UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté de technologie

Département d'Electronique

MEMOIRE DE MAGISTER

En électronique

Spécialité : Communication

Architecture pour détection de contour actif Sur FPGA

Par

ABDELLI Lotfi

Devant le jury composé de :

A.Namane	Maitre de conférences A	USD-Blida	Président
A.Nesba	Maitre de conférences A	ENS-Kouba	Examinateur
A.Guessoum	Professeur	USD-Blida	Promoteur
M.Maamoun	Maitre de conférences B	USD-Blida	Invité

Blida, Décembre 2012

ملخص

الأعمال المنجزة تقدم إسهاما في تنفيذ الكشف عن الحافة على FPGA في الوقت الحقيقي، يتم التعامل على محورين المنهج التحليلي و منهج التتبع في محيط النشيط.

في الجزء الأول اعتمدنا على طريقة كاني للتنفيذ و المعالجة في الوقت الحقيقي على FPGA

في الجزء الثاني درسنا و اقترحنا هيكل لتنفيذ منهج التتبع في محيط بالموقع في الوقت الحقيقي على FPGA

RESUME

Notre travail présente une contribution à l'implémentation sur FPGA de détecteurs de contours en temps réel. Deux axes sont traités l'approche analytique et l'approche du contour actif.

Comme première partie nous avons adapté la méthode de canny pour une implémentation et un traitement temps réel sur FPGA.

Dans la deuxième partie, nous avons étudié et proposé une architecture pour le contour actif temps réel implémentable sur FPGA.

Abstract

Our works presents a contribution to the implementation on FPGA of contour detectors in real time. Two axis are treated analytical approach and active contour approach.

At the first part, we adapted the canny method for the implementation and the real time processing on FPGA.

At the second part, we studied and proposed architecture for real time active contour implementable on FPGA.

REMERCIEMENTS

Je tiens tout d'abord à remercier Dr A.Namane, pour avoir accepté de présider le jury lors de ma soutenance. Je remercie aussi Dr M.Maamoun et madame H.Bougherira pour avoir relu ce document avec soin et pour leurs remarques pertinentes qui m'ont permis, j'espère, d'améliorer sa qualité et sa clarté. Je suis également grandement reconnaissant à Dr R.Bradai. Pour bien avoir voulu être membre de mon jury.

Je tiens aussi à remercier madame N.Bouguerira, madame Nacer et madame Assade pour avoir été des collègues, avec qui j'ai pu passer des moments formidables.

Je tiens bien évidemment à remercier tout aussi profondément mes deux directeurs de mémoire, Pr A.Guesoum et Dr M.Maamoun, pour leurs grandes qualités d'encadrants et pour la confiance qu'ils m'ont témoignée. Merci à eux pour m'avoir laissé l'entière liberté dans mes recherches, tout en étant disponible à tout instant pour répondre à mes questions, pour me donner des conseils avisés ou bien encore pour m'encourager dans les moments de doute.

Je réserve une pensée toute particulière à ma chère épouse, mes chers parents qui m'ont toujours soutenu et guidé tout au long de mes études et à ma source d'inspiration mon poupon Samy.

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS, GRAPHIQUES ET TABLEAUX

INTRODUCTION

L'évolution des systèmes de communication numériques, notamment dans les domaines du traitement et de la transmission des images, requièrent des algorithmes de plus en plus complexes nécessitant des puissances de calcul au-delà des performances des processeurs actuels. Deux grandes approches sont actuellement utilisées :

– la mise en parallèle de processeurs qui apportent la souplesse de programmation et la rapidité, mais au détriment du coût et de l'encombrement;

– les circuits spécifiques (ASIC) qui apportent la rapidité de traitement et le faible coût, mais au détriment de la souplesse d'adaptation des algorithmes.

L'électronique représente un domaine très dynamique riche en nouvelles découvertes et applications. Un domaine qui se caractérise par la tendance à la miniaturisation et la simplification. Des systèmes compacts, légers, économiques, ergonomiques et fiables.

Les FPGAs (Field Programmable Gate Array) sont les circuits qui ont permis de pousser plus loin le progrès technologique en électronique. En effet, le degré de maturité des FPGAs fait d'eux aujourd'hui une solution pour remplacer les ASICs (circuits intégrés à application spécifique) et les processeurs personnalisés dans des applications de traitement d'image.

Réduire les couts, augmenter les performances et la fiabilité d'un circuit, diminué les délais de mise en place d'un prototype opérationnel, maintenabilité et réduire l'encombrement des composants sur circuit, sont des atouts conjugués dans un FPGA. Ces atouts justifient le succès de ces composants dans tous les domaines d'applications de l'électronique, l'aéronautique, l'aérospatial, l'industrie, les télécommunications, les appareilles médicaux ….etc, tous les domaines trouvent le bon compromis en intégrant les FPGAs.

Les contours actifs présentent une solution intéressante comme approche dans des différents domaines d'application pour le suivi d'objet, la segmentation et la détection de contour ; la base du processus des contours actifs est de faire évoluer une courbe initiale vers un objet sous l'influence des forces internes et externes, plusieurs recherches ont étaient réaliser pour améliorer l'exactitude et la vitesse de la première approche du snake.

Cependant aucune implémentation hardware n'a été achevée dans les documentations.

Ce travail de mémoire a pour objectif l'implémentation de l'algorithme de détection de contour de canny en temps réel et la proposition d'une solution architectural fiable en temps réel d'un algorithme de contour actif (SNAKE) avec conception dans l'environnement de développement ISE et simulation sous ModelSim.

La démarche scientifique adoptée dans ce mémoire suit cette structuration :

- Le premier chapitre est consacré à l'état d'art des circuits reconfigurable FPGA et on traitera l'architecture interne de ces derniers.
- Le second chapitre présente brièvement quelque algorithme de détection de contour.
- Le troisième chapitre concerne la conception et l'implémentation du détecteur de contour par l'approche de canny à faible couts en temps réelle et la présentation des résultats obtenue lors de la simulation.
- Le quatrième chapitre détaille l'architecture proposé pour la conception et l'implémentation de l'algorithme de contour actif (SNAKE) et la présentation des différents modules décrit en langage de description matériel VHDL ou en schématique.
- Le cinquième et dernier chapitre est consacré pour la validation et la présentation des résultats obtenues lors de la simulation sous ModelSim.

Enfin nous clôturons ce document avec une conclusion générale ainsi que des perspectives d'application et d'exploitation de la technologie programmable FPGA qui est prometteuse et en pleine maturité.

1.1.Introduction

L'exécution d'un algorithme implanté sur circuit est beaucoup plus rapide que l'exécution séquentielle de son programme par un processeur (même fonctionnement à une fréquence beaucoup plus élevée).

La détection de contour, but de notre projet, requiert :

- D'une part une grande puissance de calcul (comme tous les traitements d'image), d'où la nécessité de son implantation.

- D'autre part une flexibilité durant sa phase de développement d'où son implémentation sur circuit programmable de type FPGA pour un prototypage rapide.

Ce chapitre est donc dédié à l'étude des principales propriétés des circuits FPGA, afin d'optimisé l'exploitation pour notre projet.

Cette partie, défini les circuits FPGA dans la section 1.2, présente ainsi dans la section 1.2 les divers domaines d'application des FPGA actuels. L'architecture de ce type de circuits est donnée section 1.3, en détaillant plus particulièrement les modèles de FPGA ciblés par les travaux présentés dans ce document. La section 1.4 décrit ensuite le mode de programmation des FPGA, du langage utilisé aux divers outils impliqués dans la chaîne de conception. Enfin, la section 1.5 discute des questions de portabilité entre les divers modèles de FPGA et argumente les choix que nous avons faits concernant notre projet.

1.2.FPGA

Un FPGA (pour Field-Programmable Gate Array) est un circuit intégré composé d'un grand nombre d'éléments logiques programmables reliés entre eux grâce à une matrice de routage elle aussi programmable. Cette structure permet au FPGA d'émuler n'importe quel circuit, à la seule condition que celui-ci ne soit pas trop gros pour ne pas épuiser les ressources logiques et de routage du FPGA.

Cependant, cette souplesse de programmabilité se paye au niveau des performances du circuit.

Bien que les FPGA soient gravés avec la même technologie que les processeurs généralistes cadencés à quelques gigahertz, la fréquence d'horloge des FPGA ne dépasse pas des centaines de mégahertz pour les modèles les plus récents. Néanmoins, la grande

liberté architecturale permet de profiter au mieux du parallélisme à grain fin des circuits implémentés, ce qui compense largement les relativement faibles fréquences atteignables.

1.3.Domaines d'application

Les cellules du FPGA étant indépendantes les unes des autres, elles peuvent très bien effectuer leurs calculs en parallèle. Le FPGA permet ainsi d'optimiser très finement des applications en jouant sur leur parallélisme. On retrouve par conséquent l'utilisation de ce type de processeur dans de nombreux domaines d'applications très calculatoires, comme entre autres le traitement du signal [1][2][3], la cryptographie [Beu03, BMP+06, MB06], la bio-informatique [4][5][6] ou encore plus généralement l'accélération de calculs scientifiques [7][8][9][10].

Une autre partie de leur domaine d'application est due à leur faible coût. En effet, fondre un circuit intégré a application spécifique(ou ASIC, pour Application-Specific Integrated Circuit) demande un investissement financier initial très important, qui couvre de l'achat des machines à la réalisation des masques de gravure du circuit. Même si le coût unitaire de production de chaque circuit est ensuite très faible, l'investissement initial devient rapidement exorbitant pour des circuits spécialisés qui ne seront tirés qu'à quelques milliers d'exemplaires. Les FPGA sont aussi des ASIC, mais leur tirage est beaucoup plus important, et donc leur coût bien plus faible. De nombreuses applications demandant un circuit spécifique choisissent ainsi la solution du FPGA comme alternative à l'ASIC pour réaliser ce circuit.

Toujours à cause de leur coût réduit ainsi que de leur souplesse de programmation, les FPGA sont aussi largement utilisés dans le prototypage rapide. Par mesure de précaution, avant d'entamer la gravure d'un circuit intégré, celui-ci est toujours soigneusement testé par simulation de manière à garantir de correction. L'utilisation de circuits FPGA permet ainsi d'accélérer grandement cette simulation par rapport à une simulation logicielle.

Enfin certains modèles de FPGA sont reconfigurables. Si certains FPGA, comme ceux conçus par Actel par exemple, sont programmés de manière irréversible grâce à des anti-fusibles [11], d'autres reposent sur une technologie SRAM (pour Static Random Access Memory), comme ceux proposés par Xilinx [12] ou Altera [13]. Cette technologie permet ainsi de reconfigurer le FPGA autant de fois que nécessaire. Cela est donc utile lorsqu'une erreur est découverte dans un circuit car il suffit alors de reprogrammer le FPGA avec une version corrigée du circuit.

Cette reconfigurabilité ouvre aussi un autre champ d'application. Le FPGA, bénéficiant ainsi d'un lien privilégié avec le processeur, peut-être reconfiguré à la volée selon les besoins de l'application en cours d'exécution et sert donc d'accélérateur de calculs dédié.

1.4.Architecture

1.4.1. Vue d'ensemble

Les cellules logiques d'un FPGA (aussi appelées PE, pour Processing Elements) sont regroupées en îlots, eux-mêmes répartis régulièrement suivant une grille rectangulaire. Chaque îlot est composé généralement de quatre, huit ou seize cellules logiques, ainsi que d'un routeur (switch box) programmable réalisant la connexion entre les entrées et sorties des cellules logiques et la matrice de routage.

Cette matrice de routage couvre elle aussi toute la surface du FPGA, sous forme de pistes horizontales et verticales. Certaines de ces pistes relient des îlots voisins, alors que d'autres traversent tout le FPGA pour permettre de relier deux îlots distants l'un de l'autre. Les switch boxes situées au niveau de chaque îlot réalisent les connexions nécessaires entre ces différents fils.

De plus, généralement, des blocs de mémoire dédiés (RAM blocks) sont répartis sur la surface du FPGA. Connectés à la matrice de routage, ces blocs offrent un espace mémoire confortable utilisable par la logique programmable voisine.

Enfin, pour permettre au FPGA de communiquer avec le monde extérieur, des cellules d'entrée/sortie (I/O blocks) sont disposées sur tout le pourtour du circuit et elles aussi reliées à la matrice de routage.

La figure 1.1 présente sur la gauche une vue simplifiée de la surface d'un FPGA type, sur lequel on peut distinguer les cellules d'entrée/sortie, les blocs de mémoire, ainsi que les îlots et la matrice de routage. Un de ces îlots est représenté plus en détails sur la droite.

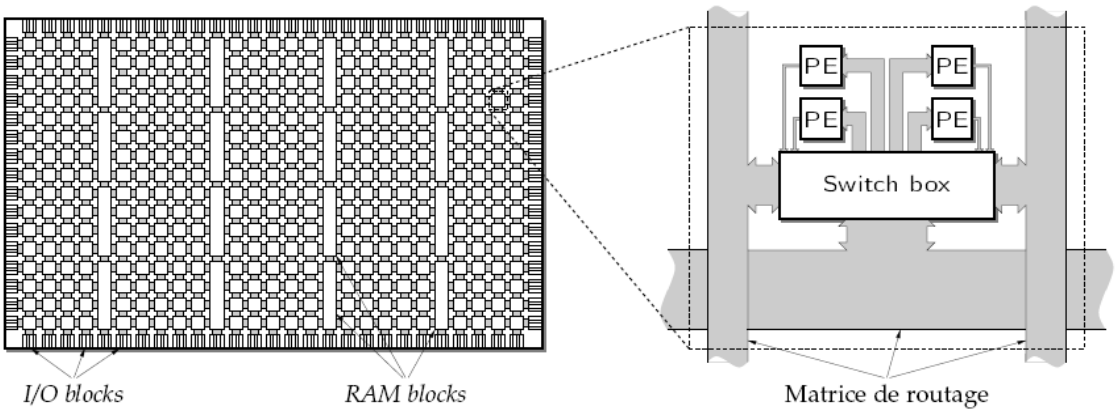


Figure 1.1 Architecture générale d'un FPGA et détail d'un îlot.

Le modèle de FPGA que nous utilisons comme cible de tous les travaux présentés dans ce document est un Virtex4 XC4VLX160 (speed grade -12), produit par Xilinx . Selon la terminologie propre à ce constructeur, les îlots sont appelés CLB (pour Configurable Logic Blocks).

Chaque CLB est composé de quatre slices répartis sur deux colonnes, chaque slice étant composé à son tour de deux cellules logiques disposées en colonne.

La figure 1.2 présente ainsi l'organisation des quatre slices composant un CLB de Virtex4, ainsi que les deux cellules logiques de chacun de ces slices. Les deux chaînes de propagation de retenue dédiées sont aussi représentées.

1.4.2. Cellules logiques

Les cellules logiques d'un FPGA consistent en général simplement en une partie de traitement et d'une partie de mémorisation. La partie traitement est assurée par une table (Look-Up Table, ou LUT) avec n bits d'adresse et 1 bit de sortie. Quant à la partie de mémorisation, c'est un registre contrôlé par un signal horloge, noté généralement clk.

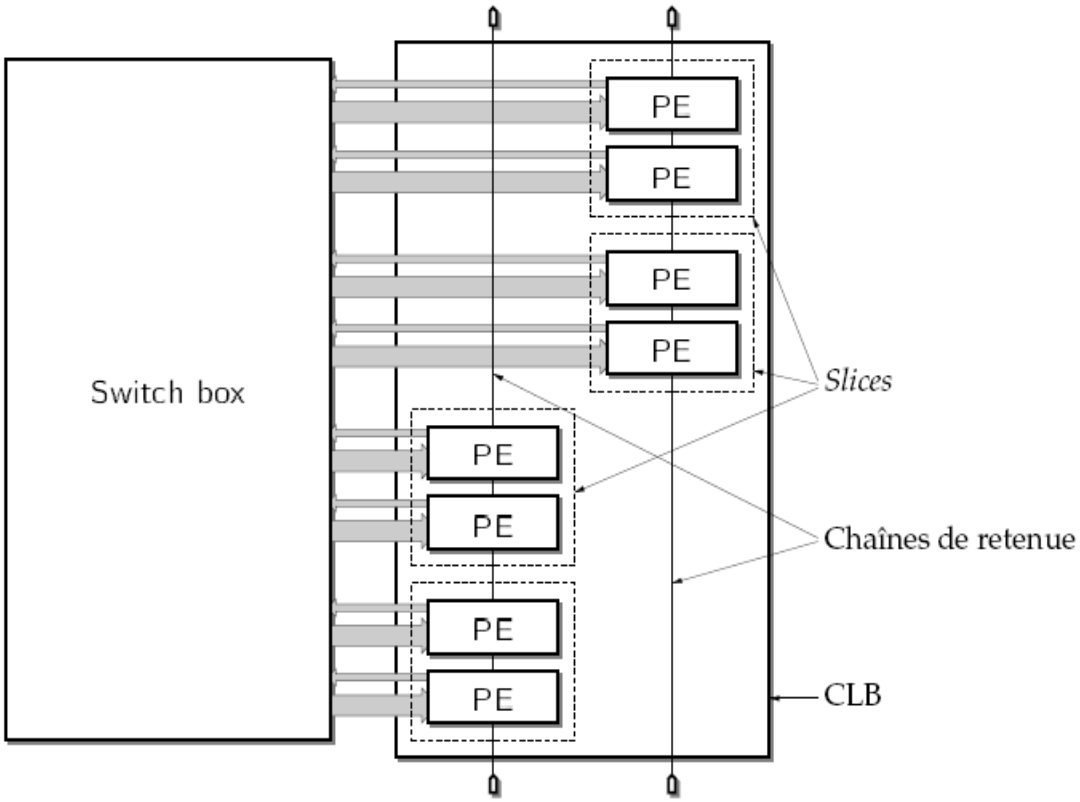


Figure 1.2 Structure d'un CLB sur Virtex4.

La table est donc une petite mémoire de 2^n mots de 1 bit, adressée par n bits provenant de la matrice de routage par l'intermédiaire de la switch box. Une LUT permet donc d'implémenter n'importe quelle fonction booléenne ayant jusqu'à n variables, grâce à la table de vérité de cette fonction. Le résultat de cette table peut soit être réinjecté directement dans la matrice de routage pour ensuite être propagé vers une autre cellule logique du FPGA, soit être mémorisé par le registre sur un front montant de l'horloge avant d'être redirigé par la switch box.

Sur la plupart des FPGA actuels, n est fixé à 4 bits, les cellules logiques que l'on trouve sur les modèles de FPGA récents sont plus complexes que les cellules présentées dans la figure 1.3. Cependant, elles partagent toutes cette structure basée autour d'une LUT et d'un registre.

La figure 1.3 représente un slice de Virtex4 et ses deux cellules logiques.

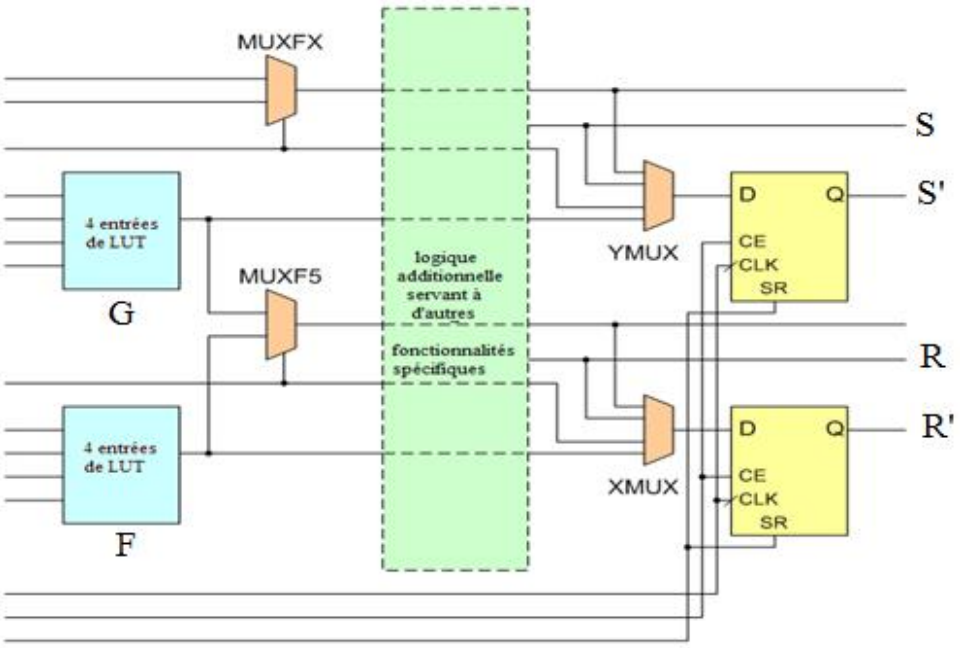


Figure 1.3 diagramme simplifié d'un slice de Xilinx virtex-4

La cellule du bas implémente la fonction booléenne $F(X_1, X_2, X_3, X_4)$et celle du haut la fonction $G(Y_1, Y_2, Y_3, Y_4)$. On peut donc observer les deux LUT sur la gauche du slice, ainsi que les deux registres correspondants sur la droite. Les sorties directes des deux tables sont respectivement R et S, et les signaux R' et S' représentent les sorties des registres.

1.4.3. Multiplexeurs

Comme nous l'avons vu, si les LUT d'un FPGA sont adressées sur n bits, il est alors possible d'implémenter des fonctions booléennes ayant jusqu'à n variables. L'implémentation d'une fonction à n+1 variables requiert alors un multiplexeur supplémentaire d'après le théorème d'expansion de Shannon :

$$F(X_1, \ldots\ldots, X_n, X_{n+1}) = \begin{cases} F_0(X_1, \ldots\ldots, X_n) & si\ X_{n+1} = 0, \\ F_1(X_1, \ldots\ldots, X_n) & sinon \end{cases}$$

avec les fonctions F_0 et F_1 implémentées sur une LUT chacune :

$$F_0(X_1, \ldots\ldots, X_n) = F(X_1, \ldots\ldots, X_n, 0), et$$
$$F_1(X_1, \ldots\ldots, X_n) = F(X_1, \ldots\ldots, X_n, 1).$$

En considérant la structure des cellules logiques présentée plus haut, le multiplexeur final nécessite l'utilisation d'une troisième LUT, chargée de sélectionner le résultat entre $F_0(X_1, \ldots\ldots, X_n)$et $F_1(X_1, \ldots\ldots, X_n)$selon la valeur de X_{n+1}.

Pour économiser cette dernière LUT, certains modèles de FPGA intègrent des multiplexeurs dédiés, permettant de sélectionner une des sorties de deux LUT voisines en

fonction de la valeur d'un signal spécifique. Il est même possible dans certains cas de recommencer l'opération récursivement sur plusieurs étages de multiplexeurs.

1.4.4. Additionneurs rapides

La structure des cellules logiques vue précédemment ne permet pas d'implémenter efficacement un additionneur, car les LUT n'ont qu'un seul bit en sortie, alors que l'addition de deux bits génère non seulement un bit de somme mais aussi un bit de retenue. De plus, la propagation de la retenue (ou bien des signaux generate et propagate dans le cas d'un additionneur préfixe) serait très lente si elle devait se faire par la matrice de routage.

Pour ces raisons, du matériel supplémentaire est dédié sur les FPGA actuels pour optimiser le calcul des additions et soustractions. Certains proposent donc de la logique additionnelle dans chaque PE permettant le calcul de la somme et de la retenue ainsi que des ressources de routage dédiées pour une propagation rapide de cette retenue. C'est par exemple le cas de toute la gamme des Virtex et Spartan de Xilinx. On peut aussi trouver d'autres types d'additionneurs, comme par exemple sur le Stratix d'Altera qui permet d'implémenter des additionneurs à sélection de retenue (carry-select adders).

1.4.5. Blocs de mémoire

Comme mentionné précédemment, les FPGA embarquent aussi des blocs de mémoire de quelques kilobits. Ces blocs sont reliés à la matrice de routage et peuvent donc être accédés par n'importe quelle partie du circuit. Sur les FPGA actuels, les blocs de mémoire ont généralement deux ports (dual-port memory), permettant deux accès concurrents. Les deux ports supportent même la plupart du temps des tailles de mots différentes.

Ainsi, les FPGA possèdent un certain nombre de RAM blocks répartis en colonnes sur la surface du circuit. Le Virtex4 utilisé dans le cadre des travaux présentés ici intègre ainsi 288 RAM blocks de 18 Kb.

1.4.6. Fonctionnalités supplémentaires

Les modèles de FPGA récents offrent un grand nombre d'autres fonctionnalités, parmi lesquelles :

1.4.6.1. Arbres de distribution d'horloge

Devant le nombre de cellules logiques et donc de registres embarqués dans les FPGA, le signal d'horloge contrôlant ces registres doit être capable de supporter une sortance (fanout) très importante tout en assurant une parfaite synchronisation sur tout le circuit. Cela est clairement impossible à réaliser en utilisant la matrice de routage.

Les FPGA possèdent donc des arbres de routage dédiés à la distribution de ce signal d'horloge. On trouve généralement plusieurs arbres de distribution sur un seul FPGA, pour permettre d'avoir différents domaines d'horloge sur un même circuit. Le Virtex4 utilisé supporte ainsi jusqu'à 12 DCM.

