

Compression JPEG2000 par High-Level-Synthesis

Rapport de projet d'Implantation de Système Embarqué

Sommaire

Introduction	3
1 Présentation du sujet	4
1.1 Système existant	4
1.2 Objectifs	4
2 Compression JPEG2000	5
2.1 Passage en Luminance	5
2.2 Transformation en ondelettes	5
2.3 Quantification	6
2.4 Passage en BitStream	6
3 Gestion de projet	8
3.1 Flot de conception	8
3.2 Diagramme de Gantt	8
4 Travail effectué	9
4.1 En logiciel	9
4.2 Implantation sur carte	11
4.3 Programme de démonstration	17
4.4 Travail restant	17
4.5 Problèmes rencontrés	17
Conclusion	18

Introduction

JPEG2000 est une norme de compression ayant pour ambition d'offrir plus de flexibilité et de performances que son prédecesseur, le JPEG. Elle permet entre autres de compresser avec ou sans perte, le décodage progressif, l'introduction de régions d'intérêt ou encore l'accès aléatoire aux données. Par rapport au JPEG, cette norme propose des images de meilleure qualité, pour un taux de compression similaire.

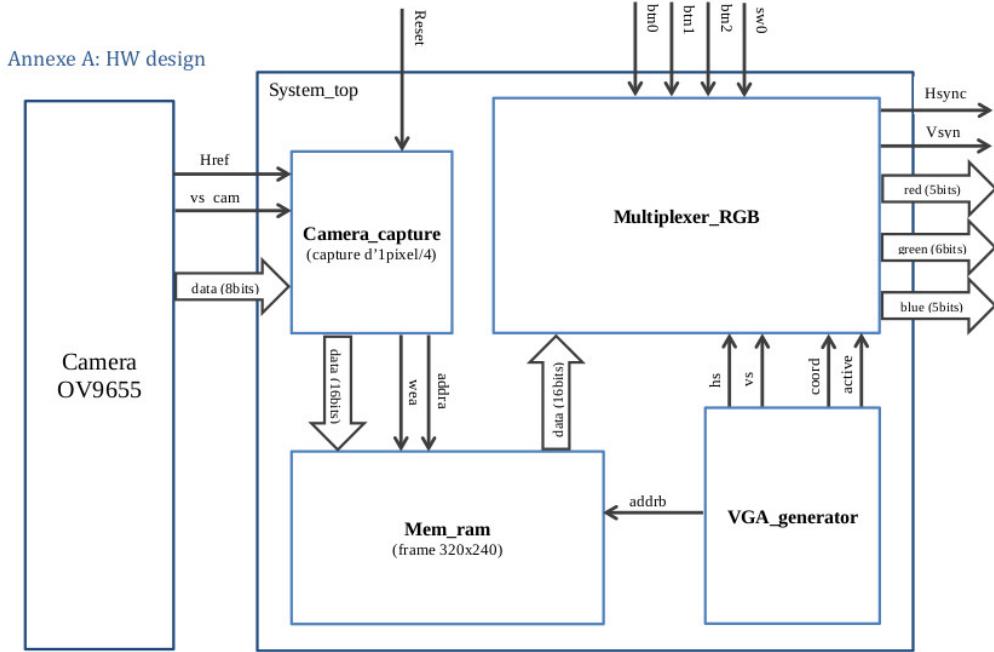
CatapultC est un outil de synthèse de haut niveau commercialisé par Mentor Graphics en 2004 et racheté par Calypto en 2011. Il permet de synthétiser du C/C++ directement jusqu'au niveau RTL. Il présente également la possibilité de faire des simulations depuis son interface graphique en utilisant des test-benchs écrits en C/C++. Il supporte également le SystemC/TLM et s'en sert pour créer l'environnement de simulation du RTL synthétisé.

Le but de ce projet est d'utiliser le logiciel CatapultC pour synthétiser un algorithme de compression JPEG2000 sur une carte Zybo Xilinx.

1 Présentation du sujet

1.1 Système existant

Le système comporte une caméra en entrée et un écran VGA en sortie. L'acquisition de l'image a été réalisée et portée sur le FPGA. A l'intérieur du système, l'image est stockée en RGB dans une mémoire RAM. Elle est ensuite envoyée vers le VGA. Les boutons poussoirs sont utilisés et permettent d'afficher une mire.



37

*Les signaux d'horloge des modules ne sont pas indiqués. CLK_CAM = CLK_VGA = 24MHz générées par le module CLK_PLL.

Figure 1: Système existant

1.2 Objectifs

Le but du projet est d'implémenter la compression d'image JPEG2000 et de la porter sur le système existant sur la carte FPGA. On utilisera pour ça l'outil de synthèse de haut niveau CatapultC.

L'algorithme de compression JPEG2000 se décompose en plusieurs tâches : la décolorisation, la transformation en ondelettes, la quantification, le codage entropique et le passage en bitstream.

On souhaite au moins arriver à porter la transformée en ondelettes et sa décompression sur la carte zybo. Le programme de démonstration consistera à afficher l'image en transformée en ondelettes et l'image décompressée sur l'écran VGA.

2 Compression JPEG2000

On voit le flux de fonctionnement de codage JPEG2000 dans la figure suivante (voir la figure 2):

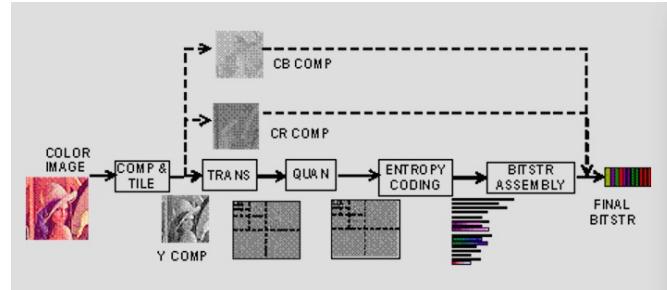


Figure 2: Diagramme de flow de fonctionnement

Le premier module est sert à couper l'image en petits morceaux et décorréliser des couleurs. Pour des photos de grand volume, on les coupe en plusieurs petits morceaux de même taille. Par exemple, une image en RGB peut être transformée en YCrCb ou RCT. Chaque tuile de chaque composant peut être traitée séparément pour éviter de travailler avec de grosses mémoires. Les données sont tout d'abord transformées dans le domaine de ondelettes, les coefficients sont ensuite quantifiés puis regroupés pour faciliter l'accès et la résolution spatiale localisée.

2.1 Passage en Luminance

On commence éventuellement à convertir l'image de modèle colorimétrique (RGB: rouge, vert et bleu) vers un modèle YCrCb (luminance, chrominance rouge, chrominance bleue) ou RCT (Reversible Component Transform). La transformation de ces espaces nous permettent d'avoir trois plans: Y, Cb et Cr. Y représente la luminance, qui nous donne une image en niveau de gris. Cr et Cb sont reliés avec les compositions de bleu et rouge. Comme le plan Y influence le plus de la vision, il ne doit subir aucune perte. En revanche, pour les plans Cr et Cb, des pertes sont tolérées.

2.2 Transformation en ondelettes

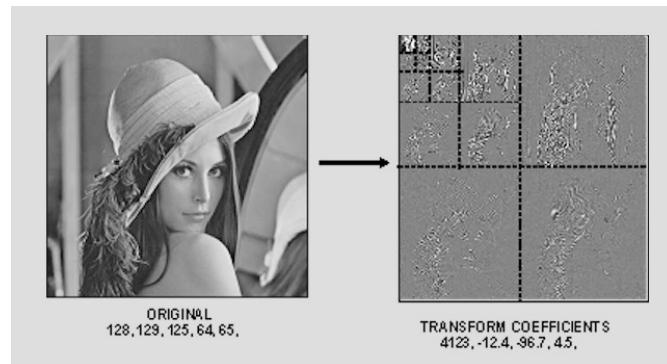


Figure 3: Diagramme de transformation en ondelette

Pour appliquer la transformée de ondelettes sur un tableau de dimension 2, par exemple une image, on pratique souvent les transformations horizontales et verticales séparément. Même s'il est possible de la concevoir sans cette séparation, le coût de compléxité en calcul s'en voit fortement augmenté et le gain n'est pas significatif. Un exemple de transformée en ondelettes est représenté dans la figure ci-dessus(voir la figure 3).

