Bi-codeur entropique JPEG-2000 pour applications mobiles

I.Aouadi O.Hammami V.Martin
ENSTA
32 Bd Victor 75739 Paris
{aouadi, hammami, vmartin}@ensta.fr

Résume

Le nouveau standard de compression d'image JPEG-2000 est venu combler les lacunes de son prédécesseur JPEG, Mais JPEG-2000 a impose de nouvelles complexités algorithmique. Le codeur entropique représente la partie la plus intensive du traitement de JPEG-2000. Dans le cadre de ce papier et dans le but d'améliorer les performances de traitement de JPEG-2000, une implémentation matérielle sur FPGA du codeur entropique est proposée. Ce papier est développé dans cinq parties : d'abord une introduction ensuite une présentation du standard et celle du codeur entropique, puis l'implémentation qui a été réalisée sur FPGA, finalement on donne les résultats obtenus et la conclusion de ce travail.

Mots clefs

JPEG-2000, codeur entropique, implémentation, FPGA.

1. Introduction

Le standard JPEG-2000 [1] de compression d'images fournit de nombreuses fonctionnalités grâce entre autres a son étape de codage entropique [2]. Le standard décrit le fonctionnement du traitement mais n'impose aucune implémentation particulière. Les implémentations du standard ou des parties du standard font donc l'objet d'études variées pour obtenir des solutions ayant un bon rapport coût-performances. Des implémentations à base de VLIW ou FPGA ont entre autres été publiées [6,7, 12]. Néanmoins la recherche d'une solution présentant un compromis entre la consommation d'énergie et la performance souhaitable pour les applications mobiles n'a pas a notre connaissance été présenté. Nous décrivons une solution matérielle focalisée sur la partie la plus intensive du traitement JPEG-2000 c'est-à-dire le codeur entropique et nous la comparons a une implémentation logicielle sur processeur embarque pour applications mobiles.

2. JPEG-2000

2.1 Chaîne de traitement

La chaîne de codage de JPEG-2000 commence par un prétraitement de l'image originale. Ce prétraitement permet de changer la représentation de l'image (facultatif), tel qu'une transformation de couleur RGB→YUV. Ensuite, on découpe l'image en « Tile ». Chaque « Tile » subit une transformée en ondelettes. Une quantification est par la suite appliquée sur chaque sous bande obtenue suite à la transformée en ondelettes (TO). Les coefficients obtenus sont traités par le codeur entropique qui permet d'obtenir les données compressées. Ces données sont finalement mises en forme pour aboutir à un fichier JPEG-2000 (figure 1).

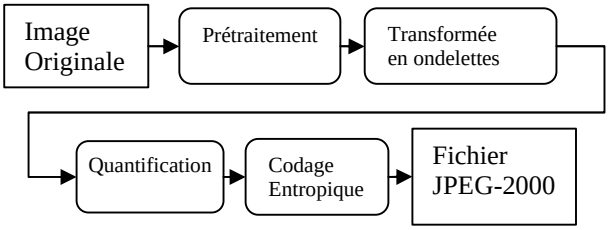

Figure 1 : Chaîne de codage JPEG-2000

Le standard JPEG2000 fournit de nouvelles fonctionnalités qui le mettent à jour avec les nouveaux circuits disponibles en ce moment. Parmi ses nouvelles fonctionnalités on peut citer : (1) Obtenir des performances supérieures à celles de son prédécesseur JPEG, notamment pour des débits très faibles. (2) Permettre d'organiser les fichiers compressés de plusieurs manières, notamment en fonction de la résolution désirée ou de la qualité de reconstruction. (3) Avoir un mode de compression sans pertes performant. (4) Fournir la possibilité de coder des parties d'une image avec une qualité supérieure à d'autres parties.

2.2. Le codeur entropique

Cette partie de JPEG-2000 représente 70% à 80% du temps total d'exécution. Le principe de base du codage entropique est le EBCOT [2]. L'idée principale consiste à : (1) Décomposer chaque sous bande en des petits blocks (64x64 max), (2) Avoir un embedded bit-stream pour chaque code-block, (3) Le codestream à la sortie est composé d'un ensemble de couches (layers). Le codeur entropique est composé de deux blocs de traitement : le bloc de formation du contexte et le codeur arithmétique (figure 2).

