République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE KASDI MERBAH OUARGLA
Faculté des Nouvelles Technologies de L'information et de la Communication

Département D'électronique et des Télécommunications

Mémoire de MASTER ACADEMIQUE
Domaine : Sciences et technologie
Filière : Électronique
Spécialité : Électronique des Systèmes Embarqués

Présenté par :
KHADRAOUI Imadeddine
MESMARI Yacine
HANED Mohammed

Thème

Implémentation d'un algorithme de traitement d'image sur un FPGA

Soutenu publiquement
le : / /2020

Devant le jury :

Dr. M.BOUZIDI	MCB	Président	UKM OUARGLA
Mr. N. MELHEGUEG	MAA	Promoteur	UKM OUARGLA
Mr. A.S.AOUF	MAA	Examinateur	UKM OUARGLA

Année Universitaire : 2019 /2020

Résumé

Les systèmes de traitement d'images embarqués basés sur FPGA offrent des ressources de calcul considérables mais présentent des défis de programmation par rapport aux systèmes logiciels. Le but de ce travail est la mise en œuvre d'un algorithme de traitement d'image utilisant une carte FPGA. Dans ce projet, certaines opérations de traitement simples sont implémentées dans Verilog telles que l'inversion, le contrôle de la luminosité et les opérations de seuil. La simulation à l'aide de Verilog en remplaçant le capteur d'image / caméra par Le code de lecture d'image Verilog prend son entrée à partir d'un dossier dans le pc.

Mots clés: systèmes embarques, réseau de portes programmables, traitement d'images, Les circuits logique programmable.

Abstract

FPGA-based embedded image processing systems offer considerable computing resources but present programming challenges when compared to software systems the goal of this work is the implementation of an image processing algorithm using an FPGA card. In this project, some simple processing operations are implemented in Verilog such as inversion, brightness control and threshold operations. A simulation using Verilog by replacing the image sensor/camera by the image reading Verilog code takes its input from a folder in the pc

Keywords: Embedded systems, Field Programmable Gate Array, images processing, programmable logic circuits.

ملخص

توفر أنظمة معالجة الصور المضمنة القائمة على FPGA موارد حاسوبية كبيرة ولكنها تواجه تحديات برمجية عند مقارنتها بأنظمة البرامج . الهدف من هذا العمل هو تنفيذ خوارزمية معالجة الصور باستخدام بطاقة FPGA في هذا المشروع. و يتم تنفيذ بعض عمليات المعالجة البسيطة عن طريق Verilog مثل الانعكاس والتحكم في السطوع وعمليات العتبة. كما قمنا بالمحاكاة باستخدام Verilog عن طريق استبدال مستشعر الصورة / الكاميرا بكود Verilog الخاص بقراءة الصورة حيث يأخذ هذا الأخير مدخلا ته من مجلد في الكمبيوتر

الكلمات المفتاحية : الأنظمة المضمنة ، مصفوفة البوابة القابلة للبرمجة ،معالجة الصور ،الدوائر المنطقية القابلة للبرمجة .

Remerciements

Tout d'abord, Nous Remercions le Dieu notre créateur de nos avoir donné la force d'achever ce travail.

Nous adressons nos remerciements à Monsieur Nacer MELHEGUEG pour ses conseils et ces orientations.

Nous tenons à remercier les membres de jury.

Nous remercions, nos enseignants du département d'électronique et communication (Université Kasdi Marbah –Ouargla), pour nous avoir accompagnés le long de notre formation. Ainsi que toute la communauté d'électronique.

Enfin nos plus chaleureux remerciements pour toute personne ayant participé de prés ou de loin à la réalisation de ce Modest travail.

Je dédie ce travail

A mes chers Parents,

Pour tous vos sacrifices pour moi, nul mot ne saura exprimer mon amour envers vous. Que Dieu vous protège et vous accorde une longue vie, car je ne pourrais jamais oublier la tendresse et l'amour dévoué par lesquels ils m'ont toujours entouré depuis mon enfance.

A mes frères et mes sœurs

Que Dieu vous garde, Je vous aime et je vous souhaite une vie pleine de succès et de réussite.

*A Mes collègues **YAINE** et **MOHAMMED**.*

À tous mes chères amies

IMAD

Je dédie ce travail

*A ceux qui sont dans mon cœur, qui ont veillés pour notre confort et sacrifié beaucoup pour notre réussite, **A mes chers Parents** que dieu les gardent.*

A mes frères et mes sœurs

Pour tous mes oncles et tantes

Mes dédicaces vont tendrement à mes chères amies

*A Mes collègues **YAINE** et **IMAD**.*

MOHAMMED

Aux personnes qui me sont les plus chères dans ce monde,

***ma mère et mon père**, pour leurs*

sacrifices, leur encouragement et leur amour. Vous

avez fait de moi ce que je suis en ce

moment, un grand merci

A mes frères et sœurs

A toute ma famille

*A Mes collègues **MOHAMMED** et **IMAD**.*

A tous mes amis

YAINE

Liste des abréviations et des symboles :

FPGA: Field Programmable Gates Arrays.

VHDL: Very high speed integrated circuit Hardware Description Language.

HDL : Hardware Description Languages.

PLD: Programmable Logique Device.

I (i, j): Matrice représentant l'image

VLSI: Very Large Scale Integrated

ASIC: Application Specific Integrated Circuits.

CPLD : Complex Programmable Logique Device.

DSP: Digital Signal Processing.

CPU: Central Processing Unit.

CLBs: Configurable Logic Bloc.

IOB: Input Output Bloc.

RAM: Random-Access Memory.

LUT: Look-Up Table.

IOB: Input Output Bloc.

TTL: Transistor–Transistor Logic.

MOS: *Metal Oxide Semiconductor*

CMOS: Complementary Metal–Oxide–Semiconductor.

RTL: Register Transfer Level.

SRAM: Static Random Access Memory.

ROM: Read Only Memory.

PROM: *Programmable Read Only Memory*

EPROM: Erasable Programmable Read Only Memory.

EEPROM: Electrically Erasable Programmable Read Only Memory.

LCA: Logic Cell Array

ISE: Integrated Software Environment

DCM: Digital Clock Manager

PSM: Programmable Switch Manger

LUT: Look Up Table

LCA (Logic Cell Array)

RGB: Red (Rouge), Green (Vert), Blue (Bleu)

DCM: Digital Clock Manager

LED: Light-Emitting Diode

JTAG : joint Test Action Group

Bitmap: Une image matricielle, image numérique décrite point par point.

PCI: *Peripheral Component Interconnect*

PLL: phase lock loop

PAL: Programmable Array Logic

LC : Logic Cell

DLL: Delay-locked loop

Chapitre II : Kit de développement-carte FPGA

Chapitre III : Présentation et Simulation

Liste des Figures

Chapitre I

Chapitre II

Chapitre III

Liste des Tableaux

Introduction Générale

Introduction générale :

Le traitement d'image numérique est utilisé dans des domaines très vastes et en expansion couvrant des applications dans les services multimédias, les arts, la médecine, l'exploration spatiale, la surveillance, l'authentification, l'inspection automatisée de l'industrie et bien d'autres domaines.

Les domaines d'application du traitement numérique de l'image sont assez hétérogènes. Capturer l'étendue de ce champ afin de développer une compréhension de base des applications de traitement d'image signifie classer les images en fonction de leur source, comme le spectre d'énergie électromagnétique (par exemple, les rayons X), les images acoustiques, ultrasoniques et électroniques ou synthétiques générées par ordinateur. Ces types d'applications impliquent différents processus, y compris l'amélioration de la qualité d'image et la détection d'objets. Il est important de noter que ce type de traitement demande des ressources appropriées, comme la disponibilité de la mémoire ou des périphériques spécifiques connectés en dehors des besoins en puissance de traitement. Tous ces éléments ne sont pas disponibles sur tous les ordinateurs à usage général ordinaires et, souvent, les procédures associées ne sont pas très efficaces en termes de temps en raison d'autres contraintes supplémentaires. En utilisant du matériel spécifique à une application comme ASIC ou FPGA, une efficacité beaucoup plus grande peut être obtenue par rapport à une implémentation logicielle. La technologie VLSI (Very Large Scale Integrated) offre aujourd'hui des solutions alternatives complexes d'implémentation matérielle.

L'utilisation de matériel configurable et de langages de programmation au niveau du système permet la mise en œuvre directe d'algorithmes de traitement d'image avec des performances améliorées et avec un court délai de mise sur le marché. Il existe de nombreuses technologies disponibles pour la mise en œuvre matérielle. Les circuits intégrés spécifiques à une application (ASIC) et les réseaux de portes programmables (FPGA) sont tous deux des dispositifs reconfigurables capables de prendre en charge des techniques telles que le parallélisme et le pipelining.

L'utilisation de matériel reconfigurable pour mettre en œuvre des algorithmes de traitement d'image minimise le coût du temps de mise sur le marché, tandis qu'un prototypage rapide avec des étapes de débogage et de vérification simplifiées est également

possible. Par conséquent, les dispositifs reconfigurables semblent être le choix idéal pour la mise en œuvre d'algorithmes de traitement d'image.

Les FPGA ont traditionnellement été configurées par des concepteurs de matériel en utilisant des langages de conception de matériel (HDL) spécifiques. Il existe déjà peu de langages disponibles offrant différents niveaux d'abstraction, mais les plus importants sont Verilog HDL (Verilog) et Very High Speed Integrated Circuits (VHSIC) HDL (VHDL).

Dans ce mémoire nous nous intéresserons à l'étude et la simulation d'un Algorithme de traitement d'image sur l'environnement de développement ISE utilisant langage de description Verilog**.**

Pour cela le plan suivant a été adopté :

Le premier chapitre est consacré à des généralités sur le traitement d'image.

Le second chapitre à la présentation de la carte et les outils de développement des circuits FPGA.

Le troisième chapitre quant à lui, il sera entièrement consacré à la présentation et simulation des quelques opérations simples de traitement d'image avec Verilog.

Enfin, nous terminerons notre travail par une conclusion générale et quelques perspectives futures.

Chapitre I : Généralités sur le traitement d'image

I.1 Introduction:

Le traitement d'image peut être défini comme l'ensemble des méthodes et techniques opérant sur l'image afin d'extraire les informations les plus pertinentes ou tout simplement pour fournir une image plus perceptible à l'œil humain.

Dans ce chapitre nous présentons quelques notions de base du domaine de traitement d'image numérique tels que : la définition d'image, les types d'image, caractéristiques d'image, système de traitement d'image, analyse élémentaire, filtrage, La convolution, segmentation et en fin quelques exemples concrets de traitement d'images.

I.2 Définition d'une image:

La définition du terme « image » lui-même, telle qu'elle est donnée par le Petit Robert, englobe une multitude de significations distinctes. Cela va de la « reproduction exacte ou représentation analogique d'un être, d'une chose », à la « représentation mentale d'origine sensible » ou à des concepts plus physiques comme un « ensemble des points » où vont converger des rayons lumineux (cas des images optiques). [1]

I.3 Image numérique :

L'image numérique est l'image dont la surface est divisée en éléments de tailles fixes appelés cellules ou pixels, ayant chacun comme caractéristique un niveau de gris ou de couleurs prélevé à l'emplacement correspondant dans l'image réelle. [1]

I.4 Caractéristiques d'une image numérique:

L'image est un ensemble structuré d'informations caractérisé par les paramètres suivants:

I.4.1 Pixel :

Le pixel (extrait des mots anglais "**Picture element**") est le plus petit élément de l'image. Il possède une valeur **I (i, j)** qui représente son niveau de gris (**I** étant la matrice représentant l'image). Les coordonnées **i, j** donnent sa position dans l'image **I.** [2]

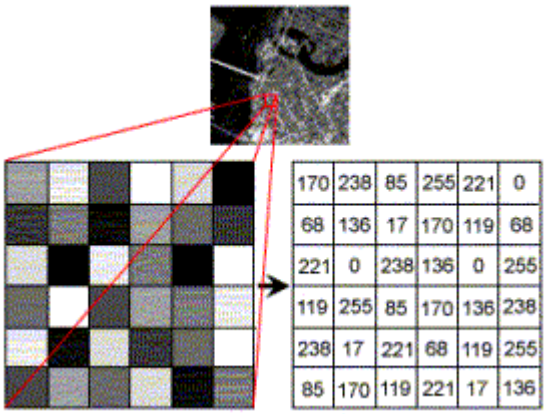


Figure.1.1: Pixels d'image

I.4.2 Résolution :

C'est la clarté ou la finesse de détails atteinte par un moniteur ou une imprimante dans la production d'images. Sur les moniteurs d'ordinateurs, la résolution est exprimée en nombre de pixels par unité de mesure (pouce ou centimètre). On utilise aussi le mot résolution pour désigner le nombre total de pixels affichables horizontalement ou verticalement sur un moniteur; plus grand est ce nombre, meilleure est la résolution. [3]

I.4.3 Dimension :

C'est la taille de l'image. Cette dernière se présente sous forme de matrice dont les éléments sont des valeurs numériques représentatives des intensités lumineuses (pixels). Le nombre de lignes de cette matrice multiplié par le nombre de colonnes nous donne le nombre total de pixels dans une image. [3]

I.4.4 Texture:

Une définition formelle de la texture est quasiment impossible. Mais d'une manière générale la texture se traduit par un arrangement spatial des pixels que l'intensité ou les couleurs seules ne suffisent pas à décrier. [1]

I.4.5 Bruit :

Un bruit (parasite) dans une image est considéré comme un phénomène de brusque variation de l'intensité d'un pixel par rapport à ses voisins, il provient de l'éclairage des dispositifs optiques et électroniques du capteur. [2]

Figure.1.2: Bruit dans l'image

I.4.6 Luminance :

La luminance est le degré de luminosité de chaque point de l'image. Elle est définit comme étant le quotient de l'intensité lumineuse d'une surface. [1]

I.4.7 Contours et textures :

Les contours représentent la frontière entre les objets de l'image, ou la limite entre deux pixels dont les niveaux de gris représentent une différence significative. Les textures décrivent la structure de ceux-ci. L'extraction de contour consiste à identifier dans l'image les points qui séparent deux textures différents. [2]

Figure.1.3 : Contour d'une image

I.4.8 Histogramme :

L'histogramme des niveaux de gris ou des couleurs d'une image est une fonction qui donne la fréquence d'apparition de chaque niveau de gris (couleur) dans l'image. Pour diminuer l'erreur de quantification, pour comparer deux images obtenues sous des éclairages différents, ou encore pour mesurer certaines propriétés sur une image.

Il permet de donner un grand nombre d'information sur la distribution des niveaux de gris (couleur) et de voir entre quelles bornes est repartie la majorité des niveaux de gris (couleur) dans les cas d'une image trop claire ou d'une image trop foncée.

La figure (1-6) montre une image avec son histogramme. [2]

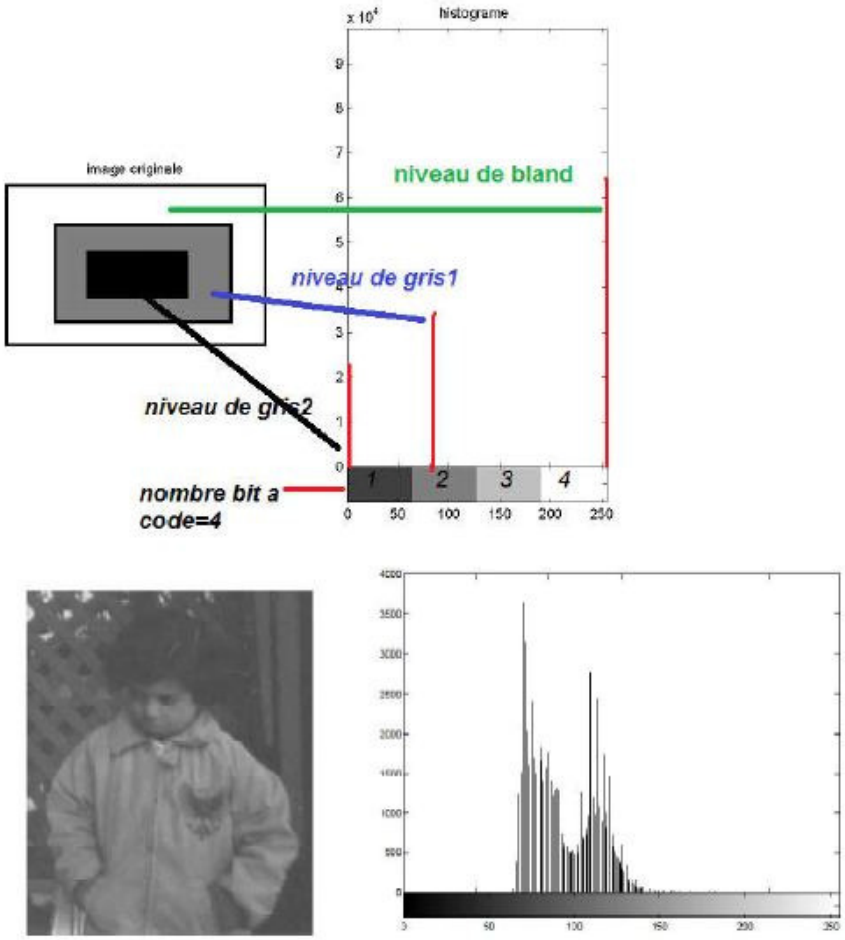


Figure.1.4: Example d'un histogramme.

