N° d'ordre 3409

THESE

présentée à

L'UNIVERSITE BORDEAUX 1

ECOLE DOCTORALE DES SCIENCES PHYSIQUES ET DE L'INGENIEUR

POUR OBTENIR LE GRADE DE

DOCTEUR

SPECIALITE : ELECTRONIQUE

Par

Ahmed BEN ATITALLAH

Etude et Implantation d'Algorithmes de Compression d'Images dans un Environnement Mixte Matériel et Logiciel

---

Soutenue le : 11 Juillet 2007

Après avis de :

M.	Noureddine ELLOUZE	Professeur à l'ENIT, Tunis, Tunisie	**Rapporteur**
M.	Patrick GARDA	Professeur à l'Université Paris VI	**Rapporteur**

Devant la Commission d'examen formée de :

M.	Lotfi KAMOUN	*Professeur à l'ENIS, Sfax, Tunisie*	**Président**
M.	Patrick GARDA	*Professeur à l'Université Paris VI*	**Rapporteur**
M.	Noureddine ELLOUZE	*Professeur à l'ENIT, Tunis, Tunisie*	**Rapporteur**
M.	Philippe MARCHEGAY	*Professeur à l'ENSEIRB*	
M.	Nouri MASMOUDI	*Professeur à l'ENIS, Sfax, Tunisie*	
M.	Patrice KADIONIK	*Maître de Conférences à l'ENSEIRB*	
M.	Patrice NOUEL	*Maître de Conférences à l'ENSEIRB*	

-2007-

A mes parents
A ma famille
A tous ceux qui me tiennent à cœur

REMERCIEMENTS

Cette thèse s'est effectuée en cotutelle entre l'équipe circuits et systèmes du Laboratoire d'Electronique et des Technologies de l'Information (LETI) de l'ENIS à Sfax, Tunisie ainsi que dans l'équipe Circuits Intégrés Numériques du Laboratoire de l'Intégration du Matériau au Système (IMS) de l'Université de Bordeaux et de l'ENSEIRB.

Je remercie Monsieur le Professeur Nouri MASMOUDI ainsi que Monsieur le Professeur Philippe MARCHEGAY pour m'avoir accueilli au sein de leur équipe et pour l'intérêt qu'ils ont porté au déroulement de mes travaux.

Je tiens à présenter ma vive gratitude à Monsieur Patrice KADIONIK, Maître de conférences à l'ENSEIRB, co-encadrant de ma thèse et Monsieur Patrice NOUEL, Maître de conférences à l'ENSEIRB, qui ont contribué activement à la réalisation de mes travaux, d'une part pour leurs conseils efficaces, leurs grandes compétences mais aussi pour leurs grandes qualités humaines.

Je remercie Monsieur le Professeur Lotfi KAMOUN, Professeur à l'ENIS, Tunisie pour l'honneur qu'il m'a fait en acceptant de présider le jury de cette thèse.

Je remercie vivement Monsieur Noureddine ELLOUZE, Professeur à l'ENIT, Tunisie et Monsieur Patrick GARDA, Professeur à l'Université Paris VI, pour l'intérêt qu'ils ont porté à mes travaux en acceptant d'examiner ce mémoire et d'en être les rapporteurs.

Je remercie vivement Monsieur Fahmi GHOZZI, Maître assistant à l'ISECS, Tunisie pour sa participation à la réalisation de ce travail.

Enfin, mes remerciements vont aussi à tous ceux qui ont participé plus ou moins indirectement au bon déroulement de ma thèse.

SOMMAIRE

INTRODUCTION GENERALE 17

CHAPITRE I: LES NORMES DE CODAGE VIDEO 23

I.1 INTRODUCTION : 23
I.2 PRINCIPE DE BASE : 23
I.2.1 Définition d'une image et des types d'images : *23*
I.2.2 Changement d'espace de couleur : *24*
I.2.3 Définition de la vidéo : *24*
I.2.4 La compression vidéo : *25*
I.2.5 Les données vidéo : *28*
I.3 LES NORMES DE CODAGE VIDEO : 29
I.3.1 Principes : *30*
I.3.2 Les normes de l'UIT-T : *32*
I.3.3 Les normes de l'ISO/MPEG : *33*
I.4 ETUDE DE LA NORME H.263 : 34
I.4.1 Introduction : *34*
I.4.2 Le principe du processus de compression : *35*
I.4.3 Estimation de mouvement : *36*
I.4.4 La Transformée de Cosinus Discrète (TCD/TCDI) : *40*
I.4.5 Quantification : *41*
I.4.6 Codage entropique : *42*
I.5 CONCLUSION : 45

CHAPITRE II: LA CONCEPTION DES SYSTEMES NUMERIQUES 47

II.1 INTRODUCTION : 47
II.2 METHODOLOGIES DE CONCEPTION DES SYSTEMES NUMERIQUES : 47
II.2.1 Généralités : *47*
II.2.2 Réalisation d'un système sur puce SoC (System on Chip) ou SoPC (System on Programmable Chip) : *48*
II.2.3 Les différentes familles de blocs IP (Intellectual Property) : *49*
II.3 LES CIRCUITS A LOGIQUE PROGRAMMABLE : 50
II.3.1 Types d'architectures et éléments des circuits FPGA : *50*
II.3.2 Les différents éléments d'un circuit FPGA : *50*
II.3.3 Exemple de circuit FPGA : la famille Altera Stratix II : *52*
II.4 ARCHITECTURES RECONFIGURABLES EMBARQUEES : 53
II.4.1 Architecture des processeurs : *53*
II.4.2 Les processeurs pour les SoPCs : *55*
II.4.3 Le processeur embarqué NIOS : *56*
II.5 LES SYSTEMES NUMERIQUES EMBARQUES : 60
II.5.1 Définition : *60*
II.5.2 Les contraintes des systèmes embarqués : *60*
II.5.3 Terminaux visiophoniques : *61*
II.5.4 Conception de systèmes embarqués dans l'approche codesign : *63*
II.6 LINUX POUR LES SYSTEMES EMBARQUES : 64
II.6.1 Le système d'exploitation Linux : *64*
II.6.2 Linux et l'embarqué : *64*
II.6.3 Linux embarqué : *65*
II.6.4 Mise en œuvre de µClinux sur NIOS II : *66*
II.7 CONCLUSION : 67

CHAPITRE III: PLATEFORME MATERIELLE DE TRAITEMENT VIDEO 71

III.1 INTRODUCTION : 71
III.2 ENVIRONNEMENT DE DEVELOPPEMENT D'UN SYSTEME SOPC : 71
III.2.1 Conception d'un système SoPC : *71*
III.2.2 Linux embarqué pour système SoPC : *73*
III.3 CARTE DE DEVELOPPEMENT : CARTE ALTERA STRATIX II 73

III.3.1 Description des éléments de la carte Stratix II : ... *74*
III.3.2 Caractéristiques du composant Stratix II d'Altera : ... *74*
III.4 SYSTEME D'ACQUISITION ET DE TRAITEMENT VIDEO : ... 74
III.4.1 Périphérique d'acquisition d'images : ... *75*
III.4.2 Restitution d'images : ... *79*
III.4.3 Système vidéo Temps Réel : ... *82*
III.4.4 Conception du système multimédia embarqué : ... *85*
III.5 CONCLUSION : ... 87

CHAPITRE IV: IMPLANTATION DU CODEUR H.263 ... 89

IV.1 INTRODUCTION : ... 89
IV.2 ETUDE DE LA COMPLEXITE : ... 89
IV.3 IMPLANTATION LOGICIELLE DU CODEUR H.263 : ... 90
IV.4 ESTIMATION DE MOUVEMENT : ... 91
IV.4.1 Algorithme de recherche exhaustive : ... *91*
IV.4.2 Réalisation d'un coprocesseur pour le calcul du SAD : ... *93*
IV.4.3 Accélération de la recherche des vecteurs de mouvement : ... *97*
IV.4.4 Evaluation des performances en fonction de l'estimation de mouvement : ... *102*
IV.5 IMPLANTATION MATERIELLE DE LA TCD/TCDI : ... 105
IV.5.1 Architecture de Loeffler : ... *105*
IV.5.2 Distribution arithmétique : ... *109*
IV.5.3 Coprocesseur TCD/TCDI bidimensionnelle : ... *113*
IV.6 IMPLANTATION MATERIELLE DE LA Q/QI : ... 119
IV.6.1 Description des instructions de Q/QI : ... *120*
IV.6.2 Architecture interne des instructions de Q/QI : ... *121*
IV.6.3 Evaluation des performances des instructions Q/QI : ... *123*
IV.7 IMPLANTATION LOGICIELLE/MATERIELLE DU CODEUR H.263 : ... 124
IV.7.1 Personnalisation et génération du système multimédia embarqué : ... *124*
IV.7.2 Évaluation des performances du codeur H.263 : ... *127*
IV.8 CONCLUSION : ... 134

CONCLUSION GENERALE ... 137

BIBLIOGRAPHIE ... 139

ANNEXES ... 151

GLOSSAIRE ... 183

Liste des figures

FIGURE 1. LES LOIS EMPIRIQUES DE L'EVOLUTION DE L'INTEGRATION ELECTRONIQUE 17
FIGURE 2. EXEMPLE DE SYSTEME SUR PUCE SOC 18
FIGURE 3. ELEMENT D'UNE IMAGE : LE PIXEL 23
FIGURE 4. SUPERPOSITION DES TROIS COULEURS : ROUGE, VERT ET BLEU 24
FIGURE 5. PRINCIPE DE BALAYAGE UTILISE POUR LA VIDEO ET LA TELEVISION 25
FIGURE 6. EXEMPLE D'EXPLOITATION DE LA REDONDANCE TEMPORELLE 27
FIGURE 7. STRUCTURE DES MACROBLOCS (A) 4 : 2 : 0, (B) 4 : 2 : 2 ET (C) 4 : 4 : 4 28
FIGURE 8. HIERARCHIE DES DONNEES DANS LE FLUX VIDEO 29
FIGURE 9. LES NORMES DE CODAGE VIDEO 30
FIGURE 10. CODAGE D'UNE IMAGE I 31
FIGURE 11. CODAGE D'UNE IMAGE P OU B 31
FIGURE 12. EXEMPLE DE CONFIGURATION DE CODAGE D'UNE SEQUENCE VIDEO 31
FIGURE 13. DIAGRAMME FONCTIONNEL D'UN CODEC VIDEO 32
FIGURE 14. STRUCTURE D'UNE IMAGE DANS LA NORME H.263 35
FIGURE 15. SCHEMA FONCTIONNEL D'UN CODEUR VIDEO DE LA NORME H.263 36
FIGURE 16. PREDICTION *FORWARD* 37
FIGURE 17. PREDICTION *BACKWARD* 37
FIGURE 18. RECHERCHE D'UN VECTEUR DE MOUVEMENT 38
FIGURE 19. ILLUSTRATION D'UN BLOC DE TAILLE 8X8 39
FIGURE 20. IMAGES DE LA BASE DE L'ESPACE TCD 40
FIGURE 21. LE PARCOURS ZIGZAG 42
FIGURE 22. L'ARBRE DE HUFFMAN 43
FIGURE 23. GENERATION DE L'ETIQUETTE POUR LA SEQUENCE "ACAAB" 45
FIGURE 24. EVOLUTION DE LA CONCEPTION NUMERIQUE 48
FIGURE 25. SOC BASE CŒURS DE PROCESSEURS 49
FIGURE 26. STRUCTURE DE BASE D'UN COMPOSANT PROGRAMMABLE 51
FIGURE 27. NIVEAU SUPERIEUR DE LA HIERARCHIE DE L'ARCHITECTURE DU CIRCUIT STRATIX II 52
FIGURE 28. ARCHITECTURE D'UN BLOC DSP 53
FIGURE 29. ARCHITECTURE D'UN ALM AU NIVEAU INFERIEUR DE LA HIERARCHIE DE L'ARCHITECTURE DU CIRCUIT STRATIX II 53
FIGURE 30. ARCHITECTURE DE VON NEUMANN 54
FIGURE 31. ARCHITECTURE HARVARD 54
FIGURE 32. SYSTEME ALTERA NIOS 56
FIGURE 33. CPU NIOS 57
FIGURE 34. INSTRUCTION PERSONNALISEE DU PROCESSEUR NIOS II 57
FIGURE 35. IMPLANTATION DU PROCESSEUR NIOS II SUR DIFFERENTS CIRCUITS FPGA D'ALTERA 58
FIGURE 36. BUS AVALON 59
FIGURE 37. CYCLE DE LECTURE 59
FIGURE 38. CYCLE D'ECRITURE 60
FIGURE 39. UNIVERSALITE DE L'UMTS 61
FIGURE 40. SCHEMA FONCTIONNEL D'UN TERMINAL MOBILE MULTIMEDIA 62
FIGURE 41. CONCEPTION TRADITIONNELLE ET *CODESIGN* 63
FIGURE 42. SYSTEMES D'EXPLOITATION POUR LES APPLICATIONS EMBARQUEES 65
FIGURE 43. IDE QUARTUS II 71
FIGURE 44. FLOT DE CONCEPTION 72
FIGURE 45. SOPC BUILDER ET MAPPING MEMOIRE 72
FIGURE 46. IDE ECLIPSE 73
FIGURE 47. CARTE ALTERA STRATIX 2S60 74
FIGURE 48. SYSTEME D'ACQUISITION, TRAITEMENT ET RESTITUTION VIDEO 75
FIGURE 49. ARCHITECTURE INTERNE DU CIRCUIT 75
FIGURE 50. SYNOPTIQUE DE L'INTERFACE CAMERA 76
FIGURE 51. L'ENTITE DE L'INTERFACE CAMERA 77
FIGURE 52. CONNEXION DE L'INTERFACE CAMERA AVEC LE SYSTEME 77

FIGURE 53. INTERFACE CAMERA 78
FIGURE 54. SIMULATION DE L'INTERFACE CAMERA 79
FIGURE 55. CARTE D'INTERFACE LANCELOT 80
FIGURE 56. SYNOPTIQUE DE LA CARTE D'EXTENSION 80
FIGURE 57. SYNOPTIQUE DE L'INTERFACE VGA 80
FIGURE 58. L'ENTITE DE L'INTERFACE VGA 81
FIGURE 59. SIMULATION DE L'INTERFACE VGA 81
FIGURE 60. SYNCHRONISATION D'UNE TRAME 82
FIGURE 61. ARCHITECTURE DU SYSTEME 83
FIGURE 62. GESTION DE LA SYNCHRONISATION DE L'INTERFACE CAMERA ET L'INTERFACE VGA 84
FIGURE 63. SYSTEME EMBARQUE NIOS II 85
FIGURE 64. SOPC BUILDER UTILISANT L'INTERFACE CAMERA ET L'INTERFACE VGA 85
FIGURE 65. EXECUTION DE L'APPLICATION µCLINUX *CAMVGA* 86
FIGURE 66. SYSTEME MULTIMEDIA EMBARQUE 87
FIGURE 67. SEQUENCES DE TEST 89
FIGURE 68. REPARTITION DU TEMPS CPU PAR BLOC DE TRAITEMENT POUR LES SEQUENCES (A) MISS AMERICA ET (B) FOREMAN 90
FIGURE 69. REPARTITION DU TEMPS CPU DANS L'EM POUR LES SEQUENCES (A) MISS AMERICA ET (B) FOREMAN 91
FIGURE 70. ORGANIGRAMME DE L'ALGORITHME FSBM 92
FIGURE 71. PRINCIPE DE FONCTIONNEMENT DU COPROCESSEUR SAD 93
FIGURE 72. L'ENTITE DU COPROCESSEUR SAD 94
FIGURE 73. LECTURE ET ECRITURE DES DONNEES DU COPROCESSEUR SAD 94
FIGURE 74. ARCHITECTURE DE L'UNITE DE TRAITEMENT (UT) 95
FIGURE 75. CALCUL DU SAD 1X16 95
FIGURE 76. ARCHITECTURE POUR LE CALCUL DU SAD 16X16 96
FIGURE 77. NOMBRE DE CYCLES POUR LE CALCUL DE SAD EN SW ET HW 97
FIGURE 78. MODELES DE RECHERCHE UTILISES DANS L'ALGORITHME *DIAMOND SEARCH* 98
FIGURE 79. EXEMPLE D'UNE RECHERCHE EN DIAMANT 99
FIGURE 80. OPTIMISATION DE L'ALGORITHME *DIAMOND SEARCH* 99
FIGURE 81. EXEMPLE D'UNE RECHERCHE EN PETIT DIAMANT 100
FIGURE 82. MODELES DE RECHERCHES UTILISEES DANS L'ALGORITHME *CROSS-DIAMOND SEARCH* 100
FIGURE 83. EXEMPLE D'UNE RECHERCHE EN CROIX DIAMANT 101
FIGURE 84. MODELES DE RECHERCHE UTILISES DANS L'ALGORITHME *HEXAGONAL SEARCH* 101
FIGURE 85. EXEMPLE D'UNE RECHERCHE EN HEXAGONE 102
FIGURE 86. OPTIMISATION DE L'ALGORITHME *HEXAGONAL SEARCH* 102
FIGURE 87. QUALITE DE DIFFERENTS ALGORITHMES D'EM POUR LES SEQUENCES (A) MISS AMERICA ET (B) FOREMAN 103
FIGURE 88. NOMBRE DE CYCLES D'HORLOGE DES DIFFERENTS ALGORITHMES D'EM POUR LES SEQUENCES (A) MISS AMERICA ET (B) FOREMAN 104
FIGURE 89. NOMBRE DE CYCLES POUR LE CALCUL DU SAD PAR SW ET HW POUR LA RECHERCHE EN HEXAGONE DES SEQUENCES (A) MISS AMERICA ET (B) FOREMAN 104
FIGURE 90. ARCHITECTURE DE LOEFFLER 106
FIGURE 91. ARCHITECTURE MODIFIEE DE L'ALGORITHME DE LOEFFLER 107
FIGURE 92. EXEMPLE D'UTILISATION DE LA PRIMITIVE DSP 108
FIGURE 93. SIMULATION TEMPORELLE DE LA TCD-1D PAR L'ARCHITECTURE DE LOEFFLER 108
FIGURE 94. CIRCUIT ROM ACCUMULATEUR (RAC) 111
FIGURE 95. CIRCUIT DE TCD-1D 111
FIGURE 96. SIMULATION TEMPORELLE DE LA TCD-1D PAR LA METHODE DA 112
FIGURE 97. (A) RESSOURCES UTILISEES ET (B) FREQUENCE DE FONCTIONNEMENT POUR LA TCD/TCDI-1D AVEC LES DIFFERENTES METHODES 113
FIGURE 98 . NOMBRE DE CYCLES POUR LE TRAITEMENT D'UN BLOC 8X8 PAR LA TCD/TCDI-1D AVEC LES DIFFERENTES METHODES 113
FIGURE 99. COPROCESSEUR TCD/TCDI-2D 114
FIGURE 100. CONNEXION DU COPROCESSEUR TCD/TCDI-2D AVEC NIOS II 114
FIGURE 101. ENTITE DU COPROCESSEUR TCD/TCDI-2D 115
FIGURE 102. TRANSFERT DE DONNEES ENTRE LE COPROCESSEUR ET LA MEMOIRE PAR DMA 116
FIGURE 103. SIMULATION TEMPORELLE DU COPROCESSEUR TCD-2D 117
FIGURE 104. INTERFACE DE L'INSTRUCTION POUR Q/QI 120
FIGURE 105. EXEMPLE DE DECLARATION ET D'UTILISATION DES INSTRUCTIONS DE Q/QI 121

FIGURE 106. ARCHITECTURE DE L'INSTRUCTION POUR LA QUANTIFICATION INTRA 121
FIGURE 107. ARCHITECTURE DE L'INSTRUCTION POUR LA QUANTIFICATION INTER 121
FIGURE 108. ARCHITECTURE DE L'INSTRUCTION POUR LA DEQUANTIFICATION 122
FIGURE 109. SIMULATION TEMPORELLE DE L'INSTRUCTION Q_INTER 122
FIGURE 110. GENERATION DU SYSTEME MULTIMEDIA EMBARQUE AVEC SOPC BUILDER 125
FIGURE 111. SYSTEME MULTIMEDIA EMBARQUE 125
FIGURE 112. EXECUTION DU CODEUR H.263 SOUS µCLINUX 126
FIGURE 113. PLACEMENT ET ROUTAGE DE NOTRE SYSTEME MULTIMEDIA SUR FPGA STRATIX II 127
FIGURE 114. REPARTITION DU TEMPS CPU EN UTILISANT LA SOLUTION (A)SW ET (B) HW/SW POUR LA SEQUENCE MISS AMERICA 127
FIGURE 115. REPARTITION DU TEMPS CPU EN UTILISANT LA SOLUTION (A) SW ET (B) HW/SW POUR LA SEQUENCE FOREMAN 128
FIGURE 116. (A) ORIGINALE, (B) RECONSTRUITE PAR SW ET (C) RECONSTRUITE PAR HW/SW DE LA 8EME IMAGE DES SEQUENCES VIDEO DE TEST POUR QP=8 132
FIGURE 117. (A) ORIGINALE, (B) RECONSTRUITE PAR SW ET (C) RECONSTRUITE PAR HW/SW DE LA 40EME IMAGE DES SEQUENCES VIDEO DE TEST POUR QP=13 133

Liste des tableaux

TABLE 1. POURCENTAGE DES COMPOSANTES DE CHROMINANCE PAR RAPPORT A LA COMPOSANTE DE LA LUMINANCE SUIVANT LA DIRECTION HORIZONTALE ET VERTICALE 28
TABLE 2. FORMATS DES IMAGES 34
TABLE 3. TABLE DE FREQUENCE 43
TABLE 4. TABLE DE CODIFICATION 44
TABLE 5. PROBABILITES DES SYMBOLES 44
TABLE 6. CARACTERISTIQUE DES PROCESSEURS RISC ET CISC 55
TABLE 7. DIFFERENCE ENTRE NIOS ET NIOS II 58
TABLE 8. LES DIFFERENTES VERSIONS DU NIOS II 58
TABLE 9. EXEMPLES DE TERMINAUX VISIOPHONIQUES 62
TABLE 10. CONNEXION DE LA CARTE D'INTERFACE CAMERA A LA CARTE DE DEVELOPPEMENT 76
TABLE 11. SIGNAUX DE L'INTERFACE CAMERA 78
TABLE 12. SIGNAUX ENTRE LE COPROCESSEUR SAD ET LE BUS AVALON 94
TABLE 13. POURCENTAGE DU TEMPS CPU EN FONCTION DU TRAITEMENT 103
TABLE 14. ÉVALUATION DU COPROCESSEUR SAD POUR LA RECHERCHE EN HEXAGONE DES SEQUENCES MISS AMERICA ET FOREMAN 105
TABLE 15. ALGORITHMES DE LA TCD/TCDI 105
TABLE 16. DIFFERENTS SYMBOLES UTILISES DANS L'ARCHITECTURE DE LOEFFLER 106
TABLE 17. RESULTATS D'IMPLANTATION DE LA TCD/TCDI 107
TABLE 18. RESULTATS D'IMPLANTATION DE LA TCD/TCDI AVEC BLOCS DSPS 108
TABLE 19. RESULTATS D'IMPLANTATION DE LA TCD/TCDI PAR LA METHODE DA 112
TABLE 20. SIGNAUX ENTRE LE COPROCESSEUR EST LE BUS AVALON 115
TABLE 21. RESULTATS D'IMPLANTATION DU COPROCESSEUR TCD/TCDI-2D 116
TABLE 22. ÉVALUATION DU COPROCESSEUR TCD/TCDI-2D POUR LES SEQUENCES MISS AMERICA ET FOREMAN 117
TABLE 23. ÉVALUATION DE LA QUALITE DE CODAGE SUR DIFFERENTES SEQUENCES POUR LE COPROCESSEUR TCD/TCDI-2D 118
TABLE 24. SIGNAUX ENTRE L'INSTRUCTION DE Q/QI ET L'UAL 120
TABLE 25. RESULTATS D'IMPLANTATION DES INSTRUCTIONS DE Q/QI 122
TABLE 26. ÉVALUATION DES INSTRUCTIONS DE Q/QI POUR LES SEQUENCES MISS AMERICA ET FOREMAN 123
TABLE 27. ÉVALUATION DE LA QUALITE DE CODAGE SUR DIFFERENTES SEQUENCES POUR LES INSTRUCTIONS DE Q/QI 124
TABLE 28. POURCENTAGE DU TEMPS CPU EN FONCTION DU BLOC DE TRAITEMENT DU CODEUR H.263 POUR LES SEQUENCES MISS AMERICA ET FOREMAN 128
TABLE 29. ÉVALUATION DE LA QUALITE DE CODAGE SUR DIFFERENTES SEQUENCES DE TESTS 130
TABLE 30. COMPARAISON DE PERFORMANCE DU CODEUR H.263 134

Partie I : Introduction générale

Introduction Générale

Introduction :

Dans le domaine des télécommunications numériques, certains traitements algorithmiques sont à la veille de transformer les moyens quotidiens de communication. Ainsi, l'audiovisuel est le théâtre de bouleversements dus notamment à l'intégration d'un concept nouveau : le *codesign* et l'intégration sur silicium d'algorithmes de compression de données. Les techniques de compression permettent aujourd'hui d'envisager de transmettre en Temps Réel des données multimédia audio/vidéo.

L'objectif de la compression vidéo est de réduire le nombre de bits nécessaire à la représentation de l'information portée par une séquence vidéo. Les améliorations apportées par la compression ne sont pas tout simplement dues à l'élimination des données redondantes, mais plutôt l'abandon des informations estimées non pertinentes, comme par exemple les informations sur des détails non perceptibles facilement à l'œil nu. Ainsi la compression fournit une représentation compacte *à priori* indiscernable visuellement de l'image originale, bien qu'en général, l'image compressée soit tout à fait différente de l'originale.

Cependant, le fait de comprimer les images produites par un dispositif d'acquisition ou de traitement introduit de nouvelles contraintes liées aux performances du processus de compression lui-même. 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 [1].

Cependant, si l'on se réfère aux lois empiriques prédictives (figure 1) de l'évolution des densités d'intégration (*Loi de Moore)* et de la complexité algorithmique (*Loi de Shannon)* pour l'évolution des applications multimédia, on constate que l'écart entre les deux courbes ne cesse de se creuser [2].

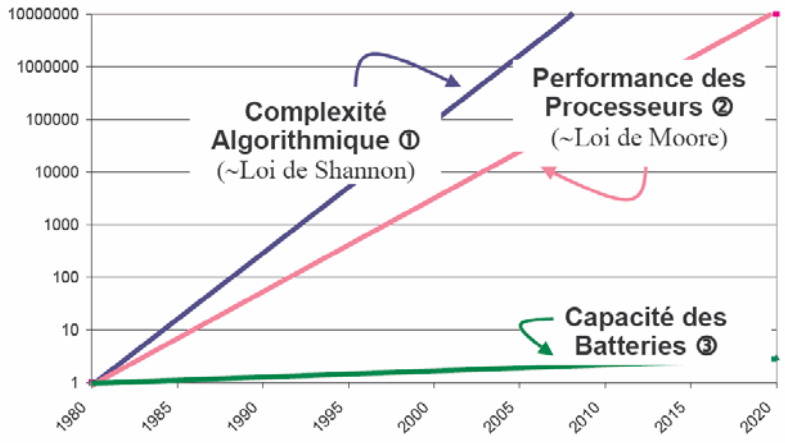


Figure 1. Les lois empiriques de l'évolution de l'intégration électronique

Parallèlement, des contraintes plus spécifiques concernant notamment le marché de l'embarqué mettent en relief des limitations préoccupantes notamment en ce qui concerne la consommation d'énergie. On ne pourra pas, à priori, compter sur les progrès des méthodes de stockage chimique de l'énergie si les tendances de la figure 1 (courbe 3) se confirment.

On voit qu'aujourd'hui le choix d'une architecture pour un système multimédia embarqué n'est pas trivial et qu'un compromis doit être trouvé entre flexibilité, consommation, performance, coût, rapidité de conception (*Time To Market*)...

Dans ce contexte, avec l'évolution des densités d'intégration en microélectronique, on est aujourd'hui capable de concevoir des systèmes entiers sur une même puce nommés Système sur Silicium (*System on Chip* (SoC) ou *System On Programmable Chip* (SoPC)). La figure ci-dessous illustre un exemple de SoC. Sur ces systèmes miniatures, on trouve aussi bien des cœurs de processeur, des bus de communication (qui permettent d'interconnecter les différents blocs), des Convertisseurs Analogique/Numérique (CAN) et Numérique/Analogique (CNA), des contrôleurs vidéo...

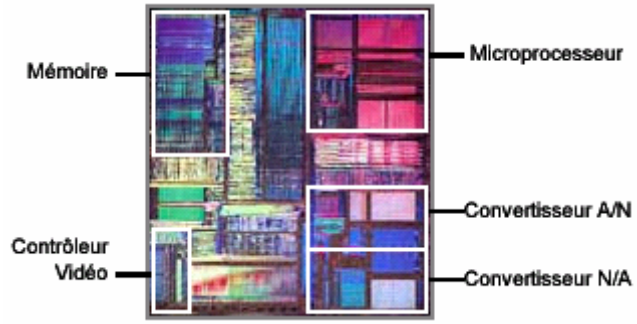


Figure 2. Exemple de système sur puce SoC

Par ailleurs, avec un niveau d'intégration de plus en plus important sur silicium, une approche système est maintenant adoptée dans la conception de systèmes numériques complexes tels que les systèmes multimédia embarqués. Le développement de systèmes numériques en langage textuel de description matérielle (VHDL) fait place à une approche objet plus système avec la mise en œuvre de blocs de propriété intellectuelle ou blocs IP (*Intellectual Property*). Une approche de conception conjointe ou *codesign* est adoptée pour un meilleur partitionnement entre matériel et logiciel. Une architecture matérielle plus complexe et fortement parallèle nécessite l'utilisation d'un système d'exploitation capable de gérer les différentes unités qui la composent, tout en tenant compte des critères de performance et de faible consommation, et de superviser l'exécution des applications qui s'y exécutent et qui sont particulièrement caractérisées par des contraintes de Temps Réel.

Objectifs de la thèse :

C'est dans le contexte général dressé précédemment que se situent les travaux de cette thèse.

La complexité des systèmes embarqués et notamment multimédia impose de mettre en œuvre une méthodologie de conception conjointe logicielle/matérielle (*codesign*) afin de la maîtriser au mieux.

Dans le cadre de la compression vidéo et notamment à bas débit (que l'on utilise beaucoup sur Internet), le premier et principal objectif est de contribuer à la constitution d'une

bibliothèque de modules IP développés en langage VHDL les plus génériques possibles afin de constituer des accélérateurs de traitement matériels de certains blocs d'une norme de compression vidéo. Ces modules IP ne doivent pas être contraints à l'usage d'une plateforme matérielle donnée.

Le deuxième objectif est bien sûr de construire une plateforme vidéo où l'on retrouve la partie acquisition vidéo, la partie traitement qui permettra de tester les modules IP dans le cadre d'une norme et enfin la partie visualisation. Cette plateforme constituant ainsi le système multimédia embarqué sert de plateforme de tests et de validations des modules IP.

Le troisième objectif et non des moindres, est de montrer aussi que l'on ne peut plus s'affranchir de la logique programmée, c'est-à-dire de logiciel, dès que la complexité à gérer devient importante. Le choix de l'usage d'un système d'exploitation pour notre système embarqué multimédia permet de gérer au mieux la complexité algorithmique. Le choix d'un système d'exploitation libre comme Linux pour l'embarqué a semblé judicieux à juste titre : Logiciel Libre, accès aux sources, flexibilité, configurabilité et connectivité réseau sont des atouts importants. La possibilité aussi de pouvoir utiliser d'autres briques logicielles libres (dont les avantages ressemblent forts à ceux des modules IP) est aussi une force.

C'est en gardant à l'esprit ces différents objectifs que les travaux de cette thèse ont été réalisés.

Plan de la thèse :

Cette thèse se compose de deux parties principales, elles-mêmes subdivisées en deux chapitres.

Dans la première partie, nous introduisons le contexte de l'étude. Le premier chapitre présente tout d'abord le concept de codage vidéo afin d'avoir une vision détaillée des principes employés dans tout type de système de compression. Nous parlerons en particulier de la norme H.263 de l'organisme UIT-T, de sa structure et de ses éléments principaux.

Le deuxième chapitre traite de la conception des systèmes multimédia embarqués sur les SoCs. On prendra comme exemple les systèmes électroniques mobiles de demain, dits de troisième génération. Dans ce cadre, on présentera une description fonctionnelle du terminal mobile, ainsi que les réalisations commerciales des terminaux de visiophonie. L'étude de faisabilité d'un tel système a nécessité la mise en place d'une méthodologie de conception au niveau système prenant en compte les contraintes de l'embarqué.

La deuxième partie traite l'adéquation algorithme-architecture pour les systèmes multimédia embarqués. Le premier chapitre présente la mise en place d'une plateforme matérielle d'acquisition et de restitution vidéo servant à l'évaluation de notre méthodologie de conception logicielle/matérielle pour les systèmes multimédia embarqués. Cette plateforme est complétée d'une interface caméra pour l'acquisition et d'une interface VGA pour la restitution des images. L'ensemble du développement a été intégré sous le système d'exploitation Linux embarqué (µClinux) et exécuté par un processeur embarqué de type RISC.

Dans le deuxième chapitre, on définit le codeur vidéo retenu pour notre étude. Il s'agit d'un codeur basé sur des blocs respectant la norme H.263 (recommandation UIT-T). Tout d'abord, une étude au niveau algorithmique des différentes configurations du codeur vidéo sur notre plateforme est réalisée afin de fournir des éléments sur la complexité et les performances. Ces éléments sont ensuite discutés et nous choisissons la configuration la

mieux adaptée à notre système multimédia embarqué. Ensuite, la question de l'architecture du système de codage vidéo est introduite. Enfin, nous analyserons l'efficacité de notre implantation logicielle/matérielle (*codesign*).

Partie II : Contexte de l'étude

Cette première partie est divisée en deux chapitres. Le premier chapitre présente un tour d'horizon des codeurs vidéo normalisés afin de définir les différentes possibilités de codage qui s'offrent aujourd'hui. Dans le deuxième chapitre, on abordera la méthodologie de conception des systèmes numériques sur les SoCs.

Chapitre I: Les normes de codage vidéo

I.1 Introduction :

Depuis quelques années, le monde du multimédia a assisté à une véritable explosion de ses applications. Les nouvelles technologies qui paraissaient révolutionnaires à leur arrivée sur le marché, sont désormais entrées dans la vie de tous les jours. Il est maintenant impensable de ne pas pouvoir téléphoner, tout comme il serait impossible pour les entreprises et les particuliers de se passer de l'Internet. Si l'échange de la parole et du texte a longtemps paru suffisant, il s'agit maintenant de transmettre des documents vidéo et audio de natures diverses. Or, la vidéo numérique pose, par sa taille importante, de nombreux problèmes, que ce soit pour la transmission (surtout pour les applications Temps Réel) ou pour le stockage. Dans ce contexte, il était devenu indispensable d'élaborer un standard international de compression vidéo.

Les deux organismes les plus actifs dans la normalisation des systèmes de compression vidéo sont l'UIT-T (Union Internationale des Télécommunications) [3] et l'ISO/MPEG (*Moving Picture Experts Group*) [4]. Les codeurs issus des ces deux groupes de normalisation reposent sur des principes de base communs, mais différents de part leur configuration choisie en fonction de l'application.

Ce chapitre présente tout d'abord le concept de codage vidéo afin d'avoir une vision détaillée des principes employés dans tout type de système de compression. Puis nous parlerons en particulier de la norme H.263 de l'organisme UIT-T, de sa structure et de ses éléments principaux.

I.2 Principe de base :

I.2.1 Définition d'une image et des types d'images :

Une image est stockée en mémoire sous forme de collection de points élémentaires appelés pixels. Nous pouvons considérer une image numérique comme une page de nombres organisés en tableau ou en matrice. Chaque nombre représente les caractéristiques du pixel. La position de chaque pixel peut être exprimée par deux coordonnées sur l'axe horizontal X et l'axe vertical Y comme le montre la figure ci-dessous.

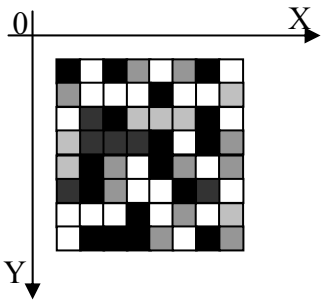


Figure 3. Elément d'une image : le pixel

Le codage d'un pixel dépend du type d'image et nous en recensons trois types :

- Les images à deux niveaux : une image en noir et blanc est l'exemple le plus courant. Toute image décrite avec deux valeurs correspond à ce type. Un bit suffira pour coder la valeur d'un pixel.

- Les images à plusieurs niveaux de gris : les images de nos téléviseurs en noir et blanc sont de ce type. La plupart des systèmes définissent 256 niveaux de gris. Mais, seuls 128 niveaux de gris sont détectables par l'œil.
- Les images couleurs : la couleur peut être codée, soit par composition de couleurs primaires, soit par composition d'informations de luminance et de chrominance. En fait, une couleur peut être représentée par un ensemble de trois coordonnées, c'est-à-dire qu'une couleur peut être reproduite par la superposition de trois couleurs primaires comme montre la figure 4. Le système RVB (Rouge-Vert-Bleu ou « RGB ») utilise les couleurs primaires : rouge, vert et bleu. La valeur du pixel doit représenter les composantes trichromatiques de la couleur. En général, nous disposons de huit bits pour coder une composante, soit 24 bits pour coder la valeur d'un pixel. Ce système RVB peut donc définir plus de 16 millions de couleurs. Des résultats expérimentaux ont prouvé que l'œil est beaucoup plus sensible aux variations fines d'intensité lumineuse (luminance) qu'à celles de la couleur (chrominance). Il en résulte que nous pouvons nous contenter de transmettre l'information de couleur avec moins de détails que l'information de luminance.

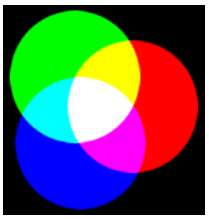


Figure 4. Superposition des trois couleurs : rouge, vert et bleu

I.2.2 Changement d'espace de couleur :

Toute longueur d'onde visible peut être visuellement simulée en convoluant le signal avec les fonctions de sensibilité des trois différents capteurs rétiniens du système visuel humain dit LMS (Large=565nm dit rouge, Medium=535nm dit vert, Short=430nm dit bleu). Dans le cas d'une compression avec perte, la reconstruction de chaque bande (RVB) risque de ne pas appréhender les structures de l'image de la même façon, engendrant différentes erreurs de reconstruction et par la même, de fausses couleurs visuellement choquantes. On préféra donc un espace de luminance et chrominance rouge et bleu YCrCb (ou YUV) où les primaires sont décorrélées, ce qui offre l'avantage de séparer les informations d'intensité lumineuse et de couleur. Un tel espace permet de gérer les premières avec plus de soin. A titre d'exemple, voici la matrice de passage de l'espace RVB à l'espace YUV :

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.229 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.212 & -0.523 & 0.11 \end{bmatrix} \times \begin{bmatrix} R \\ V \\ B \end{bmatrix}$$

I.2.3 Définition de la vidéo :

La vidéo est une succession d'images animées. Le principe fondamental de la vidéo est que l'œil humain a la possibilité de retenir pendant un certain temps (de l'ordre du dixième de seconde) toute image imprimée sur la rétine. Il suffit donc de faire défiler un nombre suffisant d'images par seconde, pour que l'œil ne se rende pas compte qu'il s'agit d'images distinctes. Il existe deux grandes familles de systèmes vidéo : les systèmes vidéo analogiques et les systèmes vidéo numériques.

I.2.3.1 La vidéo analogique :

La caméra balaye l'image bidimensionnelle qu'elle a devant elle par un faisceau d'électrons qui se déplace très rapidement de gauche à droite et plus lentement de haut en bas et produit une tension en fonction du temps. Elle enregistre ainsi l'intensité lumineuse, et à la fin du balayage, on a alors une trame. Le faisceau revient à l'origine pour recommencer. Le récepteur va recevoir cette intensité en fonction du temps, et pour reconstruire l'image, va répéter le processus de balayage.

Les paramètres précis de ce balayage varient d'un pays à l'autre mais deux grandes familles existent :

- En Europe (système PAL/SECAM, pour *Phase Alternating Line / SEquentiel Couleur Avec Mémoire*) ce système utilise 625 lignes (seulement 576 sont affichées), un rapport vertical/horizontal de 4/3 et un débit de 25 images par seconde.
- En Amérique et au Japon (système NTSC, pour *National Television Standards Committee*), on a seulement 525 lignes (483 affichées) et un débit de 30 images par seconde (figure 5).

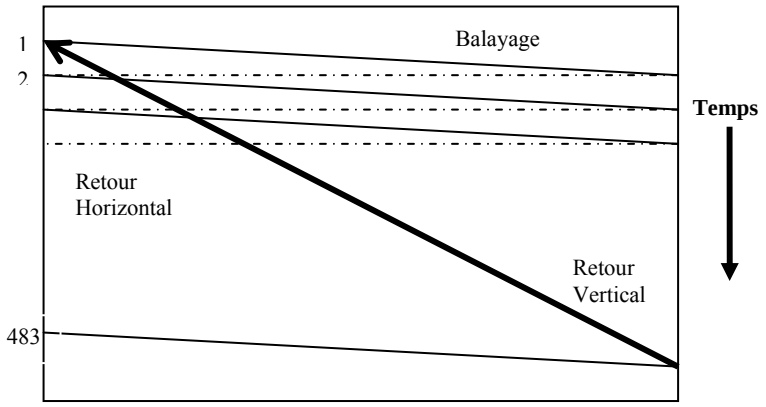


Figure 5. Principe de balayage utilisé pour la vidéo et la télévision

La télévision couleur utilise ce même principe de balayage mais au lieu de balayer l'image avec un seul faisceau, on utilise trois, un par couleur primaire : rouge, vert et bleu (RVB). Les signaux RVB sont ensuite transformés en un signal de luminance et deux signaux de chrominance. La télévision haute définition (TVHD) utilise le même principe mais double le nombre de lignes, pour obtenir une meilleure qualité. En outre, elle utilise un format 16/9 au lieu de 4/3 et ceci pour mieux s'adapter au format des films de cinéma [5].

I.2.3.2 La vidéo numérique :

La vidéo numérique est tout simplement une suite d'images formées d'une matrice de pixels. Pour obtenir des images en couleur, il faut utiliser au moins 8 bits par pixel, ce qui correspond à 256 couleurs. En fait, avec 8 bits par pixel, on obtient de la vidéo numérique noir et blanc de haute qualité. Pour la vidéo numérique couleur, on utilise 8 bits pour chaque couleur RVB, soit donc 24 bits par pixel, ce qui correspond à environ 16,8 millions de couleurs. Le principe de balayage utilisé est similaire à celui de la vidéo analogique.

I.2.4 La compression vidéo :

La compression vidéo est nécessaire pour la transmission des données vidéo numériques dans les réseaux à bande passante limitée d'aujourd'hui ainsi que pour les applications où le stockage constitue une limite. Imaginons, pour transmettre des données vidéo numériques à

24 bits par pixel brut échantillonné selon une résolution spatiale de 720x480 et une résolution temporelle de 30 images par seconde, il faudrait un débit binaire de 248 Mb/s.

Le principe fondamental de la compression vidéo est de réduire autant que possible les redondances d'informations dans les données sans que cela soit perceptible par l'œil humain. Il y a donc un compromis entre le taux de compression et la qualité de l'image. En fait, celle-ci devient de plus en plus médiocre avec l'augmentation du taux de compression.

Deux grandes méthodes de compression existent : la compression sans perte et la compression avec perte [6] :

- Dans le cas de la compression sans perte, les données décodées à l'arrivée par le récepteur sont strictement identiques aux données codées au départ par l'émetteur. Ce type de compression, permettant au mieux un taux de compression de 2:1, est évidemment insuffisant pour la compression vidéo.
- Dans le cas de la compression avec perte, les données à la sortie du décodeur sont différentes par rapport à celles à l'entrée du codeur. C'est ce type de compression qui est utilisé en vidéo car on peut accepter des pertes d'informations qui ne sont pas toujours visibles à l'œil et qui se traduisent par de nets gains de compression. En compression avec pertes, on peut atteindre des taux de compression allant jusqu'à 300:1.

La compression des données vidéo numériques sans dégradation significative de la qualité est possible lorsque les séquences vidéo affichent un degré élevé [7] :

- de redondance spatiale : corrélation entre les pixels voisins.
- de redondance temporelle : corrélation entre les images vidéo.
- de redondance psycho-visuelle : propriétés de la vue humaine.

I.2.4.1 Redondance spatiale :

C'est la corrélation dans chaque image prise indépendamment des autres qui présente des zones uniformes plus ou moins grandes dans lesquelles les pixels ont des valeurs très voisines. On peut diminuer cette redondance en codant chaque image séparément en JPEG (*Joint Photographic Experts Group*) [8]. L'utilité de ce codage est de pouvoir accéder de façon aléatoire à chaque image individuellement. Ce type de codage est appelé codage INTRA.

I.2.4.2 Redondance temporelle :

Deux images qui se suivent dans une séquence vidéo sont quasiment identiques. Ceci implique que la différence entre une image et la suivante soit relativement faible sauf lors d'un changement de plan. Autrement dit, la position d'un bloc de pixels varie généralement peu d'une image à l'autre. Le but est alors de ne stocker que ce qui est modifié lors du passage d'une image à une autre. Ce type de codage est appelé codage INTER.

La redondance des données, est illustrée dans la figure 6. Dans ces deux images, on remarque qu'il n'y a pas une grande différence puisque la camera n'a pas changé ni de plan ni de position. Donc, la seule différence se trouve dans les parties qui ont subi un mouvement. Ceci peut être démontré par la figure 6 (c) qui représente la différence entre les deux images successives (figure 6 (a) et figure 6 (b)) dont on a ajouté 128 afin de mieux visualiser les redondances temporelles. La partie grise de cette image correspond à la similitude des deux images (a) et (b).

(a) Image 143

(b) Image 144

(c) Image de différence

Figure 6. Exemple d'exploitation de la redondance temporelle

I.2.4.3 Redondance psycho-visuelle :

En plus d'éliminer les redondances temporelles et spatiales, la redondance psycho-visuelle est généralement réduite elle aussi. La mesure la plus significative est une résolution réduite du détail des couleurs par rapport au détail de la luminance qui permet un meilleur rapprochement avec les caractéristiques de la perception humaine. Les images vidéo comprennent trois matrices rectangulaires de données de pixels, lesquelles représentent le signal de luminance (luma Y) et deux signaux de chrominance (chroma Cb et Cr). Ces matrices correspondent à une représentation décomposée des trois couleurs primaires associées à chaque élément d'image. Pour la représentation des pixels vidéo numérique, il existe plusieurs formats. La figure 7 (a) illustre le format 4 : 2 : 0 où un pixel est codé sur 1,5 octet : un octet pour la luminance et ½ octet pour les deux chrominances rouge et bleu. Dans la figure 7 (b) on voit le format 4 : 2 : 2 où un pixel est codé sur 2 octets : 1 pour la luminance et ½ pour chacune des chrominances rouge et bleu. La figure 7 (c) illustre le format 4 : 4 : 4. Ce format représente chaque pixel sur 3 octets.

Le format le plus commun rencontré dans les normes de compression vidéo est huit bits avec sous-échantillonnage 4 : 2 : 0 : les deux composantes de chrominance sont réduites à la moitié de la résolution verticale et horizontale du composant luminance [9].

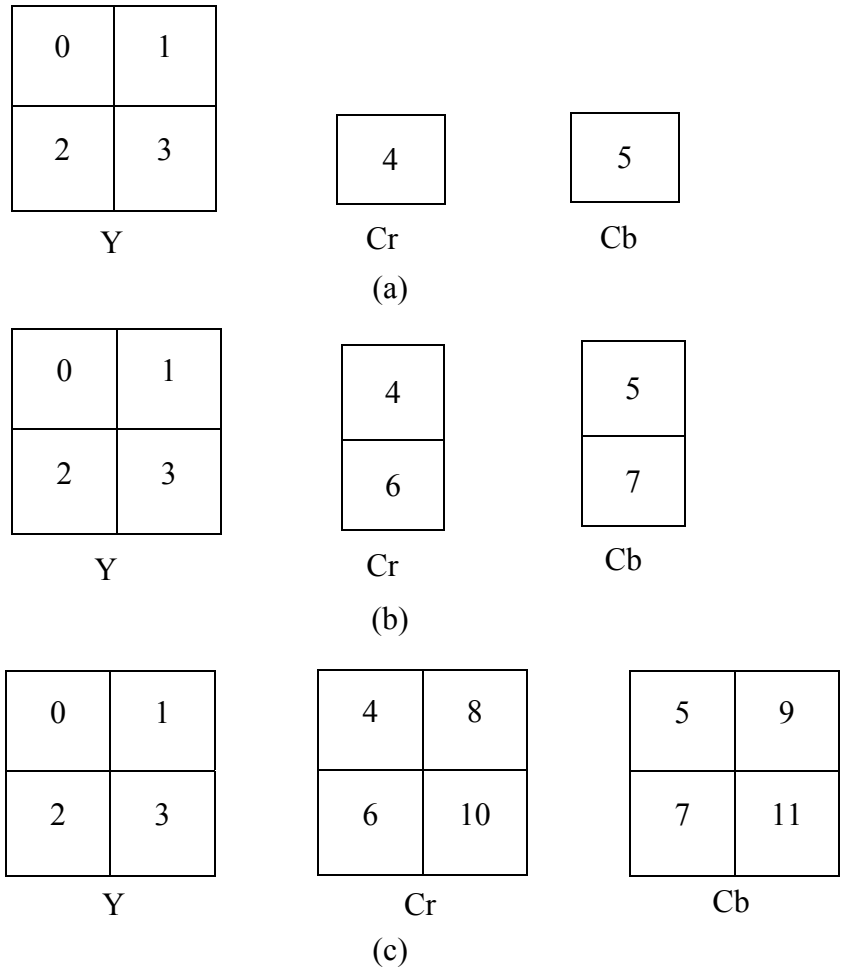


Figure 7. Structure des macroblocs (a) 4 : 2 : 0, (b) 4 : 2 : 2 et (c) 4 : 4 : 4

Pour une représentation plus détaillée du format d'image et pour distinguer le format 4 : 2 : 0 de 4 : 1 : 1, on donne dans la table 1 le pourcentage des composantes de chrominance par rapport à la composante de la luminance suivant la direction horizontale et verticale.

Table 1. Pourcentage des composantes de chrominance par rapport à la composante de la luminance suivant la direction horizontale et verticale

Format d'image	Horizontale [%]	Verticale [%]
4 :4 :4	100	100
4 :2 :2	50	100
4 :2 :0	50	50
4 :1 :1	25	100

I.2.5 Les données vidéo :

Dans le flux vidéo, les données sont hiérarchisées selon une manière bien précise. Tout d'abord, la séquence vidéo commence par un code de début de séquence, contient un ou plusieurs groupes d'images, et se termine par un code de fin de séquence.

Une image est constituée de trois matrices où chaque élément de la matrice représente un pixel. A noter que les matrices U et V sont de dimensions plus petites que la matrice Y suivant le format utilisé. L'essentiel de l'information de l'image est stocké dans la matrice Y.

En effet, l'œil est plus sensible à la luminance que la chrominance. Plusieurs formats d'images existent et on les note selon les caractéristiques de leurs matrices YUV.

Chaque image est découpée en tranche (*slice*). Une tranche est constituée d'un ou plusieurs macroblocs (MBs) adjacents, ordonnés de gauche à droite. Ce sont des éléments importants pour la gestion des erreurs. Si une tranche contient une erreur, le décodeur passe alors immédiatement à la tranche suivante. Plus il y a de tranches dans une image, meilleur est le traitement des erreurs mais plus l'image prend de place également. Enfin, un MB est une matrice constituée de 4 blocs, chacun de taille 8 x 8 pixels. Un MB contient donc 16 x 16 pixels dans l'espace de luminance et 8 x 8 pixels dans l'espace de chrominance si on utilise le format d'image 4 : 2 : 0 (figure 8).

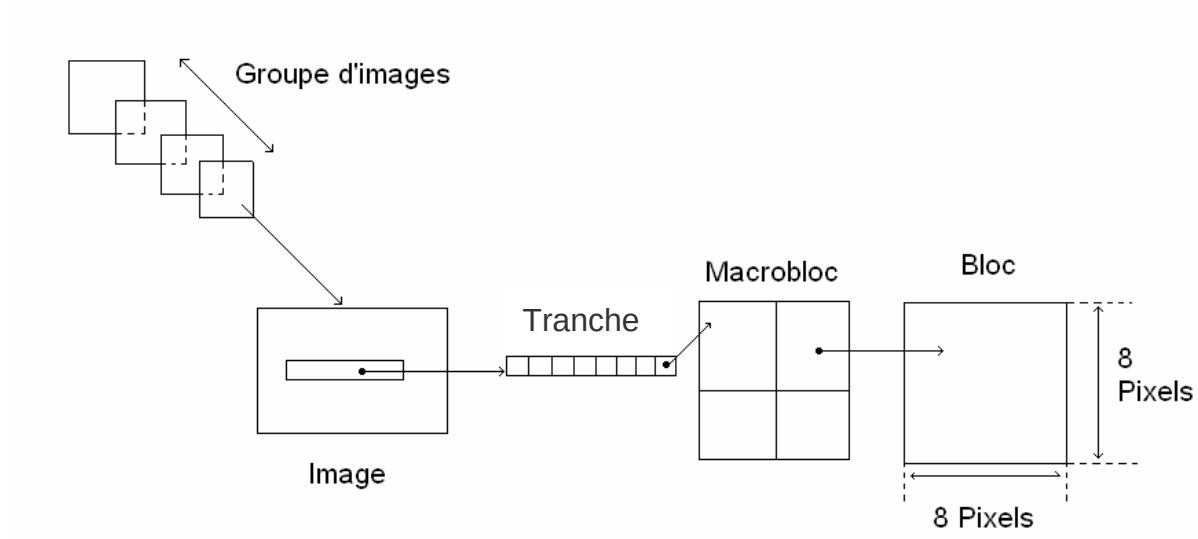


Figure 8. Hiérarchie des données dans le flux vidéo

I.3 Les normes de codage 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 (UIT-T H.261 [10], H.263 [11], ISO/CEI MPEG-1 [12], MPEG-2 [13] et MPEG-4 [14]). 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. La figure 9 représente la classification des normes de codage vidéo. D'après cette figure, on constate que la première norme de codage vidéo approuvée en 1990 est la norme H.261 par l'organisme UIT. Actuellement, les deux organismes travaillent à l'élaboration d'une norme commune à H.264 et MPEG4 [15].

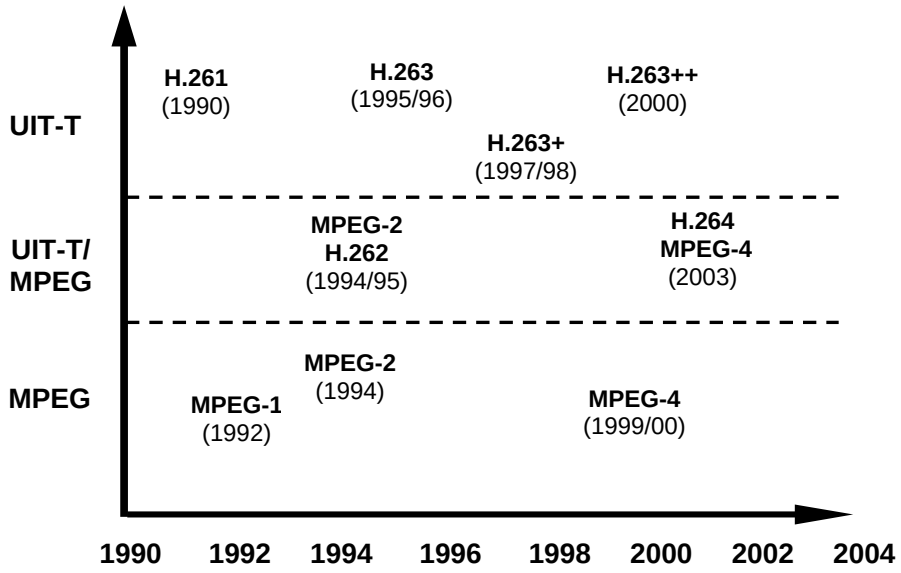


Figure 9. Les normes de codage vidéo

I.3.1 Principes :

Les codeurs normalisés exploitent les redondances présentées précédemment qui sont la redondance spatiale, temporelle et psycho-visuelle. Dans un codeur vidéo hybride à compensation du mouvement et à transformée, la prévision compensée du mouvement réduit en premier lieu les redondances temporelles. Un codage par transformée est ensuite appliqué à l'image différentielle correspondante pour réduire les redondances spatiales [7]. Dans le cas des sources fortement corrélées, comme les images naturelles, la capacité de compression de la Transformée en Cosinus Discrète, ou TCD (*Discrete Cosine Transform* : DCT), est très proche de celle de la transformée optimale. S'en suivent une quantification et un codage entropique permettant la compression des données ainsi obtenues.

