Réalisation d'une IP de scrutation et de codage entropique pour un codeur MPEG4

Khalil Hachicha[1], Patrick Garda[1]

[1]Universite Pierre et Marie Curie LISIF - SYEL - B.C. 252 4, place Jussieu - 75252 PARIS CEDEX 05

khalil.hachicha@lis.jussieu.fr

Résumé – Dans cet article, nous décrivons la conception et la réalisation d'une IP de scrutation et de codage entropique. Notre approche basée sur un traitement en parallèle a permis une réduction significative du chemin critique. La conception est entièrement synchrone et contient 4000 portes logiques. Le format de sortie est compatible avec celui du codage à longueur variable pour un codeur vidéo MPEG4 standard. L'architecture synthétisée par Précision RTL, est très performante pour la compression vidéo en temps réel.

Abstract *– In this paper, we describe the design and the implementation of the scan and the entropic coding IP. The realized architecture, synthesized by Precision RTL, offers high performances for real time video compression. A novel approach allowing the reduction of the critical path and based on parallel coding is presented. The design is fully synchronous and contains 5000 logic gates. The format of the output is compatible with the variable length coding for MPEG4 video coder.*

1. Introduction

MPEG4 part 9[1] est un projet international regroupant plusieurs laboratoires de recherche qui conçoivent des IP pour les différents blocs d'un codeur MPEG4 dans le but d'exécuter le codage de séquence grand format en temps réel. Cette bibliothèque servira de base à la normalisation pour l'ISO d'un codeur matériel pour MPEG4. La plateforme choisie est une carte WILDCARDII [2] contenant un FPGA VIRTEX de Xilinx et les protocoles de communication ont été développés et testés. Notre participation à ce projet porte sur la conception et la réalisation d'une IP de scrutation suivie du codage entropique.

La scrutation consiste à changer la disposition des différents coefficients d'un bloc de telle façon que ces derniers deviennent disposés dans un ordre allant des coefficients reflétant les basses fréquences jusqu'aux coefficients reflétant les hautes fréquences. Cette disposition plus compacte des données permet d'avoir une plus grande probabilité de valeurs non nulles suivies de valeurs plus ou moins nulles correspondant aux hautes fréquences. Le codage entropique est ensuite réalisé en deux étapes : une première consiste à mettre les données scannées sous forme de couples « run » et « level ». Le « run » correspond aux nombres de 0 qui se suivent et le « level » correspond aux nombres différents de zéro venant juste après cette succession de zéros. Il est clair que plus il y a de zéros qui se succèdent, moins on aura de couples à coder et la compression sera plus efficace. La deuxième étape consiste à retrouver pour chaque couple (run, level), à partir des différents tableaux de codage mis à disposition, un code correspondant qui va être écrit dans le bitstream. Pour cela, nous retrouvons plusieurs tableaux dont chacun est utilisé pour un cas bien spécifique. Ainsi, nous retrouvons des tableaux utilisés en cas de codage : codage Inter, codage Inter fin bloc, codage Intra et codage Intra fin bloc. La différenciation entre ces cas s'appuie sur le fait que les différentes probabilités de succession des données change considérablement dans ces 4 cas. Prenons le cas où nous sommes à la fin du bloc. Il y a plus de probabilité à retrouver beaucoup de zéros qui se succèdent. Ainsi attribuer un code plus petit à un couple (run, level) ayant un grand run est plus astucieux.

Les travaux précédents dans la conception de circuits pour le codage entropique se sont concentrés exclusivement sur la réalisation d'un débit binaire élevé à la sortie. Lei et al [3] ont proposé une architecture employant des opérations parallèles pour transformer les données d'entrée en codes, les enchaîner ensemble et les segmenter en mots binaires de 16 bits. Stephen Arthur [4] a adopté une approche de basse consommation. Il a prouvé que la recherche de codes dans les différentes tables est la source dominante de consommation. En conséquence, il a présenté une nouvelle approche de partition des tables basée sur la probabilité d'occurrence des symboles. L'inconvénient majeur de ces architectures est imputé au fait qu'elles gèrent uniquement un seul mode de codage qui se fait directement à partir de tableaux VLC. Ceci n'est pas suffisant pour assurer l'attribution de code pour chaque couple (run, level) possible. Nous nous sommes proposés ainsi, de concevoir une architecture d'un codeur complet tenant compte des trois modes de codage tels qu'ils sont proposés par la norme MPEG4[5] et permettant de gérer tous les cas possibles.