I.4.9 Contraste :

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

Si L1 et L2 sont les degrés de luminosité respectivement de deux zones voisines A1 et A2 d'une image, le contraste C est défini par le rapport. [2]

$$C = \frac{L1 - L2}{L1 + L2} \qquad \text{(I.1)}$$

I.5 Types d'images :

Nous distinguons quatre types d'images :

I.5.1 Images binaires (en noir et blanc) :

Chaque pixel est soit blanc soit noir. Puisqu'il y a uniquement deux valeurs pour chaque pixel, un seul bit est utilisé pour le coder. [4]

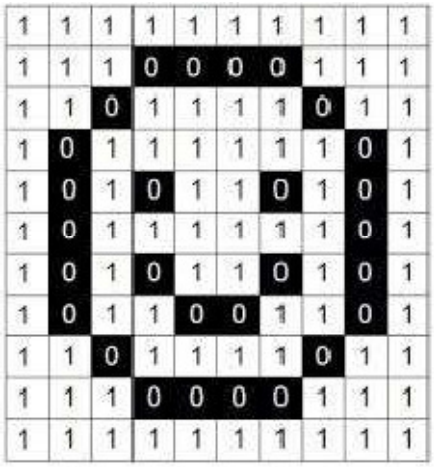


Figure.1.5 : Image binaire

I.5.2 Images à niveaux de gris (Monochromes):

Chaque pixel est un niveau de gris, allant de 0 (noir) à 255 (blanc). Cet intervalle de valeurs signifie que chaque pixel est codé sur huit bits (un octet). 256 niveaux de gris sont généralement suffisants pour la reconnaissance de la plus part des objets d'une scène. [4]

Figure.1.6 : Image niveaux de gris

I.5.3 Images en couleurs (Polychromes) :

Chaque pixel possède une couleur décrite par la quantité de rouge (R), vert (G) et bleu (B). Chacune de ces trois composantes est codée sur l'intervalle [0, 255],

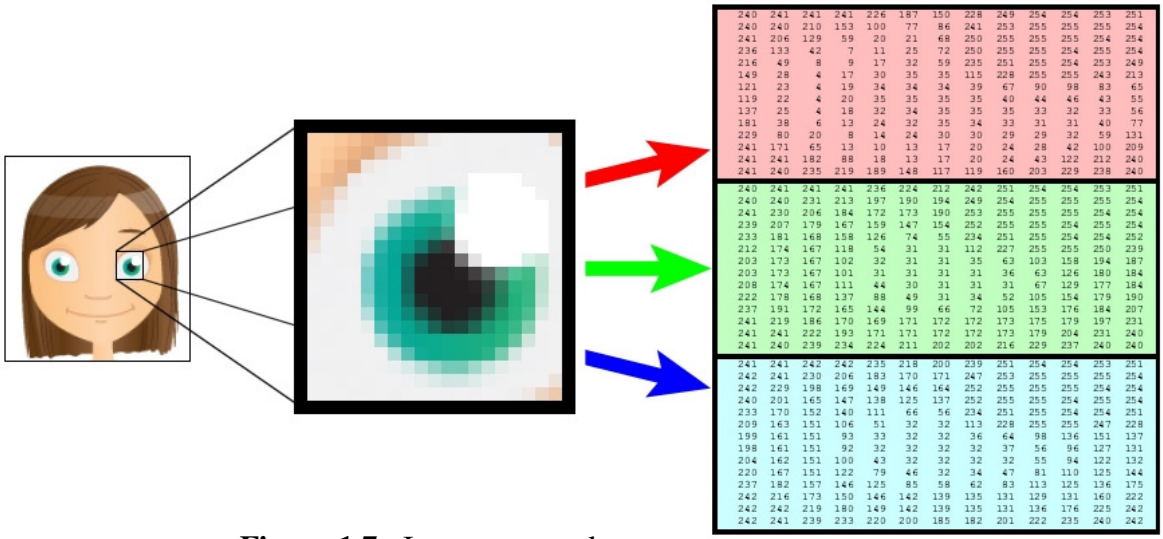


Figure.1.7 : Images en couleurs.

Ce qui donne 2553= 16 777 216 couleurs possibles. Il faut donc 24 bits pour coder un pixel. [4]

I.6 Qualité de l'image numérique :

Elle dépend, d'une part, de la qualité des images d'origine et, d'autre part, des moyens mis en œuvre pour convertir un signal analogique en signal numérique. Elle dépend aussi de :

- La qualité des périphériques de numérisation de l'image, du nombre de niveaux de gris ou de couleurs enregistrées, etc.
- La qualité de l'affichage à l'écran : définition de l'écran, nombre de teintes disponibles.

Les critères d'appréciation de la qualité d'une image, tels que cités succinctement ci-dessus, dépendent largement de la structure même de l'image réaliste ou conceptuelle et de son mode de représentation (bitmap ou vectorielle). [2]

I.7 Images bitmap et images vectorielles :

Lorsque l'on affiche une image sur l'écran d'un ordinateur, ce que l'on voit n'est qu'une succession de points. Il existe pourtant deux manières différentes de stocker une image sur une machine, on parle d'image bitmap ou bien d'image vectorielle. Dans une image bitmap, les points (ou pixels) sont stockés dans un tableau. Dans chaque case du tableau se trouve la couleur que l'on doit afficher au point correspondant. Pourtant cette méthode a ses limites, si la résolution de l'image est plus grande ou plus petite que celle de l'écran, il faut agrandir ou rétrécir l'image. Si la réduction de l'image ne pose pas de problème, l'agrandissement de l'image conduit à une dégradation de l'aspect visuel de l'image. Dans une image vectorielle, les objets sont constitués de formes géométriques simples, telles que des vecteurs, rectangles, des cercles, des ellipses ... Le principal avantage des images vectorielles par rapport aux images bitmap est qu'elles sont redimensionnables sans pertes de qualité. Nous allons voir pourquoi dans la partie suivante. [5]

Figure.1.8 : Images bitmap et images vectorielles

I.8 Système de traitement d'image:

Dans le contexte de la vision artificielle, le traitement d'images se place après les étapes d'acquisition et de numérisation, assurant les transformations d'images et la partie de calcul permettant d'aller vers une interprétation des images traitées. Cette phase d'interprétation est d'ailleurs, de plus en plus intégrée dans le traitement d'images, en faisant appel notamment à l'intelligence artificielle pour manipuler des connaissances, principalement sur les informations dont on dispose à propos de ce que représentent les images traitées (*connaissance du domaine*).

Un système de traitement d'image est généralement composé des unités suivantes :

- Un système d'acquisition et de numérisation qui permet d'effectuer l'échantillonnage et la quantification d'une image.
- Une mémoire de masse pour stocker les images numérisées.
- Un système de visualisation.
- Une unité centrale permettant d'effectuer les différentes opérations de traitement d'images. [6]

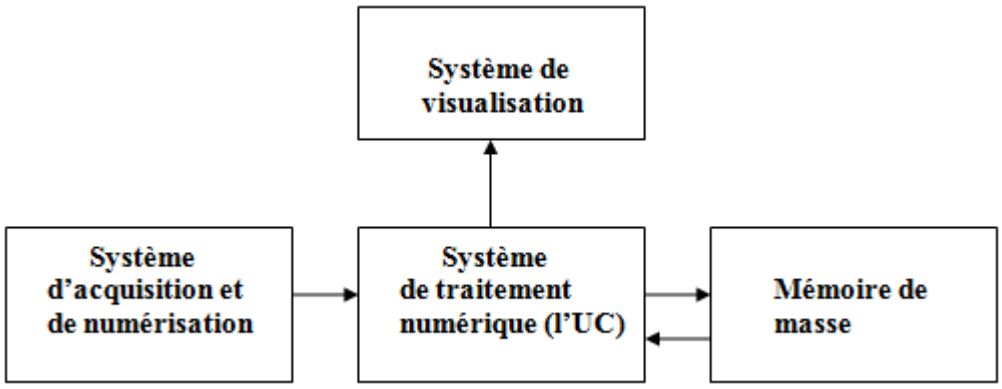


Figure I. 9 : Composition d'un système de traitement numérique

I.9 Traitement numérique des images :

Les techniques de traitement sont destinées à l'exploitation des informations contenues dans les images, ceci dans le but d'améliorer la qualité des images et de les rendre plus facilement interprétables, en d'autre termes elles permettent d'augmenter la qualité visuelle de L'image. [6]

I.9.1 Filtrage Numérique:

Les images numériques telles qu'elles sont acquises, sont très souvent inexploitables pour le traitement d'images, elles contiennent des signaux bruités. Pour remédier à cela, différents prétraitements pour l'amélioration ou la correction sont effectués

On peut scinder les filtres en deux grandes catégories :

I.9.1.1 Filtres linéaires:

Les filtres linéaires transforment un ensemble de données d'entrée en un ensemble de données de sortie par une convolution bidimensionnelle qui est une opération mathématique, ils permettent de supprimer le bruit dans l'image. Chaque filtre a une taille N x N avec N impair.

I.9.1.2 Filtres non linéaire :

Ils sont conçus pour régler les problèmes des filtres linéaires, Leur principe est le même que celui des filtres linéaires, il s'agit toujours de remplacer la valeur de chaque pixel par la valeur d'une fonction calculée dans son voisinage. La différence majeure, est que cette fonction n'est plus linéaire mais une fonction quelconque (elle peut inclure des opérateurs de comparaisons ou de classification). [6]

I.10 Conclusion :

Ce chapitre, nous l'avons voulu à ce qu'il soit une brève introduction aux concepts liés au domaine du traitement d'images. Les différentes définitions qui y sont développées sont celles des connaissances élémentaires de cette discipline, mais combien même elles sont essentielles pour l'initiation aux traitements approfondis des images.

Chapitre II : Kit de développement-carte FPGA

II.1- Introduction:

Les progrès technologiques dans le domaine des circuits intégrés ont permis la réduction des coûts et de la consommation. Les circuits intégrés spécifiques ont permis une réduction de la taille des systèmes numériques ainsi que la réalisation de circuits de plus en plus complexes, tout en améliorant leurs performances et leur fiabilité. Aujourd'hui les techniques de traitement numérique occupent une place majeure dans tous les systèmes électroniques modernes grand public, professionnels ou militaires. De plus, les techniques de réalisation de circuits spécifiques, tant dans les aspects matériels (composants reprogrammables, circuits pré-caractérisés et bibliothèques de macro-fonctions) que dans les aspects logiciels (placement-routage, synthèse logique) font désormais de la microélectronique une des bases indispensables pour la réalisation de systèmes numériques performants. Elle impose néanmoins une méthodologie de développement très structurée**.**

Les circuits FPGA (Field Programmable Gate Array) sont certainement les circuits reprogrammables ayant le plus de succès. Ce sont des circuits entièrement configurables par programmation qui permettent d'implanter physiquement, n'importe quelle fonction logique. De plus, ils ne sont pas limités à un mode de traitement séquentiel de l'information comme avec les microprocesseurs ; et en cas d'erreur, ils sont reprogrammables électriquement sans avoir à extraire le composant de son environnement. [8]

II.2- Circuits programmables:

Les circuits logiques programmables représentent une solution incontournable dans le domaine de l'électronique numérique. En effet, la possibilité de programmer un composant pour qu'il puisse fonctionner selon les besoins du concepteur est une aide précieuse pour pouvoir élaborer efficacement un circuit complexe avec des délais court et un coût de reviens faible. Contrairement aux circuits logiques prédéfini par les fabricants tels que les portes logiques (la famille 4000…) ou les fonctions logiques (les contrôleurs de bus …) qui ont des entrées sorties fixe une architecture figée un encombrement étendu (quelques portes logiques dans un boîtier volumineux) une consommation en énergie élevée réduit l'usage de ces composants pour de petites applications simple et des circuits à densités moyenne. Les PLDs (Programmable Logic Device en Anglais) offrent la possibilité de programmer l'architecture interne du composant, pour réaliser une fonction souhaitée par l'utilisateur et de configurer les pins 'entrés/sorties de celui-ci. En distinction par rapport aux microprocesseurs ou micro- contrôleurs qui sont, eux aussi programmables mais pour qui l'on peut seulement programmer le fonctionnement selon un programme existant dans

une mémoire, l'architecture interne est proposée par le fabricant tout comme les entrés/sorties. Les PLDs ont connu une évolution technologique au fil du temps depuis la parution du premier PAL (Programmable Array Logique ou réseau logique programmable) jusqu'à l'aboutissement aux FPGAs (Field Programmable Gate Array ou réseau de cellules Logiques programmables) qui sont les circuits logiques programmables les plus performant qui existent en ce moment, une évolution dû à une concurrence industrielle et scientifique**.** [9]

II.3- FPGA:

Le circuit FPGA (Field Programmable Logic Device) ou LCA (Logic Cell Array) est un circuit prédiffusé programmable. Le concept du FPGA est basé sur l'utilisation d'une LUT (Look Up Table) comme élément combinatoire de la cellule de base. En première approximation, cette LUT peut être vue comme une mémoire (16 bits en général) qui permet de créer n'importe quelle fonction logique combinatoire de 4 variables d'entrées. Chez Xilinx, on appelle cela un générateur de fonction ou Function Generator. La figure suivante représente la cellule type de base d'un FPGA.

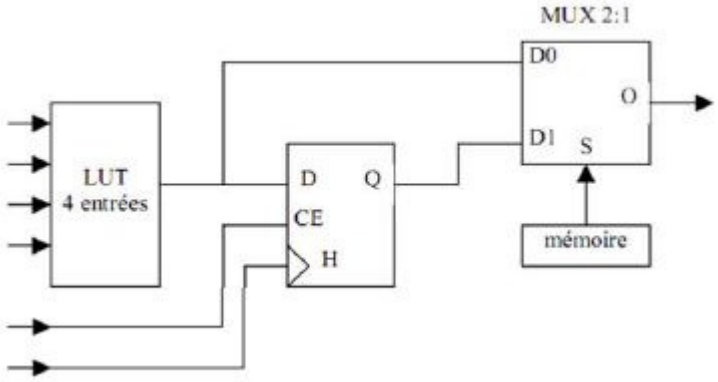


Figure. 2.1 : Cellule de base de circuit

Elle comprend une LUT 4 entrées et une bascule D (D Flip-Flop). La bascule D permet la réalisation de fonctions logiques séquentielles. La configuration du multiplexeur 2 vers 1 de sortie autorise la sélection des deux types de fonction, combinatoire ou séquentielle. Les cellules de base d'un FPGA sont disposées en lignes et en colonnes. Des lignes d'interconnexions programmables traversent le circuit, horizontalement et verticalement, entre les diverses cellules. Ces lignes d'interconnexions permettent de relier les cellules entre elles, et avec les plots d'entrées/sorties. Les connexions programmables sur ces lignes sont réalisées par des transistors MOS dont l'état est contrôlé par des cellules mémoires

SRAM. Ainsi, toute la configuration d'un FPGA est contenue dans des cellules SRAM. [10]

II.3.1- Structure et architecture:

Un circuit FPGA contient un très grand nombre de macro cellules avec une très grande souplesse d'interconnexion entre eux. Dans le FPGA, le temps de propagation dans les couches logiques du circuit dépend de l'organisation et de la distance entre les macro-cellules interconnectées. L'architecture, retenue par Xilinx, se présente sous forme de deux blocs :

- Un bloc appelé circuit configurable.
- Un bloc appelé réseau mémoire SRAM.

