=size(im);
im=double(im);
im_egal(x,y)=zeros;
nmax=max(max(im));
nmin=min(min(im));
m=max(a,b);
n=min(a,b);
  for i = 1 : x
    for j = 1 :y
        if im(i,j)<= n
            im_egal(i,j)=n;
        elseif im(i,j)>= m
            im_egal(i,j)=m;
        else
            im_egal(i,j)=round(n+(im(i,j)-nmin)*(m-n)/(nmax-nmin));
        end
    end
  end
  im_egal=uint8(im_egal);
  figure;
  colormap(gray(256));
  image(im_egal);
  title('egalisation');
```

Fichier : image_binaire.m : algorithme de binarisation

```
function im_bin=image_binaire(im,a,b)
[x,y]=size(im);
im=double(im);
im_bin(x,y)=zeros;
m=max(a,b);
n=min(a,b);
  for i = 1 : x
    for j = 1 :y
        if im(i,j)<= n || im(i,j)>= m
            im_bin(i,j)=255;
        else
            im_bin(i,j)=0;
```

```
        end
    end
  end
  im_bin=uint8(im_bin);
  figure;
  colormap (gray(256));
  image(im_bin);
  title('binarisation');
```

Fichier : image_convo.m : algorithme de convolution

```
function c=image_convo(im)
mask1=[1 2 1;2 4 2;1 2 1];
[x,y]=size(im);
im=double(im);
c(x,y)=zeros;
  for i = 1 : x
    for j = 1 :y
     if ( i==1 || i==x || j==1 || j==y )
       c(i,j)=im(i,j);
     else
         s=0;
      for k=i-1 : i+1
          for l =j-1 :j+1
              s=s+(im(k,l)*mask1(k-i+2,l-j+2));
          end
      end
        c(i,j)= mod(s,255);
        end ;
        end ;
        end ;
        c=uint8(c);
        figure;
        colormap(gray(256));
        image(c);
        title('convolution');
```

Fichier : image_filtre.m: algorithme de filtrage

```
function im_filter=filtre(im)
order=5;
[n,m]=size(im);
im_filter(1:n,1:m)=zeros;

im_d=double(im);
for i=1 :n
    for j=1:m
       im_filter(i,j)=im(i,j);
    end
end
for i=1:n-2
for j=1:m-2
    win=[im(i:(i+2),j:(j+2))];
    vec=[[win([1,1:3])] [win([2,1:3])] [win([3,1:3])]];
    kip=sort(vec);
im_filter(i+1,j+1)=kip(order);
%kip(order);
end
end
```

```
im_filter=uint8(im_filter);
figure;
colormap(gray(256));
image(im_filter);
title('filtrage');
```

Fichier : sobel.m: algorithme de détection de contours Sobel

```
  function im_det=sobel(a)
d=[1 0 -1;15 0 -15;1 -3 -1];
[n,m]=size(a);
b=zeros(n+4,m+4);
for i=3:n+2
    for j=3:m+2

        b(i,j)=a(i-2,j-2);

    end
end
[x,y]=size(b);
ab=double(b);
c=double(b);
det=double(d);

for t=3:x-3
    for v=3:y-3
             s=0;
                    for l=0:2
                        for k=0:2

                            for i=0:2
                                  for j=0:2
                                      w=t+l-i;
                                     f=v+k-j;
                                     p=(ab(w,f)*det(3-i,3-j));
                                   s=s+p;
                                  end
                            end

                        end
                    end
             c(t,v)=s;
    end
end
cac=uint8(c);
im_det=zeros(n,m);
for i=1:n
    for j=1:m

           im_det(i,j)=cac(i+3,j+3);

    end
end
figure
colormap(gray(256));
image(im_det)
title('detection');
```

Annexe B

Fichier m2vhdl.m

Programme matlab de lecture de l'image et sa conversion vers un fichier colonne.

```
load boat128;
J = int16(I);
K = double(J);
K = K';
M = reshape(K,128*128,1);
fid = fopen('output_bin.bin','wb');
fprintf(fid,'%d\n',M);
fclose(fid);
```

Fichier vhdl2m.m

Programme matlab de conversion du fichier résultat de traitement en vhdl vers un fichier image et affichage de l'image originale ainsi que l'image traitée.

```
close all;
fid = fopen('vhdl_output.bin');
[I1,cnt] = fscanf(fid,'%d',inf);
fclose(fid);
I1 = reshape(I1,128,128);
I1 = I1';
figure;
Cmap = gray(256);
Colormap(Cmap);
imagesc(I);
figure;
Cmap = gray(256);
Colormap(Cmap);
imagesc(I1);
```

Fichier `window_3x3.vhd`, le programme VHDL : générateur de fenêtre de 3x3

```
library IEEE;
use IEEE.std_logic_1164.all;
library altera_mf;
use altera_mf.altera_mf_components.all;

entity window_3x3 is
  generic (
    vwidth: integer:=8
    );

  port (
    Clk  : in std_logic;
    RSTn  : in std_logic;
```

```
    D  : in std_logic_vector(vwidth-1 downto 0);  -- l'entrée de la fenêtre
à  --chaque front descendant d'horloge en prend la valeur de D et on l'ajout à
la  --fenetre de sortie si la valeur de DV est à 1 (bit de validation de la
sortie);
    w11  : out std_logic_vector(vwidth-1 downto 0);  -- les valeurs des
pixels de la fenetre;
    w12  : out std_logic_vector(vwidth-1 downto 0);
    w13  : out std_logic_vector(vwidth-1 downto 0);
    w21  : out std_logic_vector(vwidth-1 downto 0);
    w22  : out std_logic_vector(vwidth-1 downto 0);
    w23  : out std_logic_vector(vwidth-1 downto 0);
    w31  : out std_logic_vector(vwidth-1 downto 0);
    w32  : out std_logic_vector(vwidth-1 downto 0);
    w33  : out std_logic_vector(vwidth-1 downto 0);
    DV   : out std_logic:='0' -- bit de la validation se change selon un
compteur sur les mémoires FIFO;
  );
end window_3x3;
architecture window_3x3 of window_3x3 is
  component scfifo
    generic (
      lpm_width: natural:=8;
      lpm_widthu: integer:=7;
      lpm_numwords: integer:=128
    );
  PORT
  (
    data      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    wrreq     : IN STD_LOGIC ;
    rdreq     : IN STD_LOGIC ;
    clock     : IN STD_LOGIC ;
    aclr      : IN STD_LOGIC ;
    q         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    full      : OUT STD_LOGIC ;
    empty     : OUT STD_LOGIC ;
    usedw     : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
  );
  END component scfifo;
  signal a11  : std_logic_vector(vwidth-1 downto 0);
  signal a12  : std_logic_vector(vwidth-1 downto 0);
  signal a13  : std_logic_vector(vwidth-1 downto 0);
  signal a21  : std_logic_vector(vwidth-1 downto 0);
  signal a22  : std_logic_vector(vwidth-1 downto 0);
  signal a23  : std_logic_vector(vwidth-1 downto 0);
  signal a31  : std_logic_vector(vwidth-1 downto 0);
  signal a32  : std_logic_vector(vwidth-1 downto 0);
  signal a33  : std_logic_vector(vwidth-1 downto 0);
-- les signaux de la mémoire fifoa
```

