ENIS (CES LAB) & INSA (IETR)

Rapport d'avancement

Reconfiguration et implémentation optimisée du codec d'images fixes LAR par une approche méthodologique RVC (Reconfigurable Video Coding)

Khaled Jerbi

05/05/2010

Encadré par :

Pr. Mohamed ABID (ENIS)
Pr. Olivier DEFORGES (INSA)
Mr. Mickaël RAULET (INSA)

Rapport d'avancement

Préface :

Dans ce document nous présentons en premier lieu les différentes architectures développées pour le codeur LAR ainsi que les tests effectuées sur la transformée Hadamard. Nous présentons aussi les résultats de synthèse en particulier la consommation des acteurs en termes de coups d'horloge. Les constatations et les comparaisons seront présentées au fur et à mesure. Les travaux en cours feront la clôture.

N.B : Nous avons commencé par réaliser plusieurs tests pour mieux comprendre le comportement des acteurs générés par Cal2HDL par rapport aux machines d'état, gardes, structures « if », les fonctions et l'accès simultané aux tableaux en écriture et en lecture.

I- La programmation en CAL

Le langage CAL [1] et les outils associés nous permettent de réduire la complexité en montant en niveau d'abstraction. Comme nous l'avons constaté, le CAL lui-même il peut s'écrire en plusieurs niveaux (voir figure 1) : un haut niveau multi-token très simple et facilement lisible, un bas niveau nécessaire (pour le moment) pour la génération du code HDL avec Open Forge [2], un code pipeliné dans lequel nous avons essayé de mieux gérer les mémoires pour un traitement plus rapide et finalement un code optimisé en mémoires et en traitement qui ressemble finalement à un code VHDL mais qui est très long et par conséquent illisible.
L'approche adoptée consiste à développer un code CAL au plus haut niveau ensuite le valider rapidement sur une plate forme logicielle avant de passer à un niveau inférieur.

L'usage optimal du langage CAL pour un traitement complexe est basé sur le développement d'acteurs simples réalisant ce traitement en agissant convenablement sur les différents jetons échangés. Plus l'acteur est simple et réutilisé, moins on consomme de logique derrière.
Il faut éviter les algorithmes qui utilisent plusieurs mémoires internes parce que ça crée une dépendance des traitements par rapport aux valeurs stockées ce qui ramène à un code réalisant tout un traitement dans un seul acteur. Ce dernier, par conséquent, sera très long et aura plusieurs ports d'E/S. Un design en CAL doit alors favoriser les acteurs qui échangent les données au lieu de les enregistrer c'est d'ailleurs très important pour l'aspect data-flow.

II- Le Flat LAR

Dans cette partie du codeur LAR [3] nous nous sommes intéressés à la partie quad-tree puisqu'elle génère l'image des tailles qui est indispensable pour avoir une transformée Hadamard variable en fonction de l'activité dans l'image.

II.1- L'architecture VHDL

C'est une architecture inspirée d'un code VHDL déjà établi. Elle se base sur l'idée que nous pouvons générer l'image des tailles avec un minimum de mémoire interne en considérant seulement les résultats des max et des min des quatre quadrants formant chaque block 8x8 de l'image. L'image en input est consommée pixel par pixel et ligne par ligne.

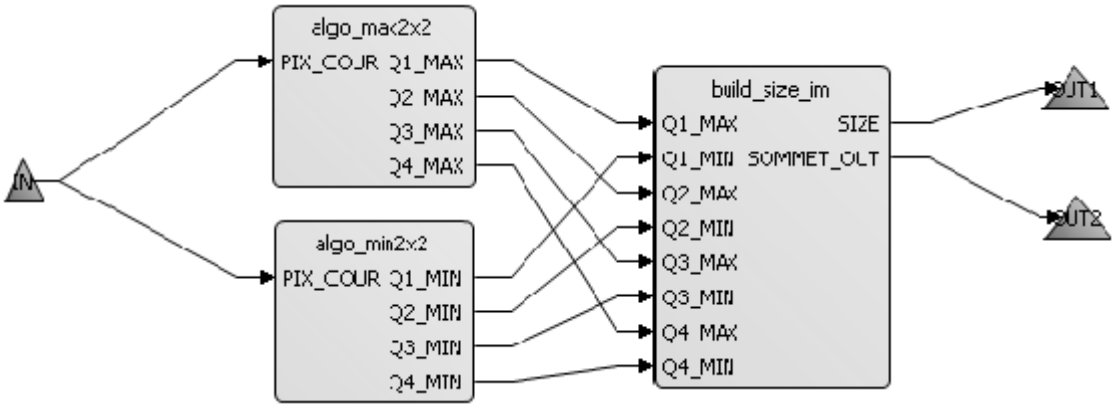

Fig.1 Quad-tree (VHDL)

