Rapport d'avancement : projet Encodage Vidéo Ensta-Bretagne Expérimental (EVEEX)

Membres du groupe : Guillaume Leinen, Jean-Noël Clink, Hussein Saad, Alexandre Froehlich, Hugo Questroy

Encadrants : Pascal Cotret, Jean-Christophe Le Lann, Joël Champeau

Abstract - résumé

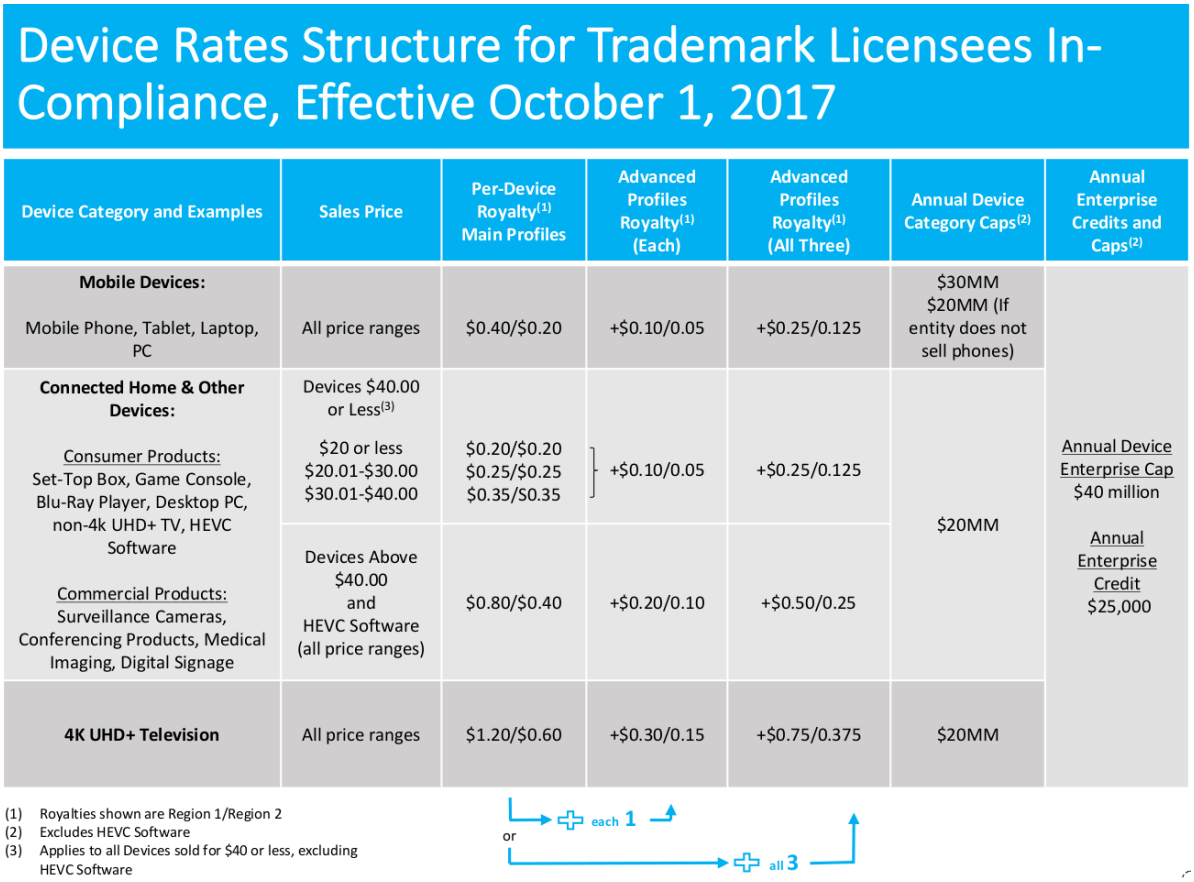
Sommaire

Remerciements

- Nos encadrants Pascal Cotret, Jean-Christophe Le Lann et Joël Champeau, qui nous aident à définir les objectifs à atteindre, ainsi qu'a résoudre des problèmes théoriques.
- Enjoy Digital, société créé par un Alumni Ensta-Bretagne, et son produit Litex qui nous sera très utile sur l'implémentation hardware.
- Le site FPGA4students pour ses tutoriels VHDL/Verilog.
- Jean-Christophe Leinen pour ces conseils sur les méthodes Agiles.

