UNIVERSITE DE BOURGOGNE

U.F.R. SCIENCES ET TECHNIQUES
ECOLE DOCTORALE E2S

LE2I – UMR CNRS 5158

Thèse
présentée par

Debyo SAPTONO

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE de BOURGOGNE
(Spécialité : Instrumentation et Informatique de l'Image)

Conception d'un outil de prototypage rapide sur le FPGA pour des applications de traitement d'images

Soutenance le 04 novembre 2011 devant le jury suivant :

Président : Monsieur Frédéric MORAIN-NICOLIER, Professeur à l'Université de Reims

Rapporteurs : Madame Ruan SU, Professeur à l'Université de Rouen
Monsieur Sarifuddin MADENDA, Professeur à l'Université de Gunadarma

Examinateurs : Madame Fan YANG, Professeur à l'Université de Bourgogne
Monsieur Vincent BROST, MCF à l'Université de Bourgogne

Remerciements

Les travaux présentés dans cette thèse ont été réalisés au sein du laboratoire Electronique et Informatique de l'Image (LE2I), UMR 5158 du CNRS, de l'Université de Bourgogne, dans le département Electronique.

Je voudrais, avant de commencer, remercier :

Monsieur Vincent BROST, Maître de Conférence à l'Université de Bourgogne pour son aide précieuse, ses conseils quotidiens et son soutien sans faille dans les moments difficiles. Sans lui, ce travail n'aurait tout simplement jamais été possible ;

Madame Fan YANG, Professeur à l'Université de Bourgogne, ma directrice de thèse, pour ses qualités humaines et qui m'a toujours soutenu et conseillé pendant ces années ;

Monsieur Frédéric MORAIN-NICOLIER, Professeur à l'Université de Reims qui m'a fait l'honneur de présider le jury de cette thèse ;

Madame Ruan SU, Professeur à l'Université de Rouan, qui a marqué son intérêt pour mon travail en acceptant d'être membre du jury ;

Monsieur Sarifuddin MADENDA, Professeur à l'Université de Gunadarma, qui a accepté d'évaluer le travail de cette thèse ;

Mes collègues et amis du laboratoire LE2I qui ont contribué au bon déroulement de cette thèse.

Table des matières

Introduction ……. 1

Chapitre I Traitement d'images en temps réel & Algorithme Architecture Adéquation 5
I.1 Chaîne de traitement d'images ……. 6
I.1.1 – Traitement d'images avec trois niveaux d'abstraction …… 6
I.1.2 – Traitement d'images en temps réel ……. 8
I.2 Architectures dédiées aux implantations matérielles et/ou logicielles des algorithmes du traitement du signal et de l'image ……. 9
I.2.1 – Rétine artificielles et Caméras intelligentes …… 9
I.2.2 – Processeurs généralistes (GPP : General Purpose Processor) 11
I.2.3 – Architectures parallélisés (DSP : Digital Signal Processor) 13
I.2.4 – Circuits programmables (FPGA : Field Programmable Gate Array) …….. 14
I.2.5 – Circuits intégrés spécifiques (ASIC : Application Specific Integrated Circuit) …….. 16
I.2.6 – Le stockage et le transfert des images ……. 17
I.3. Implantations matérielles et/ou logicielles des algorithmes de traitement du signal et de l'image …… 18
I.3.1 – Evaluation des systèmes de base ……. 18
I.3.2 – Description des techniques avancées ……... 21
I.4. Prototypage rapide des systèmes embarqués …….. 26
I.4.1 – Conception des Systèmes Embarqués ……. 26
I.4.2 – Outils Hardware/Software codesign …… 27

Chapitre II Prototypage rapide pour l'implantation des algorithmes de traitement du signal et de l'image sur FPGA …… 31
II.1. Cadre de notre travail …… 31
II.2. Prototypage rapide pour l'implantation des algorithmes de traitement du signal et de l'image sur FPGA ……. 32
II.2.1 – Flot de conception classique sur FPGA …… 32
II.2.2 – Quelques langages de programmation du FPGA …… 33
II.2.3 – Quelques outils récents de programmation des FPGA ……. 35
II.2.4 – Discussions sur les outils de la programmation du FPGA …. 38
II.3. Technologie avancée de compilation : environnement Trimaran …… 40
II.3.1 – Organisation de l'environnement Trimaran …… 40
II.3.2 – Présentation du compilateur OpenIMPACT …….. 42

Chapitre III Présentation du flot de conception proposé …… 45
III.1. Présentation générale du flot de conception proposé ……. 46
III.2. Développement de l'application dans un langage informatique et compilation ……. 47
III.3. Extraction automatique du parallélisme sous contrainte de dépendance de données …….. 48
III.3.1– Contenu du fichier code machine …… 48
III.3.2– Exemple d'extraction du parallélisme ……. 49
III.3.3– Outil d'extraction du parallélisme …… 50
III.4 Analyse automatique des caractéristiques matérielles du processeur … 52
III.5. Génération automatique du modèle VHDL du DSP C6201 …….. 54
III.5.1– Outil de génération …….. 54
III.5.2– Construction de la mémoire de programme et gestion du compteur de programme …….. 55
III.5.3– Génération des opérateurs de calcul et des chemins de données associés …….. 56
III.6. Un exemple avec le filtre de Sobel ……... 59
III.6.1– Programme source en C et compilation …….. 60
III.6.2– Extraction du parallélisme du code intermédiaire …… 60
III.6.3– Analyse des ressources matérielles …….. 60
III.6.4– Génération des codes VHDL …… 62
III.6.5– Synthèse matérielle sous ISE ……. 64
III.7. Quelques résultats de synthèse avec des algorithmes courants de traitement d'image ……. 64
III.8. Conclusion du chapitre ……. 65

Chapitre IV Présentation d'un système biométrique : reconnaissance de paumes sans Contact ……..…… 67
IV.1. Problématique de la biométrie ……. 68
IV.1.1– Termes couramment utilisées …… 68
IV.1.2– Modalités biométriques ……. 70
IV.2. Acquisition d'images de paumes …… 72
IV.2.1– Acquisition avec contact …… 72
IV.2.2– Acquisition sans contact …… 73
IV.2.3– Protocole utilisé pour l'acquisition de la base uB …… 74
IV.2.4– Constitution de la base uB …… 75
IV.3. Extraction automatique de paume de la base uB ……... 77
IV.4. Chaîne de reconnaissance biométrique …….. 81
IV.4.1– Différentes modules de la chaîne de reconnaissance ……. 81
IV.4.2– Etat de l'art sur la paume : une biométrie récente …… 82
IV.4.3– Chaîne de traitement appliquée à la base de données uB …… 84
IV.5. Résultats expérimentaux …….. 85
IV.5.1– Performances d'identification de paumes avec la base uB …. 85
IV.5.2– Performances de vérification de paumes avec la base uB …… 88
IV.5.3– Comparaison des performances avec la base publique PolyU 90
IV.6. Bilan du chapitre ……... 91

Chapitre V Prototypage rapide d'un système biométrique sur le FPGA …… 93
V.1. Implémentations matérielles des systèmes biométriques …….. 93
V.2. Présentation de la plateforme d'implémentation …… 96
V.2.1 – Plateforme matérielle ……. 96
V.2.2 – Plateforme logicielle …….. 104
V.3. Prototypage rapide d'extraction automatique de paumes …… 107
V.3.1 – Segmentation d'images ……. 107
V.3.2 – Détection de contours par le code Fremann …… 109
V.3.3 – Extraction des points caractéristiques de la main …… 110
V.3.4 – Extraction de la ROI (paume) …… 111
V.3.5 – Implémentation de la chaîne complète d'extraction de la paume 111
V.4. Prototypage rapide de reconnaissance de paumes …… 113
V.4.1 – Rapide rappel de la chaîne de reconnaissance de la paume … 113
V.4.2 – Prototypage rapide de reconnaissance de la paume …… 114
V.4.3 – Implémentation optimale de reconnaissance de la paume …. 117
V.5. Bilan du chapitre …….. 120

Chapitre VI Conclusion et Perspectives …… 121
VI.1. Conclusions ……. 121
VI.2. Perspectives ……. 123

Bibliographie …… 125

Liste des publications ……..…… 137

Introduction

Dans notre vie quotidienne, l'image joue un rôle de plus en plus important pour nous informer ou nous divertir. En parallèle, le traitement de l'information s'est lui aussi développé grâce à l'évolution de la microélectronique en proposant des systèmes de plus en plus performants pour exécuter des algorithmes très complexes. La rencontre de ces deux univers a donné naissance au traitement d'images.

Les progrès dans la capacité d'intégration des circuits électroniques ont ouvert de nouvelles perspectives pour le traitement d'images en temps réel sur des systèmes embarqués. D'un côté, des processeurs spécifiques peuvent couramment effectuer des milliards d'opérations par seconde et d'un autre côté, des composants reprogrammables comporteront dans un avenir proche plusieurs milliards de portes logiques. Ces circuits permettent de réaliser des applications avec des performances en terme de vitesse de traitement sans cesse croissantes.

Les années passées ont vu l'explosion du marché des systèmes embarqués dans de nombreux domaines industriels et grand public comme par exemple les télécommunications, les satellites, et l'imagerie médicale. Ces besoins de plus en plus importants génèrent une compétition industrielle féroce où les facteurs comme le coût, les performances et surtout le «Time To Market» deviennent prépondérants pour le succès d'un produit.

Dans ce contexte, le FPGA (Field Programmable Gate Array) avec ses grandes capacités d'intégration et de reconfiguration en font un composant clé pour développer rapidement des prototypes. Dans l'objectif d'encourager la large diffusion de ce type de circuits, il est nécessaire d'améliorer les environnements de développement pour les rendre plus accessibles à des non experts en électronique.

L'étude menée dans cette thèse consiste à proposer et valider un flot de conception original pour le prototypage rapide des applications de traitement d'images en temps réels sur FPGA. Il s'agit de

programmer un algorithme en C comme si le code allait être exécuté sur un processeur classique. Les outils que nous avons développés vont traduire automatiquement ce code en langage matériel (VHDL) en utilisant une technique avancée de compilation. Ceci revient à embarquer des processeurs dans le FPGA d'une manière optimale. L'objectif est de garder la souplesse et la flexibilité du processeur d'une part et de profiter de ressources matérielles disponibles dans le FPGA afin d'augmenter les performances en temps réel d'autre part. Plusieurs aspects ont été étudiés durant ces travaux :

- Processeurs RISPs (Reconfigurable Instruction Set Processor),
- Prototypage rapide sur FPGA,
- System on Chip (SoC),
- Méthodologie Adéquation Algorithme Architecture (AAA),
- Parallélisation des traitements sur multiprocesseurs,
- Implantation matérielle des algorithmes.

Dans le chapitre I de ce manuscrit, nous allons tout d'abord présenter la chaîne de traitement d'images standard et la classification des algorithmes avec trois niveaux d'abstraction. Ensuite, nous citerons les différents composants électroniques permettant la réalisation matérielle de ces algorithmes : processeur généraliste, DSP, FPGA, SoC… Pour terminer, nous établirons un rapide état de l'art des différentes architectures (logicielles, matérielles et mixtes) et de leur modèle d'exécution.

Nous aborderons le contexte scientifique et le cadre de notre travail au début du chapitre II. Ensuite, Il est consacré à la présentation générale du prototypage des applications sur le composant FPGA, puis à une description de l'environnement Trimaran. Celle-ci doit permettre au lecteur d'appréhender plus facilement les concepts utilisés lors de la réalisation du flot de conception.

Nous commençons le chapitre III par une description générale du flot de conception proposé. Ensuite, nous présentons chaque étape de celui-ci : le développement de l'application dans un langage informatique, l'analyse des ressources matérielles et la génération du modèle du processeur. Un exemple de détection de contours avec le filtre de Sobel complètera le chapitre. Le chapitre III rend compte aussi des différentes expérimentations qui ont permis de tester et valider le flot de conception proposé pour le prototypage rapide des applications de traitement d'images.

Le chapitre IV décrit un système biométrique sans contact basé sur la reconnaissance de paume. Ce système, développé pendant la thèse de Mlle. Audrey Poinsot présente de multiples avantages d'applications grand public : robustesse, flexibilité, bas coût et non invasif. De plus, toute la

chaîne a été élaborée en suivant la méthodologie Adéquation, Algorithme Architecture. Avec les calculs réguliers, les mémoires nécessaires réduites et le parallélisme intrinsèque important, elle forme une candidate idéale pour l'implémentation matérielle sur les systèmes embarqués.

Nous présentons les expériences menées pour le prototypage rapide du système biométrique sans contact dans le chapitre V. Celles-ci sont décomposées en deux phases : extraction automatique de la ROI (Region Of Interest) et la reconnaissance de paume. Nous avons aussi effectué une implémentation « manuelle » de la dernière phase afin d'établir une étude comparative pour illustrer les avantages et les inconvénients de notre approche.

Chapitre I

Traitement d'images en temps réel & Algorithme Architecture Adéquation

L'objectif de ce premier chapitre est de situer le domaine d'application dans lequel nous évoluons. Nous nous intéressons plus particulièrement aux traitements numériques des images en temps réel. Pour cette raison, nous allons tout d'abord présenter la chaîne de traitement d'images standard et la classification des algorithmes avec trois niveaux d'abstraction. Ensuite, nous citerons les différents composants électroniques permettant la réalisation matérielle de ces algorithmes : processeur généraliste, DSP, FPGA, SoC… Pour terminer, nous établirons un rapide état de l'art des différentes architectures (logicielles, matérielles et mixtes) et de leur modèle d'exécution. Les flots de conception associés à chacune de ces catégories d'architectures ainsi que leurs caractéristiques respectives seront également exposés.

I.1. Chaîne de traitement d'images

La modélisation d'une image consiste à transformer celle-ci, représentant une scène ou un objet, en une fonction f(x, y) où l'amplitude représente l'intensité lumineuse, brillance ou couleur, de tous les points du plan. Cette conception qui s'apparente à un échantillonnage permet de considérer une image comme formée d'un nombre limité de très petites surfaces avec les coordonnées spatiales (x, y) que l'on désigne usuellement par pixel.

I.1.1 – Traitement d'images avec trois niveaux d'abstraction

Le traitement d'images intervient dans des domaines de plus en plus nombreux et complexes pour réaliser essentiellement des tâches de contrôle, d'inspection et d'acquisition de données. Nous pouvons citer la vision industrielle, l'imagerie aérienne et spatiale, l'analyse médicale, la robotique… Le dernier dans la liste est le domaine du multimédia avec ses nombreuses applications récentes. Le traitement d'image suit un processus bien défini : établir, à partir d'une image brute, une liste de caractéristiques des scènes visualisées (ou des objets présents dans cette image) pour interpréter le contenu de l'image afin de guider ou de prendre une décision. Nous présentons cette chaîne de traitement en Figure I.1 où nous distinguons trois niveaux d'abstraction [BM02] :

- **Le traitement d'image bas niveau :**

  L'entrée du traitement d'image bas niveau est réalisée par un capteur qui peut être soit une caméra CCD (Charge Coupled Device), soit une caméra CMOS ou tout autre capteur permettant l'acquisition d'une image. Une interface est alors nécessaire pour acquérir le signal vidéo, le digitaliser de manière à travailler dans le domaine numérique.

  Après cette étape, le traitement suivant consiste à améliorer l'image, c'est le prétraitement d'images, composé d'opérations ponctuelles qui modifient le contraste de l'image, et éliminent le bruit parasite. La dernière étape consiste à extraire les caractéristiques de l'image par des algorithmes de contours, de région ou de textures. C'est une succession d'algorithmes déroulés séquentiellement afin d'obtenir les caractéristiques voulues. Par exemple, lors de la détection de contours d'une image, il est nécessaire d'effectuer en premier lieu le gradient de l'image, puis appliquer des algorithmes de seuillage pour récupérer, affiner et fermer les contours. Finalement, le résultat du traitement bas niveau est une image contenant uniquement les informations essentielles. L'extraction des caractéristiques est parfois considérée comme le traitement moyen niveau.

- **Le traitement d'image moyen niveau :**

Le traitement de moyen niveau concerne l'extraction de l'information utile dans les images traitées par les processus bas niveau. Il a essentiellement pour objectif de diminuer le volume de données : seule l'information utile est conservée. Par exemple, le chaînage des points d'un contour permet de lier les pixels ayant une caractéristique commune. Nous pouvons aussi citer l'approximation polygonale d'une chaîne de points connexes permettant de créer des entités pour lesquelles la représentation de l'information est beaucoup plus dense. En résumé, le traitement de moyen niveau consiste à transformer l'image traitée en une liste de paramètres permettant ainsi de définir les objets présents et leurs relations [Ginh99].

- **Le traitement d'image haut niveau :**

Le traitement de haut niveau permet, par comparaison entre les caractéristiques qui lui sont fournies en amont et le contenu de sa base de connaissances, d'effectuer la prise de décision nécessaire.

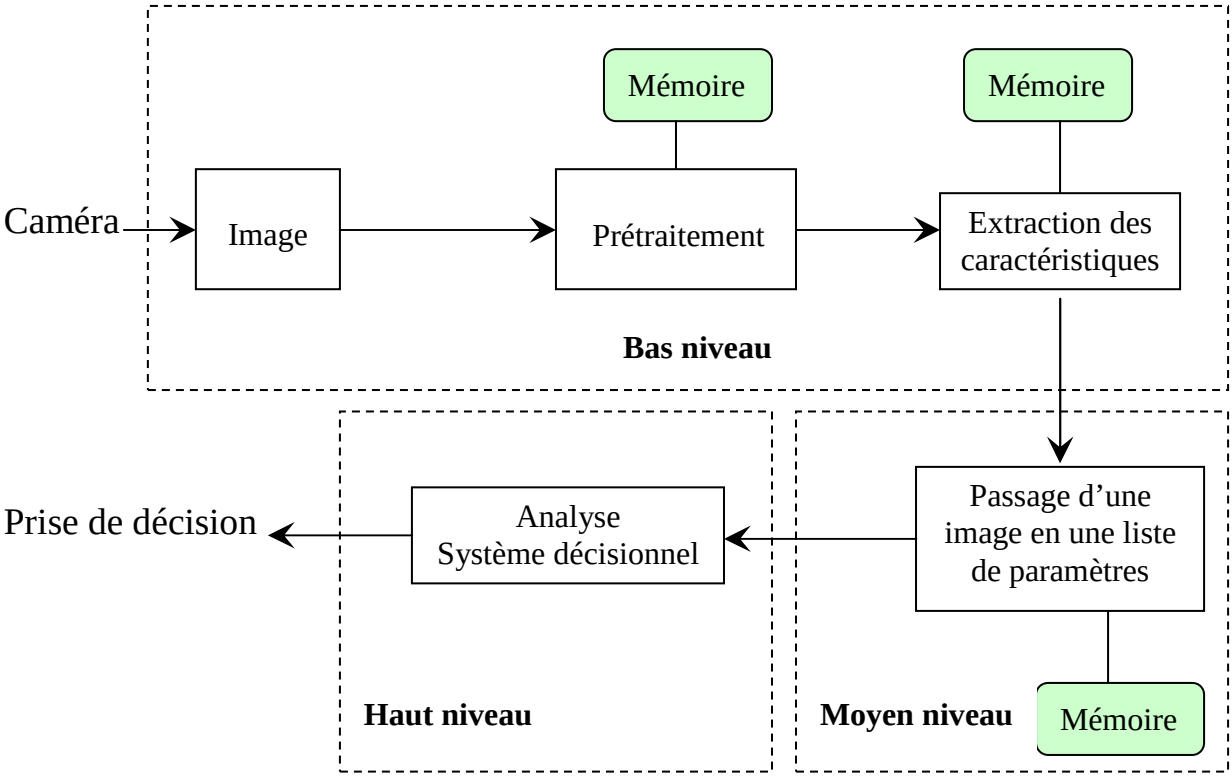


Figure I.1 : *Traitement d'images numériques en trois niveaux d'abstraction.*

Par exemple, dans le domaine de la vision industrielle, des systèmes complets ont été mis en place pour détecter et classer des défauts sur des pièces défilant sur un tapis roulant. Il est d'abord nécessaire

d'effectuer une détection de contours de la pièce considérée, d'extraire les caractéristiques principales, de les comparer avec les caractéristiques d'une pièce saine et finalement, prendre une décision pour classer cette pièce. Dans le tableau I.1, nous proposons une liste (non exhaustive) d'algorithmes que nous considérons dans la catégorie du traitement d'images avec 3 niveaux d'abstraction.

Traitement d'images	**Algorithmes**
Bas niveau : Prétraitement	Histogramme, lissage, filtres passe-bas filtres médians, filtres morphologiques, filtres de contours, transformée de Fourier, transformée en Ondelettes, DCT, transformée de Hough …
Moyen niveau : Extraction des caractéristiques et passage d'une image en une liste de paramètres	Couleur, texture, distorsion, sémantique, mouvement, profondeur, compression, étiquetage de région, fusion …
Haut niveau : Système décisionnel avec trois espaces de décision (1D : Temporel, 2D : Spatial et 3D : Spatio-temporel)	Classification, algorithme d'optimisation, qualité de représentation, similarité, homogénéité …

Tableau I.1 : *Quelques exemples d'algorithmes de traitement d'images.*

I.1.2 – Traitement d'images en temps réel

La notion de traitement en temps réel peut être définie comme suit : les informations doivent être traitées au même rythme que leur acquisition. Par exemple, si nous nous situons dans le domaine de la vidéo standard, c'est-à-dire une cadence de 25 images par seconde, le traitement d'une image se fait en 40 ms. Dans le domaine de la vision artificielle, le traitement d'images en temps réel peut être appliqué à plusieurs niveaux :

- Au niveau pixel : c'est le cas de la segmentation au sens région ou contour par exemple. Une des méthodes la plus fréquemment rencontrée dans la littérature consiste à classer chaque pixel dans une catégorie au vu de paramètres calculés à partir du voisinage (paramètre de texture, fonction d'énergie, etc.) L'opérateur de décision doit alors suivre la cadence vidéo au niveau pixel, ce qui donne des impératifs de temps de l'ordre de la centaine de nanosecondes par décision pour des signaux vidéo standard.

- Au niveau de l'image, une décision prise globalement pour toute l'image, classe celle-ci dans une catégorie donnée ; la cadence à suivre est alors de 25 décisions par seconde pour un signal vidéo standard, mais peut être beaucoup plus élevée dans le cas de la vidéo rapide (plusieurs milliers voire dizaine de milliers d'images par seconde).
- A des niveaux plus élevés encore, comme dans une séquence d'images.

Afin de satisfaire aux contraintes temporelles, des implantations matérielles d'algorithmes sont souvent nécessaires, soit sur une machine standard, soit sur une machine spécifique. Dans la section suivante, nous présentons les différents composants électroniques dédiés au traitement d'image en temps réel.

I.2. Architectures dédiées aux implantations matérielles et/ou logicielles des algorithmes du traitement du signal et de l'image

Dans cette section, nous allons décrire les différentes architectures électroniques permettant l'implantation des algorithmes de traitement du signal et de l'image en temps réel. Nous allons suivre la chaîne de traitement complète depuis l'acquisition de données jusqu'à la restitution de résultats en passant par les étapes de traitement, de stockage et de communication.

I.2.1 – Rétine artificielles et Caméras intelligentes

Grâce aux progrès de la micro-électronique, un concept nouveau a vu le jour : la « caméra intelligente ». L'idée est d'intégrer, au sein d'une caméra, un grand nombre de fonctionnalités de traitements pour pré traiter l'image en rehaussant le contraste par exemple, ou bien en adaptant une image à l'éclairage. Dans le cadre d'une utilisation pour la vidéo surveillance, la caméra peut très bien décider de stocker une séquence vidéo qui lui semble intéressante. Deux approches sont actuellement proposées pour fournir les données traitées à l'utilisateur. La première est directement liée à la possibilité d'intégrer au niveau du pixel lui même des transistors dédiés à un traitement, souvent analogique, parfois numérique avec une précision réduite. Le traitement est alors figé et limité à quelques fonctions. La seconde solution, plus classique dans les architectures dédiées au traitement d'image peut être présentée de la manière suivante : un capteur sert à l'acquisition des images qui sont transmises à la mémoire ou aux éléments de traitement. Les données peuvent être récupérées par l'intermédiaire de l'élément de communication. Un contrôleur assure le bon fonctionnement de tout ce dispositif.

Rétines artificielles : approche intégrée

Le concept de rétine fait immédiatement penser à l'œil. La biologie a effectivement inspiré les concepteurs de ces circuits. En effet, la rétine est le siège de traitements spatio-temporels et fournit au cerveau une image prétraitée. La deuxième raison de l'émergence de ces capteurs est l'utilisation de la technologie CMOS pour fabriquer des imageurs. L'utilisation de technologies identiques pour fabriquer la partie sensible et le traitement a permis de les regrouper au sein d'un même pixel. La loi de Moore montre que la densité de transistors double tous les 18 mois sur une même surface de substrat. Appliqué au domaine de rétines, l'accroissement du nombre de transistors permet d'étoffer les fonctionnalités. D'un traitement fixe, on est passé à plusieurs traitements intégrés dans un capteur puis à des capteurs programmables ou reconfigurables.

Ces rétines artificielles possèdent de nombreuses applications. Certains circuits modélisent la première couche fonctionnelle de la rétine biologique en constituant des filtres spatio-temporels permettant d'extraire les contours d'objets présents dans une image [MD95] [Sic98] [BA02]. La détection ou l'interprétation du mouvement représente aussi une grande partie des travaux publiés actuellement dans le domaine des rétines électroniques. Cette application est la mieux appropriée pour mettre en évidence les avantages et les performances que l'on peut atteindre avec des circuits de traitement dans le plan focal. L'information de mouvement est fournie en sortie du capteur et peut ainsi être utilisée pour déclencher une alarme (télésurveillance) ou engendrer une réaction d'évitement (robotique). Mais plus intéressant encore, l'interprétation du mouvement (flot optique) calculée au sein du capteur permet de piloter un véhicule autonome par exemple sans passer par un bloc d'interprétation. Le gain en compacité est évident pour de nombreuses applications. Nous pouvons citer les circuits de Wu [Moi97], de Chong [Moi97] et la matrice de Ni [NG00]. Nous pouvons aussi évoquer les travaux de Navarro [Nav02] qui a conçu un capteur capable de fournir le flot optique entre deux images successives.

Dès l'apparition des rétines en silicium, les concepteurs ont cherché à les appliquer à l'adaptation aux conditions de luminosité. Par exemple, Y. Ni propose un système d'égalisation par histogramme adaptatif pour uniformiser la vision d'une scène quelle que soit la luminosité [NDBG97]. L'égalisation par histogramme est une méthode de corrélation automatique de niveaux de gris pour rendre la distribution de ces derniers uniformes pour une matrice de photo-capteurs. Cette égalisation se fait au sein du capteur lui-même constitué d'une matrice de 64 x 64 pixels comportant 10 transistors par pixel.

De plus en plus, les chercheurs et l'industrie présentent des imageurs intégrant des processeurs complexes, soit au niveau du pixel, soit en périphérie de la matrice. Ces processeurs sont parfois analogiques ou bien numériques sur un nombre réduit de bits. L'apparition des rétines programmables permet de passer du figé vers la souplesse des circuits intégrés. Nous pouvons citer par exemple l'architecture SCAMP [DH01] et le capteur MAPP2500 de la société IVP [IVP05].

Caméras intelligentes : approche discrète

Les caméras intelligentes sont composées de différents dispositifs discrets, disponibles sur le marché. Elément important d'une caméra, l'acquisition des images est traditionnellement confiée à des imageurs CMOS ou CCD. Ils ont chacun leurs avantages et leurs inconvénients. Les CCDs bénéficient de 25 ans d'expérience en matière d'amélioration de la qualité d'image. Les capteurs CMOS sont bien plus récents et connaissent un essor important car ils ont des atouts indéniables. Le premier d'entre eux est l'utilisation d'une technologie identique à celle des éléments de traitement (processeurs, DSPs, FPGA) ce qui facilite grandement leur intégration dans des systèmes. Le second avantage est la possibilité d'adresser aléatoirement les pixels et ainsi de récupérer uniquement les parties de l'image désirées ou d'atteindre des fréquences extrêmement rapides en sortie pour de petites résolutions. Ceci est possible car chaque pixel possède son propre circuit de lecture.

Si, en matière de capteur, les choix technologiques se résument au CCD ou au CMOS, les technologies sont plus nombreuses pour les éléments de traitement. En effet, l'offre est plus que volumineuse et pas toujours facile à décrypter. Les gammes de processeurs, de DSP (Digital Signal Processor) et de FPGA (Field Programmable Gate Array), sont beaucoup trop importantes pour être décrites en détails. Ces éléments ont bénéficié, comme les capteurs, de l'évolution de la technologie CMOS et offrent globalement des performances très élevées. Cependant, sont ils tous adaptés aux traitements en temps réel sur les systèmes embarqués ? La réponse est clairement non. Les performances obtenues se font souvent au dépit de la consommation et donc certains éléments ne peuvent être utilisés dans des systèmes autonomes. Dans la suite de cette section, nous allons donner une description succincte de ces différents éléments afin de situer le cadre de notre travail.

I.2.2 – Processeurs généralistes (GPP : General Purpose Processor)

L'histoire des microprocesseurs a débuté en 1971 avec l'apparition de l'Intel 4004. Celui-ci était un microprocesseur 4 bits comportant 2300 transistors, gravés dans une technologie PMOS de 8 µm de longueur de canal avec une fréquence maximale de 750KHz. Depuis cette date, une quinzaine de générations se sont succédées. De nos jours, l'Intel Core I7 utilise une technologie de 0,032 micron avec 1 170 000 000 transistors. Les fréquences courantes atteignent 3.3 Giga Hertz. Il est à noter que depuis l'apparition de la famille Core dans les années 2005-2006, le circuit contient plusieurs cœurs de processeurs (4 dans le Core I7).

Pour implémenter une application sur un microprocesseur, on réalise d'abord une description algorithmique avec un langage de programmation quelconque qui est ensuite transcrite en séquence

d'instructions pour le microprocesseur cible. Les compilateurs sont les outils qui se chargent de cette phase de transformation. La plupart des microprocesseurs effectuent une exécution séquentielle des opérations. L'absence de parallélisme impose souvent plusieurs cycles machine pour exécuter une ligne de programme.

La figure I.2 illustre schématiquement le flot de portage et la structure d'un microprocesseur ; le code objet généré par le compilateur est chargé en RAM. Le contrôleur vient ensuite lire les instructions dans la RAM, les décode en appliquant une, ou une séquence de macro-instructions (le nombre est fonction du type de processeur : RISC, CISC) au chemin de données (« datapath »). On distingue dans chaque microprocesseur les éléments suivants :

- Le chemin de données ou « datapath » est constitué de l'unité arithmétique et logique servant aussi bien pour le traitement de l'algorithme lui-même que pour le calcul des adresses mémoires lors d'adressage complexe. Il comporte également des registres d'usage général et des registres primordiaux au fonctionnement du microprocesseur tels que le compteur de programme (PC : *Program Counter*), le pointeur de pile (SP : *Stack Pointer*), les registres de flags, et parfois le registre d'instruction (IR : *Instruction Register*).
- Le contrôleur qui contient le registre d'instructions quand celui-ci n'est pas déjà présent dans le chemin de données est la clef du modèle séquentiel de la machine Von Neumann. Il séquence et contrôle toutes les opérations effectuées à l'intérieur du processeur, et gère aussi les exceptions, telles que débordements de capacité, les requêtes d'interruptions, les transactions sur les bus, etc.

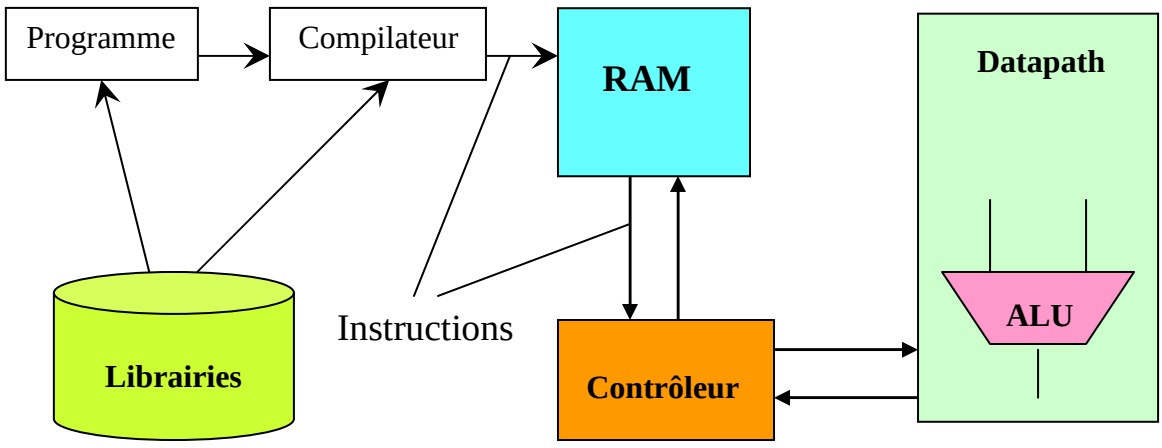


Figure I.2 : *Compilation et exécution d'un programme dans un microprocesseur.*

L'exécution de chaque instruction se décompose en 3 phases [Shi00] :

- Cycle de recherche d'instruction : Le compteur programme PC contient l'adresse de la prochaine instruction à exécuter permettant de transférer celle-ci de la RAM au registre d'instruction IR. Parallèlement, le contrôleur incrémente de 1 la valeur présente dans le compteur programme PC, qui contient alors l'adresse de l'instruction suivante.
- Cycle de décodage d'instruction : Le registre d'instruction IR contient l'instruction à exécuter, celle-ci est lue par le décodeur, qui va générer une séquence de macro-instructions nécessaire à l'exécution de cette instruction. Cette séquence commande les opérations dans le datapath ainsi que les chargements de registres, les lectures/écritures en mémoire et les opérations sur les entrées/sorties, etc.
- Cycle d'exécution : L'instruction considérée est traitée dans le datapath.

I.2.3 – Architectures parallélisés (DSP : Digital Signal Processor)

Avec l'objectif d'augmenter de manière conséquente les performances générales des microprocesseurs, des variantes autorisant un degré de parallélisme ont été développées. De nombreux DSPs (Digital Signal Processor) sont basés sur ces principes. La technologie RISC (*Reduced Instruction Set Computer*) a en effet ouvert la porte à celle du VLIW (*Very Long Instruction Word*). Cette nouvelle technologie permet aux générateurs de code exécutable de regrouper plusieurs instructions RISC en une instruction longue pour les acheminer ensemble au processeur. En ordonnançant ainsi les instructions dans le VLIW, le générateur de code peut optimiser leur exécution.

Processeurs SIMD

Les architectures SIMD (Single Instruction Multiple Data) permettent l'exécution d'une même instruction simultanément sur plusieurs données. Chaque unité de traitement reçoit ainsi une même séquence de macro-instructions, issue d'une même unité de contrôle. Chaque unité de traitement échange des données avec une banque mémoire prioritaire. Ce type de structure, apte à effectuer du parallélisme homogène, se prête bien au traitement d'applications de type multimédia où les dépendances de données sont relativement faibles.

Processeurs MIMD

Les architectures MIMD fonctionnent sur un principe similaire, mais elles sont aptes à appliquer autant d'instructions différentes qu'il y a d'unités de traitement (parallélisme hétérogène). L'inconvénient majeur, hormis les problèmes de compilation, est la présence d'autant d'unités de contrôle qu'il y a

d'unités de traitement. L'impact en surface est alors important, et se rapproche d'un système multiprocesseur. Il est à noter que peu de réalisations MIMD existent [Sas02].

Processeurs VLIW

Dans un processeur VLIW, le format d'instruction est constitué par l'agrégation de plusieurs instructions qui sont décodées puis exécutées en parallèle par les différentes unités d'exécution. Contrairement à de nombreuses architectures super-scalaires, dans lesquelles le processeur décide durant l'exécution dans quelle unité de traitement chaque instruction va être exécutée, dans une architecture VLIW, c'est le compilateur qui décide où chaque partie du mot d'instruction sera exécutée. La figure I.3 expose le format d'instruction et l'architecture de base d'un processeur VLIW. Plusieurs DSP sont basés sur ce principe, comme le TMS320 C6201 de Texas Instrument.

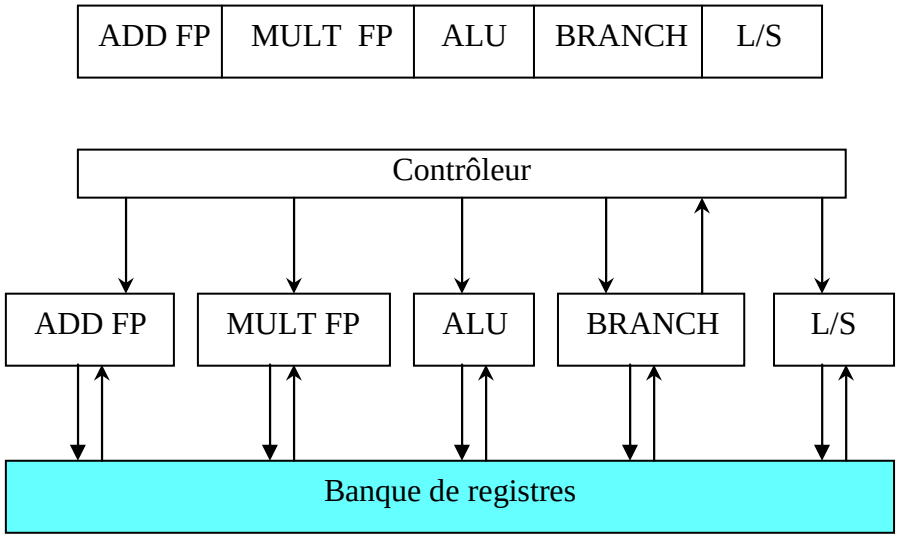


Figure I.3 : *Architecture de base de processeurs VLIW (cf. [Sas02]).*

I.2.4 – Circuits programmables (FPGA : Field Programmable Gate Array)

Nous présentons maintenant des éléments de traitement qui ne sont pas des processeurs proprement dit. Les FPGA sont des composants logiques de haute densité et reconfigurables qui permettent, après programmation, de réaliser des fonctions logiques, des calculs, et des générations de signaux. En effet, leur structure régulière (comme le montre la figure I.4) en fait des éléments très performants pour les traitements bas niveaux réguliers. Les principaux éléments composant les FPGAs sont :

- Les CLB (Configurable Logic Block) : Les CLBs constituent le cœur du FPGA. Ce sont des cellules constituées d'éléments logiques programmables où l'on trouve des bascules (registres), des LUTs (Look-Up Table), des multiplexeurs et des portes logiques disposées sous forme

matricielle, comme le montre la figure I.4. Chaque cellule est identique aux autres et peut être reliée à ses voisins par le biais de bus d'interconnexion. Ces cellules peuvent être utilisées pour créer des fonctions logiques complexes mais aussi comme éléments de stockage de variables. Au cours du temps et grâce à l'évolution de la technologie, l'architecture de ces cellules a évolué en complexité. Par ailleurs, les FPGAs intègrent de plus en plus de cellules sur une même surface. On trouve par exemple jusqu'à 400000 CLB appelés ALM (high performance Adaptation Logic Modules) dans les Stratix V de Altera [Alt05] et 1 956 000 CLBs pour les Virtex 7 de Xilinx [Xilinx].

- Les IOBs (Input Output Block) : Ces cellules d'entrées-sorties permettent d'interfacer le FPGA avec l'environnement extérieur.

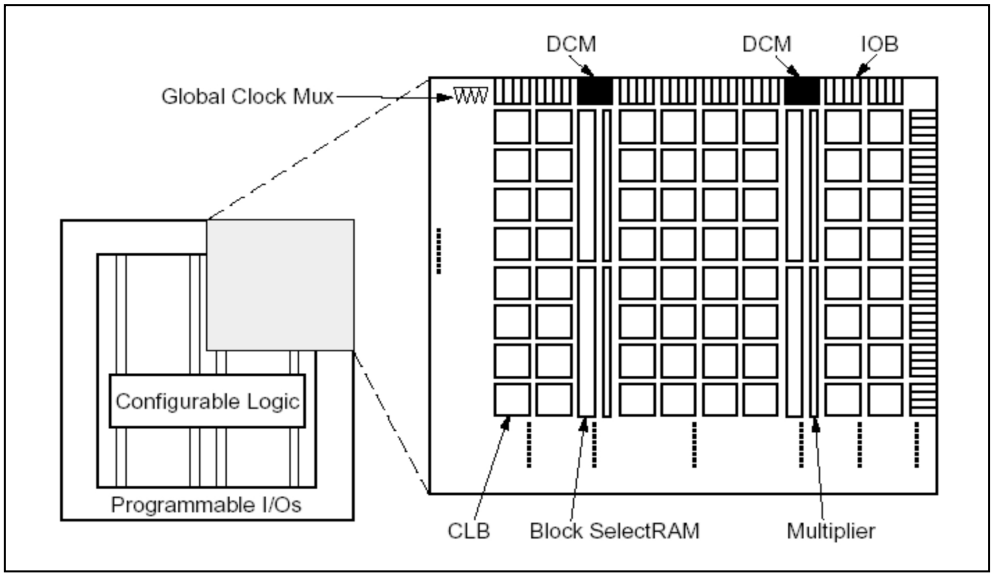


Figure I.4 : *Structure du FPGA Virtex II.*

- Les ressources d'interconnexion des cellules : Pour pouvoir réaliser des fonctions complexes à partir des fonctions de base que représentent les CLBs, il est nécessaire de disposer de ressources d'interconnexion entre ces différentes cellules. Ce sont des bus qui remplissent cette fonction. Il existe différents types de bus d'interconnexion en fonction du type de signal à propager (Les horloges ont en général des bus de transmission qui leur sont dédiés).

Depuis quelques générations de FPGAs, les fabricants ont ajouté aux ressources classiques, sur certains modèles, de nombreux éléments tels que :

- Les blocs mémoire : Les FPGAs classiques peuvent se comporter comme un espace de stockage de variables, cet espace de stockage est réparti dans tout le FPGA. Ce mode de stockage est tout de même limité en terme d'espace disponible. C'est pourquoi, pour pouvoir stocker une somme plus importante de variables sans avoir à accéder à des mémoires externes, certains fabricants de FPGAs ont introduit des blocs mémoires à l'intérieur des FPGAs, de 10 à 90 Mbits suivant les constructeurs et la taille de FPGA.

- Les multiplieurs : La logique présente dans les FPGAs permet de réaliser toutes sortes d'opérations arithmétiques (additions, multiplications, …). Ces opérations, et en particulier les multiplications sont très coûteuses en terme de ressources logiques utilisées, d'où l'intérêt de disposer dans les FPGAs de multiplieurs câblés (plus d'une centaine sont en général disponibles sur les modèles haut de gamme). De plus, le temps de calcul pour ces opérations est alors optimisé.

- Les blocs processeur : avec l'apparition des très grands FPGAs, il est maintenant possible d'intégrer des algorithmes complexes sur une seule puce. Certains FPGAs disposent aujourd'hui de cœurs de processeurs. Au début des années 2000, plusieurs fabricants proposent des cœurs de processeurs matériels au sein des FPGA (cœur Power PC pour Xilinx ou cœur ARM pour Altera). Cette disposition absente dans les FPGA depuis 2005 (Virtex 5, 6, 7 chez Xilinx par exemple) apparaît de nouveau dans les prochaines générations de FPGA proposés par Xilinx (ZYNQ 7000 disponible en 2012). Cette nouvelle famille intègre un ou plusieurs cœurs ARM dans le FPGA.

I.2.5 – Circuits intégrés spécifiques (ASIC : Application Specific Integrated Circuit)

Les circuits intégrés dédiés permettent de répondre à un cahier des charges très précis. L'architecture mise en place est alors optimale en performances pour une application donnée. Un algorithme régulier est d'autant mieux adapté à une implémentation sur un ASIC qu'il autorise la mise en œuvre efficace d'architecture pipeline et parallèle. La conception de circuits intégrés suit un certains nombres d'étapes. On distingue principalement deux familles de flot de conception numérique différenciées par le niveau de description des spécifications :

- Les flots basés sur la synthèse logique : le point d'entrée est ici une description dans un langage matériel de type VHDL ou Verilog. Celle-ci spécifie implicitement les ressources matérielles utilisées, par une description souvent structurelle. Les outils de synthèse logique génèrent alors

une description au niveau porte (netlist) dans la technologie cible, qui sera par la suite prise en charge par les outils de « back-end » gérant, entre autres, le placement et le routage.