1.4.6.2. Petits multiplieurs

Pour permettre l'implémentation efficace de circuits très calculatoires et massivement parallèles, comme on en trouve en traitement du signal ou en traitement d'images, les générations actuelles de FPGA embarquent toutes de petits multiplieurs entiers. Ces multiplieurs, accessibles par la matrice de routage, permettent donc d'accélérer les calculs, mais surtout de réduire grandement le nombre de cellules logiques utilisées par le circuit. La famille des Virtex4 de Xilinx propose ainsi des multiplieurs 18×18 bits. Ces multiplieurs partagent les mêmes ressources de routage que les RAM blocks de 18 Kb : leur utilisation est donc mutuellement exclusive. Le Virtex4 possède pour sa part 96 multiplieurs.

1.4.6.3. Blocs DSP

Les générations les plus récentes de FPGA proposent même des blocs DSP (pour Digital Signal Processing) dédiés, comme leur nom l'indique, au traitement du signal. Ces blocs sont généralement composé d'un multiplieur et d'un accumulateur (MAC, pour Multiplier and ACcumulator et permettent ainsi de réaliser des filtres de manière très efficace. On trouve ce genre de blocs DSP chez les FPGA Stratix et Stratix II d'Altera, ainsi que dans les Virtex-4 et Virtex-5 de Xilinx. Par contre, les Virtex-II n'en possèdent pas.

1.4.6.4. Cœurs de processeurs RISC

Enfin, devant l'utilisation croissante des FPGA pour l'implémentation de systèmes embarqués comme les SoC ou les NoC (pour System-on-Chip et Networkon-Chip respectivement), certains modèles de FPGA intègrent directement un ou plusieurs coeurs complets de processeurs RISC.Ainsi, les Virtex-II Pro et Virtex-4 de Xilinx embarquent jusqu'à quatre coeurs de PowerPC 405.

Pour les FPGA qui sont dépourvus de tels processeurs, tels les Virtex-II, les constructeurs proposent des solutions softcores, comme le MicroBlaze de Xilinx ou le NIOS II d'Altera, véritables processeurs implémentés dans la matrice de cellules logiques.

1.5.Programmation et méthodologie de conception

Xilinx fournit dans son pack d'outils différents soft permettant la création de systèmes embarqués sur puce, parmi ces softs on dénombre ISE (Integrated Software Environment) et EDK (Embedded Development Kit), tous deux nous offrent la possibilité de programmer des FPGA suivant l'application ciblée.

Le flot de conception standard est composé de plusieurs étapes :

- Conception et synthèse du design.
- l'implémentation et la vérification du design.

Une vue générale du flot de conception est présentée par la figure 1.4.

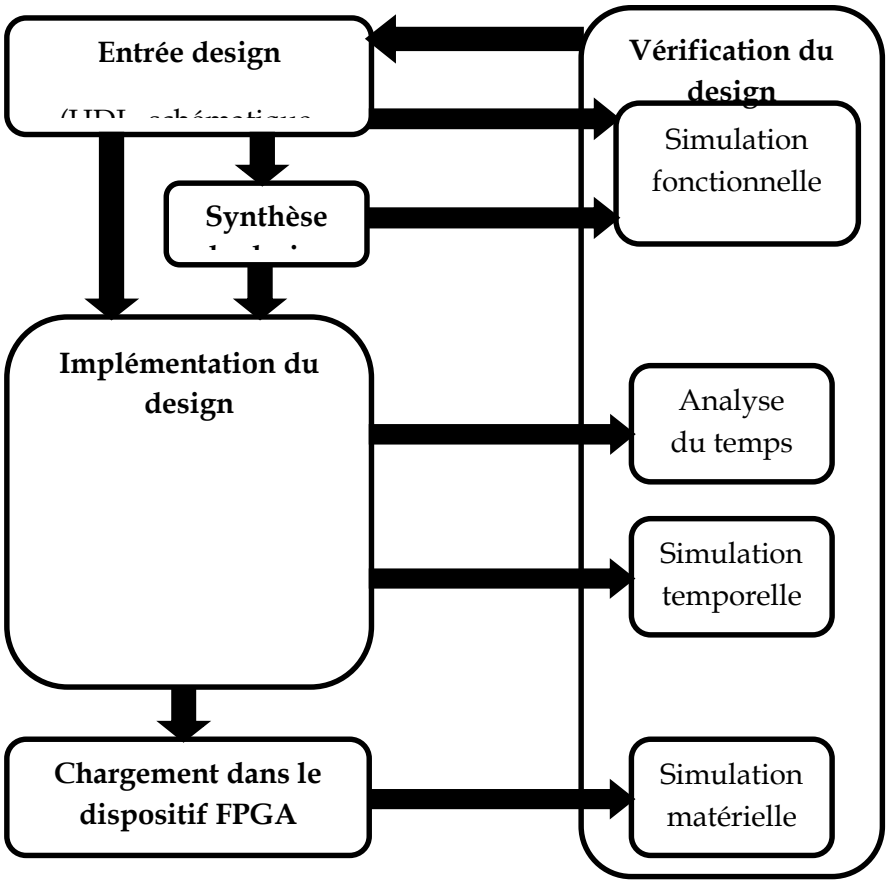


Figure 1.4 Vue générale d'un flot de conception

1.5.1. Description des circuits

Les circuits présentés dans ce document sont des implémentations directes d'algorithmes. Cependant, si ces algorithmes peuvent facilement être décrits grâce à n'importe quel langage de programmation impératif classique, cela n'est pas du tout le cas des circuits. En effet, l'implémentation matérielle d'un algorithme fait appel à des notions temporelles très fortes, comme la concurrence et la synchronisation, qui ne sont pas naturelles dans les langages classiques. Il faut donc pour réaliser un circuit recourir à un langage de description de matériel ou HDL (Hardware Description Langage), qui sera capable d'exprimer explicitement ces notions temporelles, ou bien par un design schématique (dans notre projet on a utilisé les deux).

Il existe de nombreux HDL, parmi lesquels on trouve VHDL, Verilog et System C pour les plus connus et utilisés. Si System C est plus orienté vers la modélisation de systèmes complexes composés de plusieurs processus concurrents, VHDL et Verilog sont quant à eux bien adaptés à la description d'opérateurs arithmétiques comme ceux présentés dans ce document, ainsi qu'à leur intégration dans des circuits plus importants. Nous avons ainsi choisi pour nos opérateurs d'utiliser le langage VHDL.

Le VHDL, ou Very-High-Speed Integrated Circuit Hardware Description Language, était à l'origine destiné à documenter de manière formelle la structure et le comportement de circuits intégrés. Ce n'est que par la suite que furent développés d'abord des simulateurs logiciels, puis enfin des synthétiseurs pour ce langage. VHDL fut décrit par le standard IEEE 1076 en 1987, puis révisé successivement en 1993 et 2000. Il est désormais l'un des HDL les plus utilisés, tant par l'industrie que par le monde académique.

En VHDL, l'entité de base est le composant (aussi appelé entité). Un composant est représenté par son interface extérieure, qui définit ses ports d'entrée et de sortie, ainsi que son architecture interne.

La figure 1.5 illustre la partie entité dans une description VHDL, et la figure 1.6 illustre la représentation de l'entité d'un composant.

```
29
30 entity compare is
31     Port ( clk_m : in std_logic;
32            i : in  STD_LOGIC_VECTOR (7 downto 0);
33            dout : in  STD_LOGIC_VECTOR (7 downto 0);
34            we : in std_logic;
35            comp : out  STD_LOGIC);
36 end compare;
37
```

Figure 1.5 Description VHDL de l'entité

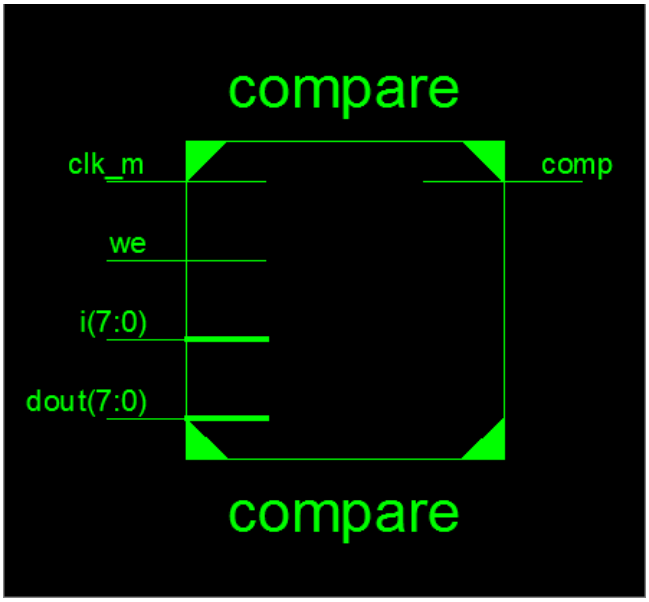


Figure 1.6 Représentation de l'entité d'un composant

L'architecture d'un composant définit à son tour les sous-composants qui la constituent et les fils qui les relient. Un circuit est alors décrit comme un composant, lui-même constitué d'autres composants, et ainsi de suite jusqu'à arriver aux composants logiques de base.

La figure 1.7 illustre la description VHDL d'une architecture, et la figure 1.8 illustre l'architecture interne d'un composant.

```
37
38  architecture Behavioral of compare is
39  signal s_stop : std_logic:='0';
40  signal cnt:integer:=0;
41  begin
42          -- comparaison de la ligne avec les données de la ram
43  process (clk_m,we)
44  begin
45  if (we='0') then
46     if (clk_m'event and clk_m='1') then
47        if(cnt/=768) then
48        cnt<=cnt+1;
49           if (i=dout) then
50           s_stop<='1';
51           else
52           s_stop<=s_stop;
53           end if;
54        else
55        cnt<=0;
56        s_stop<='0';
57        end if;
58     end if;
59  else
60  s_stop<='0';
61  end if;
62  end process;
63  comp<=s_stop;
64  end Behavioral;
```

Figure 1.7 Description VHDL d'une architecture

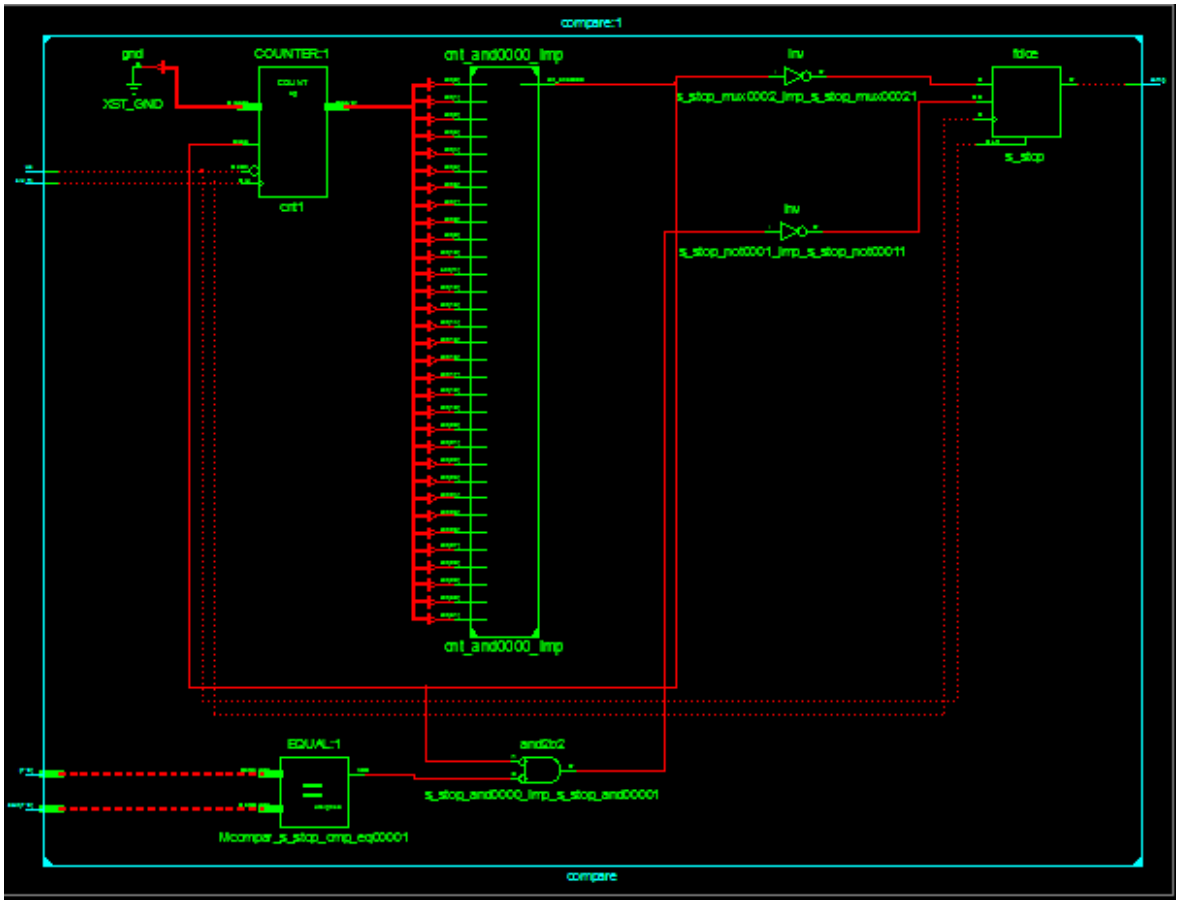


Figure 1.8 Architecture interne d'un composant

Cette représentation hiérarchique permet d'ajuster le niveau de détails nécessaires à la description d'un circuit. Pour la simulation d'un système complet, on peut ainsi se contenter d'une description de haut niveau, où les composants sont décrits par leur comportement, c'est-à-dire leur réponse aux stimulations extérieures. De l'autre côté du spectre, lorsque l'on souhaite représenter un circuit synthétisable, on peut détailler bien plus la description, en descendant jusqu'aux portes logiques. Cette description structurelle permet ainsi d'avoir un meilleur contrôle sur l'implémentation finale du circuit, et donc sur d'éventuelles optimisations de ce circuit. Il convient cependant d'éviter d'optimiser trop un circuit au niveau de sa description en VHDL sous peine de perdre en portabilité.

1.5.2. Implémentation

La description VHDL d'un circuit subit toute une suite de transformations avant que ce circuit soit effectivement configuré sur un FPGA.

1.5.2.1. Synthèse

L'étape de synthèse consiste à lire la description VHDL du circuit pour en extraire toute la structure logique. Lorsque des composants de ce circuit sont décrits de manière comportementale, c'est le rôle du synthétiseur d'inférer la logique nécessaire à la réalisation des comportements spécifiés. Cette tâche est d'ailleurs possiblement très difficile et donc parfois mal supportée.

Un autre rôle du synthétiseur est d'appliquer diverses optimisations logiques au circuit. Il cherche ainsi à minimiser toutes les expressions logiques utilisées par ce circuit.

Enfin, le synthétiseur produit une netlist, c'est-à-dire une liste de fils et de portes logiques.

1.5.2.2. Mapping

La netlist du circuit est ensuite adaptée aux primitives logiques du FPGA cible. Cette étape de ciblage technologique permet ainsi de passer d'une description structurelle générique à une description spécialisée pour une architecture précise. Les éléments de base ne sont alors plus de simples portes logiques mais des cellules programmables du FPGA.

1.5.2.3. Placement et routage

La phase de mapping ayant identifié les ressources logiques nécessaires à l'implémentation d'un circuit, la tâche de placement/routage consiste alors à disposer et

connecter ces ressources sur la surface du FPGA. Il est possible de donner au placeur/routeur des contraintes spatiales ou temporelles, pour fixer par exemple la position des ports d'entrée/sortie du circuit ou bien une période d'horloge maximale.

À l'issue de l'étape de placement/routage, le circuit obtenu est tel qu'il sera programmé sur le FPGA. C'est sur ce modèle que sont calculées les estimations de surface et de latence du circuit.

1.5.2.4. Configuration

Enfin, la dernière étape avant la programmation effective du FPGA consiste à générer un bitstream. Ce bitstream est en fait un fichier contenant tous les bits de configuration du FPGA, construit pour correspondre parfaitement au circuit décrit lors du placement/routage.

Une fois le bitstream créé, il ne reste plus qu'à le télécharger sur le FPGA grâce à une interface dédiée. Le FPGA restera ainsi configuré tant qu'il sera sous tension, ou bien lorsqu'un autre circuit y sera programmé suivant le même procédé./

Nous avons utilisé pour les travaux présentés ici la version 12.3 d'ISE, l'environnement de développement fourni par Xilinx. Cette suite logicielle inclut notamment le synthétiseur XST, ainsi que tous les outils évoqués précédemment.

1.5.2.5. Simulation logicielle

La simulation logicielle d'un circuit est une étape essentielle de sa conception. Il faut en effet s'assurer que celui-ci fonctionne correctement avant de le configurer sur le FPGA. La simulation permet ainsi de fournir au circuit des vecteurs de test en entrée, puis de vérifier que le circuit se comporte de la manière prévue. Dans le cas des opérateurs arithmétiques présentés dans ce document, cela revient donc juste à vérifier la correction des résultats calculés.

Bien entendu, la simulation ne garantit pas la correction du circuit. Mis à part pour de petits opérateurs, il est en effet impossible de réaliser une simulation exhaustive à cause de la lenteur de la simulation logicielle. Il faut donc essayer de choisir des vecteurs de test adaptés à l'opérateur pour couvrir le mieux possible les différents cas de figure.

Le simulateur logiciel utilisé pour nos opérateurs est l'édition gratuite de ModelSim Xilinx

Edition III, version 6.3f, développée par ModelTech et munie des bibliothèques de composants de Xilinx.

1.5.2.6. Simulation matérielle

Comme mentionné plus haut, la simulation logicielle est bien trop lente pour pouvoir réaliser des tests exhaustifs. Il est toutefois envisageable, pour certains opérateurs, d'effectuer ces tests en utilisant une carte de développement pour FPGA, munie de ports d'entrée/sortie rapides.

1.6.Portabilité

Comme nous avons pu le voir, de nombreux modèles de FPGA existent. Si ces modèles partagent les caractéristiques principales communes à tous les FPGA (cellules logiques et matrice de routage configurables, blocs de mémoire), les caractéristiques mêmes de ces éléments de base diffèrent grandement d'un modèle à un autre. De même, la présence ou non de petits multiplieurs câblés ou bien de blocs DSP est encore un facteur supplémentaire pouvant entraver la portabilité d'un circuit.

Pour tenter de rendre nos travaux utilisables sur le plus grand nombre d'architectures de FPGA possibles, nous avons donc dû nous affranchir des spécificités propres à uniquement certains modèles. Par conséquent, les opérateurs présentés dans ce document ne font aucune hypothèse sur le FPGA cible sur lequel ils vont être implémentés.

Ainsi, par exemple, si un opérateur requiert une multiplication entière, celle-ci sera simplement exprimée en VHDL par l'opérateur * : cette abstraction de l'architecture sous-jacente du multiplieur laisse le soin au synthétiseur de choisir la manière la plus efficace pour réaliser une multiplication sur le FPGA cible. Cela signifie donc que la portabilité ne peut être atteinte qu'en retardant le plus loin possible la spécialisation technologique du circuit, déléguant ainsi le plus gros de cette tâche au synthétiseur.

1.7.Conclusion

Dans ce chapitre nous avons étudié l'état d'art des FPGA ce qui nous a permis de conclure que la technologie FPGA s'inscrit au sommet de l'évolution des composants logiques, elle répond au besoin croissant de composants plus performants et plus économiques. Les FPGA ouvrent de grandes perspectives en matière d'application de traitement d'image. L'implémentation d'un algorithme de traitement d'image sur FPGA nécessite aussi une bonne maitrise des outils fournis par la théorie, ainsi qu'une bonne maitrise dans l'électronique numérique lors de la phase d'implantation. L'ensemble de ces points seront traité au cours des prochains chapitres. il sera plus particulièrement détaillé

et appliqué l'implémentation de deux algorithmes de détection de contour sur le virtex 4 de Xilinx.

2.1.Introduction

La détection de contour dans une image est un problème souvent posé avant toute phase de mesure ou d'interprétation.

Deux approches sont envisageables : l'approche contour ou l'approche région.

Le présent chapitre vise à introduire les concepts de base de la détection de contour en commençant par décrire un contour, ces différents types ainsi définir les différents approches utilisées pour la détection de contour.

2.2.La détection de contour

L'analyse d'image englobe l'ensemble des outils permettant la présentation de la scène perçue par le dispositif d'acquisition sous forme d'un signal 2D ou 3D ; la modélisation et le stockage de cette image sous forme numérique, le traitement, la compréhension et l'interprétation du contenue de l'image.

L'analyse d'image est née de la nécessité de remplacer ou de compléter l'observateur humain par la machine. A partir d'une image, il s'agit d'extraire les informations dont on a besoin, de les quantifier, de les traiter et de les interpréter.

La détection de contour est une étape préliminaire à de nombreuses applications de l'analyse d'images. Les contours constituent en effet des indices riches, pour toute interprétation ultérieure de l'image. Les contours dans une image proviennent des :

- discontinuités de la fonction de réflectance (texture, ombre).
- discontinuités de profondeur (bords de l'objet).

2.3.Extraction de contours

En général, les variations d'intensité dans une image correspondent à des zones pertinentes. Ces informations sont très importantes pour les opérations subséquentes à cette première phase.

Pour extraire de l'information de plus haut niveau, on doit d'abord avoir des informations de bas niveau (les contours).

Ces informations correspondent à des frontières de régions homogènes dans l'image.

2.3.1. Les types de Contours

2.3.1.1. Contours de type marche

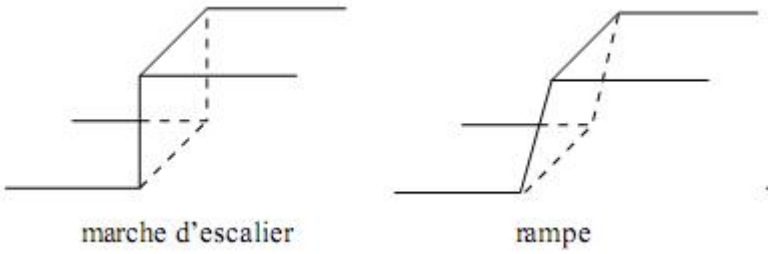


Figure 2.1 Contours de type marche

Ces contours existent généralement entre deux régions de niveaux de gris presque constants et différents; par exemple ce type de contours se produit lors du recouvrement d'un objet par un autre ou lors de la production d'ombre sur une surface. Nous pouvons caractériser ce contour par son amplitude (A) et par sa pente(p) comme illustrée dans la Figure 2.2

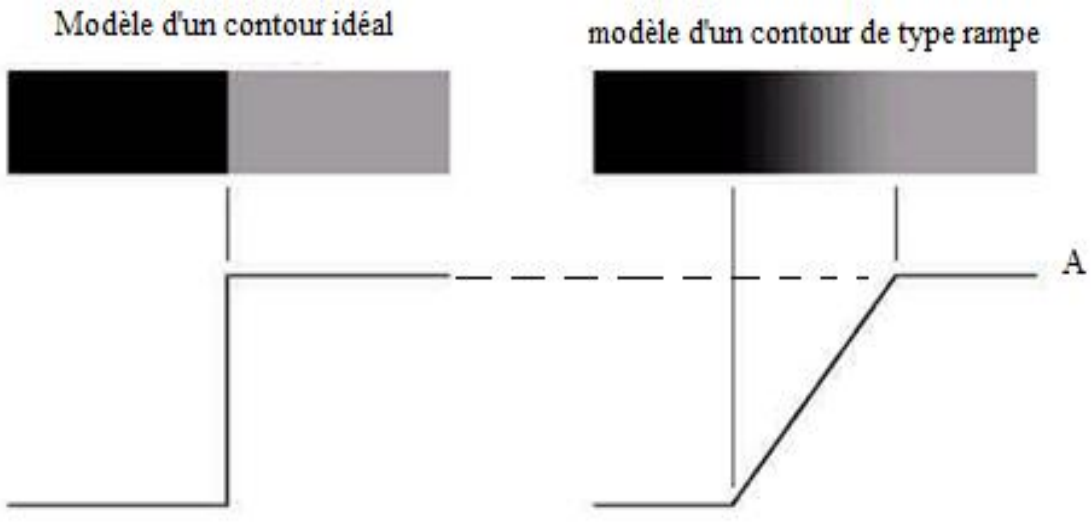


Figure 2.2 Contours de type marche caractérisé par son amplitude et sa pente

2.3.1.2. Contours de type crête

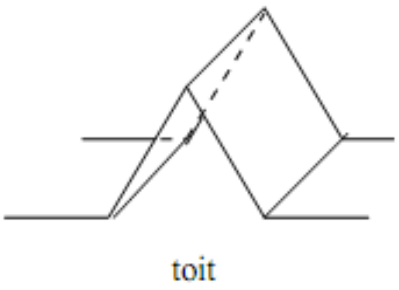


Figure 2.3 Contour de type toit

On distingue deux classes de contours :

- Les contours « toits »
- Les contours « arrêtes »

Nous trouvons ce type de contours dans les images contenant des lignes de faible ou forte intensité, telles que les images de caractères ou d'empreintes digitales. Le profil de la fonction des niveaux de gris perpendiculairement à la ligne de crête a la forme d'une cloche ou d'une vallée.

2.3.2. Les approches de détection de contour

2.3.2.1. Les approches classiques

Cette section présente un ensemble de méthodes qui ont eu historiquement une grande importance en traitement des images. Bien que certaines soient encore régulièrement employées parmi eux :

2.3.2.1.1. Les détecteurs de gradient par filtrage

Ces détecteurs reposent tous sur une recherche d'un extremum de la dérivée première ou d'un passage par zéro d'une dérivée seconde, celle-ci étant calculée de diverses manières, mais généralement par un filtrage passe-haut précédé d'un léger filtrage passe-bas pour s'affranchir des bruits.