```
signal clear  : std_logic;
signal wrreqa  : std_logic:='1';
signal rdreqa  : std_logic:='0';
signal ofulla  : std_logic;
signal oemptya : std_logic;
signal ofifoa  : std_logic_vector(vwidth-1 downto 0);
signal ousedwa : std_logic_vector(vwidth-2 downto 0);

-- les signaux de la mémoire fifob
signal wrreqb  : std_logic:='0';
signal rdreqb  : std_logic:='0';
signal ofullb  : std_logic;
signal oemptyb : std_logic;
signal ofifob  : std_logic_vector(vwidth-1 downto 0);
signal ousedwb : std_logic_vector(vwidth-2 downto 0);
signal dwrreqb: std_logic:='0';
  -- signaux de mise à jour pour le signal de validation DV
  signal ddddddddddDV: std_logic:='0';
  signal dddddddddDV: std_logic;
  signal ddddddddDV: std_logic;
  signal dddddddDV: std_logic;
  signal ddddddDV: std_logic;
  signal dddddDV: std_logic;
  signal ddddDV: std_logic;
  signal dddDV: std_logic;
  signal ddDV: std_logic;
  signal dDV: std_logic;
begin
  fifoa: scfifo
    port map (
      data  => a13,
      wrreq  => wrreqa,
      rdreq  => rdreqa,
      clock  => Clk,
      aclr  => clear,
      q   => ofifoa,
      full  => ofulla,
      empty  => oemptya,
      usedw   => ousedwa -- compteur de remplissage de la mémoire fifoa;
    );
  fifob: scfifo
    port map (
      data  => a23,
      wrreq  => wrreqb,
      rdreq  => rdreqb,
      clock  => Clk,
      aclr  => clear,
      q   => ofifob,
```

```
    full  => ofullb,
    empty  => oemptyb,
    usedw   => ousedwb -- compteur de remplissage de la mémoire fifob;
  );
clear <= not(RSTn);
clock: process(Clk,RSTn)
begin
  if RSTn = '0' then
    a11 <= (others=>'0');
    a12 <= (others=>'0');
    a13 <= (others=>'0');
    a21 <= (others=>'0');
    a22 <= (others=>'0');
    a23 <= (others=>'0');
    a31 <= (others=>'0');
       a32 <= (others=>'0');
          a33 <= (others=>'0');
          w11 <= (others=>'0');
          w12 <= (others=>'0');
          w13 <= (others=>'0');
          w21 <= (others=>'0');
          w22 <= (others=>'0');
          w23 <= (others=>'0');
          w31 <= (others=>'0');
          w32 <= (others=>'0');
          w33 <= (others=>'0');
          wrreqa <= '0';
          wrreqb <= '0';
          ddddddddDV <= '0';
        ddddddddDV <= '0';
        dddddddDV <= '0';
        ddddddDV <= '0';
        dddddDV <= '0';
        ddddDV <= '0';
        dddDV <= '0';
        ddDV <= '0';
        dDV <= '0';
        DV <= '0';
elsif rising_edge(Clk) then
          a11 <= D;
        a12 <= a11;
        a13 <= a12;
    a21 <= ofifoa;
    a22 <= a21;
    a23 <= a22;
    a31 <= ofifob;
    a32 <= a31;
    a33 <= a32;
```