Une passe verticale génère deux sous-images : un passe-haut vertical (H_v) et un passe-bas vertical

(Lv). La passe verticale est appliquée à ces deux images, générant les quatre sous-images : HvHh, LvHh, HvL et LvLh. La répétition de ces passes sur l'image LvLh permet d'obtenir des niveaux de compression de plus en plus élevés. Etant donné que la transformée en ondelettes est une transformation linéaire, on peut permute l'ordre du filtre vertical et de l'horizontal.

2.3 Quantification

Après la transformation d'ondelette, tous les coefficients d'ondelette sont uniformément quantifiés par ce qui suit (voir la figure 5):

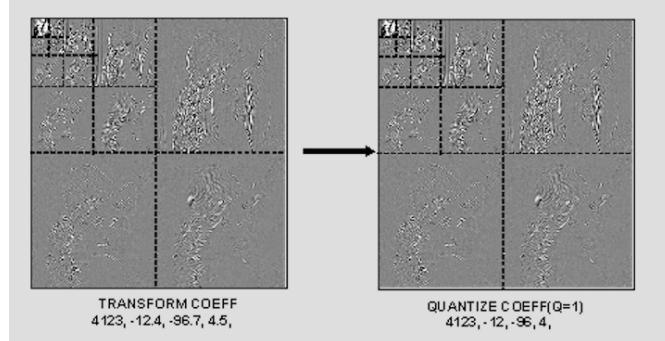


Figure 4: Diagramme de quantification

$$w_{m,n} = \text{sign}(s_{m,n}) \left\lfloor \frac{|s_{m,n}|}{\delta} \right\rfloor$$

Figure 5: Diagramme de formule pour calculer le coefficient

Où $S(m,n)$ est le coefficient de transformation, $W(m,n)$ est le résultat de quantification. Δ est la taille de l'étape de quantification. $\text{Sign}(x)$ retourne le signe du coefficient x et cette opération obtient le plus grand entier qui est inférieur ou égal à x . Un exemple de quantification est illustré dans la figure 4). Le processus de quantification convertit les coefficients d'ondelettes de nombres à virgule flottante en nombres entiers.

Le principe de la quantification est donc de convertir les coefficients d'ondelettes en entiers de sorte qu'ils puissent être plus efficacement traités par le module de codage entropique. La qualité de codage d'image n'est pas déterminée par la taille Δ du pas de quantification, mais par le flux de bits ultérieurement assemblés. La taille de l'étape de quantification par défaut est donc plus fine, par exemple, $\Delta = 1/128$.

2.4 Passage en BitStream

Le résultat venant de la quantification peut être coupé en plusieurs blocs. L'avantage de cette découpe est significatif. Premièrement, les données ayant des types différents possèdent les caractéristiques statiques différentes, on peut augmenter l'efficacité de compression en les traitant séparément. Pour plus de scalabilité, la séparation prend en compte l'importance des données.

On utilisera le codage entropique pour faire les traitements sur des blocs indépendants. Le codage entropique est un mode de compression sans perte. Le codage de Huffman et le codage arithmétique sont deux types essentiels du codage entropique.

L'avantage de codage arithmétique par rapport le codage de Huffman est que le codage de Huffman reste toujours d'un nombre entier pour coder une caractère (par exemple le codage en 1.5 bit n'est pas tolérable) mais le codage arithmétique peut le faire. Le principe du codage arithmétique est de tout d'abord de remplacer les lettres en probabilité de présence. Ensuite, on affecte à chaque lettre un intervalle entre 0 et 1 selon sa probabilité de présence et on remplace le mot entier par un nombre flottant qui lui correspond.

3 Gestion de projet

3.1 Flot de conception

CatapultC

CatapultC est un logiciel qui permet de générer automatiquement du code VHDL à partir de code C/C++. On peut aussi l'utiliser pour obtenir l'estimation de la logique requise, de la consommation du circuit...

Il existe trois règles de codage pour appliquer le code dans le CatapultC. On ne peut pas utiliser les pointeurs et faire les allocations dynamique. En plus, on ne peut qu'utiliser les paramètres fixes. Dans le bibliothèque "ac_int.h" et "ac_fixed.h", il y a deux nouveaux types de données, ac_int pour un entier et ac_fix pour un nombre à virgule fixe. Ces types permettent de mieux modéliser des données.

On peut utiliser vsim directement depuis CatapultC pour valider le module créé au niveau RTL. Le test_bench est écrit en C++ et génère du SystemC décrivant l'environnement de test du module.

Pour des raisons de linkage de librairies complexes, on n'utilise pas le vhdl généré par CatapultC. On utilise l'outil Précision qui permet de générer une netlist '.edf'. Pour éviter des conflits, il est nécessaire de spécifier à Précision de ne pas ajouter d'IO Pads. Ce réglage se fait via le menu Set Options de CatapultC.

ISE/XDK

La netlist récupérée depuis CatapultC est ajoutée au projet ISE. Il est nécessaire que ce fichier ".edf" porte le nom du composant qu'il décrit. L'intégration se fait par la modification du fichier 'system_top.vhd'. On réalise ensuite la synthèse/routage pour obtenir le system-top.bit qui permettra de générer le binaire de bootage dans XDK. La génération de ce binaire peut se faire en ligne de commande grâce à la commande bootgen. Ce binaire est chargé sur la zybo via une carte SD.

3.2 Diagramme de Gantt

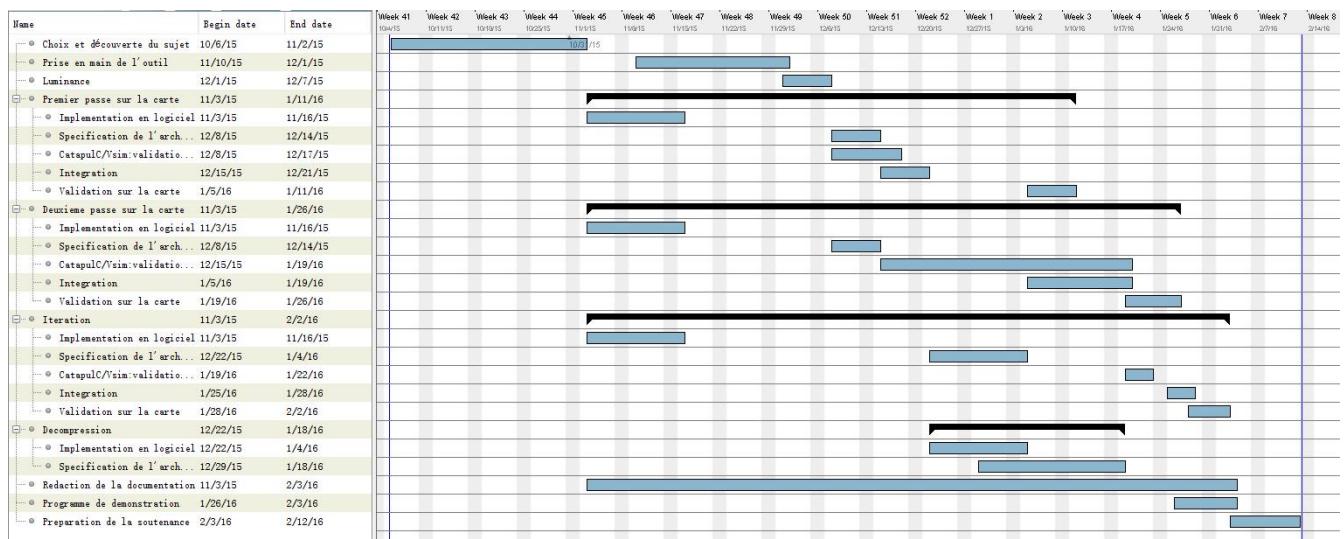


Figure 6: Diagramme de Gantt

Tout au début, nous avons travaillé ensemble pour la première partie afin de comprendre bien le sujet et familiariser l'utilisation des l'outil. Ensuite, pour économiser le temps et travailler plus efficace, nous avons réparti les tâches et travaillé en parallèle, Wanwan se concentrant sur la partie Software et Mathieu sur la partie implantation sur carte.