2.3.2.1.2. Les détecteurs de gradient par masques

A côté de ces approches très inspirés du traitement du signal, des filtres de dérivation plus empiriques ont été proposées à partir d'estimateurs locaux de l'image ou de ses dérivées.

Ces estimées sont obtenues à l'aide de masques présentés dans la figure 2.4 appliqués sur des fenêtres de pixels2x2 ou 3x3 ou des fenêtres plus grandes en cas d'images très bruitées. La somme des coefficients de ces filtres est nulle (fonction de transfert nulle à la fréquence 0), et les coefficients sont antisymétriques.

Les filtres les plus utilisés sont :

Sobel, Roberts, Gradient, Prewitt [14][15][16]

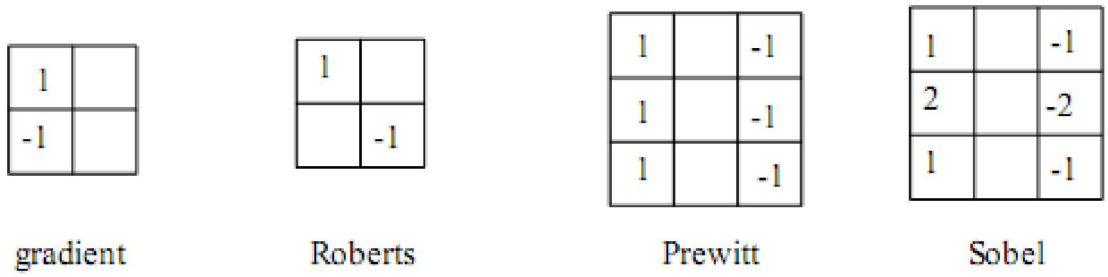


Figure 2.4 Différents masques utilisés

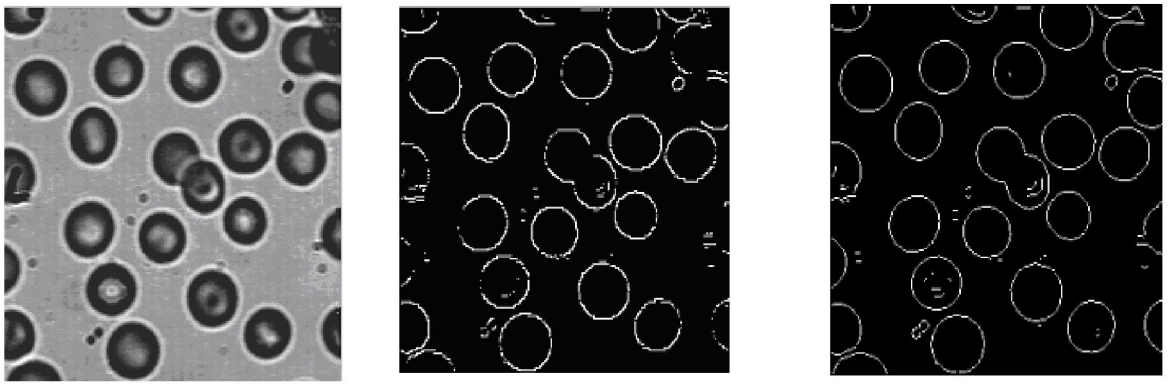


Image original operateur robert operateur sobel

Figure 2.5 Image filtré

2.3.2.1.3. Opérateur Laplacien

Cette méthode est basé sur le fait que le passage par zéro du Laplacient permet de bien mettre en évidence les extrémums de la dérivée, le passage par zéro du Laplacient correspond en effet bien au maximum du gradient dans la direction du gradient. Cette méthode tire en outre profit du fait que les zéros de la dérivée seconde constituent un réseau de lignes fermées.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 2.6 Le masque (noyaux) de Laplacien

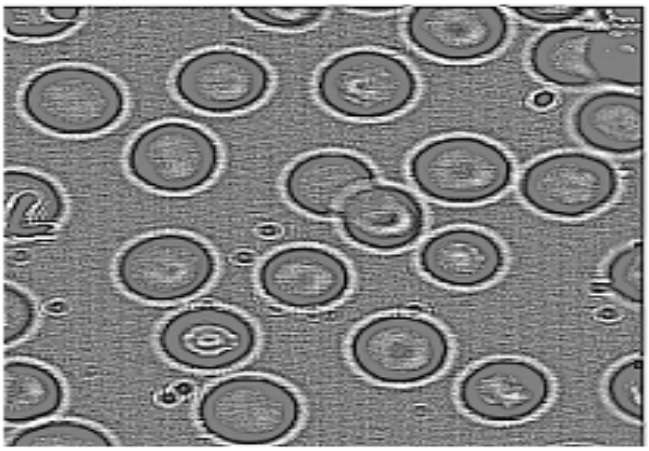


Figure 2.7 Détection de contour par Laplacien

Le Laplacien possède cependant un inconvénient majeur qui est sa grande sensibilité au bruit. En effet cet opérateur réalise une dérivée seconde de l'image et est donc très instable.

2.3.2.2. Les approches analytiques

Nous allons voir maintenant une approche qui a permis une bien meilleure compréhension des conditions d'une bonne détection de contours et qui a ainsi conduit à des détecteurs de très bonne qualité.

2.3.2.2.1. Les critères de Canny

Canny a proposé [17] un filtre déterminé analytiquement à partir de trois critères :

- ✓ Bonne détection: la probabilité de ne pas réussir à marquer des points de contours réels doit être faible ainsi que la probabilité de marquer des points n'appartenant pas a des contours.

- ✓ Bonne localisation: Les points marques comme appartenant à un contour par l'operateur doivent être aussi proches que possible du centre du contour véritable.

- ✓ Une seule réponse à un seul contour: puisque s'il y a deux réponses à un même contour, l'une d'entre elle doit être considérée fausse.

Canny suit les étapes suivantes pour la détection de contour d'une image

a. Réduction du bruit

Il est inévitable que toutes les images prises à partir d'un appareil contenant une certaine quantité de bruit.

Pour éviter que le bruit soit pris pour les bords, le bruit doit être réduit. Par conséquent, l'image est d'abord lissée par l'application d'un filtre gaussien.

La fonction gaussienne en 2D est la suivante :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad \textbf{EQ 2.1}$$

x : est la distance à l'origine sur l'axe horizontal

y : est la distance à l'origine sur l'axe vertical

σ : est l'écart type de la distribution gaussienne

Sa répartition est indiquée dans la figure 2.8

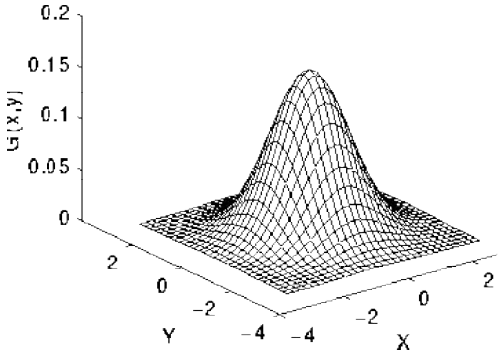


Figure 2.8 Répartition de la fonction gaussienne

L'idée du lissage gaussien est d'utiliser cette distribution 2D comme une «fonction de point de propagation», ce qui est obtenu par convolution. Comme l'image est stockée comme une collection de pixels discrète nous avons besoin pour produire une approximation discrète de la fonction gaussienne avant que nous puissions effectuer la convolution. En théorie, la distribution gaussienne est non nulle partout, ce qui exigerait un noyau de convolution infiniment grand, mais en pratique,il est effectivement nul plus que de trois écarts-types de la moyenne, et si nous pouvons tronquer le noyau à ce stade. La figure 2.9 montre un noyau de convolution approprié à valeurs entières qui se rapproche une gaussienne avec une $\sigma = 1$.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 2.9 noyau de convolution qui se rapproche une gaussienne avec une $\sigma = 1$

b. Gradient d'intensité

La détection de contour par Canny est basé la ou il y a un changement brusque d'intensité, ces zones sont considérées par la détermination des gradients de l'image.

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] \qquad \textbf{EQ 2.2}$$

Le gradient à chaque pixel de l'image lissée sont déterminés en appliquant l'opérateur de Sobel.

Canny estime le gradient dans la direction x et y, en appliquant ces deux noyaux

$$K_{GX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \textbf{EQ 2.3}$$

$$K_{GY} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad \textbf{EQ 2.4}$$

Il calcule le module et amplitude du gradient par les relations :

$$|G| = \sqrt{G_X^2 + G_Y^2} \qquad |G| = |G_x| + |G_y| \qquad \textbf{EQ 2.5}$$

c. Direction des contours

Une fois que le gradient est déterminé à chaque pixel, les orientations des contours seront calculées par la formule :

$$\theta = \arctan(\frac{|G_Y|}{|G_X|}) \qquad \textbf{EQ 2.6}$$

La figure 2.10 décrit les trois possibilités de la direction du gradient

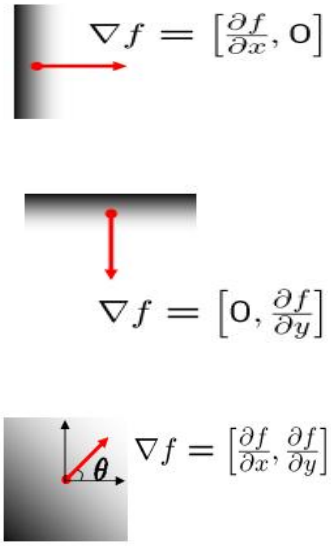


Figure 2.10 les trois possibilités de la direction du gradient

d. Suppression des non-maxima

Le but de cette étape est de convertir les bords "flous" à l'image du gradient aux bords «forts».

Cela se fait en conservant tous les maxima locaux de l'image gradient, et en supprimant tout le reste. L'algorithme est, pour chaque pixel de l'image gradient:

- Utiliser les 8 pixels voisin étant donnée une direction du gradient θ comme suit.
- Comparer la force du bord de pixel courant avec c'elle du pixel dont la direction du gradient positif et négatif .À savoir si la direction du gradient est au nord (θ =90 °), comparer avec les pixels au nord et au sud et ainsi de suite.
- si la force du bord du pixel courant est plus large que ses voisins on préserve la valeur si non on l'ignore

Les exemples suivants montrent la suppression des non-maxima

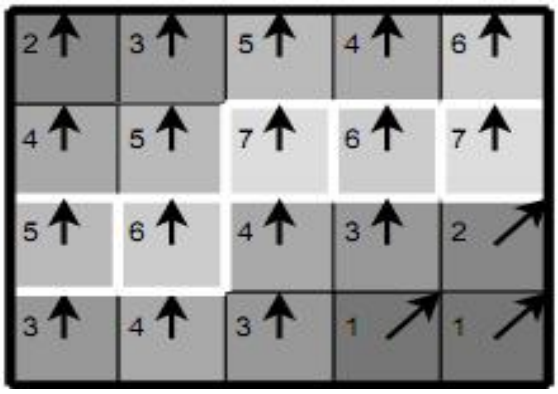


Figure 2.11 exemple -1- Suppression des non-maxima

Les résultats sont ceux marquées en blanc

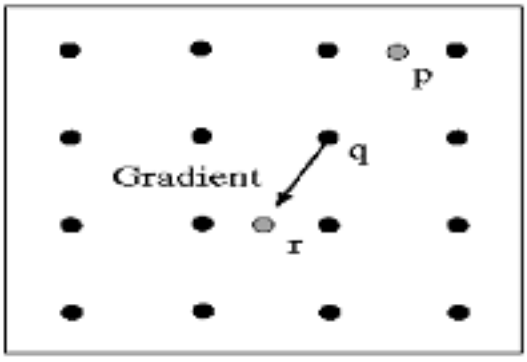


Figure 2.12 exemple -2- Suppression des non-maxima

Au point q nous avons un maximum si la valeur du gradient est plus grande que celles de p et r.

e. Seuillage des contours

La différenciation des contours sur la carte générée se fait par seuillage à hystérésis. Cela nécessite deux seuils, un haut et un bas; qui seront comparés à l'intensité du gradient de chaque point. Le critère de décision est le suivant. Pour chaque point, si l'intensité de son gradient est:

- Inférieur au seuil bas, le point est rejeté
- Supérieur au seuil haut, le point est accepté comme formant un contour
- Entre le seuil bas et le seuil haut, le point est accepté s'il est connecté à un point déjà accepté.

2.3.2.2.2. Les critères de Shen-Castan

Shen et Castan ont proposé [18] un opérateur optimisant un critère incluant la détection et la localisation. Les critères qu'ils obtiennent correspondent aux critères de détection et de localisation de Canny et les filtres obtenus sont assez similaires dans la pratique. Le calcul du filtre de lissage optimal se fait par modélisation de la frontière par un échelon d'amplitude A noyé dans un bruit blanc stationnaire additif de moyenne nulle ; Le filtre de lissage obtenu par Shen et Castan s'écrit :

$$f(x) = ce^{-\propto|x|} \qquad \textbf{EQ 2.7}$$

$$c = \frac{1-e^{-\propto}}{1+e^{-\propto}} \qquad \textbf{EQ 2.8}$$

et le filtre de dérivation correspondant :

$$h(x) = \begin{cases} d.e^{-\propto x} & si\ x \geq 0 \\ d.e^{\ \propto x} & si\ x \leq 0 \end{cases} \qquad \textbf{EQ 2.9}$$

avec $d = 1 - e^{-\propto}$ Le paramètre α définit la ``largeur'' du filtre : plus α est petit, plus le lissage effectué par le filtre est important.

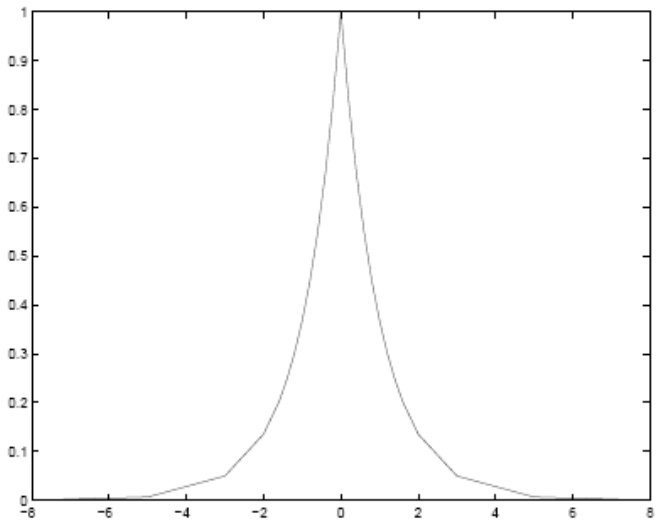


Figure 2.13 Réponse impulsionnelle du filtre de shen-castan

2.3.2.2.3. Les critères de deriche

Deriche a proposé [19] un filtre de lissage dont la dérivée est la solution exacte de l'équation de Canny étendue aux filtres à supports infinis. Le filtre de lissage correspondant est :

$$h(x) = k\,(\propto |x| + 1)e^{-\propto|x|} \qquad \textbf{EQ 2.10}$$

avec

$$k = \frac{(1-e^{-\propto})^2}{(1+2\propto e^{-\propto}-e^{-2\propto}} \qquad \textbf{EQ 2.11}$$

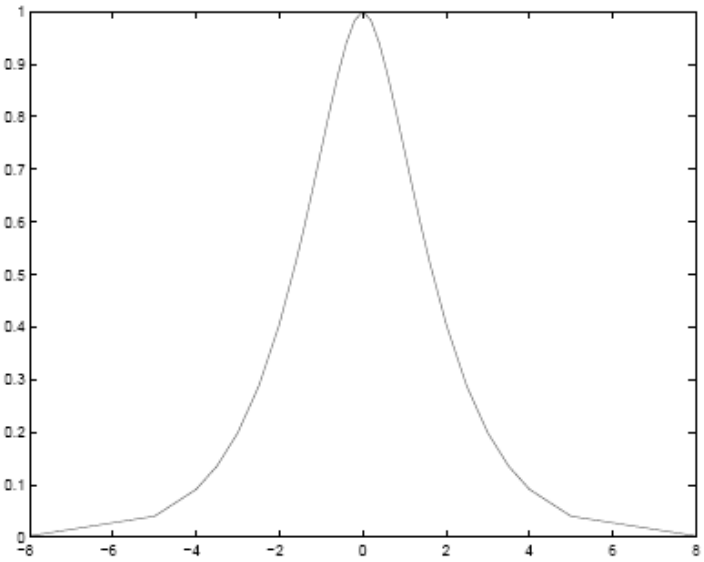


Figure 2.14 Réponse impulsionnelle du filtre de deriche

2.3.2.3. L'approche contour actif

Les contours actifs sont des techniques de segmentation permettant d'extraire un objet d'intérêt et d'une image. Cette segmentation n'est pas immédiate, elle requiert une phase dynamique du contour (d'où la dénomination "actif") qui évoluera itérativement au cours du temps artificiel t, de sa position initiale vers les bords de l'objet à extraire. Une telle évolution temporelle peut se formaliser mathématiquement sous la forme d'une équation d'évolution exprimant explicitement ou implicitement la vitesse du contour actif. Il existe plusieurs manières d'obtenir une équation d'évolution. L'une consiste à dériver cette équation de la minimisation d'une fonctionnelle d'énergie, on parle alors d'approche variationnelle. Une alternative consiste à construire une équation d'évolution par analogie avec d'autres disciplines scientifiques comme la Physique, c'est l'approche géométrique. Nous nous attacherons à décrire, puis utiliser l'approche variationnelle dans ce travail de thèse. Avec une telle approche, la fonctionnelle d'énergie peut être schématisée comme la somme de deux classes de termes énergétiques. La première classe concerne l'énergie interne du contour visant à contrôler des contraintes intrinsèques `a celui-ci, comme par exemple sa régularité. La deuxième classe est relative au terme d'attache aux données qui fera interagir le contour actif avec des caractéristiques extraites de l'image. Dans les travaux de Foulonneau [20] et Jehan-Besson [21], ce terme d'énergie externe est appelé critère et est construit à partir de descripteurs. Un descripteur est une mesure faite sur l'image permettant de caractériser une frontière ou une région. Un descripteur de frontière serait par exemple la carte des gradients de l'image, un descripteur d'une région R pourrait être la moyenne des pixels de l'image inclus dans R. Selon le choix du descripteur adopté dans la fonctionnelle d'énergie, on dérivera différents types de contours actifs plus ou moins performants en fonction de la nature de l'image à analyser.

Donc il existe différentes catégories de contours actifs. Celles-ci se différencient par la façon avec laquelle on déduit l'équation d'évolution, par le mode de représentation du contour actif, et finalement le terme d'attache aux données qui peut être soit basé sur les frontières, les régions ou bien les deux.

Historiquement, c'est la représentation du contour actif a émergé en premier dans la communauté de Vision par Ordinateur grâce aux travaux pionniers de Kass, Witkin et Terzopoulos en 1987 [22]. Une telle représentation consiste à paramétrer le contour actif Γ par un paramètre arbitraire $\mathcal{P} \in$ [a, b] et le temps $\tau \in$ [0, T].

$$\begin{cases} & [a,b] * [0,T] \rightarrow \mathbb{R}^2 \\ (\mathcal{P},\tau) & \rightarrow \boldsymbol{\Gamma}(\mathcal{P},\tau) = X(\mathcal{P},\tau) = \begin{pmatrix} x(\mathcal{P},\tau) \\ y(\mathcal{P},\tau) \end{pmatrix} \end{cases} \qquad \textbf{EQ 2.12}$$

L'évolution de ce contour actif $\boldsymbol{\rho}$ est régie par une équation aux dérivées partielles de la forme suivante :

$$\begin{cases} \frac{\partial \boldsymbol{\Gamma}(\mathcal{P},\tau)}{\partial \tau} = v(\mathcal{P},\tau) \\ \boldsymbol{\Gamma}(\mathcal{P},0) = \boldsymbol{\Gamma}_0(\mathcal{P}) \end{cases} \qquad \textbf{EQ 2.13}$$

$\boldsymbol{\Gamma}_0(\mathcal{P})$ est le contour initial qui peut être déni manuellement par exemple et v est la vitesse d'évolution de la courbe. La figure 2.15 illustre cette évolution.

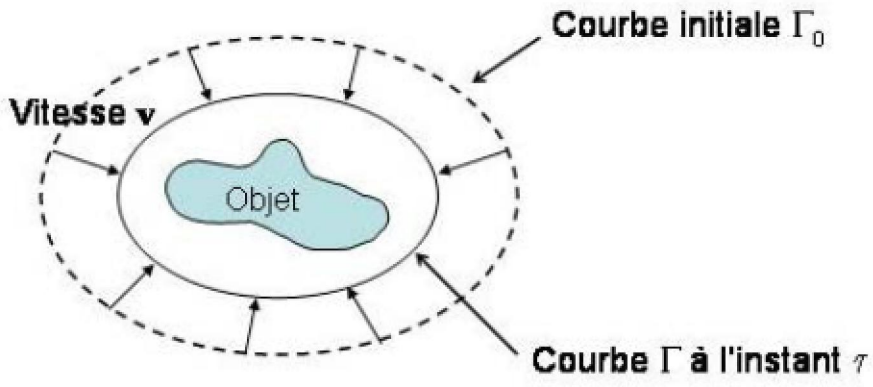


Figure 2.15 Evolution du contour actif $\boldsymbol{\Gamma}(\tau)$ avec la vitesse v vers l'objet d'intérêt

La vitesse d'évolution v du contour actif a a priori une direction quelconque. Décomposons cette vitesse dans le repère de Frenet associé à la courbe. Nous obtenons deux composantes, une composante selon la tangente unitaire **T** et une composante selon la normale unitaire intérieure à la courbe **N**. Cela donne :

$$\frac{\partial \Gamma(\mathcal{P},\tau)}{\partial \tau} = v(\mathcal{P},\tau) = F_N(\mathcal{P},\tau)N(\mathcal{P},\tau) + F_T(\mathcal{P},\tau)T(\mathcal{P},\tau) \qquad \textbf{EQ 2.14}$$

où F_N est l'amplitude de la vitesse dans la direction normale au contour et $F_T$est l'amplitude de la vitesse dans la direction tangentielle au contour. La composante normale de la vitesse modifie la géométrie de la courbe tandis que la composante tangentielle de la vitesse modifie sa paramétrisation. L'équation d'évolution peut être simplifiée en considérant que nous nous intéressons uniquement à la déformation de la courbe et non à sa paramétrisation [23]. La partie de l'équation qui nous intéresse est donc uniquement la projection sur N car c'est celle qui régit le déplacement du contour, tandis que la projection sur T ne permet que de déterminer la paramétrisation du contour. L'équation d'évolution du contour actif s'écrit donc uniquement avec une vitesse normale au contour :

$$\begin{cases} \frac{\partial \Gamma(\mathcal{P},\tau)}{\partial \tau} = v(\mathcal{P},\tau) = F_N(\mathcal{P},\tau)N(\mathcal{P},\tau) \\ \Gamma(\mathcal{P},0) = \Gamma_0(\mathcal{P}) \end{cases} \quad \textbf{EQ 2.15}$$

l'équation d'évolution de ce modèle de contours actifs est obtenue en minimisant la fonctionnelle suivante :

$$E_{image} = \alpha E_{ligne} + \beta E_{contour} + \lambda E_{term} \quad \textbf{EQ 2.16}$$

Avec :

$$E_{ligne} = I(x,y) \quad \textbf{EQ 2.17}$$

$$E_{contour} = -|\nabla I(x,y)|^2 \quad \textbf{EQ 2.18}$$

$$E_{term} = \frac{C_{yy}C_x{}^2 - 2C_{xy}C_xC_y + C_{xx}C_y{}^2}{(C_x{}^2 + C_y{}^2)^{3/2}} \quad \textbf{EQ 2.19}$$

avec $\alpha, \beta\ et\ \lambda$ des constantes positives.

2.4.Conclusion

Dans l'approche contour, on trouve les méthodes dérivatives qui sont les plus immédiates pour détecter et localiser les variations du signal.

L'avantage des opérateurs de détection des contours est la simplicité d'utilisation. Par contre, ils sont très sensibles aux bruits et donnent des contours ouverts.

Contrairement aux approches classiques de détection de contours, les modèles déformables incorporent de la connaissance à priori sur la forme de l'objet recherché. Cette connaissance peut être issue d'une base d'apprentissage où une forme référence (forme prototype), ainsi des modes de déformation peuvent être extraits. Elle peut

cependant être plus générale et se limiter à des propriétés comme la continuité ou la régularité qui caractérise le contour d'un objet.

Les contours actifs sont largement utilisés pour leur capacité à intégrer les processus de détection et de chaînage des contours en un seul processus de minimisation d'énergie; Toutefois l'estimation des paramètres et les problèmes d'initialisation font des contours actifs une méthode difficile à calibrer.

Les différentes approches de détection de contour ont été présentées dans ce chapitre. Deux méthodes ont été retenues :

- La méthode basée sur les critères de canny sera implémentée sur FPGA et décrite dans le chapitre 3.

- Le chapitre 4 décrit l'implémentation du contour actif en temps réel sur FPGA.

3.1.Introduction

Le but de ce chapitre est de présenter une partie du travail réalisé au cours de ce projet, qui consiste à implémenter l'approche analytique de canny sur FPGA ; en commençons par détaillé l'approche de canny et décrivons l'architecture proposé pour l'implémentation sur FPGA et enfin présenté les résultats obtenus sous l'environnement de développement xilinx ise.