```
      w11 <= a11;
      w12 <= a12;
      w13 <= a13;
      w21 <= a21;
      w22 <= a22;
      w23 <= a23;
      w31 <= a31;
      w32 <= a32;
      w33 <= a33;
      wrreqa <= '1';
      wrreqb <= dwrreqb;
      ddddddddDV <= dddddddddDV;
      dddddddDV <= ddddddddDV;
      ddddddDV <= dddddddDV;
      dddddDV <= ddddddDV;
      ddddDV <= dddddDV;
      dddDV <= ddddDV;
      ddDV <= dddDV;
      dDV <= ddDV;
      DV <= dDV;
     end if;
  end process;
  req: process(Clk)
  begin
  if rising_edge(Clk) then
    if ousedwa = "1111011" then -- si la mémoire FIFOA est chargée de plus de
125 ;
      rdreqa <= '1'; -- alors autoriser la lecture de la première valeur de
cette mémoire;
      dwrreqb <= '1';            --et aussi on commence à charger la mémoire
FIFOB;
    end if;
    if ousedwb = "1111011" then -- si la mémoire FIFOB est chargée de plus de
125 ;
      rdreqb <= '1';     -- alors autoriser la lecture de la première valeur
de cette mémoire;
      elsif ousedwb = "1111100" then
      ddddddddddDV <= '1';
    end if;
  end if;
  end process;
end window_3x3;
```

Fichier : `sort_3x3.vhd`, le programme VHDL : d'algorithme de tri croissant.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity sort_3x3 is
  generic (
    vwidth: integer:=8
    );
  port (
    Clk  : in std_logic;
    RSTn  : in std_logic;
    w11    : in std_logic_vector((vwidth-1) downto 0); -- les 9 valeurs
d'entrée du circuit de tri croissant;
    w12   : in std_logic_vector((vwidth-1) downto 0);
    w13   : in std_logic_vector((vwidth-1) downto 0);
    w21   : in std_logic_vector((vwidth-1) downto 0);
    w22   : in std_logic_vector((vwidth-1) downto 0);
    w23   : in std_logic_vector((vwidth-1) downto 0);
    w31   : in std_logic_vector((vwidth-1) downto 0);
    w32   : in std_logic_vector((vwidth-1) downto 0);
    w33   : in std_logic_vector((vwidth-1) downto 0);
    DVw    : in std_logic;
    DVs  : out std_logic;
    s1   : out std_logic_vector(vwidth-1 downto 0);     -- les 9 valeurs de
sortie du circuit après un tri croissant;
    s2  : out std_logic_vector(vwidth-1 downto 0);
    s3  : out std_logic_vector(vwidth-1 downto 0);
    s4  : out std_logic_vector(vwidth-1 downto 0);
    s5  : out std_logic_vector(vwidth-1 downto 0);
    s6  : out std_logic_vector(vwidth-1 downto 0);
    s7  : out std_logic_vector(vwidth-1 downto 0);
    s8  : out std_logic_vector(vwidth-1 downto 0);
    s9  : out std_logic_vector(vwidth-1 downto 0)
  );
end sort_3x3;
architecture sort_3x3 of sort_3x3 is
  -- les signaux utilisés pour la comparaison;
  signal c11_L: std_logic_vector((vwidth-1) downto 0);
  signal c11_H: std_logic_vector((vwidth-1) downto 0);
  signal c12_L: std_logic_vector((vwidth-1) downto 0);
  signal c12_H: std_logic_vector((vwidth-1) downto 0);
  signal c13_L: std_logic_vector((vwidth-1) downto 0);
  signal c13_H: std_logic_vector((vwidth-1) downto 0);
  signal c14_L: std_logic_vector((vwidth-1) downto 0);
  signal c14_H: std_logic_vector((vwidth-1) downto 0);
signal c21_L: std_logic_vector((vwidth-1) downto 0);
signal c21_H: std_logic_vector((vwidth-1) downto 0);
signal c22_L: std_logic_vector((vwidth-1) downto 0);
signal c22_H: std_logic_vector((vwidth-1) downto 0);
signal c23_L: std_logic_vector((vwidth-1) downto 0);
```

```
signal c23_H: std_logic_vector((vwidth-1) downto 0);
signal c24_L: std_logic_vector((vwidth-1) downto 0);
signal c24_H: std_logic_vector((vwidth-1) downto 0);
signal c31_L: std_logic_vector((vwidth-1) downto 0);
signal c31_H: std_logic_vector((vwidth-1) downto 0);
signal c32_L: std_logic_vector((vwidth-1) downto 0);
signal c32_H: std_logic_vector((vwidth-1) downto 0);
signal c33_L: std_logic_vector((vwidth-1) downto 0);
signal c33_H: std_logic_vector((vwidth-1) downto 0);
signal c34_L: std_logic_vector((vwidth-1) downto 0);
signal c34_H: std_logic_vector((vwidth-1) downto 0);
signal c41_L: std_logic_vector((vwidth-1) downto 0);
signal c41_H: std_logic_vector((vwidth-1) downto 0);
signal c42_L: std_logic_vector((vwidth-1) downto 0);
signal c42_H: std_logic_vector((vwidth-1) downto 0);
signal c43_L: std_logic_vector((vwidth-1) downto 0);
signal c43_H: std_logic_vector((vwidth-1) downto 0);
signal c4a1_L: std_logic_vector((vwidth-1) downto 0);
  signal c4a1_H: std_logic_vector((vwidth-1) downto 0);
  signal c4a2_L: std_logic_vector((vwidth-1) downto 0);
  signal c4a2_H: std_logic_vector((vwidth-1) downto 0);
  signal c4b0_L: std_logic_vector((vwidth-1) downto 0);
  signal c4b0_H: std_logic_vector((vwidth-1) downto 0);
  signal c4b1_L: std_logic_vector((vwidth-1) downto 0);
  signal c4b1_H: std_logic_vector((vwidth-1) downto 0);
  signal c4b2_L: std_logic_vector((vwidth-1) downto 0);
  signal c4b2_H: std_logic_vector((vwidth-1) downto 0);
  signal c51_L: std_logic_vector((vwidth-1) downto 0);
  signal c51_H: std_logic_vector((vwidth-1) downto 0);
  signal c61_L: std_logic_vector((vwidth-1) downto 0);
  signal c61_H: std_logic_vector((vwidth-1) downto 0);
  signal c71_L: std_logic_vector((vwidth-1) downto 0);
  signal c71_H: std_logic_vector((vwidth-1) downto 0);
  signal c81_L: std_logic_vector((vwidth-1) downto 0);
  signal c81_H: std_logic_vector((vwidth-1) downto 0);
  signal c91_L: std_logic_vector((vwidth-1) downto 0);
  signal c91_H: std_logic_vector((vwidth-1) downto 0);
  signal c101_L: std_logic_vector((vwidth-1) downto 0);
  signal c101_H: std_logic_vector((vwidth-1) downto 0);
  signal c111_L: std_logic_vector((vwidth-1) downto 0);
  signal c111_H: std_logic_vector((vwidth-1) downto 0);

  -- les signaux d'enregistrement des sorties de chaque étape de tri;
signal r11: std_logic_vector((vwidth-1) downto 0);
signal r21: std_logic_vector((vwidth-1) downto 0);
signal r31: std_logic_vector((vwidth-1) downto 0);
signal r41: std_logic_vector((vwidth-1) downto 0);
```

```
signal r42: std_logic_vector((vwidth-1) downto 0);
signal r43: std_logic_vector((vwidth-1) downto 0);
signal r4a1: std_logic_vector((vwidth-1) downto 0);
signal r4a2: std_logic_vector((vwidth-1) downto 0);
signal r4a3: std_logic_vector((vwidth-1) downto 0);
signal r4a4: std_logic_vector((vwidth-1) downto 0);
signal r4a5: std_logic_vector((vwidth-1) downto 0);
signal r4b1: std_logic_vector((vwidth-1) downto 0);