4 Travail effectué

4.1 En logiciel

Pour accélérer le flot de conception, nous avons convenu que le code logiciel développé devait se soumettre aux contraintes de CatapultC, en particulier utiliser les types *ac_int*.

Compression

La norme associée au JPEG2000 laisse des libertés quant à la méthode de compression utilisée. Ainsi, pour réaliser la transformée en ondelettes, nous avons le choix entre plusieurs méthodes. Nous avons fait le choix d'utiliser la décomposition de Haar pour sa simplicité de calcul : on peut la faire en traitant uniquement des entiers. Réaliser un niveau de compression revient à produire quatre images de largeur et de hauteur deux fois inférieures. Il s'agit des passes_haut et passes_bas décrits dans la partie 2.2 : Transformation en ondelettes. Le code qui effectue un niveau de compression est décrit en Annexe 1.



Figure 7: l'image origine



Figure 8: l'image compressée une fois

Décompression

On obtient l'image décompressée par l'inverse du procédé de compression plus un agrandissement de l'image LhLv. On peut ainsi observer les différents niveaux de décompression qui pourraient s'afficher successivement aux yeux de l'utilisateur au cours d'une décompression.



Figure 9: image compressée en niveau 4



Figure 10: image compressée en niveau 4 et totalement décompressée



Figure 11: image compressée en niveau 4 et décompressée sur 3 niveaux



Figure 12: image compressée en niveau 4 et décompressée sur 2 niveaux



Figure 13: image compressée en niveau 4 et décompressée sur 1 niveau

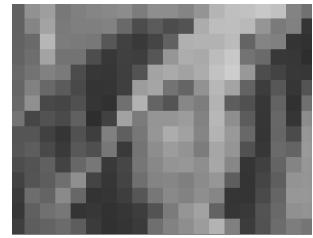


Figure 14: image compressée en niveau 4 et décompressée sur 0 niveaux

Il n'y a pas de différence significative entre l'image décompressée et l'image origine si le nombre de niveaux de décompression est inférieur à 5. Au delà, il y a un problème dû à la réduction de la hauteur de l'image. En effet, quand cette dernière devient impaire, il y a une ligne de pixels qui n'est pas traitée à la compression qui suit, ce qui est problématique au moment de reconstruire l'image au cours de la décompression.



Figure 15: image compressée en niveau 6 et totalement décompressée

Résultats

Bien que la partie significative en terme de réduction des tailles des données se fasse dans le passage en bitstream (agrémenté du codage entropique), on s'aperçoit que la seule transformée en ondelettes réduit déjà légèrement la taille des fichiers. Ces réductions ont été mesurées et relevées dans le tableau ci-dessous.

Niveau de compression	0	1	2	3	4
Taille de l'image compressée (Ko)	279	242	233	233	229
Taux de compression(%)	100	86.7	83.5	83.5	82.1

Table 1: Réduction de la taille des images

4.2 Implantation sur carte

Passage en luminance

Comme indiqué dans la partie théorique sur la compression JPEG2000, on décompose l'image en trois composantes et on traite chacune d'entre elles indépendamment. Parce que nous avons des ressources limitées sur la carte, nous nous contenterons de travailler sur la luminance, qui correspond à l'image en niveau de gris et donc à un résultat assez visuel à l'écran. Le passage en luminance nous permettra également de réduire la taille de la mémoire utilisée : les "mots" passeront de 16 à 8 bits et on pourra ajouter une deuxième mémoire, nécessaire si on veut faire notre traitement d'image par passage en ondelettes.

- **LUM2RGB**

But : Transformer les données RGB 16 bits allant de camera_capture vers la mémoire en des données Y (luminance), contenant seulement 8 bits significatifs.

On implémente ce module sous la forme d'une fonction vhdl dans "packageVGA.vhd", cette fonction est appelée dans le system top.

- **RGB2LUM**

But : A l'inverse, transformer les données Y 8 bits allant de la mémoire vers le VGA_generator en des données RGB 16 bits.

On implémente ce module sous la forme d'une fonction vhdl dans "packageVGA.vhd", cette fonction est appelée dans le system top.

- **Changement de la taille des mémoires utilisées**

But : Réduire la quantité de RAM utilisée par la mémoire dégager de la RAM libre et, plus tard, effectuer les traitements d'image nécessaires.

On crée une nouvelle mémoire en utilisant Coregen. Cette mémoire est identique à la précédente à ceci près qu'elle contient des mots de 8 bits au lieu de 16, et prend donc deux fois moins de ressources sur la carte. Les fichiers générés sont "mem_lum.ngc" et "mem_lum.vhd".

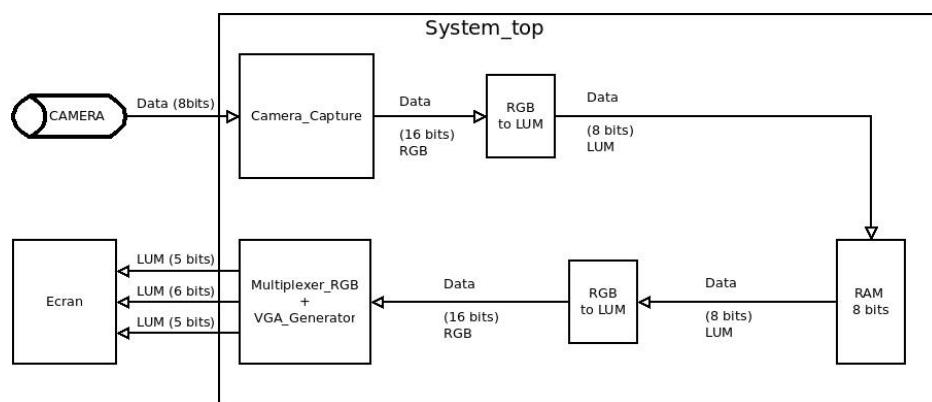


Figure 16: Système avec luminance

- **Validation**

- Dans ISE : au moins 50% des ressources de ports RAM
- Image en nuances de gris à l'écran.



Figure 17: Capture d'écran du passage en luminance porté sur la carte zybo

Une passe complète

But : Obtenir l'image transformée en ondelettes avec un niveau de compression de 1.

Validation : Affichage à l'écran de l'image après une itération de la décomposition en ondelettes. On doit avoir les quatres images LvLh, LvHh, HvLh et HvHh. L'image LvHh doit faire apparaître les gradients horizontaux alors que HvLh doit faire apparaître les gradients verticaux.

Puisque nous sommes limités par la quantité de RAM, nous ferons les séparations verticales et horizontales d'un seul coup, dans le même module CatapultC.

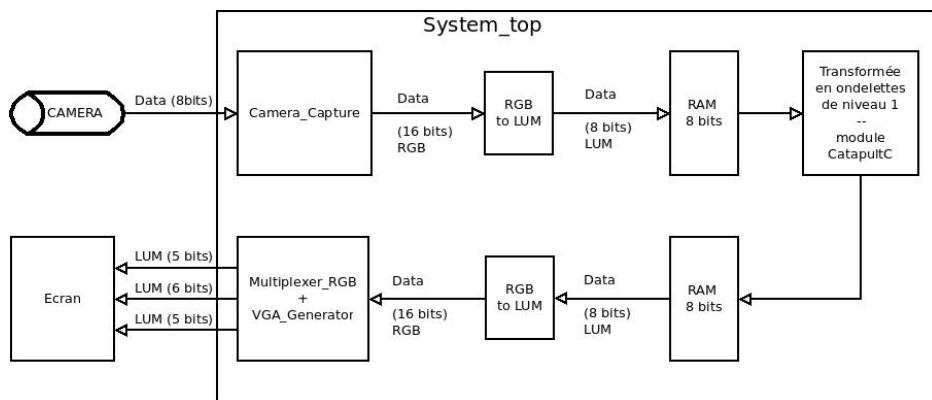


Figure 18: Système avec une seule itération

On remarquera que, pour faire apparaître plus nettement les gradients à l'écran, et ainsi valider notre implantation, nous avons pris les valeurs absolues des différences calculées au niveau des passeshaut. Dans le même but nous avons enlevé la division par 2, nécessaire à la reconstruction de l'image pour la décompression.