3.2.La détection de contour par l'approche de canny

Dans son article canny [17] a développé une forme mathématique permettant d'avoir une bonne détection, une bonne localisation et Une seule réponse à un seul contour.

Comme il a été présenté en premier chapitre on peut résumer l'approche de canny dans le schéma synoptique suivant :

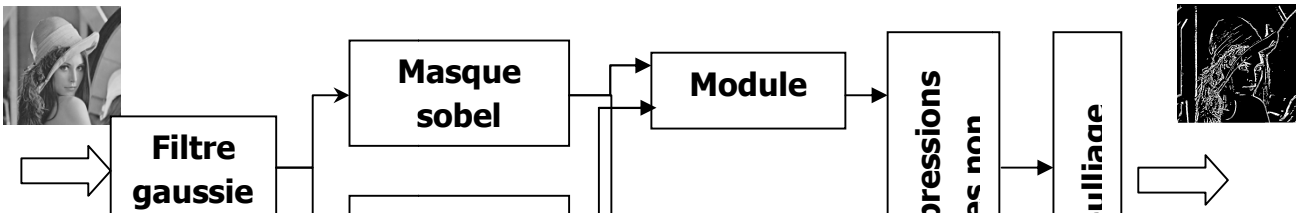


Figure 3.1 Schéma synoptique du détecteur de contour de canny

3.2.1. Réduction du bruit

L'image d'entrée est lissée par un filtre gaussien afin de réduire le bruit.

Le principe est de faire une convolution de l'image avec un masque gaussien, pour notre implémentation nous avons opté pour un masque de taille 3*3.

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

L'application d'un masque sur une image est généralement le produit de convolution du masque avec l'image ; d'une autre manière pour un masque de 3*3 ; Le filtre pointe successivement chacun des pixels de l'image. Pour chaque pixel, que nous appellerons « pixel initial », il multiplie la valeur de ce pixel par la valeur du noyau et de chacun des 8 pixels qui l'entourent par les valeurs correspondantes dans le masque. Il additionne l'ensemble des résultats et le nouveau pixel initial prend alors la valeur du résultat final.

L'exemple suivant Figure 3.2 illustre les étapes de la convolution d'une image avec un filtre 3*3. L'image est représentée sous forme d'une matrice, dont ces éléments sont les valeurs de chaque pixel.

Le pixel initial est encadré en gras, la zone d'action du noyau est la matrice qui contient le pixel initial comme centre. Cette matrice est de même taille que le masque de convolution.

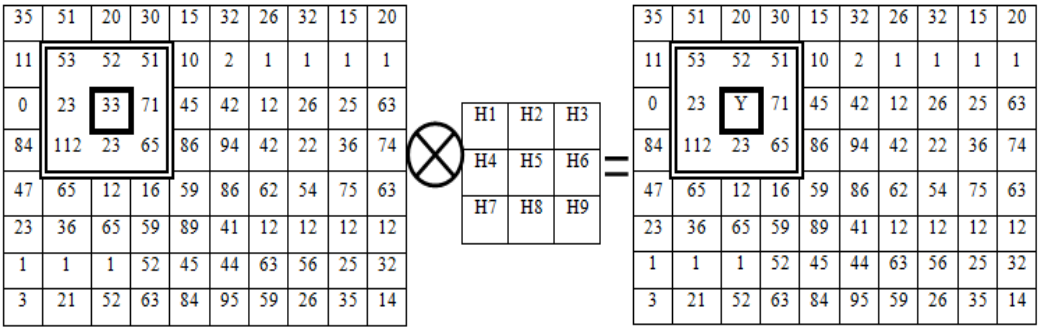


Figure 3.2 convolution d'une image avec un filtre 3*3

Le pixel initial a pris une nouvelle valeur Y qui est calculée de la manière suivante :

Y=(53*h1)+(52*h2)+(51*h3)+(23*h4)+(33*h5)+(71*h6)+(112*h7)
+(23*h8)+(65*h9)

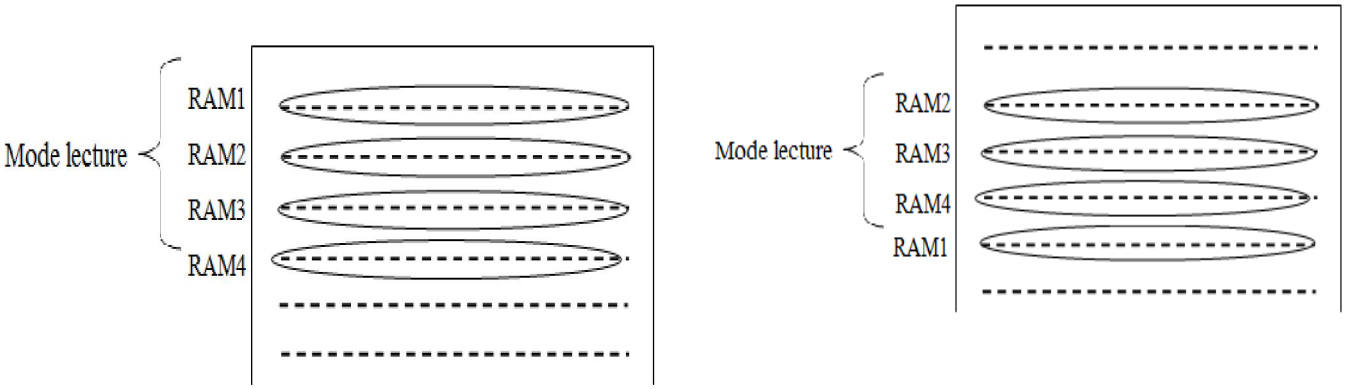
Afin d'implémenter un produit de convolution d'une image avec un filtre 2D de taille 3*3 sur FPGA, avec une description VHDL, nous avons utilisé la démarche suivante :

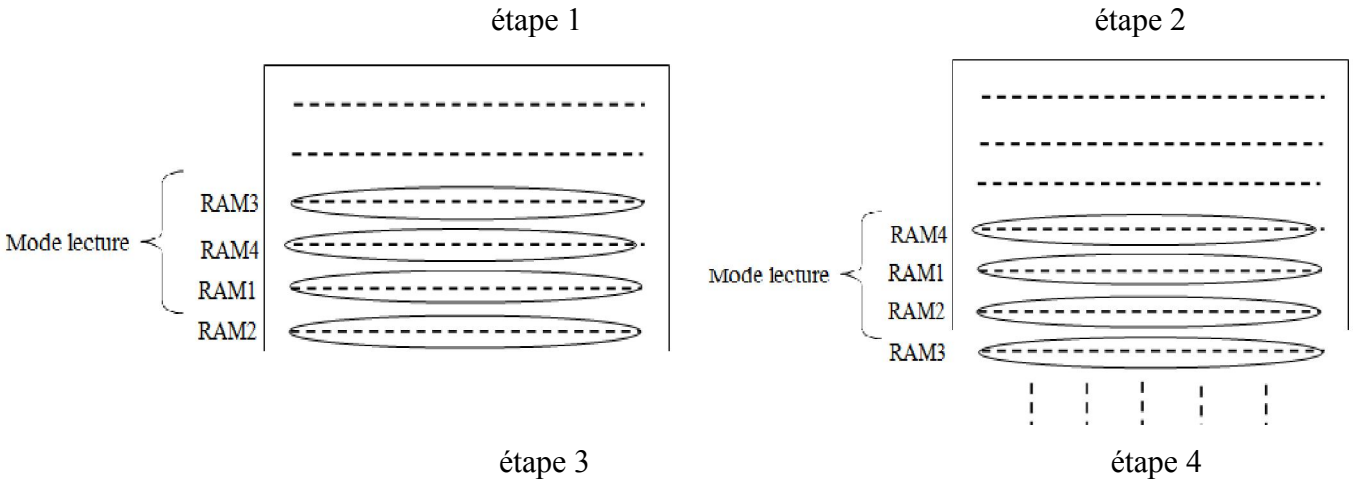
- Afin de faire la convolution avec un masque 3*3 ; il faut avoir au minimum les données de trois lignes, pour cela on a utilisé quatre RAM pour sauvegarder chaque ligne dans une RAM.
- Sachant qu'une RAM ne peut être qu'on mode lecture ou écriture et on va la remplir par les pixels d'une seul ligne ; alors on a réalisé un module (avec une description VHDL) permettant de générer les signaux de lecture / écriture et des adresses pour chaque RAM.
- Une RAM parmi quatre est en mode écriture, les autres sont en mode lecture.
- Une fois les trois premières lignes sont sauvegarder le calcul de produit de convolution peut s'effectué.

Pour la description VHDL on a utilisé des RAM block de type 16_S9.

A noter qu'il faut ajouter une bibliothèque adéquate afin que le composant soit reconnu.

- La même description VHDL est utilisée pour les quatre RAM.





La figure 3.3 représente le schéma synoptique du module permettant de générer les signaux de lecture / écriture et des adresses.

La figure 3.4 montre les signaux lecture / écriture générés pour chaque RAM avec une simulation sous modelsim.

La figure 3.5 montre les adresses générer pour chaque RAM avec une simulation sous modelsim.

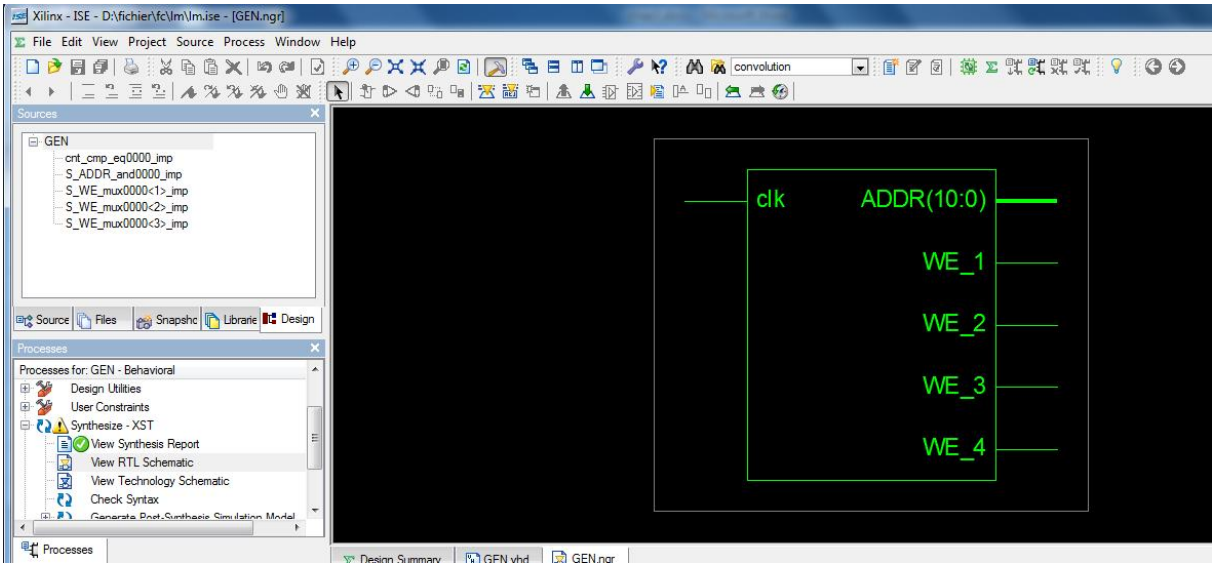


Figure 3.3 Schéma synoptique du module générateur de signaux lecture / écriture

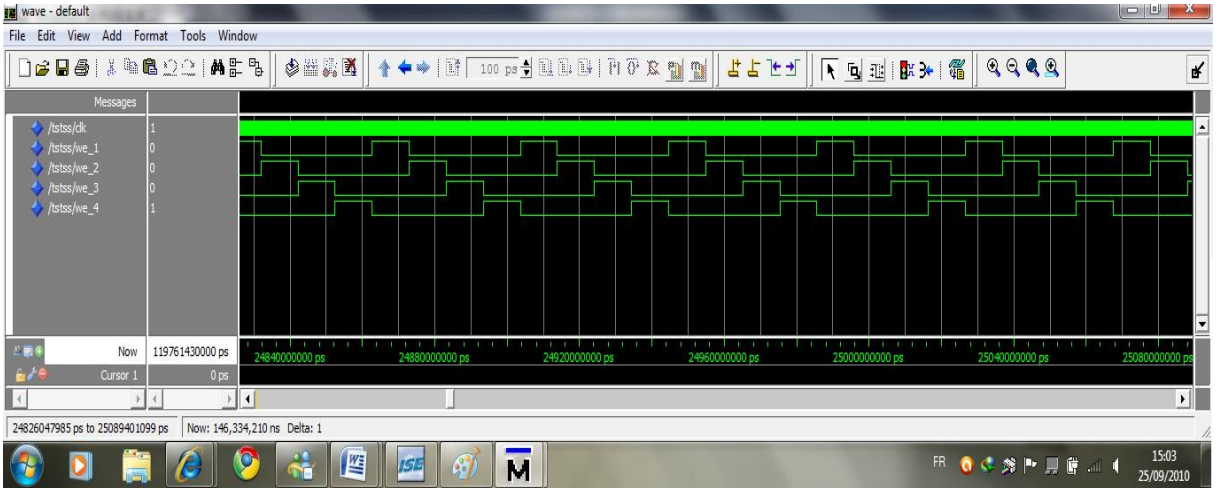


Figure 3.4 Les signaux lecture / écriture des RAM

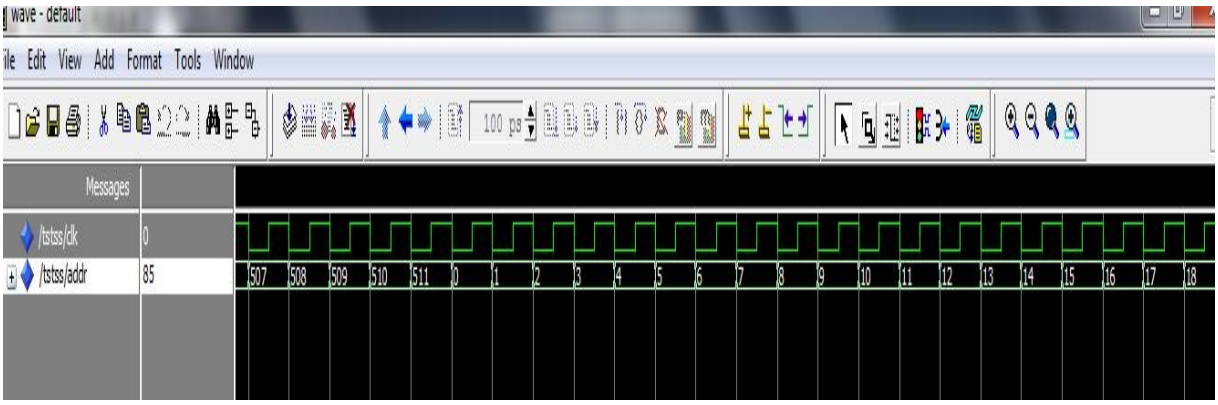


Figure 3.5 Les adresses

La dernière étape été de faire le calcul de la convolution (multiplications et additions).

- On a essayé de réduire le temps de calcul en utilisant une convolution avec des masques dont les valeurs sont des puissances de deux afin d'obtenir des décalages au lieu de multiplication.

La figure 3.6 montre une partie de la description VHDL du calcul de la convolution.

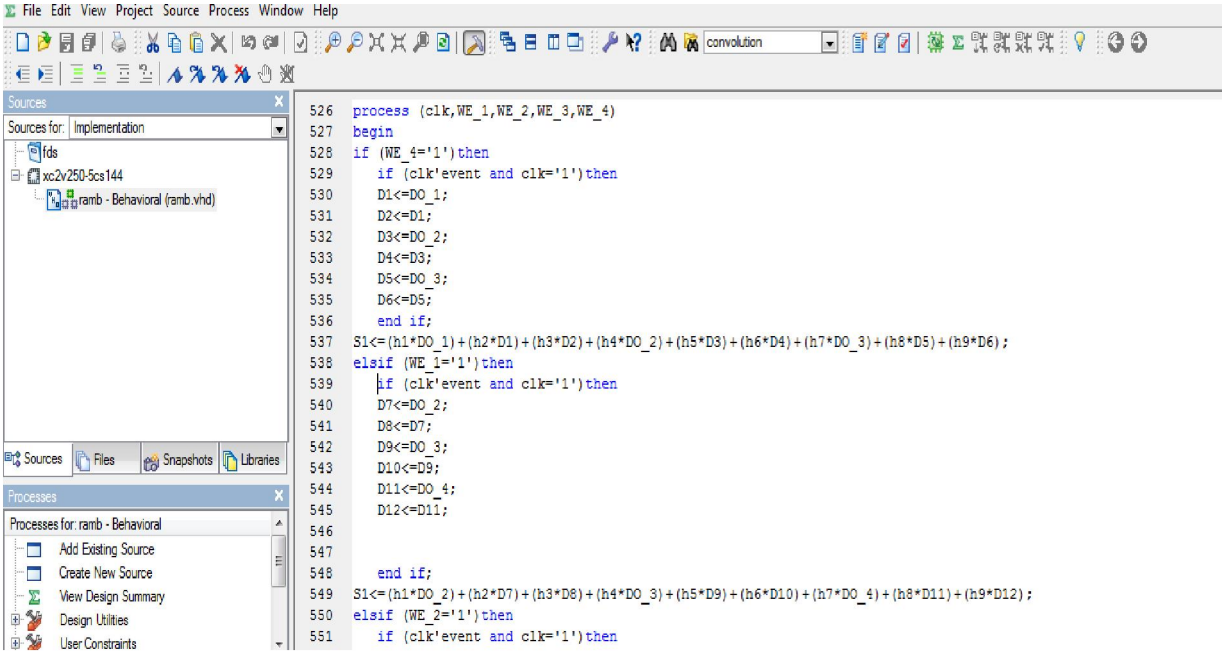


Figure 3.6 description VHDL du filtre 3*3

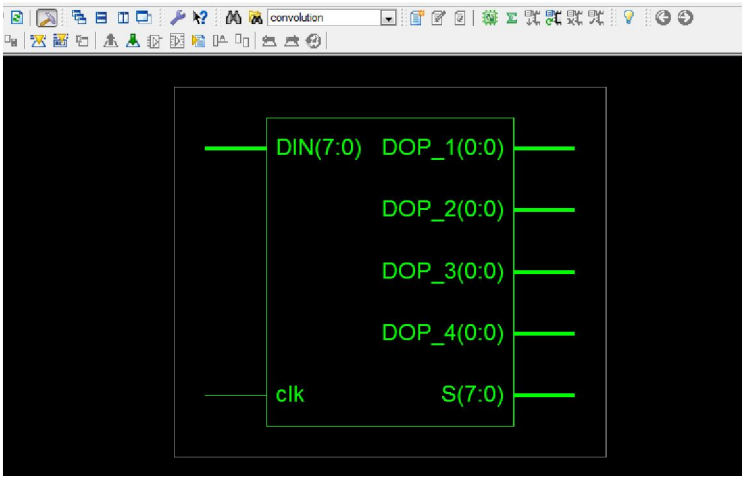


Figure 3.7 Schéma synoptique final du détecteur de contour par masque

Simulation avec ModelSim

Afin de visualiser les résultats du travail, nous avons opté pour une simulation avec une description VHDL, qui assure la manipulation des fichiers. Le Modelsim 6.3f est utilisé.

La figure 3.8 montre une partie de la description VHDL de la manipulation des fichiers.

```
84          clk <= '0';
85          wait for clk_period/2;
86          clk <= '1';
87          wait for clk_period/2;
88       end process;
89    process (clk)
90    type file_in is file of character;
91    type file_out is file of character;
92    file my_file_in : file_in open READ_MODE is "D:\LENA.IMA";
93    file my_file_out : file_out open write_MODE is "D:\lapla2.IMA";
94    variable c : character;
95    variable d : character;
96    begin
97    if (not endfile (my_file_in)) then
98    if (clk'event and clk='1') then
99    read(my_file_in, c);
100   DIN<=CONV STD LOGIC VECTOR (character'pos(c),8);
```

Figure 3.8 Description VHDL de l'appel d'un fichier

Les images, que nous avons utilisé, été dans leur format brut. Donc pour les visualisés nous avons développé un petit programme en langage C pour convertir les images brut en format bitmap (ou bien utilisé le logiciel adobe photo shop).

3.3. Conception et Implémentation à base d'FPGA

3.3.1. Lissage

La figure 3.9 illustre un schéma bloc représentant les entrés et les sorties de ce module.

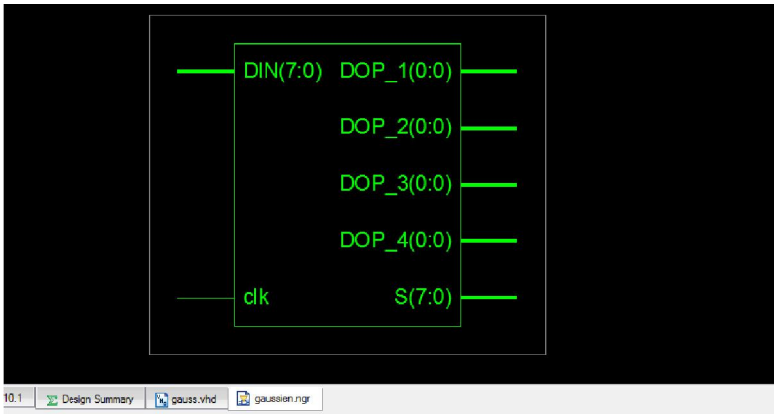


Figure 3.9 schéma bloc du filtre gaussien

En premier lieu, on a fait lissage sur les images de la avec un filtre gaussien 3*3, et on a obtenue le résultat présenté dans la figure 3.10.

Figure 3.10 Images lissé par masque gaussien

3.3.2. Gradient d'intensité

Comme il a été expliqué précédemment, Canny applique deux filtres de Sobel (horizontal et vertical) sur l'image lissée par le masque gaussien.

Puisque la taille du masque reste la même, donc nous réutilisons le même bloc du filtre 3*3. Le résultat obtenu est celui des figures 3.11 et 3.12

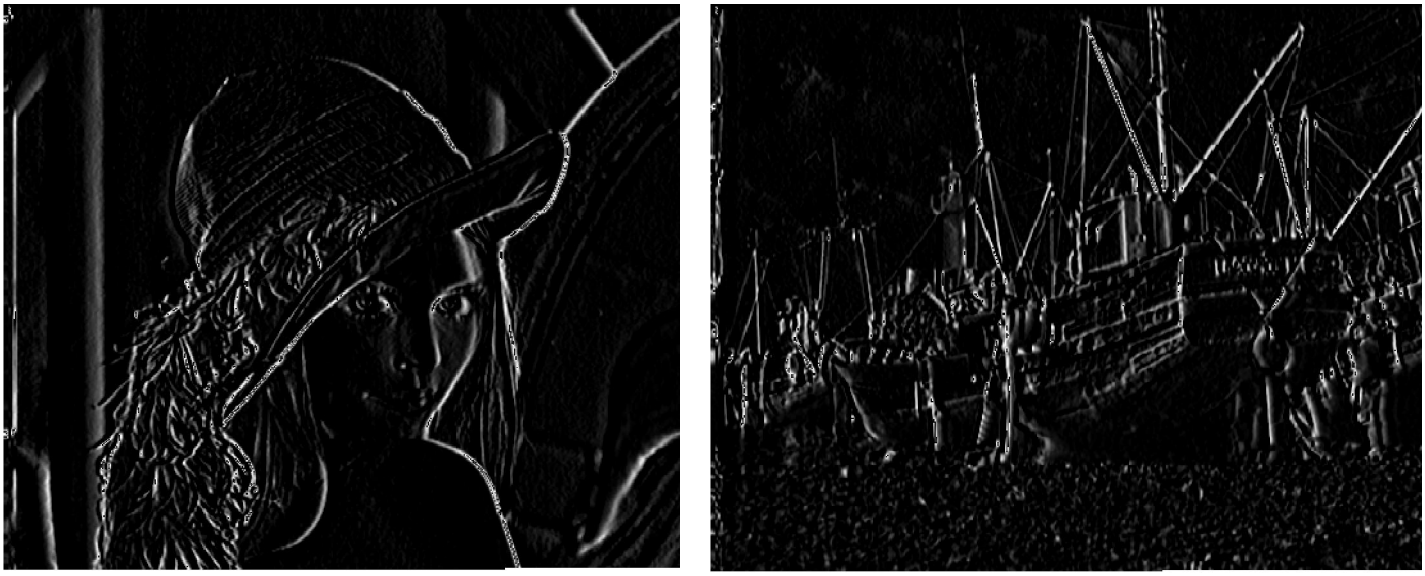


Figure 3.11 Implémentation du masque Sobel horizontale

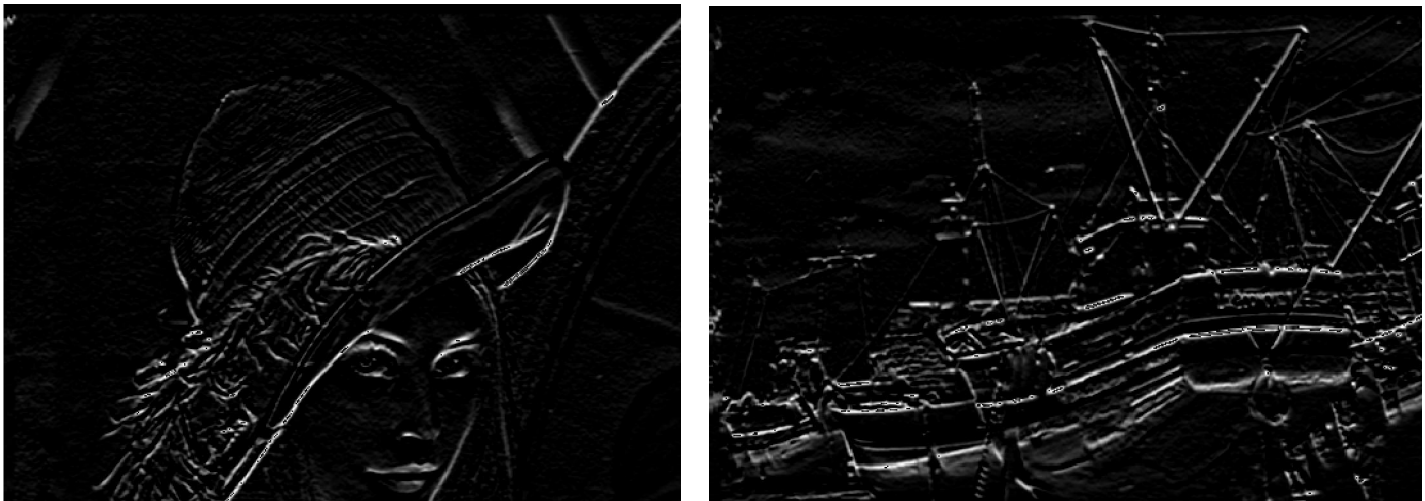


Figure 3.12 Implémentation du masque Sobel vertical

Le résultat du module du gradient est sur la figure 3.13.