signal r4b4: std_logic_vector((vwidth-1) downto 0);
signal r4b5: std_logic_vector((vwidth-1) downto 0);
signal r51: std_logic_vector((vwidth-1) downto 0);
signal r52: std_logic_vector((vwidth-1) downto 0);
signal r53: std_logic_vector((vwidth-1) downto 0);
signal r54: std_logic_vector((vwidth-1) downto 0);
signal r55: std_logic_vector((vwidth-1) downto 0);
signal r56: std_logic_vector((vwidth-1) downto 0);
signal r57: std_logic_vector((vwidth-1) downto 0);
signal r61: std_logic_vector((vwidth-1) downto 0);
signal r62: std_logic_vector((vwidth-1) downto 0);
signal r63: std_logic_vector((vwidth-1) downto 0);
signal r64: std_logic_vector((vwidth-1) downto 0);
signal r65: std_logic_vector((vwidth-1) downto 0);
signal r66: std_logic_vector((vwidth-1) downto 0);
signal r67: std_logic_vector((vwidth-1) downto 0);
signal r71: std_logic_vector((vwidth-1) downto 0);
signal r72: std_logic_vector((vwidth-1) downto 0);
signal r73: std_logic_vector((vwidth-1) downto 0);
signal r74: std_logic_vector((vwidth-1) downto 0);
signal r75: std_logic_vector((vwidth-1) downto 0);
signal r76: std_logic_vector((vwidth-1) downto 0);
signal r77: std_logic_vector((vwidth-1) downto 0);
signal r81: std_logic_vector((vwidth-1) downto 0);
signal r82: std_logic_vector((vwidth-1) downto 0);
signal r83: std_logic_vector((vwidth-1) downto 0);
signal r84: std_logic_vector((vwidth-1) downto 0);
signal r85: std_logic_vector((vwidth-1) downto 0);
signal r86: std_logic_vector((vwidth-1) downto 0);
signal r87: std_logic_vector((vwidth-1) downto 0);
signal r91: std_logic_vector((vwidth-1) downto 0);
signal r92: std_logic_vector((vwidth-1) downto 0);
signal r93: std_logic_vector((vwidth-1) downto 0);
signal r94: std_logic_vector((vwidth-1) downto 0);
signal r95: std_logic_vector((vwidth-1) downto 0);
signal r96: std_logic_vector((vwidth-1) downto 0);
signal r97: std_logic_vector((vwidth-1) downto 0);
signal r101: std_logic_vector((vwidth-1) downto 0);
signal r102: std_logic_vector((vwidth-1) downto 0);
```

```
signal r103: std_logic_vector((vwidth-1) downto 0);
signal r104: std_logic_vector((vwidth-1) downto 0);
signal r105: std_logic_vector((vwidth-1) downto 0);
signal r106: std_logic_vector((vwidth-1) downto 0);
signal r107: std_logic_vector((vwidth-1) downto 0);
signal r111: std_logic_vector((vwidth-1) downto 0);
signal r112: std_logic_vector((vwidth-1) downto 0);
signal r113: std_logic_vector((vwidth-1) downto 0);
signal r114: std_logic_vector((vwidth-1) downto 0);
signal r115: std_logic_vector((vwidth-1) downto 0);
signal r116: std_logic_vector((vwidth-1) downto 0);
signal r117: std_logic_vector((vwidth-1) downto 0);

-- les signaux de mise à jour du signal de validation DV;
signal ddddddddddddddDV: std_logic:='0';
signal dddddddddddddDV: std_logic;
signal ddddddddddddDV: std_logic;
signal dddddddddddDV: std_logic;
signal ddddddddddDV: std_logic;
signal dddddddddDV: std_logic;
signal ddddddddDV: std_logic;
signal dddddddDV: std_logic;
signal ddddddDV: std_logic;
signal dddddDV: std_logic;
signal ddddDV: std_logic;
signal dddDV: std_logic;
signal ddDV: std_logic;
signal dDV: std_logic;
begin
  process(Clk,RSTn)
  begin
    if RSTn = '0' then  -- initialisation de tous les signaux intermédiaires
à zéro;
      c11_L <= (others=>'0');
      c11_H <= (others=>'0');
      c12_L <= (others=>'0');
      c12_H <= (others=>'0');
      c13_L <= (others=>'0');
      c13_H <= (others=>'0');
      c14_L <= (others=>'0');
      c14_H <= (others=>'0');
      c21_L <= (others=>'0');
      c21_H <= (others=>'0');
c22_L <= (others=>'0');
c22_H <= (others=>'0');
c23_L <= (others=>'0');
c23_H <= (others=>'0');
c24_L <= (others=>'0');
c24_H <= (others=>'0');
```

```
c31_L <= (others=>'0');
c31_H <= (others=>'0');
c32_L <= (others=>'0');
c32_H <= (others=>'0');
c33_L <= (others=>'0');
c33_H <= (others=>'0');
c34_L <= (others=>'0');
c34_H <= (others=>'0');
c41_L <= (others=>'0');
c41_H <= (others=>'0');
c42_L <= (others=>'0');
c42_H <= (others=>'0');
c43_L <= (others=>'0');
c43_H <= (others=>'0');
c4a1_L <= (others=>'0');
c4a1_H <= (others=>'0');
c4a2_L <= (others=>'0');
c4a2_H <= (others=>'0');
c4b0_L <= (others=>'0');
      c4b0_H <= (others=>'0');
      c4b1_L <= (others=>'0');
      c4b1_H <= (others=>'0');
      c4b2_L <= (others=>'0');
      c4b2_H <= (others=>'0');
      c51_L <= (others=>'0');
      c51_H <= (others=>'0');
      c61_L <= (others=>'0');
      c61_H <= (others=>'0');
      c71_L <= (others=>'0');
      c71_H <= (others=>'0');
      c81_L <= (others=>'0');
      c81_H <= (others=>'0');
      c91_L <= (others=>'0');
      c91_H <= (others=>'0');
      c101_L <= (others=>'0');
      c101_H <= (others=>'0');
      c111_L <= (others=>'0');
      c111_H <= (others=>'0');
      r11 <= (others=>'0');
      r21 <= (others=>'0');
      r31 <= (others=>'0');
      r41 <= (others=>'0');
      r42 <= (others=>'0');
      r43 <= (others=>'0');
      r4a1 <= (others=>'0');
r4a2 <= (others=>'0');
r4a3 <= (others=>'0');
```

```
r4a4 <= (others=>'0');
r4a5 <= (others=>'0');
r4b1 <= (others=>'0');
r4b4 <= (others=>'0');
r4b5 <= (others=>'0');
r51 <= (others=>'0');
r52 <= (others=>'0');
r53 <= (others=>'0');
r54 <= (others=>'0');
r55 <= (others=>'0');
r56 <= (others=>'0');
r57 <= (others=>'0');
r61 <= (others=>'0');
r62 <= (others=>'0');
r63 <= (others=>'0');
r64 <= (others=>'0');
r65 <= (others=>'0');
r66 <= (others=>'0');
r67 <= (others=>'0');
r71 <= (others=>'0');
r72 <= (others=>'0');
r73 <= (others=>'0');
r74 <= (others=>'0');
r75 <= (others=>'0');
r76 <= (others=>'0');
r77 <= (others=>'0');
r81 <= (others=>'0');
r82 <= (others=>'0');
r83 <= (others=>'0');
r84 <= (others=>'0');
r85 <= (others=>'0');
r86 <= (others=>'0');
r87 <= (others=>'0');
r91 <= (others=>'0');
r92 <= (others=>'0');
r93 <= (others=>'0');
r94 <= (others=>'0');
r95 <= (others=>'0');
r96 <= (others=>'0');
r97 <= (others=>'0');
r101 <= (others=>'0');
r102 <= (others=>'0');
r103 <= (others=>'0');
r104 <= (others=>'0');
r105 <= (others=>'0');
r106 <= (others=>'0');
r107 <= (others=>'0');
```

```
r111 <= (others=>'0');
      r112 <= (others=>'0');
      r113 <= (others=>'0');
      r114 <= (others=>'0');
      r115 <= (others=>'0');