La couche dite "circuit configurable" est constituée d'une matrice de blocs logiques configurables CLB permettant de réaliser des fonctions combinatoires et des fonctions séquentielles. Tout autour de ces blocs logiques configurables, nous trouvons des blocs entrés/sorties (IOB) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs. 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. [11]

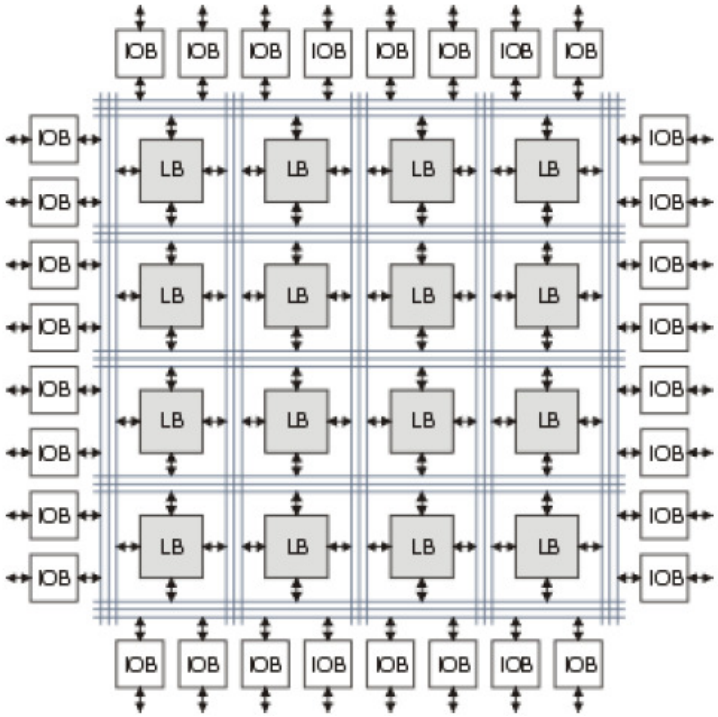


Figure.2.2: Architecture des fpga

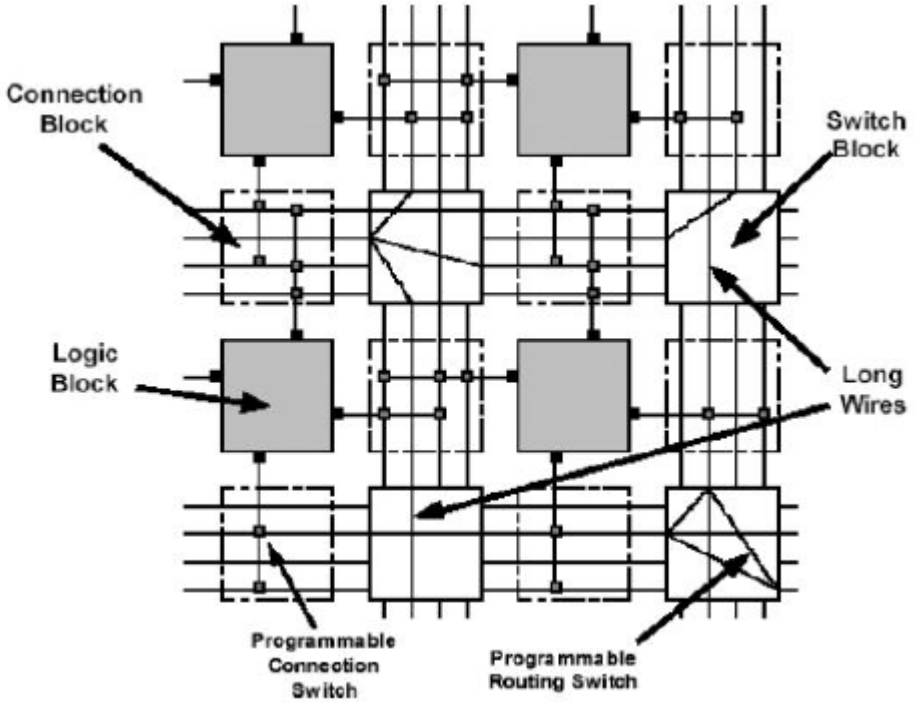


Figure.2.3: Vue interne d'un fpga

II.3.2- Programmation:

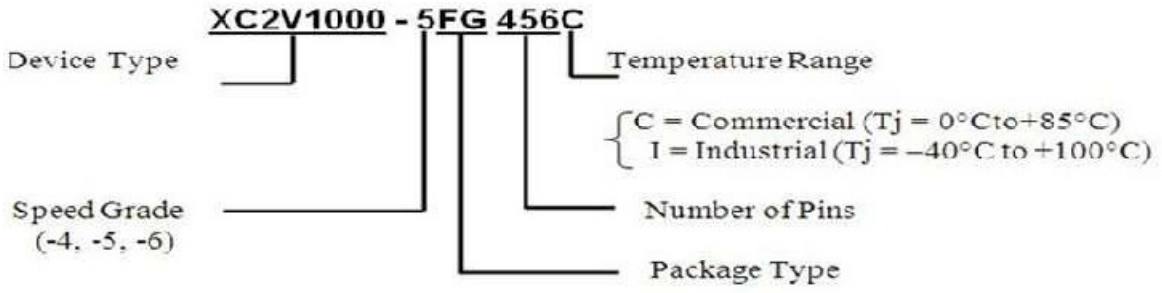
Les circuits FPGA sont programmables ou reprogrammables sur les cartes implantées par l'utilisateur. Cette reconfiguration est une propriété nécessaire face aux systèmes à charges et contraintes variables. Les dernières technologies qui sont utilisées ont permis la satisfaction des besoins forts en densité. Selon l'architecture des FPGA, chaque point d'interconnexion peut être réalisé selon deux technologies qui définissent deux classes différentes des FPGA à savoir **La technologie de programmation SRAM e**t la technologie de programmation **à base d'inti-fusible.** La technologie de programmation SRAM permet d'avoir une reconfiguration rapide des FPGA. Les points de connexions sont des ensembles des transistors commandés. Quant la technologie de programmation **à base d'inti-fusible**, les points d'interconnexions sont de type ROM. c'est-à-dire la modification du point est irréversible.

L'avantage des circuits programmables FPGA est de pouvoir être reprogrammable, Contrairement aux circuits intégrés de type ASIC. Ce qui rend cette solution modulable et donne la possibilité de modifier le programme générique de base afin de le rendre spécifique au circuit utilisé. Ces circuits programmables permettent aussi d'obtenir les

meilleures performances car la technologie FPGA utilise du parallélisme matériel, qui offre une puissance de calcul supérieure à celle des processeurs de signaux numériques (DSP). BDTI, une importante société d'analyse et de « *benchmarking* », a publié des études montrant que les FPGA peuvent offrir une puissance de traitement par dollar plusieurs fois supérieure à celle d'une solution DSP dans certaines applications. Contrôler les entrées et sorties (E/S) au niveau matériel permet d'obtenir des temps de réponse plus courts ainsi que des fonctionnalités spécifiques, qui répondent mieux aux besoins de l'application. [12]

II.3.3- Nomenclature des circuits FPGA :

La nomenclature des FPGA correspond à ce qui suit :



Cet exemple est la nomenclature de Virtex II de la compagnie Xilinx qui possède 1000 * 1000 portes (1 Million) et 456 pins, type de package FG et destiné aux applications commerciales**.** [11]

II.4- Eléments de différentes architectures :

Les circuits FPGA sont constitués de plusieurs blocs :

- Les éléments combinatoires et séquentiels CLB (Configurable Logic Bloc).
- Les éléments d'entrée / sortie IOB (Input Output Bloc), ils sont associés aux broches de circuit.
- Les éléments de mémorisation.
- Les éléments de contrôle et d'acheminement de l'horloge DCM (Digital Clock Manager)
- Les éléments de routage PSM (Programmable Switch Manger) qu'est une matrice d'interconnexions. [10]

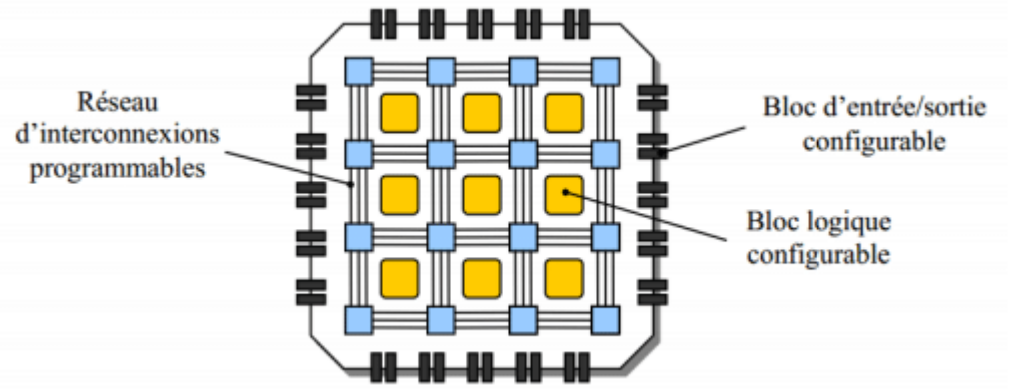


Figure 2.4 : Architecture générale d'un FPGA.

II.4.1- Eléments logiques :

Ce sont les éléments de base de tout FPGA. Grâce à leur configuration on peut réaliser dans ces blocs toutes les opérations de logique combinatoire. Ces blocs ont souvent la même constitution et cela malgré la différence de fabricants et d'architectures. Ils sont généralement constitués d'une ou plusieurs LUTs (Look Up Table) qui contiennent, après configuration, la table de vérité de la fonction logique qu'elles doivent réaliser ou alors un ensemble de valeurs qui sont mémorisées comme dans une ROM. La taille des LUTs est généralement de 4 entrées. Les LUTs sont généralement suivis d'un registre de sortie, ce qui permet de synchroniser, si nécessaire, la sortie sur une horloge. [13]

II.4.2- Eléments de mémorisation :

Pour des applications plus importantes, les FPGAs demandent souvent des capacités de stockage (comme en traitement d'images). La nécessité d'intégrer des blocs de mémoires directement dans l'architecture des FPGAs est vite devenue très important. De cette façon les temps d'accès à la mémoire sont diminués puisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit. [13]

II.4.3- Eléments de routages :

S'il y a des éléments plus importants que d'autres dans les FPGAs, ce sont les éléments de routages. En effet les ressources de routages représentent la plus grosse partie de silicium consommée sur la puce réalisant le circuit. Ces ressources sont composées de segments (de longueurs différentes) qui permettent de relier entre eux les autres éléments via des matrices de connexions. Le routage de ces ressources est un point critique du

développement d'une application sur un FPGA, et les méthodes utilisées pour les ASICs ne sont plus tout à fait valables (du fait de la segmentation des ressources). Si ces éléments sont importants c'est parce qu'ils vont déterminer la vitesse et la densité logique du système. Par exemple les matrices de routage (programmable Switch) sont, physiquement, réalisées grâce à des transistors de cellules SRAMs, qui ont une résistance et une capacité, ce qui entraîne l'existence de constantes de temps**.** [14]

II.4.4- Eléments d'entrées/sorties:

Le composant ne vie pas seul sur une carte (ou très rarement), il appartient à un système d'ensemble pouvant contenir des parties micro programmées comme dans le cas du "Co-Design" (conception conjointe logiciel matériel). Le circuit doit donc avoir un lien avec son environnement, c'est le but des éléments d'entrées/sorties. Ceux-ci peuvent bénéficier de protections, de buffer ou d'autres éléments permettant la gestion des entrées et des sorties. En particulier il est à noter que les circuits actuels proposent différentes normes pour les niveaux d'entrées et de sorties (par exemple: LVTTL 2 - 54mA, PCI 5V, PCI 3.3V, HSTL…) qui par configuration peuvent êtres choisies afin de s'adapter à l'environnement**.** [14]

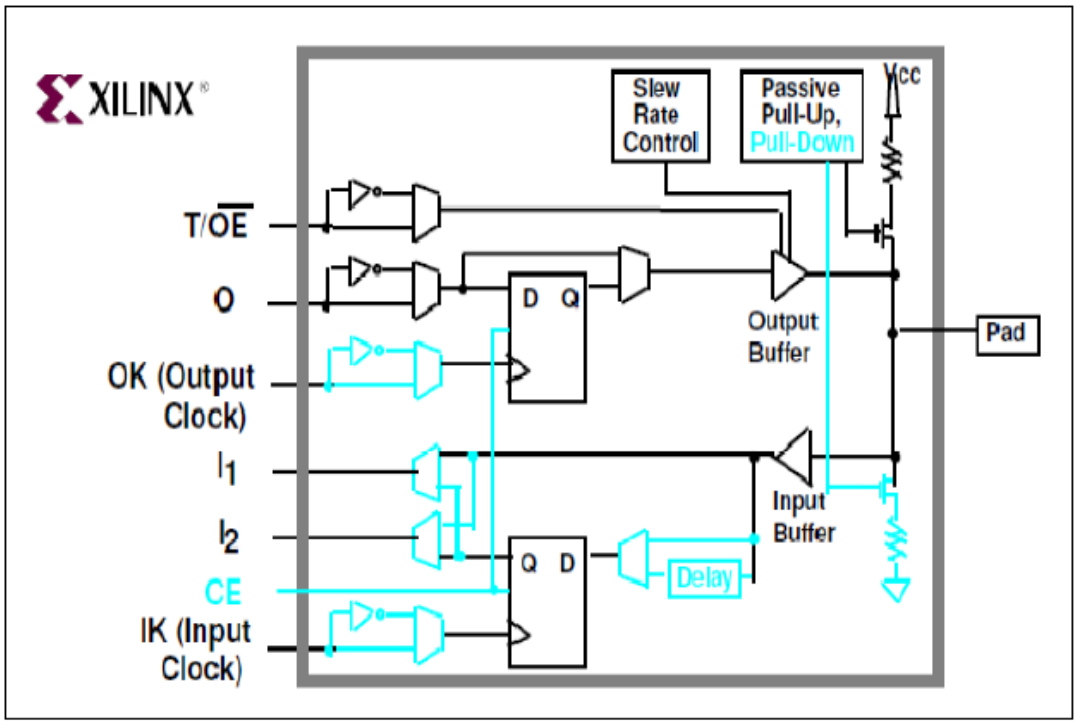


Figure .2.5 : Élément d'entrées sorties (XC4000E-Xilinx)

II.4.5- Eléments de contrôle et d'acheminement des horloges:

Il paraît évident que dans tout système électronique relativement important, il faut disposer d'horloges et qu'elles sont souvent d'une importance capitale pour le bon fonctionnement du système (pensons aux systèmes de télécommunications par exemple). Alors, bien sur, les FPGAs sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routages spécialement adaptées au transport d'horloges sur de longues distances (bufférisassions des lignes). Aussi, pour assurer d'avoir la même horloge dans tout le circuit (synchronisation des signaux) les circuits ont reçu des éléments d'asservissement des horloges (des PLLs ou des DLLs) qui permettent souvent de créer à partir d'une horloge d'autres horloges à des fréquences multiples de la fréquence de l'horloge incidente. Ces arbres d'horloge permettent donc de disposer sur le circuit de fonctions travaillant à des fréquences différentes. Afin de mieux percevoir tous les éléments et architectures précités nous allons présenter les composants Virtex de Xilinx et Apex d'Altera. Ces présentations sont succinctes et ne sont là que pour monter la diversité des architectures de FPGAs, ce qui nous permettra de mieux choisir la modélisation à apporter à ces dernières**.** [14]

II.5- Familles des FPGAs de Xilinx :

Xilinx est une entreprise américaine de semi-conducteurs créé en 1984. Inventeur du FPGA avec un premier produit en 1985, Xilinx fait partie des plus grandes entreprises spécialisées dans le développement et la commercialisation de composants logiques programmables, et des services associés tels que les logiciels de CAO électroniques. Xilinx fabrique une large gamme de FPGA pour diverses applications. En effet, l'offre commerciale de Xilinx est découpée en plusieurs gammes : (FPGA hautes performances : gamme Virtex, FPGA pour la fabrication en grande série : gamme Spartan. Les plus onéreux sont les FPGA Virtex (Virtex II/pro, Virtex4, Virtex5, Virtex6, Virtex7). En 2010, Xilinx a lancé les FPGA-Virtex6, en technologie 40nm, avec 3 catégories différentes (LX, LXT et SXT). Même si les Virtex6 ne disposent pas de processeur PowerPC405, la dernière catégorie est optimisée, spécialement, pour les applications lourdes en traitement numérique du signal. [12]

II.5.1- Etude de la carte FPGA VIRTEX-II du Xilinx:

Dans ce travail, nous allons utiliser la carte FPGA de Xilinx Virtex-II **XC2V1000-4FG456C** qui est représentée dans les **figures 2.6** et **2.7**. Cette carte servira à implémenter un algorithme de traitement d'images. Nous commençons son étude par une présentation des parties constituant la carte Virtex-II.