- Les flots basés sur la synthèse d'architecture : dans ce cas les spécifications se font dans un langage soit de haut niveau (C, Pascal, langage propriétaire) faisant abstraction de toute caractéristique matérielle, soit en VHDL purement comportemental. L'outil qui synthétise l'architecture travaille alors sous contrainte de surface (nombre d'opérateurs arithmétiques disponibles), ou/et de performances (débit), et se charge des phases d'allocation et d'ordonnancement. La description obtenue est souvent composée d'opérateurs sur une technologie générique, et nécessite donc de passer par un outil de synthèse logique pour implémenter la description sur une technologie silicium.

Il est à noter que les solutions basées sur la synthèse d'architectures utilisent souvent le formalisme des processeurs [Kluw99], où l'on distingue un chemin de données et un contrôleur. Le chemin de données est figé lors des phases d'allocation/ordonnancement de l'architecture basées sur des outils de synthèse d'architecture ou bien des heuristiques telles des algorithmes génétiques ou recuits simulés. Le contrôleur quant à lui est également figé, contrairement au décodage séquentiel d'instructions dans une mémoire comme pour un processeur. Les techniques bien maîtrisées de synthèse de machine d'états finis sont souvent utilisées dans cette optique.

I.2.6 – Le stockage et le transfert des images

Nous ne présentons pas ici toutes les mémoires utilisées dans les réalisations mais plutôt les types de mémoires employées pour stocker et récupérer efficacement les données de l'image dans les systèmes embarqués et autonomes.

Le type majoritaire dans ces systèmes est la SDRAM (Synchronous Dynamic RAM). Ce type de mémoire est utilisé depuis longtemps dans les ordinateurs. Elle est peu coûteuse et relativement performante. En général, la capacité utilisée est de quelques dizaines de GOctets. Le second type de mémoire utilisé est la flash ROM. Cette mémoire non volatile, permet de stocker les programmes à exécuter et les systèmes d'exploitation temps réel utilisés. Là encore, la taille des mémoires utilisées est variable de quelques centaines de KOctets à plusieurs centaines de MOctets. Comme les processeurs, les performances des mémoires progressent très rapidement.

Un nouveau type de mémoire est disponible depuis quelques années : les MRAMs (Magnetic Random Acces Memory) [AG04]. Cette mémoire n'utilise plus une charge électrique pour stocker les

informations binaires, mais une charge magnétique, comme dans le cas des disques durs par exemple. La MRAM possède à la fois les avantages de la SDRAM (vitesse – environ 10 nanosecondes de temps d'accès a priori), de la DRAM (haute capacité, coût réduit) et de la Flash Memory (non volatile), tout en consommant très peu. Les lecteurs trouveront une description technique de ce type de mémoire dans la thèse de B. Heyrman [Hey05].

Afin de transmettre les données prétraitées on non, les systèmes de traitement d'images sont souvent équipés de ports de communications performants. Une interface couramment utilisée est l'interface réseau Ethernet associée au protocole TCP/IP qui permet de transfert à 100 Mbits/s et même 1 Gbits/s. Cette interface réseau offre de nombreux avantages. Outre sa rapidité, elle permet de contrôler la caméra à distance, d'exploiter tous les protocoles de communications existants sur ce type de réseau (FTP, Telnet, etc.). On trouve aussi des liaisons de type CameraLink [IVP05] autorisant des débits jusqu'à 3,6 Gbits/s ainsi que des systèmes proposant des liaisons de type USB3.0 autorisant des débits jusqu'à 5 Gbits/s.

I.3. Implantations matérielles et/ou logicielles des algorithmes de traitement du signal et de l'image

I.3.1 – Evaluation des systèmes de base

Pour évaluer les systèmes dédiés au traitement du signal et d'images, de multiples métriques ont été proposées. Ici, nous ne citons que les critères couramment utilisés qui permettront par la suite de qualifier chacune des architectures présentées dans la section précédente :

- Le coût est un des critères primordiaux qui détermine souvent en dernier lieu la viabilité d'une solution. Nous considérons les coûts humains, relatifs entre autres aux phases de conception et de validation, et les coûts matériels.
- La flexibilité définit la capacité d'adaptation d'une solution aussi bien aux impératifs non déterminables en phase de conception, tels que l'évolution des algorithmes et des standards, qu'en terme d'extrapolation potentielle sur les générations futures.
- Les performances intrinsèques à l'application expriment les performances potentielles du système.
- L'ergonomie qualifiera la facilité d'utilisation et de mise à jour d'un système considéré.
- La consommation quantifie l'énergie dissipée qui est un critère primordial pour un système autonome.

Avec ces critères, nous effectuons une simple évaluation des architectures de base dédiées aux implantations matérielles et/ou logicielles des algorithmes de traitement du signal et de l'image.

Implémentation sur un matériel dédié (ASIC)

Cette approche peut être retenue pour les applications fortement contraintes ou destinées au grand public. Elle conduit aux meilleures performances en termes de consommation, de vitesse et de surface de silicium. Cependant, le coût de développement d'un ASIC est très élevé car sa réalisation est spécifique à l'application et les étapes de description et d'optimisation sont longues à tous les niveaux de la conception (de l'algorithme à l'implémentation physique). De plus, un ASIC est de fait figé et n'est donc pas réutilisable pour une autre application ce qui limite fortement son intérêt dans le domaine d'applications où les normes et les standards sont évolutifs et conduisent fréquemment à des mises à jour. C'est pourquoi, les algorithmes sont de moins en moins entièrement implémentés sous formes d'ASIC et utilisent souvent une architecture programmable à laquelle est adjointe des blocs ASICs pour les traitements très coûteux. Une alternative à l'ASIC est une architecture spécialisée non plus pour une application mais pour un domaine d'application particulier (comme par exemple le codage vidéo) : l'ASIP pour « Application Specific Instruction set Processor ». Ces derniers ont une architecture mixte possédant des accélérateurs dédiés pour les traitements critiques et autorisant la programmation de nouveaux algorithmes ayant les mêmes contraintes de traitement.

Implémentation sur un processeur programmable (GPP et DSP)

La puissance de calcul et la flexibilité sont les objectifs premiers du GPP qui sont atteints grâce à une fréquence de fonctionnement élevée, des architectures matérielles et logicielles adaptées : super scalaire, fort pipeline, prédiction de branchement, cache multi-niveaux, instructions spécialisées … Cependant, le GPP n'est pas adapté aux applications embarquées du traitement du signal et de l'image pour deux principales raisons : sa dissipation d'énergie est généralement élevée et surtout il est peu adapté au temps réel. En résumé, ces processeurs généralistes sont flexibles, ergonomiques et leurs coûts sont souvent raisonnables, mais ils posent d'évidents problèmes de performances.

Le DSP allie la flexibilité d'un processeur programmable et les performances attendues pour le traitement du signal temps réel (opérateurs arithmétiques spécialisés, bande passante importante, instructions spécifiques). Les architectures parallélisées telles que les VLIWs permettent d'accroître fortement la puissance de calcul au détriment de la consommation par rapport aux solutions matérielles et la facilité de programmation par rapport à la solution GPP.

Implémentation sur un FPGA

Une approche alternative réside dans l'utilisation de circuits reconfigurables. Ceux-ci proposent un compromis intéressant entre les flexibilités du processeur et les performances de la logique câblée. Les FPGA, circuits reconfigurables à granularité fine sont les plus utilisés d'entre eux. Ces circuits reprogrammables datent d'une vingtaine d'années. A l'époque les densités d'intégration ont largement retardé leur développement car la majoration du coût en surface relatif à l'usage d'une technologie reconfigurable ainsi que l'absence totale d'outils étaitent des handicaps difficilement surmontables. Aujourd'hui, les technologies silicium ont atteint des capacités d'intégration inconcevables il y a encore quelques années, et les outils existent. Les FPGA sont devenus des éléments incontournables des flots de conception de circuits numériques complexes. Leur apport au niveau de la phase de prototypage des systèmes en cours de conception est indéniable, et les derniers modèles de FPGA [Altera] [Xilinx] affichent des capacités de plusieurs millions de portes logiques équivalentes – de quoi envisager la réalisation de systèmes avec une complexité conséquente.

La figure I.5 affiche une comparaison des performances et des flexibilités des trois architectures de base : ASIC, FPGA et GPP (DSP). Bien que la flexibilité des circuits reconfigurables conduise à une surface et à une consommation bien supérieures à celle d'un ASIC, cette solution reste un très bon compromis entre les circuits câblés et les processeurs programmables.

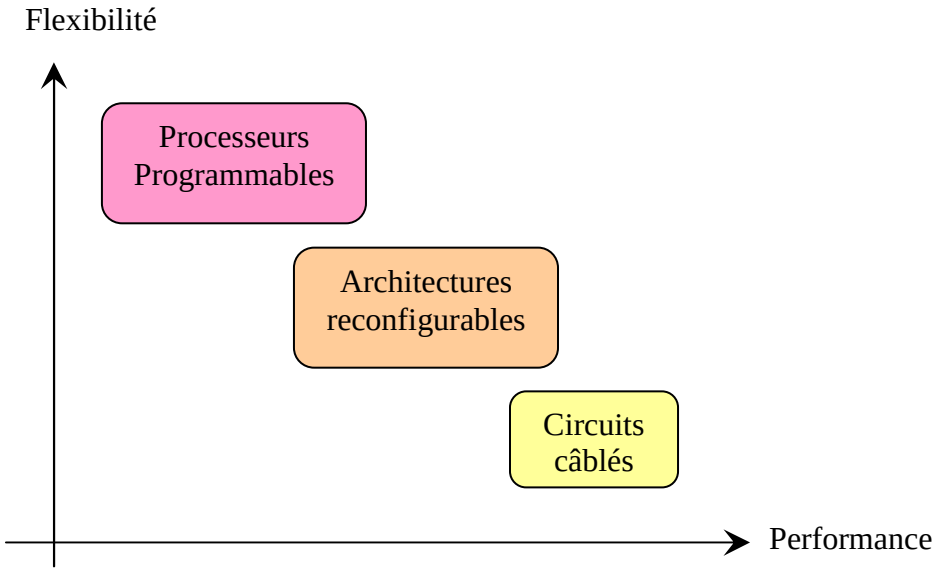


Figure I.5 : *Architectures reconfigurables : un bon compromis.*

I.3.2 – Description des techniques avancées

Dans cette section, nous présentons les nouvelles solutions se profilant comme des alternatives efficaces aux architectures de base exposées précédemment. Ces méthodes cherchent soit à combiner plusieurs architectures de base, soit à mieux exploiter leurs caractéristiques, dans le but d'implémenter plus efficacement des algorithmes de traitement du signal et de l'image.

Systèmes parallèles

Sur une seule carte électronique, plusieurs architectures de base peuvent être intégrées. Par exemple, nous trouvons couramment des systèmes composés de deux ou trois processeurs. De nombreuses machines parallèles pour le traitement d'image fonctionnent aujourd'hui avec des multi-processeurs spécialisés pour le traitement du signal. Nous pouvons aussi citer, par exemple, la carte Neuroseight [Mal02] qui contient plusieurs ASICs neuronaux pour réaliser des applications en parallèle. Pour des systèmes homogènes (duplication des architectures de base), nous pouvons les classer en machines parallèles SIMD, MIMD, etc. Pour des systèmes hétérogènes, la situation est plus compliquée, car il s'agit des systèmes mixtes.

Systèmes mixtes

La plupart des algorithmes peuvent être implantés en logiciel, mais il est assez courant d'utiliser des composants matériels spécifiques pour satisfaire les contraintes de performances et valider en temps réel une application spécifique. La méthodologie de conception est plus complexe que dans le cas d'approches classiques. Ces aspects de conception seront discutés dans la section suivante. Ici, nous donnons un exemple pour illustrer le système mixte. C'est le cas de la carte vidéo pour PC. Une carte vidéo est généralement constituée d'un co-processeur vidéo avec un jeu d'instructions dédié à l'accélération de traitement comme les déplacements de fenêtres. Les applications visées par le marché des ordinateurs grand public sont fortement orientées autour de la vidéo : la visualisation de films encodés en MPEG1, MPEG2 ou en MPEG4, ou encore la vidéoconférence. Ces applications sont excessivement exigeantes en ressources, et l'IDCT (Inverse Direct Cosinus Transform) est certainement l'algorithme commun s'avérant le plus exigeant. Dans ce contexte, la solution retenue par les constructeurs de ces cartes tel ATI consiste alors, conjointement au processeur vidéo, à intégrer un composant matériel d'IDCT accélérant l'ensemble de ces applications [Sas02].

Systèmes sur puce (SoC : System on Chip)

La constante évolution technologique des semi-conducteurs et l'apparition d'applications de plus en plus complexes et gourmandes en ressources de calcul (systèmes de vision embarqués, téléphonie portable, en particulier les mobiles de quatrième génération) ont encouragé l'émergence d'un nouveau type de circuit : le System-on-Chip (SoC). Un SoC peut se définir comme un système mixte matériel/logiciel implanté sur une puce unique. On peut ainsi retrouver dans un même circuit un microprocesseur, de la mémoire contenant son code logiciel, des blocs de propriété intellectuelle gérant une application particulière ou encore des modules d'architectures matérielles dédiées. Ces systèmes sont composés [Sas02] :

- de macro-blocs, tels que des cœurs de processeurs ou des cœurs dédiés à des applications spécifiques, mais aussi de blocs d'origine diverses, tels que des cœurs reconfigurables, des systèmes Radio-Fréquence, etc.
- d'une logique dédiée à l'interfaçage de tous les cœurs du système.

Les macros-blocs autorisent la conception par utilisation de blocs préconçus, qui sont souvent vendus en tant que cœurs **IP** (*Intellectual Property*). Un IP est un composant virtuel dont on peut distinguer deux types : le premier correspond à un cœur de processeur (RISC, DSP, …), le deuxième correspond à une fonction ou à un opérateur spécifique. Il peut apparaître sous différentes formes :

- les « Soft cores » : Ce sont des blocs décrits dans un langage de description matériel (HDL : Hardware Description Language) synthétisable. Ils sont indépendants de toute technologie et sont très souples d'utilisation.

- les « Firm cores » : Ce sont des blocs ayant subi une optimisation en surface et vitesse par des techniques de placement relatif. Décrits en HDL structurel, ils font appel à des composants élémentaires d'une librairie générique. Plus souples que les IPs matériels, ils sont portables sur plusieurs technologies.

- Les « Hard cores » sont eux optimisés et «placés-routés» pour une technologie donnée. Le bloc est donc implicitement caractérisé en termes de performances et de surface.

G. Sassatelli [Sas02] a comparé les spécificités des différentes familles de blocs (voir le tableau I.2) : chaque bloc est caractérisé par un rapport entre performances et flexibilité constant. Plus un bloc sera proche d'une technologie cible, plus il sera performant et moins il sera flexible.

	Soft Cores	Firm Cores	Hard Cores
Représentation	comportementale	structurelle	masque
Technologie	indépendant	générique	fixe
Flexibilité	très grande	grande	très faible
Portabilité	illimité	librairie	process
Performances	haut niveau	limité	optimum
Protection	très faible	faible	grande

Tableau I.2 : *Comparaisons des différents modèles de blocs IP (cf. [SasO2]).*

Les blocs IPs, basés sur le principe de réutilisation (design reuse), contribuent à augmenter la productivité et à diminuer les temps de conception (time to market). Ces facteurs sont cruciaux dans les contextes industriels à forte compétitivité. Le marché du téléphone mobile est un exemple caractéristique, où un constructeur propose une nouvelle génération de terminaux tous les 6 mois en moyenne. On trouve actuellement trois types de SoC pour le téléphone mobile cellulaire. Le premier, le plus classique, effectue tous les traitements numériques sur des cœurs de processeurs. Cette solution se révélera bientôt inadéquate à cause des contraintes de performances amenées par les nouvelles applications à caractère multimédia. Le deuxième type consiste à intégrer un cœur dédié pour accélérer le traitement des algorithmes les plus exigeants des applications visées. Le dernier type de SoC consiste à implanter un cœur reconfigurable dans lequel on vient synthétiser une architecture pour répondre ponctuellement à une application donnée.

Reconfiguration dynamique

La capacité de re-programmation, partielle ou totale, pendant le traitement est une des caractéristiques des composants FPGAs. C'est ce qu'on a coutume d'appeler reconfiguration dynamique. Le terme reconfiguration dynamique ne s'applique pas uniquement aux FPGAs mais également à toute architecture qui peut modifier sa structure de traitement des données à tout moment. Cette possibilité a très tôt intéressé les chercheurs qui ont ainsi développé plusieurs thèmes autour de la reconfiguration dynamique : aspect architectures, aspect outils… D'un autre coté, l'aspect reconfiguration dynamique des FPGA est encore très peu utilisé au niveau industriel. Ceci est, en grande partie, dû au manque d'outils adaptés facilitant la mise en œuvre de celle-ci. Cependant, l'augmentation de la complexité et de la taille des composants ces dernières années permet d'implanter des algorithmes de plus en plus complexes. Dans ce cas, on peut envisager de changer une partie de l'implantation au cours de traitement, pour s'adapter à une nouvelle situation ou à des données de types différents et la reconfiguration dynamique partielle permet d'effectuer ces modifications sans perturber le reste du système implanté. De plus, les outils ont évolué et prennent maintenant en compte la possibilité

d'utilisation de la reconfiguration dynamique partielle. Il est donc à prévoir que, dans les prochaines années, on verra le développement d'applications utilisant la reconfiguration dynamique.

Les architectures reconfigurables peuvent êtres classées en 2 catégories en fonction de leur granularité : les architectures à grain fin ou architecture à reconfiguration au niveau porte [Ardoise] [BB05] [Abe05] [NAPA] [GARP] et les architectures à grain épais ou architectures à reconfiguration au niveau fonctionnel [Ring] [DART] [RaPID]. Les FPGAs sont des circuits reconfigurables à grain fin [Har00], c'est-à-dire que leur cellule de base est dédiée à des manipulations au niveau du *bit*. Les algorithmes de traitement du signal et de l'image font un usage intensif d'opérations arithmétiques, opérant au niveau *mot*. L'utilisation d'un réseau programmable de cellule plus complexe (FPFA : Field Programmable Function Array [Smi97] [Bou01]), comme architecture l'architecture de RCP de la société Chameleon [Chameleon] ou encore l'architecture IRAM de l'Université de Berkeley [Pat97], conduit à une architecture mieux adaptée au traitement du signal.

Les architectures reconfigurables à grain épais sont réalisées avec des opérateurs arithmétiques câblés tels que des additionneurs et des multiplieurs qui manipulent des mots. Par ailleurs, des opérateurs de mémorisation de type registres permettent de stocker des résultats intermédiaires. La reconfiguration de l'architecture consiste à déterminer la fonction réalisée par chacun des opérateurs (grain épais) et l'ordre de passage des données dans ces opérateurs. Il y a beaucoup moins d'opérateurs arithmétiques sur une architecture à grain épais que de CLBs dans un FPGA. Il en résulte une diminution considérable du nombre de configurations possibles et donc de la durée nécessaire à reconfigurer l'architecture. De ce fait, les architectures à grain épais s'adaptent mieux à la reconfiguration dynamique.

Des travaux universitaires ont été aussi menés dans cette voie. Citons par exemple les architectures DART et Systolic Ring. DART [Dav03] se compose de trois ressources principales : des clusters chargés de l'exécution des opérations (partie opérative), un contrôleur de tâche qui a en charge l'assignation des traitements aux clusters (partie contrôle) et aux mémoires. La figure I.6 illustre le concept de cette architecture. Chaque cluster est composé d'un cœur de FPGA, de DataPath Reconfigurables (DPRs) qui contiennent les unités fonctionnelles (ALU, multiplieurs) interconnectées via un réseau segmenté et d'un contrôleur gérant les ressources de calcul.

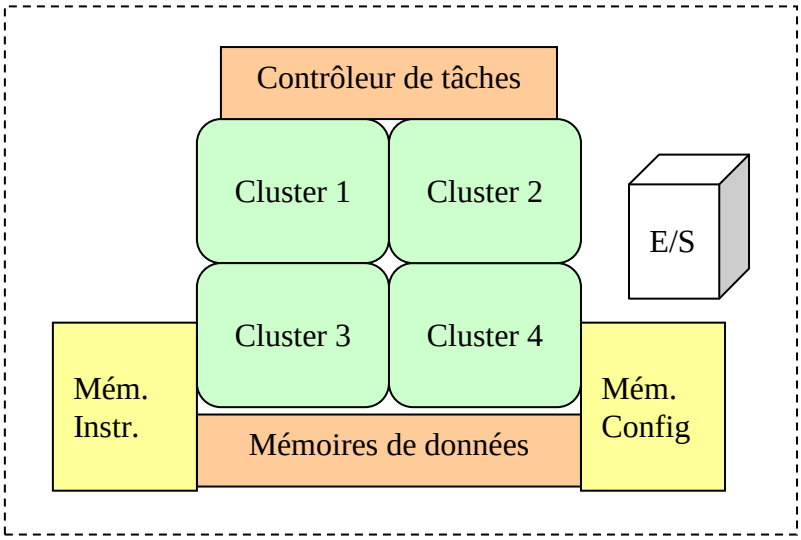


Figure I.6 : *Architecture haut niveau de DART.*

Systolic Ring [Ben04] dont le concept est présenté dans la figure I.7 contient deux couches qui communiquent chacune avec une mémoire. La couche opérative est constituée de Dnodes qui réalisent les calculs et de ressources d'interconnexion. Un Dnode est composé d'une ALU, d'un multiplieur et de registres locaux. Les ressources d'interconnexion, organisées en anneau, permettent de chaîner des Dnodes pour réaliser les traitements. Cette couche opérative va lire et écrire uniquement dans la mémoire de données. La seconde couche, appelée couche de contrôle, se charge de lire les configurations stockées dans la mémoire programme, de les décoder pour les appliquer à la couche opérative et la configurer (mise en place des chemins de données et des opérations à effectuer sur chaque Dnode).

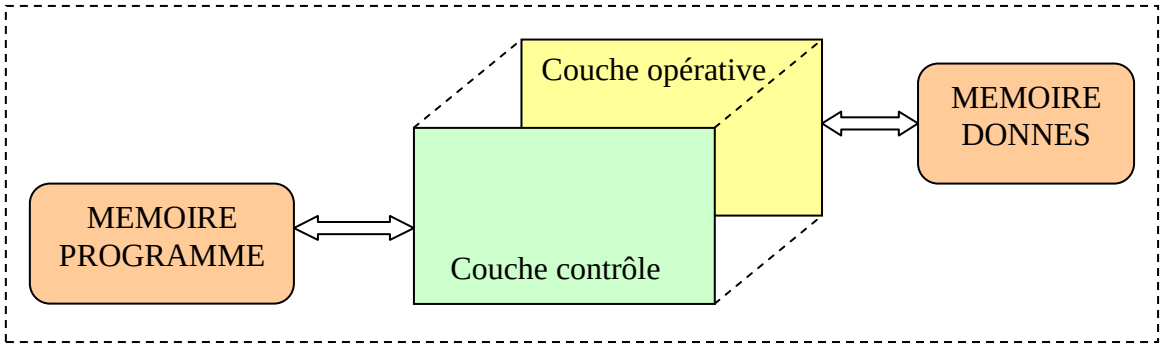


Figure I.7 : *Architecture conceptuelle du Systolic Ring.*

I.4. Prototypage rapide des systèmes embarqués

I.4.1 – Conception des Systèmes Embarqués

La diminution constante de la taille des circuits intégrés, prédite par la loi de Moore, avec les retombées favorables qui en découlent sur le prix et la performance a permis l'essor des systèmes embarqués. Ces systèmes correspondent à des équipements souvent autonomes contenant une «intelligence» qui leur permet d'être en interaction directe avec l'environnement dans lequel ils sont placés. Les systèmes embarqués sont présents dans des applications de plus en plus nombreuses, par exemple les cartes à puce, les systèmes mobiles communicants (tels que les téléphones mobiles, les mobiles dans les réseaux ad hoc), l'automobile, l'avionique, les capteurs intelligents, la santé ou l'électronique grand public.

Comme l'illustrent ces applications, les comportements critiques doivent être garantis dans ces systèmes car les défaillances peuvent avoir des conséquences graves. Dans d'autres cas, il s'agit de rechercher des compromis entre plusieurs critères comme la qualité de service, les coûts, les délais de conception ou la consommation d'énergie des équipements mobiles. Les systèmes embarqués intègrent des applications de plus en plus complexes tout en exigeant des performances plus importantes. Pour répondre à cette demande, il est nécessaire de mettre au point des architectures plus évoluées en termes de puissance de calcul. Cette augmentation des performances nécessite un parallélisme accru qui tend à augmenter fortement certains critères comme la surface de silicium et la consommation d'énergie. La recherche de compromis architecturaux est donc nécessaire lors de la conception des systèmes embarqués.

L'augmentation incessante de la complexité des systèmes embarqués et l'apparition de systèmes complets intégrés sur une même puce (SoC) ont entraîné ces dernières années un renouvellement de la méthodologie de conception. Historiquement, les étapes de conception, d'implantation et de validation matérielles et logicielles se déroulaient séquentiellement. Aujourd'hui, la spécification du système, la conception matérielle, le développement logiciel et la synthèse des interfaces, sont réalisés simultanément (ou du moins sont recouvrants). De plus, la validation et le test interviennent maintenant plus tôt dans le flot de conception. La problématique du concepteur se situe désormais au niveau du système architectural : quels circuits (DSP, RISC, GPP, ASIC, FPGA, SoC) ? quel bus ? quel protocole de communication ? quelle quantité de mémoire « on Chip » et « off-Chip » ? Quelle quantité de matériel faut-il ajouter ? Existe-t-il des IPs ré-utlisables [Roux03] ?

La conception de systèmes mixtes contenant des composants matériels et logiciels n'est pas un problème nouveau. Les techniques classiques de conception imposent de séparer le matériel du logiciel très tôt dans le cycle de conception. Les chemins suivis par les concepteurs des différentes parties

restent indépendants en n'autorisant l'interaction que lors de la phase d'intégration. Cette approche présente peu de flexibilité dans l'évaluation des différentes options de conception et de répartition du matériel et du software. Avec des méthodologies de conception différentes pour le matériel et le software, il est également difficile d'optimiser le produit à concevoir en tant que système global. Une telle approche est inadéquate spécialement lors du processus de conception de systèmes digitaux requérant des performances strictes et un temps de cycle de conception réduit pour des raisons économiques évidentes. Les développements dans plusieurs domaines de la technologie ont amené une ré-examination des frontières traditionnelles entre le matériel et le software et ont jeté un doute sur l'utilité de maintenir une frontière rigide entre les deux. La tendance actuelle, principalement lorsqu'il s'agit de systèmes temps réel complexes, est de tenir compte dès la phase de spécification des interactions entre les deux parties. Ainsi, la séparation est effectuée le plus tard possible dans le cycle de conception qui tend à unifier la spécification pour qu'elle englobe le matériel et le software. Cette approche est nommée « *hardware/software codesign* », ou « *codesign* ».

I.4.2 – Outils Hardware/Software codesign

Le tableau I.3 présente quelques outils du commerce permettant un partitionnement et/ou une co-simulation hardware/software à partir d'une spécification au niveau système. Ces outils réalisent généralement le co-design des systèmes embarqués en trois étapes.

- Dans un premier temps, la conception commence à partir d'une spécification unique décrivant son architecture et/ou son comportement. L'objectif de cette spécification réalisée à l'aide d'un langage de haut niveau est d'obtenir un modèle simulable et représentatif de l'implémentation finale. Dans cette description, on doit pouvoir spécifier des tâches qui communiquent entre elles via des signaux.
- Dans un deuxième temps, on effectue le partitionnement ainsi que le choix des protocoles de communications. Lors de cette phase, le choix est fait de l'implémentation en matériel de tel traitement et en logiciel de tel autre ou de l'utilisation d'un IP réalisant le traitement souhaité.
- Les partitions obtenues doivent ensuite être vérifiées et validées avant de passer à la phase de synthèse et de mise en œuvre. Une analyse des performances est également effectuée à cette étape. Elle consiste en une mesure des taux d'occupation des mémoires embarquées, des charges des cœurs de processeur (s'il y a lieu), de l'activité des transitions sur les bus et enfin de la dissipation d'énergie du matériel dédié …

Un feed-back permet de revenir à la phase de partitionnement en affinant différemment les poids associés aux contraintes si les partitions obtenues ne s'avèrent pas satisfaisantes. La figure I.8 illustre le flot de conception co-design générique utilisé pour la conception des SoCs.

Compagnie / outils	Fonctionnalité
Cadence Design Systems / Cierto Virtual Component Co design VCC	Diagramme de spécification comportemental, Diagramme de l'architecture, Processus d'allocation, Modèle de performances pour la simulation, Simulateur, Environnement d'analyse, Passerelle vers des outils d'implémentation.
Co-Design Automation / Superlog	*Système :* interfaçage, protocole, machine d'état, file d'attente, … *Software :* processus dynamique, tableaux, structures, pointeurs (C/C++, Java), *Hardware :* événements, concurrence, logique multivaluée, tâches/fonctions, logique, séquentiel/combinatoire (Vérilog/VHDL), *Validation :* Générateur des tests aléatoires.
Mentor Graphics / Seamless CVE Monet	Co-vérification HW/SW, Prototypage virtuel, Optimisations dynamiques, Supporte les architectures multi-processeurs, API simulateur/debogger ouvert, Kit d'intégration de processeurs (nombreux CPUs, DSPs supportés), Vérification formelle des SoCs, Outil de génération de testbench, Synthèse comportementale.
Co Ware / Co Ware N2C Design system	Spécifications exécutables : C/C++, SystemC, Modélisations fonctionnelles et modèles de performances, Génération automatique d'interfaces, Conception hardware par raffinement du modèle C C2HDL, intégration d'IPs, Co-simulation HW/SW.
Xlinx / Forge	Générateurs C/C++, Java vers Verilog, Personnalisation des contraintes, Vérification, simulation, caractérisation, Partitionnement, Allocation HW/SW, parallèle, pipeline, distribué.
Foresight System Inc. / Foresight Co-design System	Environnement de modélisation et de simulation pour la conception des SoCs, Description VHDL, Verilog, C, C++, Co-simulation avec les outils de Mentor Graphics Seamless CVE et Modelsim.

Tableau I.3 : *Quelques outils de conception haut niveau (cf. [Roux03]).*

Les avantages majeurs des approches co-design sont les suivants :

- Performances : Le niveau de performance du produit final est potentiellement élevé par la présence de deux degrés de liberté majeurs lors de la conception : il est possible d'une part de choisir, ou concevoir un cœur câblé avec une grande latitude de rapports performances/coûts ; il est également possible de choisir le processeur parmi un vaste choix de modèle disponibles, autorisant d'ajuster encore une fois la solution aux besoins.

- Flexibilité : L'intégration d'un processeur dans la solution autorise un niveau de flexibilité bien plus conséquent que les solutions matérielles pures. Il est ainsi possible d'envisager des mises à jour du système après sa livraison par la modification du programme d'exploitation (ou « firmware »).

Par contre, les coûts de mise en œuvre de telles solutions sont souvent plus importants. Le flot de conception (figure I.8) illustre la sur-complexité relative à la conception d'un système mixtes en terme de validation : la validation par simulation est encore moins envisageable que dans le cas de systèmes matériels ou logiciels, de par l'hétérogénéité des modèles comportementaux disponible et des temps processeurs mis en jeu. Le recours au prototypage est souvent impératif et pose souvent des problèmes non négligeables. Cela implique d'avoir un exemplaire du processeur définitif, de l'interfacer sur la plateforme de prototypage et de gérer la synchronisation entre logiciel et matériel par l'intermédiaire d'une logique additionnelle qu'il faut également concevoir.

D'ailleurs, de nombreux problèmes restent soulevés par les communautés scientifiques et industrielles qui s'intéressent au co-design :

- La co-spécification des systèmes mixtes: il n'existe toujours pas de formalisme universellement adopté, qui permette de décrire l'ensemble du système à l'aide d'un langage de haut niveau sans introduire de biais vers l'une ou l'autre des implémentations (hardware ou software), ce qui compromettrait l'efficacité des étapes ultérieures ;
- L'analyse des caractéristiques statiques et dynamiques des systèmes à concevoir ;
- La détermination d'un modèle permettant de modéliser l'ensemble de l'application, de servir de noyau aux outils de codesign durant toutes les étapes, de maintenir la cohérence des informations associées aux différentes entités et de permettre la co-validation du système avant et après le partitionnement.
- Le développement d'une méthode d'ordonnancement permettant d'établir des schémas d'exécution tenant compte de l'aspect séquentiel des traitements, de la concurrence, de la communication et des différents aspects de synchronisation ;

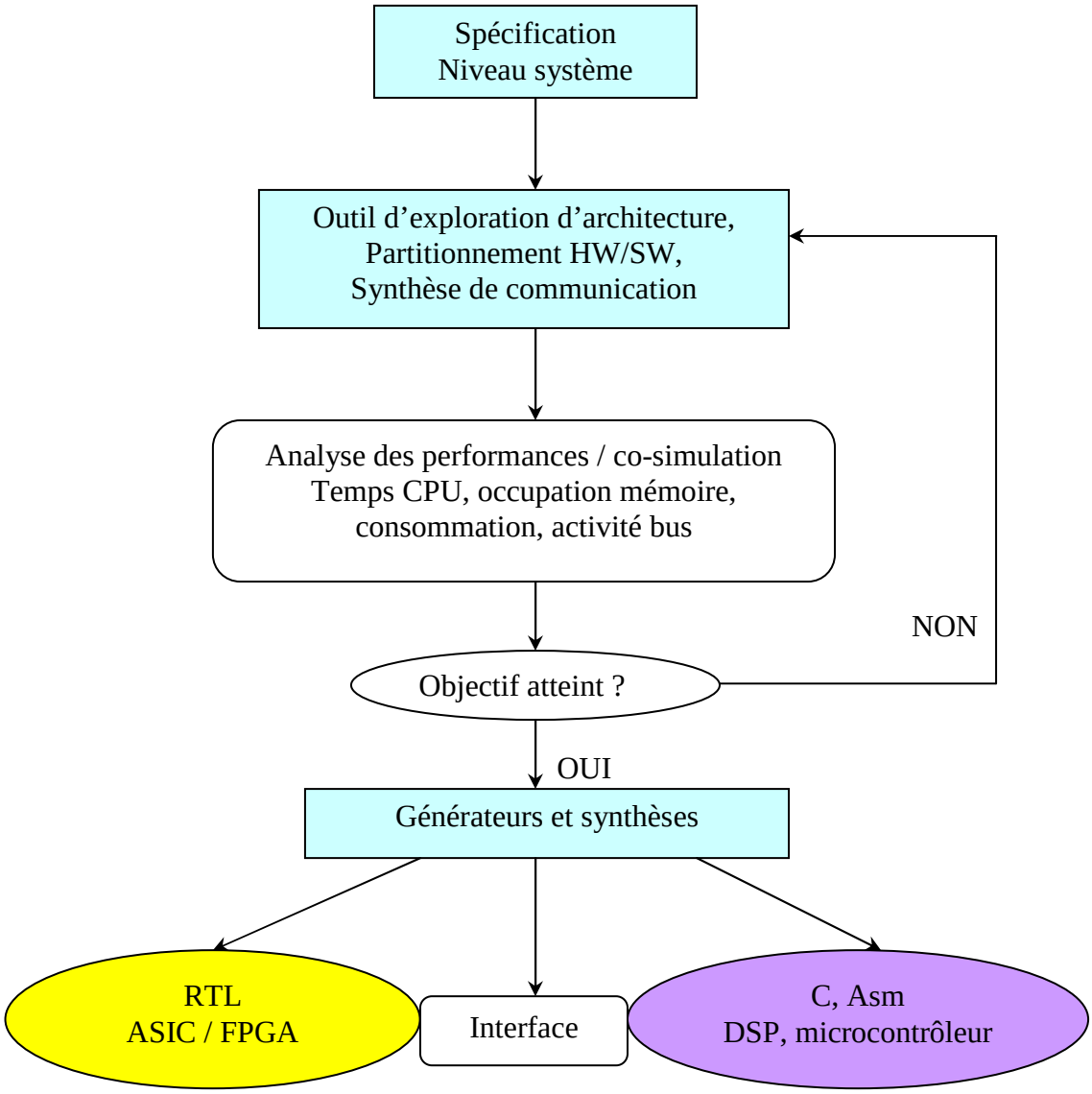


Figure I.8 : *Méthodologie de Conception de SoC.*

- Le développement d'outils de co-validation par co-simulation et/ou par preuve formelle ;
- Le développement d'outils de synthèse en fonction de l'architecture cible adoptée...

En lien avec le contexte scientifique des travaux présentés dans ce manuscrit, nous allons décrire plus en détails les méthodes et les outils existants pour l'implantation matérielle des applications sur le composant FPGA. Nous allons voir que la flexibilité et l'adaptabilité du cycle de développement jouent un rôle primordial pour réduite le Time-To-Market (TTM).

Chapitre II

Prototypage rapide pour l'implantation des algorithmes de traitement du signal et de l'image sur FPGA

Dans ce chapitre, nous précisons d'abord le cadre de notre travail. Il s'agit de développer des outils avancés pour réaliser le prototypage rapide des algorithmes de traitement du signal et de l'image sur FPGA. Ensuite, nous présentons un rapide état de l'art des méthodes et des outils existants dédiés à l'implantation matérielle des applications sur des systèmes embarqués. Nous décrivons en troisième partie une technique avancée de compilation : l environnement Trimaran, qui est utilisée dans notre flot de conception.

II.1. Cadre de notre travail

Le cadre de notre travail se limite pour l'instant au prototypage rapide pour l'implantation des algorithmes de traitement du signal et de l'image sur FPGA. Nous proposons une alternative des approches existantes. L'idée est de combiner les avantages des processeurs et des composants FPGA.

La densité croissante des FPGAs permet le prototypage rapide des circuits numériques à grande complexité. Aussi, il est possible de tester rapidement la validité de concepts architecturaux nouveaux : l'implémentation complète d'un processeur sur des circuits FPGA est aujourd'hui à notre portée. Dans ce contexte, nous proposons un flot de conception original pour le prototypage rapide des applications de traitement d'images sur FPGA. Nous programmons un algorithme en C ou C++ comme si le code était destiné à être exécuté sur un processeur classique. Des outils développés vont traduire ce code en VHDL en suivant le modèle du processeur VLIW. Au final, des modèles VHDL de processeurs VLIW vont êtres embarqués dans le FPGA. L'objectif est de garder la souplesse et la flexibilité des processurs et en même temps de profiter des ressources matérielles disponibles sur le FPGA afin d'augmenter les performances en temps réel.

Les expériences des implantations des applications sur des DSPs nous ont permis d'observer un phénomène de sous utilisation de tels processeurs, c'est-à-dire que les ressources matérielles disponibles sur un DSP ne sont pas toutes utilisées pour une application donnée. Par exemple, pour un processeur VLIW, toutes les unités fonctionnelles ne sont pas forcément nécessaires ; pour chaque unité, parfois,

seulement quelques instructions sont appelées ; de même, les registres disponibles sur chaque datapath ne sont pas tous utilisés … Basé sur ce phénomène de sous-utilisation des DSPs, nous effectuons une étape d'optimisation. Le processeur VLIW embarqué dans le FPGA ne possède que les ressources qui sont juste nécessaires pour la réalisation d'une application donnée. Grâce à cette étape d'optimisation, nous économisons les ressources matérielles du FPGA, ce qui nous permet d'implanter plusieurs modèles de processeur VLIW adaptatifs afin de réaliser une machine parallèle.

II.2. Prototypage rapide pour l'implantation des algorithmes de traitement du signal et de l'image sur FPGA

Les FPGA sont des composants numériques reconfigurables dont on définit le fonctionnement interne par le biais d'un langage de description de type HDL (Hardware Description Language, par exemple le VHDL pour VHSIC HDL c'est-à-dire Very High Speed Intergrated Circuit HDL). On définit ainsi une architecture de traitement dans laquelle on peut utiliser toutes les ressources internes du composant, ainsi que les entrées/sorties, les communications avec d'autres composants, etc.

L'évolution rapide des techniques de fabrication et des technologies ont permis de voir la naissance de FPGA de plus en plus performants et de plus en plus complexes. On trouve ainsi dans les FPGAs de dernière génération des processeurs embarqués, des liens de communications ultrarapides, et bien d'autres fonctionnalités. Ils sont aussi de plus en plus denses (jusqu'à 1 milliard de portes logiques équivalentes) et de plus en plus rapides (jusqu'à 500 MHz). Tout cela permet la réalisation de systèmes très complexes et très performants. De plus, le potentiel de reconfigurabilité statique et dynamique permet de tenir compte de l'évolution des standards et ainsi de modifier l'architecture interne alors que la carte et le reste des composants restent figés. Les FPGAs sont souvent utilisés soit pour le prototypage des systèmes complexes (par exemple SoC), soit comme des systèmes dédiés au traitement du signal et d'image.

II.2.1 – Flot de conception classique sur FPGA

Le flot de conception d'une application sur FPGA est habituellement réalisé en plusieurs étapes. Tout d'abord, les spécifications algorithmiques permettent de définir l'architecture par une démarche d'Adéquation Algorithme Architecture. On décrit ensuite cette architecture avec un langage HDL. On peut ensuite simuler le système, et le modifier si nécessaire. Viennent ensuite les phases de synthèse et de placement routage, qui consistent à déterminer quels éléments vont effectivement être utilisés dans le FPGA et comment ils vont être connectés entre eux, à quel endroit du composant ils seront placés, etc. Chaque phase nécessite une vérification du bon fonctionnement global et des timings, et si nécessaire des modifications.

Les langages de description matérielle comme VHDL ou Verilog, utilisés habituellement pour le développement de FPGA, sont de nature concurrente. La programmation en VHDL implique une bonne connaissance non seulement de l'algorithme mais aussi de l'architecture du FPGA et du compilateur utilisé. Pour mieux exploiter le parallélisme intrinsèque de l'algorithme, il faut exécuter les tâches de traitement en parallèle (non séquentiel) pour satisfaire aux contraintes temporelles. Pour la plupart des spécialistes du contrôle ou du traitement du signal qui sont souvent des chercheurs et des ingénieurs de développement en logiciel, ces langages matériels ne sont pas familiers et d'une utilisation parfois délicate. C'est probablement une des principales raisons qui freine la démocratisation de la technologie FPGA. Pour tenter d'apporter une réponse appropriée à ce problème, de nouveau langages de conception de circuits, basés sur le langage C, ont été proposés. On peut citer entre autres le langage System-C [Pan01] [Synopsys] de Synopsys ou le Handel-C [Celo01] de Celoxica.

II.2.2 – Quelques langages de programmation du FPGA

Handel-C

Commercialisé par Celoxica, Handel-C [Celo01] ajoute quelques extensions, comme le parallélisme ou la gestion d'une horloge, au langage ANSI/ISO-C. Selon ses concepteurs, il permet à des ingénieurs en logiciel de réaliser des circuits numériques très rapidement. Le style de programmation est en effet identique à celui du C standard et il est inutile d'étudier la philosophie des langages exclusivement dédiés au matériel. En particulier, Handel-C permet d'exprimer très facilement le parallélisme matériel et les synchronisations. En voici les principales caractéristiques :

- une affectation de variable prend exactement un cycle d'horloge,
- les tailles des variables (y compris celles liées aux entrées/sorties) peuvent être définies au bit près, ceci afin d'occuper le moins de ressources possibles,
- les constructions parallèles sont autorisées ; les branches les plus rapides attendent la fin de l'opération de la branche la plus lente avant de continuer,
- il existe des canaux de communication entre processus,
- des primitives *ram* et *rom* permettent de gérer directement les blocs de mémoire enfouis dans les FPGA ou bien externes.

