Optimisation de JPEG2000 sur système sur puce programmable

Imed Aouadi

Avant-propos

Le travail présenté dans cette thèse a été réalisé au sein du Laboratoire Electronique et Informatique (LEI) de l'Ecole Nationale Supérieure de Techniques Avancées (ENSTA de Paris). Je remercie Monsieur Alain SIBILLE, responsable du laboratoire LEI à l'ENSTA de m'avoir accueilli et donné les moyens pour effectuer mon travail de recherche.

Je tiens particulièrement à exprimer ma profonde gratitude à Monsieur Omar HAMMAMI, enseignant chercheur à l'ENSTA, pour son encadrement et son suivi. Son soutien, ses conseils, ses encouragements permanents tout au long de ces années de recherche, son dynamisme et sa disponibilité ont joué un rôle déterminant dans ce travail. Qu'il trouve ici, l'expression de ma profonde reconnaissance.

Je remercie vivement, Monsieur Dominique HOUZET maître de conférence à l'INSA de Rennes, et Monsieur Lionel TORRES professeur à l'université de Montpellier II, rapporteurs de cette thèse, pour leurs excellents commentaires et leurs remarques judicieuses qui ont fortement contribués au manuscrit.

Je remercie également Monsieur Mohamed AKIL professeur à l'ESIEE, Monsieur Habib MEHREZ professeur à l'université de Jussieu, et Monsieur Alain MERIGOT professeur à l'université de Paris XI, pour avoir accepté de participer au jury.

Merci à tous les membres du laboratoire LEI de l'ENSTA. Qu'ils trouvent ici l'expression de ma reconnaissance pour leur témoignage de sympathie et d'amitié.

Enfin merci à ma famille et mes amis, pour m'avoir accompagné au cours de toutes ces années.

SOMMAIRE

Liste des figures

Chapitre I

Chapitre II

Chapitre III

Chapitre IV

Chapitre V

Chapitre VI

Chapitre VII

Chapitre VIII

Liste des tableaux

Chapitre II

Chapitre III

Chapitre IV

Chapitre V

Chapitre VI

Chapitre VII

Résumé

Récemment le domaine du traitement de l'image, de la vidéo, et l'audio a connu plusieurs évolutions importantes au niveau des algorithmes et des architectures. L'une de ces évolutions est l'apparition du nouveau standard ISO/IEC de compression d'image JPEG2000 qui succède à JPEG. Ce nouveau standard présente de nombreuses fonctionnalités et caractéristiques qui lui permettent d'être adapté à une large panoplie d'applications. Mais ces caractéristiques se sont accompagnées d'une complexité algorithmique beaucoup plus élevée que JPEG et qui le rend très difficile à optimiser pour certaines implémentations ayant des contraintes très sévères en terme de surface, de temps d'exécution ou de consommation d'énergie ou de l'ensemble de ces contraintes. L'une des étapes clé dans le processus de compression JPEG2000 est le codeur entropique qui constitue à lui seul environ 70% du temps de traitement global pour la compression d'une image. Il est donc essentiel d'analyser les possibilités d'optimisation d'implémentations de JPEG2000.

Les circuits FPGA sont aujourd'hui les principaux circuits reconfigurables disponibles sur le marché. S'ils ont longtemps été utilisés uniquement pour le prototypage des ASIC, ils sont aujourd'hui en mesure de fournir une solution efficace à la réalisation matérielle d'applications dans de nombreux domaines. Vu le progrès que connaît l'industrie des composants FPGA du point de vue capacité d'intégration et fréquence de fonctionnement, les architectures reconfigurables constituent aujourd'hui une solution efficace et compétitive pour répondre aussi bien aux besoins du prototypage qu'à ceux des implémentations matérielles.

Dans ce travail nous proposons une démarche pour l'étude des possibilités d'implémentations de JPEG2000. Cette étude a débuté avec l'évaluation d'implémentations logicielles sur plateformes commerciales. Des optimisations logicielles ont été ajoutées en utilisant des librairies SIMD spécialisées exploitant du parallélisme à grain fin. Suite à cette première étude on a réalisé une implémentation matérielle d'un bi codeur entropique sur FPGA qui a servi comme coprocesseur pour deux plateformes distinctes l'une étant une machine hôte et l'autre un système industriel embarqué. Suite à cette étape nous avons fait évoluer l'implémentation en passant à une deuxième approche qui est l'approche système sur puce programmable. Dans cette dernière partie nous avons effectué le partitionnement matériel/logiciel du codeur entropique sur FPGA, puis une implémentation multi codeur a été réalisée sur FPGA et utilisée comme coprocesseur sur puce pour la création d'un système sur puce programmable. Ces différents travaux ont permis de couvrir une partie de l'espace des applications que JPEG2000 peut cibler. En même temps ces implémentations donnent une vue globale sur les possibilités des implémentations de JPEG2000 ainsi que leurs limites. De plus cette étude représente un moyen pour décider de l'adéquation architecture application de JPEG2000.

***Mots clés** : codeur entropique, FPGA, JPEG2000, optimisation, partitionnement logiciel/matériel, système sur puce programmable (SOPC).*

Abstract

Recently the field of video, image and audio processing has experienced several significant progresses on both the algorithms and the architectures levels. One of these evolutions is the emergence of the new ISO/IEC JPEG2000 image compression standard which succeeds to JPEG. This new standard presents many functionalities and features which allows it to be adapted to a large spectrum of applications. However, these features bring up new algorithmic complexities of higher degree than those of JPEG which in turn makes it very difficult to be optimized for certain implementations under very hard constraints. Those constraints could be area, timing or power constraints or more likely all of them. One of the key steps during the JPEG2000 processing is entropy coding that takes about 70 % of the total execution time when compressing an image. It is therefore essential to analyze the potentialities of optimizations of implementations of JPEG2000.

FPGA devices are currently the main reconfigurable circuits available on the market. Although they have been used for a long time only for ASIC prototyping, they are able today to provide an effective solution to the hardware implementation of applications in many fields. Considering the progress experienced by the FPGA semiconductor industry on integration capacity and working frequency, reconfigurable architectures are now an effective and competitive solution to meet the needs of both prototyping and final hardware implementations.

In this work we propose a methodology for the study of the possibilities of implementation of JPEG2000. This study starts with the evaluation of software implementations on commercial platforms, and quickly extended through software optimizations based on specialized SIMD libraries exploiting fine grain concurrency. Following this first study we carried out a hardware implementation of an entropic dual-coder on FPGA which was used as a coprocessor on both a host machine and on an embedded industrial platform. After this implementation we evolved our approach to a system approach. In this last part we carried out hardware/software partitioning of the entropic coder on FPGA, then a multi-coder implementation was realized on FPGA and used like coprocessor on chip for the creation of a system on programmable chip. These various works allowed us to cover a large part of the applications space that JPEG2000 can target. At the same time these implementations give a global vision on the possibilities and limits of the implementations of JPEG2000. Furthermore this study is a support to decide architecture-application mapping for JPEG2000 implementation.

Key words*: entropy coder, FPGA, hardware/software partitioning, JPEG2000, optimization, system on programmable chip (SOPC)*

Chapitre I Introduction

Récemment les algorithmes et les architectures pour la compression des images ont connus une évolution importante. Cette évolution s'est effectuée selon deux principales directions. Du point de vue algorithmique de nouvelles techniques permettent le développement de méthodes plus robustes pour la réduction de la taille des données. Ces méthodes sont extrêmement vitales pour un grand nombre d'application qui manipule des données numériques.

Du point de vue architecture il est maintenant possible de mettre un procédé de compression sophistiqué sur une seule puce relativement peu coûteuse, ce qui a stimulé le développement des systèmes multimédia.

Dans le cas de la compression sans perte (lossless) les compromis liés à ce mode de compression sont selon trois axes :

Efficacité du codage, ceci peut être mesuré en bit par pixel (échantillon), elle est limitée par l'entropie de la source. Plus l'entropie de la source est grande plus il est difficile a compresser (exemple un bruit aléatoire).

Temps de codage, celui-ci est lié à la complexité du processus de codage ou de décodage. Il peut être réduit si on augmente la capacité de calcul du composant de traitement. Pour certaines applications ce temps est contraint ce qui impose le choix de la technique de codage.

Complexité du codeur : elle peut être mesurée à l'aide de la quantité de ressources utilisées en terme de mémoire et du nombre d'opérations arithmétiques. Le nombre d'opérations est donné par MOPS (million of operations per second). Le MIPS (Millions of Instruction Per Second) est parfois utilisé. Si on ajoute l'aspect mobile de l'application, la consommation d'énergie peut être considérée comme caractéristique de la complexité de codage.

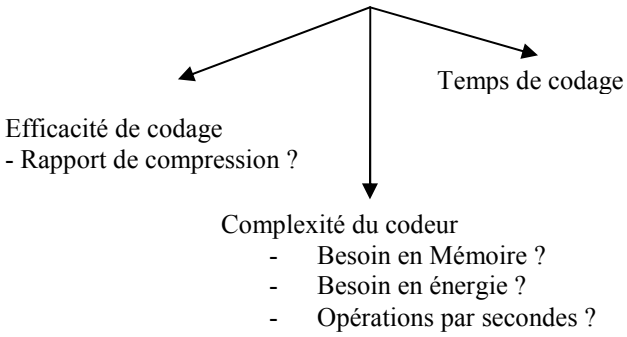


Figure Ch I - 1 : Les compromis en mode de codage sans perte

Pour la compression avec perte (Lossy) en plus des trois axes de compromis pour la compression sans perte, on ajoute un quatrième axe qui est la qualité du signal : il est utilisé pour caractériser le signal à la sortie du décodeur. Plusieurs mesures sont proposées pour la qualité du signal parmi lesquelles le SNR (Signal-to-Noise Ratio), le PSNR (Peak-SNR), et le MSE (Mean Square Error).

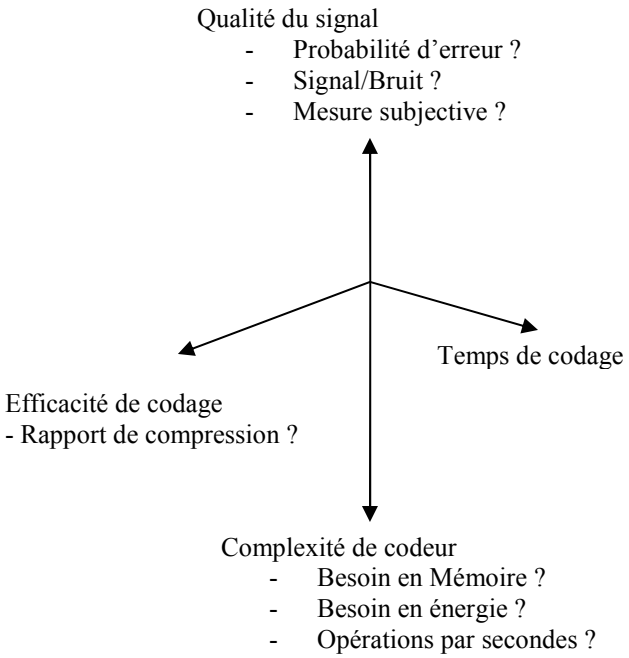


Figure Ch I - 2 : Les compromis en mode de codage avec perte

Parmi les techniques de compression des images on trouve JPEG, JBIG, JPEG-LS, JPEG2000 qui sont issues des comités de standardisation « Joint Photographic Experts Group » et « Joint Bi-level Image experts Group » et MPEG-4-Visual Texture Coding (VTC) réalisé par « Moving Picture Experts Group ». Chacun de ces standards possède des caractéristiques algorithmiques permettant d'obtenir plus au moins de meilleurs compromis **[1], [2], [3]**

Le plus populaire de ces standards est le JPEG il a été créé vers la fin des années 80. JPEG utilise les principaux modes suivants : baseline, lossless, progressive et hiérarchique. Le mode baseline est le plus utilisé, il supporte le codage avec perte seulement. Le mode lossless est moins populaire il ne supporte pas le mode avec perte. Dans le mode baseline l'image est divisée en bloc de 8x8 pixels et chacun de ces blocs est transformé avec une DCT (Discret Cosine Transform). Les échantillons du bloc transformé sont quantifiés avec une quantification uniforme, puis parcourus en zigzag ensuite on leur applique un codage entropique avec un codeur de Hoffmann. Le pas de quantification est spécifié dans un tableau de référence qui est le même pour tous les blocs. Les coefficients de la DCT des différents blocs sont codés séparément en utilisant un schéma prédictif.

Le mode lossless est basé sur un algorithme complètement différent qui utilise un schéma prédictif. La prédiction est basée sur les trois voisins les plus proches (nearest three causal neighbors) ainsi sept différents prédicteurs sont définis. L'erreur de prédiction est codée avec le codeur de Hoffmann. Ce mode est dit L-JPEG. Les modes progressif et hiérarchique sont tous les deux avec perte ils diffèrent respectivement par le codage et le calcul des coefficients de la DCT. Ils permettent une reconstruction d'image avec respectivement une moindre qualité et une faible résolution en décodant partiellement le bitstream. Le mode progressif code les coefficients quantifiés avec un mélange de sélection spectral et d'approximation successive, alors que le mode hiérarchique utilise une approche pyramidale multi résolution pour le calcul des coefficients de la DCT.

JBIG a été développé pour la compression sans perte des images binaires (bi-level). Il peut aussi coder des images en niveau de gris ou en couleur avec un nombre limité de bit par pixel. Il est utilisé essentiellement pour les fax. L'évolution de JBIG dit aussi JBIG1 est le JBIG2

qui est venu améliorer les performances de ces prédécesseurs pour la compression des images binaires. Il supporte aussi bien la compression avec et sans perte contrairement au JBIG1 qui ne supporte que la compression sans perte. La capacité de compression a été multipliée par un facteur allant de 3 à 10. Il supporte, aussi, la compression d'un document multi-pages.

JPEG-LS est parmi les derniers standards de compression d'image fixe. Il prévoit une compression « Quasi-sans perte » (near lossless). Dans sa Partie I, le système baseline, utilise la prédiction adaptative, la modélisation du contexte, et le codage de Golomb. En plus, il comporte un détecteur de régions plates pour les codées dans le mode run-lenght. La compression « Quasi-sans perte » est obtenue en fixant l'erreur maximale permise pour les échantillons. La Partie II présente des extensions telles que le codeur arithmétique. Cet algorithme a été conçu pour avoir une faible complexité tout en fournissant des taux de compression élevés. Cependant, il ne permet ni la scalabilité, ni la correction d'erreur, ni toute autre fonctionnalité semblable.

Le MPEG-4 VTC est l'algorithme utilisé dans le MPEG-4 pour la compression de la texture visuelle et les images fixes. Il est basé sur la DWT (Discret Wavelet Transform), la quantification scalaire, le codage zero-tree et le codage arithmétique. La DWT utilise le filtre 9/3 de Daubechies. La quantification peut être de trois types : simple (SQ), multiple (MQ) ou binaire (bi-level BQ). En SQ chaque coefficient de la DWT est quantifié une seule fois. En MQ une première quantification avec un grand pas est appliqué puis les informations issues sont codées, ensuite une deuxième quantification plus fine est appliquée aux erreurs et les nouvelles informations sont ensuite codées. Ce processus peut être répété plusieurs fois. Le BQ est semblable au SQ mais les informations sont fournies par bit-plane. Deux modes de balayage sont utilisés : le tree-depth (TD) et le bande-par-bande (BB). Seulement le dernier prévoit la scalabilité de la résolution. L'aspect unique du MPEG-4-VTC est sa capacité de coder les objets à formes arbitraires. Le MPEG-4 VTC ne supporte pas le mode de codage sans perte.

JPEG2000 est le dernier standard ISO/ITU-T pour le codage des images fixes. Notre description sera réduite à celle de la partie I de ce standard qui représente le cœur du système. JPEG2000 est basé sur la DWT, la quantification scalaire, la modélisation du contexte, le codage arithmétique et l'allocation débit post-compression. Deux types de filtre sont utilisés pour la DWT : le filtre Le Gall 5/3 pour le codage sans perte et le filtre de Daubechies 9/7 pour le codage avec perte. La quantification suit une approche scalaire de dead-zone et elle est indépendante pour chaque sous-bande. Chaque sous-bande est partagée en code-blocks qui seront codés par le modélisateur de contexte et le codeur arithmétique. Les données codées sont ensuite organisées en couches, qui représentent les niveaux de qualité, puis dans des paquets qui formeront le code-stream de sortie. Plusieurs fonctionnalités sont disponibles telles que la reconstruction progressive par qualité et/ou par résolution, l'accès aléatoire au code-stream, et le pouvoir de coder différemment une région de l'image dite ROI (Region Of Interest) (voir le Chapitre suivant)

Après cet exposé des différents standards de codage d'image on peut dire qu'il sont en général des combinaisons d'un ensemble de techniques avec plus au moins une adaptation pour un certains types de données ou d'applications. Chacune de ces techniques permet de réaliser un meilleur compromis selon un ou plusieurs axes tel qu'il est présenté dans les **figures Ch I -1** et **Ch I -2**. Ces techniques sont plus au moins complexes et coûteuses en temps de calcul avec une variation de la qualité de la compression selon la complexité de ces techniques. JPEG2000 est l'un des standards qui regroupe les techniques de codage les plus complexes telles que la DWT, le découpage en code-blocks, la modélisation de contexte, le codage arithmétique, mais cela se traduit par une meilleure qualité de résultat avec une grande variété des fonctionnalités supportées par le standard. Pour ces différents standards le problème le

plus difficile est la détermination de la complexité du codeur qui dépend en général de l'application. Il peut être le nombre de cycle, la quantité de la mémoire, la bande passante de la mémoire, le nombre de portes logiques. Pour pouvoir comparer ces différents standards on utilise généralement une mesure de performance d'une implémentation logicielle sur un PC. Ceci nous permettra seulement d'avoir une idée de la complexité de chacun de ces standards et non pas une mesure précise.

En combinant les techniques de codages citées précédemment JPEG2000 présente de nouveaux défis pour être implémenté et optimisé. Vu les possibilités offertes par JPEG2000, il est potentiellement candidat pour être adopté pour un grand nombre d'applications telles que l'imagerie médicale, le cinéma numérique, la surveillance, les satellites, les applications mobiles (appareil photo, PDA, téléphone cellulaire,…). Ces différentes applications présentent plusieurs exigences et contraintes : le temps, la qualité, ou la consommation d'énergie,…. Vu cette variété d'applications et d'exigences, trouver une implémentation et une architecture pouvant satisfaire l'ensemble de ces contraintes se révèlent très difficile. Quelle implémentation et quelle architecture choisir ? Bien que la réutilisation des algorithmes et des techniques de codages soit le principe même de l'évolution des capacités de la scalabilité et de la qualité du codage des standards, ceci n'est en rien le cas pour l'implémentation de chaque standard.

Comme JPEG2000 est récemment sorti, les possibilités d'intégration dans les différentes applications ne sont pas encore suffisamment étudiées. Seulement quelques modèles de vérification et des implémentations logicielles sont disponibles. Les implémentations matérielles sont quasi inexistantes seulement quelques unes qui ne concernent qu'une partie du codeur (généralement la DWT) sont disponibles. Le travail de cette thèse a été mené dans le cadre de cette situation problématique pour l'application et l'exploitation des avantages de JEPG2000 dans le maximum de catégories d'applications.

Cette thèse se déroule dans le cadre du projet **E**tudes d'opt**I**misation algo**R**ithmiques de JP**E**G2000 (EIRE) qui est soutenu par le ministère de l'industrie dans le cadre du Réseau National de la Recherche en Télécommunications (RNRT). Ce projet a débuté en novembre 2001 et s'est terminé en février 2004. L'objectif du projet est de proposer des améliorations au schéma de base de JPEG2000, de réduire la complexité algorithmique, d'améliorer la qualité tant de l'encodeur que du décodeur, et d'assurer une utilisation optimale de JPEG2000, y compris dans un contexte d'interopérabilité avec d'autres schémas de codage.

Les travaux réalisés dans le cadre du projet permettront de fournir la meilleure qualité de service dans un contexte limité en ressources (bande passante, capacité de traitement…), et de démontrer la possibilité d'implémenter efficacement JPEG2000. Les applications visées par le projet seront principalement l'imagerie de type scientifique (imagerie satellite ou médicale) et la télésurveillance de par les contraintes quelles impliquent en terme de qualité et de capacité de traitement. A terme les techniques développées dans le projet pourront être utilisées dans les futurs produits et services suivants: imagerie et vidéo sur mobiles (UMTS en particulier), compression d'image à bord de satellites, télémédecine, mobiles téléguidés, restauration et transcodage de banques d'images compressées et imagerie géographique. Les principaux axes du projet sont:

- Optimisation qualitative tant au niveau de l'encodeur que du décodeur, incluant des méthodes de décodage et transcodage optimal.
- Adaptation des outils de compression JPEG2000 à du codage vidéo compatible avec les contraintes de transmission sur internet.

- Evaluation qualitative de chaque méthode proposée de façon à se placer dans un contexte réaliste et reproductible permettant de mesurer l'apport du projet par rapport au schéma de base du standard.
- Réduction de la complexité algorithmique de l'encodeur mais aussi du décodeur et étude d'une l'implantation matérielle efficace.

Comme les mesures de complexité de JPEG2000 ont montré que les parties les plus complexes sont le codage entropique qui est constitué par la modélisation de contexte, le codage arithmétique, et la DWT **[4], [5]**. Comme la DWT a été utilisée dans des standard antérieurs à JPEG2000, plusieurs études ont été menées dans le but d'optimiser son implémentation et de réduire sa complexité donc il reste à étudier le codeur entropique qui à lui seul peut dépasser 70% du temps totale du codage d'une image. L'idée derrière ce travail et de réduire le temps total de codage en réduisant celui du codeur entropique. Pour les raisons citées précédemment l'effort a été concentré, dans le cadre de cette thèse, sur l'étude de l'implémentation de cette partie du codeur JPEG2000.

Cette thèse est organisée comme suit :

Le chapitre II donne un aperçu de la partie I du standard JPEG2000 dans lequel on expliquera les principales techniques de codage : la transformation de couleur, la DWT, la quantification, le codage entropique, et la reconstruction du fichier JPEG2000. On limitera notre description à la partie encodeur, puisque les étapes de la partie décodage sont identiques en considérant la fonction inverse de chaque transformation. Une comparaison des architectures qui sont apparues dans la littérature au cours de la thèse est présentée dans la dernière partie de ce chapitre.

Dans le chapitre III on expose les différentes technologies des circuits reconfigurable du type FPGA. Ensuite on donne l'état de l'art des méthodologies et techniques de conception et de conception conjointe pour les différentes technologies (ASIC, FPGA).

Dans le chapitre IV sont regroupées les premières études de la complexité du codeur JPEG2000 sur différentes plateformes. La première étude concerne les plateformes Intel Pentium II, III, et 4 sur des PC avec différentes configurations. Cette étude a été suivie par une première tentative d'optimisation en utilisant des librairies spécialisées optimisées pour les processeurs Pentium. La seconde étude a été effectuée sur la famille des processeurs TriMedia de Philips qui ont une architecture du type VLIW. La dernière plateforme utilisée est basée sur le DSP TMS320C6711 de Texas Instruments.

Le chapitre V décrit l'utilisation d'une plateforme reconfigurable à base de composant FPGA pour implémenter le coprocesseur du codeur entropique en version parallèle. Dans ce chapitre on présente deux type de plateformes : une plateforme du type PCI utilisée sur un PC, et une plateforme du type indépendant (standalone) disposant de son propre environnement composé d'un processeur PowerPC et de l'OS VxWorks et d'un FPGA connecté au PowerPC.

Le chapitre VI présente la vision système de l'implémentation du codeur entropique. Ce chapitre est partagé en deux parties : dans la première partie on étudie le cas du partitionnement logiciel/matériel avec une mise en œuvre sur un système sur puce programmable du codeur entropique. La deuxième partie décrit l'utilisation de plusieurs codeurs entropiques comme coprocesseurs sur un système sur puce programmable.

Le chapitre VII est une étude de la possibilité de s'affranchir des plateformes logicielles et matérielles, et de leurs contraintes. Cette étude est basée sur les Réseaux de Petri. Elle représente une vue plus abstraite des problèmes de conception et d'implémentation d'un algorithme ayant une complexité élevée et nécessite un grand effort d'optimisation.

Le chapitre VIII est le dernier chapitre. C'est en même temps la conclusion de ce travail, et l'ensemble des perspectives qui peuvent être les sujets d'autres études.

Références

[1] Vasudey Bhaskaran, Konstantinos Konstantinides, Image and video compression standards algorithms and architectures Kluwer academic publishers ISBN 0-7923-9591-3, third printing 1996

[2] Diego Santa-Cruz and Touradj Ebrahimi, "An analytical study of JPEG 2000 functionalities". In Proc. of the International Conference on Image Processing (ICIP), vol. 2, pp. 49-52, Vancouver, Canada, September 10-13, 2000.

[3] Diego Santa-Cruz, Raphaël Grosbois and Touradj Ebrahimi. JPEG 2000 performance evaluation and assessment. Signal Processing: Image Communication, pages 113-130, volume 17, no. 1, Jan. 2002.

[4] Kuan-Fu Chen; Chung-Jr Lian; Hong-Hui Chen; Liang-Gee Chen, Analysis and Architecture Design of EBCOT for JPEG 2000 IEEE, International Symposium on Circuits and Systems, 2001. ISCAS 2001. The 2001, Volume: 2, 6-9 May 2001

[5] M.D. Adams and F. Kossentini, "JasPer : a Software-Based JPEG2000 Codec Implementation", IEEE ICIP-2000, Vol. 2, pp. 53-56, Sep. 2000

Chapitre II Le standard JPEG2000

1. Introduction à JPEG2000

JPEG2000 est un nouveau standard ISO et ITU pour la compression des images fixes (ISO/IEC 15444-1 et ITU-T T.800). JPEG2000 est la seule norme de compression d'images à couvrir avec un seul système l'ensemble des besoins de compression d'images avec ou sans pertes.

En plus de fournir une meilleure performance en terme de taux de compression à qualité équivalente, JPEG2000 permet d'accéder à différentes représentations des images (résolution spatiale, région d'intérêt…), et permet de s'adapter aux conditions d'utilisation (capacité de traitement du terminal réalisant le décodage, capacité du canal de transmission…). L'ensemble des données nécessaires étant contenues dans un fichier compressé unique. Cela est rendu possible grâce aux modes de codage de progressivité en résolution et en qualité ainsi qu'avec le concept de région d'intérêt. Un seul module de compression peut fournir plusieurs représentations de façon simultanée. Une seule fonction de compression peut fournir les deux représentations (pleine résolution, résolution intermédiaire, qualité intermédiaire) de façon simultanée, en utilisant un mode particulier du codage, appelé « progressivité ». Cela permet de coder dans le même flux de données les deux représentations, la plus simple en terme de résolution étant utilisée directement pour être émise et servant aussi d'information de base pour, en complément de l'information d'enrichissement, former les images de pleine résolution.

JPEG2000 est adapté à tous les types d'images: photographies, images scientifiques, clichés médicaux, vues de télésurveillance, et cela avec des applications allant de la transmission d'images sur internet à la photographie numérique en passant par l'impression, la télécopie, l'imagerie médicale et satellite ainsi que les télécommunications mobiles. De plus, JPEG2000 est prévu pour fonctionner dans un contexte de transmission en environnement bruité (marqueurs de resynchronisation pour lutter contre les pertes de paquets), caractéristique fondamentale pour le domaine des communications mobiles.

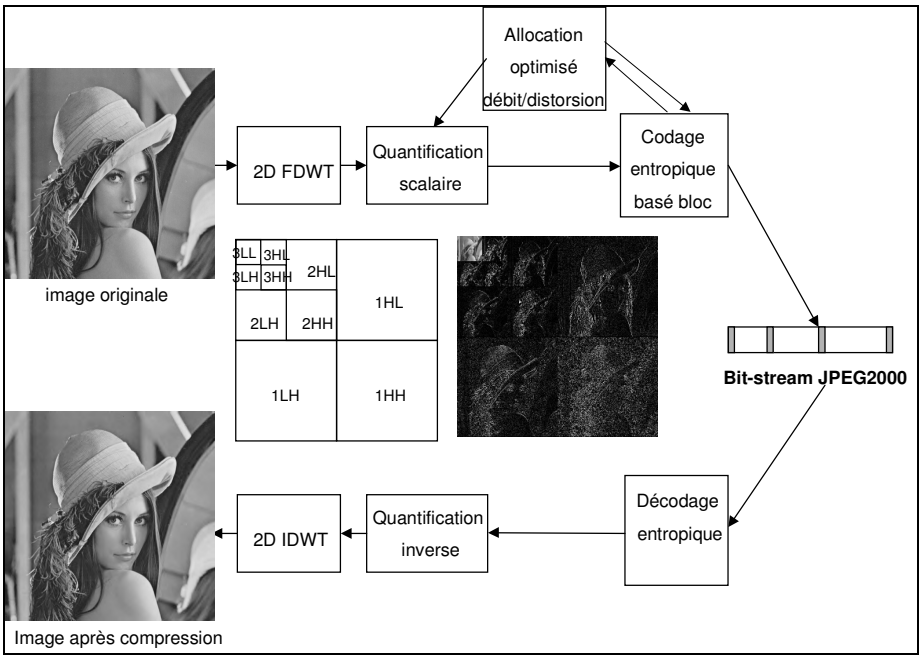


Figure Ch II - 1 : Constituants de JPEG2000

Les caractéristiques principales de JPEG2000 sont les suivantes:

- Progressivité en qualité ou en résolution,

- Compression avec ou sans perte dans le même algorithme,
- Accès aléatoire aux données,
- Décompression partielle des données,
- Grande taille d'images,
- Précision élevée (>=8bits).

Même si les blocs généraux du diagramme ressemblent à ceux du standard JPEG conventionnel, il y a des différences radicales dans tous les processus de chaque bloc du diagramme. D'une manière simplifiée le système est décrit de la façon suivante:

- si l'image source est multicomposante, chaque composante sera traitée de manière indépendante. Par exemple, une image couleur RVB sera décomposée en une image R, une image V et une image B.
- les images sont découpées (optionnel) en imagettes rectangulaires dites *tile*. La composante imagette est l'unité de base de l'image originale ou reconstruite.
- une transformation en ondelettes est appliquée sur chaque imagette. Une imagette est décomposée en différents niveaux de résolution.
- les niveaux de décomposition sont formés de sous-bandes des coefficients qui décrivent les caractéristiques de fréquence des zones locales des composantes imagettes.
- les coefficients des sous-bandes sont quantifiés et classés dans des blocs de coefficients (code-blocks).
- les bit-planes des coefficients dans un code-block sont codés entropiquement.
- le codage peut être réalisé de telle manière que certaines régions d'intérêt peuvent être codées à une meilleure qualité que le fond de l'image.
- des marques sont ajoutées au flux de bits (*bitstream*) pour permettre la détection d'erreurs.
- le flux de bits possède un en-tête principal qui décrit l'image originale et les différents modèles de décomposition et codage utilisés pour localiser, extraire, décoder et reconstruire l'image avec la résolution désirée, la fidélité souhaitée, les régions d'intérêt et autres caractéristiques.

La figure suivante montre l'ensemble des modules algorithmiques (optionnels pour certains d'entre eux) :

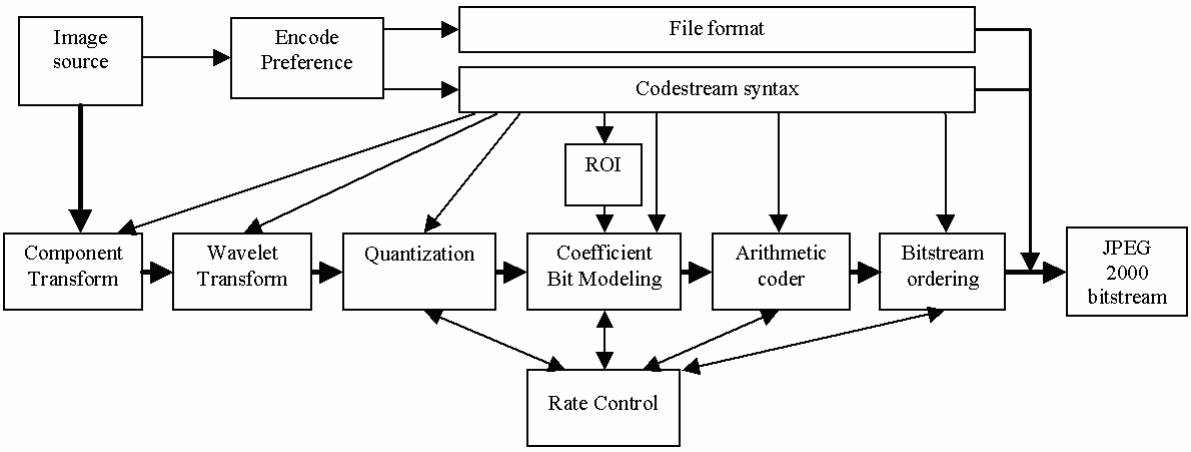


Figure Ch II - 2 : Diagramme de l'encodeur JPEG2000

1.1 Modèle d'image utilisé par JPEG2000

Avant de présenter en détail le mécanisme de codage utilisé dans JPEG2000, il est nécessaire de bien comprendre le modèle d'image qui est utilisé. Une image est composée de plusieurs composantes (limitées à 2^{14}). Chaque composante est constituée d'une matrice rectangulaire d'échantillons. Les valeurs des échantillons de chaque composante sont des valeurs entières (avec ou sans signe) de précision pouvant aller de 1 à 38 bits/échantillons.

Le signe et la précision des échantillons sont spécifiés individuellement par composante. Ces composantes peuvent être utilisées pour représenter les informations spectrales de l'image originale. Par exemple, une image en couleur RGB est composée de 3 composantes : une pour représenter la couleur rouge, une autre pour la couleur verte, et une autre pour la couleur bleue. Dans le cas simple d'une image à nuance de gris, on ne retrouve qu'une seule composante correspondant à l'information de luminance de l'image. Les composantes entre elles peuvent avoir des tailles différentes. Par exemple lorsque l'on représente une image constituée des informations de luminance et de chrominance, les échantillons de luminance sont souvent plus finement échantillonnés que les échantillons de chrominance. De plus JPEG2000 gère le fait que les instants d'échantillonnage puissent être non coïncidents les uns avec les autres, comme c'est le cas pour des images vidéo de type 4 :2 :0 par exemple.

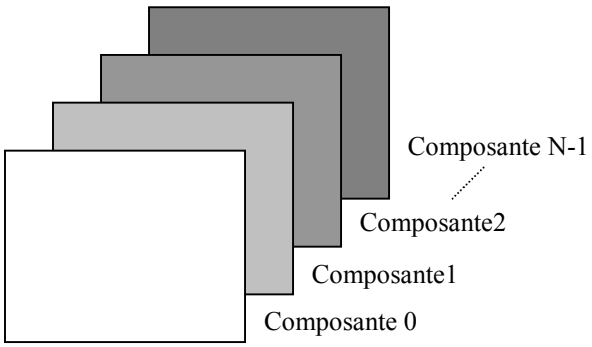


Figure Ch II - 3 : Représentation d'une image composée de N composantes numérotées de 0 à N-1.

L'image est repérée selon un système de coordonnées basé sur une grille rectangulaire appelée grille de référence, de taille *Xsiz * Ysiz*. La largeur et la hauteur de cette grille de référence ont été limitées à 2^{32}-1 points, pour limiter la taille maximale de l'image à coder. Cette limite impose la taille maximale d'une image qu'un codeur peut traiter. L'image est repérée dans cette grille par son point supérieur gauche de coordonnées (X0siz, Y0siz), comme le montre la figure suivante.

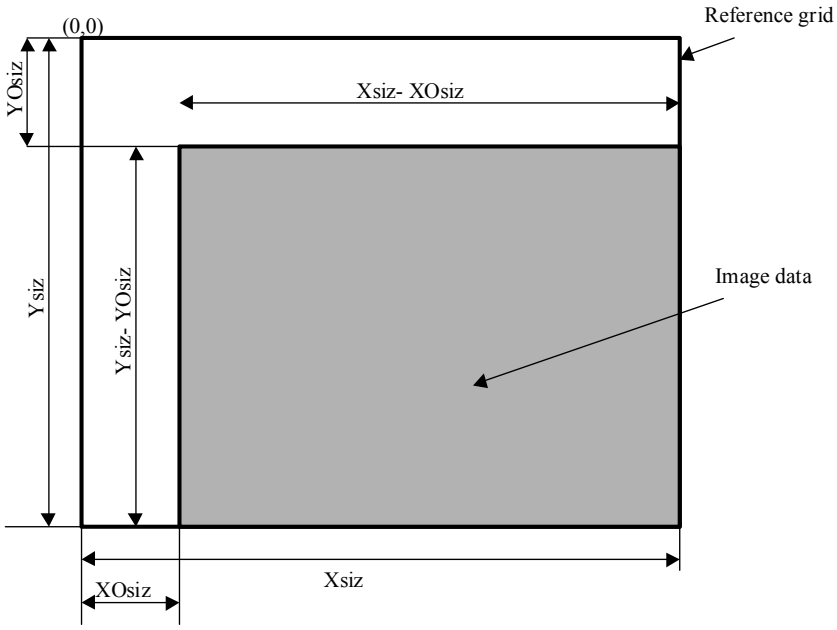


Figure Ch II - 4 : Représentation de la grille de référence

L'aire située entre les points (XO_{siz}, $YOsiz$) et (X_{siz}-1, $Ysiz$-1) (zone ombrée de la **figure Ch II - 4**) correspond à la zone image. Toutes les composantes de l'image sont projetées sur la zone image de cette grille de référence. Puisque ces composantes ne sont pas toutes échantillonnées suivant la même résolution de la grille de référence, des informations supplémentaires sont nécessaires dans le but d'établir cette projection.

Pour chaque composante, on indique la période d'échantillonnage horizontale et verticale exprimée en unité de la grille de référence. Ces deux valeurs pour l'i^{ème} composante sont notées $XR_{siz}(i)$ et $YR_{siz}(i)$ respectivement. Ces deux valeurs spécifient une grille d'échantillonnage rectangulaire constituée de points dont les positions horizontales et verticales sont des multiples entiers de $XR_{siz}(i)$ et $YR_{siz}(i)$. Tous les points qui tombent dans la zone image constituent des échantillons de la composante en question. En terme de système propre des coordonnées, une composante aura comme points limites :

$$\left(\left\lceil \frac{XO_{siz}}{XR_{siz}(i)} \right\rceil, \left\lceil \frac{YO_{siz}}{YR_{siz}(i)} \right\rceil\right) \quad et \quad \left(\left\lceil \frac{X_{siz}}{XR_{siz}(i)} \right\rceil, \left\lceil \frac{Y_{siz}}{YR_{siz}(i)} \right\rceil\right)$$

Sa taille sera donnée par :

$$\left(\left\lceil \frac{X_{siz}}{XR_{siz}(i)} \right\rceil - \left\lceil \frac{XO_{siz}}{XR_{siz}(i)} \right\rceil\right) * \left(\left\lceil \frac{Y_{siz}}{YR_{siz}(i)} \right\rceil - \left\lceil \frac{YO_{siz}}{YR_{siz}(i)} \right\rceil\right)$$

La taille de la zone image est: $(X_{siz}\text{-}XO_{siz}) * (Y_{siz}\text{-}YO_{siz})$

Pour une image donnée, différentes combinaisons des paramètres X_{siz}, Y_{siz}, XO_{siz} et YO_{siz} peuvent être choisies. Le fait de changer les valeurs de XO_{siz} et YO_{siz} (tout en gardant la taille de la zone image constante) permet de réaliser un certain nombre d'opérations de base sur l'image comme un décalage ou un recadrage.

Chaque composante peut être ensuite divisée en tiles. La découpe en tile permet de répartir l'image en des zones rectangulaires qui seront codées de façon indépendante et de limiter les ressources nécessaires pour le codage en terme de mémoire **[9]**, **[12]**, **[20]**, **[21]**, **[16]**, **[27]**. Toutes les opérations, comme le mixage des composantes, la transformée en ondelettes, la quantification et le codage entropique sont appliqués indépendamment sur chaque tuile. La tuile est l'unité de base de l'image originale ainsi que de l'image reconstruite.

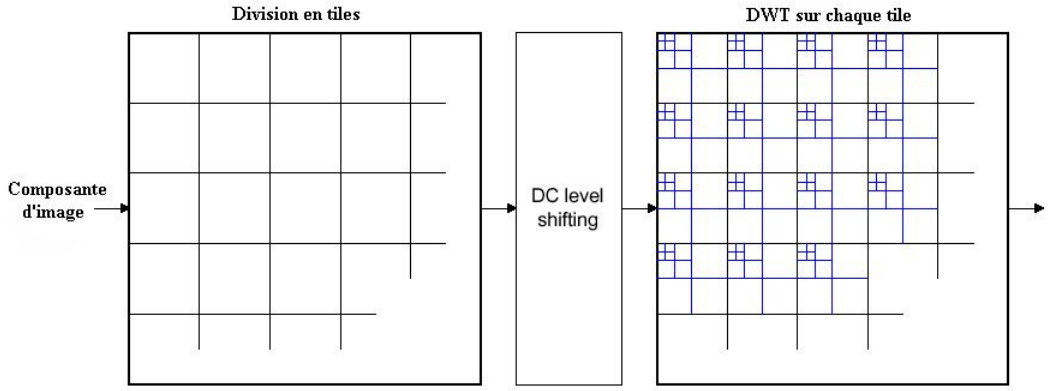


Figure Ch II - 5 : Division en tuiles, décalage en DC et DWT sur chaque composant.

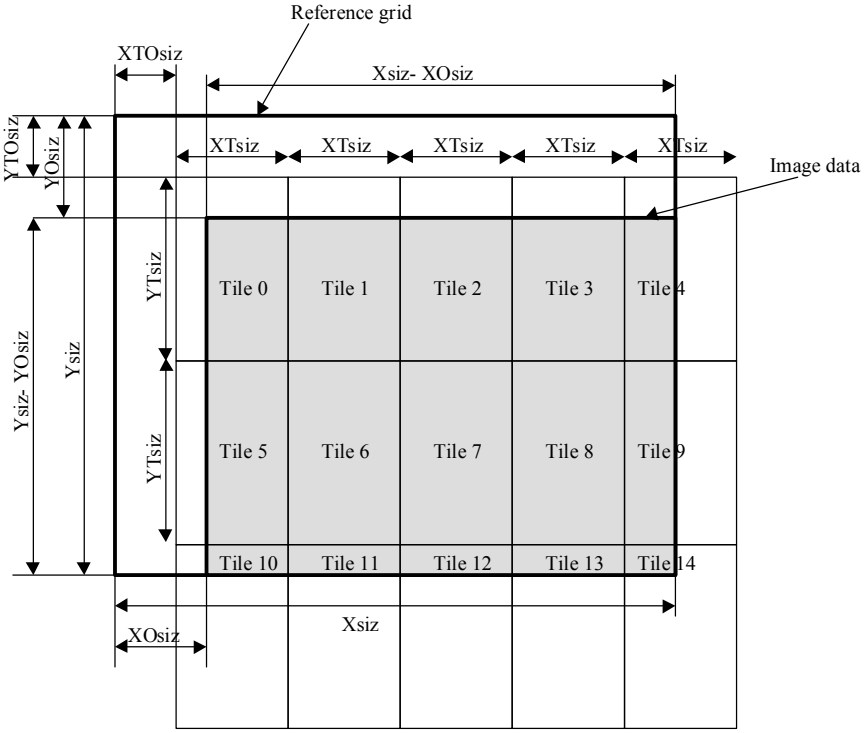


Figure Ch II - 6 : système de coordonnées dans la grille de référence

Les tiles réduisent les demandes de mémoire et comme elles sont reconstruites indépendamment, elles peuvent être utilisées pour décoder des parties spécifiques de l'image au lieu de décoder toute l'image. Les tiles sont de tailles identiques (excepté aux frontières de l'image) et ne se recouvrent pas.

Cette division de l'image en tiles a des effets sur la qualité de l'image compressée. Les petits tiles génèrent plus d'artefacts que les tiles de plus grande taille **[9]**. C'est à dire que les grands tiles ont des performances visuelles meilleures que les petits tiles. La dégradation de l'image est plus grande pour des bas débits que pour des hauts débits.

Les tiles rectangulaires de dimensions XT_{siz} * YT_{siz} sont repérés dans la grille de référence par l'origine du premier tile donné par XTO_{siz} et YTO_{siz}. Les tiles situés aux frontières de la zone image peuvent avoir des tailles plus petites. Ils sont numérotés de gauche à droite puis de haut en bas en commençant par zéro. En faisant correspondre la position de chaque tile sur la grille de référence par rapport au système de coordonnées de chaque composante individuelle, un découpage de chaque composante peut être obtenu. Un tile ayant le coin supérieur gauche et le coin inférieur droit aux positions respectives (tx_0, ty_0) et (tx_1-1, ty_1-1), alors dans le système de coordonnées d'une composante particulière, ce tile aura le coin supérieur gauche et le coin inférieur droit aux positions respectives suivantes (tcx_0, tcy_0) et (tcx_1-1, tcy_1-1) avec:

$$(\mathrm{tcx}_0, \mathrm{tcy}_0) = \left(\left\lceil \frac{\mathrm{tx}_0}{\mathrm{XR}_{\mathrm{siz}}} \right\rceil, \left\lceil \frac{\mathrm{ty}_0}{\mathrm{YR}_{\mathrm{siz}}} \right\rceil \right)$$

$$(\mathrm{tcx}_1, \mathrm{tcy}_1) = \left(\left\lceil \frac{\mathrm{tx}_1}{\mathrm{XR}_{\mathrm{siz}}} \right\rceil, \left\lceil \frac{\mathrm{ty}_1}{\mathrm{YR}_{\mathrm{siz}}} \right\rceil \right)$$

1.2 Décalage de la composante continue

Le recalage des données convertit les valeurs des composantes non signées de l'image en valeurs signées. Cette opération a pour but d'assurer que la dynamique des données est distribuée symétriquement autour de la valeur zéro. Soit Ssiz le nombre de bits représentant les valeurs des pixels d'une des composantes de l'image. La plage dynamique peut être soit $[-2^{Ssiz-1}, 2^{Ssiz-1}-1]$ ou $[0, 2^{Ssiz}-1]$ suivant que les valeurs sont signées ou non signées. Dans le cas de valeurs non signées, l'intervalle dynamique nominal n'est pas centré autour de zéro. La valeur à retrancher à chaque pixel est de 2^{Ssiz-1}.

Cette opération permet aussi, quelle que soit la représentation signée ou non signée des données en entrée, d'avoir la même représentation interne. Bien évidemment on conservera l'information permettant de savoir si les données à reconstituer en sortie de décodage étaient signées ou non signées. Le décodeur réalise l'opération inverse. Si les valeurs de l'image décodée doivent être non signées, on ajoutera alors 2^{Ssiz-1} aux données.

2. Transformée en couleur

JPEG2000 permet la compression d'images à plusieurs composantes telles que des images couleurs. La représentation chromatique de JPEG2000, utilisée pour les images couleur comprenant trois composantes, est du type (Y, Cb, Cr), bien que d'autres représentations puissent être utilisées par le standard. Selon l'espace couleur de l'image considérée, une transformation des composantes est nécessaire pour changer de représentation chromatique et ainsi passer d'une représentation type (R, G, B) à la représentation type de JPEG2000.

Cette transformation s'obtient très simplement en faisant une combinaison linéaire des composantes RGB. Dans le cas d'une compression sans perte on utilise la formule RCT (manipulation de nombres entiers), pour une compression avec perte on se sert de la formule ICT (nombres réels).

Pour l'ICT :

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

L' ICT doit être utilisé avec une transformation en ondelettes irréversibles (filtres de Daubechies 9/7). Les transformations directes et inverses sont obtenues à partir des équations suivantes **[4]**, **[20]**, **[25]**:

La RCT, transformée couleur réversible à utiliser dans le cas de compression sans perte, est une approximation de la transformée précédente :

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0 & -1 & 1 \\ 1 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Le codage des trois composantes obtenues se fera alors indépendamment comme le montre la figure suivante :

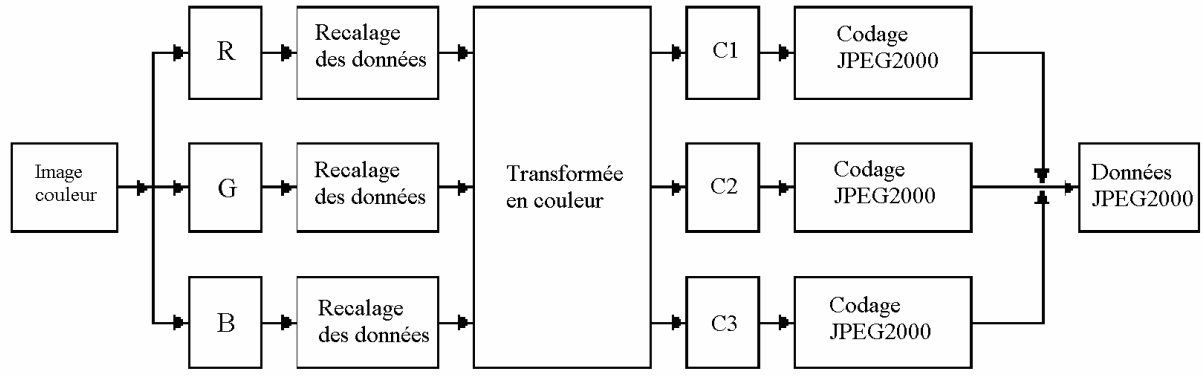


Figure Ch II - 7 : Codage des composantes de couleur

3. Transformée en ondelettes

La transformée en ondelettes est utilisée pour l'analyse des composantes imagettes pour différents niveaux de décomposition **[5]**, **[13]**, **[30]**, **[31]**. Ces niveaux de décomposition contiennent un nombre de sous-bandes, composées chacune de coefficients qui décrivent les caractéristiques horizontales et verticales de l'imagette originale.

Dans la première partie du standard JPEG2000 sont permises seulement des décompositions de puissance 2 sous la forme d'une décomposition dyadique. On passe d'un niveau N-1 à N par un filtrage passe-bas et passe-haut sous échantillonnés d'un facteur 2 sur les lignes puis sur les colonnes. Chaque sous-bande représente l'activité du signal dans les différentes bandes de fréquence à différentes résolutions spatiales. Sur un niveau, on obtient une décomposition de l'image en sa composante principale (bande LL) et trois composantes de détails qui représentent les détails horizontaux, verticaux et diagonaux de l'image. Si le nombre de niveaux de résolutions spatiales est N_L, on a alors $3*N_L+1$ sous-bandes différentes.

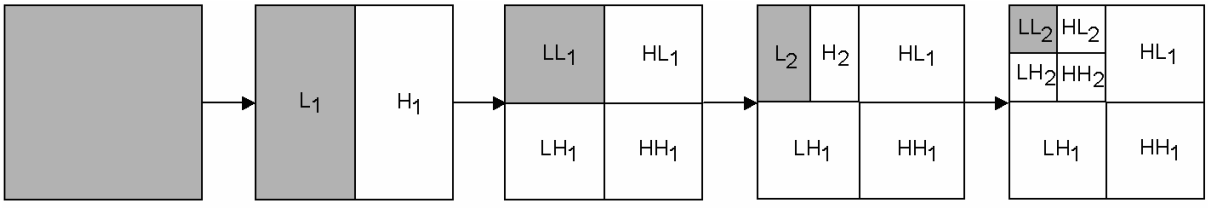


Figure Ch II - 8 : Transformée en ondelettes

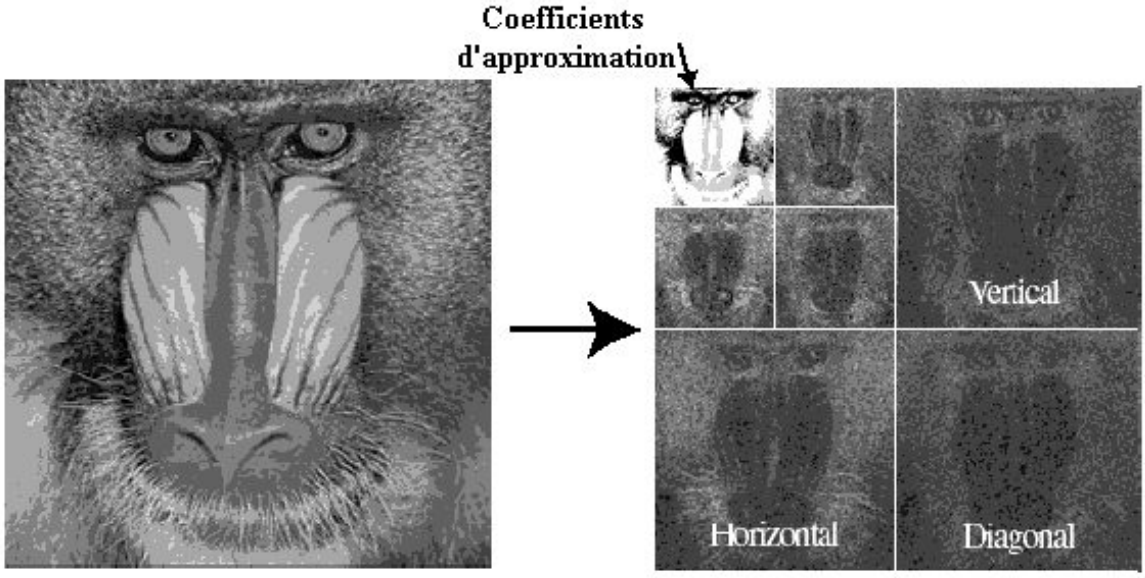


Figure Ch II - 9 : Décomposition d'une image en ondelettes en deux niveaux.

Pour réaliser la transformée en ondelettes discrète, le standard utilise une décomposition unidimensionnelle (1-D) de la sous-bande pour les deux directions horizontales et verticales, avec des filtres passe-bas et passe-haut. Les coefficients donnés par le filtre passe-haut représentent une version résiduelle des coefficients originaux, nécessaires pour la reconstruction parfaite de l'image originale à partir des coefficients passe-bas. La transformée en ondelettes discrètes peut être irréversible ou réversible. La transformée irréversible de base est calculée avec le filtre 9/7 de Daubechies **[1]**. La transformée réversible est réalisée avec le filtre Le Gall 5/3 **[11]**.

Le signal doit être d'abord élargi périodiquement **[30]**. Cette extension est utilisée pour assurer la correspondance entre les coefficients du masque de filtrage et les coefficients qui se trouvent au bord du signal. La dimension de l'extension dépend de la taille du masque de filtrage.

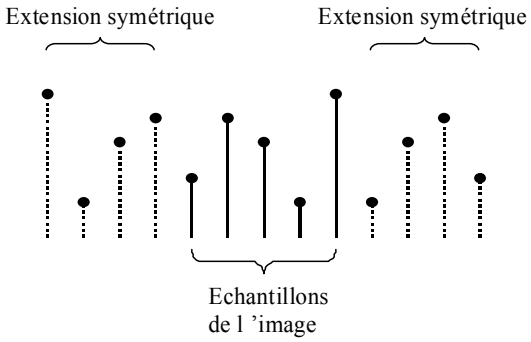


Figure Ch II - 10 : Extension des échantillons

La transformée est appliquée indépendamment sur chaque tile et sur chaque composante. Elle est réalisée en deux étapes, à l'aide d'une décomposition en sous-bandes à une dimension, appliquée tout d'abord de façon horizontale puis verticale et cela de façon itérative jusqu'à ce que le nombre de niveaux de décomposition ait été atteint On décompose suivant N niveaux de résolution. Ces niveaux de résolution contiennent alors les coefficients d'ondelettes qui décrivent les caractéristiques spatiales locales pour un certain facteur d'échelle du tile. Chacun se décompose en sous-bandes qui contiennent les contributions fréquentielles verticales, horizontales et diagonales de l'image pour un niveau de résolution donné.

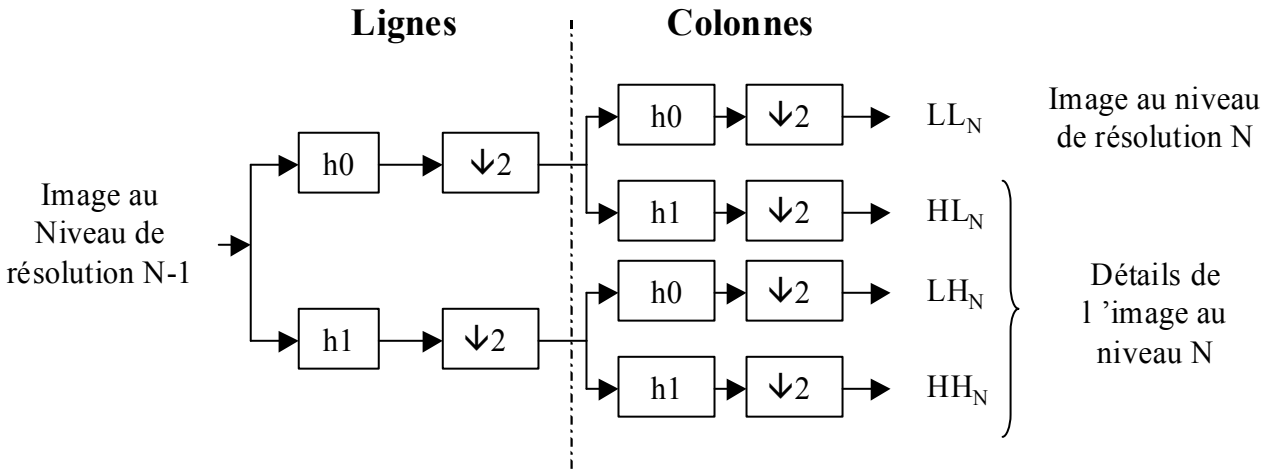


Figure Ch II - 11 : Banc de filtres d'analyse pour la transformée en Ondelettes

Lors du décodage, l'image sera reconstruite en opérant la transformée en ondelettes inverse selon le schéma suivant :

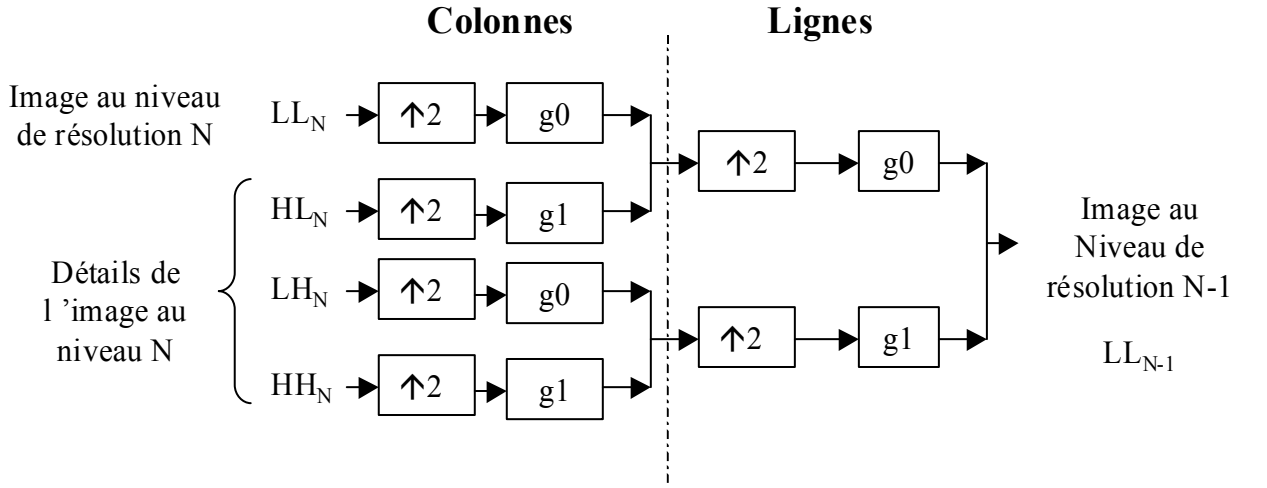


Figure Ch II - 12 : Transformée ondelettes inverse

Soient X(n) la séquence unidimensionnelle des échantillons d'une ligne ou d'une colonne et Y(n) la séquence unidimensionnelle des échantillons entrelacés des coefficients des deux sous-bandes créés après filtrage et décimation. Les coefficients pairs Y(2n) correspondent à la sortie du filtre passe-bas et Y(2n+1) à la sortie du filtre passe-haut. On a :

$$Y(2n) = \sum_{k=1}^{N} h0(k) \times X(2n+k) \qquad Y(2n+1) = \sum_{k=1}^{N} h1(k) \times X(2n+1+k)$$

Où h0(k) et h1(k) correspondent aux filtres passe-bas et passe-haut.

Base 5/3

Dans le cas du filtre 5/3, les coefficients sont les suivants :

h0(n)= (-1/8 1/4 3/4 1/4 -1/8)

h1(n)=(-1/2 1 -1/2)

Ce qui donne les réponses en fréquence suivantes :

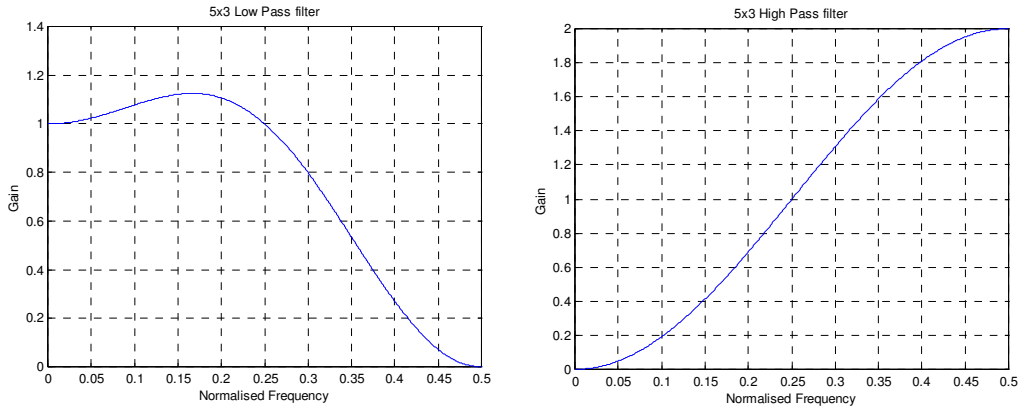


Figure Ch II - 13 : Les réponses en fréquences du filtre 5/3

Base 9/7

Dans le cas du filtre 9/7, les coefficients sont les suivants :

h0(n)= (0.026748757410810 -0.016864118442875 -0.078223266528990 0.266864118442875 0.602949018236360 0.266864118442875 -0.078223266528990 -0.016864118442875 0.026748757410810)

h1(n)=(0.091271763114250 -0.057543526228500 -0.591271763114250 1.115087052457000 -0.591271763114250 -0.057543526228500 0.091271763114250)

Ce qui donne les réponses en fréquence suivantes :

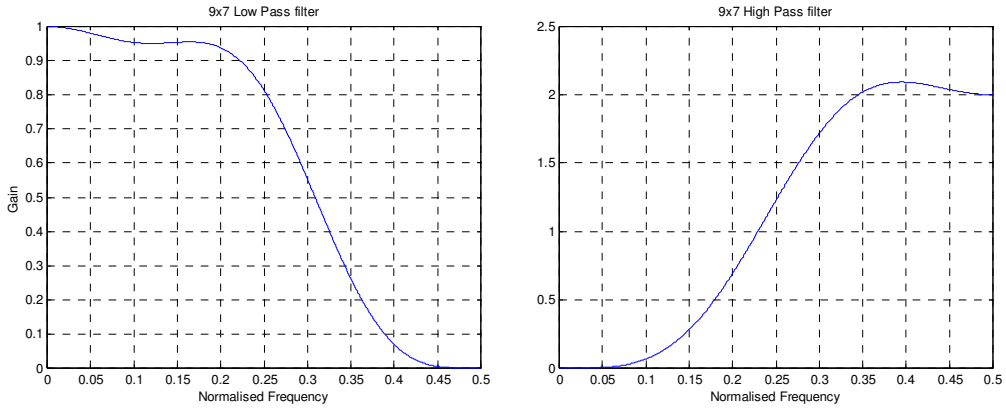


Figure Ch II - 14 : Les réponses en fréquences du filtre 9/7

Comment sont reliés les coefficients des filtres d'analyse et de synthèse entre eux ? Soit h0 et h1 les deux filtres d'analyse et g0 et g1 les deux filtres de synthèse. La reconstruction parfaite impose aux filtres d'analyse et de synthèse les conditions suivantes aux transformées en Z des filtres:

$$H_0(z)\,G_0(z)+H_1(z)\,G_1(z)=2$$

$$H_0(-z)\,G_0(z)+H_1(-z)\,G_1(z)=0$$

Pour les filtres de synthèse 5/3, on obtient les coefficients suivants :

g0(n)=(1/2 1 1/2)

g1(n)=(-1/8 –1/4 3/4 -1/4 -1/8)

Ce qui donne les réponses suivantes en fréquence:

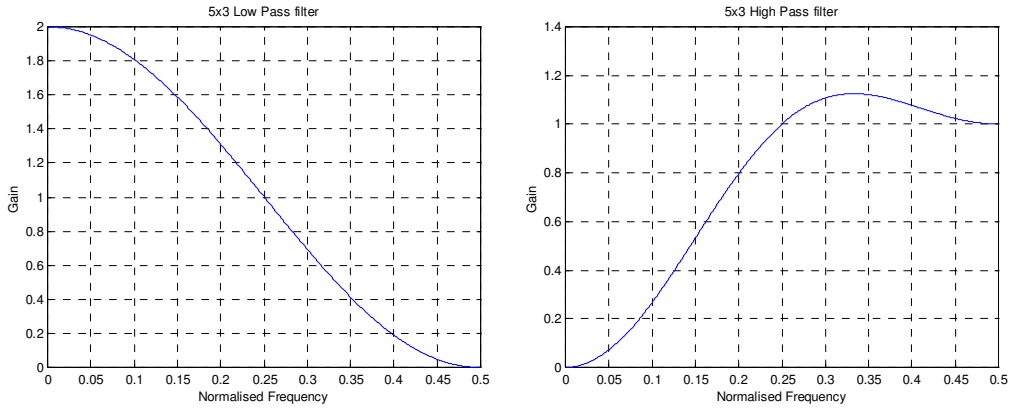


Figure Ch II - 15 : Les réponses en fréquences du filtre de synthèse 5/3

Pour les filtres 9/7 de synthèse on obtient les coefficients suivants :

g0(n)=(-0.091271763114250 -0.057543526228500 0.591271763114250 1.115087052457000 0.591271763114250 -0.057543526228500 -0.091271763114250)

g1(n)=(0.026748757410810 0.016864118442875 -0.078223266528990 -0.266864118442875 0.602949018236360 -0.266864118442875 -0.078223266528990 0.016864118442875 0.026748757410810)

On obtient les réponses en fréquence suivantes pour les filtres de synthèse 9/7:

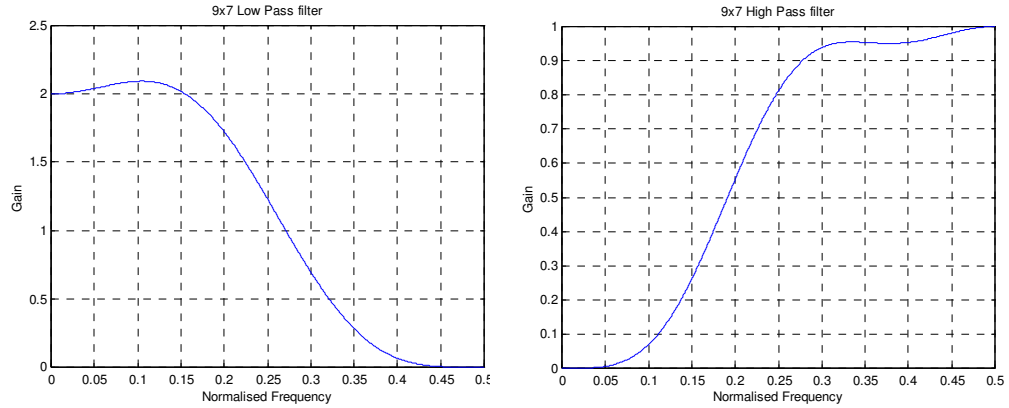


Figure Ch II - 16 : Les réponses en fréquences du filtre de synthèse 9/7

Pour réduire la complexité algorithmique, JPEG2000 propose d'implanter la transformée en ondelettes en utilisant la méthode du lifting (lifting-scheme) **[28], [29], [33]**. Celle-ci permet de minimiser le nombre d'opérations ainsi que la taille de la mémoire, en réalisant simultanément les filtrages passe-bas et passe-haut.

Le filtrage convolutif réalise une série de produits entre les deux filtres et le signal. Le filtrage basé sur le lifting est constitué d'une séquence très simple d'opérations de filtrage, pour lesquelles les valeurs paires alternatives du signal sont remplacées par la sommation des valeurs impaires, et les valeurs impaires du signal sont remplacées par la sommation des valeurs paires **[2]**, **[14]**, **[6]**, **[28]**, **[29]**. Pour le cas réversible (non destructif) les résultats sont arrondis à une valeur entière.

Dans le cas de la transformée 5/3 on obtient la structure suivante :

$$\begin{cases} y(2n+1) = x(2n+1) - \left\lfloor \dfrac{x(2n) + x(2n+2)}{2} \right\rfloor \\ y(2n) = x(2n) + \left\lfloor \dfrac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor \end{cases}.$$

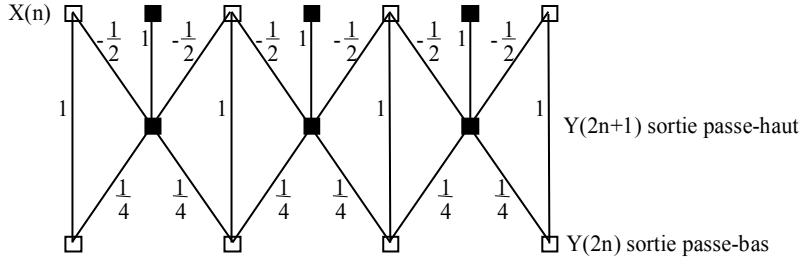


Figure Ch II - 17 : Structure du filtre 5/3

Dans le cas de la transformée 9/7 on obtient la structure et les équations suivantes :

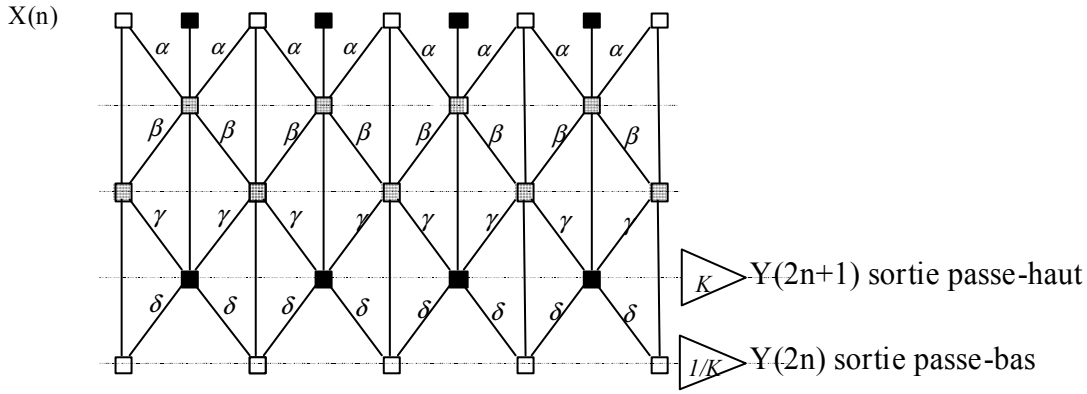


Figure Ch II - 18 : Structure du filtre 9/7

$$\begin{aligned} Y(2n+1) &= X(2n+1) + \alpha \times (X(2n) + X(2n+2)) \\ Y(2n) &= X(2n) + \beta \times (Y(2n-1) + Y(2n+1)) \\ Y(2n+1) &= Y(2n+1) + \gamma \times (Y(2n) + Y(2n+2)) \\ Y(2n) &= Y(2n) + \delta \times (Y(2n-1) + Y(2n+1)) \\ Y(2n+1) &= Y(2n+1) \times K \\ Y(2n) &= Y(2n) \times \left(\frac{1}{K}\right) \end{aligned}$$

$$\begin{aligned} \alpha &= -1.586134342 \\ \beta &= -0.052980118 \\ \gamma &= 0.882911075 \\ \delta &= 0.443506852 \\ K &= 1.230174105 \end{aligned}$$

L'implantation de la transformée en ondelettes nécessite de stocker toute l'image et de réaliser des filtrages sur les deux directions, verticale et horizontale. Si le filtrage horizontal est très simple, le filtrage vertical est, par contre, plus compliqué. Le filtrage selon les lignes nécessite le parcours d'une seule ligne ; le filtrage selon les colonnes nécessite le parcours d'une partie de l'image, voire de toute l'image.