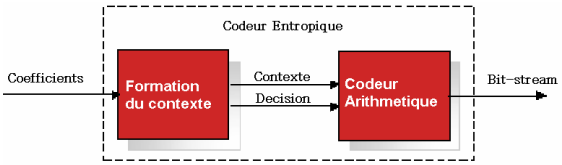

Figure 2 : schéma du codeur entropique

2.2.1. Formation du contexte

La formation du contexte est un processus au niveau des bit-planes qui balaye chaque bit-plane d'un code-block et génère un contexte et une décision. Pour chaque bit-plane (sauf le premier) on effectue trois passes de codage qui sont :

(1) *significance pass*,
(2) *Refinment pass*,
(3) *clean-up pass*.

Pour le premier bit-plane (MSB non nul) on applique seulement le *clean-up pass*.

Dans la figure suivante on représente la décomposition d'un code block en plusieurs bits plans. Chaque bit-plane est balayé à la manière montrée dans les figures 3 et 4.

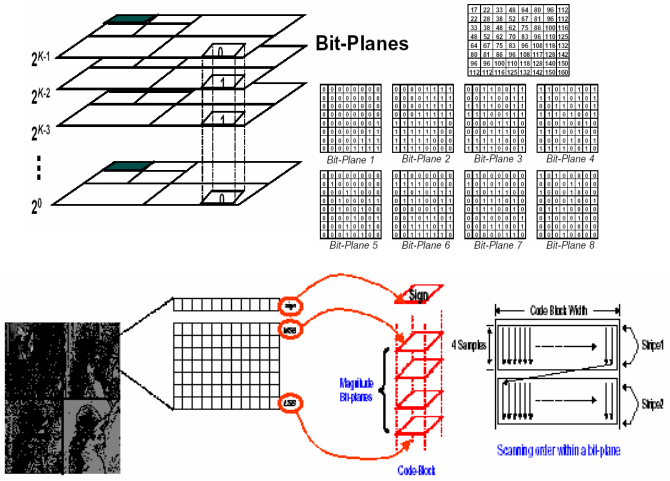

Figure 3 : Décomposition en bit plans

A chaque position du bit-plane une fenêtre de 3x3 voisins est utilisée pour la détermination du contexte.

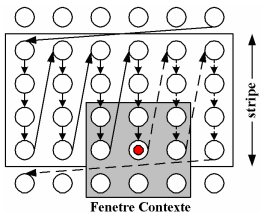

Figure 4 : Fenêtre Contexte

2.2.2. Le codeur arithmétique

Le codeur arithmétique compresse les coefficients de la TO en un codestream en utilisant les contextes générés par le bloc de formation de contexte. Le codeur arithmétique a à son entrée un contexte et un symbole (décision) et un Bitstream en sortie. L'algorithme du codeur entropique est détaillé dans les figures de l'annexe.

3. Architecture et Implémentation

La réduction du temps d'exécution du codeur entropique passe principalement par un macroparallelisme du traitement des code blocs. Pour cela nous proposons une solution matérielle implémentant deux codeurs entropiques en parallèle.

3.1. Architecture

L'architecture du bi-codeur est formée de deux codeurs identiques possédant chacun des ressources indépendantes. Chaque codeur utilise un banc de RAM pour charger les bits plans et un banc de mémoire pour stocker le bit-stream. Chaque banc de mémoire a besoin d'une interface qui assure la liaison avec le codeur. Chaque codeur est forme en plus des interfaces : (1)d'un module pour la formation du contexte, (2) d'une FIFO pour le stockage des valeur intermédiaire,(3)d'un module pour le codage arithmétique.

Le module de formation du contexte utilise un ensemble de bloc de mémoire sur le FPGA qui lui permettent de stocker les contextes et les valeurs intermédiaires du traitement. Certains de ces mémoires utilisent un accès à double port pour améliorer la récupération des données.