Dans le cadre des systèmes embarqués dédiés au traitement d'images, Handel-C est de plus en plus utilisé pour le prototypage rapide des systèmes numériques. Nous pouvons citer par exemple les travaux de M. Edwards et B. Fozard [EdFo] et ceux de S.M. Loo et *al.* [Loo02]. M. Edwards et B. Fozard ont développé un environnement de conception des systèmes mixtes : Mariachi. Dans un premier temps,

l'algorithme de traitement d'images a été spécifié en langage C. Ensuite, les tâches de traitement ont été réparties entre la réalisation logicielle et la réalisation matérielle en fonction d'un certain critère de partitionnement. Pour les modules destinés à la réalisation matérielle, une translation du C en Handel-C est nécessaire. Ils valident leur cycle de développement sur une carte électronique insérée dans un PC. Les modules matériels, définis en Handel C, sont synthétisés pour le FPGA Virtex XCV1000 de la société Xilinx en utilisant une librairie prédéfinie. Quant aux modules logiciels, ils sont implémentés sur le PC. S.M. Loo et al. ont effectué une comparaison entre VHDL et Handel-C avec l'implémentation de la Transformée Cosinus Discret. Ils ont montré que 1). Le nombre de lignes de programme en Handel-C est beaucoup plus petit que celui en VHDL, 2). La vitesse de traitement est comparable entre les deux versions d'implantation et 3). L'implémentation avec Handel-C utilise à peu près 2 à 5 fois plus des ressources matérielles que l'implantation directe en VHDL.

System-C

Après avoir constaté que C et C++ sont actuellement parmi les langages les plus utilisés, la société Synopsys a développé un nouvel environnement basé sur C++. L'un des objectifs de System-C est de fournir un outil permettant la description et la simulation de systèmes incluant des parties logicielles et matérielles. Une fois les tests achevés, le compilateur CoCentric System-C génère une *netlist* exploitable par les outils de placement-routage. Soutenu par des compagnie telles Xilinx, Synopsys, Innoveda ou Cadence, System-C concurrencera certainement VHDL.

JHDL

Les chercheurs provenant des milieux académiques ou industriels proposent aussi d'autres alternatives à VHDL et Verilog pour faciliter la conception de circuits numériques comme par exemple JHDL [JHDL]. Lors du développement d'un système comportant des composants logiciels et des modules fonctionnant sur un FPGA, la méthode de travail usuelle consiste à décrire et tester séparément logiciel et matériel. JHDL [JHDL] permet la description, la simulation et l'exécution de l'application au moyen d'un unique langage issu de *Java*. JHDL propose un ensemble de classes modélisant les divers composants d'un circuit numérique comme des composants logiques, des bascules ou des additionneurs. Chaque famille de FPGA dispose de plus d'une bibliothèque définissant ses particularités. Il est ainsi possible d'instancier une LUT d'un Virtex. Ce langage implique de bonnes connaissances du matériel utilisé.

Programmation graphique

Si l'apparition de compilateur C-vers-VHDL a amélioré l'accessibilité à la technologie FPGA, les langages textuels restent un frein car ils ne permettent pas de décrire de façon intuitive la conception de la logique du circuit. Décrire en lignes de texte des tâches parallèles ou des routines de cadencement,

par exemple, se révèle particulièrement malaisé [Ten04]. Basé sur cette idée, des environnements graphiques de programmation ont été proposés. Par exemple, Labview [Labview] propose des outils graphiques de haut niveau, pour réaliser la simulation HIL (Hard-ware-in-the loop), en particulier de capteurs sur des circuits FPGA.

Une autre option est l'environnement Matlab-Simulink plus XSG (Xilinx System Generator). A.T. Moreo et *al*. [ML05] ont illustré les avantages de cette approche. Matlab-Simulink-XSG [Matlab] [XSG] offre un environnement idéal pour le prototypage rapide des algorithmes de vision. Il combine les capacités et les toolbox fournis par Matlab avec la puissance de langage de description matérielle comme VHDL. Il permet de simuler et de tester en même temps les modules logiciels et matériels, donc applicable comme un environnement de co-design. Des blocs graphiques correspondant aux tâches de traitement peuvent être soit spécifiés par le fichier Matlab (haut niveau d'abstraction), soit modélisé par XSG hardware blocs en utilisant la librairie d'IP de Xilinx. Une fois le système testé, les blocs matériels peuvent être synthétisés et téléchargés sur un composant programmable comme un FPGA.

Nous pouvons aussi citer l'environnement Scilab-Scicos-SynDExIC développé à l'INRIA de Rocquencourt [Camp05] en collaboration avec l'ESIEE de Paris [Niang05]. Cet outil universitaire fournit des fonctionnalités similaires à l'environnement Matlab-Simulink-XSG, mais en libre service.

II.2.3 – Quelques outils récents de programmation des FPGA

Pour la programmation de FPGA, nous pouvons trouver une large palette d'outils et de langages depuis le haut niveau d'abstraction où la configuration de FPGA est générée à partir des descriptions algorithmiques jusqu'aux outils structurels où la configuration de FPGA est synthétisée à partir des blocs spécifiques FPGA. Il existe un compromis entre le temps de développement et l'efficacité matérielle. Une durée relativement importante de développement et les connaissances spécifiques des experts sont nécessaires pour obtenir de hautes performances matérielles. Traditionnellement, les programmeurs de FPGA choisissent les outils en fonction de leur coût, de leur performance, et aussi de la facilité d'utilisation.

Dans ce contexte, K. Benkrid et *al*. [Ben07] ont proposé un flot de développement dans le but d'atteindre les hautes performances matérielles et en même temps, de garder la facilité d'utilisation, en utilisant un unique environnement de développement. Ils ont intégré les deux langages Handel-C et HIDE. Le dernier consiste en une description structurelle et matérielle développé à Queen's University of Belfast en basant sur le graphe du flot de signal (Signal Flow Graphs). Un algorithme classique de traitement d'images (convolution 2D) a été d'abord testé et implémenté sur un Virtex-E, respectivement

avec les langages Handel-C et HIDE. Ensuite, conforté par la comparaison des résultats obtenus, ils ont inséré HIDE dans l'environnement de développement Handel-C (voir Figure II-1). Les auteurs pensent que la meilleure approche d'implémentation matérielle ne peut être obtenue qu'en confrontant plusieurs flots de conceptions et en tirant le profit des points forts de chacun.

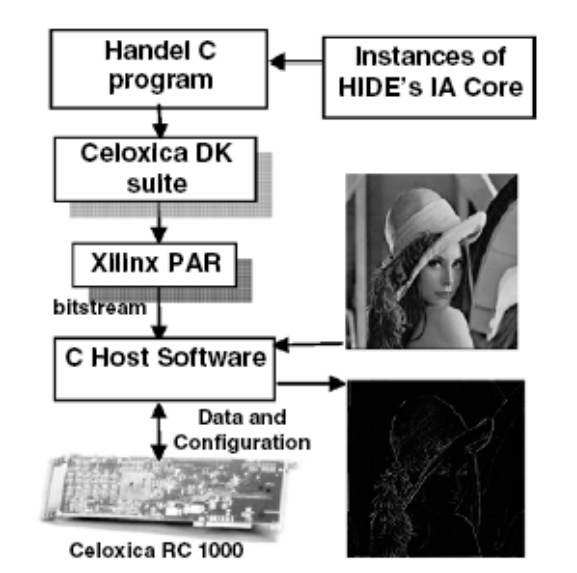


Figure II.1 : *Intégration de deux langages dans un unique environnement (cf.* [Ben07]).

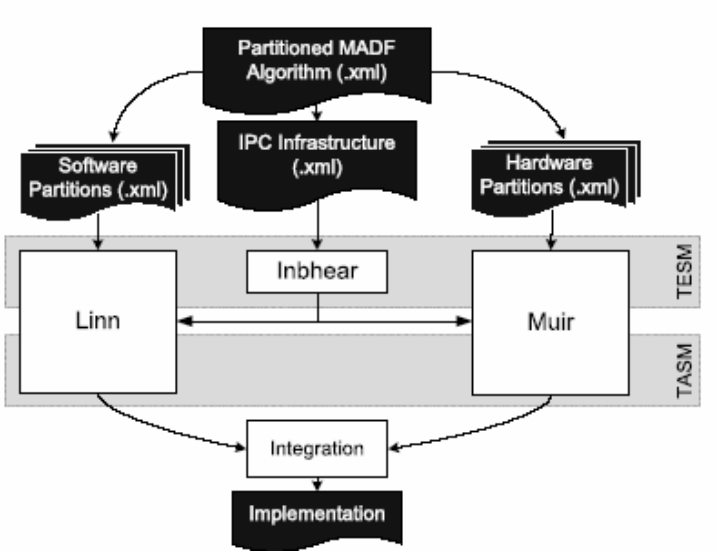


Figure II.2 : Architecture du projet d'implémentation *Abhainn (cf.* [McA07]).

La Figure II.2 illustre l'architecture du projet d'implémentation *Abhainn* [McA07] ciblé aux plateformes hétérogènes à base de FPGA. J. McAllister et ses collègues visent à transformer les expressions algorithmiques en langages d'exécution sur des systèmes embarqués. La particularité de leur approche se concentre sur les points suivants : 1) réalisation, exploration et optimisation des fonctionnalités matérielles dédiées, 2) capacité de traitements hétérogènes et 3) réalisation des communications inter processeurs. La méthodologie *Abhainn* est concrétisée à travers 4 principaux

processus : la spécification, le partitionnement, l'implémentation et la transformation. L'outil est composé de 3 modules interactifs : *Linn* pour le partitionnement logiciel, *Muir* pour le partitionnement matériel, et *Inbhear* pour le lien de communications inter processeurs. L'efficacité de *Abhainn* a été démontrée par deux exemples de développement : un FBF (Fixed Beamformer) et un ABF (Adaptative Beamformer). Pour le premier, l'implémentation avec une forte contrainte de ressources matérielles a été achevée, en illustrant la capacité de réalisation et d'optimisation des fonctionnalités matérielles dédiées. Dans le second, les auteurs montrent comment les fonctionnalités hétérogènes et les communications inter processeurs ont été réalisées.

Ces dernières années, les processeurs configurables émergent pour offrir la possibilité d'adapter un jeu d'instructions à une application spécifique par l'introduction des unités fonctionnelles configurables à l'architecture interne des processeurs. Avec leur capacité de reconfiguration, les FPGAs facilitent grandement le développement des processeurs RISP (Reconfigurable Instruction Set Processor). Il est certain que ces RISPs vont jouer un rôle très important dans les futurs plateformes embarquées de type SoC. Ceci est dû à leurs capacités prometteuses de relever les défis technologiques et de marché. La flexibilité de développement des RISPs en présence des blocs logiques reconfigurables permet d'envisager un RISP spécifique pour chaque application. Pouvoir facilement réduire le cycle de vie d'un produit est un élément incontournable pour des marchés incertains.

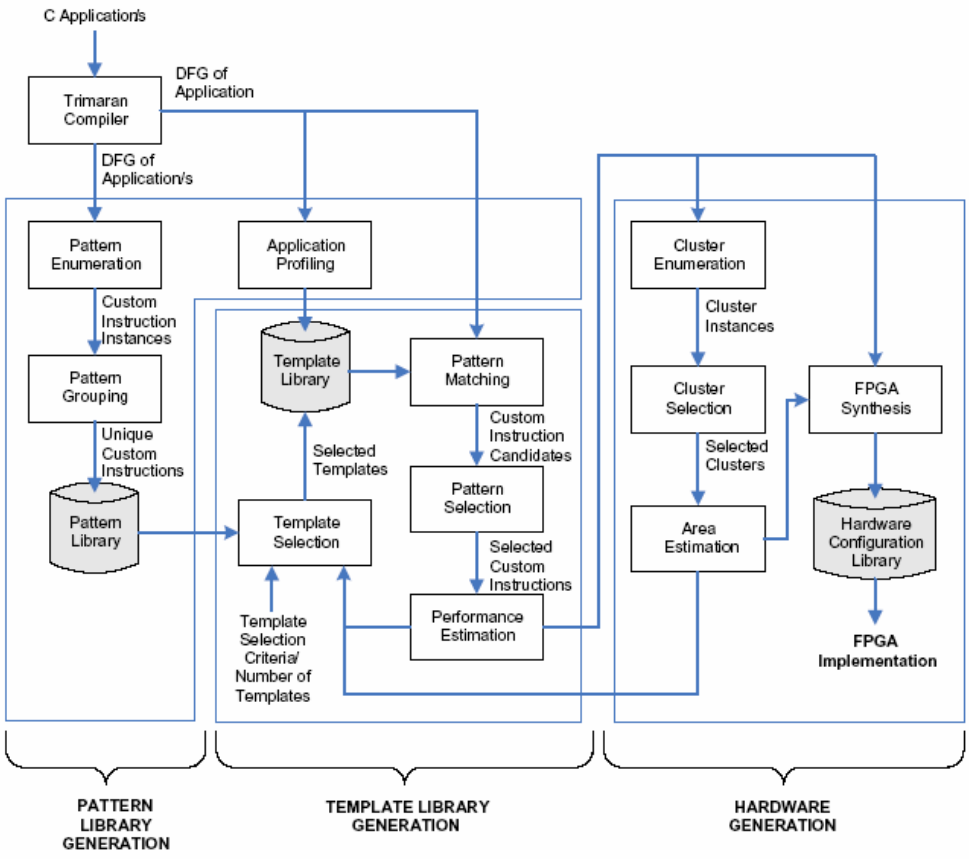


Figure II.3 : Flot de conception RISP proposé par S.K. Lam et T. Srikanthan *(cf.* [Lam09]).

S.K. Lam et T. Srikanthan [Lam09] ont introduit un flot d'exploration architecturale pour embarquer le RISP sur le FPGA avec une structure reconfigurable (voir Figure II.3). Pour une application donnée, celui-ci est capable d'identifier rapidement un jeu d'instructions réduit et d'indiquer son coût en surface sans passer par le processus de synthèse. Il peut aussi être utilisé pour définir la taille optimale des FPGAs en terme de coût et de consommation d'énergie. Les résultats expérimentaux obtenus avec 5 groupes d'applications (générique, automobile industriel, image, réseau, et sécurité informatique) montrent que le nombre de candidats d'instructions peut être réduit entre 30% et 70% avec une dégradation marginale des performances.

D'une manière similaire, K. Seto et M. Fujita ont présenté leur flot de développement pour générer des instructions optimales avec un nombre limité des ports d'entrée-sortie et une longueur fixe d'instructions comme pour le format RISC. Leur outil applique la technique de synthèse de haut niveau sur des sous graphes pour réaliser la génération et l'optimisation des unités fonctionnelles configurables. Quelques algorithmes de traitement d'images comme FFT, DCT et DJPEG ont été testés et implémentés sur le FPGA Altera Stratix II en utilisant l'approche proposée.

II.2.4 – Discussions sur les outils de la programmation du FPGA

La tendance actuelle en architecture vise à intégrer sur un seul circuit un système complexe logiciel et matériel. Des outils qui proposent un haut niveau d'abstraction et spécifient les composants logiciels et matériels en utilisant le même modèle facilitent la conception des systèmes mixtes. Dans ce contexte, les environnements graphiques sont plus adaptables au prototypage rapide des systèmes embarqués. Pour le moment, les fabricants proposent des outils d'interface entre Simulink et leurs composants matériels IP. La possibilité offerte à des non experts en électronique de concevoir et valider des composants matériels encourage la large diffusion des FPGAs.

L'approche C-vers-VHDL facilite grandement la description et la simulation des circuits numériques. Leur inconvénient majeur réside dans le fait que, sans connaissances en architecture, l'ingénieur de développement ne peut pas exploiter pleinement les ressources matérielles disponibles sur le FPGA pour réaliser des implantations optimales et performantes.

Dans les travaux précédents, nous avons proposé un flot de conception pour implémenter un DSP VLIW sur le FPGA [Brost]. Notre objectif consiste à garder la flexibilité de la programmation DSP pour réduire le cycle de développement, et en même temps, à profiter de ressources matérielles disponibles sur le FPGA pour augmenter les performances en temps réel. En introduisant cette approche, nous avons pu d'abord exploiter plus facilement le parallélisme intrinsèque d'une application

en utilisant le compilateur du processeur VLIW, ensuite, implémenter des unités fonctionnelles correspondant à l'aide d'un générateur du modèle VHDL du processeur. L'aspect le plus innovant de notre travail concerne la notion de modularité. En effet, chaque modèle VHDL généré ne contient qu'un nombre minium de unités ; chaque unité possède un jeu d'instruction minimum, justes nécessaires pour une application donnée. Cette notion M^3 (Minimum Mandatory Modules) nous a permis d'économiser les ressources matérielles nécessaires et de réduire la consommation électronique. Dans [Brost], V. Brost a visé l'architecture du DSP VLIW TMS320 C6201 et utilisé l'environnement Code Composer Studio (CCS) du TI (Texas Instrument) pour 1) transformer le code source C en Hexa, et 2) exploiter le parallélisme des instructions VLIW.

Durant cette thèse, nous voulons généraliser la méthodologie M^3 aux processeurs RISP. Par rapport aux travaux précédents, nous offrons plus de flexibilité à plusieurs niveaux : la longueur de l'instruction VLIW est modulaire (non fixée à 256 bits comme pour le DSP C6201), comme le nombre d'unités fonctionnelles (non fixé entre 1 et 8) et les types d'opérations de chaque unité.

Pour chaque application du traitement de signal et des images, nous utilisons d'abord l'environnement Trimaran pour convertir le code C ou C++ en une représentation intermédiaire. Ensuite, nous exposons et analysons toutes les opérations qui peuvent être exécutées parallèlement à l'aide d'un outil spécifique. L'objectif est de réduire au minimum le nombre de cycles d'exécution. Un modèle VHDL du processeur RISP optimal est automatiquement généré après. A la dernière étape, tous les fichiers VHDL seront regroupés et synthétisés avant d'être testés sur un FPGA.

Notre approche, dans un premier temps, fournit de multiples avantages par rapport aux outils existants :
Le flot de conception proposé possède plus de flexibilité au niveau de la programmation. Son entrée est composée de codes C ou C++. Il s'agit de sources classiques et standard, non orientées en matériel comme Handel-C ou System C. Grâce à nos outils spécifiques, les développeurs logiciel peuvent réaliser le prototypage rapide des applications sur le FPGA en ignorant complètement des aspects matériels (langage de description, architecture interne de FPGA …). Ceci constitue un facteur clé pour encourager la large utilisation des composants électroniques programmables.

Ce flot permet de réaliser des implémentations performantes et efficaces. L'extraction du parallélisme nous conduit à un nombre minimal de cycles d'exécution et en même temps, notre outil génère un modèle du processeur RISP utilisant les ressources matérielles justes nécessaires pour une application ciblée. Les bonnes performances en terme de « surface/vitesse » peuvent ainsi être obtenues.

Il s'agit d'un flot ouvert et adaptatif. Dans ce manuscrit, nous présentons seulement l'étape basique de la méthodologie M^3 en n'exploitant que le parallélisme de bas niveau. Avec la méthode proposée, nous

pouvons aussi profiter du parallélisme de haut niveau, car les ressources matérielles ont été utilisées d'une manière optimale et économique. Grâce à la capacité adaptative de notre outil, avec une application donnée ou un FPGA ciblé, nous pouvons aussi effectuer l'exploration de l'espace architectural (Design Space Exploration) pour respecter simultanément les contraintes algorithmiques et architecturales.

II.3. Technologie avancée de compilation : environnement Trimaran

Trimaran est un environnement de développement destiné à expérimenter des compilations optimales pour des architectures à instructions parallèles. Cet environnement est issu de la collaboration de plusieurs universités et centres de recherches. On notera la présence du CAR (Compiler and Architecture Research Group) des laboratoires de Hewlett-Packard ainsi que le groupe de recherche IMPACT (Illinois Microarchitecture Project utilizing Advanced Compiler Technology) de l'Université de l'Illinois. Trimaran intègre un nombre d'outils importants de recherches concernant la compilation comme par exemple des stratégies d'allocation de registre ou des élaborations de pipelines logiciels et matériels. Dans cette section, nous présentons d'abord l'organisation générale de l'environnement Trimaran avant de détailler l'outil OpenIMPACT, compilateur utilisé dans notre flot de conception.

II.3.1 – Organisation de l'environnement Trimaran

Trimaran est composé de trois outils principaux (voir Figure II.4) :

- Le compilateur OpenIMPACT destiné à produire, à partir d'un programme écrit en langage C, un code intermédiaire appelé Lcode. Cette représentation est optimisée pour une architecture à instructions parallèles, mais n'est pas déterminée pour une cible précise.
- Le compilateur ELCOR utilisé pour générer un code pour une architecture donnée spécifique. Cette architecture est paramétrée à l'aide de fichiers HMDES (High Machine DEScription langage) pour décrire son modèle. Cet outil produit un code intermédiaire nommé REBEL.
- Le simulateur facilite la mise au point du code généré pour une machine spécifique par le compilateur ELCOR. Il permet d'exécuter le code REBEL et de fournir différentes statistiques concernant le programme simulé.

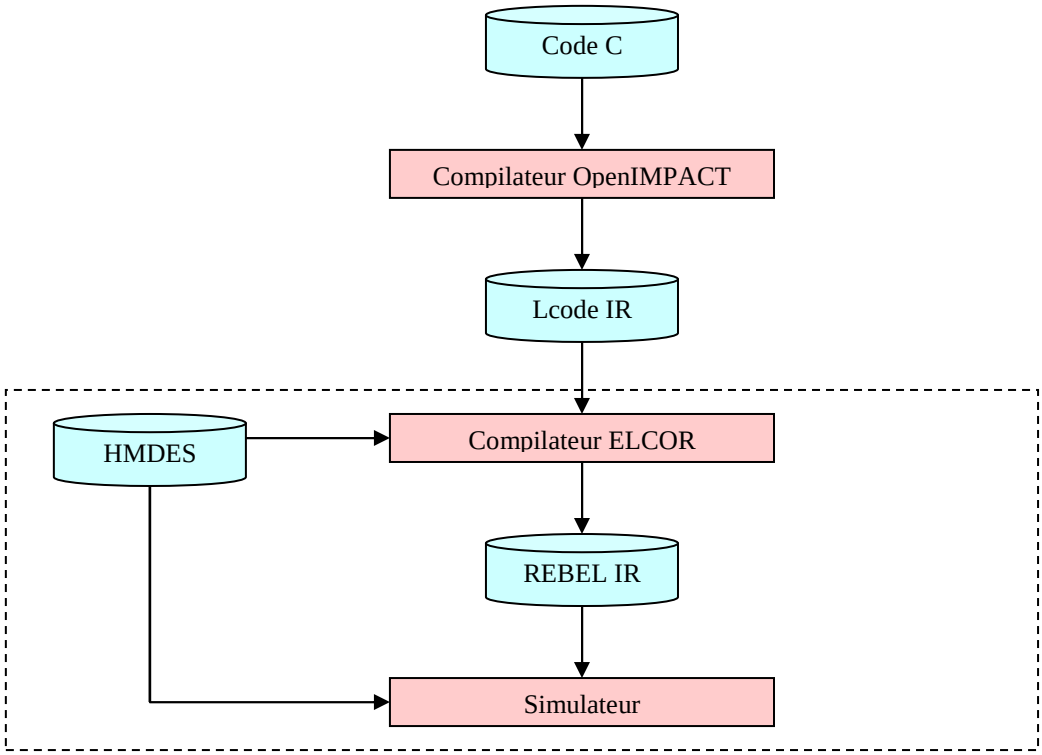


Figure II.4 : Organisation de l'environnement Trimaran *(cf.* [Laks05]).

L'environnement Trimaran intègre les éléments suivants :

- Une architecture paramètrable de type ILP (Instruction Level Parallelism) nommé HPL-PD [Kath00] [Tri].
- Un compilateur fortement optimisé, complètement ouvert, et conçu pour être facilement modifié et adapté par des spécialistes des recherches dans le domaine de la compilation. La représentation intermédiaire Lcode bénéficie des dernières techniques modernes de compilation comme le contrôle des flux et des dépendances de données.
- Un simulateur optimisé pour tester les architectures de type HPL-PD, facilement paramétrable, permet de disposer d'informations relatives pour la phase d'exécution comme l'utilisation des ressources matérielles du processeur ainsi que la fréquence d'appel des branchements.
- Une interface graphique pour les utilisateurs GUI facilite la configuration et le paramétrage de l'environnement Trimaran (voir Figure II-5).
- Divers outils graphiques qui permettent la visualisation des représentations intermédiaires et des performances de compilation et de simulation.

Les concepteurs d'origine de Trimaran, dès le début du projet, ont voulu en faire un environnement très ouvert, largement documenté, à destination de la communauté travaillant sur des architectures multi-processeurs. A l'heure actuelle, Trimaran est utilisé par beaucoup d'universités dans le monde et de nombreux articles de journaux et de conférences témoignent de sa grande utilité dans les domaines des

langages de programmation, des compilateurs et des architectures matérielles des ordinateurs [Cart99] [Smel03] [Rabb03] [Rabb04].

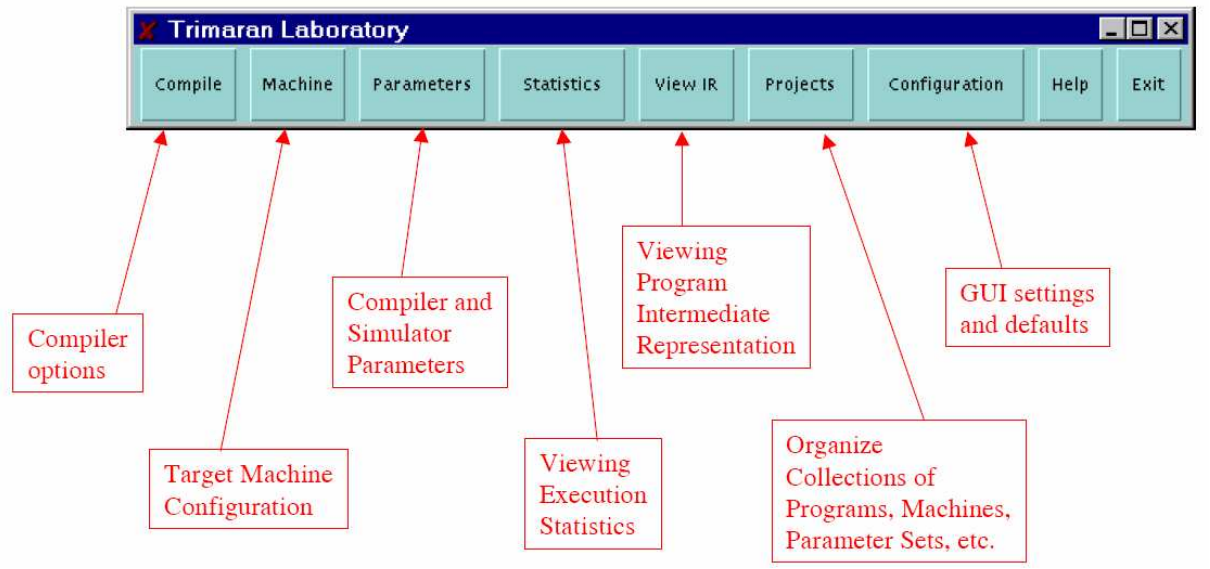


Figure II.5 : Interface graphique de l'environnement Trimaran *(cf.* [Tri]).

II.3.2 – Présentation du compilateur OpenIMPACT

L'infrastructure du compilateur OpenIMPACT est développée pour permettre la conception, l'implémentation et le test de nouvelles optimisations. Celles-ci concernent d'une part la compilation et d'autre part l'évaluation de nouvelles architectures muti-processeurs innovantes. La particularité d'OpenIMPACT est d'offrir la possibilité d'expérimenter des compilations orientées ILP.

OpenIMPACT est un compilateur multi passes. Le code généré à la sortie est largement contrôlé par un « Frontal » de compilation qui effectue une série de transformation du langage source d'entrée. Nous pouvons trouver dans la figure II-6 les différentes phases correspondantes :

- Une technique de « Frontal » développée par EDG (Edison Design Group) analyse le code C et génère un arbre syntaxique nommé Pcode,
- Avant de convertir ce Pcode en Lcode, une série d'optimisations est effectuée : « flattening », « profiling », « inlining » et « inter procedural analysis »,
- La représentation intermédiare Lcode, subit elle-même aussi une série de transformations destinée à faciliter l'implémentation finale sur une cible de processeur définie par l'utilisateur.

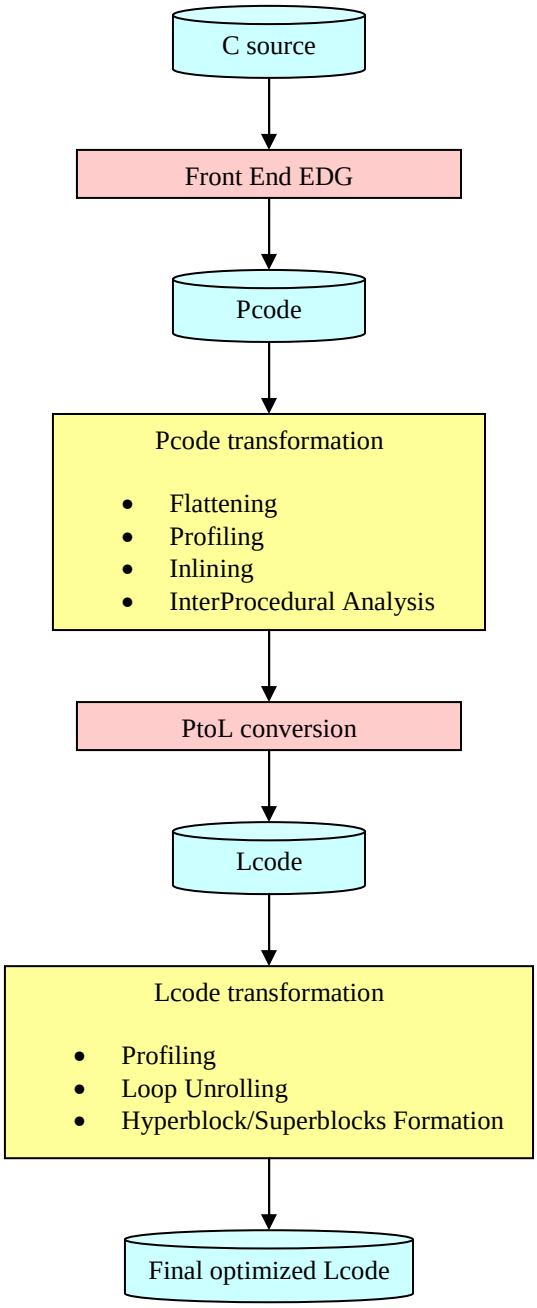


Figure II.6 : Les différentes phases effectuées par le compilateur OpenIMPACT.

OpenIMPACT, en plus de sa polyvalence, possède une robustesse et des performances comparables avec des compilateurs classiques comme *gcc* par exemple. De nombreux tests ont été réalisés et présentés au symposium international sur les architectures de processeur en 2004 [Sias04]. Nous l'avons choisi pour deux raisons principales : les hautes performances pour des architectures muti-processeurs et l'accès facile à la représentation intermédiaire qui est indépendante des cibles matérielles.

II.4. Conclusions

Dans ce chapitre II, nous avons exposé les différentes tendances pour le prototypage sur le FPGA. La plupart des outils proposent des environnements de développement pour faciliter la conception des circuits sur puces, mais souvent, des connaissances spécifiques matérielles sont encore nécessaires implicitement à plusieurs niveaux.

Notre objectif est d'élaborer une méthodologie permettant aux non-spécialistes en électronique de pouvoir intégrer facilement leurs algorithmes au sein des FPGAs. En utilisant notre chaîne de développement, ils peuvent rester dans leur environnement habituel de programmation en ignorant complètement les aspects matériels.

Le domaine de traitement d'images est caractérisé par un fort volume de données (ex. séquences vidéo), souvent associé à des calculs fortement itératifs. Pour répondre aux contraintes en temps réel, il est nécessaire d'extraire les parallélismes intrinsèques de chaque application. L'environnement Trimaran est conçu pour exposer d'une manière optimale, les parallélismes des algorithmes à plusieurs niveaux, en particulier au niveau instructions.

Chapitre III
Présentation du flot de conception proposé

Introduction

Nous présentons dans ce chapitre la méthodologie proposée pour le prototypage rapide des applications de traitement d'images en temps réel sur la cible FPGA. Nous avons conçu, sur circuit reconfigurable, un modèle modulaire du processeur de traitement d'image dont le jeu d'instructions est variable et sera ciblé pour une application donnée. En utilisant ce flot de conception, nous programmons un algorithme en C comme pour un développement sur tout processeur. Les outils développés vont traduire ce code en VHDL en essayant de profiter au maximum du parallélisme intrinsèque des algorithmes traités. L'objectif est de garder la souplesse et la flexibilité des processeurs et en même temps de profiter des ressources matérielles disponibles au sein des FPGAs, notamment leur grande aptitude à paralléliser des traitements, afin d'augmenter les performances en temps réel.

Nous commençons le chapitre par une présentation générale du flot de conception proposé. Ensuite, nous détaillons chaque étape de celui-ci : le développement de l'application dans un langage informatique, la parallélisation du programme assembleur intermédiaire, l'analyse des ressources matérielles et la génération du modèle VHDL du processeur. Un exemple de détection de contours avec le filtre de Sobel suivra et quelques résultats concernant l'implantation d'algorithmes courants de traitement d'image terminerons la présentation à la fin du chapitre.

III.1. Présentation générale du flot de conception proposé

L'étude menée ici consiste à prototyper des applications de traitement d'images en utilisant un modèle VHDL d'un processeur spécifiquement conçu et optimisé par une exploitation des parallélismes pour une application donnée. La figure III.1 affiche le flot de conception proposé. Nous partons d'un programme initialement de traitement d'image écrit en C. Nous obtenons le code assembleur générique intermédiaire nommé *Lcode* à l'aide du compilateur OpenIMPACT présent dans l'environnement Trimaran décrit dans le chapitre II de ce manuscrit. Ensuite, nous analysons ce code machine afin d'en effectuer une parallélisation automatique sous contrainte de dépendance de données. Une phase d'analyse du programme parallélisé est ensuite effectuée pour connaître les ressources matérielles nécessaires de l'application : les registres nécessaires, les chemins de données ainsi que les différents opérateurs de calcul. Ces différentes informations constituent une base de données matérielle et nous permet de générer les fichiers VHDL correspondant à un modèle de processeur taillé pour l'application. Les étapes de parallélisation, d'analyse et de génération des codes VHDL sont automatiquement réalisées au moyen des outils développés sous Visual C++. A la fin, ces fichiers VHDL sont synthétisés et implémentés sur un FPGA.

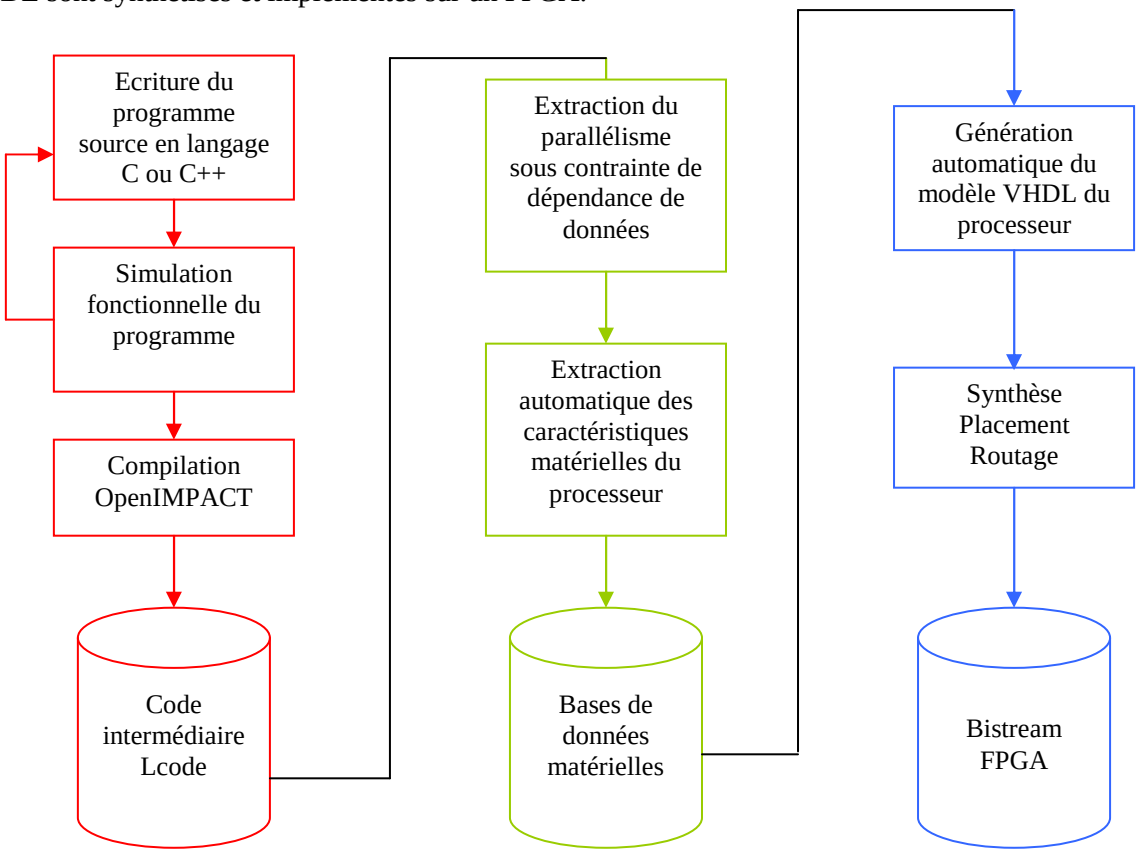


Figure III.1 : *Schéma général du flot de conception.*

Il est à noter que notre modèle de processeur possède des ressources matérielles juste nécessaires pour exécuter l'application ciblée. Ceci peut permettre d'implanter plusieurs processeurs dans un seul FPGA afin de former une machine en parallèle.

Nous décomposons le flot de conception proposé en quatre étapes principales que nous détaillons par la suite :

- développement de l'application dans un langage informatique et compilation,
- extraction automatique du parallélisme sous contrainte de dépendance de données,
- analyse des ressources matérielles juste nécessaires pour l'application,
- génération du modèle VHDL du processeur taillé pour l'application.

III.2. Développement de l'application dans un langage informatique et compilation

De façon classique, le programme de l'application est écrit à l'aide des langages informatiques proposés par un environnement de développement. On utilise ici le compilateur OpenIMPACT proposé dans l'environnement Trimaran. Cet outil a été choisi ici car il dispose d'un grand nombre d'options de compilation qui vont nous permettre d'optimiser la phase de parallélisation de notre flot de conception. De plus, ce compilateur génère un assembleur générique indépendant d'une cible particulière de processeur.

Après l'écriture du programme, le fonctionnement des algorithmes est simulé à l'aide des outils proposés par Trimaran. Cette phase permet de réaliser deux opérations importantes dans le cas d'un prototypage rapide :

- Une validation fonctionnelle de l'application envisagée,
- Une évaluation des performances en terme d'exécution des traitements.

En effet, le compilateur de l'environnement Trimaran propose un ensemble d'optimisation de compilation comme par exemple le déroulage de boucle ou l'ajout de macro-instructions spécifiques au traitement du signal (MAC multipliate/accumulate par exemple). L'ajout de ces instructions en fonctions des caractéristiques du circuit FPGA peut être très intéressant en terme de performance de traitement final. Une fois l'étape de simulation validée, le code de représentation intermédiaire Lcode est généré.

Pour cette phase de développement du programme, le processus est strictement identique à un prototypage rapide destiné à une implantation avec un processeur standard. Seul le fichier Lcode est

utilisé dans l'étape suivante du flot de conception : parallélisation automatique sous contrainte de dépendance de données.

III.3. Extraction automatique du parallélisme sous contrainte de dépendance de données

L'idée d'extraire le parallélisme intrinsèque d'un programme et d'exécuter plusieurs instructions simultanément sur un processeur n'est pas nouvelle. Nous pouvons citer par exemple la famille de DSP C6x de Texas Instrument au sein duquel huit unités de calcul peuvent opérer en parallèle. L'objectif, pour le compilateur de ces DSP, est d'optimiser le placement des instructions parallélisées sur les différentes unités présentes. Ici, nous proposons d'élaborer un modèle de processeur non figé. A priori, aucune limite ne sera donnée en ce qui concerne le nombre de registres utilisés ou le nombre d'opérateurs de calcul pouvant être utilisés simultanément. Ces éléments proviendront de la phase d'extraction du parallélisme de l'algorithme.

III.3.1 – Contenu du fichier code machine

Le fichier Lcode représente le résultat de la compilation d'un programme traduit en langage assembleur générique indépendant d'un type spécifique d'une cible de processeur. Cet assembleur est du type « opérande à registre » et n'utilise que deux opérations (LOAD/STORE) pour l'accès à la mémoire. On trouve un exemple de langage Lcode dans le tableau III.1. Chaque ligne comporte une instruction basique composée d'un code opération, au moins d'un registre opérande et d'un registre résultat.

Nous n'allons pas décrire ici de façon exhaustive le langage intermédiaire Lcode. On y retrouve des instructions arithmétiques, logiques et de contrôle présentes dans tout assembleur. Par exemple, dans le tableau III.1, l'instruction de la première ligne représente une addition du contenu des deux registres R64 et R12 dans le registre résultat R1. Les opérations présentes dans cet exemple constituent une suite d'instructions **séquentielles** générées par la compilation d'un programme. Nous allons maintenant voir le principe d'extraction du parallélisme du langage Lcode.

Tableau III.1 : Extrait d'un fichier Lcode.

```
add [(r 1 i)] [(r 64 i)(r 12 i)]
add [(r 67 i)] [(r 58 i)(r 7 i)]
lsl [(r 68 i)] [(r 8 i)(i 1)]
sub [(r 69 i)] [(r 67 i)(r 68 i)]
lsl [(r 70 i)] [(r 9 i)(i 1)]
add [(r 71 i)] [(r 69 i)(r 70 i)]
sub [(r 72 i)] [(r 71 i)(r 10 i)]
abs [(r 14 i)] [(r 1 i)]
```

III.3.2 – Exemple d'extraction du parallélisme

Chaque code intermédiaire est constitué d'une suite d'instructions séquentielles basiques n'utilisant que des registres. Si nous analysons la dépendance de données présentes dans la suite d'instruction, nous pouvons voir qu'un certain nombre d'opérations peuvent être exécutées de façon parallèle. Il est à noter que pour les instructions d'accès à la mémoire, il y aura une contrainte supplémentaire concernant le nombre d'accès simultanés à l'espace mémoire. Cette contrainte est imposée par le contrôleur d'accès mémoire disponible dans le composant FPGA. Pour l'ensemble des exemples ci-dessous, ce nombre d'accès simultanés sera limité à quatre.

Considérons le code intermédiaire réorganisé suivant constitué de neuf instructions séquentielles :

1	R1 = Load @1
2	R2 = Load @2
3	R3 = Mul (R1,R2)
4	R4 = Load @3
5	R5 = Add (R3,R4)
6	Store (R5,@4)
7	R6 = Load @5
8	R7 = Add (R6,1)
9	Store (R7,@6)
10	…

En analysant la séquence d'instructions et en respectant les contraintes de dépendance de données et d'accès à l'espace mémoire, nous obtenons la suite de traitements parallèles représentée sous la forme de graphe ci-dessous :

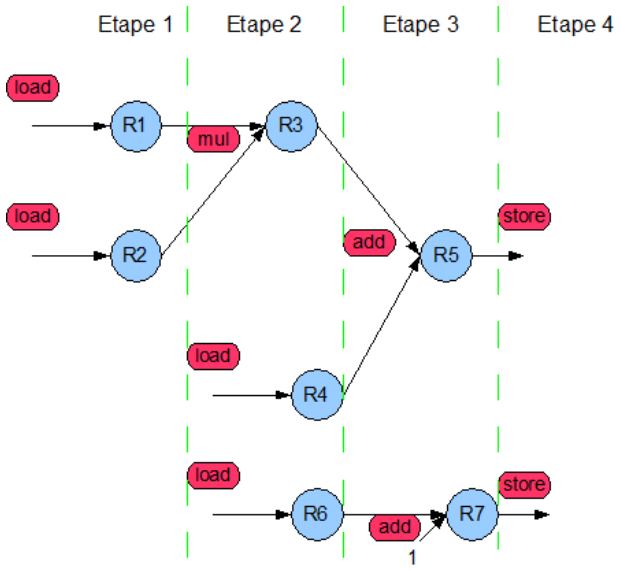


Figure III.2 : *Extraction du parallélisme au niveau des instructions.*

Nous voyons que le traitement séquentiel précédent constitué de neuf étapes d'exécution a été transformé dans une suite parallèle constituée de quatre étapes (voir aussi le tableau ci-dessous). Dans cet exemple, le nombre d'opérateurs de traitement capable de travailler simultanément à implanter dans le modèle VHDL du processeur sera de quatre. On peut définir une métrique pour évaluer le taux de parallélisation **Tp** = nombre d'instructions de la suite séquentielle / nombre d'instructions de la suite parallèle. Dans cet exemple : **Tp = 9/4 = 2,25**.