Par exemple, si on utilise le filtre 9/7 on a besoin de filtrer 9 lignes de l'image pour pouvoir commencer le filtrage en colonne. La transformée en ondelettes basée sur les lignes dépasse ces difficultés, donnant exactement les mêmes coefficients que la transformée en ondelettes traditionnelle **[7]**, **[8]**, **[20]**.

4. Quantification scalaire

La transformation en ondelettes est suivie d'une quantification scalaire permettant de réduire la dynamique des données. Cette opération est destructive, sauf si le pas de quantification est égal à 1 et les coefficients sont des entiers, comme ceux produits par

l'ondelettes réversible 5/3. Chaque coefficient de la transformation $c_b(u,v)$ de la sous-bande b est quantifié à valeur $q_b(u,v)$ suivant l'équation **[20]**, **[21]** :

$$q_b(u,v) = signe(c_b(u,v))\left\lfloor \frac{|c_b(u,v)|}{\Delta_b} \right\rfloor$$

Où Δ_b est le pas de quantification fonction de la dynamique et du type de la sous-bande, du nombre de niveaux de décomposition. La dimension du pas de quantification Δ_b est représentée par rapport à la dimension dynamique de la sous-bande b. C'est à dire que le standard JPEG2000 supporte de différentes tailles du pas de quantification pour chaque sous-bande **[30]**. Néanmoins, une seule taille du pas de quantification est permise pour une sous-bande. La dimension dynamique dépend du nombre des bits utilisés pour représenter l'imagette de l'image originale et du choix de la transformée en ondelettes. Les gains des filtres LL, LH, HL et HH n'étant pas identiques, on utilise un pas de quantification différent suivant la sous-bande à laquelle appartiennent les coefficients (seulement dans le cas d'une compression avec pertes). Pour la compression réversible, la taille du pas de quantification doit être égale à 1.

Δ_b sera représenté sous une forme (ε_b, μ_b) correspondant à :

$$\Delta_b = \left(1 + \frac{\mu_b}{2^{11}}\right) \cdot 2^{R_b - \varepsilon_b}$$

μ_b est un entier 11 bits non signé et ε_b un entier non signé codé sur 5 bits. Dans le cas d'une compression sans perte, seule la valeur $R_b = \varepsilon_b$ est signalée.

R_b correspond à la dynamique des coefficients d'ondelettes pour la sous-bande concernée. Soit R_i le nombre de bit par pixel de la composante à coder et G_b le gain obtenu après transformée en ondelettes pour la sous bande, on a :

$R_b = R_i + \log(G_b)$

Les valeurs de quantification seront ou non présentes dans les données codées. Soit elles sont indiquées pour chaque sous-bande, on parle alors de quantification explicite, soit elles sont recalculées à partir d'une seule valeur indiquée, n celle de la sous-bande LL, on parle alors de quantification implicite. Dans ce cas les valeurs de quantification sont recalculées de la façon suivante :

$(\varepsilon_b, \mu_b)=(\varepsilon_{LL}-n_{LL}+n_b, \mu_{LL})$

n_b étant le nombre de niveau de décomposition depuis l'image d'origine pour la sous-bande concernée.

5. Région d'intérêt

L'encodeur JPEG2000 permet de coder certaines régions de l'image avec plus de qualité que d'autres appelées Région d'Intérêt (ROI). Certains coefficients après la transformée en ondelettes sont identifiés comme appartenant à la région d'intérêt, les autres appartenant à la région de moindre intérêt appelée arrière plan. Le codage de régions d'intérêt s'opère entre la quantification scalaire et le codage entropique. Les coefficients appartenants à la ROI sont multipliés par une puissance de 2. Cette multiplication a pour conséquence de déplacer chaque bit constituant le coefficient ROI dans des plans supérieurs (au delà de la dynamique maximum des coefficients) par rapport aux coefficients d'arrière-plan, comme montré dans la

figure suivante. On a ainsi augmenté artificiellement la dynamique des coefficients de la région d'intérêt et donc leur importance en terme de signal.

La forme de la région d'intérêt n'est pas signalée au décodeur, seulement la présence d'une région d'intérêt et la valeur du décalage opéré sont signalées. Au décodage, les données seront redécalées de façon à retrouver la dynamique d'origine. Cette technique est appelée le « max-shift ». Ce codage de régions d'intérêt dispose de nombreuses propriétés. La plus intéressante est que les régions d'intérêt peuvent être de formes quelconques et être disjointes.

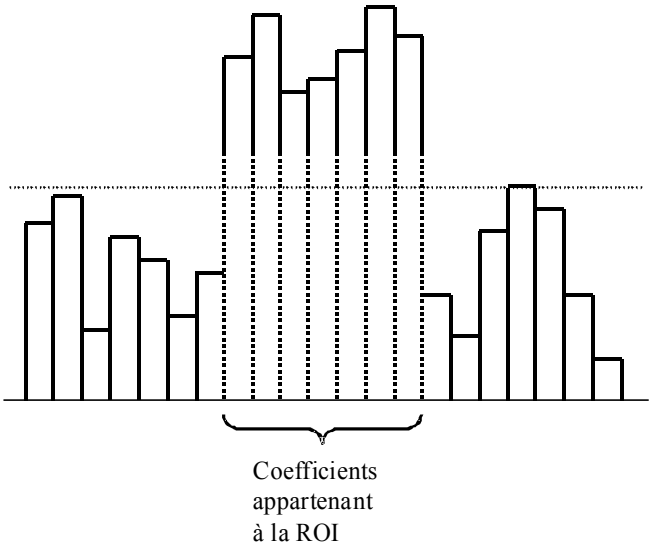


Figure Ch II - 19 : Localisation de la ROI

6. Codage entropique basé plan de bits (*bit-plane*) et blocs

Le codage entropique de JPEG2000 se compose des éléments suivants :

- Découpe en blocs appelés code-blocks,
- Conversion de la représentation des valeurs des coefficients en valeur absolue/signée,
- Modélisation des données binaires (modélisation du contexte),
- Codage arithmétique.

6.1 Découpe en blocs

Après quantification scalaire, les coefficients issus des différentes sous-bandes sont rangés en blocs, appelés code-blocks (CB), de forme rectangulaire, de taille paramétrable, leurs hauteurs et largeurs correspondant à une puissance de deux, et le produit largeur hauteur ne devant pas dépasser 4096 avec une hauteur minimale de 4.

Chaque code-block est codé indépendamment, sans aucune référence aux autres blocs de la même sous-bande ou d'une autre. Ce codage indépendant offre des avantages importants, comme un accès spatial aléatoire au contenu de l'image, une manipulation géométrique efficace, calcul parallèle durant le codage ou décodage, etc. Une présentation plus détaillée de ce processus est présentée dans **[27]**.

Le découpage d'une sous-bande en blocs est réalisé en superposant une grille constituée de rectangles de taille $2^{xcb} * 2^{ycb}$. L'origine cette grille est positionnée au point *(0,0)* dans le système de coordonnées propre à la sous-bande. Les paramètres XO_{siz} et YO_{siz} de la grille de référence déterminent la position du coin supérieur de chaque sous-bande (tbx_0, tby_0). A son tour, la quantité (tbx_0, tby_0) affecte la position de chaque bloc dans la sous-bande. Par

conséquent, les paramètres XO_{siz} et YO_{siz} jouent un rôle important dans le comportement de l'étage 1 du codeur en affectant les positions des blocs dans les sous-bandes. Un choix typique pour la taille nominale d'un bloc est 64*64 ce qui correspond à des valeurs de *xcb* et *ycb* égales à 6.

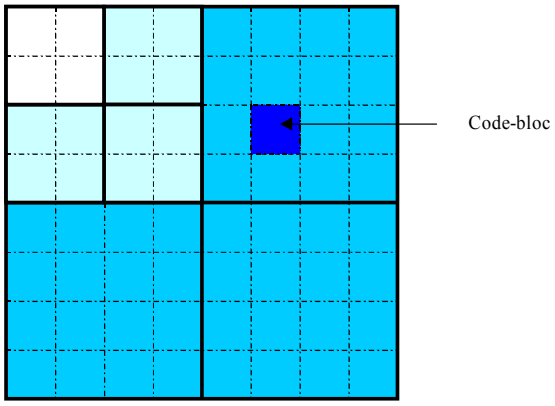


Figure Ch II - 20 : Partitionnement en code-blocks

A travers une sous-bande, on regroupe les code-blocks en precincts comme indiqué dans la figure suivante. Par ce regroupement, on obtient l'ensemble des coefficients d'un niveau de résolution pour une position spatiale donnée. La taille de ces precincts peut varier d'une sous-bande à l'autre. Une manière de garder toute cohérence serait de prendre des precincts variant d'un facteur 2 entre deux niveaux de résolution, ce qui correspond au facteur d'échelle de la transformée.

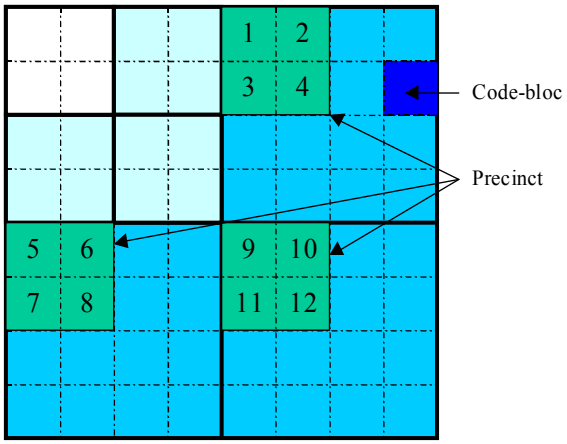


Figure Ch II - 21: Regroupement et ordre des codes-blocks dans un precinct

Le découpage d'une sous-bande particulière en precincts est dérivé du découpage de sa sous-bande parente *LL* (située à un niveau de résolution plus haut). Chaque niveau de résolution a une taille nominale de precinct. La largeur et la hauteur nominale de ces precincts doivent être une puissance de deux, sujet à certaines contraintes, excepté dans le cas où cette largeur et hauteur sont toutes les deux maximales (2^{15}). La sous-bande *LL* associée à chaque niveau de résolution est divisée en precincts. Cette opération est réalisée en superposant dans le système de coordonnées de la sous-bande LL une grille dont les dimensions en *x* et en *y* sont des multiples de 2^{PPx} et 2^{PPy}. L'origine de cette grille est identique à l'origine du système de coordonnées de la sous-bande *LL*. Les dimensions des precincts sur les côtés de la sous-bande peuvent être inférieures à la taille nominale. Chaque precinct résultant est ensuite projeté dans ses sous-bandes situées à un niveau de résolution plus bas. Ceci est accompli en utilisant la transformation suivante :

$$(u,v)=\left(\left\lceil\frac{x}{2}\right\rceil,\left\lceil\frac{y}{2}\right\rceil\right)$$

Où *(x, y)* et *(u, v)* sont respectivement les coordonnées dans la sous-bande LL et la sous-bande dérivée. Vu la façon dont le partitionnement des precincts est réalisé, les limites de ces precincts sont toujours alignées avec les limites des blocs. Certains precincts peuvent aussi être vides.

Supposons que la taille nominale d'un bloc soit $2^{xcb'} * 2^{ycb'}$. Ceci signifie que chaque precinct est constitué d'un groupe de $(2^{PPx'-xcb'} * 2^{PPy'-ycb'})$ blocs où

$$PPx' = \begin{cases} PPx & \text{pour } r=0 \\ PPx-1 & \text{pour } r>0 \end{cases}$$

$$PPy' = \begin{cases} PPy & \text{pour } r=0 \\ PPy-1 & \text{pour } r>0 \end{cases}$$

Et *r* est le niveau de résolution.

L'ordre de balayage des code-blocks présent dans un precinct est représenté dans la figure suivante :

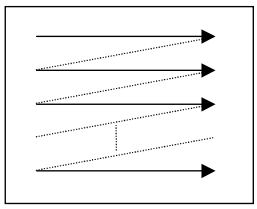


Figure Ch II - 22 : Ordre de balayage des blocs à l'intérieur d'une precinct.

Les code-blocks sont en fait des tableaux de coefficients qui peuvent être représentés par un tableau tridimensionnel binaire position-amplitude constitué de Nb bit-planes où Nb représente la dynamique de la sous-bande

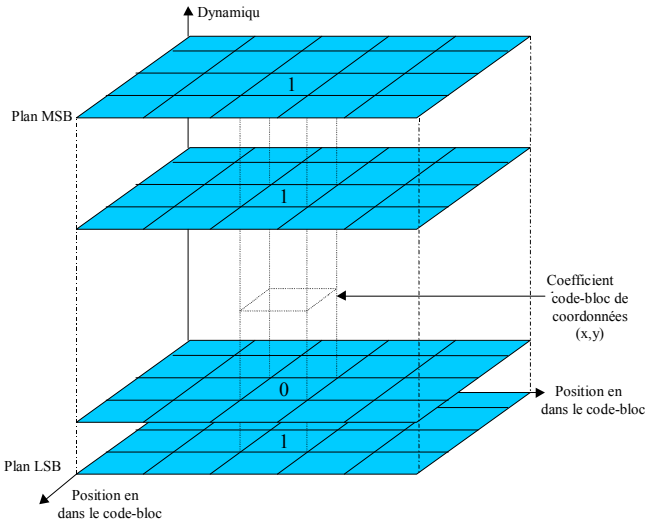


Figure Ch II - 23 : Représentation en bit-plane d'un code-block.

L'indépendance de ces code-blocks permet un parcours et un codage indépendants des coefficients de chaque code-block. Pour tous les coefficients, on adopte une représentation (signe, valeur absolue) des données.

6.2 Modélisation binaire des coefficients

Les échantillons du bloc sont rangés en bit-planes, ces plans démarrant par le bit de poids fort (MSB).

Chaque bloc sera codé au moyen d'un codeur par bit-plane se décomposant en trois étapes: la passe de signification (*Significance Pass*), la passe d'affinage (*Refinement Pass*), et la passe de nettoyage (*Cleanup Pass*). Ces passes ont comme objectif de préparer le codage entropique réalisé par un codeur de la famille des codeurs arithmétiques (codeur MQ) et dérivé du codeur QM mis au point par IBM et déjà utilisé de façon alternative à un codage de Huffmann, dans le standard ISO/IEC 10918-1 ou ITU-T T.81 dit JPEG.

Le bit représentant le coefficient dans le bit-plane courant ne pourra être codé que par une seule des trois passes. Chaque bit-plane est balayé dans un ordre prédéfini. Les données sont rangées par bandes de quatre valeurs de haut, qui seront balayées de haut en bas puis de la gauche vers la droite jusqu'à la prochaine bande (voir **figure Ch II - 21**). Chaque passe de codage génère une séquence de symboles qui seront utilisés comme entrées du codeur arithmétique qui les codera.

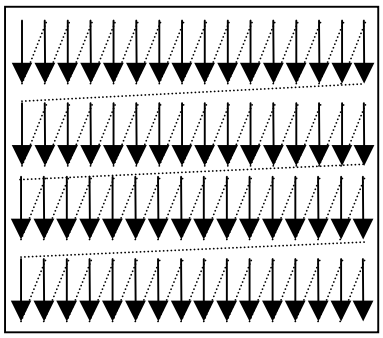


Figure Ch II - 24: Ordre de balayage d'un bit-plane

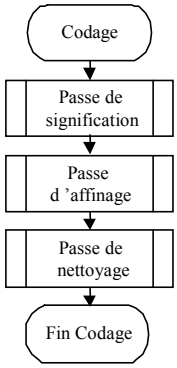


Figure Ch II - 25 : Passes de codage

Avant de décrire ces trois passes, indiquons qu'un échantillon est dit significatif si le bit de poids le plus fort de ce coefficient a déjà été codé. A chaque coefficient d'un code-block est associé une variable d'état appelée signification. Cette variable devient vraie (coefficient significatif) quand le premier bit à 1 est trouvé (MSB du coefficient). Pour chaque coefficient, on définit également un vecteur de contexte qui se réfère à l'état de ses huit voisins. Tous les coefficients voisins à l'extérieur d'un code-block sont considérés non significatifs. Le nombre de contexte possible est cependant réduit à 19 (0 à 18). Les règles d'assignement du contexte sont dépendantes du type de passe.

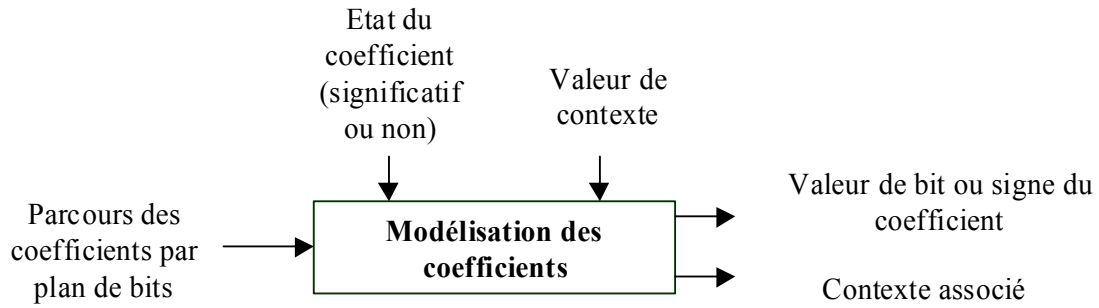


Figure Ch II - 26 : Le bloc de modélisation des coefficients

Les valeurs de contexte et les valeurs binaires issues de chaque passe sont ensuite envoyées au codeur arithmétique.

Passe de signification: codage des bits (0 ou 1) des échantillons non significatifs ayant au moins 1 voisin significatif plus éventuellement transformation d'un échantillon non significatif en significatif dans le cas du codage d'un bit à 1 (il s'agit alors du premier bit a 1 pour cet échantillon).

Passe d'affinage : codage des bits (0 ou 1) des échantillons significatifs non encore codés.

Passe de nettoyage : codage du reste des bits des échantillons non significatifs (0 ou 1). Si on code un bit à 1 (premier bit à 1 pour cet échantillon), on fait alors passer l'échantillon à l'état significatif. Cette passe contient aussi un mécanisme de codage par plage permettant de coder les séquences consécutives de zéros (Run-length coding).

Après ces trois passes, tous les bits du bit-plane ont été codés. Au début du codage, les échantillons étant tous non significatifs, le premier bit-plane n'est codé que par la passe de nettoyage. Si on appelle Ncb la dynamique du code-block pour la sous-bande b et N0 le nombre de bit-planes constitués uniquement de "0", alors le codage d'un code-block s'effectue en Np=3(Ncb-N0-1)+1 passes.

Une fois le codage d'un code-block terminé, on recommence l'opération pour le code-block suivant et ceci jusqu'à avoir codé tous les code-blocks.

6.3 Codage arithmétique

6.3.1 Principe du codage arithmétique

Dans le cas du **codage arithmétique non adaptatif**, on part d'une séquence de N mots choisis dans un alphabet A. Dans un modèle non-adaptatif, la distribution des mots de l'alphabet est connue et n'évolue pas. Le principe du codage arithmétique se distingue de codages de type Huffman par le fait qu'il n'assigne pas à un mot de la source un autre mot de longueur variable en fonction de sa statistique, mais il intervient au niveau de la séquence complète à coder : en effet le résultat fourni est la probabilité d'occurrence de la séquence complète. On assigne donc à chaque réalisation possible de la séquence sa probabilité. Le mot de code est donc un réel appartenant à un sous-intervalle de [0,1], le mot de code étant choisi avec une précision suffisante

Pour pouvoir distinguer au décodage à quel sous-intervalle appartient le mot de code. Les mots de code les plus courts sont ainsi assignés aux séquences les plus probables.

On donne ici un algorithme général permettant de coder une séquence :

On initialise un intervalle I=[0,1)

Pour chaque mot de la séquence :

- Subdivision de l'intervalle courant en sous-intervalles proportionnels à chaque probabilité des mots de l'alphabet.
- Sélection du sous-intervalle correspondant à l'ajout du symbole courant. Le sous-intervalle sélectionné devient le sous-intervalle courant.

Quand l'ensemble de la séquence a été codé, on sort suffisamment de bits pour caractériser le sous-intervalle de manière unique. La fin de codage reste à gérer soit par indication du nombre de données codées (longueur du mot de code) ne soit pas addition d'un mot de fin de séquence dédié.

La représentation du mot de code est donnée en binaire décimale. La longueur du dernier sous-intervalle est la probabilité de la séquence complète. On traite ci-dessous un exemple de manière à illustrer la méthode globale.

On part d'un alphabet de trois mots A={a,b,c}. Les probabilités a priori associées à chaque mot sont respectivement Pa=1/2, Pb=1/3, Pc=1/6. La figure suivante illustre l'algorithme précédent appliqué à cet exemple. La séquence à coder est (a,b,c). Le mot de code solution est alors 011001.

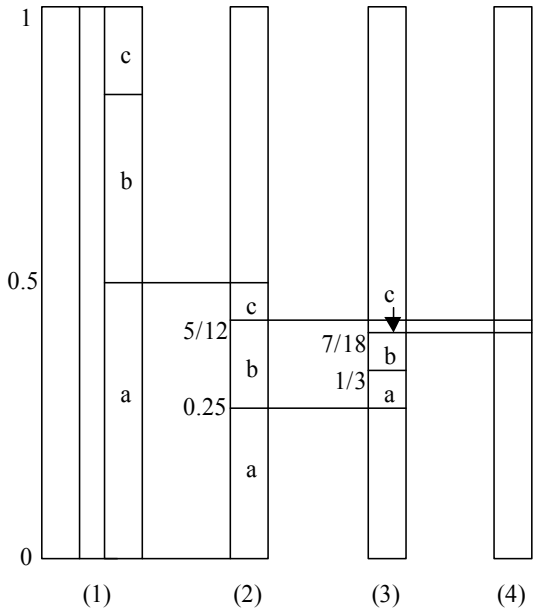


Figure Ch II - 27 : Illustration du codage arithmétique non adaptatif

Les quatre étapes de codage illustrées par la figure ci-dessus sont :

(1) : Subdivision de l'intervalle initial et sélection de l'intervalle correspondant à l'événement a.

(2) : Subdivision de l'intervalle courant et sélection de l'intervalle correspondant à l'événement b.

(3) : Subdivision de l'intervalle courant et sélection de l'intervalle correspondant à l'événement c.

(4) : Intervalle final dont la longueur correspond à la probabilité de la séquence (a,b,c) Ps=1/36. L'intervalle final est [7/18, 5/12). Comme l'intervalle [0.011001, 0.011010] est entièrement contenu dans I final, alors le mot de code 011001 suffit pour caractériser de manière unique l'intervalle solution.

Dans le cas du codage arithmétique adaptatif, on a davantage besoin de connaître les probabilités à priori de l'alphabet. Un modèle statistique prédéfini est sensiblement plus performant, mais il semble que le coût de transmission au décodeur du modèle statistique soit identique au coût d'apprentissage du modèle adaptatif.

Comme on peut le voir dans la structure précédente, le codeur arithmétique est un codeur statistique qui nécessite l'aide d'un organe de modélisation. Celle-ci permet de définir la distribution de l'alphabet des données.

Les probabilités estimées dans le cas adaptatif influent évidemment sur l'efficacité de compression : plus la précision est grande, plus l'efficacité sera importante. Ainsi les performances du codeur sont extrêmement liées au modèle statistique utilisé.

Le principe de codage décrit précédemment comporte cependant deux inconvénients majeurs : il nécessite une grande précision arithmétique en particuliers pour des séquences peu probables et le mot de code n'est décidé qu'à la fin d'une séquence complète. Pour pallier

à ces inconvénients, la solution est la mise en oeuvre d'un **codeur à sortie incrémentale**. Sans rentrer dans la mise en oeuvre pratique du codeur qui reste loin d'être immédiate, on présente ici le principe général d'expansion permettant de remédier aux deux inconvénients majeurs énoncés ci-dessus.

La solution générale est de sortir le bit significatif de poids fort dès que celui-ci est décidable. Une fois que le bit est sorti, si l'on revient à une représentation binaire du nombre sur un registre de taille fixe, la sortie du bit peut s'interpréter comme un décalage sur la gauche. On obtient dans le même espace de représentation, une représentation binaire de la partie décimale non décidée (une représentation de la partie inconnue de l'intervalle final). Ce décalage par la gauche correspond à un doublement de l'intervalle courant d'où le terme d'expansion. La figure suivante illustre ce concept.

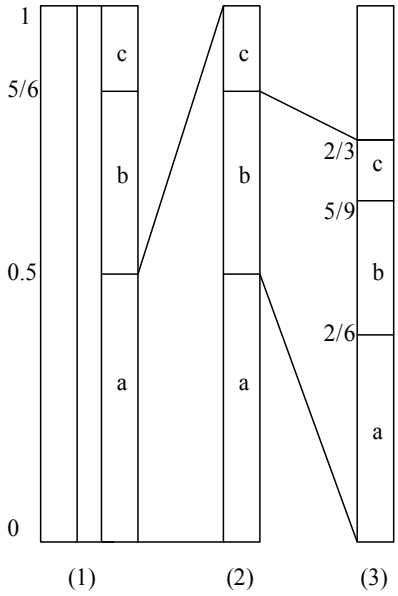


Figure Ch II - 28 : Illustration du codage arithmétique à sortie incrémentale

Les étapes mises en œuvre sont les suivantes :

1. : Subdivision de l'intervalle initial. L'intervalle représentant « a » permet de décider du premier bit 0. le registre ci-dessous illustre l'opération de sorties des bits. Le décalage des bits se traduit par un doublement de l'intervalle courant (2)
2. Subdivision de l'intervalle doublé. L'intervalle représentant « b » permet de décider du deuxième bit 1.
3. Subdivision de l'intervalle doublé. L'intervalle représentant « b » permet de décider des derniers bits en procédant de la même manière que pour l'exemple précédent.

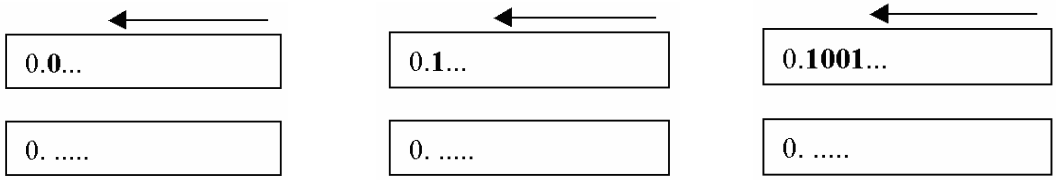


Figure Ch II - 29 : (a) mot de code : 0, (b) mot de code : 01, (c) mot de code : 011001

Bien évidemment ceci reste un principe général et les différents codeurs arithmétiques se proposent de gérer de manière différente la précision et la gestion de certains cas, comme les cas oscillant autour d'une probabilité 0.5.

6.3.2 Codeur arithmétique JPEG2000 (MQ)

Le codeur arithmétique utilisé dans JPEG2000, appelé codeur MQ prend comme entrées les valeurs binaires et le contextes associés issus de l'étape précédente de modélisation binaire des coefficients, et cela dans l'ordre des passes de codage.

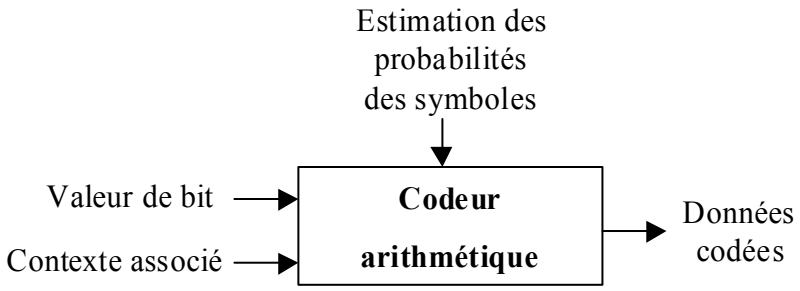


Figure Ch II - 30 : Schéma fonctionnel du codeur arithmétique

Le codeur est un codeur arithmétique binaire. Plutôt que de représenter les intervalles associés aux probabilités de "0" ou "1", il a été choisi de représenter les données à l'aide des symboles LPS (Less Probable Symbol) et MPS (More Probable Symbol) qui représentent respectivement les probabilités d'occurrence de l'espèce minoritaire et majoritaire. Bien évidemment, il faut garder trace du sens attribué à l'une ou l'autre des variables à savoir qui de "0" ou de "1" est l'espèce minoritaire.

Ainsi l'intervalle courant est représenté par l'intervalle I que l'on divise alors en deux sous-intervalles correspondant à l'espèce minoritaire et majoritaire. D'un point de vue représentation, on donne toujours comme intervalle inférieur le LPS. Moyennant une approximation dans la représentation, le codeur est implémenté de sorte que les hypothèses suivantes soient vraies:

- I-Qe est la longueur du sous intervalle pour le MPS
- Qe est la longueur du sous-intervalle pour le LPS
- Où Qe est la probabilité de l'espèce minoritaire.

Cette approximation est vraie si I est maintenu dans un intervalle de représentation proche 1. Ceci est réalisé notamment par des phases d'expansion. Le mot de code C généré est alors un pointeur qui pointe sur la base de l'intervalle courant. Ainsi si le bit suivant appartient à l'espèce majoritaire, en notant Qe la probabilité du LPS, alors il vient

C=C+Qe.

Le pointeur pointe alors la base du sous-intervalle correspondant au MPS dans l'intervalle courant. Si le bit est l'espèce minoritaire, alors le pointeur reste inchangé. MPS, LPS et Qe sont évidemment des fonctions du contexte.

La figure suivante représente les conventions utilisées dans le codeur arithmétique JPEG2000. I est l'intervalle courant, Qe la probabilité du LPS, C le pointeur sur le sous-intervalle courant. L'avantage de ce type de représentation par pointage de l'intervalle courant est que l'on ne fait que des additions au lieu de multiplications, plus gourmandes en terme de précision.

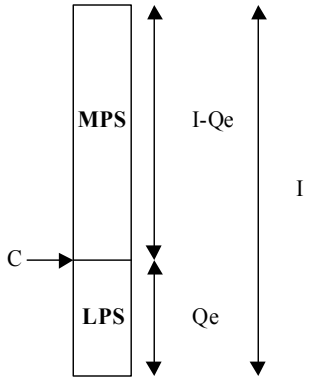


Figure Ch II - 31 : Conventions utilisées dans le codeur arithmétique

6.4 Allocation binaire optimisée débit/distorsion

JPEG2000 ne spécifie aucune méthode à employer pour réaliser le contrôle du taux de compression, mais propose d'utiliser une méthode d'optimisation débit/distorsion basé sur la méthode de Lagrange. On sélectionne alors les données issues des passes de codage, code-block par code-block en cherchant à minimiser l'erreur, et cela pour un débit fixé. On peut réaliser cette fonction plusieurs fois pour plusieurs débits fixés, ce qui permet d'obtenir une séparation des données en couches. Chaque couche permet d'accroître la qualité de l'image reconstituée en terme de PSNR. Ainsi chaque couche contient les données des différentes passes des code-blocks qui permettent d'atteindre un taux de compression donné en bits par pixel (bpp) pour une distorsion que l'on veut minimale. Le concept de couche est illustré dans la figure suivante où on représente la contribution de chaque code-block pour chaque couche:

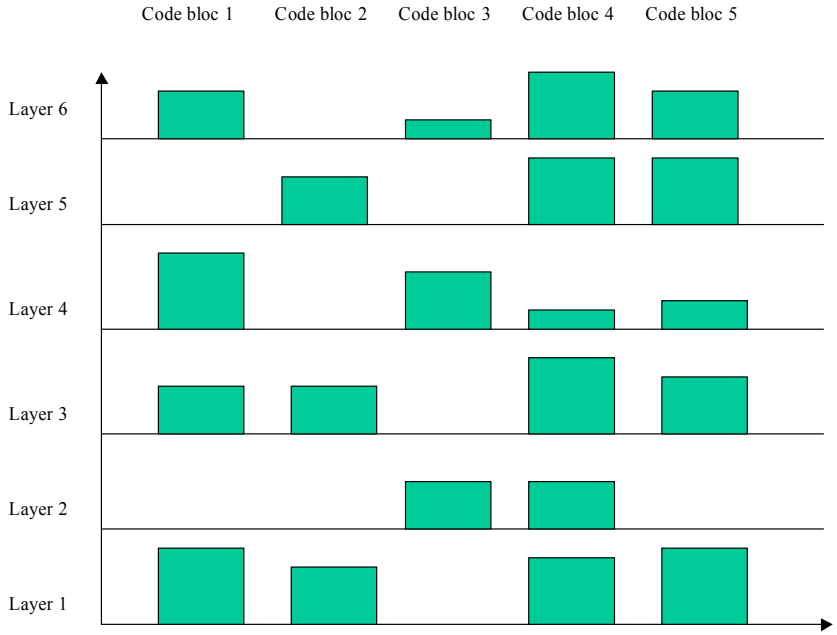


Figure Ch II - 32 : Illustration du concept de couches de qualité

C'est la formation de ces couches qui permet la fonctionnalité de progressivité en qualité de JPEG2000. Une couche peut être interprétée comme un incrément de qualité pour l'image entière en pleine résolution. L'optimalité est garantie par une procédure d'allocation débit/distorsion qui permet d'allouer les ressources entropiques dans le but d'atteindre un taux de compression fixé avec une distorsion minimale. Un exemple de courbe d'allocation débit/distorsion obtenue par JPEG2000 est présenté ci-dessous dans laquelle sont représentées

les valeurs d'erreur quadratique moyenne obtenues par rapport au taux de compression. Ces valeurs ont été obtenues à partir de l'image woman.

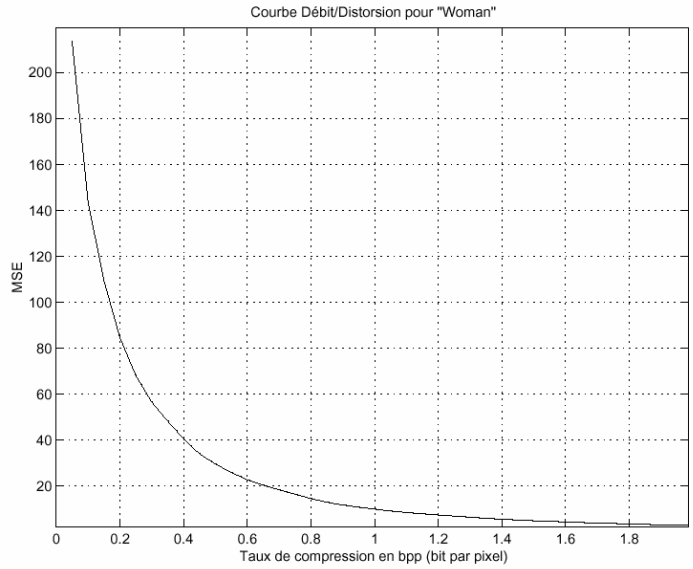


Figure Ch II - 33 : Exemple de courbe de performance de l'allocation Débit/Distorsion de JPEG2000.

Le principe d'optimisation proposé est basé sur l'utilisation de la méthode des multiplieurs de Lagrange. On va d'abord commencer par définir une courbe débit-distorsion obtenue en calculant tous les points de troncature d'un code-block. Les coordonnées du point sont obtenues en codant le code-block jusqu'à une certaine passe de codage, on lui associe alors une distorsion (puisque l'on n'a pas mis toutes les données) et un "débit" (la longueur des données une fois celles-ci codées par le codeur entropique).

On ne conserve que les points qui, une fois reliés entre eux, forment une courbe convexe. Ainsi, on obtient pour le codeur utilisé une frontière au dessous de laquelle on ne peut accéder. Le meilleur compromis débit-distorsion pour le code-block sera donc obtenu quand nous utiliserons les points présents sur cette courbe. Pour un codeur et un débit donné, la distorsion est minimale pour les points de troncature présents sur la courbe.

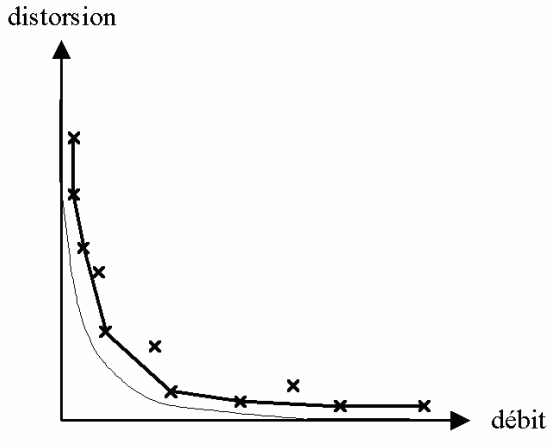


Figure Ch II - 34 : Courbe RD opérationnelle

A chaque code-block B_i, on associe une distorsion $D_i^{n_i}$ et une longueur finale (comprenant les entêtes eux aussi compressés) après codage $R_i^{n_i}$, n_i étant le point de troncature du code-block. Si n_i=0, cela signifie que l'on a tronqué entièrement le code-block,

n_i=1 indique que l'on a pris seulement les données issues de la première passe de codage, n_i=2 que l'on a pris les deux premières passes de codage, etc.…

L'image obtenue après la transformation par ondelettes étant divisée en multiples code-blocks, pour l'image reconstruite finale on aura donc : $D = \sum_i D_i^{n_i}$ et $R = \sum_i R_i^{n_i}$

On veut donc trouver l'ensemble $\{n_i\}$ des points de troncatures qui minimise D sous la contrainte $R \le R_{\max}$.

La méthode de Lagrange consiste à minimiser $D + \lambda R$ pour un λ donné

$\Rightarrow$ On trouve alors la plus petite distorsion possible associée au débit correspondant à l'ensemble $\{n_i\}$ trouvé.

Or, comme on a $D + \lambda R = \sum_i (D_i^{n_i} + \lambda R_i^{n_i})$, il est clair que minimiser $D + \lambda R$ revient à minimiser chacun des $D_i^{n_i} + \lambda R_i^{n_i}$

$\Rightarrow$ Pour chaque code-block B_i **,** on va donc chercher le point de troncature n_i qui minimise $D_i + \lambda R_i$, λ étant identique pour tous les CB.

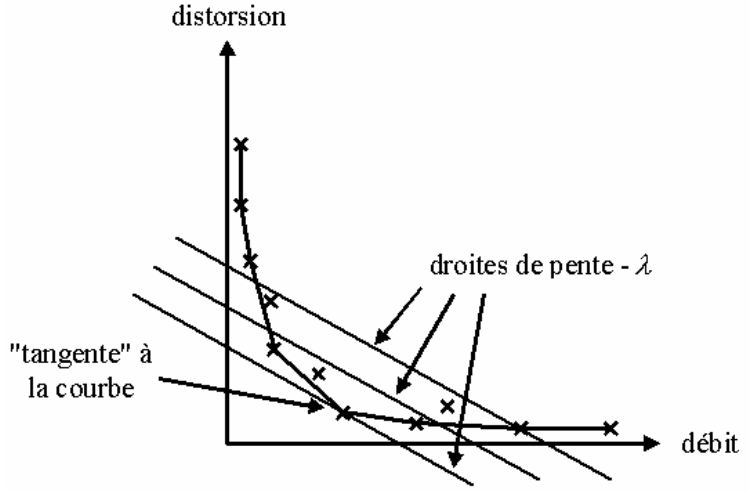


Figure Ch II - 35 : Recherche de la plus faible distorsion pour un débit donné

En effet, posons $J = D_i^{n_i} + \lambda R_i^{n_i} \Leftrightarrow D_i^{n_i} = J - \lambda R_i^{n_i}$

Minimiser J revient donc à trouver la droite de pente - λ la plus "basse" tout en ayant au moins un point en commun avec la courbe RD opérationnelle (voir **figure Ch II - 35**). La courbe RD étant convexe, minimiser J permet alors de trouver un point tangent à la courbe RD opérationnelle, et donc à trouver, pour un λ donné, la distorsion minimale associée au débit correspondant. Un λ élevé nous donnera un point correspondant à un débit faible et une distorsion élevée, un λ faible correspondra à un débit élevé et à une distorsion faible.

On modifiera le λ jusqu'à obtenir un $R \le R_{\max}$, avec un R le plus proche possible $R_{\max}$.

7. Syntaxe et ordonnancement des données codées

7.1 Ordonnancement des données

L'aboutissement du codage est la création du flux de données compressées JPEG2000. Celui est constitué de marqueurs et de segments contenant la totalité de l'information nécessaire au décodage. Les données codées sont regroupées en paquets dans un ordre dépendant de l'ordre de progression choisi. Un paquet est l'élément modulaire minimal dans JPEG2000. Il contient au maximum l'ensemble des informations des blocs d'un niveau de résolution d'un tile. Il contient au minimum les informations d'une couche et d'un bloc pour un niveau de résolution pour un precinct et d'un tile.

Un paquet contient toutes les données appartenant à un precinct donné, une résolution donnée et une couche donnée d'une composante donnée. C'est donc pour l'image une unité en terme de localisation, composante, qualité et résolution.

Il existe différentes possibilités d'arranger les paquets dans le train binaire final. La manière d'ordonner ces paquets est appelée ordre de progression. Il y a cinq ordres de progression possibles :

- progression par couche-résolution-composante-precinct (LRCP),
- progression par résolution-couche-composante-precinct (RLCP),
- progression par résolution-precinct-composante-couche (RPCL),
- progression par precinct-composante-résolution-couche (PCRL),
- progression par composante-precinct-résolution-couche (CPRL).

Dans le cas du premier arrangement, les paquets sont d'abord ordonnés par couche, ensuite par résolution puis par composante et pour terminer par precinct. Ce type de progression correspond au schéma de reconstruction progressive par qualité. Le second type de progression quant à lui correspond au schéma de reconstruction progressif par résolution. Les trois derniers types de progression correspondent à des schémas beaucoup plus ésotériques. Il est également possible de spécifier ses propres schémas de progression mais au détriment de l'efficacité de codage.

Dans le scénario le plus simple, tous les paquets d'une composante-tile particulière apparaissent ensemble dans le train binaire final. La possibilité existe cependant d'intercaler les paquets de différentes composantes-tiles pour permettre une meilleure flexibilité de l'arrangement des données. Si, par exemple, la reconstruction progressive par qualité des différents tiles d'une image est souhaitée, il serait certainement intéressant d'inclure ensemble tous les paquets associés à la première couche des différentes composantes-tiles suivies par tous les paquets associés avec la deuxième couche et ainsi de suite.

L'ensemble des paquets concernant un tile peut être segmenté en parties appelées « Morceau de tiles » (Tile-parts). Chacun peut comprendre un nombre variable de paquets entiers y compris zéro. Cependant chaque tile doit être au moins rattaché à un Tile-part. Les données de Tile-part permettent une organisation des données ou une modification de l'ordre de progression des paquets. Ainsi par exemple, on peut imaginer faire une progression multicouche, ou chaque couche permet d'atteindre un taux de compression intermédiaire donné et ainsi chaque Tile-part pourrait contenir les données d'une couche. Dans le deuxième cas, on peut imaginer que l'on veuille intervertir les ordres de progression et ainsi changer l'ordonnancement des données qui suivront.

7.2 Entêtes et segments

On s'intéresse ici à la syntaxe élémentaire de JPEG2000. Les données codées commencent tout d'abord par un entête dit entête principal définissant les paramètres généraux utilisés pour le codage. Puis par tile on trouve un ou plusieurs entêtes de Tile-Part, chaque tile pouvant être séparé en plusieurs morceaux. Le mode par défaut associe un Tile-part unique par tile. Les deux figures suivantes donnent une représentation de la structure des données codées, dans le cas où un seul morceau compose le tile et dans le cas où plusieurs morceaux sont utilisés par tile.

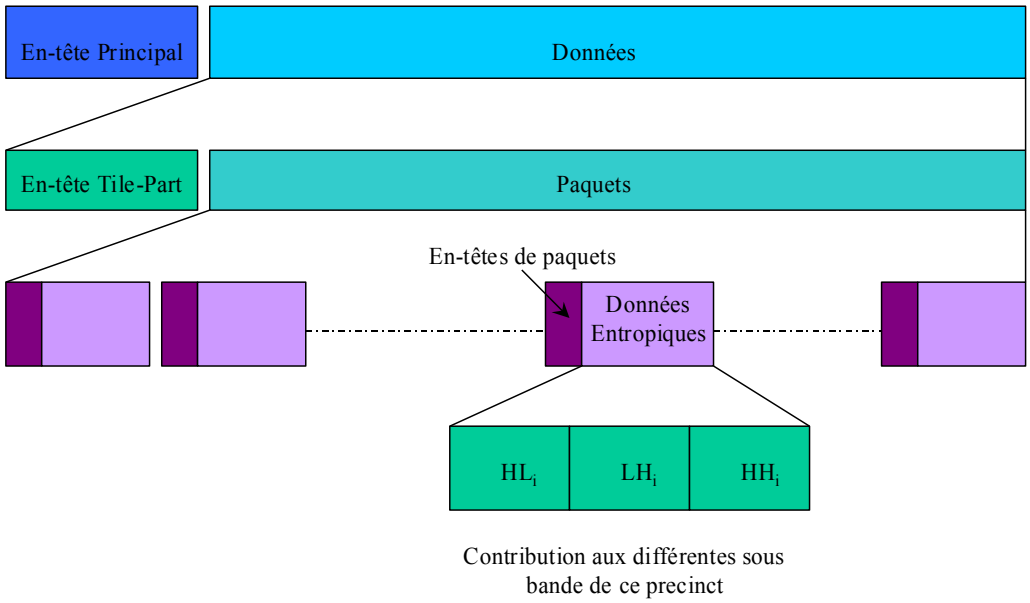


Figure Ch II - 36 : Syntaxe hiérarchique mono-Tile-part par défaut de JPEG2000

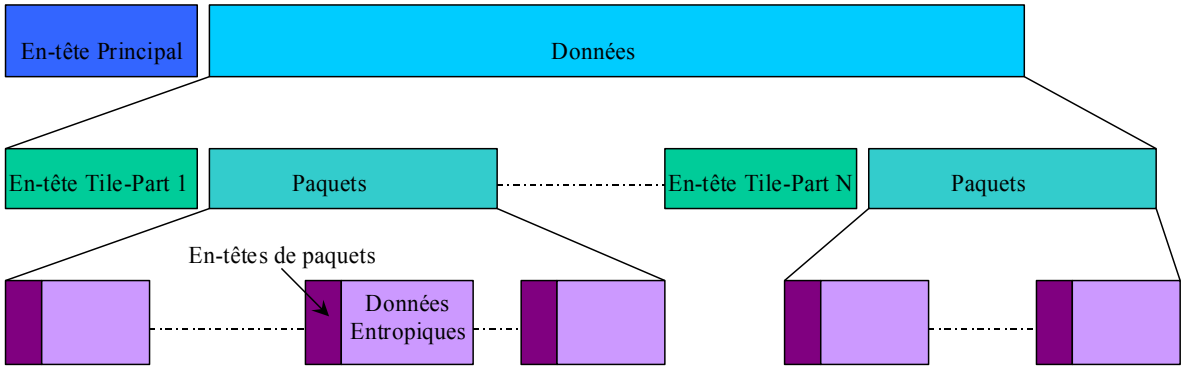


Figure Ch II - 37 : Syntaxe hiérarchique multi-Tile-part de JPEG2000

Le train binaire est une séquence de segments contenant les différents paramètres et informations permettant la description du flux de données codées.

L'entête principal contient toutes les informations globales caractéristiques de l'image nécessaires au décodage. Les paramètres spécifiés dans les segments de l'entête principal sont les paramètres par défaut de codage de l'image.

Les tiles-parts sont constitués d'un entête et leur corps contient un ensemble de paquets, fonction de l'ordre de progression désiré et des options attachées à ce Tile-part. Par défaut, l'image ne comporte qu'un Tile-part. Des données de description sont optionnelles dans l'entête de Tile-part. Si celles-ci sont présentes, elles ne sont valables que pour le Tile-part concerné et préemptent les données de l'entête principal.

En mode standard, les paquets sont constitués d'un entête et des données entropiques. Il n'y pas de marqueurs de début, ni de fin. De façon optionnelle, on peut ajouter des marqueurs de début et de fin d'entêtes de paquets. Les données à l'intérieur du paquet sont organisées ainsi : si le paquet appartient à la bande LL, le paquet contient les données y référant. Si il appartient à un niveau de résolution supérieur, les données des différentes sous-bandes sont organisées ainsi : Contributions HL, LH puis HH.

L'élément de base composant le flux de données compressé est le segment. Un segment est composé de 3 parties, un identifiant permettant de savoir de quel type de marqueur il s'agit, sa longueur et enfin les paramètres du segment. Certains types de segments n'ont pas de champs longueur et paramètres.

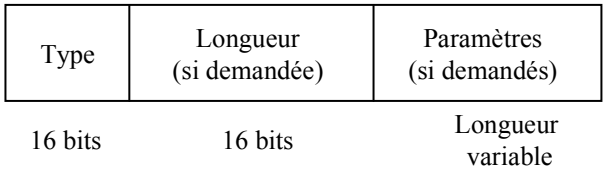


Figure Ch II - 38 : Composition d'un segment

Tous les segments, entêtes de paquet et corps de paquet ont des longueurs qui sont des multiples d'un octet. En conséquence, les champs type des différents segments sont toujours situés à des endroits multiples d'un octet. Ceci permet leur recherche plus facilement. Par ailleurs le train binaire est toujours un multiple d'un octet.

8. Etat de l'art des implémentations JPEG2000

8.1 Implémentations logicielles et analyse des performances

Nous présentons dans la suite de ce chapitre les performances du JPEG2000, par rapport aux autres standards de compression reprises dans **[27]**.

Les algorithmes utilisés dans **[27]** pour le jugement des performances sont :

- JPEG2000 *verification model* 8.6, language C, source : www.jpeg.org ;
- JPEG2000 (Jasper), language C, source : Image Power, University of British Columbia ;
- JPEG2000 (JJ2000), language Java, source : Canon Research/EPFL/Ericsson ;
- JPEG, language C, source : Independent JPEG Group
- JPEG-LS (SPMG), language C, source ; University of British Columbia ;
- *Lossless* (non-destructif) JPEG, langage C, source : Cornell University ;
- PNG, langage C, source : Ftp://ftp.uu.net/graphics/png.

Les performances sont présentées par rapport aux deux types de compression : avec et sans perte (destructive et non-destructive). Les algorithmes ont été appliqués sur plusieurs images de différents types, appartenant surtout à l'ensemble des images de test du standard JPEG2000 (www.jpeg.org).

Les résultats de la compression, jugés par rapport au PSNR (le pic du rapport signal bruit), ont montré que JPEG2000 a un plus de 2dB sur tous les autres standards, pour tous les taux de compression. Mais cette supériorité de la performance de JPEG2000 diminue à mesure que le débit augmente. En effet, du point de vue de la compression, JPEG2000 offre un facteur de compression meilleur de 10 à 20% que le JPEG de base, pour un débit de 1 bit par pixel (bpp).

L'efficacité de la compression non-destructive des algorithmes déjà mentionnés est présentée dans le **tableau Ch II - 1 [26]**. Le JPEG2000 a un comportement équivalent au JPEG-LS pour le cas des images naturelles. En tenant compte du fait que le JPEG-LS est de manière significative moins complexe que le JPEG2000, il est donc raisonnable d'utiliser JPEG-LS pour les compressions non-destructives. Mais dans ce cas, la caractéristique de généralité du JPEG2000 est sacrifiée.

Tableau Ch II - 1 : Taux de compression pour le codage non-destructif.

	J2KR	JPEG-LS	L-JPEG	PNG*
Bike	1.77	1.84	1.61	1.66
Café	1.49	1.57	1.36	1.44
Cmpnd1	3.77	6.44	3.23	6.02
Chart	2.60	2.82	2.00	2.41
Aerial2	1.47	1.51	1.43	1.48
Target	3.76	3.66	2.59	8.70
Us	2.63	3.04	2.41	2.94
Moyenne	**2.50**	**2.98**	**2.09**	**3.52**

Dans le tableau suivant on donne une comparaison entre le PSNR de JPEG2000 et celui de JPEG pour lors du décodage d'une image.

Tableau Ch II - 2 : PSNR, en dB, correspondant au MSE moyen de 200 exécutions du décodage de l'image "cafe" transmise à travers un cannal buitée avec différents bit error rates (BER) et taux de compression, pour JPEG baseline et JPEG2000 avec les filtres reversible and non-reversible (respectivement JPEG 2000R et JPEG 2000NR).

BER	JPEG2000R				JPEG2000NR				JPEG			
	0.25	0.5	1.0	2.0	0.25	0.5	1.0	2.0	0.25	0.5	1.0	2.0
0	22.64	26.21	31.39	38.27	23.06	26.71	31.91	38.93	21.94	25.39	30.34	37.23
1e-6	22.45	26.01	30.25	36.06	22.99	26.20	29.70	34.85	21.77	25.11	29.18	28.29
1e-5	20.37	23.35	25.91	25.80	21.11	23.06	25.80	24.68	20.42	22.61	22.33	19.67
1e-4	16.02	16.20	16.52	17.16	16.14	16.57	16.29	16.71	16.16	15.38	14.49	12.02

Depuis l'apparition du nouveau standard de compression d'images JPEG2000, les chercheurs qui travaillent dans ce domaine ont proposé certaines méthodes d'amélioration des performances pour toutes les étapes de la compression. Dans la section suivante, nous nous proposons de ne présenter que quelques travaux récents sur les implémentations matérielles du standard JPEG2000.

8.2 Implémentations matérielles

La complexité de JPEG2000 a suscité la conception de circuits dédiés pour réduire le temps d'exécution. La section suivante décrit les circuits commerciaux ayant apparus sur le marché, ou ayant fait l'objet d'une annonce suivie de documents techniques relativement restreints décrivant les fonctionnalités proposées.

8.2.1 Panorama

a - Insilicon Codec

La société InSilicon propose un cœur IP JPEG2000 CODEC **[32]**.

*W3C, PNG (Portable Network Graphics) Specification, Oct. 1996, http://www.w3.org/TR/REC-png

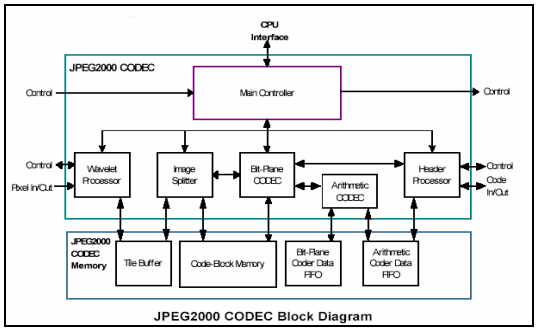


Figure Ch II - 39 : InSilicon JPEG2000 Codec

Le coeur se présente sous la forme d'un code source Verilog RTL, de modèles comportementaux Verilog ainsi que d'un ensemble de scripts permettant son intégration dans un SOC. Le cœur n'ayant pas été implémenté dans une technologie quelconque il n'est malheureusement pas possible de fournir des informations sur l'implémentation physique. Le diagramme bloc précédent reprend les composants classiques du flot JPEG2000 mais ne mentionne rien sur l'implémentation ou les choix architecturaux.

b - Amphion CS6510

Amphion propose avec le CS6510 un codeur JPEG2000 annonce comme disponible au second semestre 2002.

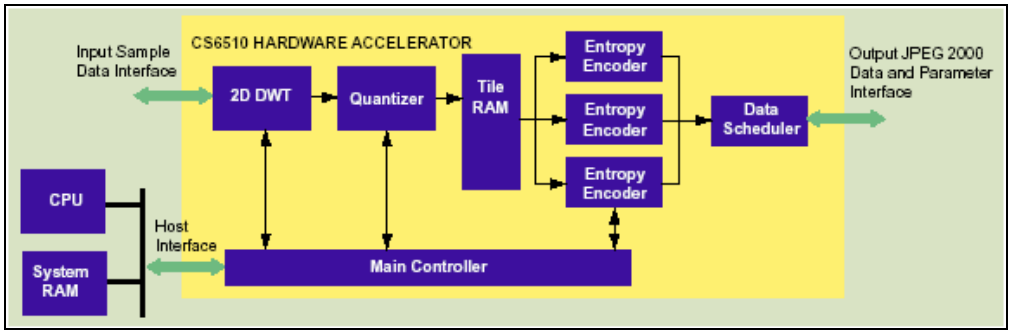


Figure Ch II - 40 : Diagramme block du CS6510 de AMPHION

Ce circuit fonctionnant à 150 Mhz occupe 130K portes logiques et 48KB de RAM. Quoique le nombre de codeurs entropiques semble être de trois sur la figure ci-dessus, la documentation ne l'explicite pas indiquant seulement un « certain nombre ».La taille maximum de l'image a traiter est très importante $2^{31}x2^{31}$ alors que la taille maximum du tile 128x128 est assez médiocre. La aussi aucune information ne permet de déduire les temps respectifs d'exécution des composants internes principaux. Aucune information n'est fournie sur la tension utilisée pour le circuit.

c - Analog Devices ADV-JP2000

Analog Devices **[33]** propose le ADV-JP2000. Ce circuit décrit ci-dessous fonctionne à une fréquence de 20 Mhz maximum inclut une transformée en ondelettes 5/3 (pas de 9/7) et un codeur entropique. Le circuit n'est pas totalement conforme à la norme. Le ADV-JP2000 propose 2 modes d'opérations : encode et décode. Dans le mode encode il accepte une seule tile et génère le flot de code-blocks conforme au standard. Le ADV-JP2000 communique par un protocole asynchrone mais permet aussi un mode par interruption. Enfin le circuit supporte le mode DMA.

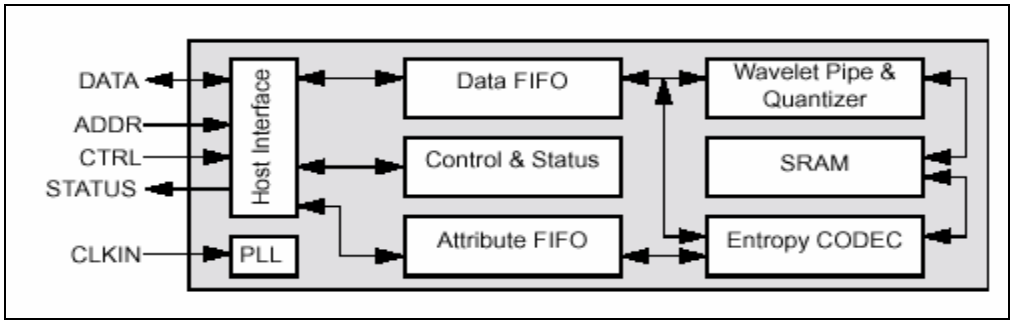


Figure Ch II - 41 : ADV-JP2000 de ANALOG DEVICE

Aucune information n'est fournie sur les temps de propagation de sous-modules ou des implémentations choisies. Le circuit fonctionne à 1.5-1.8 V pour le cœur et 3.3V pour les entrées sorties.

d - Blackbird de Image Power

Enfin le codeur de Image Power (Blackbird) a été annoncé mais aucune documentation détaillée n'est disponible.

Suite à ce panorama rapide on peut en conclure que globalement les architectures choisies reflètent assez le flot de traitement JPEG2000 mais que peu de détails d'implémentations peuvent être extraits des documentations disponibles. Dans une logique d'amélioration sur l'existant nous n'avons pas trouvé de détails suffisamment significatifs que nous pourrions considérer comme utiles pour nos travaux. L'architecture optimale et les modes de fonctionnement originaux restent ouverts à la recherche.

8.2.2 Critères

Dans les tableaux suivants on a essayé de regrouper l'ensemble des informations utiles fournies par les différents concepteurs de circuits. Les critères retenues sont : (1) le débit en Mpixels/sec dans les deux modes sans pertes et avec pertes (2) taille des composants (3) transformée en ondelettes implémentée (4) consommation d 'énergie (5) taille du circuit (mm^2).

Tableau Ch II - 3 : Aspects généraux

	INSILICON Codec	**AMPHION CS6510**	**ANALOG DEVICE: ADV-JP2000**	**CAST RC_2DDWT**	**Image Power Blackbird**
Taux de compression	>60:1	≤50:1	___	___	___
Type de filtrage	5/3	5/3, 9/7	5/3	5/3, 9/7	___
Conformité a la norme	ISO/IEC JTC 1/SC 29/WG 1	ISO/IEC 15444-1	___	___	ISO/ Part 1
Codage	Oui	Oui	Oui	Oui	Oui
Décodage	Oui	Non	Oui	Oui	Oui
Disponibilité	Verilog RTL	___	Chip	VHDL	IP core
Type de conception	Synchrone	synchrone	___	Synchrone	___
Surface	___	130K portes+48Ko RAM	7mmx7mm	___	___
Programmable	Oui	Oui	Oui	Oui	___
Performance	___	60Msamples/s	>10Mpixels/s	≤40Mpixels/s	20→80Mpixel/s
Lossless/lossy	Oui	Oui	Oui	___	___

Bit/échantillon	≤10	≤12	8-10 réversible 8-14 irréversible	8, 16 I/O 18 précisions interne	___
Clock	___	150MHz	20 Mhz	___	___
Technologie	___	0.18um	___	___	___
Interface CPU	Oui	Oui	Oui (asynchrone)	___	Oui
Vendeur	InSilicon	TSMC	ANALOG DEVICE	CAST, Inc	Image Power

Tableau Ch II - 4 : les spécificités

	INSILICON Codec	**AMPHION CS6510**	**ANALOG DEVICE: ADV-JP2000**	**CAST RC_2DDWT**	**Image Power Blackbird**
Taille Max. image	Programmable	2^{31}x2^{31} & programmable	___	512x512 & programmable	___
Taille Max. tile	___	128x128	160x128 256x256 single component	___	___
Taille Min. tile	___	___	8x8	___	___
Niveaux de subsampling	___	4/composant	___	___	___
Niveau de décomposition	Programmable	Programmable	___	Programmable	___
Nombre de tile traite	1	1	1	___	___
Component modes	___	___	3	___	___
software	___	__	Oui	___	___
FIFO format	___	___	16bit	___	___
DMA access mode	___	___	Singl/dual	___	___
protocol	___	EnHndshk	FullHndshk	___	___
Pixel format configuration	___	___	Oui	___	___
Compatibilité de connexion	___	___	___	___	AMBA standard

8.3 Applications de JPEG2000

Au cours du processus d'étude du standard JPEG2000, plusieurs propositions d'applications ont été annoncées parmi lesquelles : le cinéma numérique avec la partie Motion-JPEG2000 du standard **[37]**, l'imagerie médicale **[39]**, les appareils mobiles de troisième génération **[38]**. Dans **[34], [35], [36]**, on trouve aussi des implémentations de JPEG2000 visant les applications vidéo numérique. D'autres implémentations sur différentes technologies **[34], [36], [40],** ont été proposées sans aucune indication précise sur le type d'applications visées. En général, la majorité des travaux sur JPEG2000 sont restés au stade d'études sans réelle concrétisation au travers d'applications.

9. Conclusion

Nous avons présenté dans ce chapitre, d'abord une vue générale du standard de compression JPEG2000 avec toutes ses spécificités et ses fonctionnalités. Les principales

parties de la chaîne de compression sont détaillées dans ce chapitre. Ensuite on retrouve l'état de l'art des implémentations logicielles et matérielles de JPEG2000, ainsi qu'une étude au niveau microarchitectural des ressources utilisées par des codeurs JPEG2000, sur des plateformes du commerce. Enfin on a résumé les propositions et les études visant l'application de JPEG2000 particulièrement la partie Motion-JPEG2000. Ce chapitre nous permettra d'en dériver des conclusions pour notre approche d'implémentation optimisée de JPEG2000.

Dans la suite on présente l'état de l'art des méthodologies de conception des systèmes sur puce, particulièrement les algorithmes de partitionnement HW/SW. Au début de cette étude on introduit les plateformes reconfigurable. Ensuite, on donne une idée globale sur les méthodologies de conception SOC, leur classement, ainsi que leurs évolutions. Enfin, on consacre une partie pour les méthodologies de conception SOPC.

10. Références

[1] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, Image coding using the wavelet Transform, IEEE Trans. Image Processing, pp 205-220, April 1992.

[2] M. D. Adams, Reversible wavelet transforms and their application to embedded image compression, M.S. thesis, Univ. Victoria, Canada, 1998.

[3] M. Boliek, C. Christopoulos and E. Majani (editors), "JPEG2000 Part I Final Draft International Standard," WG1 N1855, August 18, 2000.

[4] V. Bhaskaran and K. Konstantinides, "Image and Video Compression Standards: Algorithms and Applications", 2 nd Ed., Kluwer Academic Publishers, 1997.

[5] A. Bovik, Ed., Handbook of Image & Video Processing. San Diego, CA: Academic, 2000.

[6] A. Bilgin, P. J. Sementilli, F. Sheng, and M. W. Marcellin, "Scalable image coding using reversible integer wavelet transforms", IEEE Trans. Image Processing, vol 9, pp 1972-1977, Nov. 2000.

[7] C. Chrysafis and A. Ortega, "An Algorithm For Low Memory Wavelet Image Compression", Proc. IEEE Int. Conf. Image Processing, Vol. III, pp. 354-358, Kobe, Japan, Oct. 1999.

[8] C. Chrysafis and A. Ortega, "Line-Based, Reduced Memory, Wavelet Image Compression," IEEE Trans. Image Processing, Vol. 9, No 3, pp. 378-389, March 2000.

[9] C. A. Christopoulos, A. N. Skodas, and T. Ebrahimi, "The JPEG 2000 still image coding system : An overview", IEEE Trans. Consumer Electron., vol. 46, pp 1103-1127, Nov. 2000.

[10] M. C. Larabi, N. Richard and C. Fernandez-Maloigne, "Core experiment result on color spaces", in ISO/IEC JTC1/SC29/WG1 N21, JPEG 2000, 24 mars 2001.

[11] D. Le Gall and A. Tabatabai, "Subband Coding of Digital Images Using Symmetric Short Kernel Filters and Arithmetic Coding Techniques", Proc IEEE Int. Conf. ASSP, NY, pp. 761-765, 1988.

[12] M. J. Gormish, D. Lee, and M. W. Marcellin, "JPEG 2000 : Overview, architecture and applications",in Proc. IEEE Int. Conf. Image Processing (ICIP 2000), vol II, pp 29-32, Vancouver, Canada, 10-13 Sept. 2000.

[13] V. K. Goyal, "Theoretical doundations of transforming coding", IEEE Sig. Proc. Magazine, vol. 18, pp 9-21, Sept. 2001.

[14] J. Kovacevic and W. Sweldens, "Wavelet Families of Increasing Order in Arbitrary Dimensions", IEEE Trans. Image Processing, Vol. 9, No. 3, pp. 480-496, March 2000.

[15] E. Martinez-Uriegas, "Multiple Color Spaces and Component Transforms" in JPX : Update from WG1-1745 for Part II, ISO/IEC JTC1/SC29/WG1 N1818, JPEG 2000, Jul-24-2000.

[16] M. W. Marcellin, M. Gormish, A. Bilgin, M. Boliek, "An Overview of JPEG 2000," Proc. IEEE Data Compression Conference, pp 523-541, Snowbird, Utah, March 2000.

[17] M. J. Nadenau and J. Reichel, "Opponent Color, Human Vision and Wavelets fir Image Compression", Proc. Of the 7th Color Imaging Conference, pp. 237-242, Scottsdale, Arizona, November 16-19, 1999.

[18] Todd Newman, "Specifying the color space of encoded images", ISO/IEC JTC1/SC29/WG1 N1339, JPEG 2000, 7 July 1999.

[19] ISO/IEC JTC1/SC29/WG1 N575, "JPEG-LS (14495) Final CD", July 1997.

[20] "Progressive Lossy to Lossless Core Experiment with a region of Interest: Results with the S, S+P, Two-Ten Integer Wavelets and with the Difference Coding Method", ISO/IEC JTCI/SC29/WGI N741, Mars 1998.

[21] "Core experiment on Impoving the Performance of the DCT: Results with the Visual Quantization Method, Deblocking Filter and Pre/Post Processing", ISO/IEC JTCI/WGI N742, Mars. 1998.

[22] "JPEG 2000, Requirements and Profiles", ISO/IEC JTCI/SC29/WGI N1271, Mars 1999.

[23] Press Release of the 23rd WGI Singapore Meeting, ISO/IEC JTCI/SC29/WGI N1959, 8 Dec. 2000.

[24] W. B. Pennebaker and J. L. Mitcell, "JPEG: Still Image Data Compression Standard", Van Nostrand Reinhold, New York, 1993.

[25] K. R. Rao and J. J. Hwang, "Techniques and Standards for Image, Video and Audio Coding", Englewood Cliffs, NJ: Prentice-Hall, 1996.

[26] D. Santa Cruz and T. Ebrahimi: "An Analytical Study of the JPEG2000 Functionalities", in Proc. IEEE Int. Conf. Image Processing (ICIP 2000), vol. II, pp 49-52, Vancouver, Canada, 10-13 Sept. 2000.

[27] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard", IEEE Signal. Processing Mag., vol. 18, pp 36-58, Sept. 2001.

[28] W. Sweldens, “The Lifting Scheme: A Custom-Design Construction of Biorthogonal Wavelets”, Appl. Comput. Harmonic Analysis, vol. 3, no. 2, pp 186-200, 1996.

[29] W. Sweldens, “The Lifting Scheme: Construction of Second Generation Wavelets,” SIAM J. Math. Anal., vol. 29, no. 2, pp 511-546, 1997.

[30] B. E. Usevitch, A tutorial on modern lossy wavelet image compression: Foundations of JPEG 2000, IEEE Signal. Processing Mag., vol. 18, pp 22-35, Sept. 2001.

[31] M. Vetterli, Wavelets, approximation and compression, IEEE Signal. Processing Mag., vol. 18, pp 59-73, Sept. 2001.

[32] InSilicon : http://www.insilicon.com/products/ProductPages/JPEG/jpeg_2k_codec.shtml

[33] D.Taubman and M.W.Marcellin, “JPEG2000 – Image Compression Fundamentals, Standards and Practice”, Kluwer Academic Publishers, Nov. 2001.

[34] Hung-Chi Fang, Chao-Tsung Huang, Yu-Wei Chang, Tu-Chih Wang, Po-Chih Tseng, Chung-Jr Lian, Liang-Gee Chen, “81MS/s JPEG 2000 Single Chip Encoder with Rate Distortion Optimization”, IEEE International Solid-State Circuits Conference, ISSCC 2004, San Fransisco, February 15-19, 2004.

[35] Hideki Yamauchi, Kenji Mochizuki, Kazuhiko Taketa, Tsuyoshi Watanabe, Tsugio Mori, Yuh Matsuda, Yoshifumu Matsushita, Akio Kobayashi, Shigeyuki Okada, “A 1440x1080 Pixels 30 Frames/sec Motion-JPEG2000 Codec for HD Movie Transmission”, IEEE International Solid-State Circuits Conference, ISSCC 2004, San Fransisco, February 15-19, 2004.

[36] Yamauchi, H.; Okada, S.; Taketa, K.; Ohyama, T.; Matsuda, Y.; Mori, T.; Watanabe, T.; Matsuo, Y.; Yamada, Y.; Ichikawa, T.; Matsushita, Y, “Image processor capable of block-noise-free JPEG2000 compression with 30 frames/s for digital camera applications”; IEEE International Solid-State Circuits Conference, 2003, Digest of Technical Papers, ISSCC,, 2003, Page(s):46 - 477 vol.1, 2003.

[37] Fossel, S.; Fottinger, G.; Mohr, J, “Motion JPEG2000 for high quality video systems”. Consumer Electronics, IEEE Transactions on Volume 49, Issue 4 Page(s):787 - 791, Nov. 2003.

[38] M.; Djafarian, K.; Chaoui, J.; Mazzocco, D.; Masse, Y, “Enabling JPEG2000 on 3G wireless mobiles”, through OMAP/spl trade/ architecture Peresse Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02). IEEE International Conference on Volume 4, Page(s): IV-3796 - IV-3799 vol.4, 13-17 May 2002.

[39] Anastassopoulos, G.K.; Tsalkidis, A.D.; Stephanakis, I.; Mandellos, G.; Simopoulos, K “Application of JPEG 2000 compression in medical database image data”; Digital Signal Processing, 2002. DSP 2002. 2002 14th International Conference on Volume 2, Page(s):539 - 542 vol.2, 1-3 July 2002.

[40] Heng-Ming Tai; Men Long; Su Yang; Dawei Zhou, “Implementation of JPEG2000 codec on a fixed-point DSP”, Consumer Electronics, 2001. ICCE. International Conference on, Page(s):128 - 129, 19-21 June 2001.