Cette architecture implique une latence considérable puisque dans chaque slice de 8*XSIZE nous devons attendre (7*XSIZE+8) pour commencer à avoir les résultats en output. Toutefois, il faut noter que des traitements partiels de comparaison sont réalisés en même temps que les jetons sont consommés.
L'algorithme en question comporte plusieurs structures conditionnelles « if » pour avoir des informations sur le quadrant courant, la max courant et comment mettre à jour le x et le y locaux et globaux (x et y en local sont les coordonnées dans un block 8x8 et x et y en global c'est les coordonnées dans le slice 8x8). Nous avons donc essayé de diminuer le nombre des conditions en utilisant des « gardes ». Il est à noter que les gardes en CAL ne consomment pas de temps contrairement à un test de « if » qui va prendre un coup d'horloge ultérieurement.

II.2- L'architecture CAL

Dans cette architecture nous avons essayé de profiter au mieux des points forts du langage CAL en visant un design data-flow dans lequel nous avons séparé la partie gestion de mémoire de la partie traitement tout en essayant de favoriser l'utilisation d'un même acteur pour réaliser des fonctions semblables en agissant sur des paramètres. L'astuce était donc de changer l'ordre des données de façon à avoir, pour chaque acteur, des sorties correspondant exactement aux entrées souhaitées par l'acteur suivant comme présenté dans la figure 2.

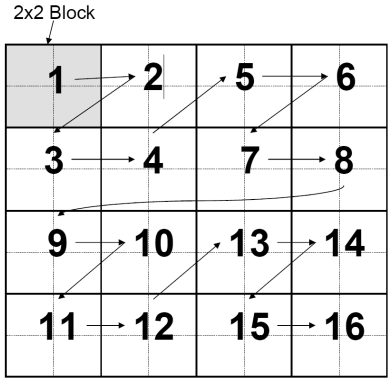

Fig.2 Ordre des blocks 2x2

Ensuite nous avons développés seulement 3 acteurs très simples (Max, Min et Gradstep) pour réaliser tout le Quad-tree qui finalement va comporter 8 acteurs qui représentent des duplications des 3 acteurs de base comme le montre la figure 3.

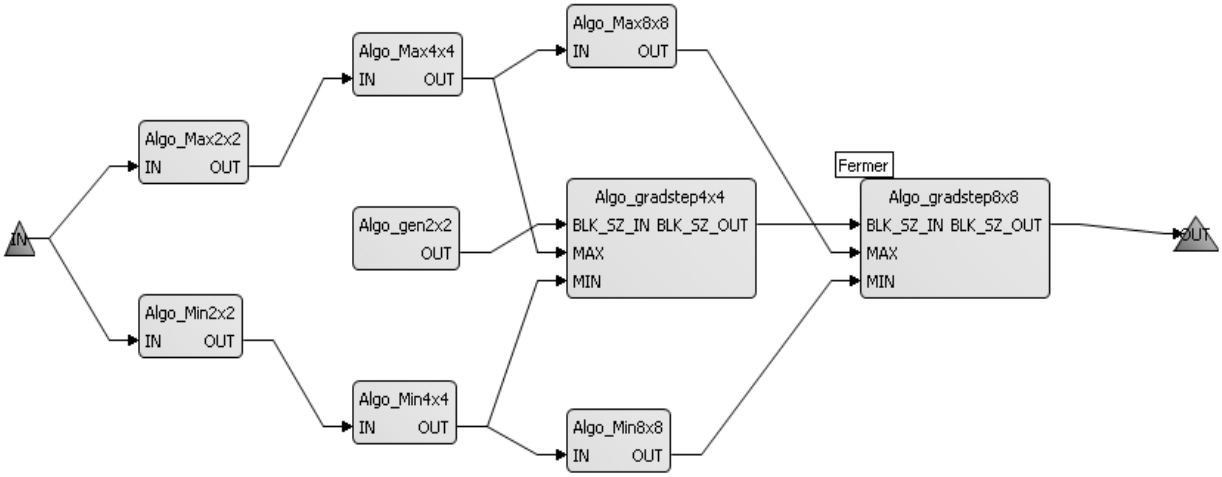

Fig.3 Quad-tree (CAL)

Si nous considérons l'acteur max, ce dernier génère un max pour quatre valeurs introduites quelconques. En mettant les données comme présenté dans la fig2 dans « algo_max_2x2 », on aura en sortie les valeurs des max_2x2 dans le bon ordre pour appliquer directement «algo_max_4x4 » qui a exactement le même code CAL que l'acteur « algo_max_2x2 ». Idem pour « algo_max_8x8 » et le même principe s'applique aussi sur les trois acteurs « min » et les deux acteurs « Gradstep ».