1	R1 = Load @1	R2 = Load @2	R4 = Load @3	R6 = Load @5
2	R3 = Mul (R1,R2)	R7 = Add (R6,1)		
3	R5 = Add (R3,R4)	Store (R7,@6)		
4	Store (R5,@4)			
5	….			

III.3.3 – Outil d'extraction du parallélisme

L'outil d'extraction a été développé en langage C++. Il a deux rôles principaux :

- Paralléliser sous contrainte de dépendance de donnée et d'accès à la zone mémoire,
- Réduire le nombre de registres utilisés.

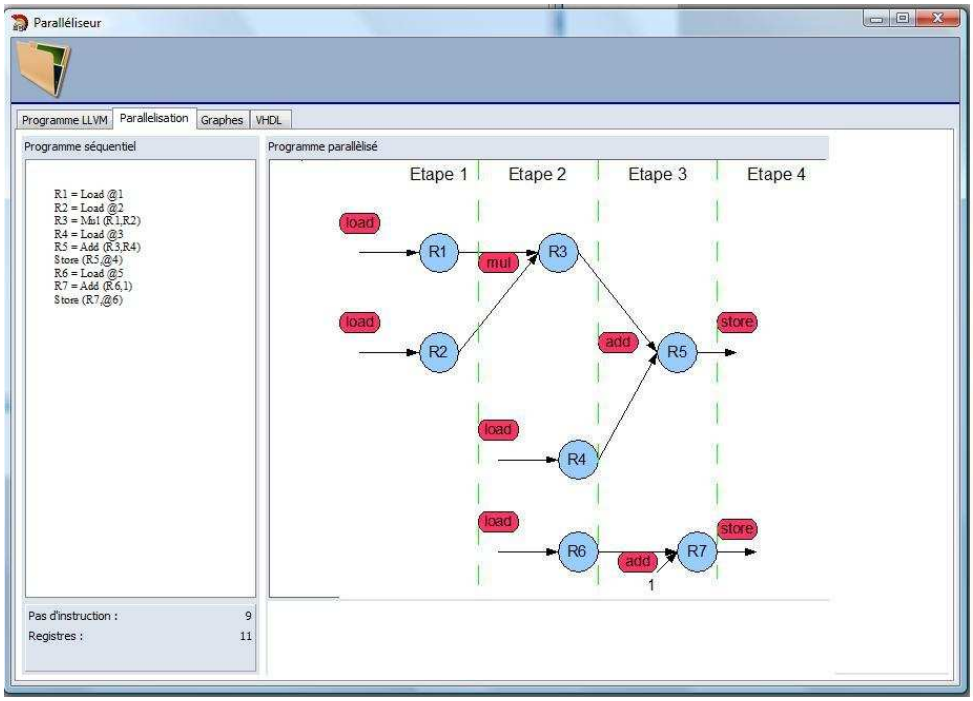


Figure III.3 : *Vue graphique de l'outil développé pour l'extraction de ILP.*

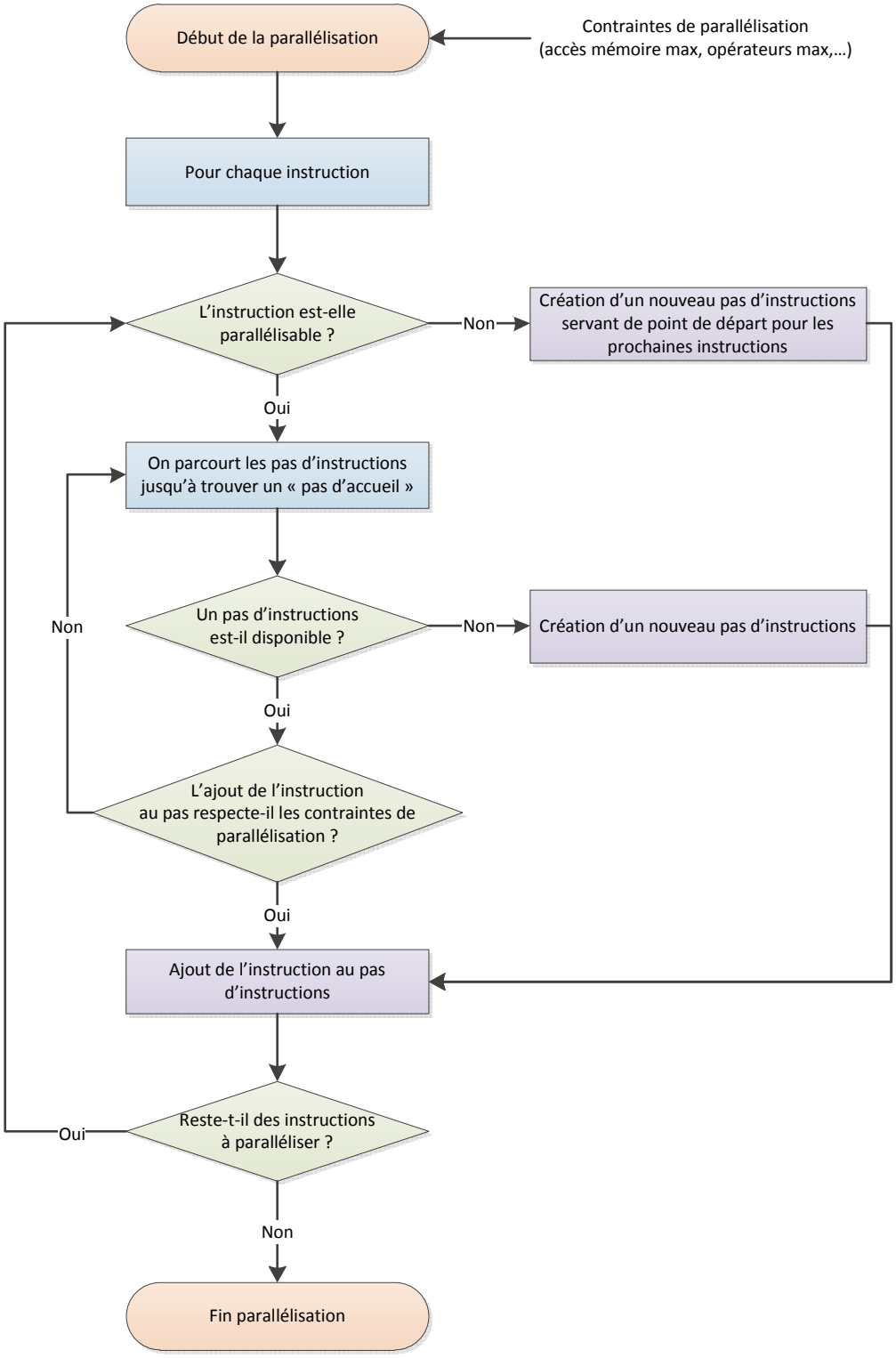


Figure III.4 : Principe d'extraction du parallélisme.

La Figure III.4 illustre le principe suivi pour extraire le parallélisme au niveau des instructions. A l'entrée de l'outil, on précise les contraintes à respecter : nombre d'accès simultanés à l'espace de mémoire, nombre maximum d'opérateurs fonctionnels ou d'autres. Pour chaque instruction séquentielle, on vérifie d'abord si elle n'est pas une instruction spécifique (saut, branchement …) pour

décider de la création d'un nouveau pas d'instruction dans le processus parallèle. Ensuite, on examine si l'ajout de l'instruction au pas courant respecte la dépendance de données. Toutes les instructions séquentielles de Lcode subissent le même traitement avec cette étape d'extraction du parallélisme au niveau des instructions.

Réduire le nombre de registres utilisés consiste en un processus d'optimisation. Ceci permet de réduire la complexité du processeur RISP généré et de minimiser les ressources matérielles nécessaires. Pour chaque utilisation d'un registre, on vérifie qu'il n'existe pas un autre registre disponible sur la même plage d'utilisation, pouvant le remplacer. Par exemple, R1= R2 + R3 pourrait devenir R2= R2 + R3 ou R3=R2 + R3.

III.4 Analyse automatique des caractéristiques matérielles du processeur

Après l'extraction du parallélisme, nous allons décrire la phase qui consiste à assembler les différents éléments matériels nécessaires pour générer un modèle VHDL du processeur taillé pour une application donnée. Ceci passe par une analyse automatique du code intermédiaire Lcode parallélisé fourni par l'étape de développement décrite dans la section précédente.

L'outil d'analyse a été développé en langage C++. Il est organisé autours de plusieurs modules. A partir du fichier Lcode parallélisé, chaque module analyse les lignes contenues dans le fichier et extrait les informations qui lui sont utiles. Le regroupement de tous les résultats obtenus permet de construire une base de données contenant une liste exhaustive des caractéristiques matérielles du processeur à générer.

Module d'analyse des opérateurs utilisés

Ce module recense les différents opérateurs de calcul nécessaires par le processeur pour l'application. On retrouve toutes les sortes d'opérations classiques : logiques, arithmétiques et de contrôle.

Programme d'extraction phase 1 : *Recensement des opérateurs*

Opérateurs utilisées : *4*

Module d'analyse des registres utilisés

Après la phase précédente de parallélisation qui inclut aussi une phase de réduction de registres, ce module analyse le nombre et le nom des registres utilisés :

Programme d'extraction phase 2 : *Analyse des registres*

Registres utilisés : R0, R1, R5, R7, R8

Module d'analyse des registres en entrée pour chaque opérateur

Lors de l'exécution d'une instruction, deux opérandes d'entrée sont nécessaires. Celles-ci proviennent des bancs de registres. Ce module établit une liste des registres en entrée pour chaque opérateur.

Programme d'extraction phase 3 : *Analyse des registres d'entrée*

Opérateur 1 :
Opérande 1 : R0, R5
Opérande 2 : R8, R5, R9

Opérateur 2 :
Opérande 1 : R1, R4, R6
Opérande 2 : R9, R0

Opérateur 3 :
Opérande 1 : R1, R2, R3
Opérande 2 : R0, R1, R1

Opérateur 4 :
Opérande 1 : R5
Opérande 2 : R5, R0

Module d'analyse des opérateurs autorisés à écrire dans chaque registre

Après l'exécution d'une instruction, un opérande de sortie est disponible. Celle-ci doit être transférée dans un registre. Le module permet de répertorier, pour chaque registre utilisé, les opérateurs qui auront accès en écriture à celui-ci.

Programme d'extraction phase 4 : Analyse des registres de sortie

R0 :	op 1, op 4
R1 :	op 1, op 3
R5 :	op 2, op 3, op 5
R8 :	op 2
R2 :	op 3, op 2
R9 :	op 4
R11 :	op 3
R15 :	op 5

L'ensemble des résultats obtenus par les différents modules constitue la base de données matérielle associée au processeur. Cette base sera utilisée pour la dernière étape de notre flot de conception : la génération du modèle VHDL du processeur.

Un module statistique, associé à la base de donnée permettrait la visualisation rapide d'une synthèse des différents éléments matériels qui constituent le processeur. Ceci pourrait donner une idée de la taille finale du processeur dans le FPGA.

III.5. Génération automatique du modèle VHDL du DSP C6201

L'idée d'embarquer un processeur dans un FPGA n'est pas nouvelle. Ce processeur peut être inséré lors de la fabrication du FPGA. Nous pouvons citer par exemple un cœur Power PC pour le Virtex II Pro [Power], E7V CSoc de Triscend [E7V] et Excalibur de la société Altera [Exalibur]. Une autre solution consiste à modéliser en langage matériel un processeur et l'implanter dans le FPGA lors du développement de l'application comme les processeurs MicroBlaze de Xilinx [Xilinx] et Leon SPARC V8 [Leon]. Ici, nous proposons de générer, pour une application donnée, un modèle VHDL qui n'intégre que les éléments matériels du processeur strictement nécessaires au fonctionnement de l'application ciblée.

A partir des données fournies par l'outil d'analyse des ressources matérielles du processeur, un troisième outil appelé générateur construit automatiquement un modèle VHDL adapté qui possède le nombre d'opérateurs de calcul et le nombre de registres justes nécessaires pour exécuter l'application envisagée.

III.5.1 – Outil de génération

L'outil de génération a aussi été développé en langage C++. Cet outil a besoin de trois types de données distincts :

- **des données constantes** : ce sont plusieurs fichiers VHDL prédéfinis pour le processeur et nécessaires pour toute application. On trouve par exemple dans ces fichiers une description matérielle du compteur de programme.
- **des données variables** : c'est la base de données matérielle fournie par l'outil d'analyse. Les deux éléments principaux concernent les registres utilisés et les instructions utilisées pour chacune.
- **des données de paramétrage** : elles sont fournies par l'utilisateur. Ces données de paramétrage permettent l'utilisation de différents périphériques. Ceux-ci peuvent êtres des

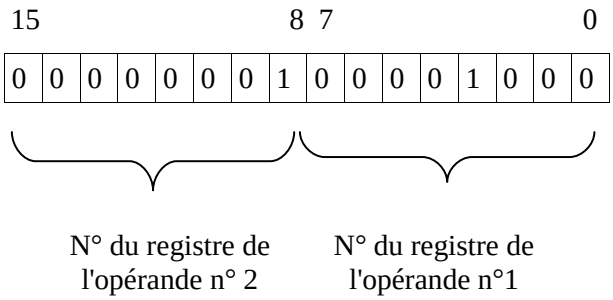
périphériques répondant à une interface AMBA par exemple, protocole qui s'impose pour les IP FPGA.

Dans la suite de cette section, nous allons détailler les principes d'élaboration du modèle VHDL pour comprendre le fonctionnement de celui-ci. Nous traiterons dans un premier temps de la construction de la mémoire de programme nécessaire pour chaque opérateur et chaque registre puis nous évoquerons la gestion du compteur de programme avant de terminer par la description des opérateurs de calcul ainsi que les chemins de données associés.

III.5.2 – Construction de la mémoire de programme et gestion du compteur de programme

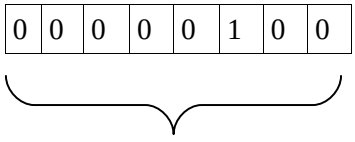
La mémoire du programme est composée de deux modules :

- Un module avec le programme qui définit, pour chaque opérateur, les différents registres d'entrée avec le mot d'instruction suivant :



  Pour cet exemple, le code indique une opération entre les registres R8 et R1.

- Un module avec le programme qui définit, pour chaque registre, le numéro de l'opérateur autorisé à écrire dans ce registre avec le mot d'instruction suivant :



  Pour cet exemple, l'opérateur 4 est autorisé à écrire dans le registre correspondant.

Le compteur de programme implanté dans le modèle du processeur est un compteur sur 32 bits. L'assembleur intermédiaire Lcode n'utilise qu'une seule instruction de branchement. La gestion du compteur va être relativement simple. Lors d'une instruction de saut le compteur vient prendre

l'adresse de branchement dans le registre utilisé. Il est à noter que le chargement de l'adresse de saut a une latence de 2.

III.5.3 – Génération des opérateurs de calcul et des chemins de données associés

Généralement, chaque opérateur de calcul reçoit deux opérandes en entrée, effectue un traitement et fournit un opérande de sortie. Les données dans le processeur sont stockées dans des registres de 32 bits. A partir de l'analyse du programme, le nombre de registres utilisés est déterminé. Le générateur de code VHDL n'implante donc que les registres nécessaires.

La génération des chemins de données en entrée d'opérateur de calcul

Chaque opérateur reçoit deux données d'entrée stockées dans des registres. Dans notre modèle VHDL, on considère chaque donnée d'entrée provenant de la sortie d'un multiplexeur de registre. Le générateur va créer automatiquement un ensemble de multiplexeurs pour les deux entrées de chaque opérateur. Prenons un petit exemple : l'analyse matérielle a déterminé pour un programme d'application que l'opérateur 1 a besoin des connections avec les registres suivants lors de l'exécution :

Opérande 1 de l'opérateur 1: registre R0, R4, R15, R3

Opérande 2 de l'opérateur 1 : registre R6, R8

Le générateur va donc construire les deux multiplexeurs :

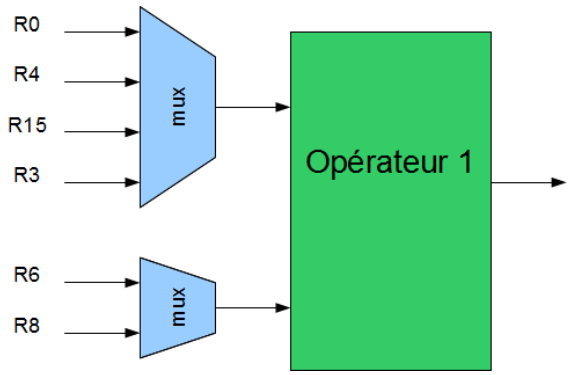


Figure III.5 : *Chemin de données en entrée pour l'opérateur 1.*

Les commandes des multiplexeurs sont pilotées par les bits représentant les numéros des registres d'entrée dans le code de l'instruction défini dans la mémoire programme de l'opérateur considéré (cf III.5.2)

Dans ce code d'instruction, on distingue le champ *src1* (bit 0 à 7) qui permet de déterminer le numéro du premier registre, le champ *src2* (bit 15 à 8) permettant d'identifier le numéro du deuxième registre. Il est à noter que le nombre de registres disponible est ici limité à 256.

La génération des chemins de données en sortie d'opérateur

Chaque unité de calcul stocke sa valeur de sortie dans un registre. Après la phase d'analyse du code, on connaît, pour chaque registre, la liste des opérateurs qui doivent accéder à celui-ci. Le générateur crée, pour chaque registre, un multiplexeur dont les entrées proviennent des sorties d'opérateur. Par exemple, si l'outil d'analyse a identifié les connexions suivantes :

R0 : contenu modifié par les opérateurs 1,2 et 5
R6 : contenu modifié par les opérateurs 3 et 4

Le générateur construit les deux multiplexeurs :

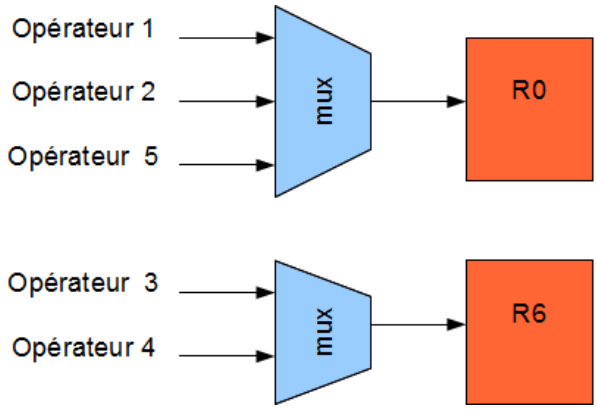


Figure III.6 : *Chemin de données en sortie pour les registres R0 et R6.*

Les commandes des multiplexeurs sont pilotées par l'octet représentant le numéro des opérateurs dans le code de l'instruction défini dans la mémoire programme du registre considéré (cf. III.5.2).

La génération des opérateurs de calcul

A l'aide de la base de données matérielle, le générateur fournit un module VHDL pour chaque opérateur de calcul. Un exemple est donné ci-dessous pour une génération de trois opérateurs :

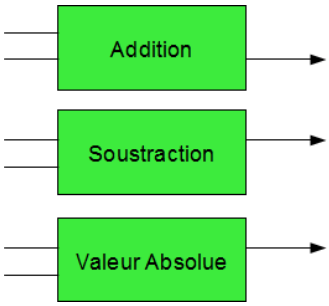


Figure III.7 : *Génération des opérateurs de calcul.*

La figure III.8 montre l'architecture simplifiée du modèle VHDL généré. On peut distinguer 5 opérateurs de calcul, les bancs de registres ainsi que les chemins de données associés. Les différents champs des mots d'instruction issus de la mémoire programme (signaux **mp**) pilotent les multiplexeurs d'entrées et de sorties.

La mémoire externe de donnée utilise une mémoire externe au FPGA. L'interface avec cette mémoire est assurée par le bief d'un contrôleur mémoire *multiport* (4). Dans notre modèle, ce sont les unités de calcul (par l'intermédiaire des opérations load/store) qui accèdent au contrôleur (signaux *mem* sur la figure III.8). Le générateur assure les connexions entre les unités et le contrôleur mémoire lorsqu'une une instruction d'accès à la mémoire est nécessaire au sein d'une unité. Après génération de l'ensemble des fichiers VHDL, un fichier bitstream pour le FPGA est obtenu à l'aide des outils de synthèse et de placement/routage adaptés à la cible (ISE dans notre cas).

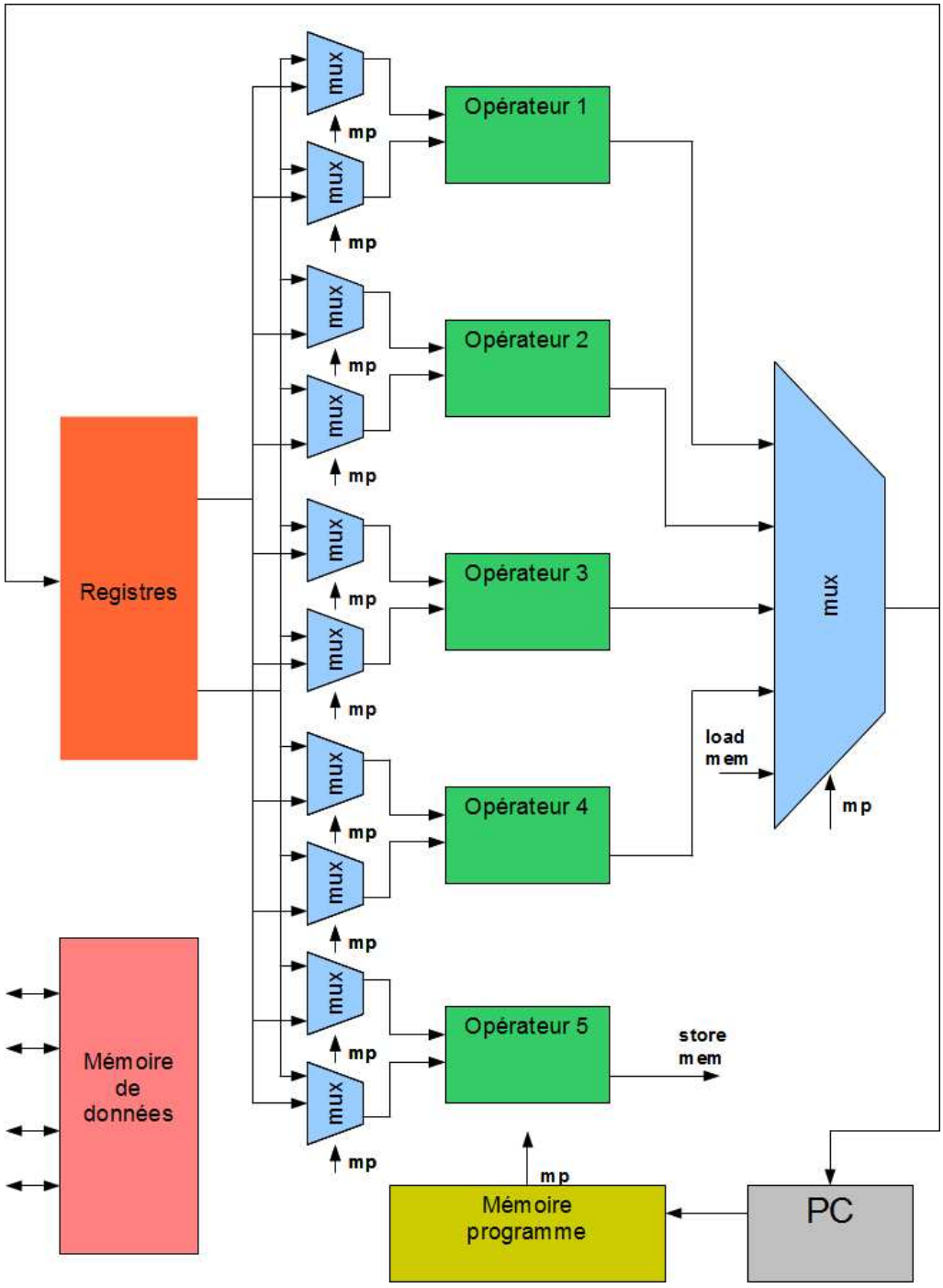


Figure III.8 : *Architecture simplifiée d'un processeur.*

III.6. Un exemple avec le filtre de Sobel

Nous allons, à l'aide d'un exemple simple, décrire l'enchaînement des différentes étapes du flot de conception permettant l'implantation directe d'un algorithme à partir de sa description en langage informatique. Il s'agit du filtre de Sobel qui est couramment utilisé pour effectuer la détection de

contours dans une image. Il constitue, en général, la première étape d'une chaîne de traitement. Ce filtre est implémenté comme une convolution de l'image avec deux masques de taille 3x3.

III.6.1 – Programme source en C et compilation

La première étape de notre flot de conception consiste à développer dans un langage informatique l'algorithme considéré. Ce programme sera écrit en langage C. Après la compilation, nous obtenons un code intermédiaire Lcode, suite de lignes d'assembleur générique. On trouvera ci-dessous un extrait de ce code.

```
   ....
R1=Load(R1)
R4=Load(R5)
R7=Load(R7)
R5=SHL(R4,R3)
R15=SHL(R15,R3)
R4=Add(R5,R1)
R5=Add(R4,R7)
R1=Sub(R13,R5)
R4=Add(R1,R17)
R5=Add(R4,R15)
R11=Sub(R11,R9)
    ....
```

III.6.2 – Extraction du parallélisme du code intermédiaire

Le premier outil effectue l'extraction ILP sous contrainte de dépendance de donnée et avec des accès mémoire simultanés limité à deux. La figure III.9 montre le résultat d'extraction sous forme de graphe pour cette application du filtrage de Sobel. Nous avons réussi à obtenir un taux de parallélisation de **Tp** = 2,7 (27/10).

III.6.3 – Analyse des ressources matérielles

Le deuxième outil analyse les ressources matérielles nécessaires à l'aide des différents modules et constitue une base de donnée matérielle. Nous présentons ci-dessous deux tableaux de synthèse résumant les différentes ressources matérielles nécessaire pour l'application Sobel.

Tableau III.2 : Ressources globales nécessaires pour le filtre de Sobel.

Nombre de registres	22
Nombre de registres après réduction	7
Nombre d'opérateurs	10

Tableau III.3 : Synthèse des opérateurs nécessaires.

Opérateur	Description	Nombre
load	Lecture mémoire	2
sub	Soustraction	2
add	Addition	2
lsl	Décalage à gauche	1
abs	Valeur absolue	2
store	Ecriture mémoire	1

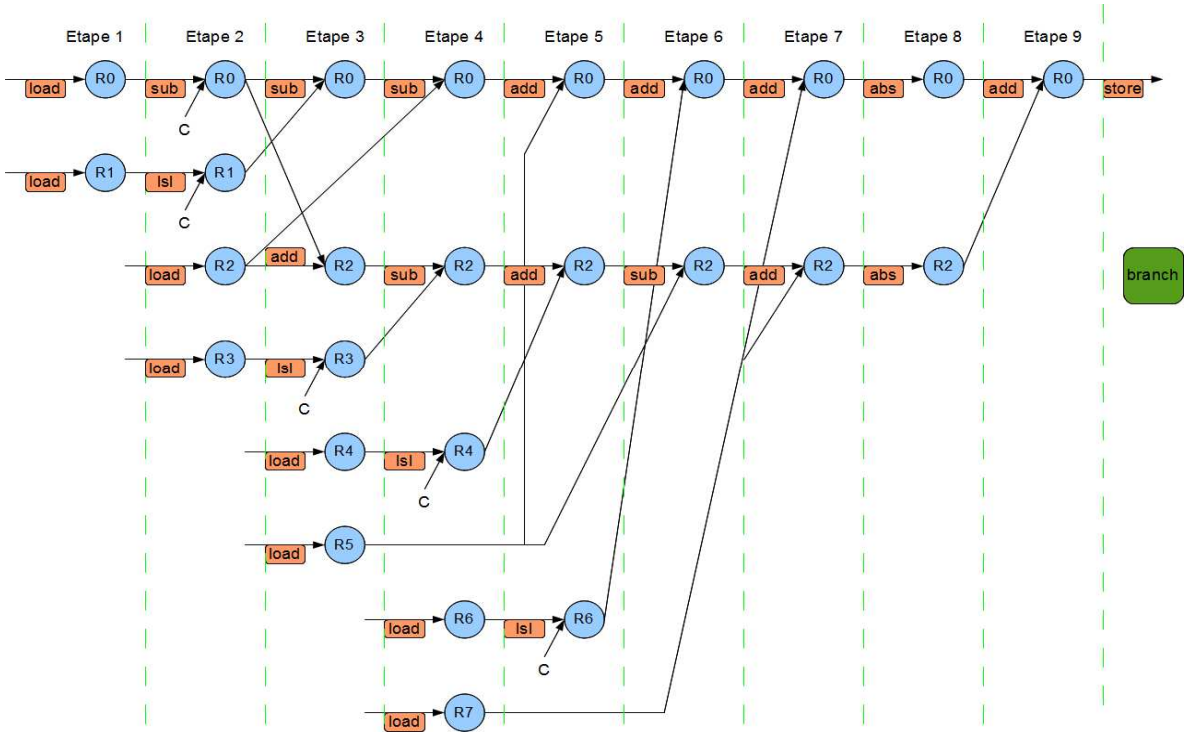


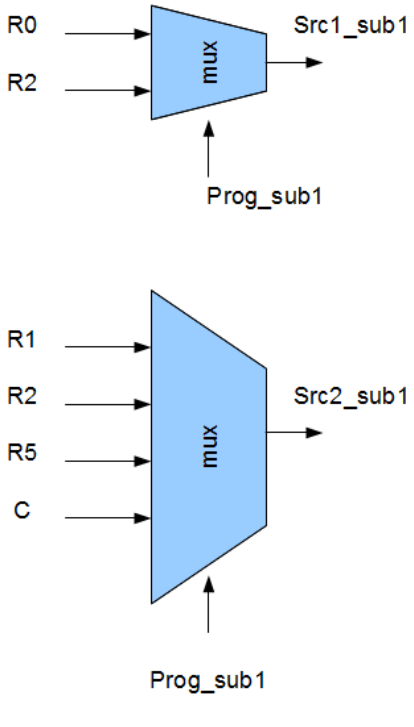
Figure III.9 : *Graphe des opérations généré par l'outil d'extraction ILP pour l'algorithme de Sobel.*

III.6.4 – Génération des codes VHDL

Nous allons décrire l'implantation matérielle des principaux blocs fonctionnels. La description de chaque bloc comporte un diagramme permettant de recenser les différents signaux d'entrée et de sortie ainsi qu'un exemple de code VHDL correspondant.

Génération des chemins de données en entrée

Ce sont des multiplexeurs qui établissent les chemins de données entre les registres et les différents opérateurs. Il existe deux multiplexeurs par opérateur. La figure III.10 montre un exemple pour le premier opérateur de soustraction.




```
Multiplexeur  src1 pour opérateur
soustraction

with prog_sub1(7 downto 0) select
        src1_sub1<=
                R0 when X"00",
                R2 when X"02",
                X"00000000" when others;
```

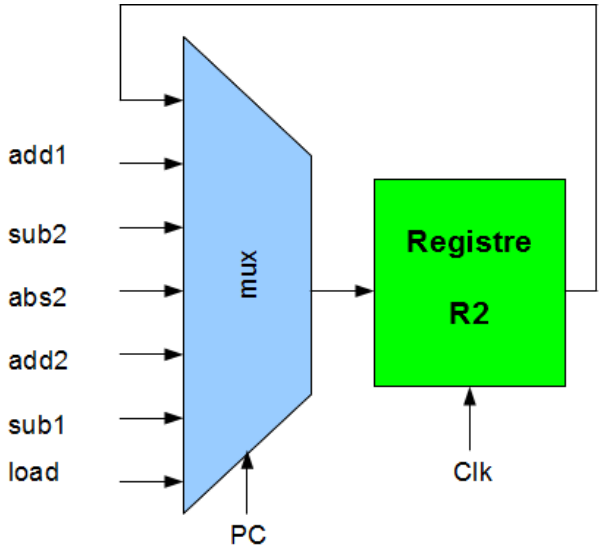
```
Multiplexeur  src2 pour opérateur
soustraction


with prog_sub1(15 downto 8) select
        src2_sub1 <=
                R1 when X"00",
                R2 when X"02",
                R5 when X"05",
                C  when X"FF",
                X"00000000" when others;
```

Figure III.10 : Génération des chemins de données pour l'opérateur de soustraction.

Génération des chemins de données en sortie

Ce sont des multiplexeurs qui permettent d'établir les connections entre les opérateurs de calcul et les registres. Il existe un multiplexeur par registre utilisé. Nous donnons l'exemple ci-dessous pour le registre R2.




```
with PC_R2 select

R2_tmp <=
        add1_out  when X"04",
        sub2_out  when X"03",
        abs2_out  when X"08",
        add2_out  when X"05",
        sub1_out  when X"02",
        load_out  when X"00",
        R2 when others;


process(Clk)
begin
If rising_edge(Clk) then
        R2 <= R2_tmp;
end if;
end process;
```

Figure III.11 : *Génération des chemins de données en sortie pour le registre R2.*

Génération du modèle VHDL

A partir de la base de donnée matérielle, un fichier VHDL est généré pour chaque opérateur présent. Ensuite, les différents fichiers VHDL sont regroupés pour synthétiser l'architecture complète du processeur pour l'application Sobel avec les ressources matérielles justes nécessaires. Le schéma complet de l'architecture générée est donné dans la figure III.12.

```
---  Code VHDL des opérateurs

sub1_out <= Src1_sub1 - Src2_sub1;
sub2_out <= Src1_sub2 - Src2_sub2;
add1_out <= Src1_add1 + Src2_add1;
add2_out <= Src1_add2 +Src2_add2;
lsl_out <= Src1_lsl lsl Src2_lsl;
abs1_out <= Src_abs1 when Src_abs1 > 0 else  -Src_abs1;
abs2_out <= Src_abs2 when Src_abs2 > 0 else  -Src_abs2;
load1_out <= load (Src_load1,@_load1);
load2_out <= load (Src_load2,@_load2);
store_out <= store (Src_store,@_store);
```

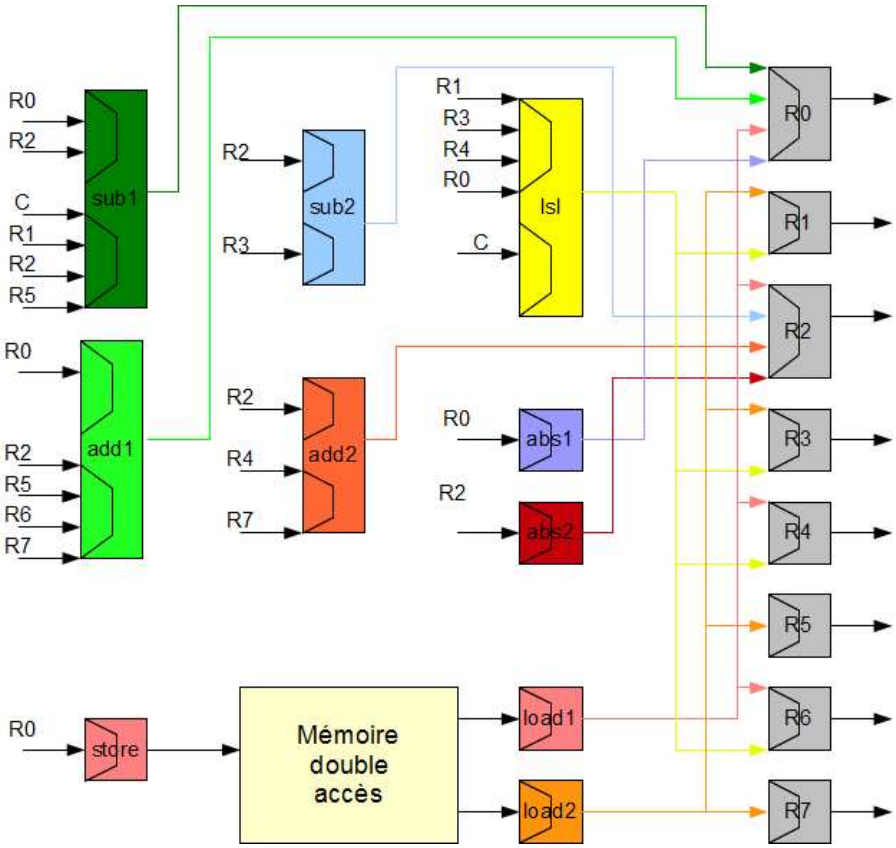


Figure III.12 : *Architecture du processeur pour l'algorithme de Sobel.*

III.6.5 – Synthèse matérielle sous ISE

Nous avons utilisé l'outil ISE de la société Xilinx pour synthétiser le code VHDL sur une cible FPGA Virtex 6 (voir le chapitre 5 pour une description détaillée). Au total, 1121 slices et 57 blocRAM sont nécessaires pour cette application. Nous avons atteint une vitesse de traitement de 56,6 images/seconde pour une taille d'image de 512x512 pixels. La fréquence maximum d'utilisation obtenue lors de la synthèse est de 252 MHz.

III.7. Quelques résultats de synthèse avec des algorithmes courants de traitement d'image

Nous avons appliqué notre flot de conception à quelques algorithmes de traitement d'images. Après compilation des sources C, extraction du parallélisme ILP, analyse des ressources matérielles et génération des modules VHDL, nous avons effectué la synthèse sur une cible FPGA Virtex 6 de chez Xilinx. On trouvera dans le tableau III.4 les résultats concernant l'implantation sur le FPGA.

Tableau III.4 : Résultats obtenus pour un ensemble d'algorithmes de traitement d'images.

Algorithme	Sobel	Conv. 3x3	Conv. 5x5	FDCT 8x8	SAD 8x8
Taux de parallelisation Tp	2,7	4,5	4,7	11,2	7,1
Nombre de slices	1121	1348	869	4665	3388
Nombre de DSP48	0	5	2	14	3
Nombre de BlocRAM	57	57	57	57	57
Frequence (MHz)	252	202	207	194	203
***Vitesse de traitement (*images/s)**	56,6	38,5	13,2	221,7	454,5

Nous pouvons noter que une accélération moyenne de 6**X** a été obtenue en phase d'extraction du parallélisme ILP. Un faible pourcentage des ressources matérielles a été utilisé pour réaliser ces algorithmes : 6,03% de slices, 0,63% de DSP48 et 13,7% de BlocRAM disponibles sur le FPGA Virtex 6. Ceci nous permettrait d'envisager d'autres traitements plus complexes.

III.8. Conclusion du chapitre

Dans ce chapitre, nous avons décrit un flot de conception qui permet d'implanter une application sur FPGA en utilisant un modèle VHDL de processeur RISP. L'originalité de notre approche, par rapport à des modèles de processeurs VHDL existants comme Microblaze ou Leon Sparc, consiste à dimensionner le modèle du processeur en fonction de l'application. Une analyse automatique des ressources utilisées par le processeur permet de n'implanter dans le modèle que le matériel juste nécessaire pour une application donnée. On peut mentionner deux avantages principaux de la méthode proposée :

- un non expert en électronique peut développer une application sur FPGA en utilisant un langage de programmation évolué. La connaissance d'un langage de description matériel n'est pas indispensable car la génération d'un modèle VHDL est effectuée automatiquement,

- après la mise au point du modèle du processeur réalisé une fois pour toute, le modèle VHDL partiel du processeur généré sera toujours synthétisable pour n'importe quelle application.

La méthode proposée, en gardant la souplesse de la programmation d'une part et la possibilité d'utiliser des bibliothèques existantes dans le domaine du traitement d'image d'autre part, permet d'atteindre l'objectif fixé concernant le prototypage rapide. L'inconvénient majeur de la méthode est bien sur l'impossibilité de rivaliser, pour un algorithme donné, en terme de temps de traitement ou de surface occupée sur le FPGA avec des modules développés spécifiquement en langage matériel (IP).

Pour utiliser ce flot de conception, il est nécessaire d'évaluer le compromis entre le gain réalisé en temps de développement d'une application et les performances attendues. Dans la suite de ce manuscrit, nous allons utiliser la méthode proposée pour une application complète de biométrie : l'extraction et la reconnaissance de paumes de la main. Pour ce faire, nous allons d'abord décrire la chaîne de traitement dans le chapitre suivant.

Chapitre IV

Présentation d'un système biométrique : reconnaissance de paumes sans contact

---

La modalité biométrique de paumes, traditionnellement captée sous contraintes et avec contact, fournit de riches informations permettrant d'identifier efficacement des sujets humains. Dans ce chapitre, nous décrivons un système biométrique adapté aux applications grand public. Il possède de multiples avantages : bas coût, plus d'hygiène, non invasive, dû aux caractères sans contact, facilement réalisable sur des architectures embarquées.

Nous commençons par l'acquisition des données sans contact : le dispositif nécessaire, le protocole utilisé ainsi que le logiciel associé. Après avoir obtenu les images de main, il faut passer par une étape d'extraction automatique de paumes avant de les envoyer aux modules de reconnaissance. Le classifieur situé à la sortie de la chaîne de traitement prendra une décision soit sur l'identité de la personne en cas d'identification, soit sur l'authenticité de la personne pour des applications de vérification. Ce chapitre est largement inspiré du manuscrit rédigé par Mlle. Audrey Poinsot : thèse soutenue en janvier 2011 sous la direction de Madame Fan Yang.

IV.1. Problématique de la biométrie

Les sociétés modernes s'appuient de plus en plus sur les technologies numériques, et elles ont de ce fait de plus en plus besoin de gérer automatiquement l'identité des individus. C'est pourquoi le marché de la biométrie est très florissant depuis une vingtaine d'années : il existe aujourd'hui un nombre important de solutions biométriques industrielles et académiques, aux propriétés variées.

IV.1.1 – Termes couramment utilisées

La *biométrie* désigne l'ensemble des techniques de *vérification d'identité* et d'*identification* des personnes, qui sont basées sur la mesure de caractéristiques individuelles physiques ou comportementales. Le terme "biométrie" est apparu au XVIIIe siècle, simultanément avec le terme "anthropométrie" auquel il est très lié. L'intérêt de l'époque pour l'utilisation de l'anthropométrie à des fins judiciaires est à l'origine de la biométrie moderne. Mais cette dernière est véritablement née dans les années 1960-1970 avec l'arrivée de l'informatique, qui a rendu possible l'automatisation des tâches.

Aujourd'hui, la biométrie n'est plus restreinte à une utilisation policière, car les besoins en sécurité de nos sociétés modernes s'intensifient. Notre identité est vérifiée à de nombreuses occasions : par exemple, lors d'un paiement distant, ou encore lors de l'accès à une zone restreinte. Les méthodes traditionnelles d'identification utilisent des objets (badges, clefs, etc.), et/ou des connaissances (mot de passe, numéro d'identification personnel ou PIN, etc.). Mais ce type de représentation de l'identité a le désavantage de pouvoir être perdu ou volé. La biométrie utilise des caractéristiques inhérentes à l'individu, ce qui rend l'oubli impossible et complique (voire interdit) le vol. Son utilisation offre donc une alternative naturelle et sûre pour qualifier l'identité.

Comme le suggère sa définition, la biométrie, comme tout système de gestion automatique de l'identité, peut avoir deux finalités pratiques : si le but est de certifier une identité revendiquée par l'utilisateur, on parle de *vérification d'identité* ou d'*authentification*, tandis que si l'on cherche à déterminer si l'individu fait partie de la base de données du système, et que l'on est capable, le cas échéant, de fournir son identité, on parle d'*identification*. Les systèmes biométriques peuvent avoir une dernière utilisation plus théorique : la *reconnaissance 1 parmi N*. Celle-ci repose sur l'hypothèse que tous les individus susceptibles de se présenter devant le système sont enregistrés dans la base de données. Sa finalité est donc de déterminer lequel des utilisateurs correspond le mieux à la personne à reconnaître. Un système d'identification ajoute la possibilité de vérifier si l'utilisateur appartient réellement à la base.

En pratique, l'identification comporte une étape de reconnaissance 1 parmi N, puis une étape de vérification de l'identité choisie. C'est pourquoi nous nous intéresserons particulièrement à ces deux dernières notions, qui nous permettront de qualifier les performances théoriques des systèmes biométriques. Nous utiliserons enfin dans ce mémoire la notion de reconnaissance au sens large lorsque nous ne voudrons pas faire de distinction entre authentification et identification.

