**Université Badji Mokhtar - Annaba** **Université de Bourgogne - Dijon**

**FACULTE DES SCIENCES DE L' INGENIORAT**

**DÉPARTEMENT D' ELECTRONIQUE**

**T H E S E**

**DE DOCTORAT EN SCIENCE**

**SPECIALITE : ELECTRONIQUE**

Pour l'obtention du grade de Docteur en Science, en cotutelle
entre l'université Badji Mokhtar d'Annaba et l'université de Bourgogne de Dijon

---

Intitulée : # TRAITEMENT DES SIGNAUX ET IMAGES EN TEMPS RÉEL « IMPLANTATION DE H.264 SUR MPSoC »

---

Présentée par : **MESSAOUDI KAMEL**

Soutenue le 19 Décembre 2012 devant le jury de soutenance composé de :

| | | | |
|---|---|---|---|
| **FARAH** | **Nadir** | **PROFESSEUR, Université d' Annaba** | **Président** |
| **TOUMI** | **Salah** | **PROFESSEUR, Université d' Annaba** | **Rapporteur** |
| **BOURENNANE** | **El Bay** | **PROFESSEUR, Université de Dijon** | **Rapporteur** |
| **BOURIDANE** | **Ahmed** | **PROFESSEUR, Université de New Castle** | **Examinateur** |
| **CHAOUI** | **Allaoua** | **PROFESSEUR, Université de Constantine** | **Examinateur** |
| **TANOUGAST** | **Camel** | **MAITRE DE CONFERENCES, Université de Metz** | **Examinateur** |
| **BILBAULT** | **Jean Marie** | **PROFESSEUR, Université de Dijon** | **Invité** |
| **BOUKEZZOULA** | **Nasser-Eddine** | **MAITRE DE CONFERENCES, Université de Sétif** | **Invité** |

# RÉSUMÉ

Cette thèse est élaborée en cotutelle entre l'université Badji Mokhtar (Laboratoire LERICA) et l'université de bourgogne (Laboratoire LE2I, UMR CNRS 5158). Elle constitue une contribution à l'étude et l'implantation de l'encodeur H.264/AVC. Durent l'évolution des normes de compression vidéo, une réalité sure est vérifiée de plus en plus : avoir une bonne performance du processus de compression nécessite l'élaboration d'équipements beaucoup plus performants en termes de puissance de calcul, de flexibilité et de portabilité et ceci afin de répondre aux exigences des différents traitements et satisfaire au critère « Temps Réel ». Pour assurer un temps réel pour ce genre d'applications, une solution reste possible est l'utilisation des systèmes sur puce (SoC) ou bien des systèmes multiprocesseurs sur puce (MPSoC) implantés sur des plateformes reconfigurables à base de circuit FPGA.

L'objective de cette thèse consiste à l'étude et l'implantation des algorithmes de traitement des signaux et images et en particulier la norme H.264/AVC, et cela dans le but d'assurer un temps réel pour le cycle codage-décodage. Nous utilisons deux plateformes FPGA de Xilinx (ML501 et XUPV5). Dans la littérature, il existe déjà plusieurs implémentations du décodeur. Pour l'encodeur, malgré les efforts énormes réalisés, il reste toujours du travail pour l'optimisation des algorithmes et l'extraction des parallélismes possibles surtout avec une variété de profils et de niveaux de la norme H.264/AVC.

Dans un premier temps de cette thèse, nous proposons une implantation matérielle d'un contrôleur mémoire spécialement pour l'encodeur H.264/AVC. Ce contrôleur est réalisé en ajoutant, au contrôleur mémoire DDR2 des deux plateformes de Xilinx, une couche intelligente capable de calculer les adresses et récupérer les données nécessaires pour les différents modules de traitement de l'encodeur. Ensuite, nous proposons des implantations matérielles (niveau RTL) des modules de traitement de l'encodeur H.264. Sur ces implantations, nous allons exploiter les deux principes de parallélisme et de pipelining autorisé par l'encodeur en vue de la grande dépendance inter-blocs. Nous avons ainsi proposé plusieurs améliorations et nouvelles techniques dans les modules de la chaine Intra et le filtre anti-blocs. A la fin de cette thèse, nous utilisons les modules réalisés en matériels pour la l'implantation Matérielle/logicielle de l'encodeur H.264/AVC. Des résultats de synthèse et de simulation, en utilisant les deux plateformes de Xilinx, sont montrés et comparés avec les autres implémentations existantes.

**Mots clés :** Implantation matérielle, Codesign, L'encodeur H.264/AVC, Temps réel, Gestion de mémoire, Contrôleur mémoire, Parallélisme, Pipelining, SoC, MPSoC, FPGA, Plateforme de prototypage, Xilinx, ML501, XUPV5.

# ABSTRACT

This thesis has been carried out in joint supervision between the Badji Mokhtar University (LERICA Laboratory) and the University of Burgundy (LE2I laboratory, UMR CNRS 5158). It is a contribution to the study and implementation of the H.264/AVC encoder. The evolution in video coding standards have historically demanded stringent performances of the compression process, which imposes the development of platforms that perform much better in terms of computing power, flexibility and portability. Such demands are necessary to fulfill requirements of the different treatments and to meet "Real Time" processing constraints. In order to ensure real-time performances, a possible solution is to made use of systems on chip (SoC) or multiprocessor systems on chip (MPSoC) built on platforms based reconfigurable FPGAs.

The objective of this thesis is the study and implementation of algorithms for signal and image processing (in particular the H.264/AVC standard); especial attention was given to provide real-time coding-decoding cycles. We use two FPGA platforms (ML501 and XUPV5 from Xilinx) to implement our architectures. In the literature, there are already several implementations of the decoder. For the encoder part, despite the enormous efforts made, work remains to optimize algorithms and extract the inherent parallelism of the architecture. This is especially true with a variety of profiles and levels of H.264/AVC.

Initially, we proposed a hardware implementation of a memory controller specifically targeted to the H.264/AVC encoder. This controller is obtained by adding, to the DDR2 memory controller, an intelligent layer capable of calculating the addresses and to retrieve the necessary data for several of the processing modules of the encoder. Afterwards, we proposed hardware implementations (RTL) for the processing modules of the H.264 encoder. In these implementations, we made use of principles of parallelism and pipelining, taking into account the constraints imposed by the inter-block dependency in the encoder. We proposed several enhancements and new technologies in the channel Intra modules and the deblocking filter. At the end of this thesis, we use the modules implemented in hardware for implementing the H.264/AVC encoder in a hardware/software design. Synthesis and simulation results, using both platforms for Xilinx, are shown and compared with other existing implementations.

**Keywords :** Hardware implementation, Codesign, H.264/AVC encoder, Real time, memory management, Memory controller, Parallelism, Pipelining, SoC, MPSoC, FPGA, Prototyping platform, Xilinx, ML501, XUPV5.

# الملخـص


تم تطوير هذه الأطروحة تحت الإشراف المشترك بين جامعة باجي مختار (مختبرLERICA) وجامعة bourgogne (مختبرLE2I , UMR-CNRS 5158). هذه الأطروحة هي مساهمة لدراسة وتنفيذ التشفير H.264/AVC الخاص بضغط الفيديو. مع الوقت و تطور أنظمة ضغط الفيديو، واقع واجد يزداد تأكدا أكثر فأكثر : الأداء الجيد لعملية الضغط يتطلب تطوير معدات أكثر كفاءة من حيث القدرة الحاسوبية، والمرونة والقابلية خاصة من أجل تلبية متطلبات العلاجات المختلفة وتلبية معيار "الوقت الحقيقي Temps Réel ". لضمان الوقت الحقيقي لهذا النوع من التطبيقات، الحل الممكن هو استخدام (نظم على رقاقة systèmes sur puce) أو أنظمة متعددة المعالجات على رقاقة واحدة (systèmes multiprocesseurs sur puce MPSoC) و ذلك باستعمال منصات تحتوي على رقاقات FPGA.

الهدف الرئيسي من هذه الأطروحة هو دراسة وتنفيذ خوارزميات لمعالجة الإشارات والصور و على وجه الخصوص تلك المستعملة في المشفر H.264/AVC. من أجل ضمان الوقت الحقيقي لتحقيق هذا المشفر نستخدم اثنين من منصات الرقاقات FPGA : ML501 et XUPV5 التابعة لشركة Xilinx. حاليا هناك بالفعل العديد من تطبيقات فك الترميز H.264/AVC، لكن على الرغم من الجهود الهائلة المبذولة، لا يزال هناك عمل هائل على مستوى الخوارزميات و كذا على مستوى استخراج التوازي الممكن خاصة مع مجموعة متنوعة من التشكيلات والمستويات من المشفر H.264/AVC.

في خطوة هي الأولى من هذه الأطروحة، نقترح تنفيذ وحدة تحكم للذاكرة الخارجية خصيصا لترميز H.264/AVC. يتم إجراء هذا عن طريق إضافة وحدة تحكم ذكية إلى وحدة تحكم الذاكرة DDR2 على كل منصات الرقاقات FPGA من نوع Xilinx ، هذه الطبقة الذكية تكون قادرة على حساب العناوين واسترجاع البيانات اللازمة لتجهيز وحدات مختلفة من التشفير. ثم بعد ذلك، فإننا نقترح تطبيقات على المستوى البرمجي (RTL) لمختلف وحدات المعالجة للتشفير H.264/AVC. في هذه التطبيقات، استخدمنا مبادئ التوازي و Pipeline وذلك نظرا لشدة التداخل و الترابط في البيانات. و بذلك نكون قد اقترحنا عدة تحسينات وتقنيات جديدة لتحقيق وحدات المعالجة للتشفير H.264/AVC ومقارنتها مع تطبيقات أخرى قائمة.



**الكلمات المفتاحية :** نماذج تطبيقية , Codesign , نظام التشفيرH.264 , الوقت الحقيقي , وحدة التحكم في الذاكرة , مبادئ التوازي , Pipelining , SoC , MPSoC , FPGA , منصات الرقاقات FPGA , Xilinx , ML501 , XUPV5.

*A la mémoire de mon père*
*Puisse Dieu, le tout puissant,*
*l'avoir en sa sainte miséricorde*

*A ma mère*
*A celle que j'aime*
*A ma petite fille Maissane*
*A ma famille*
*A tous ceux qui me tiennent à cœur*

# *Remerciements*

Cette thèse s'est effectuée en cotutelle entre l'équipe systèmes embraqués du Laboratoire LERICA de l'université Badji Mokhtar de Annaba (Algérie) et l'équipe architectures du laboratoire l'Electronique, Informatique et Image (LE2I) de l'université de Bourgogne de Dijon (France).

Je remercie Professeur **TOUMI Sala**h pour m'avoir confié ce sujet de thèse, pour l'intérêt qu'il a porté au déroulement de mes travaux et d'avoir contribué à l'amélioration de la qualité de cette thèse.

Je remercie Professeur **BOURENNANE El-Bay** pour m'avoir confié ce sujet de thèse et pour m'avoir accueilli au sein de leur équipe et surtout pour ce goût du savoir qu'il a réussi à me faire partager.

Je remercie Monsieur **FARAH Nadir**, Professeur à l'université de Annaba, qui m'a fait l'honneur de présider ce jury.

Je suis reconnaissant à Monsieur **TANOUGAST Camel**, Maître de conférences (HDR) de l'université Paul Verlaine-Metz (Laboratoire Interfaces Capteurs et Microélectronique), et Monsieur **CHAOUI Alaoua**, Maître de conférences (HDR) de l'université Mentouri-Constantine qui ont accepté d'être rapporteurs de mon manuscrit de thèse.

Je remercie Monsieur **Ahmed BOURIDANE**, Professeur de l'université de New Castle-UK, Monsieur **Jean-Marie BILBAULT**, Professeur de l'université de bourgogne-Dijon (Laboratoire LE2I), et Monsieur **Nasser Eddine BOUKEZZOULA**, Maître de conférences de l'université Farhat Abbass-Sétif, qui m'ont beaucoup honoré par leurs présences en tant que membres de jury de cette thèse.

Je souhaite également remercier les deux équipes qui m'ont accueillie et dans lesquelles j'ai eu plaisir à travailler. L'équipe architectures du laboratoire LE2I de l'université de Bourgogne de Dijon-France et l'équipe systèmes embraqués du Laboratoire LERICA de l'université Badji Mokhtar de Annaba-Algérie. Les ambiances chaleureuses et décontractées m'ont permis de réaliser mes travaux dans de très bonnes conditions, soutien essentiel pour achever mes travaux de thèse.

J'adresse aussi tous mes remerciements à l'ensemble des personnes qui ont contribué à la réalisation et l'amélioration de mes travaux de thèse ainsi qu'à la rédaction de mon manuscrit.

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# Table des matières



## Partie I : Etat de l'art

### Chapitre I : Conception des systèmes embarqués.



### Chapitre II : La compression vidéo et la nouvelle norme de compression H.264/AVC

**Chapitre III : Les outils de conception des systèmes et le traitement d'images temps réel**

# Partie II : Implémentations matérielles

**Chapitre IV : Implantation de la gestion de mémoire pour l'encodeur H.264 sur des plateformes Virtex5 de Xilinx**

# INTRODUCTION GÉNÉRALE

## Cadre général

Les progrès continus des technologies CMOS ont conduit au développement de systèmes complets dans une seule puce de silicium (SoC pour System on Chip). Sur ces puces nous trouvons des dizaines de millions de transistors pour assurer l'implantation physique des différents composants constituant le système (Processeurs, DSPs, registres, mémoires embarquées, etc.). En plus, selon plusieurs statistiques le nombre de transistors augmente de 60% par an (ce qui confirme la loi empirique de Gordon Moore qui dit que les performances des processeurs doivent approximativement doublées tous les 18 mois).

De nos jours, on trouve de plus en plus des puces dans des applications grand public diverses et variées. Elles équipent les nouvelles générations de téléphones mobiles, assurent des fonctions critiques dans les voitures (freinage ABS, gestion des airbags, etc.), constituent le cœur des consoles de jeux, elles sont utilisées aussi dans la plupart des appareils multimédia (les PCs, les assistants personnels PDA, les appareils photo, les téléphones mobiles, etc.). Un nombre considérable d'appareil de haute technologie permet un accès à une multitude de fichiers multimédia par l'intermédiaire d'un réseau de distribution. L'hétérogénéité des possibilités d'affichage et de transport implique diverses adaptations de ces contenus multimédias afin de les rendre diffusables. En effet, la demande des communications de multimédia sur des applications mobiles et portatives se développe de plus en plus. Il est essentiel de réaliser des communications de multimédia, mettant en application une norme visuelle de compression dans tous les multimédias traitants le système sur puce.

Le standard H.264, connu aussi sous le nom de MPEG-4/AVC (AVC pour Advance Video Coding) ou encore sous le nom de MPEG-4/part10, a été développé par le groupe de travail JVT (Joint Video Team) en mai 2003 et finalisé en mars 2005. Ce standard apporte un gain en qualité d'image de 2dB et une réduction du débit binaire de 50% par rapport au standard de compression MPEG-2. Ce qui fait de la norme H.264/AVC la plus adaptée à une large gamme d'applications avec des débits variant entre 100Kbps et 30Mbps. En effet, cette nouvelle norme, avec plusieurs profils et niveaux, peut considérablement réduire les conditions de largeur de bande et de stockage pour des données de multimédia.

Durant l'évolution des normes de compression vidéo, une réalité sûre est vérifiée de plus en plus : avoir une bonne performance du processus de compression nécessite l'élaboration d'équipements beaucoup plus performants en termes de puissance de calcul, de flexibilité et de portabilité et ceci afin de répondre aux exigences des différents traitements et satisfaire au critère « Temps Réel ». En effet, les applications dans les domaines des télécommunications, du multimédia et du traitement vidéo deviennent de plus en plus complexes et présentent des contraintes d'utilisation critiques. Les solutions classiques, basées seulement sur l'utilisation des processeurs standards, deviennent incapables de répondre aux exigences de ces applications, et cela malgré l'augmentation des fréquences de fonctionnement et les progrès au niveau de la microarchitecture des processeurs (pipelines, caches, exécution super-scalaire, etc.).

Pour assurer un temps réel pour ce genre d'applications, une solution possible est l'utilisation des systèmes multiprocesseurs sur puce (MPSoC) implantés sur des plateformes reconfigurables à base de circuit FPGA. Ces systèmes peuvent rassembler, sur une même puce, un ou bien plusieurs processeurs homogènes ou hétérogènes, des DSP, des accélérateurs matériels, des mémoires et des dispositifs d'interconnexion. Cette flexibilité dans le nombre et la nature des composants (surtout pour les accélérateurs matériels), va nous permettre de réaliser le nombre désiré des traitements tout en respectant le temps réel. En effet, le nombre des accélérateurs matériels dans un système sur puce ainsi que le nombre des traitements en parallèle dans ces accélérateurs ne sont pas limités que par la capacité des circuits reconfigurables utilisés.

## Objectif de la thèse

La quantité de calculs dans les modules de traitement de l'encodeur H.264/AVC dépasse les capacités des processeurs génériques actuels. Ceci va encore se vérifier dans le futur, du fait des améliorations significatives dans les standards de compression vidéo. L'objectif de cette thèse consiste à l'étude et l'implantation des algorithmes de traitement des signaux et images et en particulier la norme H.264/AVC, et cela dans le but d'assurer un temps réel pour le cycle codage-décodage.

Une alternative consiste en l'utilisation de plateformes multi-composants hétérogènes de type DSP/FPGA. Les DSP sont utilisés pour leur simplicité de programmation et les FPGA pour leurs performances. Dans notre cas, nous utilisons deux plateformes FPGA de Xilinx (ML501 et XUPV5). En plus de plusieurs périphériques, ces deux plateformes de la génération Virtex5 disposent de circuits FPGA avec un nombre de ressources matérielles important, ce qui semble suffisant pour l'implantation des différents modules de traitement de l'encodeur H.264/AVC. Dans la littérature, il existe déjà plusieurs implémentations du décodeur. Pour l'encodeur, malgré les efforts énormes réalisés, il reste toujours du travail pour l'optimisation des algorithmes et l'extraction des parallélismes possibles surtout avec une variété de profils et de niveaux de la norme H.264/AVC.

## Contributions

Dans un premier temps, nous allons étudier la norme H.264 en se basant sur deux grands aspects : les besoins en mémoire de l'encodeur et les détails d'implantation de chaque module de traitement. En effet, l'optimisation d'une implémentation nécessite une très bonne connaissance de l'architecture. Nous allons voir que la complexité de la norme ne réside pas uniquement dans la partie traitement et la partie commande, mais aussi dans le degré de dépendance inter-bloc dans les différents modules et par conséquent la gestion de la mémoire pour ces modules.

L'utilisation des mémoires externes pour l'implantation des algorithmes de traitement d'image est indispensable, et cela en vu de l'utilisation énorme des ressources mémoires par ces algorithmes. De même, l'utilisation des contrôleurs pour la gestion mémoire est une nécessité qui dégage la partie traitement de plusieurs fonctions. La première contribution dans cette thèse consiste à la proposition et l'implantation d'un contrôleur mémoire conçu spécialement pour l'encodeur H.264/AVC. Ce contrôleur est réalisé en ajoutant, au contrôleur mémoire DDR2, une couche intelligente capable de calculer les adresses et récupérer les données nécessaire pour les différents modules de traitement de l'encodeur. Dans ce contrôleur nous exploitons tous les détails de l'encodeur ainsi que les besoins en

mémoire des différents modules. Le contrôleur proposé doit être aussi adapté aux différents profils et niveaux de la norme H.264/AVC.

La deuxième contribution de cette thèse consiste à l'implantation en langage VHDL (niveau RTL) des modules de traitement de l'encodeur H.264. Sur ces implantations, nous allons exploiter le maximum de parallélisme autorisé par l'encodeur en vue de la grande dépendance inter-blocs. Nous allons ainsi proposer plusieurs améliorations et nouvelles techniques dans les modules de la chaine Intra et le filtre anti-blocs. Nous allons donner des perspectives pour les autres modules d'estimation et de compensation de mouvement et l'encodage entropique. Cette contribution aura comme finalité la réalisation d'une bibliothèque d'IPs pour l'implantation de l'encodeur H.264/AVC dans un schéma hybride logiciel/matériel avec une partie soft exécutée sur un processeur reconfigurable (MicroBlaze de Xilinx dans notre cas) et une partie hard (les accélérateurs proposés).

## Organisation de la thèse

Ce manuscrit s'articule en deux grandes parties : la première est un état de l'art organisé en trois chapitres et la deuxième présente nos propositions et implémentations ainsi que les résultats de simulation et de synthèse obtenus. Cette partie est organisée aussi en trois chapitres.

**Partie1 :** **Chapitre1** présente une étude sur la conception des systèmes numériques embarqués. Nous introduisons dans un premier lieu l'évolution des systèmes numériques et les circuits logiques programmables. Ensuite, nous présentons les systèmes sur puce (SoC) ainsi que le flot de conception conjointe logicielle/matérielle. Nous introduisons à la fin de ce chapitre les plateformes multi-composants reconfigurables.

**Chapitre2** présente, dans un premier temps, les aspects sur les images, les vidéos et la compression vidéo ainsi que les normes de compression. Dans un deuxième temps, nous examinerons la norme H.264/AVC d'un point de vue théorique, nous détaillons les différents modules de traitement et en particulier les modules de la chaine de prédiction Intra ainsi que le filtre anti-blocs. Ces modules seront l'objet d'une implémentation matérielle au chapitre5. Nous présentons aussi l'organisation des données dans l'encodeur H.264 pour la définition d'une stratégie de gestion de la mémoire dans le chapitre4. A la fin de ce chapitre, nous donnons des statistiques de la norme, des exemples d'implantations logicielles et matérielles de l'encodeur et du décodeur ainsi que les perspectives des successeurs de cette norme.

**Chapitre3** présente les outils de développement des systèmes embarqués temps réels. Premièrement, nous introduisons la notion du temps réels avec des exemples. Ensuite nous décrivons les langages de description des systèmes temps réels pour prendre la décision sur l'utilisation de VHDL comme langage de programmation. En fin nous étudions les outils de conception des systèmes de plusieurs compagnies, pour prendre une décision sur les outils de simulation, de synthèse et d'implantation à utiliser. Nous justifions ainsi nos choix de langages et d'outils de conception d'un système embarqué de traitement vidéo temps réel (H.264/AVC).

**Partie2 :** **Chapitre4** présente une étude et une implémentation matérielle d'un contrôleur mémoire conçu spécialement pour l'encodeur H.264/AVC. Nous commençons par la description des

deux plateformes multi-composants (ML501 et XUPV5 de Xilinx) utilisées dans nos implémentations. Nous décrivons ainsi l'architecture de la mémoire DDR2 utilisée dans les deux plateformes, en plus du contrôleur DDR2 fourni avec cette mémoire. La première contribution de cette thèse est donnée dans la deuxième partie de ce chapitre. Cette contribution consiste à l'ajout d'une couche intelligente, au contrôleur mémoire initial, chargée des calculs des adresses des données nécessaires pour les différents modules de traitement de l'encodeur H.264. Dans cette couche, nous exploitons toutes les informations sur les besoins en mémoire des modules de traitement. Ce qui en résulte un vrai contrôleur mémoire H.264/AVC.

**Chapitre5** présente des propositions d'architectures matérielles pour les différents modules de traitement dans l'encodeur H.264/AVC. Les propositions sont basées sur les principes de parallélisme et de pipelining. Le principe consiste à exploiter les caractéristiques des mémoires DDR2 et des mémoires BRAM des FPGAs pour le traitement des données sur 128bits au lieu de 32bits dans des implémentations existantes. Avec une taille de bus de 128bits nous pouvons traiter un bloc de 4×4pixels à la fois. Cette taille représente le maximum de parallélisme, dans un macrobloc 16×16pixels, autorisé par H.264 en vue de la dépendance inter-blocs dans les deux modules de prédiction (Intra et Inter). Nous exploitons cette taille maximale autorisée pour proposer des solutions matérielles pour les différents modules. Les différentes propositions sont validées par la synthèse en utilisant les deux plateformes (ML501 et XUPV5), nous comparons ainsi les résultats avec les autres implémentations existantes.

**Chapitre6** présente l'étape finale pour l'implantation d'une application de traitement vidéo dans un système embraqué (SoC). Dans ce chapitre les différents modules implémentés dans le chapitre5 sont intégrés dans un système à base de processeur MicroBlaze.

Le manuscrit conclut finalement par les principales contributions et résultats obtenus durant les années de la thèse et présente quelques perspectives.

# Chapitre 1

# Conception des Systèmes Embarqués

*Loi empirique de **Moore 1965** « Pour une surface de silicium donnée, la capacité d'intégration (le nombre de transistors dans un processeur) double tous les 18 mois ».*
*(Gordon Moore - cofondateur d'Intel)*

## 1. Introduction

Les ordinateurs et, plus généralement, les systèmes électroniques et informatiques sont désormais devenus omniprésents dans la vie de tous les jours : téléphones portables, baladeurs, PDAs, cartes de paiement, etc., et même dans des applications moins « voyantes » mais pas moins répandues (conduite assistée pour voitures, monitorage du trafic, contrôles d'accès, systèmes de sécurité, etc.). Il est presque impossible de tourner la tête sans voir au moins une application.

L'évolution des technologies de fabrication ont permis d'augmenter non seulement la capacité d'intégration, mais aussi la fréquence de fonctionnement des composants. Gordon Moore [1] a prédit dans les années 60 que la croissance de la complexité des circuits intégrés suivrait une courbe exponentielle (voir doubleraient tous les deux ans) alors que la taille des transistors diminuerait dans les mêmes proportions. En pratique, l'architecture interne des processeurs se révèle de plus en plus complexe, de sorte à améliorer et accélérer l'exécution d'un programme, ou d'un ensemble de programmes [2].

Pour répondre aux exigences du marché, avec une diversité d'applications, l'électronique a dû progressivement s'adapter pour devenir portable, puis ultra miniaturisée : c'est la naissance des systèmes (numériques) embarqués. Ces systèmes ont vu leur importance progresser au rythme de l'importance prise par les microprocesseurs. Le marché des microprocesseurs est un marché qui croît de façon exponentielle. Ce marché a aussi tiré le marché des systèmes embarqués (et des télécommunications). En effet, selon quelques statistiques [3], 90% des processeurs sont destinés à l'industrie des systèmes embarqués.

## 2. Conception des systèmes numériques

Un calcul numérique est un ensemble de calculs réalisés sur un système informatique, encore appelé système numérique (ou ordinateur). Pendant longtemps, les calculs sont effectués par les ordinateurs et entièrement pris en charge par le processeur central (CPU). Or ce processeur s'avère insuffisant dans un certain nombre de domaines. Des circuits plus efficaces ont été créés et travaillant en parallèle avec le processeur, spécialement pour des tâches bien définies.

En plus, les évolutions technologiques acquises dans le monde industriel actuel ont permis l'intégration de plus en plus de fonctions sur le même circuit intégré numérique. Aujourd'hui, on arrive à embarquer plusieurs microprocesseurs, des accélérateurs matériels, des systèmes de

communications diverses, des systèmes d'exploitation, etc. Ceci permet donc de réaliser toutes les fonctions nécessaires pour effectuer des traitements informatiques complexes sur la même puce, d'où la naissance de la notion des systèmes sur puce (SoC pour System on Chip), successeurs des circuits spécialisés ASIC (Application Specific Integrated Circuit). Dans cette première partie, nous définissons les différents processeurs et les différents circuits programmables.

### 2.1. Architectures à base d'un processeur

Les GPP (General Purpose Processor) sont des processeurs dont l'image est souvent associée à celle de l'ordinateur personnel (PC). Leur popularité est liée à leur flexibilité, leur simplicité d'utilisation et leur puissance de calcul [2]. L'association processeur-PC n'est toujours vraie, puisque les processeurs destinés aux systèmes embraqués occupent la plus grande partie du marché des processeurs. Par contre, les processeurs destinés aux systèmes embarqués sont différents des processeurs destinés aux ordinateurs grand public. D'abord le concept CISC (Complex Instruction Set Computer) a subi au cours des années une « cure d'amincissement » afin d'arriver à des architectures RISC (Reduced Instruction Set Computer) dans lesquelles les instructions et les modes d'adressage complexes étaient bannis [4]. Des principes de simplification ont été mis en œuvre (comme par exemple le codage uniforme des instructions) afin de permettre notamment d'avoir des compilateurs fournissant de bonnes performances [3].

#### 2.1.1. Les processeurs d'usage général (GPP)

La puissance de calcul et la flexibilité sont les objectifs premiers d'un GPP. Ils sont atteints grâce à une fréquence de fonctionnement élevée et des architectures matérielles et logicielles adaptées : super scalaire, fort pipeline, prédiction de branchement, caches multi-niveaux, instructions spécialisés (par exemple l'instruction multimédia : MMX pour le Pentium et AltiVec pour le PowerPC) et extension SIMD (Single-Instruction Multi-Data) [5]. Cependant, un GPP n'est pas bien adapté aux applications embarquées du traitement du signal et de l'image, et cela à cause de deux principales raisons :

- Dissipation d'énergie généralement élevée dans les GPP ;
- Les GPP ne sont pas adaptés au temps réel pour les applications complexes.

On retrouve dans la plupart des systèmes sur puce des processeurs à usages généraux comme par exemple PowerPC, ARM, MIPS, ST-Microelectronics, etc. Ces processeurs ont des caractéristiques plus ou moins rapprochées. Ils sont destinés à effectuer des traitements ordinaires, sans qu'ils aient des performances optimales. La différence remarquable de ces processeurs utilisés dans les systèmes sur puce par rapport aux processeurs ordinaires, de la famille 80xx d'Intel et des 68xx de Motorola par exemple, réside dans le fait qu'ils utilisent du pipeline et de la mémoire cache afin de limiter le nombre d'accès à la mémoire centrale du système et de minimiser, par conséquent, l'utilisation du bus et de la consommation d'énergie du système [6].

#### 2.1.2. Les processeurs de traitement du signal

Certains systèmes numériques intègrent également des processeurs à usages spécifiques pour des applications de traitement du signal et des images (DSP pour Digital Signal Processors). Ces processeurs spécialisés sont surtout destinés aux traitements d'images de type «bas niveau» pour

lesquels les calculs sont réguliers (les traitements sont identiques pour toutes les données d'entrée) et doivent être réalisés à hautes fréquences [7].

Les DSP sont des processeurs simplifiés relativement semblables aux GPP. Leur particularité essentielle est qu'ils sont conçus pour effectuer des calculs en temps réel et intègrent donc de nombreux opérateurs. En effet leur architecture interne est dimensionnée pour le calcul intensif. Suivant les modèles, ils permettent de réaliser des opérations sur des nombres soit à virgule fixe, soit à virgule flottante. Ces processeurs permettent également un accès rapide aux données par des adressages particuliers [8]. Leur jeu d'instructions est souvent plus réduit que celui d'un processeur traditionnel et peut être programmé soit en assembleur, soit avec un compilateur C dédié. Ces processeurs ont la possibilité de réaliser plusieurs instructions en parallèle et possèdent des opérateurs de calculs arithmétiques flottants très performants. Dans un DSP, les transferts entre les périphériques et la mémoire d'une part et entre la mémoire interne et la mémoire externe d'autre part, sont réalisés par le DMA (Direct Memory Access), qui permet de décharger le cœur de calcul, et de paralléliser les deux opérations (accès à la mémoire et le traitement). Des interfaces série, PCI, mémoire sont également intégrées au DSP.

Les DSP sont accompagnés d'outils de développement performants. Les débogueurs fournissent des informations de bas niveau pour valider rapidement une application. Les fabricants fournissent en général des compilateurs capables d'optimiser fortement un programme et d'informer le développeur sur le résultat en ajoutant des commentaires dans le programme (boucle optimisée, nombre de cycles, instructions utilisées). Des simulateurs permettent d'obtenir des informations supplémentaires sur l'exécution d'une application (charge de calcul, comportement des mémoires caches, temps d'exécution) [2].

Un DSP est utilisé dans plusieurs domaines d'application qui nécessitent l'utilisation de filtres numériques ou adaptatifs, des FFTs, dans l'instrumentation (analyse transitoire, spectrale), dans le domaine médical (monitoring, échographie, imagerie médicale), dans les applications de contrôle (asservissement, robotique), ainsi que dans le multimédia, l'imagerie, le militaire (radar, guidage de missile), les télécommunications (modems radio, cryptage de données, répéteurs de ligne) et les applications grand public (automobile, électroménager), etc.

### 2.1.3. Les processeurs reconfigurables

En plus des GPP et des DSP, on retrouve également des processeurs reconfigurables qui peuvent intégrer des unités fonctionnelles selon le besoin. Ces processeurs peuvent être taillés sur mesure selon les besoins de l'application et de gagner par conséquent énormément de ressources. En effet, un processeur softcore (synthétisable) est un processeur implémenté sur un système reprogrammable comme un FPGA (Field Programmable Gate Array). Si on intègre plusieurs processeurs sur la même puce, on parle alors de système sur puce programmable (System on Programmable Chip ou SoPC) [6].

Architecture très flexible de par sa nature, une implémentation softcore d'un processeur peut être reconfigurée en tout temps contrairement à un processeur dit hardcore dont le cœur dispose de sa propre puce qui ne peut être modifiée [9]. Un processeur softcore s'adapte donc aux besoins de ses développeurs et aux contraintes matérielles (périphériques, performances, consommation, etc.). Toutefois, ces performances sont inférieures à celles d'un processeur hardcore. Un softcore est en

général programmé dans un langage de description matérielle comme le VHDL (acronyme de VHSIC-HDL : Very High Speed Integrated Circuit Hardware Description Language) ou le Verilog. Parmi les processeurs softcore les plus connus, on peut citer : le Microblaze de Xilinx[1], le NIOS de la société Altera[2], Openrisc d'Opencore[3] et le LEON de chez Gaisler Research[4].

### 2.2. Les circuits logiques programmables

Un circuit logique programmable, ou réseau logique programmable, est un circuit logique intégré qui peut être reprogrammé après sa fabrication. Il est composé de nombreuses cellules logiques élémentaires librement assemblables. Un circuit logique programmable permet la réalisation de circuits logiques à grand nombre de variables d'entrée et de sortie, selon la spécification de l'utilisateur et en utilisant un même support matériel.

Les premiers circuits programmables ont été les PROM (Programmable Read Only Memory). Dans ces circuits, le bus d'adresses est utilisé comme entrée et le bus de données comme sortie du circuit logique. Les limitations de ce type de composants pour la réalisation d'architectures logiques ont vite trouvé leurs limites et nous avons vu apparaître les FPLA (Field Programmable Logic Array).

Actuellement, on trouve différentes familles de circuits programmables tels que les PLD (Programmable Logic Device), les CPLDs (Complex Logic Programmable Device) et les FPGAs [10]. La différence entre ces composants est structurelle. Les CPLDs sont des composants pour la plupart reprogrammables électriquement ou à fusibles, peu chers et très rapides (fréquence de fonctionnement élevée) mais avec une capacité fonctionnelle moindre que les FPGA. La figure suivante montre les différentes familles des circuits programmables :

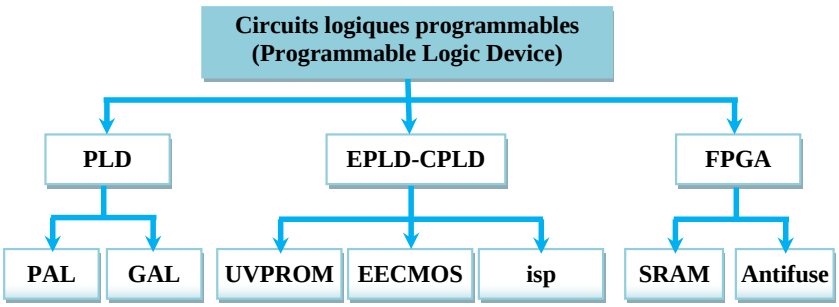



Figure 1.1. Les différents circuits programmables.

#### 2.2.1. Les PLDs et les CPLDs

Les PLDs sont des circuits, disponibles sur catalogue, exclusivement numériques. Les PLDs sont proposés avec deux grandes familles : PAL (Programmable Array Logic), et GAL (Généric Array Logic). Les PLDs présentent les avantages suivants : personnalisation et mise en œuvre simple, outils de développement peu coûteux (souvent gratuits), pas de retour chez le fabriquant, en plus ces circuits sont idéaux pour un prototypage rapide. Les inconvénients des PLDs peuvent être résumés à : une consommation globale accrue par les circuits de configuration, les PLDs sont peut-être chers pour de

[1] Xilinx inc. www.xilinx.com.
[2] Altera inc. www.altera.com.
[3] www.opencore.com.
[4] Aeroflex Gaisler. www.gaisler.com.

grandes séries, les primitives du fabricant pouvant être complexes, la performance globale est dépendante pour beaucoup de l'outil utilisé.

Toujours dans un souci de performances, les PALs ont été regroupés sous le terme de SPLD (Simple Programmable Logic Devices). Pour encore augmenter les capacités de l'architecture, plusieurs SPLDs peuvent être intégrés dans la même puce pour constituer un CPLD [11]. En effet, les CPLDs sont des assemblages de macro-cellules programmables « simples » réparties autour d'une matrice d'interconnexion. Les temps de propagation de chaque cellule sont en principe prévisibles [9]. Les CPLDs sont proposés avec deux grandes familles : Les EPLDs (Erasable Programmable Logic Device) qui sont des circuits programmables électriquement et effaçables aux ultra-violet, et les EEPLDs (Electrical Erasable Programmable Logic Device) qui sont des circuits effaçables électriquement. Avec les CPLDs, les concepteurs ont atteint la densité d'intégration maximale pour ce type de circuits, de nouvelles études ont conduit à l'apparition des FPGAs.

### 2.2.2. Les FPGAs

Les FPGAs, ou bien les réseaux logiques programmables, sont des composants électroniques programmables de la famille des PLDs. Un FPGA est un ensemble de blocs logiques élémentaires que l'utilisateur peut interconnecter pour réaliser les fonctions logiques de son choix. La densité des portes est importante et sans cesse en évolution. L'avantage d'un FPGA est leur grande souplesse dans leur technologie permettant une réutilisation à volonté et en un temps très court (quelques millisecondes) dans des algorithmes différents. Le progrès technologique permet de faire des composants toujours plus rapides et à plus haute intégration, autorisant la programmation d'applications importantes [3].

Grâce à l'évolution des procédés de fabrication, ces composants peuvent actuellement supporter des applications complexes. Ils sont constitués d'un réseau de blocs logiques, de blocs mémoires, de blocs dédiés et d'entrées/sorties. L'ensemble est relié par un réseau d'interconnexions programmable. Les blocs logiques permettent de réaliser des opérations avec quelques variables à travers une LUT (Look Up Table) [7]. Le résultat peut être éventuellement stocké dans un registre, les blocs RAM permettent d'implanter des mémoires adressables et des FIFO (First In, First Out), les blocs dédiés permettent de réaliser facilement de nombreuses opérations de traitement (blocs DSP), de gérer l'horloge, ou des interfaces de communication (RocketIO, Ethernet, PCI Express). Les nombreux ports permettent également de connecter des périphériques de la plateforme matérielle à base de FPGA.

Les FPGA se programment grâce à leurs LUT et leur réseau d'interconnexion. La programmation se fait avec un langage de programmation hardware tel que le VHDL ou bien le Verilog. L'outil de développement transforme cette description en un fichier de configuration du FPGA en plusieurs étapes : le HDL (Hardware Design Language) doit d'abord être synthétisé (transformé en éléments logiques de base), puis les éléments doivent être placés sur le composant (placement) et enfin interconnectés (routage) [6].

La Figure1.2 montre la structure interne d'un FPGA de type matrice symétrique. Il s'agit de l'architecture que l'on retrouve dans les FPGA actuels. L'utilisateur peut programmer la fonction réalisée par chaque cellule (CLB: Configurable Logic Block). On programme aussi les interconnexions entre les cellules. Les FPGA les plus récents sont configurables en une centaine de millisecondes. Les FPGA sont utilisés pour un développement rapide et bon marché des ASIC.

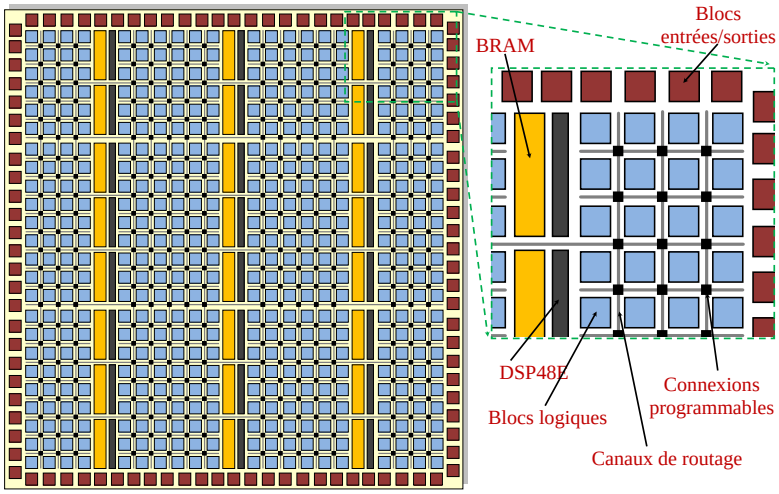



Figure 1.2. Architecture conceptuelle d'un FPGA (Architecture, Interconnexions et CLB).

### 2.2.3. Les ASICs

Un ASIC est un circuit électronique dédié. Il peut regrouper sur une même puce tous les éléments actifs nécessaires à la réalisation d'une fonction ou d'un ensemble électronique. Ce composant n'est pas modifiable. Comparé à un FPGA, il offre une capacité d'intégration plus élevée, une vitesse plus rapide et un coût plus faible. En effet, les ASIC permettent l'intégration d'un nombre très important d'opérateurs de calculs et de mémoire. Cependant, le développement d'un ASIC est très long. Il s'agit d'un circuit intégré conçu exclusivement pour le projet ou l'application qui l'utilise. Grâce aux ASIC les concepteurs peuvent désormais regrouper dans un seul boîtier toutes les fonctions nécessaires à leur application en occupant une surface comparable à celle d'un simple composant. Dans un produit, le circuit intégré est un élément porteur d'avenir, fiable, performant et peu encombrant, de plus il assure la confidentialité du savoir-faire [3].

Les ASICs n'ont d'intérêt que pour les grandes séries pour compenser les coûts de développement et de production. Il n'est pas possible de modifier le composant après sa production, une erreur peut provoquer la nécessité de repasser par la phase de développement et coûte par conséquent très cher [2]. Dans l'industrie, grâce aux ASICs, il est possible de compléter (ou remplacer) des fonctions mécaniques, pneumatiques, hydrauliques ou électriques, abaissant les coûts de production. En particulier, il permet de concevoir les circuits électroniques de liaison entre le monde physique : capteur (analogique), et l'unité de traitement d'information et de commande. Les ASICs sont utilisés aussi dans l'implémentation avec une très haute densité des algorithmes de traitement d'images [8].

## 3. Les systèmes numériques embarqués

De nos jours, les systèmes électroniques embarqués ont été introduits dans de nombreux domaines d'applications tels que l'automobile, l'avionique, les systèmes multimédia, les appareils électroménagers, les terminaux de communication sans fil, etc. Les concepteurs doivent donc gérer les développements de systèmes électroniques embarqués multidisciplinaires et multiprocesseurs. L'intégration électronique permettra, à titre d'exemple, de combiner les fonctions d'un téléphone, d'un navigateur Internet, d'un appareil photo numérique, d'un écran couleur, de lecteurs multimédia et d'un PDA (Personal Digital Assistants) au sein d'un unique objet portable. Les systèmes électroniques mobiles d'aujourd'hui et de demain, dit de $X^{eme}$ génération, ont un marché potentiel.

L’architecture générale d’un système sur puce intègre trois principaux types de composants qui sont : les unités de traitements, les mémoires et les systèmes de communication. On peut retrouver également des adaptateurs (wrapper ou bien bridge) qui servent à traduire les protocoles de communication pour connecter des composants ayant des entrées/sorties incompatibles [6].

### 3.1. Codesign : Conception conjointe logicielle/matérielle

La conception des systèmes contenant des composants hardware et software n’est pas un problème nouveau. Les techniques classiques de conception de systèmes imposent de séparer le hardware du software dans le cycle de conception. En outre, le consensus grandissant pour un modèle de systèmes informatiques «en couches» semble éviter aux développeurs software de telles préoccupations. Ainsi, les concepteurs de hardware spécifient leurs produits sans totalement apprécier les besoins, en ressources matérielles, des logiciels qui s'exécuteront dessus. De leur côté, les concepteurs de logiciels ne tirent pas suffisamment profit des ressources matérielles de la plate-forme hardware sur laquelle seront implantés leurs produits (Le software et le hardware sont deux activités qui ont toujours été menées de manières relativement indépendantes). Finalement, la phase d'intégration des deux parties nécessite souvent des modifications parfois majeures du logiciel et/ou du matériel. Cette approche présente peu de flexibilité dans l’évaluation des différentes options de conception et de répartition du hardware et du software [12].

La tendance actuelle, principalement lorsqu'il s'agit de systèmes temps réel complexes (systèmes critiques embarqués par exemple), est de tenir compte dès la phase de spécification des interactions entre les deux parties. Ainsi, la séparation est effectuée le plus tard possible dans le cycle de conception tendant à unifier la spécification pour qu'elle englobe le hardware et le software. Cette approche est nommée "hardware/software codesign", ou "codesign". Le système est conçu à partir d'une spécification unique décrivant son architecture et/ou son comportement. Après la phase de spécification survient une phase de partitionnement du système ayant pour but de décomposer ce dernier en partitions comportant trois parties :

- Une partie matérielle implantée sous forme de circuits hardware. Cette partie peut être un IP récupéré à partir d’une bibliothèque ou bien accélérateur matériel réalisé spécialement pour une tâche précise ;
- Une partie logicielle implantée sous forme d'un programme exécutable sur un processeur. Le processeur peut être un GPP ou bien un processeur reconfigurable (configuré à l’avance selon les besoins de l’application) ;
- Une interface de communication entre les deux parties.

Les partitions obtenues doivent ensuite être vérifiées et validées avant de passer à la phase de synthèse et de mise en œuvre. Un feed-back permet de revenir à la phase de partitionnement en affinant différemment les poids associés aux contraintes si les partitions obtenues ne s'avèrent pas satisfaisantes [11].

#### 3.1.1. Flot de la conception conjointe logicielle/matérielle

Le développement de systèmes complexes fait de plus en plus appel à des composants logiciels et matériels spécifiques. La conception conjointe logicielle/matérielle apporte des solutions à ce type de développement. Elle repose sur un ensemble d’étapes qui permettent de synthétiser, à partir d’une

description du système au niveau fonctionnel, un système sur puce intégrant des composants logiciels et matériels qui respectent les contraintes de conception (temps et surface) imposées au départ. Un flot de conception standard est généralement constitué de trois étapes principales : la spécification, le partitionnement, et la vérification conjointe HW/SW (Figure 1.3).

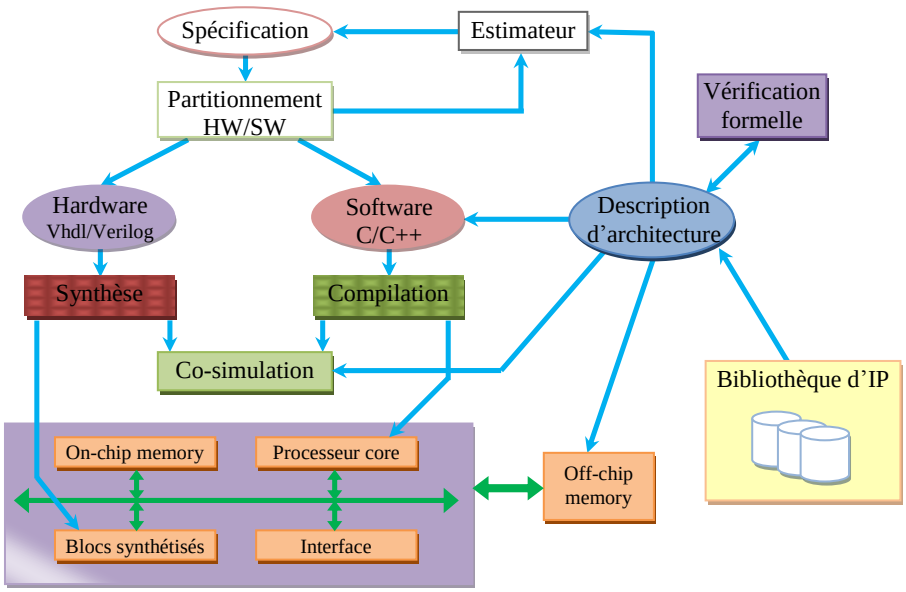



Figure 1.3. Flot de conception standard [11]

D'après cette figure, les étapes principales de la conception conjointe logicielle/matérielle peut être résumées comme suite :

1. Le point de départ est une spécification au niveau du système de l'application à réaliser. Cette spécification permet de décrire le comportement fonctionnel du système sans tenir compte de l'aspect architectural de celui-ci. A ce niveau de la conception, on ne pourra en aucun cas obtenir des résultats pertinents au niveau des performances et timing.
2. L'étape qui suit la vérification fonctionnelle du modèle au niveau système, sera donc le partitionnement logiciel/matériel. A ce stade de conception, on intègre les détails architecturaux de communication et d'ordonnancement des opérations. Plusieurs tentatives de simulation seront effectuées pour choisir la meilleure répartition entre les parties logicielles et les parties matérielles.
3. Après le découpage logiciel/matériel de la spécification d'entrée, l'étape de synthèse appelée aussi d'implémentation aura lieu. Dans cette étape, nous obtenons une description au niveau RTL pour la partie matérielle et un code source (e.g. C ou C++) pour la partie logicielle du système. Bien évidement, la vérification et la validation de la fonctionnalité du modèle ainsi généré devrait être faite. A ce stade de conception, nous pouvons effectuer des analyses de performances de l'architecture au niveau cycle et au niveau bit à travers des co-simulations.
4. La dernière étape du flot de conception consiste en premier lieu à la synthèse logique de la partie RTL du système. Ensuite les fonctions logiques qui ont été synthétisées seront placées et routées sur le plan de la puce. Ce processus est réalisé à l'aide des outils de synthèses commerciaux comme par exemple : Synplify [13], XST [14], Leonardo Spectrum, etc. Quant à la partie logicielle du système, elle sera compilée pour générer une image hexadécimale. Enfin, une co-simulation sera établie après la synthèse logique afin d'obtenir les performances en surface et en

consommation d'énergie. Une fois l'architecture est validée, nous passons aux tests réels sur les cartes reconfigurables de type FPGA [6].

Plusieurs outils et compagnies ont adopté cette approche de codesign par le fait de sa simplicité de synthèse et d'intégration de nouveaux composants. Parmi ces outils de développement notons celui proposé par Xilinx (EDK pour Embedded Development Kit).

### 3.1.2. Les composants virtuels (IPs)

Les outils de développement et de conception, des circuits spécialisés, sont de plus en plus performants malgré l'augmentation considérable de la capacité des composants matériels. Cependant il est indispensable de réutiliser des conceptions déjà réalisées dans ces outils de développement [2]. En effet, afin de réduire le temps de conception et de prototypage des systèmes sur puce, une approche de réutilisation des composants a été introduite. Cette approche consiste à reprendre des modèles de composants au niveau RTL qui sont considérés comme étant des propriétés intellectuelles (IP) [5]. Les IPs peuvent être définis comme des composants virtuels, ce sont des circuits génériques ainsi que des outils associés que l'on peut trouver sur le marché ou bien sous une forme téléchargeable (Open Source).

Un IP peut être une fonction de base ou bien un bloc plus complexe. L'intégration de cœurs de processeurs est également possible, tel que le MicroBlaze ou bien le NIOS. Ces processeurs sont relativement simples, et personnalisables en fonction de l'application. L'assemblage des IPs permet de décrire rapidement des applications complexes. On peut utiliser des IP sans les avoir conçus, en ayant uniquement une description haut-niveau (vue externe et fonctionnelle) de leur comportement. Un composant IP peut apparaître sous différentes formes :

- **IP Logiciel** (softcore) : Le composant est livré sous sa forme HDL synthétisable. Son principal avantage est sa portabilité. La propriété du fichier source est en soi la meilleure documentation, on peut ainsi maintenir le produit pendant des années et éventuellement modifier à volonté la source et même changer de technologie cible. L'inconvénient majeur est qu'il ne peut être prédictif en termes de superficie, puissance et temps. Le travail d'optimisation du circuit final est à la charge de l'intégrateur du système.
- **IP Matériel** (hardcore) : Dans ce cas, la description du bloc IP est au niveau porte logique, optimisée pour une technologie particulière (optimisation garantie). Cela englobe la netlist entière, le routage et l'optimisation pour une librairie technologique spécifique et un layout personnalisé. L'inconvénient est qu'il est moins flexible car le processus est dépendant de la technologie, par contre il a l'avantage d'être prédictif.
- **IP Firm** (firmcore) : Le bloc IP firmcore offre un compromis entre les deux autres IPs, plus flexible que le hardcore, plus prédictif en termes de performance et de surface que le softcore (par exemple une interface USB où la vitesse de la norme ne pourra être assurée que dans une technologie bien définie). En général, le travail de synthèse HDL est déjà réalisé pour une technologie cible donnant lieu à une description par netlist (format EDIF par exemple).

Dans un système complexe, les IPs communicants à travers un réseau de communication sont aussi susceptibles d'être réutilisables après reconfiguration. La conception se résume à l'assemblage de l'ensemble des unités de calculs qui ont été choisies au préalable et les faire communiquer à travers un

réseau de communication. L'inconvénient majeur de cette approche réside dans le fait que le concepteur est confronté très souvent aux problèmes d'adaptation des entrées/sorties des composants réutilisés avec le réseau d'interconnexions. Ce problème d'adaptation amène le concepteur à intégrer des adaptateurs de bus autrement dit, des interfaces de bus entre le réseau de communication et les composants du système [6].

### 3.1.3. Les accélérateurs matériels

Les avancées actuelles dans la technologie des semi-conducteurs et des méthodologies de conception permettent le développement des SoC. Les derniers circuits FPGA permettent également le développement de systèmes complets. La tendance actuelle est donc à l'assemblage dans une même puce de plusieurs composants éventuellement hétérogènes pour répondre au mieux aux exigences des systèmes multimédia embarqués. Ces composants peuvent être aussi bien des cœurs de processeurs, des cœurs de DSP, et même des accélérateurs matériels.

Un accélérateur matériel consiste à confier une fonction spécifique effectuée par le processeur à un circuit dédié qui effectuera cette fonction de façon plus efficace. En effet, les accélérateurs matériels représentent des unités de calculs intensifs. Ils peuvent être des composants matériels issus de la synthèse de haut niveau (HLS), des propriétés intellectuelles (IPs) réutilisables ou bien un composant matériel conçu spécialement pour réaliser une tâche précise. L'utilisation des IPs est une voie très exploitée de nos jours dans la conception des systèmes embarqués à cause de sa simplicité de conception et de son efficacité dans le flot d'intégration.

Les accélérateurs matériels peuvent intégrer des fonctions de calculs (FFT, encodeur vidéo, etc.), des contrôleurs d'entrée/sortie (UART, VGA, PS2, etc.), des réseaux de communication (bus partagé, bus matriciel, réseau sur puce, etc.) et également des mémoires qui peuvent être considérées comme étant des IPs. Plusieurs méthodologies de conception des systèmes embarqués ont été proposées pour la réutilisation des IPs [9]. Sur le site [15] plusieurs accélérateurs matériels sont proposés, allant d'une simple fonction jusqu'aux processeurs reconfigurable et même des accélérateurs pour des systèmes de traitement complets (exemple la compression d'image JPEG). Ces blocs IPs sont open source et peuvent ainsi être modifiés à volonté.

## 3.2. Les systèmes sur puce (SoC)

Un système monopuce, appelé encore SoC ou système sur puce, désigne l'intégration d'un système complet, avec de nombreux composants, sur une seule puce. Un Soc peut intégrer, aujourd'hui, plusieurs millions de transistors sur une surface de silicium de quelques millimètres carré [16].

### 3.2.1. Définitions d'un SoC

Le terme SoC est devenu très populaire dans le milieu industriel malgré l'absence d'une définition standard [17]. Certains considèrent qu'un circuit complexe en fait automatiquement un SoC, mais cela inclurait probablement chaque circuit existant aujourd'hui [18]. Une autre définition considère le système embarqué comme un système électronique et informatique autonome ne possédant pas des entrées/sorties standards comme un clavier ou un écran d'ordinateur. Une définition plus appropriée de système monopuce serait : un système complet sur une seule pièce de silicium, résultant de la cohabitation de nombreuses fonctions déjà complexes en elles mêmes : processeurs, DSP, mémoires,

bus, convertisseurs, blocs analogiques, etc. Il doit comporter au minimum une unité de traitement de logiciel (un processeur qui exécute un logiciel dédié pour la réalisation d'une fonctionnalité précise) et doit dépendre d'aucun (ou de très peu) des composants externes pour exécuter sa tâche, en conséquence, il nécessite à la fois du matériel et du logiciel [19]. En plus un système embarqué n'exécute pas une application scientifique ou commerciale.

La figure suivante présente les caractéristiques principales d'un système embarqué typique. Il se compose d'un cœur de processeur (CPU), d'un processeur de signal numérique (DSP), de la mémoire embarquée, et de quelques périphériques tels qu'un DMA et un contrôleur d'E/S. Le CPU peut exécuter plusieurs tâches via l'intégration d'un OS (système d'exploitation). Le DSP est habituellement responsable de décharger le CPU en faisant le calcul numérique sur les signaux de provenance du convertisseur A/N.

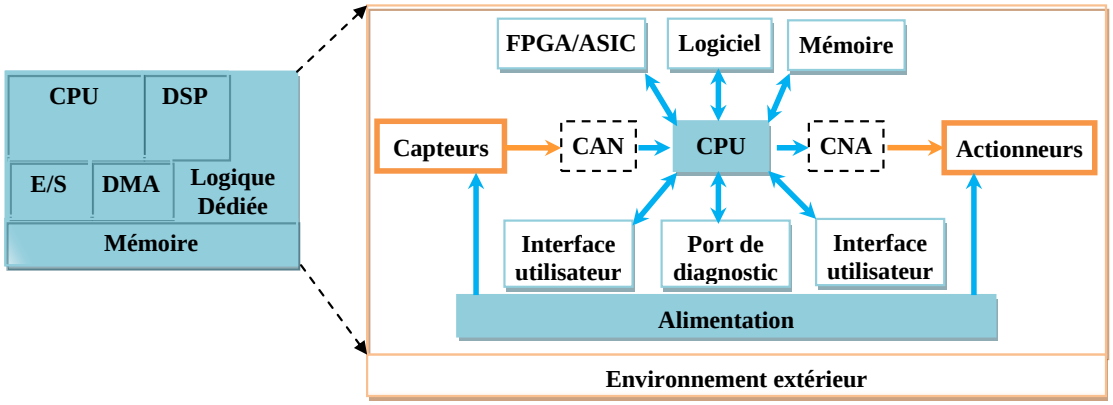



Figure 1.4. Système embarqué (SoC) typique.

### 3.2.2. Les contraintes liées aux systèmes embarqués

Les contraintes lors de la construction d'un système embarqué sont relatives au temps, imposés par l'interaction du système avec son environnement et aussi au caractère limité et/ou périssable des ressources dont dispose le système (énergie, mémoire, etc.). A ces contraintes s'ajoutent des contraintes de sûreté de fonctionnement plus ou moins importantes selon le type d'utilisation du système embarqué. En plus, un système embarqué exécute des tâches prédéfinies et a un cahier des charges contraignant à remplir, qui peut être de l'ordre de coût (le système ne doit pas être cher tout en tenant compte de la performance), d'espace compté ou bien les contraintes physiques (le système doit présenter un faible encombrement et un faible poids) ; de puissance de calcul ; de consommation énergétique (la plus faible possible), etc. Des choix doivent donc être réalisés pour privilégier un ou plusieurs de ces critères. Par exemple, pour un ordinateur portable, l'autonomie est souvent négligée par rapport aux contraintes de performances et de taille. En effet, un ordinateur peu performant perd tout son intérêt, d'autant plus qu'il est souvent possible de travailler à proximité d'une prise de courant. Pour un téléphone portable au contraire, l'intérêt étant de pouvoir téléphoner de partout, l'autonomie est un critère primordial [6].

Dans tous les cas, les ressources de l'architecture choisie doivent être économisées, les calculs doivent être optimisés, la place des données et des programmes minimisée. La miniaturisation et les gains en puissance des unités de calcul permettent de plus en plus de répondre à ces attentes mais les besoins

augmentent énormément et les futures applications nécessiteront encore une gestion optimisée des ressources pour être fiables dans un contexte embarqué.

### 3.2.3. Les SoC et les SoPC

L'approche SoC a été créée dans un premier temps pour le développement d'ASIC mais a été étendue pour le développement de FPGA. On parle alors de SoPC pour System on Programmable Chip. Le SoPC (acronyme a été inventé par Synopsys) correspond à l'intégration d'un ou plusieurs cœurs de processeurs et de ses périphériques sur une même puce programmable de type FPGA. Le logiciel est alors situé soit dans une mémoire du circuit FPGA si l'empreinte mémoire le permet, soit dans une mémoire externe le plus souvent [4]. Les deux approches peuvent être résumées à :

**L'approche SoC** (généralement en technologie ASIC) : répond aux besoins de performances et d'intégration mais elle est peu adaptée à l'évolutivité des systèmes et reste réservée aux grands volumes de production. En effet, un SoC peut être retenu pour les applications destinées au grand public. Il permet de meilleures performances en termes de consommation, de vitesse et de surface, mais, la fabrication et le test sont des étapes longues et coûteuses. De plus, un SoC est figé et n'est donc pas réutilisable pour une autre application.

**L'approche SoPC** (technologie FPGA) : Spécialement pour résoudre les problèmes de développement et les prototypages rapides, ainsi que les composants reconfigurables en quelques « ms » et à volonté. Il permet donc un développement et prototypage rapide du système. En contrepartie, la densité d'intégration est généralement moindre (~10 Millions de portes) et la consommation d'énergie est plus grande avec une performance plus faible que celle du SoC.

Les composants FPGA permettent d'implanter deux types de SoPC. Le premier type de SoPC utilise uniquement des IP soft de processeur tels que le MicroBlaze et le NIOS. Le nombre de processeurs 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 processeurs (PowerPC pour Xilinx, ARM pour Altera). Ceci n'empêche en rien l'utilisation de processeurs Soft (netlist). Ces deux types de SoPC utilisent les mêmes flots de conception. Chez Xilinx, tout le flot de conception est regroupé sous la forme d'un seul environnement dit Xilinx Platform Studio (XPS) [20].

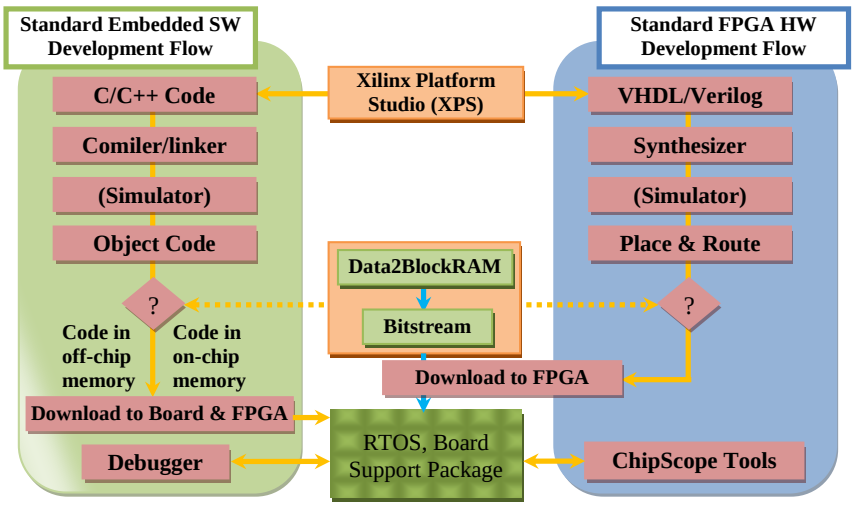



Figure 1.5. Flot de conception Xilinx [6].

### 3.3. Communications sur puce

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 et pour cette raison plusieurs types de communication sur puce ont été proposés. Un système de communication permet d'établir des connexions entre tous les composants dans un SoC. Il se présente sous forme de bus ou bien de réseau de communication. La plupart des systèmes embarqués sont basés sur des architectures de bus [21]. Quant aux réseaux, on les retrouve dans certaines architectures complexes, qui nécessitent une bande passante considérable [22].

#### 3.3.1. Les différents types des communications sur puce

**Les connexions point-à-point** : permettent d'établir des liaisons directes entre les composants du SoC. Il existe deux types de connexions point à point avec ou sans mémoire partagée. Généralement dans le domaine des SoC, on utilise des FIFOs pour la réalisation de ce type de connexion, la profondeur de la FIFO est déterminée par l'utilisateur ou bien par l'outil de conception lors de la phase d'exploration d'architecture.

**Les bus** : Un bus représente un canal unique de communication. Ce canal supporte le transfert de toutes les données entre les différents composants du SoC. Une architecture de bus intègre deux principaux éléments qui sont le canal et l'arbitre. A l'heure actuelle, on retrouve une grande variété de bus standards, plus ou moins performants, offerts par les industriels. Il existe deux types d'architectures de bus, des architectures dites partagées et des architectures matricielles.

- Les bus partagés sont très répandus par leur souplesse et légèreté dans l'occupation de ressources, mais l'ennui dans l'utilisation des bus partagés réside dans le fait qu'on est limité par la bande passante du système. Le bus partagé intègre un seul et unique arbitre qui sera responsable de la gestion des accès au bus. Dans ce type d'architectures, à chaque instant, un seul composant peut avoir accès au bus.
- Contrairement aux architectures partagées, les architectures matricielles (crossbar) augmentent davantage le parallélisme du système au détriment des ressources matérielles allouées. Les bus matriciels nécessitent des ressources matérielles considérables. Il existe deux topologies différentes dans la conception des bus matriciels : des architectures distribuées et des architectures centralisées. La différence entre les deux architectures réside dans la façon de gérer l'arbitrage du bus. Actuellement, les concepteurs de SoC ont plus tendance à s'orienter vers des conceptions de bus matriciels.

**Les réseaux sur puce (NoC)** : Comme le problème majeur de la communication revient toujours à la limitation en bande passante du bus qui reste plus ou moins insuffisante pour la plupart des applications complexes, les réseaux sur puce (NoC : Network-On-Chip) ont été introduits afin d'y remédier à cette limitation et d'intégrer davantage de nouveaux processeurs et de composants matériels. Un réseau sur puce est constitué d'un ensemble de composants et de routeurs. Les routeurs sont reliés entre eux via des canaux de communication. Cependant, l'inconvénient majeur de ce type d'architecture revient à l'existence de blocages (deadlocks), dus à des accès simultanés à des

ressources partagées, et à la difficulté d'analyser les performances effectives du réseau (latence, deadlocks, etc.) [6].

### 3.3.2. Exemple : Le Bus CoreConnect

Spécialement pour des systèmes sur puce, IBM a proposé l'architecture CoreConnect composée de trois bus qui permettent la connexion : de cœurs de processeur, de logique dédiée, et différents autres accélérateurs matériels. Les trois bus de CoreConnect sont :

- **Processor Local Bus (PLB)** : Généralement, le PLB fournit un chemin de données large bande. En effet il est utilisé pour connecter des composants tels que les cœurs de processeurs, les interfaces avec les mémoires externes, et les contrôleurs DMA (Direct Memory Acces).
- **On-chip Peripheral Bus (OPB)** : 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.
- **Device Control Register (DCR)** : Le DCR est principalement utilisé pour accéder aux registres de l'état et de contrôle au sein des différents bus PLB et OPB.

Xilinx a repris le comportement de tous les bus CoreConnect d'IBM et les a décrits en VHDL, ce qui permet d'avoir des bus paramétrables. En effet, pour simplifier la connexion d'un module logique utilisateur à un des bus CoreConnect, Xilinx a ajouté une interface de bus préconçue paramétrable appelée IPIF (IP InterFace). L'IPIF gère les signaux du bus, le protocole de communication et plus généralement, l'ensemble des caractéristiques du bus. L'IPIF intègre aussi une interface de gestion, de la logique utilisateur, appelée IPIC (IP InterConnect). Lorsque la logique utilisateur est conçue avec un IPIC, elle peut être portable et facilement réutilisable sur d'autres 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, simplifient la tâche de connexion de la logique utilisateur. Ces fichiers sont les User Core Reference Design [23]. Un système de traitement réalisé par EDK doit être connecté le bus OPB et/ou le bus PLB, le périphérique ajouté au système (accélérateur matériel) doit être connecté à OPB ou bien PLB [20]. La figure suivante montre la connexion du bus PLB dans un système à base d'un MicroBlaze, d'une DDR2 et d'une mémoire CompactFlash.

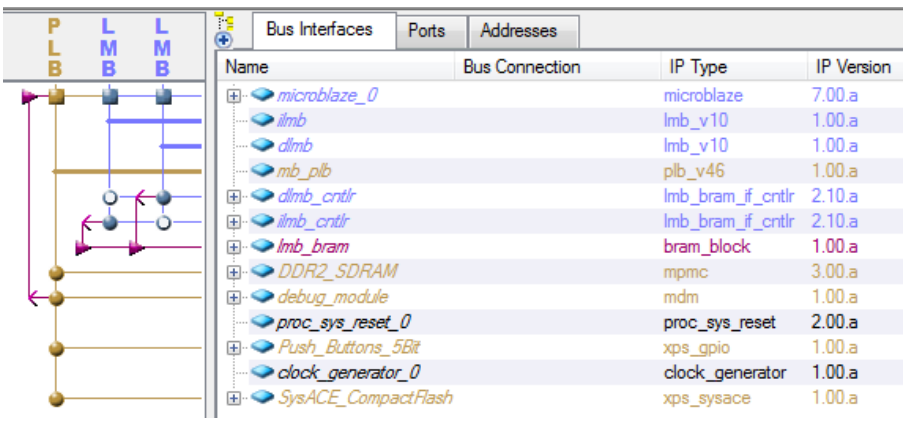



Figure 1.6. Le bus PLB dans un système à base de MicroBlaze.

## 4. Plateformes multi-composants

La maîtrise des SoCs nécessite des compétences en électronique et en informatique d'un niveau assez élevé. Pour aider les nouveaux utilisateurs de SoCs à bien maîtriser l'ensemble de la chaîne de mise en place d'applications, des plateformes de prototypage rapide existent aujourd'hui sur le marché.

Ces plateformes utilisent un gros FPGA comme cœur et disposent, en plus, de mémoires SRAM, de convertisseurs A/N et N/A, des interfaces (boutons poussoirs, voyants LED, connecteurs d'usage général, connecteurs pour debug, zone de prototypage, connecteur RS232 ainsi qu'un connecteur JTAG), d'oscillateurs, sortie VGA, etc. Généralement, la configuration des composants se fait par liaison JTAG.

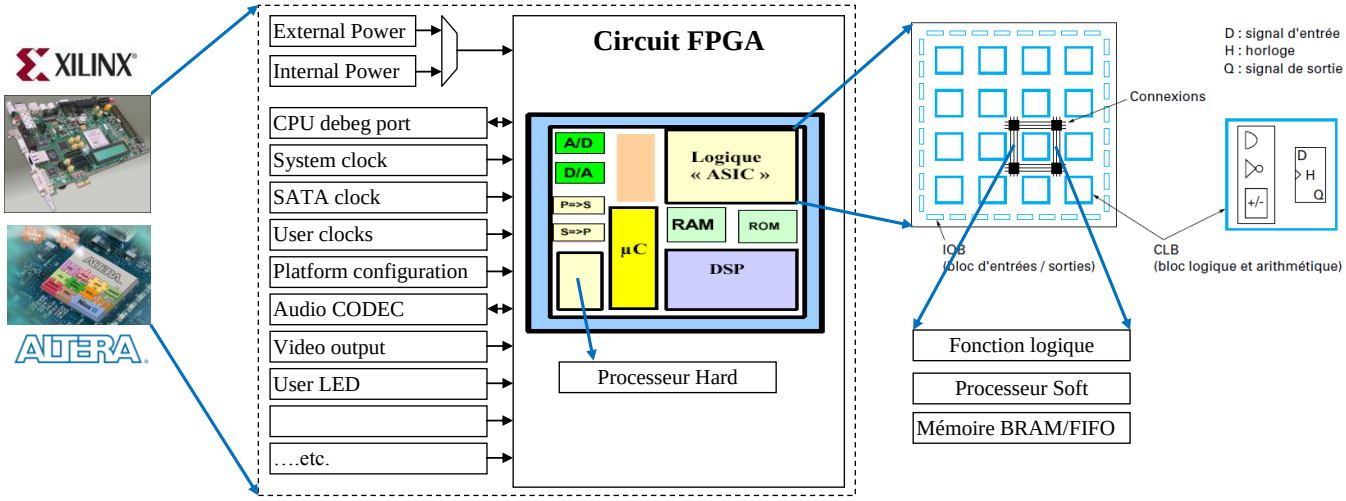



Figure 1.7. Détails d'une plateforme multi-composants.

### 4.1. Les deux grandes familles des FPGA

Actuellement, on trouve sur le marché des circuits FPGA (de faible, moyenne et haute densité) produits par les deux principaux producteurs de circuits logiques programmables : Xilinx et Altera. Sur le même marché, on trouve plusieurs autres producteurs de circuits FPGA, on peut citer à titre d'exemple : Actel, Abound Logic, Achronix, Atmel, Cypress, Lattice Semiconductor, etc. L'architecture d'une cellule logique élémentaire (CLE) varie fortement d'un producteur à l'autre. Dans cette partie de notre travail, nous décrivons les deux sociétés Xilinx et Altera, avec des exemples pour chaque famille de FPGA.

#### 4.1.1. La famille Xilinx

Xilinx, Inc. (1984) est une entreprise américaine de semi-conducteurs. Inventeur du FPGA avec un premier produit en 1985, Xilinx fait partie des plus grandes entreprises spécialisées dans le développement et la commercialisation de composants logiques programmables, et des services associés tels que les logiciels de CAO électroniques ; création des blocs IP et formation. En effet, Xilinx vend également des spécifications d'architectures ("IP Cores"). Cela peut aller de fonctions très simples comme des compteurs jusqu'à des systèmes complets comme des microcontrôleurs. Xilinx est également à l'origine du processeur softcore MicroBlaze.