Trois modes de codage sont possibles et conduisent à distinguer trois types d'image :

- Les images INTRA, dites de type I : Les images « INTRA » sont codées intégralement, sans aucune référence aux images voisines de la séquence vidéo. C'est la redondance spatiale qui est exploitée et éliminée à l'aide de la TCD. Cette opération est réalisée après analyse de l'image en trois plans (Y, Cr, Cb). Chacun de ces plans est décomposé en blocs de 8x8 pixels et transformés en matrices de coefficients fréquentiels classés selon l'amplitude des signaux. Ces coefficients sont ensuite quantifiés et codés en utilisant un codage entropique comme montre la figure 10. Les images I sont les plus coûteuses en débit mais servent de point de référence dans une séquence vidéo. Chaque changement de plan dans une séquence vidéo commence obligatoirement par une image de type I.

- Les images INTER prédites, dites de type P : Les images « Prédictives » exploitent à la fois la redondance spatiale et la redondance temporelle des images d'une séquence vidéo. Elles sont codées à partir de l'image « I » ou « P » précédente à l'aide de vecteurs de mouvement obtenu par estimation de mouvement. Les images sont découpées en blocs de 16 x 16 pixels. Les vecteurs de mouvement sont ensuite calculés en fonction du déplacement de chacun de ces blocs de pixels d'une image à la suivante puis codés avec un Codage à Longueur Variable (CLV).

- Les images INTER prédites bidirectionnelles, dites de type B : elles ne sont pas utilisées par tous les codeurs. Elles sont codées avec une estimation du mouvement par rapport à une image précédente et une image suivante. Elles ne servent jamais de référence. Les images de type I et P servent de référence aux images P et B, les images de type B ne servent jamais de référence. Le codage d'une image P et B est illustré par la figure 11.

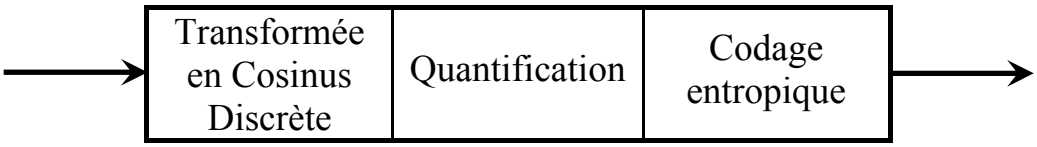


Figure 10. Codage d'une image I

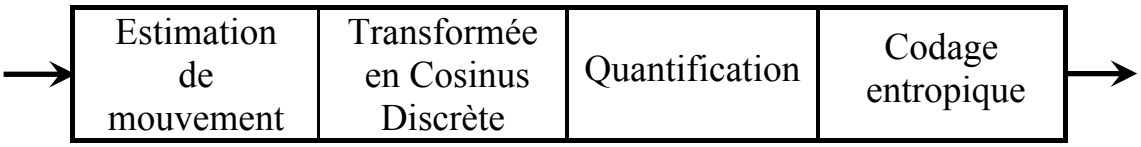


Figure 11. Codage d'une image P ou B

Le codage d'une séquence vidéo consiste en une succession d'images INTRA et INTER agencées de manière à satisfaire les contraintes de débit, de qualité et complexité (figure 12).

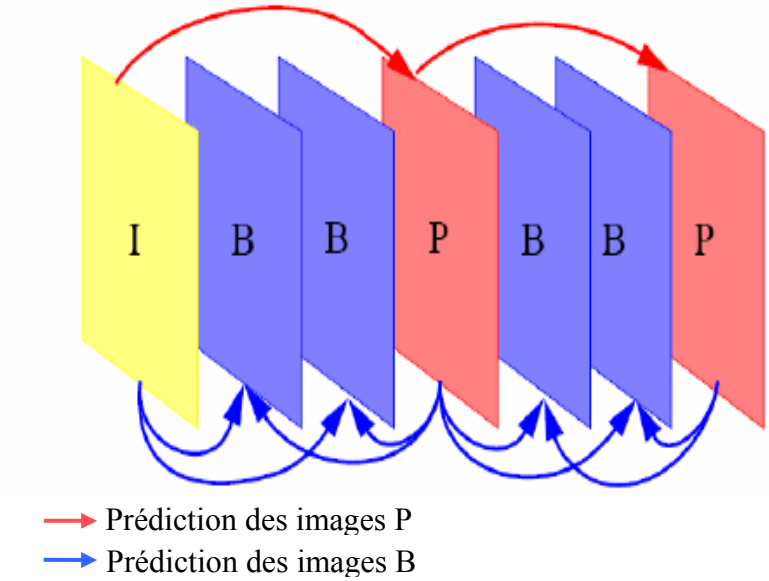


Figure 12. Exemple de configuration de codage d'une séquence vidéo

Le diagramme fonctionnel d'un codeur vidéo selon les principes exposés précédemment est présenté par la figure 13.

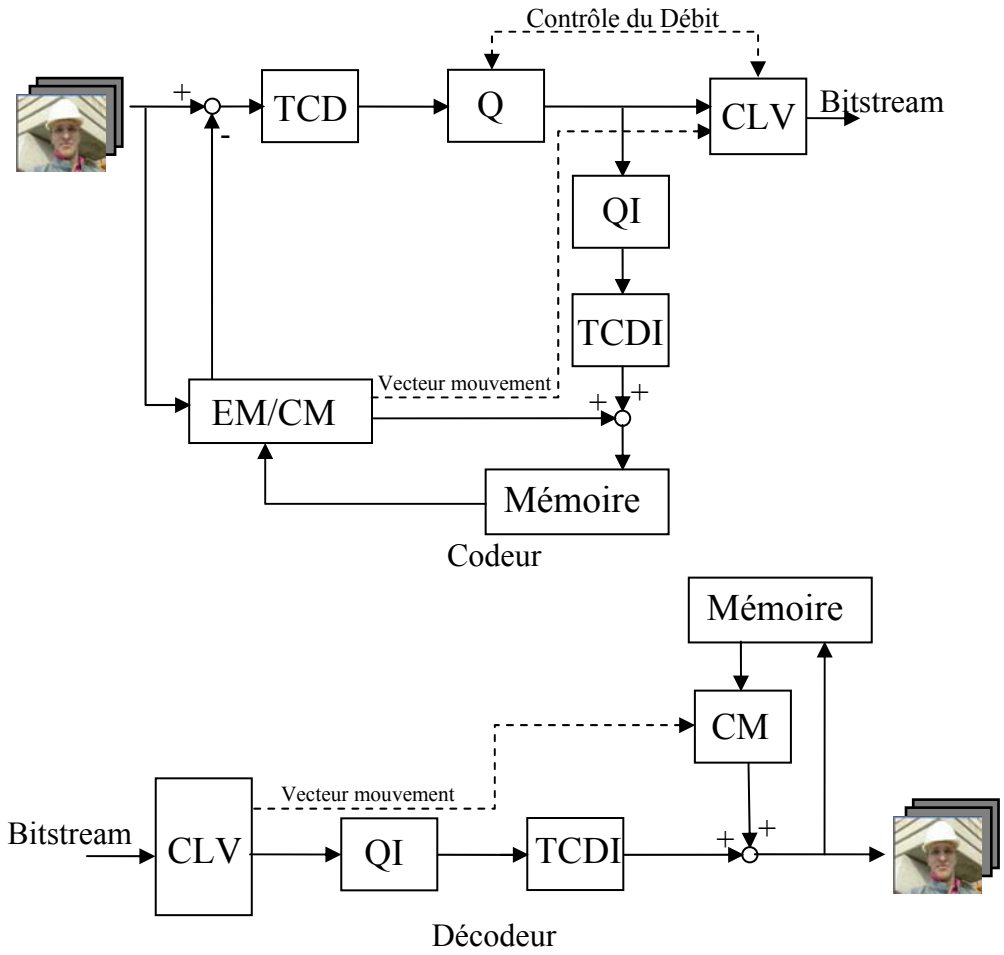


Figure 13. Diagramme fonctionnel d'un codec vidéo

I.3.2 Les normes de l'UIT-T :

I.3.2.1 H.261 :

La norme H.261 [10] est la première norme vidéo (réalisée en 1990). Elle a été développée pour les applications de visiophonie à des débits px64 kb/s ou p est compris entre 1 et 30 pour le réseau RNIS (Réseau Numérique à Intégration de Services).

Le format d'image traité est le QCIF (144x176 pixels) et optionnellement le CIF (288x352 pixels) et le Sub-QCIF (96x128). La fréquence image de base est 29.97 Hz mais peut être réduite en sautant 1, 2 ou 3 images entre chaque image transmise. Chaque MB subit une TCD de taille 8x8. Les coefficients ainsi obtenus sont alors quantifiés puis codés par un codage à longueur variable. La prédiction se fait INTER image et peut être améliorée par une compensation de mouvement et un filtrage. Les images peuvent être codées I ou P mais pas B.

La compensation de mouvement est spécifiée au niveau du décodeur comme suit : chaque bloc peut être affecté d'un vecteur de mouvement dont les dimensions horizontales et verticales ne peuvent excéder +/-15 pixels. Le vecteur de mouvement estimé sur la luminance pointe un bloc inclus dans l'image précédemment reconstruite. Les chrominances sont prédites par le même vecteur de mouvement dont les coordonnées ont été divisées par 2 et arrondies à l'entier le plus proche. Cette prédiction correspond à la construction d'une image P.

Le filtrage (optionnel) est un filtre spatial à deux dimensions travaillant au niveau pixel sur des blocs de taille 8x8 intervenants dans la boucle de codage.

L'arrangement du flux vidéo est structuré en 4 niveaux : image, groupe de blocs (GOB), macrobloc (MB) et bloc.

I.3.2.2 H.263 :

H.263 [11] est une norme de codage vidéo pour la communication vidéo à très bas débit dont la première version a été adoptée en 1995. Elle vise les applications de visiophonie et visioconférence sur RTC et RNIS. Cette norme repose sur les principes mis en place par la recommandation H.261.

Les formats supportés par cette norme sont le Sub-QCIF (96×128 pixels) et le QCIF (144x176 pixels) et optionnellement le CIF (288x352 pixels), le 4CIF (576x704 pixels) et le 16CIF (1152x1408 pixels). Les fréquences d'images restent inchangées par rapport à H.261.

Le décodeur doit avoir la possibilité d'effectuer des compensations de mouvement, le codeur pouvant exploiter ou non cette technique qui a évolué depuis H.261. En effet, la précision des vecteurs de mouvement n'est plus en pixel entier mais en demi-pixel, ce qui améliore grandement la qualité de la vidéo. L'utilisation d'image B est désormais possible.

En plus de cette configuration de base, six options de codage ont été ajoutées afin d'augmenter les performances du codage et ses fonctionnalités. La version 2 de la recommandation H.263 (1998), souvent appelée H.263+ [16] met en œuvre douze options supplémentaires et permet désormais de définir des formats et des fréquences d'image personnalisées. Les options ajoutées améliorent la qualité et la robustesse aux erreurs. La dernière version de H.263 (2000) appelé H.263++ [17] ajoute trois options. Outre l'amélioration en termes de qualité et de taux de compression, elle prend mieux en compte la transmission vidéo Temps Réel sur un réseau à qualité de service non garantie (perte de paquets…).

I.3.3 Les normes de l'ISO/MPEG :

I.3.3.1 MPEG-1 :

Le standard MPEG1 [12], standardisé en novembre 1992, a pour objectif le codage d'images en mouvement et du son qui leur est associé, en vue d'un stockage numérique selon un débit pouvant atteindre 1.5Mb/s.

Bien que les codeurs soient optimisés pour une résolution d'image SIF (352x240x29.97 progressif ou 352x240x25, en YUV 4 : 2 : 0) selon un débit de 1.5Mb/s, il est possible de modifier le débit ou la résolution. Les séquences MPEG-1 sont découpées selon une structure de couches emboîtées : une séquence est découpée en groupe d'une quinzaine d'images (GOP), chacune pouvant être encodée de façon INTRA (I), INTER (P) ou Bidirectionnelle (B). Chaque image est découpée en tranches permettant la synchronisation. Chaque tranche est découpée en MB, support de la compensation de mouvement en mode INTER. Chaque MB est découpé en blocs, sur lesquels est calculée la TCD en mode INTRA. La taille de ces différentes couches peut être spécifiée lors du codage dans la mesure où chaque couche possède son propre en-tête. Cela permet également de poursuivre le décodage en présence de tranches erronées.

I.3.3.2 MPEG-2 :

Le standard MPEG-2 [13] a été finalisé en novembre 1994 initialement élaboré pour la transmission de vidéo de qualité TV ayant un débit compris entre 4 et 9Mb/s. Le standard permet donc désormais la compression progressive ou entrelacée de vidéo selon des débits

allant de 1.5Mb/s à 100Mb/s, couvrant ainsi la TVHD, les médias de stockage interactifs (ISM), la TV par câble.

Les couches structurant les flux MPEG-2 sont similaires à celles définies par MPEG-1, aussi les améliorations se situent au niveau de la compensation de mouvement, de la gestion des coefficients TCD. Il est également possible de rajouter des blocs de chrominance lorsque le signal d'entrée est en 4 : 2 : 2. Pour la luminance, la taille des blocs peut varier de 8x8 à 1x1 et pour la compensation de mouvement, les MBs peuvent être décomposés selon des critères spatio-temporels, en sous macro blocs ayant chacun son propre vecteur de mouvement.

I.3.3.3 MPEG-4 :

Le standard MPEG-4 [14] est une norme générique de compression destinée à la manipulation d'objets multimédia. Elle permet le codage d'une grande variété de format vidéo (taille, résolution, fréquence image) mais aussi le codage d'objets vidéo de forme arbitraire, d'images fixes ainsi que d'objets synthétiques 3D. De ce fait, cette norme adresse une large gamme d'applications audiovisuelles allant de la visioconférence à la production audiovisuelle en passant par le streaming sur Internet.

Bien que les demandes pour MPEG-4 et H.263 soient différentes, certains domaines se recouvrent pourtant. Par exemple, les deux standards tentent de coder efficacement les vidéos naturelles pour des débits compris entre 24kb/s et 64kb/s. Ces recouvrements font que de nombreux participants à l'élaboration de MPEG-4 Vidéo, participent également aux efforts de codage à bas débit de l'UIT-T.

I.4 Etude de la norme H.263 :

I.4.1 Introduction :

La recommandation H.263 de l'UIT-T [11] définit le premier codeur/décodeur capable d'assurer une bonne qualité à faible débit. La norme H.263 possède 5 formats : sub-QCIF, QCIF, CIF, 4CIF et 16CIF. Le format du sous-échantillonnage utilisé dans cette norme est le format 4 : 2 : 0, c'est-à-dire les deux composantes de chrominance sont réduites à la moitié de la résolution verticale et horizontale de la composante luminance.

La table suivante nous donne les formats des images utilisées par la norme H.263.

Table 2. Formats des images

Format de l'image	Nombre de pixels pour la luminance (dx)	Nombre de pixels pour la luminance (dy)	Nombre de pixels pour la chrominance (dx/2)	Nombre de pixels pour la chrominance (dy/2)
sub-QCIF	128	96	64	48
QCIF	176	144	88	72
CIF	352	288	176	144
4CIF	704	576	352	288
16CIF	1408	1152	704	576

Dans la norme H.263, le flux vidéo est structuré en quatre niveaux : image, groupe de blocs (GOB), macrobloc (MB) et enfin bloc comme montre la figure 14 pour une image QCIF. D'après cette figure, on constate bien que chaque image est formée de groupes de blocs ou

GOB, chaque GOB est lui-même composé de blocs nommés MB (16 x 16 pixels) et chaque MB comprend à son tour quatre blocs de 8 x 8 pixels de luminance et deux de 8 x 8 pixels de chrominance.

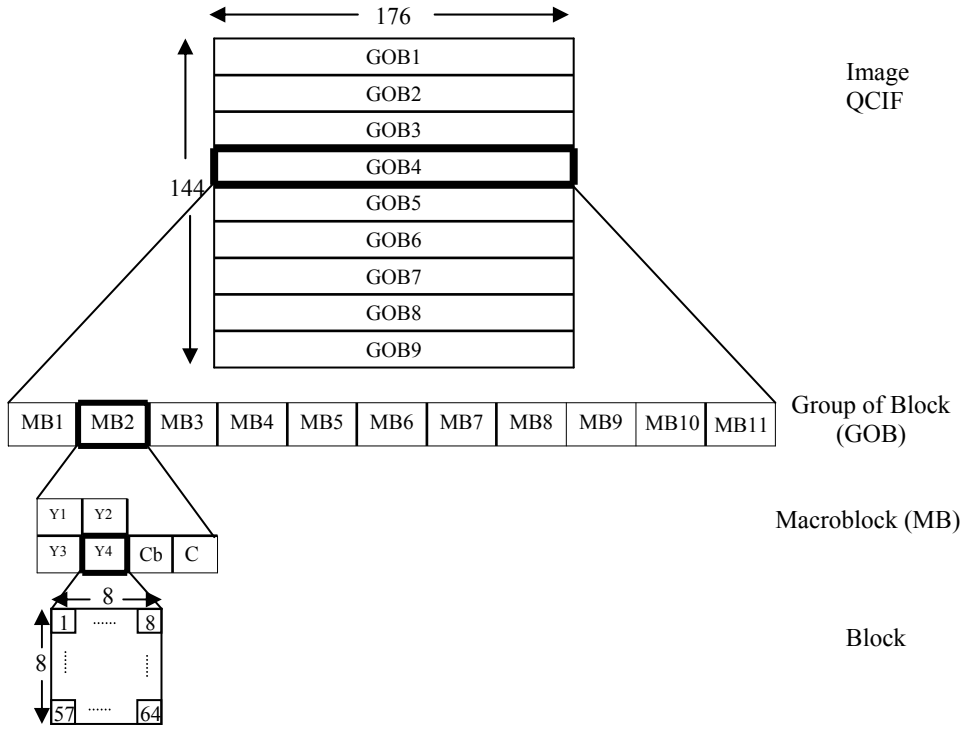


Figure 14. Structure d'une image dans la norme H.263

I.4.2 Le principe du processus de compression :

La norme H.263 utilise deux types de codage :

- Le codage INTRA : est utilisé pour la première image de la séquence vidéo et à chaque changement de plan. Il n'exploite que les redondances spatiales.
- le codage INTER : se sert des redondances temporelles. Il est employé pour les séquences d'images similaires contenant des objets en mouvement.

Le codage d'une image de référence I ou codage INTRA est identique à la norme JPEG : transformation TCD de chaque bloc 8x8 pixels suivie d'une quantification et enfin un codage entropique. L'image est sauvegardée sous la forme de MBs pour être exploités par le codage INTER. Ce dernier consiste à suivre le mouvement et à déplacer des blocs. Dans ce codage, on se base sur une estimation de mouvement pour comparer les MBs de l'image courante avec ceux de l'image précédente. On choisit celui qui s'en approche le plus et on calcule un vecteur de translation correspondant au déplacement. La différence entre les valeurs des MBs courants et prédits constitue des blocs d'erreurs. Ces derniers sont à leur tour divisés en blocs de 8 x 8 pixels pour subir un traitement de type JPEG identique au codage INTRA. La figure 15 est l'illustration de ce processus.

Les coefficients de quantification sortis du codage INTRA ou INTER, l'information permettant de distinguer les codages INTRA et INTER, le vecteur de mouvement pour le codage INTER et la taille du pas du coefficient de quantification sont alors multiplexés. Une méthode de compression basée sur l'entropie est appliquée à ce flot d'informations qui est ensuite transmis à une mémoire tampon.

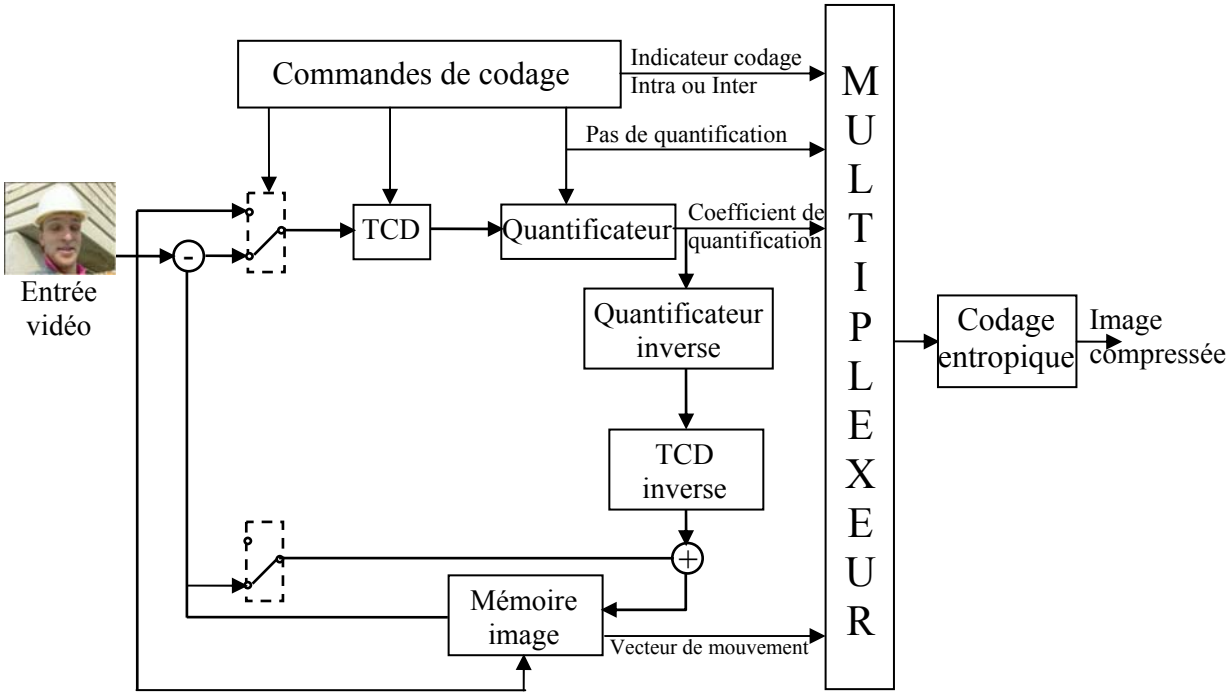


Figure 15. Schéma fonctionnel d'un codeur vidéo de la norme H.263

Dans les prochains paragraphes, nous décrirons les éléments constitutifs de ce codeur vidéo.

I.4.3 Estimation de mouvement :

Dans les codages par estimation de mouvement, la technique d'estimation de mouvement utilisée influe significativement sur le gain en compression. Cette estimation correspond à la prédiction des pixels de l'image courante à partir des pixels de l'image précédente et/ou suivante. Elle exploite les redondances temporelles présentes au sein d'une séquence vidéo. La plupart des techniques d'estimation de mouvement reposent sur les principes suivants [7] :

- La luminosité est uniforme le long des déplacements. Les variations dans le temps de la luminosité sont négligeables. Elles correspondent à un changement visuel mais pas à un réel déplacement.
- Les problèmes d'occlusion sont négligés. En effet, ces problèmes se produisent lorsqu'une partie de l'arrière plan caché apparaît. Aucune zone de l'image de référence ne lui correspond normalement, ce qui est rarement le cas dans une séquence vidéo réelle.

I.4.3.1 Prédictions avant et arrière (*forward and backward*) :

L'estimateur de mouvement permet d'estimer le mouvement des blocs entre deux images consécutives. Le mouvement calculé va permettre de prédire l'image *n+1* à partir des pixels de l'image *n* et les vecteurs mouvements. Deux types de prédiction sont possibles : la prédiction avant *forward* et la prédiction arrière *backward*.

La prédiction *forward* consiste à trouver les blocs de l'image *n* dans l'image *n+1*. Comme la montre figure 16, la mise en correspondance des blocs avec prédiction *forward* répond à la question : où va le bloc courant ?

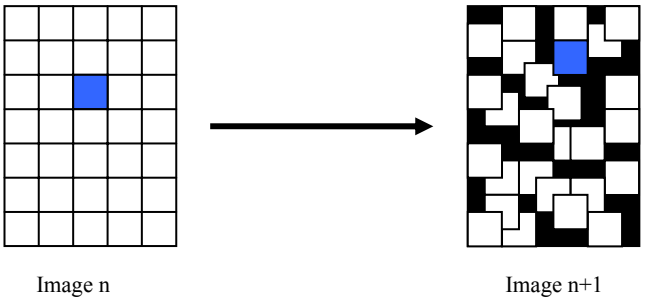


Figure 16. Prédiction *forward*

En revanche, la prédiction *backward* consiste à trouver les blocs de l'image *n+1* dans l'image *n*. Dans ce cas la question est : d'où vient le bloc courant ?

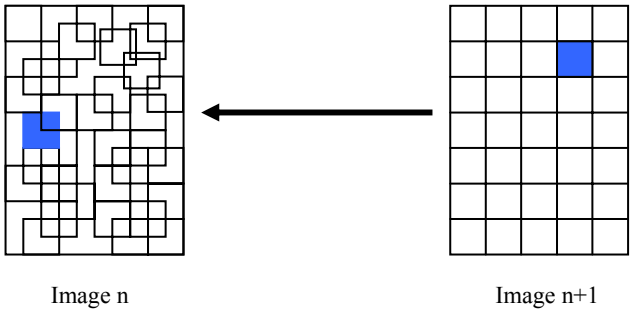


Figure 17. Prédiction *backward*

Les deux méthodes donnent des résultats différents. En effet, l'image prédite par prédiction *forward* présente des trous puisque les blocs n'ont pas un mouvement unique pour toute l'image. Lors de la prédiction, certains blocs se recouvrent et laissent apparaître des zones non prédites et par conséquent, noires (figure 16). Ce problème est entièrement résolu par l'utilisation de la prédiction *backward* **(**figure 17) qui ne laisse entrevoir aucun trou dans la prédiction.

I.4.3.2 Différentes méthodes d'estimation de mouvement (EM) :

La norme H.263 ne spécifie pas la technique d'estimation de mouvement qui peut donc être choisie par le concepteur [18]. De nombreux algorithmes d'estimation de mouvement sont développés peuvent être regroupés en trois grandes catégories:

- Les méthodes basées sur l'équation des flux optiques [19] : Ces méthodes effectuent une analyse des gradients spatiaux-temporels de la luminance Y des pixels. Elles reposent sur l'hypothèse que les changements de luminosité en chaque point de l'image sont uniquement dus à un déplacement. Soit un pixel x(x1, x2). A l'instant t, l'équation des flux optiques s'écrit :

$$dY = \frac{\partial Y(x,t)}{\partial x_1} \cdot dx_1(t) + \frac{\partial Y(x,t)}{\partial x_2} \cdot dx_2(t) + \frac{\partial Y(x,t)}{\partial t} \cdot dt$$

- Les méthodes pixel-récursives [20] : le but dans ce cas est de converger vers un minimum de la différence entre l'image courante et une image compensée en mouvement. Les estimations de déplacement sont faites de façon récursive au niveau pixel et sont du type prédiction-correction de la forme :

$$v(x,t+\Delta t) = v_{prev}(x,t+\Delta t) + u_{prev}(x,t+\Delta t)$$

$$avec \begin{cases} v(x,t+\Delta t) \ est \ l'estimation \ du \ déplacement \\ v_{prev}(x,t+\Delta t) \ est \ la \ prédiction \ du \ déplacement \\ u(x,t+\Delta t) \ est \ la \ correction \ de \ déplacement \\ \Delta t \ est \ l'indice \ temporel \ de \ l'image \ de \ référence \end{cases}$$

L'estimateur de mouvement du pixel x à l'instant t est ainsi réactualisé jusqu'à convergence. L'algorithme pixel-récursif vise à minimiser la différence entre l'image compensée en mouvement et l'image courante. Cette méthode conduit à un champ dense de vecteurs mouvements souvent incompatibles avec la contrainte de compression.

- Les méthodes par correspondance de blocs (BMA : *Block Matching Algorithm*) : Elles sont largement utilisées dans les codeurs vidéo normalisés du fait de leur meilleur compromis entre complexité et efficacité de codage, et de leur plus grande facilité d'implantation [21]. C'est pourquoi, nous considérons par la suite uniquement ces méthodes.

I.4.3.3 Méthodes par correspondance de blocs (*Block Matching*) :

Le *Block Matching* est certainement la méthode d'estimation de mouvement la plus utilisée vue sa faible complexité algorithmique. Cette méthode est généralement utilisée en estimation *backward*.

Le principe général de la méthode par correspondance de blocs est d'exploiter les redondances temporelles existant entre des images consécutives. Elle consiste à associer à chaque bloc de l'image *n* un bloc d'une image *n+1.* Chaque image est subdivisée en blocs de taille 16x16 qui est fixée par la norme. Chaque bloc est par la suite considéré comme étant un objet indépendant. Nous faisons l'hypothèse que le mouvement des pixels est uniforme à l'intérieur d'un bloc. L'algorithme consiste, pour un bloc de l'image courante, à choisir un bloc dans l'image de référence et à calculer un critère de comparaison entre ces deux blocs. L'opération est réitérée en choisissant un autre bloc jusqu'à ce que tous les blocs d'une zone déterminée de l'image de référence (appelée « fenêtre de recherche ») soient testés ou jusqu'à un critère d'arrêt arbitraire soit vérifié. Le bloc le plus semblable est ainsi identifié dans l'image de référence (figure 18).

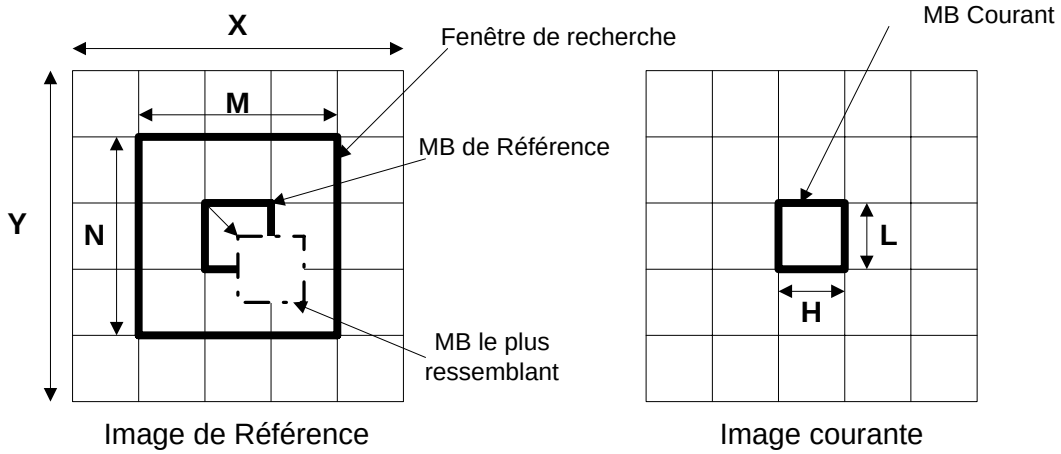


Figure 18. Recherche d'un vecteur de mouvement

Avec :
X, Y : largeur et hauteur de l'image à encoder.
M, N : dimensions de la fenêtre de recherche. Ici M=N= 2p+1 où p est le déplacement maximum.
L, H : dimensions d'un MB.

Nous obtenons de cette manière un vecteur de mouvement associé à chaque bloc. L'utilisation d'une fenêtre de recherche permet de limiter le nombre de blocs de référence que l'algorithme doit tester. La taille de la fenêtre dépend d'un déplacement maximal autorisé fixé par l'utilisateur. En effet, un petit déplacement pénalise les grands déplacements et augmente naturellement l'erreur de prédiction dans le cadre d'une vidéo dynamique. Cependant, peu de blocs seront testés et l'algorithme sera alors rapide. A l'inverse, un grand déplacement autorisé permet de donner plus de libertés aux vecteurs mouvements mais augmente de manière quadratique le nombre de blocs testés. I1 y a donc un compromis entre qualité et rapidité de l'algorithme. On pourrait alors considérer la taille de la fenêtre de recherche comme un point de paramétrage permettant de choisir un compromis entre qualité de l'estimation et rapidité.

Initialement, on peut placer l'origine d'un bloc en son centre. Cependant, il est beaucoup plus astucieux de le situer dans le coin supérieur gauche (figure 19) pour deux raisons : premièrement, les blocs ont en général des côtés de taille paire, ce qui implique qu'il n'y a pas de pixel central. Deuxièmement, les tests d'appartenance à la fenêtre sont moins coûteux en temps de calcul avec cette convention [9].

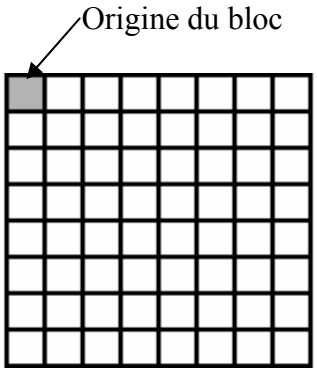


Figure 19. Illustration d'un bloc de taille 8x8

Le *Block Matching* vise à minimiser un certain critère de distorsion (erreur quadratique, erreur absolue…). Le choix de l'opérateur de distorsion n'est pas imposé par les normes des comités de standardisation (MPEG et UIT-T) mais la somme des valeurs absolues des différences (SAD : *Sum of Absolute Difference*) sur la luminance des pixels est utilisée le plus souvent [18]. L'équation ci-dessous correspond au calcul d'une distorsion pour un MB de 16x16 pixels.

$$SAD(x,y) = \sum_{j=0}^{15}\sum_{i=0}^{15} \left| Ycur(i,j) - Yprev(x+i, y+j) \right| \qquad \textbf{Eq : 1}$$

Avec :
Ycur : la luminance du macrobloc courant.
Yprev : la luminance du macrobloc de référence.
(x,y) : les coordonnées du macrobloc de référence dans la fenêtre de recherche.

I.4.4 La Transformée de Cosinus Discrète (TCD/TCDI) :

L'idée de base est de trouver un mode de description d'image où chaque paramètre amène vraiment une information nouvelle afin d'obtenir un code moins redondant. Cette étape n'apporte aucune compression. La transformée TCD est de même nature qu'une transformée de Fourier (FFT : *Fast Fourier Transform*). La TCD prend un ensemble de points du domaine spatial et le transforme en une représentation identique du domaine des fréquences. Nous définissons également la fonction inverse nommée TCDI (Transformée de Cosinus Discrète Inverse) qui traduit la représentation fréquentielle en représentation spatiale.

Les équations 2 et 3 représentent la définition mathématique de la TCD et de son inverse (TCDI) [18] [22].

$$y_{k,l} = \frac{c(k)c(l)}{4} \sum_{n=0}^{7} \sum_{m=0}^{7} x_{n,m} \cos\left(\frac{(2n+1)k\pi}{16}\right) \cos\left(\frac{(2m+1)l\pi}{16}\right) \qquad \textbf{Eq : 2}$$

$$x_{n,m} = \frac{1}{4} \sum_{k=0}^{7} \sum_{l=0}^{7} c(k)c(l)y_{k,l} \cos\left(\frac{(2k+1)n\pi}{16}\right) \cos\left(\frac{(2l+1)m\pi}{16}\right) \qquad \textbf{Eq : 3}$$

$$avec \quad c(\alpha) = \begin{cases} \frac{1}{\sqrt{2}} & pour\ \alpha = 0 \\ 1 & pour\ \alpha \neq 0 \end{cases}$$

Le but de la TCD utilisée dans la norme H.263 est la décorrélation des blocs 8x8 de pixels originaux ou de pixels différentiels à mouvement compensé et d'en compresser l'énergie dans le moins de coefficients possible. L'algorithme le plus commun pour mettre en application la TCD bidimensionnelle par blocs 8x8 est la TCD sur 8 points pour chaque ligne, suivie de la TCD sur 8 points pour chaque colonne.

La figure 20 montre comment une TCD bidimensionnelle est calculée en multipliant chaque pixel du bloc d'entrée par des termes qui représentent des ondes en cosinus échantillonnées de différentes fréquences spatiales. Un coefficient TCD donné est obtenu quand on ajoute le résultat de la multiplication de chaque pixel dans le bloc. La TCD éclate une zone d'image en fréquences à deux dimensions. La plus basse d'entre elles est placée dans le coin supérieur gauche. Les fréquences horizontales croissent vers la droite et les fréquences verticales croissent vers le bas.

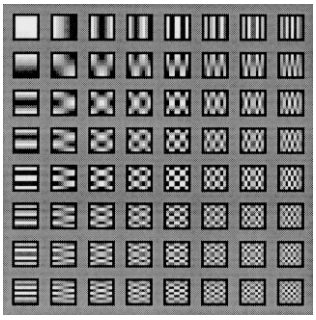


Figure 20. Images de la base de l'espace TCD

La TCD par bloc 8x8 donne un coefficient DC et 63 coefficients AC. Le coefficient DC est la moyenne des échantillons transformés et représente les détails les plus bruts du bloc d'image (plus basse fréquence spatiale). Les coefficients AC représentent les détails les plus fins de l'image (fréquences spatiales plus élevées).

Ci-dessous, on donne un exemple de matrice de pixels avec sa transformée.

72	108	110	103	104	106	116	115
72	113	115	101	92	96	102	103
71	104	97	108	127	133	134	131
78	119	108	97	109	118	119	120
72	109	104	105	101	108	107	112
64	97	100	105	104	105	102	105
67	110	111	114	114	117	118	123
69	113	101	98	99	109	112	119

Matrice de pixels d'entrée

836	-73	-27	-42	-43	-33	-19	-12
4	4	3	1	-3	3	0	6
-6	5	2	-10	-2	2	0	1
-21	14	-8	-9	4	6	3	0
-1	7	13	0	-4	-3	0	0
26	-17	-2	6	0	-2	-1	2
1	-19	-3	13	6	0	-1	0
17	-17	-4	4	4	1	-1	-2

Matrice de pixels de sortie

Les composantes trouvées en position (0,0) transportent une information plus utile que les composantes hautes fréquences. Lorsqu'on s'éloigne des composantes continues de l'image, on constate non seulement que les coefficients ont tendance à avoir de faibles valeurs mais aussi, qu'ils deviennent moins importants pour la description de l'image. Ce qui signifie qu'en effectuant la fonction TCD sur une matrice de pixels, nous concentrons la représentation de l'image dans les coefficients situés en haut et à gauche de la matrice de sortie.

Cette étape ne compresse donc pas les données. On n'a que des pertes de précision dues aux erreurs d'arrondis lors des calculs. On considère la transformation TCD comme une étape conservatrice qui prépare les données à la phase non conservatrice du processus, la quantification [23].

I.4.5 Quantification :

Après avoir défini la TCD comme transformation à utiliser, le comité UIT a entrepris un travail difficile : comment éliminer les portions d'image non significatives ? C'est ici qu'intervient la compression non conservatrice. La quantification (Q) n'est que le processus de réduction du nombre de bits nécessaires au stockage d'une valeur entière par la diminution de la précision de la qualité.

L'œil humain est plus sensible aux erreurs de reconstitution liées aux basses fréquences spatiales qu'à celles liées aux hautes fréquences. Les changements linéaires lents de l'intensité ou de la couleur sont importants pour l'œil. Les changements rapides dans les hautes fréquences peuvent être imperceptibles, c'est pourquoi on peut les éliminer. Pour chaque coefficient dans la matrice TCD, il faut calculer une valeur quantifiée correspondante en divisant chaque coefficient TCD par un pas de quantification. La formule de quantification pour chaque coefficient est la suivante [18] :

$$C^{q}_{m,n} = \frac{C_{m,n}}{Q_{m,n}}, \qquad 0 \le m,n \le 7 \qquad \textbf{Eq : 4}$$

Avec $C_{m,n}$ coefficient de la TCD, $Q_{m,n}$ valeur du quantum pour la coordonnée (m, n). Le résultat obtenu est arrondi à l'entier le plus proche. Dans la norme H.263, la quantification est

exécutée en utilisant le même pas de quantification dans un MB (c'est à dire en utilisant une matrice uniforme de quantification). Le pas de quantification (QP) choisi doit être dans l'intervalle de 1 à 31, seulement pour le premier coefficient (coefficient DC) d'un bloc INTRA, qui a un pas de quantification égal à huit.

Le paramètre de quantification est le principal moyen par lequel on peut contrôler le degré de compression et de réduction correspondante de la fidélité de l'information vidéo compressée. L'effet net est habituellement une variance réduite entre les coefficients quantifiés par rapport à la variance entre les coefficients TCD originaux de même qu'une réduction du nombre de coefficients non nuls, ce qui améliore l'efficacité du codage entropique, sujet dont nous allons maintenant traiter.

I.4.6 Codage entropique :

Le concept du codage entropique se fonde sur la théorie de l'information développée par Shannon. La stratégie consiste à exploiter l'analogie entre la probabilité d'un événement et la quantité d'information à transmettre. Le codage entropique permet d'encoder différents symboles (luminance, chrominance…) en leur attribuant un mot binaire. La longueur de ce mot dépend de la fréquence d'apparition du symbole considéré. La réalisation d'un code attribuant aux symboles les plus fréquents les mots les plus courts a pour conséquence de réduire le débit du signal.

Avant l'exécution du codage entropique, les coefficients TCD quantifiés sont arrangés en un ensemble unidimensionnel par balayage en zigzag (figure 21). Cet arrangement place le coefficient DC en premier dans l'ensemble, et les coefficients AC restants sont classés de la basse fréquence à la haute fréquence. Cette méthode de balayage produit une représentation compressée des coefficients TCD du fait qu'un grand nombre de coefficients sont quantifiés à zéro.

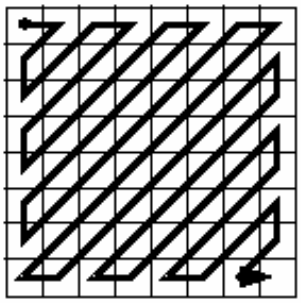


Figure 21. Le parcours zigzag

La norme H.263 spécifie deux méthodes de codage entropique : codage de Huffman [24] et le codage arithmétique [25]. La norme H.263 *baseline* utilise le codage de Huffman, par contre, le codage arithmétique est optionnelle [11] [18].

Le codage de Huffman spécifie l'utilisation des tables de codage au cours de la compression et de la décompression. Ces tables doivent être indiquées dans le codec (codeur/décodeur). Par contre, le codage arithmétique n'impose pas l'utilisation des tables. Les tables de Huffman peuvent être spécifiées pour une image donnée ou utilisées par défaut pour toutes les images, et ceci suivant l'implémentation utilisée. Des codes de longueur variable ou CLV (VLC : *Variable Length Code*) sont souvent utilisés avec des codes de Huffman.

Le codage arithmétique donne une compression de 5 à 10 % meilleure que le codage de Huffman [23]. Mais il est un peu plus lent et compliqué. Ceci peut être expliqué par le fait que le codage arithmétique est une méthode de codage par nombre en virgule flottante. En effet, il

permet de remplacer une séquence de symboles par un nombre en virgule flottante appartenant à l'intervalle [0, 1].

I.4.6.1 Codage de Huffman :

Le codage de Huffman est un procédé dans lequel la longueur des symboles codés varie dans le sens inverse de leur probabilité d'apparition (c'est à dire que les symboles de forte probabilité auront un code court alors que les symboles de faible probabilité auront un code long). Le modèle fournit au processus de codage la probabilité ou un compteur de fréquences pour chaque symbole rencontré dans le flot d'entrée. L'algorithme crée des codes de longueurs variables sur un nombre entier de bits en fonction des probabilités fournies par le modèle.

La technique de Huffman est basée sur la construction d'un arbre binaire de décodage. Cet arbre est élaboré de façon ascendante en commençant par les feuilles de l'arbre et en remontant vers la racine. Par exemple, on considère le texte ayant la table de fréquence suivante.

Table 3. Table de fréquence

Symbole	A	B	C	D	E
Fréquence	25	7	6	6	5

La figure 22 illustre la construction de l'arbre de Huffman pour notre exemple. D'après cette figure on constate que, les symboles sont disposés individuellement sous forme d'une chaîne de nœuds. Ces nœuds sont les feuilles de l'arbre de décodage. Ils forment la liste des nœuds libres au début du processus et ils seront connectés par un arbre binaire. Chaque nœud a un poids, la fréquence ou la probabilité de l'apparition des symboles.

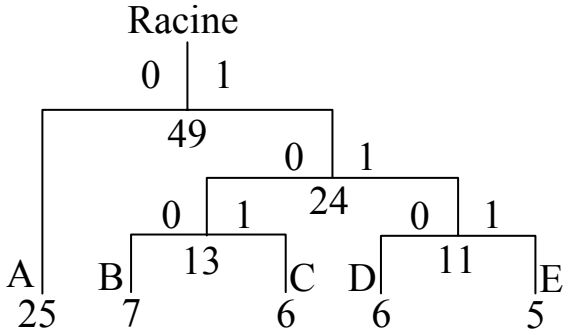


Figure 22. L'arbre de Huffman

L'arbre est créé d'après les étapes suivantes :

- Les deux nœuds libres de plus faible poids sont sélectionnés. Un nœud parent de ces deux nœuds est créé. Nous lui affectons comme poids, la somme des poids de ses deux fils.
- Le nœud parent est ajouté à la liste des nœuds libres tandis que ses deux fils en sont retirés.
- Un des fils est désigné comme le chemin pris à partir du nœud parent pour décoder un bit 0, l'autre est pris pour décoder un bit 1.
- Nous répétons les étapes précédentes jusqu'à ce qu'il ne reste plus qu'un seul nœud parent libre. Le nœud restant est la racine de l'arbre.

Après avoir déterminé l'arbre, nous procédons au codage de chaque symbole du flot d'entrée. Pour obtenir le code d'un symbole donné, il suffit de parcourir l'arbre de Huffman à

partir du nœud feuille jusqu'à la racine en inversant l'ordre des bits. La table 4 représente la table de codification de notre exemple.

Table 4. Table de codification

Symbole	A	B	C	D	E
Code	0	100	101	110	111

I.4.6.2 Codage arithmétique :

Le codage arithmétique se singularise par sa capacité à coder chaque symbole sur un nombre non entier de bits. En réalité, il n'assigne pas un mot de code à chaque symbole mais il associe un point de l'intervalle [0,1] à un ensemble de symboles [26]. Le principe repose sur le découpage de l'intervalle [0,1]. Chaque symbole se voit attribuer une partition de l'intervalle dont la taille est égale à sa probabilité d'occurrence. L'ordre de rangement est mémorisé pour être utilisé lors du décodage.

Le codage arithmétique est généralement plus performant que le codeur d'Huffman. Il tend vers la limite inférieure théorique mais cependant il est gourmand en ressources et nécessite de connaître *à priori* l'intégralité du signal avant de pouvoir procéder au codage [23].

Les différentes étapes de l'algorithme de codage sont :

- L'initialisation : nous affectons à chaque symbole une plage d'intervalle basée sur sa probabilité d'apparition fournie par le modèle. Les bornes externes de cet intervalle sont zéro et un.
- Le traitement du message : nous initialisons un intervalle de travail en prenant comme bornes zéro et un. Le premier symbole est représenté par la plage qui lui est affectée à l'étape 1. Chaque symbole suivant restreint davantage l'intervalle et il est représenté par sa plage relative dans la plage précédente. Ainsi le flot de données est traduit par un nombre contenu dans la dernière plage calculée.
- On rajoute un symbole spécial pour déterminer la fin du message où l'on donne la longueur du flot avec le message codé pour permettre au décodeur de déterminer la fin du message.

Par exemple, supposons qu'on veut coder une partie "acaab" d'une longue séquence avec une probabilité d'apparition indiquée dans le Table 5.

Table 5. Probabilités des symboles

Symbole	**Probabilité**	**Intervalle**
a	0.7	[0, 0.7]
b	0.1	[0.7, 0.8]
c	0.2	[0.8, 1.0]

L'intervalle initial [0, 1] va être divisé en trois sous-intervalles suivant les probabilités des symboles de la séquence. Ce qui donne les sous-intervalles suivants : [0, 0.7], [0.7, 0.8] et [0.8, 1.0].

Dans cette exemple, le 1^{èr} symbole est "a", l'étiquette appartient donc à l'intervalle [0, 0.7]. Après que le 1^{èr} symbole soit codé, les limites inférieures et supérieures de l'intervalle sont respectivement 0 et 0.7 pour le symbole suivant. L'intervalle [0, 0.7] va être divisé en trois

sous-intervalles : [0, 0.49], [0.49, 0.56] et [0.56, 0.7] correspondant respectivement aux symboles "a", "b" et "c".

Le 2ème symbole est "c", de probabilité 0.2. Ainsi, le nouveau sous-intervalle sera [0.56, 0.7]. Ce dernier va être divisé en trois sous-intervalles : [0.56, 0.658], [0.658, 0.672] et [0.672, 0.7].

Le 3ème symbole est "a". Le nouveau sous-intervalle sera en conséquence [0.56, 0.658]. Le sous-intervalle [0.56, 0.658] va être partagé à son tour en trois sous-intervalles : [0.56, 0.6286], [0.6286, 0.6286] et [0.6286, 0.658].

Le 4ème symbole est "a", donc le nouvel intervalle est [0.56, 0.6286].

Et ainsi de suite jusqu'à que l'on code la totalité de la séquence. On aura à la fin, le sous-intervalle suivant [0.60802, 0.61488]. Un nombre contenu dans l'intervalle final comme 0.60803 code sans ambiguïté le message "acaab". La figure 23 est une représentation de ce processus.

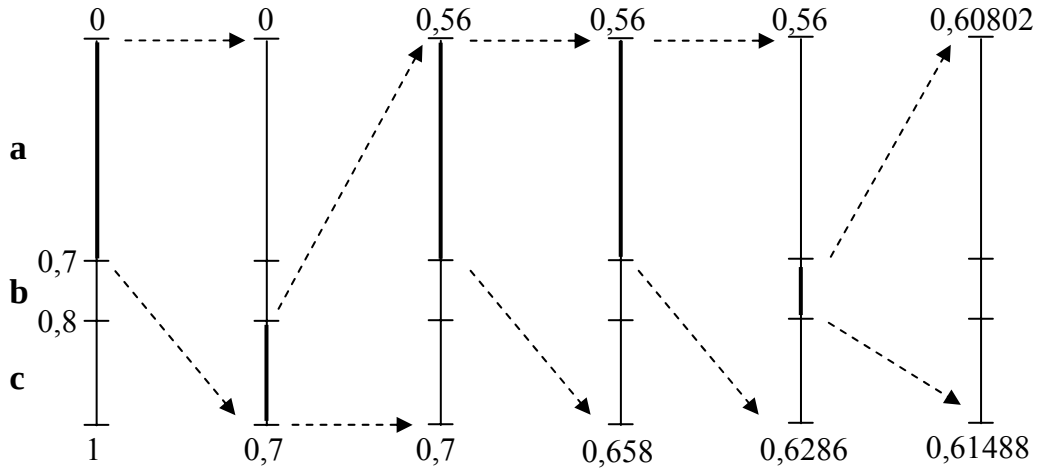


Figure 23. Génération de l'étiquette pour la séquence "acaab"

I.5 Conclusion :

Dans ce chapitre, nous avons présenté les différents principes de codage vidéo ainsi que les techniques utilisées. De plus, nous avons expliqué l'importance de l'utilisation des redondances spatiales et temporelles dans les codeurs vidéo pour le codage INTRA et INTER. La majorité des codeurs et décodeurs sont conformes aux règles de normalisation qui reposent sur la mise en œuvre du mode INTRA basée sur la TCD et du mode INTER qui utilise la compensation et l'estimation de mouvement. Puis, nous avons présenté la structure et les éléments principaux du codeur de la norme H.263 de l'organisme UIT-T.

Dans ce qui suit, nous abordons le concept de système numérique embarqué. Nous prendrons l'exemple des systèmes électroniques mobiles de troisième génération. Dans ce cadre, nous aborderons l'étude de la norme H.263. Nous définirons une méthodologie de conception logicielle/matérielle autour des analyses architecturales.

Chapitre II: La conception des systèmes numériques

II.1 Introduction :

Les avancées actuelles dans la technologie des semi-conducteurs et des méthodologies de conception permettent le développement de systèmes numériques complexes sur puce « SoC », des dispositifs pouvant contenir des millions de transistors. Les derniers circuits FPGA (*Field Programmable Gate Array*) permettent également le développement de systèmes complets. Ainsi, un système qui était auparavant implanté sur une carte, peut dorénavant être conçu sur une puce unique offrant l'avantage d'être compact et de supporter un très grand nombre de traitements arithmétiques. 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 [27]. Ces composants peuvent être aussi bien des cœurs de processeurs, des cœurs de DSP, des accélérateurs matériels... La réalisation de ces systèmes a nécessité la mise en place d'une méthodologie de conception logicielle/matérielle (*codesign*) prenant en compte les contraintes de l'embarqué.

II.2 Méthodologies de Conception des systèmes numériques :

II.2.1 Généralités :

Dans l'approche traditionnelle, un système numérique est un assemblage sur une carte de différents composants discrets représentant chacun une fonction particulière plus ou moins complexe telle qu'additionneur, mémoire, composant d'interface, gestionnaire d'interruption, processeur... Si une erreur de conception était faite, il était au minimum nécessaire d'ajouter des fils entre les composants, ou au pire, de refaire une carte pour régler le problème, c'est-à-dire reprendre complètement son routage. Plus le système numérique est complexe, plus ces composants sont nombreux, plus la carte est chère, et plus les perturbations électromagnétiques sont importantes. Un besoin existait donc de pouvoir modifier la logique sans modifier les cartes et aussi de diminuer le nombre de composants sur une carte numérique. En effet, moins il y a de composants remplissant un même cahier des charges, moins la carte est chère, et plus les fonctions sont intégrées, plus il est possible de proposer une carte moins encombrante. Les améliorations des processus de fabrication des composants électroniques ont permis de répondre de mieux en mieux à ces besoins.

Dans ce contexte, l'*International Technology Roadmap for Semiconductors* [28] affirme que les processeurs contiennent en moyenne près de 100 millions de transistors en 2007 et en prévoit près de 1.5 milliard pour 2013. L'évolution des technologies de fabrication de circuits permettent l'intégration d'un système numérique sur un même composant : c'est le concept du *single chip*.
Ceci est en fait lié à la loi empirique de Moore qui stipule que pour une surface de silicium donnée, on double le nombre de transistors intégrés tous les 18 mois [29]. La loi de Moore a radicalement changé la façon de concevoir les systèmes numériques aujourd'hui puisque l'on peut procéder à l'implantation d'algorithmes complexes pour les systèmes numériques de futures générations. On travaille maintenant au niveau système (ou fonctionnalité) et non au niveau porte logique. Cette évolution de la conception peut être résumée sur la figure 24.

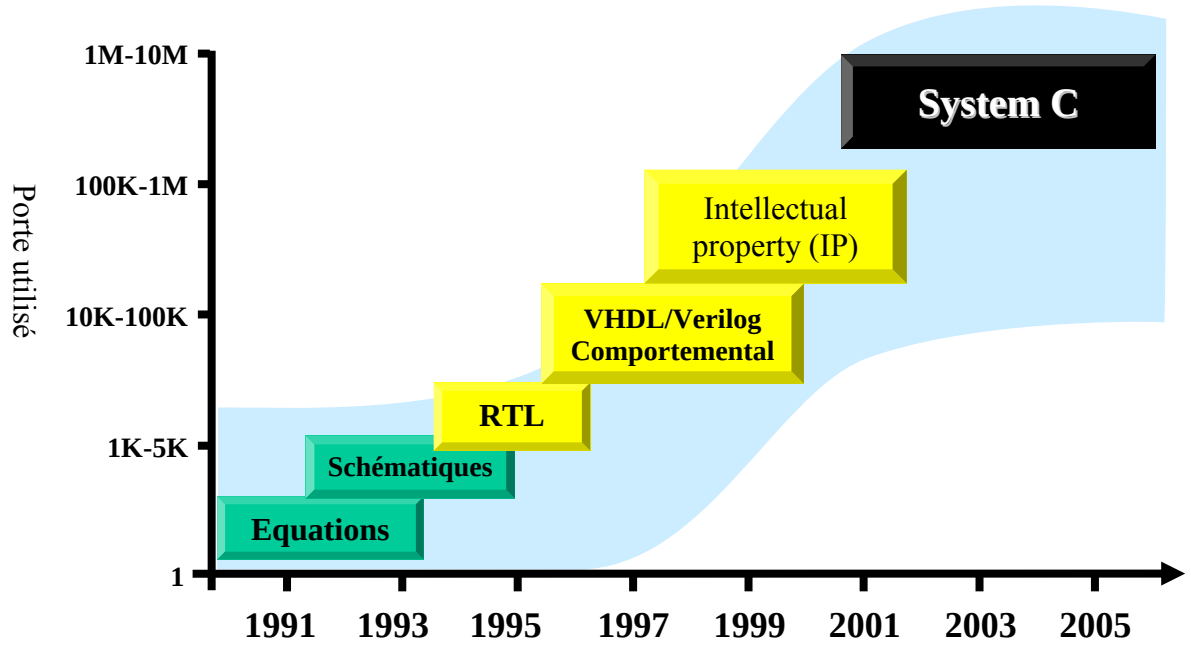


Figure 24. Evolution de la conception numérique

L'approche « schématique » au niveau portes logiques et fonctions de base RTL (*Register Transfer Logic*) semble aujourd'hui délaissée pour la conception des systèmes complexes au profit d'une approche « textuelle ». L'approche schématique reste cependant toujours valable et est plutôt réservée à la conception des petits systèmes...