Figure 3.13 implémentation du module du gradient

3.3.3. Calcul de la direction du gradient

Pour la direction du gradient, il faut calculer $\theta = Arctan\ (\frac{Sy}{Sx})$

Sx et Sy sont les résultats de la convolution avec les filtres de Sobel horizontal et vertical respectivement. Nous utilisons des approximations sur les directions obtenues pour avoir seulement les 4 angles 0°, 45°, 90° et 135° ; alors:

Pour $0° < \theta < 22.5°$ alors en prend $\theta = 0°$;

Pour $22.5° < \theta < 67.5°$ alors en prend $\theta = 45°$;

Pour $67.5° < \theta < 112.5°$ alors en prend $\theta = 90°$;

Pour $112.5° < \theta < 157.5°$ alors en prend $\theta = 135°$.

La figure 3.14 représente le schéma bloc du module qui calcul la direction de gradient.

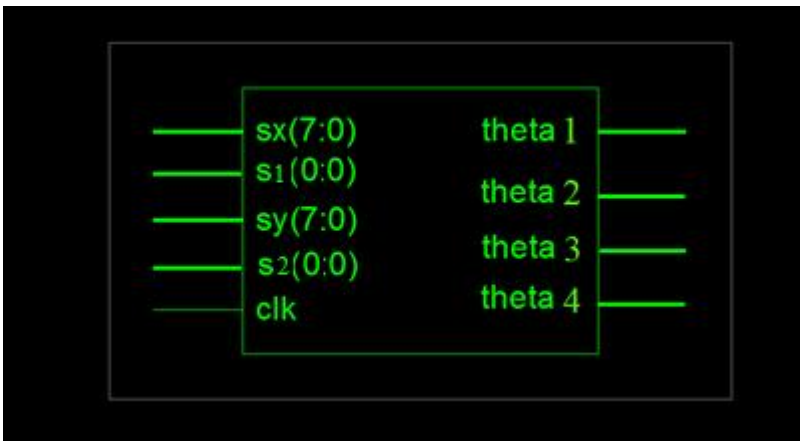


Figure 3.14 schéma bloc du calcul de la phase

Pour ces approximations on obtient respectivement :

$0 < \frac{Sy}{Sx} < 0.41$;

$0.41 < \frac{Sy}{Sx} < 2.41$;

$\frac{Sy}{Sx} > 2.41$;

$-0.41 < \frac{Sy}{Sx} < -2.41$.

Puisque le but de notre travail été d'avoir des résultats en temps réelle alors nous avons opté pour l'approche suivante, afin de réduire les opérations d'où le temps de calcul.

$$0 < \frac{Sy}{Sx} < 0.5 \quad \Rightarrow \quad 0 < Sy < \frac{Sx}{2}$$

$$0.5 < \frac{Sy}{Sx} < 2 \quad \Rightarrow \quad \frac{Sx}{2} < Sy < 2 * Sx$$

$$\frac{Sy}{Sx} > 2 \quad \Rightarrow \quad Sy > 2 * Sx$$

$$-0.5 < \frac{Sy}{Sx} < -2 \quad \Rightarrow \quad -\frac{Sx}{2} < Sy < -2 * Sx$$

La figure 3.15 représente le schéma synoptique de notre approche

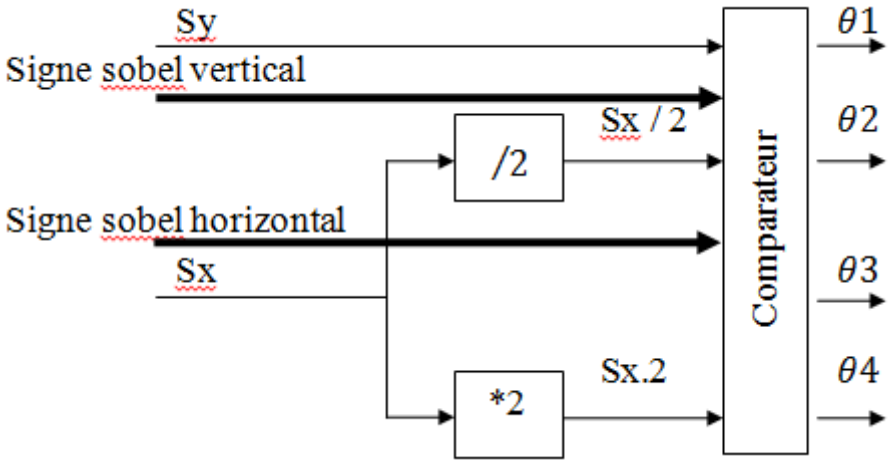


Figure 3.15 schéma synoptique de l'approche de calcul de la phase

Sachant que la division sur deux et la multiplication par deux n'est que des décalages à droite et à gauche respectivement ; alors nous n'aurons aucune opération arithmétique à faire sauf des comparaisons.

3.3.4. Suppression des non-maxima

Pour déterminer si un pixel est un maximum local, il faut comparer son module de gradient avec les deux pixels adjacents selon sa direction de gradient, alors cette étape ne comporte aucun calcul sauf des comparaisons.

La figure 3.16 illustre un exemple de la suppression des non maximum locaux.

La figure 3.17 représente le schéma synoptique du module de la suppression de non maxima locaux.

La figure 3.18 montre le schéma bloc de la suppression pour l'exemple de la figure 3.16.

1	2	3
4	5	6

7	8	9

*Si A5 à une direction $\theta = 45°$ alors on compare A5 avec A3 et A7.

*Si A5=Max(A3,A5,A7) alors A5 garde sa valeur

*Sinon A5=0

Figure 3.16 exemple de la suppression des non maxima

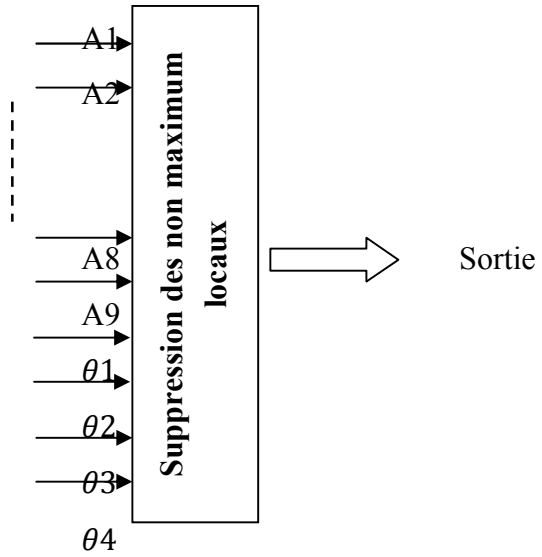


Figure 3.17 schéma synoptique de la suppression de non maxima locaux

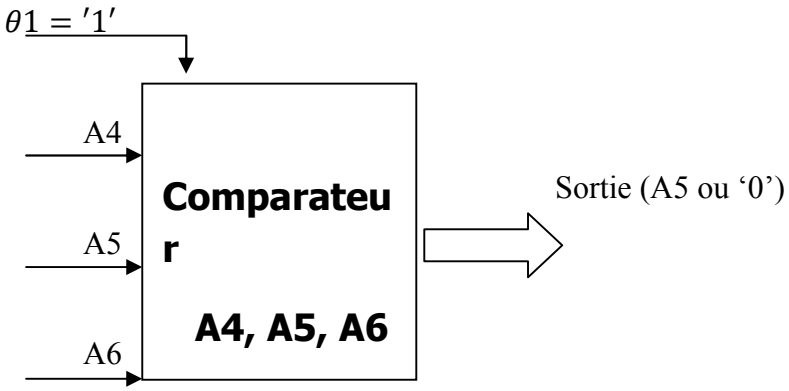


Figure 3.18 schéma bloc de la suppression

Les résultats obtenus sont dans la figure 3.19

Figure 3.19 Suppression des non maximum locaux

3.3.5. Seuillage

Le seuillage utilisé dans notre travail est un seuillage par hystérésis dynamique, donc on a deux seuils haut et bas,

- si la valeur du pixel est supérieure au seuil haut alors ce pixel appartient au contour (Blanc)
- Si la valeur du pixel est inférieure au seuil bas alors ce pixel n'appartient pas au contour (noir)
- Si la valeur du pixel est comprise entre les deux seuils alors se pixel dépend du pixel précédent.

Les deux seuils changes en fonction de la position du pixel, on calcul la moyenne et on s'adapte à la zone ou il se positionne le pixel.

La figure 3.20 représente le schéma synoptique du module de seuillage par hystérésis dynamique.

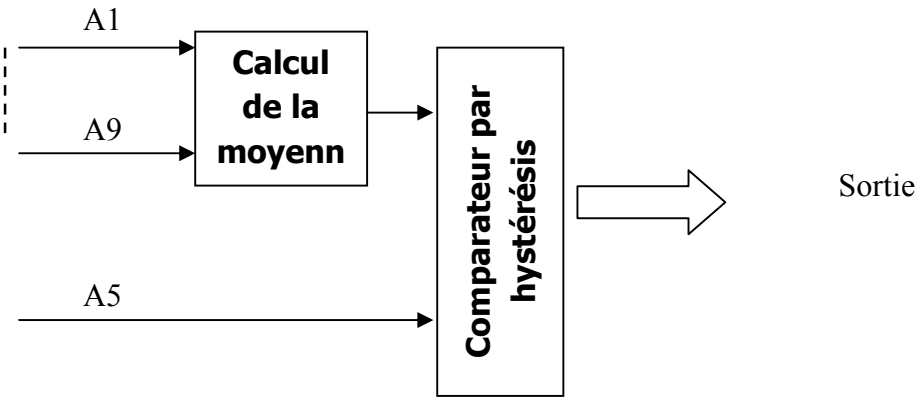


Figure 3.20 schéma synoptique du seuillage par hystérésis dynamique

Le résultat obtenu est présenté dans la figure 3.21

Figure 3.21 détection de contour par approche de Canny sur FPGA

4. Conclusion

Les résultats obtenus de l'implémentation de l'algorithme de canny sous l'environnement ISE sont satisfaisantes, en plus ; la fréquence de travail qui à été abouti sur FPGA est de 200MHz, d'où l'on peut dire que la détection de contour à été réaliser en temps réel avec succès.

4.1.Introduction

Le but de ce chapitre est de proposé une architecture d'un contour actif à faible cout et en temps réel afin de l'implémenter sur FPGA.

L'algorithme choisit pour l'implémentation est celui des snake vu la simplicité de son principe et les avantage qu'il offre.

Afin d'accomplir ce travail ; le présent chapitre a été structuré sur deux phases de la manière suivante :

- Au cours de la première nous allons présenter l'architecture proposée et la description en VHDL de chaque module utilisé.
- La deuxième partie présente les résultats obtenus avec l'outil de simulation modelsim.

4.2.Implémentation de l'algorithme des snakes

Un contour actif est un ensemble de points qu'on va tenter de déplacer pour leur faire épouser une forme. Il s'agit d'une technique d'extraction de données utilisée en traitement d'images. L'idée de cette méthode est de déplacer les points pour les rapprocher des zones de fort gradient tout en conservant des caractéristiques comme la courbure du contour ou la répartition des points sur le contour ou d'autres contraintes liées à la disposition des points.

Au démarrage de l'algorithme, le contour est disposé uniformément autour de l'objet à détourer puis il va se rétracter pour en épouser au mieux ses formes. De la même manière, un contour actif peut aussi se dilater et tenter de remplir une forme, il sera alors situé à l'intérieur de celle-ci au démarrage de l'algorithme.

A chaque itération, l'algorithme va tenter de trouver un meilleur positionnement pour le contour pour minimiser les dérives par rapport aux contraintes utilisées. L'algorithme s'arrêtera lorsque le nombre maximum d'itérations aura été atteint. On utilise les notions d'énergies interne et externe pour caractériser respectivement la forme du contour et tous les éléments qui lui sont propres, et le positionnement du contour sur l'image.

La figure 4.1 illustre un exemple de déroulement d'une recherche en contour actif pour détourer un objet :

Figure 4.1 exemple d'un contour initiale

Chaque itération peut se représenter de la manière suivante :

- calcul des énergies interne et externe, caractérisant le contour lui-même et son positionnement sur l'image.
- pour chaque point du contour, détermination d'une nouvelle position, sur laquelle le contour devrait mieux minimiser les écarts de contraintes.
- arrangement du contour pour qu'il respecte des contraintes d'écartement entre les points, de régularité de points ...

4.2.1. L'énergie interne

L'énergie interne ne dépend pas de l'image ni de la forme à détourer, elle ne dépend que des points du contour.

Elle regroupe des notions comme la courbure du contour ou la régularité d'espacement des points. En effet, le contour doit conserver une forme arrondie en minimisant les dérivées d'ordre 1, 2, ... et doit empêcher un point de se détacher trop loin du reste du contour. Idéalement, l'énergie interne est minimale pour un cercle où tous les points sont régulièrement espacés.

4.2.2. L'énergie externe

L'énergie externe correspond à l'impact du contour sur l'image. Pour la calculer Kass a proposé :

$$E_{exter} = w_{ligne}E_{ligne} + w_{contour}E_{contour} + w_{term}E_{term}$$

Dont

$E_{ligne} = I(x, y)$

$E_{contour} = -|\nabla I(x, y)|^2$

$$E_{term} = \frac{C_{xx}C_y^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{(C_x^2+C_y^2)^{3/2}}$$

Avec

$$C_x = [-1 \quad 1] \otimes I(x,y)$$

$$C_y = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \otimes I(x,y)$$

$$C_{xx} = [1 \quad -2 \quad 1] \otimes I(x,y)$$

$$C_{yy} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \otimes I(x,y)$$

$$C_{xy} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \otimes I(x,y)$$

4.2.3. Utilisation des deux énergies

Chaque position du contour actif donne une énergie interne et une énergie externe dont la somme doit être minimisée et influencera les mouvements des points du contour actif.

Après avoir calculé l'énergie globale dégagée par le contour et par son positionnement sur l'image, il convient de déterminer comment le faire évoluer pour minimiser cette énergie. Pour cela, une méthode simple et intuitive est d'observer les pixels voisins immédiats de chaque point du contour pour déterminer pour chacun d'eux l'énergie globale du snake, chaque meilleur voisin devenant un point du contour.

4.2.4. Architecture proposé pour l'implémentation

L'organigramme de la figure 4.2 représente l'architecture proposé pour l'implémentation.

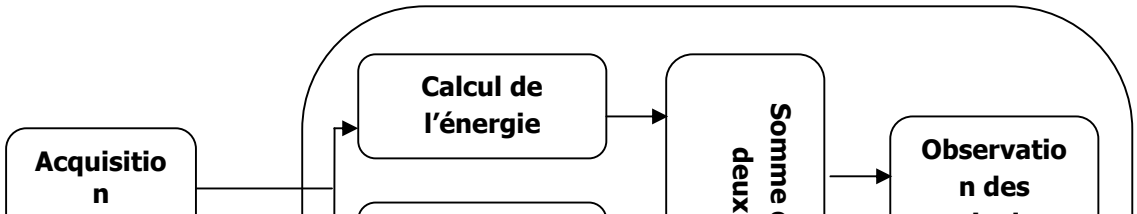


Figure 4.2 Organigramme présentant l'architecture proposée pour l'implémentation du contour actif

4.2.5. Acquisition d'image

Au cours de notre projet les images utilisé sont pris avec un appareil photo et sauvegarder dans le PC, ensuite on a changé leur format en RAW c'est-à-dire un format brut en code ascii à l'aide de l'outil adobe photo shop afin de les lire pixel par pixel dans l'outil de développement xilinx ise 12.3.

La lecture de l'image pixel par pixel se fait avec une description VHDL de la manière suivante :

Figure 4.3 procédure pour la lecture d'un fichier image en VHDL

4.2.6. Acquisition du contour initial

Pour l'acquisition du contour initial on a développé un code source matlab qui permet d'extraire les positions du contour initial, de les trier par ordre croissant et de les sauvegarder dans un fichier texte en code ascii.

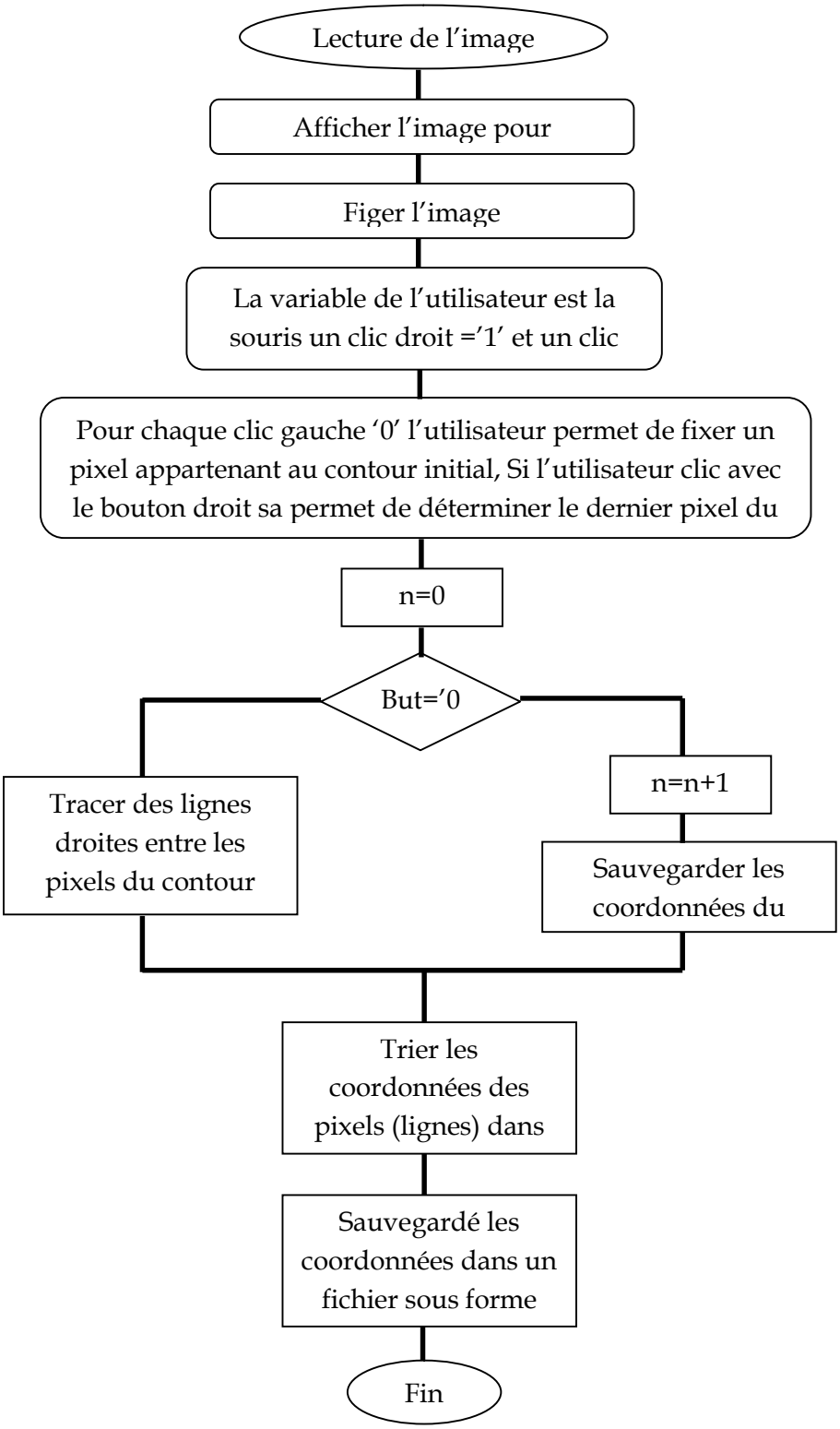


Figure 4.4 algorithme d'acquisition du contour initial

4.2.7. Sauvegarde des positions des pixels du contour

Après avoir sauvegardé les positions des pixels du contour initial dans un fichier texte, on va lire ce dernier avec une description VHDL afin de sauvegarder les positions dans une RAM bloque.

Lors de la lecture de l'image, on lit en parallèle le contenue de la RAM et détecter l'existence d'un pixel formant le contour initial.

Sachant que les positions des pixels formants le contour initial sont écrites dans la RAM de la manière suivante :

J

+1

+1

+n

+n

Avec « I » la position selon x du premier pixel formant le contour initial, et « J » la position selon y du premier pixel formant le contour initial.

Alors pour détecter la position selon x il faut lire la RAM avec un pas d'adresse de 1 et pour lire la position selon y il faut lire la RAM avec un pas d'adresse de 2 ; pour cela on a fait une description en VHDL assurant cette adressage, voir la figure 4.5.

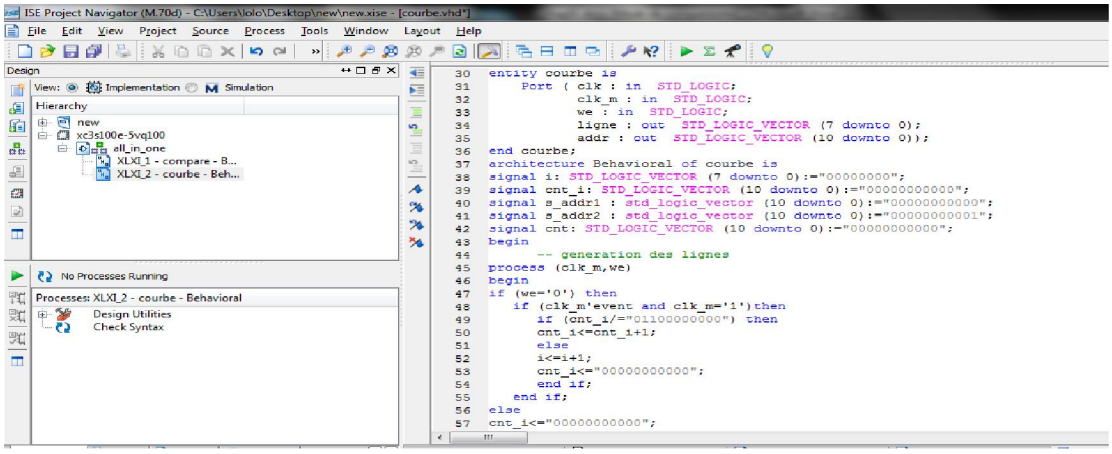


Figure 4.5 description VHDL d'adressage de la RAM de sauvegarde du contour initial

Les sorties de ce module vont être les entrées d'un autre module décrit en VHDL permettant de la position lu de la RAM avec la position du pixel arrivé de l'image et d'indiquer si on est dans la présence un pixel formant le contour initial ; la figure 4.6 illustre la description VHDL de ce module.