Ayant fait les deux séparations d'un seul coup, nous avons malencontreusement inversé les positions des images LvHh et HvLh. Ainsi, sur la capture d'écran ci-dessus, on voit que les contours horizontaux du téléphone sont mis en évidence sur l'image en haut à droite (HhLv) alors que les contours horizontaux le sont sur l'image en bas à gauche (LhHv). Ca aurait été l'inverse si nous avions appliqué les séparations horizontales et verticales successivement

Nous obtenons bien l'image souhaitée à l'écran :



Figure 19: Transformée en ondelettes, niveau 1

Itération des passes

But : Pouvoir réaliser la transformée en ondelettes avec un niveau de compression allant de 0 à 7 (0 correspondant à recopier l'image d'origine sans la traiter).

Pour pouvoir itérer les passes faites sur l'image, il nous faut pouvoir effectuer un certain nombre de traitements de l'image. Or, étant limités par les capacités de la zybo, on ne peut pas ajouter autant de mémoires que de passes. On a donc décidé d'utiliser nos deux mémoires RAM, tour à tour en tant que source et destination.

Avec cette solution, nous nous devions toutefois d'éviter les problèmes suivants :

- Camera_capture souhaite écrire les données venant de la caméra dans *ram1*. Or, pendant le traitement de l'image, nous voulons également utiliser la mémoire pour stocker une image intermédiaire, donc la lire et l'écrire.

Pour remédier à ce problème, nous avons implémenté une machine à état (fsm) dans le system top du projet ISE. Pour distinguer les différents états décrits, nous avons eu besoin de modifier le module *Camera_Capture* pour en faire remonter le signal *cam_writing*, qui nous dit si *Camera_Capture* est en train d'écrire une image dans *ram1*.

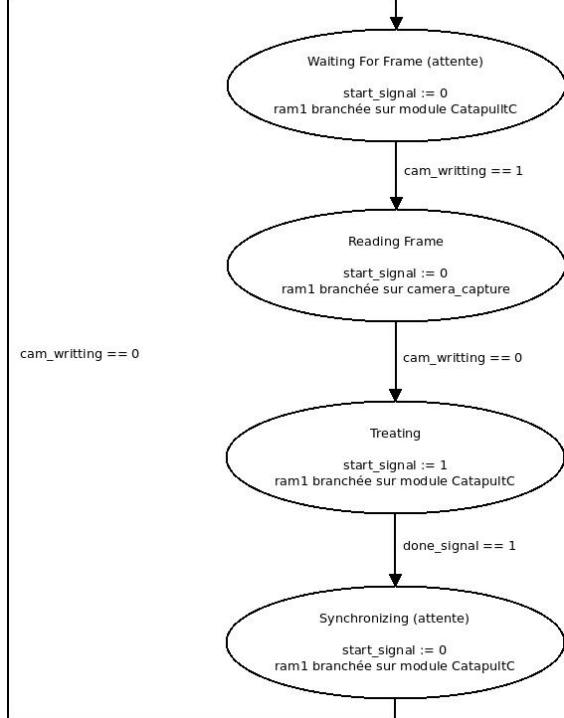


Figure 20: Machine à états de la gestion de ram1

- L'autre problème posé par cette solution est que la sortie VGA lit en continu ce qui se situe dans

ram2. Or pendant qu'on fait le traitement d'image, cette mémoire n'est pas stable, et ses ports de lecture sont utilisés. Pour remédier à cela, on a décidé de gagner de la place pour pouvoir ajouter une troisième mémoire *ram3*, qui ne contiendra que les images complètement traitées et ne sera lue que par les modules consacrés à la sortie VGA.

Pour gagner de la place, nous avons décidé de réduire la taille des mémoires en optimisant le système d'adressage. Dans la version que nous avons récupérée, les coordonnées (x,y) étaient concaténées. L'image faisant 320*240 pixels, il fallait 9 bits pour coder l'adressage en x et 8 en y, l'adresse totale comportait donc 17 pixels et la mémoire 131072 mots. Le système d'adressage que nous avons reconnu code l'adresse (x,y) par $320*y + x$. On réduit ainsi le nombre de mots à $320*240 = 76800$ mots. On a donc la place d'ajouter une troisième mémoire.

Les fichiers donnés en entrée de CatapultC ont été adaptés avec ce nouveau système d'adressage. Concernant le projet ISE, nous avons implémenté ce changement d'adresse via une fonction *map_addr* dans *packageVGA.vhd*. Cette fonction est appelée dans *Camera_Capture* pour le stockage des données de la caméra et dans *VGA_generator* pour la lecture des pixels par le VGA.

Ainsi, l'architecture finale de notre solution est la suivante :

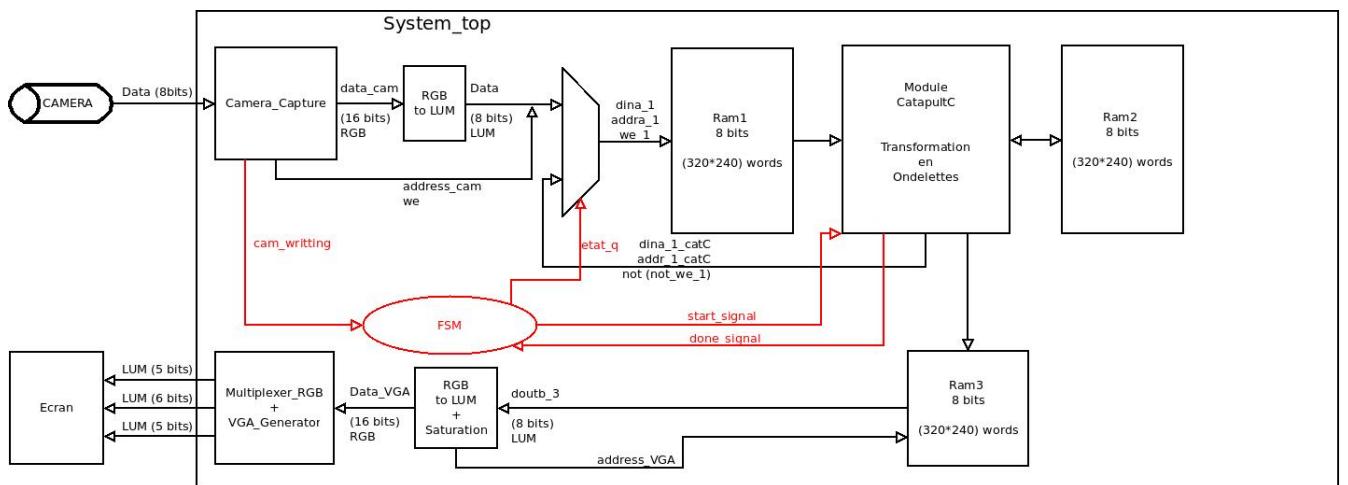


Figure 21: Architecture finale

Validation : Affichage des 8 différents niveaux de compression à l'écran avec mise en évidence des gradients horizontaux et verticaux pour chacune des images intermédiaires. Voici quelques captures d'écran de la manipulation :



Figure 22: Transformée en ondelettes, niveau 2, saturé



Figure 23: Transformée en ondelettes, niveau 4, saturé

Décompression

Le passage de la décompression dans CatapultC a du être démarré dans l'urgence à cause du retard qu'avait pris le projet. CatapultC lève beaucoup de warnings sur les boucles concernant l'algorithme de décompression, qui fait que le logiciel HLS saute ces boucles, rendant le module vhdl inutile. N'ayant pas pu trouver l'origine de ces warnings et le projet touchant à son terme, nous n'avons pas pu mener cette partie à son terme.

