République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE POLYTECHNIQUE

Département de Génie Electrique

Mémoire de fin d'étude

En vue de l'obtention du diplôme d'Ingénieur d'Etat en Automatique

Thème :

Conception et implémentation d'un algorithme de vision artificielle sur FPGA

Présenté par :

M. AIB Abdelghani

Proposé et dirigé par :

M. C. LARBES (ENP)

Juin 2008

(بسم الله الرحمان الرحيم)

الملخص:

من أجل التحكم في ذراع آلي بالزمن الحقيقي مختلف مراحل الانجاز و التطبيق على دارة FPGA uH لأحدى خوارزميات الرؤية الاصطناعية التي تعتمد أساسا على ثوابت مفصلة في رسالة التخرج هاته.

كلمات مفتاحية: دارة FPGA ، بطاقة RC203E، لغة Handel-C، الرؤية الاصطناعية، ثوابت Hu.

Résume :

En vu de la commande d'un bras manipulateur en temps réel, les différents étapes de conception et d'implémentation sur FPGA, d'un algorithme de vision artificielle, basé sur les invariants de Hu, sont présentées dans ce mémoire.

Mots clés : Circuit FPGA, RC203E, langage Handel-C, vision artificielle, Hu.

Abstract :

To command a manipulator arm in real time, the various steps of conception and implantation on FPGA of artificial vision algorithm, based on the Hu invariants, are presented in this thesis.

Keywords: FPGA circuit, RC203E, Handel-C language, artificial vision,Hu.

Remerciements

Quelques lignes ne pourront jamais exprimer la reconnaissance que nous éprouvons envers tous ceux qui, de pris ou de loin, ont contribué, par leurs conseils, leurs encouragements ou leurs amitiés à l'aboutissement de ce travail.

Mes vifs remerciements accompagnés de toute ma gratitude vont tout d'abord à mon promoteur M. C. LARBES Pour m'avoir proposé ce sujet, pour les conseils qu'il n'a cessé de me prodiguer et surtout pour la confiance qu'elle m'a accordé pour la réalisation de ce projet.

Ma reconnaissance va à tous mes enseignants à département de l'automatique et à l'ENP, en général.

Dédicaces

Arrivé à ce stade, n'est que le fruit du milieu familial qui m'est propice, le fruit de l'équilibre et du sacrifice de mes Parents, Ces êtres chers que Dieu les protège.

Qu'il m'est agréable en ce moment de partager ce bonheur avec eux.

Je dédie ce modeste travail à mes chers Parents qui ont été de tout temps, les plus proches , qui n'ont jamais ménagé leurs efforts, leurs encouragements et leur soutien avec abnégation et patience

Sans oublier mes Grands Parents.

A mon frère et mes sœurs.

A mes Oncles, Tantes et cousins et alliés de la famille.

En ce moment je ne peux oublier.

A l'ensemble des amis que j'ai connu pendant mes études et à ceux qui ont prodigué leurs vifs conseils, encouragements et témoigné de leur amitié.

Abdelghani

Sommaire

Chapitre 2 Les circuits FPGA (Field Programmable Gate Array)

Chapitre 3 Implémentation software d'une procédure de reconnaissance

Chapitre 4 Implémentation hardware sur FPGA d'une architecture de reconnaissance

Listes des figures et tableaux

Les figures

Les tableaux

Introduction générale

Introduction générale

La vision stéréoscopique humaine permet à l'homme de se déplacer et d'appréhender son environnement : détecter un objet, reconnaître un visage dans une scène et ressentir les émotions des gens, perceptibles à travers leur expressions, sont quelques exemples de fonctions dévolues à la vision et quotidiennement exécutées de manière implicite. Bien que celles-ci soient réalisées avec un minimum d'effort, elles traduisent lorsqu'on tente de les analyser l'existence d'un ensemble de processus extrêmement complexes.

Dans les trois dernières décennies, les chercheurs ont essayé de doter de capacité de vision les systèmes automatisés tels les robots afin de les rendre plus intelligents et d'élargir ainsi l'éventail de leurs capacités. Cependant beaucoup de chemin reste à faire : la façon dont le cerveau humain traite l'information visuelle est très mal connue et les tâches de traitement d'image couramment exécutées par les systèmes automatisés sont très lourdes en temps calcul. En développant les systèmes de vision, les concepteurs sont essentiellement confrontés à deux possibilités : implémenter les algorithmes de vision sous forme logicielle et les exécuter sur des processeurs standards (ou des DSP) ou les implémenter dans des architectures matérielles spécialement conçues pour l'application. Bien que les architectures matérielles spécifiques puissent offrir des vitesses de traitement beaucoup plus élevées et puissent exploiter beaucoup plus facilement le parallélisme que l'on rencontre fréquemment dans les algorithmes de traitement d'images, celles-ci sont restées longtemps ignorées car les ressources nécessaires ne sont pas toujours disponibles à un prix raisonnable et le délai de conception est relativement long. Ces différents aspects ont contribué au choix fréquent de la première option, à savoir le développement d'algorithmes tournant le plus rapidement possible sur des processeurs standards.

Depuis quelques années, une troisième solution pour les concepteurs de systèmes de vision a vu le jour du fait de la croissance très rapide des circuits logiques programmables en termes de vitesse, de ressources disponibles et de performances des outils de développement. Les circuits logiques programmables ont également été utilisés avec succès dans des applications autres que celles liées au traitement d'images, telle que le filtrage numérique multivoies, la FFT, la formation de voies dans les antennes acoustiques large bandes. Ces circuits permettent aux développeurs de configurer économiquement et rapidement leur

structure interne en fonction des caractéristiques de l'algorithme de traitement choisi car ils éliminent les phases les plus lourdes et les plus coûteuses que l'on rencontre lors de la conception de circuits intégrés spécifiques tels les ASIC, à savoir leur fabrication. Ils permettent également une réduction significative des temps de mise au point puisqu'il est très facile et très rapide pour le concepteur de modifier le design, le recompiler et reprogrammer le composant.

Le but des travaux présentés dans ce mémoire est d'évaluer les performances des circuits FPGA pour les applications de robotique et de vision temps réel basées sur le principe de la reconnaissance des formes par la méthode des moments.

Le mémoire présenté est composé de quatre chapitres :

Chapitre 1 : Il aborde les différentes notions et définitions de base utilisées en traitement d'images ainsi que quelques méthodes d'extractions des paramètres utilisées dans la reconnaissance des images.

Chapitre 2 : Il présente les circuits logiques programmables avec ses différentes classes et en particulier les circuits FPGA, leur architecture et leur principe de fonctionnement ainsi que les étapes de leur programmation.

Chapitre 3 : Ce chapitre est dédie à l'implémentation software d'une procédure de reconnaissance basée sur les invariants de Hu, sous Borland C++ Builder6.

Chapitre 4 : Dans ce dernier chapitre, l'implémentation hardware d'une architecture de reconnaissance basée sur les invariants de Hu sous la carte RC203E de Celoxica, est présentées.

Enfin nous terminons par une conclusion générale.

Introduction à la vision artificielle et reconnaissance des formes

1 Définitions et concepts sur traitement des images [1]

1.1 Définition d'une image

L'image est définie comme étant la reproduction exacte, ou la reproduction analogique d'une scène réelle.
Visuellement c'est une représentation bidimensionnelle. Elle contient en chaque point l'intensité lumineuse, en ce point. Donc, l'image peut être décrite sous forme d'une fonction I(x.y) de brillance analogique continue, définie dans un domaine borné tel que x et y sont les coordonnées spatiales d'un point de l'image et I est une fonction d'intensités lumineuses ou de couleurs. Sous cet aspect, l'image est inexploitable par la machine, ce que nécessite sa numérisation.

1.2 Image numérique

L'image fournie par le capteur est analogique. Pour pouvoir la manipuler comme une donnée informatique, il faut avant tout lui faire subir une transformation qui la rendra compréhensible par l'ordinateur. Elle doit être numérisée.
L'image numérique est représentée par une matrice de points appelés pixels, ayant chacun comme caractéristique : un niveau de gris, ou une couleur prélevée à l'emplacement correspondant dans l'image réelle, ou un code calculé à partir d'une description interne de la scène à représenter.
La valeur de chaque pixel représente l'intensité lumineuse au point considéré de l'image. Cette numérisation s'effectue par des amplitudes de l'intensité lumineuse.

1.3 Pixel

Le pixel est la plus petite entité qui compose l'image. Ce mot est très utilisé dans le domaine de traitement d'image. Il est caractérisé par deux attributs :

- sa position dans la matrice d'image (n° de ligne, n° de colonne).
- une valeur numérique indiquant sa couleur, ou son niveau de gris.

1.4 Résolution

La résolution est le nombre de points utilisés pour représenter l'image, ou bien le nombre de pixels disponibles en ligne (X) et en colonne (Y) de l'image.
On peut dire que plus la résolution augmente, plus l'image est fine et plus elle requiert d'espace mémoire pour le stockage.

1.5 Bruit

Un bruit dans une image est un phénomène de brusque variation d'intensité d'un pixel par rapport à ses voisins.
Il est généralement causé par les mauvaises conditions d'éclairage, les dispositifs électroniques du capteur ou par l'image elle même.

1.6 Contour

Un contour est un ensemble de pixels formant une frontière entre deux ou plusieurs régions voisines. L'épaisseur du contour est d'un pixel ou plus (un pixel dans le cas idéal). Une frontière existe s'il y a eu discontinuité de niveau de gris entre deux pixels adjacents (une brusque variation de niveau de gris ou de couleur).
Le contour peut être défini dans la théorie de graphe par un chemin fermé, tel que, les sommets sont les pixels et la variation d'intensité entre les pixels représente les arrêtes.

1.7 Région

Une région est un ensemble de pixels connexes et homogènes. Un pixel appartient à une région donnée s'il vérifie les caractéristiques de celle-ci (intensité moyenne, le centre de gravité …). Une région est limitée par un contour.

1.8 Voisinage

En manipulant l'image, on doit se déplacer dans cette dernière, le déplacement doit obéir à des règles de voisinage. Pour se déplacer d'un pixel à un pixel voisin, on utilise généralement deux types de voisinage :

- un voisinage à 4 pixels
- un voisinage à 8 pixels

1.9 Filtrage

Lorsque l'image ne peut pas être exploitée directement ou après détection de contour, on introduit dans la chaîne de traitement, un étage de filtrage destiné à améliorer la perception ou à simplifier l'image pour en faciliter l'analyse par les étages situés en aval.
Dans ce contexte, les taches assignées au filtrage d'image sont :

- le renforcement de l'homogénéité des régions (réduction de bruit).
- la préservation de la forme des régions.
- le renforcement des différences entre les pixels appartenant à des régions adjacentes (rehaussement de contraste).

Elles peuvent être réalisées par des filtres linéaires ou non linéaires. Le filtrage non linéaire, est souvent fait au prix d'une augmentation de la complexité algorithmique et de temps de traitement.

2. Vision artificielle

La vision artificielle est sans doute l'un des principaux champs d'application de la Reconnaissance des Formes (RF). De nos jours, elle a trouvé une place prépondérante dans un domaine tel que l'imagerie biomédicale. Du diagnostic à la chirurgie assistée par ordinateur en passant par la simulation. Les techniques d'imagerie constituent une aide constante au praticien. Par exemple, dans la détection des cellules cancéreuses du cerveau, les images sont acquises en Imagerie par Résonance Magnétique (IRM).

Au cours de l'étape de segmentation, on va rechercher dans les images les différentes régions du cerveau : matière blanche, matière grise, liquide cérébro-spinal, etc. ....

On cherche ensuite à détecter d'éventuelles tumeurs bénignes ou malignes à partir de caractéristiques extraites de régions de l'image. Bien évidemment, il ne s'agit là que de fournir une aide au diagnostic validée ultérieurement par le praticien tant le coût associé à un mauvais diagnostic est grand.

En biologie marine, certaines bactéries prennent une forme hexagonale en présence de toxines particulières. En couplant un microscope électronique avec un ordinateur, on peut mettre en place une méthode de recherche de motifs hexagonaux dans une image de ces bactéries.
Des systèmes fondés sur ce principe ont déjà été mis en oeuvre pour déceler (indirectement) des traces de pollution.
Dans un autre domaine, les industriels cherchent à contrôler la qualité de fabrication de leurs produits. La vision artificielle peut être utilisée dans un système de détection de défauts. On peut imaginer diverses pièces usinées passant sur un tapis roulant et filmées en continu par une caméra. Avec un traitement en temps réel de cette image, on peut, par exemple, vérifier si l'angle formé par deux pièces assemblées est conforme au cahier des charges. [2]

Les étapes nécessaires à une bonne décision par vision sont décrites dans les paragraphes suivants [3].

2.1 Codage des images

L'opération de codage consiste à transformer un ensemble des données analogique en un ensemble des données numérique de manière à pouvoir les traiter par ordinateur. Cette transformation doit se faire de la manière la plus fidèle possible, c'est-à-dire sans perte d'informations pertinentes et en conservant les propriétés essentielles de l'objet physique. En reconnaissance de la parole, par exemple, le signal obtenu doit reproduire à la fois les différences de timbre de la voix et sa mélodie. En vision, on doit être capable de retrouver la même disposition des objets de la scène, les mêmes rapports de dimensions et de distances entre les objets ainsi que les mêmes nuances de couleur. De manière générale, on exige du codage que la forme obtenue soit la plus intelligible possible même si, dans la plupart des cas, on est incapable d'exprimer clairement ses propriétés.

2.2 Prétraitement des images

Le rôle du prétraitement est de « préparer » les données reçues du capteur à la phase suivante d'analyse consacrée à l'extraction des paramètres. Cette phase n'est possible et surtout fiable que si les données du capteur sont dénuées de bruit, corrigées de leurs erreurs éventuelles, homogénéisées, normalisées et réduites à l'essentiel. Toutes les techniques

élaborées dans ce sens se gardent de modifier les propriétés essentielles des formes, ce qui pourrait conduire, dans le cas contraire à de graves erreurs d'analyse et plus tard de reconnaissance.

2.3 Analyse des images

Le but de l'analyse est d'extraire les propriétés caractéristiques de l'objet et de les exprimer sous une forme numérique ou symbolique. La représentation obtenue servira de base aux étapes ultérieures d'apprentissage et de la reconnaissance.

Le problème consiste à déterminer les propriétés caractéristiques de chacun des objets qui permettent de distinguer à coup sûr un objet des autres. Reconnaître ensuite un objet inconnu consistera à déterminer ses propriétés, à les comparer à celles des objets de référence et à prendre une décision de reconnaissance. Le problème des critères de reconnaissance se pose et nous voyons intuitivement que le choix des critères et celui de la représentation ne sont pas indépendants.

Le choix des critères suppose une connaissance a priori de ce qu'est la forme et la représentation de la forme doit tenir compte des critères choisis pour fournir les informations nécessaires.

On décrit rarement un objet dans sa totalité. En général, on cherche à isoler des parties sur lesquelles on effectuera des mesures (paramètres) reflétant l'identité de la forme générale. On a coutume, pour classer les différentes méthodes d'analyse, de faire la distinction entre deux types d'approche :

- **L'analyse globale**

Cette approche est fondée sur l'étude globale des propriétés de la forme, sans distinction de composition ou de structure. De manière générale, on trouve deux grands types de mesures :

- Numérique : les mesures correspondent à des calculs quantitatifs (diamètre, aire, coefficients de Fourier, amplitude……..) ;
- Logique : les mesures correspondent à des calculs qualitatifs. L'objet est représenté par un vecteur binaire où chaque composante indique la présence ou l'absence d'une certaine propriété.

- **L'analyse structurelle**

L'approche globale est très efficace sur des formes simples. Elle devient totalement inutile sur des formes riches en informations structurelles. C'est le cas en analyse des images où souvent la décision n'est pas uniquement la détermination d'un seul nom ou le calcul d'une seule mesure mais plutôt la description d'une certaine situation ; par exemple, les objets se trouvant dans l'image et leur disposition mutuelle. Cette description se réalise souvent par l'analyse de l'objet en termes de ses formes primitives et de leur relation. C'est ce que l'on appelle l'approche structurelle.

Un objet est généralement défini par une association plus ou moins complexe de formes primitives. Cette association est décrite par des règles indiquant chacune une décomposition possible d'une forme en sous formes et leur agencement relatif.

2.4 Reconnaissance des images

La reconnaissance regroupe les deux tâches d'apprentissage et de décision qui jouent des rôles assez proches dans les systèmes de RF.

En effet, à partir de la même description de la forme en paramètres, elles tentent, toutes les deux, d'attribuer cette forme à un modèle de référence. Le résultat de l'apprentissage est, soit la réorganisation ou le renforcement des modèles existants en tenant compte de l'apport de la nouvelle forme, soit la création d'un nouveau modèle qui représente la forme entrée. Le résultat de la décision est un « avis » sur l'appartenance ou non de la forme aux modèles de l'apprentissage.

- ✓ **L'apprentissage**

La décision nécessite de définir clairement la connaissance que l'on a sur les formes traitées. Cette définition repose sur l'apprentissage qui se charge d'acquérir la connaissance et de l'organiser en classes ou modèles de référence. Cela exige donc de l'apprentissage de bien définir les classes de formes et leur séparatrices (ou les modèles) de manière à bien distinguer les familles homogènes des formes et donc à identifier les nouvelles formes par rapport à elles. Le rôle du concepteur du système n'est pas négligeable ici. Il peut contribuer de diverses manières à la réalisation d'un bon apprentissage, par exemple, en fournissant un bon choix de formes de référence, ou en donnant au système les bons critères de modélisation de sa connaissance de l'univers.

Il existe deux types d'apprentissage, supervisé et non supervisé.

- L'apprentissage supervisé : l'apprentissage est dit supervisé si les différentes familles des formes sont connues a priori et si la tâche d'apprentissage est guidée par un superviseur ou professeur.
- L'apprentissage non supervisé : on l'appelle aussi, suivant l'approche utilisée, classification automatique, inférence ou encore apprentissage sans professeur. Il s'agit, à partir d'échantillons de référence et de règles de regroupement ou de modélisation, de construire automatiquement les classes ou les modèles sans l'intervention de l'opérateur.

✓ **La décision**

La décision est l'ultime étape de reconnaissance. A partir de la description en paramètres, elle recherche, parmi les modèles d'apprentissage en présence, ceux qui sont les plus « proches». La notion de proximité a un sens différent en fonction de la nature de la représentation et du type de la méthode. La décision peut conduire à un succès si la réponse est unique (un seul modèle répond à la description de la forme). Elle peut conduire à une confusion si la réponse est multiple (plusieurs modèles correspondent à la description, ce qui dénote une certaine ambiguïté volontaire ou non au sein de l'apprentissage). Enfin, la décision peut conduire à un rejet de la forme si aucun des modèles ne correspond à sa description.

3 Prétraitements d'images

Le traitement d'images (ou prétraitement) regroupe l'ensemble des processus visant à améliorer les caractéristiques d'une image [4].

- Le lissage local : il s'agit de supprimer le bruit, ou les petites variations, présent dans une image. L'intensité d'un pixel est transformée en fonction des intensités sur un petit voisinage du pixel.
- L'amélioration d'images consiste à modifier les caractéristiques visuelles de l'image (contraste, ...) pour faciliter son interprétation par l'oeil humain.

- La restauration d'images a pour but de supprimer les dégradations subites par une image à l'aide de connaissance *a priori* sur ces dégradations.

3.1 Lissage local

Le lissage local (ou filtrage) consiste à supprimer le bruit présent dans une image en étudiant, pour chaque pixel, les valeurs d'intensité sur son voisinage.

3.1.1 Moyennage

Une première classe d'approche est basée la redondance d'informations. La nouvelle valeur d'un pixel est calculée par moyennage des valeurs sur un voisinage.

Cette opération linéaire peut être vue comme la convolution discrète de l'image par un masque

$$I'(i,j) = \sum_{(m,n)\in v} h(m,n) I(i-m, j-n) \qquad (1.1)$$

$$\text{Tel que } \sum_{(m,n)\in v} h(m,n) = 1$$

Ou I est l'intensité de l'image d'origine, I' est l'intensité de l'image filtrée, v est le voisinage utilisé et h est le masque de convolution.

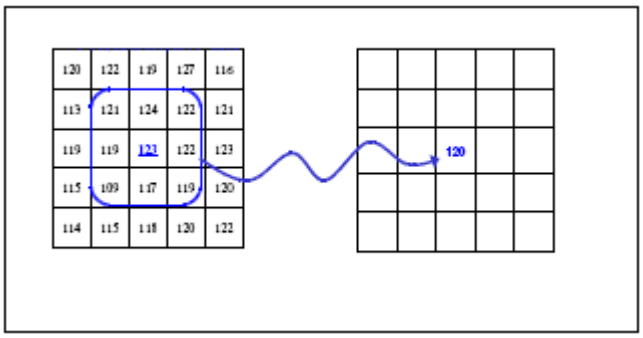

Figure 1.1 : Moyennage sur un voisinage 3X3

Le moyennage sur un voisinage 3x3 :

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

- ✓ Le moyennage est un filtre passe-bas.
- ✓ Rend l'image floue, en particulier les contours.
- ✓ Élimine les dégradations locales de faibles dimensions. Valide lorsque les objets présents dans l'image sont de dimensions supérieures aux dégradations.

Une amélioration du filtre moyen consiste à jouer sur les valeurs des coefficients du masque :

$$h_1 = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Dans le cas des filtres binomiaux pour lesquels les valeurs des coefficients sont générées par le triangle de Pascal :

$$h_2 = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- C'est une approximation discrète d'un filtre Gaussien adaptée au filtrage des bruits Gaussiens.

3.1.2 Filtres médians

Les filtres de moyennage ont tendance à rendre l'image floue et donc à perdre de l'information sur les contours caractérisés par de fortes variations d'intensité.
Pour diminuer cet effet, on ne moyenne plus sur le voisinage mais on prend la valeur médiane sur ce voisinage. C'est le filtre médian.
Exemple pour un voisinage 3x3 :

$$h = \begin{bmatrix} 2 & 12 & 12 \\ 2 & 12 & 60 \\ 2 & 2 & 12 \end{bmatrix}$$

La valeur médiane est ici 12.

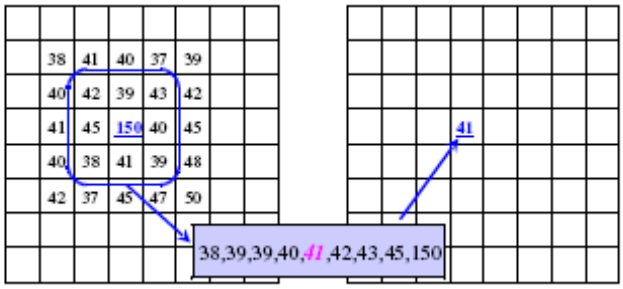

Figure 1.2 : Filtres médians sur un voisinage 3X3

Caractéristiques :

- ✓ Filtre non linéaire.

- ✓ Élimine le bruit impulsionnel.
- ✓ Préserve l'information de contour et peut être appliqué itérativement.
- ✓ Élimine les contours très fins. Un voisinage adapté permet de limiter cet effet.

3.2 Amélioration d'images

L'amélioration d'images consiste à modifier les caractéristiques visuelles de l'image de manière à en faciliter son interprétation par l'oeil humain. Il peut s'agir de rehausser les contrastes, d'accentuer certaines intensités pour mettre en valeur une région, ... Les histogrammes sont fréquemment utilisés pour effectuer ce type d'opérations.

3.2.1 Les histogrammes

L'histogramme d'une image $hist(i)$ est la fonction qui associe à une valeur d'intensité i le nombre de pixels dans l'image ayant cette valeur.

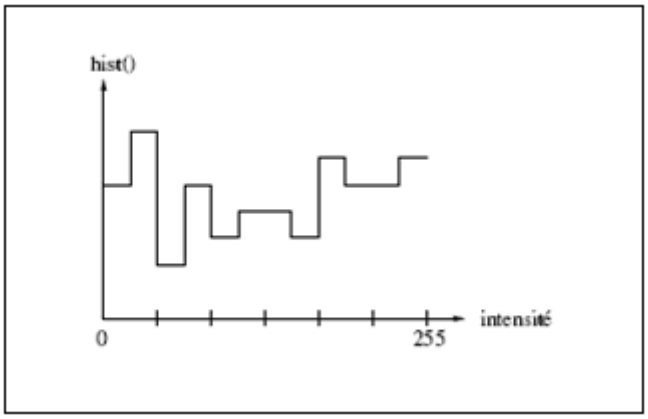

Figure 1.3 : L'histogramme d'une image

- ✓ Pour une image couleur, il y a un histogramme par composante.
- ✓ L'histogramme peut être normalisé pour donner une estimation de la densité de probabilité des pixels :

$$p(i) = \frac{hist(i)}{\sum_j hist(j)} \qquad (1.2)$$

$$\text{Tel que } \sum_i p(i) = 1$$

- ✓ Un histogramme peut avoir un pic (unimodale), deux pics (bimodale) ou plusieurs pics (multimodale).

3.2.2 Modifications d'histogrammes

Pour modifier les caractéristiques de l'image (accentuer les contrastes en général), une approche générale consiste à appliquer une fonction qui associe à chaque valeur d'intensité dans l'image une nouvelle valeur. Cette fonction va modifier l'histogramme de l'image.

Soit $i \text{ tel que } i\langle MaxInt$ les valeurs d'intensité de l'image traitée, on considère alors les transformations du type : $i' = T(i)$

Qui donne une nouvelle valeur d'intensité i' pour chaque valeur i de l'image. On suppose que la fonction $T()$ est telle que :

- $T(i)$ est monotone (souvent croissante) sur l'intervalle. Cette condition assurant que l'ordre des intensités est préservé après transformation.
- $0 \le T(i) \le MaxInt$ pour $0 \le i \le MaxInt$ qui garantit que la nouvelle image est cohérente avec les niveaux d'intensité autorisés.
- La transformation inverse satisfait les deux conditions précédentes.

3.2.3 Transformations linéaires

Les transformations linéaires d'histogrammes sont nombreuses et variées. Elles permettent d'accentuer une zone d'intensité ou de modifier la répartition des valeurs d'intensité. Un exemple de transformation linéaire est le recadrage.