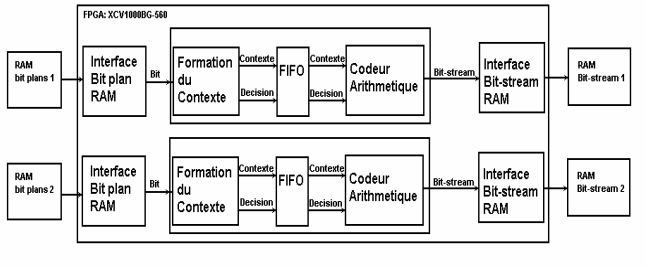

Figure 5 : architecture du bi-codeur

La majeure spécificité de cette implémentation est la combinaison entre une implémentation matérielle et logicielle de la même chaîne de codage contrairement a la majorité des implémentations qui sont soit totalement logicielles soit totalement matérielles [8], [9].

3.2. Implémentation

D'abord pour implémenter la chaîne JPEG 2000 différentes analyses ont été menés tel que [5], [6], puis des méthodes d'optimisation [4] et plusieurs architectures [5], [8], [9] ont été proposés. Dans le cadre de cette étude l'implémentation du codeur entropique a été effectué sur un circuit FPGA implanté sur une carte Celoxica dont l'architecture est construite autour d'un Virtex XCV1000BG-560 de chez Xilinx. L'architecture de la carte est détaillée dans la figure suivante :

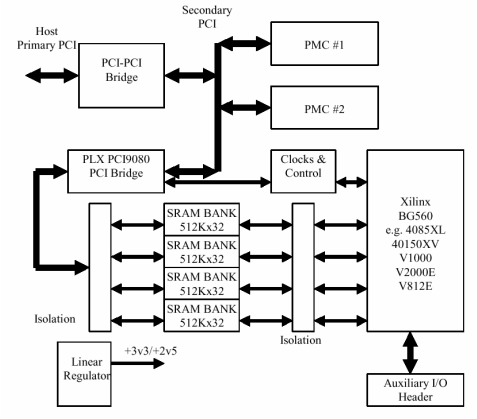

Figure 6 : architecture de la carte Celoxica

Le test et la validation des deux codeurs entropiques ont été réalisés par intégration dans une chaîne de codage JPEG-2000 implémentée en C++.
Le fonctionnement de la chaîne de codage de JPEG-2000 complète est décrit dans la figure 7. La connexion entre la partie logicielle et la partie matérielle de cette chaîne est assurée par le bus PCI, et les bancs de RAM de la carte Celoxica.

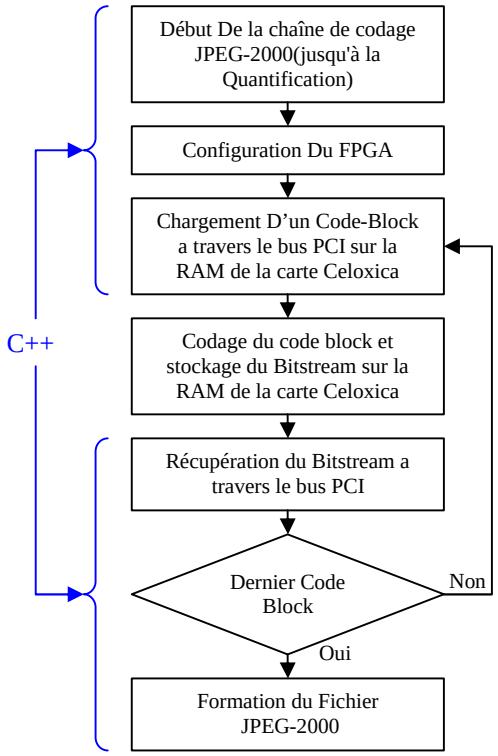

Figure 7 : Fonctionnement de la chaîne de codage JPEG-2000

4. Résultats

Les résultats sur les ressources utilisées sur le composant par les deux codeurs après synthèse et placement routage donnent les résultats suivants :

Table 1 : ressources utilisées :

Utilisation de la Logique :	
Nombre Total des Slice Registers	1,946 out of 24,576 7%
La Distribution de la logique:	
Nombre des Slices occupés	4,630 out of 12,288 37%
Nombre des Block RAMs	32 out of 32 100%