L'approche textuelle, on utilise des langages de description de matériel comme VHDL (*Very high speed integrated circuit Hardware Description Language*) ou Verilog pour synthétiser une fonction numérique. Ces langages de description de matériel sont en fait des langages de programmation qui 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 (par exemple *Precision* de Mentor Graphics). Les fabricants de FPGA proposent maintenant leur propre synthétiseur.

Dans le développement des systèmes numériques complexes, il existe des besoins qui reviennent fréquemment. Certaines sociétés ont développé ou rassemblé des modules répondant à ces besoins et les mettent sur le marché sous le nom de blocs IP (*Intellectual Property*) (fonctions mathématiques : FFT, DCT, FIR, interfaces bus : PCI, RapidIO, coupleurs divers : UART, HDLC…). Ces modules IP peuvent être achetés ou téléchargés librement sur Internet. On peut ainsi voir la conception d'un système numérique complexe comme un assemblage de modules IP [30].

Les langages de description de matériel sont aussi intéressants pour la facilité de modification et de réutilisation d'un design précédent pour un nouveau design : c'est le *design reuse*.

II.2.2 Réalisation d'un système sur puce SoC (*System on Chip*) ou SoPC (*System on Programmable Chip*) :

Un SoC est un ensemble de blocs fonctionnels intégrés dans un composant électronique avec au moins un processeur comme élément de traitement. La figure 25 représente un SoC qui est basé sur des cœurs de processeurs. A partir de cette figure, on constate que seuls les

blocs de traitements particulièrement coûteux sont réalisés en matériel, l'essentiel de l'application étant implanté en logiciel et exécuté par les cœurs de processeurs [31]. Ceci conduit à une plus grande flexibilité sur les choix algorithmiques et permet une mise à jour moins contraignante des applications embarquées.

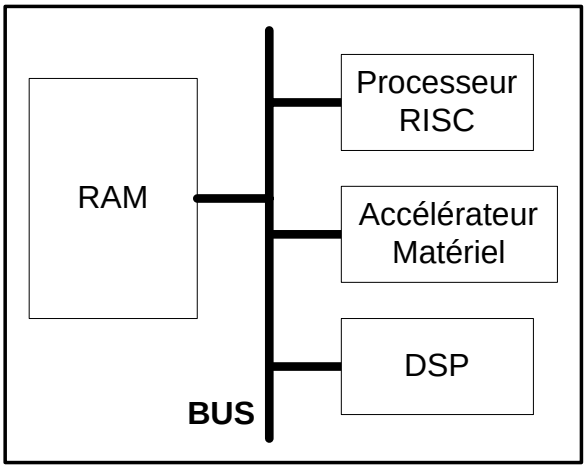


Figure 25. SoC basé cœurs de processeurs

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 SoC peut être retenu pour les applications destinées au grand public. Il permet des meilleures performances en termes de consommation, de vitesse et de surface. Mais, la fabrication et le test sont des étapes longues et coûteuses [32] [33]. De plus, un SoC est figé et n'est donc pas réutilisable pour une autre application. Par contre, le SoPC est un composant reconfigurable à volonté. Il permet donc un développement et prototypage rapide du système. Mais, en contrepartie, on peut avoir une consommation d'énergie plus grande avec une performance plus faible que celle du SoC [34].

Vue l'importante croissance de la complexité des SoCs/SoPCs, particulièrement dans des domaines tels que le traitement de signal intensif, il devient de plus en plus nécessaire de réutiliser dans leur conception des composants déjà élaborés. Les blocs IP sont généralement catalogués et échangés sur le réseau Internet en respectant les standards VSIA (*Virtual Socket Interface Alliance*) [35] [36], cet organisme les a renommés composants virtuels.

II.2.3 Les différentes familles de blocs IP (*Intellectual Property*) :

Un bloc ou un composant IP est un composant virtuel qui peut apparaître sous différentes formes [37] [38] :

- IP Logiciel *softcore* : le composant est livré sous sa forme HDL (*Hardware Design Language*) synthétisable, c'est à dire flexible. 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 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, le bloc IP est ciblé sur une technologie particulière et le travail d'optimisation est garanti. Cela englobe la netlist entière, le routage et l'optimisation pour une librairie technologique spécifique, 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 le *softcore* et le *hardcore*, plus flexible que le *hardcore*, plus prédictif en termes de performance et de surface que le *softcore*. 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).

II.3 Les circuits à logique programmable :

Actuellement, on trouve différentes familles de circuits programmables tels que les CPLDs (*Complex Logic Programmable Device*) et les FPGAs. La différence entre ces deux types de 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.
Par contre, ceux-ci sont des composants VLSI constitués de blocs mémoires vives, entièrement reconfigurables [39]. Ces blocs sont structurés en LUT (*Look Up Table*), flip-flop, RAM et l'ensemble dispose d'un vaste système d'interconnexions. Le progrès de la conception des circuits électroniques permet d'avoir des composants toujours plus rapides et à plus haute densité d'intégration, ce qui permet de programmer des applications importantes comme par exemple les applications vidéo. À l'heure actuelle, on compte une dizaine de fabricants, le marché étant nettement dominé par les sociétés Altera [40] et Xilinx [41].

II.3.1 Types d'architectures et éléments des circuits FPGA :

Classiquement pour les architectures des circuits FPGA, on peut rencontrer trois topologies différentes :

- Architecture de type îlots de calcul : Dès le départ, Xilinx a choisi ce type d'architecture. Cette architecture FPGA est constituée d'une matrice plane d'éléments. Ces éléments constituent les ressources logiques et de routages programmables du FPGA.

- Architecture de type hiérarchique : Dans cette architecture, il existe plusieurs plans dans le FPGA. Mais, ces plans ne sont pas physiques, ils correspondent aux niveaux de hiérarchie logique. C'est à dire qu'un élément d'un niveau logique peut contenir des éléments de niveau logique inférieur, d'où la notion de hiérarchie. Chaque niveau logique reprend la topologie d'une architecture du type îlots de calcul avec un routage dédié pour chaque niveau. Cette architecture se trouve dans les FPGAs d'Altera.

II.3.2 Les différents éléments d'un circuit FPGA :

Les éléments constitutifs d'un FPGA sont toujours à peu près les mêmes quelle que soit l'architecture choisie. Chaque fabricant ayant ses variantes par rapport à un autre. Nous pouvons citer un certain nombre de ces éléments :

a) Les éléments logiques :

Ce sont les blocs de base de tout circuit FPGA. On peut réaliser dans ces blocs toutes les opérations de logique combinatoire (dans la limite d'un nombre d'entrées). Ces blocs ont souvent la même constitution et cela malgré la différence de fabricants et d'architectures. La structure la plus courante est celle de la figure 26 qui présente la structure de base. Ces structures sont généralement constitués d'une ou plusieurs tables LUT (*Look Up Table*) qui contiennent, après configuration, la table de vérité de la fonction logique qu'elles doivent réaliser ou alors un ensemble de valeurs qui sont mémorisées comme dans une mémoire ROM. La taille des tables LUT est généralement de 16x1 (c'est à dire qu'elles disposent de 4

entrées). Les tables LUT sont suivies d'un registre de sortie, ce qui permet de synchroniser, si nécessaire, la sortie sur une horloge. La plupart des blocs logiques de bases sont munis d'une chaîne de propagation rapide de retenue afin de former de petits additionneurs rapides.

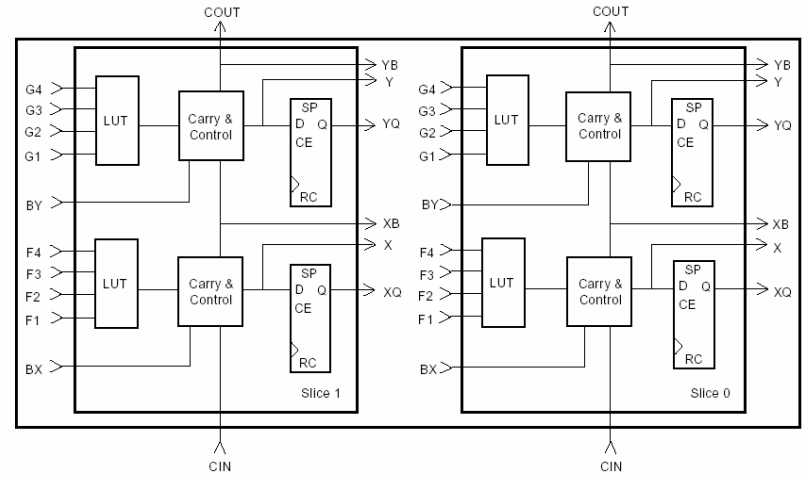


Figure 26. Structure de base d'un composant programmable

b) Les éléments de mémorisation :

Actuellement, Les FPGAs sont utilisés pour des applications plus importantes qui demandent souvent des capacités de stockage (par exemple les applications du traitement vidéo). La nécessité d'intégrer des blocs de mémoire directement dans l'architecture des FPGAs est vite devenue capitale. De cette façon, les temps d'accès à la mémoire sont diminués puisqu'il n'est plus nécessaire de communiquer avec des éléments extérieurs au circuit.

c) Les éléments de routage :

Les éléments de routages sont les composants les plus importants dans les FPGAs. En fait, ces éléments représentent la plus grosse partie du silicium consommée sur la puce du circuit. Ces ressources sont composées de segments (de longueurs différentes) qui permettent de relier entre eux les autres éléments via des matrices de connexions. Le routage de ces ressources est un point critique du développement d'une application sur un FPGA. Ces éléments sont très importants puisqu'ils vont déterminer la vitesse et la densité logique du système. Par exemple, les matrices de routage sont physiquement réalisées grâce à des transistors de cellules SRAMs, qui ont une résistance et une capacité, ce qui entraîne l'existence de constantes de temps.

d) Les éléments d'entrées/sorties :

Le but des éléments d'entrées/sorties est de relier un circuit avec son environnement extérieur. Ceux-ci peuvent bénéficier de protections, de buffer ou d'autres éléments permettant la gestion des entrées et des sorties. En particulier il est à noter que les circuits actuels proposent différentes normes pour les niveaux d'entrées et de sorties qui par configuration peuvent êtres choisies afin de s'adapter à l'environnement.

e) Les éléments de contrôle et d'acheminement des horloges :

L'horloge est un élément essentiel pour le bon fonctionnement d'un système électronique. Les circuits FPGA sont prévus pour recevoir une ou plusieurs horloges. Des entrées peuvent être spécialement réservées à ce type de signaux, ainsi que des ressources de routage spécialement adaptées au transport d'horloges sur de longues distances. Les circuits FPGA disposent des éléments d'asservissement des horloges (des PLLs ou des DLLs) afin d'avoir la même horloge dans tout le circuit (synchronisation des signaux). Ces éléments permettent de

créer à partir d'une horloge d'autres horloges à des fréquences multiples de la fréquence de l'horloge incidente.

II.3.3 Exemple de circuit FPGA : la famille Altera Stratix II :

Altera a lancé au début de l'année 2004 un nouveau composant le Stratix II [42] [43]. Ce composant est marqué par un certain nombre de changements par rapport aux architectures classiques des premiers FPGA Altera (Flex et Apex) à trois niveaux de hiérarchie. Le circuit Stratix II comme le circuit Stratix dont il a hérité de nombreuses caractéristiques, est moins hiérarchique et n'a plus que deux niveaux de hiérarchie. Le niveau le plus haut (figure 27) consiste en un ensemble d'éléments configurables LAB (*Logic Array Bloc*) qui sont répartis en matrice. A ce même niveau, des mémoires de différentes densités (512 bits *M512*, 4 Kbits *M4K* et 512 Kbits *MegaRAM*) sont réparties sur la matrice, ainsi que des blocs dits "blocs DSP" apparaissant sur la figure 28. Ces derniers intègrent des fonctions matérielles telles que multiplieurs, accumulateurs, additionneurs, multiplexeurs et registres et permettent, entre autre, de réaliser des multiplieurs 36 bits ou des opérateurs MAC de 18 bits. Au niveau inférieur, les LABs sont constitués de 8 ALM (*Adaptive Logic Module*) et d'un réseau de connexions locales. Les ALMs, schématisés à la figure 29, sont réalisés autour d'un bloc de logique combinatoire à 8 entrées, de deux additionneurs et des registres de sortie. Le bloc combinatoire est en fait réalisé avec deux LUTs à quatre entrées et de quatre LUTs à trois entrées.

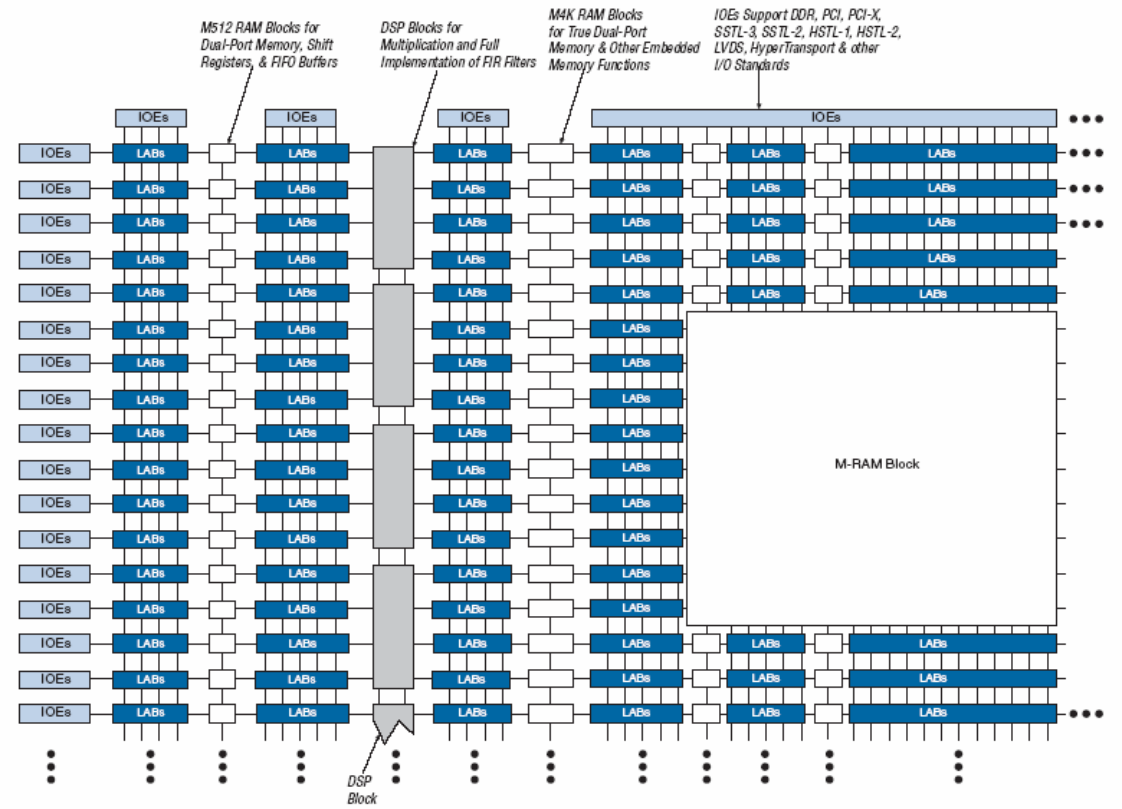


Figure 27. Niveau supérieur de la hiérarchie de l'architecture du circuit Stratix II

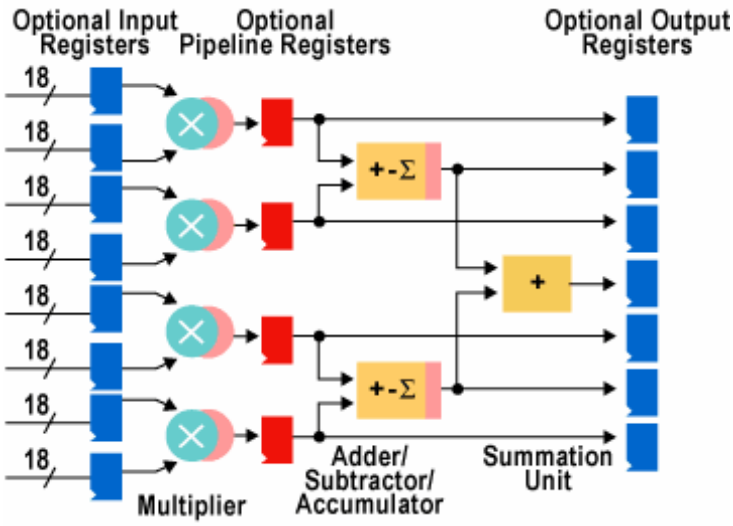


Figure 28. Architecture d'un bloc DSP

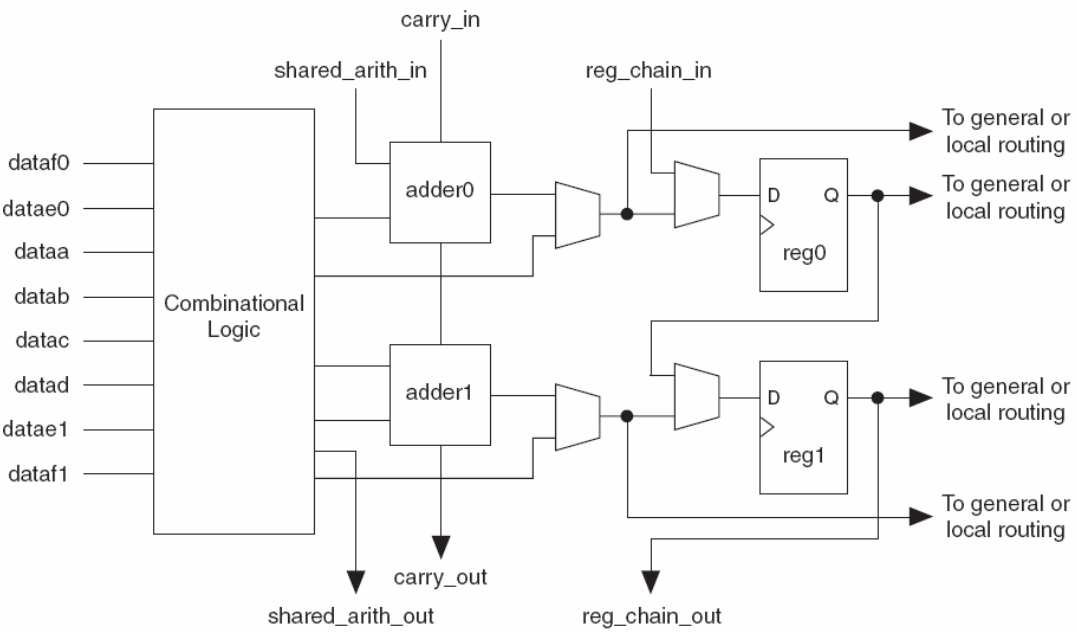


Figure 29. Architecture d'un ALM au niveau inférieur de la hiérarchie de l'architecture du circuit Stratix II

II.4 Architectures reconfigurables embarquées :

Le SoPC correspond à l'intégration d'un ou plusieurs cœurs de processeur et de ses périphériques sur une même puce programmable de type FPGA [44]. 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.

II.4.1 Architecture des processeurs :

L'architecture d'un processeur est un élément important qui conditionne directement les performances du processeur.

II.4.1.1 Architectures de Von Neumann et de Harvard :

Il existe principalement deux approches qui sont l'architecture de Von Neumann et de Harvard.

II.4.1.1.1 Architecture de Von Neumann :

Un processeur basé sur une structure Von Neuman stocke les programmes et les données dans le même espace mémoire [45] [46]. Une instruction contient le code opératoire et l'adresse de l'opérande. Cette architecture est caractérisée par un processeur et une mémoire reliés par un bus comme la montre la figure ci-dessous.

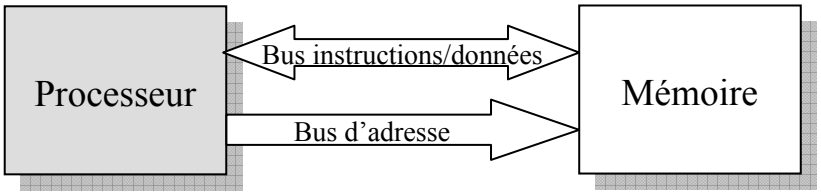


Figure 30. Architecture de Von Neumann

Les instructions et les données sont stockées dans la mémoire. Pour y accéder, un bus d'instructions et de données permet de les transférer de la mémoire au processeur. La désignation de l'instruction ou de la donnée désirée s'effectue par une adresse transmise par un bus. Cette adresse sélectionne l'instruction ou la donnée requise par le processeur.

II.4.1.1.2 Architecture de Harvard :

Cette structure se distingue de l'architecture de Von Neuman par le fait que les espaces mémoire programme et données sont séparés. L'accès à chacune des deux mémoires se fait via un chemin distinct (figure 31). Cette organisation permet de transférer une instruction et des données simultanément, ce qui améliore les performances.

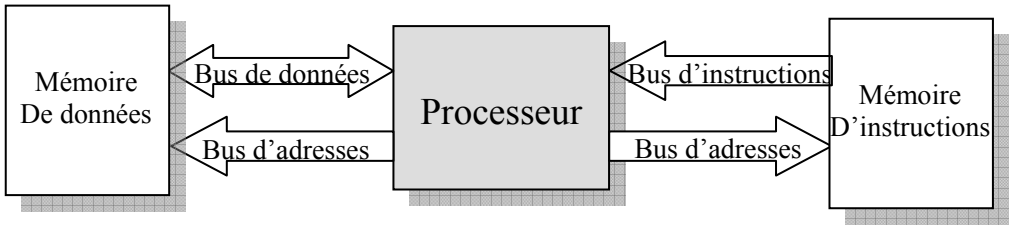


Figure 31. Architecture Harvard

L'architecture généralement utilisée par les processeurs est la structure Von Neuman. L'architecture Harvard est utilisée en particulier dans des processeurs spécialisés tel que le DSP (*Digital Signal Processor*) pour des applications Temps Réel comme les applications de traitement vidéo [47].

II.4.1.2 Les architectures CISC et RISC :

II.4.1.2.1 Architecture CISC:

L'architecture CISC (*Complex Instruction Set Computer*) est utilisée entre autres par tous les processeurs de type x86, c'est-à-dire les processeurs fabriqués par Intel, AMD... Les processeurs basés sur l'architecture CISC peuvent traiter des instructions complexes qui sont directement implantées sur le silicium de la puce afin de gagner en rapidité d'exécution sur ces commandes [48] [49].

Cette architecture se caractérise par :

- Un jeu d'instructions très riche et une grande variété de modes d'adressage.
- Un grand nombre d'instructions.
- Des instructions complexes et de longueur variable afin de répondre à la grande variété des instructions des langages de haut niveau.

II.4.1.2.2 Architecture RISC :

Si l'on se résout à se passer des instructions complexes, on pourra économiser la surface de silicium. C'est ce qui a été fait avec le processeur (*Reduced Instruction Set Computer*).

Toutefois, sur ce concept de processeur RISC viennent se greffer de nouvelles idées telles que :

- Les instructions doivent rester d'une longueur fixe identique.
- Les modes d'adressage doivent se simplifier via une architecture de registre à registre et des accès mémoire obtenus avec des instructions *Load* et *Store*.
- Il faut multiplier le nombre des registres internes dans le microprocesseur, des registres spécialisés mais aussi beaucoup de registres généraux pour limiter les aller et retour en mémoire.
- Le format des instructions doit être sur trois opérandes. Cela permet, en une seule instruction, de désigner à la fois la source et la cible.
- Il convient d'utiliser intensivement des mémoires cache pour accélérer les opérations.

II.4.1.2.3 Comparaison entre CISC et RISC :

Les processeurs RISC peuvent différer de leurs équivalents CISC de huit façons qui sont résumées dans la table ci-dessous.

Table 6. Caractéristique des processeurs RISC et CISC

	RISC	**CISC**
1	Instructions simples ne prenant qu'un cycle	Instructions complexes prenant plusieurs cycles
2	Seules les instructions *LOAD* et *STORE* font des accès mémoire	Toutes les instructions peuvent faire des accès à la mémoire
3	Traitement pipeline	Peu ou pas de traitement pipeline
4	Instructions exécutées par le matériel	Instructions interprétées par un microprogramme
5	Instructions au format fixe	Instruction en format variable
6	Peu d'instructions et de modes d'adressage	Beaucoup d'instructions et de mode d'adressage
7	Toute la complexité est dans le compilateur	Toute la complexité est dans le microprogramme
8	Plusieurs jeux de registres	Un seul jeu de registres

II.4.2 Les processeurs pour les SoPCs :

Le choix d'un processeur pour les SoPCs peut se faire sur différents critères [50] :

- **Processeur *hardcore* :** Certains circuits FPGAs contiennent un ou plusieurs processeurs de façon non partagée avec les ressources standards. Xilinx offre un

processeur PowerPC et Altera un processeur ARM. Dans ce cas, la fréquence d'utilisation est maximale et toute la ressource du circuit FPGA reste disponible.

- **Processeur propriétaires (*firmcore*) :** On ne peut pas l'utiliser dans un circuit FPGA autre que celui pour lequel il est prévu. On trouve principalement au niveau des processeurs propriétaires le processeur NIOS d'Altera [51] et le processeur Microblaze de Xilinx [52]. On s'intéressera essentiellement au processeur NIOS d'Altera par la suite.
- **Processeur *softcore* libre :** Il est écrit en langage de description matériel (VHDL/Verilog) dont le code source peut être librement distribué et implanté dans n'importe quel circuit FPGA. On est alors indépendant du type de circuit FPGA. On trouve principalement au niveau des processeurs *softcore* libres le processeur Leon [53], le processeur OpenRisc [54], le processeur F-CPU [55], le processeur LatticeMico32 [56]… Ce se sont généralement des processeurs 32 bits ayant une architecture de type Harvard avec un jeu d'instructions réduit RISC.
- **Autres :** Le site http://www.us.design-reuse.com/ recense une centaine de processeurs adaptés aux SoCs.

II.4.3 Le processeur embarqué NIOS :

Le processeur embarqué NIOS est un processeur à cœur logiciel de type *firmcore*, c'est à dire exclusivement dédié à la famille d'Altera. Le processeur NIOS peut être associé à une large gamme de périphériques, des instructions personnalisées et des accélérateurs pour créer un SoPC (figure 32). Le cœur logiciel de processeur embarqué NIOS est configurable et évolutif, pour permettre aux intégrateurs systèmes de disposer d'une solution SoPC souple et très robuste. Ce processeur peut être facilement combiné avec la logique d'utilisateur et être programmé dans un FPGA.

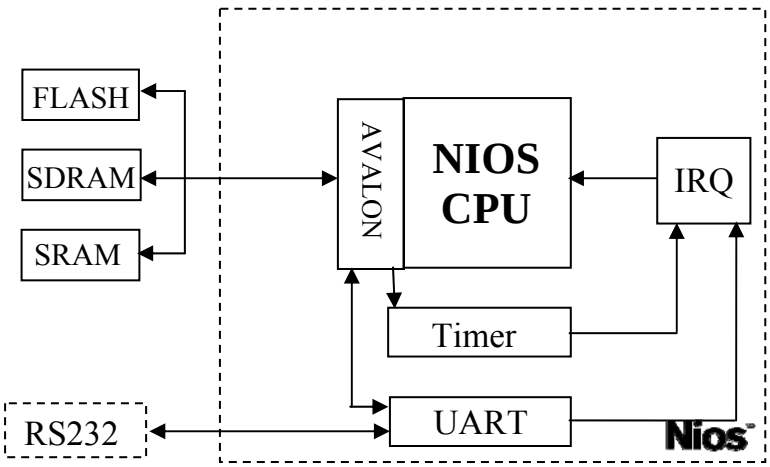


Figure 32. Système Altera NIOS

La figure 32 décrit le système NIOS. Il est constitué du processeur NIOS, du bus Avalon et des périphériques (contrôleur mémoire, UART, timer…). Le processeur NIOS est le cœur du système, il est connecté aux différents périphériques à travers le bus Avalon. Ce bus doit être configuré en maître/esclave. L'interface du bus Avalon est générée automatiquement par l'outil de génération d'Altera NIOS (*SOPC Builder*).

II.4.3.1 Processeur NIOS :

Le processeur NIOS (figure 33) est un processeur RISC entièrement synchrone, son architecture interne de type Harvard. Il possède au maximum 6 niveaux de pipeline cadencé à 50 MHz avec une largeur de bus de 32 bits. Ses performances sont de 30 à 80 MIPS (*Million Instructions per Second*). Il est possible d'accélérer certains traitements, en ajoutant des instructions personnelles (décrites en VHDL) au processeur NIOS. De cette manière, il est possible de réaliser de la surcharge d'operateurs ou simplement d'étendre les jeux d'instruction. D'après la figure 34, on voit bien qu'on peut ajouter à l'Unité Arithmétique et Logique (UAL) du processeur NIOS essentiellement deux types d'instruction : combinatoire (un seul cycle) ou séquentiel (multi cycle) [57].

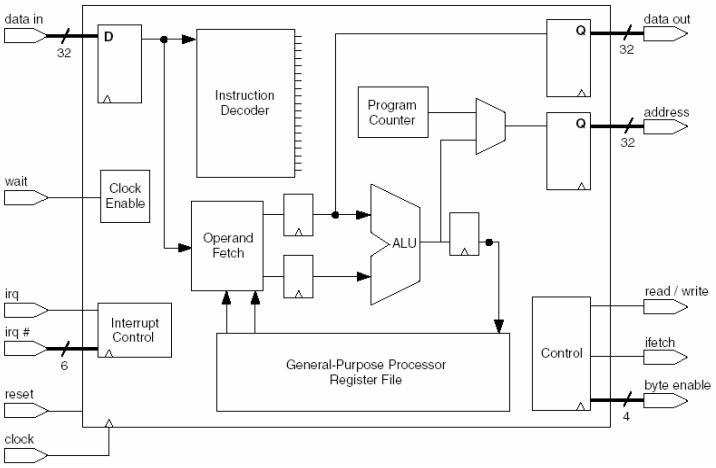


Figure 33. CPU NIOS

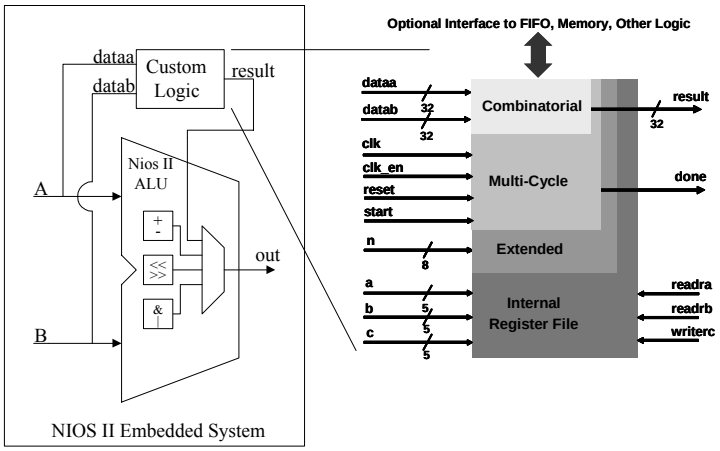


Figure 34. Instruction personnalisée du processeur NIOS II

Actuellement, Altera propose une deuxième version plus performante de NIOS : NIOS II [58]. La table 7 illustre la différence entre ces deux versions.

Table 7. Différence entre NIOS et NIOS II

NIOS	NIOS II
■ Architecture RISC pipeline ■ Instructions 16 bits ■ Liste de registres avec fenêtrage ■ Données sur 16 ou 32 bits ■ 64 niveaux d'interruption ■ Cache d'instructions et de données ■ Instructions personnalisées	■ Architecture RISC pipeline ■ Instructions 32 bits ■ Liste de registres fixe ■ Données sur 32 bits ■ 32 niveaux d'interruption ■ Cache d'instructions et de données ■ Instructions personnalisées

La société Altera propose trois versions pour le processeur NIOS II. La table 8 illustre ces trois versions. Une première version *Economy* qui utilise moins de surface, une deuxième version *Standard* qui permet un compromis entre surface et rapidité, une dernière version *Fast* qui est plus rapide que les deux autres.

Table 8. Les différentes versions du NIOS II

	NIOS II /f	**NIOS II /s**	**NIOS II /e**
Pipeline	6 niveaux	5 niveaux	Non
Multiplication Matériel	1 Cycle	3 Cycle	Par logiciel
Branch Prediction	Dynamic	Static	Non
Cache d'Instructions	Configurable	Configurable	Non
Cache de données	Configurable	Non	Non
Instructions Personnalisés	Supérieur à 256		

La figure 35 représente les performances en DMIPS *(Dhrystons Million Instructions per second*, unité issue du benchmark dit de Dhrystons) et la surface occupée des différentes versions du NIOS II sur différentes familles de FPGA d'Altera (Stratix II, Stratix, Cyclone). D'après cette figure, on constate que l'implantation du processeur NIOS II (version *Fast*, *Standard* et *Economy*) sur circuit FPGA Stratix II donne de meilleures performances (225 DMIPS@205 MHz, 133 DMIPS@180 MHz et 31 DMIPS @209 MHz respectivement) et une occupation de surface la plus faible (1319 ALUTs, 1029 ALUTs et 483 ALUTs respectivement) par rapport à un autre FPGA.

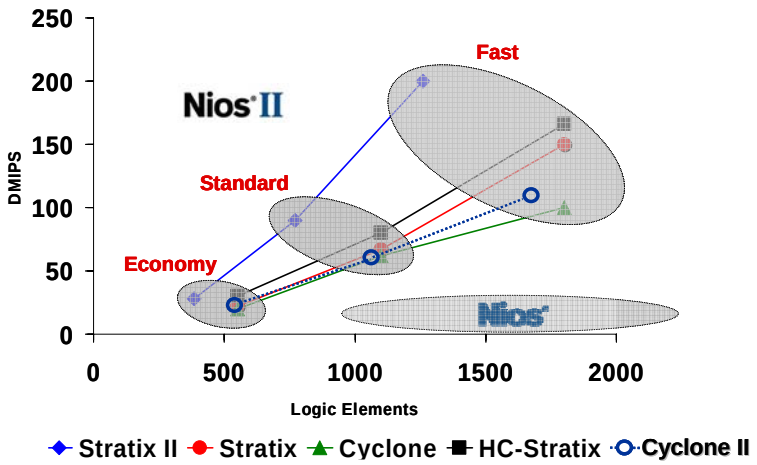


Figure 35. Implantation du processeur NIOS II sur différents circuits FPGA d'Altera

II.4.3.2 Bus Avalon :

Le bus Avalon peut être vu comme un ensemble de signaux prédéfinis permettant de connecter un ou plusieurs IP. La figure 36 présente le bus Avalon. Ce bus comprend un décodeur d'adresse, un multiplexeur de données, un générateur de cycles d'attente et un contrôleur d'interruption. Les utilisateurs peuvent facilement intégrer leurs propres périphériques avec le reste du système basé sur le processeur NIOS.

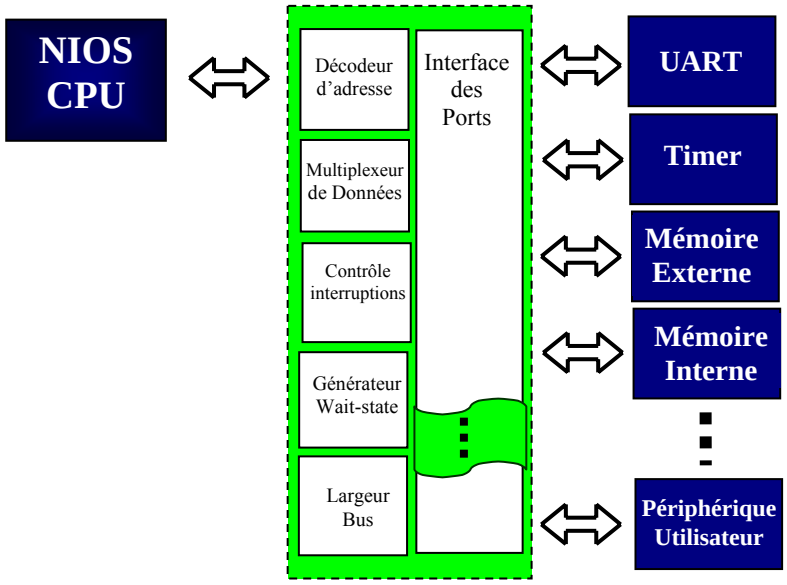


Figure 36. Bus Avalon

Le bus Avalon permet la connexion entre des composants maîtres ou esclaves. Il supporte plusieurs maîtres sur le bus. Un arbitrage est nécessaire au partage d'une même ressource partagée par les circuits maîtres. Cette architecture multi maître fournit la grande flexibilité dans la conception des systèmes.

Les figures 37 et 38 représentent un exemple de déroulement des cycles de lecture et d'écriture (respectivement) sur le bus Avalon du système.

Figure 37. Cycle de lecture

(A) : Le cycle de lecture commence par un front montant de *clk*.
(B) : Le port maître fournit les signaux *read_n* et *address*.
(C) : Le bus Avalon présente les données à lire *readdata* si le signal *wait_request* est à « 0 ».
(D) : Le port maître capture les données *readdata* sur le prochain front montant. Puis le transfert se termine et un autre cycle peut recommencer.

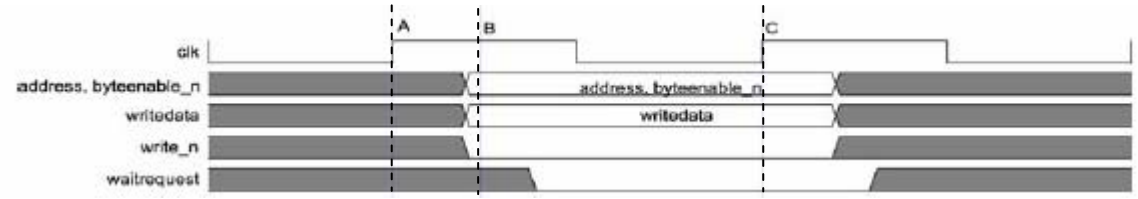


Figure 38. Cycle d'écriture

(A) : Le cycle d'écriture commence par un front montant de *clk*.
(B) : Le maître fournit les signaux *address*, *write_n* et *writedata*.
(C) : si le signal *wait_request* est à « 0 » sur le front montant de *clk*, alors le transfert se termine et un autre cycle de lecture ou écriture peut recommencer.

II.5 Les systèmes numériques embarqués :

Sans nous en rendre compte, nous utilisons actuellement tous de nombreux systèmes embarqués. En effet de nos jours, ces systèmes électroniques ont été introduits dans de nombreux domaines d'applications tels que l'automobile, l'avionique, les systèmes multimédia, les appareils électroménagers ou bien des terminaux de communication sans fil. Les concepteurs doivent donc gérer les développements de systèmes électroniques embarqués multidisciplinaires et multiprocesseurs. L'intégration électronique permettra à terme 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 de demain dit de troisième génération ont un marché potentiel dont le revenu mondial est estimé à 320 milliards de dollars US à l'horizon 2010 [59]. Ces terminaux visiophoniques représentent un bon exemple pour les systèmes multimédia embarqués d'où l'intérêt de présenter l'architecture de base ces terminaux.

II.5.1 Définition :

Un système embarqué peut être défini comme « *un système électronique et informatique autonome ne possédant pas des entrées/sorties standards comme un clavier ou un écran d'ordinateur (PC)* » [27].

II.5.2 Les contraintes des systèmes embarqués :

Les contraintes liées à l'électronique embarquée sont :

- **Les contraintes physiques :** il doit présenter un faible encombrement et un faible poids.
- **La dissipation de puissance :** la dissipation est directement reliée à l'autonomie du système embarqué.
- **Le temps de développement :** en plus des contraintes de temps de développement de l'architecture système, il faut prendre en compte le temps de développement des fonctions logicielles associées.
- **Le coût :** le système ne doit pas être cher tout en tenant compte de la performance.

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.

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.

II.5.3 Terminaux visiophoniques :

II.5.3.1 Généralités :

Le marché des terminaux mobiles est en train d'évoluer d'un environnement dominé par la voix à un environnement ou tous les média sont susceptibles de coexister. L'apparition des services Internet, d'échange de données, d'écoute de musique et de consultation de clip vidéo ne fait désormais plus aucun doute. L'UMTS [60] [61] est le réseau supportant cette évolution. Il permet de faire converger vers le monde de la diffusion, celui d'Internet et celui du monde mobile (figure 39).

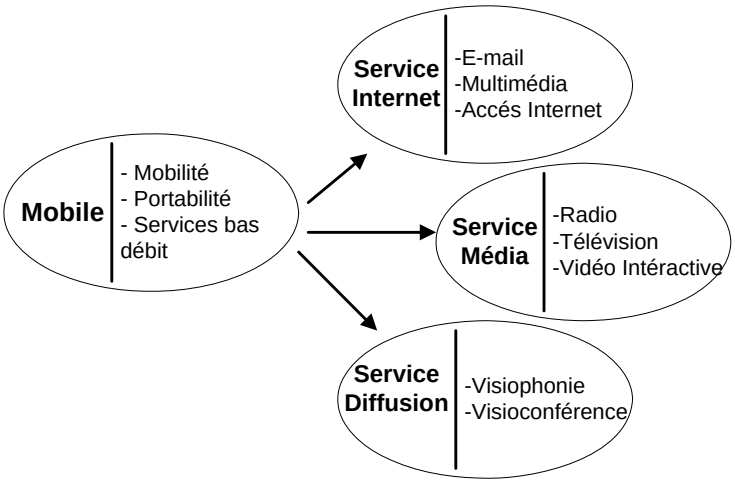


Figure 39. Universalité de l'UMTS

Différents concepts de terminaux mobiles ont vu le jour ces dernières années. Ils intègrent les mêmes principales fonctionnalités présentées dans le schéma fonctionnel d'un terminal mobile visiophonique de la figure 40. D'après cette figure, on constate qu'un terminal mobile multimédia est constitué de différents périphériques tels qu'une caméra pour l'acquisition et un écran pour l'affichage des images, un codeur vidéo pour la compression et la décompression des séquences vidéo. La compression est utilisée soit pour transmettre les données vidéo sur le réseau ou soit pour le stockage. Par contre, la décompression est utilisée pour afficher les images sur l'écran du téléphone. De même, on trouve un micro, un haut parleur, un codeur audio...

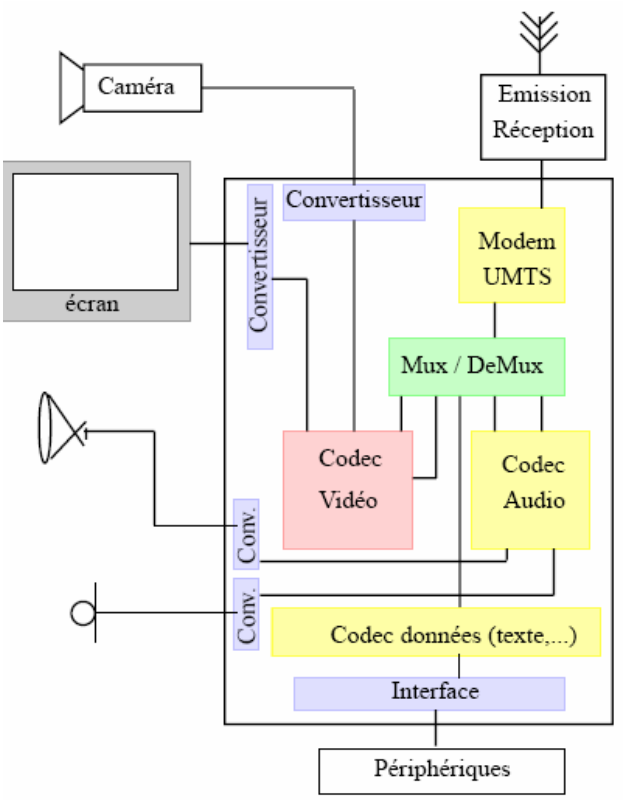


Figure 40. Schéma fonctionnel d'un terminal mobile multimédia

La table 9 donne une description de quelques réalisations proposées sur le marché. Ces terminaux visiophoniques utilisent les codeurs vidéo H.263 et MPEG4. La résolution vidéo est de format QCIF (176x144). L'autonomie peut atteindre 5h dans le cas du mobile SAGEM.

Table 9. Exemples de terminaux visiophoniques

Fabriquant	Caractéristiques
Samsung SGH-Z230 (2006) [62]	Visiophonie mobile : codage H.263, MPEG4 Résolution vidéo : QCIF (176x144 pixels) Résolution de l'écran : 176x220 pixels Autonomie : 2h30
LG KU311 (2006) [63]	Visiophonie mobile : codage H.263, MPEG4 Résolution vidéo : QCIF (176x144 pixels) Résolution de l'écran : 176x220 pixels Autonomie : 4h
Sagem My 600x (2006) [64]	Visiophonie mobile : codage H.263, MPEG4 Résolution vidéo : QCIF (176x144 pixels) Résolution de l'écran : 176x220 pixels Autonomie : 5h
Toshiba TS 705 (2006)[65]	Visiophonie mobile : codage H.263, MPEG4 Résolution vidéo : 128x96 pixels Résolution de l'écran : 176x220 pixels Autonomie : 3h

II.5.3.2 Contraintes de la visiophonie mobile :

Avant de pouvoir transmettre une image, il faut la capturer. Les premiers éléments à spécifier sont donc la caméra et les convertisseurs associés qui définissent les différentes caractéristiques de la vidéo : la résolution, la fréquence, l'échantillonnage des pixels (luminance, chrominances). La résolution minimale acceptable sur un terminal mobile est le QCIF (176x144) en 4 : 2 : 0 (luminance en pleine résolution et les chrominances en quart de résolution). Nous fixons également une fréquence d'image minimale égale à 10 Hz afin de conserver une bonne fluidité de la vidéo.

Un deuxième élément jouant sur la qualité de service est l'écran. Les contraintes sont similaires à celles de la caméra (résolution, fréquence image, nombre de bits par pixel) avec la taille de l'écran et sa fréquence de rafraichissement. L'écran est un matériel critique du mobile du fait de sa consommation. De plus, si on veut intégrer de la vidéo, il doit être en couleur et d'une taille suffisante pour une bonne visibilité de la vidéo. Le choix de la technologie adaptée est également un point important, dans le cas de la visiophonie mobile, le rendu doit être aussi visible en environnement intérieur qu'extérieur.

L'une des contraintes qui s'impose dans le cas d'un terminal mobile est l'autonomie. Pour tout système embarqué, la dissipation d'énergie est un problème majeur. Des optimisations tant au niveau algorithmique qu'au niveau architectural seront nécessaires afin de satisfaire à cette condition de faible consommation. D'autres sources non négligeables de dissipation d'énergie sont également présentes dans un terminal mobile. La plus coûteuse est l'écran qui présente aujourd'hui la plus grande part de la consommation d'un terminal [66]. On peut aussi citer l'émetteur/ récepteur RF, les divers périphériques (micro, haut parleur…) [67].

II.5.4 Conception de systèmes embarqués dans l'approche *codesign* :

Durant la conception d'un SoC, le concepteur aura à choisir le composant programmable qui sera le cœur du système, la plupart des architectures sont basées sur des processeurs à usage général. Ces processeurs sont extensibles du fait que beaucoup d'applications peuvent y être implantées, tandis que les performances obtenues peuvent être inférieures à celles obtenues avec des processeurs dédiées à des applications spécifiques. Mais l'approche de conception mixte logicielle/matérielle permet à l'application d'atteindre des performances inaccessibles aux approches de conception classiques [68] [69]. Les réalisations logicielles sont préférées pour des raisons d'évolution et de coût. Par contre, les réalisations matérielles sont dédiées aux fonctionnalités nécessitant des circuits spécialisés ou des performances élevées.

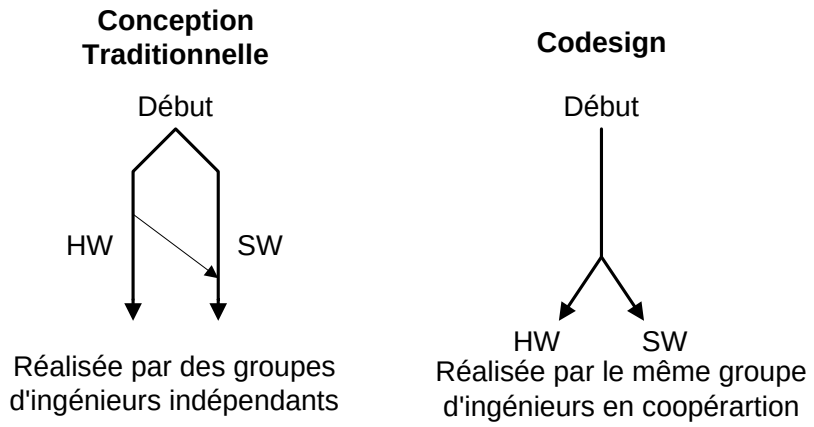


Figure 41. Conception traditionnelle et *codesign*

Le *codesign* implique donc une conception en même temps du matériel et du logiciel. La figure 41 illustre la différence entre la méthodologie de conception traditionnelle et le *codesign*. En fait, dans la conception traditionnelle, la définition de l'architecture est suivie par le découpage des tâches qui vont être réalisées par les équipes du matériel et du logiciel. L'équipe du matériel réalise une description du système en utilisant un langage de description matériel tel que le VHDL ou le Verilog. Puis elle réalise la synthèse et la génération des circuits intégrés en utilisant des outils de CAO. L'équipe du logiciel est responsable de l'écriture du code qui va être compilé et exécuté sur des processeurs d'usage général. Ensuite, les équipes réalisent l'intégration physique des deux parties.

Mais, on constate que le manque d'interaction entre ces deux équipes pendant les étapes de développement peut générer plusieurs problèmes d'intégration. Dans [69], il est rapporté que 71.5 % des projets « système embarqué » n'atteignent pas 30 % des performances attendues durant la phase de conception.

Ces problèmes peuvent être évités par l'utilisation d'une méthodologie de conception conjointe logicielle/matérielle. En effet, les équipes du logiciel et du matériel travaillent ensemble et à chaque étape de conception, elles réalisent l'intégration et le test des spécifications. Les opérations de test et d'intégration génèrent une augmentation du temps de conception. L'intégration finale lors du prototypage est réalisée sans difficulté. On constate que la méthodologie de conception conjointe réduit le temps total de conception puisqu'elle réduit le nombre de retours à des étapes antérieures de conception provoqués par la détection d'erreur.

II.6 Linux pour les systèmes embarqués :

II.6.1 Le système d'exploitation Linux :

Linux [70] est un système d'exploitation entièrement gratuit développe initialement par Linus Thorvald au début des années 90 comme étant une implémentation Minix qui ne devait fonctionner que sur des architectures x86. Le nombre exponentiellement croissant de développeurs travaillant sur Linux lui a permis d'intégrer rapidement toutes les fonctionnalités de base présentes sur UNIX et conquérir la plupart des architectures de processeurs tels que Sparc, Power PC ou Alpha et dominer le marché des serveurs web.

Linux est conforme à la norme IEEE POSIX (*Portable Operating System Interface X*), ce qui signifie que les programmes développés sous Linux peuvent être recompilés facilement sur d'autres systèmes d'exploitation compatibles POSIX. Linux est un système d'exploitation libre. En fait, le code source des différents composants du système est disponible gratuitement. Ce même code source peut être distribué gratuitement en respectant les règles de la licence GPL (*General Public Licence*).

II.6.2 Linux et l'embarqué :

Vue la complexité croissante des systèmes de traitement de l'information, on est arrivé à un stade où il devient impossible de gérer les ressources matérielles d'une architecture sans avoir recours à une logique programmée (logiciel) plus ou moins complexe qui assure l'accès abstrait aux ressources (virtualisation des accès aux ressources) et cela afin de masquer l'accès au matériel. Cette interface logicielle est appelée système d'exploitation.

Durant les dix dernières années, on a assisté à la démocratisation des systèmes informatiques notamment les systèmes informatiques embarqués qui était jusqu'à un passé récent confinés aux applications hautement critiques (nucléaire, avionique…). Une telle

démocratisation a permis aux systèmes embarqués de conquérir divers domaines (du téléphone mobile à l'ordinateur de poche…) [71].

Ce progrès a sollicité de façon considérable le domaine de conception des architectures et des systèmes d'exploitation embarqués car au contraire d'un système informatique classique, dans un système embarqué, il est impossible de s'abstraire des contraintes liées à l'environnement physique dans lequel il s'exécute.

En fait, une architecture embarquée est caractérisée à la fois par une forte interaction avec l'environnement extérieur dans lequel elle évolue imposant des contraintes Temps Réel plus ou moins fortes selon le type d'utilisation de l'architecture, ainsi que par le caractère limité des ressources dont elle dispose que se soit en terme de capacité de stockage, d'énergie ou de bande passante pour les communications avec le monde extérieur [72][73].

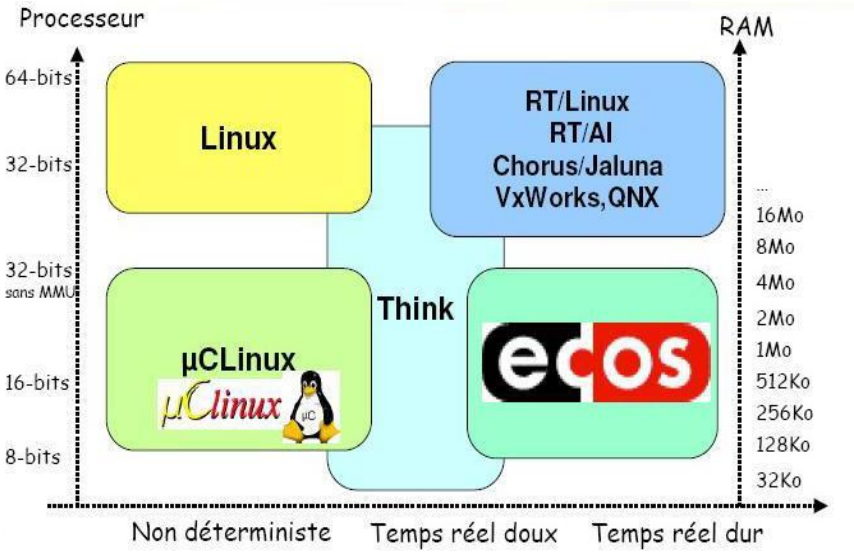


Figure 42. Systèmes d'exploitation pour les applications embarquées

Le choix d'un système d'exploitation (OS : *Operating System*) dépend essentiellement des contraintes imposées et des ressources disponibles [74]. La figure 42 illustre une classification des systèmes d'exploitation pour l'embarqué en fonction des contraintes temporelles et des ressources mémoire disponibles.

II.6.3 Linux embarqué :

Intégrer Linux dans un système embarqué (Linux embarqué) présente des avantages et inconvénients que nous allons détailler [75].

II.6.3.1 Avantages :

- **Avantage d'un système d'exploitation :** Linux, comme un grand nombre d'autres systèmes d'exploitation, est multitâche et multiutilisateur. Il est possible d'y exécuter plusieurs processus qui se partagent les ressources (CPU, mémoire, ports de communication…) de la machine. De même, il permet d'avoir un système de fichiers à disposition. Enfin, les protocoles TCP/IP sont pris en charge par Linux. Cela inclut les pilotes pour une multitude de cartes reéseau Ethernet. Tous les clients et serveurs sont disponibles sous Linux que ce soit FTP, Telnet, SNMP, SMTP, SLIP, PPP...