Xilinx fabrique une large gamme de FPGA et de CPLD pour diverses applications. En effet, l'offre commerciale de Xilinx est découpée en plusieurs gammes : (FPGA hautes performances : gamme

Virtex, FPGA pour la fabrication en grande série : gamme Spartan et CPLD : gammes XC9500 et Coolrunner). Les plus onéreux sont les FPGA Virtex (Virtex II/pro, Virtex4 et Virtex5). Les composants Virtex5 sont disponibles, à bon prix, avec plusieurs catégories (LX, LXT, SXT, TXT et FXT). Les Virtex5 offrent une densité supérieure à 12 Méga portes logiques équivalente. Ils peuvent atteindre une fréquence de fonctionnement de 550 MHz. Ils sont fabriqués avec la technologie 65nm. Chaque CLB est constitué de 6 slices. Chaque slice contient un générateur de fonctions, une logique de traitement de la retenue (carry logic), des portes logiques arithmétiques, un large multiplexeur et deux éléments de stockages, 256bits de RAM distribuée au lieu de 64 bits dans le Virtex4, un registre de 128 bits au lieu de 64 bits dans le Virtex4. Le nombre de blocs de RAM varie entre 32 et 384 blocs. Chaque bloc RAM a une taille de 36Kbits (18Kbits pour le Virtex4) avec une largeur de bus de données maximales de 36 bits, ces mémoires sont doubles ports (Dual Port). Le nombre maximum d'I/O utilisateur est de 960. Les composants Virtex5 possèdent aussi entre 32 et 384 DSPs (DSP48E de 25×18-bit MAC à 550 MHz), jusqu'à 12 DCM (Digital Clock Manager), jusqu'à 6 PLL (Phase Lock Loop). La catégorie FXT est la seule qui dispose d'un ou bien deux processeurs embarqués type PowerPC405 (directement inclut au sein du FPGA). Dans notre travail nous utilisons deux plateformes de prototypage avec deux FPGA Vitex5 : la plateforme ML501 avec un FPGA type LX50 et la plateforme XPUV5 avec un FPGA type LX110T.

En 2009, Xilinx a lancé les FPGA-Virtex6, en technologie 40nm, avec 3 catégories différentes (LX, LXT et SXT). Même si les Virtex6 ne disposent pas de processeur PowerPC405, la dernière catégorie est optimisée, spécialement, pour les applications lourdes en traitement numérique du signal. Ensuite, en 2010, Xilinx a lancé les FPGA-Virtex7 (avec une seule catégorieXC7VxxxT) fabriqués en technologie 28nm. Afin de satisfaire à terme les applications très exigeantes en performances de calcul et en bande passante tout en limitant la puissance consommée, Xilinx va proposer dans le courant du second semestre 2011 des circuits qui couplent plusieurs FPGA Virtex7 au sein d'un même boîtier via une technologie d'interconnexion 3D. Selon Xilinx, le boîtier Virtex7 LX2000T en technologie 28 nm sera le premier FPGA multi-puce au monde et affichera une densité de portes logiques 3,5 fois supérieure au FPGA 40 nm le plus puissant actuellement disponible.

En plus du matériel, Xilinx commercialise toute une gamme d'outils de développement pour exploiter ses composants. Les principaux outils sont :

- Xilinx ISE : Environnement de développement intégré ;
- Xilinx EDK : Environnement de développement intégré ciblant les processeurs intégrés au FPGA ;
- Xilinx ChipScope Pro : Outil de débogage temps-réel des circuits FPGA.

### 4.1.2. La famille Altéra

Le concurrent de Xilinx dans le marché des FPGA est Altera avec un chiffre d'affaire de $1.29 milliard en 2006. En effet, la société américaine Altera, créée en 1983, est un fabricant de composants reprogrammables (FPGA, CPLD et ASIC) et des processeurs embarqués. Pour la gamme haute performances on trouve les FPGA : Stratix et Stratix GX (2002), Stratix II (2004), Stratix II GX (2005), Stratix III (2006) et Stratix IV et HardCopy (2008). Pour les FPGA de grande série on trouve : Cyclone (2002), Cyclone II (2004), Cyclone III (2007), Cyclone IV et Arria GX (2007), etc.

Enfin, pour les CPLD, on trouve les exemples les plus utilisés : MAX 3000A et MAX 7000 (1991), MAX II (2007) et MAX V (2010). Altera est aussi à l'origine des processeurs softcores NIOS et NIOS-II ainsi que le bus Avalon. Le bus Avalon est un bus informatique destiné à l'implémentation sur du matériel programmable (FPGA). Ses principaux atouts sont la simplicité de son architecture, une conception prévue pour optimiser l'utilisation des ressources matérielles et un support multi-maître.

### 4.2. Les processeurs embarqués

Selon plusieurs études réalisées ces dernières années, les processeurs destinés aux ordinateurs tels que nous les connaissons actuellement ne représenteront plus que 10% des processeurs fabriqués et vendus dans le monde. Les 90% qui restent sont destinés aux puces spécialisées comme celles que l'on trouve dans les téléphones mobiles, les tablettes et les tas d'appareils de plus en plus intelligents, etc. En effet, avec l'augmentation exponentielle des performances et de la densité d'intégration des portes logiques, les cœurs de processeurs sont devenus essentiels dans le développement des SoC et SoPC (Temps de mise sur le marché réduit). Un cœur de processeur est la partie matérielle qui traite les données en fonction du décodage des instructions contenues en mémoire. La grande quantité de données à prendre en compte ainsi que les contraintes en temps souhaitées, font du cœur de processeur un élément critique d'un système. Le cœur de processeur idéal doit donc fournir le maximum de performances sur une surface la plus réduite possible. Il doit être le plus flexible possible et ne pas subir de dégradation de performances en fonctions des différents environnements rencontrés (Outils de développement, taille de la cible ou électronique adjacente).

Les cœurs de processeurs peuvent être, soit des macro-cellules spécialisées, dites processeur Hardcore (PowerPC et ARM), soit des modules synthétisables, dits processeur softcore (MicroBlaze, NIOS-II, etc.). Ces softcores sont configurables à travers plusieurs paramètres (profondeur du pipeline, taille du cache, présence ou non de multiplieurs, etc.). Un processeur softcore est réalisé à partir de la logique programmable présente sur la puce. Ainsi, un concepteur peut mettre autant de processeurs softcore que la puce le permet. A titre illustratif, un processeur MicroBlaze occupe seulement 4% de la surface d'un FPGA de la famille Virtex5-LX50 (composant à 28.800 cellules logiques), et 1,7% de la surface d'un FPGA de la famille Virtex5-LX110T (composant à 69.120 cellules logiques). A l'inverse, le concepteur ne choisit pas le nombre de processeurs hardcore présents dans un FPGA. L'intérêt majeur des FPGA est la possibilité de faire du prototypage rapide, la proximité des cœurs de processeurs avec de la logique programmable rend possible la spécialisation de ces processeurs pour concevoir un système sur puce.

#### 4.2.1. Les processeurs hardcore

Le cœur d'un processeur hard est décrit au niveau physique (layout sur silicium). Dans ce cas, l'optimisation est maximale en termes de vitesse et de surface utilisée (technologie fixe). Les performances du système sont parfaitement connues et ne varient pas d'un système à l'autre (surface, consommation, délais). Cependant, le cœur du processeur hard souffre d'une trop faible flexibilité ce qui le rend très spécifique à l'application pour laquelle il a été étudié. Par exemple, la taille des microcodes utilisés est fixe ce qui interdit l'ajout de nouvelles fonctionnalités.

### a. Le processeur PowerPC405 de Xilinx

Le PowerPC405 d'IBM [24] est un processeur RISC 32 bits avec une architecture Harvard. Son architecture pipeline est de 5 étages avec une fréquence de fonctionnement de 400MHz. Il dispose de 32 registres de 32 bits, une mémoire cache d'instructions et de données de 16 KB chacune et d'une unité de gestion de mémoire (Memory Management Unit). Sur les circuits FPGA Xilinx supportant le processeur PowerPC, l'accès aux mémoires internes (Blocs RAM du FPGA) se fait à travers une unité "On-Chip memory Controllers".

Le cœur de processeur matériel PowerPC est enfoui dans quelques catégories de circuits FPGA (Virtex5-FXT par exemple), c'est à l'utilisateur de rajouter les périphériques qui sont nécessaires à son fonctionnement. Le processeur PowerPC 405 ainsi que le MicroBlaze sont exploitables grâce à l'outil EDK de Xilinx [23].

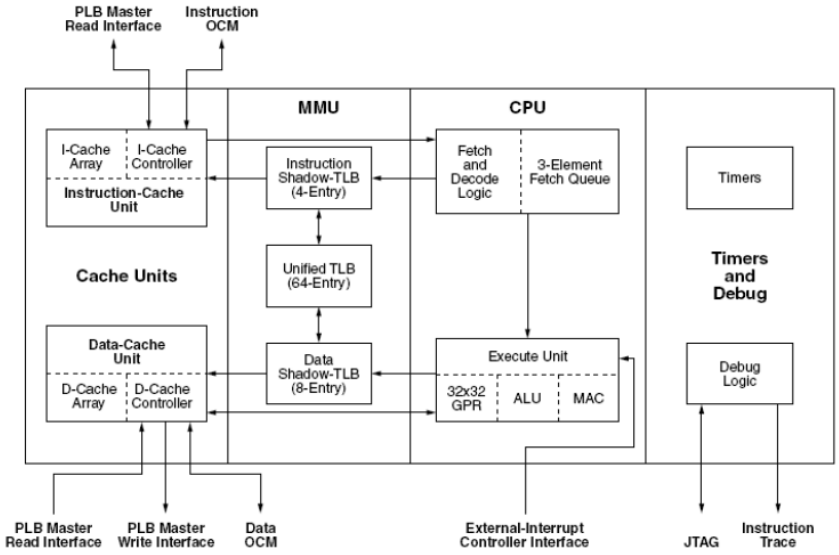



Figure 1.8. Architecture interne du PowerPC405

### b. Le processeur ARM

L'architecture ARM était originellement conçue pour un ordinateur de la société Acorn, puis elle a été complétée pour devenir une offre indépendante pour le marché de l'électronique embarquée. ARM est l'acronyme de Advanced Risc Machine (auparavant Acorn Risc Machine). Une particularité des processeurs ARM est leur mode de vente. En effet, ARM Ltd ne produit ni ne vend ses processeurs sous forme de circuits intégrés. La société vend les licences de ses processeurs de façon à ce qu'ils soient intégrés dans le silicium par d'autres fabricants. Actuellement, la majorité des grands fondeurs de puces proposent de l'architecture ARM. En plus, ARM Ltd a développé la norme AMBA qui est aujourd'hui un standard de bus informatique.

Les architectures ARM sont des RISC 32 bits. Dotées d'une architecture relativement plus simple que d'autres familles de processeurs, et bénéficiant d'une faible consommation, ces processeurs sont devenus dominants dans le domaine de l'informatique embarquée, en particuliers la téléphonie mobile et les tablettes. Après plusieurs versions (ARM1«1985»…ARM6«2001»), le cœur le plus célèbre est l'ARM7TDMI qui comporte 3 niveaux de pipeline. ARM Ltd a ensuite développé le cœur ARM9 qui comporte 5 niveaux de pipeline, MMU, double cache de 8 ou bien 16 Ko. Cela permet ainsi la

réduction du nombre d'opérations logiques sur chaque cycle d'horloge et par conséquent un progrès des performances en vitesse. Ensuite, plusieurs autres versions de ARM ont été proposées en doublant la mémoire cache, en ajoutant des ports AHB et des DSP, etc.

Dans le cadre de son Embedded Initiative, Altera unifie ses outils de conception pour FPGA dans un flux unique basé sur son logiciel Quartus-II. Il inclut un nouvel outil baptisé Qsys simplifiant l'interconnexion de blocs de propriété intellectuelle disparates. Cet outil unifié gère également l'intégration de cœurs de processeurs d'ARM, de MIPS et d'Altera lui-même couplés avec un FPGA dans un module multi-puces.

### 4.2.2. Les processeurs softcore

La capacité d'intégration des FPGA permet d'y implémenter des processeurs IPs soft. Ce sont des processeurs décrits à l'aide d'un langage de description matériel (VHDL ou bien Verilog) pouvant être synthétisable. Ces processeurs reprennent exactement le fonctionnement et les performances des autres processeurs implémentés en hard, l'avantage ici réside dans leur capacité d'être paramétrée. En effet, un processeur softcore s'adapte aux besoins de ses développeurs ainsi qu'aux contraintes matérielles (périphériques, performances, ressources, consommation, etc.). Le fait de paramétrer un processeur permet d'avoir une flexibilité et une facilité à faire un compromis performance-coût accélérant ainsi le cycle de conception. Cependant, les performances d'un softcore sont, généralement, inférieures à celles d'un processeur hardcore. Parmi les produits existant en ce moment sur le marché on peut citer :

#### a. Le processeur MicroBlaze de Xilinx

Spécialement pour la conception des SoC, Xilinx a proposé le MicroBlaze, un processeur RISC "soft IP" à 3 étages avec une architecture Harvard et avec 32 registres internes de 32 bits (Figure 1.9). Il dispose d'un bus d'instructions et de données internes et externes ((ILMB, DLMB, IOPB et DOPB). Le Processeur MicroBlaze, tout comme le NIOS, est très facilement configurable occupant selon le choix des options de 900 à 2600 éléments logiques et pouvant fonctionner sur une fourchette de fréquences à partir de 80MHz (exemple : en utilisant une Virtex5-LX50, avec un pipeline à 5 niveaux et sans MMU, le MicroBlaze occupant 1027 LUTs et fonctionne à 235 MHz).

Le processeur comporte 70 options de configuration permettant à l'utilisateur de sélectionner ou de paramétrer les composants internes selon ses besoins. Parmi les options configurables, on peut citer :

- Un pipeline à 3 ou 5 niveaux,
- Utilisation des multiplieurs câblés du FPGA,
- Opérateur de division,
- Opérateur de décalage (Barrel Shifter),
- FPU (Floating Point Unit),
- Mémoires cache instructions et données,
- Logique de débug (hardware breakpoints).

Le bus utilisé avec le MicroBlaze est le bus OPB (On-chip Peripheral Bus). C'est un bus IBM Core-Connect utilisé aussi avec les processeurs PowerPC. Ce bus autorise un maximum de 16 maîtres et un nombre d'esclaves illimité. Il dispose d'une politique d'arbitrage (Bus multi-maîtres) paramétrable. Dans un système sur puce programmable à base de processeur MicroBlaze, le bus OPB est utilisé afin

de connecter les périphériques dont les besoins en communication sont faibles. Autrement, Xilinx fournit le lien FSL (Fast Simplex Link) permettant des accès rapides (2 fronts d'horloge) des périphériques vers le MicroBlaze et vice versa (8 connexions FSL par MicroBlaze). Un autre type de bus LMB (Local Memory Bus) est utilisé pour accéder aux blocs RAM du FPGA. Dans le MicroBlaze, ce bus est utilisé pour les instructions et les données et il assure des accès rapides à la mémoire (1 front d'horloge) [23].

En plus, de nombreux périphériques sont fournis avec le MicroBlaze, afin de constituer un microcontrôleur complet et personnalisable. Il y a, entre autres : contrôleur mémoire (SRAM, Flash), contrôleur mémoire SDRAM, UART lite, Timer/compteur avec fonction PWM, interface SPI, contrôleur d'interruptions, GPIO (entrées-sorties génériques), convertisseurs A/N et N/A Delta-Sigma, DMA, etc. [26].

MicroBlaze peut fonctionner en utilisant plusieurs systèmes d'exploitation (Xilinx MicroKernel, uClinux, FreeRTOS, etc.). Ces systèmes d'exploitation sont constitués généralement d'un ensemble de bibliothèques permettant d'obtenir des fonctions basiques telles que : pilotes de périphériques, séquencement de tâches, système de fichiers FAT, pile TCP/IP (avec le logiciel libre lwip), etc.

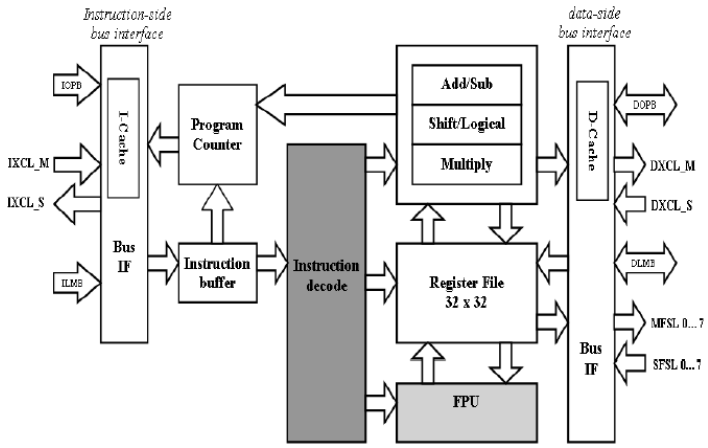



Figure 1.9. Architecture interne du MicroBlaze 4.0 [27].

### b. Les processeurs NIOS et NIOS-II de Altera

Le NIOS est un Processeur softcore propriétaire de la société Altera. Il est basé sur un cœur RISC 32 bits et doté du bus Avalon. Le NIOS-II est la deuxième génération de processeurs embarqués à cœur logiciel d'Altera, il annonce déjà une nouvelle aire pour l'électronique et l'informatique embarqué. NIOS-II incorpore de nombreuses améliorations sur l'architecture d'origine NIOS, le rendant plus adapté pour une large gamme d'applications d'électronique embarqué et systèmes de contrôle [28]. Contrairement au MicroBlaze, Synopsys DesignWare offre une licence de NIOS-II pour les standard-cell ASICs, et grâce à cette licence, les concepteurs peuvent utiliser NIOS pour une production de masse d'ASIC. Pour bien gérer les applications à base de processeur NIOS, Altera a développé trois principaux outils de développement :

- L'outil Quartus pour l'implémentation du NIOS dans un FPGA ;
- Le développement du cœur et de ses composants ("Briques IP") s'effectue à l'aide du "SOPC Builder" ;

- Le développement du logiciel s'effectue en C sous NIOS-IDE (Windows) ou en utilisant nios2-toolchain sous Linux.

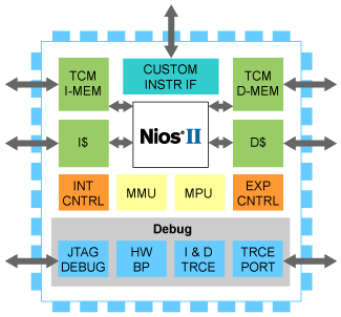



Figure 1.10. Les processeurs NIOS et NIOS-II de Altera

La nature embarquée de NIOS-II permet au concepteur de spécifier et générer un processeur sur mesure pour des besoins d'applications spécifiques. Les concepteurs peuvent étendre les fonctionnalités de base en y ajoutant une unité de gestion de mémoire préétablie pour définir des instructions personnalisées ou des périphériques personnalisés.

### 4.2.3. Les processeurs softcore et Open Source

L'idée fondamentale, des processeurs open source, est de distribuer des bibliothèques ou bien des codes sources d'application gratuitement, afin de permettre aux utilisateurs de l'employer, modifier, corriger et améliorer, comme pour Linux, GCC, Apache ou encore gdbm et mysql.

#### a. Le processeur LEON

Le processeur Leon, développé par l'Agence Spatiale Européenne (ESA), emploie une architecture interne de 32 bits, entièrement décrite en VHDL et complètement synthétisable. Son architecture est de type SPARC-V8 (Scalable Processor ARChitecture). Tout comme l'architecture RISC (Reduced Instruction Set Computer), l'architecture SPARC ne spécifie pas une implémentation matérielle précise mais un standard libre de droits. L'architecture SPARC est directement dérivée de l'architecture RISC. Elle a été conçue pour s'adapter à différents types de systèmes en termes de performances, prix et flexibilités. D'autre part, dans la norme SPARC, il existe deux versions d'architectures : une version SPARC-V8 pour une implémentation 32 bits et une autre version SPARC-V9 en 64 bits. Le principal avantage de l'architecture SPARC est l'implémentation des registres du processeur en « Register Windows ». Cette architecture permet des économies en nombres d'instructions de type load/store en mémoire et offre au compilateur la possibilité de produire un code plus efficace. En effet, comparé à d'autres types d'architectures RISC, le compilateur possède une plus grande flexibilité d'affectation des registres aux variables de l'application. Pour des programmes complexes décrits avec des langages à haut niveau d'abstraction (type C++), la réduction en termes de nombres d'instructions exécutées est significative [9].

Le modèle VHDL, de LEON, implémente une unité centrale de 32-bit conforme au modèle d'architecture IEEE-1754 (SPARC-V8). Il est conçu pour des applications embarquées avec les caractéristiques suivantes : les caches de données et d'instructions séparés, multiplieur et diviseur matériel, contrôleur d'interruption, unité de déboguage avec tampon de trace, deux timers de 24 bits, deux UARTs, fonction power-down, chien de garde, port d'E/S de 16 bits, contrôleur de mémoire

flexible, ethernet MAC et interface PCI. De nouveaux modules peuvent facilement être ajoutés en utilisant les bus AMBA, AHB/APB. Le modèle VHDL est complètement synthétisable avec la plupart des outils de synthèse et peut être implémenté sur FPGAs ou ASICs. La simulation peut être faite avec tout VHDL-87 simulateurs.

Le modèle de LEON VHDL est fourni avec deux licences : GNU Public License (GPL) et Lesser GNU Public License (LGPL). Le LGPL concerne le modèle de LEON lui-même tandis que le reste des fichiers et des testbenchs est fourni sous GPL. LEON2 produit les meilleurs résultats aux deux tests le Dhrystone 2.1 benchmark et le Stanford benchmark pour trois configurations différentes, il donne les meilleures performances par cycle d'horloge pour tous les tests et toutes les configurations.

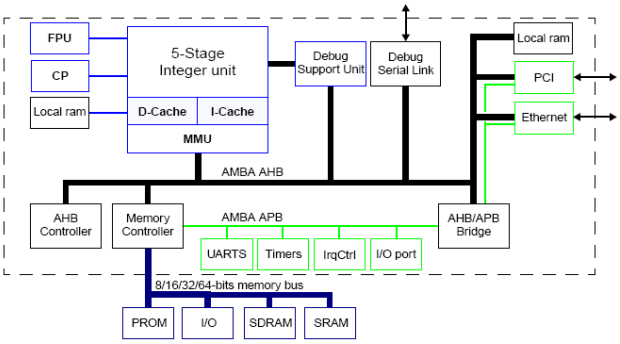



Figure 1.11. Architecture interne du processeur leon2.

**b. Le processeur OR1200**

OR1200 (OpenRisc) est un synthétisable RISC scalaire de 32 bits avec une microarchitecture Harvard, 5 niveaux de pipeline. Il fonctionne à 33 Mégahertz sur un FPGA Virtex2. Il est open source, d'un modèle écrit en Verilog, son code source est disponible sous licence GNU GPL. Il a été conçu pour des applications embarquées et pour des réseaux industriels. Sa conception s'appuie sur la performance, la souplesse et l'économie d'énergie. Il existe des SDK complets (Kit de développement du logiciel) disponibles avec des utilités binaires, compilateur C/C++, débogueur et simulateur architectural. L'équipe Opencore a également développé une plateforme appelé ORP (plateforme de référence d'OpenRisc), le but est d'avoir une plateforme commune qui permet aux développeurs des SOCs de créer un nouveau OpenRisc en un temps minimal de réalisation et de vérification [12].

Le processeur OR1200 est constitué d'un cœur CPU/DSP, d'un bloc de gestion de mémoire MMU (Memory Management Unit), d'une mémoire cache configurable (ICache, Dcache), d'une unité de débogage (Debug Unit), d'un contrôleur d'interruption programmable (PIC : Programmable Interrupt Controller), d'un bloc de gestion de puissance (Power management), et d'un Tick Timer qui mesure le temps nécessaire pour exécuter des programmes. Il peut être connecté à une mémoire à travers une interface Wishbone. Toute la documentation pour programmer ce processeur est disponible sur le site d'OpenCores.

Le processeur OR1200 possède un pipeline d'exécution de profondeur 5 étages. Il est doté de 53 instructions de taille fixe de 32 bits. Ce processeur présente plusieurs modules internes (gestion du compteur de programme, recherche d'instruction, module de contrôle et décodage d'instructions, registres, etc.). Le processeur peut communiquer avec d'autres composants en échangeant des données à travers un système de bus Wishbone.

# Chapitre 2

# La Compression Vidéo et la Nouvelle Norme de Compression H.264/AVC

*« Recommandation : Un standard ne définit pas un encodeur, mais le résultat qu'un encodeur doit produire » (UIT-T).*

## 1. Introduction

Ce chapitre est réservé à l'étude de la norme de compression vidéo H.264 connue aussi sous le nom MPEG4-part10. Nous définissons dans un premier temps le codage vidéo ainsi que l'évolution des normes de compression. Dans un deuxième temps, nous décrivons les spécificités de la nouvelle norme de compression H.264/AVC. Dans cette partie, nous donnons l'importance aux trois modules de traitement (le module de codage Intra, le module de codage Inter et le module de filtre anti-blocs). Ces trois modules sont jugés les plus lourds du point de vue temps de calculs, et auront l'objet d'une implémentation matérielle par la suite dans le chapitre 5. Les modules de transformation et de quantifications directes et inverses sont aussi jugé importants et peuvent être réalisé en matériel. Pour cela, nous avons détaillé au maximum l'encodeur H.264/AVC, ainsi que l'organisation des données à l'entrée et à la sortie de chaque module.

Les performances et les nouveautés de la norme H.264 sont données à la fin de ce chapitre, avec des exemples d'applications et des implémentations logicielles et matérielles de l'encodeur et du décodeur. Nous donnons, enfin, notre conclusion qui doit être influé sur nos décisions pour l'implémentation matérielle et/ou logicielle des différents modules de l'encodeur dans les chapitres de la deuxième partie.

## 2. Le codage vidéo

### 2.1. Définitions

Une image en physique est une représentation d'un objet, produite par la réunion des rayons ou des faisceaux lumineux qui en émanent et se reconstituent sur un miroir, sur un écran ou sur l'œil qui perçoit cette image. Une image numérique est une image échantillonnée selon les deux axes spatiaux. Donc, une image numérique est composée d'une grille de points élémentaires appelés pixels [4]. Le nombre de pixels par image, le nombre de bits pour le codage de chaque pixel et la nature de ces pixels constituent les principales caractéristiques d'une image numérique.

Une vidéo numérique est une séquence d'images numériques [29]. L'œil humain a comme caractéristique d'être capable de distinguer environ 20 images par seconde. Ainsi, en affichant plus de 20 images par seconde, il est possible de tromper l'œil et de lui faire croire à une image animée [30]. En plus des caractéristiques d'une image numérique fixe, une vidéo numérique est caractérisée par le nombre d'images par seconde (FPS = Frame Per Second). Le codage vidéo consiste à l'opération d'identification et de représentation d'une vidéo à l'aide d'un code informatique.

### 2.1.1. L'espace colorimétrique des pixels

Pour représenter une couleur, une première approche consiste à estimer l'ensemble des composantes spectrales, en filtrant le signal de couleur par de nombreux filtres à bande étroite. L'intensité de chaque sortie filtrée donnerait une estimation des longueurs d'onde présentes dans le spectre. Seulement cela nécessite d'utiliser un trop grand nombre de capteurs. Des expériences psychovisuelles d'égalisation ont montré qu'en combinant trois stimuli de longueurs d'onde particulières, il est possible de synthétiser presque toutes les couleurs existantes. L'espace de couleurs RGB est fondé sur les trois couleurs monochromatiques suivantes : Rouge (700 nm), Vert (546 nm) et Bleu B (435.8 nm). Le mélange de ces 3 couleurs permet d'obtenir toutes les autres [8].

### 2.1.2. Le format YUV

En plus du système RGB, il existe d'autres systèmes de couleurs tels les systèmes YUV ou YCbCr. Ces systèmes exploitent le fait que le cerveau traduit le signal trichromatique perçu par l'œil, comme un signal composé de trois composantes, une luminance Y et deux chrominances Cb et Cr [31]. La compression d'images et de vidéo exploite ce principe. Cette représentation est obtenue par la transformation suivante :

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0{,}229 & 0{,}587 & 0{,}114 \\ 0{,}596 & -0{,}274 & -0{,}322 \\ 0{,}212 & -0{,}523 & 0{,}110 \end{bmatrix} x \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad 2.1$$

On préférera donc un espace de luminance et chrominances (YUV) où les primaires sont décorrélées, ce qui offre l'avantage de séparer les informations d'intensité lumineuse et de couleur.

### 2.1.3. Format d'image (première idée de compression)

Comme l'œil humain est moins sensible à la couleur qu'à l'intensité lumineuse (question de proportion de bâtonnets et cônes dans la rétine [32]), on sous-échantillonne la couleur, en prenant la moitié des informations (format 4:2:2) ou bien le quart des informations (format 4:2:0) [29] :

- Format (4:4:4) : Chaque composante est codée de la même manière, il n'y a pas de sous-échantillonnage des chrominances.
- Format (4:2:2) : Les deux chrominances ont le même nombre de lignes que la luminance, mais deux fois moins de colonnes.
- Format (4:2:0) : Les chrominances ont deux fois moins de lignes et deux fois moins de colonnes que la luminance.

Chaque composante du signal vidéo peut être codée sur 8 à 10 bits. 8 bits sont suffisants pour représenter une composante puisque l'œil ne distingue pas deux niveaux voisins. D'une manière générale, les composantes du signal vidéo sont le plus souvent codées sur 8 bits avec un format 4:2:0. On a donc en moyenne 1,5 octet par pixel [2].

## 2.2. Problématique

Le Tableau 2.1 récapitule les paramètres des formats standards des vidéos numériques [33]. Le débit nécessaire à la transmission des données non compressées est calculé pour un format en 4:2:0 avec 8 bits par composante. Ces formats représentent d'énormes quantités de données alors que le débit maximum, en Mbps (Mégabits par seconde), proposé par les fournisseurs d'accès est estimé à [32] :

- Réseaux large bande : 10 Mbps ;
- Satellite / internet : 1-2 Mbps ;
- Connexion ADSL : 30 Mbps ;
- Vidéo conférences : 128-1000 kbps ;
- PDA/ tel Mobile 3G : 160-320 kbps.

Les débits sont ainsi beaucoup trop élevés par rapport aux bandes passantes disponibles pour leur diffusion. Il est donc nécessaire de compresser les données vidéo pour permettre leur diffusion avec des débits réalistes. Il faut aussi définir des normes de codage vidéo pour que ces données compressées puissent être décompressées par tout le monde en vue d'une visualisation.

| **Définition** | **QCIF** | **CIF** | **SD (NTSC)** | **SD (PAL, SECAM)** | **SD (DVD)** | **HD (720p)** | **HD (1080i)** | **HD (1080p)** |
|---|---|---|---|---|---|---|---|---|
| **Balayage** | 176×144 progressif | 352×288 progressif | 640×480 entrelacé | 768×576 entrelacé | 720×576 entrelacé | 1280×720 progressif | 1920×1080 entrelacé | 1920×1080 progressif |
| **Fréquence** | 10 à 30 Hz | 10 à 30 Hz | 60 Hz | 50 Hz | 50 à 60 Hz | 24 à 60 Hz | 50 et 60 Hz | 24 à 30 Hz |
| **Débit** | 380 Ko à 1.1 Mo/s | 1.5 à 4.6 Mo/s | 13.8 Mo/s | 16.6 Mo/s | 15.6 à 18.6 Mo/s | 33.2 à 83 Mo/s | 78 à 93 Mo/s | 75 à 93 Mo/s |

Tableau 2.1 : Formats standards des vidéos numériques.

## 2.3. La compression vidéo

La compression vidéo (codage vidéo) a pour but de réduire et de supprimer la redondance spatiale et temporelle des données vidéo de façon à permettre leurs transmissions et leurs stockages sur des bandes fréquentielles étroites et des supports de capacités limités. Le processus consiste à traiter la source vidéo à l'aide d'un algorithme (voir plusieurs algorithmes) afin de créer un fichier compressé prêt pour la transmission et/ou le stockage. Pour lire le fichier compressé, un algorithme inverse est appliqué, ce qui permet d'obtenir une vidéo contenant pratiquement le même contenu que la source vidéo d'origine (Figure 2.1). Une paire d'algorithmes travaillant ensemble est appelée un codec vidéo (codeur/décodeur).

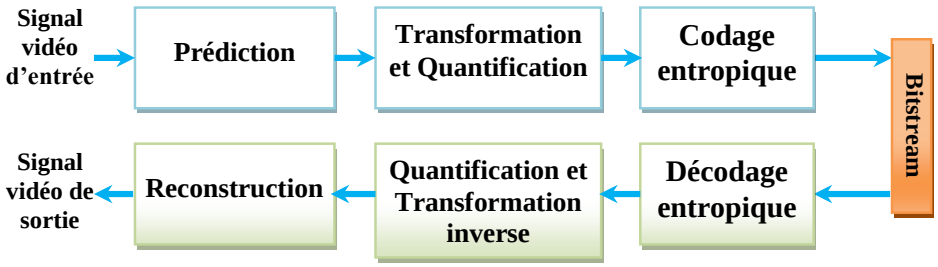



Figure 2.1 : Chaine directe et inverse d'un codec vidéo.

### 2.3.1. Processus de codage vidéo

Un codeur vidéo consiste à mettre en œuvre séquentiellement des traitements de base (prédiction, transformation et codage) pour produire un flux binaire compressé. La condition sur les traitements de base est qu'ils soient réversibles et produisent une quantité de données compressées et inférieurs aux flux binaire initiale.

**Prédiction :** Une image d'une vidéo est subdivisée en macroblocs (N×N pixels). Le codeur calcule une prédiction du macrobloc courant d'après les données déjà codées provenant soit de la même image

(prédiction Intra) ou bien à partir d'autres images (prédiction Inter). Le codeur soustrait ensuite la prédiction ainsi formée du macrobloc courant pour former un résidu (compensation) [34].

**Transformation et quantification :** C'est une manipulation mathématique (généralement fréquentielle) permet de concentrer l'information dans un nombre réduit de coefficients [29]. La majorité des méthodes de compression utilise des transformations sur les blocs prédits ou bien sur les blocs résidus. La quantification est l'étape de compression non-conservative qui diminue la quantité d'information contenue dans les coefficients transformés. Elle consiste à diviser chacun des coefficients d'un bloc par un pas de quantification QP et de n'en conserver que la partie entière. Les coefficients DCT quantifiés sont ensuite réorganisés et réordonnancés dans un vecteur unidimensionnel en les balayant en zigzag pour augmenter le nombre des zéro successifs, et par conséquent l'augmentation de l'efficacité de l'étape suivante (codage entropique).

**Codage binaire :** Le codage binaire (codage statistique) consiste à exploiter les propriétés statistiques des coefficients quantifiés en utilisant des mots courts pour représenter les événements les plus probables et des mots plus longs pour les occurrences rares.

### 2.3.2. Processus de décodage

Un décodeur vidéo consiste à effectuer les opérations inverses du codeur (décodage, transformation inverse et reconstruction) pour produire une séquence vidéo décodée la plus proche possible de la séquence initiale [34].

**Décodage du flux binaire :** Le décodeur vidéo reçoit le flux binaire compressé. La première étape consiste à décoder chacun des éléments de syntaxe et extrait les coefficients de transformée quantifiés et les paramètres de compression. Ces informations sont ensuite utilisées dans les deux étapes suivantes pour inverser le processus de codage et recréer la séquence d'images initiale.

**Quantification et transformation inverses :** Premièrement, les coefficients sont remis à l'échelle, ceci en les multipliant par le bon pas de quantification. Comme la quantification est une compression non conservative, généralement on n'obtient pas les mêmes valeurs que les coefficients de départ. Ensuite, la transformée inverse est utilisée pour reconstruire les macroblocs ou bien les résidus.

**Reconstruction :** Pour chaque macrobloc, le décodeur reforme une prédiction identique à celle effectuée par le codeur. Il lui ajoute ensuite les résidus précédemment calculés pour reconstruire le macrobloc, qui pourra enfin être affiché comme une partie d'une image de la séquence vidéo.

### 2.3.3. Evaluation de la qualité des images décodées

La compression vidéo avec pertes exploite les failles du système visuel humain pour réduire la quantité des informations à transmettre. La qualité des images est donc réduite de façon plus ou moins visible. L'évaluation de la qualité d'une image ou d'une vidéo est très complexe. Il existe des métriques de qualité objective, basées sur un calcul de distorsions comme le PSNR (Peak Signal to Noise Ratio) très utilisé dans le traitement du signal en général. Le PSNR mesure une erreur quadratique moyenne non pondérée sur toute l'image.

$$PSNR = 10 \times \log_{10}\left(\frac{MaxAmplitude^2}{MSE}\right) = 10 \times \log_{10}\left(\frac{\left(2^n - 1\right)^2}{\frac{1}{N} \times \sum_0^{N-1} \left(originale - reconstruite\right)^2}\right) \qquad 2.2$$

### 2.3.4. Types des codeurs vidéo

Le schéma de codage vidéo de type hybride est le plus courant puisqu'il est utilisé dans toutes les normes de codage vidéo (MPEG-x, H.26x). Il est appelé hybride parce que c'est un schéma en boucle fermée qui utilise les informations déjà codées/décodées pour réaliser le codage de la partie courante à l'aide de différents modules. Le schéma hybride est toujours de la même forme, seuls les modules qui le composent (prédiction, transformation et codage binaire) diffèrent d'un codeur à un autre [29]. Autres que le schéma hybride, il existe d'autres types de schémas de codage vidéo basés sur des approches en ondelettes, à savoir :

1. Les codeurs basés ondelettes (en boucle ouverte) utilisent généralement un filtrage temporel compensé en mouvement couplé à une transformée en ondelettes 2D. On les appelle schémas "t+2D" puisqu'ils réalisent d'abord une décomposition temporelle avant d'effectuer la décomposition 2D spatiale classique.
2. Les codeurs par analyse-synthèse qui utilisent souvent des transformations en ondelettes. Dans ce type de schéma, l'étape d'analyse correspond à estimer le mouvement des objets de la séquence vidéo à l'aide de maillages 2D déformables et à décorréler les informations de mouvement et de texture, les textures sont redressées selon le mouvement estimé.

Ces schémas ondelettes sont tout aussi performants et intéressants que les schémas de type hybride, mais ils n'ont jamais été retenus pour des solutions normalisées.

## 2.4. L’estimation de mouvement et la compression vidéo

L'estimation de mouvement est une opération clé pour la compression vidéo. Cette opération contribue pour une grande partie à l'efficacité de la compression en éliminant les redondances temporelles. C'est aussi l'opération nécessitant le plus de puissance de calcul : jusqu'à 60% de la charge de calcul d'un encodeur vidéo H.264 par exemple [2]. Un effort particulier doit être alors mis sur l'estimation de mouvement, tant sur le plan algorithmique que sur l'implantation (logicielle ou bien matérielle).

### 2.4.1. Estimation de mouvement à un nombre entier de pixels

L'estimation de mouvement est largement utilisée pour l'élimination de la redondance temporelle entre les images d'une séquence vidéo. Parmi plusieurs méthodes, l’estimation de mouvement par correspondance de blocs (Block Matching) est adoptée par la plupart des standards de codage vidéo [35]. Cette méthode reste la plus naturelle, elle est basée sur la corrélation spatiale entre deux images. Les valeurs des pixels sont directement exploitées afin de mettre en correspondance deux régions dans deux images successives. La position relative des deux régions donne directement le mouvement.

La méthode Block Matching suppose la subdivision de l’image prédite en macroblocs non chevauchés et de tailles identiques, ensuite elle cherche pour chacun de ces macroblocs le plus ressemblant dans l’image de référence (ou bien dans les images de référence). L’algorithme Block Matching commence la recherche à partir d’un point de départ (généralement les mêmes coordonnées du bloc-cible) et choisit un ensemble de points candidats autour de ce point selon une distance D appelée "pas de recherche" (Figure 2.2) [36]. Avec D égale à (N+2p) et (p) représente le déplacement maximum du macrobloc de taille (N×N).

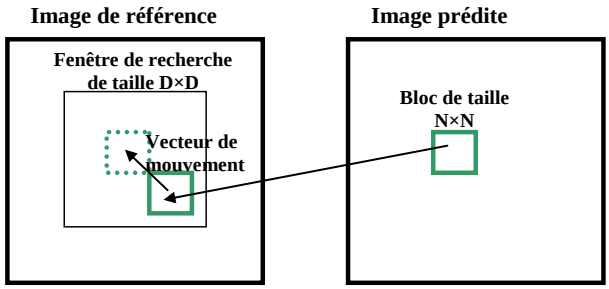



Figure 2.2 : Principe d'estimation de mouvement.

La recherche dans la fenêtre D×D peut être exhaustive (Full Search Algorithm : FSBM), où le macrobloc de l'image P est comparé avec tous les macroblocs possibles dans la fenêtre de recherche (avec un déplacement d'un pixel). La recherche peut être aussi concentrée sur des endroits bien définis, cela est exprimé par plusieurs dérivés de l'algorithme FSBM, on trouve : l'algorithme TSS (Three Step Search), l'algorithme 2-D LOGS (2-D Logarithmic Search), l'algorithme BBGDS (Block-based Gradient Decent Search), l'algorithme FSS (Four Step Search), l'algorithme HEXBS (Hexagon-based Search), l'algorithme DS (Diamond Search), et l'algorithme EPZS (Enhanced Predictive Zonal Search). De nombreuses variantes de ces algorithmes existent dans la littérature [4]. Ces algorithmes donnent de bons résultats pour des fenêtres de recherche et des images relativement petites. Dans certaines séquences avec des mouvements plus élevés et irréguliers, ces algorithmes sont susceptibles de tomber dans un local minimum. Les résultats expérimentaux montrent que certains de ces algorithmes causeront des dégradations d'images de 1 à 2 dB [37].

### 2.4.2. Compression vidéo par estimation et compensation de mouvement

Dans un encodeur vidéo, le module d'estimation et de compensation de mouvement est toujours le bloc le plus lourd du point de vue quantité et complexité de calculs. L'image prédite P est découpée en macroblocs (de taille N×N pixels) sur lesquels est appliqué la procédure suivante :

- Chercher le macrobloc, dans l'image de référence, le plus semblable au macrobloc de l'image prédite en utilisant une des méthodes d'estimation de mouvement.
- Le macrobloc retenu devient alors le prédicteur du macrobloc courant et lui est soustrait pour former un macrobloc résiduel (Compensation de mouvement).
- Le bloc résiduel est codé par transformation et transmis avec les deux vecteurs de mouvement vers l'étape suivante de codage entropique.

Le but de la compensation du mouvement est de fournir une information supplémentaire au décodeur afin d'alléger l'énergie résiduelle nécessaire à la prédiction de la trame future. Donc dans un encodeur, l'estimateur va associer des vecteurs de déplacement à la trame, quant au compensateur, il calculera le résidu. Le décodeur utilise le vecteur de mouvement pour recréer le bloc prédit et décode le bloc résiduel, l'ajoute au prédicteur et reconstruit le bloc original [38].

### 2.4.3. Critères de mesure de distorsions

La recherche par correspondance de blocs est basée sur la comparaison des macroblocs. Cette méthode d'estimation de mouvement vise à minimiser un certain critère de distorsions. Le choix de l'opérateur de distorsions n'est pas imposé par les normes de codage vidéo des deux comités de standardisation (ISO/IEC, UIT-T) [39]. Par contre, il existe plusieurs critères permettant d'évaluer l'amplitude de

l'erreur de prédiction. Les plus courants, appliqués sur des macroblocs 16×16 pixels, sont présentés ci-après [36].

- La Somme des Différences Absolues (SAD) :

$$\mathrm{SAD}(k,l) = \sum_{r=0}^{15}\sum_{c=0}^{15} \mathrm{abs}\left[I(i+r, j+c) - P(k+r, l+c)\right] \qquad 2.3$$

P , I : La luminance du macrobloc prédite et du macrobloc de référence ;

(i, j) : Les coordonnées du macrobloc recherché dans l'image I ;

(k, l) : Les coordonnées du macrobloc courant dans l'image P.

- La Somme des Erreurs Quadratiques (SSE) :

$$\mathrm{SSE}(k,l) = \sum_{r=0}^{15}\sum_{c=0}^{15} \left[I(i+r, j+c) - P(k+r, l+c)\right]^2 \qquad 2.4$$

- La Somme des Différences Absolues Transformées (SADT) :

$$\mathrm{SADT}(k,l) = \sum \mathrm{abs}\left(H\left[I(i:i+15, j:j+15) - P(k:k+15, l:l+15)\right]\right) \qquad 2.5$$

Où H signifie la transformée de Hadamard appliquée sur le macrobloc résiduel.

Le SAD (Sum of Absolute Difference) est le critère le plus utilisé dans les normes de codage vidéo à cause de sa complexité réduite (nécessite seulement des soustractions et des additions). Le SSE (Sum of Squared Errors), qui représente l'énergie résiduelle, est moins utilisé que le précédent à cause de sa complexité (nécessite des multiplications) et de sa sensibilité au bruit (une grande différence due au bruit sur un pixel de l'image a un grand impact sur la SSE). Le SATD (Sum of Absolute Difference Transform) est largement répandue dans un contexte d'encodage vidéo avec compensation de mouvement : elle représente le coût de codage du résidu grâce à une transformée rapide. Elle nécessite néanmoins l'implantation de la transformée de Hadamard, qui peut s'avérer coûteuse pour une estimation de mouvement temps réel.

## 2.5. Les normes de compression vidéo

Au cours des dernières années, l'intérêt pour le multimédia a entraîné l'investissement d'une somme importante d'efforts de recherche sur le codage vidéo dans les universités et l'industrie, efforts qui se sont traduits par l'élaboration de plusieurs normes [4]. Ces normes visent un vaste éventail d'applications aux exigences variées quant au débit binaire, à la qualité d'image, à la complexité et au délai ainsi qu'à l'amélioration des rapports de compression.

Les standards de compression vidéo ont pour but d'assurer la compatibilité entre encodeur et décodeur, tout en laissant le champ libre aux fabricants pour développer des produits innovants et compétitifs. D'où la définition qu'*Un standard ne définit pas un encodeur, mais le résultat qu'un encodeur doit produire*. Autrement dit, il porte sur la syntaxe du flux et la manière de le décoder [2].

### 2.5.1. Les normes de l'UIT-T

l'UIT-T (1993) connue aussi sous le nom CCITT (Comité Consultatif International Téléphonique et Télégraphique, jusqu'en 1993) est une partie de la plus ancienne organisation internationale technique de coordination UIT (1865). Les standards internationaux produits par l'UIT-T sont appelés des « Recommandations ». Chaque catégorie de normes UIT-T correspond une lettre de l'alphabet, la

référence de la norme étant complétée d'un nombre. Le groupe VCEG (Video Coding Experts Groups) de l'UIT-T est spécialisé beaucoup plus dans des applications de type vidéoconférence faible latence avec de très faibles débits, les standards développés par ce groupe correspond une lettre H :

- Les protocoles de communications multimédias (H.223, H.225, H.245).
- Les normes de compression vidéo : H.261 [40] (Codec vidéo pour services audiovisuels à px64 kbit/s, 1991), H.263 [41] (Codage vidéo pour communications à faible débit, 1996).

### 2.5.2. Les normes de l'ISO/MPEG

L'organisation ISO, créée en 1947, a pour but de produire des normes internationales dans les domaines industriels et commerciaux appelées normes ISO. Le groupe MPEG (Moving Picture Experts Group) de l'ISO travaille spécialement sur le codage générique des images et du son associé et leurs transports à travers des réseaux pour la télévision numérique. Les aspects systèmes (synchronisation, transport, stockage) sont définis dans la norme ISO/CEI 13818-1. Les aspects compression sont définis dans les normes ISO/CEI13818-2 et ISO/CEI13818-3. MPEG a développé les normes suivantes :

- MPEG-1 (1988-1992) [42] : première norme audio et vidéo utilisé aussi pour les Vidéo CD. Ce format offre une résolution à l'écran de 352×240 pixels à 30 images par seconde ou de 352×288 à 25 images par seconde avec un débit d'environ 1,5 Mbit/s. MPEG-1 comprend le populaire format audio MPEG-1/partie3 (MP3).
- MPEG-2 [43](1994): norme applicable au codage de l'audio et la vidéo, ainsi que leur transport pour la télévision numérique : télévision numérique par satellite, télévision numérique par câble, télévision numérique terrestre, et pour les vidéodisques DVD ou SVCD. C'est notamment le format utilisé jusqu'à présent pour la TV sur ADSL. Les débits habituels sont de 2 à 6 Mbps pour la résolution standard (SD), et de 15 à 20 Mbps pour la haute résolution (HD).
- MPEG-4 [44] (1993-1999): norme applicable aux bas débits jusqu'à 2 Mbit/s, exclus de la matrice des décodeurs de MPEG-2. Permet, entre autres, de coder des objets vidéo/audio, le contenu 3D et inclut le DRM. La partie 2 de MPEG-4 (Visual – finalisée en 1999) est compatible avec la partie baseline de H.263 et a connu du succès grâce à la mise en application DivX ainsi que dans les téléphones mobiles. La partie 10 appelée MPEG-4/AVC permet des gains d'un facteur 2 à 3 par rapport à MPEG-2 et a déjà été retenue comme le successeur de celui-ci pour la TV-HD, la TV sur ADSL et la TNT. L'extension de cette partie, appelée Scalable Video Coding (SVC) permet de proposer différents niveaux de qualité à partir d'un même flux codé.
- MPEG-7 : Norme de description pour la recherche du contenu multimédia.
- MPEG-21 : Norme proposant une architecture pour l'interopérabilité et l'utilisation simple de tous les contenus multimédia.
- MPEG-A : tourné vers les applications multimédia. En cours de standardisation.

### 2.5.3. H.264/AVC ou bien MPEG4/AVC part 10

En 1998, l'UIT-T lança le projet H.26L dans le but de créer une nouvelle architecture de codec ayant pour but un gain en efficacité de codage d'un rapport au moins égal à 2 par rapport aux standards existants. Un autre but était de créer une interface simple pour pouvoir adapter le codec aux différents protocoles de transport (commutation de paquets et de circuits). Le codec a été développé en s'assurant

qu'il serait transposable sur plateformes à un coût raisonnable en tenant compte des progrès réalisés par l'industrie des semi-conducteurs. En 2001, le projet H.26L avait atteint ses objectifs en taux de compression [45].

En décembre 2001, et après s'être réunis pour la définition en commun du standard MPEG-2/H.262, les deux groupes, VCEG de l'UIT-T et MPEG de l'ISO/IEC, ont à nouveau mis leurs travaux en commun au sein du JVT (Joint Video Team) pour aboutir un nouveau standard, adopté de manière identique des deux côtés [32]. Le H.264/AVC est le nom employé par l'UIT-T, l'ISO/IEC préférant pour sa part opter pour l'appellation MPEG-4 Partie10/AVC, la norme étant présentée comme un nouveau élément de la série de normes MPEG-4. Cette nouvelle norme est approuvée en mai 2003 par l'UIT-T et en octobre 2003 par l'ISO [46]. La version finale de H.264/AVC est approuvée en mars 2005 [47]. Après cette version, plusieurs autres améliorations ont été approuvées (2009).

### 2.5.4. Evolution des normes de compression vidéo et leurs applications

La figure 2.3 montre l'évolution des différents standards et normes de codage des séquences vidéo des deux comités UIT-T et ISO/IEC. Cette figure représente aussi une classification des normes de codage vidéo. Le tableau 2.2 récapitule ces normes et leurs applications [48].

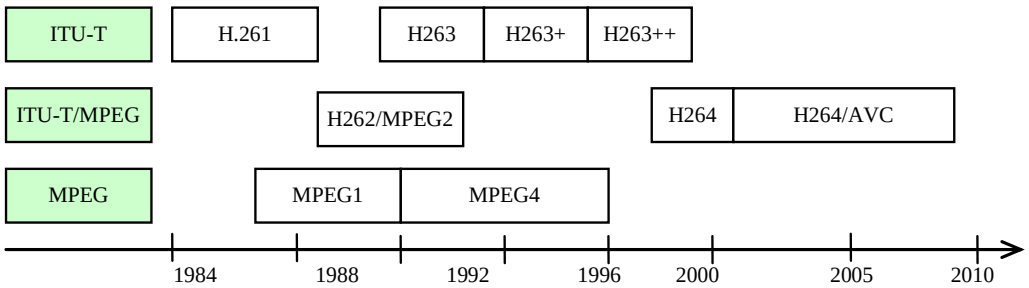



Figure 2.3 : Evolution des standards de compression vidéo.

| Standard | Application |
|---|---|
| MPEG-1 | Vidéo-CD, CDI |
| MPEG-2 | Télévision numérique, Satellite, TNT, TV Haute Définition, DVD |
| MPEG-4 Visual | Internet, vidéo mobile, VOD (Video On Demand), Studio |
| MPEG-4/AVC H.264/AVC | Internet, vidéo mobile, Haute Définition, VOD (Video On Demand), Studio |

Tableau 2.2 : Les normes de codage vidéo et leurs applications.

## 3. La nouvelle norme de compression H.264/AVC

La norme H.264 est spécialement conçue pour remédier à plusieurs faiblesses des normes de compression vidéo précédentes. Cette norme est basée sur un codage hybride fondé sur les blocs et une amélioration des concepts et outils existants, notamment pour les prédictions Intra et Inter. L'amélioration des performances est significative sur le taux de bits avec la même qualité visuelle, mais la complexité algorithmique est bien plus élevée [38]. H.264 offre les avantages suivants :

- A une qualité vidéo équivalente, H.264 offre la réduction moyenne du débit de 50% comparé à MPEG-2. Ce qui donne la possibilité de transmettre de la télévision à haute définition et de la télévision mobile avec un taux de bits raisonnable ;
- Les erreurs de transmission sur différents réseaux sont tolérées (Tolérance d'erreurs) ;

- Latence réduite et meilleure qualité en cas de latence supérieure ;
- Spécification de syntaxe simple facilitant l'implémentation logicielle et matérielle ;
- Décodage correspondant exactement à la source et définissant de façon exacte les numérisations qui doivent être effectuées par un encodeur et un décodeur pour éviter l'accumulation d'erreurs.

### 3.1. Présentation générale

Le document « Recommandation H.264 : Advanced Video Coding [46]», publié en 2003, ne définit pas un codec à proprement parler, mais plutôt une syntaxe pour le flux vidéo compressé et une méthode pour décoder cette syntaxe et produire une séquence vidéo affichable [34].

#### 3.1.1. Division des images en blocs et macroblocs

Une séquence vidéo compressée en utilisant H.264/AVC doit être constituée de plusieurs images, de type différent, structurées en GOP (Group Of Pictures). Un GOP (Figure 2.4) est une suite d'images codées suivant trois méthodes : codage intra-image (I-frame), prédictif (P-frame) et bidirectionnel (B-frame) [4].

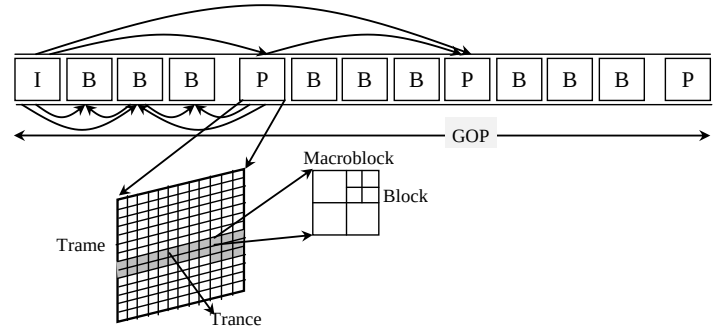



Figure 2.4 : Organisation des images dans une séquence vidéo H.264.

Dans un GOP, l'image "I" est une image de référence codée, indépendamment des autres, par prédiction Intra. Elle fournit les points d'accès dans le train binaire, elle sert aussi au codage des images de type P et B, son taux de compression est moyen. L'image "P" est une image codée par prédiction Inter en utilisant l'estimation et la compensation de mouvement à partir des autres images codées et décodées (I et/ou P). Dans la dernière image de type "B", les macroblocs peuvent être prédits soit par simple compensation de mouvement avant ou arrière à partir d'images "P" ou "I" soit par double compensation de mouvement avant et arrière.

Chaque image (ou bien trame) est partitionnée en macroblocs de dimension fixe qui couvrent une zone rectangulaire de l'image (16×16 échantillons de luminance et de 8×8 échantillons des deux composants de chrominance) représentant les unités maximales de codage (Figure 2.5). Ces macroblocs sont séparés en sous-macroblocs qui peuvent avoir les tailles suivantes (16×8), (8×16), (8×8), (8×4), (4×8) ou (4×4) [49].

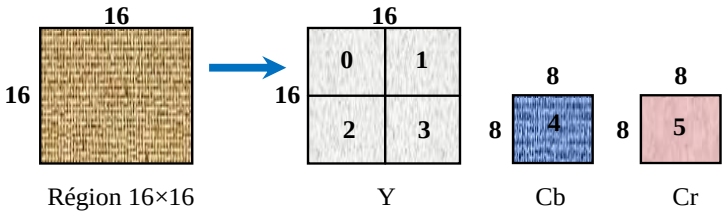



Figure 2.5 : Les macroblocs 16×16 et 8×8 dans H.264.

### 3.1.2. Les types de prédiction dans H.264

La séquence en entrée, d'un codeur vidéo, est une succession d'images. Chaque image est découpée en slices (tranches) qui représentent en général des sous-ensembles d'une image. Un slice est, donc, une partie de l'image (ou l'image entière) constitué d'un ensemble de macroblocs et codé indépendamment [50]. L'ordre de transmission des macroblocs dans le flux dépend de la table d'allocation de macroblocs et ne correspond pas nécessairement à l'ordre de balayage de la matrice.

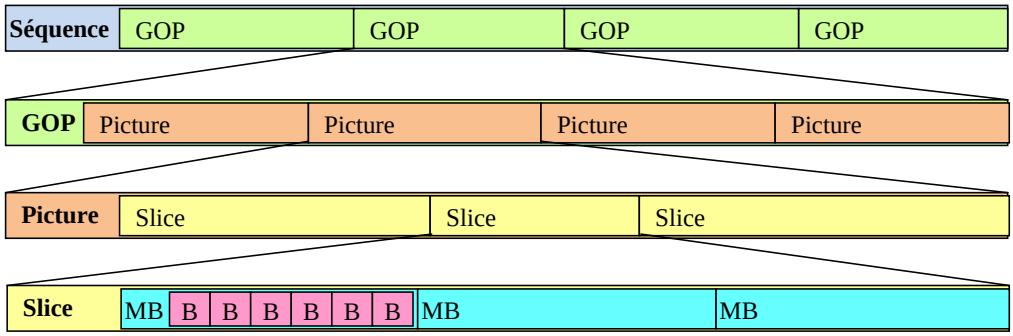



Figure 2.6 : Structure du train binaire.

H.264 prend en charge cinq types de codage par slice. En plus des slices (I, P et B), H.264 prend en charge deux autres types de slices : SP (commutation P) et SI (commutation I). Ces deux slices sont spécifiées pour une commutation efficace entre les flux codés à différents débits. Les signaux de prédiction inter-image des flux pour une slice SP sélectionnée sont quantifiés dans le domaine transformé, les forçant dans une gamme d'amplitude plus approximative. Cette dernière permet un codage de faible débit binaire du signal de différence entre les flux. Les images SI sont définies pour parvenir à une correspondance parfaite des images SP si la prédiction inter-image ne peut être utilisée en raison d'erreurs de transmission [39].

### 3.1.3. Les critères de distorsion dans H.264

Dans un encodeur H.264, les deux modes de prédiction sont basés sur la comparaison des macroblocs dans la même image (prédiction Intra) ou bien à partir des images différentes (prédiction Inter). La comparaison se résume à la minimisation d'un des critères de distorsion. Le choix de l'opérateur de distorsion n'est pas imposé par les deux comités de standardisation (ISO/IEC, UIT-T), mais le plus souvent utilisé c'est le SAD [51].

## 3.2. Vue d'ensemble du codec H.264/AVC

La norme visuelle H.264 partage un certain nombre de dispositifs avec les normes passées, y compris H.263 et MPEG-4. Principalement, H.264 est basée sur un modèle hybride de la modulation de code d'impulsion différentielle adaptative (ADPCM=Adaptive Differential Pulse Code Modulation) et une transformation basée sur le codage des entiers semblable à la transformation des cosinus discret (DCT) utilisé dans des normes antérieures [48]. Ce codage complexe est fait pour tirer profit des redondances temporelles et spatiales qui se produisent dans les images visuelles successives. L'utilisation d'algorithmes complexes au niveau de l'encodeur implique une charge plus importante des calculs. Par comparaison, l'encodeur H.264 est dix fois plus complexe que l'encodeur MPEG-2 pour un gain de 50% en taille de fichier [38].

### 3.2.1. Flux de données dans H.264

Le standard H.264/AVC regroupe une couche vidéo (Video Coding Layer | VCL), qui représente le contenu vidéo, et une couche réseau (Network Abstraction Layer | NAL), qui comprend les en-têtes appropriées pour le transport de l'information par des couches transports ou des supports de stockages particuliers. La conception du VCL suit l'approche de codage vidéo par blocs, comme dans toutes les normes antérieures (MPEG et ITU) [34].

La sortie du processus de codage des données est VCL (une séquence de bits représentant les données vidéo codées) qui sont mappés à des unités NAL avant la transmission ou le stockage. Chaque unité NAL contient une séquence d'octets bruts de charge utile (RBSP), un ensemble de données correspondant à des données vidéo codées ou des informations d'en-tête. Une séquence vidéo codée est représentée par une séquence des unités NAL qui peuvent être transmis sur un réseau en mode paquet ou une liaison de transmission à haut débit ou stockées dans un fichier [39].

### 3.2.2. L'encodeur H.264/AVC

La norme H.264 permet le codage du signal vidéo au format chroma 4:2:0, avec des images progressives ou entrelacées, éventuellement combinées dans une même séquence [4]. La couche VCL est définie pour représenter de manière efficace le contenu des données vidéo. Le concept de la couche de codage vidéo H.264/AVC est semblable à celui des autres normes telles que MPEG-2. Il s'agit d'un hybride de prédiction temporelle et de prédiction spatiale, allié à un codage par transformée. Le standard H.264/AVC est un codeur en boucle fermée [49], le codeur H.264 comprend le décodeur (en boucle) afin de réaliser la prédiction pour les blocs suivants ou bien l'image suivante (Figure 2.7). Les coefficients quantifiés subissent un rééchelonnement et une transformée inverses similaires à ceux du décodeur, aboutissant au reliquat de prédiction décodé. Le reliquat est ajouté à la prédiction. Le résultat de cette opération est transmis au filtre anti-blocs permettant d'éliminer certaines dégradations produites par le module de quantification qui engendre des pertes d'informations. Ce filtre lisse les images de référence en bordure des blocs. Les macroblocs et slices décodés sont ensuite stockés en mémoire. Les blocs décodés de l'image courante sont utilisés pour le calcul des prédicteurs Intra. De même, les images précédemment décodées et filtrées sont utilisées pour le codage Inter.

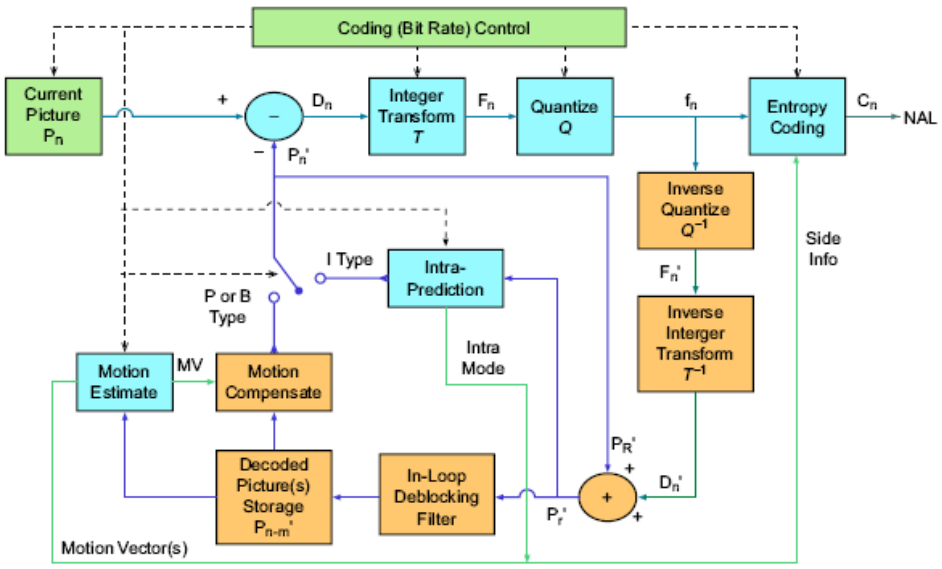



Figure 2.7 : Diagramme de l'encodeur H264 [39].

Dans la chaine de compression directe, et afin que le décodeur soit capable de retrouver le prédicteur Intra/Inter, les modes de prédiction Intra et les vecteurs correspondant au mouvement dans la prédiction Inter sont transmis au décodeur. Ces informations sont envoyées avec les blocs résiduels dans l'encodeur entropique pour former le train binaire [50].

#### 3.2.2.1. Le codage par prédiction dans H.264

Dans les images de type Intra, chaque macrobloc peut être transmis en utilisant un type de codage parmi d'autres en fonction de celui utilisé pour le codage de la tranche. Contrairement aux normes de codage vidéo précédentes où la prédiction intervenait dans le domaine transformé, la prédiction spécifique à la norme H.264/AVC intervient toujours dans le domaine spatial en se référant aux échantillons voisins de blocs déjà codés/décodées mais non filtrés. Dans tous les types de codage de tranches, deux classes de codage intra-image sont pris en charge: Intra4×4 et Intra16×16. Dans le mode Intra4×4, chaque bloc (4×4pixels) de luminance recourt à un des neuf modes de prédiction définit dans le logiciel de référence [47]. En effet, outre la prédiction DC, huit modes de prédiction directionnelle sont spécifiés. Dans le mode Intra16×16 (recommandé dans le cas des zones d'image régulières) une prédiction uniforme est réalisée pour l'ensemble de la luminance d'un macrobloc. Pour ce mode de prédiction, quatre modes de prédiction coexistent. La prédiction des échantillons de chrominance d'un macrobloc repose toujours sur une technique semblable à celle utilisée pour la luminance (Intra16×16). La prédiction intra-image au-delà des limites de la tranche n'est pas autorisée afin de maintenir l'indépendance des tranches les unes par rapport aux autres.

Outre les types de codage intra-image des macroblocs, plusieurs types de codage prédictif ou à compensation de mouvement sont définis pour les macroblocs d'une tranche P. Des partitions avec des macroblocs et des sous-macroblocs sont prises en charge par la syntaxe correspondant aux types de prédiction Inter (Inter16×16, Inter16×8, Inter8×16 et Inter8×8). Dans le mode Inter8×8, un élément syntaxique supplémentaire est transmis pour déterminer les autres types de prédiction (Inter8×4, inter4×8 ou Inter4×4).

Le signal de prédiction pour chaque bloc de luminance N×M à codage prédictif est obtenu en déplaçant une zone de l'image de référence. Il est définit par un vecteur de translation et un index de référence d'image. Par conséquent, si le macrobloc est codé en utilisant le mode Inter8×8, et que chaque sous-macrobloc est codé en utilisant l'Inter4×4, un maximum de 16 vecteurs de mouvement peut être transmis pour un seul macrobloc de tranche P [49].

#### 3.2.2.2. La transformation et la quantification dans H.264

A l'instar des normes de codage vidéo précédentes, la norme H.264/AVC a également recours au codage par transformée du reliquat de la prédiction. Toutefois, dans le cas de H.264/AVC, la transformée est appliquée à des blocs de taille 4×4 pixels. On utilise ici, au lieu d'une transformée en cosinus discrète (DCT), une transformée des nombres entiers dont les propriétés sont pratiquement les mêmes que celles d'une DCT. Cette transformation est utilisée pour éviter les erreurs d'arrondi de la DCT. De même pour la transformée inverse réalisée par des opérations sur des entiers exactes, ceci implique que les discordances de la transformée inverse sont évitées [51].

Une transformation supplémentaire (2×2) est appliquée aux quatre coefficients DCT de chaque composant de chrominance. En plus, si un macrobloc est codé en mode Intra16×16, une troisième

transformation de Hadamard (4×4) est appliquée avec comme entrée les 4×4 composantes continues résultantes de la DCT des blocs 4×4 du signal de luminance. L'empilage des transformées de blocs équivaut à une extension de la longueur des fonctions de la transformée. Les coefficients de cette dernière transformation sont utilisés pour le calcul du coût pour le mode Intra16×16 [52].

Pour quantifier les coefficients de la transformée, la norme H.264/AVC utilise une quantification scalaire. Un des 52 quantificateurs est sélectionné pour chaque macrobloc par le paramètre de quantification (QP). Les quantificateurs sont disposés de manière à obtenir une augmentation approximative de 12,5% de la taille du pas de quantification lorsque QP augmente de 1 [53].

Les coefficients quantifiés d'un bloc sont en général analysés en zigzag et transmis par codage entropique. Pour les blocs formant partie d'un macrobloc codé en mode trames, une autre structure d'analyse est utilisée. Les coefficients DCT 2×2 des chrominances sont analysés suivant l'ordre de balayage de la matrice. Toutes les transformées de la norme H.264 peuvent être implémenté en appliquant simplement des ajouts ou des décalages binaires aux valeurs entières sur 16 bits. Donc, la sortie de la DCT et de la quantification est une matrice creuse contenant quelques valeurs non nulles. Avant l'envoi vers l'encodage d'entropie, les coefficients nuls sont regroupés et recodés dans un ordre plus économique en terme de place. Avec un parcours de la trame en mode Zig-Zag, nous allons obtenir une longue série de zéro. En utilisant une méthode RLE (Run-Length Encoding), nous regroupons les zéro successifs.

#### 3.2.2.3. La reconstruction des images

Parmi les nouveautés de H.264 est la reconstruction des images dans l'encodeur comme dans le décodeur, et cela dans le but de manipuler les mêmes images au niveau de l'encodeur qu'au niveau du décodeur. En effet, dans l'encodeur et le décodeur H.264, les macroblocs résiduels après quantification et transformation inverses sont combinés avec les macroblocs déjà décodé pour former le nouveau macrobloc reconstruit. Pour le codage intra, nous ajoutons le macrobloc résiduel au macrobloc reconstruit selon le mode intra choisi dans l'étape de codage, alors que pour le codage inter-image nous ajoutons ce résiduel au macrobloc reconstruit suivant les vecteurs de mouvement indiqués dans le bitstream [39].

Généralement, pour améliorer la qualité visuelle des images après encodage/décodage, il est nécessaire d'utiliser un filtrage après l'étape de transformation inverse dans le décodeur. En effet, l'un des artéfacts les plus visibles, dans les techniques basées sur les blocs, est l'apparition de l'effet de blocs dans les images décodées (la visibilité de la structure en blocs ou bien l'effet de pixellisation). Les bords d'un bloc sont en général reconstitués avec moins de précision que les pixels intérieurs.

De nombreux filtres, débloquant atténuant les contours des blocs et MB, sont définis dans la littérature, et cela par lissage de pixels. H.264/AVC inclus un filtre anti-blocs (deblcoking filter) lissant les trames lors de la reconstruction. Ce filtre est jugé, dans plusieurs travaux, hautement adaptatif où la puissance du filtrage est contrôlée par les valeurs de plusieurs éléments syntaxiques. La pixellisation est alors réduite sans affecter outre mesure la netteté du contenu et la qualité subjective est considérablement améliorée [49].

En plus, l'une des nouveautés de H.264 est l'utilisation du filtre en boucle au niveau de l'encodeur, cela est dans le but de manipuler les mêmes images, pour la reconstruction Inter, au niveau de

l'encodeur qu'au niveau du décodeur. Ceci permet une prédiction de meilleure qualité apportant des gains dans le bit-rate typiquement de 5 à 10 %, sans diminution de la qualité tout en produisant la même qualité objective que la vidéo non filtrée [54].

#### 3.2.2.4. Le codage entropique

Le codage entropique est une méthode de compression sans perte ; ainsi, il n'y a aucune dégradation de l'image décodée. L'encodeur d'entropie utilisera la redondance statistique pour recoder les vecteurs et les coefficients de manière plus compacte. Donc, l'entrée de l'encodeur est une séquence des vecteurs de mouvement, des modes de prédiction Intra et des blocs résiduels, et la sortie est une séquence compressée plus un en-tête.

La norme H.264/AVC prend en charge deux méthodes de codage entropique. Le codage par défaut utilise un unique ensemble illimité de mots-codes défini pour tous les éléments syntaxiques, à l'exception des coefficients de transformée quantifiés. Ainsi, au lieu de concevoir une table de code à longueur variable (VLC) différente pour chaque élément syntaxique, seul la correspondance à la table de mots-codes unique est personnalisée en fonction des statistiques de données. La table unique est un code Golomb exponentiel aux propriétés de décodage simples et régulières.

Pour transmettre les coefficients de transformée quantifiés, on utilise la méthode CAVLC (Context-Adaptive Variable Length Coding), plus sophistiquée pour ce type de données. Dans ce cas, les tables VLC de plusieurs éléments syntaxiques sont remplacées en fonction des éléments syntaxiques déjà transmis. Comme ces tables sont spécialement conçues pour cadrer avec les statistiques correspondantes, les performances du codage entropique sont améliorées par rapport à celles obtenues en utilisant une seule table VLC.

L'efficacité du codage entropique peut encore être améliorée avec le codage adaptatif CABAC (Context-Adaptive Binary Arithmetic Coding). L'utilisation du codage arithmétique permet d'assigner un nombre de bits fractionnaire à chaque symbole d'un alphabet, ce qui s'avère extrêmement bénéfique pour les probabilités de symbole nettement supérieures à 0,5. En outre, l'utilisation de codes adaptatifs permet une adaptation à des statistiques de symbole variables. Une autre propriété du CABAC est son adaptation au contexte. Les statistiques des éléments syntaxiques déjà codés sont utilisées pour évaluer les probabilités conditionnelles. Ces dernières sont utilisées pour alterner plusieurs modèles de probabilités estimés [38].

Dans la norme H.264/AVC, le moteur fondamental du codage arithmétique et son estimation de probabilités connexe sont définis comme des méthodes de faible complexité, sans multiplication, qui utilisent uniquement les commutations et les vérifications de tables. Par rapport au CAVLC, le CABAC garantit en général une réduction du débit binaire de 10 à 15% lors du codage de signaux TV d'une même qualité.

### 3.2.3. Le décodeur H.264

Le processus de décodage consiste à interpréter les symboles codés à partir d'un bitstream conformes, et traiter ces données conformément à la spécification du standard, afin de générer des séquences vidéo reconstruite. Ainsi, le processus de décodage est généralement constitué de deux voies principales : la génération des macroblocs prédits et le décodage des macroblocs résiduels. Les valeurs des

échantillons des macroblocs résultant de ces deux voies sont additionnées [55]. Le schéma généralisé d'un décodeur H.264/AVC est donné sur la figure suivante.