Introduction

Aujourd'hui, le contenu vidéo, en particulier le *streaming* , représente 61% de la bande passante mondiale [1]. Cependant si les derniers algorithmes présentes des performances très convaincantes, force est de constater que les codecs anciens comme le MPEG-4 (mp4) sont encore très largement utilisés. Il n'existe pas aujourd'hui de codec *open-source* doté de performances en accord avec les besoins d'aujourd'hui que ce soit au niveau de la qualité d'image ou de la bande-passante nécessaire. Car en effet, et cela est peu connu, les codecs largement utilisés comme le H.265 ou le MP4 sont en source fermée (closed-source), et demande des royalties pour une utilisation commerciale par une entreprise.



Device Rates Structure for Trademark Licensees In-Compliance, Effective October 1, 2017

Device Category and Examples	Sales Price	Per-Device Royalty[1] Main Profiles	Advanced Profiles Royalty[1] (Each)	Advanced Profiles Royalty[1] (All Three)	Annual Device Category Caps[2]	Annual Enterprise Credits and Caps[2]
Mobile Devices: Mobile Phone, Tablet, Laptop, PC	All price ranges	$0.40/$0.20	+$0.10/0.05	+$0.25/0.125	$30MM $20MM (If entity does not sell phones)	Annual Device Enterprise Cap $40 million Annual Enterprise Credit $25,000
Connected Home & Other Devices: Consumer Products: Set-Top Box, Game Console, Blu-Ray Player, Desktop PC, non-4k UHD+ TV, HEVC Software Commercial Products: Surveillance Cameras, Conferencing Products, Medical Imaging, Digital Signage	Devices $40.00 or Less[3] $20 or less $20.01-$30.00 $30.01-$40.00	$0.20/$0.20 $0.25/$0.25 $0.35/$0.35	+$0.10/0.05	+$0.25/0.125	$20MM	
	Devices Above $40.00 and HEVC Software (all price ranges)	$0.80/$0.40	+$0.20/0.10	+$0.50/0.25		
4K UHD+ Television	All price ranges	$1.20/$0.60	+$0.30/0.15	+$0.75/0.375	$20MM	

(1) Royalties shown are Region 1/Region 2
(2) Excludes HEVC Software
(3) Applies to all Devices sold for $40 or less, excluding HEVC Software

Figure 1: Carte des tarifs pour utiliser HVEC (h.265). Cela peut représenter un coût conséquent pour les entreprises

Parallèlement à ça, il existe depuis plusieurs d'années une technologie de circuits imprimés appelé Field programmable Gate Array (ou FPGA). Une puce FPGA est un circuit imprimé **reconfigurable** fonctionnant à base de portes logiques (par opposition au processeur qui ne peut être reprogrammé). Cette technologie prend son envol à partir des années 90, mais c'est aujourd'hui que les entreprises, et plus particulièrement les géants du silicium (intel, AMD) s'intéresse de près à cette technologie. En effet, elle dispose de plusieurs avantages qui en font une technologie de rupture dans certains domaines d'applications:

- Le caractère reconfigurable permet un prototypage de circuit intégré final, tout en ne passant pas par une couche d'émulation logicielle coûteuse et peu performante.
- Son architecture en portes logiques permet un grand parallélisme dans les calculs, ce qui permet d'accélérer considérablement certaines taches parallélisables (compression/décompression de fichier, calculs 3D, deep-learning, réseau de neurones, etc...).
- Son implémentation "hardware" d'un algorithme (à contrario de l'utilisation d'un jeu d'instruction pour les processeurs) permet une optimisation poussé du produit, que ce soit en terme de performance, ou de consommation électrique, le dernier représentant un enjeu important de l'électronique moderne.