Dans cet article, une architecture d'encodeur parallèle complet sera présentée ; nous détaillerons les unités de scrutation, de prétraitement, de contrôle, de codage et finalement l'unité de paquetage. Nous donnerons le diagramme de fonctionnement de l'unité principale de codage et les différentes optimisations pour accélérer le traitement dans cette unité. Enfin nous présenterons les résultats et analyserons l'architecture proposée.

2. Description de l'IP

Pour concevoir l'IP du module de scrutation et de codage entropique, la première étape de notre travail a été de décomposer notre IP en différentes unités afin de faciliter le travail de test. La décomposition s'est faite d'une manière judicieuse en tenant compte de la façon dont l'algorithme est organisé (figure 1)

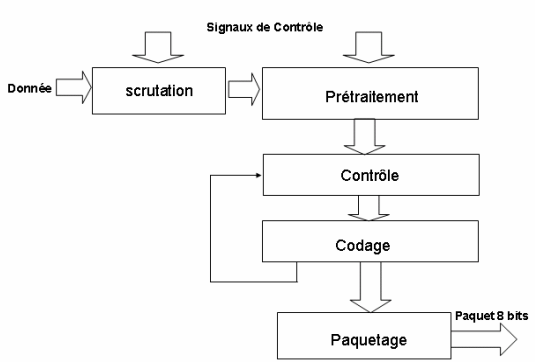

FIG. 1 Schéma des blocs

1.1 Unité de scrutation

Cette unité peut effectuer la scrutation de trois manières différentes : la scrutation verticale, la scrutation horizontale et la scrutation normale (Figure 2).

Le type à choisir dépend de l'utilisation ou non de la prédiction des coefficients Ac. Trois cas se présentent :

- pas de prédiction de coefficients Ac : mode de scrutation normal,
- prédiction des coefficients Ac à partir de la ligne 1 : mode de scrutation horizontal,
- prédiction des coefficients Ac à partir de la colonne 1 : mode de scrutation vertical.

Dans les 64 premiers coups d'horloge, nous chargeons le premier bloc dans une zone mémoire. Le chargement se fait en série (échantillon par échantillon). Aucune donnée n'est envoyée vers le deuxième bloc durant ce temps là. Le type de scrutation est décidé par l'intermédiaire de 2 signaux de contrôle qui reflètent le mode de prédiction choisi pour les coefficients Ac. Une fois ces 64 cycles écoulés, l'envoi des données scrutées, correspondant au premier bloc commence, en même temps que le deuxième bloc est chargé. Ce travail se poursuit jusqu'au chargement complet des 6 blocs.

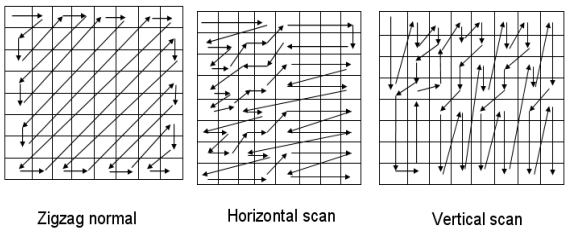

FIG. 2 : Les différents types de scrutation

1.2 Unité de prétraitement

Le bloc de prétraitement fournit à celui du codage tous les signaux nécessaires dans le but de choisir les tableaux de codage qui doivent être utilisés. Dans cette unité, la première étape consiste à déterminer le type de données que l'on vient de recevoir à savoir :

- est il Ac ou Dc ?
- correspond t il à la luminance ou à la chrominance ?
- est il le dernier échantillon dans le bloc ?