- **Coût du logiciel :** Linux est un Logiciel Libre (sans coût apparent) ! Les systèmes d'exploitation embarqués commerciaux de WindRiver ou QNX entraînent un coût important non seulement pour pouvoir commencer le développement (achat des outils de développement et des licences) mais ensuite par le reversement d'une redevance sur chaque système développé (*royalties*).
- **Portabilité :** Linux a été porté vers des dizaines de plateformes différentes. Les sources de Linux, en libre diffusion, intègrent déjà le code source pour plusieurs architectures de processeurs (x86, PowerPC, ARM, MIPS, SH, 68K…)
- **Un marché :** Les technologies actuelles évoluent d'une façon importante. Le développement d'un projet ne doit pas durer des années, sinon le produit est totalement dépasse lors de sa disponibilité sur le marché. En choisissant Linux, un concepteur dispose d'un système d'exploitation qui évolue constamment. Là où QNX doit employer des dizaines de développeurs pour maintenir leur produit à jour, Linux dispose de milliers de programmeurs motivés. De plus, Linux ayant pris une importance énorme, la plupart des nouvelles technologies sont souvent conçues sur ce système. Enfin, la conception d'applications destinées à des systèmes utilisant Linux embarqué peut être faite sur la plateforme de développement sous Linux (PC).

II.6.3.2 Inconvénients :

Résultat d'un travail communautaire, Linux a quelques inconvénients qu'il est judicieux de connaître :

- **Sécurité :** Le fait que Linux soit constamment amélioré par de nombreux développeurs lui a permis de devenir un système d'exploitation sécurisé au maximum. Mais la disponibilité des sources fournit le moyen de trouver une faille dans le système d'exploitation, cette faille peut être rapidement corrigée si la personne qui l'a localisée est une personne de bonne fois (ce qui est le cas dans la plupart des cas), si cette faille est localisée par une personne malveillante elle pourra l'exploiter pour son propre compte.
- **Open Source :** L'*Open Source* n'est pas qu'un avantage. Cela permet de pouvoir profiter de logiciels gratuitement (Logiciels Libres). Cependant, tous les programmes soumis à la licence GPL doivent toujours être distribués sous licence GPL. Ainsi, on n'a pas le droit de récupérer Linux, d'y apporter certaines modifications puis de le revendre sous une licence plus restrictive. Tout programme dérivé des sources de Linux doit être diffusé selon la licence GPL. Le portage de Linux pour une plateforme spécifique est un travail important qui ne peut pas être protégé par une licence spéciale.

II.6.4 Mise en œuvre de µClinux sur NIOS II :

Le projet µClinux [76] est le portage du noyau Linux (Linux embarqué) pour les processeurs dépourvus de MMU (*Memory Management Unit*). µClinux est un système d'exploitation multitâche qui cible les applications embarquées telles que le contrôle/commande dans les automobiles, les téléphones cellulaires et les appareils photo numériques... C'est un système d'exploitation fortement configurable et adaptable et peut être facilement configuré pour satisfaire les exigences spécifiques à l'application [75].

La société Microtronix [77] a réalisé il y a quelques années le portage de µClinux pour le processeur de première génération NIOS. Ce portage est payant bien que basé sur des projets GPL, non disponible au libre téléchargement et n'a pas été complètement intégré dans le projet µClinux. Actuellement, le portage de µClinux pour le processeur NIOS II est complet sous licence GPL, on peut le télécharger gratuitement à partir de [78].

II.7 Conclusion :

Dans ce chapitre, on a pu voir que les avancées actuelles dans la technologie des semi-conducteurs et des méthodologies de conception permettent le développement de systèmes numériques complexes sur puce. Les derniers circuits FPGA permettent également le développement de systèmes complets. Ces systèmes sont constitués généralement de cœur(s) de processeur, d'accélérateur(s) matériel(s), de périphériques d'entrées/sorties…

L'approche *codesign* est utilisée afin de faciliter la conception d'un tel système. Cette architecture matérielle complexe et fortement parallèle nécessite l'usage d'un système d'exploitation tel que Linux capable de gérer les différentes unités qui la composent, tout en tenant compte des critères de performance et de superviser l'exécution des applications.

Partie III : Conception du système multimédia embarqué

Cette deuxième partie est divisée en deux chapitres. Le premier chapitre est consacré à la description de la conception d'une plateforme matérielle d'acquisition, de traitement et de restitution vidéo. Le deuxième chapitre présente l'étude et l'implantation du codeur H.263 sur cette plateforme en utilisant la conception conjointe logicielle/matérielle.

Chapitre III: Plateforme matérielle de traitement vidéo

III.1 Introduction :

Ce chapitre est consacré à l'étude et à la conception d'une plateforme matérielle d'acquisition, de traitement et de restitution vidéo servant de préalable à toute étude d'algorithme de traitement vidéo. La plateforme matérielle s'articule autour de la carte Stratix II d'Altera complétée d'une interface caméra et d'une interface VGA. Le cœur du système met en œuvre le module IP NIOS II d'Altera dans l'environnement de développement Quartus II d'Altera. Les modules IPs d'acquisition, restitution vidéo ainsi que des modules IPs de base nécessaires (module FIFO, DMA (*Direct Memory Access*) ont été développés. Le système d'exploitation Linux embarqué µClinux a été utilisé pour le contrôle logiciel de la plateforme matérielle. Enfin, l'ensemble des blocs IPs a servi à constituer une bibliothèque.

III.2 Environnement de développement d'un système SoPC :

III.2.1 Conception d'un système SoPC :

Quartus II est un logiciel proposé par la société Altera [79] permettant la gestion complète d'un flot de conception FPGA. La figure 43 présente l'interface graphique de Quartus II.

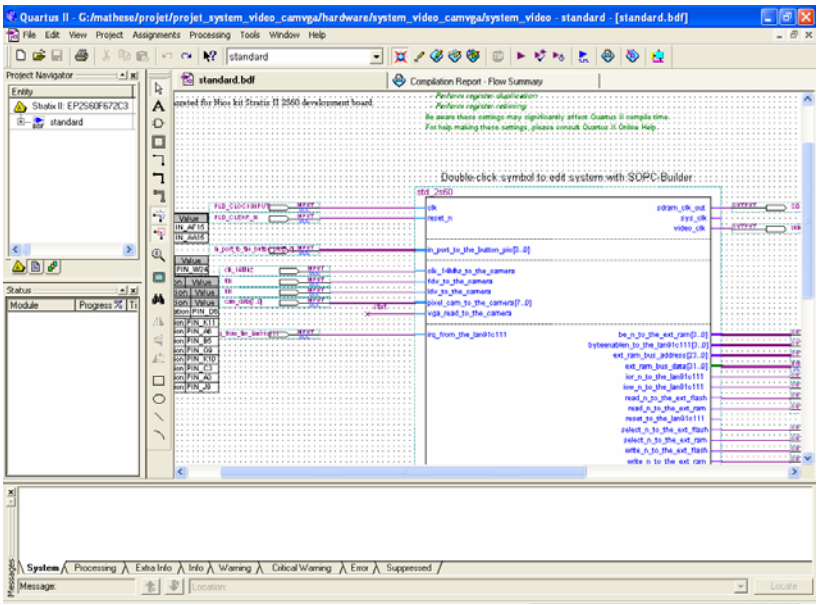


Figure 43. IDE Quartus II

Ce logiciel permet de faire une saisie graphique ou une description VHDL/Verilog d'architecture numérique, d'en réaliser une simulation en utilisant le simulateur *ModelSim* de Mentor Graphics, une synthèse et une implantation sur FPGA. Il comprend une suite de fonctions de conception au niveau système permettant d'accéder à la large bibliothèque d'IPs d'Altera et un moteur de placement/routage intégrant la technologie d'optimisation et des solutions de vérification [80]. D'une manière générale, un flot de conception ayant pour but la configuration de composants programmables se déroulent de la manière suivante (figure 44).

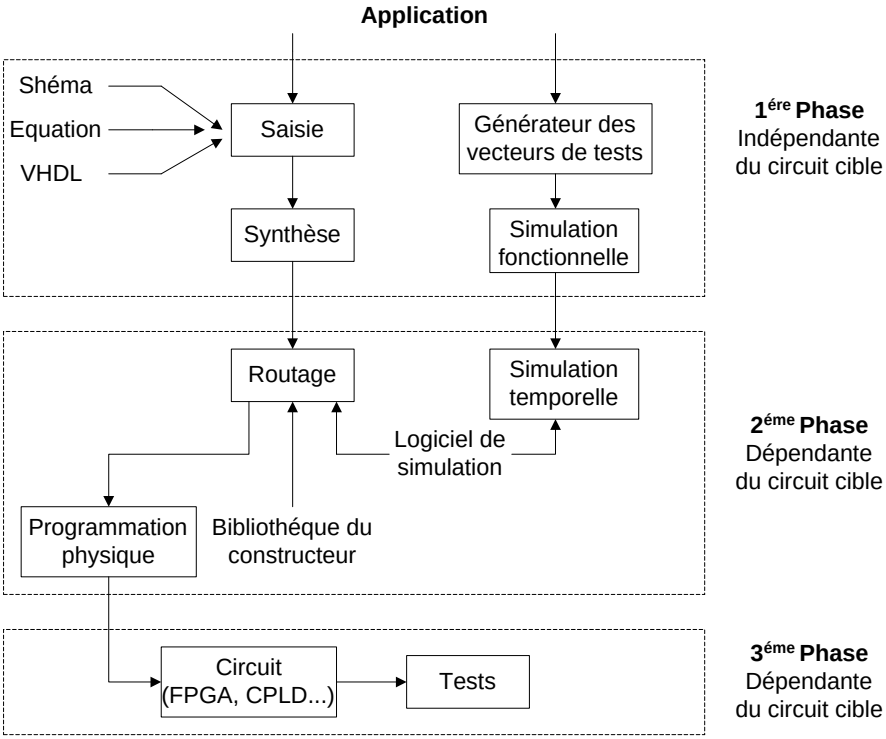


Figure 44. Flot de conception

L'IDE *(Integrated Design Entry*) Quartus II intègre l'outil SOPC Builder qui permet de construire un système SoPC intégrant divers périphériques d'E/S tels que le processeur NIOS II, les contrôleurs de SRAM et de SDRAM, un contrôleur DMA (*Direct Memory Access*)… De même, on peut intégrer son propre composant dans le design sous forme d'un bloc IP externe (Interface caméra, VGA…). On peut ainsi intégrer autant de périphériques que l'on veut, n'étant limité que par le nombre de broches et de cellules logiques du circuit FPGA. Le mapping mémoire et le niveau des interruptions du design sont fixés durant cette phase. La figure 45 montre la mise en œuvre de l'outil SOPC Builder. C'est en fait la première passerelle avec le logiciel embarqué.

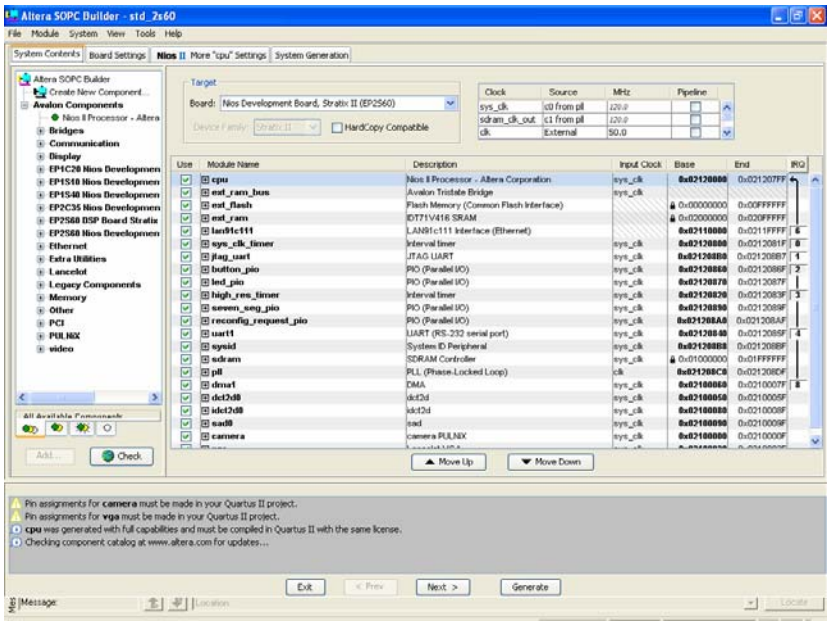


Figure 45. SOPC Builder et mapping mémoire

A l'issue de la phase de construction du système SoPC, Quartus II génère le projet en intégrant tous les modules IPs. Après synthèse, on a le fichier de programmation du circuit FPGA correspondant au design SoPC mais aussi un kit de développement logiciel qui comprend tous les fichiers en langage C (.h et .c) pour piloter les périphériques d'E/S d'Altera. C'est en fait la deuxième passerelle avec le logiciel embarqué. L'offre de *codesign* apparaît ici avec la possibilité de développer une partie de l'application par matériel ou de le faire en logiciel par langage C.

III.2.2 Linux embarqué pour système SoPC :

Le développement d'une application embarquée dans le système SoPC nécessite l'utilisation d'un système d'exploitation capable de gérer les différentes unités qui la composent, tout en tenant compte des critères de performance [75]. La société Microtronix a réalisé le portage de µClinux pour le processeur NIOS II. µClinux est une adaptation du noyau Linux 2.6 pour le processeur NIOS II. La mise en œuvre de µClinux sur le processeur NIOS II nécessite l'utilisation d'un outil de développement logiciel qui est l'IDE Eclipse [81] avec des plugins Altera pour s'interfacer à Quartus II. Eclipse permet de créer un projet µClinux (système de fichiers root et noyau µClinux) et de faire une compilation croisée d'une application µClinux. La figure 46 représente l'environnement graphique de l'Eclipse.

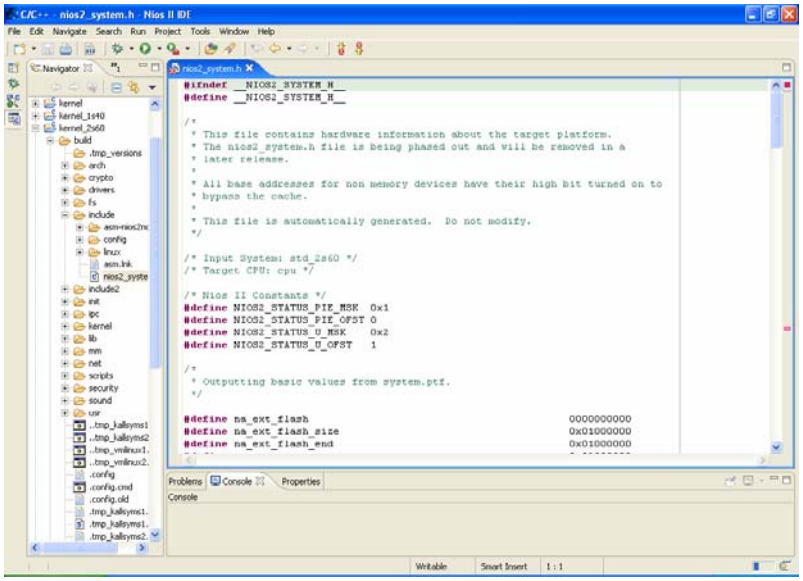


Figure 46. IDE Eclipse

III.3 Carte de développement : carte Altera Stratix II

Pour pouvoir réaliser un système SoPC, on a besoin d'un matériel spécifique (carte cible). Altera propose des cartes de développement pour mettre en œuvre notamment son offre de *codesign*. Ces cartes intègrent toutes un circuit FPGA associé à des périphériques externes (JTAG, SRAM, SDRAM…). La carte cible choisie est une carte haute gamme Stratix 2S60 [82] dont la figure 47 donne une vue d'ensemble. L'Annexe A représente le synoptique de cette carte.

Figure 47. Carte Altera Stratix 2S60

III.3.1 Description des éléments de la carte Stratix II :

La carte Stratix 2S60 comprend les fonctionnalités suivantes :

- FPGA : (Stratix EP2S60 F672 C3) : comporte 60440 Eléments Logiques (48352 ALUTs) dans un boîtier de 672 broches (FBGA).
- CPLD : (MAX EPM7128AE) : ce composant n'est pas disponible à l'utilisateur. Il est dédié au contrôle de la configuration de la carte (boot sur la mémoire FLASH par exemple).
- Flash (AM29LV065D) : cette mémoire de 16 Mo permet de stocker de manière non-volatile un programme de boot, la configuration du circuit FPGA ou le programme utilisateur (exécuté par le processeur NIOS II). Le raccordement au circuit FPGA se fait par l'intermédiaire du contrôleur (CPLD).
- SRAM (2*IDT71V416) : de taille 1Mo, 2 modules de 256K*16 chacun, connectés au circuit FPGA par l'intermédiaire du contrôleur (CPLD).
- SDRAM 16 Mo (MT48LC4M32B2) : 4M*32bits.
- Oscillateur (50 MHz) : fournit le signal d'horloge au circuit FPGA.
- Interfaces : interface Ethernet (LAN91C111), afficheurs 7 segments, 2 connecteurs RS232, 4 boutons poussoirs, 2 connecteurs JTAG.

III.3.2 Caractéristiques du composant Stratix II d'Altera :

Le circuit Stratix II est basé sur une technologie 90 nm et 1.2 V. Ce circuit offre une densité de 48352 ALUTs, 310 Ko de mémoire embarquée, 288 blocs DSPs (9bits x 9bits), 6 PLLs et 493 entrées/sorties. Ce composant est optimisé pour l'implantation de systèmes SoPC.

III.4 Système d'acquisition et de traitement vidéo :

Nous avons conçu et réalisé une plateforme matérielle d'acquisition de traitement et de restitution vidéo servant à l'évaluation de notre méthodologie de conception logicielle/matérielle pour les systèmes multimédia embarqués [83]. La figure 48 représente notre plateforme.

Figure 48. Système d'acquisition, traitement et restitution vidéo

Le système réalisé est composé d'une caméra numérique permettant l'acquisition de l'image et la génération des signaux numériques. Ces signaux sont traités par la carte cible Altera Stratix 2S60. Après traitement, les échantillons de l'image ainsi que les signaux de contrôle sont envoyés vers l'interface VGA connectée à un moniteur VGA. L'image restituée a une résolution de 640x480 en niveaux de gris. Pour cela, un bloc IP pour l'acquisition de l'image de la caméra a été développé en VHDL. Le module contrôleur VGA est le module Lancelot [84] qui est un matériel libre fourni sous forme d'un bloc IP.

Dans ce paragraphe, on va présenter les deux modules d'acquisition et de restitution ainsi que la synchronisation entre eux pour le bon fonctionnement du système. L'ensemble du design est sous contrôle du système d'exploitation sous µClinux [85].

III.4.1 Périphérique d'acquisition d'images :

III.4.1.1 Module d'acquisition :

L'acquisition des images est réalisée par une caméra PULNiX TM-9701 [86]. La caméra fournit en sortie des pixels sur 8 bits, un signal d'horloge clk_14MHz et les signaux de synchronisation LDV (*Line Data Valid*) et FDV (*Frame Data Valid*). L'Annexe B représente les caractéristiques de la caméra PULNiX.

Cette caméra génère différents signaux numériques en mode différentiel, qui sont incompatibles avec les entrées de la carte à base de FPGA. Pour cette raison, on a réalisé une carte d'adaptation de signaux numériques différentiels en signaux numériques mono polaire. Cette carte est basée sur le circuit « DS26C32ATN » [87] (figure 49).

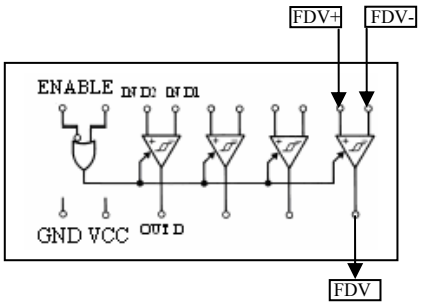


Figure 49. Architecture interne du circuit

L'Annexe C présente le schéma de la carte interface caméra. Pour connecter cette carte à la carte Stratix II, on a utilisé le connecteur J11 de la carte de développement. La table 10 présente pour chaque signal, les broches utilisées du connecteur.

Table 10. Connexion de la carte d'interface caméra à la carte de développement

Connecteur J11 utilisé	Nom du signal	Nom de la broche
J11-17	Clk_14Mhz	H10
J11-15	LDV	D6
J11-13	FDV	A5
J11-18	Cam_data [0]	K11
J11-16	Cam_data [1]	A6
J11-14	Cam_data [2]	B5
J11-12	Cam_data [3]	G9
J11-10	Cam_data [4]	K10
J11-8	Cam_data [5]	C3
J11-6	Cam_data [6]	A3
J11-4	Cam_data [7]	J9

III.4.1.2 Interface caméra :

L'interface caméra est un bloc IP permettant d'établir la liaison entre le module externe (carte d'interface caméra) et le bus Avalon. La structure générale de cette interface est présentée par la figure 50.

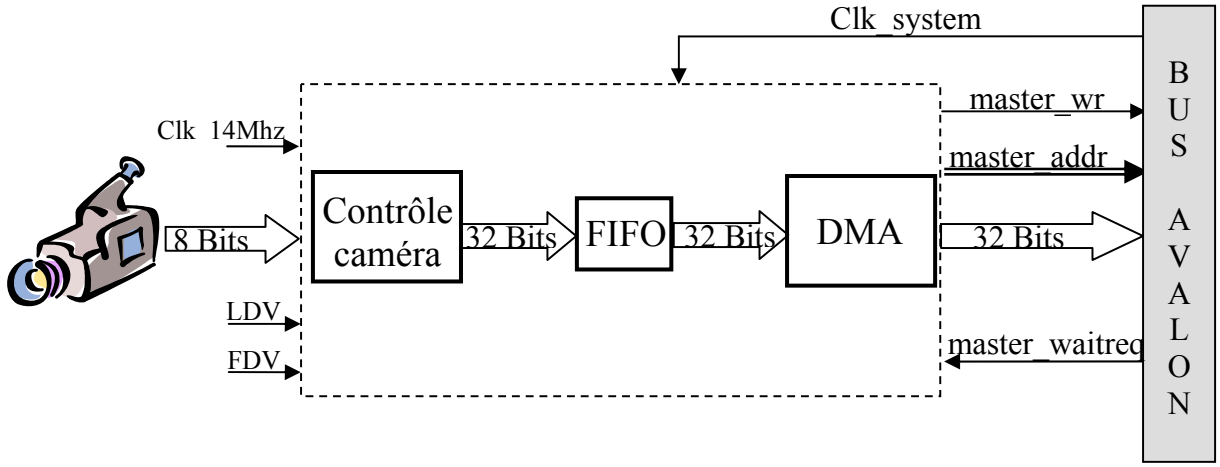


Figure 50. Synoptique de l'interface caméra

L'interface caméra permet d'envoyer les données provenant de la caméra et d'autres signaux vers le bus Avalon. Elle est constituée de trois modules. Un module permet d'envoyer les pixels provenant de la caméra vers le module FIFO sur 32 bits. En effet, dans le but d'utiliser la totalité de la taille du bus 32 bits, il a fallu concaténer l'ensemble de 4 pixels de 8 bits sur un mot de 32 bits. C'est le rôle du module contrôle caméra.

Une mémoire FIFO qui permet de mémoriser une ligne d'image (640 pixels). Elle joue le rôle de tampon entre l'écriture des données et leur lecture par le DMA. L'écriture sur la FIFO est rythmée par l'horloge de la caméra (14 MHz) avec une fréquence divisée par 4. Par contre, la lecture est rythmée par l'horloge système (120 MHz). En effet, il faut que la lecture des données de la FIFO vers la mémoire soit assez rapide pour suivre le flux de la caméra. Ce qui implique l'importance du DMA.

Le principe essentiel du mode DMA est de pouvoir écrire en même temps qu'il lit et ce par comptage ou décomptage du nombre d'octets à transférer. Ainsi avec le DMA, on peut

effectuer deux opérations à la fois, ce qui accélère notablement la quantité d'informations traitées et donc la vitesse de transfert de données.

Le troisième module c'est le DMA qui permet de transférer les données de la FIFO vers la mémoire à travers le bus Avalon en envoyant les signaux *master_w*, *master_addr* et *master_wrdata*. Le cycle d'écriture peut rester en attente si le bus Avalon envoie le signal *master_waitreq* (le bus est occupé).

III.4.1.2.1 Description de l'interface :

La description de l'interface caméra se fait en deux étapes :

La première étape consiste à la description VHDL, supportée par les fichiers *camera_nios.vhd*, *interface_cam.vhd* et *camera_avalon.vhd* (Annexe D). Le fichier *camera_nios.vhd* décrit les signaux d'E/S de l'interface caméra ainsi que le fonctionnement de l'interface. La figure 51 décrit l'entité de l'interface.

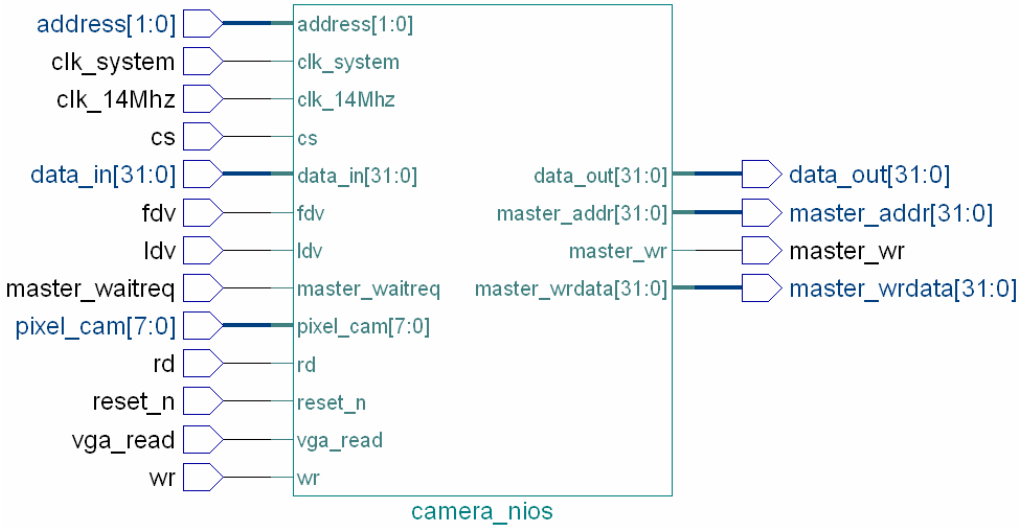


Figure 51. L'entité de l'interface caméra

Le fichier *camera_avalon.vhd* décrit le fonctionnement du DMA et les signaux utilisés pour transférer les données entre l'interface caméra et les périphériques du système (processeur NIOS II, RAM…) à travers le bus Avalon (figure 52).

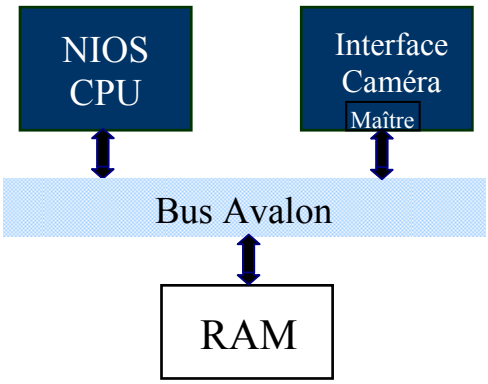


Figure 52. Connexion de l'interface caméra avec le système

La figure 52 montre le port maître qui est connecté à la mémoire à travers le bus Avalon. L'écriture des données dans la mémoire se fait en utilisant le DMA. La table 11 indique les signaux utilisés par le bus Avalon.

Table 11. Signaux de l'interface caméra

Signal	Largeur	Direction	Description
Clk_system	1	In	Horloge du système
Reset_n	1	In	Reset
Master_waitreq	1	In	Signal d'attente généré par le bus Avalon
Master_w	1	Out	Signal d'écriture actif à « 1 »
Master_addr	32	Out	Adresse d'écriture des données
Master_wrdata	32	Out	Données à écrire
Chipselect	1	In	Sélection de l'interface
Read	1	In	Signal de lecture actif à « 0 »
Write	1	In	Signal d'écriture actif à « 0 »
Address	2	In	Adressage de l'interface
Data_in	32	In	Donnée à écrire dans l'interface
Data_out	32	Out	Donnée à lire de l'interface

La deuxième étape est de créer le fichier *class.ptf*. C'est grâce à ce fichier, que le logiciel de développement pourra identifier et implanter l'interface dans le système. Il contient toutes les informations de connexion définies précédemment. La figure 53 présente l'interface qui apparaît lors de l'ajout du périphérique dans SOPC Builder.

Figure 53. Interface caméra

III.4.1.2.2 Simulations de l'interface caméra :

Pour vérifier le bon fonctionnement de l'interface, on a utilisé l'outil de simulation *Modelsim*. Le chronogramme de la simulation fonctionnelle est présenté à la figure 54. Ce chronogramme vérifie le bon fonctionnement de l'interface caméra. En fait, l'écriture de donnée *master_wrdata* sur le bus s'effectue à chaque cycle d'horloge système *clk_system* et tant que le signal *chipselec* et *master_wr* sont actifs à « 1 ». Le cycle d'écriture est en attente tant que le signal *master_waitreq* est actif à « 1 ».

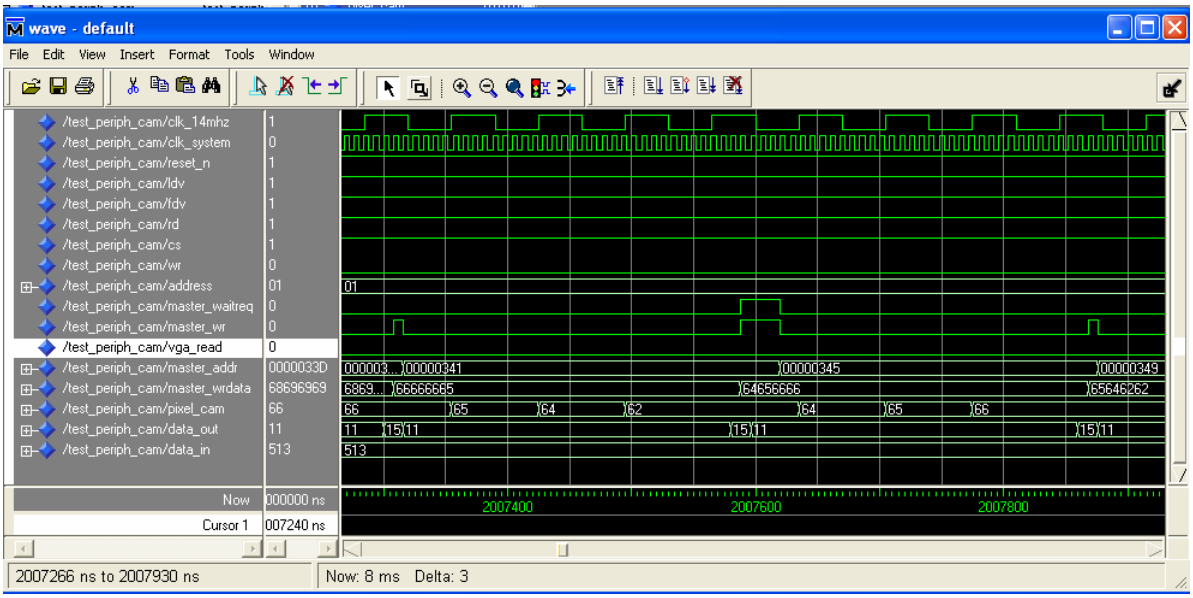


Figure 54. Simulation de l'interface caméra

III.4.1.2.3 Résultats d'implantation de l'interface caméra :

Les résultats issus de la synthèse par Quartus II sont les suivants :

- ALUTs : 416 / 48,352 (<1 %).
- Pins E/S : 149 / 493 (30 %).
- Blocs RAMs: 16,384 / 2,544,192 (<1 %).
- Fréquence maximale : 241,84 MHz.

III.4.2 Restitution d'images :

Les écrans VGA couleurs sont des écrans RVB. Ces écrans reçoivent trois signaux 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. Sur l'écran VGA, ces signaux sont de types analogiques.

Lancelot est un contrôleur vidéo VGA pour le processeur NIOS d'Altera permettant de transférer des données vidéo vers un moniteur VGA. Le périphérique est constitué de deux parties : le cœur (interface VGA) et le matériel (carte d'extension).

III.4.2.1 Module de restitution :

La carte d'extension (figure 55) peut être montée sur la carte de développement Altera Stratix II en utilisant les connecteurs J15 et J16 (Annexe A). L'annexe E représente pour chaque signal la broche utilisée. Cette carte est basée sur un convertisseur vidéo N/A qui permet la conversion numérique/analogique des signaux provenant de l'interface VGA pour l'afficher sur le moniteur avec une résolution de 640x480 en niveaux de gris.

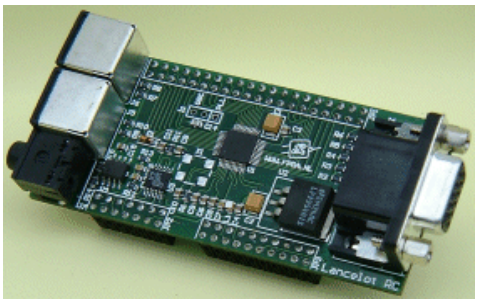


Figure 55. Carte d'interface lancelot

Le synoptique de la carte d'extension VGA est donné par la figure 56. Elle permet de transmettre les couleurs (rouge, vert, bleu) et les signaux de synchronisation à l'écran à travers le connecteur VGA.

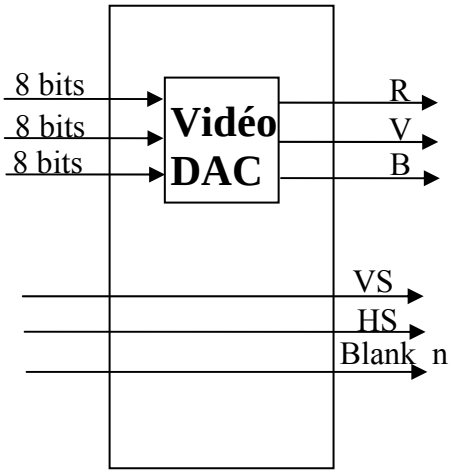


Figure 56. Synoptique de la carte d'extension

III.4.2.2 Interface VGA :

Une interface VGA est nécessaire pour établir la liaison entre le bus Avalon et le module externe (carte d'extension). La structure générale de l'interface VGA est présentée sur la figure 57.

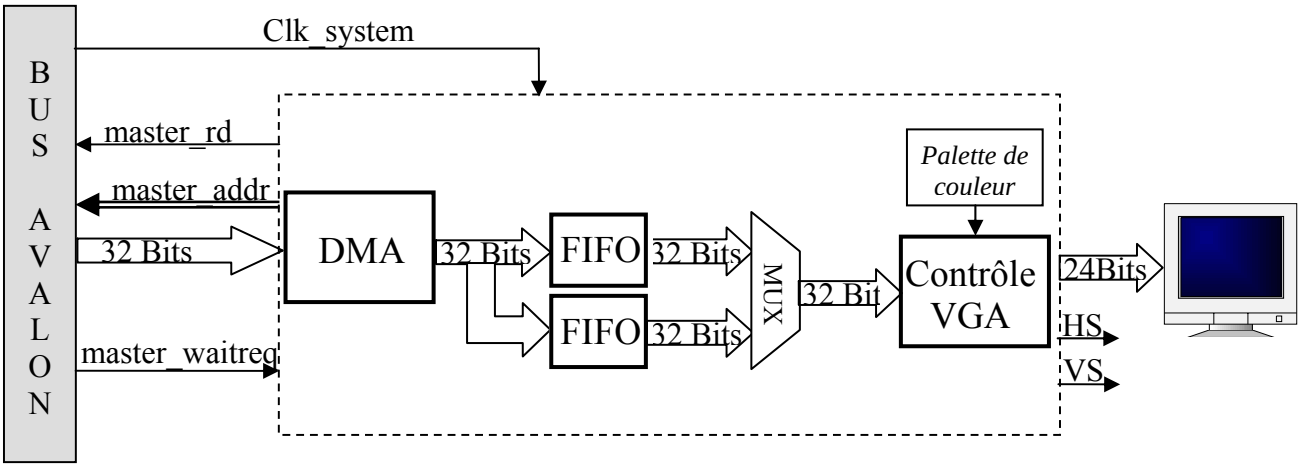


Figure 57. Synoptique de l'interface VGA

L'interface VGA reçoit les données provenant de la mémoire à travers le bus Avalon et les envoie vers le module externe pour afficher l'image. Cette interface est constituée de trois modules. Le module DMA permet de transférer les données de la mémoire vers la FIFO en utilisant les signaux *master_rd* et *master_addr*.

Un module tampon est constitué de deux FIFO qui ont une même profondeur (une ligne d'image soit 640 pixels). En effet, si le DMA écrit dans la première FIFO, le module contrôleur VGA lit l'autre. Ce dernier module envoie les signaux « R », « V », « B » et les signaux de synchronisation vers la carte d'extension. L'écriture dans la mémoire FIFO est rythmée par l'horloge système avec une fréquence de 120 MHz. Par contre, la lecture est rythmée par l'horloge VGA 25 MHz. La figure 58 définit les E/S de l'interface.

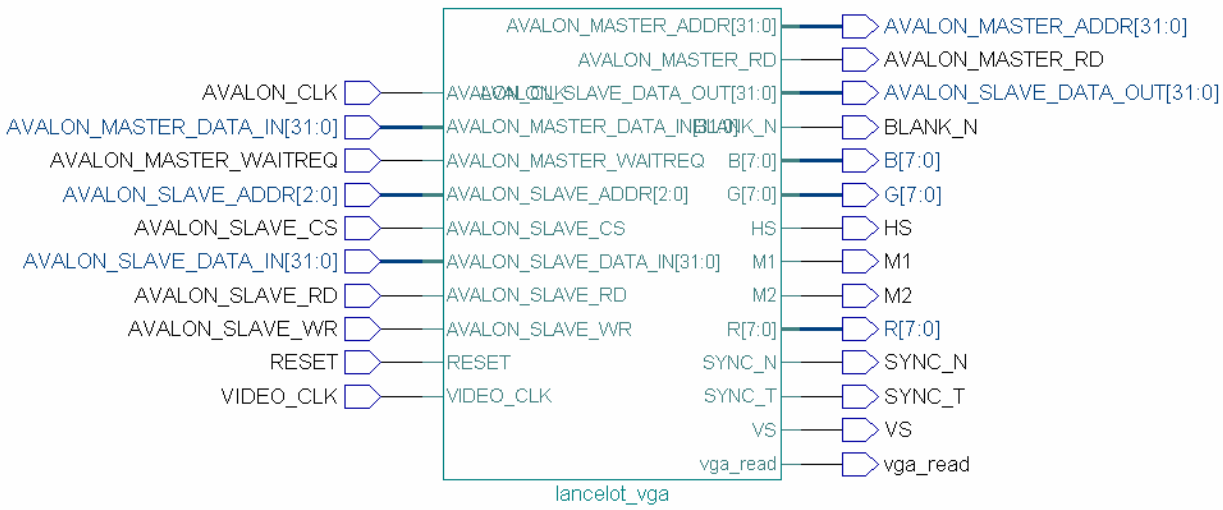


Figure 58. L'entité de l'interface VGA

III.4.2.2.1 Simulations de l'interface VGA :

Le chronogramme de la simulation fonctionnelle est présenté à la figure 59. Ce chronogramme vérifie le bon fonctionnement de l'interface VGA. En fait, la lecture des données *master_rddata* du bus s'effectue à chaque cycle d'horloge système « *clk_system* » et tant que le signal *chipselect* et *master_rd* sont actifs à « 1 ». Le cycle de lecture est en attente tant que le signal *master_waitreq* est actif à « 1 ».

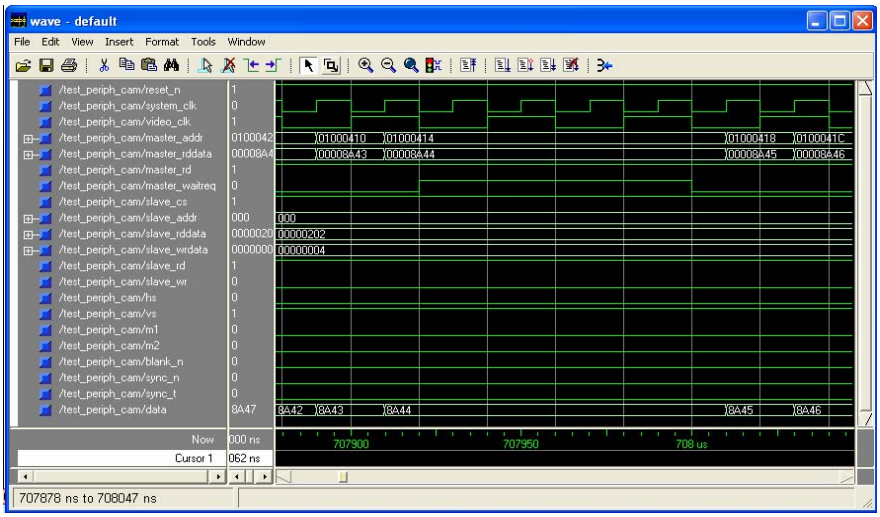


Figure 59. Simulation de l'interface VGA

III.4.2.2.2 Résultats d'implantation de l'interface VGA :

Les résultats issus de la synthèse par Quartus II sont les suivants :

- ALUTs : 686 / 48,352 (<1 %).
- Pins E/S : 171 / 493 (30 %).
- Blocs RAMs : 16,384 / 2,544,192 (<1 %).
- Fréquence maximale : 224,77 MHz.

III.4.3 Système vidéo Temps Réel :

Si certains traitements peuvent être effectués en temps différé (hors ligne) comme par exemple dans le domaine de l'analyse des données, d'autres nécessitent un traitement de données en Temps Réel. Il s'agit dans ce cas de respecter des contraintes de latence (durée entre l'acquisition d'une entrée et la production de la sortie correspondante). La vitesse de traitement doit être compatible avec le débit des données en entrée et en sortie. L'avènement des télécommunications numériques, le domaine de la télévision numérique, du traitement vidéo ont amplifié considérablement le besoin de traitement en Temps Réel.

Notre objectif est de faire l'acquisition, le traitement et la visualisation des séquences d'images et d'avoir le maximum de temps libre pour faire le traitement en Temps Réel des algorithmes de traitement vidéo.

III.4.3.1 Contraintes temporelles :

La caméra fournit 30 images par seconde non entrelacées. De son côté le moniteur VGA nécessite 60 images pas seconde. D'un autre côté, l'image provenant de la caméra sera tronquée de 768x484 pixels à 640x480 pixels au niveau du circuit interface. On constate que chaque image caméra sera affichée deux fois. Le chronogramme de la figure 60 présente le temps mis pour afficher une image par le VGA qui est égal à 16.7 ms.

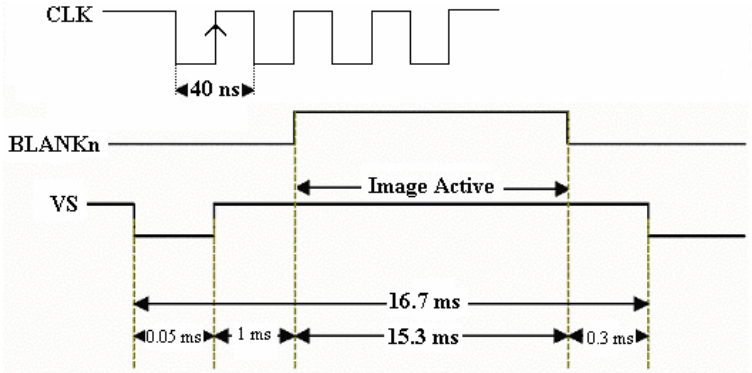


Figure 60. Synchronisation d'une trame

La caméra et le moniteur VGA partagent une même zone mémoire de 640x480 octets. Pour que le système fonctionne, il faut donc être capable de lire ou écrire dans cette mémoire 640x480x90= 27648000 fois par seconde. On a conçu pour chaque interface (caméra et VGA) un composant DMA (*Direct Memory Access*) permettant le transfert direct de données entre l'interface et la mémoire du système. Si l'on compte 3 cycles d'horloge par écriture, ce qui est le cas dans un transfert type DMA, cela nous impose, pour l'horloge système, une fréquence minimale de 82 MHz.

L'asynchronisme existant entre la caméra, le circuit FPGA et l'interface VGA obligent d'intercaler des mémoires de type FIFO pour assurer la continuité du flux de pixels. La caméra fonctionne avec une horloge d'échantillonnage de 14 MHz, l'interface VGA avec une horloge de 25 MHz, tandis que l'horloge système a été fixée à 120 MHz. Des deux côtés, les FIFO ont une capacité de 640 octets, ce qui correspond à une ligne complète.

Si l'on accorde de plus la taille du mot transféré à celle du bus, soit 32 bits, on obtient un gain de 4 dans la vitesse de transfert (4 pixels à la fois). La fréquence minimale est alors ramenée à 20,5 MHz. Au final, dans notre cas, les transferts DMA vers ou depuis la mémoire ne vont consommer que 20,5/120 = 17% du temps total.

III.4.3.2 Synchronisation entre l'interface caméra et l'interface VGA :

Notre système est constitué de trois maîtres qui peuvent accéder à un même esclave (RAM) à travers le bus Avalon (figure 61).

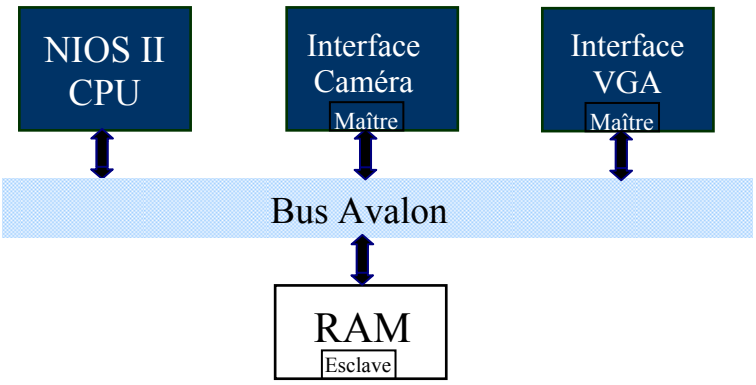


Figure 61. Architecture du système

Pour permettre l'acquisition et la restitution de l'image, les interfaces caméra et VGA doivent partager la même zone mémoire. Un conflit peut être provoqué si les deux interfaces accèdent en même temps à la mémoire, puisque le bus Avalon envoie le même signal *master_waitreq* à ces deux interfaces.

Pour le bon fonctionnement du système, une synchronisation est nécessaire entre ces deux interfaces. En fait une seule interface peut accéder à la mémoire RAM à la fois. Une interface ne peut commencer le transfert de données que si et seulement si l'autre termine. Dans notre cas c'est l'interface VGA qui est prioritaire puisqu'une discontinuité de transfert de données entre la mémoire et la FIFO de l'interface VGA provoque une perturbation au niveau de l'affichage de l'image. En fait, le transfert DMA caméra commence que lorsque le transfert DMA VGA est terminé. Ces deux composants DMA ont été écrits en VHDL afin d'accélérer le transfert de données entre l'interface caméra, l'interface VGA et la mémoire. L'organigramme de la figure 62 représente le protocole de communication entre ces deux interfaces.

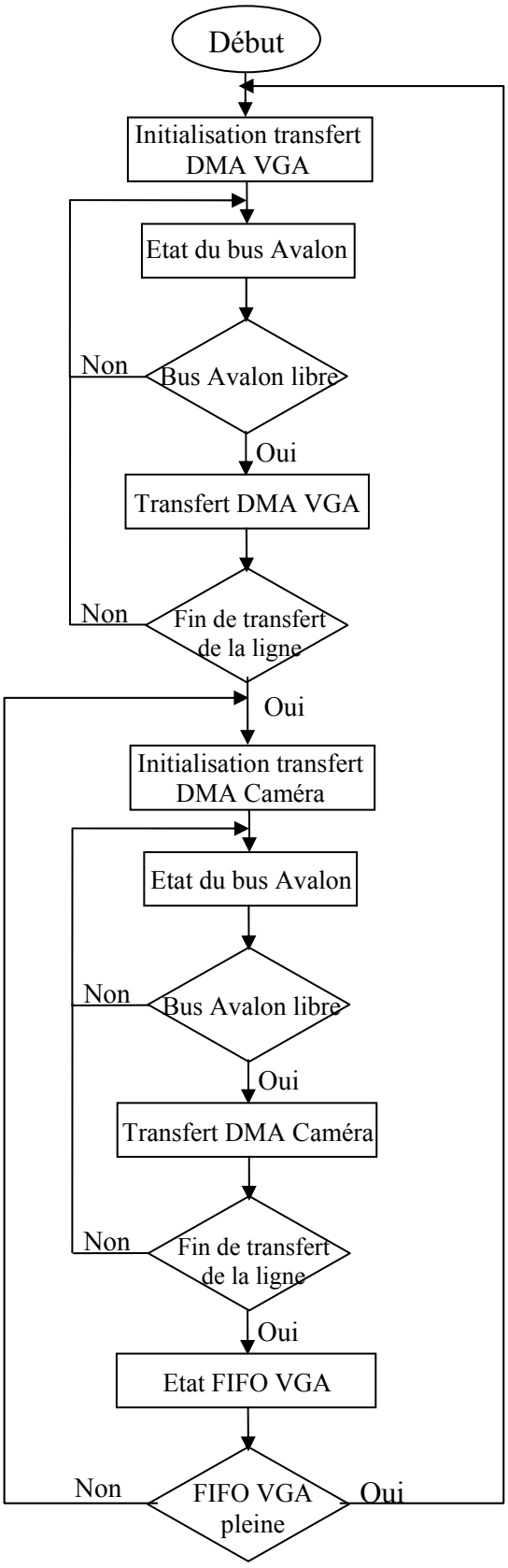


Figure 62. Gestion de la synchronisation de l'interface caméra et l'interface VGA

III.4.4 Conception du système multimédia embarqué :

La figure 63 décrit le système embarqué réalisé. Il est constitué de différents blocs IPs tels que le processeur NIOS II, le bus Avalon et différents périphériques (interface caméra, interface VGA, contrôleur SRAM,…). Le processeur NIOS II est le cœur de notre système embarqué. Il est connecté aux différents périphériques à travers le bus Avalon.

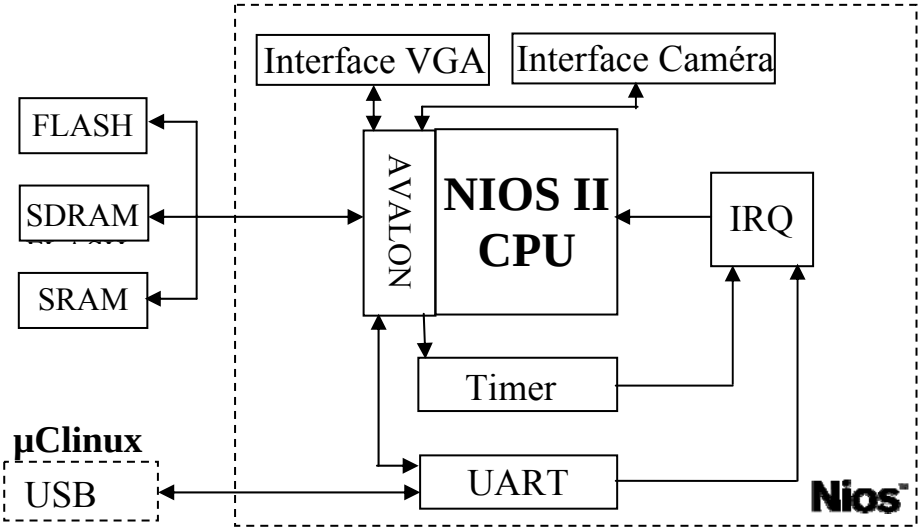


Figure 63. Système embarqué NIOS II

Ces différents composants sont associés, connectés et générés en utilisant l'outil Altera SOPC Builder, comme présenté à la figure 64.

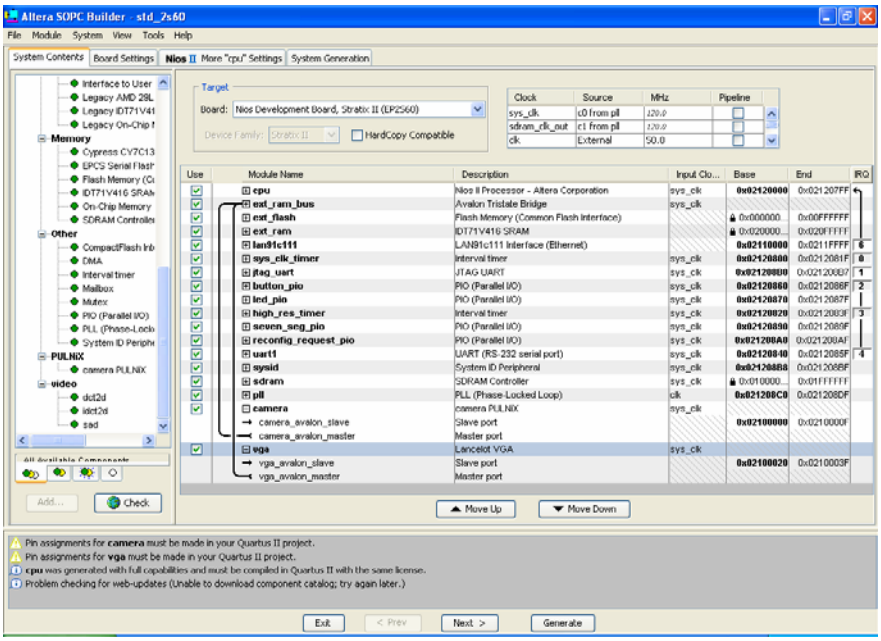


Figure 64. SOPC Builder utilisant l'interface caméra et l'interface VGA

III.4.4.1 Résultats d'implantation :

Les résultats issus de la synthèse par Quartus II sont les suivants :

- ALUTs : 6,873 / 48,352 (14 %).
- Pins E/S : 209 / 493 (42 %).
- Blocs RAMs : 1,154,816 / 2,544,192 (45 %).
- Blocs DSPs : 8/288 (3%)
- Fréquence de fonctionnement : 120 MHz.

L'implantation des différents périphériques constituant notre système multimédia embarqué laisse suffisamment d'espace sur le composant FPGA Stratix II pour l'ajout d'autres blocs IPs ou l'intégration matérielle d'algorithmes de traitement vidéo en Temps Réel.

III.4.4.2 Validation du système embarqué :

Après le développement de la première partie de l'application sous forme de blocs IPs pour bénéficier d'une accélération matérielle, il reste à développer la deuxième partie sous forme logicielle pour bénéficier de la souplesse de l'exécution du code par le processeur NIOS II embarqué dans le système. Afin de valider notre application, µClinux a été intégré au système cible pour contrôler le système d'acquisition, traitement et de restitution d'image [88]. La figure 65 présente l'exécution de l'application µClinux *camvga* sur la carte cible. Le code source C est présenté en Annexe F. Le système multimédia embarqué réalisé est illustré par la figure 66.

Figure 65. Exécution de l'application µClinux *camvga*

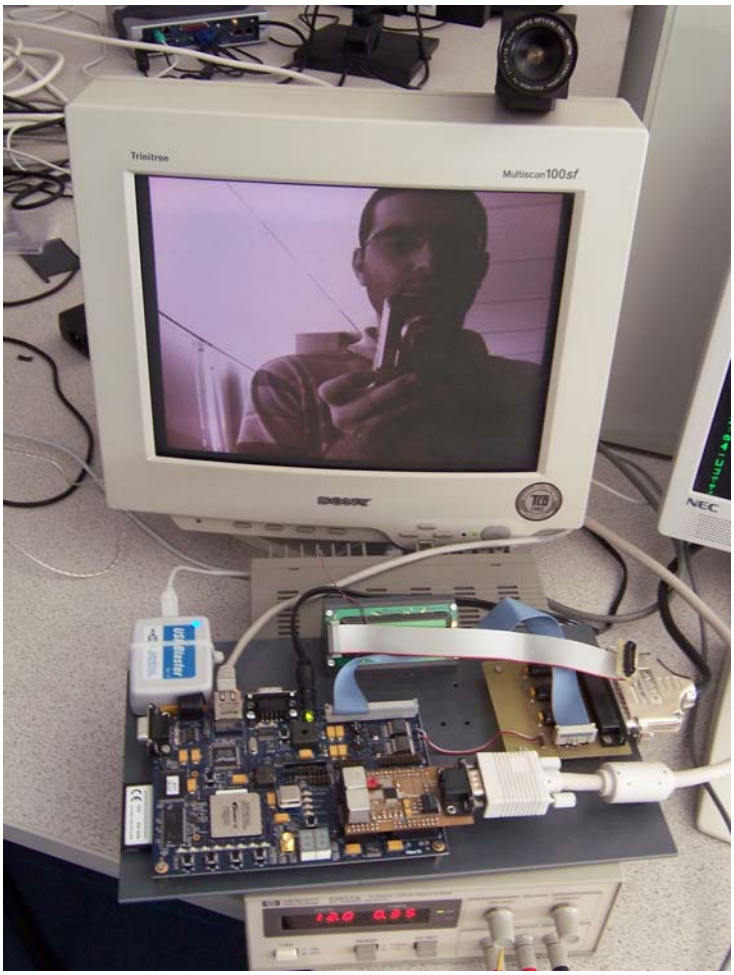


Figure 66. Système multimédia embarqué

III.5 Conclusion :

Au cours de cette étude, on a présenté le développement d'une plateforme matérielle d'acquisition et de restitution vidéo en Temps Réel en utilisant la conception mixte logicielle /matérielle. Cette plateforme se base sur la carte de développement Stratix II d'Altera qui est connectée à la carte d'interface caméra et à celle de l'interface VGA. Le cœur du système est le processeur embarqué NIOS II d'Altera. Les modules IPs d'acquisition et de restitution vidéo ont été développés afin d'interfacer les deux modules externes avec la carte de développement.