Le recadrage consiste à modifier l'intervalle des valeurs d'intensité (la dynamique) de façon à obtenir pour l'image améliorée un intervalle de valeurs maximal.

Si $[i1, i2]$ est l'intervalle de l'image traitée, alors la transformation équivalente s'écrit simplement :

$$T(i) = MaxInt\,(i - i1)/(i2 - i1) \qquad (1.3)$$

4. État de l'art sur les méthodes d'extraction des paramètres

Le nombre de méthodes de caractérisation est relativement élevé. On distingue deux grandes approches comme cela est d´écrit par **la figure 1.4** : l'approche contour qui caractérise la forme à partir de son contour et l'approche globale qui étudie la forme dans son ensemble.

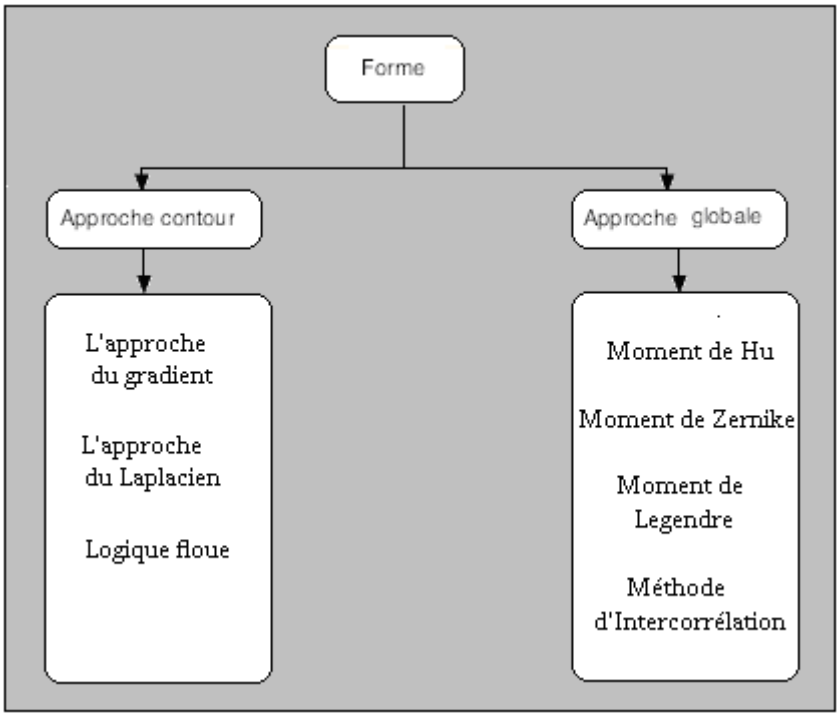

Figure 1.4 : Différentes approches pour la phase de caractérisation

4.1 Approche globale pour la phase de caractérisation [5]

4.1.1 Reconnaissance d'une forme en utilisant l'intercorrélation

Le principe du procédé avancé se décompose en deux phases, une phase de programmation et une phase d'analyse.

Phase de programmation : Cette phase consiste à construire un masque binaire à partir de l'image à reconnaître et à la stocker dans les points mémoire situés dans le pixel. Le masque binaire est obtenu en seuillant le signal fourni par l'élément photosensible. A partir de ce masque et de son complémentaire, on peut définir deux grandeurs d'apprentissage :

$$S_{B0} = \sum\sum M(x,y) \times \mathrm{Im}(x,y) \qquad (1.4)$$

$$S_{N0} = \sum\sum (1 - M(x,y)) \times \mathrm{Im}(x,y) \qquad (1.5)$$

Où Im(x, y) représente la valeur délivrée par l'élément photosensible et M(x, y) la valeur binaire du contenu dans le point mémoire SRAM. Les valeurs du masque M sont obtenues par seuillage de l'image (Im) d'apprentissage.

Phase d'analyse Au cours de la phase d'analyse le masque M est appliqué sur une nouvelle image. On obtient donc en sortie deux nouvelles valeurs :

$$S_B = \sum\sum M(x,y) \times \mathrm{Im}(x,y) \qquad (1.6)$$

$$S_N = \sum\sum (1 - M(x,y)) \times \mathrm{Im}(x,y) \qquad (1.7)$$

Si l'image d'analyse est différente de l'image de référence, les valeurs SB et/ou SN sont modifiées par rapport aux valeurs SB0 et SN0. En considérant la distance entre les points d'apprentissage et les points de décision, on peut conclure quant à la similitude ou non avec l'objet d'apprentissage.

4.1.2 Reconnaissance d'une forme en utilisant les moments

Dans [6], l'auteur introduit les moments en faisant le parallèle avec leur utilisation en physique. En effet, les moments sont utilisés en physique pour décrire la répartition des masses dans un corps. En analyse d'image, on peut envisager la même démarche en associant le niveau de gris d'un point de l'image à la masse élémentaire en un point. On comprend donc que les différents moments fournissent des informations concernant l'arrangement spatial de l'objet.

D'une façon générale le moment ϕ_{pq} d'ordre $p+q$ est défini comme étant la description d'une image Im(x, y) sur une base décrite par une fonction $f_{pq}(x,y)$:

$$\Phi_{pq} = \sum\sum f_{pq}(x,y) \times \mathrm{Im}(x,y) \qquad (1.8)$$

Où x et y sont les coordonnées de l'image.

L'ensemble des ϕ_{pq} représente les caractéristiques d'une image. Ces paramètres seront représentatifs de la forme et devront posséder certaines caractéristiques comme l'invariance en rotation, en translation et être le moins sensible possible au bruit. Ces paramètres sont connus sous le nom d'invariants ou bien de descripteurs.

Dans le cadre de la reconnaissance de formes, les moments les plus utilisés sont les moments géométriques et les moments de Zernike.

a) Les moments géométriques / les invariants de Hu

Historiquement, en 1962 Hu fut le premier à présenter des travaux sur l'analyse d'image en utilisant les moments. Il utilisa les moments géométriques pour générer une famille d'invariants qui fût utilisée pour la reconnaissance automatique de caractères. Nous allons dans cette partie définir les moments de Hu, énoncer leurs caractéristiques et propriétés puis nous allons vérifier l'influence de la translation, de la rotation ainsi que le bruit.

Comme nous l'avons cité plus haut, Hu a introduit la famille d'invariants portant son nom en utilisant les moments géométriques. Ces moments sont définis par :

$$m_{pq} = \sum\sum \mathrm{Im}(x,y) x^p y^q \qquad (1.9)$$

Ainsi définis, les moments géométriques ne sont pas invariants en rotation et en translation. Par contre les moments géométriques centrés sont invariants en translation.

Soit x0 et y0 le barycentre de l'image, défini par :

$$\begin{cases} x_0 = \dfrac{m_{10}}{m_{00}} \\ y_0 = \dfrac{m_{01}}{m_{00}} \end{cases}$$

On définira les moments géométriques centrés par la fonction :

$$\mu_{pq} = \sum\sum \mathrm{Im}(x,y)(x-x_0)^p (y-y_0)^q \qquad (1.10)$$

Cependant ils ne sont pas invariants en rotation. Pour résoudre cette problématique Hu forma une famille d'invariants en rotation au moyen de combinaisons de moments géométriques centrés. Les cinq premiers invariants de Hu sont définis dans les équations : 1.11, 1.12, 1.13, 1.14, et 1.15. De plus, un cœfficient peut être introduit de manière à rendre ces paramètres invariants au facteur d'échelle.

$$\phi_1 = \mu_{20} + \mu_{02} \qquad (1.11)$$

$$\phi_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2 \qquad (1.12)$$

$$\phi_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2 \qquad (1.13)$$

$$\phi_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2 \qquad (1.14)$$

$$\phi_5 = (\mu_{30} - 3\mu_{12})\ (\mu_{30} + \mu_{12})\ [(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + (3\mu_{21} - \mu_{03})\ (\mu_{03} + \mu_{21})\ [-(\mu_{03} + \mu_{21})^2 + 3(\mu_{12} + \mu_{30})^2] \qquad (1.15)$$

b) Les moments de Zernike

Les moments de Zernike introduits en 1934 sont définis par les équations : 1.16, 1.17, et 1.18. C'est donc le produit d'une fonction complexe : 1.17 par un polynôme radial, 1.18.

Ces moments définissent donc des valeurs complexes. Il faudra donc deux masques (réel et imaginaire) pour calculer les invariants de zernike.

$$z_{pq} = \frac{(p+1)}{\pi(N-1)^2} \sum_{x=1}^{N} \sum_{y=1}^{N} v_{pq}^{*}(r,\theta)\,\mathrm{Im}(x,y) \qquad (1.16)$$

Avec

$$\begin{cases} x = r\cos(\theta) \\ y = r\sin(\theta) \end{cases} \quad \Rightarrow \quad \begin{cases} r = \sqrt{(x^2 + y^2)} \\ \theta = \tan^{-1}(\frac{y}{x}) \end{cases}$$

$$\text{Et} \quad v_{pq}^{*} = R_{pq}(r)\exp(iq\theta) \qquad (1.17)$$

$$R_{pq}(r) = \sum_{s=0}^{\frac{p-|q|}{2}} (-1)^s \frac{(p-s)!}{s!(\frac{p-2s+|q|}{2})!(\frac{p-2s-|q|}{2})!} r^{p-2s} \qquad (1.18)$$

Khotanzad a démontré dans [7] que le module des moments de Zernike est invariant en rotation. Dans [8], Raveendran a introduit des invariants en translation composes par des combinaisons de moments de Zernike. Certains de ces invariants sont représentes par les équations : 1.19, 1.120, 1.21, et 1.22.

$$z_{trans00} = z_{00} \qquad (1.19)$$

$$z_{trans11} = z_{11} - 2Az_{00} \qquad (1.20)$$

$$z_{trans20} = z_{20} - 3A^{*}z_{11} - 3Az_{11}^{*} + 6AA^{*}z_{00} \qquad (1.21)$$

$$z_{trans22} = z_{22} - 3Az_{11} + 3A^{2}z_{00} \qquad (1.22)$$

Tel que A est un paramètre complexe et A^{*} son conjugué.

4.2 Approche contour (structurelle) pour la phase de caractérisation

L'approche vue précédemment ne permet pas de prendre en compte l'information structurelle et contextuelle d'une forme. Ces informations sont pourtant importantes dans un

grand nombre de problèmes de reconnaissance de formes. C'est, par exemple, le cas en analyse d'images où souvent la décision n'est pas uniquement la détermination d'un seul nom ou le calcul d'un seul nombre mais plutôt la description d'une certaine situation, par exemple les objets se trouvant dans l'image et leur disposition mutuelle. Cette description peut être réalisée par l'analyse de l'objet en termes de ses composants, de leurs propriétés et de leur connexion.

Nous allons étudier les trois méthodes les plus courantes :

- L'approche du gradient, c'est à dire la recherche du maximum de la dérivée première.
- L'approche du Laplacien qui revient à chercher les pixels où le Laplacien s'annule. En d'autres termes la recherche du passage par zéro de la dérivée seconde.
- L'approche basée sur la théorie de la logique floue

4.2.1 Détection de contour par l'approche du gradient [4]

Les contours représentent des variations (sauts brusques ou transitions) de l'intensité lumineuse. Dans le cas d'une scène plane, ils représentent la projection des contours des objets de la scène. Dans le cas d'une scène 3-D, les contours des objets dans une image sont supposés être également les projections des contours délimitant des objets de la scène.

Les contours étant considérés comme marquant les transitions fortes de niveaux de gris. On applique une dérivation spatiale (gradient) sur l'image. Les opérateurs de détection de contours, basés sur le gradient (dérivée première), permettent de quantifier la largeur de la transition (amplitude du gradient obtenu) et sa direction (direction du gradient) en examinant, pour chaque pixel, un voisinage.

Le gradient d'une image est le vecteur $\nabla I(u,v)$ défini par :

$$\nabla I(u,v) = \left(\frac{\partial I(u,v)}{\partial u} \quad \frac{\partial I(u,v)}{\partial v}\right)^T \qquad (1.23)$$

Il est donc caractérisé par un module m et une direction ϕ dans l'image :

$$m = \sqrt{\left(\frac{\partial I(u,v)}{\partial u}\right)^2 + \frac{\partial I(u,v)}{\partial v})^2} \qquad (1.24)$$

$$\phi = \arctan\left(\frac{\frac{\partial I(u,v)}{\partial u}}{\frac{\partial I(u,v)}{\partial v}}\right) \qquad (1.25)$$

L'amplitude est directement liée à la quantité de variation locale des niveaux de gris, et la direction du gradient est orthogonale à la frontière qui passe au point considéré.

Les points de contours sont ensuite détectés en recherchant **les maximums locaux** du module dans la direction du gradient.

Il s'agit de comparer le gradient G en un point M avec les gradients G1 et G2 des deux voisins pris dans la direction du gradient. Si G>G1 et G>G2, alors M est un maximum local.

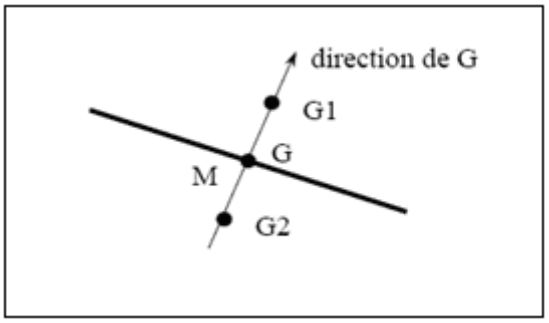

Figure 1.5 : Calcul d'un maximum local selon le gradient

Cette méthode se résume en plusieurs étapes:

- Le calcul de la dérivée partielle ainsi que le module et la direction du gradient
- La recherche des maximums locaux dans la direction du gradient
- Le seuillage

Le détecteur de Sobel est une approximation discrète de cette dérivée sur un voisinage (3x3) :
Détecte les composantes du gradient selon l'horizontale

$$G_u = \frac{\partial I}{\partial u} \approx I(u,v) * O_u \qquad (1.26)$$

Détecte les composantes du gradient selon la verticale

$$G_v = \frac{\partial I}{\partial v} \approx I(u,v) * O_v \qquad (1.27)$$

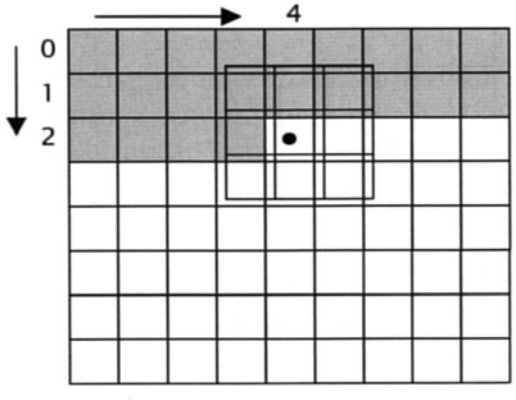

Figure 1.6 : Les directions horizontale et verticale de Sobel

Masques de convolution

$$O_u = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$O_v = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = O_u^T$$

$$G_u(u,v) \approx \left| -I(u-1,v-1) + I(u-1,v+1) - 2I(u,v-1) + 2I(u,v+1) - I(u+1,v-1) + I(u+1,v+1) \right| \quad (1.28)$$

$$G_v(u,v) \approx \left| -I(u-1,v-1) - 2I(u,v-1) - I(u+1,v-1) + I(u-1,v+1) + 2I(u,v+1) + I(u+1,v+1) \right| \quad (1.29)$$

$$G = \sqrt{G_u^2 + G_v^2} \approx |G_u| + |G_v| \qquad (1.30)$$

$$\arg(G) = \arctan\left(\frac{G_V}{G_U}\right) \qquad (1.31)$$

4.2.2 Détection de contour par l'approche du Laplacien [4]

Les opérateurs linéaires basés sur la dérivée première (gradient) sont tous directionnels, ils privilégient des directions (le détecteur de Sobel privilégie les horizontales et les verticales). Le détecteur de Laplace est un opérateur linéaire isotrope utilisant une approximation discrète de la dérivée seconde spatiale (Laplacien) sur un voisinage (3x3). Seule, l'amplitude peut être déterminée :

$$L(u,v) = \nabla^2 I(u,v) = \frac{\partial^2 I}{\partial u^2} + \frac{\partial^2 I}{\partial v^2} \approx I(u,v) * O_L \qquad (1.32)$$

Voici deux masques de convolution du détecteur de Laplace correspondant à deux approximations possibles de la dérivée seconde spatiale dans un voisinage (3x3) :

$$O_L = \frac{1}{4}\begin{vmatrix} 0 & -1 & 0 \\ 1 & 4 & 1 \\ 0 & -1 & 0 \end{vmatrix} \text{ et } O_L = \frac{1}{8}\begin{vmatrix} 0 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{vmatrix}$$

• Les contours correspondent aux passages à zéro du Laplacien de l'image.

• Etant basé sur la dérivée seconde, le Laplacien amplifie plus le bruit que le filtre de Sobel.

Cette méthode se résume en plusieurs étapes:

- Un filtrage passe-bas pour atténuer la sensibilité au bruit.
- Le calcul du Laplacien.
- La recherche des passages par zéro du Laplacien.
- Le seuillage.

4.2.3 Détection de contour utilisant la logique floue [9]

L'utilisation de règles binaires dans la détection de contours fonctionne relativement bien sur des images du type synthétique (i.e.: générées par ordinateur). Ces images ne contiennent peu ou pas de bruit et les contrastes sont bons. Dans ce cas précis, les algorithmes de détection de contours n'ont pas à être robustes. Par contre, dans un cas réel, où l'on doit tenir compte des imperfections de l'image il est important d'avoir un système de détection qui soit immunisé contre les variations locales de l'image. L'avantage de la logique floue c'est qu'elle permet de faire des raisonnements sur la base de niveau de vérité, donc elle s'accommode très bien avec le bruit.

La discussion portera sur deux différentes implémentations de détecteurs flous de type Mamdani. Elles utilisent toutes les deux la même structure d'opérateur et ce dans le but de comparer quelle est la méthode la plus robuste. La conception de l'opérateur se divise en trois étapes: fuzzyfication, règles d'inférence et défuzzyfication.

a) Structure de l'opérateur

Dans cette étude, le détecteur de contours utilise les différences entre un pixel central Pn et les pixels Xi qui l'entourent pour déterminer si le pixel central fait partie ou non d'un contour dans l'image.

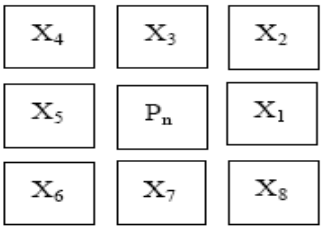

Figure 1.7 : Structure du masque de l'opérateur

La définition de l'opérateur flou se fait de manière linguistique. Il se décrit comme suit: si le pixel central est trop différent de ceux qui l'entourent, il sera considéré comme faisant partie d'un contour. Alors le pixel Pn de l'image résultant deviendra noir. Si le détecteur le considère comme similaire à son entourage, le pixel Pn sera blanc dans l'image résultante. La position relative des pixels est définie par un masque (Figure **1.7**) que l'on déplace sur l'image.

b) **Fuzzyfication**

L'étape de la fuzzyfication consiste à convertir chaque entrée en des variables linguistiques similaires à des sous-ensembles flous. Cela permet d'établir le degré d'appartenance (u) des entrées pour chaque variable linguistique. Les variables linguistiques utilisées pour déterminer si un pixel fait partie ou non d'un contour sont les suivantes: l'étiquette que l'on nomme *écart* contient deux règles floues qui se nomment *petit* et *grand*. Petit et Grand représentent le niveau de différence entre le pixel central et les autres pixels.

L'étiquette des règles floues pourrait se lire comme suit: Si l'ÉCART est PETIT alors le pixel sera BLANC, si l'ÉCART est GRAND alors le pixel sera NOIR. Les règles floues de l'étiquette sont représentées à **la figure 1.8**. Deux manières de calculer les variables d'entrées ont été implémentées.

Méthode #1 : On fait la moyenne des pixels Xi et on prend la valeur absolue de la différence avec Pn. On utilise x comme entrée "crisp".

$$Xmoy = 1/8 \sum X_i \qquad Xi\ i=1,2,\ldots,8 \qquad (1.33)$$

$$x = | Pn - Xmoy | \qquad (1.34)$$

Méthode #2 : On prend la valeur absolue de la différence de chaque pixel Xi avec Pn. On utilise les xi comme entrée "crisp".

$$xi = | Pn - Xi |\ i=1,2,\ldots,8 \qquad (1.35)$$

Il est à noter que la méthode #1 produit une seule entrée (1.33) alors que la méthode #2 produit huit entrées (1.35) au moteur d'inférence.

Le raisonnement flou est implémenté à partir de fonction triangulaire en raison de leur simplicité (**Figure 1.8**). Il est à noter que l'allure de la courbe (sigmoïde ou autre) n'augmente pas la complexité de calcul lorsque celle-ci est implémentée sous forme de "look-up table".

Dans le graphique de **la figure 1.8**, la valeur **v** représente la variance de l'image. Dans l'application des méthodes floues #1 et #2, la variance **v** a été calculée préalablement à la détection. Le choix de la valeur du seuil qui sépare les sous-ensembles flous PETIT et GRAND est souvent déterminé par expérimentation. Un nombre de valeurs peut donner de bons résultats (moyenne des pixels, niveau de bruit, etc.), par contre l'utilisation de la variance est répandue car elle est intimement reliée à l'amplitude du gradient. En effet, le gradient est un excellent indice de contour, par contre, il ne faut pas que l'algorithme confonde chaque variation avec un contour. C'est entre autre pour cette raison que plusieurs méthodes de détection de contour utilisent la variance car elle donne un ordre de grandeur du gradient, donc un ordre de grandeur des transitions dans l'image.

Bien entendu, pour une implémentation en temps réel ou encore lorsque que les capacités de calculs sont limitées, ont peut aussi calculer la variance locale et l'utiliser comme estimateur de la variance globale. Cela permet de minimiser le temps de calcul en temps réel. Cette piste reste à développer.

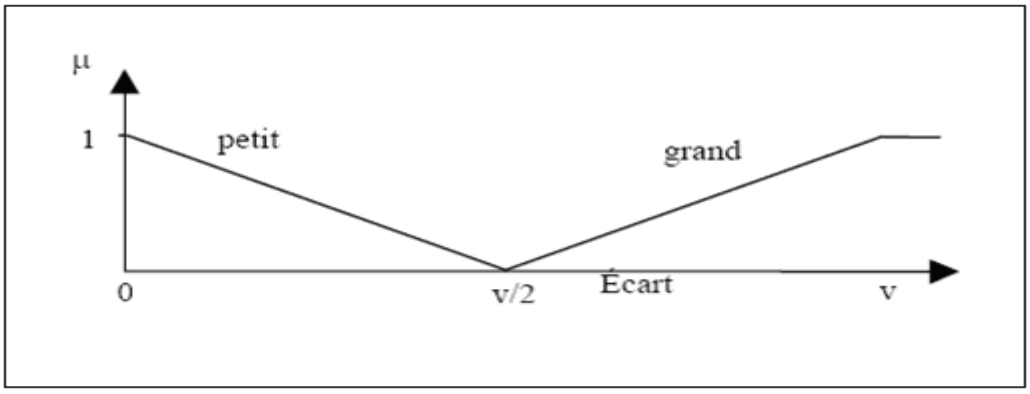

Figure 1.8 : Fonction d'appartenance

c) Moteur d'inférences

Les règles de contrôle dictent la manière dont le moteur d'inférence prendra sa décision, à savoir dans le cas des méthodes #1 et #2, est-ce que le pixel central fait partie ou non d'un contour?

Une règle est un énoncé conditionnel qui se compose de deux parties. La première partie est l'antécédent qui représente les conditions à remplir et la seconde est le conséquent qui est l'action qui va être posée. Ici les antécédents sont les xi (1.35) pour la méthode #2 et le x (1.34) pour la méthode #1. Les conséquents sont les mêmes pour les des méthodes, ils sont

NOIR ou BLANC. Les deux méthodes utilisent le même moteur d'inférence dans le sens ou seule la quantité d'entrée diffère.

Le moteur d'inférence #1 utilise les règles de décisions suivantes:

IF |Pn –Xmoy| is **PETIT THEN** pixel is BLANC

IF |Pn –Xmoy| is **GRAND THEN** pixel is NOIR

Le moteur d'inférence #2 utilise les règles de décision suivantes:

IF |Pn –X1| is **PETIT and** |Pn –X2| is **PETIT and** |Pn –X3| is **PETIT and** |Pn –X4| is **PETIT and** |Pn –X5| is **PETIT and** |Pn –X6| is **PETIT and** |Pn –X7| is **PETIT and** |Pn –X8| is **PETIT THEN** pixel is BLANC

IF |Pn –X1| is **GRAND and** |Pn –X2| is **GRAND and** |Pn –X3| is **GRAND and** |Pn –X4| is **GRAND and** |Pn –X5| is **GRAND and** |Pn –X6| is **GRAND and** |Pn –X7| is **GRAND and** |Pn –X8| is **GRAND THEN** pixel is NOIR. Il existe d'autres méthodes pour réaliser le moteur d'inférence. Les opérateurs de type FIRE (Fuzzy Inference Ruled by Elseaction) utilisent des règles du type: **IF** A est petit **THEN** pixel est BLANC **ELSE** pixel est NOIR]

Il est à noter que l'opération **AND** floue représente l'union des sous-ensembles flous. Cette opération de conjonction est implémentée avec la fonction *min*.

d) Défuzzyfication

La défuzzyfication consiste à faire correspondre les degrés flous associés aux conséquents (variables de sortie du moteur d'inférence) en une valeur non floue. Pour chaque fonction d'appartenance (PETIT ou GRAND) pour la variable de sortie, on combine les degrés d'appartenance flous avec la fonction *max.*

$$\mu petit = \max(0.1, 0.5, 0.7, \ldots, 0.3) = 0.7 \qquad (1.36)$$

Matlab simplifie énormément la tâche de défuzzification, à cause de cela, la méthode des centroïdes a été sélectionnée. Il existe toutefois un grand nombre de méthodes, toutes ont leurs avantages et inconvénients il incombe au concepteur de faire le bon choix. Par exemple, pour une application en temps réel une approximation du centroïde serait possiblement suffisante.