Dans tous les cas, l'utilisation d'un système biométrique est précédée d'une phase d'enregistrement (ou enrôlement) au cours de laquelle l'information biométrique est capturée, traitée, et stockée pour une comparaison ultérieure. Lors de la vérification d'identité, le stockage est souvent réalisé sur une carte ou un badge que l'utilisateur présente à chaque authentification. Mais il est également possible que l'information biométrique soit stockée dans une base de données. L'authentification requiert alors un code ou un mot de passe, qui représentera l'identité clamée par l'utilisateur, et qui sera associé à l'information biométrique de l'individu dans la base. Pour ce qui est de l'identification, nous avons vu que son fonctionnement impose l'existence d'une base de données accessible par le système.

En France, l'installation de systèmes biométriques est contrôlée par la Commission Nationale Informatique et Liberté (CNIL) : aucun équipement ne peut être mis en place sans son autorisation préalable. La CNIL privilégie les systèmes dont le stockage des données est réalisé sur des supports individualisés, notamment pour les caractéristiques pouvant laisser des traces comme l'empreinte digitale. Seules les applications nécessitant un haut niveau de sécurité (par exemple l'accès à une centrale nucléaire) sont autorisées à utiliser des bases de données.

La biométrie s'introduit cependant de plus en plus dans la vie courante, et les solutions disponibles sont nombreuses. Afin de présenter les diverses applications de la biométrie, nous les avons classées en trois groupes :

- *Les applications commerciales* se rapportent à l'accès à des ressources protégées ou à des zones réglementées : coffre fort, pointeuse, distributeur automatique de billets, contrôle d'accès grand public (comme au parc d'attraction Walt Disney World en Floride), contrôle d'accès industriel (comme sur les sites classés Seveso II, présentant des risques d'accidents majeurs), etc.
- *Les applications gouvernementales* concernent tous les papiers d'identité délivrés par l'état : passeport, visa, carte d'électeur. Les techniques utilisées pour l'identification doivent prendre en compte le grand nombre de personnes enregistrées dans la base.
- *Les applications médico-légales* englobent toutes les techniques utilisées par la police scientifique et par les instituts médico-légaux (empreintes digitales, ADN, dentition, etc.), ou encore par les organismes de lutte contre le terrorisme (identification sans la coopération du sujet grâce au visage, à la démarche, etc.).

IV.1.2 – Modalités biométriques

Les caractéristiques utilisées pour la reconnaissance biométrique sont généralement classées en deux groupes. Certaines modalités sont en effet directement basées sur ce que l'on est, par l'enregistrement de traits physiques. Elles sont alors qualifiées de modalités morphologiques. D'autres sont plutôt basées sur ce que l'on fait ou ce que l'on produit. Dans ce cas, c'est une action de l'utilisateur qui est enregistrée, et ces dernières sont donc qualifiées de modalités comportementales.

Un grand nombre d'identificateurs biométriques ont été étudiés au cours de ces dernières décennies (voir Figure IV.1). Beaucoup d'entre eux sont d'ores et déjà utilisés dans des systèmes commerciaux. L'empreinte digitale, le visage, l'iris, la forme de la main, l'empreinte palmaire, la rétine, l'oreille, ou encore le réseau veineux de la main et du doigt, sont des exemples de biométries morphologiques ; la signature, la voix, la démarche, la dynamique de frappe au clavier, ou même le sourire, sont quant à eux des exemples de modalités comportementales. Nous ajouterons à cela les modalités physiologiques comme l'ADN, les battements cardiaques, la conductivité de la peau, ou les ondes cérébrales. Chaque modalité possède des avantages et des inconvénients. En générale, le choix d'une modalité particulière est guidé par le contexte de l'application et de niveau de sécurité demandé.

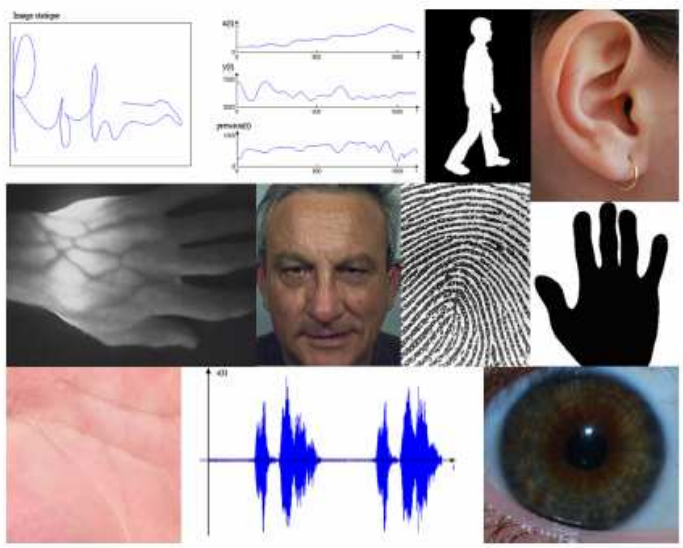


Figure IV.1 : *Exemples de caractéristiques biométriques : signature dynamique (image statique, coordonnées, et pression exercée sur le stylo), démarche, oreille, veines de la main, visage, empreinte digitale, géométrie de la main, paume, voix et iris* (*cf.* [Poin11]).

A. Poinsot [Poin11] a effectué une étude comparative sur les spécificités de chaque modalité à partir des articles scientifiques [Jain04] [IBG02] [Kapc06]. Le Tableau IV.1 permet de réaliser un premier classement des biométries. Mais il montre aussi que les experts ne sont pas totalement unanimes. De

plus, certains des critères choisis dépendent de l'état de la technique au moment de l'étude, ce qui implique que les résultats présentés sont susceptibles de changer au cours du temps. En outre, le choix de la caractéristique la plus adaptée à un contexte donné pour une application réelle nécessite des critères plus précis que ceux qui sont proposés, comme l'influence de l'environnement, la facilité d'intégration, les besoins de maintenance, la taille des capteurs, etc. Malgré tout, nous pouvons constater que la paume de la main est une modalité bien classée en fonction de plusieurs critères importants : universalité, unicité, stabilité, mesurabilité, performance, acceptabilité et in-falsifiabilité.

Tableau IV.1 : Comparaison des modalités selon les trois études : le degré de réponse aux critères est exprimé sur 3 niveaux : fort (feu vert), moyen (feu orange), et faible (feu rouge) (*cf.* [Poin11]).

Étude	Jain *et al.* 2004 [5]							IBG 2002 [6]				Kapczyński 2006 [7]		
Critères	universalité	unicité	stabilité	mesurabilité	performance	acceptabilité	infalsifiabilité	unicité	effort	intrusivité	coût	unicité	effort	intrusivité
Empreinte digitale														
Main (Géométrie)														
Visage														
Iris														
Signature														
Voix														
Paume								-	-	-	-	-	-	-
Veine de la main								-	-	-	-	-	-	-
Oreille								-	-	-	-	-	-	-

IV.2. Acquisition d'images de paumes

Tout comme l'empreinte digitale, l'empreinte palmaire a d'abord été recueillie par apposition de la paume encrée sur du papier, la numérisation étant réalisée grâce à un scanneur [You02]. Influencée par cette première utilisation, elle a longtemps été considérée comme une biométrie "de contact", c'est-à-dire nécessitant un contact direct avec le capteur. Aujourd'hui, il existe une grande variété de systèmes d'acquisition qui travaillent directement sur la paume avec ou sans contact.

IV.2.1 – Acquisition avec contact

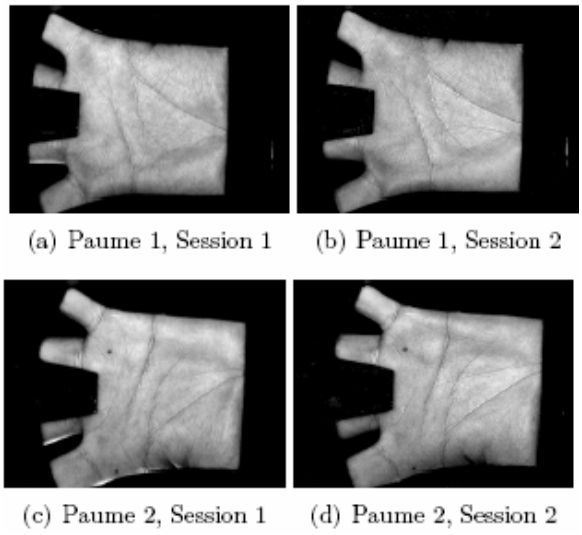
Certaines études utilisent le scanneur : la main, posée directement sur le plateau de l'appareil, peut alors être acquise en haute résolution avec une illumination constante et un fond uniforme [Conn05] [Lin05] [Savi07] [Kong08] [Kong09]. Ce type d'acquisition nécessite peu d'apprentissage de la part de l'utilisateur, car les doigts et la paume sont naturellement tendus par le contact avec la vitre. Cependant, la pression engendrée par ce contact peut faire apparaître des zones aplaties sur la paume. De plus, le temps nécessaire pour scanner la main n'est pas négligeable, ce qui est incommode pour l'utilisateur. Pour finir, la position de la paume dans l'image est variable, il est nécessaire de la localiser, et de la redresser avant de s'en servir pour la reconnaissance.

Le professeur Zhang et son équipe ont été les premiers à proposer un système dédié, combinant les techniques d'acquisition de la forme de la main et le principe du scanner : l'appareil est constitué d'un boîtier formant un environnement semi-clos, d'un plateau muni de chevilles, qui présentent une zone vitrée, et d'une caméra CCD autour de laquelle est placée une source de lumière annulaire (voir Figure IV.2). Contrairement au scanner, cet appareil permet de réaliser les acquisitions de manière instantanée. L'utilisation de chevilles facilite de plus la localisation de la main. Quelques exemples d'images acquises grâce à cet outil sont présentés Figure IV.3.



Figure IV.2 : *Exemples de systèmes d'acquisition de la paume (figures extraites des travaux de Kong et al. [Kong09] (gauche), et Kumar et al. [Kuma06] (droite)).*

Pour s'affranchir des problèmes liés au contact avec la vitre (zone de pression, nettoyage régulier, etc.), Kumar *et al.* proposent une autre procédure d'acquisition [Kuma06] : pour s'enregistrer, les sujets posent le dos de leur main sur un support noir, au dessus duquel se trouve un appareil photographique (Figure IV.2). Un tel système fonctionne presque sans contact mais garde l'avantage de la focale fixe : il bloque en effet la distance entre le capteur et la main, qui présente alors la même taille sur chaque image.



(a) Paume 1, Session 1 (b) Paume 1, Session 2

(c) Paume 2, Session 1 (d) Paume 2, Session 2

Figure IV.3 : *Exemples d'images issues de la base PolyU.*

Dans ce mémoire, la base publique utilisée pour tester les algorithmes de reconnaissance de paume est la base PolyU [Poly06] obtenue avec l'outil développé par le professeur Zhang et ses collègues, dont nous avons parlé précédemment. Comme nous pouvons remarquer sur les figures IV.2 et IV.3, le fond est noir, l'illumination constante, et la position de la main fortement contrainte par le contact avec le support et l'utilisation de chevilles de placement.

Cette base compte 193 sujets, qui ont chacun fournit environ 20 images de leur main droite et 20 images de leur main gauche. Les acquisitions ont été réalisées au cours de 2 sessions, espacées en moyenne de 2 mois. Il peut être noté qu'aucun accessoire n'est visible sur les acquisitions. Pour finir, nous soulignerons que les images ont été acquises à basse résolution et contiennent au final 384 × 284 pixels.

IV.2.2 – Acquisition sans contact

D'autres études, de plus en plus nombreuses, s'affranchissent totalement de l'obligation de contact [Li06] [Doub07] [Jiang07] [Goh08]. Les images de mains sont simplement acquises grâce à des appareils photographiques ou à des webcams dont la résolution peut être assez faible.

L'utilisation de telles images nécessite une phase d'extraction de la région d'intérêt. Pour cela, la principale difficulté consiste à localiser la main : cette étape peut être assez simple si le fond est uni (comme lorsqu'on utilise un scanneur), mais se complique lorsqu'il n'est pas contrôlé. Pour ce qui est de la normalisation de la fenêtre de paume, en plus des problèmes de rotation évoqués pour les systèmes à focale fixe, les possibles changements d'échelle doivent être pris en compte : la distance entre le capteur et la main n'étant plus fixée, la taille apparente de la main change d'une photographie à l'autre.

De plus, l'utilisation d'images de main acquises sans contact est rendue variable par l'illumination et la déformabilité de la caractéristique : si les doigts sont tendus et écartés, la zone palmaire est également tendue et bien visible, tandis que si les doigts sont courbés et/ou coller entre eux, l'apparition de plis et de creux modifient l'aspect de la paume. Les occlusions sont plus rares sur la paume elle-même, mais le port d'accessoires (bagues, montre, etc.) peut rendre la segmentation de la main et l'extraction de la zone d'intérêt plus difficiles.

Pour tous les systèmes, se pose le problème du choix de la main à acquérir. D'après [Duta08], certaines études font état de résultats légèrement supérieurs lorsqu'ils sont obtenus avec la main gauche. Ceci s'explique par le fait que la majorité de la population est droitière : la main gauche, qui est moins sollicitée, est donc moins déformable et plus stable dans le temps.

IV.2.3 – Protocole utilisé pour l'acquisition de la base uB

Durant sa thèse, Audrey Poinsot a construit une base de données contenant les images de main en utilisant un dispositif sans contact. Nous présentons dans cette section cette base de données uB (université de Bourgogne).

Le dispositif d'acquisition de la base uB se compose d'une table, d'un tabouret, d'une toile verte, de d'un webcam Logitech QuickCam Pro 9000 USB, et d'un ordinateur muni d'un écran, d'un clavier, et d'une souris. La webcam est fixée sur le bord gauche de la table et est dédiée à l'acquisition des images de mains gauches. Elle est utilisée à la résolution de 1600 × 1200 pixels.

La toile verte est placée dans le champ de la caméra dans le but de donner un fond vert aux images de mains. En effet, un fond uni facilite la segmentation de l'image : nous avons choisi un fond de couleur verte. Celle-ci est la plus lumineuse des couleurs de bases qui sont utilisées pour coder les images couleurs (Rouge, Vert, Bleu, dans la représentation RVB), et est de plus très peu présente dans la peau humaine

L'écran permet quant à lui de diffuser la prévisualisation de la webcam afin que les sujets puissent se placer correctement. Un logiciel d'acquisition nommé AcquiBase a été spécialement conçu pour que les participants puissent s'enregistrer eux-mêmes. Il a été développé en C++ sous l'environnement de travail Microsoft Visual Studio 2005, et se base principalement sur la bibliothèque *Video For Windows* (VFW).

L'interface graphique de ce logiciel est présentée en Figure IV.4. Il propose plusieurs fonctionnalités accessibles sous forme de boutons et d'items du menu. Il permet en particulier de recommencer l'acquisition si l'image est jugée inutilisable par l'utilisateur, ce qui se produit surtout lorsqu'elle est rendue floue par un mouvement inopportun de ce dernier.

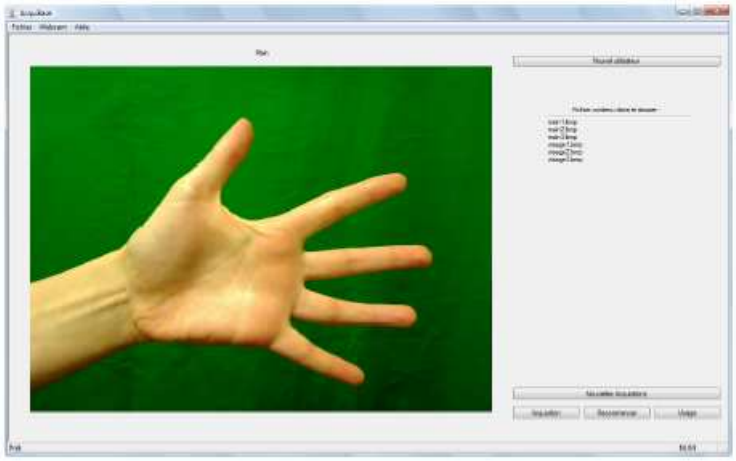


Figure IV.4 : *Interface graphique du logiciel AcquiBase.*

Le protocole d'acquisition est donc largement inspiré des protocoles existants, mais des efforts ont été faits pour le rendre réaliste dans un contexte grand public. Par exemple, entre chaque session, les conditions d'illumination, le positionnement face au matériel d'acquisition, et l'apparence, sont différents. De plus, il a été demandé aux sujets de ne pas rester statiques entre deux prises d'une même session : le simple fait de se relever et de retirer sa main introduit une certaine variabilité notamment sur la position. Ainsi, la stabilité au cours du temps du système testé peut être vérifiée, et les phases d'enrôlement et de reconnaissance peuvent être simulées en choisissant des exemples provenant de sessions différentes pour l'entraînement et le test.

IV.2.4 – Constitution de la base uB

La base uB contient 1170 images, qui ont été enregistrées en 3 sessions de 3 acquisitions sur 130 sujets, une acquisition correspondant à l'enregistrement d'une image de main. La durée minimale qui sépare deux sessions est d'un jour et peut aller jusqu'à un mois.

L'enregistrement de la base a été effectué sur deux sites : le premier est un bureau peu lumineux, principalement éclairé par des lampes halogènes situées au plafond, le second est une salle de travaux pratiques avec de larges fenêtres, éclairée par la lumière du jour. La proportion de la base acquise sur chacun des sites est respectivement d'un tiers/deux tiers. La Figure IV.5 propose quelques exemples d'images issues de la base uB.

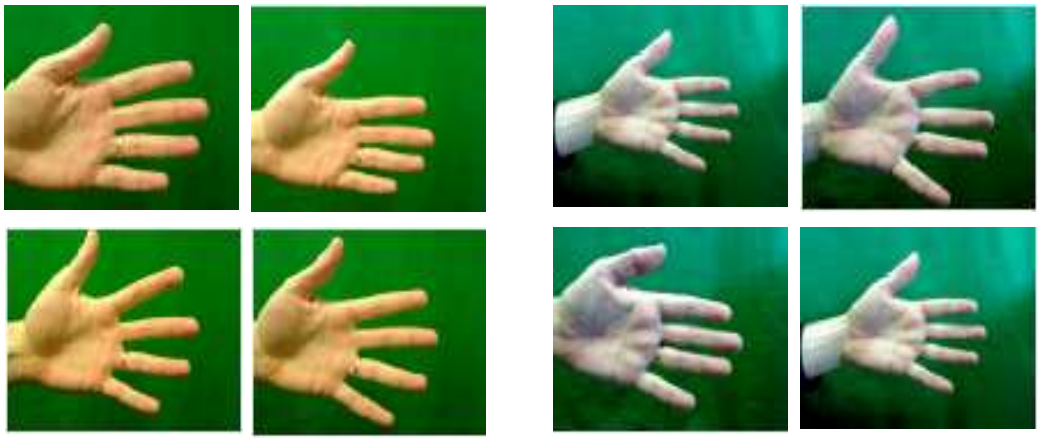


Figure IV.5 : *Exemples d'images issues de la base biométrique uB.*

Comme nous venons de le voir l'acquisition de la main est réalisée sur un fond vert. Le port d'accessoires, tels que les bracelets ou les bagues, est à l'appréciation du sujet. Aucune image ne présente d'occlusion de la région palmaire, mais sa périphérie est parfois masquée par un vêtement ou un pansement. Le placement est totalement libre : il a simplement été demandé aux sujets de mettre leur main gauche face à la caméra, de manière horizontale, et de façon à ce qu'elle soit entièrement visible sur l'écran. Il leur a également été demandé de veiller à ce que leurs doigts ne se touchent pas entre eux. La distance entre la main et la caméra est à leur convenance, tout comme sa position selon les axes horizontaux et verticaux à l'écran.

Selon les acquisitions, la taille de la main, ainsi que son orientation et sa position, peuvent varier significativement. Les images de paumes ne sont donc pas directement utilisables et elles nécessitent une étape de prétraitement correspondant à l'extraction de la région d'intérêt.

IV.3. Extraction automatique de paume de la base uB

Nous avons choisi, pour l'extraction des paumes, une approche classique en trois étapes. La première d'entre elles correspond à la segmentation de la main et est basée sur le seuillage de l'image ; la deuxième correspond à la localisation de points caractéristiques et utilise la courbe de contour de la main ; enfin, la troisième correspond à l'extraction de la région d'intérêt, et est inspirée de la méthode présentée en [Doub07].

Rappelons que les images de mains de la base uB ont été acquises sans contact devant un fond vert grâce à une webcam. La segmentation de la main que nous avons utilisée est donc tout naturellement basée sur la couleur. Nous avons en effet choisi d'effectuer un seuillage de seuil fixe sur la composante rouge de l'image (exprimée dans le système de couleur RVB), et cela car l'utilisation d'un fond vert nous garantit que les pixels les plus rouges appartiennent à la main (voir Figure IV.6). Le seuil a été fixé à 70. Il a été choisi d'après l'histogramme d'une unique image.

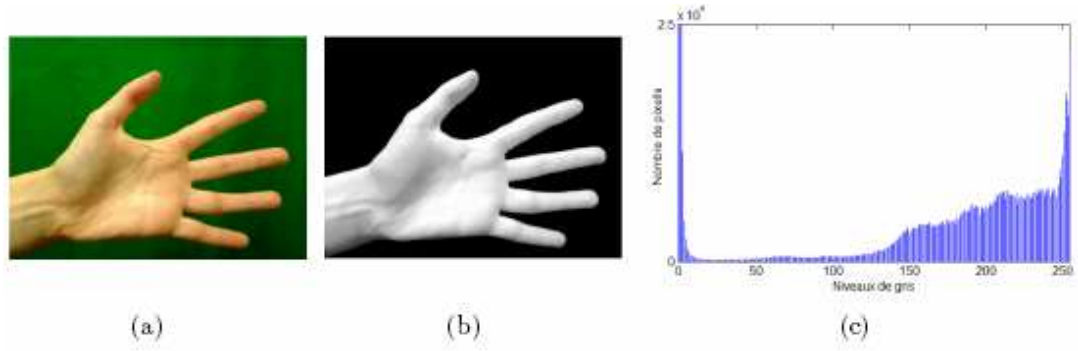


Figure IV.6 : *Propriété de la composante rouge des images de la base biométrique uB. (a) image d'origine, (b) composante rouge de l'image, (c) histogramme de la composante rouge.*

La Figure IV.6 (c) illustre ce choix en présentant l'histogramme d'une image type de la base uB. La séparation des niveaux de gris des pixels appartenant au fond et à la main y est bien visible : les pixels de fond forment en effet un pic fin et d'amplitude élevée sur les plus faibles niveaux de gris (il a été tronqué pour des raisons de visualisation, mais la plus haute barre atteint 1 millions des pixels), tandis que les pixels de main s'étendent sur une plus large plage de valeurs comprise entre 130 et 255.

Afin de nous assurer que chaque pixel blanc appartient à la main, nous effectuons sur les images segmentées un filtrage des îles 4-connexes. La main correspondant au plus grand objet de l'image, il suffit en effet d'éliminer toutes les composantes de taille moindre pour parfaire la segmentation. Cette étape est nécessaire pour préparer l'extraction de contour : elle permet d'une part de garantir que le contour suivi est bien celui de la main, puisque c'est le seul objet de l'image, et elle permet d'autre part

de simplifier le parcours du contour, car elle supprime toutes les composantes reliées à la main uniquement par un chemin 8-connexe, lesquelles correspondent à des irrégularités.

Pour calculer cette courbe de contour, nous utilisons un algorithme classique de suivi de contour basé sur le parcours du 8-voisinage des pixels dans un ordre défini par le dernier déplacement observé : *Code de Fremann*. Celui-ci est initialisé sous le premier pixel blanc rencontré lorsque l'image est lue colonne par colonne, afin de nous assurer que le contour débute au niveau du poignet. Le résultat de cet algorithme est illustré en Figure IV.7. Afin de faciliter l'implémentation des traitements d'extraction des points caractéristiques, le vecteur contenant les coordonnées des points de contour est ré-échantillonné à une taille fixe de 2048 points.

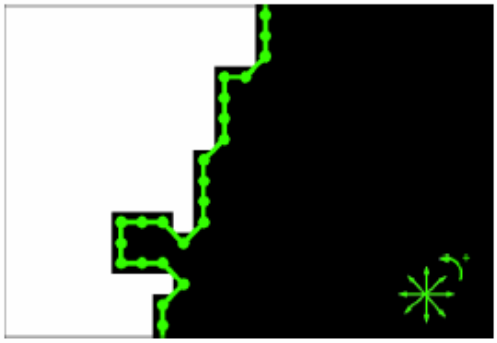


Figure IV.7 : *Extraction du contour de la main après seuillage (la main correspond à la forme blanche).*

Les points caractéristiques sont calculés en deux étapes : leur position est tout d'abord initialisée directement sur les coordonnées du contour, puis elle est réajustée par un algorithme itératif. En effet, l'une des consignes données aux sujets étant d'orienter leur main de telle sorte que leurs doigts se trouvent à la droite de l'image, les bouts des doigts et les vallées entres les doigts coïncident avec les maxima et les minima de l'abscisse des points du contour (Figure IV.8 (a)). De plus, le pouce se situant en haut de l'image, la position de son extrémité correspond au maximum des ordonnées des points du contour.

Pour calculer les extremas locaux de la courbe des abscisses, nous recherchons simplement les zéros de la dérivée. En outre, nous effectuons un test sur la distance entre les points trouvés pour supprimer les éventuels faux positifs. La distance minimale autorisée est calculée en fonction des plus grandes distances observée dans le signal considéré.

Mais les positions ainsi calculées peuvent facilement varier en fonction de l'orientation de la main, et sont de ce fait peu précises. C'est pourquoi un algorithme de réajustement a été mis au point (Algorithme 1). Celui-ci repose sur l'hypothèse qu'entre deux doigts, le point de contour le plus éloignés de chaque extrémité simultanément, correspond à la vallée. De même, le bout du doigt est le point de contour le plus éloigné des deux vallées qui l'entourent.

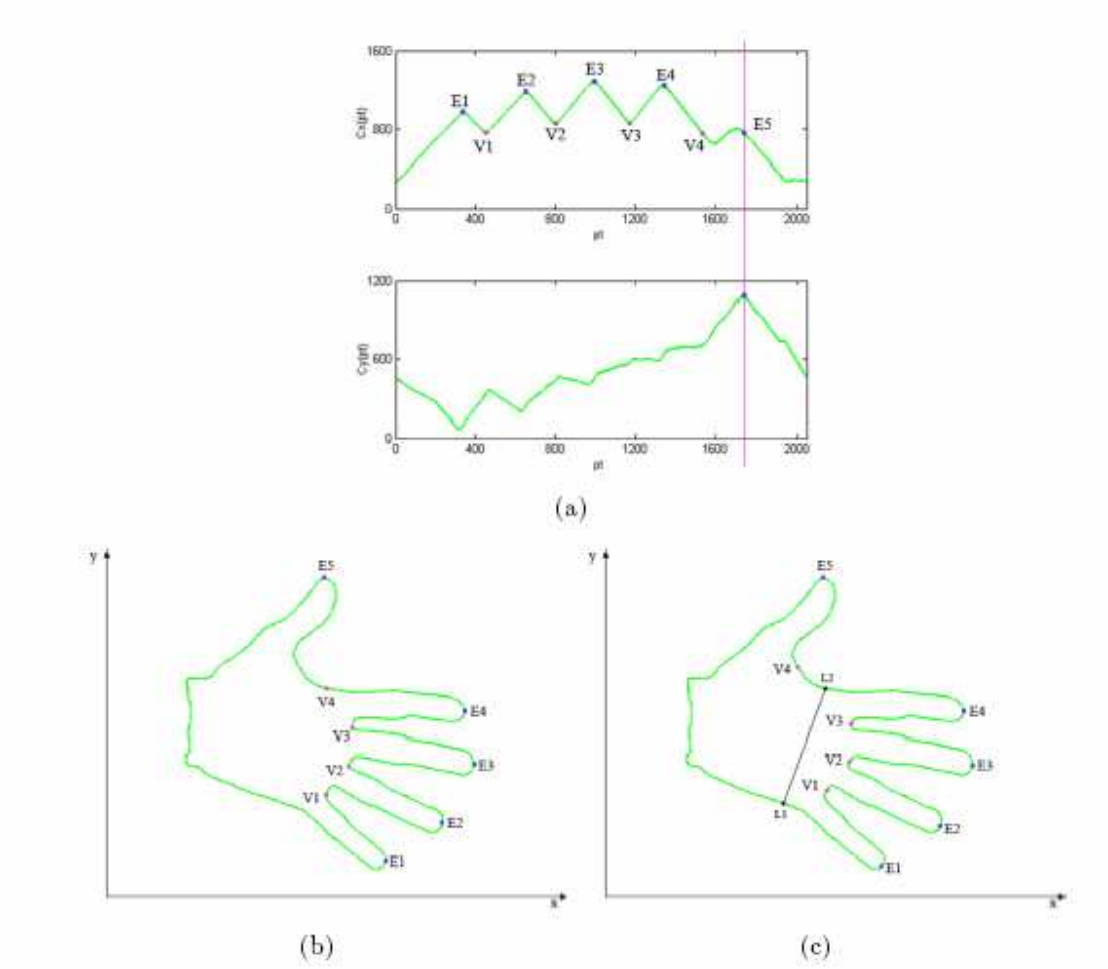


Figure IV.8 : *Extraction des points caractéristiques : (a) visualisation des coordonnées des points du contour, (b) initialisation des points caractéristiques d'après les coordonnées de contour, et (c) positions finales des points caractéristiques.*

Le pouce et l'auriculaire, qui n'ont qu'un voisin parmi les points caractéristiques sont des cas particuliers. La position du pouce n'est pas réajustée, car elle ne nous est pas utile dans la suite du traitement. La position de l'auriculaire, par contre, influe sur l'effet global de l'algorithme, car c'est la première à être réajustée. Elle est donc recalculée en se basant uniquement sur la distance entre son voisinage et la vallée auriculaire/annulaire.

L'effet de ce réajustement est visible en Figure IV.8. Comme nous pouvons le constater, les bouts des doigts sont initialisés aux points de plus grandes abscisses de chaque doigt, et les vallées aux points de

plus faibles abscisses situés entre les doigts. L'étape de réajustement permet de les replacer respectivement au milieu des bouts des doigts, et au milieu des creux des vallées.

Algorithme 1 Ajustement des points caractéristiques

Pour chaque point caractéristique P_i **faire**

1. Définir un voisinage V_{P_i} autour de P_i
2. Calculer la somme des distances euclidiennes (D) entre chaque point de V_{P_i} et les points caractéristiques P_{i-1} et P_{i+1} :

$$D(j) = l_2(V_{P_i}(j), P_{i-1}) + l_2(V_{P_i}(j), P_{i+1})$$

3. Redéfinir le point P_i : le point du voisinage correspondant au maximum de D remplace P_i.

Fin Pour

Deux autres points caractéristiques sont extraits du contour : il s'agit des points L_1 et L_2 permettant de calculer la largeur de la paume. Leur position est déduite de la position des vallées V_1 et V_4, et des extrémités E_1 et E_4. L_2 est en effet le point de contour situé aux $4/5^e$ de la courbe (E_4V_4), et L_1 est le point situé à droite de l'auriculaire (E_1) à une distance égale aux $6/5^e$ de (E_1V_1). La largeur de la paume est alors égale à la longueur du segment $[L_1L_2]$.

La fenêtre de paume est construite selon l'un des modèles classiques. La Figure IV.9 représente les six nouveaux points qui doivent être calculés pour la définir. Leur localisation dépend uniquement de la position des vallées V_1 et V_4, ainsi que de la largeur de la paume $||L_1L_2||$. En effet, les paramètres qui permettent de définir cette fenêtre ne sont pas constants, car l'acquisition est réalisée sans contact à une distance quelconque de l'objectif, ce qui implique que la main a une taille apparente variable. La largeur de la paume nous permet de normaliser le nouveau système de coordonnées : la distance $||O_1O_2||$ entre le segment de référence $[V_2V_4]$ et le carré de paume est fixé à $||L_1L_2||/10$. Le côté du carré $(||A_1A_2||)$ est quant à lui fixé à $2 * ||L_1L_2||/3$.

Il ne reste plus qu'une étape pour que la paume soit normalisée : sa rotation. L'angle de rotation a été calculé en fonction de la ligne tracée entre les deux points V_1 et V_3 (voir [Poin11]). Dans beaucoup d'études, la rotation a lieu avant la définition de la fenêtre, et cela afin de repérer plus facilement le carré. En effet, si la rotation est effectuée sur l'image entière, les bords de la fenêtre d'extraction sont horizontaux et verticaux, le carré est donc très facile à localiser. Cependant, le coût calculatoire de la

rotation est alors très élevé. C'est pourquoi nous avons choisi d'effectuer la rotation uniquement sur la plus petite fenêtre contenant les 4 coins du carré. Dans ce cas, le centre de rotation correspond à A_3.

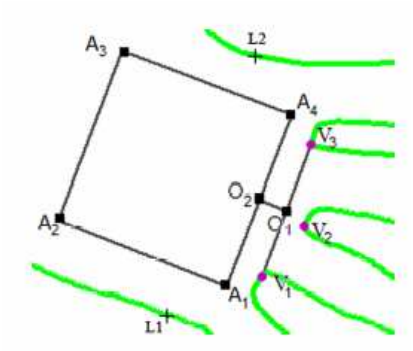


Figure IV.9 : *Définition de la fenêtre de paume.*

La méthode d'extraction de paume que nous venons de décrire a permis de traiter sans problème les 1170 images de mains de la base uB. La principale source de difficultés découle de la présence de bijoux, ou de vêtements. En effet, ils sont parfois mal segmentés ce qui altère le contour et lui ajoute des pics. Cependant, notre méthode de suppression des faux positifs lors de la détection des zéros de la dérivée suffit à corriger le problème. Malgré tout, il est difficile de juger la qualité de l'extraction : il n'est par exemple pas possible d'annoter les fichiers à la main, car le programme extrait une fenêtre particulière d'une zone de l'image. La comparaison des fenêtres entre elles demande la définition d'une métrique spécifique, ce qui correspond à un autre module de notre système de reconnaissance biométrique, qui sera présenté à la section suivante.

IV.4. Chaîne de reconnaissance biométrique

IV.4.1 – Différentes modules de la chaîne de reconnaissance

La reconnaissance biométrique est un problème de classification. Les systèmes biométriques sont donc avant tout des systèmes de reconnaissance de formes. De ce fait, ils peuvent être découpés en quatre modules (comme présenté en Figure IV.10), qui sont détaillés ci-dessous.

1. *Le module d'acquisition* : c'est dans ce module que la caractéristique est numérisée. Le résultat de l'acquisition correspond souvent à une image 2D (à 2 dimensions) : c'est le cas pour l'empreinte digitale, l'iris, la géométrie de la main, et bien d'autres biométries. Mais il peut aussi s'agir d'une scanographie 3D (visage, oreille), d'un signal 1D (voix, ECG), ou encore d'un enregistrement vidéo (démarche, mouvement des lèvres). La capture de la caractéristique est l'une des étapes clefs du traitement, car la qualité de l'acquisition influe largement sur les performances du système. C'est pourquoi ce module peut inclure un léger prétraitement de type correction d'illumination, filtrage, etc.

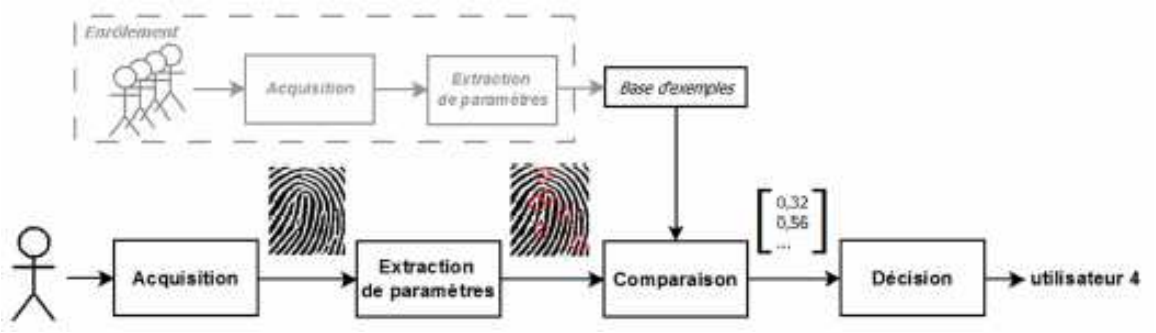


Figure IV.10 : *Les différentes modules de la chaîne de reconnaissance biométrique : exemple d'un système d'identification par l'empreinte digitale.*

2. *Le module d'extraction des paramètres* : ce module permet d'extraire l'information significative par un traitement adapté. Cette opération permet d'obtenir une représentation compacte de la caractéristique. C'est lors de cette étape que l'on calcule par exemple l'emplacement des minuties de l'empreinte digitale.

3. *Le module de comparaison* : au cours de l'enrôlement, chaque utilisateur fournit quelques échantillons du (ou des) trait(s) biométrique(s), dont les paramètres sont immédiatement extraits et stockés dans une base de données que l'on appelle la base d'exemples. Ce module permet de comparer les paramètres extraits de la caractéristique fournie par l'utilisateur avec ceux de la base d'exemples, afin de calculer des scores de similarité. Si l'on est dans une optique d'identification, la base d'exemples contient les paramètres de plusieurs utilisateurs ; s'il s'agit d'une authentification, celle-ci contient uniquement les paramètres de l'utilisateur courant.

4. *Le module de décision* : la décision de la classification est prise dans ce module à partir des scores de similarité. Pour les systèmes d'authentification, la décision est de type OUI/NON. Les systèmes de reconnaissance 1 parmi N, donnent l'identité de la personne inclue dans la base de données qui ressemblent le plus à l'utilisateur.

IV.4.2 – Etat de l'art sur la paume : une biométrie récente

La main est particulièrement sollicitée en biométrie : alors que l'empreinte digitale fait partie des caractéristiques les plus utilisées, de nombreux systèmes basés sur la géométrie de la main, et depuis peu sur les réseaux veineux, sont également commercialisés. Pourtant, la paume est une biométrie relativement récente. La recherche dans le domaine s'est d'abord fixée sur la reconnaissance d'images de paumes haute résolution, mais la majorité des travaux porte aujourd'hui sur les images basse résolution, car elles sont plus faciles à acquérir pour des applications civiles et commerciales.

En effet, la surface de la paume laisse apparaître trois niveaux d'information : (a) les lignes de la main, qui ont été recensées et nommées par les chiromanciens (lignes de coeur, de tête, de vie, etc.), (b) les plis et les rides, qui correspondent aux lignes irrégulières plus fines que les lignes principales, et (c) les crêtes papillaires. L'acquisition de ces dernières permet d'utiliser des traitements comparables à ceux des empreintes digitales avec parfois de meilleurs résultats, car la zone considérée est plus étendue. Cependant elle n'est possible qu'en haute résolution après un lourd traitement d'élimination des rides et des plis. C'est pourquoi la plupart des études s'intéressent plutôt aux deux premiers niveaux d'information.

L'enchevêtrement des lignes et des plis de la main visibles à ces niveaux forme un dessin propre à chaque individu (même jumeau), qui est tout à fait exploitable pour la reconnaissance biométrique. Ce paragraphe est dédié à la description des méthodes utilisant cette propriété des images de paumes basse résolution.

En tant que biométrie récente, la paume est très influencée par les autres modalités biométriques. De nombreuses études proposent par exemple de tester leurs algorithmes indifféremment sur des bases de paumes et de visages [Wan09] [Yan08] [Zuo10] [Nann08] [Jing04] [Jing06]. Il s'agit en l'occurrence de méthodes holistiques basées sur la réduction d'espace et s'attachant souvent à résoudre le problème du petit jeu de données.

En effet, la grande majorité des études considèrent les images de paumes comme des images texturées, et utilisent donc des méthodes globales. Si nous voulions faire la différence entre les méthodes locales et globales comme pour le visage, nous ne pourrions d'ailleurs mettre dans la première catégorie que les solutions basées sur la comparaison des lignes principales. Ce sont en effet les seules méthodes qui utilisent des connaissances à priori sur la structure de la paume. Dans leur état de l'art sur les techniques de reconnaissance d'images de paumes encrées, You *et al.* [You02] décrivent une solution de ce type : celle-ci se base sur une méthode géométrique pour extraire les positions de plusieurs points particuliers permettant de qualifier la forme des lignes de la main.

La plupart des méthodes utilisées pour la reconnaissance de paumes sont donc holistiques. La paume est vue comme un enchevêtrement de lignes plus ou moins fines ayant chacune leur propre orientation. L'information majoritairement traitée se trouve donc dans l'orientation des contours de l'image. C'est pourquoi les transformations employées sont souvent de type temps-fréquence (filtres de Gabor, transformée de Fourier), multi-résolutions (transformée en ondelettes), ou approximation de gradient (filtres de Sobel).
Les méthodes globales de reconnaissance de paumes peuvent être classées en cinq groupes : les méthodes par codage élémentaire, les approches lignes, les méthodes statistiques, les méthodes de

réduction d'espace, et enfin les méthodes spécifiques. Dans la section suivante, nous allons détailler les méthodes de codages appliquées à notre base de données uB.

IV.4.3 – Chaîne de traitement appliquée à la base de données uB

Les méthodes de codage ont été introduites par John Daugman dans les années 1990 [Daug93]. Elles ont été conçues pour la reconnaissance d'iris et sont aujourd'hui particulièrement utilisées pour la reconnaissance de paumes : [Zhang03], [Kong02], [Kong04], etc. Ces méthodes sont basées sur les filtres de Gabor et elles possèdent des propriétés très intéressantes pour notre contexte d'étude. En effet, elles ont été définies par Daugman pour répondre aux contraintes liées à l'utilisation de grandes bases de données, leur temps d'exécution est donc très faible. De plus, les paramètres ont été choisis pour n'utiliser que peu de mémoire : Daugman propose pour l'iris un vecteur de paramètres de moins de 256 octets, afin qu'il puisse être stocké sur les bandes magnétiques à trois pistes situées à l'arrière des cartes bleues.

Les filtres de Gabor sont très employés en reconnaissance de formes en général. Ils proposent un compromis temps-fréquence optimal, qui permet de mettre en valeur l'information des textures orientées. La réponse impulsionnelle d'un filtre de Gabor est le produit d'une gaussienne et d'une fonction sinusoïdale. Il peut donc être exprimé grâce à l'expression suivante :

$$G(x,y) = \exp\left[-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right]\times\cos\left(2\pi\frac{x'}{\lambda}\right)$$

avec

$$\begin{cases} x' = (x-\xi)\cos(\theta) - (y-\nu)\sin(\theta) \\ y' = (x-\xi)\sin(\theta) + (y-\nu)\cos(\theta) \end{cases}$$

et

(ξ, ν) représentant le centre de la fonction,

θ : orientation

λ : période de l'onde

σ : dérivation standard

γ : ratio de l'aspect visuel



Figure IV.11 : *Réponse impulsionnelles d'un filtre de Gabor 2D pour différentes orientation (*$\theta = \pi/8 \times \{0, 1, 2, 3, 4, 5, 6, 7\}$).

Cette formulation met en valeur la capacité de ces filtres à extraire de l'image les motifs d'une orientation et d'une fréquence données. La Figure IV. 11 illustre cette propriété en donnant l'allure d'un filtre de Gabor pour plusieurs orientations.

La première méthode de codage utilisée pour la reconnaissance de la paume a été proposée par Zhang *et al.* [Zhang03] [Kong02]. La caractéristique extraite grâce à cette méthode a simplement été appelée "Palmcode". Celle-ci est obtenue en seuillant l'image de paume après l'avoir convoluée avec un filtre de Gabor dont les paramètres ont été fixés à priori. Le seuil choisi pour binariser l'image est fixé à 0.

La comparaison entre les codes de paumes est réalisée grâce à la distance de Hamming normalisée. Celle-ci correspond au pourcentage de pixels différents dans les codes testés. Ils ajoutent également une certaine élasticité à cette distance pour pallier les défauts d'extraction. Une tolérance en translation est en effet introduite par le calcul des 25 distances entre l'image de test et l'image de référence décalée de -2 à 2 pixels horizontalement et verticalement. La distance finale correspond alors au minimum des scores ainsi obtenus.