Chapitre III Méthodologies de Conception

1. Introduction

Les systèmes électroniques sont de plus en plus complexes vu la capacité d'intégration. Cette capacité d'intégration permet de mettre des systèmes entiers sur une seule puce, ce qui rend la conception et la vérification de ce type de systèmes difficile et coûteuse. En même temps les moyens et les outils de conception ne connaissent pas la même évolution. Comme le montre la figure suivante, le gap entre la capacité de conception et celle d'intégration se creuse sans cesse. Cette complexité pousse donc les concepteurs à chercher des méthodes qui leur permettent de concevoir et de valider leurs systèmes pendant des périodes qui sont de plus en plus courtes (respecter le time to market : TTM).

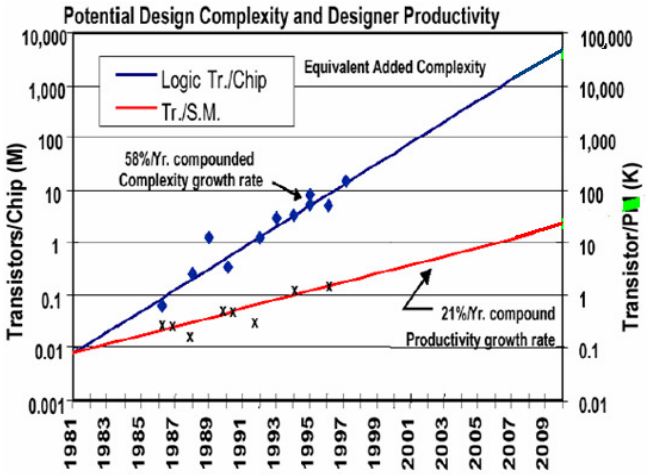


Figure Ch III - 1 : Ecart entre la capacité de conception et celle d'intégration

Sur la figure suivante, on peut voir que quelle que soit la technologie utilisée, la phase de conception et de validation est la plus coûteuse. De plus l'impact de cette phase sur le reste du processus de fabrication est vital. Pour cette raison, différents efforts sont déployés dans le but de trouver de meilleures méthodes de conception. En effet différentes méthodologies de conception sont proposées. Certaines sont basées sur la standardisation d'autres sur la réutilisation, les plus récentes sont basées sur la reconfigurabilité.

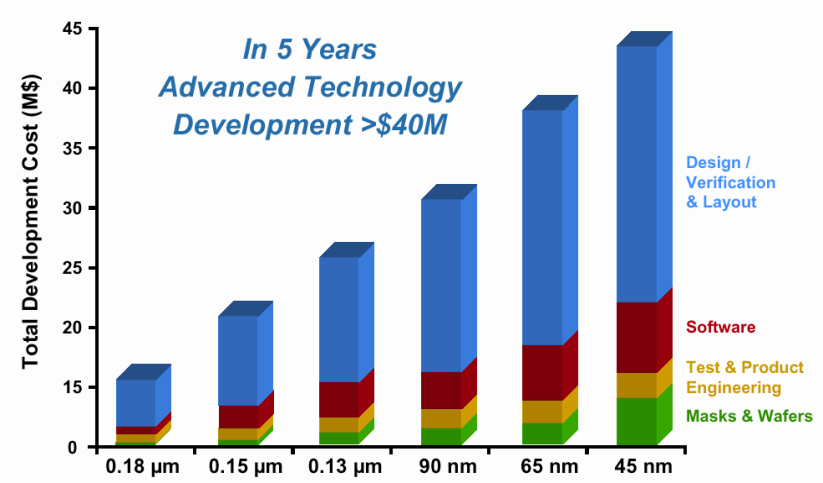


Figure Ch III - 2 : Evolution du coût de la conception et de la validation par rapport à celui de la technologie

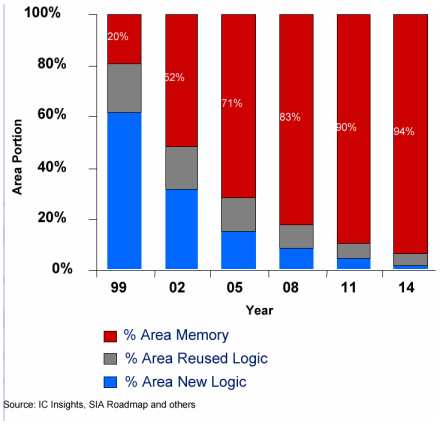


Figure Ch III - 3 : Evolution de la taille de la mémoire dans les SOC

2. Plateformes reconfigurables

L'utilisation des circuits programmables a commencé par les composants PROM (Programmable Read-only Memory), pour lesquels le bus d'adresse est utilisé comme entrée du circuit logique, et le bus de donnée comme sortie. L'insuffisance de ce circuit pour la réalisation d'architectures logiques a poussé les concepteurs à trouver une alternative qui s'est concrétisée par l'apparition des FPLA (Field Programmable Logic Array). Le coût et la faible performance de ces circuits ont favorisé le développement des PAL (programmable Array Logic). Toutes les variétés des composants PAL ont été regroupées sous le terme SPLD (Simple Programmable Logic Devices). Pour augmenter la capacité en logique de l'architecture des SPLD plusieurs composants sont intégrés dans la même puce d'où l'apparition des Complex PLD. La difficulté d'étendre l'architecture du CPLD à des densités plus élevées rend une nouvelle approche indispensable, d'où l'apparition des FPGA (Field Programable Gate Array).

Depuis leur lancement au début des années 80, les circuits programmables ont révolutionné le domaine de l'électronique en apportant un plus de par leur simplicité d'utilisation, ainsi que leur coût sur le marché. Cette nouvelle génération de circuits a, ainsi, permis ainsi aux ingénieurs, qu'ils soient dans le domaine de la recherche ou dans le domaine de l'application, d'être plus indépendant des usines de microélectronique pour tester leurs applications et mener un projet exigeant des circuits spécifiques à terme.

Ces circuits à logique programmable, permettent d'implémenter n'importe quel algorithme en utilisant des outils de conception et un langage de programmation fourni par le constructeur.

L'une des premières firmes à avoir mis sur le marché ce type de circuit et qui détient la plus grosse part de l'offre, est la société Xilinx, elle est américaine et existe depuis les années 80. Cependant il existe d'autres sociétés qui se sont alignées sur le même principe de circuits que la première.

2.1 Les circuits FPGA

L'architecture des FPGA comme le montre la **figure Ch III - 4** est semblable à celle des Mask-Programmable Gate Array (MPGA). Elle est constituée d'un ensemble de blocs logiques qui peuvent être interconnectés par programmation pour réaliser différents circuits. La différence majeure entre FPGA et MPGA est que le MPGA est programmé à l'aide d'interconnexion métal alors que les FPGA sont programmés via des interrupteurs électriquement programmables.

2.1.1 Les blocs logiques

Un bloc logique du FPGA peut être aussi simple qu'un transistor ou aussi complexe qu'un microprocesseur. Il est typiquement capable d'implémenter plusieurs fonctions logiques combinatoires et séquentielles. Les FPGA utilisent des blocs logiques qui sont basés sur un ou plusieurs des éléments suivants :

- Paires de transistors
- Porte logique telles que les porte NAND et XOR a deux entrées
- Des multiplexeurs
- Des look-up table (LUT)
- Une structure AND-OR

2.1.2 Les technologies de programmation

Il existe trois types de technologies d'interrupteur programmable :

- SRAM où l'interrupteur est un transistor contrôle par un bit de la SRAM
- Antifuse lorsqu'il est programmé, la résistance entre les deux composants terminaux devient faible.
- EPROM où l'interrupteur est un transistor à porte flottante qui peut être contrôlée en injectant une charge dans sa porte flottante

2.1.3 L'architecture du routage

L'architecture de routage dans un FPGA est la manière selon laquelle les interrupteurs programmables et les segments de raccordement sont positionnés pour permettre l'interconnexion des blocs logiques.

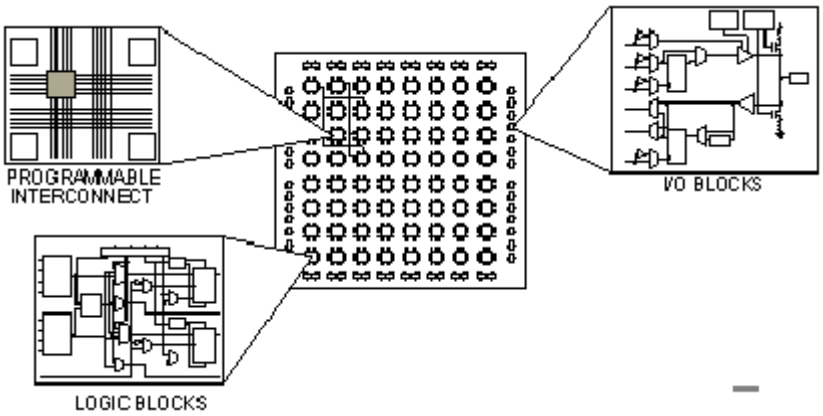


Figure Ch III - 4 : Architecture interne d'un circuit FPGA

Dans **[33]** on trouve un suivi des architectures FPGA présentées par un nombre de fabricants de composant tel que Actel, Xilinx et Altera.

2.2 Exemple de circuit FPGA

Comme dans le reste de ce rapport on ne s'intéresse qu'à la technologie Xilinx on limitera nos exemples aux architectures des différentes familles de composant que propose Xilinx. Pour faire une comparaison avec les autres technologies on propose de consulter **[33], [34], [35].**

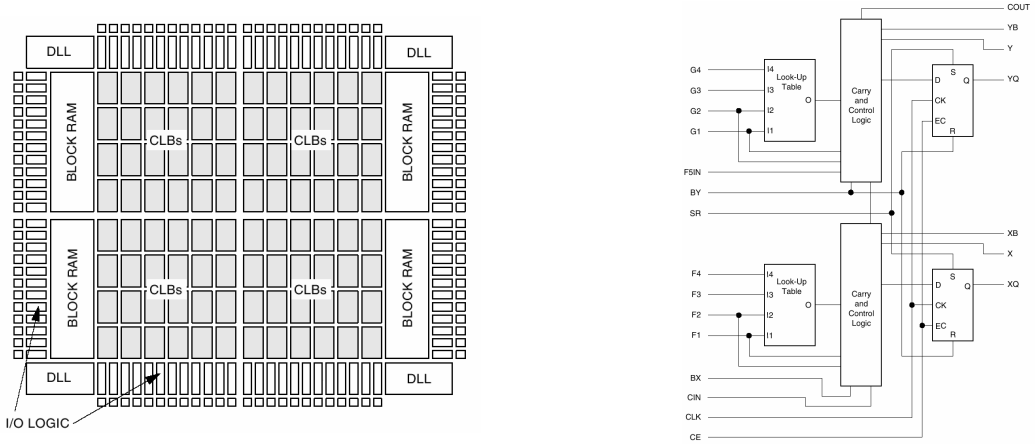
Pour les familles FPGA de Xilinx les critères qui permettent de caractériser un composant sont :

- Les portes logiques (System Gates) : c'est l'ensemble de portes logiques qui ont servi à construire tous les éléments d'un FPGA. Ils incluent aussi bien la logique programmable que les blocs de RAM ainsi que le reste des éléments.
- Les cellules logiques (Logic Cells) : elles contiennent un générateur de fonction, de la carry logic, et un élément de stockage.
- Les blocs logiques (CLB array) : les CLB (Configurable Logic Blocks) sont les éléments fonctionnels pour la construction de la logique. Ils contiennent un certain nombre de Logic Cells (LC) selon la famille du FPGA.
- Les entrées/Sorties utilisateur (Maximum available User I/O) : ils présentent le nombre maximum de pins que l'utilisateur peut utiliser comme entrées et/ou comme sorties. Ce nombre varie selon le paquetage et la densité de la logique du composant.
- La RAM distribuée (Total Distributed RAM Bits) : c'est la taille maximale de la RAM qui peut être implémentée en utilisant les CLB
- Les blocs de RAM (Total Block RAM bits) : c'est de la RAM embarquée. Ce sont des blocs de RAM avec des tailles en bit fixes pour une famille de FPGA. Le nombre de blocs varie d'un composant à l'autre dans la même famille et entre les familles de FPGA.

En parlant des FPGA on peut aussi rencontrer des termes comme le nombre de Slices, les DLL (Delay-Locked Loops). Un CLB est généralement constitué par un certain nombre de slices qui ne sont en fait que des groupements de LC identiques. Les DLLs sont utilisés pour compenser les délais de l'horloge, la correction du rapport cyclique et le contrôle du domaine de l'horloge (division, multiplication, déphasage).

2.2.1 Spartan-II

Les composants Spartant-II sont fabriqués avec la technologie 0.18 micron. Ils offrent une densité variant de 15000 à 20000 portes logiques et peuvent atteindre une fréquence de fonctionnement de 200MHz. L'architecture du Spartan-II est donnée par la **figure Ch III - 5**. Un CLB est constitué par deux slices dont chacun est formé par 2 LC. Le nombre maximum d'I/O utilisateur est de 284. Le nombre de bloc de RAM varie entre 4 et 14 blocs. Chaque bloc de RAM a une taille de 4Kbits avec une largeur de bus de données maximale de 16 bits (➔256 mots de 16 bits) **[30]**.

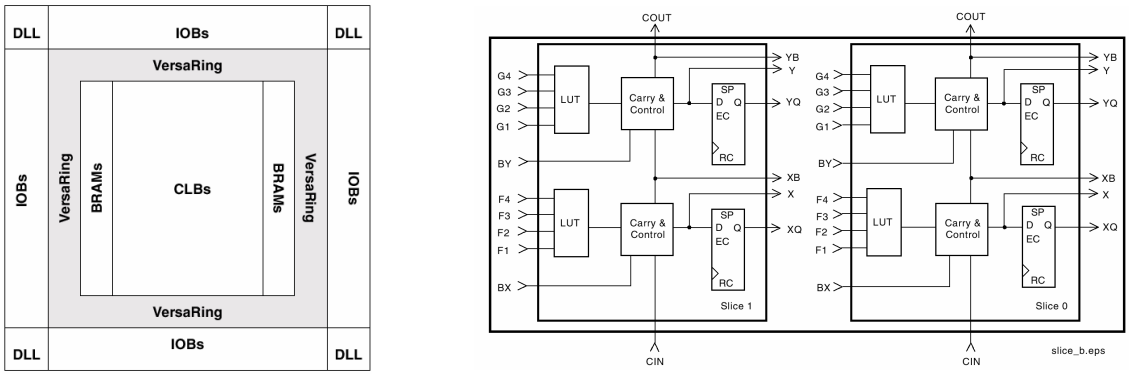


(a) (b)

Figure Ch III - 5 : (a) Architecture du Spartan II, (b) Architecture d'un Slice (deux Slice identiques par CLB)

2.2.2 Virtex

Les composants Virtex présentent une meilleure densité que la famille Spartan-II. Ils offrent une densité variant de 56K à 1M portes logiques et peuvent atteindre une fréquence de fonctionnement de 200MHz. Ils sont fabriqués avec la technologie 0.22 micron 5 couches de métal. L'architecture du Virtex est donnée par la **figure Ch III - 6**. Chaque CLB est constitué par deux slices dont chacun est formé par 2 LC. Le nombre de blocs de RAM varie entre 8 et 32 blocs. Chaque bloc de RAM a une taille de 4Kbits avec une largeur de bus de données maximale de 16 bits (➔256 mots de 16 bits). Le nombre maximum d'I/O utilisateur est de 512 **[31]**.

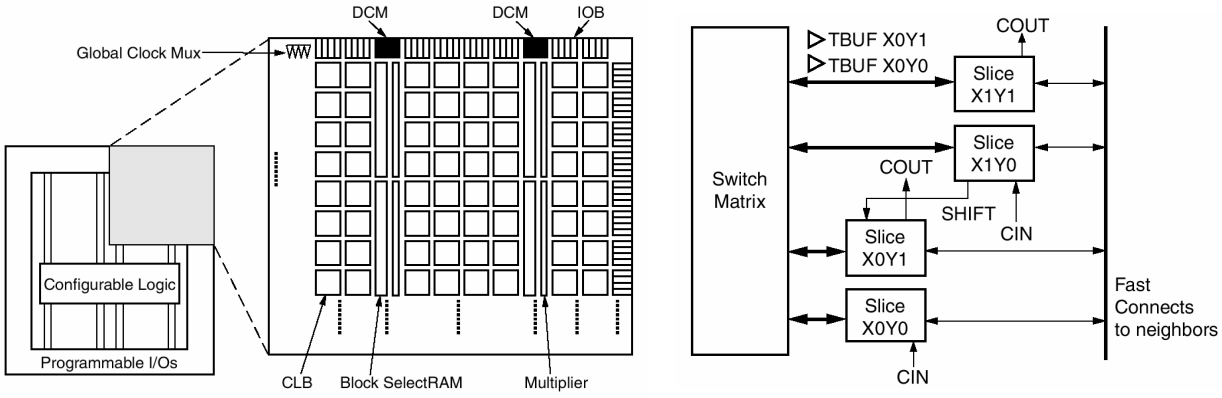


(a) (b)

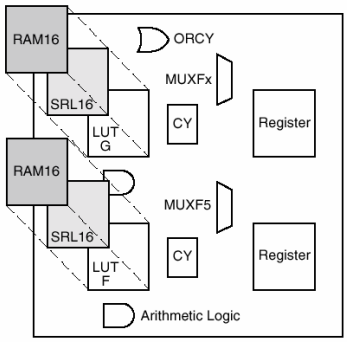
Figure Ch III - 6 : (a) Architecture du Virtex, (b) Composition d'un CLB (2 Slices)

2.2.3 Virtex-II

Les composants Virtex-II présentent une meilleure densité que les familles Spartan-II et Virtex. Ils offrent une densité variant de 40K à 8M portes logiques et peuvent atteindre une fréquence de fonctionnement de 420MHz. Ils sont fabriqués avec la technologie 0.15 micron 8 couches de métal avec des transistors haute performance 0.12 micron. L'architecture du Virtex-II est donnée par la **figure Ch III - 7**. Chaque CLB est constitué par 4 slices. Chaque slice contient un générateur de fonction, de la carry logic, des portes logiques arithmétiques, un large multiplexeur et deux éléments de stockages. Le nombre de blocs de RAM varie entre 4 et 168 blocs. Chaque bloc de RAM a une taille de 18Kbits avec une largeur de bus de données maximales de 36 bits (➔512 mots de 36 bits). Ces mémoires sont à porte double (Dual-Port). Le nombre maximum d'I/O utilisateur est de 1108. Les composants Virtex-II possèdent aussi des multiplieurs 18x18 bits (de 4 a 168 multiplieurs), et jusqu'à 12 DCM (Digital Clock Manager) **[32]**.



(a) (b)



(c)

Figure Ch III - 7 : (a) Architecture du Virtex-II, (b) Les éléments d'un CLB, (c) Le Slice d'unVirtex-II.

Ces même familles de composants FPGA possèdent leurs équivalents avec une extension des blocs de RAM et on les distingue par le « E » ajouté à leurs noms respectifs : SpartanE, VirtexE,….

D'autres familles de FPGA sont développées par Xilinx telles que le Virtex-II-Pro (voir **[18]** et le chapitre VI) et le Virtex 4 **[18]**.

3. Les méthodologies de conception SOC

Pour commencer on doit donner la définition de l'élément essentiel de ce chapitre qui est le SoC. Qu'est ce qu'un SOC (System-on-Chip) ou système sur puce ? Différentes définitions ont été données, dont la plus pertinente est celle qui définit un SoC en tant que système ASIC complexe et indépendant sur une seule puce qui contient au moins un processeur. Peu ou pas de périphérique externe viennent compléter le fonctionnement du système **[1]**.

Un SOC peut être défini comme un IC (Integrated Circuit) complexe qui intègre la majorité des fonctionnalités d'un produit final sur une seule puce. En général un SOC incorpore un processeur, de la mémoire sur puce, et des fonctions d'accélération. Il présente des interfaces avec des composant périphériques. La conception SOC combine des composants logiciels (SW) et matériels (HW). Parce que la conception SOC peut être interfacée avec le monde réel, il peut souvent incorporer des composants analogiques, et peut être dans le futur des systèmes opto/microélectroniques mécaniques.

La Virtuel Socket Interface Alliance (VSIA), formé en 1996 pour entretenir le développement et la reconnaissance des standards pour la conception et l'intégration des blocs réutilisable d'IP (Intellectual Property), définit une puce système comme « un composant hautement intégré. Il est aussi un système sur silicium, un système sur puce, un système LSI (Large Scale Integrated), un système ASIC, et aussi comme un composant à intégration au niveau système (System-Level Integration : SLI device). Dataquest a défini un composant SLI en tant que : composant contenant plus que 100 mille portes avec au moins un élément programmable et de la mémoire sur puce. Plus récemment Dataquest présente un SLI ou un SOC en tant que : circuit intégré dédié à une application spécifique qui contient un module de calcul (microprocesseur, DSP,…), de la mémoire, et de la logique sur une seule puce.

Concevoir un système sur puce signifie généralement un important travail manuel et une grande expertise pour choisir l'architecture, concevoir les interfaces, écrire les modules de gestion de périphériques et/ou configurer les systèmes d'exploitation commerciaux. La tâche la plus difficile est de faire fonctionner l'ensemble de ces éléments qui sont conçus sur mesure pour les besoins d'une application particulière. L'organisation VSIA essaye de réaliser une méthode basée sur l'assemblage automatique d'IP préconçus. Un grand nombre de méthodologies adopte ce concept en utilisant des composants et des interfaces standard de

HW et de SW. Les pénalités de performances de la solution architecturale choisie sont acceptées en raison de la nécessité de répondre à des pressions toujours croissantes de temps de mise sur le marché.

Plusieurs travaux ont ainsi émergés proposant d'accélérer le flot de la conception avec de plus en plus d'automatisation à travers de nouveaux genres d'outils capables de traiter la conception des systèmes sur puce. Ces outils se concentrent sur l'automatisation du raffinement de la communication et la réutilisation de blocs préconçus avec la génération automatique des interfaces.

Un flot de conception standard est généralement constitué de trois étapes principales : (1) la spécification, (2) le partitionnement, et (3) la vérification conjointe HW/SW. La figure suivante représente un flot de conception standard.

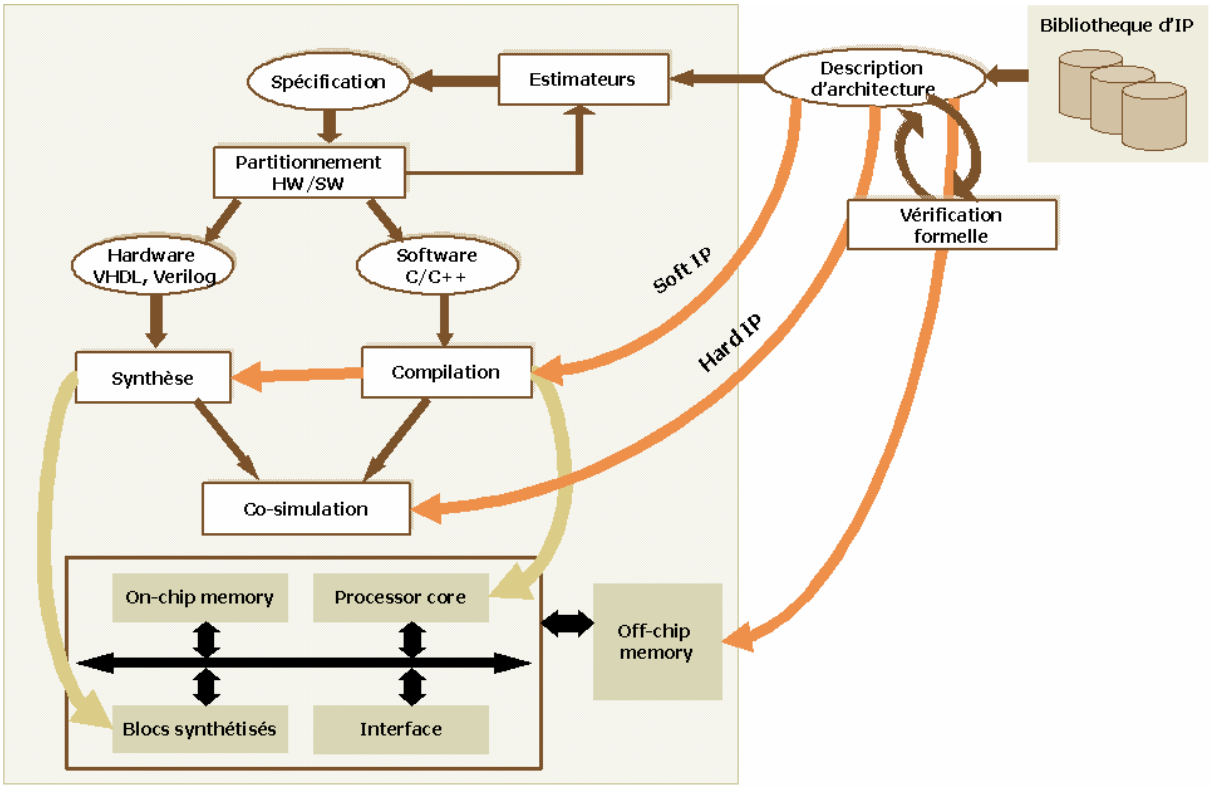


Figure Ch III - 8 : Flot de conception standard

Parmi les flots de conception SOC on trouve l'outil MCSE (Méthodologie de Conception de Systèmes Electroniques) [2], [3] qui présente une méthodologie pour la conception des systèmes électronique HW/SW en partant de plus haut niveau d'abstraction. Elles proposent une approche descendante pour la spécification ascendante pour l'implémentation basée sur plusieurs niveaux d'abstraction et couvrant un grand nombre de contraintes du système. L'architecture cible se compose d'une partie HW d'une partie SW et d'une partie mixte qui est traitée par une approche de conception conjointe (codesign) proposée par l'outil MCSE.

Comme les SOC peuvent être des ASIC on peut donc se référer aux méthodologies de conception des puces ASIC. L'exemple de flot de conception d'IBM présenté ci-dessous donne une vue globale des principales étapes de la conception d'un ASIC.

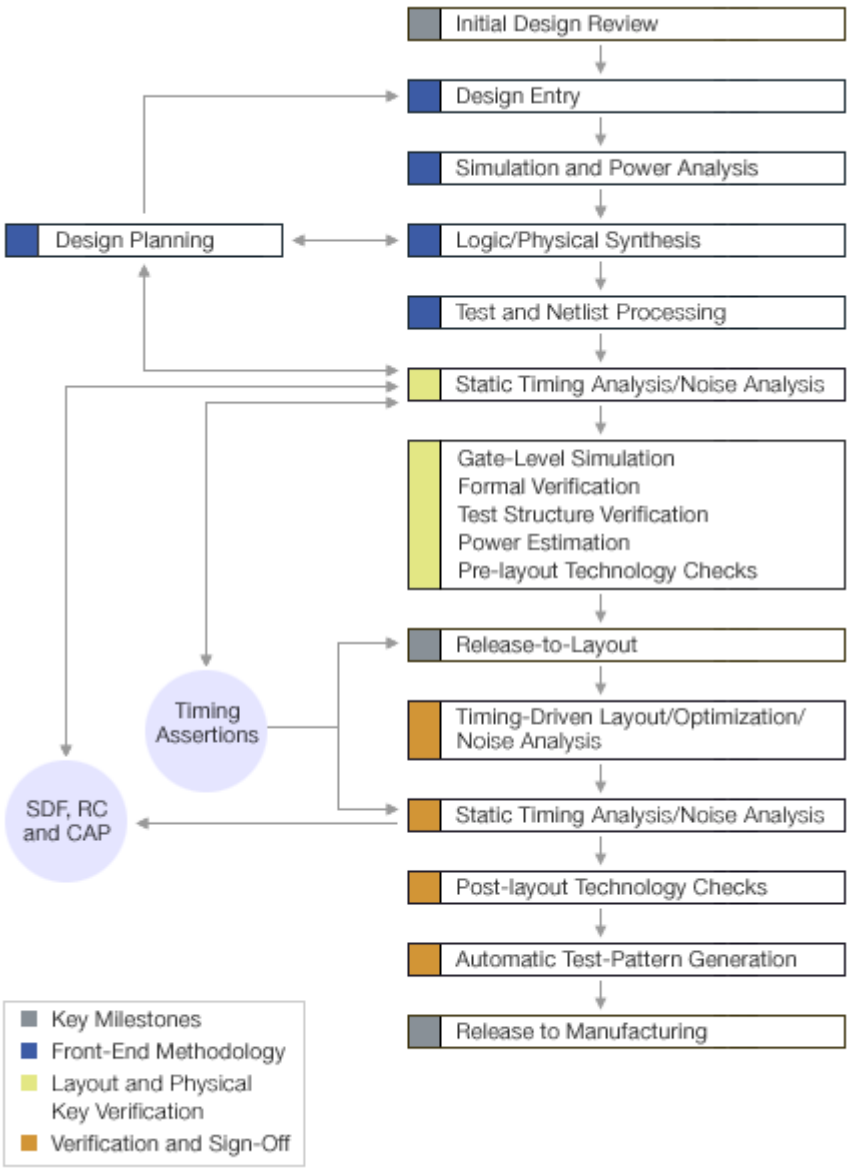


Figure Ch III - 9 : Flot de conception IBM

3.1 Le codesign HW/SW

Plusieurs grands groupes de recherche ont proposé ou mis au point d'autres méthodologies de conception conjointe. Les méthodologies les plus connues sont résumées dans le **tableau Ch III – 1 [28]**.

Dans le flot de conception, la partie qui nous concerne le plus dans le cadre de cette thèse est le partitionnement HW/SW et les méthodologies de conception conjointes. Cette étape est déterminante dans la conception d'un SOC car elle fixe l'architecture finale du système ainsi que sa performance. Plusieurs algorithmes de partitionnement ont été proposés pour automatiser cette étape du flot de conception.

Tableau Ch III - 1 : Résume des techniques de codesign

	Langage de spécification	Technique de découpage	Architecture	Validation logicielle/matérielle
Ptolemy	FDS (Flot de données synchrone)	Découpage automatique oriente logiciel ; optimisation du temps global d'exécution	Mono processeur	Processus légers communicant à travers des portes (sockets) ; communication synchrone
Cosyma	Cx (C parallèle)	Découpage automatique oriente logiciel, basé sur des estimations dynamiques de performance	Mono processeur	Processus communicant à travers le modèle CSP ; utilisation de portes (Socket)
SpecSyn	SpecChart	Découpage automatique basé sur la technique de groupement	Multi processeur	Processus communicant à travers le modèle CSP
Co-Saw	CSP	Découpage interactif base sur la technique de groupement	Mono processeur	Sockets du système UNIX
Corba	VHDL	Découpage non automatique oriente matériel	Mono processeur	Simulation dirigée au VHDL. Le logiciel est décrit en langage assembleur
Vulcan II	HardwareC	Découpage automatique oriente matériel	Mono processeur	Co-simulation au niveau assembleur ; communication dirigée par des évènements discrets
Rassp	VHDL	Découpage non automatique oriente matériel	Mono processeur	Co-simulation à base de modèles détaillés des processeurs et ASIC au niveau des portes
Lycos	VHDL ou C	Découpage manuel	Mono processeur	Processus communicant
Tosca	SpeedChart	Découpage manuel	Mono processeur	Validation par prototypage physique
Codes	SDL, StateChart	Découpage manuel	Mono processeur	Validation par prototypage physique
MCSE	Formalisme MSCE	Découpage interactif base sur des estimations de performance	Multi processeur	Co-simulation au niveau modèle suivie par simulation VHDL et C++
CoWare	POPE/SystemC	Synthèse des interfaces et raffinement des processeurs	Multi processeur	Validation par prototypage physique

3.2 Les algorithmes pour le partitionnement HW/SW

Dans cette partie on présentera quelques algorithmes de partitionnement HW/SW. Ces algorithmes sont basés sur plusieurs heuristiques et techniques d'optimisation. Ces différents algorithmes se différencient par leurs objectifs. L'algorithme **MAGELLAN** par exemple, vise à minimiser la latence des graphes de tâches. L'algorithme **COSYN** optimise, en plus des critères communs tels que la surface et le temps d'exécution, la consommation d'énergie.

L'algorithme **MIBS** est une combinaison entre le mapping et le choix de l'implémentation de chaque tâche. Ce dernier utilise l'algorithme **GCLP** qui est un algorithme de référence (cet

algorithme est réutilisé un peu partout : **MAGELLAN**, **MIBS**,…. GCLP est développé par Asawaree Kalavade, et Edward A.Lee dans le cadre du projet Ptolemy.

Dans une autre catégorie d'algorithme on retrouve l'algorithme génétique **MOGAC** qui utilise une technique différente de celle des précédents algorithmes.

3.2.1 L'algorithme GCLP

Cet algorithme résout le problème de partitionnement binaire. A chaque étape (mapping d'un noeud du Directed Acyclic Graph) cet algorithme choisit l'objectif de mapping adéquat selon la valeur de Criticité Globale (Global Criticality : GC) et la phase locale (Local Phase : LP). La GC est une estimation de la Criticité du temps (time criticality) à chaque étape de l'algorithme. La valeur de GC est comparée à un seuil pour dire si le temps est critique ou non. La LP est une classification des nœuds. Cette classification est basée sur des caractéristiques intrinsèques. Les nœuds sont classés en trois types :

- Extrimities : ce sont les noeuds qui imposent a priori leur mapping,
- Repellers : ce sont les nœuds, qui selon leurs structures, ont une préférence de mapping.
- Normals : ce sont les nœuds qui ne sont ni extrimities ni repellers.

La préférence du mapping est quantifiée en accord avec le type du noeud, cette quantité est utilisée pour modifier le seuil. Pour plus d'information concernant cet algorithme on peut se référer à **[5]**, et **[6]**. Les performances du GCLP sont exposées dans **[5]**.

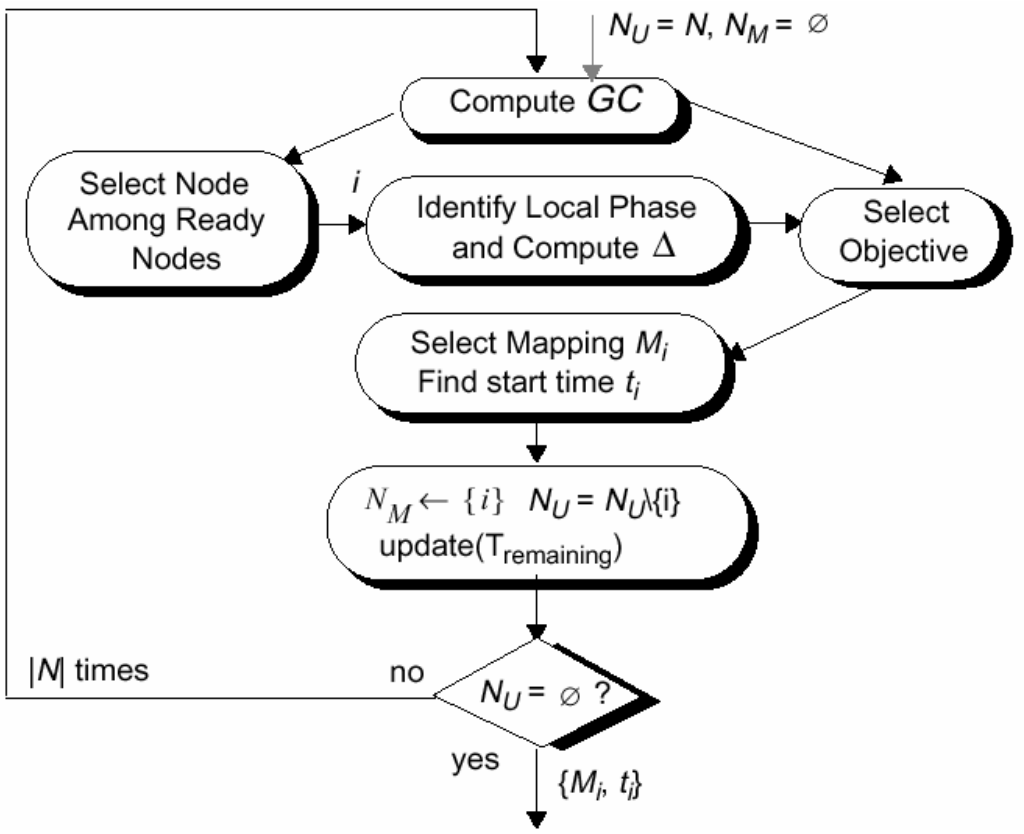


Figure Ch III - 10 : Algorithme GCLP

3.2.2 L'algorithme MAGELLAN

Cet algorithme est illustré par la **figure Ch III - 11**. MAGELLAN utilise comme entrée un graphe de tâches hiérarchiques de contrôle de flot de données (Hierarchical Control Dataflow Task Graph). Il supporte 5 types de tâches : tâche feuille (leaf task), tâche d'appel (call task), tâche cas (case task), tâche boucle (loop task), tâche hiérarchique (hierarchical task), et il fournit deux types supplémentaires : tâche contrôle (control task), tâche compteur (counter task). L'algorithme utilise un répartiteur (partitioner) et un Ordonnanceur (scheduler) de manière itérative. Le répartiteur et l'ordonnanceur appliquent une approche hiérarchique du type descendant (top-down). Une solution initiale est générée par l'ordonnanceur. Ce dernier effectue un mapping, une optimisation et un ordonnancement de l'application. Pour réaliser l'optimisation, l'Ordonnanceur utilise le déroulement de boucle (loop unrolling), le pipelining de boucle, l'ordonnancement spéculatif des éléments du chemin d'une tâche cas (case task), et l'optimisation du nombre d'implémentations d'une tâche d'appel. Le répartiteur tente ensuite d'améliorer cette solution en effectuant plusieurs mouvements. Le répartiteur exécute deux modes de mouvement : un mode hiérarchique, et un mode feuille (leaf). Le répartiteur reste en mode hiérarchique jusqu'à ce que il n'y ait plus d'amélioration ensuite il passe au mode feuille. MAGELLAN termine lorsque le répartiteur est incapable d'effectuer un nouveau mouvement.

Dans **[4]** on retrouve plus de détails sur le fonctionnement de cet algorithme ainsi que les résultats obtenus. Le cas d'étude qui a été traité dans **[4]** est JPEG.

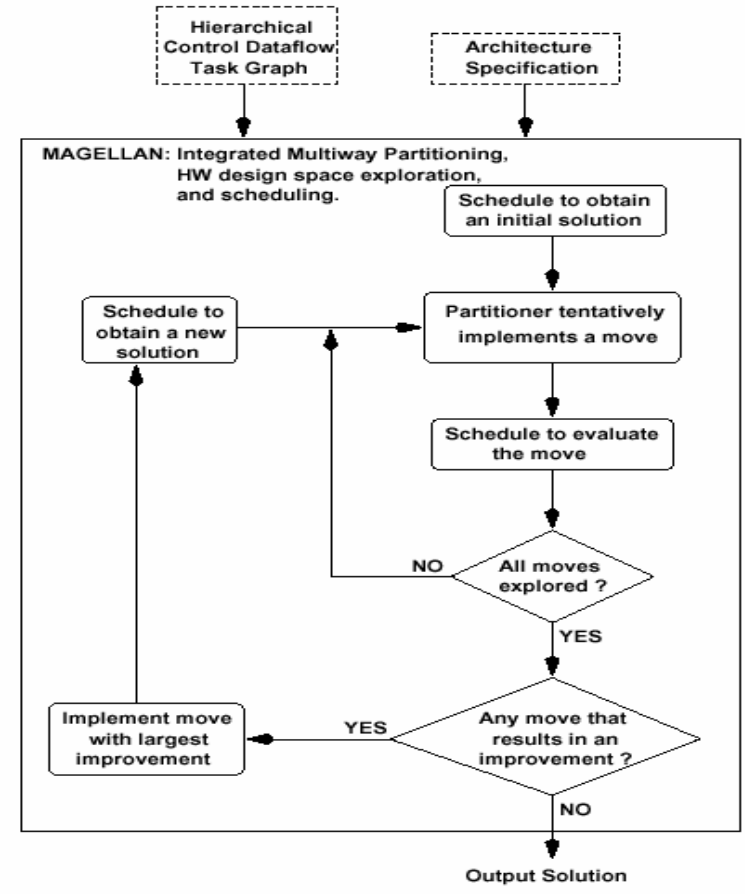


Figure Ch III - 11 : Algorithme MAGELLAN

3.2.3 L'algorithme MIBS

Cet algorithme utilise comme entrée un graphique acyclique dirigé (Directed Acyclic Graph DAG). Il fonctionne de manière itérative selon le schéma suivant :

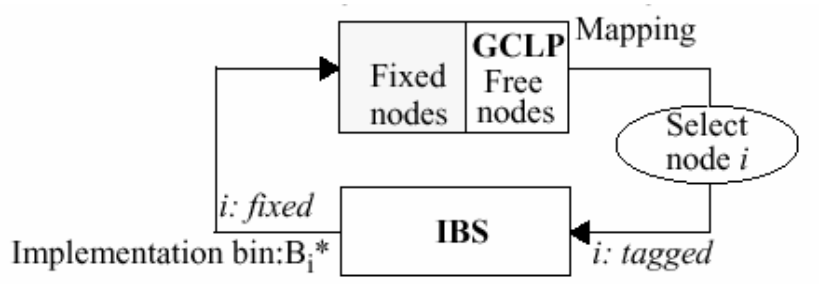


Figure Ch III - 12 : Algorithme MIBS

Il utilise les algorithmes : GCLP décrit précédemment, et l'algorithme IBS illustré par la **figure Ch III - 13**.

Un nœud du graphe de tâche est dit fixé si son mapping ainsi que son casier d'implémentation (implementation bin) ont été déterminés. Un nœud est dit libre si son mapping et son casier d'implémentation ne sont pas encore déterminés. Un nœud est dit étiqueté (tagged) s'il est mappé et reste à déterminer son casier d'implémentation.

Dans **[5]** on trouve plus de détails sur cet algorithme ainsi qu'une description des performances.

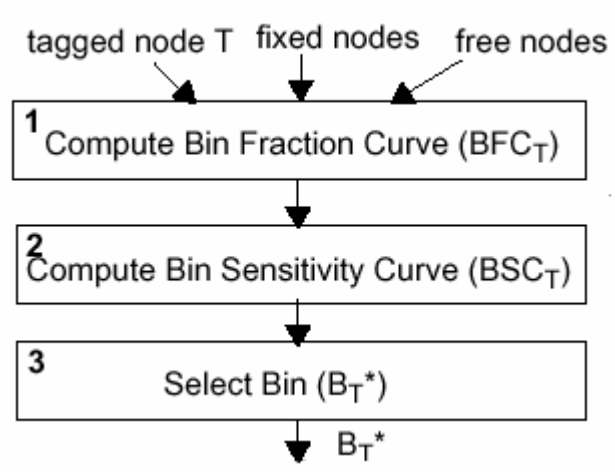


Figure Ch III - 13 : Algorithme IBS

3.2.4 L'algorithme COSYN

L'algorithme COSYN est le premier qui prend en compte l'optimisation de la consommation d'énergie. Cet algorithme commence par effectuer en première étape une analyse (parsing) du graphe de taches et des contraintes système/tâche et les bibliothèques, puis il crée une structure de données. L'étape de clustering permet le regroupement des tâches pour réduire l'espace de recherche, ce qui réduit considérablement la complexité de l'algorithme. Les clusters sont rangés par rapport à leur importance et leur priorité. L'étape d'allocation détermine le mapping des tâches aux éléments de traitement (PEs).

Cet algorithme contient deux boucles : une boucle externe pour l'allocation de chaque « cluster », une boucle interne pour l'évaluation de toutes les allocations pour chaque « cluster ». A chaque « cluster » l'algorithme crée et associe un tableau d'allocation. L'étape suivante est l'ordonnancement qui détermine l'ordre d'exécution des tâches et des

communications ainsi que le temps de fin de chaque tâche et de chaque communication. Cet algorithme supporte l'ordonnancement préemptif et non-statique. L'insertion de l'étape d'ordonnancement dans la boucle interne facilite la tâche de l'étape d'évaluation des performances. L'étape d'évaluation de l'allocation compare l'allocation courante avec les allocations précédentes.

Dans **[7]** on retrouve plus de détails sur les techniques utilisées par COSYN surtout pour l'optimisation de la consommation d'énergie.

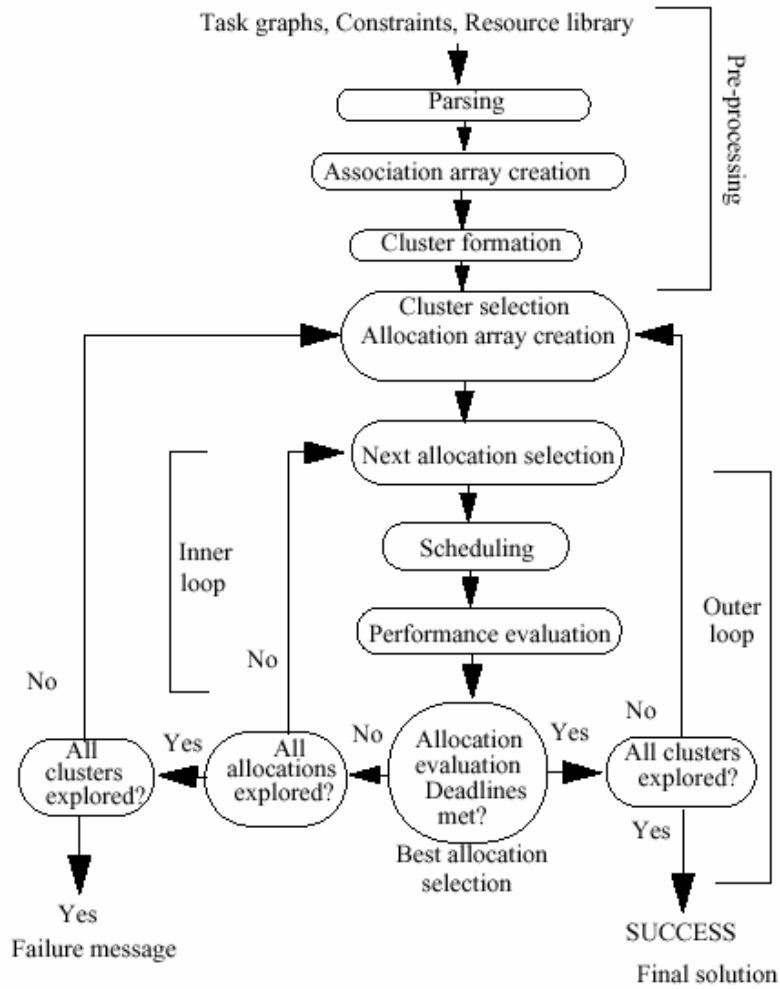


Figure Ch III - 14 : Algorithme COSYN

3.2.5 L'algorithme MOGAC

MOGAC est un système de co-synthèse HW/SW. Ce système est basé sur l'algorithme génétique décrit par la **figure Ch III - 15**.

Ce système fait évoluer en parallèle un ensemble de solutions. Chaque solution est initialisée de manière aléatoire. Après l'initialisation MOGAC évalue chaque solution : il détermine les coûts en fonction du prix et de la consommation d'énergie. Ces coûts sont ensuite comparés aux contraintes fixées par l'utilisateur pour déterminer le niveau de violation de ces contraintes. Les solutions sont par la suite triées en utilisant un critère multiobjectif (multiobjective criterion). Les solutions de rang bas sont terminées et ceux de rang élevé se reproduisent pour les remplacer. Des opérations de « Crossover » et de « Mutation » modifient les nouvelles solutions, d'où l'apparition d'une nouvelle génération. La condition d'arrêt n'est atteinte que si plusieurs générations sont terminées sans aucune

amélioration des solutions. Avant d'arrêter, MOGAC effectue un « prune » des solutions médiocres et remet le reste des solutions à l'utilisateur.

Dans **[8]** On trouve plus d'information sur le fonctionnement et les performances de l'algorithme MOGAC.

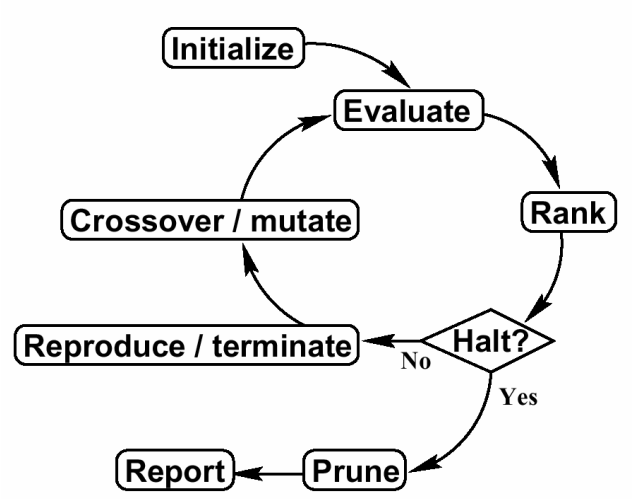


Figure Ch III - 15 : Algorithme MOGAC

3.2.6 Autres algorithmes

Cet algorithme est présenté dans **[6]**. Il a été conçu pour la résolution du problème de conception conjointe pour les applications multiples (Multi-application). Cet algorithme utilise des versions modifiées de l'algorithme GCLP. Il effectue d'abord une extraction de la mesure de « commonality » à travers les applications, puis il applique l'algorithme GCLP.

3.2.7 Quelques résultats

Dans ce paragraphe on essayera de récupérer quelques résultats à partir de la littérature. Ces résultats mettent en évidence les performances et les caractéristiques des algorithmes déjà présentés.

Dans le premier tableau (**tableau Ch III - 2**) on compare les performances de l'algorithme MIBS avec celles de ILP (Integer Linear Program). Pour le ILP a été formulé un problème de 15 nœuds avec 718 contraintes et 396 variables (dont 381 variables entières). Le ILP a été résolut à l'aide de CPLEX* sur une station SPARC **[5]**.

Tableau Ch III - 2 : Comparaison de ILP et MIBS

Scénario	Surface matérielle	Temps d'exécution
ILP	158	3.5 heures
MIBS	181	3 minutes
comparaison	1.1456 fois plus grand	70 fois plus rapide

Le deuxième tableau (**tableau Ch III – 3**) fait une comparaison entre l'algorithme COSYN et MOGAC pour l'exemple Parakash & Parker. La valeur entre crochet « < > » indique la durée la plus élevée pour la fin du graphe de tâche **[8]**.

* CPLEX est un logiciel d'optimisation développé par Ilog. Il résout des problèmes de programmation linéaires. http://www.ilog.com/products/cplex/

Tableau Ch III - 3 : Comparaison entre COSYN et MOGAC

Exemple <performance>	No. Tache	COSYN		MOGAC		
		Prix	Temps CPU(s)	Prix	Temps CPU(s)	Temps accordé (tuned time)(s)
Prakash & Parker1<4>	4	N.A	N.A	7	3.3	0.2
Prakash & Parker1<7>	4	5	0.2	5	2.1	0.1
Prakash & Parker2<8>	9	N.A	N.A	7	2.1	0.2
Prakash & Parker2<15>	9	5	0.4	5	2.3	0.1

D'après ces deux tableaux on remarque que le seul gain est au niveau temps CPU, par contre les résultats du partitionnement sont très similaires.

3.3 Classification des méthodologies de conception

Les méthodes de conception SOC utilisées de nos jours peuvent être divisées en trois familles : Timing Driven Design (TDD), Block-Based Design (BBD), et Platform-Based Design (PBD). Ces familles varient selon les technologies utilisées, la capacité de conception, et l'investissement dans la réutilisation. Voir le marché de la conception électronique de cette manière, nous aide à identifier où une équipe de conception donnée est située par rapport à l'évolution de la méthodologie de conception. Cela nous aide aussi à déterminer quelles technologies et méthodologies de conception sont nécessaires pour faciliter la transition à une nouvelle étape. L'expérience montre que les compagnies qui réalisent la transition le plus rapidement ont plus de succès sur le marché. On doit noter, qu'il existe des zones de nuance entre ces familles où certains groupes de conception peuvent être situés. De plus le processus de transition est en série par nature. En effet le passage de TDD au PBD est un processus à plusieurs étapes.

3.3.1 Timing-Driven Design

TDD est la meilleure approche pour la conception des ASIC complexes de tailles modérées. Elle est basée sur l'utilisation des processus DSM (interconnect-dominated Deep SubMicron), sans aucune utilisation significative d'une conception hiérarchique. La méthodologie de conception antérieure à la TDD est le Area-Driven Design (ADD). L'objectif du ADD est la minimisation de la logique. L'équipe de conception du ADD est petite et homogène. Lorsque celle-ci rencontre un problème à atteindre certaines performances ou des contraintes de consommation d'énergie qui nécessitent le passage à la méthodologie TDD les symptômes suivants apparaissent :

- Boucle sur la synthèse et le placement sans convergence du temps et de la surface.
- Le fournisseur d'ASIC met long temps pour contourner chaque boucle
- Pas d'anticipation plutôt dans le processus de conception pour une augmentation de la taille
- Répétition de l'optimisation de la surface, du temps, et de la consommation d'énergie
- La création des vecteurs de test est tardive

Ces symptômes sont souvent causés par :

- Absence ou inefficacité du floor planning au niveau RTL ou porte logique
- Aucun processus pour gérer et inclure de manière incrémentale les changements tardifs de la conception RTL dans la conception physique

- Modélisation inefficace de l'infrastructure de la puce (clock, test, power) durant le floor planning
- Mauvaise gestion de la logique du chemin des données

La complexité des produits combinée à un grand nombre de portes logiques, et un TTM de plus en plus court, demandent que la conception et la vérifications soient accélérées, que les compromis soient résolus à un niveau haut de la conception, et que l'interconnexion soit gérée le long du processus de conception et non pas laissée à la fin. Ce sont toutes les différences par rapport à l'approche de conception ADD.

Une méthodologie de conception plus centrée sur le floor plan supportant le changement incrémental, peut alléger ces problèmes. Les outils de floor-planning et d'analyse de temps peuvent être utilisés pour localiser, pendant la conception, les surfaces sensibles au placement. La méthodologie permet alors d'avoir des résultats de placement bien ancrés dans le processus d'optimisation de la conception.

Le Passage du RTL au silicium représente le plus grand risque pour les conceptions qui sont contrôlées par des contraintes du type temporelles, surface, ou puissance. Typiquement, ceci est géré en commençant par la conception physique bien avant que la vérification RTL soit complète. Le recouvrement de ces processus réduit le TTM et contrôle le risque d'ordonnancement, mais au prix d'un surcoût d'ingénierie.

3.3.2 Block-Based Design

L'augmentation de la complexité de conception, la nouvelle relation entre le système, le RTL et la conception physique, ainsi que l'augmentation de la réutilisation des fonctions au niveau système sont des raisons suffisantes pour aller au-delà de la méthodologie TDD. Les symptômes déterminant si le passage de la méthodologie TDD vers la BDD est nécessaire sont :

- l'équipe de conception devient plus spécifique à l'application, et les sous-systèmes, tels que le traitement embarqué, la compression de données, et la correction d'erreurs, est exigée
- Plusieurs équipes de conception sont formées pour travailler sur des parties spécifiques du système
- Les ingénieurs d'ASIC éprouvent des difficultés pour développer des testbenches réalistes et complets
- les erreurs de synchronisation d'interface entre les sous-ensembles augmentent considérablement.
- L'équipe de conception recherche les VCs (Virtual Cores) en dehors de leur groupe pour accélérer le développement de produit.

Idéalement, en BBD la modélisation comportementale est réalisée au niveau système, où on effectue les compensations HW/SW et la Co-vérification fonctionnelle du HW/SW en utilisant la simulation SW et/ou l'émulation du HW. Les nouveaux composants conçus sont alors partagés et mappés sur les blocs fonctionnels RTL spécifiés, et qui doivent respecter les contraintes de temps, de puissance, et de surface. Ceci est contraire à l'approche TDD, où le RTL est capturé le long des restrictions de la frontière de synthèse. La BBD a besoin d'un floor planner efficace au niveau bloc qui peut estimer rapidement la taille des blocs RTL.

BBD utilise les technologies suivantes :

- Outils d'analyse algorithmiques de système de haut niveau et d'application spécifique (Application-specific, high-level system algorithmic analysis tools) : ces outils fournissent la productivité dans la modélisation des algorithmes et de l'environnement opérationnel du système. Ils peuvent être liés aux outils de vérification des langages de description à travers les technologies et les standards de co-simulation tels que le Open Model Interface (OMI) et les capacités de conception basées HDL telles que la synthèse RTL via la génération de HDL et la synthèse comportementale.
- floor planning de Blocs : Il fournit les contraintes spécifiques de l'interconnexion top-level de la puce. Il supporte aussi les modèles d'infrastructure d'horloge, le test, et l'architecture bus, qui est la base d'une véritable abstraction temporelle basée bloc hiérarchique.
- Synthèse intégrée et conception physique (Integrated synthesis and physical design) : Cette technologie permet au concepteur de gérer l'influence croissante des effets de la conception physique durant le processus de synthèse en éliminant le besoin d'itérer entre les différents outils de synthèse, de placement et de routage pour atteindre la convergence de la conception. En utilisant les combinaisons intégrées, le processus de synthèse respecte mieux les contraintes du top-level d'une manière plus prédictible

3.3.3 Platform-Based Design

PBD est la nouvelle étape de l'évolution de la technologie. PBD dépasse les possibilités cumulatives des technologies TDD et BBD, avec plus d'extension de la réutilisation et de la hiérarchisation de la conception. PBD peut diminuer le TTM global pour les premiers produits, et augmenter les occasions et la vitesse de livrer des produits dérivés.

Les symptômes déterminant si une méthodologie de PBD est plus appropriée incluent :

- un nombre significatif de conceptions fonctionnelles se répètent dans/et entre les groupes, mais il y a peu de réutilisation entre les projets. Ce qui se réutilise est au niveau RTL
- les bugs de la conception fonctionnelle cause de multiple itération et/ou des reboucles (re-spins)
- L'analyse des projets montre que les compromis architecturaux (HW/SW, Virtual Core (VC) selection) ont été sous optimaux. Les changements dans les produits dérivés sont abandonnés à cause du risque d'introduire des erreurs.
- Les ICs passent beaucoup de temps sur les équipements de test durant la production, d'ou l'augmentant du coût global.
- Les VCs existant sont constamment remodelés.

Comme la BBD, la PBD est une méthodologie de conception hiérarchique qui commence au niveau système. Là où la PBD diffère de la BBD est qu'elle atteint le summum de sa productivité à travers une extension planifiée de la réutilisation de conception. La productivité est améliorée en utilisant des blocs prédictibles et près vérifiés qui ont des interfaces standardisées. Plus la réutilisation est bien planifiée, moins sont les changements apportés aux blocs fonctionnels. La méthodologie PBD sépare la conception en deux domaines d'intérêt : écriture de blocs, intégration de la puce

La méthodologie PBD utilise les technologies suivantes :

- Les outils de conception algorithmique et architecturale au niveau système et au niveau haut et les technologies de codesign HW/SW (High-level, system-level algorithmic and

architectural design tools and HW/SW co-design technologies) : ces outils, fournissent un environnement pour choisir le composant, repartir le HW et le SW, définissent les interfaces et les contraintes, et effectuent la vérification fonctionnelle à l'aide des modèles de VCs.

- Outil de layout physique concentré sur la planification du bus et l'intégration de bloc (Physical layout tools focused on bus planning and bloc integration) : ces outils, utilisés tôt dans le processus de la conception, sont essentiels pour la PBD. Les effets de la conception physique influencent la topologie et l'architecture du système. De tels outils permettent la prédiction du placement routage hiérarchique nécessaire à la PBD.

- Outils d'écriture de vérification fonctionnelle de VC (VC-authoring functional verification tools) : la vérification dans la PBD se fait au niveau des interfaces : de bloc a bloc, de bloc au bus, du HW au SW, du numérique à l'analogique, et de la puce à son environnement. De ce fait les outils pour écrire les VCs doivent évoluer pour fournir une vérification complète de la fonction du bloc et de séparer les interfaces VC de la fonction du noyau. les outils de simulation conformes à l'OMI permettent la co-simulation à différents niveaux d'abstraction, en partant du niveau algorithme/architecture jusqu'au niveau porte.

Tableau Ch III - 4 : Résumé des méthodologies

Caractéristiques de la conception	TDD	BBD	PBD
complexité	5000 to 250K portes	150k to 1.5M portes	300K portes et plus
Niveau	RTL	Comportemental/RTL	Architecture et évaluation de VC
Equipe de conception	petite, concentrée	Multidisciplinaire	Multigroupe, multidisciplinaire
Conception primaire	Logique dédiée	Blocs et interfaces dédiées	Interfaçage au système et au bus
réutilisation	aucune	Occasionnelle soft, firm, et hard	firm et hard planifiés
Optimisation primaire	Synthèse, architecture au niveau porte	Floor planning, architecture bloc	Architecture du système VCs compatible silicium
Granularité primaire	portes et mémoire	clusters fonctionnels, cores	VCs
Architecture du bus	aucune/dédiée	dédiée	Standardisée/spécifique application multiple
Architecture du test	aucune/scan	Scan/JTAG/BIST/dédiée	Hiérarchicque, scan/JTAG/BIST/custom parallèle
Signal Mixte	aucun	A/D, PLLs	Fonctions, interfaces
Spécification des Contraintes/but	Contraintes logiques	Contraintes Bloc-budget	Contraintes interface
Niveau de vérification	RTL/porte	Bus fonctionnel jusqu'au niveau cycle/RTL/porte	Mixte (ISS au RTL avec HW et SW)
co-vérification HW/SW	aucune	Fonctionnalité HW/SW et interfaces	Seulement interfaces HW/SW
Objet du Partitionnement	limitations de la synthèse (hiérarchique)	Fonctions (hiérarchique)	Fonction/communications (hiérarchique)
Placement	Flat	hiérarchique	hiérarchique
Routage	Flat	Flat	hiérarchique
Analyse du Timing	Flat	Flat avec hiérarchie limitée	hiérarchique
Calcul du Delay	Flat	Flat	hiérarchique
Vérification physique	Flat	Flat avec hiérarchie limitée	hiérarchique

4. Flot de conception SOPC (System On Programmable Chip)

4.1 Les aspects généraux

Le SOPC vient de SOC. Cet acronyme a été inventé par Synopsys. Une bonne définition d'un SOPC est : « un ensemble de bloc fonctionnels construit dans un composant programmable avec au moins un élément de traitement »

De nos jours les composants FPGA haute densité, et les IP permettent l'intégration de systèmes complexes sur la même puce. De nouvelles stratégies et méthodologies de conception doivent être développées pour répondre aux besoins et aux nouvelles contraintes. Ces contraintes peuvent être intrinsèques, liées à l'architecture et à la composition du composant et extrinsèques liées à la technologies utilisée et à l'environnement de ce type de composant. Pour s'adapter à ces nouveaux besoins de conception et trouver une meilleure méthodologie plusieurs travaux on été menés.

Dans **[20]** on trouve une approche pour le mapping du chemin des données et du contrôle dans les FPGA à look-up table. La méthode proposée commence par le niveau RTL et suit les librairies de modules paramétrables. L'environnement du mapping inclus un algorithme implicite de minimisation des états de FSM (Finate State Machine).

Le papier **[21]** ajoute que les méthodologies de conception pour les SOPC nécessitent l'utilisation de celles consacrées aux SOCs vu que les FPGA présentent des IP Hard tels que les multiplicateurs, le cœur du processeur,…**[17]**, **[18]**, qui peuvent poser des problèmes d'intégration. Le travail en progression présenté dans **[22]** propose entre autre une nouvelle vision des systèmes SOPC multiprocesseur qui prend en considération la configuration du système d'exploitation temps réel. Il introduit le problème de la co-configuration HW/SW. Lorsque on parle de réutilisation le problème des interfaces apparaît comme le handicap principal pour le développement des SOPC.

Les utilisations de bus possédant des interfaces standard facilitent l'intégration, éliminent la logique inutile et accélèrent la production des IP. Mais une fois l'IP est conçue pour cibler un bus standard particulier, l'adapté pour un autre bus nécessite du temps et de l'effort. Avec l'apparition de nouveau bus standard et différentes méthodes d'interconnexions ce problème s'accentue. Les wrappers permettent de faciliter le problème d'interconnexions mais leur influence sur la performance décourage, souvent, leur utilisateur.

Dans **[23]** Une nouvelle méthodologie est présentée pour contourner le problème des interfaces. Elle permet d'automatiser la connexion des IP à une large variété d'interfaces avec un faible dépassement à travers l'utilisation d'une couche logique d'adaptateur d'interface spéciale. Dans le même contexte on retrouve l'approche introduite dans **[24]** qui décrit l'implémentation d'une infrastructure de communication qui fournit un nombre de on-chip-IP-Socket. En utilisant l'aspect de reconfiguration dynamique partielle, différentes IP peuvent être plugguées dans ces Socket durant l'exécution. Ceci débouche sur un système reconfigurable qui s'adapte aux différents besoins.

Le travail présenté dans **[25]** passe à un niveau d'abstraction plus élevé en utilisant les Réseaux de Petri **[29]** pour l'adaptation des interfaces et facilite leur vérification.

Généralement on remarque que la méthodologie PBD reste dominante. Elle est présentée par différents fabricants de composants FPGA comme la méthodologie de conception idéale pour améliorer la productivité des équipes de conception **[17]**. Ces flots de conception sont une association du flot de conception HW et du flot de conception SW avec une étape de co-simulation ou de déboguage comme le montre la **figure Ch III - 16**. Cette étape est plus ou moins avancée dans le processus de conception. Vu que cette famille de composant est

reconfigurable et qu'elle est essentiellement consacrée au prototypage, cette étape est souvent la dernière phase du processus de conception.

Cependant, comme la conception devient de plus en plus complexe et que les concepteurs travaillent en équipes, un flot de conception hiérarchique basé bloc est souvent plus efficace. Dans cette approche, chaque sous-bloc peut avoir sa propre netlist, donc l'optimisation peut être réalisée de manière individuelle sur les différents sous-blocs. Après avoir optimisé tous les sous-blocs, on peut les intégrer dans une conception finale et faire l'optimisation à un niveau supérieur si on le désire. La synthèse et l'optimisation de chaque sous-bloc séparément peuvent fournir de meilleurs résultats.

Les composants FPGA permettent d'implémenter deux types de SOPC. Le premier type de SOPC utilise uniquement des IP soft de processeur tel que le MicroBlaze pour Xilinx **[18]** et Nios pour Altera **[17]**. Le nombre de processeurs qui peuvent être intégrés sur un même composant dépend des ressources du composant utilisé.

Le deuxième type de composants en plus de la logique reconfigurable, implémente en dur un ou plusieurs cœurs de processeur (PowerPC pour Xilinx, ARM pour Altera). Ceci n'empêche en rien l'utilisation de processeurs Soft (netlist). La famille des Virtex-II-Pro (voir Chapitre VI) de chez Xilinx et celle des Excalibur de chez Altera sont de bons représentants de ce type de composant. Dans **[26]** est présenté un exemple de conception faisant intervenir un composant du type Virtex-II-Pro.

Ces deux types de SOPC utilisent les mêmes flots de conception. Par exemple celui des composant Virtex-II-Pro est présenté par la **figure Ch III -16**. Tout le flot est présenté sous la forme d'un seul environnement dit Xilinx Platform Studio (voir le Chapitre VI)

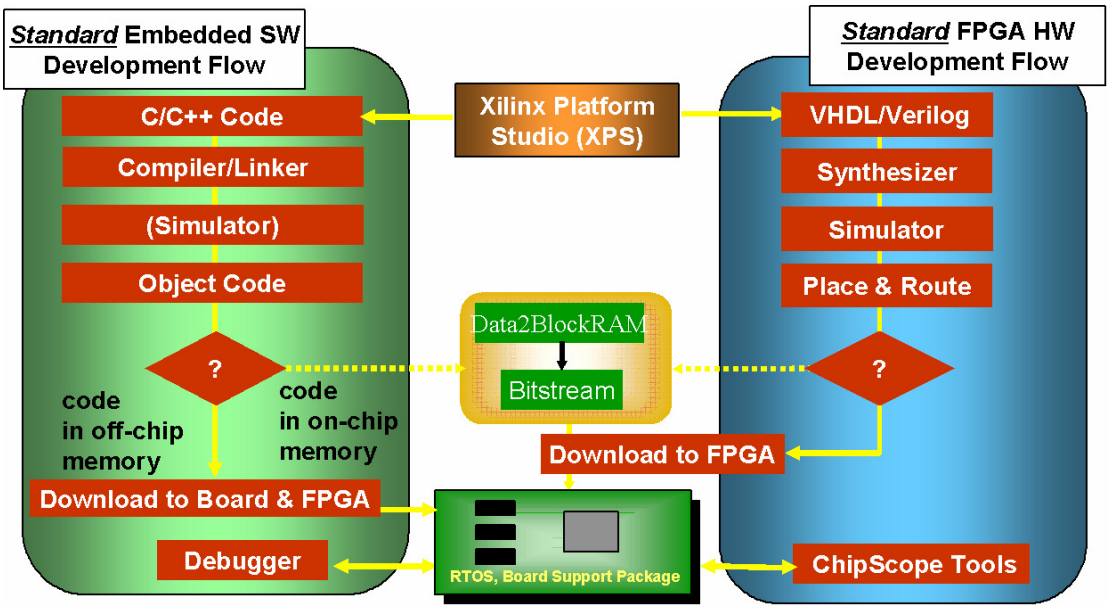


Figure Ch III - 16 : Flot de conception de Xilinx

L'avantage des composants contenant un noyau dur de processeur tel que le virtex-II-Pro, est que le cœur du processeur peut fonctionner à une fréquence beaucoup plus élevée que celle du reste de la logique. Alors que les processeurs Soft subissent la fréquence maximale de la conception globale. Bien que les composants qui intègrent un noyau de processeur dur imposent une contrainte supplémentaire au concepteur, ceci ne veut pas nécessairement dire que ce type de composant va réduire ou limiter les performances de la conception.

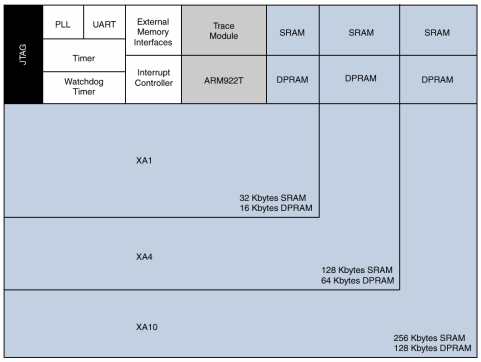


Figure Ch III - 17 : Architecture d'un Excalibur d'Altera

4.2 La reconfiguration dynamique : un nouveau défi

Les composants et architectures ayant de la Logique Dynamiquement reconfigurable (DRL) tels que les composants FPGA imposent de nouveaux défis à la communauté d'automatisation de la conception. Le défi principal présenté par les composants DRL est la latence de reconfiguration, qui doit être réduite au minimum afin de maximiser le temps d'exécution de l'application. Afin de réaliser de la reconfiguration en temps d'exécution, la spécification du système (typiquement, un graphique de tâche) doit être divisée en segments temporels exclusifs (appelés les contextes de reconfiguration). Ce processus est habituellement connu comme le partitionnement temporel et c'est une manière d'introduire le problème de la latence de reconfiguration. Ce problème est d'autant plus important que la dynamicité de l'application est élevée.

Une approche différente est de trouver un ordre d'exécution pour un ensemble de tâches qui respecte les objectifs de la conception (c-à-d., réduire au minimum le temps d'exécution de l'application). Ceci est connu en tant que *DRL multicontext scheduling*. Il y a un grand nombre d'approches de codesign HW/SW des systèmes embarqués qui emploient différentes techniques pour le partitionnement et l'ordonnancement. Cependant, les composants et les architectures DRL changent plusieurs hypothèses de base des processus de codesign HW/SW. La flexibilité de la reconfiguration dynamique (configurations multiples, reconfiguration partielle, etc...) exige de nouvelles méthodologies et le développement de nouveaux algorithmes, car les approches conventionnelles de codesign ne prennent pas en considération les nouveaux aspects et caractéristiques des composants et architectures DRL.

Plusieurs travaux ont été menés pour proposer des méthodologies de conception plus appropriées aux composants et aux architectures DRL. Traditionnellement, les problèmes de codesign du HW/SW, et ceux des composants et des architectures DRL, ont été étudiés indépendamment.