5 Conclusion

Dans ce chapitre on a présenté un état de l'art sur les systèmes de vision artificielle, le chapitre suivant les circuits FPGA son architecture et son fonctionnement.

Les circuits FPGA

1 Les circuits logiques programmables

De nombreuses familles de circuits programmables et reprogrammables sont apparues depuis les années 70 avec des noms très divers suivant les constructeurs**. La figure 2.1** donne une classification possible des circuits numériques en précisant où se situe chaque type de ces circuits dans cette classification. [10]

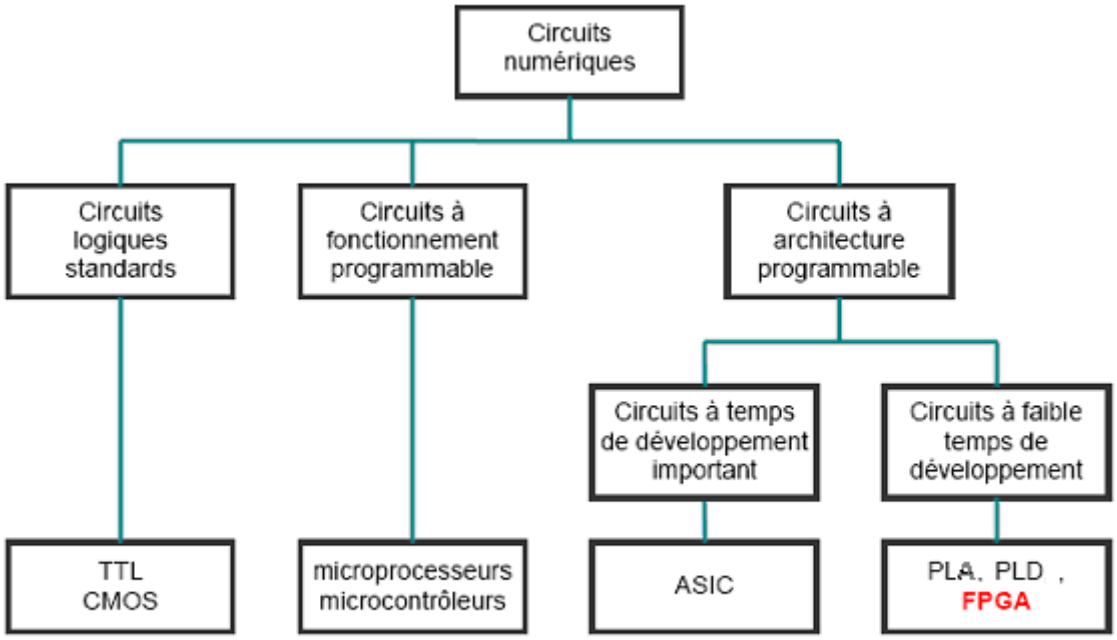

Figure 2.1 : Classification des circuits numériques

1.1Les ASICs (Application Specific Integrated Circuit) [11]

Par définition, les circuits ASIC regroupent tous les circuits dont la fonction peut être ***personnalisée*** d'une manière ou d'une autre en vue d'une application spécifique, par opposition aux circuits standards dont la fonction est définie et parfaitement décrite dans le catalogue des composants.

Les ASIC peuvent être classés en plusieurs catégories selon leur niveau d'intégration, en fait un ASIC est défini par sa structure de base (réseau programmable, cellule de base, matrice, etc.). Sous le terme ASIC deux familles sont regroupées, les semi personnalisés (avec des réseaux prédéfinis) et les personnalisés (non préfabriqué, qu'on optimise pour créer son propre composant).

D'une manière générale l'utilisation d'un ASIC conduit à de nombreux avantages provenant essentiellement de la réduction de la taille des systèmes. Il en ressort, une réduction du nombre des composants sur le circuit imprimé. La consommation et l'encombrement s'en trouvent considérablement réduits. ·

Le concept ASIC par définition assure une optimisation maximale du circuit à réaliser. Nous disposons alors d'un circuit intégré correspondant réellement à nos propres besoins.
La personnalisation du circuit donne une confidentialité au concepteur et une protection industrielle.

Enfin, ce type de composant augmente la complexité du circuit, sa vitesse de fonctionnement et sa fiabilité.

Dans l'approche des circuits du type ASIC, l'inconvénient majeur réside dans le fait du *passage obligatoire chez le fondeur* ce qui implique des frais et un temps de développement élevés du circuit.

Les ASICs sont généralement plus convenables pour les productions en série de conceptions déjà vérifiées et non pour les prototypes.

1.2 Les PLDs (Programmable Logic Device) [12]

Ce sont des chips qui peuvent être programmés pour se comporter comme une conception arbitraire. Un PLD peut être programmé pour une implémentation aussi simple qu'une opération en logique combinatoire comme il pourrait l'être pour des conceptions beaucoup plus importantes. Il Comprend une matrice de portes AND connectée à une matrice de portes OR il est alors représenté sous forme de sommes de produits.
La figure suivante montre deux types de connexions dans les PLDs.

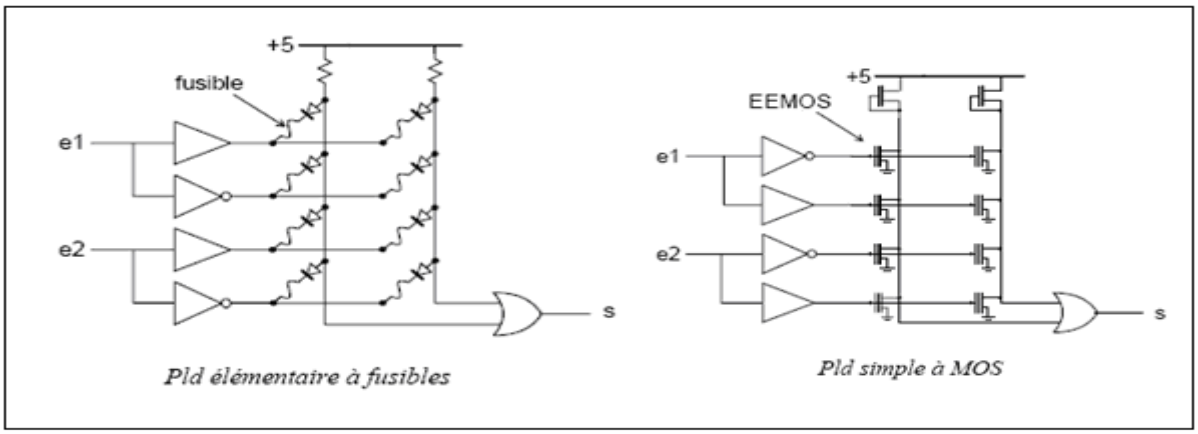

Figure 2.2 : les connexions dans un PLD

1.3 Les EPLDs "Erasable PLD" [13]

Qui sont des circuits effaçables par rayons ultraviolets, ils peuvent être reprogrammés.

1.4 Les PALs "Programmable Array Logic"

C'est la partie essentielle des PLD, elle est constituée d'un plan de portes AND programmables suivi d'un plan de portes OR fixes ou figées. **Figure 2.3** [12].

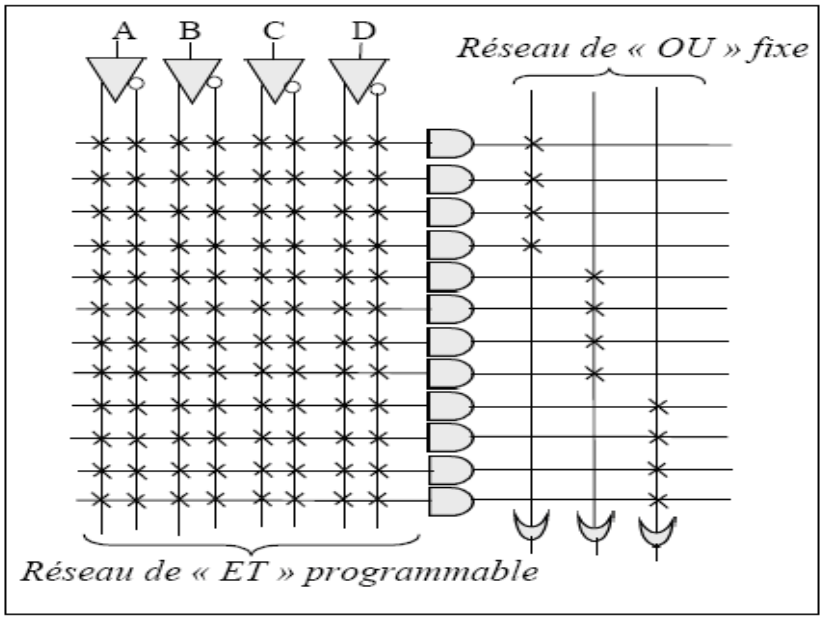

Figure 2.3: les connexions dans un PAL

1.5 Les EEPLDs "Electrically Erasable PLD" [13]

Programmables et effaçables électriquement, ils peuvent être reprogrammés sur site. Les limites de l'architecture du PLD résident dans le nombre de bascules, le nombre de signaux d'entrées/sorties, la rigidité du plan logique ET/OU et des interconnexions. Précisons que ces composants très souples d'emploi sont limités à des fonctions numériques et adaptées à des productions de petites séries et ne présentent aucune garantie quant à la confidentialité.

1.6 Les PLAs "Programmable Logic Array"

Sont constitués par un plan de portes AND suivi d'un plan de portes OR. Mais dans ce cas les connexions dans les deux plans sont programmables.

La figure suivante représente un type de représentation d'un PLA [12].

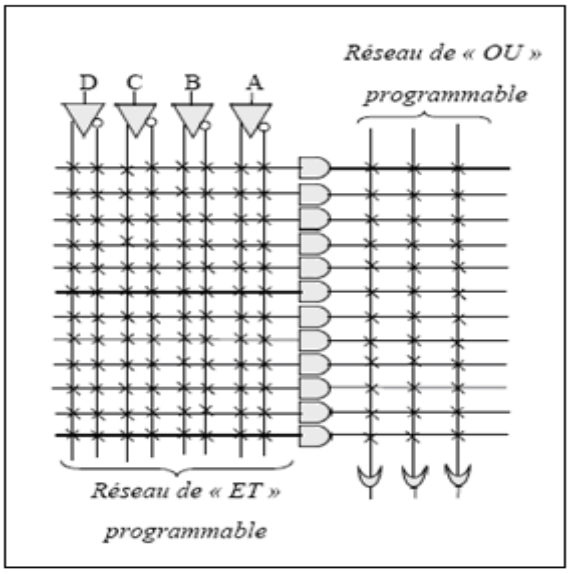

Figure 2.4 : les connexions d'un PLA

1.7 Les FPGA"Field Programmable Gate Arrays" ou "réseaux des blocs programmables"

Les circuits FPGAs sont des composants entièrement reconfigurables ce qui permet de les reprogrammer à volonté afin d'accélérer notablement certaines phases de calculs. Les circuits FPGAs sont constitués d'une matrice de blocs logiques programmables entourés de

blocs d'entrée/sortie programmables. L'ensemble est relié par un réseau d'interconnexions programmable.

Les FPGAs ne sont pas optimisés pour une application bien déterminée, par conséquent ils consomment plus d'énergie que les ASICs. Par contre ils sont beaucoup plus simples à programmer et à reprogrammer, ce qui raccourcit les cycles de conception et permet de suivre l'évolution de l'application pour laquelle, ils ont été conçus.

Les FPGAs sont plus convenables pour les prototypes et pour les productions en série limitées qui ne sont pas de la qualité des ASICs. [11]

1.8 Les Critères de performances dans les circuits programmables [12]

- **Puissance de calcul**

Les premiers chiffres accessibles concernent les nombres d'opérateurs utilisables.

- **Vitesse de fonctionnement**

Les comportements dynamiques des FPGAs et des PLDs simples présentent des différences marquantes. Les premiers ont un comportement prévisible, indépendamment de la fonction programmée ; les limites des seconds dépendent de la fonction, du placement et du routage. Une difficulté de jeunesse des FPGAs a été la non reproductibilité des performances dynamiques en cas de modification, même mineure, du contenu d'un circuit. Les logiciels d'optimisation et les progrès des architectures internes ont pratiquement supprimé ce défaut ; mais il reste que seule une analyse et une simulation post synthèse, qui prend en compte les paramètres dynamiques des cellules, permet réellement de prévoir les limites de fonctionnement d'un circuit.

- **Taille du système**

Il y a une forte dépendance entre la taille du système et la densité d'intégration.

L'augmentation de la densité d'intégration produit des systèmes de taille réduite.

La figure suivante représente la densité d'intégration dans les différentes familles des circuits logiques :

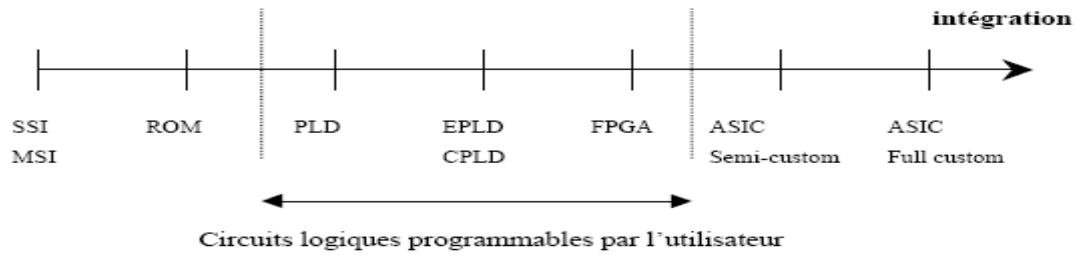

Figure 2.5 : l'intégration dans les circuits logiques

• **Temps de développement**

Le temps de développement est proportionnel au degré d'intégration dans les circuits logiques.

• **Nombre de portes équivalentes**

Le nombre de portes est sans doute l'argument le plus utilisé dans les effets d'annonce. En 1997 la barrière des 100000 portes est largement franchie. Plus délicate est l'estimation du nombre de portes qui seront inutilisées dans une application, donc le nombre réellement utile de portes.

• ***Nombre de cellules***

Le nombre de cellules est un chiffre plus facilement interprétable : le constructeur du circuit a optimisé son architecture, pour rendre chaque cellule capable de traiter à peu près tout calcul dont la complexité est en relation avec le nombre de bascules qu'elle contient (une ou deux suivant les architectures). Trois repères chiffrés : un 22V10 contient 10 bascules, la famille des CPLDs va de 32 bascules à quelques centaines et celle des FPGAs s'étend d'une centaine à quelques milliers. Dans les circuits à architectures cellulaires, il est souvent très rentable d'augmenter le nombre de bascules si cela permet d'alléger les blocs combinatoires.

• ***Nombre d'entrée-sortie***

Le nombre de ports de communication entre l'intérieur et l'extérieur d'un circuit peut varier dans un rapport de deux, pour la même architecture interne, en fonction du boîtier choisi. Les chiffres vont de quelques dizaines à quelques centaines de broches d'entrées-sorties.

• ***Capacité mémoire***

Les FPGAs à SRAM contiennent des mémoires pour stocker leur configuration. La plupart des familles récentes offrent à l'utilisateur la possibilité d'utiliser certaines de ces mémoires en tant que telles. Par exemple, la famille 4000 de XILINX permet d'utiliser les mémoires de configuration d'une cellule pour stocker 32 bits de données ; la cellule correspondante n'est évidemment plus disponible comme opérateur logique. Les capacités de mémorisation atteignent quelques dizaines de kilobits.

• **Consommation**

Les premiers circuits programmables avaient plutôt mauvaise réputation sur ce point. Tous les circuits actuels ont fait d'importants progrès en direction de consommations plus faibles.

• ***Le compromis vitesse consommation***

Règle générale des circuits numériques, encore plus vraie dans le monde des technologies MOS que dans celui des technologies bipolaires : pour aller vite il faut de la puissance. Les notices fournissent communément des courbes de consommation pour des éléments classiques, comme un compteur synchrone 16 bits, en fonction de la fréquence d'horloge. Le passage à 3,3 V des tensions d'alimentation permet une économie non négligeable de puissance, pour les mêmes valeurs de courant.

2 Les FPGAs "Field Programmable Gate Arrays"

2.1 Historique

Le principe de la logique programmable remonte au début des années 1960, le concept ayant été proposé par Estrin. Mais ce n'est qu'au cours des années 1980 que les premières réalisations matérielles sont introduites sur le marché. L'apparition de ce type de circuit s'est d'abord faite au travers de circuits logiques programmables simples de type PAL (Programmable Array Logic), qui se programment comme des mémoires de type ROM et sont utilisés pour implémenter des fonctions combinatoires simples, telles des décodeurs d'adresse ou des contrôleurs de bus.

Avec les évolutions en microélectronique, différentes familles de circuits programmables ont vu le jour : les CPLD (Complex Logic Programmable Device), puis les FPGA (Field Programmable Gate Arrays), introduits par la société Xilinx en 1985. Avec l'apparition de circuits de plus en plus performants et reprogrammables à volonté, l'industrialisation et la commercialisation de ce type de circuits s'est faite à grande échelle. À l'heure actuelle, on compte une dizaine de fabricants, le marché étant nettement dominé par les sociétés Xilinx, Altera (circuits reprogrammables) et Actel (circuits non reprogrammables). Actuellement ces fabricants mettent sur le marché des circuits de type FPGA qui offrent désormais l'équivalent de dix millions de portes logiques programmables, à des fréquences de fonctionnement atteignant les 200MHz. [14]

2.2 Architecture des circuits FPGA (Architecture retenue par Xilinx)

Les circuits FPGA possèdent une structure matricielle de deux types de blocs (ou cellules). Des blocs d'entrées/sorties et des blocs logiques programmables. Le passage d'un bloc logique à un autre se fait par un routage programmable. Certains circuits FPGA intègrent également des mémoires RAM, des multiplieurs et même des noyaux de processeurs. [15]

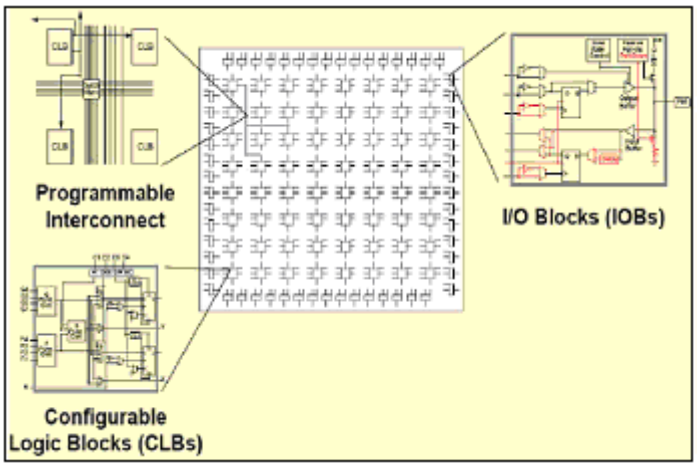

Figure 2.6 : Architecture interne du FPGA

Dans ce qui suit, on va faire une description de l'architecture utilisée par Xilinx, car c'est sur des circuits Xilinx qu'on va implémenter nos programmes.

L'architecture retenue par Xilinx se présente sous forme de deux couches :

• Une couche appelée ***Circuit Configurable***,

• Une couche réseau mémoire ***SRAM*** (Static Read A*ccess* Memory).

➢ **Circuit configurable**

La couche dite « circuit configurable » est constituée:

✓ d'une matrice de **blocs logiques configurables** (**CLB**) permettant de réaliser des fonctions combinatoires et des fonctions séquentielles (**figure 2.7**).

✓ Tout autour de ces blocs logiques configurables, on trouve des **blocs d'entrées/sorties** (**IOB**) dont le rôle est de gérer les entrées-sorties réalisant l'interface avec les modules extérieurs (**figure 2.6**).

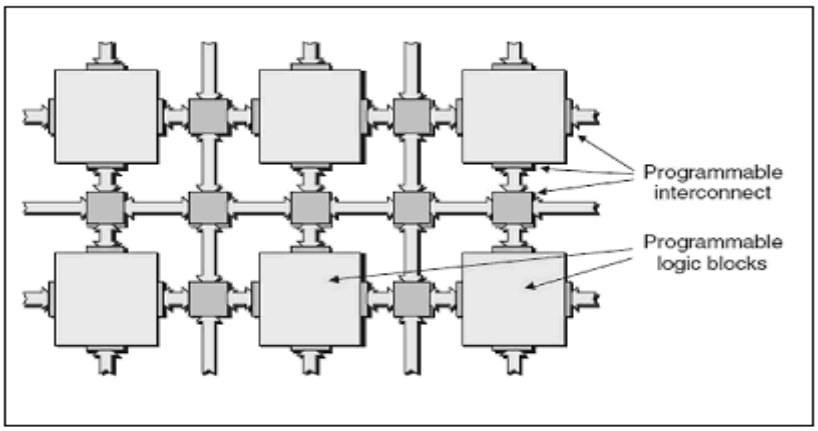

Figure 2.7 : Blocs et Interconnections programmables

- **Réseau mémoire SRAM**

La programmation du circuit FPGA, appelé aussi LCA (Logic Cells Arrays), consistera en l'application d'un potentiel adéquat sur la grille de certains transistors à effet de champ servant à interconnecter les éléments des CLB et des IOB, afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux. Ces potentiels sont tout simplement mémorisés dans le réseau mémoire SRAM (**figure 2.8**). [10]

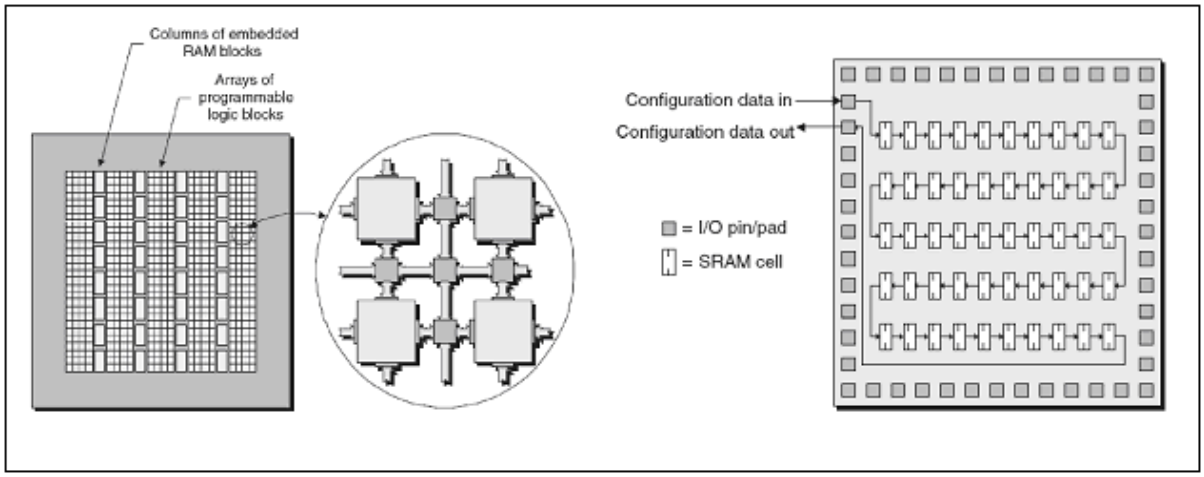

Figure 2.8 : Situation du réseau SRAM

La programmation d'un circuit FPGA est volatile, la configuration du circuit est donc mémorisée sur la couche réseau SRAM et stockée dans une ROM externe. Un dispositif interne permet à chaque mise sous tension de charger la SRAM interne (**figure 2.9**) à partir de la ROM. Ainsi on conçoit aisément qu'un même circuit puisse être exploité successivement avec des ROM différentes puisque sa programmation interne n'est jamais définitive. On voit ici tout le profit que l'on peut tirer de cette souplesse en particulier lors d'une phase de mise au point. Une erreur n'est pas rédhibitoire, mais peut aisément être réparée.

La mise au point d'une configuration s'effectue en deux temps : Une première étape purement logicielle va consister à dessiner puis simuler logiquement le circuit fini. Dans la seconde étape, on effectuera une simulation matérielle en configurant un circuit réel. On pourra alors vérifier si le fonctionnement réel correspond bien à l'attente du concepteur, et si besoin est identifier les anomalies liées généralement à des temps de transit réels légèrement différents de ceux supposés lors de la simulation logicielle, ce qui peut conduire à des états instables voire même erronés.

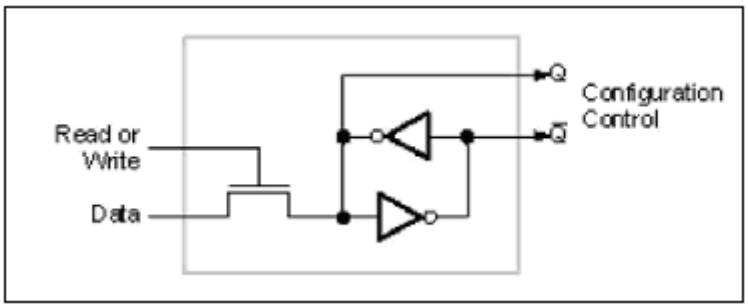

Figure 2.9 : Structure d'une cellule SRAM

Les circuits FPGA du fabricant Xilinx utilisent deux types de cellules de base : les cellules d'entrées/sorties appelées IOB (Input Output Bloc), et les cellules logiques appelées CLB (Configurable Logic Bloc). Ces différentes cellules sont reliées entre elles par un réseau d'interconnexions configurable. On décrit dans ce qui suit chacun de ces composants.