En examinant ces avantages, on comprend vite l'intérêt d'une tel technologie dans la compression et le traitement vidéo.

Afin de répondre à cette problématique nouvelle, L'ENSTA Bretagne voudrait développer un algorithme de compression vidéo, qui soit open-source et doté de performances convaincantes (définies par la suite), et par la suite d'implémenter cette algorithme sur une carte FPGA. Ce projet d'algorithme constitue notre travail et se nomme EVEEX (projet Encodage Vidéo Ensta-Bretagne Expérimental).

Dans ce rapport, nous allons définir les exigences de fonctionnement d'un tel système, puis présenter notre travail actuel (et futur), que ce soit au niveau de l'algorithme, ou de son implémentation sur une carte FPGA.

Définition des exigences

Afin de définir clairement nos objectifs pour ce projet, il est primordial de définir les exigences, qu'elles soient fonctionnelles ou physique (programmation).

Nous l'avons aborder dans l'introduction, la problématique des codecs vidéos est primordiale dans la gestion de la bande passante globale ainsi que de l'impact énergétique d'Internet. Pour permettre une amélioration collaborative et un accès universelle, il est donc primordial que le projet soit open-source. Cet algorithme doit permettre l'ingestion d'un flux vidéo, provenant par exemple d'une camera, la compression de celui-ci, le formatage des données compressés, l'envoi de ces données à travers le réseau, le décodage des données reçu via le réseau, la décompression des données compressés ainsi que l'affichage de celles-ci.

Enfin, l'algorithme doit être développer dans un langage permettant une implémentation sur fpga. Nous verrons plus tard que le choix du language est un point crucial dans la réalisation du projet.

Nous avons donc défini un certain nombre d'exigences avec les performances attendus lorsqu'elle sont pertinentes, ainsi que notre certitude quand à la réalisation de ces exigences.

Les points de vocabulaires au niveau de exigences seront définis par la suite dans le rapport et dans un glossaire.

Numéro identifiant l'exigence	exigence	performances attendus	certitude quand à la réalisation de cette exigence (en %)
1	Le projet doit-être intégralement open-source et accessible gratuitement. (exigence non fonctionnelle)	None	100% => après analyse, il est probable que l'intégralité des solutions à incorporer soit open-source. En revanche, la chaîne de compilation sur FPGA dépend du fabricant de la carte choisi par un utilisateur. Si ces logiciels (Intel Quartus, Xilinx Vivado) sont gratuits, il ne sont en revanche pas open-source. Malgré tout il existe des solutions FPGA (carte + IDE) open-sources comme Lattice.
2.1	L'algorithme doit pouvoir recevoir un flux photos et vidéo "brut" et le convertir en format exploitable	Conversion d'un flux RGB en flux YUV/YCbCr	100%, la conversion étant un simple calcul matriciel, il est facile à implémenter dans un language haut-niveau (python), et possiblement implémentable de façon direct en HDL sur la carte FPGA.
2.2.1	l'algorithme doit compresser les données brut	Dans un premier temps, performances analogues au MPEG-1 => 20:1 pour une photo, 100:1 pour vidéo	50%, si le taux de compression image semble atteignable (à l'heure actuelle environ 10:1 en python), le taux vidéo va demander pas mal d'optimisation du code en langage c.
2.2.2	l'algorithme doit décompresser des données compressés	Identiques ou supérieurs à l'encodage	100%, le décodage d'un arbre du Huffmann est plus rapide que l'encodage.
2.2.3	l'algorithme doit compresser les données d'une manière originale (pas une copie de MPEG)	None	80%, Nous pensons incorporer une taille variable de macrobloc de traitement, ce qui nous différencie fondamentalement du MPEG