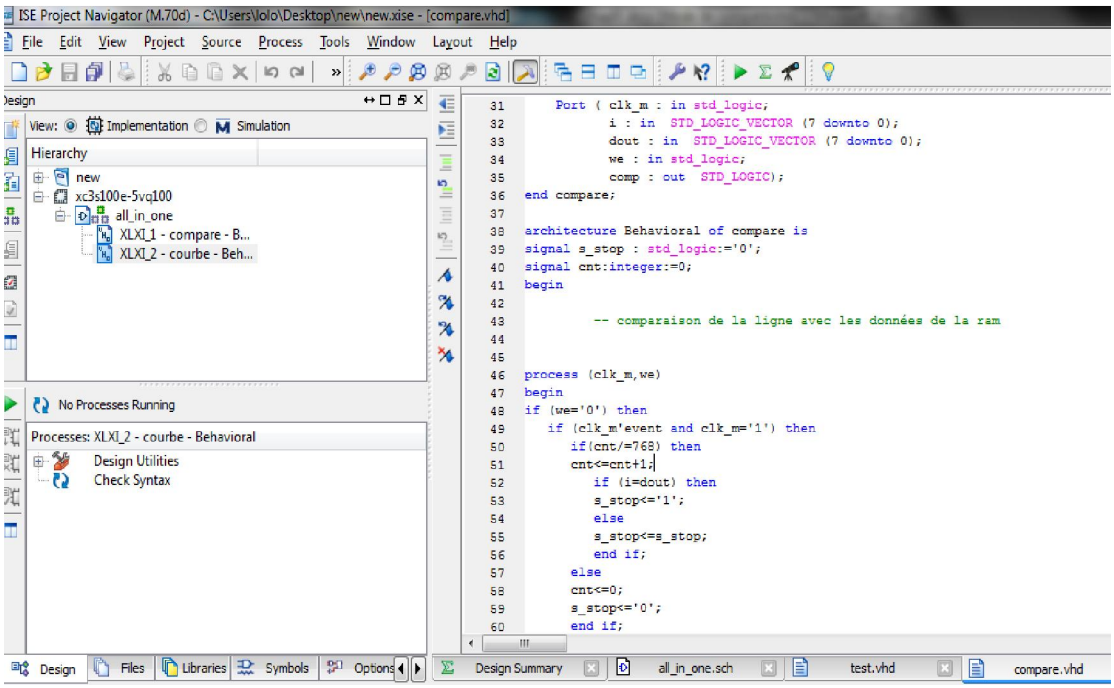


Figure 4.6 description VHDL du comparateur

Pour la lecture et l'écriture dans la RAM on a utilisé deux fréquences différentes afin de lire les positions à partir de la RAM plus rapidement au début de chaque ligne de l'image ; pour cela on a utilisé un DCM (Digital Clock Manager) qui nous permet d'avoir une fréquence multiplier, et un deuxième DCM qui nous permet d'avoir une horloge déphasé de 90° afin de comparer les positions de la RAM avec les pixels de l'image pour qu'il n'y aura pas d'erreur.

On a rassemblé les deux modules décrit en VHDL avec la RAM et les deux DCM en schématique, voir la figure 4.7.

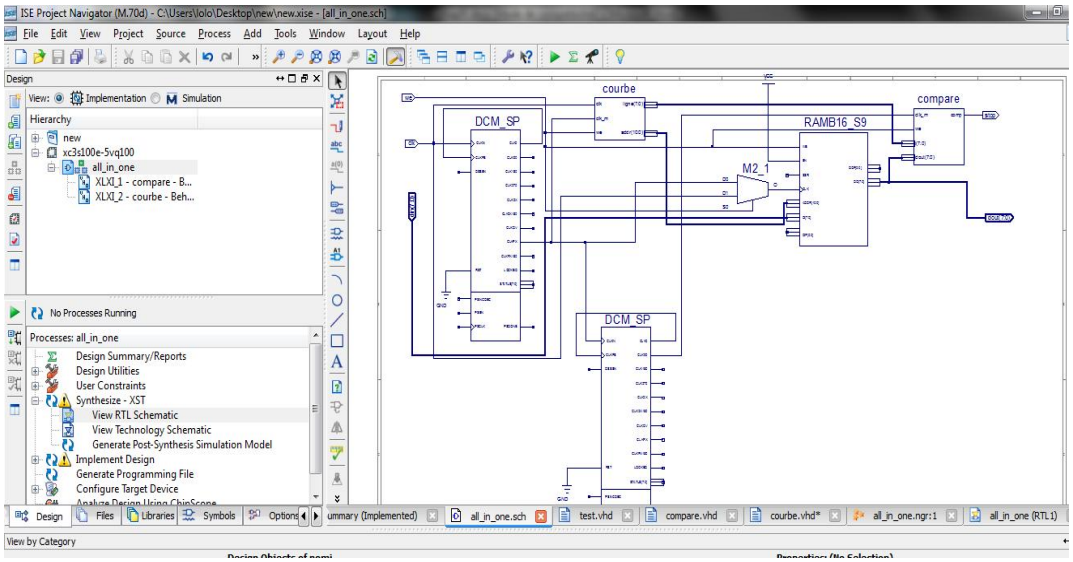


Figure 4.7 schématique du module de détection des pixels du contour initial

4.2.8. Calcul des énergies

4.2.8.1. Calcul de l'énergie externe

$E_{exter} = w_{ligne}E_{ligne} + w_{contour}E_{contour} + w_{term}E_{term}$

L'énergie ligne est tout simplement l'intensité du pixel

$E_{ligne} = I(x, y)$

L'énergie contour est l'opposé du gradient d'intensité élevé au carrée

$E_{contour} = -|\nabla I(x, y)|^2$

Le calcul du gradient d'intensité a été détaillé dans le chapitre III section 2.2

L'énergie terminaison :

$$E_{term} = \frac{C_{xx}C_y^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{(C_x^2+C_y^2)^{3/2}}$$

Cette énergie est composé de cinq variable ($C_x, C_y, C_{xx}, C_{yy} et C_{xy}$), chaque variable est calculé séparément et en même temps que les autre afin de gagner du temps et d'avoir un résultat en temps réel.

Chaque variable est obtenu par un produit de convolution de l'image avec un masque comme suit :

$$C_x = [-1 \quad 1] \otimes I(x,y)$$

$$C_y = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \otimes I(x,y)$$

$$C_{xx} = [1 \quad -2 \quad 1] \otimes I(x,y)$$

$$C_{yy} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \otimes I(x,y)$$

$$C_{xy} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \otimes I(x,y)$$

- Pour obtenir la variable C_x en temps réel on sauvegarde toujours un pixel en utilisant un latch, le résultat est obtenu après un top d'horloge.

Le produit de convolution 1D pour obtenir C_x ce fait de la manière présenté dans la figure 4.8:

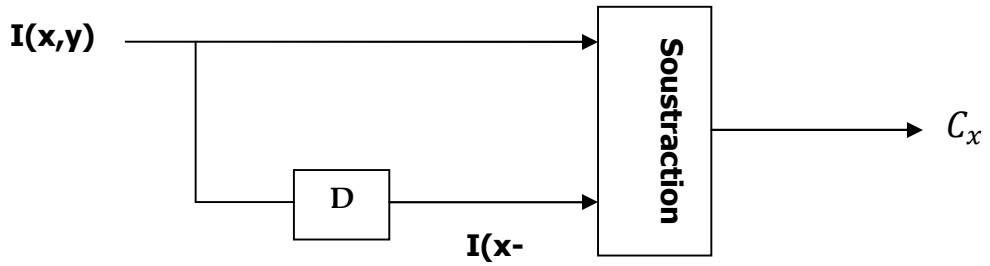


Figure 4.8 schéma synoptique d'une convolution 1D selon x avec deux coefficients

- Pour obtenir la variable C_{xx} en temps réel on sauvegarde toujours deux pixels en utilisant des latch, le résultat est obtenu après deux tops d'horloge.

Le produit de convolution 1D pour obtenir C_{xx} ce fait de la manière présenté dans la figure 4.9:

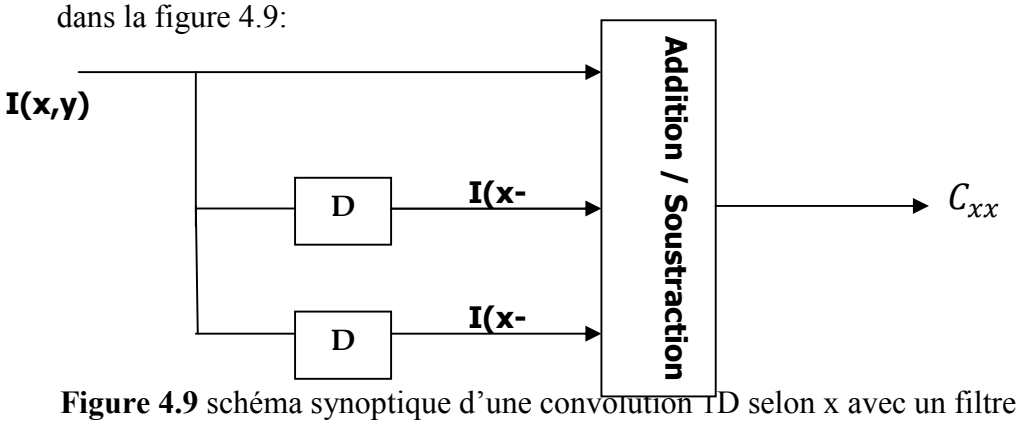


Figure 4.9 schéma synoptique d'une convolution 1D selon x avec un filtre de trois coefficients

Le produit de convolution 1D pour obtenir C_y ce fait de la manière présenté dans la figure 4.10:

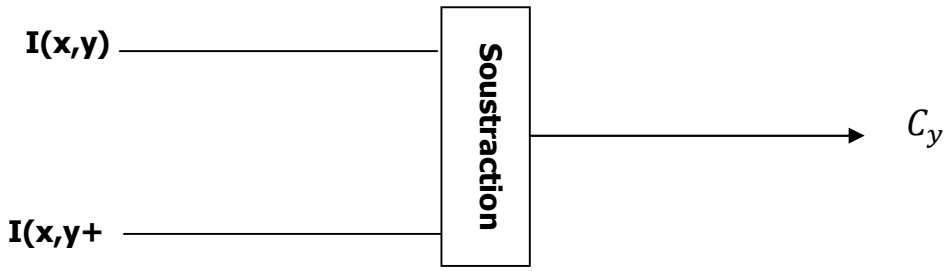
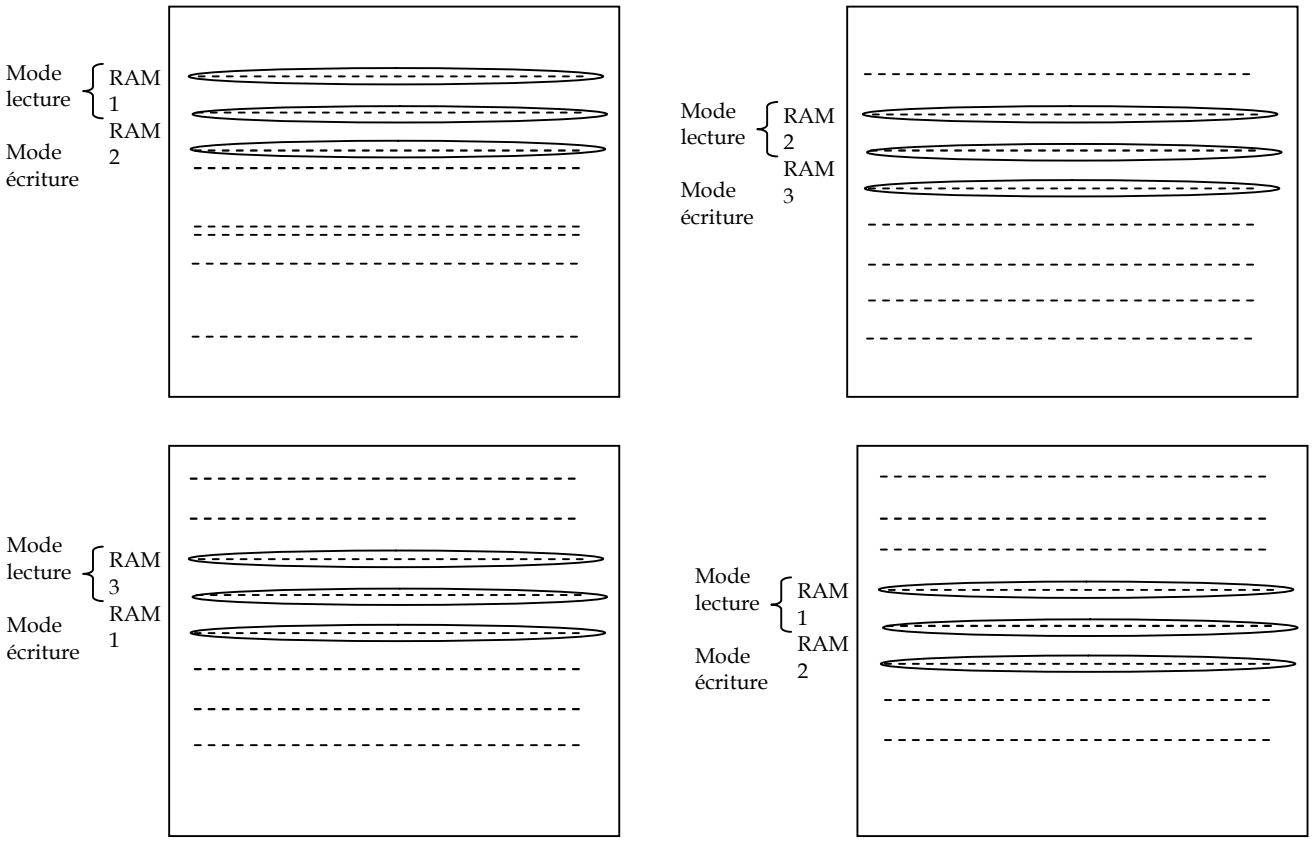


Figure 4.10 schéma synoptique d'une convolution 1D selon y avec un filtre de deux coefficients

On a utilisé dans ce module trois RAM bloque afin de sauvegarder au minimum trois lignes, les deux première RAM en mode lecture pour faire la convolution et la troisième en mode écriture pour l'utilisé lors de la prochaine convolution, la figure illustre le fonctionnement de cette architecture.



- Le produit de convolution 1D pour obtenir C_{yy} ce fait de la manière présenté dans la figure 4.11:

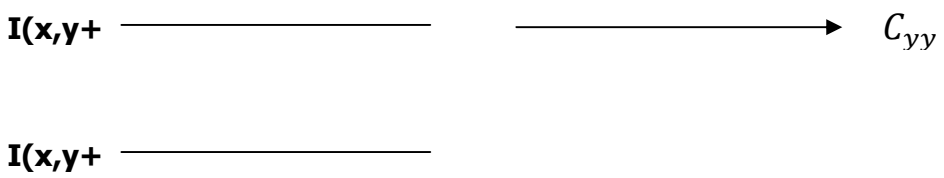


Figure 4.11 schéma synoptique d'une convolution 1D selon y avec un filtre de trois coefficients

La description VHDL de ce module est presque similaire à la précédente sauf qu'il y a une autre RAM bloque de plus dans ce module afin d'avoir trois RAM en mode lecture pour faire la convolution et une RAM en mode écriture.

- Le produit de convolution 2D pour obtenir C_{xy} ce fait de la manière présenté dans la figure 4.12:

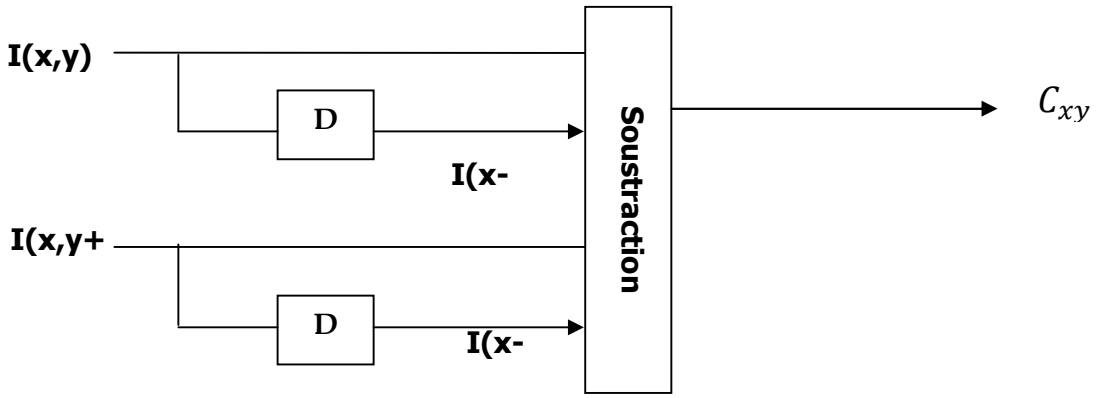


Figure 4.12 schéma synoptique d'une convolution 2D selon y avec un filtre de quatre coefficients

Dans la description VHDL de ce module on a utilisé trois RAM bloque, deux en mode lecture pour faire la convolution et une en mode écriture.

Pour calculer le numérateur il suffit de multiplier et d'additionné le résultat obtenue des cinq variables sauf que un problème est apparue ?

Pour obtenir le premier résultat de la variable $C_x$il faut sauvegarder au moins un pixel alors le résultat est obtenue après un top d'horloge.

Pour obtenir le premier résultat de la variable C_{xx}il faut sauvegarder au moins deux pixels alors le résultat est obtenue après deux tops d'horloge.

Pour obtenir le premier résultat de la variable $C_y$il faut sauvegarder au moins une ligne et un pixel alors le résultat est obtenu après n+1 tops d'horloge (n représente la largeur de l'image).

Pour obtenir le premier résultat de la variable C_{yy}il faut sauvegarder au moins deux lignes et un pixel alors le résultat est obtenu après 2n+1 tops d'horloge (n représente la largeur de l'image).

Pour obtenir le premier résultat de la variable C_{xy}il faut sauvegarder au moins deux lignes et deus pixels alors le résultat est obtenu après 2n+2 tops d'horloge (n représente la largeur de l'image).

Pour résoudre ce problème on a réalisé un module de ligne à retard permettant de retardé $C_x, C_{xx}, C_y\ et\ C_{yy}$ afin d'avoir le résultat en même temps queC_{xy}, ce module a été réalisé en schématique sous l'environnement de développement xilinx ise 12.3 avec des registre a décalage LUT existant déjà dans la bibliothèque, la figure 4.13 présente le schématique de la ligne à retard utilisé.

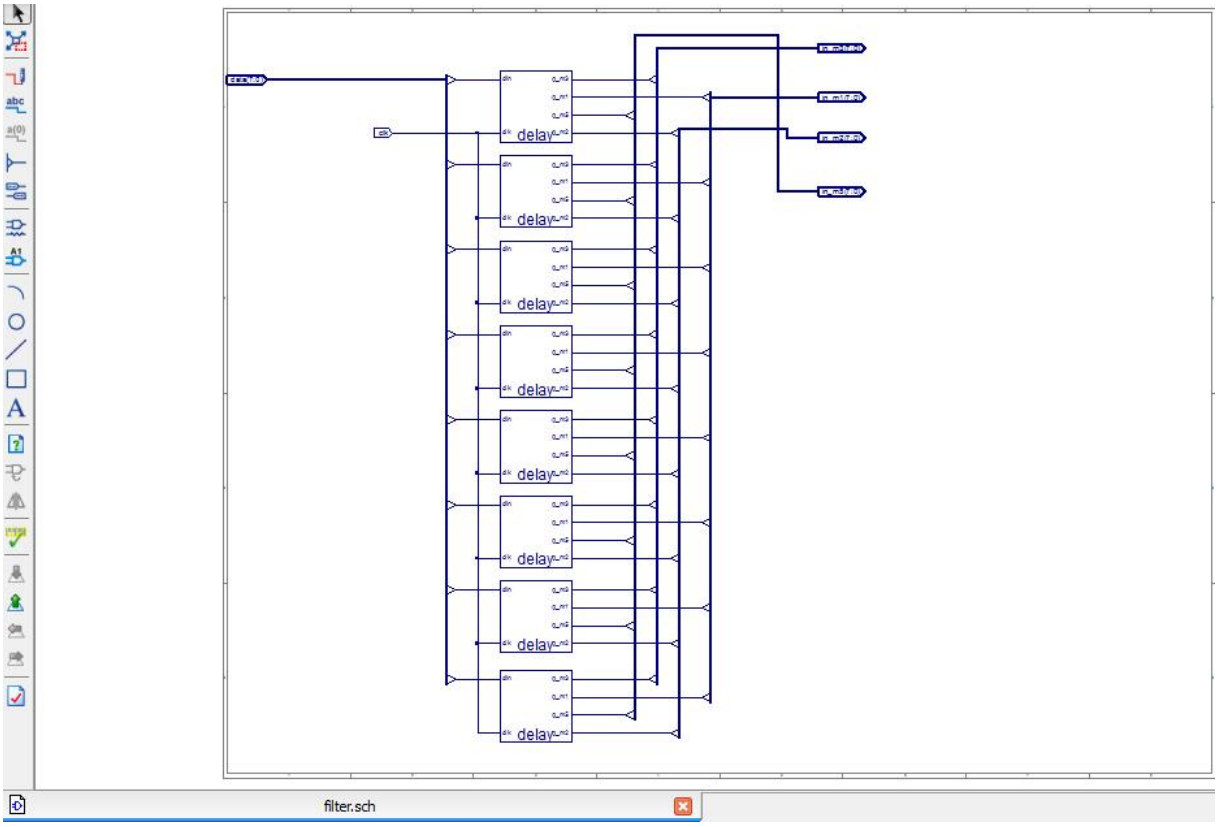


Figure 4.13 Schématique de la ligne à retard

Après cet étage il suffit de calculer le numérateur avec des multiplications et additions des cinq variables pour cela on a créé un module assurant le calcul du numérateur.

Pour compléter le calcul de E_{term} on a calculé le numérateur et il reste le dénominateur.

Puisque l'implémentation d'une opération de division sur FPGA est une tâche difficile et pend trop de temps on a opté d'implémenter le résultat de $\frac{2^{26}}{(C_x^2+C_y^2)^{3/2}}$ sur une ROM afin de ne pas perdre le temps de calcul et d'économiser les ressources de notre FPGA.

On a multiplié par 2^{26} pour agrandir l'ordre du résultat.

Pour cela on calculer tous les cas possible sur matlab et on les a sauvegardé dans un fichier texte ensuite on a réalisé un mini programme en c++ permettant d'utiliser les données inscrites sur ce fichier et créer un autre fichier contenant la description VHDL de la ROM.

Pour avoir le résultat E_{term} il faut multiplier le résultat du numérateur avec le résultat obtenu de la ROM (sachant que la ligne d'adresse de la ROM est la concaténation de$C_x \; et \; C_y$) et en fin faire le décalage de 26 bit pour faire la division sur2^{26}.

Après avoir calculéE_{ligne}, $E_{contour}$ et E_{term} on a fixé dans notre approche $w_{ligne} = w_{contour} = w_{term} = 1$ et on additionné les énergies afin d'avoir E_{exter}. La figure 4.14 présente le schématique réalisé pour calculé E_{exter}.

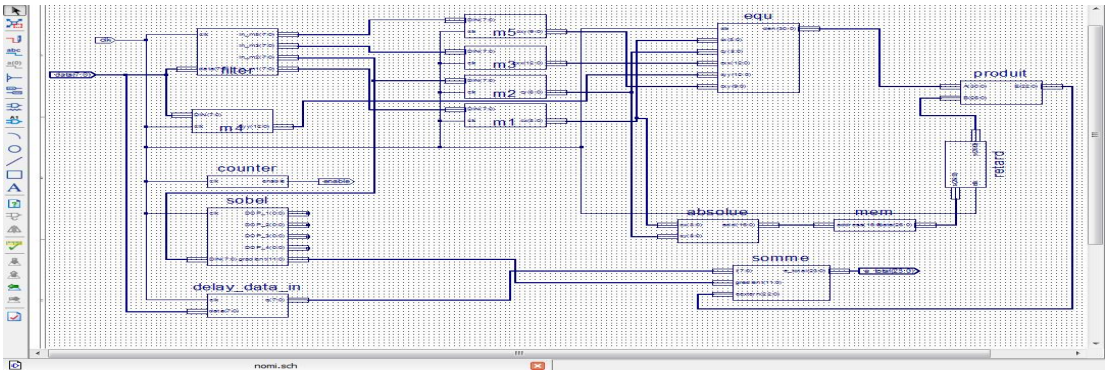


Figure 4.14 Schématique du circuit réalisé pour le calcul deE_{exter}.

4.2.8.2. Energie interne

L'énergie interne correspond à la morphologie et aux caractéristiques de la courbe telles que la courbure, la longueur, etc. donc pour la calculer on aura besoin des données sur la courbe entière à chaque moment d'évolution de la courbe or que le but de notre travaille été d'accélérer le calcul et de travaillé en temps réel, donc dans notre approche on a supposé l'énergie interne nulle.

4.2.9. Observation des pixels

Comme on a détaillé déjà précédemment dans la section 4.2.7, le module de sauvegarde des pixels permet aussi de détecter la présence d'un pixel formant le contour initial.

Dans le cas de la présence d'un pixel formant le contour, il faut comparer l'énergie de ce pixel avec celle des huit pixels de voisinage, et déplacer la position de ce pixel vers le pixel qui a l'énergie minimale.

A la fin de chaque itération on doit sauvegarder les nouveaux positions des pixels formant le contour, pour cela on a proposé d'utiliser une autre RAM bloque qui a le

même rôle que celle utiliser dans la section 4.2.7 et on va les utiliser de la manière suivante :

- Les positions du contour initial sont sauvegardées dans la RAM_1.
- Les positions du contour après la 1ère itération sont sauvegardées dans la RAM_2.
- Les positions du contour après la 2ème itération sont sauvegardées dans la RAM_1.
- Les positions du contour après la n-1 itération sont sauvegardées dans la RAM_2.
- Les positions du contour après la nième itération sont sauvegardées dans la RAM_1.

4.2.10. Vérification de la condition d'arrêt

Afin que l'algorithme s'arrête on vérifie la condition d'arrêt qui est le nombre d'itérations limité par l'utilisateur et on écrit le contenue de la RAM (les positions des pixels formant le contour) dans un fichier texte; après on lit ce dernier avec le matlab et on affiche le contour afin de vérifier s'il a convergé ou non.

4.3. Conclusion

Dans ce chapitre nous avons présenté le principe de fonctionnement des contours actifs, et nous avons détaillé l'architecture développée pour l'implémentation ainsi que les différents composants (avec leur description VHDL) que nous avons conçus et utilisées pour l'implémentation. Dans le chapitre suivant nous allons présenter les résultats de simulations obtenues et nous les commenterons.