Cette architecture implique elle aussi une latence considérable puisqu'il faut attendre 7*XSIZE+7 pixels pour commencer à avoir des sorties sans avoir de résultats partiels durant le stockage. Nous avons donc pensé à remédier à ce problème en utilisant une gestion de mémoire en ping-pong. Avec cette architecture nous avons doublé la taille de la mémoire de stockage en 2* 8*XSIZE. Ensuite nous stockons 8*XSIZE valeurs dans la première moitié et on commence à avoir des sorties dans le bon ordre tout en remplissant en même temps l'autre moitié de la mémoire. Et ça continue ainsi en basculant d'une demi-mémoire à une autre.
Bien que cette architecture garde une importante latence initiale comme les deux précédentes, elle nous permet d'éliminer les latences ultérieures en passant d'un slice 8*XSIZE à un autre ce qui donne par la suite une régularité parfaite des sorties.

III- Le codage de fréquence (la transformée Hadamard)

Dans cette partie, nous avons déjà développé 4 architectures plus ou moins différentes pendant le stage de master (janvier-mars 2009).

- Une architecture super rapide avec 4 blocks H2 en parallèle et des sorties à chaque coup d'horloge (4 sorties pour la H2, 16 pour la H4 et 64 pour la H8) en considérant un papillon.
- Une architecture semblable en considérant un seul block H2.
- Une architecture avec transfert des tailles pour une transformée conditionnelle.
- Une architecture sans transfert de données avec des transformées s'appliquant sur tous les blocks de l'image et à la fin une unité de tri pour prendre seulement les résultats utiles.

III.1- Architecture basée sur les H2

En essayant de développer l'Hadamard en haut niveau (article EMC'10), nous avons vu qu'on peut ramener les transformés H4 et H8 en H2 en ajoutant des unités de gestion mémoire ce

qui correspond parfaitement avec l'approche CAL qui tend à unifier les acteurs. Nous avons donc eu une nouvelle architecture. La figure 4 présente l'Hadamard H4 réalisée avec une H2 et deux unités de gestion mémoire.

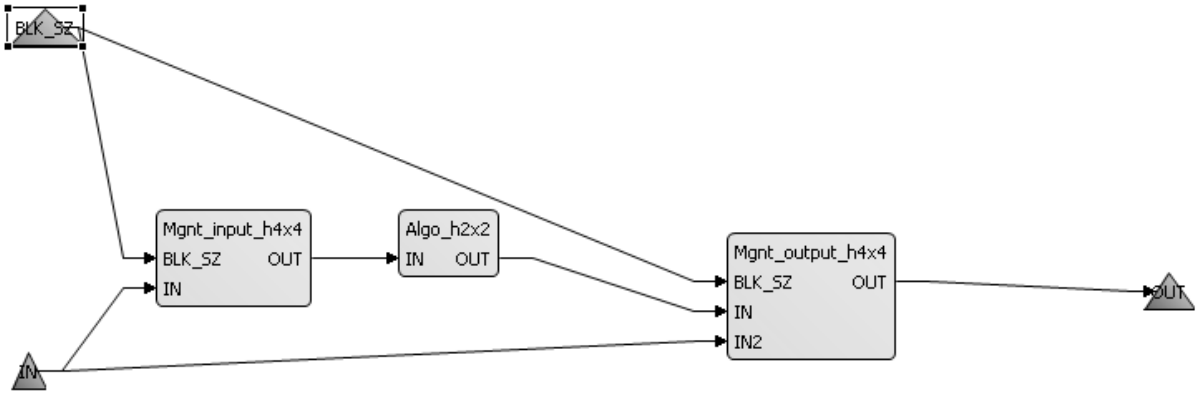

Fig.4 H4 réalisée avec une H2

Au début, la H2 n'était pas pipelinée. Elle sauvegarde 4 jetons un par un dans un tableau de 4 cases. Ensuite réalise le traitement tout en sauvegardant les 4 résultats dans un autre tableau de 4 cases et fini par sortir ces résultats un par ce qui donne un temps de traitement de 17 coups d'horloge (4 pour lire, 4 pour écrire et 9 pour le traitement puisqu'il ya plusieurs accès dans la même mémoire et 4*3 opérations d'addition et de soustraction qui, forcément, ne s'effectuent pas au même instant. Voir simulation sans la figure 5.