Numéro identifiant l'exigence	exigence	performances attendus	certitude quand à la réalisation de cette exigence (en %)
2.3.1	l'algorithme doit pouvoir formater les données compresser afin qu'elle puissent être envoyé en réseau.	None	100%, nous avons une bonne maîtrise du bitstream en python et l'implémentation c est en cours.
2.3.2	l'algorithme doit pouvoir recevoir les données par le réseau et les comprendre	None	100% pour les mêmes raisons
2.4.1	l'algorithme doit permettre un affichage d'une image décodé	affichage VGA sur la carte fpga	100% si on utilise Litex, et déjà réalisé en vhdl natif
3.1	l'algorithme doit pouvoir s'exécuter sur une carte FPGA	identiques ou supérieurs à la version pc de l'algorithme	70%, avec l'utilisation d'un SOC riscV l'implémentation C devrait être jouable. Le principal point bloquant est la gestion de la mémoire sur la carte FPGA.
4.1	l'algorithme implémenté sur fpga doit induire une faible consommation électrique	Inférieure à la consommation d'un PC exécutant l'algorithme. (<30W)	100%, les fpga consomment nativement peu d'énergie (notre modèle s'alimente par port micro-usb)

On rajoute à ces exigences les fonctions définissant les relations entre l'algorithme et les acteurs externes. Pour cela la méthode du diagramme en Pieuvre est utilisé. Elle permet d'illustrer clairement les fonctions accomplies par le système (algorithme EVEEX).