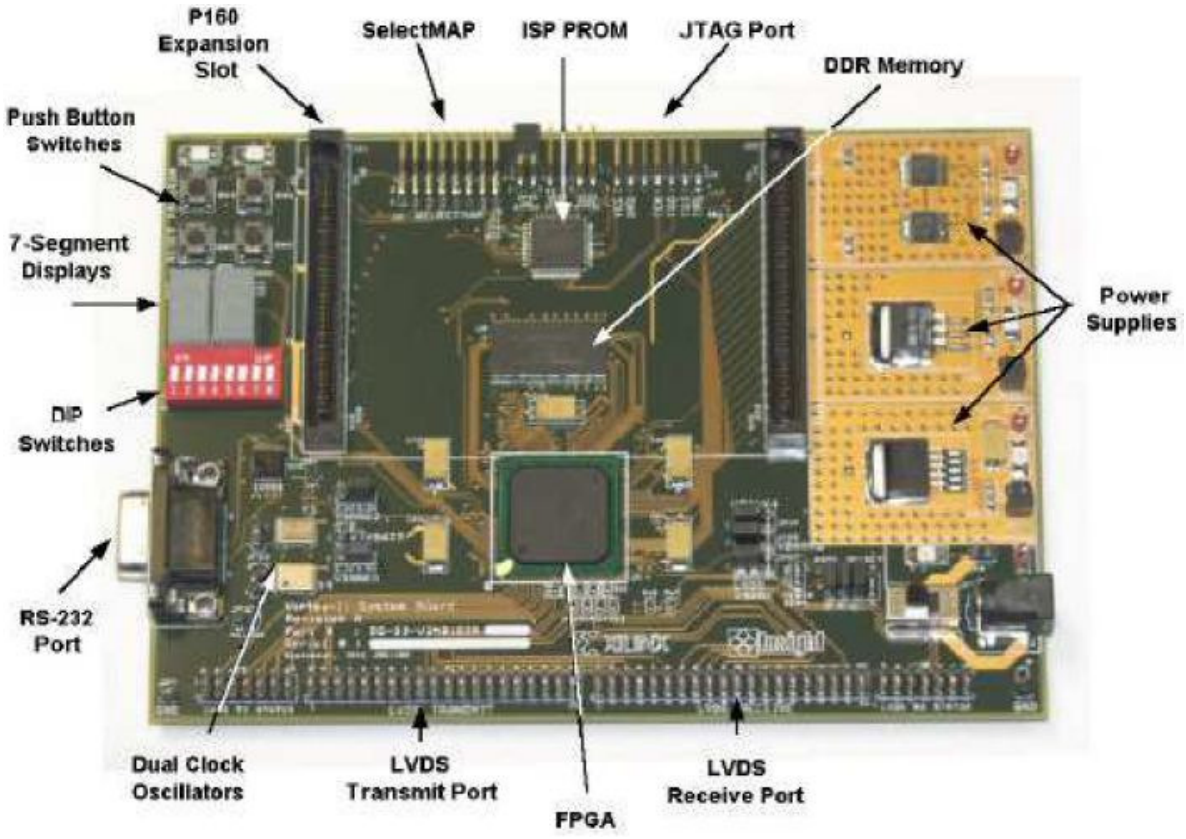


Figure.2.6 : Vue externe de la carte

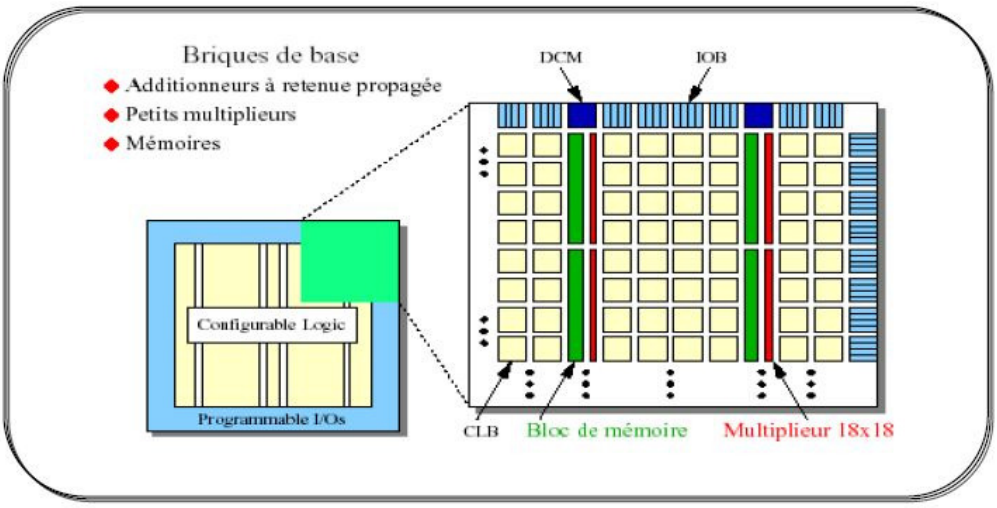


Figure.2.7 : Architecture interne de la famille VIRTEX-II

La carte de développement Virtex-II utilise le dispositif Xilinx Virtex-II XC2V1000-4FG456C. La famille Virtex-II est une plate-forme FPGA développée pour la haute performance, utilisant des IP et des modules personnalisés. La famille Virtex-II fournit des solutions complètes pour les télécommunications, sans fil, réseau, vidéo et DSP applications. La performance et la densité de la famille Virtex-II avec les E/S pris en charge tels que LVDS, PCI et DDR, permettent aux concepteurs de répondre aux exigences des applications de télécommunication et autres. [15]

II.5.1.1- Différents composants de la carte :

On peut résumer les principaux composants de la carte comme suit :

- Elle contient 1 million de portes dans un package de type FG avec 456 pins.
- Elle intègre une mémoire DDR de 16 M *16 et une ISP PROM.
- Elle possède deux clock de 100 M et 24 M contrôlés par des jumpers (respectivement J24 et J23) de plus elle contient un soket pour ajouter d'éventuel oscillateur.
- 2 afficheurs 7 segments pour les données.
- Une LED pour le test qui correspond au pin A9.
- Un Switch et des boutons poussoirs pour le contrôle.
- De port de connexion RS232.
- Le port JTAG qui sert à programmer In Situ Programming (ISP) la PROM et le FPGA.
- Port d'expansion pour connecter une carte esclave si on veut augmenter les ressources.
- Un bloc d'alimentation pour les différents composants.
- Des dizaines de jumpers pour commander les différents composants. En suivant la documentation fournie par le constructeur on modifie les jumpers selon nos besoins.
- Des outils arithmétiques intégrés comme les multiplieurs, les multiplexeurs et les registres. [11]

On détail les composants suivants de la carte:

II.5.1.1.1.Mémoire DDR:

La carte de développement Virtex-II fournit 32 Mo de mémoire DDR. Cette mémoire est mise en œuvre à l'aide du dispositif DDR Micron MT46V16M16TG-75 16Mx16. Le diagramme de haut niveau de l'interface DDR est illustré ci-dessous (**Figure .2.8**) : [15]

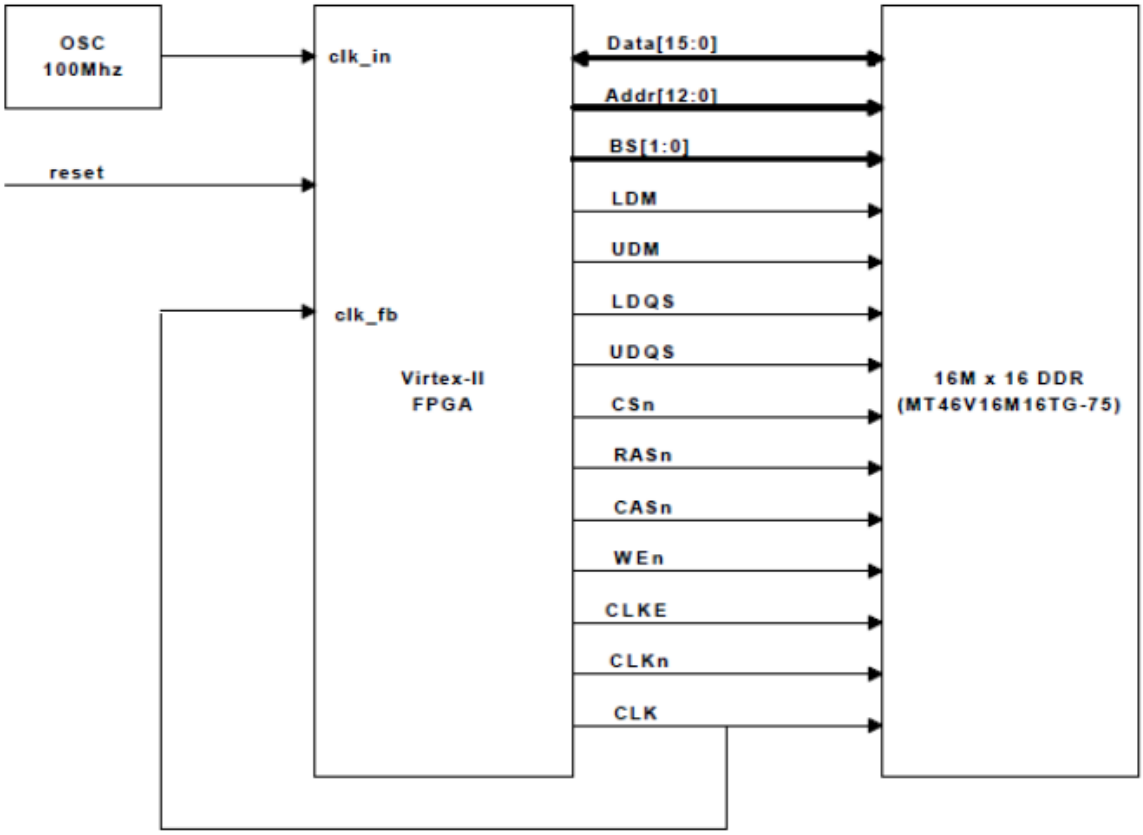


Figure .2.8: Diagramme de haut niveau de l'interface DRD**.**

II.5.1.1.2. Génération d'horloge:

La carte de développement Virtex-II dispose de deux oscillateurs fonctionnant à 100 Mhz (CLK.CAN2) et 24 Mhz (CLK.CAN1). L'oscillateur fonctionnant à 100 Mhz est activé lorsque le cavalier JP24 est ouvert et désactivé lorsque JP24 est fermé. JP23 commande l'oscillateur 24 Mhz. Une troisième prise de l'horloge est fournie pour l'utilisateur. [15]

Le **tableau 2.1** suivant fournit une brève description de ces signaux d'horloge :

Signal Name	Virtex-II Pin #	Direction	Description
CLK.CAN2	B11	Input	On-board 100 MHz Oscillator
CLK.CAN1	A11	Input	On-board 24 MHz Oscillator
CLK.CAN3	F12	Input	User clock socket (2.5V supply)

Tableau 2.1: Description des signaux d'horloge

II.5.1.1.3.Port RS232:

La carte de développement Virtex-II fournit un port RS232 qui peut être géré par le FPGA Virtex-II. Le sous-ensemble des signaux RS232 est utilisé sur la carte de développement Virtex-II pour mettre en œuvre cette simple interface (signaux TD et RD). [15]

II.5.1.1.3.1 Interface RS232:

La carte de développement Virtex-II dispose d'un port DB-9 pour connecter un simple port RS232. La carte utilise le pilote TI MAX3221 RS232 pour piloter les signaux RD et TD. L'utilisateur fournit le Code RS232 UART, qui réside dans l'FPGA Virtex-II. [15]

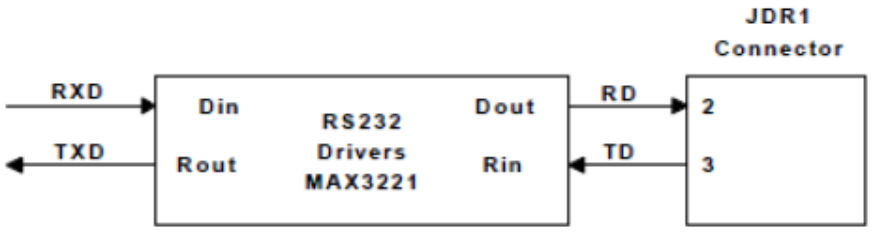


Figure 2.9 : Interface RS232.