Les techniques de partitionnement HW/SW et d'ordonnancement peuvent être différenciées de plusieurs manières. Par exemple, le partitionnement peut être classifié comme à grain fin (fine-grained) s'il partage les spécifications du système au niveau bloc de base, ou comme à grain grossier (coarse-grained) si la spécification du système est partagée au niveau processus ou tâche. Par contre, l'ordonnancement du HW/SW peut être classifié comme statique ou dynamique. Une politique d'ordonnancement est dite statique si les tâches sont exécutées dans un ordre fixe déterminé au préalable, et dynamique quand l'ordre d'exécution est décidé en ligne (on-line). La séquence de tâches HW/SW peut changer dynamiquement dans les systèmes embarqués complexes, puisque de tels systèmes doivent souvent fonctionner dans différentes conditions.

Bien qu'il y ait eu beaucoup de travaux dans d'ordonnancement HW/SW statique, le problème d'ordonnancement dynamique dans le codesign de HW/SW a été traité dans seulement quelques efforts de recherches. Une stratégie pour l'exécution d'ordonnancement dynamique temps réel du HW/SW est présentée dans **[16]**. Dans **[9]** un aperçu de plusieurs approches pour l'ordonnancement de logiciel à dominante contrôle et à dominante flux de données ont été présentées. D'autre part, plusieurs approches peuvent être trouvées dans la littérature traitant la minimisation de latence de reconfiguration.

Des techniques de prefetching de configuration ont été employées pour la minimisation de la latence. Elles sont basées sur l'idée de charger le prochain contexte de reconfiguration avant qu'il soit demandé, d'où, le recouvrement de la reconfiguration du composant et de l'exécution de l'application **[12]**, **[15]**. Toutes ces approches traitent le problème de la minimisation de latence de reconfiguration, mais elles n'étudient pas le partitionnement et l'ordonnancement HW/SW.

Des plus récents efforts de recherches ont mené l'étude de ce problème encore non résolu. Dans **[10]**, un algorithme pour le partitionnement et l'ordonnancement HW/SW présente le partitionnement temporel, et l'ordonnancement du contexte. Dans **[14]** est présenté un algorithme de partitionnement à grain fin de HW/ SW (au niveau boucle). Les deux approches précédentes sont semblables à celle de **[11]** qui tient compte du temps de reconfiguration en effectuant le partitionnement, mais elles ne considèrent pas les effets du prefetching de la configuration pour la minimisation de latence. Dans **[13]**, cette approche a été introduite, et une approche de cosynthèse HW/SW pour les composants partiellement reconfigurables a été présentée. Un algorithme de cosynthèse HW/SW à amélioration itérative qui permet l'optimisation de l'architecture des systèmes hétérogènes pour les FPGA dynamiquement reconfigurables a été présenté dans **[27]**. L'algorithme permet de maximiser la vitesse en prenant en compte les contraintes de coût.

La reconfiguration dynamique est utilisée dans plusieurs travaux tels que **[36]**, et **[37]** pour obtenir une meilleure flexibilité, et pour réduire la consommation d'énergie ainsi que l'utilisation des ressources. Ceci concerne particulièrement les systèmes mobiles **[38]**, **[39]**. Le travail de **[40]** nous concerne de près car il présente une implémentation dynamiquement reconfigurable du MQ-décodeur de JPEG2000.

5. Conclusion

Dans ce chapitre on a présenté les différentes approches et méthodologies de conception des SOC ainsi que leur évolution. Cette évolution est principalement liée à l'évolution de la technologie et la capacité d'intégration, ainsi qu'à la complexité des applications. Dans le cadre de cette présentation on a fait un aperçu de quelques exemples de méthodologies de codesign ainsi que certains algorithmes qui ont été développés par différents groupes de recherche pour rendre automatique la tâche de partitionnement HW/SW. Dans la deuxième partie de ce chapitre on a présenté les approches et les méthodologies de conception des SOPC. La relation entre la conception SOC et SOPC a été aussi évoquée. Avec les SOPC de nouveaux défis de conception sont apparus. Ceci ne veut pas dire qu'il a eu une rupture entre les deux types de systèmes, mais le recours aux méthodologies de conception propres aux SOC reste courant vue la nature même des technologies utilisées dans les SOPC.

Dans la suite on continue une analyse des implémentations logicielles de JPEG2000 sur différentes plateformes. Le but principal de cette partie est de déterminer le niveau de complexité de JPEG2000 par rapport aux différentes architectures étudiées, et d'en tirer une vision des possibles optimisations.

6. Références

[1] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, Lee Todd, "Surviving the SOC Revolution : A Guide to Platform-Based Design", Kluwer Academic Publishers, ISBN 0-7923-8679-5, 1999

[2] J.P. Calvez, D.Isidoro,"A CoDesign Experience with the MCSE Methodology**"**, Proceedings of the Third International Workshop on Hardware/Software Codesign, 1994, Pages:140 – 147, 22-24 Sept. 1994

[3] J.P. Calvez, O.Pasquier, D.Isidoro, D. Jeuland,"CoDesign with the MCSE Methodology**",** EUROMICRO 94. System Architecture and Integration. Proceedings of the 20th EUROMICRO Conference, Pages:19 - 26, 5-8 Sept. 1994.

[4] Karam S.Chatah, Ranga Vemuri, "MAGELLAN : Multiway Hardware Software Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs", 9th International Workshop on Hardware/Software Co-Design, April 25 - 27, 2001

[5] Asawaree Kalavade, Edward A.Lee, "The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling, and Implementation-bin Selection", The Morgan Kaufmann Systems On Silicon Series, Readings in hardware/software co-design, Section: System-level partitioning, synthesis, and interfacing Pages: 293 - 312, 2001

[6] Asawaree Kalavade, P.A.Subrahmanyam, "Hardware/Software Partitioning for Multi-function Systems", International Conference on Computer Aided Design, Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design, San Jose, California, United States,Pages: 516 - 521, 1997

[7] Niraj K.Jha, Bharat P.Dave, Ganesh Lakshminarayana, "COSYN: Hardware-Software Co-synthesis of Embedded Systems", Design Automation Conference, 34th Conference on (DAC'97), Anaheim, CA. , June 09 - 13, 1997

[8] Robert P.Dick, Niraj K.Jha, "MOGAC : A Multiobjective Genetic Algorithm for Hardware-Software Co-synthesis of Distributed Embedded Systems", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 17 ,Pages:920 - 935, 10 , Oct. 1998,.

[9] F. Balarin, L. Lavagno, P. Murthy, and A. S. Vincentelli, "Scheduling for Embedded Real-Time Systems," IEEE Design and Test, Jan–Mar. 1998.

[10] K. Chatta and R. Vemuri, "Hardware–Software Codesign for Dynamically Reconfigurable Architectures," in Proc. of FPL'99, Glasgow, Scotland, Sept. 1999.

[11] J. Fleischman *et al*., "A hardware/software Prototyping Environment for Dynamically Reconfigurable Embedded Systems," in Proc. CODES'98, Seattle, WA, Mar. 1998.

[12] S. Hauck, "Configuration Prefetch for Single Context Reconfigurable Co-processors,"in Proc. ACM/SIGDA Int. Symp. FPGA, Monterey, CA, pp. 65–74, Feb. 1998.

[13] B. Jeong *et al*., "Hardware/Software Cosynthesis for Run-Time Incremen-tally Reconfigurable FPGA's," in Proc. of Asia South-Pacific Design Au-tomation Conf. (ASP-DAC'2000), pp. 43–48, Japan, Yokohama, Jan. 2000

[14] Y. Li *et al*., "Hardware/Software Co-design of Embedded Reconfigurable Architectures," in Proc. 37th Design Automation Conf, DAC'2000.

[15] R. Maestre, F. J. Kurdahi, N. Bagerzadeh, H. Singh, R. Hermida, and M.Fernandez, "Kernel Scheduling in Reconfigurable Computing," in Proc. DATE, Munich, Germany, Mar. 1999.

[16] V. Mooney and G. De Micheli, "Real Time Analysis and Priority Scheduler Generation for Hardware-Software Systems with a Synthe-Sized Run-Time System," in Proc. Int. Conf. Computer-Aided Design (ICCAD'97), pp. 605–612, San Jose, CA, Nov. 1997.

[17] www.altera.com

[18] www.xilinx.com

[19] www.ibm.com

[20] J.Abke, E.Barke, "A Direct Mapping System for Datapath Module and FSM Implementation into LUT-Based FPGAs", Design, Automation and Test in Europe Conference and Exhibition, 2002 , Pages:1085, 4-8 March 2002

[21] R.Venkata, W.Wong, T.Tran, V.Chan, T.Hoang, H.Lui, B.Ton, S.Shumurayev, Chong Lee, Shoujun Wang, Huy Ngo, M.Kabani, V.Maruri, Tin Lai, Tam Nguyen, A.Zaliznyak, Mei Luo, Toan Nguyen, K.Asaduzzaman, S.Maangat, J.Lam, R.Patel, "Architecture and Methodology of a SOPC with 3.25 Gbps CDR Based Serdes and 1 Gbps Dynamic Phase Alignment", Custom Integrated Circuits Conference, 2003. Proceedings of the IEEE 2003, Pages:659 - 662, 21-24 Sept. 2003

[22] H.Takada, S.Honda, R.Nishiyama, H.Yuyama, “Hardware Software Co-Configuration for Multiprocessor SOPC(work-in-progress report)”, IEEE Workshop on Software Technologies for Future Embedded Systems, 2003, Pages:7 - 8, 15-16 May 2003

[23] T.-L. Lee, N. W. Bergmann, “An Interface Methodology for Retargetable FPGA Peripherals”, In Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Las Vegas, Nev., June 2003.

[24] H.Kalte, D.Langen, E.Vonnahme, A.Brinkmann, U.Ruckert, “Dynamically Reconfigurable System on Programmable Chip”, 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002, Pages:235 - 242, 9-11 Jan. 2002

[25] J.A.de Oliveira Filho, M.E.de Lima, P.R.Maciel, “Petri Net Based Interface Analysis for Fast *IP-Core* Integration”, First ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2003. MEMOCODE '03, Pages:34 - 42, 24-26 June 2003

[26] S.Sezer, C.Toal, Xing Yu, “A Pipelined SOPC Architecture for Data Link Layer Protocol Processing”, IEEE International [Systems-on-Chip] SOC Conference 2003,Pages:277 - 278, 17-20 Sept. 2003

[27] R.Czarnecki, S.Deniziak, K.Sapiecha, “An interactive improvement Co-Synthesis Algorithm for Optimization of SOPC Architecture with Dynamically Reconfigurable FPGAs”, Euromicro Symposium on Digital System Design 2003, Pages:443 - 446, 1-6 Sept. 2003

[28] Amer BAGHDADI, “Exploration et Conception Systématique d’Architectures Multiprocesseurs Monopuces Dédiées a des Applications Spécifiques”, Thèse préparée au laboratoire TIMA Grenoble France, dirigée par Ahmed Amine JERRAYA, soutenue le 14 Mai 2002.

[29] Tadao Murata “Petri nets: Properties, Analysis and Applications”, Proceedings of the IEEE, Volume: 77 , Issue: 4, Pages:541 - 580, April 1989.

[30] Spartan-II 2.5V FPGA Family: Complete Data Sheet, www.xilinx.com

[31] Virtex™ 2.5 V Field Programmable Gate Arrays, www.xilinx.com

[32] Virtex-II Platform FPGAs: Complete Data Sheet, www.xilinx.com

[33] Jonathan Rose, Abbas El Gamal, Alberto Sangiovanni-Vincentelli, “Architecture of Field-Programmable Gate Arrays”, Proceedings of the IEEE, Volume: 81, Issue: 7, Pages: 1013-1029, July 1993.

[34] Ma Xiaojun, Tong Jiarong, “Design and implementation of a new FPGA architecture”; Proceedings. 5th International Conference on ASIC, Volume 2, 21-24 Oct. 2003 Page(s): 816 - 819 Vol.2, 2003

[35] S. Brown, “FPGA architectural research: a survey”; Design & Test of Computers, IEEE Volume 13, Issue 4, Page(s):9 - 15, Winter 1996.

[36] Bourennane, E.B.; Bouchoux, S.; Miteran, J.; Paindavoine, M.; Bouillant, S., "Cost comparison of image rotation implementations on static and dynamic reconfigurable FPGAs", Acoustics, Speech, and Signal Processing, 2002. Proceedings. (ICASSP '02). IEEE International Conference on Volume 3, Page(s): III-3176 - III-3179, May 2002.

[37] Jian Liang; Tessier, R.; Goeckel, D., "A dynamically-reconfigurable, power-efficient turbo decoder", Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on Page(s):91 - 100, 20-23 April 2004.

[38] Gerard J. M. Smit, Paul J. M. Havinga, Lodewijk T. Smit, Paul M. Heysters, and Michel A. J. Rosien. “Dynamic reconfiguration in mobile systems”, In Field-Programmable Logic and Applications, pages 171-181, September 2002

[39] Paul J. M. Havinga, Lodewijk T. Smit, Gerard J. M. Smit, Martinus Bos, and Paul M. Heysters, “Energy management for dynamically reconfigurable heterogenous mobile systems”, In Proceedings of the Heterogeneous Computing Workshop (in conjunction with IPDPS 2001), April 2001

[40] Sophie Bouchoux, El-Bay Bourennane, Johel Miteran, Michel Paindavoine, "Implementation of JPEG2000 Arithmetic Decoder on a Dynamically Reconfigurable ATMEL FPGA", IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design (ISVLSI'04) Lafayette, Louisiana, February 19 - 20, 2004

Chapitre IV Evaluation de Performances

1. Introduction et Motivations

Les exigences relatives à JPEG2000 peuvent être très variées suivant les applications considérées. Cela rend nécessaire de considérer un large éventail d'utilisations dans des environnements très distincts dans leurs contraintes en temps, en consommation d'énergie, en espace utilisé et bien entendu en coût. Dans les applications exigeant un temps de réponse contraignant il est important d'explorer les possibilités des implémentations logicielles avant de s'investir dans la conception de circuits dédiés. Nous effectuons dans ce chapitre : (1) une analyse de deux logiciels implémentant le codage JPEG2000 JasPer v.1.500.4 **[1]** et Kakadu v2.2.3 **[2]** sur différentes plateformes Intel et ce en utilisant un ensemble d'images de caractéristiques différentes et en explorant l'effet des optimisations du compilateur puis en faisant intervenir les librairies spécialisées de chez Intel **[43]**. (2) Nous passons ensuite à d'autre types de plateformes qui sont les DSP, en étudiant les performances de JPEG2000 sur une plateforme TriMedia de chez Philips **[44]**. (3) Enfin nous précisons les conclusions retenues suite à ces évaluations et les orientations de nos travaux sur la question.

2. Evaluation de JPEG2000 sur les plateformes Intel

2.1 Microarchitecture des Processeurs Intel Pentium II, III et 4

Notre expérience d'évaluation de performance s'est voulue aussi large que possible mais l'existence d'outils appropriés ainsi que l'accès aisé à des plateformes cibles nous a restreint à des plateformes à base de processeurs Intel. Les processeurs choisis pour cette expérience sont des processeurs Intel: le Pentium II, III et 4. Nous décrivons très brièvement chaque processeur.

2.1.1 Intel Pentium II

Le Pentium II **[8]** est le premier processeur de la lignée Intel sur lequel il est possible d'appliquer les outils d'analyse de performances. La microarchitecture du PII est la P6 qui est une architecture d'un processeur superscalaire avec exécution spéculative. Le PII intègre les instructions spécialisées MMX (multimedia extension).

2.1.2 Intel Pentium III

Le Pentium III **[7]** possède la microarchitecture P6 d'un processeur superscalaire avec exécution out-of-order.

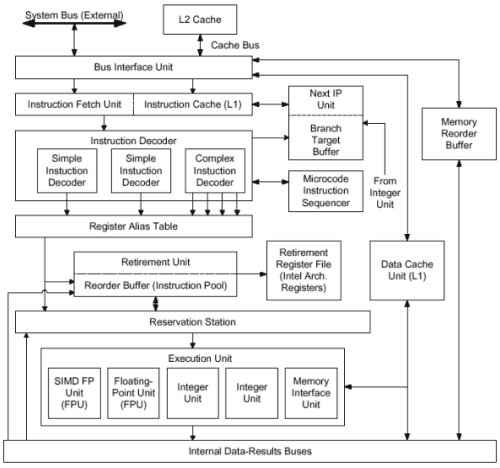


Figure Ch IV - 1 : Microarchitecture du Pentium III

Le Pentium III implémente la première partie des instructions SSE (Streaming SIMD Extensions).

2.1.3 Intel Pentium 4

Le Pentium 4 est basé sur la microarchitecture NetBurst **[7]** optimisée pour les applications multimédia avec l'ajout de 144 nouvelles instructions 128bits SIMD appellées SSE2 (Streaming SIMD Extension 2) **[8]**.

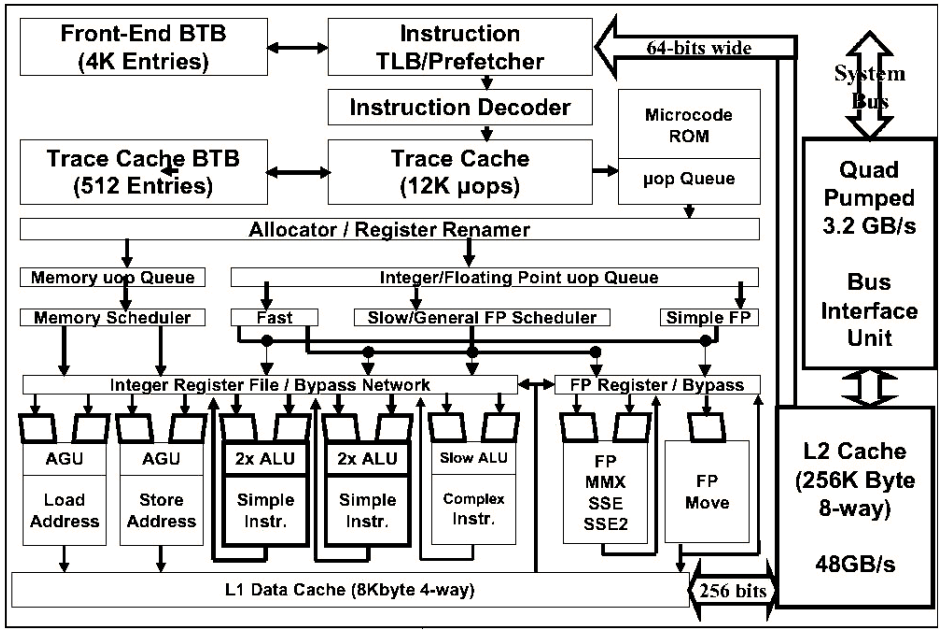


Figure Ch IV - 2 : Microarchitecture du P4

Contrairement aux autres architectures Intel, le P4 inclut un trace cache qui remplace le traditionnel cache instruction. Le trace cache conserve les micro opérations (uops) c'est-à-dire les instructions (12K uops) déjà décodées ce qui est un gain de temps important compte tenu du décodage complexe des instructions IA-32. De plus, le trace cache assemble et construit des traces de uops. Cela permet donc d'établir des chemins d'exécution plus ordonnés que ne le sont des lignes traditionnelles de cache instruction. Enfin le concept de trace permet d'inclure une instruction de branchement et l'instruction cible dans le même flot évitant ainsi les ruptures du pipeline.

2.2 Compilateur Microsoft Visual C++ v.6.0.et Optimisations

Les codeurs logiciels Jasper et Kakadu sélectionnés pour cette étude et écrits respectivement en ANSI C et C++ ont été compilés avec le compilateur Microsoft Visual C++ 6.0. Lors de l'implémentation logicielle sur des microarchitectures complexes il est absolument indispensable d'explorer les différentes optimisations de code fournies par un compilateur **[13]**. Celles-ci sont nombreuses et suivant le compilateur, plus ou moins élaborées et contrôlables par l'utilisateur. Le compilateur Microsoft Visual C++ v.6.0 permet de nombreuses optimisations de code. Les options de compilation retenues pour l'optimisation sont décrites dans le tableau suivant :

Tableau Ch IV - 1 : Options de Compilation

Options de Compilation	équivalence	signification
O1	/Og/Os/Oy/Ob1/Gs/Gf/Gy	Crée le code le plus petit
O2	Og/Oi/Ot/Oy/Ob1/Gs/Gf/Gy	Cree le code le plus rapide
Gf		Active le string pooling
Gs		Contrôle les probes du stack
Gy		Active le link au niveau fonction
Ob1		développe les fonctions marqué comme **inline** ou **__inline** ou, une fonction membre C++, défini dans la déclaration d'une classe (par défaut avec /O1, /O2, et /Ox)
Og		Utilise l'optimisation globale
Oi		Génère des fonctions intrinsèques
Os		Favorise le code le plus petit
Ot		Favorise le code le plus rapide

Les différentes options de compilation ont été testées mais les statistiques produites par les différents exécutables n'ont pas permis de pouvoir considérer que ces options avaient un impact significatif sur l'exécution et ce en considérant de manière orthogonale les deux versions du codeur et les différentes images sélectionnées pour l'ensemble des plateformes. Par conséquent tous les résultats présentés dans la suite de ce document le sont sur la base des optimisations par défaut. Aucun autre compilateur dans l'environnement Windows n'a été évalué.

2.3 Analyse de Performances avec Vtune

Vtune **[9]** est un environnement logiciel de mesure et d'analyse de performance d'applications s'exécutant dans un environnement Windows sur processeurs Intel. Il existe essentiellement trois modes de collecte d'information : (1) échantillonnage basé sur le temps, (2) échantillonnage par évènement et (3) utilisation des compteurs de performance (hardware performance counters). Les deux premiers modes extraient des informations de performance soit de manière périodique (temps) soit sur la base d'évènements. Le dernier mode se base sur l'existence de compteurs de performance sur les processeurs pour extraire l'information.

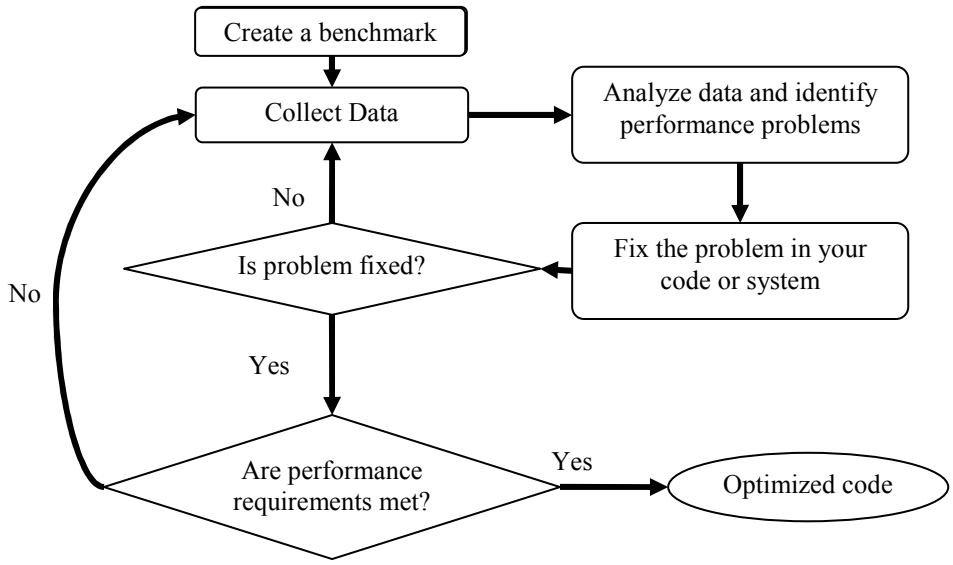


Figure Ch IV - 3 : Flot de Traitement VTune

Le flot de traitement sous VTune est un cycle de collection d'information- optimisation-analyse utilisé en général pour l'optimisation d'applications **[10]**. L'utilisation de VTune n'exige pas de recompilation particulière sauf lors de modifications volontaires. Dans notre cas nous n'avons pas modifié le code mais uniquement utilisé l'outil pour l'extraction d'informations de performances.

2.4 Méthodologies et Plateformes Utilisées

Dans l'objectif de tester de la manière la plus objective possible nous avons constitué un jeu de test de cinq (5) images tests qui sont décrites dans le paragraphe suivant tandis que nous avons varié les plateformes de test pour tenir compte de la variété des environnements possibles d'utilisation de JPEG2000.

2.4.1 Méthodologie d'analyse et Base de données d'images

En l'absence d'une base de données d'images reconnue et standardisée pour les tests du codeur JPEG2000, nous avons sélectionné un ensemble d'images pour nos expériences. Nous avons varié : (1) le format des images (2) leurs tailles (3) leurs niveaux de couleurs. Les images utilisées pour ce test comportent quelques images connues comme Lena, Barbara et Baboon ainsi que des images générées au sein du Laboratoire comme enaho.

Tableau Ch IV - 2 : Caracteristiques des Images Tests

N	Image	Format	Résolution	Taille (kb)	type
1	baboon	BMP	512x512	786.486	Couleur (24 bits)
2	lena	PPM	512x512	786.486	Couleur
3	Babara	PGM	512x512	262.182	Niveau de gris
4	bird	BMP	256x256	66.614	Niveau de gris
5	enaho	BMP	1080x721	2336.096	Couleur (24 bits)

2.4.2 Caractéristiques des Plateformes Utilisées

Nous avons sélectionné trois plateformes pour les expériences. Celles-ci sont décrites ci-dessous.

Tableau Ch IV - 3 : Plateformes de Test

Paramètre	Plateforme 1	Plateforme 2	Plateforme 3
Processeur	P4	P III	P II
Fréquence	2 GHz	733 MHz	400 MHz
Mémoire Centrale	1 GB	256MB	128 MB
OS	MS Windows 2000	MS Windows XP	MS Windows 2000
Compilateur	MS Visual C++ v.6.0.	MS Visual C++ v.6.0.	MS Visual C++ v.6.0
Logiciel d'analyse	Vtune 6.0	Vtune 6.0	Vtune 6.0

Les caractéristiques des plateformes utilisées sont décrites en se basant sur les spécifications utilisées pour SPEC **[28]**.

2.5 Résultats

Notre méthodologie de caractérisation suit les approches en architecture des ordinateurs **[16]** et les travaux de caractérisation effectués régulièrement **[17-22]** sur les benchmarks importants comme SPEC CPU **[15]**. Les paramètres évalués sont : (1) les temps d'exécution et leurs décompositions (2) le comportement de la hiérarchie mémoire (3) le comportement des TLB (4) la prédiction de branchement (5) la distribution dynamique des instructions. Ceci est effectué sur les trois plateformes.

2.5.1 Temps d'exécution Application/Systèmes et Instructions par Cycle

Le temps d'exécution des applications a été décomposé en temps utilisateur, temps OS. Il apparaît donc que les applications passent la majorité du temps dans le temps utilisateur. Cela aussi suggère une première estimation du temps qui pourrait être réduit dans le cadre d'une implémentation dédiée faisant donc moins d'appel au système opératoire.

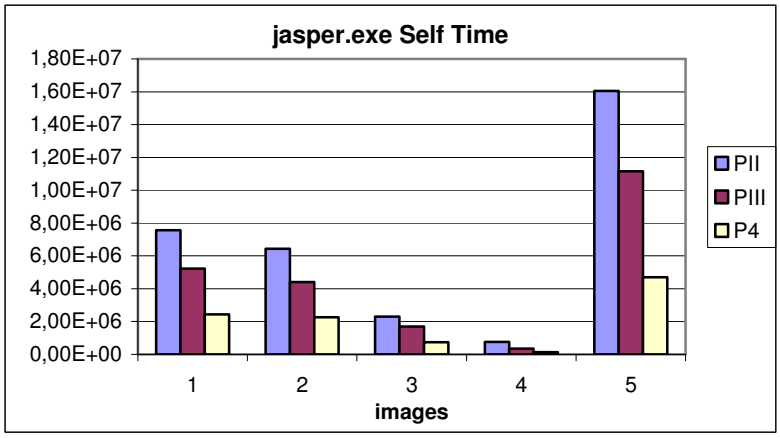


Figure Ch IV - 4 : Comparaison des Temps d'Exécution de Jasper sur PII, PIII, P4

La figure précédente confirme l'intuition que le temps d'exécution décroît en passant du PII au P4. Par contre cette réduction du temps d'exécution est bien entendu très dépendante de la taille de l'image traitée. Ainsi pour l'image ayant la taille la plus importante (enaho.bmp), cette réduction est la plus importante. Ceci n'est plus vrai pour bird.bmp ou la réduction est moins significative. Une conséquence directe est qu'une optimisation du codeur JPEG2000 a d'autant plus d'intérêt que l'on traite des grandes images.

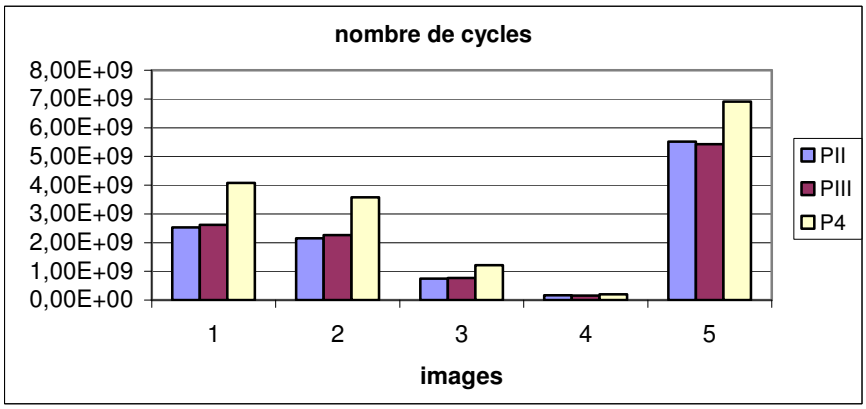


Figure Ch IV - 5 : Nombre de cycle horloge pour l'exécution de Jasper sur PII, PIII, P4

On pouvait penser que le nombre de cycles horloges nécessaires à l'exécution aurait pu aussi décroître en passant du PII au P4 par l'exploitation de davantage de parallélisme instructions mais ceci n'est pas confirmé. La raison tient au fait que les mesures sur le nombre

d'instructions par cycle restent très basses vu que le compilateur comme il a été mentionné précédemment ne parvient pas à générer une optimisation de l'exécution au niveau parallélisme instructions.

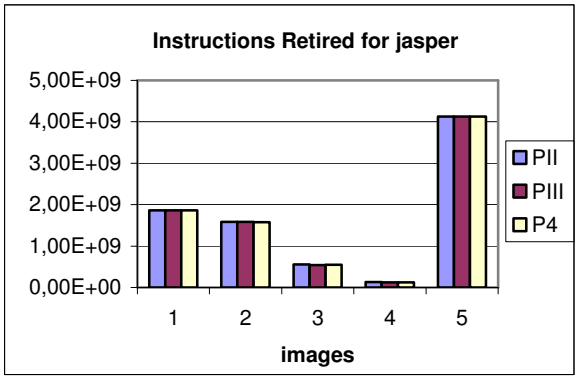


Figure Ch IV - 6 : Nombre d'instructions exécutées pour Jasper sur PII, PIII et P4.

Le nombre d'instructions est aussi globalement le même démontrant la aussi que les variations des jeux d'instructions à travers des différentes plateformes ont très peu joué. Ceci confirme l'aspect conservateur du compilateur.

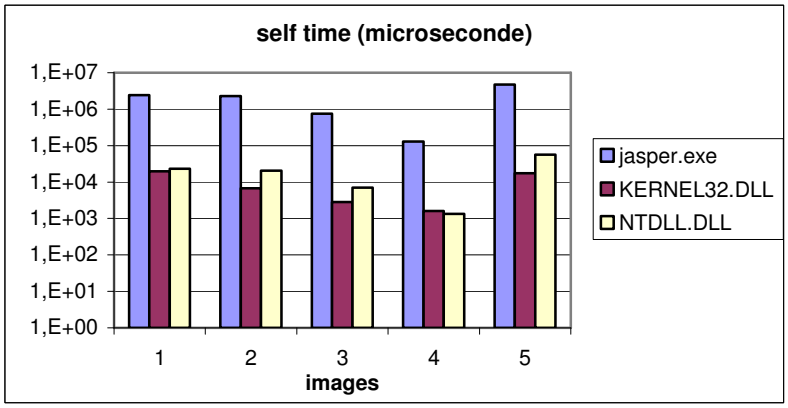


Figure Ch IV - 7 : Temps d'exécution des principaux modules pour jasper (P 4)

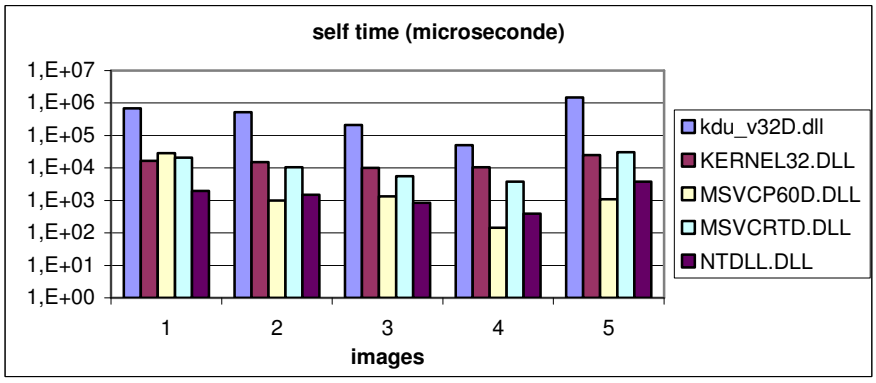


Figure Ch IV - 8 : Temps d'exécution des principaux modules pour Kakadu (P4)

Les deux figures précédentes démontrent qu'autant pour Jasper que Kakadu la majorité du temps d'exécution se fait au bénéfice de l'utilisateur. Le temps d'exécution des modules KERNEL32, NTDLL, MSVCP60D, MSVCRTD, qui font partie de l'OS, est négligeable par rapport à celui de l'application (voir les figures **Ch IV – 9, Ch IV – 10**). Donc les optimisations sur le traitement JPEG2000 seront les plus à même d'apporter une réduction du temps d'exécution.

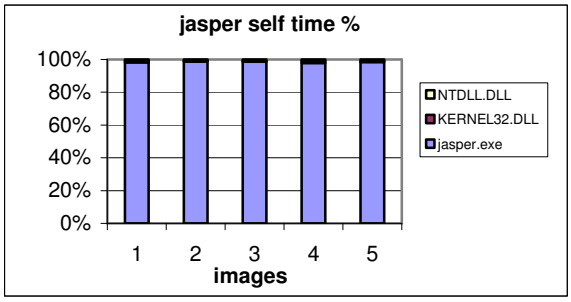


Figure Ch IV - 9 : Pourcentage du temps d'exécution des modules pour Jasper (P4)

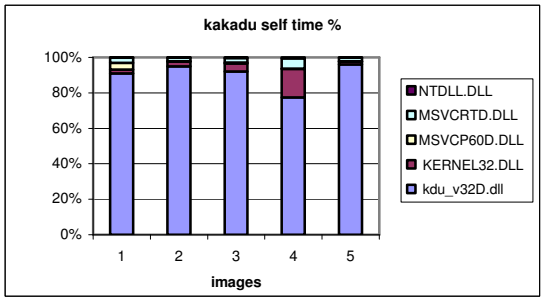


Figure Ch IV - 10 : Pourcentage du temps d'exécution des modules pour Kakadu (P4)

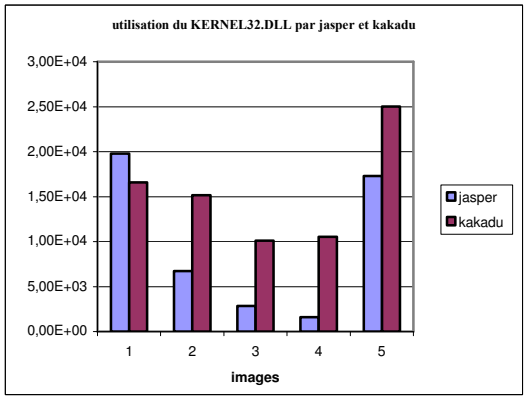


Figure Ch IV - 11 : Comparaison entre Kakadu et Jasper pour l'utilisation du KERNEL32.DLL (P4)

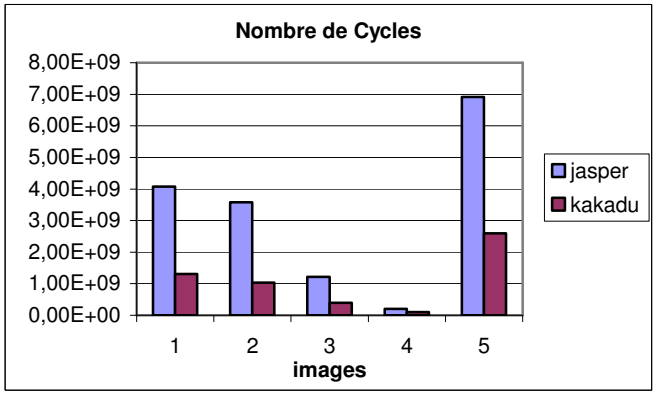


Figure Ch IV - 12 : Nombre de cycles d'horloge pour jasper (le module jasper.exe) et Kakadu (le module kdu_v32D.dll) (P4)

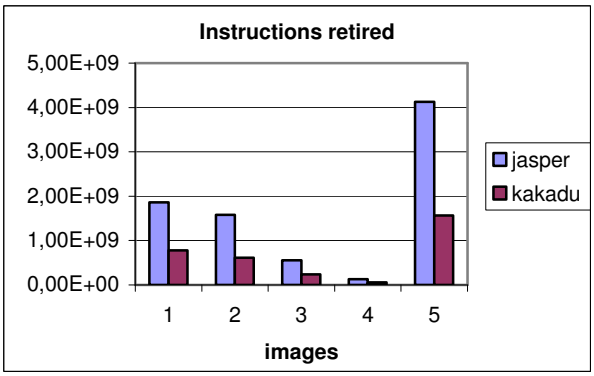


Figure Ch IV - 13 : Nombre d'instructions pour jasper (le module jasper.exe) et Kakadu (le module kdu_v32D.dll) (P4)

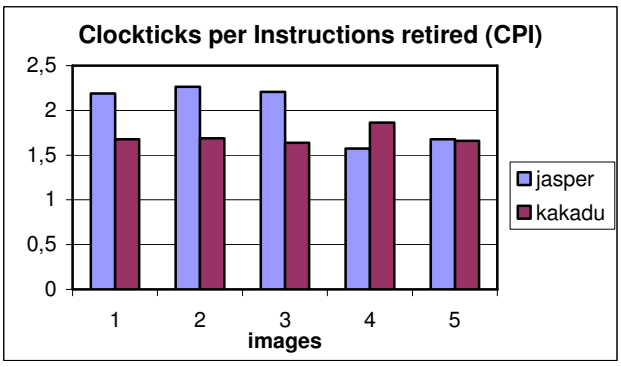


Figure Ch IV - 14 : Cycles d'horloge/instruction pour jasper (le module jasper.exe) et Kakadu (le module kdu_v32D.dll) (P4)

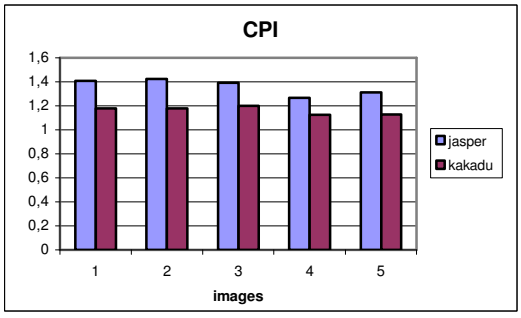


Figure Ch IV - 15 : CPI pour le PIII

Le CPI qui en moyenne est supérieure à 1.5 P4 par rapport à 1 pour le PIII démontre un parallélisme très pauvre au niveau instructions malgré les possibilités matérielles importantes du P4. Les différentes possibilités d'optimisation avec le compilateur MS Visual C++, déjà présentées, n'ont pas eu d'effet sur l'extraction d'un parallélisme au niveau instructions.

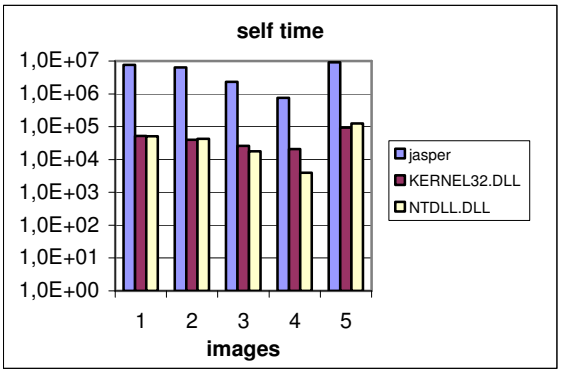


Figure Ch IV - 16 : Self time pour le PII

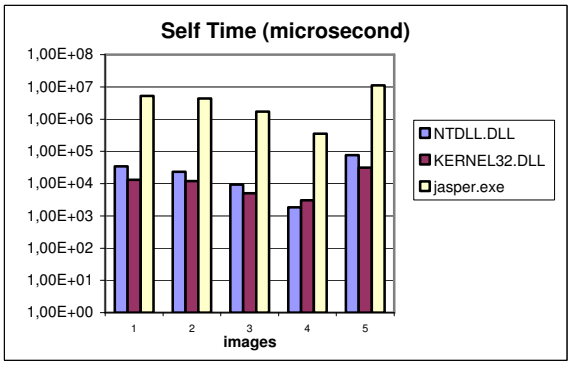


Figure Ch IV - 17 : Selftime pour le PIII Jasper

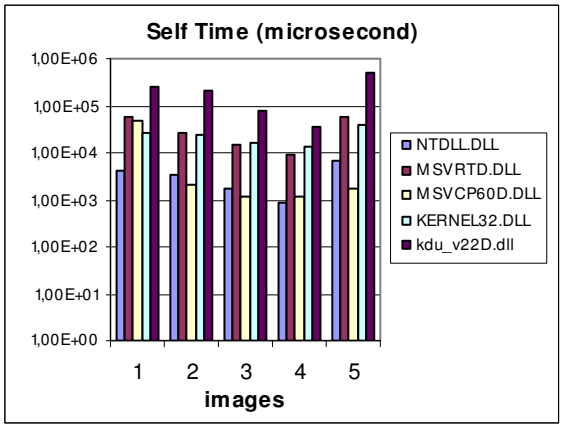


Figure Ch IV - 18 : Selftime pour le PIII Kakadu

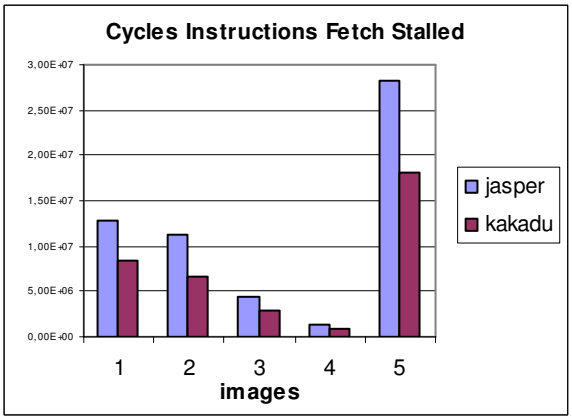


Figure Ch IV - 19 : Nombre de Stalls pour le PIII

2.5.2 Impact des Caches L1 et L2 (Cache Miss Ratios)

L'étude du comportement des caches sur les trois plateformes a été effectuée en analysant, lorsque cela était possible, les défauts de cache instructions et données, et cela sur autant de niveaux de la hiérarchie mémoire que possible (L 1, L2, L3).

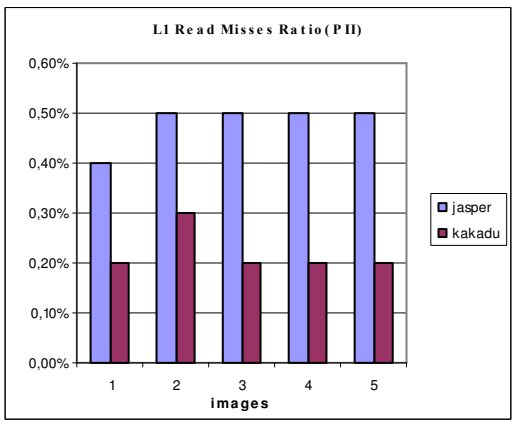


Figure Ch IV - 20 : L1 Read Misses Ratio (PII)

Figure Ch IV - 21 : L1 Read Misses Ratio (PIII)

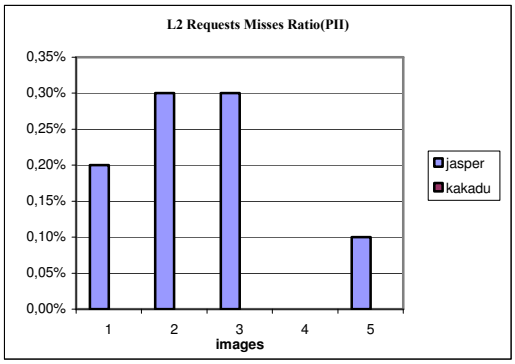


Figure Ch IV - 22 : L2 Requests Misses Ratio (PII)

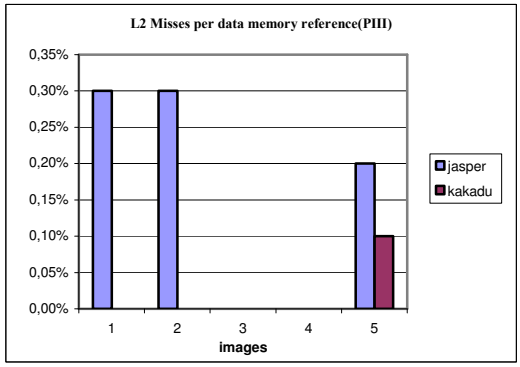


Figure Ch IV - 23 : L2 Misses per Data Memory Reference

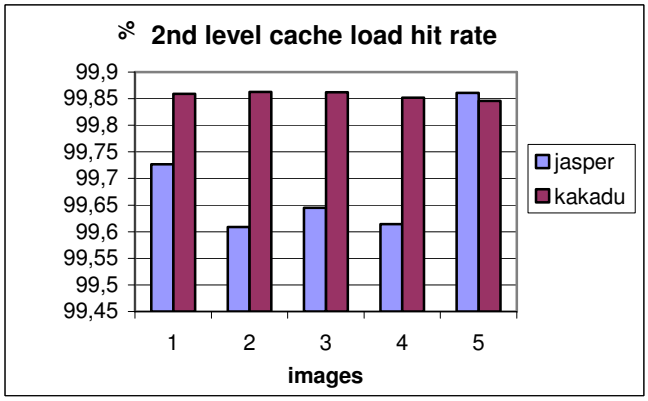


Figure Ch IV - 24 : Taux d'utilisation du cache 2eme niveau

2.5.3 TLB (Translation Lookaside Buffer)

Le TLB instructions est a priori très sensible à des appels fréquents ayant plusieurs fonctions de petites tailles dans un exécutable ainsi qu'à des changements fréquents dus aux branchements. Nous avons analysé les défauts de TLB instructions pour le PII et le PIII.

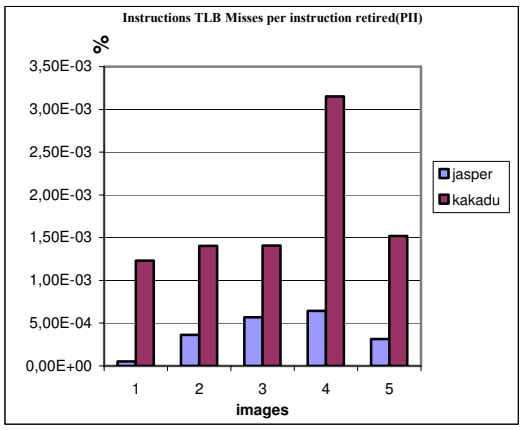


Figure Ch IV - 25 : Instructions TLB Misses /Instructions Retired (PII)

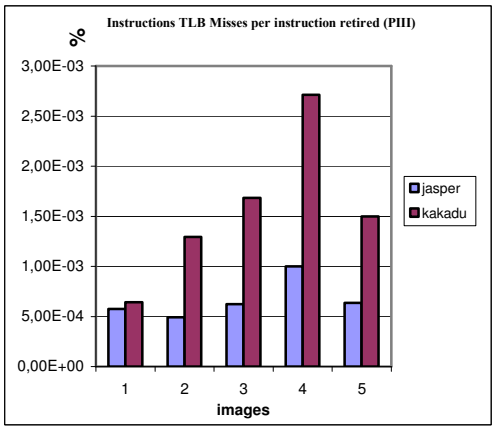


Figure Ch IV - 26 : Instructions TLB Misses /Instructions Retired (PIII)

2.5.4 Impact de la Prédiction de Branchements

L'analyse de la prédiction des branchements dans JPEG2000 est importante. En effet le codeur entropique en particulier effectue de nombreux tests qui se traduisent par des instructions de branchements conditionnels affectant directement les performances des

processeurs à pipeline profond. Quoiqu'il soit possible de réduire le nombre des ces tests en observant que l'échange conditionnel dans la procédure MQ-encoder par exemple, doit toujours être accompagné de renormalisation et que la plupart des symboles ont des probabilités telles que la renormalisation est relativement peu fréquente. Il est donc, nécessaire d'évaluer ce taux de succès de la prédiction de branchement.

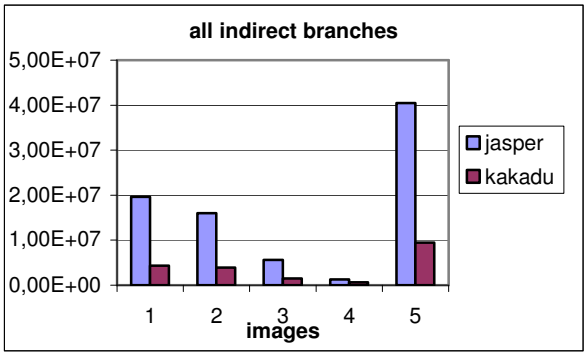


Figure Ch IV - 27 : All Indirect Branches

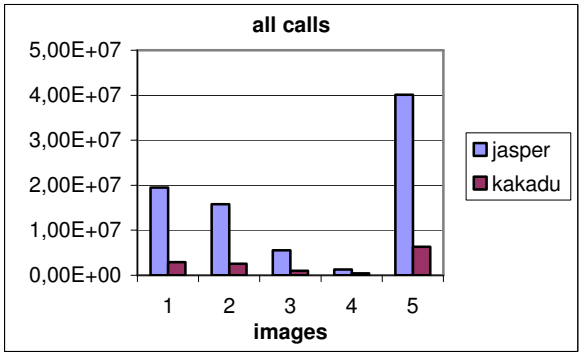


Figure Ch IV - 28 : All Calls

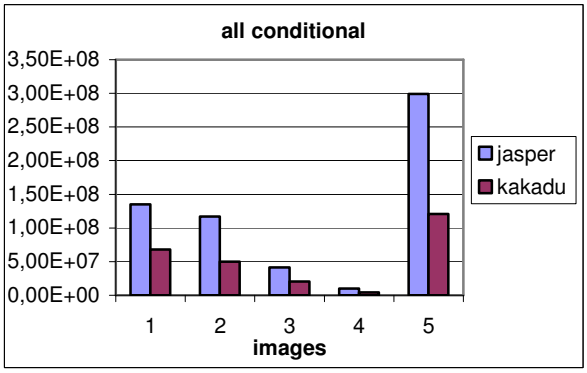


Figure Ch IV - 29 : All Conditionnals

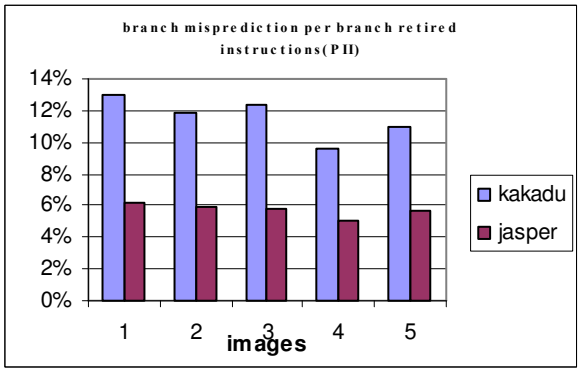


Figure Ch IV - 30 : Branch Misprediction Rate PII

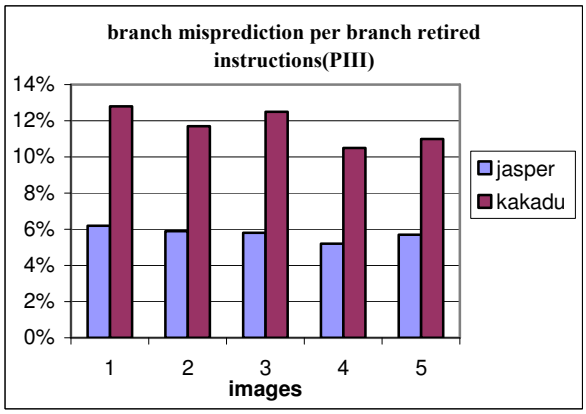


Figure Ch IV - 31 : Branch Misprediction Rate PIII

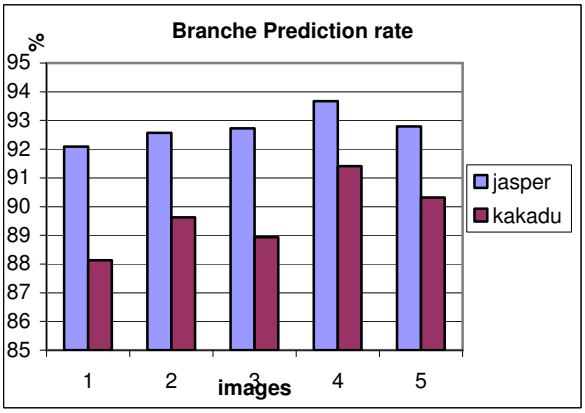


Figure Ch IV - 32 : Prédiction de Branchements (P4)

Les taux obtenus en moyenne de plus de 92% pour Jasper et au dessus de 88% pour Kakadu sont en fait meilleurs que pour la plupart des applications SPEC entières **[18]** et même pour les applications du type desktop **[21]**.

Les BTB (Branch Target Buffer) sont une partie du cache où le processeur sauvegarde les informations sur les branchements précédents. Les défauts de BTB sont aussi relativement faibles.

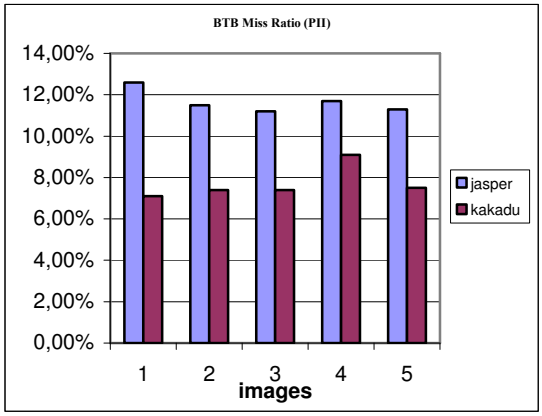


Figure Ch IV - 33 : BTB Miss Ratio PII

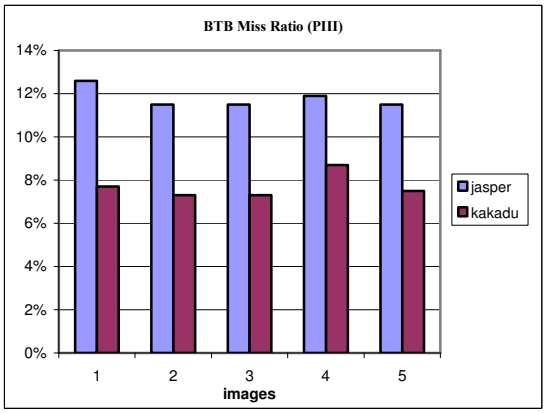


Figure Ch IV - 34 : BTB Miss Ratio PIII

2.5.5 Distribution Dynamique des Instructions et Taux d'utilisation des unités fonctionnelles

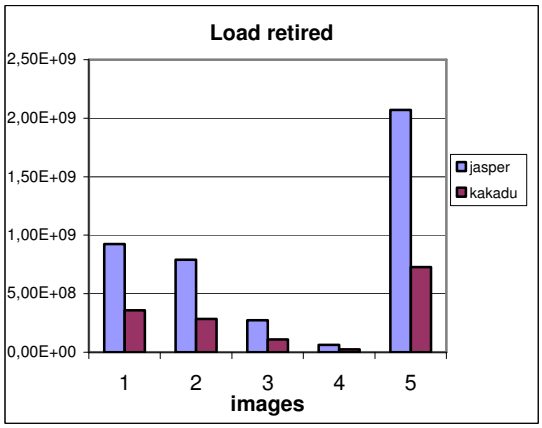


Figure Ch IV - 35 : Nombre d'Instructions de lecture effectuées (P4)

Figure Ch IV - 36 : Nombre d'Instructions x87 retirées (P4)

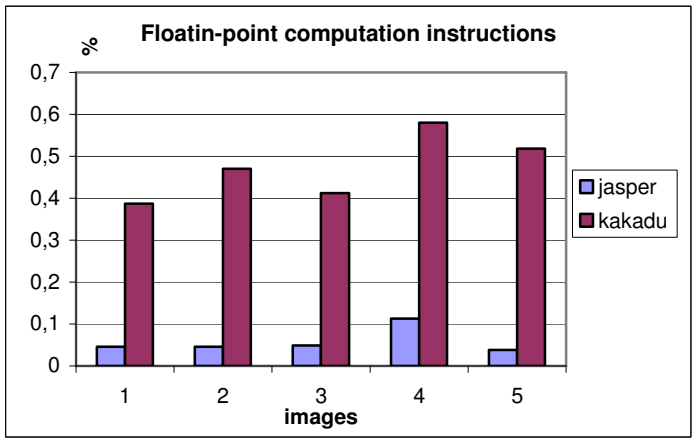


Figure Ch IV - 37 : Nombre d'Instructions Flottantes retirées (P4)

Bien que Jasper soit plus volumineux, les instructions utilisées sont moins complexes que celles de Kakadu.

Les caractéristiques des distributions d'instructions sur les trois processeurs démontrent des distributions très proches des distributions obtenues sur les SPEC CINT (SPEC CPU Integer).

2.5.6 Pentium 4 et Trace Cache

Comme décrit précédemment le Pentium 4 bénéficie d'un trace cache qui a pour rôle de réduire les défauts d'accès au code et de ce fait a le même objectif qu'un cache d'instructions traditionnel. L'approche du Trace Cache est par contre plus sophistiqué puisqu'elle tente de construire des corrélations basées sur des prédictions entre l'accès au code et la prédiction de branchement. L'analyse des deux codeurs JPEG2000 montre un nombre de défauts de Trace Cache plus important pour Kakadu que pour Jasper. Ceci peut être attribué au fait que Kakadu est plus complexe a priori que Jasper mais aussi à la différence de programmation entre C et C++ qui introduit dans ce dernier cas davantage de dynamicité.

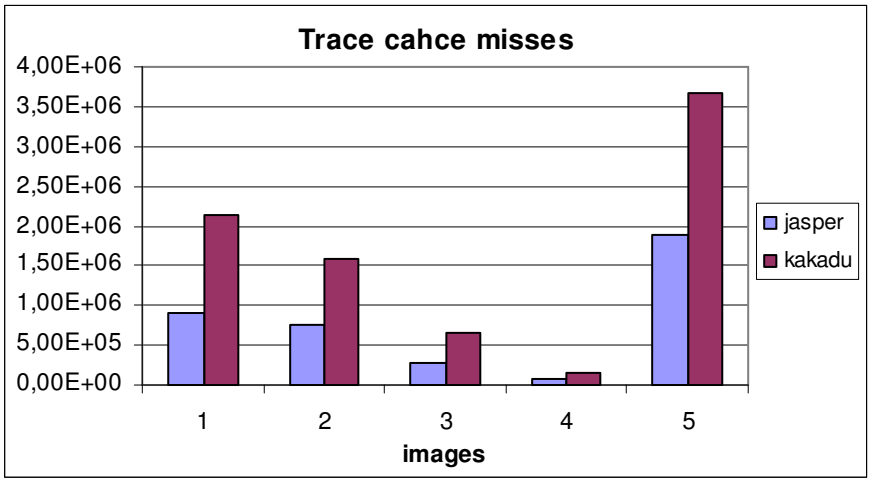


Figure Ch IV - 38 : Trace Cache Miss Ratios (P 4)

2.5.7 Variation du nombre de cycles par pixel.

Nous avons aussi analysé le nombre de cycles par pixel en fonction des formats d'images.

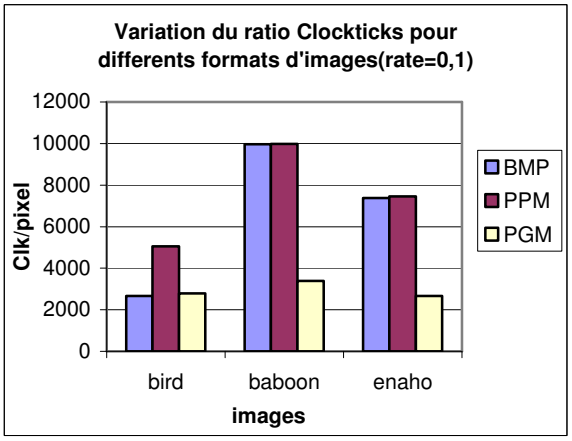


Figure Ch IV - 39 : CPP Taux 0.1 (Jasper /PII)

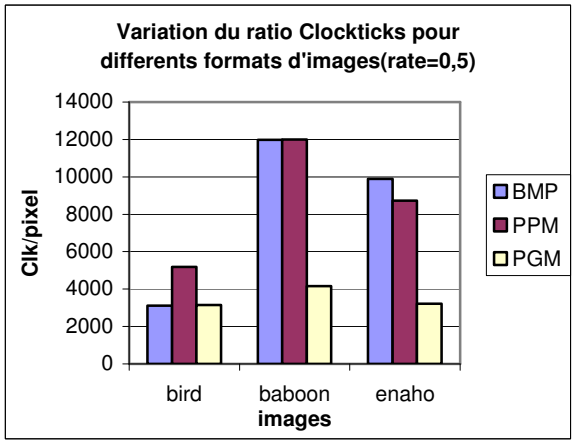


Figure Ch IV - 40 : CPP taux 0.5 (Jasper /PII)

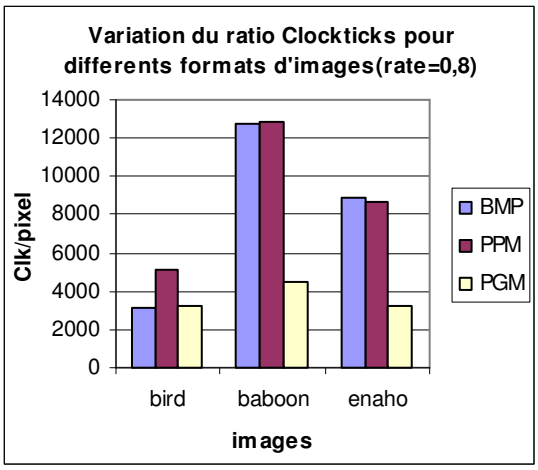


Figure Ch IV - 41 : CPP taux 0.8 (Jasper /PII)

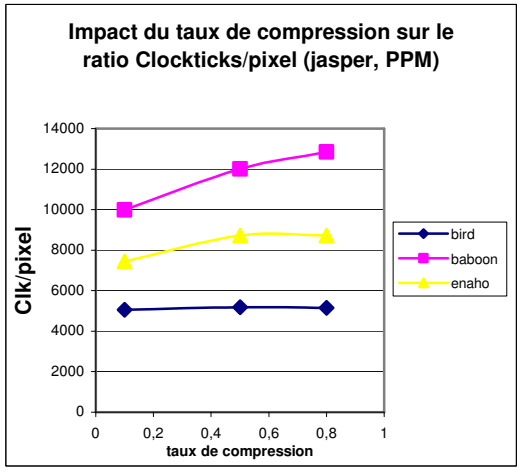


Figure Ch IV - 42 : CPP

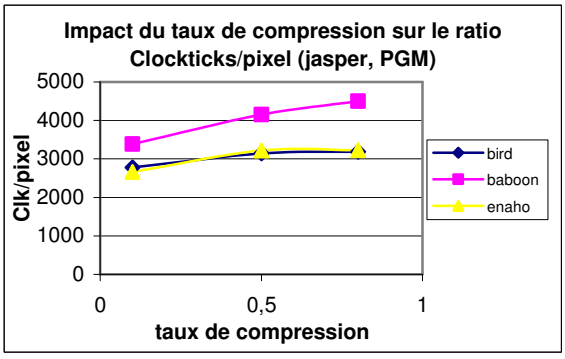


Figure Ch IV - 43 : CPP

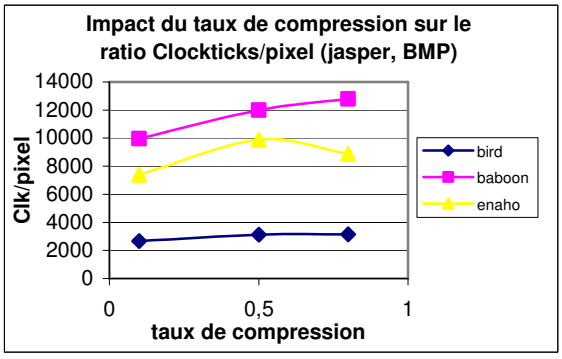


Figure Ch IV - 44 : CPP

Les figures précédentes montrent que le clk/pixel varie différemment en fonction du taux de compression et du format de l'image. Ce qui induit que des choix peuvent être éventuellement effectués pour exploiter cette différence.

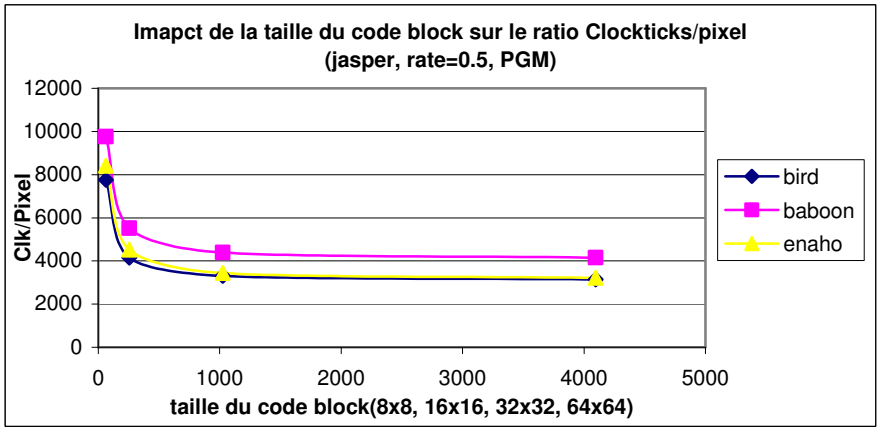


Figure Ch IV - 45 : CPP

Le même travail a été repris pour l'application Kakadu avec les mêmes options. Pour cette application, ces mesures ont été faites par rapport au module principal « kdu_v22D.dll » :

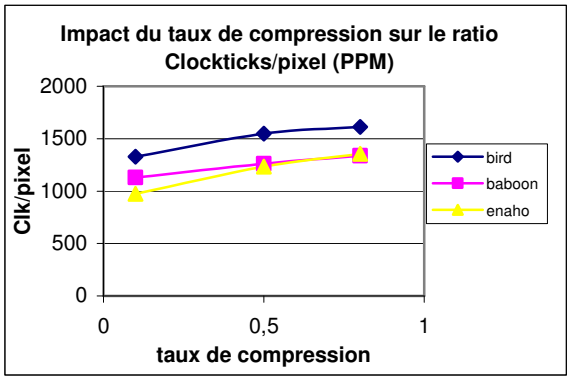


Figure Ch IV - 46 : CPP pour des images ppm

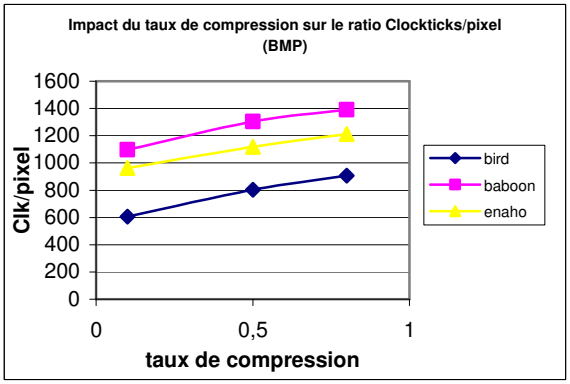


Figure Ch IV - 47 : CPP pour des images bmp

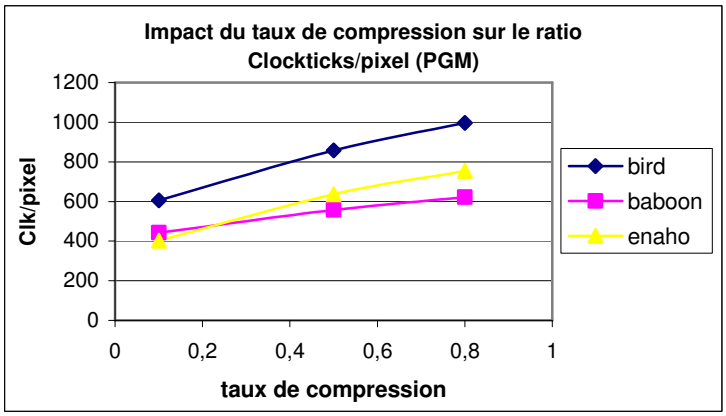


Figure Ch IV - 48 : CPP pour des images pgm

Un des paramètres que nous souhaitions explorer aussi était l'impact de la taille du code block sur clocktick/pixel. On voit que l'augmentation de la taille du code-block réduit de manière non linéaire le nombre de clock/pixel nécessaire.

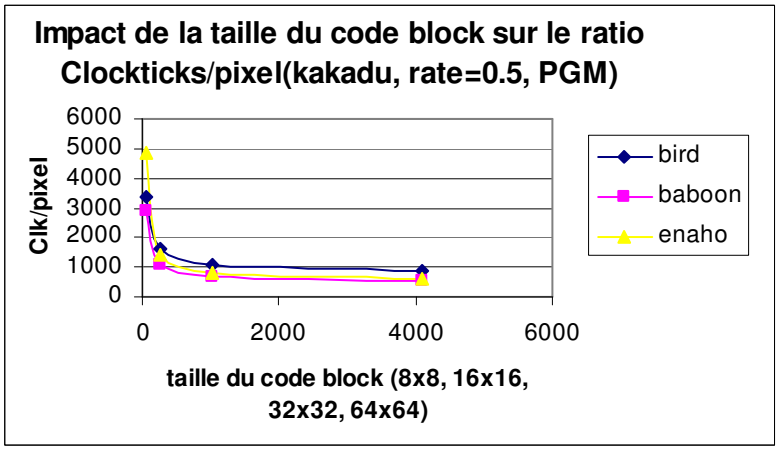


Figure Ch IV - 49 : CPP en fonction de la taille du code-block

2.6 Discussion

Il apparaît à la vue de l'étude précédente que globalement l'exécution sur processeurs généraliste de type Intel peut difficilement être optimisée sans effort important entre autre par un codage au niveau assembleur pour une exploitation maximale du parallélisme des instructions. Le problème de cet effort est : (1) qu'il est difficilement portable sachant que chaque processeur propose des instructions qui lui sont propres et qui potentiellement permettent un gain de temps d'exécution (2) il représente un nombre d'hommes/mois a priori

non négligeable (3) il ne permet pas d'avoir une méthode d'optimisation qui soit suffisamment générique. D'un point de vue microarchitectural il existe de plus une différence importante entre le P4 et la microarchitecture P6 rendant difficile la mise en commun de techniques d'optimisation bas niveau. A un niveau de granularité plus élevé il reste à réécrire le code pour exploiter le parallélisme au niveau thread et le support matériel associé en particulier pour le P4. Là encore l'approche d'optimisation dépend de la machine. D'autres aspects microarchitecturaux auraient pu être avantageusement présentées comme la spéculation de données et de dépendances **[13]**. Une autre alternative radicalement opposée à l'implémentation logicielle optimisée est la conception de circuits dédiés. Dans la partie suivante en essayera d'évaluer l'impacte de certaines optimisations du code, ainsi l'utilisation du multithreading sur la performance de la chaîne de codage JPEG2000 sur une plateforme Pentium 4.