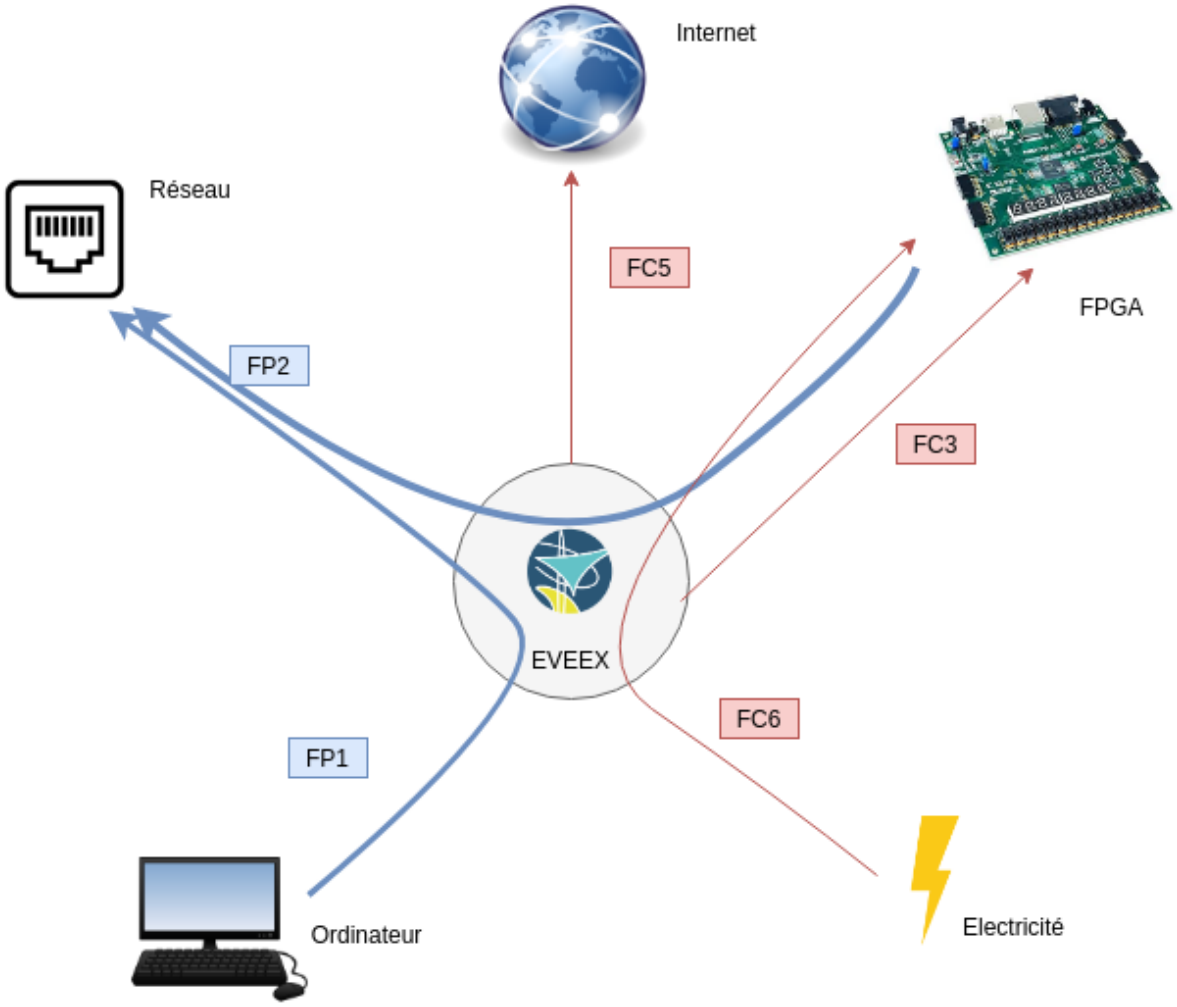


Figure 2: diagramme "Pieuvre" du projet EVEEX

- FP1: transmission d'images / vidéo entre 2 pc par réseau
- FP2: transmission d'images / vidéo entre 2 fpga par réseau
- FC1: les données doivent être compressés (*Pour être transmises par le réseau*)
- FC2: les 2 machines doivent être reliés en réseau
- FC3: Un FPGA doit pouvoir comprendre l'algorithme
- FC4: l'algorithme doit être différent des algorithmes existants (copyright)
- FC5: l'algorithme doit être open-source
- FC6: l'algorithme doit permettre une réduction de la consommation électrique (sur FPGA)

L'architecture physique étant plus spécialisé et technique, nous l'aborderons au sein de la prochaine partie, consacré au fonctionnement interne de l'algorithme et de son architecture de code.

Réalisation de l'algorithme

La réalisation de l'algorithme constitue une part très importante du projet, puisque les performances d'EVEEX découlent directement des choix en termes de traitement.

Pour cette algorithme de traitement des données, nous nous sommes basé sur le fonction du MJPEG, car il est relativement simple à appréhender.

Le fonctionnement de l'algorithme est détaillé dans le diagramme en bloc si dessous. Il est composé de plusieurs phases, dont certaines necessitant quelques concepts mathématiques.

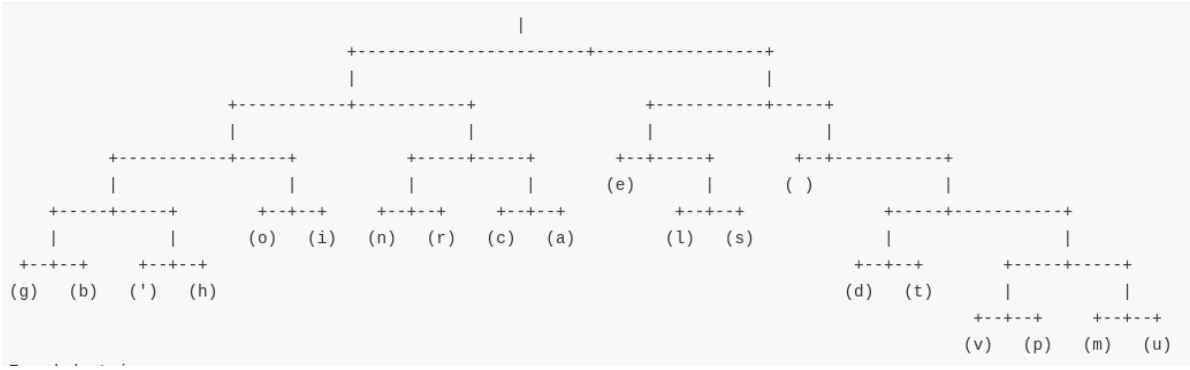
L'image, au format RGB (que sort nativement la plupart des cameras), est tout d'abord converti au format Chrominance/luminance (YUV). Cela permet notamment de séparer la matrice originelle en 2 matrices indépendantes, et cela est souhaitable après DCT par la suite.