      r116 <= (others=>'0');
      r117 <= (others=>'0');
      s1 <= (others=>'0');
      s2 <= (others=>'0');
      s3 <= (others=>'0');
      s4 <= (others=>'0');
      s5 <= (others=>'0');
      s6 <= (others=>'0');
      s7 <= (others=>'0');
      s8 <= (others=>'0');
      s9 <= (others=>'0');
      ddddddddddddDV <= '0';
      ddddddddddddDV <= '0';
      ddddddddddDV <= '0';
      dddddddddDV <= '0';
      ddddddddDV <= '0';
      dddddddDV <= '0';
      ddddddDV <= '0';
      dddddDV <= '0';
      ddddDV <= '0';
      dddDV <= '0';
      ddDV <= '0';
      dDV <= '0';
      DVs <= '0';
    elsif rising_edge(Clk) then
      if DVw = '1' then
        -- etape 1
        if w11 < w12 then
          c11_L <= w11;
          c11_H <= w12;
        else
          c11_L <= w12;
          c11_H <= w11;
        end if;
        if w13 < w21 then
          c12_L <= w13;
          c12_H <= w21;
        else
          c12_L <= w21;
          c12_H <= w13;
        end if;
        if w22 < w23 then
```

```
    c13_L <= w22;
    c13_H <= w23;
  else
    c13_L <= w23;
    c13_H <= w22;
  end if;
  if w31 < w32 then
    c14_L <= w31;
    c14_H <= w32;
  else
    c14_L <= w32;
    c14_H <= w31;
  end if;
  r11 <= w33;
  -- étape 2
  if c11_L < c12_L then
    c21_L <= c11_L;
    c21_H <= c12_L;
  else
    c21_L <= c12_L;
    c21_H <= c11_L;
  end if;
  if c11_H < c12_H then
    c22_L <= c11_H;
    c22_H <= c12_H;
  else
    c22_L <= c12_H;
    c22_H <= c11_H;
  end if;
  if c13_L < c14_L then
    c23_L <= c13_L;
    c23_H <= c14_L;
  else
    c23_L <= c14_L;
    c23_H <= c13_L;
  end if;
  if c13_H < c14_H then
    c24_L <= c13_H;
    c24_H <= c14_H;
  else
    c24_L <= c14_H;
    c24_H <= c13_H;
  end if;
  r21 <= r11;
 -- etape 3
  if c21_L < c23_L then
    c31_L <= c21_L;
    c31_H <= c23_L;
```

```
        else
  c31_L <= c23_L;
  c31_H <= c21_L;
end if;
if c21_H < c23_H then
  c32_L <= c21_H;
  c32_H <= c23_H;
else
  c32_L <= c23_H;
  c32_H <= c21_H;
end if;
if c22_L < c24_L then
  c33_L <= c22_L;
  c33_H <= c24_L;
else
  c33_L <= c24_L;
  c33_H <= c22_L;
  end if;
if c22_H < c24_H then
  c34_L <= c22_H;
  c34_H <= c24_H;
else
  c34_L <= c24_H;
  c34_H <= c22_H;
end if;
r31 <= r21;
-- etape 4
r41 <= c31_L;
if c31_H < c32_L then
  c41_L <= c31_H;
  c41_H <= c32_L;
else
  c41_L <= c32_L;
  c41_H <= c31_H;
end if;
if c32_H < c33_L then
  c42_L <=  c32_H;
  c42_H <= c33_L;
else
  c42_L <=  c33_L;
  c42_H <= c32_H;
end if;
if c33_H < c34_L then
  c43_L <= c33_H;
  c43_H <= c34_L;
  else
  c43_L <= c34_L;
```

```
  c43_H <= c33_H;
end if;
r42 <= c34_H;
r43 <= r31;
-- etape 4a
r4a1 <= r41;
if c41_L < c42_H then
  c4a1_L <= c41_L;
  c4a1_H <= c42_H;
else
  c4a1_L <= c42_H;
  c4a1_H <= c41_L;
end if;
if c41_H < c42_L then
  c4a2_L <= c41_H;
  c4a2_H <= c42_L;
else
  c4a2_L <= c42_L;
  c4a2_H <= c41_H;
end if;
r4a2 <= c43_L;
r4a3 <= c43_H;
r4a4 <= r42;
r4a5 <= r43;
-- etape 4b
r4b1 <= r4a1;
if c4a1_L < c4a2_L then
  c4b0_L <= c4a1_L;
  c4b0_H <= c4a2_L;
else
  c4b0_L <= c4a2_L;
  c4b0_H <= c4a1_L;
end if;
if c4a2_H < r4a2 then
  c4b1_L <= c4a2_H;
  c4b1_H <= r4a2;
else
  c4b1_L <= r4a2;
  c4b1_H <= c4a2_H;
end if;
if c4a1_H < r4a3 then
c4b2_L <= c4a1_H;
          c4b2_H <= r4a3;
        else
          c4b2_L <= r4a3;
          c4b2_H <= c4a1_H;
        end if;
        r4b4 <= r4a4;
```

```
        r4b5 <= r4a5;
       -- etape 5
        if r4b1 < r4b5 then
          c51_L <= r4b1;
          c51_H <= r4b5;
        else
          c51_L <= r4b5;
          c51_H <= r4b1;
        end if;
        r51 <= c4b0_L;
        r52 <= c4b0_H;
        r53 <= c4b1_L;
        r54 <= c4b1_H;
        r55 <= c4b2_L;
        r56 <= c4b2_H;
        r57 <= r4b4;
       -- etape 6
if r51 < c51_H then
  c61_L <= r51;
  c61_H <= c51_H;
else
  c61_L <= c51_H;
  c61_H <= r51;
end if;
r61 <= c51_L;    -- L
r62 <= r52;
r63 <= r53;
r64 <= r54;
r65 <= r55;
r66 <= r56;
r67 <= r57;
-- etape 7
if r62 < c61_H then
  c71_L <= r62;
  c71_H <= c61_H;
else
  c71_L <= c61_H;
  c71_H <= r62;
end if;
r71 <= r61;    -- L
r72 <= c61_L;    -- 2L
r73 <= r63;
r74 <= r64;
r75 <= r65;
r76 <= r66;
r77 <= r67;
       -- etape 8
        if r73 < c71_H then
```

```
          c81_L <= r73;
          c81_H <= c71_H;
        else
          c81_L <= c71_H;
          c81_H <= r73;
        end if;
        r81 <= r71;     -- L
        r82 <= r72;     -- 2L
        r83 <= c71_L;    -- 3L
        r84 <= r74;
        r85 <= r75;
        r86 <= r76;
        r87 <= r77;
       -- etape 9
        if r84 < c81_H then
          c91_L <= r84;
          c91_H <= c81_H;
        else
          c91_L <= c81_H;
          c91_H <= r84;
        end if;
        r91 <= r81;     -- L
        r92 <= r82;     -- 2L
        r93 <= r83;     -- 3L
        r94 <= c81_L;    -- 4L
        r95 <= r85;
        r96 <= r86;
        r97 <= r87;
       -- etape 10
        if r95 < c91_H then
          c101_L <= r95;
          c101_H <= c91_H;
        else
          c101_L <= c91_H;
          c101_H <= r95;
        end if;
        r101 <= r91;     -- L
        r102 <= r92;     -- 2L
        r103 <= r93;     -- 3L
        r104 <= r94;     -- 4L
        r105 <= c91_L;   -- M
        r106 <= r96;
        r107 <= r97;
       -- etape 11
if r106 < c101_H then
  c111_L <= r106;
  c111_H <= c101_H;
else
```

```
  c111_L <= c101_H;
  c111_H <= r106;
end if;
r111 <= r101;    -- L
r112 <= r102;    -- 2L
r113 <= r103;    -- 3L
r114 <= r104;    -- 4L
r115 <= r105;    -- M
r116 <= c101_L;  -- 4L
r117 <= r107;
-- etape 12
if r117 < c111_H then
  s8 <= r117;  -- 2H
  s9 <= c111_H;  -- H
else
  s8 <= c111_H;  -- 2H
  s9 <= r117;  -- H
end if;
s1 <= r111;    -- L
s2 <= r112;    -- 2L
s3 <= r113;    -- 3L
s4 <= r114;    -- 4L
s5 <= r115;    -- M
s6 <= r116;    -- 4H
s7 <= c111_L;    -- 3H

ddddddddddddDV <= ddddddddddddDV;
ddddddddddddDV <= ddddddddddddDV;
ddddddddddDV <= ddddddddddDV;
dddddddddDV <= dddddddddDV;
ddddddddDV <= ddddddddDV;
dddddddDV <= dddddddDV;
ddddddDV <= ddddddDV;
dddddDV <= dddddDV;
ddddDV <= ddddDV;
dddDV <= dddDV;
ddDV <= ddDV;
dDV <= dDV;
DVs <= dDV;
      end if;
      if DVw = '1' then
        ddddddddddddDV <= '1';
      end if;
    end if;
  end process;

end sort_3x3;
```