3. Utilisation des librairies SIMD specialisées IPP 3.0

Intel a développé une librairie C, Integrated Performance Primitives (IPP), pour interfacer le précédent jeu d'instructions dans le but de faciliter son intégration et son utilisation. Chaque fonctionnalité du jeu d'instructions possède une unique interface quel que soit le processeur. La librairie contient les différentes versions assembleurs et se charge par le biais de DLLs de détecter le processeur et d'associer aux différentes fonctions C la fonction assembleur correspondante.

En plus de fournir les opérations de base, Intel a inclus dans la librairie IPP trois grands domaines d'application : le traitement du signal, le traitement d'image et les matrices. Pour chaque domaine Intel a déjà implémenté un grand nombre de fonctionnalités standard et des standards nécessaires à tout programmeur travaillant dans ce domaine. Dans la version 3.0, Intel a intégré des étapes du standard JPEG2000. Les différentes étapes sont :

- La transformée couleur de l'espace RGB dans l'espace luminance/chrominance,
- les transformées en ondelettes réversible (entiers signés sur 16 bits) et irréversible (flottants signés sur 32 bits),
- le codeur entropique (données signées 32 bits).

La libraire fournit de plus les opérations inverses pour l'implémentation d'une chaîne de décodage. Les caractéristiques de ces fonctions écrites en assembleur sont le respect du standard JPEG2000 et une implémentation des différents modes non standards (modes Causal, Reset, Restart,...).

Cette librairie C a été utilisée pour réaliser les implémentations optimisées du codeur entropique JPEG2000. La documentation des IPP concernant le standard JPEG2000 peut être trouvée à la référence suivante **[43]**.

3.1 Analyse des performances

Les implémentations qui ont été testées sont l'implémentation en C avec les fonctions IPP et l'implémentation utilisant la technique de multithreading **[42]**. La chaîne logicielle de codage du projet EIRE sert de référence comme chaîne de codage. De plus une chaîne logicielle concurrente de codage JPEG2000, Kakadu version 3.0.8 **[2]**, a été intégrée dans l'analyse des performances. Le tableau ci-dessous présente les conditions d'évaluation de performances.

Tableau Ch IV - 4 : Environnement de test

Plate-forme de test	
Microprocesseur	Intel Pentium 4 à 3.06 GHz
Mémoire	2 Go de DDR
Système d'exploitation	Microsoft Windows XP édition 2002 Service Pack 1
Environnement de développement	Visual C/C++ 6.0
Librairie IPP	
Version	3.0
Benchmark	Intel Vtune 7.0
Paramètres pour la chaîne de codage	
Type de compression	Sans perte
Taille des code-blocks	32x32
Nombre de niveaux de décomposition pour la transformée en ondelettes	5

Le benchmark utilisé est VTune 7.0 d'Intel. Afin que les résultats obtenus soient cohérents et exploitables, l'utilisation de ce benchmark a toujours eu lieu dans les mêmes conditions. Ces différents collecteurs de données ont les caractéristiques suivantes :

Les critères suivants ont été retenus pour orienter l'analyse des différentes implémentations :

- nombre de cycles et µ-opérations effectués,
- occupation du processeur,
- nombre moyen de threads en cours d'exécution pour une chaîne de codage,
- longueur moyenne de la file d'attente des threads dans le processeur.

3.2 Images de test

Afin de tester la chaîne de codage avec les différentes implémentations du codeur entropique, il a été nécessaire d'utiliser des images de tests. Six images couleurs et deux images en niveaux de gris ont été utilisées pour réaliser les tests. Ces images sont issues de la bibliothèque d'images du projet EIRE. La précision des images originales est de 24 bits par pixel.

Tableau Ch IV - 5 : Base des images de test

Images	Taille (Ko)	Taille compressée (Ko)	Dimensions	Couleur	Nb de code-blocks 32x32
Lena.ppm	769	439	512x512	oui	777
03.ppm	1153	396	768x512	oui	1164
14.ppm	1153	496	768x512	oui	1164
Bike.ppm	1501	728	640x800	oui	1641
Fresque_3583.ppm	3601	1780	1280x960	oui	3678
Station_1280x1024.ppm	3841	1656	1280x1024	oui	3858
Lena.pgm	769	139	512x512	non	259
AERIAL2.pgm	4097	2813	2048x2048	non	4096

Les chaînes de codage qui servent de référence pour tester les implémentations réalisées sont les suivantes :

- Kakadu, version 3.0.8 (Kakadu),

- Chaîne logicielle de codage du projet EIRE (ChRef).

Les différentes implémentations testées sont :

- implémentation du codeur entropique avec les fonctions IPP (ChIPPs),
- implémentation intégrant les fonctions IPP dans 1, 2, 4, >4 threads (ChIPPs_1_T, ChIPPs_2_T, ChIPPs_4_T ou ChIPPs_64_T),
- implémentation intégrant les fonctions IPP avec 64 threads mais avec des codes blocs de taille maximale 64x64 (ChIPPs_64_T_64x64).

Les images sont les suivantes :

Lena.ppm 	Lena.pgm 
03.ppm 	14.ppm 
Bike.ppm 	AERIAL2.pgm 
Fresque_3583.ppm 	Station_1280x1024.ppm 

3.3 Résultats

3.3.1 Nombre de cycles

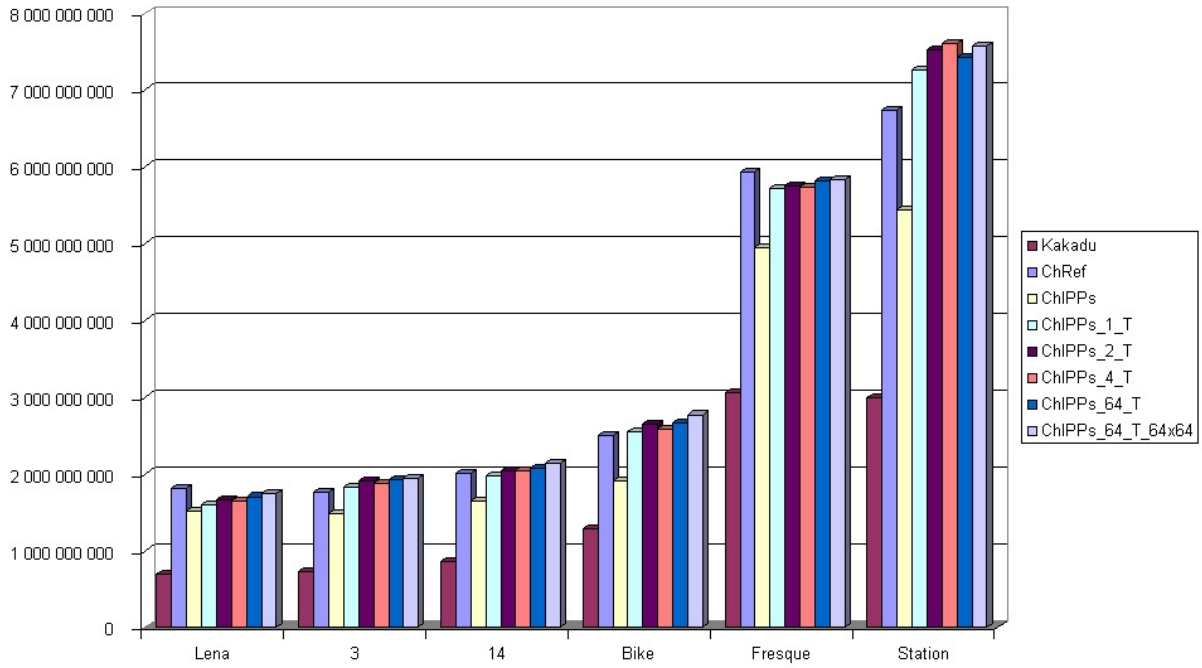


Figure Ch IV - 50 : Résultat : nombre de cycles effectués – images couleurs

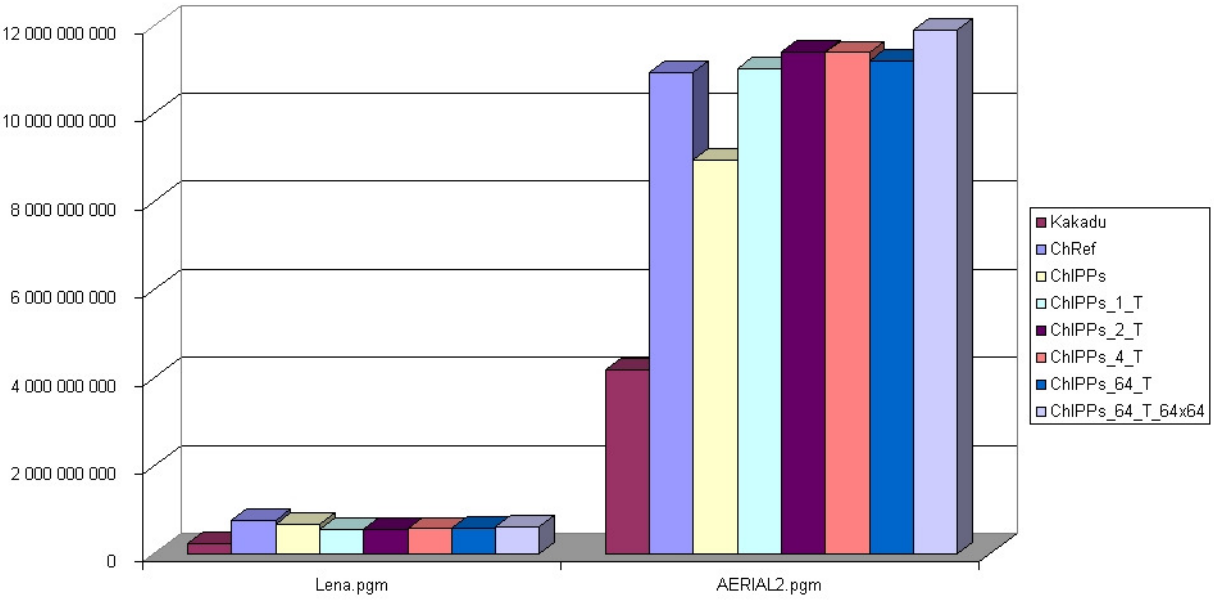


Figure Ch IV - 51 : Résultat : nombre de cycles effectuées – images en niveaux de gris

Tableau Ch IV - 6 : Comparaison entre la chaîne de référence du projet EIRE et cette même chaîne modifiée avec les IPPs

Images (.ppm)	Projet EIRE -Référence (1)	Projet EIRE - IPP (2)	Rapport (1)-(2) sur (1)
Lena.ppm	1 801 064 495	1 506 228 922	16,4%
03.ppm	1 761 430 968	1 486 016 719	15,6%
14.ppm	2 010 770 313	1 646 039 047	18,1%
Bike.ppm	2 504 142 553	1 913 681 700	23,6%
Fresque_3583.ppm	5 928 573 376	4 941 505 932	16,6%
Station.ppm	6 739 276 544	5 432 659 245	19,4%
Lena.pgm	749 182 067	672 850 900	10,2%
AERIAL2.pgm	10 916 202 906	8 949 641 232	18,0%

Kakadu est la chaîne de codage la plus performante. Cette chaîne de codage est écrite en C++ et serait optimisée en utilisant des instructions assembleurs. Kakadu constitue actuellement le logiciel de référence le plus performant pour l'utilisation du standard JPEG2000. En ce qui concerne le projet EIRE, dans tous les cas de figure, la chaîne de codage dont le codeur entropique est réalisé avec les fonctions IPP est plus performante que la chaîne initiale du projet (en moyenne 15% de gain en nombre de cycle). Les versions basées sur des codeurs entropiques réalisées avec les fonctions IPP et intégrées dans des threads perdent l'avantage de l'utilisation des IPP dans la gestion des threads. Cependant les performances restent équivalentes à la chaîne initiale du projet.

3.3.2 µOpérations effectuées

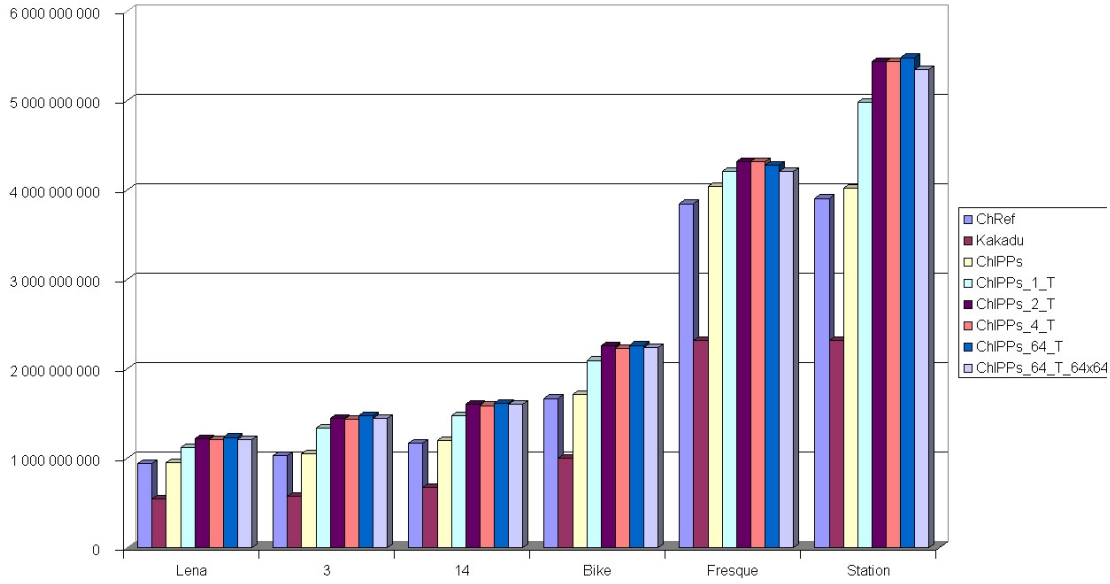


Figure Ch IV - 52 : Résultat : nombre de µ opérations effectuées

En laissant de côté le cas de Kakadu, le critère des µ-opérations montrent que les chaînes de codage à base de fonctions IPP et de threads exécutent plus de µ-opérations que la chaîne de référence. Tout d'abord le rajout de la gestion des threads entraîne un coût supplémentaire non négligeable. Ensuite le jeu d'instructions SIMD devrait théoriquement diviser par un

maximum théorique de 4 le nombre d'instructions mais il faut remarquer que le codeur entropique traite chaque code-block par un parcours séquentiel des bit-planes.

Le jeu d'instructions SIMD est performant dans le cas de calculs parallèles, ce qui n'est pas le cas ici. De plus pour pouvoir intégrer les fonctions IPP au sein de la chaîne de codage du projet EIRE il a été nécessaire de procéder à un formatage des données. En effet la chaîne du projet fournit des données signées 16 bits en entrée du codeur entropique alors que les appels fonctions IPP nécessitent des données signées 32 bits. Une opération de shift a été utilisée pour déplacer les 16 bits des données en entrée dans, les 16 premiers bits des données passées au codeur entropique. En précisant au codeur entropique implémenté par la librairie IPP de ne coder que les 16 premiers bit-planes des données entrantes, il était possible d'obtenir la même séquence de données codées quelles que soient les chaînes de codage. En conséquence toutes les chaînes utilisant les fonctions IPP devaient utiliser plus de µ-opérations que la chaîne de codage du projet EIRE.

Comment se fait-il alors que la simple utilisation des fonctions IPP permette un gain de performance comparativement à la chaîne de référence ? La réponse vient de la nature des fonctions IPP. Les fonctions IPP ne sont rien d'autre que des fonctions écrites en assembleur que l'on appelle depuis une interface C. Les résultats montrent que l'utilisation d'un assembleur optimisé de manière spécifique pour une architecture donnée permet de diminuer le temps d'exécution. De plus dans la mesure où ces instructions font bien un appel à l'unité spécialisée correspondante du processeur elles libèrent les autres unités d'exécution qui peuvent être utilisées pour d'autres instructions, d'où un autre gain de temps d'exécution.

Remarque

Les résultats sont identiques pour l'image AERIAL2.pgm. Dans le cas de Lena.pgm les résultats ont été rejetés vu le peu de données contenues dans l'images, l'image est de taille modeste et ne contient qu'une seule composante.

3.3.3 Nombre moyen de threads en cours d'exécution

La version multithread de l'implémentation des IPP a été testée pour connaître le nombre moyen de threads qu'exécute chaque chaîne de codage basée sur cette technique. Sur une plate-forme monoprocesseur il est illusoire de croire que cette technique permettra d'optimiser la chaîne de codage à base d'IPP.

Les spécifications précisent qu'un Pentium 4 équipé de la technologie Hyper-Threading peut exécuter 2 threads en même temps. Cependant comme le Pentium 4 ne possède qu'une seule unité d'exécution pour les instructions SSE2, on ne pourra exécuter qu'un seul thread appelant les fonctions IPP par cycle d'horloge. Pourquoi s'intéresser alors à une version multithread ? La raison est la suivante et sera développée de manière informelle à la fin de ces résultats, le codeur entropique ne code pas tous les bit-planes d'un code-block donné mais seulement les derniers bit-planes non nuls. Mis à part le premier bit-plane de signe, il est courant que les premiers bit-planes en partant du Most Significant Bit-Plane (MSB) soient remplis de zéro. Le codeur entropique ne les codera pas et commencera le codage au premier bit-plane non nul. La première conséquence est que le temps d'exécution est inégal d'un code-block à un autre et que de plus, cela dépend de l'image traitée (voir plus loin des courbes montrant cette charge théorique). Comme chaque code-block peut être codé de manière indépendante il est donc envisageable de créer plusieurs codeurs entropiques et de les déployer en parallèle.

Dans le cas d'une version multithread sur une plate-forme multiprocesseurs, les chaînes de codage peuvent répartir les threads codeurs entropiques sur chaque processeur en laissant au

système le choix de la répartition des charges. Cette répartition se fera de manière dynamique dans le cas de l'implémentation logicielle réalisée et elle est directement exécutable sur toutes plateformes multiprocesseurs d'Intel possédant la technologie SSE2.

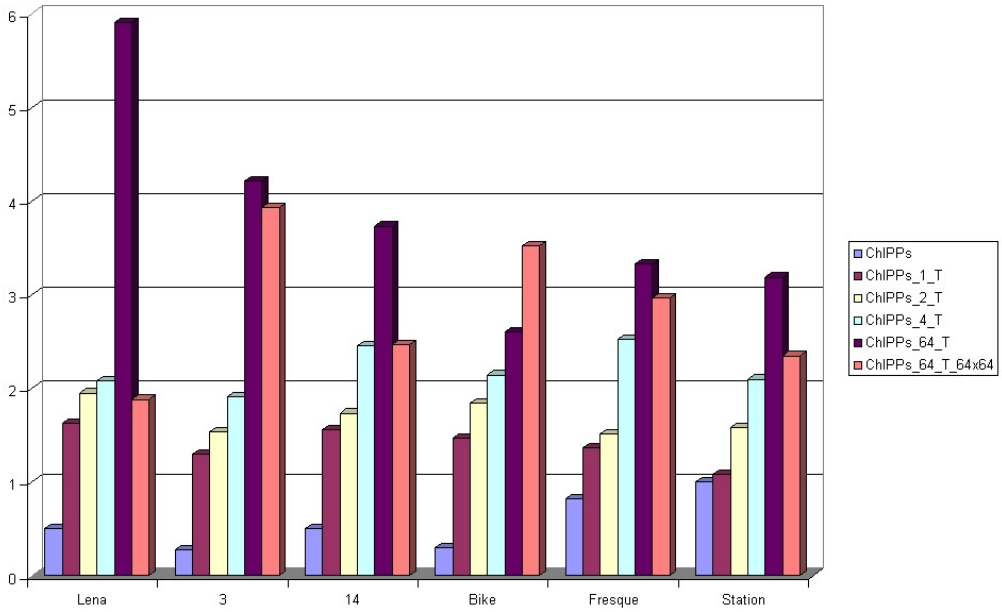


Figure Ch IV - 53 : Résultat : nombre moyen de threads en cours d'exécution

Dans le cas monoprocesseur, comme le thread principal gère la chaîne de codage et délègue l'utilisation des instructions SSE2 pour chaque code-block à un thread fils, on peut envisager que dans le meilleur des cas où un thread fils utilise l'unité d'exécution spécialisée pour les SSE2, le thread principal ou un autre thread fils puisse se servir des autres unités d'exécution du processeur. La version multithread qui utilise au mieux le processeur (soit 2 threads en cours d'exécution) et qui ne sature pas le processeur (>2 threads en cours d'exécution) est la version multithread avec un nombre maximum de threads égal à 4.

3.3.4 Nombre moyen de threads en attente d'exécution au sein du processeur

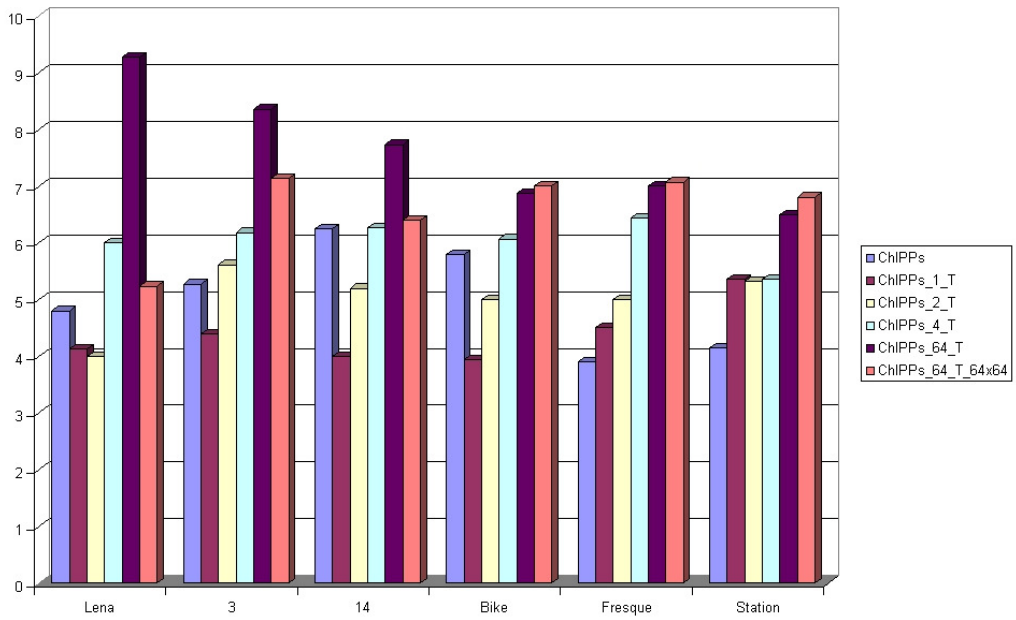


Figure Ch IV - 54 : Résultat : nombre moyen de threads dans la file d'attente du processeur

S'intéresser à la longueur de la file d'attente d'exécution des threads au sein du processeur permet de détecter le cas où des programmes saturent le processeur en threads. Le benchmark utilisé pour obtenir ces chiffres, contraint le temps d'échantillonnage minimal à 50ms ce qui n'est pas optimal vu que l'exécution d'un thread codeur entropique nécessite moins de 50ms. Il semblerait que le comportement de ce critère tend vers le résultat obtenu pour l'image couleur la plus grande (l'image Station_1280x1024.ppm). En conséquence la valeur optimale du critère serait 5 threads en moyenne dans la file d'attente du processeur. Les versions multithreads à 2 et 4 threads seraient donc les meilleures chaînes pour ce critère.

3.3.5 Occupation du processeur

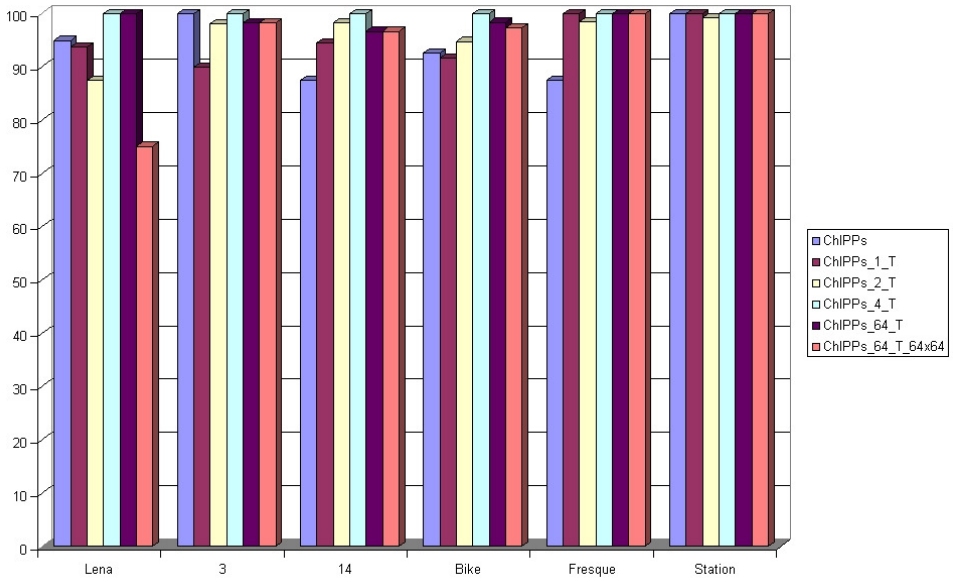


Figure Ch IV - 55 : Résultat : occupation du processeur

Le critère d'occupation du processeur permet de détecter des programmes en attente (accès à des données en mémoire, sur un disque,…). La technologie de multithreading au sein d'un Pentium 4 permet de passer d'un thread à un autre quand ce cas de figure se présente au niveau d'un thread. L'optimisation de cette occupation du processeur est atteinte avec la chaîne de codage utilisant un maximum de 4 threads.

A défaut de profiter de la technique SIMD pour effectuer des calculs massivement parallèles, puisque le codeur entropique est plus efficacement traité en séquentiel qu'en parallèle, les fonctions IPP permettent une implémentation assembleur optimisée du codeur entropique. Cette implémentation s'avère plus performante qu'une implémentation optimisée en full C. Dans le cas d'une plate-forme multiprocesseurs, il est envisageable de déployer des codeurs entropiques en parallèle pour pouvoir coder plusieurs code-blocks de manière simultanée. En effet les versions multithreads ont des performances équivalentes à la chaîne de codage du projet EIRE sur un système monoprocesseur. Donc on ne devrait que les améliorer dans un environnement multiprocesseur.

Dans le cas de l'implémentation multithread, la répartition des charges est réalisée de manière transparente par le système. Il reste cependant à trouver selon le nombre de processeurs disponibles la meilleure configuration concernant le nombre maximal de threads que le système peut créer. Une série de tests devrait permettre d'extraire un modèle de ces meilleures configurations puis de l'intégrer dans la chaîne de codage. Il est possible depuis un code C de connaître en cours d'exécution le nombre de processeurs (ou unités logiques) présents dans le système (par le biais d'instructions assembleurs). Le modèle de

synchronisation des threads pourrait aussi être revu et intégrer un accès aléatoire à la liste de threads en cours d'exécution plutôt que de conserver le modèle séquentiel déjà implémenté.

3.3.6 Répartition de la charge des codeurs entropiques

Il a été abordé dans la présentation des résultats précédents la notion de répartition de la charge de calcul au niveau des codeurs entropiques déployés en parallèle. Les graphiques suivants montrent sur 2 images de taille identique (images 03.ppm et 14.ppm) le nombre de bit-planes réellement codés pour chaque code blocs.

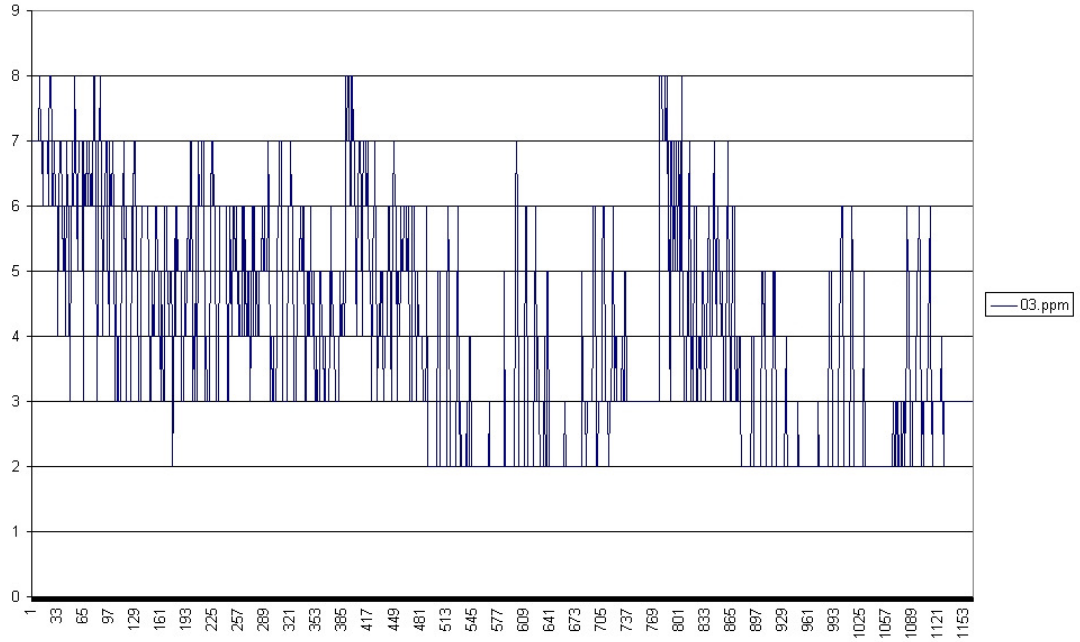


Figure Ch IV - 56 : nombre de bit-planes réellement codés pour l'image 03.ppm

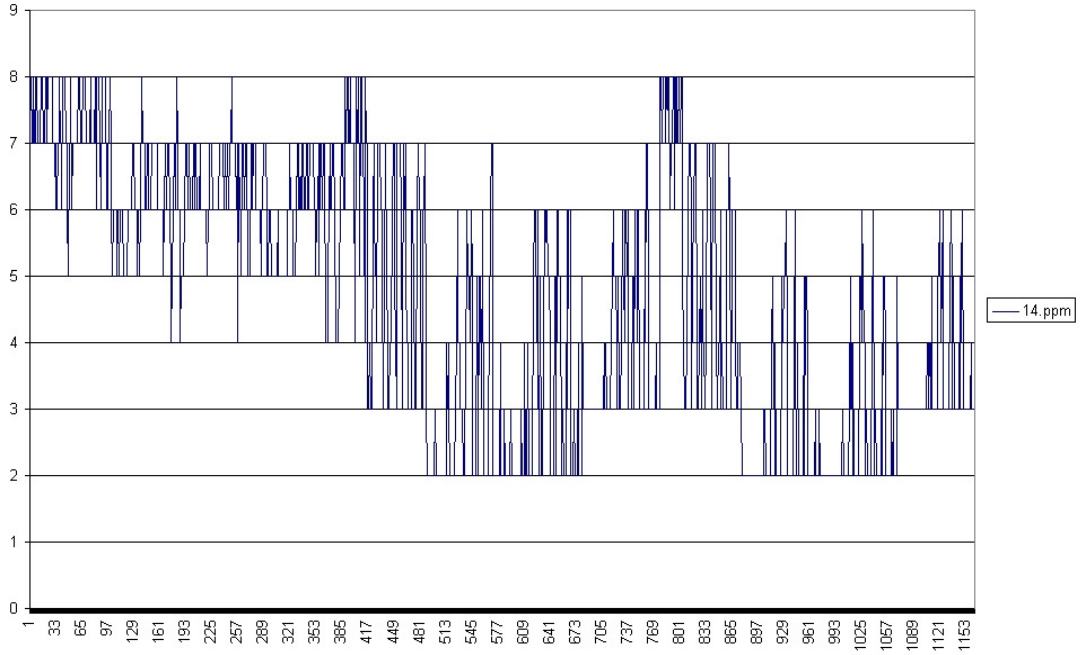


Figure Ch IV - 57 : nombre de bit-planes réellement codés pour l'image 14.ppm

Ces courbes sont différentes et ce ne sont pas des cas isolés. Si l'on suppose qu'en moyenne la charge d'un codeur entropique suit ces courbes, cette charge aura une répartition différente d'une image à une autre. Si l'on utilise un déploiement de codeurs entropiques en parallèle et qu'on répartit le traitement des code-blocks de manière séquentielle, on prend le risque de répartir inégalement la charge de travail des codeurs entropiques.

Dans le cas du déploiement des codeurs entropiques en parallèle par le biais de threads on évite le cas de figure où un ou plusieurs codeurs entropiques attendent que les autres codeurs terminent leur tâche. En effet dans un cas stationnaire où le nombre de threads en cours d'exécution correspond au nombre maximal autorisé de threads, dès qu'un thread/codeur entropique termine sa tâche il peut être remplacé immédiatement par un autre thread/codeur entropique. Virtuellement cela reviendrait à associer un codeur entropique pour chaque code-block de l'image et à n'autoriser l'exécution en parallèle que d'un nombre limité. On pourrait nommer ce principe « un code-block – un codeur entropique ».

Cette étude a permis de démontrer que l'utilisation d'un assembleur optimisée dans une implémentation logicielle du standard JPEG2000 permet d'améliorer les performances en terme de temps d'exécution (**15% de gain constaté pour l'implémentation partielle en assembleur du codeur entropique**). Ainsi l'usage de l'assembleur pourra être étendu à d'autres parties de la chaîne de codage pour améliorer un peu plus les performances de celles-ci. Dans la suite on passera à un autre type de plateforme à savoir les DSP.

4. Evaluation sur plateformes multimédia et DSP

Dans cette partie ont évaluera les performances du codeur JPEG2000 sur la famille de composant TriMedia de chez Philips vues les spécificités architecturales de ce type de composant ainsi que le jeu d'instructions particulier supporté par les composants TriMedia. Ensuite on essayera de réaliser la même opération avec le DSP TMS320C6711.

4.1 Evaluation sur les processeurs TriMedia de Philips

4.1.1 La famille TriMedia

a - Le TM1000

Le TM1000 est le premier membre de la famille TriMedia des processeurs multimédia de chez Philips. Il possède un ensemble d'instructions optimisées pour le traitement audio, vidéo et du graphique 3D. Parmi les caractéristiques du TM1000 on a :

Un processeur VLIW (Very Long Instruction Word) **[23]**, **[28]**, **[45]** dit le DSPCPU pour la coordination de toute les opérations sur puce en plus de son implémentation des parties critiques des algorithmes multimédia. Il utilise aussi, un OS temps réel contrôlé par les interruptions des autres unités.

Une unité d'entrée/sortie multimédia qui opère indépendamment.

Un coprocesseur multimédia qui opère indépendamment et en parallèle avec le DSPCPU pour réaliser les opérations spécifiques aux algorithmes multimédia importants.

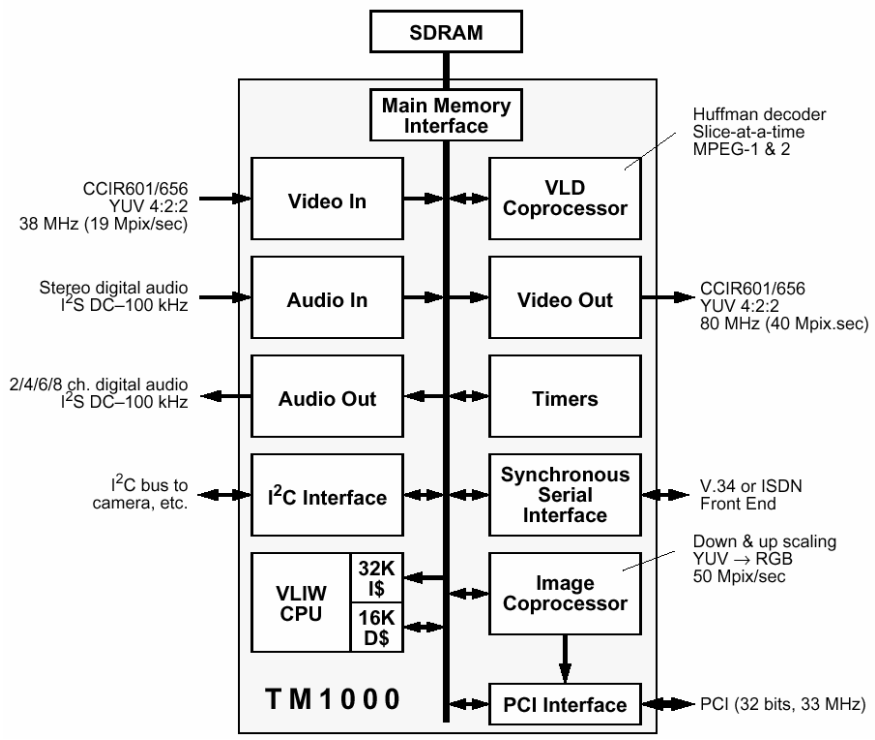


Figure Ch IV - 58 : architecture du TM1000

Le cœur du TM1000 est 32 bits. Ce processeur est dit DSPCPU pour le distinguer des processeurs généraliste, et faire savoir qu'il est conçu pour effectuer du traitement de signal tout en conservant une programmation avec un langage de haut niveau comme tout processeur généraliste. La **figure Ch IV – 58** présente le diagramme bloc du TM1000 **[46]**

Le TM1000 utilise des instructions du type VLIW, qui permettent de réaliser jusqu'à cinq opérations simultanées. Ces opérations peuvent cibler n'importe lesquelles des 27 unités fonctionnelles du DSPCPU, incluant les unités arithmétiques entières, flottantes, et les unités de traitement de données parallèles du type DSP. Le cœur du DSPCPU utilise 16 Ko de cache donnée et 32 Ko de cache instruction. Les deux caches sont séparés. Le cache donnée est dual-port pour permettre deux accès simultanés

b - Le TM1100

C'est le successeur du TM1000, les nouvelles caractéristiques implémentées par le TM1100 sont **[47]**:

- La fréquence de fonctionnement du DSPCPU et du coprocesseur est de 133MHz
- La fréquence de la SDRAM est Jusqu'à 133 MHz
- Amélioration de la sortie vidéo
- De nouvelle instruction du DSPCPU
- Un coprocesseur d'authentification/descrambling de DVD
- Amélioration de la lecture PCI DMA

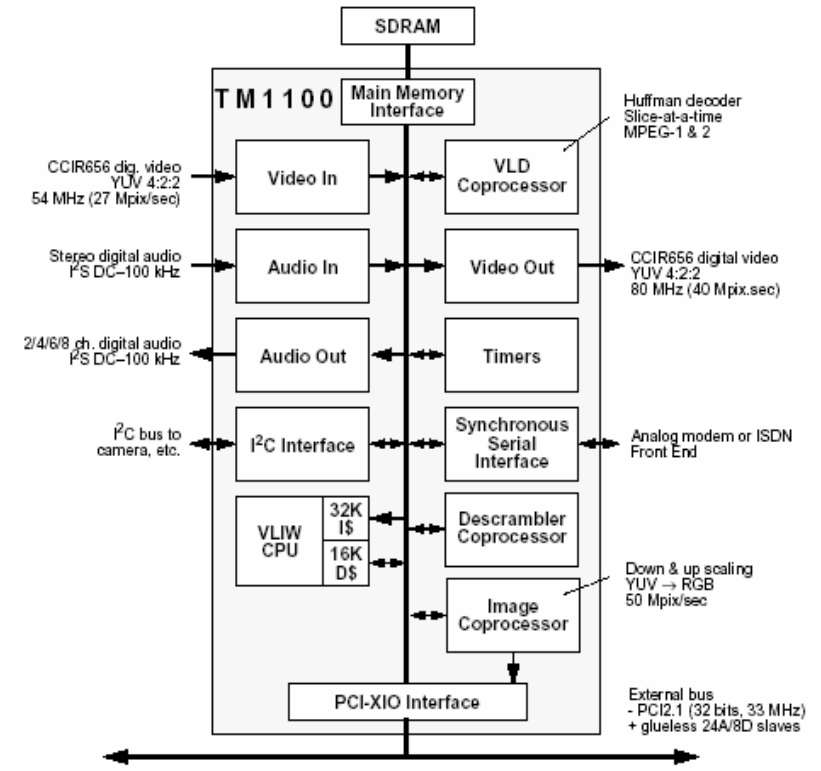


Figure Ch IV - 59 : architecture du TM1100

c - Le TM1300

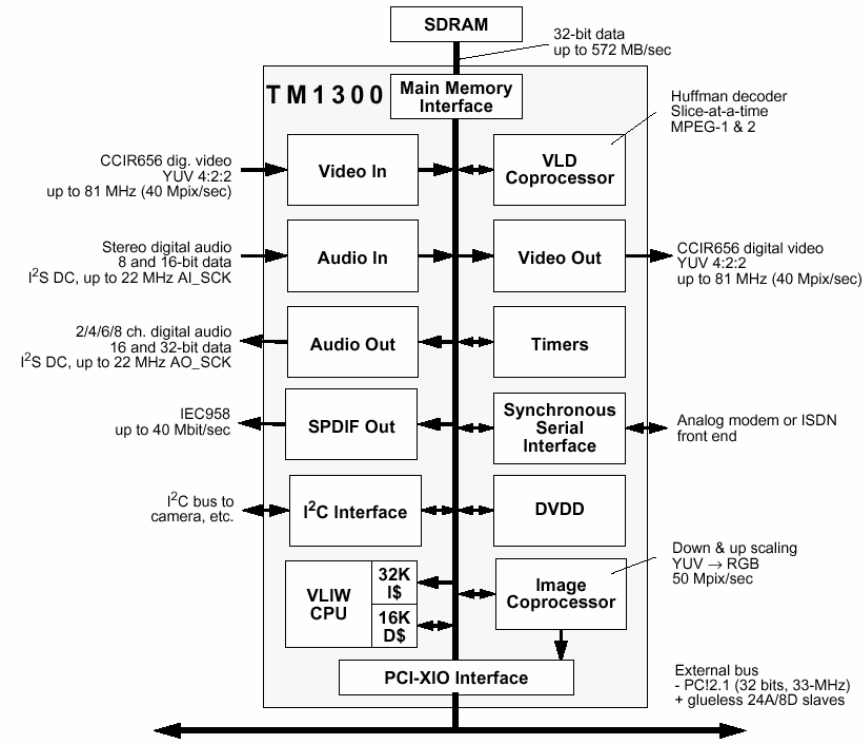


Figure Ch IV - 60 : architecture TM1300

Le TM1300 est le successeur du TM1100. Les nouvelles caractéristiques du TM1300 sont **[48]**:

- La fréquence de fonctionnement du DSPCPU et du coprocesseur est de 166MHz

- La fréquence de la SDRAM est Jusqu'à 143 MHz
- Amélioration de l'Entrée/sortie vidéo : jusqu'à 81 MHz.
- De nouvelles instructions du DSPCPU

4.1.2 L'environnement SDE (Software Development Environment)

Le diagramme suivant (**figure Ch IV - 61**) décrit le flot de développement de l'environnement TriMedia SDE. Le SDE possède un ensemble d'outils pour compiler, déboguer, analyser et optimiser les performances, et faire la simulation pour la famille de processeur TM1x. Le compilateur TriMedia intègre un ensemble de possibilité d'optimisation. Le tableau suivant résume l'ensemble de ces options avec la correspondance avec chaque niveau d'optimisation. Les optimisations du compilateur TriMedia sont basées sur le profilage (profiling), le déroulement des boucles, l'inlining de fonction, et l'analyse d'alias.

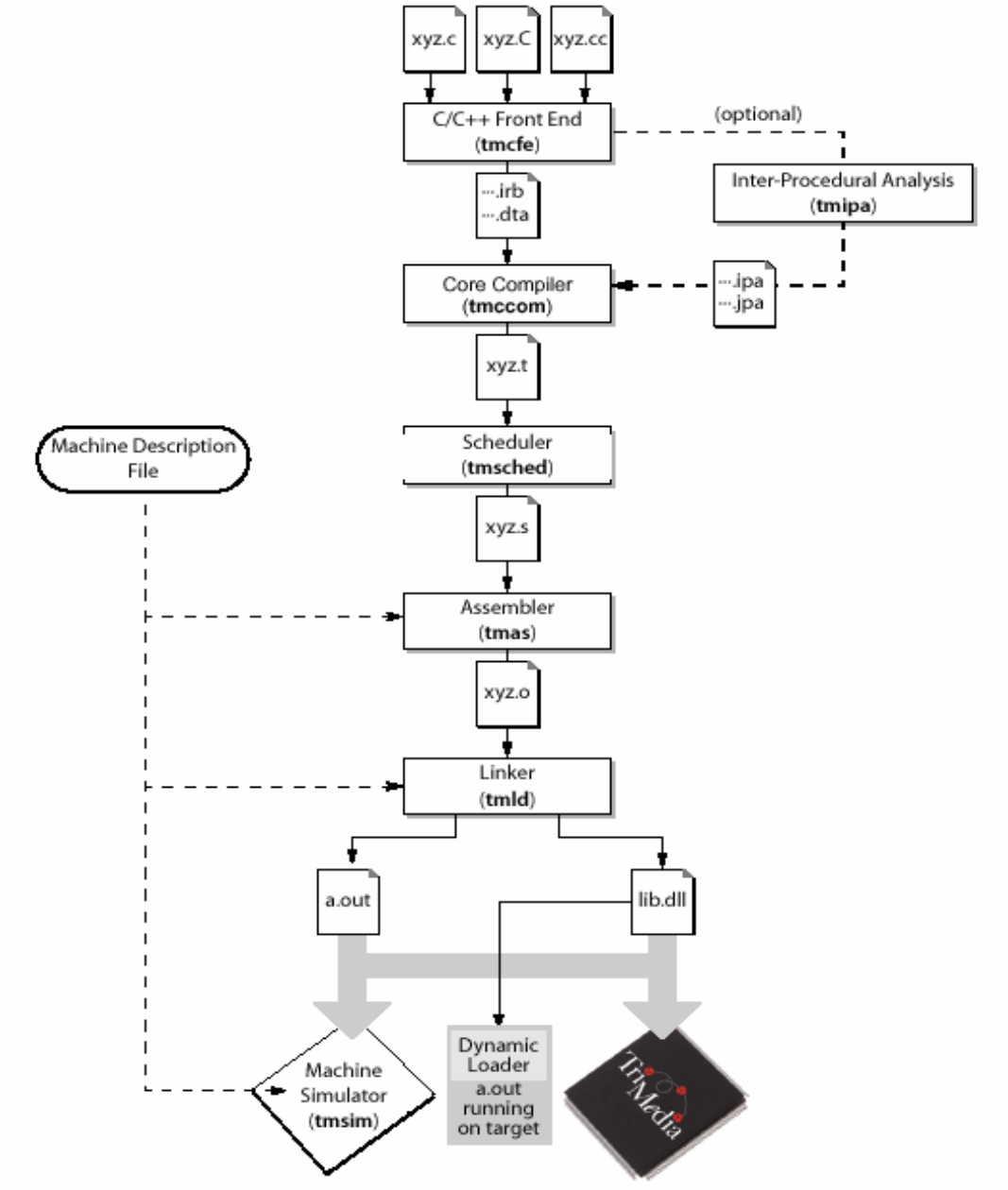


Figure Ch IV - 61 : Flot de développement de SDE

Tableau Ch IV - 7 : Les options d'optimisation du compilateur de SDE

		Optimization Level					
Optimization	Variable	-O0	-O1	-O2	-O3	-O4	-O5
Alias Analysis	-Xalias	0	1	3	3	4	4
Call Modification analysis	-Xcallmod	0	1	1	1	2	2
Constant Propagation	-Xconstp	0	0	2	2	2	2
Copy propagation	-Xcopyp	0	0	2	2	2	2
Loop Forward Code Motion	-Xfcm	0	0	1	2	2	2
Control Flow Analysis	-Xflow	0	0	1	1	1	1
Inlining Level	-Xinllev	0	0	0	0	1	1
Main Optimization	-Xmopt	0	1	3	4	4	4
Register Allocation	-Xreg	0	0	1	1	3	3
Loop Unrolling	-Xunroll	0	0	1	1	1	1
Extra Optimization	-Xxopt	0	0	2	2	3	5
Express Reordering	-Xzone	0	0	1	1	1	1

4.1.3 Les mesures de performances

a - Mesure du CPI (cycle per Instruction)

Pour faire l'évaluation du codeur JPEG2000 sur les différentes plateformes TriMedia, on a choisi JasPer 1.500.4 comme implémentation. Comme cette implémentation du codeur utilise des entiers 64 bits, des modifications ont été apportées à JasPer afin de le rendre compatible avec les architectures cibles. Toutes les mesures on été réalisées par le profiling au niveau simulation (par d'implémentation sur carte TriMedia). Les options utilisées pour effectuer les différentes mesures sont :

Au niveau compilation

- Optimisation globale –O0, -O3, -O5
- Boucle : -Xunroll= 0, 5, 10

Au niveau simulation

Le simulateur supporte la simulation au niveau cycle de la puce et de la mémoire externe. Il permet la collecte de statistiques concernant le fonctionnement du système tel que le nombre de cycle, les défauts de cache, les instructions exécutées en parallèle,…

- L'architecture cible TM1000, TM1100
- Taille de la mémoire : 256 Mo, 512 Mo

Pour les différentes combinaisons des deux architecture cibles et des mémoires utilisées on a mesuré le CPI dans chaque cas. Dans cette étude, plusieurs images ont été utilisées pour permettre une comparaison de l'impact des données d'entrée sur les performances de la chaîne de codage sur ce type d'architecture.

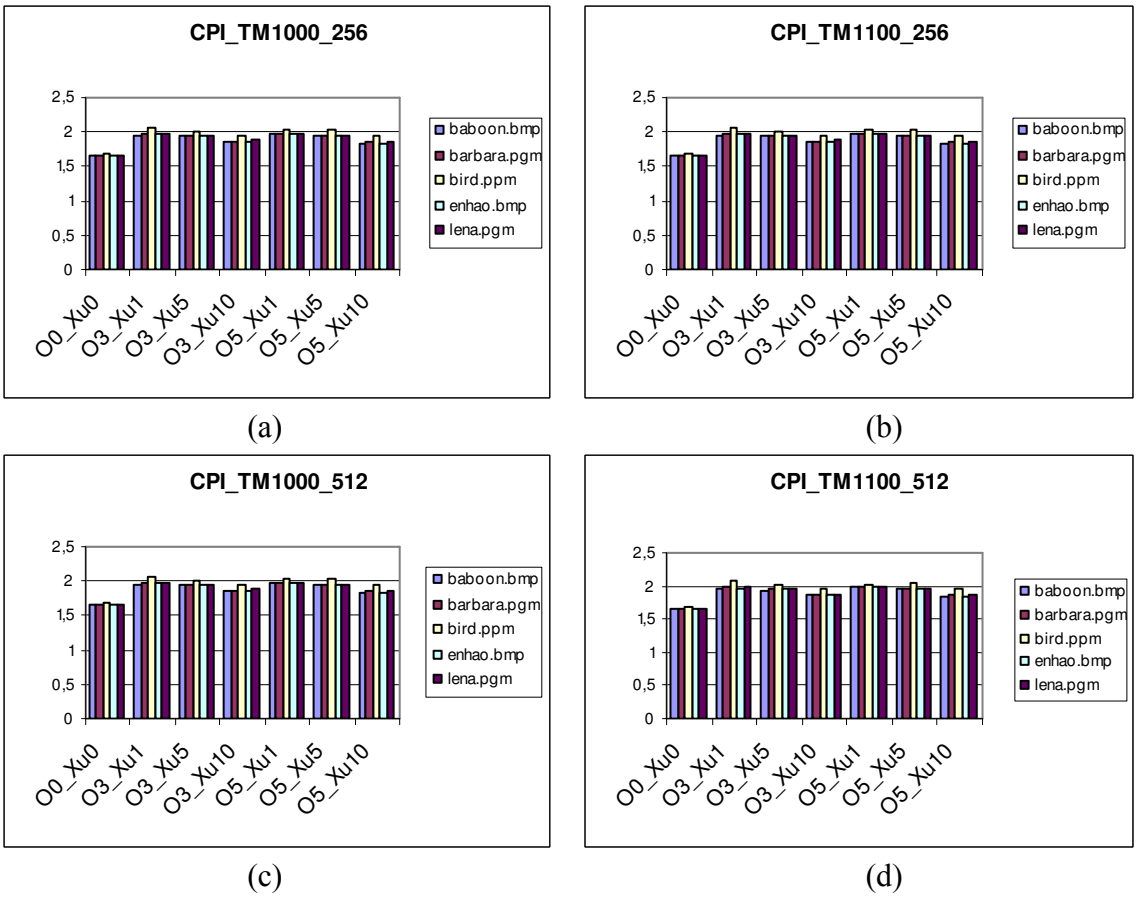
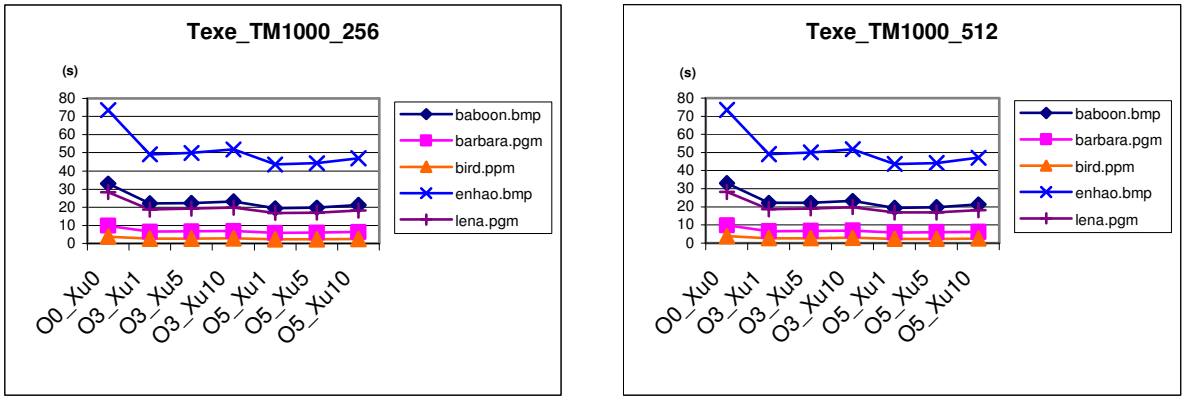


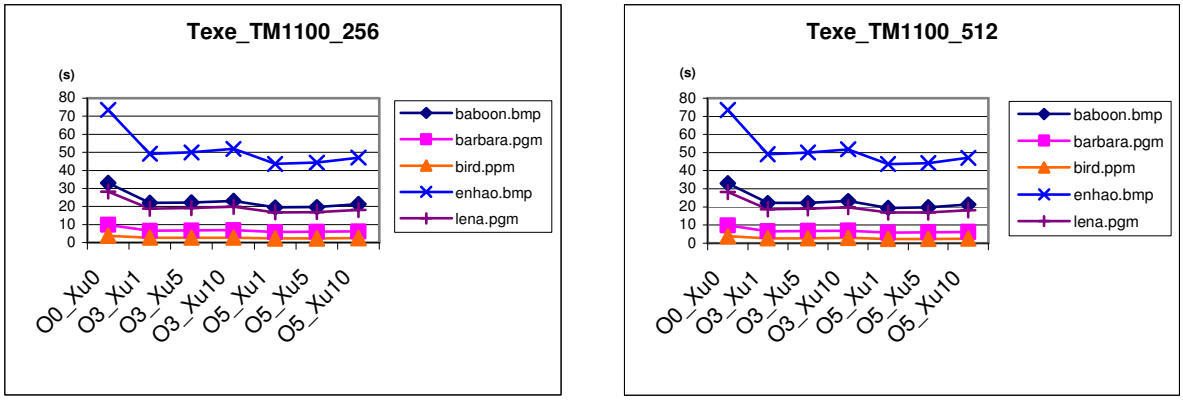
Figure Ch IV - 62 : Mesures du CPI

Selon ces figures on remarque que les meilleurs CPI sont obtenus pour les options -O0 et -Xu0 (Optimisation faible). Le CPI est d'autant plus élevé que les options d'optimisation sont fortes. Cela peut être dû au type d'instructions utilisées, et qui deviennent de plus en plus complexes avec l'augmentation du niveau d'optimisation. De plus le CPI est grand pour les images de petites tailles telle que l'image bird.

b - Mesure du Temps d'exécution

Sur les figures suivantes on remarque que l'impact de l'optimisation est plus visible pour les images de grandes tailles comme l'image enaho.bmp, par contre pour les petites images l'effet de l'optimisation est presque imperceptible.





(c) (d)

Figure Ch IV - 63 : Temps d'exécution

4.2 Evaluation sur le DSP de Texas Instruments

4.2.1 Le DSP TMS320C6711

Le TMS320C6000™ est une plateforme DSP (Digital Signal Processor) membre de la famille des DSP TMS320™. La génération des DSP TMS320C67x (comprenant les composants : TMS320C6711, TMS320C6711B, TMS320C6711C, et TMS320C6711D) intègre des composants à virgule flottante. Elle utilise l'architecture VelociTI, qui est une architecture VLIW avancée. L'architecture VelociTI est très déterministe, elle possède quelques restrictions sur où et quand le fetch, l'exécution, la sauvegarde des instructions sont effectués. Les caractéristiques de l'architecture VelociTI permettent de réduire la taille du code, une meilleure flexibilité du code et des types de donnée, et d'avoir des branchements totalement pipelinés. Le processeur C67x est composé de trois principales parties : le CPU, les périphériques, et de la mémoire. Il dispose aussi de huit unités fonctionnelles qui peuvent opérer en parallèle. Ces unités sont :

- 4 ALU (Arithmetic and Logic Unit) virgule fixe et flottante,
- 2 ALU virgule fixe,
- 2 multiplicateurs virgule fixe et flottante,

Le parallélisme est déterminé pendant la compilation car il n'y a aucun mécanisme matériel pour l'analyse de la dépendance des données. Le bus donnée de la mémoire du programme de largeur 256 bits permet la lecture simultané de 8 instructions de 32-bits chacune **[50]**.

Le C67x possède une architecture à deux niveaux de cache. Le cache programme de niveau 1 (L1P) est un cache de 32-Kbit. Le cache données de niveau 1 (L1D) est de 32-Kbit (2-way set-associative cache). Le niveau 2 mémoire/cache (L2) se compose d'un espace mémoire de 512-Kbit qui est partagé entre le programme et les données. La mémoire L2 peut être configurée en tant que : mémoire, cache, ou combinaisons des deux. Parmi les périphériques on trouve un contrôleur DMA et Une interface avec de la mémoire externe **[51]**.

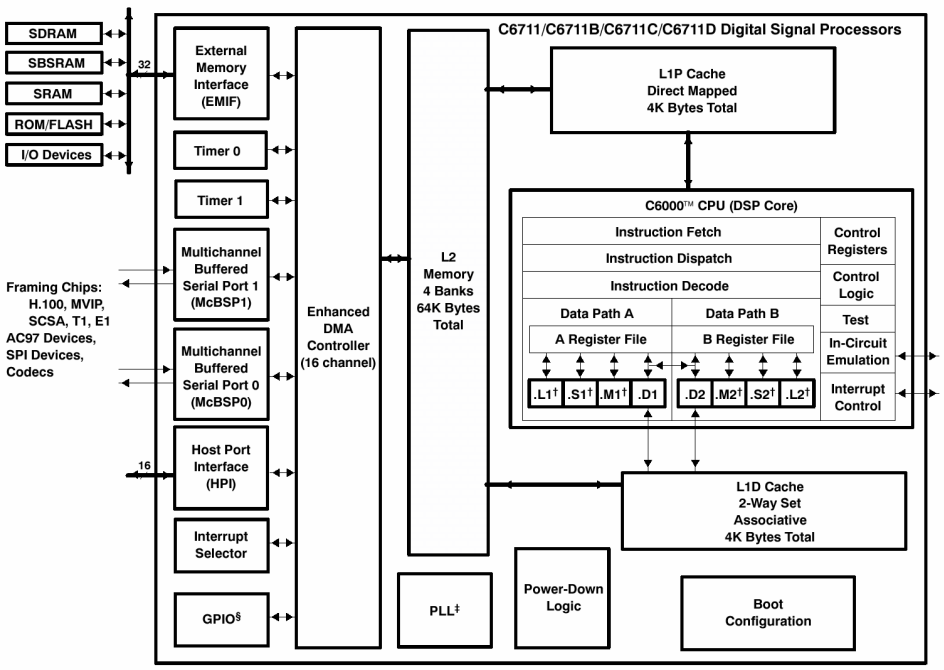


Figure Ch IV - 64 : Architecture du TMS320C6711

Le CPU possède deux chemins de donnée (A et B) dans lesquels s'effectue le traitement. Chaque chemin de donnée possède 4 unités fonctionnelles (.L, .S, .M, .D) et un banc de registre de 16x32 bits.

Le C6711/C6711B peut exécuter jusqu'à 900 millions d'opérations à virgule flottante par seconde (MFLOPS) à une fréquence de 150 MHz. Le C6711/C6711B peut produire deux MAC par cycle pour un total de 300 MMAC.

Le C6711C/C6711D peut exécuter jusqu'à 1200 MFLOPS à une fréquence de 200 MHz. Ces performances peuvent atteindre 1500 MFLOPS pour une fréquence de 250 MHz (pour le C6711D). Il peut aussi produire deux MAC par cycle pour un total de 400 MMAC.

4.2.2 Code Composer Studio™ v2

Le Code Composer Studio (CCS) est un environnement de développement qui supporte les plateformes DSP TMS320C6000 et TMS320C5000 de Texas Instruments **[52]**. Il supporte toutes les phases de développement à savoir la conception, le codage, le déboguage, l'analyse et l'optimisation. Ces principaux éléments sont:

- un environnement de développement qui intègre un éditeur, un débogueur, un gestionnaire de projet, et un profileur
- compilateur C/C++, assembleur et linker (outils de génération de Code) **[49]**
- simulateur d'Instruction
- un noyau temps réel logiciel (DSP/BIOS™)
- un échange temps réel entre l'hôte et la cible (RTDX™)
- des outils d'analyse et de visualisation temps réel des données

Le PCB (**P**rofiling **B**ased **C**ompiler) permet de déterminer facilement d'explorer le compromis entre la taille du code et les performances. Il permet de déterminer les performances de chaque fonction sous différentes options de compilation et de choisir les options qui conviennent le plus avec les besoins de l'application

Lors de la tentative d'évaluation des performances de l'implémentation JasPer de JPEG2000 sur la plateforme TMS320C6711, un problème de compatibilité est survenu. Il s'est avéré que les librairies utilisées pour l'implémentation de JasPer n'étaient pas totalement transportables pour l'environnement du CCS. Cela ne nous a pas permis d'avancer sur cette évaluation ni d'avoir des résultats de simulation ou d'implémentation.

4.2.3 Quelques exemples d'utilisations du DSP

Plusieurs travaux dans le domaine du traitement de l'image et du signal ciblent la famille des DSP TMS320C67x. Dans **[53]**, **[54]** on trouve une étude des possibilités d'optimisation en vu d'une implémentation temps-réel de l'algorithme de traitement d'image Retinex. Cette implémentation servira pour un système de vision utilisé par les pilotes d'avions. Le travail de **[56]** est une comparaison entre deux implémentations de la transformée en ondelettes : celle du lifting scheme et celle d'un banc de filtres. Le papier **[55]** présente l'optimisation du code du HVXC MPEG-4 pour le codage de la parole en vu d'une implémentation temps-réel. L'étude présentée dans **[57]** nous concerne particulièrement, car il s'agit de l'implémentation de Kakadu sur le TMS320C6701. Des optimisations avec les options de la compilation et du code assembleur on été utilisées pour améliorer les performances. Les techniques d'optimisation sont basées principalement, sur l'utilisation d'une meilleure allocation de la mémoire, l'activation du pipeline logiciel, l'utilisation des fonctions, de mots clés, et des directives intrinsèques, et le paquetage des données. Cette étude a engendré un gain global en temps de 17% (DWT 31%, Quantification 35%, transformée en couleur 43%)

5. Discussion

La première étape de l'approche suivie dans ce chapitre, consiste à analyser la complexité du codeur JPEG2000 et à maximiser ses performances sur des plateformes de type généralistes ou spécialisés et ce dans le double objectif : d'établir des limites supérieures de performance, et de décider si une conception ultérieure de circuits dédiés sera nécessaire. La première étude concerne les plateformes à base de processeurs Intel Pentium II, III, et 4 sur des PC avec différentes configurations. Suite aux conclusions tirées à partir de cette évaluation nous avons effectué une première optimisation en utilisant des librairies spécialisées exploitant les unités vectorielles SIMD ainsi que le multithreading. Cette combinaison représente l'ensemble des niveaux de parallélisme pouvant être atteints sur ce type de machine : parallélisme pipeline superscalaire, parallélisme SIMD, et multithreading. Il en résulte que pour deux codes distincts : JasPer et Kakadu, les compilateurs n'exploitent pas suffisamment le parallélisme intrinsèque de l'application. Plus particulièrement le multithreading peut au-delà de deux threads avoir un effet inverse et dégrade les performances d'un code. Aussi, l'utilisation des instructions SIMD reste restreinte à des parties très localisées du code.

La seconde étude a été effectuée sur des processeurs multimédia de Philips de la famille TriMedia, qui présentent une architecture du type VLIW (TM1000, TM1100) dont l'exploitation efficace dépend fortement de la qualité des compilateurs associés. Pour chacun de ces deux processeurs on a utilisé deux tailles de mémoires différentes (256 Mo, 512 Mo). On a mesuré le CPI ainsi que le temps d'exécution pour les différentes architectures en faisant varier les options d'optimisation du compilateur. Cette étude nous a permis de remarquer que les deux processeurs TriMedia sont peu adaptés à ce type de traitement sauf peut être si on utilise une implémentation optimisée pour les architectures VLIW. En effet le compilateur n'a pas pu apporter des améliorations importantes dans le temps d'exécution.

La dernière plateforme utilisée, est basée sur le DSP TMS320C6711 de Texas Instruments. Cette plateforme n'a pas été exploitée vu les problèmes de réadaptation de code auxquelles on n'a pas trouvé de solution.

Cette étude nous a permis de conclure que les possibilités d'optimisation sur les processeurs généralistes ou spécialisés sont limitées, par conséquent il faut prévoir d'autres issues, notamment le traitement coprocesseur dans un premier temps.

Nous souhaitons nous orienter vers une conception SOC **[30-31]** de familles d'IP pour JPEG2000 qui varieront suivant les applications et contraintes associées. Nous voulons dans un premier temps développer un modèle SystemC **[32-33]** du codeur JPEG2000 et effectuer de nombreuses explorations de partitionnement logiciel matériel sur la base d'une architecture multiprocesseur sur puce **[27]** avec des circuits dédiés reconfigurables qui sera notre modèle principal d'architecture (plateforme). Cette exploration se fera avec des outils propriétaires au Laboratoire permettant d'effectuer des évaluations précises au niveau cycle des architectures tout en paramétrant au maximum toutes les différentes caractéristiques essentielles au niveau architectural. Il est à noter que plusieurs sociétés, comme il a été présenté précédemment dans le chapitre II et dans ce chapitre, proposent ou travaillent sur des optimisations de ce type par l'offre de bibliothèques spécialisées, ou d'IP synthétisable.

Les orientations sont le test de l'approche coprocesseur dédié par l'utilisation d'une plateforme Celoxica **[40]** à base de circuits FPGA Virtex permettant d'émuler le mode d'exécution souhaité. L'exploration d'une implémentation sur FPGA est absolument incontournable d'autant plus que la nouvelle génération de circuits Xilinx Virtex-II (voir le chapitre VI) inclura jusqu'à 4 processeurs PowerPC sur la même puce ce qui se rapprochera complètement de notre volonté de concevoir un système sur puce avec circuit dédié reconfigurable.

6. Conclusion

Nous avons présenté dans ce chapitre une étude au niveau microarchitectural des ressources utilisées par des codeurs JPEG2000 sur des plateformes du commerce. D'abord nous avons effectué une évaluation sur de plateforme Pentium II, III, et 4 d'Intel. Suite aux conclusions tirées à partir de cette évaluation nous avons tenté une première optimisation en utilisant des librairies spécialisées ainsi que une exécution multithread. Ensuite nous avons utilisé une autre catégorie de composants qui sont les DSP. Une première évaluation au niveau SW a été réalisée sur les composants TriMedia de Philips, puis sur le DSP TMS320C6711 de Texas Instruments. Finalement nous avons dérivé un ensemble de conclusions pour notre approche pour une implémentation optimisée de JPEG2000.

Dans la partie suivante on va présenter deux études de l'utilisation du traitement coprocesseur sur deux plateformes différentes. La première plateforme est un PC avec une carte PCI, la seconde, est un systèmes embarque. Dans les deux cas le coprocesseur est implémenté sur un composant FPGA.

7. Références

[1] M.D.Adams, "Jasper Software Reference Manual", V.1.500.4, ISO/IEC JTC1/SC29/WG1 (ITU-T SG8) N2415, Dec. 2001.

[2] Kakadu Software www.kakadusoftware.com

[3] ISO/IEC 9945-1: Information Technology – JPEG 2000 image coding system – Part 5: Reference Software, 2002.

[4] IA-32 Intel Architecture Software Developer's Manual, Volume 1 : Basic Architecture, Intel Corp.www.intel.com

[5] IA-32 Intel Architecture Software Developer's Manual, Volume 2 : Instruction Set Reference, Intel Corp.www.intel.com

[6] IA-32 Intel Architecture Software Developer's Manual, Volume 3 : System Programming Guide, Intel Corp.www.intel.com

[7] Pentium III Reference Manual

[8] Pentium II Processor Developer's Manual, Intel Corp.

[9] G.Hinton and al.,"The Microarchitecture of the Pentium 4 Processor", Intel Technology Journal Q1, 2001. http://www.intel.com/technology/itj/

[10] S.Thakkar and T.Huff, "The Internet Streaming SIMD Extensions", Intel Technology Journal Q2, 1999. http://www.intel.com/technology/itj/

[11] J.H.Wolf III, "Programming Methods for the Pentium III Processor's Streaming SIMD Extensions Using the VTune Performance Enhanacement Environement", Intel Technology Journal Q2, 1999. http://www.intel.com/technology/itj/

[12] A.Bik and al., "Efficient Exploitation of Parallelism on Pentium III and Pentium 4 Processor-Based Systems", Intel Technology Journal Q1, 2001. http://www.intel.com/technology/itj/

[13] R. Allen and K. Kennedy ," Optimizing Compilers for Modern Architectures: A Dependence-based Approach", Morgan Kaufman Pub., Oct. 2001

[14] Vtune User Manual http://www.intel.com

[15] SPEC CPU 2000 http://www.spec.org

[16] J.Hennessy and D.Patterson, "Computer Organization & Design- The HardwareSofwtare Interface", Morgan Kaufman Pub., 1998.

[17] J.L.Henning, "SPEC CPU 2000: Measuring CPU Performance in the New Millennium", IEEE Computer, pp.28-25, July 2000. http://www.research.ibm.com/journal/rd41-3.html

[18] R.Cooksey and D.Grunwald," Characterization of the SPEC2000 Benchmark Suite", HPCA-7.

[19] R.Blake and J.S.Breese, "Automatic Bottleneck Detection", Microsoft Research , MSR-TR-95-10, 1995.

[20] D.Kaeli, "Parameter Value Characterization of Windows-NT based applications", Workload Characterization: Methodology and Case Studies, IEEE Computer Society, pp.142-149, 1999.

[21] D. Lee, P. Crowley, J. Baer, T.Anderson, B.Bershad, "Execution Characteristics of Desktop Applications on Windows NT", 25th Annual International Symposium on Computer Architecture (ISCA), June 1998.

[22] P.G.Emma, "Understanding some simple processor-performance limits", IBM Journal of Research and Development, PP.215-232, vol.41, No.3, 1997. http://www.research.ibm.com/journal/rd41-3.html

[23] C.Basoglu, W.Lee and J.S.O'Donnell, "The MAP1000A VLIW Mediaprocessor", pp.48-59, IEEE Micro, Vol.20, No.2, March/April 2000.

[24] K.Diefendorff, P.K.Dubey, R.Hochsprung and H.Scales, "Altivec Extension to PowerPC Accelerates Media Processing", pp.85-95, IEEE Micro, Vol.20, No.2, March/April 2000.

[25] J.Ahn and W.sung, "Multimedia Processor-Based Implementation of an Error Diffusion Halftoning Algorithm Exploiting Subword Parallelism", pp.129-138, IEEE Trans. on Circuits and Systems for Video Technology, Vol.11, No.2, Feb. 2001.

[26] C.Yoon, R.Woo, J.Kook and al., "An 80/20-Mhz 160-mW Multimedia Processor Integrated With Embedded DRAM, MPEG-4 Accelerator, and 3D Rendering Engine for Mobile Applications", pp.1758-1767, Vol.36, No.11, IEEE Journal of Solid-State Circuits, Nov.2001.