On a pu voir en outre que la mise en œuvre de µClinux sur le processeur NIOS II a pu être finalisée. L'introduction d'un système d'exploitation est un apport essentiel dans l'approche de conception *codesign*.

Dans le prochain chapitre, on étudiera l'intégration du codeur de compression vidéo H.263 sur cette plateforme en utilisant l'approche de conception *codesign*.

Chapitre IV: Implantation du codeur H.263

IV.1 Introduction :

Ce chapitre est consacré à l'étude et à l'implantation dans un environnement logiciel/matériel du codeur H.263. Afin de déterminer quels blocs cibles doivent être implantés sous forme matérielle plutôt que logicielle, on a effectué une étude détaillée des différents blocs du codeur en utilisant des séquences vidéo de test. L'étude et l'évaluation de la performance du codeur avant et après optimisation est réalisée sur la plateforme de traitement vidéo décrit dans le chapitre précédent. Enfin, une interprétation des résultats obtenus est présentée.

IV.2 Etude de la complexité :

Le codeur étudié est basé sur le développement originel de la société *Telenor Research and Development* en Norvège [89] satisfaisant à la norme H.263. Les performances du codeur sont liées aux techniques de codage employées qui sont fixées par la norme.

L'étude de la complexité de codeur est faite sur le processeur embarqué NIOS II (version *fast*) à 120 MHz dont la puissance de calcul est de 142 DMIPS et la taille du cache d'instructions et de données est de 64 Ko. La multiplication et la division sont câblées. Le système d'exploitation µClinux est exécuté par le processeur NIOS II pour gérer les différentes unités du système et superviser l'exécution du codeur. L'interface réseau Ethernet est utilisée afin de récupérer et visualiser les séquences vidéo de test reconstruites sur un ordinateur PC. Les séquences de test de la figure 67 correspondent à des séquences soit statiques en ce qui concerne les séquences Claire, Akiyo et Miss America, soit dynamiques pour les séquences Foreman, Carphone et News.

Figure 67. Séquences de test

Dans l'étude qui suit, les séquences Miss America et Foreman sont principalement utilisées pour les tests et l'estimation de performance du codeur. L'ensemble des tests utilisent une recherche exhaustive des vecteurs de mouvement dont la fenêtre de recherche est de taille +/- 15 avec différents valeurs du pas de quantification (QP=8, 13, 31) pour des séquences

QCIF@10 Hz. Le timer du processeur NIOS II (*high_res_timer*) est utilisé pour l'estimation du temps de traitement des différents blocs du codeur.

La qualité du codage est mesurée par des tests objectifs en calculant la valeur du PSNR (*Peak Signal To Noise Ratio*) qui représente le rapport signal/bruit de l'image reconstruite par rapport à l'image d'origine. Plus la valeur du PSNR est élevée, plus l'image compressée est proche de l'image source. De même, on peut utiliser la métrique SSIM [90] (*Structural SIMilarity*) permettant de mesurer la qualité perceptuelle d'une image. En fait, cette méthode consiste à mesurer l'erreur de visibilité entre une image reconstruite et une image de référence en utilisant une variété de propriété du système visuel humain (SVH). La valeur de SSIM est comprise entre 0 et 1. Plus la valeur de SSIM proche de 1, plus les images se ressemblent (les expressions de PSNR et de SSIM sont données en annexe G).

IV.3 Implantation logicielle du codeur H.263 :

Le codeur vidéo considéré ici est de résolution QCIF@10 Hz. Sans option d'amélioration de codage, ce système permet de coder 0,3 fps (*frame per second*).

Il semble, par conséquent, nécessaire d'étudier la répartition du temps CPU suivant les différents traitements constituant l'algorithme de codage afin de pouvoir envisager la réalisation de systèmes Temps Réel traitant des images de plus grande résolution à des fréquences plus élevées. La question qui peut se poser est dans quelle mesure certains blocs sont-ils intéressants à réaliser par matériel ? La figure 68 donne une répartition du temps CPU du codeur H.263 pour les séquences Miss America et Foreman pour QP=13.

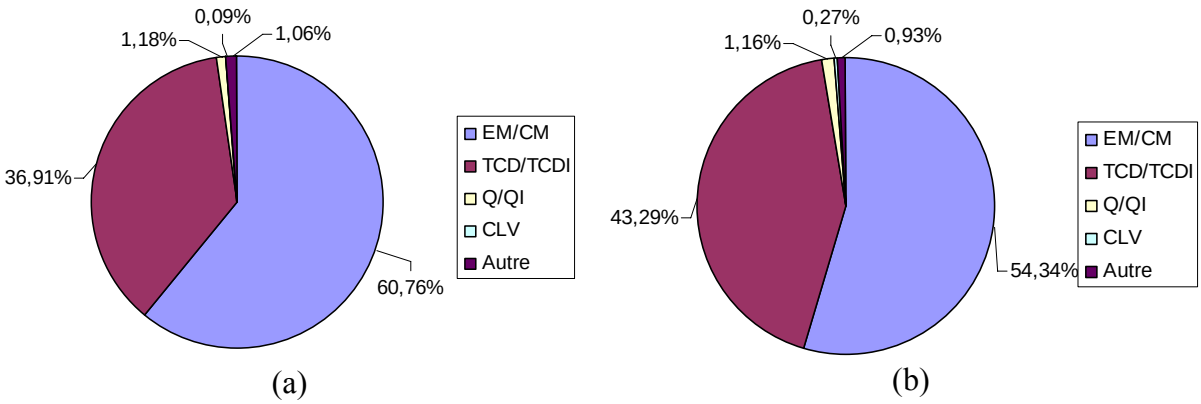
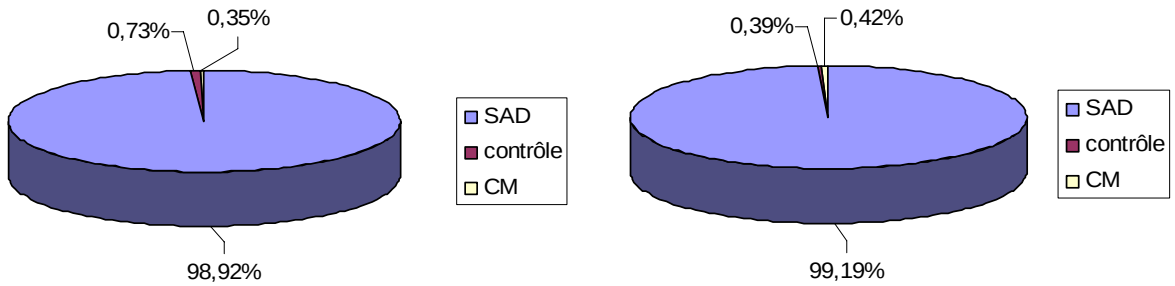


Figure 68. Répartition du temps CPU par bloc de traitement pour les séquences (a) Miss America et (b) Foreman

Le graphe de la figure 68 montre que la complexité est fortement localisée dans les deux blocs de traitement du codeur : l'EM/CM (60.76% du temps CPU) et la TCD/TCDI (36.91% du temps CPU) dans le cas de la séquence Miss America. Une étude plus fine sur le bloc d'estimation/compensation de mouvement (figure 69) indique que les ressources sont principalement utilisées par l'opérateur de calcul de distorsion, à savoir, la somme des valeurs absolues des différences (SAD). En effet, ce calcul de distorsion dans le cas de Miss America représente à lui seul 60% du temps CPU total pour le codeur H.263.



(a) (b)

Figure 69. Répartition du temps CPU dans l'EM pour les séquences (a) Miss America et (b) Foreman

Ces résultats permettent de faire une première hypothèse sur 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 sous forme logicielle.

L'estimation de mouvement (EM) est un bloc qui nécessite, quant à lui, une grande flexibilité en raison du nombre important de techniques d'estimation du mouvement existantes. Cependant, cette estimation passe toujours par un calcul de distance qui est généralement une somme de valeurs absolues de différences (SAD) et qui correspond à l'opérateur le plus largement utilisé lors du codage. Même si une grande partie de l'algorithme d'estimation de mouvement doit être réalisée en logiciel afin de lui conférer la flexibilité nécessaire à l'intégration de nouvelles options, la spécification de l'opérateur de calcul de la somme des valeurs absolues des différences reste quant à elle très stable. En fait, ce traitement est le plus coûteux et de plus, il se prête bien à une réalisation matérielle du fait de la possibilité de paralléliser les calculs. Sans aller jusqu'à une réalisation d'un estimateur de mouvement entièrement câblé qui serait coûteuse et empêcherait toute modification de la technique d'estimation de mouvement, il est intéressant de réaliser l'opérateur de calcul de la somme des valeurs absolues des différences en matériel.

D'autre part, la Transformée en Cosinus Discrète et sa transformée inverse (TCD/TCDI) sont, quant à elles, généralement entièrement câblées dans les systèmes visant les applications vidéo et constituent par conséquent des excellents candidats à une réalisation matérielle sous forme d'un bloc IP. Par ailleurs, vue la bonne régularité de l'équation de la quantification/quantification inverse (Q/QI), elles peuvent être implantées sous forme matérielle en utilisant les instructions personnalisées du processeur NIOS II. Enfin le codage entropique et le reste de codage (contrôle du système) ne requièrent pas beaucoup de ressources et sont à priori réalisables par logiciel.

IV.4 Estimation de mouvement :

IV.4.1 Algorithme de recherche exhaustive :

Le plus primitif des BMAs est l'algorithme de recherche exhaustive FSBM (*Full Search Block Matching*) qui consiste à parcourir de manière complète l'ensemble de la fenêtre de recherche. En procédant de cette manière, le bloc retourné par l'algorithme sera celui qui minimise le critère de comparaison. L'organigramme de la figure 70 présente les différentes étapes à suivre pour obtenir le minimum de SAD et du vecteur de mouvement. Dans cet organigramme, *centre_x* et *centre_y* présentent les coordonnées de l'origine du bloc.

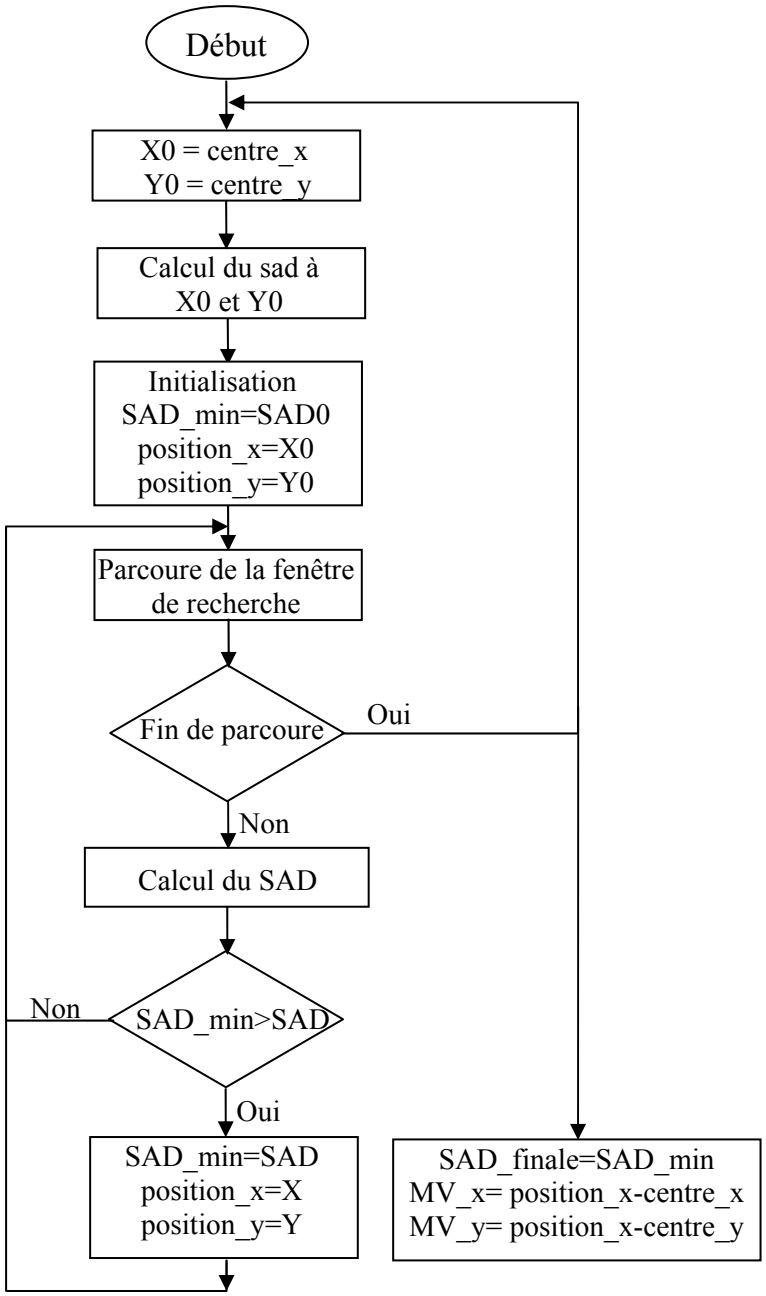


Figure 70. Organigramme de l'algorithme FSBM

D'après la figure 70, on constate que le FSBM permet de sélectionner l'optimum des vecteurs de mouvements parmi tous les vecteurs possibles à l'intérieur de la fenêtre de recherche ce qui correspond à $(2p+1)^2$ calculs de distorsion (vecteur mouvement $\in$ [-p, p]). Par exemple, pour une image QCIF (176x144 pixels) et p=15, une recherche exhaustive des vecteurs de mouvements nécessite donc le calcul de 77439 distorsions. Un calcul de distorsion faisant appel à 768 opérations élémentaires (addition, soustraction, valeur absolue), la recherche des vecteurs mouvements d'une image nécessite environ 60 millions d'opérations. Ce nombre élevé d'opérations rend cette approche inadaptée pour la plupart des applications Temps Réel d'où l'intérêt d'utiliser un accélérateur matériel pour le calcul de SAD afin de paralléliser le traitement des données et minimiser le temps mis pour effectuer le calcul [91] [92].

IV.4.2 Réalisation d'un coprocesseur pour le calcul du SAD :

Les algorithmes de recherche de vecteurs de mouvement utilisent la somme des différences absolues (SAD) en tant que mesure de distorsion. On calcule la somme SAD entre tous les pixels de luminance des MBs candidats et des MBs cibles. La fonction de SAD est donnée par l'équation 5. Cette métrique est très simple et facile à calculer. En plus, elle peut être facilement intégrée par un système de compression vidéo.

$$SAD(x,y) = \sum_{j=0}^{15}\sum_{i=0}^{15} \left| Ycur(i,j) - Yprev(x+i, y+j) \right| \qquad \textbf{Eq : 5}$$

IV.4.2.1 Spécifications du coprocesseur SAD :

D'après l'équation 5, le coprocesseur qui sera réalisé pour le calcul de SAD, doit effectuer la somme des valeurs absolues de différences entre un MB de l'image courante et un MB appartenant à une fenêtre de recherche de taille MxN pixels d'une image de référence. Les pixels correspondant à la luminance de l'image sont codés sur 8 bits. Le principe de fonctionnement de notre coprocesseur est présenté par l'organigramme de la figure 71.

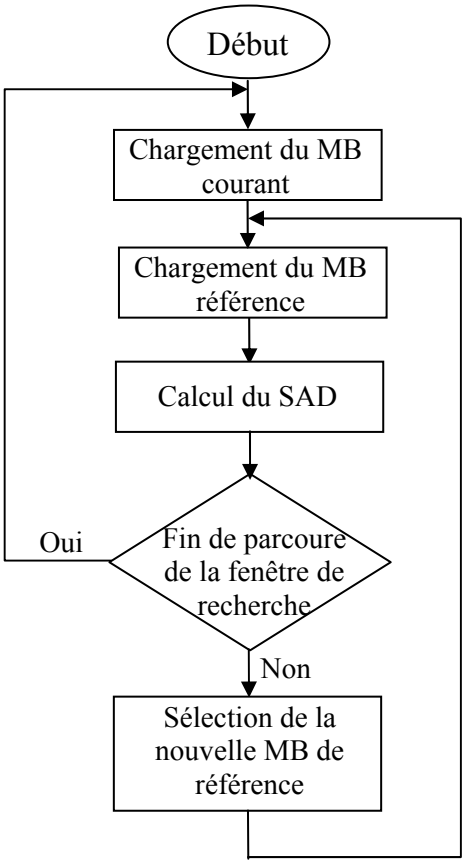


Figure 71. Principe de fonctionnement du coprocesseur SAD

Au début, l'estimation de mouvement place la fenêtre de recherche à l'origine du MB courant puis elle sélectionne le MB de référence. Le coprocesseur charge les données issues du MB courant et de référence, puis il effectue le calcul de SAD. L'étape suivante consiste à vérifier si le critère d'arrêt est activé, dans ce cas il retourne à l'état initial, sinon, il charge le nouvel MB de référence (pour le même MB courant) puis il effectue le calcul de SAD et ainsi de suite. L'entité du coprocesseur est donnée par la figure 72.

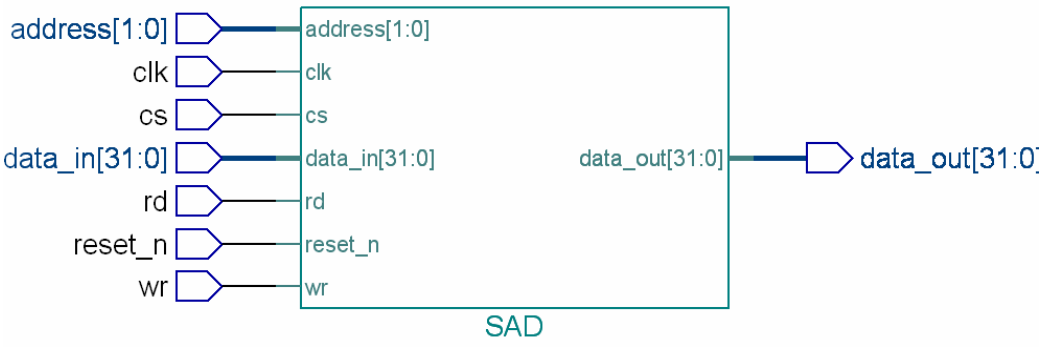


Figure 72. L'entité du coprocesseur SAD

La table 12 énumère les signaux utilisés par le coprocesseur avec leurs descriptions.

Table 12. Signaux entre le coprocesseur SAD et le bus Avalon

Signal	Largeur	Direction	Description
Clk	1	In	Horloge du système
Reset_n	1	In	Reset actif à « 0 »
address	2	In	Adresse des registres du coprocesseur
cs	1	In	Sélection du périphérique
rd	1	In	Signal de lecture actif à « 1 »
wr	1	In	Signal d'écriture actif à « 1 »
data_in	32	In	Données 32 bits lues du bus Avalon
data_out	32	Out	Données 32 bits envoyées vers le bus Avalon

La liaison entre les parties matérielle et logicielle se fait à travers le fichier *system.h* (Annexe H) qui contient l'adresse des registres de notre bloc IP. Le transfert de données entre le coprocesseur et la mémoire se fait à travers le bus Avalon en utilisant les registres *np_SAD_COEFFIN* pour l'écriture des données et *np_SAD_COEFFOUT* pour la lecture des données du coprocesseur. Les pixels sont concaténés sur 32 bits afin d'utiliser la totalité du bus et présentés à l'entrée du coprocesseur (signal *data_in*). Le calcul de SAD est effectué après le chargement du MB courant et de référence. La figure 73 représente un exemple de lecture et écriture des pixels du coprocesseur SAD.

```
//Déclaration des différents registres dans un fichier *.h
typedef volatile struct
{
  int               np_SAD_STATUS;
  int               np_SAD_CONTROL;
  int               np_SAD_COEFFIN;
  int               np_SAD_COEFFOUT;
}
 np_sad;

//déclaration d'une variable qui pointe sur l'adresse du coprocesseur SAD
np_sad *sad=(np_sad *) na_sad0;

//écriture des pixels dans le coprocesseur
sad->np_SAD_COEFFIN=(*curr)|((*(curr+1))<<8)|((*(curr+2))<<16)|((*(curr+3))<<24);

//lecture de la valeur du SAD calculé
sad_val=sad->np_SAD_COEFFOUT;
```

Figure 73. Lecture et écriture des données du coprocesseur SAD

IV.4.2.2 Architecture interne du coprocesseur SAD :

Le principal composant de notre coprocesseur est l'Unité de Traitement (UT). L'architecture de cette unité est donnée par figure 74. Elle est constituée de registres, d'une unité pour effectuer l'addition/soustraction et d'une unité de calcul de la valeur absolue. Les données 32 bits sont présentés à travers les ports *data_A* et *data_B* à l'entrée de l'unité. Cette unité contient 4 modules à deux entrées chacune sur 8 bits permettant de calculer en parallèle et en un cycle d'horloge la valeur absolue de différence.

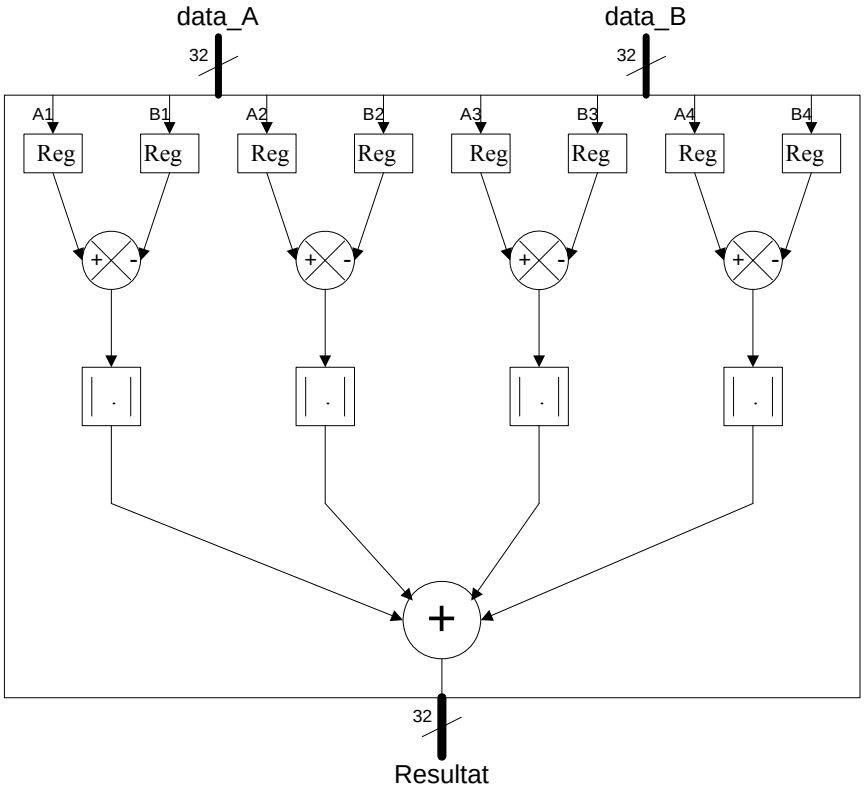


Figure 74. Architecture de l'Unité de Traitement (UT)

Le calcul de la somme des valeurs absolues de différence d'une ligne de MB (16 pixels) se fait en appliquant en parallèle 4 unités de traitement comme le montre la figure 75, ce qui permet de calculer toute la ligne du MB en un seul cycle d'horloge. Cette opération est suivie par l'addition des résultats obtenus par les différentes unités de traitements.

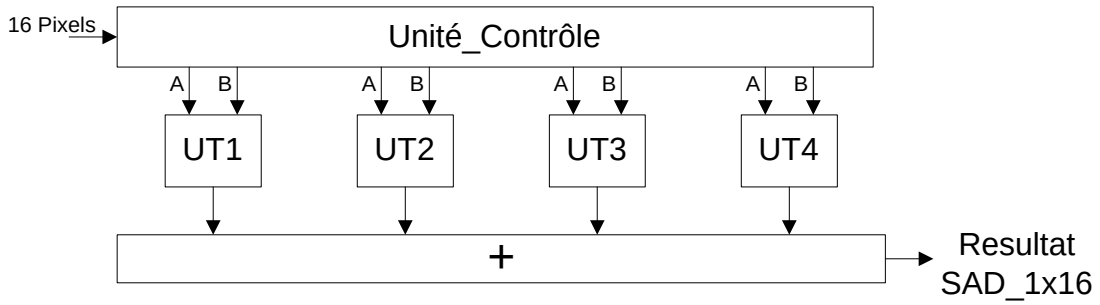


Figure 75. Calcul du SAD 1x16

Le calcul du SAD 16x16 se fait en utilisant le module SAD1x16. En fait, deux méthodes sont possibles lors de l'appel de ce module. La première méthode fait séquentiellement l'appel du module SAD 1x16 (à chaque cycle d'horloge). Dans ce cas, le résultat de calcul du SAD des 16 lignes du MB est obtenu dans 16 cycles d'horloge après le chargement du MB courant et de référence. Cette méthode est utilisée si on veut optimiser en surface de FPGA. Par contre, pour optimiser le temps de traitement, on utilise le module SAD1x16 en parallèle comme le montre la figure 76. Dans ce cas, la valeur du SAD est obtenue avec la dernière valeur du MB de référence introduit puisque tout le calcul est effectué en logique combinatoire.

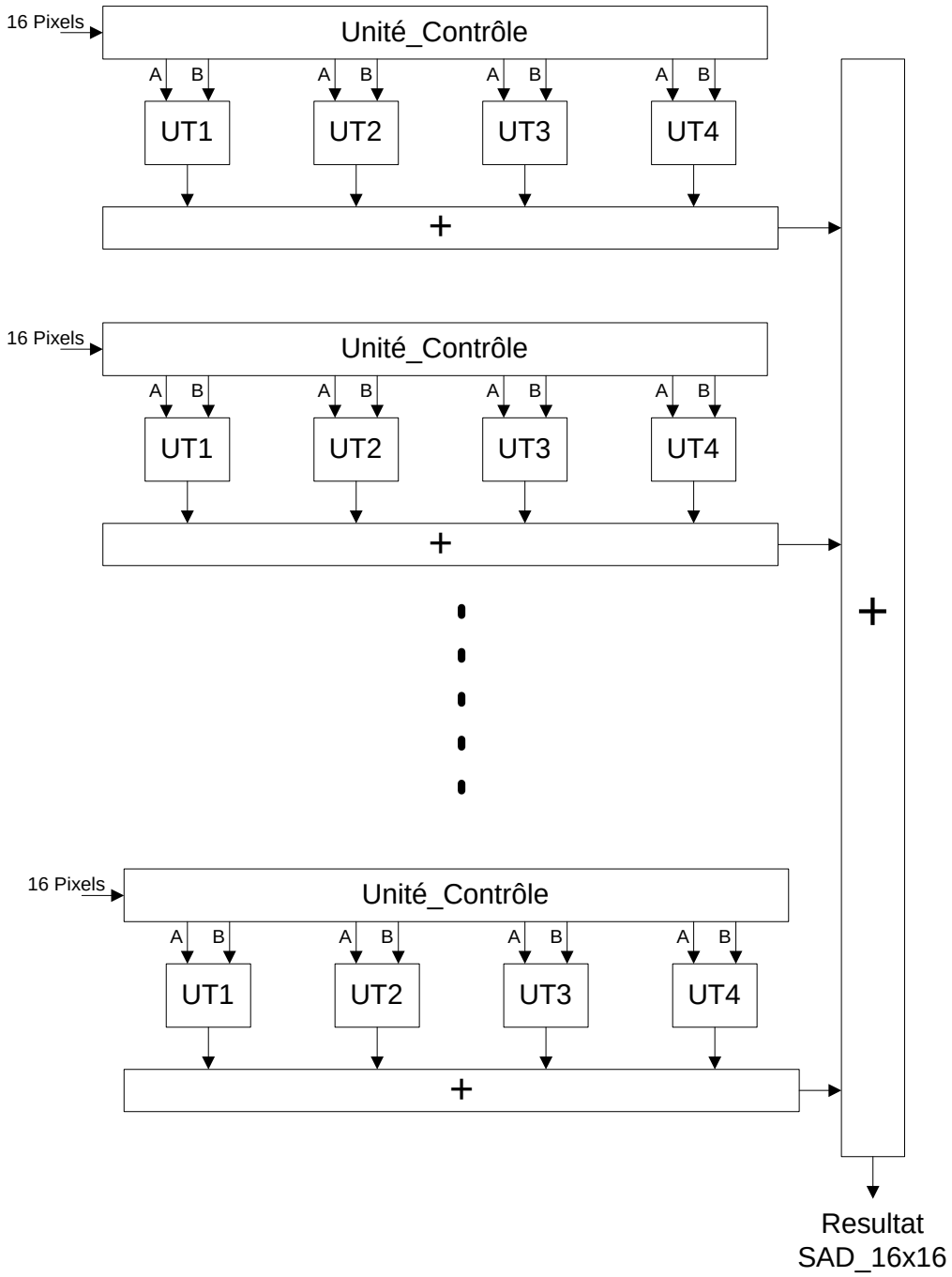


Figure 76. Architecture pour le calcul du SAD 16x16

IV.4.2.3 Résultats d'implantation :

La synthèse par Quartus sur FPGA Stratix II EP2S60 a donné les résultats suivants :

- ALUTs : 8,445 / 48,352 (17 %).
- Pins E/S : 71 / 493 (14 %).
- Blocs RAMs : 0 / 2,544,192 (0 %).
- Blocs DSPs : 0/288 (0%).
- Fréquence maximale : 150 MHz

Les résultats suivants montrent que la fréquence maximale générée par Quartus est de l'ordre de 150 MHz. Cette fréquence est confirmée par la simulation temporelle (simulation post/routage) qui est égal à 142 MHz.

IV.4.2.4 Evaluation des performances du coprocesseur SAD :

Afin d'évaluer les performances du coprocesseur SAD, on l'a intégré avec l'ensemble du design puis on a mesuré le temps de traitement. La figure 77 illustre le nombre de cycles nécessaires pour l'estimation de mouvement en logiciel (SAD_SW) et matériel (SAD_HW) en utilisant les séquences de test Miss America et Foreman avec QP=13. D'après cette figure, on constate que la solution matérielle pour le calcul de SAD est 2 fois plus rapide que la solution logicielle.

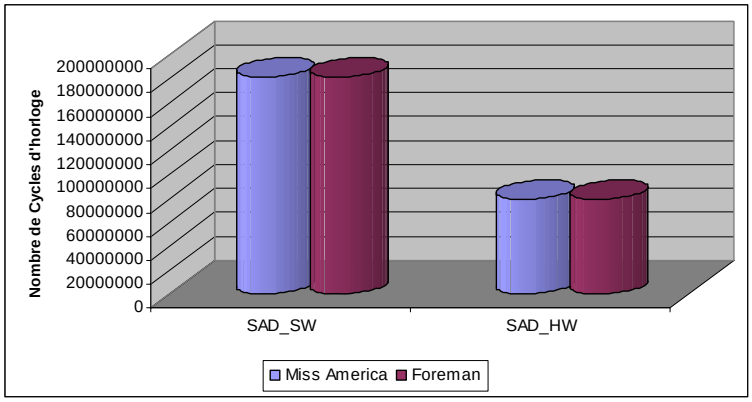


Figure 77. Nombre de cycles pour le calcul de SAD en SW et HW

Bien que l'on ait utilisé un accélérateur matériel pour le calcul du SAD dans le cas de la recherche exhaustive, on constate que le temps alloué à l'estimation reste encore élevé (675 ms au lieu de 1512 ms dans le cas de la séquence Miss America). Cela est dû à l'accès mémoire intensif, d'où l'intérêt d'utiliser des algorithmes d'estimation de mouvement rapides pour minimiser les points de recherche. Dans le prochain paragraphe, on va étudier les différents algorithmes de recherche les plus utilisés en tenant compte de la qualité de l'image et de la rapidité de recherche.

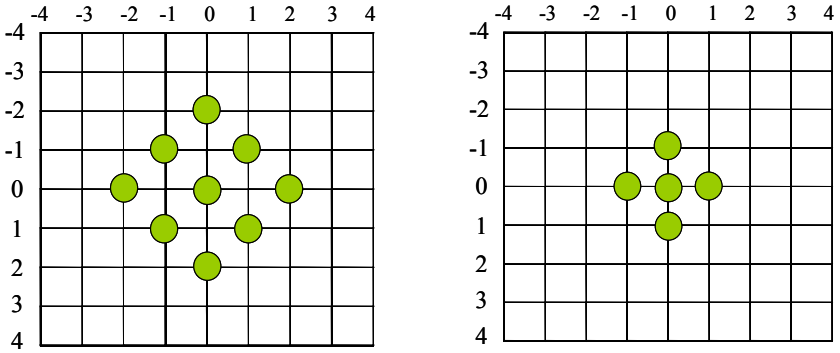
IV.4.3 Accélération de la recherche des vecteurs de mouvement :

Pour les algorithmes d'estimation de mouvement rapide, une hypothèse supplémentaire est faite. La variation de la distorsion dans la fenêtre de recherche est monotone, c'est-à-dire que la distorsion est supposée croissante dans toutes les directions à partir du minimum global. Ces algorithmes se terminent lorsqu'une frontière de la fenêtre de recherche est atteinte ou lorsque le vecteur courant satisfait à un certain critère de convergence.

Quelques exemples de réalisations sont présentés : la recherche en diamant [93], la recherche en hexagone [94] et la recherche en croix diamant [95]. Mais de nombreuses variantes de ces méthodes existent dans la littérature.

IV.4.3.1 Algorithme de recherche en diamant *Diamond Search* :

L'algorithme *Diamond Search* (DS) [93] utilise deux modèles de recherche illustrés dans la figure 78. Le premier modèle appelé *Large Diamond Search Pattern* (LDSP) comprend neuf points dont huit points sur le bord du diamant à une distance de deux pixels par rapport au centre. Le neuvième point est situé au centre pour composer une forme de diamant. Le second modèle nommé *Small Diamond Search Pattern* (SDSP) contient cinq points dont quatre sont sur le bord situés à une distance d'un pixel par rapport au centre. Le cinquième est situé au centre.



(a) *Large Diamond Search Pattern* (b) *Small Diamond Search Patten*

Figure 78. Modèles de recherche utilisés dans l'algorithme *Diamond Search*

L'algorithme de recherche en diamant se résume de la manière suivante :

- **Première étape** : Le LDSP est centré sur l'origine du bloc courant et les neuf blocs sont testés. Si le bloc minimisant la distorsion est celui du centre, l'algorithme passe directement à la troisième étape. Sinon, il continue avec la seconde.
- **Deuxième étape** : L'origine du bloc minimisant le critère calculé précédemment est repositionnée pour être le centre d'un nouveau LDSP. Les blocs du modèle sont alors testés successivement. Si le nouveau bloc minimisant le critère est le bloc central, on continue avec la troisième étape. Sinon, on répète la deuxième étape.
- **Troisième étape** : On échange le modèle LDSP avec le modèle SDSP et on teste les blocs. Le bloc obtenu lors de cette étape représente la solution finale pour obtenir le vecteur mouvement du bloc.

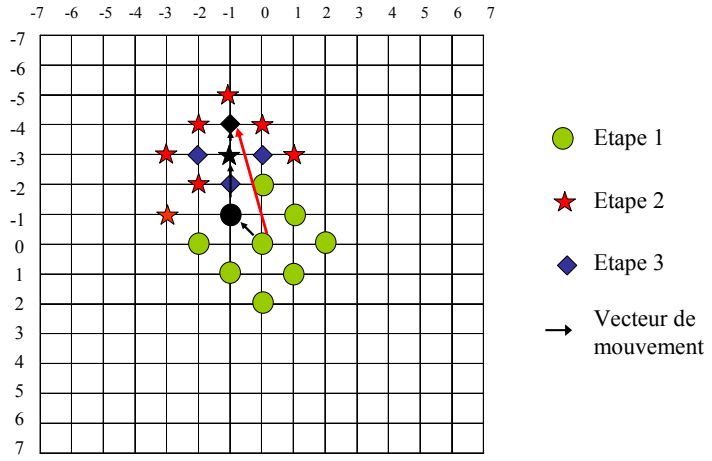
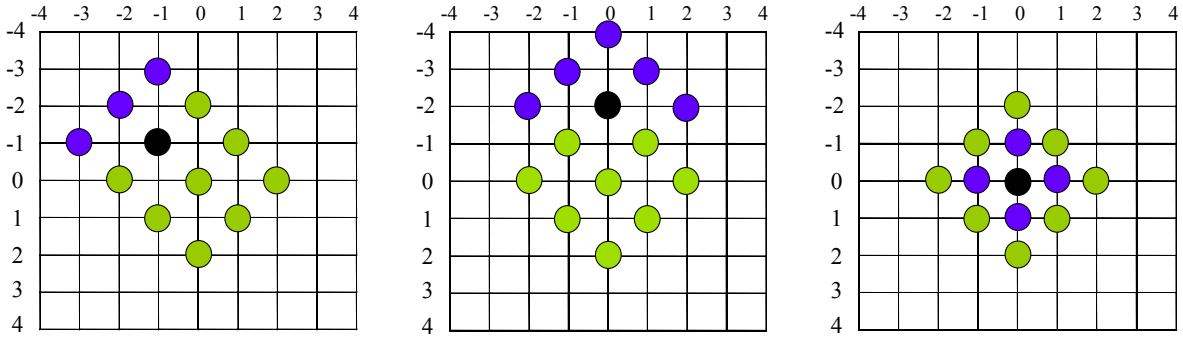


Figure 79. Exemple d'une recherche en diamant

La figure 79 illustre le déroulement de l'algorithme en indiquant la différence entre les trois étapes. Lors de l'appel récursif de la deuxième étape, nous pouvons remarquer que les modèles LDSP se recouvrent partiellement. Ainsi, dans le but d'optimisation, nous considérons que lors de cette étape, trois cas sont possibles et représentés sur la figure 80. Dans le premier cas (figure 80 (a)), le bloc minimisant le critère se trouve sur le bord du LDSP. Pour compléter le modèle lors de cette étape, il est nécessaire de tester seulement trois blocs placés comme l'indique le schéma. De même, dans le deuxième cas (figure 80 (b)), le bloc se trouve dans un coin du diamant et seuls cinq blocs doivent être testés. Enfin, dans le dernier cas (figure 80 (c)), les quatre blocs de la troisième étape de l'algorithme sont testés comme expliqué précédemment.



(a) bloc sur le bord (b) bloc dans un coin (c) bloc au centre

Figure 80. Optimisation de l'algorithme *Diamond Search*

IV.4.3.2 Algorithme de recherche en petit diamant *Small Diamond Search* :

L'algorithme *Small Diamond Search* (SDS) utilise la même méthode de recherche que la recherche en diamant sauf que la forme de recherche est *Small Diamond* au lieu d'une forme en diamant classique. Cela permet une recherche précise pour les faibles mouvements. La figure 81 illustre le déroulement de l'algorithme en indiquant la différence entre les différentes étapes. On arrête la recherche si l'on est au bord de la fenêtre de recherche ou si la distorsion minimale se trouve au centre de la modèle de recherche.

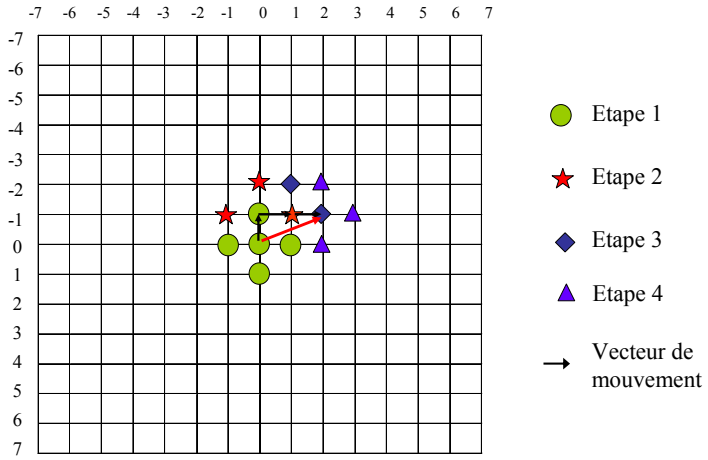
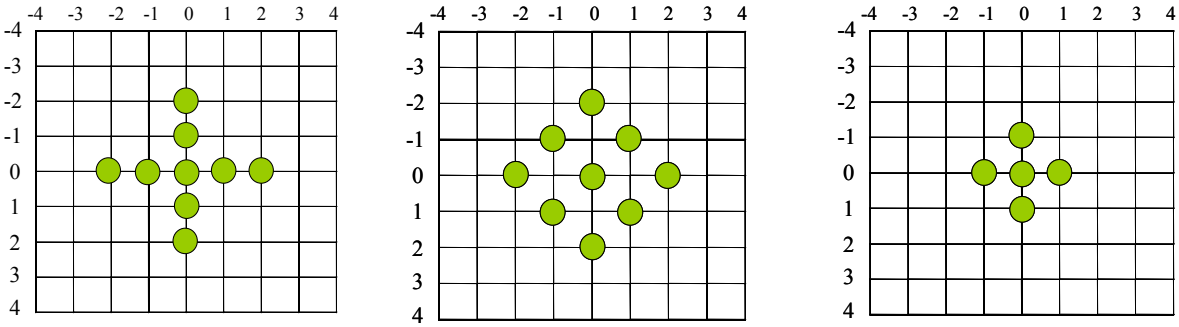


Figure 81. Exemple d'une recherche en petit diamant

IV.4.3.3 Algorithme de recherche en croix diamant *Cross-Diamond Search* :

L'algorithme *Cross-Diamond Search* (CDS) [95] est variante de *Diamond Search* utilise trois modèles de recherche illustrés par la figure 82. Le premier modèle appelé *Cross Search pattern* (CSP) comprend neuf points dont le neuvième point est situé au centre de la croix. Le deuxième modèle appelé *Large Diamond Search Pattern* (LDSP) comprend neuf points dont huit points sur le bord du diamant à une distance de deux pixels par rapport au centre. Le neuvième point est situé au centre pour composer une forme de diamant. Le troisième modèle nommé *Small Diamond Search Pattern* (SDSP) contient cinq points dont quatre sont sur le bord situés à une distance d'un pixel par rapport au centre. Le cinquième est situé au centre.



(a) *Cross Search Pattern* (b) *Large Diamond Search Pattern* (c) *Small Diamond Search Pattern*

Figure 82. Modèles de recherches utilisées dans l'algorithme *Cross-Diamond Search*

L'algorithme de recherche en croix diamant se résume en 4 étapes :

- **Première étape** : Le CSP est centré sur l'origine du bloc courant et les neuf blocs sont testés. Si le bloc minimisant la distorsion est celui du centre, alors cette étape représente la solution finale pour obtenir le vecteur mouvement du bloc. Sinon, l'algorithme continue avec la seconde.
- **Deuxième étape** : Dans cette étape quatre blocs sont ajoutés au modèle CSP pour former un LDSP qui est centré sur l'origine du bloc courant et les quatre blocs sont testés. Le bloc minimisant la distorsion est choisi le centre d'un nouveau LDSP pour la troisième étape.
- **Troisième étape** : L'origine du bloc minimisant le critère calculé précédemment est repositionnée pour être le centre d'un nouveau LDSP. Les blocs du modèle sont alors testés successivement. Si le nouveau bloc minimisant le critère est le bloc central, on

continue avec la quatrième étape. Sinon, on répète la troisième étape.

- **Quatrième étape** : On échange le modèle LDSP avec le modèle SDSP et on teste les blocs. Le bloc obtenu lors de cette étape représente la solution finale pour obtenir le vecteur mouvement du bloc.

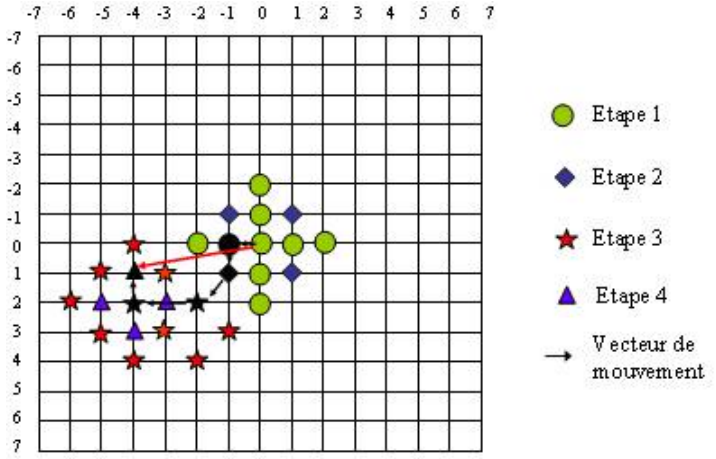
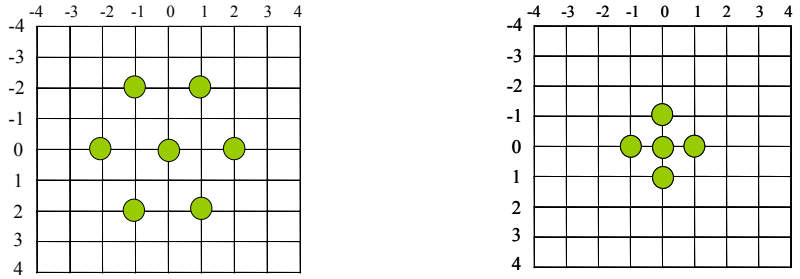


Figure 83. Exemple d'une recherche en croix diamant

La figure 83 illustre le déroulement de l'algorithme en indiquant la différence entre les quatre étapes. On constate qu'à partir de la deuxième étape l'algorithme devient similaire au DS.

IV.4.3.4 Algorithme de recherche en hexagone *Hexagonal Search* :

L'algorithme *Hexagonal Search* (HEX) présenté dans [94] tente à proposer une alternative à l'algorithme DS tout en gardant la même philosophie. En effet, l'algorithme utilise, comme dans le DS, deux modèles de recherche illustrés dans la figure 84 : un modèle large (*Large Hexagonal Search Pattern* : LHSP) et un modèle petit (*Small Hexagonal Search Pattern* : SHSP). La première remarque que l'on peut faire est que le SHSP est identique au SDSP. Deuxièmement, le LHSP contient sept points alors que le LDSP en contient neuf. Cela permet d'entrevoir la diminution du nombre de points testés par l'algorithme hexagonal.



(a) *Large Hexagonal Search Pattern* (b) *Small Hexagonal Search Pattern*

Figure 84. Modèles de recherche utilisés dans l'algorithme *Hexagonal Search*

Enfin, nous pouvons noter que l'hexagone permet de recouvrir l'espace de manière optimale. L'algorithme HEX, dont un exemple de déroulement est donné par la figure 85, est en tout point similaire au DS mis à part l'utilisation des modèles de recherche LHSP et SHSP à la place des modèles LDSP et SDSP.

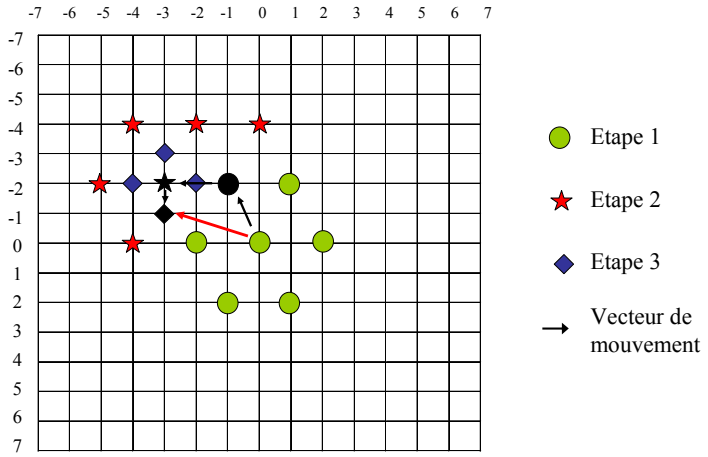


Figure 85. Exemple d'une recherche en hexagone

On retrouve également dans le HEX, l'optimisation présenté dans le DS. Nous pouvons voir sur la figure 86 que, quel que soit l'emplacement du bloc minimisant le critère, seuls trois blocs devront être testés pour l'itération suivante. Ceci représente un avantage par rapport à l'algorithme DS au niveau de l'implémentation car il n'y a pas de cas particulier, tout est géré identiquement.

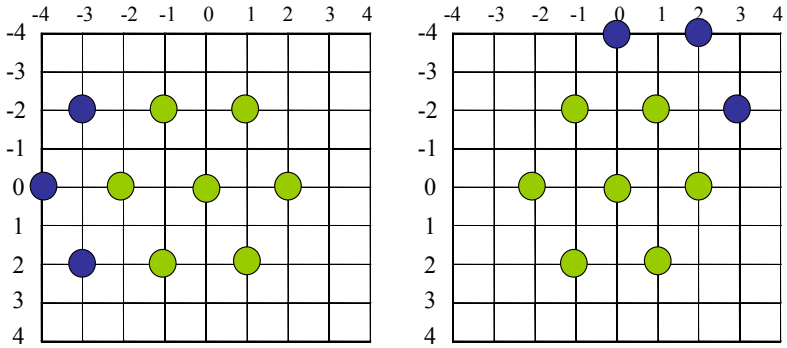


Figure 86. Optimisation de l'algorithme *Hexagonal Search*

IV.4.4 Evaluation des performances en fonction de l'estimation de mouvement :

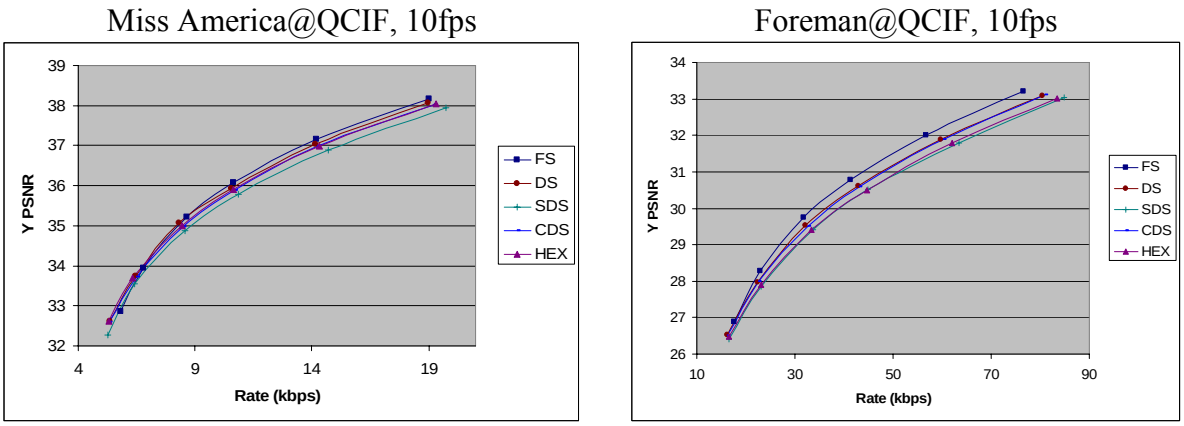
Le choix de la méthode de *Block Matching* est essentiel. La qualité et la rapidité de l'estimation de mouvement en dépendent. On présente dans cette partie l'évaluation des différents algorithmes présentés. L'évaluation de performance est réalisée sur les séquences Miss America et Foreman codées à 10 Hz avec différentes valeurs de quantification (QP=8, 13, 31). Les estimations du taux d'occupation du processeur en fonction des différents traitements composant le codage vidéo pour les différents algorithmes d'estimation de mouvement sont données par la table 13. Ces mesures sont faites pour QP=13. D'autres mesures (QP=8 et 31) sont fournies en annexe I.

Table 13. Pourcentage du temps CPU en fonction du traitement

	EM/CM	TCD/TCDI	Q/QI	CLV	Autres	fps
Miss America						
FS	60,76	36,91	1,18	0,09	1,06	0,40
DS	3,76	89,75	2,84	0,23	3,41	0,96
SDS	2,52	91,63	2,86	0,25	2,93	0,96
CDS	3,45	91,26	2,89	0,24	2,16	0,97
HEX	3,36	90,37	2,86	0,24	3,18	0,96
Foreman						
FS	54,34	43,29	1,16	0,27	0,93	0,36
DS	3,93	91,54	2,37	0,60	1,57	0,73
SDS	3,11	92,06	2,34	0,63	1,68	0,71
CDS	4,06	90,76	2,34	0,59	2,25	0,72
HEX	3,08	92,31	2,38	0,61	1,61	0,72

D'après cette table, on voit très clairement que l'utilisation d'une technique de recherche rapide des vecteurs de mouvement réduit fortement la part de temps passé dans l'EM (de 60% du temps CPU pour la recherche exhaustive à environ 4% pour les méthodes rapides). En fait, l'utilisation de ces techniques de recherche entraîne une réduction des accès mémoire et du nombre d'appels à l'opérateur de distorsion SAD.

Passons maintenant à l'analyse de la qualité du codage en fonction de l'estimation de mouvement. La figure 87 illustre la variation du PSNR en fonction du débit visé pour les séquences Miss America et Foreman (les tables de mesure de qualité pour ces deux séquences ainsi que d'autres séquences sont fournies en annexe J). Si l'on classe les différentes méthodes de *Block Matching* selon le PSNR, on obtient par ordre décroissant : *Full Search*, *Diamond Search*, *Cross Diamond Search*, *Hexagonal Search* et *Small Diamond Search*.
Il est évident que toute méthode rapide ne peut pas dépasser qualitativement la méthode exhaustive.

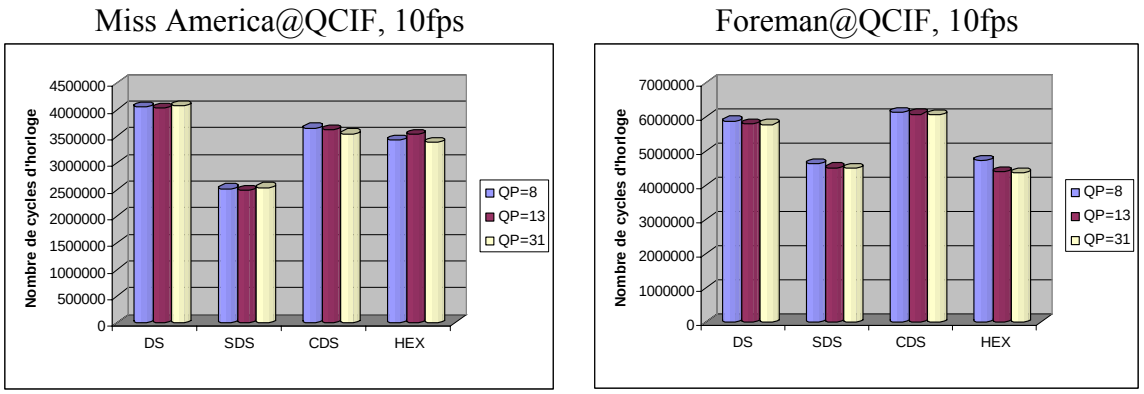


(a) (b)

Figure 87. Qualité de différents algorithmes d'EM pour les séquences
(a) Miss America et (b) Foreman

Nous étudierons maintenant la rapidité de recherche. La figure 88 illustre le nombre de cycles d'horloge nécessaire pour chaque algorithme de recherche avec différents pas de quantification (QP=8, 13, 31). D'après cette figure, on remarque que la méthode de recherche la plus rapide est SDS mais avec une qualité de PSNR la plus faible et ceci pour les deux

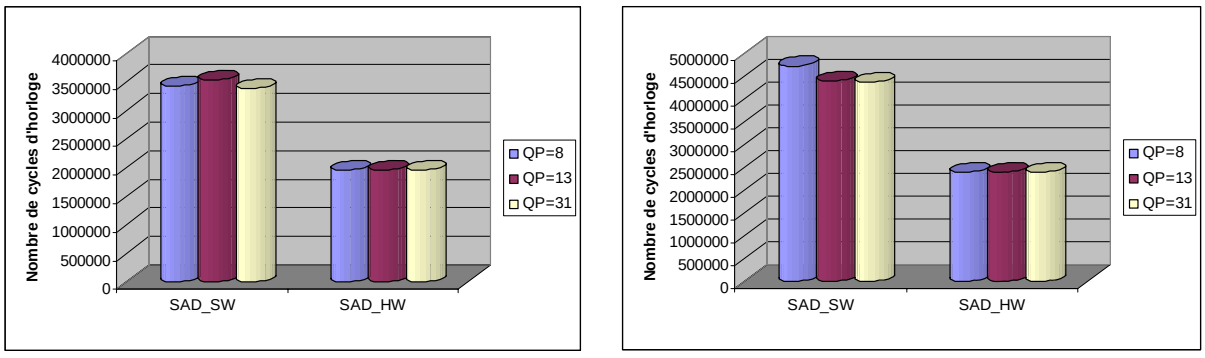
séquences. Par contre, la recherche en DS et CDS nécessite un nombre de cycles d'horloge le plus important avec une qualité de PSNR la plus élevée. En effet, si l'on cherche un compromis entre la rapidité de recherche et la qualité d'image, on peut choisir la méthode de recherche en hexagone qui permet une qualité d'image et rapidité de recherche acceptable [97]. D'après ces résultats on constate que la valeur de complexité d'une recherche en hexagone est 50 fois plus faible que la complexité d'une recherche exhaustive dans le cas de Miss America (40 fois dans le cas Foreman).