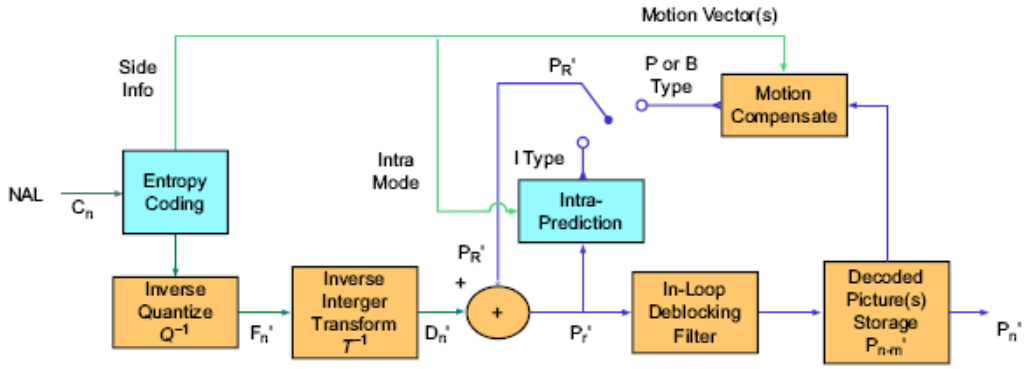



Figure 2.8 : Architecture du décodeur H.264 [39].

### 3.2.4. Les profils et les niveaux de H.264/AVC

Dans la norme H.264, trois profils sont développés, chaque profil est adapté à des domaines d'applications bien définis [49]. Un profil de base (baseline profile) utilise uniquement des images de type 'I' et 'P'. Ce profil utilise uniquement les algorithmes Intra et Inter, l'encodeur entropique CAVLAC et le filtre anti-blocs. Le deuxième profil est le profil principal (main profile) qui manipule, en plus des images 'I' et 'P', des images intermédiaires de type 'B'. Pour ce profil, des améliorations sont apportées sur l'algorithme inter-prédiction et le codage entropique CABAC. Ensuite le profil étendu (Extended profile), utilise en plus des images de type SI et SP avec des améliorations significatives au niveau des algorithmes. Aux trois premiers profils, un nouveau profil (Hight profile) a été ajouté [32]. La Figure 2.9 montre l'interrelation entre les trois versions. En plus, le tableau 2.3 montre les détails des trois profils [56].

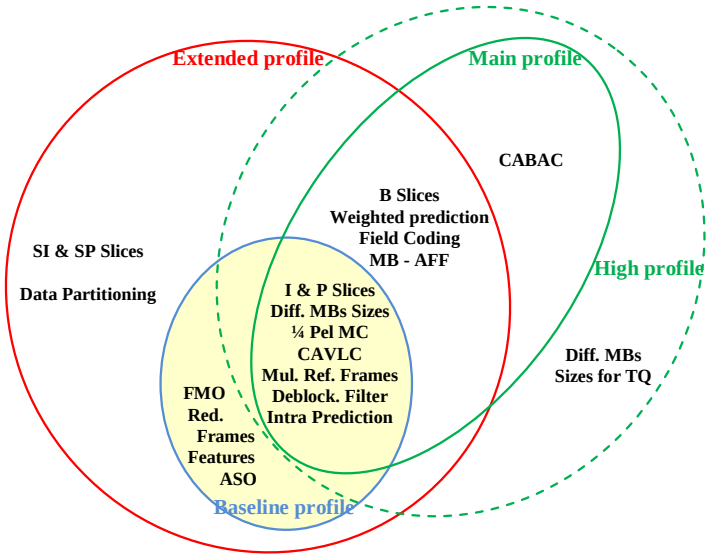



Figure 2.9 : Les versions de H.264/AVC et les outils correspondants.

En plus, récemment trois dérivées du dernier profil sont définies (Hi10P, Hi422P et Hi444P) [57]. Les détails de chaque profil est comme suit :

- Le profil High10 (Hi10P) : Ce profil va au-delà des applications grand public et s'appuie sur le profil High, ajoutant jusqu'à 10 bits de précision par pixel.

- Le profil High4:2:2 (Hi422P) : Le profil principal pour les applications professionnelles, il s'appuie sur le profil High10, ajoutant le support pour la quantification 4:2:2 jusqu'à 10 bits/pixel.
- Le profil High4:4:4 (Hi444P) : Ce profil s'appuie sur le profil High4:2:2, ajoutant le support pour la quantification 4:4:4, jusqu'à 12 bits/pixel et en plus le support pour un mode sans perte efficace.

| Outils | Baseline | Main | Extended | High |
|---|---|---|---|---|
| I and P Slices | X | X | X | X |
| CAVLC | X | X | X | X |
| CABAC | | X | | X |
| B Slices | | X | X | X |
| Entrelacé (PicAFF, MBAFF) | | X | X | X |
| Enh. Error Resil. (FMO, ASO, RS) | X | | X | X |
| Further Enh. Error Resil (DP) | | | X | X |
| SP and SI Slices | | | X | X |
| Weighted prediction | | X | X | X |
| Intra prediction | X | X | X | X |
| Variable block size MC | X | X | X | X |
| ¼ pel MC | X | X | X | X |
| Multiple ref. frame | X | X | X | X |
| In loop deblocking filter | X | X | X | X |
| 8×8 Transform | | | | X |
| 10 bits | | | | X |
| 4:2:2 and 4:4:4 | | | | X |
| lossless | | | | X |

Tableau 2.3 : Les trois profils de la norme H.264.

Chaque profil a ensuite différents niveaux de capacité, appelés « Levels ». On trouve par exemple 14 levels pour le profil principal. Tous comportent des outils communs (les slices, le type de prédiction, le type de codage, etc.). Donc, un level spécifie des bornes en termes de taille d'image, de fréquence d'image et de débit compressé.

La norme MPEG-4 AVC permet un fort taux de compression au prix d'une complexité élevée. Tout en respectant la norme, la complexité d'un décodeur peut être bornée grâce aux profils et levels. Quant à l'encodeur, l'implantation des outils est au choix du fabriquant, indépendamment de la norme. En effet, un encodeur respectant la norme, à un profil et un level donné, peut ne pas implémenter tous les outils afin de réduire sa complexité. Cependant, cela se traduit généralement par une perte d'efficacité. Des compromis sont souvent réalisés sur l'un ou bien sur plusieurs modules de l'encodeur [2].

### 3.3. Le codage intra-image

Comme mentionné auparavant, la prédiction et l'encodage Intra exploitent la redondance spatiale de l'image de référence de type I, celle-ci est décomposée en macroblocs chacun de taille 16×16 pixels. Ce principe est identique au MPEG-2 et MPEG-4 part-2, mais les techniques suivantes sont spécifiques au standard H.264 :

- Les macroblocs de taille 16×16 pixels peuvent être décomposés en 16 blocs de taille 4×4 pixels ;
- 4 prédictions sont définies pour les macroblocs 16×16 (vertical, horizontal, DC et plan) ;
- 9 prédictions sont définies pour les blocs 4×4 (vertical, horizontal, DC et 6 diagonaux).

Dans le codage intra-image, la prédiction d'un macrobloc de l'image de référence est réalisée en utilisant les pixels de macroblocs voisins précédemment encodés et reconstruits mais non filtrés, à l'aide de deux méthodes (Intra16×16 et Intra4×4). Généralement, dans une implémentation du codeur

intra, les deux méthodes (avec leurs modes) sont évaluées, et on sélectionne la meilleure combinaison en comparant les coûts de calcul [58].

### 3.3.1. Prédiction Intra4×4

Dans la Figure 2.9, les pixels {a, b, c, …, p}, du bloc en cours de traitement, sont interpolés à partir des pixels, en haut et à gauche, déjà décodés {A, B, C, …, L}, et cela suivant plusieurs modes [39]. Les modes disponibles pour chaque bloc, dépendent des pixels voisins disponibles. Si des pixels voisins manquent (exemple du premier bloc de la trame), des valeurs par défaut sont utilisées. Dans H.264, neuf modes ont été choisis pour l'Intra4×4.

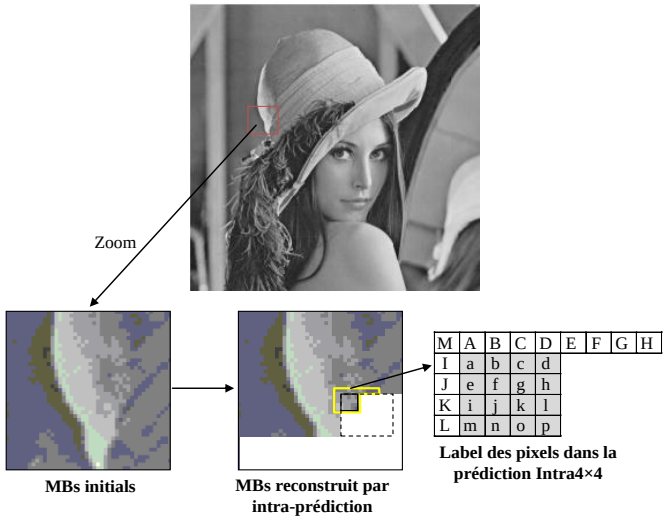



Figure 2.10 : Prédiction Intra4×4 à l'aide des pixels voisins dans H.264.

Dans les deux types de prédiction, le calcul du mode se fait par interpolation des voisins selon un angle défini dans la Figure 2.11. Pour l'Intra4×4, huit directions ont été choisies. Si nous prenons le premier mode (direction verticale), les pixels {a, b, c, … et p} sont interpolés verticalement, ainsi les pixels a, e, i et m reçoivent la valeur de A. Un calcul plus complexe est utilisé en mode 7 où l'angle d'interpolation est de $\frac{11\pi}{8}$ radians, ce qui nous donnera par exemple a = $\frac{A+B}{2}$ et e = $\frac{A+2B+C}{4}$. Deux autres modes différents sont aussi utilisés, le mode « DC » où les pixels prennent la moyenne des valeurs de A, B, C, D, I, J, K et L, et le mode « PLANE » où la prédiction est un dégradé oblique, calculé par moyenne des valeurs obliques (une moyenne des deux sens de la direction « mode 3 ») [51].

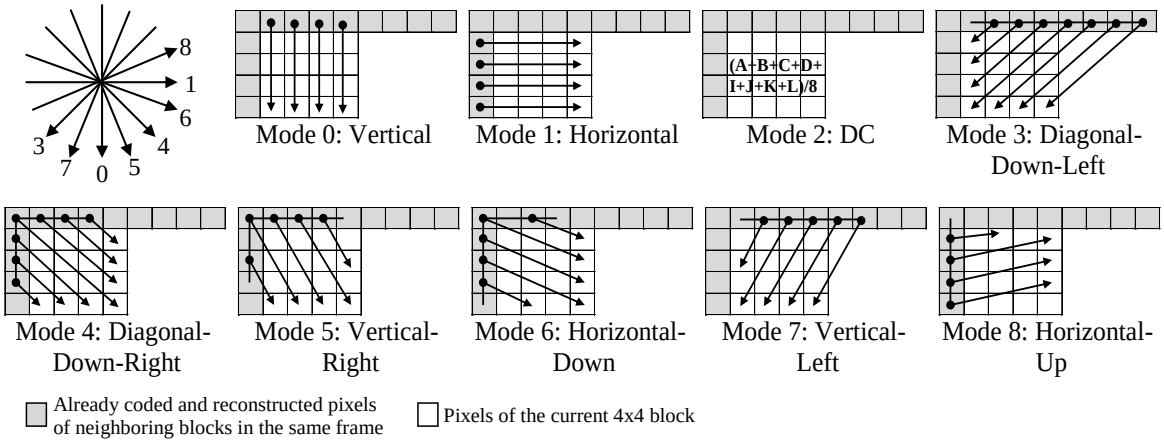



Figure 2.11 : Les 9 modes de prédiction Intra4×4 dans H.264.

### 3.3.2. Prédiction Intra16×16

La même stratégie est appliquée pour l'Intra16×16 avec 4 modes uniquement. La Figure 2.12 montre ces 4 modes basés sur les directions (verticale, horizontale et plane), en plus le mode DC où les pixels du bloc P prennent la moyenne de l'ensemble des pixels H et V :

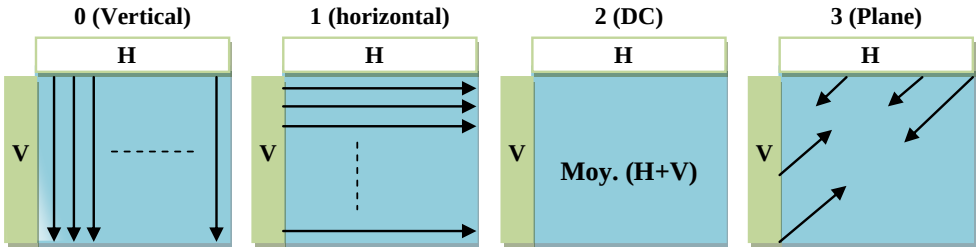



Figure 2.12 : Les 4 modes de prédiction intra16×16 dans H.264.

Les différents modes déjà étudiés, sont appliqués au signal luminance. Pour les blocs 8×8 pixels de chrominance, 4 modes Intra4×4C sont adaptés. Les 4 modes de chroma ont la même architecture que les modes Intra16×16 de la luminance, sauf que l'ordre est inversé (DC=mode0, horizontale=mode1, verticale=mode2 et plane=mode3) [39].

### 3.3.3. Mode de décision et coût de calcul du codage intra-image

Le mode de décision n'est pas spécifié dans la norme H.264/AVC. Cette étape est laissée au choix du programmeur. Les recommandations de H.264/AVC [46] définissent deux types d'algorithmes de décision de mode (haute et basse). Dans la décision de mode de faible complexité, la distorsion est évaluée par le SAD ou bien SATD. Le taux est estimé par le nombre de bits nécessaires pour coder les informations de mode. Dans la décision de mode de haute complexité, la distorsion est évaluée par SSE. Le taux est estimé par le nombre de bits nécessaires pour coder les informations de mode et le résidu. Cette dernière méthode nécessite plus de calculs mais conduit à un meilleur rendement, elle est moins appropriée pour des applications en temps réel [39].

Pour la prédiction Intra4×4, il est nécessaire de calculer 9 coûts pour chaque bloc (16 blocs dans un macrobloc). Chaque calcul de coût nécessite au moins 16 soustractions avec des valeurs absolues et 15 additions. De même pour la prédiction Intra16×16 avec quatre coûts pour chaque macrobloc, et chaque coût nécessite 256 soustractions, 17 transformations de Hadamard (sur des blocs 4×4 pixels) et 255 additions au moins. L'algorithme Full Search consiste à calculer tous les coûts pour les deux modes de prédiction. Le mode (les modes) au minimum de coût est sélectionné. Plusieurs autres algorithmes ont été proposés dans le but d'accélérer les calculs [59-61], et cela par sélection des modes de prédiction.

## 3.4. Le codage inter-image

Le codage inter-image est l'opération de prédiction des macroblocs de l'image prédite (P) à partir des images déjà traitées (de référence). H.264 a la spécification d'utiliser comme images de référence les images déjà compressées, décompressées ***et filtrées***. La prédiction inter-image se résume dans les deux opérations d'estimation et de compensation de mouvement. Le but du codage inter-image est, donc, de représenter l'image à coder en fonction des images de référence et d'un champ de vecteur de mouvement. L'image résiduelle est ensuite encodée par transformation et envoyée avec les vecteurs de mouvement, ce qui assure un taux de compression important et une qualité vidéo meilleure.

### 3.4.1. Prédiction à partir des images de référence

H.264/AVC utilise une ou bien plusieurs images, parmi un nombre d'images précédemment codées, comme référence pour la compensation en mouvement d'un macrobloc. Un macrobloc dans une image P peut être prédit à partir d'une image de référence, et un macrobloc dans une image B peut être prédit à partir d'une, de deux ou bien plusieurs images de référence. Pour chaque pixel, la prédiction est obtenue en calculant la moyenne des pixels de référence. Dans H.264, les images B peuvent aussi servir de référence pour les autres images de même type (Figure 2.13). Deux listes d'images de référence sont gérées au codeur et au décodeur. Plusieurs images de chaque liste peuvent servir comme référence pour prédire l'image courante [2].

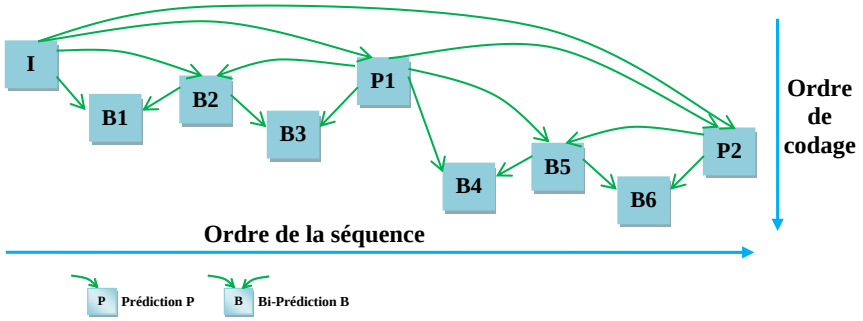



Figure 2.13 : Ordre de codage des images I, P et B dans H264.

### 3.4.2. Estimation et compensation de mouvement dans H.264

Dans un encodeur vidéo, une opération d'estimation de mouvement est nécessaire afin de produire un champ de vecteur le plus précis possible. Les performances d'un encodeur dépendent, donc, de la qualité des champs de vecteurs estimés. Cependant, l'estimation de mouvement est une opération qui requiert de la puissance de calcul. Pour une source vidéo au format 4:2:0, un macrobloc représente l'unité de base pour la prédiction par compensation de mouvement.

#### 3.4.2.1. Choix de partitionnement pour l'estimation de mouvement dans H.264

Dans H.264, le « mode » définit le choix de partitionnement du macrobloc. Une fonction de choix de partitionnement examine tous les partitionnements possibles et choisit le mode minimisant l'erreur selon un critère de distorsions choisi. Le partitionnement est appliqué au macrobloc du signal de luminance (Figure 2.5), les deux chrominance suivront le même mode que celui choisi pour la luminance, mais possèderont des vecteurs de mouvement propres. Un mode « Skipped » définit un macrobloc qui ne sera pas traité.

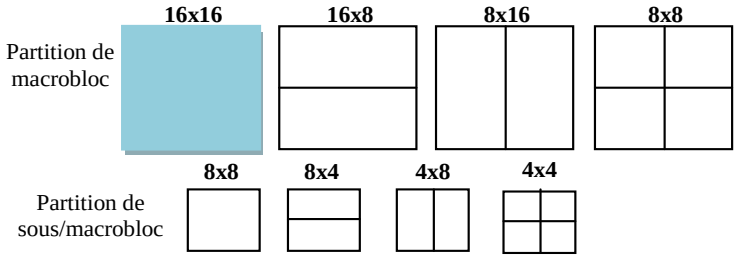



Figure 2.14 : Taille de bloc variable dans H.264.

Chaque macrobloc peut être découpé de quatre manières différentes, ce qui implique que la compensation de mouvement peut être appliquée à une partition 16×16, à deux partitions 8×16, à deux

partitions 16×8 ou bien à quatre partitions 8×8. Si le mode 8×8 est choisi, les quatre sous-macroblocs peuvent à leurs tour être découpés en une partition 8×8, deux partitions 8×4, deux partitions 4×8 ou quatre partitions 4×4 (Figure 2.14). Ces partitions donnent accès à un grand nombre de combinaisons pour chaque macrobloc. Cette méthode est connue sous le nom de compensation de mouvement à structure d'arbre [39].

#### 3.4.2.2. Vecteurs de mouvement entiers dans H.264

Chaque partition est prédite à partir d'une zone de même taille dans une image de référence en utilisant la méthode Block Matching. Le vecteur de mouvement donne la position relative du bloc de référence. Il peut dépasser les limites de l'image. Dans ce cas, les pixels du bord de l'image sont étendus pour reconstruire le bloc de référence. Les vecteurs de mouvement sont également compressés. Ils sont prédits à partir d'un médian, calculé avec les vecteurs des blocs voisins. Seule la différence est codée et transmise au décodeur. Ainsi, plus un champ de vecteur est homogène, moins il coûte cher à coder[38].

Dans H.264, chaque partition ou sous-macrobloc nécessite un vecteur de mouvement. Chaque vecteur doit être codé et transmis et le choix de découpage retenu doit également être inclus dans le bitstream final. Choisir une grande taille de partition (16×16, 16×8, 8×16) entraîne qu'un petit nombre de bits est nécessaire pour signaler les vecteurs de mouvement et le type de partition. Par contre, le résidu obtenu peut contenir une grande quantité d'énergie dans les zones d'image très détaillées. Une petite taille de partition (4×8, 8×4, 4×4) entraîne un résidu d'énergie faible mais nécessite un grand nombre de bits pour coder les vecteurs de mouvement et le type de partition. De plus, le choix de la taille de partition a un impact significatif sur les performances de compression. De manière générale, une grande partition est plus appropriée pour des zones homogènes et une petite pour des zones de détails [35].

Chaque composante de chrominance d'un macrobloc (Cb et Cr) a une résolution moitié moindre que la luminance. Chaque bloc de chrominance est partitionné de la même manière que la luminance, avec une taille de partition divisée par deux horizontalement et verticalement (une partition 8×16 de Luminance correspond à une partition 4×8 de chrominance). Les composantes horizontales et verticales de chaque vecteur de mouvement sont ensuite dédoublées pour les blocs de chrominance.

#### 3.4.2.3. Estimation de mouvement fractionnel dans H.264

Dans H.264 avec un format 4:2:0, la précision des vecteurs de mouvement est d'un quart de pixel pour la luminance et un huitième de pixel pour la chrominance (Figure 2.15). Les mouvements dans les images naturelles n'étant pas limités à un nombre entier de pixels, la qualité de la prédiction est donc améliorée. Dans une première étape, l'estimation de mouvement trouve, pour chaque macrobloc prédit, la meilleure position entière (mouvement entier) qui donne le minimum d'erreur selon le critère choisi. Ensuite, le codeur cherche les positions demi-pixel immédiatement à côté de cette meilleure position entière pour vérifier s'il y a des positions fractionnelle qui minimise encore l'erreur. Enfin et si nécessaire, les positions quart de pixel, à côté de la meilleure position de demi-pixel, sont calculées et comparées. La dernière position relative (nombre entier, demi-pixel, quart de pixel) qui assure le minimum d'erreur est choisie comme vecteur de mouvement [51].

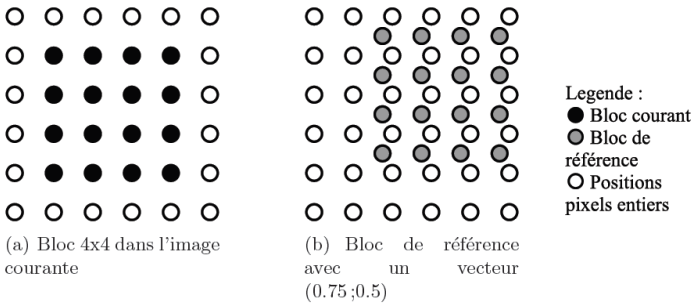
Les valeurs des échantillons aux positions fractionnaires ne sont pas disponibles, il est nécessaire de les interpoler. Le filtre d'interpolation est fixé par la norme afin qu'il soit identique au codeur et au décodeur. Les valeurs de luminance demi-pixel sont interpolées avec un filtre séparable à six

coefficients (1,-5,20,20,-5,1). Elles sont calculées à partir de six pixels adjacents horizontalement et/ou verticalement. Par exemple b, h, et j dans la Figure 2.16 peuvent être calculés de la manière suivante :

$$b = \frac{E - 5F + 20G + 20H - 5I + J}{32}$$

$$h = \frac{A - 5C + 20G + 20M - 5Q + S}{32}$$

$$j = \frac{aa - 5bb + 20b + 20s - 5gg + hh}{32} \text{ ou } j = \frac{cc - 5dd + 20h + 20m - 5ee + ff}{32}$$




(a) Bloc 4x4 dans l'image courante

(b) Bloc de référence avec un vecteur (0.75 ;0.5)

Figure 2.15 : Exemple de prédiction sub-pixélique.

Les valeurs de luminance quart-pixel sont interpolées en faisant une moyenne entre deux échantillons demi-pixels. Par exemple, a et e sont calculés de la manière suivante :

$$a = \frac{G + b + 1}{2} \text{ and } e = \frac{h + b + 1}{2}$$

1. *Echantillons entiers (blocs ombrés avec lettres majuscules) ;*
2. *Positions d'échantillons fractionnaires (blocs non ombrés avec lettres minuscules) :*
   - *½-pixel*
   - *¼-pixel*

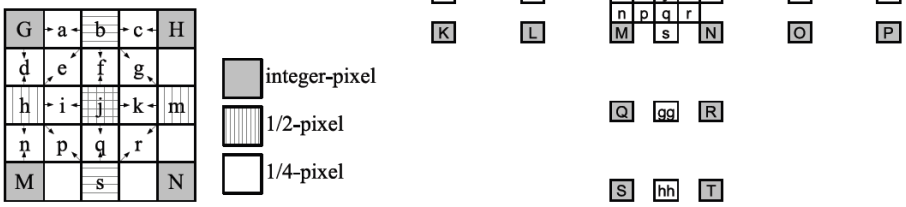


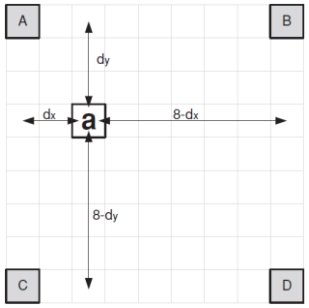

Figure 2.16 : Filtre sous-pixélique de luminance H.264.

Les échantillons de chrominance sont calculés plus simplement, avec un filtre bilinéaire. Pour un format 4:2:0 dans H.264, l'estimation de mouvement à ¼ de pixels pour le signal de luminance nécessite une résolution à 1/8 de pixels pour les deux composantes de chrominance. Les échantillons interpolés sont générés à 1/8 pixels entre les positions entières pour chaque composante chrominance (Figure 2.17), et cela en utilisant une interpolation linéaire. Chaque position sous-pixélique (a) est une combinaison linéaire des positions entière voisines A, B, C et D (Equation 2.8).

$$a = Round\left(\frac{(8 - dx).(8 - dy).A + dx.(8 - dy).B + (8 - dx).dy.C + dx.dy.D}{64}\right) \quad 2.6$$

Dans plusieurs cas, la compensation de mouvement sous-pixélique donne des meilleures performances par rapport à la compensation à pixels entiers. Cette amélioration est plus importante pour l'interpolation à ½ de pixel que pour l'interpolation à un ¼ de pixel [51]. En effet, l'interpolation

demi-échantillon donne un gain notable sur la compensation de mouvement entier de l'échantillon, par contre l'interpolation quart-échantillon donne une légère amélioration, enfin l'interpolation huitième échantillon donne une petite amélioration par rapport aux deux autres.




dx = 2 ;
dy = 3 ;
a = round([30A + 10B + 18C + 6D]/64).

Figure 2.17 : Interpolation sous-pixélique à 1/8 de pixel du signal de chrominance.

L'introduction du filtre à six coefficients et l'augmentation de la précision au quart de pixel permettent à H.264 d'augmenter ses performances de codage grâce à une prédiction de meilleure qualité par rapport à MPEG-2 par exemple, où la précision est au demi-pixel. La compensation de mouvement du standard H.264/AVC est améliorée par rapport à ses prédécesseurs au détriment de la complexité. L'introduction de nombreux modes, telle que la taille de bloc variable et l'amélioration de la précision des vecteurs par exemple, augmentent en effet la complexité. La puissance de calcul nécessaire est alors très élevée, notamment au codeur où tous les modes doivent être évalués et comparés. Afin de réduire l'impact du nombre de modes sur la puissance de calcul, des compromis peuvent être effectués (réduction du nombre de modes ou de la précision), ce qui donne une estimation de mouvement plus grossière et plus rapide. Cela a bien sûr un impact sur les performances d'encodage.

#### 3.4.2.4. Le mode SKIP (Skipped mode)

Outre les modes de macrobloc à compensation de mouvement décrit ci-dessus, un macrobloc de tranche P peut également être codé selon le mode Skipped. Dans ce cas, ne sont transmis ni le signal d'erreur de prédiction quantifié, ni l'index de référence, ni le vecteur de mouvement. Le signal reconstitué est obtenu de la même manière que le signal de prédiction d'un macrobloc référençant l'image. En général, le vecteur de mouvement utilisé pour reconstituer le macrobloc Skipped est le même que le prédicteur du vecteur de mouvement pour le macrobloc. Toutefois, dans certaines conditions, un vecteur de mouvement zéro est utilisé.

### 3.4.3. Prédiction inter-image des images B

Au contraire des normes de codage vidéo précédentes, le concept des tranches B est généralisé dans la norme H.264. Par exemple, des images type B déjà traitée peuvent servir de référence aux autres images de même type. La différence fondamentale entre les tranches B et P est que les premières sont codées pour que certains macroblocs ou blocs puissent utiliser une moyenne pondérée de deux valeurs de prédiction à compensation de mouvement distinctes pour générer le signal de prédiction. En général, les tranches B ont recours à deux mémoires différentes appelées première et seconde mémoire d'images de référence. La répartition des images dans chaque mémoire est gérée au niveau du contrôle de la mémoire. Une opération très semblable à celle des images MPEG-2 B peut ainsi être effectuée.

Dans les tranches B, quatre types de prédiction inter-image sont possibles: liste0, liste1, double (interpolation) et directe. La prédiction liste0 indique que le signal de prédiction est formé en utilisant

la compensation de mouvement issue d'une image de la première mémoire, la liste1 utilise quant à elle une image de la seconde pour générer le signal de prédiction. En mode double prédiction, le signal est formé par une moyenne pondérée d'un signal de prédiction liste0 et liste1 à compensation de mouvement. Le mode de prédiction directe est déduit des éléments syntaxiques précédemment transmis et peut-être une prédiction liste0 ou liste1 ou encore double.

Les tranches B utilisent un partitionnement de macrobloc similaire à celui des tranches P. Outre les modes Inter16×16, Inter16×8, Inter8×16, Inter8×8 et les modes Intra, un type de macroblocs utilisant le mode direct est également proposé. Par ailleurs, pour chaque partition de macroblocs, la méthode de prédiction (liste0, liste1 ou double) peut être choisie séparément. Une partition de 8×8 d'un macrobloc de tranche B peut aussi être codée en mode direct. Si aucun signal d'erreur de prédiction n'est transmis pour le mode de macrobloc direct, on parle alors du mode SKIP de la tranche B. Il peut être codé de manière très efficace à l'instar du mode SKIP dans les tranches P. Le codage du vecteur de mouvement est semblable à celui des tranches P avec les modifications appropriées, parce que les blocs voisins peuvent être codés à l'aide de modes de prédiction différents.

### 3.5. La transformation et la quantification directes et inverses

Dans un encodeur vidéo y compris la norme H.264/AVC, après l'étape de prédiction, les données organisées sous forme de macroblocs sont transformées et quantifiées, et cela avant l'opération finale de codage et de réarrangement en bitstream. De même dans un décodeur, une transformée inverse est préalable à la reconstruction et l'affichage. Plusieurs transformations sont spécifiées dans la norme H.264 : une transformation de base 4×4, des transformations dérivées 4×4 et 2×2 et une transformation haute 8×8. En effet, dans un codec H.264, des transformations et des quantifications directes et inverses sont appliquées sur des blocs 4×4 pixels de luminance et de chrominance. La transformation est une approximation à l'échelle de la fameuse transformation DCT qui peut être calculée en utilisant une arithmétique entière simple. Une étape de normalisation est incorporée dans les opérations de quantification directe et inverse.

#### 3.5.1. Développement de la transformée et de la quantification directes

La transformée de base 4×4 utilisée dans la norme H.264/AVC est une approximation d'échelle de la DCT. La transformation et la quantification sont restructurées de telle sorte que la complexité de calcul soit réduite au minimum. Cet objectif est atteint en réorganisant les processus en un noyau de transformation (Core part) et une partie de mise à l'échelle (Scaling part).

Considérons un bloc 4×4 pixels, ce bloc est traité par une transformation DCT à deux dimensions suivie par une quantification (la quantification dans ce cas est une simple division par un coefficient de quantification $\mathbf{Q_{step}}$ suivie par un arrondissement du résultat). Réorganisons la DCT en un noyau de transformation $\mathbf{(C_f)}$ et une matrice de mise à l'échelle $\mathbf{(S_f)}$. L'étape suivante est de mettre à l'échelle le processus de quantification par une constante ($2^{15)}$ et de compenser en divisant par la même constante et en arrondissant le résultat. Et enfin, combiner $\mathbf{S_f}$ et le processus de quantification dans $\mathbf{M_f}$, la figure suivante montre le principe détaillé, où : $M_f = \frac{S_f.2^{15}}{Q_{step}}$.

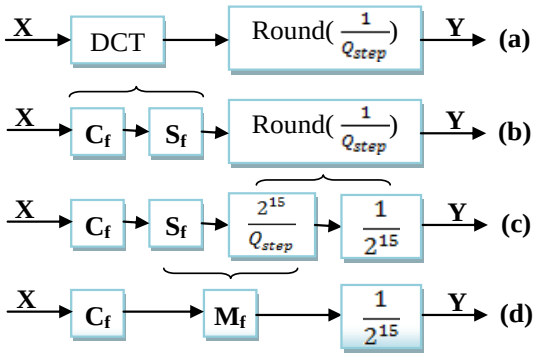



Figure 2.18 : Développement de la transformée et la quantification directes.

### 3.5.2. Développement de la quantification et de la transformée inverses

Considérons l'opération de quantification inverse ou bien de remise à l'échelle (re-scaling), cette opération est suivie par la transformée inverse IDCT à deux dimensions. De même que la transformée directe, réorganisons la IDCT en un noyau **($C_i$)** et une matrice **($S_i$)**. L'étape suivante consiste à corriger le processus de remise à l'échelle par une constante ($2^6$) et à compenser le résultat final en divisant et en arrondissant. Ensuite, combiner le processus de remise à l'échelle et $\mathbf{S_i}$ dans $\mathbf{V_i}$. La figure suivante montre le principe détaillé, avec : $V_i = S_i . 2^6 . Q_{step}$

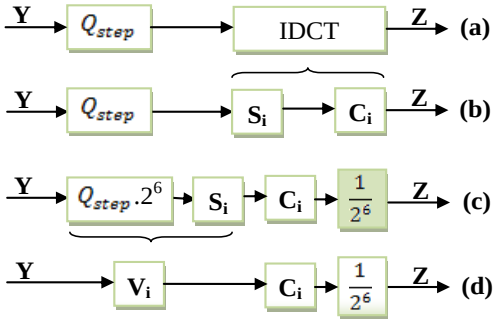



Figure 2.19 : Développement de la transformée et la quantification inverses.

### 3.5.3. Elaboration des matrices de la transformée directe

Considérons la transformation DCT appliquée sur un bloc X de taille 4×4 [39] :

$$Y = A * X * A^T \qquad 2.7$$

Où ( * ) indique le produit matriciel, et :

$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}$, $a = 1/2$, $b = \sqrt{1/2}\cos(\pi/8) = 0{,}653\ldots$ et $c = \sqrt{1/2}\cos(3\pi/8) = 0{,}2706\ldots$

Les lignes de la matrice A sont orthogonales et ont des normes unité (les lignes sont orthonormées). L'utilisation de ces coefficients sur un processeur en pratique nécessite des approximations sur les nombres fractionnaires b et c. Considérons, donc, l'approximation suivant :

$$A_1 = Cf \,.\, Rf \approx A \qquad 2.8$$

Avec : $C_f = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{vmatrix}$ ; et $R_f = \begin{vmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \\ 1/2 & 1/2 & 1/2 & 1/2 \\ 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} & 1/\sqrt{10} \end{vmatrix}$

Le symbole ( . ) dénote la multiplication élément par élément (Hadamard-Schur produit). Noter aussi que la nouvelle matrice $A_1$ est orthonormée. Cette approximation est choisie pour minimiser la complexité de mise en œuvre de la transformation (la multiplication par $\mathbf{C_f}$ ne demande que des additions et des changements binaire) tout en conservant les bonnes performances de compression. La transformée bidirectionnelle devienne :

$$Y = A_1 * X * A_1^T = [C_f . R_f] * X * [C_f^T . R_f^T] \quad 2.9$$

Après réorganisation on obtient :

$$Y = [C_f * X * C_f^T] . [R_f . R_f^T] = [C_f * X * C_f^T] . S_f \quad 2.10$$

La nouvelle transformation devienne alors :

$$W = C_f X C_f^T \quad 2.11$$

Les valeurs absolues des coefficients de la matrice $C_f$ sont soit de 1 ou 2. Ainsi, l'opération de transformation représentée par l'équation 2.12 peut être calculée en utilisant des additions signées et de décalage à gauche, et cela dans le but d'éviter les multiplications par des nombres réels.

Comme montrer dans la Figure 2.18, la matrice [$\mathbf{R_f.R_f^T}$] est combinée avec l'équation de quantification. Après quelques simplifications et réarrangement, les formules de post-mise à l'échelle et de quantification seront données par :

$$Z_{ij} = \text{round}\left(W_{ij} . \frac{MF}{2^{qbits}}\right) \quad 2.12$$

Avec $qbits = 15 + (QP \text{ DIV } 6)$

QP est le paramètre de quantification qui permet à l'encodeur de contrôler avec précision et souplesse le compromis entre le débit et la qualité, il peut prendre n'importe quelle valeur entière de 0 à 51. ($Z_{ij}$) est un élément dans la matrice des résultats du processus de quantification. (*MF*) est un facteur de multiplication qui dépend de QP et la position de l'élément dans la matrice comme indiqué dans le tableau 2.4.

| $QP$ | (i,j) ∈ {(0,0), (2,0), (2,2), (0,2)} | (i,j) ∈ {(1,1), (1,3), (3,1), (3,3)} | Autres Positions |
|---|---|---|---|
| 0 | 13 107 | 5 243 | 8 066 |
| 1 | 11 916 | 4 660 | 7 490 |
| 2 | 10 082 | 4 194 | 6 554 |
| 3 | 9 362 | 3 647 | 5 825 |
| 4 | 8 192 | 3 355 | 5 243 |
| 5 | 7 282 | 2 893 | 4 559 |

Tableau 2.4 : Facteur de multiplication (MF).

Le facteur MF reste inchangeable pour les valeurs de QP> 5. Il peut être calculé en utilisant l'équation suivante :

$$MF_{QP>5} = MF_{QP=(QP \bmod 6)} \quad 2.13$$

L'équation 2.12 peut être représentée avec l'arithmétique entière comme suit :

$$|Z_{ij}| = SHR(|W_{ij}| . MF + f, qbits) \quad 2.14$$

$$Sign(Z_{ij}) = Sign(W_{ij}) \quad 2.15$$

SHR est une procédure de décalage à droite du premier argument avec un nombre de bit égal au deuxième argument. ( $f$ ) est défini dans le logiciel de référence de H.264 égale à de $2^{qbits}/3$ pour les blocs intra et à $2^{qbits}/6$ pour les blocs inter [62].

En plus de la DCT avec des nombres entiers, H.264 utilise la transformée de Hadamard pour le calcul des coûts dans le module Intra16×16. Une transformée de Hadamard $W_{DC}$ d'un bloc 4×4 [$X_{DC}$] est utilisée comme suit :

$$W_{DC} = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{vmatrix} [X_{DC}] \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{vmatrix} \quad 2.16$$

### 3.5.4. Elaboration des matrices de la transformée inverse

De même pour la transformation inverse (IDCT) à deux dimensions appliquée sur un bloc de taille 4×4, dont la formule initiale est donnée par :

$$\mathbf{Z = A^T * Y * A} \quad 2.17$$

Cette transformation est combinée avec la quantification inverse (multiplication par QP) pour avoir des opérations simples. En effet, le processus inverse se réorganise en une partie de rééchelonnement et en une autre partie de transformation inverse à base d'entiers. L'opération de mise à l'échelle de base est donnée par :

$$W'_{ij} = Z_{ij} . \frac{2^{qbits}}{MF} . 64 = Z_{ij} . V_{ij} \quad 2.18$$

Les valeurs de la matrice V sont définies dans la norme pour $0 \leq QP \leq 5$ (Tableau 2.5).

| $QP$ | (i,j) ∈ {(0,0), (2,0), (2,2), (0,2)} | (i,j) ∈ {(1,1), (1,3), (3,1), (3,3)} | Autres positions |
|---|---|---|---|
| 0 | 10 | 16 | 13 |
| 1 | 11 | 18 | 14 |
| 2 | 13 | 20 | 16 |
| 3 | 14 | 23 | 18 |
| 4 | 16 | 25 | 20 |
| 5 | 18 | 29 | 23 |

Tableau 2.5 : Facteur de multiplication (V).

L'équation de transformation dans ce cas est réalisée avec une arithmétique entière en utilisant uniquement des additions, des soustractions et des décalages. La nouvelle formule est donnée par :

$$Z' = C_i{}^T W' C_i \quad 2.19$$

Où la matrice Ci est donnée par :

$$C_i = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{vmatrix}$$

## 3.6. Le filtre anti-blocs (Deblocking filter)

Dans la norme H.264, un filtre est appliqué pour chaque bloc 4×4 pixels décodé, afin de réduire les distorsions à cause de l'application des transformations directes et inverses sur des blocs et non pas sur l'image entière. Le filtre est appliqué après la transformation inverse dans l'encodeur juste avant la

reconstruction et le stockage des macroblocs pour les utiliser dans des futures prédictions Inter, et dans le décodeur juste avant la reconstruction et l'affichage. Le filtre possède deux avantages :

- Les bords des blocs sont lissés, ce qui assure une amélioration visuelle des images décodées (en particulier à un accroissement des taux de compression) ;
- Les macroblocs filtrés sont utilisés pour l'estimation et la compensation de mouvement au niveau de l'encodeur qu'au niveau du décodeur. Donc, l'encodeur et le décodeur manipulent les mêmes images pour la reconstruction des macroblocs.

Le filtre est appliqué successivement, aux frontières verticales et horizontales des blocs 4×4 pixels, dans un macrobloc, comme le montre la Figure 2.20, suivant l'ordre suivant :

1. Filtrer les 4 limites verticales de la composante de luminance (dans l'ordre A, B, C, D) ;
2. Filtrer les 4 limites horizontales de la composante de luminance (dans l'ordre E, F, G, H) ;
3. Filtrer les 2 limites verticales de chacune des composantes de chrominance (dans l'ordre I, J) ;
4. Filtrer les 2 limites horizontales des deux composantes de chrominance (dans l'ordre K, L).

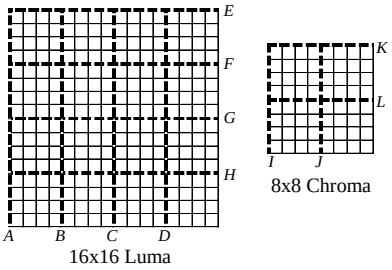



Figure 2.20 : Les frontières verticales et horizontales sujet du filtrage dans un macrobloc.

Chaque opération de filtrage affecte jusqu'à trois pixels de chaque côté de la frontière comme montré dans la Figure 2.21. Sur cette figure, il est mentionné de chaque côté d'une frontière verticale ou horizontale des blocs adjacents p (p0, p1, p2, p3) et q (q0, q1, q2, q3). Suivant plusieurs facteurs (le quantifieur utilisé, le type de codage des blocs, le type de décomposition du macrobloc, etc.), plusieurs résultats sont possibles, allant à des pas de pixels filtrés à 3 pixels de chaque coté qui sont filtrés.

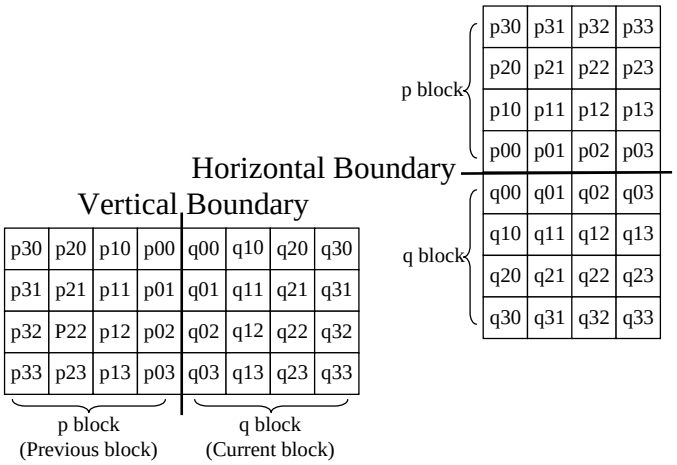



Figure 2.21 : Les pixels affectés par le filtre suivant les frontières verticales et horizontales.

### 3.6.1. Choix de la force de frontière (Boundary Strength)

Le choix de l'ordre de filtrage dépend de la force de frontière (BS) et du gradient de l'image résiduelle. BS est choisie selon les règles suivantes :

| | |
|---|---|
| p et/ou q sont codés en Intra et la bordure filtrée est une bordure du MB | BS = 4 |
| p et q sont codés en Intra et la bordure filtrée n'est pas une bordure du MB | BS = 3 |
| p et q ne sont pas codés en Intra ; p et q contiennent des coefficients codés | BS = 2 |
| p et q ne sont pas codés en Intra ; p et q ne contiennent pas des coefficients codés ; p et q utilisent des images de référence différentes ou bien un nombre différent de l'image de référence ou bien ont des vecteurs de mouvement différents | BS = 1 |
| Autres cas possibles | BS = 0 |

Tableau 2.6 : Calcul de la force de la frontière (BS).

L'idée de calculer la valeur de BS consiste à appliquer le filtre au maximum à des endroits où il est susceptible d'avoir une importante distorsion comme par exemple dans la limite d'un macrobloc Intra-codé ou dans la limite entre les blocs qui contiennent des coefficients codés.

La décision de filtrage au niveau d'une frontière sur les pixels (p2, p1, p0, q0, q1, q2) est prise uniquement si BS > 0 ; et $|p0-q0| < \alpha$ and $|p1-p0| < \beta$ and $|q1-q0| \leq \beta$. Les seuils α et β sont définis dans la norme H.264 en fonction des paramètres de quantification de deux blocs adjacents contenant les pixels p et q, l'idée de la décision de filtrage est de lancer le filtre quand il ya un changement significatif (gradient) à travers la frontière (p-q) dans l'image originale. Avec la remarque que la définition d'un changement significatif dépend de QP. Lorsque QP est petit, on obtient un gradient très faible à travers les frontières des blocs qui est généralement dû aux caractéristiques d'image ce qui implique le choix des seuils plus faibles pour éviter de filtrer les frontières de l'image (des informations utiles). Lorsque QP est plus grand, les distorsions des frontières sont plus significatives ce qui implique le choix des seuils plus élevés.

### 3.6.2. Implémentation logiciel du filtre

Après le calcul de BS, un filtre est appliqué (ou non) selon les règles suivantes [39] :

a) Si BS < 4: Un filtre RIF (4-Tap Finite Impulse Response Filter) est appliqué avec comme entrées les pixels p1, p0, q0 et q1, produisant les pixels filtrés P0 et Q0. Si | p2-p0 | est inférieur au seuil β, un autre filtre (4-Tap) est appliqué avec comme entrées les pixels p2, p1, p0 et q0, produisant la sortie filtrée P1. Si | q2-q0 | est inférieur à β, un filtre (4-Tap) est appliqué avec les entrées q2, q1, q0 et p0, pour produire la sortie filtrée Q1.

b) Si BS = 4 : Le filtrage est appliqué selon les règles dans le tableau 2.7.

| Bloc | Condition | L'entrée | Le filtre FIR | La sortie |
|---|---|---|---|---|
| **p** | If $\|p2-p0\| <\beta$ and $\|p0-q0\| <$ round ($\alpha$ / 4 ) et le bloc est un signal Luma | p2, p1, p0, q0, q1 | 5-tap | P0 |
| | | p2, p1, p0, q0 | 4-tap | P1 |
| | | p3, p2, p1, p0, q0 | 5-tap | P2 |
| | Si non | p1, p0, q1 | 3-tap | P0 |
| **q** | If $\|q2-q0\| < \beta$ and $\|p0-q0\| <$ round($\alpha$ / 4 ) et le bloc est un signal Luma | q2, q1, q0, p0, p1 | 5-tap | Q0 |
| | | q2, q1, q0, p0 | 4-tap | Q1 |
| | | q3, q2, q1, q0, p0 | 5-tap | Q2 |
| | Si non | q1, q0, p1 | 3-tap | Q0 |

Tableau 2.7 : Implémentation du filtre lorsque BS = 4.

### 3.6.3. Organigramme du filtre

Selon le tableau 2.7, pour que le filtre soit hautement adaptatif, plusieurs conditions sont ajoutées. Ces conditions déterminent si une extrémité d'un bloc 4×4pixels sera filtrée ou non filtrée, avec prise en compte de la valeur de BS. L'algorithme du filtre peut ne rien modifier ou bien modifier jusqu'aux 3

pixels sur les deux faces d'un bord des blocs, et cela en fonction des résultats de ces conditions [63]. La Figure 2.22 montre l'algorithme global du filtre anti-blocs.

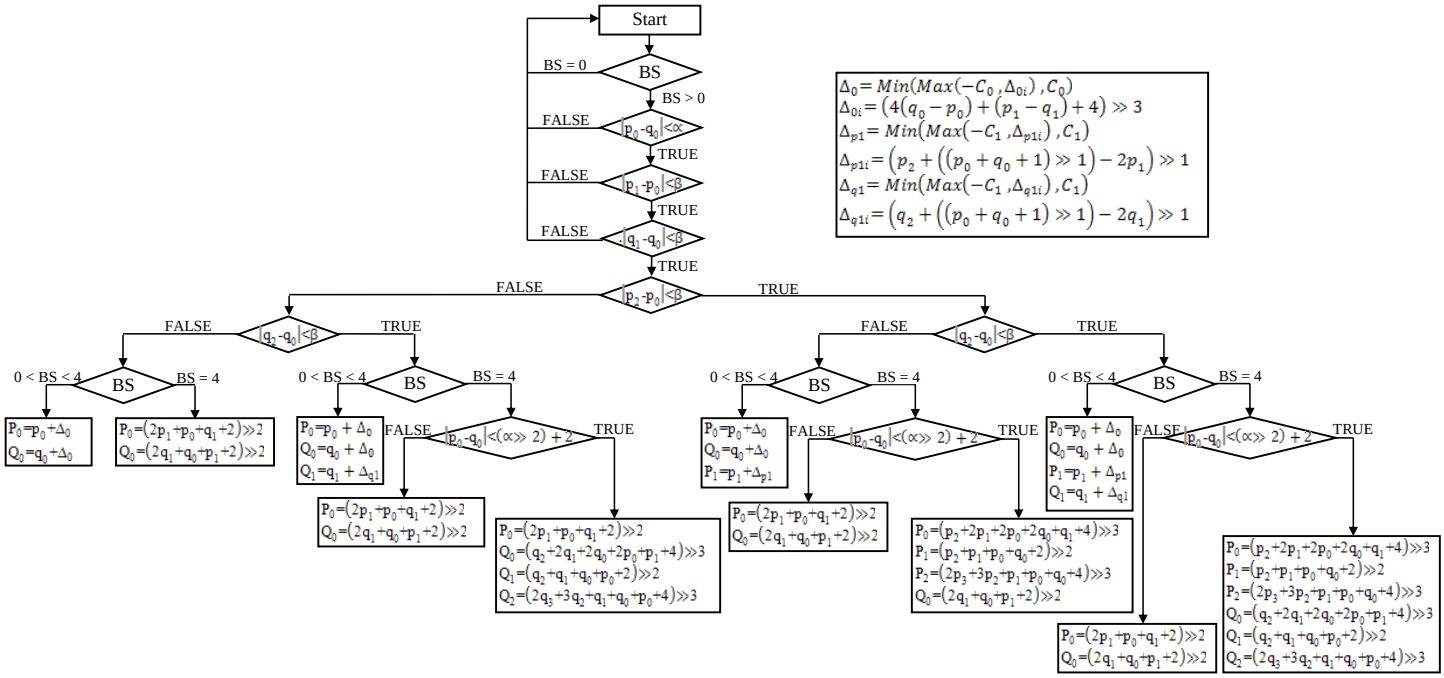



Figure 2.22 : L'organigramme du filtre anti-blocs utilisé dans H.264 [63].

## 3.7. Organisation des données à la sortie de la chaîne directe

Dans un codeur vidéo, composé de plusieurs modules de traitement, il est nécessaire de connaître le type, la taille et l'ordre des données d'entrée, aussi il faut connaître les mêmes caractéristiques des données en sortie, pour pouvoir les arranger avant leurs traitements par d'autres modules, leurs enregistrement dans une mémoire, ou bien leurs affichages. En effet, dans les modules de traitement d'un encodeur, l'ordre de traitement des données (des macroblocs) n'est pas toujours séquentiel, vue la grande dépendance de traitement entre ces macroblocs. Dans cette partie de notre travail, nous allons indiquer l'ordre des macroblocs traités par les différents modules de traitement dans l'encodeur H.264.

### 3.7.1. Flux de données de la prédiction Intra

Les données dans un macrobloc résiduel traité en mode Intra sont transmises, après transformation et quantification, selon l'ordre illustré dans la Figure 2.23. Si le macrobloc est codé en utilisant le mode Intra16×16, le bloc marqué "-1" est transmis en premier, contenant les coefficients DC de chaque bloc de luminance. Ensuite, les blocs de luminance résiduelle de "0" à "15" sont transmis dans l'ordre indiqué (avec des coefficients DC mis à zéros). Juste après les blocs "16" et "17" (2×2 coefficients DC des deux composantes couleurs Cb et Cr). Enfin, les blocs de chrominance résiduelle de "18" à "25" sont envoyés (avec des coefficients DC mis à zéros).

Si le macrobloc est codé en utilisant le mode Intra4×4, le bloc marqué "-1" n'est pas transmis, et les blocs de luminance résiduelle de "0" à "15" sont transmis dans l'ordre indiqué (avec leurs coefficients DC). Les blocs de chrominance résiduels sont envoyés avec la même méthodologie que dans le mode Intra16×16.

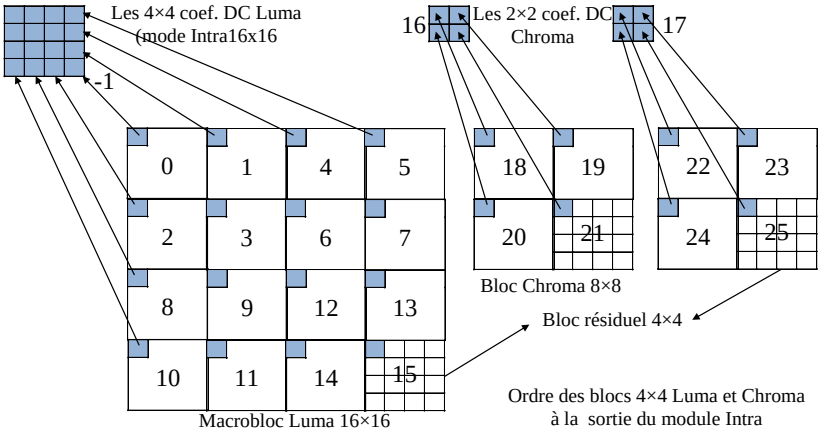



Figure 2.23 : Ordre de transmission des blocs à l'encodeur entropique dans le mode Intra.

En plus des blocs résiduels, le choix des modes de prédiction Intra, pour chaque bloc (16 modes Intra4×4) ou bien pour le macrobloc entier (un mode Intra16×16), doit être signalé au décodeur et cela pourrait nécessiter un grand nombre de bits. Toutefois, pour des blocs voisins, les modes intra sont fortement corrélés. En effet sur la Figure 2.24, pour chaque bloc courant $B_{k,l}$, l'encodeur (et par conséquent le décodeur) calcule un mode le plus probable, en fonction des blocs de voisinage, selon la méthode suivante : Si $B_{k-1,l}$ et $B_{k,l-1}$ sont à la fois codées en mode Intra4×4 et sont tous les deux dans la tranche en cours, le mode le plus probable est le minimum entre les deux modes; sinon le mode le plus probable est fixé à 2 (mode DC) [64].

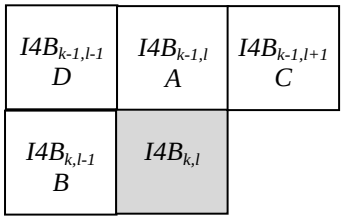



Figure 2.24 : Les blocs de voisinage du bloc $I4B_{k,l}$.

Pour chaque bloc résiduel transmis, l'encodeur envoie un drapeau. Si le drapeau est égal à "1", le mode le plus probable est utilisé, dans ce cas le mode de prédiction Intra est codé sur un seul bit (le drapeau). Si le drapeau est "0", un autre paramètre est envoyé pour indiquer un changement de mode. Ce paramètre est calculé de la manière suivante :

1. Si le mode Intra calculé pour le bloc courant est inférieur que le mode le plus probable, le paramètre transmis est égal au mode calculé ;
2. Sinon, le paramètre transmis est égal au mode calculé moins un.

De cette façon, seulement 8 valeurs de sélecteur des modes restants sont requis (0 à 7) pour signaler le mode Intra actuel (qui peut varier 0-8) [39]. Dans ce cas le mode du bloc courant est codé sur 4 bits (un bit pour le drapeau avec une valeur 0 et 3 bits pour le paramètre calculé).

### 3.7.2. Flux de données de la prédiction Inter

Coder un vecteur de mouvement pour chaque partition peut coûter un nombre important de bits, surtout si des petites tailles des partitions sont choisies. Les vecteurs de mouvement pour les partitions de voisinage sont souvent fortement corrélés de sorte que chaque vecteur de mouvement est prédit à partir des autres vecteurs de proximité. Un vecteur de mouvement prédit (MVP) est formé sur la base

des vecteurs de mouvement précédemment calculés. La différence entre le vecteur courant et le vecteur prédit (MVD) est codé et transmis. Le calcul de MVP dépend de la taille de la partition de compensation de mouvement et de la disponibilité des vecteurs à proximité.

Soit le macrobloc (ou bien sous-macrobloc) en cours de traitement (E), le macrobloc immédiatement à la gauche (A), le macrobloc immédiatement au-dessus (B) et le macrobloc au-dessus et à droite (C). S'il y a plus d'une partition immédiatement à gauche de E, la plus haute de ces partitions est choisie comme A. S'il ya plus d'une partition immédiatement au-dessus de E, le plus à gauche d'entre eux est choisi comme B. La Figure 2.25.a illustre le choix des partitions de voisinage lorsque toutes les partitions ont la même taille (16×16), et la Figure 2.25.b montre un exemple avec des partitions de prédiction lorsque les voisins ont des tailles différentes [51]. Le vecteur MVP est calculé de la manière suivante :

1. Pour toutes les partitions à l'exception des partitions 16×8 et 8×16, le vecteur MVP est la moyenne des vecteurs de mouvement pour des partitions A, B et C ;
2. Pour les partitions 16×8, le MVP de la partition supérieure 16x8 est prédite à partir de B et le MVP de la partition inférieur 16x8 est prédite à partir de A ;
3. Pour les partitions 8×16, le MVP de la partition à gauche 8×16 est prédite à partir de A et le MVP de la partition à droite 8×16 est prédite à partir de C ;
4. Pour les macroblocs en mode Skipped, le MVP de la partition 16×16 est généré comme dans le cas 1.

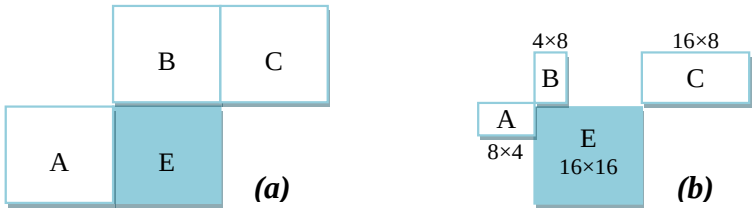



Figure 2.25 : Macroblocs en cours et de voisinage avec des tailles différentes.

Si l'un ou plusieurs des macroblocs de voisinage n'est pas disponible (les limites de l'image par exemple), le choix de MVP est modifié. Au niveau du décodeur, le MVP vecteur prédit est formé de la même manière et ajouté au vecteur de différence (MVD) décodé. Dans le cas d'un macrobloc en mode skipped, il n'y a pas de vecteur décodé, et un macrobloc compensé en mouvement est produit en utilisant le MVP comme vecteur de mouvement.

### 4. Performances et applications

Bien que H.264/AVC soit toujours fortement basé sur une architecture de codage hybride et prédictif, certains outils permettent d'améliorer l'efficacité de codage : la compensation en mouvement sur des blocs de taille variable avec une précision au quart de pixel et des images de références multiples, l'amélioration des modes "skipped" et "direct", une prédiction spatiale directionnelle pour le codage Intra, un filtre réducteur d'effets de blocs dans la boucle, etc. Ce codeur a été amélioré aussi, en incluant par exemple, une taille de transformée 4×4 basée sur les entiers, un filtrage hautement adaptatif, un codage hiérarchique des blocs ou le codage arithmétique adaptatif contextuel (CABAC, CAVLC) [29].

En plus H.264/AVC comprend de nombreuses autres nouvelles techniques qui lui permettent de compresser beaucoup plus efficacement les vidéos que les autres normes et fournit plus de flexibilité aux applications dans un grand nombre d'environnements réseau. Ces fonctionnalités principales comprennent [57] :

1. Les tranches de commutation (appelées SP et SI) permettent à un codeur de diriger un décodeur pour que ce dernier puisse s'insérer dans un flux vidéo entrant, ceci permet du streaming vidéo à débit variable et un fonctionnement en « trick mode » (mode truqué).
2. L'ordonnancement flexible des macroblocs (FMO : Flexible macroblock ordering, alias slice groups) et l'ordonnancement arbitraire des tranches (ASO : Arbitrary slice ordering) sont des techniques de restructuration de l'ordonnancement des régions fondamentales de l'image (macroblocs). Typiquement utilisées pour améliorer la résistance aux erreurs et aux pertes, ces techniques peuvent également être utilisées à d'autres fins.
3. Les tranches redondantes (RS : Redundant slices) permettent d'améliorer la résistance aux erreurs et aux pertes en permettant au codeur de transmettre une version additionnelle de tout ou partie de l'image dans une qualité moindre.
4. Des informations supplémentaires d'amélioration (SEI : Supplemental enhancement information) et des informations d'état qualitatif de la vidéo (VUI : Video usability information) sont des informations supplémentaires qui peuvent être insérées dans le flux pour améliorer son usage pour un grand nombre d'applications.
5. La numérotation des images permet la création de sous-séquences ainsi que la détection et la dissimulation de la perte d'images entières.
6. Le comptage de l'ordre des images permet de conserver l'ordre des images et du son dans des images décodées isolément des informations de minutage (ce qui permet à un système de transporter, contrôler et/ou changer l'information de minutage sans affecter le contenu des images), etc.…

Ces techniques aident H.264 à dépasser significativement les standards précédents, dans une grande variété de circonstances et dans une grande variété d'environnements d'application. H.264 peut fonctionner souvent nettement mieux que la vidéo MPEG-2 en obtenant la même qualité avec un débit binaire diminué de moitié, voire plus.

### 4.1. Applications de H.264

Le format H.264 offre, en plus de son efficacité de compression, une grande flexibilité en termes d'options de compression et de support de transmission. Un encodeur H.264 peut utiliser une grande variété de méthodes de compression, s'adaptant ainsi à des applications présentant des contraintes spécifiques comme de la transmission mobile temps réel à bas débit, de la télévision haute-définition ou encore de la production vidéo professionnelle. Les problèmes du stockage et de la transmission ont été intégrés dans la démarche de conception du standard, débouchant sur un format compressé paquétisé et des options permettant de réduire les effets des erreurs de transmission, comme des codes correcteurs d'erreurs [4]. H.264 a été adopté pour de nombreuses applications, parmi lesquelles [34] :

- Les DVD haute-définition (HD-DVD et Blu-Ray) ;
- La télévision haute-définition en Europe ;

- Les produits vidéo de la marque Apple (les téléchargements vidéo sur la plate-forme iTunes, les vidéos sur iPod et MacOS) ;
- Les applications vidéo de l'OTAN et du Département de la Défense américain ;
- La télévision mobile ;
- Les contenus vidéo en streaming sur Internet ;
- La vidéoconférence, etc.…

Le plus grand avantage du format H.264 sur les formats antérieurs est sans doute son efficacité de compression. En comparaison des standards MPEG-2 et MPEG-4 Visual, H.264 fournit en effet une meilleure qualité d'image à débit binaire égal ou un débit plus faible à qualité équivalente. Par exemple, un DVD simple-couche peut stocker un film d'environ deux heures au format MPEG-2 ; avec le format H.264, on pourrait stocker plus de quatre heures de vidéo en qualité DVD sur le même disque. Evidemment, ce gain en taille mémoire a une contrepartie : la complexité, multipliée par 10 (voir 20) par rapport aux standards précédents.

### 4.2. Produits et mises en œuvre de H.264/AVC

Comme de nombreuses autres normes vidéo du groupe ISO/CEI MPEG, le H.264/AVC dispose d'une application logicielle de référence, qui peut être gratuitement téléchargée. Le principal objectif de cette application est de donner des exemples des différentes possibilités du H.264/AVC, plutôt que de fournir un produit réellement utilisable et performant. Des applications matérielles de référence sont aussi en cours de normalisation. La figure suivante montre les transpositions logicielles de l'encodeur H.264/AVC, et ceci sur la base du logiciel de référence JM [47].

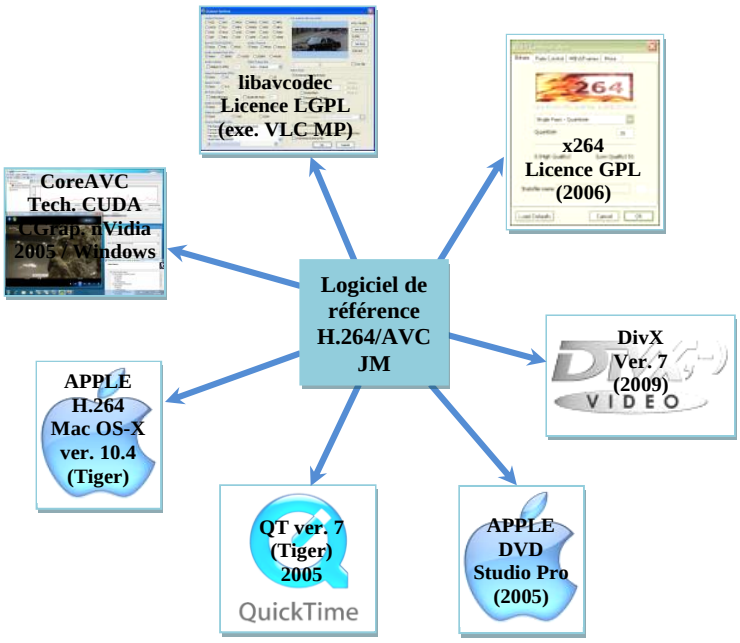



Figure 2.26 : Les transpositions logicielles de H.264/AVC.

Pour les applications matérielles, plusieurs entreprises produisent des puces capables de décoder des vidéos H.264/AVC. Les puces capables de décoder en temps réel des vidéos hautes définitions sont entre autres les suivantes : Broadcom (BCM7411), Conexant (CX2418X), Sigma Designs (SMP8634, SMP8635, EM8622L, et EM8624L), STMicroelectronics STB7100, STB7109, NOMADIK (série STn 8800/8810/8815), WISchip (Micronas USA, Inc.) DeCypher 8100, Neotion NP4 - NEOTION

Processor 4. Ce type de puces permet un large déploiement de matériel à bas coût capable de jouer des vidéos H.264/AVC aux définitions de télévision standard et hautes. De nombreux matériels (décodeurs) sont d'ores et déjà disponibles depuis 2005, cela va des produits de grande consommation peu onéreux à des codeurs temps réel à base de FPGA pour la diffusion. La figure suivante montre quelques exemples des décodeurs H.264 implanté sur des applications temps réel.


Figure 2.27 : Les transpositions matérielles de H.264/AVC.

### 4.3. Statistiques pour la norme H.264

A la fin de ce chapitre, il semble nécessaire d'étudier la répartition du temps CPU suivant les différents modules de traitements du codec H.264/AVC. Cette étude doit nous permettre, par la suite dans la deuxième partie de cette thèse, de prendre décisions pour d'implémentation matérielle et/ou logicielle de l'encodeur. Plusieurs travaux, surtout pour le décodeur, mentionnent une telle répartition, mais il n y a pas une étude de référence pour H.264. Les estimations de temps de calcul de chaque module dépendent de plusieurs paramètres : le type de CPU utilisé, le profil et la version du codec ainsi que les modules prise ne compte dans la même version, la stratégie d'accès à la mémoire, la séquence vidéo manipulée, etc. La Figure 2.27, par exemple, montre 2 répartitions différentes du temps de calcul dans le décodeur H.264 : la première (a) représente une distribution des temps d'exécution sur une architecture SIMD optimisée [65], la deuxième (b) représente une répartition du temps d'exécution du décodeur de référence sur un Pintium4 (2,4 GHz, 512 Mo de RAM, 8Ko de cache, etc.) [66].

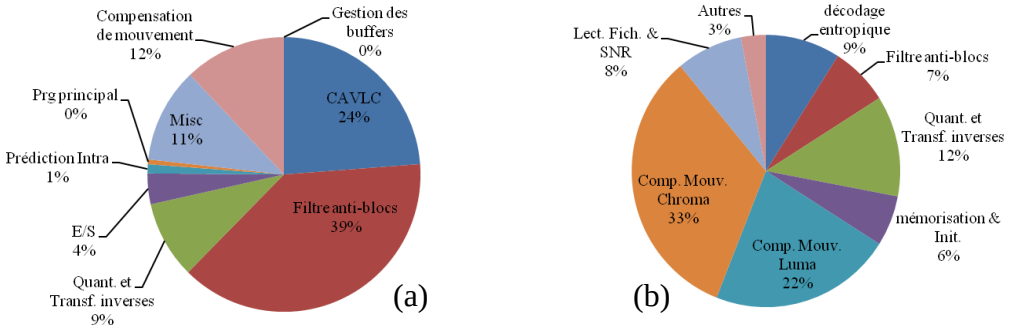



Figure 2.28 : Répartition du temps d'exécution du décodeur H.264/AVC.

La Figure 2.28.a montre une analyse de temps d'exécution de l'encodeur de référence H.264 (JM9.5). Dans cette étude, une séquence vidéo FOREMAN (300 images au format CIF, IPPPPP) est utilisé sur un processeur Pentium4 (3.0GHz, 512 Mo de RAM) sous Microsoft Windows XP [67]. De même, la Figure 2.28.b montre une autre répartition du temps d'exécution d'un encodeur H.264 optimisé et implémenté sur un DSP TI-DM642 [68].

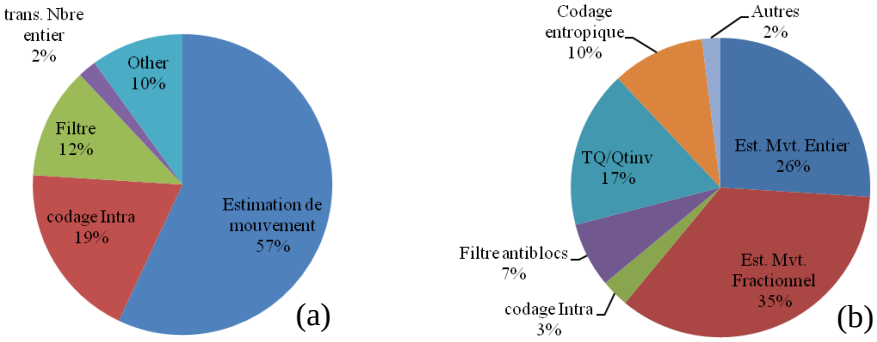



Figure 2.29 : Répartition du temps d'exécution de l'encodeur H.264/AVC.

### 4.4. H.264/AVC et après ?

Depuis la fin de la normalisation de la norme H.264/AVC en 2005, le groupe JVT s'est focalisé sur la normalisation de l'annexe scalable nommée H.264/SVC [69] et l'annexe multi-vue nommée H.264/MVC [70]. Cependant les activités classiques d'amélioration du codage vidéo ont continué durant cette période. Afin d'encourager de nouvelles contributions, il a été décidé au 26eme meeting VCEG la création d'un logiciel, nommé KTA (Key Technical Area) [71]. Ce logiciel regroupe l'ensemble des outils proposés depuis la fin de la normalisation de H.264/AVC et qui apportent des gains jugés significatifs par le groupe VCEG. Le jugement de l'efficacité d'une méthode est un compromis entre l'efficacité de compression, la complexité de calcul et de mémoire mais aussi de son impact sur la résistance aux erreurs. Le JM KTA est basé sur le logiciel de référence de la norme : le JM11.0 [47].

Dans JM KTA plusieurs propositions et outils ont été validés pour les différents modules de l'encodeur H.264. Tous ces outils impliquent une modification du décodeur. Ces outils seraient donc normatifs s'ils étaient intégrés dans une future norme. Cependant le KTA peut intégrer des outils non normatifs comme l'outil Rate Distortion Optimized Quantization (RDOQ) qui optimise la sélection du QP au niveau macrobloc et son extension à la quantification de la chrominance [50].

## 5. Conclusion

Après cette étude détaillée de la norme de compression vidéo H.264/AVC, il est facile d'identifier les modules qui prennent le plus du temps de calculs, et cela dans le but de prendre décision sur l'implémentation matérielle et/ou logicielle de ces modules. En effet, pour la réalisation de systèmes temps réel, traitant des images de plus grande résolution à des fréquences plus élevées, la principale question qui peut se poser est la répartition matérielle et logicielle des traitements : blocs qui doivent être totalement ou partiellement réalisés sous forme matérielle (IP matériel) et blocs susceptibles d'être implantés en logiciels. Le choix du type d'implémentation dépond des outils et des matériels utilisés.

Cette étude nous permet aussi d'extraire des informations de parallélisme possible dans la norme. Ces informations seront utiles lors d'une implémentation matérielle, pour savoir les modules qui peuvent être exécutés en parallèle et les tâches qui peuvent être réalisées en pipeline. Dans ce contexte, nous avons remarqué une grande dépendance de traitement des données, surtout dans le module de prédiction Intra et le filtre anti-blocs. En effet, dans plusieurs cas, le traitement d'un bloc de données et conditionné par la fin du traitement d'un autres bloc ou bien la fin d'un autre traitement sur le même bloc. Cette dépendance de données serra un grand inconvénient pour l'implémentation matérielle des différents modules de l'encodeur H.264.

En plus de dépendance de données, cette étude nous a permet aussi de suivre le transfert des données traitées entre les différentes modules. Cela nous permet par la suite de définir une stratégie de gestion de mémoire interne et externe dans un circuit FPGA par exemple. Nous allons aussi utiliser tous les détails pour l'implémentation matérielle (niveau RTL) des différents modules, et cela avant l'implémentation de ces modules dans un schéma à base de microprocesseur MicroBlaze.

# Chapitre 3

# Les Outils de Conception des Systèmes de Traitement d'Images Temps Réel

*« La correction du système ne dépend pas seulement des résultats logiques des traitements, mais dépend en plus de la date à laquelle ces résultats sont produits »*
*(John A. Stankovic, 1988).*

## 1. Introduction

Les systèmes informatiques temps réel se différencient des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude des résultats. Autrement dit le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés. Le développement de ces systèmes à l'aide de FPGA nécessite l'emploi des outils spécialisés. Un outil de développement comporte, en général, un ou bien plusieurs logiciels de conception, une carte d'émulation pour la vérification du fonctionnement électrique de l'architecture conçue, et une station de programmation et de commande (PC). Le logiciel sert à aider le concepteur à développer la méthode de conception de l'architecture. Généralement, une méthode de conception comporte quatre phases [8] : une phase de saisie de l'architecture, une phase d'implantation, une phase de vérification et une phase de tests électriques.