[27] T.Koyama, K.Inoue, H.Hanaki, M.Yasue and E.Iwata, "A 250-Mhz Single-Chip Multiprocessor for Audio and Video Signal Processing", pp. 1768-1774, Vol.36, No.11, IEEE Journal of Solid-State Circuits, Nov.2001.

[28] H.Okano and al.,"An 8-way VLIW Embedded Multimedia Processor Built in a 7-layer Metal 0.11 □m CMOS Technology", IEEE International Solid-state Circuits Conference, San Francisco, Feb.2002.

[29] J.smith and A.Dhodapkar, 3Dynamic Microarchitecture Adaptation via Co-Designed Virtual Machines", IEEE International Solid-state Circuits Conference, San Francisco, Feb.2002.

[30] H.Chang,"Surviving the SOC Revolution – A Guide to Platform-Based Design", Kluwer Academic Pub., 1999.

[31] R.Seepold and al., "Virtual Components Design and Reuse", Kluwer Academic Pub., 2001.

[32] S.Swan, "An Introduction to System Level Modeling in SystemC 2.0", May 2001.

[33] SystemC ver.2.0., User's Guide, Oct.2001 http:///www.systemc.org

[34] InSilicon : http://www.insilicon.com/products/ProductPages/JPEG/jpeg_2k_codec.shtml

[35] InSilicon : http://www.insilicon.com/products/ProductPages/JPEG/jpeg2000.shtml

[36] CAST Inc : http://www.cast-inc.com/cores/rc_2ddwt/rc_2ddwt.pdf

[37] http://www.advanced-ip.jp/aipproduct/core/datasheets/rc_2ddwt.pdf

[38] Amphion : http://www.amphion.com/cs6510.html

[39] JPEG2000 Co-processor ADV-JP2000, Preliminary Technical Data, Analog Devices 2001.

[40] Celoxica www.celoxica.com

[41] D.Taubman and M.W.Marcellin, "JPEG2000 – Image Compression Fundamentals, Standards and Practice", Kluwer Academic Publishers, Nov. 2001.

[42] Hyper-Threading technology : http://www.intel.com/technology/hyperthread/index.html

[43] IPP 3.0 : http://www.intel.com/software/products/ipp/ipp30

[44] http://www.semiconductors.philips.com/

[45] J.Ahn and W.sung, "Multimedia Processor-Based Implementation of an Error Diffusion Halftoning Algorithm Exploiting Subword Parallelism", pp.129-138, IEEE Trans. on Circuits and Systems for Video Technology, Vol.11, No.2, Feb.2001.

[46] G. Slavenburg, "TM1000 Preliminary Data Book", March 24, 1997 http://www.semiconductors.philips.com/

[47] G. Slavenburg, "TM1100 Preliminary Data Book", January 13, 1999 http://www.semiconductors.philips.com/

[48] G. Slavenburg, "TM-1300 Media Processor Data Book", May 30, 2000 http://www.semiconductors.philips.com/

[49] "TMS320C6000 Optimizing Compiler User's Guide", Literature Number: SPRU187I, p. 1-1 – 6-9, April 2001 http://www.ti.com/

[50] "TMS320C6000 CPU and Instruction Set Reference Guide", Literature Number: SPRU189F, 2-1 – 2-12, October 2000 http://www.ti.com/

[51] TMS320C6711/11B/11C/11D Floating-Point Digital Signal Processors (Rev. L) 31 May 2004 http://www.ti.com/

[52] TMS320C6000 Code Composer Studio Tutorial Feb. 2000 http://www.ti.com/

[53] Glenn D. Hines, Zia-ur Rahman, Daniel J. Jobson, Glenn A. Woodell and Steven D. Harrah, "Real-Time Enhanced Vision System", SPIE Defense & Security Symposium 2005, Orlando, Florida, March 28 - April 1, 2005

[54] Glenn Hines, Zia-ur Rahman, Daniel Jobson, Glenn Woodell, "DSP Implementation of the Retinex Image Enhancement Algorithm", SPIE Defense & Security Symposium 2004, Orlando, Florida, April 12-16, 2004

[55] Trestino Cosmo, Alessandro Maso, Gian Antonio Mian, "Real Time Implementation of the HVXC MPEG-4 Speech Coder", 5th international conference on digital audio effects DAFx-02 Hamburg, Germany, september 26-28, 2002

[56] Stefano Gnavi, Barbara Penna, Marco Grangetto, Enrico Magli, Gabriella Olmo, "Wavelet Kernels on a DSP: a Comparaison between Lifting and Filter Banks for Image Coding", EURASIP Journal on Applied Signal Processing, Special issue on implementation of DSP and communication systems, Vol.9 pp 981-989 sept. 2002

[57] Pedro O. Jara, Ivan Garcia, and Bryan Usevitch, "Analysis and Optimization of JPEG2000 in the TMS320C6701", Global Signal Processing Expo GSPx Dallas, TX, March 31-April 3, 2003.

Chapitre V Conception du codeur entropique pour circuits FPGA

1. Introduction

L'étude de la norme a démontré que les exigences relatives à JPEG2000 peuvent être très variées suivant les applications considérées. Les nombreux groupes de travail (WG) de la norme placent JPEG2000 dans diverses conditions de fonctionnement. Cela rend nécessaire de considérer un large éventail d'utilisations dans des environnements très distincts dans leurs contraintes en temps de réponse, en consommation d'énergie, en espace utilisé et bien entendu en coût. Dans une logique d'amélioration sur l'existant nous n'avons pas trouvé de détails suffisamment significatifs que nous pourrions considérer comme utiles pour nos travaux. L'architecture optimale et les modes de fonctionnement originaux restent ouverts à la recherche. Pour surpasser ce manque d'information nous avons essayé jusqu'à ce chapitre et le long des chapitres précédents de regrouper l'ensemble des informations nécessaires pour établir un plan d'attaque pour l'optimisation de JPEG2000. Dans ces chapitres on a réalisé : (1) l'étude de la norme JPEG2000, (2) l'étude de l'état de l'art sur les implémentations matérielles et logicielles de JPEG2000, (3) la présentation des méthodologies de conception des systèmes sur puces et sur puces programmables, et (4) l'analyse des limites des implémentations logicielles sur plate-forme commerciales avec une première approche pour l'optimisation de JPEG2000 pour ce type de plateforme.

Dans ce chapitre qui est la succession logique des chapitres précédents on va présenter une première approche pour l'optimisation de JPEG2000. Cette approche se résume dans l'utilisation d'une IP d'un codeur Entropique comme coprocesseur pour l'amélioration des performances de JPEG2000. Cette approche bénéficie de la possibilité de l'indépendance du traitement des différents code-blocks, cela nous permettra de n'utiliser plus qu'une seule instanciation de l'IP du codeur entropique.

L'utilisation d'une implémentation du codeur entropique comme coprocesseur a été réalisée sur deux plateformes différentes : Une première implémentation a été réalisée sur un composant de la famille Virtex de chez Xilinx sur une carte RC1000PP de chez Celoxica. La deuxième plateforme est une carte SBC405GP de chez Wind River. Cette dernière dispose du microprocesseur PowerPC405 d'IBM **[1]** et du composant FPGA VirtexE le XCV405E **[6]**.

2. Les architectures proposées dans la litterature

Plusieurs études proposent des implémentations HW de la partie I du standard JPEG2000 **[10]**, **[13]**, **[14]** ou du Tier-1 de EBCOT **[11]**, **[12]**, **[15]**, **[23]**. Ces architectures concernent aussi bien le codage que le décodage. Généralement lorsqu'on dit implémentation JPEG2000 on s'intéresse plus à la partie DWT **[28-34]** et au Tier-1 de EBCOT. Les architectures proposées ont été conçues pour résoudre les problèmes les plus critiques de JPEG2000 tels que la gestion de la mémoire, l'accélération du codage. Le nombre des travaux concernant le Tier-1 de EBCOT prouvent que la majorité des problèmes de gestion de mémoire et de codage sont concentrés dans cette partie du codeur. Les architectures proposées traitent un ou plusieurs problèmes à la fois et utilisent différentes techniques d'optimisation telles que de nouvelles organisations de la mémoire, le codage sélectif, la parallélisation du codage. Ces architectures visent plusieurs technologies, certaines ont été implémentées sur des FPGA, d'autres sur des circuits dédiés, et d'autres architectures ont été présentées au stade de la simulation.

2.1 Les problèmes traités

Les problèmes que pose généralement JPEG2000 et particulièrement le Tier-1 de EBCOT et la DWT sont :

(1) Taille de la mémoire utilisée pour effectuer le codage d'une image ou d'une partie de cette image. Pour mieux gérer ce problème les concepteurs se limitent généralement à une partie de l'image (Tile) pour pouvoir limiter l'utilisation de la mémoire sur puce qui offre une meilleure vitesse d'accès mais très coûteuse à implémenter. L'architecture de **[10]** permet de réduire de 20% la taille de la mémoire utilisée par EBCOT. **[20]** présente une réduction par un facteur 7 de la taille de la mémoire nécessaire pour le codeur Entropique. Ce problème concerne aussi la DWT.

(2) le nombre des Accès mémoire influence considérablement sur le temps du traitement. Pour cette raison plusieurs travaux ont proposé des architectures qui permettent de réduire les accès mémoire. Plusieurs travaux ont pris en charge ce problème. Dans **[21]** on réduit le transfert de données aussi bien pour la DWT que pour EBCOT, alors que dans **[23]** seul EBCOT est étudié.

(3) un traitement séquentiel du à la nature de l'algorithme qui nécessite trois passages de codage successif, et que le traitement s'effectue au niveau bit, bit-plane par bit-plane. Dans **[15]**, **[23]**, **[26]** différentes architectures ont été proposées pour la parallélisation du codage.

Concernant la transformée en ondelettes, la structure lifting-scheme utilisée dans JPEG2000 est la plus adaptée pour une implémentation VLSI. Elle permet de réduire considérablement l'utilisation des blocs logiques et de la mémoire. Le tableau suivant **[30]** donne une comparaison de l'utilisation des ressources pour les structures : lifting-scheme et banc de filtres.

Tableau Ch V - 1 : Lifting-scheme vs banc de filtres de l'algorithme pyramidale de Mallat

	Lifting-scheme	Banc de filtres
# additionneurs (1D)	4	6
# additionneurs (2D)	12	18
Mémoire (décomposition)	<1 Ko	84 Ko
Mémoire (recomposition)	<1 Ko	112 Ko

L'implémentation de la DWT multirésolution présente un compromis majeur, celui de l'utilisation de la mémoire et des ressources logiques. En effet une décomposition en plusieurs niveaux nécessite, soit l'utilisation d'une mémoire d'une taille suffisante pour le stockage des données nécessaires au traitement et d'un seul bloc de traitement, soit l'utilisation d'autant de blocs de traitement que niveau de décomposition et d'une mémoire beaucoup plus petite que précédemment. La majorité des implémentations proposées dans la littérature cherchent à réaliser le meilleur compromis mémoire/blocs de traitement **[28-34]**.

2.2 Les techniques d'optimisation

Plusieurs techniques d'optimisation ont été mises au point pour améliorer les performances du codeur JPEG2000 dans sa globalité ou en intervenant seulement au niveau du codeur entropique. L'ensemble de ces techniques peut être partagé en deux ensembles. Des techniques d'optimisation au niveau macroscopique, et au niveau microscopique. Au niveau macro les principales techniques se résument dans l'utilisation de plusieurs codeurs entropiques **[13]** voir plusieurs codeurs JPEG2000 **[26]** surtout pour le codage vidéo. L'utilisation du multiprocesseur à mémoire partagée pour l'exécution des implémentations de référence de JPEG2000 a été étudiée dans **[22].**

Au niveau micro différentes techniques sont utilisées pour améliorer les performances de JPEG2000, réduire le nombre d'accès et la taille de la mémoire. Parmi ces techniques on trouve le codage sélectif qui peut s'appliquer à trois niveaux : au niveau échantillon colonne d'un stripe **[11]**, **[12]**, et passage de codage **[14]** (sample skiping, column skiping, pass skiping). Une deuxième technique utilisée dans plusieurs architectures exploite les différents modes

d'interruption de JPEG2000 (CAUSAL, RESTART, RESET) pour pouvoir exploiter le micro parallélisme offert par l'algorithme du Tier-1 de EBCOT. Ainsi il est possible de réaliser les trois passages de codage en parallèle.

Au niveau codeur arithmétique des techniques de multiplexage **[15]** ou de codage de plusieurs symboles en même temps **[27]** permettent l'amélioration du débit de sortie du codeur entropique. En plus de ces techniques plusieurs architectures proposent des organisations particulières de la mémoire utilisée par le odeur entropique **[14]** ou par la DWT **[13], [30]** pour réduire le nombre d'accès. Plusieurs autres astuces permettent de gagner plus ou moins des cycles d'horloge tels que la duplication de registre ou l'utilisation d'une distribution particulière de registre (registre Gobang) **[18].**

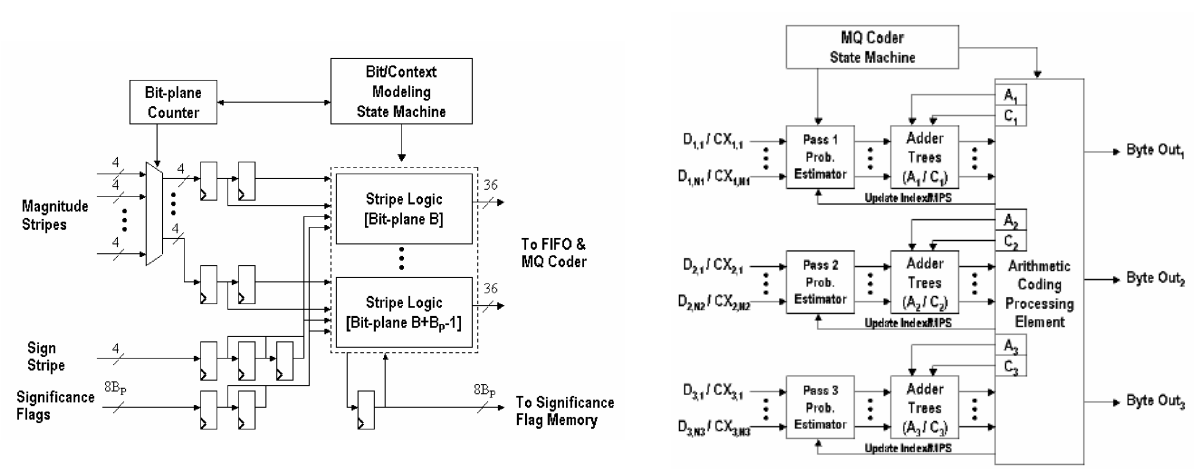
2.3 Les architectures et les technologies cibles

Les architectures proposées dans **[10-27]** ont été conçues et implémentées sur différentes technologies. Certaines architectures sont du type VLSI, d'autres sont des circuits dédiés. Ces derniers sont utilisés comme coprocesseurs implémentés sur des composants FPGA ou sur un ASIC **[24], [25]**. D'autres implémentations sont fournies au niveau langage HDL avec des résultats de simulation et des estimations de la surface et du temps d'exécution.

2.4 Exemples

2.4.1 Architecture

Cette architecture est présentée dans **[23].** Elle exécute en parallèle les trois passages du codage. Une architecture adaptée du codeur arithmétique permet de suivre le débit de sortie du module formation du contexte.

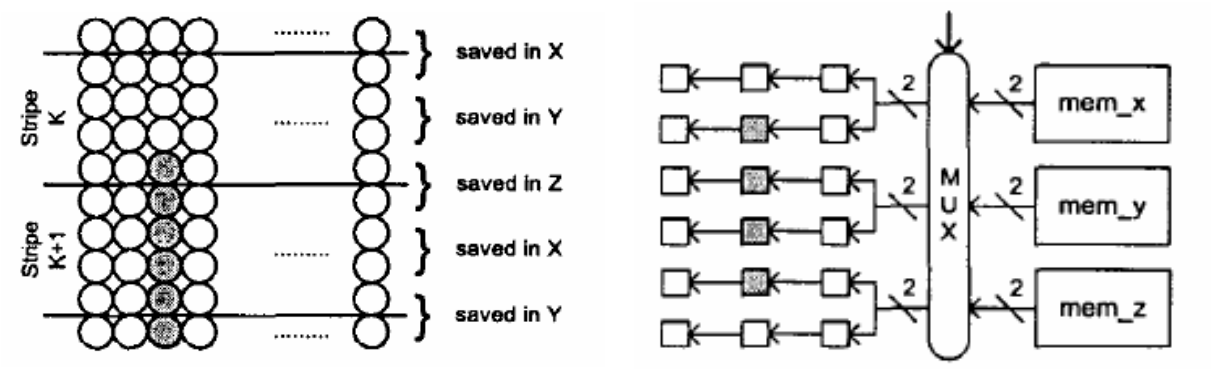


(a) (b)

Figure Ch V - 1 : (a) Formation de contexte, (b) codeur arithmétique

2.4.2 Mémoire

Dans **[19]** on propose une manière d'organiser la mémoire qui permet de faciliter l'accès mémoire d'un codeur entropique (la partie Formation de contexte) sans recourir à l'utilisation des modes d'interruption.



(a) (b)

Figure Ch V - 2 : Organisation de la mémoire (a) découpage des stripes (b) organisation de ces stripes dans des blocs mémoires

2.5 Discussion

Dans cette partie on a exposé plusieurs architectures qui implémentent le codeur JPEG2000 ou seulement le codeur entropique. Ces architectures utilisent plusieurs techniques d'optimisation. Les techniques les plus pertinentes et qui sont réutilisées dans la majorité des conceptions sont : au niveau macro, et l'utilisation de plusieurs codeur/décodeur JPEG2000 ou entropique seulement, en parallèle, et au niveau micro, le codage sélectif, et le passage de codage parallèle.

Dans le cas de l'utilisation de plusieurs codeurs en parallèle le seul problème qui se pose est le coût en surface et en énergie. Car l'utilisation d'un certain nombre de codeurs implique une duplication des ressources autant de fois que nécessaire pour garantir une complète indépendance de ces codeurs d'où une meilleure performance. Le partage de certaines ressources telles que les mémoires peut dégrader considérablement les performances des codeurs en ajoutant des modules de contrôle et d'arbitrage. La duplication des ressources peut être interne à la puce comme externe, ce qui contribue davantage à l'augmentation de la consommation d'énergie. Cette question a été omise par la majorité des papiers **[10-27]**. Aucune étude ne montre l'impact de la multiplication du nombre de codeur sur cette consommation.

Au niveau micro, l'application des passages de codage en parallèle nécessite l'utilisation des modes d'interruption (CAUSAL). Elle impose aussi d'adapter l'architecture du codeur arithmétique qui récupère des symboles des différents passages de codage qu'il doit réordonner. Pour contourner cet ordonnancement l'utilisation des modes « RESTART » et « RESET »est inévitable comme dans **[23]**. Ces modes de codage du standard passent ainsi, du niveau d'option, au niveau d'obligation.

3. Première implémentation

3.1 Environnement

L'environnement de développement utilisé pour cette implémentation est composé de deux parties, la partie SW qui est l'environnement de développement ISE (Integrated Software Environment) de Xilinx, une partie HW qui est la carte RC1000-PP construite autour du composant Virtex XCV1000. La partie SW est complétée par une chaîne de codage JPEG2000 issue du projet EIRE. Cette chaîne de codage nous permettra de faire le test et la validation de l'IP implémenter.

3.1.1 ISE

ISE inclut des éditeurs et outils pour la saisie du système, il dispose aussi d'un ensemble d'outils de synthèse et d'implémentation. Ils sont regroupés dans un seul flot de conception représenté dans la **figure Ch V - 3**. Ces différents outils sont détaillés comme suit :

1-Saisie de conception (Design Entry)

- éditeur HDL
- Editeur de Machie a Etat Fini(StateCAD)
- Editeur de schéma (Engineering Capture System (ECS))
- générateur d'IP (CORE Generator)

2-Synthèse

- XST - Xilinx Synthesis Technology
- Intégration de Leonardo Spectrum de Mentor Graphics.
- Intégration de Synplify de Synplicity.

3-Simulation

- Générateur de TestBench HDL
- Intégration du simulateur ModelSim de Model Technology.

4-Implémentation

- Translate
- MAP
- Placement et routage (PAR)
- Floor planner
- Editeur FPGA
- Timing Analyzer
- XPower
- Fit (seulement CPLD)
- Chipviewer (seulement CPLD)

5-Programmation du composant et formatage du fichier de programmation

- BitGen
- iMPACT

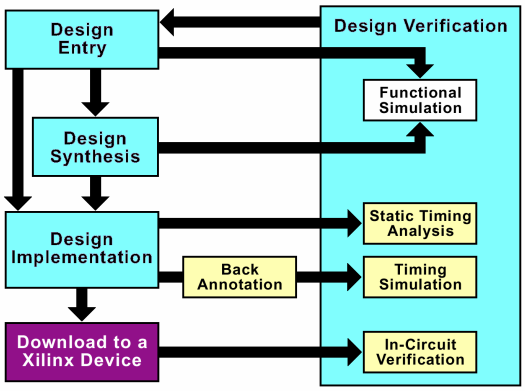


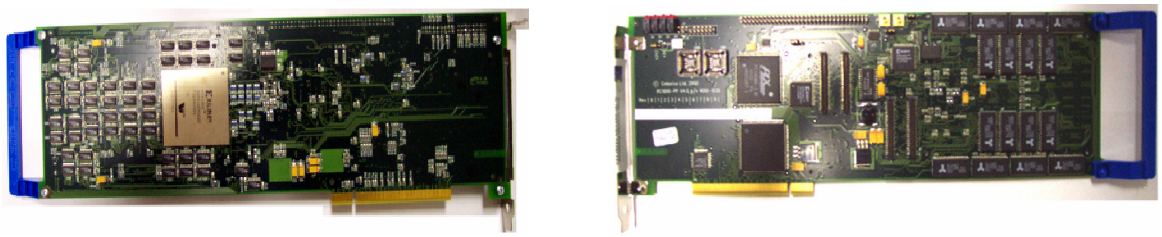
Figure Ch V - 3 : flot de conception ISE

3.1.2 Chaîne de Codage EIRE

La chaîne de codage EIRE est une implémentation C++ du codeur JPEG2000. Elle nous a été fournie sous la forme de librairies. Elle permet de fournir les données de l'ensemble des code-blocks, et de récupérer les bitstreams pour reconstruire le fichier JPEG2000.

3.1.3 Celoxica RC1000-PP

La carte Celoxica RC1000-PP est une carte PCI standard pour PC sur laquelle est implémentée un FPGA, dans le cas présent un XCV1000 de Xilinx. Sur cette dernière on peut connecter jusqu'à 2 cartes PMC supplémentaires. La carte présente aussi 4 bancs de mémoire de 2 Mo chacune. Ces mémoires sont accessibles par le FPGA directement et par le PC via le bus PCI **[8]**.



(a) (b)

Figure Ch V - 4 : (a) Carte Celoxica Face composant FPGA, (b) Carte Celoxica Face Composants PCI/mémoires

Cette carte est utilisée pour la conception de prototype sur FPGA en utilisant un ordinateur pour l'envoi et la récupération des données. Elle présente 3 méthodes différentes pour transférer les données :

- du Bus PCI vers le FPGA via les mémoires
- 2 ports 8 bits unidirectionnels communicant directement entre le Bus PCI et le FPGA
- le PLX User In et le PLX User Out pour générer une communication série

Le FPGA possède deux de ses entrées d'horloges connectées, soit à des générateurs d'horloge, soit à une horloge extérieure ou l'horloge du Bus PCI. Il est donc possible de se synchroniser avec le Bus PCI, sinon les générateurs d'horloges permettent d'obtenir une plage de fréquence allant de 400 kHz à 100 MHz.

Avec la carte on dispose d'une libraire C++ spéciale permettant la configuration de la carte, l'initialisation du FPGA et du générateur d'horloge et le transfert des données **[7]**, **[9]**.

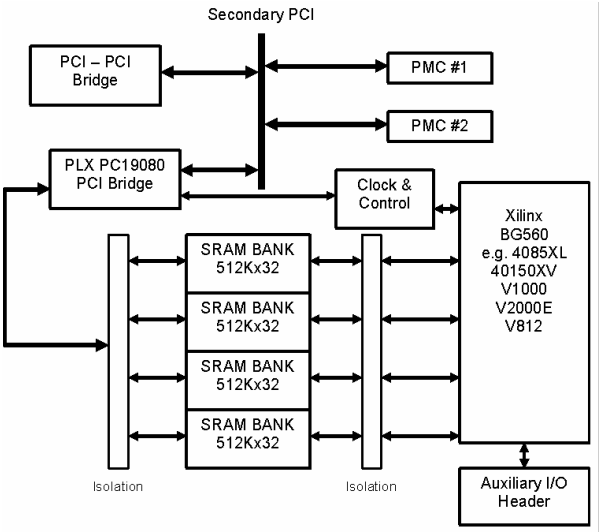


Figure Ch V - 5 : Architecture de la carte RC1000-PP

3.1.4 Caractéristique du XVC1000

Voici deux tableaux récapitulatifs des capacités du XVC1000 et un schéma représentant leur positionnement. Le paquetage utilisé est le BG-560 disposant de 404 I/O utilisateur, avec un speed grade égale a -6 **[6]**.

Tableau Ch V - 2 : Caracteristiques du XCV1000

Caractéristiques	XCV1000
CLB array (RowxCol.)	64x96
Logic Cells	27.648
System Gates	1.124.022
Max. Block RAM Bits	131.072
Max. Distributed RAM Bits	393.216
Delay Locked Loops (DLLs)	4
I/O Standards Supported	17
Speed Grade	4, 5, 6
Max. Avail. User I/O	512

Tableau Ch V - 3 : Performances du Virtex (-6)

Function	**Bits**	**Virtex -6**
Register-to-Register		
Adder	16	5.0 ns
	64	7.2 ns
Pipelined Multiplier	8 x 8	5.1 ns
	16 x 16	6.0 ns
Address Decoder	16	4.4 ns
	64	6.4 ns
16:1 Multiplexer		5.4 ns
Parity Tree	9	4.1 ns
	18	5.0 ns
	36	6.9 ns

Comme les blocs de RAM sont disposés de part et d'autre du composant FPGA, on a adopte la même disposition pour les deux codeurs entropiques implémentés. Cette disposition permet de réduire le temps de propagation entre les différents éléments du codeur et les blocs de RAM. Cette disposition suit aussi celles des entrées/sorties des RAM de la carte Celoxica

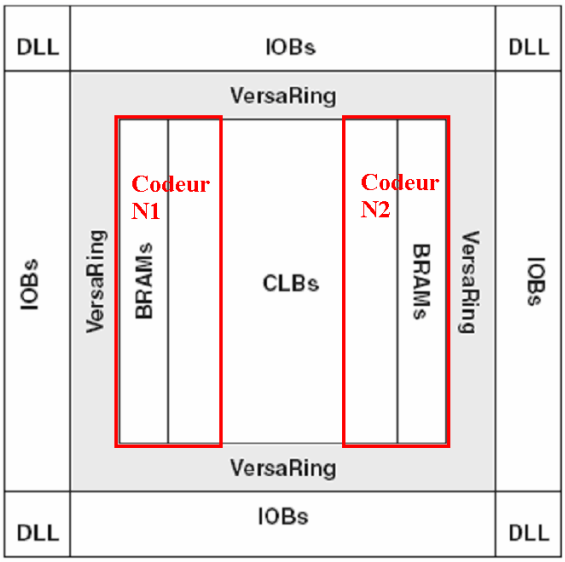


Figure Ch V - 6 : Structure du Virtex

3.2 Architecture

Le codeur entropique a été implémenté sous la forme de trois modules principaux : (1) Un composant pour la formation du contexte qui permet de générer les couples (Contexte, Décision) à partir des coefficients du code-block, (2) Une FIFO : elle permet de séparer le fonctionnement du premier module et le module du traitement arithmétique vu la discordance de leurs débits respectifs, (3) Un module de codage Arithmétique (MQ-coder) : il consomme les couples (Contexte, Décision) stockés dans la FIFO pour générer les données compressées dites bitstream.

En plus des ces modules deux modules d'interfaçage avec les mémoires externes on été prévus pour la lecture des coefficients des code-blocks, et le stockage des bitstreams correspondants.

Comme sur la carte Celoxica on dispose de 4 bancs de mémoire accessibles de manière indépendante, donc on a opté pour une implémentation de deux codeurs en parallèle comme le montre la figure suivante. L'utilisation des mémoires indépendantes nous permet d'éviter les conflits d'accès et le recourt à un arbitrage, ce qui ne peut que dégrader les performances du codage.

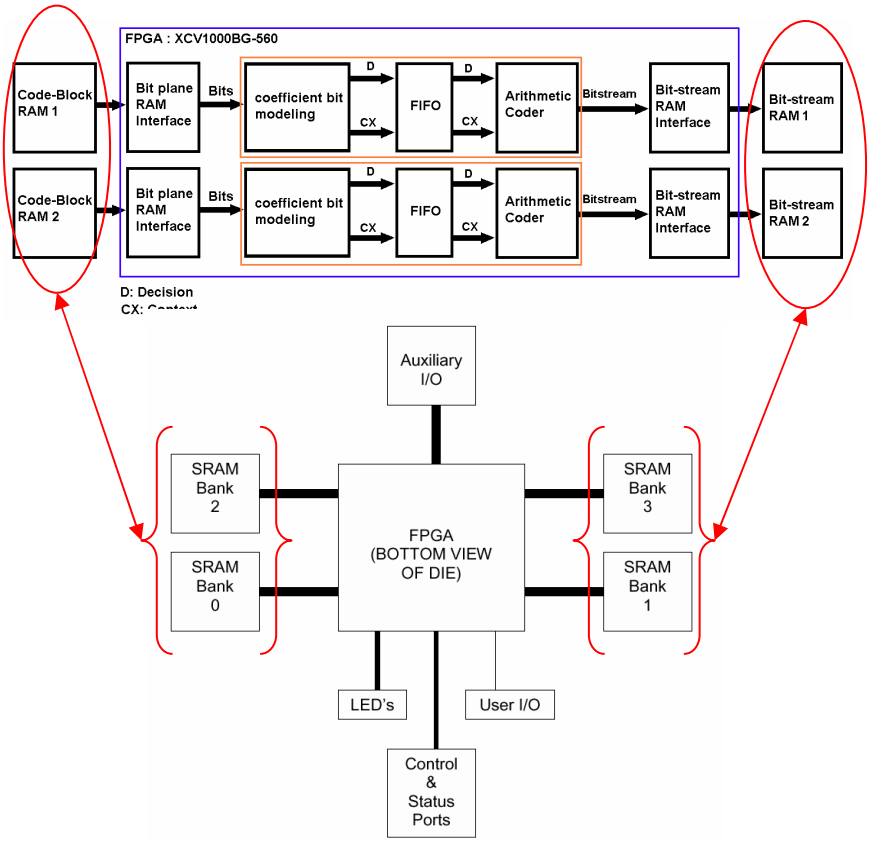


Figure Ch V - 7 : Architecture du bi-codeur entropique

3.3 Test et validation

Le test et la validation des deux codeurs implémentés sur le Virtex XCV1000 ont été réalisés en intégrant ces codeurs dans la chaîne de codage EIRE. L'implémentation matérielle des codeurs entropiques est venue remplacer l'implémentation SW de la chaîne d'origine comme le montre la figure suivante. Cette intégration a nécessité l'ajout d'une fonction SW qui utilise les librairies C fournies avec la carte pour assurer le transfert des données de et vers le composant FPGA. Le diagramme de la **figure Ch V - 9** explique le mode de fonctionnement de la chaîne mixte complète.

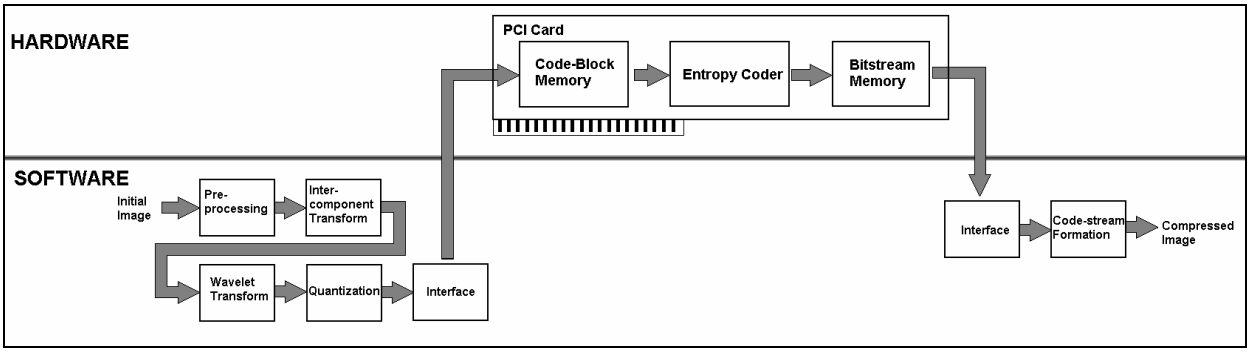


Figure Ch V - 8 : Intégration du codeur entropique dans la chaîne EIRE

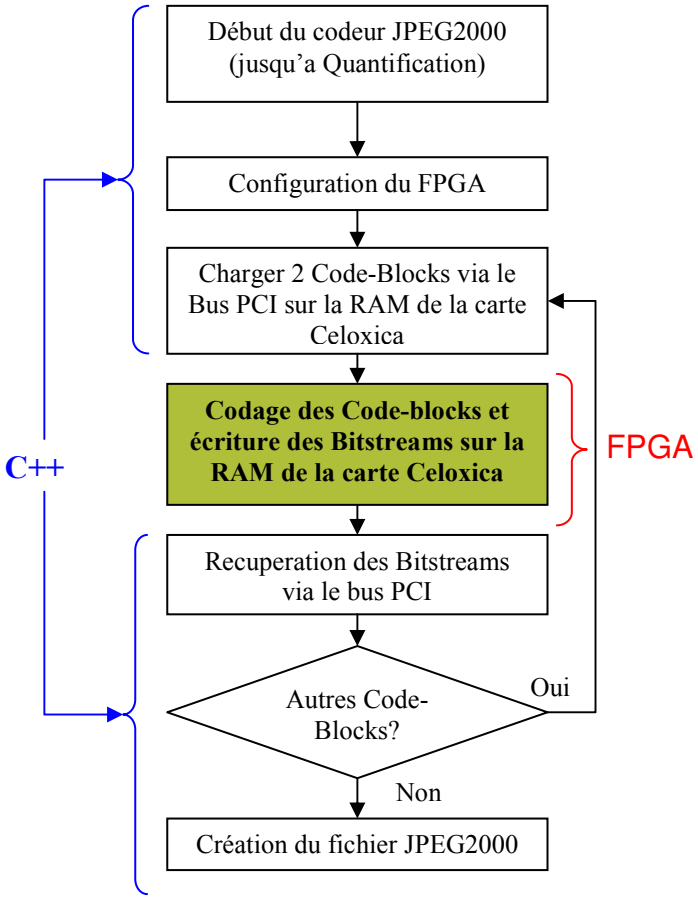


Figure Ch V - 9 : Diagramme de fonctionnement de la chaîne mixte SW/HW

3.4 Résultats

3.4.1 Performances

Par rapport au temps final d'exécution, la solution sur FPGA semble loin derrière la référence de la chaîne EIRE. Cependant il est nécessaire de décomposer ce temps pour en apercevoir les possibilités futures.

Le déroulement du codage des différents code-blocks se décompose comme suit :

1. Configuration du FPGA et initialisation
2. Envoi dans la mémoire de chaque codeur d'un code-block en entier
3. Activation des 2 codeurs

4. Attente de la fin de la part des 2 codeurs
5. Récupération des 2 résultats
6. Activation du Reset
7. Envoi dans la mémoire de chaque codeur d'un code-block suivant
8. boucler sur 4 jusqu'à la fin de tous les code-blocks

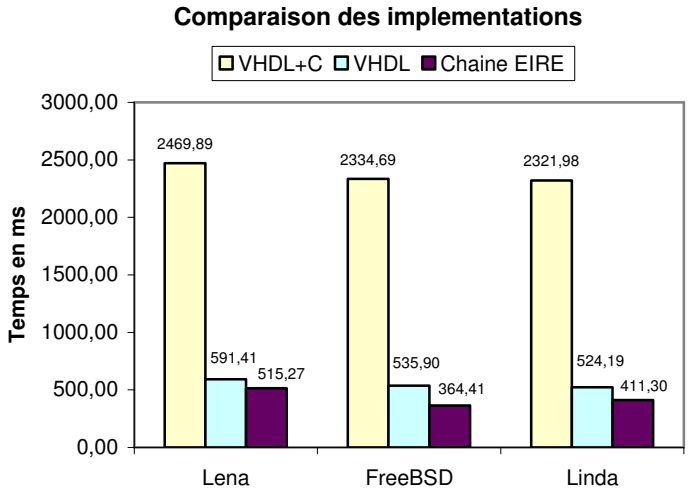


Figure Ch V - 10 : Comparaison des temps d'exécution des versions SW et HW du codeur entropique

VHDL & C++ : représente le temps de préparation des données, transfert de données, traitements des données et récupération des données

VHDL : représente le temps de traitements des données seul

Chaîne EIRE : représente l'équivalent de VHDL & C++ mais entièrement en logiciel sur le Pentium4

L'écriture des code-blocks et la récupération des résultats via le bus PCI sont grandement consommateurs de temps. En effet ils représentent ici 1964 ms soit plus de 79% du temps total. Ce temps énorme est dû au fait que l'on doit entièrement copier les code-blocks avant de lancer le calcul et que la récupération ne puisse se faire qu'après la fin du calcul. De plus il faut attendre la fin de la récupération des résultats avant d'envoyer le code-block suivant.

Afin d'améliorer les temps de transfert, il serait utile de pouvoir charger plusieurs code-blocks dans la mémoire d'un codeur et que ce dernier puisse les traiter à la chaîne (pipeline de code-blocks).

En comparant les temps de codage pur (sans les transferts de données) il apparaît donc possible d'avoisiner les temps de calcul du Pentium en utilisant 4 codeurs en parallèle. A condition d'utiliser une plateforme sur laquelle le transfert des données soit optimal.

Les mesures de temps de calcul ont été effectuées principalement sur trois images de tailles égales à 512x512 pixels chacune. L'implémentation parallèle sur cette carte n'a pas été poussée plus en avant au vu des problèmes de communication via le bus PCI, ne permettant pas une utilisation optimisée.

3.4.2 Résultats de l'implémentation

La synthèse, le placement et le routage ont été effectués en optimisant la fréquence de la conception. La simulation de consommation a été faite pour une température ambiante de 25°C avec des alimentations de 2.5V et 3.3V.

Tableau Ch V - 4 : Occupation des ressources

XCV1000	
occupation des slices	2088/12288➔17%
occupation des Block RAM (1 blocks=4096 bits)	28/32➔87%
Fréquence	45 MHz
Consommation d'énergie	542 mW @ 30 MHz

Ces résultats sont, pour les taux d'occupation, tirés des rapports de placement/routage et, pour la fréquence, la plus haute avec laquelle le codeur a fonctionné. Les rapports indiquaient une fréquence maximum de 50 MHz.

Dans cet exemple, nous pouvons voir que ce FPGA n'est pas spécialement bien adapté, car seulement 17% des slices sont utilisés. Un FPGA de la famille des « Extended Memory » conviendrait mieux pour son utilisation, notamment le XCV2000E, supporté par la carte Celoxica. Ainsi il serait possible, en s'intéressant à la communication Host-FPGA et en utilisant une seule mémoire externe par codeur, d'arriver à des résultats intéressants, mais toujours limités par la vitesse du bus PCI. Dans notre cas, les longueurs dues au transfert via le bus PCI et le traitement non continu des code-blocks n'ont pas été optimisées. Seul sont intéressants les gains au niveau du FPGA car ils sont la partie réutilisable de la conception totale et peuvent être transposables sur d'autres plateformes.

4. Deuxième implémentation

4.1 Les outils de développement de Wind River

4.1.1 L'environnement de développement Tornado

La suite d'outils de Tornado fournit un environnement visuel et automatisé pour le développement des applications basées VxWorks. VxWorks est le composant d'exécution (runtime) de la plateforme Tornado. C'est un système d'exploitation temps réel (RTOS) qui est largement répandu dans l'industrie. Dans Tornado nous pouvons charger de manière incrémentale des modules dans un système cible. Tornado supporte le langue C et C++ avec des compilateurs qui fournissent plusieurs optimisations. Il intègre également un simulateur (VxSim Lite) qui utilise les outils Wind View pour le diagnostic et l'analyse au niveau système. Ces outils permettent de fournir une visibilité détaillée du comportement dynamique des applications VxWorks embarquées fonctionnant sur le simulateur.

Sur la figure suivante on distingue les différents composants de l'environnement Tornado avec ceux de VxWorks **[2]**.

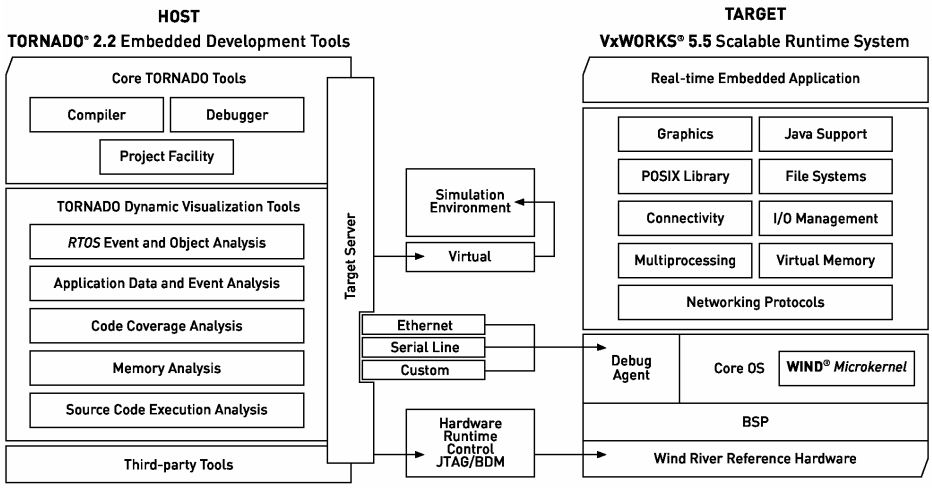


Figure Ch V - 11 : Environnement de développement Tornado

4.1.2 Plateforme de développement HW

Le PowerPC 405GP par son bon équilibre entre les performances et la consommation d'énergie (1,5W @ 200 mégahertz) est bien adapté pour les applications embarquées. Dans **[4]**, **[5]** nous pouvons trouver une confirmation de la large utilisation du PowerPC dans les applications industrielles embarquées. Mais dans le cas de l'algorithme complexe comme JPEG2000 les performances se trouvent énormément affectées, ainsi le coprocessing s'avère nécessaire. Le Coprocessing ne devrait pas corrompre l'équilibre entre les performances et la consommation d'énergie, c'est pourquoi le FPGA est une bonne alternative pour surmonter un tel problème. Nous choisissons le SBC405GP parce que c'est une plateforme industrielle qui utilise le PowerPC, et les composants FPGA.

a - Carte mère SBC405GP

La référence de conception de Wind River SBC405GP une carte indépendante. Elle utilise un PowerPC 405BG d'IBM fonctionnant jusqu'aux 266MHz **[1]**. Des connecteurs industriels standard permettent d'accéder à toutes les voies de communication du processeur. La carte SBC405GP convient au développement SW et/ou le prototypage de HW. En plus du PowerPC cette carte contient 64 Moctets SDRAM et 16 Moctets de flash. La connexion au host est assurée par deux ports de communication RS232 et une connexion Ethernet 10/100BaseT. Les connexions PCI, PMC, et JTAG sont également disponibles. Les figures suivantes sont : le schéma bloc de la carte, et sa photo **[3]**.

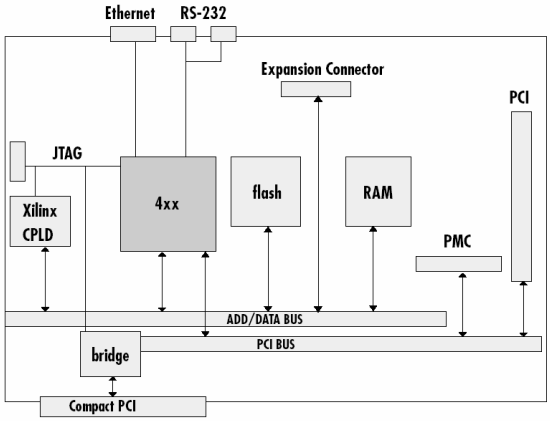


Figure Ch V - 12 : Schéma bloc de la SBC405GP

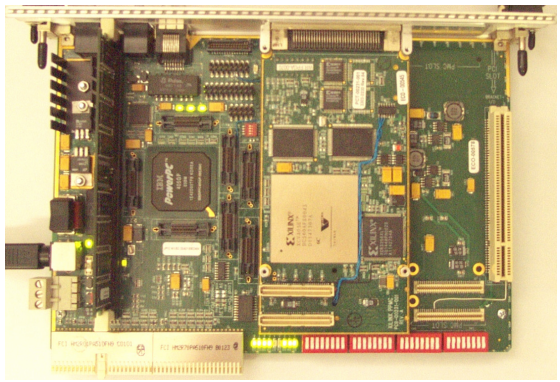


Figure Ch V - 13 : photo de la carte

b - Carte FPGA PMC

La carte FPGA PMC est une référence de conception Wind River de coprocessing elle est utilisée pour l'évaluation et le prototypage des applications reconfigurables. Elle est construite autour du composant XCV405E de Xilinx **[6]** qui est étroitement couplé au microprocesseur PowerPC par l'intermédiaire d'une interface de bus locale PPC de 32 bits de Wind River pouvant fonctionner à une fréquence allant jusqu'à 66 MHz. Le PMC est branché sur la carte mère par le biais d'une connexion PMC, 32 bits, 33MHz. Une SSRAM 4Mo, et deux bancs d'EEPROM (XC18V04) sont disponibles sur la carte **[3]**.

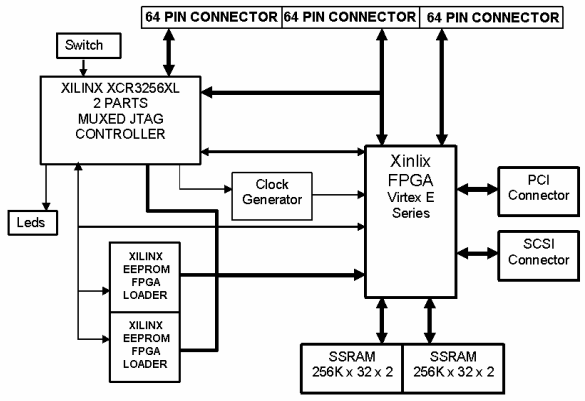


Figure Ch V - 14 : Schéma bloc de la carte FPGA PMC

4.2 Architecture et implémentation sur la SBC405GP

L'implémentation du codeur réalisée sur la carte PMC est une reprise de l'architecture de base du codeur déjà utilisée pour la carte Celoxica, mais pour des raisons d'adaptation avec l'architecture globale de la carte certaines modifications ont étaient apportées à la partie interface des codeurs Entropiques. Ces modifications permettent de couvrir certains manques dus à l'architecture de la carte Celoxica. Avec l'utilisation des BRAM Dual-Port, le PowerPC peut continuer à charger les Code-blocks pendant que le codeur effectue le traitement des code-blocks déjà chargés. De même pendant la récupération des Bitstreams des code-blocks traités, le PowerPC peut lire les bitstreams disponibles pendant que le codeur écrit le bitstream du code-block en traitement. Ces BRAM sont représentées sur la **figure Ch V -15** par Mem 1 et Mem 2. Un module de contrôle a été prévu pour informer le PowerPC de l'état du traitement.

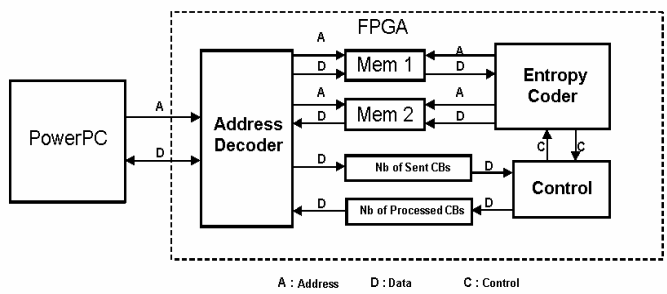


Figure Ch V - 15 : Architecture du codeur Entropique

4.3 Résultats

4.3.1 Ressources

Le composant XCV405E permet d'implémenter au maximum deux codeurs entropiques. Les résultats d'occupation sont représentés dans le tableau suivant. La fréquence de fonctionnement est limitée à celle disponible sur la carte qui est de 33 MHz.

Tableau Ch V - 5 : Resources

Number of Slices	47 %, 2268 out of 4800
Number of BRAMs	87 %, 122 out of 144
Frequency of the PowerPC-BRAMs connection	50 MHz
Frequency of the coders	46 MHz

4.3.2 Performances

Le tableau ci-dessous résume les performances de la chaîne EIRE exécutée sur le PowerPC, et de la chaîne mixte SW/HW du codeur JPEG2000 que nous proposons. L'unité de mesure dans ce cas est le tick qui est généré par le Timer interne du PowerPC. Un tick correspond à 16 ms.

Les mesures on été réalisées sur l'image Lena.ppm (512x512 pixels). On remarque qu'au début il y a plus de ticks pour la chaîne mixte que pour la chaîne EIRE. Ces ticks sont dus à la configuration du FPGA (environ 240 ms). Le speed up obtenu entre les deux implémentations du codeur entropique est d'environ 4,25.

Tableau Ch V - 6 : Performances

	Chaîne EIRE du Codeur JPEG2000	Chaîne du Codeur JPEG2000 SW/HW
Début	12	27
Initialisation	32	32
Charger une image	99	99
DWT+EC	195	82
Ecrire le Fichier JPEG2000	41	41
Total en ticks	394	281
Total en seconde	6,566404	4,683146
% EC / total codeur	76,41%	42,68%
Détails du codeur entropique pour chaque composant de l'image		
Compo1	49	11
Compo2	50	12
Compo3	50	12
Total en tick	149	35
Total en seconde	2,483234	0,58331

La consommation d'énergie est comme suite:

Tableau Ch V - 7 : Consommation d'energie

PowerPC	XCV405E
1,5 W @200MHz	0,95 W @ 33MHz

La consommation d'énergie du PowerPC est fournie par le constructeur **[1]**. Celle du XCV405E a été mesurée à l'aide de l'outil XPower de ISE.

5. Conclusion

Dans ce chapitre on présente l'implémentation d'une IP du codeur entropique sur une première plateforme du type PCI, et la réutilisation de cette IP pour une deuxième implémentation sur une deuxième plateforme industrielle du type standalone.

Pour la première plateforme on a pu approcher les performances d'un Pentium 4 @2GHz (P4) (90 % du temps d'exécution sur P4) avec une consommation d'énergie négligeable par rapport au P4 (75,3W@2GHz ➔ un gain>99%). Cette première implémentation nous permet de découvrir les contraintes imposées par la communication et par l'architecture de la carte ne permettant pas d'obtenir de meilleures performances. Dans le cas de cette plateforme le coprocessing n'est pas mis en valeurs si on le compare aux performances du P4. L'implémentation réalisée sur la deuxième plateforme montre bien l'importance du co-

processing vu le gain en temps obtenu. Le chargement de plusieurs code-blocks par le PowerPC sur les BRAM du FPGA a permis de réduire considérablement les latences du transfert des données. Ceci nous a permis de maximiser le gain en temps. La SBC405 est différente de la première plateforme du fait qu'elle soit indépendante et possède son propre processeur avec le système d'exploitation correspondant. De plus le type de communication entre le FPGA et le PowerPC permet l'accès direct aux ressources du FPGA sans passer par un intermédiaire comme dans le cas de la carte Celoxica (bus PCI). La réutilisation de L'IP du codeur entropique a permis un gain de temps énorme pour le passage entre les deux plateformes. Cette implémentation nous prépare à aborder le sujet des systèmes sur puce et sur puce programmable. En effet il existe un fort potentiel d'atteindre de meilleures performances en ayant le processeur et le coprocesseur à proximité l'un et l'autre avec un partage de ressources comme les mémoires.

6. Références

[1] http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/6A2B659677FC3F9287256B7900701F5E/$file/405GP-405GPr_pb.PDF

[2] http://www.windriver.com/products/development_tools/ide/tornado2/

[3] http://www.windriver.com/products/development_tools/development_boards/sbc405gp/

[4] http://www.thales-computers.com/sbccpu.asp

[5] B.Blander, D.Czenkusch, R.Devins, R.Stever, "An embedded PowerPC™ SOC for test and measurement applications" Microelectronics Div., IBM Corp., Essex Junction; ASIC/SOC Conference. Proceedings. 13th Annual IEEE,VT, USA, 2000

[6] http://www.xilinx.com

[7] RC1000-PP Function Reference Manual, version 1.22, http://www.celoxica.com

[8] RC1000-PP Hardware Reference Manual, version 2.22, http://www.celoxica.com

[9] RC1000-PP Software User Guide, version 1.20, http://www.celoxica.com

[10] Kishore Andra, Chaitali Chakrabarti, Tinku Acharya, "A High-Performance JPEG2000 Architecture", IEEE Transactions on Circuits and Systems for Video Technology, pages:209- 218, Volume: 13, Issue: 3, Mar 2003.

[11] Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, Liang-Gee Chen, "Analysis and Architecture Design of Block Coding Engine for EBCOT in JPEG2000", IEEE Transactions on Circuits and Systems for Video Technology, pages: 219-230, Volume: 13, Issue: 3, Mar 2003.

[12] Yijum Li, Ramy E.Aly, Beth Wilson, Magdy A. Batoumi, "Analysis and Enhancement for EBCOT in High Speed JPEG2000 Architectures", Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on, Volume: 2, Pages:II-207 - II-210 vol.2, 4-7 Aug. 2002.

[13] Paul Schumacher, Marrk Paluszkiewicz, Robert Turney, "Analysis of a JPEG2000 Encoder Implemented on a Platform FPGA", Global Signal Processing Expo 2004:, Santa Clara, CA, September 27-30, 2004.

[14] Hong-Hui Chen, Chung-Jr Lian, Te-Hao Chang, Liang-Gee Chen, "Analysis of EBCOT Decoding Algorithm and its VLSI Implementation for JPEG2000", ISCAS 2002 IEEE International Symposium on Circuits and Systems, Scottsdale Princess Resort Scottsdale, Arizona, May 26-29, 2002.

[15] Jen Shiun Chiang, Yu-Sen Lin, Chang-Yo Hsieh, "Efficient Pass-Parallel Architecture for EBCOT in JPEG2000", ISCAS 2002 IEEE International Symposium on Circuits and Systems, Scottsdale Princess Resort Scottsdale, Arizona, May 26-29, 2002.

[16] Kishore Andra, Tinku Acharya, Chaitali Chakrabarti, "Efficient VLSI Implementation of Bitplane Coder of JPEG2000", Proc. of SPIE Applications of Digital Image Processing, 2001.

[17] Hiroshi Tsutui, Takahiko Masuzaki, Tomonori Izumi, Takao Onoye, Yukihiro Nakamura, "High Speed JPEG2000 Encoder by Configurable Processor", Asia-Pacific Conference on Circuits and Systems, 2002. APCCAS '02, Page(s): 45- 50 vol.1, 2002.

[18] Hung-Chi Fang, Tu-Chih Wang, Chung-Jr Lian, Te-Hao Chang, Liang-Gee Chen, "High Speed Memory Efficient EBCOT Architecture for JPEG2000", Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03, Volume: 2, Pages:II-736-II-739 vol.2, 25-28 May 2003.

[19] Yun-Tai Hsiao, Hung-Der Lin, Kun-Bin Lee, Chein-Wei Jen, "High Speed Memory Saving Architecture for the Embedded Block Coding in JPEG 2000", IEEE International Symposium on Circuits and Systems, 2002. ISCAS 2002, On pages: V-133- V-136 vol.5, 2002.

[20] Michael Dyer, David Taubman, Saeid Nooshabadi, "Memory Efficient Pass-Parallel Architecture for JPEG2000 Encoding", Int. Symposium on Signal Processing and its Applications , ISSPA, July 2003.

[21] Mu-Yu Chiu, Kun-Bin Lee, Chein-Wei Jen, "Optimal Data Transfer and Buffering Schemes for JPEG2000 Encoder", IEEE Workshop on Signal Processing Systems (SIPS 2003), Pages: 177- 182, 2003.

[22] Peter Meerwald, Roland Norcen, Andreas Uhl, "Parallel JPEG2000 Image Coding on Multiprocessors", Proceedings of the International Parallel & Distributed Processing Symposium, IPDPS '02, Fort Lauderdale, FL, USA, 2002.

[23] Paul Schumacher, "An Efficient, Optimized JPEG2000 Tier-1 Coder Hardware Implementation", Visual Communications and Image Processing 2003. Edited by Ebrahimi, Touradj; Sikora, Thomas. Proceedings of the SPIE, Volume 5150, pp. 1089-1096, 2003.

[24] Hung-Chi Fang, Chao-Tsung Huang, Yu-Wei Chang, Tu-Chih Wang, Po-Chih Tseng, Chung-Jr Lian, Liang-Gee Chen, "81MS/s JPEG 2000 Single Chip Encoder with Rate Distortion Optimization", IEEE International Solid-State Circuits Conference, ISSCC 2004, San Fransisco, February 15-19, 2004.

[25] Hideki Yamauchi, Kenji Mochizuki, Kazuhiko Taketa, Tsuyoshi Watanabe, Tsugio Mori, Yuh Matsuda, Yoshifumu Matsushita, Akio Kobayashi, Shigeyuki Okada, "A 1440x1080 Pixels 30 Frames/sec Motion-JPEG2000 Codec for HD Movie Transmission", IEEE International Solid-State Circuits Conference, ISSCC 2004, San Fransisco, February 15-19, 2004.

[26] Manjunath Gangadhar, Dinesh Bhatia, "FPGA based EBCOT Architecture for JPEG2000", 2004 IEEE International Conference on Field-Programmable Technology (FPT'04), 6-8 December, 2004

[27] Michael Dyer, David Taubman, Saeid Nooshabadi, "Improved Throughput Arithmetic Coder for JPEG2000", IEEE International Conference on Image Processing, ICIP2004, Singapore. October 24-27 2004

[28] Dang, P.P.; Chau, P.M.; "A high performance, low power VLSI design of discrete wavelet transform for lossless compression in JPEG 2000 standard", Consumer Electronics, 2001. ICCE. International Conference on , 2001.

[29] Kishore Andra, Chaitali Chakrabarti, and Tinku Acharya, "VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform", IEEE Transactions on signal processing, vol. 50, no. 4, April 2002

[30] Diou, C.; Torres, L.; Robert, M., "A wavelet core for video processing"; Image Processing, 2000. Proceedings. 2000 International Conference on, Volume 2, Page(s):395 - 398 vol.2, 10-13 Sept. 2000

[31] Diou, C., Torres, L., Robert, M., "An embedded core for the 2D wavelet transform", Emerging Technologies and Factory Automation, 2001, Proceedings. 2001 8th IEEE International Conference on Volume 2, Page(s):179 - 186 vol.2, 15-18 Oct. 2001.

[32] Dimitroulakos, G., Zervas, N.D., Sklavos, N., Goutis, C.E., "An efficient VLSI implementation for forward and inverse wavelet transform for JPEG2000", Digital Signal Processing, 2002. DSP 2002. 2002, 14th International Conference on Volume 1, Page(s):233 - 236 vol.1, 1-3 July 2002

[33] Movva, S., Srinivasan, S., "A novel architecture for lifting-based discrete wavelet transform for JPEG2000 standard suitable for VLSI implementation", VLSI Design, 2003. Proceedings. 16th International Conference on, 2003

[34] Benderli, O., Tekmen, Y.C., Ismailoglu, N., "A real-time, low latency, FPGA implementation of the 2-D discrete wavelet transformation for streaming image applications", Digital System Design, 2003. Proceedings. Euromicro Symposium on, Page(s):384 - 3891-6 Sept. 2003

Chapitre VI Conception SOPC

1. Introduction

Dans ce chapitre on présente deux stratégies d'exploration des implémentations du codeur entropique. Une première étude basée sur le partitionnement HW/SW a été menée. Cette étude a pour but l'exploitation des différentes ressources de la plateforme cible à savoir le Virtex-II-Pro qui a la particularité par rapport aux anciens circuits tels que le Virtex et le Virtex-II d'englober en plus de la logique reconfigurable, le noyau d'un processeur qui dans le cas de Virtex-II-Pro est un PowerPC 405.

L'enjeu s'est porté principalement, sur comment choisir la meilleure méthode pour réaliser le partitionnement HW/SW, quelle architecture choisir, comment programmer l'algorithme pour un SOC, et comment simuler des architectures mixtes ? Quelle référence et méthodologie de test fallait-il utiliser ? Pour faire par la suite des exécutions sur carte.

La deuxième étude est un complément de la première. Elle consiste à exploiter le macro parallélisme que présente le codeur entropique pour implémenter plusieurs codeurs matériels en parallèle. Trois versions ont été implémentées dans cette étude. Ces versions sont (1) monocodeur, (2) bi-codeur et (3) quadri-codeur. Dans le cas de cette étude le problème de test s'est imposé. Pour les implémentations issues des deux études, deux plateformes basées sur des composants Virtex-II-Pro ont été utilisées. La première plateforme est une carte Memec base sur le Virtex-II-Pro XC2VP4. La deuxième est une carte Alpha Data construite autour du XC2VP20.

Les tests et les mesures de performance ont été effectués pour la première étude sur quelques code-blocks de l'image Lena de taille 512x512 pixels. Pour la seconde étude plusieurs images ont été utilisées.

Dans ce chapitre on commencera par donner un aperçu sur les circuits reconfigurables particulièrement les circuits Virtex-II-Pro. Ensuite on présentera les plateformes utilisées pour la réalisation des études citées précédemment. Une partie sera réservée pour les architectures HW et SW qui ont été implémentées. Finalement on donnera l'ensemble des résultats obtenus de point de vue temps d'exécution et utilisation des ressources.

2. Les circuits Virtex-II-Pro

Xilinx a mis depuis sur le marché depuis 2003, une nouvelle technologie de circuits FPGA intégrant de nouvelles fonctionnalités. Ces dernières permettent la réalisation d'architectures qui auparavant s'avéraient très compliquées à entreprendre.

Parmi ces nouveaux circuits; Le Virtex-II-Pro (**Figure Ch VI - 1**) qui est un composant FPGA intégrant un cœur de processeur PowerPC 405 de chez IBM pouvant fonctionner à plus de 300MHz. Afin de rendre le codesign plus simple à réaliser, la famille de composant Virtex-II-Pro possède plusieurs atouts tels que : l'intégration d'un ou deux cœurs de processeur PowerPC 405. Un grand nombre de slices pouvant atteindre 44,096 unités (3008 pour le XC2VP4, et 9240 pour le XC2VP20) permet d'implémenter des algorithmes que la complexité rend gourmands pour la consommation des ressources logiques. Des blocs de mémoire de 18Kb (28 pour le XC2VP4, et 88 pour le XC2VP20) sont aussi disponibles, ce qui diminuera l'accès vers des ressources externes ce qui peut être fatal pour le circuit surtout du point de vue consommation d'énergie. Elles offrent aussi des multiplieurs 18x18 bits, des DCM (Digital Clock Manager), et des blocs de RoketIO transceivers.

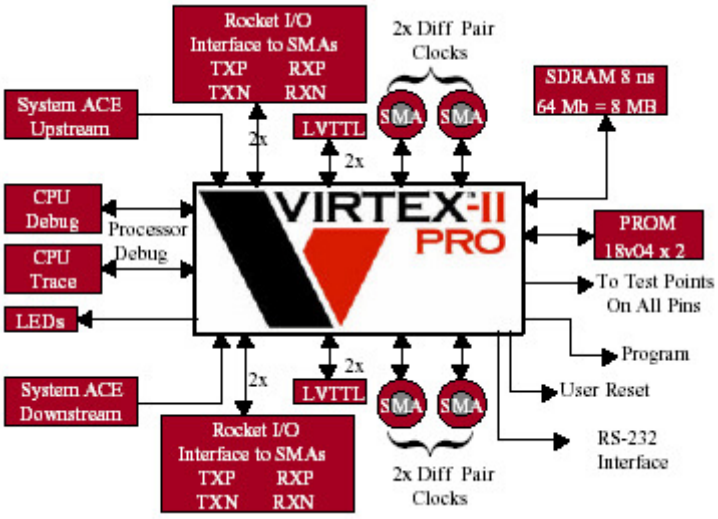


Figure Ch VI - 1 : Vue Globale du Virtex-II-Pro

Le choix de ce circuit ne s'est pas effectué de manière hasardeuse, mais en étudiant préalablement son architecture ainsi que toutes les opportunités que ce dernier peut nous offrir Le principe dans Virtex-II-Pro est que le coeur du processeur PowerPC 405 se greffe directement sur le FPGA (**Figure Ch VI - 2**), c'est à l'utilisateur de rajouter les périphériques selon ses besoins ce qui permet de moduler à notre gré l'architecture à implémenter. En effet, en ce qui concerne le partitionnement HW/SW et plus précisément dans le cas d'un codeur entropique, la flexibilité de l'architecture est un grand avantage car elle permet de se greffer sur le PowerPC à différents niveaux : au niveau de l'OPB pour de la logique lente, et sur le PLB pour la logique rapide. Dans le cas de la seconde étude, le PowerPC sert comme contrôleur pour l'ensemble des codeurs qui sont implémentés sur la logique reconfigurable, en plus il peut aussi exécuter une partie de la chaîne de codage de JPEG2000.

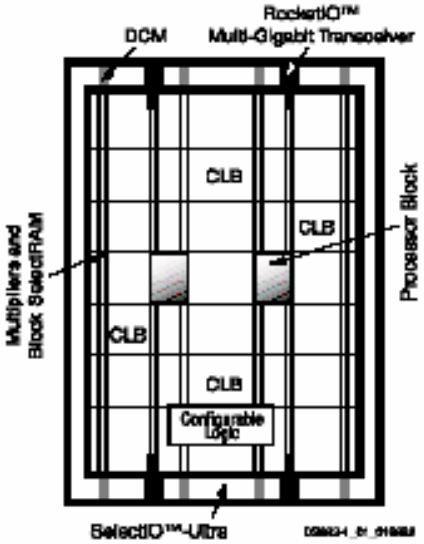


Figure Ch VI - 2 : Architecture interne du Virtex-II-Pro

Pour cela Xilinx fournit des IP soft écrites en langage VHDL et qui permettent d'exploiter le PowerPC 405 et lui fournissent tous les périphériques nécessaires à son fonctionnement. Les premières IP à implémenter autour du processeur sont le « Processor Local Bus » et le « On Chip Memory Controller », la première constitue le bus de communication sur lequel viennent se greffer d'autres IP, quant au «On Chip Memory Controller » permet de connecter le PowerPC 405 avec les BRAM du FPGA où sera logé le code à exécuter.

Une application peut se limiter à une configuration avec un PowerPC 405, une IP « processor Local Bus » et un « On Chip Memory Controller », l'utilisateur peut, comme nous l'avions expliqué dans le précédent paragraphe, rajouter des bus ou bien des contrôleurs de mémoire externe ou autres IP, selon les exigences de son architecture.

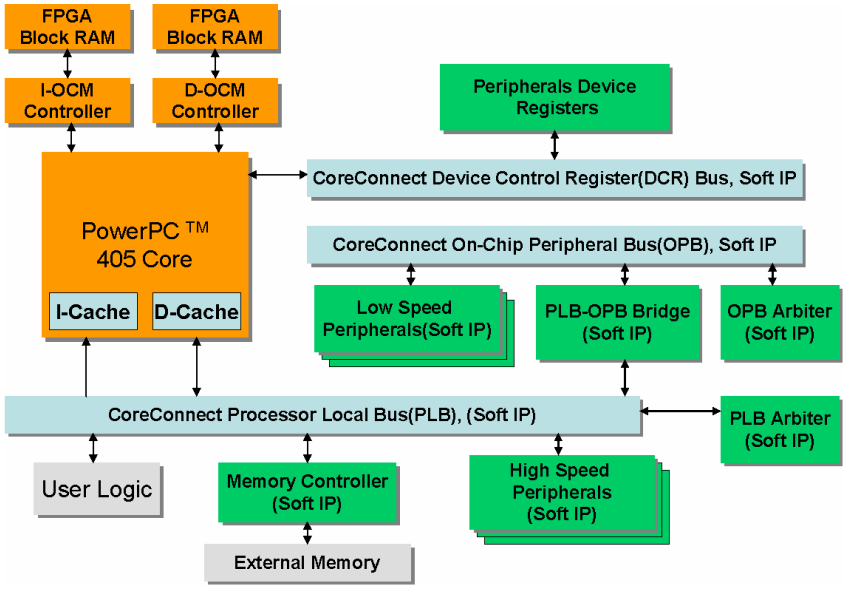


Figure Ch VI - 3 : Architecture standard autour du PowerPC 405 sur Virtex-II-Pro

En ce qui concerne la consommation d'énergie du PPC 405, le constructeur évalue approximativement sa consommation moyenne à 1mW par MHz. Nous basons nos calculs de consommation d'énergie sur cette donnée vu que nous ne disposons pas d'un moyen efficace tel qu'un outil logiciel, qui nous permette le calcul en dynamique de l'énergie consommée. De plus il est impossible de mesure cette consommation sur la plateforme.

2.1.1 Les ressources du Virtex-II-Pro

La famille Virtex-II-Pro dispose d'un nombre suffisant de ressources permettant la réalisation d'applications SOC et ce avec une électronique associée très réduite. Le tableau 1 donne un récapitulatif des ressources dont dispose le composant de la famille Virtex-II-Pro que nous avions utilisé, en l'occurrence le XC2VP4, et le XC2VP20.

Tableau Ch VI - 1 : Caracteristiques de la famille Virtex-II-Pro/Virtex-II-Pro X

Device	RocketIO Transceiver Blocks	PowerPC Processor Blocks	Logic Cells	CLB (1 = 4 slices = max 128 bits)		18X 18 Bit Multiplier Blocks	Block SelectRAM+		DCMs	Maximum User I/OPads
				Slices	Max Distr RAM(Kb)		18Kb Blocks	Max Blocks RAM (Kb)		
XC2VP2	4	0	3,168	1,408	44	12	12	216	4	204
XC2VP4	4	1	6,768	3,008	94	28	28	504	4	348
XC2VP7	8	1	11,088	4,928	154	44	44	792	4	396
XC2VP20	8	2	20,880	9,280	290	88	88	1,584	8	564
XC2VPX20	8	1	22,032	9,792	306	88	88	1,584	8	552
XC2VP30	8	2	30,816	13,696	428	136	136	2,448	8	644
XC2VP40	0, 8, or 12	2	43,632	19,392	606	192	192	3,456	8	804
XC2VP50	0 or 16	2	53,136	23,616	738	232	232	4,176	8	852
XC2VP70	16 or 20	2	74,448	33,088	1,034	328	328	5,904	8	996
XC2VPX70	20	2	74,448	33,088	1,034	308	308	5,544	8	992
XC2VP100	0 or 20	2	99,216	44,096	1,378	444	444	7,992	12	1,164

2.1.2 Les éléments du CoreConnect

Les systèmes logiques sur puce sont souvent conçus de manière à être dimensionnés pour une seule application particulière. Chaque application a sa propre architecture. De ce fait l'extension ou la réutilisation de telle architecture est difficile et nécessite un effort de réadaptation.

Pour cette raison IBM a proposé l'architecture CoreConnect qui offre trois bus qui permettent la connexion des cœurs de processeur, la logique dédiée, et différents autres systèmes sur puce. Ces bus sont :

- Processor Local Bus(PLB)
- On-chip Peripheral Bus(OPB)
- Device Control Register (DCR)

Généralement le PLB fournit un chemin de données large bande. En effet il est utilisé pour connecter des composants tels que les coeurs de processeurs, les interfaces avec les mémoires externes, et les contrôleurs DMA (Direct Memory Acces).

L'OPB est utilisé pour réduire la charge du bus PLB. Il est plus adapté aux composants tels que les ports série les ports parallèles, UARTs, GPIO, et tous les composants à faible largeur de bande. Un Maître sur le PLB peut accéder aux composants connectés à l'OPB via une macro passerelle (Bridge). Cette passerelle est vue en tant que maître par l'OPB et en tant qu'esclave par le PLB. Les registres d'état et de configuration de faible performance sont généralement lus et écrits via le bus DCR.