(a) (b)

Figure 88. Nombre de cycles d'horloge des différents algorithmes d'EM pour les séquences (a) Miss America et (b) Foreman

Afin d'accélérer la recherche du vecteur de mouvement par la méthode de recherche en hexagone, le calcul de distorsion a été effectué par matériel en utilisant le coprocesseur SAD. La figure 89 et la table 14 montrent qu'en effectuant le calcul de distorsion par matériel, nous avons une accélération de l'ordre de 2 par rapport à la solution logicielle pour une même qualité d'image et ceci pour les séquences Miss America et Foreman avec différentes valeurs de pas de quantification (QP=8, 13, 31).



(a) (b)

Figure 89. Nombre de cycles pour le calcul du SAD par SW et HW pour la recherche en hexagone des séquences (a) Miss America et (b) Foreman

Table 14. Évaluation du coprocesseur SAD pour la recherche en hexagone des séquences Miss America et Foreman

	Miss America			Foreman		
QP	**8**	**13**	**31**	**8**	**13**	**31**
SAD_SW						
Cycles d'horloge	3425290	3536683	3383716	4721319	4398578	4374555
PSNR-Y(dB)	38,03	35,9	32,61	33,01	30,5	26,48
SAD_HW						
Cycles d'horloge	1997903	1996480	1986629	2403927	2401108	2468723
PSNR-Y(dB)	38,03	35,9	32,61	33,01	30,5	26,48
Accélération						
Gain	1,7	1,8	1,7	2	1,9	1,8

IV.5 Implantation matérielle de la TCD/TCDI :

Vue l'importance de la TCD/TCDI [98] dans le domaine des applications vidéo, plusieurs algorithmes et architectures ont été proposés pour une implantation efficace de la TCD/TCDI [99] [100] [101] [102]. En général, la base de comparaison entre les algorithmes de la TCD/TCDI est le nombre de multiplications et d'additions. La table 15 illustre les algorithmes de la TCD/TCDI les plus connus.

Table 15. Algorithmes de la TCD/TCDI

	Chen [103]	**Wagh [104]**	**Lee [105]**	**Malvar [106]**	**Chan [107]**	**Loeffler [108]**
MUL	16 (13)	14	12	12	12	11
ADD	26 (29)	32	29	31	29	29

On constate d'après cette table que l'algorithme de Chen utilise 16 MUL (multiplication) et 26 ADD (addition) qui peuvent être optimisés à 13 MUL et 29 ADD. Par contre l'algorithme le plus optimisé est l'algorithme de Loeffler qui utilise 11 MUL et 29 ADD. Cet algorithme a été retenu pour l'implantation de la TCD/TCDI du fait du nombre minimal d'additions et de multiplications. Dans ce paragraphe, on va étudier les différentes architectures qui peuvent être utilisées pour l'implantation de la TCD/TCDI.

IV.5.1 Architecture de Loeffler :

IV.5.1.1 Etude de l'architecture :

Loeffler a montré qu'il existe une classe d'algorithme de la TCD/TCDI qu'utilise 11 multiplications et 29 additions seulement en se basant sur l'équation de la TCD (équation 2 du paragraphe I.4.4). La figure 90 illustre ce type d'algorithme.

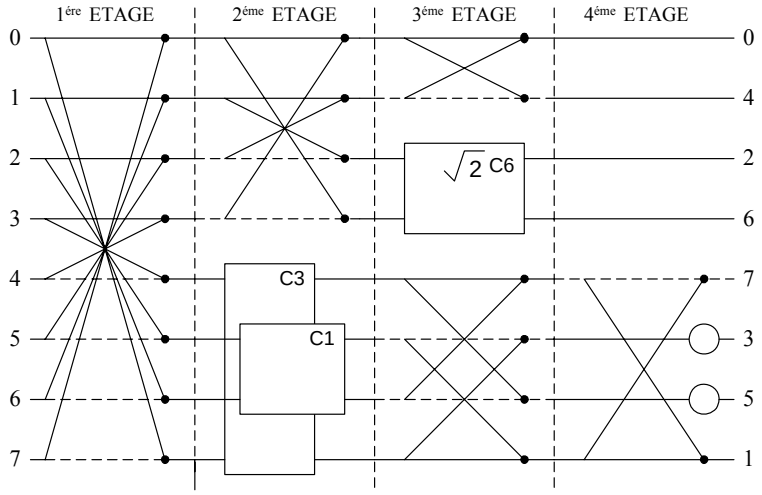


Figure 90. Architecture de Loeffler

Les étages de l'algorithme numérotés de 1 à 4 doivent être exécutés en série et ne peuvent pas être évaluées en parallèle à cause de la dépendance des données. Au niveau l'étage 2, l'algorithme est séparé en deux parties, une partie paire et une autre impaire. La partie paire est une simple TCD d'ordre 4 qui doit être partagée en deux parties paire et impaire au niveau de l'étage 3. La table 16 illustre les différents types des symboles qui constituent les blocs de l'algorithme, tout en précisant leurs interprétations et l'effort de calcul correspond.

Table 16. Différents symboles utilisés dans l'architecture de Loeffler

Symbole	Equation	Nbr opérations
I_0 — O_0 I_1 — O_1 Circuit papillon	$O_0 = I_0 + I_1$ $O_1 = I_0 - I_1$	1 ADD
I_0, I_1 — kcn — O_0, O_1 Boîtier de Loeffler	$O_0 = I_0.K.Cos\frac{n\Pi}{2N} + I_1.k.Sin\frac{n\Pi}{2N}$ $O_1 = -I_0.k.Sin\frac{n\Pi}{2N} + I_1.k.Cos\frac{n\Pi}{2N}$	2 ADD 4 MUL
I — ○ — O Multiplication par $\sqrt{2}$	$O = \sqrt{2}.I$	1 MUL
I — ● — O Multiplication par $1/\sqrt{2}$	$O = \frac{1}{\sqrt{2}}.I$	1 MUL

Les équations du deuxième symbole, nommé boîtier de Loeffler, peuvent être effectuées en utilisant 3 additions et 3 multiplications seulement au lieu de 4 multiplications et 2 additions. Cela est possible si on applique l'équivalence montrée dans les équations 6 et 7.

$$y_0 = a.x_0 + b.x_1 = (b-a).x_1 + a.(x_0 + x_1) \qquad \textbf{Eq : 6}$$

$$y_1 = -b.x_0 + a.x_1 = -(b+a).x_0 + a.(x_0 + x_1) \qquad \textbf{Eq : 7}$$

Afin d'augmenter l'exactitude du calcul lors de l'implantation matérielle de l'architecture de Loeffler, on a modifié dans la figure 90 le placement du coefficient $\sqrt{2}$ comme le montre la figure 91. En effet, au lieu de multiplier par $\sqrt{2}$, on fait une multiplication par $1/\sqrt{2}$. Cette méthode évite le problème de débordement qui pourrait présenter une perte d'exactitude du calcul [109] [110]. La figure 91 présente l'architecture modifiée de Loeffler. Les coefficients utilisés par l'algorithme de Loeffler pour la TCD/TCDI sont fournis en annexe K.

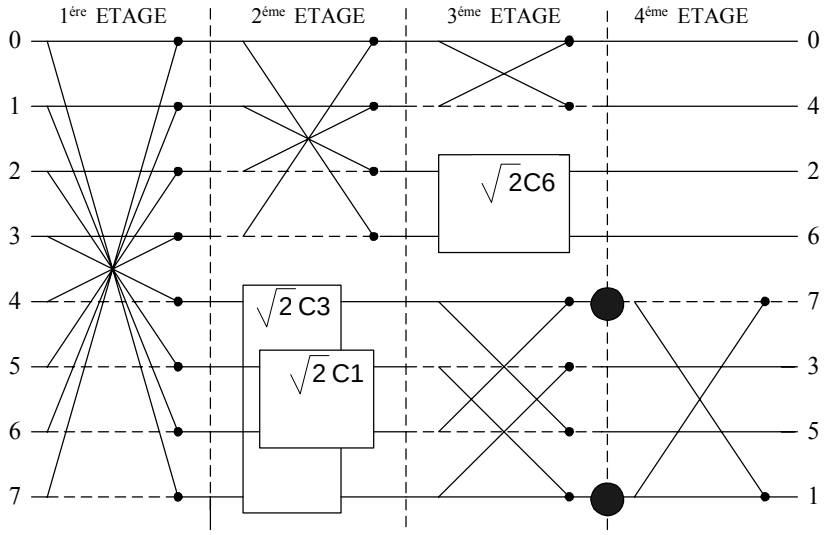


Figure 91. Architecture modifiée de l'algorithme de Loeffler

Les résultats d'implantation de la TCD/TCDI unidimensionnelle (TCD-1D) en utilisant l'architecture de Loeffler sur circuit FPGA Stratix II EP2S60 sont donnés par la table 17.

Table 17. Résultats d'implantation de la TCD/TCDI

	TCD	**TCDI**
ALUTs	1969 (4%)	1572 (3%)
E/S	195 (40%)	195 (40%)
Blocs RAMs	0%	0%
Blocs DSPs	0%	0%
Fmax (MHz)	70	113

D'après cette table, on voit que la fréquence maximale pour la TCD et la TCDI évaluée par Quartus est de l'ordre de 70 MHz et 113 MHz respectivement. Cette fréquence est confirmée par la simulation temporelle qui est égal à 77 MHz et 111 MHz pour la TCD et TCDI respectivement.

IV.5.1.2 Optimisation de l'architecture de Loeffler :

Le circuit FPGA Stratix II intègre des blocs DSPs pour des applications de traitement de signal. Ces blocs DSPs sont optimisés pour l'implantation des différentes fonctions DSP telles que multiplieurs, accumulateurs, additionneurs avec un maximum de performance et un nombre minimum de portes logiques. Ces blocs peuvent être configurés dans différents largeurs de 9x9 bits à 36x36 bits afin de supporter un large spectre d'application. Dans le but d'augmenter la performance de l'architecture de Loeffler, ces blocs ont été utilisés pour l'implantation de la multiplication. La figure 92 présente un exemple de la réalisation d'une multiplication 14x18 en utilisant le primitive DSP.

```
lpm_mult_component : lpm_mult
      GENERIC MAP (
            lpm_widtha => 14,
            lpm_widthb => 18,
            lpm_widthp => 32,
            lpm_widths => 32,
            lpm_type => "LPM_MULT",
            lpm_representation => "SIGNED",
            lpm_hint => "DEDICATED_MULTIPLIER_CIRCUITRY=YES,MAXIMIZE_SPEED=6"
      )
      PORT MAP (
            dataa => dataa,
            datab => datab,
            result => sub_wire0
      );
```

Figure 92. Exemple d'utilisation de la primitive DSP

Les résultats d'implantation de la TCD/TCDI-1D en utilisant l'architecture de Loeffler avec les blocs DSPs sur FPGA Stratix II EP2S60 sont donnés par la table 18.

Table 18. Résultats d'implantation de la TCD/TCDI avec blocs DSPs

	TCD	**TCDI**
ALUTs	617 (1%)	496 (1%)
E/S	195 (40%)	195 (40%)
Blocs RAMs	0%	0%
Blocs DSPs	22(8%)	22(8%)
Fmax (MHz)	125	120

D'après cette table, on remarque que la fréquence maximale pour la TCD et la TCDI évaluée par Quartus est de l'ordre de 125 MHz et 120 MHz respectivement. Cette fréquence est confirmée par la simulation temporelle qui est égal à 111 MHz pour la TCD et la TCDI. La figure 93 illustre la simulation temporelle de la TCD. D'après cette figure, on voit bien que le principe de pipeline est utilisé dans notre architecture. En fait, la TCD est appliquée sur un bloc de taille 8x8. Les données de chaque ligne du bloc sont présentées à l'entrée de la TCD en parallèle sur 12 bits. Le traitement de la première ligne est effectué en 4 cycles puisque les 4 étages de la TCD sont vides puis après sept cycles, on obtient les valeurs traitées des lignes restantes. Au total, le traitement d'un bloc de taille 8x8 par la TCD-1D nécessite 11 cycles horloge avec l'architecture de Loeffler.

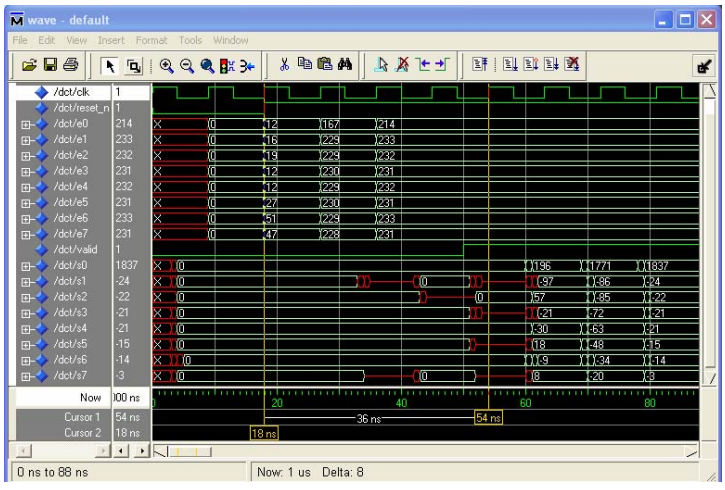


Figure 93. Simulation temporelle de la TCD-1D par l'architecture de Loeffler

IV.5.2 Distribution arithmétique :

L'utilisation de la distribution arithmétique permet d'implanter une somme des produits sans l'utilisation d'une multiplication. Cela est réalisé par le stockage d'un nombre fini de résultats intermédiaires dans une mémoire ROM. Puis, une sommation des produits intermédiaires est réalisée par une succession d'additions et de décalages [111] [112]. Par conséquent, cette technique permet de réduire le nombre de portes logiques nécessaires pour l'intégration. De même, elle assure une structure régulière pour l'intégration des algorithmes numériques basés sur des coefficients constants.

IV.5.2.1 Etude de l'architecture :

La méthode de la distribution arithmétique peut être utilisée pour l'implantation de la TCD/TCDI [113] [114]. En fait, la forme de la TCD-1D est donnée par l'équation suivante :

$$y_l = \frac{c(l)}{2}\sum_{m=0}^{7} x_m \cos\left(\frac{(2m+1)l\pi}{16}\right) \qquad \textbf{Eq : 8}$$

On peut écrire l'équation 8 sous la forme suivante :

$$y_l = \sum_{m=0}^{7} a_m^l x_m$$
$$avec \quad a_m^l = \frac{c(l)}{2}\cos\left(\frac{(2m+1)l\pi}{16}\right) \qquad \textbf{Eq : 9}$$

Utilisant la propriété symétrique du terme a_m^l en *m*, on peut faire seulement la sommation du produit de 4 termes, donc on aura :

$$y_l = \begin{cases} \sum_{m=0}^{3} a_m^l (x_m + x_{7-m}) & pour \quad l \quad paire \\ \sum_{m=0}^{3} a_m^l (x_m - x_{7-m}) & pour \quad l \quad impaire \end{cases} \qquad \textbf{Eq : 10}$$

Pour une écriture plus simple de l'équation 10, on peut définir U_m suivant *l* paire ou impaire comme une somme ou différence de x_m. On aura l'équation suivante :

$$y_l = \sum_{m=0}^{3} a_m^l u_m$$
$$avec \quad u_m = \begin{cases} (x_m + x_{7-m}) & pour \quad l \quad paire \\ (x_m - x_{7-m}) & pour \quad l \quad impaire \end{cases} \qquad \textbf{Eq : 11}$$

Nous pouvons écrire U_m comme une sommation d'un ensemble de bits (note : B est le nombre de bits du coefficient U_m)

$$u_m = -u_m^{(0)} + \sum_{j=1}^{B-1} u_m^{(j)} 2^{-j} \qquad \textbf{Eq : 12}$$

Substituant l'équation 12 dans l'équation 11 on aura :

$$y_l = \sum_{m=0}^{3} a_m^l \left[-u_m^{(0)} + \sum_{j=1}^{B-1} u_m^{(j)} 2^{-j} \right] \qquad \textbf{Eq : 13}$$

On peut réordonner la sommation ci-dessus. On aura l'équation suivante :

$$y_l = \sum_{j=1}^{B-1} \left[\sum_{m=0}^{3} a_m^l u_m^{(j)} \right] 2^{-j} - \sum_{m=0}^{3} a_m^l u_m^{(0)} \qquad \textbf{Eq : 14}$$

Supposant que :

$$F(a^l, u^{(j)}) = \sum_{m=0}^{3} a_m u_m^{(j)} \qquad \textbf{Eq : 15}$$

Substituant l'équation 15 dans l'équation 14, on aura :

$$y_l = \sum_{j=1}^{B-1} F\left(a^l, u^{(j)}\right) 2^{-j} - F\left(a^l, u^{(0)}\right) \qquad \textbf{Eq : 16}$$

L'équation 16 présente l'expression fondamentale pour l'implantation de la TCD en utilisant la méthode de la distribution arithmétique. Pour chaque valeur de *l* le terme $F(a^l, u^{(j)})$ ne peut avoir que 16 valeurs. Ces valeurs peuvent être calculées et stockées dans une table de type LUT (*Look Up Table*) ou en mémoire ROM. La multiplication par 2^{-j} correspond à un décalage à droite de *j* bits.

Cependant, le calcul de *y* peut être effectué par la procédure suivante :

- Accès à la mémoire.
- Addition ou soustraction.
- Décalage à droite d'un bit.

La figure 94 illustre le diagramme synoptique pour l'implantation de l'équation 14. L'information est traitée en série. Après chaque cycle, la sortie de l'accumulateur est décalée d'un bit. La somme finale est stockée en B cycles. Ce circuit est référencé généralement comme *ROM-Accumulateur* (RAC).

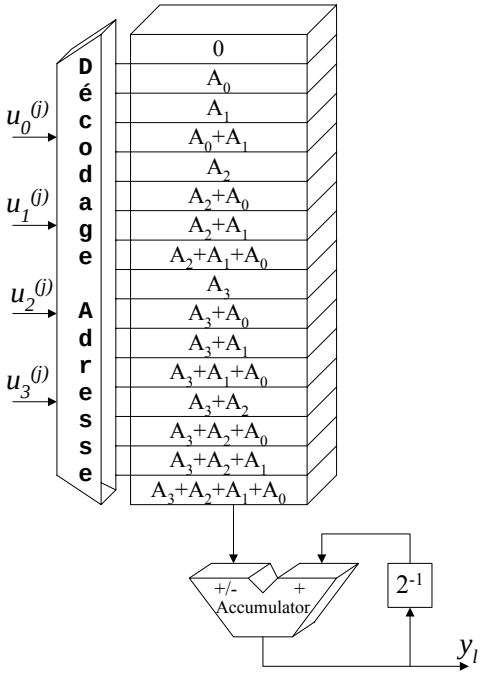


Figure 94. Circuit ROM Accumulateur (RAC)

On peut appliquer la distribution arithmétique (DA) pour l'implantation de la TCD-1D puisque les coefficients de la matrice TCD sont constants et ne varient pas au cours de la procédure (ces coefficients sont fournis en annexe K). Donc, pour calculer la TCD-1D de 8x8 pixels par bloc, il faut utiliser 8 RAC. La figure 95 illustre le diagramme synoptique d'une telle architecture.

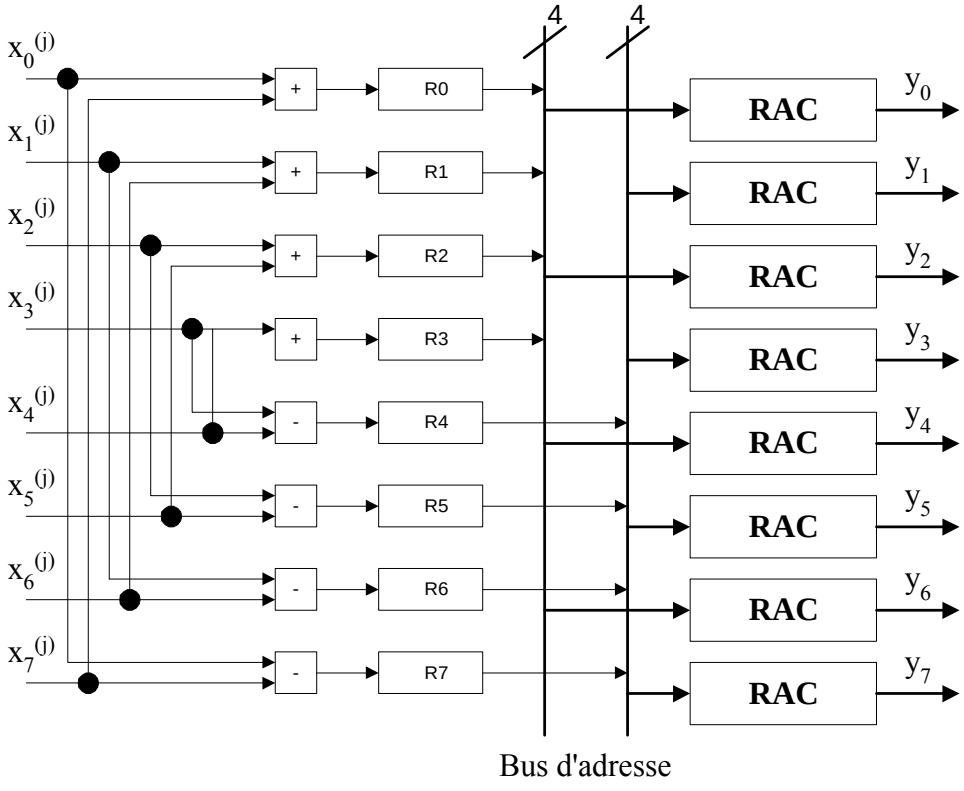


Figure 95. Circuit de TCD-1D

Chaque coefficient va être calculé à partir d'une RAC qui va contenir tous les produits partiels possibles. Chaque ROM va être adressée par un bus constitué par les bits les moins signifiants LSB de chacun des registres à décalage. Chacun des produits partiels obtenus va être additionné avec le contenu de l'accumulateur correspondant, le résultat de cette dernière étape va être décalé à droite d'un bit. Le nombre d'itérations dépend de la valeur du B (B est le nombre de bits du coefficient U_m). Le résultat de la TCD-1D est obtenu en parallèle (y_0 à y_7). L'avantage de cette architecture est qu'elle n'utilise pas de multiplications, elle utilise seulement des additions et des décalages.

IV.5.2.2 Résultats d'implantation :

Les résultats d'implantation de la TCD/TCDI-1D en utilisant la méthode de la distribution arithmétique sur FPGA Stratix II EP2S60 sont donnés par la table 19.

Table 19. Résultats d'implantation de la TCD/TCDI par la méthode DA

	TCD	**TCDI**
ALUTs	499 (1%)	499 (1%)
E/S	195 (40%)	195 (40%)
Blocs RAMs	0%	0%
Blocs DSPs	0%	0%
Fmax (MHz)	281	229

D'après cette table, on constate que la fréquence maximale pour la TCD et la TCDI évaluée par Quartus est de l'ordre de 281 MHz et 228 MHz respectivement. La simulation temporelle post-synthèse donne 166 MHz et 250 MHz pour la TCD et la TCDI respectivement. La figure 96 illustre la simulation temporelle de la TCD par la méthode DA. D'après cette figure, on voit que le traitement d'un bloc de taille 8x8 par la TCD nécessite 112 cycles horloge. En fait, les données de chaque ligne du bloc sont présentées à l'entrée de la TCD en parallèle sur 12 bits (dans notre cas B=12). Le traitement d'une ligne nécessite 12 cycles pour effectuer les décalages des bits et 2 cycles pour le contrôle. Par conséquent, le traitement de chaque ligne nécessite 14 cycles.

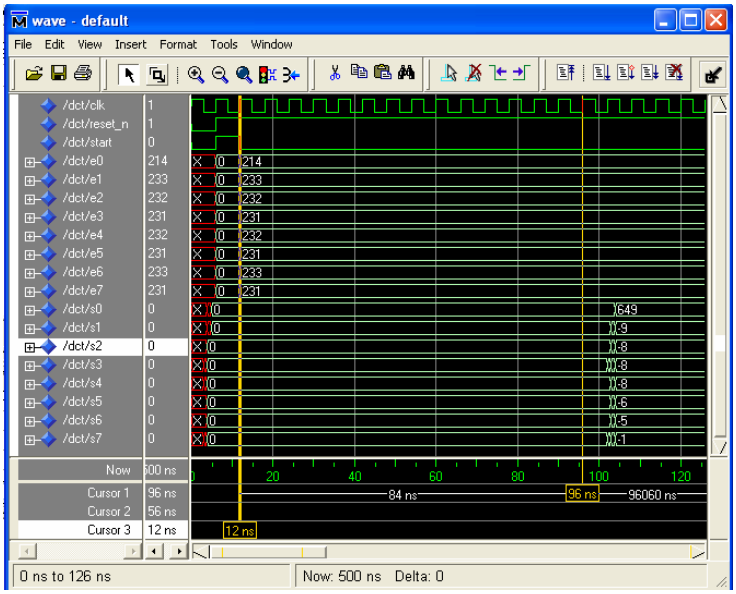
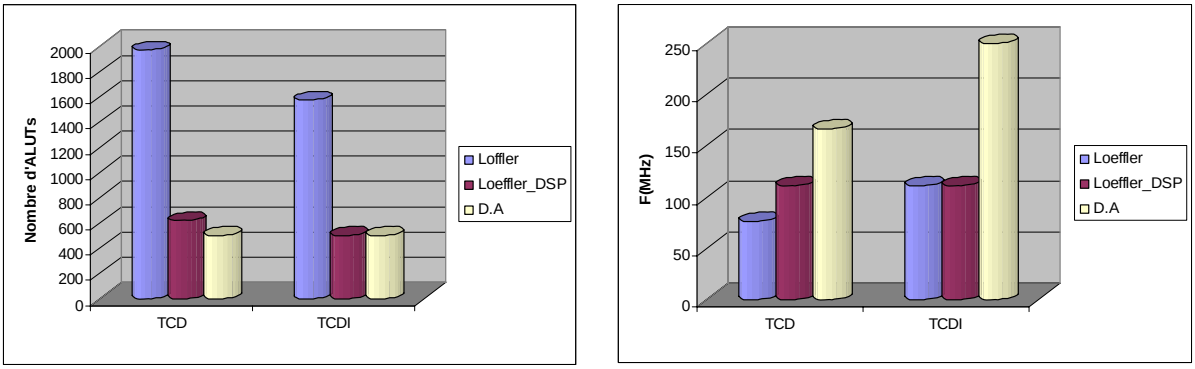


Figure 96. Simulation Temporelle de la TCD-1D par la méthode DA

IV.5.2.3 Interprétation des résultats :

L'ensemble des résultats concernant les ressources utilisées, la fréquence de fonctionnement et le nombre de cycles de traitement des trois méthodes utilisées pour l'implantation de la TCD-1D (architecture de Loeffler, architecture de Loeffler_DSP et la méthode DA) est illustré par les figures 97 et 98. D'après ces figures, on constate que la méthode DA a la fréquence la plus élevée (dans un rapport de 2 au maximum) mais elle nécessite 112 cycles pour le traitement d'un bloc 8x8 par la TCD-1D alors que l'architecture de Loeffler ne nécessite que 11 cycles donc 10 fois moins. La solution Loeffler s'avère au minimum 5 fois plus rapide. En ce qui concerne les ressources FPGA utilisées, elles sont sensiblement équivalentes et ne constituent donc pas un facteur de choix décisif. De ce fait, cette architecture a été retenue pour la réalisation d'un coprocesseur pour la TCD bidimensionnelle (TCD-2D) [115].



(a) (b)

Figure 97. (a) Ressources utilisées et (b) fréquence de fonctionnement pour la TCD/TCDI-1D avec les différentes méthodes

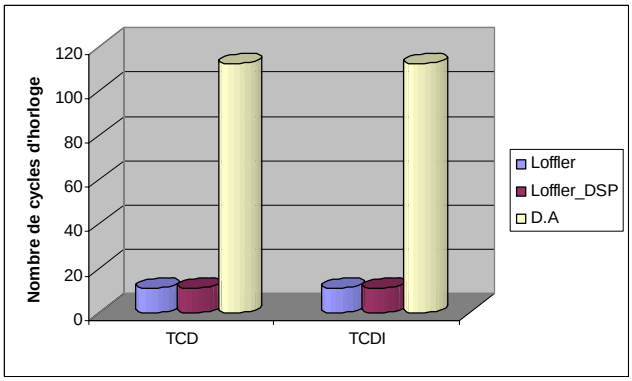


Figure 98 . Nombre de cycles pour le traitement d'un bloc 8x8 par la TCD/TCDI-1D avec les différentes méthodes

IV.5.3 Coprocesseur TCD/TCDI bidimensionnelle :

IV.5.3.1 Principe de fonctionnement du coprocesseur TCD/TCDI-2D :

Le coprocesseur réalisé permet de faire le traitement par la TCD/TCDI bidimensionnelle. La TCD/TCDI opère sur des blocs d'image de taille 8x8. Le traitement consiste à considérer chaque bloc de l'image selon la direction horizontale (suivant les lignes), puis selon la direction verticale (suivant les colonnes) [116] [117] [118]. On applique donc deux traitements unidimensionnels selon deux directions différentes pour obtenir un traitement

bidimensionnel. L'architecture interne du coprocesseur TCD/TCDI est illustrée par la figure 99.

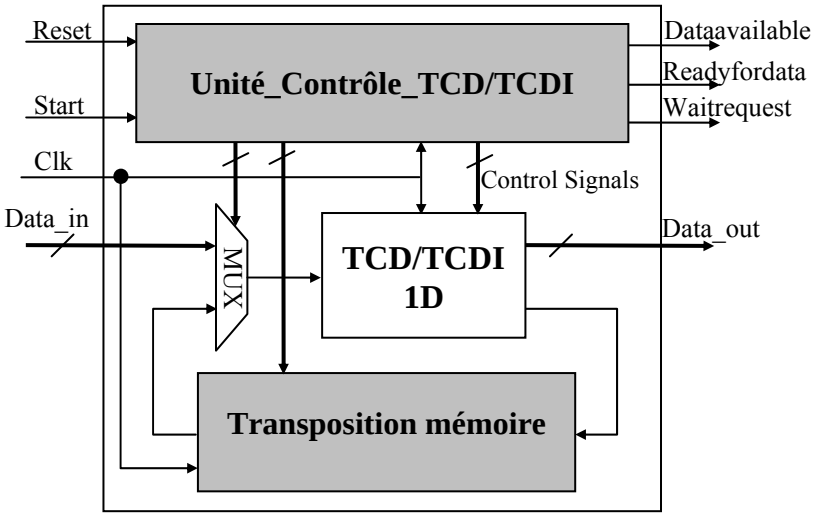


Figure 99. Coprocesseur TCD/TCDI-2D

L'unité de contrôle du coprocesseur reçoit à l'entrée tous les signaux de contrôle (*Start*, *Reset*) et génère des signaux de contrôle internes permettant de gérer le fonctionnement du coprocesseur et des signaux de contrôle pour la communication avec le bus Avalon (*dataavailable*, *readyfordata*, *waitrequest*). Cette unité active la lecture du bloc de l'image suivant les lignes à travers le signal *Data_in* quand le signal *Start* est activé. Chaque ligne reçue est traitée par l'unité TCD/TCDI et stockée dans une mémoire intermédiaire. Cette mémoire joue le rôle de tampon entre les deux étapes de traitement (traitement suivant les lignes puis les colonnes). Les données sont stockées dans la mémoire de telle sorte qu'elles se présentent correctement à l'entrée de la seconde étape. Cette étape est similaire à la première étape mais ici on fait le traitement suivant les colonnes. Dans cette étape, les données sont envoyées de la mémoire vers l'unité TCD/TCDI à travers un multiplexeur. Les pixels traités sont disponibles à travers le signal *Data_out*. Les signaux *Data_in* et *Data_out* sont connectés au bus Avalon. Le coprocesseur TCD/TCDI-2D lit/écrit les données depuis/vers la mémoire à travers ce bus. 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, le transfert de données est effectué par matériel en utilisant un DMA comme montre la figure 100. En fait, le processeur est écarté et c'est le DMA qui prend le bus pour réaliser ce transfert.

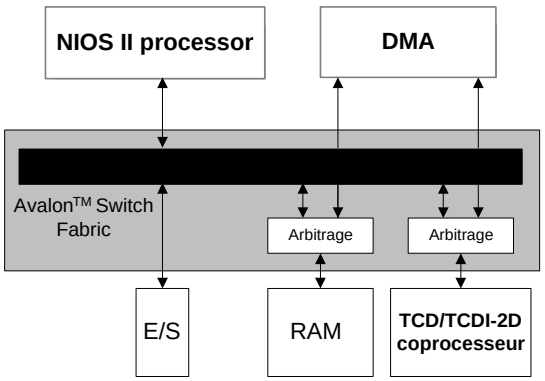


Figure 100. Connexion du coprocesseur TCD/TCDI-2D avec NIOS II

L'unité TCD/TCDI-1D utilise l'architecture modifiée de Loeffler. 11 multiplications et 29 additions sont nécessaires pour le traitement de 8 pixels par la TCD/TCDI-1D. Par contre, 176 multiplications et 464 additions sont nécessaires pour le traitement d'un bloc d'image par TCD/TCDI-2D. Conformant à la spécification IEEE 1180-1990, les coefficients de la TCD/TCDI ont été implantés en utilisant 12 bits de précision [119] [120]. La TCD/TCDI utilise 24 registres afin de conserver les valeurs intermédiaires calculées. Avec cette configuration, la TCD/TCDI-1D est achevée après 11 cycles d'horloge. Par contre, le coprocesseur nécessite 97 cycles d'horloge pour la TCD/TCDI-2D.

IV.5.3.2 Description du coprocesseur TCD/TCDI-2D :

La figure 101 illustre l'entité globale du coprocesseur. Elle est constituée de différents signaux d'E/S pour communiquer avec le reste du système.

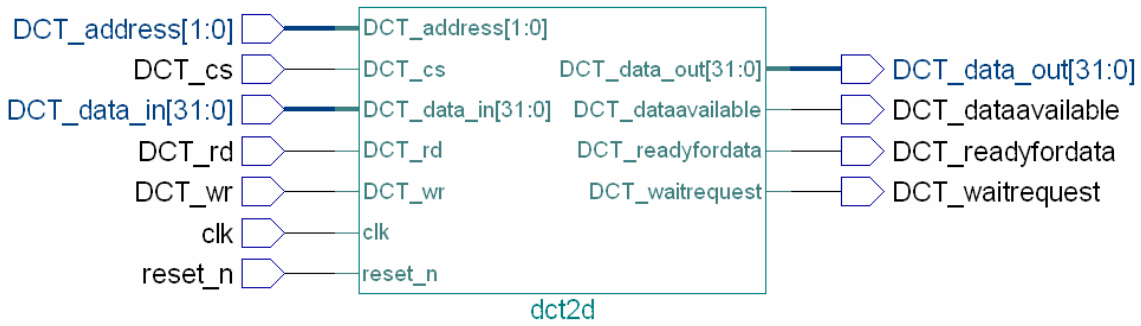


Figure 101. Entité du coprocesseur TCD/TCDI-2D

La table 20 énumère ces signaux avec leurs descriptions.

Table 20. Signaux entre le coprocesseur est le bus Avalon

Signal	Largeur	Direction	Description
Clk	1	In	Horloge du système
Reset_n	1	In	Reset actif à « 0 »
DCT_address	2	In	Adresse du coprocesseur
DCT_cs	1	In	Sélection du périphérique
DCT_rd	1	In	Signal de lecture actif à « 1 »
DCT_wr	1	In	Signal d'écriture actif à « 1 »
DCT_data_in	32	In	Données 32 bits lues du bus Avalon
DCT_data_out	32	Out	Données 32 bits envoyées vers le bus Avalon
DCT_dataavailable	1	Out	« 1 » si le coprocesseur est prêt à envoyer des données
DCT_readyfordata	1	Out	« 1 » si le coprocesseur est prêt pour recevoir des données
DCT_waitrequest	1	Out	« 1 » si le coprocesseur est occupé. Provoque des cycles d'attente

La liaison entre la partie matérielle et la partie logicielle se fait à travers le fichier *system.h*. Le transfert de données entre le coprocesseur et la mémoire se fait à travers le DMA en utilisant les registres *np_DCT_COEFFIN* pour l'écriture des données et *np_DCT_COEFFOUT* pour la lecture des données du coprocesseur. La figure 102 présente un exemple de lecture et d'écriture des pixels dans le coprocesseur TCD-2D.

```
//Déclaration des différents registres dans un fichier *.h
typedef volatile struct
{
  int              np_DCT_STATUS;
  int              np_DCT_COEFFIN;
  int              np_DCT_COEFFOUT;
}
 np_dct;

//déclaration d'un variable qui pointe sur l'adresse du coprocesseur TCD
np_dct *dct=(np_dct *) na_dct2d0;
//écriture des pixels dans le coprocesseur TCD-2D par le DMA
dma1->np_dmacontrol=0x0;  //Arrêt du DMA
dma1->np_dmastatus=0x0; // Initialisation des registres du DMA
dma1->np_dmareadaddress=(int)block11; //Adresse des données dans la RAM
dma1->np_dmawriteaddress=(int)&dct->np_DCT_COEFFIN;//Adresse données TCD
dma1->np_dmalength=data_block_length; // Nombre de mots à transférer * 4
dma1->np_dmacontrol=0x28C; // 0x28C : write,EoP,word
while (dma1->np_dmastatus & np_dmastatus_busy_mask);

//lecture des données du coprocesseur TCD-2D par le DMA
dma1->np_dmacontrol=0x0;
dma1->np_dmastatus=0x0;
dma1->np_dmareadaddress=(int)&dct->np_DCT_COEFFOUT;
dma1->np_dmawriteaddress=(int)block12;
dma1->np_dmalength=data_block_length;
dma1->np_dmacontrol=0x18C; // 0x18C : Read,EoP,word
while (dma1->np_dmastatus & np_dmastatus_busy_mask);
```

Figure 102. Transfert de données entre le coprocesseur et la mémoire par DMA

Les résultats d'implantation du coprocesseur TCD/TCDI-2D sur FPGA Stratix II EP2S60 sont donnés par la table 21.

Table 21. Résultats d'implantation du coprocesseur TCD/TCDI-2D

	TCD	**TCDI**
ALUTs	1512 (3%)	1363 (3%)
E/S	74 (15%)	74 (15%)
Blocs RAMs	1536(1%)	1536(1%)
Blocs DSPs	22(8%)	22(8%)
Fmax (MHz)	120	120

D'après cette table, on constate que la fréquence maximale du coprocesseur TCD/TCDI-2D évaluée par Quartus est de l'ordre de 120 MHz. Cette fréquence est confirmée par la simulation temporelle qui donne 125 MHz et 100 MHz pour le coprocesseur TCD et TCDI respectivement. La figure 103 illustre la simulation temporelle du coprocesseur TCD-2D. D'après cette figure, on voit les données sont disponibles sur le bus que si le signal *waitrequest* est à 0.

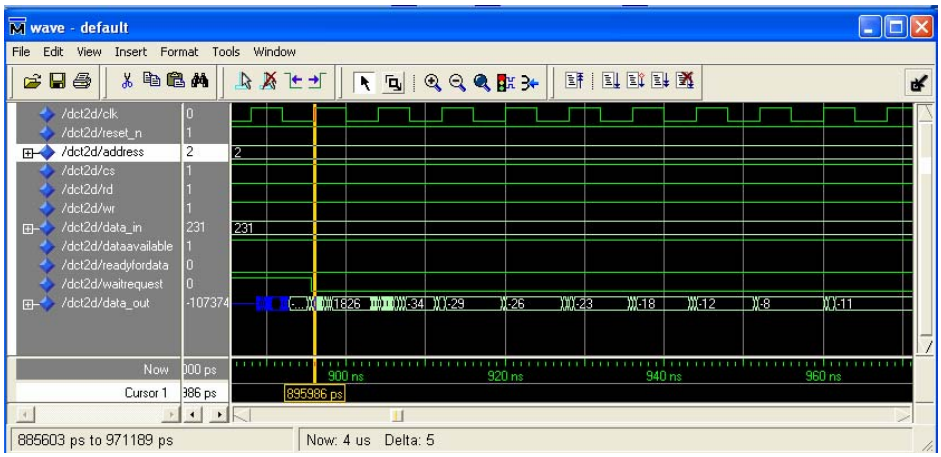


Figure 103. Simulation temporelle du coprocesseur TCD-2D

IV.5.3.3 Évaluation des performances du coprocesseur TCD/TCDI-2D :

L'évaluation des performances du coprocesseur TCD/TCDI-2D est réalisée sur les séquences Miss America et Foreman codées à 10 Hz avec différentes valeurs de quantification (QP=8, 13, 31). L'algorithme de recherche exhaustive est utilisé dans cette évaluation avec une fenêtre de recherche de taille +/-15. La table 22 représente le nombre de cycles d'horloge pour traiter une image par la TCD/TCDI-2D par la méthode logicielle (calcul flottant de la TCD/TCDI), logicielle optimisée (calcul entier de la TCD/TCDI en utilisant 12 bits de précision pour les constantes de la TCD/TCDI) et matérielle. D'après cette table, on constate que la solution logicielle optimisée permet un facteur d'amélioration d'ordre de 37 par rapport à la solution logicielle. Par contre, en utilisant un accélérateur matériel pour la TCD/TCDI-2D, on peut avoir une amélioration de l'ordre de 6 du temps de traitement par rapport à la solution logicielle optimisée.

Table 22. Évaluation du coprocesseur TCD/TCDI-2D pour les séquences Miss America et Foreman

		Miss America			Foreman		
	QP	8	13	31	8	13	31
TCD/TCDI_SW_Flottant							
Cycles d'horloge	**TCD**	92396938	92500394	92357613	94952379	95066966	95153101
	TCDI	31562036	18140351	3346439	68865748	50176475	18593324
	TCD+TCDI	123958974	110640745	95704052	163818127	145243441	113746425
TCD/TCDI_SW_Entier							
Cycles d'horloge	**TCD**	2414932	2450216	2404841	2417464	2462890	2428530
	TCDI	899363	560997	99714	1906777	1479234	547538
	TCD+TCDI	3314295	3011213	2504555	4324241	3942124	2976068
Accélération							
Gain		37	37	38	38	37	38
TCD/TCDI_HW							
Cycles d'horloge	**TCD**	400582	399167	412612	400649	399685	401029
	TCDI	148508	89910	18054	322331	244357	89694
	TCD+TCDI	549090	489077	430666	722980	644042	490723
Accélération							
Gain		6	6	6	6	6	6

Table 23. Évaluation de la qualité de codage sur différentes séquences pour le coprocesseur TCD/TCDI-2D

	QP	PSNR-Y (dB)	SSIM	Bit Rate (kb/s)
TCD/TCDI_SW_Flottant				
Miss America	8	38,16	0,9946	19,03
	13	36,07	0,9913	10,66
	31	32,87	0,9815	5,8
Foreman	8	33,21	0,9944	76,52
	13	30,79	0,9901	41,31
	31	26,9	0,9753	17,62
Claire	8	37,5	0,998	20,92
	13	34,84	0,9963	11,8
	31	30,95	0,9901	5,61
Carphone	8	34,64	0,9968	50,16
	13	32,01	0,994	26,75
	31	27,9	0,9844	11,3
TCD/TCDI_SW_Entier				
Miss America	8	38,05	0,9945	18,31
	13	36,03	0,9912	10,51
	31	32,85	0,9813	5,76
Foreman	8	33,06	0,9942	74,87
	13	30,71	0,9899	40,79
	31	26,94	0,9756	17,77
Claire	8	37,39	0,998	20,28
	13	34,75	0,9962	11,51
	31	30,62	0,9901	5,61
Carphone	8	34,49	0,9967	49,27
	13	31,95	0,994	26,40
	31	27,88	0,9843	11,15
TCD/TCDI_HW				
Miss America	8	38,12	0,9946	18,96
	13	36,03	0,9912	10,73
	31	32,83	0,9813	5,79
Foreman	8	33,12	0,9943	77,66
	13	30,78	0,9901	41,5
	31	26,91	0,9754	17,58
Claire	8	37,39	0,998	21,22
	13	34,82	0,9963	11,85
	31	30,55	0,99	5,6
Carphone	8	34,54	0,9967	50,98
	13	31,99	0,994	26,92
	31	27,9	0,9844	11,25

La TCD/TCDI opère sur des blocs de taille 8x8. Le nombre de cycles d'horloge nécessaire pour le traitement par logiciel d'un bloc image par la TCD/TCDI-2D est de l'ordre de 159800 cycles (1,33 ms pour 120MHz) mais de 4200 cycles (0,035 ms pour 120 MHz) pour la solution logicielle optimisée et de 720 cycles (0,006 ms pour 120MHz) si le traitement est

effectué par matériel. D'après ces résultats, à titre indicatif, on constate que la solution matérielle est 222 fois plus rapide que la solution logicielle et 6 fois plus rapide que la solution logicielle optimisée. Ceci illustre le fait que le processeur NIOS, dépourvu de coprocesseur flottant est très peu performant pour ce genre d'opérations [121].

Passons maintenant à l'analyse de la qualité de codage produite par le codeur H.263 en utilisant le coprocesseur TCD/TCDI-2D. Pour cela, on a calculé le PSNR-Y (PSNR du Luminance) et le SSIM pour différentes séquences tels que Miss America, Foreman, Carphone et Claire. La table 23 montre une perte quasiment nulle au niveau de la qualité de codage par rapport à la solution logicielle optimisé. Par contre, on note une perte acceptable en qualité de PSNR pour la solution matérielle par rapport à la solution logicielle non optimisée (par exemple 0 dB dans le cas de Carphone pour QP=31 et -0,4 dB dans le cas de Claire pour QP=31) et une perte négligeable au niveau de la valeur de SSIM (par exemple 0 dans le cas de Foreman pour QP=13 et -0,0002 dans le cas de Miss America pour QP=31) avec un débit binaire qui reste presque constant.

IV.6 Implantation matérielle de la Q/QI :

À l'intérieur d'un MB (INTRA ou INTER), on utilise le même pas de quantification (QP) pour tous les coefficients (AC) qui peuvent avoir une valeur comprise entre 1 et 31 à l'exception du premier coefficient (DC) des blocs INTRA. La valeur du coefficient DC est quantifiée uniformément avec un pas de quantification de 8. Les équations de quantification pour le mode INTRA et INTER ne sont pas normalisées par la norme H.263. En fait, c'est le comité UIT-T qui a proposé dans leur modèle de test (TMN8) [122] deux équations de quantification correspondant à ces deux modes [123]. Les coefficients INTRA (non DC) et INTER (DC ou non DC) sont quantifiés selon la relation donnée par l'équation 17.

$$|LEVEL| = \begin{cases} \dfrac{|COF|}{2.QP}, & INTRA \\ \dfrac{|COF| - \dfrac{QP}{2}}{2.QP}, & INTER \end{cases} \qquad \textbf{Eq : 17}$$

Après calcul de la valeur |*LEVEL*|, on ajoute le signe de *COF* pour obtenir la grandeur *LEVEL*. Cette relation est définie comme suit :

$$LEVEL = sign(COF).|LEVEL| \qquad \textbf{Eq : 18}$$

La règle de reconstruction de base par quantification inverse (QI) est exprimée par l'équation 18 pour tous les coefficients quantifiés différents de zéro.

$$|REC| = \begin{cases} QP.(2.|LEVEL|+1), & si \quad QP = "impair" \\ QP.(2.|LEVEL|+1) - 1, & si \quad QP = "pair" \end{cases} \qquad \textbf{Eq : 19}$$

Après calcul de la valeur |*REC*|, on ajoute le signe de *LEVEL* pour obtenir la grandeur *REC*. Cette relation est définie comme suit :

$$REC = sign(LEVEL).|REC| \qquad \textbf{Eq : 20}$$

On définit les paramètres suivants :

- *COF* : coefficient de transformée à quantifier.
- *LEVEL* : version quantifiée du coefficient de transformée qui doit être dans l'intervalle [-127,127].
- *REC* : Valeur de coefficient reconstruite.
- « / »: Division par troncature.

Vue la bonne régularité des équations de la Q/QI et afin de réduire le temps nécessaire à la Q/QI, celles-ci peuvent être implantées sous forme matérielle en utilisant les instructions personnalisées du processeur NIOS II.

IV.6.1 Description des instructions de Q/QI :

Afin d'implanter les équations de Q/QI, des instructions multi-cycles ont été utilisées. La figure 104 décrit l'interface globale de ces instructions. Elles sont constituées de différents signaux d'E/S permettant de communiquer avec l'Unité Arithmétique et Logique du processeur NIOS II. Les entrées *Dataa* et *Datab* et la sortie *Result* sont sur 32 bits et synchronisées par l'horloge *Clk*. Le signal *Dataa* prend la valeur de *COF* dans le cas de quantification (*LEVEL* dans le cas de déquantification). Le signal *Datab* possède la valeur de *QP* pour la Q/QI. Le signal *Result* fournit la valeur de *LEVEL* dans le cas de quantification (*REC* dans le cas de déquantification).

Figure 104. Interface de l'instruction pour Q/QI

La table 24 énumère ces signaux avec leur description.

Table 24. Signaux entre l'instruction de Q/QI et l'UAL

Signal	Largeur	Direction	Description
Clk	1	In	Horloge du système
Clk_en	1	In	Clk_en actif à « 1 »
Reset	1	In	Reset actif à « 1 »
Start	1	In	Start actif à « 1 » : Début d'exécution de l'instruction
Dataa	32	In	Première entrée de l'instruction sur 32 bits
Datab	32	In	Deuxième entrée de l'instruction sur 32 bits
Result	32	Out	Sortie de l'instruction sur 32 bits

La figure 105 représente un exemple de déclaration et d'utilisation des instructions de Q/QI. Cette déclaration définit la liaison entre la partie matérielle et la partie logicielle exécutée sous µClinux.

```
//Déclaration de différentes instructions pour Q/QI
#define ALT_CI_QUANT_INTRA_DC_N 0x00000000
#define ALT_CI_QUANT_INTRA_DC(A) __builtin_custom_ini(ALT_CI_QUANT_INTRA_DC_N,(A))

#define ALT_CI_QUANT_INTRA_N 0x00000001
#define ALT_CI_QUANT_INTRA(A,B) __builtin_custom_ini(ALT_CI_QUANT_INTRA_N,(A),(B))

#define ALT_CI_QUANT_INTER_N 0x00000002
#define ALT_CI_QUANT_INTER(A,B) __builtin_custom_ini(ALT_CI_QUANT_INTER_N,(A),(B))

#define ALT_CI_DEQUANT_DC_N 0x00000003
#define ALT_CI_DEQUANT_DC(A) __builtin_custom_ini(ALT_CI_DEQUANT_DC_N,(A))

#define ALT_CI_DEQUANT_N 0x00000004
#define ALT_CI_DEQUANT(A,B) __builtin_custom_ini(ALT_CI_DEQUANT_N,(A),(B))

//Exemple d'utilisation des instructions de Q/QI
qcoeff = ALT_CI_QUANT_INTER(dct_val,QP_mod_quant)
rcoeff = ALT_CI_DEQUANT(qcoeff,QP_mod_dequant)
```

Figure 105. Exemple de déclaration et d'utilisation des instructions de Q/QI

IV.6.2 Architecture interne des instructions de Q/QI :

Les figures 106 et 107 présentent les architectures utilisées pour l'implantation matérielle de l'équation de quantification INTRA et INTER. Par contre, la figure 108 décrit l'architecture matérielle pour l'implantation de l'équation de déquantification. Dans ces architectures, les entrées *Dataa* et *Datab* ainsi que la sortie *Resultat* sont synchronisées par l'horloge système. Le résultat du calcul est obtenu en deux cycles d'horloge.

Les instructions de quantification INTRA et INTER utilisent 16 bits de précision pour l'implantation efficace du coefficient $1/2QP$. Ce coefficient est calculé par le logiciel une fois le pas de quantification sélectionné, puis envoyé à la partie matérielle à travers le signal *Datab*. Ce signal est constitué de deux parties : une partie contient la valeur du coefficient calculé sur 24 bits et une autre partie possède la valeur du QP sur 8 bits. La valeur calculée par l'instruction est décalée à droite 16 fois afin d'obtenir le résultat final. Le comparateur vérifie si ce résultat appartient à l'intervalle [-127,127]. Sinon, le résultat final sera -127 ou 127.

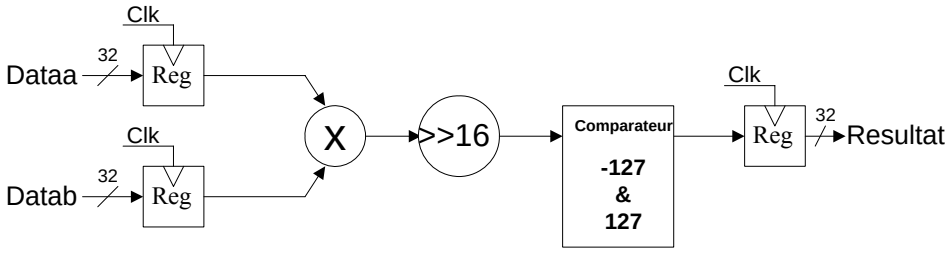


Figure 106. Architecture de l'instruction pour la quantification INTRA

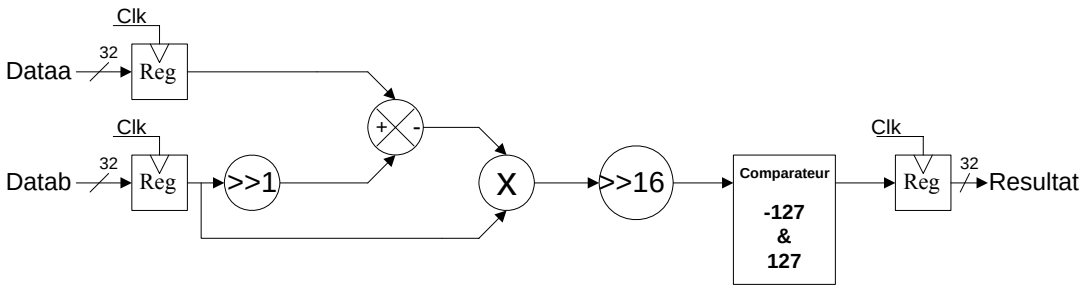


Figure 107. Architecture de l'instruction pour la quantification INTER

D'après la figure 108, on voit que l'instruction de déquantification utilise un multiplexeur. Ce multiplexeur permet de sélectionner la sortie qui dépend du premier bit du signal *Datab*.

- Si Datab(0) = '0' alors QP est impair. Dans ce cas, on obtient directement le résultat.
- Si Datab(0) = '1' alors QP est pair. Dans ce cas, on retranche 1 du résultat.