De nos jours, avec l'apparition de plus en plus de systèmes temps réel plus performants, la gestion de la complexité avec les outils d'aide à la conception traditionnels (masque schématique par exemple) devient une tâche pénible, coûteuse, voire impossible, surtout quand on considère les contraintes de mise en marché d'un produit. Il était donc nécessaire de trouver de nouvelles méthodes de conception plus évolutives qui permettront l'implémentation de larges et complexes systèmes. Généralement ces méthodes de conception sont commercialisées sous formes d'outils de conception des systèmes qui utilisent des langages de programmation de différents niveaux d'abstraction. Le but ultime de ces outils associés, avec un langage de conception et de simulation puissant, est de générer le dessin des masques d'un circuit à partir de sa description comportementale haut niveau. Ainsi, le concepteur peut se limiter à la conception, à la modélisation et à la simulation de son produit, sans tenir compte des détails de mise en œuvre aux niveaux schématique et physique [9].

Dans un premier temps dans ce chapitre, nous présentons la notion du temps réel avec plusieurs définitions et applications possibles. Dans un deuxième temps, nous présentons les langages de description du matériel et/ou du logiciel, ces langages sont classés selon plusieurs niveaux d'abstraction. Enfin, nous présentons les outils de conception des systèmes temps réels, spécialement les outils de Xilinx, nous justifions ainsi nos choix de langage et d'outils de synthèse de simulation et d'implémentation d'un système de traitement vidéo temps réel (H.264/AVC).

## 2. Notion du temps réel

### 2.1. Définitions

Le temps réel est un concept un peu vague qui donne la possibilité de plusieurs définitions différentes [72]. Généralement, ce terme exprime un mode de fonctionnement dans le processus d'acquisition et de traitement de données. Une première définition, plus générale, consiste qu'un système est dit temps réel lorsque l'information après acquisition et traitement reste encore pertinente. Plus précisément, cela veut dire que dans le cas d'une information arrivant de façon périodique, les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information. Un temps maximum d'exécution est garanti et non pas un temps moyen. Pour cela, il faut que le noyau ou le système temps réel soit déterministe et préemptif. Nous pouvons dire que le temps réel représente la disponibilité immédiate (à temps) d'informations pour l'utilisateur.

Une autre définition, spécialement pour les systèmes numériques, considère qu'un système temps réel est une association logiciel/matériel où le logiciel permet, entre autres, une gestion adéquate des ressources matérielles en vue de remplir certaines tâches ou fonctions dans des limites temporelles bien précises.

#### 2.1.1. Le temps réel et la vitesse des processeurs

D'ordre général, un système temps réel n'est pas forcément plus rapide qu'un système à temps partagé[1]. Il devra par contre satisfaire à des contraintes temporelles strictes, prévues à l'avance et imposées par le processus extérieur à contrôler. Une confusion classique est de mélanger temps réel et rapidité de calcul du système, donc puissance du processeur (microprocesseur, microcontrôleur et DSP). On entend souvent : "être temps réel, c'est avoir beaucoup de puissance de calcul (être plus rapide)", ce n'est pas toujours vrai. En fait, être temps réel dans une application, c'est être capable d'acquitter l'interruption périodique (moyennant un temps de latence d'acquittement d'interruption imposé par le matériel), traiter l'information et la signaler au niveau de la sortie prévue de l'application, et cela dans un temps inférieur au temps entre deux interruptions périodiques consécutives [72].

On est donc lié à la contrainte durée entre deux interruptions générées par le processus extérieur à contrôler. Si cette durée est de l'ordre de la seconde (comme dans le cas du contrôle d'une réaction chimique), il ne sert à rien d'avoir un système à base de processeur puissant comme un PentiumIV, un simple processeur 8 bits ou Microchip PIC ou même un processeur 4 bits fera amplement l'affaire, ce qui permettra de minimiser les coûts sur des forts volumes de production. Si ce temps est maintenant de quelques microsecondes (comme dans le cas d'une application de traitement d'image), il convient de choisir un processeur nettement plus performant (un processeur 32 bits par exemple).

#### 2.1.2. Les deux types des systèmes temps réel

Il est évident que la structure d'un système temps réel dépendra des contraintes différentes. On distingue le temps réel strict ou dur (hard real-time) et le temps réel souple ou mou (soft real-time) suivant l'importance accordée aux contraintes temporelles [73].

[1] Le temps partagé consiste à partager le temps du processeur entre les différentes tâches.

Les systèmes dits à contraintes souples acceptent des variations dans le traitement des données de l'ordre de la demi-seconde ou la seconde. On peut citer l'exemple des systèmes multimédia : si quelques images ne sont pas affichées, cela ne met pas en péril le fonctionnement correct de l'ensemble du système. Ces systèmes se rapprochent fortement des systèmes d'exploitation classiques à temps partagé. Ils garantissent un temps moyen d'exécution pour chaque tâche, avec une répartition égalitaire du temps CPU entre processus.

Les systèmes dits à contraintes strictes sont des systèmes pour lesquels une gestion stricte du temps est nécessaire pour conserver l'intégrité du service rendu. On citera comme exemples les contrôles de processus industriels sensibles comme la régulation des centrales nucléaires ou les systèmes embarqués utilisés dans l'aéronautique. Ces systèmes garantissent un temps maximum d'exécution pour chaque tâche, avec une répartition totalitaire du temps CPU entre tâches.

On peut ainsi considérer qu'un système temps réel strict doit respecter des limites temporelles données même dans la pire des situations d'exécution possibles. En revanche un système temps réel souple doit respecter ses limites pour une moyenne de ses exécutions. On tolère un dépassement exceptionnel, qui sera peut-être rattrapé à l'exécution suivante. Les systèmes à contraintes dures doivent répondre à trois critères fondamentaux : le déterminisme logique, le déterminisme temporel et la fiabilité.

Dans les deux types de système temps réel, et pour tout système de n tâches, la condition dans l'équation 3.1 est nécessaire mais pas suffisante à sa faisabilité. Dans cette équation, $C_i$ représente le temps de calcul de la tâche n° i et $T_i$ représente sa période. Une valeur supérieure à 1 signifierait que le système nécessite plus de temps d'exécution que le processeur ne peut en fournir [72].

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \qquad 3.1$$

### 2.1.3. Systèmes embarqués temps réel

On entend souvent parler de temps réel dès que l'on parle de système embarqué. En effet, un système embarqué doit généralement respecter des contraintes temporelles dures. Dans ce sens, et plus des autres définitions données dans le chapitre1, un système embarqué peut être défini comme un système électronique et informatique autonome et temps réel, qui est dédié à une tâche bien précise. Il doit généralement respecter des contraintes temporelles dures et on y trouve enfoui un système d'exploitation ou un noyau temps réel (Real Time Operating System, RTOS) [74].

A l'heure actuelle, les systèmes embarqués tentent d'exploiter au mieux les progrès de l'électronique et de l'informatique. On développe ainsi des systèmes de plus en plus performants et exigeants en termes de puissance de calcul embarquée, de débits de données à gérer, de capacité de stockage et de miniaturisation. Les matériels utilisés sont très divers, il n'est plus rare de voir des caméras vidéo sur des systèmes embarqués, lesquelles seront souvent associées à d'autres capteurs complémentaires. On aboutit de plus en plus à des systèmes capables d'exécuter des algorithmes de fusion de données.

## 2.2. Exemples d'applications des systèmes temps réel

Le temps réel est indispensable pour les applications dynamiques qui ne peuvent être satisfaites en temps différé. Pourtant, un système temps réel est un système pour lequel le respect des contraintes

temporelles dans l'exécution des traitements est aussi important que le résultat de ces traitements. Cela signifie souvent que l'application est suffisamment performante pour traiter à la volée et sans perte d'information toutes les données provenant d'un ou de plusieurs capteurs, sans atteindre la capacité maximale de calcul qu'elle a à sa disposition.

Les applications temps réel embarquées peuvent être des systèmes installés sur des plateformes mobiles (systèmes d'instrumentations dans les avions, automobiles, plateformes robotiques mobiles, etc.) et fournissant ainsi certains services tels que la présentation d'informations sur l'environnement du véhicule (détection d'obstacles, cartographie, etc.), le contrôle de tout ou partie de ce véhicule (réactions en cas de danger, pilotes automatiques, etc.) ou encore le relais pour l'échange de données (satellites de communications) [72]. Le temps réel est présent aussi dans l'industrie de production (contrôle de machine et de procédés), les marchés boursiers (transactions, cotations). En plus, le temps réel est une réalité centrale dans les communications (TV numérique, chat, webcams, vidéoconférences), dans les installations (interfaces, captation de mouvement, capteurs, reconnaissance vocale, détections et mesures) dans les jeux vidéos (game-play, interfaces, captation 3D, etc.), dans les instruments de musiques (synthétiseurs, systèmes midi), etc.

Il y a fort à parier que la décennie qui va suivre verra ce type d'applications se multiplier dans notre environnement, notamment dans le domaine de l'assistance à la conduite ou encore au niveau des systèmes de sécurité. Ces applications seront de plus en plus complexes, faisant intervenir des capteurs variés et exigeants. Elles devront notamment savoir communiquer, d'une part avec des stations fixes de l'infrastructure qui les entoure, et d'autre part entre elles.

### 2.3. Temps réel pour la compression vidéo

Le traitement des images dynamiques en temps réel nécessite l'utilisation des circuits électroniques rapides, capables de manipuler de grandes quantités d'informations générées par la source vidéo. Citons les applications d'enregistrement de vidéo sur PC (à partir d'une carte Tuner/TV) avec compression logicielle. Dans ce cas et comme nous avons dit auparavant, pour assurer le temps réel, on est lié à la contrainte durée entre deux interruptions (entre deux images successives). Il convient donc avant de concevoir un tel système de connaître la durée minimale entre 2 interruptions. Pour l'exemple de la compression vidéo, avec une séquence d'image arrivant à des instants biens définis selon le format et la fréquence vidéo, le temps réel pour la compression vidéo se mesure alors avec deux notions clés [2] :

- La latence qui représente le temps nécessaire à une image pour traverser tous les éléments de l'encodeur.
- La cadence où on considère qu'un encodeur vidéo est "temps réel" lorsqu'il est capable de tenir la fréquence d'acquisition des images.

Les premières applications industrielles du traitement d'images en temps réel sont apparues dans les années 1980. Durant cette période, les systèmes de vision industriels, basés principalement sur des machines spécialisées, ont été utilisés plutôt à usage expérimental et c'est seulement à la fin des années 1990 que ces systèmes de vision arrivent à maturité et qu'ils sont réellement utilisés comme moyens de contrôle et de mesure dans les chaînes de production industrielle. Cette maturité a été atteinte grâce à l'évolution de la technologie qui a été en grande partie marquée par l'arrivée massive sur le marché

industriel d'une part des caméras CCD de bonne résolution et d'autre part des micro-ordinateurs fonctionnant à des fréquences supérieures à 100 MHz [7]. Ces caméras permettent ainsi de garantir de bonnes conditions d'acquisition d'images et les micro-ordinateurs rapides et performants peuvent exécuter des traitements d'images de plus en plus complexes et cela en un temps de calcul compatible avec les contraintes industrielles.

Dans un exemple complet de traitement d'image en temps réel, la chaîne doit contenir tout le cycle d'acquisition, de traitement et de restitution du signal vidéo. Donc le temps réel pour nous est le fait d'avoir le même flux d'images entre la source vidéo en entrée et le moniteur de restitution en sortie [8]. Dans un exemple simple, où on considère que le traitement est le même pour toutes les images d'une séquence à une fréquence de 25 images par seconde, le temps entre deux interruptions dans ce cas est égale à 1/25 secondes (c'est le temps maximal pour le traitement d'une image entière). Ce n'est pas le cas dans les standards de compression actuels, avec une très grande dépendance de traitement entre les différentes images de la même séquence. L'exemple type est la norme H.264/AVC, où la séquence vidéo est traitée par groupe d'image (GOP).

Le tableau suivant montre un exemple du temps nécessaire pour le traitement d'images de différents formats (ce temps est calculé aussi pour chaque macrobloc). Le nombre de cycles d'horloge, nécessaire pour le traitement d'un macrobloc, est calculé à la base d'une fréquence de 100Mz (fréquence de fonctionnement de la plateforme Virtex5-ML501 de Xilinx).

| | **QCIF** | **CIF** | **SD (NTSC)** | **SD (PAL, SECAM)** | **SD (DVD)** | **HD TV (720p)** | **HD DVD (1080i)** |
|---|---|---|---|---|---|---|---|
| Nombre de pixels/image | 176×144 | 352×288 | 640×480 | 768×576 | 720×576 | 1280×720 | 1920×1080 |
| Fréquence image (Hz) | 10 | 10 | 60 | 50 | 25 | 25 | 50 |
| Débit (Y) | 247,5 Ko | 990 Ko | 8,79 Mo | 10,54 Mo | 9,88 Mo | 22 Mo | 49,44 Mo |
| Débit (YCrCb/4:2:2) | 371,25 Ko | 1,485 Mo | 13,18 Mo | 15,80 Mo | 14,80 Mo | 33 Mo | 74 Mo |
| Nombre de MB/image | 11×9 | 22×18 | 40×30 | 48×36 | 45×36 | 80×45 | 120×67,5 |
| Tem. trait. d'une image | 3/10 s | 3/10s | 1/10 | 3/25 | 3/25 | 3/25 | 3/25 |
| Temps de traitement/MB | 3,03 ms | 3,03 ms | 83,33 µS | 69,44 µS | 74,074 µS | 33,33 µS | 14,814 µS |
| Nombre de cycles/MB | $303.10^6$ | $303.10^6$ | $8,333.10^3$ | $6,944.10^3$ | $7,407.10^3$ | $3,333.10^3$ | $1,481.10^3$ |

Tableau 3.1 : Exemples du temps nécessaire pour le traitement d'une image.

### 2.4. Contraintes de conception des systèmes embarqués temps réel

Dans toute démarche de conception d'un système temps réel, le coût de l'étude ne doit pas dépasser un certain pourcentage du chiffre d'affaires. Les réalisations de systèmes spécialisés sous forme de circuits intégrés monolytiques sur un substrat (VLSI) nécessitent un coût d'étude important. Ces réalisations sont soit réservées à des fonctions standards destinées à être reproduites en très grande quantité et réservées à de très vastes marchés, soit à des fonctions pour lesquelles le coût ne compte pas mais qui nécessitent de très hautes performances. Cette démarche couvre un pourcentage relativement faible de l'activité de conception des systèmes électroniques.

En plus du coût de l'étude, le temps de mise en marché est un important facteur pour la conception des systèmes. En effet, ce qui est nouveau aujourd'hui devient ancien demain. Pour cela, et parmi les méthodes de marketing des sociétés est de parler des générations du futur. Nous sommes dans les $3^{eme}$ et $4^{eme}$ générations de portables téléphoniques, et nous parlons déjà de la $5^{eme}$ génération. En plus de ces deux contraintes (coût de l'étude et temps de mise en marché) pour la conception des systèmes embarqués temps réel, d'autres contraintes, d'ordre technologique, sont aussi imposés.

### 2.4.1. Contraintes liées à l'architecture système

Aujourd'hui, la maîtrise des technologies microélectroniques permet d'implanter des centaines de millions de transistors sur un même substrat. Il en résulte que la complexité des circuits peut être telle que pour créer une architecture logique et/ou arithmétique, il est nécessaire d'utiliser une méthode de conception à l'aide de systèmes de développement sur ordinateur.

La majorité des circuits programmables, aujourd'hui, ne souffre pas de point de vue ressources disponibles. Les circuits disposent de plus en plus de la logique programmable, de mémoires de différents types, des convertisseurs, des DSP et même des processeurs intégrés. La question à nos jours, est comment gérer ces ressources, surtout avec des applications qui ne cessent pas de se compliquer. En plus dans un système embarqué, il faut aussi penser à plusieurs autres paramètres. Nous pouvons citer entre autres : la consommation d'énergie, l'encombrement, la température de fonctionnement et leur refroidissement, le poids, la résistance aux chocs et aux vibrations, etc.

Une autre contrainte, aussi d'ordre technologique, c'est la possibilité de reprogrammation de circuit. Cette caractéristique est assurée par les circuits FPGA et non pas par les ASIC. Si on exige cette contrainte, par exemple dans le domaine de la reconfiguration dynamique, on est obligé d'utiliser un type de circuit et non pas un autre.

### 2.4.2. Contraintes liées à l'architecture mémoire

L'architecture mémoire, dans les systèmes embarqués temps réel, joue un rôle de plus en plus important et retient toute l'attention des concepteurs d'aujourd'hui, surtout pour les applications de traitement d'images et de multimédia. Ces applications manipulent des volumes importants de données qui sont parfois fortement dépendantes et nécessitent l'intégration d'une mémoire globale partagée. L'architecture mémoire pour de telles applications devienne assez complexe et occupe jusqu'à 70% de la surface du système et une grande partie du temps de conception. Ainsi, elle influe d'une façon importante sur les performances de l'architecture. En effet, en ciblant une architecture mémoire distribuée et/ou partagée, le nombre d'architectures possibles croit d'une façon exponentielle par rapport au nombre de processeurs, et ces différents choix peuvent engendrer des coûts de conception très différents. Ainsi, il est impératif au concepteur de faire le choix optimal.

## 2.5. Les solutions technologiques pour assurer le temps réel

Pour les applications haut-débit nécessitant des traitements itératifs, généralement les processeurs (la solution soft) deviennent incapables d'assurer un temps réel. D'autres solutions matérielles sont nécessaires, à savoir les DSPs et les plateformes à base de FPGA. En effet, parmi les solutions proposées pour assurer un temps réel pour les systèmes embarqués temps réel, c'est l'intégration de ces systèmes dans un SoPC, qui peut contenir un ou bien plusieurs processeurs, des DSP, des accélérateurs matériels, etc. Le but ici est de partager les tâches sur plusieurs processeurs, et d'affecter les tâches qui demandent le maximum de temps aux processeurs parallèles spécialisés.

### 2.5.1. Choix du matériel

Les plateformes à base de DSP sont avantageuses en terme de coût, de reconfigurabilité et de temps de développement. Ils sont conçus pour effectuer des calculs en temps réel et intègrent donc de nombreux opérateurs. Ils permettent également un accès rapide aux données par des adressages particuliers. Ces

circuits ont aussi la possibilité de réaliser plusieurs instructions en parallèle et possèdent des opérateurs de calculs arithmétiques flottants très performants [8]. Les plateformes à base de DSP sont une solution pour étudier les applications à bande étroite (bande limitée par le débit binaire autorisé). Ils offrent un nombre restreint de traitement parallèle des données (nombre limité par le manque de parallélisme et la limitation du nombre d'opérations autorisées).

Une autre solution pour ce type de calcul, est l'utilisation des circuits ASIC, ces circuits intégrés spécifiques permettent d'implanter avec une très haute densité des algorithmes de traitement d'images. La conception de ces circuits fait appel à des outils de conception assistée par ordinateur (CAO) spécialisés dans le domaine de la microélectronique. L'avantage des circuits ASIC est de permettre l'intégration d'un nombre très important d'opérateurs de calculs et de mémoire tout en assurant une grande vitesse de calcul. Ils présentent, par contre, les inconvénients de temps et de coût de développement qui sont relativement importants.

Selon plusieurs travaux, l'utilisation des plateformes à base de FPGA semble le plus intéressant pour un calcul purement parallèle. En effet, a cause de sa reconfigurabilité, les circuits FPGA sont devenus les plus utilisés dans les applications temps réels, surtout avec l'avancement technologique actuel, où on peut trouver des plateformes mixtes autorisent l'intégration de plusieurs types d'IP. En plus des IPs (généralement limités en terme de parallélisme), il est possible de définir de nouveaux accélérateurs matériels spécialement conçus pour des tâches précises. Dans chaque accélérateur on a toute la liberté de conception (parallèle ou bien séquentielle), la seule limite est de ne pas dépasser la capacité de circuit FPGA.

### 2.5.2. Choix de la méthode de gestion de mémoire

Les mécanismes de gestion de mémoire et l'utilisation de caches mémoire engendrent des fluctuations dans le temps d'exécution des systèmes. En effet, dans un système de traitement parallèle, il ne suffit pas uniquement d'assurer le matériel de traitement (DSP/FPGA), il faut aussi préparer les données à partir de la mémoire à temps [75]. Pour la bonne gestion de mémoire, plusieurs paramètres doivent être pris en compte : le type de mémoire, la capacité de mémoire, le temps de latence, la cadence, etc.

Aujourd'hui, sur le même circuit FPGA (ou bien sur la même carte de prototypage), les constructeurs nous assure plusieurs types de mémoires. On trouve des mémoires complètement externes de type DDR2/DDR3, Mémoire flash ou bien des mémoires de type SDRAM sur la carte, ou bien encore des mémoires BRAM programmable dans le FPGA. Les constructeurs de FPGA, nous permettent aussi d'utiliser la logique programmable pour la création des BRAM, des registres et des FIFO. En plus, dans les circuits FPGA récents, de la famille virtex5 de Xilinx par exemple, les mémoires BRAM sont reconfigurables, et donnent la possibilité de choisir la taille des mots (bus de données de 8, 16, 32 ou 64bits), la direction des entrées/sorties et le type de la mémoire, etc. Dans ce cas, les mémoires sont données en Kbits, et c'est à l'utilisateur de les configurer selon les besoins des applications.

### 2.5.3. Les solutions technologiques dans les accélérateurs matériels

Au cours des années, les constructeurs de microprocesseurs, ont mis au point un certain nombre d'améliorations technologiques permettant d'optimiser leur fonctionnement. Ces techniques sont utilisées aussi pour la conception des accélérateurs matériels spécialement pour des applications temps

réel. Le but dans les 2 cas est d'effectuer le plus grand nombre d'opérations dans le plus petit temps possible.

### 2.5.3.1. Le parallélisme

En informatique, le parallélisme consiste à exécuter simultanément, sur des processeurs différents, des instructions relatives à un même programme. Cela se traduit par le découpage d'un programme en plusieurs processus traités en parallèle afin de gagner en temps d'exécution. Les architectures parallèles sont devenues le paradigme dominant pour tous les ordinateurs. En effet, depuis longtemps déjà, l'augmentation de la vitesse de calcul passait par une architecture comportant de nombreuses unités de calcul. La création de processeurs multi-cœurs, traitant plusieurs instructions en même temps au sein du même composant, résout ce dilemme pour les machines de bureau depuis le milieu des années 2000. Pour être efficace, les méthodes utilisées pour la programmation des différentes tâches qui constituent un programme sont spécifiques à ce mode de calcul, c'est-à-dire que les programmes doivent être réalisés avec cette optique. Ces méthodes ont initialement été développées sur des machines sophistiquées, des superordinateurs qui comptent de nombreux processeurs.

Le même concept est utilisé dans le matériel, où le parallélisme consiste à implémenter des architectures d'électroniques numériques et les algorithmes spécialisés pour celles-ci, permettant de traiter des informations de manière simultanées [72]. Ce type de technologie nécessite toutefois une synchronisation et une communication entre les différents processus. Par rapport à une architecture multiprocesseurs, l'idée majeur dans un accélérateurs est d'adaptée le circuit à la tâche réalisé. En effet, nous ne sommes pas besoins de 2 processeurs pour la réalisation de 2 additions en parallèle, il suffit de programmer 2 additionneurs sur un ASIC ou bien sur un FPGA.

### 2.5.3.2. Le pipeline

Le pipeline (ou pipelining) est une technologie visant à permettre une plus grande vitesse d'exécution des tâches (des instructions dans le cas du logiciel) en parallélisant des étapes. Inspiré aussi de la microarchitecture d'un microprocesseur, un pipeline est un élément d'un circuit électronique dans lequel les données avancent les unes derrière les autres, au rythme du signal d'horloge.

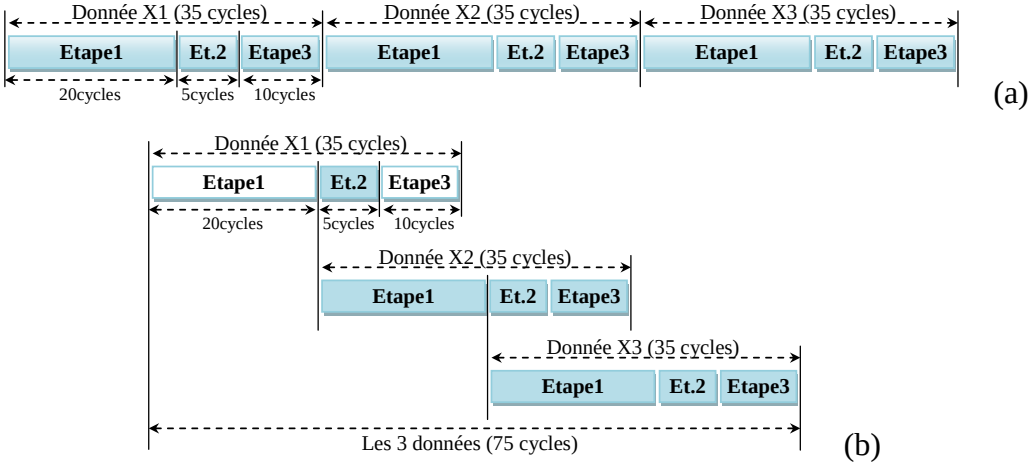



Figure 3.1. Principe de pipeline.

Le pipeline est un concept naturel s'inspirant du fonctionnement d'une ligne de montage industriel. Considérons un processus de trois étapes (1, 2 et 3), le traitement d'une donnée X1 s'effectue dans une

seule de ces étapes à la fois. Si les trois étapes prennent respectivement 20, 5 et 10 cycles d'horloge par exemple, alors le traitement de X1 prendra 35 cycles et le traitement de 3 données prendra 105 cycles (Figure 3.1.a). Si maintenant on adopte le principe de pipeline, le traitement des 3 données prendra 75 cycles d'horloge (Figure 3.1.b).

L'objectif du pipeline est d'être capable de réaliser chaque étape en parallèle avec les étapes en amont et en aval. Ceci est effectué par l'insertion des registres tampons (pipeline registers) entre les différentes étapes. Cette technique nécessite toutefois une synchronisation des données en entrées du pipeline.

Dans un système en pipeline, le temps global de traitement de 'N' données est égal à la somme du temps maximum entre les différents étapes multipliée par 'N' et la somme des temps des autres étapes. dans l'exemple de la figure 3.1, le temps nécessaires pour le traitement de 3 données est égal à (3×20)+(5+10) = 75 cycles d'horloge. Il est possible aussi de combiner les 2 techniques de parallélisme et de pipelining pour avoir le maximum d'efficacité. Pour programmer toutes ces solutions sur des circuits programmables, plusieurs langages et outils ont été proposé.

## 3. Conception des systèmes temps réel

L'apparition des circuits logiques programmables de type PLD, CPLD ou FPGA a permis de ne pas se limiter aux fonctions proposées par les fabricants des circuits intégrés logiques. En effet, l'utilisateur peut créer toutes les fonctions logiques qu'il souhaite avec comme limitation, la place disponible dans le circuit choisi et/ou la vitesse de fonctionnement de celui-ci. En effet, la disponibilité de composants de plus en plus denses, notamment ceux des deux constructeur Xilinx et Altera, permettent l'intégration de fonctions de plus en plus complexes. Si la saisie de schéma était le mode de description usuel des applications simples, les descriptions textuelles (HDL) ont depuis pris le pas dans les applications complexes et denses.

De même, les outils de développement mis à la disposition des utilisateurs par les fabricants de ces circuits doivent donc permettre de passer de la description du comportement d'une fonction logique à son câblage dans le circuit et cela de la manière la plus simple possible. La plupart du temps, la description du comportement des fonctions logiques est faite par l'utilisation de langage dit de "description comportementale".

### 3.1. Evolution des méthodes de conception

Les progrès continus des technologies CMOS et la multiplication des systèmes électroniques grand public ont conduit au développement de systèmes complets dans une seule puce de silicium. On trouve de plus en plus ces puces dans des applications diverses et variées utilisées dans la vie courante. Elles équipent les nouvelles générations de téléphones mobiles, assurent des fonctions critiques dans les voitures (freinage ABS, gestion des airbags, etc.), constituent le cœur des consoles de jeux (PlayStation 3 de Sony, par exemple) et elles sont utilisées dans la plupart des appareils multimédia comme les lecteurs/encodeurs vidéo portables, les appareils photo numériques, les assistants personnels (PDA), etc.

L'augmentation de la capacité d'intégration rend les systèmes électroniques de plus en plus complexes. Cette capacité d'intégration permet de concentrer des systèmes entiers sur une seule puce ce qui rend

la conception et la vérification de ces systèmes fastidieuses et coûteuses. Par contre, les moyens et les outils de conception ne connaissent pas la même évolution comme le montre la figure 3.2. La différence entre la capacité de conception et la capacité d'intégration ne cesse d'augmenter. Ces faits poussent les concepteurs à chercher des méthodes qui leurs permettent de concevoir et valider leurs systèmes dans des délais toujours plus courts (respecter le Time To Market : TTM).

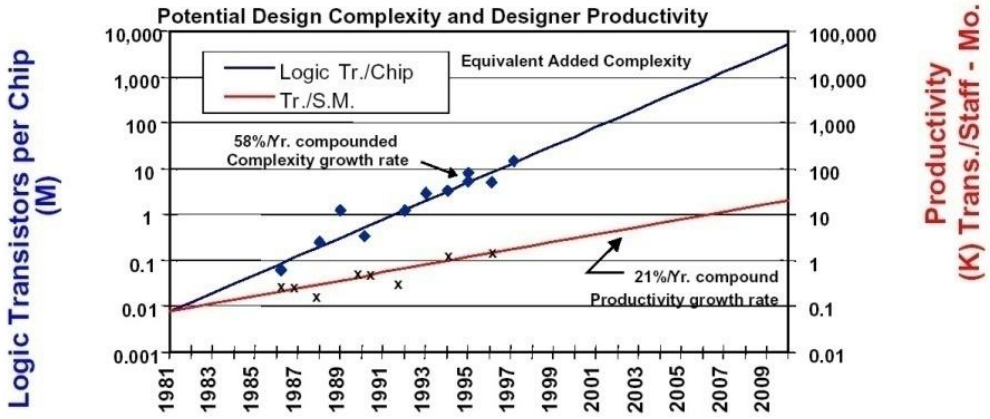



Figure 3.2. Divergence entre la productivité de la conception et la complexité des circuits [11].

La figure 3.3 présente les évolutions exponentielles que l'on peut constater depuis le début des années 80. La performance des processeurs a approximativement doublé tous les 18 mois, en s'appuyant notamment sur le doublement tous les 18 mois du nombre de transistors par unité de surface (loi de Moore). En fait, c'est une combinaison de l'augmentation des fréquences de fonctionnement d'une part, et des progrès au niveau de la microarchitecture des processeurs résultant de l'augmentation du nombre de transistors qui a permis ce gain de performance de 60% par an. Cependant, cette figure montre que les besoins des applications, exprimées par la complexité algorithmique ou loi de Shannon, croît encore plus vite que les performances. Les besoins des générations successives de téléphones cellulaires (1G, 2G, 3G) indiqués sur la figure illustrent cette situation. Une autre exponentielle est aussi très significative : le différentiel croissant entre l'évolution des performances et l'évolution de la durée de vie des batteries fait de la performance en fonction de l'énergie consommée un facteur décisif.

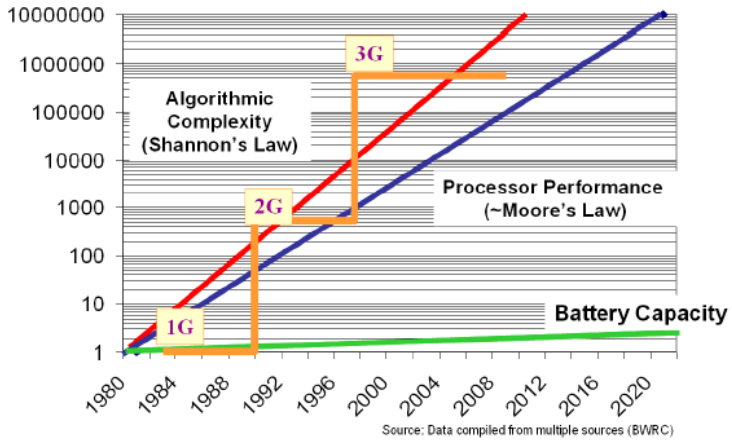



Figure 3.3. Des exponentielles significatives [23].

L'évolution exponentielle des performances ne s'est pas accompagnée d'une évolution régulière des méthodes de conception pour concevoir les circuits et les systèmes électroniques. Au contraire, elle s'est traduite par l'apparition régulière de "nouvelles" méthodes de conception, ou plus précisément, de

changement dans la granularité des éléments de base considérés pour la conception. La figure 3.4 résume l'évolution des méthodes de conception depuis le début des années 70. On constate qu'une nouvelle méthode de conception prend comme "élément de base" un ensemble ("mer") d'éléments de base de la méthode précédente [23].

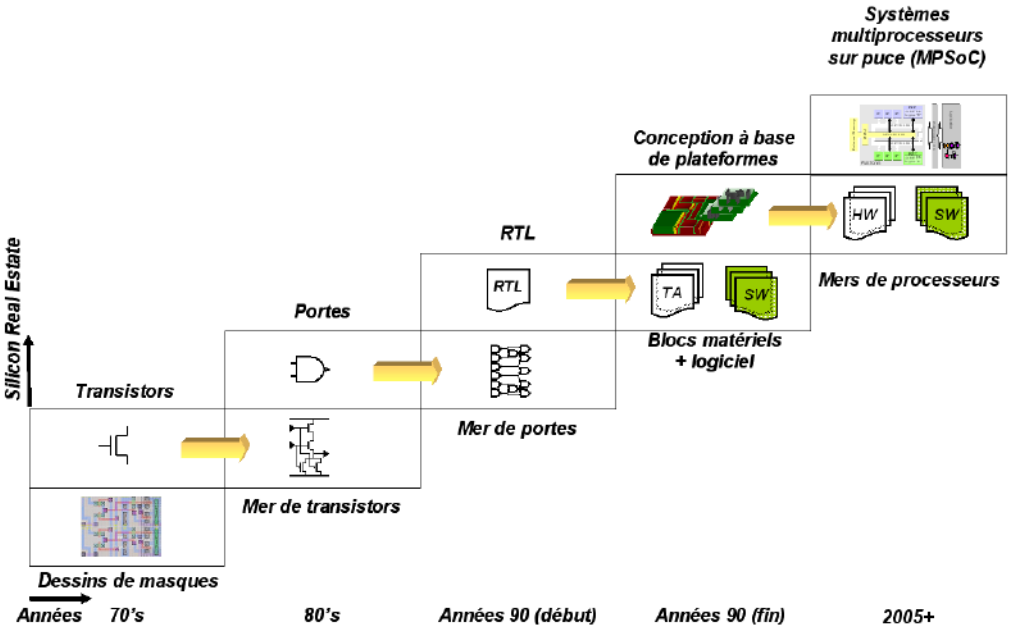



Figure 3.4. Evolution des méthodes de conception.

### 3.2. Les langages de description de matériel

En électronique, un langage de description de matériel (HDL) est un langage d'une classe de langages informatiques, de langages de spécification, ou de langages de modélisation, pour la description formelle et la conception de circuits électroniques, et plus généralement de circuits logiques numériques. Un HDL peut décrire le fonctionnement du circuit, sa conception et son organisation, il permet aussi de vérifier son fonctionnement par des moyens de simulation. En effet, les HDL sont des standards d'expressions, fondées sur des textes, pour décrire la structure spatiale et temporelle et le comportement des systèmes électroniques. Comme tous les langages de programmation, la syntaxe et la sémantique d'un HDL inclut des notations explicites pour exprimer la simultanéité. Toutefois, contrairement à la plupart des autres langages, un HDL doit inclure la notion explicite de temps, qui est un attribut principal pour une conception matérielle.

Il est certainement possible de représenter la sémantique des langages de programmation de matériel à l'aide des langages traditionnels tels que C++, le fonctionnement de tels programmes doivent être complétés par des bibliothèques de classe lourde et vaste. Pour remédier à cet inconvénient, plusieurs langages de description de matériel ont été proposés depuis les années 70. En effet, après plusieurs langages (ISP « Instruction Set Processor - développé à l'université Carnegie Mellon, 1977», KARL « développé à l'Université de Kaiserslautern, 1977 », ABL « 1980 », ABEL « 1983 »), le premier HDL moderne, Verilog, a été présenté par Gateway Design Automation en 1985. En 1987, une demande auprès du département américain de la défense a conduit à l'élaboration de VHDL. La dernière itération de Verilog, officiellement connu sous le nom IEEE 1800-2005 SystemVerilog, introduit de nombreuses nouvelles fonctions (classes, variables aléatoires, et les propriétés/assertions) pour

répondre aux besoins croissants de la randomisation du fichier testbench, la hiérarchie de conception, et la réutilisation. Le SystemVerilog est le premièr grande HDL à offrir une orientation objet [12].

SystemC est aussi considéré comme étant un langage de description de matériel. C'est le fruit de contributions de plusieurs sociétés. En 1989, Synopsys met son outil commercial Scenic dans le domaine libre, et crée la version 0.9 de SystemC. Une première contribution de Frontier Design donne lieu à la version 1.0, et une autre de CoWare aboutit en 2000 à la version 1.1 (première version officielle de SystemC). L'OSCI (Open SystemC Initiative) est alors crée, rassemblant une multitude de sociétés et laboratoires de recherche, cette organisation est en charge de diffuser, promouvoir et rédiger les spécifications de SystemC [6].

Le SystemC permet aux concepteurs de disposer de modèles de composants matériels et logiciels à un haut niveau d'abstraction. HDL-simulation a permis aux ingénieurs de travailler à un niveau supérieur d'abstraction que la simulation au niveau schématique, et donc une plus grande capacité de conception des centaines de milliers de transistors. Un autre niveau plus haut que le niveau de SystemC, est celui des langages de spécification (comme par exemple UML pour Unified Modeling Language). Ce haut niveau d'abstraction, généralement sous forme graphique, permet de comprendre dès le début de la conception les interactions entre les différents éléments, de trouver les meilleurs compromis et de procéder rapidement à des tests et des vérifications à travers la simulation. Certes, le développement d'un modèle de composant a un certain coût mais la productivité s'en trouve rapidement améliorée à travers la réutilisation des modèles de composants [76].

Malheureusement, à l'heure actuelle aucune méthodologie ne nous permet de développer un flot de conception complet pour les systèmes sur puce à partir d'une description haut niveau. En plus, les langages haut niveau généralement utilisés uniquement pour la simulation. Pour la synthèse et l'implémentation physique de ces descriptions, nous avons généralement besoin des outils de conversion qui sont généralement payants ou bien propriétaires, comme par exemple SystemCrafter[2], HandelC-Synthesis [3], VCC (Virtual Component Co-design), CoWare, GAUT et STARSOC [6] développé au niveau du laboratoire LE2I[4]. En plus, les codes générés automatiquement, par ces outils, à partir d'une description haut-niveau, ne sont pas forcément optimaux (voir même non synthétisables), une opération de raffinement sur ces codes est toujours nécessaire.

### 3.2.1. Les HDLs (VHDL et Verilog)

Le langage VHDL a été créé pour le développement de circuits intégrés logiques complexes. Il doit son succès, essentiellement, à sa standardisation par IEEE[5] sous la référence (IEEE 1076.87 et 1164.93), qui a permis d'en faire un langage unique pour la description, la modélisation, la simulation, la synthèse et la documentation. Le VHDL est écrit indépendamment d'une technologie particulière ou d'une chaîne de conception, il facilite le passage entre les différentes technologies utilisables qui évoluent sans cesse [9]. La description VHDL peut prendre l'un des trois types de modèles :

- Le modèle comportemental qui décrit la fonctionnalité d'un objet par un algorithme séquentiel ou une table de vérité, sans référence à une implémentation quelconque ;

[2] http://www.mentor.com/products/fpga/handel-c/
[3] http://www.systemcrafter.com/
[4] LE2I, Laboratoire Electronique, Informatique et Image, Université de bourgogne, Dijon. http://le2i.cnrs.fr/ .
[5] IEEE, Institute of Electrical and Electronics Engineers.

- Le modèle flux de données qui décrit le flux entre l'entrée et la sortie au niveau bit, par des équations élémentaires ;
- Le modèle structurel qui représente la constitution de l'objet en un ensemble d'objets élémentaires interconnectés (proche du schéma).

De même, le Verilog est un langage utilisé pour la description des systèmes numériques allant d'un simple flip-flop (bascule D) jusqu'aux microprocesseurs. Il peut modéliser un système par n'importe quelle vue, structurelle ou comportementale, à tous les niveaux de description. De plus il peut servir non seulement à simuler un système mais aussi à le synthétiser, c'est-à-dire être transformé par des logiciels adaptés (synthétiseurs) en une série de portes logiques prêtes à être gravées sur le silicium [5].

Les deux langages de description comportementale (VHDL et Verilog) ont des domaines d'utilisation plus vastes que la synthèse de circuits logiques. Ceci présente l'avantage d'un langage unique pour les différents niveaux de conception d'un projet qui va de la description et simulation du cahier des charges, à la description des fonctions logiques synthétisées dans les différents circuits.

### 3.2.2. Les langages de simulation de haut niveau

Le premier langage considéré comme haut niveau est le SystemVerilog qui est une extension majeure de la norme IEEE 1364-2001, qui spécifie le langage Verilog, spécialement conçue pour supporter de plus hauts niveaux d'abstraction pour la modélisation et la vérification. Il a été développé par Accellera pour améliorer la productivité dans la conception des circuits intégrés à grande échelle basés sur des IP et des systèmes de bus. Il supporte la modélisation à haut niveau et la vérification avec des assertions. Il permet un appel direct de fonctions en C/C++/SystemC en utilisant son Direct Programming Interface (DPI). SystemVerilog possède les types de données comme le C et la possibilité de créer des classes en ayant aussi les signaux au niveau matériel. De plus, il est possible de créer ses propres structures [77]. Des notions d'interfaces existent, ainsi que des primitives de niveau système et des mécanismes de communication telle que les boîtes aux lettres et les sémaphores, etc.

Un autre langage haut niveau le SystemC qui permet une modélisation de systèmes au niveau comportemental. SystemC est une bibliothèque en C++ orienté objet pour la modélisation logicielle et matérielle, il n'est pas un langage à part entière mais un ensemble de classes C++ qui introduisent les concepts nécessaires à la modélisation du matériel (par exemple la notion de processus concurrents, la notion du temps et quelques types de données matériels spécifiques). Conservant les fonctionnalités du C++, il reste possible de décrire des fonctions purement logicielles. SystemC permet donc de modéliser des systèmes matériels, logiciels, mixtes ou même non-partitionnés. Il est donc particulièrement approprié à la conception de systèmes de type SoC. SystemC intègre également la possibilité de simuler le modèle conçu, puis, par raffinements successifs, d'aboutir à une représentation implémentable. Depuis Décembre 2005, SystemC est standardisé auprès de l'IEEE sous le nom de IEEE 1666™-2005 (l'actuelle version2.2.0 stable de SystemC). Bien que libre, la librairie SystemC ne peut être téléchargée que depuis le site de l'OSCI (http://www.systemc.org).

Le SystemC comporte plusieurs avantages dont : la simulation rapide, plusieurs niveaux d'abstraction, le partitionnement matériel/logiciel à partir d'une même source, les protocoles de communication, la création de spécifications exécutables du système, etc. Cependant, SystemC n'a pas été proposé afin de remplacer le HDL. Aussi le passage d'un SystemC comportemental à un SystemC synthétisable

demande beaucoup de temps et de travail manuel. De plus, la précision d'une simulation SystemC n'égale pas encore la précision des métriques extraites d'une implémentation réelle sur FPGA [6].

Dans le même esprit que SystemC, Handel-C a été créé par la compagnie Celoxica pour la description des systèmes numériques mixtes matériel/logiciel. Il est basé sur le standard ANSI-C pour l'implémentation de fonctionnalités en matériel, l'exploration architecturale et le codesign. Pour supporter la description matérielle, Handel-C gère les longueurs variables des structures de données (vecteurs de bits) ainsi que le traitement parallèle des événements et des communications. La différence principale avec SystemC est qu'il n'utilise pas d'automates à états finis (Finite State Machines) mais une méthode propriétaire de description des écoulements périodiques et parallèles.

### 3.2.3. Les langages de spécification de haut niveau

Parmi les langages hauts niveau, utilisé récemment à un niveau d'abstraction plus élevé dans les systèmes embarqués, on note le langage UML. Ce langage crée par les informaticiens, spécialement pour la conception des systèmes logiciels, a été standardisé par l'OMG[6]. En effet, UML a pour but de permettre la modélisation de n'importe quel système informatique. UML est riche en notation, il définit treize diagrammes organisés en trois catégories : six diagrammes de structure (qui décrivent la structure statique du système) dont le diagramme de classe et le diagramme de structure composite, trois diagrammes de comportement dont le diagramme d'activité et quatre diagrammes d'interactions dont le diagramme de séquence. La dernière évolution majeure de ce langage, UML2, permet une modélisation encore plus précise de la structure et du comportement d'un système. Elle autorise en fait le passage d'une approche centrée sur le code à une approche dirigée par les modèles [78].

Une des particularités d'UML est qu'il peut se spécialiser pour des domaines particuliers via le mécanisme de profil. Un profil est une collection d'extensions et éventuellement de restrictions qui décrivent un domaine particulier. Ce mécanisme simple et efficace pour étendre UML, a permis de définir plusieurs profils adaptant UML à la majorité des domaines d'application. Plus d'une dizaine de profils ont été standardisée par l'OMG. On se sert en général du profil pour introduire les concepts manquants pour les systèmes que l'on cherche à représenter comme par exemple les systèmes embarqués temps réel, tel est l'exemple du profil MARTE (Modeling and Analysis of Real-Time and Embedded systems). Ce profil a été proposé par l'OMG afin de tenir compte des aspects logiciels et matériels des systèmes embarqués. Il a pour objectif d'étendre UML pour l'utiliser dans une approche de développement dirigée par les modèles. MARTE définit les fondations de la modélisation des systèmes embarqués temps réel. L'intention n'est pas de définir de nouvelles méthodologies de conception ou de nouvelles techniques d'analyse des systèmes embarqués temps réel, mais de les soutenir par une riche base d'annotations [76].

En plus du langage UML, la modélisation d'un système peut être réalisée en utilisant Matlab. Cet outil permet la modélisation à plusieurs niveaux d'abstraction, il possède son propre flot de conception. Il permet aussi un prototypage rapide avec les FPGA [77]. En plus des fonctionnalités de traitement du signal et de l'image, l'aspect de conception du contrôle, etc., le logiciel Matlab fournit des modèles de

[6] OMG (Object Management Group), organisme créé en 1989 à l'initiative de grandes sociétés (HP, Sun, Unisys, American Airlines, Philips...). Son rôle est de promouvoir des standards qui garantissent l'interopérabilité entre applications orientées objet, développées sur des réseaux hétérogènes. http://www.omg.org/.

design de base pour le contrôle. Il permet, donc, de décrire l'algorithme par la création d'un modèle et la simulation de ce modèle par la suite. Il permet ensuite le prototypage rapide et la génération du code HDL pour les systèmes embarqués.

Matlab permet un raffinement progressif partant d'un haut niveau d'abstraction en allant vers le bas niveau. Il permet deux types de génération de code automatique. Premièrement, l'outil Real-Time Workshop peut générer automatiquement du code C-ANSI à partir d'un modèle Simulink pour le téléchargement vers un processeur DSP ou un processeur embarqué. De plus, des mises en œuvre matérielles peuvent être générées par le générateur de système de Xilinx [78].

### 3.2.4. Classification des langages de description de matériel

L'approche « schématique » pour la conception des systèmes complexes au niveau des portes logiques et des fonctions de base RTL semble aujourd'hui délaissée au profit d'une approche «textuelle». En effet, on travail maintenant au niveau système (fonctionnalité) et non pas au niveau porte logique. L'approche schématique reste cependant toujours valable et est plutôt réservée à la conception des petits systèmes [11]. La figure 3.5 montre une classification hiérarchique des langages de description de matériel.

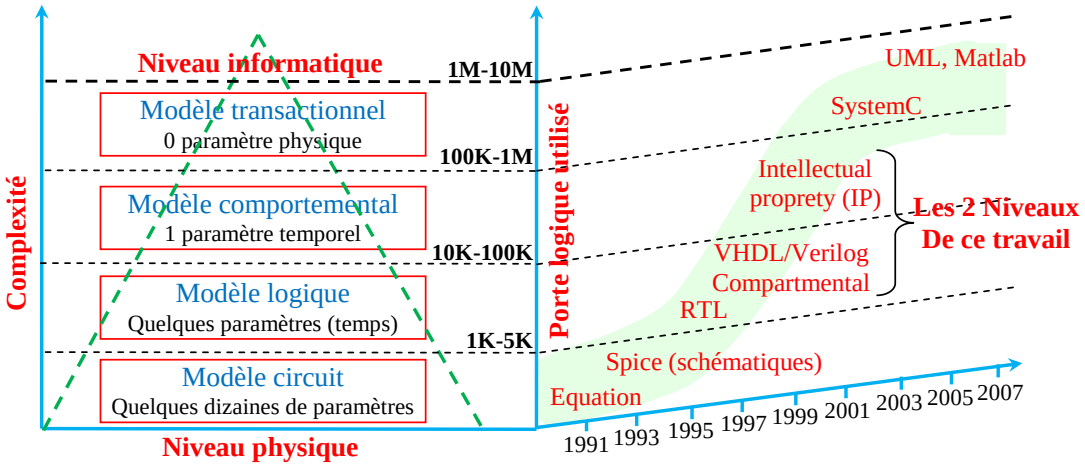



Figure 3.5. Classification hiérarchique des langages de description matérielle.

Dans l'approche textuelle, on utilise des langages de description de matériel comme VHDL ou Verilog pour synthétiser une fonction numérique. Ces langages de description de matériel sont utilisés conjointement avec un compilateur ou un simulateur. Ces langages deviennent un standard et leur choix participe ainsi à la pérennité du produit. Il existe à l'heure actuelle d'excellents synthétiseurs mixtes et multi-technologiques. Les fabricants de FPGA proposent maintenant leur propre synthétiseur.

Dans ce travail, qui consiste à l'implémentation de l'encodeur H.264/AVC, nous avons choisi le niveau HDL (le langage VHDL) pour la conception des accélérateurs matériels et le niveau IP pour la conception du système à base de processeur MicroBlaze de Xilinx. En effet, nous allons proposer par la suite dans les chapitres 4 et 5 des implémentations pour les différents modules de traitement de l'encodeur H.264, ensuite dans le chapitre 6, nous allons proposer l'utilisation des IPs proposés par Xilinx avec nos accélérateurs pour construire un système de codage vidéo sur une plateforme reconfigurable.

## 4. Les outils de conception des systèmes temps réel

De nos jours, une exploration rapide de plusieurs architectures est requise et peut être possible grâce aux outils de conception et de codesign. L'évaluation des outils de conception et des méthodes qui y sont associées sera nécessaire pour connaître leurs capacités dans la réalisation d'un système sur puce. De plus, la diminution du temps de conception peut être réalisée en combinant judicieusement les outils entre eux. Idéalement, il faut des outils nécessitant un court temps de simulation de la spécification à l'implémentation physique. En effet, pour la conception d'un système embarqué temps réel, il est nécessaire d'installer plusieurs outils pour pouvoir fonctionner correctement [19].

### 4.1. Généralités sur les outils de développement

Concevoir un système sur puce signifie généralement un important travail manuel et une grande expertise pour choisir l'architecture, concevoir les accélérateurs si nécessaire, 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. Un grand nombre de méthodologies adopte le concept d'utilisation des composants et des interfaces standard de HW et de SW (les IP). 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 [11].

#### 4.1.1. Les outils de synthèse

Le synthétiseur est un compilateur particulier capable, à partir d'un HDL, de générer une description structurelle du circuit. A travers le style d'écriture de la description synthétisable (niveau RTL), le synthétiseur va reconnaître un certain nombre de primitives qu'il va implanter et connecter entre elles. Ce sera au minimum des portes NAND, NOR, NOT, des bascule D, des buffers d'entrées-sorties, mais cela peut être aussi des compteurs, des multiplexeurs, des mémoires RAM, des registres, des additionneurs, des multiplieurs ou tout autres blocs particuliers.

Les outils de synthèse, permettent de transformer le code source HDL en un fichier Netlist (liaisons inter-cellules logiques), sont essentiellement fournis par 3 grands fournisseurs. En effet, les fabricants de circuits travaillent avec des sociétés partenaires, développant les outils logiciels adaptés à leurs circuits. Citons les outils de synthèse les plus utilisés à l'heure actuelle :

- La société Mentor Graphics, leader actuel du secteur, développe un logiciel de synthèse appelé « Leonardo Spectrum » ;
- La société Synopsys commercialise « FPGA Compliler II » et « FPGA Express » ;
- La société Cadence propose quant à elle l'outil « Synplify ».

Dans ce cas, le VHDL décrit la fonctionnalité souhaitée et ceci indépendamment de la technologie. Le nombre de primitives trouvées par le synthétiseur donne une évaluation de surface (pour un ASIC) ou de remplissage (pour un FPGA). La technologie choisie apporte toutes les données temporelles relatives aux primitives, notamment les temps de traversée de chaque couche logique sont connus. Une première évaluation de vitesse d'ensemble du circuit peut ainsi être réalisée par le synthétiseur (en considérant des retards fixes par porte, ce qui constitue une approximation grossière). Outre la

description VHDL, le concepteur a la possibilité de fournir au synthétiseur des contraintes temporelles (réalistes) ou des caractéristiques électriques des entrées-sorties. Le synthétiseur, par un certain nombre d'itérations, essaiera d'optimiser le produit surface-vitesse.

### 4.1.2. Simulation et simulateur numérique

Durant cette dernière décennie, la progression constante de la puissance des ordinateurs associée à l'abaissement considérable des coûts a ouvert la possibilité de réaliser des simulations numériques sur des ordinateurs personnels bon marché. La simulation est une des étapes clés d'un flot de conception, c'est à ce niveau que le design d'un circuit (au niveau RTL) est vérifié d'un point de vue fonctionnel, avant synthèse. Cette opération, qui a toujours existé en conception électronique, est censée vérifier que les fonctions d'un circuit prévues dans les spécifications sont bien réalisées.

Pour y parvenir, il faut écrire des suites de données de test (testbench), qu'on applique au fichier de conception. L'objectif étant ensuite de comparer les réponses obtenues en sortie du simulateur aux réponses attendues selon la fonction testée. Un simulateur est un dispositif technique permettant de reproduire de façon virtuelle le comportement d'un phénomène réel.

En particulier un simulateur sera utilisé quand le système réel est inobservable ou difficilement observable pour toutes sortes de raisons (dimension, sécurité, coût, inexistence, etc.). Alors comme étant un élément incontournable d'un flot de conception, les simulateurs numériques sont devenus le cœur d'environnements complets de vérification. Ils sont désormais capables pour les plus puissants d'entre eux de gérer plusieurs langages simultanément, de supporter les assertions ou de faire tourner de manière native un testbench.

Depuis quelques années, les tests de validation fonctionnelle d'un circuit sont tous écrits dans des langages standard, comme VHDL, Verilog, ou plus récemment SystemVerilog, voire C/C++ pour des besoins liés à des constructions spécifiques ou pour des raisons de culture d'entreprise. Désormais, l'ensemble des simulateurs du marché supporte un ou plusieurs de ces langages.

### 4.1.3. Les outils de codesign

Des outils de partitionnement sont nécessaires pour aider les concepteurs à partager ce qui devra être implémenté en hardware ou en software. Mais ces composants de logique programmable rendent difficile les tests du système en fonctionnement. C'est pourquoi cette technologie fait appel à des blocs IP qu'il suffit d'intégrer. Ces outils permettent de pouvoir faire des simulations et des débogages softwares. Chez Xilinx l'outil EDK permet d'effectuer le découpage «automatique» des parties software et hardware de son projet. L'équivalent s'appelle «SoPC Builder » pour la conception sur des composants de la société Altera. En plus, pour pouvoir intégrer et vérifier les blocs matériels et logiciels, le concepteur doit utiliser des outils de vérification avancés comme la co-vérification ou encore des outils de preuve formelle. La difficulté est de pouvoir vérifier en parallèle la conception hardware et la conception software. Ces outils développés pour la conception de SoC sont maintenant utilisés pour la conception de système sur FPGA.

## 4.2. Flot de prototypage FPGA générique

La figure 3.6 montre le flot générique de prototypage basé sur une plateforme FPGA. Comme indiqué la première étape dans le flot est une étape de spécification du modèle ciblé par le prototypage. Il

convient de signaler ici que quelque soit le langage utilisé pour la description du modèle, une condition nécessaire au bon déroulement du processus de prototypage est que la description soit synthétisable. La notion de synthétisabilité impose d'une part que la description du modèle se fasse au niveau RTL ; et que d'autre part elle tienne compte de certaines spécificités des circuits FPGA ciblés. Il convient de dire qu'une description complètement synthétisable sur une plateforme FPGA donnée peut s'avérer être incompatible avec une autre plateforme (problèmes de ressources, circuit non routable). Cependant, il est utile de préciser que le développement des technologies FPGA tend à uniformiser leur architectures ce qui réduit considérablement l'impact de cette limitation.

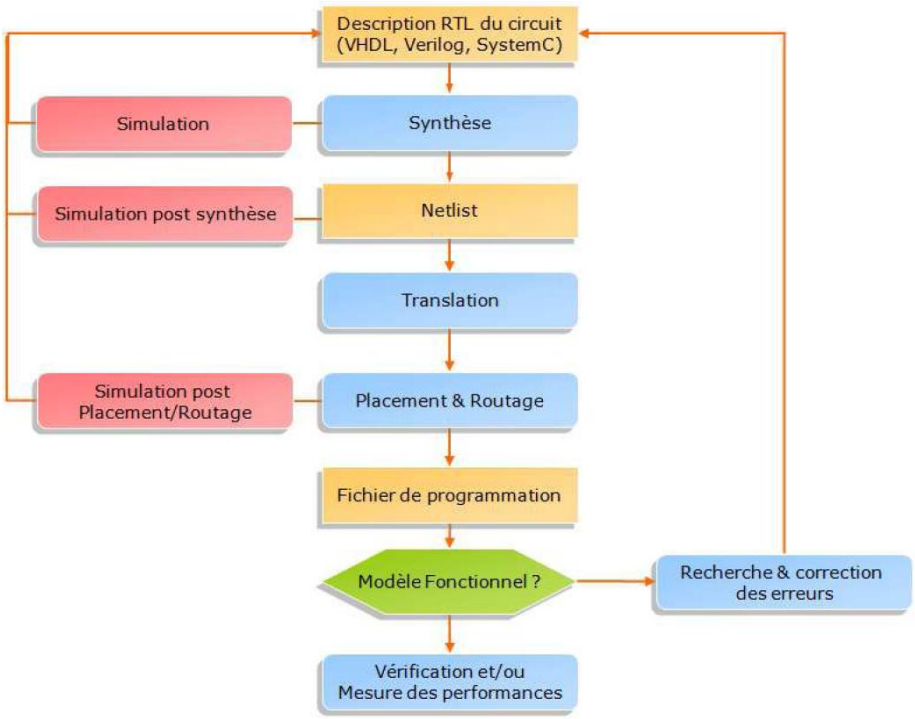



Figure 3.6. Flot générique de prototypage FPGA.

Lors de l'étape de synthèse, la description HDL du circuit est transformée en un assemblage (Netlist) de primitives de base (portes logiques, bascules, etc.). A ce point il est possible d'effectuer une étape de simulation post-synthèse, l'intérêt de cette étape étant de vérifier l'équivalence du comportement du circuit pré/post synthèse. En particulier, il est important de vérifier les propriétés temporelles du circuit.

La translation transforme la Netlist issue de la phase de synthèse en des primitives FPGA. En plus des primitives standards, la Netlist issue de cette phase peut contenir des primitives plus élaborées. La translation est généralement optimisée de façon à tirer profit au mieux des ressources FPGA.

Avant la phase de placement et de routage, chaque élément de la Netlist post-translation est associé à un type de cellule logique FPGA, il vient ensuite le placement qui place les éléments de la Netlist (correspondant maintenant à des cellules) physiquement sur le circuit FPGA, en tenant compte d'un certain nombre de contraintes : délais de propagation, placement par rapport aux E/S, etc. Une fois l'emplacement physique de chaque cellule fixé, le routeur effectue le calcul des interconnexions entre les cellules physiques en respectant la connectivité des cellules comme indiqué dans la Netlist donnée en entrée. Il est possible ici aussi d'effectuer une simulation post placement et de routage du circuit afin de vérifier une fois de plus l'équivalence avec les spécifications d'entrée.

Une fois chacune des cellules de la Netlist mappées sur une cellule physique, et une fois que les interconnexions entre cellules ont été calculées, le fichier de configuration (bitstream) du FPGA peut être généré. Le bitstream est une image binaire du circuit FPGA, il y détermine pour chaque élément (cellule logique, table de correspondance LUT, élément de routage, cellule mémoire) l'état logique. L'exécution proprement dite du circuit est possible après que le fichier ait été chargé sur le circuit FPGA. Il est alors possible de vérifier les fonctionnalités du modèle et d'effectuer des mesures de performances.

### 4.3. Exemples des outils de développements des systèmes embarqués

La conception des SoC requiert de la programmation au niveau logiciel et/ou matériel. Il est important d'avoir un outil permettant l'intégration des modules sous différentes formes de textes ou de graphiques et permettant l'incorporation de divers langages comme VHDL, C et C++. Un mode graphique permet de visualiser plus facilement le fonctionnement d'un module en représentant la fonctionnalité voulue sous forme graphique, par exemple une machine à états finis.

#### 4.3.1. HDL Designer de Mentor Graphics

Mentor Graphics Corporation[7] a été considéré comme étant le leader des outils de création, d'analyse, de synthèse et de gestion de circuits intégrés numériques basés sur des standards. En 2005, cette compagnie a annoncée la sortie d'un nouvel environnement, de création et de vérification simultanées, disponible dans la toute dernière version de la suite d'outils HDL Designer Series. Largement adopté comme solution de conception HDL la plus efficace pour les circuits ASIC et FPGA, l'environnement HDL Designer Series complet convient aussi bien aux ingénieurs travaillant individuellement qu'aux grandes équipes de conception. Cette solution se traduit par une réduction significative du cycle de conception et des modifications.

L'utilisation de HDL Designer permet d'accroître la productivité des concepteurs. Il servira de contrôleur de version, de gestionnaire de bibliothèques et il fournit un ensemble d'autres fonctionnalités connexes à la gestion d'un projet de conception d'un système embarqué. De plus, son interface permet de démarrer une simulation avec d'autres langages. Une représentation graphique et modulaire d'un système est plus facilement compréhensible lors de la conception [77]. En plus, la conception, la vérification, l'analyse, le débogage et la création de la documentation des conceptions sont désormais beaucoup plus rapide, nécessitant parfois cinq fois moins de temps. En effet, la fonctionnalité intégrée de vérification des conceptions statiques permet aux concepteurs qui ont recours à HDL Designer d'analyser le code RTL avec précision au cours de la création d'une conception. Ils peuvent rapidement identifier et résoudre les violations de règles responsables de la modification des processus de simulation et de synthèse ou, en fin de cycle, de modifications sur silicium. La suite d'outils HDL Designer Series fournit une plateforme complète de développement HDL ; elle inclut également les outils de synthèse, ModelSim et Precision.

#### 4.3.2. Le logiciel Synplify de Cadence

Le logiciel Synplify est un outil de synthèse, son objectif premier quel que soit l'algorithme utilisé, est de transformer la conception d'un circuit du point de vue des échanges de données au niveau registres,

[7] L'un des principaux fournisseurs mondiaux de solutions de conception électronique. http://www.mentor.com/.

écrite en VHDL ou Verilog, en une représentation de ce circuit au niveau portes logiques, associée à un calcul des délais (le timing) et à une estimation de la surface finale du circuit. Le but étant d'obtenir le plus petit circuit possible.

Il réalise entre autre la synthèse physique du composant FPGA. La définition de la notion de synthèse physique, apparue plus récemment, est plus délicate. On peut dire qu'il s'agit d'une opération d'optimisation du circuit, grâce à un premier placement des cellules associé à un routage global, qui tient compte d'informations physiques issues des bibliothèques des cellules du fondeur. La synthèse physique, qui utilise en entrée les fichiers fournis par les outils de synthèse logique, se positionne donc avant les opérations de placement et de routage détaillé.

Le logiciel Synplify s'intègre parfaitement dans le flot de conception de Xilinx. En fait il remplace l'outil de synthèse XST du fabriquant Xilinx fournit avec ISE. L'interface entre le logiciel ISE et le logiciel Synplify est totalement automatisée.

### 4.3.3. ModelSim de Mentor Graphics

ModelSim est un outil de simulation HDL de Mentor Graphics. Il est aussi intégré au flot de conception Xilinx. Il permet une simulation à un plus bas niveau d'abstraction. ModelSim est un simulateur VHDL largement répandu et un simulateur mixte combinant le VHDL et le Verilog. Les produits ModelSim ont une architecture unique basée sur des technologies comme Optimized Direct Compile pour des compilations et des simulations rapides par le Single Kernel Simulation (SKS). De plus, il incorpore le langage TCL/TK pour une grande flexibilité et une vérification rapide.

Une personnalisation facile est permise par le TCL/TK qui permet de développer des interfaces graphiques. Il possède également des connexions FLI/PLI (Foreign Language Interface/Programming Language Interface) permettant une intégration du C/C++. Les nouvelles versions possèdent un débogueur intégré pour faciliter la vérification d'un système. ModelSim inclut un comparateur de chronogramme, des analyseurs de performance et la capacité de donner la couverture de code. Le comparateur de chronogramme permet d'augmenter la vitesse de vérification entre deux simulations pour identifier les erreurs. Il permet aussi d'analyser la vitesse de simulation en identifiant les congestions. Celui-ci peut découvrir les codes qui ne sont pas efficaces, les cellules d'une bibliothèque non accélérées, des signaux inutiles dans la liste de sensibilité, etc. La couverture de code permet de connaître combien de fois l'exécution d'un segment de code a été effectué. Il est possible de mettre un moniteur sur un signal et de le voir directement dans le testbench [77].

### 4.3.4. SOPC Builder d'Altera

SOPC Builder est un outil puissant de développement des systèmes. Il permet de définir et de générer un SoPC complet, en beaucoup moins de temps que d'utiliser les méthodes traditionnelles d'intégration manuelle. SOPC Builder est inclus dans le cadre du logiciel Quartus-II. Il est utilisé pour la création des systèmes basés sur le processeur NIOS. Toutefois, SOPC Builder est un outil d'usage général pour créer des systèmes qui peuvent ou non contenir un processeur et peut comprendre un processeur soft autre que le processeur NIOS [28].

SOPC Builder automatise la tâche d'intégration des composants matériels. En utilisant les méthodes de conception traditionnelles, il est nécessaire d'écrire manuellement les modules HDL pour connecter et

pour rassembler les pièces élémentaires du système. Avec l'utilisation de SOPC Builder, il est possible de spécifier les composants du système, il est également possible de définir et d'ajouter des composants personnalisés ou choisir parmi une liste des composants fournis.

Le logiciel Quartus se présente sous la forme d'une interface graphique et permet à l'utilisateur de définir les différents éléments qui composent le système que l'on souhaite réaliser. De plus il permet aussi de paramétrer les différentes caractéristiques des périphériques et de créer les interfaces de ceux ci avec le processeur NIOS. Lorsque la définition du système est finie, le SOPC Builder génère les éléments nécessaires au développement de la partie logicielle et matérielle. En effet, lors de la définition du système, l'utilisateur rajoute des périphériques ou des interfaces vers des périphériques externes au processeur NIOS. Tous les périphériques connus par le SOPC Builder sont soit fournis par Altera soit rajoutés par l'utilisateur par l'intermédiaire de la création d'une bibliothèque. Il permet la génération de code VHDL ou bien Verilog selon le choix. La deuxième partie du SOPC Builder est la génération du système qui consiste à générer les fichiers nécessaires au développement matériel et logiciel.

### 4.4. Les Outils de développement de Xilinx

Comme mentionnée dans le premier chapitre, Xilinx est une compagnie de composants électroniques programmables. En plus du matériel, Xilinx fourni des outils de conception des systèmes nous permettant de mettre au point des projets sur leurs différentes plateformes reconfigurables. En effet, Xilinx fournit dans son pack d'outils différents soft permettant la création de systèmes embarqués sur puce, parmi ces softs on dénombre ISE (Integrated Software Environment) et EDK. L'outil ISE est spécialement pour la réalisation des projets d'IPs matériels à partir d'un code HDL. L'outil EDK, englobant SYSTEM GENERATOR FOR PROCESSOR et Xilinx Platform Studio (XPS), va nous permettre d'établir un lien direct entre la partie matérielle et logicielle d'un système [23]. Tous les deux nous offrent la possibilité d'avoir un bitstream pour la programmation des FPGA suivant l'application ciblée [20].

#### 4.4.1. Flots de conception Xilinx

Le flot de conception standard de Xilinx est composé de deux étapes principales : la conception et la synthèse du design et puis l'implémentation et la vérification de ce design. Une vue générale du flot de conception est présentée par la figure 3.7 [19]. La conception d'un système embarqué inclut typiquement quatre phases (la création et la vérification de la plateforme matérielle et la plateforme logicielle).

Dans l'outil EDK, la plateforme matérielle est définie par le fichier MHS (Microprocessor Hardware Specification). La plateforme de vérification permet à l'utilisateur de définir le modèle de simulation pour chaque composant du système (processeur et périphériques). Si l'application logicielle est disponible sous sa forme exécutable, elle peut être utilisée pour initialiser les mémoires.

La plateforme logicielle est définie par le fichier MSS (Microprocessor Software Specification). La création et la vérification d'une application logicielle passe par différentes étapes : d'abord l'écriture du code en C, C++ ou assembleur qui va être exécuté sur les plateformes logicielles et matérielles. 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 [14].

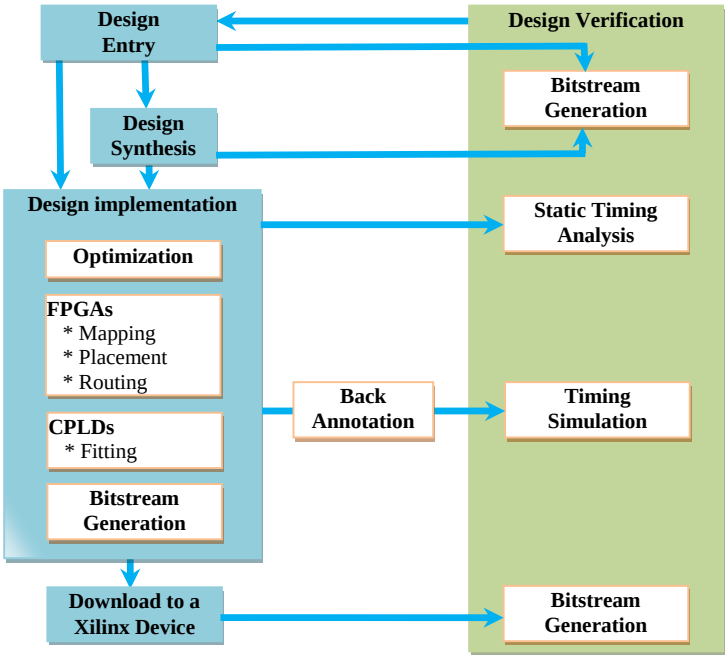



Figure 3.7. Flot de conception Xilinx générique

La conception et la simulation sont les deux étapes principales dans le flot de conception Xilinx. Les tâches de conception permettent de passer d'une description à une autre pour arriver au ficher bitstream de configuration. En effet, une synthèse logique permet de passer d'une description RTL de l'architecture à une description au niveau porte logique (Netlist). La description d'éléments logiques est optimisée suivant les contraintes de vitesse, de surface ou de consommations imposées par le concepteur. L'outil de synthèse remplace les éléments logiques génériques par ceux spécifiques au FPGA ciblé. Le placement et le routage permettent de convertir la description matérielle en un fichier de configuration. L'outil de synthèse génère ce fichier qui sert à la configuration des matrices d'interconnexion du circuit FPGA. A chaque étape de la conception, l'environnement de CAO permet d'effectuer des simulations afin de valider chaque étape de l'implantation :

- Simulation fonctionnelle (niveau RTL = Register Transfer Level) ;
- Simulation post-synthèse (niveau porte logique) ;
- Simulation post-layout (niveau physique).

L'environnement ISE, de la société Xilinx, est utilisée à la fois pour la conception et pour l'implantation sur puce. Les simulations comportementales VHDL sont, généralement, réalisées avec l'outil ModelSim.

### 4.4.2. Xilinx ISE

ISE est un environnement intégré de développement de systèmes numériques ayant pour but la synthèse/implémentation matérielle sur FPGA. Les designs peuvent être décrits sous trois formes principales : sous forme de schémas, sous forme de HDL ou bien sous forme de diagrammes d'états. Pour cela, ISE intègre différents outils permettant de passer à travers tout le flot de conception d'un système numérique. Il inclut des éditeurs et outils pour la saisie du système, il dispose aussi d'un

ensemble d'outils, de synthèse et d'implémentation, regroupés dans un seul flot de conception. Les détails de ces outils sont donnés comme suit :

1. Saisie de conception (Design Entry) : Editeur HDL, éditeur de Machine à Etat Fini (StateCAD), éditeur de schéma (Engineering Capture System (ECS)) et générateur d'IP (CORE Generator) ;
2. Synthèse : XST (Xilinx Synthesis Technology), Leonardo Spectrum de Mentor Graphics et Synplify de Cadence ;
3. Simulation : Générateur de TestBench HDL et intégration du simulateur ModelSim de Model Technology ;
4. Implémentation : Translate, MAP, placement et routage (PAR), floor planner, éditeur FPGA, timing Analyzer XPower, Fit (seulement CPLD) et Chipviewer (seulement CPLD) ;
5. Programmation du composant et formatage du fichier de Bitstream : BitGen et iMPACT.

### 4.4.3. Xilinx EDK

L'environnement EDK permet de développer une application complète à base de processeur embarqué et de l'intégrer à un FPGA. EDK donne accès à tous les réglages nécessaires pour l'application embarquée que l'on souhaite créer. Il permet la programmation d'un ou plusieurs MicroBlazes et de leurs périphériques. Il va s'occuper de placer les programmes des MicroBlazes en mémoire lors de la programmation du FPGA.

EDK permet d'une part la création des accélérateurs matériels au niveau RTL à partir d'une description en VHDL (il est possible d'ajouter directement le code synthétisé avec une extension .ngc), et d'autre part de générer l'architecture globale du système en incluant les processeurs logiciels (CPU) ainsi que les contrôleurs d'entrée/sortie nécessaires. L'ensemble de ces composants vont être interconnectés via un bus, qui est lui aussi généré en fonction des besoins architecturaux de l'application. EDK permet aussi :

- Génération du logiciel : permet l'extraction des parties logicielles de l'application et de générer par la suite le code machine correspondant à chaque processeur de la plateforme matérielle.
- Génération des communications : permet de générer le système de communication de l'architecture en fonction du nombre de composants constituants la plateforme matérielle.