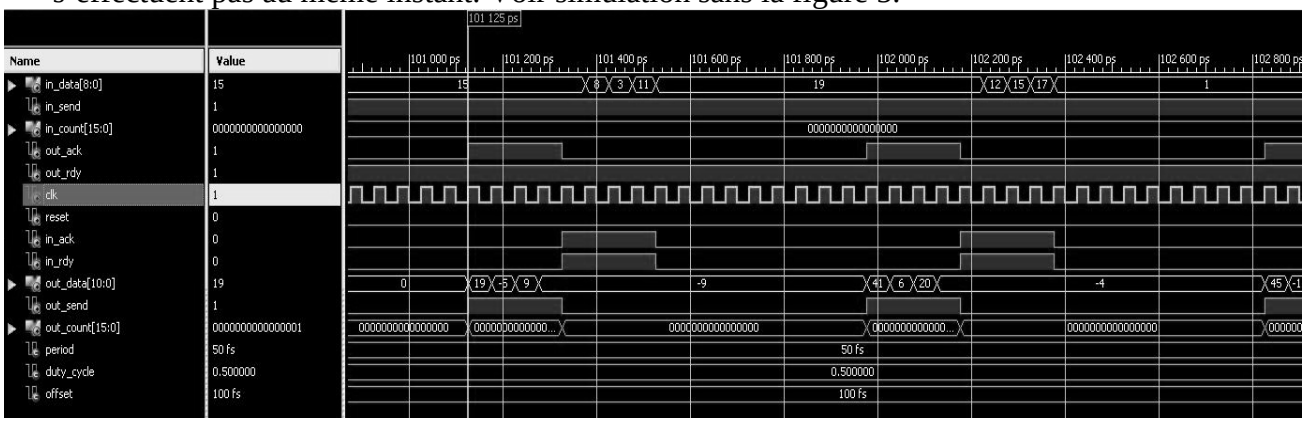

Fig.5 Simulation H2 sans pipeline

En ajoutant le ping-pong avec une mémoire de 8 cases, les résultats ont commencé à s'améliorer. Ainsi, nous avons 4 coups d'horloge pour lire les 4 premières valeurs ensuite arrive le traitement qui prend 14 coups d'horloge pour chaque ensemble de 4 valeurs sorties. Le retard est causé par le faite que nous continue à utiliser le même tableau pour lire et écrire et aussi puisque une opération x1+x2-x3-x4 par exemple s'effectue en deux coups d'horloge.
La solution était donc d'appliquer le ping-pong sur deux tableaux distincts au lieu de deux moitiés d'un même tableau. Avec cette méthode nous avons obtenu un traitement des 4 valeurs en 7 coups d'horloge ce qui est très proche du cas optimal (4 coups d'horloge).

Dans une troisième architecture, nous avons considéré 8 registres et nous avons optimisé les calculs de façon à obtenir le cas idéal (4 coups d'horloge pour traiter et sortir 4 jetons !).

Ces résultats, bien que parfaits n'étaient pas retenus pour le design final de l'Hadamard complète puisque nous avons développé du CAL très long et en très bas niveau ce qui ne

correspond pas à notre objectif derrière l'utilisation du CAL. Toutefois, nous gardons l'idée qu'avec le CAL on peut toujours optimiser si nous descendons en niveaux.

III.2- Architecture avec sorties concaténées sur un seul bus

Dans cette architecture, nous avons visé un traitement très rapide semblable à l'utilisation de plusieurs FIFOs pour un échange simultané des données à la différence de concaténer ces sorties dans un seul bus. L'idée était, pour la H2 par exemple, d'envoyer 4 données de 9 bits concaténées dans un bus de 36 bits pour chaque pixel reçu de l'image initiale. Des unités de gestion de mémoire ont été prévues pour envoyer les données dans le bon ordre vers les Hadamards suivantes et en même temps réaliser une sauvegarde conditionnelle des sorties de l'Hadamard précédente en fonction de des coordonnées position X et position Y (voir fig8).