Le module de comparaison fournit la mesure de similarité et le score final nous permet de prendre une décision de reconnaissance. Dans la section suivante, nous présentons les performances obtenues en appliquant la méthode de « Palmcode » à la base de données uB avec deux configurations d'application : identification et vérification.

IV.5. Résultats expérimentaux

IV.5.1 – Performances d'identification de paumes avec la base uB

L'identification biométrique de type 1 parmi N est une classification : on choisit parmi les N identités illustrées dans la base d'exemples, l'identité correspondant à l'échantillon testé. On ne peut donc se trouver que dans l'un des deux cas suivants :
– le système a choisi la bonne identité ;
– le système fournit une identité erronée.

Les systèmes d'identification vont donc être évalués grâce à leur taux de bonne reconnaissance (RR pour "Recognition Rate"). Celui-ci est obtenu après classification de tous les éléments de la base de test: il s'agit du rapport entre le nombre d'échantillons bien classés et le nombre total de tests.

La chaîne de traitement proposée a été conçue pour assurer un niveau de sécurité suffisant à une utilisation grand public. Ses performances ont été principalement testées sur la base uB. Rappelons qu'il

s'agit d'une base réelle de grande taille acquise sans contact pour s'adapter au contexte de l'étude. La base uB contient des images de mains recueillies sur 130 sujets. Chaque sujet a fournit 9 échantillons de main au cours de 3 sessions, à raison de 3 acquisitions par session. Le contexte de l'étude nous impose de travailler avec peu d'exemples par personne : les bases d'exemples utilisées pour le test du système contiennent donc seulement 1 à 3 échantillons par personne.

Nous avons défini deux protocoles de test différents :

- **protocole 1** : tous les échantillons qui n'ont pas été placés dans la base d'exemples sont testés ;
- **protocole 2** : seuls les échantillons ayant été enregistrés au cours d'une session différente de celle dont sont issus les échantillons de la base d'exemples sont testés.

Pour une base d'exemples donnée contenant *n* échantillons par personne (avec n entre 1 et 3), le taux de bonne reconnaissance est donc calculé grâce à $(9-n)\times 130$ tests pour le protocole 1, et $6\times 130 = 780$ tests pour le protocole 2. Le protocole 1 est le plus répandu dans la littérature, car il permet de prendre en considération toute la variabilité de la base de données. C'est le protocole que nous avons utilisé par défaut dans nos tests de reconnaissance 1 parmi N. Le protocole 2 est utilisé pour s'assurer de la robustesse du système aux changements temporels. Nous l'avons utilisé sur la base uB en complément du protocole 1, pour comparer les performances obtenues.

En identification comme en authentification, les résultats sont très dépendants de la base d'exemples. Afin de remédier à cela, nous avons effectué plusieurs tests, dont nous avons ensuite moyenné les résultats. Dans le cadre du protocole 1, le calcul d'un indicateur est réalisé à l'aide de 9 tests, et les échantillons de la base d'exemples sont choisis aléatoirement pour chaque test : dans le cas où cette base ne contient qu'un échantillon par personne, chacune des 9 paires d'échantillons de la base de données sont placées tour à tour dans la base d'exemples ; dans le cas où la base contient 2 échantillons par personne, 2 paires d'échantillons sont choisies aléatoirement parmi les 36 possibles. Dans le cadre du protocole 2, les échantillons de la base d'exemples sont pris dans la même session. Dans le cas où elle contient 3 échantillons par personne, seulement 3 tests sont donc possibles. Dans ce manuscrit, les taux d'erreur moyennés seront nommés **ARR** ("Averaged Recognition Rate").

Les résultats obtenus pour la paume sont présentés dans les Tableaux IV.2 et IV.3. Intuitivement, les filtres de Gabor dont l'orientation correspond à l'orientation globale des lignes de la paume devraient obtenir les meilleurs résultats. Conformément à la définition du filtre de Gabor que nous avons choisi et à l'allure d'une paume de main gauche, cette orientation devrait être obtenue pour $\theta = \pi/4$. Or le Tableau IV.2 confirme cette idée, puisque les filtres obliques et en particulier les filtres d'orientation $\theta = \pi/4$ obtiennent effectivement les performances les plus élevées.

Le choix du σ est plus empirique. Toutefois, la différence entre les ARR restant assez faible sur la gamme de σ choisie, l'influence de ce paramètre reste modérée. Le meilleur ARR est obtenu pour σ = 4.59. Le filtre de Gabor pour lequel $\theta = \pi/4$, σ = 4.59, $\gamma = 0.5$ et $\lambda = \sigma/0.56$ est donc choisi pour traiter les images de paumes dans notre système.

Tableau IV.2 : Choix des paramètres de Gabor pour la reconnaissance de paumes : ARR calculé sur la base uB dans le cas où la base d'exemples contient 1 échantillon par personne. (*cf.* [Poin11]).

ARR (%)	$\theta = \pi/4$	$\theta = \pi/4$	$\theta = \pi/4$	$\theta = \pi/4$
σ = 3.06	86.7	89.9	85.7	87.2
σ = 4.59	86.1	90.7	86.8	89.4
σ = 6.12	85.8	89.8	86.9	89.4

Tableau IV.3 : ARR du système proposé calculé selon les protocoles 1 et 2 sur la base uB avec une base d'exemples contenant 1, 2 et 3 échantillons par personne.

ARR (%)	1 exemple par sujet	2 exemples par sujet	3 exemples par sujet
Protocole 1	90.67	96.39	97.55
Protocole 2	88.63	93.36	94.87

Le Tableau IV.3 nous permet de visualiser l'effet de la taille de la base d'exemples sur les résultats. Il est évident que plus celle-ci contient d'échantillons plus les performances augmentent. Mais il est intéressant de noter qu'en passant d'un à deux échantillons par personne les résultats augmentent vraiment significativement : la différence entre les ARR est d'environ 6%. Le passage de 2 à 3 exemples par personne n'induit quant à lui qu'une faible augmentation de performance. Les expériences menées pour remplir la seconde ligne suivent le protocole 2. Elles recréent ainsi un enrôlement plus réaliste, puisque les échantillons de la base d'exemples sont enregistrés au cours d'une même session. Ce tableau nous permet aussi de mieux visualiser l'influence de la variabilité des échantillons de la base d'exemples sur les performances. Les taux de bonne reconnaissance présentés en seconde ligne sont globalement inférieurs à ceux da première ligne, mais restent assez élevés : le système obtient des ARR, qui se situent entre 88.63% dans le cas où la base d'exemples contient 1 échantillon par personne, et 94.87% dans le cas où elle en contient 3. En outre, ces résultats sont bien plus stables : les écarts types calculés sur les 9 tests moyennés sont effectivement plus faibles pour le protocole 2.

IV.5.2 – Performances de vérification de paumes avec la base uB

Le principe de l'authentification biométrique est de vérifier l'identité clamée par un utilisateur à l'aide d'un échantillon biométrique. Il s'agit d'une classification binaire : l'échantillon biométrique testé appartient en effet soit à l'utilisateur réel, soit à un imposteur. Ces systèmes sont donc sensibles à deux types d'erreurs :

- *la fausse acceptation*, qui a lieu lorsqu'un imposteur est reconnu comme étant l'utilisateur réel ;
- *le faux rejet*, qui a lieu quand l'utilisateur réel est considéré comme étant un imposteur.

Ces erreurs sont qualifiées par le taux de fausses acceptations (ou FAR pour "False Acceptance Rate"), et le taux de faux rejets (ou FRR pour "False Recognition Rate"), qui sont définis selon les expressions suivantes :

$$FAR = \text{nb fausses acceptations sur nb tests imposteurs}$$
$$FRR = \text{nb faux rejets sur nb tests utilisateurs réels}$$

Chacun de ces taux dépend du seuil de détection choisi dans le module de décision. Comme illustré Figure IV.12, les distributions des scores obtenus pour des utilisateurs véritables et des imposteurs présentent quasiment toujours une zone de recouvrement. Le choix du seuil de décision implique donc de faire un compromis entre le FRR et le FAR.

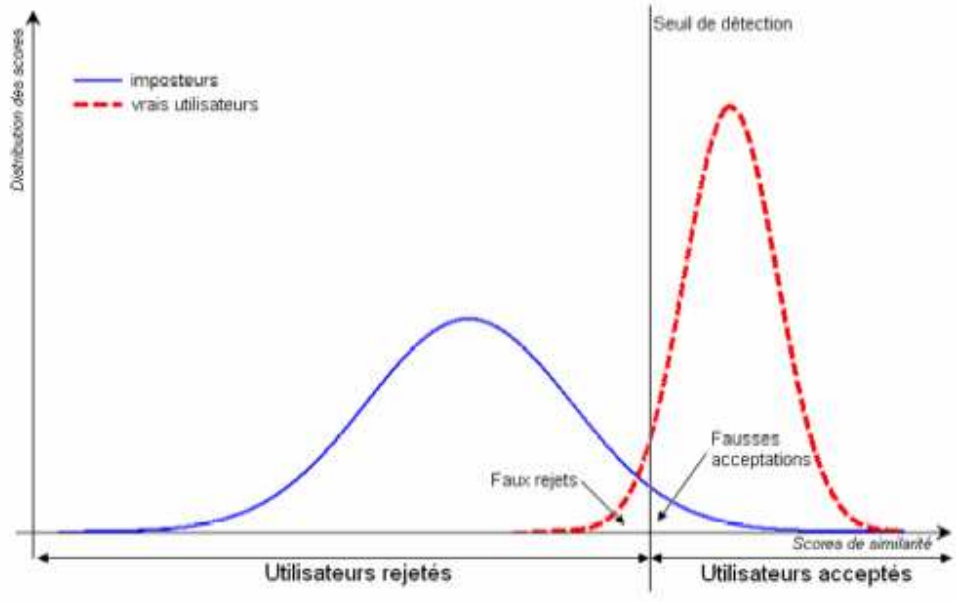


Figure IV.12 : *Courbes de distribution des scores « vrai utilisateurs » et « imposteurs ».*

Pour visualiser les performances indépendamment du seuil choisi, on utilise classiquement la courbe ROC (pour "Receiver Operating Curve"). Il s'agit d'une courbe représentant les variations du FAR en

fonction du FRR (voir Figure IV.13). Elle est calculée point par point pour différents seuils de détection à partir des distributions des scores "vrais utilisateurs" et "imposteurs". En effet, pour chaque seuil *se* :

$$ROC(se) = [FRR(se), FAR(se)]$$

Ces distributions sont elles-mêmes calculées grâce aux scores obtenus sur la base de test pour des échantillons enregistrés respectivement sur des personnes identiques et des personnes différentes.

Afin de comparer plus rapidement les systèmes d'authentification biométrique entre eux, une valeur numérique est souvent extraite de cette courbe. Il s'agit du taux d'égale erreur ou EER (pour "Equal Error Rate") : il correspond au point de fonctionnement pour lequel le FRR est égal au FAR.

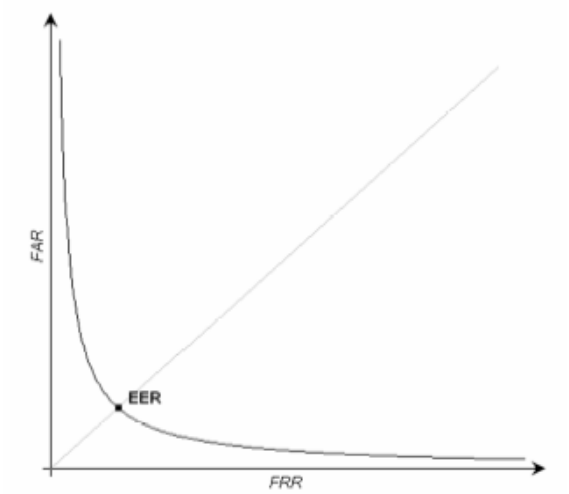


Figure IV.13 : *Courbes ROC, FAR, FRR et EER.*

Nous avons choisi de calculer le taux d'égale erreur de la base uB uniquement suivant le protocole 1. En effet, si l'on se place dans le cas où la base d'exemples contient 1 échantillon par personne, il est possible pour chaque utilisateur de réaliser 1161(= 129 × 9) tests "imposteur" et seulement 8 tests "vrai utilisateur". Face à ce déséquilibre, nous avons trouvé plus judicieux d'utiliser tous les échantillons à notre disposition afin de ne pas réduire encore le nombre de tests "vrai utilisateur" déjà faible. Comme dans le cas d'identification, nous utilisons les d'erreur moyennés AEER (Averaged Equal Error Rate) pour tester les performances de vérification.

Les résultats du système en terme d'authentification sont présentés au Tableau IV.4. On y retrouve les mêmes tendances qu'en reconnaissance 1 parmi N : la modalité de la paume obtient de très bon résultats ; les performances augmentent significativement lorsque l'on passe d'un à deux échantillons par personne dans la base d'exemples, et plus légèrement lorsque l'on passe de deux à trois.

Tableau IV.4 : AERR du système proposé calculés selon les protocoles 1 et 2 sur la base uB avec une base d'exemples contenant 1, 2 et 3 échantillons par personne.

AERR (%)	1 exemple par sujet	2 exemples par sujet	3 exemples par sujet
Protocole 1	5.89	2.44	1.56

IV.5.3 – Comparaison des performances avec la base publique PolyU

Les tests ont également été menés sur une base publique PolyU, et ceci afin de valider les résultats obtenus sur la base uB, mais aussi de comparer le système proposé aux systèmes existants [Jing07] [Yao07]. Il est à noter que la base PolyU a été construite avec de fortes contraintes (avec contact, éclairage constant, voir Section IV.2.1). Elle compte 119 sujets, pour lesquels ont été acquises 20 images de paumes au cours de deux sessions. Nous avons donc choisi le même protocole de test que [Jing07] [Yao07]. Il s'agit d'un protocole équivalent au protocole 1 utilisé sur la base uB : 1, 2, ou 3 échantillons par sujet sont sélectionnés aléatoirement dans la base de données pour former la base d'exemples, et le reste des échantillons constitue la base de test. Chaque ARR ou AEER est calculé grâce au moyennage des résultats de 20 tests réalisés avec des bases d'exemples différentes.

Le paramétrage des filtres de Gabor est le même que celui qui a été choisi pour les tests menés sur la base uB. Ainsi, le système générique utilisé pour les tests de la base uB est confronté à une nouvelle base, afin de vérifier sa robustesse aux données d'entrée. Les performances de reconnaissance sont présentées dans le Tableau IV.5. La comparaison effectuée nous permet de tirer plusieurs remarques.

Tableau IV.5 : Comparaison de méthodes de reconnaissance biométrique sur la base PolyU.

AERR (%)	1 exemple par sujet	2 exemples par sujet	3 exemples par sujet
Yao et al.[Yao07]	62.7	-	-
Jing et al.[Jing07]	-	63.3	64.3
Méthode proposée	89.8	96.0	97.2

Les résultats d'authentification du système sont eux aussi comparables à ceux de reconnaissance. Kumar et Zhang [Kuma09] obtiennent un EER de 0.75%, tandis que Ribari`c *et al.* [Riba08] obtiennent un EER de 2.29% avec respectivement 4 et 5 couples d'échantillons par personne dans la base

d’exemples. Nous restons donc dans les mêmes plages de valeurs avec un AEER de 1% obtenu avec 3 exemples par personnes dans la base d’exemples.

IV.6. Bilan du chapitre

Dans ce chapitre, nous avons présenté un système biométrique développé par Mlle. Audrey Poinsot au sein de notre équipe de recherche. Il présente de multiples avantages d’applications grand public : plus d’hygiène, souplesse d’utilisation, bas coût, non invasif.

Pour garder le caractère de « user-friendliness », nous avons utilisé une série de prétraitement pour extraire automatique ROI (paume) depuis une image de la main gauche. Ensuite le filtre Gabor permet de faire sortir les caractéristiques discriminantes de la paume avant la mesure de similarité. Un classifieur de type NN (*Near Neighbour*) basé sur la distance Hamming en associant une certaine élasticité nous indique la décision de reconnaissance.

Les performances obtenues avec la base uB et les comparaisons effectuées sur la base publique PolyU montre la grande robustesse de notre système, pour l’identification et aussi pour la vérification biométrique. Les performances d’un système utilisant deux échantillons par personne dans la base d’exemples sont particulièrement intéressantes.

Le coût calculatoire et l’utilisation mémoire de la chaîne proposée sont minimes. Il existe en outre de nombreuses possibilités de parallélisation du traitement dues à sa modularité, mais également à un autre niveau à l’utilisation d’images. Nous allons donc à présent discuter les implémentations matérielles de cette chaîne dans le chapitre suivant.

Chapitre V

Prototypage rapide d'un système biométrique sur le FPGA

Dans ce chapitre, nous allons présenter les expériences menées sur le prototypage rapide du système biométrique basé sur la reconnaissance de paumes décrit au chapitre précédent. Nous commençons par un état de l'art rapide sur les implémentations matérielles des systèmes biométriques. Ensuite, nous détaillons la plateforme utilisée ainsi que l'architecture commune choisie. Les expérimentations pour les deux phases de traitement : l'extraction automatique de paumes et la reconnaissance seront exposés dans la troisième partie de ce chapitre. Pour la première, le prototypage rapide a été réalisé en utilisant le flot de conception proposé. Pour la seconde phase, nous proposons une étude comparative pour illustrer l'avantage de notre approche.

V.1. Implémentations matérielles des systèmes biométriques

La plupart des systèmes de reconnaissance biométrique sont développés sur des ordinateurs classiques à l'aide de langages haut niveau (Matlab, Langage C, etc.) en vue d'une utilisation logicielle. Ce type de conception est totalement axé sur les performances de reconnaissance, et ne possède pas de réelle limitation sur l'utilisation mémoire ou encore sur le coût calculatoire des algorithmes. Pourtant, la biométrie est souvent intégrée à des systèmes autonomes dédiés tels que les téléphones portables, les systèmes d'ouverture de portes, ou encore les distributeurs automatiques de billets. Or pour cela, ils sont implantés sur du matériel dont les capacités de stockage sont limitées, et dont les fréquences de fonctionnement sont plus faibles que celle des ordinateurs d'usage personnel (PC). C'est pourquoi nous nous intéressons dans ce paragraphe à l'utilisation des architectures embarquées en biométrie.

Développer une application biométrique sur un système embarqué possède plusieurs avantages. Premièrement, les composants embarqués ont un rapport performance/coût assez intéressant : les systèmes embarqués dédiés à la biométrie sont développés pour répondre à la demande croissante de systèmes d'authentification forte à faible coût. Certains de ces composants (FPGA, ASIC) offrent en outre d'importantes possibilités de parallélisation, qui peuvent être utilisées pour accélérer les calculs de traitement de l'image les plus complexes. En effet, ces derniers peuvent souvent être appliqués à différentes zones de l'image simultanément et indépendamment, ce qui les rend massivement parallélisables. Les systèmes embarqués permettent enfin d'améliorer la sécurité de l'application, car les mécanismes et les données sont enfouis à l'intérieur du système. Leur accès est donc rendu difficile aux utilisateurs malveillants pour qui le système est une boîte noire.

Convaincus par cette dernière affirmation, Choi *et al.* [Choi06] ont étudié la possibilité d'implémenter la comparaison des échantillons biométriques sur une carte à puce. Ceci est basé sur le fait que de nombreux systèmes de vérification utilisent les cartes à puces pour stocker les échantillons biométriques des utilisateurs. Au cours du processus d'authentification, l'échantillon est donc lu par le système de reconnaissance afin d'être comparé avec l'échantillon courant. Or cet échange engendre un risque de sécurité, car l'échantillon peut être intercepté. Implanter un module de comparaison à l'intérieur de la carte à puce, permet de supprimer cette étape et de protéger les données stockées.

Le composant fonctionnel d'une carte à puce est un SoC qui contient un petit processeur et un peu de mémoire (voir Figure V.1). Seuls des traitements peu coûteux en calculs comme en mémoire peuvent donc y être implantés. Choi *et al.* travaillent sur la voix, et les algorithmes qu'ils utilisent sont basés sur une machine à support de vecteurs (SVM). Ils démontrent à travers leur étude que les SVM sont adaptés à la réalisation d'un système d'authentification du locuteur pouvant s'exécuter sur une carte à puce avec des performances raisonnables.

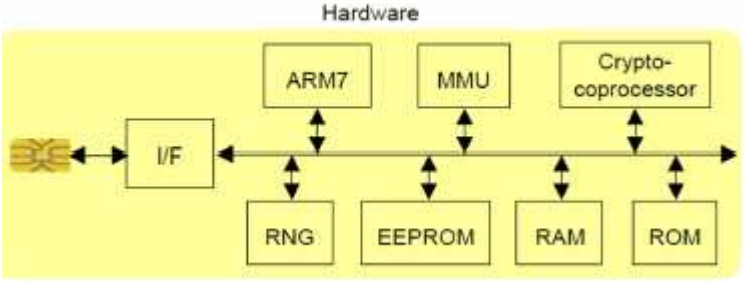


Figure. V.1 : *Configuration de la carte à puce utilisée par Choi et al. pour l'implantation d'algorithmes de vérification du locuteur (figure extraites de l'article [Choi06]).*

Les travaux de Choi *et al.* ne sont bien sur pas les seuls à avoir été consacrés à l'implantation de systèmes biométriques sur des cartes à puce. Chung *et al.* [Chung04], tout comme Yang *et al.* [Yang06]

se sont par exemple intéressés à l'optimisation d'algorithmes d'authentification d'empreintes digitales basés sur l'extraction des minuties.

Dans leur étude [Choi06], Choi *et al.* proposent également une solution pour les applications qui nécessitent l'utilisation d'une base centralisée (comme l'identification du locuteur dans un centre d'appel). Ils ont pour cela adapté leur système algorithmique basé sur les SVM, et l'ont implanté sur un FPGA. Leurs travaux révèlent que l'utilisation des FPGA peut augmenter de 50 fois la vitesse d'exécution par rapport à une implémentation logicielle. Sur le même principe, l'étude proposée par López-Ongil *et al.* [Lopez04] démontre que l'implantation d'un système de reconnaissance de la forme de la main sur FPGA peut être trois fois plus rapide que l'implémentation d'un système équivalent sur un GPP de coût similaire.

Il existe de nombreuses études qui comparent ainsi les implantations matérielles de divers algorithmes et arrivent à des conclusions analogues. Nous citerons encore Yang et Paindavoine [Yang03], qui ont implanté un système de localisation et de reconnaissance du visage basé sur les réseaux de neurones sur divers composants, et notamment sur un FPGA et un DSP. Leurs résultats prouvent encore une fois que les FPGA permettent d'atteindre des vitesses de traitement beaucoup plus élevées (un FPGA permet de traiter 14 images par seconde alors qu'un DSP ne permet d'en traiter que 4,8). Mais ils sont nuancés par la perte de performance que cela induit : les DSP permettent en effet d'atteindre des performance de reconnaissance sensiblement plus élevées que les FPGA (un taux de bonne reconnaissance de 98,2% pour le DSP, et de seulement 92% pour le FPGA).

La rapidité des implantations FPGA provient de leur capacité à paralléliser massivement les tâches. Ils sont d'ailleurs souvent employés pour accélérer un traitement particulier. Yoo *et al.* [Yoo07] proposent par exemple des systèmes multimodaux (visage+empreinte digitale et iris+empreinte digitale) composés d'un GPP, qui commande le système, et d'un FPGA programmé pour traiter les tâches les plus lourdes telles que la localisation du visage, la détection de l'iris, et le traitement des empreintes digitales.

Certains, comme Kumar*et al.* [Kuma07], ont imaginé des algorithmes permettant d'exploiter au maximum cette capacité. Ils présentent en effet dans leur étude, un système de reconnaissance du visage basé sur la décomposition en composantes principales (PCA), qui traite l'image de visage morceau par morceau de manière indépendante. L'algorithme utilisé est appelé Wheight Modular Principal Component Analysis car il consiste à découper l'image en fenêtres, puis à appliquer une PCA sur chacune d'elles, et enfin à fusionner les paramètres obtenus par une somme pondérée. Le traitement de chaque fenêtre peut donc être exécuté en parallèle sur un FPGA, ce qui permet de réduire significativement le temps de calcul.

V.2. Présentation de la plateforme d'implémentation

Le choix d'une plate-forme de test pour valider une méthodologie de développement est un élément important. En effet, celle-ci doit permettre une prise en main rapide et être adaptée au domaine de prototypage envisagé (traitement d'image pour notre cas). Dans cette section, nous allons énoncer les critères qui ont permis de définir la plate-forme de test sous ses aspects matériel et logiciel.

V.2.1 – Plateforme matérielle

Pour valider le flot de conception proposé dans le domaine du traitement d'image en temps réel, nous avons défini un cahier des charges pour choisir une plateforme matérielle de test. Celle-ci doit répondre aux critères suivants :

- disposer d'un FPGA d'une capacité suffisante permettant l'implantation de plusieurs modèles de processeur pour former une machine parallèle,
- posséder une mémoire vive externe permettant de stocker un nombre d'images couleur important de taille variable,
- proposer une interface simplifiée (parallèle) permettant de connecter un capteur CMOS couleur OEM ou un capteur de type Webcam,
- dialoguer avec un ordinateur externe (USB, Série, Ethernet …) afin de pouvoir faciliter la mise au point de la chaîne de traitement.
- respecter un budget raisonnable (< 3000 €).

Après une étude des cartes disponibles au moment du début du projet (Janvier 2008) et répondant à nos critères, notre choix s'est porté sur une carte fournie par Xilinx : la carte ML605 (voir Figure V.2). Nous allons détailler les caractéristiques des composants utilisés pour la validation de notre flot de conception après une description générale de la carte.

V.2.1.1 – Synoptique de la carte ML605

Dans la Figure V.3, les principaux éléments de la carte ainsi que leurs inter-connexions sont représentés. Nous pouvons noter la présence des composants suivants :

- *Sortie vidéo* : Codec DVI Chrontel CH7301C,
- *FPGA Virtex 6 XC6VLX 240T,*
- *Interface Ethernet* : Marvell Alaska PHY device 88E1111,
- *Mémoire externe RAM* : Micron 512 MB MT4JSF6464HY-1G1,

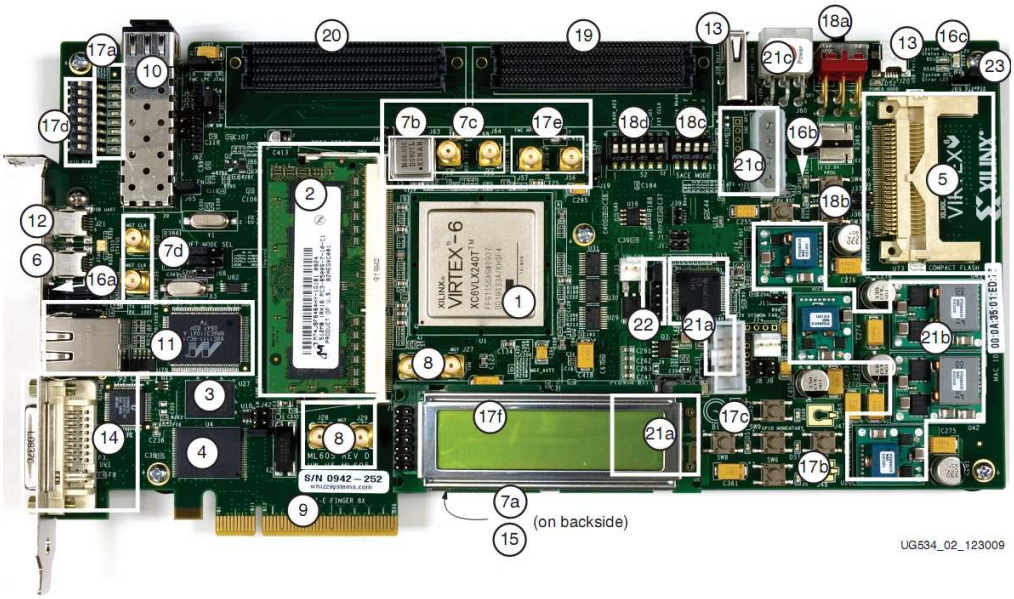


Figure V.2 : *Aperçu de la carte ML605.*

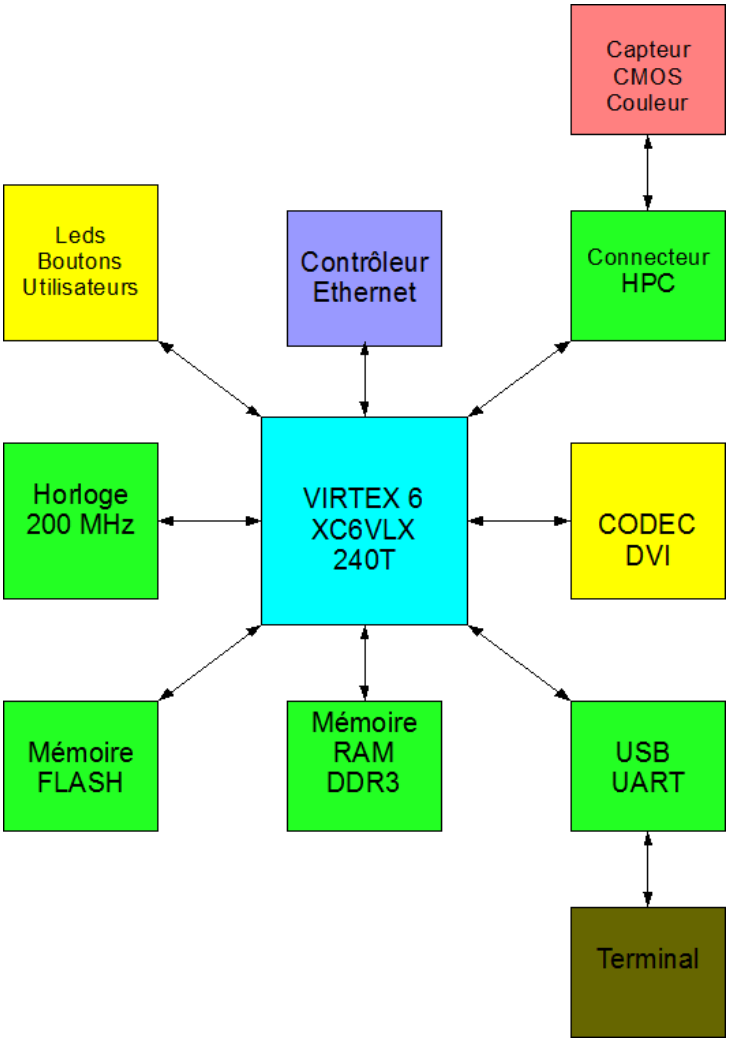


Figure V.3 : *Synoptique de la carte ML605.*

- *Mémoire externe FLASH* : 128 Mb Xilinx XCF128X-FTG64C,
- *Convertisseur USB UART* : CP2103GM USB-to-UART bridge,
- *Contrôleur USB* : Cypress CY7C67300 EZ-Host.

V.2.1.2 – Entrées/Sorties d'image

Notre plateforme est destinée à traiter des flux d'images provenant de deux sources différentes :

Capteur CMOS Omnivision OV7620

Ce capteur VGA (640x480) couleur est en liaison avec le FPGA présent sur la carte par le connecteur HPC. L'interface est de type parallèle avec un bus de donnée vidéo de 16 bits et quelques signaux de synchronisation. La vitesse d'acquisition maximale est de 60 images/s. Une connexion 2 fils utilisant un protocole I2C permet la configuration du capteur.

Capteur de type Webcam connecté par un bus USB

Beaucoup de capteur CMOS disposent d'une interface intégrée USB. Si cette connexion est bien évidemment présente sur tous les ordinateurs de bureau ou les portables, la présence d'une interface USB fonctionnant en mode HOST n'est pas toujours garantie sur une carte d'expérimentation à base de FPGA. La ML605 dispose du circuit CYPRESS CYC67300 permettant de répondre à nos besoins. Ce circuit, comme on peut le constater en Figure V.4, n'est pas une simple interface USB. Il dispose d'un cœur de processeur RISQ 16 bits. L'interface utilisateur dispose d'un grand nombre de connexion et permet l'adressage d'un espace mémoire propre. Le bus de contrôle permet de travailler en mode DMA avec un élément extérieur et gère les interruptions. Dans notre dispositif, ces éléments facilitent le transfert des images acquises par le capteur d'image. Dans les notes d'application de la ML605, XILINX fournit une IP permettant de faciliter la programmation du circuit CYC67300.

Codec DVI Chrontel CH7301C pour la sortie d'image

Pour notre carte d'expérimentation, nous avons souhaité disposer d'une sortie d'image, directement connectée à un écran VGA. Ceci est possible grâce à la présence d'un circuit de sortie DVI. Ce circuit codec [CH7301C] (voir Figure V.5) permet d'effectuer l'encodage d'un signal vidéo numérique au format DVI (Digital Video Interface) ou DFP (Digital Flat Panel). Le Codec est connectée aux entrées/sorties du FPGA. Le format d'entrée du Codec permet de gérer des images RGB 24 bits. La résolution maximale de l'image est de 1600x1200 pixels. Un module implanté dans le FPGA permet de lire des images depuis la mémoire et de gérer les signaux d'entrée du Codec. Les différents modes de fonctionnement du codec sont programmables via une interface série 2 fils.

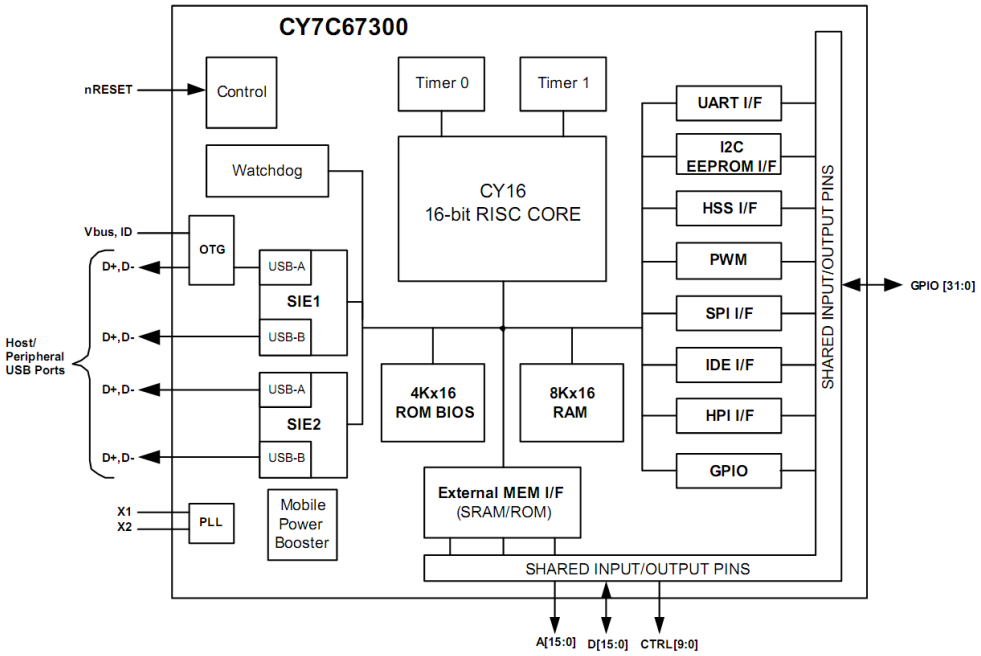


Figure V.4 : *Synoptique du contrôleur USB CYPRESS CYC67300.*

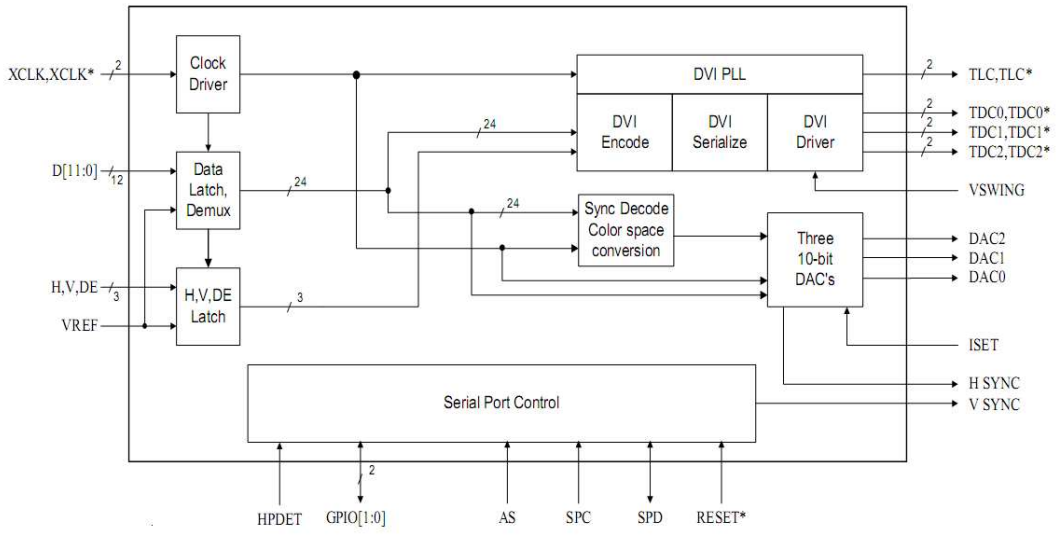


Figure V.5 : *Synoptique et les différents signaux du codec DVI Chrontel CH7301C.*

V.2.1.3 – Mémoires externes

Mémoire RAM dynamique DDR3

La carte ML605 dispose d'un connecteur SODIMM qui permet de disposer d'un module de mémoire dynamique de type DDR3 de référence MT4JSF6464HY de la société Micron. Les caractéristiques de ce bloc mémoire sont résumées ci dessous :

- 512 Mega Octets organisé en 64X64 Mega Octets,
- vitesse de transfert possible jusqu'à 85 Gbit/s,
- fréquence des bus : 800 MHz.

Les accès mémoire se font généralement sur des données rangées consécutivement en mémoire. Ainsi le mode d'accès en *rafale* (**burst mode**) permet d'accéder aux sept données consécutives à la première sans temps de latence supplémentaire. Dans ce mode en rafale, le temps d'accès à la première donnée est égal au temps de cycle auquel il faut ajouter le temps de latence, et le temps d'accès aux sept autres données est uniquement égal aux temps de cycle.

Mémoire Flash Numonyx JS28F256P30

Pour une plateforme destinée au prototypage de systèmes embarqués, il est intéressant de disposer d'une zone de mémoire non volatile pour deux raisons principales :

- stockage de paramètres de configuration pour le dispositif embarqué,
- stockage du *bistream* de configuration du FPGA après la mise au point.

La carte ML605 possède une mémoire FLASH. Cette mémoire dispose d'un bus de donnée, d'un bus d'adresse et d'un bus de contrôle permettant un accès facile en écriture et en lecture, sans la nécessité de disposer d'un contrôleur de mémoire particulier.

V.2.1.4 – Le FPGA Virtex 6-XC6VLX240T

Le FPGA intégré dans la plateforme matérielle fait partie de la famille Virtex 6 de Xilinx [Xilinx]. Ce circuit programmable intègre :

Des CLB (Configurable Logic Bloc) : Xilinx indique souvent la capacité de ses circuits en *Slices*. Un slice, considéré comme une unité de base, comporte un ensemble de 4 LUT à 6 entrées, 8 bascules D et de la logique de contrôle. Dans la famille Virtex, 2 slices forment un CLB. La Figure V.6 montre le schéma de principe d'un Slice. Le FPGA Virtex 6-XC6VLX 240T dispose de 37680 Slices.

Des IOB (Input Output Bloc) : ces blocs d'entrées et de sorties sont principalement composés de buffers qui permettent de s'adapter à différents standards de niveau de tension. Certaines entrées sont plus particulièrement dédiées aux signaux d'horloge. Le Virtex utilisé sur notre carte dispose de 720 entrées-sorties.

Des blocs de calcul adaptés au traitement du signal (DSP48): Le FPGA utilisée dispose de 768 blocs DSP48 (voir Figure 7). Depuis l'introduction de la famille Virtex 5 au milieu des années 2000, Xilinx intègre des blocs de calcul spécifiquement dédiés au traitement du signal : ces blocs permettent d'effectuer les calculs suivants :

- multiplication 25x18 bits,
- multiplication avec accumulation (MAC),
- additionneur à 3 bloc d'entrées 48 bits,
- registre à décalage 48 bits,
- module de calcul logique (et, ou, ou exclusif) sur 48 bits,
- comparateur sur 48 bits,
- détecteur de « pattern ».

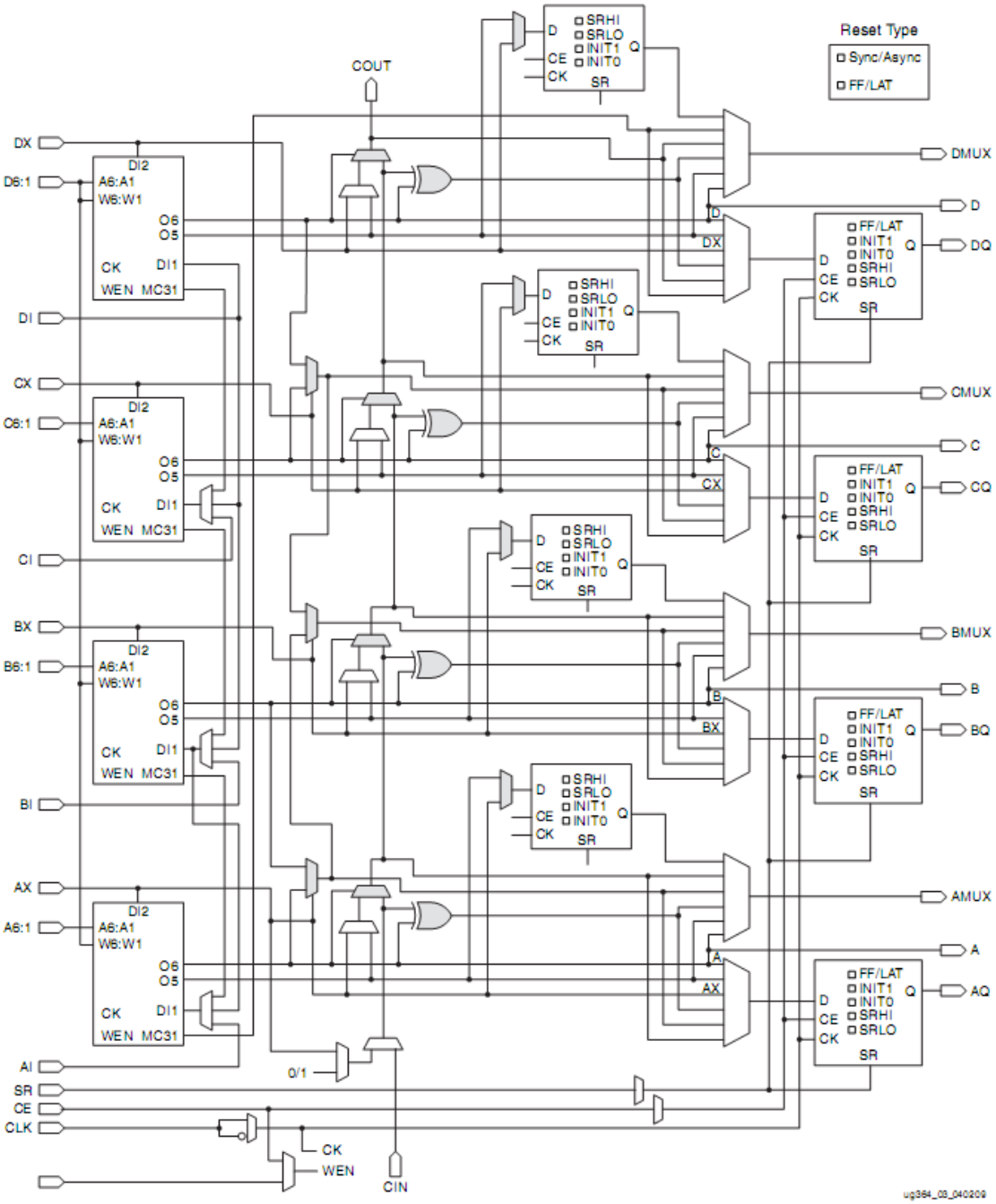


Figure V.6 : *Schéma d'un Slice de FPGA Virtex 6.*

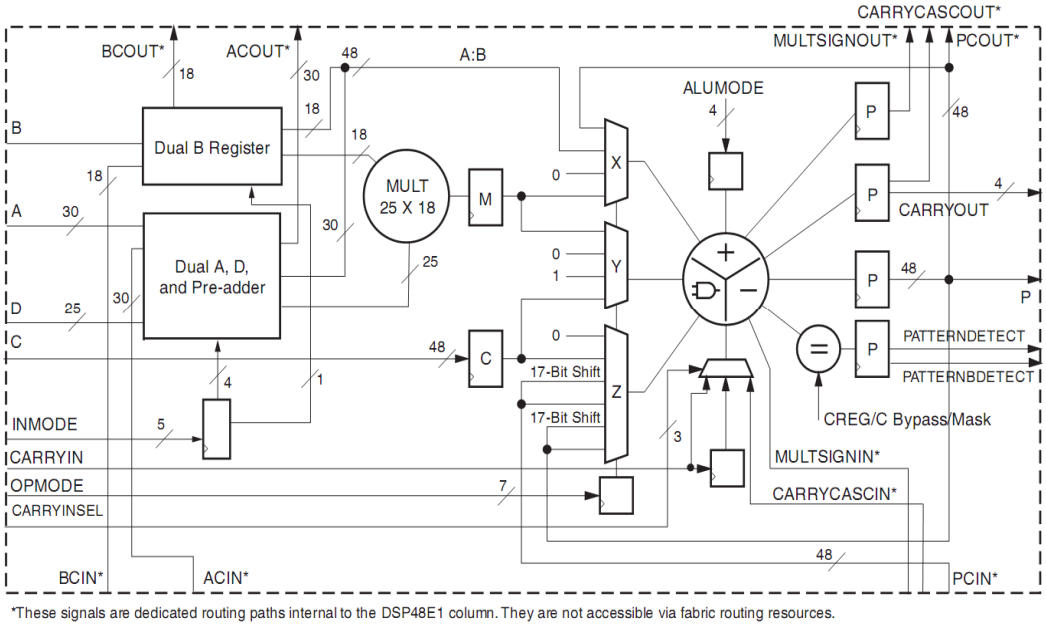


Figure V.7 : *Schéma d'un bloc DSP48 du FPGA Virtex 6.*

Des MMCM (Mixed-Mode Clock Manager) : les MMCM permettent la génération de signaux d'horloge. Un seul oscillateur génère plusieurs signaux avec des fréquences multiples ou sous multiples de la fréquence d'entrée. Le Virtex 6-XC6VLX 240T dispose de 12 MMCM. La figure V.8 montre le synoptique du module MMCM. Six sorties indépendantes en fréquence et en phase (8 phases possibles) sont présentes en sortie d'un MMCM.