La connaissance de ces détails est possible puisque nous connaissons l'ordre d'arrivée des données. En effet, les quatre premiers blocs correspondent aux données de luminance, les deux derniers correspondent à ceux de chrominance. Dans chaque bloc, le premier échantillon correspond au coefficient Dc et ceux qui suivent à des coefficients Ac. Le 63ème échantillon correspond aux derniers échantillons dans le bloc. Nous avons ainsi créé des compteurs d'échantillons et de blocs qui permettent de décider le type d'échantillon que nous avons à traiter. Ces compteurs sont créés dans des processus à part. Un autre processus calcule, pour chaque échantillon, sa valeur absolue et un troisième processus s'intéresse aux calculs du run comptant le nombre de zéros qui se succèdent. Le dernier processus contient une machine à état interne qui change d'état suivant le type de données à traiter. Si le coefficient est un Dc, nous envoyons seulement le « level » de l'échantillon. Si le coefficient est un Ac, nous envoyons le run calculé, le complément A2 du « level » et le bit de signe du « level».

1.3 Unité de contrôle

Le bloc de contrôle est nécessaire pour deux raisons. La première est que le bloc de traitement envoie les données correspondant aux couples (run, level) d'une manière non continue. La deuxième est que le nombre de cycles nécessaires pour trouver le code correspondant aux couples envoyés varie selon le mode de codage : normal, escape mode 1, escape mode 2 ou escape mode 3. Ainsi, l'unité de contrôle reçoit des signaux du bloc de codage pour savoir si ce dernier peut recevoir de nouvelles données et permet la mémorisation de toutes les données que le bloc de codage ne peut pas traiter à un instant précis.

1.4 Unité de codage

Le troisième bloc, est celui du codage, dans lequel nous essayons de trouver les différents codes.

- Pour les coefficients Dc, nous écrivons dans la mémoire le code correspondant au nombre de bits sur lesquels ils sont définis, suivi de la valeur du coefficient DC,
- pour les coefficients Ac, plusieurs paramètres entrent en jeux dans le codage : est ce le dernier couple ? est ce que le codage est en mode Inter ou Intra ? Ces

informations sont récupérées par l'intermédiaire de différents signaux issus du bloc de prétraitement.

Quatre tableaux VLC pour quatre modes de codage : le mode « Inter », « Inter_last », « Intra » et « Intra_last » sont prévue pour ce fait. Nous avons stocké ces tableaux dans différents emplacements mémoires et pour chaque couple de données, nous essayons de voir si le couple possède un code correspondant dans la mémoire ou non.
Nous avons rencontré ici un problème dû au fait que les codes correspondant aux différents couples sont de longueur variable. C'est pour cette raison que pour chaque code, un mot binaire de 4 bits, indiquant la taille du code que nous devons récupérer, est ajouté au code lui même. Suite à la récupération du code, nous écrivons le résultat dans un tampon prévu à cet effet pour stocker le flux. Chaque fois qu'un code est écrit, un compteur s'incrémente du nombre de bits écrits.
Il est à noter que la recherche d'un code correspondant à un couple de données peut échouer. Ceci engendre de nouveaux calculs et traitements et par conséquent des retards qu'il faut absolument prendre en compte lors de la conception. Nous avons essayé de développer une architecture permettant d'avoir les meilleures performances. Le séquencement des traitements et l'optimisation de l'architecture sont détaillés dans le paragraphe suivant.

1.4.1 Explication détaillée

Cycle 1 :

- traiter le coefficient Dc,
- chercher le code correspondant au nombre de bits sur lesquels le coefficient Dc est écrit.

Cycle 2 :

Pour le coefficient Dc, nous écrivons le code trouvé :
- écrire le code de la dimension du coefficient Dc,
- écrire la valeur du coefficient,
- écrire le signe du coefficient.

Pour les coefficients Ac, nous :
- lisons le code correspondant au 1er couple (run, level),
- mémorisons le run et level dans deux tampons.

Cycle 3 :

Pour le coefficient Ac, deux situations sont possibles :

1er cas :

Si la longueur du code lu pendant le cycle 2 est nulle, cela signifie que le code correspondant au couple précédent n'a pas été trouvé. Dans ce cas, nous passons au mode « escape 1 ». Un signal « busy » est mis à 1 et envoyé à l'unité de prétraitement pour qu'elle arrête l'envoi de nouveaux couples.