Ensuite l'image est découpé en **macroblocs** de 16px x 16 px pour l'instant. Cela permet de faciliter le traitement et de paralléliser les taches. C'est que nous allons nous différencier des autres algorithmes, en rendant cette taille de macroblocs **variable** en fonction du contenu du macrobloc. Si un macrobloc présente un taux de contraste élevé, on réduit sa taille, alors que si c'est un aplat de couleur on l'augmente. Cela permettra à priori d'augmenter le taux de compression.

Après cet étape, on effectue divers transformations des matrices macroblocs afin de les compresser:

- Une Transformation en Cosinus Direct ou DCT

La partie suivante concerne la mise en format des données. On utilise pour cela un arbre binaire de Huffman qui permet à la fois de compresser et de formater les données selon une trame précise. On appellera la trame à transmettre un **Bitstream**.




```
Encoded string :
10101001100110000110011011011011100100110101000010100010010111110101111100000101011
1001110101110000000100100110101111111101011101010011111001100010111110111101010110
0010111011001100100100110111100001111100100001 0
String decoded back : le chic de l'ensta bretagne sur la compression vide
```

L'arbre se base sur la récurrence des caractères afin de les ordonner et d'adresser à chaque caractère un mot binaire. Il suffit de remplacer "caractère" par "valeur" au sein d'une matrice pour ordonner la donnée.

Implémentation de l'algorithme sur FPGA

Au niveau du FPGA, les choses ont évolués récemment.

Nous étions partis pour développer entièrement en langage HDL l'algorithme, en passant pour les fonctions compliqués par un outil de *High Level Synthesis* ou HLS. Cependant nous nous sommes rendu compte qu'il serait compliqué de continuer ainsi et cela pour de multiples raisons:

- La HLS permet en théorie de transformer un code écrit en c en un code HDL comme le VHDL ou le Verilog. Le problème de la HLS est qu'elle induit de l'opacité dans le code machine finale (on ne contrôle pas la génération de code). Aussi la syntaxe c nécessaire à son interprétation par le logiciel de HLS est extrêmement contraignante (impossible de faire des mallocs, typage limité aux types primaires,etc...[4]).
- les langages HDl comme le VHDL sont extrêmement durs à appréhender, la courbe d'apprentissage est très abrupte (la syntaxe est très différente, c'est très bas niveau, les

types limités à des mots binaires, etc...). il est assez déraisonnable d'apprendre la programmation machine en HDL en seulement quelques mois
- l'interface de développement Xilinx Vivado, si elle permet quelques fonctionnalités intéressantes notamment pour l'analyse de code et l'optimisation, est très lourde à installer (80 go sur le disque dur) et très difficile à prendre en main (environ 2 semaines pour comprendre le fonctionnement du framework et les principales fonctionnalités). On voudrait privilégier un IDE plus simple.
- enfin, la gestion des I/O ainsi que de la RAM demande dans la grande majorité des cas de faire appel à des bibliothèques de xilinx, qui sont gratuites pour certaines (et très chères pour d'autres []), et qui sont surtout *closed-sourced*.

Sur conseil de nos encadrants, nous nous sommes donc intéresser à une solution alternative proposé par Florent Kermarrec, ENSTA promotion 2008 et sa société "Enjoy Digital" spécialisé dans la fabrication de solutions FPGA sur-mesures. Cette personne à développer un outil appeler **"LITEX"**.