2.2.1 CLB (Configurable Logic Bloc)

C'est l'unité fondamentale du bloc logique qui fournit des éléments utilitaires pour la logique combinatoire et la logique synchrone, y compris les éléments du stockage de base : buffers à 3 états à associer avec chaque CLB. Les CLBs incluent quatre parties identiques appelées SLICE, **Figure. 2.10** [10] et deux buffers à 3 états.
Chaque SILICE est équivalente et contient :

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

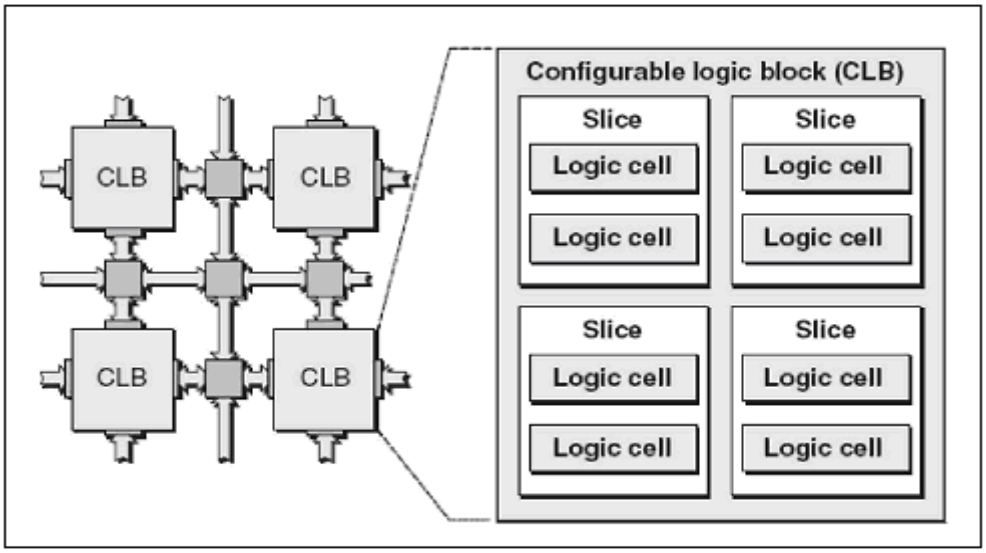

Figure 2.10 : Bloc CLB

Les blocs logiques configurables sont les éléments déterminants des performances du circuit FPGA (**figure 2.10**). Chaque CLB est un bloc de logique combinatoire composé de générateurs de fonctions à quatre entrées (LUT) et d'un bloc de mémorisation/synchronisation composé de bascules D.

Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB.

La LUT (Look Up Table) est un élément qui dispose de quatre entrées, il existe donc $2^4 = 16$ combinaisons différentes de ces entrées. L'idée consiste à mémoriser la sortie correspondant à chaque combinaison d'entrée dans une petite table de 16 bits, la LUT devient ainsi un petit bloc générateur de fonctions. **La figure 2.11** montre le schéma simplifié d'un CLB de la famille XC4000 de Xilinx.

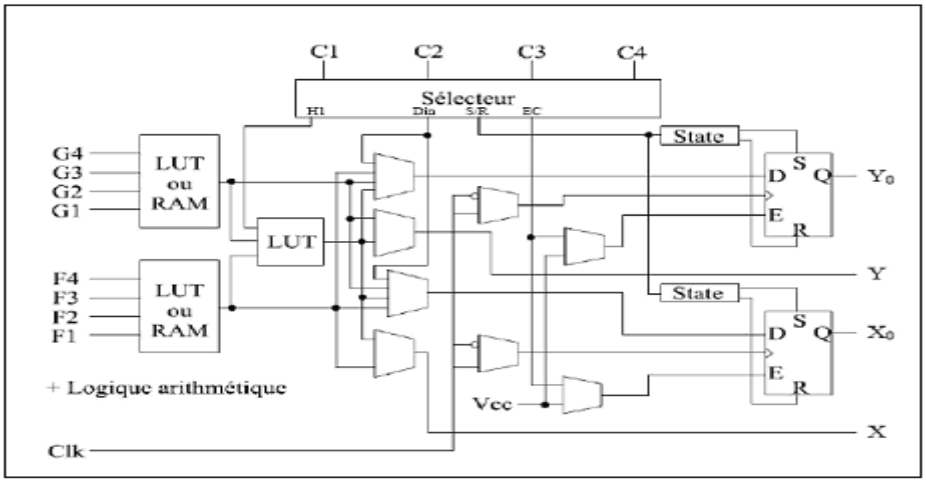

Figure 2.11 : Schéma d'une cellule logique (XC4000 de Xilinx)

2.2.2 IOB (Input Output Bloc)

Ils constituent l'interface entre les bornes du circuit et les CLB. Le dispositif de routage Versa Ring offre les ressources nécessaires à l'interconnexion des CLB aux IOB. Nous pouvons ainsi modifier le système implanté sur le FPGA sans interférer avec l'attribution des bornes. Cette caractéristique s'avère importante si nous souhaitons développer des nouvelles versions d'un produit tout en conservant la compatibilité au niveau du boîtier.

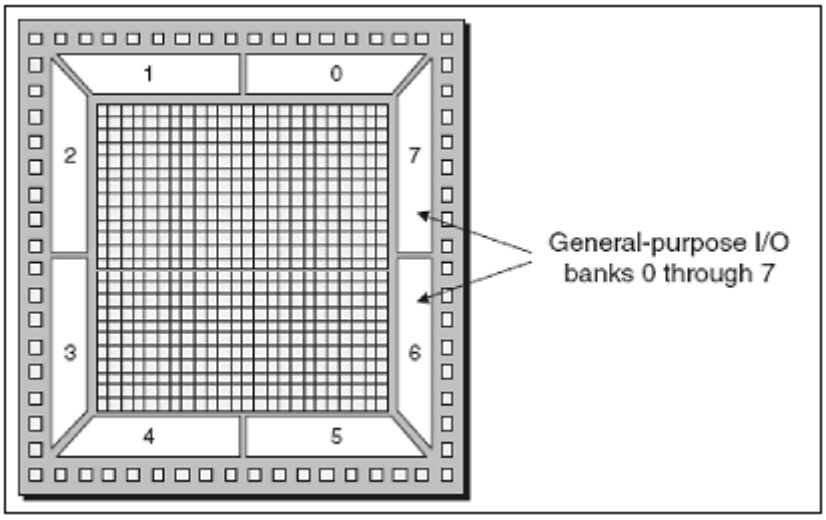

Figure 2.12 : Exemple de IOB

Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signal bidirectionnel ou être inutilisé (état haute impédance). **La figure 2.13** présente la structure de ces blocs.

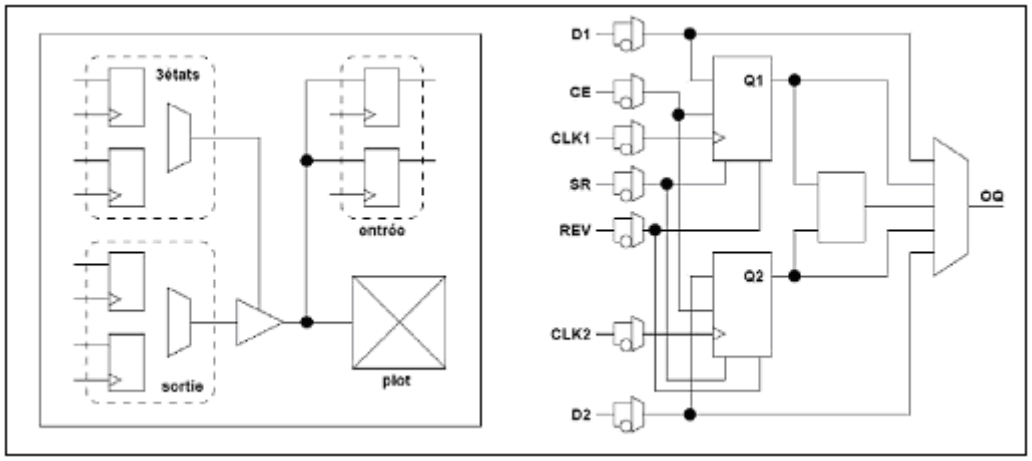

Figure 2.13 : Schéma d'un bloc d'entrée/sortie (IOB)

- **Configuration en entrée**

Premièrement, le signal d'entrée traverse un buffer qui, selon sa programmation, peut détecter soit des seuils TTL soit des seuils CMOS. Il peut être routé directement sur une entrée directe de la logique du circuit FPGA ou sur une entrée synchronisée. Cette synchronisation est réalisée à l'aide d'une bascule de type D, le changement d'état peut se faire sur un front montant ou descendant. De plus, cette entrée peut être retardée de quelques nanosecondes pour compenser le retard pris par le signal d'horloge lors de son passage par l'amplificateur. Le choix de la configuration de l'entrée s'effectue grâce à un multiplexeur (program controlled multiplexer). Un bit positionné dans une case mémoire commande ce dernier.

- **Configuration en sortie**

Nous distinguons les possibilités suivantes :

- ✓ inversion ou non du signal avant son application à l'IOB ;
- ✓ synchronisation du signal sur des fronts montants ou descendants d'horloge ;
- ✓ mise en place d'un "pull-up" ou "pull-down" dans le but de limiter la consommation des entrées sorties inutilisées ;
- ✓ signaux en logique trois états ou deux états. Le contrôle de mise en haute impédance et la réalisation des lignes bidirectionnelles sont commandés par le signal de commande « Out Enable », lequel peut être inversé ou non.

Chaque sortie peut délivrer un courant de 12mA. Ainsi toutes ces possibilités permettent au concepteur de connecter au mieux une architecture avec les périphériques extérieurs.

2.3 Programmation et configuration des circuits FPGAS

La configuration est le processus de charger des données spécifiques à une conception dans un ou plusieurs FPGAs pour définir l'opération fonctionnelle des blocs internes ainsi que leur interconnexion, et leurs temps de configuration dépendent du mode de configuration sélectionnée.

Dans la littérature on peut distinguer 4 modes de configuration et programmation des circuits FPGAs comme le montre **la figure 2.14** [13].

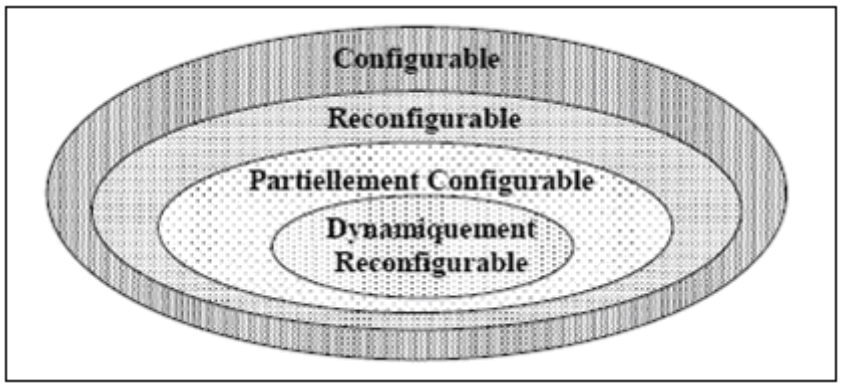

Figure 2.14 : Classification des circuits FPGAs selon leurs configurations

2.3.1 Circuit configurable

Un circuit Configurable est un circuit programmé et chargé par différentes données, où les interconnections d'un FPGA sont programmées afin de donner un fonctionnement spécifique pour un tel circuit.

2.3.2 Circuit reconfigurable

C'est le même principe de configuration, sauf que cette fois en reconfigurant le circuit FPGA une deuxième fois pour l'utiliser dans une autre fonction comme on peut garder la dernière configuration du circuit. On peut même effacer cette configuration et on configure le circuit une nouvelle fois.

2.3.3 Circuit partiellement reconfigurable

Un dispositif ou un circuit est défini comme partiellement reconfigurable (dans la littérature on trouve aussi la terminologie Run Time Reconfiguration RTR globale) s'il est possible de le reconfiguré sélectivement, tandis que l'état de repos du reste du dispositif est inactif, mais il conserve son information configurée. Encore, il ne semble pas y avoir n'importe quel dispositif sur le marché qui soit partiellement reconfigurable, mais non aussi dynamiquement reconfigurable, La reconfiguration partielle permet de rendre un FPGA effectif, multiple fonctions, et change des fonctions pendant le fonctionnement du système.

2.3.4 Circuit dynamiquement reconfigurable

Un circuit FPGA est reconfigurable dynamiquement (dans la littérature on trouve la terminologie Run Time Reconfiguration RTR locale) s'il peut être partiellement reconfiguré durant son fonctionnement, c à d une partie du circuit correspondant à certaines fonctions logiques et leur interconnections peut être changées sans affecter le fonctionnement de la logique restante.

On peut aussi parler de reconfiguration dynamique dans le cas où plusieurs circuits FPGAs sont connectés entre eux et il s'agit de reconfiguré un seul composant FPGA tout en maintenant les autres circuits en fonctionnement.

2.4 Application des FPGA

Les FPGA sont utilisés dans de nombreuses applications, on en cite dans ce qui suit quelques unes: [16]

- Prototypage de nouveaux circuits ;
- Fabrication de composants spéciaux en petite série ;
- Adaptation aux besoins rencontrés lors de l'utilisation ;
- Systèmes de commande à temps réel ;
- DSP (Digital Signal Processor) ;
- Imagerie médicale.

2.5 Conclusions

La technologie logique programmable se positionne comme une solution intermédiaire entre processeurs programmables et circuits dédiés.

En regard de l'application «La reconnaissance des formes » une évaluation des contraintes nous conduit d'une part à retenir le principe du circuit programmable(Virtex-II), le langage Handel-C et d'autre part à utiliser des composants disposants de ressources hétérogènes :

- ✓ granularité fine pour les nombreuses fonctions logiques nécessaires.
- ✓ possibilités de routage sans dégrader les temps de propagation.

3 La famille Virtex-II de Xilinx

3.1 Architecture de la famille Virtex-II

La famille Virtex -II, a une capacité de 4 à 10 millions de portes dans une technologie de pointe : 0.15µm à 8 niveaux de métallisation. Leur architecture reste très près de celle du Virtex (des blocs multiplieurs câblés font leur apparition). Cette famille contient une architecture optimisée pour une grande vitesse (horloge système interne 420 MHz, cadence externe de 33 MHz à 133 MHz) et une consommation d'énergie minimale.

Elle combine entre la flexibilité dans l'intégration et la densité d'intégration. Elle comporte deux modules de base : les cores générateurs (fonctions DSP, fonctions mathématique, microprocesseur) et les modules personnalisés. Elle apporte des solutions pour les télécommunications, les réseaux, la vidéo et les applications DSP. C'est la première famille à avoir intégré des multiplieurs et des blocs Select RAM dans sa structure interne.

Les technologies de pointe 0.15 µm / 0.12 µm CMOS à 8 couches de métallisation, le processus et l'architecture Virtex-II sont optimisées pour une grande vitesse d'horloge avec une faible consommation d'énergie. Elle combine entre la flexibilité dans l'intégration et la densité d'intégration plus de 10 millions de portes logiques, son architecture est divisée en deux blocs principaux, les blocs d'entrée/sortie programmable et bloc logique interne programmable comme le montre **la figure 2.15** [13].

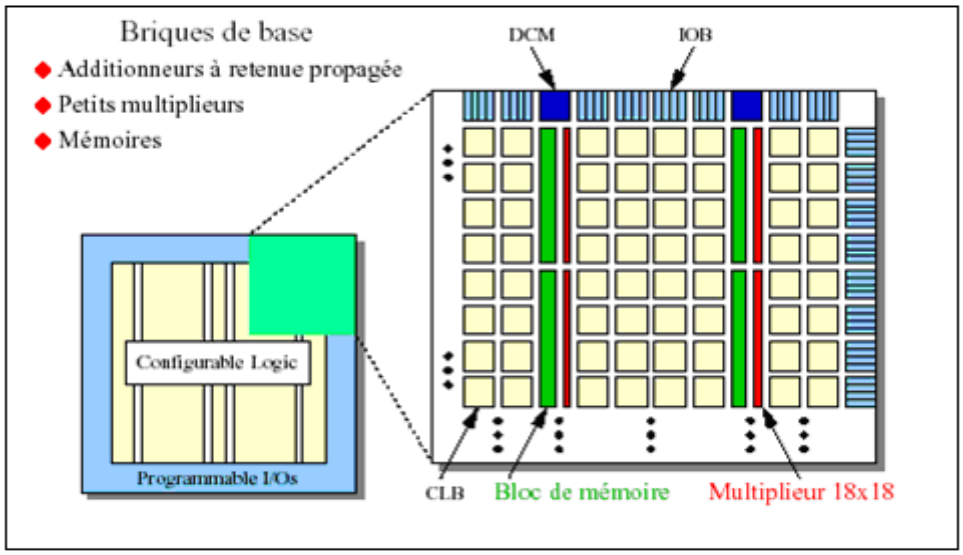

Figure 2.15 : Architecture interne de la famille VIRTEX-II

3.1.1 Les blocs Multiplieurs

Les blocks multiplieurs sont des multiplieurs de18x18 bits. Le circuit VIRTEX-II incorpore une grande densité de blocs multiplieurs. Chaque multiplieur peut être associé au bloc Select RAM ou peut être utilisé indépendamment **Figure 2.15**.

3.1.2 Les blocs DCM (Digital Clock Manager)

Le DCM produit le nouveau système d'horloges (soit intérieurement ou extérieurement au FPGA), il produit une large gamme de fréquences de l'horloge par multiplication et même par division. Le bloc logique programmable contient plus de 12 DCM voir (**Figure 2.15)**.

3.2 Kit de développement RC203E de celoxica [17]

La famille FPGA Virtex-II possède les outils avancés pour répondre à la demande à des applications de haute performance. Le kit de développement RC203E de celoxica fournit une excellente plateforme pour explorer ces outils, l'utilisateur peut alors utiliser toutes les ressources disponibles avec rapidité et efficacité.

La figure 2.16 présente une photo de la carte de développement et de ses outils.

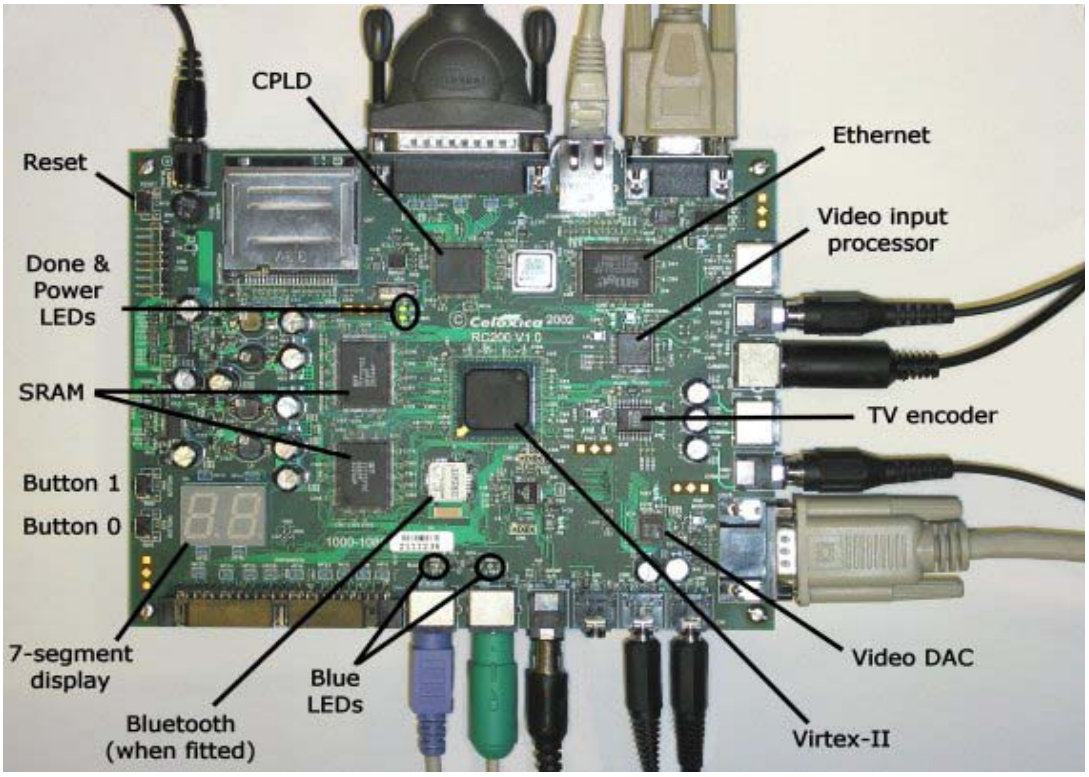

Figure 2.16 : Carte de développement RC203E de celoxica

Les différents organes qui peuvent connecter avec le kit de développement RC203E sont schématisés dans la figure suivante :

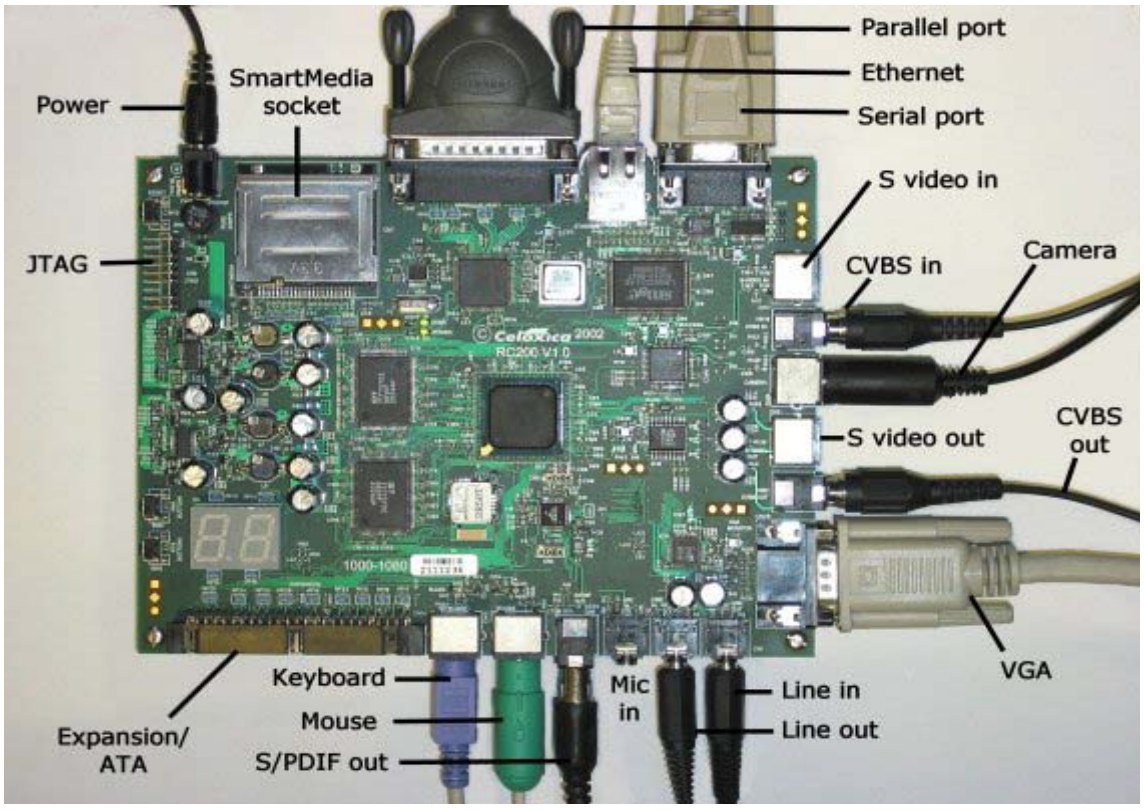

Figure 2.17 : Les différentes Connexions dans RC203E

3.2.1 Description de la carte de développement

Un diagramme simplifié de la carte de développement celoxica RC203E est donné par **la figure 2.18**.

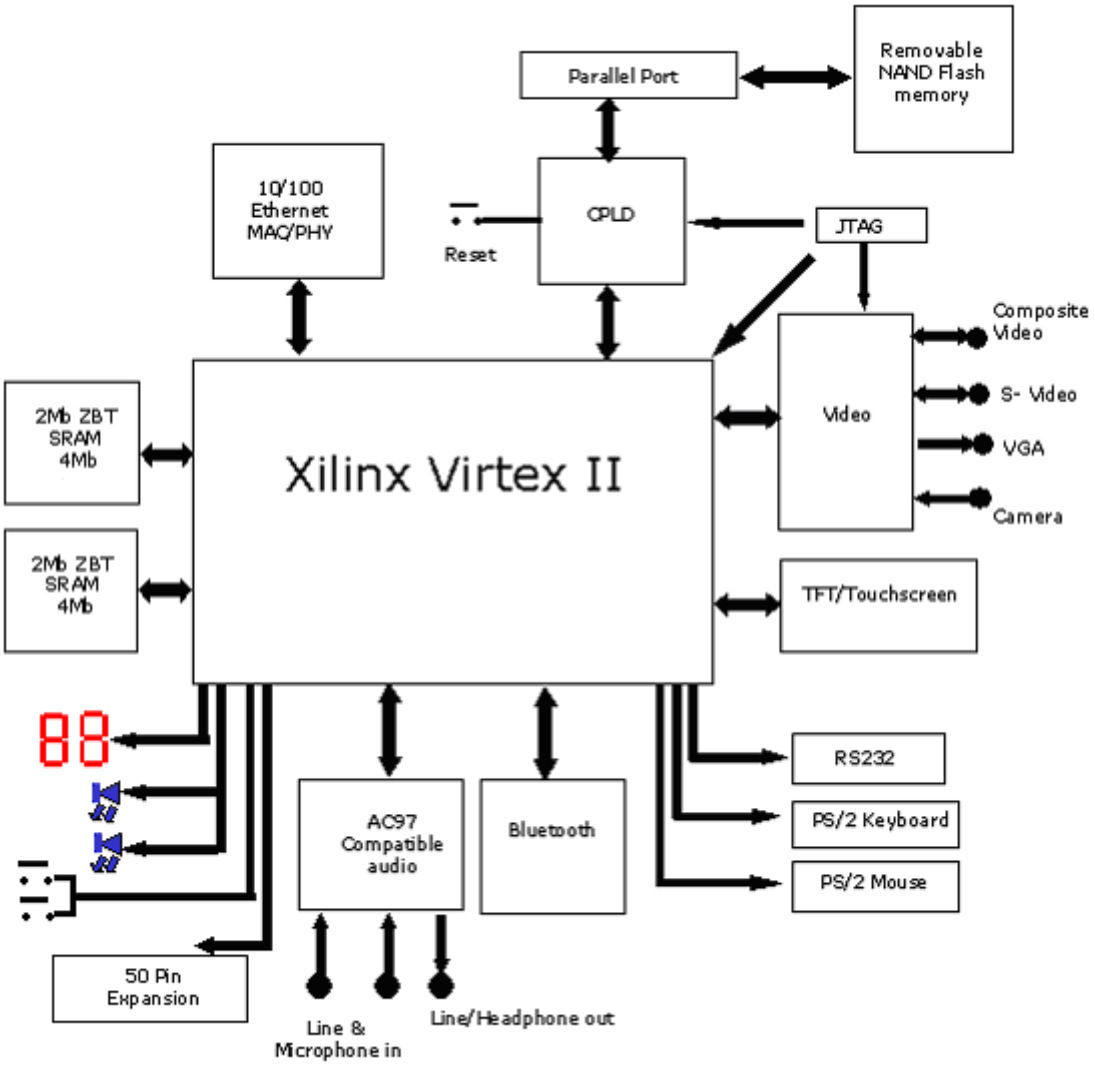

Figure 2.18 : Diagramme de la carte de développement RC203E

La carte de développement RC203E contient le circuit FPGA **XC2V3000-4FG456C**. Ce circuit fait partie de la famille Virtex-II, qui est une famille de circuits développés pour des applications haute performance telles que les télécommunications, l'imagerie et les applications DSP. Il possède 50 broches dont 33 peuvent être utilisées en entrées/sorties.