Nous recalculons alors le couple (run, level) en modifiant la valeur du level. La nouvelle valeur du level est calculée à l'aide de sa valeur mémorisée pendant le cycle 1 et de la valeur « escape_level » déterminée en fonction de la valeur du run. La détermination de cette valeur se fait à partir de tableaux qui dépendent de la nature du codage (Inter ou Intra) et de la position du couple (run, level) dans le bloc à coder (dernier ou non).

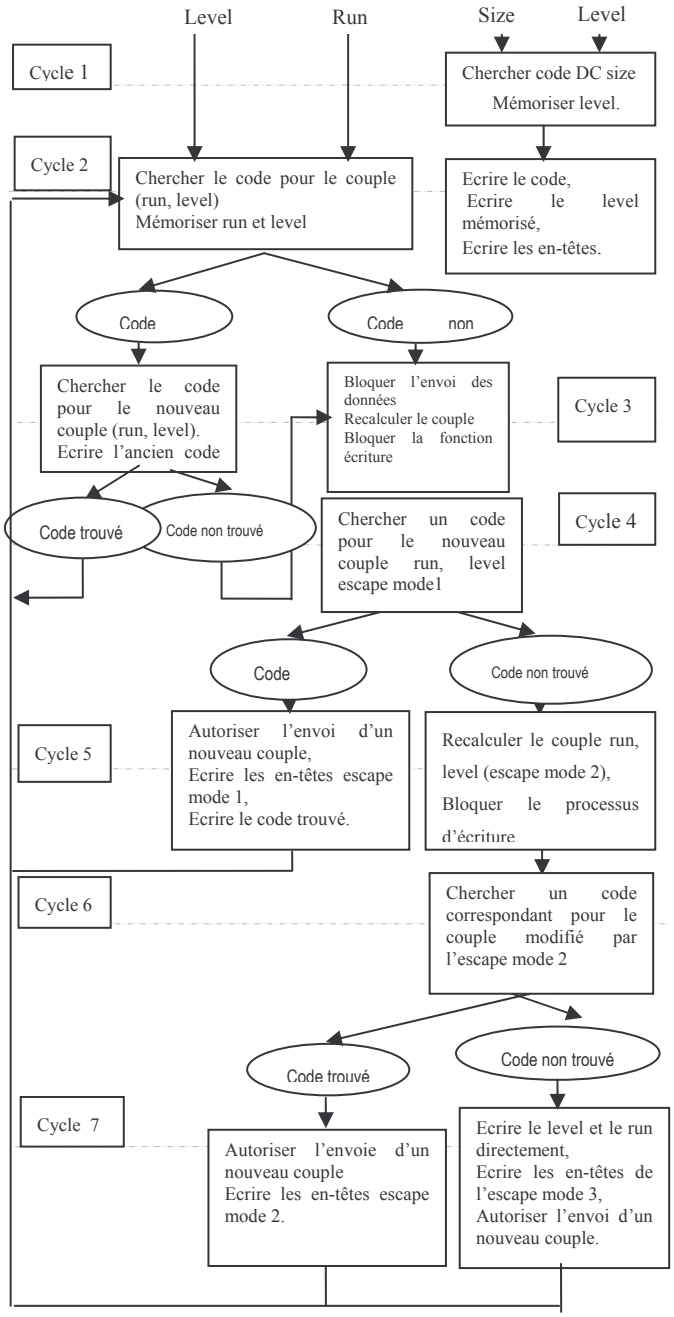

FIG. 3 Diagramme de l'unité de codage

2ème cas

Si la longueur du code lu au cours du cycle 2 est différente de 0, alors le code correspondant au 1er couple (run, level) est trouvé dans la mémoire. Dans ce cas, nous stockons le code correspondant dans un tampon et nous activons la fonction

écriture pour le cycle suivant. En même temps nous récupérons le nouveau couple (run, level).

Cycle 4 :

Si nous sommes dans le mode escape numéro 1, nous recherchons le code correspondant au couple (run, level) re-calculé au cours du cycle 3.

Cycle 5 :

– 1er cas :

Si le code correspondant au nouveau couple re-calculé n'a pas été trouvé, alors aucune opération de lecture ou d'écriture n'est effectuée lors du cycle suivant. Le drapeau «busy» reste à 1. Nous re-calculons alors le couple en changeant cette fois-ci la valeur du run à partir des tableaux produits à cet effet et nous passons ainsi à l'« escape mode 2 ».