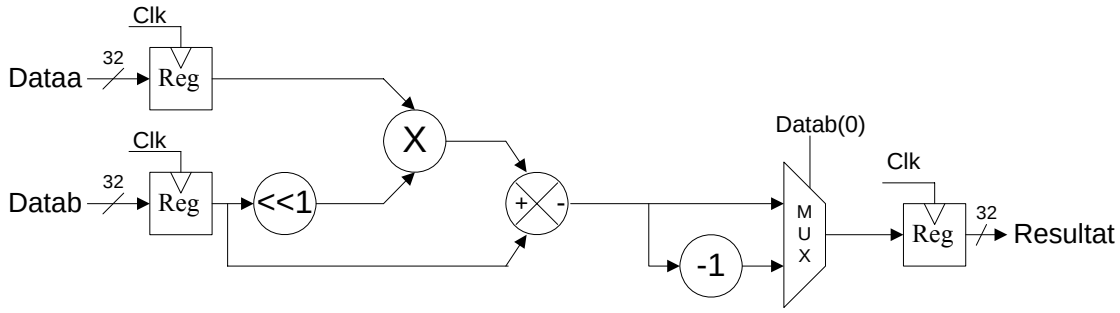


Figure 108. Architecture de l'instruction pour la déquantification

Les résultats d'implantation des instructions pour la Q/QI sur circuit FPGA Stratix II EP2S60 sont donnés par la table 25.

Table 25. Résultats d'implantation des instructions de Q/QI

	Q_INTRA_DC	Q_INTRA	Q_INTER	QI_INTRA_DC	QI
ALUTs	18 (<1%)	79 (<1%)	89 (<1%)	0%	160 (<1%)
E/S	64 (13%)	100 (20%)	100 (20%)	64 (13%)	100 (20%)
Blocs RAMs	0%	0%	0%	0%	0%
Blocs DSPs	0%	2 (<1%)	2 (<1%)	0%	8 (3%)

La fréquence de fonctionnement de ces différentes instructions et de l'ordre de 120 MHz. La figure 109 illustre la simulation temporelle de l'instruction de quantification INTER. Le résultat de calcul est disponible en deux cycles d'horloge.

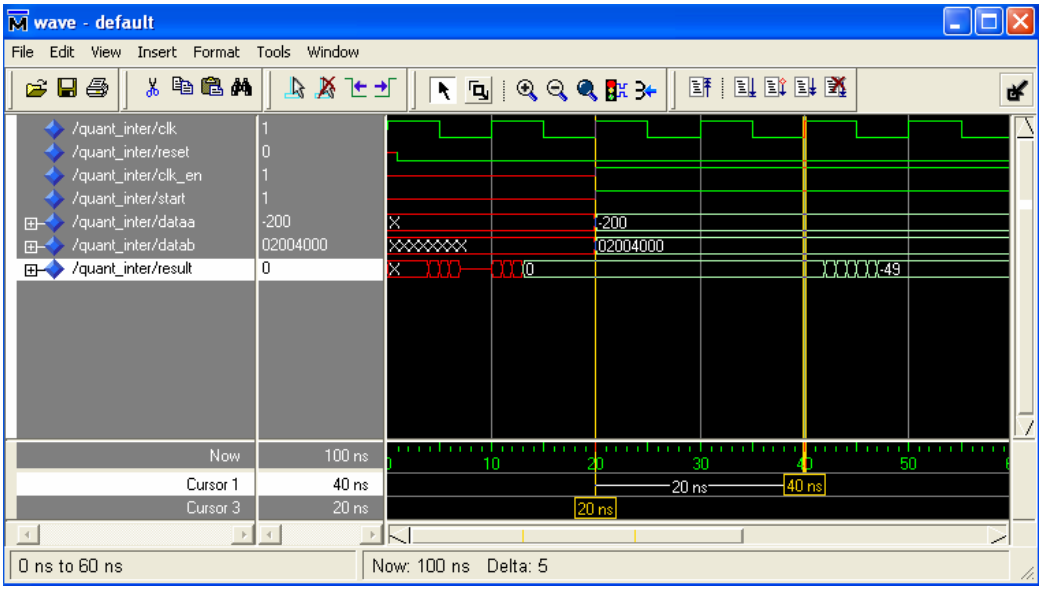


Figure 109. Simulation temporelle de l'instruction Q_INTER

IV.6.3 Evaluation des performances des instructions Q/QI :

L'évaluation des performances des instructions de Q/QI est réalisée sur les séquences Miss America et Foreman codées à 10 Hz avec différentes valeurs de quantification (QP=8, 13, 31). L'algorithme de recherche exhaustive est utilisé dans cette évaluation avec une fenêtre de recherche de taille +/-15. La table 26 présente le nombre de cycles d'horloge pour traiter une image par la méthode logicielle ou matérielle. D'après cette table, on constate qu'en utilisant un accélérateur matériel pour Q/QI, cela permet d'avoir une amélioration d'un facteur de 4 du temps de traitement par rapport à la solution logicielle. En fait, le traitement d'un bloc de taille 8x8 par exemple par l'instruction de quantification INTER nécessite 1200 cycles d'horloge. Cette mesure comprend 128 cycles d'horloge pour le traitement de type matériel et 1072 cycles d'horloge pour les accès mémoire. Ces résultats montrent que 89% du temps est alloué aux accès mémoires.

Table 26. Évaluation des instructions de Q/QI pour les séquences Miss America et Foreman

		Miss America			Foreman		
	QP	8	13	31	8	13	31
Q/QI_SW							
Cycles d'horloge	**Q**	3394559	3383492	3385624	3431254	3428078	3393838
	QI	294757	160534	34579	583857	463768	163574
	Q+QI	3689316	3544026	3420203	4015111	3891846	3557412
Q/QI_HW							
Cycles d'horloge	**Q**	709589	709329	719721	724970	711003	709844
	QI	265008	144794	27429	513363	368678	139360
	Q+QI	974597	854123	747150	1238333	1079681	849204
Accélération							
Gain		4	4	5	3	4	4

Afin d'évaluer la qualité des images produites par le codeur H.263 en utilisant les instructions de Q/QI, on a calculé le PSNR-Y et le SSIM. La table 27 donne la qualité de codage sur les séquences Miss America, Foreman, Carphone et Claire. Cette table montre une perte acceptable en qualité de PSNR pour la solution matérielle par rapport à la solution logicielle (par exemple 0 dB dans le cas de Miss America pour QP=8 et -0,41 dB dans le cas de Claire pour QP=31) et une perte négligeable au niveau des mesures sur la qualité perceptuelle de l'image (par exemple 0 dans le cas de News pour QP=8 et -0,0003 dans le cas de Foreman pour QP=31) avec un débit binaire qui reste presque constant.

Table 27. Évaluation de la qualité de codage sur différentes séquences pour les instructions de Q/QI

	QP	PSNR-Y (dB)	SSIM	Bit Rate (kb/s)
Q/QI_SW				
Miss America	8	38,16	0,9946	19,03
	13	36 ,07	0,9913	10,66
	31	32 ,87	0,9815	5,8
Foreman	8	33,21	0,9944	76,52
	13	30,79	0,9901	41,31
	31	26 ,9	0,9753	17,62
Claire	8	37,5	0,998	20,92
	13	34,84	0,9963	11,8
	31	30,95	0,9901	5,61
Carphone	8	34,64	0,9968	50,16
	13	32,01	0,994	26,75
	31	27,9	0,9844	11,3
Q/QI_HW				
Miss America	8	38,16	0,9946	19,03
	13	35,98	0,9911	10,31
	31	32,83	0,9813	5,76
Foreman	8	33,21	0,9944	76,52
	13	30 ,64	0,9898	39 ,86
	31	26,86	0,9751	17, 38
Claire	8	37,5	0,998	20,92
	13	34,72	0,9962	11,37
	31	30 ,54	0,9899	5,55
Carphone	8	34,64	0,9968	50,16
	13	31,86	0,9938	25,83
	31	27,84	0,9842	11,2

IV.7 Implantation logicielle/matérielle du codeur H.263 :

IV.7.1 Personnalisation et génération du système multimédia embarqué :

Le système multimédia embarqué réalisé est constitué par les éléments suivants :

- CPU NIOS 32 bits.
- Coprocesseur TCD/TCDI-2D.
- Coprocesseur SAD.
- Instructions pour la Q/QI.
- Interface caméra.
- Interface VGA.
- Bus de SRAM externe (Avalon Tri-state Bridge).
- Interface SRAM (1 Mo).
- Interface FLASH (16 Mo).
- Contrôleur SDRAM (16 Mo).
- Contrôleur Ethernet.

- UART pour la communication.
- Timer.

Ces différents composants sont associés, connectés et générés en utilisant l'outil Altera SOPC Builder comme présenté à la figure 110.

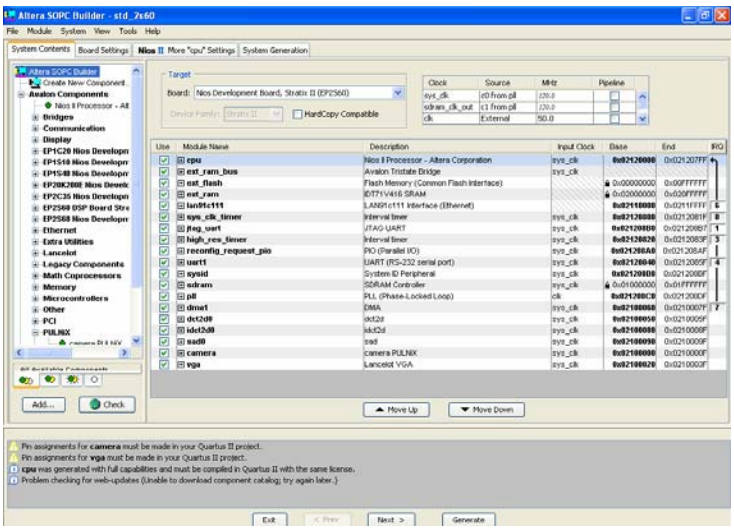


Figure 110. Génération du système multimédia embarqué avec SOPC Builder

La figure 111 décrit le système multimédia embarqué réalisé. Il est constitué du processeur NIOS II, des accélérateurs matériels pour le codeur H.263 tel que coprocesseur pour TCD/TCDI-2D et SAD, des instructions pour Q/QI ainsi que des interfaces caméra et VGA pour l'acquisition et la restitution de l'image. Ces blocs IP sont connectés entre eux à travers le bus Avalon.

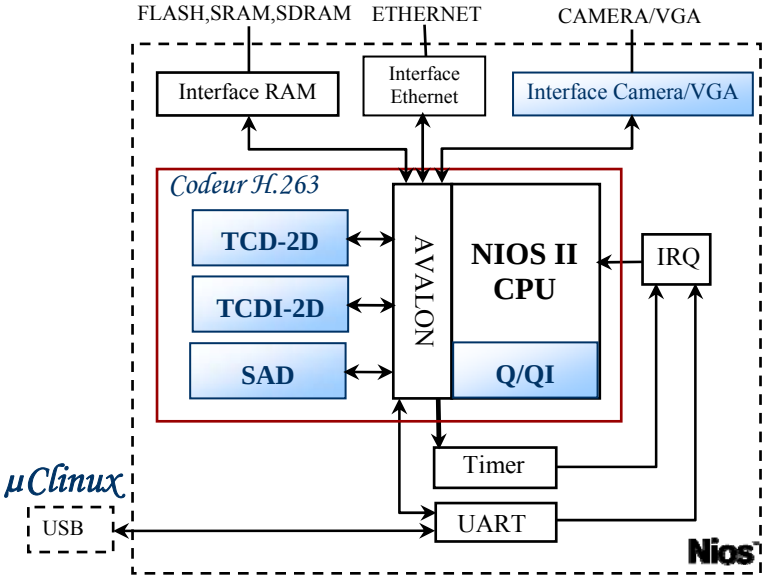


Figure 111. Système multimédia embarqué

Après le développement des différents blocs IPs afin de bénéficier de la performance du matériel, on a développé pour chaque accélérateur matériel la partie logicielle correspondante pour effectuer les communications avec le reste du système. Le codeur H.263 est exécuté sous µClinux comme le montre l'exemple de la figure 112 en indiquant le fichier d'entrée et de

sortie ainsi que les paramètres de configuration du codeur. Le codeur H.263 peut recevoir des séquences vidéo de la caméra. Ces séquences seront traitées par le codeur. Ensuite les séquences reconstruites seront affichées à l'écran en Temps Réel.

Figure 112. Exécution du codeur H.263 sous µClinux

Les résultats d'implantation du système multimédia embarqué sur FPGA Stratix II EP2S60 sont donnés comme suit :

- ALUTs : 20,550/ 48,352 (43 %).
- Pins E/S : 181 / 493 (37 %).
- Blocs RAMs : 1,157,888 / 2,544,192 (46 %).
- Blocs DSPs : 64/288 (22%).
- PLL : 1/6 (17%)
- Fréquence maximale : 120 MHz

Ces résultats montrent que l'implantation des différents périphériques constituant notre système laisse suffisamment de ressources dans le circuit FPGA pour ajouter d'autres blocs IP. La figure 113 illustre la disposition des différents blocs IPs composant notre système multimédia embarqué sur FPGA Stratix II EP2S60. Cette disposition est générée automatiquement par Quartus lors de la phase de placement et routage.

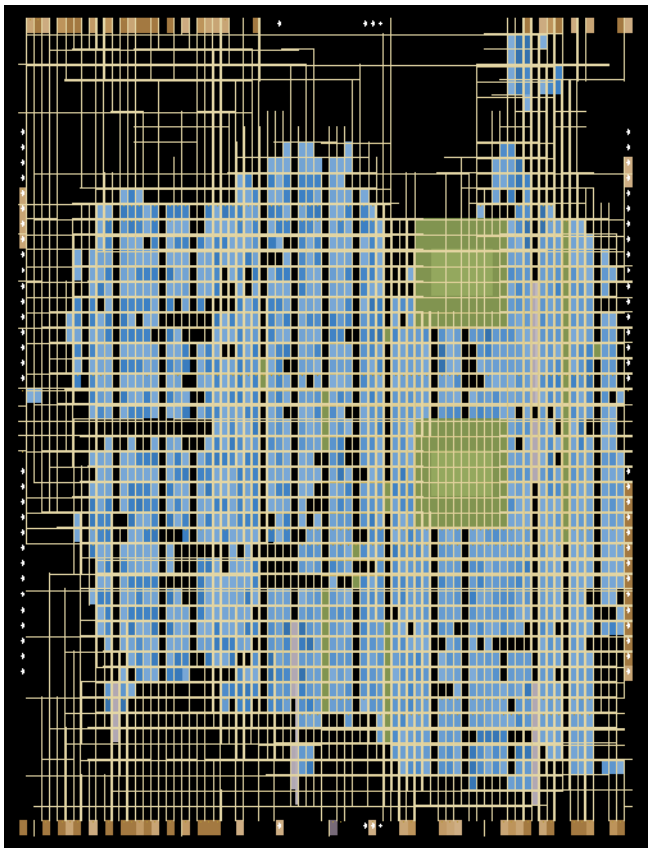


Figure 113. Placement et routage de notre système multimédia sur FPGA Stratix II

IV.7.2 Évaluation des performances du codeur H.263 :

Dans ce paragraphe, l'évaluation de notre conception logicielle/matérielle va être faite par rapport à la solution logicielle du codeur dans laquelle le calcul TCD/TCDI est effectué sur des entiers. Les mesures de pourcentage du taux d'occupation du processeur en fonction des différents blocs de traitement composant le codeur H.263 sont illustrées par les figures 114 et 115. On y représente la solution logicielle ainsi que la solution mixte logicielle/matérielle pour les séquences Miss America et Foreman. Ces mesures sont données pour QP=13. Par contre, la table 28 présente le pourcentage du temps CPU en fonction du bloc de traitement pour ces deux solutions avec QP=8, 13 et 31.

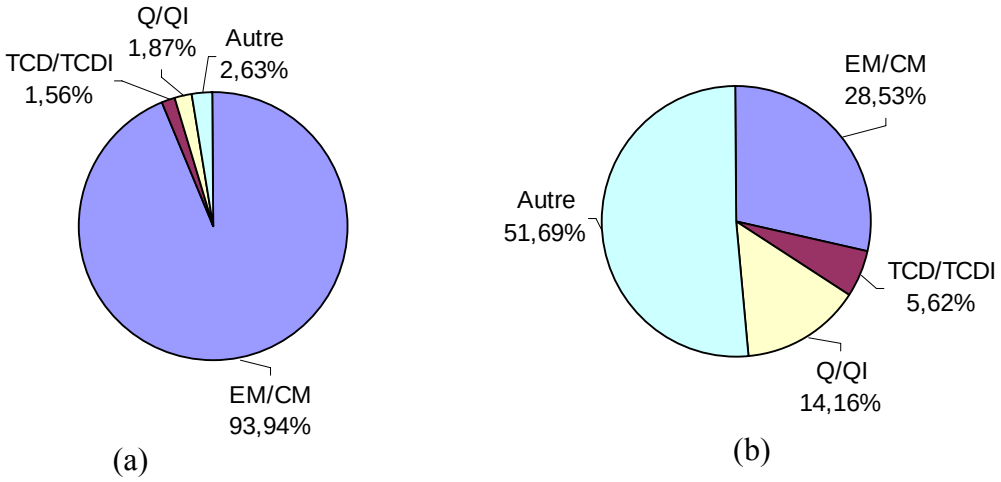


Figure 114. Répartition du temps CPU en utilisant la solution (a)SW et (b) HW/SW pour la séquence Miss America

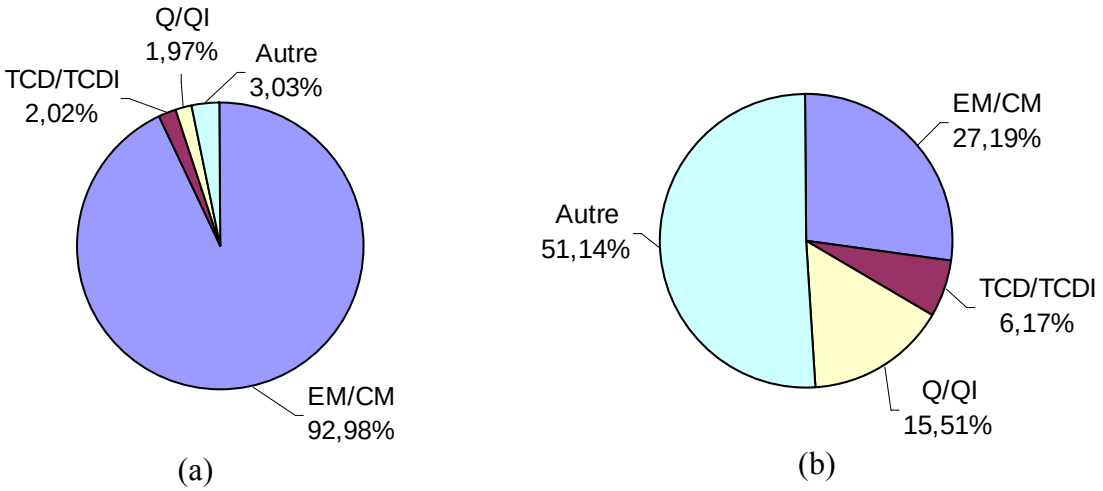


Figure 115. Répartition du temps CPU en utilisant la solution (a) SW et (b) HW/SW pour la séquence Foreman

D'après les figures 114 et 115, on remarque qu'après optimisation, 51% du temps CPU au lieu de 3% avant optimisation sont nécessaires pour le CLV et les autres tâches du système de codage (contrôle du système, accès mémoire…), ceci pour les deux séquences Miss America et Foreman. Aussi d'après ces deux figures, on peut dire que le temps alloué à l'estimation de mouvement et à la quantification reste encore élevé. Ce temps est dû aux accès mémoire qui sont importants surtout dans le cas de L'EM/CM.

Par contre, d'après la table 28 on voit bien qu'on obtient un facteur de réduction de l'ordre de 20 dans le temps de codage de la solution logicielle/matérielle par rapport à la solution logicielle. Donc en utilisant l'approche *codesign* pour le codeur H.263, on arrive à coder des séquences QCIF@15 Hz. Ce qui souligne l'intérêt du *codesign* dans l'implantation des codeurs vidéo [125].

Table 28. Pourcentage du temps CPU en fonction du bloc de traitement du codeur H.263 pour les séquences Miss America et Foreman

	Miss America			Foreman		
QP	**8**	**13**	**31**	**8**	**13**	**31**
H.263_SW						
EM/CM	93,79	93,94	94,34	92,5	92,98	93,98
TCD/TCDI	1,71	1,56	1,3	2,19	2,02	1,54
Q/QI	1,87	1,87	1,77	2	1,97	1,84
CLV	0,25	0,24	0,04	0,69	0,44	0,16
Autre	2,38	2,49	2,55	2,62	2,59	2,48
fps	0,62	0,62	0,62	0,61	0,61	0,62
H.263_SW_HW						
EM/CM	27,99	28,53	30,21	24,85	27,19	31,25
TCD/TCDI	5,97	5,62	5,11	6,27	6,17	5,36
Q/QI	14,95	14,16	12,87	15,97	15,51	13,63
CLV	5,62	2,91	0,97	13,15	8,63	3,67
Autre	45,47	48,78	50,84	39,76	42,51	46,09
fps	12,81	13,84	14,73	10,15	11,10	12,77
Accélération						
Gain	20,7	22,3	23,6	16,6	18,2	20,6

Passons maintenant à l'analyse de la qualité de codage. Les métriques PSNR et SSIM ont été utilisées afin de mesurer les qualités objectives du codage produit par le codeur H.263 avant et après optimisation. La table 29 illustre la qualité du codage pour les séquences vidéo de test telles que Miss America, Foreman, Claire, Carphone, Akiyo et News. Cette table montre une perte négligeable au niveau de la qualité perceptuelle de l'image pour la solution logicielle/matérielle par rapport à la solution logicielle (par exemple 0 dans le cas de Akiyo pour QP=8 et -0,0031 dans le cas de Foreman pour QP=31). Cette mesure est donnée par la métrique SSIM. Par contre, les évaluations de la qualité d'image réalisée à base d'une mesure quantitative du PSNR ont montré une perte acceptable en qualité pour la solution logicielle/matérielle par rapport à la solution logicielle (par exemple -0,01 dB dans le cas de Akiyo pour QP=8 et -0,51 dB dans le cas de Foreman pour QP=31). Enfin, on constate que le débit binaire reste presque constant avec une augmentation du nombre d'images que l'on peut coder par seconde.

On remarque d'après la table 29, que le nombre d'images codées par seconde dépend essentiellement de la dynamique de l'image ainsi que du pas de quantification choisi. En fait, plus le pas de quantification est grand, plus le nombre de valeurs nulles augmente. Dans ce cas, le codeur nécessite moins d'efforts pour coder l'image. D'autre part, dans le cas où la vidéo ne contient pas beaucoup de mouvements comme par exemple le cas des séquences statiques où les séquences sont presque similaires, l'image résiduelle (différence entre l'image courante et l'image prédite) obtenue par l'estimation de mouvement contient plusieurs coefficients nuls. Dans ce cas aussi, le codeur requiert moins de temps pour coder les images en termes de codage entropique, TCD/TCDI et Q/QI.

Les images a, b et c de la figure 116 présentent respectivement l'image originale, l'image reconstruite par la solution logicielle et celle reconstruite par la solution logicielle/matérielle de la 8^{éme} image des séquences vidéo de test pour QP=8. Même chose pour la figure 117, mais on a testé la 40^{éme} image pour QP=13. D'après ces deux figures, on ne constate pas de différences perceptuelles au niveau des mesures subjectives entre l'image traitée par la méthode logicielle et celle par la méthode logicielle/matérielle. Ces résultats sont confirmés par les mesures objectives données par la métrique SSIM.

Table 29. Évaluation de la qualité de codage sur différentes séquences de tests

	QP	PSNR-Y/Cb/Cr (dB)			SSIM	Bit Rate (kb/s)	fps
H.263_SW							
Miss America	8	38,05	38,43	37,62	0,9945	18,31	0,62
	13	36,03	37,28	35,38	0,9912	10,51	0,62
	31	32,85	35,77	32,58	0,9813	5,76	0,62
Foreman	8	33,06	37,90	39,08	0,9942	74,87	0,61
	13	30,71	36,48	37,44	0,9899	40,79	0,61
	31	26,94	34,60	35,12	0,9756	17,77	0,62
Claire	8	37,39	37,81	40,01	0,998	20,28	0,63
	13	34,75	35,67	38,54	0,9962	11,51	0,63
	31	30,62	32,17	34,91	0,9901	5,61	0,64
Carphone	8	34,49	39,48	39,37	0,9967	49,27	0,61
	13	31,95	37,90	37,44	0,994	26,40	0,62
	31	27,88	35,42	34,92	0,9843	11,15	0,63
Akiyo	8	35,51	38,05	40,09	0,996	20,75	0,62
	13	32,90	35,54	37,95	0,9927	11,13	0,62
	31	29,06	30,58	35,01	0,9819	4,62	0,63
News	8	33,80	37,65	38,36	0,9949	51,16	0,61
	13	30,97	35,24	36,33	0,9901	28,59	0,61
	31	26,16	31,65	33,53	0,9693	10,56	0,61
H.263_SW_HW							
Miss America	8	37,90	38,48	37,61	0,9943	19,58	12,81
	13	35,72	37,28	35,41	0,9902	10,22	13,84
	31	32,51	35,96	32,25	0,98	5,35	14,73
Foreman	8	32,91	37,82	39,06	0,994	84,72	10,15
	13	30,35	36,48	37,47	0,9891	43,07	11,1
	31	26,43	34,57	35,26	0,9725	16,49	12,77
Claire	8	37,20	37,87	40,09	0,9979	21,29	13,48
	13	34,41	35,76	38,59	0,996	11,27	14,11
	31	30,29	32,08	35,44	0,9894	5,04	14,83
Carphone	8	34,39	39,52	39,46	0,9966	53,89	11,24
	13	31,65	37,82	37,36	0,9935	26,75	12,19
	31	27,57	35,34	35,04	0,9831	10,30	13,81
Akiyo	8	35,52	38,08	40,01	0,996	21,87	13,58
	13	32,71	35,29	37,92	0,9923	11,01	14,14
	31	28,91	30,84	34,73	0,9812	4,42	15
News	8	33,76	37,69	38,45	0,9948	53,41	12,41
	13	30,79	35,27	36,28	0,9896	28,25	13,06
	31	26,08	31,64	33,55	0,9687	10,13	14,17

Miss America

PSNR-Y = 38,07 dB
PSNR-Cb = 38,55 dB
PSNR-Cr = 37,56 dB
SSIM = 0,9943

PSNR-Y = 38,04 dB
PSNR-Cb = 38,49 dB
PSNR-Cr = 37,85 dB
SSIM = 0,9943

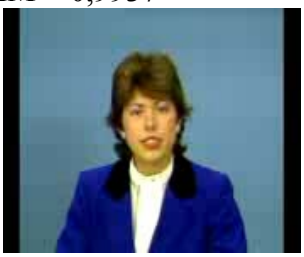
Foreman

PSNR-Y = 32,99 dB
PSNR-Cb = 37,35 dB
PSNR-Cr = 38,43 dB
SSIM = 0,9940

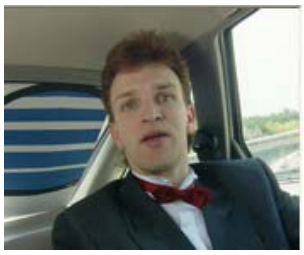
PSNR-Y = 32,77 dB
PSNR-Cb = 37,37 dB
PSNR-Cr = 38,44 dB
SSIM = 0,9937

Claire

PSNR-Y = 37,57 dB
PSNR-Cb = 37,85 dB
PSNR-Cr = 40,21 dB
SSIM = 0,9981



PSNR-Y = 37,24 dB
PSNR-Cb = 37,95 dB
PSNR-Cr = 40,31 dB
SSIM = 0,9979



Carphone

PSNR-Y = 34,16 dB
PSNR-Cb = 39,36 dB
PSNR-Cr = 39,49 dB
SSIM = 0,9963

PSNR-Y = 34,04 dB
PSNR-Cb = 39,53 dB
PSNR-Cr = 39,54 dB
SSIM = 0,9962

Akiyo

PSNR-Y = 35,49 dB
PSNR-Cb = 38,25 dB
PSNR-Cr = 40,28 dB
SSIM = 0,9961

PSNR-Y = 35,61 dB
PSNR-Cb = 38,18 dB
PSNR-Cr = 40,25 dB
SSIM = 0,9962

News

PSNR-Y = 33,80 dB
PSNR-Cb = 37,32 dB
PSNR-Cr = 38,22 dB
SSIM = 0,9945

PSNR-Y = 33,75 dB
PSNR-Cb = 37,06 dB
PSNR-Cr = 38,32 dB
SSIM = 0,9945

(a) (b) (c)

Figure 116. (a) Originale, (b) reconstruite par SW et (c) reconstruite par HW/SW de la 8ème image des séquences vidéo de test pour QP=8

Miss America

PSNR-Y = 35,75 dB
PSNR-Cb = 37,48 dB
PSNR-Cr = 34,83dB
SSIM = 0,9910

PSNR-Y = 35,52 dB
PSNR-Cb = 37,29 dB
PSNR-Cr = 34,92 dB
SSIM = 0,9905

Foreman

PSNR-Y = 30,45 dB
PSNR-Cb = 36,65 dB
PSNR-Cr = 37,52 dB
SSIM = 0,9893

PSNR-Y = 30,07 dB
PSNR-Cb = 36,67 dB
PSNR-Cr = 37,38 dB
SSIM = 0,9891

Claire

PSNR-Y = 34,71 dB
PSNR-Cb = 35,8 dB
PSNR-Cr = 38,44 dB
SSIM = 0,9962

PSNR-Y = 34,15 dB
PSNR-Cb = 35,65 dB
PSNR-Cr = 38,31 dB
SSIM = 0,9959

Carphone

PSNR-Y = 31,88 dB
PSNR-Cb = 37,92 dB
PSNR-Cr = 36,74 dB
SSIM = 0,994

PSNR-Y = 31,62 dB
PSNR-Cb = 37,65 dB
PSNR-Cr = 37,04 dB
SSIM = 0,9937

Akiyo

PSNR-Y = 30,88 dB
PSNR-Cb = 35,45 dB
PSNR-Cr = 37,7 dB
SSIM = 0,9925

PSNR-Y = 32,71 dB
PSNR-Cb = 35,19 dB
PSNR-Cr = 37,71 dB
SSIM = 0,9923

News

PSNR-Y = 30,9 dB
PSNR-Cb = 35,88 dB
PSNR-Cr = 36,67 dB
SSIM = 0,9909

PSNR-Y = 30,8 dB
PSNR-Cb = 36,02 dB
PSNR-Cr = 36,49 dB
SSIM = 0,9905

(a) (b) (c)

Figure 117. (a) Originale, (b) reconstruite par SW et (c) reconstruite par HW/SW de la 40ème image des séquences vidéo de test pour QP=13

IV.8 Conclusion :

Dans ce chapitre, on a étudié l'implantation dans un environnement logiciel/matériel du codeur H.263.

Une étude détaillée des différents blocs du codeur a été élaborée afin de sélectionner les blocs cibles à implanter sous forme logicielle ou sous forme matérielle. Cette étude a abouti à une implantation matérielle de l'opérateur de calcul de la somme des valeurs absolues des différences (SAD), TCD/TCDI et Q/QI. L'implantation logicielle a été estimée préférable en ce qui concerne la méthode de recherche du vecteur de mouvement (l'algorithme de recherche en hexagone a été retenu) ainsi que le codage entropique.

L'étude et l'évaluation de la performance du codeur avant et après optimisation ont été réalisées sur notre plateforme de traitement vidéo à base du processeur NIOS II exécutant µClinux. La fréquence de l'horloge système est de 120 MHz. Avec ces caractéristiques, on obtient un facteur de réduction de l'ordre de 20 dans le temps de codage de la solution logicielle/matérielle par rapport à la solution logicielle. Notre codeur permet de coder des séquences QCIF@15Hz. La table 30 illustre une comparaison entre notre codeur et d'autres réalisations.

Si l'on compare notre réalisation avec [125] (S. H. Lee et al.), on constate que notre architecture permet plus de flexibilité qu'un ASIC pour une performance quasi équivalente. Notre réalisation a de bonnes performances par rapport à [126] (S. M. Akramullah et al.) avec une fréquence processeur nettement plus faible.

Table 30. Comparaison de performance du codeur H.263

Ref	Fps	CLK (MHz)	Spécification
S. H. Lee et al. [125]	QCIF@15 Hz	40 MHz	ASIC pour H.263 processeur ARM720T pour le contrôle
S. M. Akramullah et al. [126]	QCIF@12 Hz QCIF@18 Hz QCIF@45 Hz	167 MHz 233 MHz 600 MHz	Sun Ultra SPARC-1 VIS Processeur PII MMX Processeur PIII MMX
T. Q. Nguyen et al. [127]	QCIF@21 Hz	200 MHz	Processeur dédié pour les applications multimédia (VIRAM)
K. T. Shih et al. [128]	QCIF@20 Hz	200 MHz	DSP (1600 MIPS)
M. Harrand et al. [129]	CIF@20 Hz	54 MHz	ASIP : architecture multiprocesseur dédiée
Notre réalisation	QCIF@15 Hz	120 MHz	Processeur NIOS II (142 MIPS) plus des accélérateurs matériels

Partie IV : Conclusion générale

Conclusion Générale

L'objectif de cette thèse est la contribution au développement et à la conception d'un système multimédia embarqué en utilisant la méthodologie de conception logicielle/matérielle (*codesign*). Il en a découlé la constitution d'une bibliothèque des modules IPs (*Intellectual Property*) pour les applications vidéo.

On a ainsi réalisé une plateforme matérielle d'acquisition et de restitution vidéo servant de préalable à toute étude d'algorithmes de traitement vidéo. La plateforme matérielle s'articule autour d'une carte Stratix II d'Altera complétée d'une interface caméra et d'une interface VGA. Le cœur du système met en œuvre le processeur d'Altera NIOS II dans l'environnement de développement Quartus II d'Altera. Cette architecture matérielle a nécessité l'utilisation d'un système d'exploitation Linux embarqué tel que μClinux permettant de gérer les différentes unités du système et de superviser l'exécution des applications.

En second lieu, le concept de codage vidéo a été abordé afin d'avoir une vision détaillée des principes employés dans tout type de système de compression. On s'est intéressé en particulier à la norme H.263 de l'organisme UIT-T, de sa structure et de ses éléments principaux. Par la suite, une étude au niveau algorithmique des différentes configurations du codeur vidéo sur notre plateforme a été réalisée afin de pouvoir envisager la réalisation d'un système multimédia embarqué Temps Réel traitant des images de plus grande résolution à des fréquences plus élevées. La faisabilité d'un tel système a nécessité la mise en place d'une méthodologie de conception logicielle/matérielle (*codesign*) prenant en compte les contraintes de l'embarqué.

Ce travail nous a permis de constater que la norme H.263 de l'organisme UIT-T comprend systématiquement les quatre blocs suivants : EM/CM, TCD/TCDI, Q/QI et CLV. Les algorithmes décrivant ces blocs sont plus ou moins évolutifs en fonction des versions. L'estimation de mouvement est un bloc candidat à une réalisation matérielle. En effet, sans aller jusqu'à une réalisation d'un estimateur de mouvement entièrement câblé qui serait coûteuse et empêcherait toute modification de la technique d'estimation de mouvement, il est intéressant de réaliser l'opérateur SAD par matériel. Le bloc TCD/TCDI est généralement entièrement câblé dans les systèmes visant les applications vidéo et constitue par conséquent un excellent candidat à une réalisation sous forme d'un bloc IP. Par ailleurs, vue la bonne régularité de l'équation de la quantification/quantification inverse (Q/QI), elle peut être implantée par matériel en utilisant les instructions personnalisées du processeur NIOS II. Les tables des codes du Codage à Longueur Variable sont remises à jour régulièrement. C'est pourquoi cette partie est le plus souvent réalisée par logiciel, ce qui permet de les reconfigurer à volonté.

La plateforme matérielle d'acquisition et de restitution vidéo réalisée a servi à l'évaluation de notre méthodologie de conception logicielle/matérielle pour les systèmes multimédia embarqués. La fréquence de fonctionnement de notre plateforme est de 120 MHz. Le codeur H.263 avec des accélérateurs matériels pour le SAD, TCD/TCDI et Q/QI a été testé sur la plateforme réalisée en utilisant la caméra pour l'acquisition d'images et l'interface VGA pour l'affichage des images à l'écran. Le codeur permet de coder des séquences QCIF@15Hz.

Les blocs IPs d'acquisition, restitution vidéo ainsi que les accélérateurs matériels pour les algorithmes de traitement vidéo tels que les opérations SAD, TCD/TCDI et Q/QI ont été développés en VHDL et testés afin de constituer une bibliothèque de blocs IP.

Il reste à souligner l'aspect particulier du circuit FPGA au niveau physique. Ce travail nous a permis de maîtriser tous les problèmes d'optimisation d'une telle cible technologique. Ce savoir-faire reste acquis pour tout autre projet.

En conclusion, les perspectives que l'on peut envisager pour faire suite à cette thèse sont l'utilisation des blocs IP développés pour l'implantation logicielle/matérielle de la nouvelle norme de compression vidéo à faible débit H.264 sur notre plateforme ou sur une plateforme comportant un processeur embarqué plus puissant que le processeur NIOS II (processeur PowerPC d'un circuit Virtex de Xilinx) pour avoir plus de performances au niveau logiciel [130][131].

Enfin, le portage d'une extension Temps Réel pour Linux embarqué sur le processeur NIOS II (et pour le processeur PowerPC d'un circuit Virtex) est à envisager pour un meilleur contrôle du déterminisme concernant l'exécution de la partie logicielle…

BIBLIOGRAPHIE

[1] : F. Cappello, J.P. Sansonnet, « Introduction au Parallélisme et aux Architectures Parallèles », Techniques de l'Ingénieur, traité Electronique

[2] : P. Benoit, « Architectures des Accélérateurs de Traitement Flexibles pour les Systèmes sur Puces », Thèse, Université de Montpellier II ,11 octobre 2004.

[3] : UIT-T : Union Internationale des Télécommunications – Télécommunications, http://www.itu.ch

[4] : MPEG : Motion Picture Expert Group, http://www.mpeg.org.

[5] : Sébastien Crespin « http ://screspin.free.fr/mpeg/rapport.htm ».

[6] : E. Incerti, « Compression d'images : Algorithmes et Standards », Vuibert, Paris, 2003

[7] : G. Côté and L. Winger, « Progrès Récents dans le Domaine de la Compression Vidéo », IEEE Canadian Review Printemps 2002.

[8] : G. K. Wallace, « The JPEG Still Picture Compression Standard », IEEE Trans. on Consumer Electronics, Vol. 38, Feb. 1992.

[9] : M. Ghanbari, « Standard Codecs: Image Compression to Advanced Video Coding », Book Published by The institution of Electrical Engineers, London, 2003.

[10] : H.261 : Video Codec for Audiovisual Services at p x 64 kb/s. Recommandation H.261 à l'UIT-T. Mars 1993.

[11] : H.263 : Video Coding for Low Bit rate Communication. Première Recommandation H.263 à l'UIT-T. Mars 1996.

[12] : Standard MPEG-1 : ISO/IEC 11172-2, Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1,5 Mb/s.

[13] : Standard MPEG-2 : ISO/IEC 13818-2, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information.

[14] : Standard MPEG-4 : ISO/IEC 14496-2, Information Technology – Coding of Audio-Visual Objects.

[15] : F. Loras, J. Fournier, «H.264/MPEG-4 AVC, un Nouveau Standard de Compression Vidéo », CORESA'03, Lyon, France, 16-17 Janvier 2003.

[16] : H.263+ : Video Coding for Low Bit rate Communication. Deuxième recommandation H.263 à l'UIT-T. Février 1998.

[17] : H.263++ : H.263 Annexe U, V, W and X. Compléments de la Recommandation H.263 à l'UIT-T. Janvier 2000.

[18] : G. Côté, B. Erol, M. Gallant, and F. Kossentini, « H.263+: Video Coding at Low Bit Rates » IEEE Trans. on Circuits And Systems for Video Technology, Vol. 8, No. 7, pp.849-866, Nov. 1998.

[19] : J.L. Barron et al. « Systems and Experiment Performance of Optical Flow Technique », Int. Journal of computer vision, pp 43-76, 1994.

[20] : A.N. Netravali and J.D. Robbins, « Motion Compensated Television Coding », Bell syst. Tech. J., Vol.58, pp 631-670, Mars 1979.

[21] : M. Gallant, G. Cote, F. Kossentini, « An Efficient Computation-Constrained Block-Based Motion Estimation Algorithm for Low Bit Rate Video Coding », IEEE Trans. on Image Processing, Vol. 8, No. 12, pp. 1816-1823, Dec. 1999.

[22] : A. Docef, F. Kossentini, K. Nguuyen-Phi, I.R Ismaeil., « The Quantized DCT and Its Application to DCT-based Video Coding »**,** IEEE Trans. on Image Processing, Vol. 11, No. 3, pp. 177-187, March 2002.

[23] : La Compression des Images Numériques. Paris: Hermés, 1995.

[24] : Huffman, D.A., « A Method for the Construction of Minimum Redundancy Codes », Proceedings IRE, vol. 40, pp. 1098-1101, 1962.

[25] : G. L. J. Rissanen, « Arithmetic Coding », IEEE Trans. on Communication, June 1981.

[26] : R. M. N. I. H. Witten and J. G. Cleary, « Arithmetic Coding for Data Compression », Communication of the ACM, pp.520-540, 1987.

[27] : P. Kadionik, « Les Systèmes Embarqués : une Introduction », Linux Magazine, Hors Série 24, février-mars 2006.

[28] : ITRS Roadmap, http://public.itrs.net/

[29] : P. Kadionik, « Linux et le Système sur Silicium », Linux magasine N°11, Avril 2005, France.

[30] : A. Hanczakowski, « Système Complet sur une Puce et Réutilisation de Blocs », Techniques de l'Ingénieur, traité Electronique.

[31] : F. Gharsalli, D. Lyonnard, S. Meftali, F. Rousseau, A. A. Jerraya « Unifying Memory and Processor Wrapper Architecture in Multiprocessor SoC Design » ISSS '02, October 2-4, 2002, Kyoto, Japan.

[32] : M. Robert, « ASICs et Logiciels CAO Associés », Techniques de l'Ingénieur, traité Electronique.

[33] : Technologies SoC, http://www.rennes.supelec.fr/ren/perso/jweiss/fpga/soc_jw/index.htm

[34] : H. Kalte, D. Langen, E. Vonnahme, A. Brinkmann, and U. Rucket, « Dynamically Reconfigurable System-on-Programmable-Chip », PDP, January 2002, Gran Canaria Island, Spain.

[35] : VSI Alliance, « Architecture Document », Version 1.0, 1997, disponible à l'adresse http://www.vsi.org.

[36] : G. Savaton, E. Casseau, E. Martin, C. Lambert-Nebout, « Composants Virtuels Comportementaux pour Applications de Compression d'Image », GRETSI 01, France 2001.

[37] : Y. Huang, W. T.Cheng, « Using Embedded Infrastructure IP for SOC Post-Silicon Verification », DAC 2003, June 2-6, 2003, Anaheim, California, USA.

[38] : M. A. Dziri, « Modèles d'Intégration d'Outils et de Composants Logiciel/Matériel pour la Conception des Systèmes Hétérogènes Embarqués » Thèse, Institue National Polytechnique de Grenoble, 26 mai 2004.

[39] : F. Gohzzi, « Optimisation d'une Bibliothèque de Modules Matériels de Traitement d'Images. Conception et test vhdl, Implémentation Sous Forme FPGA », Thèse, No. 2789, Ecole doctorale de Sciences Physiques et de l'Ingénieur, Université de Bordeaux, 26 janvier 2004.

[40] : Altera, http://www.altera.com

[41] : Xilinx, http://www.Xilinx.com

[42] : D. Lewis and Al, « The Stratix II Logic and Routing Architecture », FPGA'05, February 20–22, 2005, Monterey, California, USA.

[43] : Altera Startix II Architecture
http://www.altera.com/products/devices/stratix2/st2-index.jsp

[44] : Tom Kean « Cryptographic Rights Management of FPGA Intellectual Property Cores », FPGA 2002, February 24-26, 2002, Monterey, California, USA.

[45] : J. Von Neumann, « First Draft of a Report on the EDVAC », IEEE Annals of the History of Computing, Vol. 15, No. 4, pp.27-75, 1993.

[46] : M.D. Godfrey and D.F. Hendry, « The Computer as Von Neumann Planned It », IEEE Annals of the History of Computing, Vol. 15, No. 1, pp. 11-21, 1993.

[47] : D. MADON, « Processeur RISC Multitâche », Thèse, No. 2159, Ecole Polytechnique Fédérale de Lausanne, 2000

[48] : H. Lilen, « Microprocesseurs », DUNOD, Paris, 1995.

[49] : A. Tanenbaum, « Architecture de L'Ordinateur », InterEditions, Paris, 1991.

[50] : N. Souissi, A. Ben Atitallah, F. Ghozzi, M. Ben Ayed, N. Masmoudi « Etude

Comparative de Deux Processeurs Softcores NIOS II et LEON 3 », JTEA'06, Hammamet, Tunisie, 12-14 Mai 2006.

[51] : NIOS Embedded Processor User's Guide, Altera Corporation, January 2002. http://www.altera.com/literature/lit-nio.jsp

[52] : Le processeur Microblaze, http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze

[53] : Le processeur Leon, http://www.gaisler.com/index.html.

[54] : Le processeur OpenRisc, http://www.opencores.org/projects.cgi/web/or1k/overview.

[55] : Le processeur F-CPU, http://www.f-cpu.org.

[56] : Le processeur LatticeMico32, http://www.latticesemi.com/.

[57] : Altera "NIOS Custom Instructions Tutorial", June 2002, http://www.altera.com/literature/tt/tt_nios_ci.pdf

[58] : Nios II Integrated Development Environment http://www.altera.com/products/ip/processors/nios2/ni2-index.html

[59] : P. Coussy, « Synthèse d'Interface de Communication pour les Composants Virtuels », Thèse, No. 27, LESTER, Université de Bretagne Sud, 2003.

[60] : 3GPP, http://www.3gpp.org.

[61] : Report on UMTS, « Final Draft 2.5 Enabling UMTS (3G) Services and Applications », No. 11, Report from the UMTS Forum, Juillet 2000.

[62] : Samsung, http://www.samsung.com/fr/products/mobilephone/multimedia/sgh_z230wraftm.asp

[63] : LG, http://www.agence.francetelecom.com/mx/?tp=F&ref=15161&IDCible=1&type=4&donnee_appel=FTASN&id=194531171532215&sv=5-184713_B#oc

[64] : Sagem, http://www.agence.francetelecom.com/mx/?tp=F&ref=15121&IDCible=1&type=4&donnee_appel=FTASN&id=194531171532215&sv=5-184713_B#oc

[65] : Toshiba, http://www.agence.francetelecom.com/mx/?tp=F&ref=19001&IDCible=1&type=4&donnee_appel=FTASN&id=194531171532215&sv=5-184713_B#oc

[66] : F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, H. De Man, « Global Communication and Memory Optimizing Transformations for Low Power Signal Processing Systems », IEEE workshop on VLSI signal processing, Vol. 7, pp. 178-

187, Oct 1994.

[67] : W. Mangione-Smitn et al., « A Low Power Architecture for Wireless Multimedia Systems: lessons learned from building a power hog », ISLPED, pp 23-28, Monterey, 1996.

[68] : M. Abid, « Contribution à la Conception des Systèmes Electroniques Mixtes Logiciels/Matériels », Thèse, Ecole Nationale d'Ingénieurs de Tunis, 2000.

[69] : J. P. Jamont, « Diamond: Une Approche pour La Conception de Systèmes Multi-agents Embarqués», Thèse, Institut National Polytechnique de Grenoble, 2005.

[70] : Linux, http://www.linux.org/

[71] : P. Kadionik, C. Consel, A. Ben Atitallah, P. Nouel, L. Burgy, N. Palix, W. Jouve. J. Lancia «The HomeSIP Project: Using Open Source SIP Stack for Home Automation », FOSDEM'07, Bruxelles, Belgique, 24-25 février 2007

[72] : P. Ficheux, « Linux Embarqué », Edition Eyrolles, 2002.

[73] : P. Kadionik, « Linux embarqué : le pari réussi d'une PME bordelaise », RTS 2006, Embedded Systems Conférence, Paris, France, 5-7 avril 2006.

[74] : P. Kadionik, « La Mise en Œuvre de Linux pour l'Embarqué », Linux Magazine, Hors Série 25, avril-mai 2006.

[75] : P. Kadionik, P. Nouel, A. Ben Atitallah, «Linux et le System on Chip SoC », LSM '05, Dijon, France, 5-9 juillet 2005

[76] : The µClinux project http://www.uClinux.org.

[77] : Microtronix, http://www.microtronix.com/.

[78] : Le portage µClinux pour le processeur NIOS II, www.niosforum.com/forum.

[79] : Logiciel Quartus II d'Altera, http://www.altera.com/products/software/products/quartus2/qts-index.html.

[80] : P. Kadionik, P.Nouel, A. Ben Atitallah, P. Dondon « L'Enseignement des Systèmes Numériques », CETSIS '05, Nancy, France, 25-27 Octobre 2005.

[81] : IDE Eclipse, http://www.eclipse.org/.

[82] : Nios II Development Kit, Stratix II Edition, ALTERA 2006, http://www.altera.com/products/devkits/altera/kit-niosii-2S30.html

[83] : A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi, Ph.Marchegay « Hardware Platform Design for Real-Time Video Applications », in Proc. IEEE ICM'04, pp. 722-725, Dec. 2004.

[84] : Lancelot Home Page, « VGA video controller », http://www.fpga.nl/lancelot.html.

[85] : A. Ben atitallah, P. Kadionik, P. Nouel, «Système d'acquisition vidéo à base d'opencores et de µClinux », LSM '05, Dijon, France, 5-9 juillet 2005

[86] : PULNiX « TM-9701 Progressive Scanning Full Frame Shutter Camera » Data Sheet.

[87] : National Semiconductor « DS26C32AT/DS26C32AM Quad Differential Line Receiver » Data Sheet, juin 1998.

[88] : A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi, Ph.Marchegay « Real-Time video system Design based on the NIOS II Processor and µClinux », IP/SOC'05, Grenoble, France, 7-8 Décembre 2005.

[89] : Telenor Research Development, Norvège, http://www.telenor.com/rd/

[90] : Z. Wang, A. C. Bovik, H. R. Sheikh et E. P. Simoncelli, « Image Quality Assessment: From Error Visibility to Structural Similarity », IEEE Trans. On Image Processing, Vol. 13, No. 4, pp. 600-612, April 2004.

[91] : S. Wong, B. Stougie, S. Cotofana, « Alternatives in FPGA-based SAD Implementations », in Proc. IEEE FTP'02, pp. 449-452, Dec. 2002.

[92] : S. Saponara and L. Fanucci, « Data-adaptive Motion Estimation Algorithm and VLSI Architecture Design for Low-power Video Systems », IEE Proc. Computers and Digital Techniques, Vol. 151, No. 1, pp. 51-59, Jan 2004.

[93] : S. Zhu and K. K. Ma, « A New Diamond Search Algorithm for Fast Block Matching Motion Estimation », IEEE Trans. Image Process., Vol. 9, No. 2, pp. 287-290, Feb. 2000.

[94] : C. Zhu, X. Lin, and L. P. Chau, « Hexagon-Based Search Pattern for Fast Block Motion Estimation », IEEE Trans. On Circuits And Syst. For Video Technology, Vol. 12, pp. 349-355, May 2002.

[95] : C. H. Cheung and L. M. Po, « A Novel Cross-Diamond Search Algorithm for Fast Block Motion Estimation », IEEE Trans. Circuits Syst. Video Technol., Vol. 12, No. 12, pp. 1168-1177, Dec. 2002.

[96] : J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, « A Novel Unrestricted Center-Biased Diamond Search Algorithm for Block Motion Estimation », IEEE Trans. Circuits Syst. Video Technol., Vol. 8, pp. 369-377, Aug. 1998.

[97] : A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi, H. Levi « An efficient HW/SW Implementation of the H.263 Video Coder in FPGA », IEEE ICECS '06, Nice, France, 10-13 December 2006.

[98] : James F. Blinn, « What's the Deal with the DCT? », IEEE Tran. On Computer Graphics and Applications, Vol. 13, No. 4, pp. 78-83, July. 1993.

[99] : K. Kim and J. S. Koh, « An Area Efficient DCT Architecture For MPEG-2 Video Encoder », IEEE Trans. On Consumer Electronics, Vol. 45, No. 1, pp. 62-67, Feb. 1999.

[100] : R. Endrigo, C. Porto and L. V. Agostini, « Project Space Exploration on the 2-D DCT Architecture of a JPEG Compressor Directed to FPGA Implementation », in Proc. DATE'04, Vol. 3, pp. 224-229, Feb. 2004.

[101] : H. Lim, Member, V. Piuri and Earl E. Swartzlander « A Serial-Parallel Architecture for Two-Dimensional Discrete Cosine and Inverse Discrete Cosine Transforms », IEEE Trans. on Computers, Vol. 49, No. 12, pp. 1297-1309, Dec. 2000.

[102] : Hyuk-Jae Lee Jae-Beom Lee and Hyeran Byun « Comparisons of Fast 2D-DCT Algorithms for Parallel Programmable Digital Signal Processors », in Proc. HPC-Asia '97, pp. 209-213, May 1997.

[103] : W.c Chen, C.h Smith and S.C. Fralick, « A fast Computational Algorithm for thr Discrete Cosine Transform », IEEE Trans. On Communications, Vol. 25, No. 9, pp.1004-1009, Sept.1997.

[104] : M. D. Wagh and H. Ganesh, « A new algorithm for the discrete cosine transform of arbitrary number of points », IEEE Trans. On Computers, Vol. 29, No. 4, pp. 269-277, Apr. 1980.

[105] : B. G. Lee, « A new algorithm to compute the discrete cosine transform », IEEE Trans. on Signal Processing, Vol. 32, No. 6, pp. 1243-1245, Dec. 1984.

[106] : H. Malvar, « Fast Computation of Discrete Cosine Transform Through Fast Hartley Transform », Electron. Lett., Vol. 22, No. 7, pp. 352–353, Mars 1986.

[107] : Y. Chan and W. Siu, « A Cyclic Correlated Structure for the Realization of Discrete Cosine Transform », IEEE Trans. Circuits and Systems, Vol. 39, No.2, pp. 109–113, Feb. 1992.

[108] : C. Loeffler and A. Lightenberg, « Practical Fast 1-D DCT Algorithms With 11 Multiplications », in Proc. IEEE ICASSP '89, Vol. 2, pp. 988-991, May 1989.

[109] : T.J. van Eijndhven and F.W. Sijstermans, « Data Processing Device and method of Computing the Cosine Transform of a Matrix », PCT Patent WO 99948025, to Koninklijke Philips Electronics, World Intellectual Property Organization, International Bureau, 1999.

[110] : Rohini Krishnan et al., « Design of a 2D DCT/IDCT Application Specific VLIW Processor Supporting Scaled and Sub-sampled Blocks », in proc. VLSI'03, pp. 177-182, Jan. 2003.

[111] : Radhika S. Grover, Weijia Shang and Qiang Li, « A Faster Distributed Arithmetic Architecture for FPGAs », FPGA'02, February 24-26, 2002, Monterey, California, USA.

[112] : S. Khawam, T. Arslan and F. Westall, « Synthesizable Reconfigurable Array Targeting Distributed Arithmetic for System-on-Chip Applications », in Proc. PDPS'04, pp. 150, April 2004.

[113] : Y. H. Chan, W. C. Siu, « On The Realization of Discrete Cosine Transform Using the Distributed Arithmetic », IEEE Trans. on Circuits And Systems, Vol. 39, No. 9, pp. 705-712, Sept. 1992.

[114] : Sungwook Yu and Earl E. Swartzlande, « DCT Implementation with Distributed Arithmetic », IEEE Trans. on Computers, Vol. 50, No. 9, pp. 985-991, Sept. 2001.

[115] : A. Ben Atitallah, P. Kadionik, F. Ghozzi, P. Nouel, « Optimization and implementation on FPGA of the DCT/IDCT algorithm », in Proc. IEEE ICASSP'06, vol. 3, pp. 928-931, May 2006.

[116] : T.S.Chang, C.S.Kung, and C.W. Jen, « A Simple Processor Core Design for DCT/IDCT », IEEE Trans. On Circuits and Systems For Video Technology, Vol. 10, No. 3, pp. 439-447, April 2000.

[117] : H. Lim, V. Piuri and E. E. Swartzlander, « A Serial-Parallel Architecture for Two Dimensional Discrete Cosine and Inverse Discrete Cosine Transforms », IEEE Trans. On Computers, Vol. 49, No. 12, pp. 1297-1309, Dec. 2000.