La carte contient également deux mémoires SRAM de 4MB. Elle présente 2 générateurs d'horloges internes. Elle contient aussi un circuit de remise à zéro « Reset » activé par un bouton poussoir.

Deux LEDs et deux afficheurs 7 segments à cathode commune sont présents sur la carte, et peuvent être utilisés durant la phase de test et de debugging. Il existe aussi 2 entrées exploitables par l'utilisateur (DIP switch) qui peuvent être mis statiquement à un état haut ou bas.

La carte possède une interface RS232, une interface JTAG pour programmer l'ISP PROM et configurer le circuit FPGA ainsi qu'un connecteur de câble parallèle qui peut aussi être utilisé pour configurer le FPGA.

Il existe également un port VGA, un port camera et un port de composition vidéo IN/OUT, une interface audio de type AC'97 compatible, un connecteur clavier PS/2, un connecteur sourie, et une mémoire flash (SmartMedia flash memory) pour charger le fichier bit.

La carte de développement comporte aussi un circuit programmable CPLD pour la configuration/reconfiguration de la carte, un connecteur Ethernet MAC/PHY 10/100 BaseT et trois générateurs de tension (12v, 5v, 3.3v).

3.2.2 Chargement du programme sur la carte de développement

La carte de développement RC203E supporte plusieurs méthodes de configuration de son circuit FPGA. Le port parallèle peut être utilisé directement pour configurer le FPGA, les ports SmartMedia et JTAG peut être aussi utilisés pour configurer le FPGA.

4 Le langage de programmation du FPGA : le Handel-C [18]

Le langage Handel-C est développé par l'université d'Oxford, en Grande-Bretagne. Il permet de générer des circuits, mais n'est pas un langage de description de circuits, c'est un langage séquentiel. Les programmes écrits en Handel-C peuvent d'ailleurs être traduits en C-ANSI `a l'aide d'un compilateur disponible, et exécutés sur des machines séquentielles. Le compilateur produit une netlist qui doit ensuite être appliquée aux circuits reconfigurables à l'aide des outils commerciaux. Pour obtenir un circuit performant, il est nécessaire de paralléliser. Les outils de parallélisation automatique ne sont pas suffisamment performants, les concepteurs de Handel-C ont donc choisi de laisser le programmeur exprimer les différentes formes de parallélisme : parallélisme de processus, parallélisme d'assignation et parallélisme d'expression.

Handel-C a permis de réaliser des applications impressionnantes, comme des jeux vidéo sans ordinateur, en branchant un moniteur VGA directement en sortie d'un circuit Xilinx qui pilotait `a la fois l'affichage et le jeu. Le même programme, compile sur une Sun Sparc-5 s'exécute 5 fois plus lentement.

C'est un langage qui permet de générer des circuits logiques, sans être pour autant un HDL (Hardware Description Langage). Il est plutôt destiné à compiler des algorithmes de haut niveau pour en faire des portes logiques (niveau hardware), sans que l'utilisateur ne se préoccupe de savoir exactement comment fonctionne le calculateur. C'est en fait un langage de programmation plutôt qu'un HDL. En un sens, le Handel-C est au hardware ce qu'un langage de haut niveau conventionnel est `a l'assembleur.

4.1 Les programmes en Handel-C

La syntaxe du Handel-C est basée sur celle du C dont elle reprend de nombreuses structures. Aussi les Programmes en Handel-C sont-ils intrinsèquement séquentiels. L'ordre d'écriture des instructions correspond exactement à leur ordre d'exécution. Bien sur, comme tout langage qui se respecte, le Handel-C contient des structures de contrôle du flot du programme. Par exemple, l'exécution d'une partie de code peut être conditionnée par la valeur d'une expression, ou un bloc de code peut-être répété un certain nombre de fois grâce à une boucle.

Il est important de noter que les circuits logiques générés par le Handel-C sont exactement ceux nécessaire à l'exécution du programme. Il n'y a pas d'interpréteur comme en assembleur ; les portes logiques constituant le circuit Handel-C final sont des instructions assembleur du système Handel-C.

4.2 Les programmes parallèles

Puisque le compilateur Handel-C génère du hardware de bas niveau, il est possible d'accroître les performances du système en utilisant le parallélisme. Bien que le séquentiel soit inhérent au Handel-C, on peut (et c'est même parfois primordial) demander au compilateur de créer du hardware pour exécuter des instructions en parallèle. Et il s'agit d'un vrai parallélisme : deux instructions seront exécutées exactement au même instant par deux

structures du hardware bien distinctes (contrairement aux ordinateurs classiques qui en donnent seulement l'illusion).

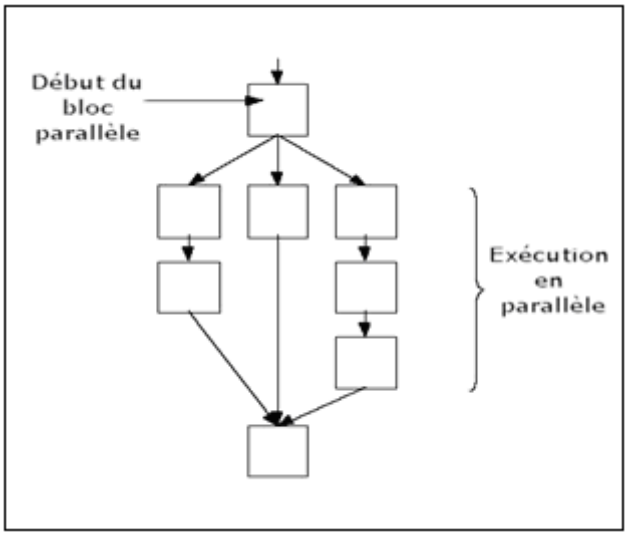

Figure 2.19 : Séparation et regroupage des branches parallèles

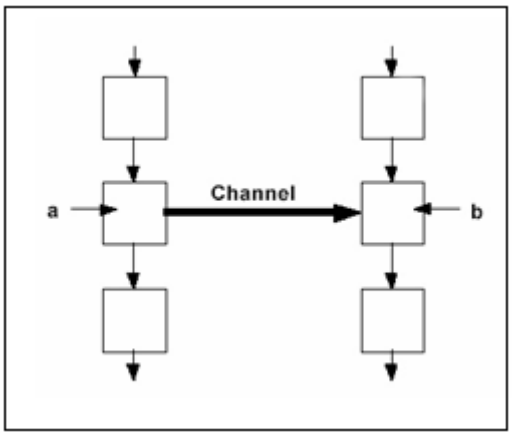

Figure 2.20 : Communication par canal entre branches parallèles.

Si une branche arrive au niveau du canal avant l'autre, elle attend jusqu''a ce que cette dernière soit elle aussi prête pour le transfert de données.

Quand un bloc parallèle est rencontré, chacune de ses branches est exécutée simultanément au départ du bloc en question. Les flots d'exécution parallèles se rejoignent à la fin du bloc quand toutes les branches ont été exécutées. Toute branche se terminant "très rapidement" est obligée d'attendre la plus lente avant de continuer (c'est-à-dire avant que l'on ne sorte du bloc parallèle).

4.3 Les canaux de communication

Les canaux constituent le lien entre les branches parallèles. Une des branches fournit des données sur le canal tandis que l'autre la lit. Les canaux servent aussi à synchroniser les différentes branches parallèles car le transfert de données ne peut avoir lieu que si chacune des branches est prête pour le faire : si la branche émettrice n'est pas prête pour la communication alors la réceptrice doit attendre, et vice versa (voir **figure 2.20**).

4.4 Portée et partage des variables

La portée des déclarations est, comme en C traditionnel, basée autour des blocs de code (un bloc étant encadré par des accolades ...). En gros, cela signifie, d'une part, que les variables globales doivent être déclarées en dehors de tout bloc et, d'autre part, qu'une variable déclarée dans un bloc sera valide dans celui-ci ainsi que dans tous ses sous blocs. **La figure 2.21** nous en donne une illustration.

Puisque les structures parallèles sont de simples blocs de code, une variable peut-être valide dans deux branches parallèles : il suffit par exemple d'avoir un bloc principal ou on déclare sa variable et d'avoir deux sous blocs parallèles. Ceci peut entraîner des conflits de ressource si plusieurs sous blocs parallèles accèdent en même temps à la variable en question. La syntaxe du Handel-C stipule qu'une variable ne peut pas être accédée par plus d'une branche parallèle à la fois.

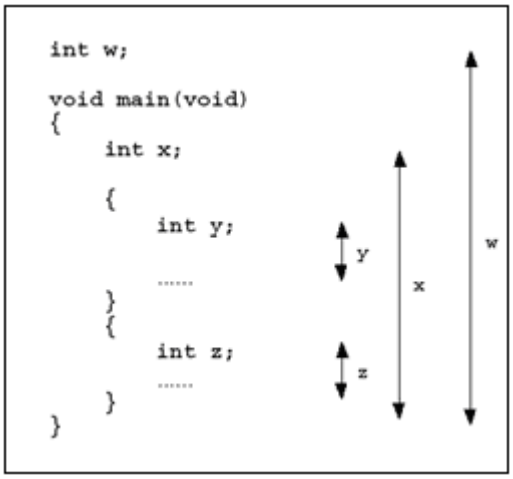

Figure 2.21 : Domaine de validité des variables

4.5 Différences fondamentales entre le Handel-C et le C

Lors du portage d'un programme du C vers le Handel-C, il convient de faire attention, car le Handel-C, bien qu'il soit basé sur la syntaxe du C, comporte tout de même quelques particularités importantes. La liste ci-dessous en pressente les principales :

- ✓ Les flottants (nombres à virgules) n'existent pas en Handel-C. Il n'y a que des entiers.
- ✓ Les pointeurs non plus.
- ✓ Dans un cycle d'horloge, les RAM et les ROM ne peuvent avoir qu'une seule entrée accédée. Ainsi deux branches parallèles ne peuvent pas accéder à une RAM en même temps par exemple.
- ✓ Lors de l'accès des tableaux, l'index doit être une constante définie explicitement avant la compilation.

Ayant un tableau tab, il est donc impossible d'écrire une boucle sur une variable n pour accéder à tab[n].

Conclusion Etant donné que nous devons porter un programme du C vers le Handel-C, il nous faudra donc trouver des moyens de contourner ces restrictions ci-dessus.

5 Etapes nécessaires au développement d'un projet sur FPGA

Le développement en Handel-C nécessite l'utilisation de deux outils : le simulateur et le synthétiseur. Le premier va nous permettre de simuler notre description ; cet outil interprète directement le langage Handel-C et il comprend l'ensemble du langage. L'objectif du synthétiseur est très différent : il doit traduire le comportement décrit en Handel-C en fonctions logiques de bases, celles-ci dépendent de la technologie choisie ; cette étape est nommée « synthèse ». L'intégration finale dans le circuit cible est réalisée par l'outil de placement et routage. Celui-ci est fourni par le fabricant de la technologie choisie.

Le langage Handel-C permet d'écrire des descriptions d'un niveau comportemental élevé. La question est de savoir si n'importe quelle description comportementale peut être traduite en logique ?

Avec les outils actuels, il est possible de disposer des fichiers résultat pour chaque étape. Le même fichier de simulation est ainsi utilisable pour vérifier le fonctionnement de la

description à chaque étape de la procédure de développement. **La figure 2.22** donne les différentes étapes nécessaires au développement d'un projet sur circuit FPGA. [15]

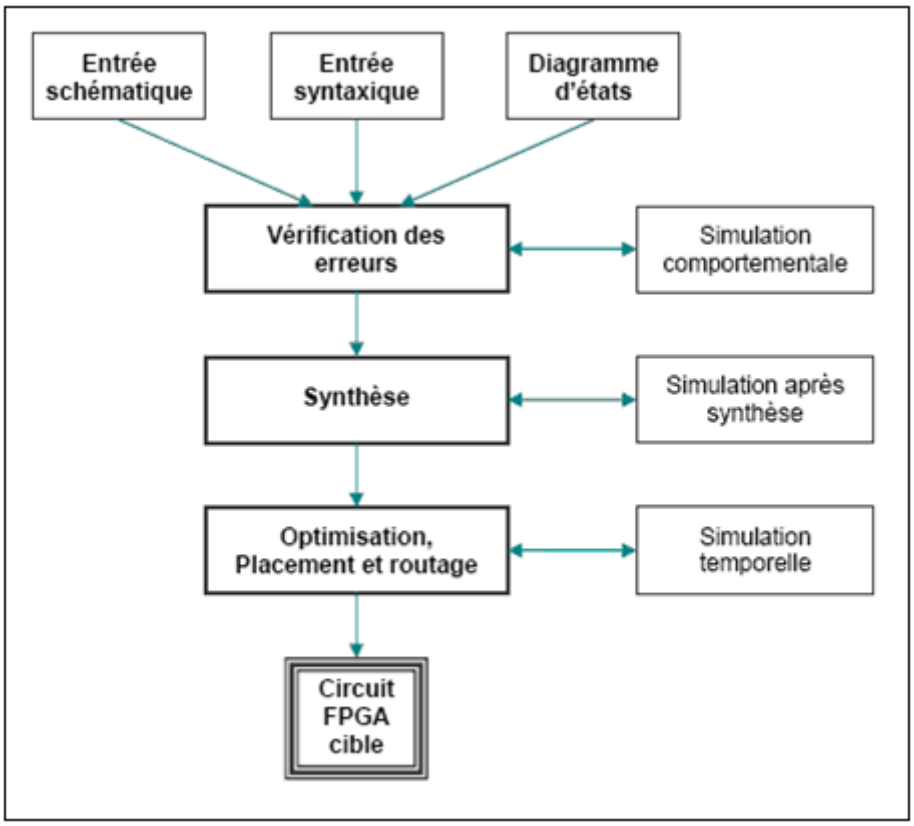

Figure 2.22 : Organisation fonctionnelle de développement d'un projet sur circuit FPGA

5.1 Saisie du texte Handel-C

La saisie du texte Handel-C se fait sur le logiciel «DK». Ce logiciel propose une palette d'outils permettant d'effectuer toutes les étapes nécessaires au développement d'un projet sur circuit FPGA. Il possède également des outils permettant de mettre au point une entrée schématique ou de créer des diagrammes d'état, qui peuvent être utilisés comme entrée au lieu du texte Handel-C.

La figure 2.23 montre comment se présente le logiciel «DK».

La saisie du texte se fait sur la partie droite de l'écran, on voit en haut à gauche la hiérarchie du projet.

Il faut commencer par créer un projet, ensuite inclure des fichiers sources dans lesquels il faut saisir le texte Handel-C désiré. On peut inclure autant de sources qu'on veut dans un projet.

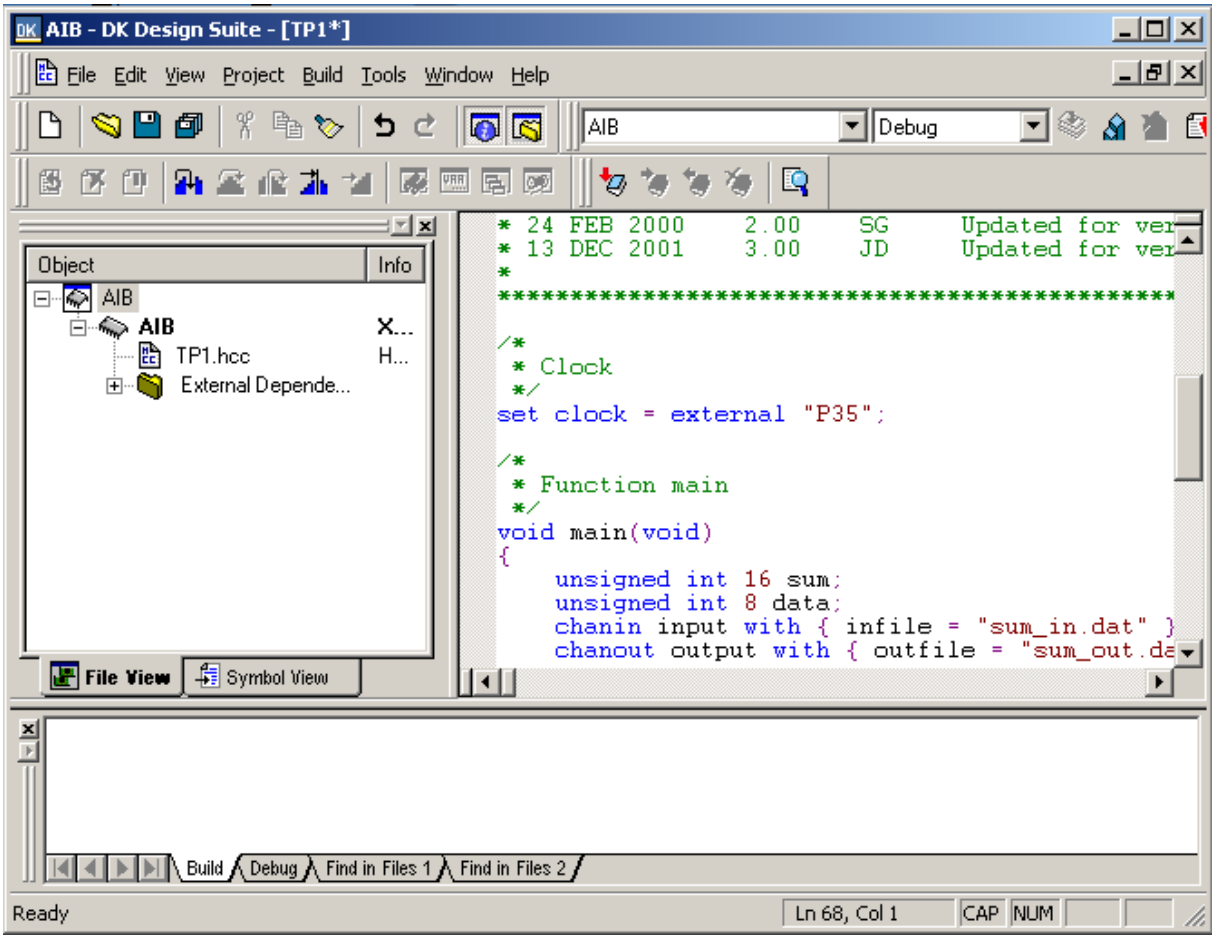

Figure 2.23 : Vue d'ensemble du logiciel « DK »

5.2 Vérification des erreurs

Cette étape est effectuée en appuyant sur le bouton « CTRL+F7 » ou sur le menu Build→compile. Elle permet de vérifier les erreurs (errors) de syntaxe du texte Handel-C et d'afficher les différentes alarmes « warnings » liées au programme. S'il y'a des erreurs dans le programme, il ne peut pas être synthétisé, mais la présence d'alarmes n'empêche pas de poursuivre normalement les autres étapes du développement.

Contenant la description cette étape permet donc de valider la syntaxe du programme et de générer la « netlist », qui est un fichier de l'application sous forme d'équations logiques.

5.3 Synthèse

La synthèse permet de réaliser l'implémentation physique d'un projet. Le synthétiseur a pour rôle de convertir le projet, en fonction du type du circuit FPGA cible utilisé, en portes

logiques et bascules de base. Pour nous le résultat de l'étape de synthèse est un fichier EDIF. Donc on va choisir sur DK le type de fichier de sortie puis on démarre la synthèse à partir de Build→Build nom de projet.

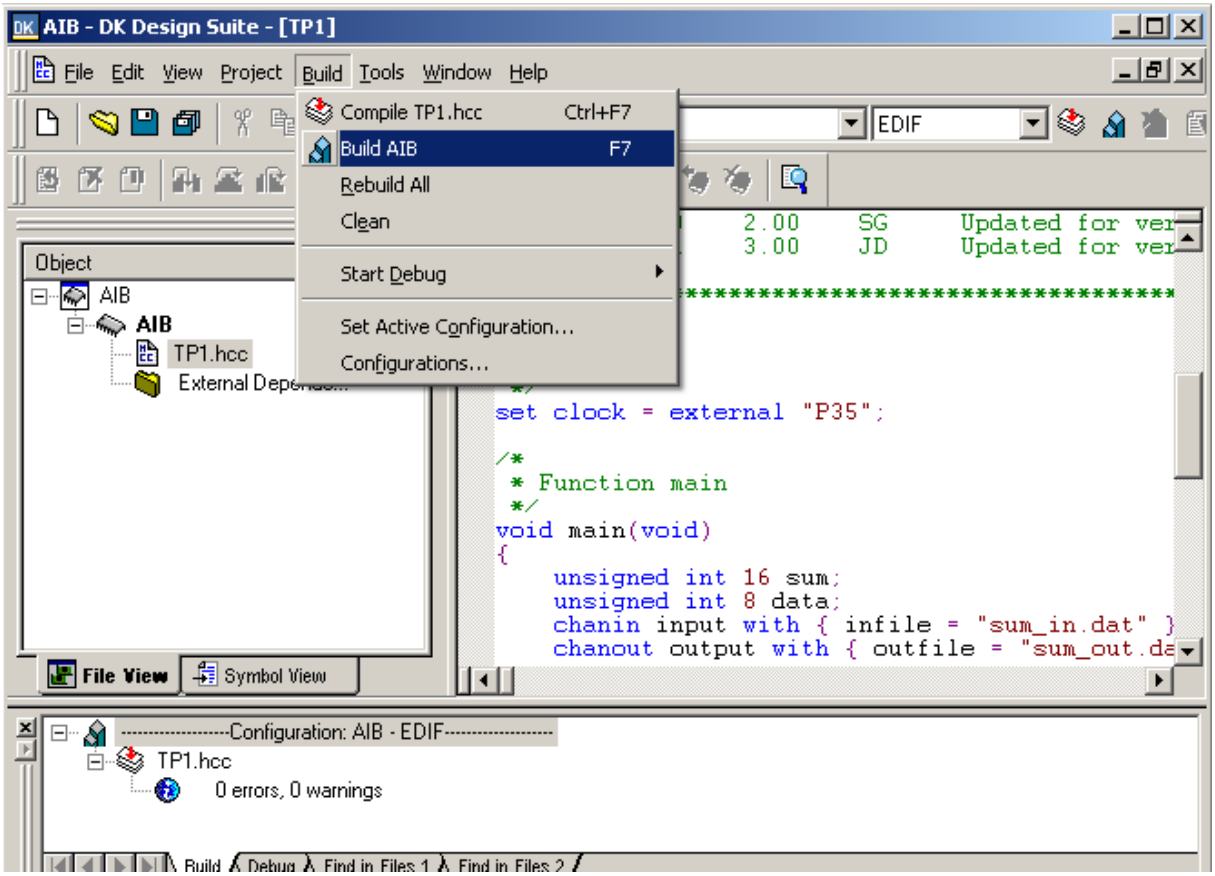

Figure 2.24 : Aperçu de l'étape de synthèse sur DK

De plus, le synthétiseur permet à l'utilisateur d'imposer des contraintes de technologie (User constraints) : par exemple fixer la vitesse de fonctionnement (Create Timing Constraints), délimiter la zone du circuit FPGA dans laquelle le routage doit se faire (Create Area constraints) ou affecter les broches d'entrées/sorties (Assign Package Pins).

5.4 Simulation

L'outil de simulation utilisé est le « PAL virtual Platform » (**figure 2.25**). La simulation permet de vérifier le comportement d'un design avant implémentation dans le composant cible.

Elle représente une étape essentielle qui nous fera gagner du temps lors de la mise au point sur la carte. Il faut juste noter qu'un projet peut être simulé même s'il n'est pas synthétisable.

Lors de l'étape de simulation comportementale, on valide l'application indépendamment de l'architecture et des temps de propagation du futur circuit cible. La phase de simulation après synthèse valide l'application sur l'architecture du circuit cible, et enfin la simulation temporelle prend en compte les temps de propagation des signaux à l'intérieur du circuit FPGA cible.