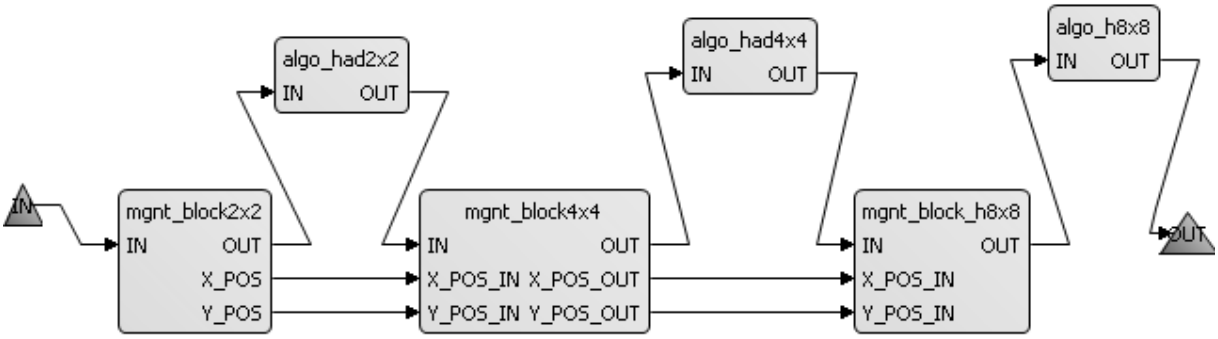

Fig.8 Architecture avec concaténation des données

Nous avons commencé par la H2 et nous avons obtenu de très bons résultats surtout que Cal2HDL arrive à pipeliner le traitement une foi l'action est unique et répétitive. Par conséquent, nous avons obtenu une fréquence des sorties 4 fois plus rapide que la fréquence max du circuit !

L'inconvénient est que cette architecture ne peut être étendue pour les Hadamards H4 et H8 puisque ces dernières demandent des bus de largeur énorme arrivant jusqu'à 945 bits pour la H8 ce qui n'est pas supportable ni par les outils de génération de code VHDL (Cal2HDL ou le back-end VHDL de Orcc) ni par les outils de synthèse tel que ISE de Xilinx qui ne peut gérer des données de tailles supérieures à 32 bits.

IV- Récapitulation et résultats de synthèse

IV.1- L'Hadamard H2

Dans le tableau 1 nous présentons les résultats de synthèse des différentes architectures de l'Hadamard 2 en fonction des caractéristiques de fréquences et de consommation de surface :

Archi H2 / Carac	Sans pipeline	Pipeline 2 tabs	Pipeline 8 buff	Pipeline concat
Fréquence max (MHz)	149.	157.5	214.4	154.1
Fréquence des sortie MHz)	8.8	22.5	214.4	154.1
Number of 4 input LUTs	234 (1%)	269 (1%)	614 (1%)	147 (1%)
Number of	160 (1%)	169 (1%)	369 (2%)	103 (1%)

occupied Slices				

TAB.1 Récap H2

Interpretation:
Le design sans pipeline représente les moins bons résultats. L'absence de pipeline diminue considérablement la fréquence et les tableaux pour le stockage des données en entrée et pour le stockage des résultats augmentent le nb des LUT. L'avantage de cette architecture est le temps de développement qui est très rapide puisque nous utilisons séparément une action pour la lecture, une action pour le traitement et une autre pour l'écriture des données.
L'architecture de la H2 pipelinée avec deux tableaux en entrée garde l'espect haut niveau du développement CAL. La fréquence de 22.5 MHz est relativement élevée pour un traitement de l'image en hardware. La consommation en surface est pratiquement dans la moyenne des quatre architectures.
L'architecture avec 8 buffers est la plus gourmande en consommation de surface mais en même la plus rapide. Le seul inconvénient est que pour le développement nous avons dû écrire du CAL en bas niveau.
Du faite qu'il contient une seule action, le design de l'Hadamard avec concaténation des données consomme le moins de surface. La fréquence des sortie est très élevée (fréquence max du circuit).
Entre 22.5, 154 et 214 MHz, nous nous demandons la quelle est la plus adéquate au reste du design du LAR ? Est-t-il vraiment nécessaire d'aller aussi vite que 200MHz dans l'Hadamard alors que 22 MHz sont suffisants pour traiter des images de tailles usuelles en temps réel ?