Optimisation

CatapultC nous donne pour chacune de nos solutions le *latency_time*, qui correspond à une estimation du temps de traitement d'une image. La caméra liée à la carte zybo capture 24 images par secondes. Pour ne pas ralentir la fréquence, il nous faudrait donc avoir un *latency_time* inférieur à 41 ms.

Constatant que ce temps n'était pas satisfaisant (216 ms), nous avons souhaité optimisé l'utilisation faite des deux mémoires ram1 et ram2 au cours du traitement. Le passage de l'ancienne version (voir annexe 2) à la nouvelle (voir annexe 3) a apporté une réduction significative du *latency_time* (116 ms) sans impliquer une augmentation trop forte de l'*area_size*.

On a ensuite voulu améliorer encore le temps d'exécution en déroulant certaines boucles. Le rapport de répartition du temps d'exécution de CatapultC ci-dessous nous a fait cibler la boucle *passe_y* en priorité. Puis, nous avons essayé de dérouler toutes les boucles intéressantes, comme montré ci-dessus.

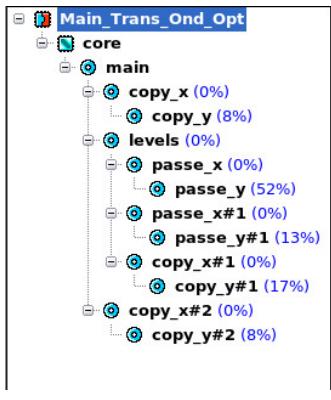


Figure 24: Répartition du temps d'exécution entre les boucles

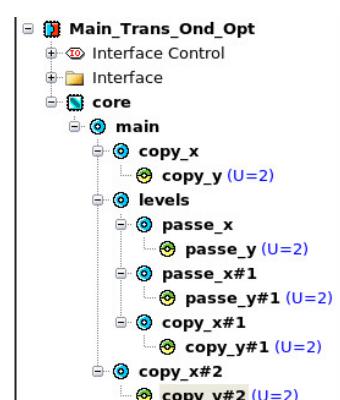
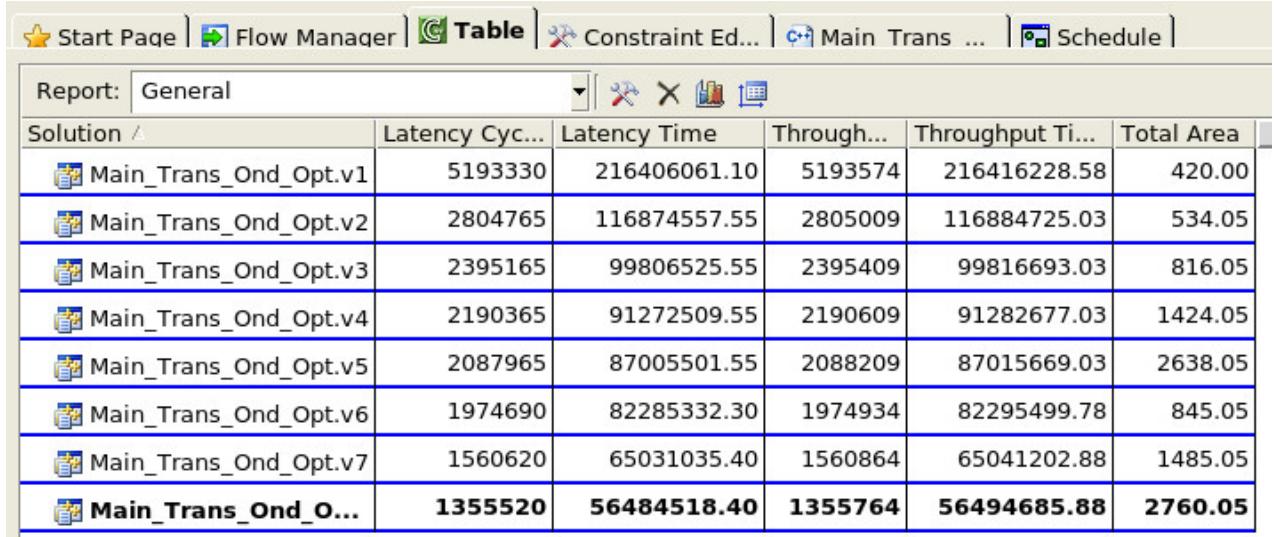


Figure 25: Boucles déroulées sur deux itérations

Les captures d'écran ci-dessous montrent les *latency_time* et *area_size* pour toutes les solutions testées :

- v1 : gestion naïve des deux mémoires (voir annexe 2)
- v2 : gestion optimisée des deux mémoires (voir annexe 3)
- v3, v4 et v5 : Unroll de la boucle *passe_y* sur des séquences de 2, 4 et 8 itérations.
- v6, v7 et v8 : Unroll des boucles sur des séquences de 2, 4 et 8 itérations.



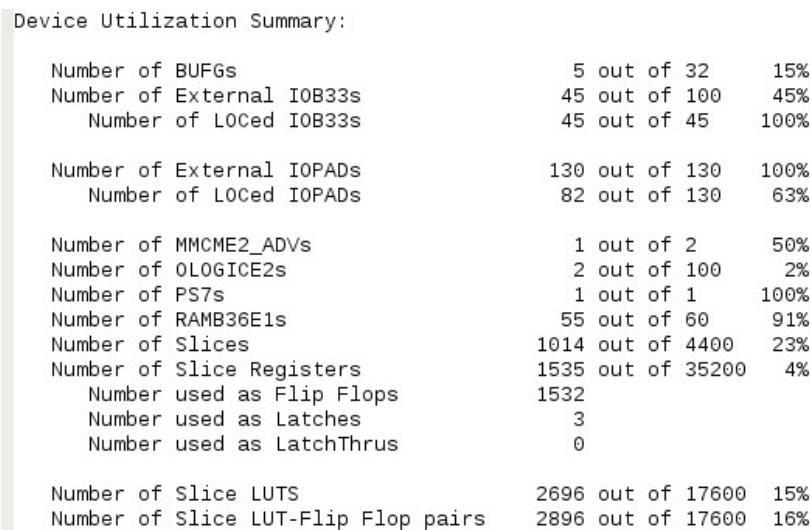
The screenshot shows the CatapultC software interface with the 'Table' tab selected. The main window displays a report table with the following columns: Solution /, Latency Cyc..., Latency Time, Through..., Throughput Ti..., and Total Area. The table lists eight solutions, each with its name, latency values, throughput values, and total area. The last row is highlighted in yellow, showing the best performance metrics.

Solution /	Latency Cyc...	Latency Time	Through...	Throughput Ti...	Total Area
Main_Trans_Ond_Opt.v1	5193330	216406061.10	5193574	216416228.58	420.00
Main_Trans_Ond_Opt.v2	2804765	116874557.55	2805009	116884725.03	534.05
Main_Trans_Ond_Opt.v3	2395165	99806525.55	2395409	99816693.03	816.05
Main_Trans_Ond_Opt.v4	2190365	91272509.55	2190609	91282677.03	1424.05
Main_Trans_Ond_Opt.v5	2087965	87005501.55	2088209	87015669.03	2638.05
Main_Trans_Ond_Opt.v6	1974690	82285332.30	1974934	82295499.78	845.05
Main_Trans_Ond_Opt.v7	1560620	65031035.40	1560864	65041202.88	1485.05
Main_Trans_Ond_O...	1355520	56484518.40	1355764	56494685.88	2760.05

Figure 26: Rapport de CatapultC

On constate qu'une fois que l'on a déroulé une boucle sur un certain nombre d'itérations, il ne coûte pas très cher en terme d'espace mémoire de dérouler les autres. En faisant cela, le gain en *latency_time* est néanmoins non-négligeable. En déroulant les boucles sur 8 itérations, on arrive à 56 ms, ce qui est proche du résultat que l'on souhaitait.