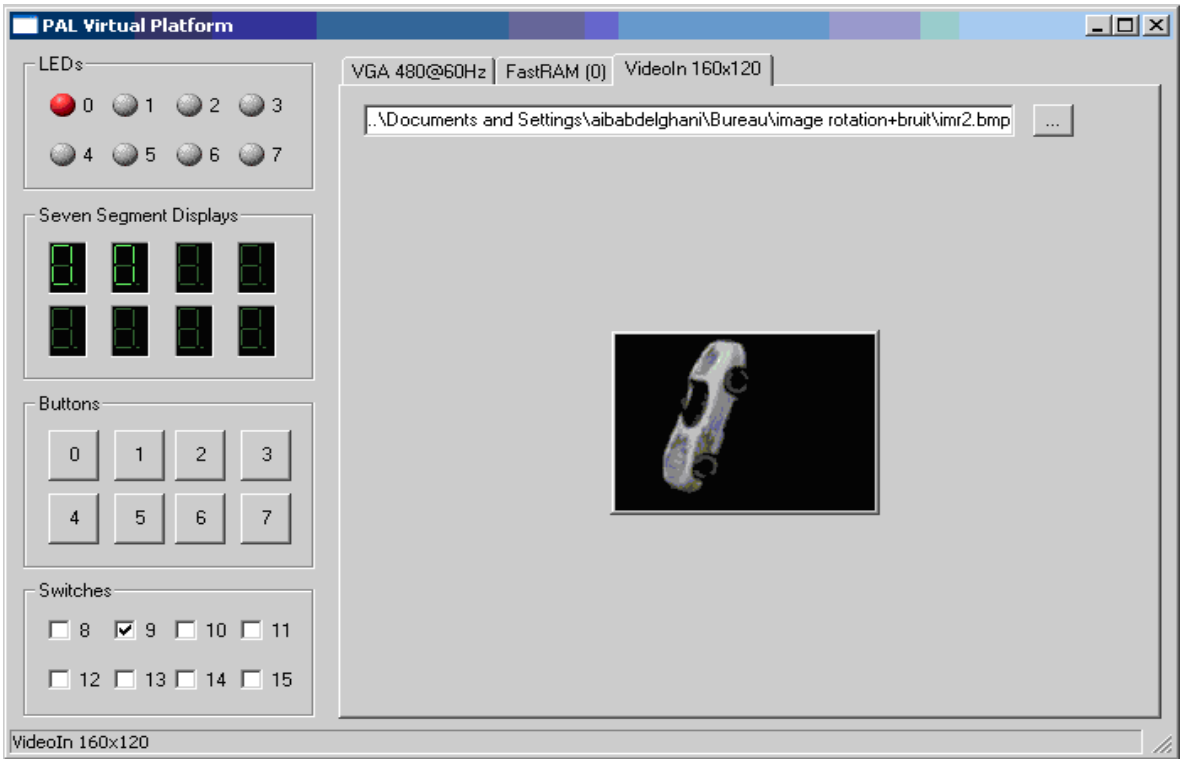

Figure 2.25 : L'outil de simulation « PAL virtual Platform »

5.5 Optimisation, placement et routage

Pendant l'étape d'optimisation, l'outil cherche à minimiser les temps de propagation et à occuper le moins d'espace possible sur le circuit FPGA cible. Le placement et routage permettent de tracer les routes à suivre sur le circuit afin de réaliser le fonctionnement attendu. **La figure 2.26** donne un aperçu de l'outil de placement et routage « FPGA Editor » (ISE de Xilinx) qui permet de visualiser et d'éditer le circuit routé.

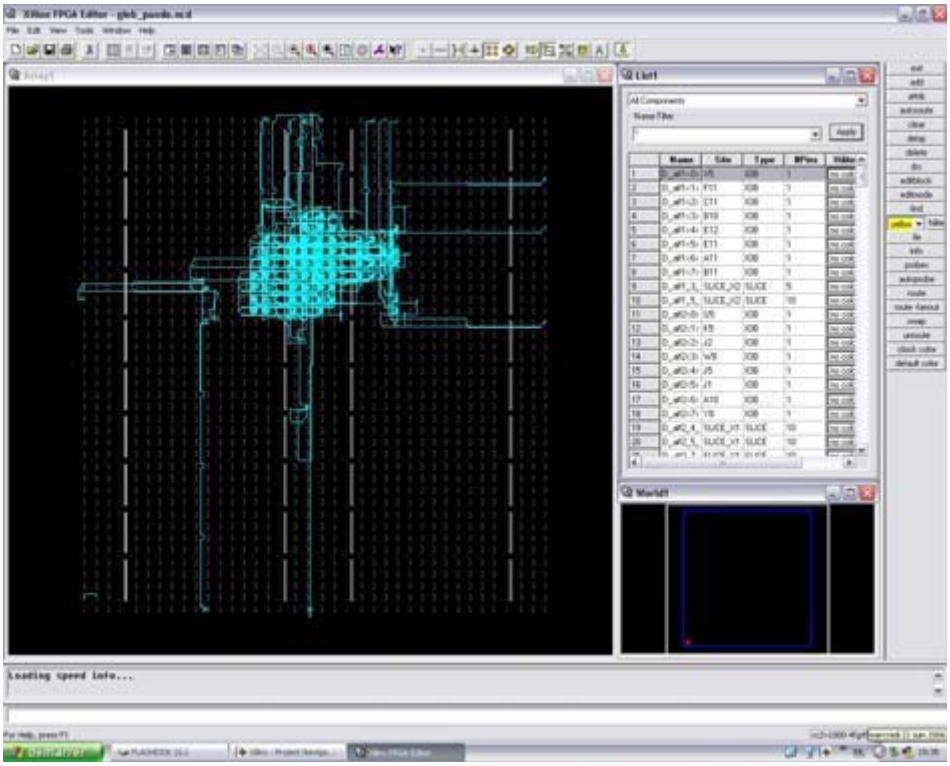

Figure 2.26 : Aperçu de l'outil « FPGA Editor »

5.6 Programmation du composant et test

Dans cette dernière étape (Generate Programming Files), on génère le fichier à charger sur le circuit FPGA à travers l'interface JTAG. Une fois le programme chargé sur le circuit, on peut tester et visualiser les résultats directement sur la carte de développement Virtex-II à travers les nombreuses interfaces qu'elle offre, soit directement sur les deux afficheurs 7 segments, soit à travers l'interface RS232 ou bien à travers l'interface LVDS 16 bits.

6 Conclusion

Dans le présent chapitre on a présenté l'architecture des circuits FPGA ainsi que le principe de leurs fonctionnements et les étapes de leurs programmations, dans le chapitre suivant en vont entamer l'implémentation software de notre algorithme de vision.

Implémentation software

1 Introduction

Dans ce chapitre, nous présentons les différentes étapes que nous avons effectuées sur la reconnaissance des images avec les invariants de Hu décrits dans le Chapitre 1. Ces invariants ont l'avantage d'avoir un fort taux de robustesse vis-à-vis de la translation et de la rotation. Pour réaliser notre logiciel, on a utilisé C++Builder6 comme environnement de programmation.

2 C++Builder6

Tout d'abord C++ est un outil RAD, c'est-à-dire tourné vers le développement rapide d'application (Rapid Application Development) sous Windows. En un mot, C++Builder6 permet de réaliser de façon très simple l'interface des applications et de relier aisément le code utilisateur aux événements Windows, quelle que soit leur origine (sourie, clavier, événement système, etc.…).

Pour ce faire C++Builder6 repose sur un ensemble très complet de composants visuels prêts à l'emploi. La quasi-totalité des contrôles de Windows (boutons, boites de saisies, listes déroulantes, menus et autres barres d'outils) y sont représentés, regroupés par famille. Leurs caractéristiques sont éditables directement dans une fenêtre spéciale intitulée éditeur d'objets. L'autre volet de cette même fenêtre permet d'associer du code au contrôle sélectionné [20].

Avec C++Builder6 on a utilisé la bibliothèque de développement 3D OpenGL (Open Graphic Library) pour réaliser une animation 3D d'un robot à quatre degré de liberté qui peut être commander directement par les résultats de notre procédure de reconnaissance.

3 La reconnaissance des images avec les invariants de Hu

Les invariants Hu sont des descripteurs géométriques de la répartition de luminosité des pixels dans l'image. Ils ont l'avantage d'être moins sensible au rotation et translation des objets dans l'image. Dans notre application on va utiliser les trois premiers invariants de Hu décrits dans les équations (1.11), (1.12) et (1.13).

3.1 Schéma global de la procédure de reconnaissance

La figure suivante représente l'organigramme détaillé de la procédure de reconnaissance avec toutes les tâches nécessaires pour une implémentation software.

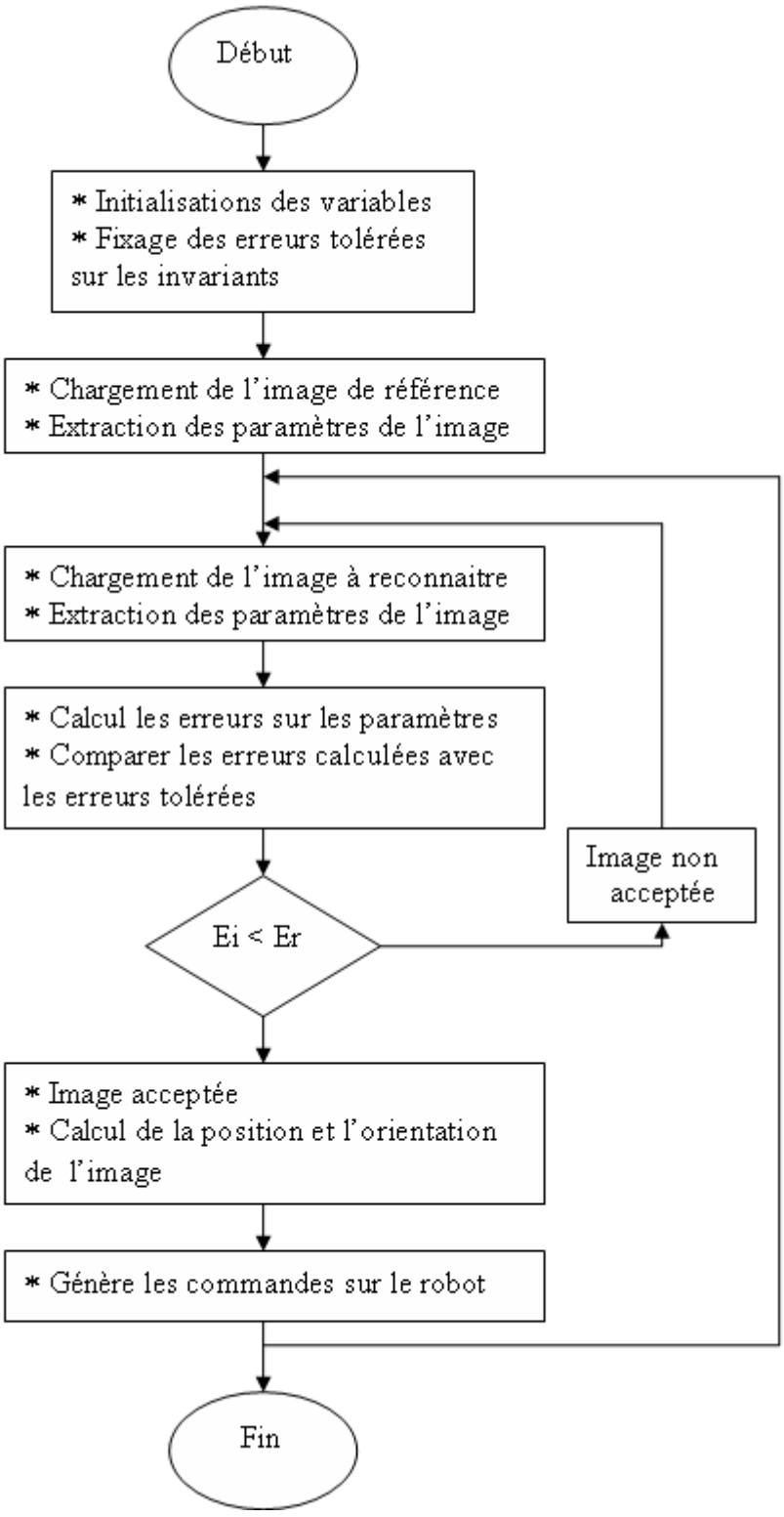

Figure 3.1 Schéma descriptif de la procédure de reconnaissance

3.2 Etude de performance sur les invariants de Hu

3.2.1 Effet de la translation

Afin d'étudier les effets de la translation, nous avons utilisé 16 clichés représentant un objet translaté sur un fond noire. Quelques exemples de ces images sont données à **la figure:3.2.** Nous avons calculé les 3 premiers invariants pour chaque image puis nous avons calculé la moyenne de l'erreur relative. L'erreur relative est calculée de la manière suivante:

$$E = \frac{|x - x_0|}{x_0} \times 100 \qquad (3.1)$$

Où x est la valeur d'un invariant pour une image ayant subie une translation et x_0 est la valeur calculée pour l'image originale.

Nous nous sommes limités à 3 invariants car comme l'affirme [19], l'erreur engendrée par des invariants d'ordre plus élevée est très importante. L'erreur relative a été reportée à **la figure3.4.** L'erreur maximale a été obtenue pour l'invariant d'ordre 1, elle est de 3.5%.

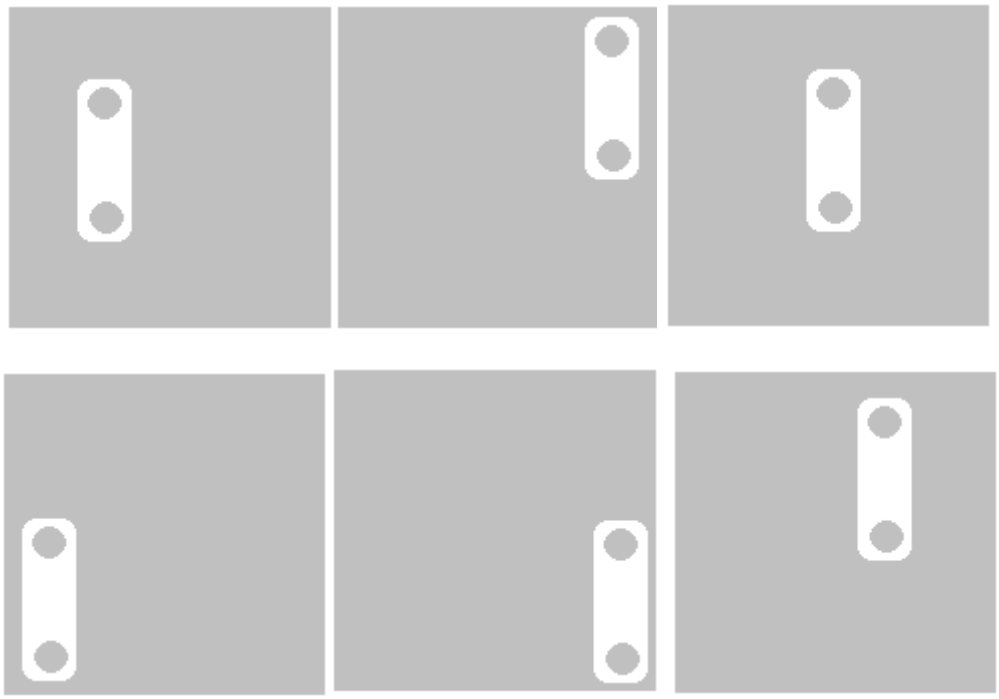

Figure 3.2 Exemple d'images utilisées (translation)

3.2.2 Effet de la rotation

De la même manière, pour étudier l'influence de la rotation sur les invariants de Hu, nous avons réalisé la rotation d'un objet par rapport à son centre de gravité par pas de15°. Quelques exemples d'images utilisées sont visibles sur **la figure 3.3.** Nous avons calculé les trois premiers invariants. L'erreur relative (**figure:3.5**) est au maximum de 7% pour l'invariant n°1.

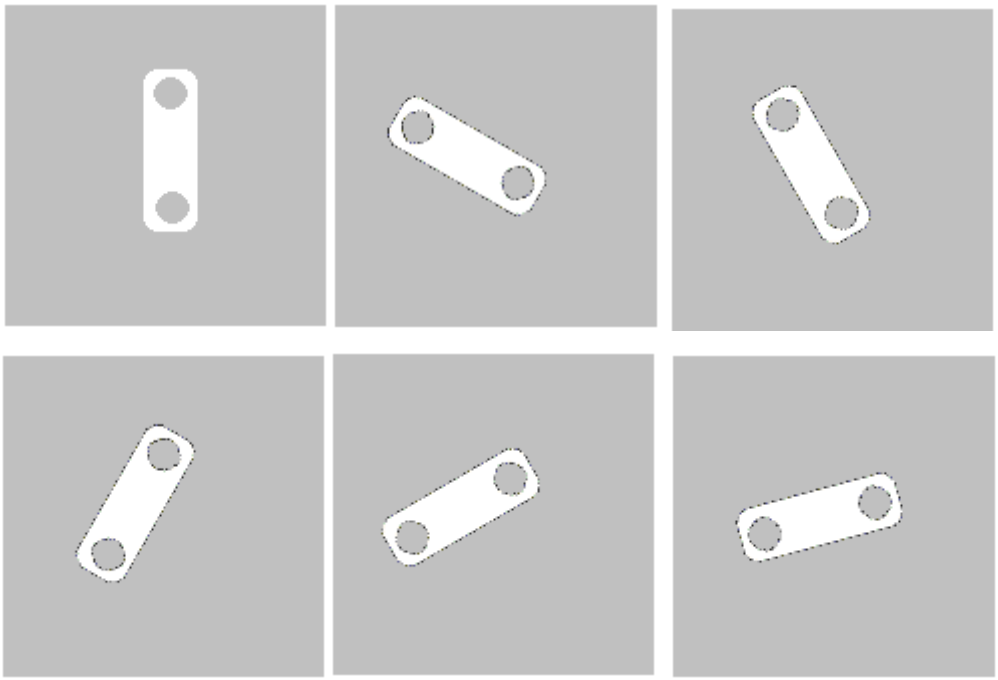

Figure 3.3 Exemple d'images utilisées (rotation)

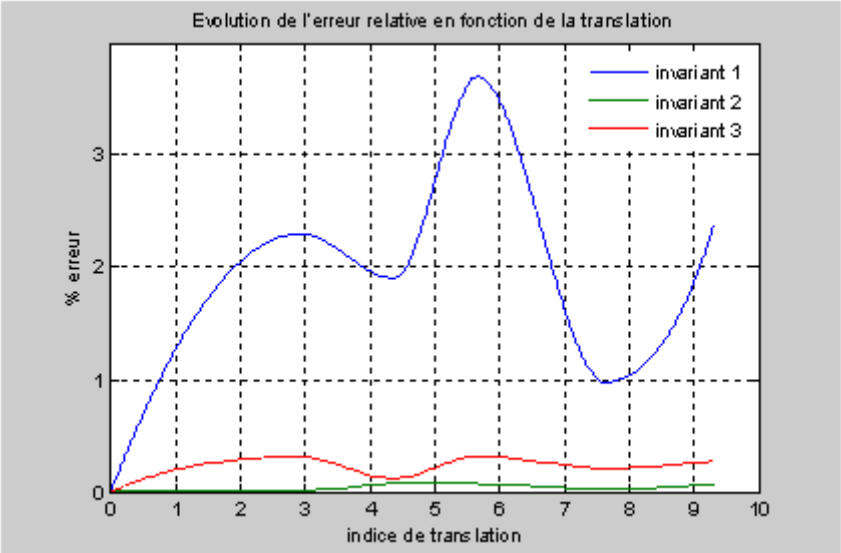

Figure 3.4 Invariants Hu: erreur relative (translation)

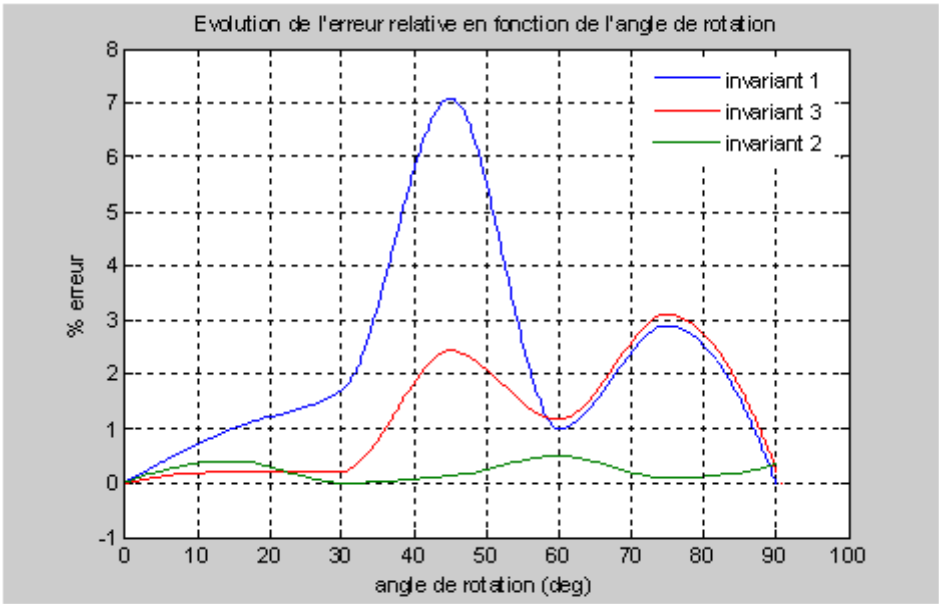

Figure 3.5 Invariants Hu: erreur relative (rotation)

3.3 Commande d'un robot manipulateur

Notre but est de convertir les résultats de l'étape de reconnaissance (les coordonnes de la pièce et son orientation) vers des consignes pour l'asservissement d'un robot manipulateur à quatre degrés de libertés de type (RRTR).

3.3.1 La structure physique du robot

Le robot est constitué de trois segments, un premier segment qui est la base du robot a une articulation en rotation, un deuxième segment relié avec le premier par une articulation en rotation, et un troisième segment qui peut glissé (translation) dans le deuxième segment. Enfin, poignet avec une rotation (Roll) est relié au troisième segment.

La figure suivante pressente la structure physique de robot avec les différents mouvements de rotations et de translation.

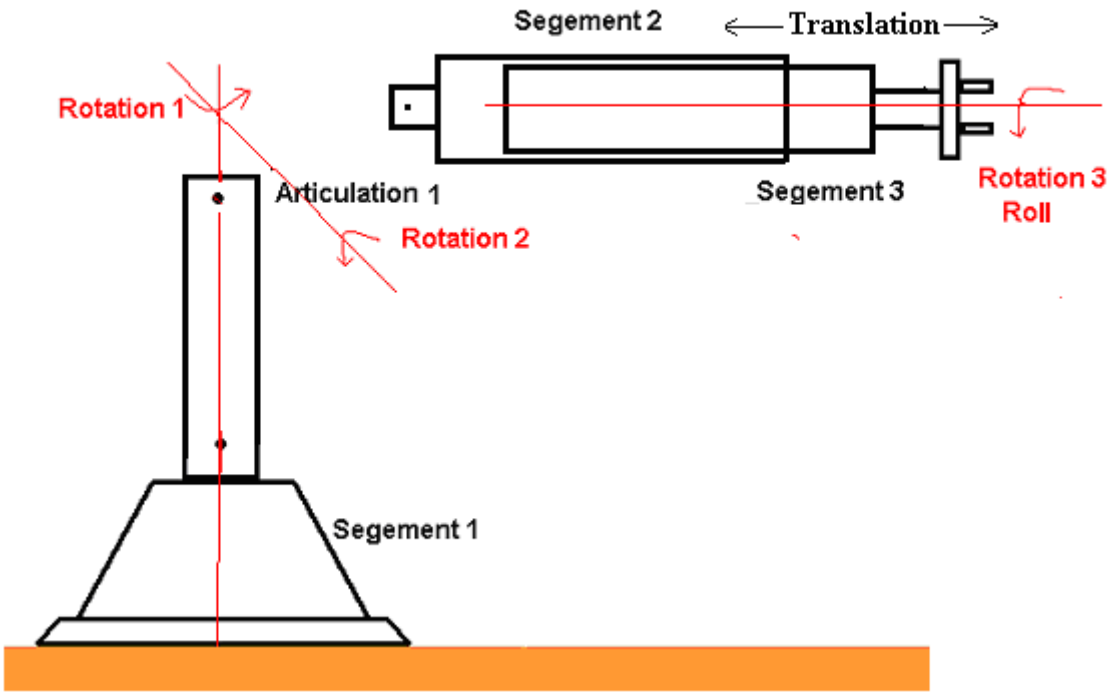

Figure 3.6 La structure physique de robot

3.3.2 Génération des consignes

La génération des consignes consiste a transformer les coordonnées cartésiennes et l'angle de rotation de la pièce dans l'image en des angles de rotation et indice de translation du robot.

La figure suivante donne les entrées sorties de la procédure de génération des consignes. (Modèle cinématique inverse)

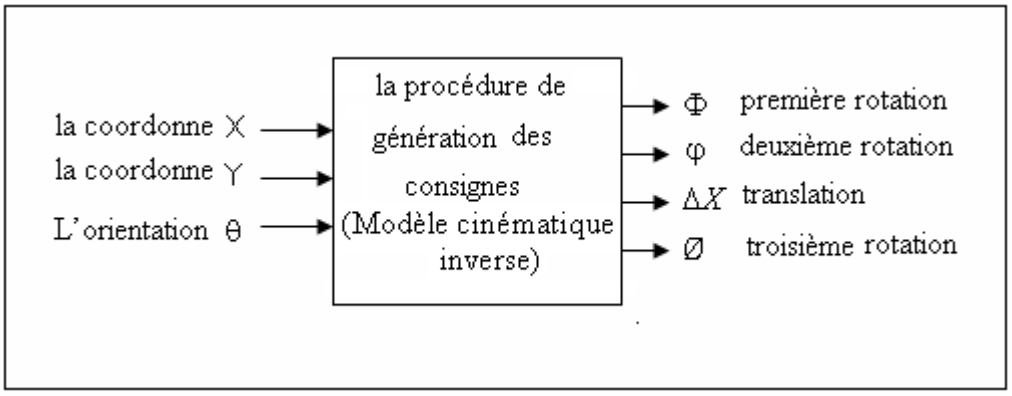

Figure 3.7 La procédure de génération des consignes

- **Calcul de première rotation (Φ)**

La figure suivante présente la projection du premier angle de rotation dans le plan de l'image.

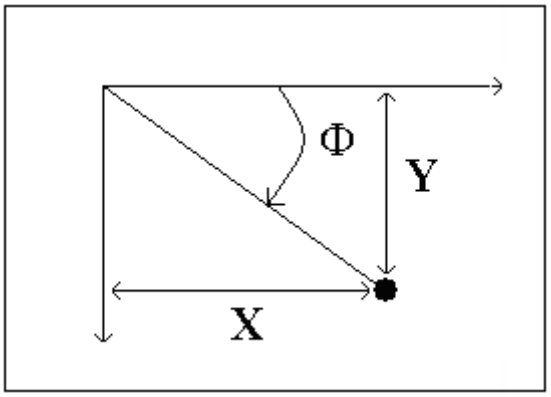

Figure 3.8 Le premier angle dans le plan d'image

Donc

$$\Phi = arctg(\frac{Y}{X}) \qquad (3.2)$$

- **Calcul de la deuxième rotation (φ)**

Dans la figure suivante on a le deuxième angle de rotation (φ) dans l'espace 3D.