5.1. Introduction

Pour implémenter un circuit sur un FPGA il faut qu'on dispose une description logique du circuit (schématique, diagramme d'état ou d'une description VHDL), et aussi d'un environnement de développement qui est choisi en fonction de composant sur lequel le circuit sera implémenté spécialement le logiciel du fabricant (ISE).

Des partie des descriptions VHDL ont été présenté dans le chapitre précèdent.

Dans ce chapitre, il sera présenté l'environnement de développement de XILINX et les résultats de l'implémentation et de simulation.

5.2. Présentation de l'environnement de développement XILINX ISE

C'est le logiciel de programmation produit par XILINX (CPLD, FPGA, Spartan et Virtex…) téléchargeable gratuitement sur le site internet de XILINX (dans sa version web pack). Il intègre différents outils permettant de passer à travers le flot de conception d'un système numérique. Il dispose de :

- Un éditeur de textes, de schémas et diagrammes d'états.
- D'un compilateur VHDL et verilog.
- D'un simulateur.
- D'outils pour la gestion des contraintes temporelles.
- D'outils pour la synthèse.
- D'outils pour la vérification.
- D'outils pour l'implémentation sur FPGA.

C'est un outil de développement complet pour toutes les gammes de produits XILINX

5.2.1. Les étapes pour l'implémentation d'une spécification HDL sur un FPGA

Pour implémenter une spécification HDL sur un FPGA, un la plateforme ISE quatre étapes importantes sont tel illustré dans la figure 5.1.

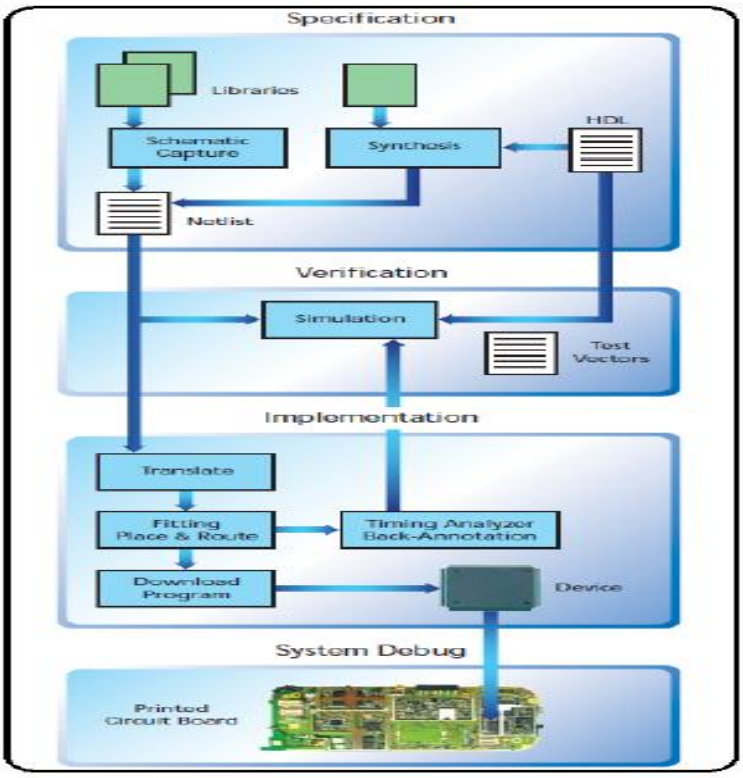


Figure 5.1 Les étapes pour l'implémentation d'une spécification HDL sur un FPGA

Spécification

Le terme de spécification est un terme qui regroupe les trois modes (schématique, diagramme d'état ou HDL) de saisie d'un circuit électronique. La spécification HDL est synthétisé pour générer un fichier appelé NETLIST qui décrit les interconnexions entre les registres.

Vérification

La vérification du design est une étape parallèle ou le concepteur observe le comportement du code et observe s'il se comporte tel qu'il est supposé. Un simulateur simule le circuit à condition de lui fournir les vecteurs de test. Les vecteurs de teste peuvent se présenter sous plusieurs formes, la plus courante est les TESTBENCHS rédigé dans un langage de description matériel pour entrer les instructions au simulateur. En appliquant les vecteurs de test sur le code, le simulateur fournit des sorties du circuit.

Implémentation

Une fois le Netlist (les interconnexions entre les portes logiques) décrit la conception en utilisant les portes logiques en tenant compte du fabricant et d'une famille de composants bien définie. Une fois la vérification est terminée, le circuit est

implémenter sur le composant en spécifiant les références exactes de celui-ci a savoir : le boitier, la fréquence de travail et les autres options spécifiques à chaque composant. Cette étape ce termine par un rapport de tous les sous-programmes exécutés, les warning, les erreurs, les I/O utilisés, des données qui permettent de savoir si le composant choisit et le mieux adapté pour l'application ciblée.

Débuggage du système

Après chargement des interconnexions sur le FPGA, des tests peuvent être effectué sur le circuit implémenté afin de voir les réactions du circuit et son comportement et détecter les anomalies afin de les corriger. En cas d'anomalie la spécification est revue et corrigée en conséquence. Si les tests physique s'avèrent concluants, la spécification est validée et le prototype est considéré opérationnel.

Interface graphique de l'environnement ISE 12.3

L'ISE contrôle tous les aspects d'un design flow. L'interface Project Navigator donne accès à toutes les ressources d'un projet et aux outils de l'implémentation. Elle procure aussi un accès aux fichiers et documents associer au projet (voir figure 5.2).

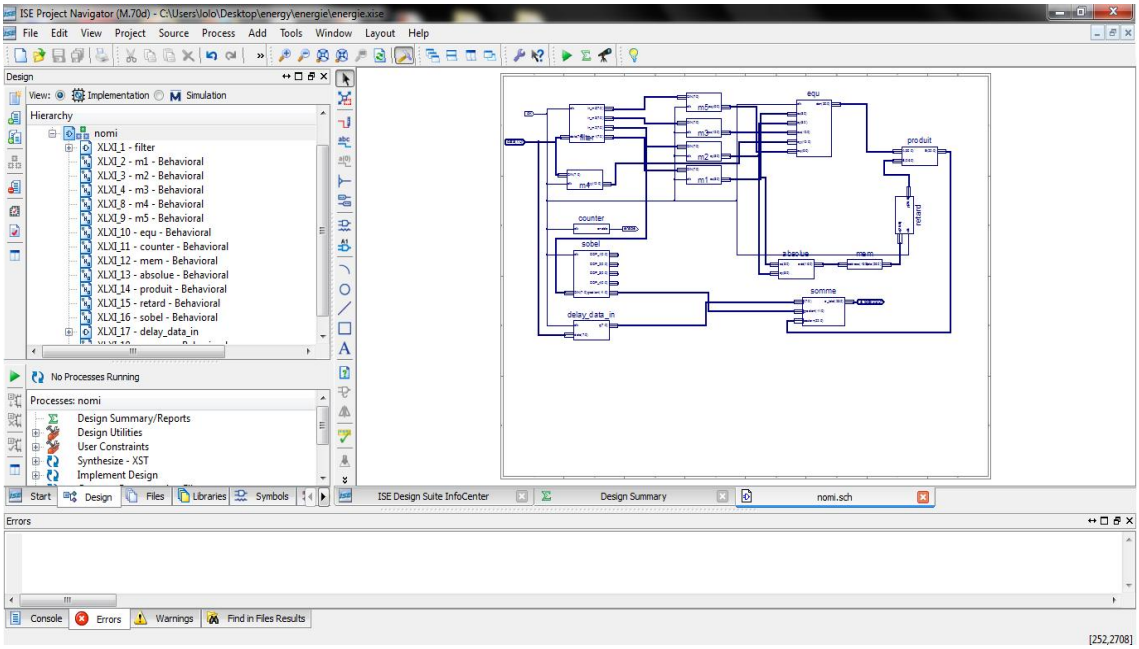


Figure 5.2 Project Navigator de l'environnement de développement ISE 13.2

L'interface du Project Navigator se divise en quatre sous-fenêtres principales :

Source Windows: Elle affiche les éléments inclus dans le projet de façon hiérarchique

Process Windows: affiche les processes disponibles pour la source sélectionnée.

Transcript Windows: affiche les messages d'état, les erreurs et les warning.

MDI (multi documents interface) Windows : c'est l'espace de travail, elle permet de visualiser les rapports HTML, les fichiers textes ASCII, les schématiques et la fenêtre de simulation.

5.3.synthèse et implémentation

L'étape de synthèse reçoit en entrée le code VHDL ou le schématique après vérification et simulation et le fichier de contraintes.

L'étape de synthèse génère quatre fichiers :

5.3.1. Xilinx Specific file au format .NGC

C'est le fichier principal de la synthèse et il comporte les données logiques qui constituent la conception et les contraintes.

5.3.2. RTL (Registre Transfert Level) Shematic

C'est un schéma représentatif de la conception pré-optimisée au niveau RTL. C'est une représentation en symboles génériques tels des additionneurs, multiplexeurs, compteurs ….etc. il est généré à la fin de l'étape de synthèse, il permet d'avoir un aperçu du circuit tout au début du processus d'implémentation.

5.3.3. Technology Shematic

C'est une représentation schématique du fichier NGC, elle figure en termes d'éléments logiques optimisés pour une architecture ou une technologie bien définie. Par exemple, schématique avec des LUTs, buffers d'I/O …etc. Elle est générée après l'étape d'optimisation et la spécification de la technologie utilisée par le processus de synthèse. Ce fichier permet de visualisé la conception au niveau technologique du code VHDL. A ce niveau figure le schéma tel qu'il sera implémenté sur le composant FPGA.

5.3.4. Le fichier LOG

C'est un rapport qui contient les résultats de la synthèse. Il contient toutes les informations relatives au fichier d'entrée, le nom du fichier et la prise en charge des contraintes. Des informations relatives au fichier de sortie, le nom du fichier et son

format NGC. Les messages d'erreurs et warning. Ainsi que les ressources internes nécessaires pour l'implémentation.

5.3.5. Le rapport de synthèse

L'environnement de développement ISE, fournit un rapport de synthèse sous forme de tableaux contenant les informations utiles liées au design.

5.4. Simulation et résultats d'implémentation de l'algorithme des SNAKE

Dans cette section nous allons présenter les schémas représentatifs RTL de chaque code VHDL ou schématique développé lors de la réalisation de notre projet.

Comme il a été détaillé dans le chapitre précèdent, afin d'implémenté l'algorithme des SNAKE on l'a décomposé en plusieurs modules comme suit :

5.4.1. Sauvegarde des positions des pixels du contour

La figure 5.3 représente le schéma RTL mis en place pour sauvegarder les positions des pixels dans une RAM et détecter la présence d'un pixel formant le contour initial.

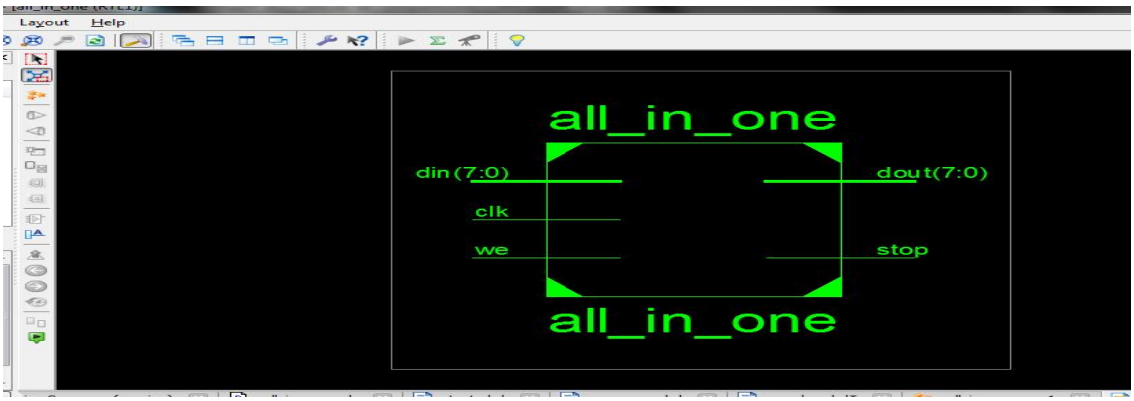


Figure 5.3 Schéma RTL du circuit de détection de contour initial

La figure 5.4 illustre le résultat de la simulation de ce module sous modelsim.

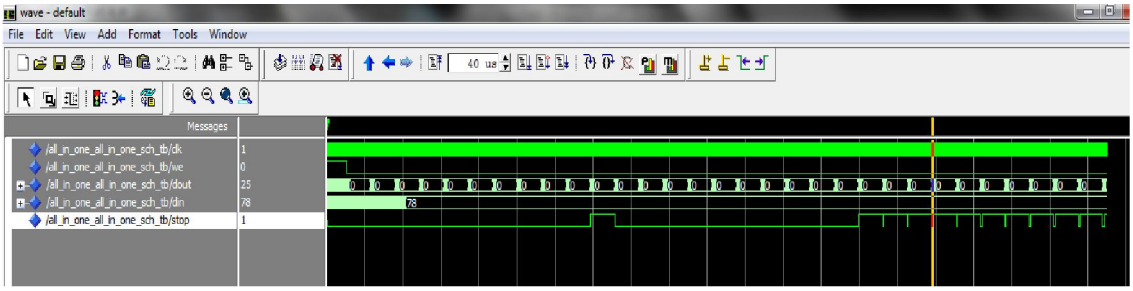


Figure 5.4 résultat de simulation du circuit de détection de contour initial

On remarque qu'à chaque présence d'un pixel formant le contour initial on aura un état haut pendant la durée de cette ligne.

Pour le calcul de l'énergie externe on a utilisé les modules suivants :

Calcul de C_x

La figure 5.5 illustre le schéma RTL du module.

La figure 5.6 illustre le résultat de simulation sous modelsim avec une entrée DIN qui provient d'une image brut de taille 512*512 sauvegardé dans le PC

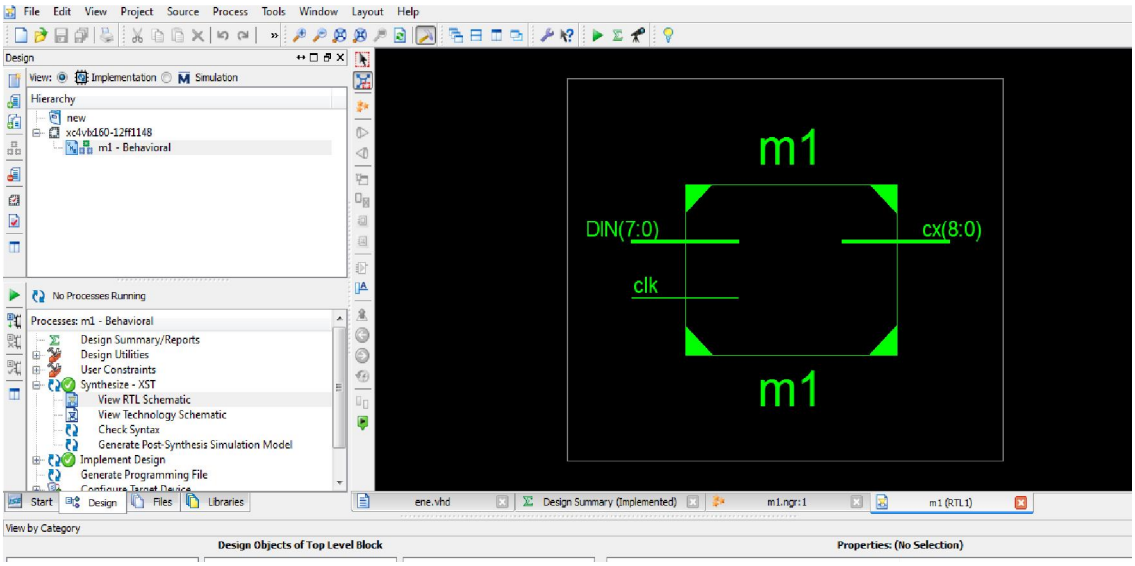


Figure 5.5 Schéma RTL du module de calcul de C_x

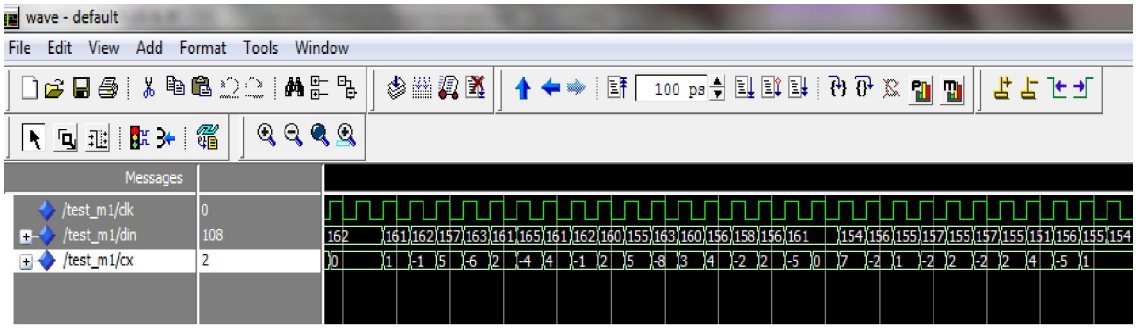


Figure 5.6 résultat de simulation du module de calcul de C_x

Calcul de C_y

La figure 5.7 illustre le schéma RTL du module.

La figure 5.8 illustre le résultat de simulation.

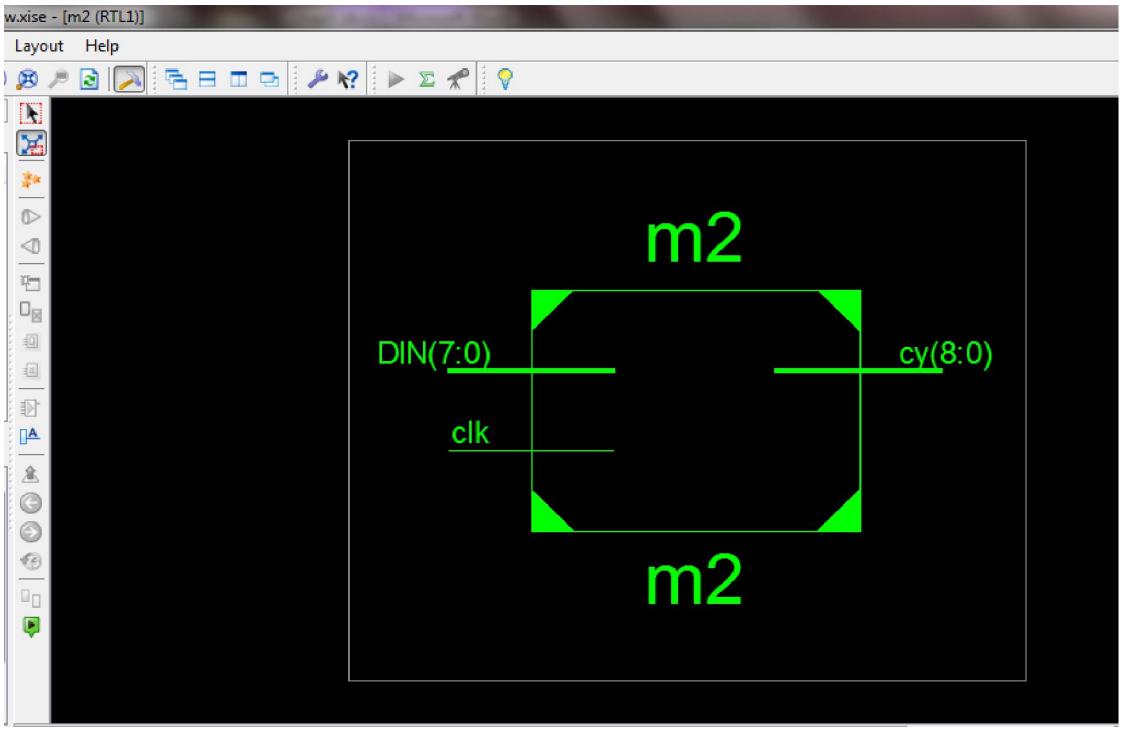


Figure 5.7 Schéma RTL du module de calcul de C_y

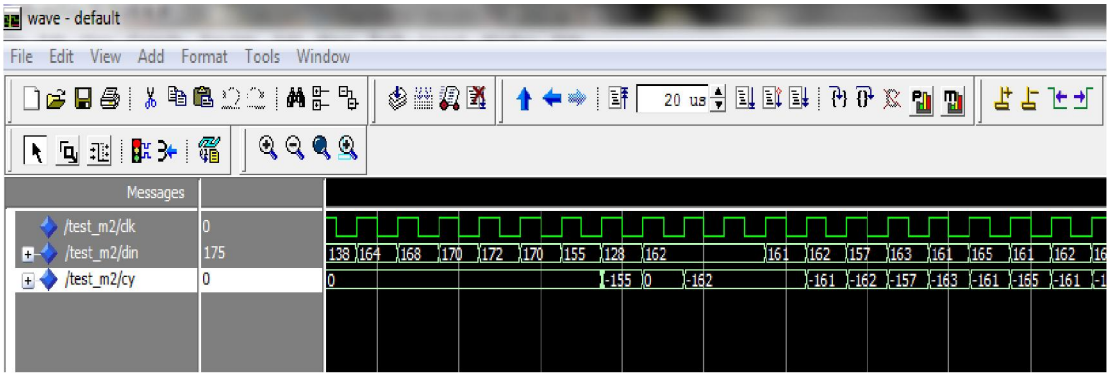


Figure 5.8 résultat de simulation du module de calcul de C_y

Calcul de C_{xx}

La figure 5.9 illustre le schéma RTL du module.

La figure 5.10 illustre le résultat de simulation.

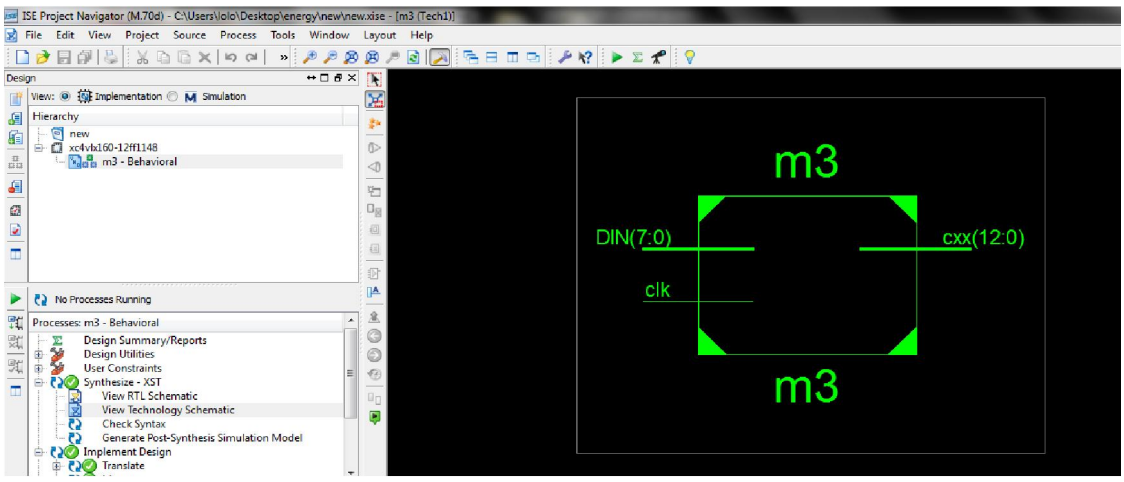


Figure 5.9 Schéma RTL du module de calcul de C_{xx}

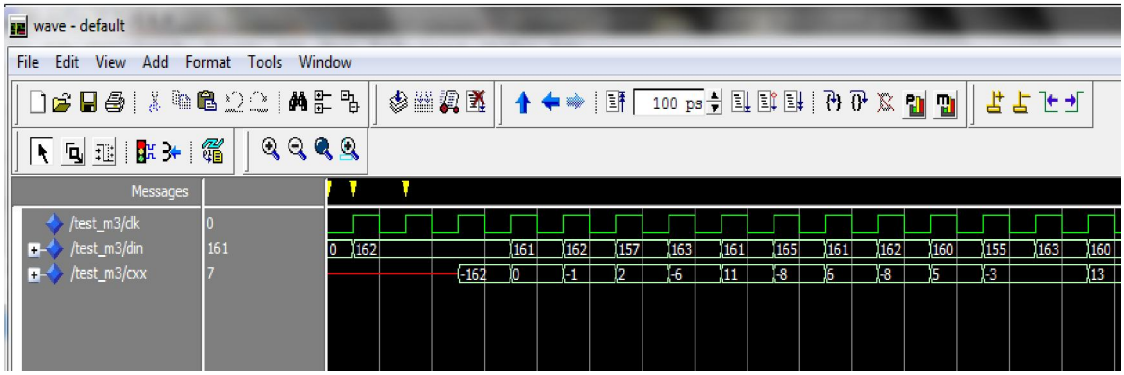


Figure 5.10 résultat de simulation du module de calcul de C_{xx}

Calcul de C_{yy}

La figure 5.11 illustre le schéma RTL du module.

La figure 5.12 illustre le résultat de simulation.