On se rend bien compte que les codeurs n'utilisent que 37% des ressources logiques du composant mais par contre l'ensemble des ressources mémoires, ce qui limite a 2 les codeurs dus aux ressources en Block RAM du composant. L'implémentation des deux codeurs permet une fréquence de fonctionnement maximale de 30 MHz. Les résultats ont été comparés avec une machine ayant un processeur Pentium 4M [10,11] à 1.6 Ghz avec une mémoire de 1 Go. L'OS de la machine est Windows XP professionnel. Les résultats obtenus représentant le Temps d'Exécution du Code C++(TECC++) et le Temps d'Exécution du Bi-codeur Matériel (TEBM) sont présentés dans le tableau suivant :

Table 2 : Les performances

image	Format	dimension	TECC++ (ms)	TEBM (ms)
Images Couleur				
Lena	ppm	512x512	644,08	1178,54
FreeBSD	ppm	512x512	455,51	1103,75
Linda	ppm	512x512	514,12	1092,13
Images Noir &blanc				
Lena	pgm	512x512	210,2	413,27
Arial2	pgm	2048x2048	3981,17	6 698,75

Les TEBM mesurés ne prennent pas en compte le temps de chargement des codes blocks sur les bancs de RAM de la carte Celoxica. Les paramètres de compression sont : (1) une taille de code block de 32x32, (2) une décomposition en ondelette 5/3(Lossless) de 5 niveaux, (3) une transformation de couleur RGB→YUV (pour les images couleur seulement).
Les paramètres de surface et de consommation d'énergie sont les suivants :

Table 3 : Consommation

	P4 M	XCV1000
consommation	25 W	542mw

Nous n'avons pas pu faire de comparaison avec les implémentations commerciales vue qu'elle ne fournissent pas, ni les mesure de performances des codeurs, ni le nombre de codeurs implémentés.

5. Conclusion

Les résultats montrent que le processeur Pentium 4 M permet un speed up de 2 en moyenne pour une fréquence 53 fois plus grande que celle du bi-codeur entropique. Par contre la consommation est de 0.0268 celle du processeur mobile ce qui rend notre solution plus adaptée

aux applications multimédia mobiles. Cette solution peut alors être associée a un processeur simple pouvant être très facilement intégré sur les 60% restant du composant.

Annexe :

Les procédures du codeur arithmétique (MQ-Coder) :