Fichier: `rc_counter.vhd`, le programme VHDL : compteur de position des lignes et des colonnes dans le fichier image.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity rc_counter is
  generic (
    num_cols: integer:=128;
    num_rows: integer:=128
    );
  port (
    Clk  : in std_logic;
    RSTn  : in std_logic;
    En  : in std_logic;
    ColPos  : out integer;
    RowPos  : out integer
  );
end rc_counter;
architecture rc_counter of rc_counter is
begin
  process(RSTn,Clk,En)
    variable ColPos_var: integer:=0;
    variable RowPos_var: integer:=0;
  begin
    if RSTn = '0' then
      ColPos_var := -1;
      ColPos <= 0;
      RowPos_var := 0;
      RowPos <= 0;
    elsif rising_edge(Clk) then
      if En = '1' then
        ColPos_var := ColPos_var +1; --la position j de pixel incrémentée
chaque cycle;
        if ColPos_var = num_cols then
          RowPos_var := RowPos_var +1; -- la position i de pixel incrémentée
après chaque 128 cycles (chaque ligne);
          ColPos_var := 0;
          if RowPos_var = num_rows then
            RowPos_var := 0;
          end if;
        end if;
        ColPos <= ColPos_var;
        RowPos <= RowPos_var;
      end if;
    end if;
```