EDK est proposé en plusieurs versions : MDK 1.9 (fin 2001), … EDK 3.1 (2003) jusqu'aux versions 8.2 et 9.2 supportant les FPGA Virtex5. Nous utilisons cette dernière version dans nos implémentations malgré l'appariation d'autres versions (EDK10.1, EDK11.1, EDK12.1, EDK12.2, EDK12.4). En effet, selon plusieurs pratiquants de ce domaine, la version 9.2 est la plus stable, elle contient tous les outils nécessaires pour la conception des systèmes embraqués temps réel. D'autres pratiquants préfèrent toujours la plus récente version, à cause de la présence des fonctionnalités bien précises. Exemple des scientifiques travaillant sur la reconfiguration dynamique, préfèrent travailler avec la version 12.1 qui offre une rubrique spécialement pour ce type d'applications.

## 4.5. Les outils utilisés

Pour la création d'un système temps réel, plusieurs critères sont nécessaires pour le choix des outils à utiliser. Parmi ces critères est le type de matériels utilisés (des cartes de prototypages Xilinx dans notre cas) où chaque fournisseur propose ces propres outils. Un autre critère aussi concernant la nature et les

contraintes du système temps réel lui-même, le langage (les langages) de programmation peut aussi influer sur le choix des outils, etc. Généralement, il est préférable d'utiliser plusieurs outils de différents fournisseurs, chacun dans l'une des étapes de conception. La figure suivante montre l'interaction des différents outils présentés dans les paragraphes précédente.

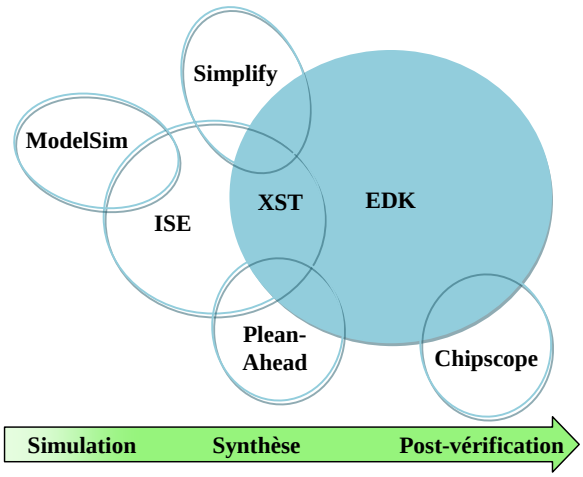



Figure 3.8. Choix des outils de conception des systèmes temps réel.

Dans notre cas, ou le but initial est l'implantation de l'encodeur H.264/AVC dans une plateforme reconfigurable, et après le choix de VHDL comme langage de programmation matériel, nous avons utilisé ISE pour la conception et l'implantation des accélérateurs matériels, ModelSim pour la simulation et la vérification des architectures, et enfin EDK pour l'intégration des différents accélérateurs matériels dans un système complet à base de processeur MicroBlaze.

## 5. Conclusion

Grâce aux blocs logiques configurables et aux ressources de routage programmables, les circuits FPGA peuvent être utilisés pour implémenter des fonctionnalités matérielles personnalisées sans jamais modifier physiquement le matériel. Des tâches de calcul numérique sont développées en utilisant des logiciels et à l'aide des langages de conception de matériel, les logiciels sont capable de simuler, de synthétisés et de compiler ensuite un fichier bitstream qui contient des informations concernant la façon dont les composants de routage et de logique du FPGA devraient être configurés et connectés. Dans la deuxième partie, pour la création et la vérification des accélérateurs matériels et leurs intégrations dans un système à base de processeurs, nous avons choisi les étapes suivantes :

- Description en VHDL au niveau RTL des blocs constituant le système temps réel (les modules de l'encodeur H.264) à partir de la solution architecturale retenue ;
- Validation fonctionnelle de la description VHDL au niveau RTL à l'aide de vecteurs de test (testbench) en utilisant le logiciel ModelSim ;
- Validation des performances de la description VHDL au niveau RTL et Synthèse de la description VHDL pour la cible FPGA retenue en utilisant le logiciel ISE ;
- Placement-routage de la Netlist sur le FPGA retenu et création des fichiers .ngc ;
- Intégration des accélérateurs dans un système à base de processeur MicroBlaze avec les différents IPs de Xilinx (Bus, DDR2, système ACE, etc.) en utilisant le logiciel EDK.

# Chapitre 4

# Implantation de la Gestion de Mémoire pour l'Encodeur H.264 sur des Plateformes Virtex5 de Xilinx

## 1. Introduction

Pour répondre aux besoins du temps réel, d'un grand nombre d'applications multimédias et de traitement vidéo, des solutions d'implémentations matérielles sur des plateformes reconfigurables ont été proposées. La plupart des solutions existantes sont basées sur des modules de traitement avec des entrées/sorties de 32bits. Toutefois, ces solutions ne donnent pas la méthode de gestion de mémoire pour ces données ou bien censé être réalisée par un composant logiciel.

Sur les circuits FPGA avec des mémoires internes de capacités limitées, et suivant l'énorme demande des ressources mémoires dans les algorithmes de traitement d'image et de vidéo, les implantations matérielles de ces algorithmes sont tous orientées vers l'utilisation des mémoires externes. Ces mémoires sont de type bien défini pour chaque plateforme FPGA (exemple de la carte Xilinx ML501 avec une mémoire externe de type DDR2-SDRAM).

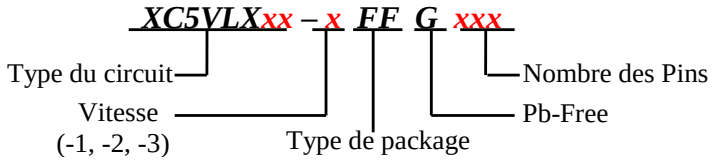
Pour avoir accès à une mémoire externe, quelque soit le type, on utilise un contrôleur mémoire qui sert comme intermédiaire entre le programme utilisateur et la mémoire physique. La compagnie Xilinx a mis des outils (CORE Generator) pour la génération des IPs, et ceci dans le but d'éviter les travaux multiples sur les mêmes projets. On y trouve dans ce cadre des additionneurs standards, des multiplieurs, des blocs mémoires, etc. Dans la version Xilinx ISE 9.2i.4, nous trouvons un générateur de contrôleur mémoire MIG2.0 (Memory Interface Generator).

Dans ce chapitre, nous commençons par une description des plateformes de prototypage utilisées ainsi que les structures de la mémoire DDR2 et le contrôleur de cette mémoire, ensuite nous présentons un contrôleur mémoire spécialement pour l'encodeur H.264. Ce contrôleur est réalisé par l'ajout d'une couche intelligente au contrôleur DDR2 initial. Dans cette couche, nous exploitons toutes les informations sur les besoins en mémoire des modules de l'encodeur H.264.

## 2. Les plateformes de prototypage de Xilinx

Comme nous avons dit dans le chapitre 1, une plateforme de prototypage est constituée d'un circuit FPGA, de mémoires de différent types, de convertisseurs A/N et N/A, des interfaces d'entrées/sorties, d'un ou bien plusieurs oscillateurs, d'une sortie VGA, etc. Dans nos implémentations, nous utilisons 2 plateformes de la famille Virtex5 de chez Xilinx : La première plateforme est la carte ML501 [79] conçue autour de la puce FPGA Viretex5-LX50 [80] et la deuxième plateforme est la carte XUPV5 [81] avec un circuit FPGA Virtex5-LX110T [82]. Les deux cartes utilisées ont presque les mêmes caractéristiques et les mêmes composants externes mais avec 2 circuits FPGA différents.

Les FPGA Virtex5 sont des circuits gravés en technologie 65nm, offrent par conséquent une densité supérieure à 12M portes logiques équivalente. Ils peuvent atteindre une fréquence de fonctionnement de 550MHz. La famille Virtex5-LX est utilisée généralement pour des applications hautes performances. En effet, la mention LX signifie que le circuit FPGA est particulièrement optimisé en terme de ressources logiques et de mémoires embarquées. Ce qui rend ces circuits bien adaptés au prototypage d'architectures complexes massivement parallèles telles que les applications de traitement de la vidéo. La formule suivante montre les détails du circuit suivant le nom de la puce FPGA [82] :




## 2.1. La carte Xilinx ML501

ML501 est une plateforme de développement des SoC pour de multiples utilisations. Les caractéristiques de cette carte assurent une solution parfaite pour l'implantation des applications sur FPGA surtout pour les applications qui ont besoin de ressources mémoires généreuses étendues et aussi de plus de flexibilité et d'efficacité.

### 2.1.1. Description des éléments de la carte

Le cœur de la plateforme ML501 est le circuit FPGA de type ***Virtex5 XC5VLX50-1FFG676***, les caractéristiques de ce circuit sont données dans le tableau 4.1, un autre tableau comparatif est donné dans l'annexe A1. La plateforme ML501 dispose aussi d'une mémoire externe (DDR2-SDRAM SODIMM) de capacité 256MB, généralement de type ***Micron MT4HTF3264HY-53E***. En plus du circuit FPGA et de la DDR2, la carte ML501 est une plateforme riche en composants pour l'évaluation et le développement des multiples applications, ces composants permettent un accès facile et pratique aux ressources disponibles dans le circuit FPGA [80]. Les ports de la vidéo, de l'audio et les ports de communication ainsi que les ressources mémoires généreuses donnent la fonctionnalité et la flexibilité à cette plateforme et la rendent une plateforme de développement FPGA typique. La plateforme ML501 dispose de :

1. **Mémoires :** Xilinx Platform Flash XCF32P ; 9Mb ZBT synchronous SRAM ; 32MB Intel P30 StrataFlash ; 2MB SPI Flash ; System ACE™ CompactFlash configuration controller avec un connecteur CompactFlash ; IIC-EEPROM de 8Kbits.
2. **Connectivité :** Port JTAG utilisé avec un câble parallèle(III) ; câble parallèle(IV) ou bien plateforme de téléchargement USB ; Ethernet PHY 10/100/1000 tri-speed ; Codec Stéréo AC97 avec connecteurs line-in/microphone, line-out/headphone, et audio difital SPDIF ; Piezo Speaker ; Port PS/2 pour souri/clavier ; des E/S d'extension ; Port série type RS-232 ; Connecteur vidéo (DVI/VGA).
3. **Composants divers :** Générateur de système d'horloge programmable ; CPLD type XC95144XL de Xilinx ; support d'oscillateur d'horloge (3.3V) avec un oscillateur à 100MHz ; contrôleur de température et de tension ; boutons-poussoirs à usage général ; afficheur LCD de 16 caractères sur 2 lignes ; entrée d'alimentation ; LED d'indication (d'alimentation, d'initialisation et d'activation) ainsi que plusieurs switchers et ports d'E/S.

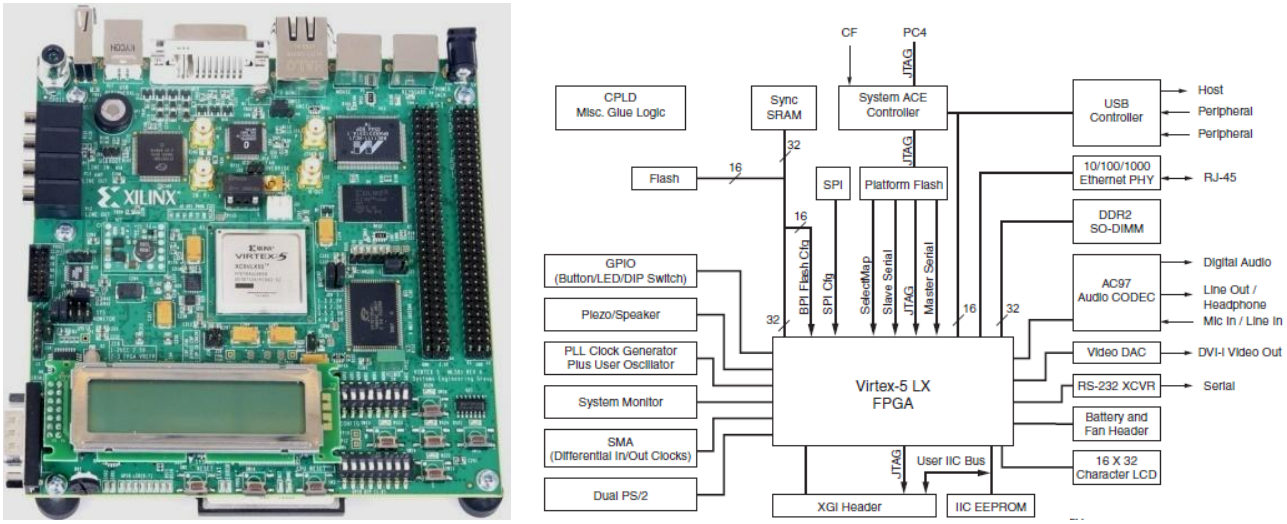



Figure 4.1. La carte Virtex5-ML501 de Xilinx [80].

### 2.1.2. Le circuit FPGA associé

Le circuit FPGA XC5VLX50-1FFG676 est un circuit de la deuxième génération ASMBL (Advanced Silicon Modular Block) de Xilinx. En plus des caractéristiques hautes performances assurées par Xilinx dans des versions précédentes, les nouveaux circuits contiennent plusieurs blocs IP : des BRAM, des FIFOs, des DSPs, interfaces DCM, générateurs d'horloge, etc. La puce LX50 intègre également un bloc de gestion des horloges qui permet une répartition d'horloges parfaite, il dispose d'un maximum de 560 E/S (tableau 4.1) [82]. Lors de la synthèse des implémentations, il est possible de choisir de réaliser les mémoires et les multiplieurs par des LUTs ou bien par des BRAMs et des DSP48E. Le circuit LX50 dispose de 14 Banks avec des impédances contrôlées, la tension d'alimentation de chaque Bank peut varier de 1,8 à 3,3 Volts [83]. Nous pouvons configurer ce circuit à l'aide de plusieurs dispositifs : câble JTAG, System ACE controller, Platform Flash PROM, Linear Flash memory et SPI Flash memory [80].

## 2.2. La carte Xilinx XUPV5

Idéalement conçue pour les recherches et les études de haut-niveau, la plateforme d'évaluation XUPV5 (Figure 4.2) est l'un des outils de développement les plus puissants de la compagnie Xilinx, architecturé autour d'un puisant circuit FPGA de type Virtex5 ***XC5VLX110T-1FFG1136***. XUPV5 est l'une des cartes conçues par Xilinx spécialement pour les universitaires (programme Xilinx-université). Cette plateforme contient les mêmes composants externes que la carte ML501, la seule différence réside dans le circuit FPGA avec plus des ressources et des composants matériels embarqués (tableau 4.1).

## 2.3. Détails des blocs élémentaires dans la Virtex5

Le tableau suivant montre les détails des 2 circuits FPGA (Virtex5-LX50 et Virtex5-LX110T). Pour comparer ces deux cartes avec les autres cartes Virtex5, consulter le tableau dans l'annexe A1.

| **Circuit FPGA** | CLB (Configurable Logic blocks) | | | DSP 48E | Blocs BRAM | | | Total I/O Banks | Max User I/O |
|---|---|---|---|---|---|---|---|---|---|
| | (Row×col) | Slices | Max RAM distr. | | 18 Kb | 36 Kb | Max(Kb) | | |
| **XC5VLX50** | 120 × 30 | 7.200 | 480 Kb | 48 | 96 | 48 | 1.728 | 17 | 560 |
| **XC5VLX110T** | 160 × 54 | 17.280 | 1.120 Kb | 64 | 296 | 148 | 5.328 | 20 | 680 |

Tableau 4.1 : Les différentes caractéristiques des 2 circuits Virtex5 (LX50 et LX110T) [82].

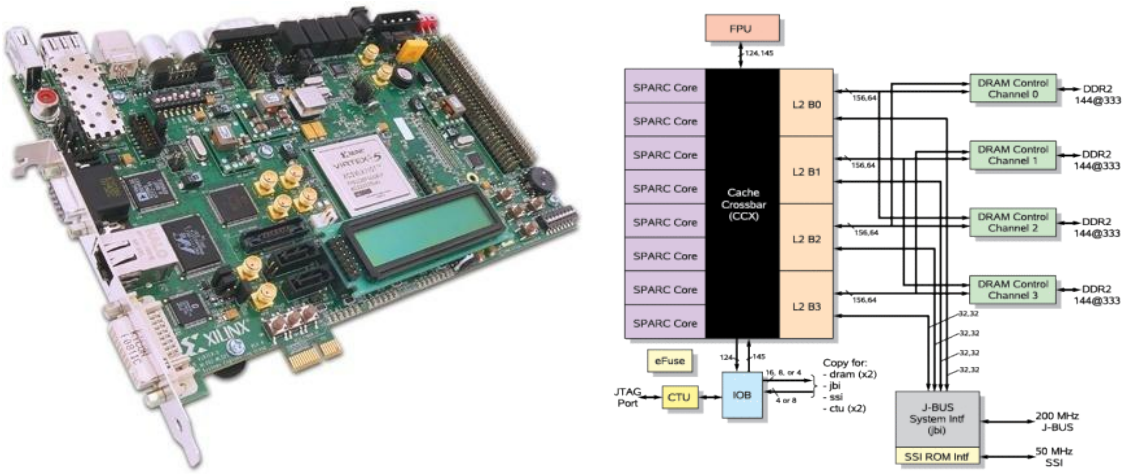



Figure 4.2. La plateforme XUPV5-LX110T de Xilinx [81].

### 2.3.1. Les blocs RAM/FIFO

Les circuits FPGA Virtex5 contiennent des blocs RAM de 36Kbits, chaque bloc peut être configuré soit comme deux indépendantes RAM de 18Kb, ou un seul Bloc de 36Kb. Les Blocs RAM de 36Kb peuvent être configurés comme un 64Kx1 (deux blocs de 36Kb), 32Kx1, 16Kx2, 8Kx4, 4Kx9, 2Kx18, ou encore 1Kx36. Les blocs RAM de 18Kb peuvent être configurés en tant que 16Kx1, 8Kx2, 4Kx4, 2Kx9, ou encore 1Kx18. Les sorties des différentes mémoires sont latchées (sur des Latchs/Registres) afin de donner plus de flexibilité d'utilisation [84].

Comme dans les Virtex4, les opérations de lecture/écriture sont synchrones, les deux ports (A et B) de chaque bloc mémoire sont symétriques et totalement indépendants, ne partageons que les données stockées. Chaque port peut être configuré indépendamment de l'autre dans l'une des largeurs disponibles. En plus, le port en écriture de (A et/ou B) peut être différent du port en lecture.

Les blocs RAM 36Kb peuvent être auto générés par l'outil CORE GENERATOR de Xilinx (les outils Bloc Memory GEN et FIFO GEN), et peuvent être configurés en RAM (suivant les différentes configurations), en ROM ou bien en FIFO. Au moment de la création, les blocs RAM peuvent être organisés en deux types (Synchronous Dual-Port RAM ou bien Single-Port RAM).

### 2.3.2. Les blocs logiques configurables

Les blocs logiques configurables (CLBs) sont les principales ressources logiques d'implémentation matérielle des circuits séquentiels ainsi que des circuits combinatoires. Chaque élément du CLB est connecté à une matrice de commutation pour l'accès à la matrice de routage général. Un élément CLB contient une paire de Slices (tranches), ces deux slices ne sont pas connectés l'un à l'autre, et chaque slice est organisé comme une colonne (Figure 4.3). Chaque slice dans une colonne a une chaîne procédée indépendamment. Pour chaque CLB, les slices dans le fond du CLB sont nommés slice(0), et les slices dans le top du CLB sont nommés slice(1). Généralement, un "X" suivi d'un numéro identifie la position de chaque slice dans un couple et un "Y" suivi d'un numéro indique la position de la colonne du slice (exemple : Slice X2Y1 indique le slice 2 dans la colonne 1).

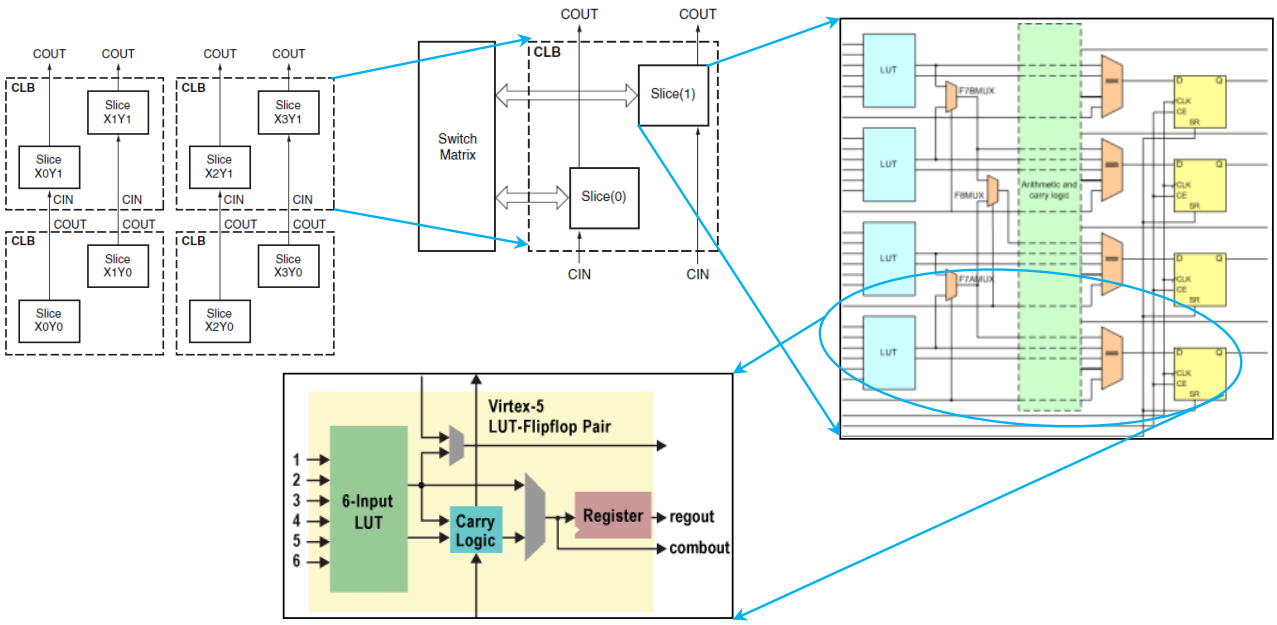



Figure 4.3. Architectures et organisations des slices et des CLBs dans les FPGAs Virtex5.

Chaque slice contient 4 générateurs de fonctions logiques (ou bien look-up tables LUT), 4 éléments de stockage, des multiplexeurs, et carry logique. Ces éléments sont utilisés par les slices destinés à réaliser la logique, l'arithmétique, et les fonctions ROM, ce qui donne le nom (SLICEL). En plus, quelques slices de type (SLICEM) sont utilisés pour réaliser des fonctions supplémentaires de stockage (BRAM) et de décalage. Seuls les SLICEM peuvent être configurés en tant que registre de décalage à 32bits (SRLC32E), deux ou bien plusieurs registres peuvent être montés en cascade pour concevoir des registres à décalage à 64bits, 96bits, 128bits, etc. Les tableaux dans l'annexe A2 montrent le type et le nombre des mémoires RAM distribuées et des mémoires ROM qui peuvent être générées sur la Virtex5-LX50, ainsi que le nombre des LUTs utilisées dans chaque type.

### 2.3.3. Les DSP48E

Un bloc DSP48E est composé d'un multiplieur, à deux opérandes sur 18bits chacun, suivi de multiplexeurs et d'un additionneur/soustracteur sur 48bits (Figure 4.4), il permet de réaliser les opérations de multiplication en un cycle d'horloge. Les blocs DSP48E sont organisés au sein du circuit sous forme de colonnes (Figure 1.2), cette organisation permet de cascader les résultats de plusieurs blocs. Les circuits Virtex5 LX50 et LX110T abritent 48 et 64 blocs DSP48E respectivement.

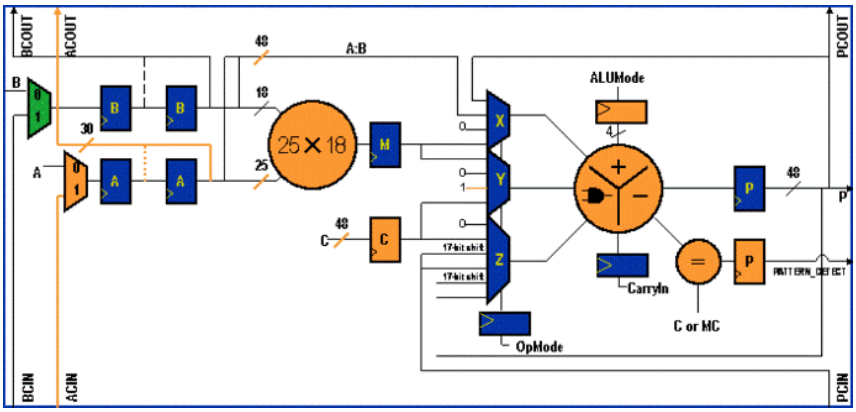



Figure 4.4. Architecture du DSP48E utilisé dans les Virtex5 [84].

## 2.4. La mémoire DDR2 utilisée sur les plateformes Xilinx

La mémoire DDR2-Micron (MT4HTF3264HY-53$^{E}$), utilisée dans les plateformes de Xilinx, est organisée en 4 Banks de données adressées par les deux lignes d'adresse (BA0, BA1), chaque Bank est organisée en une matrice de cellules ($2^{13}$ lignes × $2^{10}$ colonnes), chaque cellule est composée de 64bits (Figure 4.5). Les lignes sont adressées par A0-A12 et les colonnes sont adressées par les mêmes lignes A0-A9, donc la nécessité d'utilisation d'un multiplexeur au niveau du contrôleur DDR2. En effet, le contrôleur reçoit en entrée une adresse logique sous forme Bank&Adresse-ligne&Adresse-colonne sur 25bits, et envoie en sortie, vers la mémoire DDR2, une adresse physique multiplexée des lignes et des colonnes [83].

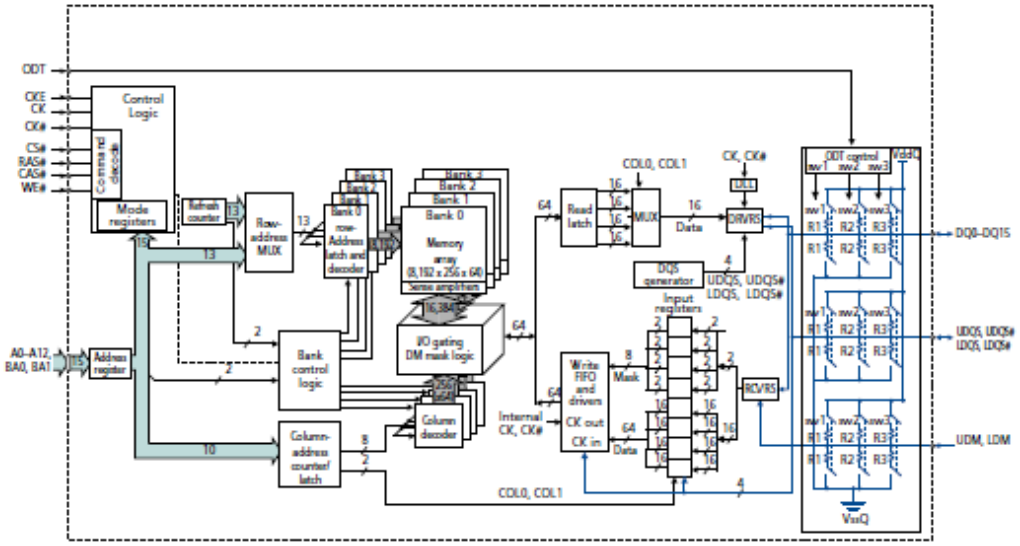



Figure 4.5. Architecture interne de la DDR2 type MT4HTF3264HY-53$^{E}$ [85].

Si on manipule uniquement le signal de luminance (sur 8bits), chaque cellule mémoire dans la DDR2 peut contenir 8pixels. De même, une ligne de mémoire peut contenir 8×$2^{10}$pixels, et une image de 256×256pixels occupe 8lignes de mémoires [85]. A partir de cette architecture, on remarque que la DDR2 favorise la lecture des données en paquet et même plusieurs paquets à la fois en mode Burst.

A l'aide de l'outil MIG2.0 [86] de Xilinx, nous pouvons générer un contrôleur mémoire spécialement pour le modèle MT4HTF3264HY-53$^{E}$. Avec ce contrôleur, et sur demande de l'utilisateur, un modèle Verilog de la mémoire est généré uniquement pour la simulation (non synthétisable). Ce modèle sert à simuler le fonctionnement du contrôleur avant le passage vers l'implantation physique.

## 2.5. Le contrôleur mémoire DDR2

Pour avoir accès à une mémoire externe, quelque soit le type, on utilise un contrôleur mémoire qui sert comme intermédiaire entre le programme utilisateur et la mémoire physique. Le contrôleur sert aussi à effectuer les opérations de commandes, de calculs des adresses physiques à partir des adresses logiques, d'enregistrement des données et des adresses dans des FIFOs pour la lecture/écriture en mode Burst, de rafraichissement automatique, etc. Le programme utilisateur s'occupe uniquement de la partie génération des adresses logiques ainsi que des commandes de lecture/écriture.

### 2.5.1. Génération du contrôleur mémoire

Nous utilisons l'outil MIG2.0 pour la génération d'un contrôleur mémoire DDR2. Cet outil est l'un des CORE-Generator fourni avec ISE9.2i.4. Plusieurs étapes sont nécessaires pour créer le contrôleur mémoire, ces étapes sont bien détaillées dans [86]. L'étape qui semble la plus importante est comment choisir le type exact du modèle de la DDR2. Le modèle utilisé dans notre cas n'existe pas dans la bibliothèque initiale de ISE, mais nous pouvons l'ajouter comme un nouvel élément suivant les caractéristiques fournies dans [85] et [83]. A la fin de l'opération de génération du contrôleur, un dossier (mig) est généré dans le répertoire de travail, ce dossier contient en lui-même plusieurs autres dossiers avec des documentations, des exemples de simulation et de synthèse, ainsi que les fichiers VHDL du contrôleur mémoire. La figure suivante montre le contrôleur mémoire généré avec un DCM (Digital Clock Manager).

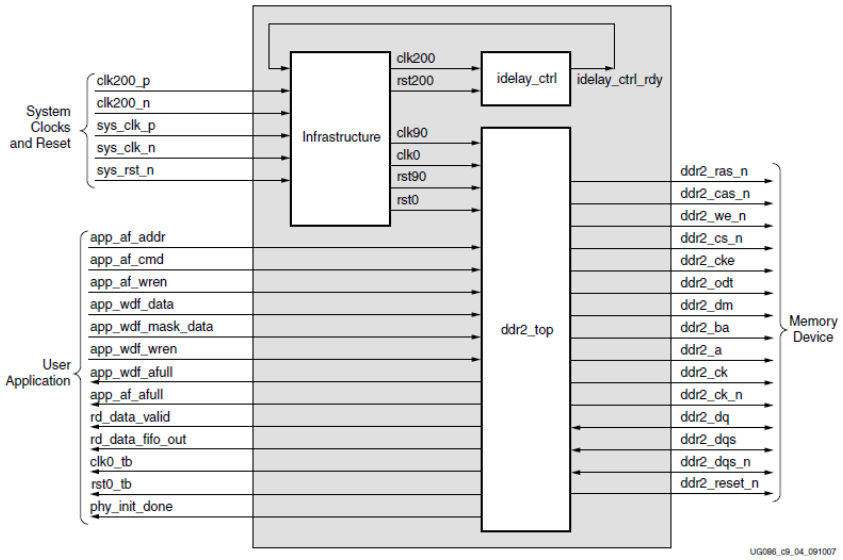



Figure 4.6. Entrées/Sorties du contrôleur mémoire avec DCM [86].

### 2.5.2. Communication contrôleur-mémoire

L'utilisateur logique (la partie intelligente dans notre cas) communique avec le contrôleur de mémoire par l'intermédiaire d'une FIFO à base d'une interface utilisateur. Cette interface se compose de trois bus connexes :

- Un bus de commande/adresse de type FIFO, ce bus accepte les commandes de lecture/écriture ainsi que l'adresse mémoire correspondante ;
- Un bus des données en écriture de type FIFO qui accepte l'écriture des données lorsqu'une commande d'écriture est envoyée sur le bus de commande ;
- Un bus de lecture de type FIFO, sur lequel les données sont lues avec une commande de validation de lecture, vers les blocs de traitement.

### 2.5.3. Enregistrement d'une image dans la DDR2

La lecture des macroblocs à partir d'une mémoire DDR2 nécessite des connaissances préalables de l'organisation des pixels dans la mémoire, de la taille des bus des données et des adresses, des modes de lecture et d'écriture, ainsi que le nombre de données lues à chaque fois, etc. La plus petite cellule de la DDR2 est de 64bits, cela est équivalent à 8 pixels du signal de luminance. En plus les opérations de

lecture/écriture s'effectuent avec deux cellules à la fois en mode normal (mot de 128bits), et 4 mots ou bien 8 mots de 128bits en mode Burst.

L'architecture d'une DDR2 ne favorise pas la lecture des bits et des pixels séparément, mais elle favorise la lecture en paquet (au minimum la lecture de plusieurs mots de 128bits ce qui représente la taille du bus de données). Cette structure favorise, donc, la lecture séquentielle des données, la lecture aléatoire est par contre très couteuse de point de vue temps d'accès. Dans le contrôleur mémoire proposé pour l'encodeur H.264, nous allons utiliser cette caractéristique pour lire un macrobloc ou bien une ligne de macroblocs à partir de la DDR2 dans une autre mémoire locale, et cela dans le but de limiter l'accès à la mémoire externe.

### 2.5.4. Simulation du contrôleur généré

L'idée du contrôleur H.264 consiste à ajouter une couche, à l'entrée du contrôleur DDR2, pour la gestion de la mémoire spécialement pour l'encodeur H.264. Mais avant d'ajouter cette couche, nous avons testé le fonctionnement du contrôleur mémoire en écrivant et en lisant des images dans la DDR2. Pour la simulation, nous avons utilisé ModelSim6.1 avec un testbench qui permet l'alimentation du contrôleur par des adresses et des données de test, et cela pour les deux modes de lecture et d'écriture. Le fichier doit aussi vérifier l'état du contrôleur avant chaque opération. Plusieurs exemples de simulation ont été validés pour tester le fonctionnement correct du contrôleur avec la DDR2 et pour calculer les temps de réponse de chaque composant. Dans l'un des exemples de simulation, nous avons choisi le bus d'adresses comme étant un compteur incrémenté par 8 à chaque fois (mode Burst = 4), le bus de données prend des valeurs sur 128bits (des pixels d'une image par exemple). La figure 4.7 représente une simulation de quelques signaux du contrôleur.

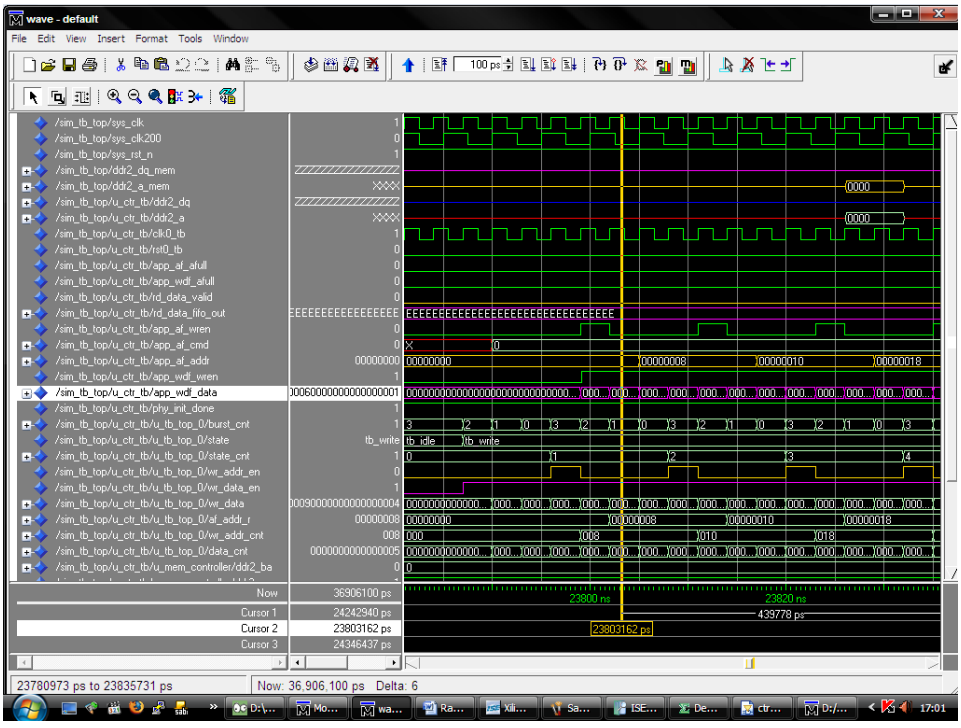



Figure 4.7. Simulation du fonctionnement du contrôleur DDR2 (Burst = 4).

Nous pouvons comparer nos résultats avec les résultats synthétiques fournis dans [86] selon les deux modes de lecture et d'écriture. Exemple en mode d'écriture (Figure 4.8), le temps de réponse de la

mémoire, y compris le contrôleur, est équivalent à 19 cycles d'horloge (Latence). Après l'écriture d'un premier mot de 128bits (64bits sur le front montant et 64bits sur le front descendant), les autres mots sont écrits en chaque cycle d'horloge (Cadence). Les mêmes mesures sont vérifiées pour le mode de lecture.

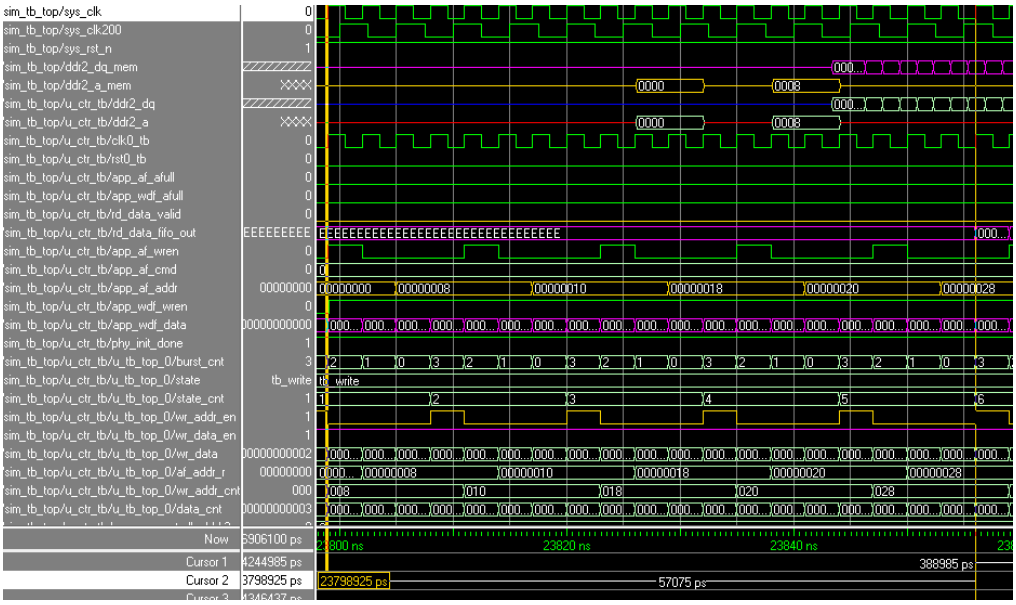



Figure 4.8. Temps de réponse global du contrôleur en mode d'écriture.

### 3. Gestion de mémoire pour l'encodeur H.264

Généralement, pour spécifier une architecture matérielle reconfigurable d'une application de traitement de vidéo, il faut définir auparavant la plateforme de travail constituée d'un capteur, d'une plateforme à base de FPGA et d'un organe d'affichage. Le circuit FPGA constitue l'organe de base de l'implantation, car c'est suivant leurs caractéristiques que les différents composants matériels vont être réalisés. La figure 4.9 montre un exemple d'une plateforme de traitement de vidéo, cette plateforme permet d'assurer la gestion des entrées/sorties dans le cas d'une implémentation matérielle d'un encodeur. Les images sont initialement captées, ensuite stockées dans la mémoire (à cause de la dépendance des traitements), traitées par les modules de l'encodeur et enfin affichées sur un écran ou bien stockés dans une autre mémoire. Les traitements sont généralement appliqués sur des blocs et des macroblocs dans les images enregistrées.

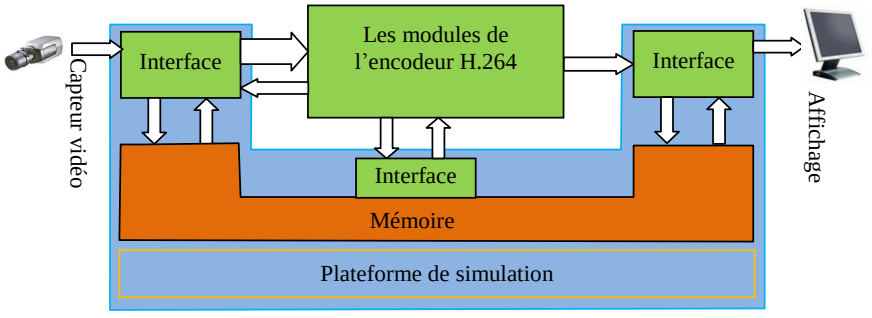



Figure 4.9. Plateforme de traitement vidéo pour l'encodeur H.264.

Pour une bonne gestion de la mémoire, plusieurs paramètres doivent être pris en compte : le type et la capacité de la mémoire utilisée, le temps lié à la latence, la cadence, etc. Aujourd'hui, sur la même

carte de prototypage à base de FPGA, nous disposons de plusieurs types de mémoires : nous trouvons des mémoires externes de type DDR2/DDR3, des mémoires flash ou bien des mémoires de type SRAM, mais aussi des mémoires embarquées (on chip RAM ou bien BRAM). Dans les récents circuits FPGA de la famille virtex5 de Xilinx par exemple, les mémoires BRAM de 36 Kbits sont reconfigurables, et offrent la possibilité de définir la taille des mots (8, 16, 32 et 64bits), la direction des données et le type de la mémoire, etc. Dans ce cas, les mémoires sont données en Kbits et il appartient à l'utilisateur de les configurer selon les besoins de l'application à implémenter.

Dans un encodeur H.264, le traitement d'une image peut dépendre d'une ou bien plusieurs images dans la même séquence, ce qui rend nécessaire l'enregistrement de plusieurs images à la fois. Dans les circuits FPGA actuels, la mémoire interne est généralement faible pour l'enregistrement de plusieurs images de la séquence vidéo, il est donc nécessaire d'utiliser l'une des mémoires externes (généralement la mémoire DDR). La capacité de ces mémoires est généralement assez grande pour enregistrer un groupe d'images (GOP) quelque soit le format des images manipulées. Les mémoires internes, par contre peuvent servir à enregistrer les macroblocs en cours de traitement.

En plus, la dépendance inter-bloc dans les modules de l'encodeur H264 nous empêche d'envisager plus de parallélisme sans surcoût matériel significatif. Pour l'implémentation matérielle des modules de traitement de l'encodeur H.264, nous proposons dans le chapitre 5 des architectures basées sur les principes de parallélisme de données et de pipeline. En effet, nous avons exploité les caractéristiques, de la DDR2 et des mémoires internes des circuits FPGA, pour proposer des architectures avec des entrées/sorties sur 128bits (bloc de 4×4 pixels). Dans ce chapitre, nous proposons un contrôleur mémoire capable d'assurer ces données (128bits) spécialement pour l'encodeur H.264.

### 3.1. Les besoins en mémoire de l'encodeur H.264 et les deux solutions possibles

La figure 4.10 schématise les modules de l'encodeur H.264 (le codage Intra, le codage Inter et le filtre anti-blocs) qui utilisent d'une façon directe la mémoire externe ainsi que leurs besoins en mémoire interne (blocs et macroblocs). L'idée de la gestion de mémoire consiste à récupérer les macroblocs, en cours de traitement, à partir de la mémoire externe, et de les stocker dans la mémoire interne (les préparer pour les modules de traitement), et cela pour limiter les accès à la DDR2. Les données sont ensuite transférées, selon la demande des modules de traitement, bloc par bloc.

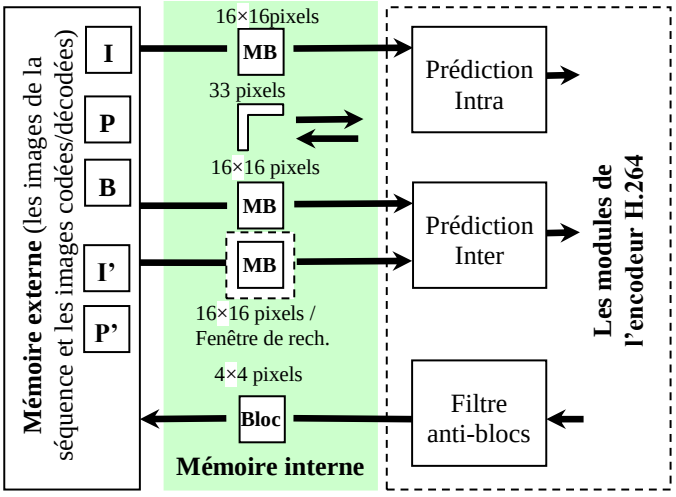



Figure 4.10. Les modules de l'encodeur H.264 en liaison directe avec la mémoire.

Pour assurer des entrées/sorties de 128bits aux modules de l'encodeur H.264 implémentés en matériels, nous avons exploité 2 solutions possibles. Dans la première solution, les modules de l'encodeur sont utilisés dans une architecture à base de processeur. Afin d'adapter nos accélérateurs matériels (128bits) au reste du système à travers le bus (le bus PLB par exemple), nous avons utilisé des mémoires tampon type FIFO. Ces FIFOs ont des entrées sur 32bits et des sorties sur 128bits. Elles servent à collecter les données avant leurs transferts vers les modules de traitement [87]. La figure 4.11 montre un exemple d'adaptation des modules de l'encodeur H.264 avec le bus PLB.

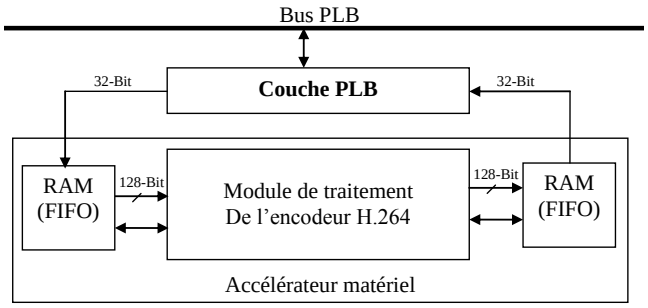



Figure 4.11. Adaptation des modules de l'encodeur H.264 avec le bus PLB.

Dans une telle implémentation, le contrôleur mémoire DDR2 est intégré dans un schéma à base de processeur (MicroBlaze dans notre cas), ce processeur peut lire et écrire des mots sur 8bits (ou bien 32bits) avec des adresses logiques variant entre 0x90000000 et 0x9fffffff. C'est au contrôleur mémoire de générer l'adresse physique pour récupérer les données à partir de la DDR2.

Dans la deuxième solution, les modules de l'encodeur réalisé en matériel (accélérateurs) sont reliés avec la mémoire externe à travers un contrôleur mémoire spécialement conçu pour l'encodeur H264. Nous avons proposé un contrôleur mémoire qui joue le rôle de contrôleur de caches de données pour les différents modules de traitement [88]. L'idée, dans ce cas, est d'exploiter au mieux la largeur des données (128bits) à la sortie des mémoires DDR2 en évitant le bus de 32bits. Cet organe d'adressage est constitué d'un contrôleur mémoire DDR2 et d'une partie intelligente chargée des calculs d'adresses en prédiction des besoins de chacun des modules de l'encodeur H264. Ceci permet aussi de décharger l'encodeur de la partie adressage. La figure 4.12 montre le schéma de principe du contrôleur proposé.

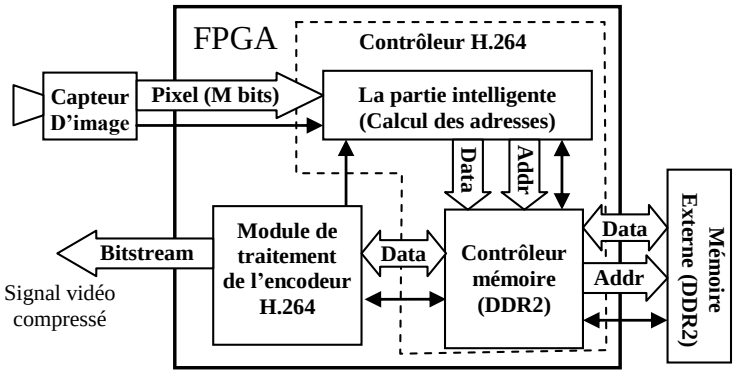



Figure 4.12. Principe du contrôleur mémoire pour l'encodeur H.264.

### 3.2. Historique des implantations de la gestion de mémoire pour H.264

Dans la littérature, de nombreuses architectures et implémentations matérielles sont proposées pour les différents modules de l'encodeur H.264/AVC. Plusieurs autres travaux sont consacrés pour

l'implémentation du décodeur, et cela pour des applications diverses. Il existe une variété de résultats liés à la conception et l'implémentation des codecs H.264, mais il y a seulement quelques travaux qui parlent de la gestion de mémoire et des entrées/sorties des modules de traitement. Cette partie est nécessaire pour obtenir une implémentation matérielle complète de l'encodeur sur une plateforme reconfigurable. Les auteurs proposent, généralement, l'utilisation des bus de données (32bits) fournis avec les outils de conception.

Dans [36][89][90] les auteurs proposent des méthodes de gestion de mémoire spécialement pour le décodeur H.264, la gestion de la mémoire est limitée à l'enregistrement des images de référence nécessaires pour la reconstruction des autres images dans la séquence. Au niveau du décodeur l'accès à la mémoire est séquentiel et limité à la lecture des macroblocs (le décodeur ne contient pas les modules de prédictions Intra et Inter). Ce n'est pas le même cas dans l'encodeur H.264 avec des macroblocs chevauchés et peuvent être calculés à partir de plusieurs images. Dans [91] les auteurs proposent des mémoires multiples (12 mémoires single port) pour charger les macroblocs vers le module d'estimation de mouvement de l'encodeur H.264, mais la méthode de lecture et d'écriture de ces mémoires n'est pas mentionnée. Les auteurs dans [92] ont présenté une technique d'implémentation pour optimiser l'encodeur H.264 sur une architecture multiprocesseur symétrique embarquée. Enfin, les auteurs dans [93] ont proposé une implémentation complète de l'encodeur H.264 en utilisant une architecture en pipeline, où tous les modules ont été optimisés. Dans ces deux derniers travaux, les auteurs ont proposé une nouvelle technique pour optimiser la gestion de la mémoire, et cela pour exploiter le sous-système de mémoire sur une plateforme embarquée, mais les caractéristiques, l'architecture de la plateforme et la méthode de gestion de mémoire ne sont pas fournis.

### 3.3. Contrôleur mémoire proposé pour l'encodeur H.264

L'utilisation d'une mémoire externe, pour l'enregistrement des images de la séquence, nécessite un organe intelligent pour assurer la lecture à temps des données nécessaires aux modules de traitement ainsi que la récupération des données traitées. Cet organe d'adressage est constitué d'un contrôleur mémoire DDR2, fourni suivant le type de la mémoire utilisée, et d'une partie intelligente chargée des calculs des adresses nécessaires à la lecture et l'écriture des données, ainsi que la synchronisation des différents accès à la mémoire. Dans la partie intelligente, on doit exploiter aussi, toute les informations concernant la norme H.264 ainsi que l'organisation des données à l'intérieure de la DDR2, ceci est dans le but d'assurer les données suivant un ordre bien défini. Dans la figure 4.13, nous proposons l'architecture du contrôleur mémoire intelligent spécialement conçu pour la norme H.264. Dans cette figure, la séparation de la partie intelligente et la partie contrôleur est importante, dans le but de rendre le contrôleur adapté à plusieurs types de mémoire. En effet, pour adapter l'implantation matérielle à une mémoire bien définie, via le contrôleur H.264, il suffit de changer le contrôleur fourni avec cette mémoire et de modifier la partie qui calcule les adresses suivant la structure interne de cette nouvelle mémoire.

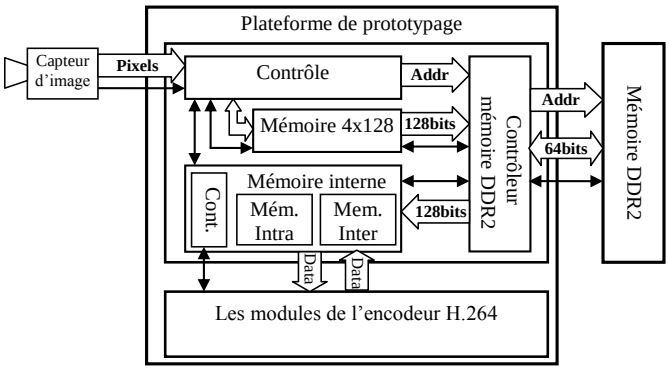



Figure 4.13. Le contrôleur mémoire proposé pour l'encodeur H.264.

Cette architecture nous permet aussi de fixer et d'assurer la taille des bus de données pour une implémentation matérielle de l'encodeur, et cela suivant les besoins de chaque unité de traitement, et suivant le parallélisme choisi. La partie intelligente permet aussi de collecter les pixels, captés à partir du capteur d'image (les pixels d'entrée) ou bien à partir du filtre anti-blocs (les blocs filtrés), pour former des paquets de données, ensuite le transfert de ces paquets en mode Burst, ce qui limite le nombre des accès à la mémoire.

### 3.3.1. Description fonctionnelle du contrôleur

La figure précédente montre l'utilisation de 2 types de mémoire (interne et externe), la mémoire externe consiste à enregistrer les images de la séquence, tandis que la mémoire interne est réservée uniquement pour l'enregistrement des macroblocs en cours de traitement.

- Initialement les pixels d'entrée (sur 8bits ou bien 32bits) sont collectés, dans une mémoire interne, par paquet de 128bits (4 paquets en mode Burst4) ;
- Les paquets collectés sont transmis ensuite vers la mémoire externe, les adresses et les signaux de commande sont assurés par l'unité de contrôle. Ces deux premières étapes sont répétées périodiquement et sans arrêt avec l'écoulement des pixels à l'entrée ;
- La partie lecture est constituée d'une mémoire locale (pour les macroblocs en cours de traitement) et d'une partie de contrôle pour assurer la synchronisation entre l'écriture et la lecture des pixels ;
- La partie de contrôle est constituée d'un module intelligent de lecture pour le rechargement de la mémoire interne à partir de la mémoire externe par les macroblocs. Le calcul des adresses et des signaux de commande pour la DDR2 est assuré par cette partie ;
- Les modules de traitement prend périodiquement les données disponibles dans les mémoires internes, et s'occupent uniquement des traitements et non pas des opérations d'adressage.

### 3.3.2. Lecture des données pour le module de prédiction Intra

Les données d'entrée, pour le module de codage Intra, sont les pixels d'un macrobloc, chargés dans un ordre bien défini (bloc par bloc), plus un ensemble de pixels de voisinage extrait des blocs déjà traités mais non filtrés comme l'indique le logiciel de référence de H.264 [46]. Les macroblocs dans l'image de référence sont chargés à partir de la mémoire externe dans une mémoire interne (16×16pixels), et cela dans un ordre séquentiel de gauche à droite et de haut en bas. Le même ordre est appliqué pour la lecture des 16 blocs dans le même macrobloc, et cela à partir de la mémoire locale vers le module de

traitement. Le module de contrôle consiste à gérer les partitions du macrobloc ainsi que l'ordre de transfert des blocs vers le module de prédiction Intra.

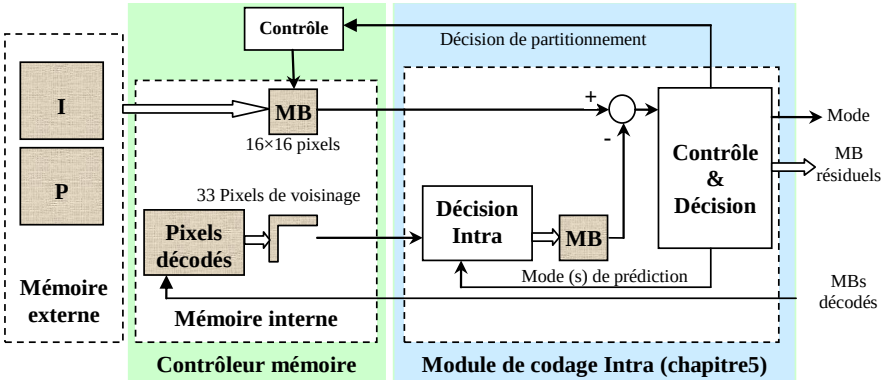



Figure 4.14. Gestion de mémoire pour le module de codage Intra.

Dans le contrôleur mémoire H.264 proposé, une autre proposition de gestion de mémoire à été vérifiée, cette proposition consiste à lire 16 lignes d'image à la fois (chaque ligne avec M pixels). Cette solution est dans le but d'avoir une lecture séquentielle des pixels des images de référence (en mode Burst) avec l'inconvénient de la capacité importante de la mémoire interne utilisée. En effet, cette mémoire doit contenir M/16 macroblocs à la fois au lieu d'un seul macrobloc dans la première proposition. Pour basculer à cette deuxième solution, il suffit de changer la méthode de calcul des adresses dans la partie de contrôle de la figure 4.14 ainsi que la capacité de la mémoire interne.

Dans les deux cas, deux autres mémoires interne sont utilisées pour l'enregistrement des pixels de voisinage, une mémoire de taille (M+1)pixels pour l'enregistrement des pixels de voisinage-haut et une mémoire 4pixels (ou bien 16pixels) pour l'enregistrement des pixels de voisinages-gauche. Ces 2 mémoires de voisinage vont être réutilisées par la suite pour l'implémentation matérielle du module de prédiction Intra dans le chapitre 5. Pour la prédiction Intra4×4 par exemple, la figure 4.15 montre la méthode de lecture des blocs et la désignation des pixels de voisinage. Les mémoires de voisinage sont remplis de la méthode suivante :

- La mémoire M+1 pixels est initialisée par des 127 pour la première ligne des macroblocs, à la fin du traitement de ces derniers la mémoire M+1 pixels est remplie par la dernière ligne des pixels, pour servir comme voisinage-haut aux macroblocs de la ligne suivante.
- La mémoire 4 pixels est initialisée par des 127 au début de traitement de chaque ligne de macroblocs, à la fin du traitement d'un bloc cette mémoire est remplie par les quatre pixels à droite de ce dernier, pour servir comme voisinage-gauche au bloc suivant dans la même ligne.

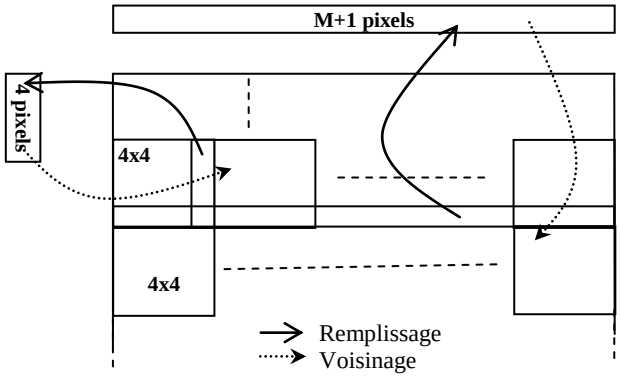



Figure 4.15. Désignation des blocs Intra4×4 et leurs pixels de voisinage.

### 3.3.3. Lecture des données pour le module de prédiction Inter

Les données d'entrée, pour le module de codage Inter, sont deux macroblocs chargés à partir de la mémoire externe. Le premier (P-MB) est chargé à partir de l'image prédite en cours de traitement et le deuxième (R-MB) à partir d'une fenêtre de recherche dans l'image de référence. Cette dernière fenêtre est estimée autour de la position du macrobloc en cours de traitement. Pour limiter le nombre des accès à la mémoire, dans notre implémentation, le P-MB et la fenêtre de recherche sont stockés dans une mémoire interne avant leurs transferts vers le module de traitement bloc par bloc. Les P-MB sont chargés à partir de la mémoire externe dans un ordre séquentiel de gauche à droite et de haut en bas. Par contre la fenêtre de recherche, pour chaque P-MB, est extraite à la fois de la mémoire externe et la fenêtre de recherche précédente. C'est la raison pour laquelle nous avons choisi de stocker toute cette fenêtre dans la mémoire interne, la sélection de R-MB dans cette fenêtre sera plus facile selon l'algorithme de recherche choisie pour l'estimation de mouvement. En effet, la norme H.264 n'exige pas une méthode fixe pour cette étape, elle exige par contre la manière de présenter les résultats.

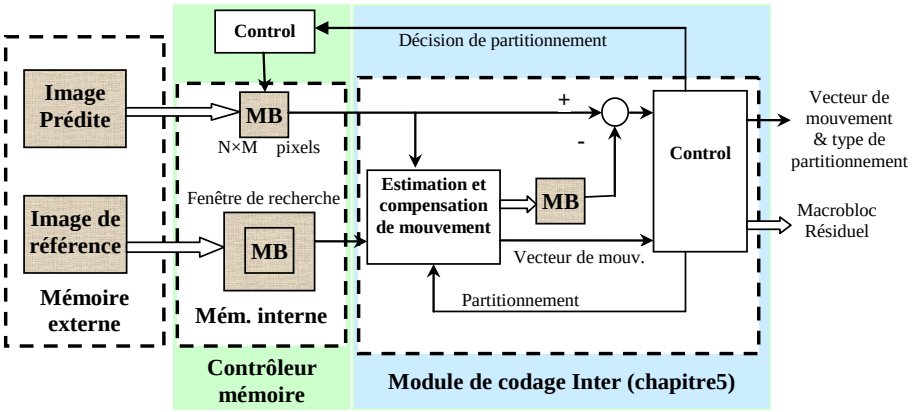



Figure 4.16. Gestion de mémoire pour le module de codage Inter.

Le module de contrôle, dans ce cas, consiste à gérer les partitions du macrobloc (de 4×4 jusqu'au 16×16) ainsi que l'ordre de transfert de ces partitions vers le module de prédiction Inter. Comme dans la prédiction Intra, nous pouvons lire en une seule fois les 16 lignes de l'image prédite et les 16 lignes de l'image de référence avec l'inconvénient de la capacité mémoire interne importante.

### 3.3.4. Enregistrement des données traitées par le filtre anti-blocs

Dans un encodeur H.264, les macroblocs résiduels codés et décodés, par la TQ et $TQ^{-1}$, sont ensuite transférés à l'entrée du filtre anti-blocs dans l'ordre B1-B16 (Figure 4.17). A la sortie du filtre, les blocs sont transférés à la mémoire DDR2 (à travers le contrôleur mémoire) dans l'ordre suivant : V1, H1, H2, H3, H4, V2, B1, B2, B3, V3, B5-B11. A la fin du traitement, les autres blocs seront enregistrés comme suit : les Blocks B4, B8, B12 et B16 sont stockés dans la mémoire de voisinage-vertical pour servir comme voisinage-gauche aux blocs suivant dans la même ligne, et les Blocs V4, B13, B14, B15 et B16 sont stockés dans la mémoire de voisinage-horizontal pour servir comme voisinage-haut aux macroblocs en dessous (voir chapitre5). La même stratégie est appliquée au signal de luminance qu'au signal de chrominance.

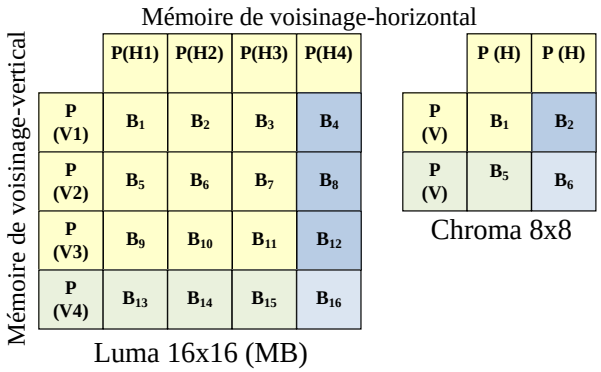



Figure 4.17. Traitement des blocs par le filtre anti-blocs.

La gestion de la mémoire pour le filtre consiste à récupérer les blocs filtrés et le calcul d'une adresse correspondante dans la DDR2. En effet, les images filtrées sont enregistrées séparées dans la mémoire externe pour être utilisées par la suite comme images de références pour les autres images dans la même séquence. L'enregistrement des blocs peut être réalisé en mode Burst en utilisant les FIFO d'écriture du contrôleur mémoire.

### 3.4. Description architecturale du contrôleur

La figure 4.17 montre l'architecture logicielle choisie pour l'implémentation matérielle de l'encodeur H.264. Cette architecture est composée principalement d'un fichier de simulation (sim_tb_top), qui contient à son tour deux grandes composantes :

1. Une partie uniquement pour la simulation, composée de deux modèles de mémoire non synthétisables (mémoire image et mémoire DDR2) pour l'enregistrement des images de la séquence ;
2. Une deuxième partie qui représente l'implémentation synthétisable de l'encodeur H.264. Cette partie est composée de deux grandes parties :
   - Une partie contrôleur mémoire intelligent qui permet l'enregistrement des images dans la DDR2, ensuite la lecture des macroblocs objet de traitement à partir de cette mémoire dans une autre mémoire locale ;
   - Une deuxième partie qui représente l'implémentation matérielle des différents modules de traitement de l'encodeur H.264.

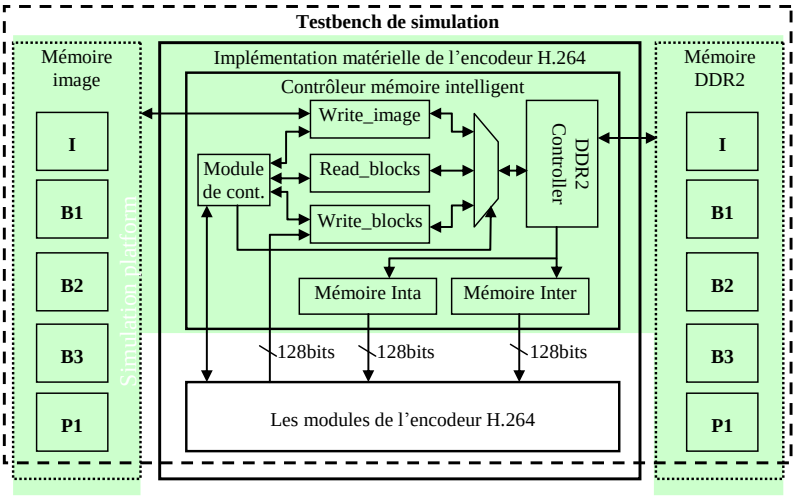



Figure 4.18. Architecture matérielle du contrôleur mémoire H.264.

La première partie avec le contrôleur intelligent constitue une plateforme de simulation de l'encodeur H.264. Cette plateforme permet la simulation des différents modules de l'encodeur par chargement de différentes données nécessaires pour le traitement. La partie contrôleur intelligent est constituée de plusieurs composants :

- Un contrôleur DDR2 (généré par l'outil MIG) pour assurer la communication avec la mémoire externe ;
- Un module write_image qui permet la lecture des pixels en entrée, les collectés dans une mémoire locale (4×128bits), ensuite les transférer à la DDR2 (mode Burst) à travers le contrôleur mémoire ;
- Un module read_blocs chargé de calculs des adresses et des signaux de commande pour la lecture des macroblocs en cours de traitement dans une mémoire interne. La lecture des macroblocs est effectuée suivant la demande des modules de l'encodeur et le module de contrôle ;
- Un module write_blocs pour l'enregistrement des macroblocs déjà traités et filtrés avant leurs transferts à la DDR2 en mode Burst ;
- Un module de contrôle pour synchroniser les différentes opérations ainsi que la synchronisation des accès à la mémoire ;
- Une mémoire locale pour l'enregistrement des macroblocs pour les modules Intra et Inter, cette mémoire constitue le point d'alimentation ces modules par les blocs (4×4pixels).

### 3.5. Résultats d'implémentation matérielle du contrôleur

L'architecture matérielle proposée pour le contrôleur mémoire intelligent a été décrite en langage VHDL. Nous avons utilisé ModelSim6.1 pour la simulation et Xilinx-ISE9.2 pour la synthèse. Pour la simulation, un fichier testbench a été écrit spécialement pour remplacer la partie d'acquisition d'images. Le fichier doit fournir les pixels de l'image à une fréquence (fpixels), calculé selon le format de la séquence vidéo d'entrée (le format CIF par exemple avec 352×288 pixels/image, 10 images/seconde). A l'entrée du contrôleur, les pixels sont collectés par groupes de 16 (128bits) à la fréquence fpixels, ensuite ils sont envoyés par groupe de 4 (en mode Burst) vers la mémoire DDR2 à travers le contrôleur mémoire à la fréquence du circuit FPGA. La figure 4.19 montre le timing du transfert des pixels d'une image vers la mémoire DDR2.

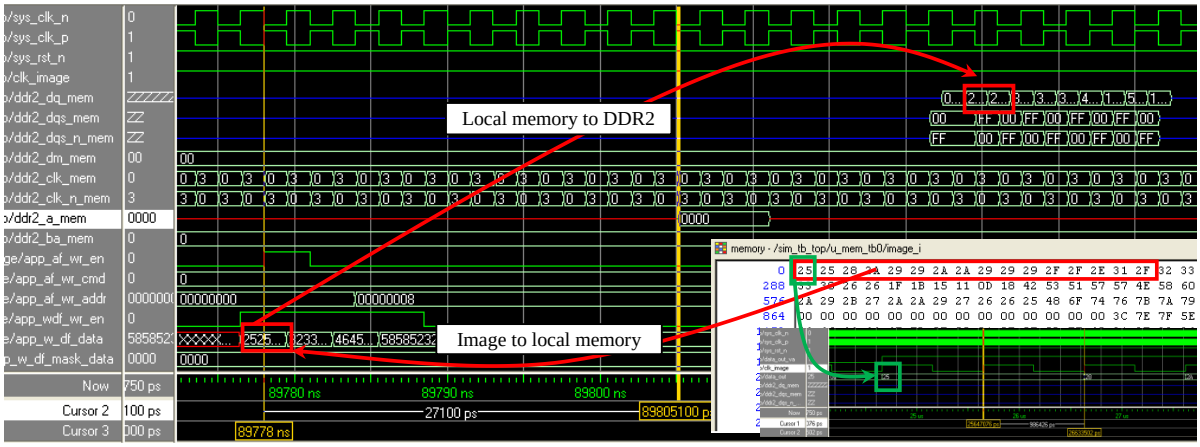



Figure 4.19. Collection et écriture des pixels dans la mémoire DDR2.

La lecture de la mémoire externe vers la mémoire interne est effectuée périodiquement, selon les besoins des modules de traitement de H.264. Le module de codage Intra4×4, par exemple, a besoin

pour chaque prédiction d'un bloc (4×4 pixels) et de 13 pixels de voisinage. Le bloc est chargé directement depuis la mémoire externe à la mémoire interne et les pixels de voisinage sont réutilisés à partir des calculs précédents. A la fin de chaque traitement d'un bloc, des pixels sont extraits pour servir comme voisinage pour des traitements futurs. La figure 4.20 montre la désignation et la lecture d'un bloc et ses pixels voisins. Il y a une similitude avec les autres modes de prédiction Intra et Inter ainsi que pour le filtre anti-blocs.

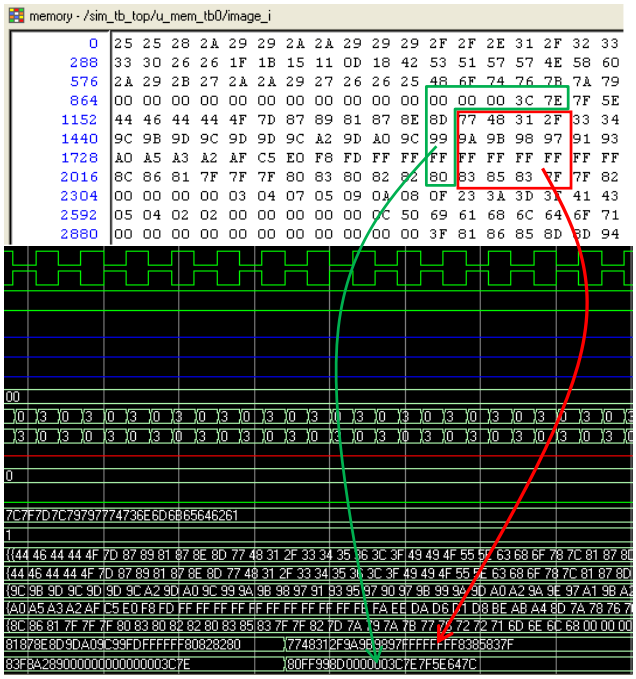



Figure 4.20. Gestion des données pour le module de prédiction Intra4×4.

Dans le tableau 4.2 nous montrons une estimation de la capacité des mémoires interne et externe utilisées dans le contrôleur H.264 avec le pourcentage par rapport à la capacité du mémoire externe DDR2 et la capacité du mémoire interne de la plateforme ML501. Les résultats sont donnés pour les deux profils de la norme H.264 (de base et principale) et pour deux types de séquence d'images (CIF et HDTV).

| **Profil** | | **Mémoire DDR2 utilisée** | | **Mémoire locale utilisée** | |
|---|---|---|---|---|---|
| | | *Image CIF* | *Image HDTV* | *Image CIF* | *Image HDTV* |
| **Baseline** | *Capacité* | 297 Kb | 2.637 Mb | 101.875 Kb | 240.25 Kb |
| | *Pourcentage* | 0.113% | 1.03% | 5.89 % | 13.90 % |
| **Main** | *Capacité* | 1.16 Mb | 10.747 Mb | 101.875 Kb | 240.25 Kb |
| | *Pourcentage* | 0.453% | 4.12% | 5.89 % | 13.90 % |

Tableau 4.2 : Estimation des mémoires interne et externe utilisées dans le contrôleur H.264.