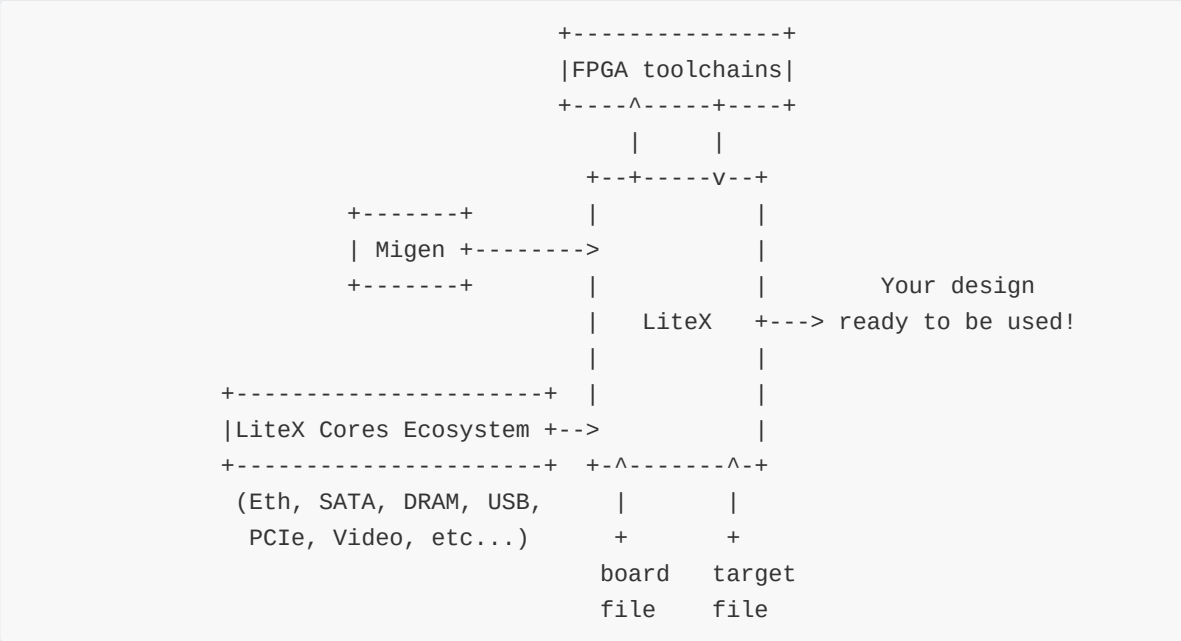


Figure 4: Design flow de LiteX présent sur le github de Enjoy Digital

LiteX est un framework de développement fpga basé sur le langage **Migen**, lui-même reprenant la syntaxe et le fonctionnement de python. LiteX permet de développer des Firmwares FPGA aussi bien en bas-niveau avec un syntaxe python, qu'en haut niveau via la création d'un SOC d'architecture RiscV, capable notamment d'accueillir un kernel linux et d'exécuter du code c de manière normale (sans les inconvénients de la HLS). Il permet aussi de gérer nativement les principales entrées/sorties (ethernet, vga, usb) ainsi que le RAM, et ce par un code complètement open-source.

Toutes ces caractéristiques le rendent idéal pour notre algorithme. L'intérêt d'un tel système est aussi la possibilité, afin d'améliorer les performances de l'algorithme, de migrer certaines parties du code en VHDL afin de l'exécuter de la manière la plus optimisé possible.

L'objectif pour nous est ainsi de compiler notre code c pour l'architecture RiscV (qui est open-source), de créer une architecture processeur "sur-mesure" grâce à LiteX, et d'optimiser les phases critiques de l'algorithme afin d'exploiter le parallélisme du FPGA.

Concernant l'état du projet en lui-même. nous développons sur une carte Digilent Nexys4 DDR, donc nous utilisons la toolchain vivado. Pour l'acquisition vidéo nous avons opté pour des caméras OV7670. Elles filment en 480p et on l'avantage d'être très peu chère (1.5 € l'unité), ce qui est pratique quand on débute dans le domaine (nous avons déja brûlé une camera).

Nous avons adapté un code existant en vhdl afin d'afficher sur un écran en VGA le retour de la caméra 7670. La qualité d'image n'est pas la meilleure mais elle devrait être suffisante pour exploiter l'algorithme