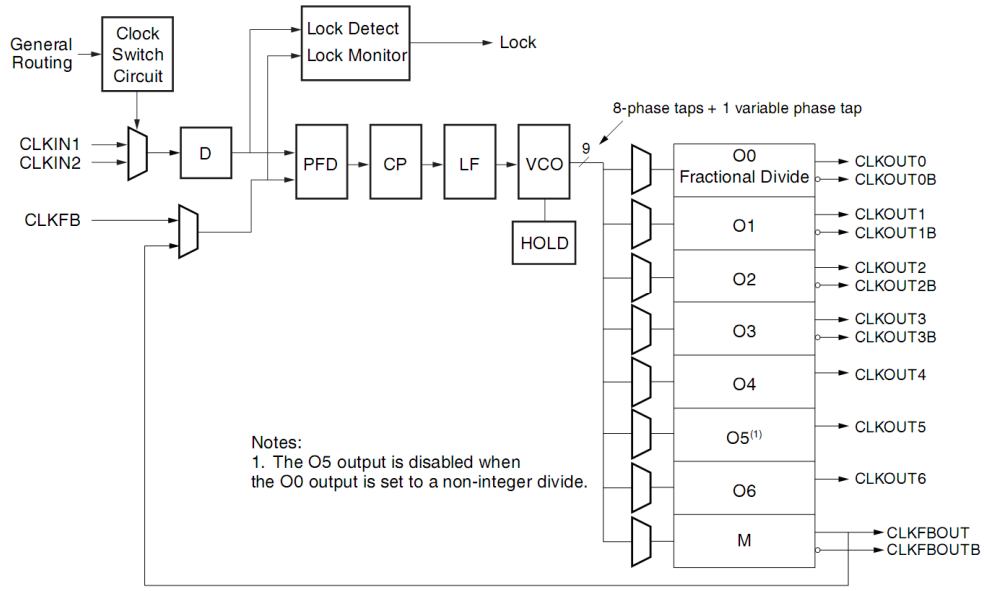


Figure V.8 : *Synoptique d'un MMCM de Virtex 6.*

Des blocs mémoires (BlocRAM) : Chaque virtex 6 possède un certain nombre de ressources mémoire appelées BlocRAM. Chaque BlocRAM a une taille de 36 Kbits, et le bus de donnée peut-être configuré de différentes tailles. Les valeurs disponibles sont 1, 2, 4, 9, 18 ou 36 bits. Les blocs peuvent être utilisés en simple ou double accès. Le Virtex 6-XC6VLX 240T dispose de 416 BlocRAM ce qui donne une capacité totale de stockage interne de 14976 Kbits. Ces blocRAM sont utilisables dans un mode double accès et XILINX propose une IP permettant d'implémenter simplement un mode de fonctionnement en quadruple accès. Ceci facilite grandement l'élaboration et le test de dispositif multi-softcore.

Une IP de gestion de la RAM dynamique externe : la gestion des mémoires dynamiques reste complexe due notamment aux besoins de générer des signaux périodiques destinés à assurer le fonctionnement du rafraîchissement dynamique. Xilinx, par la fourniture d'une IP, permet une grande simplification de la gestion des mémoires. Des options de configuration de l'IP permettent d'implémenter un accès simultané à plusieurs modules utilisateurs (jusqu'à 4). Cette particularité sera utilisée pour permettre d'implanter le processeur vu au chapitre III (4 accès mémoires simultanés nécessaires).

V.2.2 – Plateforme logicielle

Après avoir choisi la plateforme matérielle de test, nous avons défini une architecture logicielle qui permet une souplesse d'utilisation importante. Cette architecture, constituée d'un ensemble de modules fonctionnels, est présentée dans la figure V.9 :

- un module d'acquisition de l'image (acquisition des image depuis les capteurs CMOS et stockage en mémoire DDR3),
- un module de restitution de l'image (lecture depuis la mémoire DDR3 et envoi au circuit de codage DVI),
- des modèles de processeur à tester,
- un mécanisme, appelé répartiteur mémoire, permettant de partager la mémoire DDR3 entre le dispositif d'acquisition de l'image, le dispositif de restitution et les différents blocs processeurs à tester. Ce dispositif utilise une partie de la logique du FPGA pour réaliser un multiplexeur/demultiplexeur qui partage les bus de données et les bus d'adresse du contrôleur de mémoire,
- un module destiné à assurer la configuration des différents circuits contrôleur (USB, Capteur CMOS, circuit DVI). Ces circuits utilisent une interface I2C,
- un module permettant de générer les différentes fréquences d'horloge nécessaires au fonctionnement de l'ensemble,
- un module softcore de controle PicoBlaze permettant la synchronisation de l'ensemble.

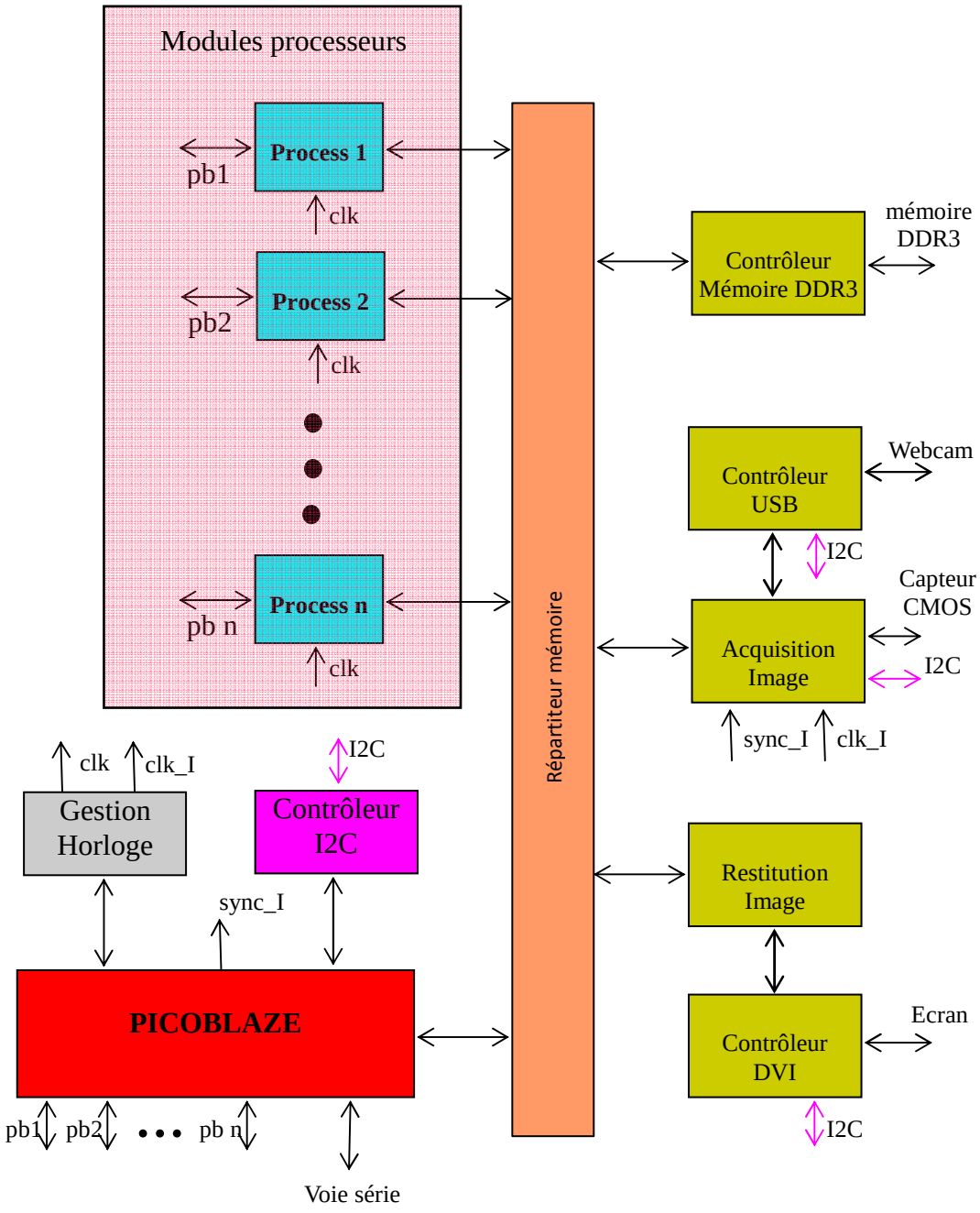


Figure V.9 : *Structure de l'architecture logicielle.*

Nous décrivons brièvement le fonctionnement de chaque module de l'architecture logicielle :

Le **module d'acquisition** réalise les fonctions suivantes :

- gestion des signaux de synchronisation du capteur CMOS OV7620,
- gestion des échanges avec le contrôleur USB (récupération des valeurs pixel de la webcam),
- gestion des entrelacements de l'image,
- gestion des différents protocoles (RGB, CCIR656...),
- stockage des informations RGB dans la mémoire,
- mise à la taille d'image désirée (Cif, QCif, QVGA, VGA ...).

Le **module de restitution** assure les fonctions suivantes :

- lecture des informations RGB dans la mémoire,
- gestion des entrelacements de l'image,
- gestion des différents protocoles (VGA, SVGA, CCIR656...).

Le dispositif d'accès et de partage de la mémoire, appelé **Répartiteur mémoire**, est un élément important du dispositif de test. En effet, différents éléments doivent avoir accès de façon séquentielle à la mémoire. On a donc implanté dans le FPGA un multiplexeur/démultiplexeur permettant de gérer les signaux des bus mémoire.

Les **modules processeurs** sont les modules générés par notre flot de conception vu au chapitre 3. Chaque fonction écrite en C génère son propre module (process 1, process 2... process n).

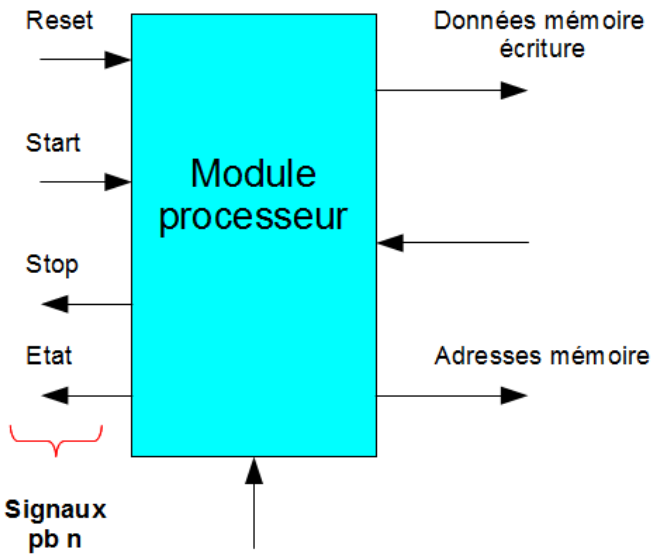


Figure V.10 : *Module processeur et ses connexions.*

Le **PicoBlaze** joue un rôle très important de synchronisation de l'ensemble de l'application de test. En effet, c'est lui qui pilote les multiplexeurs du module de répartition de la mémoire et aussi le démarrage et la remise à zéro des différents modules processeurs. Le programme du PicoBlaze permet, par l'intermédiaire de la voie série, d'utiliser les fonctions de mise au point suivantes :

- fonctionnement des modules processeur en mode pas à pas avec possibilité de point d'arrêt,
- visualisation du contenu de la mémoire,
- commande manuelle du répartiteur mémoire,
- commande manuelle des modules d'acquisition et de restitution,
- configuration manuelle des périphériques USB.

La structure de cette plateforme logicielle a été rendue suffisamment modulaire pour pouvoir prendre en compte les évolutions futures suivantes :

- changement de la carte de test,
- adaptation à différents périphériques de capture d'image.

V.3. Prototypage rapide d'extraction automatique de paumes

Dans cette section, nous allons décrire les implémentations sur la plateforme de la première phase du système biométrique. Il s'agit d'extraire automatiquement la zone de paume à partir d'une image de la main acquise sans contact. Nous décomposons le traitement en plusieurs étapes et pour chacune, nous rappelons d'abord le calcul à effectuer avant de donner le résultat du prototypage rapide. L'extraction de la paume nécessite une localisation de la main, suivi de la localisation de la paume et une normalisation (rotation, remise de l'échelle) dû à la spécificité de l'acquisition sans contact.

V.3.1 – Segmentation d'images

L'étape de segmentation consiste à effectuer un seuillage sur la composante rouge de l'image RVB : puisque nous utilisons un fond vert, tous les pixels possédant un niveau de rouge important appartiennent à la main.



Figure V.11 : *Images obtenues lors de la segmentation : image d'origine (gauche), image après le seuillage (au milieu) et image finale après l'érosion et la suppression de débris.*

Une érosion et une suppression de débris ont été réalisées sur l'image après le seuillage. La segmentation traite des images de 1600 x 1200 pixels. Figure V.11 montre les images obtenues lors de cette étape.

Les algorithmes correspondants à cette étape, décrits en langage C, ont fait l'objet d'une implémentation matérielle en utilisant le flot de conception décrit au chapitre III. Après compilation par OpenIMPACT, une représentation intermédiaire en langage Lcode est obtenue. Le parallélisme intrinsèque de ce code est ensuite extrait pour obtenir une représentation concurrente de l'algorithme visible sous forme de graphe. La Figure V.12 montre l'extraction ILP (Instruction Level Parallelism) d'une partie du traitement.

Après cette phase d'extraction, un outil recense les différents éléments matériels (opérateurs de calcul, registres …) justes nécessaires à l'élaboration des modèle de processeur destinés à effectuer l'étape de segmentation. La partie d'un processeur définie par le graphe d'extraction de la Figure V.12 est visible sur la Figure V.13.

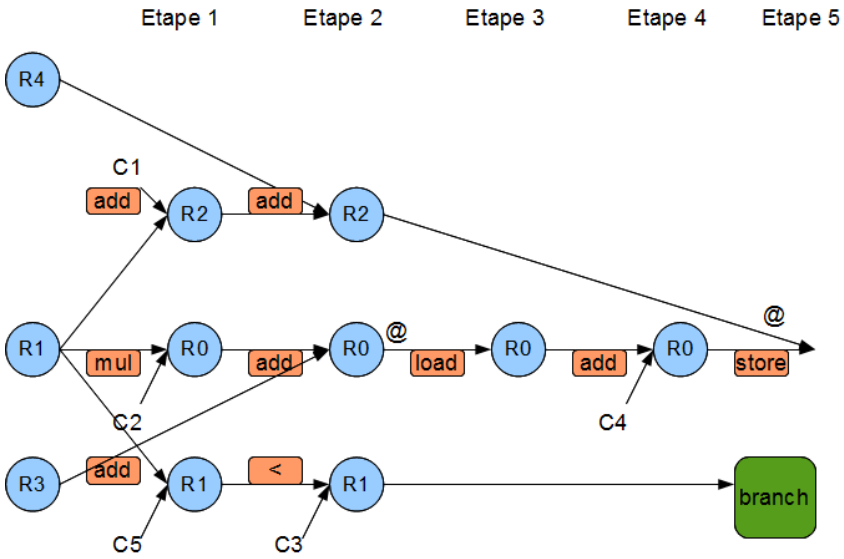


Figure V.12 : *Graphe d'extraction ILP d'une partie de traitement.*

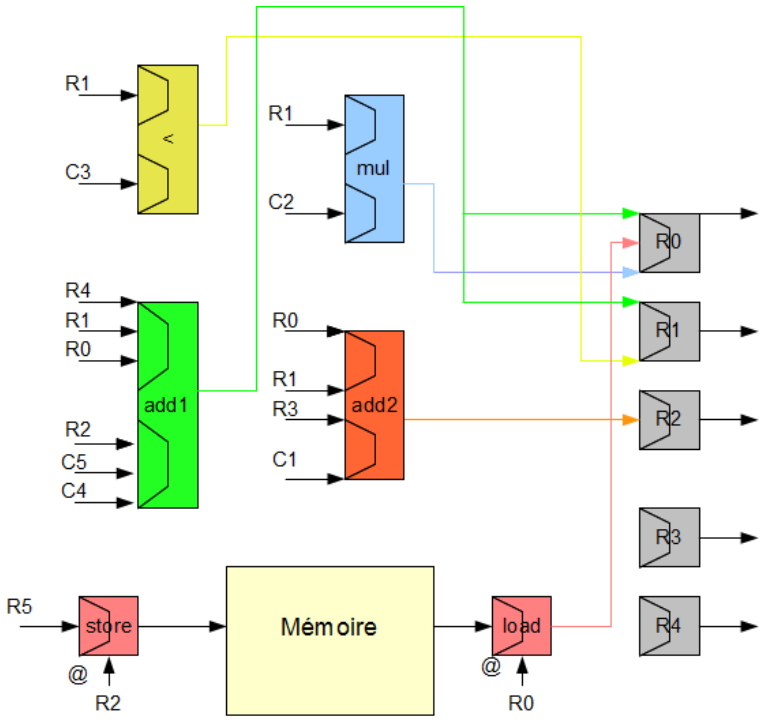


Figure V.13 : *Partie d'un processeur généré à partir du graphe de la Figure V.11.*

La dernière phase consiste en une génération automatique du modèle VHDL, une synthèse matérielle réalisé par ISE et un téléchargement du *bitstream* dans le FPGA de la plateforme. Les résultats de prototypage sont donnés dans le Tableau V.1.

Tableau V.1 : Résultats de prototypage rapide pour l'étape de segmentation.

Fonction	Tp (Taux de parallélisation)	Slices utilisés	Freq. (MHz)	Nombre de cycles nécessaires	Temps de traitement (µs)
Seuillage	3.6	393	380-376	9625600	25600
Erosion	4.2	720	320-320	14451200	45160
Suppression de débris	3.9	518	330-320	2094400	6545

Nous pouvons remarquer qu'un Taux moyen de parallélisation de Tp = 4 a été obtenu. Il y a très peu de ressources matérielles utilisées pour cette étape. Vu la taille importante de l'image, le nombre de cycles nécessaires est conséquent, même après l'extraction ILP. Toutefois, le temps de traitement reste raisonnable pour une application en temps réel. Dans la colonne indiquant la fréquence, deux valeurs sont présentées : la première correspond à la valeur théorique de synthèse et la seconde à la valeur pratique disponible sur notre plateforme.

V.3.2 – Détection de contours par le code Fremann

Après la segmentation, nous effectuons un suivi de contours par un algorithme nommé « code Fremann » basé sur l'analyse de voisinage (voir Section IV.3). La Figure V.14 illustre le résultat de détection de contours de la main.

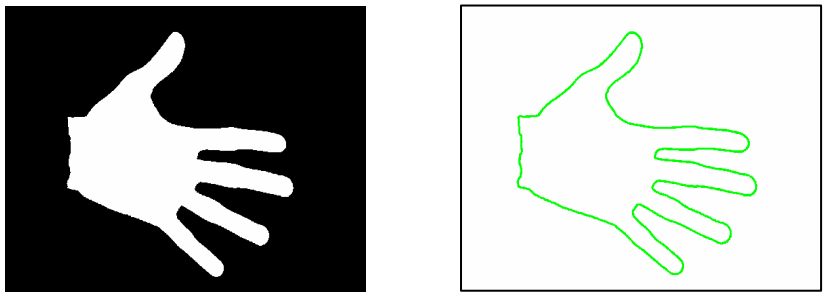


Figure V.14 : *Suivi de contours de main par le code de Fremann.*

Les coordonnées (x, y) de chaque point du contour sont stockées dans un vecteur à deux dimensions. Pour faciliter et alléger la suite des traitements, le vecteur de coordonnées est ré-échantillonnés à une taille fixe de 2048 points.

Nous avons suivi la même démarche que pour l'étape de segmentation pour implémenter ce code de Fremann sur le FPGA. Le processeur généré utilise 2020 slices, nécessite 76700 cycles d'exécution, et le temps de traitement correspond à 270 µs. Avant l'élaboration du processeur, l'extraction ILP nous permet d'obtenir un Tp = 6.3.

V.3.3 – Extraction des points caractéristiques de la main

Les points caractéristiques sont calculés en deux sous étapes : Carpointinit et Carpointadjust. Leur position est tout d'abord initialisée directement sur les coordonnées du contour, puis elle est réajustée par un algorithme itératif (voir section IV.3).

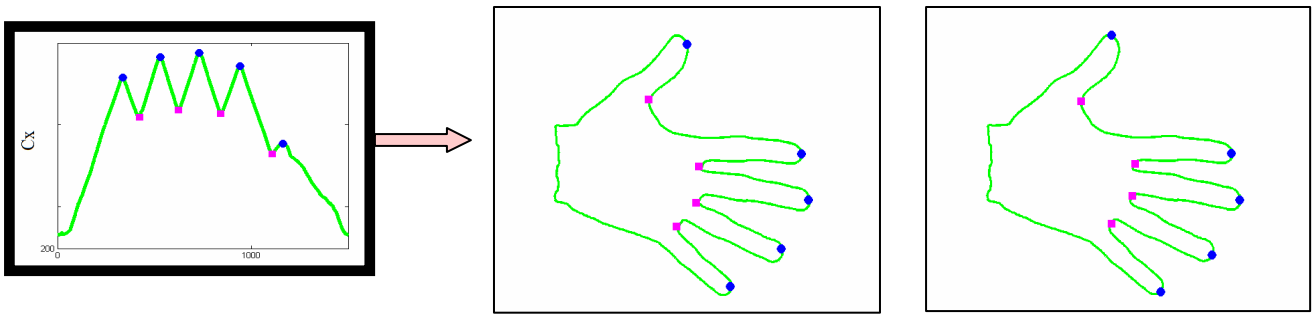


Figure V.15 : *Extraction des points caractéristiques de la main : coordonnées de l'abscisse des points de contour (gauche), initialisation des points caractéristiques (milieu) et résultat après l'ajustement.*

Tableau V.2 : Résultats de prototypage rapide pour l'extraction des points caractéristiques.

Fonction	Tp (Taux de parallélisation)	Slices utilisés	Freq. (MHz)	Nombre de cycles nécessaires	Temps de traitement (µs)
Carpointinit	5.8	545	295-290	65622	226
Carpointadjust	3.1	1267	310-304	72924	240

Le Tableau V.2 montre que comme pour le suivi de contours (code Fremann), le temps de traitement ainsi que le nombre de cycles nécessaire est considérablement réduit par rapport à la segmentation. Puisque nous traitons des données de la taille réduite (2048 points).

V.3.4 – Extraction de la ROI (paume)

Pour extraire la ROI (Region Of Interest), un carré de taille proportionnelle à la largeur de la main est d'abord défini à partir des points caractéristiques. L'image extraite est alors normalisée : elle subit une rotation et un sous échantillonnage pour atteindre la résolution fixe de 64 × 64 pixels. Enfin elle est convertie en niveau de gris. La ROI pour la phase de reconnaissance correspond à une image de 64 x 64 x 8 bits.

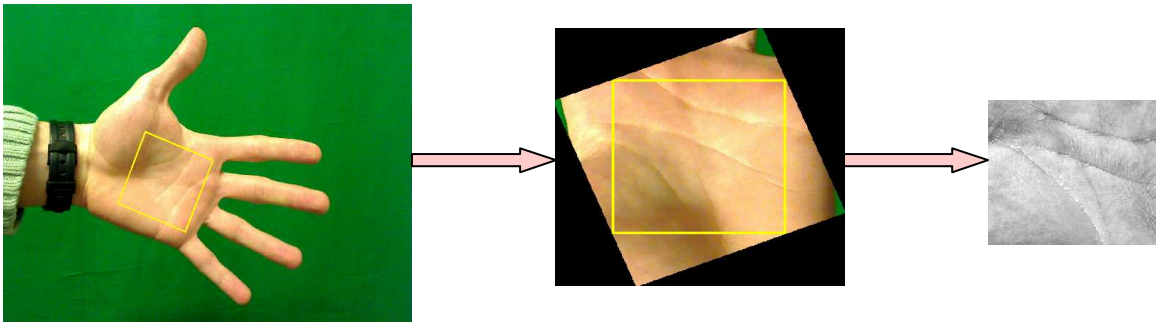


Figure V.16 : *Extraction de la paume : définition de la fenêtre ROI (gauche), résultats après la rotation (milieu) et la normalisation (droite).*

Le processeur généré pour cette étape d'extraction de la ROI utilise 1566 slices, nécessite 81967 cycles d'exécution, et le temps de traitement correspond à 295 µs. Nous avons obtenu un Tp = 8.3.

V.3.5 – Implémentation de la chaîne complète d'extraction de la paume

Figure V.17 montre les éléments utilisés pour l'implémentation complète de la chaîne d'extraction automatique de paume. Au FPGA Virtex 6, nous avons ajouté un capteur CMOS pour acquérir les images de mains ; la mémoire DDR3 est aussi nécessaire pour stocker les images d'origine et les résultats intermédiaires des différentes étapes.

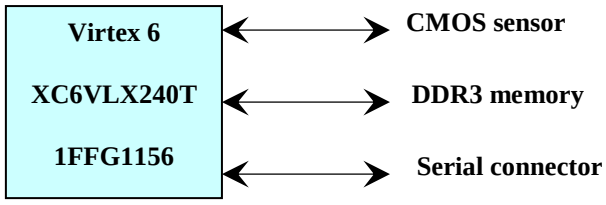


Figure V.17 : *Eléments utilisés pour l'implémentation de la phase de reconnaissance.*

En utilisant le flot de conception proposé, nous avons embarqué 7 processeurs RISP correspondant aux sept principales fonctions d'extraction de la ROI (voir Figure V.18). Pour synchroniser les modèles VHDL de processeurs, nous avons inséré un cœur de micro-contrôleur : PicoBlaze. Il s'agit d'un IP de

la société Xilinx et programmable en Assembleur. Quand une image de main arrive au FPGA, il déclenche d'abord le bloc d'acquisition et celui de la gestion de mémoire pour stocker l'image d'origine. Ensuite, il démarre séquentiellement les sept processeurs RISP à travers 7 signaux différents (pb*i*, i= 1 à 7). En même temps, il pilote aussi la mémoire partagée DDR3 en passant par le bloc de gestion de mémoire.

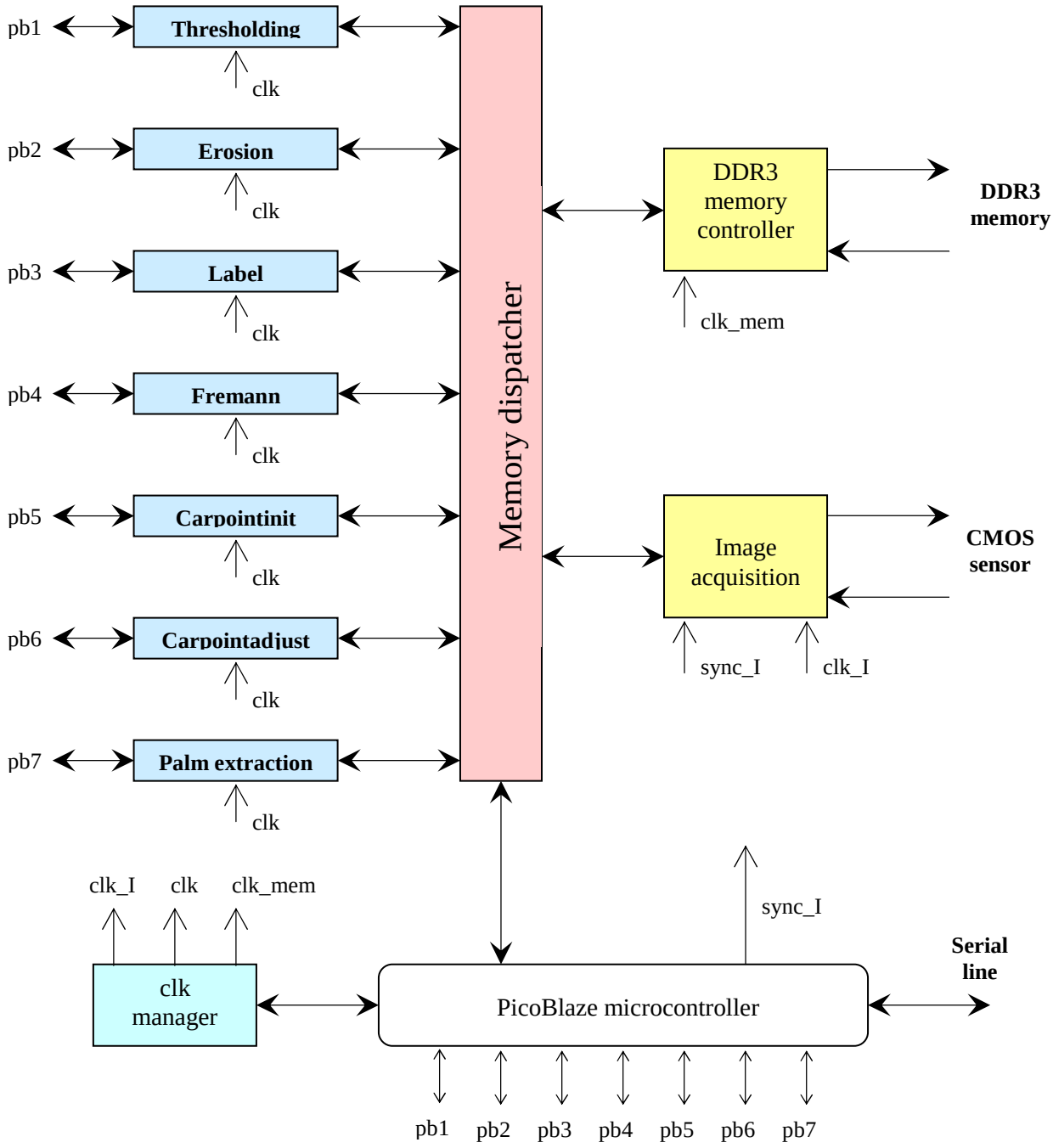


Figure V.18 : *Implémentation de la chaîne complète d'extraction de la ROI.*

Le bloc de gestion de l'horloge génère les différentes valeurs en fonction de la fréquence d'exécution de chaque processeur. Les résultats d'implémentation matérielle de la chaîne complète sont donnés au Tableau V.3. Nous pouvons remarquer que très peu de ressources matérielles sont nécessaire pour l'extraction de la ROI et une image de la main a été traitée en 78.3 ms.

Tableau V.3 : Résultats de prototypage rapide pour l'extraction de la ROI.

Ressources matérielle utilisées	Slices	DSP 48	BlocRAM
Nombre	8731	0	1
Pourcentage	23.2%	0%	0.24%

V.4. Prototypage rapide de reconnaissance de paumes

V.4.1 – Rapide rappel de la chaîne de reconnaissance de la paume

La paume présente une texture riche en lignes et en stries qui permet de différencier les personnes. Au vu de ses performances et dans l'optique de réduire le temps de calcul, nous avons choisi un filtre de Gabor ellipsoïdal réel pour extraire les caractéristiques pertinentes de paume. Ce filtrage a été réalisé à l'aide d'une convolution de l'image de paume avec un masque de 9×9 dont les coefficients ont été choisis empiriquement (voir Section IV.4). Ce dernier est suivi d'un seuillage du résultat visant à limiter la taille de la matrice de caractéristiques ainsi que le temps de calcul lors de la phase de comparaison.

La mesure de similarités utilisée consiste en une distance de Hamming normalisée, qui est une comparaison pixel à pixel utilisant l'opérateur booléen *ou exclusif*. Comme l'extraction de l'image de paume n'est pas forcément idéale, nous avons introduit une tolérance en translation : nous calculons la distance pour plusieurs décalages (5 pixels de décalage maximal horizontalement et verticalement), puis nous choisissons la distance minimale.

Nous rappelons que la ROI obtenue après l'extraction automatique de la paume correspond à une région de 64 x 64 pixels, représentée par le niveau de gris. La Figure V.19 illustre la taille d'images durant la phase de reconnaissance.

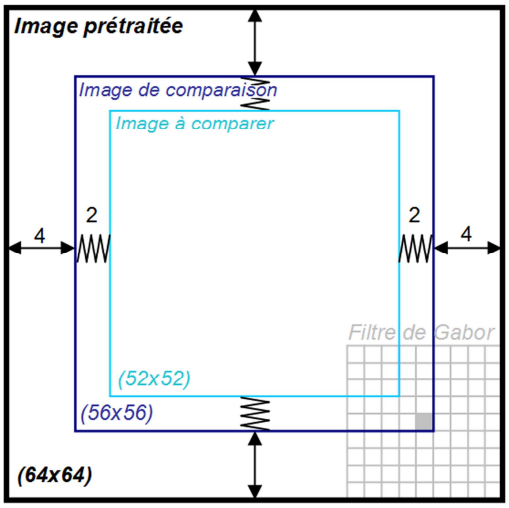


Figure V.19 : *Taille d'images durant la reconnaissance : image pré-traitée d'origine = 64 x 64 pixels, image d'exemple = 56 x 56 pixels à cause de l'effet du bord avec la convolution 9 x 9, et image de test = 52 x 52 pixel avec 5 translations horizontales et verticales.*

V.4.2 – Prototypage rapide de reconnaissance de la paume

Après la phase de prétraitement qui consiste à extraire la ROI d'une image de main, la paume a été soumise à la phase de reconnaissance qui compare les traits biométriques à tester aux exemples d'apprentissage. En lien avec les performances de reconnaissance présentées dans le chapitre IV, nous avons choisi la configuration suivante pour les implémentations matérielles : 25 sujets dans la base d'apprentissage avec 2 exemples par sujet. Donc, la base d'exemples se compose de 25 x 2 = 50 images de la taille 56 × 56 pixels (voir Figure V.19). Il est à noter que pour chaque paume de test, 5×5=25 comparaisons ont été effectuées afin d'introduire certaines tolérances en translations verticales et horizontales.

La Figure V.20 affiche le graphe d'extraction ILP obtenu avec notre outil qui correspond à la convolution d'un pixel avec une ligne de coefficients [1 x 9]. Le Taux de parallèlisation est de Tp = 3.4. Le nombre de cycles nécessaire est de 56 x 56 pixels x (9 x 8 +1) = 228928.

La Figure V.21 montre le schéma du modèle VHDL du processeur destiné à effectuer la convolution [9 x 9]. Une fréquence de 220 MHz a été obtenue lors de la synthèse sous ISE avec un nombre de slices utilisé de 1316 et 3 DSP48. Le temps de traitement pour une image de 56 x 56 pixels est de 1.04 ms. Il est à noter que la binarisation du résultat de convolution a été aussi réalisé lors de cette étape.

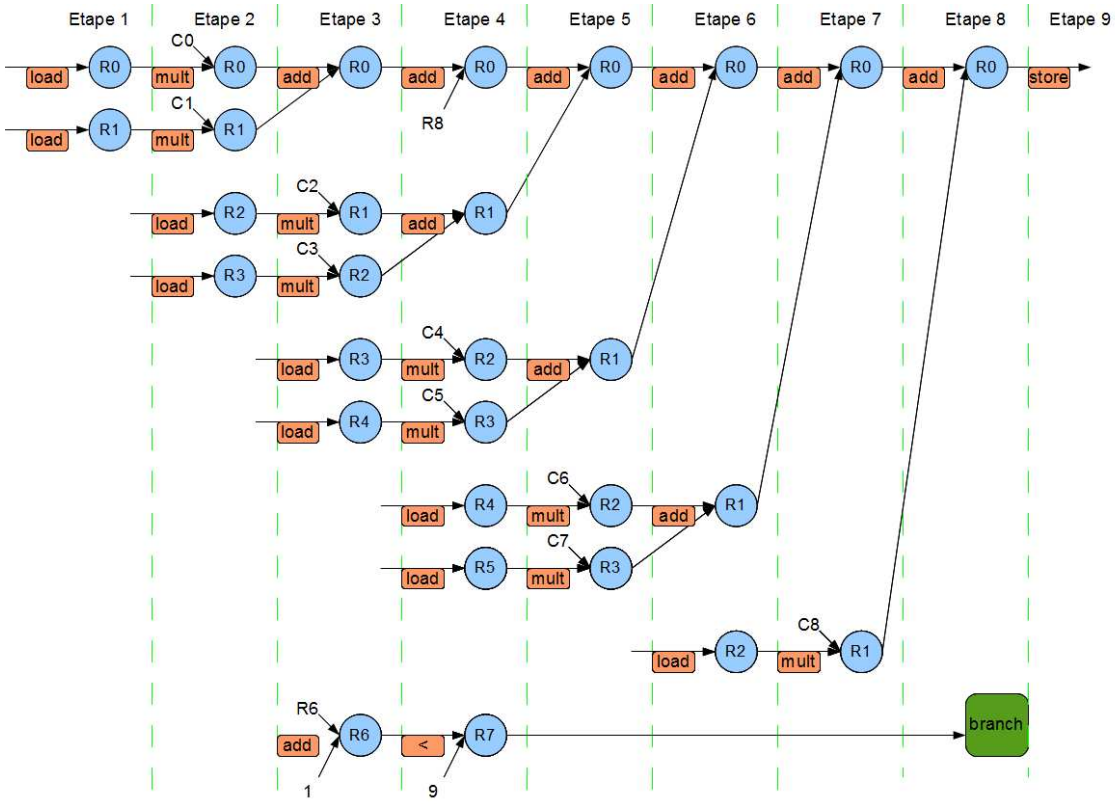


Figure V.20 : *Extraction de l'ILP du noyau de convolution [1 x 9].*

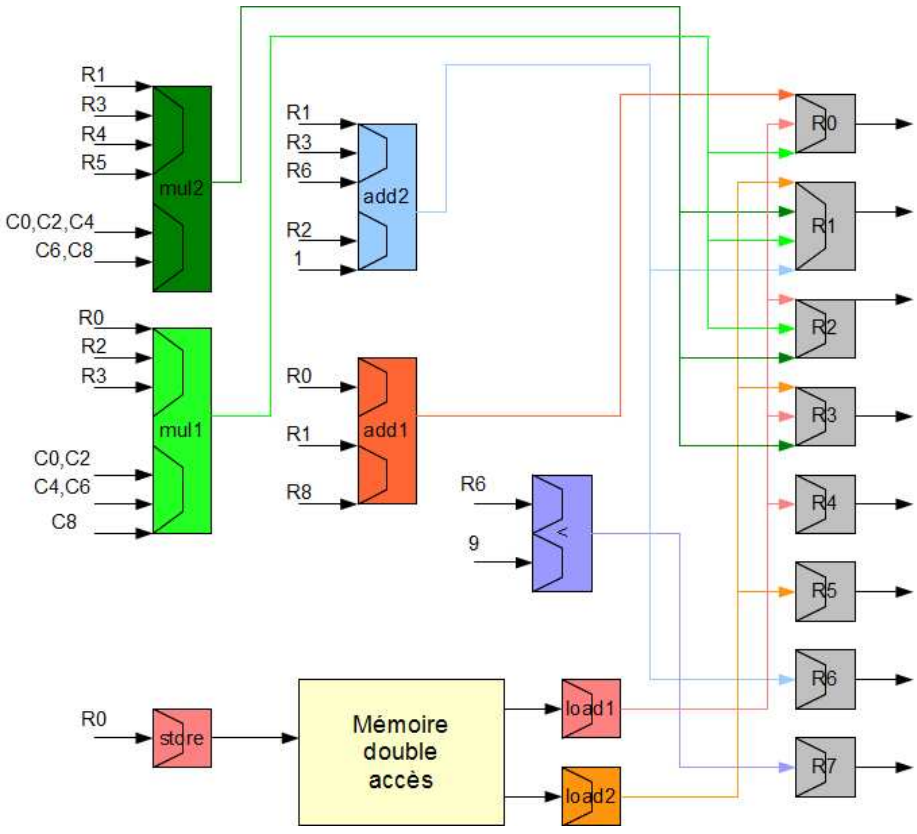


Figure V.21 : *Schéma VHDL du modèle de processeur généré pour la convolution [9 x 9].*

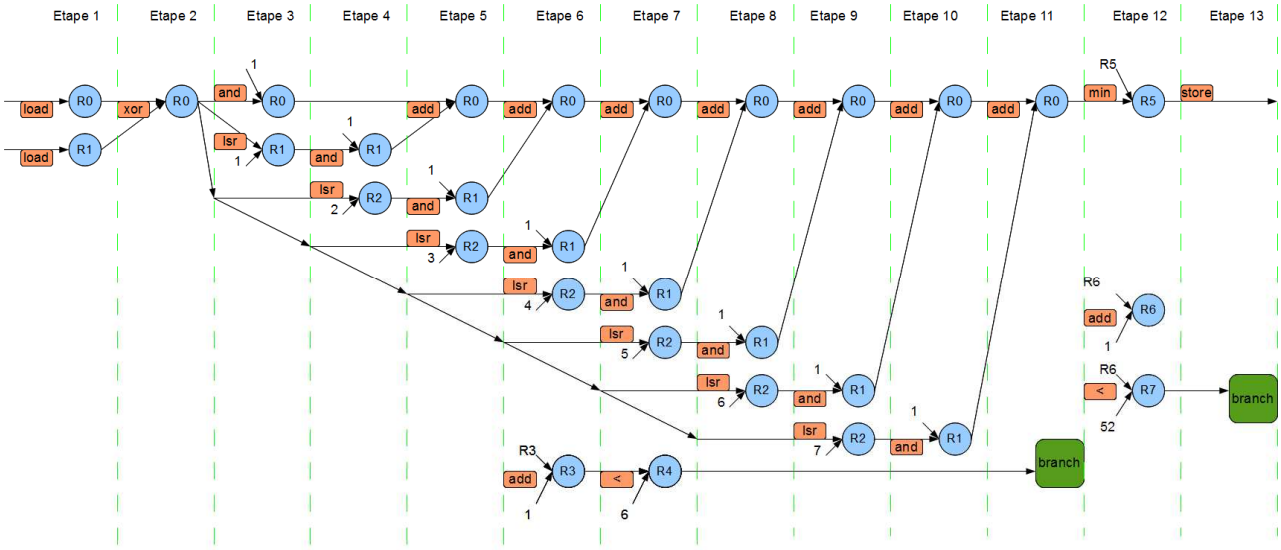


Figure V.22 : *Extraction de l'ILP de l'étape de décision : les étapes 1 à 11 comparent 8 bits (somme des OU exclusif). Ceci se répète d'abord 7 fois (8 x 7 = 56 pixel d'une ligne d'image de test), puis 52 fois grâce aux étapes 12 et 13.*

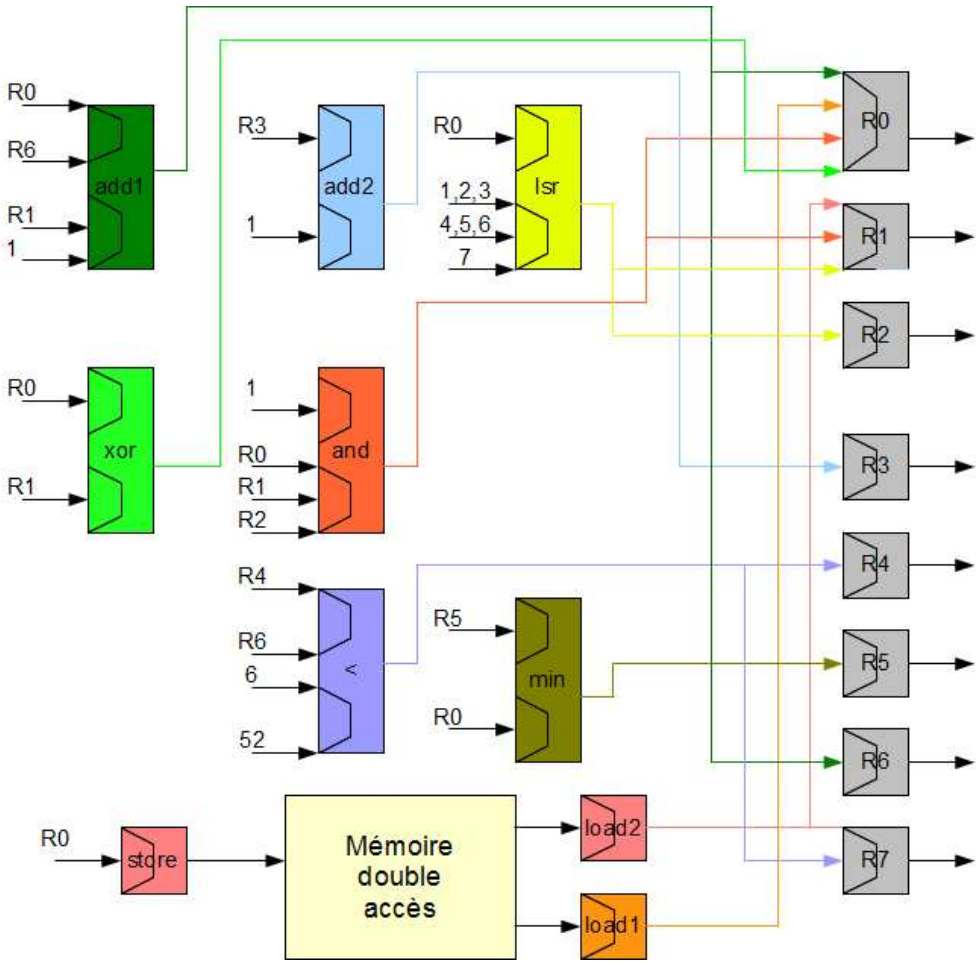


Figure V.23 : *Schéma du modèle VHDL du processeur destiné à la comparaison d'un exemple d'apprentissage sans translation.*

La Figure V.22 affiche le graphe d'extraction ILP correspondant à la somme des OU exclusifs effectuée pour calculer la similarité de l'image de test avec un exemple de la base d'exemples. Avec la tolérance horizontale et verticale, cette opération est réalisée 25 fois avant de stocker le minimum obtenu parmi les 25. La décision finale résulte de la répétition du traitement précédent pour chaque exemple (50 dans notre cas). Le nombre de cycles nécessaire est de ((11 x 7) + 2) x 52 lignes x 25 translations x 50 sujets = 5135000.