Le tableau 4.3 montre les résultats de synthèse de l'implantation du contrôleur intelligent en utilisant les 2 plateformes ML501 et XUPV5. Pour la première plateforme, le taux d'utilisation des ressources est autours de 10% et 4% pour la deuxième, ce qui donne la possibilité de l'implémentation des modules de l'encodeur sur le reste de la surface du circuit FPGA. En plus des Luts, 6 BRAM (12%) sont utilisés pour assurer les l'implantation des différentes mémoires locales dans le cas de ML501 et 9% dans le cas de XUPV5. L'implémentation peut fonctionner à une fréquence maximale de 114 MHz et 155 MHz sur les deux plateformes respectivement.

| | *Virtex5 LX50* | | | *Virtex5 LX110T* | | |
|---|---|---|---|---|---|---|
| | ***Disponible*** | ***Utilisé*** | ***%*** | ***Disponible*** | ***Utilisé*** | ***%*** |
| ***Slice Registers*** | 28 800 | 2 860 | 9 % | 69 120 | 2879 | 4% |
| ***Slice LUTs*** | 28 800 | 3 052 | 10 % | 69 120 | 2894 | 4% |
| ***Bloc BRAM/FIFO*** | 48 | 6 | 12 % | 148 | 6 | 4% |

Tableau 4.3 : Résultats de synthèse du contrôleur mémoire proposé.

## 4. Conclusion et perspective

Dans cette partie de la thèse, nous avons proposé un contrôleur mémoire intelligent conçu spécialement pour l'encodeur H.264/AVC, et cela dans le but de construire une plateforme de simulation en temps réel. Le contrôleur proposé est important pour la gestion des entrées et des sorties pour les différents modules de traitement suivant l'ordre défini dans le logiciel de référence. Le contrôleur est capable d'assurer les données d'entrée pour les différents modules de l'encodeur, il est capable aussi d'enregistrer les pixels, captés à partir d'une source vidéo ainsi que les blocs traités par le filtre, dans la mémoire externe. L'idée de base du contrôleur consiste à exploiter les caractéristiques des mémoires internes et externes des plateformes de prototypage pour fournir des données sur 128 bits (bloc 4×4 pixels qui représentent la taille idéale pour le parallélisme vue la dépendance inter-bloc dans H.264). Des résultats de simulation et de synthèse ont été présentés en utilisant deux plateformes de prototypage. La plateforme de simulation proposée avec le contrôleur intelligent peut être utilisée dans de nombreuses implémentations temps réel du codec H.264.

Après la conception du contrôleur mémoire qui doit être utilisé avec les différentes implémentations du chapitre 5, nous avons eu l'idée pour le rendre plus dynamique. En effet, l'inconvénient majeur de cette implémentation est qu'elle est statique, le contrôleur mémoire est spécialement conçu sur mesure pour une application H.264, et la partie de calcul des adresses dans le contrôleur mémoire est figé :

- Le contrôleur mémoire est réalisé pour un seul type de DDR2, et par conséquent pour une seule plateforme de prototypage ;
- La taille des images dans la vidéo d'entrée est considérée fixe malgré qu'elle soit déclarée comme données génériques dans l'implémentation proposée ;
- Les besoins en mémoire des différents modules sont calculés pour une seule version de H.264 (même pour un seul profil de la version). En effet, l'organisation des images dans la DDR2 et leurs ordres de lecture/écriture changent d'une version à une autre de H.264/AVC.

Si on change l'un de ces paramètres (type de DDR2, taille des images, version de l'application ou encore l'application en elle-même), nous sommes dans l'obligation de rentrer dans les détails de notre code (VHDL) pour changer la partie de calcul des adresses, ce qui n'est pas un travail facile. Comme perspective de cette partie de notre travail, il est intéressant d'ajouter une couche haut-niveau au contrôleur, pour la spécification et la modélisation des besoins, et de fixer des outils pour changer les lignes de codes chargés de calcul des adresses en utilisant les informations fournies par la première couche.

# Chapitre 5

# Implantations des Modules de l'Encodeur H.264/AVC sur une Plateforme Reconfigurable

## 1. Introduction

Dans ce chapitre, nous présentons des nouvelles solutions pour l'implantation matérielle des modules de l'encodeur H264/AVC. Ces implémentations sont basées sur les deux principes de pipelining et de parallélisme des données. Nous exploitons ainsi le maximum de parallélisme autorisé par H.264, dans un macrobloc 16×16pixels, en vue de la dépendance inter-blocs dans les modules de traitement. En effet, la dépendance des traitements dans H.264 nous empêche d'envisager plus de parallélisme sans un surcoût matériel significatif. Pour cela, nous avons utilisé des entrées/sorties sur 128bits (bloc de 4×4 pixels) et nous avons comparé les performances obtenues avec celles basées sur des architectures à 32bits. Cette nouvelle approche permet la réduction des ressources FPGA utilisées et les besoins en mémoires internes ainsi qu'une réduction du temps de traitement d'un macrobloc.

Comme un premier travail dans ce chapitre, nous avons modifié les deux figures 2.2 et 2.5 du codec H.264/AVC dans le sens de montrer que dans H.264 les traitements sont appliqués aux macroblocs et non pas aux images entières (Figure 5.1), et aussi pour montrer la dépendance inter-blocs dans les modules de traitement de l'encodeur et du décodeur.

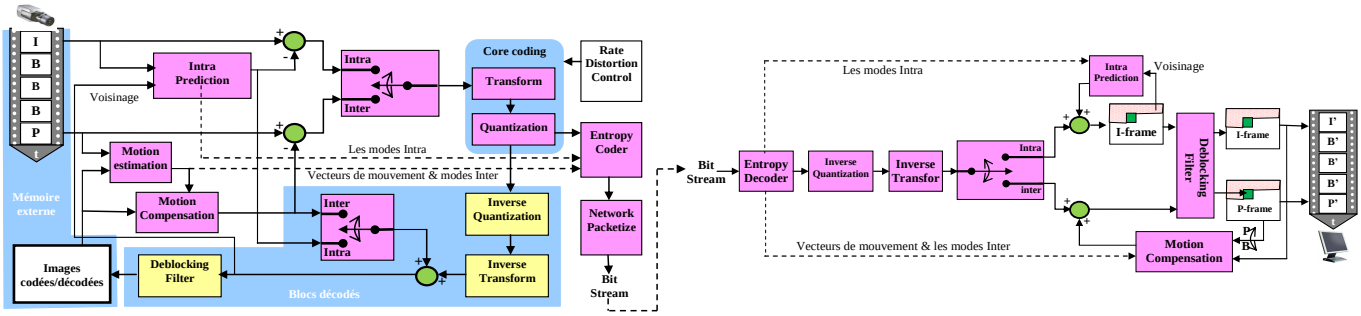



Figure 5.1. Traitement des macroblocs dans un Encodeur/Décodeur H.264/AVC.

## 2. Implémentation matérielle pour la chaine de prédiction Intra

Après une étude détaillée de l'encodeur H.264 dans le chapitre 2, nous avons constaté qu'il est impossible d'implanter la prédiction Intra sans avoir implanter les autres modules de transformation et quantification directes et inverses, et cela à cause de la dépendance inter-blocs et inter-macroblocs dans ces modules. En effet, nous avons besoin des résultats du premier bloc codé et décodé (après transformation et quantification directes et inverses) pour fixer les pixels de voisinage du bloc suivant avant leur traitement. En plus, nous avons besoin des 16 résultats de l'Intra4×4 pour les comparer avec le coût de l'Intra16×16, et cela avant de prendre la décision sur le type de partitionnement et avant de passer au traitement du macrobloc suivant (désignation des pixels de voisinage du macrobloc suivant). Pour cela, la chaine de prédiction Intra doit être implémentée en un seul module.

## 2.1. La chaine de prédiction Intra

La figure 5.2 montre le flux de codage Intra dans H.264/AVC. Premièrement le processus de prédiction Intra calcule, par comparaison des coûts, un mode Intra16×16 ou bien 16 modes Intra4×4. Ensuite les blocs résiduels sont calculés et transférés vers les modules de transformation directe et de quantification directe. La sortie de ces deux modules est envoyée à la fois vers l'encodage entropique et la chaine inverse. Cette chaine inverse est composée de deux modules de quantification et de transformation inverses, plus le module de reconstruction pour produire les blocs décompressés. A partir de ces blocs reconstruit et non filtrés sont extraits des pixels de voisinage pour les blocs suivants codés en Intra [94]. Après le filtrage, ces blocs reconstruits sont utilisés pour la prédiction de futures images codées en Inter.

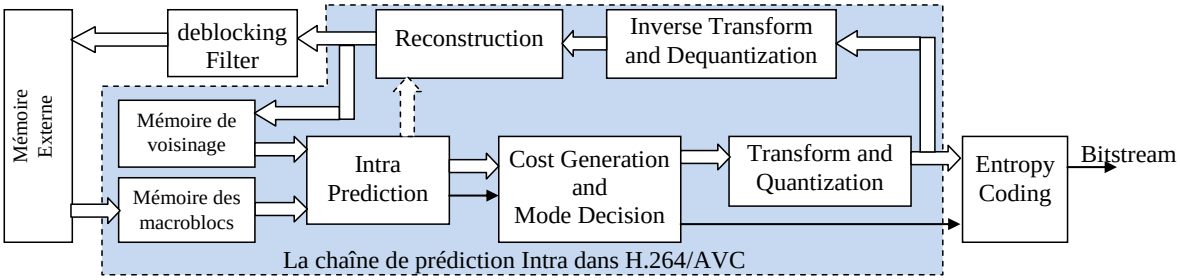



Figure 5.2. Flux de codage Intra dans l'encodeur H.264/AVC.

Comme mentionné dans le chapitre 2, si un macrobloc est codé en Intra, un autre macrobloc prédit (P-MB) est formé soit pour le macrobloc complet ou bien pour chaque bloc dans ce macrobloc. Le calcul de P-MB est basé sur les pixels de voisinage, à gauche et en haut, précédemment codés et reconstruits mais non filtrés. En effet, en utilisant quatre modes de prédiction Intra16×16, des P-MBs sont calculés pour le macrobloc 16×16 de luminance (I16MB) et en utilisant neuf modes de prédiction Intra4×4, des P-Bs sont calculés pour les 16 blocs 4×4 (I4B) [95]. Les composantes de chrominance sont prédites par blocs de 8×8 pixels, suivant une technique similaire à la prédiction de la composante de luminance [96]. Notez que le mode de prédiction qui en résulte pour les deux composantes de chrominance devrait être le même. Récemment, neuf modes de prédiction Intra8×8, pour un bloc de luminance 8×8, ont été ajoutés pour le profil haut de H.264/AVC [97].

La prédiction Intra dans H.264/AVC est basée sur le concept d'optimisation du taux de distorsion RDO (Rate-Distortion Optimization) [98], ce qui signifie que l'encodeur estime le macrobloc à coder en utilisant toutes les combinaisons de modes et choisir celui qui donne les meilleures performances en termes de RDO [93]. En effet, dans le logiciel de référence H.264/AVC [47], une recherche complète (Full Search Algorithm) est utilisée pour examiner l'ensemble des modes possibles, pour trouver le meilleur mode Intra16×16 ou bien les 16 meilleurs modes Intra4×4 [99]. Les principales étapes pour la prédiction Intra peuvent être résumées comme suit :

1. Trouver le meilleur mode de prédiction Intra16×16 ;
2. Trouver les 16 meilleurs modes de prédiction Intra4×4 ;
3. Comparer les coûts et sélectionner le (s) meilleur (s) mode (s) ;
4. Trouvez le meilleur mode de prédiction Intra du signal de chrominance.

Selon cette structure de prédictions Intra dans H.264, le nombre de combinaisons de modes pour les blocs de luminance et de chrominance dans un macrobloc est égal à M8·(M4·16+ M16), où M8, M4 et

M16 représentent le nombre de modes pour les blocs Chroma8×8, les blocs Luma4×4 et les macroblocs Luma16×16 respectivement. Cela signifie que, pour un macrobloc, 592 différentes combinaisons sont possibles [95]. Si la prédiction Intra8×8 est ajoutée, le nombre de combinaisons de mode s'élève à 736 [96].

La figure 5.3 montre l'organigramme d'implémentation logiciel de la chaine de prédiction Intra. Les deux organigrammes (a) et (b) montrent les détails des deux modules de prédiction (Intra4×4 et Intra16×16). Ces deux modules sont supposés réaliser en parallèle, ensuite les coûts sont comparés pour prendre la décision sur le type de partitionnement et les modes choisis (Figure 5.2.c).

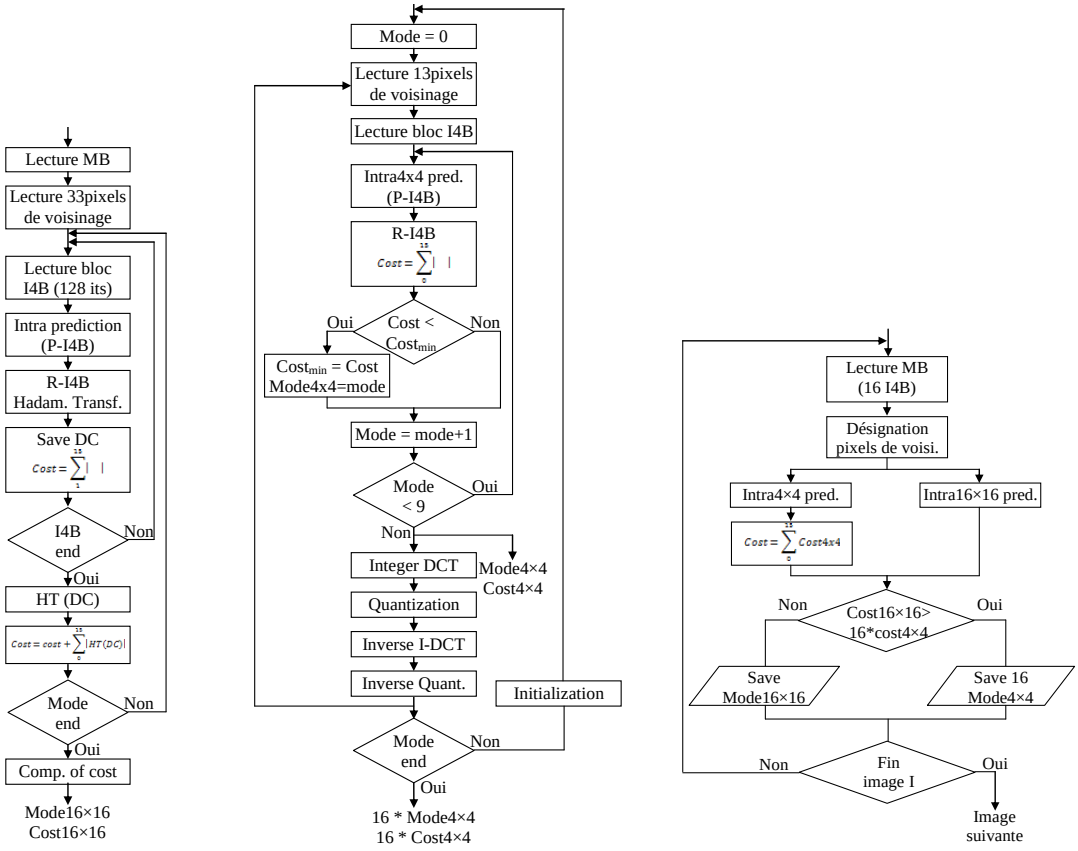



Figure 5.3. Implémentation logiciel de la chaine de prédiction Intra.

### 2.2. Les implémentations existantes de la chaine Intra

Dans H.264/AVC, une taille de bloc variable pour la prédiction Intra, maximise l'efficacité de codage basé sur l'optimisation de RDO. En conséquence, la complexité de traitement et la charge de calcul augmentent de façon spectaculaire [99]. Généralement, deux méthodes sont possibles pour réduire la complexité du mode de sélection dans la prédiction Intra : simplifier la fonction de coût ou bien minimiser le nombre des modes examinés [100]. La deuxième méthode consiste à chercher l'optimum par la recherche dans quelques modes seulement, ceci est significativement plus efficace pour réduire la complexité de traitement [97]. Dans ce sens, de nombreux algorithmes rapides ont été proposés. Pan et al. [98] [101] ont proposé un algorithme de décision Intra-mode rapide en utilisant l'histogramme de détection de contours. Cependant, ce stade de prétraitement consomme encore plus de temps de codage. La performance de cette méthode est d'environ 55~65% plus rapide que la méthode RDO au prix de 2-5% de bits supplémentaires à cause de minimums locaux. Meng et al. [100] ont proposé un

algorithme jugé efficace, pour la prédiction Intra, basé sur les sous-pixels échantillonnés dans les blocs 4×4, dans le but de sélectionner les modes hautement probables. Cette méthode permet la réduction du temps de calcul jusqu'à 70%, avec une augmentation de débit binaire. Hsu et al. [102] ont proposé un algorithme rapide basé sur les informations locales obtenues par le calcul des paramètres caractéristiques. Cette méthode permet de réduire le temps d'encodage d'environ 26% avec 1,4% moins de bits supplémentaires utilisés et pas plus de 0,7dB de PSNR est sacrifié. Elyousfi et al. [103] ont proposé un algorithme de sélection basé sur les informations directionnelles du mode de prédiction. Les modes Intra des blocs de voisinage, déjà calculés, sont utilisés pour sélectionner la moitié des modes candidats. Cette méthode réduit la complexité d'un maximum de 78,25% avec le maintien de la qualité PSNR d'environ 2% en moyenne. Park et al. [96] ont proposé une décision sélective de prédiction Intra basée sur le résultat du meilleur mode de prédiction des macroblocs. Le mode Intra16×16 est utilisé pour sélectionner les modes candidats pour les plus petits blocs (4×4). Huang et al. [104] ont proposé l'utilisation d'une table spécifique pour les modes probables pour chaque bloc, selon des modes de voisinage déjà calculés. Ce processus de décision peut être accéléré en cherchant seulement dans une liste limitée de modes avec plus de possibilités de trouver la solution optimale. Cette méthode améliore un peu la qualité par rapport aux méthodes précédentes, mais avec encore un PSNR considérable par rapport à la recherche exhaustive en raison de propagation d'erreur. Les auteurs dans [100] ont proposé la même technique avec une réduction de 50% dans le calcul de chaque mode.

Malgré toutes ces accélérations dans le module de prédiction Intra, l'implémentation logicielle de ces propositions reste loin du temps réel surtout avec des applications qui ne cesse pas de se compliquer. En effet, pour répondre aux besoins en temps réel d'un grand nombre d'applications multimédias utilisant H.264/AVC, plusieurs implémentations matérielles ont été proposées dans le but d'accélérer les calculs dans les différents modules de traitement y compris le module de prédiction Intra. Babionitakis et al. [93] ont proposé une implémentation matérielle complète pour le codeur H.264. Un algorithme de recherche complet (FSA) est utilisé pour les deux modules Intra4×4 et Intra16×16. Le calcul des pixels prédit est effectué par deux composantes différentes (Luma16×16 et Luma4×4). Chaque composante de prédiction calcule un pixel prédit en un cycle d'horloge pour chaque mode de prédiction disponible. Suh et al. [105] ont proposé une implémentation matérielle pour le module de prédiction Intra seulement. La méthode proposée peut traiter un macrobloc en 927 cycles d'horloge. Wang et al. [106] ont proposé l'utilisation d'une transformation en nombre entier pour le calcul des coûts des modes Intra16×16 et l'utilisation de l'algorithme SADT pour le calcul des coûts des modes Intra4×4. Cette implémentation est jugée capable d'achever une meilleure efficacité avec le même PSNR. Ku et al. [94] ont proposé une implémentation matérielle pour le module de prédiction Intra sous forme d'un IP particulièrement pour des applications vidéo numériques. Dans [107] un exemple de programme VHDL de la chaine de prédiction Intra est fourni en open-source. Dans cette implémentation matérielle, les composants ne sont pas finalisés et le module Intra16×16 n'est pas pris en compte. La synthèse de ce programme en utilisant la plateforme Xilinx ML501 donne comme résultat : 15% de LUTs et 9% des slices-registres, en plus de l'utilisation de 2 BRAM et 2 DSP. La fréquence maximale de fonctionnement est de 116 MHz. Nous avons utilisé ces programmes VHDL comme référence pour nos implémentations.

## 2.3. Les implémentations proposées pour la chaîne Intra

Toutes les architectures matérielles proposées pour l'encodeur H.264 utilisent un bus de données de 32bits. Cette limitation de données en entrée requiert d'autres techniques de parallélisme (parallélisme des modes et parallélisme des composants), ce qui limite aussi les possibilités de pipeline et augmente considérablement le nombre de cycles d'horloge pour le traitement d'un macrobloc ainsi que le nombre de registres tampons utilisés pour les résultats intermédiaires.

Dans l'implémentation proposée, nous supposons que l'entrée du module Intra est un bloc entier (4×4 pixels extrait d'un macrobloc en cours de traitement, ce qui représente un bus de 128bits) en plus des pixels de voisinage disponibles. La sortie est un macrobloc résiduel (bloc par bloc) plus un mode ou bien 16 modes de prédiction. En effet, pour chaque macrobloc, un mode16×16 ou bien 16 mode4×4 sont calculés pour le signal de luminance et un mode unique pour les deux composantes de chrominance. Les mode4×4 et le mode16×16 sont choisis parmi un nombre limité de modes suivant la disponibilité des pixels de voisinage (en fonction de la position de macrobloc dans l'image) et suivant l'algorithme de recherche utilisé (algorithme de recherche complet / algorithmes de recherche rapide). Le mode qui donne le coût minimal sera sélectionné comme mode en sortie. La figure 5.4 montre les différents composants matériels proposés pour l'implantation de la chaîne de prédiction Intra.

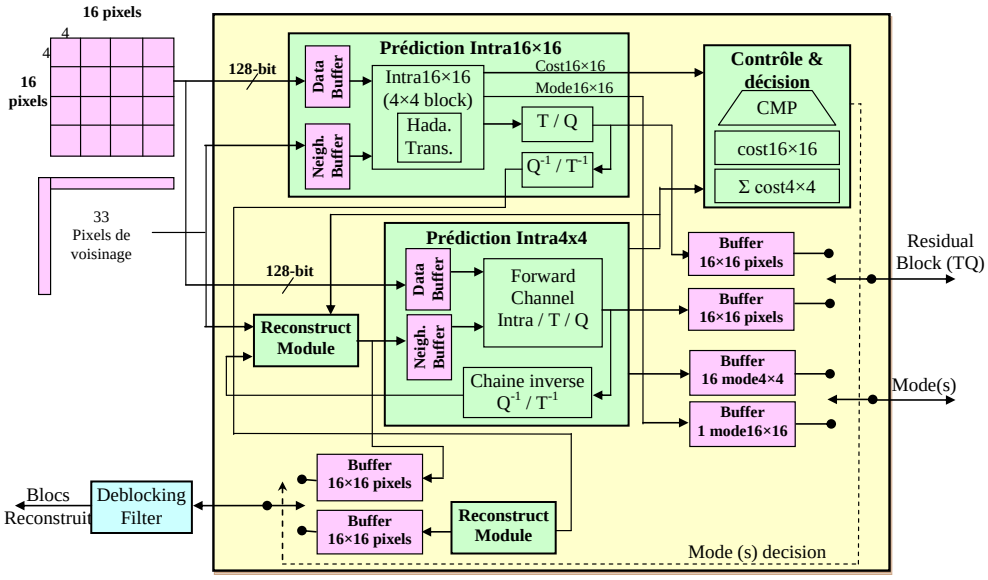



Figure 5.4. L'architecture globale proposée pour la chaîne de prédiction Intra.

L'objectif initial de cette implantation, de la chaîne Intra, consiste à parvenir à une implémentation matérielle capable de prendre en charge la multitude des algorithmes proposés dans la littérature (Full/Fast). En effet, cette architecture doit être adaptée aux algorithmes rapides dont l'objectif final est de minimiser le nombre de modes de prédiction réalisés. L'architecture doit aussi assurer des traitements parallèles et en pipeline.

### 2.3.1. Analyse de la complexité du module de prédiction Intra

Plusieurs études de l'encodeur H.264 ont montré que le module de prédiction Intra prend une grande partie du temps de calcul dans une implémentation logicielle (soit 20% dans [67] et [68]) et cela par rapport aux autres normes de compression vidéo où la prédiction Intra est effectuée dans le domaine fréquentiel. Ce module nécessite aussi un nombre important de composants dans une implémentation

matérielle (Figure 5.4). Le module Intra16×16 contient deux transformations de Hadamard en cascade et plusieurs autres composants. Le module Intra4×4 est composé de deux grandes parties : le calcul des coûts et le calcul des transformées quantifiées des blocs résiduels. En plus, le module de prédiction Intra contient aussi la chaine inverse pour reconstruire les blocs (après codage/décodage). Les pixels de voisinage sont extraits à partir de ces blocs reconstruits, cela limite le parallélisme de données et le pipeline de tâches. En effet, le module de codage Intra ne peut pas traiter deux blocs (dans le même macrobloc) en même temps à cause de la dépendance inter-blocs.

Notre contribution dans cette partie de la thèse consiste à proposer une implémentation matérielle capable de réaliser les modes de prédiction en série (Figure 5.5.b) et non pas en parallèle (figure 5.5.a), contrairement aux autres méthodes déjà implémentées et publiées. La réalisation des modes en série nous assure les avantages suivants :

1. Nous pouvons programmer le nombre souhaité de modes de prédiction (Intra4×4 et Intra16×16) :
   - Selon la position du macrobloc dans l'image (la disponibilité des pixels de voisinage) ;
   - Selon la méthode de recherche des modes choisis (exhaustive/rapide). En effet, l'architecture matérielle proposée doit être adaptée aux différents algorithmes rapides.
2. Le module de prédiction Intra peut être divisé en plusieurs tâches, ce qui donne la possibilité de parallélisme et de pipelining. En effet, la seule tâche qui change pour les différents modes de prédiction concerne la tâche de calcul du mode, les autres tâches restent les mêmes pour tous les modes (Intra4×4 ou bien Intra16×16). Nous pouvons utiliser le même composant pour les différents modes pour compenser l'utilisation des 4 composants en parallèle (4×32bits).

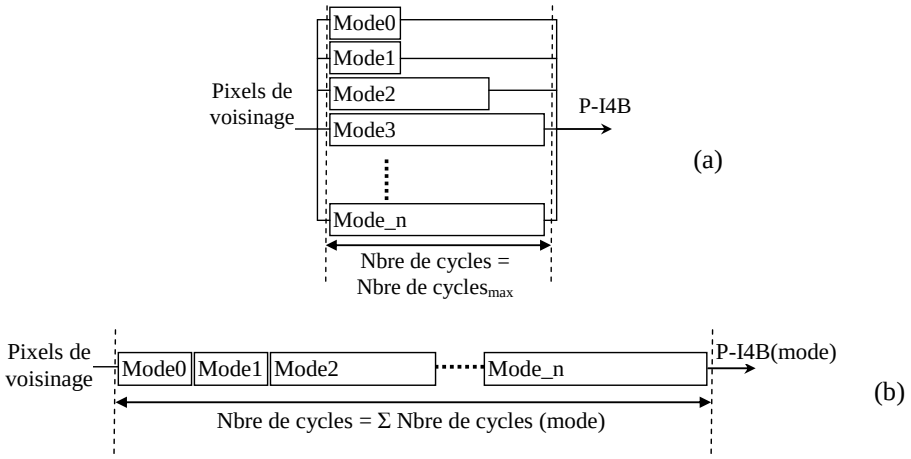



Figure 5.5. Les deux méthodes possibles pour l'implantation des modes de prédiction Intra.

### 2.3.2. Approche basée sur le pipeline

Si nous adoptons la stratégie précédemment discuté pour l'implantation du module Intra, nous pouvons bénéficier d'une architecture en pipeline. Ce qui permet de réduire considérablement le nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc sans une augmentation considérable des ressources utilisés. Chaque module de traitement est divisé en un nombre fini de tâches. La spécificité de la prédiction Intra est que les mêmes tâches sont utilisées pour tous les modes de prédiction à l'exception de la première tâche qui consiste à calculer le bloc prédit. La figure 5.6 montre la possibilité de réalisation des modes de prédiction en pipeline. Cette technique permet une réduction considérable du temps de traitement.

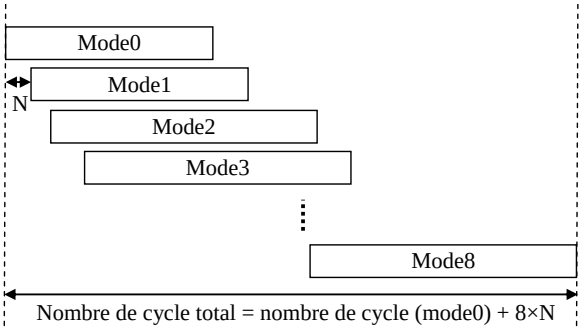



Figure 5.6. Réalisation des modes de prédiction Intra en pipeline.

### 2.3.3. Description du module Intra4×4

La figure 5.7 montre les détails de l'architecture proposée pour l'implantation du module Intra4×4. Les données d'entrée sont supposées de 128bits (bloc I4B de 4×4pixels). L'idée de base de cette architecture consiste à lire et à traiter un I4B à la fois pour fournir plus de possibilités de parallélisme. En plus, avec le traitement de données sur 128bits, au lieu de 32bits, il est possible de simplifier les équations représentant le traitement souhaité.

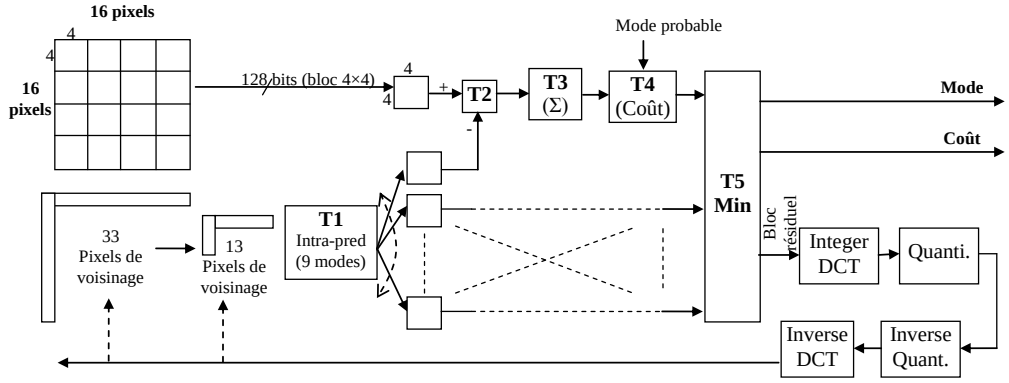



Figure 5.7. Architecture matérielle proposée pour le module Intra4×4.

Après un simple calcul, nous obtenons une meilleure perspective des ressources matérielles utilisées, ce calcul va être confirmé par la synthèse de l'implémentation matérielle proposée en utilisant une plateforme FPGA. Les implémentations existantes utilisent un composant matériel de base avec des entrées/sorties de 32 bits, ce composant est reproduit, ensuite, 9 fois en parallèle pour les 9 modes possibles. Dans l'implémentation proposée, le même composant de 32 bit est reproduit 4 fois en parallèle pour assurer le traitement des 128bits (4×32bits) en entrée. Par contre, les modes sont réalisés en série (en utilisant le même composant avec des données en pipeline). La figure 5.8 montre la décomposition du module Intra4×4 en 6 tâches élémentaires. Ces tâches sont les mêmes pour les 9 modes de prédiction à l'exception de T1. Les modes pour T1 sont implantés en parallèle.

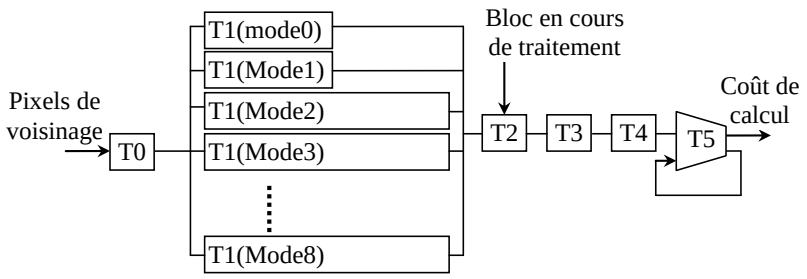



Figure 5.8. Réalisation en série (en pipeline) des tâches dans le module Intra4×4.

Dans la tâche T1, les traitements du mode0 et du mode1 sont de simples affectations de pixels, ils peuvent être réalisés en un cycle d'horloge. Le mode2 (DC) consiste à calculer la moyenne des 8pixels qui nécessite 7 opérations d'additions et une opération de division par 8 (la division par 8 est réalisée par décalage à droite de 3bits). Les autres modes exigent également des opérations d'addition, de multiplication et de division en fonction de la position du pixel et le mode choisi. Les tâches du module Intra4×4 sont données comme suit :

- Tâche T0 : Initialisation et lecture des données à partir des mémoires locales ;
- Tâche T1 : Calcul du bloc prédit (P-I4B) pour les modes 0-8. La durée de cette tâche est égale à un cycle d'horloge pour les modes (0 et 1) et à 2 cycles d'horloge ou bien plus pour les autres modes ;
- Tâche T2 : Calcul du bloc résiduel (I4B – P-I4B) ;
- Tâche T3 : Calcul de la valeur absolue des éléments du bloc résiduel ;
- Tâche T4 : Calcul du coût. Cette tâche est réalisée sur 2 cycles d'horloge ou bien plus ;
- Tâche T5 : Comparaison du coût calculé avec les coûts des autres modes.

Figure 5.9 montre le principe de pipeline dans le module Intra4×4. Malgré l'utilisation des registres intermédiaires pour assurer le pipeline, cette technique permet de réduire considérablement le temps de calcul pour les 9 modes de l'Intra4×4 sans une augmentation significative des ressources utilisées.

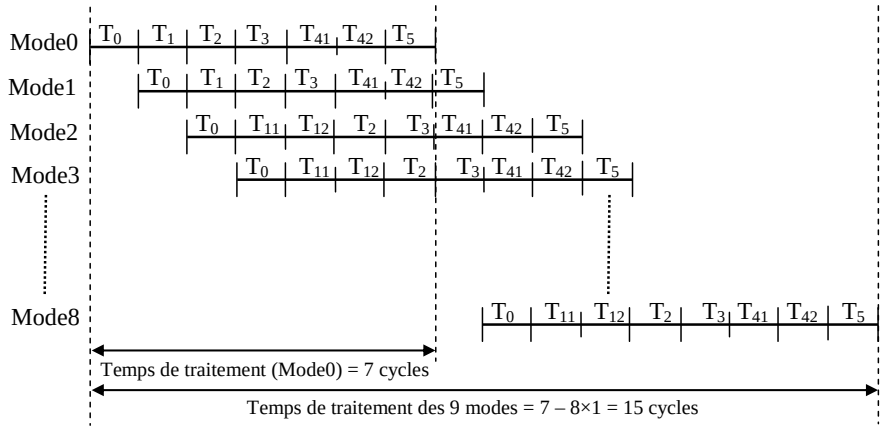



Figure 5.9. Réalisation des tâches du module Intra4×4 en pipeline.

### 2.3.4. Description du module Intra16×16

La figure 5.10 montre les détails de l'implémentation matérielle du module de prédiction Intra16×16. Avec une entrée de 128bits, le module Intra16×16 nécessite des résultats intermédiaires pour calculer les coûts des 4 modes possibles avant leurs comparaisons, ceci nécessite l'utilisation des registres intermédiaires (des tampons). Chaque coût est calculé à partir de l'ensemble des coûts des 16 blocs dans le même macrobloc. Les résultats intermédiaires peuvent être des blocs de différences, des blocs de coefficients DC ou bien des coûts intermédiaires, et cela suivant la stratégie d'implantation choisie. Généralement, deux réalisations sont possibles :

1. Prendre un bloc (4×4) et lancer les 4 modes en série (comme dans le module Intra4×4). Après la transformation du bloc résiduel, le coefficient continu (DC) est enregistré dans un registre tampon4×4 (un tampon par mode) et les autres coefficients sont additionnés et enregistrés dans un autre registre tampon. Cette procédure est répétée pour les 16 blocs pour avoir enfin 4 coûts pour

les 4 modes, ensuite il faut ajouter la somme des coefficients DC au coût de chaque mode, et finalement le choix du mode est obtenu avec le minimum de coût.

Cette première méthode nécessite la lecture de chaque bloc une seule fois ; ce qui limite l'accès à la mémoire interne, mais elle présente l'inconvénient de l'utilisation intensive des registres tampons et l'opération de comparaison est réalisée à la fin (ce qui représente encore un temps de retard).

2. La deuxième possibilité consiste à appliquer le même mode sur les 16 blocs pour calculer un seul coût, ce qui donne la possibilité de pipeline. Ensuite refaire le même traitement pour les 3 autres modes de prédiction. Malgré que l'accès à la mémoire interne soit multiplié par 4, cette technique réduit considérablement le nombre des registres tampon utilisés. L'opération de comparaison des coûts peut être réalisée à la fin de calcul de chaque mode (comparaison de 2 coûts au lieu de 4 dans la première possibilité).

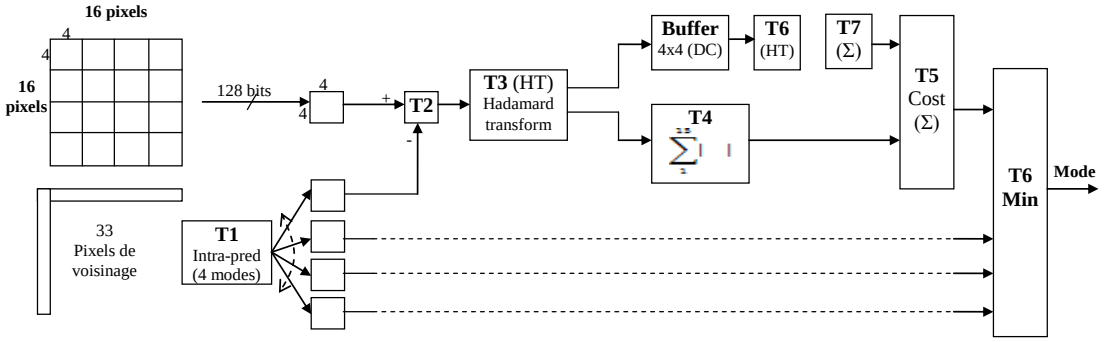



Figure 5.10. Architecture matérielle proposée pour le module Intra16×16.

Dans l'implémentation matérielle proposée, nous retenons la deuxième solution. Selon cette stratégie, nous proposons 6 tâches pour le module Intra16×16 :

- Tâche T0 : Initialisation et lecture des données à partir de la mémoire locale ;
- Tâche T1 : Calcul du bloc prédit (P-I4B) pour chaque bloc (I4B) ;
- Tâche T2 : Calcul du bloc résiduel (I4B – P-I4B) ;
- Tâche T3 : Calcul de la transformée de Hadamard du bloc résiduel. Cette partie nécessite 5 cycles d'horloge. Le coefficient DC est enregistré dans un registre tampon et les autres coefficients sont utilisés dans T4 ;
- Tâche T4 : Calcul de la somme des coefficients de la transformée de Hadamard. Cette tâche est réalisée en 2 (ou bien plus) cycles d'horloge ;
- Tâche T5 : Addition des coûts des 16 blocs pour le même mode (coût16 = Σ coût4) ;

Pour assurer un fonctionnement parallèle et rapide, deux autres tâches sont ajoutées :

- Tâche T6 : Calcul de la moyenne des pixels de voisinage pour le mode2 (DC). Cette tâche est réalisée au début en parallèle avec le Mode0 pour préparer le résultat au Mode2,
- Tâche T7 : Comparaison des coûts des 4 modes. Cette tâche est répétée 3 fois après le calcul des coûts des Mode1, Mode2 et Mode3.

La transformation de Hadamard est utilisée à 2 reprises dans le schéma proposé, une première reprise pour la transformation des blocs de différence et une deuxième reprise pour la transformation des coefficients DC. Pour éviter cette multiple utilisation, nous proposons l'utilisation des mêmes tâches

(T3 et T4) pour calculer le coût de la matrice des DC, avec l'inconvénient du temps de retard de quelques cycles d'horloge pour chaque mode. La figure 5.11 montre la réalisation du module Intra16×16 en tâches ainsi que la possibilité de pipelining.

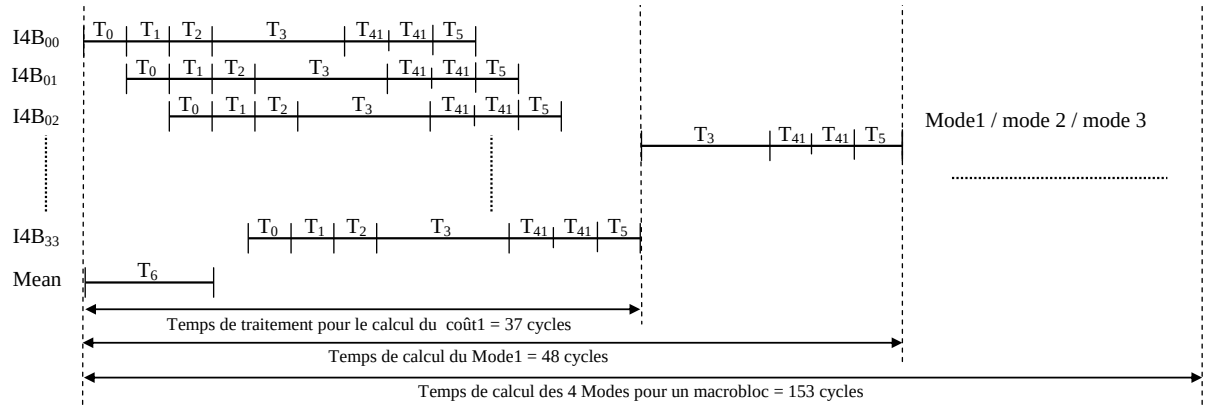



Figure 5.11. Réalisation des tâches du module Intra16×16 en pipeline.

Les quatre modes de prédiction Intra8×8 pour les deux composants de chrominance sont très semblables aux quatre modes de la prédiction Intra16×16. Le traitement Intra8×8 est indépendant des autres modules de traitement, pour cela la composante matérielle Intra16×16 est utilisée pour la prédiction Intra8×8 dans l'implémentation matérielle proposée.

### 2.3.5. Sorties du module de prédiction Intra

La décision du mode retenu est réalisée par comparaison des coûts (le coût16×16 et la somme des 16 coût4×4). La sortie du module Intra est un seul Mode16×16 ou bien 16 Mode4×4. En plus du mode (des modes) une information supplémentaire est à transférer au module suivant (les blocs résiduels selon les modes choisis). Ces blocs doivent être transformés, quantifiés et transférés avec les modes vers l'étape finale du codage entropique. Notons que pour le module Intra4×4, les blocs (résiduels/transformés/quantifiés) sont déjà calculés pour les 9 modes, ce qui donne 2 possibilités :

- La première possibilité consiste à garder les blocs résiduels déjà calculés (après transformation et quantification) dans des mémoires tampons pour éviter de refaire les mêmes calculs ;
- La deuxième possibilité consiste à enregistrer les modes prédis uniquement, pour éviter les mémoires tampons, ensuite refaire les calculs du module Intra4×4 pour un seul mode (le mode choisi après comparaison des coûts).

### 2.3.6. Implémentation matérielle pour la transformation des nombres entiers

Dans la transformation des nombres entiers donnée par l'équation (2.11), il est impossible de calculer le produit matriciel ($W = C_f * F$) avant de calculer tous les éléments de la matrice ($F = X * {C_f}^T$). Si on utilise une transformation 1D par la lecture de 32 bits à la fois, il faut attendre 4 temps, pour compléter la matrice F, avant de commencer les 4 autres transformations 1D pour le calcul de la matrice W avec la nécessité d'utilisation des buffers pour les éléments de la matrice F. Ceci limite les possibilités de parallélisme et de pipeline et augmente considérablement le nombre de cycle d'horloge pour le traitement d'un bloc (13cycles) ou bien d'un macrobloc (148cycles) [105]. La figure 5.12 montre la dépendance entre éléments dans le calcul des coefficients de la transformation des nombres entiers.

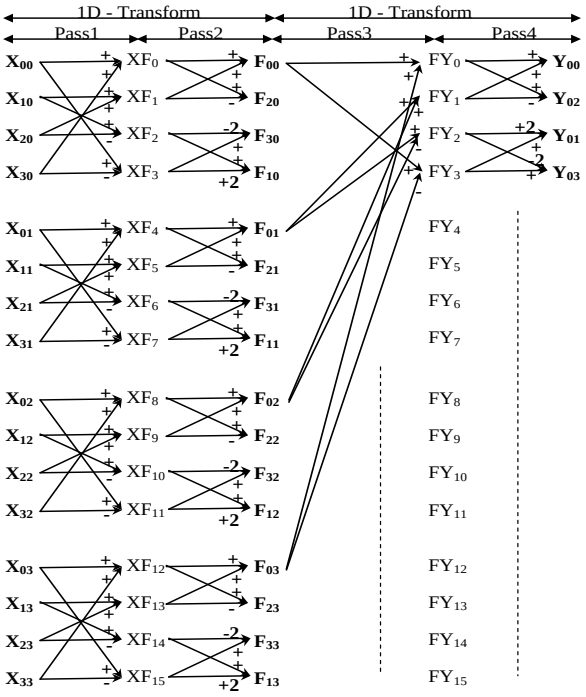



Figure 5.12. Dépendances des calculs dans la transformation des nombres entiers.

La lecture des données en parallèle sur 128bits démunit considérablement le nombre de cycles d'horloge (5cycles pour le traitement d'un bloc), et donne la possibilité de pipeline ce qui démunit considérablement le temps de traitement d'un macrobloc (20cycles uniquement). En plus avec l'utilisation des données sur 128bits nous obtenons la possibilité de simplification des opérations de calcul. La figure 5.13 montre la possibilité de pipeline dans l'implémentation proposée pour le module de transformation. Les tâches dans ce cas sont les équations de calculs des différents résultats intermédiaires (XF à partir de X, F à partir de XF, YF à partir de F et Y à partir de YF).

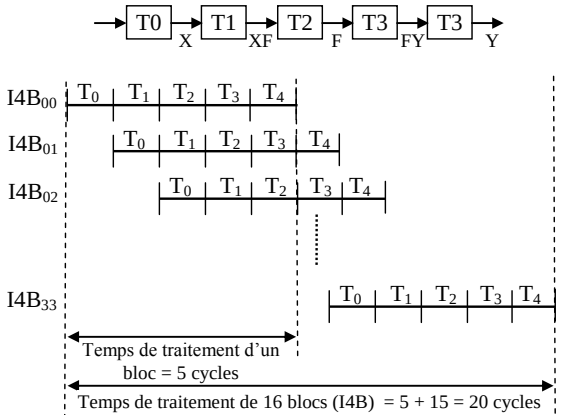



Figure 5.13. Implantation en pipeline de la transformation des nombres entiers.

### 2.3.7. Implémentation matérielle pour le module de quantification

Le module de quantification (mise à l'échelle) utilisé dans l'encodeur H.264 a été optimisé pour être réalisé avec un minimum de ressources matérielles, notamment en évitant les opérations en virgule flottante utilisée dans d'autres encodeurs. Malgré les significatives simplifications, l'équation 2.14 contient une multiplication par un nombre naturel de 14bits. Ce nombre (tableau 2.4) varie en fonction de la valeur QP et de la position de l'élément dans la matrice d'entrée (dans le macrobloc). Dans l'implémentation proposée, les éléments du tableau 2.4 sont stockés dans une mémoire ROM. Pour la

multiplication, nous utilisons les DSP48Es du circuit FPGA. Ces DSPs permettent de réaliser les opérations de multiplications en un cycle d'horloge et permettent aussi l'enregistrement des entrées/sorties dans les registres internes. Afin d'éviter les opérations d'addition sur un grand nombre de bits dans l'équation 2.14, nous proposons de réaliser l'addition après l'opération de décalage. Ce qui donne l'équation suivante :

$$|Z_{ij}| = SHR(|W_{ij}|.MF, qbits) + SHR(f, qbits) \quad (5.1)$$

Le même principe de parallélisme de données est utilisé dans ce module. Les coefficients transformés d'entrée (4×4) sont traités par des tâches multiples en cascades avec des registres intermédiaires pour donner la possibilité de pipeline. En effet, si le mode Intra16×16 est sélectionné, le module peut traiter le premier bloc en 4 cycles d'horloge, les autres blocs sont traités en un cycle de plus. Par conséquent, le nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc sera égal à 19 cycles.

Le même principe est utilisé pour l'implantation des autres modules de la chaine inverse (la quantification inverse et la transformation inverse). Les entrées/sorties sont supposés en 128bits, et les traitements sont divisés en tâches avec des registres intermédiaires pour assurer des architectures en pipeline.

### 2.3.8. Architecture matérielle globale

En plus des modules de traitement élémentaires dans la prédiction Intra, un module principal est nécessaire pour assurer la gestion des entrées/sorties avec les mémoires nécessaires. Ce module contient aussi la stratégie de liaison entre les différents modules élémentaires ainsi que la partie de contrôle et de décision. En effet, ce module contient une mémoire de capacité 16×128bits pour l'enregistrement du macrobloc en cours de traitement (le contrôleur mémoire proposé dans le chapitre 4 contient déjà cette mémoire). Dans ce cas, le contrôleur est chargé de remplir cette mémoire par les macroblocs suivant l'ordre défini dans le logiciel de référence de H.264. En plus de la mémoire 16×128bits, le module de codage Intra contient une autre mémoire pour l'enregistrement des pixels de voisinage pour les 2 modules élémentaires (Intra16×16 et Intra4×4). Une procédure est spécialement définie pour le calcul de ces pixels de voisinage à partir des blocs déjà traités et reconstruits.

La partie de contrôle de ce module contient également une partie de comparaison des coûts (16.coût4×4 et un coût16×16), pour prendre la décision sur le type de partitionnement du macrobloc. Si la décision est de maintenir la taille 16×16, les blocs résiduels calculés selon le mode16×16 sélectionné seront transformés et quantifiés. Si la décision est de partitionner le macrobloc en 16 blocs, chacun de ces blocs sera transformé et quantifié.

## 2.4. Résultats de simulation et de synthèse

L'architecture matérielle proposée pour le module Intra est écrite en langage VHDL. Pour vérifier le bon fonctionnement de l'architecture, le design a été déployé sur la plateforme de prototypage ML501. Les modules élémentaires ont été simulés séparément avant leurs intégrations dans un module global (le module principal de la prédiction Intra). En effet, pour chaque module, un testbench a été écrit pour vérifier son fonctionnement sur des données synthétiques, et pour calculer le nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc. Entre les différentes tâches dans le même module élémentaire, nous avons ajouté des registres pour assurer des architectures en pipeline, ce qui

explique le grand nombre de registres utilisés (29% de la plateforme ML501). Ces structures en pipeline ont été vérifiées avec les données d'entrée multiples pour chaque cycle d'horloge.

Le tableau 5.1 résume les résultats de synthèse de ces architectures en utilisant la plateforme ML501 et donne aussi une comparaison avec les résultats des implémentations proposées dans [107] (basées sur des entrées/sorties de 32bits). On devrait s'attendre à ce que le surcoût en ressources matérielles soit multiplié par un facteur de 4 lors du passage d'une implémentation 32bits à celle de 128bits (4×32). Avec l'approche proposée, les ressources consommées n'ont augmenté que de 2 fois pour les LUT et les BRAM alors que les slices-registres ont augmentée de 3 fois. En outre, l'implémentation proposée est une version complète du module Intra, l'implémentation [107] est fondée uniquement sur la prédiction Intra4×4. Pour cette augmentation en ressources utilisées, nous avons amélioré le temps de traitement d'un macrobloc de 74% (687 au lieu de 1200 cycles d'horloge). La fréquence de fonctionnement est augmentée de 39% (233MHz au lieu de 167MHz) à cause des architectures en pipeline utilisées dans nos implémentations, cela explique aussi l'augmentation du nombre de slices-registres utilisé.

| | Ressources / Modules \ Performances | Silice-registres | LUTs | BRAM /ROM | DSP48E | Fréquence max (MHz) | Nom. Cycles d'horloge | Possibilité de pipeline |
|---|---|---|---|---|---|---|---|---|
| **Les implémentations proposées (128 bits)** | Intra4×4 | 1 070 | 1 584 | 2 | / | 229,861 | 15/bloc | Oui |
| | Intra16×16 | 1 603 | 1 672 | / | / | 261,361 | 150/MB | Oui |
| | Transformation directe | 1 150 | 776 | / | / | 457,247 | 5/bloc, 20/MB | Oui |
| | Quantification | 355 | 943 | ROM 64×42bit | 16 | 366,569 | 4/bloc, 19/MB | Oui |
| | Quantification inverse | 451 | 739 | ROM 64x15bit | 16 | 366,569 | 4/bloc, 19/MB | Oui |
| | Transformation inverse | 1 187 | 912 | / | / | 457,247 | 5/bloc, 20/MB | Oui |
| | Total de la chaîne intra | 8 456 | 9 087 | 9 BRAM | 32 | 233,032 | 687 | / |
| **Les implémentations dans [107] (32 bits)** | Intra4×4 | 450 | 601 | 2 | / | 238,903 | / | Non |
| | Intra16×16 | Non programmé | | | | | | |
| | Transformation directe | 101 | 105 | / | / | 443,321 | / | Non |
| | Quantification | 35 | 130 | ROM 64×42bit | 1 | 308,471 | / | Non |
| | Quantification inverse | 26 | 93 | ROM 64x15bit | 1 | 371,913 | / | Non |
| | Transformation inverse | 776 | 715 | / | / | 304,507 | / | Non |
| | Total de la chaîne intra | 2 536 | 4 376 | 4 RAM | 2 | 167,653 | 1200 | Non |

Tableau 5.1. Résultats de synthèse des différents modules élémentaires de la prédiction Intra.

Le tableau 5.2 récapitule les résultats de synthèse de l'architecture globale proposée pour le module de prédiction Intra en utilisant les deux plateformes ML501 et XUPV5. Ce module utilise 31% des LUTs de la ML501 et 13% de la XUPV5, il utilise aussi 29% des slices-registres de la ML501 et 12% de la XUPV5. En plus des LUTs et des slices-registres, le module proposé utilise 9 BRAM et 32 DSP48E ce qui représente (18%, 66%) et (6%, 50%) sur les 2 plateformes respectivement. L'implémentation fonctionne à une fréquence maximale de 233,032MHz sur les 2 plateformes.

| | | **Slice-registres** | **LUTs** | **fully used Bit Slices** | **BRAM** | **DSP48E** | **Fréquence max (MHz)** |
|---|---|---|---|---|---|---|---|
| **ML501** | Utilisé | 8 456 | 9 087 | 5 222 | 9 | 32 | 233,032MHz |
| | Disponible | 28 800 | 28 800 | 12 321 | 48 | 48 | |
| | Taux d'utilisation | 29% | 31% | 42% | 18% | 66% | |
| **XUPV5** | Utilisé | 8 452 | 9 079 | 5 218 | 9 | 32 | 233,032MHz |
| | Disponible | 69 120 | 69 120 | 12 313 | 148 | 64 | |
| | Taux d'utilisation | 12% | 13% | 42% | 6% | 50% | |

Tableau 5.2. Reliquat des ressources FPGA utilisées pour le module Intra.

Dans le tableau 5.3, nous comparons notre implémentation avec les autres implémentations dans [93] [94] [105-107]. Nous prendrons en considération les modules implémentés, les détails des modules, les caractéristiques et les performances des implémentations.

| | **Implé. proposée** | **[94]** | **[106]** | **[105]** | **[93]** | **[107]** |
|---|---|---|---|---|---|---|
| **Les modules implémentés** | | | | | | |
| Intra4×4 | Yes | Yes | Yes | Yes | Yes | Yes |
| Intra16×16 | Yes | Yes | Yes | Yes | Yes | No |
| Transfo. Nombre entier | Yes | Yes | Yes | Yes | Yes | Yes |
| Quantification | Yes | Yes | Yes | Yes | Yes | Yes |
| Quantification inverse | Yes | Yes | Yes | Yes | Yes | Yes |
| Transformation inverse | Yes | Yes | Yes | Yes | Yes | Yes |
| Module de reconstruct. | No | Yes | Yes | N/A | N/A | No |
| **Les détails des modules** | | | | | | |
| Transfo. de Hadamard | Yes | Yes | No | Yes | N/A | No |
| Utilist. Des valeurs DC | Yes | Yes | No | Yes | N/A | No |
| Mode de préd. (Plane) | Yes | No | Yes | N/A | N/A | No |
| **Caractéristiques et performances des implémentations** | | | | | | |
| Pipelining | Mode/MB-based | MB-based | MB-based | MB-based | MB-based | MB-based |
| Pixel parallelism | 16 pixels | 4 pixels | 4 pixels | 4 pixels | 4 pixels | 4 pixels |
| Fréq. fonctionne. max | 233 MHz | 103 MHz | 85 MHz | 54 MHz | N/A | 136 MHz |
| Nbre cycles/macrobloc | 687 | <1080 | <1300 | 927 | N/A | 1200 |
| Nbre portes/taille puce | N/A | 103K | 85K | 192K | 354,638 µm² | N/A |
| Ressources FPGA | 31% | N/A | N/A | N/A | N/A | 15% |

Tableau 5.3. Comparaison de l'implémentation proposée avec les implémentations existantes.

### 2.5. Conclusion

Dans cette partie de notre travail, nous avons présenté une nouvelle architecture matérielle pour les modules de la chaîne de prédiction Intra utilisée dans l'encodeur H.264/AVC. Les architectures proposées sont basées sur les principes de parallélisme de données et de pipelining des tâches. Nous avons proposé des architectures efficaces pour les modules élémentaires ainsi que le module global, des architectures pour l'Intra-prédiction, la transformation et la quantification (TQ), la quantification et la transformation inverses ($QT^{-1}$), ainsi que le module de décision et de contrôle. Les entrées en 128bits sont supposées assurées par un contrôleur mémoire intelligent (chapitre 4) conçu spécialement pour l'encodeur H.264. Pour assurer des architectures en pipeline, les modes de prédiction (dans les modules Intra4×4 et Intra16×16) sont réalisés en série et non pas en parallèle. Ceci aide à la gestion de la mémoire pour les différents modules et donne la possibilité de simplification des équations de traitement, ce qui conduit à une augmentation du débit de données pour chaque bloc et chaque macrobloc sans une augmentation significative des ressources FPGA utilisées. L'architecture proposée peut traiter un macrobloc en 687 cycles d'horloge. La synthèse de cette architecture, en utilisant les deux plateformes ML501 et XUPV5, donne comme résultats la consommation de 29% et 12% des ressources FPGA respectivement avec une fréquence maximale de fonctionnement de 233 MHz.

## 3. Implémentation matérielle pour le filtre anti-blocs

Le filtre anti-blocs peut être étudié et implémenté indépendamment des autres modules de l'encodeur H.264/AVC, des simulations conçues spécialement pour ce module peuvent être appliquées. L'implantation matérielle de ce filtre est réalisée selon plusieurs stratégies à base de 4 ou bien 5 filtres

directionnels et avec des données de 32bits ou bien de 128bits. Des résultats de synthèse, présentés à la fin de ce chapitre, montrent l'intérêt de subdivisons des filtres, l'intérêt d'ajouter un 5$^{eme}$ filtre et l'intérêt d'utiliser des données sur 128bits.

### 3.1. Les implémentations existantes

Les difficultés rencontrées lors de l'implantation du filtre anti-blocs utilisé dans H.264/AVC manifestent dans l'ordre de filtrages des blocs. L'ordre séquentiel proposé par la norme H.264/AVC [46] est schématisé dans la figure 5.14.

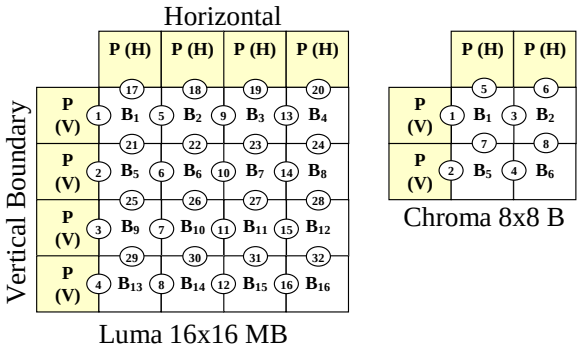



Figure 5.14. Ordre de filtrage proposé dans la norme H.264/AVC.

La limite imposée est que si un pixel ou bien un groupe de pixels sont impliqués dans le filtrage vertical et horizontal, ce dernier doit précéder le premier. Malgré cette limite, l'ordre présenté dans la figure 5.14 offre plusieurs possibilités pour l'implantation du filtre en explorant différents ordres de filtrage et en visant un fonctionnement plus rapide grâce à l'utilisation du parallélisme ou à des solutions qui utilisent moins de mémoire et moins d'accès à la mémoire. Dans ce sens, plusieurs auteurs ont proposé des différents ordres de filtrage pour faciliter l'accès à la mémoire et pour donner la possibilité de traitement parallèle. Khurana et al. [108] ont proposé différents ordres de filtrage où on bascule entre le filtrage horizontal et le filtrage vertical. Sheng et al. [109] ont proposé un ordre avec une fréquence plus élevée de changement de direction verticale-horizontale. Li et al. [110] ont proposé une solution impliquant un degré de parallélisme où le filtrage vertical et horizontal se produisent en même temps (sur des blocs différents), cela implique l'accélération des traitements au prix d'avoir utilisé deux unités de filtrage. L'ordre de cette méthode est présenté dans la figure 5.15.

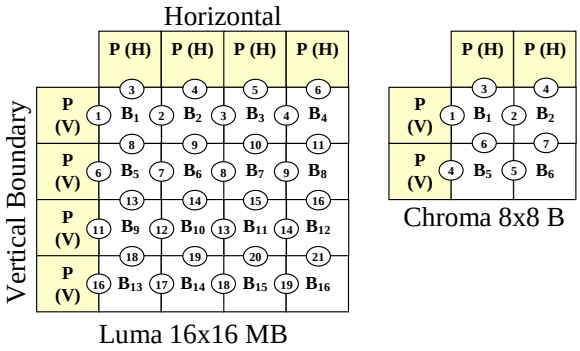



Figure 5.15. Ordre de filtrage proposé par Li et al. [110].

Chen et al. [111] ont proposé une stratégie de filtrage avec seulement 18 étapes (pour le signal de luminance) au lieu de 21 étapes utilisées dans [110], l'ordre proposé est présenté dans la figure 5.16. Dans l'implémentation matérielle de cette stratégie, les auteurs ont utilisé deux unités de filtrage (1D-unité), un registre 4×4pixels, un circuit de transposition et une SRAM 16×32bits.

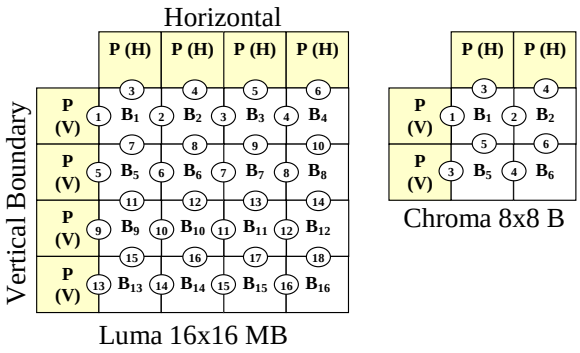



Figure 5.16. Ordre de filtrage proposé par Chen et al. [111].

### 3.2. Les implémentations matérielles proposées pour le filtre anti-blocs

Pour cette partie de la thèse, nous avons proposé trois nouvelles implémentations matérielles du filtre, selon l'ordre de filtrage donné par Chen et al. Nous avons commencé par la programmation de cette proposition en VHDL et en utilisant des données sur 32bits avec l'ajout d'une partie de gestion de mémoire (paragraphe 3.2.1). Nous avons utilisé 2 filtres élémentaires (1D-Unité) comme mentionné dans [111] pour réaliser les filtrages Vertical et Horizontal en même temps (en parallèle) sur des blocs différents.

Dans une première proposition et en utilisant le même principe que la précédente, nous avons eu l'idée de subdivision des 2 filtres élémentaires (Vertical et Horizontal) en 4 filtres directionnels chacun dans une direction d'une frontière (Vertical-Droite, Vertical-Gauche, Horizontal-Haut et Horizontal-Bas). Cette technique ne change rien dans l'implantation des filtres élémentaires mais offre une meilleure gestion de la mémoire, ce qui nous a donné l'idée d'ajouter un $5^{eme}$ filtre directionnel. Ce filtre est utilisé spécialement pour le filtrage des frontières Vertical-Droite des blocs de voisignage. Avec cette technique nous évitons au maximum les multiplexeurs ainsi que le circuit de transposition.

Dans une deuxième implémentation, nous avons dupliqué l'implémentation initiale (à base de 4 filtres directionnels) pour le traitement des données sur 128bits (4×4pixels), nous avons constaté que le nombre de ressources matérielles utilisées dans cette implémentation n'a pas dupliqué de même ordre que la duplication des données (4×32). En effet, en utilisant des données sur 128bits la partie de contrôle d'une frontière sera la même pour les 4 lignes de 4 pixels dans le même bloc, en plus le traitement des 4 lignes en même temps implique des simplification dans les équations du filtre et une meilleur gestion de la mémoire. Ensuite dans une troisième implémentation, nous avons ajouté un $5^{eme}$ filtre comme dans l'implémentation en 32bits. Avec l'utilisation des données de 128bits et l'utilisation des 5 filtres directionnels, nous évitons au maximum les registre tampans et les multiplexeurs ainsi que les circuits de transposition, ce qui donne la possibilité de pipelining. Cela a influé considérablement sur le nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc sans une augmentation significative des ressources matérielles.

#### 3.2.1. Stratégie d'implantation

La figure 5.17 montre le schéma de base proposé pour l'implantation du filtre anti-blocs. Dans le but d'intégrer le filtre dans un schéma complet de l'encodeur H.264, trois étapes sont nécessaires :

1. Définir une stratégie de gestion de la mémoire pour l'enregistrement du macrobloc en cours de traitement ainsi que les 8 blocs de voisinage. Il est nécessaire aussi de fixer le type de mémoire (BRAM/registre) à utiliser pour l'enregistrement de ces bloc.
2. Définir une stratégie de traitement en utilisants des modules élémentaires ainsi que la méthode de transfert des données entre ces modules (l'architecture globale du filtre).
3. Définir l'architecture matérielle des modules élémentaires (filtres directionnels) où les traitement sont effectués en 3 étapes ésentielles :
   - Sélection de BS ;
   - Prise de décision sur le type de filtre à appliquer ;
   - Réalisation de l'opération du filtre sur les pixels désignés.

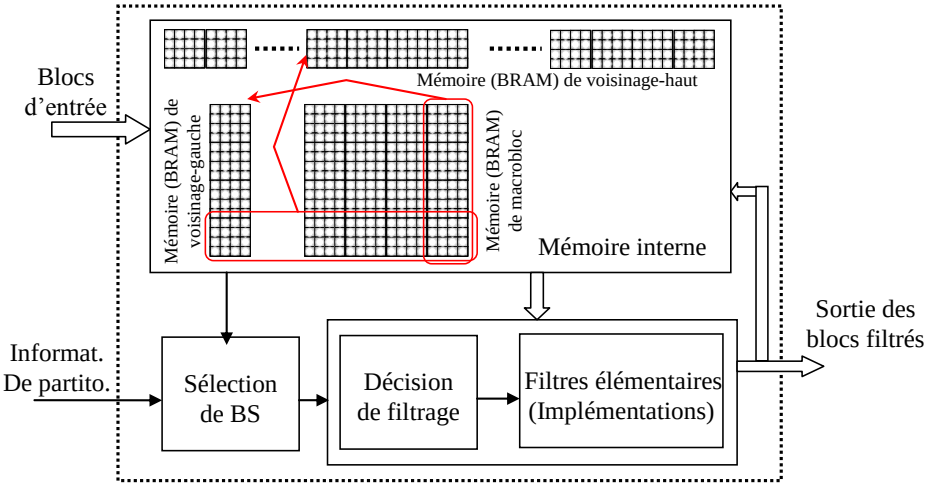



Figure 5.17. Schéma de base pour l'implantation du filtre anti-blocs utilisé dans H.264/AVC.

### 3.2.1.1. Gestion de la mémoire pour le filtre anti-blocs

Dans une telle implémentation, deux solutions sont possibles pour la gestion de la mémoire :

- Utilisation des mémoires internes pour l'enregistrement du bloc en cours de traitement ainsi que les blocs de voisinage ;
- Enregistrement des macroblocs codés/décodés par TQ/$QT^{-1}$ dans une mémoire externe, et relecture des blocs avec leurs blocs de voisinage pour chaque opération de filtrage. Cette opération peut être simplifiée par la lecture d'un macrobloc de taille 20×20pixels.

Pour nos implémentations, nous avons utilisé des mémoires internes pour l'enregistrement des 16 blocs en cours de traitement, des 4 blocs de voisinage-gauche et de l'ensemble des blocs de voisinage-haut dans la même ligne de l'image (Figure 5.17). Si par exemple nous manipulons des images au format CIF (288×352), il nous faut une mémoire de voisinage-haut avec une capacité de 88 blocs de 128pixels. Après chaque fin de traitement d'un macrobloc, les 4 blocs à droite (du macrobloc traité) seront enregistrés dans la mémoire de voisinage-gauche pour servir comme bloc de voisinage au macrobloc suivant, et les 4 blocs en bas (plus le quatrième bloc de voisinage-gauche) seront enregistrés dans la mémoire de voisinage-haut pour servir comme voisinage aux autres macroblocs en dessous (la ligne suivante des macroblocs). Cette stratégie de gestion de la mémoire a les avantages suivants :

- Limiter le nombre d'accès à la mémoire externe, ce qui implique qu'on gagne du point de vue temps de traitement, et par conséquent la réduction de la consommation d'énergie ;

- Permet de définir la stratégie désirée de lecture des données pour les différents modules de traitement, surtout avec la nature reconfigurable des mémoire BRAM ;
- Réutilisation des mêmes mémoires pour l'enregistrement des différentes données ;
- Permet de définir des mémoires avec des entrées/sorties de 128bits en utilisant les mémoires de 36Kbits des circuit FPGA Virtex5. Dans ce cas, chaque cellule de 128bits est un bloc, ce qui facilite la lecture et l'écriture de ces blocs par les modules élémentaires de traitement.

#### 3.2.1.2. Calcul des valeurs de BS

Les valeurs de BS pour chaque bloc sont supposées données par les modules de traitement en amont dans l'encodeur H.264 (les chaines de prédiction Intra et Inter), ces valeurs sont calculées en fonction du type de traitement effectué, en fonction de la position et le type du macrobloc et en fonction de la méthode de partitionnement. La procédure de calcul de BS est simple et se résume dans le diagramme suivant :

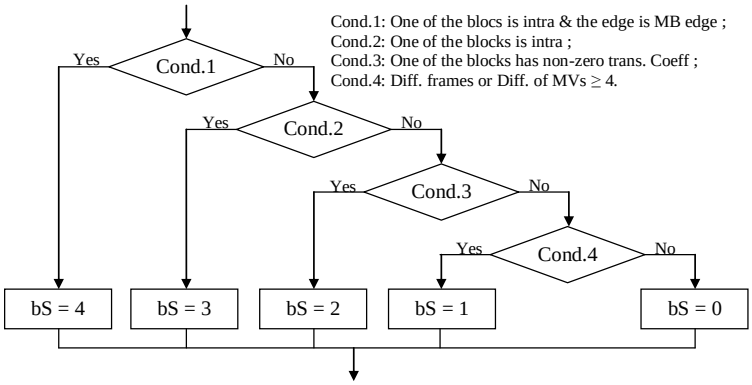



Figure 5.18. Organigramme de calcul des valeurs de BS.

Pour les implémentations matérielles proposées, il suffit de définir une mémoire de 16 éléments (3bits au minimum) pour l'enregistrement des valeurs de BS, ensuite définir une stratégie de lecture de ces valeurs pour chaque filtre directionnel. Avant chaque traitement, ces filtres doivent lire le bloc à traiter à partir de la mémoire d'entrée, les blocs de voisinage à partir des deux mémoires de voisinage et la valeur de BS à partir de la mémoire de BS ou bien à partir de la valeur de BS des filtres directionnels en amont. Les résultats de simulation dans les figures 5.26 et 5.27 montrent le principe de transfert des valeurs de BS entre les différents filtres directionnels.

### 3.2.2. Implémentation en 32bits à base de 5 filtres directionnels

Dans cette implémentation, les données manipulées sont organisées en 32bits, ce qui implique que les mémoires embarquées BRAM (36Kbits), utilisées pour l'enregistrement des blocs en cours de traitement ainsi que leurs blocs de voisinage, sont aussi organisés en 32bits. A la sortie du filtre Vertical-Haut, un buffer 4×4pixels est ajouté pour l'enregistrement des 4lignes de pixels traités. Ce buffer est utilisé aussi comme circuit de transposition. En effet, après 4 enregistrements les pixels dans ce buffer sont transférés (avec transposition) dans un autre buffer 4×4 pour servir comme entrée aux 2 filtres horizontaux. Deux autres buffers sont utilisés aux deux sorties de ces filtres : le premier pour l'enregistrement des lignes de pixels à la sortie du filtre Horizontal-Bas avant leurs transferts en sortie du filtre, le deuxième pour l'enregistrement des lignes de pixels traités à la sortie du filtre Horizontal-Haut avant leurs transferts dans la mémoire de voisinage-gauche.

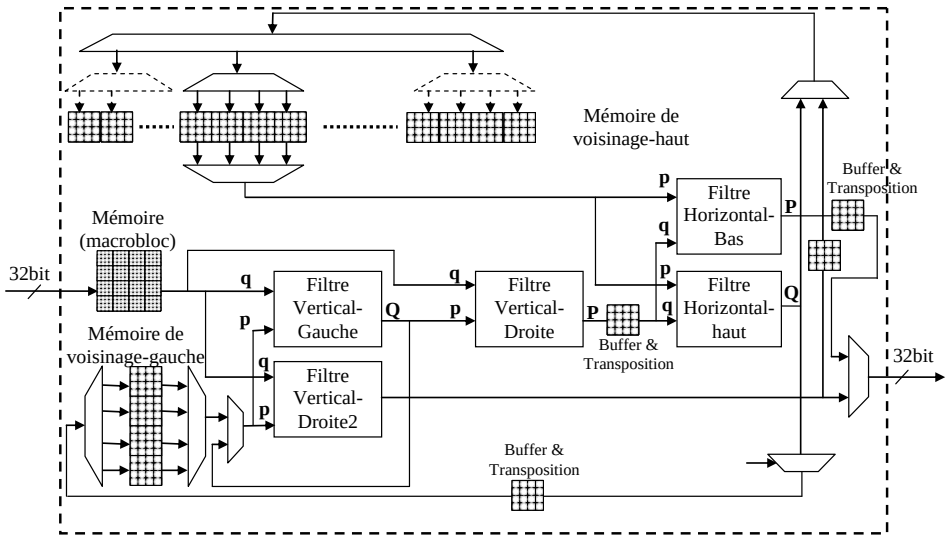



Figure 5.19. Implémentation matérielle proposée avec 32bits/5filtres élémentaires.