Pour simplifier la connexion d'un module logique de l'utilisateur à un des bus de CoreConnect, ce dernier peut se servir de l'une des interfaces de bus portable préconçue dite IPIF (IP interface). L'IPIF prend en compte les signaux du bus et le protocole de la communication et l'ensemble des caractéristiques du bus. L'IPIF présente une interface pour la logique utilisateur dite IPIC (IP Interconnect). Lorsque la logique utilisateur est conçue avec un IPIC elle peut être portable et facilement réutilisable avec différent bus en changeant seulement l'IPIF. Des fichiers VHDL qui instancient l'IPIF et fournissent le code nécessaire à l'utilisateur pour ajouter ses modules, simplifie la tache de connexion de la logique utilisateur. Ces fichiers sont les User Core Reference Design.

Dans les figures suivantes on présente les deux principales méthodes pour ajouter de la logique utilisateur. Dans la première figure (**figure Ch VI - 4**) la logique utilisateur est externe au système du processeur. Elle utilise l'IPIC pour se connecter au bus via l'IPIF. Dans la deuxième figure la logique utilisateur fait partie du système du processeur. Elle est compactée avec l'IPIF et l'IPIC dans un seul module.

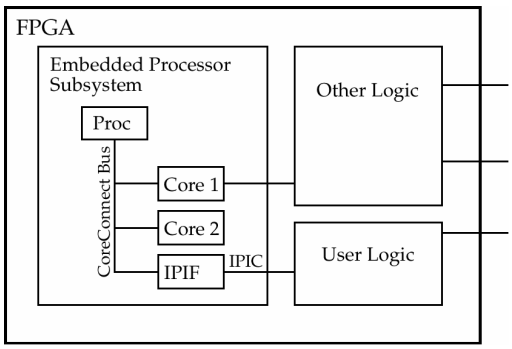


Figure Ch VI - 4 : La logique utilisateur externe au système du processeur

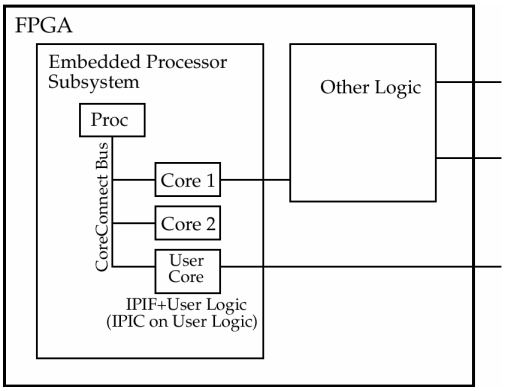


Figure Ch VI - 5 : La logique utilisateur est interne au système du processeur

3. Flots de conception Xilinx

Pour la création et la vérification d'un système embarqué, Xilinx propose un ensemble d'outils qui sont regroupés dans deux grands logiciels qui sont ISE (integrated Software Environment) et XPS (Xilinx platform studio). Les outils de ISE sont utilisés par XPS.

La conception d'un système embarqué inclut typiquement les phases suivantes :

- Création de la plateforme matérielle
- Vérification de la plateforme matérielle
- Création de la plateforme logicielle
- Création et vérification de l'application logicielle

La plateforme matérielle est définie par le fichier MHS (Microprocessor Hardware Specification). C'est la connexion d'un ou plusieurs processeur et périphériques sur les bus de ce dernier. L'utilisateur peut définir et ajouter ses propres périphériques. L'outil XPS fournit des moyens graphiques pour la création du fichier MHS.

Le fichier MHS définit l'architecture, les connexions et le mapping des adresses du système. A partir du fichier MHS l'outil Platform Generator (PlatGen) crée la netlist du système sous différents formats (NGC, EDIF) et les wrappeurs HDL des niveaux TOP pour que l'utilisateur puisse intégrer ses composants au système. Suite à l'application de PlatGen les outils ISE sont utilisés pour compléter l'implémentation du système. La création d'une plateforme matérielle est résumée dans la figure suivante :

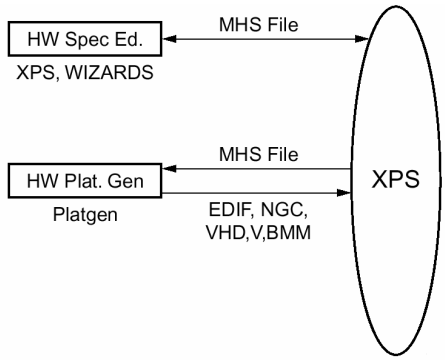


Figure Ch VI - 6 : Création d'une plateforme matérielle

La plateforme de vérification nécessite la plateforme matérielle. Elle permet à l'utilisateur de définir le modèle de simulation pour chaque composant du système (processeur et périphériques). L'outil SimGen crée le fichier de simulation (HDL ou d'autres modèles compiler) à partir du MHS, ainsi que les fichiers de commande. Si l'application logicielle est

disponible sous sa forme exécutable, elle peut être utilisée pour initialiser les mémoires. La figure suivante représente la plateforme de vérification.

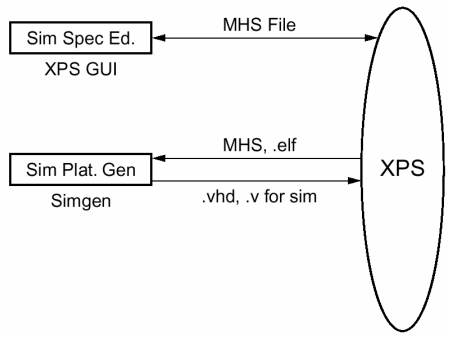


Figure Ch VI - 7 : Vérification d'une plateforme matérielle

La plateforme logicielle est définie par le fichier MSS (Microprocessor Software Specification). Ce fichier regroupe les drivers et les librairies pour l'adaptation des paramètres des périphériques et ceux des processeurs, les routines d'interruption les composants d'entrées/sorties. Le fichier MSS est utilisé par l'outil Library Generator (LibGen) pour l'adaptation des drivers et librairies et les routines d'interruption. Le processus de la création de la plateforme logicielle est donné dans la figure suivante :

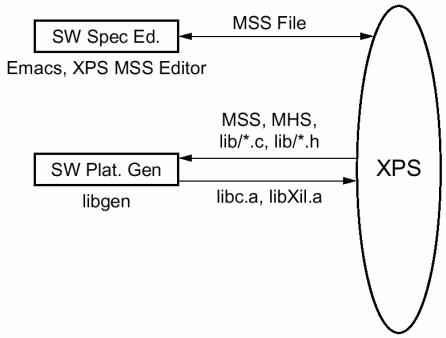


Figure Ch VI - 8 : Création d'une plateforme logicielle

La création et la vérification d'une application logicielle passe par : d'abord l'écriture du code en C, C++ ou assembleur qui va être exécuté sur les plateformes SW et HW. Ensuite ce code est compilé et linké à l'aide de l'outil GNU (d'autres outils peuvent aussi être utilisés) pour générer le fichier exécutable sous le format ELF (Executable and Link Format). Finalement, XMD et le débogueur de GNU (GDB) sont utilisés pour déboguer l'application. La figure suivante représente le processus de création et de vérification d'une application SW.

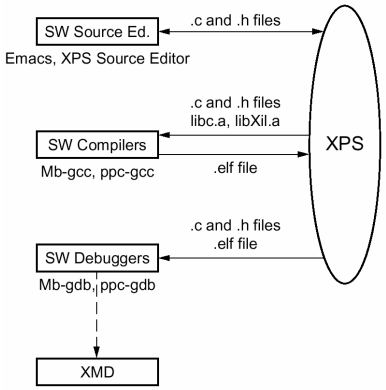


Figure Ch VI - 9 : Création et vérification d'une application software

Les processus supportés par l'outil XPS sont principalement :

- Création du fichier MHS
- Création du MHS
- Réalisation de la matrice de connexion des différents bus
- L'adaptation des drivers, des librairies, et des contrôleurs d'interruptions
- La gestion des fichiers sources

Les détails de ces différents processus sont représentés dans la figure suivante :

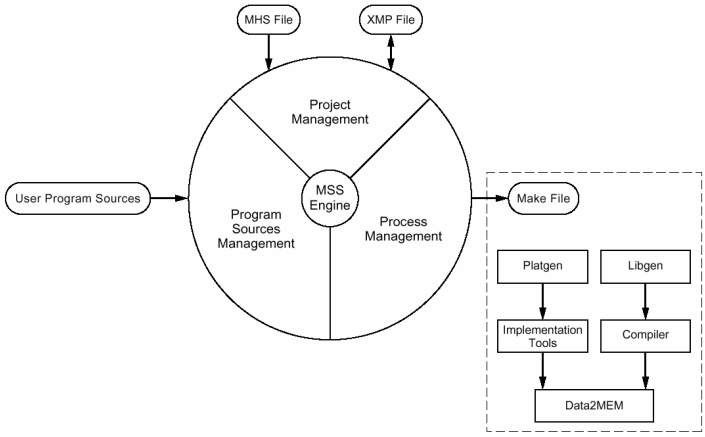


Figure Ch VI - 10 : Les Processus supportes par XPS

4. Les plateformes matérielles

4.1 Carte MEMEC Virtex-II-Pro XC2VP4

La carte Memec est fabriquée et vendue par la société INSIGHT-MEMEC. C'est une carte de développement avec comme élément principal le Virtex-II-Pro 2VP4FG456 ainsi que tous les périphériques nécessaires afin de tester le composant suivant différentes configurations. L'architecture de la carte est détaillée dans la **Figure Ch VI - 12** sur cette carte on trouve en plus du Virtex-II-Pro, une SDRAM de 8Mx32, 4 sources d'horloge, un port RS-232, des connections permettant l'ajout de modules standards, un port JTAG pour la configuration et le déboguage.

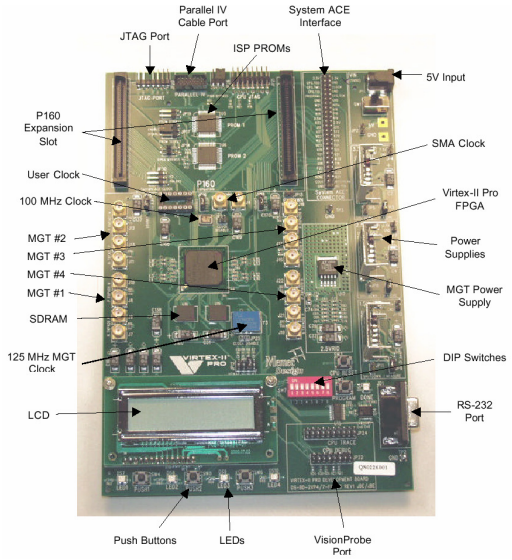


Figure Ch VI - 11 : La carte de développement Virtex-II-Pro.

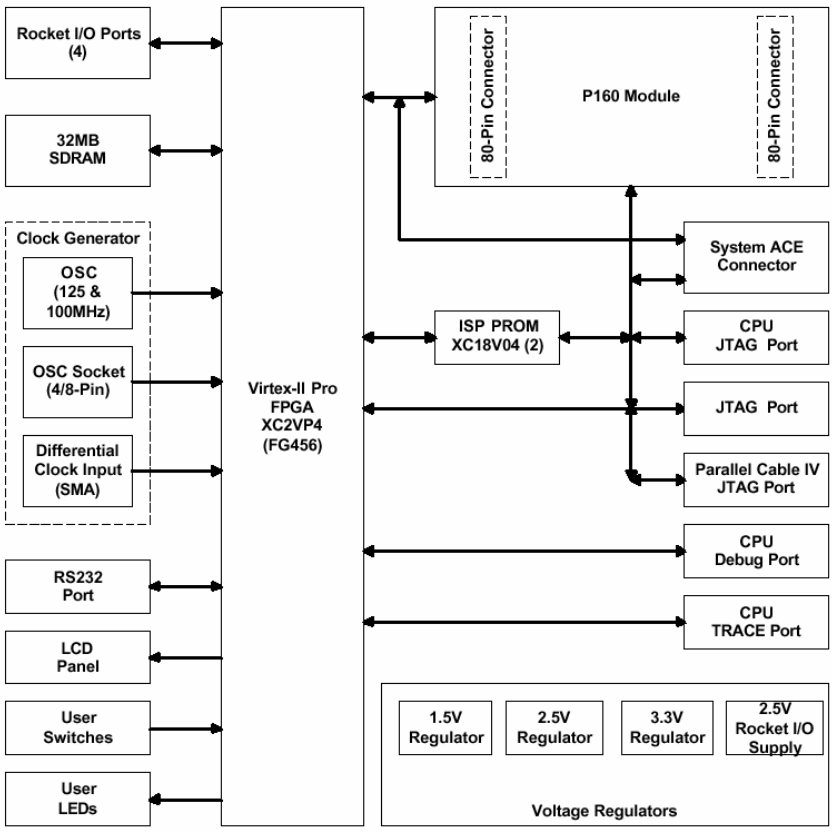


Figure Ch VI - 12 : Ressources de la carte MEMEC

4.2 Carte Alpha Data ADM-XPL

La carte ADM-XPL est une carte PMC qui est greffée sur une carte mère PCI. Donc l'exploitation de cette carte se fait dans notre cas à travers le bus PCI. En plus du Virtex-II-Pro cette carte dispose d'un banc de mémoire ZBT SRAM de 1024x64 bits, de 128 Mo de DDR SDRAM, et de 16 Mo de mémoire Flash. La bande passante du bus PCI peut atteindre 528Mo/s. L'architecture de la carte est donnée dans la **Figure Ch VI - 13**

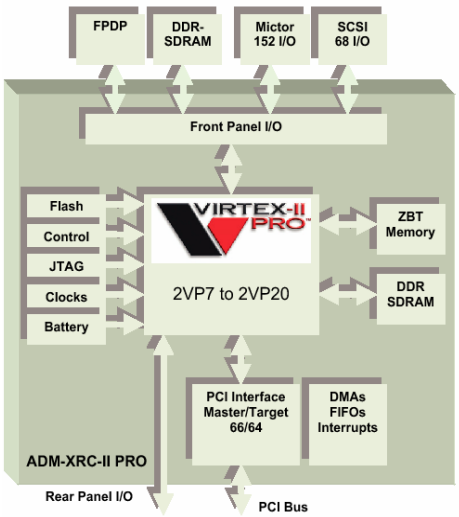


Figure Ch VI - 13 : Architecture de la carte ADM-XPL

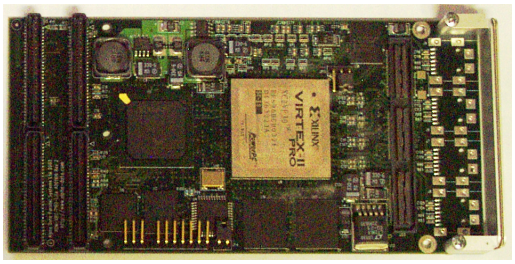


Figure Ch VI - 14 : Photo de la carte ADM-XPL

5. Implémentation du Codeur Entropique

5.1 Partitionnement HW/SW du Codeur Entropique

Le but de ce travail est principalement de réaliser différentes architectures de codeur entropique selon un partitionnement HW/SW sur une plateforme Virtex-II-Pro. Les deux modules du codeur entropique à savoir la formation des contextes (FC) et le codage arithmétique (CA) feront l'objet de ce partitionnement. En effet quatre architectures sont à implémenter :

Tableau Ch VI - 2 : Les versions implémentées

	Formation des contextes (FC)		Codage arithmétique (CA)	
	Logiciel	Matériel	Logiciel	matériel
Version 1	X		X	
Version 2		X	X	
Version 3	X			X
Version 4		X		X

5.1.1 Version 1

FC: Soft, CA: Soft

Cette première façon d'implémenter le codeur entropique consiste à exécuter en soft, par le processeur embarqué sur le circuit Virtex-II-Pro, la FC ainsi que le CA. Ces deux programmes ont été écrits en langage C puis compilés à l'aide de l'outil de conception XPS afin qu'ils soient implémentés sur le Virtex-II-Pro. Ces derniers prendront place dans les BRAMs du circuit FPGA où le processeur PPC405 se chargera d'exécuter les instructions s'y trouvant, puis par la suite de sortir les résultats et les afficher à l'aide de l'UART via le port série de l'ordinateur sur l'écran à l'aide de l'outil Hyperterminal de Windows. Une architecture nécessaire au fonctionnement du cœur PPC405 doit d'abord être choisie et implémentée en utilisant les IP de CoreConnect d'IBM fournis par l'outil. Ces IP sont : le PLB bus, PLB2OPB_bridge (interface entre les deux types de bus), le bus OPB ainsi que les GPIOs et l'UART. Le Virtex-II-Pro, permet d'adapter les ressources du cœur PPC405 selon les besoins de ce dernier.

Une fois l'architecture établie et le programme compilé, on est passé à la simulation, où le but principal était la mesure du temps d'exécution total d'un code-block de l'image Lena avec cette manière d'implémenter. Ensuite une exécution sur la carte a permis de valider le fonctionnement du codeur sur le Virtex-II-Pro de la carte Memec.

Donc, pour cette première version les résultats ont été obtenus par simulation, ainsi que par exécution sur la carte MEMEC Virtex-II-Pro 2vp4 dans sa révision 2. L'architecture de cette première version est donnée par la **Figure Ch VI - 15**.

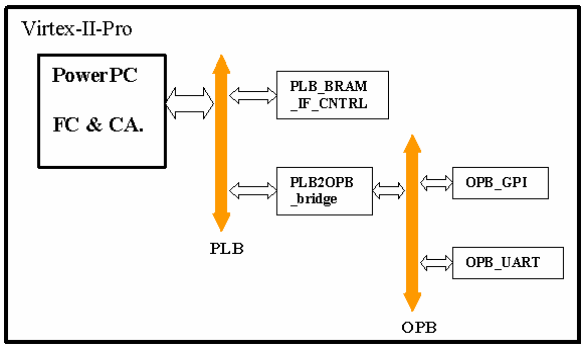


Figure Ch VI - 15 : Architecture de la version 1

Par la suite, cette première version d'implémentation du codeur entropique a été améliorée, pour être plus réaliste, en séparant les données des code-blocks du programme à exécuter par le PPC405, afin d'avoir une occupation moindre de l'espace BRAMs réservé principalement au code source, cela a été réalisé en chargeant les code-blocks (les données) dans la SDRAM à proximité du Virtex-II-Pro.

5.1.2 Version 2

FC: Hard, CA: Soft

Dans cette version, l'implémentation a été faite en mettant la FC sur la logique reconfigurable (écrit en VHDL), et le CA en logiciel qui sera exécuté par le processeur PPC405 .

Au démarrage la partie FC, cherche les coefficients du code-block à partir d'une BRAM sur le FPGA afin qu'il commence son traitement et la génération des contextes. Au fur et à mesure que le module FC fournit des données, ces dernières sont dirigées et stockées dans une FIFO capables de stocker 32 mots de 6 bits (5 bits pour le contexte, 1 bit pour la décision).

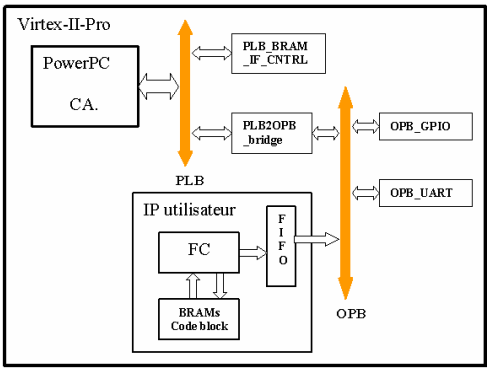


Figure Ch VI - 16 : Architecture de la version 2

D'autre part, dès que le module CA, qui lui, est exécuté par le PowerPC, détecte que la FIFO n'est plus vide, ce dernier commence à traiter les données afin qu'il génère les données compressées. Il s'arrêtera une fois qu'il aura vidé la FIFO et que le module FC aura terminé.

Certains problèmes sont apparus avec la synthèse du module FC. En effet la netlist générée par le compilateur proposé par Xilinx dans l'outil de conception XPS, étant le XST (Xilinx Synthesis Technology), présente des instabilités de fonctionnement. Nous avons par conséquent été contraints à passer vers un autre compilateur afin de compiler le programme du FC. Pour cela nous avons utilisé le FPGA EXPRESS 3.6.1 de chez Synopsys fourni avec l'outil de conception de Xilinx Foundation 4.1i. Les étapes étaient comme suit : nous avons synthétisé le module FC avec FPGA EXPRESS, puis récupéré la netlist générée, cette dernière a été utilisée avec XPS et connectée sur le bus OPB.

Sur cette deuxième version les résultats ont été obtenus d'abord par simulation. Ensuite l'exécution a permis de récupérer à travers le port série (UART) du PPC405 les données compressées relatives aux code-blocks traités et de les afficher sur écran.

Le choix de la connexion du module FC sur le bus OPB nous a été imposé par l'outil dans sa version 3.2i, qui ne fournissait pas les drivers relatifs à l'utilisation du bus PLB.

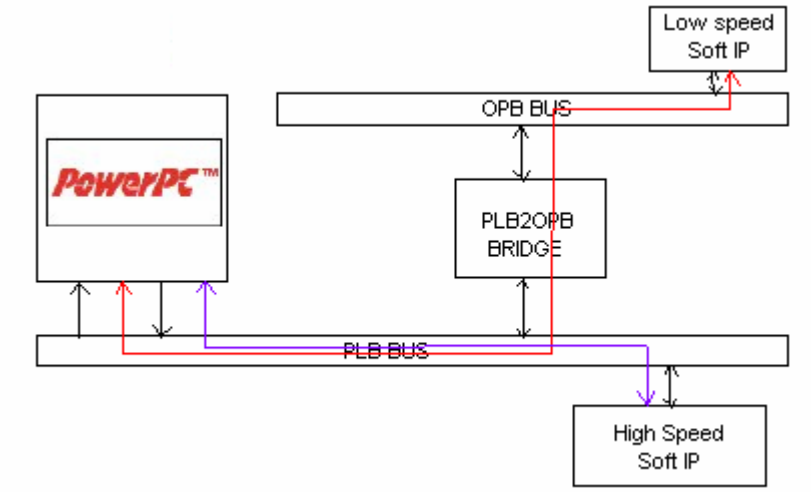


Figure Ch VI - 17 : chemin de donnée de l'OPB IPIF vs le PLB IPIF.

Sur la figure ci-dessus, le trait en rouge illustre le chemin de données emprunté lors de l'utilisation d'une IP greffée sur le bus OPB (OPB IPIF) et le trait en bleu illustre le chemin de données emprunté lors de l'utilisation d'un PLB IPIF. On remarquera que le partitionnement d'un algorithme quelconque réalisé avec un OPB IPIF aura un temps d'exécution plus important que celui réalisé avec un PLB IPIF vue l'importance des chemins empruntés.

On remarquera que l'occupation des cellules du circuit FPGA du Virtex-II-Pro, est plus importante que dans la version précédente (version 1), cela est dû principalement à l'implémentation d'une partie du codeur entropique en HW. Dans ce cas, il va falloir faire très attention aux temps de propagation sur circuit FPGA et imposer s'il le faut des contraintes de temps afin d'assurer et de stabiliser le fonctionnement de l'implémentation.

5.1.3 Version 3

FC: Soft, CA: Hard

Le but de cette version était essentiellement d'explorer une autre alternative de partitionnement du codeur entropique. Contrairement à la version précédente (version 2), dans celle-ci, le module FC; sera exécuté par le PPC405, les instructions seront logées dans les BRAMs du FPGA (joueront le rôle d'une ROM). Quant au codeur arithmétique, il sera dans cette version, implémenté en HW après avoir était écrit en VHDL

L'exécution de cette implémentation se fera de la manière suivante : au démarrage le PPC405 est contraint de booter sur les PLB-BRAMs afin de commencer à exécuter les instructions s'y trouvant et qui le mèneront au programme du module FC. Le PPC405 traitera les données du code-block se trouvant dans les BRAMs et enverra après chaque traitement le résultat apparu en sortie vers une FIFO paramétrée à 32 mots de 6 bits.

Du côté matériel, où se trouve implémenté le module CA, dès que se dernier détecte la présence d'une donnée dans la FIFO, il se met à coder, et ce, jusqu'à ce que la FIFO soit vide et que la FC ait fini le codage. Cette même partie se charge aussi d'envoyer les données vers

le PPC405 FIFO afin qu'il puisse les afficher sur l'écran de l'ordinateur à l'aide de la liaison RS232. Sachant que ces données (bitstream) passent d'abord par une deuxième FIFO où elles seront stockées le temps que le PPC405 finissent le traitement sur les valeurs précédentes.

L'architecture proposée pour cette version est donnée dans la **Figure Ch VI - 18**, où l'on peut remarquer le PPC405 comme élément principal, avec les différentes ressources nécessaires pour notre cas à savoir le PLB bus, BRAMs, et l'OPB bus. On remarquera aussi que l'IP utilisateur exécutant le CA est greffée sur le bus OPB.

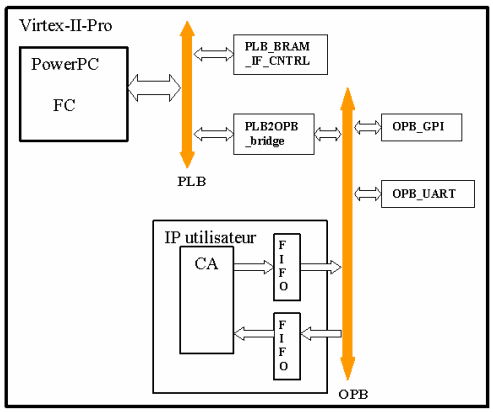


Figure Ch VI - 18 : Architecture de la version 3

Dans cette version les résultats ont été obtenus principalement par simulation vu que dans celle ci nous avons exploré la nouvelle manière de partitionnement utilisée déjà dans la version 2, qui consiste à créer une IP codeur entropique qui vient se greffer directement sur le bus OPB du processeur PPC405.

5.1.4 Version 4

FC: Hard, CA: Hard

Le but de cette version était de mettre l'ensemble des modules du codeur entropique en matériel sur le FPGA et d'utiliser le PPC405 rien que pour récupérer les données sortantes du codeur entropique. Tout comme la version 2 et 3, nous avons considéré le codeur entropique comme étant une IP s'exécutant dans ce cas indépendamment du PPC405. À l'instar de la version 2 nous avons eu recours au FPGA-EXPRESS car sous XST nous avons eu un disfonctionnement du codeur entropique lors de la simulation. Nous avons donc récupéré la netlist générée par la compilation sous FPGA EXPRESS puis on l'a injectée dans XPS et fait l'interface entre l'IP (codeur entropique) et le bus OPB.

Au démarrage du Virtex-II-Pro, le module VHDL en l'occurrence L'IP codeur entropique, se met à charger les données des code-blocks à partir d'une RAM modélisée en VHDL, et exécute au fur et à mesure le module FC. Les données générées seront chargées dans une FIFO, paramétrée à 32 mots de 6 bits.

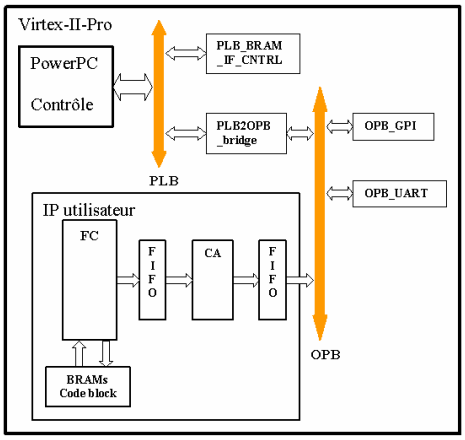


Figure Ch VI - 19 : Architecture de la version 4

Quant au CA, qui lui se trouve en amont du FC, il reste en attente qu'une valeur se charge dans la FIFO, dès que cela est fait, il commence à tirer les valeurs une par une et à les traiter. Ce dernier s'arrêtera une fois après avoir vidé complètement la FIFO. Les données en sortie du codeur arithmétique (codeur entropique plus généralement), seront envoyées vers le PPC405 après qu'elles aient été stockées temporairement dans une autre FIFO, le processeur se chargera de les afficher une par une via la liaison RS232.

Le codeur entropique, dans ce cas, a été considéré comme étant une IP greffée sur le bus OPB, il se chargera de lire les données d'un code-block qui sont situées dans une BRAM sur le FPGA, et d'exécuter la FC et le CA et fournira ensuite en sortie les bitstreams qui seront disponibles sur le PPC405.

5.2 Implémentation de plusieurs Codeurs Entropiques

Ce travail représente la suite logique du premier travail déjà présenté. Il consiste à faire la comparaison entre 3 architectures différentes. Une première architecture implémente un seul codeur entropique, qui sera utilisé essentiellement comme référence pour la comparaison des deux autres architectures. La deuxième architecture implémente deux codeurs entropiques qui peuvent être lancés en parallèle. La troisième architecture utilise 4 codeurs entropiques en parallèle. Dans cette partie toutes les architectures utilisent le bus PLB pour la connexion des codeurs entropiques vu que Xilinx a fourni les « User Core Reference Design » correspondant au bus PLB à partir de la version EDK 6.1i.

Le codeur entropique implémenté dans cette partie est composé d'une FIFO pour la récupération des coefficients du code-block que le codeur va traiter, une FIFO pour le transfert des données compressées qui ont été générées par le codeur, et le module de codage.

Le codeur est compacté avec l'IPIF comme dans le cas de la **Figure Ch VI - 5**. Dans le cas de cette partie le PowerPC s'occupe de la récupération des code-blocks à partir de la machine hôte via le bus PCI de la carte ADM-XPL pour les transférer vers le codeur entropique. Il se charge aussi du transfert des données compressées vers la machine hôte. Le transfert des données entre le bus PCI et le PowerPC passe par une FIFO se trouvant dans une IP dédiée pour la communication avec le bus PCI. Cette IP est connectée au bus PLB. La **Figure Ch VI - 20** présente une architecture avec 4 codeurs connectés au bus PLB. Les deux autres architectures sont semblables à celle de la **Figure Ch VI - 20** avec seulement le nombre de codeurs qui varie.

Les architectures ont été intégrées dans une chaîne de codage JPEG2000 pour les tester et les valider. Le fonctionnement de cette chaîne de codage est détaillé dans la **Figure Ch VI - 21**.

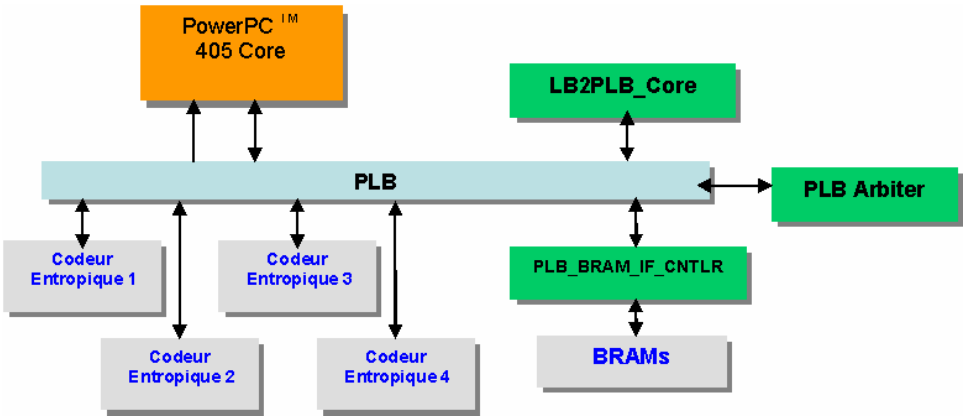


Figure Ch VI - 20 : Architecture de l'implémentation multi codeur entropique

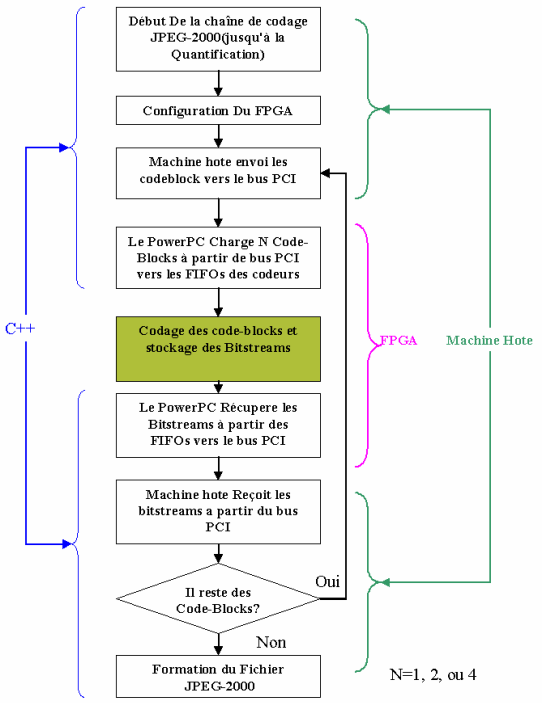


Figure Ch VI - 21 : Diagramme du fonctionnement de la chaîne mixte HW/SW

6. Les résultats

6.1 Première partie

Comme il a été annoncé les mesures de performance pour cette partie du travail ont été réalisées sur 9 code-blocks de l'image lena.ppm de taille 512x512 pixels. Les code-blocks sont obtenus suite à une transformation de couleur RGB→YUV, et une transformée en ondelettes 5/3 avec 5 niveaux de décomposition. Les code-blocks sont de taille 16x16 coefficients.

La première version est la plus lente car tout le codeur s'exécute sur le PowerPC. Sur la logique reconfigurable on implémente seulement l'architecture nécessaire au fonctionnement du cœur du PowerPC tel que le PLB, l'OPB, la passerelle entre les deux bus (PLB2OBP_bridge), le Contrôleur des BRAM (BRAM_IF_CNTLR), et l'UART.

La deuxième version, vu que le module FC est implémenté sur la logique reconfigurable, on remarque une augmentation dans la consommation des ressources. On remarque aussi une

amélioration de 74.5% dans le temps d'exécution. Avec une augmentation de 38% en nombre de Slices et 39% dans les BRAMs.

Dans la troisième version c'est la partie AC qui est implémentée sur la logique reconfigurable. Cette version est moins performante que la version 2 ce qui confirme la complexité de la partie FC du codeur entropique. Par rapport à la première version celle-ci ne rapporte que 12.8% de gain avec une augmentation des slices de 44% sans variation du nombre des BRAMs.

La version 4 est de loin la plus performante. Avec la mise du codeur entropique sur la logique reconfigurable le PowerPC n'intervient que pour la récupération des données compressées. Cette version est aussi celle qui consomme le plus de ressources.

Le nombre de cycles nécessaires pour le traitement de chacun des code-blocks pour les 4 versions est donné dans le tableau suivant.

Tableau Ch VI - 3 : Nombre de cycles des différentes versions pour différents code-blocks

N code-block	Nbre de Cycles version 1	Nbre de Cycles version 2	Nbre de Cycles version 3	Nbre de Cycles version 4
1	4955800	1354225	4062400	29445
2	5382600	1361774	4712800	32422
3	4703500	1176637	4165900	28726
4	4775600	1214319	4119600	29267
5	4689600	1192169	4117000	28292
6	3993000	995103	3547400	21965
7	5611600	1381626	4894300	30916
8	4603000	1136376	4072300	26173
9	3998500	1053624	3518600	24528
moyenne	4745911	1207317	4134477	27970

La figure ci-dessous est la représentation graphique du tableau précédent. Les variations entre les temps d'exécution des code-blocks pour une même version sont dues à la quantité de données traitées. En effet chaque code-block contient un nombre de bit-plane à traiter. Ces variations sont plus visibles pour les versions 1 et 3 mais les proportions de variation par rapport à la moyenne sont constantes entre les versions. Par exemple la variation entre le temps du code bloc 6 et celui du code bloc 7 est d'environ 32% du temps moyen pour chaque version.

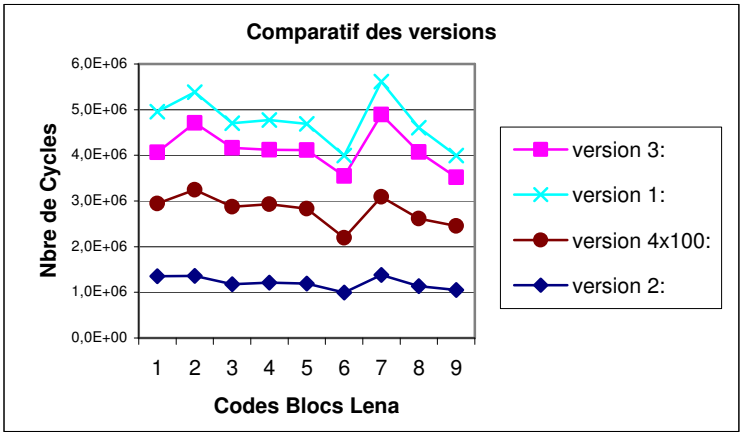


Figure Ch VI - 22 : Comparaison entre les 4 versions du partitionnement HW/SW

Dans le tableau suivant on donne la consommation des ressources en nombre de Slices et de BRAM pour chaque version. Le Virtex-II-Pro XC2VP4 contient 3008 Slices et 28 BRAM de

18Kbit chacune. Toutes les architectures implémentées ont été exécutées à la fréquence 100MHz.

Tableau Ch VI - 4 : Consommation de ressources

	version 1	version 2	version 3	version 4
Nombre de Slices	43%	81%	87%	99%
Nombre de PPC405s	100%	100%	100%	100%
Nombre de Bloc RAMs	57%	96%	57%	96%
Fréquence MHz	100	100	100	100

Ce travail représente une première exploration des possibilités d'implémentation du codeur entropique sur un système embarqué. La limitation de ce travail à une implémentation sur le bus OPB et l'absence d'une comparaison avec une implémentation sur le bus PLB sont dues à un ensemble de contraintes liées aux outils de développement et à l'implémentation initiale. Avec la version 3.2i de EDK, les IPIFs de l'OPB ne disposaient pas de FIFO, donc la comparaison ne se serait pas faite sur les mêmes architectures d'où les erreurs qui peuvent survenir que ce soit de point de vue consommation de ressource ou performance.

Ce travail s'est aussi limité aux tests sur des parties d'une image et non sur une image ou un ensemble d'image. Ceci est dû à la nécessité d'adaptation de l'architecture pour le chargement des données via la liaison UART de l'ensemble des code-blocks sur la SDRAM. Le chargement des code-blocks interfère avec le codage vu qu'il va utiliser les mêmes ressources à savoir le PLB et l'OPB.

L'absence du code C++ de la chaîne JPEG2000 compilé pour le Virtex-II-Pro nous a privé de tester et de valider les versions implémentées dans une chaîne de codage complète. La taille de certain code tel que Kakadu ou Jasper laisse penser qu'il est impossible que le XC2VP4 puisse contenir une chaîne complète de codage. Malgré toutes les lacunes de cette partie de travail, elle représente une étape importante dans l'étude de l'implémentation du codeur entropique sur un système embarqué. Ce travail a permis d'abord de confirmer la complexité du module FC du codeur entropique pour un processeur du type générique. Ensuite il a permis d'avoir une aide à la décision concernant la version à retenir selon les besoins de l'application à implémenter. Finalement nous pouvons dire que cette étude était une occasion pour découvrir le concept des composants Virtex-II-Pro qui offrent une bonne plateforme pour le partitionnement HW/SW.

6.2 2eme partie

Entre la première partie et celle-ci il y a eu des évolutions importantes. Ces évolutions concernent d'abord la plateforme de développement : passage de EDK 3.2i à EDK 6.1i puis 6.2i, et utilisation du composant Virtex-II-Pro XC2VP20 au lieu du XC2VP4. Cette évolution a permis aussi le passage du PLB à l'OPB pour la connexion du codeur au PowerPC avec l'arrivée des « User Core Reference Design » correspondant au PLB avec EDK 6.1i. Elle a aussi permis l'implémentation de plusieurs codeurs sur la logique reconfiguration vue l'importance des ressources disponibles sur le XC2VP20. Ces évolutions concernent aussi l'amélioration du fonctionnement du codeur et l'optimisation de ces ressources.

6.2.1 Version mono codeur

La version mono codeur a été exécutée de deux manières : la première est sans l'utilisation de la mémoire cache du processeur. Dans la seconde exécution, la mémoire cache a été activée ce qui a permis un gain dans le temps de 27% par rapport à la première exécution pour l'image lena. Cette version tourne à la fréquence de 100MHz.

Tableau Ch VI - 5 : Ressources et performances de la version mono codeur

Nbre de PPC405s	1 / 2	50%	
Nbre de RAMB16s	34 / 88	38%	
Nbre de Slices	2374 / 9280	25%	
Nbre de Cycle LENA	SANS CACHE	92670000	0,926 sec à 100MHz
Nbre de Cycle LENA	AVEC CACHE	67639000	0,676 sec à 100MHz
Fréquence		100MHz	
Pentium 4		2,00GHz	0,515 sec à 2,00GHz

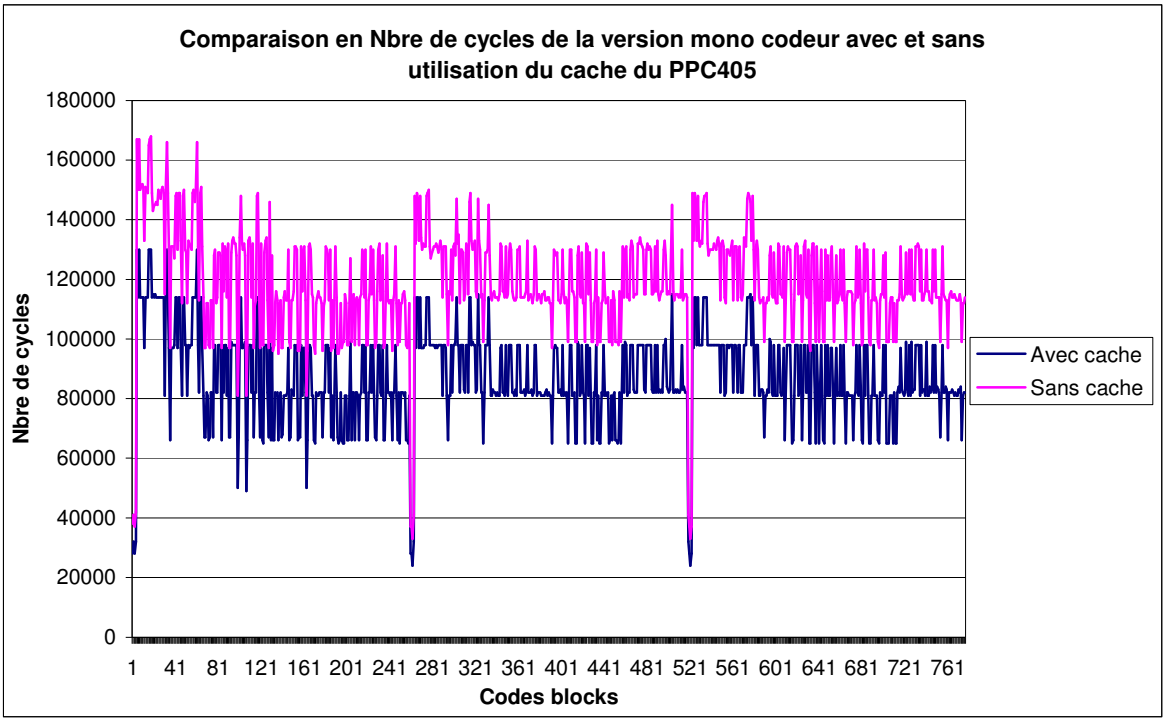


Figure Ch VI - 23 : Impact de l'utilisation du cache sur la performance de la version mono codeur

6.2.2 Version bicodeur

Cette version qui intègre deux codeurs entropiques a aussi été exécutée avec et sans activation de la mémoire cache. Le gain dans le temps remarque ici est de 24%. La fréquence de fonctionnement dans le cas de cette architecture a été affectée par le chemin de données qui est plus long et la taille du circuit. La fréquence maximale est de 76MHz. Sur la **Figure Ch VI - 24** le temps pris en compte le temps maximal de l'exécution de deux code-blocks.

Tableau Ch VI - 6 : Ressources et performances de la version bi-codeur

Nbre de PPC405s	1 / 2	50%	
Nbre de RAMB16s	49 / 88	55%	
Nbre de Slices	3937 / 9280	42%	
Nbre de Cycle LENA	SANS CACHE	47191000	0,620 sec à 76MHz
Nbre de Cycle LENA	AVEC CACHE	35703000	0,470 sec à 76MHz
Fréquence		76MHz	
Pentium 4		2,00GHz	0,515 sec à 2,00GHz

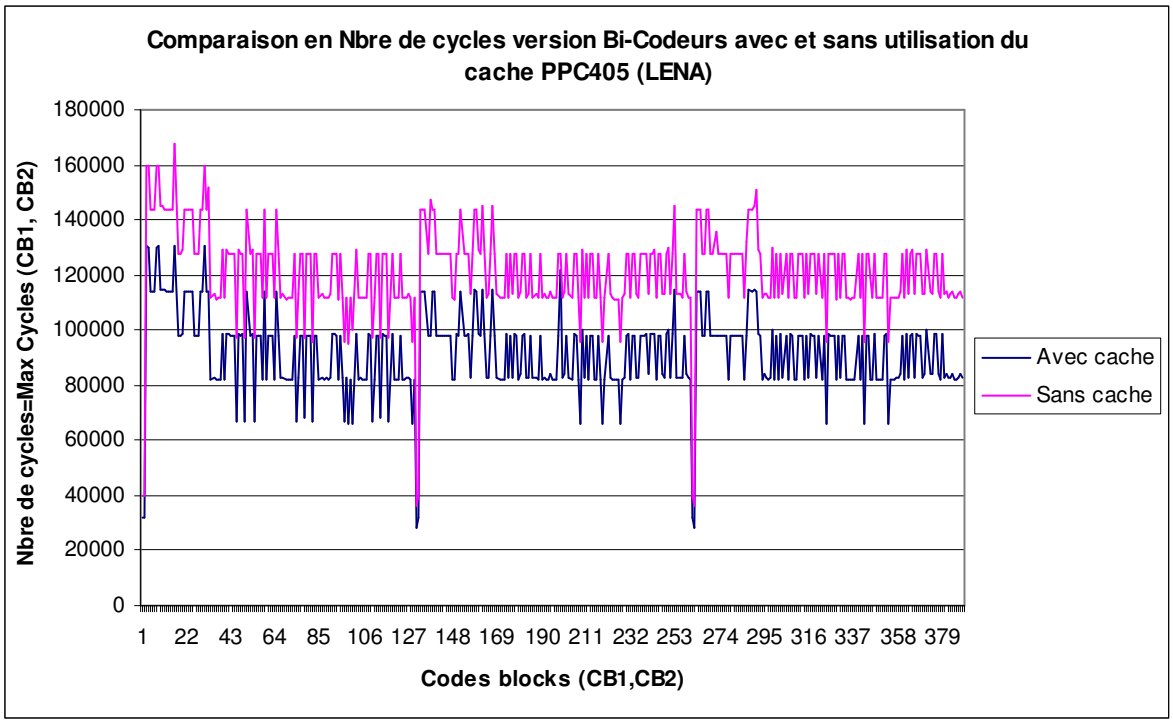


Figure Ch VI - 24 : Impact de l'utilisation du cache sur la performance de la version Bicodeur

6.2.3 Version quadri codeurs

Vu l'importance du gain dans le temps d'exécution avec la mémoire cache active, cette version a été exécutée uniquement avec activation de la mémoire cache.

Tableau Ch VI - 7 : Ressources et performances de la version quadi codeur

Nbre de PPC405s	1 / 2	50%	
Nbre de RAMB16s	79 / 88	89%	
Nbre de Slices	7091 / 9280	76%	
Nbre de Cycle LENA	AVEC CACHE	23155000	0,305 à 76MHz
Fréquence		76MHz	
Pentium 4		2,00GHz	0,515 sec à 2,00GHz

Plusieurs images on été utilisées pour le test et la validation des différentes versions implémentées. Ces images sont de taille et de format différents. La **Figure Ch VI - 25** présente une comparaison entre les temps d'exécution des 3 versions et une implémentation logicielle du codeur qui s'exécute sur un Pentium 4 à 2 GHz.

Tableau Ch VI - 8 : Comparaison de différentes versions du codeur entropique

	mono codeur	bi codeurs	quadri codeurs	C++
Lena 512x512	0,676	0,470	0,305	0,515
FreeBSD 512x512	0,611	0,334	0,285	0,364
Linda 512x512	0,608	0,324	0,281	0,411
Carte 768x512	0,452	0,304	0,181	0,368
Noise 1024x1024	1,395	0,929	0,538	1,223
Aerial2 2048x2048	3,993	2,715	1,674	3,184

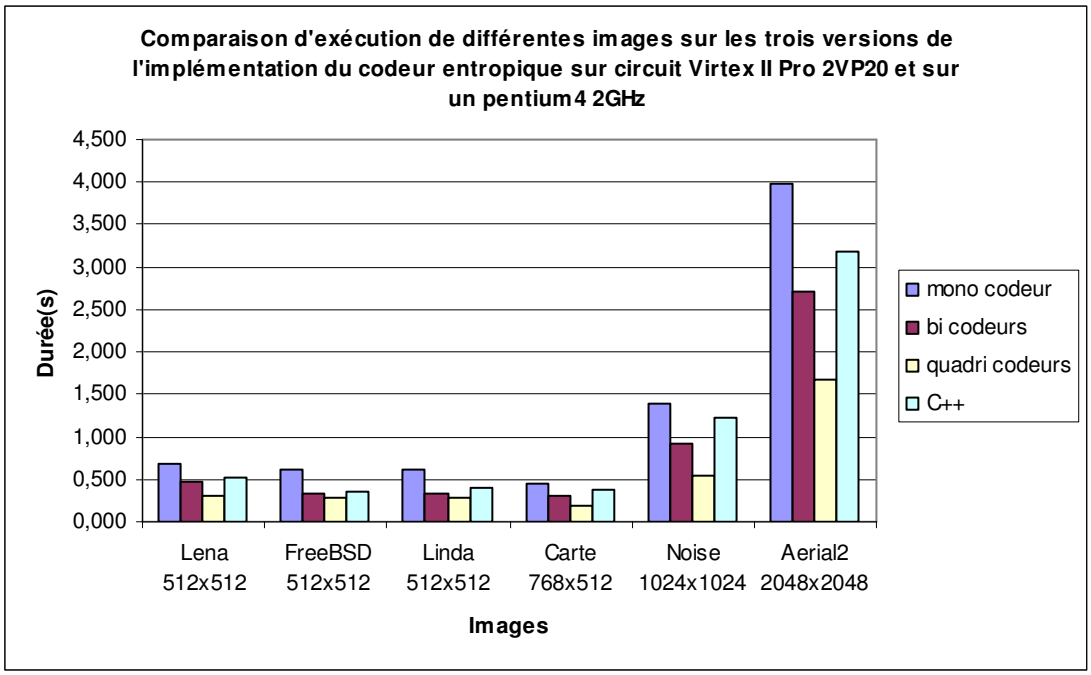


Figure Ch VI - 25 : Comparaison des performances des différentes versions du codeur HW avec celles du SW sur le P4@2GHz

Dans cette partie aussi et pour les mêmes raisons que pour la première partie l'intégration dans une chaîne complète de codage JEPG2000 s'exécutant sur le PowerPC n'a pas pu être réalisée bien que le XC2VP20 possède suffisamment de ressource pour le faire.

L'exploitation d'un seul des deux PowerPC. Mais l'utilisation des deux PowerPC n'est pas importante vu qu'on n'implémente pas la chaîne de codage JPEG2000 et la charge du PowerPC utilisée n'est pas très significative. En plus l'utilisation du second PowerPC aurait pu créer un problème d'accès du bus PCI pour le chargement des code-blocks et l'envoi des bitstream vers la machine hôte.

Le plus de cette partie de l'étude, est que l'intégration des différentes implémentations du codeur était faites à l'aide d'une chaîne de codage qui s'exécute sur la machine hôte.

6.3 Consommation d'énergie

Pour les composants Virtex-II-Pro la consommation d'énergie est liée à deux éléments du circuit qui sont de natures différentes : le processeur PowerPC qui est un circuit ASIC, et la logique reconfigurable. Pour le processeur la consommation d'énergie est donnée par le constructeur elle est de 0.9 mW/MHz. Par contre l'énergie dissipée par la partie reconfigurable dépend de la fréquence certes, mais aussi de l'utilisation des ressources telles que le nombre des Slices, les BRAMs, et les autres ressources(DCM, RocketIO, RAM Distribue, les entrées/sorties.)

7. Conclusion

Dans ce chapitre on expose un travail d'exploration de l'implémentation du codeur entropique de JPEG2000. Cette exploration a été réalisée en deux étapes. Une première étape a été réservée au partitionnement SW/HW. Cette étape était l'occasion de découvrir le concept des composants Virtex-II-Pro qui combinent la logique reconfigurable avec un ou plusieurs cœurs de processeur. Elle a permis de mesurer les variations des performances d'un programme complexe tel que le codeur entropique sur la plateforme Virtex-II-Pro. En implémentant les 4 versions du codeur on a pu découvrir les problèmes liés au couplage d'une

partie logicielle avec du matériel. Et que le coprocessing peut se faire dans le processeur lui-même avec la possibilité de reconfiguration.

Malgré les lacunes de cette étape telles que les tests sur des code-blocks et non sur des images complètes et l'utilisation du bus OPB pour la connexion des modules matériels, ce travail était une confirmation de la complexité de la partie FC du codeur entropique. Il a permis de déterminer les versions les plus performantes parmi les quatre possibilités, à savoir la version 2 et 4. De ce fait cette étude fournit une aide pour l'implémentation de la chaîne de codage JPEG2000 sur un système embarqué dont ont connaît les ressources et les limites.

La deuxième étape représente la continuité de la première. Elle reprend la version la plus performante de la première étape pour l'améliorer. Cette amélioration se résume dans la mise en parallèle de plusieurs codeurs. En effet 3 versions ont été implémentées : une version mono codeur, une version bicodeur et une version quadri-codeur. La reprise de la version avec un seul codeur est due au fait que l'architecture a été adaptée pour être connectée au bus PLB au lieu du bus OPB et que le codeur lui-même a pu être amélioré de point de vue performance et consommation de ressource. Le faite que la deuxième plateforme de développement (ADM-XPL) utilise le bus PCI, a facilité le test et la validation des différentes versions par intégration dans une chaîne de codage JPEG2000. En plus ces tests ont été effectués sur plusieurs images avec des tailles et des formats différents.

L'impact de l'implémentation des codeurs en parallèle ne donne pas nécessairement le speed up attendu (2 pour le bi codeur et 4 pour le quadri codeur) vu l'influence de la connexion avec processeur. Ce dernier impose un certain débit des données en entrée et en sortie des codeurs. Plus le nombre de codeurs connectés sur le PLB est grand plus le programme de gestion des codeurs est compliqués.

En plus de l'exploration, ce travail a permis d'avoir une idée sur les systèmes embarqués qui disposent de la logique reconfigurable. Le coprocessing de proximité permet d'améliorer les performances d'un système embarqué de manière très significative, en gardant la consommation d'énergie presque constante.

8. Références

[1] Katherine Compton, Scott Hauck, “Reconfigurable Computing: A Survey of Systems and Software”, ACM Computing Surveys (CSUR) Volume 34, Issue 2, Pages: 171 – 210, June 2002

[2] Stephen Brown and Jonathan Rose “Architecture of FPGAs and CPLDs: A Tutorial," *IEEE Design and Test of Computers*, Vol. 13, No. 2, pp. 42-57, 1996.

[3] Embedded system tool guide http://www.xilinx.com/ise/embedded/edk_docs.htm

[4] User core templates reference design http://www.xilinx.com/ise/embedded/edk_docs.htm

[5] Virtex-II-Pro and Virtex-II-ProX Platform FPGAs: Complete Data Sheet http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp

[6] ADM-XPL Xilinx Virtex-II Pro reconfigurable computer http://www.alpha-data.com/adm-xpl.html

[7] www.insight.fr.**memec**.com

Chapitre VII Analyse théorique

1. Introduction aux Réseaux de Petri

Les Réseaux de Petri (RdP) sont un outil graphique et mathématique pour la modélisation de plusieurs systèmes. Il sont généralement utilisés pour étudier les systèmes concurrents asynchrones, distribués, parallèles, non déterministes, et /ou stochastiques. Historiquement le concept du RdP trouve ses origines dans la thèse de Carl Adam Petri présentée en 1962 à la faculté de mathématique et physique de Darmstadt, en Allemagne de l'ouest **[34]**.

Les définitions concernant les RdP sont de deux sortes:

- des définitions portant sur l'aspect structurel d'un réseau : quelles sont les actions, quels sont les sites, comment définir l'état d'un système représenté par un réseau, quelles sont les conditions pour qu'une action soit possible, quelles sont les conséquences d'une action.
- Des définitions portant sur l'aspect comportemental d'un réseau : nous présentons le fonctionnement du réseau, autrement dit nous expliquons ce qui se passe quand une action est exécutée et quand plusieurs actions sont exécutées simultanément.

1.1 Aspect structurel

Intuitivement un RdP est donné par des objets de quatre types : les places qui correspondent à des sites, le marquage d'une place (représentant l'état d'un site) est un nombre pouvant indiquer la satisfaction de conditions, ou plus généralement le nombre de ressources présentes dans le site, les transitions qui correspondent aux actions, la fonction de transition donne pour chaque transition les conditions qui doivent être remplies pour chacun des sites afin que cette action soit possible. Elle indique aussi l'effet de chaque transition sur l'état des sites.

Plus formellement nous avons les définitions suivantes :

Définition 1 :
Un réseau de Petri R est défini par R=(P, T, W), ou :
P est un ensemble fini $\{p_1, p_2, \ldots, p_m\}$ de places,
T est un ensemble fini $\{t_1, t_2, \ldots, t_n\}$ de transitions,
$W : (P \times T) \cup (T \times P) \rightarrow N$ est la fonction de valuation
Un marquage M est une fonction de P dans N.
Un réseau de Petri marqué est défini par $R_m=(R, M_0)$, ou R est un réseau de Petri, et M_0 est un marquage appelé marquage initial.
Un réseau de Petri peut être également vu comme un graphe bipartite, l'ensemble des sommets étant composé de places et de transitions. $G(u)=\{v \in T \cup P / W(u,v) \neq 0\}$ et $G^{\sim}(u)=\{v \in T \cup P / W(v,u) \neq 0\}$ sont les ensembles des successeurs et des prédécesseurs de u.
La fonction Pré est la restriction de la fonction W à PxT et la fonction Post la restriction de la fonction W à TxP.

Si $M_0(p)=k$, on dit que la place p contient k marques ou que le marquage de p est k. la fonction Pré donne les conditions sur les sites pour que l'exécution d'une action soit possible, et la fonction Post indique l'effet d'une action sur un site, i.e W(p,t)=k signifie que l'action t utilise k ressources dans le site p ; de façon duale, la fonction Post indique l'effet de l'exécution d'une action sur les sites, i.e W(t, p)=k signifie que l'action t crée k ressources dans le site p.

Il est parfois commode de pouvoir, soit renommer les transitions, soit amalgamer certaines d'entre elles pour mettre en évidence une interprétation commune. D'où l'introduction des RdP étiquetés.

Définition 2 :
Un réseau de Petri étiqueté est défini par (R, h) ou :
R est un RdP
H : T→A∪{ε} avec A alphabet de la fonction d'étiquetage, qui s'étend de façon naturelle en un morphisme de T* dans A*, (i.e $h(u_1 \ldots u_n)=h(u_1) \ldots h(u_n)$), L'étiquetage d'une suite de transitions est la suite des étiquettes.

Remarque :

L'étiquetage correspond à la signification donnée aux transitions. On utilise en général la même étiquette pour plusieurs transitions pour indiquer qu'elles correspondent à un même évènement. Une transition est étiquetée par **ε** quand on veut l'ignorer.

1.2 Représentation des RdP

1.2.1 Représentation graphique

L'un des aspects les plus agréable des RdP est qu'ils sont extrêmement aisés à visualiser. En effet on peut représenter un RdP comme un graphe bipartite.

- Les places sont représentées par des ronds
- Les transitions par des rectangles

Soit (u, v) ∈ TxP∪PxT. Si W(u, v)>0, il y a un arc de u vers v, value par W(u, v), (si W(u,v)=1, la valuation 1 est omise). Ce graphe représente la structure du système.

Enfin, le marquage initial est représenté à l'aide de marques (on parle aussi de jetons) disposées dans les places.

1.2.2 Représentation matricielle

Il est possible de représenter par des matrices la fonction W. pour cela on numérote les places et les transitions i.e $P=\{p_1,\ldots, p_n\}$ et $T=\{t_1,\ldots, t_p\}$. Aux fonctions Pré et Post correspondent deux matrices **n**x**p** à coefficients dans N. Pré(i, j) indique le nombre de marques que doit contenir la place p_i pour que la transition t_i soit franchissable. De façon duale, Post(i, j) contient le nombre de marques déposées dans la place p_i lors du franchissement de t_j. Post (i, j)-Pré(i, j) donne la modification pour la place p_i résultant du franchissement de t_j. Un marquage M sera alors représenté par un vecteur M~ de dimension n à coefficients dans N.

Définition 3 :
Soit R=(P,T,W) un RdP avec $P=\{p_1,\ldots,p_n\}$,et $T=\{t_1,\ldots,t_p\}$.
On appelle matrice de prècondition Prè la matrice **n** x **p** à coefficients dans N définie par $Prè(i,j)=W(p_i,t_j)$,
On appelle matrice de postcondition Post la matrice **n** x **p** à coefficients dans N définie par $Post(i,j)=W(t_j,p_i)$.
La matrice C=Post-Prè est appelée matrice d'incidence du réseau.

La définition suivante traduit le fait qu'une action est possible à partir d'un état du système dès lors qu'il y a suffisamment de ressources dans tous les sites concernés par cette action et que dans certains sites l'action utilise ou crée des ressources

Définition 4 :
Soit R=(P,T,W) un RdP, et t une transition. On dit que la transition t est franchissable pour M quand :
$\forall p \in P$, $M(p) \geq W(p,t)$ on note ceci M(t>. si t est franchissable pour M, on dit que le franchissement de t fait passer de M à M' quand : $\forall p \in P$, $M'(p)= M(p)-W(p,t)+W(t,p)$. on note ceci M(t>M'

Définition 5 :
Soit x une séquence de transitions. La séquence x est franchissable à partir de M et conduit a l'état M' ce qui sera noté M(x>M' ssi :
Si $x=\varepsilon$ M'=M
Si $x \neq \varepsilon$, $x=t_1 \ldots t_n$
Il existe une suite de marquages $M=M, M_1, \ldots, M_n=M'$ tels que $\forall$ i=0…n $M_i(t_i+1>M_i+1$.

La notion de séquence de franchissement amène naturellement à deux définitions : celle de langage associé à un réseau qui est celui des séquences franchissables ce qui correspond à l'ensemble des comportements ou évolutions possibles du système ; et celle des marquages accessibles via une séquence franchissable, qui traduit le fait qu'il existe une évolution du système aboutissant à une configuration.

Définition 6 :
Soit (R, M_0) un réseau marqué.
$L(R,M_0)=\{x \in T^*/M_0(x>\}$. $L(R,M_0)$ est le langage des séquences franchissables ou langage du réseau. Soit (R,M_0,h) un réseau marqué étiqueté. $L(R,M_0,h)=\{h(x), x\ L(R,M_0)\}$ est le langage du réseau étiqueté
Un marquage M est dit accessible à partir du marquage M_0 ssi il existe une séquence x de T* telle que $M_0(x>M$.
$ACC(R,M_0)$ est l'ensemble des marquages accessibles à partir de M_0 i.e l'ensemble d'accessibilité de R_m

L'ensemble d'accessibilité décrit l'ensemble des états possibles du système.

Définition 7 :
Un marquage accessible, est un blocage, quand aucune transition n'est franchissable à partir de M

Une situation de blocage indique que le système ne peut plus évoluer

1.3 Propriétés comportementales

Les RdP servent à modéliser des systèmes parallèles. Ces systèmes se doivent de répondre a certaines contraintes (par exemple, il peut être impératif que le système ne s'arrête jamais i.e qu'il y ait régime permanent ou bien que certaines actions ne soient jamais définitivement ineffectuables).

Définition de quelques propriétés comportementales :

Soit $R=(P,T,W,M_0)$, un RdP.

Quasi-vivacité : il s'agit de savoir si il existe une évolution du système permettant l'exécution d'une action donnée. R est t-quasi-vivant ssi $\exists u \in T^*$ telle que $M_0(ut>$. On dit aussi que M_0 est t-quasi-vivant. Q(R,t) est l'ensemble des marquages initiaux t-quasi-vivants.

Vivacité : on peut savoir si quelle que soit l'évolution du système on peut toujours aboutir à l'exécution d'une action donnée. R est t-vivant ssi $\forall\ x \in T^*$, telle que $M_0(x>$, alors $\exists u \in T^*$ telle que $M_0(xut>$. V(R,t) est l'ensemble des marquages t-vivants. Si le réseau est t-vivant pour toutes les transitions, on dit qu'il est vivant.

Accessibilité : on cherche s'il existe une évolution du système conduisant à une configuration donnée. Un marquage M est accessible ssi $\exists\ x \in T^*$ telle que $M_0(x>M$. Acc(R,M_0) est l'ensemble des marquages accessibles à partir de M_0.

Réseau borné : on cherche si le nombre de configurations possibles du système est fini. Le réseau est borné si l'ensemble des marquages accessibles est fini. UB(R) est l'ensemble des marquages initiaux rendant le réseau non borné.

Terminaison finie : on cherche si le système évolue quel que soit son comportement vers une situation de blocage. Un réseau est à terminaison finie si le langage L(R,M_0) est fini ce qui revient à dire que $L\omega(R,M_0)$ est vide. NB(R) : est l'ensemble des marquages initiaux tels que le réseau ne soit pas à terminaison finie i.e tel qu'il existe des comportements infinis.

Dans le tableau suivant on donne quelques interprétations des places et des transitions d'un RdP.

Tableau Ch VII - 1 : Interprétation des RdP

Places d'entrées	Transition	Places de sorties
Prèconditions	Évènement	Postconditions
Données d'entrée	Traitement	Donnée de sortie
Signaux d'entrée	Processeur de signal	Signaux de sortie
Ressources nécessaires	Tache	Ressources libérés
Buffers	Processeur	buffers

1.4 Extension des RdP

Nous présentons dans cette partie les modifications des règles de fonctionnement des réseaux. Ces règles augmentent la puissance des réseaux, c'est-à-dire que certaines opérations impossibles à réaliser par les RdP peuvent être effectuées grâce à l'emploi de ces règles. Le prix à payer est que certaines propriétés qui étaient décidables pour les RdP deviennent indécidables pour ces nouveaux réseaux.

1.4.1 Réseaux à arc inhibiteurs

Comme les RdP à l'origine ne peuvent pas tester si une place ne contient pas de marque, le concept des arcs inhibiteur a été introduit pour permettre la modélisation des systèmes qui nécessite ce test. Dans le cas d'un arc inhibiteur la transition t en sortie de la place p ne peut être franchie que si p est vide. Voir l'exemple de **[42]**.

1.4.2 Réseaux avec priorités

Les réseaux avec priorité sont utilisés quand le choix entre plusieurs transitions franchissables est imposé. Par exemple si plusieurs processus ont besoin d'une même ressource, une stratégie possible consiste à attribuer cette ressource au processus le plus lent qui en a besoin

1.4.3 RdP temporisé

Le concept du temps n'était pas supporté par la définition originale des RdP. La nécessité d'évaluer les performances, de résoudre les problèmes d'ordonnancement des systèmes dynamiques, a imposé l'introduction de la notion du temps sur les RdP. Cette notion peut être liée aux places et/ou aux transitions. Parmi les approches de représentation du temps dans un RdP on trouve celle de Merlin où il associe deux temps a et b avec $0 \leq a \leq b \leq \infty$. Si une transition est active a l'instant θ elle est franchissable à partir de l'instant $\theta+a$ et doit être franchie avant l'instant $\theta+b$. L'approche de Ramchandani où l'on associe une durée à une transition. Cette transition est franchissable des qu'elle est active **[5]**.

1.4.4 RdP stochastique

Ce sont des RdP temporisés pour lesquels la durée associée à une transition, est une variable aléatoire continue ou discrète positive qui possède une distribution exponentielle. Dans le cas continu la distribution est donnée par $FX(x)=P[X \leq x]=1-Exp(-\lambda.x)$ où λ est la fréquence de franchissement de la transition t à laquelle est associé X.

1.4.5 RdP coloré

Les RdP colorés sont une représentation plus compacte que les RdP conventionnels. Dans les RdP colorés les jetons représentent des types simples (entier, réel, caractères,…) ou complexes (structure). Dans **[39]** on trouve la définition formelle des RdP colorés. Ils permettent une meilleure clarté et compréhension de la représentation d'un système.

En plus de ces extensions des RdP, on trouve plusieurs autres RdP hybrides. En effet on trouve des RdP qui combinent les caractéristiques de ces extensions tel que l'utilisation du temps avec les RdP colorés, ou les RdP stochastiques colorés,….

1.5 Les outils pour la manipulation des RdP

Plusieurs outils, dont une majorité développée dans le cadre universitaire, permettent de simuler l'évolution des marques d'un RdP, de vérifier des propriétés d'un RdP, de déterminer les matrices d'incidences, et de déterminer le graphe de marquage, et la synthèse de RdP en circuits.

Parmi ces outils ont trouve Design/CPN développé par l'université Aarhus au Denmark. Il permet la saisie graphique et la simulation des RdP colorés avec ou sans temporisation. Il permet aussi la hiérarchisation des RdP et la construction du graphe de marquage.

L'outil TimeNet développé par le groupe d'évaluation de performance de l'université technique de Berlin (Technische Universität Berlin) **[40]**. C'est un outil pour la modélisation et l'analyse des RdP stochastiques. Xpetri est un outil basé sur une extension des RdP stochastiques. L'outil Petrify permet la synthèse et l'optimisation des RdP en circuits de contrôle asynchrones. D'autres outils sont présentés dans **[41]**.

1.6 Exemple

Dans cet exemple on donne une représentation graphique et matricielle d'un RdP simple qui contient 4 places et trois transitions (**figure Ch VII - 1**)
Représentation graphique

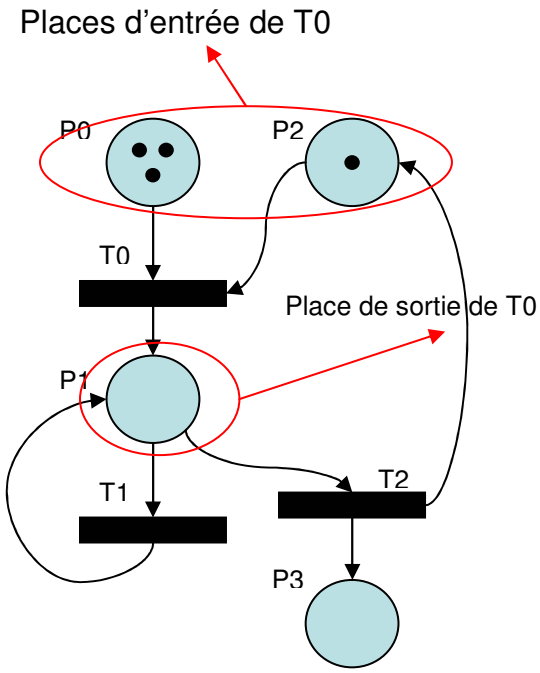


Figure Ch VII - 1 : exemple d'un RdP

Représentation matricielle :

C : étant la matrice d'incidence

$$Post = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad Pre = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad C = \begin{pmatrix} -1 & 0 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Dans la figure ci-dessous on donne le graphe d'état correspondant au RdP précédent. Ce graphe d'état montre que le RdP de cet exemple peut être soit en boucle infinie soit atteindre un état de blocage. La boucle infinie est activée chaque fois que la place P1 contient un jeton.