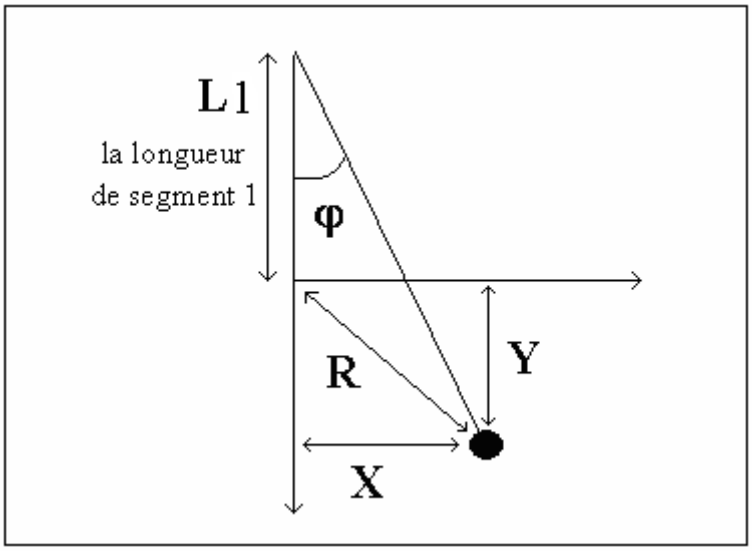

Figure 3.9 Le deuxième angle dans le plan d'image

On a $R = \sqrt{\left(X^2 + Y^2\right)}$ d'où

$$\varphi = arctg\left(\frac{R}{L1}\right) = arctg\left(\frac{\sqrt{\left(X^2 + Y^2\right)}}{L1}\right) \qquad (3.3)$$

- **Calcul de translation (ΔX)**

La figure suivante présente la translation de troisième segment dans l'espace 3D.

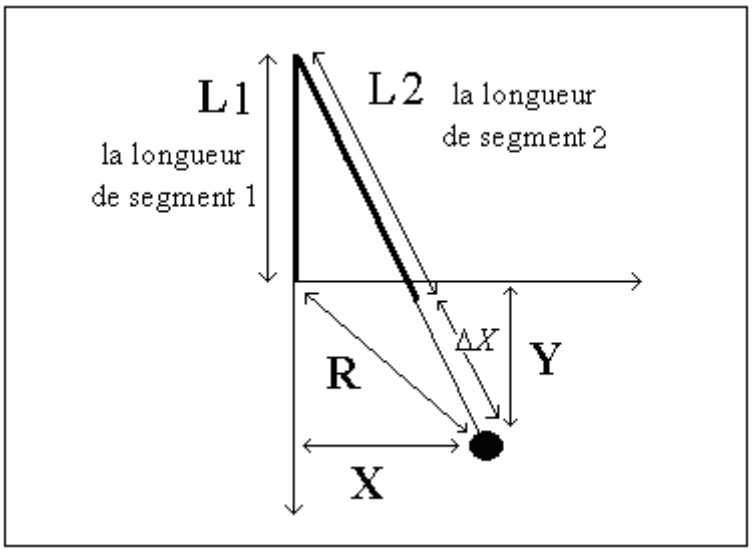

Figure 3.10 La translation dans le plan d'image

On a $\left(L2 + \Delta X\right)^2 = \left(L1\right)^2 + R^2$ d'où

$$\Delta X = \sqrt{\left((L1)^2 + R^2\right)} - L2 = \sqrt{\left((L1)^2 + X^2 + Y^2\right)} - L2 \qquad (3.4)$$

- **Calcul de la troisième rotation (Ø)**

La commande de la rotation de pince (la troisième rotation) se fait directement par l'angle d'orientation de la pièce. Soit ϕ^r l'angle de rotation pour la pièce de référence donc :

$$\phi = \phi^r + \theta \qquad (3.5)$$

3.4 Résultats d'implémentation

Dans cette partie on va détailler notre procédure de reconnaissance. La figure suivante donne un aperçu sur notre application.

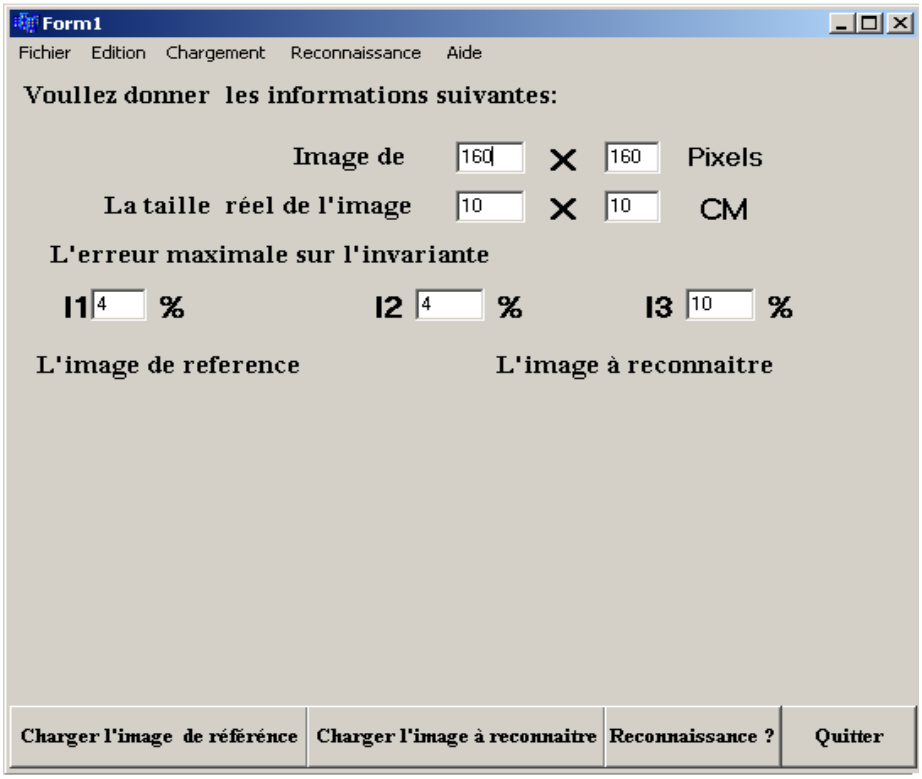

Figure 3.11 Aperçu de procédure de la reconnaissance

L'utilisation de notre application ce fait en trois étapes :

- **Etape d'apprentissage**

Dans cette étape on va fixer la résolution des images, la taille réelle des images et l'erreur tolérée sur les invariants. L'utilisateur peut utiliser les paramètres par défaut de notre application puis en charger l'image de référence et l'image à reconnaître.

La figure suivante présente la fenêtre de cette étape :

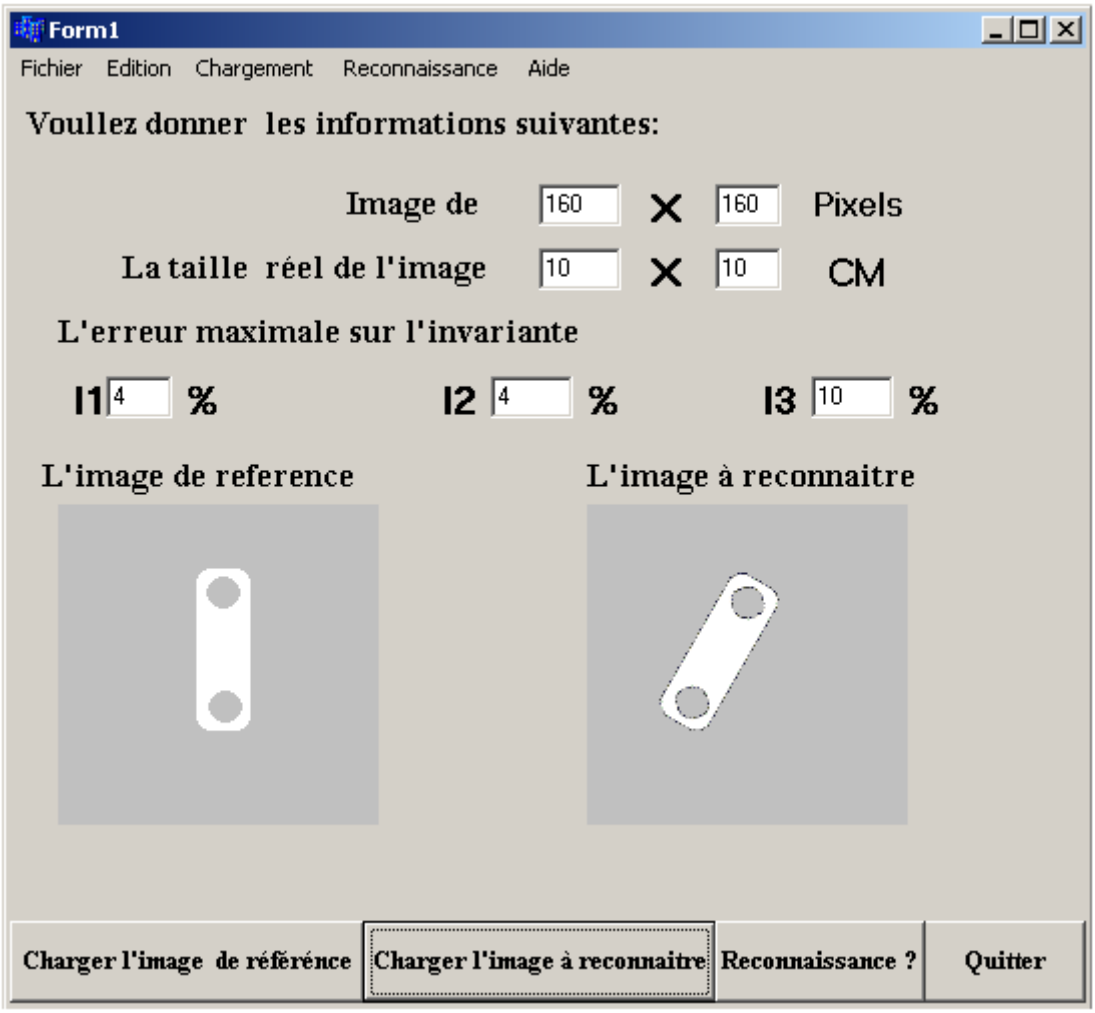

Figure 3.12 La fenêtre de l'étape d'apprentissage

- **Etape de reconnaissance**

Dans cette étape la procédure d'extraction des paramètres va démarrer pour donner les invariants de chaque image et à partir de ces invariants on calcul les erreurs sur les paramètres. Ces erreurs sont ensuite comparées avec les erreurs tolérées fixées par l'utilisateur.

Deux cas sont envisagés, soit au moins une des erreurs calculée est supérieure à l'erreur tolérée et dans ce cas la fenêtre suivante va apparaître :

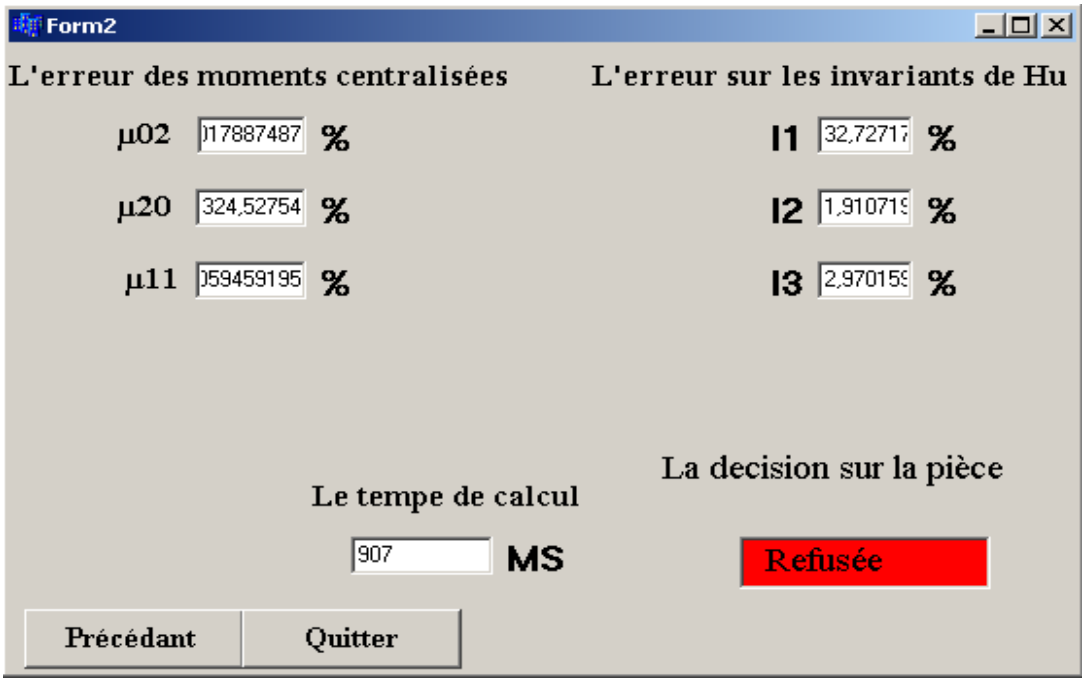

Figure 3.13 Fenêtre affichée en cas d'une image refusée

Soit toutes les erreurs calculées sont inférieures aux erreurs tolérées dans ce cas la fenêtre active donne la position et l'orientation de la pièce dans l'image. La figure suivante donne la fenêtre affichée dans ce cas.

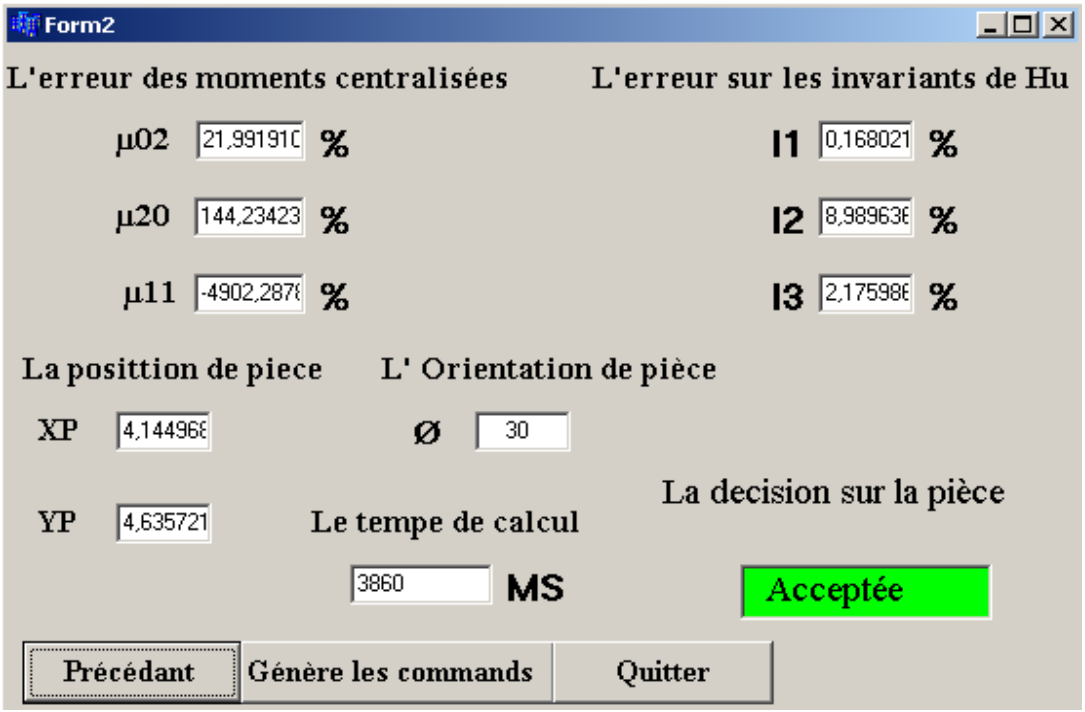

Figure 3.14 Fenêtre affichée en cas d'une image acceptée

- **Etape de génération des commandes**

Cette étape n'a pas lieu d'être que si la pièce est acceptée, dans ce cas, on doit convertir les coordonnées cartésiennes et l'angle de rotation obtenues par l'étape précédente vers des commandes de mouvements du bras.

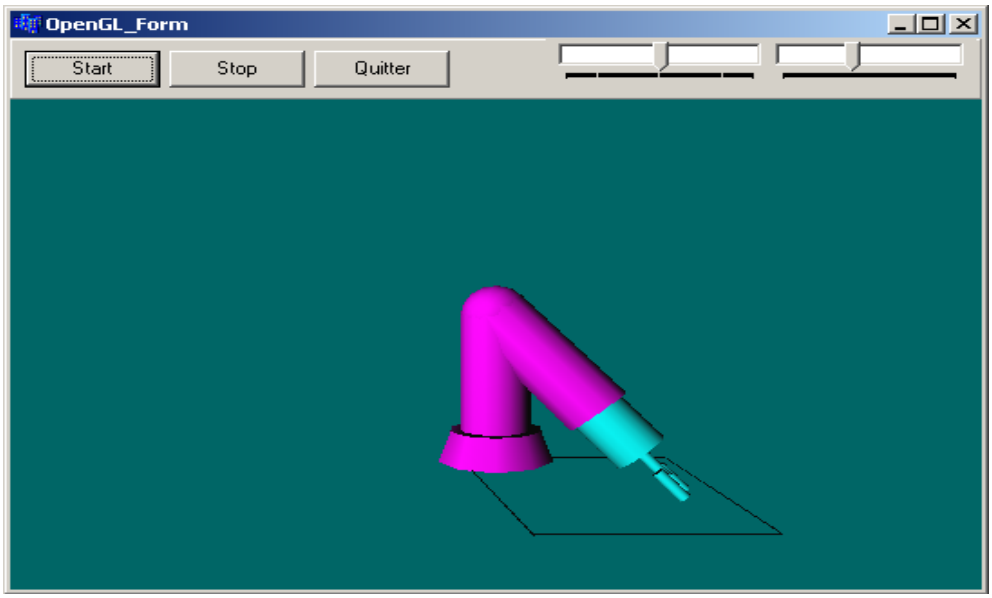

Figure 3.15 Le bras après la génération des commandes

Implémentation hardware

1 Introduction

Pour l'implémentation hardware d'une procédure de reconnaissance on a utilisé l'outil de développement DK/PDK avec le langage Handel-C. Ces outils permettent de réaliser une architecture temps réel parallèle sur la carte RC203E de Celoxica avec un haut niveau d'abstraction. La figure suivante présente l'organigramme de fonctionnement de notre architecture.

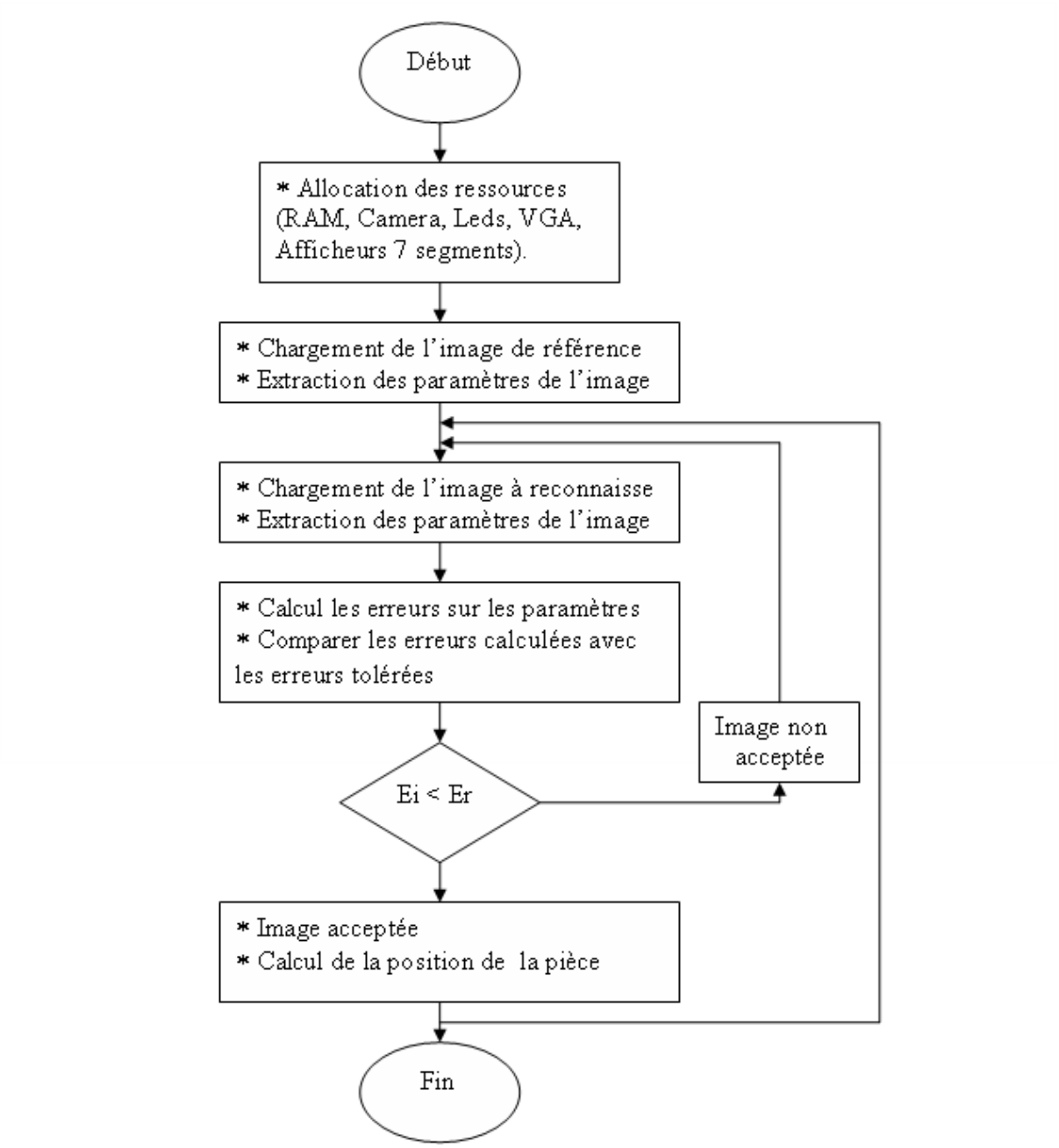

Figure 4.1 Schéma descriptive de l'architecture de reconnaissance sur FPGA

2. Les étapes d'implémentation

2.1 L'étape d'apprentissage

Dans cette étape l'image sera acquise à partir d'une ressource externe, une camera ou une capteur d'images par exemple. Pour ce la on va concevoir un macro programme pour la réception et le stockage des pixels dans la mémoire avec leurs adresses adéquat.
Les deux fonctions principales de ce macro programme sont la fonction de lecture des pixels à partir de ressource vidéo et la fonction chargement de la mémoire.
La lecture d'un pixel à partir de VideoIn est assurée par la fonction suivante :

```
PalVideoInRead (VideoIn, &(X), &(Y), & (Pixel));
```

Le chargement de la mémoire ce fait par la fonction suivante :

```
PalFrameBuffer16Write (FBPtr, X, Y, Pixel);
```

Tel que VideoIn est une ressource vidéo programmée en entrée et FBPtr est un pointeur vers une ressource mémoire pour le stockage des pixels.

2.2 L'étape d'extraction des paramètres

Pour l'extraction des caractéristiques de l'images qui sont les invariants de Hu dans notre cas on a utilisé une macro procédure, sa fonction est de calculer les invariants de Hu et la position de la pièce dans l'image. Le prototype de cette macro est le suivant :

```
static macro proc Draw (VideoIn, FBPtr, VideoOut, xrp,yrp,i1,i2);
```

Tel que FBPtr est un pointeur vers la mémoire qui contient l'image, xrp et yrp sont les cordonnées du centre de gravité de la pièce, i2 et i2 sont le premier et le deuxième invariant de Hu, respectivement.
Tout d'abord on va lire les pixels à partir de la mémoire avec la fonction suivante :

```
PalFrameBuffer16Read (FBPtr, x, y, & (Colour));
```

Tel que Colour est un pixel en RGB (24 bits : rouge, vert et bleu).

Pour transformer un pixel couleur de format RGB (24 bits) en niveaux de gris (8 bits) on utilise la formule suivante [4] :

$$\mathrm{Im} = 0.3 * R + 0.59 * G + 0.11 * B \qquad (4.1)$$

Puis on calcul les trois moments géométriques suivants M_{00}, M_{10} *et* M_{01} à partir de la forme générale suivante :

$$M_{pq} = \sum\sum \mathrm{Im}(x, y) x^p y^q \qquad (4.2)$$

A travers ces moments on peut déduire la position du centre de gravité de la pièce par les relations suivantes :

$$\begin{cases} x_0 = \dfrac{M_{10}}{M_{00}} \\ y_0 = \dfrac{M_{01}}{M_{00}} \end{cases} \qquad (4.3)$$

Le calcul des moments centralisés d'ordre p+q se fait par l'équation (4.4).

$$\mu_{pq} = \sum\sum \mathrm{Im}(x, y)(x - x_0)^p (y - y_0)^q \qquad (4.4)$$

Enfin les deux premiers invariants de Hu sont donnés par les deux relations suivantes :

$$\phi_1 = \mu_{20} + \mu_{02} \qquad (4.5)$$

$$\phi_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}{}^2 \qquad (4.6)$$

2.3 L'étape de décisions

Dans cette étape on calcul les erreurs relatives engendrées par les invariants par la relation suivante :

$$e_i = 100 \times \left(\frac{\phi_i^r - \phi_i}{\phi_i^r}\right) \qquad (4.7)$$

Avec e_i est l'erreur relative d'ordre i, ϕ_i^r et ϕ_i sont les invariants d'ordre i de l'image de référence et l'image à reconnaître respectivement.

Ces erreurs sont comparées avec les erreurs tolérées fixées préalablement pour avoir une décision sur la pièce.

3 La simulation

La simulation permet de valider le projet à chaque étape de développement, c'est donc dans cette étape qu'on peut visualiser les résultats obtenus par les macro développés précédemment. Pour cela, on a utilisé l'outil de simulation **« PAL virtual Platform »**. Grâce à son niveau d'abstraction il est très utilisé pour la simulation des circuits développés en FPGA surtout celles de la famille Celoxica.