Avec l'utilisation des données en 32bits nous économisons les ressources matérielles dans les filtres élémentaires, mais la partie adressage devient plus compliquée. Nous remarquons aussi une augmentation du nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc. En plus, dans une telle implémentation, l'utilisation des registres intermédiaires 4×4 pixels et les circuits de transposition (Horizontal-Vertical et Vertical-Horizontal) est obligatoire. Néanmoins cette implémentation présente les inconvénients suivants :

- La taille des compteurs (d'adressage) est plus grande, dans ce cas nous comptons les lignes de 4pixels et non pas les blocs de 4×4pixels ;
- L'utilisation des compteurs en entrée et en sortie des filtres élémentaires pour la synchronisation des tâches. Dans le cas d'une implémentation à base de 128bits, il suffit d'utiliser un seul compteur en entrée ;

Dans cette implémentation, l'ajout d'un cinquième filtre directionnel réduit le nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc à 115 cycles au lieu de 120 dans les implémentations à base de 2 filtres élémentaires (4 filtres directionnels). Le nombre de LUTs consommés reste important à cause de l'utilisation des circuits de transposition et des registres tampons. La figure suivante montre le timing et l'ordre de traitement des lignes de 4pixels par les 5 filtres directionnels.

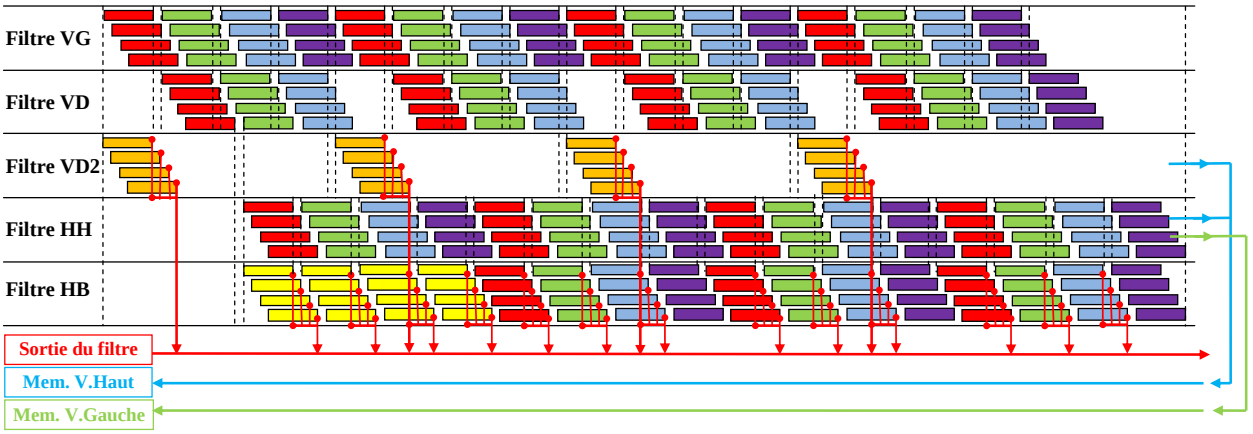



Figure 5.20. Ordre et timing de traitement dans l'implémentation 32bits/5filtres directionnels.

### 3.2.3. Implémentation en 128bits à base de 4 filtres directionnels

Dans cette proposition, nous avons utilisé uniquement 4 filtres directionnels en cascade pour assurer un traitement parallèle des blocs (Figure 5.21). Les deux filtres directionnels (Vertical-Gauche et Vertical-Droite) sont exécutés en parallèle sur des blocs différents, le même fonctionnement pour les deux autres filtres directionnels (Horizontal-Haut et Horizontal-Bas). Pour assurer l'ordre de filtrage dans ces filtres directionnels avec des données multiples dans plusieurs mémoires, nous utilisons au moins 5 multiplexeurs (128bits), au lieu de 3 multiplexeurs (32bits) utilisés dans la première implémentation.

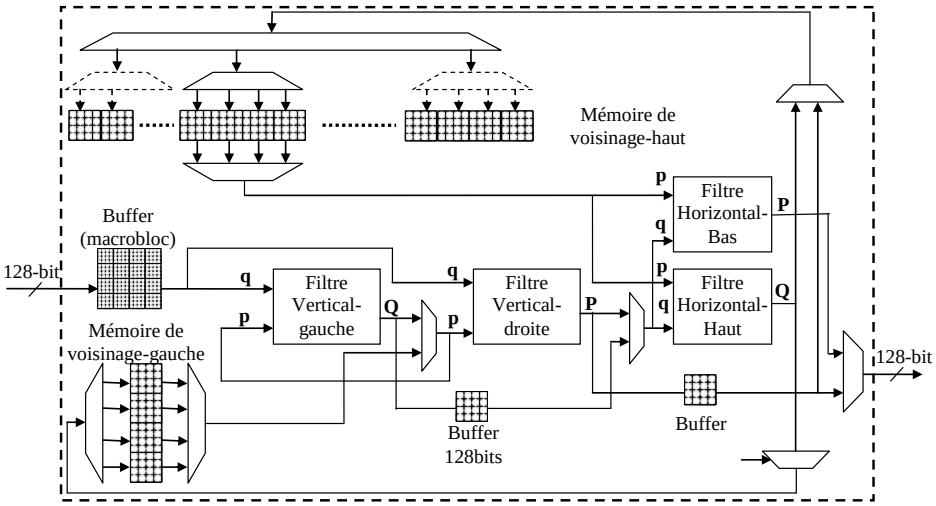



Figure 5.21. Implémentation matérielle proposée avec 128bits/4filtres élémentaires.

Au contraire à la première implémentation, les blocs B4, B8, B12 et B16 (Figure 5.14) sont enregistrés dans des buffers de 128bits avant leurs filtrages haut et bas, pour donner l'avantage aux blocs de voisinage-gauche à être traité par le filtre Horizontal-Droite (Figure 5.22). A ce moment les filtres Vetical-Haut et Vertical-Bas seront occupés par le traitement des blocs précédents, et le traitement des blocs enregistrés dans le buffer est déclenché par la sortie de validation de ces deux filtres. La figure 5.22 montre aussi le timing et l'ordre de traitement des blocs par les 4 filtres directionnels.

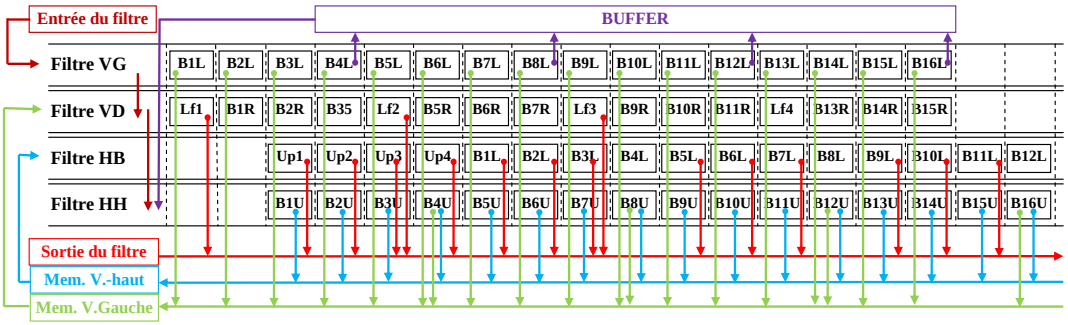



Figure 5.22. Ordre et timing de traitement dans l'implémentation 128bits/4filtres directionnels.

### 3.2.4. Implémentation en 128bits à base de 5 filtres directionnels

Dans cette implémentation, nous avons utilisé les mêmes 4 filtres directionnels que l'implémentation précédente avec en plus un 5$^{eme}$ filtre directionnel (Vertical-Droite2), ce filtre est ajouté spécialement pour les blocs de voisinage gauche (Figure 5.23). En utilisant cette structure, si par exemple, les deux filtres horizontaux manipulent le bloc B1, les deux autres filtres (Vertical-Droite et Vertical-Gauche)

manipulent les blocs B2 et B3 successivement. Ce qui implique un traitement parallèle des blocs. Les deux filtres (Vertical-Droite et Vertical-Droite2) travaillent en alternative. Si le premier fonctionne l'autre serait en arrêt en forçant l'entrée de sélection. Si le deuxième fonctionne, le premier serait désactivé en forçant la valeur de BS à zéro (pas de filtrage). Cela est valable pour les blocs B4, B8, B12 et B16 (Figure 5.14) où les bords droits ne sont pas filtrés. L'utilisation d'un filtre directionnel supplémentaire a comme avantages :

- Les données dans les 5 filtres directionnels seront connectées directement sans besoin des multiplexeurs. Les conditions (les multiplexeurs) seront uniquement sur les entrées de sélection ;
- La désignation des blocs complètement traités (pour la sortie) et les blocs intermédiaires (pour les mémoires de voisinage) sera plus facile, ce qui facilite aussi le calcul des adresses mémoires pour ces blocs.

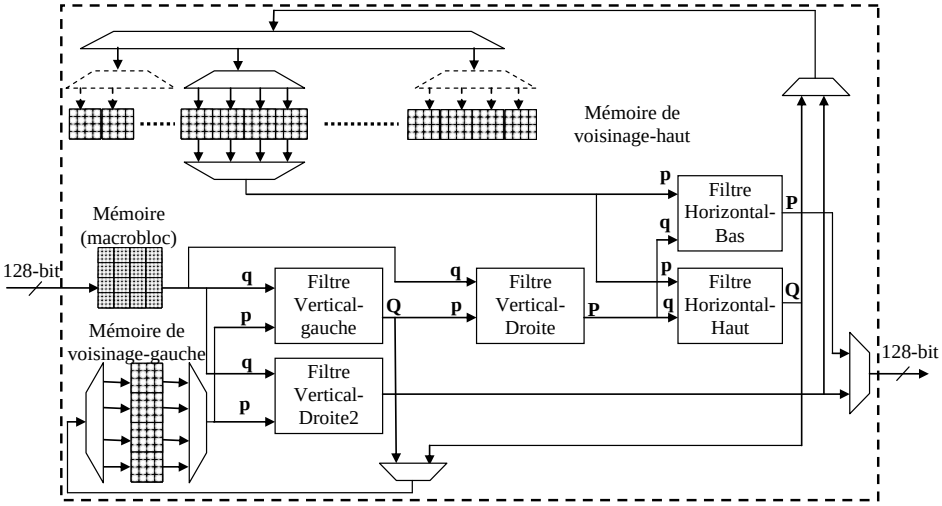



Figure 5.23. Implémentation matérielle proposée avec 128bits/5filtres élémentaires.

Sur la figure 5.24, nous montrons l'ordre et le timing de filtrage des différents blocs dans les 5 filtres directionnels. Nous montrons aussi l'ordre et le timing des données en sortie et les données intermédiaires enregistrés dans les mémoires de voisinage.

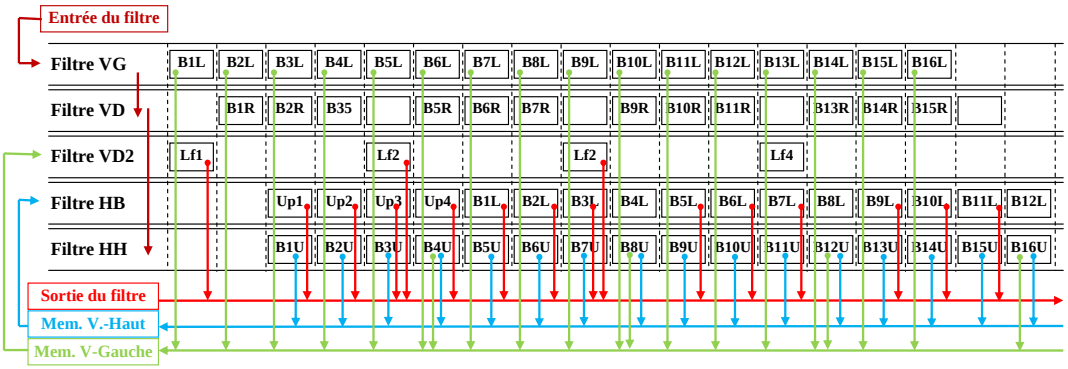



Figure 5.24. Ordre et timing de traitement dans l'implémentation 128bits/5filtres directionnels.

Les deux implémentations proposées, à base de 128bits, utilisent prèsque les mêmes principes, plusieurs idées de réalisations matérielles sont communes, comme par exemple :

1. Les blocs aux bords des images ne sont pas filtrés, ils ne possèdent pas de voisinage. Ce n'est pas le cas pour les blocs situés à l'intérieur des images avec des blocs de voisinage dans les quatre

directions. Cela provoque une irrégularité et, par conséquent, accroît la complexité de l'unité de contrôle du filtre. Afin d'éviter cette irrégularité, nous proposons d'appliquer le filtre pour tous les blocs et en jouant uniquement sur les valeurs de BS. Nous avons attribué des zéros aux pixels de voisinage et des zéro pour les valeurs de BS afin d'éviter le filtrage de ces bords sans provoquer une irrégularité dans l'unité de commande.

2. L'utilisation des multiplixeurs pour assurer des données et des adresses uniques pour les variables affectées aux blocs, cela implique une affectation aux mémoires internes lors de la synthèse. Ce qui implique la minimisation des LUTs utilisés.
3. Plusieurs données techniques pour les différentes implémentations matérielles sont résumées dans l'annexe B1.

### 3.2.5. Implantation des filtres élémentaires

Chaque filtre directionnels doit traiter l'un des blocs d'entrée (q ou bien p) suivant une direction, et cela suivant les valeurs des entrées (BS, Alpha, Beta et tc0) calculées dans le module principal. En utilisant l'organigramme de la figure 2.22, l'architecture interne est presque la même pour les quatre filtres directionnels. Les formules sont les mêmes, les seules différences résident dans :

- Le nombre de pixels à traiter (en fonction de l'ordre du filtre) ;
- Les pixels utilisés dans le traitement (en fonction de la direction du filtre) ;
- La formule utilisée (en fonction de la position du pixel et des valeurs de BS, Alpha, Beta et tc0).

Le schéma bloc des filtres élémentaires est donné sur la figure 5.24. Dans chaque filtre élémentaire, nous proposons l'utilisation des variables différentes pour chaque condition, cela augmente le nombre de slices-registres utilisés mais diminue considérablement le nombre des LUTs utilisés (éviter au maximum les multiplexeurs sur ces variables). Cela donne aussi la possibilité des architectures en pipeline. En effet, l'utilisation des variables (des registres) différentes pour chaque étape nous a permis d'injecter des données successives à l'entrée du filtre pour les traiter en pipeline sans chevauchement.

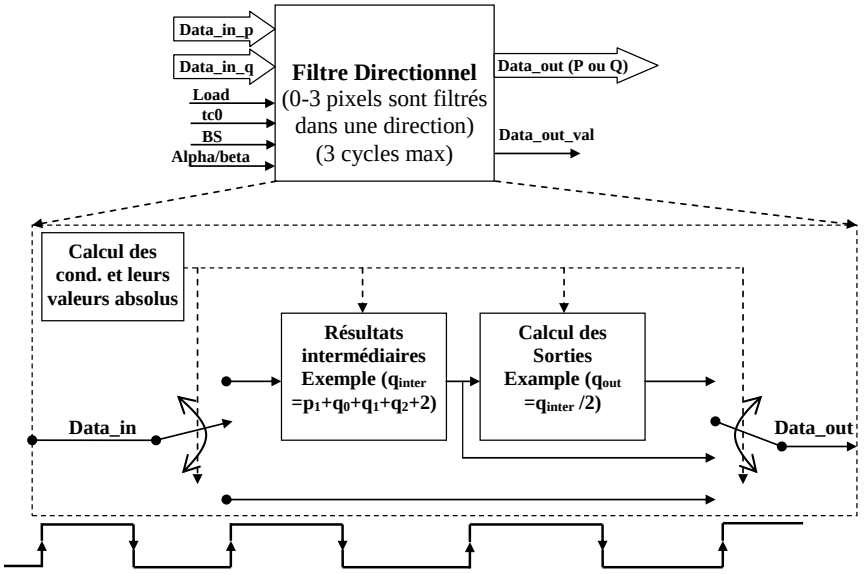



Figure 5.25. Schéma bloc des filtres directionnels proposés.

### 3.3. Résultats de simulation et de synthèse

Pour la vérification de nos implémentations, nous avons utilisé ISE9.2 pour la création et la gestion des projets ainsi que la synthèse des architectures, et ModelSim6.1 pour la simulation des résultats. Trois niveaux de simulation sont réalisés :

1. Simulation d'architecture : cette étape consiste à vérifier le fonctionnement global des architectures proposées (l'interaction entre les différents modules élémentaires). Un fichier testbench est écrit spécialement pour cette simulation ;
2. Simulation des fonctions : cette étape consiste à vérifier le fonctionnement des modules élémentaires. A ce niveau les architectures matérielles des modules sont définies et simulées ;
3. Simulation de l'implémentation avec des données réelles : à ce niveau les architectures matérielles des filtres directionnels sont simulées en utilisant un testbench qui permet de lire des fichiers des images bruitées (effet des blocs). Ce testbench permet aussi de collecter et réarranger les résultats. A ce niveau de simulation plusieurs conditions sont définies pour la synchronisation des données en entrées et en sortie.

La figure 5.26 montre les résultats de simulation de l'implémentation proposée à 4 filtres directionnels avec des données sur 128bits. Le testbench permet de lire un fichier image, le transférer bloc par bloc à l'entrée du filtre ainsi que la récupération et la réorganisation des blocs filtrés. Les chronogrammes réalisés par ModelSim, nous permettent de suivre les déplacements des blocs jusqu'à la sortie du filtre, ceci dans le but de vérifier les différentes étapes de filtrage. Ces chronogrammes nous permettent aussi de calculer le temps de réponse de chaque module élémentaire ainsi que le nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc. En effet, pour cette implémentation, le nombre de cycles nécessaires pour le traitement d'un macrobloc variant entre 59 et 74 cycles.

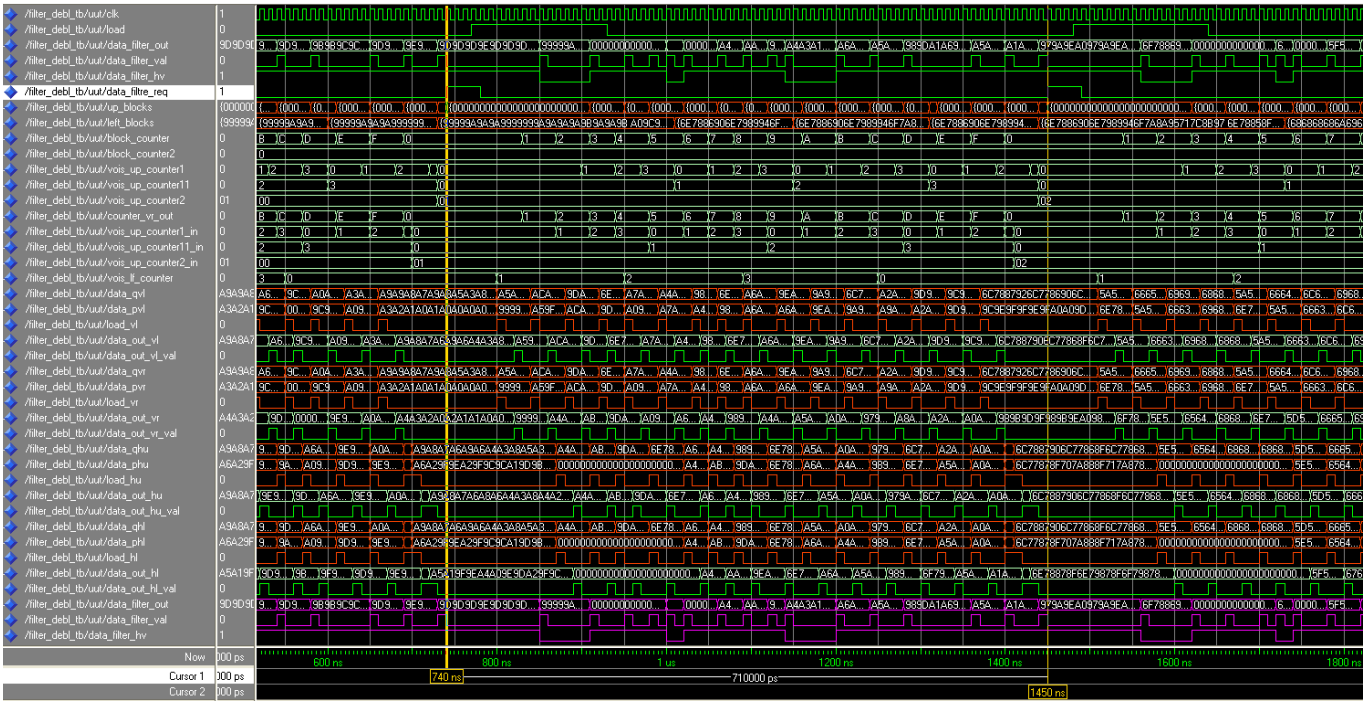



Figure 5.26. Résultats de simulation des filtres élémentaires pour l'implémentation 128bits/4filtres.

La figure 5.27 montre les résultats de simulation de l'implémentation proposée à base de 5 filtres directionnels et avec des données sur 128bits. Le nombre de cycles nécessaires pour le traitement d'un

macrobloc, dans cette proposition, variant entre 55 et 71 cycles d'horloge. Sur ce chronogramme, nous remarquons que :

- Lors de la 4$^{eme}$ impulsion du signal « laod_vl », la valeur BS_VR bascule à zéro et retourne à sa valeur initiale, cela signifie qu'on n'applique pas de filtrage Horizontal-Droite sur le quatrième bloc (B4 dans la figure 5.14). la même remarque pour les blocs B8, B12 et B16. Ce basculement dans la valeur BS_VR est assuré par l'utilisation d'une variable supplémentaire BS_VR3.
- L'adressage de la mémoire de voisinage-haut commence par 1 et non pas par 0, pour avoir une capacité de quatre lignes d'images plus un bloc supplémentaire d'adresse 0.

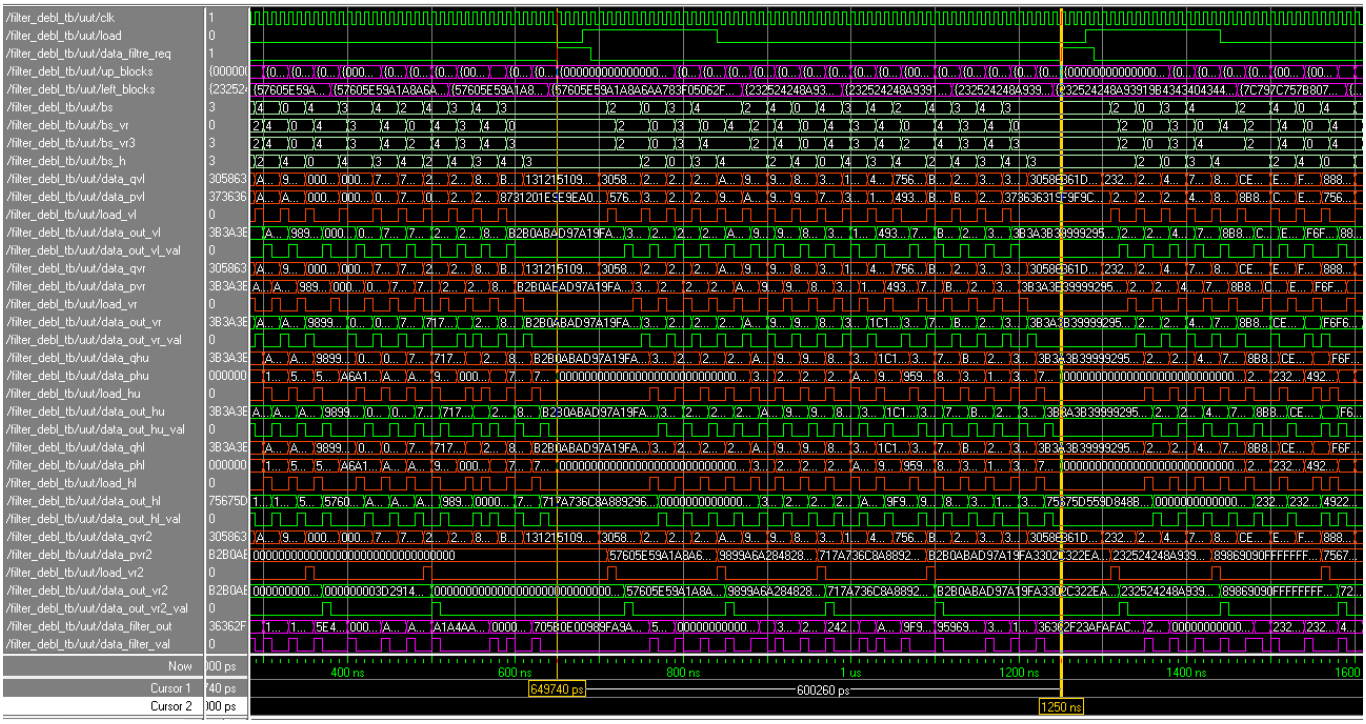



Figure 5.27. Résultats de simulation des filtres élémentaires pour l'implémentation 128bits/5filtres.

Le tableau 5.4 montre les résultats de synthèse des architectures proposées en utilisant les deux plateformes de prototypage ML501 et XUPV5. Ce tableau donne aussi une comparaison entre les trois implémentations basées sur des données de 32bits et de 128bits.

| | Filtre anti-blocs avec 32bits/5filtres élémentaires | | Filtre anti-blocs avec 128bits/4filtres élémentaires | | Filtre anti-blocs avec 128bits/5filtres élémentaires | |
|---|---|---|---|---|---|---|
| | Utilisé | % | Utilisé | % | Utilisé | % |
| Virtex5 ML501 (LX50) | | | | | | |
| Nombre de Slices-Registres | 2 846 | 9% | 5 812 | 20% | 6 288 | 21% |
| Nombre de LUTs utilisés | 5 404 | 18% | 9 274 | 32% | 7 444 | 25% |
| Nombre de blocs RAM/FIFO | 3 | 6% | 10 | 20% | 13 | 27% |
| Virtex5 XUPV5 (LX110T) | | | | | | |
| Nombre de Slices-Registres | 2 846 | 4% | 5 812 | 8% | 6 286 | 9% |
| Nombre de LUTs utilisés | 5 404 | 7% | 9 274 | 13% | 7 506 | 10% |
| Nombre de blocs RAM/FIFO | 3 | 2% | 10 | 7% | 12 | 8% |
| Fréq. max. de fonctionnement | 189,139MHz | | 165,44MHz | | 170,95MHz | |
| Nbre de cycles d'horloge/MB | 115 Cycles/MB | | 59 – 74 Cycles/MB | | 55 – 71 Cycles/MB | |

Tableau 5.4. Résultats de synthèse du module de filtrage (filtre anti-blocs).

Nous pouvons déduire de ce tableau que la troisième implémentation proposée (128bits / 5 filtres directionnels) possède les avantages suivants : la réduction de l'usage des mémoires tampons dans les modules ; l'élimination des circuits de transposition dans le filtre ; la possibilité de traitement en parallèle sans l'augmentation de ressources matérielles ; la possibilité de traitement en pipeline, ce qui augmente la fréquence des traitements. En effet, en passant d'une implémentation de 32bits à 128bits, cela ne cause qu'une légère augmentation des ressources en faveur d'une réduction importante du nombre de cycles nécessaires au traitement d'un macrobloc. En effet, le nombre de cycles d'horloge nécessaires au traitement d'un macrobloc est ramené à 55 cycles au lieu de 115 cycles, ce qui représente un gain plus de 100%.

### 3.4. Utilisation du filtre anti-blocs avec des images réelles

Un testbench créé spécialement pour les implémentations proposées, nous a permis de lire les fichiers images, de les transférer dans l'accélérateur matériel, et de les récupérer et les réarranger dans un autre fichier. A l'aide des fonctions Matlab, nous pouvons lire ces fichiers et afficher les différentes images pour vérifier le fonctionnement du filtre. La figure suivante montre le filtrage des 2 images de Lena (256×256 Pixels) bruitées en ajoutant l'effet des blocs. La première image filtrée est une image après compression-décompression avec un taux de compression de 1/4 et la deuxième image avec un taux de compression de 1/16. Sur ces figures, nous remarquons que le filtre est hautement adaptatif (comme mentionné dans le logiciel de référence), il permet le filtrage et le lissage des surfaces loin des contours d'image, sur ces contours le filtrage est conditionné par plusieurs paramètres (force de frontière).


Figure 5.28. Utilisation du filtre anti-blocs sur des images réelles.

### 3.5. Conclusion

Dans cette partie de notre travail, nous avons proposé des implémentations matérielles pour le filtre anti-blocs utilisé dans les codecs H.264/AVC. Après une étude détaillée de ce filtre, nous avons constaté que la complexité du filtre ne réside pas dans l'implantation des filtres élémentaires mais dans la gestion de la mémoire pour ces filtres ainsi que dans la décision sur l'ordre de filtre à implémenter (la partie de contrôle). Pour que nos architectures soient les plus efficaces, nous suggérons que les blocs de voisinage soient enregistrés dans des mémoires internes pour éviter l'accès intensif à la mémoire externe, pour cela nous avons implanté une nouvelle stratégie de gestion de la mémoire interne (BRAM) pour le filtre anti-blocs. Nous avons, ensuite, proposé des implémentations matérielles pour le filtre selon plusieurs stratégies en variant la taille du bus de données et le nombre de filtres directionnels utilisés. Les résultats de synthèse montrent que l'utilisation de 5 filtres directionnels avec un bus de 128bits donne un meilleur compromis entre les ressources matérielles utilisées, la fréquence de traitement et la fréquence maximale de fonctionnement. En effet, une telle implémentation utilise 21%, 25% et 27% des slices-registres, des LUTs et des mémoires BRAM respectivement en utilisant la plateforme ML501. En utilisant la plateforme XUPV5 ces résultats deviennent 9%, 10% et 8%.

Les trois propositions produisent la même qualité des images filtrées. Par contre l'ajout d'un cinquième filtre directionnel dans la troisième proposition, avec des données en 128bits, donne les avantages suivants :

- Les filtres élémentaires sont liés directement sans faire appel aux multiplexeurs. Par conséquent, le nombre de LUTs utilisés dans la troisième proposition a diminue. En effet, selon les résultats de synthèse, le nombre de luts est diminué de 24% malgré l'utilisation d'un filtre supplémentaire.
- Le nombre de cycle d'horloge nécessaire pour le traitement d'un MB est diminué de 4 cycles dans la troisième implémentation par rapport à la deuxième et de plus de 100% par rapport à la première implémentation.

Malgré que le filtre soit le module le plus petit de l'encodeur H.264, il reste parmi les modules les plus compliqué à cause de la partie de contrôle et de gestion de la mémoire. Il consomme lui-seul 25% des LUTs en utilisant la plateforme ML501, contre 31% pour tous les modules de la chaine de prédiction Intra.

Les trois implémentations proposées sont présentées dans le but de montrer les intérêts d'utilisation des données sur 128bits ainsi que les avantages d'ajouter un cinquième filtre directionnel. Nous pouvons choisir entre ces trois implémentations suivant les contraintes imposées par l'application à réaliser. Nous avons aussi utilisé ces trois implémentations comme un exercice pour la reconfiguration dynamique [112], surtout avec des entrées/sorties qui changent d'une implémentation à une autre.

## 4. Les autres modules de l'encodeur H.264

En plus des modules implantés de l'encodeur H.264/AVC, il reste les modules d'estimation et de compensation de mouvement ainsi que le module de codage entropique. Pour l'estimation et la compensation de mouvement, plusieurs travaux ont été réalisés et implantés [113-117] sur des plateformes multi-composants avant même l'apparition de la norme. En effet, l'estimation de mouvement est utilisée dans toutes les normes de compression vidéo. Plusieurs efforts ont été

concentrés pour ce module dans le but d'implanter des applications diverses sur des plateformes reconfigurables. Un exemple complet de programme VHDL, avec les fichiers de simulation et de synthèse, est disponible sur le site web d'opencores [15]. La même remarque pour le module de codage entropique, des programmes VHDL sont disponibles en open-sources.

Les figures 5.29 et 5.30 montrent le taux d'utilisation des ressources FPGA (LUTs et slices-registres consommés par les modules de l'encodeur H.264/AVC) en utilisant les deux plateformes ML501 et XUPV5. Après l'implantation des modules de la chaine de prédiction Intra et le filtre anti-blocs, il reste 44% des LUTs et 50% des slices-registres sur la plateforme ML501. De même, il reste 77% des LUTs et 79% des slices-registres sur la plateforme XUPV5. Ce qui rend cette dernière plateforme la plus adaptée à l'implantation des autres modules de l'encodeur H.264 sur un SoC. Une opération de raffinement des implémentations est toujours nécessaire dans le but de minimiser le nombre des ressources utilisées dans les différents modules.

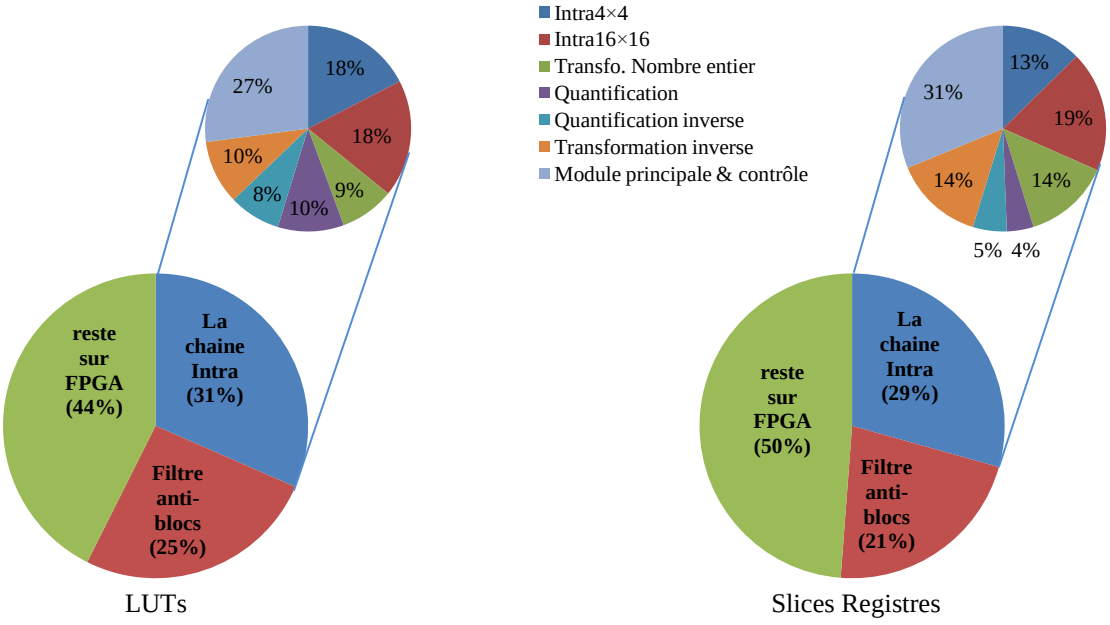



Figure 5.29. Taux d'utilisation des ressources FPGA (Plateforme ML501).

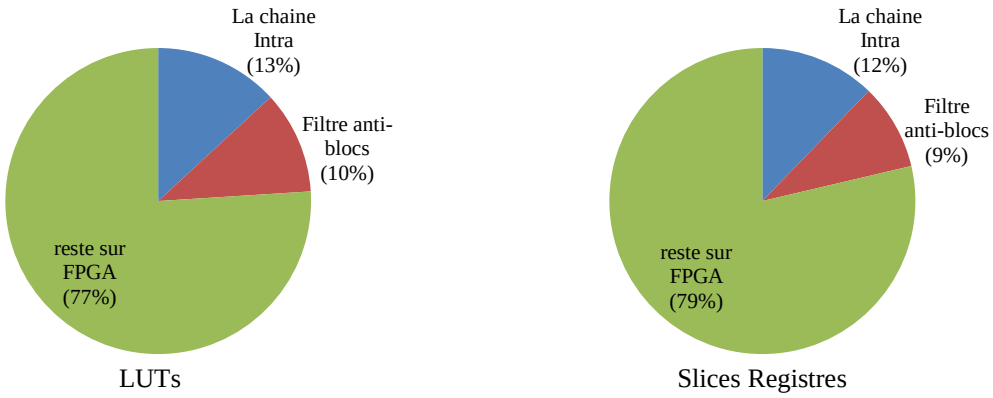



Figure 5.30. Taux d'utilisation des ressources FPGA (Plateforme XUPV5).

## 5. Conclusion

Dans ce chapitre, nous avons montré l'utilité de la parallélisation par les données dans le cas particulier de la norme H264/AVC. En effet, dans l'encodeur H.264, nous avons souvent besoin de traitements de blocs (4×4) d'où l'intérêt de travailler sur des tailles de données de 128bits au lieu de 32bits habituellement. Dans ce sens, nous avons proposé des implémentations matérielles pour les modules de la chaine de prédiction Intra et pour le filtre anti-blocs. Ces propositions nous ont permis d'avoir des gains en ressources et en mémoire utilisées ainsi qu'une augmentation de la fréquence des traitements. Les implémentations proposées nous permis aussi une réduction du temps d'exécution (nombre de cycles d'horloge nécessaire pour le traitement d'un macrobloc) et cela par l'exploitation du pipeline (soit un gain de 43% pour le module Intra et plus que 100% pour le module du filtre anti-blocs). Tous ces résultats ont été confirmés par la synthèse et l'implantation en utilisant les deux plateformes multi-composants ML501 et XUPV5 de Xilinx.

Pour la conception et la synthèse des implémentations matérielles, nous avons utilisé l'outil ISE de Xilinx. Pour la simulation, ISE fait l'appel de ModelSim à travers des testbenchs conçus pour chaque module. A la fin de ces implémentations, nous obtenons les fichiers (.ngc) des propositions pour les intégrer, par la suite dans le chapitre 6, dans des schémas à base de processeurs MicroBlaze en utilisant l'outil EDK. En perspective à ce travail, nous expérimentons actuellement la reconfiguration dynamique sur les différents modules.

# Chapitre 6

# Implantation Logicielle/ Matérielle de l'Encodeur H.264/AVC

## 1. Introduction

Ce chapitre est consacré à l'implantation matérielle de l'encodeur H.264/AVC (le codage Intra et le filtre anti-blocs) dans un environnement logiciel/matériel. La première étape consiste à la conception d'une plateforme matérielle d'acquisition et de restitution vidéo avant de passer à l'étape d'intégration des accélérateurs (la partie traitement). Dans un système de traitement vidéo temps réel, l'objectif consiste à avoir le maximum de temps libre entre l'acquisition et l'affichage au profil de la partie traitement. La plateforme matérielle réalisée s'articule autour de la carte XUPV5 de Xilinx avec plusieurs périphériques d'E/S, ce qui permet l'intégration des modules de l'encodeur H.264 dans le design sous forme d'IPs externes.

L'outil EDK de Xilinx permet la création d'un système à base d'un processeur MicroBlaze, de modules IPs de base (FIFO, contrôleur DMA pour Direct Memory Access, contrôleur DVI/VGA, etc.) ainsi que des IPs conçus pour des traitements spécifiques (Figure 6.1). Nous pouvons ainsi intégré autant de périphériques que l'on veut, n'étant limité que par le nombre de broches et de cellules logiques du circuit FPGA. EDK permet donc la création automatique du système, la gestion des drivers pour les périphériques ainsi que l'exécution du système d'exploitation pour le contrôle logiciel de la plateforme matérielle.

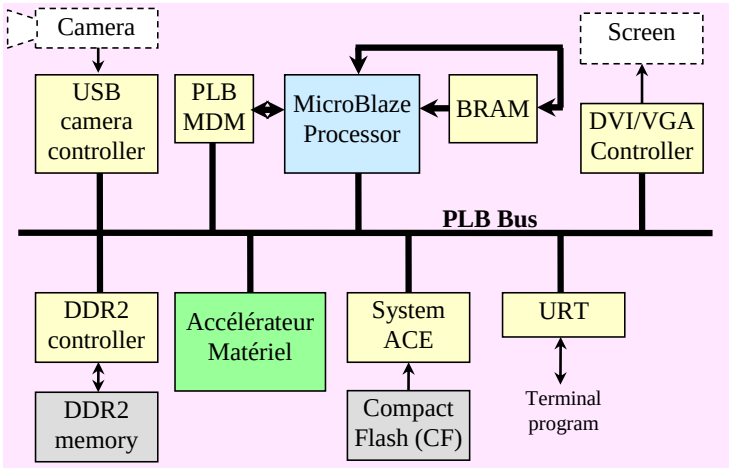



Figure 6.1. Plateforme matérielle pour l'implantation d'une application de traitement vidéo.

Après la création et la vérification de la plateforme matérielle, nous ajoutons les modules de traitement de l'encodeur H.264. Nous utilisons les modules implantés et synthétisés (fichiers .ngc) dans le chapitre 5. Une étape d'adaptation est nécessaire pour la liaison des entrées/sorties des modules au bus du système. Après le développement du matériel, l'ajout d'une coche logicielle est nécessaire pour le bon fonctionnement du système.

## 2. Préparation de la plateforme d'acquisition et de restitution vidéo

Pour la création d'un système embarqué (SoC/SoPC) en utilisant une plateforme reconfigurable, la première étape consiste à la création de la plateforme matérielle à base des IPs standards, cela est en fonction des besoins de l'application à implanter. La plateforme matérielle est constituée d'un ou bien de plusieurs processeurs ainsi que les périphériques connectés à ces processeurs à travers un bus. Pour la plateforme réalisée, nous utilisons un seul processeur (MicroBlaze de Xilinx) autour duquel sont associés plusieurs composants de base. Nous ajoutons par la suite les accélérateurs matériels réalisés. Pour la création du système, nous utilisons l'outil EDK9.2 (ou bien EDK12.2 récemment).

Dans la plateforme réalisée, la partie d'acquisition est remplacée par la lecture d'un fichier image à partir de la mémoire (Compact-Flash) en utilisant le contrôleur (System-ACE). Pour l'affichage nous utilisons un écran à travers le contrôleur DVI/VGA (XPS-TFT). Nous utilisons aussi une URT pour la communication série avec un hyper-terminal, et dans un premier temps, nous utilisons le contrôleur mémoire pour l'enregistrement des images dans la DDR2. Nous pouvons, également, utilisé le contrôleur mémoire proposé dans le chapitre 4 afin de diminuer le temps d'accès mémoire et augmenter les performances du système.

La partie logicielle, dans l'implémentation proposée, est limitée à la gestion des drivers des périphériques, au système d'exploitation et à un programme (.C) pour la gestion du système. Ce programme permet la lecture des pixels, leurs enregistrements dans la DDR2, leurs transferts (par bloc) dans les accélérateurs matériels ainsi que la récupération des pixels traités et leurs affichages ou bien leurs réorganisation dans la mémoire Compact-Flash. Il n'y aura pas, pour le moment, des modules de l'encodeur H.264/AVC réalisés en logiciel.

### 2.1. Création de l'architecture à l'aide de EDK

L'environnement XPS permet la création et la gestion d'un système sur puce complet tant au niveau matériel que logiciel. Il permet la génération de la plateforme matérielle en intégrant tous les modules IPs nécessaires. Après la synthèse, nous obtenons le fichier de programmation du circuit FPGA correspondant au design mais aussi un kit de développement logiciel qui comprend tous les fichiers (.h et .c) pour piloter les périphériques d'E/S. En effet, l'outil IDE (Integrated Development Environment) permet de créer des projets logiciels, de les compiler et de les déboguer, il permet aussi de télécharger le logiciel sur la carte. Toutes ces fonctionnalités permettent de rendre l'outil EDK un environnement complet de codesign. L'offre de codesign apparaît ici avec la possibilité de développer une partie de l'application par matériel ou bien de la réaliser en logiciel (programme en C exécuté par le processeur).

#### 2.1.1. Flux matériel

La première étape consiste à créer un projet dans EDK qui contient tous les modules matériels nécessaires en utilisant BSB (Base System Builder support). La bibliothèque des IPs dans EDK contient déjà tous les IPs de base nécessaires à la conception d'un système complet. En effet, l'utilisation de BSB permet la création d'un projet autonome, en sélectionnant le modèle et les références de la plateforme de prototypage FPGA. Selon le type et le nombre des processeurs utilisés, nous configurons et ajoutons les mémoires et les périphériques.

Au cours de la création du système, l'étape qui semble la plus importante est la partie de configuration du processeur MicroBlaze : la fréquence de fonctionnement (125MHz), la mémoire locale des données et des instructions (64Kb), l'unité de calcul de virgule flottante, etc. En plus du processeur MicroBlaze, l'outil EDK est capable de gérer d'autres IPs ainsi que leurs drivers. Pour la plateforme d'acquisition et de restitution vidéo réalisée, nous configurons un système composé de :

- Processeur MicroBlaze ;
- UART RS232 : protocole de communication série ;
- Contrôleur mémoire DDR2-SDRAM ;
- Contrôleur SystemACE-CompactFlash ;
- Bus (PLB, LMB) : Pour la connexion des périphériques avec le processeur MicroBlaze ;
- Timer : Permet d'obtenir des informations de synchronisation ;
- Générateur d'horloge et système de reset.

Nous ajoutons par la suite le contrôleur vidéo (XPS-TFT) ainsi qu'une interface XPS-IIC (Inter Integrated Circuit) spécialement pour ce contrôleur vidéo. Ces deux composants existent déjà dans la bibliothèque de EDK, il suffit de les ajouter et les connecter au bus PLB du système. La figure suivante montre les différents composants de la plateforme ainsi que leurs liaisons avec le bus PLB.

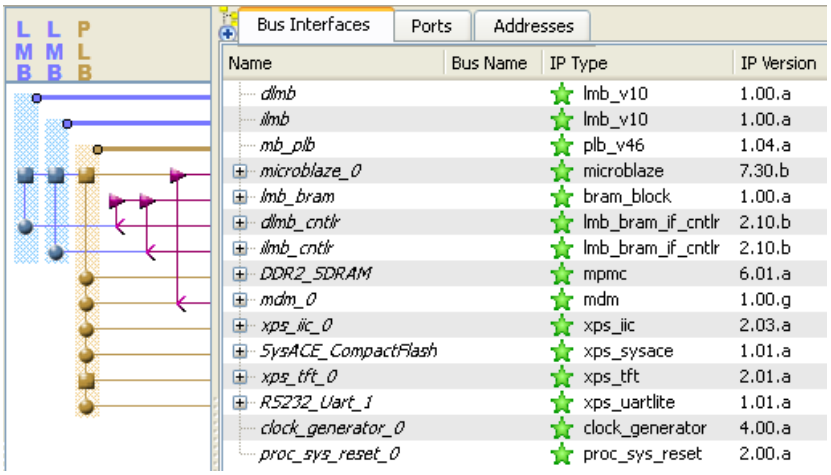



Figure 6.2. Système à base de processeur MicroBlaze.

Une fois tous les modules sont ajoutés et connectés au bus PLB, le système de génération d'adresse sera créé en spécifiant la plage d'adresses nécessaires pour chaque périphérique. Enfin, le système est prêt pour la synthèse pour produire tous les Netlists des modules.

| Instance | Base Name | Base Address | High Address | Size | Bus Interfac | Bus Name |
|---|---|---|---|---|---|---|
| microblaze_0's Address Map | | | | | | |
| dlmb_cntlr | C_BASEADDR | 0x00000000 | 0x0000FFFF | 64K | SLMB | dlmb |
| ilmb_cntlr | C_BASEADDR | 0x00000000 | 0x0000FFFF | 64K | SLMB | ilmb |
| xps_iic_0 | C_BASEADDR | 0x81600000 | 0x8160FFFF | 64K | SPLB | mb_plb |
| SysACE_CompactFlash | C_BASEADDR | 0x83600000 | 0x8360FFFF | 64K | SPLB | mb_plb |
| RS232_Uart_1 | C_BASEADDR | 0x84000000 | 0x8400FFFF | 64K | SPLB | mb_plb |
| mdm_0 | C_BASEADDR | 0x84400000 | 0x8440FFFF | 64K | SPLB | mb_plb |
| xps_tft_0 | C_SPLB_BASEADDR | 0x86E00000 | 0x86E0FFFF | 64K | SPLB | mb_plb |
| DDR2_SDRAM | C_MPMC_BASEADDR | 0x90000000 | 0x9FFFFFFF | 256M | SPLB0 | |

Figure 6.3. Les adresses affectées aux différents modules du système.

### 2.1.2. Flux logiciel

Une fois le design est créé, le XPS va générer les fichiers suivants :

- System.xmp : définie le fichier global du projet XPS ;
- System.mhs : définie le fichier de spécification matériel (hardware) du microprocesseur ;
- System.mss : définie le fichier de configuration logiciel (software) du microprocesseur ;
- System.ucf : définie le fichier d'implémentation des contraintes ;
- Applications logicielles de test : deux fichiers (.C) pour tester le fonctionnement des mémoires et des périphériques.


Figure 6.4. Les fichiers générés après la création du système.

Une plateforme logicielle est un ensemble de pilotes logiciels (les pilotes de périphériques), de bibliothèques (Generator tool configures librairies) et éventuellement un système d'exploitation sur lequel tourne une application, en plus des fichiers systèmes et des gestionnaires d'interruption pour le système de processeur embarqué en prenant MSS (Microprocessor Software Specification). Le processus est contrôlé par le processeur MicroBlaze grâce à un programme écrit en C. En effet, nous avons programmé 3 tâches différentes : la première pour la récupération des pixels en entrée et leurs enregistrements dans la mémoire DDR2, la deuxième consiste à lire ces pixels bloc par bloc, à partir de la DDR2, pour les modules de traitement, pour la troisième étape elle consiste à la récupération des blocs traités et leurs enregistrement dans une mémoire ou bien leurs affichages sur l'écran à travers le connecteur VGA/DVI.

Une application logicielle est stockée en mémoire en attendant son exécution par le processeur. Elle peut être stockée soit en mémoire interne du FPGA (BRAM) soit en mémoire externe (DDR par exemple). Une ou bien plusieurs applications logicielles peuvent être créées pour s'exécuter sur le MicroBlaze, ces applications apparaissent sous l'onglet Applications (Figure 6.5).

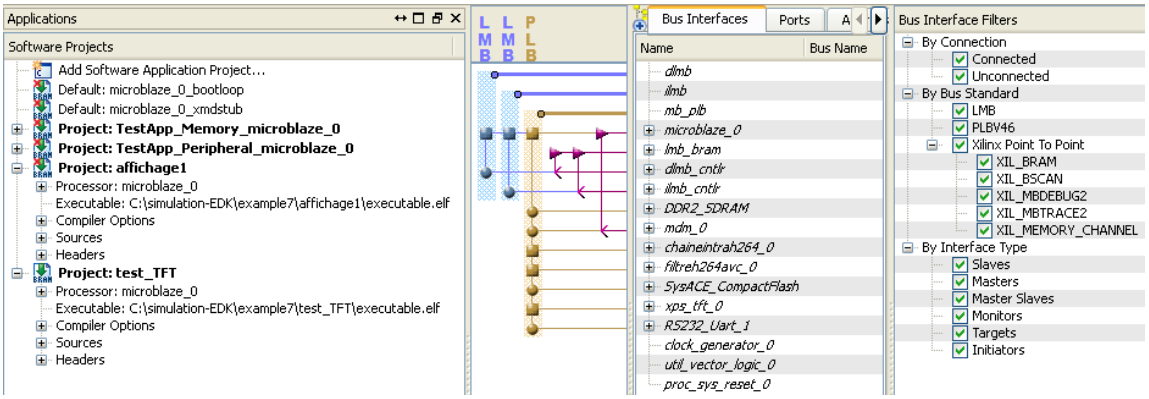



Figure 6.5. Plateforme logiciel pour le contrôle de la plateforme matérielle.

### 2.1.3. L'environnement EDK

Après l'ajout et la configuration des différents IPs dans le système, ainsi que la connexion de ces IPs au bus PLB et la génération des adresses, l'outil EDK permet la synthèse de l'ensemble du système (la création des fichiers .ngc global) ainsi que la création du Bitstream de la plateforme matérielle. Nous obtenons ainsi les différents résultats de synthèse (nombre de ressources matérielles utilisées) des différents modules dans le système. De même pour la plateforme logicielle, un Bitstream sera généré après avoir compilé les programmes et les bibliothèques associées.

Après la phase de synthèse du système et la génération des Bitstreams dans l'environnement EDK, l'étape suivante consiste à mettre en œuvre le système en utilisant l'outil graphique PlanAhead de Xilinx. Avec cet outil, nous pouvons produire les différentes configurations qui seront assemblés pour générer un Bitstream du système. La figure 6.6 présente l'interface graphique de l'outil EDK12.2 avec différentes fenêtres de description, de création, de synthèse, etc.

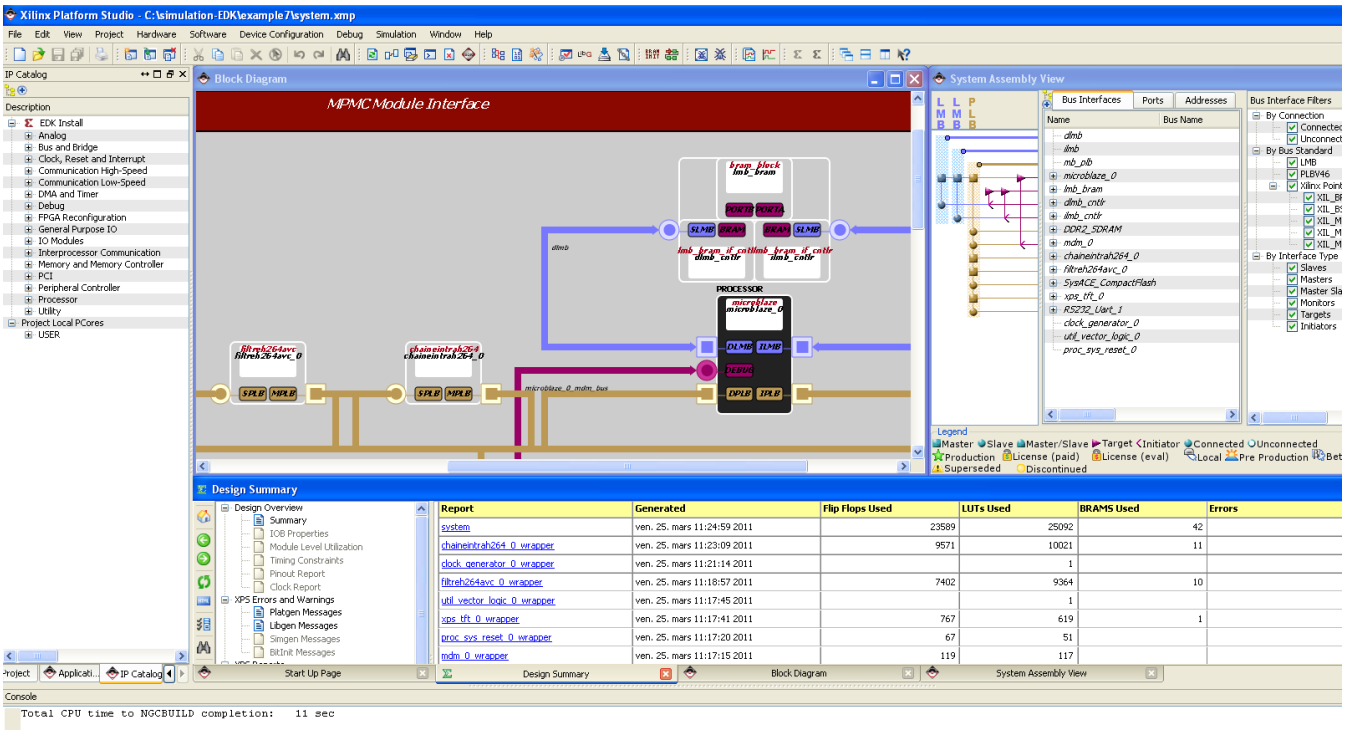



Figure 6.6. Interface graphique de l'outil EDK12.2 de Xilinx.

Le logiciel XPS fait appel à l'utilitaire Impact qui utilise le port USB pour programmer le circuit FPGA (matériel configurable) via l'interface JTAG du Virtex5. Le débogueur XMD utilise aussi l'interface JTAG pour télécharger le logiciel qui roulera sur le MicroBlaze. Le MicroBlaze pourra communiquer de manière bidirectionnel avec un PC via le port série (UART-COM1). En effet, le système envoie des données vers l'hyper-Terminal via le port série RS232 de la plateforme XUPV5.

## 2.2. L'acquisition des images

Dans la plateforme matérielle réalisée, la partie d'acquisition d'image est remplacée par la lecture d'un fichier image à partir de la mémoire Compact-Flash. En effet, en utilisant EDK, nous pouvons générer un système embarqué d'acquisition d'images à partir de l'un des périphériques externes de la plateforme multi-composants vers la mémoire DDR2. L'image peut être réorganisée (selon l'ordre de traitement des blocs dans l'image) avant son stockage dans la mémoire (Figure 6.7). Ce prétraitement

va nous permettre par la suite de faciliter la lecture des blocs de 128bits nécessaire pour les accélérateurs matériels. En effet, avant la lecture des pixels (bloc de pixels) en cours de traitement, le processeur doit calculer une adresse suivant la position de ces pixels dans l'image. Le système devrait également lire les blocs après le traitement pour être affiché ou bien enregistré dans une mémoire (interne ou bien externe). Dans ce sens, une étude basée sur la comparaison de PSNR sera possible.

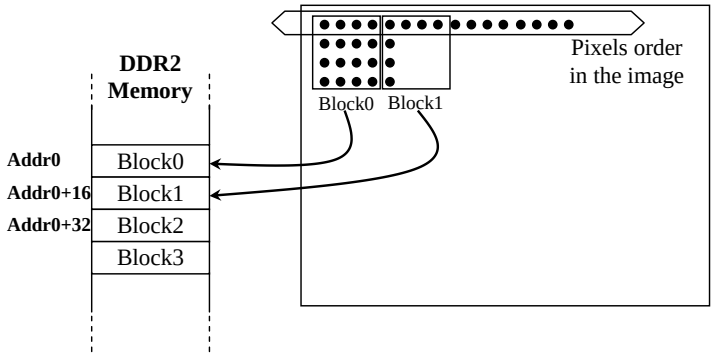



Figure 6.7. Réorganisation des images dans la DDR2 selon l'ordre des blocs.

Pour mesurer les erreurs de calcul entre les images originales et les images traitées, nous pouvons lire et écrire des fichiers image dans une mémoire externe (Compact-Flash par exemple). Ces fichiers peuvent être traité ensuite en utilisant le logiciel Matlab. La procédure suivante représente la partie logicielle (programme C) chargée de la lecture des images à partir de la mémoire Compact-Flash.

```
/* opens and reads from CF a 640x480 bitmap file, placing it in memory at baseaddr in a format for the tft */
int read_image(char FileName[], int baseaddr)
  {  unsigned char readBuffer[1920];
     SYSACE_FILE *infile;
     int i, j, numread, temp, writeaddr;
     HEADER header;
     INFOHEADER infoheader;
     infile = sysace_fopen(FileName, "r");

     if (infile)
        {       header.type = ReadUShort(infile);
                header.size = ReadUIntLil(infile);
                header.reserved1 = ReadUShort(infile);
                header.reserved2 = ReadUShort(infile);
                header.offset = ReadUIntLil(infile);
                infoheader.size = ReadUIntLil(infile);
                infoheader.width = ReadUIntLil(infile);
                infoheader.height = ReadUIntLil(infile);
                infoheader.planes = ReadUShort(infile);
                infoheader.bits = ReadUShort(infile);
                infoheader.compression = ReadUIntLil(infile);
                infoheader.imagesize = ReadUIntLil(infile);
                infoheader.xresolution = ReadUIntLil(infile);
                infoheader.yresolution = ReadUIntLil(infile);
                infoheader.ncolors = ReadUIntLil(infile);
                infoheader.importantcolors = ReadUIntLil(infile);

                /* Process the data */
                for (j=infoheader.height-1;j>=0;j--)
                  {  numread = sysace_fread(readBuffer, 1, 1920, infile);
                     for (i=0;i<infoheader.width;i++)
                       {  temp = ((((readBuffer[i*3+2] << 8) | readBuffer[(i*3)+1]) << 8) | readBuffer[(i*3)]);
                          writeaddr = baseaddr+(j*1024+i)*4;
                          XIo_Out32(writeaddr, temp);
                       }
                  }
                sysace_fclose(infile);
                return 1;
        }
     else {  return 0;  }
  }
```

## 2.3. L'interface DVI/VGA

Les écrans DVI/VGA couleurs sont des écrans RVB. Ces écrans reçoivent trois signaux analogiques de couleur (Rouge, Vert et Bleu) à partir desquels ils vont reproduire l'image. En plus de ces trois signaux, le connecteur vidéo achemine un signal d'intensité et des signaux de synchronisation horizontaux et verticaux à l'écran. Les deux plateformes multi-composants (ML501 et XUPV5) disposent déjà d'un connecteur de sortie DVI/VGA. Il suffit d'ajouter le contrôleur pour ce connecteur à partir de la bibliothèque des IPs. En effet, un composant matériel (xps.tft.2.0.1a) est disponible dans la bibliothèque des IPs de EDK12.2 (dans la rubrique IO-modules). Dans le système à base de processeur MicroBlaze, il suffit d'ajouter ce composant et de le connecter au bus PLB d'un côté et au convertisseur numérique/analogique du connecteur DVI/VGA de l'autre côté. Du côté du bus, ce composant est reconnu par le système à travers une adresse (une plage d'adresses) comme tous les autres périphériques, de l'autre côté ce composant est connecté aux entrées/sorties du circuit FPGA (les pins du port DVI/VGA) à travers le fichier UCF.

La figure suivante montre le contrôleur TFT ainsi que l'interface IIC utilisés dans la carte XUPV5. Ces deux composants permettent de transmettre les couleurs (Rouge, Vert, Bleu) et les signaux de synchronisation à l'écran à travers le connecteur DVI/VGA. Ils sont nécessaires pour établir la liaison entre le processeur et l'écran d'affichage (à travers le bus PLB).

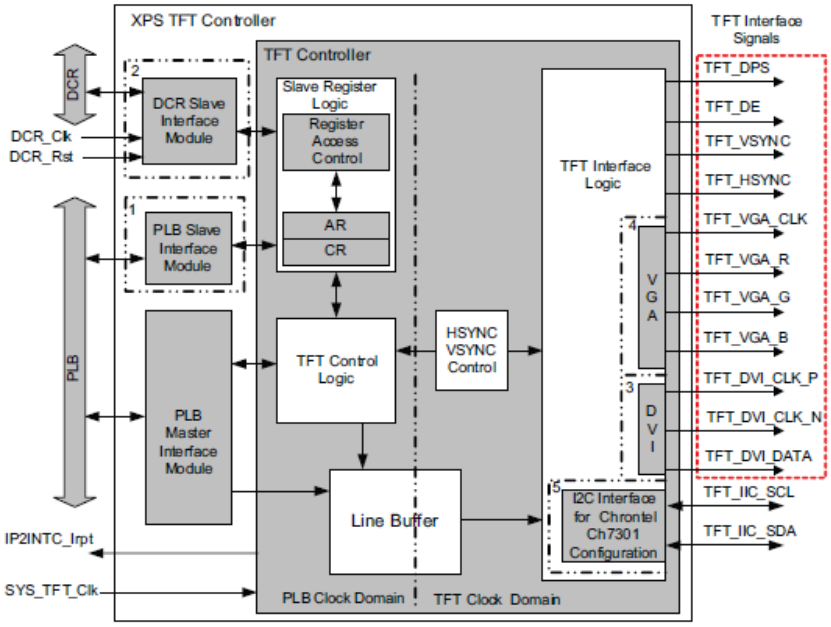



Figure 6.8. Contrôleur DVI/VGA et l'interface IIC dans la plateforme XUPV5 [118].

En plus de la partie matérielle du contrôleur DVI/VGA, une partie logicielle est nécessaire pour la gestion et la synchronisation des affichages. Nous avons utilisé les exemples donnés dans [119] et [120] pour ajouter, à notre programme, quelques fonctions pour la gestion et le contrôle d'affichage des images.

## 2.4. Résultats de synthèse de la plateforme

Le tableau suivant montre les résultats de synthèse de la plateforme matérielle réalisée pour l'acquisition et la restitution vidéo. Cette plateforme est composée de plusieurs composants autours du processeur MicroBlaze, à savoir : le contrôleur DDR2-SDRAM, le contrôleur System-ACE, le

contrôleur UART, le bus et les mémoires, le contrôleur TFT ainsi que l'interface IIC. Les pourcentages sont calculés par rapport à la carte de prototypage XUPV5. Sur cette carte, la plateforme proposée occupe 10% des slices et 8% des LUTs ainsi que 14% des mémoires BRAMs et 4% des DSP48E, ce qui donne la possibilité d'implanter des accélérateurs matériels pour l'encodeur H.264/AVC en utilisant le reste des ressources (Figure 6.9).

| | Slices | | LUTs | | BRAMs | | DSP48E | |
|---|---|---|---|---|---|---|---|---|
| | Utilisé | % | Utilisé | % | Utilisé | % | Utilisé | % |
| MicroBlaze | 1400 | 2,02% | 1360 | 1,97% | 0 | / | 3 | 4,68% |
| Mb_plb, ilmb, dlmb, dlmb_cntlr, ilmb_cntlr, lmb_bram (wrapper) | 170 | 0,24% | 459 | 0,66% | 16 | 10,81% | 0 | / |
| RS232_uart_1_wrapper | 152 | 0,22% | 133 | 0,19% | 0 | / | 0 | / |
| DDR_SDRAM_wrapper | 3694 | 5,34% | 2517 | 3,64% | 5 | 3,38% | 0 | / |
| Sysace_compactflash_wrapper | 213 | 0,30% | 101 | 0,15% | 0 | / | 0 | / |
| Mdm_0_wrapper | 119 | 0,17% | 117 | 0,17% | 0 | / | 0 | / |
| proc_sys_reset_0_wrapper | 67 | 0,10% | 51 | 0,07% | 0 | / | 0 | / |
| xps_iic_0_wrapper | 407 | 0,59% | 441 | 0,63% | 0 | / | 0 | / |
| xps_tft_0_wrapper | 759 | 1,10% | 610 | 0,88% | 1 | 0,67% | 0 | / |
| clock_generator_0_wrapper | 0 | / | 1 | / | 0 | / | 0 | / |
| Total ressources | 6981 | 10,10% | 5790 | 8,38% | 21 | 14,19% | 3 | 4,68% |

Tableau 6.1. Résultats de synthèse de la plateforme de traitement à base de processeur MicroBlaze.

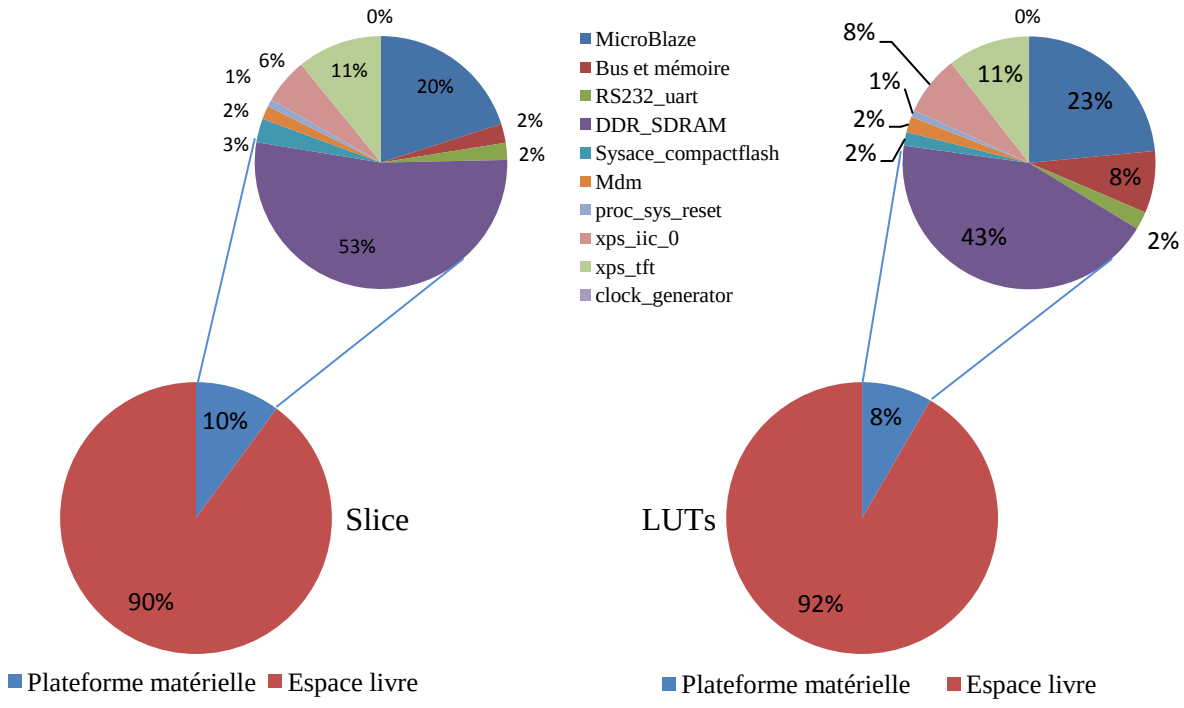



Figure 6.9. Taux d'utilisation des ressources matérielles de la plateforme de traitement.

La figure suivante montre les détails des implémentations (après le placement et le routage) des différents composants de la plateforme matérielle d'acquisition et de restitution vidéo. Nous remarquons une légère différence entre les résultats de synthèse et les résultats d'implantation.

| Module Name | Partition | Slices | Slice Reg | LUTs | LUTRAM | BRAM/FIFO | DSP48E | BUFG | BUFIO | BUFR | DCM_ADV | PLL_ADV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| system | | 190/4483 | 20/6236 | 234/5387 | 0/255 | 0/26 | 0/3 | 2/7 | 0/8 | 0/0 | 0/0 | 0/1 |
| DDR2_SDRAM | | 0/2097 | 0/3348 | 0/2413 | 0/92 | 0/9 | 0/0 | 0/0 | 0/8 | 0/0 | 0/0 | 0/0 |
| RS232_Uart_1 | | 0/104 | 0/131 | 0/120 | 0/19 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| SysACE_CompactFlash | | 0/100 | 0/202 | 0/97 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| clock_generator_0 | | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/4 | 0/0 | 0/0 | 0/0 | 0/1 |
| dlmb | | 0/1 | 0/1 | 0/1 | 0/1 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| dlmb_cntlr | | 0/8 | 0/2 | 0/6 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| ilmb | | 0/1 | 0/1 | 0/1 | 0/1 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| ilmb_cntlr | | 0/4 | 0/2 | 0/2 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| lmb_bram | | 0/0 | 0/0 | 0/0 | 0/0 | 0/16 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| mb_plb | | 0/181 | 0/94 | 0/260 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| mdm_0 | | 0/91 | 0/118 | 0/113 | 0/23 | 0/0 | 0/0 | 0/1 | 0/0 | 0/0 | 0/0 | 0/0 |
| microblaze_0 | | 0/852 | 0/1185 | 0/1173 | 0/20 | 0/0 | 0/3 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| proc_sys_reset_0 | | 0/30 | 0/33 | 0/23 | 0/2 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| system | | 16/16 | 0/0 | 64/64 | 64/64 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| xps_iic_0 | | 0/359 | 0/388 | 0/416 | 0/20 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| xps_tft_0 | | 0/449 | 0/711 | 0/464 | 0/13 | 0/1 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |

Figure 6.10. Résultats d'implantation de la plateforme d'acquisition et de restitution vidéo.

## 3. Intégration d'un accélérateur matériel dans un système à base de MicroBlaze

L'objectif de cette partie de la thèse consiste à l'intégration des accélérateurs, réalisés préalablement dans le chapitre 5, dans la plateforme matérielle d'acquisition et de restitution vidéo. Les accélérateurs peuvent être intégrés indépendamment ou bien ensemble. Le but est de tester le fonctionnement physique de ces accélérateurs, dans un système logicielle/matérielle, en utilisant des séquences vidéo en temps réel. En effet, le processeur MicroBlaze permet la lecture des images de la séquence, les injecter bloc par bloc dans les modules de traitement (accélérateurs matériels) et de récupérer les résultats, cela permet de comparer les images traitées par la suite. Dans les systèmes de traitement de la vidéo, il est nécessaire de tester les modules par couple (traitement-traitement inverse), comme dans l'exemple du paragraphe 3.2, et cela dans le but de comparer les images avant et après chaque traitement. D'après le flot de conception de EDK, plusieurs étapes sont nécessaires pour ajouter nos accélérateurs et cela dans le but d'obtenir une architecture embarquée de l'application (H.264/AVC ou bien l'un de ces modules de traitement).

### 3.1. L'ajout d'un accélérateur matériel

Le PLB est un élément du bus Core-Connect de IBM conçus pour le raccordement du processeur aux périphériques hauts performances en utilisant EDK. Afin que le processeur communique correctement avec les accélérateurs matériels proposés, nous ajoutons à chaque composant une couche d'interface appelée PLB-IPIF qui facilite la connexion au bus PLB. Cette couche est générée par un outil, fourni dans EDK, nommée « Create and Import Peripheral ». Une petite intervention dans le code généré est nécessaire pour instancier le composant avec l'interface. Avec la couche PLB-IPIF, nous utilisons deux FIFOs à l'entrée et à la sortie du module et nous constituons une machine à états finis (state machine) pour la synchronisation des lectures/écritures (FIFO-Module et Module-FIFO).

Les FIFOs sont créées séparément en utilisant l'outil Core-Generator de Xilinx. A la fin de la procédure de création, un fichier .ngc de la FIFO sera créé, et il suffit d'instancier ce fichier ainsi que les fichiers .ngc des modules de traitement (chapitre 5) dans la couche PLB-IPIF. Le résultat est un accélérateur matériel prêt à être ajouté au système. Après la connexion de tous les périphériques et les modules proposés, le processeur reconnaît chacun d'entre eux en fonction de son adresse dans le système.

### 3.2. Exemple : coprocesseur TQ/QT-1

L'accélérateur matériel dans ce cas est supposé composé de la transformation des nombres entiers, la quantification, la quantification inverse et la transformation inverse. Le but dans ce cas est d'appliquer

un traitement direct et inverse sur des images (dans la séquence) pour vérifier le bon fonctionnement des différents modules de traitement (implantation physique sur FPGA après l'étape de simulation). Nous utilisons une couche PLB comme montré sur la figure 6.11, cette couche est connectée d'un côté à l'accélérateur et de l'autre côté au bus PLB du système. Le système (le processeur) reconnaît l'accélérateur comme l'un des composants élémentaires avec une adresse bien choisie par le système.

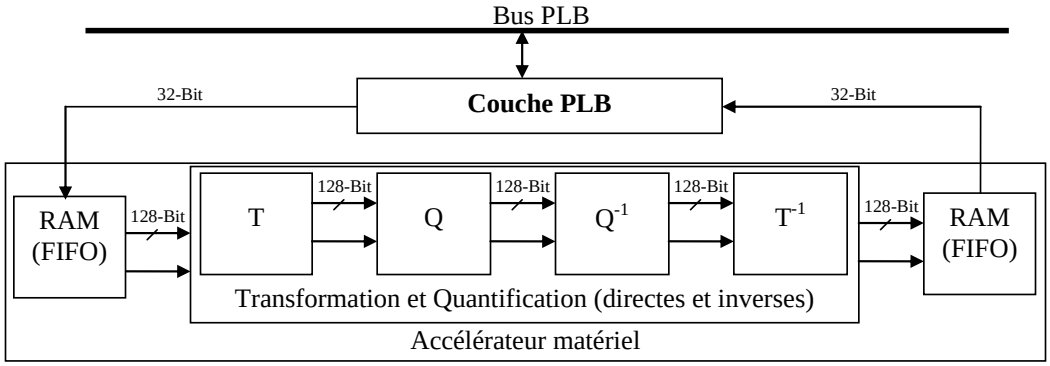



Figure 6.11. Intégration d'un accélérateur dans un système à base de processeur MicroBlaze.