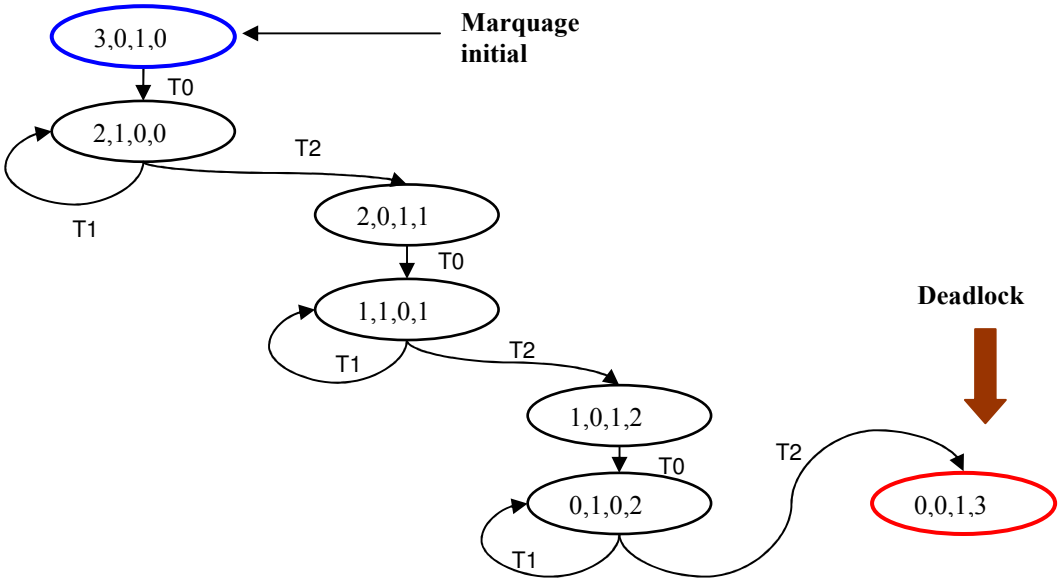


Figure Ch VII - 2 : Graphe d'état

2. Modélisation du codeur entropique

Comme les RdP sont capables de représenter le parallélisme, de vérifier différentes propriétés d'un système ayant un fonctionnement concurrent, et de mesurer les performances, on a choisi de les utiliser comme un moyen de conception pour le codeur entropique en vue de déceler les possibilités de parallélisation au niveau macro (plusieurs passes de codage en parallèle) et micro (plusieurs instructions en parallèle). En effet dans la littérature on trouve plusieurs travaux qui utilisent les RdP pour l'étude et l'analyse des systèmes temps réel ainsi que les mesures des performances de systèmes parallèles **[2]**, **[3]**, **[7]**, **[8]**, **[18]**, **[35]**.

Les RdP sont un moyen très simple et efficace pour faire de la modélisation des SOC à un niveau d'abstraction beaucoup plus élevé que les techniques basées sur des langages de conception tel que SystemC. Ce qui est encourageant aussi c'est qu'il y a des travaux qui se font en parallèle pour pouvoir faire la synthèse de circuits à partir de la modélisation en RdP, ou la transformation d'un modèle RdP en code SystemC comme dans **[38]**, ou en VHDL comme dans **[46]**. Dans **[9]**, **[16]**, **[24]**, **[28]** on trouve plusieurs façons selon lesquelles les RdP ont été utilisés pour la description et la synthèse de certains modules de contrôle. Les RdP ont été aussi utilisés comme un moyen de conception pour faciliter l'intégration des IP dans les systèmes SOC **[32]**, **[47]**. Ces derniers papiers mettent donc en avant le problème de la réutilisation dans la conception SOC. Dans **[22]**, les RdP représentent un niveau de description intermédiaire entre le VHDL et les systèmes de transactions symboliques (Symbolic Transaction Systems) pour la vérification formelle d'un système. Ceci nous rappelle l'utilisation de la conception au niveau TLM (Transaction Modeling Level) pour faciliter la validation des SOC.

Plusieurs autres problèmes liés à la conception SOC ont fait l'objet d'une utilisation des RdP comme moyen de conception, de vérification, et aussi comme modèle de performance. Dans **[23]** c'est le problème d'équilibrage de charge dans un système embarqué, qui est étudié. Le papier **[33]** présente et étudie le problème d'ordonnancement à l'aide des RdP. Le partitionnement HW/SW a été présenté dans **[43]**. Le problème de la consommation d'énergie a été analysé dans **[43]**.

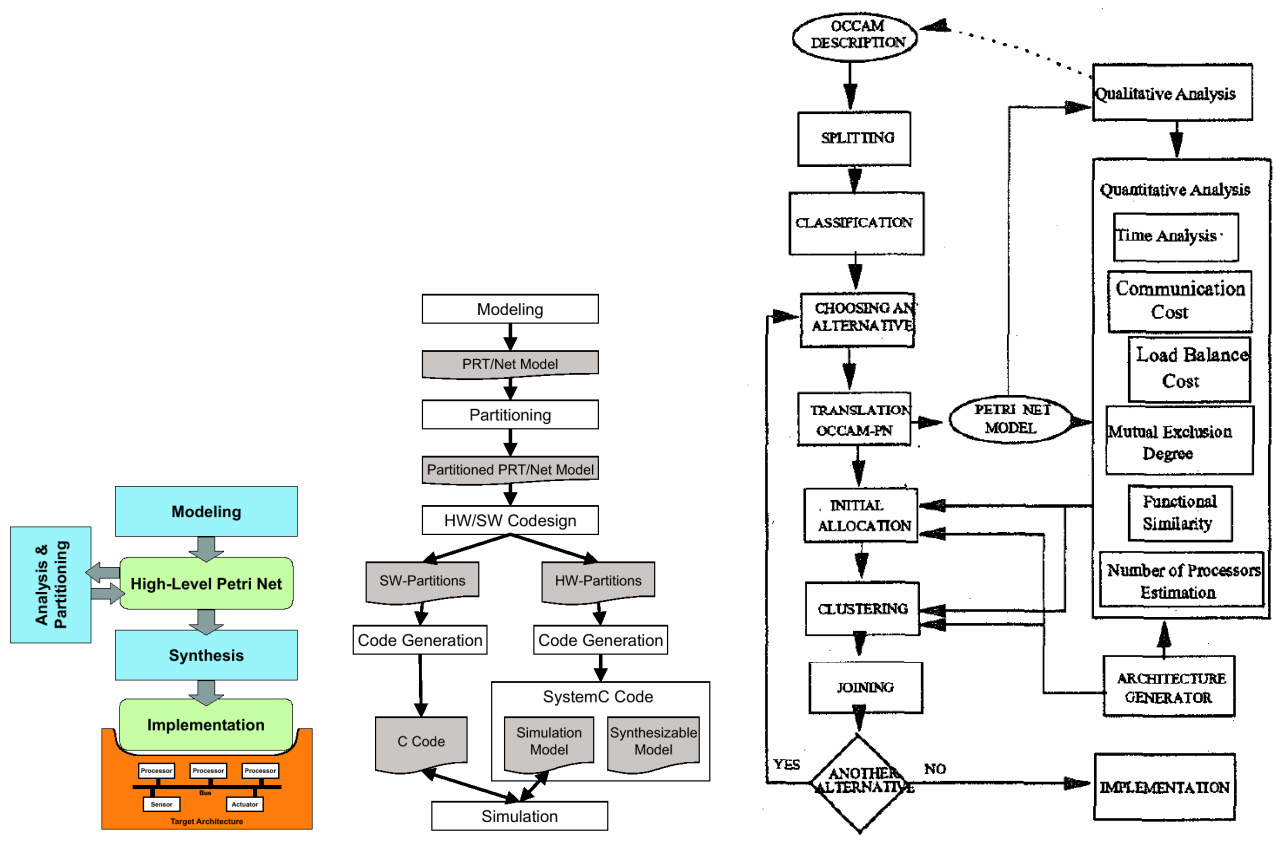
Selon la nature du système à étudier, différentes variétés de RdP sont utilisées pour la modélisation des SOC. Ces RdP peuvent être hiérarchiques, **[20]**, plus ou moins compacts **[10]**, **[21]** et souvent avec une adaptation pour un type particulier de systèmes (systèmes embarqués **[15]**, systèmes multiprocesseur à mémoire distribuée **[45]**). Cette adaptation suit aussi la nature de l'étude à mener.

Comme tout modèle de calcul (model of computation) **[11]** les propriétés intrinsèques des RdP permettent une grande liberté pour la modélisation mais des limitations peuvent être imposées par le système à modéliser.

Le codeur entropique a la spécificité d'avoir une forte dépendance de données ce qui fait que la partie de contrôle des modules du codeur entropique sera définie par le flot de donnée. Cette spécificité du fonctionnement va être le premier grand problème dans la démarche de modélisation à l'aide des RdP. En effet un certain nombre de types de RdP seront inutilisables ou difficilement applicables dans le cadre de cette étude tels que les RdP utilisés par l'outil Petrify qui sont des graphes de transition qui représentent la partie contrôle d'un système sans la possibilité de représenter l'interaction avec le flot de données. Par contre ce même outil permet la synthèse d'une netlist à partir d'un graphe de transition.

On remarque aussi que l'ensemble des études présentées ci-dessus ont traité les problèmes de la conception SOC de manière indépendante les uns des autres. Chacun de ces problèmes a été traité selon des méthodologies différentes, en déployant des outils et des types de RdP

différents (faits sur mesure pour certains cas). Les figures ci-dessous présentent quelques exemples de méthodologies pour étudier différents problèmes tels que l'équilibrage de charge dans **[23]**, la génération et du code SystemC à partir d'un RdP **[38]**, et le partitionnement HW/SW **[43]**.



(a) (b) (c)

Figure Ch VII - 3 : (a), (b), (c) : Trois méthodologies pour l'utilisation des RdP

Notre approche vise à réaliser un assemblage de toutes ces méthodologies pour réunir l'ensemble des problèmes liés à la modélisation SOC derrière un seul ensemble de modèles RdP. Cet ensemble de modèles doit présenter une diversité suffisante et des caractéristiques nécessaires pour traiter séparément ou en groupe les problèmes de la modélisation SOC. Dans notre approche en plus des problèmes communs à la conception SOC on a voulu introduire la recherche du parallélisme à différents niveaux d'un système (micro, macro). La **figure Ch VII - 4** décrit le flot souhaité.

La première spécification est celle provenant d'une norme de type ISO comme pour JPEG2000. Cette norme utilise le langage naturel accompagné éventuellement de figures et de schémas descriptifs des algorithmes dans leurs versions fonctionnelles non optimisées (séquentiel). Le principe de base est alors de débuter par la modélisation par un RdP simple et ce pour plusieurs familles de RdP. L'étape suivante est alors d'appliquer de manière exploratoire et automatique un ensemble de règles de réécritures sur les RdP obtenus dans l'étape précédente et sous contraintes de ressources en communication ou en calcul obtenues soit par un fichier d'intervalles de contraintes défini par l'utilisateur soit par rétro-annotation de la partie back-end du flot. Ainsi notre expérience dans les différentes implémentations nous a permis de définir le type de contraintes pouvant apparaître lors de la phase d'implémentation et qui peuvent être donc intégrées au plus haut niveau. Le premier résultat est alors un ensemble de solutions au sens de Pareto pouvant couvrir le spectre parallélisme

maximal-ressources maximal-consommation d'énergie maximale en parallèlisme minimal-ressources minimales- consommation d'énergie minimale. En se basant sur des méthodes telles que celles présentées dans **[22]** et **[38]** pour la génération du code on essayera de faire la synthèse des différents modèles pour obtenir des versions SystemC. Le langage SystemC **[48-50]** est le standard de facto pour les spécifications exécutables pour la modélisation système. Les modèles SystemC seront alors synthétisés de manière exploratoire comme décrit dans **[53]** en se basant sur l'outil Synopsys CoCentric SystemC Compiler **[51-52]**. L'implémentation utilisera en entrée le code VHDL généré pour implémenter le circuit et fournir des contraintes de rétro-annotation pour le front-end.

La tâche la plus complexe dans ces différentes étapes est la création des règles de réécriture des RdP car c'est elle qui déterminera la qualité des résultats obtenus en post synthèse.

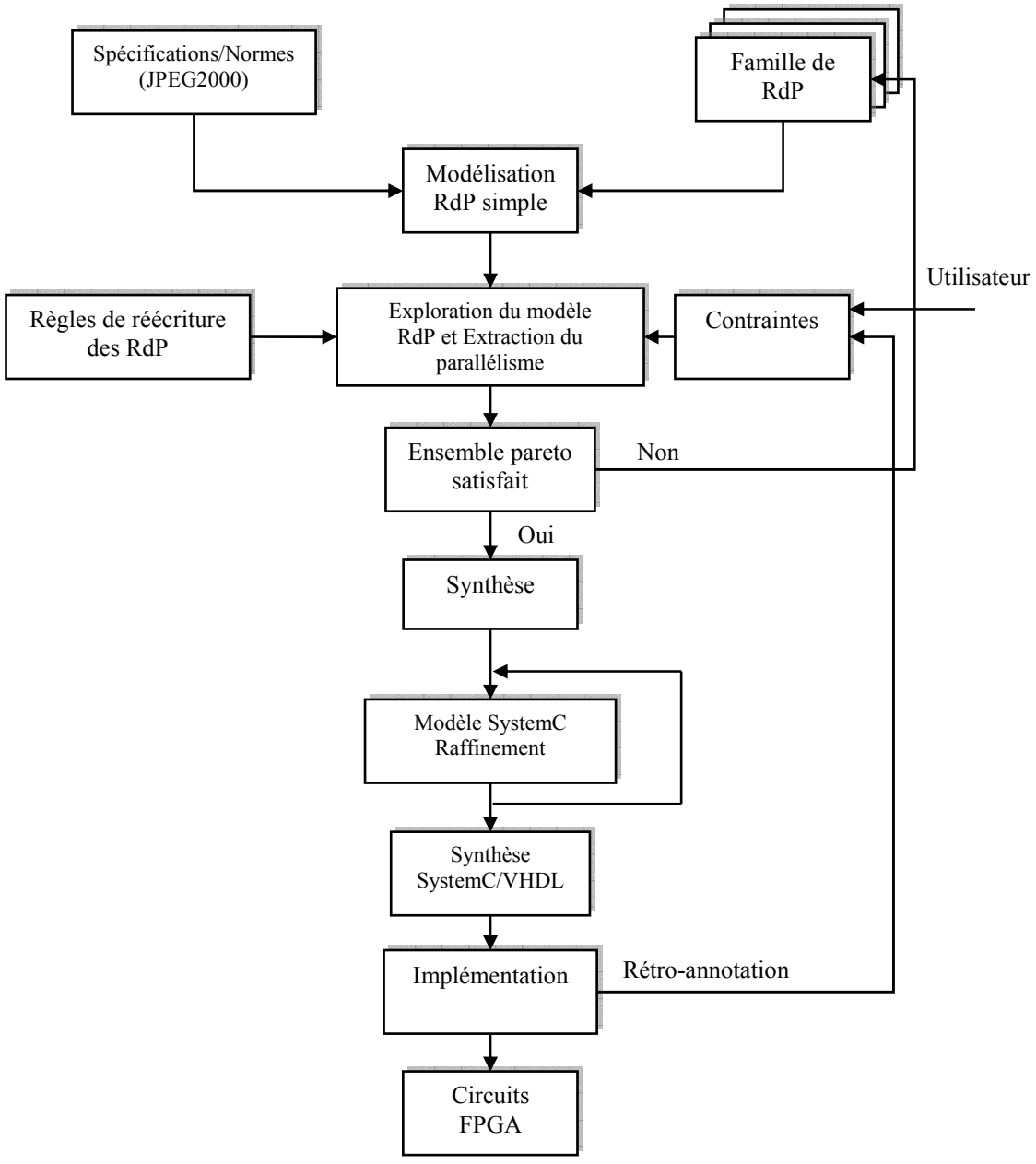


Figure Ch VII - 4 : Méthodologie générale pour l'utilisation des RdP

Pour faciliter la représentation du flot de données on a choisi comme type de RdP les RdP colorés, vu leur compacité et leur pouvoir d'expression. Ce type de RdP a été manipulé à

l'aide de l'outil Design/CPN qui supporte la saisie, la simulation et l'analyse de quelques propriétés des RdP colorés **[39]**.

2.1 L'algorithme globale

La modélisation du codeur entropique peut se faire de deux manières : soit une modélisation hiérarchique du type Top-Down ou en procédant à l'inverse en commençant par le niveau le plus bas (bottom-up).

Dans un premier lieu on a essayé de voir le modèle global du codeur entropique qui nous a permis d'avoir une première idée sur le comportement du codeur cette idée rejoint la constatation faite lors de la conception VHDL du codeur à savoir que c'est le troisième passe de codage qui peut être handicapant pour le fonctionnement du codeur.

La modélisation du codeur entropique part de l'algorithme de base :

Pour chaque Code-Block faire :
(1) *Initialisation,*
(2) *Pour chaque bit-plane du code block faire :*
Si *bit-plane≠premier bit-plane :*
(a) *Premier passe de codage,*
(b) *Deuxième passe de codage*
Sinon
(c) *Troisième passe de codage.*

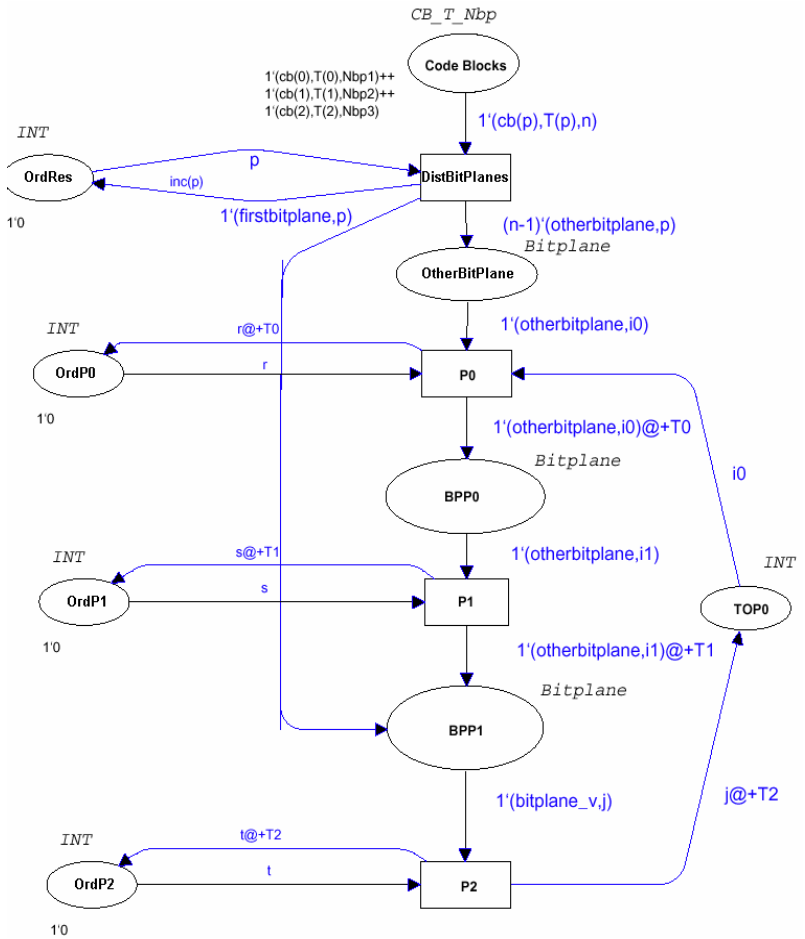


Figure Ch VII - 5 : le codeur entropique

Le modèle de cet algorithme est représenté dans la **figure Ch VII - 5**. En effet, une transition est attribuée à chaque passe de codage (P0, P1, P2). La condition pour la séparation entre les premiers bit-planes et le reste des bit-planes a été réalisée à l'aide de la transition « DistBitPlanes ». La place « Code Blocks » contient l'ensemble des code-blocks à traiter. Chaque code-block est défini par le triplet (cb(i), T(i), Nbpi) où i est l'indexe du code-block cb(i), T(i) sa taille, et Nbpi le nombre de bit-plane qui le constitue. Dans notre cas, seul trois code-blocks sont représentés avec Nbp1=Nbp2=Nbp3=2.

Le choix du nombre des code-blocks ainsi que le nombre de bit-plane par code-block est totalement arbitraire. Les places « OtherBitPlane », « BPP0 », « BPP1 », représentent les ressources (les bit-planes) à traiter par chaque passe de codage. Pour chaque code-block le premier bit-plane est mis directement dans « BPP1» et le reste des bit-planes dans « OtherBitplane ». Une place « OrdRes » a été prévue pour ordonnancer le traitement des code-blocks. Elle permet aussi de varier cet ordonnancement surtout lorsque les code-blocks ont des nombres de bit-planes différents.

Les places : « OrdP0 », « OrdP1 », « OrdP2 » permettent d'avoir une simulation temporelle conforme à la réalité : ne pas traiter un nouveau bit-plane que si la tâche est libre. Ces dernières n'ont aucune influence sur le comportement du modèle en dehors de la simulation temporelle et peuvent être éliminées. La place « TOP0 » permet de marquer la fin du traitement d'un bit-plane et « P0 » peut commencer le traitement d'un nouveau bit-plane.

Pour conserver le même esprit de la modélisation, c'est-à-dire, s'abstraire de toutes architectures, aucune contrainte n'a été imposée sur les ressources (les places ne sont pas bornées). De plus les ressources sont immédiatement disponibles pour chaque passe de codage. La taille et le format des données ne sont pas précisés dans ce modèle, car ils ne doivent pas influencer le comportement de base du modèle.

Les premières constatations montrent qu'à ce niveau du système, seul un pipeline est envisageable, à condition de savoir assurer une totale indépendance entre les ressources de chaque code-block. En associant une durée d'exécution à chaque passe de codage (T0, T1, T2), plusieurs configurations sont donc possibles. Le cas idéal pour le fonctionnement du pipeline est T0=T1=T2. Si cette condition n'est pas réalisable ou si elle est sous optimale, on peut avoir d'autres configurations qui peuvent influencer sur l'équilibrage de la charge des tâches. Pour tester l'impact de la durée de chaque passe de codage, on effectue l'ordonnancement des trois code-blocks. La simulation montre que le meilleur équilibrage des trois passes de codage est obtenu pour la configuration suivante : T2<T0 et T2<T1.

Cependant, vu la distribution des bits traités par chaque passe de codage (comme le montre la **figure Ch VII - 6** tirée de **[37]**), cette distribution du temps d'exécution ne reflète pas la réalité, mais la condition reste nécessaire pour garder un meilleur fonctionnement du codeur. Pour l'implémentation du codeur, l'approche de **[38]** est insuffisante pour assurer le fonctionnement voulu suite à l'analyse réalisée.

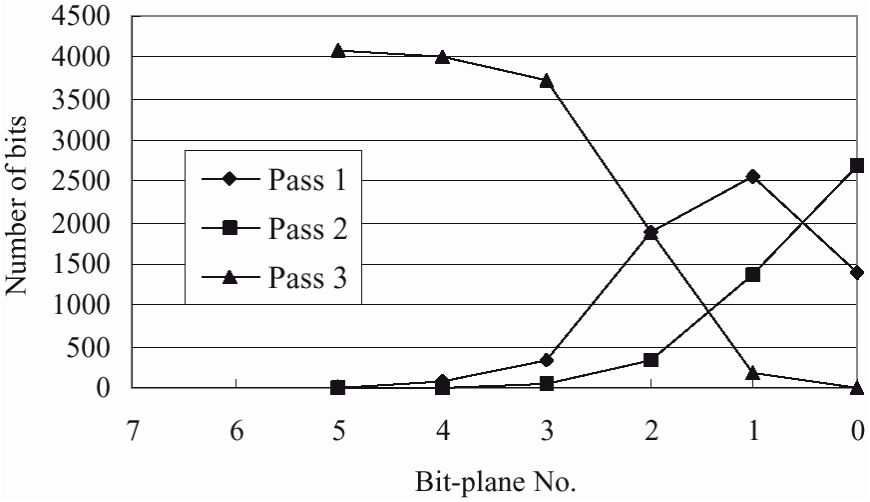


Figure Ch VII - 6 : Distribution des bits traités dans chaque passe de codage

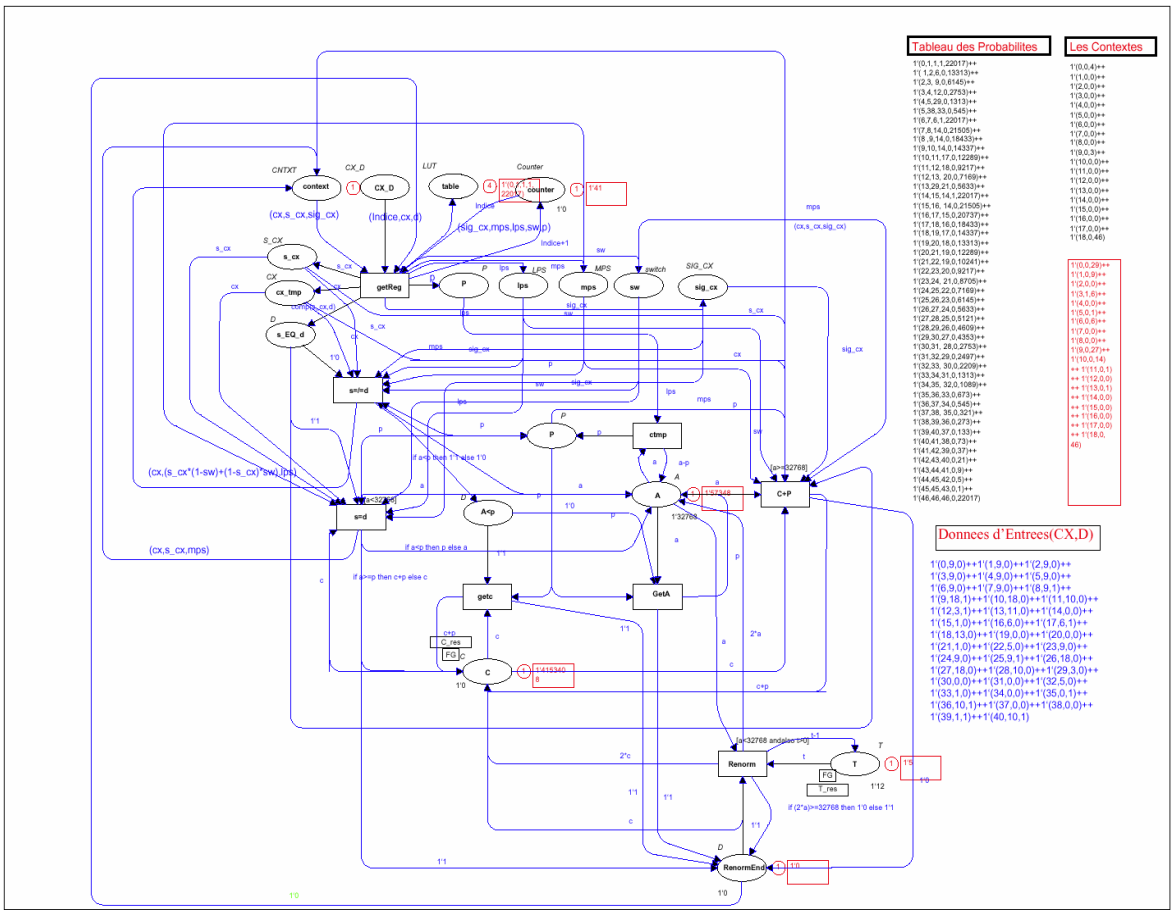
2.2 Le codeur arithmétique

Dans la deuxième étape on a réalisé le modèle fonctionnel du codeur arithmétique. Ce modèle se compose de deux parties qui sont la procédure de codage globale et la fonction pour la génération des données de sortie du codeur arithmétique. Cette décomposition n'est pas hiérarchique mais sous la forme de deux sous modules ayant des ressources en communs représentés par deux places équivalentes à deux registres partagés.

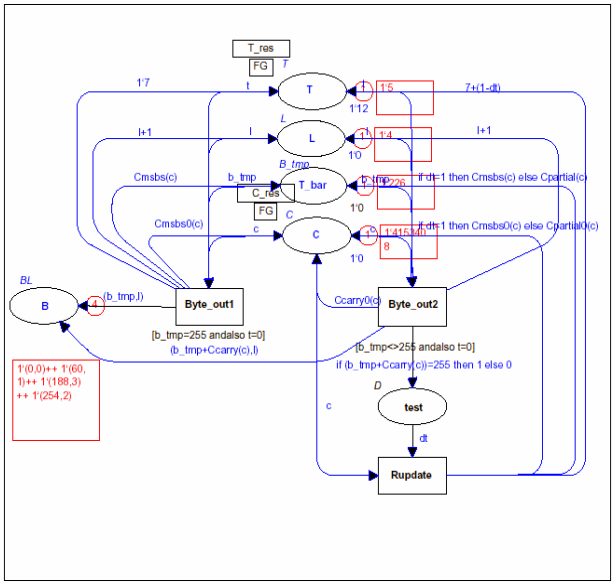
Contrairement à l'utilisation d'un langage de programmation sur les RdP des figures ci-dessous on distingue clairement l'ensemble des ressources nécessaires au fonctionnement du codeur avec les chemins de donnée possibles. Ces ressources sont principalement des places qui représentent soit des registres utilisés pour les calculs intermédiaires soit des mémoires pour mettre les données d'entrées et de sorties

La particularité du modèle est le nombre d'opérations conflictuelles dues aux différents tests ce qui présente un grand nombre de combinaisons de chemins de données. Ceci rend impossible l'utilisation de l'outil Design CPN pour la génération du graphe d'état et l'analyse de ce graphe même pour une faible quantité de données d'entrée, car ce graphe est totalement données-dépendant.

Une vue plus simplifiée du chemin de données est présentée dans la **figure Ch VII - 8**. Cette représentation nous permet de déterminer le graphe d'état (précisément graphe de recouvrement) plus facilement (**figure Ch VII - 9**). Sur la **figure Ch VII - 8** on présente les chemins de données de l'algorithme principal du codeur arithmétique (sans le chemin de la procédure de renormalisation). N représente la quantité de données à traiter. Les transitions sont les équivalents des branchements de l'algorithme de codage arithmétique.



(a)



(b)

Figure Ch VII - 7 : Le codeur arithmétique (a) fonction globale, (b) transfert de données compressées

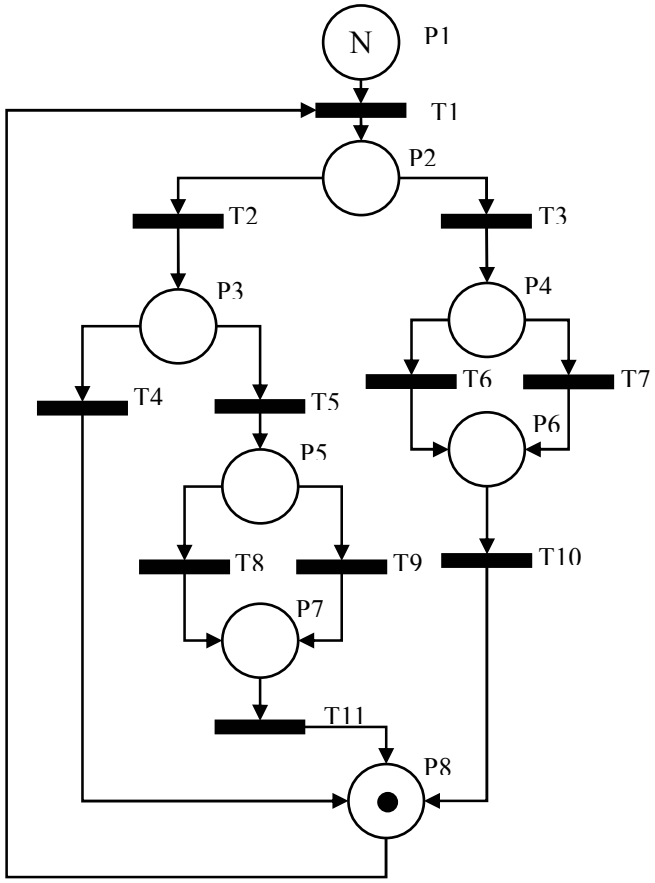


Figure Ch VII - 8 : Simplification du chemin des données

Le graphe de recouvrement de l'algorithme du codeur arithmétique représenté par la **figure Ch VII - 8** confirme que la complexité des branchements dépend des données. Selon la donnée déjà codée et celle qui va l'être le nombre de test à effectuer sera différent. Ce nombre de test peut être déterminé à l'aide de la simulation du RdP de la **figure Ch VII - 7** (a) et (b). L'avantage de ce modèle est qu'il est indépendant de toute architecture matérielle ou du compilateur (aucun processus de prédiction ou d'optimisation de code). Donc ce modèle peut permettre de cibler les points critiques de passage des données.

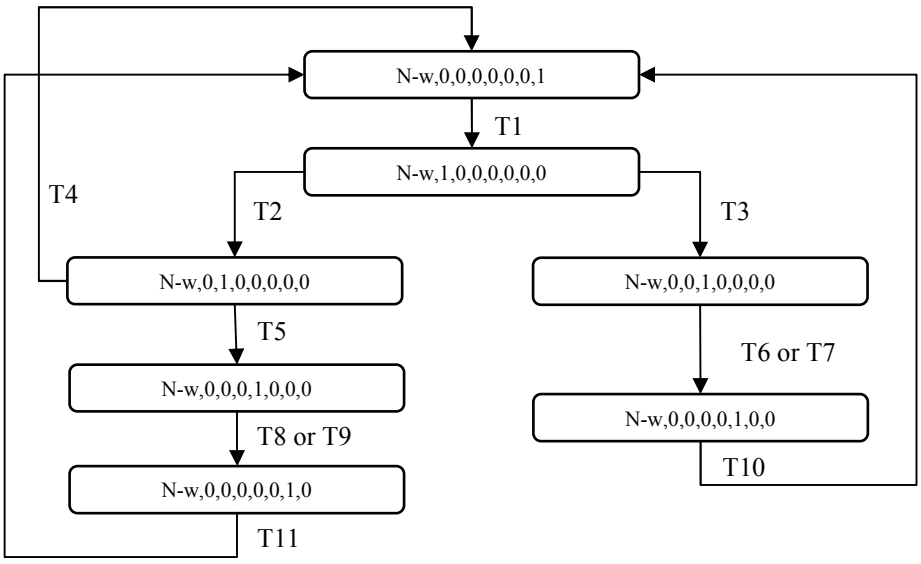


Figure Ch VII - 9 : Graphe de recouvrement

3. Discussion

Bien que le RdP soit un parmi les modèles de calcul très puissant du point de vue formel et représentation, le manque d'outils d'analyse automatique complique la tâche de modélisation et la rend dans certain cas impossible vu la taille des modèles.

La méthodologie qui a été adaptée présente plusieurs lacunes pour lesquelles on ne dispose pas de réponse dans l'immédiat. Parmi ces lacunes on peut citer la façon selon laquelle peut être représenté un modèle (Top-down ou Bottom-up) et son influence sur les performances du modèle ou la famille de modèles. Un autre problème majeur qui peut affecter notre méthodologie est la création des règles de réécriture.

4. Conclusion

L'analyse théorique d'une application dans un modèle de calcul en vue d'une implémentation système sur puce est prépondérante puisqu'elle sert de guide pour l'exploration de différentes implémentations logicielles et matérielles. Le codeur entropique a été ainsi modélisé par réseaux de pétri et a permis une extraction d'un parallélisme temporel entre les trois passes du codeur. Ce parallélisme temporel est difficile à détecter dans des modèles de calcul orientés contrôle plus que données comme le permettent les réseaux de pétri. De plus cette étude est un premier pas pour une nouvelle orientation des méthodologies de modélisation SOPC à un haut niveau d'abstraction avec la possibilité de réaliser des vérifications formelles de notre conception. Ce flot est en cours de mise en œuvre au sein du laboratoire.

5. Références

[1] Nasima F.Shakirova, “a new approach to the analysis of petri nets : parallel processes and predictability of scenarios”, Proceeding of the 9th International Conference on Neural Information Processing (ICONIP'02) November 18-22, 2002, Singapore

[2] A.Ferscha, “a petri net approach for performance oriented parallel program design”, Journal of Parallel and Distributed Computing, Vol. 15: Special Issue on Petri Net Modeling of Parallel Computers, pages 188-206. Jully 1992

[3] Ugo Buy, Robert Sloan, “a petri net based approach to real time program analysis”, International Workshop on Software Specifications & Design Proceedings of the 7th international workshop on Software specification and design, 1993, California

[4] Miguel Felder, Angelo Gargantini, Angelo Morzenti, “a theory of implementation and refinement in timed petri nets”, Proceedings of 1st international conference on Temporal Logic ICTL-94, Bonn, Germany, July 11-14, pp. 365-381,Springer-Verlag, New York, 1994

[5] Takaharu Hirai, “an application of a temporal linear logic to timed petri nets”, Proceeding of Petri Nets’99 Workshop on Applications of Petri nets to intelligent system development, pp.2–13,June 1999

[6] Bernard Berthomieu, Miguel Menasche, “an enumerative approch for analysing time petri nets”, Proceedings of the Information Processing 83, IFIP - 9th World Congress, Paris, September 1983

[7] Gianfranco Balbo, Giovanni Chiola, Steven C.Bruell, “an example of modeling and evaluation of a concurrent program using colored stochastic petri nets Lamport s fast mutual exclusion algorithm”, IEEE Transactions on Parallel and Distributed Systems Vol. 3, No. 2, March 1992

[8] Zonghua Gu, Kang G.Shin, “an integrated approach to modeling and analysis of embedded real time systems based on timed petri nets”, 23rd International Conference on Distributed Computing Systems May 19 - 22, 2003 Providence, Rhode Island

[9] L.Lloyd, K.Heron, A.M.Koelmans, A.V.Yakovlev, “asynchronous microprocessors from high level model to FPGA Implementation”, Journal of Systems Architecture, Vol. 45, No. 12-13, pages 975-1000. 1999

[10] Matthew B.Dwye,r Lori A.Clarke, “a compact petri net representation for concurrent programs”, 17th International Conference on Software Engineering April 23 - 30, 1995 Seattle, Washington, USA

[11] Edward A.Lee, Alberto Sangiovanni-Vincentelli, “comparing models of computation”, International Conference on Computer Aided Design Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design San Jose, California, United States Pages: 234 – 241, 1997

[12] Wei Jen Yeh, Michal Young, “compositional reachability analysis using process algebra”, International Symposium on Software Testing and Analysis Proceedings of the symposium on Testing, analysis, and verification Victoria, British Columbia, Canada Pages: 49 – 59, 1991

[13] Soren Christensen, Jens Baek Jorgensen, Lars Michael Kristensen, “cpn tool for colored petri net”, http://www.daimi.au.dk/designCPN/

[14] Javier Esparza, Mogens Nielsen, “decidability issues for petri nets”, Brics Report Series RS-94-8, May 1994, ISSN 0909-0878

[15] Luis Alejandro Cortes, Petro Eles, Zebo Peng, “definitions of equivalence for transformational synthesis of embedded systems”, 6th IEEE International Conference on Complex Computer Systems (ICECCS'00) September 11 - 15, 2000 Tokyo, Japan

[16] Venkatesh Kurapati, Meng Chu Zhou, Reggie Caudill, “design of sequence controllers using petri net models”, Proceeding of IEEE Int. Conf. on Systems, Man, and Cybernetics, Vol. 5, Vancouver, Canada, Oct. 1995, pages 3469-3474. 1995

[17] Gianfranco Ciardo, “discret time markovian stochastic petri nets”, Computation with Markov Chains, p. 339-358, Raleigh, NC, USA, Jan 1995.

[18] Armin Zimmermann, Jorn Freiheit, Gunter Hommel, “discret time stochastic petri nets for modeling and evaluation of real time systems”, Parallel and Distributed Processing Symposium., Proceedings 15th International, Pages:1069 - 1074, 23-27 April 2001

[19] Enric Pastor, Jordi Cortadella, “efficient encoding schemes for symbolic analysis of petri nets”, Design, Automation, and Test in Europe Proceedings of the conference on Design, automation and test in Europe Le Palais des Congrés de Paris, France Pages: 790 – 795, 1998

[20] Luis Alejandro Cortes, Petro Eles, Zebo Peng, “hierarchies-for-the-modeling and verification of embedded systems”, SAVE Project Report, Dept. of Computer and Information Science, Linköping University, Linköping, February 2001

[21] Jorn W.Janneck, Robert Esser, “higher order petri net modeling techniques and applications”, Workshop on Software Engineering and Formal Methods, Petri Nets 2002, Adelaide, Australia

[22] Gert Dohhmen, “Petri Nets as Intermediate Representation between VHDL and Symbolic Transition Systems”, Proceedings Euro-DAC'94 with Euro-VHDL'94, pp. 572-577, IEEE Comupter Society Press, September 1994

[23] Carsten Rust, Friedhelm Stappert, Stefan Schamberger, “Integrating load balancing into petri net based embedded system design”, The European Simulation and Modeling Conference, University of Naples II, Naples, Italy, October 27-29, 2003

[24] Jordi Cortadella, Luciano Lavagno, Ellen Sentovich, “logic synthesis techniques for embedded control code optimization”, Proceedings of the International Workshop on Logic Synthesis (IWLS'97), Tahoe City, California, May 1997

[25] Miguel Canales, Bruno Gaujal, “marking optimization and parallelism of marked graphs”, http://www.inria.fr/rrrt/rr-2049.html

[26] Matthias Gries, “modeling a memory subsystem with petri nets case study”, Kluwer Academic Publishers, Editors: Alex Yakovlev, Luis Gomes, Luciano Lavagno, pages 291-310, March 2000, ISBN 0-7923-7791-5

[27] MengChu Zhou, Frank DiCesare, Dianlong Guo, “modeling and performance analysis of a ressource sharing manufacturing system using stochastic Petri nets”, Proceedings of the 5th IEEE International Symposium on Intelligent Control, 1990, Philadelphia, PA, USA, pages 1005-1010. Piscataway, NJ, USA: IEEE Service Center, 1990

[28] A.V.Yakovlev, A.M.Koelmans, A.Semenov, “Modelling analysis and synthesis of asynchronous control circuits using Petri Nets”, Integration: The VLSI Journal, Vol. 21, pages 143-170. 1996

[29] Soren Christensen, Laure Petrucci, “modular state space analysis of coloured petri net”, G. de. Michelis and M. Diaz (Eds.): Application and Theory of Petri Nets 1995. Proceedings of the 16th International Conference, Turin, Italy, Lecture Notes in Computer Science 935, Springer-Verlag, pp. 201-217, 1995.

[30] P.Mitrevski, M.Gusev, “performance evaluation of branch and value prediction using discret event simulation of fluid stochastic petri nets”, Proceedings of the Second International Conference on Informatics and Information Technology (CiiT'01), Molika, 20-23.Dec.2001

[31] Enric Pastor, Oriol Roig, Jordi Cortadella, Rosa M.Badia, “petri net analysis using boolean manipulation”, Valette, R.: Lecture Notes in Computer Science, Vol. 815; Application and Theory of Petri Nets 1994, Proceedings 15th International Conference, Zaragoza, Spain, pages 416-435. Springer-Verlag, 1994

[32] Julio A. de Oliveira Filho, Manoel E. de Lima, Paulo Romero Maciel, “petri net based interface analysis for fast IP-Core integration”, First ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE’03) June 24 - 26, 2003 Mont Saint-Michel, France

[33] W.M.P van der Aalst, “petri net based scheduling”, Computing Science Reports No. 95. Eindhoven University of Technology, 1995

[34] Tadao Murata, “ Petri Nets : Properties, Analysis and Applications”, Proceedings of the IEEE, Vol. 77, No. 4, pages 541-580. April 1989

[35] Tadao Murata, Jaegeol Yim, “petri net methods for reasoning in real time control systems”, Circuits and Systems, 1995. ISCAS '95., 1995 IEEE International Symposium on ,Volume: 1 ,Pages:517 - 520 vol.1, 28 April-3 May 1995

[36] P.Lutz, N.Djemel, A.Bourjault, “petri net modeling for multiproducts assembly systems including testing”, Proceedings of the 1994 International Conference on Robotics and Automation, San Diego, CA, USA, May 1994, Volume 2. IEEE Computer Society Press, 1994.

[37] Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, and Liang-Gee Chen, “Analysis and Architecture Design of Block Coding Engine for EBCOT in JPEG 2000”, IEEE Transaction on Circuits and Systems for video technology, Vol. 13, No. 3, March 2003

[38] Carsten Rust,Ac im Rettberg,and Kai Gossens, ‘’From High-Level Petri Nets to SystemC’’, IEEE International Conference on Systems, Man & Cybernetics, Hyatt Regency, Washington, D.C., USA, 5 - 8 October 2003

[39] Kurt Jensen, “An Introduction to the Theoritical Aspects of Coloured Petri Nets”, De Bakker JW, de Roever WP, Rozenberg G, Eds. A Decade of Concurrency. LNCS 803, Springer-Verlag, 230-272, 1994.

[40] http://pdv.cs.tu-berlin.de/~timenet/

[41] http://www.daimi.au.dk/PetriNets/tools/

[42] Guy Vidal-Naquet, Annie Choquet-Geniet, “Réseaux de Pétri et systèmes parallèles”, Edition Armand Colin, collection Acquis Avancés de l’Informatique (2AI), 1992.

[43] Paulo Maciel, Enda Barros, Wolfgang Rosenstie, “a petri net based approach for performing the initial allocation in hardware/software codesign”, In: Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'98), 11-14 October 1998, San Diego, CA, pages 505-510. 1998

[44] Ashok K.Murugavel N.Ranganathan, “petri net modeling of gate and interconnect delays for power estimation”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems Volume 11 , Issue 5 (October 2003) Special section on low power Pages: 921 - 927, 2003.

[45] A.Frescha, G.Haring, “petri-net-based-modeling of parallel programs executing on distributed memory multiproc systems”, Periodica Polytechnica Ser. El. Eng., Vol. 35, No. 3, pp 193 - 219, 1991

[46] Enrique Soto, Miguel Pereira, “Implementing a Petri Net Specification in a FPGA Using VHDL”, The International Workshop on Discrete-Event System Design, DESDes’01,June 27-29, 2001; Przytok near Zielona Gora, Poland

[47] Julio A. de Oliveira Filho, Manoel E. de Lima, Paulo Romero Maciel, Juliana Moura, Bruno Celso, “A Fast IP-Core Integration Methodology for SoC Design”, 16th Symposium on Integrated Circuits and Systems Design (SBCCI'03), September 08 - 11, 2003, São Paulo, Brazil, pages 131-136. IEEE Press, September 2003

[48] SystemC version 2.0. User’s Guide (update for SystemC 2.0.1) www.systemc.org

[49] Functional Specification for SystemC 2.0 www.systemc.org

[50] Master-Slave Communication Library – A SystemC Standard Library Version 2.0.1 www.systemc.org

[51] Synopsys CoCentric SystemC Compiler Behavioral User and Modeling Guide vU-2003-6

[52] Synopsys CoCentric SytemC Compiler RTL user and Modeling Guide vU-2003.06

[53] S.Chtourou and O.Hammami, Space Exploration of Behavioral SystemC Synthesis Options on Area and Performance, Third IEEE International Conference on Systems, Signals & Devices SSD'05 March 21-24, 2005, Sousse – Tunisia

[54] S.Chtourou and O.Hammami, SystemC Modelling of JPEG-2000 Entropy Coder, en preparation

Chapitre VIII Conclusion Générale

La méthodologie générale adoptée dans ce travail est répartie en deux grandes phases : La première phase est l'étude et l'évaluation des performances de deux implémentations SW de la partie I de JPEG2000 sur des plateformes commerciales. Les deux implémentations en question ont été réalisées en C/C++. La première est connue sous le nom de JasPer dans sa version 1.500.4 et qui a été considérée au début comme l'implémentation de référence de JPEG2000. La deuxième implémentation est Kakadu. Ces deux implémentations ont permis de faire des mesures de performance et l'étude du comportement par rapport aux différentes plateformes.

Les plateformes cibles utilisées dans cette étude sont formées par deux familles de composants : des processeurs généralistes de la famille Pentium d' Intel (le Pentium II, III, et 4), et des DSP avec deux architecture distinctes : le premier processeur est un DSP de Texas Instruments de la série TMS320C6xxx (TMS320C6711) les seconds sont des processeurs multimédia de la famille TriMedia de Philips de la série TM1xxx (TM1000, TM1100, et TM1300). Les processeurs TriMedia possèdent une architecture VLIW. Cette étude a permis de découvrir les points faibles de JPEG2000 sur ces différentes plateformes, et elle a ainsi permis de lancer une première phase d'optimisation de l'implémentation C/C++ réalisée dans le cadre du projet EIRE. Ces optimisations ont visé les Pentiums en utilisant des librairies spécialisées d'IPP d'Intel ainsi que du multithreading supporté par le Pentium 4. Pour le DSP et les processeurs TriMedia on a fait varier les options de la compilation et la taille de la mémoire.

Cette étude a confirmé que le temps d'exécution du codeur entropique représente la partie principale du temps d'exécution ce qui représente un défi majeur pour l'utilisation de JPEG2000 dans certaines applications très contraignantes de point de vue temps. De plus on s'est rendu compte que la nature du traitement qui s'opère au niveau bit pose des problèmes de cache et d'accès mémoires.

La seconde phase de ce travail est autour des implémentations matérielles du codeur entropique JEPG2000. L'approche suivie a été d'aborder tout le spectre de l'utilisation du HW. D' abord ces implémentations mettent en avant l'utilisation de coprocesseur avec des versions multicodeur dans une chaîne de codage SW de JPEG2000. Une première implémentation a été réalisée sur la carte Celoxica, avec la mise en parallèle de deux codeurs. Les points de faiblesse de cette implémentation sont d'abord la connexion PCI, ensuite l'architecture de la carte qui impose un mode de communication très contraignant.

Dans le cadre de l'utilisation du coprocesseur, on a changé de type de plateforme pour passer de l'utilisation d'une machine hôte à un système industriel embarqué. Cette nouvelle plateforme est la SBC405GP de Wind River qui est une carte indépendante avec son propre processeur et OS. Le composant FPGA Virtex XCV405E a servi pour l'implémentation des deux codeurs entropiques qui ont été intégrés à la chaîne de codage JPEG2000 SW qui s'exécute sur le PowerPC. Cette plateforme nous impose aussi des contraintes de communication entre le FPGA et le processeur vu l'architecture figée de la carte. La plateforme SBC405 représente aussi une charnière entre l'approche coprocesseur et l'approche système de l'implémentation du codeur entropique.

L'approche système a été étudiée en utilisant des plateformes Virtex-II-Pro. La première approche système consiste à faire le partitionnement HW/SW du codeur entropique. Ce partitionnement a généré plusieurs versions. Les modules HW, de cette première étude de l'approche système, ont été dans l'ensemble connectés au bus OPB du PowerPC. La seconde partie concerne l'implémentation multicodeur sur le bus PLB du PowerPC. L'ensemble de cette étude est résumé dans la **figure Ch VIII - 1**.

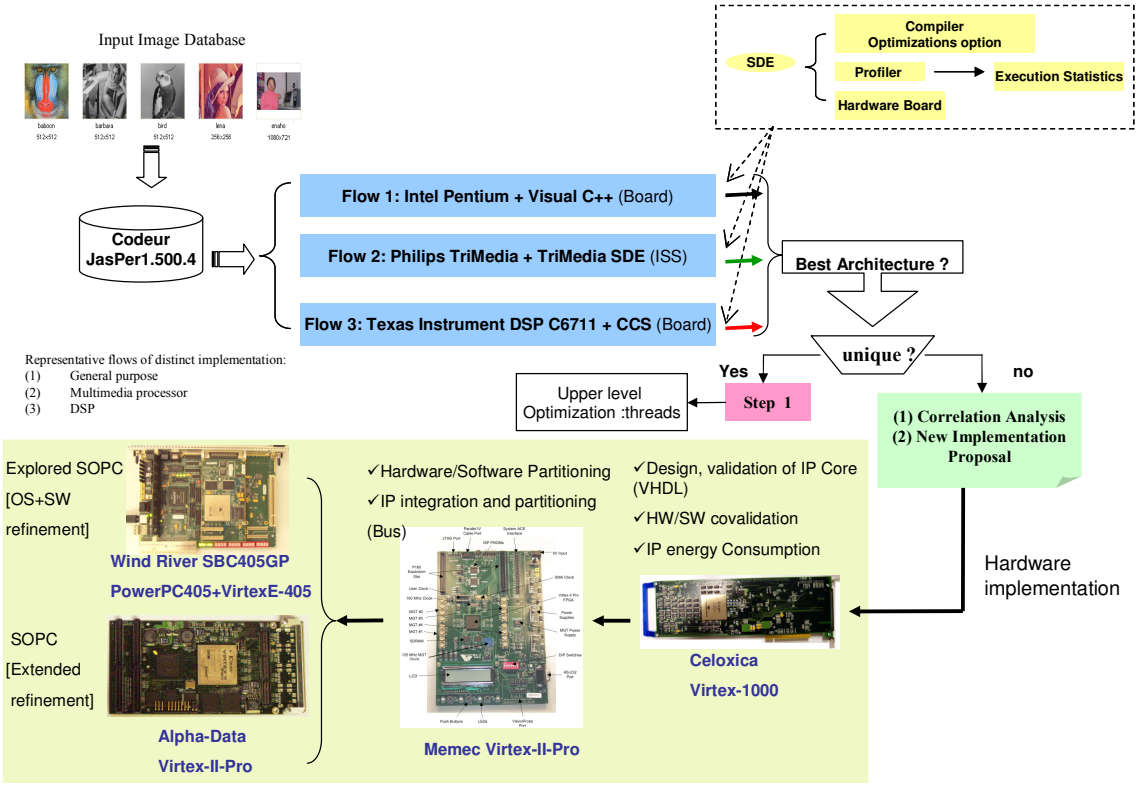


Figure Ch VIII - 1 : Méthodologies et travaux dans le projet EIRE

Ces études ont permis d'analyser une bonne partie du spectre des solutions d'implémentation matérielles et logicielles du standard JPEG2000 et particulièrement le codeur entropique.

Dans la partie d'évaluation des performances des implémentations SW sur des processeurs Pentium, DSP, et TriMedia, le choix des processeurs a été fait de manière intuitive car on ne disposait d'aucun moyen pour décider lequel de ces processeurs serait le plus représentatif de chacune des familles utilisées pour l'évaluation. Pour cette raison on a essayé de varier quand c'etait possible les processeurs pour la même famille. Bien que le code à évaluer fût le même, on ne pouvait faire que des comparaisons inter familles de processeur, de l'impact des optimisations, vu l'hétérogénéité des compilateurs et des options offertes à chacune de ces familles.

Le passage à l'implémentation matérielle du codeur entropique comme coprocesseur sur une machine hôte nécessite le choix de la plateforme. Dans ce cas plusieurs paramètres sont à prendre en considération tel que le débit de sortie de la DWT sur la machine hôte, la bande passante du support de communication avec la plateforme d'implémentation, la largeur du bus de données de ce support, et les ressources du composant FPGA qui va accueillir l'IP du codeur entropique. Vu que sur une machine hôte les possibilités d'ajout de matériels sont très limitées, le choix de la carte Celoxica laisse penser que tout ces paramètres sont suffisamment respectés pour réaliser une implémentation matérielle très performante, mais on se rend compte finalement que les contraintes liées à l'architecture de la carte ainsi qu'à la chaîne de codage EIRE viennent s'imposer et compromettent les performances de l'IP. Parmi ces contraintes on peut citer : les différentes interfaces utilisées au niveau SW et HW pour assurer la communication, le support de communication qui est le bus PCI 32 bits/33MHz, le fonctionnement de la chaîne de codage EIRE, les ressources de la carte, les ressources du composant Virtex. Ce type de contrainte a été retrouvé lors du portage de l'implémentation sur la carte SBC405 de Wind River. Donc on peut dire que quel que soit notre choix de la

plateforme on serait amené à gérer ces contraintes et essayer de minimiser leur impact sur les performances de l'IP. De plus la réutilisation d'une IP portable pose beaucoup de problèmes d'adaptation qui peuvent s'avérer fatals du point de vue performance. Dans le cas particulier de l'utilisation d'une IP comme coprocesseur, l'impact de la réutilisation dépend principalement de l'architecture SW et HW globale du système.

Dans l'implémentation système et pendant sa première phase à savoir le partitionnement HW/SW on a pu corriger l'idée donnée par l'évaluation des performances SW qui montrent que le codeur arithmétique est la partie qui nécessite le plus de temps de traitement. Le partitionnement de l'IP du codeur entropique en deux IP permet de mettre en HW l'une ou l'autre des deux IP selon les besoins de l'application avec un effort de réadaptation qui est beaucoup moins conséquent que celui pour les plateformes précédentes. Comme la partie précédente, l'effort de réadaptation s'opère tant au niveau HW qu'au niveau SW. Pour cette étude, on a gardé deux tâches principales, vu que le partitionnement a une granularité plus fine, et que la forte dépendance des données auront engendré des surcoûts de communication non négligeables. Vu ce nombre de tâches, on n'a donc pas, fait recours à des algorithmes de partitionnement tels que ceux cités dans le chapitre III. Par contre tout algorithme de partitionnement basé sur les mesures du temps d'exécution aura de forte chance de se tromper et de décider que c'est le codeur arithmétique qui sera implémenté en HW. Ceci est dû à la forte dépendance des données dans le codeur entropique ainsi qu'à la nature même du traitement qui est intrinsèquement séquentiel.

Dans la partie système où on utilise plusieurs codeurs entropiques connectés au bus PLB, l'effort d'adaptation de l'IP au bus est beaucoup moins élevé que toutes les approches précédentes. En effet cet effort est effectué une seule fois. Dans cette approche, quel que soit le nombre de codeurs entropiques utilisés, une seule adaptation HW est nécessaire. Mais du point de vue SW, on doit réadapter la partie qui contrôle les codeurs entropiques. Une approche de multithreading peut réduire considérablement l'effort de développement d'un code séquentiel qui contrôle la totalité des codeurs connectés au bus PLB.

Le point commun à toutes ces études est la décision du nombre de codeurs entropiques parallèles possibles. Certes ce nombre dépend de la bande passante du support de communication en amont et en aval des codeurs implémentés en HW, mais il ne faut pas oublier l'impact du débit de sortie de la DWT qui détermine le nombre de codeurs qu'on peut mettre en parallèle sur un ou plusieurs supports de communication différents. Ceci n'a pas été pris en compte dans le cas de cette étude car ce nombre, indépendamment du débit de la DWT est limité par la loi d'Amdhal qui dit que :

$$Speedup = \frac{Temps\ d'exécution\ sans\ accélération}{Temps\ d'exécution\ avec\ mécanisme\ d'accélération}$$

Dans le cas de la chaîne JPEG2000, si on suppose que le temps de communication entre le codeur entropique et le reste de la chaîne de codage est négligeable cette loi peut s'écrire sous la forme suivante :

$$Speedup = \frac{Trest + Tcodeur}{Trest + Tcodeur / Ncodeurs}$$

Où :

- Trest : le temps d'exécution de la chaîne de codage JPEG2000 sans le codeur entropique
- Tcodeur : le temps d'exécution du codeur entropique
- Ncodeurs : le nombre de codeurs exécutés en parallèle

La variation du Speedup en fonction du nombre de codeurs est représentée dans la figure suivante :

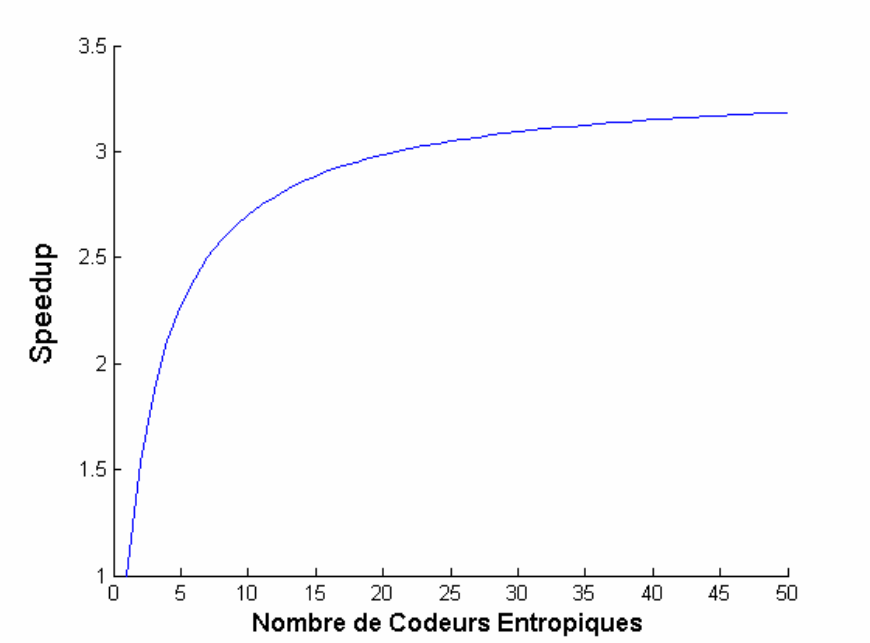


Figure Ch VIII - 2 : Variation du Speedup en fonction du nombre de codeurs entropiques

Cette courbe est obtenue en fixant les proportions de temps comme suit : Tcodeur=70%, et Trest=30% du temps d'exécution de la chaîne JPEG2000. Dans le cas général le Speedup ne peut pas dépasser 1/Trest (1/30% dans notre cas), donc dans notre cas on a : Speedup<3,33 quel que soit le nombre de codeurs en parallèle.

La dépendance des données est très forte dans un codeur JPEG2000. Par conséquent, l'implémentation matérielle d'un codeur complet, où tous les modules seront conçus sur mesure, est le moyen idéal pour atteindre de meilleures performances. Néanmoins, ceci nous a été impossible dans le cadre de ce travail pour les raisons suivantes. D'abord la spécification du projet EIRE nous a imposé l'étude du codeur entropique seul. De plus, un tel travail dépasse le cadre du projet EIRE (temps de développement estimé>2 ans : la durée du projet). Ensuite les implémentations logicielles de JPEG2000 (qui ne sont pas les notres) ne permettent pas d'avoir ni mesures précises de la communication entre les différents modules de JPEG2000 (DWT, codeur entropique, reconstruction du fichier), ni de voir le pipeline du codage. Enfin, l'architecture logicielle du codeur JPEG2000, que ce soit Kakadu, JasPer, ou le codeur EIRE, est à l'opposé de ce que peut être une architecture matérielle. Par exemple les modes de communication entre les deux parties du codeur entropique d'une implémentation matérielle et d'une implémentation logicielle sont complètement différentes (généralement une FIFO pour la première implémentation, appel de fonction pour la deuxième).

De plus, puisque JPEG2000 offre aussi la possibilité de partager une image en tile, l'étude d'une implémentation multicodeur-multiprocesseur pour le traitement de plusieurs tiles en parallèle peut faire l'objet d'une étude plus poussée de l'implémentation HW de JPEG2000. Cette étude peut réutiliser, et faire recours aux résultats obtenus dans l'étude présentée le long de ce document.

Une première idée, c'est d'adapter l'architecture du processeur d'Intel MXP5800 qui est donnée dans la **figure Ch VIII - 3** à un composant FPGA comme ceux de la famille Virtex-II ou Virtex-II-Pro sinon le Virtex 4. L'ISP (Image Signal Processor) peut être remplacée par d'autre processeur soft comme le MicroBlaze et/ou hard comme le PowerPC. Les unités

d'accélérateur sur le processeur d'ISP peuvent être la réutilisation du travail présenté précédemment.

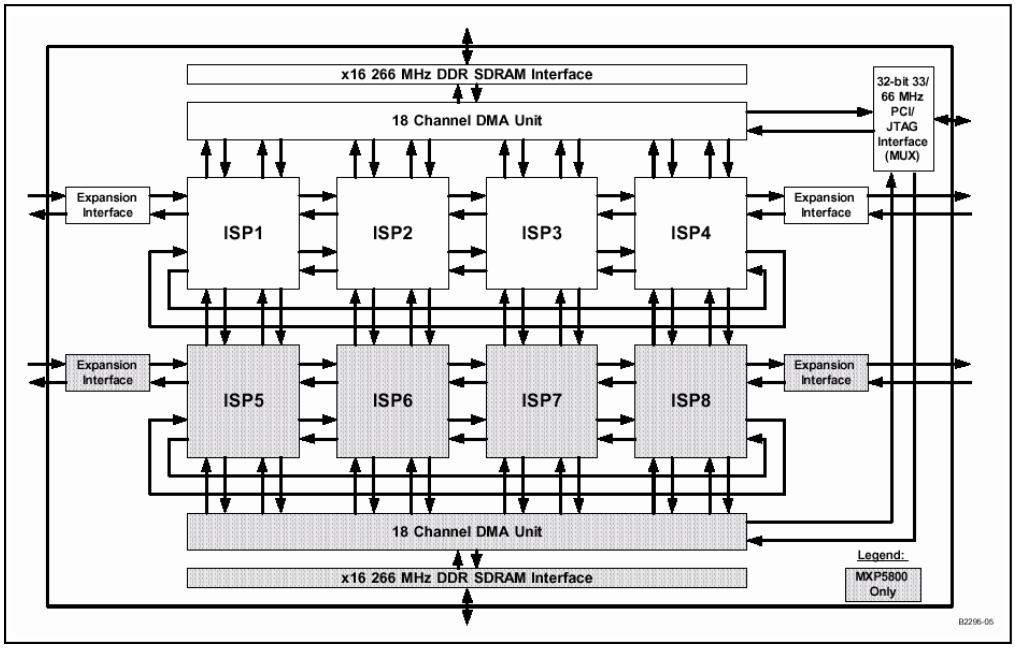


Figure Ch VIII - 3 : Architecture du MXP5800

L'avantage de ce choix est la possibilité de modifier dynamiquement les blocs d'accélération par différents types d'accélérateurs comme la DWT (JPEG2000), DCT/IDCT, Codeur/décodeur Huffman (JPEG),…. À la différence du processeur ISP nous ne sommes pas obligés d'implémenter tous les accélérateurs en même temps.

Liste des publications

Internationales

SSIP 2002	Imed Aouadi, Omar Hammami, "Microarchitectural Characterization of JPEG-2000 Software", Proceeding of the 9th International Workshop on Systems, Signals and Image Processing - Recent Trends in Multimedia Information Processing, Manchester, UK, November 2002
ICM 2002	Omar Hammami, Enhao Zheng, Imed Aouadi "Performance Evaluation of JPEG-2000 on VLIW Architectures" - Proceeding of the 14th IEEE International Conference on Microelectronics, Beirut, Lebanon, December 2002
PCS 2003	Imed Aouadi, Omar Hammami, "A SOPC Oriented FPGA Implementation of the JPEG-2000 Entropy Coder", Picture Coding Symposium, Saint Malo, France, Avril 2003
EUROMICRO-DSD 2004	Imed Aouadi, Omar Hammami "Analysis and Hardware Design of a Scalable Dual JPEG-2000 Entropy Code", Euromicro Symposium on Digital System Design (DSD'04) August 31 - September 03, 2004 Rennes, France
EUSIPCO 2004	Imed Aouadi, Riad Benmouhoub, Omar Hammami, "Exploring JPEG-2000 Entropy Coder Implementations on Xilinx Virtex-II Pro Platform",12th European Signal Processing Conference September 6-10, 2004 Vienna, Austria
SASIMI 2004	Imed Aouadi, Riad Benmouhoub, Omar Hammami, "System on programmable chip platform based design of JPEG-2000 entropy coder", The 12th Workshop on Synthesis And System Integration of Mixed Information technologies October 18 - 19, 2004. Kanazawa, Japan
ICIT 2004	Imed Aouadi, Omar Hammami, "Low Power JPEG-2000 Image Compression for Industrial Embedded Applications", IEEE International Conference on Industrial Technology December 8 - 10, 2004 tunisia
ICCE 2005	Imed Aouadi, Omar Hammami, "System on a programmable chip Oriented JPEG-2000 Entropy Coder Implementation for Multimedia Embedded Systems", International Conference on Consumer Electronics, January 10-12 2005 Las Vegas USA

Nationales

CORESA 2004	Imed Aouadi, Omar Hammami, "Bi-codeur entropique JPEG-2000 pour applications mobiles", Compression et Représentation des Signaux Audiovisuels, 25 et 26 mai 2004, Lille Métropole Imed Aouadi, Omar Hammami, "Modélisation par Réseaux de Petri du codeur entropique de JPEG-2000", Compression et Représentation des Signaux Audiovisuels, 25 et 26 mai 2004, Lille Métropole

Journal

IEEE transaction on industrial electronics	I.Aouadi, O.Hammami, "Embedded Industrial Boards vs Embedded Platform FPGA for Industrial Embedded Applications-A Case Study"

Glossaire

A

ADD	Area-Driven Design
ALU	Arithmetic and Logic Unit
ASIC	Application Specific Integrated Circuit

B

BBD	Block-Based Design
bpp	bits par pixel
BRAM	Random Access Memory
BTB	Branch Target Buffer

C

CCS	Code Composer Studio
CLB	Configurable Logic Blocks
COSYN	Hardware Software Co-Synthesis for Embedded Systems algorithm
CPI	Cycle Per Instruction
CPLD	Complex Programmable Logic Devices
CPU	Central Processing Unit

D

DCM	Digital Clock Manager
DCR	Device Control Register
DCT	Discret Cosine Transform
DDR SDRAM	Double Data Rate SDRAM
DLL	Delay-Locked Loops
DMA	Direct Memory Acces
DRL	Dynamically Reconfigurable Logic
DSP	Digital Signal Processor
DVD	Digital VideoDisc
DWT	Discret Wavelet Transform

E

EBCOT	Embedded Block Coding with Optimal Truncation
EDIF	Electronic Design Interchange Format
EDK	Embedded Development Kit
EEPROM	Electrically-Erasable PROM
EIRE	Etudes d'optImisation algoRithmiques de JPEG2000
ELF	Executable and Link Format
EPROM	Erasable PROM

F

FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
FPLA	Field Programmable Logic Array

FSM	Finate State Machine

G

GCLP	Global Critically/Local Phase driven algorithm
GDB	GNU Debugger
GPIO	General Purpose Input/Output

H

HDL	Hardware Description Language
HVXC	Harmonic and Vector eXcitation Coding
HW	Hardware

I

IC	Integrated Circuit
ICT	Irreversible Color Transform
ILP	Interger Linear Program
IP	Intellectual Property
IPIC	IP Interconnect
IPIF	IP interface
IPP	Integrated Performance Primitives
ISE	integrated Software Environment
ISO	International Standards Organisation
ISP	Image Signal Processor
ITU-T	International Telecommunication Union

J

JBIG	Joint Bi-level Image experts Group
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group

L

LC	Logic Cells
LPS	Less Probable Symbol
LSI	Large Scale Integrated
LUT	Look-Up Table

M

MAC	Multiplication/ACcumulation
MCSE	Méthodologie de Conception de Systèmes Electroniques
MFLOPS	Million FLoating-Point Operations Per Second
MHS	Microprocessor Hardware Specification
MIBS	Mapping and Implementation Bin Selection
MIPS	Millions of Instruction Per Second
MMAC	Million of MAC
MMX	MultiMedia eXtension
MOGAC	MultiObjective Genetic Algorithme for hardware-software Co-synthesis
MOPS	Million of Operations Per Second
MPEG-4-VTC	Moving Picture Experts Group-4-Visual Texture Coding
MPGA	Mask-Programmable Gate Array
MPS	More Probable Symbol
MSB	Most Significant Bit

MSE	Mean Square Error
MSS	Microprocessor Software Specification

O

OMI	Open Model Interface
OPB	On-chip Peripheral Bus
OS	Operating System

P

PAL	programmable Array Logic
PBD	Platform Based Design
PCB	Profiling Based Compiler
PCI	Periperal Component interconnecte
PLB	Processor Local Bus
PMC	PCI Mezzanine Card
PNG	Portable Network Graphics
PROM	Programmable Read-only Memory
PSNR	Peak-SNR

R

RAM	Random Access Memory
RCT	Reversible Color Transform
RdP	Réseaux de Petri
RGB	Red, Green, Blue
ROI	Region Of Interest
ROM	Read Only Memory
RTL	Register Transfer Level
RTOS	Real Time Operating System
RVB	Rouge Vert Bleu

S

SDE	Software Development Environment
SDRAM	Synchronous Dynamic RAM
SIMD	Single Instruction, Multiple Data
SLI	System-Level Integration
SNR	Signal-to-Noise Ratio
SOC	System on Chip
SOPC	System On Programmable Chip
SPLD	Simple Programmable Logic Devices
SRAM	Static Random Access Memory
SSE	Streaming SIMD Extensions
SW	Software

T

TDD	Timing Driven Design
TLB	Translation Lookaside Buffer
TLM	Transaction Modeling Level
TTM	Time To Market

U

UART	Universal Asynchronous Receiver Transmitter

UMTS	Universal Mobile Telecommunications System

V

VC	Virtual Cores
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLIW	Very Long Instruction Word
VLSI	Very-Large-Scale Integration
VSIA	Virtuel Socket Interface Alliance

W

WG	Work Group

X

XMD	Xilinx Microprocessor Debugger
XPS	Xilinx platform studio
XST	Xilinx Synthesis Technology

Z

ZBT SRAM	Zero Bus Turnaround SRAM