```
  end process;

end rc_counter;
```

Fichier : `ro_filt_3x3.vhd`, le programme VHDL : filtre d'ordre par tri de données.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity ro_filt_3x3 is
  generic (
    vwidth: integer:=8;
    order: integer:=5; --l'ordre du filtre ;
    num_cols: integer:=128;
    num_rows: integer:=128
    );
  port (
    Clk  : in std_logic;
    RSTn  : in std_logic;
    D  : in std_logic_vector(vwidth-1 downto 0); -- Signal d'entrée;
    Dout   : out std_logic_vector(vwidth-1 downto 0); --signal de sortie
après traitement;
    DV  : out std_logic
  );
end ro_filt_3x3;
architecture ro_filt_3x3 of ro_filt_3x3 is
  component sort_3x3 -- le composant de tri ;
  generic (
    vwidth: integer:=8
    );
  port (
    Clk  : in std_logic;
    RSTn  : in std_logic;
    w11   : in std_logic_vector((vwidth-1) downto 0);
    w12   : in std_logic_vector((vwidth-1) downto 0);
    w13   : in std_logic_vector((vwidth-1) downto 0);
    w21   : in std_logic_vector((vwidth-1) downto 0);
    w22   : in std_logic_vector((vwidth-1) downto 0);
    w23   : in std_logic_vector((vwidth-1) downto 0);
    w31   : in std_logic_vector((vwidth-1) downto 0);
    w32   : in std_logic_vector((vwidth-1) downto 0);
    w33   : in std_logic_vector((vwidth-1) downto 0);
    DVw    : in std_logic;
    DVs  : out std_logic;
    s1  : out std_logic_vector(vwidth-1 downto 0);
    s2  : out std_logic_vector(vwidth-1 downto 0);
    s3  : out std_logic_vector(vwidth-1 downto 0);
```

```
    s4  : out std_logic_vector(vwidth-1 downto 0);
    s5  : out std_logic_vector(vwidth-1 downto 0);
    s6  : out std_logic_vector(vwidth-1 downto 0);
    s7  : out std_logic_vector(vwidth-1 downto 0);
    s8  : out std_logic_vector(vwidth-1 downto 0);
    s9  : out std_logic_vector(vwidth-1 downto 0)
);
  end component sort_3x3;
 -- les signaux intermediaires;
  signal w11: std_logic_vector((vwidth-1) downto 0);
  signal w12: std_logic_vector((vwidth-1) downto 0);
  signal w13: std_logic_vector((vwidth-1) downto 0);
  signal w21: std_logic_vector((vwidth-1) downto 0);
  signal w22: std_logic_vector((vwidth-1) downto 0);
  signal w23: std_logic_vector((vwidth-1) downto 0);
  signal w31: std_logic_vector((vwidth-1) downto 0);
  signal w32: std_logic_vector((vwidth-1) downto 0);
  signal w33: std_logic_vector((vwidth-1) downto 0);
  signal DVw: std_logic;
  signal DVs: std_logic;
  signal s1: std_logic_vector(vwidth-1 downto 0);
  signal s2: std_logic_vector(vwidth-1 downto 0);
  signal s3: std_logic_vector(vwidth-1 downto 0);
  signal s4: std_logic_vector(vwidth-1 downto 0);
  signal s5: std_logic_vector(vwidth-1 downto 0);
  signal s6: std_logic_vector(vwidth-1 downto 0);
  signal s7: std_logic_vector(vwidth-1 downto 0);
  signal s8: std_logic_vector(vwidth-1 downto 0);
  signal s9: std_logic_vector(vwidth-1 downto 0);

   component window_3x3 -- le composant de génération de la fenêtre;
  generic (
    vwidth: integer:=8
    );
  port (
    Clk  : in std_logic;
    RSTn  : in std_logic;
    D  : in std_logic_vector(vwidth-1 downto 0);
    w11  : out std_logic_vector(vwidth-1 downto 0);
    w12  : out std_logic_vector(vwidth-1 downto 0);
    w13  : out std_logic_vector(vwidth-1 downto 0);
    w21  : out std_logic_vector(vwidth-1 downto 0);
    w22  : out std_logic_vector(vwidth-1 downto 0);
    w23  : out std_logic_vector(vwidth-1 downto 0);
    w31  : out std_logic_vector(vwidth-1 downto 0);
    w32  : out std_logic_vector(vwidth-1 downto 0);
    w33  : out std_logic_vector(vwidth-1 downto 0);
```

```
    DV  : out std_logic:='0'
 );
end component window_3x3;
component rc_counter -- le composant de comptage de la position du pixel
traité;
generic (
  num_cols: integer:=128;
  num_rows: integer:=128
  );
port (
  Clk  : in std_logic;
  RSTn  : in std_logic;
  En  : in std_logic;
  ColPos  : out integer;
  RowPos  : out integer
);
end component rc_counter;
signal ColPos: integer:=0;
signal RowPos: integer:=0;
signal ColPos_c: integer:=0;
signal RowPos_c: integer:=0;
signal rt1: integer:=0;
signal rt2: integer:=0;
signal rt3: integer:=0;
signal rt4: integer:=0;
signal rt5: integer:=0;
signal rt6: integer:=0;
signal rt7: integer:=0;
signal rt8: integer:=0;
signal rt9: integer:=0;
signal rt10: integer:=0;
signal rt11: integer:=0;
signal rt12: integer:=0;
signal rt13: integer:=0;
signal rt14: integer:=0;
signal rt15: integer:=0;
signal rt16: integer:=0;
signal flag: std_logic:='0';
begin
  sort_3x3x: sort_3x3
    generic map (
      vwidth => 8
    )
    port map (
      Clk  => Clk,
      RSTn  => RSTn,
      w11   => w11,
```

```
      w12   => w12,
      w13   => w13,
      w21   => w21,
      w22   => w22,
      w23   => w23,
      w31   => w31,
      w32   => w32,
      w33   => w33,
      DVw    => DVw,
      DVs  => DVs,
      s1  => s1,
      s2  => s2,
      s3  => s3,
      s4  => s4,
      s5  => s5,
      s6  => s6,
      s7  => s7,
      s8  => s8,
      s9  => s9
    );
 window_3x3x: window_3x3
  generic map (
    vwidth => 8
  )
  port map (
    Clk  =>  Clk,
    RSTn  =>  RSTn,
    D  =>  D,
    w11  =>  w11,
    w12  =>  w12,
    w13  =>  w13,
    w21  =>  w21,
    w22  =>   w22,
    w23  =>  w23,
    w31  =>  w31,
    w32  =>  w32,
    w33  =>  w33,
DV  =>  DVw
  );
rc_counterx: rc_counter
  generic map (
  num_cols  => 128,
  num_rows  => 128
  )
  port map (
  Clk    => Clk,
  RSTn    => RSTn,
```

```
  En    => RSTn,
  ColPos    => ColPos,
  RowPos    => RowPos
);
ro_filt_proc: process(RSTn,Clk)
begin
  if RSTn = '0' then
    ColPos_c <= 0;
    rt1 <= 0;
rt2 <= 0;
rt3 <= 0;
rt4 <= 0;
rt5 <= 0;
rt6 <= 0;
rt7 <= 0;
rt8 <= 0;
rt9 <= 0;
rt10 <= 0;
rt11 <= 0;
rt12 <= 0;
rt13 <= 0;
rt14 <= 0;
rt15 <= 0;
rt16 <= 0;
RowPos_c <= 0;
Dout <= (others=>'0');
DV <= '0';
flag <= '0';
elsif rising_edge(Clk) then

  -- correction des compteurs;
  ColPos_c <= ((ColPos-16) mod 128);
  rt1 <= ((RowPos-1) mod 128);
  rt2 <= rt1;
  rt3 <= rt2;
  rt4 <= rt3;
  rt5 <= rt4;
  rt6 <= rt5;
  rt7 <= rt6;
  rt8 <= rt7;
  rt9 <= rt8;
  rt10 <= rt9;
  rt11 <= rt10;
  rt12 <= rt11;
  rt13 <= rt12;
  rt14 <= rt13;
  rt15 <= rt14;
```

```
  rt16 <= rt15;
  RowPos_c <= rt16;

if (ColPos_c = num_cols-1) or (RowPos_c = num_rows-1) or (ColPos_c
= num_cols-2) or (RowPos_c = 0) then
    Dout <= (others=>'0');
    else
        if order = 1 then
          Dout <= s1;
        elsif order = 2 then
          Dout <= s2;
        elsif order = 3 then
          Dout <= s3;
        elsif order = 4 then
          Dout <= s4;
        elsif order = 5 then
          Dout <= s5;
        elsif order = 6 then
          Dout <= s6;
        elsif order = 7 then
          Dout <= s7;
        elsif order = 8 then
          Dout <= s8;
        elsif order = 9 then
          Dout <= s9;
        end if;
      end if;
      if ColPos >= 16 and RowPos >= 1 then
        DV <= '1';
           flag <= '1';
      elsif flag = '1' then
              DV <= '1';
      else
        DV <= '0';
      end if;
    end if;
  end process;
end ro_filt_3x3;
```