IV.2- Le Quad-tree

Pour le Flat LAR nous avons considéré le Quad-tree auquel nous avons développé 3 architectures différentes : une architecture écrite en VHDL, une autre écrite en CAL réalisant le même algorithme que le code développé en VHDL et finalement une architecture haut niveau développée en CAL et pipelinée avec un ping-pong. Le tableau 2 récapitule les résultats de synthèse des différents designs.

Quad / Carac	VHDL	VHDL_to_CAL	CAL
Fréquence max (MHz)	149.8	125.6	249.8
Fréquence des sortie MHz)	149.8	62.8	24
Number of 4 input LUTs	750 (2%)	912 (2%)	680 (2%)
Number of occupied Slices	692 (4%)	594 (3%)	457 (2%)

Tab.2 Récap Quad-tree

Interprétation :
Du faite qu'il soit développé directement en bas niveau en utilisant VHDL, le premier design est forcément le plus rapide avec un pipeline permettant d'avoir une sortie tout les coups d'horloge du système.

Le deuxième design écrit en CAL et inspiré de l'algorithme VHDL garde une fréquence très élevée de 62 MHz mais consomme plus de logique puisque le design généré par Cal2HDL comporte plusieurs signaux de synchronisation et surtout des FIFOs pour l'échange des données ce qui peut être optimisé en développant directement en VHDL.
La dernière architecture développée en utilisant des acteurs CAL dupliqués présente la fréquence maximale de circuit la plus élevée. Mais avec une sortie tous les 10 coups d'horloge la fréquence des outputs est de 24 MHz. Comme pour la Hadamard H2, cette fréquence peut être encore améliorée en utilisant un CAL de plus bas niveau pouvant arriver à une sortie par deux coups d'horloge et ainsi une fréquence de sortie de 125 MHz !

V- Conclusion et perspectives

Les différentes architectures développées que ce soit pour le Flat LAR ou pour le codeur de fréquence nous ont permis d'explorer au mieux les performances du générateur de code HDL (Cal2HDL). Les codes CAL inspirés du code VHDL ont donné de très bons résultats mais ils sont très bas niveau ce qui a engendré des codes complexes, difficiles à lire et non-réutilisables. L'architecture CAL est devenue très intéressante quand nous avons ajouté le pipeline avec ping-pong et elle garde encore une marge d'optimisation. Ce design est en plus validé sur une plateforme logicielle avec Orcc en codant et décodant parfaitement des images et des vidéos.

Suite aux travaux réalisés, nous avons ajouté des acteurs de normalisation et de quantification pour achever la partie Hadamard. Actuellement, nous essayons d'utiliser les codes *.xlim générés pas Orcc pour en générer du Verilog avec le hardware compiler d'Open Forge. Ce compilateur supporte la majorité des codes xlim de Orcc. Les codes non supportés sont en cours d'étude.
Nous notons aussi que le hardware compiler d'Open Forge peut fournir des informations très importantes sur le fonctionnement de chaque action et de chaque acteur. Ces informations sont retrouvées ultérieurement dans la simulation. Nous envisageons alors de changer le code source de façon à visualiser ces résultats dans la console de Cal2HDL.

References

[1] J. Eker and J. Janneck, "CAL Language Report," University of California at Berkeley, Tech. Rep. ERL Technical Memo UCB/ERL M03/48, Dec. 2003.

[2] R. Gu, J. W. Janneck, S. S. Bhattacharyya, M. Raulet, M. Wipliez, and W. Plishker, "Exploring the concurrency of an MPEG RVC decoder based on dataflow program analysis," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 19, no. 11, pp. 1646–1657, 11 2009. [Online]. Available: dx.doi.org/10.1109/fTCSVTg .2009.2031517http://hal.archives-ouvertes.fr/hal-00440492/en/

[3] O. D´eforges, M. Babel, L. B´edat, and J. Ronsin, "Color LAR Codec: A Color Image Representation and Compression Scheme Based on Local Resolution Adjustment and Self-Extracting Region Representation," IEEE Trans. Circuits Syst. Video Techn., vol. 17, no. 8, pp. 974–987, 2007.

Publications

"Fast Hardware implementation of an Hadamard Transform Using RVC-CAL Dataflow Programming"
Khaled Jerbi, Matthieu Wipliez, Micka¨el Raulet, Olivier Déforges, Marie Babel and Mohamed Abid

Lu et approuvé par Mr. Mohamed ABID

.