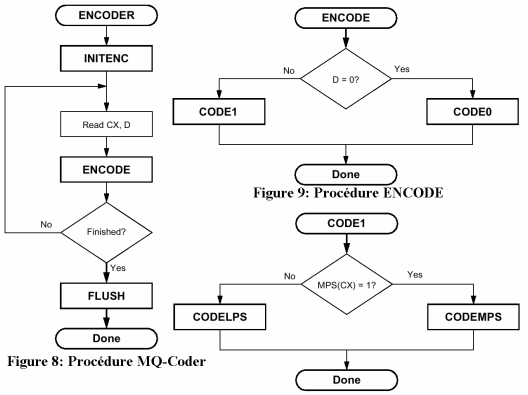

Figure 8: Procédure MQ-Coder

Figure 9: Procédure ENCODE

Figure 10: Procédure CODE1

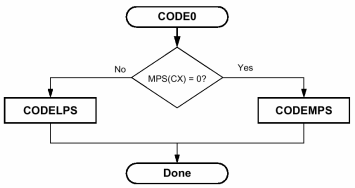

Figure 11: Procédure CODE0

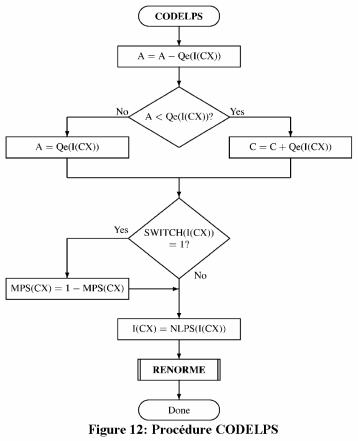

Figure 12: Procédure CODELPS

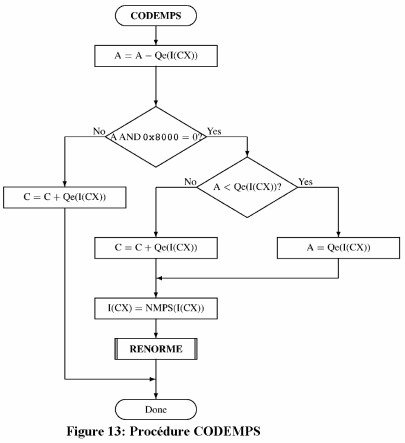

Figure 13: Procédure CODEMPS

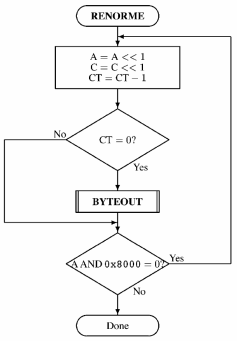

Figure 14: Procédure RENORME

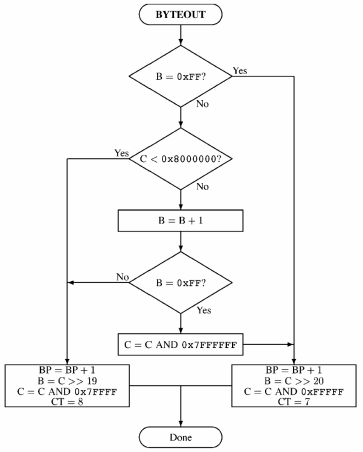

Figure 15: Procédure BYTEOUT

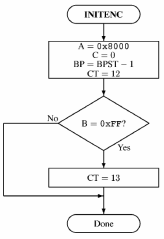

Figure 16: Procédure INITENC

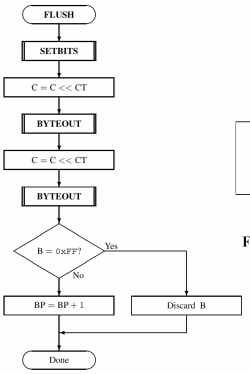

Figure 17: Procédure FLUSH

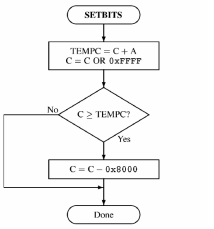

Figure 18: Procédure SETBITS

Références

[1]JPEG 2000 Part I with Cor.1 Cor.2, Cor.3 and DCor.4, Amd.1, FPDAM.2, ISO/IEC JTC1/SC29/WG1 N2513R2, May 2002

[2]D. Taubman. High performance scalable image compression with EBCOT. IEEE Transactions on Image Processing, 9(7):1158--70, July 2000.

[3]Michael Dyer, David Taubman, Saeid Nooshabadi, Memory Efficient Pass-parallel architecture for JPEG2000 Encoding, ISSPA, July 2003, Paris, France.

[4]Yijun Li, Ramy E.Aly,and Magdy A.Bayoumi Samia A. Mashali, Parallel High-Speed Architecture for EBCOT in JPEG2000. ICIP, Sept.2003, Barcelona, Spain.

[5]Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, Liang-Gee Chen Analysis and Architecture Design of Block Coding Engine for EBCOT in JPEG 2000, IEEE Trans. on circuits and systems for viedo technology, vol.13, no.3, march 2003.

[6]Imed Aouadi, Omar Hammami, Microarchitectural Characterization of JPEG-2000 Software, Proceedings of the 9th International Workshop on Systems, Signals and Image Processing – Recent Trends in Multimedia Information processing, Manchester, World Scientific, ISBN 981-238-243-7, pp. 267-277, 7-8 Nov.2002.

[7]Imed Aouadi, Omar Hammami, A SOPC Oriented FPGA Implementation of the JPEG-2000 Entropy Coder, IEEE Picture Coding Symposium PCS03, Saint Malo France, April 23-25, 2003.

[8]Barco silex JPEG-2000 Encoder BA112JPEG2000E Factsheet

[9]Alma technologies, JPEG 2000 Encoder JPEG2K_E.

[10] Intel Pentium M Processor Datasheet June 2003 Order number: 252612-002

[11] Intel Pentium M Processor for Embedded Applications – Thermal Design Guide April 2003 Order Number: 273885-001.

[12]O.Hammami, E.Zheng and I.Aouadi: Performance Evaluation of JPEG-2000 on VLIW Architectures, Proc. of the 14th IEEE International Conference on Microelectronics, pp.202-205, ISBN 0-7803-7573-4, Beirut, Lebanon, Dec.11-13, 2002.