II.5.1.1.3.2 Description du Signal RS232:

Le **tableau 2.2** suivant montre les signaux RS232 et leurs affectations aux broches du FPGA Virtex-II. [15]

Signal Name	Virtex-II Pin #	Description
RXD	A7	Received Data, RD to DB9
TXD	B7	Transmit Data, TD from DB9

Tableau 2.2: Affectation des broches du RS232

II.5.1.1.4 Port JTAG :

La carte de développement Virtex-II fournit un connecteur JTAG qui peut être utilisé pour programmer la PROM de la carte et configurer le FPGA Virtex-II. Deux options de connexion sont fournies, J2 est utilisé pour la connexion standard JTAG, et JP29 est utilisé pour la connexion Xilinx parallèle IV avec câble JTAG. [15]

II.5.1.1.4.1 Connecteur standard JTAG :

La **figure 2.10** suivante montre l'affectation des broches du connecteur JTAG J2 sur La carte de développement Virtex-II. [15]

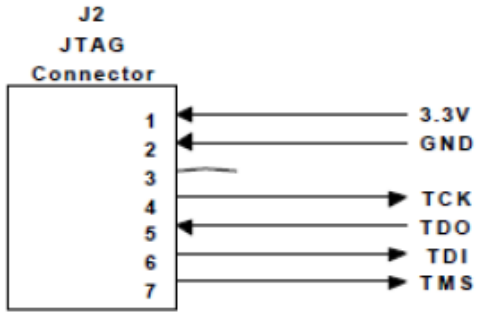


Figure 2.10: Connexion J2 JTAG

II.5.1.1.4.2 Câble parallèle IV Port :

La **figure 2.11** suivante montre les connexions du connecteur du câble parallèle IV. Le Câble IV parallèle peut également être utilisé pour configurer le FPGA via mode Slave de configuration de série. [15]

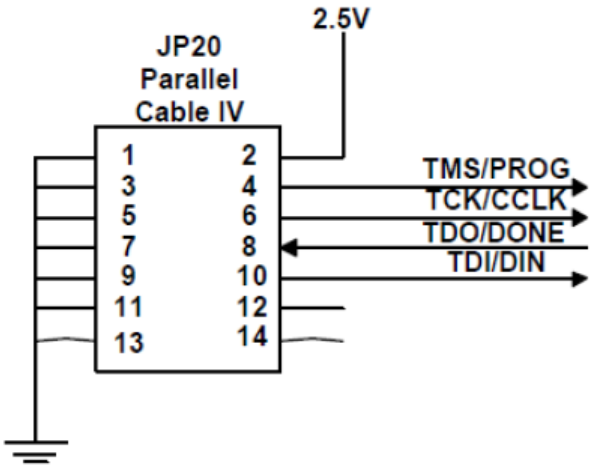


Figure 2.11 : Port parallèle IV JP29

II.5.1.1.4.3 Chaîne JTAG:

La **figure 2.12** suivante montre la chaîne JTAG sur la carte de développement Virtex-II. Le Jumper JP22 offre la possibilité de retirer la PROM de la chaîne JTAG pour la connexion directe à l'FPGA. [15]

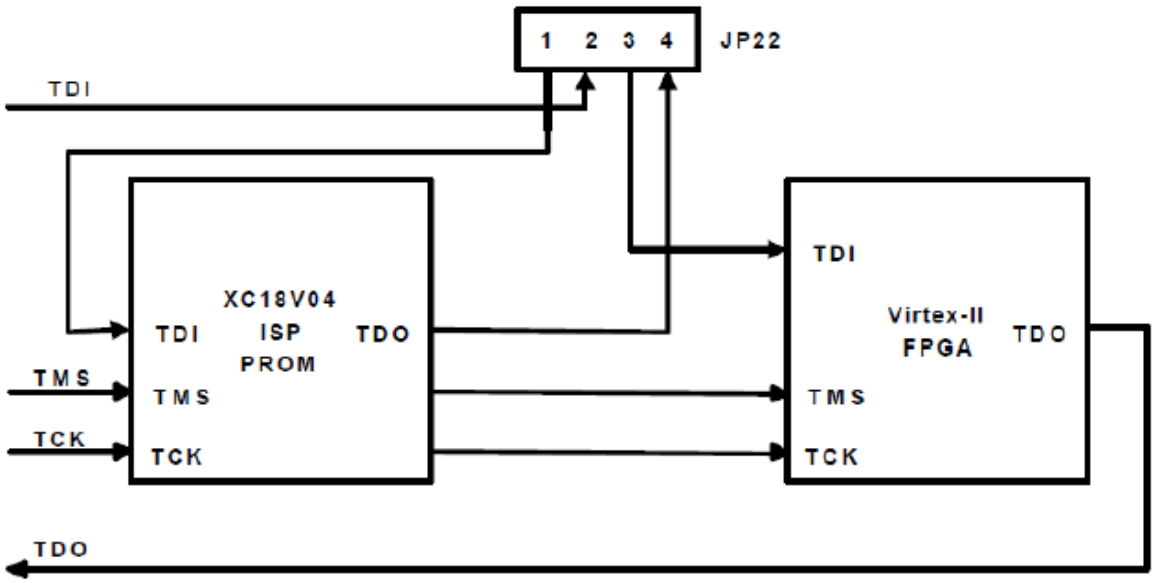


Figure 2.12 : Chaîne JTAG de la carte de développement Virtex-II

II.5.1.1.4.4 Réglages des cavaliers dans la chaîne JTAG :

Le **tableau 2.3** suivant indique les cavaliers à mettre pour avoir la chaine JTAG sur. [15]

Jumper	Setting	Description
JP28	1-2 Closed	Disable PROM
	2-3 Closed	Enable PROM (normal setting)
JP22	1-2, 3-4	PROM in chain (normal setting)
	2-3	Remove PROM from chain (FPGA only)

Tableau 2.3: Configuration de la chaîne JTAG

La carte Virtex II offre alors une très grande gamme de solution pour les applications de télécommunication, réseau, vidéo et les DSP. [11]

Ce tableau donne quelques nombre sur la structure interne de la Virtex II:

Device	System Gates	CLB (1 CLB = 4 slices = Max 128 bits)			Multiplier Blocks	SelectRAM Blocks		DCMs	Max I/O Pads(1)
		Array Row x Col.	Slices	Maximum Distributed RAM Kbits		18-Kbit Blocks	Max RAM (Kbits)		
XC2V1000	1M	40 x 32	5,120	160	40	40	720	8	432

Tableau 2.4 : Quelques nombre sur la structure interne de la Virtex II

La **figure 2.13** suivante résume en général toutes les parties de la carte :

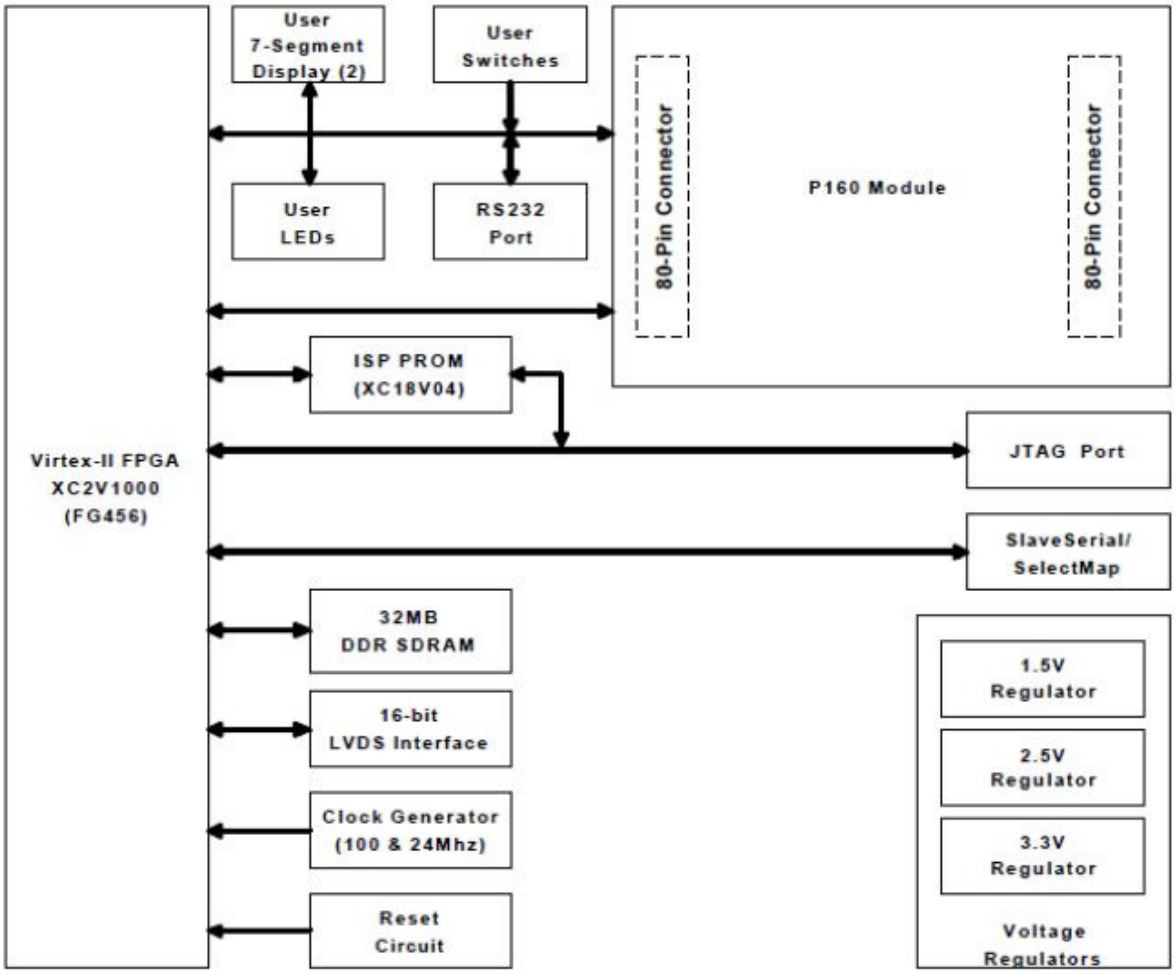


Figure .2.13 : Schéma bloc de la carte de développement Virtex-II

D'après la figure, on définit les parties suivantes de la carte :

II.5.1.2- Différents blocs de la carte :

II.5.1.2.1- Bloc d'entrée/sortie programmable (IOB ou Input/Output Block):

Ils constituent l'interface entre les bornes du circuit et les CLB. Le dispositif de routage VersaRing offre les ressources nécessaires à l'interconnexion des CLB aux IOB. Nous pouvons ainsi modifier le système implanté sur le FPGA sans interférer avec l'attribution des bornes. Cette caractéristique s'avère importante si nous souhaitons développer des

nouvelles versions d'un produit tout en conservant la compatibilité au niveau du boîtier. [11]

II.5.1.2.2. Eléments logiques (CLB ou Configurable Logic Block:

Composés de quatre cellules logiques (LC ou Logic Cell) réparties en deux tranches identiques, ils servent à construire les circuits numériques implantés sur le FPGA. Chaque LC contient essentiellement :

II.5.1.2.2. 1 Eléments logiques configurables (CLBs) :

C'est l'unité fondamentale du bloc logique qui fournit des éléments utilitaires pour la logique combinatoire et la logique synchrone, y compris les éléments du stockage de base : buffers à 3 états à associer avec chaque CLB. Les CLBs incluent quatre parties identiques appelées SLICE et deux buffers à 3 états.

Chaque SILICE est équivalente et contient :

- Deux générateurs de la fonction (F & G).
- Deux éléments du stockage.
- Des portes de la logique arithmétiques.
- Grands multiplexeurs.
- Une large fonctionnalité.
- Une logique de retenue rapide.
- Une chaîne de cascade Horizontale (porte OU). [11]

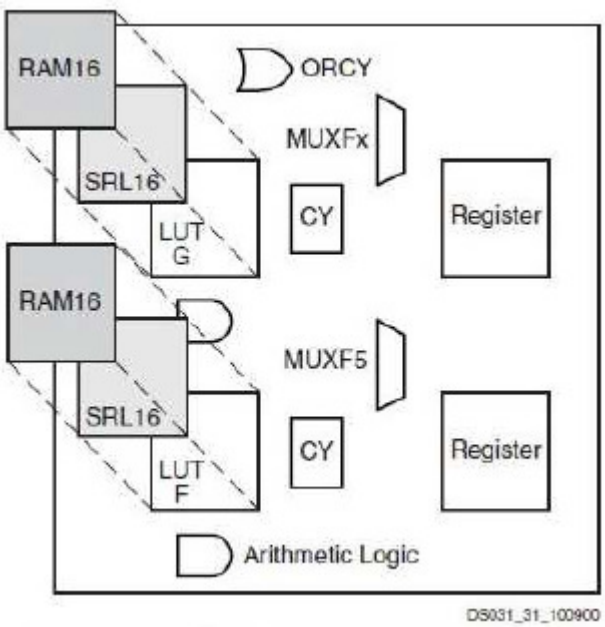


Figure.2.14: Virtex II slice configuration

II.5.1.2.2. 2 Eléments mémoires (Select RAM) :

Ils possèdent des signaux d'initialisation (set et reset) synchrones ou asynchrones. Le bloc Select RAM produit de large élément de stockage (18 Kbit), programmable de 16K x 1 bit à 512 x 36 bits et peut être en deux blocs RAM (dual-port RAM). [11]

II.5.1.2.2.3 Blocs Multiplieurs :

Les blocks multiplieurs sont des multiplieurs de 18x18 bits. Le circuit VIRTEX-II incorpore une grande densité de blocs multiplieurs. Chaque multiplieur peut être associé au bloc Select RAM ou peut être utilisé indépendamment. [11]

II.5.1.2.2.4 Blocs DCM (Digital Clock Manager) :

Le DCM produit le nouveau système d'horloges (soit intérieurement ou extérieurement au FPGA), il produit une large gamme de fréquences de l'horloge de multiplication et même la division, le bloc logique programmable contient plus de 12 DCM.

Par exemple, CLK2X et CLK2X180 doublent la fréquence de l'horloge tandis que CLKDV devise la fréquence sur des nombre : 1.5, 2, 2.5, 3……..,7.5, 8, 9……16.

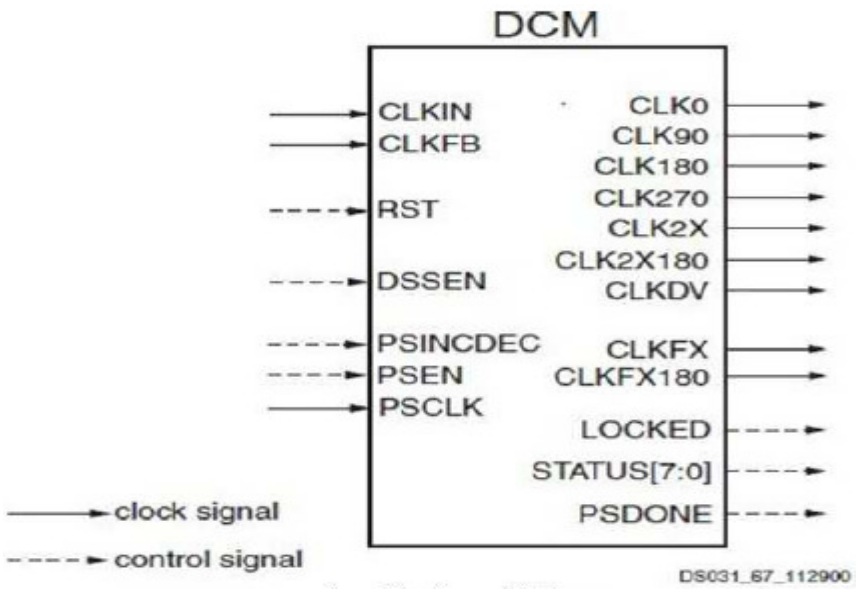


Figure .2.15: Schéma d'un DCM

Pour la construction de la Virtex II, Xilinx a utilisé les technologies de pointe 0.15 µm /0.12 µm CMOS à 8 couches de métallisation, le processus et l'architecture Virtex-II sont optimisés pour une grande vitesse d'horloge avec une faible consommation d'énergie. Elle combine entre la flexibilité dans l'intégration et la densité d'intégration plus de 10 millions de portes logiques pour les dernières versions. [11]

II.6- Utilisation de la carte:

La carte de développement Virtex-II prend en charge plusieurs méthodes de configuration du FPGA Virtex-II. Le port JTAG sur la carte de développement FPGA Virtex-II peut être utilisé pour configurer directement le FPGA Virtex-II, ou de programmer le XC18V04 FAI PROM. Une fois la PROM programmée, elle peut être utilisée pour configurer le FPGA Virtex-II. Le port série SelectMap/esclave sur cette carte de développement peut également être utilisé pour configurer le FPGA Virtex-II.