Le passage sous ISE n'ayant pas posé de problèmes d'usage excessif de ressources, on a conservé cette solution. Le résultat n'est toutefois pas très stable sur carte.



The screenshot shows the Xilinx Place and Route software interface with the 'Device Utilization Summary' section selected. The table provides a detailed breakdown of device resources used during the synthesis and implementation process.

Device Utilization Summary:		
Number of BUFGs	5 out of 32	15%
Number of External IOB33s	45 out of 100	45%
Number of LOCed IOB33s	45 out of 45	100%
Number of External IOPADs	130 out of 130	100%
Number of LOCed IOPADs	82 out of 130	63%
Number of MMCME2_ADVs	1 out of 2	50%
Number of OLOGICE2s	2 out of 100	2%
Number of PS7s	1 out of 1	100%
Number of RAMB36E1s	55 out of 60	91%
Number of Slices	1014 out of 4400	23%
Number of Slice Registers	1535 out of 35200	4%
Number used as Flip Flops	1532	
Number used as Latches	3	
Number used as LatchThrus	0	
Number of Slice LUTS	2696 out of 17600	15%
Number of Slice LUT-Flip Flop pairs	2896 out of 17600	16%

Figure 27: Extrait du rapport de Place and Route de ISE

4.3 Programme de démonstration

Pour pouvoir montrer le travail effectué, nous avons ajouté quelques fonctionnalités à la carte :

- Générateur de mire (préalable au projet) : SW1, BTN2, BTN1, BTN0
L'activation de SW1 montre l'image normalement traitée par la carte. Si SW1 est désactivé, on a un écran noir. En appuyant sur certains boutons, on affiche :
 - BTN0 : l'image normalement traitée par la carte.
 - BTN1 ou BTN 2 : des mires horizontales ou verticales générées par le vhdl.
- Contrôle du niveau de compression : BTN0
Pour montrer que notre programme pouvait réaliser différents niveaux de compression, nous avons ajouté un process *sync_nbLev* permettant de contrôler le niveau de compression souhaité via les boutons poussoirs de la zybo.
Un appui sur le bouton BTN0 augmente le niveau de compression d'une unité. Par défaut, le niveau est de 1, il va de 0 à 7. Le niveau courant de la carte est affiché en binaire par les leds 1, 2 et 3, la led 3 correspondant au bit de poids fort.
- Saturation : SW3
L'image transformée en ondelettes n'est pas toujours bien visible, car le gradient des bits de l'image d'origine n'est pas forcément très élevé. Pour faire apparaître cette transformation à l'écran malgré tout, nous avons ajouté la possibilité de saturer l'image en activant le switch SW3.
- Freeze : SW2
Certaines images sont plus intéressantes à analyser que d'autres. Pour pouvoir les observer tranquillement, nous avons ajouté la possibilité de "freezer" l'image en désactivant le switch SW2.

4.4 Travail restant

- Décompression
Il reste un souci dû au fait qu'avec une compression trop grande, la hauteur de l'image à traiter devient impaire, cela posant problème dans la reconstitution de l'image. Nous n'avons pas eu le temps de porter la décompression sur la carte pour parfaire notre programme de démonstration.
- Quantification
En fait, l'étape de quantification sert à transformer des coefficients d'ondelettes en entiers, mais nos coefficients sont déjà entiers. Donc on n'a pas fait cette partie.
- Passage en Bitstream
Nous n'avons malheureusement pas eu le temps de nous intéresser au passage en bitstream dans un format compatible JPEG2000. De plus cette étape n'est pas simple car la norme de syntaxe du .jp2 est propriétaire.

4.5 Problèmes rencontrés

Les problèmes qui ont ralenti notre travail sont, en plus de l'apprehension de la compression JPEG2000, majoritairement liés à la découverte de CatapultC. Il a en effet fallu du temps pour comprendre qu'il fallait (dés)activer certaines options pour rendre les fichiers créés par Précision exploitables dans le projet ISE. Les simulations dans CatapultC ont également nécessité certaines manipulations pour pouvoir être faites. L'ensemble de ces réglages/précautions sont détaillés dans le manuel utilisateur. Le problème qui nous a fait perdre le plus de temps a été de découvrir après des semaines de débug que les signaux *write_enable* générés par CatapultC sont en logique inversée.

Conclusion

En conclusion, bien que nous n'ayons pu aller au bout de nos objectifs par manque de temps, nous sommes relativement satisfaits du travail que nous avons réalisé.

Nous avons en effet pu porter la transformée en ondelettes jusqu'à la carte zybo, balayant ainsi tout le flot de conception. Ainsi, concernant la compression JPEG2000, nous avons pu réaliser une transformée en ondelettes par décomposition de Haar, et observer des premiers résultats de compression.

Nous avons également pu appréhender le logiciel HLS CatapultC (ses contraintes et quelques unes de ses capacités d'optimisation) et intégrer le module généré à un projet ISE existant et le porter avec succès sur la carte zybo.

Annexe 1 : Code effectuant un niveau de compression

```
void Trans_Ond(ac_int<8, false> Src[256*512], ac_int<8, false> Dst[256*512], int hi, int wi) {  
  
    ac_int<9,false> x;  
    ac_int<8,false> y;  
    ac_int<17,false> as1, as2, as3, as4;  
    ac_int<17,false> ad1, ad2, ad3, ad4;  
  
    hsplit_x : for (y = 0; y < (hi)/2; y++) {  
        hsplit_y : for (x = 0; x < (wi)/2; x++) {  
  
            // @ des pixels source à traiter pour cette itération  
            as1 = (512*(2*y)) + 2*x; // 2x , 2y  
            as2 = (512*(2*y)) + 2*x+1; // 2x+1, 2y  
            as3 = (512*(2*y+1)) + 2*x; // 2x , 2y+1  
            as4 = (512*(2*y+1)) + 2*x+1; // 2x+1, 2y+1  
  
            // @ pixels destination  
            ad1 = (512*y) + x; // LxLy  
            ad2 = (512*y) + x+wi/2; // LxHy  
            ad3 = (512*(y+hi/2)) + x; // HxLy  
            ad4 = (512*(y+hi/2)) + x+wi/2; // HxHy  
  
            // Calcul des pixels destination  
            Dst[ad1] = moy(moy(Src[as1], Src[as2]), moy(Src[as3], Src[as4]));  
            Dst[ad2] = sub(moy(Src[as1], Src[as2]), moy(Src[as3], Src[as4]));  
            Dst[ad3] = moy(sub(Src[as1], Src[as2]), sub(Src[as3], Src[as4]));  
            Dst[ad4] = sub(sub(Src[as1], Src[as2]), sub(Src[as3], Src[as4]));  
        }  
    }  
}  
  
ac_int<8,false> moy(ac_int<8,false> v1, ac_int<8,false> v2) {  
  
    return (v1+v2)/2;  
}  
  
ac_int<8,false> sub(ac_int<8,false> v1, ac_int<8,false> v2) {  
  
    return (v1-v2)/2;  
}
```

Annexe 2 : Utilisation naïve de ram1 et ram2

```
void Main_Trans_Ond_Opt(ac_int<8, false> Src[HEIGHT_IMAGE*WIDTH_IMAGE] ,
ac_int<8, false> Dst[HEIGHT_IMAGE*WIDTH_IMAGE] ,
ac_int<8, false> Vga[HEIGHT_IMAGE*WIDTH_IMAGE] ,
ac_int<3, false> nbLevels) {

int i, hi, wi;

hi = HEIGHT_IMAGE;
wi = WIDTH_IMAGE;

image_copy(Src, Dst, hi, wi); // Rules the level 0 of compression

levels : for(i=0; i<nbLevels; i++) {

    // We do one level treatment from Src to Dst
    Trans_Ond(Src, Dst, hi, wi);

    // Now we only take care of the top-left part of the picture
    hi = hi/2;
    wi = wi/2;

    // We write back the picture in Dst into Src for the next level treatment
    image_copy(Dst, Src, HEIGHT_IMAGE, WIDTH_IMAGE);
}

// The final picture is available in Dst, we copy it into Vga
image_copy(Dst, Vga, HEIGHT_IMAGE, WIDTH_IMAGE);
}
```