La Figure V.23 montre le schéma du modèle VHDL du processeur destiné à effectuer une partie de la comparaison. Une fréquence de 200 MHz a été obtenue avec un nombre de slices utilisé de 976 et aucun DSP48. Le temps de la prise de décision pour une image de test est de 25 ms.

V.4.3 – Implémentation optimale de reconnaissance de la paume

Pour évaluer notre flot de conception, nous avons aussi implémenté la phase de reconnaissance de la paume d'une manière manuelle. C'est-à-dire, l'extraction du parallélisme de différents niveaux a été réfléchie et réalisée par un expert en électronique. De même, nous avons aussi essayé de tirer au maximum des avantages et des spécificités de la plateforme de test.

Nous avons gardé la même configuration d'implémentation que le prototypage rapide (voir Section V.4.2) : 25 sujets dans la base d'exemples avec 2 exemples par sujet ; 25 comparaisons par exemple avec la tolérance horizontale et verticale. L'application a été directement programmée en langage de bas niveau VHDL en tenant compte de ses parallélismes intrinsèques.

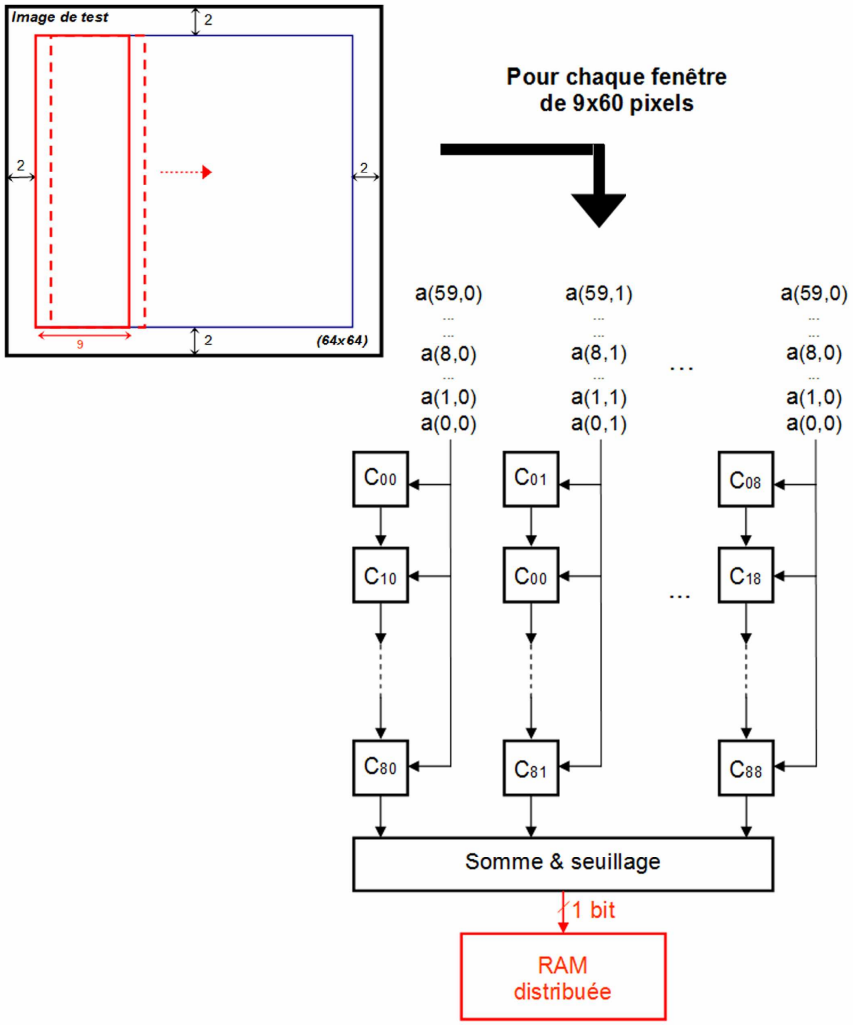


Figure V.24 : *Structure parallèle de l'étape de filtrage de Gabor : l'image de paume est distribuée dans les fenêtres successives de 60 x 9 pixels, une architecture de 9 lignes x 9 colonnes de DSP48 Slices réalise 81 multiplications et accumulations simultanément.*

Les Figures V.24 et V.25 illustrent la structure parallèle pour les deux étapes distinctes de la reconnaissance : réalisation de 81 multiplications et accumulations (convolution avec le masque 9 x 9) en un seul cycle et parallélisation massive de 50 comparaisons basées sur la distance de Hamming. Le cœur PicoBlaze contrôle et synchronise ces deux étapes de traitement.

Tableau V.4 : Comparaison des deux implémentations matérielles.

Ressources matérielle utilisées	Slices	DSP 48	BlocRAM
Prototypage rapide	2292 (6%)	3 (0.4%)	0 (0%)
Implémentation optimale	8566 (22.7%)	89 (11.6%)	2 (0.5%)

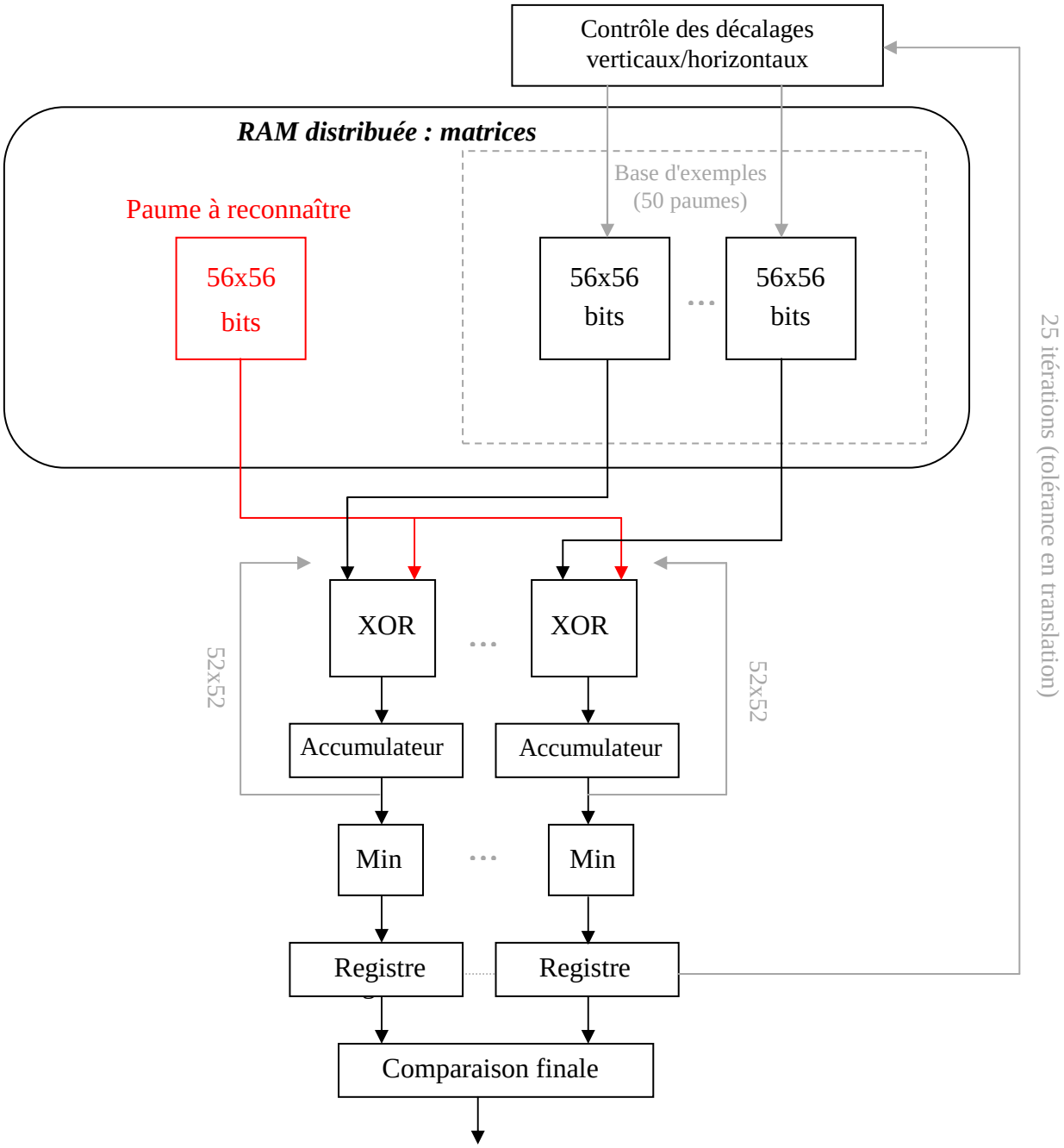


Figure V.25 : *Structure parallèle de l'étape de comparaison : 50 distances de Hamming sont calculées en même temps (en 52 x 52 cycles) et ceci est répété 25 fois (correspondant aux 25 décalages).*

La vitesse obtenue avec les deux implémentations est respectivement de 26.04 ms et de 0.43 ms. L'implémentation optimale est 60 fois plus rapide par rapport au prototypage rapide en suivant le flot de conception. Par contre, la première utilise beaucoup plus de ressources matérielles du fait de l'exploitation du parallélisme au maximum à tous les niveaux. Le nombre de slices important présent

dans la seconde est dû en partie au stockage des exemples dans la mémoire distribué disponible sur le FPGA Virtex 6. Tandis que ces exemples sont mémorisés dans la mémoire externe DDR3 de la plateforme pour le prototypage rapide.

Il est à noter que l'implémentation optimale a été réalisée par un expert en VHDL connaissant bien les FPGAs. Le temps de développement et de mise au point est estimé à 2 semaines complètes. Par contre, en utilisant notre flot de conception, n'importe quel développeur en informatique peut embarquer une application sur le FPGA sans aucune connaissance matérielle. Le temps de développement est réduit au minimum : une fois le code C mise au point, les différentes étapes d'implémentation (extraction ILP, génération du modèle et synthèse) sont quasi instantanés.

V.5. Bilan du chapitre

Dans ce chapitre, nous avons d'abord présenté la plateforme choisie pour tester et valider le flot de conception proposé sur la chaîne complète d'un système biométrique sans contact, basé sur la reconnaissance de paumes. Ensuite, les deux phases de celui-ci ont été implémentées sur le FPGA Virtex-6. L'extraction automatique de la ROI contient des calculs complexes et irréguliers. Sa réalisation matérielle a été effectuée grâce au flot de conception. Pour la phase de reconnaissance, nous avons mené une étude comparative entre deux méthodes : l'implémentation optimale et le prototypage rapide pour illustrer les avantages et les inconvénients de notre approche.

Chapitre VI
Conclusions et Perspectives

VI.1. Conclusions

Les travaux que nous venons de présenter s'inscrivent dans le contexte des systèmes embarqués dédiés au traitement d'image en temps réel avec une intégration toujours plus grande. Dans cette étude, nous avons proposé et validé un flot de conception de prototypage rapide pour l'implantation matérielle des applications sur une cible FPGA. Par la suite, nos conclusions s'attachent aux deux parties suivantes : la conception et l'expérimentation.

Dans un premier temps, les 4 aspects suivants nous permettent d'effectuer une comparaison entre le flot de conception proposé et les méthodes existantes pour mieux situer notre travail.

Prototypage rapide

Dans le domaine du prototypage rapide, beaucoup d'outils sont basés sur les variantes du langage C comme System C, Handel C et Architectural C, considérés comme des langages de description mixte logiciel/matériel. Ils possèdent des instructions capables de spécifier des comportements non séquentiels qui seront implantés dans des blocs matériels. Ce sont des langages très puissants avec des compilateurs qui peuvent effectuer la répartition logiciel/matériel, malgré le fait que l'exécutif matériel ne soit pas toujours optimal. Un facteur qui freine la diffusion plus large de ces langages réside dans le fait que les développeurs ont besoin de connaissances matérielles, et restent ainsi difficilement maîtrisables par des non-electroniciens. Notre approche propose d'embarquer des processeurs RISPs dans un composant matériel FPGA. Les ingénieurs logiciels peuvent ainsi continuer de développer leurs applications dans un environnement C ou C++. Le parallélisme intrinsèque d'un algorithme sera alors extrait par le compilateur de Trimaran pour être exécuté simultanément sur plusieurs opérateurs fonctionnels.

Adaptation aux outils co-design de haut niveau

La tendance actuelle vise à intégrer sur un seul circuit un système complexe logiciel/matériel (SoC). Des outils qui proposent un haut niveau d'abstraction et spécifient les composants logiciels/matériels en utilisant le même modèle sont apparus et facilitent la conception du SoC. Dans ce contexte, nous

pouvons facilement intégrer le flot de conception proposé dans l'environnement graphique Matlab-Simulink qui dispose d'une librairie de traitement d'images. Un algorithme peut être spécifié sous Matlab et ensuite, en fonction des caractéristiques de chaque bloc de traitement (contraintes temporelles, parallélisme intrinsèque, …), nous pouvons les distribuer soit à un composant logiciel, soit à notre modèle VHDL du processeur.

Processeur optimisé

Par rapport aux PSoCs (Programmable Systems on Chip) qui intègrent un cœur complet de processeur, nous embarquons un processeur RISP adapté à l'application ciblée. Nous pouvons considérer ce processeur RISP comme un processeur ASIP reconfigurable. Un ASIP (Application Specific Instruction set Processor) possède un jeu d'instructions et une architecture taillés pour une application donnée. Notre modèle est aussi reconfigurable car son jeu d'instructions, ses chemins de données et ses bancs de registres sont différents pour chaque application.

Performances en temps réel

Un autre aspect de notre travail consiste en la possibilité d'embarquer plusieurs modèles RISP sur un seul FPGA pour former une machine parallèle. L'objectif est d'augmenter les performances de traitement en temps réel.

Notre flot de conception applique certains critères des outils de co-design comme la flexibilité, la modularité, et la ré-utilisabilité. Les processeurs embarqués dans les travaux présentés sont des RISPs et le circuit cible est un produit Xilinx. La méthodologie peut être généralisée sur d'autres processeurs et d'autres FPGAs.

En ce qui concerne la partie expérimentale, nous avons testé et validé le flot de conception proposé en utilisant une plateforme à base de FPGA Virtex-6. Nous avons réalisé le prototypage rapide de plusieurs algorithmes de traitement d'images : une liste de traitements couramment rencontrés dans le domaine des images et une chaîne complète d'un système biométrique sans contact. Les résultats obtenus nous montrent qu'une application peut être facilement programmée en C, et ensuite automatiquement convertie en VHDL et implémentée sur un FPGA d'une manière optimale.

VI.2. Perspectives

Les perspectives à la suite de ces travaux sont multiples. Tout d'abord, à court terme, nos projets de recherches consistent à effectuer des travaux d'optimisation au niveau d'extraction ILP. Nous allons chercher le meilleur compromis entre le nombre de cycle d'exécution en parallèle (vitesse) et le nombre d'opérateurs à générer (surface). Ceci commencera par une définition de métriques adéquates.

Nous voulons aussi construire une interface entre le flot de conception et un logiciel de Co-design qui nous aiderait à définir les points suivants :

- nombre de processeurs RISPs utilisés,
- optimisation de Placement/Ordonnancement des tâches.

Nous pouvons utiliser ces outils dans deux cas de figure : soit avec des ressources matérielles complètement figées, par exemple une carte commerciale, soit pour la conception de l'architecture en cours. Dans le premier cas, ce flot de conception nous permettrait d'exploiter au maximum les ressources matérielles disponibles pour augmenter les performances en temps réel sous la contrainte de consommation minimale. Dans le second cas, une architecture optimisée taillée sur l'application pourrait être rapidement définie et simulée. Le cycle de développement réduit permettrait de connaître facilement les ressources matérielles nécessaires (nombre de Slices, nombre de Blocs RAM …) pour chaque configuration. Avec ces informations, les efforts des ingénieurs pourraient se concentrer sur l'optimisation de l'algorithme et la distribution optimale des tâches au niveau software.

Un autre travail prioritaire concerne l'amélioration et l'optimisation du modèle VHDL du processeur RISP. Une réécriture du code VHDL de certaines opérateurs de calcul serait profitable pour accélérer la vitesse du fonctionnement du modèle et minimiser encore les ressources matérielles nécessaires. Il serait intéressant aussi d'adapter et d'optimiser le code VHDL en tenant compte des spécificités des cibles FPGA utilisés (Xilinx ou Altera). Par contre, le modèle perdrait en généricité.

A moyen terme, notre objectif est d'insérer le flot de conception dans l'environnement graphique Matlab-Simulink pour proposer un outil complet de co-design et de prototypage rapide. Nous envisageons aussi d'appliquer la méthodologie proposée avec d'autres modèles de processeurs. L'évolution continue de la capacité d'intégration laisse envisager dans un avenir proche la possibilité d'implanter sur un même circuit plusieurs centaines de processeurs simples et compacts. Les travaux en collaboration avec des industriels qui consistent à concevoir et à réaliser une architecture massivement parallèle destinée à la détection et à la reconnaissance d'objets en temps réel sont en cours dans notre équipe.

Bibliographie

A

[Abe05] N. Abel, “Outils et méthodes pour les architectures reconfigurables dynamiquement à grain fin – Synthèse et gestion automatique des flux de données”, PhD thesis, Université de Cergy-Pontoise, 2005.

[AG04] Infineon Technologies AG, “Infineon presents the world’s most powerful magnetic ram property”, http://www .infineon.com, June 2004.

[Alt05] Altera. Stratix ii devices, *http://www.altera.com/products/devices/stratix2/st2-index.jsp*, 2005.

[Altera] *http://www.altera.com*.

[Ardoise] D. Demigny, M. Paindavoine and S. Weber, “Architecture à reconfiguration dynamique-Application au traitement temps reel d’image”, Traitement du Signal, 14(6), pp.615-623, 1997.

B

[BA02] M. Barbaro and *al.* , “A 100 x 100 pixels silicon retina for gradient extraction with steering filters capabilities and temporal output coding”, IEEE Journal of Solid-State Circuits, 37(2), Feb. 2002.

[BB05] S. Bouchoux and E. Bourennane, “An application based on dynamic reconfiguration of FPGAs : JPEG 2000 arithmetic decoder”, Optical Engineering, 44(10), pp. 107001 1:6, Oct. 2005.

[Ben04] P. Benoit, “Architecture des accélérateurs de traitement flexibles pour les systèmes sur puce”, PhD thesis, Université de Montpellier II, 2004.

[Ben07] K. Benkrid, A. Benkrid and S. Belkacemi, “efficient FPGA hardware development: A multi-language approach”, Journal of System Architecture, 53, 184–209, 2007.

[BM02] J. Batlle, J. Marti and al., “A new FPGA/DSP based parallel architecture for real time image processing”, Real Time Imaging, Vol.8, pp.345-356, 2002.

[Bou01] H. Bouma, “Design and implementation of an FPGA”, PhD thesis, University of Twente, 2001.

[Brost] V. Brost, F. Yang et *al*., “Multiple modular very long instruction word processors based on Field Programmable Gate Array”, Journal of Electronic Imaging, 16(2), 023001:1-10, 2007.

C

[Cart99] C. Carter and al., “Predicated static sigle assignment”, Proc. Of the International Conference on Parallel Architecture and Compilation Techniques, 1999.

[Camp05] S. Campbell, J.P. Chancelier, R. Nikoukhah, “Modeling And Simulation in Scilab/Scicos”, Edition Springer, Octobre 2005.

[Celo01] Handel-C Language Reference Manual, Version2.1, Celoxica Limited, 2001.

[Chameleon] Architecture Chameleon : http://www.chameleonsystems.com.

[CH7301C] www.chrontel.com/pdf/7301ds.pdf

[Choi06] WY Choi, D Ahn, SB Pan, KI Chung, YH Chung, and SH Chung, “SVM-based speaker verification system for match-on-card and its hardware implementation”, ETRI JOURNAL, 28(3) :320–328, JUN 2006.

[Chung04] Yongwha Chung, Daesung Moon, Sung Bum Pan, Min Kim, and Kichul Kim, “A hardware implementation of fingerprint verification for secure biometric authentication systems”, In ICIAR (2), pages 770–777, 2004.

[Conn05] Tee Connie, Andrew Teoh Beng Jin, Michael Goh Kah Ong, and David Ngo Chek Ling, “An automated palmprint recognition system”, Image and Vision Computing, 23(5) :501 – 515, 2005.

D

[DART] R. David, D. Chillet and *al.*, "DART : A Dynamically Reconfigurable Architecture dealing with future mobile telecommunications Constraints", In proceeding of the 9th reconfigurable Architecture Workshop (RAW'02), 2002.

[Daug93] J.G. Daugman, "High confidence visual recognition of persons by a test of statistical independence", IEEE Transactions on Pattern Analysis and Machine Intelligence, 15 :1148–1161, 1993.

[Dav03] R. David, "Architecture reconfigurable dynamiquement pour les applications mobiles", PhD thesis, Université de Rennes I, 2003.

[DH01] P. Dudek and P.J. Hicks, "A general – purpose CMOS vision chips with a processor-per-pixel SIMD array", in proceeding of ESSIRC'01, 2001.

[Doub07] Julien Doublet, Olivier Lepetit, and Marinette Revenu, "Contactless palmprint authentication using circular gabor filter and approximated string matching", In SIP '07 : Proceedings of the Ninth IASTED International Conference on Signal and Image Processing, pages 511–516, Anaheim, CA, USA, 2007. ACTAPress.

[Duta08] Helin Dutagaci, Bulent Sankur, and Erdem Yoruk, "Comparative analysis of global hand appearance-based person recognition", Journal of Electronic Imaging, 17(1) :011018, 2008.

E

[E7V] Triscend E5 Customizable Microcontroller Platform, March 2003.

[EdFo] Martyn Edwards and Benjamin Fozard, "Rapid prototyping of mixed hardware and software systems", Research paper, Department of Computation, UMIST www.co.umist.ac.uk

[Exalibur] Excalibur Device Overview, Altera, May 2002.

F

[Far01] P. Faraboschi, "The design of a technology platform for custom VLIW embedded processors", Computer Physics Communications, Vol.139, pp.104-108, 2001.

G

[GARP] J. Hauser and J. Wawrzybek, “GARP: A MIPS processor with a reconfigurable coprocessor”, In proceeding of IEEE Workshop on FPGAs for Custom Computing Machines, pp.12-21, April 1997.

[Ginh99] D. Ginhac, “Prototypage rapide d’applications parallèles de vision artificielle par squelettes fonctionnels”, Thèse soutenue à l’Université Blaise pascal – Clermont II en 1999.

[Glo99] A. De Gloria, “Microprocessor design for emnedded system”, Journal of Syetem Architecture, Vol.45, pp.1139-1149, 1999.

[Goh08] Goh Kah Ong Michael, Tee Connie, and Andrew Beng Jin Teoh, “Touch-less palm print biometrics: Novel design and implementation”, Image and Vision Computing, 26(12) :1551 – 1560, 2008.

H

[Har00] R. Hartenstein and H. Grünbacher, “The Roadmap to reconfigurable computing”, In proceeding of FPL 2002, LNCS, Springer –Verlag 2000.

[Hey05] B. Heyrman, “Etude de l’architecture système d’une camera intelligente et validation par portage d’applications”, PhD thesis, Université de Bourgogne, 2005.

[HG02] K.H. Hong, W.S. Gan and al., “Rapid prototyping of DSP algorithms on VLIW TMS320C6701 DSP”, Microprocessors and Microsystems, Vol.26, pp.311-324, 2002.

I

[IBG02] IBG, “Which is the best biometric technology?”, http://www.biometricgroup.com/reports/public/reports/best_biometric.html, 2002.

[IVP05] IVP. The Smart Vision Sensor 010518.pdf, *http://www.sickivp.se/html/ivpmain.html*, 2005.

J

[Jain04] A. K. Jain, Arun Ross, and Salil Prabhakar, “An introduction to biometric recognition”, IEEE Transactions on Circuits and Systems for Video Technology, 14(1): 4–20, Jan 2004.

[JHDL] *http://www.jhdl.org/documentation/guide.html*

[Jiang07] X. Jiang, W. Xu, L. Sweeney, Y. Li, R. Gross, and D. Yurovsky, “New directions in contact free hand recognition”, In International Conference on Image Processing (ICIP), volume 2, pages 389–392, 2007.

[Jing04] Xiao-Yuan Jing and David Zhang, “A face and palmprint recognition approach based on discriminant dct feature extraction”, IEEE Transactions on Systems, Man, and Cybernetics, Part B, 34(6) :2405–2415, 2004.

[Jing06] Xiao-Yuan Jing, Chen Lu, and David Zhang, “An uncorrelated fisherface approach for face and palmprint recognition”, In ICB, pages 682–687, 2006.

[Jing07] Xiao-Yuan Jing, Yong-Fang Yao, David Zhang, Jing-Yu Yang, and Miao Li, “Face and palmprint pixel level fusion and kernel dcv-rbf classifier for small sample biometric recognition”, Pattern Recognition, 40(11) :3209 – 3224, 2007.

K

[Kapc06] Adrian Kapczynski, “Relationship between it security and users’ needs”. In Proceedings of the International Multiconference on Computer Science and Information Technology, pages 297–302, Wisla, Poland, 2006.

[Kath00] V. Kathail, M. Schlasker, and B.R. Rau, “HPL-PD architecture specification:version 1.1”, technical Report HPL-9380, Hewlett Packard Laboratories, Feb. 2000.

[Kluw99] Kluwer Academic Publishers, “VLSI : Systems on a Chip”, ISBN : 0 7923 7731 1, 1999, Sélection des meilleures contributions de IFIP 10th International on VLSI.

[Kong02] Wai Kin Kong and David Zhang, “Palmprint texture analysis based on low-resolution images for personal authentication », volume 3, page 30807, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[Kong04] Adams Waikin Kong and David Zhang, “Feature-level fusion for effective palmprint authentication”, In Proc. of the 1 st ICBA, LNCS *3072*, pages 761–767, 2004.

[Kong08] Jun Kong, Yinghua Lu, Shuhua Wang, Miao Qi, and Hongzhi Li, “A two stage neural network-based personal identification system using handprint”, Neurocomput., 71(4-6) :641–647, 2008.

[Kong09] Adams Kong, David Zhang, and Mohamed Kamel, “A survey of palmprint recognition”, Pattern Recognition, 42(7) :1408 – 1418, 2009.

[Kuma06] Ajay Kumar, David C.M. Wong, Helen C. Shen, and Anil K. Jain, “Personal authentication using hand images”, Pattern Recognition Letters, 27(13) :1478 – 1486, 2006.

[Kuma07] A. Pavan Kumar, V. Kamakoti, and Sukhendu Das, “System-on-programmablechip implementation for on-line face recognition”, Pattern Recognition Letters, 28(3) :342–349, 2007.

[Kuma09] A. Kumar and D. Zhang, “User authentication using fusion of face and palmprint”, 9(2) :251–270, April 2009.

L

[Labview] *http://www.ni.com/labview/*

[Lam09] S.K. Lam and T. Srikanthan, “Rapid design of area-efficient custom instructions for reconfigurable embedded processing”, Journal of System Architecture, 55, 1-14, 2009.

[Laks05] N.C. Lakshmi and *al.*, “Trimaran: an infrastructure for research in Instruction-Level Parallelism”, LCPC 2004, Eds. R. Eigenmann, LNCS 3602, Springer – Verlag Heidelberg, 32-41, 2005.

[Leon] *http://www.hitechglobal.com/ipcores/leon3.htm*

[Li06] Qiang Li and Zhengding Qiu, “Handmetric verification based on feature-level fusion”, International Journal of Computer Science and Network Security, 6(2) :164–168, 2006.

[Lin05] Chih-Lung Lin, Thomas C. Chuang, and Kuo-Chin Fan, "Palmprint verification using hierarchical decomposition", Pattern Recognition, 38(12) :2639 – 2652, 2005.

[Loh00] F. Lohier "Méthodologie de programmation et evaluation des processeurs de traitement de signal parallèles pour le traitement d'images en temps réel", PhD Thesis, Université.

[Loo02] S. M. Loo, B. Earl Wells, J. Kulick, "Handel-C for Rapid Prototyping of VLSI Coprocessors for Real Time Systems", Southeastern Symposium on System Theory, 17-18 March, 2002, Huntsville, Alabama, USA.

[Lopez04] C. Lopez-Ongil, R. Sanchez-Reillo, J. Liu-Jimenez, F. Casado, L. Sanchez, and L. Entrena, "Fpga implementation of biometric authentication system based on hand geometry", In Proceedings of Field Programmable Logic and Application (FPL) Conference, volume 3203, pages 43–53., august 2004.

M

[Mag00] DSP Architecture directory, EDN mag, 2000, *www.ednmag.com*.

[Mal02] N. Malasné, "Localisation et reconnaissance de visages en temps réel : algorithmes et architectures", PhD thesis, Université de Bourgogne, 2002.

[MAP-CA] C. Basoglu, K. Zhao and *al.*, "The MAP-CA VLIW based Media Processor from Equator Technologies Inc and Hitachi Ltd".

[Matlab] The Math Works Inc., *http://www.mathworks.com*.

[McA07] J. McAllister, R. Woods et al., "Rapid implementation and optimisation of DSP systems on FPGA-centric heterogeneous platforms", Journal of System Architecture, 53, 511-523, 2007.

[MD95] T.G. Morris and S.P. DeWeerth, "An analog VLSI morphological image processing circuit", in Conference for Advanced Research in VLSI-Chapel Hill, North Carolina, Mar. 1995.

[ML05] A.T. Moreo, P.N. Lorente and al., "Experiences on developing computer vision hardware algorithms using Xilinx system generator, Microprocessors and Microsystems, Vol.XX, pp.1-9, 2005.

[Moi97] A. Moini, “Vision chips or seeing silicon”, Technical rapport, The centre for High Performances Integrated Technologies ans Systems, The University of Adelaide, March, 1997.

N

[Nann08] Loris Nanni and Alessandra Lumini, “Wavelet decomposition tree selection for palm and face authentication”, Pattern Recognition Letters, 29(3) :343 – 353, 2008.

[NAPA] C. Rupp, M. Landguth and *al.*, “The NAPA Adaptative Processing Architecture”, In proceeding of IEEE Symposium on FPGAs for Custom Computing Machines, pp.28-37, April 1998.

[Nav02] D. Navarro, G. Cathébras, G. Cambon, “Rétine d’acquisition du flot optique : intégration de la transformée de Cencus ”, colloque de CAO de circuits et systèmes intégrés, Paris 15,16,17 mai 2002

[NDBG97] Y. Ni, F. Devos, M. Boujrad and J.H. Guan, “A histogram equalization based adaptive image sensor for real-time vision”, IEEE Journal of Solid-State Circuits, 32(7), pp.1027-1036, July 1997.

[NG00] Y. Ni and J.H. Guan, “A 256 x 256-pixel smart CMOS image sensor for line based stereo vision applications”, IEEE Journal of Solid-State Circuits, 35(7), pp.1055-1061, July 2000.

[Niang05] P. Niang, T. Granspierre, M. Akil, Y. Sorel, “SynDEx-IC : un Environnement Logiciel pour l’Implantation d’Applications temps réel sur Circuits reconfigurables ”, JFAAA’05, 6ème Journée Francophone de la méthodologie AAA, Dijon, France, 18 Janvier – 21 Janvier 2005

P

[Pan01] P.R. Panda, “SystemC-a modeling platform supporting multiple design abstractions”, Proceeding of the 14th International Symposium on System Synthesis, pp.75-80, 2001.

[Pan08] Xin Pan and Qiu-Qi Ruan, “Palmprint recognition with improved two-dimensional locality preserving projections”, Image Vision Comput., 26(9) :1261–1268, 2008.

[Pat97] D. Patterson et al., “Intelligent RAM (IRAM) : chips that remember and compute”, IEEE international Solid State Circuits Conference, San Francisco, CA, Fev. 1997.

[Poin11] A. Poinsot, “Traitements pour la reconnaissance biométrique multimodale : Algorithmes et Architectures”, Ph.D. thesis, Université de Bourgogne, Février 2011.

[Poly06] PolyU palmprint database, http://www.comp.polyu.edu.hk/~biometrics, 2006.

[Power] Virtex-II Pro platform: introduction and Overview, Xilinx, August 2003.

R

[Rabb03] R. Rabbah and K. Palem, “Data remapping for design space optimization of embedded memory systems”, Special issue of the ACM Transaction in Embedded Computing Systems, 2003.

[Rabb04] R. Rabbah and al., “Compiler orchestrated prefetching via speculation and predication”, Proc. of the 11th Inter. Conf; on Architectural support for Programming Languages and Operating Systems, oct. 2004.

[RaPid] C. Ebeling, D.C. Cronquist and P. Frankin, “RaPid – Reconfigurable Pipelined Datapath”, In proceeding of the 6th International workshop on Field-Programmable Logic and Applications, 1996.

[Riba08] Slobodan Ribaric, Ivan Fratric, and Kristina Kiš, “A novel biometric personal verification system based on the combination of palmprints and faces”, Informatica, 19(1) :81–100, 2008.

[Ring] G. Sassatelli, G. Cambon and *al.*, “The Systolic Ring : A reconfigurable Systolic Architecture”, ACM Field Programmable Gate Array 2001 (FPGA’01), Monterey, Californie, Fev. 2001.

[Rizz02] D. Rizz and O. Colavin, “A video compression case study on a Reconfigurable VLIW Architecture”, Proceeding of the DATE’02 Conference, pp.540-546, Paris, Mars 2002.

[Roux03] S. Roux, “Adéquation Algorithme-Architecture pour le traitement multimédia embarqué”, PhD thesis, Institut Polytechnique de Grenoble, 2003.

S

[Sas02] G. Sassatelli, “Architectures reconfigurables dynamiquement pour les Systems sur puce”, PhD thesis, Université de Monpellier, 2002.

[Savi07] Tadej Savic and Nikola Pavešic, “Personal recognition based on an image of the palmar surface of the hand”, Pattern Recogn., 40(11) :3152–3163, 2007.

[Shi00] S.G. Shiva, “Computer design and Architecture”, Third edition, Marcel Dekker editor, ISBN 0 8247 0368 5, 2000.

[Sias04] J. Sias and al., “Field-testing impact EPIC research results in Itanium 2”, Proc. of the 31th annual international symposium on Computer architecture, June 2004.

[Sic98] G. Sicard, “De la biologie au silicium : une rétine bio-inspirée analogique pour un capteur de vision intelligent et adaptatif”, PhD thesis, Institut Polytechnique de Grenoble, 1998.

[Smel03] M. Smelyansky and al., “Predicate-aware scheduling: A technique for reducing resource constraints”, Proc. of the 1st International Symposium on Code Generation and Optimization, March, 2003.

[Smi97] J. Smit, “Energy complexity and architecture”, 7th International workshop on Power, Modelling, Optimisation, Simulation (PATMOS), Louvain La Neuve, Sep. 1997.

[Synopsys] *http://www.synopsys.com*

T

[Ten04] Tendance, CAO Electronique, “Les flots de FPGA complexes se structures”, n°143, Electronique, Jan. 2004.

[Tiger] “Genearal Purpose TigerSHARC Processor”, *www.analog.com/tigersharc*.

[Tri] *http://www.trimaran*.org/docs/5_hpl-pd.ps.

[Trimedia] J. T. Van Eijndhoven, F.W. Sijstermans and *al.*, “Trimedia CPU64 Architecture”, Phillips Research Laboratories and University of Amsterdam.

W

[Wan09] Minghua Wan, Zhihui Lai, Jie Shao, and Zhong Jin, “Two-dimensional local graph embedding discriminant analysis (2dlgeda) with its application to face and palm biometrics”, Neurocomputing, 73(1-3) :197 – 203, 2009.

X

[Xilinx] *http://www.xilinx.com*.

[XSG] Xilinx, System-Generator - User Manual, http://www.xilinx.com.

Y

[Yan08] Yan Yan and Yu-Jin Zhang, “Discriminant projection embedding for face and palmprint recognition”, Neurocomputing, 71(16-18) :3534 – 3543, 2008.

[Yang03] F. Yang, M. Paindavoine, “Implementation of a RBF neural network on embedded systems : Real time face tracking and identity verification”, IEEE Transactions on Neural Networks, Vol.14, No.5, pp.1162-1175, 2003.

[Yang06] Shenglin Yang, Kazuo Sakiyama, and Ingrid Verbauwhede, “Efficient and securefingerprint verification for embedded devices”, Eurasip Journal on Applied Signal Processing, pages 1–11, 2006.

[Yao07] Yong-Fang Yao, Xiao-Yuan Jing, and Hau-San Wong, “Face and palmprint feature level fusion for single sample biometrics recognition”, Neurocomputing, 70(7-9) :1582 – 1586, 2007.

[Yoo07] Jang-Hee Yoo, Jong-Gook Ko, Yun-Su Chung, Sung-Uk Jung, Ki-Hyun Kim, Ki-Young Moon, and Kyoil Chung, “Design of embedded multimodal biometric systems”, In SITIS ’07 : Proceedings of the 2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, pages 1058–1062, Washington, DC, USA, 2007. IEEE Computer Society.

[You02] Jane You, Wenxin Li, and David Zhang, “Hierarchical palmprint identification via multiple feature extraction”, Pattern Recognition, 35(4) :847 – 859, 2002.

Z

[Zhang03] David Zhang, Wai-Kin Kong, Jane You, and Michael Wong, “Online palmprint identification”, IEEE Trans. Pattern Anal. Mach. Intell., 25(9) :1041–1050, 2003.

[Zuo10] Wangmeng Zuo, Hongzhi Zhang, David Zhang, and Kuanquan Wang, “Postprocessed lda for face and palmprint recognition : What is the rationale”, Signal Processing, 90(8) :2344 – 2352, 2010.

Liste des publications

Revue (2)

- ***"Rapid design of parallel reconfigurable instruction set processor for efficient embedded image processing"***, V. Brost, F. Yang, C. Meunier, D. Saptono and A. Poinsot, Journal Sensors soumission en 2011.
- ***"VLIW architecture compilation-simulation and its implementation into FPGA"***, D. Saptono, V. Brost and F. Yang, Journal of international Review on Computer and Software, Italy, in press, 2011.

Conférences internationales (5)

- "***Design Space Exploration for a custom VLIW Architecture : Direct photo printer hardware setting using VEX Compiler***", D. Saptono, V. Brost, F. Yang, and E. Prasetyo, in 4th International Conference of SITIS, Bali, Indonesia, 2008.
- "***Development of Uclinux Platform for Computer Vision Algorithm in FPGA Devices***", D. Saptono and E. Prasetyo, The first International Workshop on Open Source and Open Content (WOSOC), Bali, Indonesia, 2008.
- "**Concept and Development of Modular VLIW Processor Based on FPGA**", D. Saptono, V. Brost, F. Yang, and E. Prasetyo, The 2th International conference of ICSEM, Bangkok, 2010.
- "***Design Space Exploration for Sobel Application using OpenIMPACT (Opensource Retargetable Compilation for VLIW Architecture)***", Debyo SAPTONO, Vincent BROST, Fan YANG, The second International Workshop on Open Source and Open Content (WOSOC), Bali, Indonesia, 2010.
- ***"Flexible VLIW processor based on FPGA for real-time image processing",*** V. Brost, C. Meunier, D. Saptono and F. Yang, International Conference of DASIP, 4-7 Nov. Finland 2011.

Conférences nationales (1)

- " ***Modular VLIW processor based on FPGA for real-time image processing*** ", V. Brost, D. Saptono, C. Meunier, F. Yang and D. Ginhac, Gretsi 2011, Bordeaux, France, 5-8 Septembre 2011.

Résumé

Ce manuscrit présente les travaux menés pour proposer un flot de conception permettant d'implanter des processeurs RISP dans un circuit reprogrammable (FPGA). Après une description des différentes solutions envisageables pour réaliser des prototypes dans le domaine du traitement d'image, ce document décrit une méthode qui consiste à générer des modèles matériels de processeurs destinés au traitement d'images, avec des opérateurs taillés sur une application donnée. Un ensemble d'expérimentations utilisant des algorithmes courants permet d'évaluer les performances du flot de conception proposé. Le prototypage rapide d'un système biométrique sans contact, basé sur la reconnaissance de paumes a été aussi réalisé sur la plateforme de test.

Abstract

This manuscript presents work to propose a development cycle to establish RISP processors in a reprogrammable chip (FPGA). After a description of the various possible solutions to produce image processing prototypes, this document describes a method which consists in generating hardware models of processor target to image processing, with operators just for a given application. Test with a set of common algorithm makes evaluate the performances of the design cycle proposed. Rapid prototyping of a contact less biometric system, based on palmprint recognition, is also realized on the test platform.