La **figure 2.12** ci-après montre le branchement pour tous les modes de configuration FPGA Virtex-II qui sont pris en charge sur La carte de développement Virtex-II. [11]

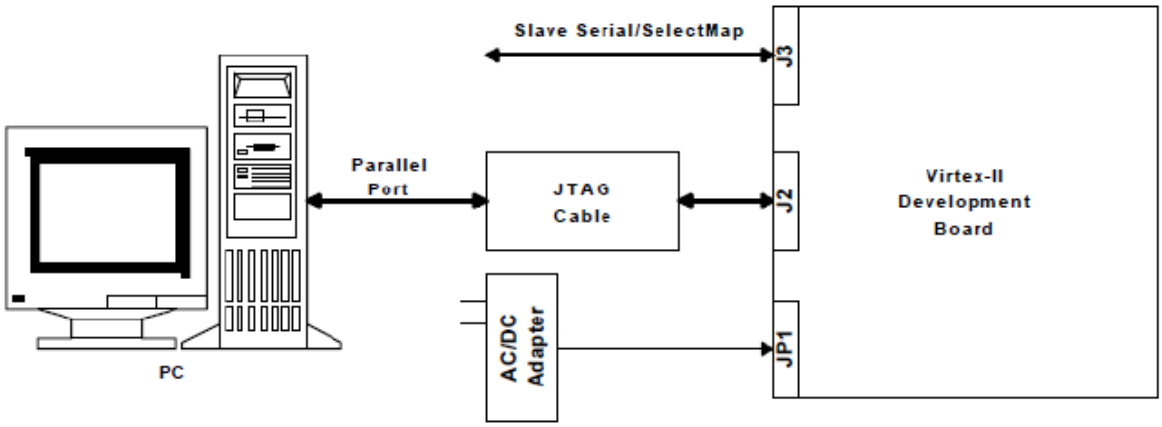


Figure 2.16 : Branchement des modes de configuration

II.6.1- Interface JTAG :

Le connecteur J2 JTAG sur la carte de développement FPGA Virtex-II peut être utilisé pour configurer le Virtex-II ou de programmer le XC18V04 FAI PROM. Le câble JTAG est connecté à la carte de développement Virtex-II via J2 à une extrémité et au port parallèle d'un PC à l'autre extrémité. [15]

II.6.1.1- Configuration du FPGA Virtex-II :

Lorsque le port JTAG est utilisé pour configurer le FPGA Virtex-II, les mesures suivantes doivent être prises:

- En utilisant le **tableau 2.5** régler le mode de configuration du FPGA Virtex-II en mode JTAG.
- Utiliser l'utilitaire de programmation JTAG Xilinx (IMPACT) pour charger le fichier binaire de conception dans le FPGA Virtex-II, associer la PROM FAI soit à un fichier dummy.mcs ou à un fichier .bsd afin de permettre au logiciel de programmation JTAG de transmettre les données à la PROM.[15]

II.6.1.2- Programmation de la XC18V04 FAI PROM :

Lorsque le port JTAG est utilisé pour programmer le FAI PROM, effectuer les étapes suivantes :

- En utilisant le **tableau 2.5** régler le mode de configuration du FPGA Virtex-II en Mode Master Serial ou Master SelectMap.
- Utiliser l'utilitaire de programmation JTAG Xilinx (IMPACT) pour charger le fichier de conception sur la PROM, associer le FPGA soit à un fichier fictif .bits ou à un fichier .bsd. permettant au logiciel de programmation JTAG de transmettre les données à l'FPGA. [15]

Mode	PC Pull-up	J1			
		1-2 (M0)	3-4 (M1)	5-6 (M2)	7-8 (M3)
Master Serial	No	Closed	Closed	Closed	Closed
Master Serial	Yes	Closed	Closed	Closed	Open
Slave Serial	No	Open	Open	Open	Closed
Slave Serial	Yes	Open	Open	Open	Open
Master SelectMap	No	Closed	Open	Open	Closed
Master SelectMap	Yes	Closed	Open	Open	Open
Slave SelectMap	No	Open	Open	Closed	Closed
Slave SelectMap	Yes	Open	Open	Closed	Open
JTAG	No	Open	Closed	Open	Closed
JTAG	Yes	Open	Closed	Open	Open

Tableau 2.5: Configuration du mode sélectionné

II.7-Conclusion :

Dans ce chapitre nous avons vu les FPGAs et la carte Virtex II de Xilinx en particulier, ainsi les périphériques constituants cette carte.

Chapitre III : Présentation et Simulation

III.1 Introduction :

Parfois, l'image prise à partir d'un appareil photo numérique n'est pas de qualité et a nécessité une certaine amélioration. Dans le chapitre I nous avons décrit l'image numérique et ses caractéristiques, types et Système de traitement. En se basant sur ces derniers on va d'abord présenter une application de traitement d'une image.

De nombreuses techniques peuvent améliorer une image numérique sans la gâcher. L'objectif principal de l'amélioration d'image est de traiter une image donnée afin que le résultat soit plus approprié que l'image originale pour une application spécifique.

III.2 Présentation de notre application :

Dans ce Chapitre, certaines opérations de traitement simples d'image sont implémentées dans Verilog telles que l'inversion, le contrôle de la luminosité et binarisation. L'opération de traitement d'image est sélectionnée puis les données d'image traitées sont écrites dans une image bitmap.

III.3 Description de notre application :

III.3 .1 Lecture d'image :

Tout d'abord, Verilog ne peut pas lire directement les images. Pour lire l'image .bmp dans Verilog, l'image doit être convertie du format bitmap au format hexadécimal. L'exemple de code Matlab présenté en annexe pour convertir une image bitmap en fichier .hex. La taille de l'image d'entrée est de 768 x 512 et le fichier image .hex inclut les données R, G, B de l'image bitmap.

Pour lire le fichier hexadécimal d'image, Verilog utilise cette commande:<<$readmemh…..>>. Après avoir lu le fichier image ".hex", les données d'image RGB sont enregistrées dans la mémoire et lues pour traitement.

III.3 .2 Traitement d'image :

L'opération de traitement d'image est sélectionnée dans le fichier "parameter.v" présenté en annexe. Pour changer l'opération, il suffit de changer de ligne de commentaire.

Le fichier "parameter.v" sert également à définir les chemins et les noms des fichiers d'entrée et de sortie. Après avoir traité l'image, il est nécessaire d'écrire les données traitées sur une image de sortie pour vérification.

III.3 .2.1 Manipulation de la luminosité:

Une zone sombre dans une image peut devenir plus claire après l'opération ponctuelle et l'opération ponctuelle couramment utilisée augmente et diminue la luminosité. Si un opérateur prend chaque valeur de pixel et y ajoute un nombre constant, cette opération

ponctuelle augmente la luminosité de l'image et un opérateur de soustraction similaire réduit la luminosité. Le but du code Verilog présenté en annexe est d'ajouter et de soustraire une valeur constante aux valeurs des pixels de l'image.

III.3 .2.2. Images négatives :

Inverser toute la gamme de contraste produit l'équivalent d'un négatif photographique, ce qui améliore parfois la visibilité des détails. Par exemple, les images radiographiques sont généralement examinées à l'aide de négatifs. L'inversion d'une partie seulement de la plage de luminosité produit un effet visuellement étrange qui peut être utilisé pour afficher les détails dans les zones ombrées et saturées.

L'inversion d'une image d'intensité est une opération ponctuelle simple qui inverse l'ordre des valeurs des pixels (multipliant par -1) et ajoute une valeur constante pour mapper le résultat à la plage autorisée.

Par exemple, en considérant la fonction décrite pour un la valeur en pixels étant a = I (u, v), dans la plage de [0, a_{max}], le résultat de l'opération de point est

$$f_{invert}(a) = -a + a_{max} = a_{max} - a \qquad \text{(III.1)}$$

Afin de convertir une image couleur en une image en niveaux de gris, les trois composantes de couleur de chaque pixel doivent être égalisées et une procédure commune consiste à faire la moyenne des trois composants de couleur.

$$R_{eq} = G_{eq} = B_{eq} = \frac{R + G + B}{3} \qquad \text{(III.2)}$$

L'inversion pour une image en niveaux de gris 8 bits avec un a_{max} = 255 est présenté comme un exemple de fonctionnement inversé.

III.3 .2.3 Opérations de seuillage:

Les opérations de seuillage sont particulièrement intéressantes pour segmentation en cours d'isolement d'un objet d'intérêt de son arrière-plan.

Le seuillage d'une image signifie transformer tous les pixels en deux valeurs seulement. Il s'agit d'un type spécial de quantification comparer les valeurs des pixels à une valeur de seuil donnée *a* th c'est généralement constant. Cela permet de séparer les valeurs de pixel dans deux classes. La fonction de seuil décrite $f_{Threshold}(a)$ doit mapper tous les pixels à l'une des deux intensités fixes les valeurs *de a* 0 ou *un* a1

$$f_{threshold}(a) = \begin{cases} a_0 & for\ a < a_{th} \\ a_1 & for\ a \geq a_{th} \end{cases} \qquad \text{(III.3)}$$

avec a 0 <a_{th} <a max [4].

Une image en noir et blanc (binaire) peut être obtenue à partir d'une image en niveaux de gris via une opération de seuillage.

Dans notre implémentation, nous utilisons le seuillage avec une valeur de seuil fixe (choisie arbitrairement) d'un 8 bits indexé image en niveaux de gris. L'opération de seuillage sera effectuée en scannant les valeurs de chaque pixel de l'image d'entrée et en remplaçant le pixel correspondant dans l'image de destination en utilisant $a_0 = 0$ et $a_1 = 255$. La valeur du seuil peut être établie en ligne avec le code de test Verilog.

III.4 Implémentation :

Dans ce qui suit nous allons présenter l'implémentation de notre application décrit avec le langage de description Verilog en utilisant Le navigateur de projet ISE.

Nous commençons ainsi par une brève description du langage Verilog, ainsi que le navigateur de projet ISE. Par la suite, nous présenterons l'algorithme d'application sous Verilog, et testeront sa fonctionnalité par des simulations.

III.4.1 Langage de description Verilog :

Le Verilog, de son nom complet Verilog HDL est un langage de description matériel de circuits logiques en électronique, utilisé pour la conception d'ASICs (application-specific integrated circuits, circuits spécialisés) et de FPGAs (field-programmable gate array).

Le sigle anglais HDL -Hardware Description Language- signifie Langage de Description du Matériel. « Verilog HDL » ne doit pas être abrégé en VHDL, ce sigle étant utilisé pour le langage concurrent VHSIC Hardware Description Language. [16].

III.4.2 Environnement de développement ISE :

Le navigateur de projet ISE sera utilisé comme outil de conception. Cet outil de Xilinx permet de créer des projets comportant plusieurs types de fichiers (HDL, schématique, UCF, EDIF, etc.), de compiler, d'effectuer des designs rule check (DRC), de créer des contraintes d'implémentation dont des contraintes de timing sur les horloges, de déterminer l'emplacement des broches, de créer des bancs d'essais de simulations (testbench) et de gérer efficacement les projets d'envergure. Le navigateur de projet ISE offre un environnement de conception centralisé extrêmement efficace qui regroupe tous les outils nécessaires à la conception, la simulation et à l'implémentation d'un projet d'envergure ainsi qu'à la configuration de la carte FPGA.

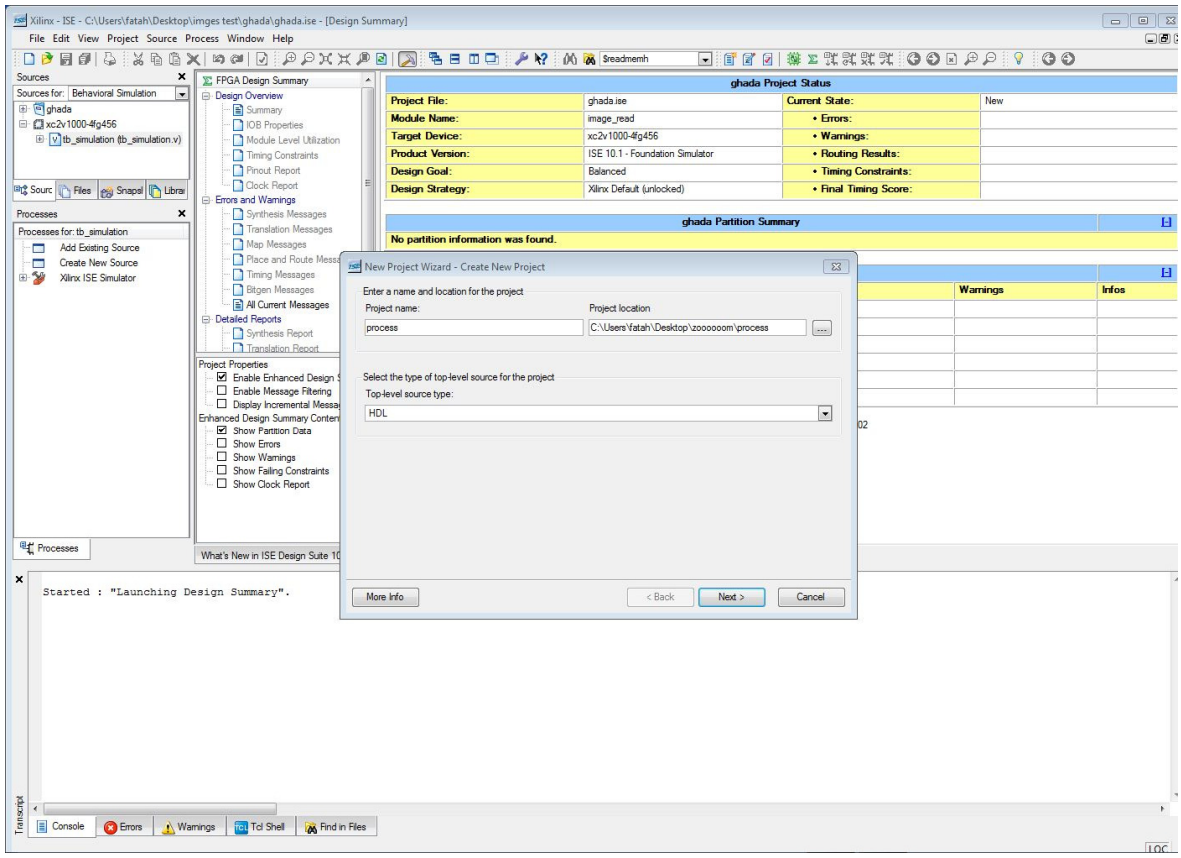


Figure .3.1: Capture d'écran de l'ISE

III.4.3 Simulation de notre application :

La simulation fait à l'aide de l'environnement de développement ISE. Dans ce dernier on va introduire le code d'application de traitement d'image sous Verilog, comme le montre la figure suivante :