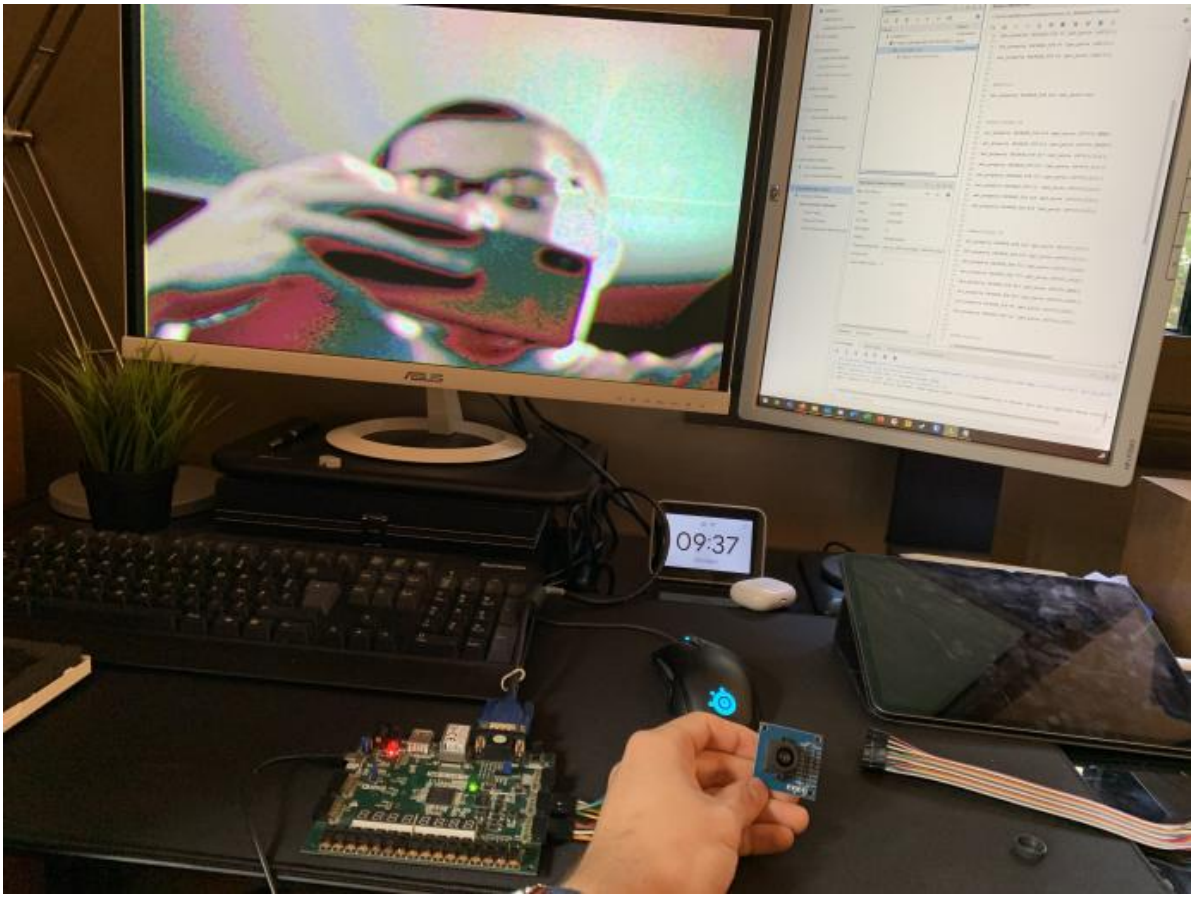


Figure 5: rendu VGA de la camera OV7670 relié à un FPGA Nexys4 (il n'y a pas de compression l'image est affiché de manière directe). La colorimétrie est lié à l'absence de blindage des câbles reliant la camera à la carte.

Pour pallier le problème du branchement de la camera (connecteur Pmod), notre encadrant Pascal Cotret nous a conçu un adaptateur maison.

Concernant le code en lui-même, nous commençons à comprendre la démarche de création d'un SOC, par notamment la réalisation des tutoriels LiteX proposé par Enjoy Digital.

Déroulement Agile du projet

Nous avons bien entendu respecter les principes de la méthode Agile pour

Annexes

Bibliographie

[1] https://fr.statista.com/infographie/21207/repartition-du-trafic-internet-mondial-par-usage/

[2] https://www.tomshardware.fr/amd-envisage-dacquerir-xilinx-inventeur-du-fpga-pour-30-milliards-de-dollars/

[4] pdf xilinx sur la HLS