L'accélérateur matériel opère sur des blocs d'image de taille 4×4pixels, ce qui représente 128bits. Afin d'adapter cet accélérateur (avec des entrées/sorties de 128bits) au reste du système à travers le bus PLB (32bits), nous utilisons des mémoires tampon de type FIFO. Ces FIFO servent à collecter les données avant leurs transferts vers les modules de traitement. Un programme C est écrit spécialement pour charger les données (blocs de pixels) dans la FIFO d'entrée en permanence et pour récupérer les données traitées, à partir de la FIFO de sortie, avant leurs transferts vers la DDR2 ou bien vers la mémoire CompactFlash.

Le tableau 6.2 montre les résultats globaux de ressources en utilisant la plateforme XUPV5. L'architecture proposée occupe 17% des slices-registres, 15% des LUTs, 17% des mémoires BRAM ainsi que 54% des DSP disponibles. La conception permet d'atteindre une fréquence d'horloge maximale de 250 MHz.

| | Slices | | LUTs | | BRAMs | | DSP48E | |
|---|---|---|---|---|---|---|---|---|
| | Utilisé | % | Utilisé | % | Utilisé | % | Utilisé | % |
| plateforme d'acquisition et de restitution vidéo | 6981 | 10,10% | 5790 | 8,38% | 21 | 14,19% | 3 | 4,68% |
| Le module $TQ/QT^{-1}$ | 5 392 | 7,80% | 4 949 | 7,16% | 5 | 3,38% | / | / |
| Total des ressources | 12 373 | 17,90% | 10 739 | 15,54% | 26 | 17,57% | 35 | 54,69% |

Tableau 6.2. Résultats de synthèse du système à base de processeur et $TQ/QT^{-1}$.

L'accélérateur matériel $TQ/QT^{-1}$ lit/écrit les données depuis/vers la mémoire à travers le bus PLB. L'utilisation du processeur pour effectuer le transfert entre le coprocesseur et la mémoire nécessite plusieurs cycles pour la lecture et l'écriture des données. Afin de diminuer le temps d'accès mémoire et augmenter les performances du système, il est possible d'effectuer le transfert de données en matériel en utilisant le contrôleur mémoire proposé dans le chapitre 4.

### 3.3. Implantation logicielle/matérielle de l'encodeur H.264/AVC

Les modules de l'encodeur H.264/AVC conçus dans le chapitre 5 sont synthétisés en utilisant ISE, des fichiers (.ngc) sont déjà générés. Nous utilisons ces fichiers avec la plateforme matérielle d'acquisition et de restitution vidéo pour calculer l'espace total occupé sur le circuit FPGA. Malgré la non

vérification du fonctionnement final de l'encodeur (à cause de l'absence des modules de traitement inverse), nous pouvons au moins vérifier le schéma en codesign. Le système sera composé par la plateforme d'acquisition et de restitution vidéo en plus de deux accélérateurs (le codage Intra et le filtre anti-blocs). Ces deux accélérateurs peuvent être intégrés indépendamment chacun avec une FIFO en entrée et une autre en sortie. Ils peuvent être intégrés aussi en cascade en utilisant les mêmes FIFOs.

Après le développement des différents blocs IPs et afin de bénéficier de la performance du matériel, nous développons pour chaque accélérateur matériel la partie logicielle correspondante, et cela dans le but d'effectuer les communications avec le reste du système. Les modules de traitement de l'encodeur H.264 peuvent recevoir des séquences vidéo à partir de la mémoire CompactFlash suivant une fréquence calculée selon le format de la séquence vidéo choisi. Ces séquences seront traitées par les modules, ensuite les séquences reconstruites seront affichées à l'écran en temps réel.

### 3.3.1. Vérification de l'architecture

Pour la vérification de l'architecture (après le transfert des Bitstreams vers le circuit FPGA), nous utilisons le débuggeur de l'outil EDK. Pour vérifier le contenu de la mémoire DDR2, par exemple, nous utilisons la commande 'xrmem' tout en indiquant l'adresse de cette mémoire ainsi que le nombre d'octets (pixels) à afficher. Nous vérifions, par conséquent, le transfert des images de la mémoire CompactFlash vers la DDR2.

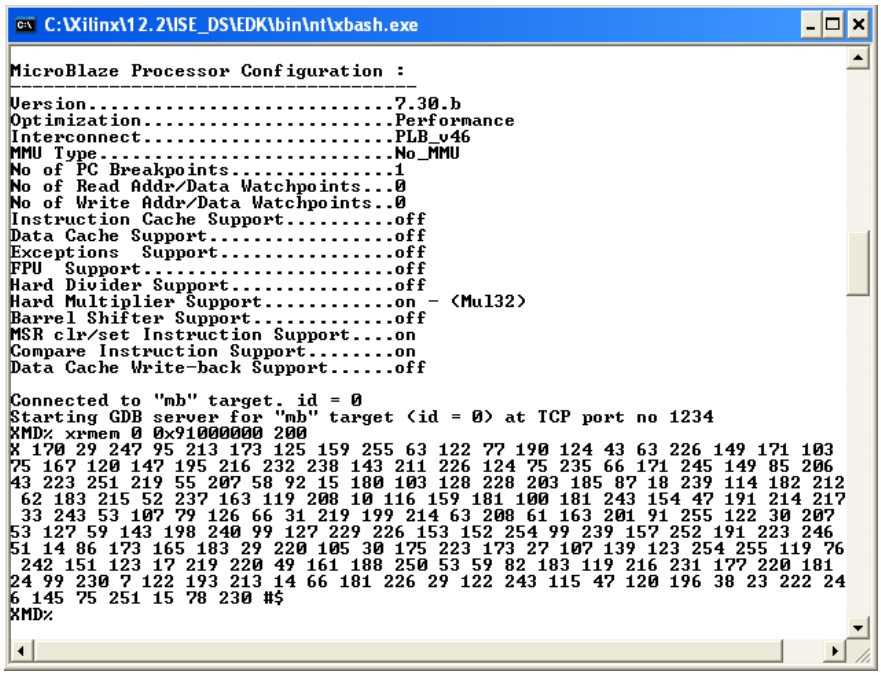



Figure 6.12. Vérification du contenue de la mémoire DDR2.

### 3.3.2. Résultats de synthèse

Le tableau 6.3 montre les résultats de synthèse de l'ensemble du système (plateforme d'acquisition et de restitution vidéo plus les modules de l'encodeur H.264/AVC) en utilisant la plateforme XUPV5. Le système réalisé consomme 36% des LUTs et 34% des slices-registres du circuit FPGA Virtex5-LX110T. En plus, 28% des mémoires BRAM et trois DSPs sont utilisés. La conception permet d'atteindre une fréquence d'horloge maximale de 155MHz. Nous remarquons que le module de codage Intra n'utilise pas des DSP48Es, malgré que dans le chapitre 5 ce même module, synthétisé

indépendamment, utilise 32 DSP (pour les modules de quantification directe et inverse). Nous pouvons forcer le synthétiseur à réaliser les multiplications dans le module Intra en utilisant les DSPs, ce qui signifie une diminution en plus dans les ressources utilisées. La figure 6.13 montre le taux d'utilisation des ressources matérielles pour l'implémentation des deux modules de l'encodeur H.264/AVC (la chaine Intra et le filtre anti-blocs) plus la plateforme d'acquisition et de restitution vidéo.

| | Slices | | LUTs | | BRAMs | | DSP48E | |
|---|---|---|---|---|---|---|---|---|
| | Utilisé | % | Utilisé | % | Utilisé | % | Utilisé | % |
| plateforme d'acquisition et de restitution vidéo | 6 616 | 9,57% | 5 707 | 9,25% | 21 | 14,19% | 3 | 4,68% |
| La chaine de prédiction Intra avec les deux FIFOs | 9 571 | 13,84% | 10 021 | 14,50% | 11 | 7,43% | / | / |
| Le filtre anti-blocs avec les deux FIFOs | 7 402 | 10,71% | 9364 | 13,54% | 10 | 6,76% | / | / |
| Total des ressources | 23 589 | 34,13 | 25092 | 36,30% | 42 | 28,38% | 3 | 4,68% |

Tableau 6.3. Résultats de synthèse du système.

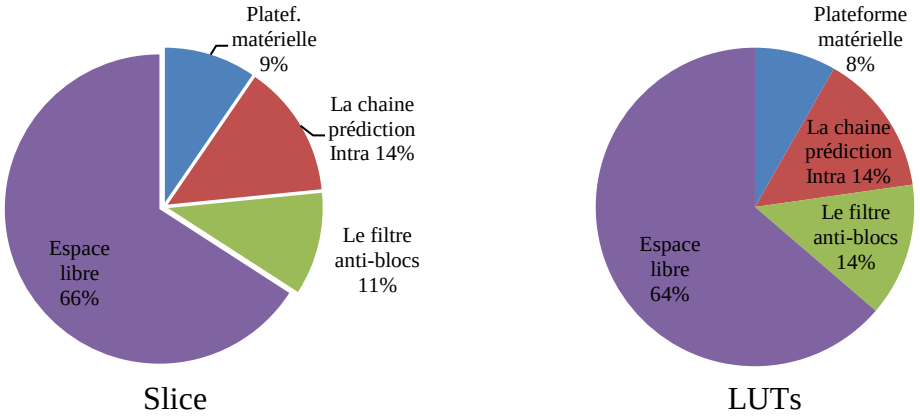
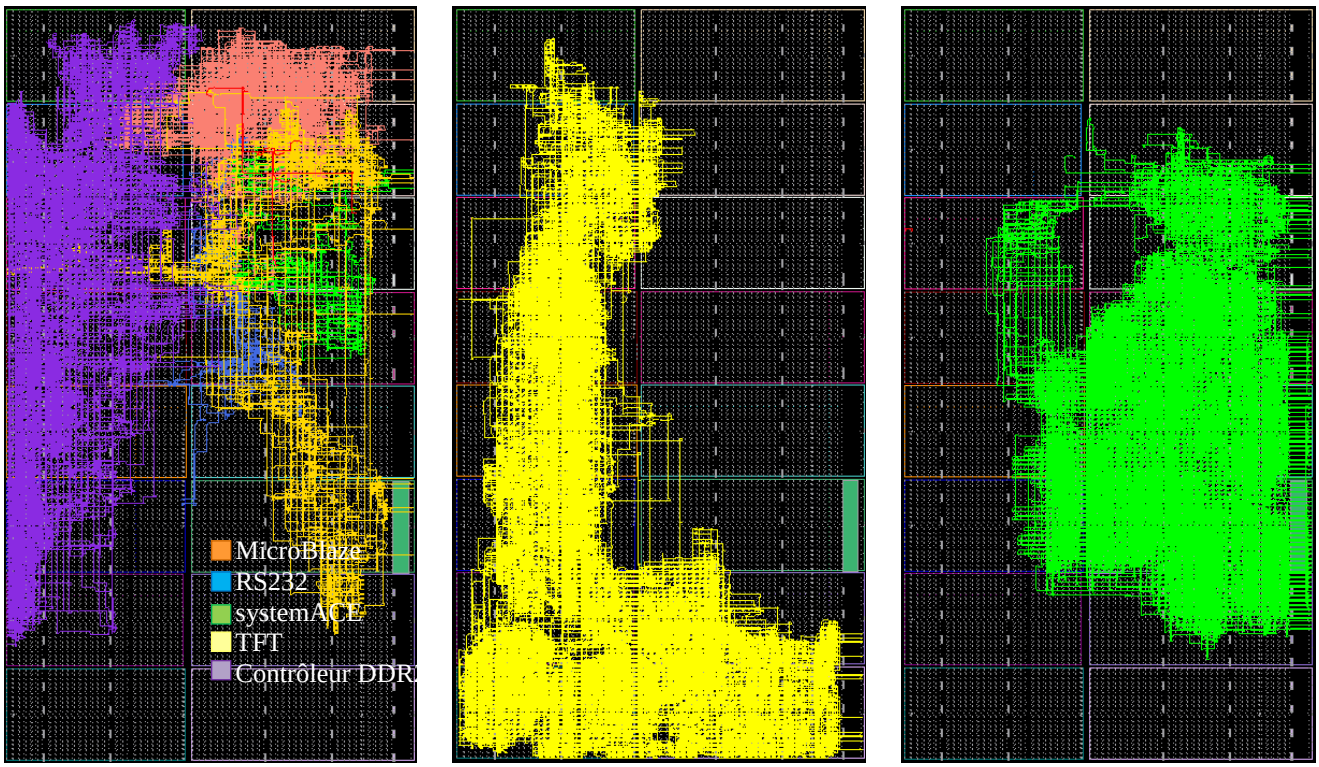



Figure 6.13. Taux d'utilisation des ressources matérielles de l'encodeur H.264/AVC.

### 3.3.3. Placement et routage

La figure 6.14 montre les résultats de placement et routage du système en utilisant la plateforme XUPV5. Nous utilisons l'outil 'FPGA Editor' de Xilinx pour montrer l'emplacement des ressources (routage des signaux) utilisées sur la surface du circuit FPGA (Virtex5-LX110T).




Plateforme matérielle | La chaine Intra | Le filtre anti-blocs

Figure 6.14. Placement et routage du système sur le circuit FPGA LX110T.

## 4. Conclusion

Dans ce chapitre, nous avons étudié et implanté l'encodeur H.264/AVC dans un environnement logiciel/matériel. L'encodeur dans ce cas est limité dans la chaine de codage Intra et le filtre anti-blocs, ces deux modules ont été ciblés à une implantation matérielle dans le chapitre 5. Dans une première étape, nous avons implanté une plateforme matérielle d'acquisition et de restitution vidéo à la base d'un processeur MicroBlaze. Ensuite, nous avons utilisé une couche PLB avec des mémoires FIFOs pour l'intégration des accélérateurs matériels.

L'interface du périphérique avec le processeur MicroBlaze intégré fournit un élément de flexibilité, qui ne peut être réalisé dans l'approche purement matérielle. La flexibilité du système permettra de configurer les différents modules. En effet, le processeur MicroBlaze assure l'acquisition des images à travers les différents ports et mémoires externes de la plateforme de prototypage, il assure aussi la réorganisation des pixels dans la mémoire DDR2 selon la stratégie souhaitée. Le processeur peut également lire les blocs et les macroblocs à partir de la mémoire DDR2 selon les besoins des modules de traitement.

Les premiers résultats de synthèse de la plateforme matérielle, en utilisant la carte XUPV5 de Xilinx, montrent que l'implantation des différents périphériques constituant le système laisse suffisamment de ressources dans le circuit FPGA pour ajouter les modules de l'encodeur H.264/AVC. L'ajout des deux modules de codage Intra et de filtrage au système constitue un véritable système matériel de traitement vidéo (une implantation partielle de l'encodeur H.264). Les résultats de synthèse du système complet montrent qu'il reste encore de l'espace libre sur le circuit FPGA (66% des slices-registres, 64% des LUTs et 72% des mémoires BRAM), ce qui donne la possibilité d'ajouter les autres modules de traitement de l'encodeur H.264.

# CONCLUSION GÉNÉRALE

## Conclusion

Les systèmes sur puce (SoC) et les systèmes multiprocesseurs sur puce (MPSoC) sont deux aspects très utilisés de nos jours, ils sont motivés par l'augmentation continue de la capacité d'intégration (à cause des progrès des technologies CMOS) d'une part, et la complexité croissante des applications embarquées d'autre part, surtout en ce qui concerne les applications du multimédia. En effet, La conception de systèmes embarqués est généralement motivée par l'évolution technologique et la généralisation du multimédia.

Aujourd'hui, le choix d'une architecture pour le traitement multimédia embarqué n'est pas trivial et qu'un compromis doit certainement être trouvé entre flexibilité, consommation, performance, coût et rapidité de conception en terme de TTM (Time-To-Market). L'objectif de cette thèse est la contribution au développement et à la conception d'un système multimédia embarqué (le codec H.264) en utilisant la méthodologie de codesign (conception logicielle/matérielle). Le but final est d'avoir une bibliothèque des modules IPs (pour les module de l'encodeur H.264/AVC) pour les utiliser, par la suite, dans des schémas divers selon la stratégie d'implantation fixée et selon l'application vidéo à implanter.

Après la présentation des différents langages et niveaux de description du matériel ainsi que les outils de développement, nous avons choisi le VHDL comme langage de programmation à cause de sa simplicité pour la description des processus parallèles et séquentiels, et nous avons choisi les outils de conception de Xilinx à cause de leurs disponibilités, leurs flots de conception simples et rapides mais aussi à cause de l'utilisation des plateformes multi-composants de la même compagnie (Xilinx). En effet, pour la validation de nos implémentations, nous avons utilisé les deux plateformes ML501 et XUPV5 disposant des circuits FPGA Virtec5-LX50 et Virtex5-LX110T respectivement.

Nous avons également étudié en détails la norme de compression vidéo H.264/AVC. Nous avons conclu que la complexité de l'encodeur ne réside pas uniquement dans la partie traitement et la partie commande, mais elle réside aussi dans le degré de dépendance inter-bloc dans les différents traitements et par conséquent dans la partie de la gestion de mémoire. Pour cela et dans une première contribution de cette thèse, nous avons proposé un contrôleur mémoire intelligent conçu spécialement pour l'encodeur H.264/AVC. Ce contrôleur est capable d'assurer les données d'entrée pour les différents modules de l'encodeur en fixant aussi la taille du bus de données selon la stratégie d'implantation. Nous avons ainsi exploité les caractéristiques des mémoires internes et externes des plateformes de prototypage pour fournir des données sur 128bits (bloc 4×4pixels qui représente la taille idéale pour le parallélisme en vue de la dépendance inter-bloc dans H.264).

L'étude de la norme H.264/AVC nous a conduit aussi à des perspectives sur les modules à implanter en matériel si nous cherchons une conception logicielle/matérielle de l'encodeur. La décision est basée sur la répartition du temps d'exécution des différents modules sur un processeur séquentiel. Le module d'estimation et de compensation de mouvement a pris la part du lion du temps de calcul mais aussi des

efforts des ingénieurs et chercheurs pour l'implantation de ce module. En effet, plusieurs techniques et implémentations ont été proposées pour l'estimation de mouvement, ceci même avant l'apparition de la norme H.264/AVC à cause de l'utilisation de ce module dans les normes de compression précédentes et dans plusieurs autres applications.

Parmi les nouveautés de la norme H.264/AVC est l'utilisation d'une nouvelle prédiction Intra appliquée dans le domaine spatial et non pas dans le domaine fréquentiel, aussi l'utilisation d'un filtre adaptative (en boucle) dans l'encodeur et le décodeur à la fois. Le temps d'exécution de ces deux modules est remarquable par rapport aux normes précédentes, il est estimé autour de 20% pour le module Intra et autour de 10% pour le filtre anti-blocs. Pour ces raisons, nous avons concentré nos efforts à la conception et l'implantation matérielle de ces deux modules. Le module Intra est constitué de toute la chaine de prédiction, de transformation, de quantification, de quantification inverse et de transformation inverse. En effet, vu la dépendance inter-blocs et inter-macroblocs dans l'encodeur H.264, nous avons remarqué qu'il impossible d'implanter le module de prédiction Intra sans l'implantation des autres modules de la chaine (T, Q, $Q^{-1}$ et $T^{-1}$).

Comme deuxième contribution de cette thèse, nous avons exploité les deux principes de parallélisme et de pipelining pour la conception des architectures matérielles pour les modules de la chaine Intra et le filtres anti-blocs. Les descriptions sont réalisées en VHDL, des simulations sont réalisées en utilisant ModelSim à travers des testbenchs conçus spécialement pour chaque module. Les résultats de synthèse en utilisant ISE de Xilinx montrent les avantages des architectures proposées. Nous avons donné enfin des statistiques graphiques des taux d'occupation de surface sur les deux circuits FPGA (LX50 et LX110T). Malgré que ces deux plateformes disposent d'assez de ressources pour réaliser les modules de la chaine Intra et le filtre anti-blocs, la deuxième plateforme est la plus adaptée pour l'implantation des autres modules de l'encodeur ainsi que les autres composants d'un système sur puce (processeur, périphériques, bus, etc.). En effet, en plus de l'espace occupée par nos implémentations, cette plateforme dispose d'un espace libre d'environ 77%. Une opération de raffinement des implémentations est toujours nécessaire dans le but de minimiser le nombre des ressources utilisées.

Une dernière étape dans la thèse consiste à l'implantation de l'encodeur H.264/AVC dans un environnement logiciel/matériel. Dans une première étape, nous avons implanté une plateforme matérielle d'acquisition et de restitution vidéo à la base d'un processeur MicroBlaze. Ensuite, nous avons utilisé une couche PLB avec des mémoires FIFOs pour l'intégration des accélérateurs matériels. Les résultats de synthèse du système proposé, en utilisant la carte XUPV5 de Xilinx, montrent que l'implantation des différents modules laisse suffisamment de ressources dans le circuit FPGA pour ajouter les autres modules de l'encodeur H.264/AVC.

Nous avons utilisé aussi les implémentations matérielles dans des applications de la reconfiguration dynamique, surtout avec des entrées/sorties qui changent d'une implémentation à une autre.

## Perspectives

Les perspectives de ce travail sont diverses et multiples. La première concerne le contrôleur mémoire H.264, pour le rendre ce contrôleur plus dynamique en ajoutant une couche haut-niveau pour la spécification et la modélisation des besoins, et de fixer les outils pour changer les lignes de codes

chargés de calcul des adresses en utilisant les informations fournies par la première couche. Pour cela nous suggérons l'utilisation du langage AOL et l'environnement MARTE pour la conception est la génération des contrôleurs mémoires selon l'application (selon le profil et la version de l'application) et selon le type de la mémoire utilisée.

Nous espérons aussi l'implantation des autres modules en utilisant les mêmes stratégies de parallélisme et de pipelining, dans le but de la conception d'une bibliothèque des IPs pour les codecs vidéo. Ensuite, et comme complément de ce travail, nous espérons que les différents modules implantés soient développés par des modèles en utilisant un standard de modélisation (IP-Xact par exemple). Notre but est également d'ajouter toutes les données nécessaires pour chaque module (Code VHDL, description, documentation, etc.), pour obtenir ainsi des modules prêts à être intégrés dans n'importe quel système en chips. Nos implémentations peuvent servir aussi comme un point de référence en cas d'une nouvelle version de la norme.

Un autre travail qui semble important consiste à la confirmation de l'efficacité de nos implémentations matérielles par l'utilisation des séquences d'images réelles. Dans ce cas une étude comparative, des performances et des qualités des images (PSNR), serait nécessaire toute en comparant les résultats obtenus avec les autres résultats publiés et les résultats donnés par le logiciel de référence de H.264/AVC. Une étude énergétique des différentes implémentations semble aussi intéressante.

## Bibliographies


[1] Gordon E. Moore, 'Cramming More Components onto Integrated Circuits', IEEE, Vol. 86 No. 1, pp. 82-85, Janvier 1998.

[2] Fabrice URBAN, 'Implantation optimisée d'estimateurs de mouvement pour la compression vidéo sur plates-formes hétérogènes multi-composants', Thèse, Institut National Des Sciences Appliquées De Rennes, Décembre 2007.

[3] Mickaël RAULET, 'Optimisations Mémoire dans la Méthodologie AAA pour Code Embarqué sur Architectures Parallèles', Thèse, Institut National Des Sciences Appliquées De Rennes, Mai 2006.

[4] Ahmed BEN ATITALLAH, 'Etude et Implantation d'Algorithmes de Compression d'Images dans un Environnement Mixte Matériel et Logiciel', Thèse, Université de Bordeaux1, Juillet 2007.

[5] Imed eddine BEGHOUL, Hamza BENNACER, 'Architectures matérielles pour les opérateurs utilisés dans les normes de compression d'image', Mémoire, Université Mentouri de Constantine, Juin 2009.

[6] Abdelhalim SAMAHI, 'Contribution à la mise en œuvre d'une plate-forme de prototypage rapide pour la conception des systèmes sur puce', Thèse, Laboratoire LE2I, Université de Bourgogne, 2007.

[7] Michel PAINDAVOINE, 'Traitement des images en temps réel - Applications industrielles', Techniques de l'Ingénieur, Mesures et Contrôle, R 6-720, 1-10.

[8] Fawzi GHOZZI, 'Optimisation d'une bibliothèque de modules matériels de traitement d'images. conception et test vhdl, implémentation sous forme FPGA', Thèse, Université de Bordeaux1, 2003.

[9] Nedjmi Djamel Eddine RACHEDI, 'Configuration d'un microprocesseur softcore « LEON2 »', Mémoire, Université Badji Mokhtar de Annaba, Juin 2007.

[10] Dragomir MILOJEVIC, 'Implémentation des Filtres non-linéaires de rang sur des architectures universelles et reconfigurables', Thèse, Université Libre de Bruxelles, 2004.

[11] Fabien L'EXCELLENT, 'Mise En Œuvre D'une Application De Traitement D'image Sur La Plate Forme Ml310', Mémoire d'ingénieur CNAM, Centre Régional Associe De Bourgogne, Centre D'enseignement De Dijon, Décembre 2006.

[12] Sami BOUKHCHAM, 'mise en place d'une plate-forme multiprocesseurs pour les systèmes embarqués', Thèse, Laboratoire LE2I, Université de Bourgogne, 2008.

[13] Synplicity Inc., 'Outil de synthèse logic « Synplify »', disponible sur le site : www.synplicity.com, 2007.

[14] Xilinx Inc., 'Outil de synthèse logique ISE-V9.2i'.

[15] Site Internet, http://www.opencores.com.

[16] Ali Erdem ÖZCAN, 'Conception et Implantation d'un Environnement de Développement de Logiciels à Base de Composants : Applications aux Systèmes Multiprocesseurs sur Puce', Thèse, Institut National polytechnique de Grenoble, Mars 2007.

[17] R. Wilson, 'Is SOC really different?', EE-Times, URL: http://www.eet.com/article/showArticle.jhtml?articleId=18303185, November 1999.

[18] Amer BAGHDADI, 'Exploration et conception systématique d'architectures multiprocesseurs monopuces dédiées à des applications spécifiques', Thèse, Institut National polytechnique de Grenoble, 2002.

[19] Khemaies GHALI, 'Méthodologie De Conception Système A Base De Plateformes Reconfigurables Et Programmables', Thèse, Université PARIS XI Orsay, Mars 2005.

[20] Xilinx Inc., 'Custom Peripheral Design Guide'.

[21] G. Micheli, S. Murali, 'An application-specific design methodology for stbus crossbar generation', Design, Automation and Test in Europe (DATE'05), pp. 1176-1181, 2005.

[22] W.J. Dally, B. Towles, 'Route packets, not wires: On-chip interconnection networks', Proc. Design and Automation Conference DAC 2001, pp. 684-689, Juin 2001.

[23] Riad BENMOUHOUB, ‘Méthodologies De Conception Pour Multiprocesseurs Sur Circuits Logiques Programmables’, Thèse, Université PARIS-SUD XI, Orsay, Ecole Nationale Supérieure De Techniques Avancées, Mai 2007.

[24] IBM Inc., ‘PowerPC405’, disponible sur le site : http://www.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_405 _Embedded_Cores/.

[25] N. Lodha, N. Rai, R. Dubey, H. Venkataraman, ‘Hardware-Software Co-design of QRD-RLS Algorithm with Microblaze Soft Core Processor’, Springer-Verlag Berlin Heidelberg, Vol. 31, pp. 197-207, 2009.

[26] P. Mu, M. Raulet, J.F. Nezan, J.G. Cousin, ‘Automatic code generation for multi-MicroBlaze system with syndex’, EURASIP Conférence, EUSIPCO2007, pp. 1644-1948, 2007.

[27] Xilinx Inc., ‘MicroBlaze Processor Reference Guide’, http://www.xilinx.com, 2004.

[28] Altera Inc., ‘Introduction to SOPC Builder’, Chapitre1, Quartus II Handbook Version 10.0, Volume 4.

[29] Antoine ROBERT, ‘Transformées orientées par blocs pour le codage vidéo hybride’, Thèse, Ecole Nationale Supérieure des Télécommunications, France, Février 2008.

[30] Site Internet, http://www.techno-science.net/?onglet=glossaire&definition=7376.

[31] Aurélie Martin, ‘Représentations parcimonieuses adaptées à la compression d’images’, Thèse, IRISA-Université De Rennes1, Avril 2010.

[32] Site Internet, http://www.planetenumerique.com/?Decouverte-du-MPEG-4-part-10-AVC.

[33] Mauricio Alvarez, Esther Salami, Alex Ramirez, Mateo Valero, ‘A Performance Characterization of High Definition Digital Video Decoding using H.264/AVC’, IEEE2005, pp. 24-33, 2005.

[34] Clément CHASTAGNOL, ‘Stéganographie en domaine vidéo compressé’, Rapport Final de Travail de Fin d'études, Ecole Centrale de Lyon, THALES Communications, 2009.

[35] Kamel Messaoudi, Salah Toumi, El-Bay Bourennane, ‘Material architecture proposition for the block matching method of motion estimate in H264 standard’, IEEE conference, The International Conference on Information & Communication Technologies: from Theory to Applications ICTTA’08, Syria, 2008.

[36] H. Loukil, A. B. Atitallah, F. Ghozzi, F., M. A. B. Ayed, N. Masmoudi, ‘A Pipelined FSBM Hardware Architecture for HTDV-H.26x’, Springer-International Journal of Electrical Systems Science and Engineering, pp.128-135, 2009.

[37] Zhibo Chen, Jianfeng Xu, Yun He, Junli Zheng, ‘Fast integer-pel and fractional-pel motion estimation for H.264/AVC’, Elsevier, JVCI Journal of Visual Communication & Image Representation, Verlag Berlin Heidelberg, pp. 264-290, 2006.

[38] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, A. Luthra, ‘Overview of the H.264/AVC Video Coding Standard’, IEEE Transactions On Circuits And Systems For Video Technology, Vol. 13, No. 7, pp. 560-576, 2003.

[39] Iain E. G. RICHARDSON, ‘H.264 and MPEG-4 Video Compression - Video Coding for Next-generation Multimedia’, The Robert Gordon University, Aberdeen, UK, Edition WILEY, 2003.

[40] H.261: Video Codec for Audiovisual Services at p x 64 kb/s, Recommandation H.261 à l’UIT-T, Mars 1993.

[41] H.263: Video Coding for Low Bit rate Communication, Première Recommandation H.263 à l’UIT-T, Mars 1996.

[42] Standard MPEG-1 : ISO/IEC 11172-2, Coding of Moving Pictures and AssociatedAudio for Digital Storage Media at up to About 1,5 Mb/s.

[43] Standard MPEG-2 : ISO/IEC 13818-2, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information.

[44] Standard MPEG-4 : ISO/IEC 14496-2, Information Technology – Coding of Audio-Visual Objects.

[45] Site Internet, http://www.compute-rs.com/fr/conseil-2037056.htm.

[46] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG. Draft of Version 4 of H.264/AVC (ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 part 10) Advanced Video Coding), 2003.

[47] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG. (ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 part 10) Advanced Video Coding- Reference Software Manual), 2009.

[48] K. Messaoudi, S. Toumi, E. Bourennane, 'Proposal and study for an architecture hardware/software for the implementation of the standard H264', CISA'08, in: American Institute of Physics (AIP), Vol. 1019 No. 1, pp. 536-540, 2008.

[49] Ralf Schäfer, Thomas Wiegand, Heiko Schwarz, 'H.264/AVC la norme qui monte', UER–RevueTechnique, pp. 1-10, 2003.

[50] Guillaume LAROCHE, 'Modules de codage par compétition et suppression de l'information de compétition pour le codage de séquences vidéo', Thèse, Ecole Nationale Supérieure des Télécommunications, France, 19 Mai 2009.

[51] Iain E. G. RICHARDSON, 'The H.264 Advanced Video Compression Standard', Livre, Edition WILEY, Second Edition, 2010.

[52] C. L. Yang, L. M. Po, W. H. Lam, 'A Fast H.264 Intra Prediction Algorithm Using Macroblock Properties', IEEE, pp. 461-464, 2004.

[53] K. Suh, S. Park, H. Cho, 'An Efficient Hardware Architecture of Intra Prediction and TQ/IQIT Module for H.264 Encoder', ETRI Journal, Vol. 27 No. 5, pp. 511-524, 2005.

[54] Z. Chen, W. Uao, U. Wang, M. Zhang, W. Zheng, 'A performance optimized architecture of deblocking filter for H .264/AVC', The Journal of China Universities of Posts and Telecommunications, Vol. 14, pp. 83-88, 2007.

[55] M. Horowitz, A. Joch, F. Kossentini, S. Member, S. Hallapuro, 'H.264/AVC Baseline Profile Decoder Complexity Analysis', IEEE transactions on circuits and systems for video technology, Vol. 13 No. 7, pp. 704-716, 2003.

[56] Paola Sunna, 'AVC/H.264 Un système de codage vidéo évolué pour la HD et SD', UER–Revue Technique, pp. 54-58, 2005.

[57] Site Internet, http://fr.wikipedia.org/wiki/H.264.

[58] J. M. GONZALEZ, 'Optimisation des performances d'un encodeur suivant la norme Advanced Video Coding pour une machine vectorielle', Licence en Informatique, Université Libre de Bruxelles, 2006.

[59] A. Elyousfi, A. Tamtaoui, E. Bouyakhf, 'Fast Mode Decision Algorithm for Intra prediction in H.264/AVC', International Journal of Computer Science and Network Security IJCSNS, Vol. 7 No. 1, pp. 356-364, 2007.

[60] F. Pan, X. Lin, S. Rahardja, K. P. Lim, Z. G. Li, D. Wu, S. Wu, 'Fast Mode Decision Algorithm for Intra-prediction in H.264/AVC Video Coding', IEEE Circuits and systems for video Tech. Vol. 15 No. 7, pp. 13-822, 2005.

[61] J. S. Park, H. J. Song, 'Selective Intra Prediction Mode Decision for H.264/AVC Encoders', International Journal of Applied Science, Engineering and Technology, pp. 214-218, 2006.

[62] I. Amer, W. Badawy, G. Jullien, 'Hardware prototyping for the h.264 4x4 transformation', IEEE, ICASSP, pp. V77-V.80, 2004.

[63] M. Parlak, I. Hamzaoglu, 'Low Power H.264 Deblocking Filter Hardware Implementations', IEEE Transactions on Consumer Electronics, Vol. 54 No. 2, pp. 808-816, Mai 2008.

[64] B. G. Kim, J. H. Kim, C. S. Cho, 'A Fast Intra Skip Detection Algorithm for H.264/AVC Video Encoding', ETRI Journal, Vol. 28 No. 6, pp. 721-731, 2006.

[65] S. Wang, S. Yang, H. Chen, C. Yang, J. Wu, 'A Multi-core Architecture Based Parallel Framework for H.264/AVC Deblocking Filters', Springer, Jou. Sign Process Syst, 2008.

[66] Yen-Kuang Chen, Eric Q. Li, Xiaosong Zhou, Steven Ge, 'Implementation of H.264 encoder and decoder on personal computers', Elsevier, Journal of Visual Communication and Image Representation (17), pp. 509-532, 2006.

[67] Seung-Hwan Kim, Yo-Sung Ho, 'Algorithm-level Optimization for Real-time H.264/AVC Encoder', IWSSIP & EC-SIPMCS, Slovenia, pp. 131-134, 2007.

[68] Dongming Zhang, Shouxun Lin, Yongdong Zhang, Lejun Yu, 'Complexity controllable DCT for real-time H.264 encoder', Elsevier, Journal of Visual Communication and Image Representation (18), pp. 59-67, 2007.

[69] Advanced Video Coding for Generic Audiovisual Services, ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC). ITU-T and ISO/IEC JTC 1, Version 8 (SVC extension), Juillet 2007.

[70] Call for Proposals on Multi-view Video Coding. ISO/IEC JTC1/SC29/WG11 N7327, 2005.

[71] KTA software coordination, http://iphome.hhi.de/suehring/tml/download/KTA/.

[72] Site Internet, http://uuu.enseirb.fr/~kadionik/embedded/linux_realtime/linux_realtime2.html

[73] Omar KERMIA, 'Ordonnancement temps réel multiprocesseur de tâches non-préemptives avec contraintes de précédence, de périodicité stricte et de latence', Thèse, Université Paris XI, Mars 2009.

[74] Patrice KADIONIK, 'SYSTEMES NUMERIQUES', Ecole nationale supérieure d'Electronique, informatique & radiocommunications, 2002.

[75] M. Albert, K. Cheng, 'Real-Time Systems: Scheduling, Analysis, and Verification', Edition Wiley, 2002.

[76] Ouassila LABBANI, 'Modélisation à haut niveau du contrôle dans des applications de traitement systématique à parallélisme massif", Thèse, Université des sciences et technologies de Lille, Novembre 2006.

[77] Mathieux DUBOIS, 'Modélisation Et Conception D'une Plate-Forme De Traitement Et Transmission De Signaux Vidéo Numériques', Thèse de maîtrise en sciences appliquées, Université de Montréal, Juillet 2004.

[78] M. Benmohammed, 'Langages de Spécification de Systèmes Numériques Intégrés', International Conference: Sciences of Electronic, Technologies of Information and Telecommunications SETIT, Tunisie, Mars 2005.

[79] Xilinx Inc., 'ML501 Getting Started Tutorial', UG228 (v1.0), Août 2006.

[80] Xilinx Inc., 'ML501 Evaluation Platform - User Guide', UG226 (v1.4), Août 2009.

[81] Xilinx Inc., 'LX110T MIG Design Creation Using ISETM10.1i SP3, MIG 2.3 and ChipScopeTMPro 10.1i', Septembre 2008.

[82] Xilinx Inc., 'Virtex-5 Family Overview', DS100 (v5.0), Février 2009.

[83] Micron Technology Inc., '512Mb: x4, x8, x16 DDR2 SDRAM Features', PDF: 09005aef82f1e6e22004 Micron Technology, 2004.

[84] Xilinx Inc., 'Virtex-5 FPGA - User Guide', UG190 (v5.2), Novenbre 2009.

[85] Micron Technology Inc., '128MB, 256MB, 512MB: (x64, SR) 200-Pin DDR2 SDRAM SODIMM Features' Document: 09005aef80eec96e, 2005.

[86] Xilinx Inc., 'Xilinx Memory Interface Generator (MIG) User Guide, DDR SDRAM, DDRII SRAM, DDR2 SDRAM, QDRII SRAM and RLDRAM II Interfaces', Document:UG086 (v2.0), 2007.

[87] K. Messaoudi, M. Touiza, E.B. Bourennane, S. Toumi, 'Hardware/Software Co-Design with MicroBlaze Soft-Core Processor for the Integer Transform Algorithm Used in the H.264 Encoder' International Review on Computers and Software (I.Re.Co.S), Vol. 5 No. 3, pp. 348-354, Mai 2010.

[88] K. Messaoudi, E.B. Bourennane, S. Toumi, E. kerkouche et O. Labbani, 'Memory Requirements and Simulation Platform for the Implementation of the H.264 Encoder Modules', IEEE International Conference on Image Processing Theory, Tools and Applications IPTA'10, pp. 133-137, Juillet 2010.

[89] C. H. Chang, M. H. Chang, and W. Hwang, ‘A flexible two-layer external memory management for H.264/AVC decoder’, IEEE-SoC conference, pp. 219-222, 2008.

[90] S. H. Ji, J. W. Park, and S. D. Kim, ‘Optimization of Memory Management for H.264/AVC Decoder’, IEEE-ICACT, pp. 65-68, 2006.

[91] H. Chung, A. Ortega, ‘Efficient memory management control for H.264’, IEEE, ICIP, pp. 35-50, 2004.

[92] K. Ouyang, Q. Ouyang, Z. Zhou, ‘Optimization and Implementation of H.264 Encoder on Symmetric Multi-processor Platform’, IEEE - Computer Science and Information Engineering, pp. 265-269, 2009.

[93] K. Babionitakis, G. Doumenis, G. Georgakarakos, G. Lentaris, K. Nakos, D. Reisis, I. Sifnaios, N. Vlassopoulos, ‘A real-time H.264/AVC VLSI encoder architecture’, Springer Real-Time Image Proc, pp. 43-59, 2008.

[94] C. W. Ku, C. C. Cheng, G. S. Yu, M. C. Tsai, T. S. Chang, ‘A High-Definition H.264/AVC Intra-Frame Codec IP for Digital Video and Still Camera Applications’, IEEE Transactions on Circuits and Systems for Video Tech., Vol. 16 No. 8, pp. 917-928, 2006.

[95] F. Pan, X. Lin, R. Susanto, K. P. Lim, Z. G. Li, G. N. Feng, ‘D. J. Wu, S. Wu, ‘Fast Mode Decision for Intra Prediction’, Doc. G013, Mars 2003.

[96] J. S. Park, H. J. Song, ‘Selective Intra Prediction Mode Decision for H.264/AVC Encoders’, International Journal of Applied Science, Engineering and Technology, pp. 214-218, 2006.

[97] B. G. Kim, J. H. Kim, C. S. Cho, ‘A Fast Intra Skip Detection Algorithm for H.264/AVC Video Encoding’, ETRI Journal, Vol. 28 No. 6, pp. 721-731, 2006.

[98] F. Pan, X. Lin, R. Susanto, K. P. Lim, Z. G. Li, G. N. Feng, D. J. Wu, S. Wu, ‘Fast Mode Decision for HW.264’, Doc. G033, Décembre 2003.

[99] C. L. Yang, L. M. Po, W. H. Lam, ‘A Fast H.264 Intra Prediction Algorithm Using Macroblock Properties’, IEEE, pp. 461-464, 2004.

[100] B. Meng, O. C. Au, ‘Fast Intra-Prediction Mode Selection for 4×4 Blocks in H.264’, IEEE - ICASSP, pp. 389-392, 2003.

[101] F. Pan, X. Lin, S. Rahardja, K. P. Lim, Z. G. Li, D. Wu, S. Wu, ‘Fast Mode Decision Algorithm for Intra-prediction in H.264/AVC Video Coding’, IEEE Circuits and systems for video Tech., Vol. 15 No. 7, pp. 813-822, 2005.

[102] C. Hsu, M. Ho, J. Hong, ‘An Efficient Algorithm for Intra-prediction in H.264’, IEEE, pp. 35-36, 2006.

[103] A. Elyousfi, A. Tamtaoui, E. Bouyakhf, ‘Fast Mode Decision Algorithm for Intra prediction in H.264/AVC’, International Journal of Computer Science and Network Security IJCSNS, Vol. 7 No. 1, pp. 356-364, 2007.

[104] Y. W. Huang, B. Y. Hsieh, T. C. Chen, L. G. Chen, ‘Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder’, IEEE Transactions on Circuits and Systems for Video Tech, Vol. 15 No. 3, pp. 378-401, 2005.

[105] K. Suh, S. Park, H. Cho, ‘An Efficient Hardware Architecture of Intra Prediction and TQ/IQIT Module for H.264 Encoder’, ETRI Journal, Vol. 27 No. 5, pp. 511-524, 2005.

[106] H. M. Wang, C. H. Tseng, J. F. Yang, ‘Improved and fast algorithm for intra 4×4 mode decision in H.264/AVC’, IEEE conférence, Int. Symp. Circuits Syst., pp. 2128–2131, 2005.

[107] Site Internet : http://hardh264.sourceforge.net/H264-encoder-manual.html.

[108] G. Khurana, A. A. Kassim, T. P. Chua e M. B. Mi, ‘A pipelined hardware implementation of In-loop Deblocking Filter in H.264/AVC’, IEEE Transactions on Consumer Electronics, Vol. 52 No. 2, pp. 536-540, 2006.

[109] C. C. Cheng, T. S. Chang, K. B. Lee, ‘An In-Place Architecture for the Deblocking Filter in H.264/AVC’, IEEE Transactions on Circuits and Systems, Vol. 53 No. 7, pp. 530-534, 2006.

[110] L. Li, S. Goto, T. Ikenaga, ‘A highly parallel architecture for deblocking filter in H.264/AVC’, IEICE Transactions on Information and Systems, E88(7), pp. 1623-1628, 2005.

[111] Z. Chen, W. Uao, U. Wang, M. Zhang, W. Zheng, ‘A performance optimized architecture of deblocking filter for H .264/AVC’, The Journal of China Universities of Posts and Telecommunications, Vol. 14, pp. 83-88, 2007.

[112] Kamel Messaoudi, Gilberto Ochoa, El-Bay Bourennane, Salah Toumi, ‘Dynamic Partial Reconfiguration Exploitation for Reducing the Resource Utilization of the Deblocking Filter in H.264 CODECS Implementations’, The Second International Conference on Systems and Information Processing (ICSIP’11), Guelma, Mai 2011.

[113] Obianuju Ndili, Tokunbo Ogunfunmi, ‘FPSoC-Based Architecture for a FastMotion Estimation Algorithm in H.264/AVC’, EURASIP Journal on Embedded Systems, 2009.

[114] Yu-Jen Wang, Chao-Chung Cheng, Tian-Sheuan Chang, ‘A Fast Fractional Pel Motion Estimation Algorithm for H.264/MPEG-4 AVC’, IEEE- ISCAS2006, pp. 3974-3977, 2006.

[115] Marcelo Porto, Luciano Agostini, Leandro Rosa, ‘High Throughput Hardware Architecture for Motion Estimation with 4:1 Pel Subsampling Targeting Digital Television Applications’, Springer-Verlag Berlin Heidelberg, pp. 36-47, 2007.

[116] Zhibo Chen, Jianfeng Xu, Yun He, Junli Zheng, ‘Fast integer-pel and fractional-pel motion estimation for H.264/AVC’, Elsevier - JVCI Journal of Visual Communication & Image Representation Verlag Berlin Heidelberg, pp. 264-290, 2006.

[117] Yu-Kun Lin, Chia-Chun Lin, Tzu-Yun Kuo, Tian-Sheuan Chang, ‘A Hardware-Efficient H.264/AVC Motion-Estimation Design for High-Definition Video’, IEEE Transactions On Circuits And Systems, Vol. 55, No. 6, pp. 1526-1535, Juillet 2008.

[118] Xilinx Inc., ‘XPS Thin Film Transistor (TFT) Controller (v2.01a)’, DS695, Mai 2010.

[119] http://www.xilinx.com/univ/xupv5-lx110t-bsb.htm.

[120] Xilinx Inc., ‘XUPV5-LX110T Standard IP Design Adding PCores’, Sep 2008.

# *Liste des publications*

## 1. Revues

1. Kamel Messaoudi, Salah Toumi, El-Bay Bourennane, 'Proposition et Implémentation d'Architectures Matérielles pour les Opérateurs Utilisés dans les Normes de Compression d'Images', Revue SYNTHESE des Sciences et de la Technologie de l'université d'Annaba, N° 25, pp. 24-31, Novembre 2012.
2. Touiza Maamar, Gilberto Ochoa Ruiz, Bourennane El-Bay, Guessoum Abderrezak, Messaoudi Kamel, 'A Novel Methodology for Accelerating Bitstream Relocation in Partially Reconfigurable Systems', Elsevier, Microprocessors and Microsystems, Impact Factor (0,6), accepté pour publication, Juillet 2012.
3. K. Messaoudi, E. B. Bourennane, S. Toumi, "Reconfigurable Hardware Intelligent Memory Controller for H.264/AVC Encoders", International Journal of Computer Science and Information Security (IJCSIS), Vol. 8 No. 9, pp. 8-16, December 2010.
4. K. Messaoudi, M. Touiza, E.B. Bourennane, S. Toumi, 'Hardware/Software Co-Design with MicroBlaze Soft-Core Processor for the Integer Transform Algorithm Used in the H.264 Encoder' International Review on Computers and Software (I.Re.Co.S), Vol. 5 No. 3, pp. 348-354, Mai 2010.

## 2. Conférences internationales

1. K. Messaoudi, E. Bourennane, S. Toumi, M. Touiza, A. Yahi, 'Etude et Implantation des Systèmes de Communication Dans les Systèmes sur Puce (SoCs et SoPCs)', ICESTI'12, International Conference on Embedded Systems in Telecommunications and Instrumentation, November 5–7, 2012, Annaba, Algeria.
2. A. Yahi, K. Messaoudi, S. Toumi, E. Bourennane, 'Hardware Architecture Design Study of the IME-FSBMA Used in H.264/AVC Encoders', ICESTI'12, International Conference on Embedded Systems in Telecommunications and Instrumentation, November 5–7, 2012, Annaba, Algeria.
3. Mayache Hichem, Benhaoues Atef, Messaoudi Kamel, Bourennane El Bey, Toumi Salah, 'Designing Mesh 2x2 network on chip adapted for Wishbone protocol', ICESTI'12, International Conference on Embedded Systems in Telecommunications and Instrumentation, November 5–7, 2012, Annaba, Algeria
4. G. Ochoa-Ruiz, M. Touiza, E. Bourennane, A. Guessoum, K. Messaoudi, 'Accelerating partial bistream relocation in highly-scalable DPR applications', ICESTI'12, International Conference on Embedded Systems in Telecommunications and Instrumentation, November 5–7, 2012, Annaba, Algeria.
5. Messaoudi Kamel, Bourennane El-Bay, Toumi Salah, Touiza Maamar, 'Accélération et Optimisation de l'Implémentation Matérielle de l'Encodeur H.264/AVC par le Parallélisme des Données', XXIIIe Colloque GRETSI'11, Bordeaux – 5-8 Septembre 2011.
6. Kamel Messaoudi, El-Bay Bourennane, Salah Toumi and Gilberto Ochoa, 'Performance Comparison of Two Hardware Implementations of the Deblocking Filter Used in H.264 by Changing the Utilized Data Width', IEEE conference, The 7th International Workshop on Systems, Signal Processing and their Applications (WoSSPA2011), Tipaza-Algeria, May 2011.
7. Gilberto Ochoa, El-Bay Bourennane, Ouassila Labbani , Kamel Messaoudi, 'model-driven approach for automatic IP integration dynamically reconfigurable designs', The 2011 Forum on specification & Design Languages (FDL'11), September 13-15, 2011, Oldenburg Germany.
8. Kamel Messaoudi, Gilberto Ochoa, El-Bay Bourennane, Salah Toumi, 'Dynamic Partial Reconfiguration Exploitation for Reducing the Resource Utilization of the Deblocking Filter in H.264 CODECS Implementations', The Second International Conference on Systems and Information Processing (ICSIP'11), Guelma, Mai 2011.
9. K. Messaoudi, E.B. Bourennane, S. Toumi, E. kerkouche et O. Labbani, 'Memory Requirements and Simulation Platform for the Implementation of the H.264 Encoder Modules', IEEE International Conference on Image Processing Theory, Tools and Applications IPTA'10, Paris, pp. 133-137, Juillet 2010.

10. K. Messaoudi, E. Bourennane, S. Toumi, M. Touiza, O. Labbani, 'A Real-Time Simulation Platform for the H.264 CODEC Modules', ICMOSS'2010 - Premier Congrès International sur les modèles, Optimisation et Sécurité des Systèmes, Tiaret, Algérie, pp. 167-171, 29 au 31 Mai 2010.
11. M. Touiza, K. Messaoudi, E. Bourennane, A. Guessoum, 'Implementation of a Real-time Signal Processing Application using Partial Dynamic Reconfiguration in FPGA', ICMOSS'2010 - Premier Congrès International sur les modèles, Optimisation et Sécurité des Systèmes, Tiaret, Algérie, pp.155-160, 28 au 30 Mai 2010.
12. K. Messaoudi, E. Bourennane, S. Toumi, E. Kerkouche, O. Labbani, 'Memory Requirements for Hardware Implementation of the H.264 Encoder Modules', MISC'2010 - International Symposium on Modelling and Implementation of Complex systems, Constantine – Algeria, 30-31 Mai 2010.
14. Kamel Messaoudi, Salah Toumi, El-Bay Bourennane, 'Material architecture proposition for the block matching method of motion estimate in H264 standard', IEEE conference, The International Conference on Information & Communication Technologies: from Theory to Applications ICTTA'08, Syria, 2008.
14. K. Messaoudi, S. Toumi, E. Bourennane, 'Proposal and study for an architecture hardware/software for the implementation of the standard H264', CISA'08, in: American Institute of Physics (AIP), Vol. 1019 No. 1, pp. 536-540, 2008.

## 3. Conférences nationales

1. K. Messaoudi, A. Yahi, S. Toumi, E. Bourennane, 'Hardware Implementation Study of Real-Time Motion Estimation Algorithm Used in the H.264/AVC Encoder', CNT'2012, Première Conférence Nationale sur les Télécommunications, 11-12 Novembre 2012, Guelma, Algérie.
2. K. Messaoudi, M. Touiza, E. Bourennane, S. Toumi, A. Guessoum, 'Hardware/Software Co-Design for the Integer Transform Algorithm Used in the H.264 Encoder', IVième Colloque National, GDR SoC-SiP : System-on-Chip & System-in-Package, Cergy (Paris) – France, 9-11 Juin 2010.
3. K. Messaoudi, E. Bourennane, S. Toumi, 'Architecture matérielle basée sur le parallélisme des données pour le module intra-coding de l'encodeur H.26', XVIème Forum des Jeunes Chercheurs, Besançon-France, 07-06-2010.
4. K. Messaoudi, S. Toumi, E. Bourennane, 'Nouvelles architectures matérielles pour la lecture des images statiques et dynamiques en temps réel', Journées sur les Signaux et Systèmes JSS'07-Guelma, 15 nov. 2007.
5. K. Messaoudi, S. Toumi, E. Bourennane, 'Proposition et étude d'une architecture parallèle de localisation des blocs d'images dans la norme H264 », JSTRE - Journée sur les systèmes temps réel embarqués, 15 Mars 2008, Ecole militaire polytechnique, Algérie.

# Annexe A1

## Tableau récapitulatif des ressources FPGA de la famille Virtex5 de Xilinx

| Device | Configurable Logic Blocks (CLBs) | | | DSP48E Slices[2] | Block RAM Blocks | | | CMTs[4] | PowerPC Processor Blocks | Endpoint Blocks for PCI Express | Ethernet MACs[5] | Max RocketIO Transceivers[6] | | Total I/O Banks[8] | Max User I/O[7] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Array (Row x Col) | Virtex-5 Slices[1] | Max Distributed RAM (Kb) | | 18 Kb[3] | 36 Kb | Max (Kb) | | | | | GTP | GTX | | |
| XC5VLX30 | 80 x 30 | 4,800 | 320 | 32 | 64 | 32 | 1,152 | 2 | N/A | N/A | N/A | N/A | N/A | 13 | 400 |
| XC5VLX50 | 120 x 30 | 7,200 | 480 | 48 | 96 | 48 | 1,728 | 6 | N/A | N/A | N/A | N/A | N/A | 17 | 560 |
| XC5VLX85 | 120 x 54 | 12,960 | 840 | 48 | 192 | 96 | 3,456 | 6 | N/A | N/A | N/A | N/A | N/A | 17 | 560 |
| XC5VLX110 | 160 x 54 | 17,280 | 1,120 | 64 | 256 | 128 | 4,608 | 6 | N/A | N/A | N/A | N/A | N/A | 23 | 800 |
| XC5VLX155 | 160 x 76 | 24,320 | 1,640 | 128 | 384 | 192 | 6,912 | 6 | N/A | N/A | N/A | N/A | N/A | 23 | 800 |
| XC5VLX220 | 160 x 108 | 34,560 | 2,280 | 128 | 384 | 192 | 6,912 | 6 | N/A | N/A | N/A | N/A | N/A | 23 | 800 |
| XC5VLX330 | 240 x 108 | 51,840 | 3,420 | 192 | 576 | 288 | 10,368 | 6 | N/A | N/A | N/A | N/A | N/A | 33 | 1,200 |
| XC5VLX20T | 60 x 26 | 3,120 | 210 | 24 | 52 | 26 | 936 | 1 | N/A | 1 | 2 | 4 | N/A | 7 | 172 |
| XC5VLX30T | 80 x 30 | 4,800 | 320 | 32 | 72 | 36 | 1,296 | 2 | N/A | 1 | 4 | 8 | N/A | 12 | 360 |
| XC5VLX50T | 120 x 30 | 7,200 | 480 | 48 | 120 | 60 | 2,160 | 6 | N/A | 1 | 4 | 12 | N/A | 15 | 480 |
| XC5VLX85T | 120 x 54 | 12,960 | 840 | 48 | 216 | 108 | 3,888 | 6 | N/A | 1 | 4 | 12 | N/A | 15 | 480 |
| XC5VLX110T | 160 x 54 | 17,280 | 1,120 | 64 | 296 | 148 | 5,328 | 6 | N/A | 1 | 4 | 16 | N/A | 20 | 680 |
| XC5VLX155T | 160 x 76 | 24,320 | 1,640 | 128 | 424 | 212 | 7,632 | 6 | N/A | 1 | 4 | 16 | N/A | 20 | 680 |
| XC5VLX220T | 160 x 108 | 34,560 | 2,280 | 128 | 424 | 212 | 7,632 | 6 | N/A | 1 | 4 | 16 | N/A | 20 | 680 |
| XC5VLX330T | 240 x 108 | 51,840 | 3,420 | 192 | 648 | 324 | 11,664 | 6 | N/A | 1 | 4 | 24 | N/A | 27 | 960 |
| XC5VSX35T | 80 x 34 | 5,440 | 520 | 192 | 168 | 84 | 3,024 | 2 | N/A | 1 | 4 | 8 | N/A | 12 | 360 |
| XC5VSX50T | 120 x 34 | 8,160 | 780 | 288 | 264 | 132 | 4,752 | 6 | N/A | 1 | 4 | 12 | N/A | 15 | 480 |
| XC5VSX95T | 160 x 46 | 14,720 | 1,520 | 640 | 488 | 244 | 8,784 | 6 | N/A | 1 | 4 | 16 | N/A | 19 | 640 |
| XC5VSX240T | 240 x 78 | 37,440 | 4,200 | 1,056 | 1,032 | 516 | 18,576 | 6 | N/A | 1 | 4 | 24 | N/A | 27 | 960 |
| XC5VTX150T | 200 x 58 | 23,200 | 1,500 | 80 | 456 | 228 | 8,208 | 6 | N/A | 1 | 4 | N/A | 40 | 20 | 680 |
| XC5VTX240T | 240 x 78 | 37,440 | 2,400 | 96 | 648 | 324 | 11,664 | 6 | N/A | 1 | 4 | N/A | 48 | 20 | 680 |
| XC5VFX30T | 80 x 38 | 5,120 | 380 | 64 | 136 | 68 | 2,448 | 2 | 1 | 1 | 4 | N/A | 8 | 12 | 360 |
| XC5VFX70T | 160 x 38 | 11,200 | 820 | 128 | 296 | 148 | 5,328 | 6 | 1 | 3 | 4 | N/A | 16 | 19 | 640 |
| XC5VFX100T | 160 x 56 | 16,000 | 1,240 | 256 | 456 | 228 | 8,208 | 6 | 2 | 3 | 4 | N/A | 16 | 20 | 680 |
| XC5VFX130T | 200 x 56 | 20,480 | 1,580 | 320 | 596 | 298 | 10,728 | 6 | 2 | 3 | 6 | N/A | 20 | 24 | 840 |
| XC5VFX200T | 240 x 68 | 30,720 | 2,280 | 384 | 912 | 456 | 16,416 | 6 | 2 | 4 | 8 | N/A | 24 | 27 | 960 |

## Annexe A2

## Constitution des mémoires RAM et ROM à l'aide des LUTs dans la famille Virtex5 de Xilinx

Le tableau suivant montre le type et le nombre des mémoires RAM distribuées et des mémoires ROM qui peuvent être générer sur la Viretex5-LX50 (sur les slice de type SLICEM), ainsi que le nombre des LUTs utilisées.

| RAM | Number of LUTs |
|---|---|
| Single-Port 32 x 1-bit RAM **(ram32x1S)** | 1 |
| Dual-Port 32 x 1-bit RAM **(ram32x1D)** | 2 |
| Quad-Port 32 x 2-bit RAM **(ram32x2Q)** | 4 |
| Simple Dual-Port 32 x 6-bit RAM **(ram32x6SDP)** | 4 |
| Single-Port 64 x 1-bit RAM **(ram64x1S)** | 1 |
| Dual-Port 64 x 1-bit RAM **(ram64x1D)** | 2 |
| Quad-Port 64 x 1-bit RAM **(ram64x1Q)** | 4 |
| Simple Dual-Port 64 x 3-bit RAM **(ram64x3SDP)** | 4 |
| Single-Port 128 x 1-bit RAM **(ram128x1S)** | 2 |
| Dual-Port 128 x 1-bit RAM **(ram128x1D)** | 4 |
| Single-Port 256 x 1-bit RAM **(ram256x1S)** | 4 |

Les mémoires RAM distribuées dans la Virtex5-LX50.

Le tableau suivant montre le type et le nombre des mémoires ROM qui peuvent être générer sur la Viretex5-LX50 ainsi que le nombre des LUTs utilisées. En plus, les slices de type SLICEM uniquement peuvent être configuré en tant qu'un registre de décalage à 32 bits (SRLC32E). Deux ou bien plusieurs registres SR32 peuvent être montés en cascade pour avoir des registres à décalage à 64bit, 96bits, 128bits, etc.

| ROM | Number of LUTs |
|---|---|
| **ROM64x1** | 1 |
| **ROM128x1** | 2 |
| **ROM256x1** | 4 |

Mémoires ROM dans la Virtex5-LX50.

## Annexe B1

### Données techniques pour les implémentations matérielles du filtre anti-blocs

Pour l'implantation matérielle du filtre, et en utilisant les recommandations de la norme H.264/AVC [47], nous supposons que :

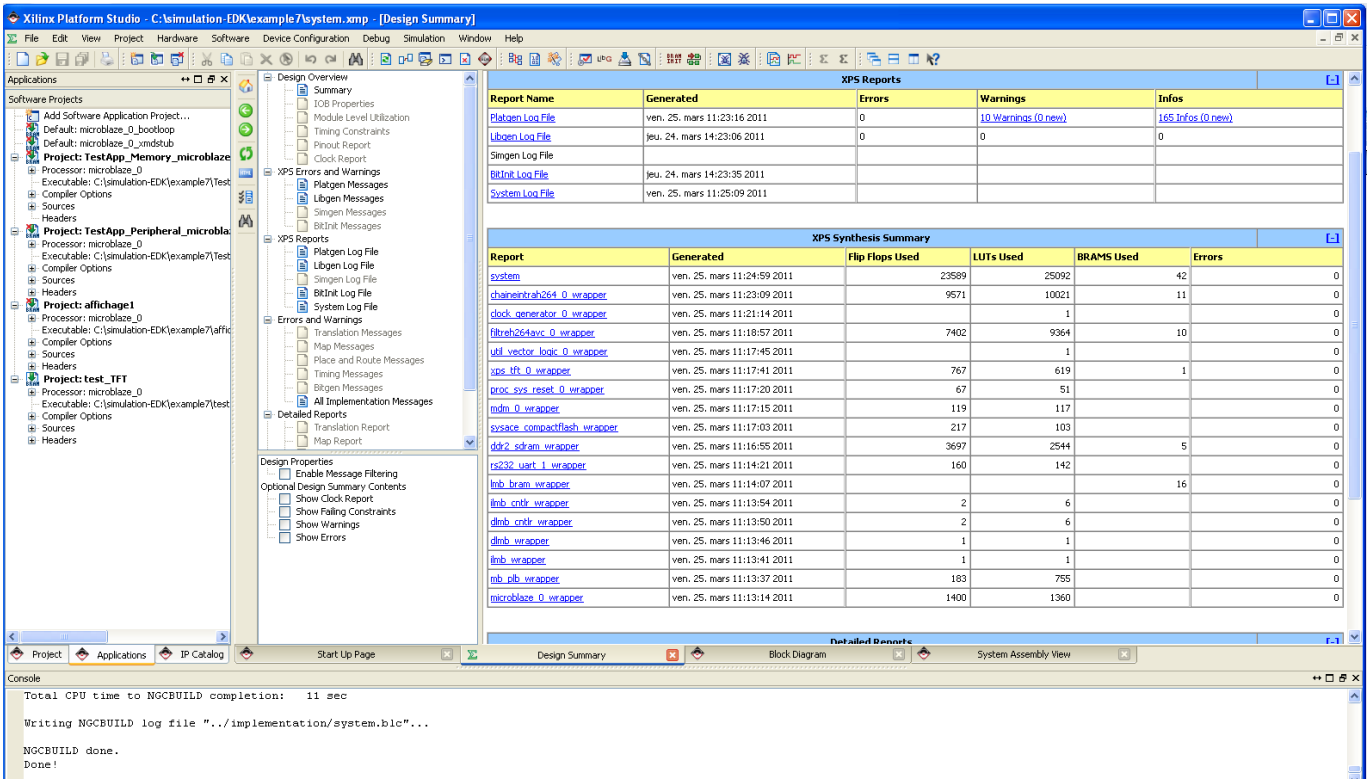
1. Les macroblocs contenant les échantillons à filtrer ne sont pas de type I_PCM, ce qui implique que $qPp = qPq = QP$ pour le signal de luminance et le signal de chrominance. Dans notre cas le facteur de quantification a une valeur unique pour tous les macroblocs. De même pour la valeur $qPav = (qPp + qPq + 1) \gg 1 = QP$.
2. La valeur de slice_alpha_c0_offset_div2 n'est pas présente dans l'en-tête de tranche pour le moment, cette valeur doit être supposée égale à 0. Ce qui implique que : FilterOffsetA = slice_alpha_c0_offset_div2 << 1 = 0. Même remarque pour la valeur FilterOffsetB = slice_beta_offset_div2 << 1 = 0.
3. Tant que la valeur de qPav est toujours positive et FilterOffsetA = FilterOffsetB = 0, les deux valeurs Index_A et Index_B seront calculées par :

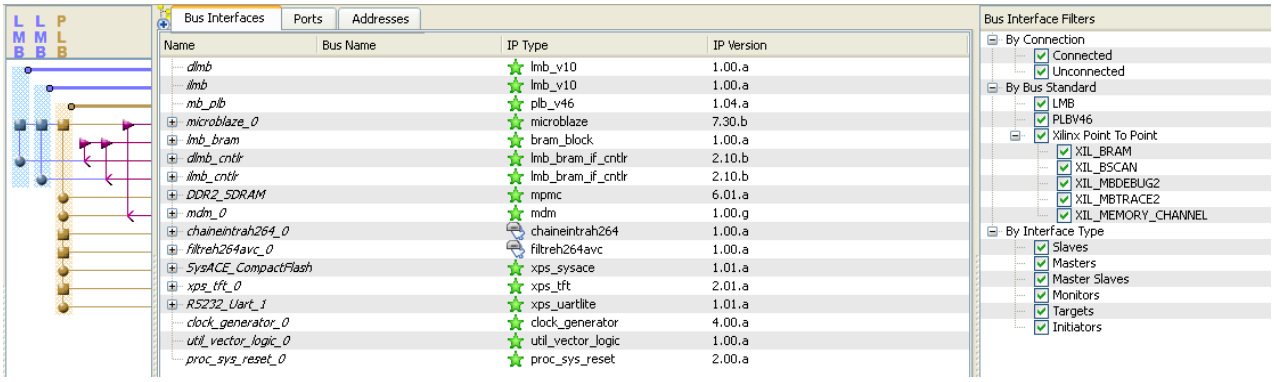
   $Index\_A = Clips3(0, 51, qPav + FilterOffsetA) = Min(51, qPav)$

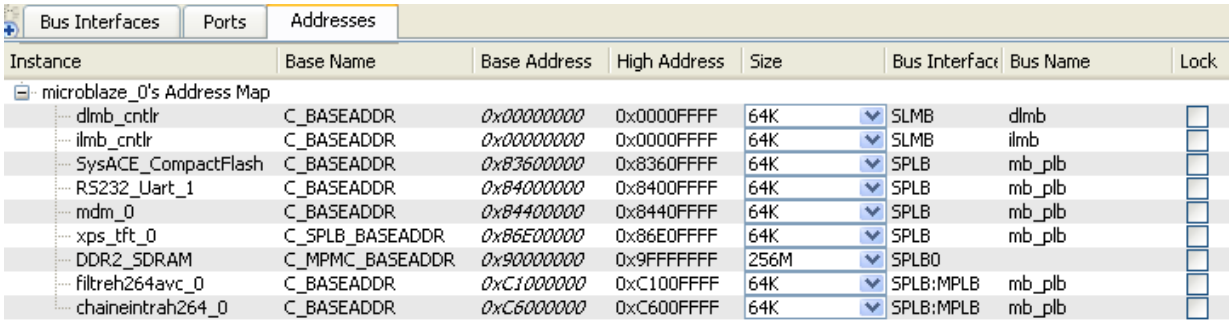
   $Index\_B = Clips3(0, 51, qPav + FilterOffsetB) = Min(51, qPav)$
4. Les variables α', β' et t'c0 sont spécifiées dans les deux tableaux, présentés dans [47], en fonction des valeurs de Index_A et de Index_B. Pour que notre filtre soit le plus dynamique possible, nous utilisons des mémoires ROM pour l'enregistrement des différents tableaux.
5. bit_depth_luma_minus8 est supposée non disponible, sa valeur doit être supposée égale à 0. Ce qui implique que : BitDepthY = 8 + bit_depth_luma_minus8 = 8. Même remarque pour BitDepthC = 8 + bit_depth_chroma_minus8 = 8.
6. Les variables de seuil α, β et tc sont calculées comme suit :
   α = α' * (1 << ( BitDepthY – 8 ) ) = α' ;
   β = β' * (1 << ( BitDepthY – 8 ) ) = β' ;
   tc0 = t'c0 * (1 << ( BitDepthY – 8 ) ) = t'c0 ;
   Ceci est valable pour le signal de luminance et le signal de chrominance.
7. Les équations dans l'organigramme de la figure 2.22 seront simplifiées car la valeur de tc0 est toujours positive. Exemple pour $\Delta_0 = \text{Clip3}(-\text{tc0}, \text{tc0}, \Delta_{01}) = \text{Min}(\text{MAX}(-\text{tc0}, \Delta_{01}), \text{tc0}) = \text{Min}(\Delta_{01}, \text{tc0})$.
8. Les valeurs α et β sont utilisées dans les conditions d'implantation des filtres élémentaires, et la valeur tc0 est utilisée pour les calculs des pixels filtrés en fonction des conditions précédentes. Ces valeurs sont calculées dans le module principal de l'implémentation, et utilisées dans les différents filtres et pour les différents blocs dans le même macrobloc. Ces valeurs sont calculées, en parallèle avec la lecture des blocs et l'initialisation des filtres, en 3 cycles d'horloge.
   Les valeurs de α et β dépendent uniquement de la valeur QP, ce qui implique que ces deux valeurs sont fixes pour tous les blocs et les macroblocs. Par contre, tc0 dépend de QP et BS, avec BS qui change d'un bloc à l'autre, ce qui implique que tc0 est calculé pour chaque bloc. Pour éviter l'accès multiple à la mémoire ROM réservée pour tc0, nous proposons l'utilisation de trois variables calculées en fonction des valeurs de α et β, et de faire le choix entre ces trois valeurs en fonction de la valeur de BS. Il faut aussi définir une stratégie de transfert des valeurs de tc0 entre les différents filtres directionnels, comme dans le cas de BS.

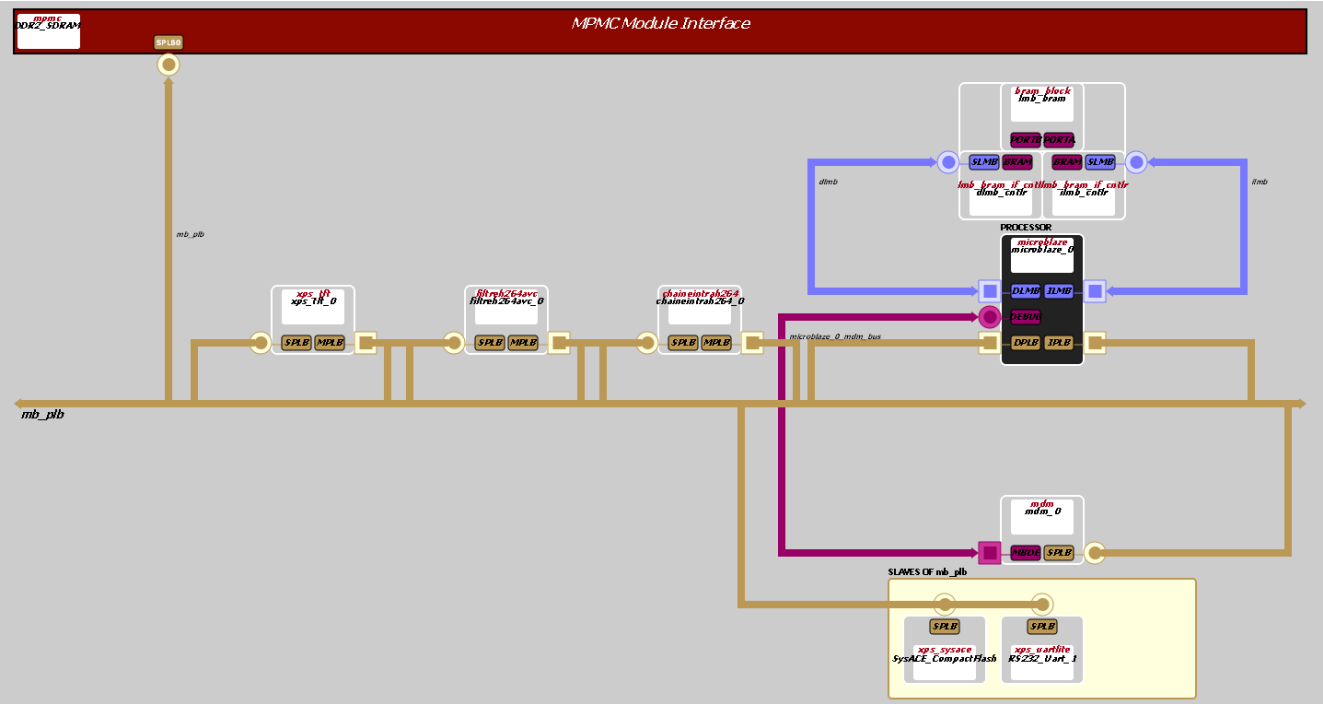
# Annexe B2

## Résultats de synthèse de l'implantation de l'encodeur H.264/AVC sur un système à base de processeur MicroBlaze




MPMC Module Interface
SPLB0
mb_plb
bram_block
lmb_bram
PORTB
PORTA
SLMB
BRAM
dlmb_cntlr
ilmb_cntlr
PROCESSOR
microblaze
DLMB
ILMB
DEBUG
DPLB
IPLB
xps_tft
filtreh264avc
chaineintrah264
SPLB
MPLB
microblaze_0_mdm_bus
mdm
MBDE
SLAVES OF mb_plb
xps_sysace
xps_uartlite