Annexe 3 : Utilisation optimisée de ram1 et ram2

```
void Main_Trans_Ond_Opt(ac_int<8, false> Src[HEIGHT_IMAGE*WIDTH_IMAGE] ,
ac_int<8, false> Dst[HEIGHT_IMAGE*WIDTH_IMAGE] ,
ac_int<8, false> Vga[HEIGHT_IMAGE*WIDTH_IMAGE] ,
ac_int<3, false> nbLevels) {

int i, hi, wi;

hi = HEIGHT_IMAGE;
wi = WIDTH_IMAGE;

image_copy(Src, Dst, hi, wi); // Rules the level 0 of compression

levels : for(i=0; i<nbLevels; i++) {

    // We do one level treatment from Src to Dst
    Trans_Ond(Src, Dst, hi, wi);

    // Now we only take care of the top-left part of the picture
    hi = hi/2;
    wi = wi/2;

    i++;
    if (i != nbLevels) {

        Trans_Ond(Dst, Src, hi, wi);

        // If we have reached the end of the "levels" loop, we copy back the Src to Dst
        if (i == nbLevels - 1) {
            image_copy(Src, Dst, hi, wi);
        }

        hi = hi/2;
        wi = wi/2;
    }
}

// The final picture is available in Dst, we copy it into Vga
image_copy(Dst, Vga, HEIGHT_IMAGE, WIDTH_IMAGE);
}
```

Annexe 4 : Manuel Utilisateur

Software

Description du dossier

Ce dossier contient cinq sous-dossiers.

Le dossier include contient les librairies utilisées : ac_int et mc_scverify.

Le dossier bin contient le fichier exécutable.

Le dossier obj contient les fichiers objets créés à la compilation.

Le dossier src contient les sources que nous avons écrites.

Le dossier image contient 2 images test en format pgm. (Le choix de cette image se fait dans src/globals.h)

les scripts à disposition

Le script run.sh donne un exemple de transformée en ondelettes et affiche la source, l'image compressée et l'image décompressée.

Le script fullDecomp.sh créé des images totalement décompressées pour différents niveaux de compression.

Le script partialDecomp créé des images comprimées au niveau 5 mais décompressées à des niveaux différents.

la commande de compilation

make

la commande de nettoyage

make clean nettoie les images générées et les fichiers objets générés

make cleanimage nettoie uniquement les images .pgm contenues dans le dossier principal

la commande d'exécution

./bin/exec

'niveau de compression'

'niveau de décompression' (0 donne une décompression complète)

'nom de l'image décompressée' (à nommer avec l'extension .pgm)

'nom de l'image compressée' (à nommer avec l'extension .pgm)

L'image générée se situe dans le dossier principal.

Implantation sur carte

CatapultC

These steps will show you how to generate the Main_Trans_Ond.Opt.edf EDIF file that is necessary to run the ISE synthesis.

1. Open CatapultC

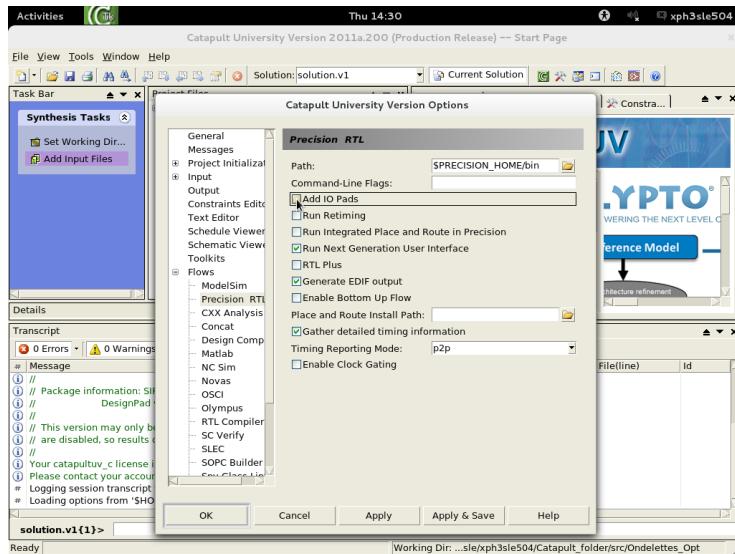
Type "catapult &" in your terminal

2. Set Precision Options

Go to Tools/Set Options.../Flows/Precision RTL.

Uncheck the "Add IO Pads" box.

!\\Must be done each time CatapultC is opened, the 'save' function doesn't seem to work.

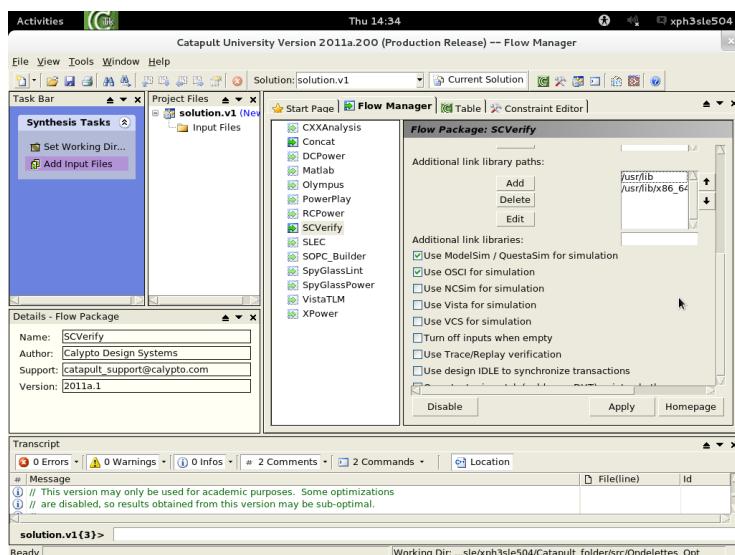


3. Enable Vsim simulation

Go to the Flow Manager tab in the top-right window.

Enable SCVerify.

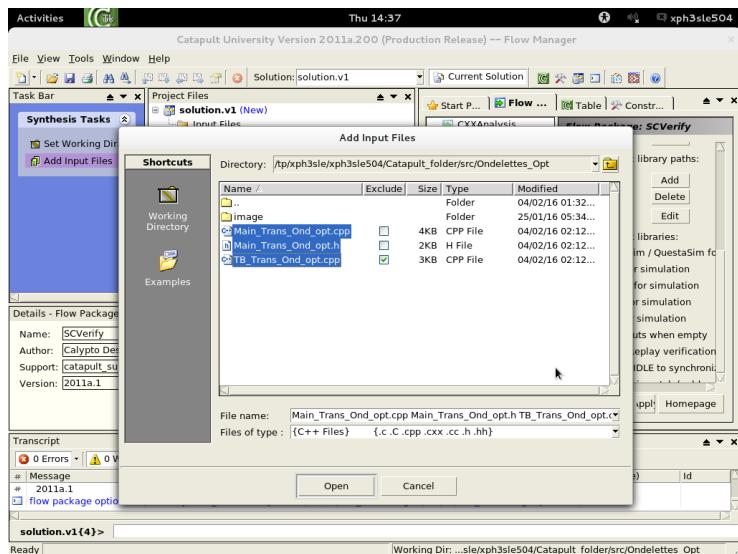
Some link library paths might need to be added : /usr/lib and /usr/lib/x86_64-linux-gnu.



4. Add input files

Via the task bar (top left), add input the input files.

Don't forget to check the "exclude" box for the testBench file.