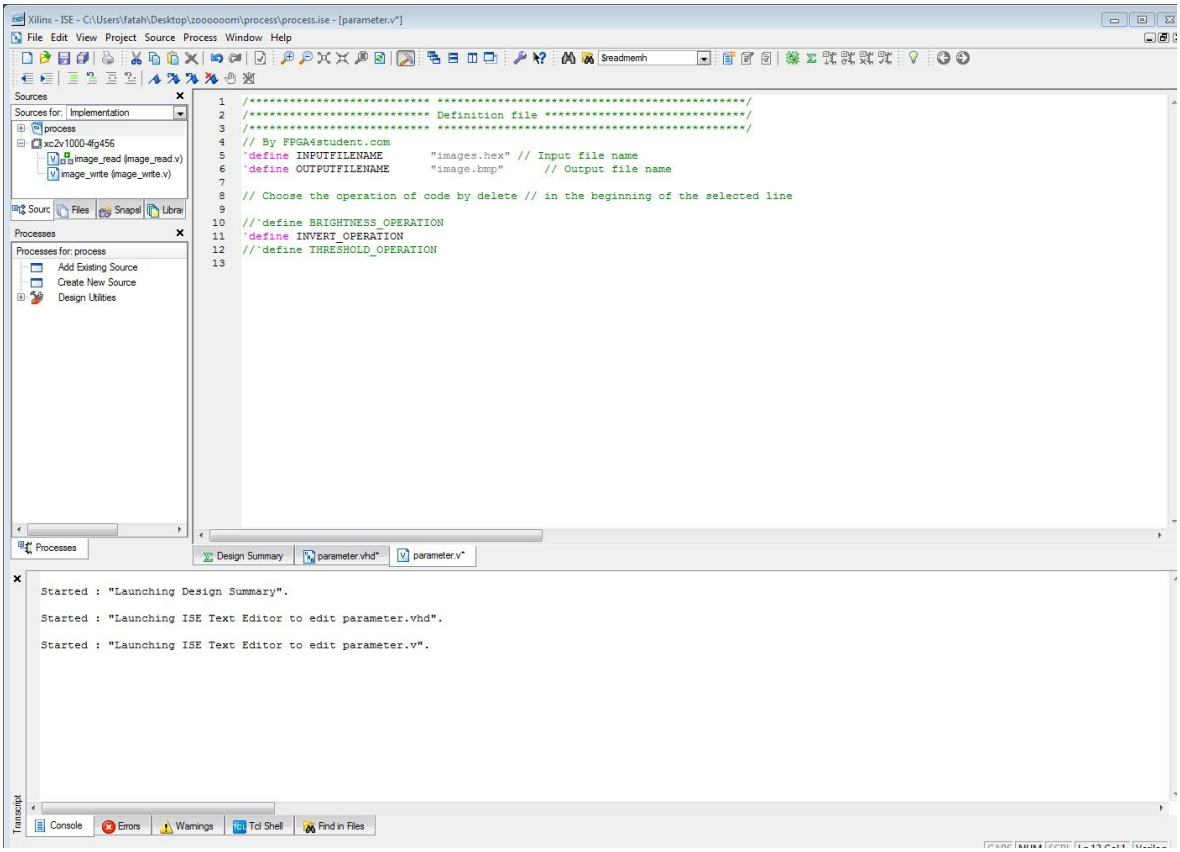


Figure 3.2 : Code de parameter.v sous ISE

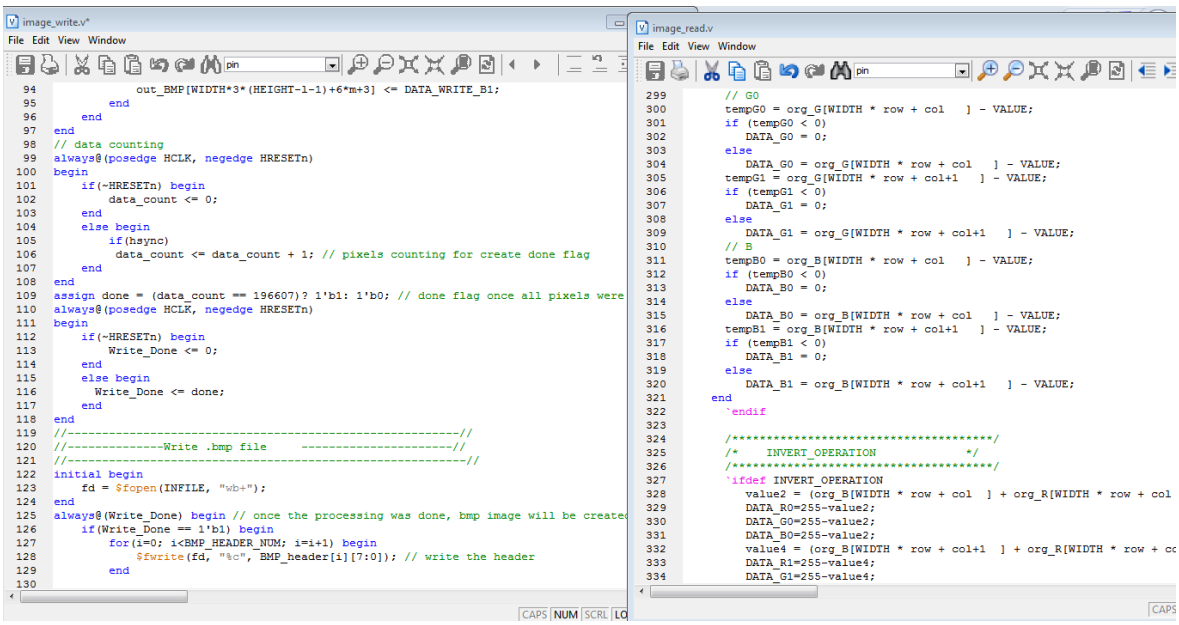


Figure 3.3 : Code de lecture et écriture d'image sous ISE

Une fois la saisie des codes de notre application terminée, on clique sur save pour enregistrer le travaille puis nous apportons le ficher hexadécimal de l'image et la mettons dans le fichier de notre travail.

Utilisons l'image suivante comme fichier bitmap d'entrée:

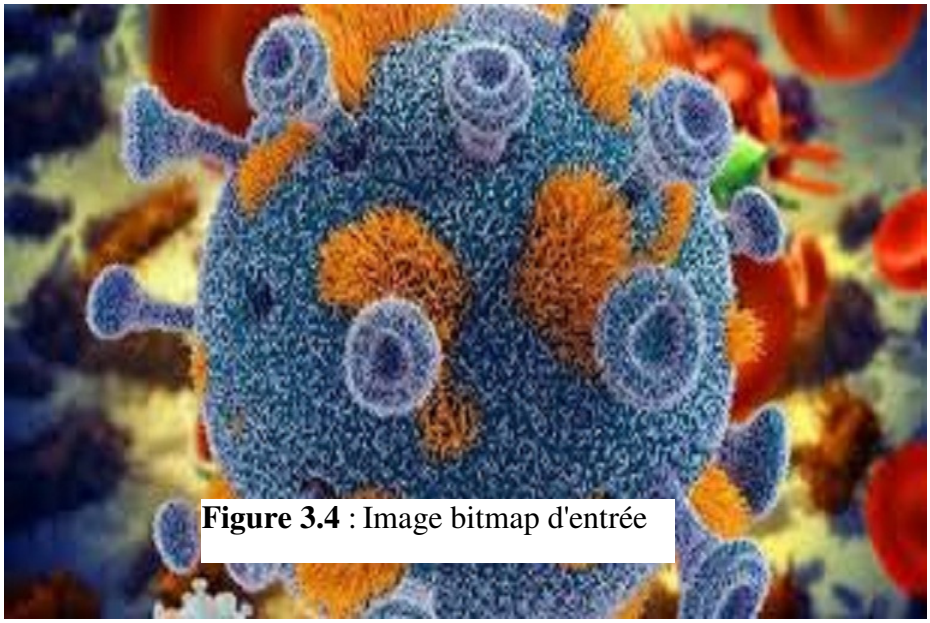


Figure 3.4 : Image bitmap d'entrée

Figure 3.5 : Déplacement de fichier hexadécimale d'image

Ensuit, nous revenons à ISE et cliquons sur "simulate behavioral model " pour exécuter le simulator puis on tape dans le console "run 6ms " pour lancer la simulation.

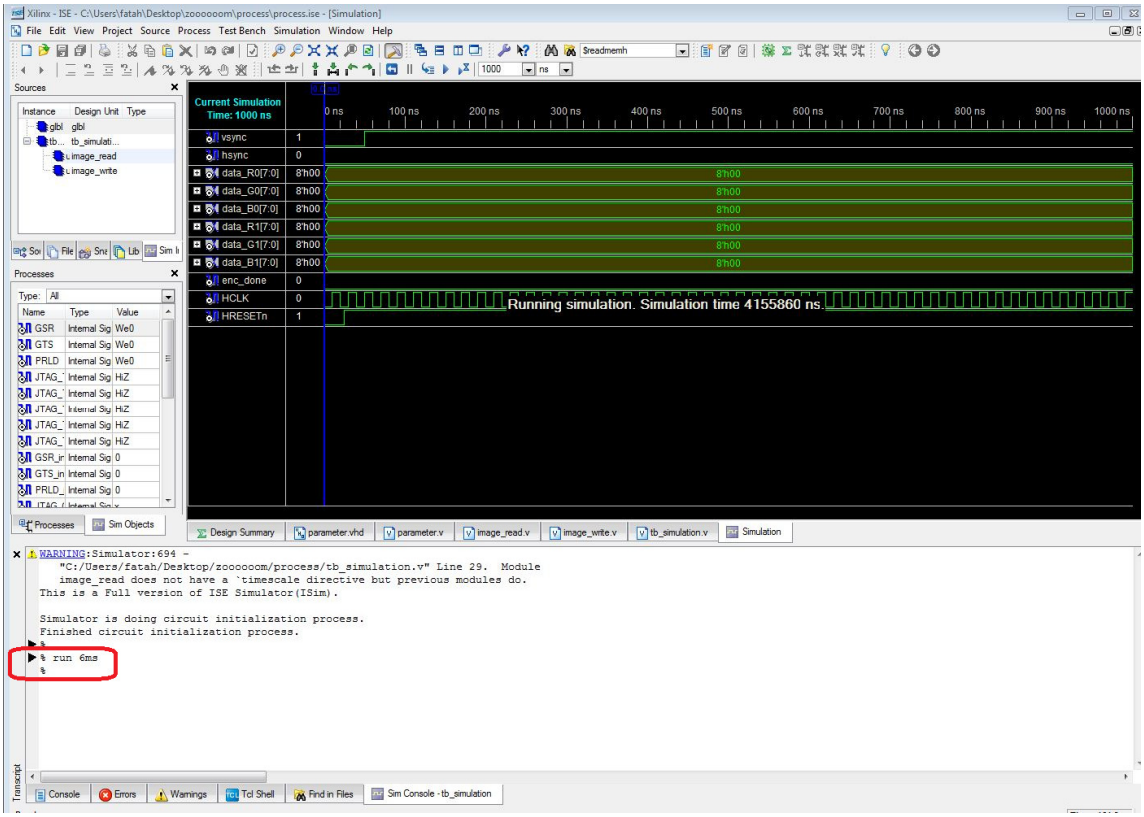


Figure 3.6: Lancement de simulation

Enfin, nous fermons la simulation et ouvrons l'image de sortie pour vérifier le résultat. Voici les images de sortie qui sont traitées par les opérations sélectionnées dans parameter.v:

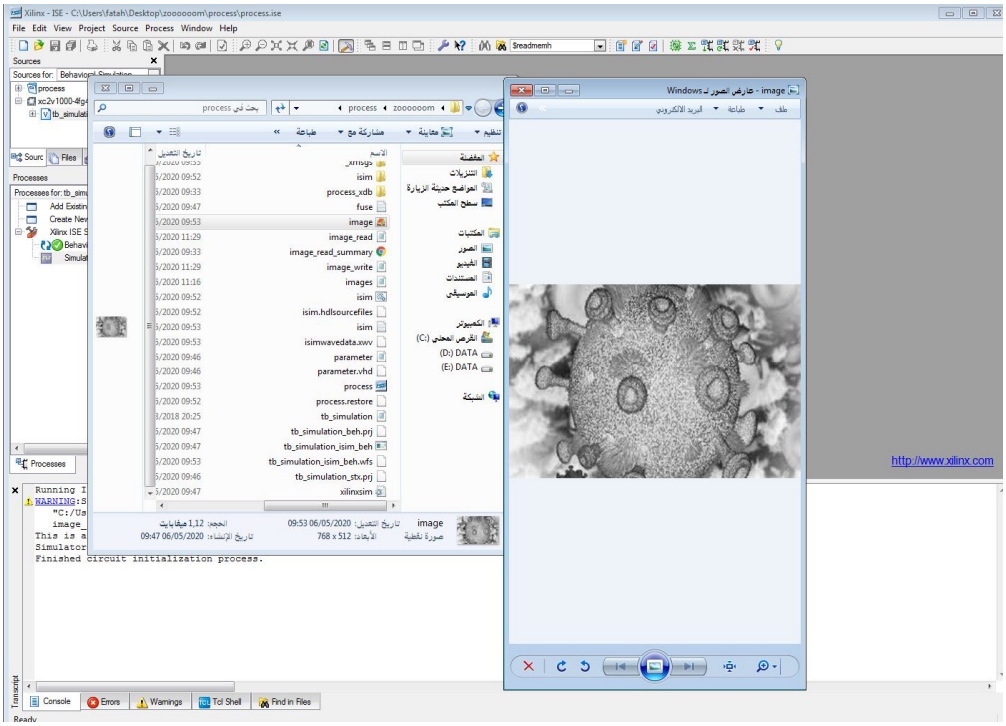


Figure 3.7: Image résultante après le traitement

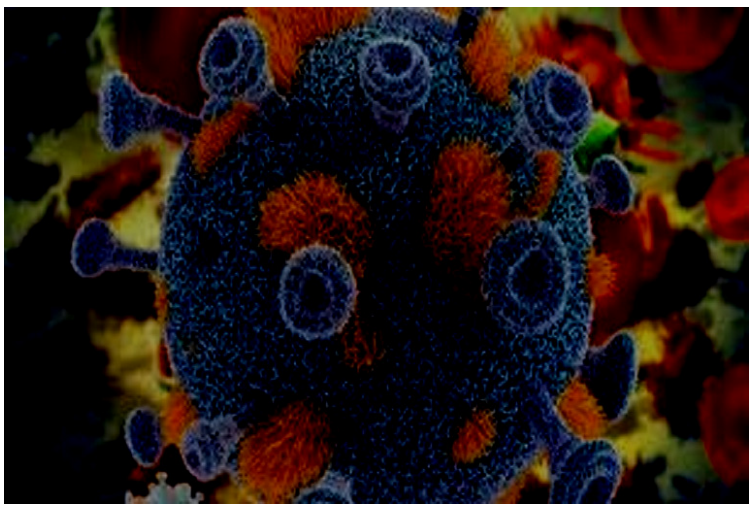


Figure 3.8: Image bitmap de sortie après diminuer la luminosité

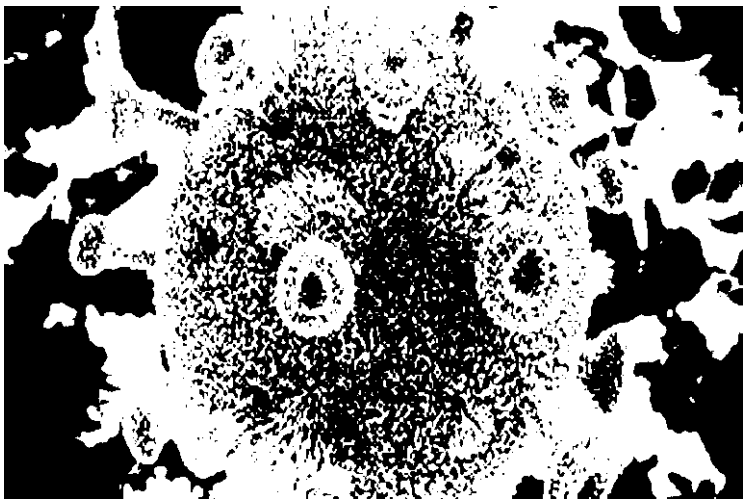


Figure 3.9: Image bitmap de sortie après l'opération de seuillage

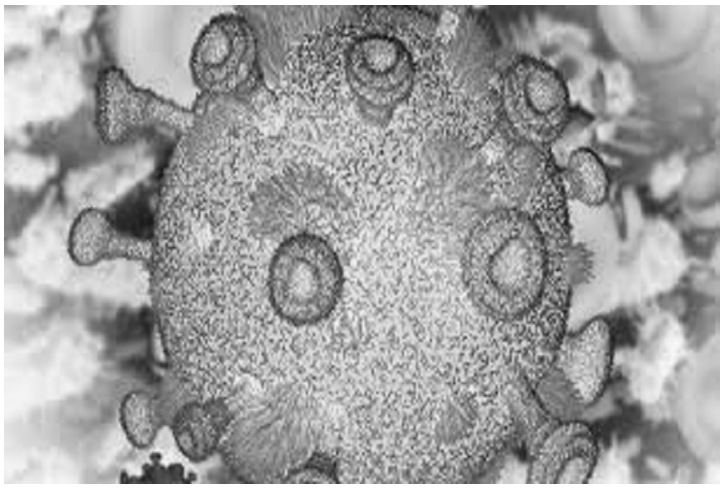


Figure 3.10: Image bitmap de sortie après l'inversion

III.5.Conclusion :

Dans ce chapitre, nous avons simulé dans l'environnement ISE à partir d'un programme en Verilog certaines opérations de traitement simples d'images telles que l'inversion, le contrôle de la luminosité et les opérations de seuillage.

Conclusion et perspectives

Conclusion et perspectives:

Les systèmes embarqués représentent un enjeu majeur dans les nouvelles technologies vu qu'ils équipent en grande partie les équipements et matériels utilisés au quotidien et ils ne cessent d'être le sujet de travaux de recherche afin de les développer.

Le traitement d'images est défini comme l'ensemble des techniques qui permettent de calculer à partir d'une image d'entrée une nouvelle image de sortie.

Notre travail consiste à implémenter un algorithme de traitement d'image sur un système embarqué de type FPGA équipant la carte de développement Virtex-II. Malheureusement, avec la fermeture continue du laboratoire universitaire en raison de la pandémie, nous n'avons pas pu tester notre travail sur la carte, mais nous avons fait une simulation avec ISIM de l'ISE et nous avons obtenu des résultats très satisfaisants.

Ce travail nous a permis d'enrichir nos connaissances théoriques au cours de notre cursus d'études.

En conclusion, pour que notre système forme un produit final, nous devrons le avec une caméra pour fonctionner en temps réel.