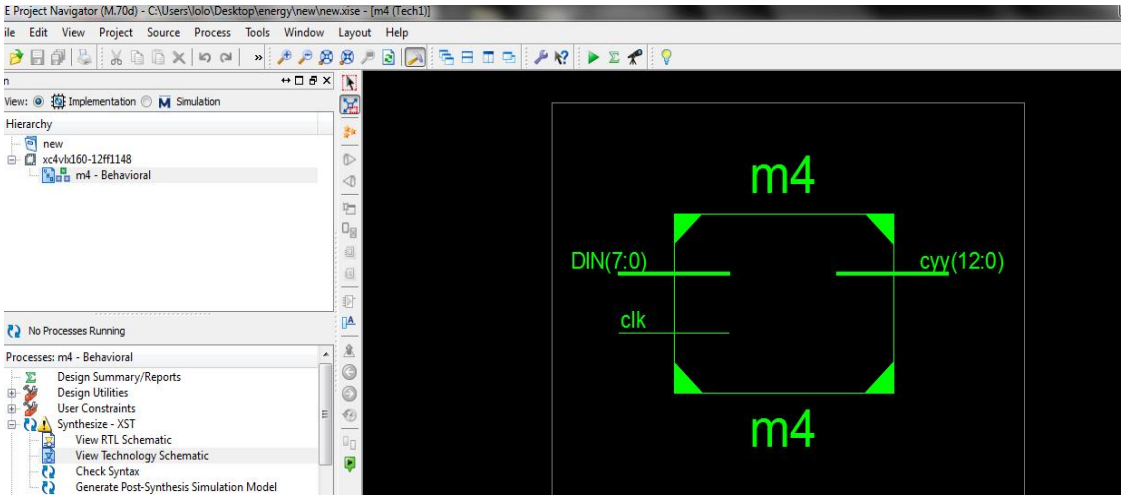


Figure 5.11 Schéma RTL du module de calcul de C_{yy}

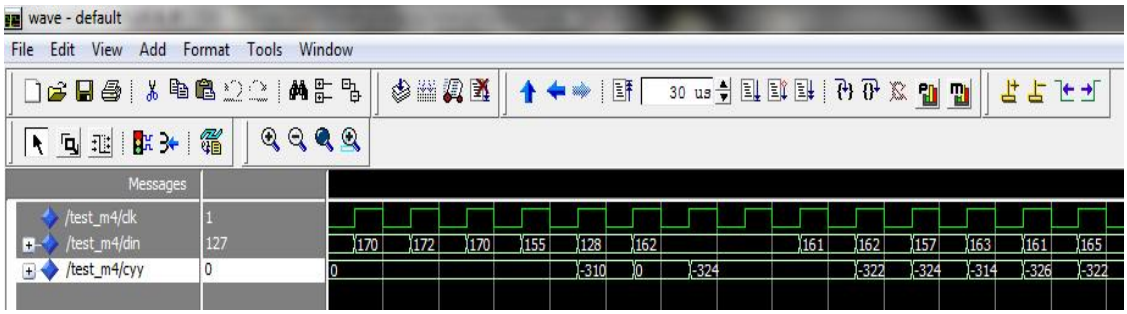


Figure 5.12 résultat de simulation du module de calcul de C_{yy}

Calcul de C_{xy}

La figure 5.13 illustre le schéma RTL du module.

La figure 5.14 illustre le résultat de simulation.

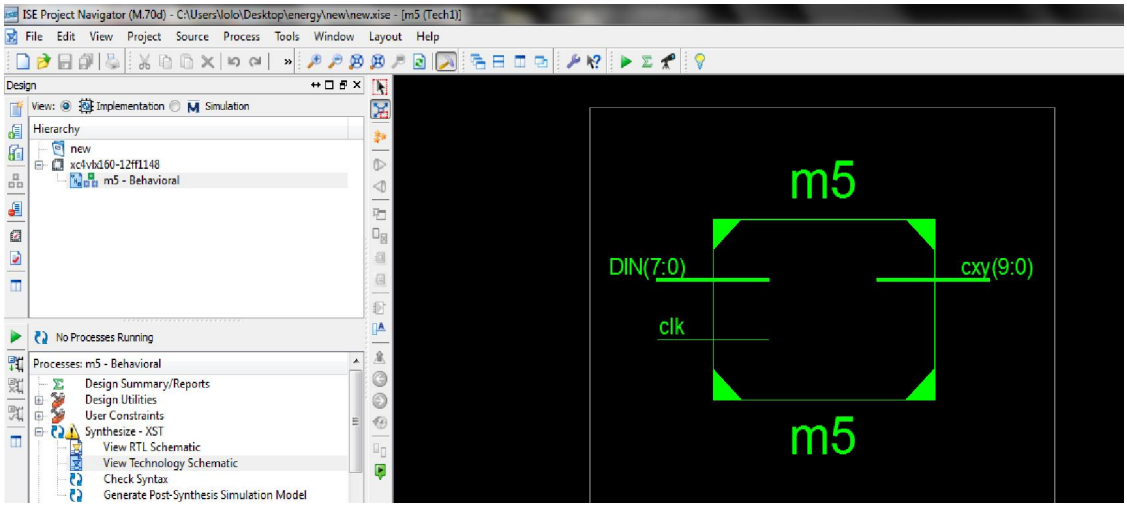


Figure 5.13 Schéma RTL du module de calcul de C_{xy}

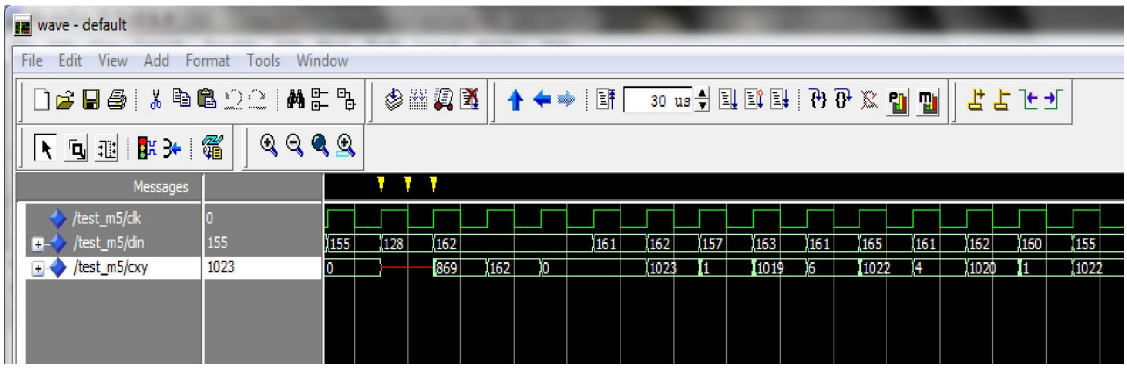


Figure 5.14 résultat de simulation du module de calcul de C_{xy}

Et comme il a été détaillé dans le chapitre précédent on avait besoin d'une ligne à retard pour synchroniser les résultats obtenue par C_x, C_y, C_{xx}, C_{yy} et C_{xy}

La figure 5.15 présente le schéma RTL du module de la ligne à retard qui a été réalisé en schématique.

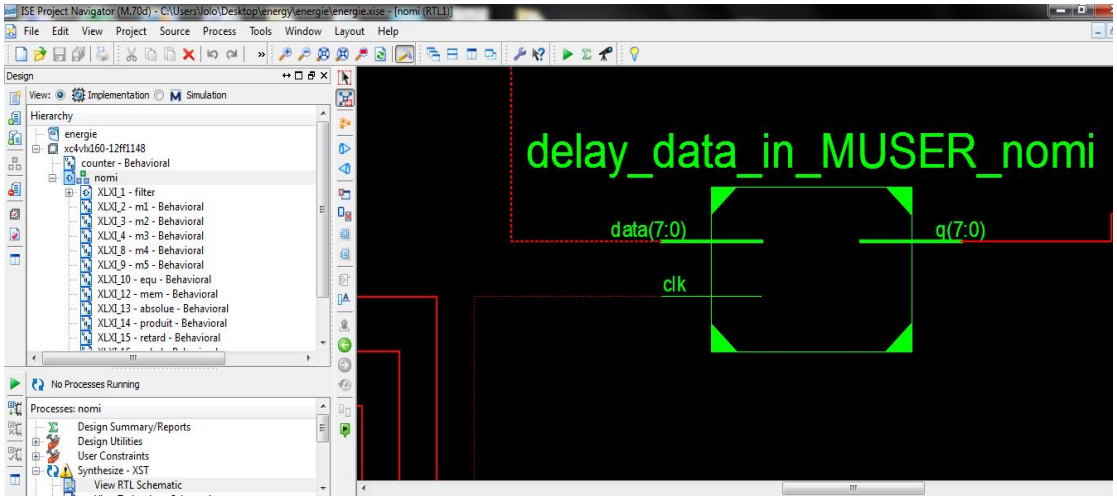


Figure 5.15 Schéma RTL du module de la ligne à retard

La figure 5.16 illustre le schéma RTL de la ROM utilisé pour sauvegarder tous les cas possible du dénominateur.

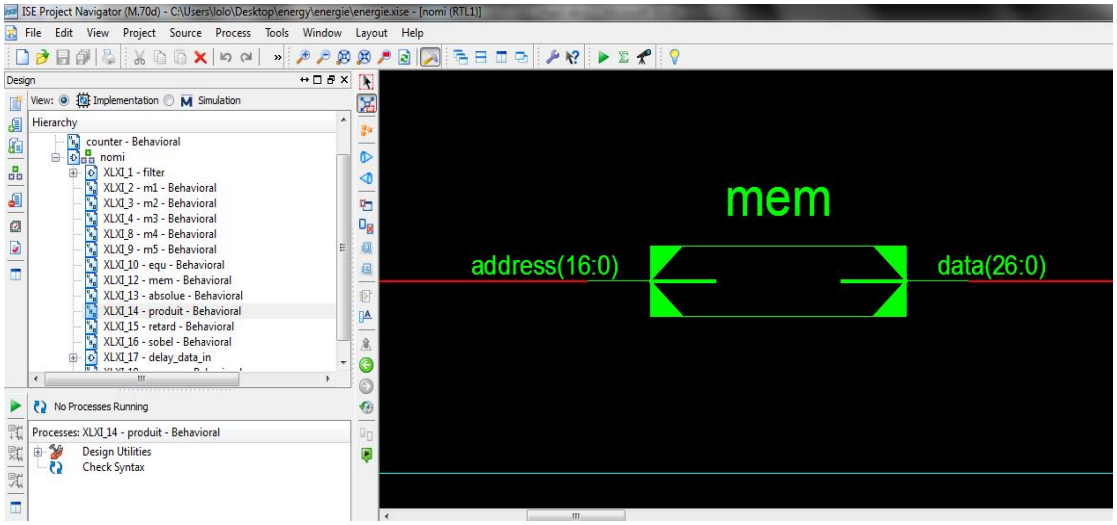


Figure 5.16 Schéma RTL de la ROM

La figure 5.17 illustre le schéma RTL global qui calcul l'énergie terminaison.

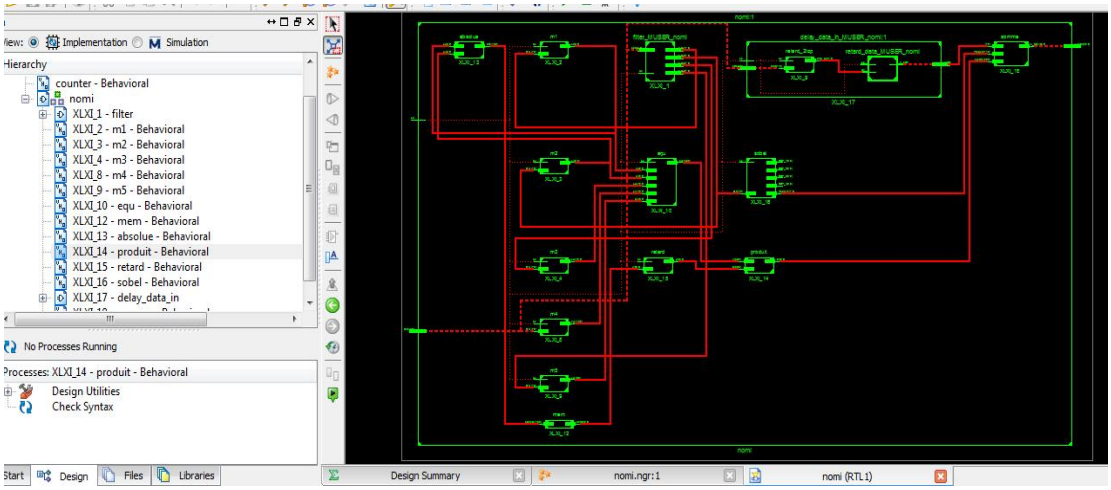


Figure 5.17 schéma RTL global pour le calcul de E_{term}

Pour le calcul de $E_{contour}$ on a réalisé un module composé de trois sous modules :

1- calcul du sobel horizontal

2- calcul du sobel vertical

3- calcul de la somme des valeurs absolue de sobel horizontal et sobel vertical.

La figure 5.18 présente une partie de ce module d'où l'on peut visualiser l'utilisation des RAMs bloque.

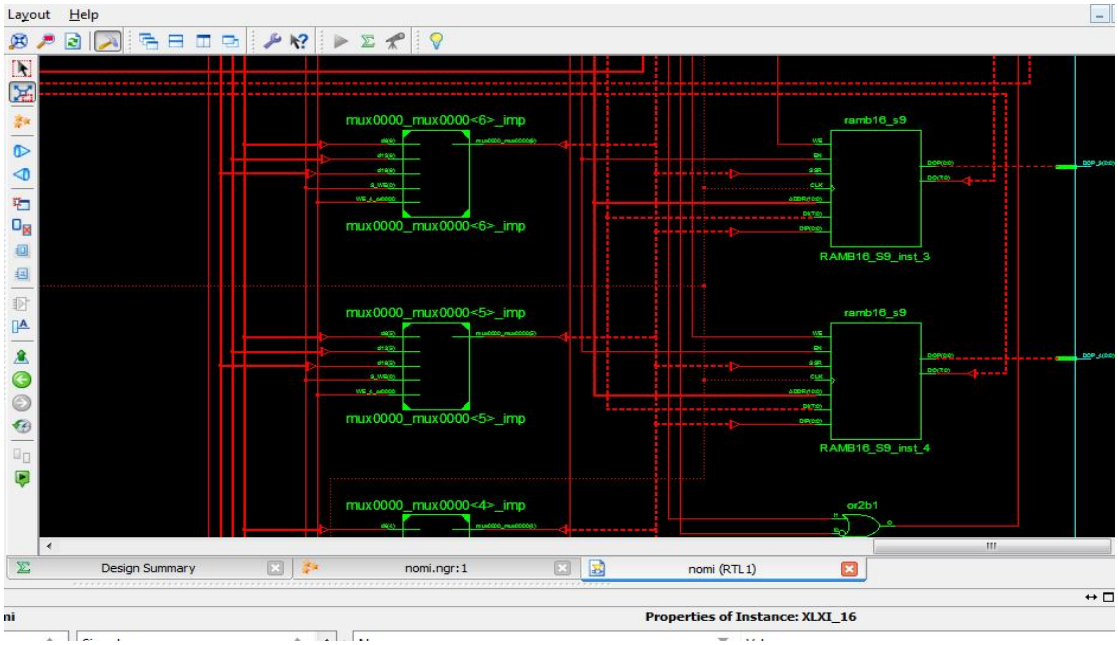


Figure 5.18 schéma RTL technologie du module pour le calcul de gradient

Pour calculer l'énergie externe reste à sommer les trois énergies dont E_{ligne}est l'image elle-même.

Ce dernier module de sommation, on la décrit en VHDL, la figure 5.19 illustre le schéma RTL de ce module.

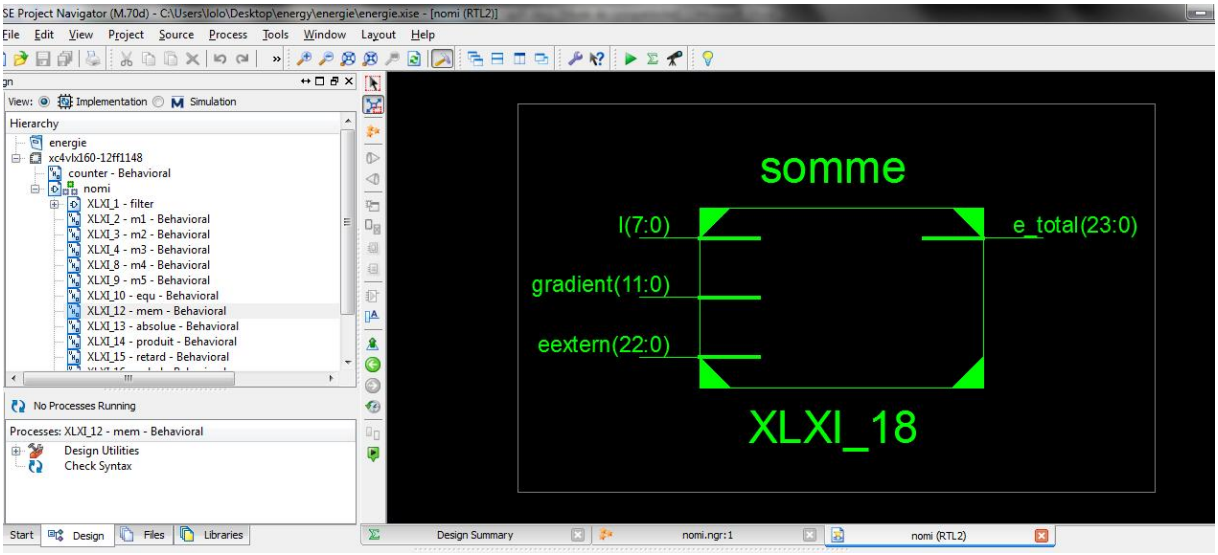


Figure 5.19 Schéma RTL du module de sommation

Tous les résultats obtenus ont été vérifiés et comparés avec les résultats obtenus sous matlab.

Pour la vérification de la condition d'arrêt le module n'a pas été réalisé vue le temps que prend une seul itération dans une simulation sur PC (environs 40 heures sous système d'exploitation linux scientifique et un processeur core2duo et 4GB de mémoire).

5.5.Conclusion

Ce dernier volet de cette étude nous permet de conclure que les résultats obtenus démontrent la justesse de l'architecture proposé et développé, et que cette méthode de conception permet d'économiser les ressources de notre FPGA par conséquent le cout est réduit, de même cette architecture permet d'avoir un contour en temps réel.

La conception et simulation du circuit numérique menée par l'appui d'outils informatiques spécialisé et l'utilisation du langage VHDL comme outil de description pour représenter le comportement et l'architecture du dispositif numérique nous a permis d'obtenir un certain niveau de réutilisabilité des différents blocs de l'architecture.

Conclusion

On a proposé une implémentation hardware pour la détection de contour par l'approche de canny et par l'approche de snake en utilisant une architecture FPGA, cette solution est basée sur l'utilisation de plusieurs modules qui travaillent en parallèle afin de réduire le temps de calcul.

Les travaux menés au cours de ce mémoire constituent l'ensemble des étapes indispensables pour l'implémentation d'un circuit logique sur un FPGA de façon générale et l'implémentation d'un algorithme de traitement d'image en particulier.

Pour parvenir à réussir l'implémentation du circuit, on a suivi la méthode scientifique en commençant par la recherche bibliographique, l'étude des documents et l'assimilation des informations et par-dessus toute l'assimilation de la méthodologie propre à l'implémentation des circuits numériques sur un FPGA. Puis viens la partie expérimentale qui est la réalisation du circuit en langage VHDL, on teste le circuit en le simulant et on résout les erreurs obtenues après la simulation, tenant compte que le temps de la simulation été vraiment important.

En perspective, ce travail peut être poussé encore plus loin dans l'implémentation d'autres algorithmes de détection de contour par approche contour actif sur un traitement de vidéo afin d'aboutir à un suivi d'objet en temps réel.

REFERENCES

[1] Simon HAENE, Andreas BURG, David PERELS, Peter LUETHI, Norbert FELBER et Wolfgang FICHTNER : FPGA implementation of Viterbi decoders for MIMO-BICM. In 39th Asilomar Conference on Signals, Systems & Computers, pages 734–738, PacificGrove, CA, USA, novembre 2005. IEEE Signal Processing Society.

[2] Mihail PETROV et Manfred GLESNER : Optimal FFT architecture selection for OFDM receivers on FPGA. In G. BREBNER, S. CHAKRABORTY et W.-F. WONG, éditeurs :IEEE International Conference on Field-Programmable Technology (FPT'05), pages 313–314, Singapore, décembre 2005. IEEE.

[3] Panan POTIPANTONG, Theerayod WIANGTONG, Phaophak SIRISUK et Apisak WORAPISHET : A scaleable FFT/IFFT kernel for communication systems using codesign approach. In G. BREBNER, S. CHAKRABORTY et W.-F. WONG, éditeurs : IEEE International Conference on Field-

Programmable Technology (FPT'05), pages 329–330, Singapore, décembre 2005. IEEE.

[4] Stéphane GUYETANT, Mathieu GIRAUD, Ludovic L'HOURS, Steven DERRIEN, Stéphane RUBINI, Dominique LAVENIER et Frédéric AIMBAULT : Cluster of reconfigurable nodes for scanning large genomic banks. Parallel Computing, 31(1):73– 96, janvier 2005.

[5] César TORRES-HUITZIL et Bernard GIRAU : FPGA implementation of an excitatory and inhibitory connectionist model for motion perception. In G. BREBNER, S. CHAKRABORTY et W.-F. WONG, éditeurs : IEEE International Conference on Field-Programmable Technology (FPT'05), pages 259–266, Singapore, décembre 2005. IEEE.

[6] Tatsuhiro TACHIBANA, Yoshihiro MURATA, Naoki SHIBATA, Keiichi YASUMOTO et Minoru ITO : General architecture for hardware implementation of genetic algorithm. In J.M. ARNOLD et K.L. POCEK, éditeurs : 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06), pages 291–292, Napa, CA, USA, avril 2006. IEEE Computer Society.

[7] Gerhard LIENHART, Andreas KUGEL et Reinhard MÄNNER : Using floating-point arithmetic on FPGAs to accelerate scientific N-body simulations. In J.M. ARNOLD et K.L. POCEK, éditeurs : 10th Annual IEEE Symposium on Field-Programmable CustomComputing Machines (FCCM'02), pages 182–191, Napa, CA, USA, avril 2002. IEEE Computer Society.

[8] Maya GOKHALE, Janette FRIGO, Christine AHRENS, Justin L. TRIPP et Ron MINNICH : Monte Carlo radiative heat transfer simulation on a reconfigurable computer. In J. BECKER, M. PLATZNER et S. VERNALDE, éditeurs : Field-Programmable Logic and Applications : 14th International Conference, FPL 2004, numéro 3203 de Lecture Notes in Computer Science, pages 94–104, Antwerp, Belgium, septembre 2004. Springer-Verlag.

[9] Yongfeng GU, Tom VANCOURT et Martin C. HERBORT : Integrating FPGA acceleration into the protomol molecular dynamics code : preliminary report. In J.M. ARNOLD et K.L. POCEK, éditeurs : 14th

Annual IEEE Symposium on Field-Programmable CustomComputing Machines (FCCM'06), pages 315–316, Napa, CA, USA, avril 2006. IEEE Computer Society.

[10] KevinWHITTON, X. Sharon HU, Cedric X. YU et Danny Z. CHEN : An FPGA solution for radiation dose calculation. In J.M. ARNOLD et K.L. POCEK, éditeurs : 14th Annual IEEE Symposiumon Field-Programmable CustomComputing Machines (FCCM'06), pages 227–236, Napa, CA, USA, avril 2006. IEEE Computer Society.

[11] Actel Corporation, Mountain View, CA, USA. Axcelerator family FPGAs, décembre 2005.

[12] Xilinx, Inc., San Jose, CA, USA. Virtex-II platform FPGAs : complete data sheet,mars 2005.

[13] Altera Corporation, San Jose, CA, USA. Stratix II device family data sheet, août 2006.

[14] L.G.Robert. *Machine perception of tree-dimensional solids*. In L Clapp C.Koester J. Tippet, D.Berkowitz and A.Vanderburgh. Optical and electro-optical information processing, pages 159-179. MIT press, Cambridge,1965.

[15] J.M.S.Prewitt. *Object enhancement and extraction*. In B.S.Lipkin and A. Rosenfeld editors. Picture processing and psychopictorics, pages 75-149. Academic press, New York.1970.

[16] I.Sobel. *Neighbourhood coding of binary images for last contour following and general array binary processing*. Computer graphics and image processing 8, 127-135 (1978).

[17] J. Canny, *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 8, No. 6, Nov 1986.

[18] J. Shen, S. Castan, *An Optimal Linear Operator for Step. Edge Detection*, IEEE conf Vision Pattern Recogn, pp. 109-114, 1986.

[19] R. Deriche: *Fast algorithms for low-level vision*: IEEE Trans PAMI-12 pp 78–87, 1990.

[20] A. Foulonneau, *Une contribution à l'introduction de contraintes géométriques dans les contours actifs orientés région*, Ph.D. thesis, Université Louis Pasteur - Strasbourg I, December 2004.

[21] S. Jehan-Besson, *Modèles de contours actifs basés régions pour la segmentation d'images et de vidéos*, Ph.D. thesis, Université de Nice-Sophia Antipolis, 2003.

[22] M. Kass, A. Witkin, and D. Terzopoulos, *Snakes : active contour models*, 1st International Conference on Computer Vision, pp. 259-268 (1987).

[23] C. Epstein et M. Cage. *The curve shortening flow*. Wave Motion : Theory, Modeling, and Computation, Springer-Verlag, 1987.