[118] : Kuan-Hung Chen, Jiun-In Guo, Jinn-Shyan Wang, and Ching-Wei Yeh « A Power-aware SNR-Progressive DCT/IDCT IP Core Design for Multimedia Transform Coding », in Proc. IEEE ICME'04, Vol. 3, pp. 1683-1686, June 2004.

[119] : IEEE Std 1180-1990, « IEEE Standard Specification for the Implementation of 8x8 Inverse Cosine Transform », Institute of Electrical and Electronics Engineers, New York, USA, International Standard, Dec. 1990

[120] : N. J. August and Dong Sam Ha, « Low Power Design of DCT and IDCT for Low Bit Rate Video Codec », IEEE Trans. On Multimedia, Vol. 6, No. 3, pp. 414-422, June 2004.

[121] : A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi, H. Levi « HW/SW Codesign of the H.263 video coder », IEEE CCECE '06, Ottawa, Canada, 7-10 Mai 2006.

[122] : ITU Telecom. Standardization Sector of ITU, « Video Codec Test Model Near-Term, Version 8 (TMN8), Release 0 », H.263 Ad Hoc Group, June 1997.

[123] : Proposal for Test Model Quantization Description, ITU-T doc. Q15-D-30, Apr. 1998.

[124] : A. Ben Atitallah, P. Kadionik, F. Ghozzi, P.Nouel, N. Masmoudi, H. Levi « An FPGA Implementation of HW/SW Codesign Architecture for H.263 Video Coding », International Journal of Electronics and Communications, 2006.

[125] : Sang-hee Lee; Myungjin Kim; Keun-Bae Kim « Modular and Efficient Architecture for H.263 Video Codec VLSI », in Proc. IEEE ISCAS'02, Vol. 5, pp. 125-128, May 2002.

[126] : S. M. Akramullah, I. Ahmad and M. L. Liou. « Optimization of H.263 Video Encoding Using a Single Processor Computer: Performance Tradeoffs and Benchmarking ». IEEE Trans. on Circuits and Systems for Video Technology, Vol. 11, No. 8, pp. 901-915, Aug. 2001.

[127] : Thinh PQ Nguyen, Avideh Zakhor, Kathy Yelick « Performance Analysis of an H.263 Video Encoder for VIRAM », in Proc. IEEE ICIP, Vol. 3, pp. 98-101, Sept. 2000.

[128] : K. T. Shih, C. Y. Tsai, H.M. Hang, « Real-time Implementation of H.263+ Using TI TMS320C6201 Digital Signal Processor », in Proc. IEEE ISCAS'03, Vol. 2, pp. 900-903, May 2003.

[129] : M. Harrand, J. Sanches, A. Bellon, J. Bulone, A. Tournier. « A Single Chip CIF 30-Hz, H261, H263 and H263+ Video Encoder/Decoder with Embedded Display Controller ». IEEE Journal of Solid-State Circuits, Vol. 34, No. 11, pp. 1627-1633, Nov. 1999.

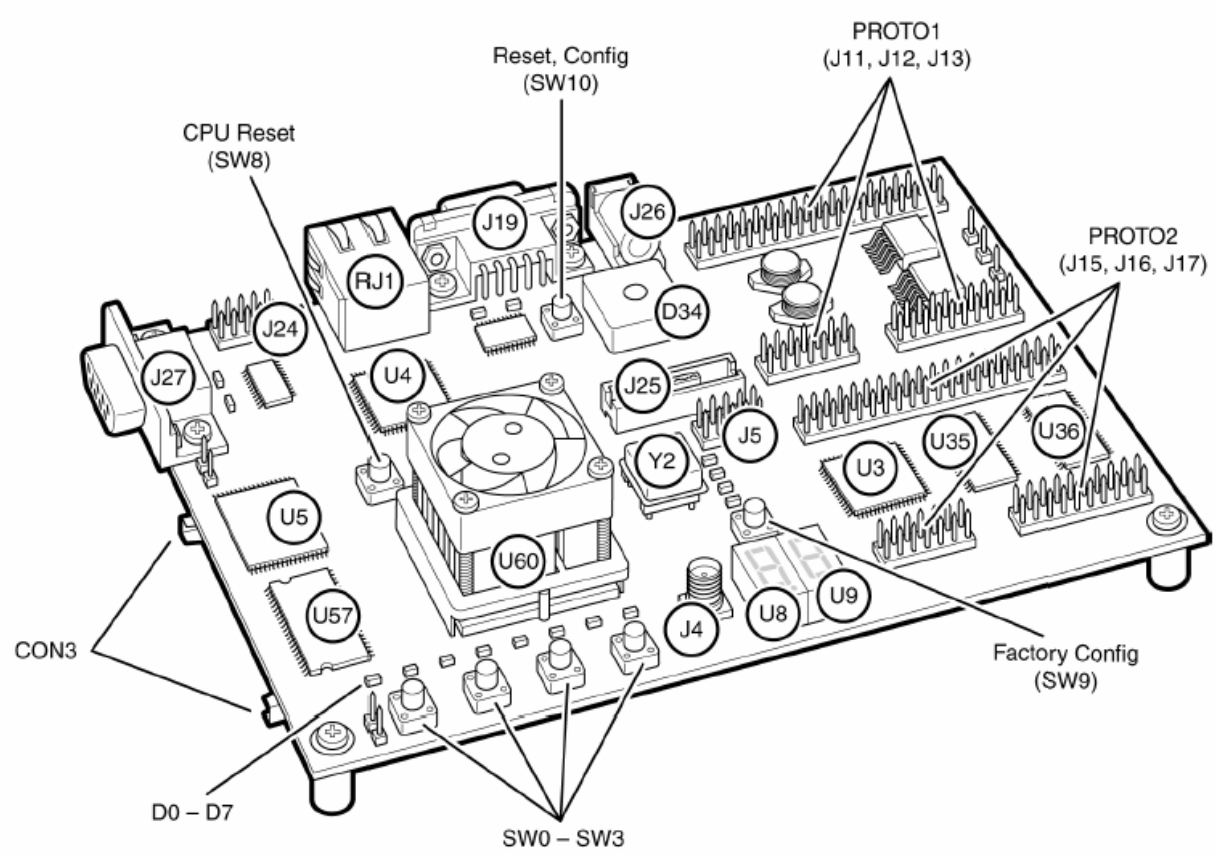
[130] : A. Ben Atitallah, T. Damak, I. Warda, P. Kadionik,N. Masmoudi « Efficient Implementation of the H.264 Video Encoder on Embedded Processor », IEEE SSD '07, Hammamet, Tunisie, 19-22 Mars 2007.

[131] : T. Damak, A. Ben Atitallah, I. Werda, N. Masmoudi « Implantation de la chaîne de codage de la norme H.264 sur un processeur embarqué », GEI '07, Monastir, Tunisie, 26-28 Mars 2007.

ANNEXES

Annexe A

Synoptique de la carte Altera Stratix II



Annexe B

Caractéristiques de la Camera PULNiX TM-9701

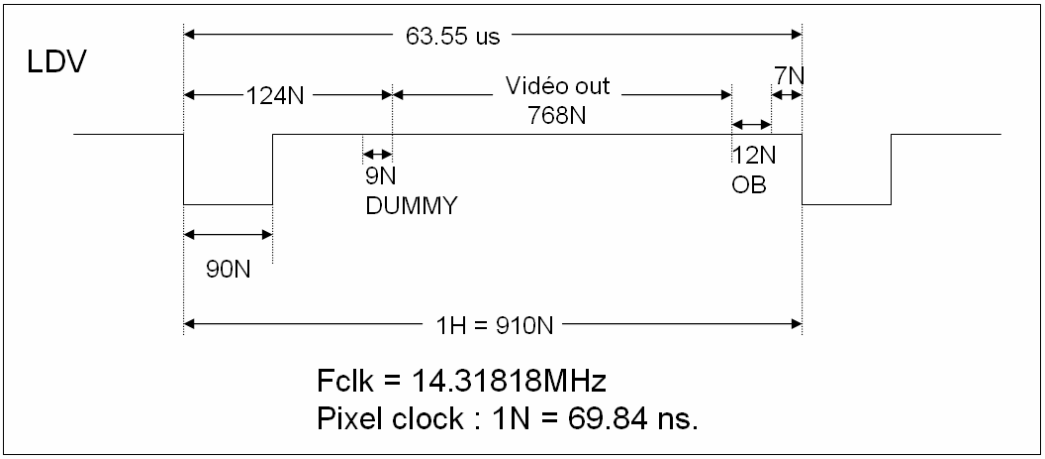
Imager:	2/3″ inch progressive scanning interline transfer CCD
Nombre pixels	768 (H) x 484 (V)
Cell size	11.6 (H) x 13.6 (V) microns progressive scan
Imager size	8.9 (H) x 6.6 (V) mm
Chip size	9.9 (H) x 7.7 (V) mm
Dynamic range	60 dB
Output sensitivity	10 µV/electron
Dark noise	40 electrons
Dark current	< 1 nA/cm^2

Scanning:	525 lines, 30 Hz or 60 Hz 2:1 interlace.
Clock	28.6363 MHz
Pixel clock	14.31818 MHz
Horizontal freq.	15.734 KHz
Vertical freq.	59.94 Hz

Video output	1V p-p composite video, 75Ω and 8-bit RS-422 output
S/N ratio	50 dB min. (AGC OFF)
AGC	ON/OFF (OFF std.)
Power requirement	DC 12V, 500 mA

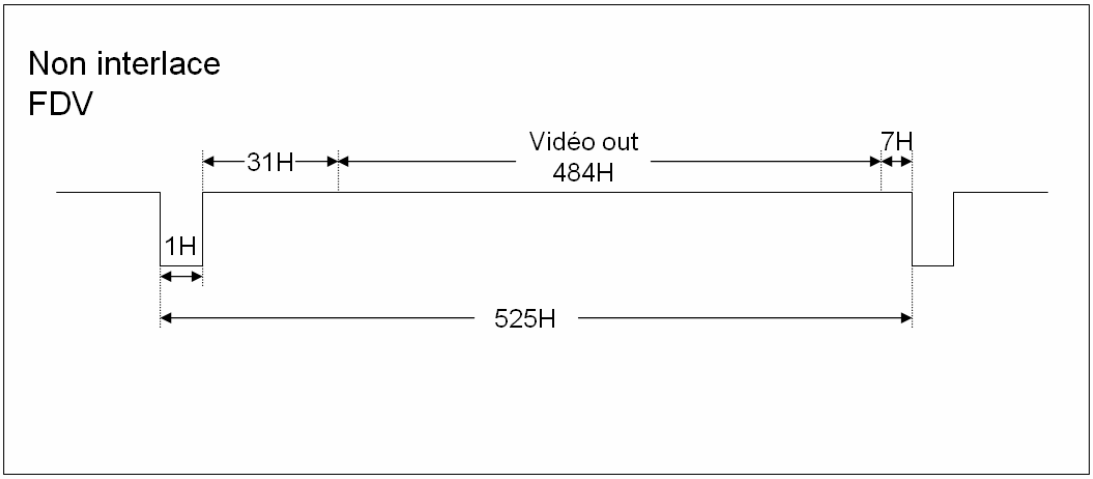
Line Data Valid:

La figure ci-dessous représente le chronogramme du signal LDV (*Line Data Valid*). Ce signal est au niveau haut lors du transfert des données d'une ligne d'image. La durée d'une ligne est de 63,55 µs (H = 910x69, 84).



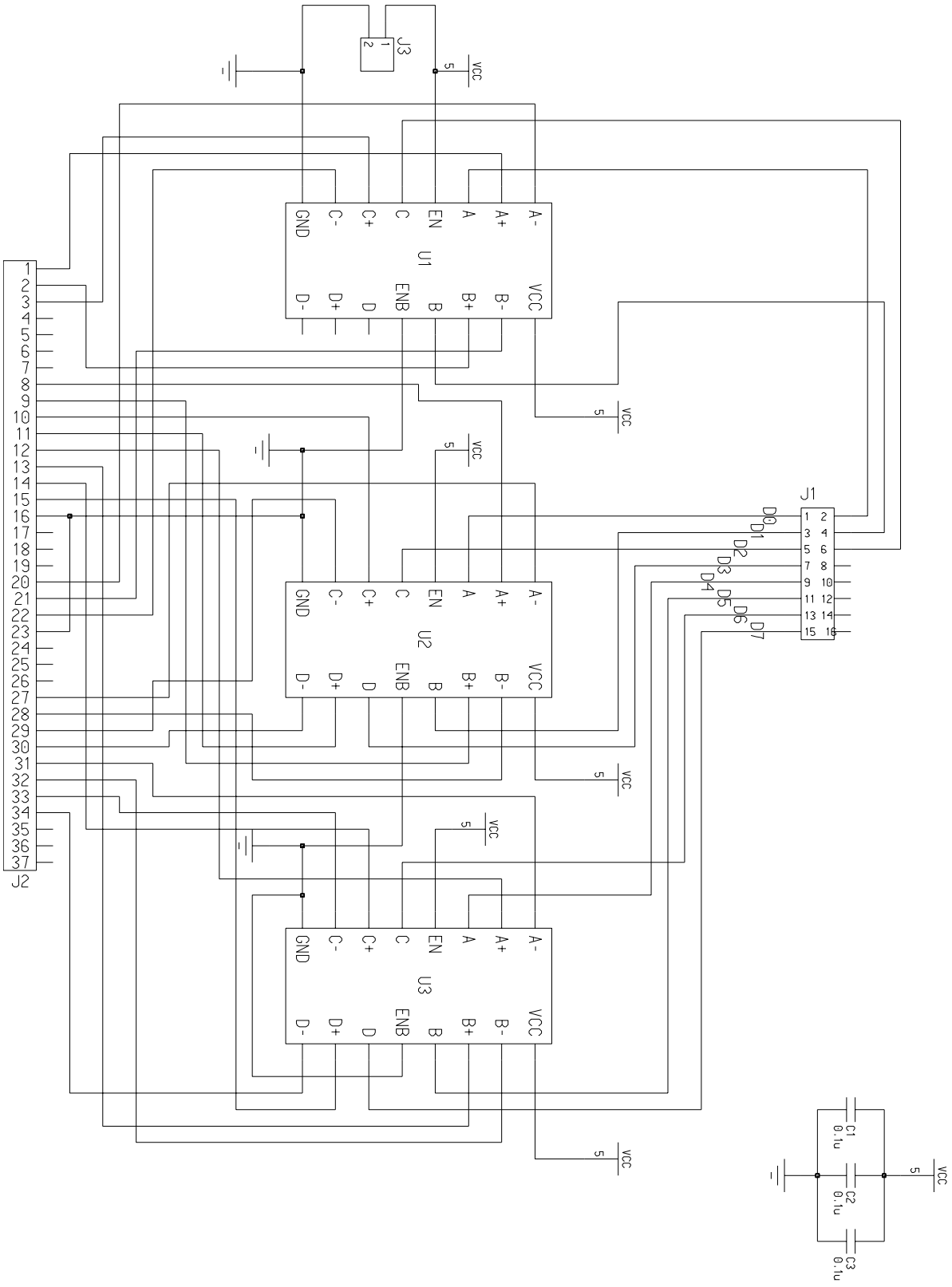
Frame Data Valid:

La figure ci-dessous représente le chronogramme du signal FDV (*Frame Data Valid*). Ce signal est au niveau haut lors du transfert des données d'une image. La durée d'une image est 525 H.



Annexe C

Schéma de la carte d'interface caméra



Annexe D

Description VHDL de l'interface caméra

Fichier camera_nios.vhd :

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL;


ENTITY camera_nios IS
  PORT(
    reset_n            : IN  std_logic;   -- reset NIOS
    clk_system               : IN  std_logic;   -- horloge NIOS
    clk_14Mhz          : IN  std_logic;   -- horloge de la caméra
    ldv                           : IN  std_logic;        -- line data valid
    fdv                           : IN  std_logic;        -- frame data valid
    pixel_cam                : IN  std_logic_vector(7 DOWNTO 0); --pixel lu de la camera
    -- Avalon slave
    rd,wr              : IN  std_logic;    --lecture/ecriture
    address            : IN  std_logic_vector(1 DOWNTO 0);    -- 3 registres
    cs                        : IN  std_logic;    -- sélection le périphérique
    data_in                      : IN std_logic_vector(31 DOWNTO 0);
    data_out           : OUT std_logic_vector(31 DOWNTO 0);  -- données à lire
    -- Avalon master
    master_waitreq        : IN std_logic;
    master_addr               : OUT std_logic_vector(31 DOWNTO 0);
    master_wrdata         : OUT std_logic_vector(31 DOWNTO 0);
    master_wr          : OUT std_logic;
    --
    vga_read                 : IN std_logic);
END ;


ARCHITECTURE driver OF camera_nios IS

  COMPONENT interface_cam
    PORT(
    reset_n        : IN  std_logic;
    clk_14Mhz   : IN  std_logic;   -- horloge de la caméra
    ldv                : IN  std_logic;   -- line data valid
    fdv                : IN  std_logic;   -- frame data valid
    pixel_cam   : IN  std_logic_vector(7 DOWNTO 0); --pixel lit de la camera
    frontfdv    : out std_logic;
    pixel32_valid  : OUT std_logic;  --  impulsion d'une periode
    pixel32     : OUT std_logic_vector(31 DOWNTO 0)  -- 4 pixels concatenes
    );
  END COMPONENT;

  COMPONENT camera_avalon
     PORT(
    clk             : IN std_logic;
    reset_n            : IN std_logic;
-- Avalon Master Port
    master_addr           : OUT std_logic_vector(31 DOWNTO 0);
    master_wrdata   : OUT std_logic_vector(31 DOWNTO 0);
    master_wr      : OUT std_logic;
    master_waitreq  : IN std_logic;

-- DMA Interface
    dma_address     : IN std_logic_vector(31 DOWNTO 0);
    dma_data       : IN std_logic_vector(31 DOWNTO 0);
    dma_read       : OUT std_logic;
    dma_request     : IN std_logic;
    dma_transfer_end : IN std_logic;
    dma_first      : IN std_logic
    );
  END COMPONENT;

  COMPONENT dcfifo
```

```
    GENERIC (  intended_device_family : string;
                lpm_width             : natural;
                lpm_numwords          : natural;
                lpm_widthu            : natural;
                clocks_are_synchronized      : string;
                lpm_type                : string;
                lpm_showahead         : string;
                overflow_checking     : string;
                underflow_checking    : string;
                use_eab                     : string;
                lpm_hint                : string);
    PORT ( wrclk     : IN std_logic ;
           wrreq     : IN std_logic ;
           data            : IN std_logic_vector (31 DOWNTO 0);
           rdempty   : OUT std_logic ;
           wrusedw   : OUT std_logic_vector (8 DOWNTO 0);
          rdusedw      : OUT std_logic_vector (8 DOWNTO 0);
           rdclk     : IN std_logic ;
           rdreq     : IN std_logic ;
           q        : OUT std_logic_vector (31 DOWNTO 0));
  END COMPONENT;

  -- camera
  SIGNAL pixel32_valid,frontfdv      : std_logic;
  SIGNAL pixel32            : std_logic_vector(31 DOWNTO 0);
  -- DMA
  SIGNAL dma_address        : std_logic_vector(31 DOWNTO 0);
  SIGNAL dma_data           : std_logic_vector(31 DOWNTO 0);
  SIGNAL dma_read,dma_request,dma_first,start:std_logic;
  SIGNAL reg                      : std_logic_vector(1 DOWNTO 0);
  -- FIFO
  SIGNAL wrreq                    : std_logic;
  SIGNAL rdempty                      : std_logic;     -- indicateur de fifo vide
  SIGNAL wrusedw,rdusedw          : std_logic_vector(8 DOWNTO 0);
  -- control register
  SIGNAL control_reg_cs    : std_logic;
  SIGNAL control_reg       : std_logic_vector(31 DOWNTO 0);
  SIGNAL start_video       : std_logic;
  -- status register
  SIGNAL status_reg_cs     : std_logic;
  SIGNAL status_reg        : std_logic_vector(31 DOWNTO 0);
  -- DMA register
  SIGNAL dma_address_reg_cs: std_logic;
  SIGNAL dma_address_reg   : std_logic_vector(31 DOWNTO 0);
  -- Pixel Counter
  SIGNAL threshold_reg_cs : std_ulogic;
  SIGNAL threshold_reg    : std_logic_vector(31 DOWNTO 0);
                                    -- seuil de reglage de la fifo
  CONSTANT ligne : natural := 640/4;    -- en mots de 32 bits

BEGIN

  interface_cam_component : interface_cam
    PORT MAP(reset_n=>reset_n,
             clk_14Mhz=>clk_14Mhz,
              ldv=>ldv,
              fdv=>fdv,
              pixel_cam=>pixel_cam,
              frontfdv=>frontfdv,
              pixel32_valid=>pixel32_valid,
              pixel32=>pixel32);

  dma_component :camera_avalon
    PORT MAP (clk =>clk_system,
              reset_n =>reset_n,
              master_waitreq => master_waitreq,
              master_addr =>master_addr,
              master_wrdata =>master_wrdata,
              master_wr =>master_wr,
              dma_address => dma_address,
              dma_data => dma_data,
              dma_read => dma_read,
              dma_request=> dma_request,
              dma_transfer_end=>rdempty,
              dma_first =>dma_first);
```

```
  dcfifo_component : dcfifo
    GENERIC MAP (
      intended_device_family => "STRATIX II",
      lpm_width => 32,
      lpm_numwords => 320,
      lpm_widthu => 9,
      clocks_are_synchronized => "FALSE",
      lpm_type => "dcfifo",
      lpm_showahead => "OFF",
      overflow_checking => "ON",
      underflow_checking => "ON",
      use_eab => "ON",
      lpm_hint => "")
    PORT MAP (
      wrclk => clk_14Mhz,
      wrreq => wrreq,
      data => pixel32,
      rdempty => rdempty,
      wrusedw => wrusedw,
      rdusedw=>rdusedw,
      rdclk => clk_system,
      rdreq => dma_read,
      q => dma_data);

-------------------------------
-- Address Select Generation --
-------------------------------
-- Address Map
-------------------------------
-- 0 : Control Register
-- 1 : Status Register
-- 2 : DMA Register
-- 3: Threshold register
-------------------------------
  control_reg_cs             <= '1' WHEN cs = '1' AND address = "00" ELSE '0';
  status_reg_cs                     <= '1' WHEN cs = '1' AND address = "01" ELSE '0';
  dma_address_reg_cs  <= '1' WHEN cs = '1' AND address = "10" ELSE '0';
  threshold_reg_cs           <= '1' WHEN cs = '1' AND address = "11" ELSE '0';
--------------------
-- Data output mux --
--------------------
  data_out <= control_reg WHEN (control_reg_cs='1'  AND rd ='1') ELSE
              status_reg WHEN (status_reg_cs='1'    AND rd ='1') ELSE
               dma_address_reg  WHEN (dma_address_reg_cs     = '1' AND rd ='1') ELSE
               threshold_reg WHEN (threshold_reg_cs='1'     AND rd ='1') ELSE
               (OTHERS => 'Z');

---------------------------------
-- Control Register (write only) --
---------------------------------
-- 0 : Start Video
---------------------------------
  control_rg:PROCESS(clk_system, reset_n,control_reg_cs)
  BEGIN
    IF reset_n = '0' THEN
      control_reg <= (OTHERS => '0');
    ELSIF (clk_system'event AND clk_system = '1') THEN
      IF (control_reg_cs='1' AND wr='1') THEN
        control_reg <= data_in;
      END IF;
    END IF;
  END PROCESS;
start_video<=control_reg(0);

--------------------------------
-- Status Register (read only) --
--------------------------------
-- 2          : FIFO plein/vide
-- 1          : Horizontal Sync
-- 0          : Vertical Sync
--------------------------------
  status_rg:PROCESS(clk_system, reset_n)
  BEGIN
    IF reset_n = '0' THEN
      status_reg <= (OTHERS => '0');
    ELSIF (clk_system'event AND clk_system = '1') THEN
      status_reg(3) <= start_video;
```

```
      status_reg(2) <= NOT rdempty;
      status_reg(1) <= ldv;
      status_reg(0) <= fdv;
    END IF;
  END PROCESS;

------------------------------
-- DMA Address Register (rw) --
------------------------------
  dma_rg:PROCESS(clk_system, reset_n,dma_address_reg_cs)
  BEGIN
    IF reset_n = '0' THEN
      dma_address_reg <= (OTHERS => '0');
    ELSIF (clk_system'event AND clk_system = '1') THEN
      IF (dma_address_reg_cs='1' AND wr='1') THEN
        dma_address_reg <= data_in;
      END IF;
    END IF;
  END PROCESS;
  dma_address <= dma_address_reg;

--------------------------------
-- initialisation de la seuil --
--------------------------------
  seuil_fifo:PROCESS
  BEGIN
    WAIT UNTIL rising_edge(clk_system);
    IF reset_n = '0'  THEN
      threshold_reg <= std_logic_vector(to_unsigned(ligne,32));
    ELSIF (threshold_reg_cs='1' AND wr='1') THEN
      threshold_reg <= data_in;
    END IF;
  END PROCESS;

------------------------------
-- pixel_valid State Machine --
------------------------------
  pixel_st:PROCESS(clk_system, reset_n)
  BEGIN
    IF reset_n = '0' THEN
      start<='0';
    ELSIF (clk_system'event AND clk_system = '1') THEN
          IF frontfdv='1' and start_video = '1'  then
             start<='1';
           ELSIF start_video = '0'  then
                start<='0';
          END IF;

    END IF;
  END PROCESS;

wrreq <= '1' WHEN (start='1' AND pixel32_valid= '1' and start_video='1') ELSE '0';

----------
-- DMA  --
----------
     dma_first <= '1' WHEN frontfdv='1' ELSE '0';
 -- dma_first=1: initilisation par l'adresse de debut pour ecrire dans la mémoire à chaque
debut de frame.

 dma_request<='1' WHEN (rdempty='0' and vga_read='0')ELSE '0';
-- rdempty=0 : it has pixel in FIFO.
-- vga_read=0: vga not busy.

END driver;
```

Fichier interface_cam.vhd :

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY interface_cam IS
  PORT(
```

```
    reset_n    : IN  std_logic;
    clk_14Mhz   : IN  std_logic;   -- horloge de la caméra
    ldv         : IN  std_logic;   -- line data valid
    fdv         : IN  std_logic;   -- frame data valid
    pixel_cam   : IN  std_logic_vector(7 DOWNTO 0); --pixel lit de la camera
    frontfdv    : out std_logic;
    pixel32_valid  : OUT std_logic;  --  impulsion d'une periode
    pixel32     : OUT std_logic_vector(31 DOWNTO 0)  -- 4 pixels concatenes
    );
END ;


ARCHITECTURE non_entrelace OF interface_cam IS

  CONSTANT Nbr_pix_ligne         : natural:= 640;
  CONSTANT Nbr_lignes            : natural:= 480;
  SIGNAL video_ON_h, video_ON_v  : std_logic;
  SIGNAL front_fdv, front_ldv    : std_logic;    -- fronts descendants
  SIGNAL blank_n                 : std_logic;              -- 1 valide pixel

BEGIN
   fronts: PROCESS(reset_n,clk_14Mhz)
    VARIABLE x,y: std_ulogic_vector(7 DOWNTO 0);
    VARIABLE zero_f,zero_l: std_logic;
  BEGIN  -- PROCESS fronts
if reset_n='0' then
zero_f:='0';
zero_l:='0';
elsif (clk_14Mhz'event AND clk_14Mhz = '1') then
    x := ldv & x(7 DOWNTO 1);                    -- decalage droite
    y := fdv & y(7 DOWNTO 1);                    -- decalage droite
    front_ldv <= '0';
    front_fdv <= '0';

    IF (x = "11111111" and (zero_l = '0') )  THEN
      zero_l := '1';
    ELSIF (x = "00000000" and (zero_l = '1'))  THEN
       zero_l := '0';
       front_ldv <= '1';
    END IF;

    IF (y = "11111111" and (zero_f = '0'))  THEN
      zero_f := '1';
    ELSIF (y = "00000000" and (zero_f = '1'))  THEN
       zero_f := '0';
       front_fdv <= '1';
    END IF;
    end if;
  END PROCESS fronts;

-- Détermination des pixels utiles
-- Image de taille 640x480
  detect: PROCESS
    VARIABLE c_h : natural;
    VARIABLE c_v : natural;
  BEGIN
    WAIT UNTIL rising_edge(clk_14Mhz);
    IF front_fdv = '1' THEN
      c_v:=0;
    END IF;

    IF front_ldv = '1' THEN
      c_h := 1;
     c_v := c_v + 1;  --compte nombre de ligne
    ELSE
      c_h := c_h + 1; --compte nombre de pixel par ligne
    END IF;

    IF c_h > 115 AND c_h < Nbr_pix_ligne + 117  THEN
      video_ON_h <= '1';
    ELSE
      video_ON_h <= '0';
    END IF;

    IF c_v > 31 AND c_v < Nbr_lignes + 32 THEN
      video_on_v <= '1';
    ELSE
```

```
      video_ON_v <= '0';
    END IF;
  END PROCESS detect;


-- 4 pixels sont concaténés
  pixels:PROCESS
    VARIABLE c :natural;
    VARIABLE  pixels : std_logic_vector(31 DOWNTO 0);  -- pour pixel32
  BEGIN
    WAIT UNTIL rising_edge(clk_14Mhz);
    IF blank_n = '1' THEN
      pixels := pixel_cam & pixels(31 DOWNTO 8) ;
      IF c < 3 THEN
        c := c+1;
        pixel32_valid<= '0';
      ELSE
        c:=0;
        pixel32_valid<= '1';
        pixel32 <= pixels;
      END IF;
    ELSE
      c:=0;
      pixel32_valid<= '0';
    END IF;

  END PROCESS pixels;

  blank_n <= video_ON_v AND video_ON_h ;
  frontfdv<=front_fdv;

END non_entrelace;
```

Fichier camera_avalon.vhd :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--USE ieee.numeric_std.ALL;
USE ieee.std_logic_unsigned.all;

ENTITY camera_avalon IS
  PORT(
    clk             : IN std_logic;
    reset_n           : IN std_logic;
-- Avalon Master Port
    master_addr           : OUT std_logic_vector(31 DOWNTO 0);
    master_wrdata   : OUT std_logic_vector(31 DOWNTO 0);
    master_wr        : OUT std_logic;
    master_waitreq  : IN std_logic;

-- DMA Interface
    dma_address     : IN std_logic_vector(31 DOWNTO 0);
    dma_data       : IN std_logic_vector(31 DOWNTO 0);
    dma_read       : OUT std_logic;
    dma_request     : IN std_logic;
    dma_transfer_end : IN std_logic;
    dma_first      : IN std_logic
    );
END ;

ARCHITECTURE behavior OF camera_avalon IS

  SIGNAL state : std_logic_vector(1 DOWNTO 0);
  SIGNAL dma_address_reg : std_logic_vector(31 DOWNTO 0);
  SIGNAL  dma_address_inc : std_logic;

BEGIN

-------------------
-- Avalon Master --
-------------------
-- The Avalon master reads video data from the external SRAM and stores it in the Lancelot
FIFO's
PROCESS(clk, reset_n)
      BEGIN
```

```
    IF reset_n = '0' THEN
      state <= "00";
    ELSIF (clk'event AND clk = '1') THEN
      IF state = "00" THEN
        IF dma_request = '1' THEN
          state <= "01";
        END IF;
      ELSIF  state = "01" THEN
         state <= "11";
      ELSIF dma_transfer_end ='1' AND master_waitreq = '0' THEN
        state <= "00";
      END IF;
    END IF;
  END PROCESS;

  master_addr <= dma_address_reg;
  master_wrdata<=dma_data;
  dma_address_inc <=  '1' WHEN master_waitreq = '0' AND state = "11" ELSE '0';
  dma_read<=  '1' WHEN (master_waitreq = '0' AND state = "11")OR (state = "01") ELSE '0';
  master_wr <= '1' WHEN state = "11" ELSE '0';

-- DMA address register
  PROCESS(clk, reset_n,dma_address_inc, dma_address)
  BEGIN
    IF reset_n = '0' THEN
      dma_address_reg <= (others => '0');
    ELSIF (clk'event AND clk = '1') THEN
      IF dma_first = '1' THEN
        dma_address_reg <= dma_address;
      ELSIF dma_address_inc = '1' THEN
          dma_address_reg <= dma_address_reg + "100";
      END IF;
    END IF;
  END PROCESS;

END behavior;
```

Annexe E

Connexions de la carte Lancelot à la carte Stratix II

Connecteur J16	Nom du signal	Nom de la broche
J16-27	B[0]	J18
J16-28	B[1]	D18
J16-29	B[2]	C22
J16-32	B[3]	A24
J16-31	B[4]	B22
J16-33	B[5]	B24
J16-36	B[6]	H17
J16-35	B[7]	K17
J16-3	G[0]	H15
J16-4	G[1]	J15
J16-5	G[2]	C16
J16-6	G[3]	A17
J16-7	G[4]	C17
J16-8	G[5]	A18
J16-9	G[6]	F17
J16-10	G[7]	K16
J16-25	R[0]	E18
J16-23	R[1]	A22
J16-21	R[2]	C21
J16-18	R[3]	C20
J16-17	R[4]	A21
J16-16	R[5]	J17
J16-15	R[6]	A20
J16-14	R[7]	C19
J16-13	BLANK_N	C18
J16-37	HS	J14
J16-39	VS	H18
J16-12	SYNC_N	A19
J16-11	SYNC_T	G17
Connecteur J15	**Nom du signal**	**Nom de la broche**
J15-11	M1	B18
J15-12	M2	B19
J15-14	VIDEO_CLK	G18

Annexe F

Fichier source en langage C pour la validation du système embarqué sous µClinux

```
/*********************************
***      Test camera + Vga      ***
*********************************
*** Written by A .BEN ATITALLAH ***
*********************************/
#include
<C:\altera\kits\nios2_51\bin\eclipse\workspace\kernel_2s60_camvga\build\include\nios2_system.h
>
#include "lancelot_vga_struct.h"
#include "camera_regs.h"

int hsync_pulse_width, hsync_back_porch_width, hsync_front_porch_width;
int vsync_pulse_width, vsync_back_porch_width, vsync_front_porch_width;
int horizontal_resolution, vertical_resolution;

np_camera *camera =(np_camera *) na_camera;
np_vga *vga=(np_vga *)na_vga;

// Initialize vga controller
void vga_init()
{
  unsigned int i;
 // table des couleurs : echelle de gris
  for(i = 0; i < 256; i++)
vga->colour_table =(i << 24) + (i << 16) + (i << 8) + i;

  // Set Lancelot RGB vga mode
  vga->control = 0x08;

// Set Lancelot 640 x 480 vga parameters
 horizontal_resolution = 640;
 vertical_resolution = 480;
 hsync_pulse_width = 95;
 hsync_back_porch_width = 40;
 hsync_front_porch_width = 25;
 vsync_pulse_width = 2;
 vsync_back_porch_width = 22;
 vsync_front_porch_width = 10;

  vga->resolution = (horizontal_resolution << 16) + vertical_resolution;
  vga->hsync_timing = (hsync_pulse_width << 16) + (hsync_back_porch_width << 8) +
hsync_front_porch_width;
  vga->vsync_timing = (vsync_pulse_width << 16) + (vsync_back_porch_width << 8) +
vsync_front_porch_width;

  vga->dma= (int)0x2070000;

 // Start Lancelot vga
  vga->control = 0x04;
}

void camera_init()
{
 camera->np_control_register = 0x00000001;
 camera->np_dma_register = (int)0x2070000; //zone d'image 0x870000->0x8BB000 (640x480=0x4B000)
}

int main(void)
{
  printf("\nStarting camera.\n");
  camera_init();
  printf("\nStarting vga.\n");
  vga_init();

return 0;
}
```

Annexe G

Définition du PSNR et du SSIM

Définition du PSNR (*Peak Signal To Noise Ratio*):

La valeur du PSNR est exprimée en décibel (dB) et est donnée par l'équation suivante :

$$PSNR = 10\log_{10}\frac{S^2}{\left[\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}\left[I(m,n)-\hat{I}(m,n)\right]^2\right]}$$

On définit les paramètres suivants :

- S : Intensité maximale des pixels (255 dans notre cas).
- $I(x,y)$: Représente l'image initiale.
- $\hat{I}(x,y)$: Représente l'image traitée.

Définition du SSIM (*Structural SIMilarity*):

C'est une métrique perceptuelle qui se rapproche du système visuel humain (SVH). Il a été crée en 2004 par Z. Wang, A. C. Bovik, H. R. Sheikh et E. P. Simoncelli. Le SVH est hautement adapté pour extraire l'information structurelle dans une image. Le SSIM peut être vu comme une mesure de similarité car il est basé sur la luminance, le contraste et la mesure au travers d'une fenêtre 8x8 de la structure.

Pour deux images x et y de taille N x N, le SSIM est défini comme suit :

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Avec σ_x l'écart type de x, σ_y l'écart type de y et σ_{xy} l'écart type de xy définis comme suit :

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{1/2}$$

$$\sigma_y = \left(\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \mu_y)^2\right)^{1/2}$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i+1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

μ_x la moyenne de x et μ_y la moyenne de y données comme suit :

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N}x_i$$

$$\mu_y = \frac{1}{N}\sum_{i=1}^{N}y_i$$

Avec C_1 et C_2 sont des constantes et N la dynamique des pixels.

Annexe H

Fichier system.h

```
#ifndef __NIOS2_SYSTEM_H__
#define __NIOS2_SYSTEM_H__
/*
 * This file contains hardware information about the target platform.
 * The nios2_system.h file is being phased out and will be removed in a
 * later release.
 *
 * All base addresses for non memory devices have their high bit turned on to
 * bypass the cache.
 *
 * This file is automatically generated.  Do not modify.
 */

/* Input System: std_2s60 */
/* Target CPU: cpu */

/* Nios II Constants */
#define NIOS2_STATUS_PIE_MSK  0x1
#define NIOS2_STATUS_PIE_OFST 0
#define NIOS2_STATUS_U_MSK    0x2
#define NIOS2_STATUS_U_OFST   1

/*
 * Outputting basic values from system.ptf.
 */
#define na_ext_flash                                            0000000000
#define na_ext_flash_size                                       0x01000000
#define na_ext_flash_end                                        0x01000000
#define na_ext_ram                                              0x02000000
#define na_ext_ram_size                                         0x00100000
#define na_ext_ram_end                                          0x02100000
#define na_lan91c111                                            0x82110000
#define na_lan91c111_irq                                                 6
#define na_sys_clk_timer                                        0x82120800
#define na_sys_clk_timer_irq                                             0
#define na_jtag_uart                                            0x821208b0
#define na_jtag_uart_irq                                                 1
#define na_high_res_timer                                       0x82120820
#define na_high_res_timer_irq                                            3
#define na_reconfig_request_pio                                 0x821208a0
#define na_uart1                                                0x82120840
#define na_uart1_irq                                                     4
#define na_sysid                                                0x821208b8
#define na_sdram                                                0x01000000
#define na_sdram_size                                           0x01000000
#define na_sdram_end                                            0x02000000
#define na_pll                                                  0x821208c0
#define na_custominstruction_cpu                                0000000000
#define na_custominstruction1_cpu                               0x00000001
#define na_custominstruction2_cpu                               0x00000002
#define na_custominstruction3_cpu                               0x00000003
#define na_custominstruction4_cpu                               0x00000004
#define na_dma1                                                 0x82100060
#define na_dma1_irq                                                      7
#define na_dct2d0                                               0x82100050
#define na_idct2d0                                              0x82100080
#define na_sad0                                                 0x82100090
#define na_camera                                               0x82100000
#define na_vga                                                  0x82100020

/* Executing ...scripts/nios2_system.h/altera_avalon_lan91c111.pm */

/* Redefining lan91c111 -> enet */
#undef na_lan91c111
#undef na_lan91c111_irq

#define na_enet                                                 0x82110000
```

```
#define na_enet_irq                                                    6

#define LAN91C111_REGISTERS_OFFSET                                0x0300
#define LAN91C111_DATA_BUS_WIDTH                                      32

/* Executing ...scripts/nios2_system.h/altera_avalon_timer.pm */

#ifndef __ASSEMBLY__
#include <asm/timer_struct.h>
#endif

/* Redefining sys_clk_timer -> timer0 */
#undef na_sys_clk_timer
#undef na_sys_clk_timer_irq

#define na_timer0                               ((np_timer*) 0x82120800)
#define na_timer0_irq                                                  0

/* Executing ...scripts/nios2_system.h/altera_avalon_jtag_uart.pm */

/* No translation necessary for jtag_uart */

/* Executing ...scripts/nios2_system.h/altera_avalon_pio.pm */
#ifndef __ASSEMBLY__
#include <asm/pio_struct.h>
#endif

/* Casting base addresses to the appropriate structure */
#undef na_reconfig_request_pio
#define na_reconfig_request_pio                  ((np_pio*) 0x821208a0)

/* Executing ...scripts/nios2_system.h/altera_avalon_uart.pm */
#ifndef __ASSEMBLY__
#include <asm/uart_struct.h>
#endif

/* Redefining uart1 -> uart0 */
#undef na_uart1
#undef na_uart1_irq

#define na_uart0                                ((np_uart*) 0x82120840)
#define na_uart0_irq                                                   4

/* The default uart is always the first one found in the PTF file */
#define nasys_printf_uart na_uart0

/* Executing ...scripts/nios2_system.h/altera_avalon_sysid.pm */

/* No translation necessary for sysid */

/*
 * Basic System Information
 */
#define nasys_icache_size                                          65536
#define nasys_icache_line_size                                        32
#define nasys_dcache_size                                          65536
#define nasys_dcache_line_size                                        32

#define nasys_program_mem                                        na_sdram
#define nasys_program_mem_size                                na_sdram_size
#define nasys_program_mem_end                                  na_sdram_end
/*
 * Redefining upload location (ext_flash) to flash_kernel.
 */
#undef  na_ext_flash
#undef  na_ext_flash_size
#undef  na_ext_flash_end
#define na_flash_kernel                                       0x00000000
#define na_flash_kernel_size                                  0x01000000
#define na_flash_kernel_end                                   0x01000000

#define nasys_clock_freq                                       120000000
#define nasys_clock_freq_1000                                     120000
#define CPU_RESET_ADDRESS                                     0000000000

#endif /* __NIOS2_SYSTEM_H__ */
```

Annexe I

Pourcentage du temps CPU en fonction du traitement effectué

	EM/CM	TCD/TCDI	Q/QI	CLV	Autres	fps
Miss America						
QP=8						
FS	58,06	39,51	1,18	0,17	1,08	0,38
DS	3,41	90,53	2,66	0,40	2,99	0,87
SDS	2,28	91,72	2,65	0,42	2,93	0,86
CDS	3,15	91,38	2,68	0,41	2,37	0,87
HEX	2,97	91,52	2,68	0,41	2,43	0,87
QP=31						
FS	64,02	33,60	1,20	0,03	1,15	0,42
DS	4,37	89,80	3,16	0,09	2,59	1,11
SDS	2,94	91,21	3,20	0,09	2,56	1,11
CDS	3,87	89,47	3,13	0,09	3,45	1,10
HEX	3,72	89,67	3,17	0,08	3,35	1,11
Foreman						
QP=8						
FS	51,47	46,11	1,13	0,41	0,88	0,34
DS	3,60	91,70	2,21	0,84	1,64	0,66
SDS	2,90	92,32	2,22	0,90	1,68	0,65
CDS	3,72	91,53	2,23	0,85	1,67	0,65
HEX	2,97	91,99	2,22	0,87	1,94	0,66
QP=31						
FS	60,03	37,63	1,18	0,11	1,05	0,40
DS	5,01	89,39	2,77	0,29	2,55	0,93
SDS	3,97	90,05	2,76	0,31	2,91	0,91
CDS	5,17	89,10	2,75	0,28	2,70	0,92
HEX	3,91	90,35	2,78	0,29	2,66	0,93

Annexe J

Qualité en fonction de l'estimation de mouvement pour différentes séquences vidéo

Séquence Miss America

QP	8	10	13	16	22	31
			FS			
PSNR Y	38,16	37,16	36,07	35,21	33,94	32,87
PSNR Cb	38,53	37,79	37,32	36,93	36,28	35,79
PSNR Cr	37,82	36,78	35,65	34,63	33,22	32,51
Bitrate (kb/s)	19,03	14,18	10,66	8,65	6,8	5,8
			DS			
PSNR Y	38,06	37,04	35,94	35,06	33,75	32,62
PSNR Cb	38,5	37,83	37,41	36,97	36,43	35,87
PSNR Cr	37,75	36,7	35,55	34,63	33,2	32,32
Bitrate (kb/s)	18,96	14,13	10,56	8,33	6,47	5,37
			SDS			
PSNR Y	37,95	36,89	35,78	34,88	33,54	32,28
PSNR Cb	38,48	37,78	37,35	37,02	36,43	35,88
PSNR Cr	37,78	36,65	35,46	34,5	33,1	32,1
Bitrate (kb/s)	19,73	14,71	10,86	8,55	6,42	5,28
			CDS			
PSNR Y	38,02	36,99	35,87	34,97	33,72	32,6
PSNR Cb	38,51	37,83	37,35	37,03	36,39	35,84
PSNR Cr	37,78	36,66	35,54	34,51	33,21	32,27
Bitrate (kb/s)	19,22	14,25	10,67	8,49	6,5	5,32
			HEX			
PSNR Y	38,03	36,99	35,9	34,99	33,7	32,61
PSNR Cb	38,49	37,82	37,29	37,01	36,37	35,89
PSNR Cr	37,76	36,73	35,47	34,49	33,33	32,24
Bitrate (kb/s)	19,32	14,32	10,65	8,43	6,33	5,33

Séquence Foreman

QP	8	10	13	16	22	31
FS						
PSNR Y	33,21	32,01	30,79	29,74	28,29	26,9
PSNR Cb	37,99	37,23	36,54	35,86	35,22	34,66
PSNR Cr	39,26	38,36	37,46	36,76	35,81	35,12
Bitrate (kb/s)	76,52	56,81	41,31	31,83	22,91	17,62
DS						
PSNR Y	33,09	31,89	30,62	29,52	27,96	26,53
PSNR Cb	37,93	37,16	36,59	35,98	35,41	34,57
PSNR Cr	39,19	38,25	37,56	36,67	35,76	35,17
Bitrate (kb/s)	80,36	59,69	42,98	32,19	22,4	16,19
SDS						
PSNR Y	33,03	31,79	30,51	29,4	27,86	26,42
PSNR Cb	37,9	37,19	36,55	35,96	35,44	34,54
PSNR Cr	39,12	38,29	37,41	36,63	35,82	35,19
Bitrate (kb/s)	84,91	63,44	44,95	33,64	23,02	16,47
CDS						
PSNR Y	33,1	31,89	30,6	29,51	28	26,5
PSNR Cb	37,92	37,16	36,61	36	35,47	34,52
PSNR Cr	39,12	38,24	37,55	36,71	35,77	35,01
Bitrate (kb/s)	80,9	60,31	43,09	32,57	22,59	16,24
HEX						
PSNR Y	33,01	31,79	30,5	29,41	27,9	26,48
PSNR Cb	37,91	37,12	36,57	35,93	35,46	34,61
PSNR Cr	39,14	38,24	37,5	36,7	35,76	35,14
Bitrate (kb/s)	83,48	62,06	44,61	33,44	23,08	16,6

Séquence Claire

QP	8	10	13	16	22	31
FS						
PSNR Y	37,5	36,22	34,84	33,73	32,34	30,95
PSNR Cb	37,9	36,89	35,91	35,17	33,95	32,27
PSNR Cr	40,32	39,43	38,69	37,7	37,47	35,86
Bitrate (kb/s)	20,92	15,93	11,8	9,34	7,04	5,61
DS						
PSNR Y	37,34	36,04	34,62	33,44	32,1	30,42
PSNR Cb	37,91	36,98	35,89	35,19	33,99	32,18
PSNR Cr	40,42	39,9	38,7	37,81	37,36	35,73
Bitrate (kb/s)	20,9	15,81	11,57	8,85	6,61	5,13
SDS						
PSNR Y	37,32	36,05	34,55	33,33	32	30,34
PSNR Cb	37,88	36,93	35,8	35,14	34,01	31,95
PSNR Cr	40,31	39,47	38,58	37,76	37,42	35,7
Bitrate (kb/s)	21,09	15,84	11,63	8,97	6,55	5,08
CDS						
PSNR Y	37,31	36,05	34,55	33,39	31,97	30,37
PSNR Cb	37,92	36,89	35,79	35,2	33,94	32,15
PSNR Cr	40,33	39,41	38,48	37,69	37,41	35,72
Bitrate (kb/s)	20,97	15,74	11,58	8,93	6,6	5,09
HEX						
PSNR Y	37,35	36,06	34,62	33,42	32,03	30,43
PSNR Cb	38,05	37	35,97	35,21	33,98	32,16
PSNR Cr	40,39	39,41	38,85	37,82	37,41	35,84
Bitrate (kb/s)	21,13	15,95	11,63	8,93	6,62	5,12

Séquence Carphone

QP	8	10	13	16	22	31
FS						
PSNR Y	34,64	33,39	32,01	30,92	29,44	27,9
PSNR Cb	39,62	38,82	38,04	37,27	36,56	35,29
PSNR Cr	39,4	38,32	37,48	36,73	35,77	35,06
Bitrate (kb/s)	50,16	36,96	26,75	20,52	14,53	11,3
DS						
PSNR Y	34,55	33,23	31,89	30,76	29,23	27,65
PSNR Cb	39,61	38,81	37,95	37,27	36,51	35,32
PSNR Cr	39,45	38,36	37,49	36,8	35,91	35,11
Bitrate (kb/s)	51,42	37,82	27,08	20,22	14,11	10,44
SDS						
PSNR Y	34,54	33,2	31,83	30,67	29,12	27,51
PSNR Cb	39,56	38,88	37,97	37,29	36,65	35,4
PSNR Cr	39,41	38,42	37,53	36,83	35,89	35,2
Bitrate (kb/s)	52,68	38,84	27,48	20,61	14,04	10,17
CDS						
PSNR Y	34,53	33,24	31,84	30,72	29,19	27,61
PSNR Cb	39,59	38,89	37,92	37,26	36,53	35,3
PSNR Cr	39,37	38,43	37,6	36,99	35,85	35,07
Bitrate (kb/s)	51,96	37,96	27,26	20,36	14,05	10,18
HEX						
PSNR Y	34,51	33,21	31,85	30,73	29,15	27,6
PSNR Cb	39,52	38,94	37,89	37,24	36,62	35,24
PSNR Cr	39,47	38,43	37,43	36,94	35,77	35,1
Bitrate (kb/s)	53,03	38,89	27,73	20,54	14,29	10,33

Annexe K

Coefficients de Loeffler

La table ci-dessous représente les coefficients utilisés par l'algorithme de Loeffler pour la transformée TCD/TCDI :

Constantes	TCD	TCDI
C1	-0.3901805878	0.3901805878
C2	-1.1111403704	1.1111403704
C3	1.6629391909	1.6629391909
C4	1.9615705013	1.9615705013
C5	0.5411961079	0.5411961079
C6	0.7653667927	-0.7653667927
C7	1.8477590084	1.8477590084
C8	1.1758755445	1.1758755445
C9	1.3870397806	1.3870397806
C10	0.7071067691	0.7071067691

Coefficients de la D.A

La forme matricielle de l'équation 11 du chapitre 4 est donnée comme suit :

$$\begin{bmatrix} y_0 \\ y_2 \\ y_4 \\ y_6 \end{bmatrix} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_3 \\ y_5 \\ y_7 \end{bmatrix} = \begin{bmatrix} d & e & f & g \\ e & -g & -d & -f \\ f & -d & g & e \\ g & -f & e & -d \end{bmatrix} \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix}$$

Avec a, b, c, d ,e, f et g représentent les constantes utilisées par l'algorithme DA :

$$a = \frac{1}{2}\cos(\frac{\pi}{4}); b = \frac{1}{2}\cos(\frac{\pi}{8}); c = \frac{1}{2}\sin(\frac{\pi}{8})$$

$$d = \frac{1}{2}\cos(\frac{\pi}{16}); e = \frac{1}{2}\cos(\frac{3\pi}{16}); f = \frac{1}{2}\sin(\frac{3\pi}{16}); g = \frac{1}{2}\sin(\frac{\pi}{16})$$

GLOSSAIRE

ALM : *Adaptive Logic Module*
BMA : *Block Matching Algorithm*
CDS : *Cross-Diamond Search*
CISC : *Complex Instruction Set Computer*
CLV : *Codage à Longueur Variable*
CM : *Compensation de Mouvement*
CPLD : *Complex Logic Programmable Device*
CSP : *Cross Search pattern*
DA : *Distribution Arritjmétique*
DMA : *Direct Memory Access*
DMIPS : *Dhrystons Million Instructions per second*
DS : *Diamond Search*
DSP : *Digital Signal Processor*
EM : *Estimation de Mouvement*
FDV : *Frame Data Valid*
FFT : *Fast Fourier Transform*
FPGA : *Field Programmable Gate Array*
fps : *frame per second*
FSBM : *Full Search Block Matching*
GOB : *Groupe de Blocs*
GPL : *General Public Licence*
HDL : *Hardware Design Language*
HEX : *Hexagonal Search*
HW : *Hardware*
IDE : *Integrated Design Entry*
IP : *Intellectual Property*
JPEG : *Joint Photographic Experts Group*
LAB : *Logic Array Bloc*
LDSP : *Large Diamond Search Pattern*
LDV : *Line Data Valid*
LHSP : *Large Hexagonal Search Pattern*
LUT : *Look Up Table*
MBs : *Macroblocs*
MIPS : *Million Instructions per Second*

MMU : *Memory Management Unit*
MPEG : *Moving Picture Experts Group*
PDA : *Personal Digital Assistants*
POSIX : *Portable Operating System Interface X*
PSNR : *Peak Signal To Noise Ratio*
QP : *Pas de Quantification*
Q : *Quantification*
QI : *Quantification Inverse*
RAC : *ROM-Accumulateur*
RISC : *Reduced Instruction Set Computer*
RTL : *Register Transfer Logic*
SAD : *Sum of Absolute Difference*
SDS : *Small Diamond Search*
SDSP : *Small Diamond Search Pattern*
SHSP : *Small Hexagonal Search Pattern*
SoC : *System on Chip*
SoPC : *System On Programmable Chip*
SSIM : *Structural SIMilarity*
SVH : *système visuel humain*
SW : *Software*
TCD : *Transformée en Cosinus Discrète*
TCDI : *Transformée en Cosinus Discrète*
UAL : *Unité Arithmétique et Logique*
UIT-T : *Union Internationale des Télécommunications - Télécommunications*
VSIA : *Virtual Socket Interface Alliance*
VHDL : *Very high speed integrated circuit Hardware Description Language*

Etude et Implantation d'Algorithmes de Compression d'Images dans un Environnement Mixte Matériel et Logiciel

Le sujet de cette thèse est la contribution au développement et à la conception d'un système multimédia embarqué en utilisant la méthodologie de conception conjointe logicielle/matérielle (*codesign*). Il en a découlé la constitution d'une bibliothèque des modules IP (*Intellectual Property*) pour les applications vidéo. Dans ce contexte, une plateforme matérielle d'acquisition et de restitution vidéo a été réalisée servant de préalable à l'évaluation de la méthodologie de conception en codesign et à toute étude d'algorithme de traitement vidéo. On s'est intéressé en particulier à l'étude et à l'implantation de la norme de compression vidéo H.263 de l'organisme UIT-T. La fréquence de fonctionnement de la plateforme est de 120 MHz. L'ensemble du développement est exécuté par le processeur NIOS II sous le système d'exploitation µClinux. Le codeur H.263 ainsi développé, grâce aux différents accélérateurs matériels pour le SAD, TCD/TCDI et Q/QI permet de coder des séquences vidéo QCIF@15Hz.

Mots Clés :
Système Multimédia Embarqué, Compression d'Images, H.263, FPGA, Processeur NIOS II, HW/SW Codesign.

Study and Implementation of Algorithms for Image Compression in a Hardware and Software Environment

The main purpose of this thesis was to contribute to the development and to the design of an embedded system for multimedia by using the HW/SW methodology (*codesign).* A library of flexible IP cores (*Intellectual Property)* for video applications was created. Within this framework, a hardware platform for video acquisition and video restitution was achieved in order to evaluate the codesign methodology approach and to study the video processing algorithm. We have studied and implemented the H.263 video encoder from the UIT-T organism. The frequency of our platform is about 120 MHz. The whole development was executed under µClinux Operating System and controlled by the NIOS II processor. The H.263 encoder was developed with different hardware accelerators for the SAD, TCD/TCDI and Q/QI operations and permits finally to code video sequences at QCIF@15Hz.

Key words:
Embedded System for Multimedia, Image Compression, H.263, FPGA, NIOS II Processor, HW/SW Codesign.