Bibliographie

Bibliographie

- [1] H. BEN CHEIKH, Z.KAOUDJA, Recherche d'images sémantique, UNIVERSITÉ KASDI MERBAH OUARGLA.2015

- [2] S.BENFRIHA, A.HAMEL, Segmentation d'image par Coopération région-contours, UNIVERSITÉ KASDI MERBAH OUARGLA.2016

- [3] D.BOUKHLOUF, *Résolution de problèmes par écosystèmes: Application au traitement d'images*, UNIVERSITE MOHAMED KHIDER-BISKRA, 2005.

- [4] A. Zerougui, Traitement d'images monochromes Détection de contours, Filtrage. (Spatial et fréquentiel) et Segmentation par Réseaux de Neurones, UNIVERSITÉ LARBI BEN M'HIDI OUM EL BOUAGHI. 2017

- [5] Introduction: image bitmap versus image vectorielle https://www.sites.univ-rennes2.fr/arts-spectacle/cian/imagenumFlash/pdf/chap3_cours35.pdf

- [6] S. Brahim, D. Belmesmar, Traitement d'image et morphologie mathématique, UNIVERSITÉ KASDI MERBAH OUARGLA.2016

- [7] Y. NEDJAR, I. MOUSSI, Application des méthodes numériques de traitement d'image sous Android, UNIVERSITÉ KASDI MERBAH OUARGLA.2018

- [8] A. ZEROUAL, Implémentation d'un contrôleur à logique floue sur une carte FPGA Virtex-II Pro-LC XC2VP4-5 FG456C, UNIVERSITÉ LARBI BEN M'HIDI OUM EL BOUAGHI.2011

- [9] T. NACHEF, Implémentation d'une instrumentation sur un FPGA, UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU.2011

- [10] M. AIT BENALI, A. AIT BENALI, Conception et réalisation d'une plateforme d'acquisition reconfigurable a base d'un circuit FPGA, UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU.2011

- [11] M. Idir, Implémentation d'un réseau de neurones d'un micro capteur sur un FPGA, UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU. 2010

- [12] D. SNAOUI, Commande Numérique en Force à Base de la Carte FPGA d'une Architecture de Télé-opération à un Seul Degré de Liberté, UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU.2016

- [13] S. DJAMA, A. MANSEUR, étude et implémentation d'un modulateur FSK sur circuit FPGA, UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU.2011

- [14] H.ABBAD, Méthodologie de développement et d'implantation sur puce FPGA d'algorithme de commande, UNIVERSITE MOHAMED KHIDER-BISKRA. 2016

- [15] M.HOUADJ, Synthèse d'un module de cryptage RSA sur un Circuit de type FPGA, UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU.2014

- [16] Verilog : https: //fr.wikipedia.org/wiki/Verilog

Annexe

Annexe:

1. Code matlab pour convertir l'image bmp à hex :

```
b=imread('image.bmp');

k=1;
for i=512:-1:1
for j=1:768
a(k)=b(i,j,1);
a(k+1)=b(i,j,2);
a(k+2)=b(i,j,3);
k=k+3;
end
end
fid = fopen('image.hex', 'wt');
fprintf(fid, '%x\n', a);
disp('Text file write done');disp(' ');
fclose(fid);
```

2. Code Verilog de lecture et traitement d'image :

```
`include "parameter.v"
module image_read
#(
  parameter     WIDTH  = 768,
         HEIGHT  = 512,
  INFILE  = "./img/kodim01.hex",
  START_UP_DELAY = 100,
  HSYNC_DELAY = 160,
  VALUE= 100,
  THRESHOLD= 90,
  SIGN=1

)
(
  input HCLK,
  input HRESETn,
  output VSYNC,

  output reg HSYNC,
  output reg [7:0]  DATA_R0,
  output reg [7:0]  DATA_G0,
  output reg [7:0]  DATA_B0,
  output reg [7:0]  DATA_R1,
  output reg [7:0]  DATA_G1,
  output reg [7:0]  DATA_B1,
  output     ctrl_done
);
parameter sizeOfWidth = 8;
parameter sizeOfLengthReal = 1179648;
localparam  ST_IDLE  = 2'b00,
   ST_VSYNC = 2'b01,
   ST_HSYNC = 2'b10,
```

```
   ST_DATA  = 2'b11;
reg [1:0] cstate,
nstate;
reg start;
reg HRESETn_d;
reg   ctrl_vsync_run;
reg [8:0] ctrl_vsync_cnt;
reg   ctrl_hsync_run;
reg [8:0] ctrl_hsync_cnt;
reg   ctrl_data_run;
reg [7 : 0]   total_memory [0 : sizeOfLengthReal-1];

integer temp_BMP   [0 : WIDTH*HEIGHT*3 - 1];
integer org_R  [0 : WIDTH*HEIGHT - 1];
integer org_G  [0 : WIDTH*HEIGHT - 1];
integer org_B  [0 : WIDTH*HEIGHT - 1];
integer i, j;

integer tempR0,tempR1,tempG0,tempG1,tempB0,tempB1;
integer value,value1,value2,value4;
reg [ 9:0] row;
reg [10:0] col;
reg [18:0] data_count;
initial begin
    $readmemh(INFILE,total_memory,0,sizeOfLengthReal-1);
end
always@(start) begin
    if(start == 1'b1) begin
        for(i=0; i<WIDTH*HEIGHT*3 ; i=i+1) begin
            temp_BMP[i] = total_memory[i+0][7:0];
        end

        for(i=0; i<HEIGHT; i=i+1) begin
            for(j=0; j<WIDTH; j=j+1) begin

org_R[WIDTH*i+j]=temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+0];
org_G[WIDTH*i+j]=temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+1;
org_B[WIDTH*i+j]=temp_BMP[WIDTH*3*(HEIGHT-i-1)+3*j+2;
            end
        end
    end
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(!HRESETn) begin
        start <= 0;
  HRESETn_d <= 0;
    end
    else
   begin

        HRESETn_d <= HRESETn;
  if(HRESETn == 1'b1 && HRESETn_d == 1'b0)
   start <= 1'b1;
  else
   start <= 1'b0;
```

```
    end
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        cstate <= ST_IDLE;
    end
    else begin
        cstate <= nstate;
    end
end
always @(*) begin
 case(cstate)
  ST_IDLE: begin
   if(start)
    nstate = ST_VSYNC;
   else
    nstate = ST_IDLE;
  end
  ST_VSYNC: begin
   if(ctrl_vsync_cnt == START_UP_DELAY)
    nstate = ST_HSYNC;
   else
    nstate = ST_VSYNC;
  end
  ST_HSYNC: begin
   if(ctrl_hsync_cnt == HSYNC_DELAY)
    nstate = ST_DATA;
   else
    nstate = ST_HSYNC;
  end
  ST_DATA: begin
   if(ctrl_done)
    nstate = ST_IDLE;
   else begin
    if(col == WIDTH - 2)
     nstate = ST_HSYNC;
    else
     nstate = ST_DATA;
   end
  end
 endcase
end
always @(*) begin
 ctrl_vsync_run = 0;
 ctrl_hsync_run = 0;
 ctrl_data_run  = 0;
 case(cstate)
  ST_VSYNC:  begin ctrl_vsync_run = 1; end
  ST_HSYNC:  begin ctrl_hsync_run = 1; end
  ST_DATA:  begin ctrl_data_run  = 1; end
 endcase
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
```

```
        ctrl_vsync_cnt <= 0;
  ctrl_hsync_cnt <= 0;
    end
    else begin
        if(ctrl_vsync_run)
   ctrl_vsync_cnt <= ctrl_vsync_cnt + 1;
  else
   ctrl_vsync_cnt <= 0;

        if(ctrl_hsync_run)
   ctrl_hsync_cnt <= ctrl_hsync_cnt + 1;
  else
   ctrl_hsync_cnt <= 0;
    end
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        row <= 0;
  col <= 0;
    end
 else begin
  if(ctrl_data_run) begin
   if(col == WIDTH - 2) begin
    row <= row + 1;
   end
   if(col == WIDTH - 2)
    col <= 0;
   else
    col <= col + 2;
  end
 end
end
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        data_count <= 0;
    end
    else begin
        if(ctrl_data_run)
   data_count <= data_count + 1;
    end
end
assign VSYNC = ctrl_vsync_run;
assign ctrl_done = (data_count == 196607)? 1'b1: 1'b0;
always @(*) begin

 HSYNC   = 1'b0;
 DATA_R0 = 0;
 DATA_G0 = 0;
 DATA_B0 = 0;
 DATA_R1 = 0;
 DATA_G1 = 0;
 DATA_B1 = 0;
 if(ctrl_data_run) begin
```

```
 HSYNC   = 1'b1;
 `ifdef BRIGHTNESS_OPERATION

 if(SIGN == 1) begin
 // R0
 tempR0 = org_R[WIDTH * row + col   ] + VALUE;
 if (tempR0 > 255)
  DATA_R0 = 255;
 else
  DATA_R0 = org_R[WIDTH * row + col   ] + VALUE;
 // R1
 tempR1 = org_R[WIDTH * row + col+1   ] + VALUE;
 if (tempR1 > 255)
  DATA_R1 = 255;
 else
  DATA_R1 = org_R[WIDTH * row + col+1   ] + VALUE;
 // G0
 tempG0 = org_G[WIDTH * row + col   ] + VALUE;
 if (tempG0 > 255)
  DATA_G0 = 255;
 else
  DATA_G0 = org_G[WIDTH * row + col   ] + VALUE;
 tempG1 = org_G[WIDTH * row + col+1   ] + VALUE;
 if (tempG1 > 255)
  DATA_G1 = 255;
 else
  DATA_G1 = org_G[WIDTH * row + col+1   ] + VALUE;
 // B
 tempB0 = org_B[WIDTH * row + col   ] + VALUE;
 if (tempB0 > 255)
  DATA_B0 = 255;
 else
  DATA_B0 = org_B[WIDTH * row + col   ] + VALUE;
 tempB1 = org_B[WIDTH * row + col+1   ] + VALUE;
 if (tempB1 > 255)
  DATA_B1 = 255;
 else
  DATA_B1 = org_B[WIDTH * row + col+1   ] + VALUE;
end
else begin


 tempR0 = org_R[WIDTH * row + col   ] - VALUE;
 if (tempR0 < 0)
  DATA_R0 = 0;
 else
  DATA_R0 = org_R[WIDTH * row + col   ] - VALUE;
 // R1
 tempR1 = org_R[WIDTH * row + col+1   ] - VALUE;
 if (tempR1 < 0)
  DATA_R1 = 0;
 else
  DATA_R1 = org_R[WIDTH * row + col+1   ] - VALUE;
 // G0
 tempG0 = org_G[WIDTH * row + col   ] - VALUE;
 if (tempG0 < 0)
```

```
   DATA_G0 = 0;
  else
   DATA_G0 = org_G[WIDTH * row + col    ] - VALUE;
  tempG1 = org_G[WIDTH * row + col+1    ] - VALUE;
  if (tempG1 < 0)
   DATA_G1 = 0;
  else
   DATA_G1 = org_G[WIDTH * row + col+1    ] - VALUE;
  // B
  tempB0 = org_B[WIDTH * row + col    ] - VALUE;
  if (tempB0 < 0)
   DATA_B0 = 0;
  else
   DATA_B0 = org_B[WIDTH * row + col    ] - VALUE;
  tempB1 = org_B[WIDTH * row + col+1    ] - VALUE;
  if (tempB1 < 0)
   DATA_B1 = 0;
  else
   DATA_B1 = org_B[WIDTH * row + col+1    ] - VALUE;
  end
  `endif
  `ifdef INVERT_OPERATION
   value2 = (org_B[WIDTH * row + col   ] + org_R[WIDTH * row + col   ]
+org_G[WIDTH * row + col   ])/3;
   DATA_R0=255-value2;
   DATA_G0=255-value2;
   DATA_B0=255-value2;
   value4 = (org_B[WIDTH * row + col+1   ] + org_R[WIDTH * row + col+1   ]
+org_G[WIDTH * row + col+1   ])/3;
   DATA_R1=255-value4;
   DATA_G1=255-value4;
   DATA_B1=255-value4;
  `endif
  `ifdef THRESHOLD_OPERATION
  value = (org_R[WIDTH * row + col      ]+org_G[WIDTH * row + col
]+org_B[WIDTH * row + col    ])/3;
  if(value > THRESHOLD) begin
   DATA_R0=255;
   DATA_G0=255;
   DATA_B0=255;
  end
  else begin
   DATA_R0=0;
   DATA_G0=0;
   DATA_B0=0;
  end
  value1 = (org_R[WIDTH * row + col+1     ]+org_G[WIDTH * row + col+1
]+org_B[WIDTH * row + col+1    ])/3;
  if(value1 > THRESHOLD) begin
   DATA_R1=255;
   DATA_G1=255;
   DATA_B1=255;
  end
  else begin
   DATA_R1=0;
   DATA_G1=0;
```

```
   DATA_B1=0;
  end
  `endif
 end
end

endmodule
```

3. Code Verilog "parameter.v"

```
`define INPUTFILENAME "your_image.hex"
`define OUTPUTFILENAME "output.bmp"
`define BRIGHTNESS_OPERATION
`define INVERT_OPERATION
`define THRESHOLD_OPERATION
```

4. Code Verilog pour l'écriture d'image de sortie :

```
module image_write #(parameter
WIDTH = 768,
HEIGHT = 512,
INFILE = "output.bmp",
BMP_HEADER_NUM = 54
)
(
input HCLK,
HRESETn,
input hsync,
input [7:0] DATA_WRITE_R0,
input [7:0] DATA_WRITE_G0,
input [7:0] DATA_WRITE_B0,
input [7:0] DATA_WRITE_R1,
input [7:0] DATA_WRITE_G1,
input [7:0] DATA_WRITE_B1,
output reg Write_Done
);

initial  begin
BMP_header[ 0] = 66;BMP_header[28] =24;
BMP_header[ 1] = 77;BMP_header[29] = 0;
BMP_header[ 2] = 54;BMP_header[30] = 0;
BMP_header[ 3] = 0;BMP_header[31] = 0;
BMP_header[ 4] = 18;BMP_header[32] = 0;
BMP_header[ 5] = 0;BMP_header[33] = 0;
BMP_header[ 6] = 0;BMP_header[34] = 0;
BMP_header[ 7] = 0;BMP_header[35] = 0;
BMP_header[ 8] = 0;BMP_header[36] = 0;
BMP_header[ 9] = 0;BMP_header[37] = 0;
BMP_header[10] = 54;BMP_header[38] = 0;
BMP_header[11] = 0;BMP_header[39] = 0;
BMP_header[12] = 0;BMP_header[40] = 0;
BMP_header[13] = 0;BMP_header[41] = 0;
BMP_header[14] = 40;BMP_header[42] = 0;
BMP_header[15] = 0;BMP_header[43] = 0;
BMP_header[16] = 0;BMP_header[44] = 0;
```

```
BMP_header[17] = 0;BMP_header[45] = 0;
BMP_header[18] = 0;BMP_header[46] = 0;
BMP_header[19] = 3;BMP_header[47] = 0;
BMP_header[20] = 0;BMP_header[48] = 0;
BMP_header[21] = 0;BMP_header[49] = 0;
BMP_header[22] = 0;BMP_header[50] = 0;
BMP_header[23] = 2;BMP_header[51] = 0;
BMP_header[24] = 0;BMP_header[52] = 0;
BMP_header[25] = 0;BMP_header[53] = 0;
BMP_header[26] = 1; BMP_header[27] = 0;
end

initial begin
    fd = $fopen(INFILE, "wb+");
end
always@(Write_Done) begin
    if(Write_Done == 1'b1) begin
        for(i=0; i<BMP_HEADER_NUM; i=i+1) begin
            $fwrite(fd, "%c", BMP_header[i][7:0]);
        end

        for(i=0; i<WIDTH*HEIGHT*3; i=i+6) begin
              $fwrite(fd, "%c", out_BMP[i  ][7:0]);
            $fwrite(fd, "%c", out_BMP[i+1][7:0]);
            $fwrite(fd, "%c", out_BMP[i+2][7:0]);
            $fwrite(fd, "%c", out_BMP[i+3][7:0]);
            $fwrite(fd, "%c", out_BMP[i+4][7:0]);
            $fwrite(fd, "%c", out_BMP[i+5][7:0]);
        end
    end
end
```