Nous allons simuler chaque macro à part pour s'assurer de son bon fonctionnement. Puis nous passerons à la simulation du système global. On a utilisé des images320x240 pixels.

La simulation consiste à forcer les entrées d'une entité et voir si les sorties sont en accord avec les résultats voulus, ce qui justifie le comportement du système décrit. De plus le simulateur nous permet de vérifier chaque variable interne pour déceler et corriger d'une façon efficace une éventuelle anomalie.

La figure suivante présente l'outil de simulation avec toutes les ressources nécessaire pour notre architecture.

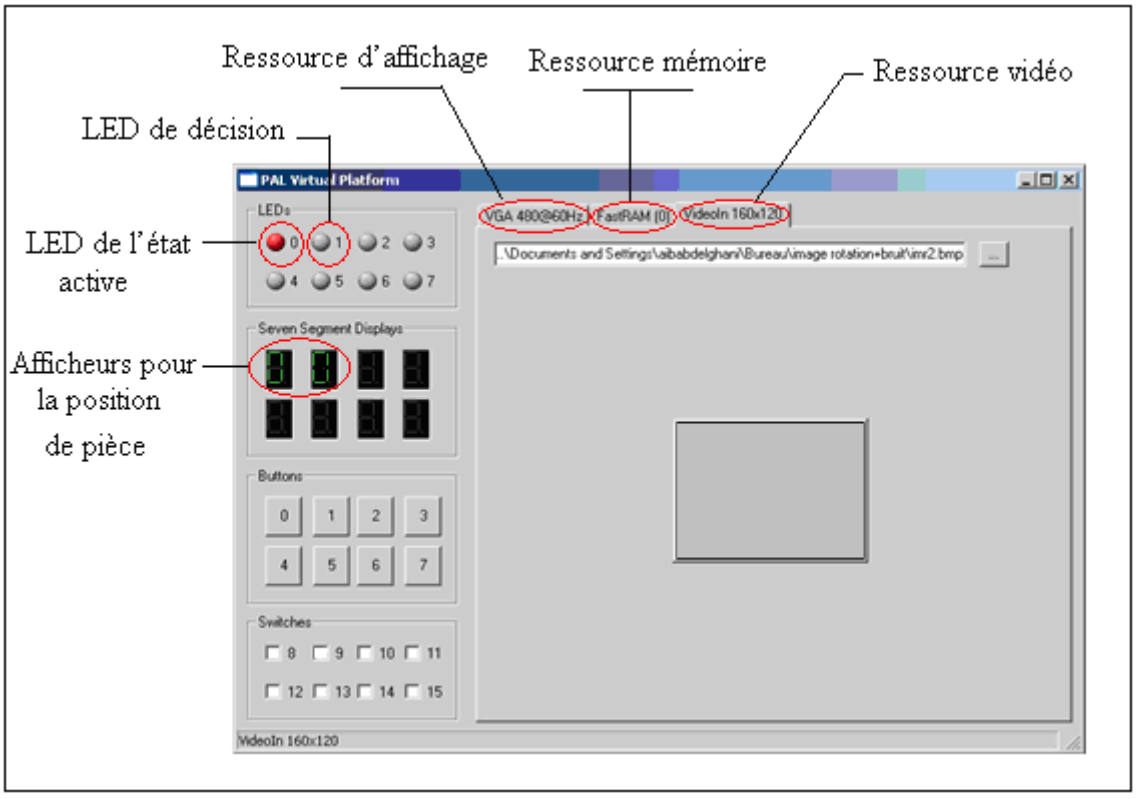

Figure 4.2 Les ressources utilisées sur le simulateur

- **Simulation de l'étape d'apprentissage**

L'apprentissage sur le simulateur consiste à choisir une image à partir du choix de son chemin d'accès, puis le simulateur va considère, cette image comme une image vidéo entrante.

Les figures suivantes donnent l'état du simulateur dans l'étape d'apprentissage.

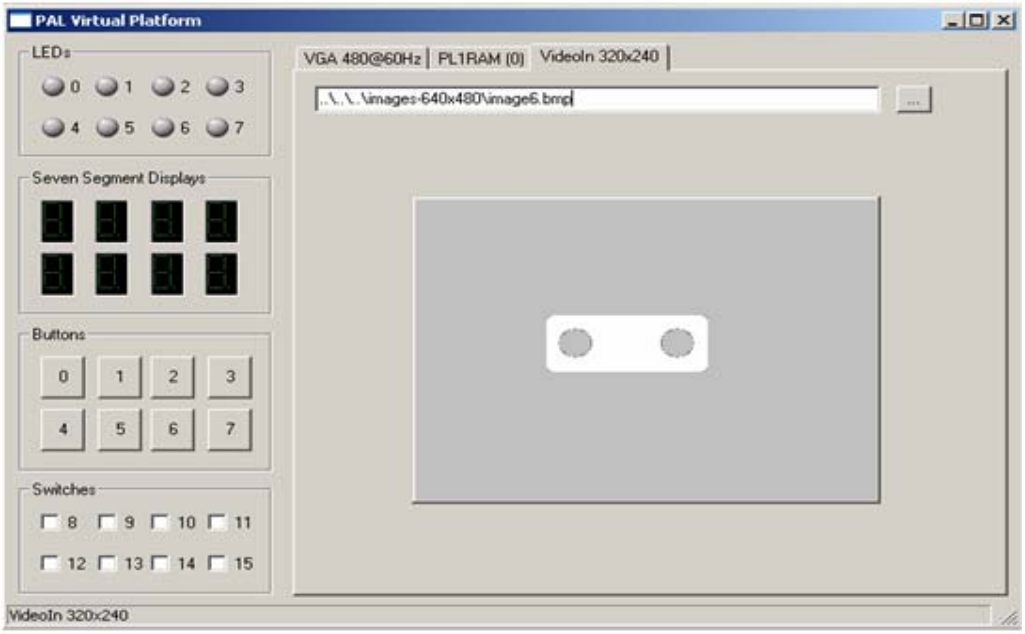

Figure 4.3 Chargement de l'image entrante

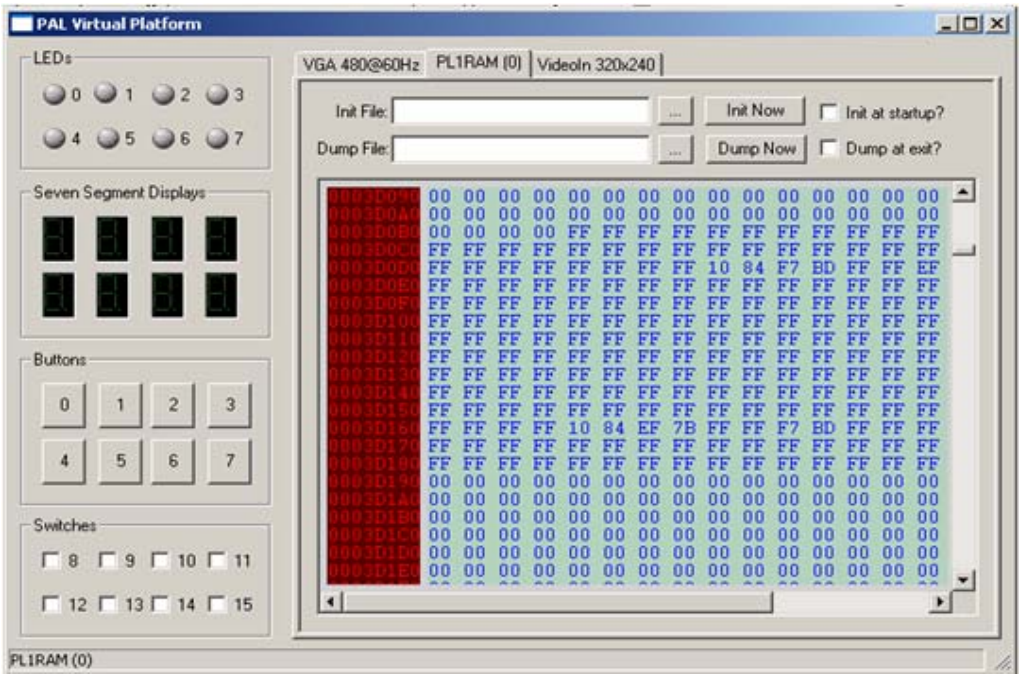

Figure 4.4 L'état de mémoire après le chargement de l'image

- **Simulation de l'étape d'extraction des paramètres**

Dés que l'image est entrée la première LED sera allumée pour indiquer le début de procédure de caractérisation. De plus, dans l'écran d'affichage l'image entrée va apparaître.
La figure (4.5) présente cette étape de simulation.

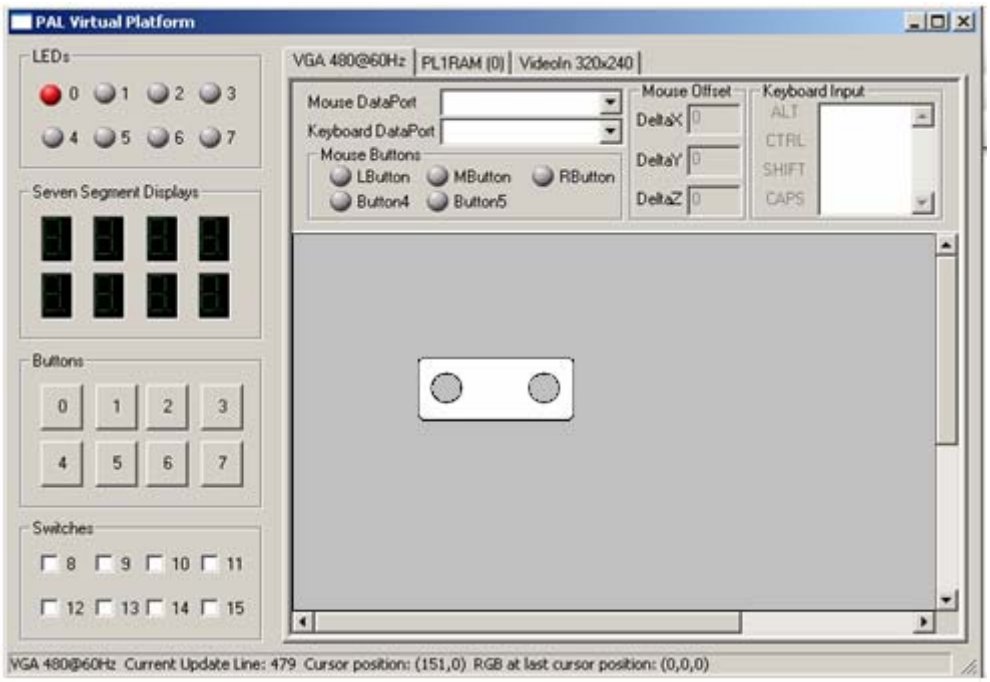

Figure 4.5 L'état du simulateur dans l'étape de caractérisation

- **Simulation de l'étape de décision**

A la fin de l'étape d'extraction des paramètres et après le calcul des erreurs relatives entre les invariants de l'image de référence et l'image à reconnaître. La décision sur la pièce sera affichée sur la deuxième LED, et les coordonnées de la pièce sur l'image sont affichées sur les deux afficheurs 7 segments.
La figure suivante donne l'état du simulateur dans le cas d'une pièce conforme à la pièce de référence.

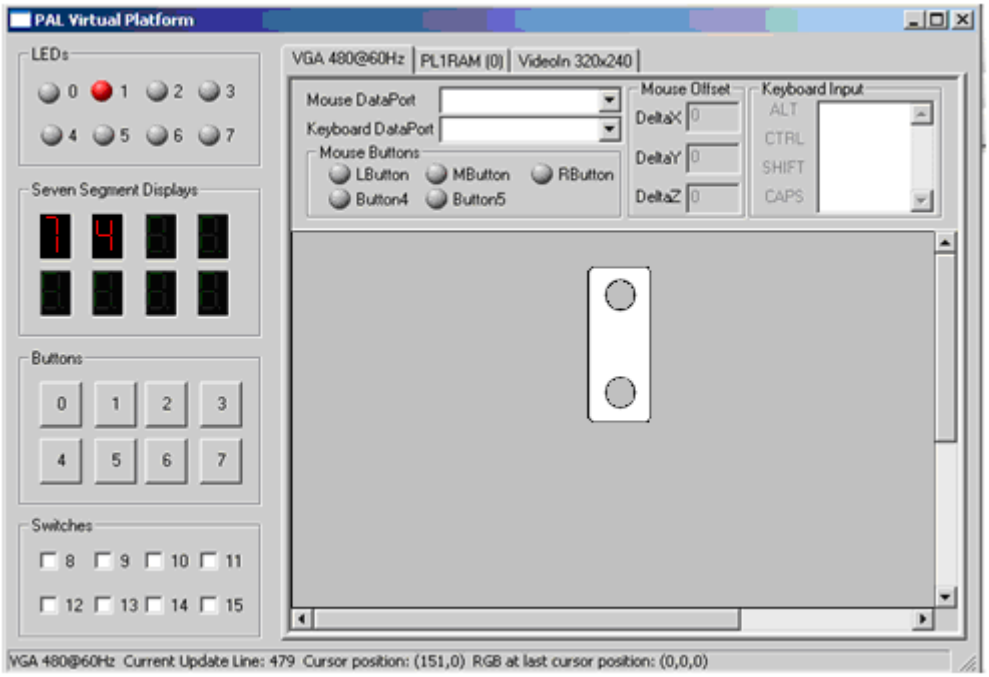

Figure 4.6 Le simulateur dans l'étape de décision

4. Le taux d'utilisation de la carte

La table suivante donne la portion d'utilisation des différentes ressources de la carte pour l'implémentation de notre architecture.

Ressources utilises	Ressources requis pour notre architecture		Ressources disponibles
Bascules (Flip-Flop)	2109	7%	28672
LUT a 4 entrées	9821	75%	14336
IOB	138	28%	484
Multiplieurs 18X18	8	8%	96
Horloges	2	12%	16

Tableau 4.1 Le taux d'utilisation des différentes ressources de la carte

Ces informations sont fournies à la fin de l'étape de synthèse. Elles représentent les composantes de l'architecture hardware réelle à implémenter sur le circuit FPGA.

- Cette architecture est dépendante de la carte cible.
- Cette architecture est optimisée.

5 Placement et routage des programmes sur la carte FPGA

Dans cette étape, l'outil de placement et routage trace les routes sur le circuit FPGA afin qu'il puisse réaliser le fonctionnement attendu. L'outil « FPGA Editor » nous permet de visualiser le routage réalisé par notre architecture sur la carte.

La figure 4.7 montre le routage sur FPGA du notre programme.

On voit que notre application utilise la majorité des ressources disponibles sur la carte.

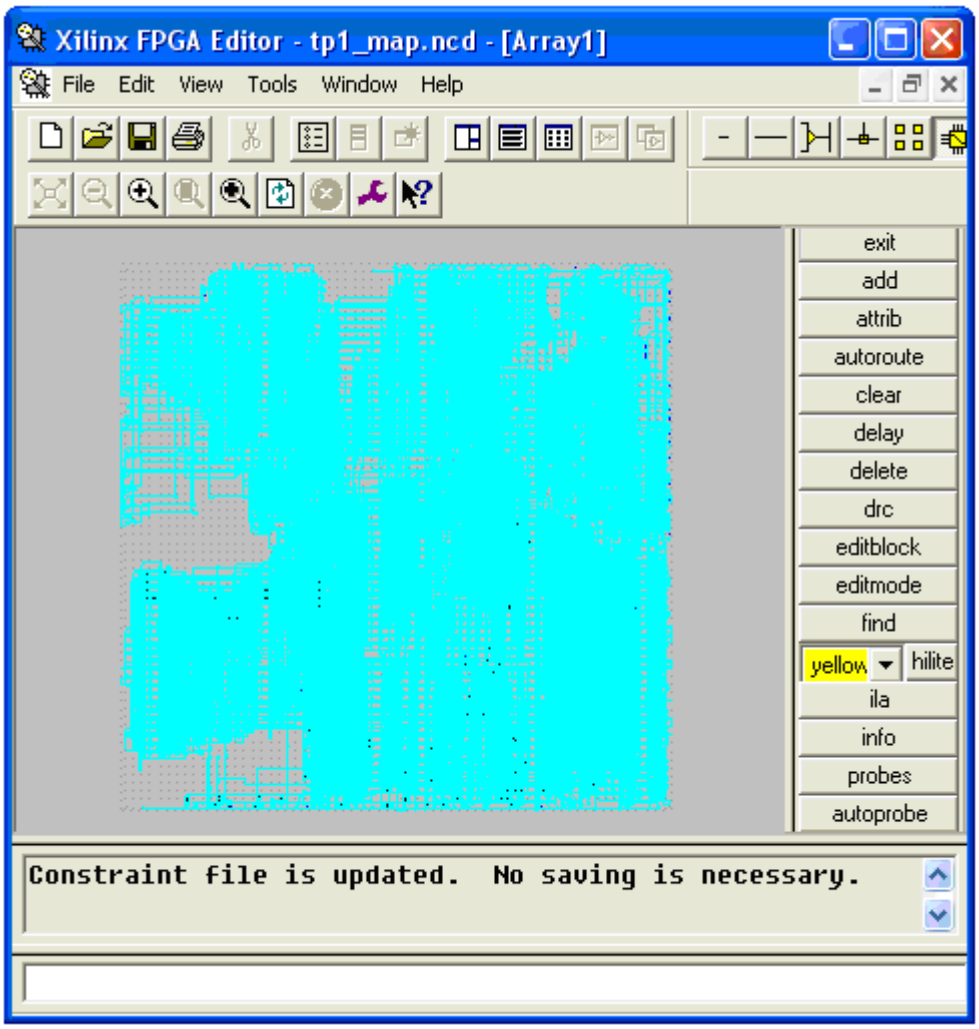

Figure 4.7 Routage du circuit FPGA pour notre architecture

6 Résultats d'implémentation sur la carte RC203E

L'implémentation sur la carte consiste à suivre les étapes d'implémentation décrites auparavant, les principales étapes sont les suivantes :

- **L'étape d'apprentissage :**

Pour l'apprentissage en utilise le PC comme une source d'images, la figure suivante présente une image sur le moniteur de PC :

Figure 4.8 aperçu de l'image dans le moniteur de PC

Dans cette étape l'état de la carte est comme suit :

Figure 4.9 l'état de la carte dans l'étape d'apprentissage

Les deux afficheurs sont à zéro et la première LED est allumée pour indiquer que la carte est en cours de chargement de l'image.

- **L'étape d'extraction des paramètres :**

Apres le chargement de l'image, l'image entrée sera affichée sur l'écran de la carte. La figure suivante montre l'image affichée sur la carte.

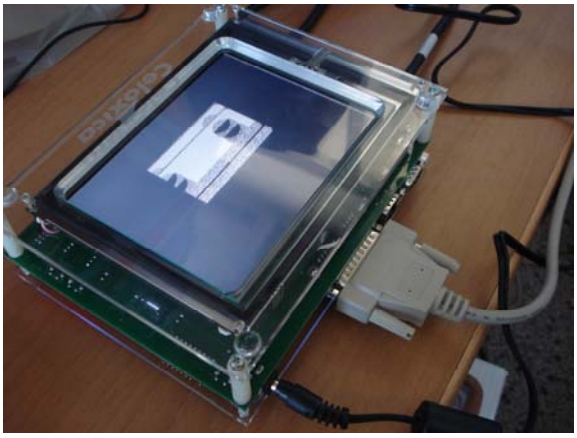

Figure 4.10 L'image sur l'écran de la carte RC203E

On remarque que l'image apparaît avec des décalages des lignes. Ce décalage est lié essentiellement à la différence de fréquence de balayage entre la carte et le moniteur du PC.

A la fin de l'étape de caractérisation, la position de centre de gravité de la pièce normalisé (10X10) est affichée sur les deux afficheurs sept segments comme indiqué par la figure suivante :

Figure 4.11 L'affichage de la position sur la carte

- **l'étape de décision :**

Pour la décision on va utiliser la deuxième LED pour indiquer si la pièce est conforme (LED allumée) ou non (LED non allumée).la figure suivante présente cette étape dans le cas ou la pièce est conforme.

Figure 4.12 L'état de la carte dans le cas où la pièce est valide

Conclusion générale

Conclusion générale

Dans ce travail, on a tenté d'explorer et d'exploiter au maximum les ressources offertes par la carte FPGA RC203E de Celoxica en implémentant un algorithme de vision artificielle basé sur les invariants de Hu, en vu de la commande d'un bras manipulateur. Cet algorithme e été implémenté ou préalable en Borland C++Builder6. Cette implémentation est simple de point de vue programmation et mise en œuvre, mais les performances liées au temps de calculs et traitements sur un ordinateur ne répondent pas aux contraints des applications temps réel.

Par contre, on a vu que l'implémentation hardware sur FPGA (carte RC203E de Celoxica) est difficile mais aboutit à des applications très rapides répondants aux exigences temps réel.

Durant notre étude, nous avons rencontré de nombreux problèmes liés essentiellement au langage Handel-C qui ne possède pas une bibliothèque qui compile les nombres réels donc on a été obligé de travailler avec les nombres entiers. On a trouvé aussi des difficultés avec la division et la puissance entières. De plus les capteurs d'images standards (Webcam, Camera et la carte graphique de PC n'ont pas la même fréquence de balayage d'image que la carte RC203E ce qui rend l'image affichée par la carte moins lisible à cause du décalage engendré par la différence de fréquence.

Bien que nous nous soyons particulièrement attachés à concevoir des architectures rapides pour la reconnaissance de formes dans le domaine temps réel, de nombreuses pistes subsistent pour améliorer les performances que nous avons obtenues. Parmi celles-ci et plus particulièrement concernant le portage sur FPGA, on peut citer :

- Utilisation d'un circuit FPGA plus performant.

- L'augmentation des nombres des invariants notamment lorsque les formes traitées sont plus complexes. Cette opération peut entraîner des problèmes de routage dans le circuit FPGA et impliquer de repenser une partie de l'architecture.

- L'implantation d'un dispositif de filtrage de bruits des images, lequel introduira inévitablement un retard supplémentaire dans l'apparition de l'image mais aboutir a plus de précision dans le calcul des invariants de Hu donc dans la vision et dans la commande du bras manipulateur.

Enfin, notre réalisation peut contribuer à concevoir un système embarqué destiné essentiellement à la reconnaissance des formes dont les applications sont très nombreuses.

Bibliographie

[1] T. Mouadh A. Sofiane, Calcul du mouvement dans une séquence vidéo par une méthode hybride, Mémoire de Fin d'Etude (P.F.E), Alger, Institut National de formation en Informatique (I.N.I),2005.

[2] C.Saint-Jean, Classification paramétrique robuste partiellement supervisée en reconnaissance des formes, thèse doctorat, France, Université de La Rochelle, 2001.

[3] A.Belaid, Y.Belaid. Reconnaissance des formes méthodes et applications. InterEditions. Paris.1992.

[4] B.Mounir, La détection de contours par la méthode de filtrage paramétrique, Projet de Fin d'Etude (P.F.E), ALGER, Institut National de formation en Informatique (I.N.I) ,2002.

[5] C.Lemaître, Utilisation de masques binaires dans une rétine de reconnaissance de Formes, Rapport de stage DEA Instrumentation et Informatique de l'image, Université de Bourgogne, Laboratoire LE2I, 2005.

[6] G Granlund J.P. Haton R Ingold M. Kocher M. Kunt, G Coray. Reconnaissances Des formes et analyse de scenes. Presses polytechniques et universitaires romandes, 2000.

[7] Alirea Khotanzad Yaw Hua Hong. Invariant image recognition by zernike moments. IEEE Transactions on pattern analysis and machine intelligence, 12(5):489–497, May 1990.

[8] Chee-Way CHONG P. RAVEENDRAN R. MUKUNDAN. Translation invariants of zernike moments. Pattern Recognition, 36:1765–1773, 2003.

[9] J.Patry, Détection de contour utilisant la logique floue, IEEE Signal Processing Letters, vol. 3, no. 6, june 1996, pp. 168-170.

[10] M.Walid, M.I.Moctar. Implémentation d'un modulateur OFDM sur un circuit FPGA. Mémoire de Fin d'Etudes, Electronique. Alger : Ecole Nationale Polytechnique (ENP), *2007.*

[11] B.Farida, D.Sabrina. Système temps réel embarque pour commander un robot mobile. Mémoire de Fin d'Etudes, Automatique. Alger : Ecole Nationale Polytechnique (ENP), *2005.*

[12] N.JULIEN. Circuits logiques programmables. *Université de Bretagne Sud. 1999.*

[13] D.Houssey. Implémentation d'une commande MPPT floue sur FPGA. Mémoire de Fin d'Etudes, Electronique. Alger : Ecole Nationale Polytechnique (ENP), *2006.*

[14] N. Abdelelah. architectures pour la stereo-vision passive dense temps réel : application a la stereo-endoscopie. Thèse Doctorat, Microsystèmes et Intégration de Systèmes. Toulouse : Université PAUL SABATIER, 2006.

[15] B.Amine, M.Choukri Adel. Implémentation sur FPGA des méthodes MPPT :"P&O" et "floue optimisée par les Algorithmes Génétiques". Mémoire de Fin d'Etudes, Electronique. Alger : Ecole Nationale Polytechnique (ENP), *2006.*

[16] G.Amar. *Implémentation d'une technique MLI en temps réel sur un circuit FPGA*. Mémoire de Fin d'Etudes, Electronique. Alger : Ecole Nationale Polytechnique (ENP), *2007.*

[17] Disponible sur <http://www.celoxica.com>. (Consulté le 20.02.2008).

[18] C. Julien ,G. Thibault ,O. Oudry. Etude d'un algorithme de détection d'objets méthodologie de parallélisation pour FPGA. Projet de deuxième année Diplôme d'Ingénieurs. Ecole Nationale Supérieure de Physique de Straspourg . France. 2001.

[19] R Mukundan S H Ong P A Lee .Image analysis by tchebichef moments. IEEE Transaction on Image Processing, 10:1357–1364, 2001.

[20] Disponible sur < *(http://www.borland.com)*>. (Consulté le 12.05.2008).