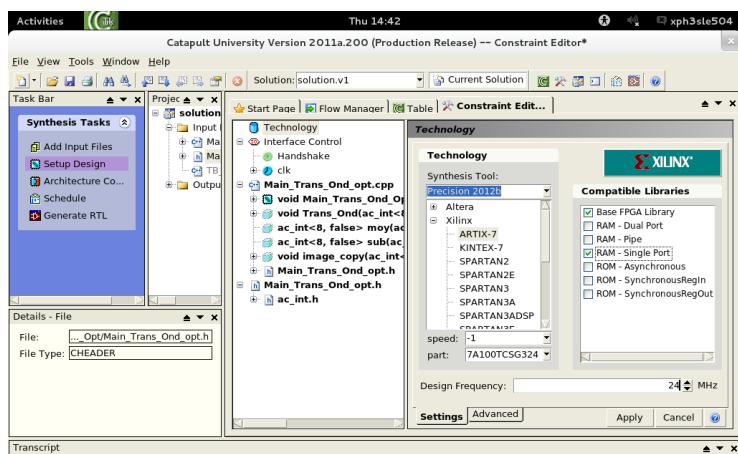


5. Setup Design

Click the Setup Design button in the Task bar window

In the Constraint Editor tab :

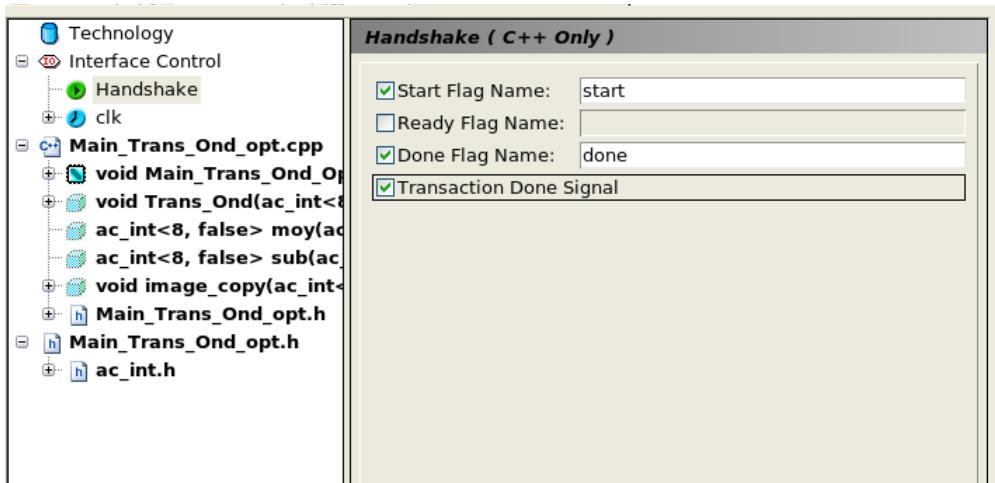
- Technology
 - Synthesis Tool : Precision 2012b/Xilinx/ARTIX-7
 - Compatible libraries : Check the RAM-SinglePort box (the Base FPGA library should be already checked)
 - Design Frequency : 24MHz
 - "Advanced tools" tab : raise the array size, up to 1flowteurs*
- Respecte leur son bordel !00024 for instance
- Apply



- Handshake

check the signals that follow then apply

- Start Flag Name : start
- Done Flag Name : done
- Transaction Done Signal



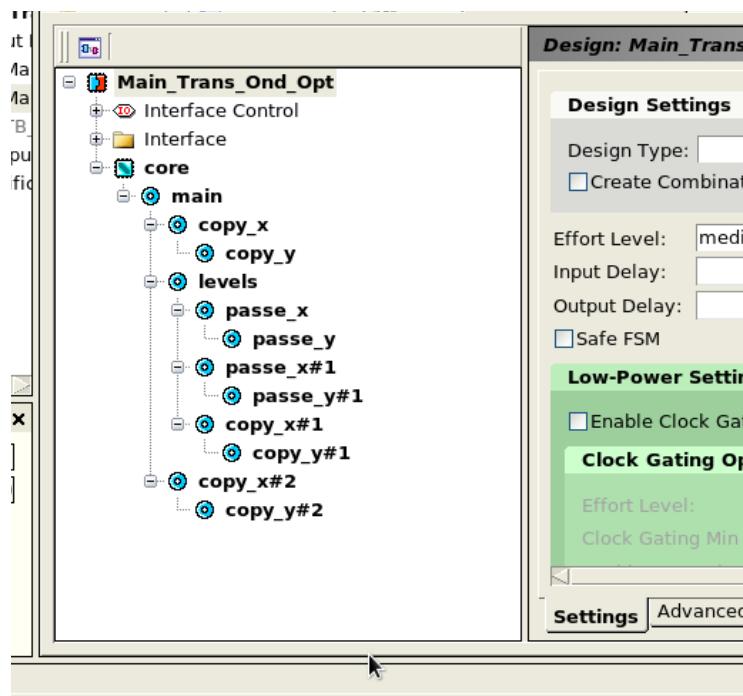
- Click on 'void Main_Trans_Ond_Opt(...)' then check "top design" and Apply

6. Architecture Constraint

Click the Architecture Constraint button in the Task bar window

!\\ Check that you have no question marks for any of your loop

If you click on a loop, you can unroll and/or pipeline your loops at this stage



7. Schedule

Click the Schedule button in the Task bar window

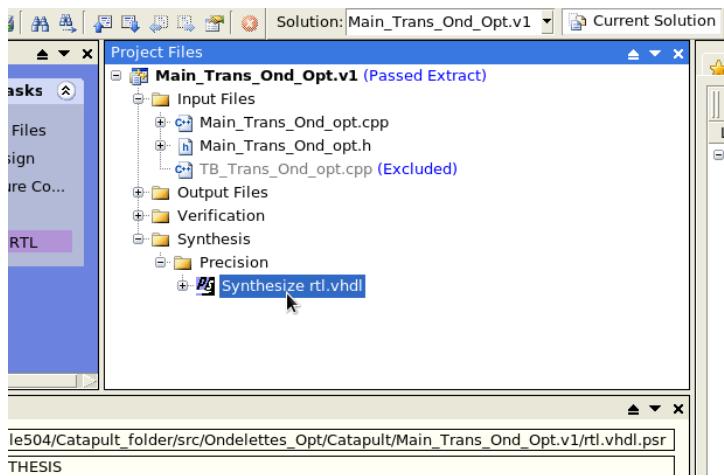
You can check the timing of your solution in the Table and Schedule tabs.

8. Generate the output files

Click the Generate RTL button in the Task bar window

In the Project files window, right click on 'Synthesis/Precision/Synthesize rtl.vhdl' then 'Launch Precision batch'

This will generate the .edf and the report files in the Catapult/Main_Trans_Ond_Opt.v1/psr.vhdl_impl folder.



9. Check your solution with SCVerify

First, in your terminal, go to the Catapult/Main_Trans_Ond_Opt_v1/folder and type : "ln -s /usr/lib/x86_64-liux-gnu/crt* ."

In the Project files window, double click on the simulation you want in 'Verification/ModelSim'.

ISE

Go to zybo_cam/zybo_cam.ise/zybo_cam/ (in our session, the zybo_cam folder is located in /TP/) Launch "ise zybo_cam.xise &"

Add the .edf generated file into the project sources and launch synthesis.

Generate binary file

Go to zybo_cam/wk_cam/cam_conf/bootimage/

Type "bootgen -image cam_config.bif -o boot.bin -w on"

Attention : ISE must be installed for the bootgen to work. Just copy boot.bin into the SD Card and insert it into the zybo.

Utilisation du programme de démonstration

- SW0 up :

- BTN0 pressed : As if SW0 was down.
- BTN1 or BTN2 pressed : Print software generated pictures on the screen.

- SW0 down :

- SW3 up : Saturate the final picture
- SW2 up : Freeze the picture
- SW1 down : Force the compression level to "1".
- BTN0 : Each pressure raises the level of compression. That level raises from 0 (no treatment) to 7 then comes back to 0. The current level is shown in binary by the leds 1, 2 and 3 (led 3 is the most significant bit).