Fichier : `filt_3x3_tb.vhd`, Le programme VHDL : testbench pour la simulation de circuit du filtrage.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use std.textio.all;
```

```
entity ro_filt_3x3_tb is
  generic(
    vwidth  : INTEGER := 8;
    order   : INTEGER := 9;
    num_cols : INTEGER := 128;
    num_rows : INTEGER := 128 );
end ro_filt_3x3_tb;
architecture TB_ARCHITECTURE of ro_filt_3x3_tb is
  component ro_filt_3x3
  generic(
    vwidth  : INTEGER := 8;
    order   : INTEGER := 9;
    num_cols : INTEGER := 128;
    num_rows : INTEGER := 128 );
  port(
    Clk   : in std_logic;
    RSTn   : in std_logic;
    D   : in std_logic_vector((vwidth-1) downto 0);
    Dout   : out std_logic_vector((vwidth-1) downto 0);
    DV  : out std_logic );
end component;
  signal Clk   : std_logic;
  signal RSTn   : std_logic;
  signal D   : std_logic_vector((vwidth-1) downto 0);
  signal Dout   : std_logic_vector((vwidth-1) downto 0);
  signal DV   : std_logic;
begin
  UUT : ro_filt_3x3 - process de filtrage
    port map
      (Clk   => Clk,
      RSTn   => RSTn,
      D   => D,
      Dout   => Dout,
      DV   => DV );

 read_from_file: process (Clk)  --process de lecture des données
    variable indata_line: line;
    variable indata: integer;
    file input_data_file: text open read_mode is "output_bin.bin";
  begin
    if rising_edge(Clk) then
      readline(input_data_file,indata_line);
      read(indata_line,indata);
      D <= conv_std_logic_vector(indata,8);
      if endfile(input_data_file) then
        report "end of file -- looping back to start of file";
        file_close(input_data_file);
```

```
        file_open(input_data_file,"output_bin.bin");
      end if;
    end if;
  end process;

  write_to_file: process (Clk) --process d'écriture des données après
traitement
    variable outdata_line: line;
    variable outdata: integer:=0;
    file output_data_file: text open write_mode is "vhdl_output.bin";
  begin
    if rising_edge(Clk) then
      outdata := CONV_INTEGER(unsigned(Dout));
      if DV = '1' then
       write(outdata_line,outdata);
       writeline(output_data_file,outdata_line);
      end if;
    end if;
  end process;

  clock_gen: process
  begin
    Clk <= '0';
    wait for 5 ns;
    Clk <= '1';
    wait for 5 ns;
  end process;

  reset_gen: process
  begin
     RSTn <= '0';
     wait for 10 ns;
   RSTn <= '1';
   wait;
  end process;
end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_ro_filt_3x3 of ro_filt_3x3_tb is
  for TB_ARCHITECTURE
    for UUT : ro_filt_3x3
     use entity work.ro_filt_3x3(ro_filt_3x3);
    end for;
  end for;
end TESTBENCH_FOR_ro_filt_3x3;
```

Références Bibliographiques

[1] : Circuits reconfigurables : Les FPGAs, par Eduardo Sanchez-EPFL.

[2]: Les circuits FPGA Concepts de base, architecture et applications, Cours SMC par François Verdier, Université de Cergy-Pontoise.

[3]: introduction aux circuits FPGA par Arnaud Tisserand (2003).

[4]: http://easytp.cnam.fr/alexandre/index_fichiers/support/ele015_cours_vol2.pdf

[5] : Opérations morphologiques, Rolf Ingold, université de FRIBOURG – Suisse (2006)(PDF).

[6] : Traitement des images astronomiques, article réalisé par T.I.P.E (Travaux Pratiques d'Initiative Personnelle Encadrés) rédiric Tran-Minh France.

[7]: circuits numériques et synthèse logiques, un outil VHDL, par Jacques WEBER et Maurice MEAUDRE- IUT de CACHAN-France (PDF).

[8] : Modélisation de circuits numériques en B en utilisant un outil graphique, Youssouf WABI, thèse doctorat-novembre 2002.

[9] : Cours de traitement d'images. Par : Vincent BARRA

[10] : cours traitement d'images « la convolution et filtre moyenneur », par Alain Boucher.

[11] : Cours de segmentation des images par C. Durosier et C. Boudry, URFIST deParis France (2005).

[12] : http://www.mon-club-elec.fr.

[13] : Introduction au microprocesseur et circuits assimilés 5éme partie (5/5) : les composants FPGA(2010).