– 2ème cas :

Si le code correspondant au couple recalculé est trouvé, nous repassons au mode normal et le signal busy est remis à 0. Dans le cycle suivant, nous récupérons le nouveau couple et nous activons la fonction écriture pour écrire dans le tampon, le code trouvé avec les différentes en-têtes correspondant à « l'escape mode 1 ».

Cycle 6 :

Si nous sommes dans « l'escape mode 2 », nous cherchons dans ce cycle le code correspondant au couple re-calculé une deuxième fois lors du cycle 5.

Cycle 7 :

Deux cas se présentent :

- si le code cherché au cours du cycle 6 est trouvé dans la mémoire, nous écrivons les différentes en-têtes correspondant l'« escape mode 2 » suivies du code en question. Egalement, nous autorisons l'envoi d'un nouveau couple en forçant le drapeau «busy» à 0,
- si le code cherché au cours du cycle 6 n'est pas trouvé, alors nous passons à l'« escape mode 3 ». Dans ce mode, aucune recherche de code n'est effectuée. A l'aide des différentes en-têtes correspondant à ce mode, nous écrivons le run et le level tels qu'ils viennent du bloc de prétraitement. En parallèle, nous autorisons l'envoi d'un nouveau couple de données.

1.5 Mise en forme du flux de sortie

Un problème est dû au fait qu'à la sortie, les codes ont des longueurs variables ce qui rend difficile la prédiction de la taille du flux à la sortie. Par conséquence, nous avons essayé de régulariser ce dernier en mettant les données sous forme de paquets de 8 bits. Nous avons créé un tampon dans lequel nous écrivons chaque nouveau code avec les en-têtes correspondantes. Un pointeur vers le dernier bit écrit est toujours mis à jour à chaque ajout d'un nouveau code. A chaque cycle, nous vérifions le nombre de bits existants dans le tampon. Si il est supérieur à 8, un signal « write » est activé et l'octet est envoyé à la sortie.

3. Implémentation

Ce travail a été effectué sous MENTOR. Nous avons utilisé le modèle « SIM5.0 » pour la simulation des résultats et « Précision » pour synthétiser l'architecture. Les différents résultats sont comme suit :

Cible : FPGA VIRTEX, 20 MHz (2V250fg256)

Fréquence maximale : *77* MHz

TAB. 1 : Utilisation des ressources

Ressources	utilisé	disponible	%
IOs	35	172	20.35 %
Global Buffers	1	16	6.25%
Functions Generators	2544	3072	82.85%
CLB Slices	1272	1536	82.81%
Dffs or Latches	610	3588	17.25%
Block RAMS	5	24	20.83%

4. Conclusion

Le module de codage entropique complet représente un bon exemple d'algorithme dont la complexité dépend des données. L'IP développée a été synthétisée et testée. Comparé aux autres architectures développées, notre design prend en compte de tous les « escape modes » comme cela a été spécifié dans la norme. Les résultats trouvés montrent de bonnes performances en vue d'un codage vidéo temps réel pour la TVHD. Nous exprimons nos remerciements aux laboratoires EPFL, LE2I pour leur aide et à Xilinx pour nous avoir fournit la WildcardII

Références

[1] ISO/IEC JTC1/SC29/WG11 AHG on MPEG-4 Part 9: Référence Hardware, Palma de Mallorca, Spain, October 2004

[2] "Virtex™-II X Platform FPGAs: Complete Data Sheet", DS110, Mar 5, 2004

[3] S.-M. Lei and M.-T. Sun, "An Entropy Coding System for Digital HDTV Application", IEEE Trans. On Circuit and Systems for Video Technology Vol.1 No.1, pp.147- 155, mars 1991.

[4] Stephen Arthur Molloy, Low Complexity, Low Power VLSI Architectures for Image Codec's, thesis University of California, pp 149-181, Los Angeles 1997.

[5] ISO/IEC 14496-5:2001.Coding of Audio-Visual Objects – Part 5: Reference Software, 2d Edition, 2001